

Schriften des Universitätsverlags 1

# Exploring linguistic corpora with MATLAB<sup>®</sup>: A guide to data analysis and visualisation

Katrin Menzel



*universaar*

Universitätsverlag des Saarlandes  
Saarland University Press  
Presses Universitaires de la Sarre



Schriften des Universitätsverlags  
Band 1

Katrin Menzel

# Exploring linguistic corpora with MATLAB<sup>®</sup>: A guide to data analysis and visualisation



*universaar*

Universitätsverlag des Saarlandes  
Saarland University Press  
Presses Universitaires de la Sarre

**Bibliografische Information der Deutschen Nationalbibliothek**

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <https://dnb.dnb.de> abrufbar.

2026 *universaar*

Universitätsverlag des Saarlandes  
Saarland University Press  
Presses Universitaires de la Sarre  
Campus B1 1  
66123 Saarbrücken  
E-Mail: [universaar@sulb.uni-saarland.de](mailto:universaar@sulb.uni-saarland.de)

ISBN 978-3-86223-368-7 (print)  
ISBN 978-3-86223-369-4 (online)  
DOI 10.22028/D291-47188

Satz: Katrin Menzel  
Umschlaggestaltung: Julian Wichert

Gedruckt auf FSC-zertifiziertem Papier.  
Druck: CPI Druckdienstleistungen GmbH, Ferdinand-Jühlke-Straße 7, 99095 Erfurt

# Table of contents

List of figures .....	IX
List of tables.....	XV
Acknowledgements .....	XVI
About the author .....	XVI
1 Overview of MATLAB® and possible applications in corpus-linguistic research.....	1
1.1 What is MATLAB®? .....	1
1.2 How this book can enrich your corpus-linguistic studies .....	2
2 Getting started with MATLAB® .....	5
2.1 Installing MATLAB® .....	5
2.1.1 Overview of installation options and key steps.....	5
2.1.2 Step-by-step installation process .....	7
2.1.3 Troubleshooting common installation issues.....	9
2.1.4 Getting updates of installed MATLAB® release.....	9
2.1.5 Installing Statistics and Machine Learning Toolbox™ and Text Analytics Toolbox™ .....	10
2.1.6 Using MATLAB Online™ .....	11
2.2 Starting and closing MATLAB® .....	11
2.3 Starting MATLAB® from Python® and using other external language interfaces .....	12
2.4 Overview of MATLAB® documentation and support resources.....	13
2.5 Assistance through the MATLAB® AI Chat Playground and MATLAB GPT .....	15
2.6 Assistance through MATLAB® Copilot .....	17
3 The MATLAB® environment .....	19
3.1 Overview and MATLAB desktop® .....	19
3.2 The MATLAB® toolbar.....	20
3.3 The Files panel .....	20
3.4 The Command Window .....	22
3.5 The Command History .....	26
3.6 The Workspace panel.....	28
3.7 The Variables editor.....	30
3.8 MATLAB® Apps .....	32
3.9 MATLAB® Copilot.....	33
3.10 MATLAB Drive and MATLAB Online™ Desktop .....	35
4 Examples of retrieving corpus data and other textual resources .....	37
4.1 Extracting linguistic data from publicly available corpora and other textual sources.....	37
4.2 Retrieving data from corpora at English-Corpora.org .....	39

4.3	Retrieving data from corpora via the Sketch Engine Interface.....	43
4.4	Retrieving data from corpora through CQPweb interfaces.....	47
5	Creating and editing variables in MATLAB®.....	53
5.1	Introduction to MATLAB® data types.....	53
5.2	Defining variables of different data types via the Command Window.....	54
5.2.1	Numeric variables.....	54
5.2.2	Character arrays.....	58
5.2.3	String arrays.....	60
5.2.4	Cell arrays.....	66
5.2.5	Categorical arrays.....	69
5.2.6	Tables.....	73
5.2.7	Structures.....	75
5.2.8	Dictionaries.....	77
5.3	Creating variables via the 'New Variable' option.....	78
5.4	Viewing and modifying variables by using the Variables editor.....	80
5.5	Importing variables from data files.....	81
5.6	Saving and loading workspace variables.....	86
6	Basic concepts of MATLAB® scripts.....	89
6.1	Overview.....	89
6.2	Writing and running MATLAB® scripts.....	91
6.3	MATLAB Live Editor®.....	97
6.4	Debugging and error handling.....	99
7	Visualising linguistic data with MATLAB®.....	103
7.1	Overview.....	103
7.2	MATLAB® plotting basics.....	104
7.3	Vertical bar charts.....	111
7.4	Horizontal bar charts.....	121
7.5	Stacked bar charts.....	122
7.6	Pie charts.....	126
7.7	Line plots.....	135
7.8	Stem plots.....	140
7.9	Scatter plots.....	143
7.10	Box plots.....	152
7.11	Histograms.....	161
7.12	Heat maps.....	164
7.13	Word clouds.....	168

7.14	Arranging multiple plots in a single figure .....	175
7.14.1	Creating a grid of subplots .....	175
7.14.2	Arranging multiple plots with tiled chart layouts.....	178
7.14.3	Overlaying multiple plots .....	179
7.15	Summary of key MATLAB® figure customisation options and plotting commands .....	184
7.16	Best practices for effective data visualisation in MATLAB® .....	189
8	Basic arithmetic operations and descriptive statistics in MATLAB® .....	191
8.1	Basic arithmetic operations and number processing .....	191
8.1.1	Overview .....	191
8.1.2	Addition .....	191
8.1.3	Subtraction .....	195
8.1.4	Multiplication and division .....	196
8.1.5	Logarithmic functions, square root and exponentiation.....	199
8.1.6	Rounding functions.....	203
8.1.7	Sorting data .....	204
8.2	Descriptive statistics – core statistical measures.....	206
8.2.1	Overview .....	206
8.2.2	Minimum and maximum .....	208
8.2.3	Measures of central tendency.....	210
8.2.4	Measures of dispersion .....	213
8.2.5	Data summary .....	214
9	Exploring inferential statistics, statistical tests and relationships between variables with MATLAB® .....	219
9.1	Overview .....	219
9.2	Working with probability distributions.....	220
9.3	Hypothesis tests.....	223
9.4	Analysis of variance.....	232
9.5	Regression models .....	235
9.6	Correlation coefficients.....	237
10	Text preprocessing and analysis with MATLAB's® Text AnalyticsToolbox™ .....	245
10.1	Overview of the Text Analytics Toolbox™.....	245
10.2	Extracting text data from files or from the web .....	246
10.3	Text preprocessing .....	248
10.4	Examples of text analysis procedures using the Text Analytics Toolbox™ .....	265
10.4.1	Creating a bag-of-words model.....	265
10.4.2	Creating a bag-of-n-grams model .....	272

10.4.3	Working with pretrained word embeddings.....	276
10.4.4	Training your own word embeddings .....	283
10.4.5	Generating summaries from text documents.....	289
10.5	Further resources on Text Analytics Toolbox™ techniques.....	293
11	Summary .....	297
	Bibliography.....	299

## List of figures

Figure 1: Selecting toolboxes during the installation process.....	7
Figure 2: Example of an AI Chat Playground interaction.....	15
Figure 3: Test of the suggested code in the execution field of the AI Chat Playground .....	16
Figure 4: MATLAB GPT.....	16
Figure 5: MATLAB Copilot (on the right of MATLAB desktop).....	17
Figure 6: The MATLAB desktop (R2026a).....	19
Figure 7: The MATLAB toolstrip.....	20
Figure 8: Selecting your current working directory.....	20
Figure 9: Accessing the Files panel.....	21
Figure 10: Renaming a folder in the Files panel.....	21
Figure 11: Creating a new folder in the Files panel.....	22
Figure 12: Creating a new variable <code>x</code> in the MATLAB Command Window .....	22
Figure 13: Creating a variable <code>a</code> in the MATLAB Command Window .....	23
Figure 14: Result of a simple calculation stored in the <code>ans</code> variable .....	23
Figure 15: Creating a string variable in the Command Window .....	24
Figure 16: MATLAB Help Browser, accessed using the <code>doc</code> command.....	25
Figure 17: Suggested variable names from workspace to complete new command .....	25
Figure 18: Suggested modifications for a box plot during command typing .....	26
Figure 19: Using the search functionality in the Command Window .....	26
Figure 20: Selecting a previous command from the Command History, opened in a pop-up window.....	27
Figure 21: Opening the Command History panel via a command .....	27
Figure 22: The Workspace panel.....	28
Figure 23: Duplicating a variable in the workspace .....	28
Figure 24: Selecting statistics to display in the Workspace panel.....	29
Figure 25: Adapting workspace preferences.....	29
Figure 26: Saving or clearing the workspace via the respective icons on the Home tab.....	30
Figure 27: Loading a saved workspace from the Files panel.....	30
Figure 28: Variables editor .....	31
Figure 29: Selecting display format of variables .....	31
Figure 30: Displaying available applications via the Apps tab.....	32
Figure 31: Interacting with Copilot through Copilot Chat panel.....	33
Figure 32: Accessing Copilot actions from the Editor .....	34
Figure 33: Code suggestions provided by MATLAB Copilot .....	34
Figure 34: File storage in MATLAB Drive .....	35
Figure 35: Using MATLAB Online via web browser .....	36
Figure 36: Setting up MATLAB Drive and MATLAB Drive Connector on your computer .....	36
Figure 37: Advanced search at OTA website.....	38
Figure 38: Corpora from English-Corpora.org .....	39
Figure 39: Results from a query for adjectives ending in “-less” in COHA.....	40
Figure 40: Imported table of adjectives ending in “-less” displayed in MATLAB.....	41
Figure 41: Querying immediate collocates of “health care” in COCA.....	41
Figure 42: Immediate collocates of “health care” in COCA.....	42
Figure 43: MATLAB variable with collocates of “health care” and absolute frequencies from COCA..	42
Figure 44: MATLAB word cloud for immediate collocates of “health care” based on COCA data .....	43
Figure 45: Screenshot of the Sketch Engine corpus dashboard for the BASE Corpus .....	44

Figure 46: Designing a query with the CQL Builder using Sketch Engine.....	45
Figure 47: Downloading concordance results from the BASE corpus in Sketch Engine.....	45
Figure 48: Downloading frequency results from the BASE corpus in Sketch Engine .....	46
Figure 49: Creating a bar chart in MATLAB .....	46
Figure 50: Customised MATLAB bar chart based on query results from BASE corpus .....	47
Figure 51: Concordance view of Europarl data on the OPUS platform after a CQP query .....	48
Figure 52: CQPweb at Saarland University.....	48
Figure 53: CQPweb at Lancaster University .....	49
Figure 54: CQP query for nouns after “electric” or related words in RSC data on Saarland University’s CQPweb interface.....	50
Figure 55: Concordance view of query results from RSC in CQPweb.....	50
Figure 56: Frequency table for query results from RSC in CQPweb.....	50
Figure 57: Text file with downloaded frequency table .....	51
Figure 58: Distribution table for query results from RSC in CQPweb .....	51
Figure 59: Bar charts with distribution of query results from RSC in CQPweb .....	52
Figure 60: Sorting values in MATLAB Variables editor and displaying selected statistical measures in workspace .....	52
Figure 61: Icons representing different data types in the workspace .....	53
Figure 62: Examples of variable assignments for numeric types .....	56
Figure 63: Creating a numeric matrix.....	57
Figure 64: Converting a full matrix to sparse format .....	58
Figure 65: Character arrays in workspace and in Variables editor and cell arrays obtained by extracting words from a character array.....	60
Figure 66: Creating a 3x2 string array .....	62
Figure 67: Examples of working with string variables.....	64
Figure 68: Preprocessing a text string and calculating the length of each word .....	66
Figure 69: Creating a cell array.....	67
Figure 70: MATLAB script for text preprocessing and type-token ratio calculation .....	69
Figure 71: Creating a categorical array for part-of-speech tags for a set of words .....	70
Figure 72: Creating categorical arrays from a string array and from a cell array of character vectors.....	71
Figure 73: Converting data into a categorical array.....	72
Figure 74: Creating a table via commands .....	73
Figure 75: Creating a table using row vectors .....	74
Figure 76: Transposing a variable.....	74
Figure 77: Resulting table after transposing row vectors .....	75
Figure 78: Creating a structure.....	76
Figure 79: Creating a bilingual dictionary in MATLAB .....	77
Figure 80: Modifying an entry from a dictionary .....	77
Figure 81: Creating a new variable via the ‘New Variable’ icon from the toolstrip.....	78
Figure 82: Size of corpus sections in BNC.....	79
Figure 83: MATLAB variable for size of BNC sections in millions of words .....	79
Figure 84: Viewing and editing data in the Variables editor .....	80
Figure 85: Opening several windows in the Variables editor for different variables .....	81
Figure 86: Importing numeric data into MATLAB from an Excel file.....	82
Figure 87: Importing data from an Excel file in the form of a table and assigning it a name .....	82
Figure 88: Importing selected data from an Excel file containing several tabs .....	83
Figure 89: Parallel concordance view in Europarl, hosted on Sketch Engine .....	84
Figure 90: Downloading query results from parallel concordance in Europarl on Sketch Engine.....	84

Figure 91: Importing selected data into a MATLAB string array from a CSV file .....	85
Figure 92: Using the <code>fileread</code> function to read the contents of a text file.....	86
Figure 93: Saving workspace variables via the <code>save</code> command .....	87
Figure 94: Creating a new script.....	90
Figure 95: Creating a new script via the Editor tab .....	90
Figure 96: Saving a script.....	91
Figure 97: Running a MATLAB script .....	93
Figure 98: Running a section from a script.....	94
Figure 99: Using a script to create a customised bar chart.....	95
Figure 100: Using a short script to create a bar chart that needs further modifications .....	96
Figure 101: Creating a copy of an existing script in the Files panel .....	96
Figure 102: Example of a live script.....	98
Figure 103: Adding a section break in a live script.....	98
Figure 104: Error message in MATLAB .....	100
Figure 105: Available plot types in the Plots tab toolstrip for selected data.....	104
Figure 106: Displaying available plot options with incompatible options greyed out .....	105
Figure 107: Customising settings of figures and plots .....	106
Figure 108: Saving a figure .....	106
Figure 109: Choosing a file format for exporting a figure .....	107
Figure 110: Printing a figure to a PDF file.....	107
Figure 111: Labels and annotations section on Figure toolstrip .....	108
Figure 112: Drop-down menu for figure labels and annotations.....	108
Figure 113: Opening the Property Inspector of a figure .....	109
Figure 114: Adapting axes properties via the Property Inspector .....	110
Figure 115: Accessing the Property Inspector.....	110
Figure 116: Accessing metadata details of Brown Corpus from Sketch Engine's Corpus Info section .....	112
Figure 117: Selecting multiple sections of fictional texts from Excel table for import into MATLAB .....	113
Figure 118: Creating a vertical bar chart .....	114
Figure 119: Navigating to Axes Properties in Property Inspector .....	115
Figure 120: Adjusting x-tick labels through the Property Inspector .....	115
Figure 121: Adding an x-axis label.....	116
Figure 122: Adjusting axis label font size via the figure's Format tab.....	117
Figure 123: Bar chart customised via the Property Inspector.....	118
Figure 124: Bar chart for sizes of sections in BNC and Strathy Corpus .....	119
Figure 125: Changing y-axis tick labels .....	120
Figure 126: 3-D bar chart for sizes of sections in BNC and Strathy Corpus.....	121
Figure 127: Horizontal bar chart for sizes of sections in BNC and Strathy Corpus .....	122
Figure 128: Creating a stacked bar chart with a MATLAB script .....	124
Figure 129: Creating a stacked bar chart using the interactive plotting options .....	124
Figure 130: Vertical stacked bar chart created by selecting the respective icon.....	125
Figure 131: Updated visualisation of stacked bar chart with title, axis labels and legend .....	125
Figure 132: Selecting specific charts to appear among the favourite plot types.....	128
Figure 133: Modifying a pie chart via the Property Inspector .....	129
Figure 134: Creating and customising a pie chart for BASE corpus data by using commands .....	130
Figure 135: Alternative way of creating and customising a pie chart for BASE corpus data by commands .....	131

Figure 136: Creating a 3D pie chart and a donut chart for BASE corpus data .....	132
Figure 137: Creating a customised pie chart from RSC data using a script.....	133
Figure 138: Customising a pie chart created by selecting the respective icon from the Plots tab.....	134
Figure 139: Pie chart with exploded wedges to highlight specific slices for emphasis.....	135
Figure 140: Frequency of "start" + <i>-ing</i> form in COHA.....	136
Figure 141: Line plot for frequencies of "start" + <i>V-ing</i> in COHA.....	137
Figure 142: Example of a line plot in MATLAB .....	138
Figure 143: Modifying axis limits via the Property Inspector.....	138
Figure 144: Plotting descriptive statistical values .....	139
Figure 145: Stem plot for ellipses in five registers from GECCo corpus.....	142
Figure 146: Sorting values of a variable with frequency data for cohesive ellipses .....	142
Figure 147: Stem plot with sorted data for ellipses in five registers from GECCo corpus .....	143
Figure 148: Scatter plot for frequency of a word per text vs. text length .....	144
Figure 149: Adjusted scatter plot showing frequency of a word per text vs. text length.....	145
Figure 150: Scatter plot for frequencies of cohesive nominal ellipses in pairs of original and translated texts from three registers from GO/ETTRANS in GECCo corpus.....	148
Figure 151: Scatter plot for frequencies of initialisms and their full forms in English data from EuroParl-UdS .....	151
Figure 152: Adjusting axis limits via the Property Inspector and applying a model fit using the Basic Fitting tool .....	152
Figure 153: 'Boxplot' and 'boxchart' icons in Plots Gallery.....	153
Figure 154: Box plot showing distribution of text length in English section EO from GECCo corpus .	154
Figure 155: Box plot for text length of English and German sections from GECCo corpus .....	155
Figure 156: Creating box plot from one variable with two columns for English and German data from GECCo corpus.....	156
Figure 157: Notched box plot created by using the <code>boxplot</code> function .....	157
Figure 158: Notched box plot created by using the <code>boxchart</code> function.....	157
Figure 159: Box plot for cohesive ellipses in five registers from German and English sections in GECCo corpus .....	160
Figure 160: Histogram showing text length distribution in German section from GECCo corpus.....	162
Figure 161: Histogram for text length distribution in German section from GECCo corpus with customised layout features .....	163
Figure 162: Adjusting x-tick steps and x-tick label rotation .....	163
Figure 163: Heatmap visualisation of translation counts for each source language over time.....	166
Figure 164: Customising a heatmap via the Property Inspector.....	167
Figure 165: Creating a heatmap via the heatmap icon on the toolbar .....	168
Figure 166: Word cloud for collocates of "health care" (freq. > 24) based on COCA data.....	170
Figure 167: Customised word cloud for collocates of "health care" in COCA .....	171
Figure 168: Importing data for word cloud from text file obtained through CQPweb interface.....	172
Figure 169: Word cloud with most frequent patterns of nouns after adjectives that have a minimum length of 10 characters (RSC microbiology texts, 1860s–1920) .....	173
Figure 170: Word cloud created from a string.....	174
Figure 171: Modifying subplots in a figure interactively.....	177
Figure 172: Generating subplots with tailored layout features via script.....	178
Figure 173: Sorted frequencies of cohesive ellipses in 5 English and German GECCo registers, overlaid in a single stem plot for direct comparison.....	183
Figure 174: Selecting distinctive markers via the figure's Format tab.....	183
Figure 175: Labels and annotation options in the MATLAB Figure tab.....	185
Figure 176: Using the Property Inspector .....	186

Figure 177: Accessing submenu of Property Inspector to modify the features of the x-axis label ....	187
Figure 178: Workspace showing variables BNC and TIME, with <code>total_counts</code> created by summing the two.....	192
Figure 179: Sorting the values of a variable interactively in the Variables editor .....	204
Figure 180: Sorting values from a table in the Variables editor.....	205
Figure 181: Displaying descriptive statistics in Workspace panel.....	206
Figure 182: Displaying descriptive statistics in a plot.....	207
Figure 183: Minimum and maximum values of a variable, shown in Workspace panel.....	208
Figure 184: Mean, median and mode of RSC text lengths for year 1910, automatically calculated by MATLAB in the workspace and saved to individual variables via commands .....	213
Figure 185: Range, variance and standard deviation of RSC text lengths for year 1910 , automatically calculated by MATLAB and saved to individual variables via commands .	214
Figure 186: Output of <code>summary</code> function for a numerical variable with key statistical values.....	215
Figure 187: Output of <code>summary</code> function for rows and columns of a matrix .....	216
Figure 188: Output of <code>summary</code> function for categorical array .....	216
Figure 189: Using the Distribution Fitter app.....	221
Figure 190: Histogram of text lengths with fitted distribution curve in the Distribution Fitter app ..	222
Figure 191: Creating a probability plot in the Distribution Fitter app.....	222
Figure 192: Exploring MATLAB's Probability Distribution Function tool.....	223
Figure 193: Output components from <code>anova1</code> function.....	233
Figure 194: Scatter plot for ellipses in pairs of original and translated texts from three registers from GO/ETRANS in GECCo corpus .....	239
Figure 195: Calculation of correlation coefficient.....	240
Figure 196: Calculation of correlation coefficient and p-value for data from GECCo corpus.....	241
Figure 197: Changing number display format to show more decimal places of correlation coefficient and p-value .....	242
Figure 198: Scatter plot for initialisms vs. their full forms in English EuroParl-UdS data with linear regression line .....	242
Figure 199: Calculation of correlation coefficient and p-value for data from EuroParl-UdS.....	243
Figure 200: Extracting text from a .txt file.....	247
Figure 201: Extracting text from a web page .....	248
Figure 202: Opening the Preprocess Text Data task in the Live Editor .....	249
Figure 203: Opening the Preprocess Text Data task via a command.....	250
Figure 204: Using the Preprocess Text Data task in the Live Editor.....	251
Figure 205: Results from the Preprocess Text Data task for a sample text .....	252
Figure 206: Displaying code generated in Preprocess Text Data task .....	252
Figure 207: Selecting option to parse dependencies .....	253
Figure 208: Code generated by Preprocess Text Data task as specified in Figure 207 and details of preprocessed text.....	254
Figure 209: Grammatical dependency parse tree generated by using the <code>sentenceChart</code> function .....	255
Figure 210: Displaying part-of-speech information with the <code>tokenDetails</code> function.....	257
Figure 211: Identifying named entities and displaying results.....	258
Figure 212: Script for lemmatising a text .....	259
Figure 213: Converting tokens to lowercase and uppercase .....	260
Figure 214: Displaying list of English stop words .....	261
Figure 215: Displaying list of German stop words.....	261
Figure 216: Removing customised list of stop words from text.....	262

Figure 217: Splitting a text into sentences .....	263
Figure 218: Creating a word cloud for a bag-of-words model .....	268
Figure 219: Retrieving terms associated with columns of TF-IDF matrix.....	272
Figure 220: Creating a bag-of-n-grams model .....	274
Figure 221: Most frequent trigrams extracted from bag-of-n-grams model after stop word removal from dataset.....	275
Figure 222: Finding closest semantic neighbours to a given word .....	278
Figure 223: Displaying semantically related words and proximity values in embedding space derived from combination of two words .....	279
Figure 224: Creating a 2D text scatter plot for word embeddings.....	282
Figure 225: Creating a 3D text scatter plot for word embeddings.....	283
Figure 226: Training a word embedding model .....	286
Figure 227: Text scatter plot visualising trained word embeddings .....	287
Figure 228: Zoomed-in region of text scatter plot .....	288
Figure 229: Summarising the content of a text with the <code>extractSummary</code> function.....	292

## List of tables

Table 1: Size of different genres in Brown Corpus, hosted on Sketch Engine.....	112
Table 2: Size of corpus sections in BNC and Strathy Corpus .....	118
Table 3: Frequencies (per million words) of three epistemic stance markers in different corpora .	123
Table 4: Number microbiology articles by female (co-)authors in RSC.....	132
Table 5: Normalised frequency of “start” + <i>-ing</i> form in COHA corpus throughout 20th century ...	136
Table 6: Cohesive ellipses per text in five registers from English part of GECCo corpus .....	141
Table 7: Frequency of cohesive nominal ellipses in three registers from corpus parts with German originals and English translations from GECCo corpus .....	147
Table 8: Frequencies of initialisms and corresponding full forms in English section of EuroParl-UdS .....	150
Table 9: Cohesive ellipsis per text in five registers from English and German part of GECCo corpus.....	159
Table 10: Immediate collocates of “health care” in COCA .....	169
Table 11: Adjective + noun patterns in RSC microbiology texts (adj. with at least 10 characters and patterns that occur at least 50 times).....	172
Table 12: Size of written corpus registers with original texts in GECCo corpus.....	176
Table 13: Cohesive ellipses per text in five registers (English and German part of GECCo corpus) ...	181
Table 14: Absolute frequencies of four words in BNC and TIME Magazine Corpus .....	192
Table 15: Frequencies of hyphenated verbs and nouns with prefixes (e.g., ‘co-’, ‘re-’, ‘pre-’) in five corpus texts .....	194
Table 16: Absolute frequencies of four words in TIME Magazine Corpus overall and specifically in 1950s.....	195
Table 17: Usage of “will go” and “going to go” in 2 decades in COHA.....	230
Table 18: Individual and total frequencies of “will go” and “going to go” in 2 decades in COHA .....	231
Table 19: Word frequency counts for two sample documents.....	266

## Acknowledgements

I would like to express my deepest gratitude to everyone who supported me throughout the process of writing this book. This book was driven by my passion for MATLAB and my desire to highlight its potential for linguists. I want to make it clear that I have no financial or professional relationship with MathWorks®. However, I would like to thank Kathi Kugler from the MathWorks team for her invaluable feedback. I also greatly appreciate the feedback provided on the book draft by Christopher Creuzig from MathWorks, as well as the organisational support from the members of the MATLAB book programme.

I am also deeply grateful to my colleagues at the University of Mannheim from the English Linguistics (Anglistik IV) Department for their constructive feedback, particularly Carola Trips, Michael Percillier, Tara Struik, Juliane Elter, Lena Kaltenbach, Alissa Selivenko and Jessica Schnell. Additionally, I would like to thank Ruth Keßler from Leipzig University for her statistical consulting.

I would also like to thank Ute Römer-Barron and the anonymous reviewers for their constructive feedback. I have occasionally encountered the concern that MATLAB might be too complex or that one might wonder why not simply use tools like Python or R for linguistic analysis. I recognise that many linguistic scholars and lecturers still prefer these more widely-used tools in the field. My intention in writing this book is to present an alternative that I have found to be particularly user-friendly. There is, however, a noticeable lack of documentation of MATLAB with examples specifically aimed at linguists. I hope this book will inspire many to try using MATLAB for their linguistic projects, especially since its license is often provided free of charge by many universities. Additionally, individuals who are not affiliated with such institutions can still access a certain number of free hours per month.

Finally, to my students, past and present, who inspired many of the ideas in this book, I offer my heartfelt thanks. Your questions, challenges and enthusiasm for learning have pushed me to innovate new ways in which MATLAB can be applied to research fields that have not yet explored its potential.

## About the author

Katrin Menzel has a PhD in linguistics and degrees in interpreting and translation studies from Saarland University, Germany. She has completed a habilitation thesis in applied linguistics and English linguistics in which she focused on word formation patterns in specialised texts from the Late Modern English period to the present, using a corpus-based approach. She has taught “MATLAB for Linguists” courses for several years at various universities, primarily to BA and MA students in linguistics and English studies with diverse levels of experience in data visualisation, analysis and programming. Currently, she is an Interim Professor of Translatology at Leipzig University.

# 1 Overview of MATLAB<sup>®</sup> and possible applications in corpus-linguistic research

## 1.1 What is MATLAB<sup>®</sup>?

MATLAB<sup>®</sup>, which stands for MATrix LABoratory, is a powerful commercial software developed by MathWorks<sup>®</sup>. It offers a unique set of features that make it highly applicable to corpus-linguistic research, a field that examines language by analysing large datasets of texts (corpora), as well as for various other areas of linguistics. MATLAB's core strength lies in its advanced computational capabilities, which are ideal for handling large datasets, performing statistical analysis and creating visual representations of data – all tasks commonly required in linguistic research.

One of the key advantages of MATLAB is its user-friendly interface, which makes it accessible even to those without a strong background in programming. In contrast to some other programming languages commonly used in the field of linguistics, MATLAB provides a very intuitive environment for conducting complex data analysis and visualisation. While other programming languages and tools have already gained popularity in linguistics for statistical analysis and data visualisation, MATLAB, although less commonly known in the field, offers a highly beginner-friendly environment. It is particularly well-suited for a wide range of tasks in corpus linguistics, including text processing and preprocessing, frequency analysis, statistical analysis and data visualisation. Its powerful computational capabilities and versatility make MATLAB an invaluable tool for handling linguistic data and conducting in-depth analysis across various linguistic domains.

Despite its strong presence in fields like engineering and the natural sciences, MATLAB has not yet been fully explored in the context of linguistics, particularly in corpus-linguistic research. This book aims to fill that gap by serving as both a textbook and an introduction for those who have never worked with MATLAB, demonstrating how it can be used for a variety of corpus-linguistic applications.

For MATLAB and product information, you may contact:

The MathWorks, Inc.

3 Apple Hill Drive, Natick, MA, 01760-2098 USA

Tel: 508-647-7000

Fax: 508-647-7001

E-mail: [info@mathworks.com](mailto:info@mathworks.com)

Web: <https://www.mathworks.com><sup>1</sup>

How to buy: <https://www.mathworks.com/store>

Find your local office: <https://www.mathworks.com/company/worldwide>

Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)

---

<sup>1</sup> All URLs cited in this book were last accessed in early 2026. Despite careful review of all referenced links at the time of publication, the author cannot guarantee the continued availability, accuracy or content of external websites. Web content may be subject to change, relocation or removal without notice. To access material as it existed at the time of publication, the Internet Archive's Wayback Machine (<https://archive.org/web/>) may be helpful. No liability is assumed for the content of external links; responsibility for the content and hosting of linked pages lies solely with their respective operators.

## 1.2 How this book can enrich your corpus-linguistic studies

This book is the first of its kind, specifically designed for linguists interested in using MATLAB as a powerful yet very accessible computing environment and programming language for analysing data from electronic text corpora. Unlike other MATLAB resources primarily aimed at engineers or scientists, this work focuses on practical and linguistically relevant examples. The exercises presented have been tested in classrooms with students from diverse linguistic backgrounds, and the feedback has been overwhelmingly positive. Whether you are an advanced BA or MA student working on a term paper or a thesis or a linguist aiming to integrate MATLAB into your research, this book provides a beginner-friendly introduction to get you started. It also presents moderately advanced concepts to help you gradually deepen your understanding, while deliberately avoiding a focus on complex programming and highly advanced statistical methods and complex algorithms. Instead, the focus remains on practical applications tailored to corpus-linguistic research. From querying linguistic corpora and importing data into MATLAB to working with the data, this book provides all the tools you need to begin integrating MATLAB into your research. Many tasks can be performed directly within MATLAB's intuitive interactive environment. Often, simple clicks can handle operations like data import and plot generation, which eliminates the need for advanced coding skills at the outset.

If you are new to MATLAB and have little or no experience with data analysis, programming or corpus linguistics in general, this book is designed to guide you through the learning process in a clear and accessible way. The primary focus of this book is on learning by doing. As you move through the chapters, you will be actively working on your computer with MATLAB. This practical, hands-on approach encourages you to learn by interacting with the software and applying the concepts as you go, rather than merely reading about them. This textbook is designed to be flexible. While it can be read from beginning to end, it is not necessary to follow this sequence. You can focus on specific chapters that address your immediate needs and interests. The examples provided for each application and function are explained in a way that allows you to replicate them easily. These should serve as practical exercises that you can perform on your own computer, with the opportunity to compare your results with the screenshots shown in the book to ensure that everything works as expected. Text data or corpora examples will be provided, or guidance will be given on how to obtain them from publicly available resources so that you can repeat the exercises on your own machine. You can then also modify these examples to reinforce your understanding and practice with your own comparable or more complex data sets.

Additionally, there is a dedicated chapter in this book that provides an overview of how to access, query, extract and download linguistic data from some publicly available corpora and text resources such as the ones on English-Corpora.org, the Sketch Engine interface, CQPweb interfaces and the Oxford Text Archive. Many of the practical examples in this book, including those for creating variables, generating plots, performing calculations and text analytics, draw on data obtained from these publicly available resources. These examples showcase clear and straightforward cases from well-known and frequently used corpora and datasets, and the examples and case studies can be easily adapted to various linguistic research contexts. Some of the examples are simplified versions of larger published case studies from research projects the author was involved in, designed to be a bit smaller here so that the example datasets can be seamlessly integrated throughout the book. This also ensures that the corpus study examples are easy to replicate, as the data can be easily queried from the corpora and, in many cases, is also provided in tables or in another format within the book.

The aim of this book is to show you how use MATLAB for a wide range of tasks in corpus linguistics, including data analysis, visualisation and exploration of linguistic patterns. There are many linguistic fields in which corpus data play a crucial role. In general linguistics, including fields like English linguistics, corpus data is used to study various aspects of language, such as syntax, lexis, morphology, semantics, register variation or language acquisition. In sociolinguistics, for instance, corpus data can help to examine language variation across different groups, such as exploring how dialects differ by region or analysing language use in relation to age, gender or social background. In historical linguistics, corpus data is essential for tracking language change over time, for example, by studying the evolution of lexical or syntactic choices. Crosslinguistic studies and translation research also make use of corpus data to compare language structures across different languages. This book will show how MATLAB can be a valuable tool for managing, analysing and visualising your data and for supporting research in these areas and beyond. Additionally, throughout the book, there are references to other materials where readers can deepen their knowledge on topics that are not covered here in full detail due to space constraints, particularly those involving theoretical and mathematical considerations. For instance, supplementary sources expand on topics such as inferential statistics and text analytics methods.

In sum, this book serves as both an introduction to MATLAB and a guide for applying it specifically to corpus-linguistic tasks. It covers essential tools and techniques, from installing MATLAB and relevant toolboxes to working with linguistic data and creating visualisations. By the end of this book, you will have a solid foundation in using MATLAB for linguistic research, and you will be empowered to take on increasingly more complex linguistic projects by using MATLAB with ease. In this book, MATLAB version R2026a (pre-release) is used for examples and instructions. Typically, MATLAB receives small updates twice a year, but these do not bring substantial changes, so it can be expected that the content of this book will remain relevant and applicable for the foreseeable future. Note that the 2025a release marked a major update, introducing a range of new features, with only a few minor changes since then. You might be interested to know that MATLAB offers free access to unlicensed users for up to 20 hours per month, which makes it an affordable option for those just getting started. Many universities also provide campus licenses, granting full access to the software. This makes it even more accessible for students and researchers.



# 2 Getting started with MATLAB®

## 2.1 Installing MATLAB®

### 2.1.1 Overview of installation options and key steps

This chapter outlines the steps to install MATLAB on your personal computer, whether you are using Windows, Linux or macOS. It also explains how to set up your MathWorks account and license to get started with MATLAB, so you can begin working with the software.

Note that throughout this book, links to the official MathWorks documentation are provided where relevant for installation instructions and for guidance on explanations of function usage. Since exact URLs may change over time, you can also use the search function on the MathWorks documentation site to look up keywords related to the topic you need and find the most current information under the corresponding page. Installing MATLAB on your computer is a straightforward process, but it does require a few key steps to ensure that everything works properly. This chapter provides general information to guide you through the installation process. While the steps are broadly applicable, specific details may vary depending on how you obtain MATLAB (e.g., through a university license or a personal subscription).

There are different ways to obtain MATLAB depending on your situation. It is important to note that many universities offer access to MATLAB through campus-wide or student licenses. These licenses allow you to install MATLAB on your personal machine without additional cost. If you are a student or staff member, you should first check your university's website or IT portal for information about access and installation instructions. Universities often provide tailored steps for installation based on their specific licensing agreements, and they may also offer technical support for any issues that arise during the installation process. If you are not affiliated with a university or your institution does not provide access, you can purchase a personal MATLAB license directly from MathWorks. MathWorks also offers free trial versions for those who wish to explore MATLAB for a limited period. When you install MATLAB on your computer, it is essential to ensure that your MathWorks account is linked to the correct license type (e.g., student, academic or trial) during the installation process to ensure proper activation.

Apart from installing MATLAB on your computer, you have the option to access it directly through your web browser using MATLAB Online. This platform lets you run MATLAB code, retrieve files from MATLAB Drive, and access many of the same features as the desktop version. If your university provides access to MATLAB Online, you can log in with your institutional email to use it. There is also the possibility for everyone (including unlicensed users) to use MATLAB Online for up to 20 hours each month for free, with the only requirement being a MathWorks account, cf. <https://www.mathworks.com/products/matlab-online/matlab-online-versions.html>.

Before starting the installation process, there are a few important points to keep in mind:

#### ➤ **MathWorks account**

To download and install MATLAB, you will need to create a MathWorks account if you do not already have one. This is required to access your MATLAB license and product downloads. If you are affiliated with a university or organisation that provides a campus-wide license, make

sure to use your institutional email address when creating your account. You can create a MathWorks account by visiting the MathWorks account creation site (<https://www.mathworks.com/mwaccount/account/create>) and following the instructions. You will need to provide details such as your name, email address and affiliation with your institution.

### ➤ University access

As mentioned above, many universities offer free access to MATLAB for students and staff, including the main programme and toolboxes. Check your university's website or IT portal for specific installation instructions. Ensure that your MathWorks account is linked to the correct academic license and contact your university's IT support or MATLAB's support if you encounter any issues.

### ➤ System requirements

Before installing MATLAB, ensure your computer meets the system requirements for your platform (Windows, macOS or Linux). System requirements for the latest version (such as MATLAB R2025b and R2026a prerelease at the time of writing this book) are available on the MathWorks website (<https://www.mathworks.com/support/requirements/matlab-system-requirements.html>) and will differ slightly depending on your operating system and hardware configuration. It is important to verify that your system has enough resources (e.g., RAM, storage and processor capabilities) to run MATLAB smoothly.

Here are some of the system requirements for MATLAB R2025b and R2026a prerelease: On *Windows*, the software currently supports Windows 10 (version 22H2), Windows 11 (version 23H2 or higher), Windows Server 2022 and Windows Server 2025. It requires a minimum of an Intel or AMD x86-64 processor with two cores, although a processor with four or more cores and AVX2 instruction set support is recommended. A future release of MATLAB will require a processor with AVX2 instruction set support. The minimum RAM is 8 GB, with 16 GB being ideal. Storage requirements include 4.2 GB for MATLAB alone, 4-6 GB for a typical installation or up to 24 GB for the full suite of products and toolboxes. A GPU supporting WebGL 2.0 with at least 2 GB of memory is also recommended for performant graphics rendering.

For *macOS*, MATLAB R2025b and R2026a prerelease are currently compatible with macOS Sequoia (15), macOS Sonoma (14) or macOS Tahoe (26). The system can be powered by an Intel x86-64 processor with two or more cores or any M-series chip, but note that MATLAB R2025b is the final release of MATLAB for Intel Macs. The minimum RAM is 8 GB, with 16 GB recommended, and storage requirements are 3.8 GB just for MATLAB, 4-6 GB for a typical installation or 17 GB for the full suite. Any Mac that supports macOS Sonoma will have a compatible GPU.

For *Linux*, supported distributions currently include Ubuntu 24.04 LTS, Ubuntu 22.04 LTS, Debian 12, Debian 13, Red Hat Enterprise Linux 9 and others. The system requires an Intel or AMD x86-64 processor with at least two cores. It also needs a minimum of 8 GB RAM (16 GB recommended), 4.1 GB of storage for MATLAB only, 4-6 GB for a typical installation or up to 25 GB for an all products installation. A GPU supporting WebGL 2.0 with at least 2 GB of memory is recommended.

Please refer to the full details in the MathWorks documentation for additional information about your specific platform.

## ➤ MATLAB toolboxes

During the installation of MATLAB on your computer, you can also choose which toolboxes you want to use. Note that this is not possible when using the online version. MATLAB toolboxes are specialised collections of functions that enhance and expand the core capabilities of MATLAB. Each toolbox is designed for specific tasks or fields, providing additional features that users can apply within MATLAB (cf. <https://www.mathworks.com/products.html>). Many universities provide access to these toolboxes for free, typically through a campus-wide license to ensure that students and staff can use a wide variety of different toolboxes. We will particularly need the Statistics and Machine Learning Toolbox™ and the Text Analytics Toolbox™ (cf. also Chapter 10) for some of the tasks in this book. You can directly select them during the installation process (Figure 1).

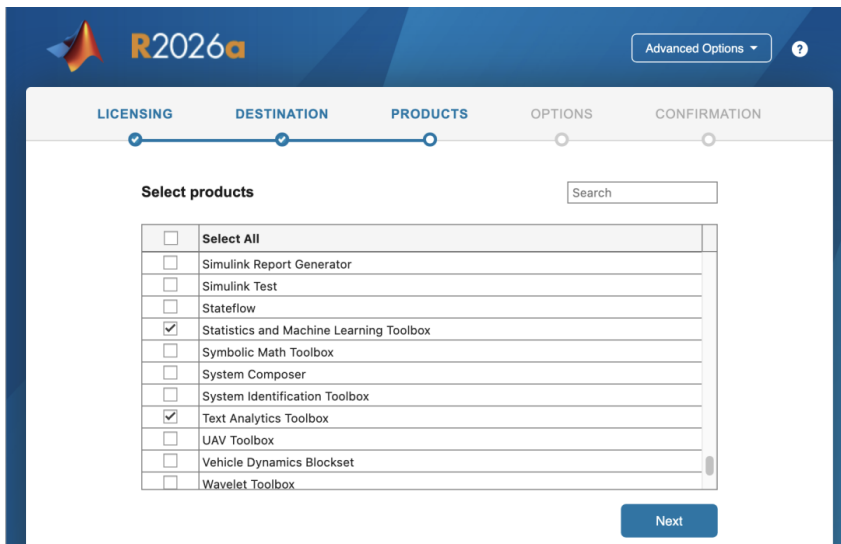


Figure 1: Selecting toolboxes during the installation process

## ➤ Updates and licensing

If you have an existing MATLAB installation, it is a good practice to check for updates or upgrade to the latest version. You can do this either from within MATLAB or by visiting the MathWorks website. Be sure that your license is still valid, particularly if you are using a trial or academic license.

### 2.1.2 Step-by-step installation process

#### ➤ Download the MATLAB Installer

To begin, you need to download the MATLAB installer. Log in to your MathWorks account and navigate to the Downloads page. Select the version of MATLAB you wish to install (e.g., MATLAB R2026a) and choose the installer that matches your operating system (Windows, macOS or Linux).

### ➤ Run the Installer

After the download is complete, run the installer and follow the prompts:

- **Windows:** Double-click the installer and follow the on-screen instructions. The default installation folder is `C:\Program Files\MATLAB\R20XXy` (e.g., `C:\Program Files\MATLAB\R2026a` for the MATLAB R2026a version), but you can change this if needed.
- **macOS:** Unzip the downloaded `.dmg` file and double-click to mount the installer as a virtual disk. Then, simply double-click the installer and follow the on-screen instructions to finish the installation process. The default installation folder is `/Applications/MATLAB_R20XXy.app`.
- **Linux:** Extract the downloaded `.zip` file to a folder and navigate to the extracted folder. Use, for instance, these commands:

```
unzip matlab_R2026a_Linux.zip -d ./
matlab_R2026a_Linux
cd ./matlab_R2026a_Linux
```

To complete the installation, navigate to the installation folder, run the install script, and follow the prompts. Use the command `xhost +SI:localuser:root` to allow root access to the graphical display, then run `sudo -H ./install` to start the installation. Afterward, execute `xhost -SI:localuser:root` to revoke root access. The `sudo` command is needed only if you do not have write permissions for the installation folder, typically the default folder at `/usr/local/MATLAB/R20XXy`.

For more detailed guidance, visit the MathWorks installation documentation (<https://www.mathworks.com/help/install/install-products.html>).

### ➤ Sign in to your MathWorks account

During installation, you will be prompted to sign in to your MathWorks account. This is necessary to link the installation with your license. Use the email associated with your university's MATLAB license if you have one. If not, you can use a personal account or set up a trial license.

### ➤ Select the installation type

You will be given options on how to install MATLAB:

- **Typical installation:** This option installs MATLAB with the most commonly used tools and features.
- **Custom installation:** If you need specific toolboxes or features, you can choose this option to customise the installation.

If you are installing MATLAB through your university, make sure you choose the appropriate options based on the license you have access to.

### ➤ **Activate your license**

Once MATLAB is installed, the next step usually is to activate it. You will be asked to provide the activation key if necessary. This key should be available from your university's software portal or provided to you by your department. Follow the prompts to complete the activation process. You can typically directly add a shortcut to your desktop during the installation process.

If you encounter any issues with activation, you can refer to the MathWorks support page or contact your university's IT department for assistance.

### **2.1.3 Troubleshooting common installation issues**

While MATLAB installation is usually straightforward, some users may encounter issues. Here are a few common problems and their solutions:

#### ➤ **Internet connection issues**

If your internet connection is interrupted during installation, try restarting the installer. If problems persist, ensure your network connection is stable and check firewall settings that may block the installer.

#### ➤ **Insufficient disk space**

MATLAB requires a certain amount of disk space, especially if you are installing additional toolboxes. Ensure that your system has enough storage space. If you are using Linux, you may need to adjust the temporary file location by changing the `tmp` directory.

#### ➤ **Licensing issues**

If you cannot find your license during installation or are asked for an activation key, verify that you have the correct MathWorks account linked to your university's license. For institutions with a campus-wide license, you may need to contact your university's IT support for assistance.

If you continue to have problems, it might be best to contact the MathWorks support team ([https://www.mathworks.com/support/contact\\_us.html](https://www.mathworks.com/support/contact_us.html)). You may include any error messages, log files or screenshots that are relevant to the issue in your help request.

### **2.1.4 Getting updates of installed MATLAB® release**

In this book, MATLAB version R2026a prerelease was used for the examples and instructions. While MATLAB regularly releases new versions with smaller updates, the 2025a release was a major update that introduced significant new features not available in earlier versions. Users are encouraged to refer to the official Release Notes for new products and updates to stay informed about the latest features, improvements and changes: <https://www.mathworks.com/help/relnotes/>.

You can regularly download and install newer versions of MATLAB than the one currently installed. For instance, if you have R2026a, you might be able to download a later R2026b

version or versions released in 2027 etc. If an update is available, MATLAB will notify you when the programme is started. To install the update, click the notification icon and select “Install Update”.

You can manually check for updates by ensuring an active internet connection and selecting Help > Check for Updates in the Home tab, which will open the Add-On Manager with the Updates tab. If an update is available, you can install it by clicking the Update button, and MATLAB will notify you if no updates are available or if you are already on the latest version.

### 2.1.5 Installing Statistics and Machine Learning Toolbox™ and Text Analytics Toolbox™

During the installation of MATLAB, you can choose which toolboxes you want to install. In this book, some examples require MATLAB’s Statistics and Machine Learning Toolbox™ and the Text Analytics Toolbox™. The core features of these toolboxes are outlined below.

The Statistics and Machine Learning Toolbox™ offers a wide range of basic and advanced features for analysing, visualising and modelling data. Many of its capabilities, such as various types of hypothesis testing, cluster analysis, regression, ANOVA, dimensionality reduction and feature extraction, are designed for slightly more advanced users. While this book will briefly introduce some of these topics, it will not cover them in great detail, as it serves as an introductory guide. For further details, also visit the MathWorks description and documentation of the toolbox:

*Statistics and Machine Learning Toolbox*

<https://www.mathworks.com/products/statistics.html>

*Get started with Statistics and Machine Learning Toolbox*

<https://www.mathworks.com/help/stats/getting-started-12.html>

The Text Analytics Toolbox™ is another powerful asset for processing and analysing textual data. In this book, we will use it to generate visualisations like word clouds (Chapters 7.13, 10.4.1 and 10.4.2), dependency graphs (Chapter 10.3) and text scatter plots (Chapter 10.4.3), but the toolbox’s capabilities extend far beyond that. The toolbox, for instance, enables text pre-processing by extracting and preparing raw text from various sources such as documents or websites, which can then be analysed linguistically. It also supports feature extraction and modelling using machine learning techniques. Moreover, it offers dimensionality reduction and clustering tools to help analyse large textual datasets by identifying key features or grouping similar data points, useful for understanding linguistic patterns or categorising language varieties. Chapter 10 will give an overview of some of the toolbox’s functionalities. For further details, visit the MathWorks description and documentation of the toolbox (<https://www.mathworks.com/products/text-analytics.html> and <https://de.mathworks.com/help/textanalytics/>).

To install toolboxes in an existing MATLAB installation, provided your license permits it, you can either rerun the MathWorks Installer to add the missing products without reinstalling MATLAB or use the Add-On Explorer within MATLAB by clicking the Add-Ons icon on the left sidebar of the desktop (cf. also [https://www.mathworks.com/help/matlab/matlab\\_env/get-add-ons.html](https://www.mathworks.com/help/matlab/matlab_env/get-add-ons.html)).

You can verify whether you have already installed a toolbox like the Text Analytics Toolbox by typing a command like `ver textanalytics`. If this toolbox is installed, you will then see information about the Text Analytics Toolbox in the Command Window, including its version.

### 2.1.6 Using MATLAB Online™

In addition to installing MATLAB on your computer, you can also access MATLAB directly from your web browser through MATLAB Online™. This feature allows you to run MATLAB code, access files from MATLAB Drive and use many of the same functionalities as the desktop version. If your university provides access to MATLAB Online, you can use it by logging in with your institution's email address.

More information on MATLAB Online can be found at: <https://www.mathworks.com/products/matlab-online.html>.

MATLAB Online is available in two versions:

- **Basic version:** Free for all users with 20 hours of use per month and access to 10 common products.
- **Full version:** Available to users with a campus-wide or institute-wide license.

## 2.2 Starting and closing MATLAB®

There are various ways to start MATLAB. On Windows platforms, you can launch MATLAB by selecting the MATLAB icon, calling it from the Windows system command line or opening a file associated with MATLAB. Another option is to start MATLAB from the MATLAB command prompt or by selecting the MATLAB executable from the Windows Explorer tool. When MATLAB is started, it automatically adds the `userpath` folder to the search path and reopens any desktop components from the previous session. The default `startup` folder is either the last working folder from the previous session or the custom path specified in the preferences, and this can be adjusted for future sessions. For more detailed instructions, you can refer to the MathWorks website: [https://www.mathworks.com/help/matlab/matlab\\_env/start-matlab-on-windows-platforms.html](https://www.mathworks.com/help/matlab/matlab_env/start-matlab-on-windows-platforms.html).

To start MATLAB on Linux platforms, navigate to the `bin` folder where MATLAB is installed and type the command `matlab` at the operating system prompt. If symbolic links were set up during installation, you can type `matlab` from the folder you specified. Once started, MATLAB opens the desktop, restoring any components that were open during the last session, and automatically adds the `userpath` folder to the search path. If the `DISPLAY` environment variable is not set or invalid, the desktop will not display. For more detailed instructions, visit [https://www.mathworks.com/help/matlab/matlab\\_env/start-matlab-on-linux-platforms.html](https://www.mathworks.com/help/matlab/matlab_env/start-matlab-on-linux-platforms.html).

To start MATLAB on macOS, you can either launch it from the Applications folder or from the Terminal window. When started, MATLAB adds the `userpath` folder to the search path and reopens any components from the previous session. The `startup` folder is either the last working folder or a custom path set in preferences. If MATLAB encounters issues with system components like Java, it automatically runs diagnostics and suggests solutions. To start from the Terminal, navigate to the `matlabroot/bin` folder and type `./matlab`. For Apple Silicon, ensure the Java Runtime Environment is properly installed. For more detailed instructions, visit

[https://www.mathworks.com/help/matlab/matlab\\_env/start-matlab-on-macintosh-platforms.html](https://www.mathworks.com/help/matlab/matlab_env/start-matlab-on-macintosh-platforms.html).

To exit MATLAB, you can click the close button on the MATLAB desktop or click the MATLAB symbol on the left side of the desktop title bar and select **Close**. Alternatively, you can type “quit” or “exit” at the command prompt in MATLAB. When closing, MATLAB may prompt you to confirm the exit and save any unsaved files. If you want a confirmation dialog before exiting, you can enable this option in the preferences under MATLAB > General > Confirmation Dialogs. For more detailed instructions, visit [https://www.mathworks.com/help/matlab/matlab\\_env/exit-matlab.html](https://www.mathworks.com/help/matlab/matlab_env/exit-matlab.html).

## 2.3 Starting MATLAB® from Python® and using other external language interfaces

In this introductory guide, the focus is primarily on MATLAB’s core functionalities. However, it is worth mentioning that you can also integrate MATLAB with external programming languages and systems using MATLAB Engine APIs. These APIs allow you to run MATLAB commands directly from within another programming environment without the need to open a full MATLAB session. For example, MATLAB Engine APIs are available for several programming languages, including C/C++, Fortran, Java and Python, cf. the MathWorks documentation under the following links:

*Integrate MATLAB with external programming languages and systems*

[https://www.mathworks.com/help/matlab/matlab\\_external/integrate-matlab-with-external-programming-languages-and-systems.html](https://www.mathworks.com/help/matlab/matlab_external/integrate-matlab-with-external-programming-languages-and-systems.html)

*External language interfaces*

<https://www.mathworks.com/help/matlab/external-language-interfaces.html>

This means that you can write Python programmes that interact with MATLAB and you can start MATLAB and execute MATLAB functions directly from your Python code. To get started, you would first need to install the corresponding MATLAB Engine API for Python. It provides a Python package, called `matlab`, that lets you call MATLAB functions from within Python. Once installed, you can call the engine in any of your current or future Python sessions. The installation process is straightforward and can be done directly from MATLAB. More details on how to install and get started with the MATLAB Engine API for Python can be found on the MathWorks website, cf.

*Install MATLAB Engine API for Python*

[https://www.mathworks.com/help/matlab/matlab\\_external/install-the-matlab-engine-for-python.html](https://www.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html)

*Get started with MATLAB Engine API for Python*

[https://www.mathworks.com/help/matlab/matlab\\_external/get-started-with-matlab-engine-for-python.html](https://www.mathworks.com/help/matlab/matlab_external/get-started-with-matlab-engine-for-python.html)

*Call MATLAB from Python*

<https://www.mathworks.com/help/matlab/matlab-engine-for-python.html>

Since this guide is not focused on integration with external languages, this section has provided only a brief overview of this topic to serve as a stepping stone for those interested in using MATLAB alongside other programming environments.

## 2.4 Overview of MATLAB® documentation and support resources

Fortunately, for those venturing into the world of MATLAB, there is already a rich array of resources available that cater to learners at every level, from the beginner to the specialist. Note that most of the examples found in the available documentation are tailored to students, researchers and professionals in fields such as engineering, mathematics, computer science and related disciplines.

One of the most comprehensive resources available is the extensive MATLAB documentation itself, which can be found on the official MathWorks website at [www.mathworks.com/help](http://www.mathworks.com/help). This documentation offers detailed explanations, examples and references for a wide range of MATLAB functions, toolboxes and features, which makes it an essential resource for both beginners and advanced users. Throughout this book, many sections include links to the relevant parts of the MATLAB documentation for readers to explore further. While the exact URLs might change over time, the documentation on the MathWorks website will always have up-to-date sections where you can find more detailed information and additional examples. You may use the search function on the MathWorks site to look for specific topics. Although the examples from the MathWorks documentation might in many cases not relate specifically to linguistic applications, they will still provide valuable insights into how to use MATLAB's features across various domains. This allows readers to deepen their understanding of MATLAB and find more details on individual functions explained in this book.

Have a look also at the section for videos on the MathWorks website where you can find different types of video resources to enhance your understanding of MATLAB: <https://www.mathworks.com/videos.html>. Use the search bar on the video page and enter keywords related to the topic you are interested in, e.g., “text” or “text analytics” and you will find a variety of videos. You can filter the video resources on the MathWorks website by various criteria, including product type (e.g., searching for videos related to a specific toolbox), video type (e.g., webinar or conference talk), application area (e.g., data analytics) and language of the video (e.g., English or German).

In addition to the documentation on their website, MathWorks offers a range of training options. Their self-paced online courses, for instance, provide individuals the flexibility to learn MATLAB at their own speed, covering everything from basic programming to more specialised topics such as machine learning and data analysis. The MATLAB Onramp tutorials (introductory courses) are free, hands-on sessions that teach the fundamentals of MATLAB in 2-3 hours and include automated assessments and feedback. You can access them through your web browser and receive a shareable progress report and course certificate upon completion. To get started, browse the self-paced online courses at: <https://matlabacademy.mathworks.com/>.

For those who prefer a more structured learning experience, instructor-led courses are available, offering expert guidance. Additionally, on-demand webinars offer in-depth sessions on specific MATLAB topics. MathWorks also hosts various events, where participants can attend conferences and workshops to stay updated on the latest developments in MATLAB applications.

MATLAB is not just a software tool but also a global ecosystem where users can share code, ideas and solutions. The MATLAB community includes forums, blogs, user-contributed files and discussions that are invaluable for anyone looking to deepen their knowledge or troubleshoot problems. This collaborative space ensures that learners and professionals alike can continually improve their understanding and application of MATLAB, cf. <https://www.mathworks.com/matlabcentral/content/communities.html>.

MathWorks also offers a feature that allows GitHub repositories to be opened directly in MATLAB Online to simplify the process of sharing and exploring MATLAB content. Users can create links to specific repositories, highlight key files to open automatically and link to MATLAB Projects for added functionality, cf. the MathWorks documentation at:

#### *Cloud file storage*

<http://www.mathworks.com/help/matlab/cloud-file-storage.html>

#### *Connect File Exchange to GitHub*

<https://www.mathworks.com/matlabcentral/fileexchange/my-file-exchange/github-app-installation-guide>

Moreover, MathWorks provides a huge list of books that explain MATLAB's functions in greater detail, serving as valuable supplementary materials for learners. Some of these books are introductory, providing a foundational understanding of key concepts, while many others are specialised for specific topics, offering in-depth coverage tailored to particular areas of interest or fields of study. Have a look at the MathWorks Academia Books section: <https://www.mathworks.com/academia/books.html>. The books section currently mentions over 2,000 titles for teachers, students and professionals, highlighting the growing use of MathWorks products like MATLAB in research and development.

“MATLAB For dummies”, for instance, by Sizemore and Mueller (2021) serves as an excellent introductory guide and is useful not only for STEM students. There are also textbooks such as Mustafy and Rahman's “Statistics and data analysis for engineers and scientists” (2024) that combine data analysis methods using MATLAB alongside other tools like Microsoft Excel® and SPSS®. Among the few books covering topics particularly relevant for linguists using MATLAB is “Text mining with MATLAB” by Banchs (2021). It offers valuable insights into areas such as mathematical models relevant to text mining and applications such as document categorisation, content analysis, text summarisation and dialogue systems. You might find that Banchs' book presents the material in a rather compact and code-heavy style, which can be challenging for readers who are entirely new to the field of text mining and using MATLAB. For this reason, it is probably best used alongside or as a follow-up to this introductory book, which is intended to provide a more general and more accessible, low-code foundation. Another book that you may find interesting in case you work with similar types of data is Boutsen and Dvorak's “MATLAB primer for speech-language pathology and audiology” (2016) that introduces the fundamentals of using MATLAB for data management in clinical and research settings in the health sciences. Similarly, Ciaburro's “MATLAB for machine learning” (2024) might also be useful for linguists interested in exploring machine learning techniques applied to linguistic data.

While the rich variety of resources available is excellent for users across a broad range of disciplines, there remains a gap when it comes to resources tailored specifically for linguists. Most of the existing MATLAB training and materials focus on engineering, mathematics or data science, and there is a limited availability of introductory examples specifically aimed at lin-

guists among the roughly 2,000 titles from the MathWorks Academia Books section. Therefore, the focus of this book is distinct. It serves as an introduction to MATLAB by drawing on examples from language corpora and providing a clear, practical approach for linguists to use MATLAB in their own research.

## 2.5 Assistance through the MATLAB® AI Chat Playground and MATLAB GPT

The MATLAB® AI Chat Playground is an interactive platform designed to assist users with MATLAB-related tasks, including coding, troubleshooting and learning (cf. <https://www.mathworks.com/matlabcentral/playground>). It is powered by advanced AI technology, specifically built on OpenAI's models, and is optimised to help with a wide range of topics. The playground is particularly focused on technical computing, which encompasses fields such as mathematics, engineering, statistics, machine learning and more. It can also be helpful for exploring related topics and text-related tasks. To use the AI Chat Playground, sign in on the MathWorks site with your account to get started. Once logged in, you can interact with the AI.

The AI can help guide you through coding tasks and will offer suggestions and explanations along the way. You can use the platform to ask questions about MATLAB features and functions. Whether you are looking for advice on how to use certain toolboxes or seeking clarification on mathematical concepts, the AI is trained to provide informative and useful answers. Figure 2 illustrates an example of an interaction with the AI Chat Playground, where the AI is asked to explain how to tokenise a document or text with MATLAB (in this case a small extract from the EcoLexicon English Corpus, a text corpus of contemporary environmental texts available on Sketch Engine, cf. Chapter 4.3).

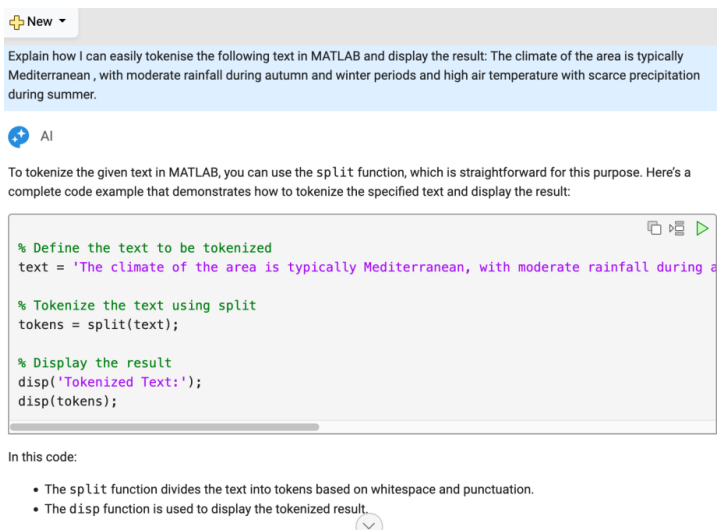


Figure 2: Example of an AI Chat Playground interaction

The response includes a suggested MATLAB code snippet which you can directly test by inserting it into the field on the right side of the interface with a click and running it (Figure 3).

The screenshot shows the MATLAB AI Chat Playground interface. On the left, there is a chat window with the following text:

Explain how I can easily tokenize the following text in MATLAB and display the result: The climate of the area is typically Mediterranean, with moderate rainfall during autumn and winter periods and high air temperature with scarce precipitation during summer.

AI

To tokenize the given text in MATLAB, you can use the `split` function, which is straightforward for this purpose. Here's a complete code example that demonstrates how to tokenize the specified text and display the result:

```
% Define the text to be tokenized
text = 'The climate of the area is typically Mediterranean, with moderate rainfall during autumn and winter periods and high air temperature with scarce precipitation during summer.'

% Tokenize the text using split
tokens = split(text);

% Display the result
disp('Tokenized Text:');
disp(tokens);
```

In this code:

- The `split` function divides the text into tokens based on whitespace and punctuation.
- The `disp` function is used to display the tokenized result.

Enter prompt here

Validate generated output before use. About Discard  
Chats may be shared with affiliates, partners, or service providers in accordance with the MathWorks Privacy Policy.

On the right, the code editor shows the same MATLAB code being executed. The output is displayed as follows:

```
Tokenized Text:
{'The' }
{'climate' }
{'of' }
{'the' }
{'area' }
{'is' }
{'typically' }
{'Mediterranean,' }
{'with' }
{'moderate' }
{'rainfall' }
{'during' }
{'autumn' }
{'and' }
{'winter' }
{'periods' }
{'and' }
```

Figure 3: Test of the suggested code in the execution field of the AI Chat Playground

Please keep in mind that AI-generated code and text may seem accurate, but they are not always guaranteed to be error-free. This platform is designed for experimentation, so be sure to verify any resulting code or suggestions before using them in your projects. Always test and validate the output to ensure it meets your needs.

Alternatively, you can use MATLAB GPT, a version of GPT developed by MathWorks in collaboration with OpenAI™. It is also specifically designed for MATLAB-related tasks based on official MATLAB documentation. After logging into your OpenAI account, you will see an option to start MATLAB GPT directly (Figure 4).

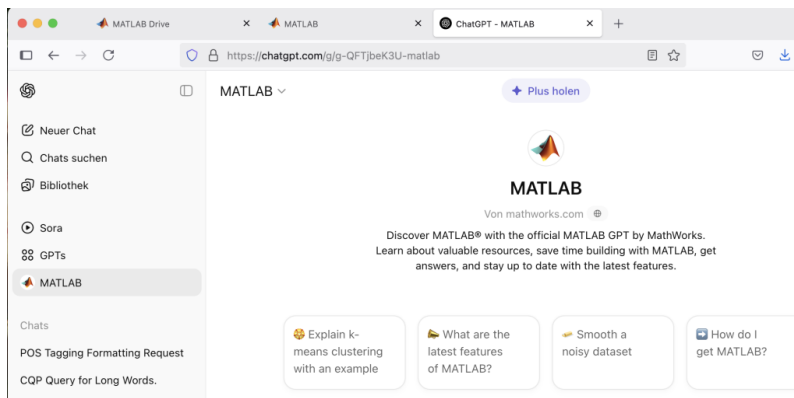


Figure 4: MATLAB GPT

If you are already using ChatGPT in your browser and need MATLAB-specific assistance, MATLAB GPT can be a useful option.

## 2.6 Assistance through MATLAB® Copilot

MATLAB Copilot is a generative AI-powered assistant integrated into the MATLAB desktop environment, introduced in MATLAB R2025a. With MATLAB Copilot (cf. Figure 5), you can ask questions in the Copilot Chat panel and receive helpful responses drawn from MathWorks resources. You can also use it for code predictions, automatic code generation and modification, explanations of code and error messages. Access to MATLAB Copilot is available to licensed users working in MATLAB R2025a or newer with an active internet connection.

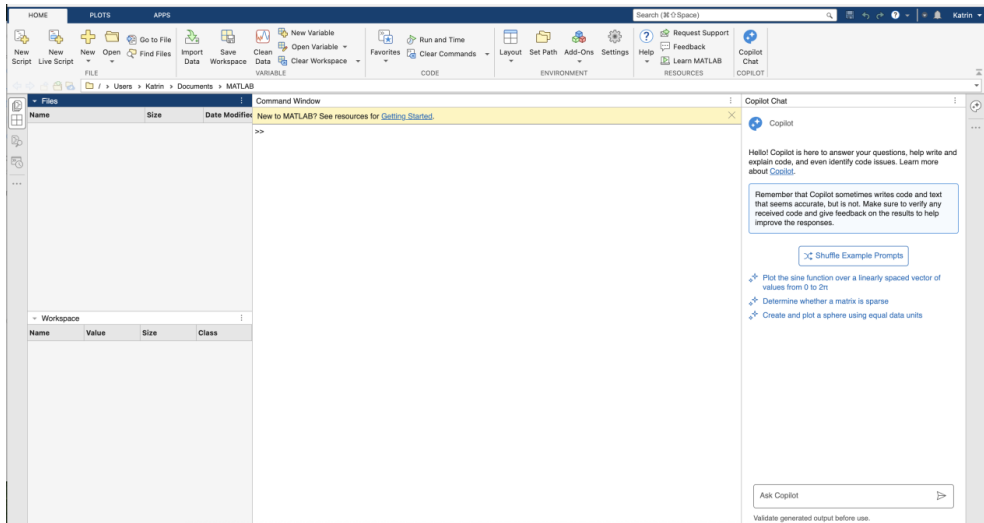


Figure 5: MATLAB Copilot (on the right of MATLAB desktop)

More information and examples of how to use MATLAB Copilot are provided in Chapter 3.9.



# 3 The MATLAB® environment

## 3.1 Overview and MATLAB desktop®

When you start MATLAB, a special window called the MATLAB desktop appears in its default layout. The MATLAB desktop is a window that contains other windows (Figure 6, cf. also [https://www.mathworks.com/help/matlab/learn\\_matlab/desktop.html](https://www.mathworks.com/help/matlab/learn_matlab/desktop.html)).

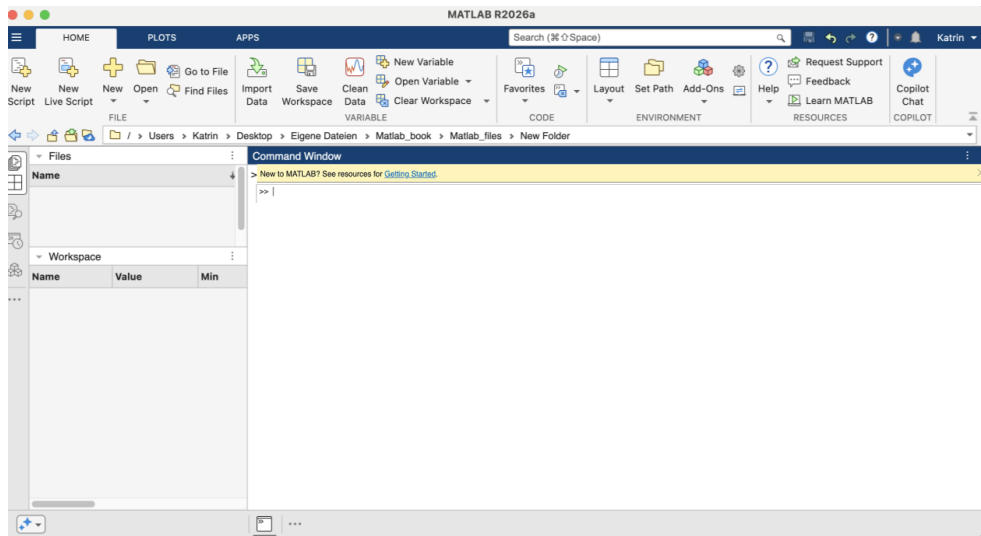


Figure 6: The MATLAB desktop (R2026a)

By default, the MATLAB desktop (which, in from version R2025a onwards, looks a little different from previous versions) includes a toolbar at the top, a current folder toolbar below the toolbar, a Files panel (renamed from Current Folder browser in R2025a) and a Workspace panel (renamed from Workspace browser in R2025a) on the left side and a Command Window on the right side of the desktop. When you open MATLAB on your computer for the first time, you will see a link to resources for getting started in the Command Window that will take you to the MATLAB documentation and collection of online tutorials. You can modify the desktop layout by selecting from a variety of predefined options or by designing a custom layout to suit your preferences. In the **Home** tab, click on **Layout** to choose a preconfigured layout (e.g., **Command Window Only** or **Three Columns**). If you want to revert to the default desktop layout in MATLAB, select **Default**.<sup>2</sup> You can also modify various other details to customise your view, e.g., change MATLAB desktop colours.<sup>3</sup>

<sup>2</sup> cf. [https://www.mathworks.com/help/matlab/matlab\\_env/change-the-desktop-layout.html](https://www.mathworks.com/help/matlab/matlab_env/change-the-desktop-layout.html)

<sup>3</sup> cf. [https://www.mathworks.com/help/matlab/matlab\\_env/beta-new-desktop-for-matlab.html](https://www.mathworks.com/help/matlab/matlab_env/beta-new-desktop-for-matlab.html)

## 3.2 The MATLAB® toolstrip

The MATLAB toolstrip is a user interface component designed to provide a convenient way to access various functionalities and features within MATLAB. It provides an easy way to use a variety of tools and commands, allowing users to perform tasks without needing to type out specific command syntax (Figure 7).



Figure 7: The MATLAB toolstrip

The toolstrip presents commands in a visually intuitive manner, allowing users to perform tasks with just a few clicks. It offers quick access to frequently used functions such as creating plots, importing data, creating new variables and running scripts. Additionally, it enables interactive plotting, allowing users to select plot types and modify properties without typing commands. The toolstrip also includes integrated help and documentation, enhancing the learning experience for new users. For example, when creating a plot, users can simply click on the **Plots** tab, select the desired plot type and then customise the plot with options available in the toolstrip, such as titles, labels and legends. This interactive approach streamlines the workflow, especially for those who prefer a graphical interface over command-line inputs.

## 3.3 The Files panel

One of the major tools within or accessible from the desktop of MATLAB’s interactive environment is the Files panel (the current folder browser). This panel on the left side provides an intuitive interface for accessing your files. When you first launch MATLAB, the Files panel is typically displayed by default in the desktop environment. Before you start working, it is important to set your current working directory, the “current folder” (cf. also <https://de.mathworks.com/help/matlab/ref/filespanel.html> and [https://mathworks.com/help/matlab/matlab\\_env/files-and-folders-that-matlab-accesses.html](https://mathworks.com/help/matlab/matlab_env/files-and-folders-that-matlab-accesses.html)) To do this, use the `cd` function or simply go to the Files toolbar underneath the toolstrip and click on the folder path or on the small “Browse for Folder” icon to choose the folder where you will store and access your files (Figure 8).

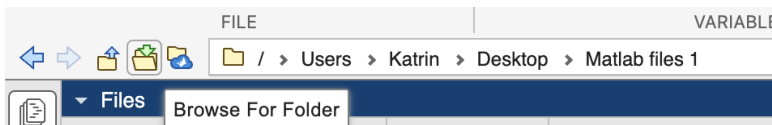


Figure 8: Selecting your current working directory

If the Files panel is not visible, you can open it from the **Home** tab by clicking on **Layout**, then ensuring the **Files** option is selected under the **Show** section (Figure 9).<sup>4</sup>

<sup>4</sup> cf. also [https://www.mathworks.com/help/matlab/matlab\\_env/manage-files-and-folders.html](https://www.mathworks.com/help/matlab/matlab_env/manage-files-and-folders.html)

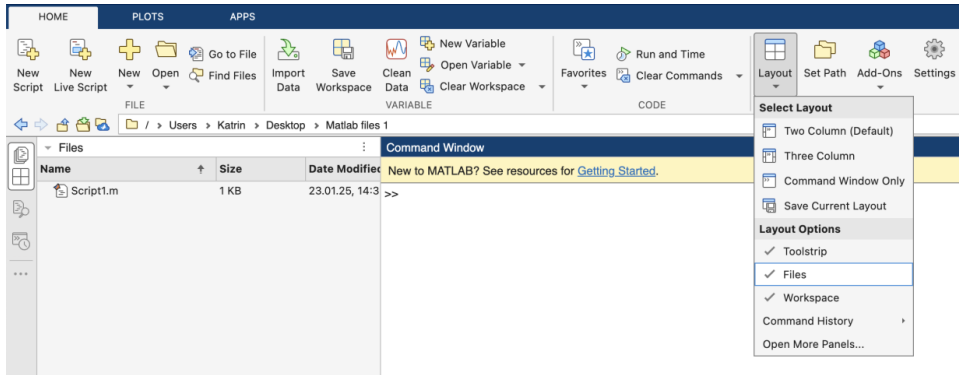


Figure 9: Accessing the Files panel

In the Files panel, you can view the contents of your current working directory, create new files or folders, open, move or rename existing files and folders and switch directories to organise your work. Essentially, it functions like a traditional file explorer, enabling easy navigation and file operations. For example, to rename a file or folder, right-click the item in the Files panel and select **Rename**, keeping in mind that your file names must start with a letter and can include letters, digits or underscores (Figure 10).

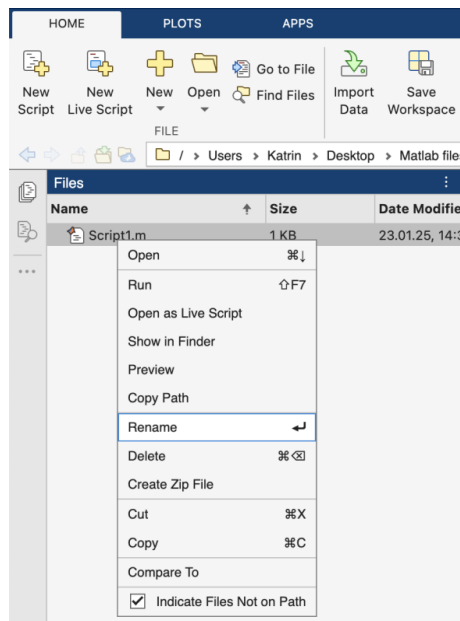


Figure 10: Renaming a folder in the Files panel

To create a new item in the Files panel, such as a folder, right-click and select the appropriate option from the dropdown menu (Figure 11).

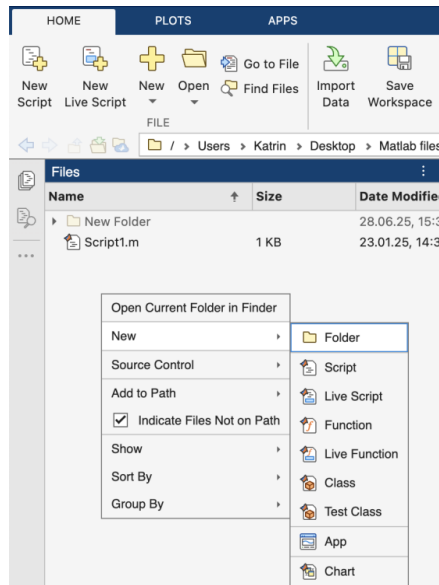
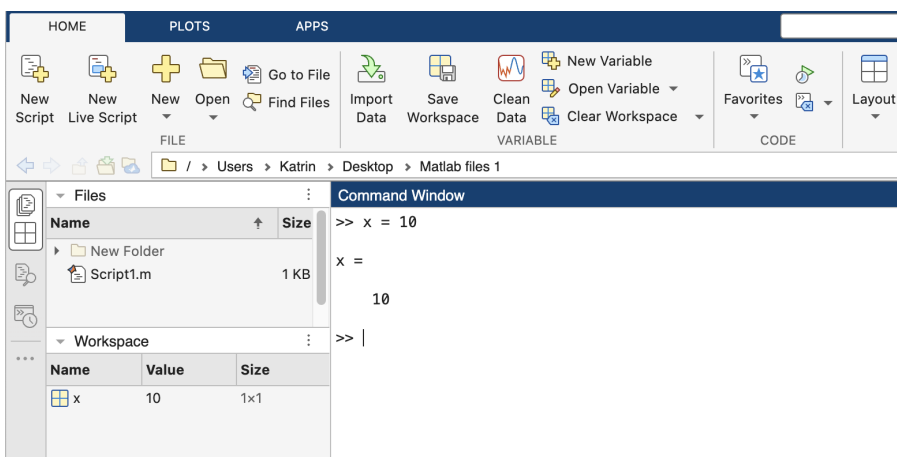


Figure 11: Creating a new folder in the Files panel

### 3.4 The Command Window

The Command Window on the right side in the default layout is used for executing commands interactively by entering them in the command line, indicated by the prompt (`>>`) (cf. <https://www.mathworks.com/help/matlab/ref/commandwindow.html>).

For instance, to create a variable via the MATLAB Command Window, simply type the variable name followed by the assignment operator (`=`) and the value you want to assign. For example, to create a variable `x` and assign it the value 10, you would type: `x = 10` (cf. Figure 12).

Figure 12: Creating a new variable `x` in the MATLAB Command Window

When you type `x = 10` in the MATLAB Command Window, several things happen: MATLAB creates a new variable named `x` in the current workspace and assigns the value `10` to the variable `x`. Now, whenever you refer to `x` in your MATLAB session, it will represent the value `10`. The variable `x` is stored in the workspace (the memory space MATLAB uses to keep track of all the variables and their values you define during a session). In the default settings, MATLAB will display the result of an assignment.

To suppress the display of output, end statements with a semicolon, for example, `x = 10;` and it will just return to the command prompt (`>>`), indicating the assignment was successful. However, if you then type `x` by itself as a command and press Enter, MATLAB will display the value of `x`, which is now `10`.

There is also the `disp` function that can be used to display the value of a variable without printing the variable name – the command `disp(x)` regardless of whether or not it is followed by a semicolon will also display the value of `x` in the Command Window.

In the Command Window you can also perform arithmetic operations and store the results in variables. You can, for instance, create a variable, say `a`, and assign it the result of `5 + 3`. The value of `a` will then be `8` (cf. Figure 13).

Command Window	Workspace		
<code>&gt;&gt; a = 5 + 3</code>	<b>Name</b>	<b>Value</b>	<b>Size</b>
<code>a =</code>	<code>a</code>	8	1×1
<code>8</code>	<code>x</code>	10	1×1
<code>&gt;&gt;</code>			

Figure 13: Creating a variable `a` in the MATLAB Command Window

If you do not specify an output variable, MATLAB uses the variable `ans` to store the results of your calculation (Figure 14). The value of `ans` changes with every command that returns an output value that is not assigned to a variable.

Command Window	Workspace	
<code>&gt;&gt; 10 + 10</code>	<b>Name</b>	<b>Value</b>
<code>ans =</code>	<code>a</code>	8
<code>20</code>	<code>ans</code>	20
<code>&gt;&gt;</code>	<code>x</code>	10

Figure 14: Result of a simple calculation stored in the `ans` variable

You can also, for instance, create a string variable (Chapter 5.2.2) via the Command Window by enclosing the desired text in double quotes (" "). Here is an example:

```
str = "Hello, World!";
```

This creates a variable `str` and assigns it the string "Hello, World!" (Figure 15). Note that in MATLAB, standard syntax highlighting colours help make commands and scripts more readable and easier to debug. By default, for instance, strings appear in purple.

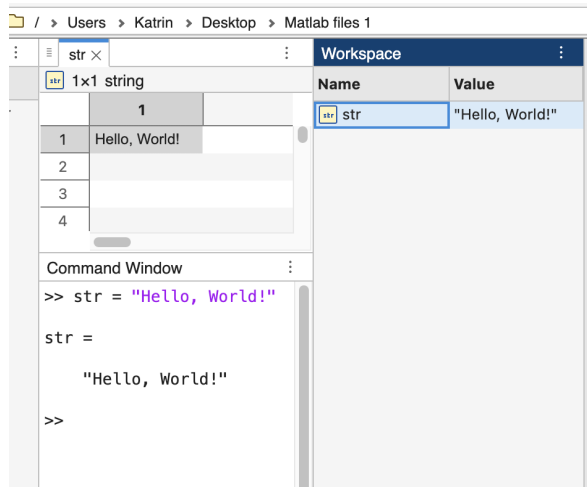


Figure 15: Creating a string variable in the Command Window

MATLAB includes a wide range of predefined callable routines. The MATLAB Function Reference (<https://www.mathworks.com/help/matlab/referencelist.html?type=function>) lists many such routines, including callable functions, which accept inputs, may return outputs and are called with parentheses, as well as command-style operations, which are typically entered without parentheses and are used primarily for their side effects, meaning they perform actions that change MATLAB's environment (for example, modifying variables or the display). There is, for instance, the command `clc`. When you enter it into the Command Window, it clears all the text that you have previously entered into the Command Window, resulting in a clear screen. After running `clc`, you can still recall commands from the Command History (cf. Chapter 3.5). If you use the command `clear`, this removes all variables from the workspace. Another example is the command `doc`. This can be used to open the MathWorks Online documentation or the MATLAB Help browser for a specific topic. The command `doc sum` in MATLAB, for instance, opens the documentation for the `sum` function, while the command `doc script` opens some information on scripts in MATLAB (Figure 16).

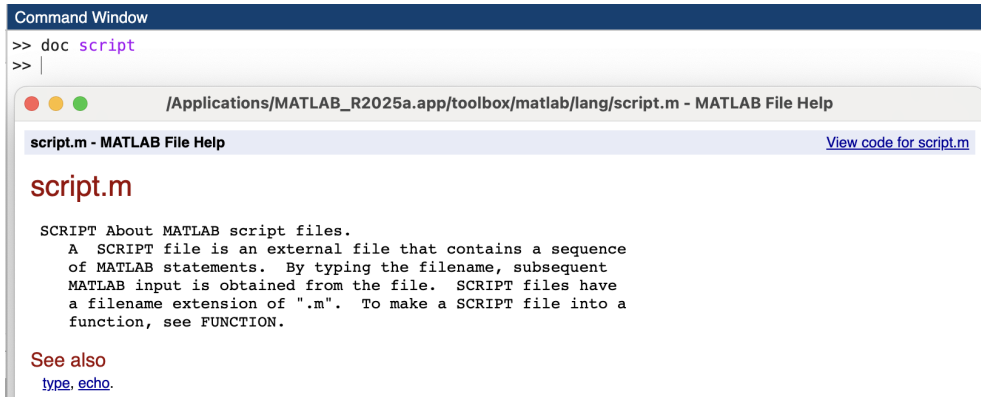


Figure 16: MATLAB Help Browser, accessed using the `doc` command

When working with commands, MATLAB offers automatic suggestions for variables and functions as you type in the Command Window. This feature helps to enhance coding efficiency by providing context-sensitive suggestions based on the current workspace variables and the available functions. As you begin typing a variable or function name, MATLAB will display a list of possible completions. You can press the Tab key to complete partially typed text automatically. Alternatively, you can select relevant items from the drop-down menu, which also helps to save time and effort by minimising the amount of typing required. Figure 17 shows an example where possible variable names from the workspace are suggested to complete a command for creating a box plot.

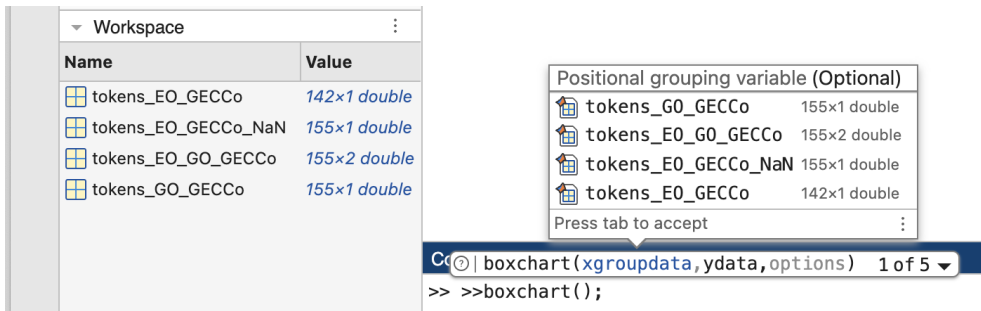


Figure 17: Suggested variable names from workspace to complete new command

Figure 18 demonstrates a scenario where we want to modify a box plot, and the drop-down menu displays available options, such as adding notches (see also Chapter 7.10).

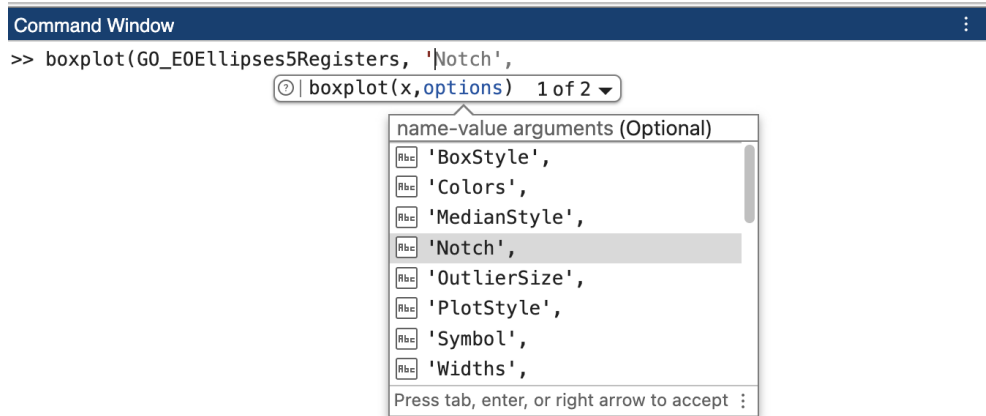


Figure 18: Suggested modifications for a box plot during command typing

In the Command Window, you can use the keyboard shortcut Ctrl + F (or Cmd + F on macOS) to open the search functionality. This allows you to search for specific text within the Command Window output (Figure 19).

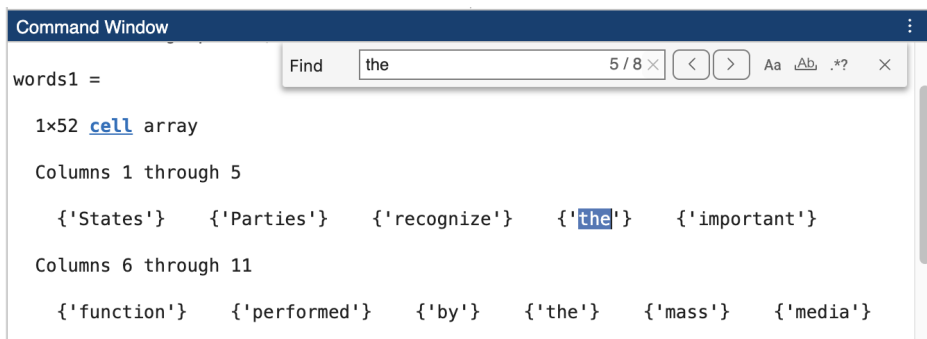


Figure 19: Using the search functionality in the Command Window

We will see more examples of commands and explore different data types in the following chapters.

### 3.5 The Command History

If you want to reuse or modify any of your previous commands, you can press the Up arrow key ↑. This will bring up the **Command History window**, where you can see a log of all the commands you have entered in the Command Window during the current session. To rerun a previous statement, you can start by pressing the Up arrow key repeatedly until the command you wish to reuse appears at the prompt (Figure 20).

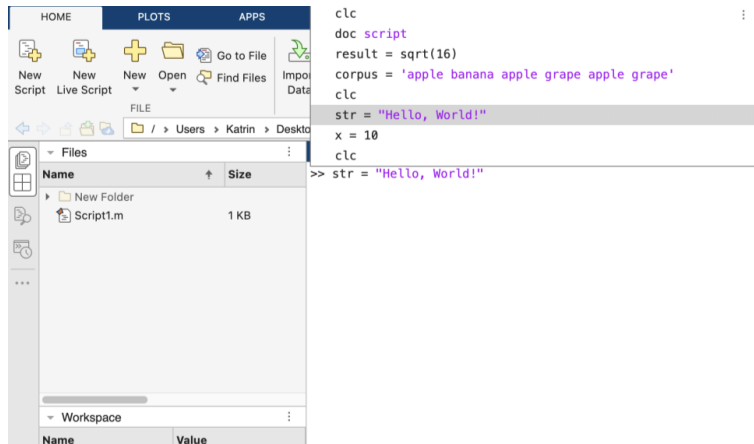


Figure 20: Selecting a previous command from the Command History, opened in a pop-up window

Once it is displayed, press Enter to execute it. To view a specific statement, type any part of the respective statement at the prompt and then press the Up arrow key. Alternatively, you can open the Command History panel by entering `commandhistory` in the Command Window (Figure 21). Then double-click on any individual entry to rerun it or click on a specific entry and press Enter to execute it.<sup>5</sup>

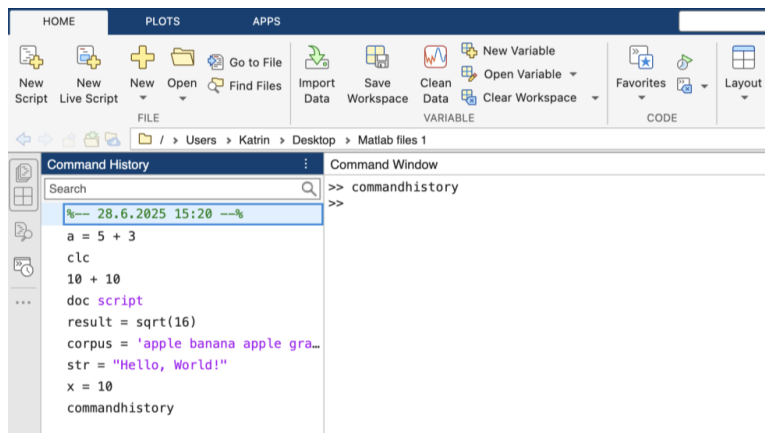


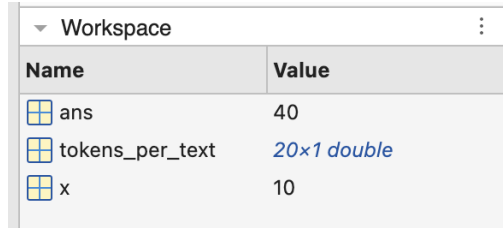
Figure 21: Opening the Command History panel via a command

If you would like to run multiple commands at once, hold down the Shift key while pressing the Up arrow key to extend your selection and then press Enter to execute all the selected commands together. If you want to modify a previous command to suit a new requirement (e.g., adjusting a calculation or changing a variable assignment such as assigning your variable `str` now the string "Hi, World!" or creating a new variable `str1`), you can reuse parts of your previous command by moving the cursor to it when it appears at the prompt, making your changes and then executing it.

<sup>5</sup> cf. also <https://www.mathworks.com/help/matlab/ref/commandhistory.html>

### 3.6 The Workspace panel

The Workspace panel on the left side of the default layout of the MATLAB desktop shows all variables currently in use (Figure 22).<sup>6</sup>



Workspace	
Name	Value
ans	40
tokens_per_text	20x1 double
x	10

Figure 22: The Workspace panel

The variables in your workspace can be easily duplicated, renamed or deleted there. For instance, in order to duplicate a variable, right-click on it and then select **Duplicate**. A copy of the selected variable will be created. Let us say you have a variable `word_freq` containing word frequencies and you want to experiment with it, you can duplicate it to keep the original intact. Right-click on `word_freq` in the workspace, select **Duplicate** (Figure 23), and a copy of `word_freq` will be created, typically named `word_freqCopy`. You can then modify values in `word_freqCopy` without affecting the original `word_freq`.

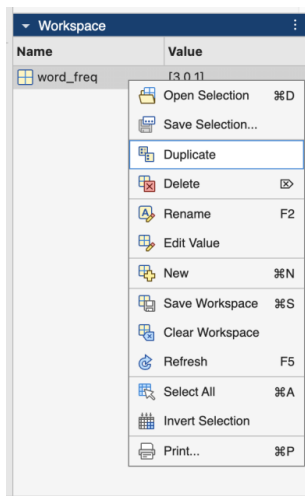


Figure 23: Duplicating a variable in the workspace

For each variable in the workspace, the Workspace panel can also display relevant statistical values such as the minimum, maximum and mean values or the standard deviation. MATLAB also updates these values automatically if you modify your variables. To display these statistics, right-click the header of any column in the Workspace panel and select the statistics you want to display (Figure 24).

<sup>6</sup> cf. also <https://www.mathworks.com/help/matlab/ref/workspacepanel.html>

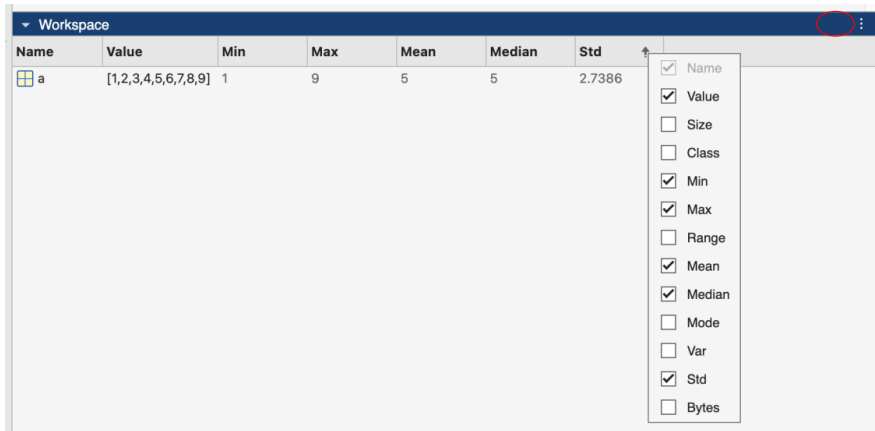


Figure 24: Selecting statistics to display in the Workspace panel

Calculating statistics on large arrays might impact performance. To improve efficiency, you can reduce the statistics shown by unchecking irrelevant columns or by excluding large arrays from statistical analysis. This can be done through the **Workspace Settings**, which are accessible by selecting **Settings** on the **Home** tab in the **Environment** section (Figure 25).

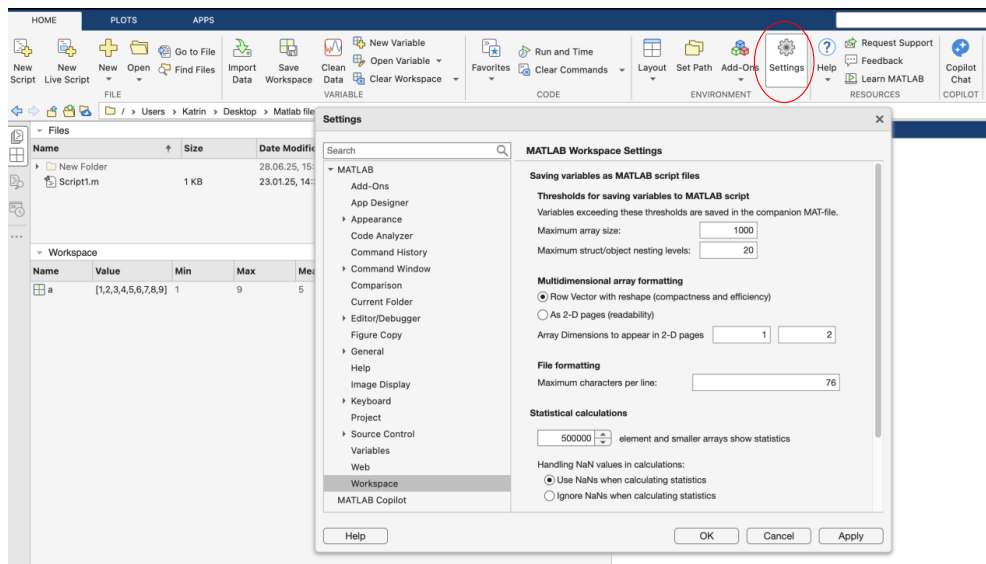


Figure 25: Adapting workspace preferences

When you click on the **Workspace Actions** button with the three dots arranged vertically in the upper right corner of your Workspace panel (cf. Figure 24 above), you will see options to save all current workspace variables to a MAT-File (a binary MATLAB file) or to delete all variables from your workspace. To save your workspace select **Save Workspace** and to delete your variables click on **Clear Workspace** from the options in the list. If you want to save a specific variable or several specific variables, you can select them in the Workspace panel, right-click on the selection and choose **Save Selection** from the context menu.

Alternatively, if you want to save all current workspace variables, you can click on **Save Workspace** in the **Variable** section on the **Home** tab and you can also clear your workspace by clicking on the respective **Clear Workspace** option there (Figure 26). This allows you to either preserve your current variables for future sessions or remove them from the workspace entirely.

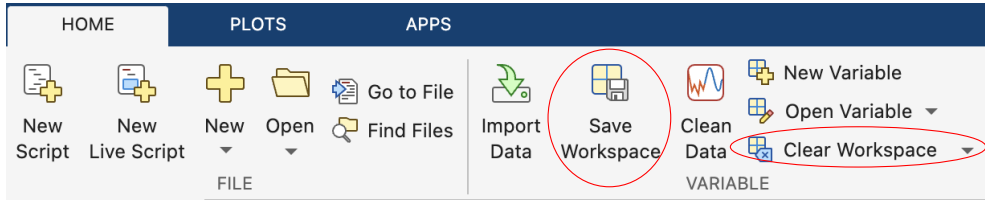


Figure 26: Saving or clearing the workspace via the respective icons on the Home tab

To load a previously saved workspace with variables, double-click on the respective MAT-file in the Files panel (Figure 27).

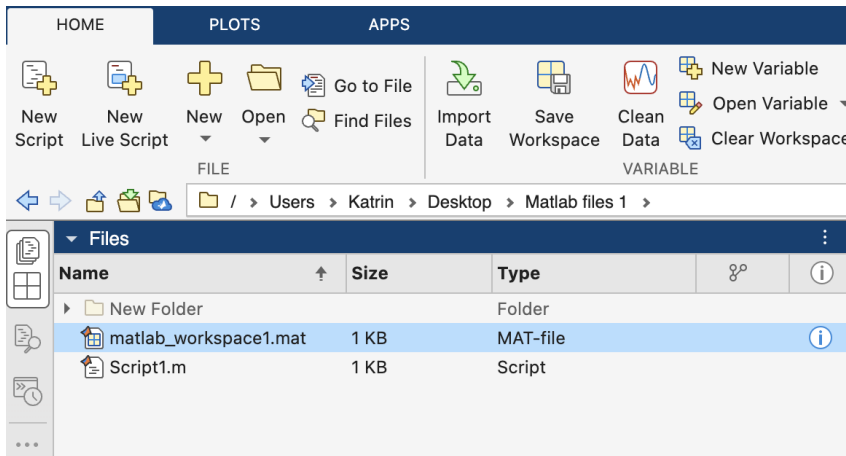


Figure 27: Loading a saved workspace from the Files panel

Keep in mind that when loading a saved workspace, variables that already exist in the current workspace may be overwritten.

### 3.7 The Variables editor

You can view and edit variables in the **Variables editor**. When you have stored variables in your Workspace panel, you can double-click on a variable name there and the Variables editor will open for the selected variable (Figure 28). Alternatively, you can use the `openvar` function, e.g., in the command `openvar("Words_per_text")`.

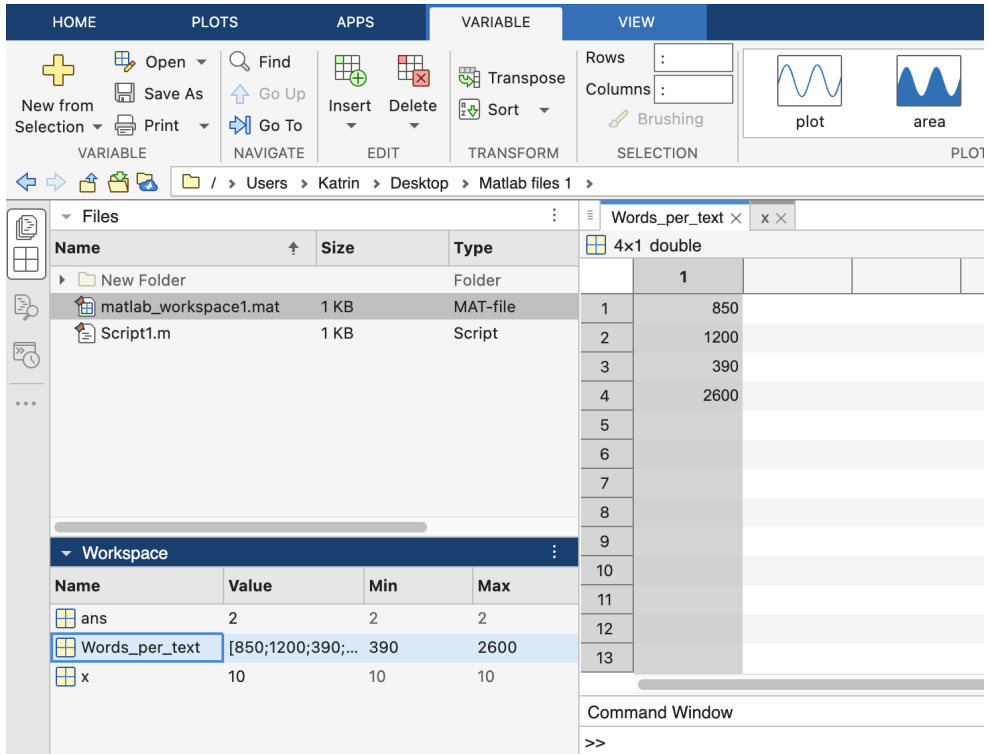


Figure 28: Variables editor

To modify the way variables are shown in the Variables editor, navigate to the **View** tab and choose a display format (Figure 29).

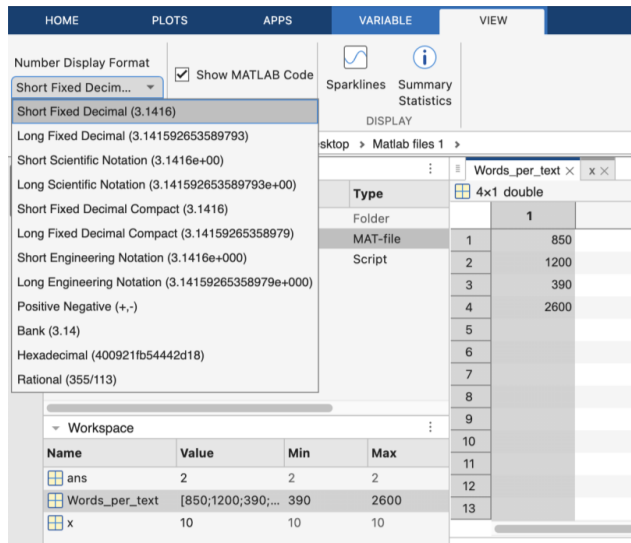


Figure 29: Selecting display format of variables

In Chapter 5.4, we will explore in greater detail how to create, edit and manage variables using the Variables editor.

### 3.8 MATLAB® Apps

MATLAB offers a collection of built-in applications that help users carry out tasks such as data analysis or visualisation. Various apps are also provided as part of specific toolboxes. For example, the Curve Fitting App is included in the Curve Fitting Toolbox. It provides an interactive, low-code environment for fitting curves and surfaces to data using a range of regression, interpolation and smoothing techniques. This app also provides goodness-of-fit statistics, confidence intervals, and the ability to export fits for further analysis. Another app, the Distribution Fitter app (cf. Chapter 9.2), comes with the Statistics and Machine Learning Toolbox. This app makes it possible to fit a wide variety of probability distributions to data from the MATLAB workspace. It can display fitted distributions over histograms and produce plots such as probability density functions and cumulative distribution functions.

You can access the available applications via the **Apps** tab on the MATLAB toolstrip, where all installed apps can be viewed and launched. You will see a selection of commonly used apps in the toolstrip, and you can access all available apps through a dropdown menu (Figure 30).

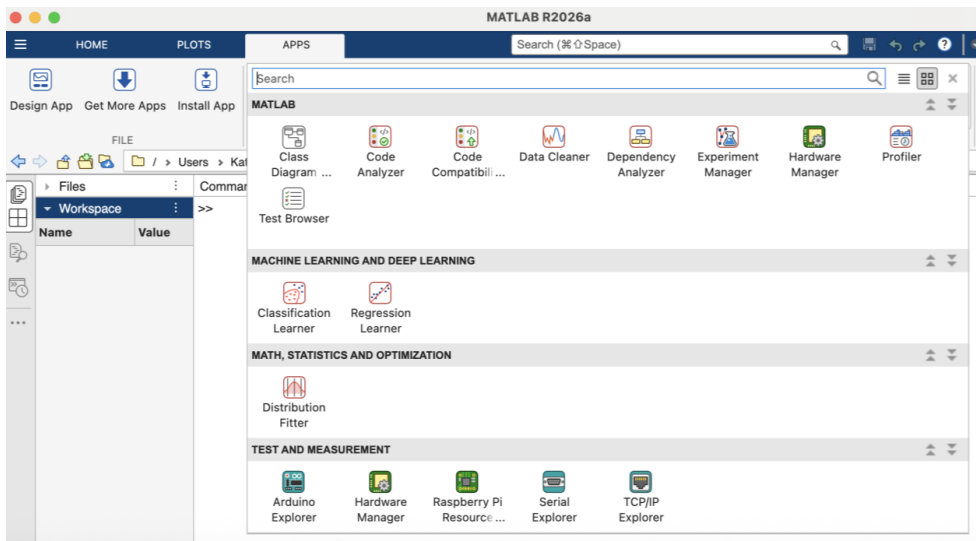


Figure 30: Displaying available applications via the Apps tab

In addition, MATLAB provides the App Designer, which allows users to create their own custom applications with interactive graphical interfaces to automate specific tasks. You can also find and download more apps through the Add-Ons feature, which is accessible from the Apps tab by clicking on **Get More Apps** or from the Home tab by clicking the **Add-Ons** button. This will open the Add-On Explorer, where you can browse a wide variety of apps, including those developed by MathWorks and contributions from the user community. The app listing provides various details such as descriptions, screenshots and user ratings. When you are ready to install an app from this selection, click the **Install** button. The app will then be added to your MATLAB environment and will appear in the Apps tab.

### 3.9 MATLAB® Copilot

As briefly introduced earlier in Chapter 2.6, MATLAB Copilot is a generative AI assistant built into the MATLAB desktop environment, starting with MATLAB R2025a. Access to MATLAB Copilot is available to licensed users through their MathWorks Account. You can interact with Copilot through the Copilot Chat panel, where you can have a natural-language conversation to help solve problems using MATLAB. To open the panel, click the Copilot Chat button on the MATLAB toolstrip. The panel will appear to the right of the Command Window in the MATLAB desktop. MATLAB Copilot functions as an interactive assistant: you can ask it questions, and it responds with conversational explanations and relevant code, drawing from MathWorks documentation and code examples. You can also try out any suggested code directly by clicking the Run button within the chat, which allows you to test suggested solutions quickly without leaving the Copilot Chat panel (Figure 31).

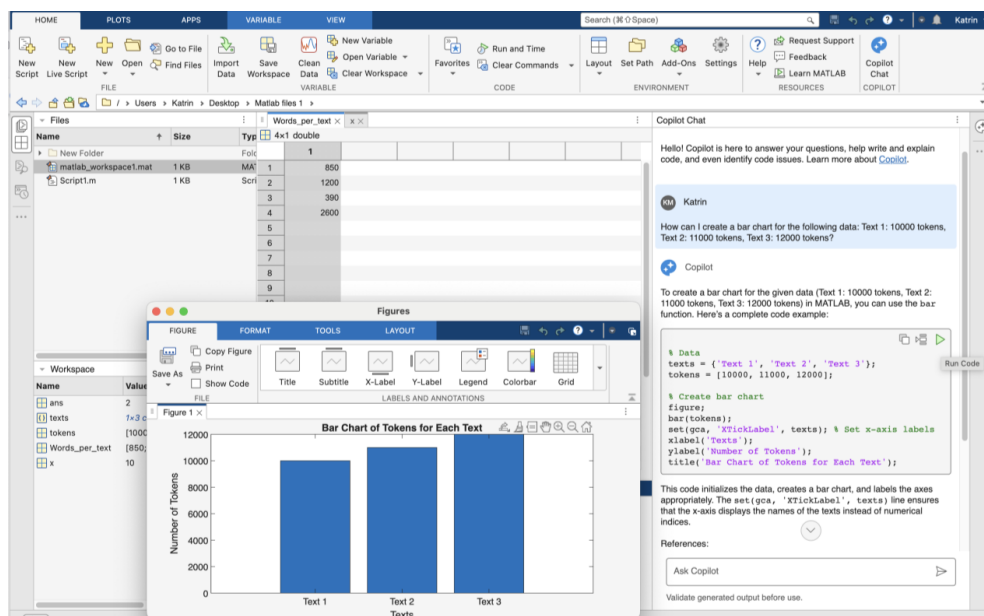


Figure 31: Interacting with Copilot through Copilot Chat panel

Copilot can help you learn by doing – it can explain your code, clarify errors and add helpful comments as you work. Copilot actions are also directly accessible from the menus in the MATLAB environment. When you are writing a script in the Editor or entering commands in the Command Window, you can access Copilot actions through the context menu which can be opened by right-clicking. Figure 32 shows an example of accessing Copilot actions directly from the Editor.

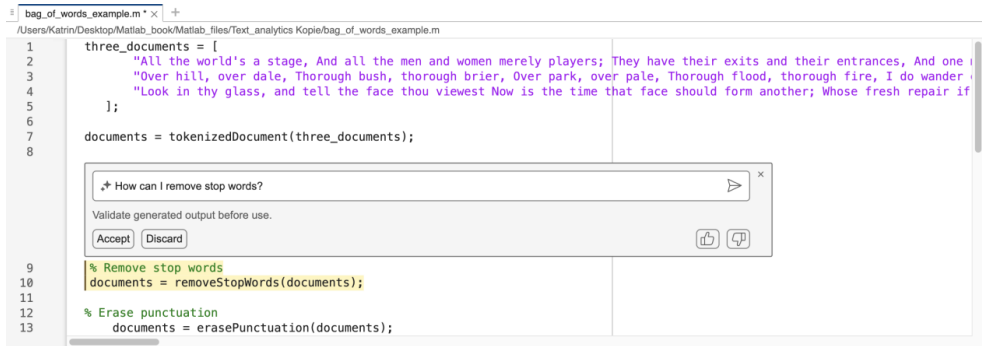
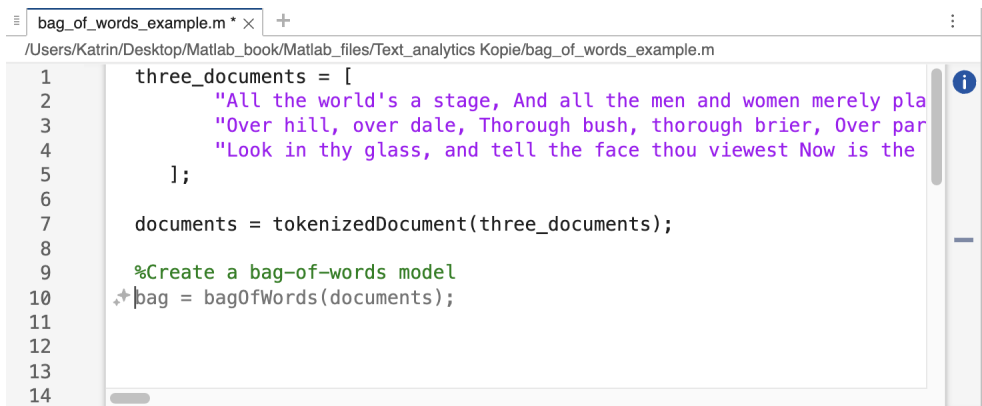


Figure 32: Accessing Copilot actions from the Editor

As you type in the MATLAB Editor or Live Editor, MATLAB Copilot provides code predictions based on your input. This autocompletion feature is enabled by default. You can generate specific code suggestions by pressing the Enter key after typing a partial command or comment (Figure 33).

Figure 33: Code suggestions provided by MATLAB Copilot<sup>7</sup>

The suggestions are context-sensitive and depend, for instance, on the current workspace variables and the files available in the current folder. If you want to adjust the behaviour of auto-completions, you can do so through the MATLAB Copilot settings. There you can enable, disable or pause auto-completions. To access these settings, go to the MATLAB toolstrip, click on Settings and select MATLAB Copilot.

For many tasks, MATLAB Copilot can be extremely helpful, and it is definitely worth trying out and learning how to write effective prompts tailored to your needs. As mentioned earlier already with regard to the MATLAB AI Chat Playground, please note that while AI-generated code and text may appear accurate, it is not always guaranteed to be free of errors. It is important to verify any code or suggestions from MATLAB's Copilot before incorporating them into your projects. MATLAB Copilot can, for instance, occasionally suggest incorrect function names, such as `removePunctuation`, when it should be `erasePunctuation`. Additionally, the solu-

<sup>7</sup> In MATLAB, comments start with % and appear in green to show they are just notes for humans.

tions Copilot offers may sometimes have limited practicality for linguistic applications. For certain questions, MATLAB Copilot might answer that it cannot help with these requests. Moreover, it might not always fully understand linguistic terminology or relationships. For example, if you ask it to create sentences starting with a definite article, it might generate examples like “A quick mouse sits,” which actually begins with an indefinite article and combines words in a way that sounds unusual. As a linguist, you might prefer different examples in such a case. Therefore, you should always carefully review the suggested answers and code and be prepared to modify them as needed to fit your specific purposes. However, the suggestions can serve as a good starting point from which to develop and refine your own solutions.

Keep in mind that many tasks in MATLAB can be done without writing any code by simply clicking buttons in the interface. Since MATLAB Copilot is primarily code-based, it may not always be able to guide you through tasks that can easily be accomplished with a few clicks, without needing to write any code. Copilot typically generates MATLAB code as output based on MathWorks general documentation and examples. When you use Copilot as a beginner, it can in many cases be helpful, but you might see and run Copilot-generated code even if you do not fully understand it yet. In such cases, you can ask Copilot to explain the code or to adapt and modify it to better suit your needs.

This book emphasises a more visual, low-code experience. Whenever possible, you will use MATLAB’s Toolstrip or interactive interfaces and tools which present commands in an intuitive, button-based layout. This allows you to perform many tasks – like creating plots, importing data and defining variables – with just a few clicks, without typing code. This click-driven approach is ideal for beginners with little coding experience. At the same time, MATLAB Copilot complements it by helping you gradually understand the underlying code and bridging the gap between visual interaction and writing code.

### 3.10 MATLAB Drive and MATLAB Online™ Desktop

You can manage your MATLAB files both locally on your computer and in the cloud using MATLAB Drive, which is a cloud-based storage solution from MathWorks. This allows you to store, manage and share your MATLAB files from anywhere with 5GB (free) or 20GB (licensed) storage (cf. Figure 34).<sup>8</sup>

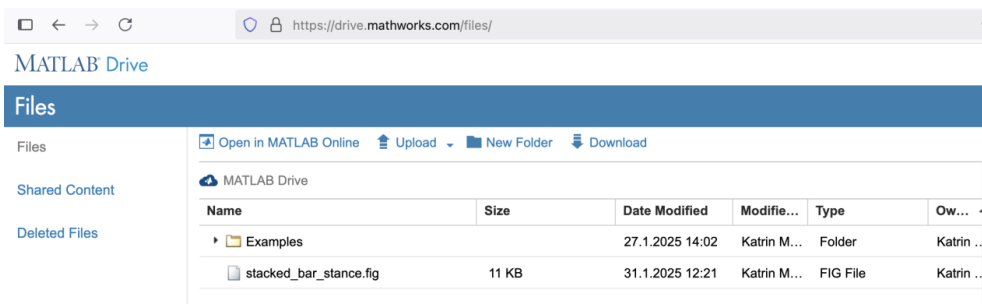


Figure 34: File storage in MATLAB Drive

<sup>8</sup> cf. also <https://www.mathworks.com/help/matlab/cloud-file-storage.html> and <https://www.mathworks.com/products/matlab-drive.html>

You can open your files from MATLAB Drive in MATLAB Online™, which provides access to MATLAB through a standard web browser (Figure 35).<sup>9</sup>

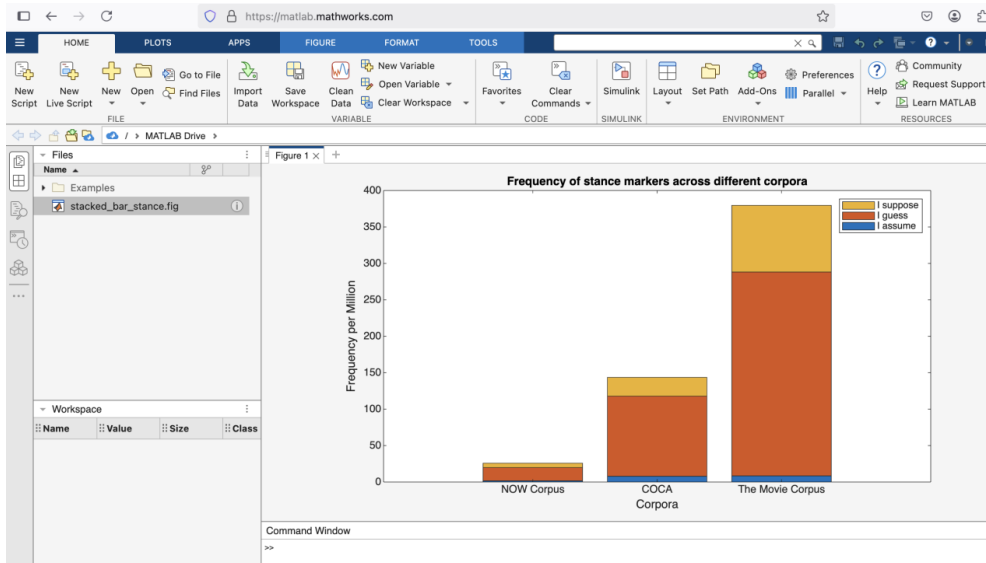


Figure 35: Using MATLAB Online via web browser

It is possible to run MATLAB Online directly in any standard web browser without the need for installations or downloads – perfect for working wherever you are. Additionally, you can collaborate by sharing your MATLAB files with others and choose whether they have view-only or editing privileges. The MATLAB Drive Connector allows you to synchronise files between your online MATLAB Drive and your local desktop MATLAB version (Figure 36).

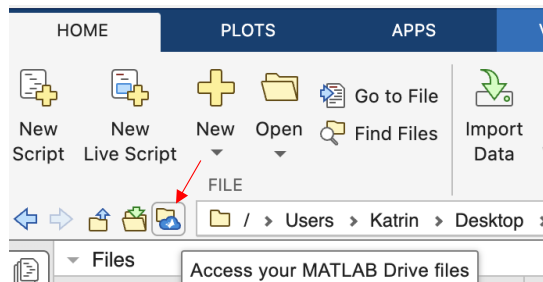


Figure 36: Setting up MATLAB Drive and MATLAB Drive Connector on your computer

It is important to note that certain file management actions are not supported in the MATLAB Online environment and some features may function slightly differently. In this book, the explanations primarily refer to the installed version of MATLAB.

<sup>9</sup> cf. also [https://www.mathworks.com/help/matlab/matlab\\_env/access-files-in-matlab-online.html](https://www.mathworks.com/help/matlab/matlab_env/access-files-in-matlab-online.html)

## 4 Examples of retrieving corpus data and other textual resources

### 4.1 Extracting linguistic data from publicly available corpora and other textual sources

This chapter provides a brief overview of how to access and acquire linguistic data from electronic corpora, explaining what corpora are and highlighting some that are publicly available and that may be particularly useful for researchers and students just starting out in the field. It focuses on introducing several sources of available corpora and corpus interfaces that provide access to large collections of linguistic data for further analysis. These include platforms such as Sketch Engine (Kilgarriff et al. 2014), English-Corpora.org (Davies n.d.) and CQPweb platforms which are hosted by various universities. While this chapter provides an overview of the general process for accessing and extracting data from corpora and other text collections, it is not intended to serve as a comprehensive guide of querying and extracting data from available resources. It offers some examples of how to extract data from selected corpora and other textual resources. For more detailed instructions and further information, users are encouraged to consult the tutorials and documentation available on the respective platforms and for the specific corpora and datasets. The following chapters will also include various examples and practical applications drawn from these resources, illustrating how these resources can be used for different types of linguistic analysis and for generating visualisations with MATLAB.

You might already know that a corpus (plural: corpora) is a collection of texts that are carefully selected and organised based on specific linguistic criteria. Corpora are usually stored in electronic, machine-readable formats, making them accessible for processing and analysis using computational tools. With the increasing availability of large, annotated linguistic corpora, researchers have access to vast amounts of data that can be used for detailed linguistic analysis. Corpora may consist of millions of words, allowing us to examine linguistic patterns on a much larger scale than would be feasible with smaller, manually gathered datasets. Queries can also facilitate the efficient retrieval of specific linguistic structures so that we can quickly extract relevant data. Accessing linguistic corpora can vary depending on the platform and the specific corpus in question. Publicly available corpora are often accessible via online platforms or repositories and may require registration or agreement to terms of use. Many corpus platforms provide web-based interfaces where users can query the data and retrieve specific information, such as word frequencies, collocations or examples of a word in context. Some corpora are made available for direct download, either in raw text form or in more structured formats such as XML, CSV or plain text.

For some tasks, however, you may also wish to retrieve your own data and compile custom datasets rather than relying solely on pre-existing corpora. This process generally requires additional steps of text preprocessing (see Chapter 10.3), since raw textual data is often noisy and not immediately suitable for linguistic analysis. For example, if you extract texts from certain file types such as PDF, you may need to perform cleaning operations, correct errors introduced through Optical Character Recognition (OCR) or remove irrelevant formatting elements. Preprocessing will ensure that the data is consistent and structured in a way that facilitates subsequent analysis with MATLAB or other tools. Some of the examples in this book will also demonstrate how to work with your own texts and datasets in addition to publicly available corpora.

There are many possible ways to collect textual data, ranging from scraping online sources, digitising printed texts or compiling materials from existing digital libraries. One example of a valuable resource for freely available texts is the Oxford Text Archive (OTA), hosted by the Bodleian Libraries at the University of Oxford (<https://ota.bodleian.ox.ac.uk/repository/xmlui/>).

You can use the search function of the website of the Oxford Text Archive to find specific texts available for download. In the advanced search of the OTA website, you can search for specific titles, authors, subjects, date ranges and more (cf. Figure 37).

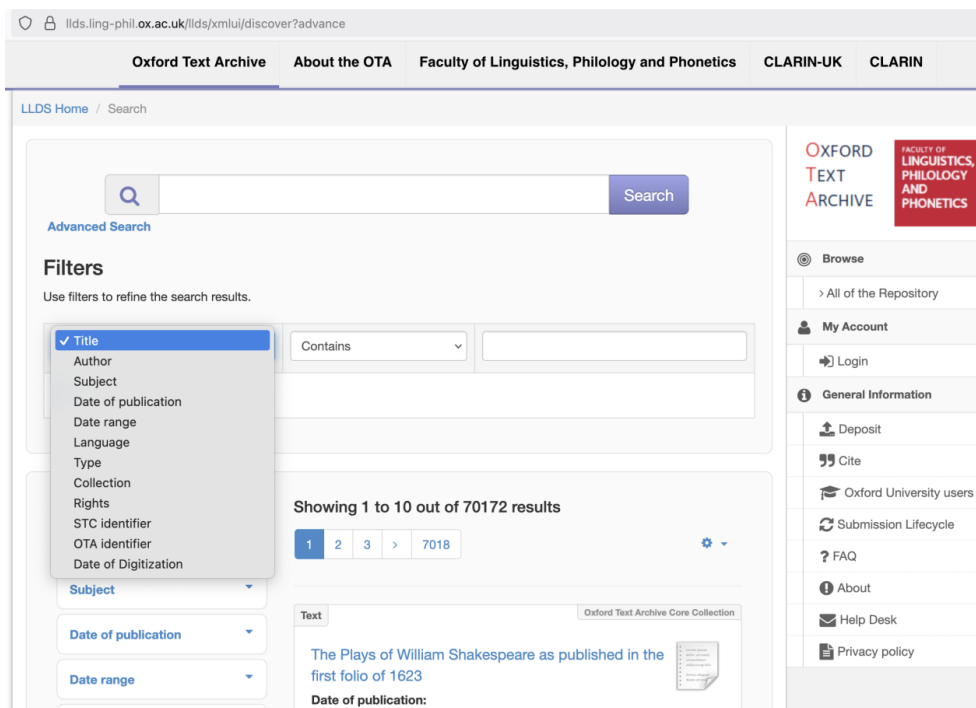


Figure 37: Advanced search at OTA website

Many texts from the OTA are available for download in multiple formats, including EPUB for e-book readers, plain text files with all tags and formatting removed, HTML for web browsers and TEI XML format, preserving the original structure and metadata of the work. Chapter 5.5 demonstrates how to read the contents of a text file – such as a novel downloaded from the Oxford Text Archive – into a MATLAB variable. In Chapters 10.3 and 10.4.4, we will use some of the texts obtained from the OTA for text preprocessing and explore text analytics methods such as training word embeddings.

In addition to the OTA, there are several other well-established repositories that provide access to large collections of textual data. Project Gutenberg (<https://www.gutenberg.org/>), for instance, offers a library of more than 75,000 free e-books, mostly consisting of literary works in the public domain and available in plain text or e-book formats. Another important resource is the Internet Archive (<https://archive.org/>), a non-profit digital library that hosts millions of free texts alongside other media. For linguists working with multilingual or specialised resources,

the CLARIN Virtual Language Observatory (VLO) provides access to hundreds of thousands of language resources (<https://vlo.clarin.eu/>).

## 4.2 Retrieving data from corpora at English-Corpora.org

English-Corpora.org is an online platform developed by Mark Davies that provides access to a variety of English language corpora (Figure 38).

The screenshot shows the English-Corpora.org website. The header includes the site name and navigation links: corpora, PDF guides, videos, related resources, users, my account, upgrade, and help. Below the header, there are tabs for 'Overview: brief | detailed' and a link for 'Expanded help files'. A paragraph of text describes the corpora as widely used for teaching and research. Below this is a table listing various corpora.

Corpus	Download	# words	Dialect	Time period	Genre(s)
News on the Web (NOW)	↓	20.3 billion+	20 countries	2010-yesterday	Web: News
iWeb: The Intelligent Web-based Corpus	↓	14 billion	6 countries	2017	Web
Global Web-Based English (GloWbE)	↓	1.9 billion	20 countries	2012-13	Web (incl blogs)
Wikipedia Corpus	↓	1.9 billion	(Various)	2014	Wikipedia
Coronavirus Corpus	↓	1.5 billion	20 countries	2020-2023	Web: News
Corpus of Contemporary American English (COCA)	↓	1.0 billion	American	1990-2019	Balanced
Corpus of Historical American English (COHA)	↓	475 million	American	1820-2019	Balanced
The TV Corpus	↓	325 million	6 countries	1950-2018	TV shows
The Movie Corpus	↓	200 million	6 countries	1930-2018	Movies
Corpus of American Soap Operas	↓	100 million	American	2001-2012	TV shows

Figure 38: Corpora from English-Corpora.org

These corpora are widely used for linguistic research and language teaching, offering billions of words of data for analysis and providing authentic examples of language in use. Initially, these corpora were developed and hosted at Brigham Young University (BYU) and were commonly referred to as the “BYU Corpora.” They were relocated to English-Corpora.org in 2019 and there is no longer any connection between the corpora and BYU.

English-Corpora.org offers limited free basic access to its corpora (with daily usage limits). Expanded access is available by joining an academic license through a university or by purchasing an individual premium account. Some corpora can also be purchased and downloaded for local use. To get started, visit the platform’s website and create a free account by clicking on the “Register” link. Once registered, you will have access to the corpora and their features.

Among these resources, the British National Corpus (BNC, Davies 2004) is one of the most widely used corpora. It contains a comprehensive collection of British English from the 1980s to 1993, comprising 100 million words drawn from a wide range of registers and text types, including spoken language, newspapers, fiction, academic writing and more. The Corpus of

Contemporary American English (COCA, Davies 2008–) includes 1 billion words spanning from 1990 to 2019, balanced across diverse registers, similar to the BNC (e.g., spoken, fiction, academic, news and others). The Corpus of Historical American English (COHA, Davies 2010), on the other hand, offers a historical perspective and can be used to track changes in American English over nearly two centuries. It comprises 475 million words covering the period from 1820 to 2019. For more recent and globally distributed data, you can turn to the News on the Web (NOW) corpus. Updated regularly, NOW includes more than 20 billion words from news articles published between 2010 and the present, spanning 20 countries. These are just a few examples of the corpora available on English-Corpora.org. All of these corpora share a unified interface that facilitates various search functionalities.

Users can, for instance, perform word and phrase searches to locate specific terms, conduct lemma searches to retrieve all forms of a word and utilise wildcard searches to find words matching particular patterns. Concordance lines offer keyword-in-context (KWIC) displays to examine word usage in various contexts. Collocate searches enable the identification of words that frequently appear near a target word. You can compare language use across specific sections of the corpus, such as different registers or time periods. Visualisation tools present search results as frequency charts or tables and display occurrences and distributions across different sections.

Let us have a look at the following example: We would like to search for frequencies of adjectives ending in the suffix “-less” in the COHA corpus to analyse their occurrences over time. We go to the link to COHA on the English-Corpora.org website, log in and use a query string such as `*less_j` and determine that we want to see the frequency for each “section” of the corpus, in this case, for each decade. `*` is a wildcard, which will match any string of characters before the string “less” and the `_j` tag refers to adjectives in COHA indicating that you are looking specifically for adjectives (excluding, for instance, nouns and verbs that might also end in “-less” such as the noun “devilless” or the verb “bless”).

We will see the results of our query in a table as in Figure 39.

SEARCH		FREQUENCY										CONTEXT										OVERVIEW									
ON CLICK: [CONTEXT] [TRANSLATE (?)] [ENTIRE PAGE] [GOOGLE] [IMAGE] [PRON/VIDEO] [BOOK] [THESAURUS (HELP)]		ALL	1820	1830	1840	1850	1860	1870	1880	1890	1900	1910	1920	1930	1940	1950	1960	1970	1980	1990	2000	2010									
1	★ USELESS	11665	370	496	507	548	607	746	751	717	771	720	626	598	589	525	541	473	451	511	596	522									
2	★ HELPLESS	10043	241	233	329	358	525	719	607	610	771	737	637	710	616	525	487	447	439	395	360	297									
3	★ ENDLESS	9458	126	158	335	303	295	336	358	377	416	445	468	501	486	523	532	596	694	795	897	817									
4	★ RESTLESS	8078	126	326	401	408	448	540	500	505	449	488	549	504	426	397	395	346	294	369	325	282									
5	★ HOPELESS	8040	174	376	406	452	441	539	529	489	497	517	529	471	428	398	390	306	308	293	283	214									
6	★ CARELESS	7161	187	367	404	482	459	462	514	438	502	498	486	354	363	314	229	256	253	197	207	189									
7	★ MOTIONLESS	6144	165	223	418	214	296	398	277	342	420	426	411	417	405	287	238	236	233	260	254	224									
8	★ RECKLESS	5578	59	250	407	270	325	495	396	393	422	364	380	264	213	178	180	176	189	173	214	230									
9	★ COUNTLESS	5136	54	184	196	186	204	205	233	209	248	290	280	292	268	194	240	231	264	340	512	506									
10	★ HARMLESS	5106	107	198	199	217	224	259	275	244	239	281	250	289	233	284	338	288	302	283	310	286									
11	★ BREATHLESS	4320	136	150	237	144	187	274	230	289	313	282	365	278	247	189	153	156	158	170	179	183									
12	★ WORTHLESS	4250	114	200	264	242	222	259	254	215	196	218	218	222	205	207	207	167	179	232	220	209									
13	★ HOMELESS	3126	7	37	46	52	71	88	90	85	100	73	84	118	128	98	67	77	259	617	534	495									
14	★ NEEDLESS	3100	72	118	154	152	171	178	221	193	235	223	176	134	120	144	143	169	116	116	146	119									
15	★ FEARLESS	2983	106	259	298	184	170	207	123	168	171	140	171	97	66	108	73	75	94	105	188	180									

Figure 39: Results from a query for adjectives ending in “-less” in COHA

You can copy this data directly from the COHA results page by selecting the data (Ctrl + C on Windows or Cmd + C on macOS) and paste it into a spreadsheet programme like Excel (Ctrl +

V on Windows or Cmd + V on macOS) and save it as a CSV file. Then, you can import the data or selected parts of it from you file into MATLAB (Figure 40, cf. also Chapter 5.5).

The screenshot shows a MATLAB window titled 'Variables - less\_adj\_COHA'. The workspace contains a variable named 'less\_adj\_COHA' which is a 101x15 table. The table displays the following data:

	1	2	3	4	5	6	7
	adj	ALL	1820	1830	1840	1850	1860
1	" USELESS"	11665	370	496	507	548	
2	" HELPLESS"	10043	241	233	329	358	
3	" ENDLESS"	9458	126	158	335	303	
4	" RESTLESS"	8078	126	326	401	408	
5	" HOPELESS"	8040	174	376	406	452	
6	" CARELESS"	7161	187	367	404	482	
7	" MOTIONLESS"	6144	165	223	418	214	
8	" RECKLESS"	5578	59	250	407	270	
9	" COUNTLESS"	5136	54	184	196	186	
10	" HARMLESS"	5106	107	198	199	217	
11	" BREATHLESS"	4320	136	150	237	144	
12	" WORTHLESS"	4250	114	200	264	242	
13	" HOMELESS"	3126	7	37	46	52	
14	" NEEDLESS"	3100	72	118	154	152	
15	" FEARLESS"	2983	106	259	298	184	

Figure 40: Imported table of adjectives ending in “-less” displayed in MATLAB

As another example, suppose you want to analyse the absolute frequencies of the top 100 collocates that immediately precede or follow “health care” in the COCA corpus. When you are logged in and go to the link of the COCA corpus on the English-Corpora.org website, you may query collocates of “health care” as seen in Figure 41 (finding all collocates one word left and one word right to “health care” in this example).



Figure 41: Querying immediate collocates of “health care” in COCA

The results you obtain may be displayed on the COCA website as shown in Figure 42.

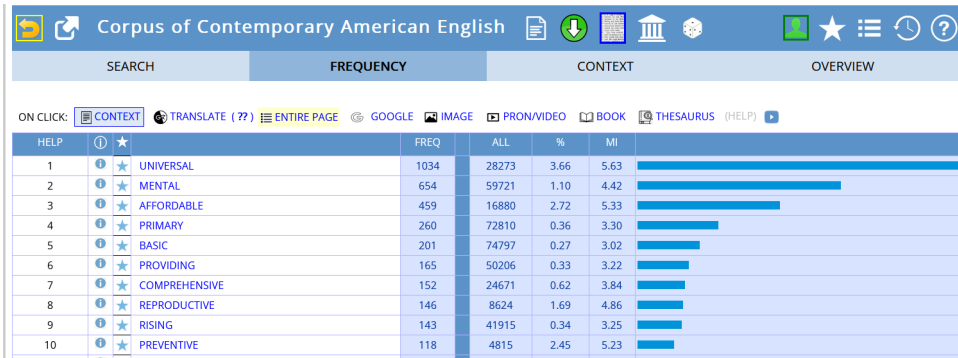


Figure 42: Immediate collocates of “health care” in COCA

To import the data you are interested in into MATLAB, the best approach is again to first use an Excel spreadsheet and then import the data from there (cf. Chapter 5.5). Your MATLAB variable, containing the most frequent immediate collocates of “health care” and their absolute frequencies in the COCA corpus, might then look like the one shown in Figure 43.

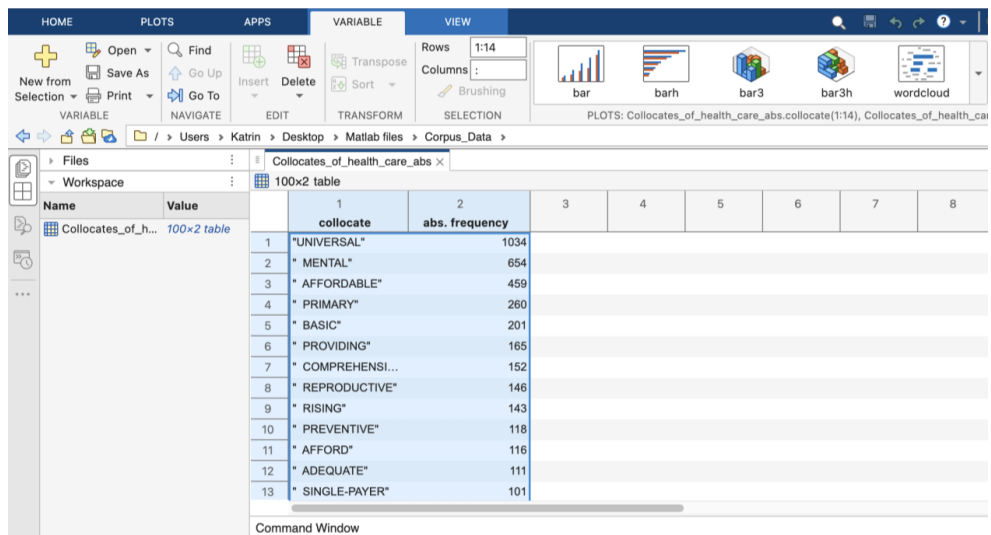


Figure 43: MATLAB variable with collocates of “health care” and absolute frequencies from COCA

Now you can, for instance, create a word cloud from these data, by clicking on the **wordcloud** icon in the **Plots** section (cf. also Chapter 7.13). Figure 44 shows the word cloud generated from this data.

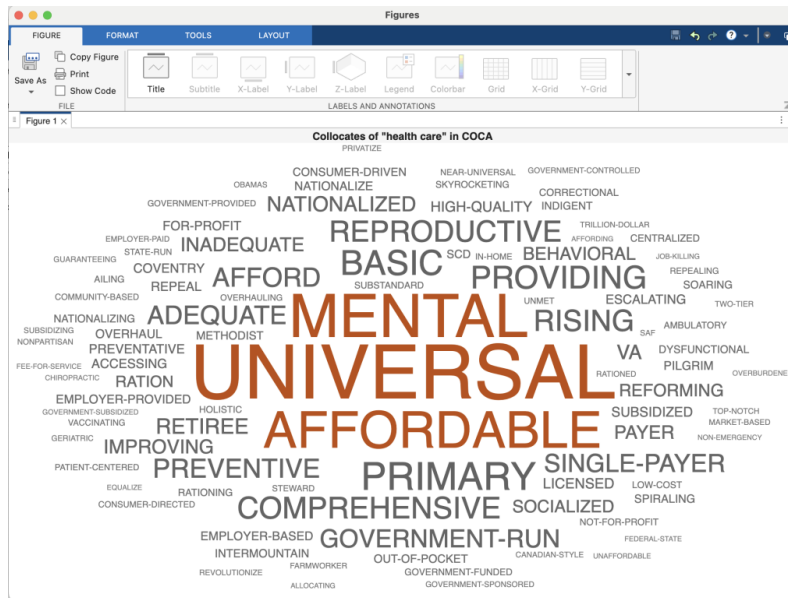


Figure 44: MATLAB word cloud for immediate collocates of “health care” based on COCA data

For detailed guidance on querying corpora from English-Corpora.org, you can refer to their comprehensive help files and tutorials, which offer step-by-step instructions on how to effectively search the available corpora. In addition to written tutorials, there are also video tutorials that visually demonstrate how to perform various searches.

### 4.3 Retrieving data from corpora via the Sketch Engine Interface

This section introduces the use of text corpora available through Sketch Engine (<http://www.sketchengine.eu>), an online text analysis tool. Sketch Engine provides access to a large collection of corpus data, which can be utilised for various linguistic teaching and research purposes (Kilgarriff et al. 2004). The platform allows users to create word lists and concordances, explore collocations and generate word sketches. Many of the corpora in Sketch Engine are enriched with linguistic annotations, such as part-of-speech tags and additional metadata. These features enable users to refine their queries and to target specific text categories within the data.

Many universities provide their members with free access to most Sketch Engine corpora, so it is worth checking whether your institution has a license. Paying subscribers enjoy access to most corpora hosted on Sketch Engine, which can be viewed here: <https://www.sketchengine.eu/corpora-and-languages/corpus-list/>. Some corpora are subject to restricted access, managed by third parties, and are available upon request. Furthermore, users can upload their own data to Sketch Engine and create custom corpora.

Sketch Engine offers a variety of open corpora that are freely accessible without requiring registration. Among the freely available corpora is the British Academic Written English Corpus (BAWE), which contains 6.9 million words extracted from academic texts produced by

university students in the UK (cf. Alsop & Nesi 2009). Another prominent open corpus is the British Academic Spoken English Corpus (BASE), consisting of 1.48 million words transcribed from recordings of academic lectures and seminars (Thompson & Nesi 2001). The Brown Corpus, a foundational resource in corpus linguistics, is also available as an open resource. Compiled by Francis and Kučera (1964), it includes around 1 million tokens of edited American English prose from 1961. The COVID-19 Research Dataset offers a substantial 1.4 billion words sourced from academic papers on the coronavirus and related fields (Wang et al. 2020). These are just a few examples of the open corpora available within Sketch Engine. In addition to these freely accessible corpora, Sketch Engine also offers a range of corpora to trial users, with a 30-day free trial period.

After selecting and loading a corpus, you will access a dashboard (Figure 45) that presents a range of functions for exploring your data.

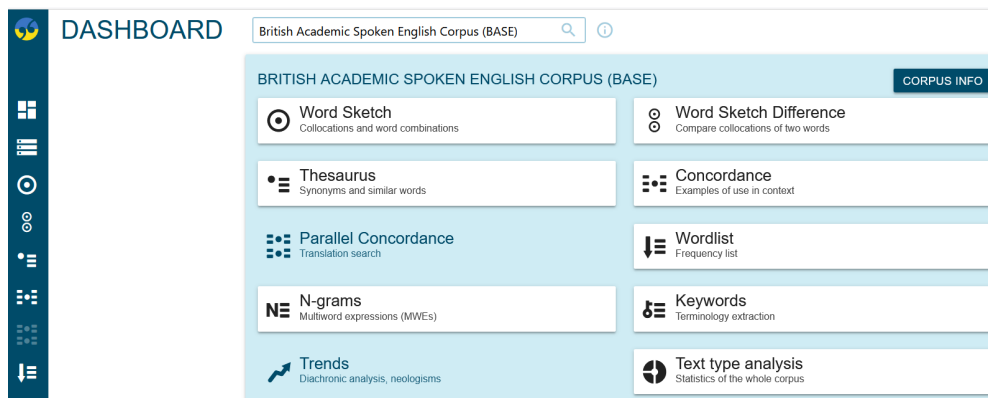


Figure 45: Screenshot of the Sketch Engine corpus dashboard for the BASE Corpus

For example, you can use the Wordlist function to generate a list of the most frequent words in the corpus and analyse their distribution. The Concordance function allows you to search for specific words or phrases within their context. Additionally, the Word Sketch function lets you examine how a word interacts with its surrounding words, while the Word Sketch Differences function allows you to compare the collocations of two words.

The Sketch Engine search interface offers both basic and advanced search options. With the basic search, you can look up specific words or phrases without needing to know query syntax. For more complex searches, the advanced options allow you to refine your queries and obtain more precise results. One way to query corpora in Sketch Engine is by using the Corpus Query Language (CQL), which is similar to CQP (Corpus Query Processor), a query language found in other corpus tools (cf. Chapter 4.4).

In CQL (as in CQP), query elements are enclosed in square brackets [ ], while search items are placed within straight quotation marks: ". Additionally, you can specify words by their corresponding part-of-speech tags to identify words based on their grammatical category, such as nouns, verbs or adjectives. You can check the tagset used in a specific corpus by visiting its information page. For further information on corpus querying and CQL, please consult the following resources from the Sketch Engine documentation: <https://www.sketchengine.eu/documentation/corpus-querying/> and <https://www.sketchengine.eu/documentation/cql-basics/>.

Sketch Engine offers a helpful tool called the CQL Builder, which simplifies the process of constructing CQL queries. In the following example, we use the CQL Builder to search for pronouns ([tag="PP.?"]) followed by the lemma assume or suppose [lemma="assume" | lemma="suppose"]) in the BASE corpus (Figure 46).



Figure 46: Designing a query with the CQL Builder using Sketch Engine

The results generated by the tools in Sketch Engine can be downloaded, although there are restrictions on the amount of data you can download. To export your concordance results from the query, click on the **Download** button (Figure 47).

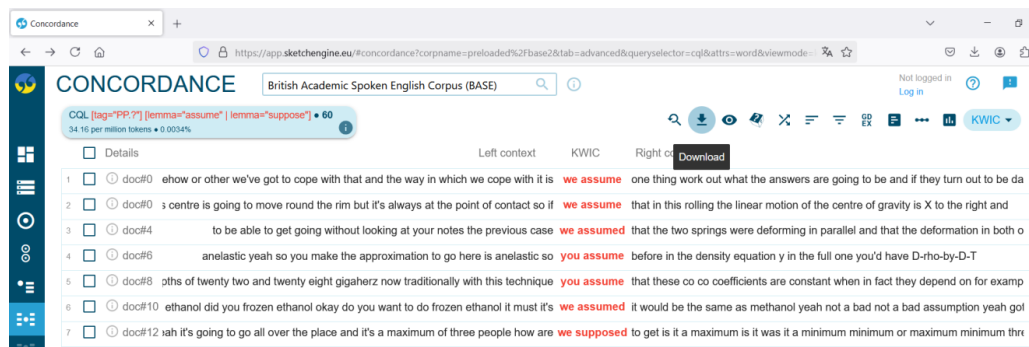


Figure 47: Downloading concordance results from the BASE corpus in Sketch Engine

Near the Download button, you will also find, along with other symbols, a Frequency icon. Clicking on it will display the frequencies of the different constructions identified by your query, in this case, combinations of a pronoun and a form of the verb “assume” or “suppose” in the BASE corpus. In the frequency view, you will also find a Download button so that you can export your frequency results in a variety of file formats (Figure 48).

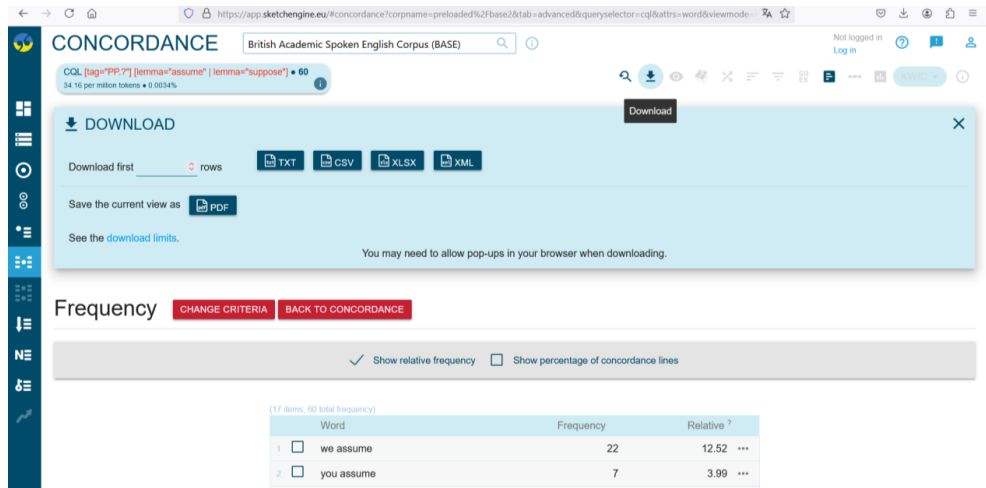


Figure 48: Downloading frequency results from the BASE corpus in Sketch Engine

Let us assume you have downloaded your results and saved them as a CSV file. You can then import the data into MATLAB from your file (cf. Chapter 5.5). For example, you could select the normalised frequencies of the top 5 constructions from the list, store them in a numeric variable and create a bar chart by clicking on the **bar** icon in the in the **Plots** section (Figure 49, cf. also Chapter 7.2).

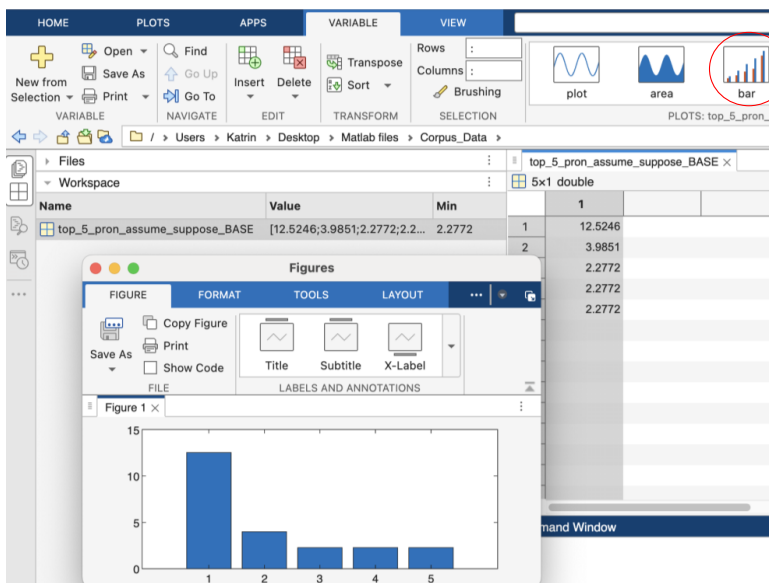


Figure 49: Creating a bar chart in MATLAB

You can further customise the plot by adjusting its appearance and adding labels as needed (Figure 50).

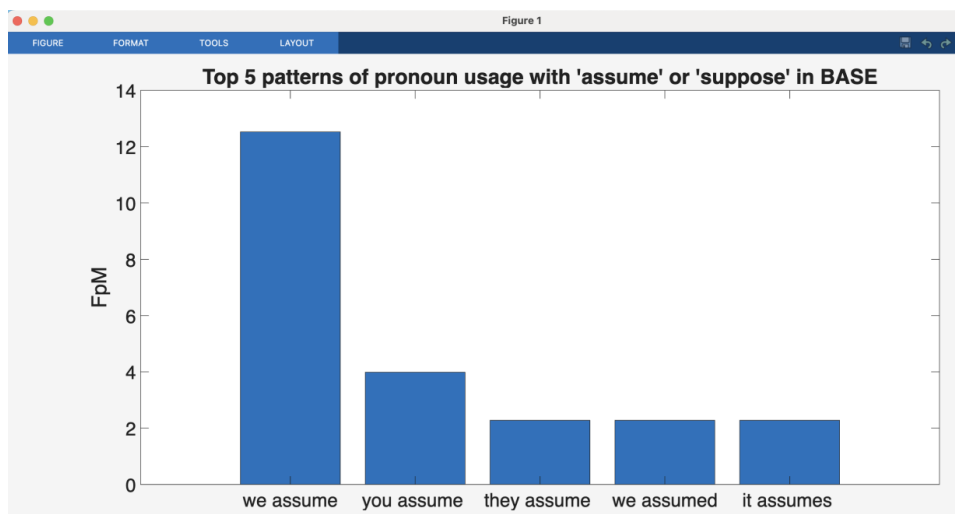


Figure 50: Customised MATLAB bar chart based on query results from BASE corpus

This section has provided a brief overview of Sketch Engine as an example of a platform offering various freely available corpora. Once the data you obtained from your queries is imported into MATLAB, it can be used for various types of analysis and to create visualisations. For further guidance, several tutorial videos are available from the Sketch Engine team on their YouTube channel (<https://www.youtube.com/@SketchEngine>). Additionally, a general overview can be found in their user guide here: <https://www.sketchengine.eu/guide/>.

## 4.4 Retrieving data from corpora through CQPweb interfaces

CQP (Corpus Query Processor, cf. Evert and CWB development team 2022), is a query language for conducting searches within annotated linguistic corpora. It was developed as part of the Corpus Workbench (CWB) and enables users to search for specific words, lemmas or part-of-speech categories in corpus data. CQP, which predates the similar, above-mentioned CQL (Corpus Query Language) from Sketch Engine, can be used through various interfaces that provide access to freely available corpora. Query languages like CQP and CQL stand out compared to simpler query options due to their exceptional flexibility and capacity to handle complex linguistic patterns. They also support the use of regular expressions, enabling users to define criteria such as matching word variations or specific character sequences or excluding certain patterns from a query.

For instance, the OPUS platform, an open-source repository of multilingual corpora, offers access to various aligned parallel datasets, featuring texts in multiple languages. Among these is the Europarl corpus (cf. Koehn 2005), comprising multilingual proceedings from the European Parliament. The OPUS platform provides a CQP-based interface for querying this corpus, accessible at <https://opus.nlpl.eu/bin/opusqcp.pl?corpus=Europarl>.

Figure 51 shows a concordance view of instances where the word "may" is followed by a verb in the English section of the Europarl corpus, along with the corresponding aligned German segments, after executing the query `[word="may"] [pos="V.*"]`.

**OPUS - Corpus query (CWB)**

Query string: `[word="may"] [pos="V.*"] :DE [ ]`  
 20 hits found

	en	de
Souchet (FR)	In addition to the extremely high cost of such a proposal , the cumbersome nature of its management <b>may cause</b> some administrations to use its complexities as a pretext not to provide the information in due time and may give rise to a certain feeling of suspicion among the people working in the sector if they have onerous additional bureaucratic constraints foisted upon them in this way .	Neben den außerordentlichen Kostenbelastungen , die ein solcher Vorschlag mit sich bringt , könnten die schwerfälligen Umsetzungsverfahren auch bestimmte Verwaltungen veranlassen , diese Schwierigkeiten zum Vorwand zu nehmen , die entsprechenden Angaben nicht rechtzeitig zu machen . Es bestünde weiterhin die Gefahr , daß zusätzliche bürokratische Auflagen in diesem Sektor bei den Betroffenen Mißtrauen hervorrufen .
Krivine	We believe that the fight against racism does not really require a legal arsenal , however good this <b>may be</b> , but rather specific	Weiterhin sind wir der Ansicht , daß der Kampf gegen den Rassismus grundsätzlich nicht von einem juristischen Instrumentarium abhängig ist , so gut dies auch sein mag

Figure 51: Concordance view of Europarl data on the OPUS platform after a CQP query

Various universities, such as Saarland University (<https://corpora.clarin-d.uni-saarland.de/cqpweb/>, Figure 52) and Lancaster University (<https://cqpweb.lancs.ac.uk/>, Figure 53), provide so-called CQPweb interfaces that grant access to a variety of freely available corpora.

Welcome to CQPweb at the Department of Language Science and Technology

Enter your username:

Enter your password:

Stay logged in on this computer:

[\[Create account\]](#) [\[Full account options\]](#)

**Corpora available on this server**

**Scientific Corpora**

DaScFlex (1998-2007)	RSC V6.0.4open with UDParse and Surprisal	Royal Society Corpus (RSC) V1.13 (1665-1869)
Royal Society Corpus (RSC) V2.0.2 (1665-1869)	Royal Society Corpus (RSC) V3.4.1 (1665-1869)	Royal Society Corpus (RSC) V3.6.0 (1665-1869)
Royal Society Corpus (RSC) V3.7.0 (1665-1869)	Royal Society Corpus (RSC) V4.0.0 (1665-1869)	Royal Society Corpus (RSC) V5.2.9 (1665-1996)
Royal Society Corpus (RSC) 6.0.1 Full – WEB VERSION	Royal Society Corpus (RSC) 6.0.1 Open – WEB VERSION	Royal Society Corpus V6.0.4 Open Web with Topics

Figure 52: CQPweb at Saarland University



Figure 53: CQPweb at Lancaster University

CQPweb is a web-based graphical user interface (GUI) for CQP. It makes querying linguistic corpora easily accessible to users without requiring them to operate with CQP in a command-line environment. This user-friendly interface simplifies tasks like searching for specific words, lemmas, part-of-speech tags and lexical or grammatical patterns with CQP by providing structured input fields and dropdown menus. Some corpora on the above-mentioned CQPweb interfaces are freely accessible to registered users, while others have restricted access or are available upon request, depending on institutional or licensing agreements.

There is a wealth of tutorials, written documentation and instructional videos available to help users navigate CQPweb interfaces and formulate CQP queries. These resources can, for instance, be found on YouTube channels (e.g., <https://www.youtube.com/@corpusworkbench>) or through academic institutions that host CQPweb instances (e.g., [https://fedora.clarin-d.uni-saarland.de/teaching/Corpus\\_Linguistics/Tutorial\\_CQP\\_I.html](https://fedora.clarin-d.uni-saarland.de/teaching/Corpus_Linguistics/Tutorial_CQP_I.html)).

For instance, consider the Royal Society Corpus (RSC, Kermes et al. 2016, Fischer et al. 2020), which is hosted on Saarland University’s CQPweb. The open release version, RSC 6.0, is available at <https://hdl.handle.net/21.11119/0000-0004-8E37-F>. The corpus can also be accessed for queries through the Saarland University CQPweb interface at <https://corpora.clarin-d.uni-saarland.de/cqpweb/>. To use the interface and run queries, you will need to create a free account. Once registered, you can begin exploring and querying the corpus. This freely available version of the RSC contains English scientific journal articles from 1665 to 1920 and is primarily based on the early centuries of the *Philosophical Transactions of the Royal Society of London*.

Let us assume we would like to see what nouns are premodified by the word “electric” or morphologically related words (e.g., “electrical”, “electricity”) in early scientific texts and how often this pattern occurs per decade in the RSC. To achieve this, we select the query mode “CQP syntax” and use the CQP query `[word="electric.*"] [pos="N.*"]`, which will identify all occurrences where a noun follows a word starting with “electric” (Figure 54).

Menu	Royal Society Corpus (RSC) 6.0.1 Open – WEB VERSION: powered by CQPweb
Corpus queries	Standard Query
Standard query	<input ]"="" electric.*"][pos="N.*" type="text" value="[word="/>
Restricted query	
Word lookup	
Frequency lists	
Keywords	
Analyse corpus	
<b>Saved query data</b>	Query mode: <input type="text" value="CQP syntax"/> <a href="#">Simple query language syntax</a> Number of hits per page: <input type="text" value="50"/> Match strategy: <input type="text" value="Standard"/> Restriction: <input type="text" value="None (search whole corpus)"/> <input type="button" value="Start query"/> <input type="button" value="Reset query"/>
Query history	
Saved queries	
Categorised queries	
Upload a query	
Create/edit subcorpora	
<b>Corpus info</b>	

Figure 54: CQP query for nouns after “electric” or related words in RSC data on Saarland University’s CQPweb interface

After running this query, you will see a concordance, displaying the instances of nouns preceded by “electric” or by words starting with this sequence of letters, along with their context in the corpus (Figure 55).

Your query "[word="electric.*"][pos="N.*"]" returned 17,593 matches in 2,986 different texts (in 78,605,737 words [17,520 texts]; frequency: 223.813 instances per million words) (0.411 seconds)			
<input type="button" value="&lt;"/> <input type="button" value="&lt;&lt;"/> <input type="button" value="&gt;&gt;"/> <input type="button" value="&gt;"/> Show Page: <input type="text" value="1"/> Line view Show in random order Choose action...			
No	Text	Solution 1 to 50	Page 1 / 352
1	103851	Secondly, while I am suspended on the Lines, if the electric Tube	be put near one of my Hands, or my Legs,
2	103851	Effects were produced not only by the Tube, but by all electric Bodies	whatsoever; and I discovered a very simple Principle, which
3	103851	they are become electric, by the Vicinity or Contact of the electric Body	. Thus Leaf-Gold is first attracted by the Tube; and acquires
4	103851	by it. Nor is it re-attracted, while it retains its electric Quality	. But if, while it is thus sustained in the Air
5	103851	&c. though rendered electrical likewise. Amber on the contrary will attract electric Glass	, and other Substances of the same Class, and will repel
6	103851	Form of a Pencil or Broom, when they have acquired an electric Quality	. From this Principle one may with the same Ease deduce the
7	104241	of those that are; and that this repulsive Force of the electrical Blood globules	, excited by Friction, as they flow in their Channels (
8	104242	EXPERIMENT III. Five glass Receivers placed one within another upon an electrical Cement	of Bees-wax and Venice Turpentine, were all exhausted: In
9	104242	EXPERIMENT X. A circular Board of nearly the same Diameter with the electric Cake	, was suspended horizontally by six silk Lines, tied to one
10	104242	the Ends of the Threads were about an Inch distant from the electric Cake	, which was directly under, and had the Ivory Ball on
11	104243	Experiment succeed only well, when the Rest was lower than the electric Area	, and the Arm was supported upon its Elbow, which was
12	104243	when rested upon a Chair, the Chair being lower than the electric Area	, that it might less affect the Effluvia, as was then
13	104243	a pendulous Body, applied to another in the Centre of an electric Area	, is to be ascribed to the Hand that holds it.

Figure 55: Concordance view of query results from RSC in CQPweb

When you use the option **Frequency breakdown** from the drop-down menu under **Choose action** (Figure 55), you will see a frequency table of your query results that shows how often specific word combinations (such as “electric current”, “electric force” or “electrical resistance”) occur in the corpus (Figure 56).

Query "[word="electric.*"][pos="N.*"]" returned 17,593 matches in 2,986 different texts (in 78,605,737 words [17,520 texts]; frequency: 223.813 instances per million words)			
Showing frequency breakdown of words in this query, at the query node; there are 1,560 different types and 17,593 tokens at this concordance position. (0.095 seconds - retrieved from cache)			
<input type="button" value="&lt;"/> <input type="button" value="&lt;&lt;"/> <input type="button" value="&gt;&gt;"/> <input type="button" value="&gt;"/> Breakdown position: <input type="text" value="Node"/> Show: <input type="text" value="Frequency breakdown of words only"/> Download frequency breakdown table (for words)			
No.	Query result	No. of occurrences	Percent
1	electric current	1250	7.11%
2	electric force	740	4.21%
3	electric field	500	2.84%
4	electric currents	496	2.82%
5	electric Fluid	401	2.28%
6	electric discharge	396	2.25%
7	electric light	332	1.89%
8	electrical resistance	302	1.72%
9	electric spark	297	1.69%
10	electric arc	273	1.55%

Figure 56: Frequency table for query results from RSC in CQPweb

You will see the option **Download a frequency breakdown table** (Figure 56), which allows you to download all the results of your query in a text file (Figure 57).

electric.nounRSC-1920.txt

Query "[word="electric.\*"] [pos="N.\*"]" returned 17,593 matches in 2,986 different texts (in 78,605,737 words [17,520 texts]); frequency: 223.813 instances per million

Showing frequency breakdown of words in this query, at the query node; there are 1,560 different types and 17,593 tokens at this concordance position. [0.120 seconds cache]

No.	Search result	No. of occurrences	Percent
1	electric current	1250	7.11%
2	electric force	740	4.21%
3	electric field	500	2.84%
4	electric currents	496	2.82%
5	electric fluid	401	2.28%
6	electric discharge	396	2.25%
7	electric light	332	1.89%
8	electrical resistance	302	1.72%
9	electric spark	297	1.69%
10	electric arc	273	1.55%
11	electrical conductivity	253	1.44%
12	electrical stimulation	198	1.13%
13	electric organ	177	1.01%
14	electrical effects	176	1.00%
15	electrical changes	164	0.93%
16	electrical excitation	163	0.93%
17	electric displacement	162	0.92%
18	electrical machine	159	0.90%
19	electric conductivity	157	0.89%

Figure 57: Text file with downloaded frequency table

When you click on **Distribution** below the **Frequency breakdown** option in the drop-down menu (cf. Figure 55 above), you will see a tabular or graphical representation of the frequency of your query results that shows how the occurrences of the queried pattern are distributed across different sections of the corpus, such as by decade – select either **Show as: Distribution Table** (Figure 58) or **Bar Chart** (Figure 59).

**Distribution of hits for query "[word="electric.\*"] [pos="N.\*"]" returned 17,593 matches in 2,986 different texts**  
**Currently displaying distribution across text classifications.**

Display distribution of:  Show as:

Cross-tabulating against:  Other actions:

**Based on classification:**

Category [.]	Words in category	Hits in category	Dispersion (no. texts with 1+ hits)	Frequency [.] per million words in category
1700	9,757,284	2,115	224 out of 3,505	216.76
1800	46,105,686	10,180	1,712 out of 9,528	220.80
1900	20,159,911	5,298	1,050 out of 3,162	262.80
Total:	76,022,881	17,593	2,986 out of 16,195	231.42

**Based on classification:**

Category [.]	Words in category	Hits in category	Dispersion (no. texts with 1+ hits)	Frequency [.] per million words in category
1730	599,977	32	10 out of 352	53.34
1740	1,006,093	430	36 out of 494	427.40
1750	1,179,112	440	46 out of 484	373.16
1760	972,672	170	24 out of 408	174.78
1770	1,501,388	581	49 out of 408	386.98
1780	1,354,124	268	42 out of 300	197.91
1790	1,335,193	194	17 out of 219	145.30
1800	1,615,564	242	40 out of 518	149.79
1810	1,446,900	176	44 out of 666	121.64
1820	1,408,184	218	27 out of 286	154.81

Figure 58: Distribution table for query results from RSC in CQPweb



Figure 59: Bar charts with distribution of query results from RSC in CQPweb

Suppose we have extracted the frequency per million values for the queried pattern for each decade from 1730 onwards from the distribution information in CQPweb (1730s: 53.34, 1740: 427.40 etc.). Based on these data, we create a variable in MATLAB. This allows us, for instance, to display values such as the minimum, maximum, median and standard deviation of our values in the MATLAB workspace or to sort them via the Variables editor (Figure 60, cf. also Chapters 3.6 and 5.4) or to create a plot from them by selecting one of the available options from the **Plots** tab (cf. Chapter 7).

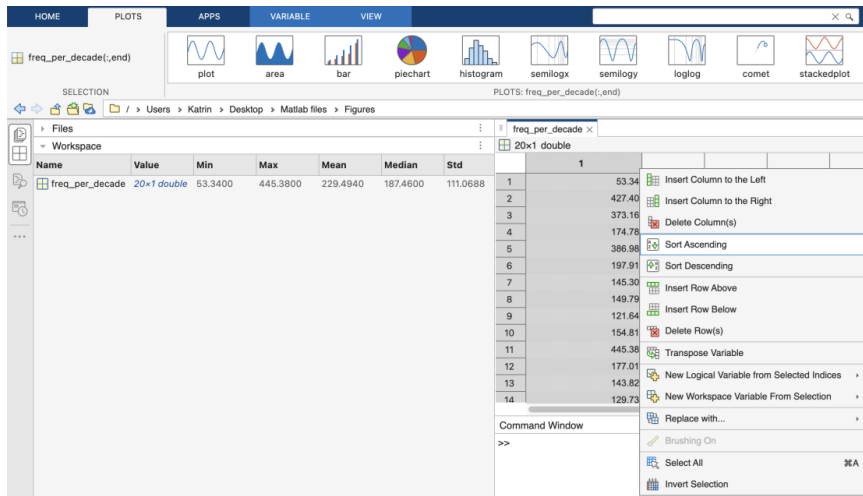


Figure 60: Sorting values in MATLAB Variables editor and displaying selected statistical measures in workspace

In summary, this chapter has highlighted several publicly available corpora. There are many more corpora available, and depending on your research needs, you may want to explore others that suit your specific requirements. This chapter has also demonstrated how you can retrieve and query data from some of the available corpora. The next chapter will guide you through the process of creating variables for your data in MATLAB, whether that data comes from corpora or other sources. This will help you organise and prepare your data for further analysis.

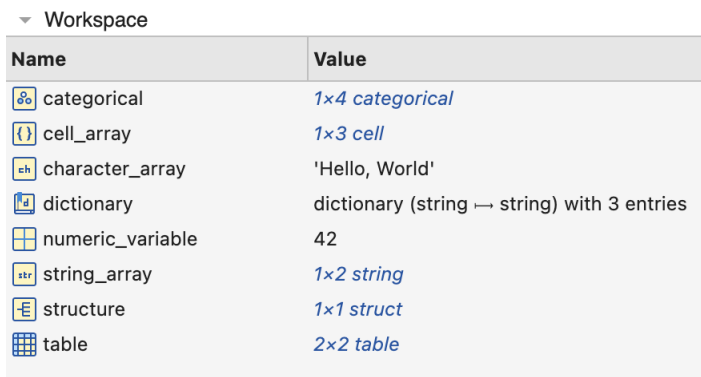
# 5 Creating and editing variables in MATLAB®

## 5.1 Introduction to MATLAB® data types

In MATLAB, several data types are available to manage a variety of data. These data types are designed to suit a wide range of tasks, from simple mathematical operations to more complex analyses, including those required for linguistic research. In this section and in Chapter 5.2, we will have a look at some key data types that may be useful for linguistic analysis, including numeric types, character and string arrays, cell arrays, categorical arrays, tables, structures and dictionaries. This section provides a short overview, while Chapter 5.2 offers a more detailed discussion with examples.

Numeric types in MATLAB are primarily used for mathematical computations. Character arrays and string arrays are designed to store sequences of characters and text data. In cell arrays, each element can hold a distinct type of data, which makes them suitable for combining different data types. Categorical arrays represent qualitative data that come from a finite set of categories. They are particularly helpful when working with non-numeric information. Tables provide a structured format that organises data into rows and columns. Structures store data in fields, and these fields can contain variables of different types and sizes. Dictionaries use key-value pairs for data storage, and this feature is useful when associative data must be managed.

The different data types are represented by specific icons in the workspace, so that users can quickly identify the type of each variable (Figure 61). Numeric data, for instance, are represented by a matrix icon in the workspace.











▼ Workspace		
Name		Value
 categorical		1×4 categorical
 cell_array		1×3 cell
 character_array		'Hello, World'
 dictionary		dictionary (string ↦ string) with 3 entries
 numeric_variable		42
 string_array		1×2 string
 structure		1×1 struct
 table		2×2 table

Figure 61: Icons representing different data types in the workspace

You can create variables for the different data types in MATLAB using several methods. One way is through the Command Window by assigning values directly. The Command Window serves as the interactive interface where you can execute commands and define variables. To create a new variable, enter a variable name in the Command Window, followed by an equal sign (=) and the value you want to assign to the variable. The value can be of various types, such as numbers, strings or arrays depending on your needs. Your variable name must start with a letter and can contain letters, digits or underscores. Variable names are case-sensitive and

should not conflict with MATLAB keywords or existing function names. You can check for valid variable names using the `isvarname` function.<sup>10</sup>

You can also create (numeric) variables via the New Variable icon in the MATLAB Home tab toolbar or by right-clicking in the Workspace panel (cf. Chapter 5.3). Additionally, when you use the Import Tool in MATLAB, you can choose the variable type for the imported data (e.g., numeric matrix, string array, table, cf. Chapter 5.5). Examples of how to create and manage different types of arrays (numeric, textual, categorical and mixed data types) via commands in the Command Window are provided in Chapter 5.2. Additionally, you can find an overview of the data types in MATLAB at the following link in the MathWorks documentation: <https://www.mathworks.com/help/matlab/data-types.html> with examples and explanations on the types of operations that can be performed with these data types. MathWorks also offers a brief video tutorial titled “Introducing MATLAB Fundamental Classes (Data Types)” available at <https://www.mathworks.com/videos/introducing-matlab-fundamental-classes-data-types-101503.html> for further learning.

To monitor memory usage and determine how much is available for variables, you can use the `memory` function (only available on Windows systems). This function provides insights into the maximum possible array size and the memory currently used by MATLAB (<https://www.mathworks.com/help/matlab/ref/memory.html>).

```
[userview, systemview] = memory
```

MATLAB also has safeguards to prevent the creation of overly large arrays. Additionally, to avoid any “Out of Memory” errors, you can take a few steps, for instance, clearing unused variables by using the `clear` function to free up memory or by using sparse matrices instead of full ones when you are dealing with matrices that have a lot of zeros (cf. Chapter 5.2.1).

## 5.2 Defining variables of different data types via the Command Window

### 5.2.1 Numeric variables

Creating numeric variables is probably one of the most common tasks in MATLAB. Numeric data in MATLAB are used for working with numbers, whether you are performing calculations or analysing data (cf. <https://www.mathworks.com/help/matlab/numeric-types.html>). The numeric types in MATLAB include integers, i.e., whole numbers like 1, -3 or 42, and so-called floating-point numbers. These are numbers that can include fractions and decimals, like 3.14 or -0.001.<sup>11</sup> In MATLAB, all numeric types allow you to perform basic operations on numeric arrays, which are collections of numbers arranged in specific layouts, such as rows, columns or

---

<sup>10</sup> <https://www.mathworks.com/help/matlab/ref/isvarname.html>

<sup>11</sup> They are called “floating-point” because the decimal point can “float” to represent very large or very small numbers, depending on the value. In contrast, fixed-point data types have a specific number of digits reserved both before and after the decimal point, providing a fixed location for the decimal. By default, MATLAB stores all numbers as double-precision floating-point numbers. This means the numbers are stored with a high level of accuracy and precision and can represent a wide range of values. However, if memory is a concern, you can choose to use single-precision floating-point numbers or integers, which require less storage (cf. [https://www.mathworks.com/help/matlab/matlab\\_prog/floating-point-numbers.html](https://www.mathworks.com/help/matlab/matlab_prog/floating-point-numbers.html)).

multi-dimensional grids. That means you can organise, manipulate and perform calculations on large sets of numbers using a single command.

You can assign a single numeric value to a variable, or you can create an array or matrix of numbers. To create a simple numeric variable, type the variable name, followed by the equal sign and the value you wish to assign, for example:

```
wordLength = 5
```

Press Enter and the variable `wordLength` will now occur in your workspace. MATLAB will also display the result in the Command Window. To suppress the display of output in the Command Window, end the statement with a semicolon, for example:

```
wordLength = 5;
```

To edit a variable via the Command Window, you can simply reassign it by entering the variable name followed by an equal sign and the new value. For example, if you want to change the value of the variable `wordLength`, you can enter the following command:

```
wordLength = 6;
```

In this case, your command will overwrite the existing value of `wordLength` (which was 5) with the new value (6).

You can also create arrays or matrices of numbers. If you would like to create an array with multiple elements in a single row, you can separate the elements using either spaces or commas, and you need to enclose the values within square brackets `[ ]`, e.g.,

```
wordLengths = [3 5 4 6]; or
```

```
wordLengths1 = [3, 5, 4, 6];
```

Both will result in a 1x4 array containing the specified elements. While spaces are often preferred for simplicity, commas can be useful for clarity, especially in longer expressions.

To create a numeric array where the elements are arranged in a single column, you can separate each element with a semicolon (`;`). This results in a column vector, where each value is stacked vertically, one below the other. Suppose we have a list of words: “apple”, “banana”, “cherry”, “date” and we want to represent the number of syllables in each of these word as a column vector. Then you can type:

```
syllables = [2; 3; 2; 1]
```

Figure 62 shows the output in the Command Window and the workspace for examples of numeric variables: a column vector (`syllables`) containing multiple values (2, 3, 2, 1) as a 4x1 array, a scalar variable (`wordLength`) representing a single number (5) and a row vector (`wordLengths1`) holding multiple values (3, 5, 4, 6) as a 1x4 array.

The screenshot shows the MATLAB workspace and command window. The workspace contains variables: syllables (2x3 matrix), wordLength (scalar 5), wordLengths (1x4 vector [3,5,4,6]), and wordLengths1 (1x4 vector [3,5,4,6]). The command window shows the following code and output:

```

>> wordLength = 5
wordLength =
    5
>> wordLengths = [3 5 4 6]
wordLengths =
    3    5    4    6
>> wordLengths1 = [3, 5, 4, 6]
wordLengths1 =
    3    5    4    6
>> syllables = [2; 3; 2; 1]
syllables =
    2
    3
    2
    1
>> |

```

Figure 62: Examples of variable assignments for numeric types

When working with data from a corpus, you may need to represent information such as the frequency of words across different documents. One effective way to organise this data is by using a matrix. In this case, each row can represent a different word, while each column corresponds to a specific document. The values within the matrix will then indicate the frequency of each word within the respective document. In MATLAB, a numeric matrix is created by specifying its values inside square brackets `[ ]`, with each row of the matrix separated by a semicolon. The numbers within each row are separated by spaces or commas.

Let us consider a small corpus consisting of three documents. We are interested in the frequency of three words: “gravity”, “motion” and “force”. The task is to represent the frequency of each of these words within each document using a matrix.

For this example:

- “gravity” appears 3 times in document 1, 1 time in document 2 and 0 times in document 3,
- “motion” appears 1 time in document 1, 2 times in document 2 and 1 time in document 3,
- “force” appears 0 times in document 1, 1 time in document 2 and 2 times in document 3.

To represent this information in MATLAB, we can create a matrix where:

- The first row represents the word “gravity”, with the values 3, 1 and 0 corresponding to the frequency of “gravity” in documents 1, 2 and 3, respectively.

- The second row represents the word “motion”, with the values 1, 2 and 1 showing the frequency of “motion” in each document.
- The third row represents the word “force”, with the values 0, 1 and 2 indicating how often “force” appears in the three documents.

Thus, you can create the corresponding matrix as follows:

```
word_freq_matrix = [3 1 0; 1 2 1; 0 1 2]
```

Upon running the above command in MATLAB, the matrix will be displayed in the Command Window as in Figure 63 (unless a semicolon is added after the command to suppress the output). It will also be added to the workspace, where you can see the variable `word_freq_matrix` listed. To view the variable in more detail, you can open the **Variables editor** by clicking on the variable in the workspace. The Variables editor provides a table-like interface for inspecting these data. It is a user-friendly tool that allows for easy viewing, editing and manipulation of variables in a more visual format (cf. top right in Figure 63). Further details on using the Variables editor will be provided in Chapter 5.4.

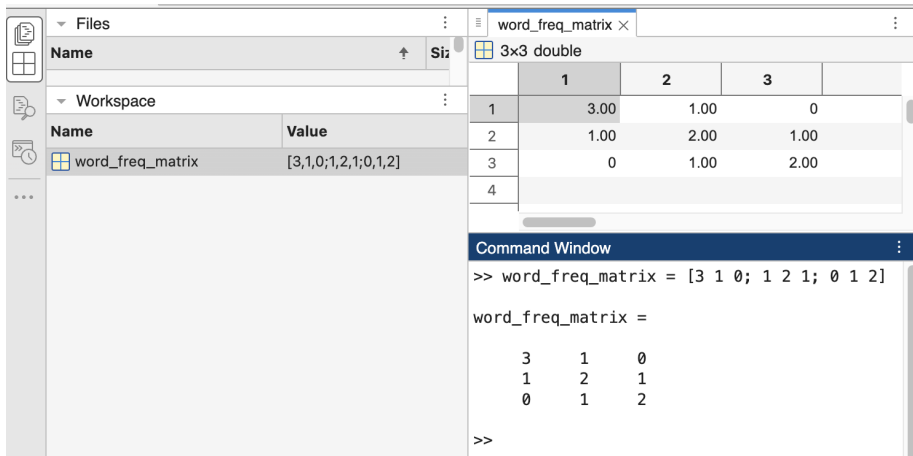


Figure 63: Creating a numeric matrix

Another option for creating a numeric variable is to click on **New Variable** in the dropdown list under **Variable** in the MATLAB **Home** tab toolbar (cf. Chapter 5.3).

If you want to work with very large arrays in MATLAB, you should consider and monitor your memory usage effectively. For instance, if you are dealing with large matrices that contain many zeros, consider using sparse matrices to save memory. You can convert a full matrix to sparse format using the `sparse` function:

```
A = [0 0 0 5; 0 2 0 0; 1 3 0 0; 0 0 4 0];
```

```
S = sparse(A);
```

With the above lines, we first create a full matrix  $A$  and then convert it into a sparse matrix  $s$  (cf. Figure 64).<sup>12</sup>

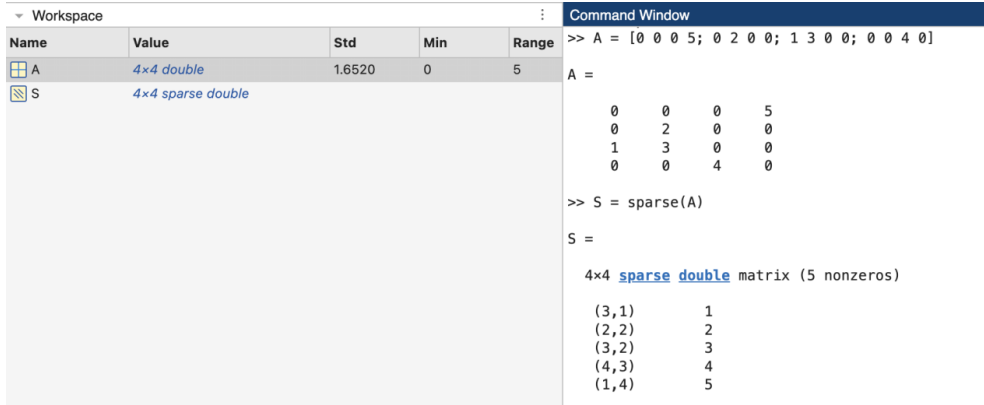


Figure 64: Converting a full matrix to sparse format

More information on working with matrices and arrays in MATLAB can be found at [https://www.mathworks.com/help/matlab/learn\\_matlab/matrices-and-arrays.html](https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html).

### 5.2.2 Character arrays

In MATLAB, text data is handled using two main types: character arrays and string arrays. These are ways of storing and working with text, but they differ in their flexibility and usage.

A character array is the traditional way of representing text in MATLAB. It is essentially a one-dimensional array, where each individual character is stored as a separate element. This means that in a character array, each letter, number or symbol is treated as a distinct unit, which makes it easy to manipulate individual characters. Character arrays are particularly useful when working with short sequences of text, such as file names, labels or other small pieces of linguistic data, but you can also use them for working with longer sequences of text.

To create a character array in MATLAB, enclose your character, word or text in single quotes, for example:

```
city = 'London';
```

You might, for instance, also store a file name or a label as a character array with commands such as `file_name = 'document.txt'` or `label = 'Linguistics'`.

Note that if you want to include a literal single quote or apostrophe within the text of a character array, you need to escape it to prevent MATLAB from interpreting it as the delimiter that marks the start or end of the string. Escaping refers to the process of including special characters within the text that would otherwise have a special meaning in MATLAB, such as the single quote itself. Suppose you want to store “London’s” in a character array. Since you are using single

<sup>12</sup> cf. also <https://www.mathworks.com/help/matlab/math/constructing-sparse-matrices.html>, <https://www.mathworks.com/help/matlab/math/sparse-matrix-operations.html> and <https://www.mathworks.com/help/matlab/math/computational-advantages-of-sparse-matrices.html>

quotes to define your character array, you need to escape the apostrophe by using two single quotes (' '):

```
city1 = 'London''s';
```

Character arrays can be used for various text processing tasks. Here are some examples of what you can do with a character array, along with relevant examples of commands.

For instance, to convert all uppercase characters in a character array to lowercase, the `lower` function can be used. Consider the following character array which contains a short extract from the UN Convention on the Rights of the Child, available at <https://www.ohchr.org/en/instruments-mechanisms/instruments/convention-rights-child>.

```
text = 'States Parties recognize the important function performed by the mass media and shall ensure that the child has access to information and material from a diversity of national and international sources, especially those aimed at the promotion of his or her social, spiritual and moral well-being and physical and mental health.';
```

To convert a text from a character array to lowercase, you can apply the `lower` function as follows:

```
text1 = lower(text);
```

The new text in `text1` will now be: `states parties recognize the important function performed ...`

You may use the `strsplit` function if you want to split the text in the character array `text` at whitespace characters and return the result as a cell array (cf. Chapter 5.2.4) of substrings.

```
words = strsplit(text);
```

The output of `words` is a  $1 \times 51$  cell array: `{'States'} {'Parties'} {'recognize'} ...`

The string “well-being” is treated as a single element of that array (`{'well-being'}`). Note that the output in this case may also include punctuation marks attached to the words, e.g., `{'health.'}`.

You can also use the `regexp` function to extract all words from a character array while ignoring punctuation and to also return them in a cell array format:

```
words1 = regexp(text, '\w+', 'match');
```

In this command, `\w+` is a regular expression pattern where `\w` matches any word character, which includes letters (both uppercase and lowercase), digits and underscores. It is equivalent to the character class `[a-zA-Z0-9_]`. The `+` sign is a quantifier that means “one or more” of the preceding element. Therefore, `\w+` matches sequences of word characters that form whole words. The expression `'match'` specifies that the function should return the actual substrings that match the regular expression. Here, the output of `words` will also be a cell array such as: `{'States'} {'Parties'} {'recognize'} ...` However, in this case, we obtain a  $1 \times 52$  cell array as “well-being” is split into two separate elements: `{'well'} {'being'}`.

Figure 65 shows the four variables `text`, `text1`, `words` and `words1` in the workspace and in the Variables editor which we have created by the previous steps – creating a character array for a text, converting it to lowercase and extracting all words from it with two different methods.

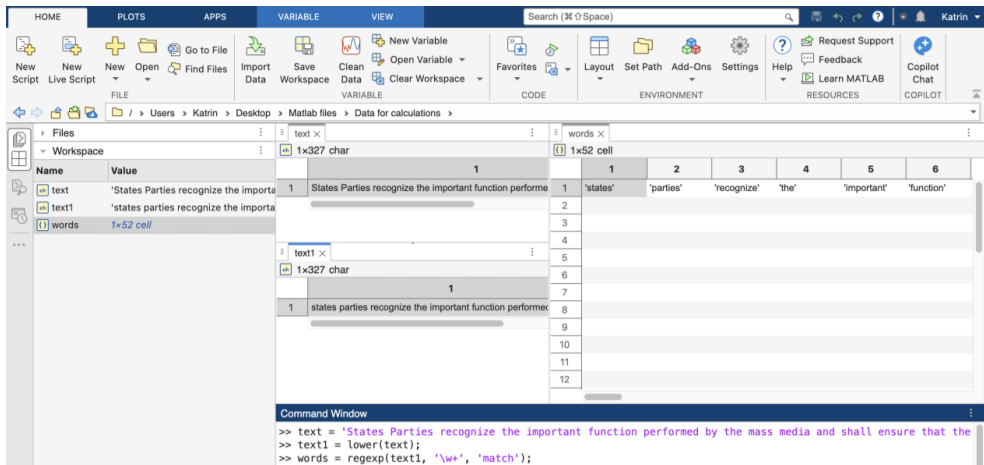


Figure 65: Character arrays in workspace and in Variables editor and cell arrays obtained by extracting words from a character array

You can use the `length` function to determine the number of elements in a character array. It returns the total number of characters, including spaces and punctuation. The command `length(text)`, for instance, will return 327.

If you have an array of a different data type, you can convert it to a character array using the `char` function:

```
Character_array = char(X);
```

This converts the input array `X` (e.g., a string array or a cell array) to a character array.<sup>13</sup> Also consider using string arrays for more advanced text manipulation, which provide additional functionalities for handling text data.

### 5.2.3 String arrays

Like character arrays, string arrays are tools for storing and manipulating text data, including individual characters, words, sentences, phrases or even entire texts. String arrays generally allow for more flexible handling of text compared to character arrays. They enable you to perform various operations such as joining strings, comparing them and modifying their contents with greater ease. A string array can be a string scalar, i.e., a string array that contains only one element such as one word, one phrase or a single piece of text. There is no strict limit on the length of the text you can store in a string array (practical limits are only determined by available memory).

<sup>13</sup> For more detailed information on using character arrays, you can refer to the MATLAB documentation at <https://www.mathworks.com/help/matlab/ref/char.html>, <https://www.mathworks.com/help/matlab/characters-and-strings.html> and [https://www.mathworks.com/help/matlab/matlab\\_prog/represent-text-with-character-and-string-arrays.html](https://www.mathworks.com/help/matlab/matlab_prog/represent-text-with-character-and-string-arrays.html).

You can create a scalar string array by enclosing text in double quotes in your command, for instance:

```
lemma = "run"
```

or

```
title_poem = ["Lines on the Antiquity of Microbes"]
```

When you want to include double quotes within a string defined by double quotes, you need to use two double quotes to represent a single double quote. This is similar to how you escape an apostrophe or single quote in a character array as explained above. In the following example, `string1` contains He said, "Where are you from?":

```
string1 = "He said, ""Where are you from?""";
```

A string array can, for instance, store the individual tokens of a sentence or a text, e.g.:

```
tokens = ["the", "cat", "sat", "on", "the", "mat"]
```

In this case, `tokens` is not a scalar. This string array would be a 1x6 string array in MATLAB. Each element of this array is a string that stores a sequence of characters. In this case, it contains individual words (tokens) that may be further analysed for frequency, word length etc.

Now, suppose you are analysing a corpus of news articles about travel and tourism. You may want to extract and track the cities mentioned across different articles. You could create a string array to store the names of cities mentioned in those articles. For example:

```
cities = ["London", "Paris", "Tokyo", "Berlin"];
```

Another scenario where a string array might be useful is when you need to store a list of stop words. Stop words are common words such as “the”, “and”, “or” and “in” that are often filtered out during text analysis. In tasks such as word frequency analysis or topic modelling, for instance, you may want to exclude stop words from the analysis because they are highly frequent and do not add much useful information to the content. By storing stop words in a string array, you can reference them and filter them out from larger bodies of text (cf. Chapter 10.3). For example, you might create a string array of specific stop words with the following command:

```
stopwords = ["the", "and", "of", "in", "on"];
```

A string array can also store collocations or word pairs that occur frequently together in the corpus, e.g.

```
collocations = ["heavy rain", "heated debate", "sharp increase", "strong evidence"].
```

String arrays may also store multiple pieces of text data in a structured format, similar to a table. The following example would be a 3x2 string array in MATLAB:

```
words = ["table", "chair";  
        "chaise", "table";  
        "mesa", "silla"];
```

Each element of this array is a string that represents a word. The individual strings are organised in a matrix format with three rows and two columns (Figure 66).

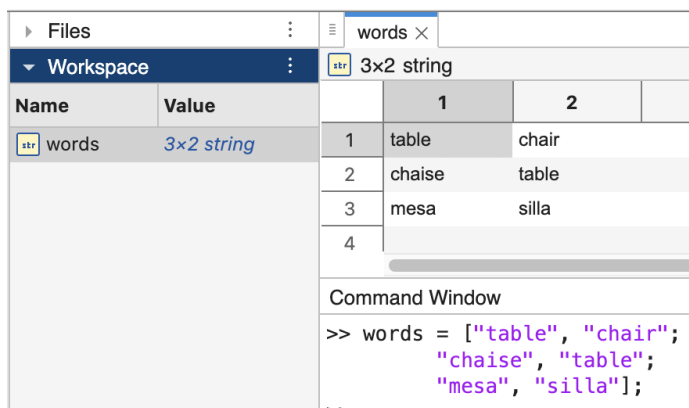


Figure 66: Creating a 3x2 string array

You can access individual elements using indexing, for example, `words(1, 1)` will return "table".

You can concatenate (combine) strings or compare them directly by using a sequence of commands like the one below:

```
word1 = "linguistic";
word2 = "analysis";
combined = word1 + " " + word2;
```

Here, the expression `word1 + " " + word2` combines the strings `word1`, a space " " and `word2` to form a new string, in this case "linguistic analysis".

To convert data into a string in MATLAB, you can use the `string` function, which allows you to create string arrays from various data types, e.g., character vectors, cell arrays or numeric arrays. Here, we have defined the following phrase as a character vector:

```
charVector = 'linguistic analysis';
```

The `string` function can be used to convert `charVector` into a string:

```
strArray = string(charVector);
```

You can split strings into individual tokens using the `split` function. This allows you to break down sentences or texts into smaller, manageable units for further analysis. For instance, consider a sentence from the Old Bailey Proceedings, a collection of historical trial transcripts from the Old Bailey courthouse in London. The texts of these proceedings are available online (cf. Hitchcock et al. 2023). They are also available in a corpus format (Huber et al. 2016) at Saarland

University's CQPweb interface<sup>14</sup> (cf. Chapter 4.4). Here is a sample sentence from the Old Bailey Proceedings that you can store as a string in a MATLAB variable:

```
sentence = "I am brother to the last witness."15
```

The `split` function can divide the sentence stored in the string variable `sentence` into individual elements, in this case, words:

```
words = split(sentence)
```

After running the command, MATLAB creates a new  $7 \times 1$  string array `words` where each word in the sentence is separated into its own element. The `split` function splits the string at whitespace characters by default. Note that punctuation at this step is not yet removed. The last element in the words array will still include the period, making it "witness.".

To split a string with a sentence into individual words using regular expressions, you can also use the `regexp` function:

```
words1 = regexp(sentence, '\s+', 'split');
```

To exclude punctuation marks from a string, you can use the `erasePunctuation` function from the Text Analytics Toolbox (cf. Chapter 10.3), e.g. to remove punctuation from the sentence, use:

```
sentence_without_punctuation = erasePunctuation(sentence)
```

Figure 67 shows the workspace and output of the previous commands for creating a string for a sentence, splitting it into individual words in a new string array and excluding punctuation marks from the original sentence.

---

<sup>14</sup> Landing page: <http://hdl.handle.net/11858/00-246C-0000-0023-8CFB-2>

<sup>15</sup> Old Bailey Proceedings, Dec. 1784. Trial of Thomas Chapman (t17841208-4). Available at: <https://www.oldbaileyonline.org/record/t17841208-4> (Accessed: 11/01/2026)

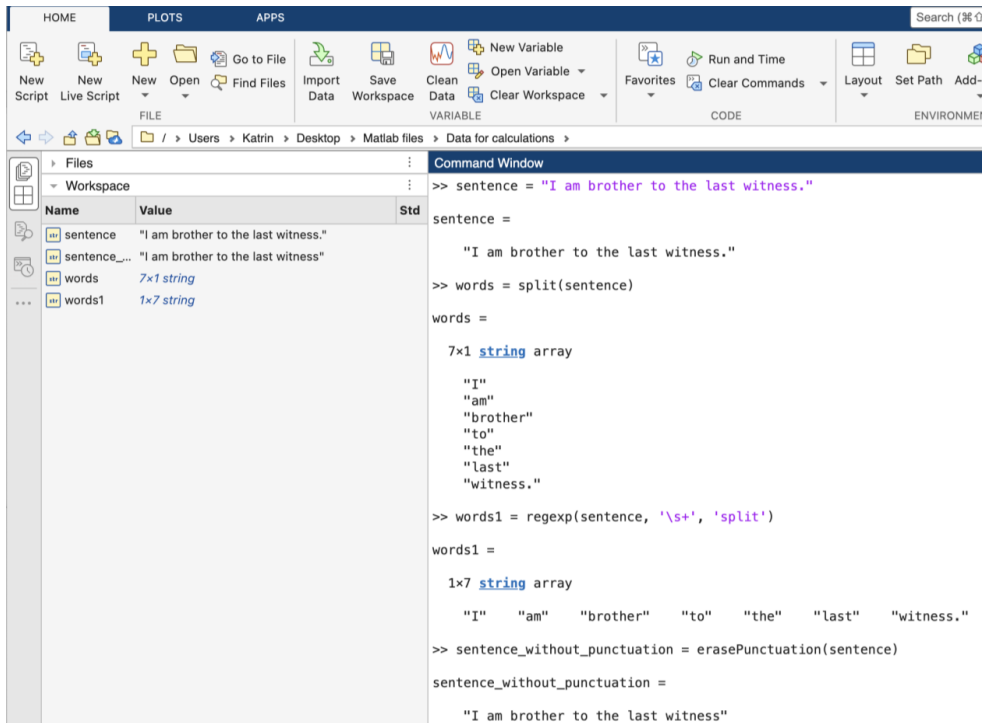


Figure 67: Examples of working with string variables

To find the length of each string in a string array, you can use the `strlength` function. For instance, `lengths = strlength(sentence)` will return the value 33 as "I am brother to the last witness." is 33 characters long, including spaces and punctuation.

If we have split our string into individual words in the string variable `words`, a command such as `lengths1 = strlength(words)` will give us a new variable `lengths1` that contains values for the length of each word (1, 2, 7, 2 etc.). If punctuation has not been removed prior to this step, it may interfere with the length. For instance, the length of "witness." will be counted as 8 due to the period still attached to the word.

Let us have a look at the following example, an extract from the Constitution of the United States. In this example, we first define the text data and then remove punctuation marks as a first text processing step. After that, we split the text into words, and subsequently we calculate the lengths of each word so that we may be able to present this information in a visualisation. Our example here is Section 3 of the Constitution of the United States (available online at: <https://www.archives.gov/founding-docs/constitution-transcript>).

First, we create a string variable `textData` containing the sample text.

```
textData = "The Senate of the United States shall be composed of two Senators from each State, chosen by the Legislature thereof, for six Years; and each Senator shall have one Vote. Immediately after they shall be assembled in Consequence of the first Election, they shall be divided as equally as may be into three Classes. The Seats of the Senators of the first Class shall be vacated at the Expiration of the second Year, of the second Class at the
```

Expiration of the fourth Year, and of the third Class at the Expiration of the sixth Year, so that one third may be chosen every second Year; and if Vacancies happen by Resignation, or otherwise, during the Recess of the Legislature of any State, the Executive thereof may make temporary Appointments until the next Meeting of the Legislature, which shall then fill such Vacancies. No Person shall be a Senator who shall not have attained to the Age of thirty Years, and been nine Years a Citizen of the United States, and who shall not, when elected, be an Inhabitant of that State for which he shall be chosen. The Vice President of the United States shall be President of the Senate, but shall have no Vote, unless they be equally divided. The Senate shall chuse their other Officers, and also a President pro tempore, in the Absence of the Vice President, or when he shall exercise the Office of President of the United States. The Senate shall have the sole Power to try all Impeachments. When sitting for that Purpose, they shall be on Oath or Affirmation. When the President of the United States is tried, the Chief Justice shall preside: And no Person shall be convicted without the Concurrence of two thirds of the Members present. Judgment in Cases of Impeachment shall not extend further than to removal from Office, and disqualification to hold and enjoy any Office of honor, Trust or Profit under the United States: but the Party convicted shall nevertheless be liable and subject to Indictment, Trial, Judgment and Punishment, according to Law.";

To remove punctuation marks from the text to ensure that they do not become part of the words, we use the `erasePunctuation` function from the Text Analytics Toolbox:

```
cleanedText = erasePunctuation(textData);
```

Then, we use the `split` function to divide the string `textData` at whitespace characters to return an array of words. In this way, each word from the text becomes an element in the `words` array.

```
words = split(cleanedText);
```

Next, we apply the `strlength` function to calculate the number of characters in each word from the `words` array. This returns an array, e.g., named `wordLengths`, that contains the lengths of all words.

```
wordLengths = strlength(words);
```

Using the previous sequence of commands, we obtain an array of 347 words and a corresponding array of 347 values for the respective lengths of these words (3, 6, 2, 3 etc. for the words "The", "Senate", "of", "the" etc.). The array has a minimum value of 1, a maximum value of 16 and a mean value of ca. 4.64, which we can see in the Workspace panel (Figure 68).

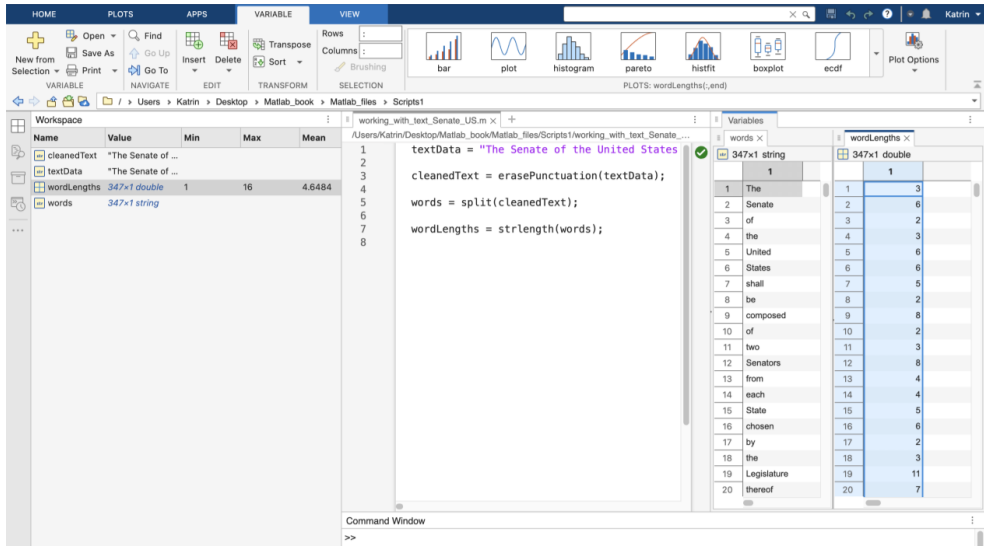


Figure 68: Preprocessing a text string and calculating the length of each word

We can then select our data from the variable `wordLengths` in the Variables editor and choose a corresponding plot type from the toolbar, e.g., a histogram or a boxplot.

For more examples of how to work with strings in MATLAB, have a look at the MathWorks documentation on this topic at the following links:

#### *Create string arrays*

[https://www.mathworks.com/help/matlab/matlab\\_prog/create-string-arrays.html](https://www.mathworks.com/help/matlab/matlab_prog/create-string-arrays.html)

#### *String arrays*

<https://www.mathworks.com/help/matlab/ref/string.html>

#### *Characters and strings*

<https://www.mathworks.com/help/matlab/characters-and-strings.html>

#### *Text in string and character arrays*

[https://www.mathworks.com/help/matlab/matlab\\_prog/represent-text-with-character-and-string-arrays.html](https://www.mathworks.com/help/matlab/matlab_prog/represent-text-with-character-and-string-arrays.html)

### 5.2.4 Cell arrays

Cell arrays in MATLAB offer a highly adaptable data structure, capable of storing elements of different types and dimensions. Each element (or “cell”) within a cell array can contain anything from numeric values and text strings to more complex data structures like other arrays. This flexibility makes cell arrays a good choice for managing diverse and varied data sets. An example of a cell array is `sentimentWords = {'happy', 0.8; 'sad', -0.7; 'excited', 0.9; 'angry', -0.6}`, where the curly brackets `{}` are used to define the elements of the array, allowing each cell to store different types of data, such as words and their corresponding sentiment scores.

To create a cell array, a versatile data type that can hold different types of data, you can use either the curly braces `{}` as in the example above or the `cell` function. Each row of the cell is separated by a semicolon and each element within a row is separated by a comma. In the following example, we create a 2-by-3 cell array `C`, where the first row contains the numbers 1, 2 and 3 and the second row contains a string 'text', a number 4.5 and another string 'more text'.

```
C = {1, 2, 3; 'text', 4.5, 'more text'};
```

Cell arrays might be useful, for instance, in tasks like frequency analysis or word counting. If you want to create a cell array with words from a dataset and their corresponding frequencies, you can use a command like the following:

```
words = {'Monday', 5; 'Tuesday', 3; 'Wednesday', 8};
```

This cell array `words` consists of two columns and three rows, where the first column contains the names of the days of the week “Monday”, “Tuesday” and “Wednesday” and the second column contains corresponding numeric values 5, 3 and 8. You can access the elements of cell arrays using curly braces. For example, `words{1, 1}` would return 'Monday' and `words{2, 2}` would return 3.

Let us create another cell array that contains strings, numbers and other cell arrays to store metadata about documents like document titles, word counts and keywords:

```
textAnalysisData = {'Document1', 1500, {'keyword1', 'keyword2', 'keyword3'};
'Document2', 2000, {'keywordA', 'keywordB'}};
```

This creates a cell array named `textAnalysisData` (cf. Figure 69). Let us assume that this cell array is structured to hold information related to textual analysis. The first row contains 'Document1' (a string representing the name of the first document), 1500 (a numeric value representing the word count or some other metric associated with the first document) and {'keyword1', 'keyword2', 'keyword3'} (a cell array containing three keywords related to the first document). The second row contains the string 'Document2', the numeric value 2000 and the cell array {'keywordA', 'keywordB'} with two keywords related to the second document.

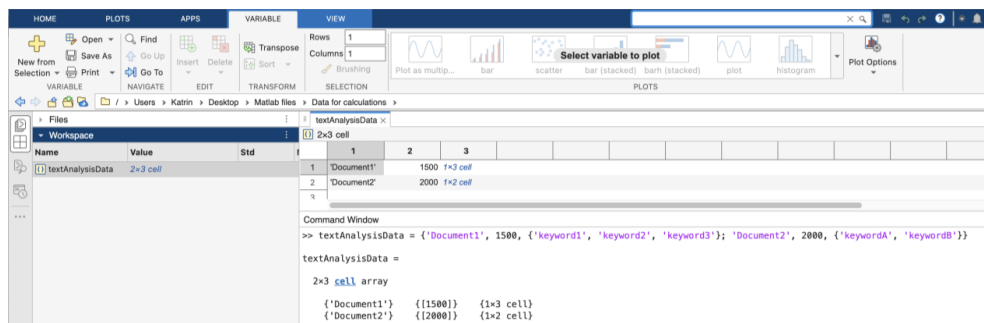


Figure 69: Creating a cell array

To create a cell array in MATLAB using the `cell` function, you can follow this example. Create a cell array of empty matrices, e.g., a 3-by-4 cell array:

```
C = cell(3, 4);
```

Then you can populate the cell array via commands, e.g.,

```
C{1, 1} = 'Document1';
C{1, 2} = 1500;
C{1, 3} = {'keyword1', 'keyword2', 'keyword3'};
C{2, 1} = 'Document2';
C{2, 2} = 2000;
C{2, 3} = {'keywordA', 'keywordB'};
```

You can use cell arrays to manipulate text data, similar to character and string arrays. Let us have a look at the text that we have already used in Chapter 5.2.2 where we first stored it as a character array:

```
text = 'States Parties recognize the important function performed by the mass
media and shall ensure that the child has access to information and material
from a diversity of national and international sources, especially those
aimed at the promotion of his or her social, spiritual and moral well-being
and physical and mental health.';
```

We have seen in Chapter 5.2.2 that we may use the `strsplit` function to split a text from a character array at whitespace characters and return the result as a cell array of substrings as in:

```
words = strsplit(text)
```

The output of `words` now is a 1×51 cell array: {'States'} {'Parties'} {'recognize'}... It also still contains punctuation marks attached to the words such as {'health.'}.

We can remove punctuation marks with the `erasePunctuation` function (from the Text Analytics Toolbox) as follows:

```
words1 = erasePunctuation(words);
```

Now let us assume we want to calculate the type-token ratio (TTR) of our text, which is calculated by dividing the total number of unique words (types) by the total number of words (tokens). This ratio can be used to analyse the diversity of vocabulary in a text. A higher TTR indicates a richer vocabulary, while a lower TTR suggests more repetition of words.

First, we can count the total number of words extracted and represented in the cell array `words` by using `length`:

```
tokens = length(words);
```

To find the number of unique words in the cell array `words`, use the `unique` function:

```
types = length(unique(words));
```

The values can then be used to calculate the type-token ratio by dividing the number of types by the number of tokens:

```
ttr_text1 = types / tokens;
```

In this example, the calculated TTR is approximately 0.83, derived by dividing 43 unique words (types) by 52 total words (tokens), which reflects a relatively rich and diverse vocabulary in this text. We may summarise the above-mentioned commands in a script (cf. also Chapter 6.2) to automate the process of word extraction and type-token ratio calculation (Figure 70).

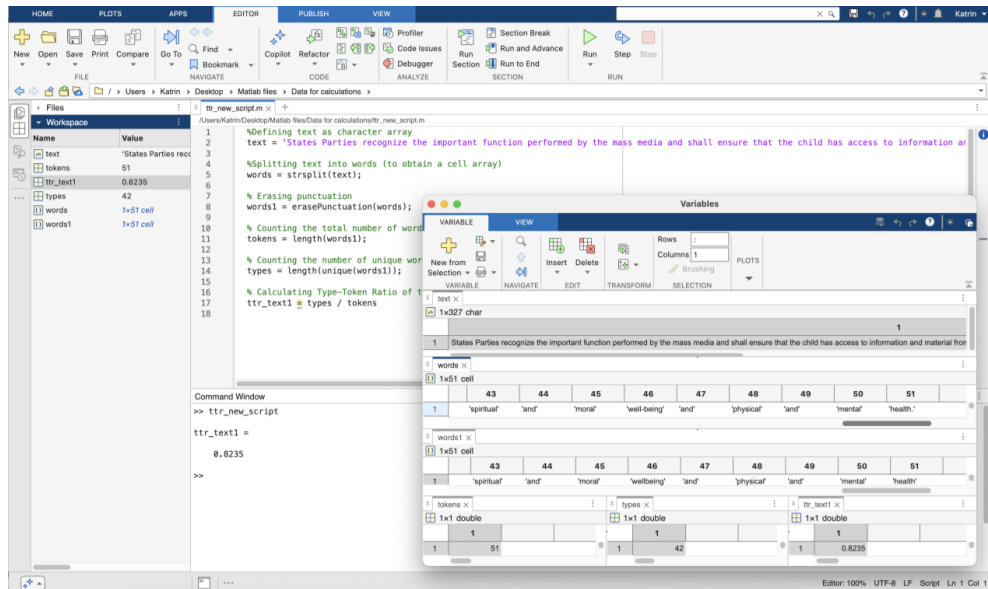


Figure 70: MATLAB script for text preprocessing and type-token ratio calculation

Alternatively, you may work in the above-mentioned example with a string array instead of a character array for the text which can then be split into individual words to return another string array by using the `split` function (cf. Chapter 5.2.3). Character arrays are less flexible for certain operations with textual data. If you use a character array, you may need to convert it into a cell array as we have done here to perform a task which can also be done more directly with string arrays.

For more examples on how to work with cell arrays, you can refer to the MATLAB documentation at [https://www.mathworks.com/help/matlab/matlab\\_prog/create-a-cell-array.html](https://www.mathworks.com/help/matlab/matlab_prog/create-a-cell-array.html) and <https://www.mathworks.com/help/matlab/cell-arrays.html>.

## 5.2.5 Categorical arrays

Categorical arrays in MATLAB are designed to handle data that consists of distinct categories or groups. These arrays are ideal for storing qualitative information that does not involve continuous or numeric values but rather belongs to a set of predefined categories. Categorical data can also be associated with numeric values in certain contexts. By using categorical arrays, you can store repeated data values more efficiently in terms of memory usage, which is especially useful when dealing with large datasets. Additionally, categorical arrays facilitate comparisons and operations on categories, making it easier to perform tasks such as sorting, grouping or summarising categorical data. Rather than storing each repeated category as a separate entity, MATLAB stores it once and uses an index to refer to it. This is particularly useful when there are a lot of repeating values, such as part-of-speech tags for the words from a text (e.g.,

“NOUN”, “VERB”), text labels or classifications. In sum, categorical arrays offer distinct advantages over other data types like strings, cell arrays or tables in specific situations, particularly when dealing with large datasets that involve repetitive, discrete values.

Categorical arrays can be created by using the `categorical` function. An example of creating a categorical array would be:

```
pos_tags = categorical(["noun", "verb", "noun"]);
```

This creates a categorical array named `pos_tags` that represents part-of-speech tags for a set of words. In this example, the array contains three entries: two noun tags and one verb tag (Figure 71).

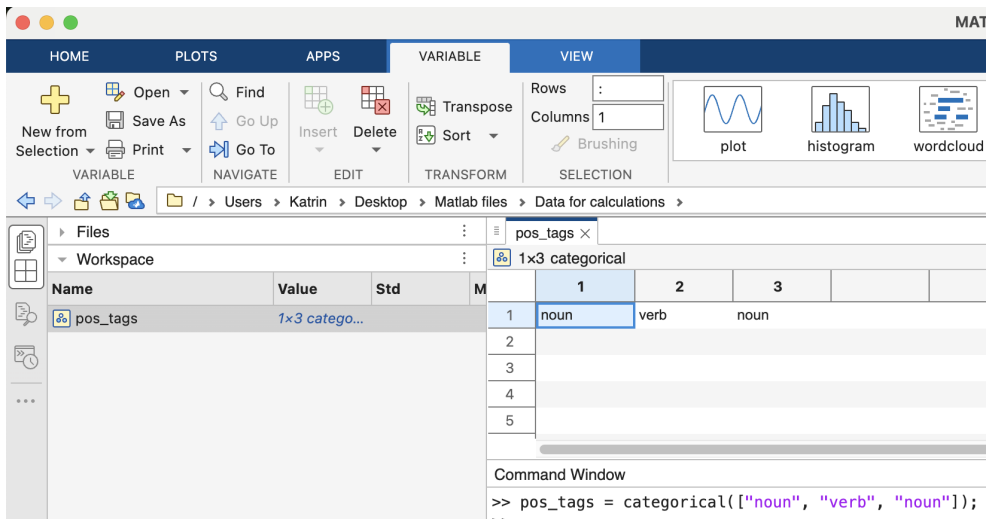


Figure 71: Creating a categorical array for part-of-speech tags for a set of words

When creating a categorical array, you can use either square brackets `[]` or curly braces `{}` depending on the type of input you are providing. Square brackets are used for creating categorical arrays from string arrays or numeric arrays. For example:

```
Tense = categorical(["past", "present", "future", "past", "present",  
"future", "past"]);
```

Curly braces `{}` can be used when you want to create a categorical array from a cell array of character vectors. For example:

```
Tense1 = categorical({'past', 'present', 'future', 'past', 'present',  
'future', 'past'});
```

These examples create categorical arrays `Tense` and `Tense1` that represent the verb tenses occurring in a text (Figure 72).

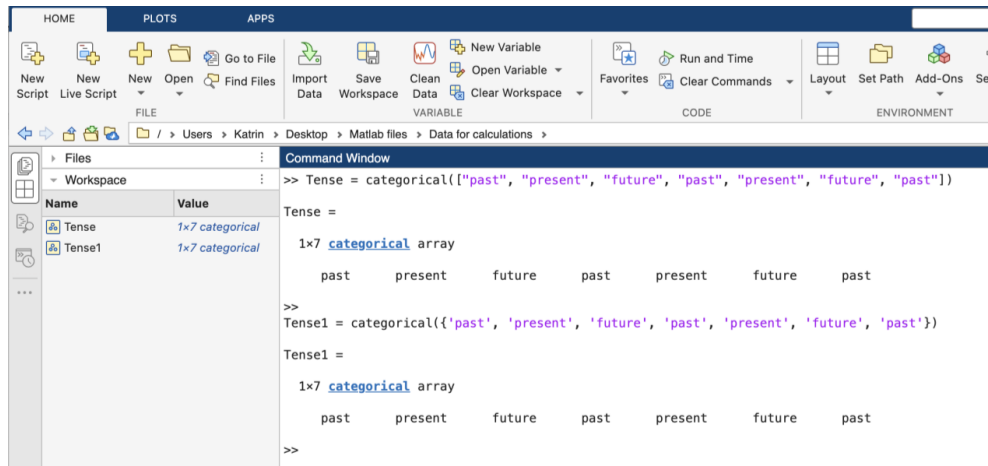


Figure 72: Creating categorical arrays from a string array and from a cell array of character vectors

A specific subtype of categorical arrays in MATLAB are ordinal categorical arrays. They possess a defined mathematical ordering among their categories. This characteristic allows for the comparison of categories using relational operations such as greater than ( $>$ ) and less than ( $<$ ). To create an ordinal categorical array, you can utilise the `categorical` function with the `Ordinal` name-value argument set to `true`, e.g.:

```
text_quality = categorical(["Poor", "Fair", "Good", "Excellent"], 'Ordinal', true);
```

In this example, the categories “Poor”, “Fair”, “Good” and “Excellent” are defined as an ordinal categorical array which is useful when you need to perform operations that rely on the order of these categories and where the ranking of the categories is important.

Let us have a look at another example and assume that you are working with a small corpus of six texts that have been processed through OCR software. Now you want to evaluate the quality of the OCR output and categorise the quality of the OCR in terms of overall accuracy using labels like “Poor”, “Fair”, “Good” and “Excellent”.

To define a set of categories for your data in MATLAB, you can create a categorical array as follows:

```
categories = categorical(["Good", "Fair", "Poor", "Excellent", "Good", "Fair"]);
```

In this example, we directly create a categorical array named `categories` that includes repeated values, i.e., a value for each of the texts in the dataset.

Alternatively, you may want to convert an existing variable from your workspace, such as a string array, into a categorical array. In this case, you might have first created a string array with the quality ratings of the six texts:

```
qualityRatings_OCR = ["Good", "Fair", "Poor", "Excellent", "Good", "Fair"];
```

If we now want to convert the string array to a categorical array, we can use:

```
qualityCategories_OCR = categorical(qualityRatings_OCR);
```

You may also already have defined the categories of OCR quality and their corresponding counts in a string and a numeric variable, such as:

```
categories = ["Poor", "Fair", "Good", "Excellent"];
counts = [10, 24, 78, 45];
```

You can then convert this data into a categorical array that reflects these counts. In order to do so use the `categorical` function along with `repelem` to repeat each category according to its count:

```
OCR_quality = categorical(repelem(categories, counts), categories);
```

The `categorical` function creates an ordinal categorical array from the data. The `repelem` function is used to repeat each element of the `categories` array according to the corresponding values in the counts array. In this code, `repelem` generates an array where “Poor” appears 10 times, “Fair” 24 times, “Good” 78 times and “Excellent” 45 times.

If you want to define the order of the categories as `Poor < Fair < Good < Excellent`, add `'Ordinal', true` and use:

```
OCR_quality = categorical(repelem(categories, counts), categories,
'Ordinal', true);
```

Figure 73 shows the output of the previous steps of converting data from a string and numerical array into a categorical array.

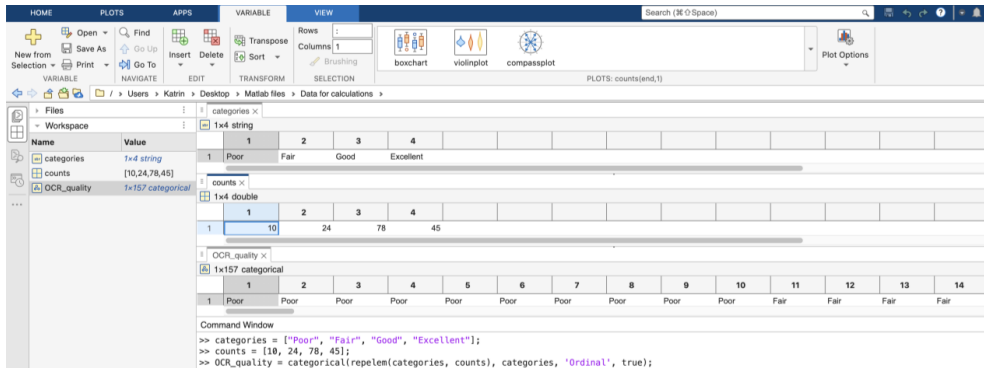


Figure 73: Converting data into a categorical array

For more examples of how to work with categorical arrays in MATLAB, have a look at the MathWorks documentation on this topic at: <https://www.mathworks.com/help/matlab/ref/categorical.html>, <https://www.mathworks.com/help/matlab/ref/categorical.categories.html>, <https://www.mathworks.com/help/matlab/categorical-arrays.html> and [https://www.mathworks.com/help/matlab/matlab\\_prog/ordinal-categorical-arrays.html](https://www.mathworks.com/help/matlab/matlab_prog/ordinal-categorical-arrays.html).

## 5.2.6 Tables

In MATLAB, tables are similar to cell arrays in that they both allow for storing heterogeneous data types, but they offer more structure and functionality for organising and accessing data. While both tables and cell arrays can hold different data types, tables are specifically designed for organising data in a row-and-column format to make it easier to work with datasets that have distinct variables (columns) and observations (rows).

When working with linguistic data, you may have already created tables outside of MATLAB to organise and annotate your dataset. These tables can, for instance, contain information on words, part-of-speech tags, frequency counts or other relevant annotations for your analysis. If these tables have been created in a spreadsheet or another software, you may want to import them into MATLAB for further processing and analysis. Chapter 5.5 will explain the process of importing data such as tables from data files into MATLAB.

To create a table via commands in the Command Window, you can use the `table` function. Each column in the table can represent different types of data, such as numeric values or strings. Let us assume you want to create a table displaying the frequency of the words “unhoused” and “homeless” in the spoken texts from the Corpus of Contemporary American English (COCA, Davies 2008–). For “unhoused” you may find 0.02 occurrences per million words in spoken texts in COCA, and for “homeless”, you may find 23.78 occurrences per million words.

You can first define your variables, e.g., by creating a cell array with the words “unhoused” and “homeless” and a numeric array with the frequencies of these words in spoken texts from COCA per million words. It is common practice to use column vectors for table entries because column vectors represent each variable as a distinct entity (row-wise) in the table. However, MATLAB also supports the use of row vectors for table entries, depending on how the data is structured and the desired output. Here is an example of how to create your variables as column vectors (a column cell array and a column numeric array) where each entry will then correspond to a row in the table:

```
words = {'unhoused'; 'homeless'};
freq_spoken = [0.02; 23.78];
```

Then you can use the following command to create a table with the words and their frequencies (cf. also Figure 74):

```
spokenTable = table(words, freq_spoken, 'VariableNames', {'Word', 'Spoken_Frequency_Per_Million'});
```

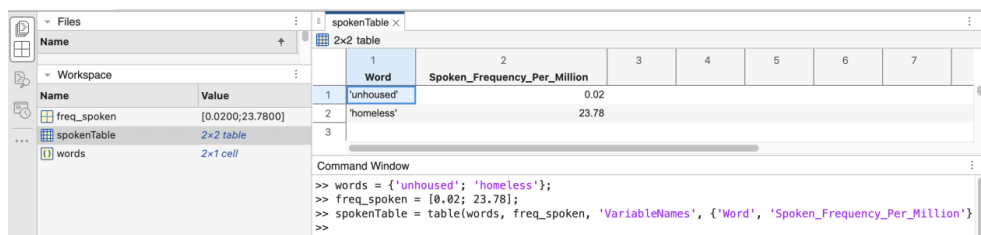


Figure 74: Creating a table via commands

Note that the 'VariableNames' parameter is used to specify the names of the columns in the table.

When using row vectors, all data for a particular variable is stored in a single row in the table. Here is an example of how to create your variables as row vectors:

```
words1 = {'unhoused', 'homeless'};
freq_spoken1 = [0.02, 23.78];
```

We can then create a table to store the words and their frequencies with this command (Figure 75):

```
spokenTable1 = table(words1, freq_spoken1, 'VariableNames', {'Word', 'Spoken_Frequency_Per_Million'});
```

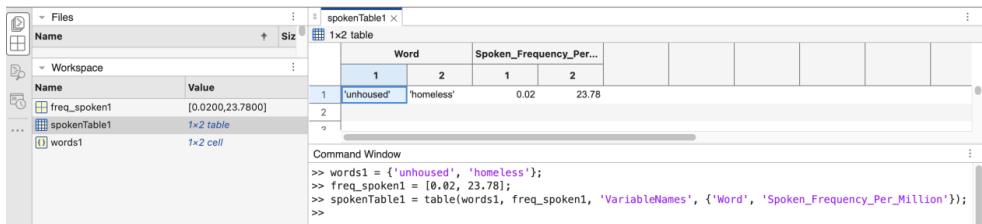


Figure 75: Creating a table using row vectors

Now, compare the layouts of the tables in Figure 74 and Figure 75, which you can see in the top right corners of the figures in the Variables editor. If we want to change the table which we created using row vectors and which you see in Figure 75 to match the layout of the table in Figure 74, we can transpose the variables `words1` and `freq_spoken1` into the appropriate column format for table entries. We may do this by right-clicking on each of these variables in the Variables editor and choosing **Transpose Variable** from the drop-down list (Figure 76).

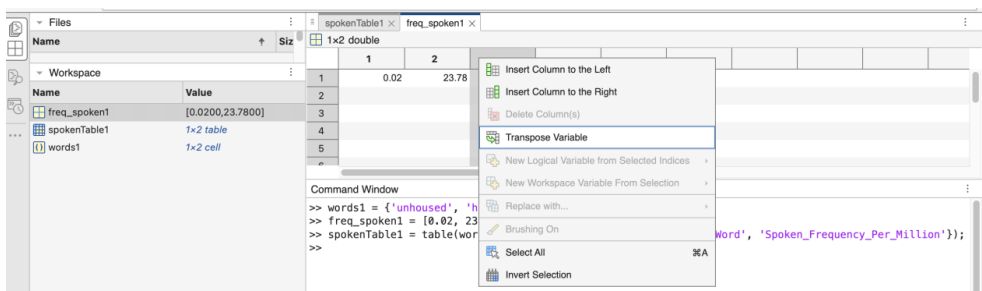


Figure 76: Transposing a variable

Alternatively, we may use the transpose operator (') when using a command such as:

```
spokenTable2 = table(words1', freq_spoken1', 'VariableNames', {'Word', 'Spoken_Frequency_Per_Million'});
```

In this command, the single quote (') operator is used to transpose the row vectors `words1` and `freq_spoken1` into column vectors before creating the table. If you have a row vector, applying

' will convert it to a column vector and if you have a column vector, applying ' will convert it to a row vector. The following screenshot shows the result of this command (Figure 77).

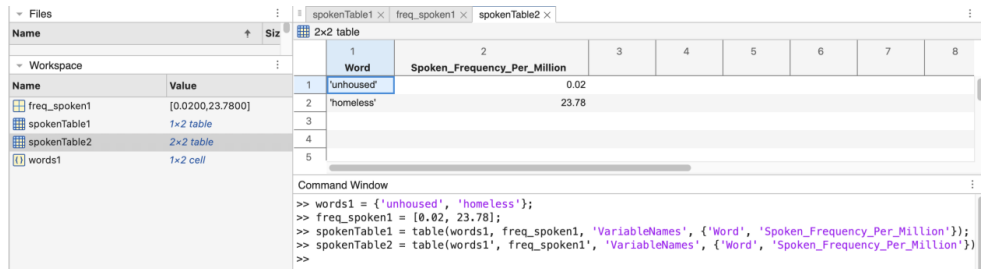


Figure 77: Resulting table after transposing row vectors

In this example, we initially defined `words` and `freq_spoken` as column vectors using `{'unhoused'; 'homeless'}` and `[0.02; 23.78]`, which required transposing them into row vectors (`'`) to create the table `spokenTable2` with our desired layout.

There are other much easier ways to create tables in MATLAB, e.g., by using the **Import** tool to import data directly from a spreadsheet (cf. Chapter 5.5).

For more examples of how to work with tables in MATLAB, have a look at the MathWorks documentation on tables at <https://www.mathworks.com/help/matlab/tables.html> and <https://www.mathworks.com/help/matlab/ref/table.html>.

## 5.2.7 Structures

Structures in MATLAB allow you to group different types of data under one variable, with each piece of data stored in a field that can be accessed by name. They are useful for managing complex or varied data, but this book will primarily focus on simpler data types. Therefore, only a brief overview of how to create structures is provided here to give you a basic understanding. You can create a structure variable using the `struct` function or by directly assigning values to fields. A structure can contain multiple fields, and each field can hold different types of data such as numeric arrays, strings, cell arrays and even other structures.

If you use the `struct` function, you can either create an empty structure or a structure with specified fields and values. `s = struct(field, value)`, for instance, creates a structure with one field and its corresponding value. `s = struct(field1, value1, ..., fieldN, valueN)` creates a structure with multiple fields, each associated with a specified value.

Let us look at a simple example of how you can use a structure to organise data from a collection of news reports, e.g., from the Corpus of News on the Web (NOW, Davies 2016–) from the English-Corpora.org website. The fields in the structure can hold different types of information related to your collection of news reports, such as the title, date and text content of each article. Let us assume you have collected the following data from texts on IT news in the NOW corpus:

*Text title 1: AiThORITY.com Joins the Digital Transformation Forum 2022 as Media Partner*  
*Content text 1: IT News Africa and industry experts are set to come together*  
*Date text 1: 22-05-17*

*Text title 2: Why the Tinder Swindler was actually a win for Tinder*

*Content text 2: Singapore's head of digital, Julian Chow, said the platform could further boost user confidence*

*Date text 2: 22-02-10*

Now we create a structure with the following commands:

```
newsArticles(1) = struct( ...
    'Title', "AiThORITY.com Joins the Digital Transformation Forum 2022 as
Media Partner", ...
    'Content', "IT News Africa and industry experts are set to come together",
    ...
    'Date', '22-05-17' ...
);

newsArticles(2) = struct( ...
    'Title', "Why The Tinder Swindler was actually a win for Tinder", ...
    'Content', "Singapore's head of digital, Julian Chow, said the platform
could further boost user confidence", ...
    'Date', '22-02-10' ...
);
```

Note that the ellipsis (...) is used to indicate that a statement or expression continues on the next line. It is a way of breaking up long lines of code into multiple lines for better readability, especially when you have many inputs or fields in a function (like `struct`). After entering these commands, the Title, Content and Date fields hold the respective data you provided, in this case as string arrays and character vectors. The Title and Content fields are defined using double quotes ("), so they are string arrays. The Date field is defined using single quotes ('), so it is a character vector in this example, cf. Figure 78.

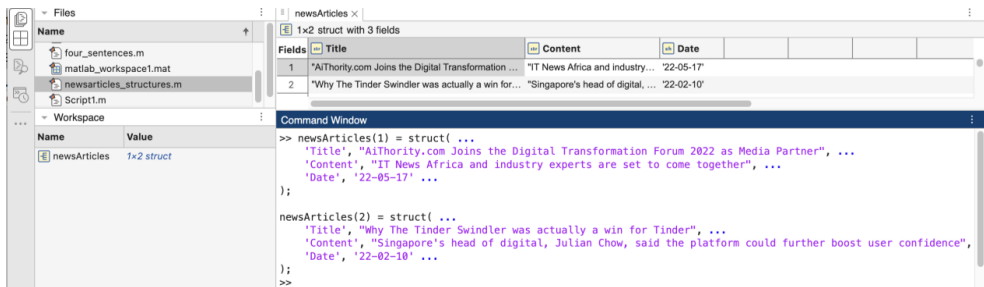


Figure 78: Creating a structure

For more examples of how to work with structures in MATLAB, have a look at the MathWorks documentation on structures at <https://www.mathworks.com/help/matlab/structures.html> and <https://www.mathworks.com/help/matlab/ref/struct.html>.

### 5.2.8 Dictionaries

In MATLAB, a dictionary is a data structure that allows you to map keys to values. It is particularly useful for tasks where you need to look up values based on unique keys. You may, for instance, create a dictionary in MATLAB for a translation memory that uses sentences as strings and allows you to look up segments containing a particular word. The dictionary supports various data types for both keys and values. You can create a dictionary using the `dictionary` function. Here is a simple example: First, we create keys (e.g., English expressions) and corresponding values (e.g., French translations):

```
EnglishExpressions = ["Hello", "Goodbye", "Thank you"];
FrenchTranslations = ["Bonjour", "Au revoir", "Merci"];
```

Then we create our small bilingual dictionary (Figure 79) with the command:

```
bilingualDict = dictionary(EnglishExpressions, FrenchTranslations);
```

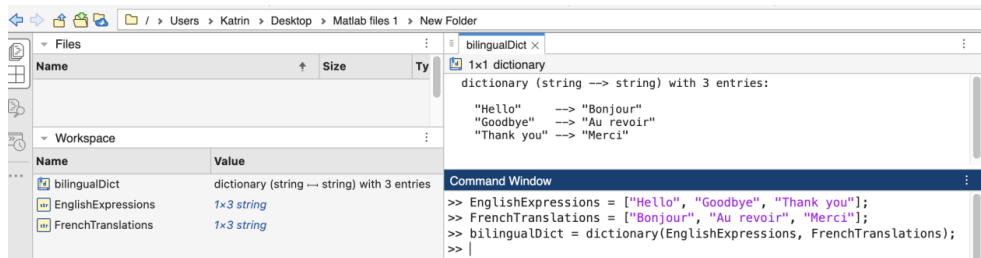


Figure 79: Creating a bilingual dictionary in MATLAB

To modify an entry in the dictionary, you can use a command as follows:

```
bilingualDict("Goodbye") = "Adieu";
```

This changes “Au revoir” as the previous French translation of “Goodbye” in the data to “Adieu” (Figure 80).

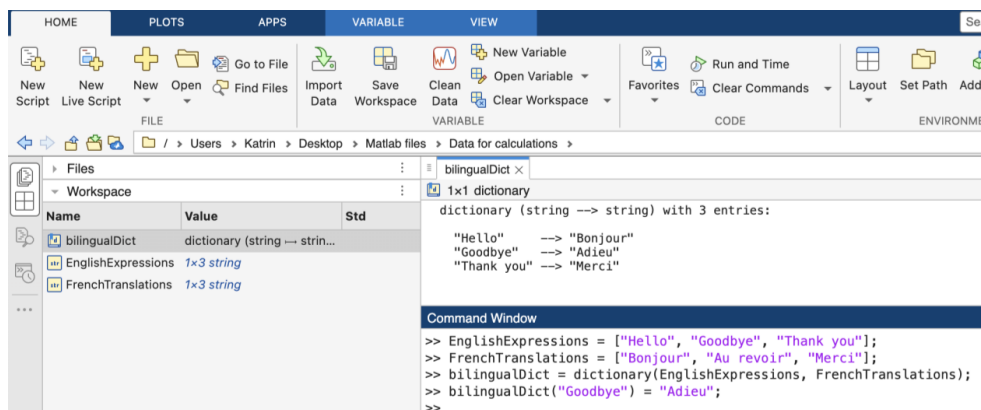


Figure 80: Modifying an entry from a dictionary

To delete an entry from a dictionary, you can use the `remove` function. If you want to remove the entry and the key for “Goodbye” from `bilingualDict` and store a new version of the dictionary as `bilingualDict1`, use:

```
bilingualDict1 = remove(bilingualDict, "Goodbye");
```

Keep in mind that using very large dictionaries in MATLAB can have performance implications, particularly regarding memory usage and lookup speed. For more examples of how to work with dictionaries in MATLAB, have a look at the MathWorks documentation at: <https://www.mathworks.com/help/matlab/ref/dictionary.html> and [https://www.mathworks.com/help/matlab/matlab\\_prog/getting-started-with-dictionaries.html](https://www.mathworks.com/help/matlab/matlab_prog/getting-started-with-dictionaries.html).

### 5.3 Creating variables via the ‘New Variable’ option

You can create numeric variables by clicking on **New Variable** in the dropdown list under **Variable** in the MATLAB **Home** tab toolbar (Figure 81).

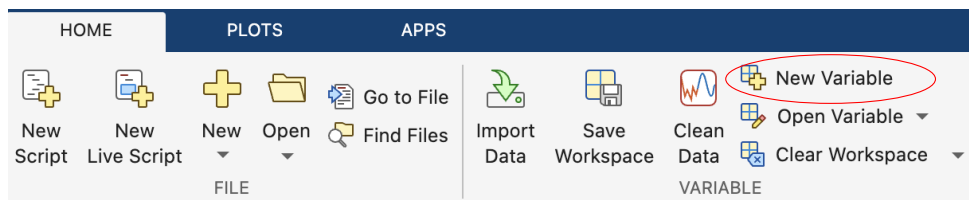


Figure 81: Creating a new variable via the ‘New Variable’ icon from the toolbar

Alternatively, you can create a new variable directly in the Workspace panel. To do this, right-click in the panel, and the option **New** will appear in the dropdown menu.

With either method, a new variable will appear with the name `unnamed` in the workspace and in the Variables editor. You can change its name in the workspace by right-clicking on the variable name and selecting **Rename**. Then type the new variable name and press Enter.

After creating a variable in one of these ways, you can enter your values manually by typing them into the respective rows in the Variables editor in MATLAB. There is also the possibility to copy your values from a file, for instance, from an Excel or CSV file (press Ctrl + C on Windows or Cmd + C on macOS) and paste them there into your variable in the MATLAB Variables editor (Ctrl + V on Windows or Cmd + V on macOS), cf. Chapter 3.7.

Suppose you want to create a numeric variable representing the sizes of the corpus sections for spoken, fictional, magazine, newspaper and academic texts from the British National Corpus (BNC, Davies 2004) available on the English-Corpora.org website. After looking up a word in the corpus using the **Chart** option, you can see the total frequency of the word in each section, as well as the size of each corpus section in millions of words and the normalised frequency of the word per million words. In this case, we want to use the size values from the row that provides information on the size of the corpus sections (Figure 82).

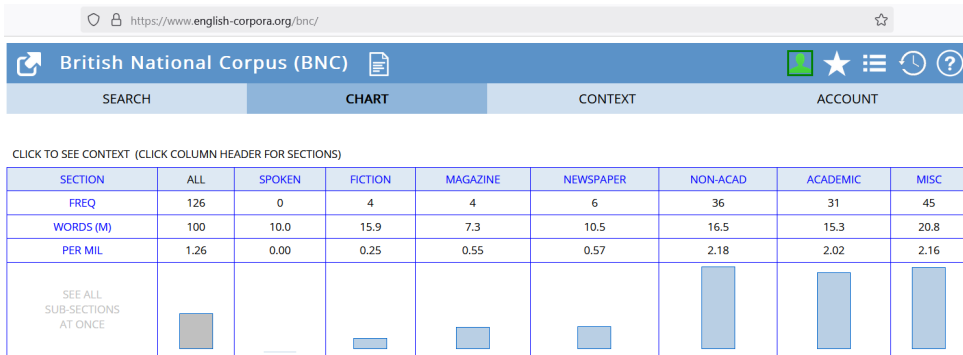


Figure 82: Size of corpus sections in BNC

You can assign a name to the variable and manually input the values we are interested in, such as 10.0, 15.9, 7.3, 10.5 and 15.3, into the cells to form a column (Figure 83).

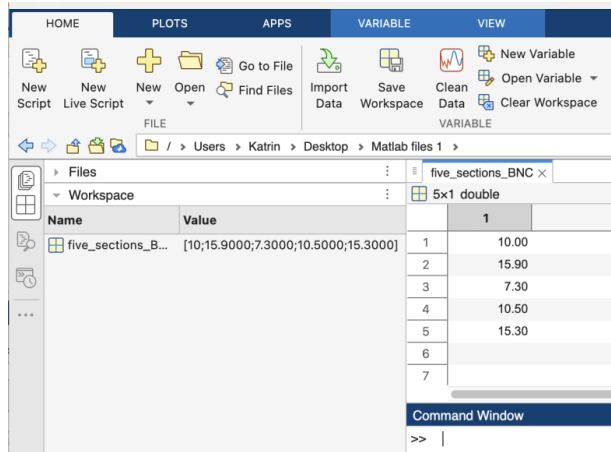


Figure 83: MATLAB variable for size of BNC sections in millions of words

These values will now be displayed in MATLAB in a specific format depending on your selected variable preferences in the settings, e.g., as 10, 15.9000 etc. or 10.00, 15.90 etc. Regardless of the formatting, the numbers now represent values in millions, consistent with how they appear in the BNC dataset (e.g., 10 means 10 million, 15.9000 means 15.9 million). If you intend to work with these values as millions, you can leave them as they are. This can be particularly useful when comparing them with other datasets that also use millions as units or when the scale (i.e., working in millions rather than absolute values) is relevant for your analysis, reporting or visualisation.

However, to make these values more practical for use in MATLAB and to prevent confusion, it is generally recommended to store them in their full numeric form (i.e., storing 10.0 million as 10000000). This approach ensures that all calculations, visualisations and analyses are conducted with the correct values, as MATLAB may otherwise treat the numbers in a format that does not match their intended scale. For instance, a value of 10.0 could be misinterpreted as simply 10, rather than 10 million.

You can save the size values for the corpus sections in a spreadsheet, then copy and paste the entire column into MATLAB using Ctrl + C and Ctrl + V on Windows or Cmd + C and Cmd + V on macOS. When you copy and paste data from available tables, keep in mind that we do not need commas used as thousand separators here (e.g., 10,000,000) as they may cause MATLAB to misinterpret the values as text and trigger an error message.

## 5.4 Viewing and modifying variables by using the Variables editor

We have already seen in Chapter 3.7 that we can view and edit variables in the **Variables editor**. Double-click on a variable name in your Workspace panel and the Variables editor will open for the selected variable so that you can view the content of the variables in the editor. It displays the data in a readable tabular format. You can modify the values of your variables directly in this editor. For example, if you have a numeric array, you can change individual values within the array without needing to use code. You can also rename your variables by clicking on a variable's name in the Variables editor and changing it.

Let us now assume that you have looked up the absolute and normalised frequencies for the word “whom” in different sections of the British National Corpus (BNC, Davies 2004) and you have created a table for this data in MATLAB. When you click on your variable name `Whom_in_BNC` in the workspace you will see your table in the Variables editor. You can directly delete or overwrite individual values in the table if you want to, for instance, if you later realise that there was an error in how you recorded the frequencies of “whom” for the section of spoken texts in the BNC.

You will find options such as deleting or sorting selected columns in the Variable tab from the toolstrip. Additionally, you can right-click on selected data from your variable in the editor to see options such as sorting your data or deleting selected values (Figure 84).

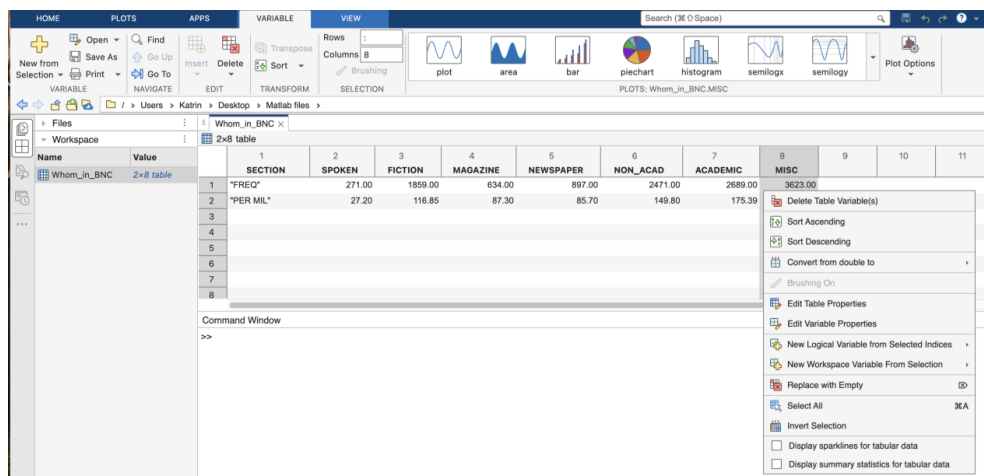


Figure 84: Viewing and editing data in the Variables editor

The Variables editor typically opens one variable at a time, but you also can open several windows in the Variables editor for different variables. The easiest way to do so is by using the

drop-down menu in the Variables editor which you can access by clicking on the three dots (:) in the top right corner your variable. Click on this menu to access options for arranging the editor windows such as **Tile All** (Figure 85).

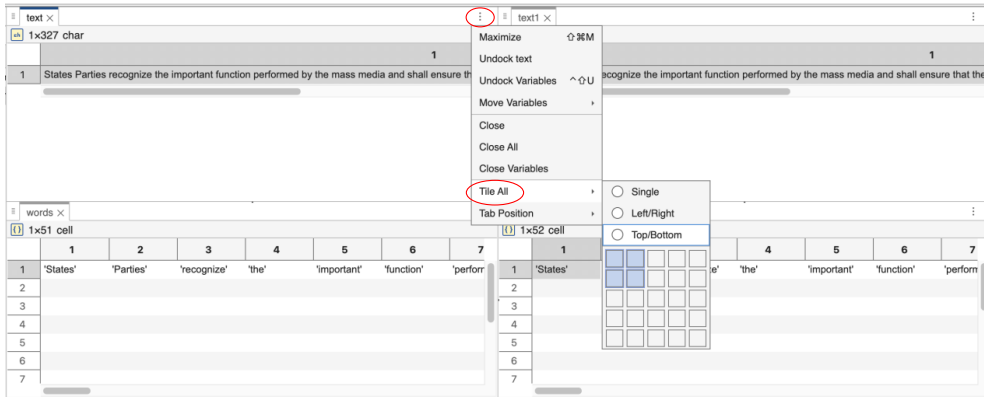


Figure 85: Opening several windows in the Variables editor for different variables

## 5.5 Importing variables from data files

To import data into MATLAB, click the **Import Data** button on the **Home** tab (Figure 86). This will open a file explorer where you can select the file you want to import data from. Ensure that the file is saved in your current working folder. You can choose from various file types such as CSV, Excel files or text files.

For example, if you have created an Excel file containing data on the size of different corpus sections (such as spoken, fictional, magazine, newspaper and academic texts from the BNC, cf. the example in Chapter 5.3), navigate to that file in the explorer and double-click on it. Once the file is opened, you can choose the data you want to import. You can also specify the output type (e.g., a numeric matrix as in Figure 86 or a table as in Figure 87) and assign names to the variables, e.g., **Register** and **Size** for the respective table columns and `Table1_BNC` as the name of the table itself (Figure 87). By default, MATLAB suggests variable names based on the file names, but you can modify them during the import process if needed.

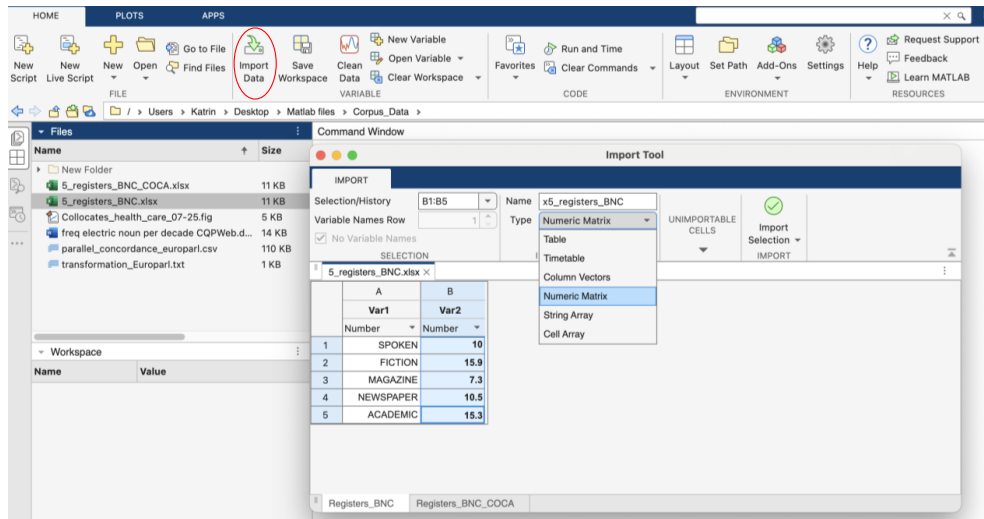


Figure 86: Importing numeric data into MATLAB from an Excel file

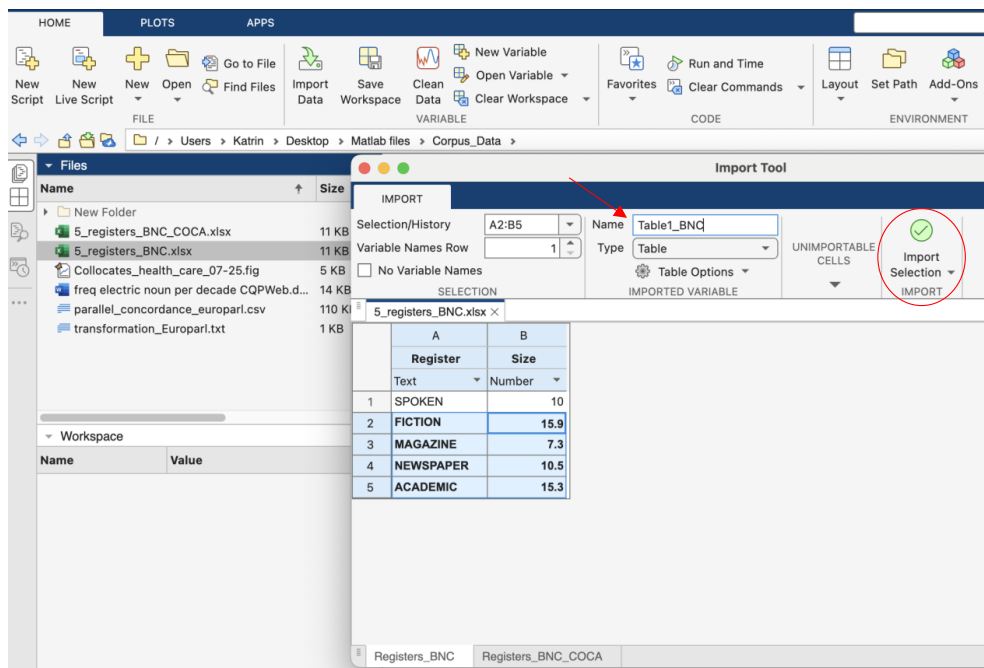


Figure 87: Importing data from an Excel file in the form of a table and assigning it a name

After selecting the output type and setting the variable name, click **Import Selection** (cf. Figure 87). The new variable will then appear in your workspace. You can also still change the variable name in the workspace after the import is complete. If your Excel file has multiple sheets, the Import tool allows you to choose which sheet you want to import data from.

You can hold down the Ctrl (or Cmd) key on your keyboard and click on individual columns or data to select multiple sections at once. You can also click and drag to highlight a range of cells or columns that you want to import (Figure 88).

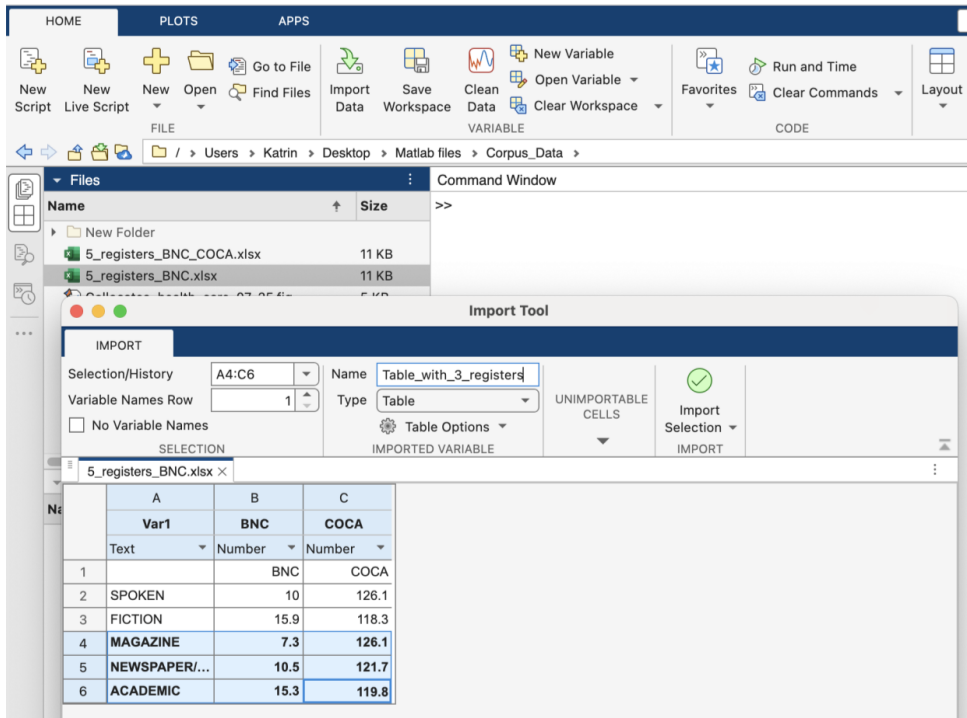


Figure 88: Importing selected data from an Excel file containing several tabs

Let us consider an example using the *Europarl spoken parallel – English* corpus (Koehn 2005), which is available for free through Sketch Engine.<sup>16</sup> The Europarl corpus is derived from the proceedings of the European Parliament and contains sentence-aligned translations in multiple languages, including English, French and Spanish, in the open version available on the Sketch Engine platform. This means that in this parallel corpus, each sentence or segment in one language (e.g., English) is paired with its corresponding sentence or segment in another language (e.g., French). It is important to note that when searching the English text, the sentences might not always be the original speeches, and the aligned French sentences may not always be direct translations. To identify whether a sentence is an original text or a translation, you can refer to the metadata, which includes information such as the speaker’s name. This allows you to determine the context of the texts you are working with.

Let us assume we are interested in English sentences from the Europarl corpus that contain the word “transformation” and we want to examine these sentences alongside their corresponding segments in French. We query the word “transformation” in the English section of the Europarl corpus and we use the parallel concordance view to display the aligned French sentences that correspond to the English sentences (Figure 89).

<sup>16</sup> [https://app.sketchengine.eu/#dashboard?corpname=preloaded%2Feuroparl7\\_filtered\\_en](https://app.sketchengine.eu/#dashboard?corpname=preloaded%2Feuroparl7_filtered_en)

Figure 89: Parallel concordance view in Europarl, hosted on Sketch Engine

Next, we download the query results from the parallel concordance by clicking the **Download** button from the options available on the top right and save them, for instance, in a text file or a CSV file (Figure 90).

Figure 90: Downloading query results from parallel concordance in Europarl on Sketch Engine

Then we import our data or selected segments and their respective aligned segments into MATLAB, e.g., as strings or in the form of a table (Figure 91). Once the data is imported into MATLAB, we can perform further analysis.

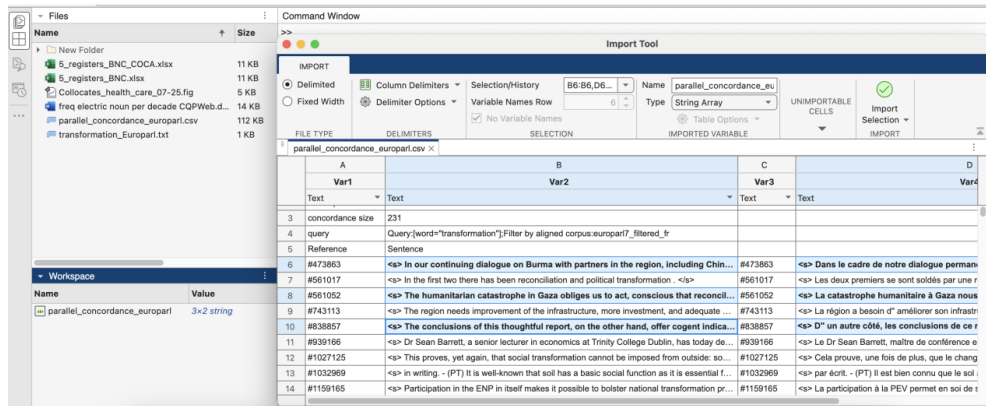


Figure 91: Importing selected data into a MATLAB string array from a CSV file

For further information on importing and selecting data interactively in MATLAB, you may refer to several resources from the official MATLAB documentation. For example, the following link provides a general overview of the Import Tool: <https://mathworks.com/help/matlab/ref/importtool.html>.

Under the following link, you can also find guidance on how to select and import data from spreadsheets interactively: [https://www.mathworks.com/help/matlab/import\\_export/select-spreadsheet-data-interactively.html](https://www.mathworks.com/help/matlab/import_export/select-spreadsheet-data-interactively.html).

Additionally, you may find the MathWorks tutorial videos helpful. The following video demonstrates how to import data from spreadsheets and text files: <https://www.mathworks.com/videos/importing-data-from-text-files-interactively-71076.html>. Another video focuses particularly on importing data from text files: <https://www.mathworks.com/videos/import-tool-enhancements-for-text-files-70406.html>.

These resources and videos will guide you through the process of importing data into MATLAB and offer various examples.

Additionally, there are various ways of importing data into MATLAB via commands. For, instance, the `fileread` function can be used to read the contents of a file and return it as a character vector (<https://www.mathworks.com/help/matlab/ref/fileread.html>). This can be particularly useful for loading text data from files for further processing or analysis. For example, to read the contents of a text file, such as the downloaded novel *The sea wolf* by Jack London (1904) from the Oxford Text Archive (OTA identifier: ota:3045, cf. <https://ota.bodleian.ox.ac.uk/repository/xmlui/handle/20.500.12024/3045>), you can use the following command when the file is saved in your current working directory (e.g., as `Sea_Wolf.txt`), cf. Figure 92:

```
fileContent_Sea_Wolf = fileread('Sea_Wolf.txt')
```

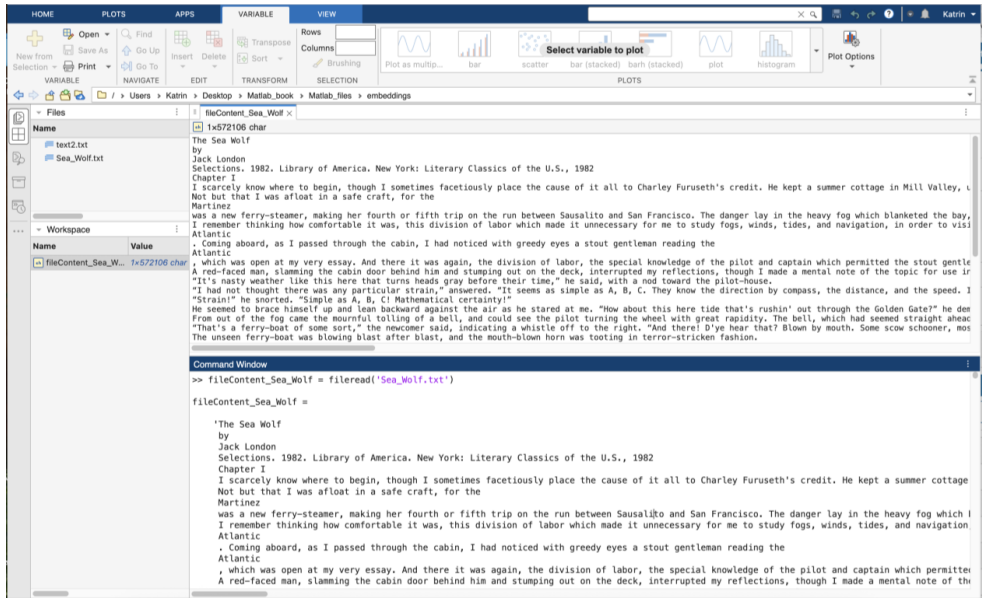


Figure 92: Using the `fileread` function to read the contents of a text file

You can specify the encoding format for reading the file, e.g., to read a file encoded in UTF-8, you can use:

```
fileContent_UTF8 = fileread('example.txt', 'UTF-8');
```

To import text from CSV and Microsoft Excel files, use `readtable` (cf. <https://www.mathworks.com/help/matlab/ref/readtable.html>).

In Chapter 10.2, we will explore in more detail how to extract text data from files, particularly focusing on functions from the Text Analytics Toolbox, which provide advanced capabilities for processing and analysing textual data.

## 5.6 Saving and loading workspace variables

You can save variables stored in your current workspace to a `.mat` file for later use. This allows you to avoid recreating or recalculating them each time you start MATLAB. There are several ways to save variables. We have already seen in Chapter 3.6 that we can save all current workspace variables by clicking on **Save Workspace** in the **Variable** section on the **Home** tab. Alternatively, you can use the Context Menu in the Workspace panel to save specific variables or the entire workspace with all variables.

Another option would be to use the `save` command in the Command Window. For instance, you can use the command `save('Saved_workspace_1.mat')` where the file name is defined within the quotation marks. This will save all variables in your current workspace to a file named `Saved_workspace_1.mat`. You will see the file then in the Files panel, from which it can be loaded back into MATLAB in a future session (Figure 93).

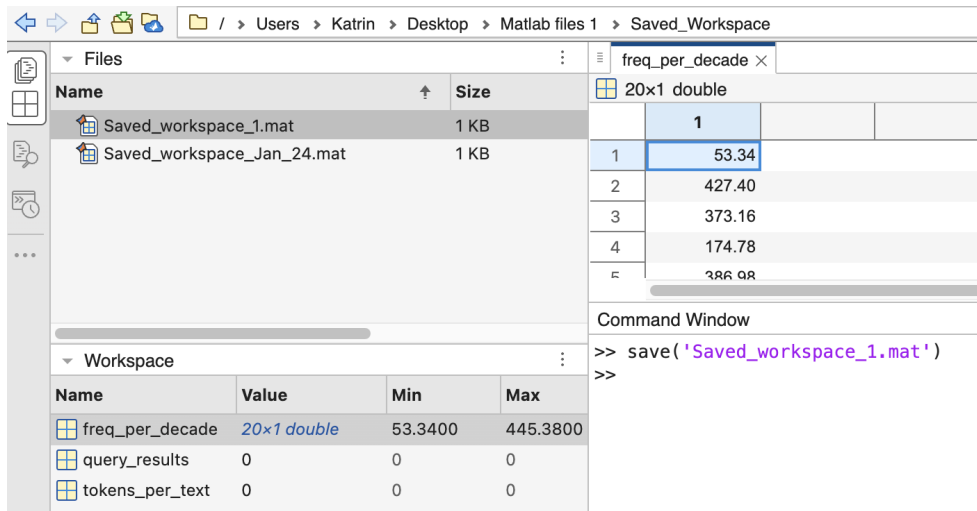


Figure 93: Saving workspace variables via the save command

If you want to save specific variables for later, you can specify them by name. For example, to save variables  $X$  and  $Y$ , in a file named `workspace_data` use:

```
save('workspace_data.mat', 'X', 'Y');
```

In our example shown in Figure 93, we may, for instance, want to save only the two variables `query_results` and `tokens_per_text` from the workspace. Here, you can use a command such as the following:

```
save('2_Workspace_Variables.mat', 'query_results', 'tokens_per_text');
```

Alternatively, you can delete irrelevant variables from the current workspace (e.g., by clicking on them in the and pressing the **Delete** key) and then save your entire workspace as described above.

It goes without saying that variable names and file names should be meaningful. They should be concise yet descriptive enough to help you remember their purpose and contents later on. For more examples of saving variables have a look at the MATLAB documentation at: <https://www.mathworks.com/help/matlab/ref/save.html> and [https://www.mathworks.com/help/matlab/matlab\\_env/save-load-and-delete-workspace-variables.html](https://www.mathworks.com/help/matlab/matlab_env/save-load-and-delete-workspace-variables.html).



# 6 Basic concepts of MATLAB® scripts

## 6.1 Overview

The main goal of this chapter is to show how scripts can streamline your workflow by automating repetitive tasks and executing commands in sequence. We will focus on using MATLAB scripts to save and run these sequences. While there are important programming concepts, such as control structures, that help manage the flow of your code, these structures will not be featured in the example scripts provided in this book. For context, a brief explanation of these concepts will be provided here. Control structures are mechanisms that determine the order in which parts of a programme are executed. A common example is the `if/else` statement, which is a type of conditional control structure that enables the script to make decisions and execute different blocks of code based on certain conditions. Loops, such as `for` and `while` loops, allow you to repeat a set of commands multiple times. Although these concepts – control structures and loops – are fundamental to many programming tasks, they are not required for the examples in this book, which concentrate on some of the basics of writing and running simple scripts in MATLAB. Likewise, debugging and error handling techniques will normally not be needed for the short scripts presented here, and we will touch upon these topics only to a limited extent. If you are interested in learning more about programming concepts, you can, for instance, explore the following online MATLAB courses:

- *Programming in MATLAB: Expand your knowledge of programming by learning more about writing scripts, managing variables and performing fundamental operation:*  
<https://matlabacademy.mathworks.com/details/programming-in-matlab/lpmlprn>
- *Programming constructs: Explore essential control structures such as loops and conditional statements to build more advanced scripts:*  
<https://matlabacademy.mathworks.com/details/programming-constructs/otmlprc>
- *Debugging and error handling: Understand how to identify, troubleshoot and correct errors in your MATLAB code:*  
<https://matlabacademy.mathworks.com/details/debugging-and-error-handling/otmldeh>

MATLAB scripts are plain text files with a `.m` file extension that typically contain a sequence of commands. The commands from a script are executed in sequence when the script is run in MATLAB. Scripts operate directly in the workspace and any variables created or modified within the script are available after execution. Scripts are especially useful for tasks that you would otherwise need to perform repeatedly from the command line or for a set of commands you want to reuse. For example, scripts are ideal when creating specific types of plots with consistent settings and predefined adjustments or when defining variables that require several steps to calculate. By organising and saving the respective commands in a script, you can ensure that tasks are performed consistently.

In general, when writing scripts, there are a few best practices to keep in mind. First, of course, it is important to use meaningful variable names. Avoid using vague names like `a` or `results` unless they are self-explanatory in context. When you choose descriptive names for your variables, this will improve clarity. It is also often useful to use comments throughout your script to explain the purpose of your code. Comments, denoted by the percentage sign in MATLAB, help you and others understand and remember what each section of the script does. Anything

written after the `%` symbol on a line is ignored by MATLAB during execution. For example, before a block of code, you may write:

```
% This section creates a table with data from the COCA corpus.
```

It is important to save your scripts regularly to avoid losing any work in case something unexpected happens. When you are working on a bigger project, using version control is a helpful way to keep track of changes you make over time. Version control lets you save different versions of your script so that you can revisit an earlier version if you want to review or restore previous work.

When you want to create a script in MATLAB, click the **New Script** button on the **Home** tab (Figure 94). This will open an empty file named “untitled” in the MATLAB Editor in which you can write, save and run scripts.

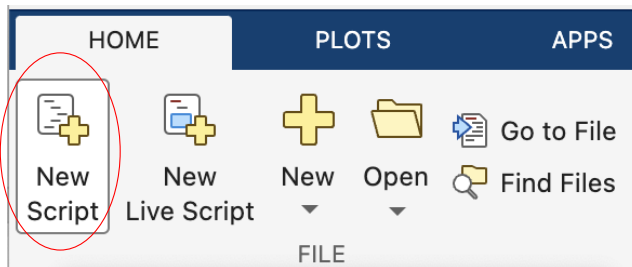


Figure 94: Creating a new script

Alternatively, you can click on the **Script** button in the **Editor** tab in the drop-down menu under **New** (Figure 95).

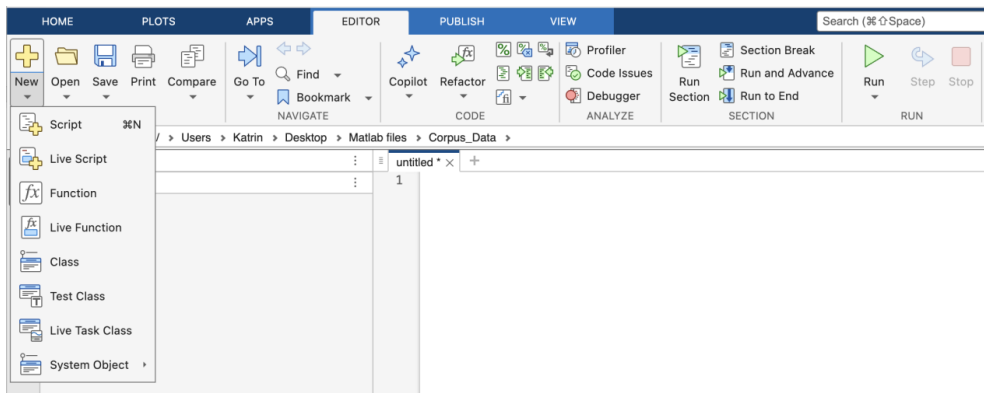


Figure 95: Creating a new script via the Editor tab

Within the MATLAB Editor, you will find various functions on the toolbar to help you work with your script. For example, you can easily save your script under a specific name (Figure 96).

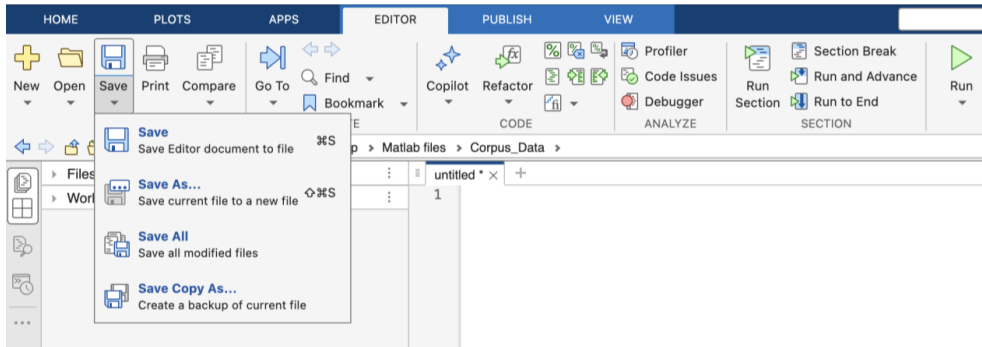


Figure 96: Saving a script

When you have opened a new script, add your code and save it. You can then run your script directly from the editor by clicking the **Run** button, which is located on the right side of the toolstrip (Figure 96). This will execute all the commands in your script in order.

Another option for running your script is to type its name without the `.m` extension in the command line.

For example, if your script is named `Script1.m`, you can run it by typing:

```
Script1
```

Alternatively, you can use the `run` function and specify the script name including the file extension as a string.

```
run('Script1.m')
```

In summary, while MATLAB scripts may not always be necessary for simple tasks, they can be highly effective for streamlining workflows by organising commands into a single file that can be executed repeatedly. In the next section, we will explore some practical examples of small scripts. In Chapter 6.3, we will also have a look at live scripts (`.mlx` files).

## 6.2 Writing and running MATLAB® scripts

In this section, small examples of MATLAB scripts will be provided, demonstrating how they can be useful for basic tasks in corpus linguistics. These examples will include simple scripts for creating variables or generating specific types of plots. More examples of scripts will be provided in some of the following chapters, where they will be applied to specific tasks and concepts as needed.

In MATLAB, standard syntax highlighting colours help make scripts more readable and easier to debug. By default, comments in scripts, for instance, appear in green, strings in purple and syntax errors in red. These colour conventions are also applied in the examples of scripts in this book. You can also customise the colours used for syntax highlighting in scripts (<https://www.mathworks.com/help/matlab/ref/matlab.colors-settings.html>).

In the first example, we will use a script to create a table. In Chapter 5.2.6 we already created a table by executing a series of commands in the Command Window. An alternative approach is to write this sequence of commands within a script. This method allows you to save the entire process in a reusable manner. In our example from Chapter 5.2.6, we wanted to create a table displaying the frequency of the words “unhoused” and “homeless” in the spoken texts from the Corpus of Contemporary American English (COCA). A small script for creating and displaying such a table might look like this:

```
words = {'unhoused'; 'homeless'};

freq_spoken = [0.02; 23.78];

spokenTable = table(words, freq_spoken, 'VariableNames', {'Word',
'Spoken_Frequency_Per_Million'});

disp(spokenTable);
```

Without comments, the script works fine, but it may be harder for someone else – or even yourself, after some time – to understand what each line of code is doing. So, while the script runs without issues, it is always better to provide a bit of explanation, especially when the code might be shared or reused later. Here is another version of the script, with comments explaining each part:

```
% Script to create a table of word frequencies for "unhoused" and
% "homeless" in the spoken texts from the Corpus of Contemporary American
% English (COCA)
% Date: February, 2025
% -----
% Step 1: Define the words we are analysing and their corresponding
% frequencies per million in spoken texts. The values have been retrieved
% from the COCA corpus.

words = {'unhoused'; 'homeless'};

freq_spoken = [0.02; 23.78];

% Step 2: Create a table with the name "spokenTable" to organise the word-
% frequency data. The table will have one column for the words and one for %
% their frequencies. We use the 'VariableNames' property to give meaningful %
% names to the columns.

spokenTable = table(words, freq_spoken, 'VariableNames', {'Word',
'Spoken_Frequency_Per_Million'});

% Step 3: Display the table in the Command Window.

disp(spokenTable);
```

When the script above is run in MATLAB, the variables will appear in the workspace and can be opened in the Variables editor. The output will also be displayed in the Command Window (Figure 97).

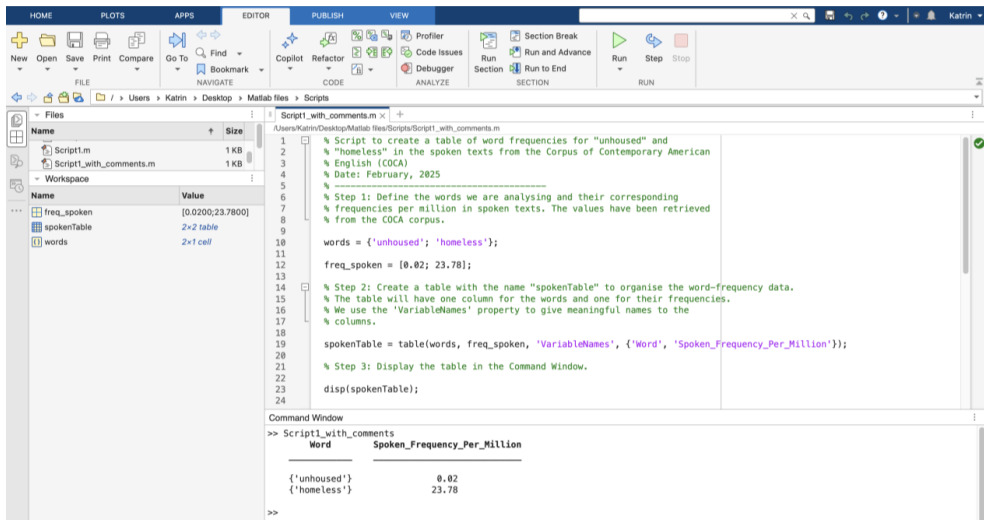


Figure 97: Running a MATLAB script

In Figure 97, you can see that the syntax highlighting in the script editor uses different colours to visually differentiate between various types of code elements, making the code easier to read, e.g., green typically represents comments as mentioned above.

You can divide your script into sections and run them individually or all at once. This is particularly useful for testing parts of your code. To create distinct sections, start a line with a double percent sign (%%). This will allow you to organise your script into logical blocks and makes it easier to execute and debug specific sections of code without running the entire script. To create a section for defining just the two variables `words` and `freq_spoken` first, you would insert a double percent sign before that step. Here is how you can structure it:

```

%% Section 1: Define variables
words = {'unhoused', 'homeless'};
freq_spoken = [0.02, 23.78];

%% Section 2: Create and display the table
...

```

This will create a section titled "Section 1: Define variables" and another section "Section 2: Create and display the table". If there was no subsequent %% marker after the first section, the script would consider everything after the first %% as part of that section until it reaches the end of the script. After you have inserted the %% for your sections, you can run, for instance, the first section independently from the rest of the script. To run all the code in the file, click the **Run** button in the Editor. To run Section 1, select it and click **Run Section** (Figure 98) or use the shortcut Ctrl (or Cmd) + Enter to run just that section.

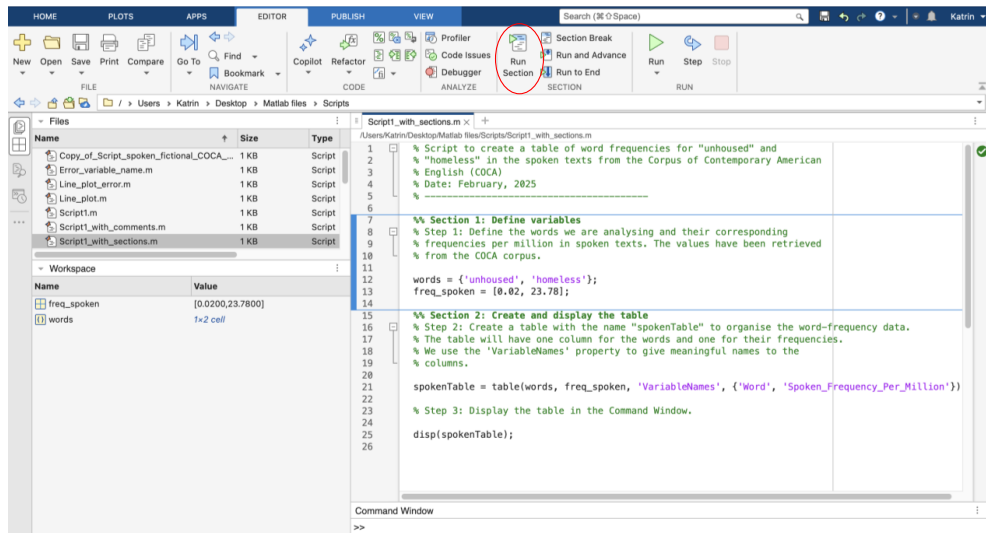


Figure 98: Running a section from a script

You can also execute a section and then automatically move to the next one by clicking **Run and Advance**. This is useful for running and testing parts of your script sequentially without manually navigating between sections.

In the following example, we will compare the frequency of the word “homeless” in two different sections from the COCA corpus: one representing spoken texts and the other representing fictional texts. We have already retrieved the frequency data from the spoken corpus section (“homeless”: 23.78 per million), and we can also obtain the data from the corpus for the fictional texts section (“homeless”: 11.65 per million). Our goal is now to write a small script in which we create variables for this data and visualise these frequencies using a bar chart, with separate bars for the word in the corpus sections of spoken and fictional texts (cf. also Chapter 7.3). You can achieve this by using a script like the one provided below:

```
% Script to create a bar chart for word frequencies in two sections of COCA
% ----
% Frequencies of word "homeless" per million words in the spoken section
% and fictional texts in COCA

freq_spoken = 23.78;
freq_fictional = 11.65;

% Create a bar chart
bar([freq_spoken; freq_fictional]');

% Set x-axis labels
set(gca, 'XTickLabel', {'Spoken', 'Fictional'});

% Add axis labels and title
xlabel('Corpus section in COCA');
ylabel('Frequency (per million)');
title('Frequency of "homeless" in spoken and fictional texts in COCA');
```

When you run this script, the following actions will occur: The script first defines the two variables `freq_spoken` and `freq_fictional`, which hold the frequencies of the word “homeless”

per million words in the spoken section and fictional texts of the COCA corpus. Then the `bar` function is called to create a bar chart. The input `[freq_spoken; freq_fictional]` organises the data into a format suitable for the bar chart, resulting in one for the spoken section and one for the fictional section. The `set(gca, 'XTickLabel', {'Spoken', 'Fictional'})` command sets the labels for the x-axis to “Spoken” and “Fictional”. `set` modifies properties of the axes, in this case, the x-axis tick labels, and `gca` stands for “get current axes”. This is a function that returns the handle to the current axes in the current figure and allows you to set properties of the axes in this figure. Finally, the `xlabel`, `ylabel` and `title` functions add descriptive labels to the x-axis and y-axis, as well as a title for the chart. The output will be a visual representation (bar chart) displaying the frequencies of the word “homeless” in both the spoken and fictional sections of COCA (Figure 99).

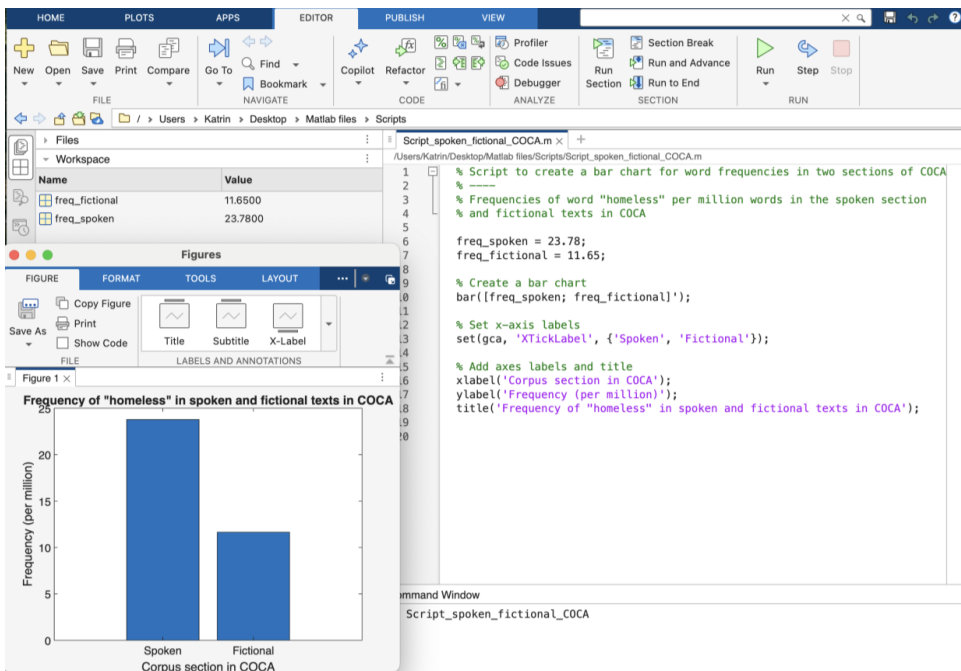


Figure 99: Using a script to create a customised bar chart

In MATLAB, you can also customise existing plots easily and interactively using various built-in features. You can combine short scripts with interactive editing. Let us assume you have just created a bar chart without any title or axis labels with the following short script:

```

% Frequencies of word "homeless"
freq_spoken = 23.78; % Fpm in spoken section in COCA
freq_fictional = 11.65; % Fpm in fictional texts in COCA

% Create a bar plot
bar([freq_spoken; freq_fictional]);

```

After running the script, you will have a figure without additional labels. You can further customise it using interactive options available in the figure window’s toolbar (Figure 100, cf. Chapter 7). That means you can click and interact with the plot, adjusting elements like the title, labels, colours or a legend, without writing any additional code.

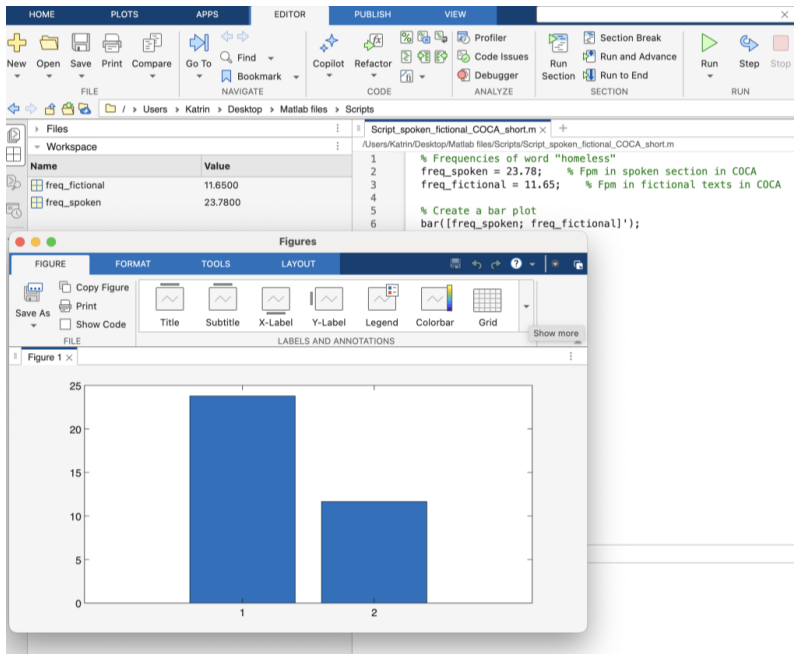


Figure 100: Using a short script to create a bar chart that needs further modifications

You can copy or modify earlier scripts, which is especially useful when you want to reuse code that works for a similar task but requires slight adjustments. Instead of starting from scratch, you can adapt your existing script to fit a new scenario. To create a copy of an existing script, go to the Files panel which is usually on the left side of the screen. Find the script you want to copy in the current folder, right-click on it, select **Copy** from the menu and then select **Paste** (or use Ctrl + C and Ctrl + V on Windows or Cmd + C and Cmd + V on macOS). The newly copied file will appear in the folder with the same name as the initial script, but with “Copy\_of\_” added at the beginning of the filename (Figure 101). You can rename the file if needed by right-clicking on it and selecting **Rename**, then typing the desired name.

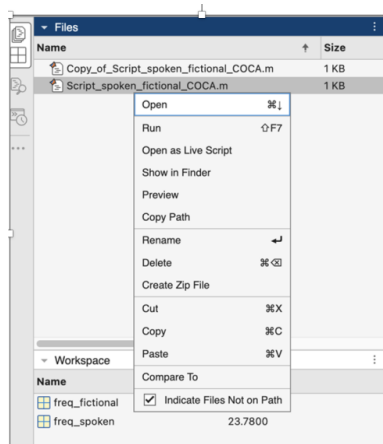


Figure 101: Creating a copy of an existing script in the Files panel

For example, initially, in our script, we wanted to analyse the frequency of the word “homeless” in two sections of the COCA corpus: the spoken and fictional texts. The script created a bar chart comparing these two sections. If the scenario expands to include a third section, such as the magazines section of the COCA corpus, we can either modify the existing script by adding a new data point or create a copy of the existing script, which we can then modify by adding a new data point for the magazines section. Once this is done, we can update the plot to include all three categories.

For more detailed information on how to create and manage scripts in MATLAB, you can consult the official documentation on scripts.<sup>17</sup>

## 6.3 MATLAB Live Editor®

MATLAB Live Editor enables users to create interactive documents that combine code, output and formatted text in a single executable file. This feature allows you to write live scripts (.mlx files) that not only include code, but also elements such as formatted text, images etc (cf. <https://www.mathworks.com/help/matlab/live-scripts-and-functions.html>). Live scripts in MATLAB may provide a more visually appealing way to present your work compared to traditional scripts. This approach makes it easier to explain your work and share it with others, for instance, if they combine code, output and explanatory text. Live scripts support interactive controls (like sliders and drop-downs) that allow users to manipulate parameters and see results in real-time (cf. [https://www.mathworks.com/help/matlab/matlab\\_prog/add-interactive-controls-to-a-live-script.html](https://www.mathworks.com/help/matlab/matlab_prog/add-interactive-controls-to-a-live-script.html)).

To create interactive forms in the Live Editor, begin by opening a new live script. In the following example, we insert the code from one of our above-mentioned scripts, e.g., the script to create a bar chart for word frequencies in two sections of COCA (Chapter 6.2). We can then add explanations before each code section using formatted text, include images, graphs or equations and use interactive controls to allow users to change features dynamically and see how the bar chart updates in real-time.

Figure 102 shows an example of a live script that also includes some formatted text and a date picker as an interactive control (choose adding text and controls from the toolbar options under the **Live Editor** tab). By clicking on **Export** in the toolbar under Live Editor you can export live scripts to various formats (PDF, Word, HTML etc.) for sharing.

---

<sup>17</sup> [https://www.mathworks.com/help/matlab/matlab\\_prog/create-scripts.html](https://www.mathworks.com/help/matlab/matlab_prog/create-scripts.html)

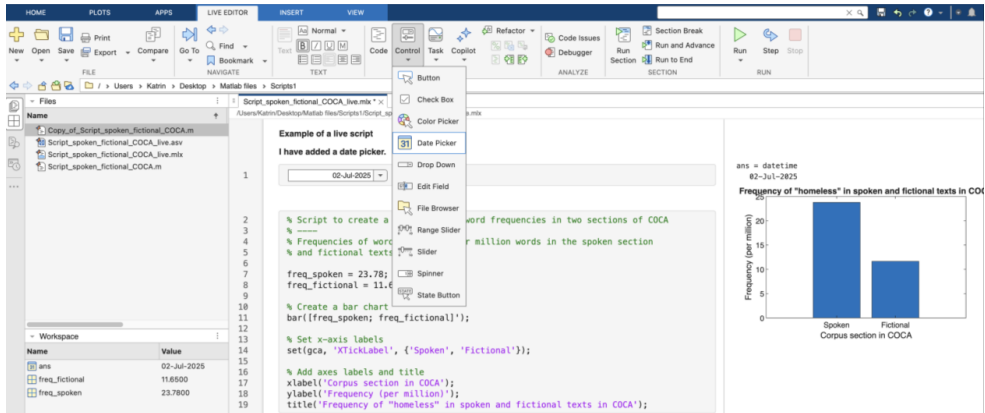


Figure 102: Example of a live script

When you select **Insert** on the toolstrip, you will also find various icons that allow you to add different elements to your live script. These include section breaks, text, tables of contents, tables, images, videos, equations etc. In Figure 103, a section break has been added by using this option from the toolstrip.

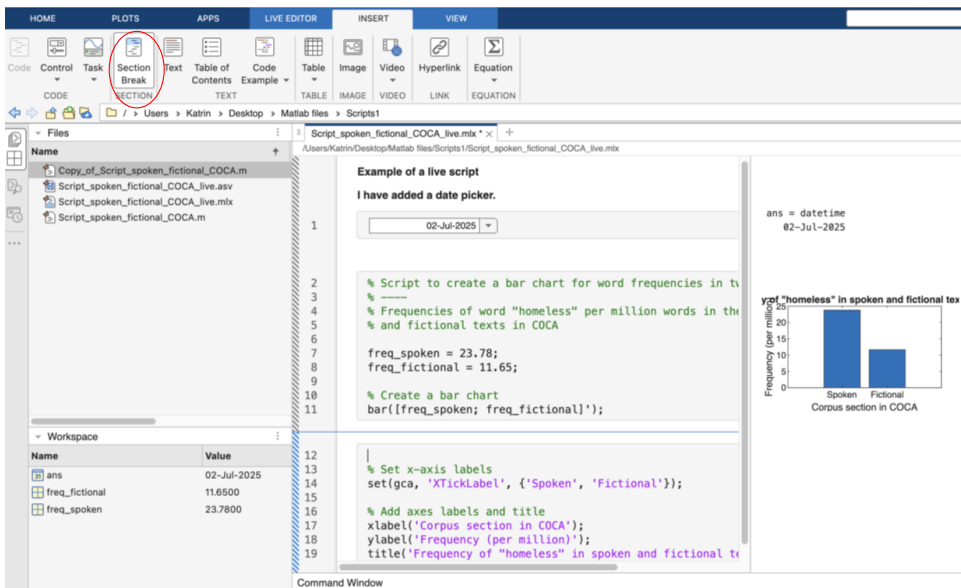


Figure 103: Adding a section break in a live script

One of the key features of the Live Editor is its ability to incorporate specific “tasks”, which are interactive point-and-click interfaces. These can be embedded into live scripts to perform specific operations and to simplify the process of exploring specific parameters. They will also automatically generate the corresponding MATLAB code for a series of steps. To add a task to your live script, you can either go to the Live Editor tab, click on **Task** and select from the available options, or you can type a relevant keyword like “preprocess” or “text” in a code block

in your live script. As you type, the Live Editor will display possible matches so that you can select and insert the desired task (cf. Chapter 10.3 for examples).<sup>18</sup>

For more information on the Live Editor and its capabilities, you can explore the following documentation and video resources:

*MATLAB Live Editor*

<https://www.mathworks.com/products/matlab/live-editor.html>

*Using the Live Editor*

<https://www.mathworks.com/videos/using-the-live-editor-117940.html>

You can also check the MATLAB File Exchange platform, which hosts a wide variety of user-contributed resources including many community-authored scripts and live scripts that you can explore and run directly:

*File Exchange*

<https://www.mathworks.com/matlabcentral/fileexchange/>

## 6.4 Debugging and error handling

This section covers some common types of errors in MATLAB, explaining how the software informs you about them and offering guidance on handling these issues. While debugging and error handling are complex topics, as mentioned earlier, only a limited selection of errors will be addressed here. Advanced debugging techniques are generally unnecessary for the short scripts presented in this book, so we will touch on these topics only briefly.

In MATLAB, common errors can arise from syntax issues, data type mismatches or improper function usage. These errors can occur when you type a command directly in the Command Window or run a script. Errors may also arise in several other situations, for example, when trying to load data from a file that does not exist or has an incorrect format. In interactive environments, such as when using tools like the Live Editor, errors may arise during interactive coding or inline plotting. Additionally, errors can occur if you try to use a function from a toolbox that is not installed on your system.

You might, for instance, encounter an error when the dimensions or sizes of the data structures you are working with, such as vectors or matrices, do not align in a way that a MATLAB function expects. For example, when plotting or performing element-wise operations, each element in one vector needs a corresponding element in the other vector. When plotting two vectors,  $x$  and  $y$ , as a line plot (cf. Chapter 7.7), the number of elements in each vector must be the same. This is because each element  $x(i)$  must correspond to a value  $y(i)$  in the other vector. If the vectors have different lengths, MATLAB cannot match the elements properly, leading to an error. Let us assume we define variables to represent the year of publication for a selection of corpus texts and the number of tokens in these texts and then plot those. Our sequence of commands, either entered subsequently into the command line or saved in a small script, might look like this:

```
years = [2020, 2021, 2022, 2023, 2024];
```

---

<sup>18</sup> cf. also [https://www.mathworks.com/help/matlab/matlab\\_prog/add-live-editor-tasks-to-a-live-script.html](https://www.mathworks.com/help/matlab/matlab_prog/add-live-editor-tasks-to-a-live-script.html)

```
num_tokens = [5600, 3500, 1300, 4900];

plot(years, num_tokens);
```

In this example, we have not added the value for the tokens for the text from 2024 yet. Since `years` now contains five elements and `num_tokens` contains four, we need to either add a value for the missing year or remove one of the years. Otherwise, MATLAB will provide an error message that points out the problem. It will also suggest to explain the error in more detail via Copilot, and if you wish, you can click on “Explain Error” (Figure 104).

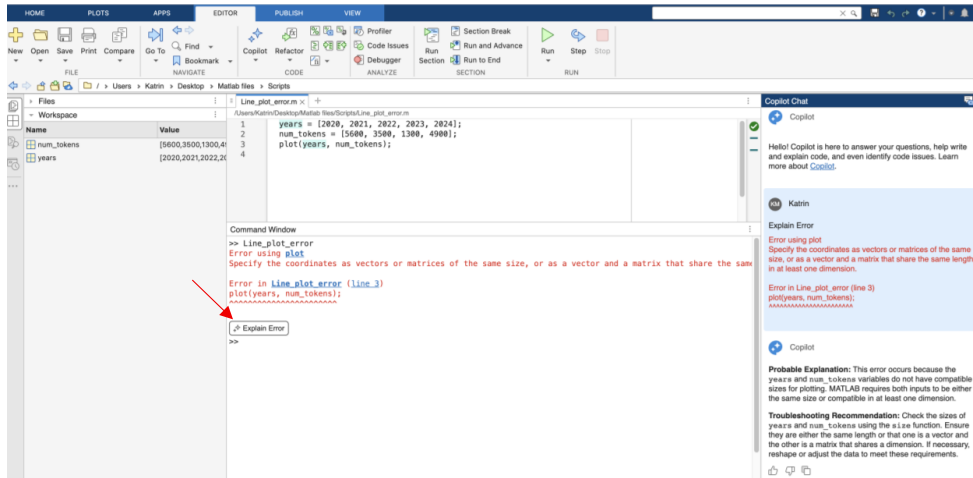


Figure 104: Error message in MATLAB

An error may also occur when trying to use a variable that has not yet been defined or when there is a typo in the variable name:

```
years = [2020, 2021, 2022, 2023];

num_tokens = [5600, 3500, 1300, 4900];

plot(year, num_tokens);
```

Here, MATLAB cannot identify the misspelled variable and generates an error indicating that the variable `year` is undefined since we initially defined the variable as `years`.

If you use a script for your sequence of commands, the error message will typically point to the specific line in the script where the issue occurs. In other cases, you might encounter a message like “This statement is incomplete.” This usually indicates a syntax issue where MATLAB expects something but does not find it, such as a missing closing parenthesis, quotation mark or similar element.

When you are unable to identify the cause of an error in MATLAB immediately, there are several strategies you can employ to troubleshoot and resolve the issue. A good starting point, of course, is to read the error message carefully and to use the information Copilot provides on the error, as it often offers valuable clues about the specific problem. Additionally, when working with scripts, MATLAB’s sections feature, discussed in Chapter 6.2, can be particularly helpful for isolating errors. By dividing your script into smaller, independent sections, you can

run each part separately to make it easier to identify where things go wrong. If an error occurs after executing one section, you can focus your attention solely on that section of the code, rather than running the entire script and getting distracted by unrelated parts. This targeted approach simplifies the debugging process. Another effective strategy is to simplify your code by commenting out sections that are not essential to the part you are testing. Temporarily disabling sections of code allows you to isolate the problem area and test smaller, more manageable portions of your script. By narrowing down the code you are working with, you can more quickly pinpoint where the issue lies.

In addition to these approaches, consulting MATLAB's official documentation can provide valuable insights. The documentation includes detailed explanations of functions, syntax and examples, which can help clarify how to use them correctly and troubleshoot issues. If the problem persists, searching online in forums or user communities can be very effective. Given MATLAB's widespread use globally, there are numerous discussions and shared solutions for common errors that other users have already encountered and resolved. To explore the topic of debugging and error handling further, you may wish to consult online MATLAB courses, such as the one referenced in Chapter 6.1.

If you encounter an issue with a MathWorks product that you are unable to resolve, you can also reach out to MathWorks Technical Support for expert assistance. Subscribers to the MathWorks Software Maintenance Service receive enhanced support, including priority access to technical help in multiple languages via phone, email and online resources, cf. [https://www.mathworks.com/support/contact\\_us.html](https://www.mathworks.com/support/contact_us.html) and [https://www.mathworks.com/help/matlab/matlab\\_env/contact-technical-support.html](https://www.mathworks.com/help/matlab/matlab_env/contact-technical-support.html).



# 7 Visualising linguistic data with MATLAB®

## 7.1 Overview

Visualisations are crucial for the analysis of linguistic patterns and allow us to explore trends, relationships and distributions in datasets. Chapter 7.1 begins with an overview of the possibilities for visualising linguistic data using MATLAB, including guidance on where to find relevant documentation. Chapter 7.2 provides a broad introduction to the key figure customisation options available in MATLAB. It includes guidance on how to locate and use MATLAB's interactive plotting features, save plots for future use and copy figures into external files such as Microsoft® Word documents. The subsequent section of Chapter 7 will then focus on specific plot types in greater detail. We will explore various plot types for visualising linguistic data derived from corpora, accompanied by practical examples. When discussing the individual plot types, we will also touch on relevant customisation options, such as adding labels, titles, legends, controlling colour schemes, axis properties, customising font size and style and adjusting marker styles. The examples will show how these options can be applied. While some of the visualisations presented maybe, in the view of some, resemble those that can be created in spreadsheet tools such as Microsoft® Excel, MATLAB provides unique strengths that go beyond standard charting. These include the ability to automate repetitive plotting tasks, handle large and complex datasets efficiently, perform advanced data analyses directly within scripts, and create highly customisable and publication-quality figures programmatically. Leveraging these capabilities allows researchers to develop reproducible workflows and integrate visualisations seamlessly with data processing and statistical analysis. Chapters 7.15 and 7.16 will provide a summary of the plot types covered so far and an overview of the most important interactive figure customisation options and plotting commands. Chapter 7.16 in particular will serve as a reminder of the best practices for effective data visualisation in MATLAB.

Various approaches for creating plots in MATLAB will be explored, including both interactive tools and command-line techniques. The interactive features of MATLAB are particularly useful for testing and experimenting with different ways to present data. On the other hand, programmatic methods may provide greater control over the visualisation process. They can also facilitate reproducibility and automation. The primary focus of this chapter will be on the interactive commands and layout adjustments that can be made using the options from the Plots tab toolbar and within the MATLAB figures themselves. These features enable users to customise their visualisations easily and with minimal or no coding effort. Alternative ways to use commands, either instead of or in combination with these options, to create and adjust figures will also be covered.

For an overview and examples of different plot types in MATLAB, you can visit the following link from the MathWorks documentation: [https://www.mathworks.com/help/matlab/creating\\_plots/types-of-matlab-plots.html](https://www.mathworks.com/help/matlab/creating_plots/types-of-matlab-plots.html).

More resources can be found at these links:

### *Graphics*

<https://www.mathworks.com/help/matlab/graphics.html>

### *2-D and 3-D Plots*

<https://www.mathworks.com/help/matlab/2-and-3d-plots.html>

You can also explore the MATLAB plot gallery here:

### Types of MATLAB Plots

<https://www.mathworks.com/products/matlab/plot-gallery.html>

Additionally, to get an overview of available commands in MATLAB related to figures, you can use several help commands in the Command Window. Typing `help graphics` will provide general information on graphics-related commands and `help plot` will provide information on commands related to plotting. For commands specific to creating two-dimensional plots, use `help graph2d`. For three-dimensional plotting, type `help graph3d`. If you are looking for more specialised graphing commands, you can use `help specgraph`. It is also worth asking MATLAB Copilot for specific help with graphs.

## 7.2 MATLAB® plotting basics

This section introduces basic information on creating figures and plots in MATLAB and outlines essential details on figure customisation options, saving plots for later use and exporting figures to external files. In these aspects, it primarily covers how to utilise MATLAB's interactive features, which allow users to make a wide range of adjustments directly through the graphical interface and customise figures with minimal coding effort.

When we select data from a MATLAB variable for plotting, we can do so by clicking on it in the Variables editor. Once the data is selected, various available plot options that are compatible with the data are displayed in a row in the **Plots** tab toolstrip (Figure 105). When you click on the small **Show More** button on the right side of the row with available plots, a drop-down menu appears with additional plot options.

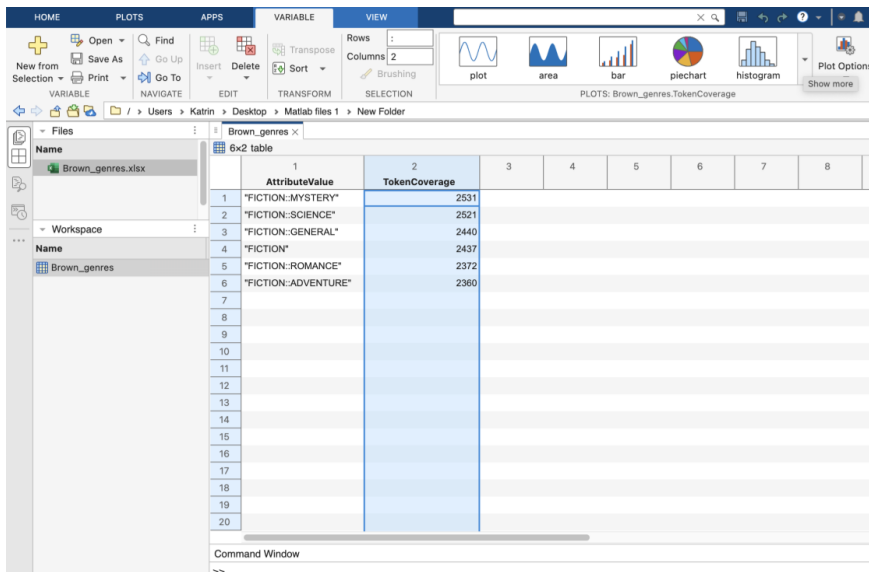


Figure 105: Available plot types in the Plots tab toolstrip for selected data

By default, MATLAB displays in the plot options only the icons for the options compatible with your selected data. However, you can opt to view all plot types by selecting **All Plots** from the dropdown menu, with incompatible options greyed out and available plot types highlighted (Figure 106).

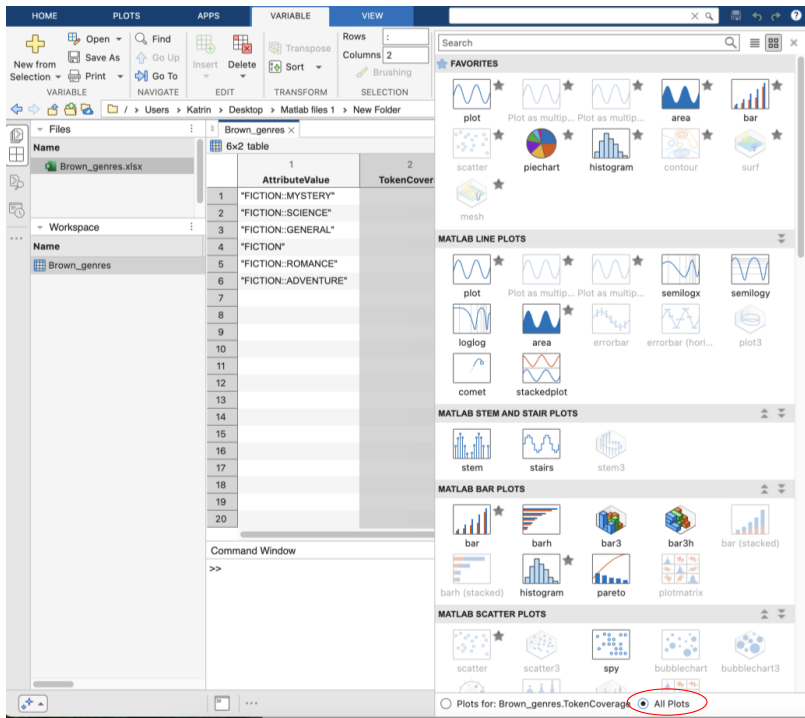


Figure 106: Displaying available plot options with incompatible options greyed out

In the dropdown menu next to each plot type icon, you can mark your favourite and frequently used graphs by clicking the star icon. To do so, mouse over the desired plot type and click the star. If the star is highlighted, it means the plot is added to your favourites. Click the star again if you want to remove a plot type from the favourites. You can access your favourite graphs later by navigating to the **Favourites** section at the top of the toolbar. These graphs will also be displayed in the toolbar for quick access.

When you click on the **Plot Options** icon on the Variables tab, you get access to several configuration options that let you customise how figures and plots behave in your session (Figure 107). These settings apply globally to all the figures and plots that you create unless otherwise specified for individual figures. You can, for instance, “dock” your figures inside the MATLAB environment. A docked figure will then appear as a tab inside the MATLAB window rather than in a separate window. You can also let MATLAB automatically add a title suggestion to your plot. Using the **Generate Linked Plot** feature from the options here allows you to create plots that are linked to workspace variables. This means that any changes made to the variables in the workspace will automatically update the linked plots in real-time.

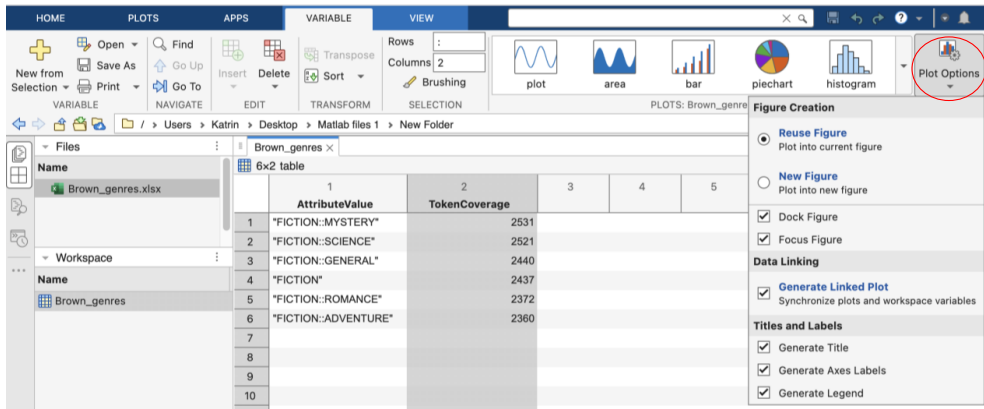


Figure 107: Customising settings of figures and plots

To save a figure for future use in MATLAB, go to the **Figure** tab and select **Save As** or choose this option by selecting the respective icon on the right side above your figure (Figure 108). This will allow you to reopen the figure in a subsequent MATLAB session. The figure also preserves all its current features, along with any customisations and formatting you may have applied (like axis labels, title or colour modifications), so that you do not have to redo them each time you open the figure.

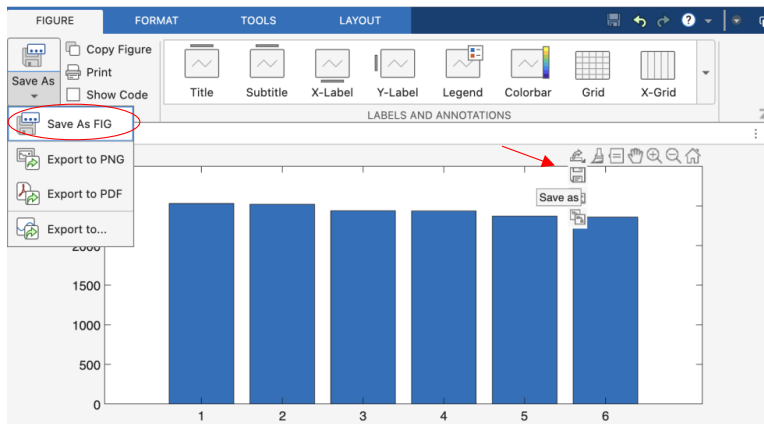


Figure 108: Saving a figure

In MATLAB, the standard format for saving figures is the `.fig` format. This format stores the figure along with all its data and allows you to reopen and interact with the figure in MATLAB later. Additionally, you can export and save your figures in various other formats when you select the “Save As” option from the toolbar. A drop-down menu will appear, offering options such as **Export to PNG** (to save your figure as an image in Portable Network Graphics format, `.png`) or **Export to PDF** (to save your figure as a document in Portable Document Format, `.pdf`, suitable for printing or sharing). When you select **Export to...** (Figure 108 above), a window will appear displaying additional options such as **JPEG** (`.jpg`), **SVG** (Scalable Vector Graphics, `.svg`), and several other supported formats for saving your figure (Figure 109).

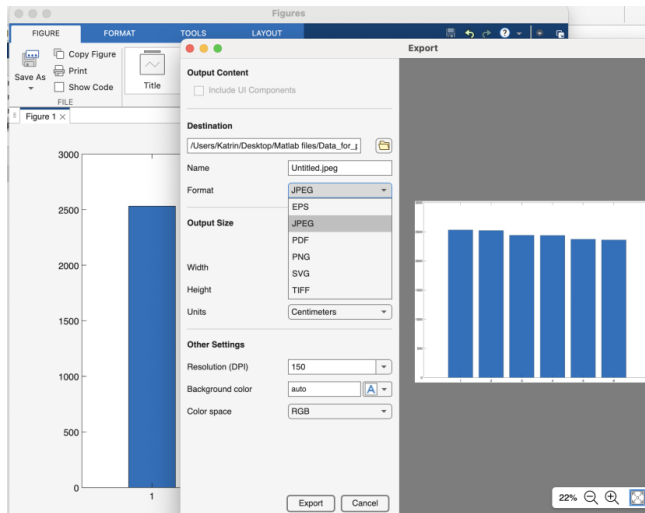


Figure 109: Choosing a file format for exporting a figure

You can also use the **Print** option from the **Figure** tab and select various layout features (Figure 110).

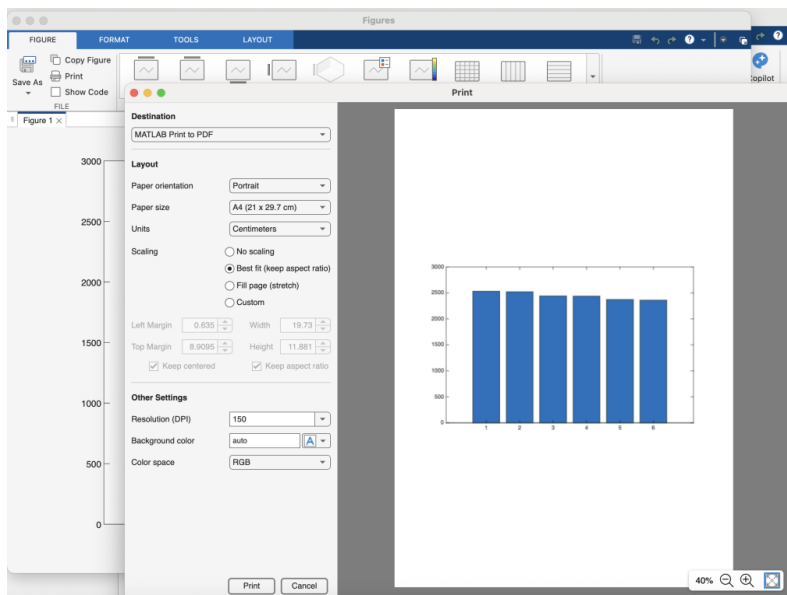


Figure 110: Printing a figure to a PDF file

Under the **Figure** tab of your figure, you can also find the possibility to select **Copy Figure**, which allows you to copy your figure and to paste it into other applications such as a Microsoft Word document. Once this option is selected, the figure is copied to your clipboard and you can paste it into a document by pressing **Ctrl (or Cmd) + V**.

To save figures in MATLAB via commands, you can use the `saveas` function for quick saves or the `print` function for more control over the output. The `saveas` function saves the current figure (`gcf`) to a specified file format and, by default, uses the format inferred from the file extension. For example, `saveas(gcf, 'New_Figure');` saves your current figure using the default `.fig` format, while `saveas(gcf, 'New_Figure.pdf');` saves it as a PDF file under the name “New Figure”. If you want more control over the output format and resolution, you can use the `print` function. For instance, `print(gcf, 'New_Figure.pdf', '-dpdf');` saves the figure also as a PDF, and `print(gcf, 'New_Figure.jpg', '-djpeg', '-r150');` saves it as a JPEG with 150 DPI (dots per inch) resolution. The resolution can be adjusted, for example, `-r300` provides print-quality graphics. Alternatively, if you prefer working with commands in MATLAB, consider using the `exportgraphics` function, which offers control over resolution and other layout features. More information about these functions can be found at the official MATLAB documentation pages.<sup>19</sup>

There are many options for editing the properties of your figure in an easy way, to insert legends, add labels etc. Have a look at the options from the list and the drop-down menu under the **Labels and Annotations** section on the toolbar in the Figure tab (Figure 111 and Figure 112).

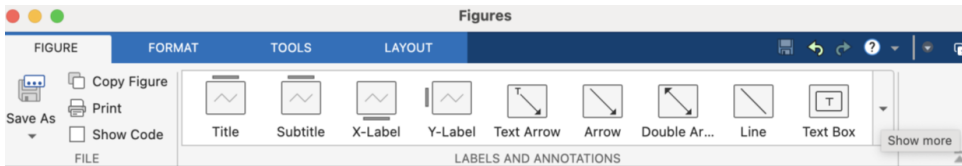


Figure 111: Labels and annotations section on Figure toolbar

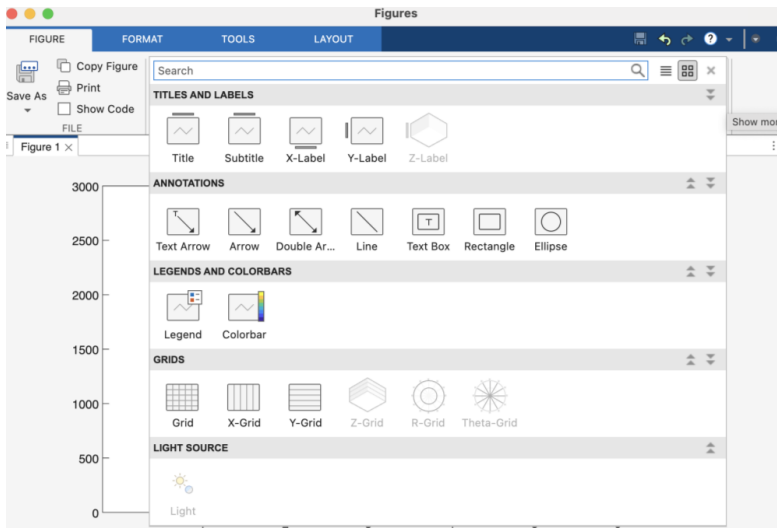


Figure 112: Drop-down menu for figure labels and annotations

When you click on **Select & Plot Edit** from the options under the **Format** tab for a figure that you have created or opened, you will find a range of interactive customisation options in the

<sup>19</sup> <https://www.mathworks.com/help/matlab/printing-and-exporting.html>

toolbar (Figure 113). This allows you to modify various figure properties, such as colours or fonts without writing any code. For more detailed interactive customisation, click on the **More Properties** icon on the right, which opens the **Property Inspector** (Figure 113, cf. also <https://www.mathworks.com/help/matlab/ref/propertyinspector.html>).

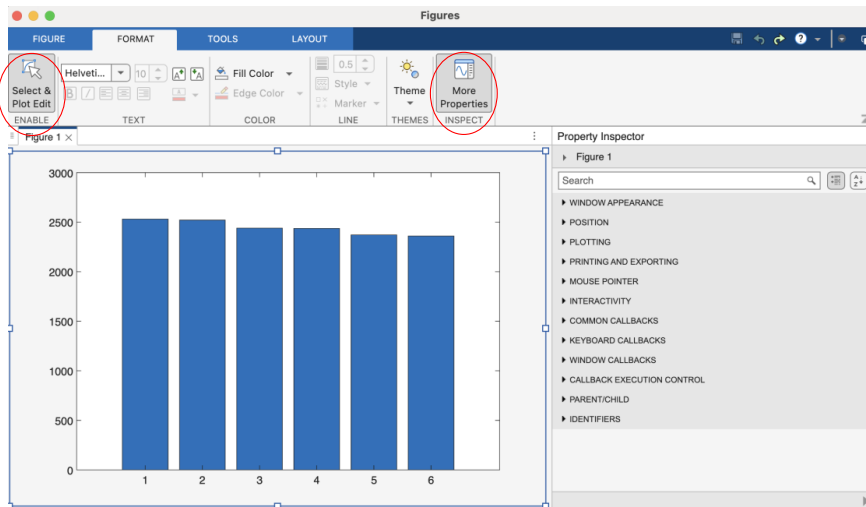


Figure 113: Opening the Property Inspector of a figure

This tool displays a comprehensive list of properties. You can click on the small triangles to expand the corresponding sections and click again to collapse them. An alphabetical view of the options is available by clicking the A–Z sorting control icon on the right. The Property Inspector allows you to customise and adjust a wide range of figure elements including axes properties (Figure 114) and it enables you to modify property values directly by editing the fields next to each property name. Any changes you make are immediately reflected in the figure. We will cover several of these figure elements in more detail in the following sections, when we apply them to specific figures.

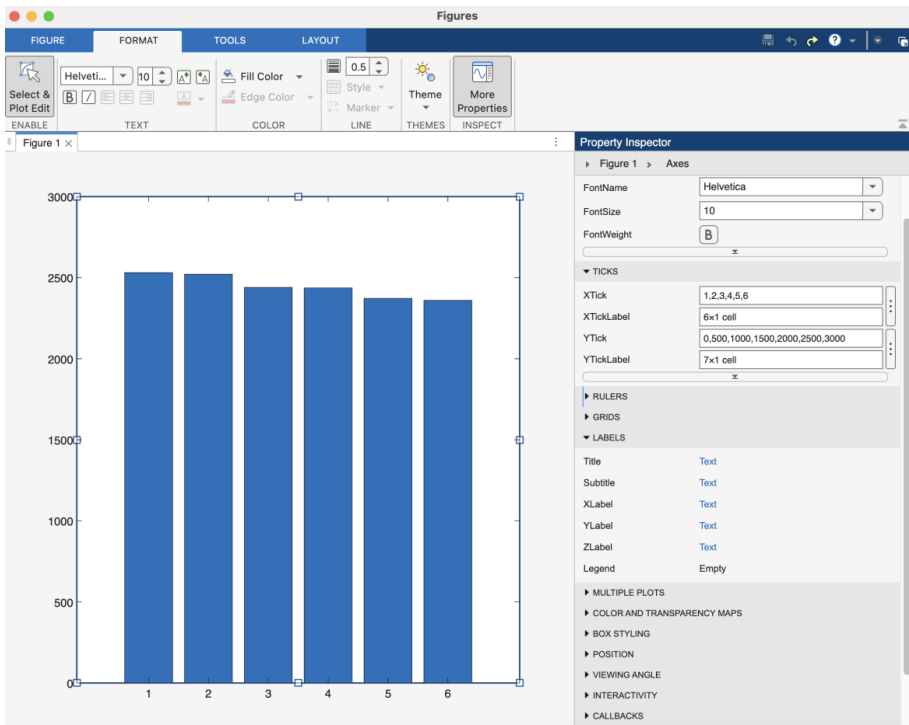


Figure 114: Adapting axes properties via the Property Inspector

You may also open the Property Inspector for your selected figure by using the `inspect` command in the Command Window. Alternatively, you can open it by right-clicking on your figure and selecting the respective option from the drop-down menu that will appear (Figure 115).

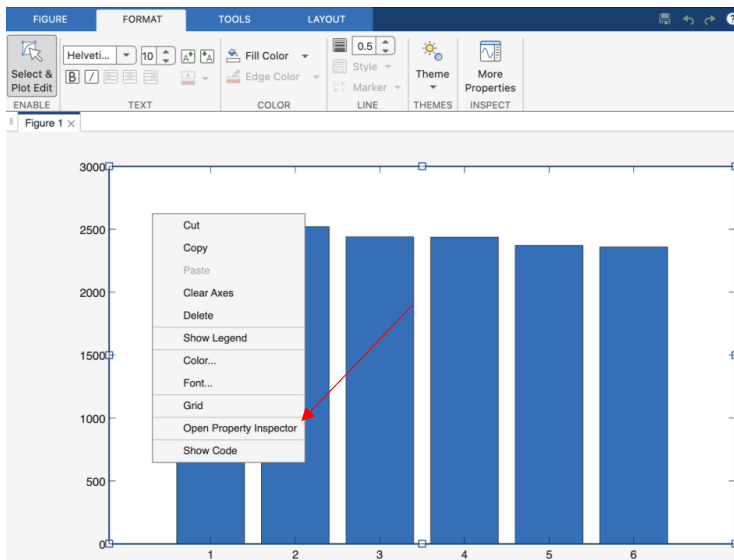


Figure 115: Accessing the Property Inspector

You can see that several functions in MATLAB can be accessed or executed in different ways. The interface is designed to be intuitive, allowing you to choose the method that works best for you. Sometimes, there are also multiple intuitive ways to execute the same function with a click. For each function, there are also corresponding commands if you prefer to work directly with them. However, for those new to programming, the interactive options might be a good starting point, as they provide an easier and more visual way to make adjustments.

As mentioned above, we will cover various options for editing specific properties of plots and figures in the following sections. Chapter 7.15 will provide a summary of key figure customisation options and plotting commands.

## 7.3 Vertical bar charts

In this section, we will explore how to create vertical bar charts in MATLAB and edit some of their key properties in an easy way. Bar charts are a widely used visualisation tool in corpus linguistics, particularly for comparing the frequencies of words, phrases or grammatical features across different categories or text types. They are especially useful for representing data that falls into distinct, non-overlapping groups. For example, a bar chart can illustrate how frequently specific linguistic features or structures, e.g., passive constructions, occur in different sections of a corpus, such as academic texts, magazine articles or works of fiction. In addition to comparing linguistic features, bar charts may also be used, for instance, to display the total number of words in each corpus section visually. An overview of bar charts in MATLAB can be found in the official MathWorks documentation under the following link: <https://www.mathworks.com/help/matlab/ref/bar.html>.

As we have already briefly discussed in Chapter 4.3, you can create a bar chart by simply clicking on the **bar** icon in the **Plots** section after having selected a variable. Additionally, in Chapter 6.2, we briefly explored how to use the `bar` function to create bar charts via the command line. To further deepen our understanding of bar charts in MATLAB, let us consider an example where we aim to visualise the sizes of selected subcorpora from the Brown Corpus, each representing a specific text genre. This well-known corpus resource, compiled by Francis and Kučera already in 1964, includes American English prose from 1961 and is available as an open resource on the Sketch Engine website (<https://www.sketchengine.eu/brown-corpus/>). We will first extract the relevant data from the Brown Corpus, then import it into MATLAB, and finally create a bar chart based on this data.

To explore the metadata in the Brown Corpus, including details like the labelled genre of the texts, navigate to the Brown Corpus on the Sketch Engine platform and click on “Corpus Info” (Figure 116). This will take you to Sketch Engine’s Text Type Analysis tool, where you can access various types of metadata included in the corpus. For more information, visit the Text Type Analysis guide on the Sketch Engine website (<https://www.sketchengine.eu/guide/text-type-analysis/>).

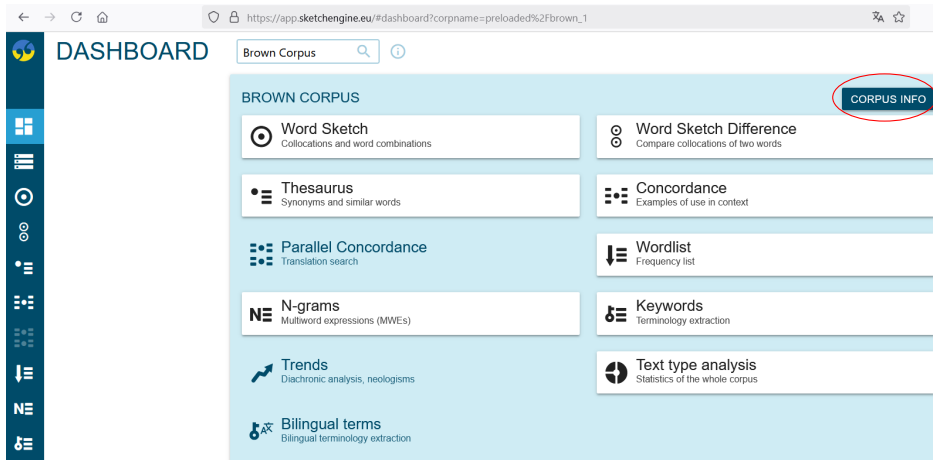


Figure 116: Accessing metadata details of Brown Corpus from Sketch Engine’s Corpus Info section

In the metadata of the Brown Corpus under the section Text Types, we will then find the “Genre” category with information on the different corpus registers. By clicking on it, we can switch between different measures, such as the number of structures or tokens. It is also possible to choose between fine-grained and broader category displays for the genres. Table 1 lists the names of the corpus sections from the Brown Corpus to which genre information has been assigned, along with their corresponding token counts. Note that the token count for the entire corpus is higher, with more than one million tokens in total, while the sum here is less than 650,000 tokens, and the information in the table reflects the genre classification at the time of writing this book.

Attribute Value	Token Coverage
LEARNED	178,942
BELLES-LETTRES	174,491
POPULAR LORE	111,400
SKILL AND HOBBIES	83,650
RELIGION	37,407
HUMOR	19,622
FICTION::MYSTERY	2,531
FICTION::SCIENCE	2,521
FICTION::GENERAL	2,440
PRESS::EDITORIAL	2,439
FICTION	2,437
FICTION::ROMANCE	2,372
FICTION::ADVENTURE	2,360
PRESS::REPORTAGE	2,254
PRESS	2,223
PRESS::REVIEWS	2,222
MISCELLANEOUS	2,217
MISCELLANEOUS::GOVERNMENT & HOUSE ORGANS	2,192

Table 1: Size of different genres in Brown Corpus, hosted on Sketch Engine

We can save this data from Table 1 in an Excel file from which we can then import all or selected data into a MATLAB variable as explained in Chapter 5.5. Let us imagine that we want to create a bar chart to visualise the sizes of the various fictional subgenres in this dataset. This would allow us to compare the number of tokens across different categories within the broader

fiction genre, such as mystery, science fiction, romance and adventure. A bar chart would make it easy to see how many tokens each subgenre has, showing their individual contributions to the overall fiction category.

Before importing the Brown Corpus data into MATLAB (saved in table format in our Excel file), we must ensure that the file is saved in the folder currently set as the MATLAB working directory. This will make the file appear in the Files panel. Once it is visible, we can select it and click **Import Data**. After that, we can proceed to select the relevant cells for import. To select multiple sections at once, hold down the Ctrl (or Cmd) key on your keyboard and click on the cells corresponding to the Brown Corpus subgenres with fictional texts and their token numbers (Figure 117). If we set the output type as “Table” and click on **Import Selection**, a table will be created in the MATLAB workspace, which we can access through the Variables editor.

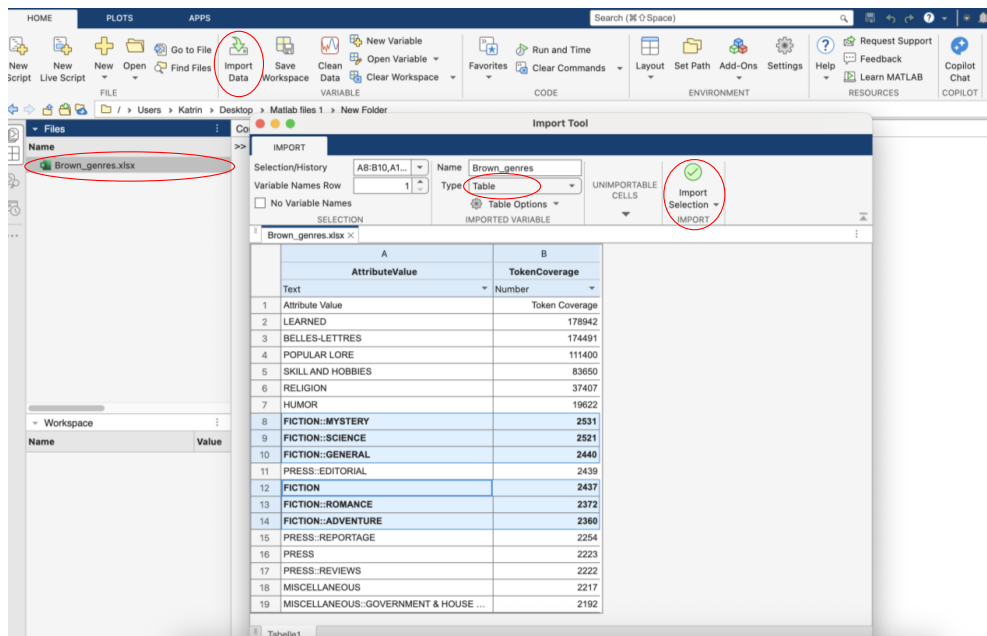


Figure 117: Selecting multiple sections of fictional texts from Excel table for import into MATLAB

After importing the data, a simple way to create a vertical bar chart for the token coverage of the selected genres is to click on the **bar** icon in the **Variable** tab toolstrip or in the **Plots** tab toolstrip after selecting the column containing the token data, which we had imported here as part of a table (cf. Figure 118).

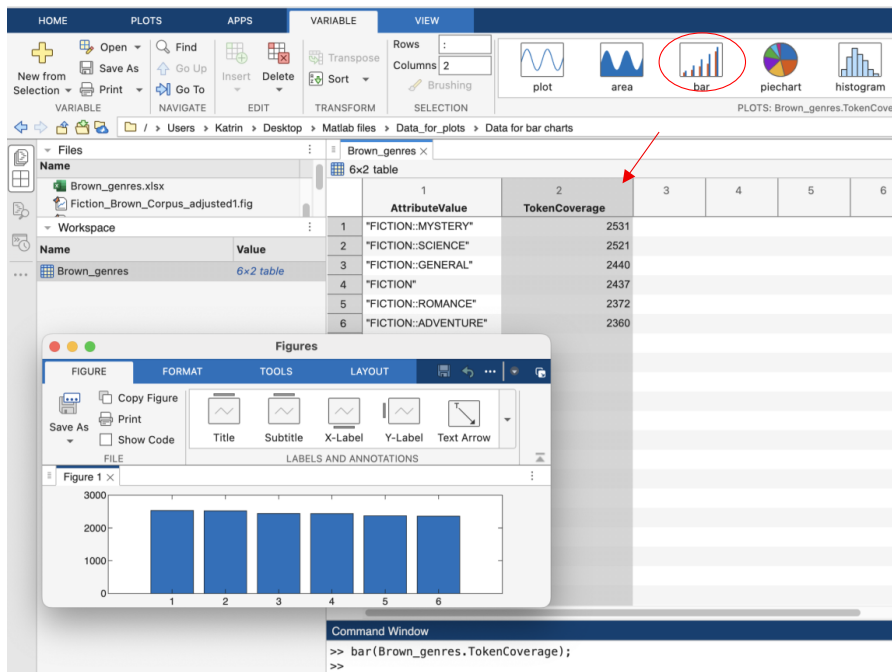


Figure 118: Creating a vertical bar chart

When you click on the bar chart icon, MATLAB will create a vertical bar chart with one bar for each element from your column. At the same time, MATLAB displays the equivalent command-line code used to create the plot in the **Command Window** – this is the line of code MATLAB executes behind the scenes. For example, if your workspace contains a table named `Brown_genres` and you select the `TokenCoverage` column to create a figure via the bar chart icon (cf. Figure 118 above), MATLAB will display the following command in the Command Window: `bar(Brown_genres.TokenCoverage)`. This interactive-to-command-line feature in MATLAB is particularly useful for learning and understanding how to replicate graphical operations programmatically. Alternatively, you can create your figure by typing this command directly into the Command Window. Entering `bar(Brown_genres.TokenCoverage)` there will generate the same bar chart as clicking the bar chart icon in the Plots section.

Save your bar chart as explained in Chapter 7.2, e.g., by clicking on **Save as** under the **Figure** tab of your figure. Let us now explore some specific adjustments that can be made to our bar chart example. In this example, we aim to replace the current numeric x-tick labels (1, 2, 3, 4, 5 and 6) with the names of the corpus sections (e.g., `FICTION::MYSTERY`, `FICTION::SCIENCE` etc.), add labels to the x- and y-axes and assign a title to the figure. We also plan to adjust the font size and colour of the axis labels to enhance the visual appearance of the plot.

To begin, we will modify the x-tick labels. Open the **Property Inspector**, e.g., by clicking on **Select & Plot Edit** under the **Format** tab in the figure window and then clicking on **More Properties**. Then navigate to the **Axes Properties** option in the Property Inspector. You can either browse through the categories (and find a link to the Axes properties under the category **Interactivity**) or simply type “Axes” in the search window to locate the Axes properties (Figure 119).

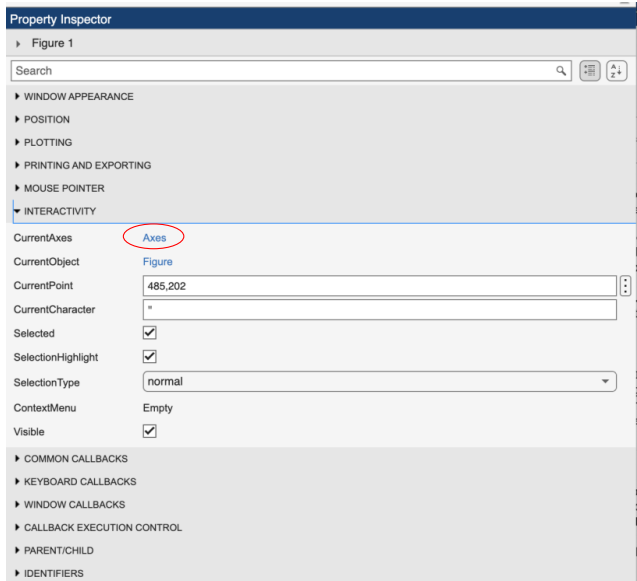


Figure 119: Navigating to Axes Properties in Property Inspector

In the **Ticks** section of the Axes Properties, you will find options for **XTick** and **XTickLabel**, which currently both display numerical values from 1 to 6. By clicking on the three horizontal dots next to these fields, a table will appear, allowing you to replace the numerical x-axis tick labels with descriptive names for the corpus sections (Figure 120). These names will then appear in place of the numbers on the x-axis of your bar chart.

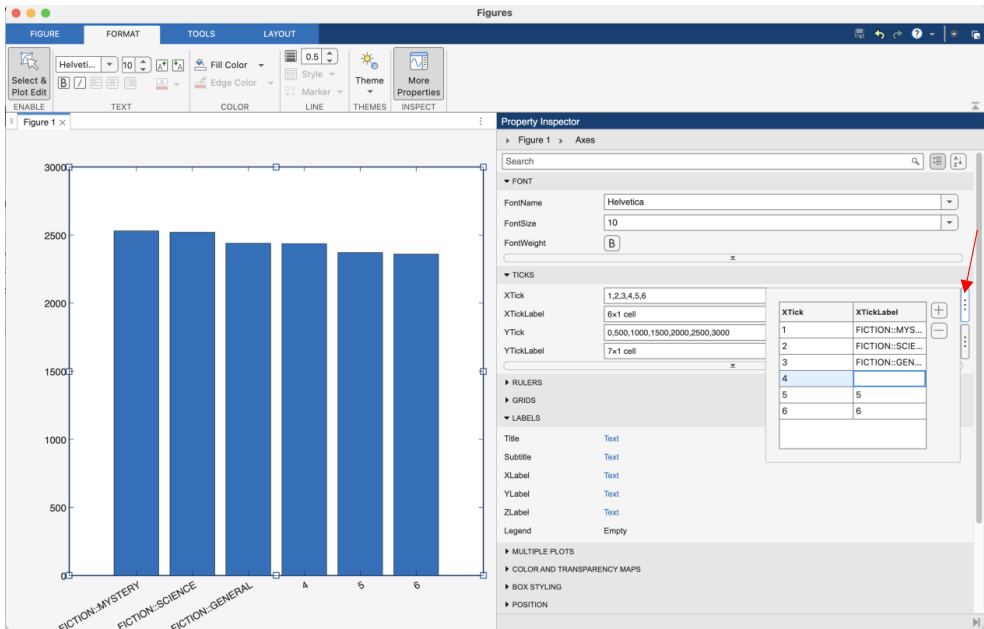


Figure 120: Adjusting x-tick labels through the Property Inspector

When you add details via the Property Inspector, the interactive-to-command-line feature does not capture those specific changes. That means that the Property Inspector allows you to modify object properties interactively, but it does not generate command-line code for every adjustment made through it. If you have access to MATLAB Copilot, you can ask it to look up the commands related to specific changes you made interactively.

You can combine both interactive methods and command-line adjustments when creating and customising your plots, depending on your preferences. Commands may allow for faster and more repeatable customisations. After having created a bar chart, you can of course also modify the x-tick labels by only using a command (cf. Chapter 6.2) instead of using the interactive Property Inspector. In our case, we can do this by using:

```
set(gca, 'XTickLabel', {'FICTION::MYSTERY', 'FICTION::SCIENCE', 'FICTION::GENERAL', 'FICTION', 'FICTION::ROMANCE', 'FICTION::ADVENTURE'});
```

After making any changes, it is advisable to save the figure again, either by overwriting the existing file or saving it under a different name to maintain multiple versions, especially if you are planning to use it in an academic paper or presentation.

Next, we will add labels to the x- and y-axes to indicate what each axis represents. You can use the `xlabel` and `ylabel` commands in the Command Window or your script, e.g., `ylabel('Size in tokens')`. Alternatively, you can add axis labels to your bar chart interactively by choosing **X Label** or **Y Label** from the **Axis Properties** of the Properties Inspector in the **Labels** category, which is located below the sections for ticks, rulers and grids (cf. Figure 120 above) or you can select **X Label** or **Y Label** conveniently from the **Labels and Annotations** toolstrip from the **Figure** tab of your figure (Figure 121).

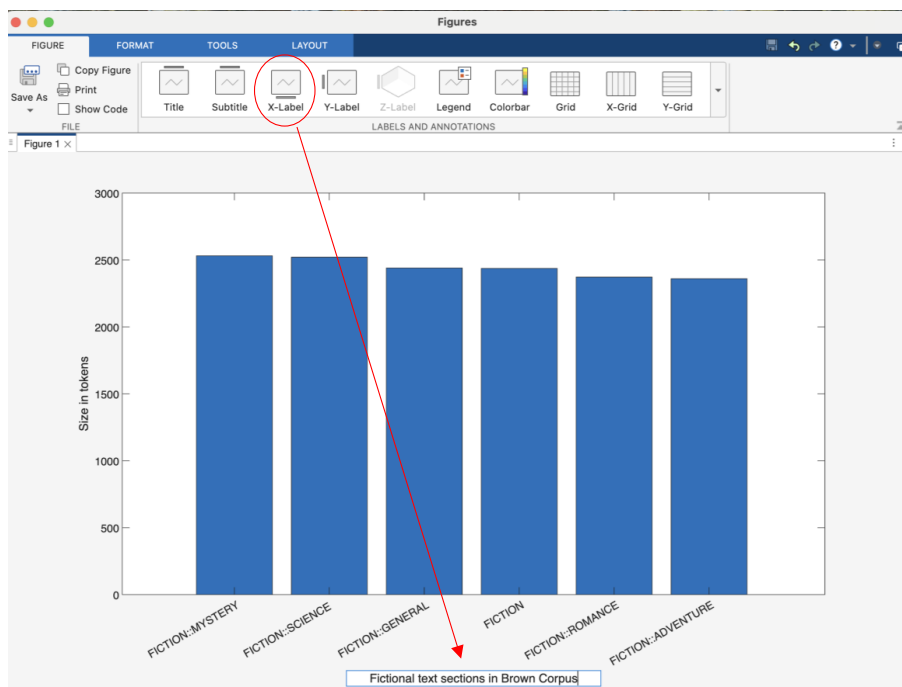


Figure 121: Adding an x-axis label

Both options (the **Labels** section of the **Axis Properties** of the Properties Inspector and the **Labels and Annotations** toolbar from the **Figure** tab) also allow you to add a title to your figure. Once the axis labels and the figure title are added, you can adjust their appearance further (e.g., to adapt the font size or colour) to suit your preferences via the **Format** tab of your figure. The Format tab of your figure provides the most commonly used customisation options. After clicking **Select & Plot Edit**, you can simply click on individual elements of your figure to format them directly (Figure 122).

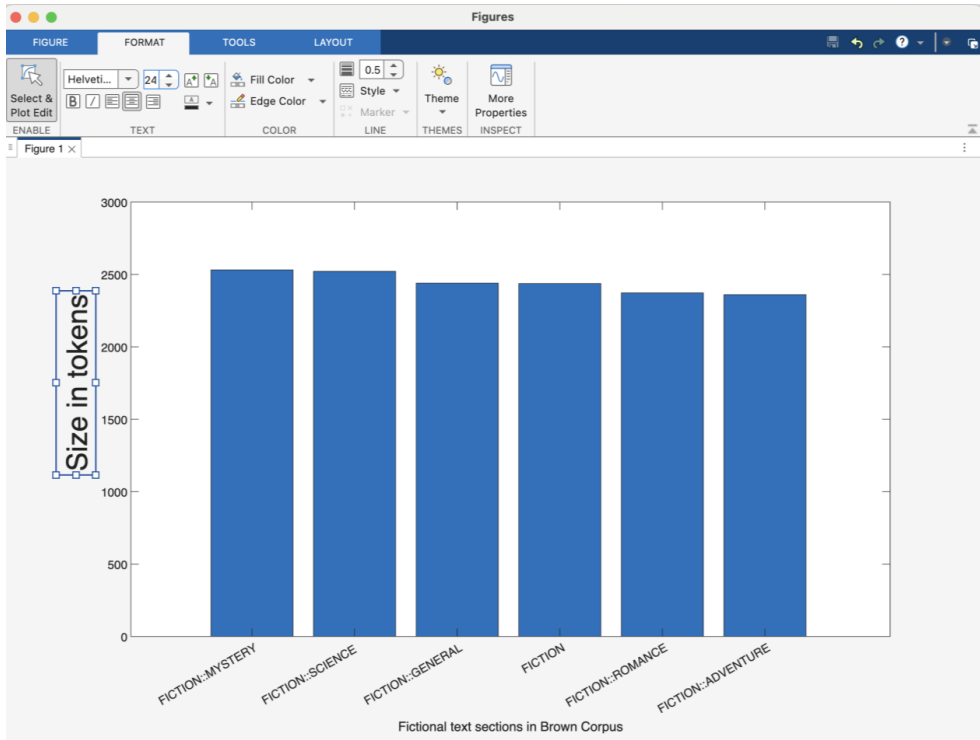


Figure 122: Adjusting axis label font size via the figure's Format tab

For the same and even more fine-grained adjustments, have another look at MATLAB's **Property Inspector**. It offers a wide range of customisation options, and exploring them is a great way to familiarise yourself with the software's features. You can modify various elements such as rulers, grids, labels and box styling to change the appearance of your figure. To get a better understanding of the available options, try navigating through the different sections and experimenting with the settings. For example, you can add grid lines, adjust the axis limits, modify the appearance of the bars and customise the axis labels by changing the font or the font size or adding colour to the background or text elements (cf. Figure 123, which is a modified version of Figure 122). By using the Ctrl (or Cmd) + Z and Ctrl (or Cmd) + Y shortcuts, you can undo and redo actions, respectively.

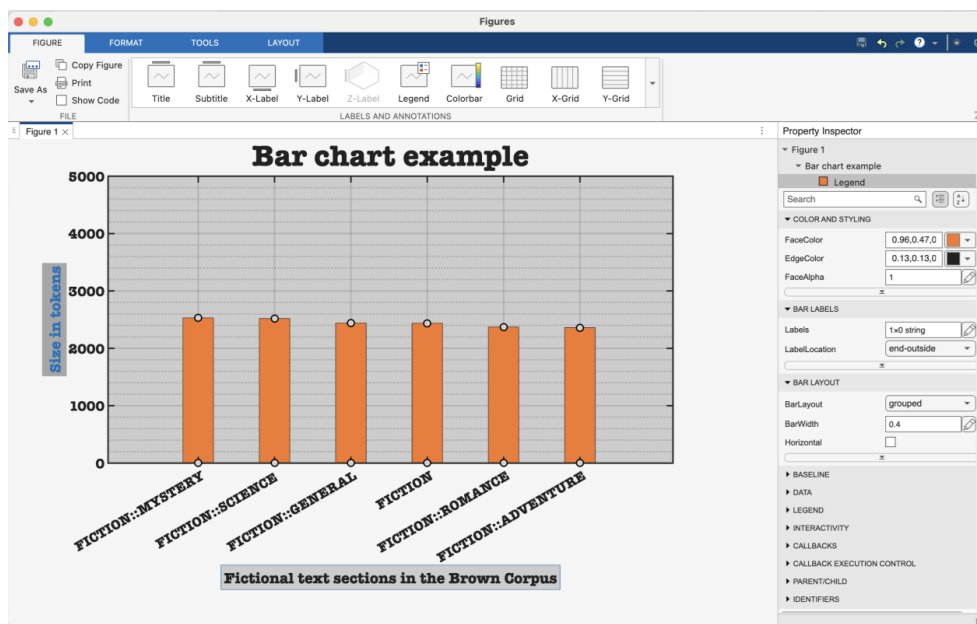


Figure 123: Bar chart customised via the Property Inspector

Let us consider another example of a vertical bar chart. In Chapter 5.3, we previously looked at how to create a numeric variable in MATLAB to represent the sizes of different corpus sections (e.g., spoken, newspaper and academic texts) from the British National Corpus (BNC) which is available on the English-Corpora.org website. By using the **Chart** option in the query interface, we obtained data on the size of each corpus section, measured in millions of words. The English-Corpora.org website also features the Strathy Corpus of Canadian English (Davies 2012–), which has a similar structure to the BNC in terms of text types, though the sizes of the corpora and their sections differ. By looking up any word in the Strathy Corpus using the **Chart** option, you will find a table that provides information on the size of each corpus section in this corpus in millions of words.

To compare the sizes of the individual sections across both corpora in a bar chart, we can first gather the size data for each section as described above. Table 2 presents the size of each corpus section in both the BNC and the Strathy Corpus. In total, the BNC contains 100 million words, while the Strathy Corpus contains 45 million words.

Corpus section	Size of section in BNC	Size of section in Strathy
Spoken	10,000,000	5,700,000
Fiction	15,900,000	3,700,000
Magazine	7,300,000	10,000,000
Newspaper	10,500,000	13,100,000
Non-academic	16,500,000	2,500,000
Academic	15,300,000	14,400,000
Miscellaneous	20,800,000	100,000

Table 2: Size of corpus sections in BNC and Strathy Corpus

Let us create a new numeric variable in MATLAB by clicking on **New Variable** in the **Variable** section in the MATLAB **Home** tab toolstrip. Once you select the variable in the workspace,

right-click on it to assign a name. After opening the variable in the Variables editor, you can manually enter values representing the sizes of the respective corpus sections into the columns provided. Alternatively, if the data already exists in an Excel file, you may import it directly from there into a numeric matrix using MATLAB's import function. Another efficient approach is to copy and paste the values directly from your spreadsheet. Select the data in Excel or another table format and use the shortcut Ctrl (or Cmd) + C to copy and Ctrl (or Cmd) + V to paste it into the empty numeric MATLAB variable. Before pasting, ensure that any commas used as thousand separators (e.g., 10,000,000) are removed, as they may cause MATLAB to misinterpret the values as text and trigger an error message.

We can then select our data in the Variables editor and click on the **bar** icon in the toolstrip to generate a bar chart or we can write a command in the Command Window, such as `bar(Sections_BNC_Strathy)` without any additional parameters, assuming that your variable is named `Sections_BNC_Strathy`. Figure 124 shows the bar chart that we create by using any of the options described above.

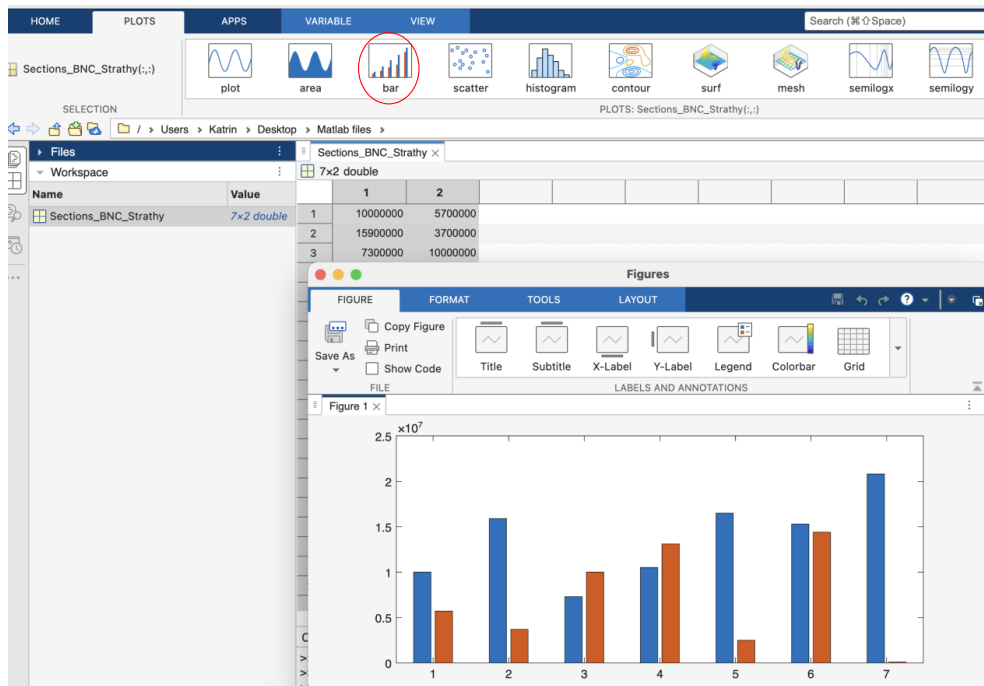


Figure 124: Bar chart for sizes of sections in BNC and Strathy Corpus

We can now add a legend by clicking the appropriate icon in the **Figure** tab toolstrip options or selecting the option the Format tab by right-clicking on the figure after having selected it. Then we can manually enter the corpus names that correspond to the bar chart colours. The first bars represent data from the BNC, indicated by one colour, while the second bars, marked by a different colour, represent data from the Strathy Corpus.

Next, we replace the x-tick labels, which currently display values from 1 to 7. As described earlier, this can be done by selecting **Axes** in the Property Inspector. In the **Ticks** section, we

enter the names of the corpus sections under **XTickLabel**. To label the y-axis, select the corresponding option in the Labels & Annotations section of the Figure tab or in the Property Inspector and type “Size in tokens”.

Furthermore, we might want to avoid displaying y-axis numbers in exponential format, such as  $2.5 \times 10^7$  and we may prefer to show them as 25 million or 25,000,000. This adjustment can also be made under **Axes Properties** in the **Ticks** section by specifying the desired format under **YTickLabel** (cf. Figure 125).

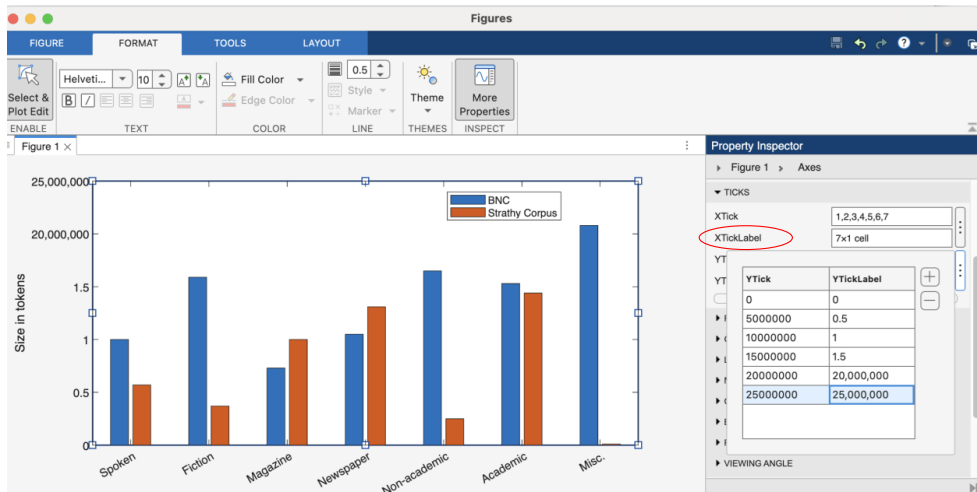


Figure 125: Changing y-axis tick labels

We can also create a 3-D bar chart for the same dataset by using the command `bar3(Sections_BNC_Strathy)` in the Command Window or by clicking on the **bar3** icon in the options from the MATLAB **Plot** toolstrip (Figure 126). For more detailed information on this function, you can refer to the official MathWorks documentation under <https://www.mathworks.com/help/matlab/ref/bar3.html>. While this plot type is an option, it may not be the most practical choice for visualising this particular dataset. However, it can still be worth experimenting with, as it offers a different perspective on the data. Once you have generated the 3-D bar chart, you can interact with it by left-clicking on the plot and holding down the mouse button. This allows you to rotate the chart, giving you a dynamic view of the data from different angles.

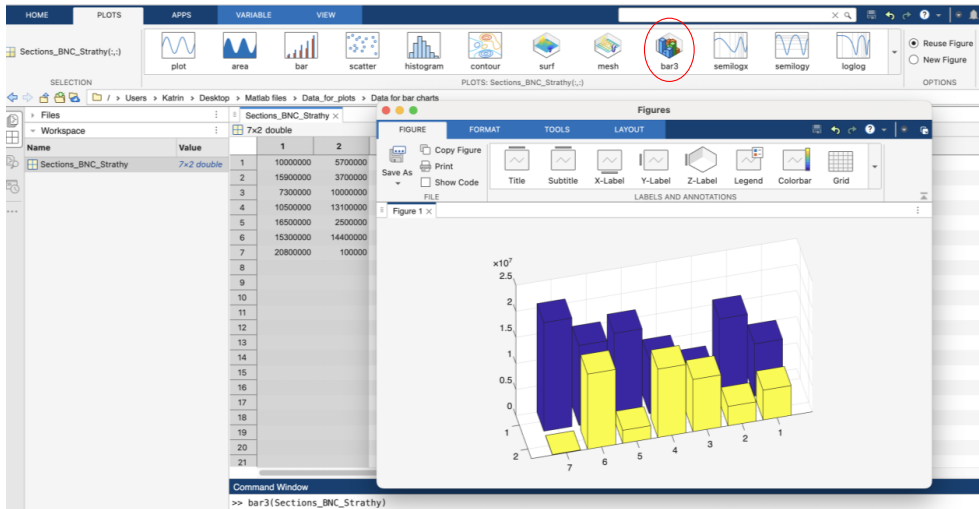


Figure 126: 3-D bar chart for sizes of sections in BNC and Strathy Corpus

## 7.4 Horizontal bar charts

The process of creating and modifying horizontal bar charts is very similar to that of vertical bar charts, as described in Chapter 7.3. A horizontal bar chart might be more convenient in certain situations, such as when the category names (in this case, corpus sections) are long, or when it is easier to compare values along the x-axis rather than the y-axis.

Let us consider the same dataset of corpus section sizes from the BNC and the Strathy corpus (see Table 2 in Chapter 7.3). To create the chart, we begin by creating a numeric variable in MATLAB with two rows, as outlined in Chapter 7.3. We may name this variable `Sections_BNC_Strathy` or choose another name. Once you have done this, select the data and either click the **barh** icon from the toolbar or enter the command `barh(Sections_BNC_Strathy)` in the Command Window. This will generate a horizontal bar chart, where the size information will be displayed along the x-axis and the corpus sections will appear on the y-axis (Figure 127). If you would like to replace the numbers 1 to 7 on the y-axis with the actual names of the corpus sections, follow these steps: First, select the **Axes** properties in the Property Inspector. Then, in the **Ticks** section, click on the three horizontal dots next to the **YTickLabel** field. This will open a table where you can input the names of the corpus sections. Once you have entered the names, they will appear in place of the numbers on the y-axis of your horizontal bar chart (see Figure 127).

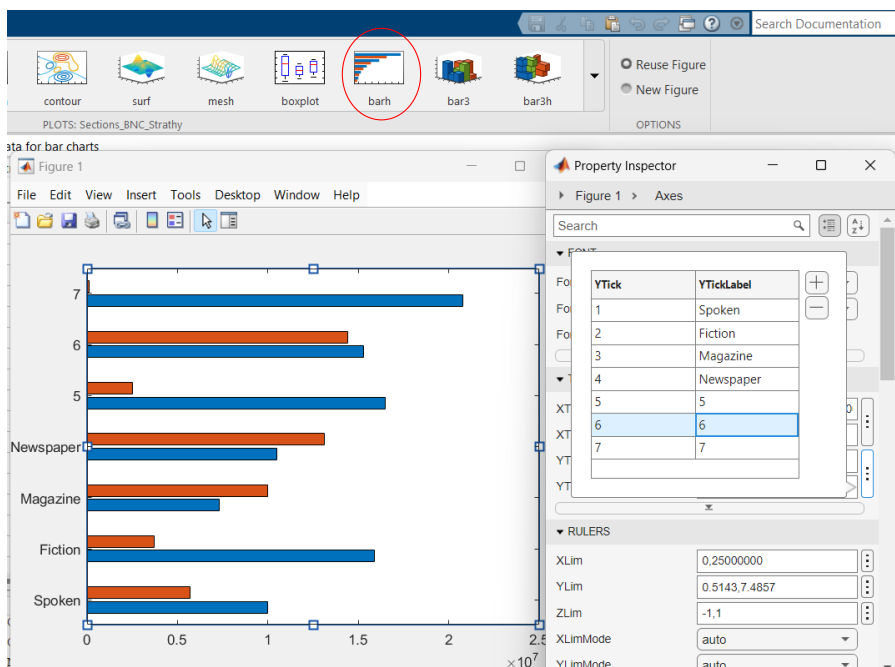


Figure 127: Horizontal bar chart for sizes of sections in BNC and Strathy Corpus

Here, we can also create a 3-D plot by clicking on the **bar3h** icon in the toolstrip or by using the command `bar3h(Sections_BNC_Strathy)`. This could be useful if you want to visualise the data from a different perspective. Similar to the 3-D vertical bar chart, you can interact with the plot by rotating it to view the data from various angles. However, depending on the dataset – and in this case as well – a 3-D bar chart might not necessarily provide a more meaningful representation, but it is worth experimenting to see if it enhances your analysis.

## 7.5 Stacked bar charts

A stacked bar chart is a type of bar plot where each bar represents a category, and the segments within the bar indicate subcategories. The stacked segments visually show how parts contribute to the whole. This is useful when you want to compare not only the total values across categories but also see the composition of those totals. For example, let us assume we want to visualise the distribution of epistemic stance markers such as “I assume”, “I guess” and “I suppose”, which are used to indicate the extent to which a speaker is claiming knowledge about a particular proposition and we aim to compare their frequencies across different corpora from the English-Corpora.org website (e.g., the NOW corpus [Davies 2016–], COCA [Davies 2008–] and the Movie corpus [Davies 2019]). Using a stacked bar chart allows us to visualise the total frequency of the three markers across the corpora as well as the proportion of each marker’s contribution to the overall total within each corpus.

We start by querying and collecting data on the frequencies per million words (fpm) for the three epistemic stance markers from the respective corpora. Once we have gathered the necessary information, we can organise it as shown in Table 3:

Stance marker	NOW corpus (fpm)	COCA Corpus (fpm)	Movie Corpus (fpm)
I assume	1.71	7.93	8.44
I guess	18.08	109.91	279.70
I suppose	6.06	25.77	91.56

Table 3: Frequencies (per million words) of three epistemic stance markers in different corpora

As always, we have the option to work primarily with commands from the Command Window or a script, to use mainly interactive tools or a combination of both. To create a script and to open a new file where you can write your commands, go to the **Home** tab and click on **New Script**. Using a script for this task will make it easy to re-run it or to modify the individual commands in it in the future. A script, for creating a stacked bar chart, for instance, might look like this:

```
% Data for stance markers across different corpora
corpora = {'NOW Corpus', 'COCA', 'Movie Corpus'};
Stance_markers = {'I assume', 'I guess', 'I suppose'};

% Normalised frequencies for each corpus by stance marker (rows: corpora,
columns: stance markers)
data = [1.71, 18.08, 6.06;      % NOW Corpus
        7.93, 109.91, 25.77;   % COCA
        8.44, 279.70, 91.56]; % Movie Corpus

% Create a stacked bar chart
figure;
bar(data, 'stacked');

% Set x-axis labels to the corpora
set(gca, 'xticklabel', corpora);

% Add labels and title
xlabel('Corpora');
ylabel('Frequency per Million');
title('Frequency of epistemic stance markers across different corpora');
legend(Stance_markers);
```

Figure 128 shows the figure that is created when we run this script.

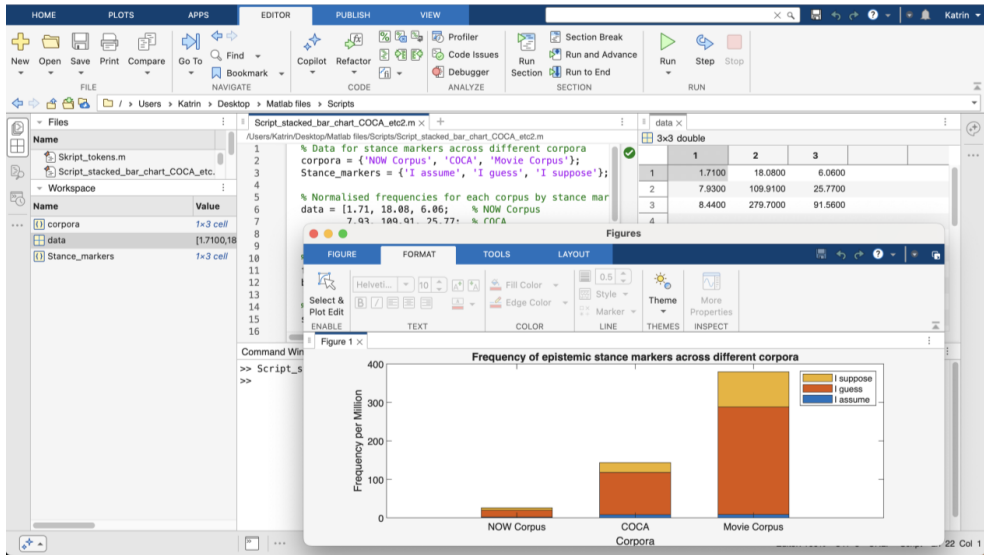


Figure 128: Creating a stacked bar chart with a MATLAB script

Instead of writing a script or coding, we can simply input or import the data directly into MATLAB using the **Variables editor** or **Import Tool**. Once the data is in the workspace, we can visualise it by selecting the desired plot type through MATLAB's **Plots** tab. Choose the **bar (stacked)** or **barh (stacked)** icon from the options under the MATLAB bar plots section to create a stacked bar chart (Figure 129). These options allow you to visualise the data with vertical bars (for **bar**) or horizontal bars (for **barh**).

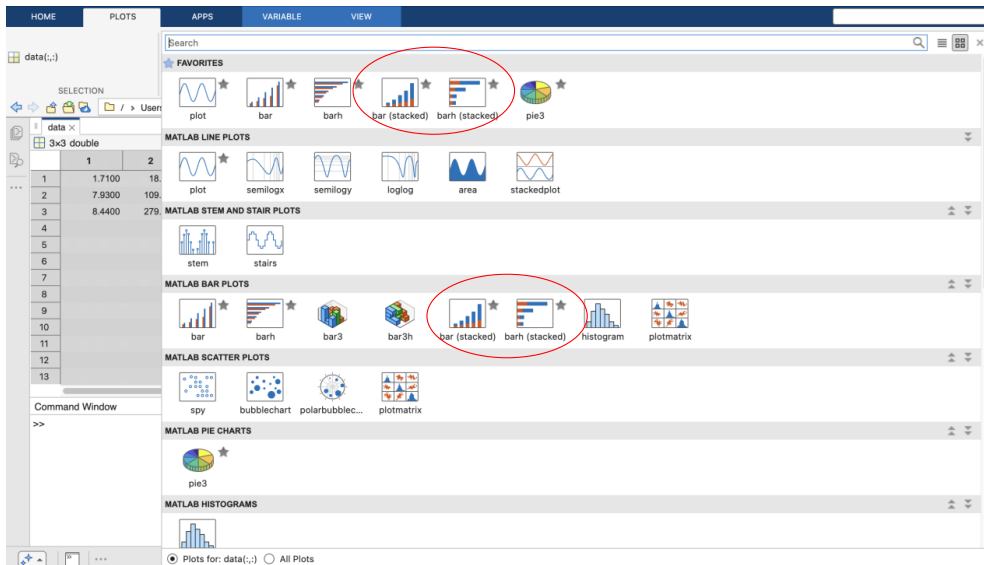


Figure 129: Creating a stacked bar chart using the interactive plotting options

Figure 130 shows a vertical stacked bar chart generated by visualising the data from a numeric matrix, based on the data in Table 3.

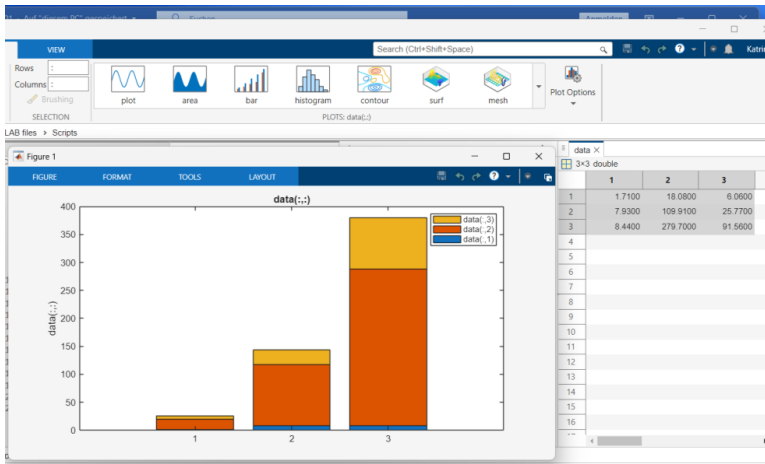


Figure 130: Vertical stacked bar chart created by selecting the respective icon

This chart, generated by selecting the **bar (stacked)** icon, still requires adjustments such as adding a title, axis labels and a legend. These can be added by clicking the corresponding icons in the Figure tab toolstrip. X-tick labels for the corpus names can be added by selecting this option under the Ticks section in the Property Inspector (Figure 131). The interactive interface lets us create the plot and customise its appearance – if we prefer this option, all without needing to write any code.

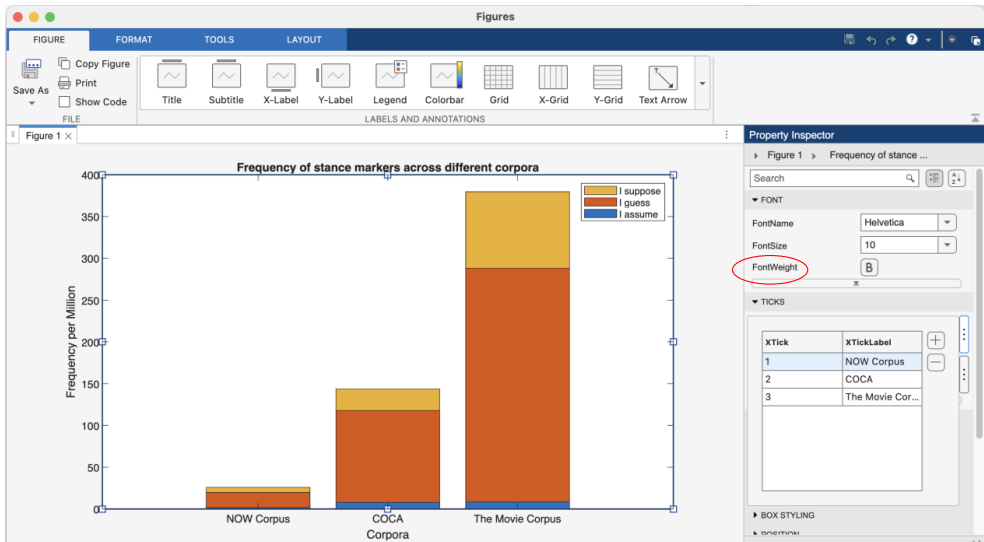


Figure 131: Updated visualisation of stacked bar chart with title, axis labels and legend

After examining these examples of bar charts, we will now look at other examples of different plot types in the following sections.

## 7.6 Pie charts

Pie charts are a type of graphical representation that display the proportional contributions of different categories within a whole. They may be used to illustrate the distribution of text categories, linguistic features or other categorical data within a corpus. Let us take the following example. We want to create a pie chart that displays the percentage contributions of male and female speakers to the British Academic Spoken English corpus (BASE, Thompson & Nesi 2001) that we have briefly discussed in Chapter 4.3 and that is available at the Sketch Engine platform (<https://www.sketchengine.eu/british-academic-spoken-english-corpus/>). The entire BASE corpus hosted on Sketch Engine contains a total of 1,520,789 tokens. To determine how many tokens in the corpus are attributed to male or female speakers, we can either refer to the Corpus Info section of the BASE corpus or run queries, applying filters through the Text Types function to include only texts with specific gender annotations. There are 1,116,423 tokens attributed to male speakers, 392,708 tokens attributed to female speakers, and the remaining tokens (which we can calculate or look up in the Corpus Info section) are labelled with categories such as “unknown.”

Now, we want to create a pie chart in MATLAB to visualise the distribution of these token counts across the male and female categories as well as the remaining tokens. We can start by defining the data for each category: tokens from texts by male speakers, female speakers and those labelled as unknown. We create new numeric variables in MATLAB by using the **New Variable** icon or by writing commands to define these variables, e.g.:

```
male_tokens = 1116423;
female_tokens = 392708;
unknown_tokens = 1520789 - male_tokens - female_tokens;
```

The third command in this sequence for `unknown_tokens` calculates the number of tokens labelled as “unknown” or similar categories with no gender annotation in the BASE corpus. It does so by subtracting the sum of `male_tokens` and `female_tokens` from the total number of tokens in the corpus, which is 1,520,789. It stores the result in the variable `unknown_tokens`, which represents the remaining tokens with unknown gender information. This works because we can perform arithmetic operations on the numerical values stored in the variables. Since the total number of tokens is known, subtracting the counts for male and female tokens from this total gives us the remaining tokens (cf. Chapter 8.1.3).

The pie chart function in MATLAB requires one numeric array or vector as input. Therefore, we can use the following command to create a vector that holds the numerical values corresponding to the different categories we want to visualise in our pie chart.

```
data = [male_tokens, female_tokens, unknown_tokens];
```

This helps to organise the data and to group the values in one place so that we can refer to that vector. Additionally, we can now use operations like `sum(data)` to see the total number of tokens, `max(data)` to find the category with the most tokens or other mathematical operations on the entire dataset (cf. also Chapters 8.1.2 and 8.2.2).

We can save these commands for defining the data in a script by going to the **Home** tab and clicking on **New Script**. This allows us to reuse or modify the code later as needed. When we

have the variables in our MATLAB workspace, we can generate a pie chart using MATLAB's commands, interactive plotting tools or a combination of both. We can use MATLAB's traditional built-in `pie()` function. We can also use the rather similar `piechart()` function instead which is part of MATLAB's newer graphics system and is designed to create pie charts with enhanced features and customisation options compared to the traditional `pie()` function. Some details regarding the customisation options for these options are different.<sup>20</sup>

So, if we want to create a pie chart from the data, there are several options available. Using the traditional function by entering `pie(data);` generates a pie chart directly from the data array. We can also specify labels for the slices by providing a second argument. However, this approach does not return a handle to the pie chart object, meaning it does not give us direct access to the chart's components for further modifications – which can make customisations a bit more difficult.

An alternative is to use the `piechart(data);` command. While both `pie` and `piechart` produce pie charts with a similar layout, the `piechart` function is generally recommended because it creates a `PieChart` object, which offers greater flexibility for customisation. For example, `PieChart` objects have properties that allow us to adjust slice colours, control the arrangement of slices (clockwise or counterclockwise) and set the position of the first slice.

To gain full control over the chart after it is created, we can also assign the output to a variable – such as `p = pie(data);` or `p = piechart(data);`. This variable acts as a handle (a reference to the chart object), which allows us to access and modify its properties programmatically. Depending on your needs, choosing the appropriate function and capturing the chart handle can make a difference in how easily you can fine-tune your visualisation.

Of course, you can also just select your variable data in the workspace and simply click on the **piechart** icon in the toolstrip. As discussed in Chapter 7.2, you can add frequently used plot types to your **Favourites** for quicker and more convenient access. This feature allows you to mark selected plot types with a star icon and place them in a dedicated list at the top of the **Plots** section, so that you do not have to search for them every time (Figure 132).

---

<sup>20</sup> cf. <https://www.mathworks.com/help/matlab/ref/pie.html> and <https://www.mathworks.com/help/matlab/ref/piechart.html>

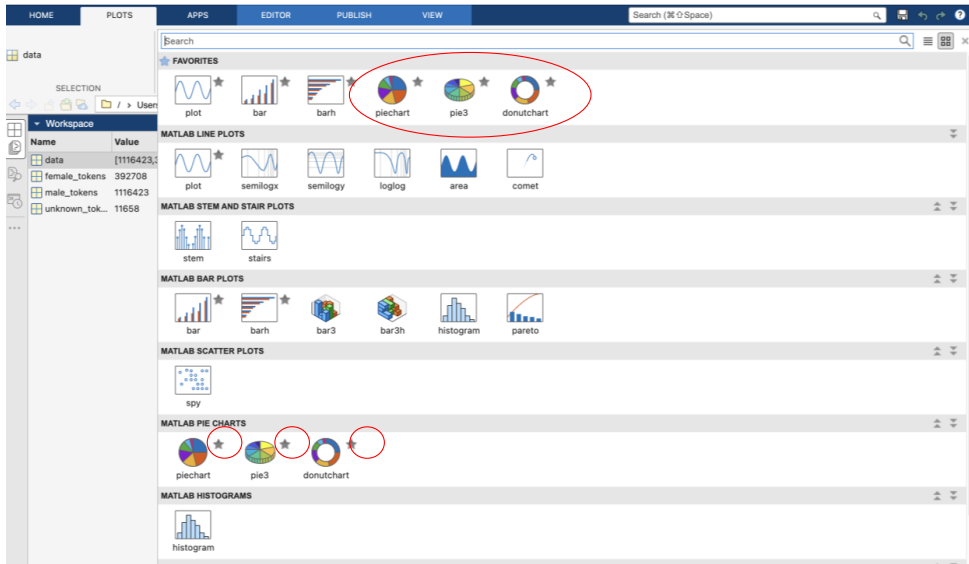


Figure 132: Selecting specific charts to appear among the favourite plot types

The difference between clicking on the icon for the pie chart in MATLAB after selecting the variable data and the other option of typing a command lies in some details with regard to how the chart is created and how you can access and customise the resulting chart object. When you click on the **piechart** icon in the **Plots** section of the MATLAB toolbar, MATLAB will automatically generate a pie chart based on the selected variable (`data`) and display it in a figure window. MATLAB will run the command `piechart(data)` behind the scenes and you will see this command appear in the Command Window. This interactive method does not assign the pie chart to a specific variable (like `p`) which you might prefer if you are planning to modify the chart programmatically in a certain way after it has been created. However, you will still be able to make various customisations directly through the figure's **Properties Inspector** (e.g., adding a title, labels and a legend, cf. Figure 133), and you can store your chart as a figure that you can continue modifying later.

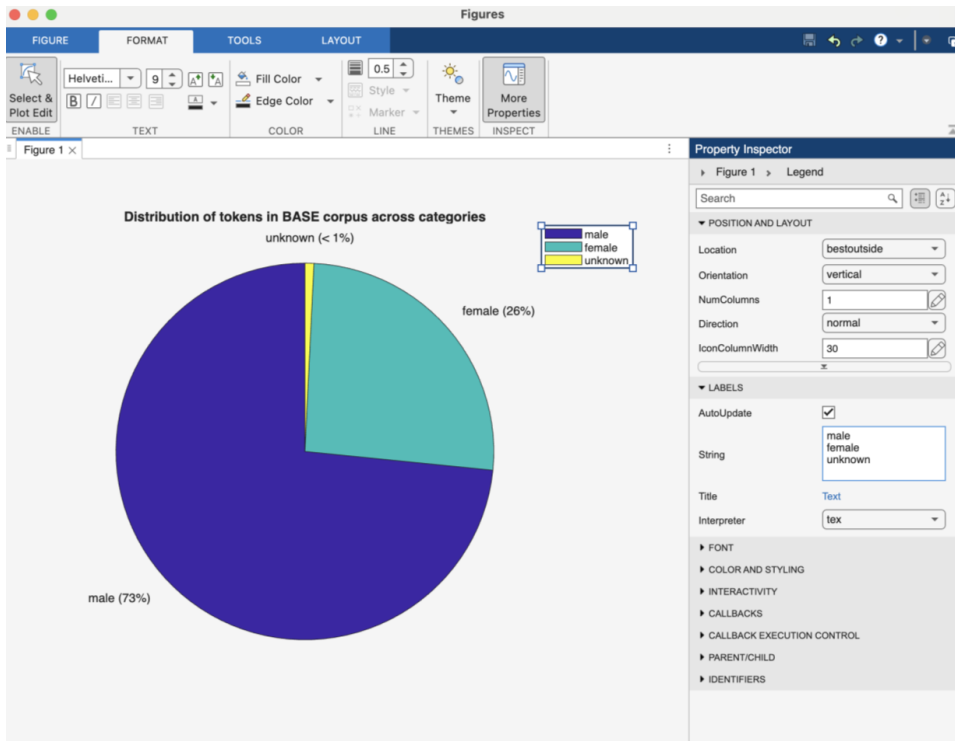


Figure 133: Modifying a pie chart via the Property Inspector

When you prefer to work with commands, you can, for instance, create the pie chart for your data as outlined above and add labels and a title via commands as follows.

```
p = piechart(data);
p.Labels = {'Male', 'Female', 'Unknown'};
title('Distribution of tokens in BASE corpus across categories');
```

Here, the labels for the pie chart are directly set through the `p.Labels` property of the `piechart` object. This method allows for easy assignment of labels for each pie slice via a command (in this case, 'Male', 'Female' and 'Unknown'). Figure 134 shows the result of our sequence of commands.

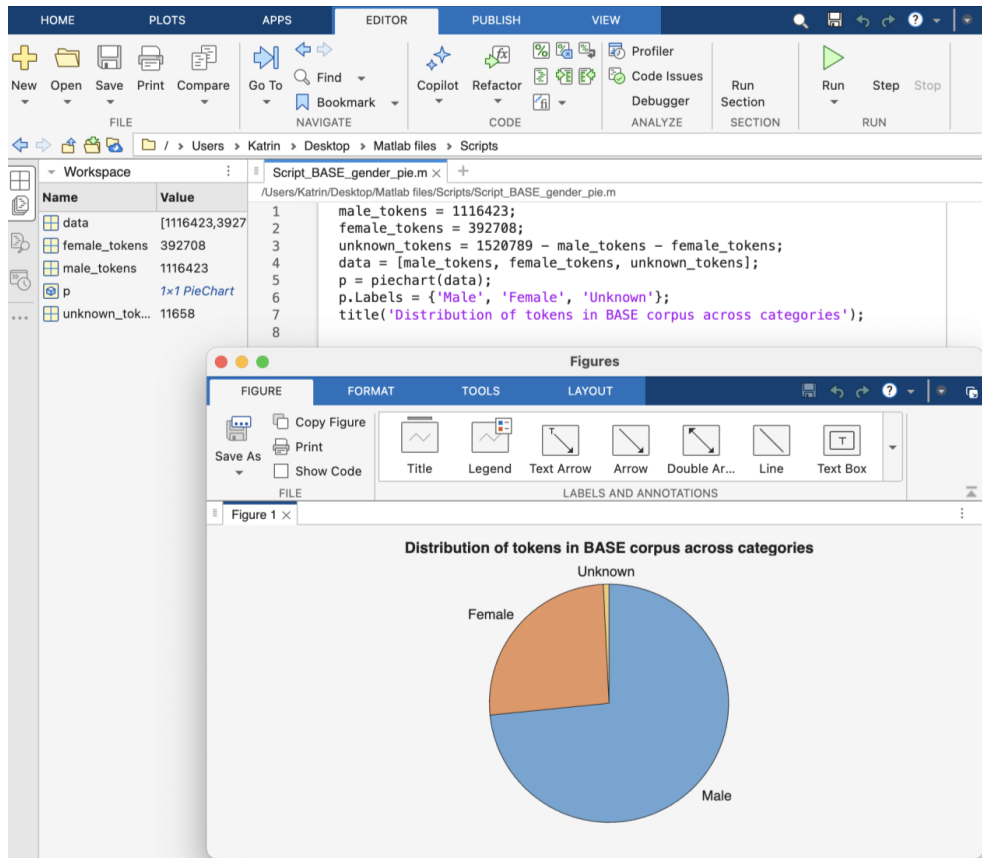


Figure 134: Creating and customising a pie chart for BASE corpus data by using commands

Alternatively, try out the following series of commands and see what the differences are.

```

male_tokens = 1116423;

female_tokens = 392708;

unknown_tokens = 1520789 - male_tokens - female_tokens;

data = [male_tokens, female_tokens, unknown_tokens];

pie(data);

legend('male', 'female', 'unknown', 'Location', 'bestoutside');

title('Distribution of tokens in BASE corpus across categories');

```

Defining the data here is the same, but in this case, the pie chart is created using the `pie()` function, and we do not assign a variable `p` to store the chart object as a variable in the workspace. Moreover, labels are not added directly to the pie chart slices. Instead, a legend is used with the `legend()` function to label the categories, specifying the entries for 'male', 'female' and 'unknown' (Figure 135).

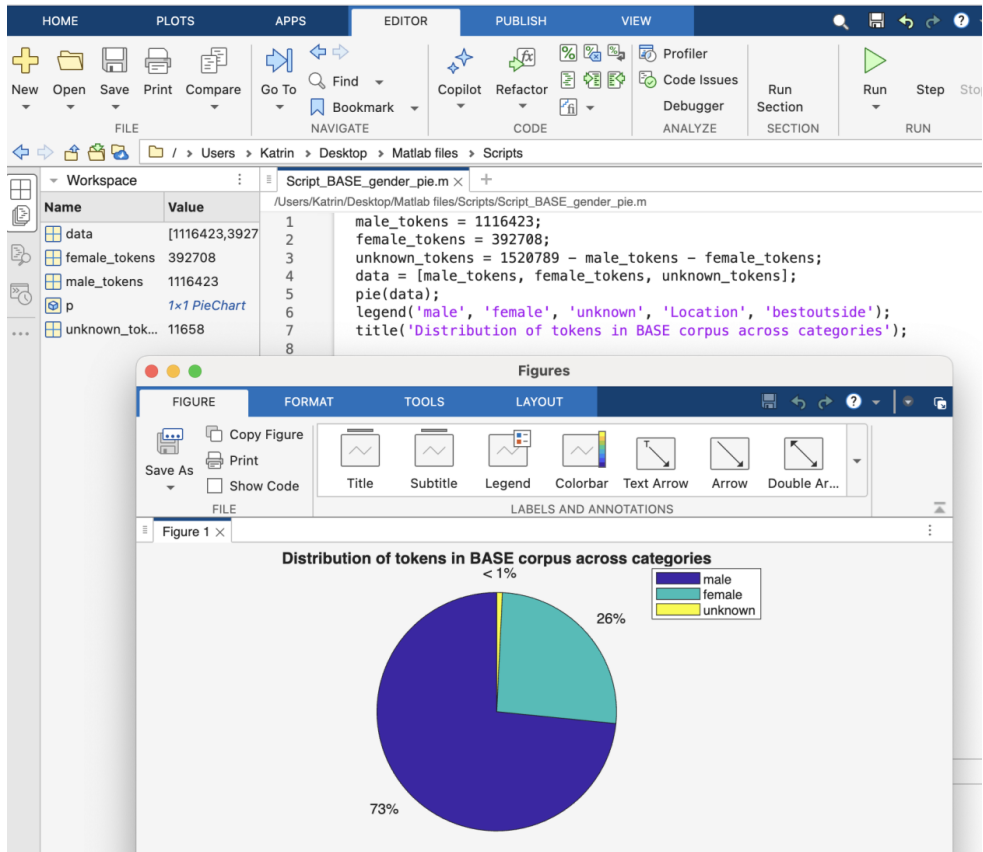


Figure 135: Alternative way of creating and customising a pie chart for BASE corpus data by commands

Note that the 'Location', 'bestoutside' option in the legend() function places the legend outside the plot area, ensuring it does not overlap with the chart itself.<sup>21</sup> Otherwise, MATLAB will automatically determine an appropriate space, typically to the right or top of the chart.

MATLAB also provides functions and interactive icons to create 3-dimensional pie charts. Either type a command such as `p = pie3(data)` or `p = donutchart(data)`; cf. Figure 136.

<sup>21</sup> cf. <https://www.mathworks.com/help/matlab/ref/matlab.graphics.illustration.legend-properties.html>

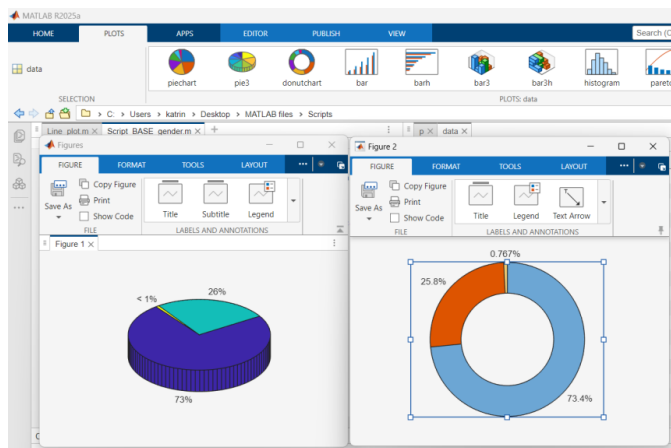


Figure 136: Creating a 3D pie chart and a donut chart for BASE corpus data

We will now explore a different dataset derived from the metadata of the Royal Society Corpus (RSC), which comprises scientific articles from journals such as the Royal Society’s *Philosophical Transactions*, spanning from the 1660s to the 1990s (Kermes et al. 2016, Fischer et al. 2020, cf. Chapter 4.4). This dataset will serve as another example for illustrating the creation of a specific pie chart. Menzel’s (2025) analysis of historical scientific publications from the RSC focuses on the rise of microbiology and highlights a notable shift: the increasing presence of female authors in this field in prestigious scientific journals beginning in the late 1880s. Between the 1880s and 1920, 57 microbiology articles authored or co-authored by women can be found in the RSC. Early microbiology texts can typically be found via their annotated text topic ‘immunology’ in the RSC. The metadata of the texts also includes author names, which makes it possible to identify contributions by both women and men. Focusing on individual contributions, Table 4 presents the names of women authors and the number of microbiology texts they authored or co-authored in scientific journals during this time span, as part of the Royal Society Corpus. For simplicity, all authors with only one paper have been grouped here into a single category called “Other authors (1 paper each).” This information can be collected by examining metadata in the corpus related to article topics and author names. The RSC Version 6.0.1 Open is accessible through the CQPweb interface of Saarland University at <https://corpora.clarin-d.uni-saarland.de/cqpweb/>.

Name	Number of articles
Grace C. Frankland	5
Maria Dawson	4
Harriette Chick	2
Muriel Robertson	7
Jean White	2
Mary Elizabeth Bruce	31
Other authors (1 paper each)	6

Table 4: Number microbiology articles by female (co-)authors in RSC

Creating a pie chart from this data allows us to visualise the distribution of microbiology articles authored or co-authored by women, as found in the Royal Society Corpus, during the analysed period. It provides a visual representation that highlights whether any female authors have a disproportionately higher number of articles compared to others, which could suggest bias when analysing women’s language in these early microbiology publications. While this information

can also be derived from Table 4 or calculated, for example, by computing percentages of total articles for each author, a pie chart offers a more intuitive visual representation of the data. This visual format may make it easier to compare proportions and spot any disparities between authors quickly. For example, it may be much easier to see immediately how large Mary Elizabeth Bruce's contribution is relative to the other female authors in this time period.

We will now define the data by creating a variable for the author names (a cell array suitable for storing strings or other data types) and another for the number of articles (a numeric array). Next, we will plot a pie chart, add labels and set an appropriate title. To keep the labels concise, only the authors' surnames will be used. Optionally, we may want to adjust the font size for better readability. Let us try to incorporate all of these desired settings into a script. A possible way to write this script could look like the following:

```
% Define the data
authors = {'Frankland', 'Dawson', 'Chick', 'Robertson', 'White', 'Bruce',
'Other authors'};
articles = [5, 4, 2, 7, 2, 31, 6];

% Create pie chart with the defined authors
figure;
pie(articles);

% Add a legend using the authors' surnames
legend(authors, 'Location', 'bestoutside');

% Add title to the chart
title('Microbiology articles by female (co-)authors in RSC (1880-1920)');

% Adjust font size of title and labels
set(gca, 'FontSize', 14);
```

When you run this script, you will create a pie chart as in Figure 137.

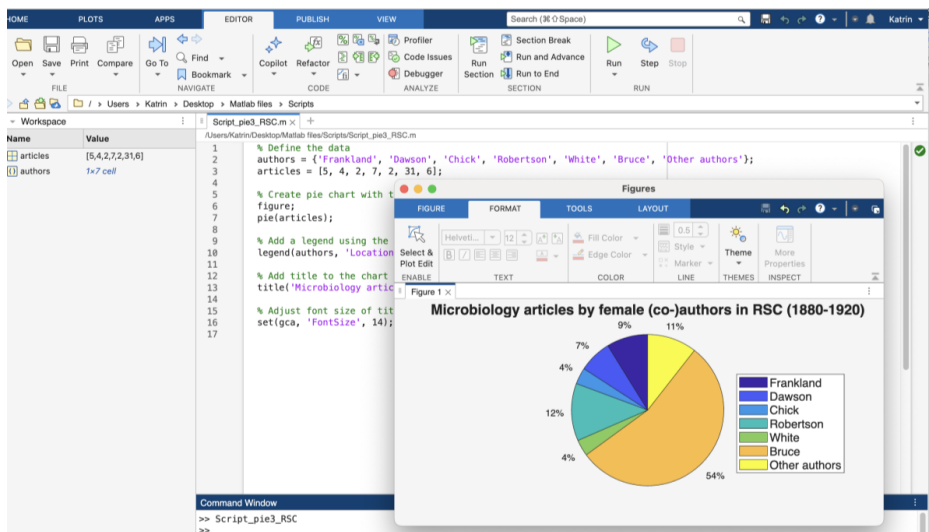


Figure 137: Creating a customised pie chart from RSC data using a script

For those new to programming in MATLAB, instead of writing an entire script to configure every desired setting of a pie chart, it may here be much easier to use the **piechart** icon in the **Plots** tab to generate the pie chart automatically. Once the variables are defined and accessible in your workspace, simply select them and click on the icon from the toolbar. Afterwards, necessary customisations such as adding or modifying any labels and setting a title can be made interactively using the options in the **Figure** and **Format** tabs, as well as the **Property Inspector** (Figure 138).

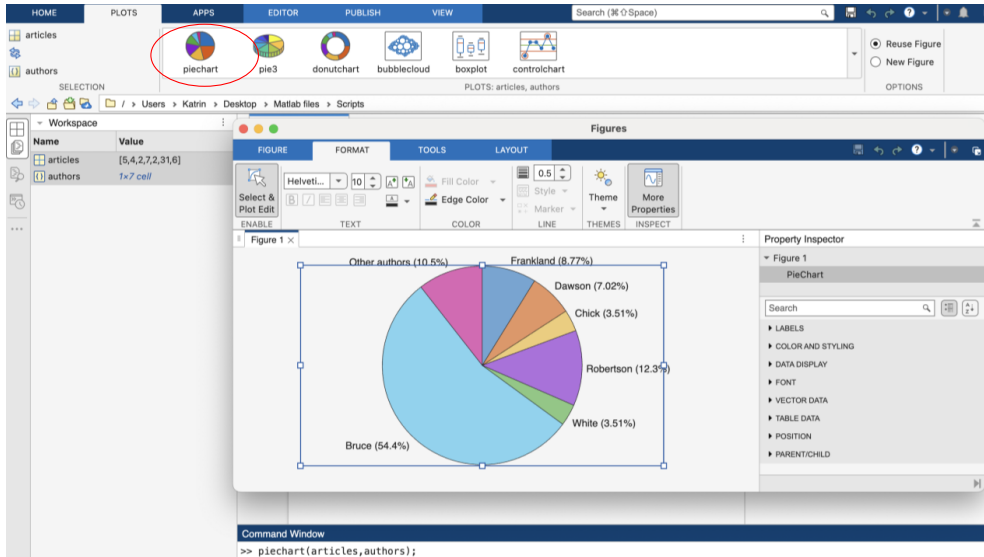


Figure 138: Customising a pie chart created by selecting the respective icon from the Plots tab

Many details of pie charts can be adjusted in MATLAB. You can explore the various options in the Property Inspector to experiment with them. Navigate through the sections, use the search function or arrange the options in alphabetical order to locate the desired customisation option. For example, using “exploded wedges” will offset specific slices of the pie, making them stand out for emphasis (Figure 139).

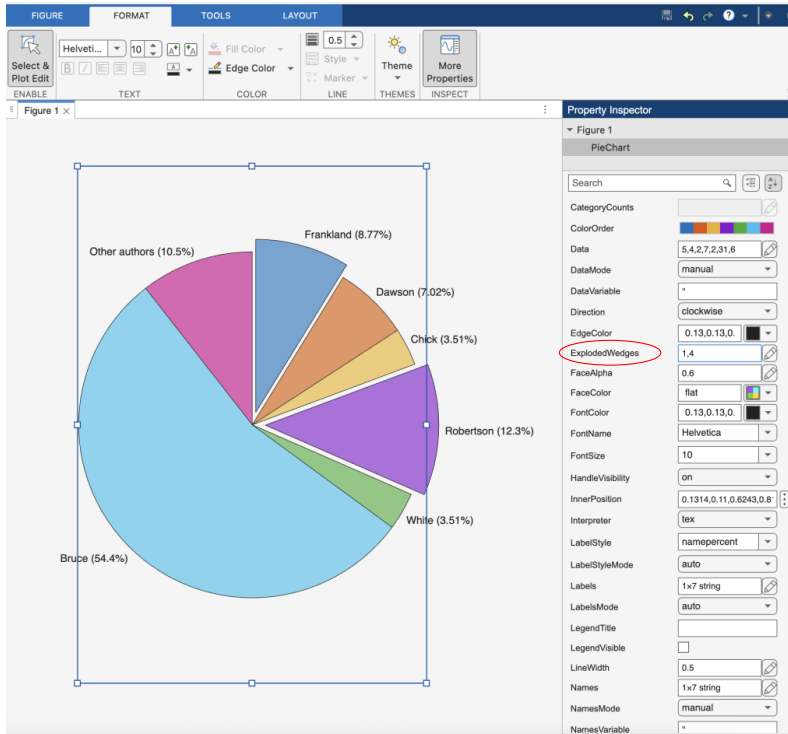


Figure 139: Pie chart with exploded wedges to highlight specific slices for emphasis

As you can see, creating pie charts in MATLAB to visualise different types of corpus data is rather quick and simple, with many customisable details that can be easily adjusted.

## 7.7 Line plots

In corpus linguistics, line plots are probably less commonly used compared to other methods like bar charts, pie charts, scatter plots or box plots. Line plots can be particularly useful when you have a continuous variable like the subsequent years or decades in a diachronic corpus.

Let us examine, for instance, the frequency development of the “start” + *-ing* construction throughout the 20th century in the Corpus of Historical American English (COHA, Davies 2010, <https://www.english-corpora.org/coha/>). To do so, we can enter the following query into the COHA search interface to find instances of the lemma “start” followed by an *-ing* participle across the entire corpus: `[start]_v?g`. By selecting the “chart view” option in the query interface, we can view the results, which include both absolute and normalised frequencies per decade (Figure 140).

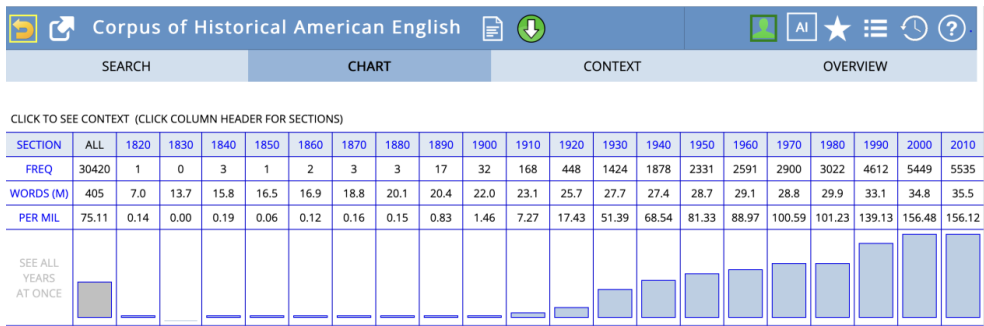
Figure 140: Frequency of “start” + *-ing* form in COHA

Table 5 presents the frequency values for each decade from 1900 to 1990, a period during which this construction notably increased in frequency.

Decade	“start”+ V-ing (fpm)
1900	1.46
1910	7.27
1920	17.43
1930	51.39
1940	68.54
1950	81.33
1960	88.97
1970	100.59
1980	101.23
1990	139.13

Table 5: Normalised frequency of “start” + *-ing* form in COHA corpus throughout 20th century

To create a line plot showing the number of tokens over the decades, we first need to define the variables: the decades for the x-axis and the corresponding frequency values for the y-axis:

```
x_decades = [1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990];
y_freq_start_V_ing = [1.46, 7.27, 17.43, 51.39, 68.54, 81.33, 88.97, 100.59, 101.23, 139.13];
```

If these two variables are in our workspace, we can plot them directly from there by selecting both and clicking on the **plot** icon from the toolstrip. In the background, the `plot` function is used to create the line plot, and you will see the respective command in the Command Window. The syntax `plot(x, y)` creates a 2D line plot of the values in `y` against their corresponding values in `x`. In our case, the variable `x_decades` is used for the x-axis, and `y_freq_start_V_ing` provides the data for the y-axis. Once we have generated this plot (Figure 141), we can interactively add labels and a title or make other adjustments using the built-in plot editing tools.

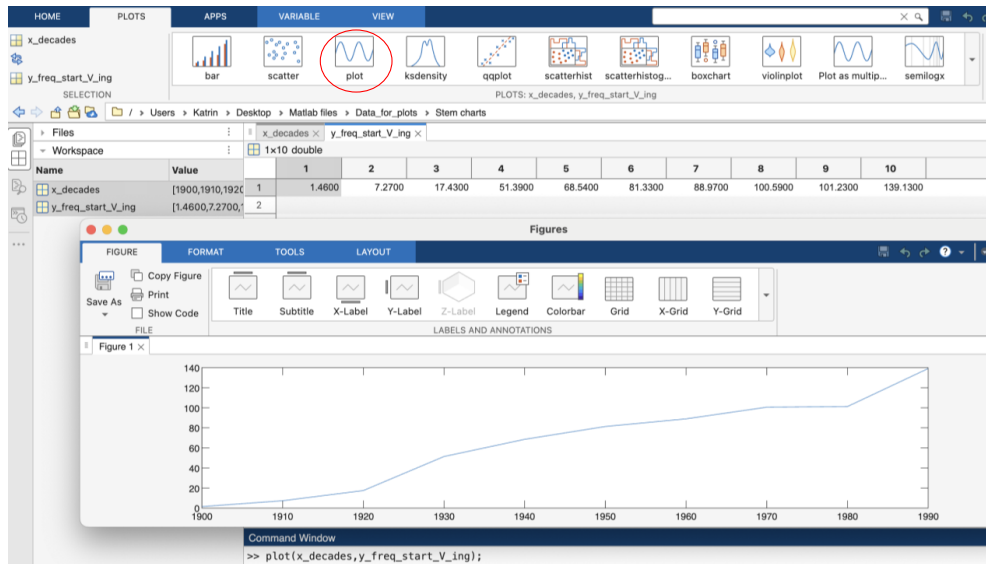


Figure 141: Line plot for frequencies of “start” + *V-ing* in COHA

To create a line plot for these data, you can also use a sequence of commands, including the `plot` function, or a script like the one below:

```
% Data for plotting
x_decades = [1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990];
y_freq_start_V_ing = [1.46, 7.27, 17.43, 51.39, 68.54, 81.33, 88.97, 100.59, 101.23, 139.13];

% Create the line plot
figure;
plot(x_decades, y_freq_start_V_ing, 'Marker','-+');
% Line plot with + markers, x_decades on the x-axis and y_freq_start_V_ing
% on the y-axis

% Add title and labels
title('Frequency of start + V-ing in COHA throughout 20th century');
xlabel('Decade');
ylabel('start + V-ing, FpM');
```

In this script, the `'-+'` argument specifies a line plot with plus signs at each data point. You can also use other symbols to customise the markers – for example, `'-o'` will display circles instead of plus signs. If you leave out this argument, the plot will show a plain line without any markers. The command `figure;` before `plot` is optional. If omitted, MATLAB will reuse the current figure or create a new one automatically if none exists. However, if you create multiple plots without `figure;`, the latest plot will overwrite the previous one in the same figure window.

Using a script like this can be useful if you plan to reuse it in future projects with your customisations already in place. If we run this script, we will obtain a line plot as shown in Figure 142.

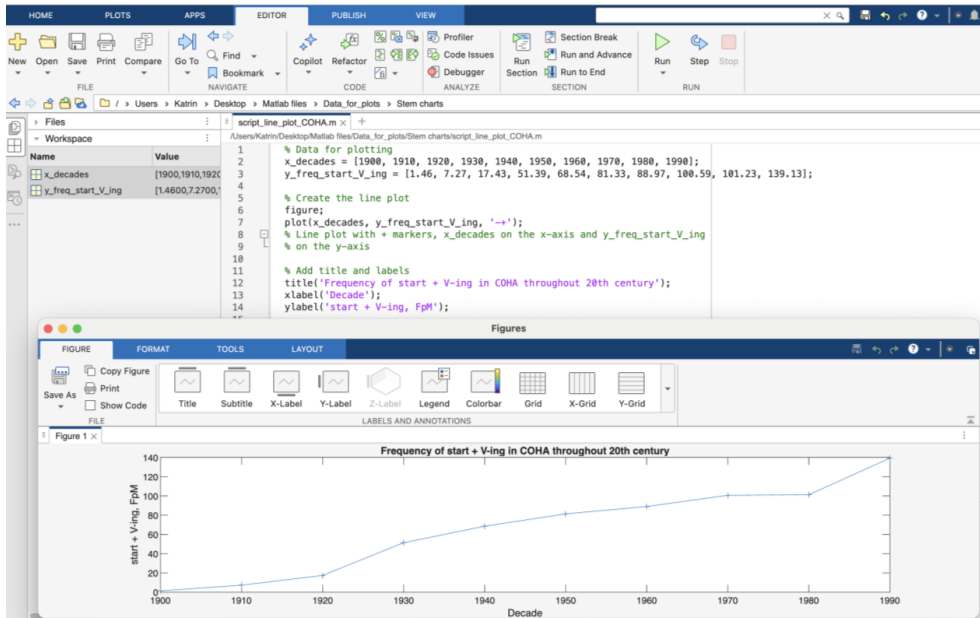


Figure 142: Example of a line plot in MATLAB

By default, MATLAB automatically selects the scaling for the x- and y-axes. We can customise these settings either through commands or by using the Property Inspector. Once the figure is created, we may want to adjust the minimum and maximum values of the axes interactively, as well as modify the intervals between tick marks. For example, to set the y-axis to range from 0 to 200 in steps of 50, open the Property Inspector, select the y-axis in the figure and edit the values directly in the **YTick** and **YLim** fields (cf. Figure 143).

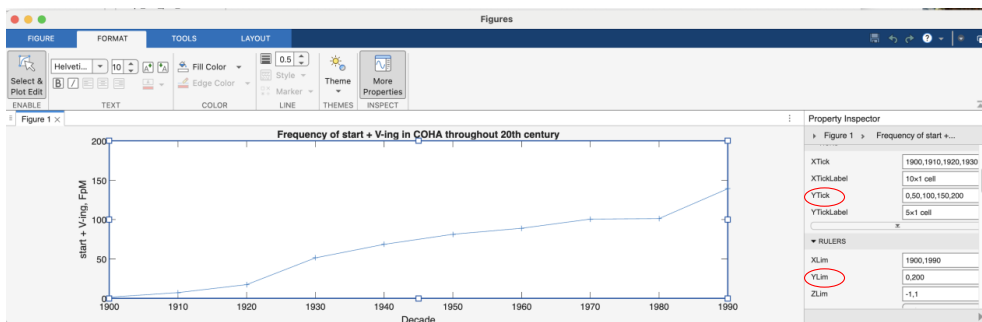


Figure 143: Modifying axis limits via the Property Inspector

Additionally, MATLAB offers commands like `xlim` and `ylim` to programmatically adjust the axis limits. For instance, adding the command `ylim([0 200]);` to your script would set the y-axis to display values between 0 and 200.<sup>22</sup>

<sup>22</sup> cf. [https://www.mathworks.com/help/matlab/creating\\_plots/change-axis-limits-of-graph.html](https://www.mathworks.com/help/matlab/creating_plots/change-axis-limits-of-graph.html)

To show descriptive statistics for the x- and y-data of a plot in MATLAB, you can go to the figure and select the **Tools** tab in the top menu. On the respective toolstrip, you will find an option labelled **Data Statistics**. Clicking on this option opens the Data Statistics dialogue box. This dialogue box will display various descriptive statistics for your dataset, including the minimum and maximum values, median, mean and standard deviation. You can choose which statistics you would like to display on the plot by selecting the checkboxes next to each statistic. For example, if you want to plot the median of your above-mentioned frequency data of the frequencies of the “start” + *-ing* construction, you can select the “Median” checkbox in the Y column. Once you do this, MATLAB will plot a horizontal line at that value on the graph to indicate the median (74.94) of the y-data (Figure 144). This may help you visualise where the median of your y-values lies in relation to the entire dataset. After selecting the statistics you want, you can click **Save to Workspace...** to save these values for further analysis or use in other MATLAB functions. This will allow you to work with the statistics outside of the plot ([https://www.mathworks.com/help/matlab/data\\_analysis/descriptive-statistics.html](https://www.mathworks.com/help/matlab/data_analysis/descriptive-statistics.html)).

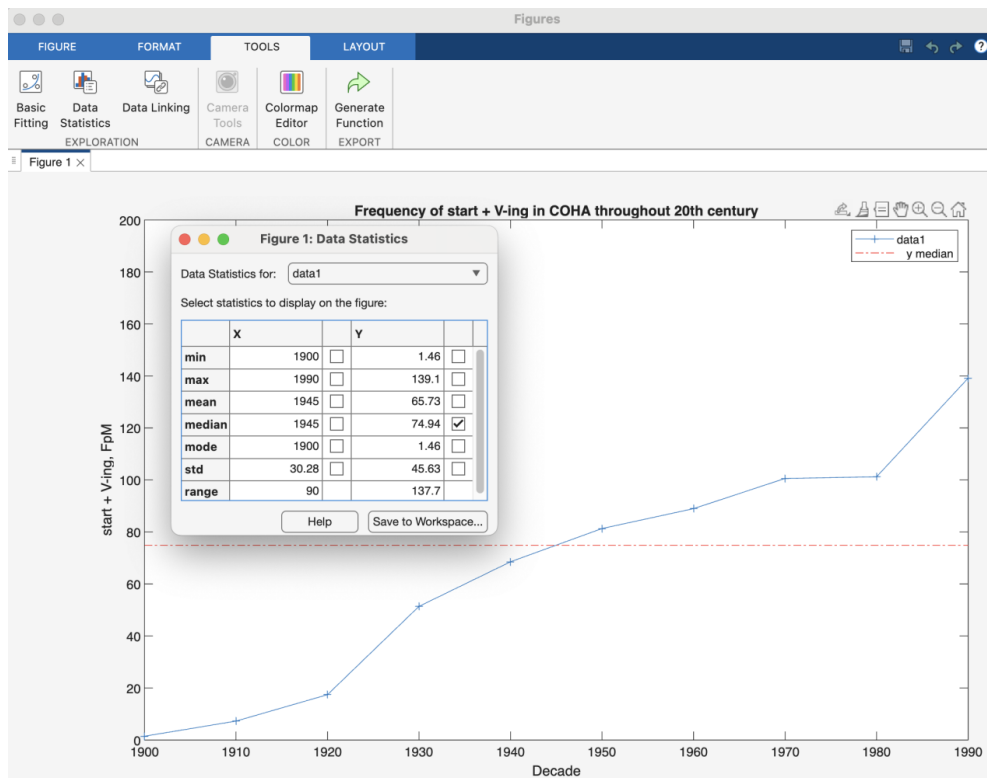


Figure 144: Plotting descriptive statistical values

## 7.8 Stem plots

A stem plot in MATLAB visualises individual data points in a sequence using vertical lines (“stems”) and markers at the data values (circles by default). The function `stem(y)` or a click in the **stem** icon on the toolbar plots the data as stems that extend from a baseline along the x-axis, with the height of each stem representing the corresponding data value. A stem plot appears somewhat similar to a bar chart, as both use vertical elements to represent magnitude.

In the following example, the linguistic feature under examination is the use of elliptical constructions that function as cohesive devices – i.e., the reduction of linguistic elements that link back cross-clausally to a textual antecedent. This includes nominal ellipses, as in *U.S. citizens don't hold foreign currencies here; they just hold domestic [ ]*, as well as verbal and clausal ellipses, illustrated by examples such as *We never get it all right in any year, and probably never will [ ]*, and *Who was playing the piano? – Peter [ ] / Peter was [ ]*. The brackets indicate the omission of textual elements that can be inferred from the preceding context, as the respective antecedents are provided earlier in the text (cf. Menzel 2017 for a detailed explanation, see also Chapter 7.9 for more examples).

As a dataset we will use a corpus called GECCo (German-English Contrasts in Cohesion, cf. Kunz et al. 2021). It was created as a resource for contrastive studies of cohesive devices and has been annotated for all types of cohesive devices drawing on Halliday and Hasan's (1976) model of cohesion, as adapted for crosslinguistic annotation. The GECCo corpus can be queried for annotated types of the five major types of cohesion (co-reference, substitution, ellipses, conjunctive relations and lexical cohesion) and their subtypes using the CQP corpus query language. Various corpus versions of GECCo are available at Saarland University's CQPweb installation upon request, cf. <https://fedora.clarin-d.uni-saarland.de/gecco/documentation.html>.

For this example, we will focus on five specific registers from the English section of the GECCo corpus. These selected registers, drawn from the original (non-translated) texts in the corpus (EO\_GECCo), include academic lectures (ACADEMIC), instruction manuals (INSTR), letters to shareholders (SHARE), prepared speeches (SPEECH) and corporate websites (WEB). The normalised frequency data for cohesive ellipses per text in these five registers has already been extracted and is presented below in Table 6.

Text GECCo EO (corpus text ID)	Cohesive ellipses (per 1,000 tokens)	Text GECCo EO (corpus text ID)	Cohesive ellipses (per 1,000 tokens)
EO ACADEMIC 001	2.8	EO SHARE 011	0
EO ACADEMIC 002	3.13	EO SHARE 012	0
EO ACADEMIC 003	0.93	EO SHARE 013	1.17
EO ACADEMIC 004	0	EO SPEECH 001	0.75
EO ACADEMIC 005	0.52	EO SPEECH 002	1.43
EO ACADEMIC 006	1.95	EO SPEECH 003	0
EO ACADEMIC 007	0.52	EO SPEECH 004	0.29
EO ACADEMIC 008	1.41	EO SPEECH 005	0.4
EO ACADEMIC 009	1.18	EO SPEECH 006	0
EO ACADEMIC 010	1.64	EO SPEECH 007	0
EO INSTR 001	0	EO SPEECH 008	0
EO INSTR 002	0.32	EO SPEECH 009	1.16
EO INSTR 003	0.28	EO SPEECH 010	1.08
EO INSTR 004	0.28	EO SPEECH 011	0
EO INSTR 005	0	EO SPEECH 012	0.49
EO INSTR 006	0	EO SPEECH 013	1.13
EO INSTR 007	0	EO SPEECH 014	0.38

Text GECCo EO (corpus text ID)	Cohesive ellipses (per 1,000 tokens)	Text GECCo EO (corpus text ID)	Cohesive ellipses (per 1,000 tokens)
EO INSTR 008	0.27	EO WEB 001	0
EO INSTR 009	0.26	EO WEB 002	0
EO INSTR 010	0	EO WEB 003	0.48
EO SHARE 001	1.09	EO WEB 004	0.59
EO SHARE 002	0	EO WEB 005	1.04
EO SHARE 003	0	EO WEB 006	1.23
EO SHARE 004	2.04	EO WEB 007	0.27
EO SHARE 005	0.22	EO WEB 008	0
EO SHARE 006	0	EO WEB 009	0.55
EO SHARE 007	0	EO WEB 010	0.39
EO SHARE 008	0.54	EO WEB 011	0
EO SHARE 009	0.43	EO WEB 012	0.24
EO SHARE 010	0.37		

Table 6: Cohesive ellipses per text in five registers from English part of GECCo corpus

You can either copy these frequencies directly into a command for a numeric variable or first save them in an Excel table and then import them into MATLAB from there. For the complete frequency data of ellipses per text and register across all texts from all corpus sections in the GECCo corpus, you may also refer to Menzel (2017: 452ff).

Here is an example of how you might assign these values to a variable in MATLAB via a command (feel free to use a shorter variable name for your own convenience):

```
EOEllipses_acad_instr_share_speech_web =
[2.80;3.13;0.93;0;0.52;1.95;0.52;1.41;1.18;1.64;0;0.32;0.28;0.28;0;0;0;0.27
;0.26;0;1.09;0;0;2.04;0.22;0;0;0.54;0.43;0.37;0;0;1.17;0.75;1.43;0;0.29;0.4
0;0;0;0;1.16;1.08;0;0.49;1.13;0.38;0;0;0.48;0.59;1.04;1.23;0.27;0;0.55;0.39
;0;0.24];
```

To plot these data as a stem plot, click on the **stem** icon after selecting your variable in the Variables editor. We will focus solely on plotting the values, without including the names of each text, as it would be impractical to display all the text names. For simplicity, the text with the corpus text ID EO\_ACADEMIC\_001 will represent data point 1 on the x-axis, with each subsequent text shown in Table 6 corresponding to the next data point. We can include lines in the plot to separate the different text registers visually if needed.

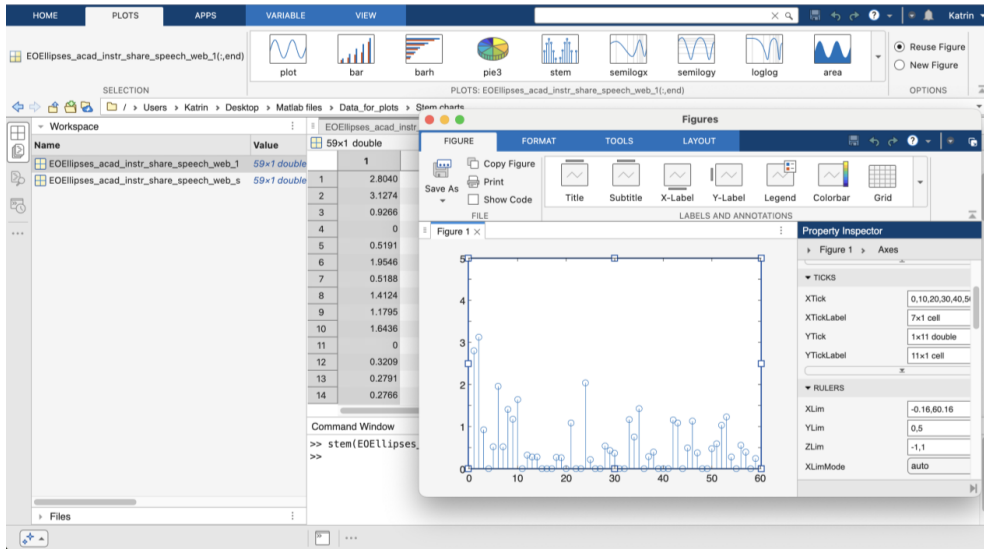


Figure 145: Stem plot for ellipses in five registers from GECCo corpus

The values in this stem plot range from 0 up to slightly above 3 (for the second text, i.e., EO\_ACADEMIC\_002). The stem plot in this case may appear a bit cluttered, and we can try other plot types to display the distribution of this data (e.g., a box plot, cf. Chapter 7.10), but let us work with this plot type for now. We may sort the data if we are interested in a general picture of the distribution of the values across the dataset (cf. also Chapter 8.1.7). To preserve the original variable while creating a sorted version, duplicate the variable in your workspace after right-clicking on it and give the new one a different name. The simplest way to sort the data now is to right-click on the new variable in the Variables editor and select **Sort Ascending** from the dropdown menu as shown in Figure 146.

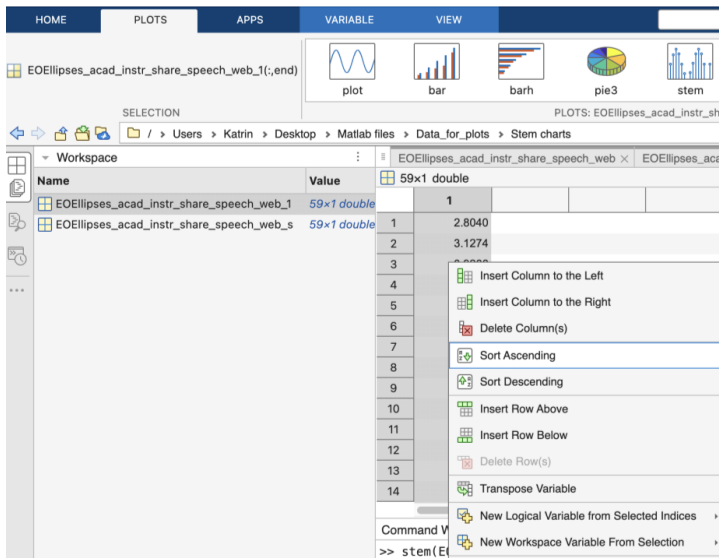


Figure 146: Sorting values of a variable with frequency data for cohesive ellipses

Then plot the data again and it will appear less like a slightly scattered collection of points and more like a curve to give you a clearer representation of the data's overall trend (Figure 147).

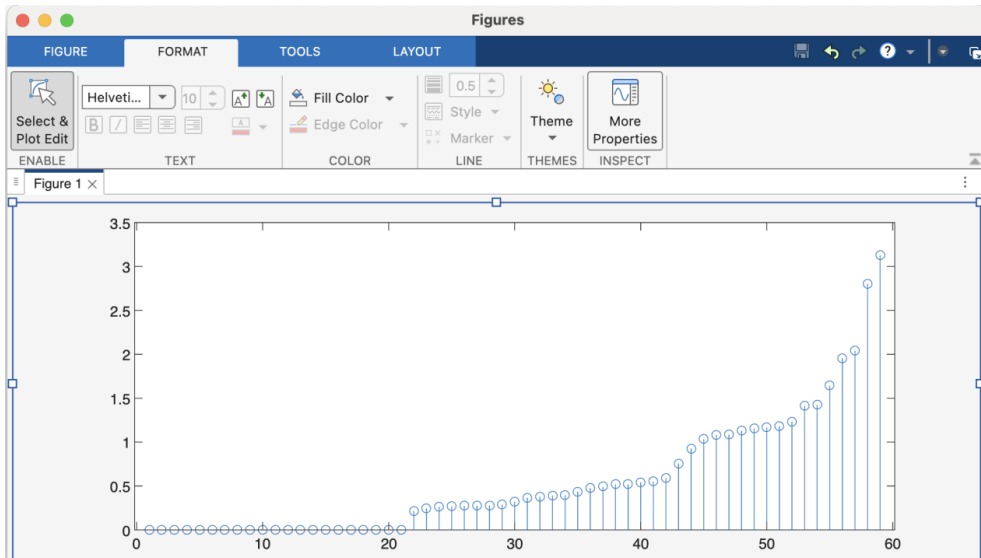


Figure 147: Stem plot with sorted data for ellipses in five registers from GECCo corpus

You can make many adjustments under the **Format** tab, such as selecting the marker style and changing it from a circle to another symbol. You can also modify the line style (or even remove the lines entirely). Click on the respective item in the plot that you want to modify, and then select the desired options from the toolbar (**Marker** or **Style**) to make adjustments (Figure 147). Additionally, the **Property Inspector** allows you to fine-tune these and various other settings.

## 7.9 Scatter plots

Scatter plots are very useful when you are working with corpus data as they provide a visual representation of potential relationships and patterns between two variables. One of the key strengths of scatter plots lies in their ability to highlight possible linear correlations. They can reveal whether two variables are positively correlated – both increasing together – or negatively correlated – where one increases while the other decreases. Additionally, scatter plots can help to determine if there is little to no linear correlation between the variables at all. Another important feature of scatter plots is their capacity to show the distribution of data points across the graph. This makes it easy to identify outliers, i.e., data points that deviate strongly from the general trend, as well as clusters, where groups of points form distinct patterns.

For example, to analyse the relationship between the frequency of a specific word in different documents and document length in words, you can use the `scatter` function to visualise the data. Consider the following dataset:

```
Frequency_of_word_x_per_text = [5, 10, 15, 20, 25];
```

```
textLength = [100, 200, 300, 400, 500];
```

You can create these variables in MATLAB by using these commands. Then select both and click on the scatter icon to create a plot as in Figure 148. In the background, the command `scatter(Frequency_of_word_x_per_text,textLength)` is executed, and you will see it in the Command Window. `TextLength` is plotted on the x-axis and `Frequency_of_word_x_per_text` is plotted on the y-axis.

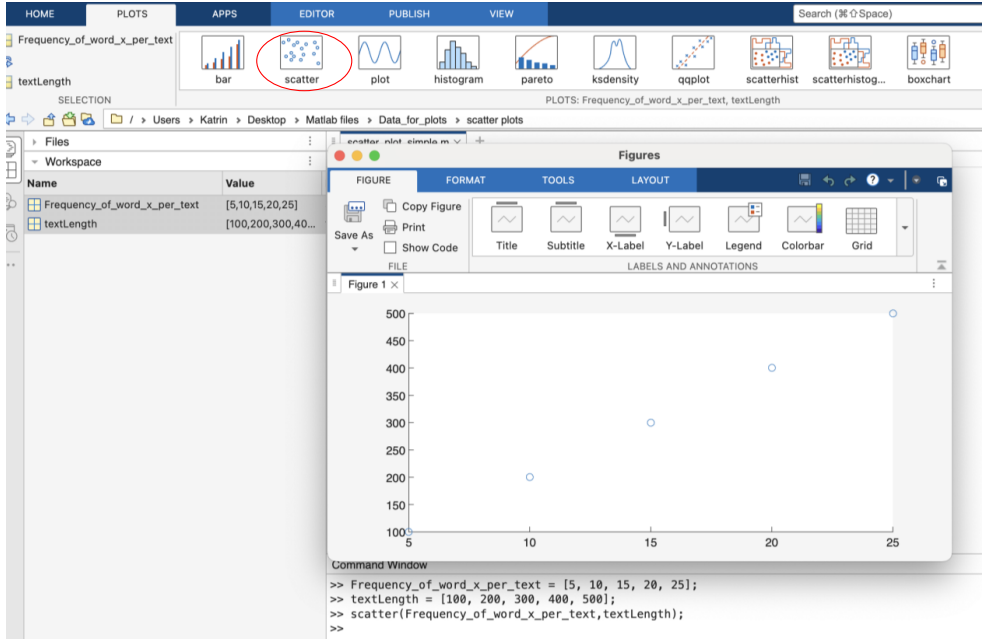


Figure 148: Scatter plot for frequency of a word per text vs. text length

The scatter plot in Figure 148 shows a linear relationship between text length and the frequency of a specific word. As the length of the text increases, the frequency of the word also increases at a constant rate. This appears in the plot as points forming a straight line with a positive slope, indicating that longer texts tend to contain the word more frequently. In this case, the plot starts at 5 on the x-axis and ranges up to 25, while the y-axis spans from 100 to 500, rather than starting at 0. To improve the clarity of the plot, you may want to adjust the axes using the `xlim` and `ylim` functions to start both axes at 0. For better visibility of the data point at (25, 500), you might also consider extending the limits to, for instance, 30 on the x-axis and 600 on the y-axis. In the Property Inspector, which you can open from the **Format** tab, you can adjust the axis limits by editing the values in the **XLim** and **YLim** fields. Additionally, you can add labels to the axes interactively using the options under the **Figure** tab (Figure 149).

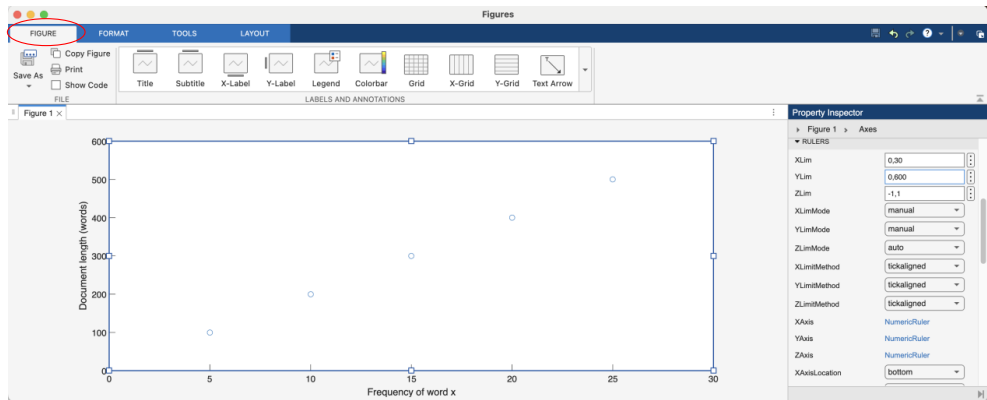


Figure 149: Adjusted scatter plot showing frequency of a word per text vs. text length

A short script to create the same plot as in Figure 149 may look like this:

```
% Sample data
Frequency_of_word_x_per_text = [5, 10, 15, 20, 25];
textLength = [100, 200, 300, 400, 500];

% Create a scatter plot
scatter(Frequency_of_word_x_per_text, textLength);
xlim([0 30]);
ylim([0 600]);
xlabel('Document Length (words)');
ylabel('Frequency of word x');
```

Let us consider another example. Suppose you are analysing the relationship between the frequency of a specific structure in original (non-translated) German texts and their translated English counterparts. A scatter plot can help determine if there is a correlation between them. By plotting the frequency of this structure for each pair of texts in a parallel corpus (German originals and their English translations), a scatter plot can reveal whether texts that frequently use this structure in the German original also have similar frequencies of this structure in their English translated versions. The plot can also indicate whether the relationship is consistent across texts or whether certain texts deviate strongly from the overall trend.

Let us look at the following dataset to illustrate this. We will use the German-English Contrasts in Cohesion corpus (GECCo) that we have already used for a stem plot in Chapter 7.8. This corpus was created as a resource for contrastive studies of textual cohesion and has been annotated for all types of cohesive devices drawing on Halliday and Hasan’s (1976) model of cohesion, as adapted for crosslinguistic annotation. The written part of the corpus is a comparable corpus and at the same time a bi-directional parallel corpus. It contains comparable source texts in English and German (EO and GO) as well as their sentence-aligned German and English translations (GTRANS and ETRANS). Various corpus versions of GECCo are available at Saarland’s University’s CQPweb installation upon request (cf. <https://fedora.clarin-d.uni-saarland.de/gecco/documentation.html>.)

We will concentrate here on the use of nominal ellipses in their function as cohesive devices in a small selection of corpus registers from the parallel corpus part from GECCo. Nominal ellipsis is the omission of the head noun in a noun phrase with the purpose to avoid explicitly mention-

ing or repeating a noun. Often nominal ellipsis involves the deletion of a noun after a descriptive adjective, numeral or adjective expressing indefinite quantities. Sometimes the deletion also extends to parts of the noun phrase that would follow the noun in the full phrasal structure (cf. discussion in Menzel 2017: 110 ff.). The following examples illustrate nominal ellipsis. The brackets represent the omission of a noun which can be inferred from the context as there is an antecedent earlier in the text.

- Nominal ellipsis in English – Example 1 a):  
*While Kim had lots of books, Pat had very few [ ].*  
(example from Huddleston & Pullum 2002: 411)
- Nominal ellipsis in German – Example 1 b):  
*‘Während Kim viele Bücher hatte, besaß Pat nur sehr wenige [ ].’*  
(own translation of Huddleston & Pullum’s example)
- Nominal ellipsis in English – Example 2 a):  
*The Alpha Magnetic Spectrometer is by far the most sensitive cosmic ray detector ever to fly in space, and with a price tag of about \$2 billion, it’s also the most expensive [ ].*  
(example adapted from Davies’ [2010] COHA corpus)
- Nominal ellipsis in German – Example 2 b):  
*Das Alpha-Magnet-Spektrometer ist bei weitem der empfindlichste kosmische Strahlendetektor, der jemals im Weltraum unterwegs war, und mit einem Preis von etwa 2 Milliarden Dollar ist es auch der teuerste [ ].*  
(own translation of the example from COHA corpus)

There are some cross-linguistic differences between English and German noun phrases that result in slightly different possibilities for nominal ellipsis in the two languages. These differences are primarily due to morphological differences in the inflectional and case systems of English and German. For instance, English texts from the GECCo corpus often require or prefer the use of the nominal substitute ‘one’ (cf. Kunz et al. 2021: 102) or repetition of the noun after adjectives, whereas German texts in the corpus more readily permit nominal ellipsis in comparable contexts:

- Nominal ellipsis in German – Example 3 a):  
*Für den praktischen Gebrauch benötigt man größere Zahlen, solche mit mehr als hundert Dezimalstellen. Für das Kochrezept habe ich kleinere [ ] genommen.*  
(German example from the GECCo corpus, corpus text ID: GECCo\_GO\_POPSCI\_008)
- Nominal substitution with ‘one’ in English – Example 3 b):  
*For practical application, large numbers are needed, numbers with more than a hundred digits. For our recipe I have chosen smaller ones.*  
(translation of example 3 a from the GECCo corpus, GECCo\_ETRANS\_POPSCI\_008)

In the GECCo corpus, nominal ellipses linking back cross-clausally to textual antecedents have been annotated as a type of cohesive ellipsis. To investigate whether there is a correlation between the frequency of these cohesive nominal ellipses in German original texts (GO) and in their English translations (ETRANS), we can create a scatter plot to visualise the relationship. We will use the registers of political essays (ESSAY), fictional texts (FICTION) and popular science texts (POPSCI) from the GECCo corpus. Several texts contain no instances of cohesive nominal ellipses at all, as this cohesive device is generally not very frequent in either English or German, and the texts in these corpus registers only range on average between 2,000 and

4,000 tokens in length. However, various German original texts and English translations among these texts do include a few occurrences of nominal ellipses, which is reflected in the frequencies per 1,000 tokens given in Table 7.<sup>23</sup>

<b>Text GECCo GO (corpus text ID)</b>	<b>Cohesive nominal ellipses (per 1,000 tokens)</b>	<b>Text GECCo ETRANS (corpus text ID)</b>	<b>Cohesive nominal ellipses (per 1,000 tokens)</b>
GO ESSAY 001	0.49	ETRANS ESSAY 001	0.41
GO ESSAY 002	1.17	ETRANS ESSAY 002	0.00
GO ESSAY 003	0.00	ETRANS ESSAY 003	0.53
GO ESSAY 004	0.00	ETRANS ESSAY 004	1.34
GO ESSAY 005	0.59	ETRANS ESSAY 005	0.00
GO ESSAY 006	0.00	ETRANS ESSAY 006	0.00
GO ESSAY 007	1.78	ETRANS ESSAY 007	0.00
GO ESSAY 008	0.00	ETRANS ESSAY 008	0.00
GO ESSAY 009	0.00	ETRANS ESSAY 009	0.93
GO ESSAY 010	0.00	ETRANS ESSAY 010	0.00
GO ESSAY 011	0.56	ETRANS ESSAY 011	0.43
GO ESSAY 012	0.00	ETRANS ESSAY 012	0.00
GO ESSAY 013	0.00	ETRANS ESSAY 013	0.00
GO ESSAY 014	0.00	ETRANS ESSAY 014	0.00
GO ESSAY 015	1.43	ETRANS ESSAY 015	1.20
GO ESSAY 016	0.00	ETRANS ESSAY 016	0.00
GO ESSAY 017	0.00	ETRANS ESSAY 017	0.00
GO ESSAY 018	0.56	ETRANS ESSAY 018	0.00
GO ESSAY 019	1.55	ETRANS ESSAY 019	1.33
GO ESSAY 020	2.46	ETRANS ESSAY 020	2.11
GO ESSAY 021	1.62	ETRANS ESSAY 021	0.00
GO ESSAY 022	0.00	ETRANS ESSAY 022	0.00
GO ESSAY 023	0.00	ETRANS ESSAY 023	0.00
GO FICTION 001	0.27	ETRANS FICTION 001	0.73
GO FICTION 002	1.32	ETRANS FICTION 002	0.24
GO FICTION 003	0.00	ETRANS FICTION 003	0.00
GO FICTION 004	0.00	ETRANS FICTION 004	0.25
GO FICTION 005	0.56	ETRANS FICTION 005	0.00
GO FICTION 006	1.25	ETRANS FICTION 006	0.88
GO FICTION 007	0.30	ETRANS FICTION 007	0.28
GO FICTION 008	1.33	ETRANS FICTION 008	0.50
GO FICTION 009	0.00	ETRANS FICTION 009	0.00
GO FICTION 010	1.87	ETRANS FICTION 010	1.76
GO POPSCI 001	0.28	ETRANS POPSCI 001	0.00
GO POPSCI 002	0.00	ETRANS POPSCI 002	0.24
GO POPSCI 003	0.80	ETRANS POPSCI 003	0.00
GO POPSCI 004	0.00	ETRANS POPSCI 004	0.00
GO POPSCI 005	0.82	ETRANS POPSCI 005	0.00
GO POPSCI 006	0.55	ETRANS POPSCI 006	0.00
GO POPSCI 007	0.00	ETRANS POPSCI 007	0.00
GO POPSCI 008	0.28	ETRANS POPSCI 008	0.00
GO POPSCI 009	0.00	ETRANS POPSCI 009	0.00
GO POPSCI 010	0.00	ETRANS POPSCI 010	0.00

Table 7: Frequency of cohesive nominal ellipses in three registers from corpus parts with German originals and English translations from GECCo corpus

<sup>23</sup> For the complete frequency data of all ellipsis types per text and register across all corpus sections in the GECCo corpus, you may also refer to Menzel 2017: 452ff.

To create a scatter plot for these data in MATLAB, we will first organise the data into vectors, e.g., copy and paste the values from Table 7 into two column vectors as two separate variables and name them, for instance, `GO_nom_ell_ESSAY_FICTION_POPSCI` and `ETRANS_nom_ell_ESSAY_FICTION_POPSCI`. The vectors to be used for a scatter plot need to be of equal length. Then we select both variables from the Workspace panel and click on the **scatter** icon in the Plot Gallery. Alternatively, we can combine both columns into a single numeric matrix variable and name it, for instance, `GO_ETRANS_nom_ell_ESSAY_FICTION_POPSCI`. Then you only need to select this variable and click on the **scatter** icon. The data from the first column will be plotted on the x-axis and the data from the second column will be plotted on the y-axis. You will see a plot as in Figure 150 to which you can add axis labels and add further modifications if desired, e.g., change the marker from the default symbol 'o' to another symbol. Marker colour and line width can also be adjusted.

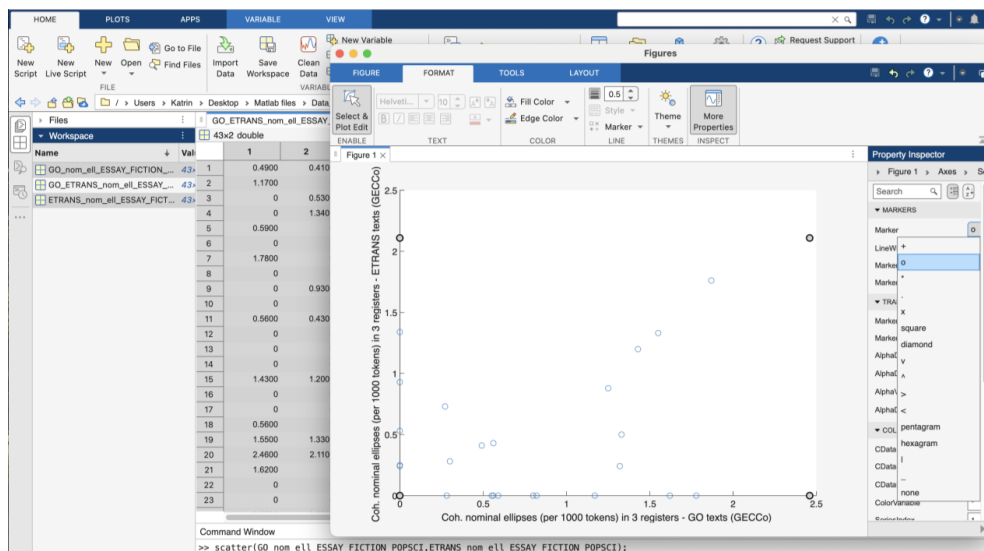


Figure 150: Scatter plot for frequencies of cohesive nominal ellipses in pairs of original and translated texts from three registers from GO/ETRANS in GECCo corpus

You can also use `scatter` function in a small series of commands and add some labels programmatically. If you have two variables (`GO_nom_ell_ESSAY_FICTION_POPSCI` and `ETRANS_nom_ell_ESSAY_FICTION_POPSCI`), use:

```
scatter(GO_nom_ell_ESSAY_FICTION_POPSCI,
ETRANS_nom_ell_ESSAY_FICTION_POPSCI);
```

If you have one variable with two columns for the data (e.g., with the name `GO_ETRANS_nom_ell_ESSAY_FICTION_POPSCI`) use:

```
scatter(GO_ETRANS_nom_ell_ESSAY_FICTION_POPSCI(:,1),
GO_ETRANS_nom_ell_ESSAY_FICTION_POPSCI(:,2));
```

Add labels for the axes and a title of the plot via command, for instance, like this:

```
xlabel('Coh. nominal ellipses (per 1000 tokens) in 3 registers - GO texts (GECCo)');
```

```
ylabel('Coh. nominal ellipses (per 1000 tokens) in 3 registers - ETRANS texts  
(GECCo)');  
  
title('Scatter plot of ellipses in GECCo German original texts vs. English  
translated texts');
```

Let us now examine another example involving the creation of a scatter plot. In this case, we will explore the frequencies of initialisms and their corresponding full forms in the English section of the EuroParl-UdS corpus. This corpus contains edited and published records of statements made in English during debates in the European Parliament (Karakanta et al., 2018). A description of the corpus, along with a link to download the data, is available here: <https://fedora.clarin-d.uni-saarland.de/europarl-uds/>. In earlier chapters, we have seen other EuroParl datasets, such as those available on the OPUS platform and SketchEngine, for different examples. The EuroParl-UdS dataset, however, is a specific version of the EuroParl data enriched with extensive metadata about both texts and speakers. Notably, the English portion of EuroParl-UdS includes only texts produced by native English speakers, whereas in the original EuroParl, the English section consists of both original and translated texts, as well as English texts produced by non-native speakers, which makes it difficult to distinguish between them.

In our example, we will now focus on the frequencies of a selection of initialisms, i.e., abbreviations formed from the initial letters of multi-word expressions. These include two subtypes: letter-by-letter initialisms, in which each letter is pronounced individually (e.g. “WTO”, standing for “World Trade Organisation”), and acronymic initialisms, which are pronounced as words (e.g. “NATO”, standing for “North Atlantic Treaty Organisation”). A scatter plot can help reveal whether there is a linear correlation between the frequencies of initialisms – across both subtypes – and their corresponding full forms within this text type and dataset. A positive correlation – where the initialisms and their full forms occur with similar frequencies – may suggest that initialisms generally function as synonyms of their long forms in these texts. This could indicate that both short and long forms tend to be used interchangeably in the texts, especially when no other synonyms are available for specialised multi-word terms or named entities other than their shortened versions. On the other hand, if the initialisms are highly conventionalised – i.e., fully familiar to both speakers and audience – the full expressions may appear only rarely or not at all in the parliamentary debates. Instead, the shorter, more manageable forms will dominate (see also the discussion and related analysis in Menzel 2024). Table 8 presents the normalised frequencies (per million tokens) of some of the most frequent initialisms in the dataset and their corresponding full forms. The frequency information was obtained using CQP queries in Saarland University’s CQPweb interface (<https://corpora.clarin-d.uni-saarland.de/cqpweb/>). The queries targeted sequences of capital letters (with an optional lowercase “s” at the end to indicate plural forms) in the English corpus section “EuroParl-UdS (en>de, en).” This is a subset of the EuroParl-UdS data with English original texts that also have a corresponding German translation in the corpus.

Initialisms <sup>24</sup>	Fpm	Full form <sup>25</sup>	Fpm <sup>26</sup>
WTO	140	World Trade Organi(s/z)ation	15
MEP(s)	153	Member(s) of the European Parliament	27
ACP	100	African, Caribbean and Pacific [Group of States]	3
IGC	85	Intergovernmental Conference,	18
NATO	75	North Atlantic Treaty Organi(s/z)ation	0
CAP	68	Common agricultural policy	58
NGO(s)	82	Non-Governmental Organi(s/z)ation(s)	8
SME(s)	72	Small and medium-sized enterprise(s)	24
CFP	51	Common Fisheries Policy	6
USA	45	United States of America	22
GDP	41	Gross domestic product	3
EPLP	36	European Parliamentary Labour Party	17
GMO(s)	41	Genetically modified organism(s)	3
VAT	29	Valued Added Tax	1
ECB	26	European Central Bank	12
ILO	18	International Labour Organi(s/z)ation	1
CFSP	16	Common foreign and security policy	23
WHO	15	World Health Organi(s/z)ation	16
ICES	12	International Council for the Exploration of the Sea	1
GSP	12	Generalised Scheme/System of Preferences	1
ECJ	11	European Court of Justice	29
TAC(s)	13	Total allowable catch(es)	3
EPA(s)	17	Economic Partnership Agreement	8
LDC(s)	15	Least Developed Countries	7
CSR	10	Corporate Social Responsibility	11
ETS	10	Emission(s) Trading Scheme(s)	13
QMV	9	Qualified majority voting/vote(s)	20
ENP	9	European neighbourhood policy	5
IMF	9	International Monetary Fund	1
EIB	9	European Investment Bank	6

Table 8: Frequencies of initialisms and corresponding full forms in English section of EuroParl-UdS

To create a scatter plot of these data in MATLAB, you have several options for defining the variables. For example, you can start by opening a new variable and either copy and paste or manually enter the frequency values as two columns (name your variable, for instance, `Initialisms_vs_full_forms`). Alternatively, you can first save the data in an Excel file and then import it into MATLAB. You can create a scatter plot in MATLAB either from a single numeric matrix containing both columns or by using two separate variables for the x and y data. Then select the data and simply click on the scatter icon or use a command such as:

```
scatter(Initialisms_vs_full_forms(:,1),Initialisms_vs_full_forms(:,2));
```

Both methods will produce a scatter plot as depicted in Figure 151. Additionally, a title and axis labels have been added interactively to this plot using the options in the toolstrip under the **Figure** tab highlighted in Figure 151.

<sup>24</sup> An “s” in brackets indicates that the initialism occurs in singular and plural forms, and the counted frequencies include both forms, e.g., “MEP” and “MEPs”.

<sup>25</sup> All variations of the terms, regardless of whether they were in uppercase or lowercase, were included in the count; for example, “Common Agricultural Policy” and “Common agricultural policy” were both considered. Additionally, the “(s/z)” notation used in some terms in this table shows that the respective word is spelled either with “s” or “z” in the data, and the counted frequencies include both forms, e.g., “World Trade Organisation” and “World Trade Organization”.

<sup>26</sup> Frequencies below 1 per million were rounded up to 1, except when the value was 0.

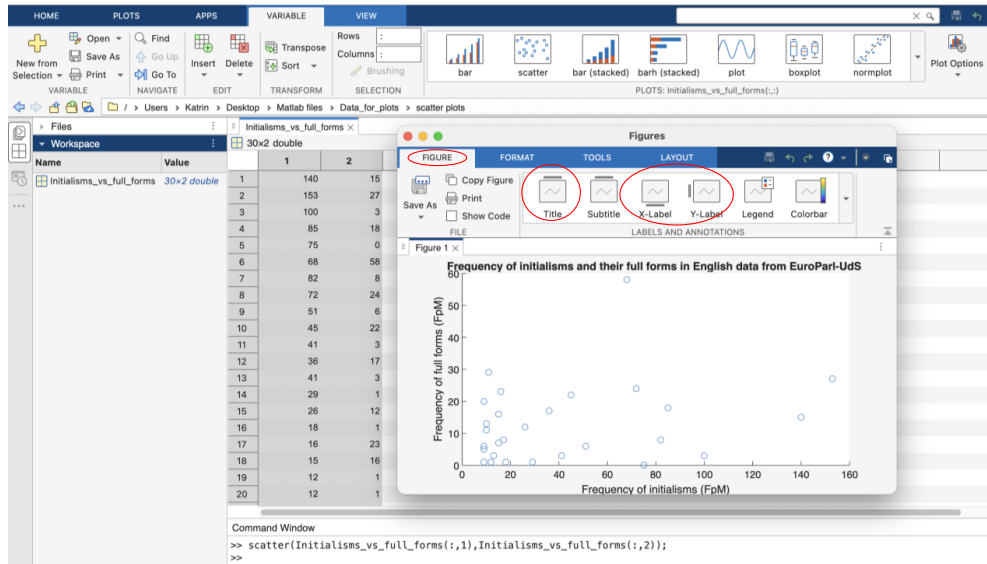


Figure 151: Scatter plot for frequencies of initialisms and their full forms in English data from EuroParl-UdS

You can make further adaptations to the plot using the interactive options such as setting both the x-axis and y-axis to the same range, for example, from 0 to 160. To do this, open the **Property Inspector** from the **Format** tab and edit the values in the **XLim** and **YLim** fields directly (Figure 152). Additionally, you may want to explore the **Basic Fitting** option for plots, available under the **Tools** tab in MATLAB, which allows you to interactively fit a model to your data. This tool offers several fitting options, including linear and polynomial fits, that can be applied directly to the plot. When you click on the Basic Fitting icon from the Tools tab, a user interface will appear, and you can select the type of fit you wish to apply. After choosing a fit (such as linear), MATLAB will display the fitted curve alongside your scatter points so that you may visually assess how well the model represents the relationship between the variables (Figure 152). For more advanced fitting and regression analysis, see MATLAB’s Curve Fitting Toolbox documentation and the Statistics and Machine Learning Toolbox documentation on the MathWorks website:

*Curve Fitting Toolbox documentation:*

<https://www.mathworks.com/products/curvefitting.html> and <https://www.mathworks.com/help/curvefit/>

*Statistics and Machine Learning Toolbox documentation:*

<https://www.mathworks.com/products/statistics.html> and <https://www.mathworks.com/help/stats/>

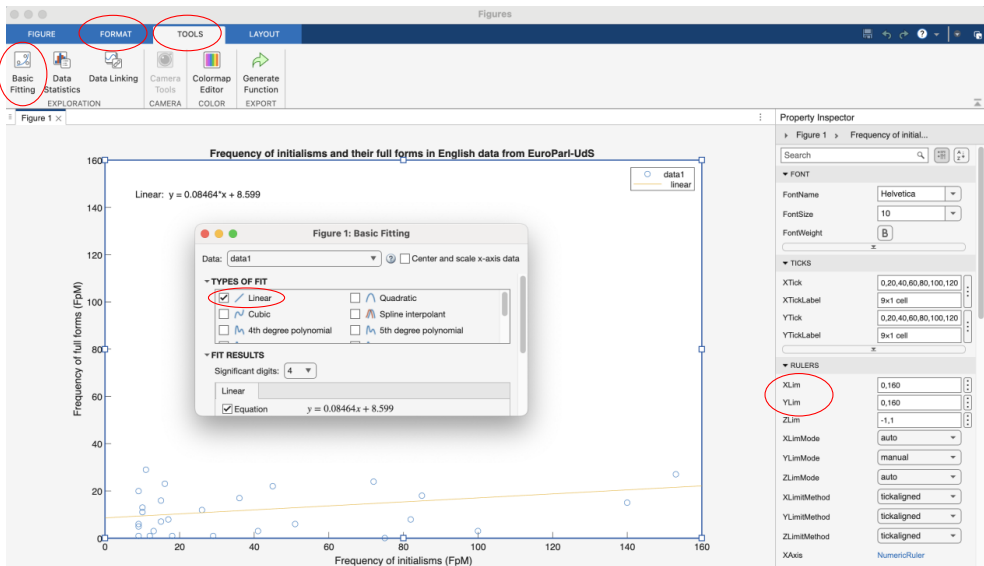


Figure 152: Adjusting axis limits via the Property Inspector and applying a model fit using the Basic Fitting tool

In the scatter plot of our data, it appears that many of the analysed initialisms have higher frequency values than their corresponding full forms. However, the plot does not show a strong or consistent trend, and the fitted line may not perfectly represent the data. The low slope of the line suggests that while there is some relationship between the variables, it is weak. If any trend exists, it is subtle, and the frequency of the initialisms seems only weakly correlated with the frequency of the corresponding full forms.

To quantify the relationship between variables like these, you can calculate Pearson's correlation coefficient (cf. Chapter 9.6). This coefficient provides a numerical value that indicates the strength and direction of the linear relationship between the two variables.

## 7.10 Box plots

A box plot (or box chart) is a graphical representation that summarises key statistics of a dataset. It is particularly useful to visualise the distribution and spread of corpus data. To create box plots in MATLAB, you can use either the `boxplot` function after having installed the Statistics and Machine Learning Toolbox or the `boxchart` function introduced in more recent versions of MATLAB.<sup>27</sup> Both plot types can be accessed via the options from Plots Gallery (Figure 153).

<sup>27</sup> cf. <https://www.mathworks.com/help/stats/boxplot.html> and <https://www.mathworks.com/help/matlab/ref/boxchart.html>

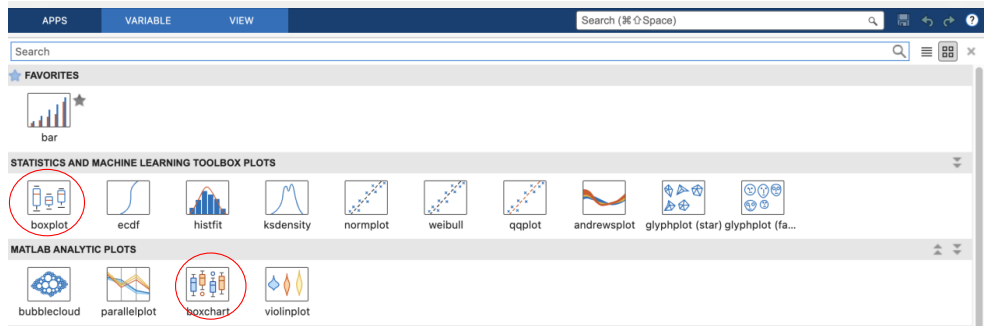


Figure 153: 'Boxplot' and 'boxchart' icons in Plots Gallery

We will particularly look at examples with the `boxplot` function from the Statistics and Machine Learning Toolbox, but the two functions are rather similar.

Let us take a look at the length of the texts in the GECCo corpus, a corpus which we have already used for various other types of plots in previous chapters. The English corpus part of the original (non-translated) texts (EO\_GECCo) consists of 142 texts. The text length values measured in tokens for each of these texts can be extracted from the corpus. These values are also listed in Menzel (2017: 452ff.). For your convenience, they are also provided here. The following MATLAB command can be used to create a variable for them:

```
tokens_EO_GECCo = [3923; 3837; 4317; 3935; 3853; 4093; 3855; 4248; 4239;
4259; 2278; 653; 725; 1132; 1790; 1058; 1557; 1737; 882; 815; 1020; 1010;
849; 840; 2461; 1092; 1138; 1168; 1809; 1856; 1065; 1122; 1012; 900; 1029;
885; 845; 1132; 1138; 3683; 3548; 3955; 3743; 3613; 3540; 3676; 3862; 3756;
3620; 4905; 5353; 3828; 4126; 4102; 3620; 5637; 4073; 3780; 3914; 3593; 3116;
3583; 3615; 3528; 4043; 3501; 3706; 3780; 3702; 2091; 3736; 3717; 4558; 3615;
3650; 2252; 3928; 3193; 1974; 3169; 2015; 3330; 3043; 3577; 3956; 3400;
3535; 1483; 3332; 3296; 2882; 3314; 1843; 2571; 2754; 4409; 4621; 2573; 3215;
1842; 2300; 2726; 3550; 2564; 856; 2650; 2102; 1723; 3478; 2521; 3122; 2229;
2694; 2595; 2770; 1848; 2024; 2659; 2647; 3494; 3586; 2224; 2215; 3634; 3591;
3497; 3442; 3601; 3653; 2970; 3258; 3691; 4206; 1696; 2896; 1621; 3639; 2930;
3618; 2546; 1933; 4085]
```

When we have this variable in our Workspace panel, the command `boxplot(tokens_EO_GECCo)` or a click on the **boxplot** icon will create a boxplot of the data in `tokens_EO_GECCo`. Since this variable is a one-column vector, the function will plot a single box representing the distribution of the data. Figure 154 shows the box plot (with an added y-grid, y-axis label and an x-axis tick indicating the name of the corpus section).

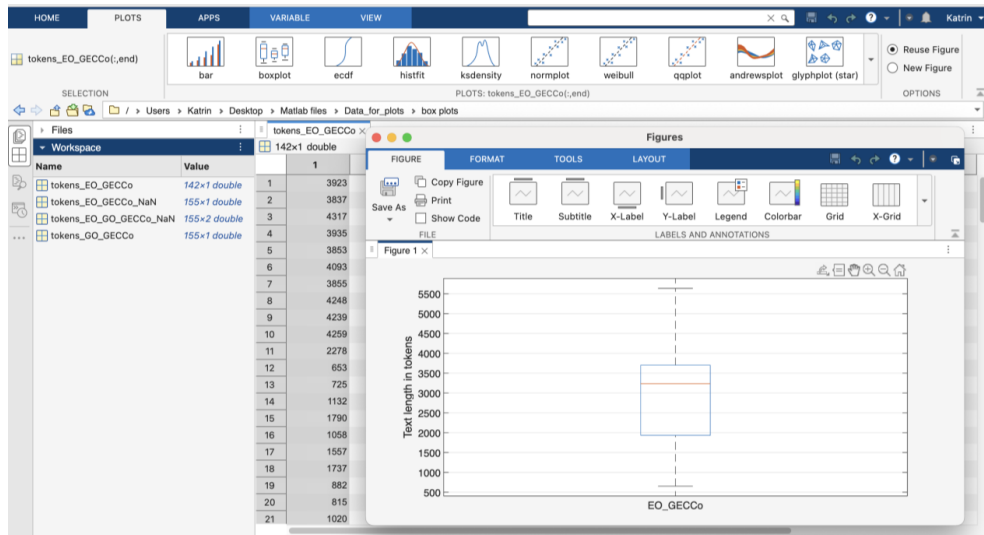


Figure 154: Box plot showing distribution of text length in English section EO from GECCo corpus

The default plot type in MATLAB is a vertical box plot, where the boxes are oriented along the y-axis, and the categories (if there were more than one) are plotted along the x-axis. If you want to change the orientation of the plot, you may add 'orientation', 'horizontal' to the `boxplot` function as in the following command:

```
boxplot(tokens_EO_GECCo, 'orientation', 'horizontal');
```

The box will then be oriented along the x-axis.

Let us now also create a variable to store the text lengths of the texts in the German original, non-translated part of the GECCo corpus (GO\_GECCo), which consists of 155 texts. The data has already been extracted (cf. Menzel 2017: 452ff.) and prepared for this variable. It can be assigned to a variable `tokens_GO_GECCo` with the following command:

```
tokens_GO_GECCo = [3784; 3785; 3547; 10131; 3271; 3964; 4066; 3815; 3790;
3550; 2035; 858; 1554; 1249; 1682; 1653; 2244; 971; 1871; 2562; 1787; 478;
1986; 1792; 1394; 646; 1460; 1782; 645; 1220; 1851; 1795; 2153; 3734; 3793;
3675; 3621; 3572; 3989; 3312; 3753; 3582; 3747; 4532; 3879; 5415; 3876; 3570;
4269; 3798; 4267; 3910; 4120; 1545; 2282; 3449; 1345; 1433; 2179; 3414; 3487;
3529; 2563; 3466; 1954; 2597; 3637; 3313; 4028; 4965; 1789; 3307; 3112; 2754;
3745; 2994; 2687; 1231; 1803; 1417; 3053; 3569; 3729; 3731; 3567; 3666; 3619;
3595; 3628; 3538; 3535; 1914; 2554; 4059; 4288; 4157; 4988; 2620; 2762; 3105;
2421; 2367; 874; 1982; 2355; 1890; 3158; 2747; 2817; 1882; 1267; 2973; 718;
3117; 2809; 2535; 1046; 1655; 640; 934; 818; 1722; 1631; 3457; 3506; 1151;
1258; 577; 2662; 1883; 1754; 1823; 658; 879; 561; 552; 516; 1806; 2831; 1779;
2314; 2436; 2640; 1252; 1277; 2834; 4079; 3401; 1185; 3959; 3652; 2918; 3354;
2656; 2572];
```

To create a boxplot for both boxes, we could use the following command:

```
boxplot([tokens_EO_GECCo, tokens_GO_GECCo]);
```

However, there is a mismatch in the lengths of the two variables that we wish to plot as boxes. `tokens_EO_GECCo` contains values for 142 texts, while `tokens_GO_GECCo` contains values for 155 texts. This length discrepancy will result in an error in MATLAB because the data used for boxplots must have the same number of elements. To resolve this, NaN values need to be added to the shorter variable (`tokens_EO_GECCo`) until both vectors have the same length, which in this case would be 155. NaN stands for ‘Not a Number’ and is used to represent missing or undefined data. In this case, the NaN values just act as placeholders to fill in the gap so that both variables have the same number of elements.<sup>28</sup> To reflect this modification, we can update the variable and rename it as `tokens_EO_GECCo_NaN`. Now the command `boxplot ([tokens_EO_GECCo_NaN, tokens_GO_GECCo])` plots two boxes – one for the English and one for the German data from the GECCo corpus (Figure 155).

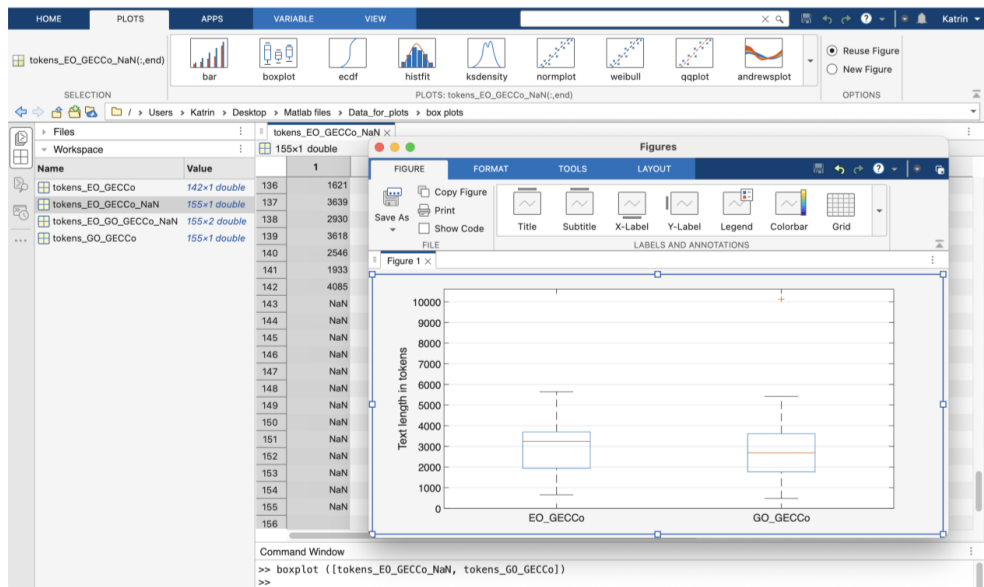


Figure 155: Box plot for text length of English and German sections from GECCo corpus

In each box, the central line indicates the median, which represents the middle value of the dataset. The bottom and top edges of the box correspond to the lower (25th percentile) and upper (75th percentile) quartiles, respectively. The range between these two quartiles is known as the interquartile range (IQR), which captures the middle 50% of the data. The whiskers extend from the edges of the box to the minimum and maximum values that are not considered outliers. Outliers in MATLAB typically are data points that fall more than 1.5 times the IQR above the upper quartile or below the lower quartile, and they are marked individually using the '+' sign. In our case, we see that the German `GO_GECCo` dataset has one notable outlier – a text containing more than 10,000 tokens. Despite this extreme value, the interquartile range and the whisker lengths appear quite similar between `EO_GECCo` and `GO_GECCo`, indicating comparable variability in typical text lengths. However, the median text length is clearly higher in the German `GO_GECCo` than in the English `EO_GECCo`.

<sup>28</sup> However, keep in mind that it can affect the interpretability and comparability of the results when there are many NaN values in box plots.

We have created the box plot based on a variable for the tokens in EO\_GECCo and one for the tokens in GO\_GECCo. Alternatively, you may put the values for the English and German data into only one variable as two columns and name the variable, for instance, `tokens_EO_GO_GECCo`. Keep in mind that here as well both columns need to be of equal length. Use NaN values to fill in any missing entries to ensure that the shorter column is not automatically filled with zeros by MATLAB. Then you can select this variable and click on the boxplot icon or write the command `boxplot(tokens_EO_GO_GECCo)`; to obtain a figure with two boxes similar to the one shown in Figure 155 above. Since `tokens_EO_GO_GECCo` is a numeric matrix with two columns, the boxplot function plots one box for each column in `tokens_EO_GO_GECCo` (Figure 156).

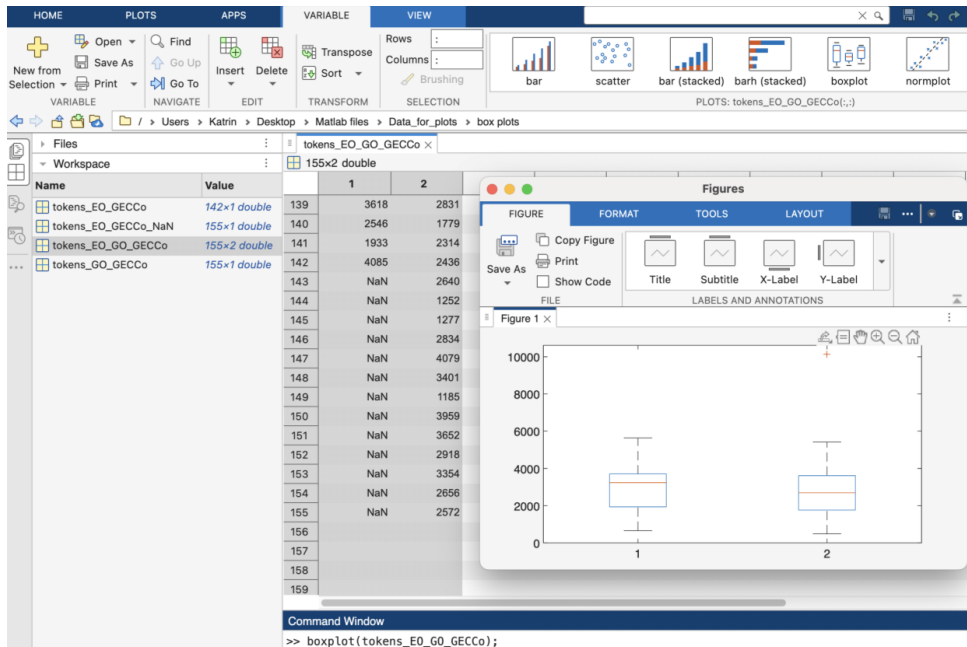


Figure 156: Creating box plot from one variable with two columns for English and German data from GECCo corpus

Notches can be added to the box plot to help compare the medians of the datasets and to determine whether the median values are significantly different from each other. If the notches of two box plots do not overlap, it suggests that the medians are significantly different at a 5% significance level. To add notches, use the `boxplot` or `boxchart` function in a command and add `'Notch', 'on'` in your command:

```
boxplot(tokens_EO_GO_GECCo, 'Notch', 'on');
```

This command produces Figure 157. Additionally, some labels have been added for clarity via the formatting options available from the figure's toolbar or the Property Inspector.

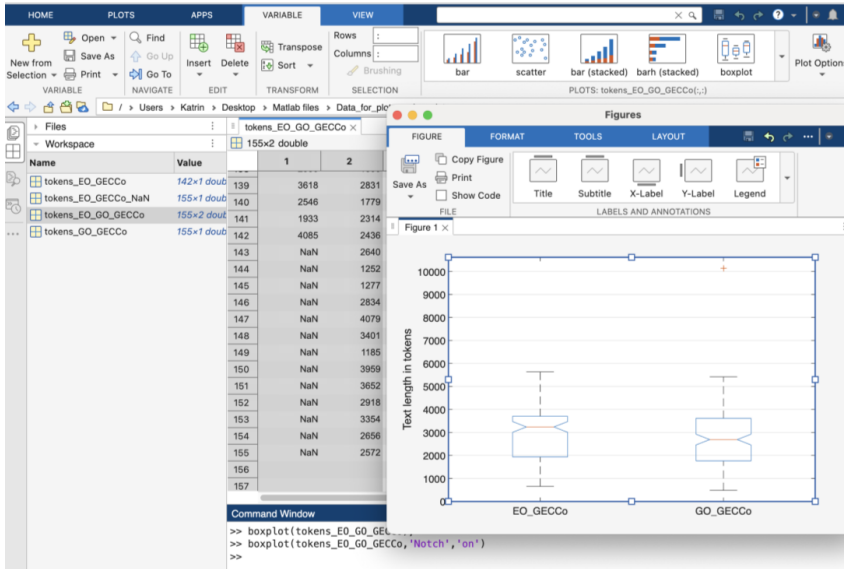


Figure 157: Notched box plot created by using the boxplot function

Using the command `boxchart(tokens_EO_GO_GECc0, 'Notch', 'on');` produces a similar box plot; however, the notches here appear as tapered, shaded regions around the median, and outliers are displayed using the 'o' symbol (Figure 158).

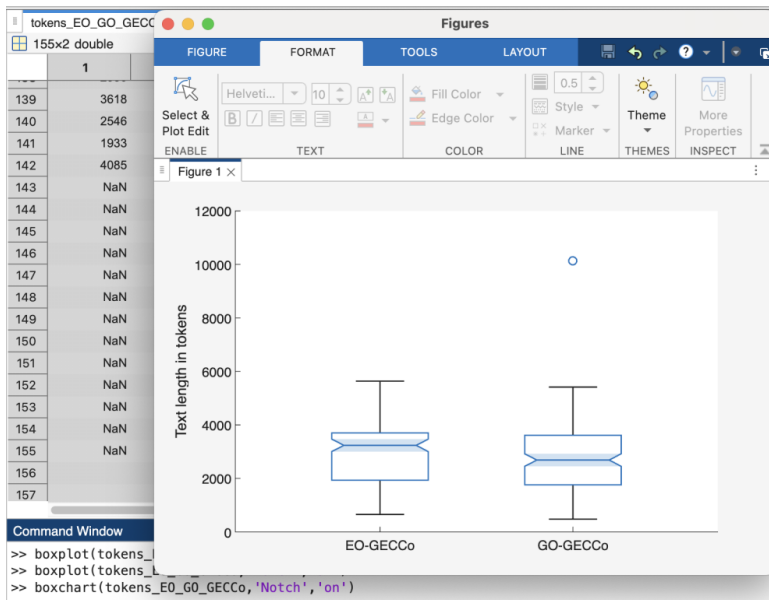


Figure 158: Notched box plot created by using the boxchart function

Let us examine another example of box plots, this time to compare English and German corpus data from the GECCo corpus with a focus on the distribution of a specific linguistic phenome-

non. In Chapter 7.8, we created a stem chart created to visualise the frequency of cohesive ellipses per text across five English registers from the GECCo corpus. In this section, we will present the same data as a box plot and compare them to corresponding data from the German component of the GECCo corpus (cf. also <https://fedora.clarin-d.uni-saarland.de/gecco/docu-mentation.html>). The corpus registers selected for this analysis include academic lectures (ACADEMIC), instruction manuals (INSTR), letters to shareholders (SHARE), prepared speeches (SPEECH) and corporate websites (WEB) from the German and English original (non-translated) corpus sections GO\_GECCo and EO\_GECCo. The normalised frequency data of cohesive ellipses per text, already extracted and provided below in Table 9, can either be directly entered into a MATLAB numeric variable or first saved in an Excel table and then imported into MATLAB. For the complete frequency data of ellipses per text and register across all texts from all corpus sections in the GECCo corpus, you may also refer to Menzel 2017: 452ff.

Text ID in GO_GECCo	Cohesive ellipses (per 1,000 tokens)	Text ID in EO_GECCo	Cohesive ellipses (per 1,000 tokens)
GO_ACADEMIC_001	1.06	EO_ACADEMIC_001	2.8
GO_ACADEMIC_002	1.06	EO_ACADEMIC_002	3.13
GO_ACADEMIC_003	2.54	EO_ACADEMIC_003	0.93
GO_ACADEMIC_004	0.39	EO_ACADEMIC_004	0
GO_ACADEMIC_005	1.22	EO_ACADEMIC_005	0.52
GO_ACADEMIC_006	5.8	EO_ACADEMIC_006	1.95
GO_ACADEMIC_007	3.44	EO_ACADEMIC_007	0.52
GO_ACADEMIC_008	1.05	EO_ACADEMIC_008	1.41
GO_ACADEMIC_009	0	EO_ACADEMIC_009	1.18
GO_ACADEMIC_010	0.56	EO_ACADEMIC_010	1.64
GO_INSTR_001	0	EO_INSTR_001	0
GO_INSTR_002	0	EO_INSTR_002	0.32
GO_INSTR_003	0	EO_INSTR_003	0.28
GO_INSTR_004	0	EO_INSTR_004	0.28
GO_INSTR_005	0	EO_INSTR_005	0
GO_INSTR_006	0	EO_INSTR_006	0
GO_INSTR_007	0.59	EO_INSTR_007	0
GO_INSTR_008	0.29	EO_INSTR_008	0.27
GO_INSTR_009	2.27	EO_INSTR_009	0.26
GO_INSTR_010	0	EO_INSTR_010	0
GO_INSTR_011	0	EO_SHARE_001	1.09
GO_INSTR_012	0	EO_SHARE_002	0
GO_INSTR_013	0	EO_SHARE_003	0
GO_INSTR_014	0	EO_SHARE_004	2.04
GO_SHARE_001	0	EO_SHARE_005	0.22
GO_SHARE_002	0	EO_SHARE_006	0
GO_SHARE_003	0.49	EO_SHARE_007	0
GO_SHARE_004	0	EO_SHARE_008	0.54
GO_SHARE_005	0	EO_SHARE_009	0.43
GO_SHARE_006	0	EO_SHARE_010	0.37
GO_SHARE_007	0	EO_SHARE_011	0
GO_SHARE_008	0	EO_SHARE_012	0
GO_SHARE_009	0	EO_SHARE_013	1.17
GO_SHARE_010	0	EO_SPEECH_001	0.75
GO_SHARE_011	0	EO_SPEECH_002	1.43
GO_SPEECH_001	0	EO_SPEECH_003	0
GO_SPEECH_002	0	EO_SPEECH_004	0.29
GO_SPEECH_003	0	EO_SPEECH_005	0.4
GO_SPEECH_004	1.06	EO_SPEECH_006	0
GO_SPEECH_005	0.95	EO_SPEECH_007	0
GO_SPEECH_006	0	EO_SPEECH_008	0
GO_SPEECH_007	0.35	EO_SPEECH_009	1.16

Text ID in GO_GECCo	Cohesive ellipses (per 1,000 tokens)	Text ID in EO_GECCo	Cohesive ellipses (per 1,000 tokens)
GO_SPEECH_008	0.53	EO_SPEECH_010	1.08
GO_SPEECH_009	1.58	EO_SPEECH_011	0
GO_SPEECH_010	0.34	EO_SPEECH_012	0.49
GO_SPEECH_011	0	EO_SPEECH_013	1.13
GO_SPEECH_012	0.64	EO_SPEECH_014	0.38
GO_SPEECH_013	0.71	EO_WEB_001	0
GO_SPEECH_014	0	EO_WEB_002	0
GO_SPEECH_015	0	EO_WEB_003	0.48
GO_SPEECH_016	0	EO_WEB_004	0.59
GO_SPEECH_017	0	EO_WEB_005	1.04
GO_SPEECH_018	0	EO_WEB_006	1.23
GO_WEB_001	0.38	EO_WEB_007	0.27
GO_WEB_002	0	EO_WEB_008	0
GO_WEB_003	0	EO_WEB_009	0.55
GO_WEB_004	1.06	EO_WEB_010	0.39
GO_WEB_005	0	EO_WEB_011	0
GO_WEB_006	0	EO_WEB_012	0.24
GO_WEB_007	0		
GO_WEB_008	0.25		
GO_WEB_009	0		
GO_WEB_010	0.34		
GO_WEB_011	0.3		
GO_WEB_012	0		
GO_WEB_013	0		

Table 9: Cohesive ellipsis per text in five registers from English and German part of GECCo corpus

When you put these data into an Excel file and import it from there into a numeric variable as two columns, NaN values can be added automatically for the shorter column with the values for the texts from the selected five registers from the English corpus section EO\_GECCo. Alternatively, define the values for the German and English data via commands as individual column vectors (including NaN values in the English data) and then combine them into one a numeric matrix in an additional step, e.g., by using the following commands:

```
GO_column1 =
[1.06;1.06;2.54;0.39;1.22;5.80;3.44;1.05;0;0.56;0;0;0;0;0.59;0.29;2.27;
0;0;0;0;0;0;0.49;0;0;0;0;0;0;0;0;0;0;1.06;0.95;0;0.35;0.53;1.58;0.34;0;
0.64;0.71;0;0;0;0;0.38;0;0;1.06;0;0;0;0.25;0.34;0.30;0;0];

EO_column2 =
[2.80;3.13;0.93;0;0.52;1.95;0.52;1.41;1.18;1.64;0;0.32;0.28;0.28;0;0;0.27
;0.26;0;1.09;0;0;2.04;0.22;0;0;0.54;0.43;0.37;0;0;1.17;0.75;1.43;0;0.29;0.4
0;0;0;0;1.16;1.08;0;0.49;1.13;0.38;0;0;0.48;0.59;1.04;1.23;0.27;0;0.55;0.39
;0;0.24;NaN;NaN;NaN;NaN;NaN;NaN;NaN];

GO_EOEllipses5Registers = [GO_column1, EO_column2];
```

Select the variable and click on the **boxplot** or **boxchart** icon from the Plot Gallery or type a command with one of these functions and your variable name, e.g., `boxplot(GO_EOEllipses5Registers)` or `boxchart(GO_EOEllipses5Registers)` to generate a plot as in Figure 159 (with labels added using the interactive formatting options).

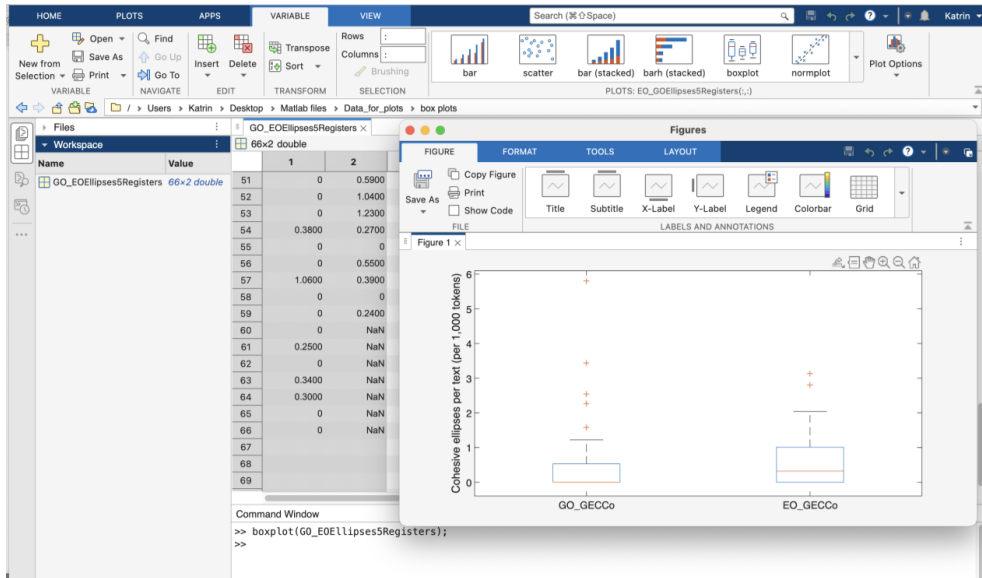


Figure 159: Box plot for cohesive ellipses in five registers from German and English sections in GECCo corpus

The boxes reveal distinct differences between the German GO\_GECCo and the English EO\_GECCo datasets. For the GO\_GECCo data, the median being at zero reflects that the majority of values are clustered at the lower end of the scale. In contrast, the median for the EO\_GECCo data lies between zero and one, suggesting that while many values are still near zero, there are more non-zero values compared to the GO\_GECCo dataset. This indicates a greater presence of cohesive ellipses in the EO\_GECCo data overall. The smaller interquartile range (IQR) for the GO\_GECCo data points to a more concentrated distribution, with values tightly clustered around zero. In contrast, the larger box for the EO\_GECCo data, with a wider IQR, reflects greater variability and a broader spread of values. This suggests that the EO\_GECCo dataset exhibits greater diversity in the occurrence of cohesive ellipses, with many texts showing moderate values, rather than being limited to zeros, while a few texts display higher frequencies. Additionally, the upper whisker for the EO\_GECCo data is higher, indicating that occurrences beyond the IQR are generally larger compared to the GO\_GECCo data. However, the GO\_GECCo data contains more extreme outliers beyond the upper whisker, with values such as 5.8 that are much higher than the rest of the data. This suggests that while the GO\_GECCo dataset is predominantly rich in texts with very low values for cohesive ellipses, it also includes exceptional texts with very high frequencies of cohesive ellipses.

We have now examined two distinct examples to demonstrate how to generate box plots in MATLAB – one illustrating the distribution of text length across selected sections of the English and German parts of the GECCo corpus from Saarland University’s CQPweb interface and another showing the distribution of ellipses a cohesive device across subsets of the English and German sections of the same corpus.

## 7.11 Histograms

A histogram is a graphical representation of the frequency distribution of data. It groups continuous data into “bins” or intervals and shows how many data points fall within each bin. That means it helps you understand how your data is distributed across different ranges of values. The height of each bar in the histogram corresponds to the frequency of data points within that particular bin. Histograms are particularly useful for identifying the shape of the data distribution, such as whether it is skewed, normally distributed or if there are any noticeable gaps. Skewness refers to the asymmetry of the distribution; a distribution is considered skewed if it has a longer tail on one side than the other. For example, a right-skewed distribution has a longer tail on the right as most data points are concentrated on the left. In contrast, a normally distributed dataset follows a symmetric bell curve, where the data is evenly distributed around the centre (cf. also Chapter 8.1.5).

In this section, we will use some data that we have already explored in Chapter 7.10 with a different plot type. There, we summarised and compared the distribution of texts with different lengths from a set of 142 original English, non-translated texts (EO\_GECCo) from the GECCo corpus and 155 original German, non-translated texts (GO\_GECCo) from the same corpus by using a box plot. Here, we will use the text length data of the German corpus section again, but we will create a histogram to summarise them. Both histograms and box plots are effective tools for visualising data distributions. However, while box plots offer a concise summary of the overall distribution and its variability, histograms provide a more detailed and granular view of the frequency of data points within specific value ranges. For your convenience, the text length data for the texts from GO\_GECCo are provided here again in a MATLAB command to define a variable.

```
tokens_GO_GECCo = [3784; 3785; 3547; 10131; 3271; 3964; 4066; 3815; 3790;
3550; 2035; 858; 1554; 1249; 1682; 1653; 2244; 971; 1871; 2562; 1787; 478;
1986; 1792; 1394; 646; 1460; 1782; 645; 1220; 1851; 1795; 2153; 3734; 3793;
3675; 3621; 3572; 3989; 3312; 3753; 3582; 3747; 4532; 3879; 5415; 3876; 3570;
4269; 3798; 4267; 3910; 4120; 1545; 2282; 3449; 1345; 1433; 2179; 3414; 3487;
3529; 2563; 3466; 1954; 2597; 3637; 3313; 4028; 4965; 1789; 3307; 3112; 2754;
3745; 2994; 2687; 1231; 1803; 1417; 3053; 3569; 3729; 3731; 3567; 3666; 3619;
3595; 3628; 3538; 3535; 1914; 2554; 4059; 4288; 4157; 4988; 2620; 2762; 3105;
2421; 2367; 874; 1982; 2355; 1890; 3158; 2747; 2817; 1882; 1267; 2973; 718;
3117; 2809; 2535; 1046; 1655; 640; 934; 818; 1722; 1631; 3457; 3506; 1151;
1258; 577; 2662; 1883; 1754; 1823; 658; 879; 561; 552; 516; 1806; 2831; 1779;
2314; 2436; 2640; 1252; 1277; 2834; 4079; 3401; 1185; 3959; 3652; 2918; 3354;
2656; 2572];
```

To create a histogram for these data in MATLAB, you can select the variable in your workspace and then click on the **histogram** icon under the **Plots** tab. In the background, the command `histogram(tokens_GO_GECCo)` will be executed and you will obtain a histogram as in Figure 160.

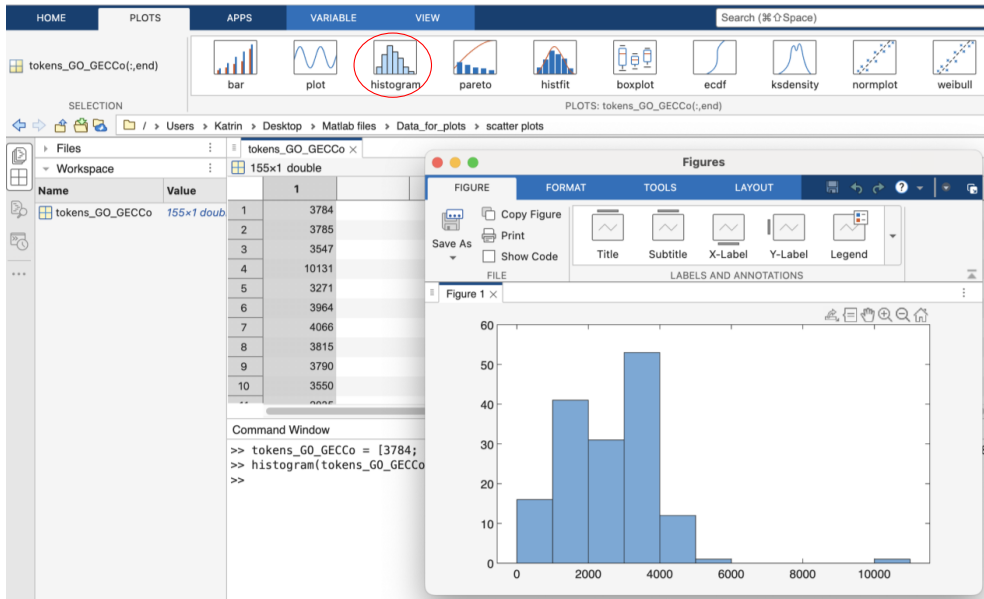


Figure 160: Histogram showing text length distribution in German section from GECCo corpus

Note that the default number of bins and axis limits in a histogram generated by clicking the histogram icon may not always be optimal. If these settings do not effectively represent your data, it is recommended to adjust them for improved visualisation. For example, when selecting an appropriate number of bins, it is important to strike a balance in bin size. The plot needs to be helpful to reveal meaningful patterns in the data without being too granular or too broad. In the figure, various layout features can be adjusted easily by using the options from the toolstrips in the **Figure** and **Format** tabs.

Figure 161 is a customised version of the histogram shown in Figure 160. A title and axis labels have been added. Furthermore, the x-axis limits have been manually set from 0 to 11,500. This has been achieved by clicking on the axis to open the axis properties in the Property Inspector and editing the values for **XLim** under the section **RULERS**. The style of the lines between the bins has also been modified. By clicking on the bins, the **BINS** section in the Property Inspector was opened, where you can customise the number of bins and the bin width. In the example shown in Figure 161, the bin width has been changed manually from MATLAB's default value (as set at 2000 for this specific case) to 500.

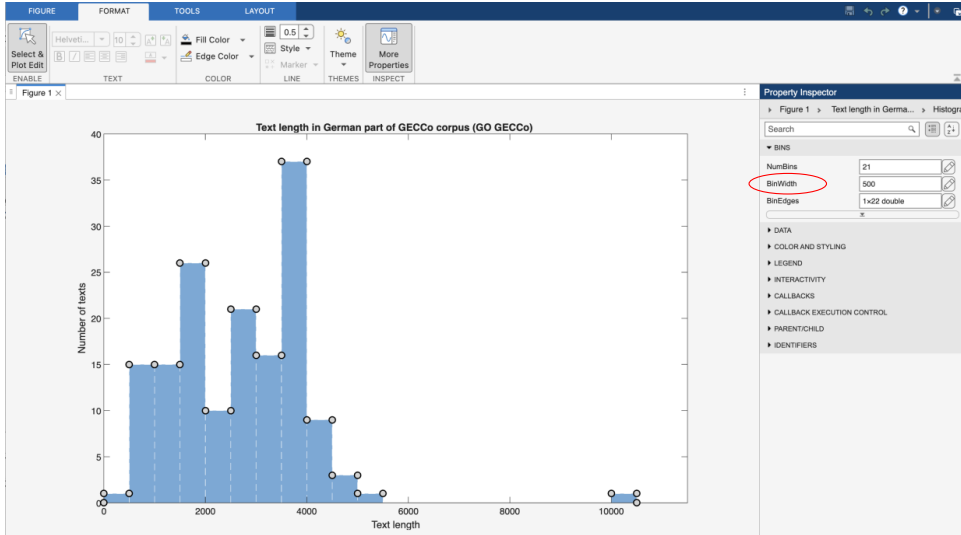


Figure 161: Histogram for text length distribution in German section from GECCo corpus with customised layout features

Additionally, we can adjust the x-tick labels. For example, if you would like the x-ticks to display steps of 500 instead of 2000, navigate to the Property Inspector. Under the **TICKS** section, click on **XTick** and a field will appear displaying the current x-ticks and their corresponding labels. In this field, you can define the step size, such as setting it to 500 (Figure 162). When you make this change, MATLAB will in this case automatically rotate the x-axis tick labels by 30 degrees to enhance readability. If you would like to change the label rotation to a different angle, such as 45 degrees, simply edit the **XTickLabelRotation** value under the **TICKS** section.



Figure 162: Adjusting x-tick steps and x-tick label rotation

For all the interactive adjustments you can make in the figure, MATLAB also provides commands that you can use programmatically. These commands are detailed in the MATLAB documentation on the MathWorks website, and they can typically also be accessed through a conversation with MATLAB Copilot.

For example, to rotate the x-tick labels to a 45-degree angle with a command, use this one if you are creating the figure programmatically:

```
xtickangle(45);
```

If you wish to set the x-ticks to show steps of 500 within a specific range such as 0 to 11500, you can use this command:

```
xticks(0:500:11500);
```

You can also manually define the tick positions by entering each value explicitly, like this:

```
xticks([0 500 1000 etc.]);
```

The corresponding x-tick labels will normally be shown automatically, but you can also use the `xticklabels` command to assign custom labels, such as:

```
xticklabels({'x = 0', 'x = 500', 'x = 1000' etc.});
```

If you need to adjust the number of bins or their width in a histogram, you can use commands like the following:

```
histogram(tokens_GO_GECCo, 'NumBins', 21);
```

This command sets the number of bins to 21. Alternatively, to set the bin width here to 500, you can use:

```
histogram(tokens_GO_GECCo, 'BinWidth', 500;
```

In MATLAB, there is also the `histfit` function which creates a histogram of a dataset and overlays a chosen probability distribution onto it. By default, it fits a normal distribution, but you can also specify other types of distributions ([www.mathworks.com/help/stats/histfit.html](http://www.mathworks.com/help/stats/histfit.html)).

## 7.12 Heat maps

Heatmaps are a graphical representation of data where individual values are represented with colours. They are particularly useful for visualising complex data sets in which you want to identify patterns quickly. In MATLAB, heatmaps can be created from matrices or tables to provide a clear visual summary of the data. Before you can create a heatmap, you first need to organise your data into a matrix or a table. Each cell corresponds to a value that will be represented by a colour in the heatmap. Then you can use the `heatmap` function for your data. You can specify the x-axis and y-axis labels, as well as the colour variable.

Let us consider an example of hypothetical English corpus data from the 18th century, organised into 20-year time periods. This dataset includes various English translations from source languages such as French, German, Latin, Italian and Spanish. Our goal is to visualise the trans-

lation counts for each source language over time using a heatmap. First, we create a cell array `sourceLanguages` containing the names of the source languages and a cell array `timePeriods` listing the time periods:

```
sourceLanguages = {'French', 'German', 'Latin', 'Italian', 'Spanish'};
timePeriods = {'1700-1719', '1720-1739', '1740-1759', '1760-1779', '1780-1799'};
```

Next, we create a matrix `textsCount` that contains the counts of translations for each combination of time period and source language.

```
textsCount = [19, 4, 3, 6, 2;
              44, 3, 12, 1, 0;
              4, 3, 12, 1, 1;
              8, 23, 1, 0, 0;
              12, 15, 0, 0, 0];
```

Then the `heatmap` function is called with the source languages and time periods as axes and the `textsCount` matrix as the data. The first argument, `sourceLanguages`, represents the individual labels of the languages for the x-axis, and the second argument, `timePeriods`, represents the labels for the individual time periods on the y-axis. The third argument, `textsCount`, is the matrix of data values that will be visualised in the heatmap:

```
h = heatmap(sourceLanguages, timePeriods, textsCount);
```

When you create the heatmap, setting its title and labelling the axes with general descriptions of what they represent can be done later using the interactive options available in the figure window. If you prefer to set the title and labels programmatically via commands, you can use the `title`, `xlabel` and `ylabel` functions:

```
title('Heatmap of translations by source language and time period');
xlabel('Source languages');
ylabel('Time periods');
```

Alternatively, we can set the title and axis labels programmatically using properties of the heatmap object `h`:

```
h.Title = 'Heatmap of translations by source language and time period';
h.XLabel = 'Source languages';
h.YLabel = 'Time periods';
```

Here is the complete code in the form of a script:

```
% Define the data
sourceLanguages = {'French', 'German', 'Latin', 'Italian', 'Spanish'};
timePeriods = {'1700-1719', '1720-1739', '1740-1759', '1760-1779', '1780-1799'};
```

```

textsCount = [19, 4, 3, 6, 2;
              44, 3, 12, 1, 0;
              4, 3, 12, 1, 1;
              8, 23, 1, 0, 0;
              12, 15, 0, 0, 0];

% Create the heatmap

h = heatmap(sourceLanguages, timePeriods, textsCount);

% Set the title and label the axes

title('Heatmap of translations by source language and time period');

xlabel('Source languages');
ylabel('Time periods');

% Alternatively use: h.Title = 'Heatmap of translations by source language
and time period'; h.XLabel = 'Source languages'; h.YLabel = 'Time periods';

```

The script will generate a heatmap, as shown in Figure 163.

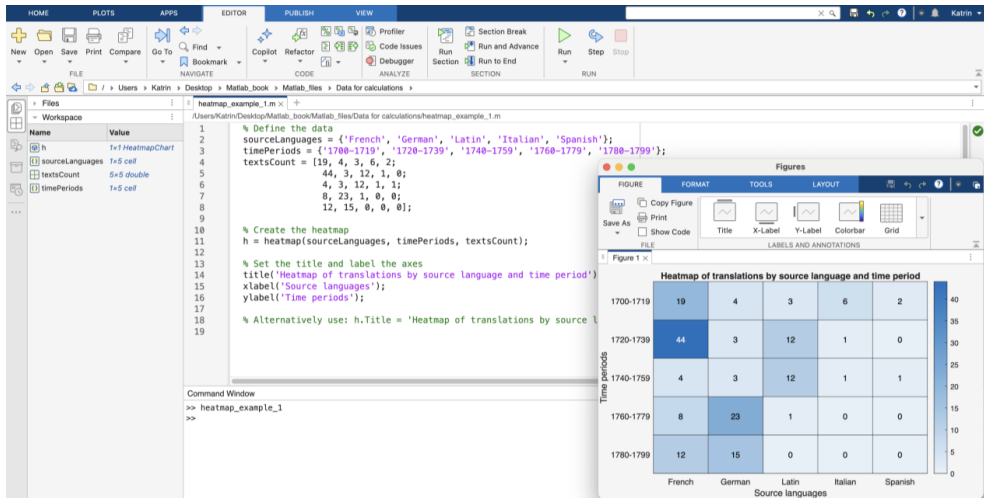


Figure 163: Heatmap visualisation of translation counts for each source language over time

We can customise the figure further via the interactive options in the figure window and the Property Inspector. Title, x-label and y-label, for instance, can be edited here with the options from the Labels & Annotations section of the Figure tab or via the Property Inspector or we can add them if we did not use any commands for them. Additionally, the desired colour scheme can be selected in the Property Inspector (cf. Figure 164).

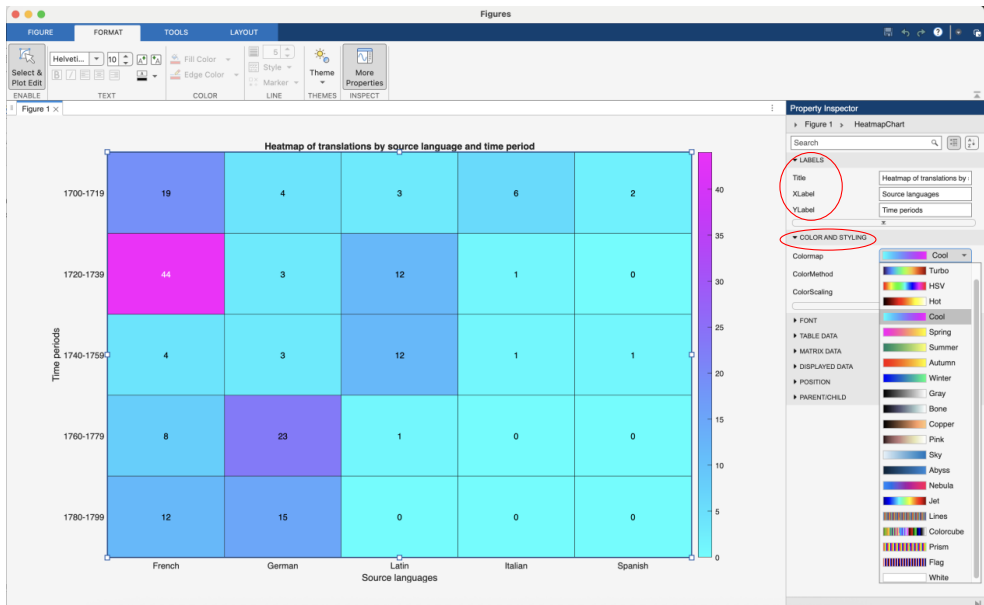


Figure 164: Customising a heatmap via the Property Inspector

It is also possible to use the `heatmap` function for variables from your workspace directly from the interactive plotting options. However, the organisation of your variables in the workspace is important for the heatmap to function correctly. The variables must be arranged and listed in the order of the x-axis, y-axis and the data values. Since variables are generally listed alphabetically in the Workspace panel, the heatmap plot icon may not be visible if the order required for heatmaps is not maintained. To resolve this issue, you can rename your variables to ensure they appear alphabetically in the desired order for the x-axis, y-axis and the data. For example, if you have variables in your workspace such as `sourceLanguages` (for the x-axis), `textsCount` (for the data) and `timePeriods` (for the y-axis), renaming `textsCount` to something like `x_textCounts` will help it appear last in the list. This adjustment will make the heatmap icon available in the toolbar when you select your respective variables in the workspace (Figure 165).

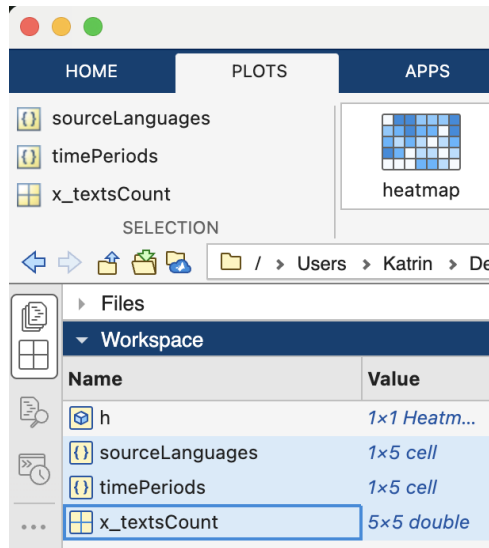


Figure 165: Creating a heatmap via the heatmap icon on the toolbar

Please also refer to the MathWorks documentation on heatmaps at the following links:

*Create heatmap chart*

<https://www.mathworks.com/help/matlab/ref/heatmap.html>

*Create heatmap from tabular data*

[https://www.mathworks.com/help/matlab/creating\\_plots/create-heatmap-from-tabular-data.html](https://www.mathworks.com/help/matlab/creating_plots/create-heatmap-from-tabular-data.html)

*Heatmap chart appearance and behaviour*

<https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.heatmapchart-properties.html>

You can also, for instance, visualise correlations between variables using a heatmap in MATLAB (cf. Chapter 9.6 and the MathWorks documentation at [https://www.mathworks.com/help/matlab/data\\_analysis/visualize-corrcoef.html](https://www.mathworks.com/help/matlab/data_analysis/visualize-corrcoef.html)), and there are many other possible applications for heatmaps.

## 7.13 Word clouds

Word clouds are an excellent way to visualise textual data – individual words or phrases – by representing their frequency in a visually engaging format. You may want to use word clouds, for instance, to visualise extracted frequencies from specific corpus queries. The size of each word or phrase in the cloud is proportional to its occurrence in the dataset which makes it easy to identify key terms at a glance. The basic `wordcloud` function is available in MATLAB without requiring any additional toolboxes. However, it is highly recommended to use the `wordcloud` function in conjunction with MATLAB's Text Analytics Toolbox (cf. Chapter 2.1.5 and Chapter 10) as it offers enhanced capabilities for creating word clouds. The examples presented in this book were carried out using MATLAB with the Text Analytics Toolbox installed.

An example of a word cloud created with MATLAB was already presented in Chapter 4.2 (Figure 44). In this example in Chapter 4.2, we extracted data for the absolute frequencies of the top collocates of “health care” in the COCA corpus from the English-Corpora.org website. In the query, we specified a search for all words that occur immediately to the left or right of the term “health care.” Let us look at this example here again and focus on the word cloud plot options. Below is a sample dataset based on our query, which includes the absolute frequencies of all immediate collocates of “health care” that appear at least 25 times in the COCA corpus (Table 10).

Word	Absolute freq.	Word	Absolute freq.
UNIVERSAL	1034	RETIREE	75
MENTAL	654	NATIONALIZED	69
AFFORDABLE	459	VA	61
PRIMARY	260	INADEQUATE	60
BASIC	201	IMPROVING	56
PROVIDING	165	PAYER	55
COMPREHENSIVE	152	BEHAVIORAL	47
REPRODUCTIVE	146	REFORMING	46
RISING	143	SOCIALIZED	45
PREVENTIVE	118	RATION	36
AFFORD	116	HIGH-QUALITY	30
ADEQUATE	111	LICENSED	28
SINGLE-PAYER	101	SUBSIDIZED	25
GOVERNMENT-RUN	100		

Table 10: Immediate collocates of “health care” in COCA

In Chapters 4.2 and 5.5, we also discussed how to import such data into MATLAB from an Excel spreadsheet. During the data import process, we need to specify the data we are interested in and import it into a variable – in this case, as a table. The first column of this MATLAB table should contain the words, while the second column should hold the frequency data. Next, to create a word cloud, we may open the table in the Variables editor, select our data there and then click on the **wordcloud** icon in the Plots section. The icon will probably be already visible in the toolstrip as one of the recommended and available plot types for the dataset, but you can find it also by expanding the full list of plots, where it appears under the “MATLAB Analytic Plots” category. Figure 166 shows the word cloud that is created for this dataset of immediate collocates of “health care” that occur at least 25 times in the COCA corpus.

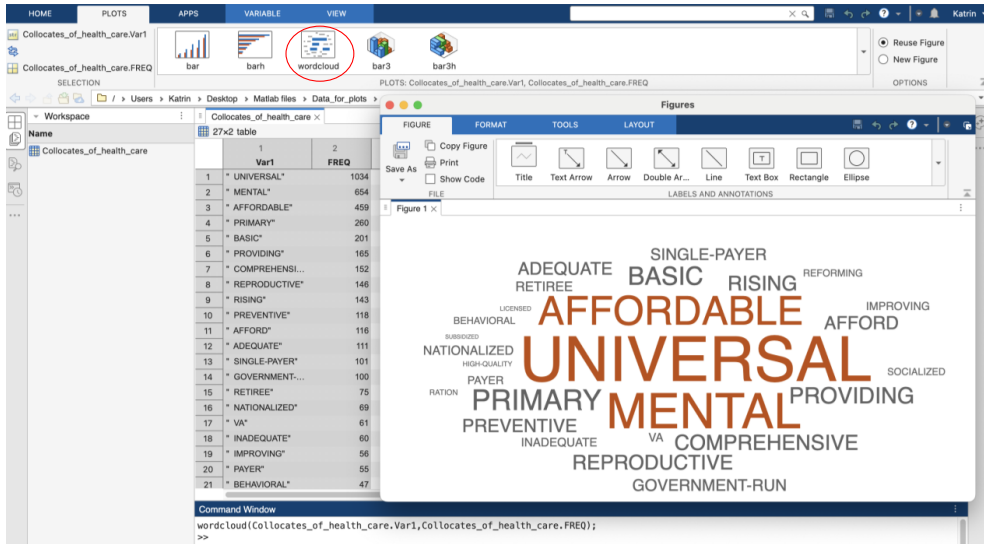


Figure 166: Word cloud for collocates of “health care” (freq. > 24) based on COCA data

As you can see in Figure 166, clicking on the word cloud icon executes a command like the one shown below, which runs in the background and is displayed in the Command Window:

```
wordcloud(Collocates_of_health_care.Var1,Collocates_of_health_care.FREQ);
```

In this example, `Collocates_of_health_care` is the name of the table that contains the data. During data import, the table columns were automatically named `Var1` and `FREQ` by default. If desired, these names can be changed by opening the table in the Variables editor and by clicking on the column headers to rename them, such as to more descriptive titles like `Words` and `Counts`. The command `wordcloud(Collocates_of_health_care.Var1, Collocates_of_health_care.FREQ);` generates a word cloud visualisation in MATLAB using data from this table. In this command, the first argument, `Collocates_of_health_care.Var1`, refers to the column in the table that contains the words to be displayed in the word cloud. The second argument, `Collocates_of_health_care.FREQ`, refers to the column that contains the corresponding frequencies or weights for each word. We have created this word cloud with a simple click on the respective icon, but you may also create your word cloud using the `wordcloud` function in the Command Window or in a script with a command as the one above. However, using the Plots section in the toolbar provides a quicker and user-friendly way to generate it, especially for initial exploration and visualisation.

You can customise various aspects of the word cloud using the options available in the **Figure** and **Format** tabs, as well as in the **Property Inspector**. For example, you can add a title by clicking the appropriate icon, insert visual elements such as arrows, change font styles and colours. By default, the most frequent words are automatically highlighted in a different colour. If you prefer all words to appear in the same colour, you can change this setting in the Property Inspector. Set the field next to **HighlightColor** to match the main colour used for the rest of the words (you will find its RGB values listed above under the **Color** category). Figure 167 shows a customised example of the same word cloud with these modifications applied, including an added arrow, different font styles and colours and an added title.

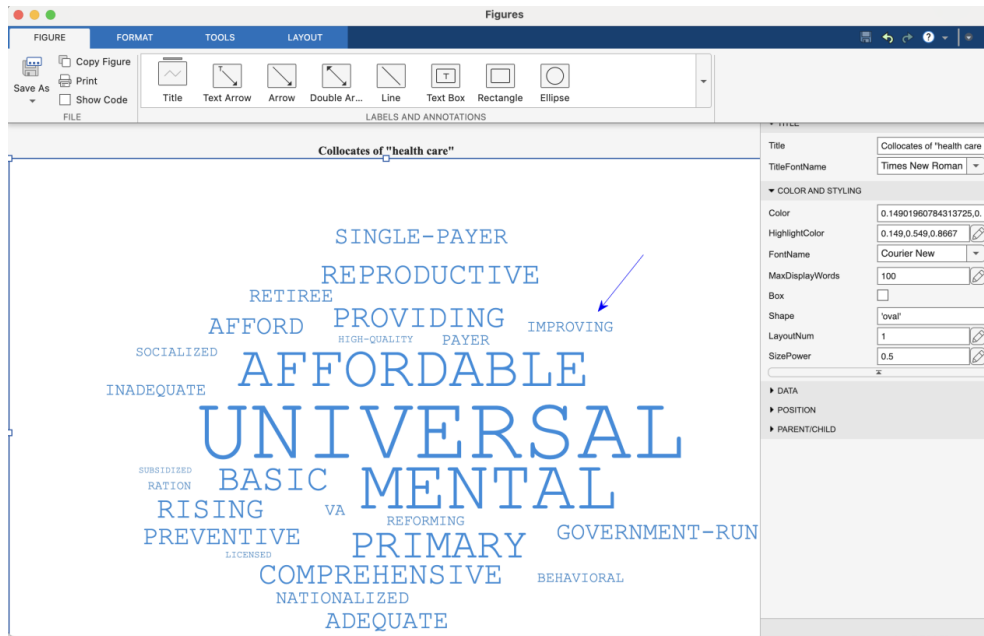


Figure 167: Customised word cloud for collocates of “health care” in COCA

To illustrate how word clouds can be used in MATLAB to visualise multi-word lexical patterns, the following example focuses on adjective + noun combinations extracted from the Royal Society Corpus (RSC, cf. Chapter 4.4). This corpus contains scientific articles published between the 1660s and the 1990s in journals such as *Philosophical Transactions* and is searchable via Saarland University’s CQPweb interface. We have previously looked at some RSC data to explore women’s authorship in early microbiology research articles to create a pie chart (Chapter 7.6). Building on that, we now turn to linguistic patterns in early microbiology texts in the RSC by all author types from the 1860s to 1920. For that purpose, we take the corpus version “Royal Society Corpus V6.0.4 Open Web with Topics”, which contains texts up to the year 1920. The corpus is enriched with metadata detailing the text topics or disciplines, derived through topic modelling.

To obtain a frequency list of adjective + noun patterns, we can run a CQP query for adjectives followed by nouns. Let us specify here that the adjective preceding the noun must have at least 10 characters in order to highlight mainly specialised term patterns. This approach is similar to the more detailed analysis of adjective + noun patterns in the RSC microbiology texts in Menzel (2025). Our CQP query also restricts the text topic to the field of microbiology, i.e., research articles that are currently annotated with a primary or secondary topic label of “immunology” in their metadata. We limit the query to texts published between 1860 and 1920; this time span can be defined in the CQPweb interface (by selecting the date range) and need not be specified in the CQP query. The CQP query in the CQPweb interface might look like this:

```
[pos="JJ" & word="[A-Za-z]{10,}"] [pos="NN.*"]::match.text_primaryTopic = "Immunology" | match.text_secondaryTopic = "Immunology"
```

You will then see the results in the concordance view in CQPweb and you can select the Frequency Breakdown option from where you can download the frequencies as a text file (cf.

Chapter 4.4). When you have downloaded a text file with the query results from CQPweb, you can import the relevant data from there into MATLAB as a table (Figure 168). For instance, you may choose to include only patterns that occur with a minimum frequency of 50 to avoid overcrowding the visualisations. When importing the data into a MATLAB variable (in this case, a table), also verify whether each entry in your dataset is indeed an adjective + noun pattern.

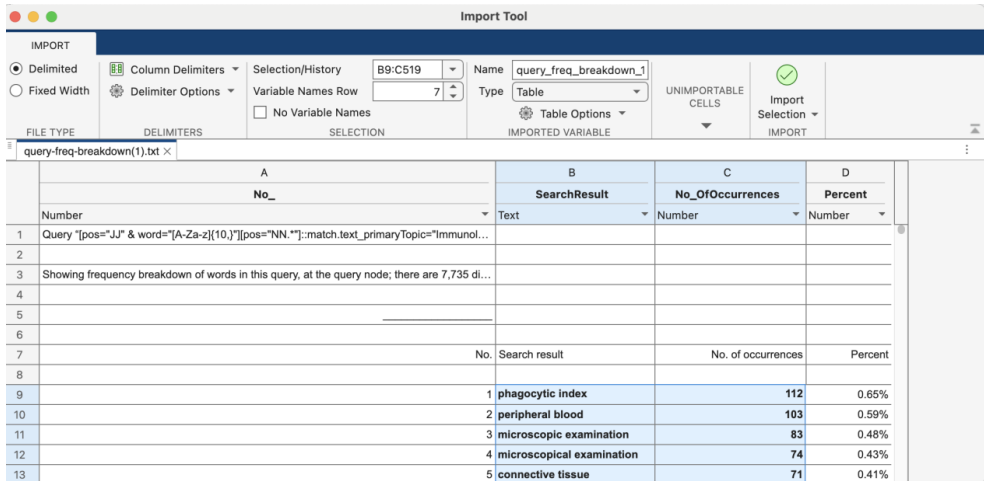


Figure 168: Importing data for word cloud from text file obtained through CQPweb interface

Table 11 also lists all adjective–noun patterns from the query results that occur at least 50 times. If you just want to try out this word cloud example without extracting the data from the RSC via CQPweb yourself, you can simply copy the contents of Table 11 into an Excel or text file and then import the data from that file as a table into a MATLAB variable.

Adj. + noun	Absolute freq.
phagocytic index	112
peripheral blood	103
microscopic examination	83
microscopical examination	74
connective tissue	71
germicidal action	68
susceptible animals	68
spontaneous generation	66
bactericidal action	65
subcutaneous tissue	62
intercellular spaces	59
consecutive days	56
tubercular infection	56
epithelial cells	55
alimentary canal	54

Table 11: Adjective + noun patterns in RSC microbiology texts (adj. with at least 10 characters and patterns that occur at least 50 times)

Plotting the data as a word cloud then is rather easy by selecting the data in the Variables editor and clicking on the **wordcloud** icon (Figure 169).

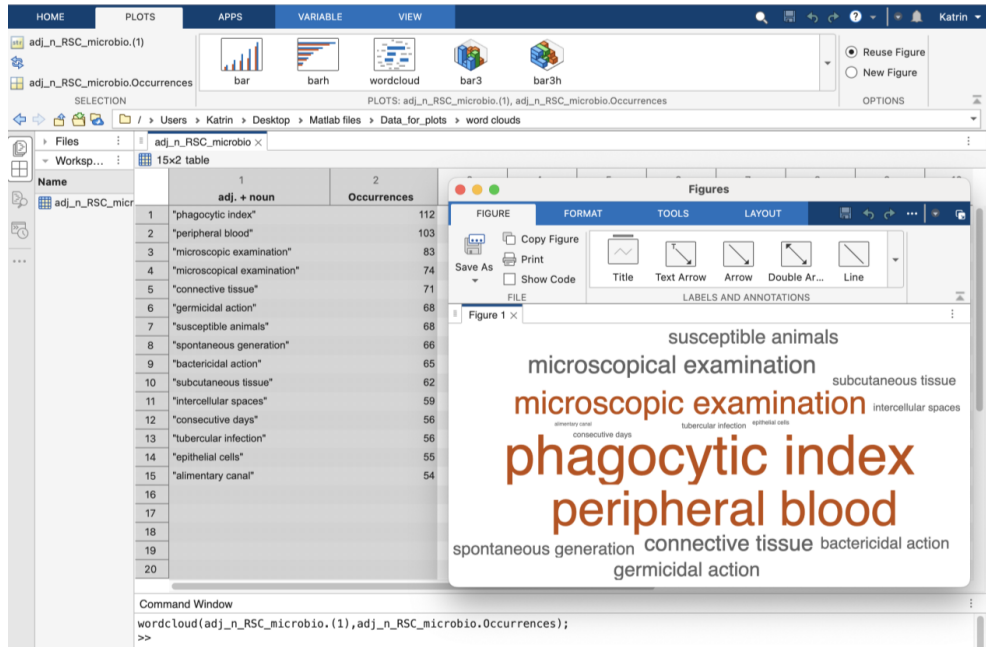


Figure 169: Word cloud with most frequent patterns of nouns after adjectives that have a minimum length of 10 characters (RSC microbiology texts, 1860s–1920)

There are many things you can do with MATLAB’s Text Analytics Toolbox when it comes to generating word clouds. We have only seen a few introductory examples here to visualise frequency data that we have extracted by specific queries from existing corpora. When you have the Text Analytics Toolbox installed, you can also create word clouds directly from string arrays. Let us have a look at a simple example to illustrate this. Imagine you have collected transcripts of speeches from American First Ladies, and now you want to visualise the themes or most common words in their speeches using a word cloud.

First, you can save the transcripts of speeches as strings. Here is a short extract from a statement by Melania Trump from the Women of Courage Awards in 2025 that you may save as a string variable (cf. <https://www.rev.com/transcripts/melania-trumps-speaks-at-the-women-of-courage-awards>).

```
str= "Good afternoon, and thank you Secretary Rubio. Throughout my life, I have harnessed the power of love as a source of strength during challenging times. Love has inspired me to embrace forgiveness, nurture empathy, and exhibit bravery in the face of unforeseen obstacles. Today we celebrate courage, a strength that is based in love. This truth is illustrated through our honorees who prove that love can inspire extraordinary valor even in the face of vulnerability. Their remarkable stories, a testament to the power of love in practice for family, community, and our world. This is especially evident in places where their passion defies convention and where it has been shown at great personal risk of stigma, persecution, or harm. The courageous Georgiana Pascu, one of our award recipients, exemplifies love in action. Georgiana is a watchdog who defends the dignity of Romanians whose voices cannot be heard. She fearlessly enters facilities designated as care centers to rescue people with disabilities who are unwittingly held captive. In fact,
```

Georgiana usually shows up unannounced and discovers the unimaginable, helpless adults and children bound, sedated, starving, and in extreme cases dying. Thanks to Georgiana's brave work, dozens of offenders have been charged with human trafficking, exploitation, and even organized crime."

The function `wordcloud(str)` creates a word cloud chart by tokenising and preprocessing your text in `str`, and then displaying the words with sizes corresponding to the word frequency counts (this currently works with English, Japanese, German and Korean text). You can type this word cloud command or select the string variable and click on the **wordcloud** icon to generate the following word cloud (Figure 170).

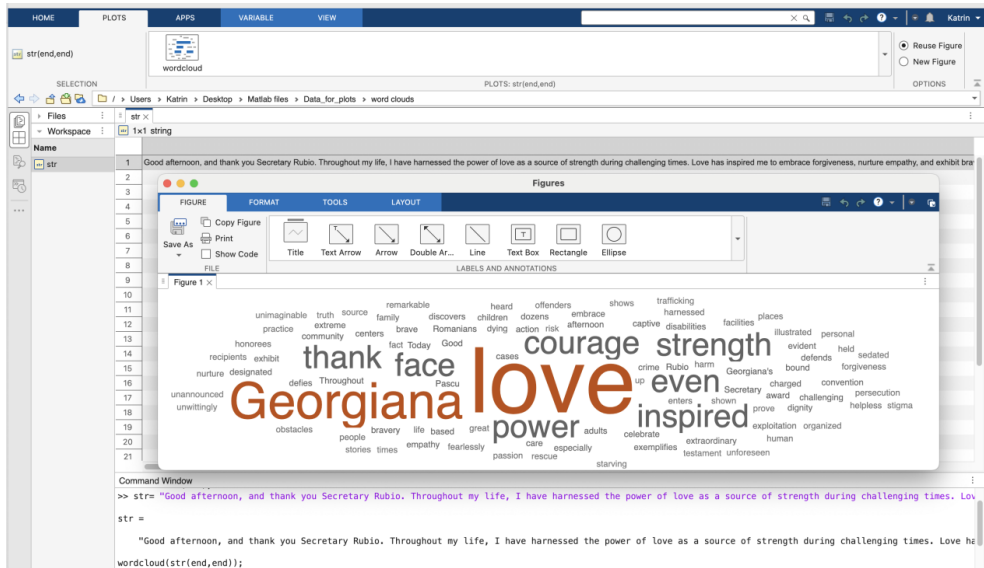


Figure 170: Word cloud created from a string

With the Text Analytics Toolbox installed you can also create word clouds directly from models like bag-of-words, bag-of-n-grams etc. (cf. Chapters 10.4.1 and 10.4.2 for examples).

If you are interested in further information on working with word clouds, you can check out the MATLAB documentation and tutorial videos on this topic, e.g.:

*Create word cloud chart from text data*

<https://www.mathworks.com/help/matlab/ref/wordcloud.html>

*Visualise text data using word clouds*

<https://www.mathworks.com/help/textanalytics/ug/visualize-text-data-using-word-clouds.html>

*Create word cloud chart from text, bag-of-words model, bag-of-n-grams model or LDA model*

<https://www.mathworks.com/help/textanalytics/ref/ldamodel.wordcloud.html>

*Count words for word cloud creation*

<https://www.mathworks.com/help/textanalytics/ref/wordcloudcounts.html>

## 7.14 Arranging multiple plots in a single figure

### 7.14.1 Creating a grid of subplots

When working with complex data visualisations, you may want to display multiple plots within a single figure window. MATLAB offers several functions to achieve this, both from the command line and within scripts. While it is possible to write detailed scripts to control the details of the figure layout, MATLAB also allows you to generate multiple plots in a single figure window using just a few simple commands. Once the plots are generated, you can further refine figure properties such as axis limits, colours and labels in an interactive way using the Property Inspector without needing to write additional code.

One possible function for this purpose is `subplot`, which allows you to divide a figure window into a grid layout and place an individual plot in each grid cell (<https://www.mathworks.com/help/matlab/ref/subplot.html>). This method is particularly useful for simple, fixed-size layouts where each subplot has equal size. The `subplot` function creates individual axes objects arranged in a matrix, and you can specify the position of each subplot. The syntax for this function is as follows:

```
subplot(m, n, p);
```

```
plot(data);
```

Here `m` is the number of rows in the layout, `n` is the number of columns and `p` specifies the position of the subplot in the grid.

For example, the command `subplot(3, 2, 5)` is used to create a grid of subplots with 3 rows and 2 columns. The third number (in this case, 5) specifies the position of the current plot. The numbering of positions follows a left-to-right, top-to-bottom order, starting with 1 in the top-left corner. So, in this example, the grid has 3 rows and 2 columns, resulting in 6 total positions. The 5th position corresponds to the plot's location, which would be in the third row, first column.

Let us now plot two bar charts within the same figure using the following example: We will use the bilingual GECCo (German-English Contrasts in Cohesion) corpus, which contains English and German texts (Kunz et al. 2021). The multilevel-annotated GECCo corpus provides comparable data across both written and spoken registers for these two languages. It was created as a resource for contrastive studies of cohesive devices. The GECCo corpus can be queried using the CQP corpus query language and various corpus versions are available at Saarland's University's CQPweb installation upon request<sup>29</sup> (cf. <https://fedora.clarin-d.uni-saarland.de/gecco/documentation.html>).

The written part of the GECCO corpus serves as a bi-directional parallel translation corpus, containing both original (non-translated) texts in English and German, as well as their sentence-aligned translations. The written part includes a variety of different corpus registers: political essays (ESSAY), fictional texts (FICTION), instruction manuals (INSTR), popular science texts (POPSCI), letters to shareholders (SHARE), prepared speeches (SPEECH), tourism leaf-

---

<sup>29</sup> Due to certain copyright restrictions, access to the GECCo corpus is not automatic even if you have access to the CQPweb interface at Saarland University; it can be granted upon request.

lets (TOU) and corporate websites (WEB). Table 12 presents the register sizes in tokens for the original English and German texts in the written part of the GECCO corpus.

GECCo corpus register	Register size in tokens: orig. Engl. texts	Register size in tokens: orig. German texts
ESSAY	34,998	35,668
FICTION	36,996	36,778
INSTR	36,167	36,880
POPSCI	35,148	36,177
SHARE	35,824	35,235
SPEECH	35,062	35,399
TOU	35,907	36,574
WEB	36,119	35,779

Table 12: Size of written corpus registers with original texts in GECCo corpus

We will compare the German and English data now using bar charts, as this will allow us to illustrate any differences or similarities in register sizes between the two languages visually.

To create two bar charts for the provided data using the `subplot` function in MATLAB, you can follow the example code below. This code will create a simple 1-by-2 subplot layout, where the first subplot displays the English data and the second subplot displays the German data.

```
% Data for English written texts
English_tokens = [34998, 36996, 36167, 35148, 35824, 35062, 35907, 36119];

% Data for German written texts
German_tokens = [35668, 36778, 36880, 36177, 35235, 35399, 36574, 35779];

% Create a figure
figure;

% First subplot for English data
subplot(1, 2, 1); % 1 row, 2 columns, first subplot
bar(English_tokens);

% Second subplot for German data
subplot(1, 2, 2); % 1 row, 2 columns, second subplot
bar(German_tokens);
```

You can then add further specifications for each subplot in the figure and via the Property Inspector (Figure 171). This allows you to change attributes like titles, axis labels etc. interactively.

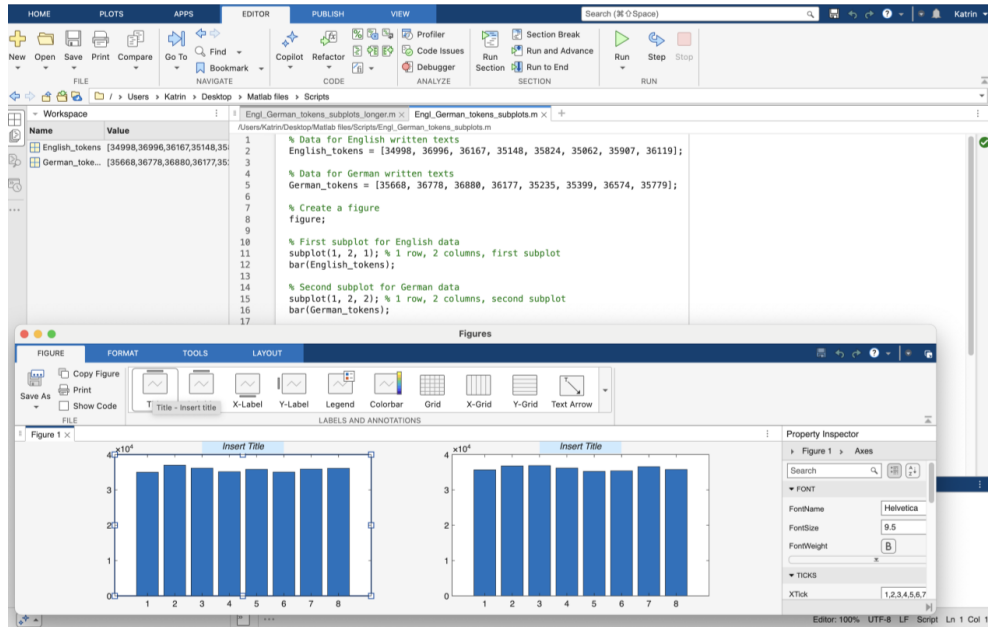


Figure 171: Modifying subplots in a figure interactively

A more programmatic approach might give you a bit more flexibility here in setting certain plot features directly through code. This would be especially useful when dealing with larger datasets or if you need to replicate the same style across multiple figures, sessions or projects in the future. This is something you might not be able to achieve as easily with manual adjustments via the Property Inspector. You would have to redo everything manually if you lost track of the settings. Moreover, when you are working with large datasets, adjusting certain layout features of plots interactively via the Property Inspector might become tedious.

We can configure more plot features programmatically via commands than in the example above, while still retaining the option to fine-tune the appearance further using the Property Inspector. In the following script, we set more features directly via commands to reduce the need for manual adjustments. For example, we no longer need to type in the x-axis tick labels in the Property Inspector for each subplot as this step is already included in this script:

```

% Data for English and German written texts from GECCO corpus
English_tokens = [34998, 36996, 36167, 35148, 35824, 35062, 35907, 36119];
German_tokens = [35668, 36778, 36880, 36177, 35235, 35399, 36574, 35779];
Registers = {'ESSAY', 'FICTION', 'INSTR', 'POPSCI', 'SHARE', 'SPEECH', 'TOU', 'WEB'};
% Since the registers in the German corpus match those in the English corpus,
we do not need two separate variables for them.

% Create a figure
figure;

% First subplot for English data
subplot(1, 2, 1);
bar(English_tokens);
set(gca, 'XTickLabel', Registers); % Set x-axis tick labels
title('Register size of original English written texts from GECCO');

```

```
ylabel('Number of tokens');

% Second subplot for German data
subplot(1, 2, 2);
bar(German_tokens);
set(gca, 'XTickLabel', Registers); % Set x-axis tick labels
title('Register size of original German written texts from GECCo');
ylabel('Number of tokens');

% Adjust layout and add shared title for the entire figure
sgtitle('Comparison of size of original written registers in GECCo');
```

This code will visually compare the token counts of the non-translated English and German written registers in the GECCo corpus side by side in a one figure, using two bar charts (Figure 172).

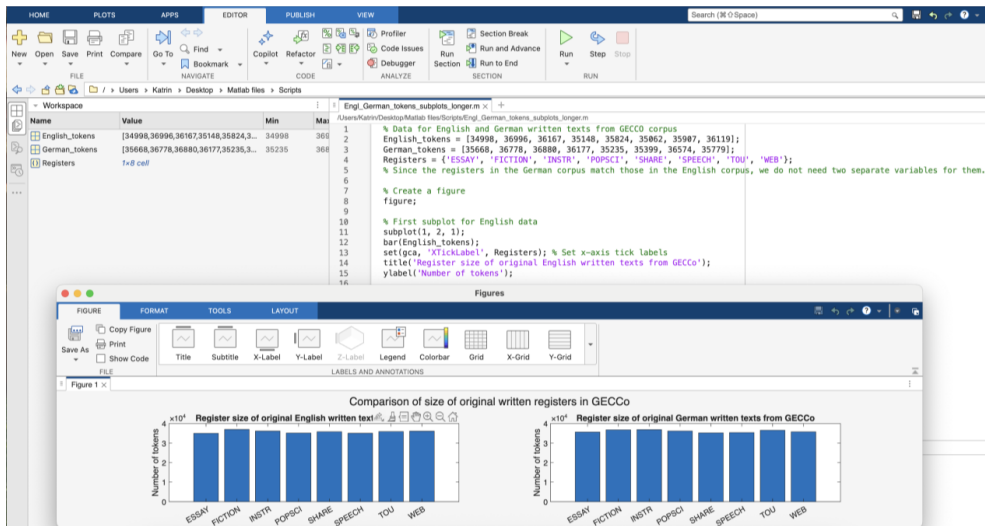


Figure 172: Generating subplots with tailored layout features via script

### 7.14.2 Arranging multiple plots with tiled chart layouts

MATLAB has recently introduced an alternative to the subplot function called the tiledlayout system. Like subplot, it allows you to arrange multiple plots in a grid layout – but with greater flexibility and control with regard to specific details such as adjusting spacing and alignment and adding shared titles across plots. You create the layout once with tiledlayout(rows, columns) and then place each plot with nexttile. In order to create a figure with two bar charts for our data from the GECCo corpus with minimal coding effort using the tiledlayout function you can write the following commands.

```
% Data for English written texts
English_tokens = [34998, 36996, 36167, 35148, 35824, 35062, 35907, 36119];

% Data for German written texts
German_tokens = [35668, 36778, 36880, 36177, 35235, 35399, 36574, 35779];

% Create tiled layout
tiledlayout(1, 2); % 1 row, 2 columns
```

```
% First tile: English data
nexttile;
bar(English_tokens);

% Second tile: German data
nexttile;
bar(German_tokens);
```

This series of commands will give you the same figure as the one in Figure 171 and you can further refine the bar charts interactively by adding labels, a title and other visual elements. To perform more of these steps programmatically, you can define a variable for the register names, as we saw earlier with the `subplot` example.

In the script below, we first define the necessary variables and then set up a tiled layout. We also add a shared title, specify subplot titles and set x-axis tick labels and x- and y-axis labels for clarity. Here, the variable `t` serves as a handle to the `tiledlayout` object, allowing us to reference and customise the entire layout. This script will produce the same figure as shown in Figure 172.

```
% Data for English written texts from GECCo corpus
English_tokens = [34998, 36996, 36167, 35148, 35824, 35062, 35907, 36119];
Registers = {'ESSAY', 'FICTION', 'INSTR', 'POPSCI', 'SHARE', 'SPEECH', 'TOU',
            'WEB'};

% Data for German written texts from GECCo corpus
German_tokens = [35668, 36778, 36880, 36177, 35235, 35399, 36574, 35779];

t = tiledlayout(1, 2); % 1 row, 2 columns
title(t, 'Comparison of size of original written registers in GECCo');

% First tile: English data
nexttile;
bar(English_tokens);
set(gca, 'XTickLabel', Registers);
title('English written texts');
ylabel('Number of tokens');

% Second tile: German data
nexttile;
bar(German_tokens);
set(gca, 'XTickLabel', Registers);
title('German written texts');
ylabel('Number of tokens');
```

### 7.14.3 Overlaying multiple plots

One way to display multiple plots in the same axes in MATLAB is to use the `hold on` function. By default, when you create a new plot, MATLAB clears the current axes and replaces the existing plot with the new one. However, using the `hold on` function changes this behaviour so that new plots are added on top of the existing ones without deleting them. This allows you to visualise multiple datasets together in a single figure, rather than in separate tiles or windows.

In the following example, we will see how to overlay two stem plots in the same axes. We will start by defining two datasets, then use the `stem` command to plot the first dataset. After that, we apply `hold on` to ensure the current plot remains visible, allowing the second dataset to be

added on the same axes. Then, we will use another `stem` command to plot the second dataset. As with any MATLAB figure, you can enhance the visualisation by adding a title, legend and axis labels – either programmatically or through the figure’s toolbar interface.

Let us take the data that we have already used in Figure 147 where we created a stem plot with sorted frequency data for cohesive ellipses in texts from five English registers from the GECCo corpus. We have also used these data in a box plot in Chapter 7.10 where we extended it with data from the German component of the GECCo corpus to visualise the English and German data of ellipses frequencies in the selected two parts from GECCo. Here, we will use the same data of ellipses frequencies in five corpus registers from both the English and German original components of the GECCo corpus, covering spoken and written language. The selected registers for our analysis include academic lectures, instruction manuals, letters to shareholders, prepared speeches and corporate websites. We combine the data from all five registers for each language to create aggregated datasets representing the English and German subsets of the corpus. Remember that elliptical constructions that function as cohesive devices link back cross-clausally to a textual antecedent. This includes nominal, verbal and clausal ellipses with such an antecedent (cf. examples of these constructions in Chapters 7.8 and 7.9).

Here we will compare the frequencies of these ellipses in the selected English and German parts from the GECCo corpus using two stem plots in one figure with sorted frequency data. The data was already provided in Table 9 above, and it is repeated here for convenience in Table 13.

Text ID in GO_GECCo	Cohesive ellipses (per 1,000 tokens)	Text ID in EO_GECCo	Cohesive ellipses (per 1,000 tokens)
GO_ACADEMIC_001	1.06	EO_ACADEMIC_001	2.8
GO_ACADEMIC_002	1.06	EO_ACADEMIC_002	3.13
GO_ACADEMIC_003	2.54	EO_ACADEMIC_003	0.93
GO_ACADEMIC_004	0.39	EO_ACADEMIC_004	0
GO_ACADEMIC_005	1.22	EO_ACADEMIC_005	0.52
GO_ACADEMIC_006	5.8	EO_ACADEMIC_006	1.95
GO_ACADEMIC_007	3.44	EO_ACADEMIC_007	0.52
GO_ACADEMIC_008	1.05	EO_ACADEMIC_008	1.41
GO_ACADEMIC_009	0	EO_ACADEMIC_009	1.18
GO_ACADEMIC_010	0.56	EO_ACADEMIC_010	1.64
GO_INSTR_001	0	EO_INSTR_001	0
GO_INSTR_002	0	EO_INSTR_002	0.32
GO_INSTR_003	0	EO_INSTR_003	0.28
GO_INSTR_004	0	EO_INSTR_004	0.28
GO_INSTR_005	0	EO_INSTR_005	0
GO_INSTR_006	0	EO_INSTR_006	0
GO_INSTR_007	0.59	EO_INSTR_007	0
GO_INSTR_008	0.29	EO_INSTR_008	0.27
GO_INSTR_009	2.27	EO_INSTR_009	0.26
GO_INSTR_010	0	EO_INSTR_010	0
GO_INSTR_011	0	EO_SHARE_001	1.09
GO_INSTR_012	0	EO_SHARE_002	0
GO_INSTR_013	0	EO_SHARE_003	0
GO_INSTR_014	0	EO_SHARE_004	2.04
GO_SHARE_001	0	EO_SHARE_005	0.22
GO_SHARE_002	0	EO_SHARE_006	0
GO_SHARE_003	0.49	EO_SHARE_007	0
GO_SHARE_004	0	EO_SHARE_008	0.54
GO_SHARE_005	0	EO_SHARE_009	0.43
GO_SHARE_006	0	EO_SHARE_010	0.37
GO_SHARE_007	0	EO_SHARE_011	0
GO_SHARE_008	0	EO_SHARE_012	0

Text ID in GO_GECCo	Cohesive ellipses (per 1,000 tokens)	Text ID in EO_GECCo	Cohesive ellipses (per 1,000 tokens)
GO_SHARE_009	0	EO_SHARE_013	1.17
GO_SHARE_010	0	EO_SPEECH_001	0.75
GO_SHARE_011	0	EO_SPEECH_002	1.43
GO_SPEECH_001	0	EO_SPEECH_003	0
GO_SPEECH_002	0	EO_SPEECH_004	0.29
GO_SPEECH_003	0	EO_SPEECH_005	0.4
GO_SPEECH_004	1.06	EO_SPEECH_006	0
GO_SPEECH_005	0.95	EO_SPEECH_007	0
GO_SPEECH_006	0	EO_SPEECH_008	0
GO_SPEECH_007	0.35	EO_SPEECH_009	1.16
GO_SPEECH_008	0.53	EO_SPEECH_010	1.08
GO_SPEECH_009	1.58	EO_SPEECH_011	0
GO_SPEECH_010	0.34	EO_SPEECH_012	0.49
GO_SPEECH_011	0	EO_SPEECH_013	1.13
GO_SPEECH_012	0.64	EO_SPEECH_014	0.38
GO_SPEECH_013	0.71	EO_WEB_001	0
GO_SPEECH_014	0	EO_WEB_002	0
GO_SPEECH_015	0	EO_WEB_003	0.48
GO_SPEECH_016	0	EO_WEB_004	0.59
GO_SPEECH_017	0	EO_WEB_005	1.04
GO_SPEECH_018	0	EO_WEB_006	1.23
GO_WEB_001	0.38	EO_WEB_007	0.27
GO_WEB_002	0	EO_WEB_008	0
GO_WEB_003	0	EO_WEB_009	0.55
GO_WEB_004	1.06	EO_WEB_010	0.39
GO_WEB_005	0	EO_WEB_011	0
GO_WEB_006	0	EO_WEB_012	0.24
GO_WEB_007	0		
GO_WEB_008	0.25		
GO_WEB_009	0		
GO_WEB_010	0.34		
GO_WEB_011	0.3		
GO_WEB_012	0		
GO_WEB_013	0		

Table 13: Cohesive ellipses per text in five registers (English and German part of GECCo corpus)

Sorting the data from texts with the fewest ellipses to those with the most allows us to visualise the full spectrum of usage across texts and to compare the shape of the curves for both languages. By overlaying the aggregated data from the five registers within each language using MATLAB's `hold on` function, we can present a single consolidated stem plot that directly compares the distribution of ellipses in English and German for the selected corpus data. You can copy the data from Table 13 into an Excel file and import them into MATLAB. For the purpose of this illustration, the frequency data are also provided below in respective commands – so you only need to copy them. Note that the frequencies shown are normalised values, representing the number of cohesive ellipses in each text per 1,000 tokens.

You can sort variables via a click on the respective variable and the selection of **Sort Ascending** or **Sort Descending** from the drop-down menu in the Variables editor (cf. Figure 60 and Figure 146 above). Alternatively, you can sort them using a command like `B = sort(A)`, as explained in the MATLAB documentation in Chapter 8.1.7.<sup>30</sup>

<sup>30</sup> cf. also <https://www.mathworks.com/help/matlab/ref/double.sort.html>

The following script creates the variables, sorts them with commands, plots them both in a single stem chart using the `hold on` function and adds labels and a legend via respective commands.

```
% Define the frequency data for cohesive ellipses in the German
% Original (GO) and English Original (EO) subsets of the GECCo corpus.
% These frequencies represent the combined data from five corpus registers
% from GECCo: academic lectures, instruction manuals, letters to
% shareholders, prepared speeches and corporate websites.

freq_ellipses_GO_acad_instr_share_speech_web = [1.06;1.06;2.54;0.39;1.22;
5.80;3.44;1.05;0;0.56;0;0;0;0;0;0;0.59;0.29;2.27;0;0;0;0;0;0.49;0;0;0;
0;0;0;0;0;0;1.06;0.95;0;0.35;0.53;1.58;0.34;0;0.64;0.71;0;0;0;0;0.38;0;
0;1.06;0;0;0;0.25;0;0.34;0.30;0;0];

freq_ellipses_EO_acad_instr_share_speech_web = [2.80;3.13;0.93;0;0.52;1.95;
0.52;1.41;1.18;1.64;0;0.32;0.28;0.28;0;0;0;0.27;0.26;0;1.09;0;0;2.04;0.22;0;
0;0.54;0.43;0.37;0;0;1.17;0.75;1.43;0;0.29;0.40;0;0;1.16;1.08;0;0.49;1.1
3;0.38;0;0;0.48;0.59;1.04;1.23;0.27;0;0.55;0.39;0;0.24];

% Sort the data in ascending order, ranging from texts with the fewest
% cohesive ellipses to those with the most.

freq_ellipses_GO_acad_instr_share_speech_web_sorted =
sort(freq_ellipses_GO_acad_instr_share_speech_web);

freq_ellipses_EO_acad_instr_share_speech_web_sorted =
sort(freq_ellipses_EO_acad_instr_share_speech_web);

% Create the stem plot for GO_GECCo data

figure; % Create a new figure

stem(freq_ellipses_GO_acad_instr_share_speech_web_sorted, 'DisplayName', 'GO
Frequencies (German data)');

hold on; % Retain the current plot

% Create the stem plot for EO_GECCo data
stem(freq_ellipses_EO_acad_instr_share_speech_web_sorted, 'DisplayName', 'EO
Frequencies (English data)');

% Add labels and legend

xlabel('Corpus texts - combined data from five registers in GECCo: ACADEMIC,
INSTR, SHARE, SPEECH, WEB (sorted)');

ylabel('Frequency of ellipses per 1000 tokens');

title('Frequencies of cohesive ellipses in German and English subsets of the
GECCo corpus');
legend show; % Display the legend
hold off; % Release the hold state
```

Running this script will generate a figure as shown in Figure 173, which displays the overlaid stem plots. Even though the number of texts differs between the English and German register datasets, their overall size in terms of tokens is comparable. This allows for a meaningful interpretation of the shape of the sorted frequency curves, independent of text count.

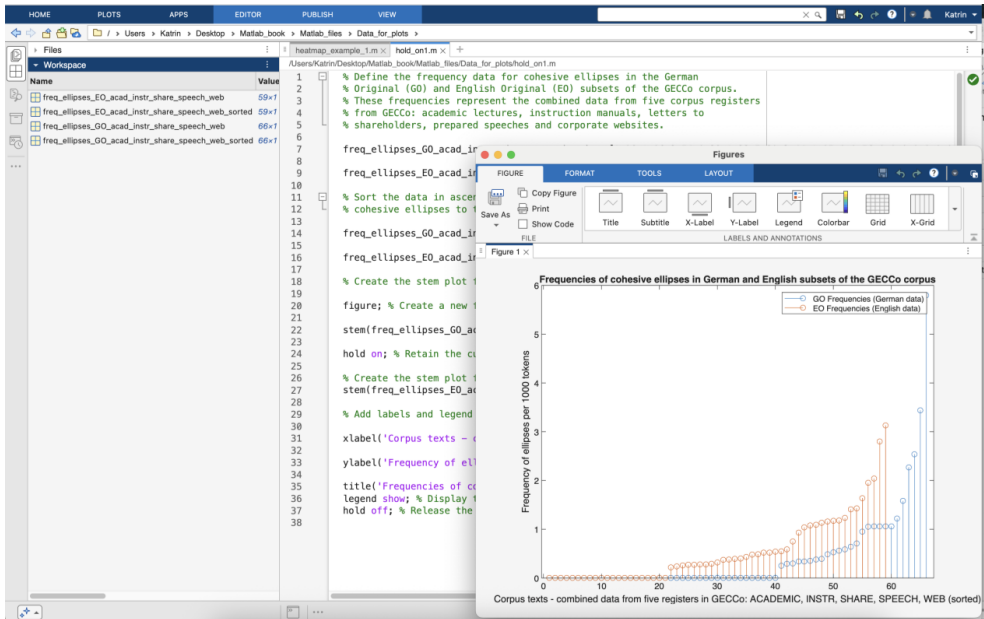


Figure 173: Sorted frequencies of cohesive ellipses in 5 English and German GECCo registers, overlaid in a single stem plot for direct comparison

In this script, the legend, title and axis labels have already been added using MATLAB commands. However, these elements can also be added and modified interactively using the figure’s toolstrip options, allowing for easier customisation without the need to write code for formatting or annotation. Further visual details – such as line styles, colours and markers – can also be modified directly within the figure window. For example, it is advisable to use different markers for each dataset if you represent them in one figure, so that both stem plots are not represented with identical symbols (such as circles) as this might otherwise lead to visual confusion when interpreting overlapping points (Figure 174).

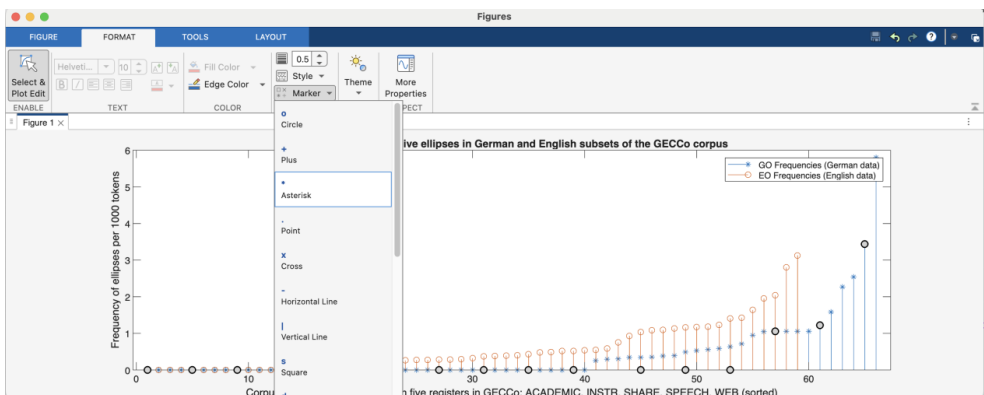


Figure 174: Selecting distinctive markers via the figure’s Format tab

In this stem plot, we can see that many texts in both the German and the English section form the GECCo corpus have zero values for cohesive ellipses. The German dataset has a larger

proportion of texts with an absolute frequency of zero cohesive ellipses, which might highlight a general difference between the two languages in the usage of ellipses as cohesive devices. The German dataset also shows a higher maximum value in fragment frequency compared to the English dataset. This higher peak may reflect that ellipsis usage in German is less evenly distributed, with certain texts standing out as outliers with especially frequent use of cohesive ellipses.

## 7.15 Summary of key MATLAB® figure customisation options and plotting commands

In the previous sections of this chapter, we have explored various customisation options in the context of individual MATLAB figure types. This section serves as a summary and overview of the most important interactive figure customisation options and plotting commands discussed, along with additional commands that may be generally useful for creating and refining MATLAB visualisations. We have seen that, using the MATLAB Property Inspector, various aspects of different plot types can be modified – including bar charts, pie charts, line plots, stem plots, scatter plots, box plots, histograms, heatmaps and word clouds. Below is a summary of key customisation features and commands for the different plot types.

Within the figures themselves, you have various interactive options to customise your figure. Under the **Figure** tab of each figure type, you will always find a toolbar for labels and annotations. For instance, options for adding a figure title – and, if desired, a subtitle – along with axis labels, a legend or a grid is represented by icons there. By clicking the **Show More** button to the right of these icons, a drop-down menu appears with additional options (cf. also Chapter 7.2). These include the ability to add annotations such as text boxes, various grid styles and different types of shapes and arrows (Figure 175). To the right of the search bar, you can switch between **Icon View** and **Details View**. The latter provides more information beneath each icon and explains briefly what actions can be performed when selecting it.

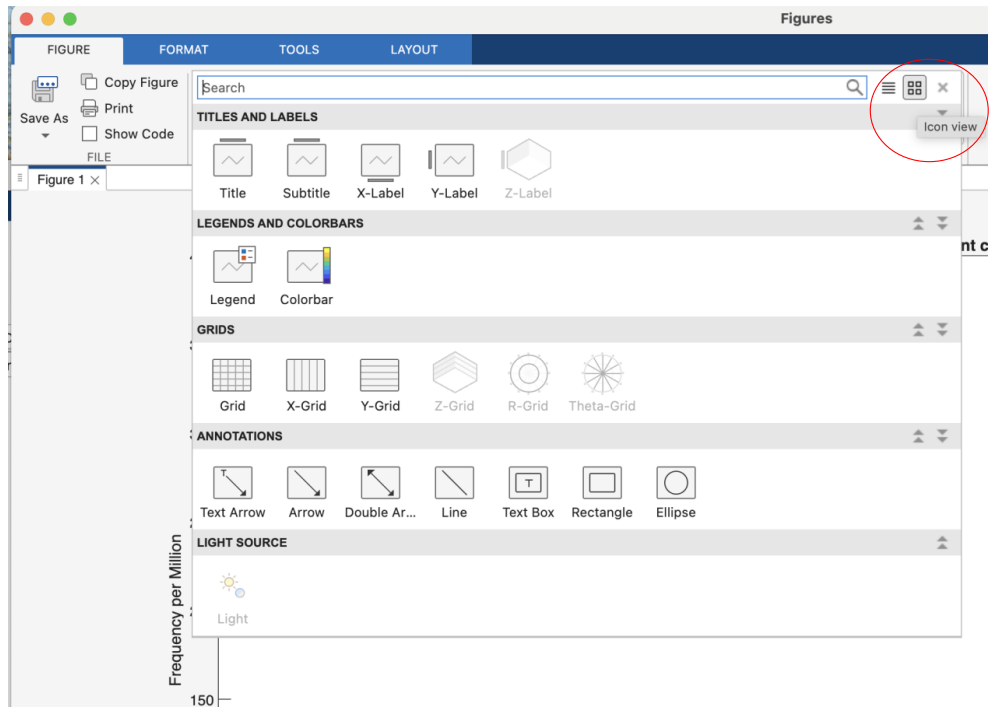


Figure 175: Labels and annotation options in the MATLAB Figure tab

Under the **Format** tab in MATLAB figures, you can click on **Select & Plot Edit**, which enables the plot edit mode and gives you access to several icons for customising the appearance of your figure. These include options to adjust font properties for titles, axis labels and legends. You can also modify line styles (e.g., solid, dashed), line width and, where applicable, marker types for plotted data. Additionally, you can change the colours of plot elements. To access more detailed settings, click on **More Properties** to open the **Property Inspector** (Figure 176). Alternatively, after creating a plot, you can open the Property Inspector by entering the command `inspect;` in the Command Window.

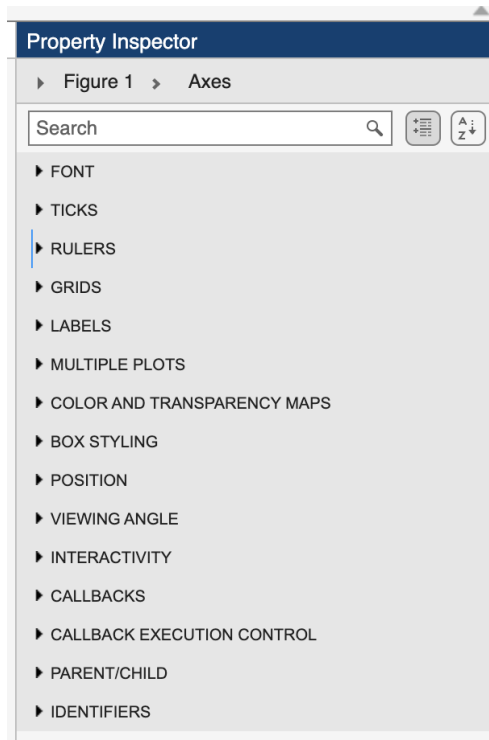


Figure 176: Using the Property Inspector

The Property Inspector provides a user-friendly interface for modifying graphics object properties. The properties you can modify via the Property Inspector are grouped into main categories such as axes properties, object properties and appearance settings. The main categories contain submenus that further break down properties into more specific settings, such as fonts, ticks, grids and labels for axes. You can organise the properties either alphabetically or by groups. Additionally, a search field is available for locating specific settings quickly. Users can either search for a setting or click directly on a plot element – such as an axis label in a bar chart or a box in a box plot – to access its associated properties automatically. These properties can then be edited directly in the corresponding field within the relevant category, and the changes will be instantly reflected in the figure. Figure 177 shows an example where the submenu **Text** of the axes properties category has been opened via a click on the x-axis label. Now you can change the text string of the axis label, the colour of the text, the font type and size, the rotation etc.

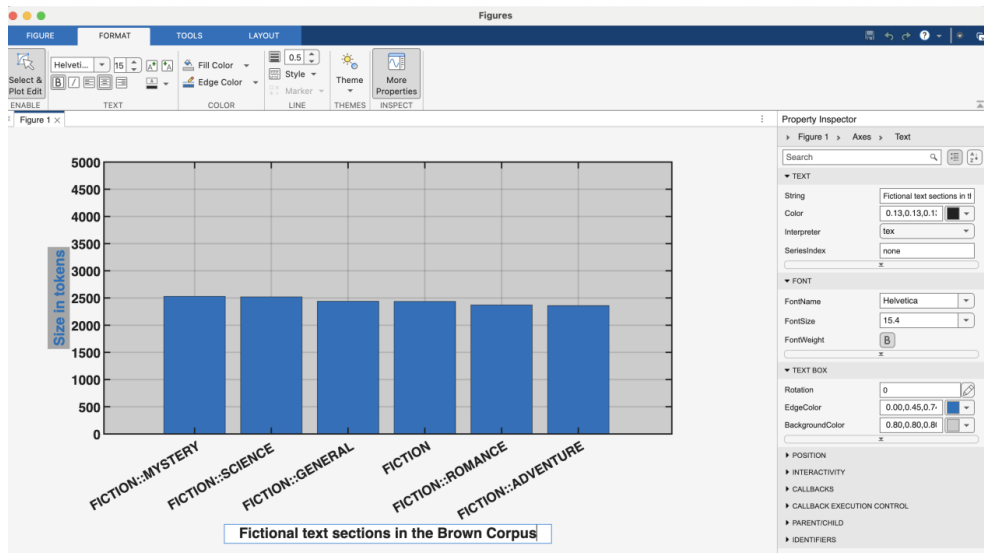


Figure 177: Accessing submenu of Property Inspector to modify the features of the x-axis label

For instance, for plots such as bar charts, you can customise various aspects interactively via the Property Inspector. You can change the fill colour of the bars, the width of the bars and the colour of the bar outlines. The appearance and behaviour of the axes can also be tailored, including axis limits, tick marks and labels. You can turn grid lines on or off to enhance visual guidance. Furthermore, the font size, style and colour of the chart title and axis labels can be modified to suit your presentation needs. The legend associated with the bar chart is also fully customisable.

For figures such as pie charts, you can edit, for instance, the labels for each slice of the chart and modify the colours of the slices interactively. The chart title is fully customisable, including adjustments to the font size, style and colour. The legend associated with pie charts can be tailored in terms of its position, font properties and general appearance. You also have the option to offset individual slices to highlight specific segments more clearly. To enhance the visual styling, you can change the edge colour and line width of the slices. The interactive options for line plots include, for instance, possibilities for customising line styles, colours, marker types, axes limits and labels. For stem plots, the appearance of the vertical stems and markers can be tailored, including their colour and line style. As always, axes settings and grid or legend options can also be modified as needed. In scatter plots, marker shape, size and colour can be changed as well as axes properties, legends etc. With box plots, you can adjust the colour of the boxes, change whisker styles and control the appearance of outliers. Here also, of course, axes limits and labels are also configurable. In histograms, the fill and edge colours of the bins can be changed, and you have control over the number and width of the bins. You can also modify various properties of heatmaps through the Property Inspector, such as adjusting the colour scheme or customising labels and the title to suit your preferences. For word clouds, the font style, size and colour can be customised, as well as the layout of the words and whether you want a highlight colour, i.e., a specific colour to emphasise the most frequent words within the visualisation.

In the **Tools** menu of the figures, you can find options for **Basic Fitting** and **Data Statistics**, which are useful for analysing and modelling your data. The **Basic Fitting** tool allows you to fit curves or surfaces to your data interactively. You can choose from various models and visualise how well the model fits your data. The **Data Statistics** tool provides basic statistical values of your data from various figure types, such as the mean, median, mode, maximum, minimum and the standard deviation.

Below is an overview and recap of some common MATLAB commands that are particularly useful when creating and customising plots. The command `plot(x, y)` generates a basic line plot of the data in `y` against the corresponding values in `x`. Other types of figures can be created using dedicated commands such as `bar(data)` for a standard bar graph or `bar(data, 'stacked')` for a stacked bar chart. You can use `scatter(x, y)` to create a scatter plot, `histogram(data)` to visualise the distribution of values and `wordcloud(words, frequencies)` to display text data graphically.

Axis labels can be added using the `xlabel` and `ylabel` commands, for example `xlabel('Corpus sections')` and `ylabel('Frequency')`.

To add a title to your plot, use the `title` function, e.g., `title('Frequency of nouns per decade')`.

Legends can be added to help identify plotted data series, e.g., `legend('English data', 'German data')`.

If you want to overlay multiple plots in the same figure, use `hold on` after the first plotting command. For instance:

```
plot(x, y);  
  
hold on;  
  
plot(x, z);
```

Axis limits can be controlled using `xlim` and `ylim`. For example, use `xlim([0 1000])` to limit the x-axis to a range of 0 to 1000 or `ylim([0 1000])` for the y-axis.

To change the orientation of certain plots, such as a box plot, you can include the orientation parameter: `boxplot(data, 'orientation', 'horizontal')`.

All of these commands are documented in MATLAB's official documentation. Have a look at the documentation on two- and three-dimensional plots and images under <https://www.mathworks.com/help/matlab/graphics.html> and the relevant subchapters on labels and styling etc. Additionally, autocompletion in the Command Window is a useful feature that suggests relevant commands and variable names as you type. MATLAB makes it easy to create and customise plots as it offers well-documented options for both command-based and interactive plotting, with the interactive features being particularly intuitive. You can use either method or a combination of both to experiment with different plot types and layouts. In many cases, it will also be very helpful to turn to MATLAB's Copilot chat for assistance with any questions that arise while creating a specific plot and customising it.

## 7.16 Best practices for effective data visualisation in MATLAB®

We have explored various plot types and specific adjustments that can be made to each of them in MATLAB. To conclude this chapter, let us summarise the key takeaways and best practices for visualising linguistic data effectively. Generally, it is useful to keep in mind some best practices for creating effective visualisations, which you will probably already be familiar with. Examples include ensuring that axes are clearly labelled with descriptive titles and units, where necessary, to provide context for the data. The scales on both axes should be carefully selected to avoid distorting the data, with intervals that are easy to interpret and consistent across multiple plots. To improve readability, unnecessary elements, such as excessive grid lines or ticks, should be removed. Colour can be a powerful tool for distinguishing between different groups or trends, but it is important to choose colours that are easily distinguishable for all viewers, including those with colour blindness. For plots involving multiple datasets, a legend should be included in a non-obtrusive location. A clear, concise title should explain the plot's purpose, and the design should be tailored to the audience, presenting only the most relevant information. Data points should be large enough to be readable without overcrowding the plot. Consistency in style, colours and layout across multiple plots will also help to make comparisons easier.

These principles of visualisation design will ensure that your plots are clear, informative and accessible to a wide range of viewers. Also be reminded that after making any changes, it is advisable to save your figure again, either by overwriting the existing file or by saving it under a different name to maintain multiple versions, especially if you plan to use it in an academic paper or presentation. It can be very useful to save different versions of your plots to track changes or experiment with various layouts and settings.

By exploring various plot types, such as bar charts, pie charts, line plots, stem plots, scatter plots, box plots, histograms, heatmaps and word clouds, we have seen how to visualise different types of corpus data in MATLAB. Each of these plot types offers distinct advantages, depending on the nature of the data and the specific relationships you wish to highlight. Vertical and horizontal bar charts are excellent for comparing different categories of data, such as the frequencies of linguistic features across various text types represented in a corpus. Stacked bar charts are particularly useful when you want to visualise the composition of different categories within a group and to compare both the total values across categories and the individual components that make up those totals. Pie charts, on the other hand, are ideal for representing the proportionate distribution of categories in a dataset. They can be used, for example, to illustrate how different proportions of a corpus are distributed across specific metadata categories, such as various speaker or author types. Line plots are especially valuable when dealing with continuous data, such as representing linguistic trends over time. For example, they can illustrate how the frequency of certain linguistic features changes across different time periods or documents within a corpus. Stem plots present data as vertical lines extending from a baseline, with markers indicating the data values. This makes it easier to identify individual data points and their magnitudes, which is particularly beneficial when analysing counts or frequencies in language data. Scatter plots are excellent for visualising the relationship between two variables. In the context of linguistic data, scatter plots can help investigate correlations, such as the relationship between the usage of two different linguistic structures within a dataset. Box plots provide a clear view of data distribution, highlighting medians, quartiles and outliers. They are useful when you want to examine the spread of data and identify any anomalies in linguistic features. Histograms, similarly, are effective for understanding the distribution of data and can help reveal the spread and central tendency of variables within a corpus. Heatmaps are useful

tools to show how different variables relate to each other by using colours to represent the amount or strength of data. Lastly, word clouds offer a simple, yet effective, way to visualise the most frequent words or phrases in a dataset. The size of each word corresponds to its frequency, making word clouds visually engaging tools.

Each of these plot types provides a unique means of representing and analysing linguistic data. However, the effectiveness of these plots hinges not only on the choice of the plot type but also on how well it is customised for clarity, accessibility and visual impact. Adjusting elements such as axis labels, colour schemes and general layout can significantly enhance the readability and interpretability of your visualisations.

# 8 Basic arithmetic operations and descriptive statistics in MATLAB®

## 8.1 Basic arithmetic operations and number processing

### 8.1.1 Overview

In the individual sections of this chapter, we will briefly introduce some fundamental MATLAB commands for basic arithmetic and mathematical operations, including addition, subtraction, multiplication, division, exponentiation, square root, logarithmic functions and rounding. Since the focus of this book is primarily on getting started with MATLAB, particularly for interactive data analysis and applications in linguistic research, this will be a concise overview. For further details, consult the MATLAB documentation specifically on this topic. For instance, the site <https://www.mathworks.com/help/matlab/arithmetic.html> provides comprehensive information on arithmetic operations, including links to related subtopics and individual functions. You can find additional information on arithmetic operations in MATLAB (i.e., array operations and matrix operations) under the following link: [https://www.mathworks.com/help/matlab/matlab\\_prog/array-vs-matrix-operations.html](https://www.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html).

In corpus linguistics, we work with datasets of varying sizes that capture features such as word frequencies, sentence lengths, grammatical patterns and co-occurrences. MATLAB offers a range of arithmetic tools to manipulate and analyse such data. This chapter introduces key arithmetic operations in MATLAB that form the foundation for quantitative corpus analysis, and it will help you to start working with numerical data in MATLAB. We will focus on a selection of practical examples that you can apply and adapt in your own corpus analyses. We begin with a look at how addition works in MATLAB.

### 8.1.2 Addition

In MATLAB, addition is simply carried out using the plus sign and it can be applied to either individual values or arrays. This means you can add two single numbers or perform element-wise addition on two arrays of the same size. For example, you might wish to add frequency counts of particular words or constructions taken from different corpora.

You can use MATLAB as a calculator for simple arithmetic operations like addition directly in the Command Window. To display the result of an operation, do not use a semicolon at the end of the command, as this suppresses the output. For example, you can simply type: `1200 + 65` or `1200 + 65 + 15`. When you execute such a command, MATLAB will compute the sum and display the result. You can also assign the result to a variable if you want to use it later, e.g., `Sum_frequencies = 1200 + 65`.

In the following example, we will calculate the total frequencies of selected words across both the British National Corpus (BNC, Davies 2004, <https://www.english-corpora.org/bnc/>) and the TIME Magazine Corpus (Davies 2007, <https://www.english-corpora.org/time/>), each containing approximately 100 million words. We first query the frequencies of the four synonyms “happily”, “joyfully”, “cheerfully” and “merrily” using the following expression in the respective corpus interfaces: `happily|joyfully|cheerfully|merrily`. Table 14 shows the absolute frequencies of these words in the British National Corpus and the TIME Magazine

Corpus. In all four cases, the words appear more frequently in the TIME Magazine Corpus than in the BNC.

Word	British National Corpus	TIME Magazine Corpus
happily	1699	2969
cheerfully	543	979
merrily	157	308
joyfully	78	213

Table 14: Absolute frequencies of four words in BNC and TIME Magazine Corpus

Suppose we first wish to calculate the total frequency for “happily” across both corpora. This can be done using simple addition:

```
Frequency_happily_BNC_TIME = 1699 + 2969;
```

The value 4668 will be assigned to the variable `Frequency_happily_BNC_TIME`. If you omit the semicolon (;) at the end of the command, MATLAB will display the result directly in the Command Window. You can also create variables for the two numbers first before adding them, e.g.:

```
Frequency_happily_BNC = 1699;
```

```
Frequency_happily_TIME = 2969;
```

```
Frequency_happily_BNC_TIME = Frequency_happily_BNC + Frequency_happily_TIME;
```

To combine all values at once, the most efficient way would probably be to store the word frequencies as arrays first:

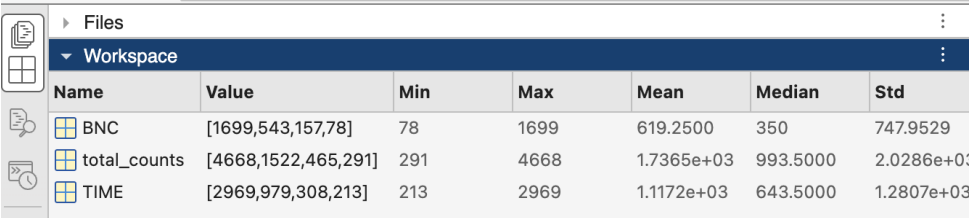
```
BNC = [1699, 543, 157, 78]; % Absolute frequencies for 'happily',  
'cheerfully', 'merrily', 'joyfully' in BNC
```

```
TIME = [2969, 979, 308, 213]; % Corresponding frequencies in TIME Magazine  
Corpus
```

Then, we simply add the arrays to obtain the total counts for each word:

```
total_counts = BNC + TIME;
```

This results in an array `total_counts` where each element represents the combined frequency for the respective word across the two corpora (Figure 178).



Name	Value	Min	Max	Mean	Median	Std
BNC	[1699,543,157,78]	78	1699	619.2500	350	747.9529
total_counts	[4668,1522,465,291]	291	4668	1.7365e+03	993.5000	2.0286e+03
TIME	[2969,979,308,213]	213	2969	1.1172e+03	643.5000	1.2807e+03

Figure 178: Workspace showing variables BNC and TIME, with `total_counts` created by summing the two

It would also be possible to add the two arrays `[1699, 543, 157, 78]` and `[2969, 979, 308, 213]` directly in one MATLAB command without first assigning them to variables. In this case you could write:

```
total_counts = [1699, 543, 157, 78] + [2969, 979, 308, 213].
```

When performing many operations like this or when working with multiple datasets, it is important to give your variables meaningful names that clearly describe their content or purpose to you, e.g., `Total_counts_4_words_BNC_TIME`. However, it is important to keep in mind that the more detailed the variable names are, the longer they can become, which may impact readability and code length. Therefore, try to find a good balance between clarity and brevity. Additionally, you can save your workspace variables with meaningful and descriptive file names to keep your data organised and easily accessible for future use. This practice will make it easier to share or revisit your work later.

MATLAB also provides the built-in function `sum` that you can use to compute the sum of elements in a vector or a matrix. If `A` is a vector, then `sum(A)` returns the sum of all elements. If `A` is a matrix, then `sum(A)` returns a row vector containing the sum of each column.

Let us look at an example:

```
A = [1, 2; 3, 4; 5, 6];
columnSums = sum(A);
```

In the example, `A` is defined as a 3x2 matrix:

```
A =
     1     2
     3     4
     5     6
```

For the given matrix `A`, the sums are then calculated for the first column as  $(1 + 3 + 5 = 9)$  and for the second column as  $(2 + 4 + 6 = 12)$ . The result will be `columnSums = [9, 12]`.

To compute the sum of each row, you can specify this dimension using `sum(A, 2)`. The second argument `2` specifies that the summation should occur across the columns for each row. That means, for each row in the matrix, MATLAB will add up all the elements in that row. Using the same matrix `A`, the command `rowSums = sum(A, 2)` will lead to the result `rowSums = [3; 7; 11]`. The sums are calculated as follows: First row:  $(1 + 2 = 3)$ , second row:  $(3 + 4 = 7)$ , third row:  $(5 + 6 = 11)$ .

The `sum` function can also handle missing values by using options like `'omitnan'` to ignore any NaN values. This can be useful when you want to compute the sum of elements in an array where missing values are common. Consider the following example:

```
A = [1, 2;
     3, NaN;
     5, 6];
```

The command `columnSums = sum(A);` will give you the following result `columnSums = [9, NaN]`. To ignore the NaN value and compute the sum of the columns without it, you can use the 'omitnan' option as in:

```
columnSumsIgnoringNaN = sum(A, 'omitnan')
```

In this case, the first column sum is  $(1 + 3 + 5 = 9)$ , the second column sum is  $(2 + 6 = 8)$ . NaN is ignored here. The result will be `columnSumsIgnoringNaN = [9, 8]`.

Let us assume you want to analyse the occurrence of verbs and nouns that begin with a hyphenated prefix, such as “co-”, “re-” or “pre-”, across several individual corpus texts. Examples include verbs like “co-occur”, “re-enter” and “pre-process” and nouns such as “co-occurrence”, “re-entry” and “pre-processing”. Suppose you extract the frequencies of these words from five different texts in a corpus that contains entire texts which can be searched individually. We can, for instance, use a dataset such as the Royal Society Corpus (RSC, <https://corpora.clarin-d.uni-saarland.de/cqpweb/>, cf. Chapter 4.4), where such patterns can be searched across different academic research papers. Table 15 presents the frequencies of hyphenated verbs and nouns with prefixes across five corpus texts.

Text	Hyphenated verbs with prefix	Hyphenated nouns with prefix
Text 1	8	5
Text 2	3	4
Text 3	5	3
Text 4	2	6
Text 5	7	4

Table 15: Frequencies of hyphenated verbs and nouns with prefixes (e.g., ‘co-’, ‘re-’, ‘pre-’) in five corpus texts

You can store this data in MATLAB as a numeric matrix:

```
hyphenated_words = [8, 5;
                    3, 4;
                    5, 3;
                    2, 6;
                    7, 4];
```

In this matrix, each row corresponds to a text, the first column to hyphenated verbs, and the second column to hyphenated nouns. To compute the sum of each column (all hyphenated verbs and hyphenated nouns in the texts), you can use:

```
columnSums_verbs_and_nouns = sum(hyphenated_words);
```

The result will be `columnSums_verbs_and_nouns = [25, 22]`.

If you want to compute the sum of each row (the total for each text), you can specify the dimension as 2:

```
rowSums_verbs_and_nouns_per_text = sum(hyphenated_words, 2);
```

This will return a column vector where each element is the sum of the corresponding row. The result will be `rowSums_verbs_and_nouns_per_text = [13; 7; 8; 8; 11]`.

You can refer to the MathWorks documentation on addition under: <https://www.mathworks.com/help/matlab/arithmetic.html>, <https://www.mathworks.com/help/matlab/ref/double.sum.html> and <https://www.mathworks.com/help/matlab/ref/double.plus.html>.

### 8.1.3 Subtraction

Many operations and functions for addition and subtraction in MATLAB work similarly, especially when dealing with arrays matrices and missing values. Subtraction uses the minus sign, and you can subtract individual values or perform element-wise subtraction on arrays of equal size. This operation is useful in corpus linguistics, for instance, when we want to remove subsets of data from a larger set. You can perform subtraction directly in the Command Window with or without assigning variable names, e.g., typing `100 - 45` or `Frequency = 100 - 45`.

Continuing with the example from Chapter 8.1.2 above, we will be looking at an example of frequency data from the TIME Magazine Corpus (<https://www.english-corpora.org/time/>) for selected synonyms (“happily”, “joyfully”, “cheerfully” and “merrily”). Now, suppose we want to refine our analysis of the TIME Magazine Corpus by excluding the frequencies of these words from a specific time period, e.g., a decade that stands out as particularly high compared to other decades and makes up a significant portion of the overall counts. The corpus interface of this corpus allows us to display frequency results by section (in this case, by decade) by ticking the box next to **Sections** before we query the words. By using a query such as: `happily|joyfully|cheerfully|merrily` we obtain the absolute frequencies for all four words in the entire corpus and across the different decades from the 1920s to the 2000s. We will note in the results that we obtain with this query that the frequencies in the 1950s stand out in particular as relatively high. Table 16 lists the total frequencies and the 1950s frequencies for the queried words in the TIME Magazine Corpus:

Word	Total Frequency	Frequency in 1950s
happily	2969	762
cheerfully	979	208
merrily	308	67
joyfully	213	35

Table 16: Absolute frequencies of four words in TIME Magazine Corpus overall and specifically in 1950s

To exclude the values from the 1950s from the overall TIME Magazine counts, we can subtract the 1950s data from the full counts using simple subtraction:

```
happily_TIME_excl_1950s = 2969 - 762;
```

Alternatively, we can first assign these numbers to variables. Using variables may generally be helpful if we want to reuse our values or perform similar operations for similar data in a further step.

```
happily_total_freq_ = 2969;
```

```
happily_1950s = 762;
```

```
happily_excl_1950s = total_happily - happily_1950s;
```

Extending this approach to all four words, we can work with arrays to subtract the 1950s frequencies from the total counts all at once:

```

TIME_total_freq = [2969, 979, 308, 213]; % Total frequencies for 'happily',
'cheerfully', 'merrily', 'joyfully' in TIME Magazine corpus

TIME_freq_1950s = [762, 208, 67, 35]; % Frequencies from 1950s for the words

TIME_freq_excl_1950s = TIME_total_freq - TIME_freq_1950s; % Subtracting
1950s frequencies for the words from the total frequencies in the corpus

```

The variable `TIME_freq_excl_1950s` will give us the frequencies of the words in the Time Magazine Corpus after excluding those from the 1950s.

There is also the `diff` function in MATLAB. You can use it to calculate the differences between adjacent elements in a vector or between rows in a matrix. Using MATLAB's `diff` function, you can calculate, for instance, the change in frequency between consecutive texts, which may help to identify trends such as increases or decreases in usage. For instance, suppose you have the counts of hyphenated verbs with prefixes (like “co-”, “re-”, “pre-”) across a sequence of five texts (cf. example above in Chapter 8.1.2 in Table 15).

```
hyphenated_verbs = [8, 3, 5, 2, 7];
```

You can calculate the differences between consecutive texts using:

```
differences_between_texts = diff(hyphenated_verbs);
```

The output variable `differences_between_texts` will contain the differences between each pair of consecutive elements in `hyphenated_verbs`. The result will be `[-5, 2, -3, 5]`. If an element in `differences_between_texts` is positive, it indicates an increase in frequency from the previous text to the current text. For instance, the value 2 indicates that the frequency increased by 2 from Text 2 (3 occurrences) to Text 3 (5 occurrences). If an element in `differences_between_texts` is negative, it indicates a decrease in frequency. For instance, the value -5 indicates that the frequency dropped from Text 1 (8 occurrences) to Text 2 (3 occurrences). If an element was zero in the result, it would indicate no change in frequency.

For additional examples and details, you can refer to the MathWorks documentation at: <https://www.mathworks.com/help/matlab/ref/double.minus.html> and <https://www.mathworks.com/help/matlab/ref/double.diff.html>.

### 8.1.4 Multiplication and division

Mathematical operations like multiplication and division are useful in corpus linguistic, for instance, for normalisation tasks. MATLAB offers two main types of multiplication and division operations: standard matrix operations and element-wise operations.

Use the `*` operator for matrix multiplication which follows linear algebra rules. We will not cover detailed examples here, but you can find some in the MATLAB documentation (<https://www.mathworks.com/help/matlab/ref/double.mtimes.html>).

Element-wise multiplication, denoted by the `.*` operator, is used when you need to multiply corresponding elements of two arrays (matrices or vectors). Unlike matrix multiplication, this operation does not follow linear algebra rules, but instead multiplies the elements one by one. Both arrays must be of compatible sizes. For more information on the requirements of this

operation, you can refer to MATLAB's official documentation at <https://www.mathworks.com/help/matlab/ref/double.times.html>.

Here is an example to illustrate this operation:

```
A = [1, 2];
```

```
B = [5, 6];
```

```
C = A .* B;
```

Here we have defined the first and second array A and B, and the result C after element-wise multiplication of A and B is [5, 12]. It has been obtained by multiplying the corresponding elements of A and B: [1\*5, 2\*6].

You can also use both the \* and the .\* operator for multiplying simple scalar values. For example, multiplying 5 by 2 using either of the multiplication operators (i.e., 5 \* 2 or 5 .\* 2) will yield the same result, 10.

Similarly, division in MATLAB can be performed using the / operator for matrix division and the ./ operator for element-wise division. Matrix division with / is used to solve systems of linear equations or to find the inverse of a matrix. The ./ operator performs element-wise division, where each element of the first array is divided by the corresponding element of the second array, e.g.,  $D = A ./ B$ . With the variables A and B as defined above, the result of D in this case will be [0.2, 0.3333]. Similar to multiplication, the arrays must be of compatible sizes.

Here, you can also use the / and the ./ operator with scalar values. For example, if you enter 10 / 2 or 10 ./ 2 in the Command Window, the output will be 5.

Division can, for example, be used to calculate ratios or proportions – such as the frequency of a word, or several words, in the 1950s relative to their total occurrences in a corpus. In Chapter 8.1.3, we saw an example from the TIME Magazine Corpus, where the total frequencies for the words “happily”, “cheerfully”, “merrily” and “joyfully” were 2969, 979, 308 and 213, and the frequencies of these words from the 1950s were 762, 208, 67 and 35. To calculate the ratios element-wise, you can define the variables as in Chapter 8.1.3 above and then use the ./ operator to divide the 1950s frequencies by the total frequencies:

```
TIME_total_freq = [2969, 979, 308, 213];
```

```
TIME_freq_1950s = [762, 208, 67, 35];
```

```
ratios = TIME_freq_1950s ./ TIME_total_freq;
```

The result, stored in ratios, will give you the proportions for each word (approximately 0.2567, 0.2125, 0.2175 and 0.1643).

Another practical example of division in linguistic contexts is calculating the type-token ratio (TTR) in a text analysis task. The TTR measures the lexical diversity of a text or a corpus and is calculated by dividing types, i.e., the number of different or unique words, by tokens, i.e., the total number of words. In an example in Chapter 5.2.2, we calculated a TTR of approximately 0.83 for a sample text with 43 types and 52 tokens using the following command:

```
ttr_text1 = types / tokens;
```

Another common task in corpus linguistics is normalising values so that they are comparable across different datasets or corpora. Raw frequency counts alone are often not meaningful unless they are adjusted for the size of the corpus. By dividing the raw counts by the total number of words in the corpus and then multiplying by a constant (such as 1,000 or 1,000,000), we can scale the values to enable meaningful comparisons across corpora of different sizes.

Suppose we have the absolute frequencies of the word “Internet” in two English datasets: the *ACL Anthology Reference Corpus* (ARC, Bird et al. 2008) and the *Directory of Open Access Journals (DOAJ corpora) – English* which are both available at the Sketch Engine website. The ACL (Association for Computational Linguistics) Anthology Reference Corpus is composed of conference and journal papers in natural language processing and computational linguistics and includes more than 18,000 papers from the ACL Anthology (<https://aclanthology.org/>) published between 1979 and 2015. The ARC corpus contains 74,875,938 tokens (including punctuation marks and other non-word elements) and 62,196,334 words. The ARC is openly available on Sketch Engine and can be accessed without an account under <https://www.sketchengine.eu/acl-anthology-reference-corpus-arc/>. The second dataset, the English part of the Directory of Open Access Journals or DOAJ corpora is comprised of English texts from journals across a wide range of academic disciplines (retrieved from <https://doaj.org/>). This corpus contains 3,349,931,737 tokens and 2,662,763,697 words. It also contains metadata such as article titles and keywords (cf. <https://www.sketchengine.eu/doaj-corpora/>). DOAJ – English is available to Sketch Engine trial users (with a 30-day free trial) and subscribers.

To find the frequency of the word “Internet” in these datasets, one option is to use the CQL (Corpus Query Language) interface in the advanced search section of Sketch Engine. With the CQL query `[word="Internet"]`, we obtain a frequency of 2,081 for this word in the ARC. Sketch Engine provides normalised values per million tokens (in this case, 27.79 per million), but for our purposes we want to normalise the frequency per 1,000 words instead. This approach excludes punctuation marks and other non-word elements, which are typically included when using token-based counts.

Using the same CQL query `[word="Internet"]`, we find that the word “Internet” appears 106,051 times in the DOAJ – English corpus. According to Sketch Engine, this corresponds to a frequency of 31.66 occurrences per million tokens.

To normalise such frequencies per 1,000 words in MATLAB, we can first define the absolute frequencies of “Internet” in each corpus, along with the total number of words in these datasets:

```
ARC_freq_Internet = 2081;
DOAJ_freq_Internet = 106051;
ARC_total_words = 62196334;
DOAJ_total_words = 2662763697;
```

After defining the variables, we calculate the normalised frequency by dividing the word’s absolute frequency by the total number of words in the respective corpus. This gives the relative frequency of the word. Multiplying this value by 1,000 then scales it to represent the frequency per 1,000 words, which makes it easier to compare the values across the two datasets that have different sizes.

```
ARC_normalised = (ARC_freq_Internet / ARC_total_words) * 1000;
DOAJ_normalised = (DOAJ_freq_Internet / DOAJ_total_words) * 1000;
```

We obtain a normalised frequency of approximately 0.0335 for `ARC_normalised` and 0.0398 for `DOAJ_normalised`.

It would also be possible to calculate these normalised frequencies for both corpora at once and to store the results in a single variable using an array:

```
normalised_frequencies_ARC_DOAJ = [(2081 / 62196334) * 1000, (106051 /
2662763697) * 1000];
```

In this example, `normalised_frequencies_ARC_DOAJ` is a 1x2 array where the first element corresponds to the normalised frequency of the word “Internet” in the ARC corpus, and the second to that in the DOAJ corpus. In this case, you can then access the individual normalised frequencies as follows:

```
ARC_normalised = normalised_frequencies_ARC_DOAJ(1);
DOAJ_normalised = normalised_frequencies_ARC_DOAJ(2);
```

These examples serve as an introduction and overview of how multiplication and division – particularly element-wise operations – play a key role in handling corpus-based numerical data.

You may also refer to the MathWorks documentation on multiplication under the links given above in this chapter and on the MathWorks documentation on the above-mentioned division operators at: <https://www.mathworks.com/help/matlab/ref/double.rdivide.html> and <https://www.mathworks.com/help/matlab/ref/double.mrdivide.html>.

There is additional information on array operations and matrix operations under: [https://www.mathworks.com/help/matlab/matlab\\_prog/array-vs-matrix-operations.html](https://www.mathworks.com/help/matlab/matlab_prog/array-vs-matrix-operations.html).

### 8.1.5 Logarithmic functions, square root and exponentiation

In corpus linguistics, as well as in many areas of statistical modelling, it is common to encounter frequency distributions that exhibit skewness or non-constant variance. This is particularly true when working with frequencies of words or constructions, where a few occur very frequently (e.g., “the”, “and”, “of”), while many others appear only rarely.

To understand this better, let us briefly talk about skewness. In statistics, when data is described as “skewed”, it means that the distribution is not symmetrical. In a positively skewed (or right-skewed) distribution, the right tail (with the higher values) is much longer than the left tail, pulling the mean to the right of the median. This is the case with word frequency distributions, where a few very common words dominate the frequency count, and the majority of words appear much less frequently. In contrast, a normal distribution is a bell-shaped curve where most of the data points cluster around the mean (average), with fewer data points appearing as you move further from the centre. In a normal distribution, the data is symmetrically distributed, with the left and right sides being mirror images of each other.

To make skewed data more suitable for analysis and improve the performance of statistical models, various non-linear transformations can be applied in MATLAB such as logarithmic

functions, square root functions or exponentiation.<sup>31</sup> These transformations can help to stabilise variance, reduce skewness and normalise the data to make it more appropriate for modelling. By applying a logarithmic transformation to the data, we compress the values and reduce the impact of the extreme frequencies (the so-called outliers).

Therefore, logarithmic transformations are widely used in corpus linguistics for their ability to handle highly skewed data, such as word frequency distributions that follow a Zipfian distribution characterised by a few very frequent words and a long tail of infrequent words, which can cause the data to have a heavy right-skew. By applying a logarithmic transformation, large values are compressed, and small values are expanded so that the distribution becomes more symmetric and closer to a normal distribution. This normalisation helps in stabilising variance and making the data more suitable for statistical techniques that assume normally distributed errors.

Commonly used logarithmic functions include the natural logarithm, the base-10 logarithm and the base-2 logarithm. In MATLAB, you can apply these logarithmic transformations using the following functions:

The function `log(X)` returns the natural logarithm of each element in array `x`.

```
Y = log(X);
```

The function `log10(X)` returns the common logarithm (base 10) of each element in array `x`.

```
Y = log10(X);
```

The function `log2(x)` returns the logarithm to the base 2 of each element in the input array `x`.

```
Y = log2(X);
```

These functions can handle various input types, including scalars, vectors and matrices.

For instance, you can generate a word frequency list from your own texts, obtain frequency data from existing corpora or use ready-made frequency lists available online. You can then apply a logarithmic transformation in MATLAB (e.g., using the `log` function) to normalise the frequency values. This transformation may help to reduce scale skew and to prepare the data for visualisation or further statistical analysis.

If you are working with your own texts, many concordancers offer built-in functions to generate frequency lists. These tools process your text files, count how often each word appears and present the data in an exportable format. If you prefer to work with existing corpora, platforms like Sketch Engine provide access to a range of professionally compiled corpora. By selecting the **Wordlist** function within a chosen corpus available via Sketch Engine, you can generate frequency lists. Note that Sketch Engine normally restricts the downloadable list to a maximum of 1,000 items per query. Access to unlimited wordlists from preloaded corpora for academic purposes is possible upon signing a research agreement and paying an individually calculated fee. For more comprehensive frequency data, you can also turn to various other established resources. Alternatively, you can use publicly available frequency lists. One example is a lemmatised frequency list of more than 6,000 words that occur more than 800 times in the 100-

---

<sup>31</sup> cf. <https://www.mathworks.com/help/matlab/exponents-and-logarithms.html> and <https://www.mathworks.com/help/matlab/math/powers-and-exponentials.html>

million-word British National Corpus (BNC). This list is available at: <https://www.kilgarriff.co.uk/bnc-readme.html>, cf. also Kilgarriff (1997). Comprehensive frequency lists for the BNC are also published in Leech et al. (2001) and are also accessible here: <https://ucrel.lancs.ac.uk/bncfreq/flists.html>.

Let us say you have obtained word frequencies from your own dataset. Below is a simplified, possible example for illustrative purposes, where only the most and least frequent words are shown, with the rest represented by ellipses:

```
1    68267    the
2    43963    a
3    39344    of
4    28786    and
5    23689    be
6    19431    in
...
...
14597    1    footman
14598    1    quirkiily
```

In MATLAB, you can represent these frequencies as a vector and apply a logarithmic transformation as follows:

```
word_frequencies = [68267, 43963, 39344, 28786, 23689, 19431, ..., 1, 1];
log_frequencies = log(word_frequencies);
```

Such a logarithmic transformations would be especially useful when dealing with highly skewed frequency distributions, as they compress the scale significantly and reduce the dominance of extremely frequent items.

Square root transformations, on the other hand, may provide an alternative for normalising corpus data by reducing skewness while preserving relative differences between frequency values. They provide a gentler smoothing effect compared to logarithmic transformations. Square root transformations can be useful when dealing with moderately skewed frequency data, where extreme values are not as pronounced as in highly skewed distributions.

Suppose you are analysing the frequency of different words in a medium-sized corpus. The most frequent words occur around 10,000 to 12,000 times, while many other words appear in the mid-frequency range of 50 to 500 occurrences. This results in a moderately skewed distribution. If you use raw frequencies, the very high-frequency words will dominate visualisations or statistical models, which will make it difficult to distinguish between words in the middle range. Applying a logarithmic transformation in such a case compresses the scale heavily. It may overly compress the mid-range values and reduce meaningful differences. A square root transformation reduces the skewness and variance more gently and preserves larger gaps between mid-frequency values compared to the log transformation. Here is another sample word frequency list (showing only selected high and mid-frequency values):

```
1    12312    the
2    11354    a
3    9362     of
4    7342     and
...
50   452      think
```

```

51     412     time
52     355    brought
53     309     great
54     258     very
55     201    again
...

```

The following commands create a vector of word frequencies and apply the square root transformation using the `sqrt` function.

```

word_frequencies = [12312, 11354, 9362, 7342, ..., 452, 412, 355, 309, 258,
201, ...];

sqrt_frequencies = sqrt(word_frequencies);

```

Exponentiation is commonly used to reverse the effects of logarithmic transformations and restore data to its original scale. For example, when word frequencies are logged to compress the scale of values for analysis, exponentiation can bring these frequencies back to their natural distribution. Exponentiation also plays a crucial role in modelling exponential growth or decay, which is common in various linguistic contexts.

For instance, as a corpus expands, the frequency of certain words, especially high-frequency terms, does not increase in a linear fashion but often follows an exponential or power-law distribution. This means that a small set of words (e.g., function words or common terms) will occur much more frequently than the vast majority of other words, and this relationship is typically represented by an exponential curve.

In the case of vocabulary expansion, exponentiation may also help to model how the number of unique words in a corpus evolves over time. As new documents are added to a corpus, the number of distinct words generally increases. However, the rate at which new words appear decreases as the corpus grows. Initially, many new words will be introduced, but after a certain point, new documents will contain fewer novel words. This vocabulary saturation is likely to follow an exponential decay pattern, where early on, vocabulary expansion is rapid, but as the corpus grows larger, the introduction of new terms slows down, and many new words are simply repetitions or variations of words already encountered.

Alternatively, consider the increase in document production over time. As decades or years pass, more documents are created for specific text types, such as scientific papers or news texts. For example, the number of published texts in the 21st century far exceeds that of the 19th century. This growth in document quantity may follow an exponential pattern, with the rate of production accelerating due to advancements in technology, wider access to publishing platforms and increased international connectivity.

In MATLAB, exponentiation can be performed using either `^` or `.^`. You can use `^` for matrix exponentiation, which is applicable only to square matrices, and you can use `.^` for element-wise exponentiation, which can be applied to arrays or matrices of any size. Both `10 ^ 3` and `10 .^ 3`, for instance, will yield the result 1,000. If you have `x = [1, 2, 3]`, the command `y_1 = x .^ 2` will square each element of `x`, and `y_2 = x .^ 3` will cube each element of `x` so that `y_1` will contain the squared values `[1, 4, 9]`, while `y_2` will contain the cubed values `[1, 8, 27]`.

Additionally, MATLAB provides a variety of tools for modelling exponential relationships, which will be covered only briefly here. You can use the `fit` function, which allows you to

create a fit to data using a specified model type (<https://www.mathworks.com/help/curvefit/fit.html>). The syntax is as follows:

```
fitobject = fit(x, y, fitType)
```

For fitting an exponential model using this function, you can use predefined fit types such as 'exp1' for a single-term exponential model or 'exp2' for a two-term exponential model, e.g., `fitObject = fit(x', y', 'exp1')`.

In the Tools menu of plots that you create with MATLAB, you can find the Basic Fitting tool ([https://www.mathworks.com/help/matlab/data\\_analysis/interactive-fitting.html](https://www.mathworks.com/help/matlab/data_analysis/interactive-fitting.html)). It allows you to fit curves or surfaces to your data interactively. You can choose from various models and visualise how well the model fits your data. cf. also Chapter 7.9 and Figure 152 therein.

Additionally, the Statistics and Machine Learning Toolbox in MATLAB offers a wide range of fitting options for data analysis and modelling, and there is also MATLAB's Curve Fitter app. In order to use the Curve Fitter app, you need to install the Curve Fitting Toolbox (cf. <https://www.mathworks.com/help/curvefit/>). This toolbox provides both an interactive app and a set of functions designed for fitting curves and surfaces to data through various methods, including regression, interpolation and smoothing.

### 8.1.6 Rounding functions

Rounding can be useful when working with linguistic data when you need to simplify numerical representations or deal with practical aspects of data analysis. Frequencies, proportions or statistical results often produce decimals with excessive precision, and rounding makes them easier to interpret and present. In MATLAB, there are different rounding functions which can be used to manipulate numerical data by rounding values to the nearest integers based on different criteria.

The function `round()` rounds each element of an array to the nearest integer. You can use it in a command such as `Y = round(X)`. If an element is exactly halfway between two integers, this function rounds away from zero by default (e.g., `round(1.5)` results in 2). It can also round to a specified number of decimal places using `round(X, N)`, where `X` is the input array and `N` is the number of decimal places to which you want to round.

For example, consider the following vector of decimal values:

```
X = [1.2147, 2.5426, 3.4815];
```

If we want to round these values to two decimal places, we can set `N` as follows:

```
N = 2;
```

The variable `Y` will contain the rounded values `[1.21, 2.54, 3.48]` if we now use the following command:

```
Y = round(X, N);
```

The `floor()` function in a command such as `Y = floor(X)` rounds each element of an array down to the nearest integer less than or equal to that element. This means it always rounds down

to the next lower integer, even for negative numbers. For example, `floor(4.8)` results in 4, and `floor(-3.2)` results in -4.

The `ceil()` function, as in `Y = ceil(X)`, rounds each element of an array up to the nearest integer greater than or equal to that element. This means it always rounds toward the next higher integer, even for negative numbers. For example, `ceil(5.3)` results in 6, and `ceil(-3.9)` results in -3.

These functions can handle various data types, including scalars, vectors, matrices and tables.

### 8.1.7 Sorting data

In corpus linguistics, sorting is often essential and can help organise data in a way that makes patterns and trends easier to see. In MATLAB, there are several ways to sort data. In previous chapters, we have already briefly seen how can sort variables in the Variables editor. Double-click on a variable in the Workspace panel to open the variable in the Variable tab or use the `openvar` function. Then select the variable that you want to sort, right-click on it in the Variables editor and choose the desired sort order (ascending or descending) from the drop-down menu (Figure 179). Alternatively, select **Sort** from the toolstrip under the Variable tab.

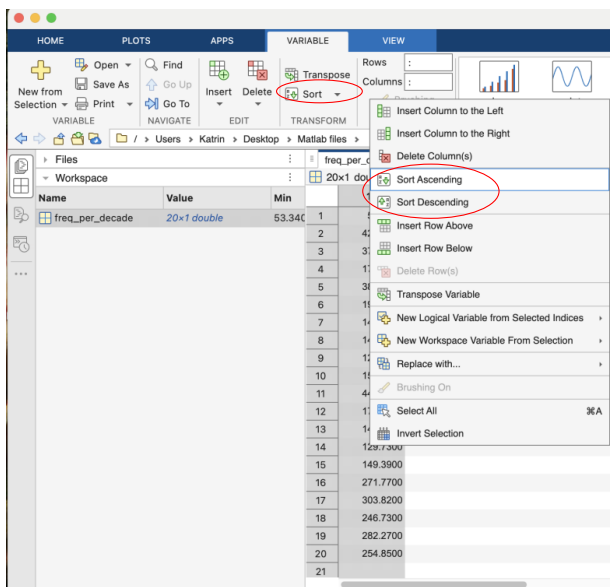


Figure 179: Sorting the values of a variable interactively in the Variables editor

If your data is in a table format, locate the variable in the table in the Variables editor that you want to sort. Hover your mouse over the column header of the column that you want to sort until an arrow appears on the right side. From there, you can choose to sort the values from largest to smallest or from smallest to largest. If the column contains words, you can sort them alphabetically. For example, consider the following table which we create with a small script using words from the Corpus of Contemporary American English (Davies 2008–, COCA, <https://www.english-corpora.org/coca/>) and their respective frequency-per-million values:

```
words = ["spending", "costs", "payments", "expenses"];
frequencies = [86.13, 97.39, 22.60, 19.54];
wordTable = table(words', frequencies', 'VariableNames', {'Word_in_COCA',
'Frequency_per_million'});
```

Figure 180 illustrates how you can sort this table based on the numerical values in the frequency column.

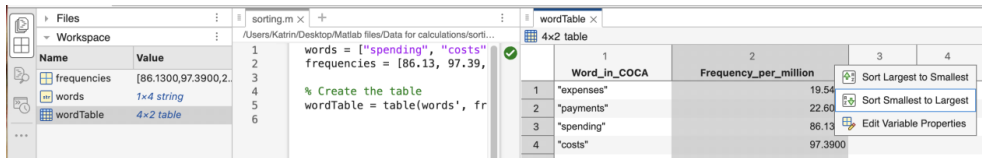


Figure 180: Sorting values from a table in the Variables editor

You can also use the `sort` function to arrange the elements of an array (e.g., a vector or a matrix) in ascending or descending order or alphabetically.<sup>32</sup> To sort the frequencies in the array `frequencies = [86.13, 97.39, 22.60, 19.54]` in ascending order, you can use:

```
sorted_frequencies1 = sort(frequencies);
```

If you want to sort these frequencies in descending order, you can use the following command:

```
sorted_frequencies2 = sort(frequencies, 'descend');
```

To sort a table based on a specific column, you can use the `sortrows` function. The following command sorts the above-mentioned table `wordTable` based on the `Frequency_per_million` column in ascending order:

```
sortedWordTable = sortrows(wordTable, 'Frequency_per_million');
```

If you want to sort this table in descending order based on the numeric values, you can use the `sortrows` function with the `'descend'` option as follows:

```
sortedWordTable1 = sortrows(wordTable, 'Frequency_per_million', 'descend');
```

To sort the table `wordTable` alphabetically by the `Word_in_COCA` column, use:

```
sortedWordTable2 = sortrows(wordTable, 'Word_in_COCA');
```

When you have a table, it also works to use the `sort` function with the `'descend'` option to sort data in your table from Z to A:

```
sortedWordTable3 = sortrows(wordTable, 'Word_in_COCA', 'descend');
```

The table `sortedWordTable3` will now contain the words sorted from Z to A (“spending”, “payments”, “expenses”, “costs”). All other columns remain aligned with their corresponding words, as each row is reordered together.

<sup>32</sup> <https://www.mathworks.com/help/matlab/ref/double.sort.html>

## 8.2 Descriptive statistics – core statistical measures

### 8.2.1 Overview

MATLAB provides built-in functions for computing descriptive statistics, such as mean, median, standard deviation etc. For advanced statistical computations, you can also utilise a wide range of additional functions from the Statistics and Machine Learning Toolbox. The toolbox provides functions and apps for describing, analysing and modelling data. It includes capabilities for descriptive statistics, visualisations, clustering, regression, classification and hypothesis testing.

This section will cover core statistical measures available through MATLAB's built-in functions, to get you started with MATLAB's capabilities for descriptive statistical analysis. It will serve as an overview to equip you with the essential tools to summarise and interpret data and lay the groundwork for more complex analysis should you wish to explore MATLAB's broader statistical functionality. While MATLAB offers an extensive range of functions for advanced statistical modelling and data analysis – especially when combined with the Statistics and Machine Learning Toolbox – it would go beyond the scope of this book to present a comprehensive treatment of all of them. Chapter 9 will provide a brief introduction to using MATLAB for inferential statistics, statistical tests and relationships between variables, without going into extensive detail.

In MATLAB, the Workspace panel provides an easy way to obtain descriptive statistics without writing any code. It displays the variables currently in memory along with their names, values etc. You can also configure the Workspace panel to show basic statistics for each variable. To do this, right-click the header of any column in the Workspace panel and select the statistics you want to display. In Chapter 3.6 with information on the Workspace panel, we have already seen this demonstrated in Figure 24, and for convenience, Figure 181 here illustrates another example.

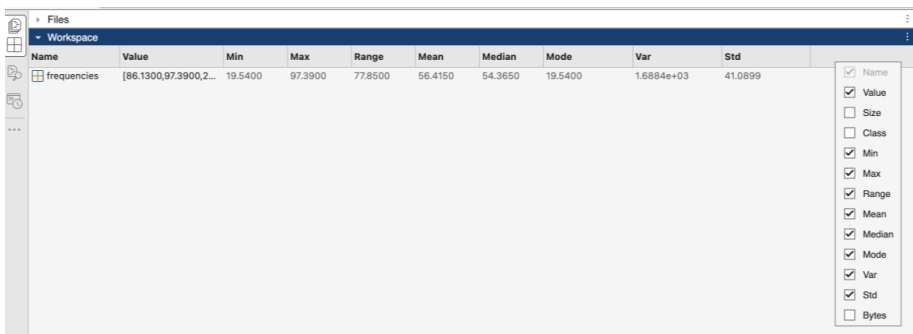


Figure 181: Displaying descriptive statistics in Workspace panel

MATLAB calculates values such as minimum, maximum, range, mean, median, mode, variance and standard deviation using its built-in functions and displays them in the Workspace panel. These statistics update automatically whenever the data in a variable change. If the data contain missing values (NaNs), you can choose in the **Workspace Settings** whether these should be included in or excluded from the calculations by selecting **Ignore NaNs when calculating statistics**. The workspace settings are accessible by selecting **Settings** on the **Home** tab in the **Environment** section (cf. also Figure 25 in Chapter 3.6).

In Chapter 7.7, we have also seen that it is possible to display descriptive statistics for the x- and y-data of plots in MATLAB by selecting the **Tools** tab in the figure window. There you find an icon labelled **Data Statistics**. When you click on it, you can see the Data Statistics dialogue box, which lists various descriptive measures for your dataset. You can choose which of these you want to display directly on the plot by ticking the respective checkboxes. An example of this was shown in Figure 144 in Chapter 7.7, and Figure 182 here illustrates another example.

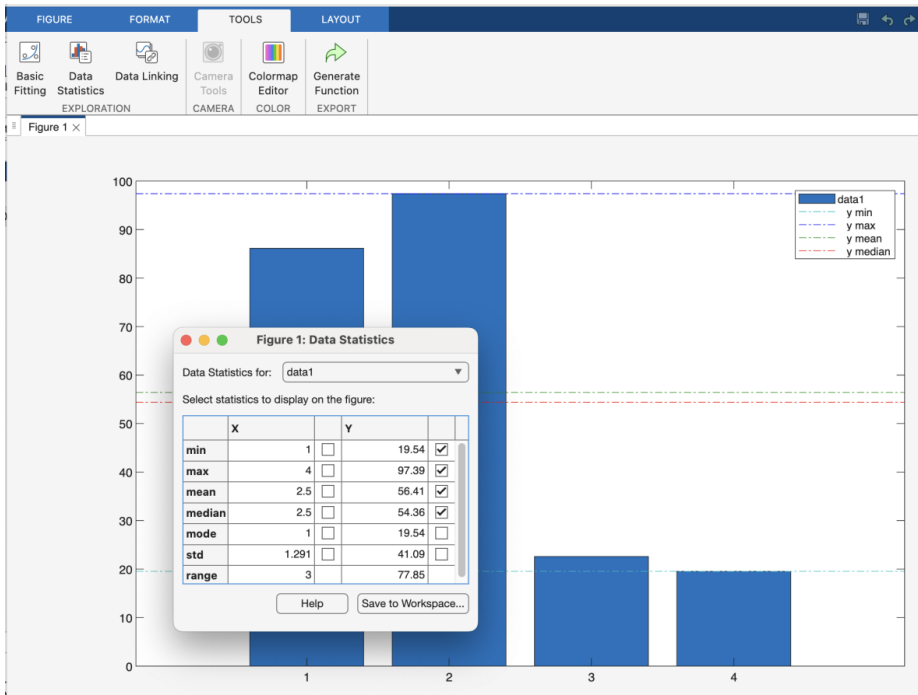


Figure 182: Displaying descriptive statistics in a plot

You can adjust how the statistics appear on the plot to suit your preferences. For example, by double-clicking a statistic line, such as the mean, you can open the submenu for the features of the mean in the Property Inspector. This allows you to change its colour, line style and other visual properties. The displayed statistics are dynamic, so if you change the underlying data in the workspace during your MATLAB session, the values and their visual representation on a plot will update automatically. However, for the plot to reflect these updates, it must remain open. If you close the plot and later modify the data, you will need to recreate the plot to visualise the updated statistics.

Additionally, you can calculate various descriptive statistics such as maximum, mean, median, minimum, mode, standard deviation and variance using commands (cf. also <https://www.mathworks.com/help/matlab/descriptive-statistics.html>). We will cover various examples in the following subsections of Chapter 8.2.

## 8.2.2 Minimum and maximum

Two of the most basic statistics are the minimum and maximum values that indicate the smallest and largest observations within a dataset. Together, they provide a quick sense of the range and boundaries of the data. You can display these values for your variables in MATLAB directly in the Workspace panel (Figure 183) or in plots as outlined in Chapter 8.2. You can also use MATLAB's built-in `min()` and `max()` functions.

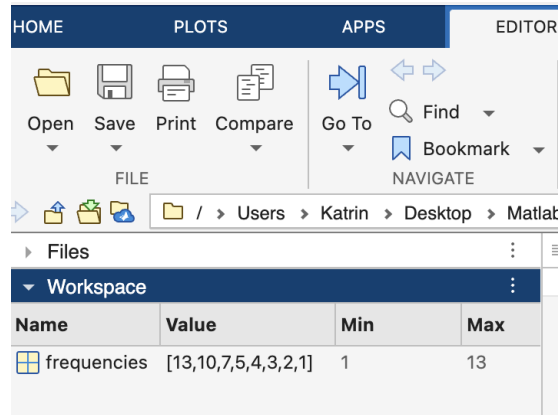


Figure 183: Minimum and maximum values of a variable, shown in Workspace panel

When you apply the `min()` or the `max()` function to a vector in MATLAB, it returns the smallest or the largest element within that vector. To illustrate this, let us consider an example based on corpus data from the CQPweb interface at Lancaster University. Specifically, we will look at the frequencies of some adverbs starting with “un-” and ending in “-ly” in the English version of the Arabian Nights (Richard Burton translation, <https://cqpweb.lancs.ac.uk/burton1001/>). We can perform a CQP query for words of this type, for example by using one of the following queries:

```
[word="[Uu]n.*ly"]
```

or

```
[word="( ?i)un.*ly"]
```

These queries search for any word that starts with `un` (with either `[Uu]` or the case-insensitive flag `( ?i)` applied), followed by zero or more arbitrary characters `(.*)`, and ending in `ly`.

Some examples from the results are *unseemly* (13), *unexpectedly* (10), *unjustly* (7), *unmannerly* (5), *untimely* (4), *unlawfully* (3), *unmercifully* (2) and *unwisely* (1). To display the lowest or the highest frequency value for this selection, we can store these values in a vector `frequencies` and then use the `min()` or `max()` function:

```
frequencies = [13, 10, 7, 5, 4, 3, 2, 1];
```

```
Min_frequencies = min(frequencies) returns the minimum frequency:
```

```
Min_frequencies = 1
```

`Max_frequencies = max(frequencies)` returns the maximum frequency:

```
Max_frequencies = 13
```

If you also want to know the position (or index) of the minimum or maximum value, you can request it as a second output (if the minimum or maximum value occurs multiple times, the functions will return the index of the first occurrence). Use:

```
[Min_frequencies, Min_Index] = min(frequencies)
```

or

```
[Max_frequencies, Max_Index] = max(frequencies)
```

Here, we obtain the minimum or maximum value as well as the index of the respective value (`Min_Index = 8`, `Max_Index = 1`). This tells us that the word with the lowest frequency appears in position 8 of the list and the word with the highest frequency appears in position 1 of the list.

For matrices, `min()` and `max()` return minimum or maximum values for each column by default, and you can specify the dimension to get row-wise values. Suppose we take the frequencies for a small selection of adverbs beginning with the prefix “un-” from this English translation of the Arabian Nights which is available on the University of Lancaster’s CQPweb interface and then add the frequencies for the corresponding adverbs without the prefix “un-“ from the same dataset. We can place these values into a single matrix as follows:

```
% Frequencies of some adverbs with "un-" prefix (unseemly, unjustly, untimely,
unlawfully, unmercifully, unwisely)
un_adv_frequencies = [13; 7; 4; 3; 2; 1];

% Frequencies of corresponding adverbs without "un-" prefix (seemly, justly,
timely, lawfully, mercifully, wisely)
adv_frequencies = [11; 16; 2; 13; 1; 5];

% Combine into a matrix
adverb_frequencies = [un_adv_frequencies, adv_frequencies]
```

This will return

```
adverb_frequencies =

    13    11
     7    16
     4     2
     3    13
     2     1
     1     5
```

To display the minimum or maximum values for each column we can use:

```
min_column_values = min(adverb_frequencies)
```

or

```
max_column_values = max(adverb_frequencies)
```

By default, the respective function will return, the minimum or maximum for each column (1 and 1 or 13 and 16 in this case).

To find the minimum or maximum values for each row, use:

```
min_row_values = min(adverb_frequencies, [], 2)
```

or

```
max_row_values = max(adverb_frequencies, [], 2)
```

Here, the respective function call computes the minimum or maximum values for each row by specifying the dimension as 2. The empty brackets [] before 2 indicate that there is no second array to compare against. If you do not include these brackets in this case, MATLAB will attempt to compare each element of the matrix with the scalar value 2.

### 8.2.3 Measures of central tendency

When you analyse linguistic data, you may want to find typical or central values. Very common measures of central tendency are the mean (the average value) and the median (the middle value when the data is sorted). The mean gives a quick overall average, while the median is less affected by extremes and may be more useful when your data has outliers. There is also the mode, i.e., the most frequently occurring value. Like the minimum and maximum, you can display these values for your variables in the Workspace panel or in plots, as outlined in Chapter 8.2.

You can also use MATLAB's built-in functions such as `mean()`, `median()` and `mode()` to perform calculations on arrays.

Let us start with the mean to illustrate how these functions work. For a simple vector `x`, you can compute the mean as follows:

```
X = [2, 3, 5, 7, 11, 13];
```

```
Mean_X = mean(X)
```

In this case, the output of `mean(X)` will be 6.8333.

When dealing with a matrix, you can specify the dimension along which to compute the mean. For example, consider the following matrix:

```
Y = [1, 2, 4; 4, 6, 10];
```

The matrix `Y` consists of two rows and three columns:

```
Y =
```

```

     1     2     4
     4     6    10
```

By default, `mean()` computes the mean of each column in a matrix. The following command will therefore calculate the mean of each column in `Y`.

```
column_Mean = mean(Y)
```

The output will be 2.5, 4.0 and 7.0 (mean of [1; 4], [2; 6] and [4; 10]). Alternatively, you can specify the dimension explicitly:

```
column_Mean = mean(Y, 1)
```

This command explicitly specifies that the mean should be calculated along the first dimension which corresponds to the columns in a 2D matrix. The result will be the same as the previous command.

To calculate the mean of each row use:

```
row_Mean = mean(Y, 2)
```

The output will be 2.3333 and 6.6667 (mean of [1, 2, 4] and [4, 6, 10]).

Let us now move on to the median. To calculate the median of a simple vector  $x$  (e.g.,  $x = [2, 3, 5, 7, 11, 13]$ ), you can use:

```
Median_X = median(X)
```

The output in this case will be 6 as the dataset has an even number of elements, and therefore the median is calculated as the average of the two middle numbers.

To calculate the median of a matrix in MATLAB, you can use the `median()` function as well. Like the `mean()` function that we saw above, it treats the columns of the matrix as vectors by default. For the matrix  $Y = [1, 2, 4; 4, 6, 10]$ , you can compute the median of each column as follows:

```
column_Median = median(Y)
```

In this specific case, the mean and median values for each column of the matrix  $Y$  are the same, and the output here will also be 2.5, 4.0 and 7.0. Alternatively, use `column_Median = median(Y, 1)` to compute the median across the first dimension (columns), which is the same as simply using `median(Y)`.

To compute the median of each row, use:

```
row_Median = median(Y, 2);
```

The output will be 2 and 6.

Let us now come to the mode. To compute the mode, you can use the `mode()` function, which returns the most frequently occurring value in an array. If there are multiple values with the same highest frequency, `mode()` will return the smallest of those values.

For  $x = [2, 3, 5, 7, 11, 13]$ , the command `Mode_X = mode(X)` will therefore return 2 since each number appears only once and 2 is the smallest value among them.

For  $Y = [1, 2, 4; 4, 6, 10]$ , the command `column_Mode = mode(Y)` and `column_Mode = mode(Y)` will return 1, 2 and 4, while `row_Mode = mode(Y, 2)` will return 1 and 4.

Let us now consider an example drawn from actual corpus data to illustrate how to use the functions for calculating measures of central tendency in MATLAB. We will use selected data from the Royal Society Corpus, which has also been used for some of the plot examples in previous chapters. At <https://corpora.clarin-d.uni-saarland.de/cqpweb/>, we can use the version RSC 6.0.1 Open to retrieve, for instance, the lengths of all texts from the year 1910.

A CQP query such as `[]::match.text_year="1910"` reveals that there 697,546 tokens in 165 different texts from that year. We can now use the **Tabulate** option in the CQPweb interface to obtain the text IDs together with the corresponding number of tokens per text. When retrieving data from CQPweb in this way, it is important to specify here that the lines should be sorted and duplicates removed, so that data for each text is retrieved only once rather than for every token matched by the query. The resulting file will contain the text IDs and their respective lengths, which can then be copied and pasted directly into a new, empty numeric variable in MATLAB or imported from our file. For convenience, the values are also provided below so that you can directly copy the command to create the array in MATLAB:

```
text_lengths_1910_RSC = [6838; 2593; 2229; 4869; 483; 9153; 8663; 2396; 4024;
1875; 5389; 2762; 2105; 2838; 4226; 649; 1348; 2151; 2343; 2704; 4089; 4642;
4655; 2085; 2680; 2993; 3187; 1540; 8150; 3003; 3599; 2816; 2674; 4846; 3714;
3132; 1979; 1756; 12491; 4695; 975; 6112; 4294; 2940; 1287; 4191; 249; 838;
3953; 5099; 6691; 3388; 3050; 3844; 4084; 3319; 3929; 2493; 5556; 1761; 2557;
1245; 4353; 3711; 3001; 434; 1072; 2080; 6313; 6798; 9387; 13325; 7082; 7189;
10082; 1086; 2311; 1253; 3974; 1317; 1624; 3555; 9178; 1050; 2977; 8610;
4393; 2254; 173; 4833; 14432; 5077; 4705; 4144; 2029; 1609; 8998; 8080; 8918;
4237; 3471; 3342; 8451; 3644; 8273; 7620; 3471; 2892; 4715; 4280; 2584; 4460;
1418; 2810; 1250; 10058; 8381; 168; 871; 3267; 1410; 4117; 2612; 1722; 2872;
14910; 3690; 4984; 5510; 2304; 2630; 4171; 1695; 6248; 5688; 4021; 639; 6223;
4511; 2597; 6014; 7542; 11799; 832; 5326; 1286; 4526; 4134; 1698; 443; 4892;
4794; 3009; 2128; 5047; 20460; 1360; 13597; 5013; 2816; 4345; 3568; 1277;
7132; 665];
```

We can now see the values for the mean, median and mode already available in the Workspace panel (Figure 184). For the mean, we observe the value  $4.2276e+03$ , which is equivalent to  $4.2276 \times 10^3 = 4,227.6$ . The median is 3568, and the mode is 2816, as this value appears twice in the dataset. If desired, we can also create individual variables for these values using the following commands:

```
Mean_text_lengths_1910_RSC=mean(text_lengths_1910_RSC);
Median_text_lengths_1910_RSC=median(text_lengths_1910_RSC);
Mode_text_lengths_1910_RSC=mode(text_lengths_1910_RSC);
```

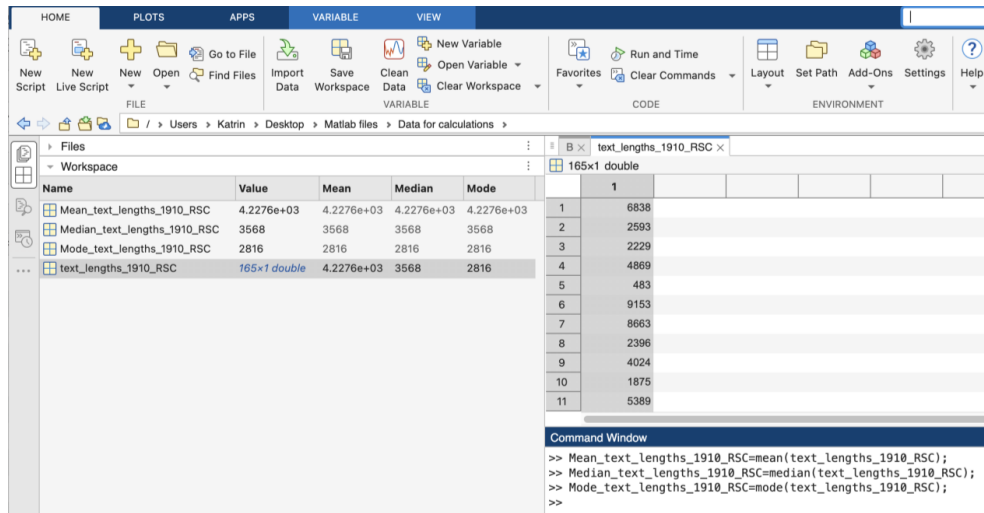


Figure 184: Mean, median and mode of RSC text lengths for year 1910, automatically calculated by MATLAB in the workspace and saved to individual variables via commands

## 8.2.4 Measures of dispersion

Measures of dispersion include the range, variance and standard deviation. These are statistical values that help us understand the spread or variability within a dataset. They indicate how much the individual values differ from one another and from the overall average. The range is the simplest of these measures and represents the difference between the largest and smallest values in a dataset. The variance measures how much the values tend to deviate from the mean by calculating the average of the squared differences from the mean. The standard deviation is the square root of the variance and shows, on average, how much the values differ from the mean, expressed in the same units as the data.

These values, like the measures of central tendency and the minimum and maximum values, can be displayed in the Workspace panel for our variables in MATLAB if we select the option to show them. We can also use specific commands to calculate them:

range: `range()`

variance: `var()`

standard deviation: `std()`

Figure 185 shows an example of the range, variance and standard deviation calculated for the lengths of all texts from the year 1910 in the Royal Society Corpus, which were also used in Chapter 8.2.3. The range in this dataset is 20,292, indicating the difference between the longest and shortest texts. The variance is displayed as  $1.0112e+07$ , which is equivalent to  $1.0112 \times 10^7 = 10,112,000$ , and measures the average squared deviation of the text lengths from the mean. The standard deviation here is  $3.1799e+03$  (i.e., 3,179.9), meaning that, on average, the text lengths deviate from the mean by approximately 3,180 tokens.

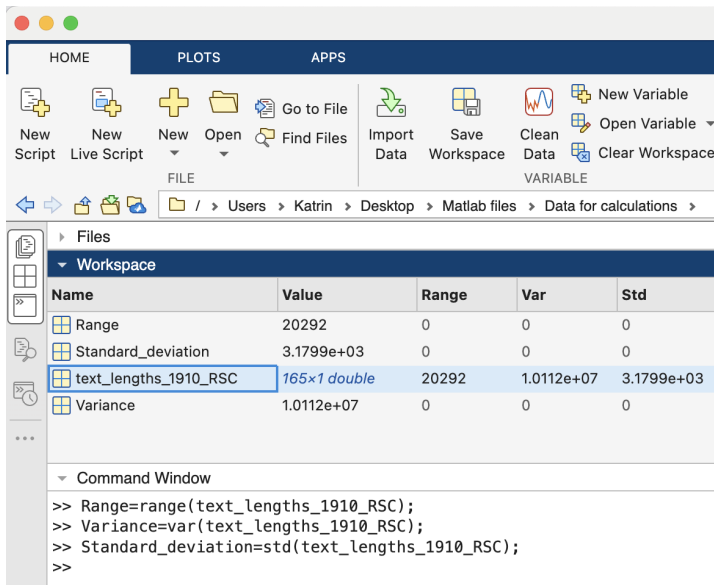


Figure 185: Range, variance and standard deviation of RSC text lengths for year 1910, automatically calculated by MATLAB and saved to individual variables via commands

## 8.2.5 Data summary

You can use the `summary` function for obtaining a snapshot of your data. This function works with different types of data. When you use the command `summary(X)`, where `X` is your dataset array, you will see a summary of the dataset and its variables in the Command Window. This summary includes key statistics and counts based on the types of variables in the dataset. If you use `s = summary(X)` returns a scalar structure `s` that contains information about the dataset and its variables. This summary will then also be available in your workspace. For numerical variables, for instance, the summary function provides the count of missing values, the minimum value, the median, the maximum value, the mean and the standard deviation (Figure 186, cf. also <https://www.mathworks.com/help/matlab/ref/double.summary.html>).

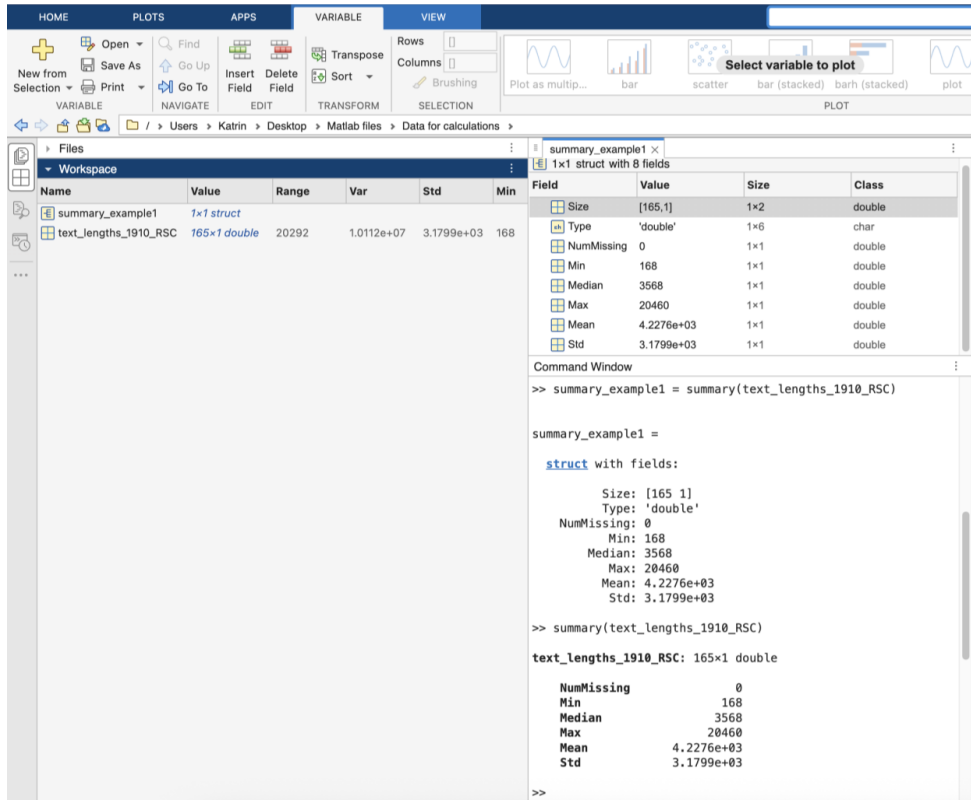


Figure 186: Output of `summary` function for a numerical variable with key statistical values

When applied to a matrix, the `summary` function provides key statistics for each column (or row). This allows for a more detailed understanding of the data distribution compared to the workspace, which typically shows only single values for variables. You can specify whether to summarise by rows or columns using the `dim` argument in the `summary` function. For example, consider the following matrix:

```
frequencies = [693, 613, 435, 473, 354;
              216, 413, 345, 645, 873];
```

To summarise the data for each column, you would use:

```
summary(frequencies)
```

or

```
summary(frequencies, 1)
```

If you want to summarise the data for each row, you would use:

```
summary(frequencies, 2);
```

Figure 187 shows the output of `summary` function, which was used to summarise the data for each column and row of the `frequencies` data as described above, with the results stored in the variables `summary_frequencies` and `summary_frequencies_rows`.

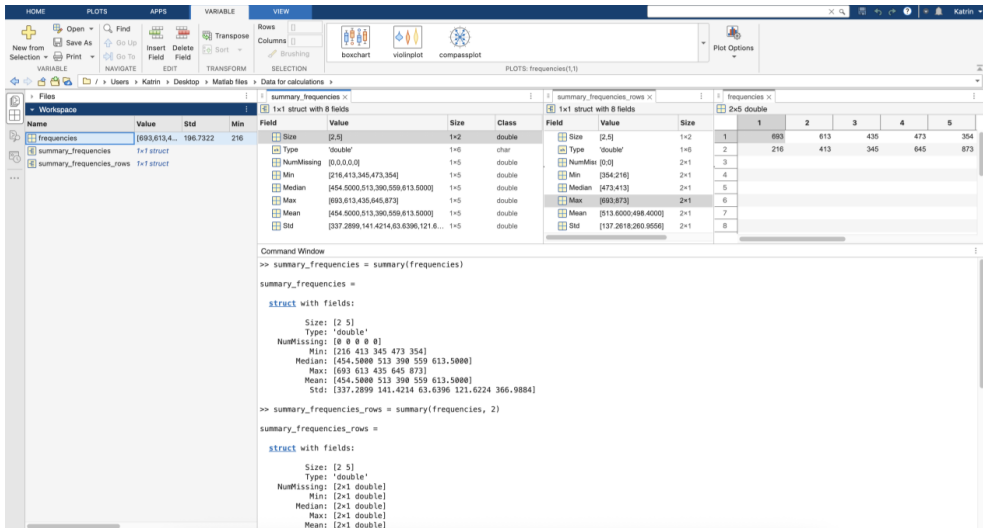


Figure 187: Output of `summary` function for rows and columns of a matrix

The `summary` function can also be used with categorical arrays to provide an overview of the data contained within them (cf. [https://www.mathworks.com/help/matlab/matlab\\_prog/plot-categorical-data.html](https://www.mathworks.com/help/matlab/matlab_prog/plot-categorical-data.html)).

When you create a categorical array, such as the one representing OCR quality categories of texts as we have seen in Chapter 5.2.5, the `summary` function will give you a count of occurrences for each OCR quality category in the dataset. First, we define the categories and counts as string and numeric variables, then we create a categorical array (`OCR_quality`) from these data and use the command `summary(OCR_quality)`, cf. Figure 188.

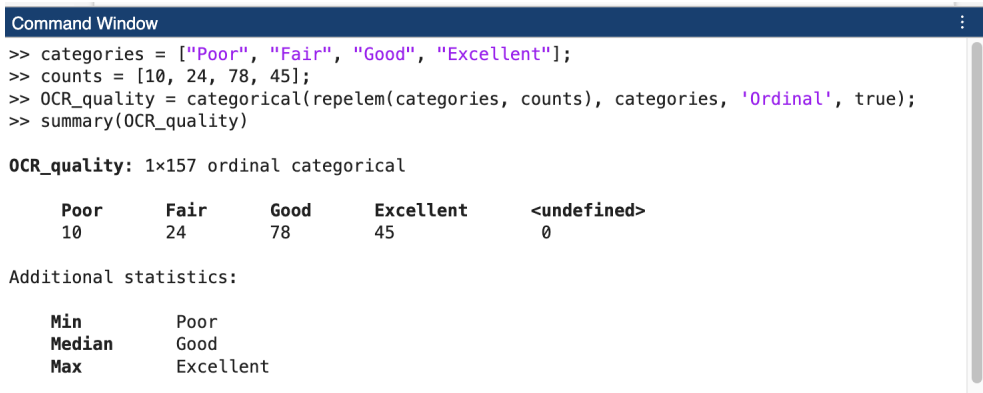


Figure 188: Output of `summary` function for categorical array

This will display the counts of each category defined in the array. The output also includes a count of undefined elements (0 in this case). The inclusion of undefined elements would be particularly relevant when some data points do not match any of the specified categories in the array. Note that in the example shown in Figure 188, we have created our categorical array with the 'Ordinal' option set to `true`, which allows MATLAB to recognise the order of the categories in the array. Here, the summary function then provides additional statistics, e.g., a “minimum” value and a “maximum” value for the categorical array. “Poor” in this case is the minimum category and “Excellent” is the maximum category, indicating that “Poor” has the lowest rank and “Excellent” has the highest rank in the scale.



# 9 Exploring inferential statistics, statistical tests and relationships between variables with MATLAB<sup>®</sup>

## 9.1 Overview

This chapter offers an overview of some key concepts in inferential statistics, feature relationships and statistical tests, with a focus on their practical application to linguistic data analysis using MATLAB. Although various relevant functions will be introduced, very detailed examples will not be provided for each due to the introductory nature of this book.

Inferential statistics involves drawing conclusions about a population based on data collected from a sample. In the context of linguistic data, the concept of a population can be adapted to refer to a broad group of language data or speakers from which a sample is drawn. Various statistical tests can be conducted in MATLAB to draw conclusions from linguistic datasets. The Statistics and Machine Learning Toolbox offers key functions including, for instance, t-tests, ANOVA and chi-squared tests. These tests serve multiple purposes, such as comparing group means, e.g., to assess whether the frequency of a particular linguistic feature differs across various subgroups within a dataset. They can also be used to explore relationships between categorical variables, such as investigating the association between language use and sociolinguistic factors like gender or age.

MATLAB and its Statistics and Machine Learning Toolbox includes a broad range of functions and methods that can be applied to linguistic datasets for various tasks. For more in-depth guidance on statistical methods for corpus exploitation, readers are encouraged to consult additional literature such as Baroni & Evert (2009), Eddington (2016) and Woodrow (2014). Further understanding of how to use MATLAB for various statistical calculations can be gained by referring to the relevant MATLAB documentation, particularly the Statistics and Machine Learning Toolbox documentation.

You can, for instance, explore MATLAB resources on statistical calculations and data analysis by visiting the MathWorks documentation sites on the following topics:

*Statistics and Machine Learning Toolbox documentation:*

<https://www.mathworks.com/help/stats/>

*Working with probability distributions:*

<https://www.mathworks.com/help/stats/probability-distributions-1.html> and <https://www.mathworks.com/help/stats/working-with-probability-distributions.html>

*Available hypothesis tests:*

<https://www.mathworks.com/help/stats/hypothesis-tests-1.html> and <https://www.mathworks.com/help/stats/available-hypothesis-tests.html>

*Analysis of variance and covariance:*

<https://www.mathworks.com/help/stats/analysis-of-variance-and-covariance.html>

*Regression models:*

<https://www.mathworks.com/help/stats/regression-and-anova.htm#reference>

*Complete reference list of functions and features available in the Statistics and Machine Learning Toolbox:*

<https://www.mathworks.com/help/stats/referencelist.html>

## 9.2 Working with probability distributions

To work with probability distributions in MATLAB for linguistic applications or in any other field, it is highly recommended to have MATLAB's Statistics and Machine Learning Toolbox installed as it offers a variety of relevant tools and functions.

Many statistical methods rely on the assumption that variables are normally distributed within groups and that model residuals also follow a normal distribution. This assumption forms the foundation for many standard procedures, such as parametric statistical tests, that allow researchers to make inferences about a population based on sample data. In nature, many variables, such as heights and weights, tend to follow a normal distribution. However, linguistic data do not always conform to this pattern. In fact, many linguistic variables exhibit skewed distributions and are not symmetrically distributed like the bell curve of a normal distribution. For example, consider word frequency in a language: a small number of words appear very frequently, while the majority of words occur very rarely. This creates a long-tailed distribution rather than a normal one. While linguistic data may not always follow a normal distribution, the normal distribution can still serve as a useful approximation in certain cases, particularly when large sample sizes are involved. The normal distribution is defined by its mean ( $\mu$ ) and standard deviation ( $\sigma$ ), and its widespread use is partly due to the Central Limit Theorem. This theorem states that, as the sample size increases, the sampling distribution of the mean of independent observations tends to follow a normal distribution, regardless of the underlying population distribution. However, other distributions may more accurately capture the characteristics of specific linguistic data. Therefore, it is crucial to understand the distribution properties of your data before applying statistical methods that assume normality.

If you have a set of sample data, such as word frequencies or other quantitative linguistic measures from your corpus, you can fit a distribution to it using the `fitdist` function, which estimates the parameters of the distribution via maximum likelihood estimation (<https://www.mathworks.com/help/stats/fitdist.html>). For example, to fit a normal distribution to your data, you can use the following command:

```
distribution = fitdist(x, 'Normal')
```

This creates a probability distribution object that represents the fitted normal distribution, where 'Normal' specifies the distribution type and the fit is based on the data in the column vector `x`.

Once you have created the distribution object, you can use the `random` function to generate data that follows the characteristics of the fitted distribution:

```
random_Numbers = random(distribution, 100, 1);
```

In this command 100 specifies the number of random samples to generate. MATLAB will draw 100 values from the fitted distribution. 1 specifies the number of columns in the output. Here, it means the result will be a column vector with 100 rows.

MATLAB also provides interactive tools like the **Distribution Fitter app**, which allows you to fit probability distributions visually without needing to write code (<https://www.mathworks.com/help/releases/R2025a/stats/model-data-using-the-distribution-fitting-tool.html>). You can access this app from the MATLAB toolstrip by clicking on the **Apps** tab and selecting **Distribution Fitter** under the **Math, Statistics and Optimization** section (Figure 189). Alternatively, you can open it by typing `distributionFitter` in the Command Window.

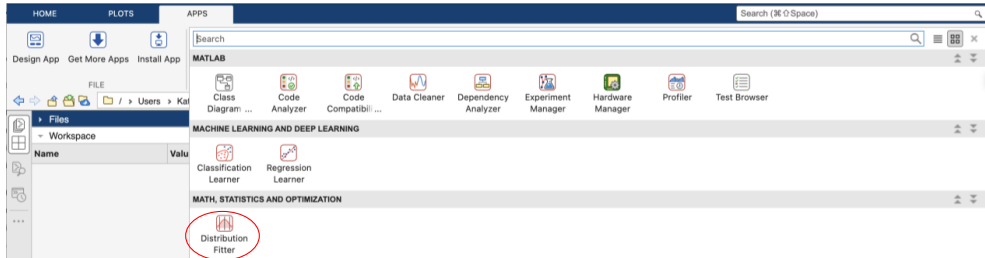


Figure 189: Using the Distribution Fitter app

This app allows you to fit various distributions to your data, visualise the fits and assess how well they capture the underlying characteristics of your data. When you have opened the app, you can import your data by clicking the **Data** button to open the Data dialog box. In this dialog, you can select a numeric array from your MATLAB workspace. For example, if you have a variable named `text_lengths_1910_RSC` (cf. Chapter 8.2.3), select it to visualise its histogram in the app. To fit a distribution to your data, click on **New Fit** in the app. From there, select your dataset and choose a distribution type from the dropdown menu. You can start with common distributions such as “Normal”, “Exponential” or others available in the app. Choosing, for instance, a normal distribution for your `text_lengths_1910_RSC` data might be reasonable if the data appears to be symmetrically distributed around a central value. After selecting the distribution, click **Apply**. The app will then display the fitted distribution along with its parameters. In this case, you will obtain a histogram of your data that illustrates the frequency of the text lengths. Overlaid on this histogram, you will see the fitted normal distribution curve (Figure 190). You can assess how well the chosen distribution fits your data by examining the plot. If the histogram displays a bell-shaped curve and is symmetric, fitting a distribution is appropriate. However, if the data is skewed or contains outliers, you may want to consider alternative distributions that better capture the characteristics of your data.

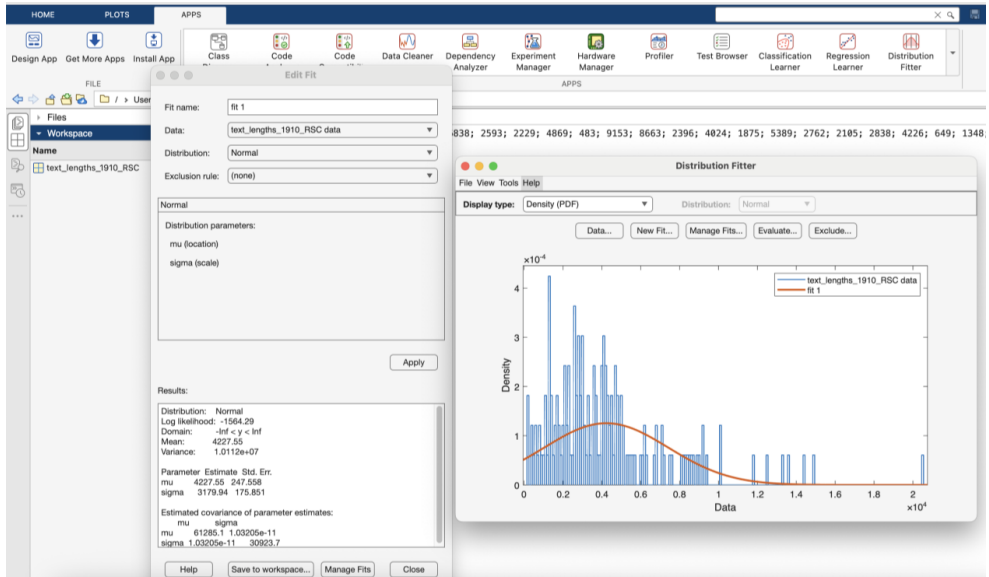


Figure 190: Histogram of text lengths with fitted distribution curve in the Distribution Fitter app

You may also use this app to generate a probability plot to assess how well a distribution such as the normal distribution fits your data. Import your data from the workspace and select **Probability Plot** as display type (Figure 191). If the points in the plot closely follow a straight line, it indicates that the normal distribution is a good fit.

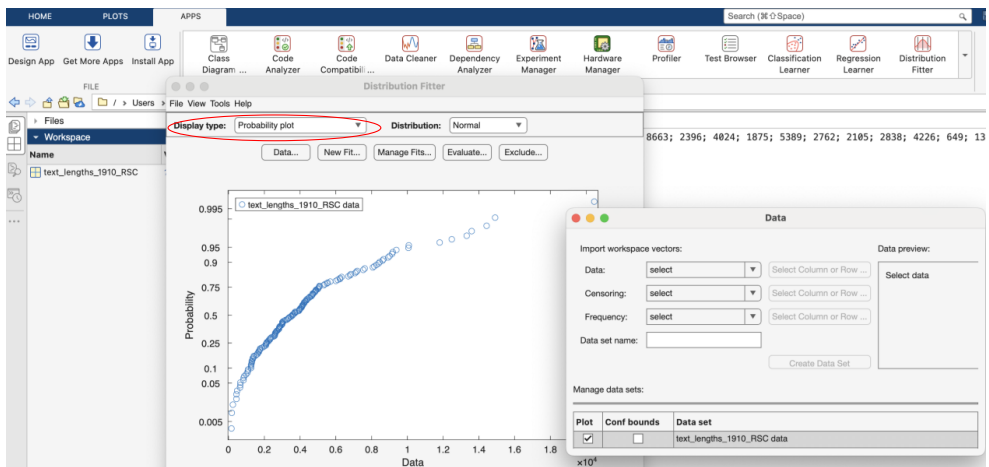


Figure 191: Creating a probability plot in the Distribution Fitter app

If you work with statistical data, MATLAB's **Probability Distribution Function Tool** is also worth exploring (cf. also the MathWorks documentation at <https://www.mathworks.com/help/stats/probabilitydistributionfunctiontool.html>). This tool is also part of the Statistics and Machine Learning Toolbox. It supports approximately 25 different probability distributions, including Normal, Exponential and Poisson, among others. To open the Probability Distribution

Function Tool, use the command `disttool`. This will launch the interface, where you can start exploring various distributions and their parameters (Figure 192).

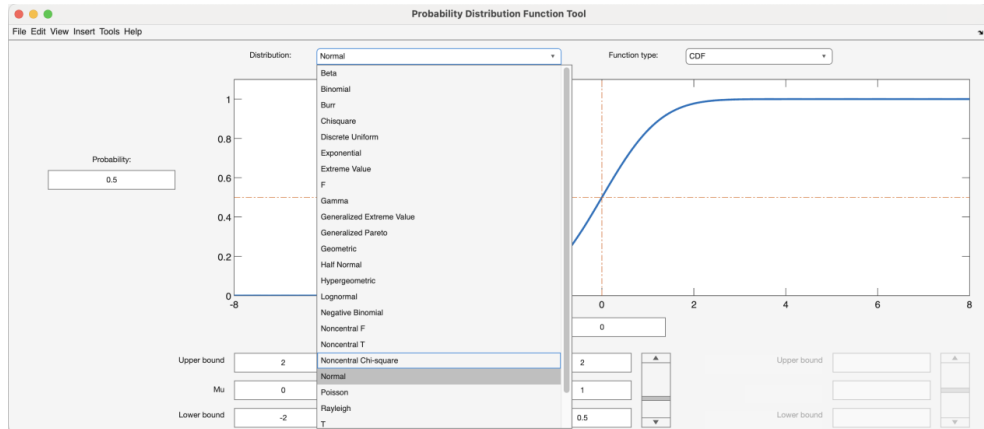


Figure 192: Exploring MATLAB's Probability Distribution Function tool

For further guidance on working with probability distributions, refer to the MATLAB documentation covering this topic and its related subtopics, for instance at:

*Probability distributions and hypothesis tests*

<https://www.mathworks.com/help/stats/probability-distributions-1.html>

*Working with probability distributions*

<https://www.mathworks.com/help/stats/working-with-probability-distributions.html>

*Discrete distributions*

<https://www.mathworks.com/help/stats/discrete-distributions.html>

*Continuous distributions*

<https://www.mathworks.com/help/stats/continuous-distributions.html>

*Exploration and visualisation*

<https://www.mathworks.com/help/stats/exploration-and-visualization.html>

## 9.3 Hypothesis tests

In MATLAB, particularly within the Statistics and Machine Learning Toolbox, a variety of hypothesis tests are available to analyse data and draw conclusions about populations based on sample data. Statistical tests play a crucial role for the analysis of experimental or observational data. They help determine whether observed differences or relationships are statistically significant or if they could have occurred by chance. For instance, tests like t-tests are commonly used to compare means across different groups and assess whether the observed differences are statistically significant. Additionally, statistical tests can be employed to assess how well your data fits a particular distribution, using tests such as the Chi-square goodness-of-fit test or the Kolmogorov-Smirnov test. These tests help determine if the sample data aligns with a specified theoretical distribution.

Understanding hypothesis testing in MATLAB is made easier through several helpful resources from MATLAB's official documentation that provides an overview of the fundamental relevant concepts, such as null and alternative hypotheses, test statistics and p-values, and the assumptions underlying different tests. Have a look, for instance, at:

*Hypothesis tests*

<https://www.mathworks.com/help/stats/hypothesis-tests-1.html>

*Available hypothesis tests*

<https://www.mathworks.com/help/stats/available-hypothesis-tests.html>

*Hypothesis test terminology*

<https://www.mathworks.com/help/stats/hypothesis-test-terminology.html>

*Hypothesis test assumptions*

<https://www.mathworks.com/help/stats/hypothesis-test-assumptions.html>

Let us take the t-test as an example. The `ttest` function in MATLAB is utilised for conducting hypothesis tests related to the means of sample data. This function assumes that the input data follows a normal distribution, or at least that the sample size is large enough so that the sampling distribution of the mean will be approximately normal. It compares the sample mean to the hypothesised population mean and it can also return a p-value, which indicates whether the observed difference is statistically significant.

The simplest form of the syntax is `h = ttest(x)`. This performs a one-sample t-test where the null hypothesis is that the data in `x` comes from a normal distribution with a mean of zero and unknown variance. The alternative hypothesis is that the mean of the population is not equal to zero. The output `h` is 1 if the test rejects the null hypothesis at the 5% significance level, and 0 if it does not reject the null hypothesis.

`h = ttest(x, y)` is used to perform a paired-sample t-test to test the null hypothesis for the difference between the two sample data sets `x` and `y`. Here, `x` and `y` are the sample data vectors that are paired observations. This test evaluates whether the means of the two related groups are significantly different from each other. The function assumes that the differences between the paired samples are normally distributed.

In a one-sample t-test, `y` can be a specified mean value `m` as in the following command:

```
h = ttest(x, m)
```

Therefore, if you were to use `h = ttest(x, m)`, where `m` is a specified mean value, the function would perform a one-sample t-test to assess the null hypothesis that the mean of the sample data `x` is equal to the specified value `m`. The alternative hypothesis posits that the mean is not equal to `m`.

To return not only the test decision `h` but also the p-value `p` of the test, use, for instance:

```
[h, p] = ttest(x, y)
```

or

```
[h, p] = ttest(x, m)
```

The p-value provides the probability of observing the data, or something more extreme, if the null hypothesis is true. A small p-value (typically less than 0.05) suggests that the null hypothesis is unlikely. To change the significance level in the `ttest` function in MATLAB, you can use the 'Alpha' name-value pair argument. Here is how you would set the significance level to 0.01:

```
[h, p] = ttest(x, y, 'Alpha', 0.01)
```

or

```
[h, p] = ttest(x, m, 'Alpha', 0.01)
```

The first of these commands conducts a paired-sample t-test and evaluates the null hypothesis that the mean difference between the paired samples `x` and `y` is zero. As the significance level is set to 0.01, the test will reject the null hypothesis if the p-value is less than 0.01. The output `h` indicates whether the null hypothesis is rejected (1) or not rejected (0), while `p` provides the p-value of the test.

The second command `[h, p] = ttest(x, m, 'Alpha', 0.01)` performs a one-sample t-test as `m` is a specified mean value against which the mean of the sample data `x` is tested. The null hypothesis states that the mean of `x` is equal to `m`. Similar to the paired-sample test, the significance level is set to 0.01 here so that the test will reject the null hypothesis if the p-value is less than 0.01. The output `h` indicates the test decision, and `p` gives the p-value which quantifies the evidence against the null hypothesis.

For a more comprehensive output, including also the confidence interval and additional statistics, you can use, for instance:

```
[h, p, ci, stats] = ttest(x, y),
```

or

```
[h, p, ci, stats] = ttest(x, m)
```

The output for `ci` provides the confidence interval for the mean of the sample `x` (or the mean difference for paired samples). It gives a range of values within which the true mean is likely to fall, with a specified level of confidence (e.g., 99% if 'Alpha' is set to 0.01).

The output for `stats` is a structure that contains additional statistics related to the test, including the t-statistic (`tstat`), which quantifies the difference between the sample mean and the hypothesised population mean (or the mean of another sample in the case of a two-sample test), scaled by the standard error of the sample mean (<https://www.mathworks.com/help/stats/tstat.html>). In addition, the output for `stats` contains the field `df` – the degrees of freedom associated with the test – and `sd` – the standard deviation of the sample data. Degrees of freedom represent the number of independent values or observations that can vary freely in the analysis. A higher `df` generally indicates a more reliable estimate, as it reflects a larger sample size and more information about the population. This value plays an important role in shaping the t-distribution used to determine the significance of the test statistic. As the number of degrees of freedom increases, the t-distribution approaches the normal distribution, resulting in a more accurate evaluation of statistical significance. The standard deviation provides a measure of

how much individual data points deviate from the sample mean. A larger standard deviation suggests greater variability in the data, meaning that the values are more spread out. In contrast, a smaller standard deviation indicates that the data points are closer to the mean, reflecting less variability. Together, the degrees of freedom and standard deviation help assess both the reliability of the test and the characteristics of the sample data.

For more detailed information about the syntax and options available with `ttest` as well as the test statistic you can refer to the MATLAB documentation at <https://www.mathworks.com/help/stats/ttest.html>.

In the context of linguistic data analysis, the `ttest` function can, for instance, be applied to assess whether the mean of a linguistic feature significantly deviates from a hypothesised value. For example, if prior research suggests that the average word length in a specific text type is 4 characters, the `ttest` function can be employed to test this hypothesis against your sample data. Consider the following (small) sample dataset representing word lengths, where the hypothesised mean is the average word length you want to test against:

```
wordLengths = [3, 5, 4, 6, 2, 4, 5, 3];  
  
hypothesizedMean = 4;
```

In a one-sample t-test, the null hypothesis typically asserts that the mean of your sample is equal to a specified value. For example, if the hypothesised mean is 4, the hypotheses would be stated as follows:

Null Hypothesis (H<sub>0</sub>): The sample mean is equal to 4.

Alternative Hypothesis (H<sub>1</sub>): The sample mean is not equal to 4.

The test then evaluates whether the observed sample mean differs significantly from the hypothesised value. Note that, in practice, word lengths in a dataset are likely not normally distributed and may follow a skewed distribution, with many short words and a few longer ones. Applying a t-test to data that may not meet the normality assumption could lead to unreliable results. However, if the sample size is sufficiently large, the distribution of the sample mean will tend towards normality, which could make the t-test a reasonable approximation. The example presented here is meant for illustrative purposes, and is important to ensure that your own data either follows a normal distribution or that you have a large enough sample size to justify using the t-test. Alternatively, testing for normality of your data beforehand is a good practice. One way to do this is through visual inspection, such as creating a histogram (cf. Chapter 7.11) which allows you to assess whether the data roughly follows a normal distribution. There is also the possibility of creating a Q-Q (quantile-quantile) plot for assessing normality. To create such a plot, use the `qqplot` function as in `qqplot(data)`; (<https://www.mathworks.com/help/stats/qqplot.html>). It compares the quantiles of your data against the quantiles of a normal distribution. If the points in the Q-Q plot fall approximately along a straight line, this suggests that your data is normally distributed. For a more formal assessment, MATLAB provides statistical tests such as the Anderson-Darling test (use the `adtest` function, cf. <https://www.mathworks.com/help/stats/adtest.html>) and the Kolmogorov-Smirnov test (use `kstest`, cf. <https://www.mathworks.com/help/stats/kstest.html>). When the assumptions of the t-test are not satisfied, particularly if the data is not normally distributed or the sample size is small, it may be more appropriate to use another test such as the Wilcoxon signed-rank test for paired data and the Mann-Whitney U test for independent samples. For the

Wilcoxon signed-rank test, use `signrank` and for the Mann-Whitney U test, use `ranksum`. You can find the documentation under <https://www.mathworks.com/help/stats/signrank.html> and <https://www.mathworks.com/help/stats/ranksum.html>.

Nevertheless, for the illustrative sake of this example, and despite these considerations, we will proceed here with a one-sample t-test on our small dataset by using the following command:

```
[h, p] = ttest(wordLengths, hypothesizedMean);
```

Here, the function returns two outputs: `h`, a binary indicator that indicates whether the null hypothesis is rejected (in this case  $h = 1$ ) or not ( $h = 0$ ), and `p`, the p-value that helps assess the statistical significance of the result. In this case, the results are  $h = 0$  and  $p = 1$ . Since  $h$  is 0, this suggests there is not enough evidence to conclude that the sample mean differs significantly from the hypothesised mean. A p-value of 1 further supports this conclusion and indicates that the observed data is highly consistent with the null hypothesis. The result suggests that there is not enough evidence to conclude that the sample mean differs significantly from the hypothesised mean and we fail to reject the null hypothesis at the 5% significance level.

If you want to compare the means of two independent groups and determine if there is a statistically significant difference in some measured linguistic feature between those groups, you can use the `ttest2` function to perform a two-sample t-test. You can find the documentation under <https://www.mathworks.com/help/stats/ttest2.html> and <https://www.mathworks.com/help/stats/hypothesis-testing.html>. This function also assumes that the data are approximately normally distributed, and it tests the null hypothesis that the means of two independent samples are equal. The syntax for `ttest2` is similar to that of the `ttest` function, but it is specifically designed for comparing two groups.

You can use `h = ttest2(x, y)` for a basic two-sample t-test. When you want to obtain both the test decision and the p-value, use `[h, p] = ttest2(x, y)`.

To perform a two-sample t-test with a specific significance level you can also use the above-mentioned 'Alpha' name-value pair argument as in:

```
[h, p] = ttest2(x, y, 'Alpha', 0.01);
```

This line performs a two-sample t-test on the data in vectors `x` and `y`, using a significance level of 0.01.

You can also use the `ci` and `stats` outputs when performing a two-sample t-test with the `ttest2` function to gain additional insights into the test results, e.g.:

```
[h, p, ci, stats] = ttest2(x, y);
```

The `ci` output provides a confidence interval for the difference between the population means to indicate the range within which the true mean difference is likely to fall. The `stats` output contains statistics such as the t-statistic, degrees of freedom and standard deviation.

The t-tests covered here, although presented with relatively simple examples, provide a useful starting point for understanding how to assess differences between groups and how to use MATLAB for that purpose. MATLAB offers many other statistical tests and methods for vari-

ous types of analyses. Due to the scope of this book, we cannot explore all of them in detail. For a deeper understanding of the tests themselves, it might be useful to consult additional references.

In addition to the t-tests we have discussed, the Statistics and Machine Learning Toolbox in MATLAB, for instance, also includes various functions for chi-square tests, which are commonly used in hypothesis testing, particularly for assessing goodness of fit and independence in categorical data, cf. the MathWorks documentation at:

*Chi-square distribution:*

<https://www.mathworks.com/help/stats/chi-square-distribution.html>

To use Pearson's Chi-square ( $\chi^2$ ) test in MATLAB, you can use the `chi2gof` function, which is designed for goodness-of-fit tests (<https://www.mathworks.com/help/stats/chi2gof.html>). This function helps you determine if your observed data significantly deviates from what would be expected under a specified distribution. In other words, it may help us determine whether the observed frequencies (such as how often something occurs in a dataset) are significantly different from the expected frequencies (what we would expect based on a hypothesis).

The following command returns a test decision `h` for the null hypothesis that the data in vector `x` follows a normal distribution, with the mean and variance estimated from `x`. If `h` is 1, the null hypothesis is rejected at the 5% significance level; if `h` is 0, it is not rejected.

```
h = chi2gof(x);
```

To return the test decision `h` and the p-value `p` of the hypothesis test, use:

```
[h, p] = chi2gof(x);
```

You can also request a third output, `stats`, from the `chi2gof` function to obtain a structure that contains information about the test statistic and other relevant details of the chi-square goodness-of-fit test including, for instance, the value of the test statistic calculated for the data and the degrees of freedom used in the test, which indicate how many values in the dataset can vary independently without violating any constraints imposed by the model. In this test, this is influenced by the number of bins used to group the data and the number of parameters estimated from the data. A higher number of degrees of freedom generally leads to a more reliable test.

When you use the `chi2gof` function with name-value pairs, you can specify additional options that modify the behaviour of the chi-square goodness-of-fit test. This allows you to test against distributions other than the default normal distribution or to change the significance level of the test.

If you want, for instance, perform the test at a 1% significance level use:

```
[h_1, p_1] = chi2gof(x, 'Alpha', 0.01);
```

Let us look at the following example: In this case, we want to see if the word “water” appears equally across the following three corpus subsections of the British National Corpus (BNC, Davies 2004, <https://www.english-corpora.org/bnc/>): Spoken, Newspaper and Fiction. We retrieve the following absolute frequencies for the word “water” from the BNC via its query interface: Spoken: 2897, Newspaper: 2845, Fiction: 6285.

The chi-square goodness-of-fit test allows us to compare the observed frequencies of the word “water” across different contexts against expected frequencies. In this case, if we assume a uniform distribution (meaning the word “water” should appear equally in each context, i.e., in each corpus subsection), we can calculate the expected frequency for each context by dividing the total frequency by the number of contexts.

First, we calculate the total frequency (12027 in this case) across all contexts:

```
observed_counts = [2897, 2845, 6285];
total_frequency = sum(observed_counts);
```

The expected frequency for each context is calculated by dividing the total frequency by the number of contexts (which is 3 in this case). This gives you the expected count of 4009 for each context ( $12027 / 3$ ), assuming the word “water” appears equally across the Spoken, Newspaper and Fiction sections in the BNC. For a uniform distribution, the expected frequency for each context is calculated as follows:

```
expected_counts = total_frequency / length(observed_counts) *
ones(size(observed_counts));
```

Here, the `length` function returns the number of elements in the largest dimension of an array. In our case, `length(observed_counts)` returns the number of contexts (which is 3: Spoken, Newspaper and Fiction). This value is used to determine how many equal expected frequencies should be calculated for the uniform distribution. The `ones` function creates an array filled with ones. When you use `ones(size(observed_counts))`, it generates an array of ones that has the same size as `observed_counts`. This is useful for multiplying by the expected frequency value to create an array where each element represents the expected count for each context under the assumption of a uniform distribution.

This result is: `expected_counts = [4009, 4009, 4009];`

This indicates that, under the uniform distribution assumption, the word “water” is expected to appear 4009 times in each context.

The `chi2gof` function then tests the null hypothesis that the observed counts follow the expected distribution.

```
[h, p] = chi2gof(observed_counts, 'Expected', expected_counts);
```

If you receive `h = 1`, it indicates that the null hypothesis is rejected at the 5% significance level. This means that the observed counts of the word “water” across the contexts (Spoken, Newspaper, Fiction) do not fit the expected uniform distribution, which means that they are significantly different from what would be expected if the word appeared uniformly across these contexts. If `h = 0`, it indicates that the null hypothesis cannot be rejected at the 5% significance level. This means that the observed data does not provide sufficient evidence to conclude that it comes from a distribution that is significantly different from the expected distribution.

The p-value returned by `chi2gof` quantifies the strength of the evidence against the null hypothesis. A small p-value (typically less than 0.05) indicates strong evidence against the null

hypothesis, while a larger p-value suggests that the observed distribution could reasonably fit the expected distribution. Therefore, if  $p$  is very low, it implies that the distribution of the word “water” is not uniform across the contexts.

In our example, we obtain the following results from the chi-square goodness-of-fit test:  $h = 1$ . This indicates that we can reject the null hypothesis and that the observed frequencies of the word “water” across the three contexts are not equal. Additionally, we obtain  $p = 0$ . In MATLAB, encountering a p-value of exactly 0 during a statistical test typically indicates that the test statistic is so extreme that the calculated p-value falls below the smallest representable value in the software’s numerical precision. This situation arises when the observed data provides overwhelming evidence against the null hypothesis. This suggests that the probability of observing the data (or something more extreme) under the null hypothesis is extremely low.

Let us have a look at another example to compare the frequencies of two different expressions of future intentions in combination with the verb “go” across two distinct time periods in COHA (<https://www.english-corpora.org/coha/>): We will look for “will go” (Form 1) and “going to go” (Form 2) in the 1980s and the 2000s and use the chi-square test. It will help to assess whether there is a significant relationship between two categorical variables: the expressions used and the time periods. Essentially, this test evaluates if the frequency of each expression varies significantly across the different time periods or if their usage is independent of the time period. If a significant association is found, it suggests that the choice of expression is influenced by the time period in which it is used.

Our null hypothesis (H0) is: The usage of “will go” and “going to go” is independent of the time period. The alternative hypothesis (H1) is: The usage of “will go” and “going to go” is not independent of the time period.

We can extract the following frequency values from COHA (Table 17):

Form	1980s	2000s
Form 1 ( <i>will go</i> )	693	613
Form 2 ( <i>going to go</i> )	216	413

Table 17: Usage of “will go” and “going to go” in 2 decades in COHA

We can then define our observed counts in a 2x2 matrix format representing the observed counts of the two categorical variables.

```
observed_counts = [693, 613; 216, 413];
```

Next, we can calculate the row totals and column totals:

```
row_totals = sum(observed_counts, 2);
```

```
column_totals = sum(observed_counts, 1);
```

Here, to calculate the totals for each row, we use the `sum` function with the second argument set to 2. To calculate the totals for each column, use the `sum` function with the first argument set to 1.

Then we calculate the grand total:

```
grand_total = sum(observed_counts(:));
```

Note that the element in brackets `(:)` in `grand_total = sum(observed_counts(:))` is used to reshape the `observed_counts` matrix into a column vector. This syntax allows you to convert the matrix into a single column vector containing all the elements of the matrix, regardless of its original dimensions. The `sum` function then computes the total sum of all elements in this reshaped column vector.

Alternatively, simply use the following command:

```
grand_total = sum(observed_counts, "all");
```

This calculates the total of all elements in the matrix. Using `"all"` simplifies the operation and allows you to obtain the grand total directly without reshaping the matrix into a vector.

Table 18 presents the total values derived from our data, calculated using the commands mentioned above.

Form	1980s	2000s	Row Total
Form 1 ( <i>will go</i> )	693	613	1306
Form 2 ( <i>going to go</i> )	216	413	629
<b>Total</b>	909	1026	1935

Table 18: Individual and total frequencies of “will go” and “going to go” in 2 decades in COHA

With these totals, we can calculate the expected counts for each cell in the contingency table using the formula:

```
expected_counts = (row_totals * column_totals) / grand_total
```

This calculates the expected counts for each cell in the contingency table based on the assumption that the two variables are independent. The output will be:

```
expected_counts =
    613.5163    692.4837
    295.4837    333.5163
```

Once we have both the observed and expected counts, we can perform the chi-square test using the `chi2gof` function:

```
[h, p] = chi2gof(observed_counts(:), 'Expected', expected_counts(:));
```

The command performs a chi-squared goodness-of-fit test to compare the observed counts in the contingency table with the expected counts based on marginal totals. The variable `observed_counts` contains the actual data, while `expected_counts` represents the theoretical distribution we are testing against. The `'Expected'` parameter is used to specify the expected counts for each category under the null hypothesis. `'Expected'` indicates to the function that the following argument (`expected_counts(:)`) should be treated as the expected counts. The function compares the expected values against the observed counts to assess whether any significant differences exist. If the observed counts deviate significantly from the expected counts, it suggests that the data may not fit the hypothesised distribution. It returns `h`, which indicates

whether to reject the null hypothesis (1 means reject, 0 means fail to reject) and  $p$ , the p-value associated with the test.

In summary, the examples provided in this section illustrate a selection of statistical tests that can be performed using MATLAB, including the t-test and the chi-square test. Additionally, there are various other functions worth exploring, such as the `fishertest` function, which returns a test decision for Fisher's exact test (<https://www.mathworks.com/help/stats/fishertest.html>). Like the chi-square test, Fisher's exact test is used to evaluate the null hypothesis that there are no non-random associations between two categorical variables, but Fisher's exact test is preferred for small sample sizes or analysing 2x2 contingency tables, while the chi-square test is more suitable for larger datasets where the assumptions of the test are met.

MATLAB and its Statistics and Machine Learning Toolbox offer a comprehensive array of functions for statistical tests. As noted earlier, the official MATLAB documentation provides detailed guidance on a wide range of these tests and their applications.

## 9.4 Analysis of variance

This section provides a brief overview of how MATLAB supports Analysis of Variance (ANOVA). It focuses on the availability of this tool rather than offering in-depth explanations or detailed examples, which are beyond the scope of this introductory text. ANOVA is used to answer questions similar to those addressed by t-tests – namely, whether there are significant differences between group means. However, while t-tests are limited to comparing two groups at a time, ANOVA is specifically designed to handle comparisons across three or more groups simultaneously.

The `anova1` function in MATLAB is used to perform one-way ANOVA, a statistical method for comparing the means of multiple groups based on a single factor. It evaluates whether observed differences in group means are statistically significant by analysing the variation between and within groups. In this function, each column of the input data is treated as a separate group. The null hypothesis assumes all group means are equal, while the alternative proposes that at least one differs.

Suppose you have collected data on how often passive constructions appear in different text types in a corpus. For example, you might have counts of passives per text in academic prose, fiction and news in your data. To analyse whether these frequencies differ significantly across text types, you can organise the data into a matrix, where each column corresponds to a specific text type and each row corresponds to an individual text sample.

```
% Sample data: Frequency of passives for three text types

academic_prose = [15, 18, 20, 22, 19];
fiction = [10, 12, 11, 14, 13];
news = [20, 21, 19, 22, 23];

% Combine the data into a single matrix (the single quote (') is used to
denote the transpose operation and to convert row vectors into column vectors)

frequencyData = [academic_prose', fiction', news'];
```

You can then perform a one-way ANOVA to test for significant differences in passive frequencies across the text types by using the `anova1` function:

```
[p, tbl, stats] = anova1(frequencyData)
```

The output will consist of several components (Figure 193).

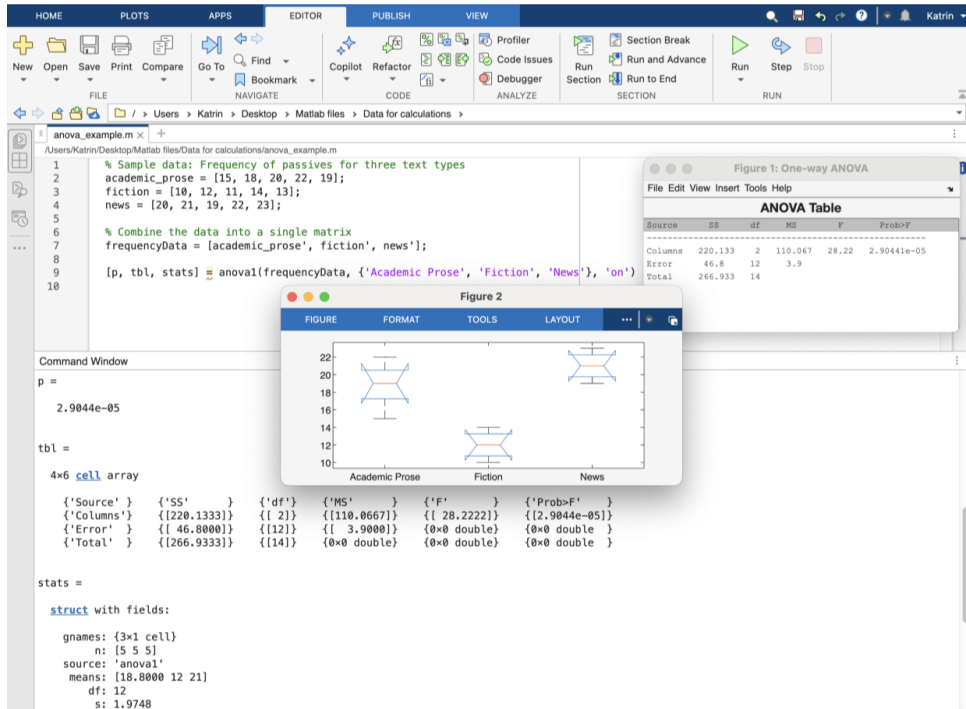


Figure 193: Output components from `anova1` function

First, the function returns a p-value that helps to determine whether the observed differences are likely due to chance. The p-value here indicates whether there are significant differences in the means of the groups represented in `frequencyData`. A p-value less than 0.05 typically suggests that at least one group mean is significantly different from the others, indicating a rejection of the null hypothesis.

The `anova1` function also produces a table. The output `tbl` is a cell array that contains the standard ANOVA table. This table helps to interpret the results by summarising key components of the analysis: sum of squares, degrees of freedom, mean squares, the F-statistic and the p-value. The `stats` output from the command provides a structure containing statistical information for further analysis. Furthermore, the `anova1` function automatically displays a box plot for each group in the data so that you can compare the distributions of the different groups visually and assess the presence of any outliers. If you want to ensure that the box plot is displayed, you can also use the command with the `display` option set to `'on'`. Additionally, by including extra parameters, one can specify the names of the groups, which enhances the interpretability of the output by labelling the groups in the plot:

```
[p, tbl, stats] = anova1(frequencyData, {'Academic Prose', 'Fiction',
'News'}, 'on');
```

To study how two independent variables (factors) influence a single outcome, MATLAB provides the function `anova2`, which performs a two-way analysis of variance (two-way ANOVA). This method evaluates whether each factor has a statistically significant effect on the outcome (main effects) and whether the two factors interact (interaction effect). It returns p-values indicating whether these effects are unlikely to have occurred by chance.

As an example, consider a study examining how two factors influence parsing accuracy. The first factor is passive construction type, distinguishing between short passives without an agentive phrase (e.g., “*The door was opened.*”) and long passives with an explicit agent (e.g., “*The door was opened by a woman.*”). The second factor is text type, such as academic prose and fiction. The dependent variable is parsing accuracy, for instance measured as a percentage. A two-way ANOVA allows us to test whether passive type and text type each affect parsing accuracy and whether the effect of passive type depends on text type. This latter case represents the interaction effect, which shows whether the influence of one factor changes across levels of the other.

To perform the analysis in MATLAB, the data must be arranged in a specific matrix format. In MATLAB’s `anova2`, Factor A (passive type) is treated as the row factor, and Factor B (text type) is treated as the column factor. The design must be balanced, meaning that each combination of factor levels contains the same number of observations (replicates). Replicated measurements are stored within the same matrix by stacking them in adjacent row blocks, with each block corresponding to a level of Factor B. The number of replicates per condition is provided separately.

The analysis is carried out using the basic syntax:

```
[p, tbl, stats] = anova2(Y, reps)
```

Here, `Y` is the data matrix in which columns represent the levels of Factor B (text type). Within each column, observations are grouped so that values for each level of Factor A (passive type) are stored together in consecutive blocks of rows. The input argument `reps` specifies the number of replicates (repeated measurements) per condition. The function returns three outputs. The first output, `p`, contains the p-values for the two main effects and their interaction. The second output, `tbl`, is an ANOVA table reporting sums of squares, degrees of freedom, mean squares, F-statistics and associated p-values. The third output, `stats`, is a structure that can be used for follow-up analyses, including post-hoc comparisons between specific groups.

It is important to note that the dependent variable should be numeric and continuous rather than categorical counts. Moreover, more than one observation per condition is required to estimate the interaction effect reliably; if only a single observation is available for each combination of factor levels, the interaction cannot be estimated and MATLAB may return NaN for that term. Providing detailed MATLAB code examples for `anova2` goes beyond the scope of this introductory book. For implementation details, readers can consult the MATLAB Statistics and Machine Learning Toolbox documentation, particularly the `anova2` reference page and related function guides.

Beyond one-way ANOVA and two-way ANOVA, MATLAB offers additional statistical methods such as Analysis of Covariance (ANCOVA) and Multivariate ANOVA (MANOVA),

which are useful for more complex data analysis scenarios. For a more detailed discussion of analytical methods involving ANOVA in linguistics, readers are referred to works that focus specifically on this topic, such as Woodrow (2014a: 73ff). For detailed documentation on these functions and their applications in MATLAB, you can also refer to the following resources from MathWorks:

*Analysis of Variance and Covariance*

<https://www.mathworks.com/help/stats/analysis-of-variance-and-covariance.html>

*One-way ANOVA*

<https://www.mathworks.com/help/stats/one-way-anova.html>

*Two-Way ANOVA*

<https://www.mathworks.com/help/stats/two-way-anova.html>

*anova1 – One-way analysis of variance*

<https://www.mathworks.com/help/stats/anova1.html>

*anova2 – Two-way analysis of variance*

<https://www.mathworks.com/help/stats/anova2.html>

These sections from the MathWorks documentation offer more in-depth explanations, syntax examples and guidance on how to use each function in MATLAB.

## 9.5 Regression models

This chapter offers a high-level summary of how MATLAB enables regression analysis. It does not aim to cover detailed explanations or offer comprehensive examples, as such topics fall outside the scope of this introductory book. It is advisable for readers to consult additional works that specifically address this subject, such as those authored by Woodrow (2014b: 85ff.). For detailed documentation on these functions and how they can be used in MATLAB, you can also check out the following resources provided by MathWorks:

*Regression*

<https://www.mathworks.com/help/stats/regression-and-anova.html>

*Linear regression*

<https://www.mathworks.com/help/stats/linear-regression.html>

*Linear regression model workflow*

<https://www.mathworks.com/help/stats/linear-regression-model-workflow.html>

*Linear and nonlinear regression*

<https://www.mathworks.com/help/curvefit/linear-and-nonlinear-regression.html>

*Nonlinear Regression*

<https://www.mathworks.com/help/stats/nonlinear-regression.html>

Regression analysis is a statistical method used to understand the relationship between a dependent variable (the outcome or response) and one or more independent variables (the predic-

tors or features). This technique is prevalent in various fields, including applied linguistics and corpus linguistics, where it helps researchers examine how different variables interact and influence each other.

In regression analysis, the goal is often to predict the value of the dependent variable based on the values of the independent variables. This is similar to other statistical methods like ANOVA and correlation, but regression specifically focuses on prediction and the strength of relationships. Regression is used to answer questions such as whether the independent variables can effectively predict the dependent variable.

The Statistics and Machine Learning Toolbox in MATLAB offers a variety of tools for performing regression. They range from linear models to more advanced techniques such as generalised linear models, nonlinear regression and nonparametric regression. By using the `fitlm` function in MATLAB, for instance, you can fit a linear regression model to your data and analyse the coefficients to assess the significance of each predictor (<https://www.mathworks.com/help/stats/fitlm.html>). This function provides detailed statistics, including t-statistics and p-values, which help determine whether the predictors significantly contribute to the model. For instance, a low p-value indicates that the corresponding predictor is statistically significant. On the other hand, the `fitglm` function is designed to fit generalised linear models, which are particularly useful when the response variable follows a distribution other than the normal distribution (<https://www.mathworks.com/help/stats/fitglm.html>).

For example, if you want to predict the likelihood of the word “absolutely” appearing in a document, you can consider several features that may influence this occurrence, such as the document’s text type (e.g., blog, fiction, news), style (formal or informal) and length (total number of words). By employing regression analysis, you can quantify the impact of each of these features on the probability of the word “absolutely” appearing.

To develop such a model, you would first gather data on the occurrences of the word “absolutely” across a collection of documents, along with the relevant features. The response variable could be defined as a binary indicator (1 if the word appears, 0 if it does not) or as a count of its occurrences, allowing for a more nuanced prediction.

The independent variables (predictors) might include:

- Text type: A categorical variable where different document types are encoded numerically (e.g., 1 for blogs, 2 for fiction, 3 for news etc.).
- Style: A binary variable indicating formality (1 for formal, 0 for informal).
- Length: A continuous variable representing the number of words in the document.

Once the data is prepared, you can use regression models from the Statistics and Machine Learning Toolbox to analyse how these features influence the presence or frequency of the word “absolutely.”

Here is a simple example of how you might implement this in MATLAB:

```
% Sample data - Note that in this example, the ellipsis (...) acts only as a
% placeholder indicating where additional user-defined data should be added.
% Although in MATLAB the ellipsis is normally used for line continuation,
% here it is included purely for illustrative purposes.
```

```

textType = [1; 2; 1; 3; ...]; % Complete this array with additional data to
have a larger dataset; 1: News, 2: Fiction, 3: Academic etc.

style = [1; 0; 1; 0; ...]; % Complete this array; 1: Formal, 0: Informal

length = [500; 300; 800; 450; ...]; % Complete this array (length of documents
in tokens)

wordPresence = [1; 0; 1; 0; ...]; % Complete this array; 1: "absolutely"
appears, 0: it does not appear

% Create a table for regression

tbl = table(textType, style, length, wordPresence);

% Fit a logistic regression model

mdl = fitglm(tbl, 'wordPresence ~ textType + style + length', 'Distribution',
'binomial');

% Display the model summary

disp(mdl);

```

By analysing the output, you can gain insights into how each feature contributes to the likelihood of the word appearing in the documents.

You may also try out MATLAB's Regression Learner app, which allows for interactive exploration and refinement of regression models once you have gathered the necessary data, cf.

#### *Regression Learner app*

<https://www.mathworks.com/help/stats/regression-learner-app.html>

#### *Train regression models in Regression Learner app*

<https://www.mathworks.com/help/stats/train-regression-models-in-regression-learner-app.html>

#### *Visualise and assess model performance in Regression Learner app*

<https://www.mathworks.com/help/stats/assess-model-performance-in-regression-learner.html>

The Regression Learner app provides an easy-to-use interface where you can fit various types of regression models, assess their performance and adjust hyperparameters without extensive coding. With the Regression Learner app, you can select from a range of regression algorithms, including linear regression, regression trees, support vector machines, among others. The app enables you to visualise results and evaluate model accuracy to make it easier to identify the best-performing model for your specific dataset.

## 9.6 Correlation coefficients

Correlation coefficients are used to measure the strength and direction of the linear relationship between two or more variables. The primary function for calculating these coefficients in MATLAB is `corrcoef`, which calculates the Pearson correlation coefficient and returns a matrix showing the correlation values between the variables in your dataset. The values returned by `corrcoef` range from -1 to 1. A value close to 1 indicates a strong positive linear relationship. A value close to -1 indicates a strong negative linear relationship. A value near 0 suggests

little or no linear relationship. However, a zero value or a value close to zero does not imply that the variables are completely independent; they may still have a nonlinear relationship that is not captured by the correlation coefficient.

When you have a matrix  $A$ , where each column represents a different variable and each row represents an observation, you can compute the correlation matrix with:

```
R = corrcoef(A);
```

The result  $R$  is a square matrix where each element shows how the variables in  $A$  relate to each other in terms of linear correlation. The diagonal values of  $R$  are always 1, as they represent the correlation of each variable with itself.

If you want to calculate the correlation between two separate vectors or columns with the same number of elements, you can use:

```
R = corrcoef(A, B);
```

MATLAB will treat  $A$  and  $B$  as two separate variables and calculate the correlation between them. The result  $R$  is a 2-by-2 matrix in which  $R(1,1)$  and  $R(2,2)$  are both 1, since each variable correlates with itself.  $R(1,2)$  and  $R(2,1)$  show the correlation coefficient between  $A$  and  $B$ .

Additionally, you can obtain p-values to assess the significance of the correlations by using:

```
[R, P] = corrcoef(A);
```

or

```
[R, P] = corrcoef(A, B);
```

This returns both the correlation coefficients and the p-values for significance testing, where the p-values indicate the likelihood that the observed correlation is due to random chance. If an off-diagonal element of  $P$  is less than the significance level (commonly set at 0.05), the corresponding correlation in  $R$  can be considered statistically significant.

Let us have a look at the following example. In Chapter 7.9, we used the data given there in Table 7 (frequencies of cohesive nominal ellipses from corpus parts with German original texts and in corpus parts with English translations from the GECCo corpus) to create a scatter plot (Figure 150 in Chapter 7.9). Cohesive nominal ellipses are structures where the missing part of a noun phrase is implied from the previous textual context. The antecedent is often found in a previous clause or sentence, and the structure of the phrase remains grammatically coherent without it (e.g., *The Alpha Magnetic Spectrometer is by far the most sensitive cosmic ray detector ever to fly in space, and with a price tag of about \$2 billion, it's also the most expensive [ ]*). The scatter plot is provided here again for convenience as Figure 194.

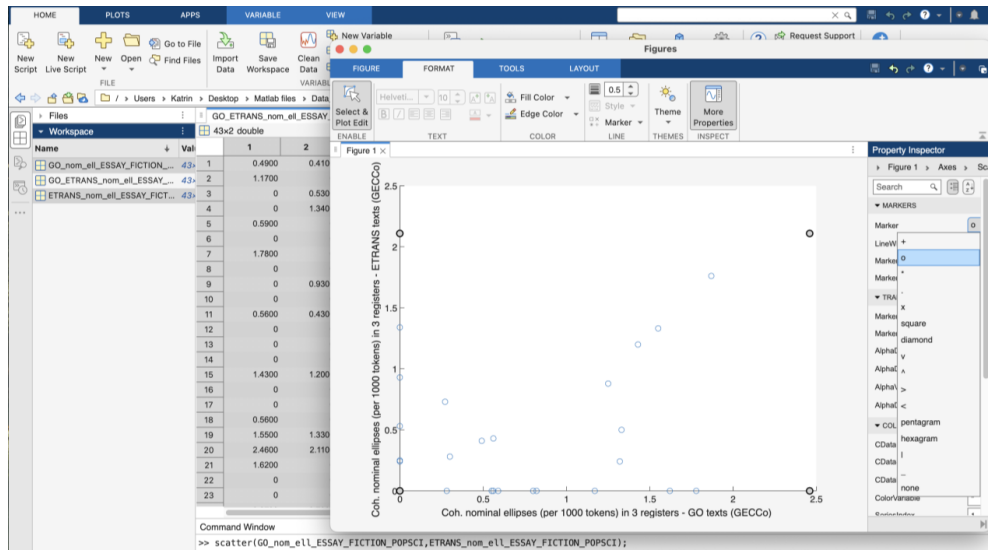


Figure 194: Scatter plot for ellipses in pairs of original and translated texts from three registers from GO/ETRANS in GECCo corpus<sup>33</sup>

The scatter plot offers a visual representation of the potential linear correlation between the frequency of nominal ellipsis as a cohesive device in German original texts (GO) from the GECCo corpus and its occurrence in the English translations (ETRANS) of the corresponding German texts. By plotting the data points, we can observe the overall trend. If the points align closely along a straight line, it suggests a linear relationship. However, a visual inspection alone may not be sufficient to assess the strength of this relationship. To quantify the degree of correlation, we can calculate the linear correlation coefficient  $R$  for the data we have used to create Figure 194.

Let us look at an example using our data from Table 7 in Chapter 7.9. We will work with the MATLAB variables `GO_nom_ell_ESSAY_FICTION_POPSCI` and `ETRANS_nom_ell_ESSAY_FICTION_POPSCI`, which contain the frequencies of cohesive nominal ellipses in the ESSAY, FICTION and POPSCI corpus registers from the GECCo corpus, as specified in Table 7. We can either use these variables separately or combine them into one matrix `GO_ETRANS_nom_ell_ESSAY_FICTION_POPSCI`.

For convenience, the commands below allow you to create the variables `GO_nom_ell_ESSAY_FICTION_POPSCI` and `ETRANS_nom_ell_ESSAY_FICTION_POPSCI` directly in MATLAB, without needing to copy the data from Table 7 manually into a new variable or to save it in an external file for subsequent import into MATLAB. You can simply copy and paste these commands into the Command Window to generate the variables.

```
GO_nom_ell_ESSAY_FICTION_POPSCI = [0.49, 1.17, 0.00, 0.00, 0.59, 0.00, 1.78,
0.00, 0.00, 0.00, 0.56, 0.00, 0.00, 0.00, 1.43, 0.00, 0.00, 0.56, 1.55, 2.46,
1.62, 0.00, 0.00, 0.27, 1.32, 0.00, 0.00, 0.56, 1.25, 0.30, 1.33, 0.00, 1.87,
0.28, 0.00, 0.80, 0.00, 0.82, 0.55, 0.00, 0.28, 0.00, 0.00];
```

<sup>33</sup> cf. Chapter 7.9

```
ETTRANS_nom_ell_ESSAY_FICTION_POPSCI = [0.41, 0.00, 0.53, 1.34, 0.00, 0.00,
0.00, 0.00, 0.93, 0.00, 0.43, 0.00, 0.00, 0.00, 1.20, 0.00, 0.00, 1.33,
2.11, 0.00, 0.00, 0.00, 0.73, 0.24, 0.00, 0.25, 0.00, 0.88, 0.28, 0.50, 0.00,
1.76, 0.00, 0.24, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00];
```

To calculate the linear correlation coefficient, we can then run the following command:

```
R = corrcoef(GO_nom_ell_ESSAY_FICTION_POPSCI,
ETTRANS_nom_ell_ESSAY_FICTION_POPSCI)
```

Alternatively, use the following command if you have combined the data into a single two-column matrix named `GO_ETTRANS_nom_ell_ESSAY_FICTION_POPSCI`:

```
R = corrcoef(GO_ETTRANS_nom_ell_ESSAY_FICTION_POPSCI)
```

We will receive a value of 0.5647 (rounded by MATLAB to four decimal places by default), which suggests a moderate positive linear correlation between the variables (Figure 195).

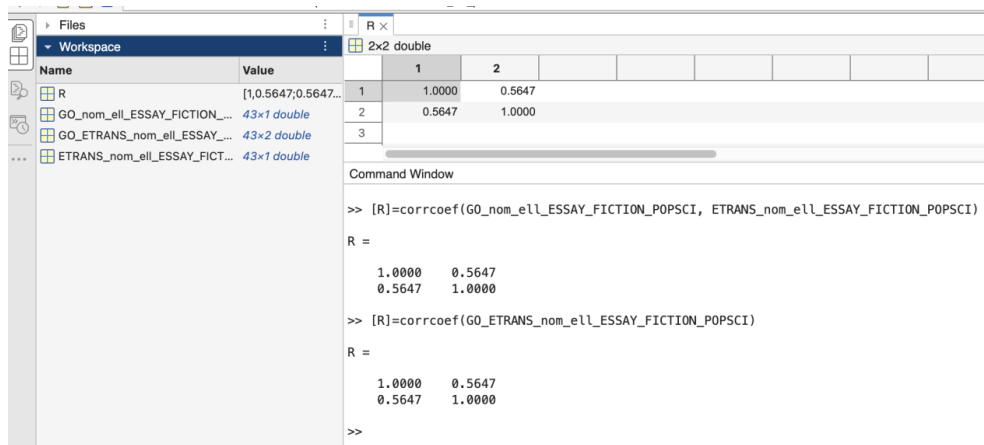


Figure 195: Calculation of correlation coefficient

This moderate positive linear correlation between the variables can be explained as follows: If grammatically possible, translators often follow the syntax of the original text closely and tend to omit the same material in the translation, especially when this helps avoid repeating a textual element that is already clear from its nearby antecedent. However, in some cases, it may be difficult or impossible to use the same type of ellipsis in English as in the original German text. In such situations, translators may insert lexical material (e.g., by repeating omitted elements) or use a substitute (e.g. the nominal substitute “one”). This may be necessary when the elliptical structure cannot be easily mirrored in the translation’s syntax due to grammatical reasons (e.g. the German phrase “ein kleines” may be rendered as “a small one”) or when an elliptical structure would create ambiguity in the target text, for example due to a greater distance between the ellipsis and its antecedent in the translation. Overall, the moderate positive correlation coefficient suggests here that in many cases where the original German text contains a nominal ellipsis, the English translation also includes a similar structure, but it is not always the case.

Let us also take a look at the p-value for these data, which can be used to test the hypothesis that there is no relationship between the observed variables. The p-value represents the proba-

bility of obtaining the observed correlation coefficient if the true correlation is actually zero. If the p-value here is smaller than the chosen significance level (by default 0.05), then the correlation between the two variables can be considered statistically significant.

To calculate both the correlation coefficient and the p-value for these data, use a command like this:

```
[R,P] = corrcoef(GO_nom_ell_ESSAY_FICTION_POPSCI,
ETTRANS_nom_ell_ESSAY_FICTION_POPSCI)
```

or

```
[R,P] = corrcoef(GO_ETTRANS_nom_ell_ESSAY_FICTION_POPSCI)
```

or

```
[R,P]=corrcoef(GO_ETTRANS_nom_ell_ESSAY_FICTION_POPSCI (:,1),
GO_ETTRANS_nom_ell_ESSAY_FICTION_POPSCI (:,2));
```

These commands will all give you the same results (Figure 196). The p-value in this case is smaller than 0.05 and suggests that our results are statistically significant.

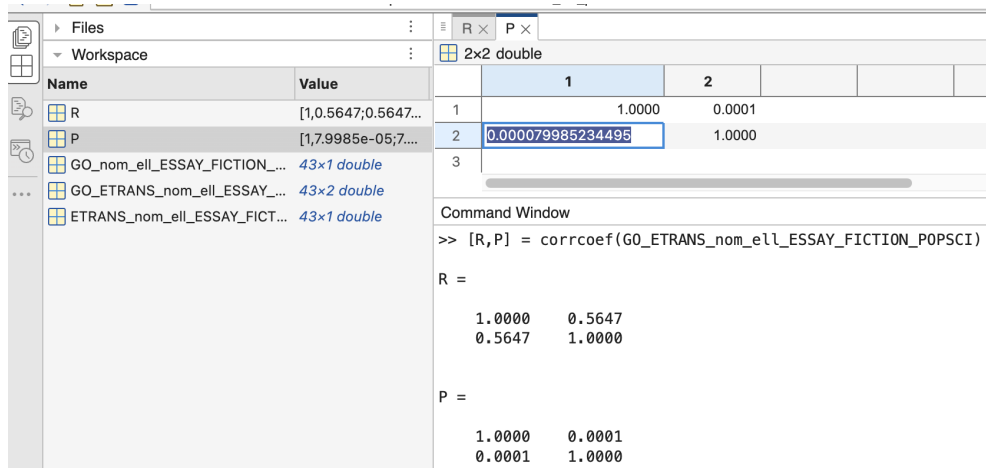


Figure 196: Calculation of correlation coefficient and p-value for data from GECCo corpus

Note that, as in this example, the p-value may appear as 0.0001 in MATLAB because of the way the numbers are formatted for display. This means the actual p-value could be smaller, but MATLAB rounds it to four decimal places by default. You can click on the p-value variable in the Workspace panel or open it in the Variables editor to see a more precise value. Alternatively, you can change the display format in MATLAB to `long` to view more decimal places (Figure 197).

```

>> format long
>> [R,P] = corrcoef(GO_ETRANS_nom_e11_ESSAY_FICTION_POPSCI)

R =

    1.000000000000000    0.564674442783865
    0.564674442783865    1.000000000000000

P =

    1.000000000000000    0.000079985234495
    0.000079985234495    1.000000000000000

```

Figure 197: Changing number display format to show more decimal places of correlation coefficient and p-value

Let us now look at another example for which we have already created a scatter plot in Chapter 7.9 (cf. Table 8, Figure 151). In this example we worked with the normalised frequencies of some of the most common initialisms and their corresponding full forms in the English data from the EuroParl-UdS corpus (e.g., “CAP” and “Common agricultural policy”). We have also already added a linear regression line via the basic fitting tool that represents the best linear approximation of the relationship between the two variables (Figure 152 above which is provided here again as Figure 198).

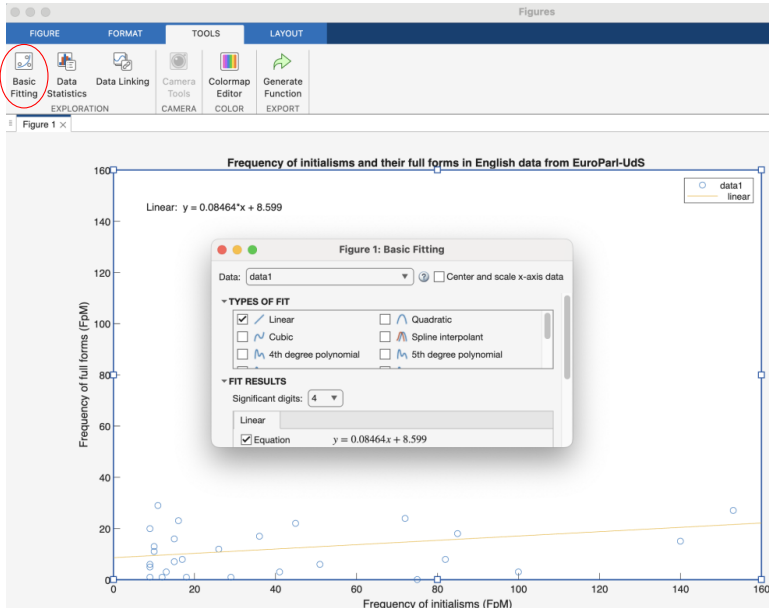


Figure 198: Scatter plot for initialisms vs. their full forms in English EuroParl-UdS data with linear regression line

The scatter plot for these data does not show a strong or consistent trend. The low slope of the line suggests that there might be some relationship between the variables, but the frequency of

the initialisms in the English texts of the European Parliament debates seems only weakly correlated with the frequency of the corresponding full forms in these texts.

We will now proceed to calculate the correlation coefficient and the p-value for these data provided in Chapter 7.9 in Table 8. We can either use two separate numeric variables or a single numeric matrix with two columns containing the frequency values, as demonstrated earlier with the `Initialisms_vs_full_forms` variable. If the data are in a single matrix, we can use the following command:

```
[R,P] = corrcoef(Initialisms_vs_full_forms)
```

The output, displayed in Figure 199, is as follows:

```
R = 0.2715
```

```
P = 0.1466
```

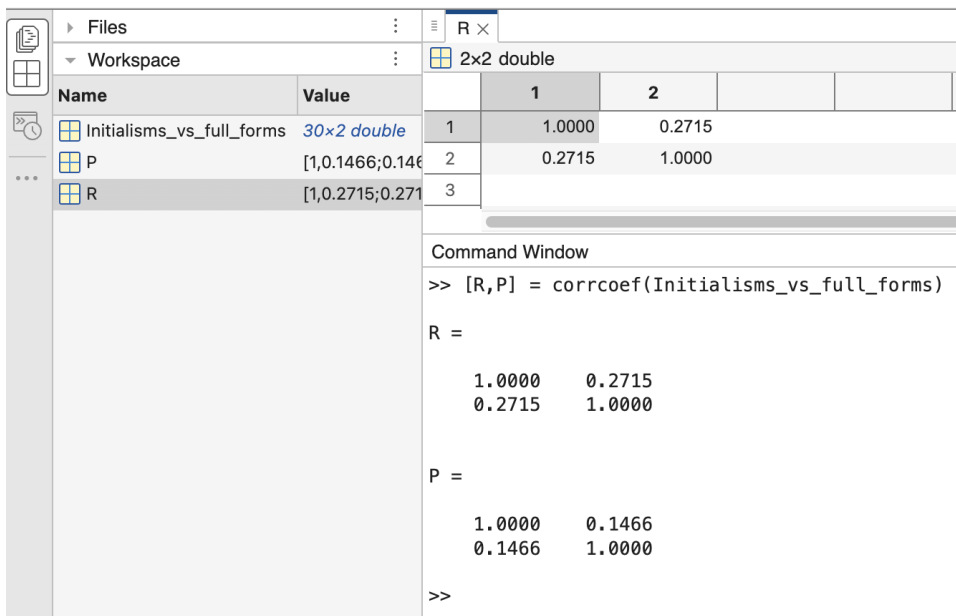


Figure 199: Calculation of correlation coefficient and p-value for data from EuroParl-UdS

The correlation coefficient  $R = 0.2715$  indicates a low positive correlation between the two variables. The more frequent the initialisms in the data, the higher the frequency of their corresponding full forms in various cases, but this relationship is not consistent. The p-value with  $0.1466$  is relatively high. As mentioned above, the typical threshold for significance is  $0.05$ . This suggests that the correlation is not statistically significant, and we cannot confidently assert a linear relationship between these variables.

To calculate the correlation coefficient in MATLAB, you can also use the `corr` function from the Statistics and Machine Learning Toolbox. The `corr` function also computes correlation coefficients between two datasets or within a single dataset and can return the correlation coefficients and the p-values that indicate the significance of the correlations.

The `corr` function allows you to specify the type of correlation using the name-value pair `'Type'`. The default behaviour of the `corr` function in MATLAB is to compute Pearson's correlation coefficient. Therefore, when you call `corr(X)`, it calculates the pairwise linear correlation coefficients between the columns of the matrix `X` using Pearson's method. Pearson's correlation is best used when working with continuous data and variables that are normally distributed. When calculating correlation coefficients for your data, keep also in mind that they can be influenced by extreme outliers, which may distort the results. Additionally, similar to *p*-values, correlation coefficients are subject to sampling variation. In larger datasets, you might need to set a higher threshold for the correlation coefficient to identify a strong relationship, as the increased sample size provides more reliable estimates of correlation.

For certain datasets, it may be more appropriate to compute correlation coefficients other than Pearson's. You can also compute Kendall's Tau Coefficient using the `corr` function. To do so specify the `'Type'` name-value pair as `'Kendall'` as in:

```
R_kendall = corr(X, 'Type', 'Kendall');
```

This allows you to assess the strength and direction of association between two variables based on the ranks of the data rather than their raw values. Kendall's correlation is particularly well-suited for ordinal data due to its robustness in handling ranks. This method is less sensitive to outliers compared to other correlation methods and works better with smaller datasets where outliers can disproportionately affect the results.

Additionally, you can compute Spearman's rank correlation coefficient using the `corr` function by specifying the `'Type'` name-value pair as `'Spearman'` as in the following command:

```
R_spearman = corr(X, 'Type', 'Spearman');
```

Like Kendall's Tau, Spearman's correlation assesses the strength and direction of the association between two variables based on their ranks.

For more details on the `corr` function and the different types of correlation coefficients (including Pearson's linear correlation coefficient, Kendall's Tau and Spearman's Rho), you can check the MATLAB documentation on correlation (cf. <https://www.mathworks.com/help/stats/corr.html>).

Also have a look at documentation under the following links to explore the MATLAB documentation on the `corrcoef` function and the topic of correlation:

*corrcoef* – Correlation coefficients  
<https://www.mathworks.com/help/matlab/ref/corrcoef.html>

*Linear correlation*  
[https://www.mathworks.com/help/matlab/data\\_analysis/linear-correlation.html](https://www.mathworks.com/help/matlab/data_analysis/linear-correlation.html)

*Identify and visualise correlated variables*  
[https://www.mathworks.com/help/matlab/data\\_analysis/visualize-corrcoef.html](https://www.mathworks.com/help/matlab/data_analysis/visualize-corrcoef.html)

# 10 Text preprocessing and analysis with MATLAB's® Text AnalyticsToolbox™

## 10.1 Overview of the Text Analytics Toolbox™

As already demonstrated in this book, MATLAB can be used for various text preprocessing and text analysis procedures that are highly useful for analysing corpus data. Many of these procedures are integrated into MATLAB's core functionality. However, the Text Analytics Toolbox expands on this by offering a suite of specialised functions designed specifically for handling and analysing text data (cf. <https://www.mathworks.com/products/text-analytics.html>). We have already covered some examples of functions from the Text Analytics Toolbox, e.g., when we created word clouds to visualise language data in Chapter 7.13. or when we removed punctuation marks from textual data as a data preprocessing step in Chapter 5.2.3.

The Text Analytics Toolbox provides a wide range of functions that facilitate the extraction, cleaning and analysis of textual data from various sources including PDF, HTML and Microsoft Word files. It allows for easy preprocessing of text, such as removing unwanted elements, lemmatising words and segmenting text into sentences. The toolbox enables the straightforward creation of tokenised documents, which break text into smaller, more manageable components. By using the toolbox, you can also identify and classify named entities in texts, such as names, organisations and locations. Visualisation options from the toolbox, such as word clouds and scatter plots, make it easy to explore text data and analyse word frequencies. Additionally, the Text Analytics Toolbox supports techniques such as working with Universal Dependencies, which can be used to analyse the syntactic structure of sentences and to create dependency graphs to visualise the relationships between the words and constituents in a sentence.

The Text Analytics Toolbox provides functions for creating bag-of-words and bag-of-n-gram models. Moreover, it includes functions for working with word embeddings, which are essential for many natural language processing tasks. With support for similarity analysis, the toolbox also enables the measurement of document similarities. The Text Analytics Toolbox can be used to build predictive models, and it supports various machine learning and deep learning models for advanced natural language processing tasks. It can be used in conjunction with other toolboxes, such as the Deep Learning Toolbox. For instance, you can apply deep learning algorithms to text analytics applications, such as text classification and language translation.

For further details, you can refer to the MathWorks documentation on the Text Analytics Toolbox:

*Text Analytics Toolbox*

<https://de.mathworks.com/help/textanalytics/>

*Get started with Text Analytics Toolbox*

<https://www.mathworks.com/help/textanalytics/getting-started-with-text-analytics-toolbox.html>

For in-depth coverage of specific functions, please refer to the relevant subchapters of the MathWorks documentation.

In the upcoming sections, we will explore techniques for extracting text from various types of documents by using functions from the Text Analytics Toolbox as well as various preprocessing steps for preparing text data for analysis. We will see examples of visualisation techniques, such as generating word clouds and dependency graphs. While we will have a look at various functionalities of the Text Analytics Toolbox for text analysis, it is important to note that not every aspect can be covered in detail due to the scope of this book. For those seeking a thorough understanding of all available text analytics options, the MathWorks documentation offers a rich collection of resources and examples that explore the extensive functionalities of the Text Analytics Toolbox, along with its integration with other toolboxes. It provides practical examples that illustrate applications such as sentiment analysis, topic modelling, the creation of word embeddings, text summarisation, keyword extraction, measuring document similarity etc. (cf. also Chapter 10.5).

## 10.2 Extracting text data from files or from the web

In MATLAB, you can create text variables and import text data using several core functionalities which we have already covered in Chapter 5. One additional easy way to import text data into MATLAB is to use the `extractFileText` function, which is part of the Text Analytics Toolbox. This function extracts the text data from various file types, including text, Microsoft Word, PDF and HTML files.

Let us assume you want to work with transcripts of speeches from Anthony Albanese, the Prime Minister of Australia, and you have a `.txt` file with the title `Speech_Australian_PM.txt` that contains his speech from 04/09/2025 given at the Street Side Medics fundraising gala in Sydney (cf. <https://pmtranscripts.pmc.gov.au/release/transcript-46917>).

If your text file with the speech is saved in your current working directory in MATLAB, you can directly use its name in one of the following commands to create either a string array or a character vector for the filename:

```
filename = "Speech_Australian_PM.txt";
```

or

```
filename = 'Speech_Australian_PM.txt';
```

Alternatively, if the file is located in a different directory, you can specify the full path to the file, for instance:

```
filename = "/Users/Katrin/Desktop/Speech_Australian_PM.txt"
```

Then extract the text from your file by using the `extractFileText` function for your MATLAB variable `filename` as in:

```
string_speech_PM = extractFileText(filename)
```

This creates a string variable `string_speech_PM` that contains the extracted text from the specified file (Figure 200).

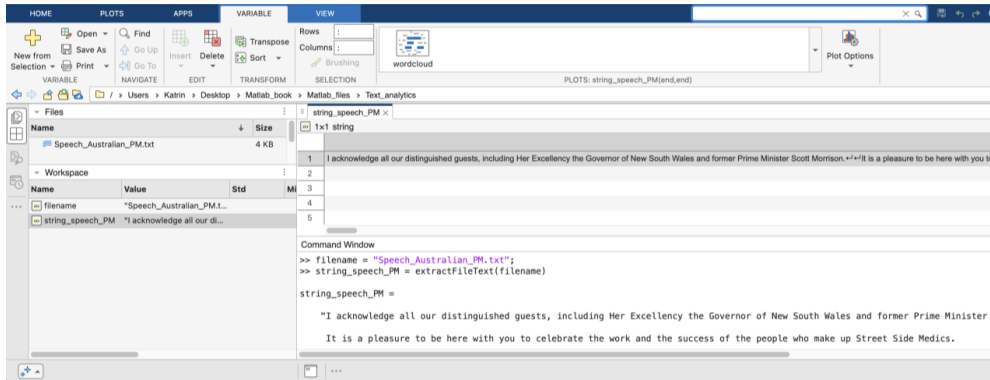


Figure 200: Extracting text from a .txt file

The `extractFileText` function works similarly with other file formats (cf. also <https://www.mathworks.com/help/textanalytics/ug/extract-text-data-from-files.html> and <https://www.mathworks.com/help/textanalytics/ref/extractfiletext.html>).

For instance, you may have a Microsoft Word document `Speech_Australian_PM.docx` that contains the above-mentioned speech. Then use the following commands to extract the text from the document:

```
filename = "Speech_Australian_PM.docx";
string_speech_PM = extractFileText(filename);
```

It works very similarly for PDF files, e.g., if your file is named `Speech_Australian_PM.pdf` use:

```
filename = "Speech_Australian_PM.pdf";
string_speech_PM = extractFileText(filename);
```

To extract text data from a saved HTML file, e.g., `Speech_Australian_PM.html`, use:

```
filename = "Speech_Australian_PM.html";
string_speech_PM = extractFileText(filename);
```

To extract text data from a string containing HTML code, use the function `extractHTMLText` (cf. also <https://www.mathworks.com/help/textanalytics/ref/htmltree.extracthtmltext.html>).

To extract text data directly from a web page, first read the HTML code using the function `webread`, and then use `extractHTMLText`, for instance:

```
url = "https://pmtranscripts.pmc.gov.au/release/transcript-46917";
code = webread(url);
string_speech_PM = extractHTMLText(code)
```

In this example `webread(url)` fetches the HTML content from the specified URL, and `extractHTMLText(code)` parses the HTML and extracts the text, which is then stored in `string_speech_PM` (Figure 201).

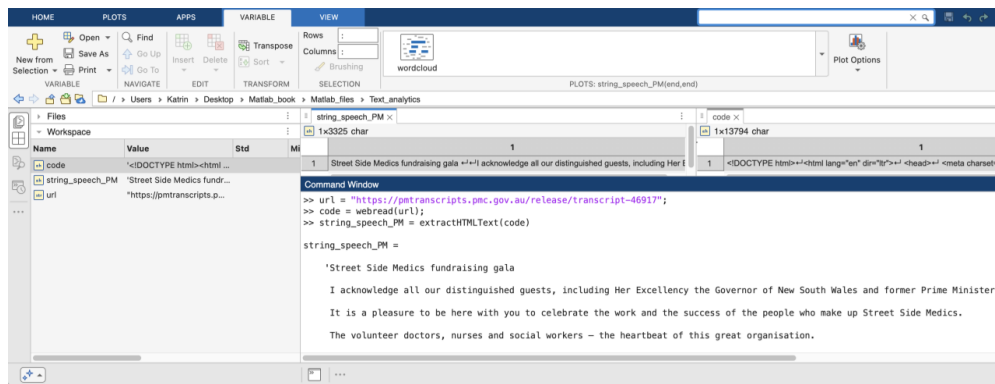


Figure 201: Extracting text from a web page

The `fileread` function, a core function in MATLAB (cf. Chapter 5.5), can be used to read the contents of a text file and return it as a character vector (<https://www.mathworks.com/help/matlab/ref/fileread.html>).

To import text from CSV and Microsoft Excel files, use MATLAB’s core function `readtable` (cf. <https://www.mathworks.com/help/matlab/ref/readtable.html>).

When you have PDF forms, you may use the `readPDFFormData` function from the Text Analytics Toolbox (cf. <https://www.mathworks.com/help/textanalytics/ref/readpdfformdata.html>).

Have a look also at the documentation under the following links where you can find an overview of various methods to import data in MATLAB, including text data:

### *Import text files*

[https://www.mathworks.com/help/matlab/import\\_export/ways-to-import-text-files.html](https://www.mathworks.com/help/matlab/import_export/ways-to-import-text-files.html)

### *Import text data into MATLAB*

<https://www.mathworks.com/help/textanalytics/gs/import-text-data.html>

## 10.3 Text preprocessing

Raw text data can be large and may contain various types of noise and elements we may want to standardise or remove. This can include case inconsistencies, such as “today” vs. “Today”, non-lemmatised words such as “run” and “running” and so-called “stop words” such as “and”, “in” or “at” that may add little value to the analysis of content words from the text. There may also be punctuation, special characters and HTML or XML tags embedded in the text, which may need to be removed. By applying preprocessing techniques from the toolbox, we can standardise and clean the data to ensure that only that information is retained which is relevant for our intended analysis. The Text Analytics Toolbox currently supports text analysis specifically in English, Japanese, German and Korean, but many functions also work with text in other languages. For more details on language support, have a look at:

*Language considerations*

<https://www.mathworks.com/help/textanalytics/ug/language-considerations.html>

*German language support*

<https://www.mathworks.com/help/textanalytics/ug/german-language-support.html>

*Japanese language support*

<https://www.mathworks.com/help/textanalytics/ug/japanese-language-support.html>

*Korean language support*

<https://www.mathworks.com/help/textanalytics/ug/korean-language-support.html>

*Language-independent features*

<https://www.mathworks.com/help/textanalytics/ug/language-independent-features.html>

Let us walk through an example of how to create a function that cleans and preprocesses text data by using the Preprocess Text Data task from the Live Editor. This task allows you to perform, for instance, HTML cleanup, tokenisation, word normalisation and the removal of specific types of words.

You can start by opening the Preprocess Text Data task in the Live Editor. Open a new live script (cf. Chapter 6.3) and select **Preprocess Text Data** in drop-down menu under **Task** in the Live Editor tab (Figure 202).

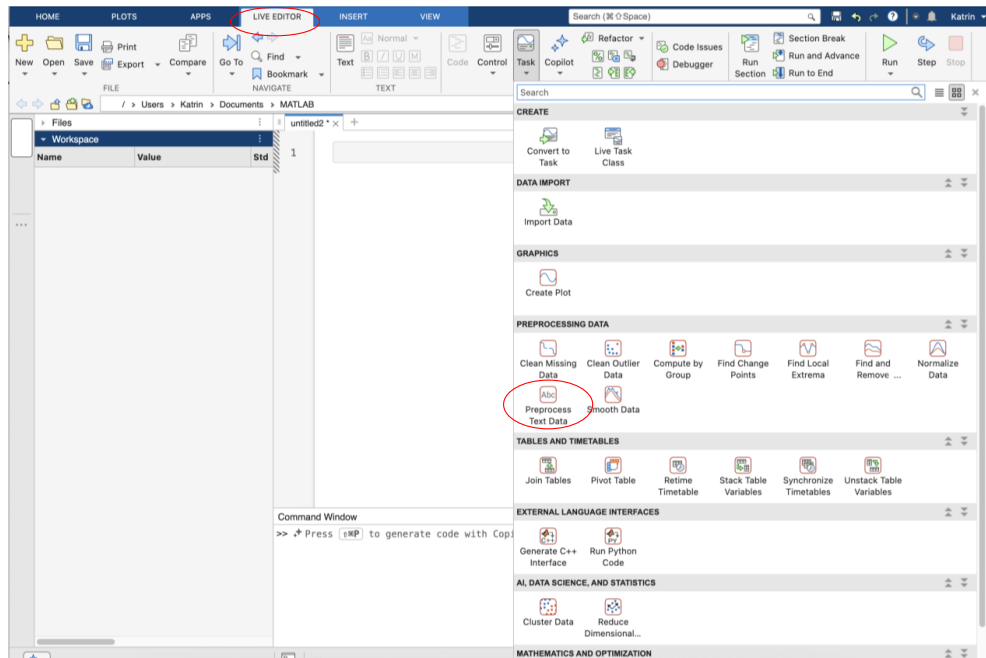


Figure 202: Opening the Preprocess Text Data task in the Live Editor

Alternatively, type a relevant keyword like “preprocess” or “text” in a code block of your live script and then select the Preprocess Text Data task from the suggested completions (cf. Figure 203).

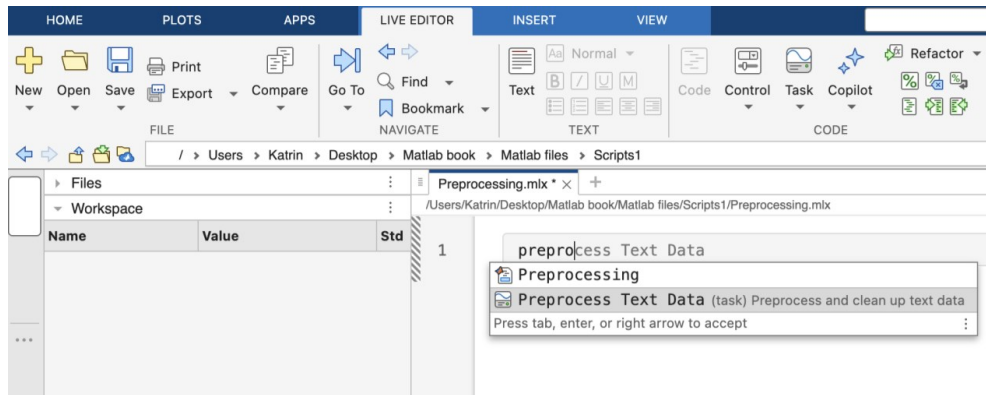


Figure 203: Opening the Preprocess Text Data task via a command

The Preprocess Text Data task provides a user-friendly interface that allows you to specify your input data and choose from various preprocessing options. For instance, we may select our string variable `textData` in the Preprocess Text Data task after having created it via the following commands:

```
filename = "Speech_Australian_PM.txt";
textData = extractFileText(filename);
```

In the Preprocess Text Data task, we may also decide to add part-of-speech tags and to detect named entities. We may lemmatise the text, apply case normalisation to lowercase and remove stop words as in the example in Figure 204.

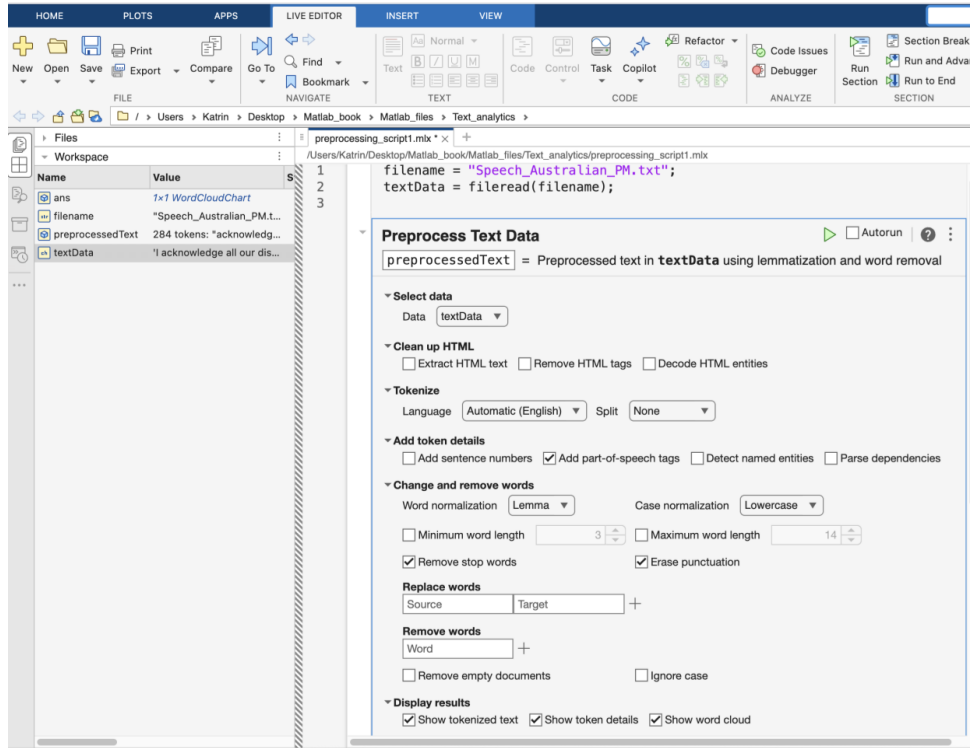


Figure 204: Using the Preprocess Text Data task in the Live Editor

We may want to display the results as a tokenised text with the respective token details and visualise the results also as a word cloud (cf. selection options shown in Figure 204). After configuring the task, run the live script. The results of this example are shown in Figure 205.



Figure 205: Results from the Preprocess Text Data task for a sample text

The Preprocess Text Data task generates code in the live script (Figure 206). The generated code reflects the options that were selected. In order to see the generated code, click **Show code** at the bottom of the task parameter area. When you click on **Hide code** above the code, the generated MATLAB code will be hidden from view.

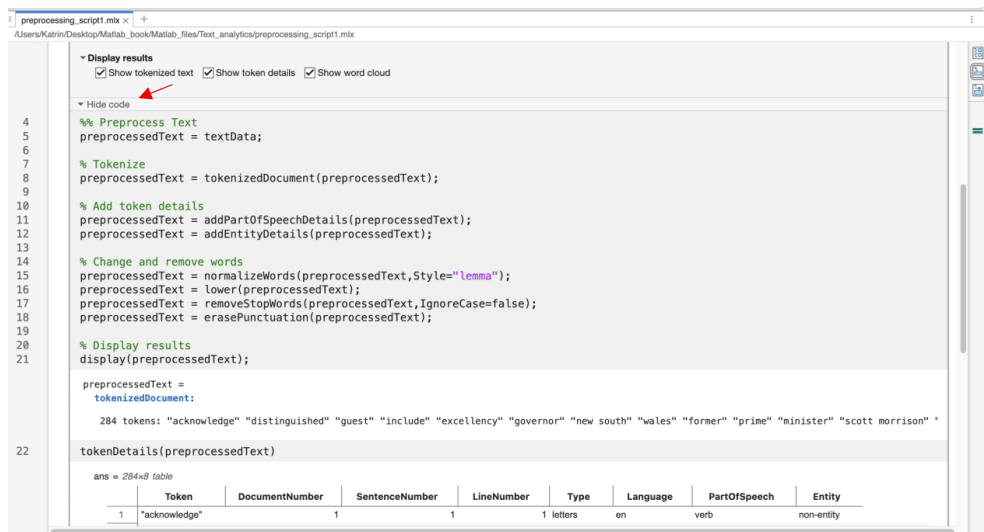


Figure 206: Displaying code generated in Preprocess Text Data task

In this task, you can also choose to add grammatical dependency details to the tokens by selecting the respective option under **Add token details** in the interface. To be able to do so, you must first download and install the “Text Analytics Toolbox Model for UDify Data”, which is available on the MathWorks File Exchange under <https://www.mathworks.com/matlabcentral/fileexchange/114705-text-analytics-toolbox-model-for-udify-data>. This pretrained deep learning model, built on BERT, is specifically designed for Universal Dependency annotation and can be used to extract dependency relationships between words in a sentence. Both the Text Analytics Toolbox and the Deep Learning Toolbox are required for its use. You can find information on the Deep Learning Toolbox at: <https://www.mathworks.com/help/releases/R2025a/deeplearning/>.

To install the Deep Learning Toolbox in MATLAB, click on the **Add-Ons** button in the toolstrip. In the Add-On Explorer then type “Deep Learning Toolbox” into the search bar. Once you find the toolbox, click on it, select the **Install** button and following any prompts to complete the installation process. You can verify whether this toolbox has been successfully installed by typing `ver` in the Command Window and checking for the Deep Learning Toolbox in the list of installed products or by checking in the Manage Add-Ons window lists which you can access via the Add-Ons button. Only when you have installed the Text Analytics Toolbox Model for UDify Data and the Deep Learning Toolbox, the option **Parse dependencies** can be selected under **Add token details** (Figure 207).

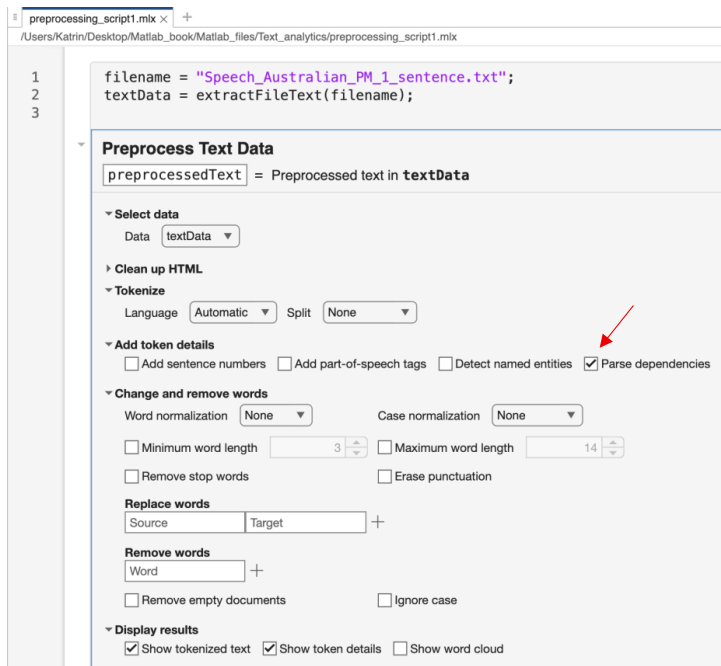


Figure 207: Selecting option to parse dependencies

In the example shown in Figure 207, a text document has been used that contains one sentence from the above-mentioned speech by the Australian Prime Minister Anthony Albanese (a speech from 04/09/2025 given at the Street Side Medics fundraising gala in Sydney, cf. <https://pmtranscripts.pmc.gov.au/release/transcript-46917> and Chapter 10.2). Figure 208 shows the code generated by the task and the details of the preprocessed text.

```

preprocessing_script1.mlx x +
/Users/Katrin/Desktop/Matlab_book/Matlab_files/Text_analytics/preprocessing_script1.mlx
Hide code
4 %% Preprocess Text
5 preprocessedText = textData;
6
7 % Tokenize
8 preprocessedText = tokenizedDocument(preprocessedText);
9
10 % Add token details
11 preprocessedText = addDependencyDetails(preprocessedText);
12
13 % Display results
14 display(preprocessedText);

preprocessedText =
tokenizedDocument:
26 tokens: It is a pleasure to be here with you to celebrate the work and the success of the people who make up Street Side Medics .

tokenDetails(preprocessedText)

ans = 26x8 table

```

	Token	DocumentNumber	SentenceNumber	LineNumber	Type	Language	Head	Dependency
1	"It"	1	1	1	1 letters	en	4	expl
2	"is"	1	1	1	1 letters	en	4	cop
3	"a"	1	1	1	1 letters	en	4	det
4	"pleasure"	1	1	1	1 letters	en	0	root
5	"to"	1	1	1	1 letters	en	6	mark
6	"be"	1	1	1	1 letters	en	4	csbj
7	"here"	1	1	1	1 letters	en	6	advmod
8	"with"	1	1	1	1 letters	en	9	case
9	"you"	1	1	1	1 letters	en	6	obl

Figure 208: Code generated by Preprocess Text Data task as specified in Figure 207 and details of preprocessed text

The `sentenceChart` function visualises the dependencies in a graphical format (Figure 209).<sup>34</sup> The `sentenceChart` function requires a single sentence as input. If you attempt to use it with a document or a variable containing multiple sentences, it will generate an error message. The function plots the grammatical dependency parse tree, where solid lines indicate dependencies and dotted lines represent subtree labels. Figure 209 shows the visualisation after a respective command has been added in the live script after the Preprocess Text Data task.

<sup>34</sup> cf. also <https://www.mathworks.com/help/textanalytics/ref/sentencechart.html> and <https://www.mathworks.com/help/textanalytics/ug/analyze-sentence-structure-using-grammatical-dependency-parsing.html>

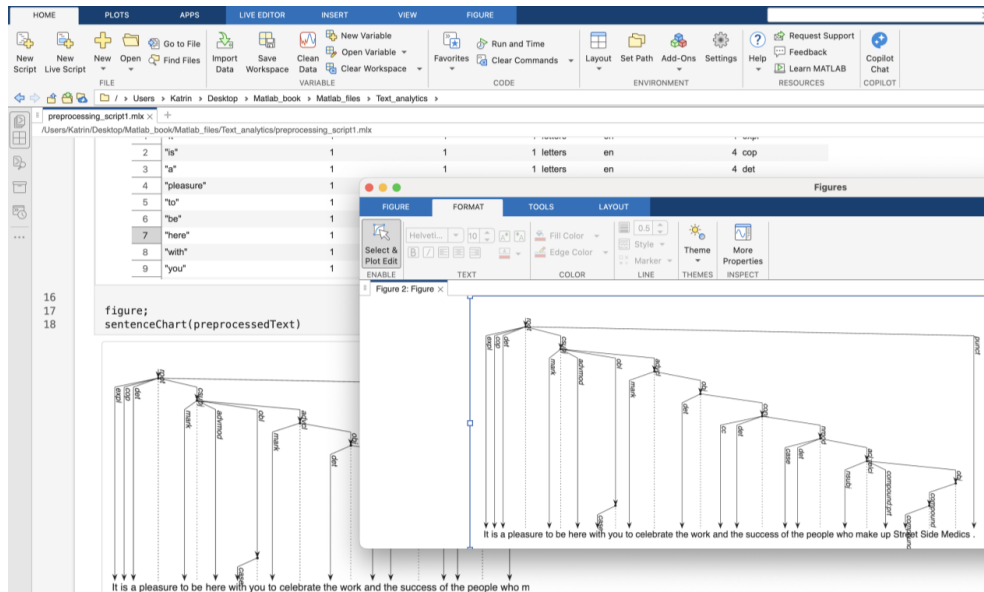


Figure 209: Grammatical dependency parse tree generated by using the `sentenceChart` function

There is also the `addDependencyDetails` function from MATLAB's Text Analytics Toolbox, which can be used to add grammatical dependency details to tokenised documents (cf. <https://www.mathworks.com/help/textanalytics/ref/tokenizeddocument.adddependencydetails.html>).

Generally, you can preprocess text data using either the predefined **Preprocess Text Data** Live Editor task as we have seen in the examples above or by executing commands directly in the Command Window. In the task, we have seen examples of the generated code for various options (Figure 206 and Figure 208).

The generated code might, for instance, look like this:

```
%% Preprocess Text
preprocessedText2 = textData;

% Clean up HTML
preprocessedText2 = eraseTags(preprocessedText2);

% Tokenize
preprocessedText2 = tokenizedDocument(preprocessedText2);

% Add token details
preprocessedText2 = addPartOfSpeechDetails(preprocessedText2);
preprocessedText2 = addEntityDetails(preprocessedText2);

% Change and remove words
preprocessedText2 = normalizeWords(preprocessedText2, Style="lemma");
preprocessedText2 = lower(preprocessedText2);
preprocessedText2 = removeShortWords(preprocessedText2, 2);
```

```

preprocessedText2 = removeLongWords(preprocessedText2,15);
preprocessedText2 = removeStopWords(preprocessedText2,IgnoreCase=false);
preprocessedText2 = erasePunctuation(preprocessedText2);
preprocessedText2 = removeEmptyDocuments(preprocessedText2);

% Display results
display(preprocessedText2);
wordcloud(preprocessedText2);

```

You can save the code generated by the Preprocess Text Data task as a script (e.g., `preprocessText.m`) and execute it whenever you need to preprocess similar text data. This will make it easy to apply the same preprocessing techniques to different datasets in the future.

In the following, an explanation is provided of some key commands and functions for preprocessing texts that may appear in the generated code and that you may wish to modify, delete or add if you use the code further in a script.

In the code above, for instance, the `eraseTags` function was used. This function removes HTML and XML tags such as `<title>` from the text. This will ensure that the text data is free from formatting artifacts that could interfere with your analysis.

The `tokenizedDocument` function converts raw text into a collection of tokens to allow you to work with individual words rather than entire sentences or paragraphs, cf. also <https://www.mathworks.com/help/textanalytics/ref/tokenizeddocument.html>. Here is an example:

```

textData = "It is a pleasure to be here with you to celebrate the work and
the success of the people who make up Street Side Medics.";

tokens = tokenizedDocument(textData);

```

The `tokens` variable now contains the tokenised representation of the input text, where each word and punctuation mark is treated as a separate item.

To add part-of-speech details after tokenisation use:

```
tokens_part_of_speech = addPartOfSpeechDetails(tokens);
```

To view the token details, you can use the following command (cf. also Figure 210):

```
tddetails = tokenDetails(tokens_part_of_speech)
```

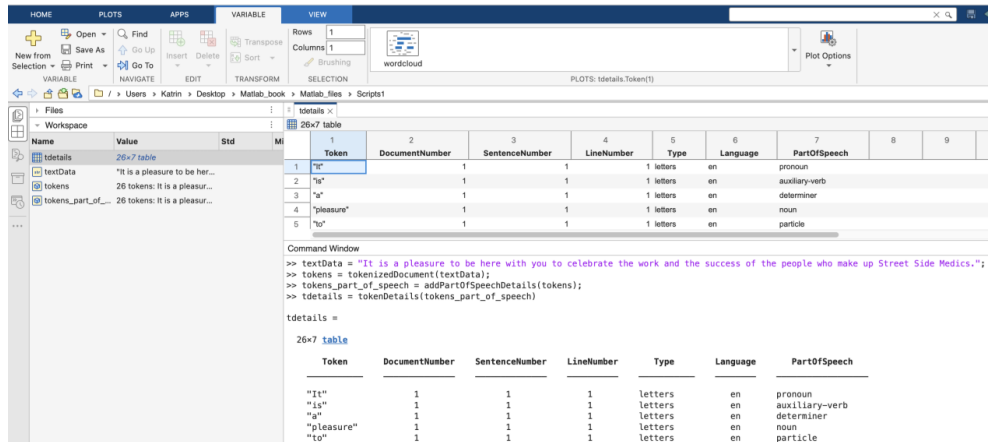


Figure 210: Displaying part-of-speech information with the `tokenDetails` function

In the table in Figure 210, we see information on the tokens, the document number, sentence number, line number, the type of the language of the text and the part-of-speech information.

In case the sentence number is not displayed, you may also use the `addSentenceDetails` function to add the sentence numbers to the table returned by `tokenDetails` (<https://www.mathworks.com/help/textanalytics/ref/tokenizeddocument.addsentencedetails.html>).

The Text Analytics Toolbox, specifically its `corpusLanguage` function, are designed to detect the language of text inputs. Currently English, Japanese, German and Korean are supported. In the table in Figure 210, we can see that English has been detected automatically as the language of the text. We may also use the `corpusLanguage` function to detect the language of textual data represented as strings. For example, we may have the following text, a sentence from a German translation of Shakespeare's *Macbeth*:

```
text = "Wann treffen wir drei uns das nächste Mal? Bei Regen, Donner, Wetterstrahl?"
```

Then we may use the following command:

```
detected_language = corpusLanguage(text)
```

This command will return `detected_language = 'de'`, which indicates that the detected language is German.

The `addEntityDetails` function from the toolbox identifies named entities (like people, organisations, locations) within texts. For instance, consider the following commands applied to a short excerpt from the above-mentioned speech by the Australian Prime Minister Anthony Albanese held on 04/09/2025 in Sydney:

```
textData = "I'd also like to acknowledge my remarkable friend, Dr Daniel Nour.";
```

```
tokens = tokenizedDocument(textData);
```

```
named_entities = addEntityDetails(tokens);
```

Here, the `tokenizedDocument` function splits the input text into tokens, which are stored in the variable `tokens`. These tokens generally correspond to individual words and punctuation marks in the text. The `addEntityDetails` function is then used to identify named entities within these tokens.

Retrieve the token details here as well by using the `tokenDetails` function.

```
tetails = tokenDetails(named_entities);
```

When you use the `addEntityDetails` function, multiword named entities can be recognised as a single named entity if the model identifies them as such. In this case, it recognises “Daniel Nour” as one entity (a person). However, the expression “Dr Daniel Nour” is not treated as a single entity. The title “Dr” is handled separately, as can be seen in the variable `tetails` (Figure 211).

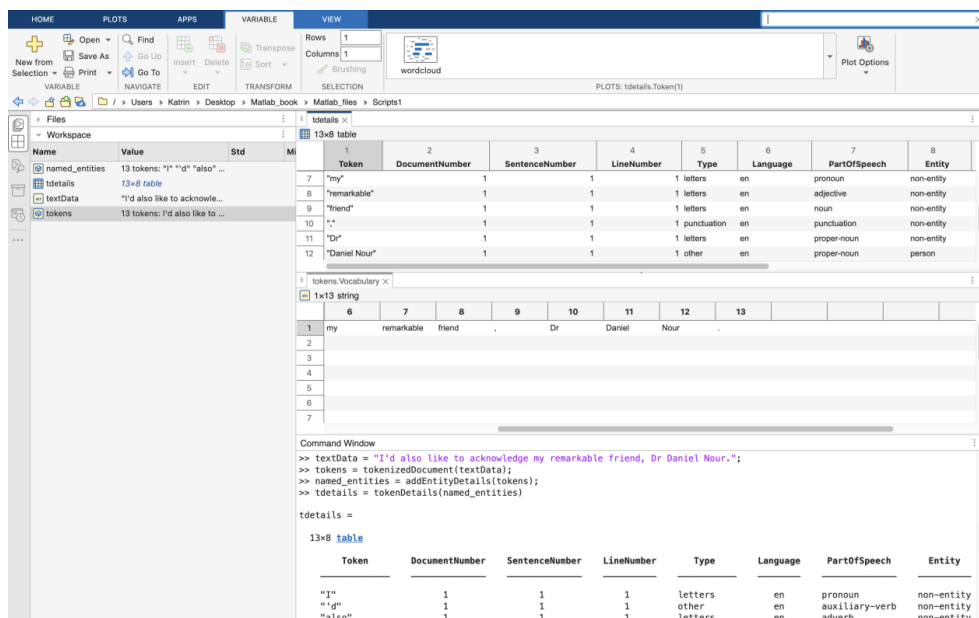


Figure 211: Identifying named entities and displaying results

Note that these preprocessing functions generally work really well, but they may sometimes not guarantee 100% accuracy from a linguistic perspective. Sometimes, users might also find that the preprocessing steps do not align perfectly with their specific requirements or preferences regarding certain details. The `addPartOfSpeechDetails` function, for instance, adds part-of-speech tags to documents by detecting parts of speech and updating token details, and this function works rather well for the supported languages (English, Japanese, German and Korean). However, the accuracy may still fluctuate depending on the complexity of the text and the context in which the function is used. Similarly, the `addEntityDetails` function identifies named entities, but you may also find that it has some limitation that arise from the underlying model’s training and the specific context of the text. For named entity recognition, you can also train your own model, cf. the following link:

*Train custom named entity recognition model*

<https://www.mathworks.com/help/textanalytics/ug/train-custom-named-entity-recognition-model.html>

Among the functions from the Text Analytics Toolbox, there is also the `normalizeWords` function which lemmatises the words from the text, i.e., it reduces the words to their base or dictionary form (e.g., “are” becomes “be” and “barriers” becomes “barrier”). The input text needs to be tokenised first before you can lemmatise it, and if you want to improve the accuracy of the lemmatisation, you may add part-of-speech details to the tokens before this step as in the following sequence of commands (summarised in a live script shown in Figure 212). The text example is taken from the above-mentioned speech by the Australian Prime Minister.

```
textData = "There are barriers for people on the street that span a lack of health education, distrust of the system and what must feel like endless practical troubles.";
```

```
tokens = tokenizedDocument(textData);
```

```
tokens = addPartOfSpeechDetails(tokens);
```

```
lemmatised_text = normalizeWords(textData, Style="lemma");
```

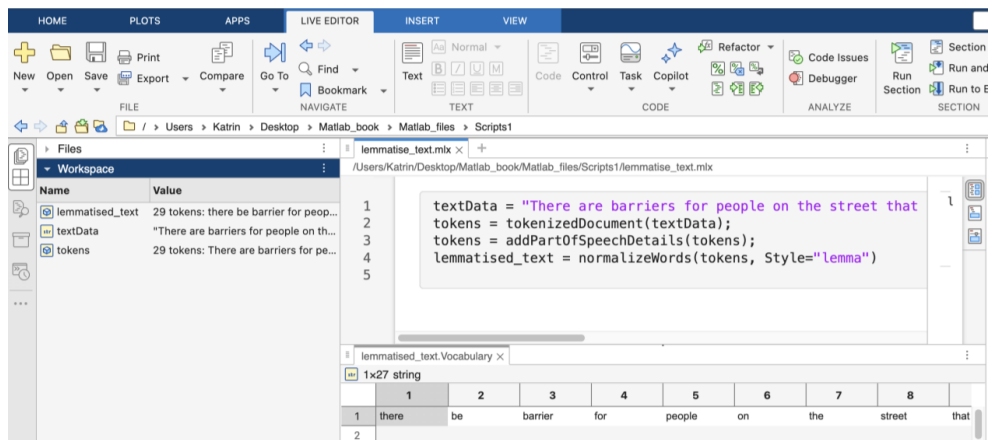


Figure 212: Script for lemmatising a text

The `lower` function converts all tokens to lowercase to prevent case sensitivity issues during your analysis:

```
lowerText = lower(textData);
```

If you are interested in converting your text to uppercase instead, you can use the `upper` function (cf. Figure 213):

```
upperText = upper(textData);
```

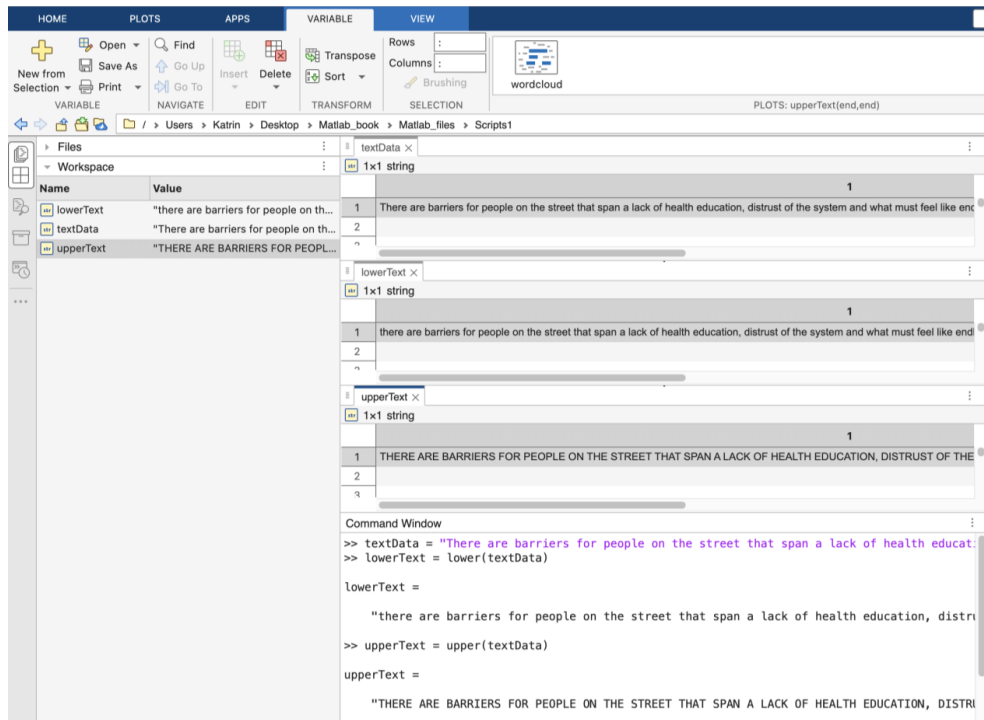


Figure 213: Converting tokens to lowercase and uppercase

Another function from the toolbox is `removeShortWords` which eliminates words with a specified length and shorter words. This can help remove noise from specific data sets. Use, for instance, the following command to remove words with 2 or fewer characters:

```
tokens = removeShortWords(tokens, 2);
```

Additionally, there is the `removeLongWords` function that removes words of a specified length and longer words that may not be relevant for certain analyses. Use, for instance, the following command to remove words with 15 or more characters:

```
tokens = removeLongWords(tokens, 15);
```

To remove punctuation marks, you may use the `erasePunctuation` function for which we have already seen some examples in this book, e.g., in Chapter 5.2.3. Use, for instance:

```
text_without_punctuation = erasePunctuation(textData);
```

The `removeStopWords` function removes common stop words from your textual data. To view the list of default stop words for different languages, use the `stopWords` function and specify the language as an argument. The function returns stop word lists for the supported languages. The following command will show you the list of English stop words (cf. also Figure 214):

```
EnglishStopWords = stopWords('Language', 'en')
```

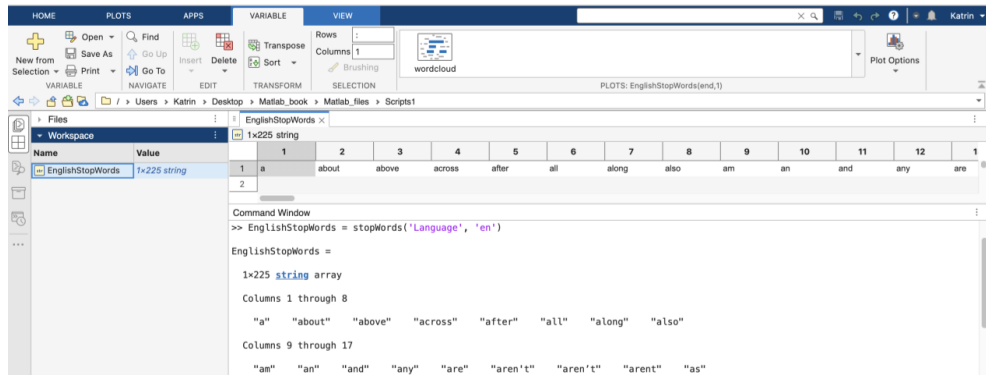


Figure 214: Displaying list of English stop words

The following command will show you the list of German stop words (cf. also Figure 215):

```
GermanStopWords = stopWords('Language', 'de')
```

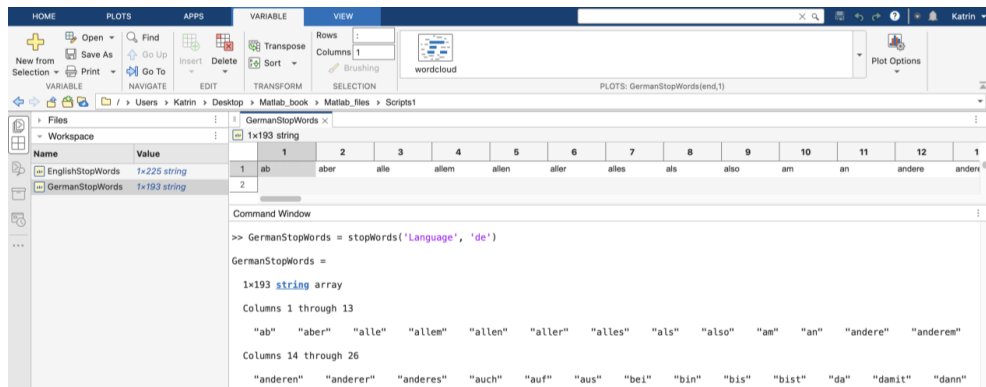


Figure 215: Displaying list of German stop words

By default, `removeStopWords(documents)` uses the predefined list of stop words based on the language of the documents and removes those tokens case-insensitively. If you want additional control, you can get the built-in list, define a list with additional entries and then apply `removeWords` to remove that combined list. Suppose we have a piece of text stored in a variable that contains words we want to remove as stop words, some of which are included in the default list and some of which are not:

```
textData = "There are barriers for people on the street that span a lack of health education, distrust of the system and what must feel like endless practical troubles. People often look around for support and are willing to try anything that might help.";
```

In this example, we want to remove the standard English stop words contained in the default list as well as the words "around" and "anything", which are not in this list. First, we obtain the default stop words for English. Next, we define the additional stop words we want to remove. Both lists are then converted to column vectors to ensure compatibility with MATLAB functions. After that, the default and custom stop words are combined into a single list. We then

tokenise the text data using `tokenizedDocument` and remove all stop words from the tokenised document with the `removeWords` function, as shown in the following series of commands (cf. also Figure 216):

```
% Get default English stop words
defaultStopWords = stopWords('Language', 'en');

% Create custom list of stop words with additional words to remove
customStopWords = ["around", "anything"];

% Ensure both are column vectors
defaultStopWords = defaultStopWords(:); % Convert to column vector
customStopWords = customStopWords(:); % Convert to column vector

% Combine lists
allStopWords = [defaultStopWords; customStopWords];

% Tokenise the text data
tokenised_document = tokenizedDocument(textData);

% Remove stop words
cleanedDocument = removeWords(tokenised_document, allStopWords)
```

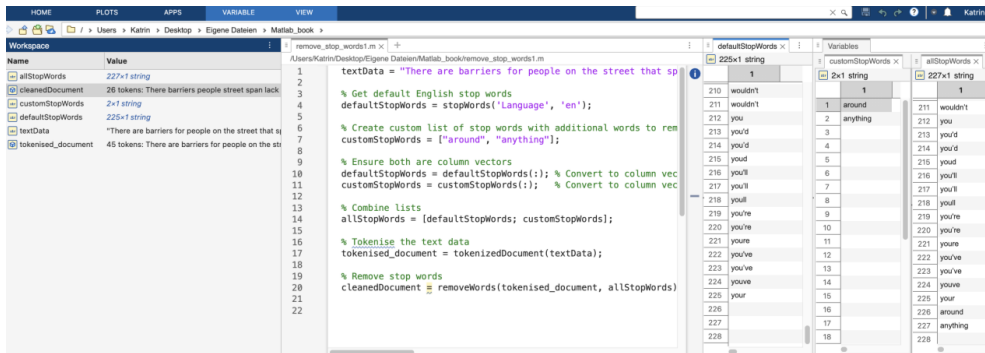


Figure 216: Removing customised list of stop words from text

Another function from the Text Analytics Toolbox is `splitSentences`, which is designed to divide text into individual sentences. To use this function, provide it with a string scalar or a `tokenizedDocument` object.<sup>35</sup>

In the following example, we read the content of a text file into a string by using the `extractFileText` function. The text file used in this example is a .txt file of *The call of the wild* text by Jack London (1903) downloaded from the Oxford Text Archive (OTA identifier: ota:3043, cf. <https://ota.bodleian.ox.ac.uk/repository/xmlui/handle/20.500.12024/3043>) and stored in the current working directory.

```
filename = 'Call_of_the_Wild.txt';
string_Call_of_the_Wild = extractFileText(filename);
```

Now we can split the text into sentences by using the `splitSentences` function (Figure 217):

```
sentences = splitSentences(string_Call_of_the_Wild);
```

<sup>35</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/tokenizeddocument.splitSentences.html>

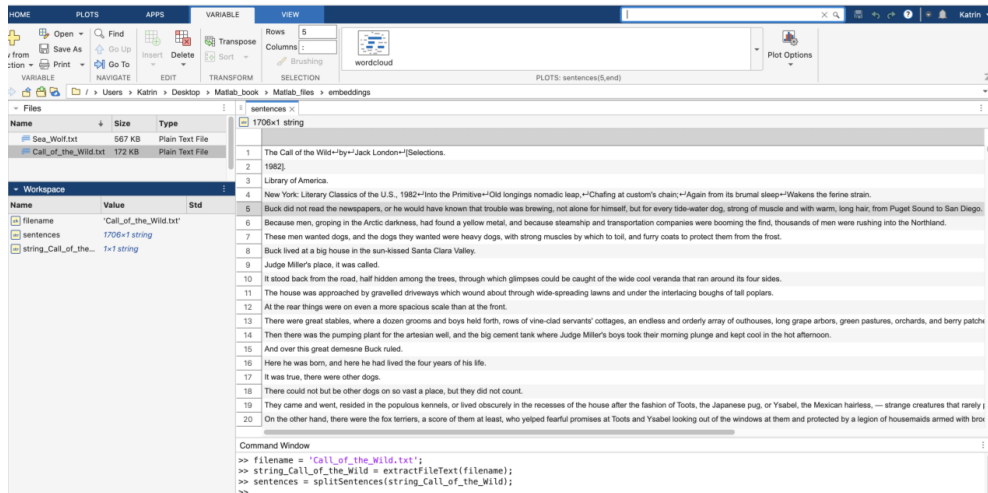


Figure 217: Splitting a text into sentences

To display the first sentences, e.g., the ten first sentences, in the Command Window, you may use the following command:

```
disp(sentences(1:10));
```

Note that the `splitSentences` function is designed to identify sentence boundaries based on punctuation and capitalisation. The function works generally well, but in a few cases, it may not accurately split individual sentences, e.g., when punctuation is used in unconventional ways or when generally infrequent abbreviations are used in the text. This means that some sentences may be split incorrectly, resulting in fragments or merged sentences. To mitigate issues with abbreviations, you can use the `addSentenceDetails` function to supply a list of specific abbreviations (e.g., start from a built-in list and add your domain-specific forms).

You can retrieve MATLAB's default list of nearly 400 common English abbreviations using the `abbreviations` function:<sup>36</sup>

```
table_English_abbreviations = abbreviations
```

There is also a predefined lists of almost 1,000 common abbreviations for German which you can retrieve by:

```
table_abbreviations_de = abbreviations('Language', 'de')
```

MATLAB's default lists of abbreviations, which can be retrieved using the `abbreviations` function, are taken into account by the `sentenceSplit` function. You can also create a custom list of abbreviations tailored to your needs, for example when working with specific English or German texts. For example, abbreviations like "Admis.," "Admtg.," and "Assn.," which are, for instance, used at UC San Diego (cf. abbreviation list under <https://blink.ucsd.edu/sponsor/blink/resources/aa.html>), may not be part of MATLAB's default list, but they might appear in specialised texts under analysis.

<sup>36</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/abbreviations.html>

The following line creates an array of custom abbreviations that you may want the sentence splitting algorithm to recognise in addition to the default list:

```
customAbbreviations = ["Admis" "Admtg" "Assn"];
```

By including these abbreviations, you will prevent the algorithm from splitting sentences incorrectly at periods that are part of these abbreviations. After creating such an array of custom abbreviations, extract the text from your file or define it directly as a string variable and create a tokenised document by using the `tokenizedDocument` function. Then use the `addSentenceDetails` function to incorporate your custom abbreviations. Finally, the `splitSentences` function can be called on the updated `tokenizedDocument` object, which now also includes the custom abbreviation details. The series of commands may look like this:

```
% Define custom abbreviations

customAbbreviations = ["Admis" "Admtg" "Assn"];

% Extract text from file with extractFileText function or define text
directly as string

string_Administrative_text = "The Admis. team will be reviewing the
applications for the Admtg. session next week. Once the evaluations are
complete, the final list will be shared with the Assn. members for
feedback.";

% Create a tokenised document

tokenised_text = tokenizedDocument(string_Administrative_text);

% Add sentence details to tokenised document with abbreviations (overwrite
existing variable tokenised_text with new data without creating new
variable name)

tokenised_text = addSentenceDetails(tokenised_text, 'Abbreviations',
customAbbreviations);

% Split text into sentences

sentences = splitSentences(tokenised_text)
```

The result will be:

```
sentences =

    2×1 tokenizedDocument:

    17 tokens: The Admis . team will be reviewing the applications for the
Admtg . session next week .
    20 tokens: Once the evaluations are complete , the final list will be
shared with the Assn . members for feedback .
```

Without defining the list, the result of the `splitSentences` function would have been:

```
sentences =

    5×1 tokenizedDocument:

     3 tokens: The Admis .
    10 tokens: team will be reviewing the applications for the Admtg .
```

```
4 tokens: session next week .  
16 tokens: Once the evaluations are complete , the final list will be  
shared with the Assn .  
4 tokens: members for feedback .
```

To explore the available text data preparation and preprocessing options in MATLAB with the Text Analytics Toolbox, you may also check out the following resources:

*Text data preparation*

<https://www.mathworks.com/help/textanalytics/text-data-preparation.html>

*Preprocess text data*

<https://www.mathworks.com/help/textanalytics/ref/preprocesstextdata.html>

*Preprocess text data in Live Editor*

<https://www.mathworks.com/help/textanalytics/ug/preprocess-text-data-in-live-editor.html>

*Prepare text data for analysis*

<https://www.mathworks.com/help/textanalytics/ug/prepare-text-data-for-analysis.html>

## 10.4 Examples of text analysis procedures using the Text Analytics Toolbox™

### 10.4.1 Creating a bag-of-words model

The bag-of-words model is a widely used text representation technique in Natural Language Processing and one of the simplest methods for converting text into a numerical format. It transforms text into a vector model, where each document is represented by a vector of word counts, to capturing the frequency of individual words while disregarding their order (cf. <https://www.mathworks.com/discovery/bag-of-words.html>).

The bag-of-words model plays an important role for tasks such as text classification, sentiment analysis and topic modelling. At its core, it breaks down documents into individual words and counts how often each word appears. The order of the words does not matter in this case.

Consider, for instance, two short documents that consist of one sentence each:

Document 1: *“We will have dinner.”*

Document 2: *“She wanted to have breakfast.”*

The first step now would be to extract all the unique words from them. This means each word (word type) is counted only once, even if it appears in both documents. The unique words from both documents (ignoring case and punctuation) are:

```
we, will, have, dinner, she, wanted, to, breakfast
```

Now we count how many times each word appears in each document (Table 19):

Word	Document 1	Document 2
we	1	0
will	1	0
have	1	1
dinner	1	0
she	0	1
wanted	0	1
to	0	1
breakfast	0	1

Table 19: Word frequency counts for two sample documents

Once we have counted how often each unique word appears in each document, the next step is to convert this information into a vector. A vector in this context is simply a list of numbers where each position in the vector corresponds to a specific word from the overall vocabulary (the unique word list we built) and each number represents how many times that word appears in the document. This means that the order of words in the original sentence no longer matters – what matters is whether the words appear and how many times.

To represent a document as a vector, we first use the vocabulary we built earlier, which contains all the unique words across both documents: [we, will, have, dinner, she, wanted, to, breakfast].

This vocabulary acts like a template or index system. Each position in the vector of a document corresponds to one of these words. Now, let us take Document 1 as an example. We check how many times each vocabulary item appears in this document. Putting these counts together gives the following vector for Document 1 as a numerical summary of the document: [1, 1, 1, 1, 0, 0, 0, 0]. Document 2 can be represented as [0, 0, 1, 0, 1, 1, 1, 1] to reflect the word counts in it. The Bag-of-Words model is generally more effective with larger datasets, where word frequency patterns are more stable and provide more meaningful insights. For smaller datasets, it can still highlight key terms and their occurrences, but the results may be less robust due to limited data.

You can use function `bagOfWords` from MATLAB's Text Analytics Toolbox to create a bag-of-words model from your text documents. Let us have a look at a small illustrative example. Suppose we have a collection of documents and want to build a bag-of-words model directly from them. As a starting point, we can define our documents as a string array. In the example below, we will use three poems by Shakespeare as our sample documents.

```
three_documents = [
    "All the world's a stage, And all the men and women merely players;
    They have their exits and their entrances, And one man in his time plays many
    parts, His acts being seven ages. At first, the infant, Mewling and puking
    in the nurse's arms. Then the whining schoolboy, with his satchel And shining
    morning face, creeping like snail Unwillingly to school. And then the lover,
    Sighing like furnace, with a woeful ballad Made to his mistress' eyebrow.
    Then a soldier, Full of strange oaths and bearded like the pard, Jealous in
    honor, sudden and quick in quarrel, Seeking the bubble reputation Even in the
    cannon's mouth. And then the justice, In fair round belly with good capon
    lined, With eyes severe and beard of formal cut, Full of wise saws and modern
    instances; And so he plays his part. The sixth age shifts Into the lean and
    slippered pantaloone, With spectacles on nose and pouch on side; His youthful
    hose, well saved, a world too wide For his shrunk shank, and his big manly
    voice, Turning again toward childish treble, pipes And whistles in his sound.
    Last scene of all, That ends this strange eventful history, Is second
```

childishness and mere oblivion, Sans teeth, sans eyes, sans taste, sans everything."

"Over hill, over dale, Thorough bush, thorough brier, Over park, over pale, Thorough flood, thorough fire, I do wander everywhere, Swifter than the moone's sphere; And I serve the fairy queen, To dew her orbs upon the green: The cowslips tall her pensioners be; In their gold coats spots you see; Those be rubies, fairy favours, In those freckles live their savours: I must go seek some dew-drops here, And hang a pearl in every cowslip's ear."

"Look in thy glass, and tell the face thou viewest Now is the time that face should form another; Whose fresh repair if now thou not renewest, Thou dost beguile the world, unbless some mother. For where is she so fair whose unear'd womb Disdains the tillage of thy husbandry? Or who is he so fond will be the tomb Of his self-love, to stop posterity? Thou art thy mother's glass, and she in thee Calls back the lovely April of her prime: So thou through windows of thine age shall see Despite of wrinkles this thy golden time. But if thou live, remember'd not to be, Die single, and thine image dies with thee."

```
];
```

Then we tokenise the documents:

```
documents = tokenizedDocument(three_documents);
```

We should also remove stop words and erase punctuation as these elements can otherwise dominate the model, and they would, for instance, distort a word cloud visualisation.

```
documents = removeStopWords(documents);
```

```
documents = erasePunctuation(documents);
```

We can simply overwrite the original variable `documents` in these steps of text preprocessing, but we could also assign the processed documents to different variable names.

Now we may proceed with creating a bag-of-words model from the tokenised and cleaned documents (cf. also <https://www.mathworks.com/help/textanalytics/ref/bagofwords.html>):

```
bag = bagOfWords(documents);
```

Once you have created a bag-of-words model, there are several analyses and operations you can perform to gain insights from your text data. You can also continue processing your text data by applying techniques such as topic modelling, sentiment analysis or classification. Under the following link, for instance, you can find instructions on how to train a simple text classifier using word frequency counts derived from a bag-of-words model: <https://www.mathworks.com/help/textanalytics/ug/create-simple-text-model-for-classification.html>.

An easy way to visualise your bag-of-words model is by plotting a word cloud. To visualise the most frequent words from your bag-of-words model, use:

```
wordcloud(bag)
```

This will plot the word cloud directly from the bag of words. Words displayed in bigger fonts and in orange are the most dominant and frequent ones in the bag-of-words model (Figure 218).



```
bag_frequent_words = removeInfrequentWords(bag, 2, 'IgnoreCase', true);
```

This command removes all words that appear two times or fewer in the bag, independent of their case. Such functionality is particularly valuable in text analysis when dealing with inconsistent capitalisation, as often found in user-generated texts or documents with heterogeneous formatting.

To add documents to an existing bag-of-words model, you can use the `addDocument` function (cf. <https://www.mathworks.com/help/textanalytics/ref/bagofwords.adddocument.html>). The syntax for this function is as follows:

```
newBag = addDocument(bag, documents);
```

This command adds the specified documents to the existing bag and results in an updated bag-of-words model stored in `newBag`.

To load and prepare the data for your bag-of-words model in MATLAB, you can also read your text data from a file and then split it into individual documents. Your input text file can contain various types of content such as a collection of news articles, product reviews or abstracts from academic papers. The input text file should be structured so that each document or relevant segment of text is separated by a newline character. First, load the text data from a file that you have saved in your current working directory, e.g.:

```
filename = "abstracts.txt";
```

Then read the text data from your file with the `extractFileText` function. It reads the content of the specified file and stores it as a string variable.

```
str = extractFileText(filename);
```

Now split the text into individual documents by using the `split` function. The following command divides the string `str` into individual segments based on newline characters:

```
textData = split(str, newline);
```

Now tokenise the data:

```
documents = tokenizedDocument(textData);
```

Optionally, apply further preprocessing steps if desired. For example, you might want to remove stop words or punctuation marks.

Then create your bag-of-words model:

```
bag = bagOfWords(documents);
```

If your text data is stored in multiple files within a folder, you can import it into MATLAB using a file datastore. This is a specialised object used to manage large collections of files. You can create a file datastore, read each file, tokenise the text and then add each document to your bag-of-words model. For detailed instructions on how to do this, please refer to the MATLAB documentation on the following topics:

*Add documents to bag-of-words or bag-of-n-grams model*

<https://www.mathworks.com/help/textanalytics/ref/bagofwords.adddocument.html>

*Datastore with custom file reader*

<https://www.mathworks.com/help/matlab/ref/matlab.io.datastore.filedatastore.html>

In case you already have a predefined list of unique words and their corresponding frequency counts, you can also create a bag-of-words model using the `bagOfWords` function directly with these lists. For example, let us assume you have the following unique words and their counts from three documents (Document 1: “*The cat sat on the mat.*”, Document 2: “*The dog chased the cat.*”, Document 3: “*A cat and a dog played together.*”):

```
uniqueWords = ["the", "cat", "dog", "sat", "on", "mat", "chased", "and",
               "played", "together"];

counts = [
    2, 1, 0, 1, 1, 1, 0, 0, 0, 0; % Document 1
    2, 1, 1, 0, 0, 0, 1, 0, 0, 0; % Document 2
    0, 1, 1, 0, 0, 0, 0, 1, 1, 1 % Document 3
];
```

Here, `uniqueWords` is a string vector or cell array containing the unique words, and `counts` is a matrix of integers representing the frequency counts of these words. You can now create the bag-of-words model as follows:

```
bag = bagOfWords(uniqueWords, counts);
```

Once you have constructed a bag-of-words model, you can analyse the frequency of each word across your documents. However, not all words will contribute equally to the analysis. Common words tend to appear frequently across many documents, which diminishes their usefulness in distinguishing one document from another. To enhance the meaningfulness of our text representation, we may employ the Term Frequency-Inverse Document Frequency (TF-IDF) method.

TF-IDF consists of two components: Term Frequency (TF) and Inverse Document Frequency (IDF). Term Frequency measures how often a word appears in a specific document relative to the total number of words in that document. For example, if a word appears 3 times in a document of 100 words, its TF would be  $3/100 = 0.03$ . In contrast, Inverse Document Frequency evaluates the importance of a word across the entire corpus. Words that appear in many documents are considered common and carry little information about any specific document, while words that appear in only a few documents are considered rare and carry more distinctive information. IDF is calculated as the logarithm of the total number of documents divided by the number of documents containing the term.

By multiplying TF and IDF, we combine these two complementary measures of a word’s importance and derive a score that highlights words that are frequent in a particular document but rare across the entire dataset. This scoring mechanism ensures that common words are down-weighted, while words that provide meaningful and unique insight into a document’s content are emphasised. Using IDF alone would ignore a word’s relevance within individual documents, and using TF alone would give too much weight to common, uninformative words.

You can generate a TF-IDF matrix from your bag-of-words model by using the `tfidf` function from the Text Analytics Toolbox:<sup>37</sup>

```
M = tfidf(bag);
```

The resulting matrix provides a representation of the importance of each word in the context of your documents. The size of the matrix will depend on the number of documents and the number of unique terms. For example, if you have 3 documents and 205 unique terms, the matrix will be  $3 \times 205$ . The matrix is typically sparse, which means that many entries will be zero. This is because not every term appears in every document. The output will indicate the number of nonzero entries, which reflects the terms that are present in the documents. For instance, if you run the command and view the first few entries of the matrix, you might see something like this for the above-mentioned small sample dataset consisting of three Shakespeare poems (preprocessed with stop word and punctuation removal):

```
M =
3x205 sparse double matrix (211 nonzeros)

(1,1)      1.0986
(1,2)      1.0986
(1,3)      1.0986
(1,4)      1.0986
(1,5)      1.0986
(1,6)      1.0986
(1,7)      1.0986
(1,8)      1.0986
(1,9)      1.0986
(1,10)     0.4055
(3,10)     0.8109
(1,11)     2.1972
...
```

The entries in the matrix represent the TF-IDF scores for each term in each document. A higher score indicates that a term is more significant in that document relative to its occurrence in the entire corpus. For instance,  $(1,1) \ 1.0986$  indicates that the first term has a TF-IDF score of 1.0986 in the first document. The repeated score of 1.0986 for several terms in the first row suggests that these terms are equally significant in that document. The entry  $(3,10) \ 0.8109$  shows the score of the tenth term in the third document to indicate its significance there. The value  $(1,11) \ 2.1972$  indicates that the eleventh term has a TF-IDF score of 2.1972 in the first document. This relatively high score suggests that this term is particularly significant in the context of the first document, possibly indicating that it appears frequently in that document while being less common across the other documents.

You can use `bag.Vocabulary` to retrieve the terms associated with the columns of the TF-IDF matrix. This allows you to see which terms correspond to specific TF-IDF scores (Figure 219):

```
terms = bag.Vocabulary
```

<sup>37</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/bagofwords.tfidf.html>

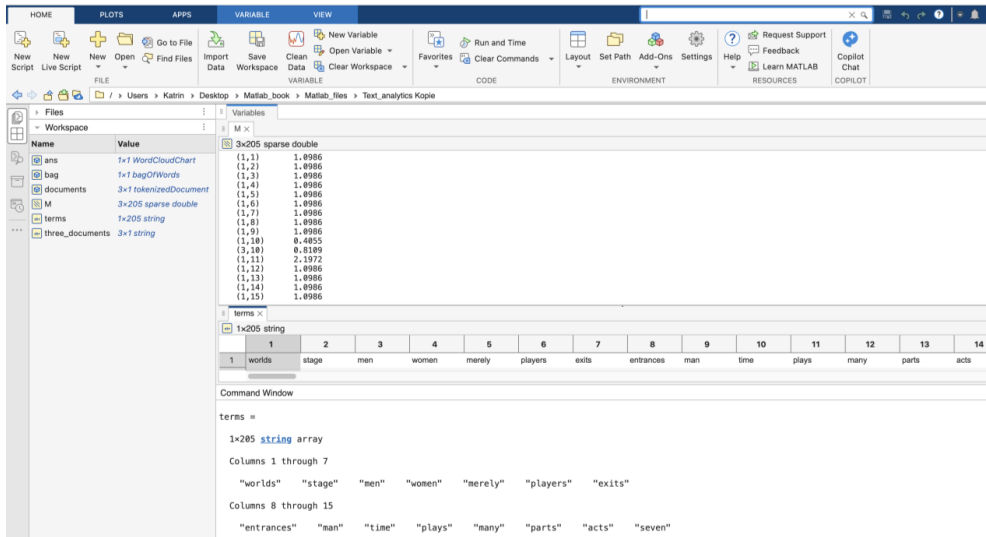


Figure 219: Retrieving terms associated with columns of TF-IDF matrix

Overall, the bag-of-words model is relatively simple and easy to implement by using the respective functions from the Text Analytics Toolbox, and it may be a good starting point for text analysis. However, the method has some limitations as it ignores the order and context of words and cannot capture meaning that comes from word sequences. Moreover, it often produces very large and sparse vectors with many zeros, especially when working with large vocabularies. Additionally, it cannot recognise semantic similarity (e.g., the words “*father*” and “*son*” are unrelated in a bag-of-word model). To overcome these limitations, methods such as bag-of-n-grams or word embeddings can be applied. These approaches will be explained in the next sections.

### 10.4.2 Creating a bag-of-n-grams model

A bag-of-n-grams model is a text representation technique that counts the occurrences of n-grams (sequences of n successive items) in text documents (cf. <https://www.mathworks.com/discovery/ngram.html>). This model captures the frequency of each unique n-gram (e.g., 2-grams / bigrams such as “ice cream”, 3-grams / trigrams like “New York City” and 4-grams / four-grams such as “United States of America”) in a dataset. N-grams can be multiword expressions or meaningful phrases, but they may also simply represent sequences of frequently co-occurring words. The bag-of-n-grams model provides more contextual information than simpler models like the bag-of-words model, which only counts individual words. By capturing sequences of words, the bag-of-n-grams model can enhance performance in tasks such as sentiment analysis, text generation and text classification, where sequences of words and the relationships between words are particularly relevant. To create a bag-of-n-grams model in MATLAB, you can use the `bagOfNGrams` function from the Text Analytics Toolbox.<sup>38</sup> This function works similarly to the `bagOfWords` function, but instead of counting individual words, it counts n-grams.

<sup>38</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/bagofngrams.html> and <https://www.mathworks.com/help/textanalytics/ug/analyze-text-data-using-multi-word-phrases.html>

For instance, if you have a file `reviews.txt` containing customer reviews which is stored in your current working directory, you can read it as follows:

```
filename = "reviews.txt";  
  
str = extractFileText(filename);
```

This `.txt` file should contain one review per line. This format allows the function to read each line as a separate document which can then be tokenised for further analysis. The `extractFileText` function reads the entire text from the file as a single string `str`. Then use `split(str, newline)` to divide the string into individual segments based on newline characters:

```
textData = split(str, newline);
```

This creates a cell array where each element corresponds to an individual review from the dataset.

Then tokenise the data:

```
documents = tokenizedDocument(textData);
```

This converts the cell array into a tokenised document array which is suitable for further text analysis and for creating the bag-of-n-grams model. Optionally, you may want to preprocess the text data with additional steps. For example, you may want to remove punctuation marks and stop words by using the following commands.

```
documents = erasePunctuation(documents);  
  
documents = removeStopWords(documents);
```

Now use the `bagOfNgrams` function to create the model and specify the n-gram length. For instance, for bigrams, you can write:

```
bag = bagOfNgrams(documents, 'NGramLengths', 2);
```

You can then use the `wordcloud` function to generate a visual representation of the most common bigrams from the dataset:

```
figure;  
  
wordcloud(bag);
```

The function displays the selected n-grams (bigrams in this case) in different sizes. The size of each n-gram corresponds to its frequency in the dataset, and the most prominent n-grams are emphasised with a highlight colour.

When you create a bag-of-n-grams model, it captures the unique n-grams in a string array, which can be accessed as follows:

```
uniqueNgrams = bag.Ngrams;
```

This will give you a string array containing all the unique n-grams present in the model.

You can also retrieve the most frequent n-grams using the `topkngrams` function. The following command, for instance, will retrieve the top 10 most frequent n-grams and their frequency values.

```
top_n_grams = topkngrams(bag, 10);
```

Figure 220 illustrates the output of a script that processes a collection of academic abstracts stored in a text file `abstracts.txt`. The script first loads the text data and splits these into individual documents. Next, the documents are tokenised, and punctuation is removed. A bag-of-n-grams model is then constructed to capture trigrams, i.e., sequences of three consecutive words. From this model, a word cloud visualisation is generated to provide an intuitive view of the most frequent trigrams across the dataset. Finally, the script extracts and displays the top eight most frequent trigrams from the model. Figure 220 includes both the script itself and the resulting output.

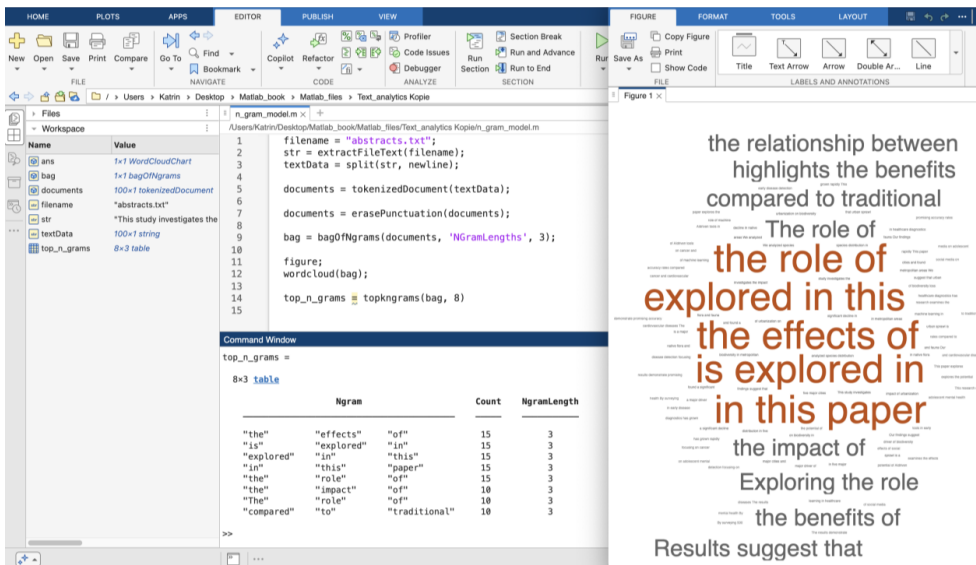


Figure 220: Creating a bag-of-n-grams model

When stop words are removed in addition to punctuation, the text will be reduced to mostly content words, which can make the resulting trigrams more topic-specific. Figure 221 presents the output, including both a word cloud of the most frequent trigrams and a list of the top eight trigrams extracted from the collection of abstracts after stop word removal. For improved readability, the maximum number of n-grams displayed in the word cloud has been reduced from the default value of 100 to 20 by using the options in the Property Inspector (for details on configurable properties cf. also <https://www.mathworks.com/help/matlab/ref/matlab.graphics.chart.wordcloudchart-properties.html>).

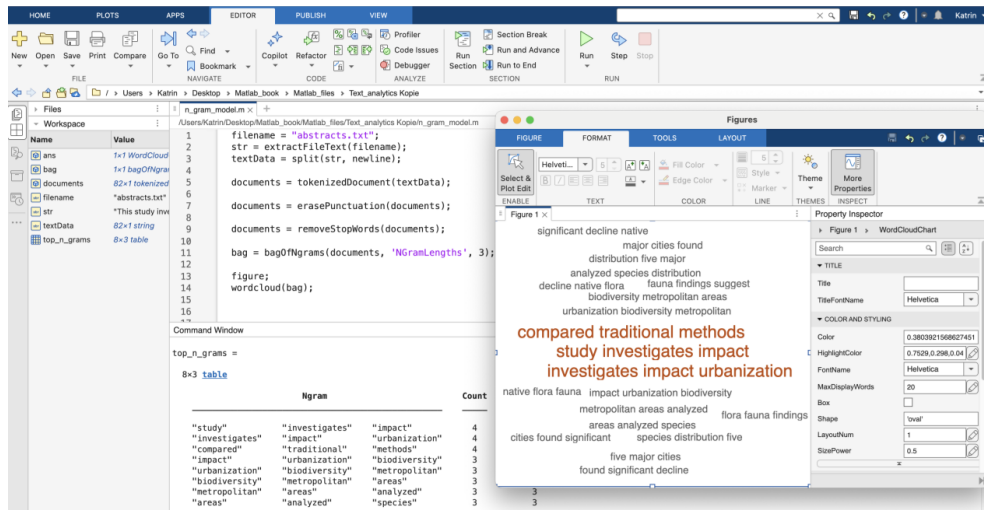


Figure 221: Most frequent trigrams extracted from bag-of-n-grams model after stop word removal from dataset

To remove specific n-grams from your bag-of-n-grams model in MATLAB, you can use the `removeNgrams` function.<sup>39</sup> First, define the n-grams you want to remove, and then call the function as shown below:

```

ngrams = ["study" "investigates" "impact"
          "investigates" "impact" "urbanization"];

bag = removeNgrams(bag, ngrams);

```

If you also want to ignore case sensitivity during removal, you can include the `'IgnoreCase', true` option:

```

bag = removeNgrams(bag, ngrams, 'IgnoreCase', true);

```

This will delete all instances of the n-grams you want to remove, regardless of how they are capitalised, for example when the word “impact” appears as “Impact”, “impact” or “IMPACT” in the n-grams.

As an alternative, consider applying a preprocessing step that converts all text to lowercase (cf. Chapter 10.3), to ensure that n-grams differing only by case will be treated as identical.

The `removeInfrequentNgrams` function is used to remove n-grams from a bag-of-n-grams model that appear infrequently, based on a specified count threshold. This function operates similarly to the `removeInfrequentWords` function for the bag-of-words model (cf. Chapter 10.4.1).<sup>40</sup>

Overall, the bag-of-n-grams model is useful for tasks where capturing short sequences of words provides more context than single-word features alone. Compared to the bag-of-words model, its main advantages include richer contextual information and the flexibility to choose different

<sup>39</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/bagofngrams.removengrams.html>

<sup>40</sup> see also <https://www.mathworks.com/help/textanalytics/ref/bagofngrams.removeinfrequentngrams.html>

n-gram lengths. However, the model also has some limitations. It can generate very large and sparse feature spaces as  $n$  increases, it may be less effective on small datasets, and it cannot capture long-range dependencies between words. Nevertheless, with careful data preprocessing steps, the bag-of-n-grams approach remains a practical and useful tool for many natural language processing tasks.

### 10.4.3 Working with pretrained word embeddings

In text analysis, word embeddings are an important technique for representing textual data in a numerical form that captures semantic meaning. MATLAB's Text Analytics Toolbox provides functionalities for working with word embeddings. You can use the `fastTextWordEmbedding` function to load a pretrained model.<sup>41</sup> The pretrained model is based on the `fastText` algorithm, which has been trained on a large corpus of 16 billion tokens of English text. The embeddings map words to high-dimensional real-valued vectors so that semantically similar words have similar vector representations and are located close to each other in the embedding space. These representations are useful in various natural language processing tasks, including text classification, clustering and similarity analysis. For more information on the pre-trained word vector model see: <https://fasttext.cc/docs/en/english-vectors.html>. Keep in mind that word embeddings can consume a significant amount of memory on your computer.

To use the pretrained `fastText` model, the required support package, i.e., the “Text Analytics Toolbox Model for `fastText` English 16 Billion Token Word Embedding” support package, must be installed. You can find it via the Add-On Explorer (cf. also <https://www.mathworks.com/matlabcentral/fileexchange/66229-text-analytics-toolbox-model-for-fasttext-english-16-billion-token-word-embedding>).

To load the pretrained `fastText` model, use the following command:

```
embeddings = fastTextWordEmbedding;
```

This line initialises the `embeddings` object, which contains the word vectors. It loads the pretrained `fastText` word embedding model, which provides 300-dimensional vector representations for approximately 1 million English words. Once the model is loaded, you can convert words into their vector representations using the `word2vec` function. You can retrieve the vector for individual words and then identify their closest neighbours in the embedding space.

To retrieve the vector representation of a word, e.g., the word “frogmen”, use the `word2vec` function.<sup>42</sup>

```
frogmenVec = word2vec(embeddings, "frogmen");
```

The resulting vector captures the semantic meaning of the word in a high-dimensional space. If the word is not found in the embedding vocabulary, the function will return a row of NaN values. Note that the `word2vec` function is case-sensitive by default. If you want to ignore case when retrieving the vector, you can use the `IgnoreCase` option:

```
frogmenVec = word2vec(embeddings, "frogmen", 'IgnoreCase', true);
```

<sup>41</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/wordembedding.html> and <https://www.mathworks.com/help/textanalytics/ref/fasttextwordembedding.html>

<sup>42</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/wordembedding.word2vec.html>

You may use the `isVocabularyWord` function to check whether a given word is in the vocabulary of your `embeddings` object, e.g.:

```
check = isVocabularyWord(embeddings, "frogmen")
```

This will return a logical value of 1 (true) if “frogmen” is in the vocabulary and 0 (false) if it is not.

You can also test multiple words at the same time using the `isVocabularyWord` function (cf. <https://www.mathworks.com/help/textanalytics/ref/wordencoding.isvocabularyword.html>).

Define the words to check, e.g.:

```
words = ["submarine", "vessel", "frogmen"];
```

Then check if the words are in the vocabulary:

```
check = isVocabularyWord(embeddings, words);
```

To identify the closest words to a given word such as “frogmen” in the embedding space, you can use the `vec2word` function.<sup>43</sup> For example, to find the top ten closest words, use:

```
closestWords = vec2word(embeddings, frogmenVec, 10)
```

If you want to find the top five closest words instead of ten, set the value in this command to 5 (cf. Figure 222):

```
closestWords = vec2word(embeddings, frogmenVec, 5)
```

---

<sup>43</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/wordembedding.vec2word.html>

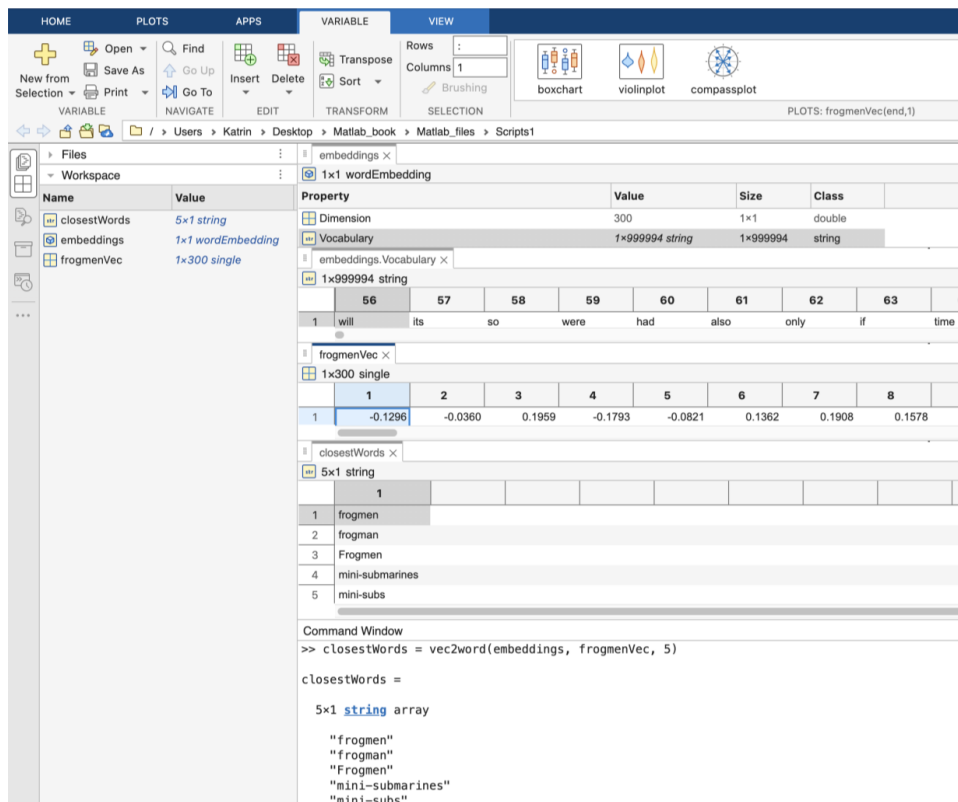


Figure 222: Finding closest semantic neighbours to a given word

The result in this case is a 5×1 string array that contains terms which are semantically related to “frogmen” and share contextual meaning within similar environments, such as diving or underwater operations (“mini-submarines” and “mini-subs”). Note that the array also contains word forms related to “frogmen” (e.g., “frogman”) as these forms also often appear in similar contexts. Additionally, the word “frogmen” which is the exact word queried in this case is also included in the results as its vector representation is the closest to itself.

You can explore relationships between words using vector arithmetic. For example, if you have words such as “submarine” and “warship” in your dataset and you want to find words related to these words, you can add the vectors of “submarine” and “warship” to create a new vector that captures the combined meaning of both words. This new vector can then be used to find other words that are semantically related by measuring the proximity of their vectors in the embedding space. To do so, first, use the above-mentioned `fastTextWordEmbedding` function to load the pretrained embedding model:

```
embeddings = fastTextWordEmbedding;
```

Then use the `word2vec` function to obtain the vector representations of “submarine” and “warship”:

```
submarineVector = word2vec(embeddings, "submarine");
```

```
warshipVector = word2vec(embeddings, "warship");
```

Now compute a new vector that combines the meanings of both words:

```
relatedVector = submarineVector + warshipVector;
```

You can then use the `vec2word` function to find the words that are closest to the resulting vector. You can specify how many words you want to retrieve (for example, ten) as follows:

```
[relatedWords, distances] = vec2word(embeddings, relatedVector, 10)
```

The `relatedWords` variable will now contain the top ten words that are semantically related to the combination of “submarine” and “warship” (Figure 223). This may again include various forms of the lemmas “submarine” and “warship” such as plural forms and the queried words themselves. The optional `distances` variable inserted in the command above provides the numerical distances of the words from the computed vector, which can help you to understand their relevance. The distances are typically computed using a metric such as Euclidean distance or cosine similarity. A smaller distance closer to 0 indicates a closer semantic relationship, while a larger distance suggests a weaker connection.

```
Command Window
>> submarineVector = word2vec(embeddings, "submarine");
>> warshipVector = word2vec(embeddings, "warship");
>> relatedVector = submarineVector + warshipVector;
>> [relatedWords, distances] = vec2word(embeddings, relatedVector, 10)

relatedWords =

    10×1 string array

    "submarine"
    "warship"
    "submarines"
    "warships"
    "ship"
    "frigate"
    "ships"
    "battleship"
    "vessel"
    "submersible"

distances =

    10×1 single column vector

    0.0837
    0.0845
    0.1944
    0.2282
    0.2664
    0.2851
    0.2977
    0.3019
    0.3070
    0.3261
```

Figure 223: Displaying semantically related words and proximity values in embedding space derived from combination of two words

Now, let us say you want to subtract another word vector from the combined vector representations of “submarine” and “warship”, for example, “destroyer”. You would first obtain the vector for “destroyer”:

```
destroyerVector = word2vec(embeddings, "destroyer");
```

Then, you can compute the new vector by subtracting the “destroyer” vector from the above-mentioned combined vector that we named `relatedVector`:

```
resultingVector = relatedVector - destroyerVector;
```

The resulting vector now represents a semantic space that captures the essence of “submarine” and “warship” while excluding characteristics specifically associated with “destroyer” among the closest semantic neighbours. You can display the results again with the `vec2word` function as in the following command which will result in the top twenty closest words:

```
[relatedWords, distances] = vec2word(embeddings, resultingVector, 20);
```

The resulting top words, including, for instance, “ship”, “undersea”, “navel” and “vessel”, indicate a focus on vessels and naval terminology. In contrast, you can also specifically add (instead of subtracting) “destroyer” to the combined vector as follows:

```
resultingVector = relatedVector + destroyerVector;
```

The presence of “destroyer” now narrows the focus to a specific class of naval vessels that are semantically close to the words “submarine” and “warship”. Now the top results will include terms like “destroyer”, “frigate”, “battleship” and “cruiser”, which are all types of warships that may destroy or engage in combat.

The MATLAB Text Analytics Toolbox provides tools for visualising word embeddings by using dimensionality reduction techniques such as t-distributed Stochastic Neighbour Embedding (t-SNE), cf. <https://www.mathworks.com/help/stats/t-sne.html>). This is particularly helpful when working with high-dimensional word vector data, as t-SNE can project these vectors for instance into a two- or three-dimensional space that can be visualised as a scatter plot (cf. <https://www.mathworks.com/help/textanalytics/ug/visualize-word-embedding-using-text-scatter-plot.html>). t-SNE works by calculating pairwise similarities between high-dimensional points and attempts to preserve these relationships in a low-dimensional space. It constructs a similarity matrix that represents the probability of points being neighbours. The optimisation process in t-SNE minimises the Kullback-Leibler divergence, which is a measure of how one probability distribution diverges from a second, expected probability distribution. In the context of t-SNE, it quantifies the difference between the probability distributions of the high-dimensional data and the low-dimensional representation. By minimising this divergence, t-SNE retains the structure of the original data in its lower-dimensional form.

In practice, this may involve loading pretrained word embeddings (e.g., using `fastTextWordEmbedding`), then extracting word vectors with `word2vec`, reducing their dimensions with the `tsne` function and displaying the results as a text scatter plot with the `textscatter` function (<https://www.mathworks.com/help/textanalytics/ref/textscatter.html>). Words that are used in similar contexts, and thus tend to be semantically related, will be projected closer together in the text scatter plot. This means you can observe semantic clusters and thematically related terms, which might otherwise be hidden. This is particularly useful for qualitative assessment of model behaviour and for uncovering latent patterns in language data.

Let us explore an example using the above-mentioned pretrained model. First, we load the pretrained word embedding. Then we may extract a set of words for visualisation. The number of words which we select to extract for visualisation, such as the first 1,000 or 5,000 words from the vocabulary, influences the clarity and interpretability of the resulting plot. Fewer words often yield clearer clusters, while more words provide broader coverage, but may complicate the visual representation of clusters as the plot will be characterised by increased visual complexity. Note that `fastText` word embeddings can include punctuation marks as tokens as `fastText` treats each token as a unique entity.

For example, if you want to visualise the first 5,000 words (or rather tokens) from the pretrained word embeddings, first load the model and then extract a set of 5,000 items for visualisation as follows:

```
embedding = fastTextWordEmbedding;  
words = embedding.Vocabulary(1:5000);
```

Then, we retrieve their vectors:

```
Vectors = word2vec(embedding, words);
```

To visualise the word embeddings, we need to reduce their dimensionality using the `tsne` function, which performs t-Distributed Stochastic Neighbour Embedding:

```
XY = tsne(Vectors);
```

The default output of the `tsne` function is two-dimensional. The resulting variable `XY` will be a matrix containing the two-dimensional coordinates of the word embeddings after applying t-SNE. Each row in this matrix corresponds to a word's new position in the 2D space.

Finally, we create a 2D text scatter plot with the following commands:<sup>44</sup>

```
figure;  
textscatter(XY, words);
```

Figure 224 shows the plot created for the word embeddings.

---

<sup>44</sup> cf. also <https://www.mathworks.com/help/textanalytics/ref/textscatter.html>

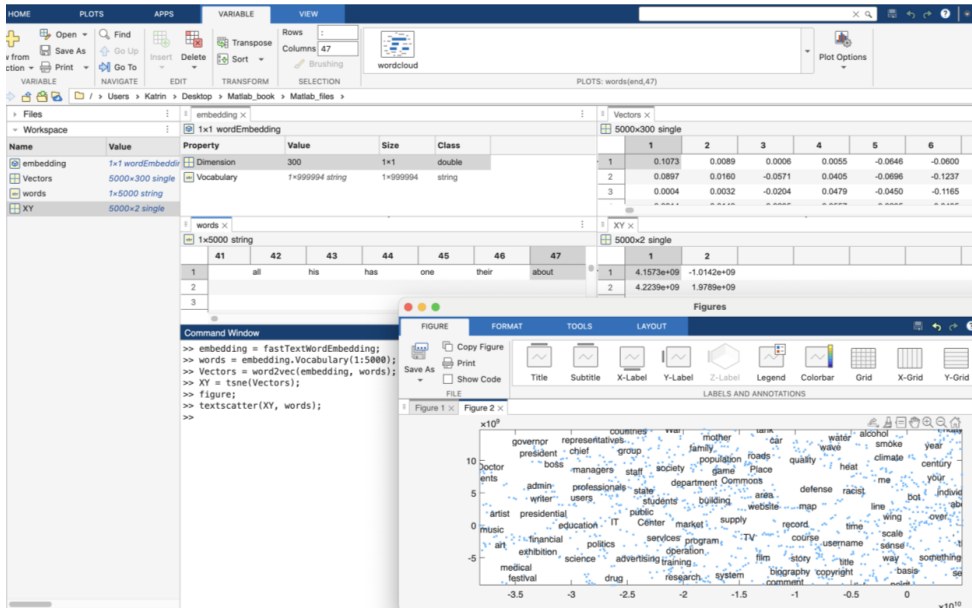


Figure 224: Creating a 2D text scatter plot for word embeddings

You can zoom in and out directly in the figure by scrolling the mouse wheel while hovering over the plot. You can also click and drag the plot to move around and explore different areas of the scatter plot. We may make further adjustments to the plot, e.g., adding a title via the interactive options in the figure window, the Property Inspector (cf. Chapter 7.2) or via a command, e.g.:

```
title('Visualisation of word embeddings for top 5000 words from fastText');
```

To use t-SNE and specify the number of dimensions to be different from two, you can use the `tsne` function with the `'NumDimensions'` name-value pair argument. You can set it to 3, which will allow you to visualise high-dimensional data in three dimensions:

```
XYZ = tsne(Vectors, 'NumDimensions', 3);
```

Due to the additional complexity, the `tsne` function can take a bit longer to run when you specify a different number of dimensions, such as three, compared to the default two dimensions.

To create a three-dimensional text scatter plot, you may now use:

```
figure;
scatter_plot = textscatter3(XYZ, words);
```

In the figure window created by the `textscatter3` function, you can interactively explore the 3D scatter plot. By hovering your mouse over the plot area, you will see that icons for functions such as rotation and zoom become visible (Figure 225).

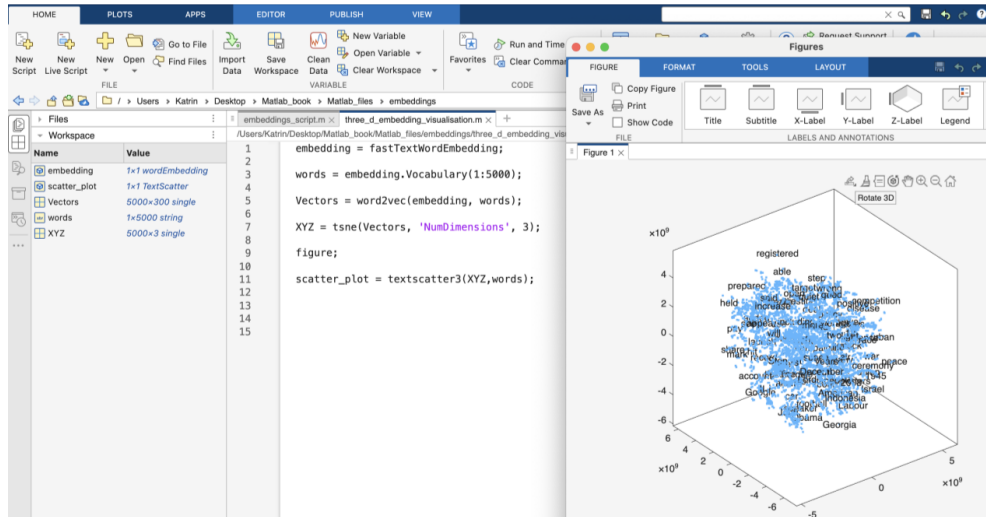


Figure 225: Creating a 3D text scatter plot for word embeddings

By using this three-dimensional visualisation, you may examine the spatial relationships between the words from different angles and observe clusters and relationships that might not be apparent in a 2D plot.

#### 10.4.4 Training your own word embeddings

You can train your own word embeddings using the `trainWordEmbedding` function, which allows you to customise the embeddings based on a specific dataset (<https://www.mathworks.com/help/textanalytics/ref/trainwordembedding.html>). When you train your own word embeddings, keep in mind that they require rather large and diverse text data as input to capture the semantic relationships between words effectively. The more varied the contexts in which a word appears, the better the model can learn its meaning.

As an example, let us train word embeddings based on two of Jack London's novels: *The call of the wild* (1903) and *The sea wolf* (1904). Both texts are available for download from the Oxford Text Archive (OTA), under the identifiers ota:3043 and ota:3045, respectively. For additional works by Jack London, you can browse the OTA collection (<https://ota.bodleian.ox.ac.uk/repository/xmlui/browse?value=London,%20Jack,%201876-1916&type=author>).

While this selected dataset is relatively small (ca. 300,000 words) compared to typical corpora used for embedding training, such as the fastText English 16 Billion Token dataset, the embeddings generated in this example will primarily serve illustrative purposes to demonstrate the steps for training word embeddings in MATLAB. We will have a look at the workflow of training embeddings from text files, reducing their dimensionality using t-SNE and visualising them with a text scatter plot.

First download the files and save them in your current working directory, then use the `extractFileText` function to read the contents of both `.txt` files separately.

```

filename1 = 'Call_of_the_Wild.txt';
string_Call_of_the_Wild = extractFileText(filename1);

```

```
filename2 = 'Sea_Wolf.txt';  
string_Sea_Wolf = extractFileText(filename2);
```

After extracting the text from both files, you can concatenate them into a single string. To do this, use the following command, which also adds a newline character between the two texts for better readability:

```
combinedText = [string_Call_of_the_Wild, newline, string_Sea_Wolf];
```

You may choose to split the combined text into sentences for further processing if your analysis requires understanding the text at the sentence level:

```
sentences = splitSentences(combinedText);
```

After splitting the text into sentences, you can then tokenise these sentences into a `tokenizedDocument` object:

```
tokenised_texts = tokenizedDocument(sentences);
```

Then use the `trainWordEmbedding` function on the combined text:

```
embeddings = trainWordEmbedding(tokenised_texts);
```

The trained word embedding object `embeddings` now contains the learned word vectors. Here, we have used a `TokenizedDocument` array as input. The `trainWordEmbedding` function can also accept a string or a cell array of strings, so you can choose to use the combined text directly if preferred.

If you want to specify parameters such as the dimension of the embeddings, you can do so by adding name-value pairs to the `trainWordEmbedding` function. For example, to set the dimension of the embeddings to 100, use:

```
embeddings = trainWordEmbedding(tokenised_texts, 'Dimension', 100);
```

The `'Dimension', 100` argument here determines how many features will represent each word. It specifies that the resulting word embeddings should have 100 dimensions. This means each word will be represented as a vector of 100 real-valued numbers. The choice of dimension can affect the model's ability to capture semantic relationships. Larger dimensions may capture more nuances but can also lead to overfitting if the dataset is small.

When using the `trainWordEmbedding`, you can specify several parameters in addition to the `Dimension` parameter to improve the performance of the model and the quality of the generated word embeddings for a specific dataset. One of the parameters is `MinCount`, which determines the minimum frequency of words to be included in the model. The default value for `MinCount` is 5. When using the `trainWordEmbedding` function without further specification, any word that appears fewer than five times in the training dataset will be discarded from the vocabulary used for training the word embeddings. For example, setting `'MinCount', 10` ensures that only words appearing at least ten times in the dataset are considered. This will filter out rarer words that may not contribute significantly to the learning process. Use, for instance the following command:

```
embeddings = trainWordEmbedding(tokenised_texts, 'MinCount', 10);
```

The `NumEpochs` parameter in the `trainWordEmbedding` function controls how many times the training algorithm goes through the entire dataset of text during training. Each complete pass through the dataset is called an epoch. The default value for `NumEpochs` is 5. Therefore, if you do not specify this parameter, the training will automatically run for five epochs. If you set `NumEpochs` to a higher number (for example, `NumEpochs, 10`), the model is given more opportunities to “see” the data. This repeated exposure helps it refine the word embeddings. For instance, if the word “man” frequently appears near “woman” in the dataset, multiple epochs allow the model to strengthen that connection and to produce embeddings that reflect semantic relationships between words in a better way. This is especially helpful when the dataset is large and diverse. However, if you use too many epochs, the model can start to overfit. This means it memorises the exact wording and specific features of the training data instead of learning general patterns. An overfitted model might perform very well on the training text because it has “memorised” it, but poorly on new, unseen text. For example, it might perfectly capture relationships within a small training sample of novels by Jack London but then fail to generalise well to other novels.

You can use the parameters `'Dimension'`, `'MinCount'` and `'NumEpochs'` in combination when calling the `trainWordEmbedding` function, e.g.:

```
embeddings = trainWordEmbedding(tokenised_texts, ...
    'Dimension', 100, ...
    'MinCount', 10, ...
    'NumEpochs', 10);
```

To retrieve the vocabulary from your trained word embedding, you can access the `Vocabulary` property of the `embeddings` object. This property contains the unique words that were included in the model during training. For instance, use the following command:

```
vocabulary = embeddings.Vocabulary;
```

This will return a string array `vocabulary` containing all unique words from the training data that were included in the model.

Below, some of the above-mentioned commands are brought together in a form that can be placed directly into a script.

```
% Read the files
filename1 = 'Call_of_the_Wild.txt';
string_Call_of_the_Wild = extractFileText(filename1);

filename2 = 'Sea_Wolf.txt';
string_Sea_Wolf = extractFileText(filename2);

% Combine the texts with a newline for better separation
combinedText = string_Call_of_the_Wild + newline + string_Sea_Wolf;

% Split the combined text into sentences
sentences = splitSentences(combinedText);

% Tokenise the sentences
tokenised_texts = tokenizedDocument(sentences);

% Train word embeddings with a specified dimension of 100
embeddings = trainWordEmbedding(tokenised_texts, 'Dimension', 100)
```

The result of running the script will be the processing of the two text files, which are combined, split into sentences and then tokenised into individual words. A word embedding model will be trained on the tokenised text, providing vector representations for each word. The respective variables will be available in the Workspace panel (Figure 226).

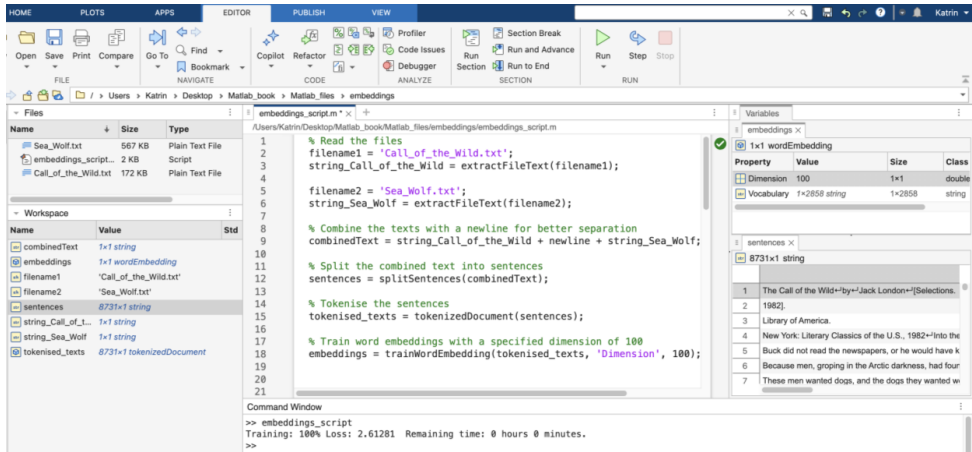


Figure 226: Training a word embedding model

When you run this series of commands, you may also see a message in the output in the Command Window such as:

```
Training: 100% Loss: 2.61281 Remaining time: 0 hours 0 minutes.
```

This shows that the training process has reached its completion. The training loss value, such as `Loss: 2.61281`, represents how well the word embedding model has learned to predict relationships between words based on the training data. A lower loss indicates better performance.

To evaluate the quality of your trained word embeddings, you may use dimensionality reduction techniques like t-SNE to visualise the embeddings. This helps in assessing how well the embeddings cluster similar words together. Extract the vocabulary and the corresponding vectors from your trained embeddings:

```
words = embeddings.Vocabulary;
V = word2vec(embeddings, words);
```

Then reduce the dimensions using t-SNE:

```
XY = tsne(V);
```

Now create a text scatter plot.

```
figure;
textscatter(XY, words);
```

Figure 227 is a text scatter plot of word embeddings trained on the above-mentioned two novels by Jack London with the commands summarised in the above-mentioned script. Each point

represents a word projected into two dimensions, with spatial proximity indicating semantic similarity. This means that words that appear closer together in the plot tend to occur in similar linguistic contexts within the novels.

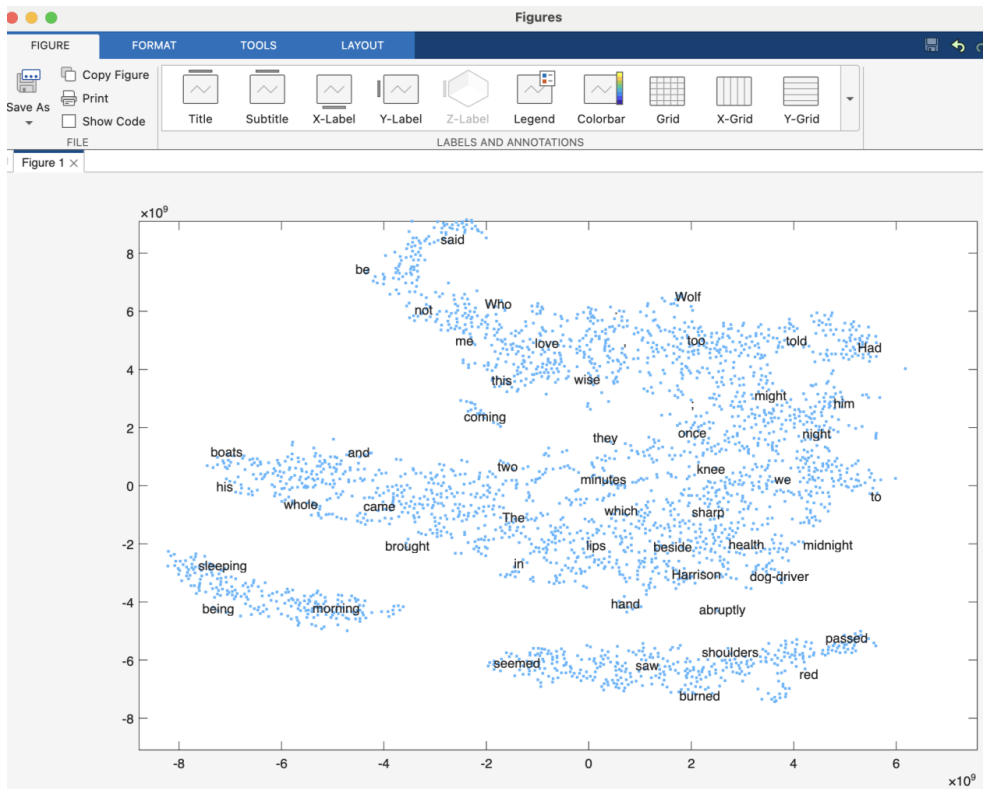


Figure 227: Text scatter plot visualising trained word embeddings

Figure 228 presents a zoomed-in view of the word embedding visualisation in the text scatter plot shown in Figure 227.

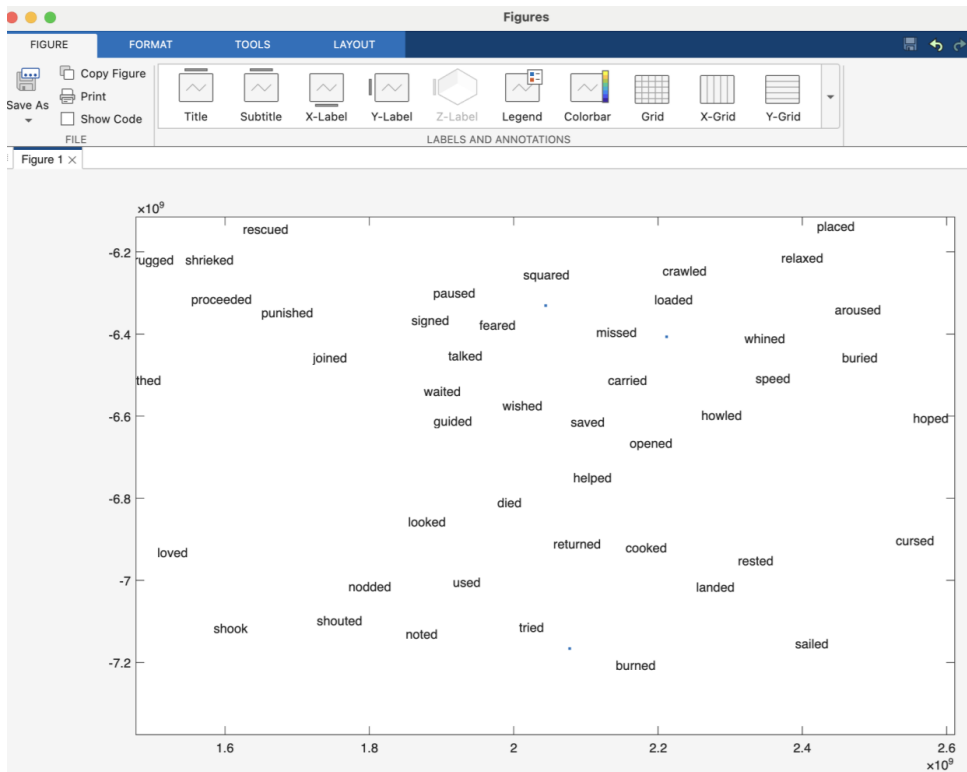


Figure 228: Zoomed-in region of text scatter plot

The noticeable clustering of verbs in past tense forms can be explained by the fact that within the novels, these verb forms frequently occur in similar contexts to describe past actions. Recurrent co-occurrence in these contexts reinforces their proximity, which is then reflected as clustering in the embedding space. As we can see in the figure that has resulted here in words like “paused”, “signed”, “feared”, “talked” etc. appearing close together, even though they are not semantically very similar. This effect can be explained by the relatively small size of the sample training dataset: with fewer contexts to draw from, the model has limited ability to capture fine-grained semantic distinctions.

In addition to inspecting text scatter plots to evaluate the quality of the embeddings you may also look for close neighbours (cf. also Chapter 10.4.3). After obtaining the vector for a word such as “man”, for instance, by using the `word2vec` function, find the nearest words to this vector using the `vec2word` function:

```
man_vector = word2vec(embeddings, "man");
[relatedWords, distances] = vec2word(embeddings, man_vector, 10)
```

When examining the nearest ten neighbours of “man” in our case, we may see words like “human”, “woman” and “sons” that are semantically related, but also less directly connected words from broader contexts such as “nature” or “moral” that may reflect broader thematic patterns where “man” is often discussed in relation to the natural world or ethics. There may be also some rather unrelated words among the closest neighbours such as “deny”. The presence

of both relevant and less relevant neighbours suggests that our example of trained embeddings is partially successful. The embeddings capture core semantic and thematic relationships, but their precision is constrained by the relatively small size of the corpus.

Use the `writeWordEmbedding` function for saving your trained word embeddings to a file. This allows you to persist the vocabulary and their corresponding vector representations in a format that can be easily reused or shared. The function writes the embeddings in UTF-8 format, specifically in the `word2vec` text format (`.vec` file format), which is widely recognised and can be used by various machine learning frameworks. By saving different versions of your embeddings, you can track changes and improvements over time. If `'embeddings'` is your trained word embedding object, then you may specify the filename and save the embeddings with these commands:

```
filename = "myWordEmbeddings.vec";  
  
writeWordEmbedding(embeddings, filename);
```

Before training your own word embeddings, you may also want to consider additional preprocessing steps to improve the quality of the embeddings. Common preprocessing steps may include removing stop words, reducing words to their base form by lemmatisation and removing punctuation. Keep also in mind if you extract text from files such as PDF without cleaning the data, there may be formatting issues, OCR errors or other forms of noise that can negatively impact the quality of the embeddings. The quality and interpretability of the result heavily depend on the dataset itself.

### 10.4.5 Generating summaries from text documents

If you want to generate a concise summary from an individual text or from a collection of texts, you may use the `extractSummary` function that generates summaries by selecting representative content from the input data.<sup>45</sup> More specifically, it simply selects a subset of the content or the documents in order to represent the overall content, and it returns the results in a `tokenizedDocument` array. This selection of content is based its relevance and importance within the overall dataset. Like many functions in the Text Analytics Toolbox, the `extractSummary` function primarily supports four different languages: English, German, Japanese and Korean.

When you call `extractSummary(documents)` for a set of different documents, the function typically chooses about one-tenth of the input documents and returns their entire content as a summary. Here is a small illustrative example using a few news headlines about species extinction as separate documents. The documents cover various aspects of species at risk, the impact of human activities on biodiversity and efforts to protect endangered species.

First, we define our data and then create an array of tokenised documents:

```
str = [  
    "Lost gecko species thought to be extinct rediscovered in South Africa  
    after more than 30 years"  
    "We know next to nothing about 99 per cent of the world's insects: Here's  
    why that's a problem"  
    "Shark and ray populations have halved since 1970 and overfishing is to  
    blame"
```

---

<sup>45</sup> cf. <https://www.mathworks.com/help/textanalytics/ref/extractsummary.html>

```

"More than 10,000 species on brink of extinction need urgent action:
Study"
"Biotech Firm Announces 'Pivotal Step' in Effort to Bring Back the Dodo"
"Scientists say they have resurrected the dire wolf"
"Over 75% of Earth's undescribed plants threatened by extinction, new
report warns"
"Plant extinction 'bad news for all species'"
"'The risk of extinction is accelerating': world's botanic gardens raise
alarm with space to protect endangered plants running out"
"Nearly half of the world's flowering plants face the threat of
extinction, study says"
"More than 1 in 3 tree species worldwide at risk of extinction. The loss
of trees is also a major threat to thousands of other plants, animals and
fungi."
"Charity plans to prevent local flower extinction - Guernsey's dwarf
pansy is now the focus of a conservation effort to prevent its extinction"
"Rarest plant flowers at secret nursery refuge. There are just four of
the dandelion-like Snowdonia hawkweed left in the wild."
"With 63% of Ireland's birds at risk of extinction, an Irish ornithologist
is utilising recordings he is making to advocate for their preservation"
];

```

```
documents = tokenizedDocument(str);
```

Then we extract a summary (i.e., a subset) of the documents which will be returned as a representative selection:

```
summary = extractSummary(documents)
```

The output in this example is as follows:

```
summary =
```

```
2×1 tokenizedDocument:
```

```
32 tokens: More than 1 in 3 tree species worldwide at risk of
extinction . The loss of trees is also a major threat to thousands of other
plants , animals and fungi .
```

```
22 tokens: ' The risk of extinction is accelerating ' : world's botanic
gardens raise alarm with space to protect endangered plants running out
```

The summary here consists of two tokenised documents that each represent key themes and concepts from the original set. In this output, the first selected document discusses the risk of extinction for tree species and its broader ecological impact. The second document highlights concerns raised by botanic gardens regarding the protection of endangered plants. The selection of these specific documents likely stems from their relevance to the themes of extinction and conservation, which are critical issues in the provided texts.

To specify the size of the summary, we can use the 'SummarySize' name-value pair argument. To return a summary with three documents, for instance, set the 'SummarySize' option to 3. This will return three documents whose contents are particularly relevant to the overall content:

```
summary = extractSummary(documents,'SummarySize',3)
```

Additionally, if you want to ensure that the summary documents appear in the same order as the input documents, you can set the 'OrderBy' option to 'position':

```
summary = extractSummary(documents, 'SummarySize', 3, 'OrderBy', 'position')
```

The `extractSummary` function can also return importance scores for each document in the summary. These scores indicate how significant each document is relative to the others in the context of the summary. To retrieve the scores, specify `scores` as the second output of the function. To extract a summary and the corresponding importance scores for a specified number of documents (three in this case) use:

```
[summary,scores] = extractSummary(documents,'SummarySize',3)
```

To extract a summary and the corresponding importance scores while also specifying the order of the documents based on their position, use a command like the following:

```
[summary, scores] = extractSummary(documents, 'SummarySize', 3, 'OrderBy', 'position');
```

The output here will be as follows:

```
summary =
  3×1 tokenizedDocument:

    22 tokens: ' The risk of extinction is accelerating ' : world's botanic
gardens raise alarm with space to protect endangered plants running out
    32 tokens: More than 1 in 3 tree species worldwide at risk of
extinction . The loss of trees is also a major threat to thousands of other
plants , animals and fungi .
    25 tokens: With 63 % of Ireland's birds at risk of extinction , an
Irish ornithologist is utilising recordings he is making to advocate for
their preservation

scores =

    0.1057
    0.1710
    0.1023
```

After retrieving the summary, if you have multiple documents or sentences as output and you want to join them into a single string, you can achieve this by using the `joinWords` function (<https://www.mathworks.com/help/textanalytics/ref/tokenizeddocument.joinwords.html>) followed by the `join` function (cf. <https://www.mathworks.com/help/matlab/ref/join.html>). The `joinWords` function convert your `tokenizedDocument` array into a string array by concatenating the words with a space:

```
summary_new = joinWords(summary);
```

Then the `join` function combines the elements of the string array into a single string, which might be easier to use or to process further than separate items in the summary:

```
summary_text = join(summary_new);
```

Let us have a look at another example where we want to summarise the content of an individual text. We first define the text to be summarised, e.g., in this case an excerpt from a news texts sourced from the Euronews website at <https://www.euronews.com/culture/2025/09/17/why-are-pacifiers-becoming-a-growing-and-viral-trend-among-adults>.

```
text = "Adult versions of pacifiers are all the rage at the moment, with a growing number of social media users adopting the viral trend to deal with the stress of everyday life. What do the experts say, and are we wrong to be so judgemental? Pacifier. Dummy. Soothie. Jar Jar Binky. Call them whatever you want, the rubber / plastic / silicone nipple substitutes are making a comeback. And they're not just for toddlers anymore. The latest craze spreading online is adult pacifiers - and before you legitimately cry out that we're backsliding as a species or that we're seemingly hellbent on disproving Darwinism, the trend is rooted in something more serious than just the latest fashion fad or attention-seeking quirk. It all kicked off with stressed out adults in China, who started posting online that they were turning to adult pacifiers to relieve their stress and anxiety. Early August, the South China Morning Post reported that adults in China were spending upwards of €60 for adult pacifiers - which are larger than the average child dummy - and some retailers were even claiming that they were selling more than 2,000 every month. More than that, e-commerce platforms such as Taobao and JD.com have seen a surge in listings for adult pacifiers.";
```

First, we split the text into sentences, then tokenise the document and finally extract a summary consisting of representative sentences from the document.

```
text_sentences = splitSentences(text);
document = tokenizedDocument(text_sentences);
summary = extractSummary(document)
```

Figure 229 shows the output of these steps which includes two representative sentences from the text.

```

1 % Define the text to be summarised
2 text = "Adult versions of pacifiers are all the rage at the moment, with a growing number " + ...
3 "of social media users adopting the viral trend to deal with the stress of everyday life. " + ...
4 "What do the experts say, and are we wrong to be so judgemental? Pacifier. Dummy. Soothie. " + ...
5 "Jar Jar Binky. Call them whatever you want, the rubber / plastic / " + ...
6 "silicone nipple substitutes are making a comeback. " + ...
7 "And they're not just for toddlers anymore. The latest craze spreading online is adult pacifiers " + ...
8 "- and before you legitimately cry out that we're backsliding as a species or that we're " + ...
9 "seemingly hellbent on disproving Darwinism, the trend is rooted in something more serious " + ...
10 "than just the latest fashion fad or attention-seeking quirk. It all kicked off with stressed out " + ...
11 "adults in China, who started posting online that they were turning to adult pacifiers to relieve " + ...
12 "their stress and anxiety. Early August, the South China Morning Post reported that adults " + ...
13 "in China were spending upwards of €60 for adult pacifiers - which are larger than " + ...
14 "the average child dummy - and some retailers were even claiming that they were " + ...
15 "selling more than 2,000 every month. More than that, e-commerce platforms such as " + ...
16 "Taobao and JD.com have seen a surge in listings for adult pacifiers.";
17
18 % Split text into sentences
19 text_sentences = splitSentences(text);
20
21 % Tokenize the document
22 document = tokenizedDocument(text_sentences);
23
24 % Extract a summary of the document
25 summary = extractSummary(document)
26
Command Window
>> text_summary_one_document
summary =
2x1 tokenizedDocument:
48 tokens: Early August, the South China Morning Post reported that adults in China were spending upwards of € 60 for adult pacifiers - which are larger than
48 tokens: The latest craze spreading online is adult pacifiers - and before you legitimately cry out that we're backsliding as a species or that we're seemi

```

Figure 229: Summarising the content of a text with the `extractSummary` function

Additionally, consider extracting keywords, as this can also assist in summarising texts. The Text Analytics Toolbox provides several options for extracting keywords from texts and documents, including the TextRank and RAKE (Rapid Automatic Keyword Extraction) algorithms. The TextRank algorithm identifies keywords by analysing the relationships between words based on their part-of-speech tags and scoring them through word co-occurrences. The RAKE

algorithm, on the other hand, focuses on the frequency of word co-occurrences and is particularly useful for extracting multi-word phrases. Here are some links to explore these options:

*Extract keywords from text data using TextRank*

<https://www.mathworks.com/help/textanalytics/ug/extract-keywords-from-text-data-using-textrank.html>

*textrankKeywords – Extract keywords using TextRank*

<https://www.mathworks.com/help/textanalytics/ref/textrankkeywords.html>

*Extract keywords from text data using RAKE*

<https://www.mathworks.com/help/textanalytics/ug/extract-keywords-from-documents-using-rake.html>

*rakeKeywords – Extract keywords using RAKE*

<https://www.mathworks.com/help/textanalytics/ref/rakekeywords.html>

## 10.5 Further resources on Text Analytics Toolbox™ techniques

The previous sections provided an overview of various techniques or working with the Text Analytics Toolbox. We covered methods for extracting text from a range of document types as well as essential preprocessing steps to prepare text data for analysis. Additionally, we explored some key functionalities within the toolbox, including the bag-of-words model, the bag-of-n-grams model and word embeddings which play an important role in transforming raw text into structured data to make it easier to extract meaningful patterns and insights.

For readers interested in a more comprehensive understanding of the wide range of text analytics capabilities when working with the Text Analytics Toolbox, the MathWorks documentation offers various resources and practical examples. These materials expand on the applications covered in this book with additional details and examples and also introduce additional techniques not covered in this introductory book such as sentiment analysis, topic modelling, text summarisation and measuring document similarity.

Below are some ideas for topics that you can explore, along with links to the relevant documentation.

You can explore a range of topics related to text data modelling under the following link:

*Data modelling*

<https://www.mathworks.com/help/textanalytics/text-data-modeling.html>.

The documentation also covers how the Text Analytics Toolbox integrates with other MATLAB toolboxes to support more complex workflows.

If you are interested in sentiment analysis, the documentation offers a comprehensive guide on using the Valence Aware Dictionary and sEntiment Reasoner (VADER) algorithm to evaluate sentiment in text. It covers the process from loading and tokenising text data to calculating sentiment scores. Additionally, you may consider creating a domain-specific sentiment lexicon to improve the accuracy of your analysis, especially when working with specialised text. For

more advanced applications, you can also train a sentiment classifier using labelled data, incorporating features extracted from the text and using a sentiment lexicon to build a supervised learning model. Here are some links to explore these techniques further:

*Analyse sentiment in text*

<https://www.mathworks.com/help/textanalytics/ug/analyze-sentiment-in-text.html>

*Sentiment analysis in MATLAB*

<https://www.mathworks.com/help/textanalytics/ug/sentiment-analysis-in-matlab.html>

*Generate domain specific sentiment lexicon*

<https://www.mathworks.com/help/textanalytics/ug/generate-domain-specific-sentiment-lexicon.html>

*Train a sentiment classifier*

<https://www.mathworks.com/help/textanalytics/ug/train-a-sentiment-classifier.html>

Additionally, for those looking to analyse text data with topic models, the MathWorks documentation demonstrates in detail how to use the Latent Dirichlet Allocation (LDA) topic model. It includes steps for loading data, preprocessing text and fitting the LDA model:

*Analyse text data using topic models*

<https://www.mathworks.com/help/textanalytics/ug/analyze-text-data-using-topic-models.html>

You can also explore how to visualise the topic probabilities of documents using an LDA model and how to fit an LDA model and visualise the correlations between LDA topics and document labels using parallel coordinates plots:

*Visualise LDA topic probabilities of documents*

<https://www.mathworks.com/help/textanalytics/ug/visualize-lda-topic-probabilities-of-documents.html>

*Visualise correlations between LDA topics and document labels*

<https://www.mathworks.com/help/textanalytics/ug/visualize-correlations-between-lda-topics-and-document-labels.html>

For those interested in document classification, the following resources provide guidance on using document embeddings and advanced models like BERT for classification tasks:

*Classify documents using document embeddings*

<https://www.mathworks.com/help/textanalytics/ug/classify-documents-using-document-embeddings.html>

*Document embedding model to map documents to vectors*

<https://www.mathworks.com/help/textanalytics/ref/documentembedding.html>

*Train BERT document classifier*

<https://www.mathworks.com/help/textanalytics/ug/train-bert-document-classifier.html>

The Text Analytics Toolbox in MATLAB also provides several methods for assessing document similarity, including BM25 similarity and cosine similarity. Additionally, it offers the

TextRank algorithm. BM25 similarity is a probabilistic model uses term frequency and inverse document frequency (cf. Chapter 10.4.1) to compute similarity scores. Cosine similarity measures the cosine of the angle between two document vectors in a multi-dimensional space, typically derived from a term frequency-inverse document frequency (TF-IDF) matrix. This method provides a numerical value between -1 and 1, where 1 indicates that the documents are identical in terms of content, and 0 indicates no similarity. TextRank scores are derived from the TextRank algorithm, which evaluates the importance of documents based on their pairwise similarities. The scores reflect the relevance of each document within the context of the dataset.

You can find the documentation and explanations of the algorithms here:

*Document similarities with BM25 algorithm*

<https://www.mathworks.com/help/textanalytics/ref/bm25similarity.html>

*Document similarities with cosine similarity*

<https://www.mathworks.com/help/textanalytics/ref/cosinesimilarity.html>

*Document scoring with TextRank algorithm*

<https://www.mathworks.com/help/textanalytics/ref/textrankscores.html>

In addition to the written documentation, MathWorks provides some video tutorials that illustrate practical applications and workflows with the Text Analytics Toolbox. Have a look at the available videos at <https://www.mathworks.com/videos.html>.

Here is a link to a webinar by MathWorks that gives a general overview of text analytics in MATLAB using the Text Analytics Toolbox. It includes a demonstration of topic modelling with vehicle repair logs and sentiment analysis using social media data:

*Text analytics in MATLAB*

<https://www.mathworks.com/videos/text-analytics-in-matlab-1544195844130.html>

Among the video resources, you will also find the following series of four tutorial videos on text analytics for biomedical applications:

*Text analytics for biomedical applications*

<https://www.mathworks.com/videos/series/text-analytics-for-biomedical-applications.html>

This series includes an introductory video, a step-by-step guide to building sentiment analysis models using Twitter data, an explanation of topic modelling and document summarisation on the basis of COVID-19 research articles as the dataset and an example of predicting hospital readmissions using text classification techniques. You can find the links to the individual videos below:

*Introduction to text analytics | Text analytics for biomedical applications, Part 1*

<https://www.mathworks.com/videos/text-analytics-for-biomedical-applications-part-1-introduction-to-text-analytics-1612360372590.html>

*Twitter sentiment analysis for biomedical applications | Text analytics for biomedical Applications, Part 2*

<https://www.mathworks.com/videos/text-analytics-for-biomedical-applications-part-2-twitter-sentiment-analysis-for-biomedical-applications-1612360392332.html>

*COVID-19 Topic modelling and document summarisation | Text analytics for biomedical applications, Part 3*

<https://www.mathworks.com/videos/text-analytics-for-biomedical-applications-part-3-covid-19-topic-modeling-and-document-summarization-1612360436581.html>

*Predict hospital readmission using text classification | Text analytics for biomedical applications, Part 4*

<https://www.mathworks.com/videos/text-analytics-for-biomedical-applications-part-4-predict-hospital-readmission-using-text-classification-1612360462487.html>

There is also a highly useful book on text mining by Banchs (2021), which has already been referenced earlier in this work. It provides a wide range of examples using functions from the Text Analytics Toolbox in MATLAB and is an excellent resource for those interested in exploring various text mining techniques. While the book's examples are based on an earlier version of MATLAB (R2020b), the concepts and methods discussed remain highly relevant and applicable.

With these resources, you will be able to deepen your understanding and specialise in specific text analytics techniques based on the functions of the Text Analytics Toolbox in MATLAB.

## 11 Summary

This book has provided a comprehensive guide to getting started with MATLAB for corpus-linguistic research. It is hoped that readers will find this book both informative and practical as they explore and apply MATLAB to linguistic data derived from corpora or in compiling and processing their own corpus data.

Starting with an introduction to MATLAB's capabilities, the book established a foundation in the software's environment and functions. Step-by-step walkthroughs of installation, the user interface and basic programming concepts provided readers with the tools to set up and navigate their own MATLAB workspace and paved the way for further exploration and application in their research endeavours.

The book then explored how to retrieve and manipulate corpus data. From there, it progressed to core MATLAB functionalities such as variable creation, data visualisation techniques, arithmetic operations and statistical analysis, all tailored for examples from the field of corpus linguistics. A dedicated section on text analytics, with a focus on text extraction, preprocessing and analysis with MATLAB's Text Analytics Toolbox, completed the coverage.

In conclusion, this book not only demystifies MATLAB as a tool for corpus linguistics but also provides a structured approach for users to integrate its powerful features into their own research. The hope is that readers will now feel empowered and confident to apply MATLAB to their own linguistic data. Overall, this book aims to spark curiosity and encourage deeper engagement with both MATLAB and corpus-linguistic research. With the concepts and techniques introduced here, readers are now prepared to tackle a wide range of linguistic research questions and refine their skills in programming and using MATLAB for textual data analysis.



# Bibliography

- Alsop, Sian & Hilary Nesi (2009). Issues in the development of the British Academic Written English (BAWE) corpus. *Corpora* 4 (1), <https://doi.org/10.3366/E1749503209000227>.
- Banchs, Rafael E. (2021). *Text mining with MATLAB®*, 2nd ed. New York: Springer.
- Baroni, Marco & Stefan Evert (2009). 36. Statistical methods for corpus exploitation. In: *Volume 2: An International Handbook*, edited by Anke Lüdeling and Merja Kytö, Berlin, New York: De Gruyter Mouton, 2009, 777-803, <https://doi.org/10.1515/9783110213881.2.777>.
- Bird, Steven, Robert Dale, Bonnie Dorr, Bryan Gibson, Mark Joseph, Min-Yen Kan, Dongwon Lee, Brett Powley, Dragomir Radev & Yee Fan Tan. 2008. The ACL Anthology Reference Corpus: A reference dataset for bibliographic research in computational linguistics. In: *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, Marrakech. European Language Resources Association (ELRA). [http://www.lrec-conf.org/proceedings/lrec2008/pdf/445\\_paper.pdf](http://www.lrec-conf.org/proceedings/lrec2008/pdf/445_paper.pdf) (Accessed: 03/03/2026).
- Boutsen, Frank R. & Justin D. Dvorak (2016). *MATLAB primer for speech-language pathology and audiology*. Plural Publishing, Inc.
- Ciaburro, Giuseppe (2024). *MATLAB for machine learning* (2nd ed.). Packt Publishing.
- Davies, Mark (2004). *British National Corpus* (from Oxford University Press). Available at <https://www.english-corpora.org/bnc/> (Accessed: 03/03/2026).
- Davies, Mark (2007). *TIME Magazine Corpus*. Available at <https://www.english-corpora.org/time/> (Accessed: 03/03/2026).
- Davies, Mark (2008–). *The Corpus of Contemporary American English (COCA)*. Available at <https://www.english-corpora.org/coca/> (Accessed: 03/03/2026).
- Davies, Mark (2010). *The Corpus of Historical American English (COHA)*. Available at <https://www.english-corpora.org/coha/> (Accessed: 03/03/2026).
- Davies, Mark (2012–). *The Strathy Corpus of Canadian English* (from the Strathy Language Unit, Queen's University). Available at <https://www.english-corpora.org/can/> (Accessed: 03/03/2026).
- Davies, Mark (2016–). *Corpus of News on the Web (NOW)*. Available at <https://www.english-corpora.org/now/> (Accessed: 03/03/2026).
- Davies, Mark (2019). *The Movie Corpus*. Available at <https://www.english-corpora.org/movies/> (Accessed: 03/03/2026).
- Eddington, David (2016). *Statistics for linguists: A step-by-step guide for novices*. Newcastle: Cambridge Scholars Publishing.
- Evert, Stephanie & CWB Development Team (2022). *The IMS Open Corpus Workbench (CWB) – CQP interface and query language manual. V3.5*. [https://cwb.sourceforge.io/files/CQP\\_Manual.pdf](https://cwb.sourceforge.io/files/CQP_Manual.pdf) (Accessed: 03/03/2026).
- Fischer, Stefan, Jörg Knappen, Katrin Menzel & Elke Teich (2020). The Royal Society Corpus 6.0: Providing 300+ years of scientific writing for humanistic study. In: *Proceedings of the 12th Language Resources and Evaluation Conference (LREC 2020)*, Marseille, May 2020, 794-802, <https://aclanthology.org/2020.lrec-1.99/> (Accessed: 03/03/2026).
- Francis, W. Nelson & Henry Kučera (1964). *Manual of information to accompany a Standard Corpus of Present-Day Edited American English, for use with digital computers*. Providence, Rhode Island: Department of Linguistics, Brown University. Revised 1971. Revised and amplified 1979.
- Halliday, Michael A. K. & Ruqaiya Hasan (1976). *Cohesion in English*. London: Longman.
- Hardie, Andrew (2012). CQPweb – Combining power, flexibility, and usability in a corpus analysis tool. *International Journal of Corpus Linguistics* 17(3), 380-409.
- Hitchcock, Tim, Robert Shoemaker, Clive Emsley, Sharon Howard, Jamie McLaughlin et al. (2023). *The Old Bailey Proceedings Online, 1674-1913* (version 9.0, Autumn 2023). Available at [www.oldbaileyonline.org](http://www.oldbaileyonline.org) (Accessed: 03/03/2026).
- Huddleston, Rodney & Geoffrey K. Pullum (2002): *The Cambridge grammar of the English language*. Cambridge: Cambridge University Press.

- Karakanta, Alina, Mihaela Vela & Elke Teich (2018). Europarl-UdS: Preserving metadata from parliamentary debates. In: *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, May 2018.
- Kermes, Hannah, Stefania Degaetano-Ortlieb, Ashraf Khamis, Jörg Knappen & Elke Teich (2016). The Royal Society Corpus: From uncharted data to corpus. In: *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia, May 2016, European Language Resources Association, 1928-1931, <https://aclanthology.org/L16-1305/> (Accessed: 03/03/2026).
- Kilgarriff, Adam, Vít Baisa, Jan Bušta, Miloš Jakubiček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlý & Vít Suchomel (2014). The Sketch Engine: Ten years on. *Lexicography*, 1(1), 7-36.
- Kilgarriff, Adam (1997). Putting Frequencies in the Dictionary. *International Journal of Lexicography* 10 (2), 135-155.
- Kilgarriff, Adam, Pavel Rychlý, Petr Smrž & David Tugwell (2004). The Sketch Engine. In: *Proceedings of the 11th EURALEX International Congress*, 105-116.
- Koehn, Philipp (2005). Europarl: A parallel corpus for statistical machine translation. In: *Proceedings of Machine Translation Summit X: Papers*, 79-86, Phuket, Thailand.
- Kunz, Kerstin, Ekaterina Lapshinova-Koltunski, José Martínez-Martínez, Katrin Menzel & Erich Steiner (2021). *German-English contrasts in cohesion. Trends in linguistics. Studies and monographs [TiLSM]*. Mouton de Gruyter.
- Leech, Geoffrey, Paul Rayson & Andrew Wilson (2001). *Word frequencies in written and spoken English: Based on the British National Corpus* London: Routledge.
- Menzel, Katrin (2025). Patterns of change in Late Modern English microbiology texts. In: Brookes, Gavin, Niall Curry, Anthony McEnery & Emma Putland. *Historical medical discourse: Corpus linguistic perspectives*. London: Routledge, 39-69, <https://doi.org/10.4324/9781003467014-3>.
- Menzel, Katrin (2024). Exploring word formation trends in written, spoken, translated and interpreted European Parliament data – A case study on initialisms in English and German. In: *Proceedings of the IV Workshop on Creating, Analysing and Increasing Accessibility of Parliamentary Corpora (ParlaCLARIN) @ LREC-COLING 2024*, 57-65, Torino, Italia. ELRA and ICCL.
- Menzel, Katrin (2017). *Understanding English–German contrasts – A corpus-based comparative analysis of ellipses as cohesive devices*, Doctoral Dissertation, Saarbrücken: Saarländische Universitäts- und Landesbibliothek, <https://doi.org/10.22028/D291-23663>.
- Mueller, John Paul & Jim Sizemore (2021). *MATLAB for dummies*, 2nd edition. Hoboken, NJ, USA: Wiley.
- Mustafy, Tofail & Md. Tauhid Ur Rahman (2024). *Statistics and data analysis for Engineers and Scientists*. Springer.
- Sketch Engine (Lexical Computing Ltd.). (n.d.). *Corpus list*. <https://www.sketchengine.eu/corpora-and-languages/corpus-list/> (Accessed: 03/03/2026).
- Thompson, Paul & Hilary Nesi (2001). *The British Academic Spoken English (BASE) Corpus*. University of Warwick and University of Reading.
- University of Oxford, Oxford Text Archive (n.d.). Available at: <https://ota.bodleian.ox.ac.uk/repository/xmlui/> (Accessed: 03/03/2026).
- Wang, Lucy Lu, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Doug Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Kinney, Yunyao Li, Ziyang Liu, William Merrill, Paul Mooney, Dewey Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex Wade, Kuansan Wang, Nancy Xin Ru Wang, Chris Wilhelm, Boya Xie, Douglas Raymond, Daniel S. Weld, Oren Etzioni and Sebastian Kohlmeier (2020). Cord-19: The COVID-19 Open Research Dataset. *ArXiv*, <https://doi.org/10.48550/arXiv.2004.10706>.
- Woodrow, Lindy (2014). *Writing about quantitative research in applied linguistics*. London: Palgrave Macmillan.
- Woodrow, Lindy (2014a). ANOVA, ANCOVA and MANOVA. In: Woodrow, Lindy. *Writing about quantitative research in applied linguistics*. London: Palgrave Macmillan, 73-84.
- Woodrow, Lindy (2014b). Writing about regression. In: Woodrow, Lindy. *Writing about quantitative research in applied linguistics*. London: Palgrave Macmillan, 85-95.

This book provides a practical introduction to MATLAB® for analysing and visualising linguistic data. It is aimed at students and researchers working with corpora and text data. MATLAB®, a software and programming environment widely used in the natural and engineering sciences, is also well suited for linguistic research and provides a powerful and intuitive environment that is easy to learn. Step by step, this book covers topics such as data import and processing, mathematical and statistical analysis, visualisation and text analysis. Until now, no handbook has focused specifically on MATLAB® examples in linguistics. The present book fills this gap.

Dieses Buch bietet eine praxisnahe Einführung in MATLAB® für die Analyse und Visualisierung linguistischer Daten. Es richtet sich an Studierende und Forschende der Sprachwissenschaft, die mit Korpus- und Textdaten arbeiten. Die in den Natur- und Ingenieurwissenschaften verbreitete Software und Programmierumgebung MATLAB® eignet sich auch hervorragend für die Sprachwissenschaft und bietet eine leistungsfähige, intuitive Umgebung, die leicht zu erlernen ist. Schritt für Schritt werden Grundlagen vermittelt, darunter Datenimport und -verarbeitung, mathematische und statistische Auswertung, Visualisierung und Textanalyse. Ein Handbuch mit MATLAB®-Beispielen zur linguistischen Datenanalyse existierte bislang nicht – dieses Buch schließt diese Lücke.

