

An Introduction to Quantitative Text Analysis for Linguistics

Reproducible Research using R

Jerid Francom

October 17, 2023

Table of contents

License	7
Credits	8
Acknowledgements	8
Build information	8
Preface	9
Rationale	9
Aims	10
Approach	11
Structure	11
Book level	11
Chapter level	12
Resources	13
Getting started	14
R and IDEs	14
R packages	15
Git and GitHub	17
Getting help	17
Conventions	19
Prose	19
Code blocks	20
Callouts	21
To the instructor	22
Basic Introduction	22
Intermediate Introduction	22
Advanced Introduction	22
Activities	23
Summary	24
Questions	24

I Orientation	26
1 Text analysis in context	28
1.1 Making sense of a complex world	29
1.1.1 Heuristic Understanding	29
1.1.2 Science to advance understanding	30
1.2 Data analysis	31
1.2.1 Emergence of data science	31
1.2.2 Computing skills, statistical knowledge, and domain knowledge	31
1.2.3 Applications of data science	32
1.3 Language analysis	33
1.4 Text analysis	36
1.4.1 Approaches	37
1.4.2 Implementation	37
1.4.3 Applications	38
Summary	39
Activities	40
Questions	41
II Foundations	42
2 Understanding data	44
2.1 Data	45
2.1.1 Populations	45
2.1.2 Samples	45
2.1.3 Corpora	46
2.2 Information	57
2.2.1 Organization	58
2.2.2 Transformation	61
2.3 Documentation	73
2.3.1 Data origin	73
2.3.2 Data dictionaries	74
Summary	75
Activities	76
Questions	76
3 Approaching analysis	78
3.1 Diagnose	79
3.1.1 Verify	80
3.1.2 Describe	82
3.2 Analyze	101
3.2.1 Explore	101

3.2.2 Predict	104
3.2.3 Infer	107
3.3 Communicate	111
3.3.1 Report	111
3.3.2 Document	112
Summary	113
Activities	114
Questions	115
4 Framing research	116
4.1 Frame	117
4.2 Connect	117
4.2.1 Research area	117
4.2.2 Research problem	118
4.3 Define	121
4.3.1 Research aim	121
4.3.2 Research question	122
4.4 Blueprint	123
4.4.1 Plan	124
4.4.2 Scaffold	127
Summary	129
Activities	130
Questions	131
III Preparation	132
5 Acquire data	134
5.1 Downloads	135
5.1.1 Manual	135
5.1.2 Programmatic	140
5.2 APIs	148
Summary	157
Activities	158
Questions	158
6 Curate datasets	162
6.1 Unstructured	163
6.1.1 Reading data	163
6.1.2 Orientation	164
6.1.3 Tidy the data	167
6.1.4 Write dataset	174

6.2	Structured	176
6.2.1	Reading data	177
6.2.2	Orientation	178
6.2.3	Tidy the dataset	179
6.2.4	Write dataset	187
6.3	Semi-structured	189
6.3.1	Reading data	189
6.3.2	Orientation	190
6.3.3	Tidy the data	192
6.3.4	Write dataset	200
	Summary	202
	Activities	202
	Questions	203
7	Transform datasets	206
7.1	Normalization	206
7.1.1	Orientation	207
7.1.2	Application	211
7.2	Recoding	217
7.2.1	Orientation	217
7.2.2	Application	219
7.3	Tokenization	225
7.3.1	Orientation	226
7.3.2	Application	227
7.4	Generation	235
7.4.1	Orientation	235
7.4.2	Application	236
7.5	Merging	242
7.5.1	Orientation	246
7.5.2	Application	247
	Summary	252
	Activities	253
	Questions	254
IV	Analysis	255
8	Exploration	257
8.1	Orientation	258
8.1.1	Research goal	258
8.1.2	Approach	258
8.2	Analysis	260
8.2.1	Descriptive analysis	263

8.2.2	Unsupervised learning	295
8.3	Summary	322
	Activities	322
	Questions	323
9	Prediction	324
9.1	Orientation	325
9.1.1	Research goal	325
9.1.2	Approach	326
9.2	Analysis	328
9.2.1	Text classification	330
9.2.2	Text regression	359
9.3	Summary	383
	Activities	383
	Questions	384
10	Inference	386
V	Communication	387
11	Reports	389
12	Collaboration	390
	References	391
	Appendices	398
A	Data	398
A.1	ANC	398
A.2	BNC	398
A.3	CABNC	398
A.4	CEDEL2	398
A.5	ENNTT	398
B	Feedback 	399
	Where to review	399
	What to look for	399
	How to submit feedback	400
	Thank yous!	400

Welcome

The goal of this textbook is to provide readers with foundational knowledge and practical skills in quantitative text analysis using the R programming language.

By the end of this textbook, readers will be able to identify, interpret and evaluate data analysis procedures and results to support research questions within language science. Additionally, readers will gain experience in designing and implementing research projects that involve processing and analyzing textual data employing modern programming strategies. This textbook aims to instill a strong sense of reproducible research practices, which are critical for promoting transparency, verification, and sharing of research findings.

This textbook is geared towards advanced undergraduates, graduate students, and researchers looking to expand their methodological toolbox. It assumes no prior knowledge of programming or quantitative methods and prioritizes practical application and intuitive understanding over technical details.

About the author

Dr. Jerid Francom is Associate Professor of Spanish and Linguistics at Wake Forest University. His research focuses on the use of language corpora from a variety of sources (news, social media, and other internet sources) to better understand the linguistic and cultural similarities and differences between language varieties for both scholarly and pedagogical projects. He has published on topics including the development, annotation, and evaluation of linguistic corpora and analyzed corpora through corpus, psycholinguistic, and computational methodologies. He also has experience working with and teaching statistical programming with R.

License

This work by Jerid C. Francom¹ is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

¹<https://francojc.github.io/>

Credits

Font Awesome Icons² are SIL OFL 1.1 Licensed

Acknowledgements

The development of this book has benefited from the generous feedback from the following people: Andrea Bowling, Caroline Brady, Declan Golsen, Asya Little, Claudia Valdez, Laura Aull, Jack Nelson, *(add your name here!)*. As always, any errors or omissions are my own.

Build information

This textbook was built with the `quarto` package (Allaire 2023) and the `bookdown` package (Xie 2023a) for R. The source code for this book is available on GitHub³.

This version of the textbook was built with R version 4.3.1 (2023-06-16) on macOS Ventura 13.5.2 with the following packages:

package	version	source
dplyr	1.1.3	CRAN (R 4.3.1)
ggplot2	3.4.3	CRAN (R 4.3.1)
here	1.0.1	CRAN (R 4.3.0)
knitr	1.44	CRAN (R 4.3.1)
qtalrkit	0.0.3.300	Github (qtalr/qtalrkit@37814b5085b0e62a95532fdb3a782d28b44f00a)
readr	2.1.4	CRAN (R 4.3.0)
reprex	2.0.2	CRAN (R 4.3.0)
rmarkdown	2.25	CRAN (R 4.3.1)
rstudioapi	0.15.0	CRAN (R 4.3.1)
scales	1.2.1	CRAN (R 4.3.0)
stringr	1.5.0	CRAN (R 4.3.0)
tibble	3.2.1	CRAN (R 4.3.0)
tidyverse	1.3.0	CRAN (R 4.3.0)

²<https://fontawesome.com/>

³<https://github.com/qtalr/book>

Preface



Draft

Ready for review.

The journey of a thousand miles begins with one step.

— Lao Tzu

▀ Outcomes

- Comprehend the book's rationale, learning goals, and pedagogical approach.
- Navigate and engage with the book's structure and content effectively.
- Interpret conventions used in the book reliably.
- Set up the computing environment and utilize textbook and support resources for an optimal learning experience.

The purpose of this chapter is to present the rationale behind this textbook, outline the key learning objectives, describe the pedagogical approach, and identify the intended audience. Additionally, this chapter will provide readers with a guide to the book's structure and the scope of its content, as well as instructions for the instructor and a summary of supporting resources available. Finally, this chapter will provide readers with information on setting up their computing environment and where to seek support.

Rationale

Data science, an interdisciplinary field that combines knowledge and skills from statistics, computer science, and domain-specific expertise to extract meaningful insight from structured and unstructured data, has emerged as an exciting and rapidly growing field in recent years, driven in large part by the increase in computing power available to the average individual and the abundance of electronic data now available through the internet. These advances have become an integral part of the modern scientific landscape, with data-driven insights now being used to inform decision-making in a wide variety of academic fields, including linguistics and language-related disciplines.

This textbook seeks to meet this growing demand by providing an introduction to the fundamental concepts and practical programming skills from data science applied to the task of quantitative text analysis. It is intended primarily for undergraduate students, but may also be useful for graduates and researchers seeking to expand their methodological toolbox. The textbook takes a pedagogical approach which assumes no prior experience with statistics or programming, making it an accessible resource for novices beginning their exploration of quantitative text analysis methods.

Aims

The overarching goal of this textbook is to provide readers with foundational knowledge and practical skills to conduct and evaluate quantitative text analysis using the R programming language and other open source tools and technologies. The specific aims are to develop the reader's proficiency in three main areas:

- **Data literacy:** Identify, interpret and evaluate data analysis procedures and results

Throughout this textbook we will explore topics which will help you understand how data analysis methods derive insight from data. In this process you will be encouraged to critically evaluate connections across linguistic and language-related disciplines using data analysis knowledge and skills. Data literacy is an invaluable skillset for academics and professionals but also is an indispensable aptitude for in the 21st century citizens to navigate and actively participate in the 'Information Age' in which we live (Carmi et al. 2020).

- **Research skills:** Design, implement, and communicate quantitative research

This aim does not differ significantly, in spirit, from common learning outcomes in a research methods course. However, working with text will incur a series of key steps in the selection, collection, and preparation of the data that are unique to text analysis projects. In addition, I will stress the importance of research documentation and creating reproducible research as an integral part of modern scientific inquiry (Buckheit and Donoho 1995).

- **Programming skills:** Apply programmatic strategies to develop and collaborate on reproducible research projects

Modern data analysis, and by extension, text analysis is conducted using programming. There are various key reasons for this: a programming approach (1) affords researchers unlimited research freedom –if you can envision it, you can program it, (2) underlies well-documented and reproducible research (Gandrud 2015), and (3) invites researchers to engage more intimately with the data and the methods for analysis.

These aims are important for linguistics students because they provide a foundation for concepts and in the skills required to succeed in the rapidly evolving landscape of 21st-century research. These abilities enable researchers to evaluate and conduct high-quality empirical investigation across linguistic fields on a wide variety of topics. Moreover, these skills go beyond linguistics research; they are widely applicable across many disciplines where quantitative data analysis and programming are becoming increasingly important. Thus, this textbook provides students with a comprehensive introduction to quantitative text analysis that is relevant to linguistics research and that equips them with valuable skills for their future careers.

Approach

The approach taken in this textbook is designed to accommodate linguistics students and researchers with little to no prior experience with programming or quantitative methods. With this in mind the objective is connect conceptual understanding with practical application. Real-world data and research tasks relevant to linguistics are used throughout the book to provide context and to motivate the learning process⁴. Furthermore, as an introduction to the field, the textbook focuses on the most common and fundamental methods and techniques for quantitative text analysis and prioritizes breadth over depth and intuitive understanding over technical explanations. On the programming side, the Tidyverse approach to programming in R will be adopted. This approach provides a consistent syntax across different packages and is known for its legibility, making it easier for readers to understand and write code. Together, these strategies form an approach that is intended to provide readers with an accessible resource to gain a foothold in the field and to equip them with the knowledge and skills to apply quantitative text analysis in their own research.

Structure

The aims and approach described above is reflected in the overall structure of the book and each chapter.

Book level

At the book level, there are five interdependent parts:

Part I “Orientation” provides the necessary background knowledge to situate quantitative text analysis in the wider context of data analysis and linguistic research and to provide a clearer picture of what text analysis entails and its range of applications.

⁴Research data and questions are primarily based on English for wide accessibility as it is the *de facto* language of academics and research. However, the methods and techniques presented in this textbook are applicable to many other languages.

The subsequent parts are directly aligned with the data analysis process. The building blocks of this process are reflected in ‘Data to Insight Hierarchy (DIKI)’ visualized in Figure 1⁵.

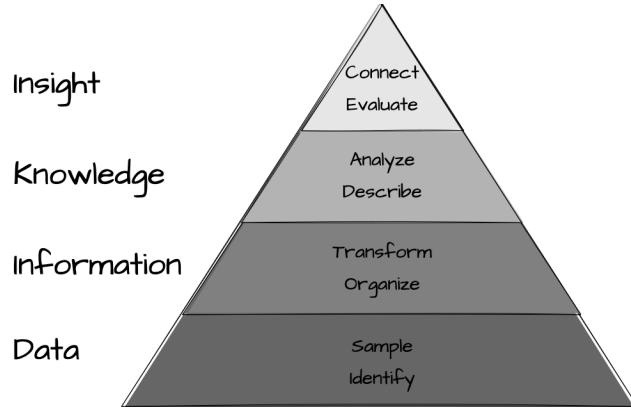


Figure 1: Data to Insight Hierarchy (DIKI)

The DIKI Hierarchy highlights the stages and intermediate steps required to derive insight from data. Part II “Foundations” provides a conceptual introduction to the DIKI Hierarchy and establishes foundational knowledge about data, information, knowledge, and insight which is fundamental to developing a viable research plan.

Parts III “Preparation” and IV “Analysis” focus on the implementation process. Part III covers the steps involved in preparing data for analysis, including data acquisition, curation, and transformation. Part IV covers the steps involved in conducting analysis, including exploratory, predictive, and inferential data analysis.

The final part, Part V “Communication”, covers the final stage of the data analysis process, which is to communicate the results of the analysis. This includes the structure and content of research reports as well as the process of publishing, sharing, and collaborating on research.

Chapter level

At the chapter level, both conceptual and programming skills are developed in stages⁶. The chapter-level structure is consistent across chapters and can be seen in Table 2.

⁵Adapted from Ackoff (1989) and Rowley (2007).

⁶These stages attempt to capture the general progression of learning reflected in Bloom’s Taxonomy (see Krathwohl (2002) for a description and revision).

Table 2: The general structure of a chapter including: the component, its purpose, where to find the resource, and the target learning stage.

Component	Purpose	Resource	Stage
Outcomes Overview	Identify the learning objectives for the chapter Provide a brief introduction to the chapter topic	Textbook	Introduction
Coding Lessons	Teach programming techniques with hands-on interactive exercises	GitHub	Skills
Content	Combine conceptual discussions and programming skills, incorporating thought-provoking questions, relevant studies, and advanced topic references	Textbook	Knowledge
Recipes	Offer step-by-step programming examples related to the chapter	Resources website	Comprehension
Labs	Allow readers to apply chapter-specific concepts and techniques	GitHub	Application
Summary	Review the key concepts and skills covered in the chapter	Textbook	Review
Questions	Assess and expand the reader's knowledge and abilities	Textbook	Assessment

Each chapter will begin with a list of key learning outcomes followed by a brief introduction to the chapter's content. The goal is to orient the reader to the chapter. Next there will be a prompt to complete the interactive coding lesson(s) to introduce reader's to key programming concepts related to the chapter though hands-on experience and then the main content of the chapter will follow. The content will be a combination of conceptual discussions and programming skills, incorporating thought-provoking questions ('Consider this'), relevant studies ('Case study'), and advanced topic references ('Dive deeper'). Together these components form the skills and knowledge phase. The next phase is the application phase. This phase will include step-by-step programming demonstrations related to the chapter (Recipes) and lab exercises that allow readers to apply their knowledge and skills chapter-related tasks. Finally the chapters conclude with a summary of the key concepts and skills covered in the chapter and a set of questions to assess and expand the reader's knowledge and abilities.

Resources

There are three main resources available to support the aims and approach of this textbook. Firstly, the textbook itself provides prose discussion, figures/ tables, R code, case studies, and thought and practical exercises. Secondly, there is a companion R package called `qtalrkit`

(Francom 2023), which includes functions for accessing data and datasets, as well as various useful functions developed specifically for this textbook. In addition, there is a comprehensive website Quantitative Text Analysis for Linguistics Resources⁷(qtalr website) that includes programming tutorials and demonstrations to enhance the reader’s recognition of how programming strategies are implemented. Finally, a GitHub repository⁸ is provided which contains both a set of interactive R programming lessons (Swirl) and lab exercises designed to guide the reader through practical hands-on programming applications. The companion `qtalrkit` package and the GitHub repository are both under active development and will be updated regularly to ensure that supplementary materials remain relevant to the content of the text⁹.

Getting started

Before jumping in to this and subsequent chapter’s textbook activities, it is important to prepare your computing environment and understand how to take advantage of the resources available, both those directly and indirectly associated with the textbook.

R and IDEs

Programming is the backbone for modern quantitative research. Among the many programming languages available, R is a popular open-source language and software environment for statistical computing. R is popular with statisticians and has been adopted as the *de facto* language by many other fields in natural and social sciences, including linguistics. It is freely downloadable from The R Project for Statistical Programming¹⁰ website and is available for macOS, Linux, and Windows¹¹ operating systems.

Successfully installing R is rarely the last step in setting up your R-enabled computing environment. The majority of R users also install an **integrated development environment** (IDE). An IDE, such as RStudio¹² or Visual Studio Code¹³, provide a **graphical user interface** (GUI) for working with R. In effect, IDEs provide a dashboard for working with R and are designed to make it easier to write and execute R code. IDEs also provide a number of other useful features such as syntax highlighting, code completion, and debugging. IDEs are not required to work with R but they are *highly* recommended.

Choosing to install R and an IDE on your personal computer, which is known as your **local environment**, is not the only option to work with R. You can also choose to work with R in the

⁷<https://qtalr.github.io/qtalrkit/>

⁸<https://github.com/qtalr>

⁹Errata for the textbook is found on the qtalr website.

¹⁰<https://www.r-project.org/>

¹¹<https://cloud.r-project.org/>

¹²<https://posit.co/products/open-source/rstudio/>

¹³<https://code.visualstudio.com/>

cloud, a **remote environment**. There are a number of cloud-based options for working with R, including RStudio Cloud¹⁴ and Microsoft Azure¹⁵. These options provide a pre-configured R environment that you can access from any computer with an internet connection. The advantage of working in the cloud is that you do not need to install R or an IDE on your local computer. The disadvantage is that you will need to be connected to the internet to work with R and the free tiers for these services are limited.

If you are new to R, you may want to consider working in the cloud to get started. If you plan to continue to work with R in the future, you will most likely want to install R and an IDE on your local computer or explore using a **virtual environment**. Virtual environments, such as Docker¹⁶, provide a way to use a pre-configured computing environment or create your own that you can share with others. Virtual environments are a good option if you want to ensure that everyone in your research group is working with the same computing environment. Pre-configured virtual environments exist for R through the Rocker project¹⁷ and can be used locally or in the cloud.

There are trade-offs in terms of cost, convenience, and flexibility when choosing to work with R in a local, remote, or virtual environment. The choice is yours and you can always change your mind later. The important thing is to get started and begin learning R. Furthermore, any of the approaches described here will be compatible with this textbook.

For more information and instructions on setting up an R environment consult the following guides.

Guides

- Installing R^a
- Choosing and setting up an IDE^b
- Working with R in remote and virtual environments^c

^a<https://qtalr.github.io/qtalrkit/articles/guide-0.html>

^b<https://qtalr.github.io/qtalrkit/articles/guide-1.html>

^c<https://qtalr.github.io/qtalrkit/articles/guide-2.html>

R packages

As you progress in your R programming experience, you'll find yourself leveraging code from other R users, which is typically provided as packages. Packages are sets of functions and/or datasets that are freely accessible for download, designed to perform a specific set of interrelated tasks. They enhance the capabilities of R. Official R packages can be found in repositories

¹⁴<https://www.rstudio.com/products/cloud/>

¹⁵<https://learn.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/r-developers-guide>

¹⁶<https://www.docker.com/>

¹⁷<https://rocker-project.org/>

like CRAN¹⁸ (Comprehensive R Archive Network), while other packages can be obtained from code-sharing platforms such as GitHub¹⁹.

💡 Consider this

The Comprehensive R Archive Network (CRAN) includes groupings of popular packages related to a given applied programming task called Task Views^a. Explore the available CRAN Task Views listings. Note the variety of areas (tasks) that are covered in this listing. Now explore in more detail one of the following task views which are directly related to topics covered in this textbook noting the associated packages and their descriptions: (1) Cluster, (2) MachineLearning, (3) NaturalLanguageProcessing, or (4) ReproducibleResearch.

^a<https://cran.r-project.org/web/views/>

You will download a number of packages at different stages of this textbook, but there is a set of packages that will be key to have from the get go. Once you have access to a working R/RStudio environment, you can proceed to install the following packages.

Install the following packages from CRAN.

- `tidyverse` (Wickham 2023)
- `rmarkdown` (Allaire et al. 2023)
- `quarto` (Allaire 2023)
- `tinytex` (Xie 2023b)
- `devtools` (Wickham et al. 2022)
- `usethis` (Wickham, Bryan, et al. 2023)
- `swirl` (Kross et al. 2020)

You can do this by running the following code in an R console:

```
# install key packages from CRAN
install.packages(c("tidyverse", "rmarkdown", "quarto", "tinytex", "devtools",
  "usethis", "swirl"))
```

For instructions on how to install the `qtalrkit` package from GitHub and download and use the interactive R programming lessons for this textbook, see the following guides.

↳ Guides

- Getting started^a

^a<https://qtalr.github.io/qtalrkit/articles/qtalrkit.html>

¹⁸<https://cran.r-project.org/>

¹⁹<https://github.com/>

Git and GitHub

GitHub²⁰ is a code sharing website. Modern computing is highly collaborative and GitHub is a very popular platform for sharing and collaborating on coding projects. The lab exercises for this textbook²¹ are shared on GitHub. To access and complete these exercises you will need to sign up for a (free) GitHub account²² and then set up the version control software `git` on your computing environment. `git` is the conduit to interfacing GitHub and for many `git` will already be installed on your computer (or cloud computing environment).

For more information and instructions on setting up version control consult the following guide.

 **Guides**

- Setting up Git and GitHub^a

^a<https://qtalr.github.io/qtalrkit/articles/guide-3.html>

Getting help

The technologies employed in this approach to text analysis will include a somewhat steep learning curve. And in all honesty, the learning never stops! Both seasoned programmers and beginners alike need assistance. Fortunately there is a very large community of programmers who have developed many official support resources and who actively contribute to official and unofficial discussion forums. Together these resources provide many avenues for overcoming challenges.

In Table 3, I provide a list of steps for seeking help with R.

Table 3: Recommended order for seeking help with R.

Order	Resource	Description
1	Official R Documentation	Access the official documentation by running <code>help(package = "package_name")</code> in an R console. Use the <code>?</code> operator followed by the package or function name. Check out available Vignettes by running <code>browseVignettes("package_name")</code> .
2	Web Search	Look for package documentation and vignettes on the web. A popular site for this is R-Universe.

²⁰<https://github.com/>

²¹<https://github.com/stars/francojc/lists/labs>

²²https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home

Order	Resource	Description
3	RStudio IDE Help Toolbar	If you're using RStudio IDE, use the "Help" toolbar menu. It provides links to help resources, guides, and manuals.
4	Online Discussion Forums	Sites like Stack Overflow and RStudio Community are great platforms where the programming community asks and answers questions related to real-world issues.
5	Post Questions with Reprex	When posting a question, especially those involving coding issues or errors, provide enough background and include a reproducible example (reprex) - a minimal piece of code that demonstrates your issue. This helps others understand and answer your question effectively.

The first place to look for help with R is the official documentation of the R package you are using. You can access this documentation by running `help(package = "package_name")` in an R console or using the `? operator` and then the package or function name. Many R packages often include "Vignettes" (long-form documentation and demonstrations). These can be accessed either by running `browseVignettes()` in an R console with the package name in quotes (e.g. `browseVignettes("tidyverse")`).

You can also search the web for package documentation and vignettes. A popular site for this purpose is R-Universe²³.

If you are using the RStudio IDE, the easiest and most convenient place to get help with either R or RStudio is through the RStudio "Help" toolbar menu. There you will find links to help resources, guides, and manuals.

There are a number of very popular discussion forum websites where the programming community asks and answers questions to real-world issues. These sites often have subsections dedicated to particular programming languages or software. The most popular of these sites is Stack Overflow²⁴. There are also R-specific discussion forums such as RStudio Community²⁵.

If you post a question on one of these communities ensure that if your question involves some coding issue or error that you provide enough background such that the community will be able to help you. This is often referred to as a **reproducible example** or "reprex". A reprex is a minimal piece of code that demonstrates the issue you are having. It is a very useful tool for both asking and answering questions.

For information on how to create a reprex consult the following guide.

²³<https://r-universe.dev/search/>

²⁴<https://stackoverflow.com/>

²⁵<https://community.rstudio.com/>

Guides

- Creating reproducible examples^a

^a<https://qtalr.github.io/qtalrkit/articles/guide-4.html>

The take-home message here is that you are not alone. There are many people world-wide that are learning to program and/ or contribute to the learning of others. The more you engage with these resources and communities the more successful your learning will be. As soon as you are able, pay it forward. Posting questions and offering answers helps the community and engages and refines your skills –a win-win.

Conventions

To facilitate the learning process, this textbook will employ a number of conventions. These conventions are intended to help the reader navigate the text and to signal the reader's attention to important concepts and information.

Prose

The following typographic conventions are used throughout the text:

- *Italics*
 - Filenames, file extensions, directory paths, and URLs.
- **Fixed-width**
 - Package names, function names, variable names, and in-line code including expressions and operators.
- **Bold**
 - Key concepts when first introduced.
- **Linked text**²⁶
 - Links to internal and external resources, footnotes, and citations including references to R packages when first introduced.

²⁶<https://qtalr.github.io/qtalrkit/>

Code blocks

More lengthy code will be presented in code blocks, as seen in Example 0.1.

Example 0.1.

```
# A function that takes a name and returns a greeting
greet <- function(name) { # function definition
  paste("Hello", name) # print greeting
} # end function definition

greet(name = "Jerid") # apply function to a name
```



```
> [1] "Hello Jerid"
```

There are a couple of things to note about the code in Example 0.1. First, it shows the code that is run in R as well as the output that is returned. The code will appear in a box and the output will appear below the box. Both code and output will appear in fixed-width font. Output which is text will be prefixed with `>`. Second, the `#` symbol is used to signal a **code comment**, a human-facing description. Everything right of a `#` is not run as code. In this textbook you will see code comments above code on a separate line and to the right of code on the same line. It is good practice to comment your code to enhance readability and to help others understand what your code is doing.

All figures, tables, and images in this textbook are generated by code blocks but only code for those elements that are relevant for discussion will be shown. However, if you wish to see the code for any element in this textbook, you can visit the GitHub repository <https://qtalr.github.io/book/>.

When a reference to a file and its contents is made, it will appear as in File 0.1.

File 0.1 example.R: Example R script

```
# Load libraries
library(tidyverse)

# Add 1 and 1
1 + 1
```

Callouts

Callouts are used to signal the reader's attention to content, activity, and other important sections. The following callouts are used in this textbook:

Content

Outcomes

Learning outcomes for the chapter appear here.

Consider this

Points for you to consider and questions to explore appear here.

Case study

Case studies for applying conceptual knowledge and coding skills covered in the chapter appear here.

Dive deeper

Links to additional resources for diving deeper into the topic appear here.

Activities

Swirl lesson

Links to swirl lessons for practicing coding skills for the chapter appear here.

Recipe

Links to demonstration programming tasks on the qtalr site for the chapter appear here.

Lab

Links to lab exercises for applying conceptual knowledge and coding skills on the qtalr GitHub repository for the chapter appear here.

Other

Tip

Tips for using R and related tools appear here.

Warning

Warnings for using R and related tools appear here.

To the instructor

Depending on the experience level and expectations of your readers, you may want to consider adopting one of the following course designs for using this textbook.

Basic Introduction

- Cover chapters 1-5 in sequence to give your readers a foundational understanding of quantitative text analysis.
- Culminate the course with a research proposal assignment that requires them to identify an interesting linguistic problem, propose ways of solving it using the methods covered in class, and identify potential data sources.
- If your readers have little to no experience with R, you may want to consider using the RStudio Cloud platform to host the course. This will provide them with a pre-installed R environment and allow them to focus on learning the material rather than troubleshooting.

Intermediate Introduction

- Cover chapters 1, 5-10 in sequence to give your readers a deeper understanding of quantitative text analysis methods. Explore additional case studies or dataset examples throughout the course if you wish to supplement your lectures.
- Culminate the course with a research project assignment that allows your readers to apply what they've learned to linguistic content of their choice.
- You may consider using the RStudio Cloud platform to host the course, but ensure that your readers have access to R and RStudio on their own computers as well.

Advanced Introduction

- Cover all 12 chapters to give your readers a thorough understanding of quantitative text analysis concepts and techniques. Devote more time chapters 5-10 providing demonstrations of how to approach different problems and evaluating alternative approaches.
- Culminate the course with a collaborative research project that requires your readers to work in groups to conduct a comprehensive analysis of a given dataset.

- Ensure that your readers install R and RStudio on their own computers as they will need full control over their coding environment.

For all course designs, it is strongly recommend that you evaluate the readers' success in understanding the material by providing a combination of quizzes, lab assignments, programming exercises, and written reports. Additionally, encourage your readers to ask questions²⁷, collaborate with peers, and seek help from the ample resources available online when they encounter scope-limited programming problems.

For more information on how to use this textbook in your course, visit the Instructor Guide²⁹ on the companson website.

Activities

At this point you should have a working R environment with the core packages including `qtalrkit` installed. You should also have verified that you have a working Git environment and that you have a GitHub account. If you have not completed these tasks, return to the guides listed above in “Getting started” of this Preface and complete them before proceeding.

The following activities are designed to help you become familiar with the tools and resources that you will be using throughout this textbook. These and subsequent activities are designed to be completed in the order that they are presented in this textbook.

➤_ Swirl lesson

What: Intro to Swirl^a

How: In the R console load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: To familiarize you with navigating, selecting, and completing swirl lessons.

^a<https://github.com/qtalr/lessons>

✍ Recipe

What: Literate Programming: writing with code^a

How: Read Recipe 0 and participate in collaborative discussion with peers.

Why: To introduce the concept of Literate Programming and how to create literate documents using R and Quarto.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-0.html>

²⁷If you are using this textbook in a course, consider using a CMS (e.g. Canvas, Blackboard, etc.) or the web-based social annotation tool Hypothes.is²⁸ to facilitate reader questions and discussion.

²⁹<https://qtalr.github.io/qtalrkit/articles/instructor-guide.html>

⚠ Lab

What: Literate programming I^a

How: Clone, fork, and complete the steps in Lab 0.

Why: To put literate programming techniques covered in Recipe 0 into practice. Specifically, you will create and edit a Quarto document and render a report in PDF format.

^a<https://github.com/qtalr/lab-0>

Summary

In the Preface, we lay the groundwork by introducing the textbook's underlying principles, learning goals, teaching methods, and target audience. The chapter also offers advice on how to navigate the book's layout, comprehend its subject matter, and make use of supplementary materials. Crucial insights from this section involve grasping the book's objectives and aims, which center around instructing readers on quantitative text analysis for linguistics using R while emphasizing reproducible research. This chapter assists readers in setting up a working R development environment ensuring they can effectively engage with the material. Moreover, the Preface provides guidance on how to get help with R and other related software tools and deciphering conventions in the text. With this foundation, you're now prepared to delve into the captivating realm of text analysis in the subsequent chapter, titled "Text Analysis in Context."

Questions

🖊 Revise/ add questions.

Conceptual questions

- How is the textbook designed to be accessible for both novice and seasoned practitioners in the area of quantitative text analysis?
- What is the purpose of the textbook and what are the three areas it aims to scaffold?
- What are the main components of each chapter, and how are they structured to support learning outcomes?
- How does the structure of the textbook and associated resources work to support learning and proficiency in areas?
- What is the role of programmatic approaches in quantitative text analysis?
- What is the relationship between R and an IDE (e.g. RStudio, VS Code)?
- What is the relationship between R and a version control system (e.g. Git)?

Technical exercises

- Install the latest version of R by following the instructions for your operating system.
<https://cran.r-project.org/>
- Install RStudio Desktop <https://www.rstudio.com/products/rstudio/download/>
- Verify a Git installation or install Git (Windows: <https://git-scm.com/downloads>). Git a version control system that allows you to track changes to files and collaborate with others through GitHub.
- Create a free GitHub account at <https://github.com/join>.
- Install the `tidyverse` package in R by running `install.packages("tidyverse")` in the R Console pane.
- Install the `swirl` package by running `install.packages("swirl")` in the R Console pane.
- Open RStudio and create a new project for this textbook. This will help you keep your code and files organized.

Part I

Orientation

In this section the aims are to: 1) provide an overview of quantitative research and their applications, by both highlighting visible applications and notable research in various fields, 2) consider how quantitative research contributes to language research, and 3) layout the main types of research and situate quantitative text analysis inside these.

-  Update the overview of Part I “Orientation” to reflect the new structure of the chapter.

1 Text analysis in context



Draft

Ready for review.

Everything about science is changing because of the impact of information technology and the data deluge.

— Jim Gray



Outcomes

- Understand the role and goals of data analysis both within and outside of academia.
- Describe the various approaches to quantitative language research.
- Identify the applications of text analysis in different contexts.

In this chapter I will aim to introduce the topic of text analysis and provide the context needed to understand how text analysis fits in a larger universe of science and the ever-ubiquitous methods of data science, with attention to how linguistics and language-related studies employ data analysis down to the particular area of text analysis.



Swirl lesson

What: Variables and vectors, Workspace^a

How: In an R console load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: To explore some key building blocks of the R programming language and to examine your local workspace in R and understand the relationship between your R workspace and the file system of your computing environment.

^a<https://github.com/qtalr/lessons>

1.1 Making sense of a complex world

1.1.1 Heuristic Understanding

The world around us is full of actions and interactions so numerous that it is difficult to really comprehend. As each individual sees and experiences this world, we gain knowledge and build up heuristic understanding about how it works and how we can interact with it. This happens regardless of your educational background. As humans we are built for this. Our minds process countless sensory inputs. They underlie skills and abilities that we take for granted like being able to predict what will happen if you see someone about to knock a wine glass off a table and onto a concrete floor. You've never seen this object before and this is the first time you've been to this winery, but somehow and from somewhere you 'instinctively' make an effort to warn the would-be-glass-breaker before it is too late. You most likely have not stopped to consider where this predictive knowledge comes from, or if you have, you may have just chalked it up to 'common sense'. As common as it may be, it is an incredible display of the brain's capacity to monitor your environment, relate the events and observations that take place, and store that information all the time not making a big fuss to tell your conscious mind what it's up to.

So wait, this is a textbook on text analysis, right? So what does all this have to do with that? Well, there are two points to make that are relevant for framing our journey: (1) the world is constantly churning out data in real-time at a scale that is daunting and (2) for all the power of the brain that works so efficiently behind the scene making sense of the world, we are one individual living one life that has a limited view of the world at large. Let me expand on these two points a little more.

First let's be clear. There is no way for anyone to experience all things at all times. But even extremely reduced slices of reality are still vastly outside of our experiential capacity, at least in real-time. One can make the point that since the inception of the internet an individual's ability to experience larger slices of the world has increased. But could you imagine reading, watching, and listening to every file that is currently accessible on the web? Or has been? (See the Wayback Machine¹.) Scale this down even further; let's take Wikipedia, the world's largest encyclopedia. Can you imagine reading every wiki entry? As large as a resource such as Wikipedia is², it is still a small fragment of the written language that is produced on the web, just the web⁴. Consider that for a moment.

To my second framing point, which is actually two points in one. I underscored the efficiency of our brain's capacity to make sense of the world. That efficiency comes from some clever evolutionary twists that lead our brain to take in the world but it makes some shortcuts that compress the raw experience into heuristic understanding. What that means is that the brain is not a supercomputer. It does not store every experience in raw form, we do not have

¹<https://web.archive.org/>

²As of 22 July 2021, there are 6,341,359 articles in the English Wikipedia³ containing over 3.9 billion words occupying around 19 gigabytes of information.

⁴For reference, Common Crawl⁵ has millions of gigabytes collected since 2008.

access to the records of our experience like we would imagine a computer would have access to the records logged in a database. Where our brains do excel is in making associations and predictions that help us (most of the time) navigate the complex world we inhabit. This point is key –our brains are doing some amazing work, but that work can give us the impression that we understand the world in more detail than we actually do. Let's do a little thought experiment. Close your eyes and think about the last time you saw your best friend. What were they wearing? Can you remember the colors? If you're like me, or any other human, you probably will have a pretty confident feeling that you know the answers to these questions and there is a chance you're right. But it has been demonstrated in numerous experiments on human memory that our confidence does not correlate with accuracy (Talarico and Rubin 2003; Roediger and McDermott 2000). You've experienced an event, but there is no real reason that we should bet our lives on what we experienced. It's a little bit scary, for sure, but the magic is that it works 'good enough' for practical purposes.

So here's the deal: as humans we are (1) clearly unable to experience large swaths of experience by the simple fact that we are individuals living individual lives and (2) the experiences we do live are not recorded in memory with perfect precision and therefore we cannot 'trust' our intuitions, at least not in an absolute sense.

💡 Consider this

How might your own experiences and biases influence your understanding of the world? What are some ways that you can mitigate these biases? Is it ever possible to be completely objective? How might biases influence the way you approach text analysis?

1.1.2 Science to advance understanding

What does that mean for our human curiosity about the world around us and our ability to reliably make sense of it? In short it means that we need to approach understanding our world with the tools of science. Science starts with a question, identifies and collects data, carefully selected slices of the complex world, submits this data to analysis through clearly defined and reproducible procedures, and reports the results for others to evaluate. This process is repeated, modified, and manipulated the procedures, asking new questions and positing new explanations, all in an effort to make inroads to bring the complex into tangible view.

In essence what science does is attempt to subvert our inherent limitations in understanding by drawing on carefully and purposefully collected slices of observable experience and letting the analysis of these observations speak, even if it goes against our intuitions (those powerful but sometimes spurious heuristics that our brains use to make sense of the world).

1.2 Data analysis

1.2.1 Emergence of data science

At this point I've sketched an outline strengths and limitations of humans' ability to make sense of the world and why science is used to address these limitations. This science I've described is the one you are familiar with and it has been an indispensable tool to make sense of the world. If you are like me, this description of science may be associated with visions of white coats, labs, and petri dishes. While science's foundation still stands strong in the 21st century, a series of intellectual and technological events mid-20th century set in motion changes that have changed aspects about how science is done, not why it is done. We could call this Science 2.0, but let's use the more popularized term **data science**. The recognized beginnings of data science are attributed to work in the "Statistics and Data Analysis Research" department at Bell Labs during the 1960s. Although primarily conceptual and theoretic at the time, a framework for quantitative data analysis took shape that would anticipate what would come: sizable datasets which would "[...] require advanced statistical and computational techniques [...] and the software to implement them." (Chambers 2020) This framework emphasized both the inference-based research of traditional science, but also embraced exploratory research and recognized the need to address practical considerations that would arise when working with and deriving insight from an abundance of machine-readable data.

Fast-forward to the 21st century a world in which machine-readable data is truly in abundance. With increased computing power, the emergence of the world wide web, and wide adoption of mobile devices electronic communication skyrocketed around the globe. To put this in perspective, in 2019 it was estimated that every minute 511 thousand tweets were posted, 18.1 million text messages were sent, and 188 million emails were sent ("Data Never Sleeps 7.0 Infographic" 2019). The data flood has not been limited to language, there are more sensors and recording devices than ever before which capture evermore swaths of the world we live in (Desjardins 2019). Where increased computing power gave rise to the influx of data, it is also one of the primary methods for gathering, preparing, transforming, analyzing, and communicating insight derived from this data (Donoho 2017). The vision laid out in the 1960s at Bell Labs had come to fruition.

1.2.2 Computing skills, statistical knowledge, and domain knowledge

Data science is not predicated on data alone. Turning data into insight takes **computing skills** (i.e. programming), **statistical knowledge**, and **domain expertise**. This triad has been popularly represented as a Venn diagram such as in Figure 1.1.

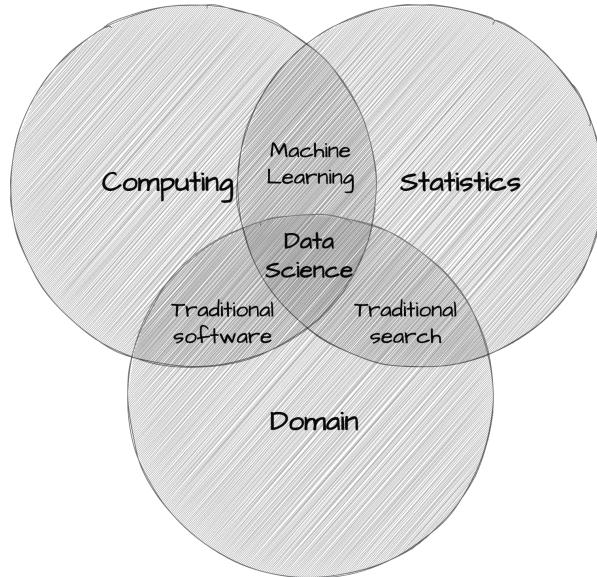


Figure 1.1: Data Science Venn Diagram adapted from Drew Conway⁶.

The **computing skills** component of data science is the ability to write code to perform the data analysis process. This is the primary approach for working with data at scale. The **statistical knowledge** component of data science is the ability to apply statistical methods to data to derive insight. **Domain expertise** provides researchers insight at key junctures in the development of a research project and aid researchers in evaluating results.

This triad of skills in combination with reproducible research practices is the foundational toolbelt of data science (Hicks and Peng 2019). **Reproducible research** entails the use of computational tools to automate the process of data analysis. This automation is achieved by writing code that can be executed to replicate the data analysis. This code can then be shared through code sharing repositories, such as GitHub, where it can be viewed, downloaded, and executed by others. This adds transparency to the process and allows others to build on previous work. This is in contrast to traditional approaches where data analysis is performed (semi-)manually, results are reported in a static document such as a report or journal article, and the data analysis process is not shared. This approach is not reproducible because the data analysis process is not transparent and cannot be replicated. This is problematic because it is difficult to evaluate the results and build on previous work. Reproducible research practices are a key component of data science and are emphasized throughout this book.

1.2.3 Applications of data science

Equipped with the data science toolbelt, the interest in deriving insight from the available data is now almost ubiquitous. The science of data has now reached deep into all aspects of life

where making sense of the world is sought. Predicting whether a loan applicant will get a loan (Bao, Lianju, and Yue 2019), whether a lump is cancerous (Saxena and Gyanchandani 2020), what films to recommend based on your previous viewing history (Gomez-Uribe and Hunt 2015), what players a sports team should sign (Lewis 2004) all now incorporate a common set of data analysis tools.

The data science toolbelt also underlies well-known public-facing language applications. From the language-capable chat applications, plagiarism detection software, machine translation algorithms, and search engines, tangible results of quantitative approaches to language are becoming standard fixtures in our lives, as seen in Figure 1.2.

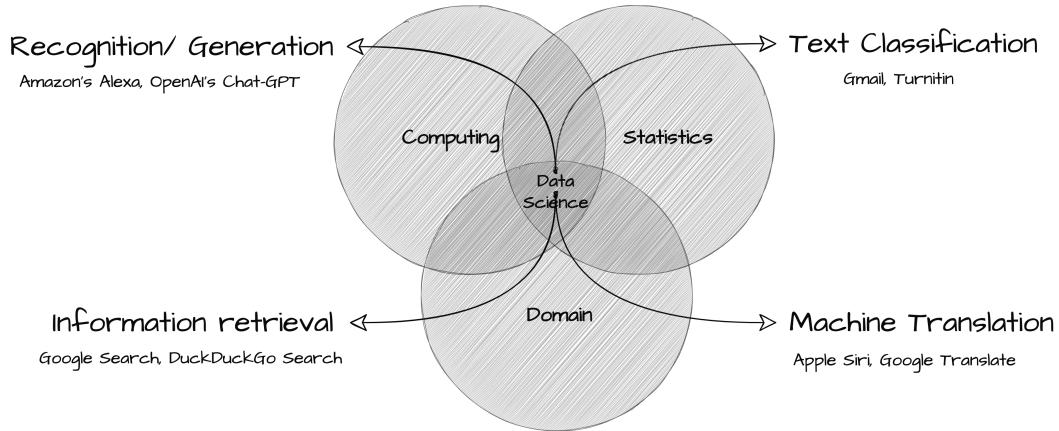


Figure 1.2: Well-known public-facing language applications

The spread of quantitative data analysis too has taken root in academia. Even in areas that on first blush don't appear readily approachable in a quantitative manner, such as fields in the social sciences and humanities, data science is making important and sometimes disciplinary changes to the way that academic research is conducted. This textbook focuses in on a domain that cuts across many of these fields; namely language. At this point let's turn to quantitative approaches to language analysis as we work closer to contextualizing text analysis.

1.3 Language analysis

Language is a defining characteristic of our species. Since antiquity, language has attracted interest across disciplines and schools of thought. In the early 20th century, the development of the rigorous approach to study of language as a field in its own right took root (Campbell 2001), yet a plurality of theoretical views and methodological approaches remained. Contemporary linguistics bares this complex history and is far from theoretically and methodologically unified.

Either based on the tenets of theoretical frameworks and/or the objects of study of particular fields, approaches to language research vary. On the one hand some language research commonly applies qualitative assessment of language structure and/ or use. **Qualitative approaches** describe and account for characteristics, or “qualities”, that can be observed, but not measured (*e.g.* introspective methods, ethnographic methods, *etc.*)

On the other hand other language research programs employ quantitative research methods either out of necessity given the object of study (phonetics, psycholinguistics, *etc.*) or based on theoretical principles (Cognitive Linguistics, Connectionism, *etc.*). **Quantitative approaches** involve measurements of properties of language that can be observed and measured (*e.g.* frequency of use, reaction time, *etc.*).

These latter research areas and theoretical paradigms employ methods that share much of the common data analysis toolbox described in the previous section. In effect, this establishes a common methodological language between other language research fields but also with research outside of linguistics.

However, there is never a one-size-fits all approach to anything –much less data analysis. And even in quantitative language analysis there is a key methodological distinction that has downstream effects in terms of procedure but also in terms of interpretation. The key distinction that we need to make at this point, which will provide context for our introduction to quantitative text analysis, comes down to the approach to collecting language data and the nature of that data. This distinction is between **experimental data** and **observational data**.

Experimental approaches start with a intentionally designed hypothesis and lay out a research methodology with appropriate instruments and a plan to collect data that shows promise for shedding light on the validity of the hypothesis. Experimental approaches are conducted under controlled contexts, usually a lab environment, in which participants are recruited to perform a language related task with stimuli that have been carefully curated by researchers to elicit some aspect of language behavior of interest. Experimental approaches to language research are heavily influenced by procedures adapted from psychology. This link is logical as language is a central area of study in cognitive psychology. This approach looks much like the white-coat science that we made reference to earlier but, as in most quantitative research, has now taken advantage of the data analysis toolbelt to collect and organize much larger quantities of data and conduct statistically more robust analysis procedures and communicate findings more efficiently.

Observational approaches are a bit more of a mixed bag in terms of the rationale for the study; they may either start with a testable hypothesis or in other cases may start with a more open-ended research question to explore. But a more fundamental distinction between the two is drawn in the amount of control the researcher has on contexts and conditions in which the language behavior data to be collected is produced. Observational approaches seek out records of language behavior that is produced by language speakers for communicative purposes in natural(istic) contexts. This may take place in labs (language development, language disorders,

etc.), but more often than not, language is collected from sources where speakers are performing language as part of their daily lives –whether that be posting on social media, speaking on the telephone, making political speeches, writing class essays, reporting the latest news for a newspaper, or crafting the next novel destined to be a New York Times best-seller. What is more, data collected from the ‘wild’ varies more in structure relative to data collected in experimental approaches and requires a number of steps to prepare the data to sync up with the data analysis toolbelt.

I liken this distinction between experimental and observational data collection to the difference between farming and foraging. Experimental approaches are like farming; the groundwork for a research plan is designed, much as a field is prepared for seeding, then the researcher performs a series of tasks to produce data, just as a farmer waters and cares for the crops, the results of the process bear fruit, data in our case, and this data is harvested. Observational approaches are like foraging; the researcher scans the available environmental landscape for viable sources of data from all the naturally existing sources, these sources are assessed as to their usefulness and value to address the research question, the most viable is selected, and then the data is collected.

The data acquired from both of these approaches have their trade-offs, just as farming and foraging. Experimental approaches directly elicit language behavior in highly controlled conditions. This directness and level of control has the benefit of allowing researchers to precisely track how particular experimental conditions effect language behavior. As these conditions are an explicit part of the design and therefore the resulting language behavior can be more precisely attributed to the experimental manipulation. The primary shortcoming of experimental approaches is that there is a level of artificialness to this directness and control. Whether it is the language materials used in the task, the task itself, or the fact that the procedure takes place under supervision the language behavior elicited can diverge quite significantly from language behavior performed in natural communicative settings. Observational approaches show complementary strengths and shortcomings.

Whereas experimental approaches may diverge from natural language use, observational approaches strive to identify and collect language behavior data in natural, uncontrolled, and unmonitored contexts, as seen in Figure 1.3. In this way observational approaches do not have to question to what extent the language behavior data is or is not performed as a natural communicative act. On the flipside, the contexts in which natural language communication take place are complex relative to experimental contexts. Language collected from natural contexts are nested within the complex workings of a complex world and as such inevitably include a host of factors and conditions which can prove challenging to disentangle from the language phenomenon of interest but must be addressed in order to draw reliable associations and conclusions.

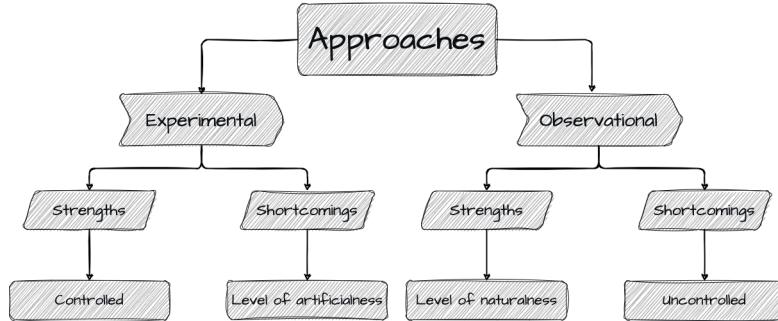


Figure 1.3: Trade-offs between experimental and observational data collection methods.

The upshot, then, is twofold: (1) data collection methods matter for research design and interpretation and (2) there is no single best approach to data collection, each have their strengths and shortcomings. In the ideal, a robust science of language will include insight from both experimental and observational approaches (Gilquin and Gries 2009). And evermore there is greater appreciation for the complementary nature of experimental and observational approaches and a growing body of research which highlights this recognition.

Case study

Manning (2003) discusses the use of probabilistic models in syntax to account for the variability in language usage and the presence of both hard and soft constraints in grammar. The paper touches on the statistical methods in text analysis, the importance of distinguishing between external and internal language, and the limitations of Generative Grammar. Overall, the paper suggests that usage-based and formal syntax can learn from each other to better understand language variation and change.

Given their particular trade-offs observational data is often used as an exploratory starting point to help build insight and form predictions that can then be submitted to experimental conditions. In this way, studies based on observational data serve as an exploratory tool to gather a better and more externally valid view of language use which can then serve to make prediction that can be explored with more precision in an experimental paradigm. However, this is not always the case; observational data is also often used in hypothesis-testing contexts as well. And furthermore, some in some language-related fields, a hypothesis-testing is not the approach for deriving knowledge and insight.

1.4 Text analysis

In a nutshell, **text analysis** is the process of leveraging the data science toolbelt to derive insight from textual data collected through observational methods. In the next subsections,

I will unpack this definition and discuss the primary components that make up text analysis including research approaches and technical implementation, as well as practical applications.

1.4.1 Approaches

Text analysis is a multifaceted research methodology. It can be used to facilitate the qualitative exploration of smaller, human-digestable textual information, but is more often employed quantitatively to bring to the surface patterns and relationships in large samples of textual data that would be otherwise difficult, if not impossible, to identify manually.

Text being text, there are a series of **data preparation** steps that must be taken to ready the data for analysis. In addition to collecting the data, the data must be organized, cleaned, and transformed into a format that is amenable to statistical analysis.

The statistical and evaluative approach employed in the analysis is dependent on the aim of the research. For research aimed at exploring and uncovering patterns and relationships in a data-driven manner, **Exploratory Data Analysis** (EDA) is employed. EDA combines descriptive statistics, visualizations, and statistical learning methods in an iterative and interactive way to provide the researcher the ability to identify patterns and relationships and to evaluate whether and why they are meaningful.

Predictive Data Analysis (PDA), applied in research for outcome prediction, is a supervised machine learning task. It uses feature sets to predict an outcome variable. Its primary evaluation metric is the prediction accuracy on new data. However, for many text analysis tasks, human interpretation is crucial to provide context and assess the significance of the results.

Research aimed at explaining relationships between variables and the population from which the sample was drawn will adopt an **Inferential Data Analysis** (IDA) approach. IDA is a theory-driven process that employs statistical models for hypothesis testing. The extent to which the results can be confidently generalized to the population is the primary evaluation metric.

As we see, text analysis can be used for a variety of purposes; from data-driven exploration and discovery to hypothesis testing and generalization.

1.4.2 Implementation

To ensure that the results of text analysis projects are replicable and transparent, programming strategies play an integral role at each stage of the implementation of a research project. While there are a number of programming languages that can be used for text analysis, R is widely adopted in linguistics research. R is a free and open-source programming language that is specifically designed for statistical computing and graphics. It has a large and active community of users and developers, and a robust ecosystem of packages which make it a

powerful and flexible language that is well-suited for core text analysis tasks: data collection, organization, transformation, analysis, and visualization. When combined with Quarto for literate programming and GitHub for version control and collaboration, R provides a robust and reproducible workflow for text analysis.

1.4.3 Applications

So what are some applications of text analysis? Most public facing applications stem from Computational Linguistic research, often known as **Natural Language Processing** (NLP) by practitioners. Whether it be using search engines, online translators, submitting your paper to plagiarism detection software, *etc.* many of the text analysis methods we will cover are at play.

💡 Consider this

What are some other public facing applications of text analysis that you are aware of?

In academia the use of quantitative text analysis is even more widespread, despite the lack of public fanfare. In linguistics, text analysis is applied to a wide range of topics and research questions in both theoretical and applied subfields, as seen in Example 1.1 and Example 1.2.

Example 1.1. Theoretical linguistics

- Hay (2002) use a corpus study to investigate the role of frequency and phonotactics in affix ordering in English.
- Riehemann (2001) explores the extent to which idiomatic expressions (*e.g.* ‘raise hell’) are lexical or syntactic units.
- Bresnan (2007) investigate the claim that possessed deverbal nouns in English (*e.g.* ‘the city’s destruction’) are subject to a syntactic constraint that requires the possessor to be affected by the action denoted by the deverbal noun.

Example 1.2. Applied linguistics

- Wulff, Stefanowitsch, and Gries (2007) explore differences between British and American English at the lexico-syntactic level in the *into*-causative construction (*e.g.* ‘He tricked me into employing him.’).
- Eisenstein et al. (2012) track the geographic spread of neologisms (*e.g.* ‘bruh’, ‘af’, ‘_____’) from city to city in the United States using Twitter data collected between 6/2009 and 5/2011.
- Bychkovska and Lee (2017) investigates possible differences between L1-English and L1-Chinese undergraduate students’ use of lexical bundles, multiword sequences which are extended collocations (*e.g.* ‘as the result of’), in argumentative essays.

- Jaeger and Snider (2007) use a corpus study to investigate the phenomenon of syntactic persistence, the increased tendency for speakers to use a particular syntactic form over an alternate when the syntactic form has been recently processed.
- Voigt et al. (2017) explore potential racial disparities in officer respect in police body camera footage.
- Olohan (2008) investigate the extent to which translated texts differ from native texts do to ‘explicitation’.

So too, text analysis is used in a variety of other fields where insight from language is sought, as seen in Example 1.3.

Example 1.3. Language-related fields

- Kloumann et al. (2012) explore the extent to which languages are positively, neutrally, or negatively biased.
- Mosteller and Wallace (1963) provide a method for solving the authorship debate surrounding The Federalist papers.
- Conway et al. (2012) investigate whether the established drop in language complexity of rhetoric in election seasons is associated with election outcomes.

💡 Consider this

Language is a key component of human communication and interaction. What are some other areas of research in and outside linguistics that you think could be explored using text analysis methods?

These studies in Examples 1.1, 1.2, and 1.3 are just a few illustrations of the contributions of text analysis as the primary method to gain a deeper understanding of language structure, function, variation, and acquisition. As a method, however, text analysis can also be used to support other research methods. For example, text analysis can be used collect data, generate authentic materials, provide linguistic annotation, to generate hypotheses, for either qualitative and/ or quantitative approaches. Together these efforts contribute to a more robust language science by incorporating externally valid data and providing methodological triangulation (Francom 2022).

In sum, the applications highlighted in this section underscore the versatility of text analysis as a research method. Whether it be in the public sphere or in academia, text analysis methods furnish a set of powerful tools for gaining insight from language data.

Summary

In this chapter I started with some general observations about the difficulty of making sense of a complex world. The standard approach to overcoming inherent human limitations in sense

making is science. In the 21st century the toolbelt for doing scientific research and exploration has grown in terms of the amount of data available, the statistical methods for analyzing the data, and the computational power to manage, store, and share the data, methods, and results from quantitative research. The methods and tools for deriving insight from data have made significant inroads in and outside academia, and increasingly figure in the quantitative investigation of language. Text analysis is a particular branch of this enterprise based on observational data from real-world language and is used in a wide variety of fields.

In the end I hope that you enjoy this exploration into text analysis. Although the learning curve at times may seem steep –the experience you will gain will not only improve your data literacy, research skills, and programmings skills but also enhance your appreciation for the richness of human language and its important role in our everyday lives.

Actitvies

The following activities build on your introduction to R and Quarto in the previous chapter. In these activities you will uncover more features offered by Quarto which will enhance your ability to produce comprehensive reproducible research documents. You will apply the capabilities of Quarto in a practical context conveying the objectives and key discoveries from a primary research article.

_recipe

What: Academic writing with Quarto^a

How: Read Recipe 1 and participate in the Hypothes.is online social annotation.

Why: To explore additional functionality in Quarto: numbered sections, table of contents, in-line citations and a document-final references list, and cross-referenced tables and figures.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-.html>

_lab

What: Literate programming II^a

How: Clone, fork, and complete the steps in Lab 1.

Why: To put into practice Quarto functionality to communicate the aim(s) and main finding(s) from a primary research article and to interpret a related plot.

^a<https://github.com/qtalr/lab-1>

Questions

Conceptual questions

- How has scientific research and exploration changed in the 21st century?
- What are the three basic skill sets that make up the data science toolbelt?
- What are the benefits of reproducible research in data science?
- Explain the trade-offs between experimental and observational data collection methods.
- What is text analysis and how is it used in various fields?
- Identify research in an area of interest in linguistics that has taken a quantitative approach to text analysis.
- In your own words, define literate programming?
- What are the benefits of literate programming?
- What are the benefits of using R and Quarto for literate programming?

Technical exercises

- Create a literate programming document in Quarto. Edit the yaml header to reflect details of the work and add your work with the data types in R to code chunks. Add, commit, and push the project to GitHub.
- In the Quarto document, explore using R to create vectors and explore their properties.
- Explore the following resources and with the goal to identify a quantitative text analysis project. Rpubs^a, GitHub^b, DataCamp^c, Kaggle^d, R-bloggers^e.
- ↗ ... more to come ...

^a<https://rpubs.com/>

^b<https://github.com>

^c<https://datacamp.com>

^d<https://kaggle.com>

^e<https://r-bloggers.com>

Part II

Foundations

Before working on the specifics of a data project, it is important to establish a fundamental understanding of the characteristics of each of the levels in the “Data, Information, Knowledge, and Insight Hierarchy (DIKI)” (see Figure 1) and the roles each of these levels have in deriving insight from data. In Chapter 2 we will explore the Data and Information levels drawing a distinction between two main types of data (populations and samples) and then cover how data is structured and transformed to generate information (datasets) that is fit for statistical analysis. In Chapter 3 I will outline the importance and distinct types of statistical procedures (descriptive and analytic) that are commonly used in text analysis. Chapter 4 aims to tie these concepts together and cover the required steps for preparing a research blueprint to conduct an original text analysis project.

2 Understanding data



Draft

Ready for review.

The goal is to turn data into information, and information into insight.

— Carly Fiorina

▀ Outcomes

- Describe the difference between data and information.
- Understand how the tidy approach to data organization can enhance the quality and usability of data.
- Articulate the importance of documentation in promoting reproducible research.

In this chapter, the groundwork is laid for deriving insights from text analysis by focusing on content and structure of data and information. The concepts of populations and samples are introduced, highlighting their similarities and key differences. Connecting these topics to text analysis, language samples, or corpora, are explored, discussing their types, sources, formats, and ethical considerations. Subsequently, key concepts in creating information from data, such as organization and transformation, are examined. Finally, the importance of documentation in quantitative research is emphasized through addressing data origin and data dictionaries.

➤_ Swirl lesson

What: Objects, Packages and functions^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: To introduce you to the main types of objects in R and to understand the role and use of functions and packages in R programming.

^a<https://github.com/qtalr/lessons>

2.1 Data

Data is data, right? The term ‘data’ is so common in popular vernacular it is easy to assume we know what we mean when we say ‘data’. But as in most things, where there are common assumptions there are important details that require more careful consideration. Let’s turn to the first key distinction that we need to make to start to break down the term ‘data’: the difference between populations and samples.

2.1.1 Populations

The first thing that comes to many people’s mind when the term population is used is human populations (derived from Latin ‘*populus*’). Say for example we pose the question –What’s the population of Milwaukee? When we speak of a population in these terms we are talking about the total sum of individuals living within the geographical boundaries of Milwaukee. In concrete terms, a **population** an idealized set of objects or events in reality which share a common characteristic or belong to a specific category. The term to highlight here is idealized. Although we can look up the US Census report for Milwaukee and retrieve a figure for the population, this cannot truly be the population. Why is that? Well, whatever method that was used to derive this numerical figure was surely incomplete. If not incomplete, by the time someone recorded the figure some number of residents of Milwaukee moved out, moved in, were born, or passed away. In either case, this example serves to point out that populations are not fixed and are subject to change over time.

Likewise when we talk about populations in terms of language we dealing with an idealized aspect of linguistic reality. Let’s take the words of the English language as an analog to our previous example population. In this case the words are the people and English is the grouping characteristic. Just as people, words move out, move in, are born, and pass away. Any compendium of the words of English at any moment is almost instantaneously incomplete. This is true for all populations, save those relatively rare cases in which the grouping characteristics select a narrow slice of reality which is objectively measurable and whose membership is fixed (the complete works of Shakespeare, for example).

In sum, (most) populations are amorphous moving targets. We subjectively hold them to exist, but in practical terms we often cannot nail down the specifics of populations. So how do researchers go about studying populations if they are theoretically impossible to access directly? The strategy employed is called sampling.

2.1.2 Samples

A **sample** is the product of a subjective process of selecting a finite set of observations from an idealized population with the goal of capturing the relevant characteristics of the target population. The **degree of representativeness** of a sample is the extent to which the sample

reflects the characteristics of the population. The degree of representativeness is crucial for research as it directly impacts of any findings based on the sample.

To maximize the representativeness of a sample, researchers employ a variety of strategies. One of the first and sometimes the easiest strategy is to increase the **sample size**. A larger sample will always be more representative than a smaller sample. Sample size, however, is often not enough. It is not hard to imagine a large sample which by chance captures only a subset of the features of the population. Another step to enhance sample representativeness is to apply **random sampling**. Together a large random sample has an even better chance of reflecting the main characteristics of the population better than a large or random sample. But, random as random is, we still run the risk of acquiring a skewed sample (*i.e.* a sample with low representativeness).

To help mitigate these issues, there are two more strategies that can be applied to improve sample representativeness. Note, however, that while size and random samples can be applied to any sample with few assumptions about internal characteristics of the population, these next two strategies require decisions depend on the presumed internal characteristics of the population.

The first of these more informed sampling strategies is called **stratified sampling**. Stratified samples make (educated) assumptions about sub-components within the population of interest. With these sub-populations in mind, large random samples are acquired for each sub-population, or strata. At a minimum, stratified samples can be no less representative than random sampling alone, but the chances that the sample is better increases. Can there be problems in the approach? Yes, and on two fronts. First knowledge of the internal components of a population are often based on a limited or incomplete knowledge of the population (remember populations are idealized). In other words, strata are selected subjectively by researchers using various heuristics some of which are based on some sense of ‘common knowledge’.

The second front on which stratified sampling can err concerns the relative sizes of the sub-components relative to the whole population, which is known as **balance**. Even if the relevant sub-components are identified, their relative size adds another challenge which researchers must address in order to maximize the representativeness of a sample.

Together, large randomly selected and balanced stratified samples set the benchmark for sampling. However, hitting this ideal is not always feasible. There are situations where sizeable samples are not accessible. Alternatively, there may be instances where the population or its strata are not well understood. In such scenarios, researchers have to work with the most suitable sample they can obtain given the limitations of their research project.

2.1.3 Corpora

A key feature of a sample is that it is purposely selected to model a target population. In text analysis, a purposely sampled collection of texts, of the type defined here, is known as a

corpus (*pl.* corpora). A set of texts or documents which have not been selected purposely lack a **sampling frame**, and therefore is not a corpus. The sampling frame, hence the populations modeled, in any given corpus will vary. It is key to vet corpora to ensure that the resource's sampling frame and the research project's target populations align as closely as possible to safeguard the integrity of research findings later in the research process.

💡 Consider this

The 'Standard Sample of Present-Day American English' (known commonly as the Brown Corpus) is widely recognized as one of the first large, machine-readable corpora. Compiled by Kucera and Francis (1967), the corpus is comprised of 1,014,312 words from edited English prose published in the United States in 1961. Given the sampling frame for this corpus visualized in Figure 2.1, can you determine what language population this corpus aims to represent? What types of research might this corpus support or not support?

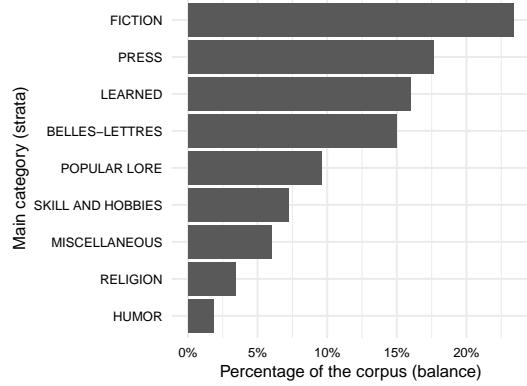


Figure 2.1: Overview of the sampling frame of the Brown Corpus.

Types

Let's take a look at some key characteristics, attributes, and features that distinguish corpora.

Reference

The least common and most ambitious corpus resources are those which aim to model the characteristics of a language population. These are known as **reference corpora**. These are projects designed with wide sampling frames, and require significant investments of time in corpus design and implementation (and continued development) that are usually undertaken by research teams (Ädel 2020).

The American National Corpus (ANC)¹ or the British National Corpus (BNC)² are corpora which aim to model the general characteristics of a variety of the English language, the former

¹<https://www.anc.org/>

²<http://www.natcorp.ox.ac.uk/>

of American English and the later British English. Reference corpora exist for other languages as well: Spanish Reference Corpus of Present-Day Spanish (CREA)³, German The German Reference Corpus (DeReKo)⁴, Turkish Turkish National Corpus (TNC)⁵, and many others.

💡 Consider this

Of note is the fact that, at present, most of the world's languages lack reference corpus resources, or any corpus resources whatsoever. "Low-resourced" languages are often less studied, resource scarce, less available in born-digital formats, etc. (Magueresse, Carles, and Heetderks 2020).

Visit the Clarin overview^a on reference corpora and then visit LRE Map^b. Can you find a reference corpus for a language you speak or are interested in studying? If not, consider what can be done to address this gap in the research community.

^a<https://www.clarin.eu/resource-families/reference-corpora>

^b<https://lremap.elra.info/>

Specialized

Specialized corpora aim to represent more specific populations. The population may be defined either by modality, genre, time, location, or speaker-oriented characteristics, or some combination thereof. What specialized corpora lack in breadth of coverage, they make up for in depth of coverage by providing a more targeted representation of specific language populations.

The Santa Barbara Corpus of Spoken American English (SBCSAE)⁶, as you can imagine from the name of the resource, aims to model spoken American English. No claim to written English is included. There are even more specific types of corpora which attempt to model other types of sub-populations such as academic writing⁷, computer-mediated communication (CMC)⁸, language use in specific regions of the world⁹, a country¹⁰, a region of a country¹¹, etc.

💡 Consider this

³<http://corpus.rae.es/creanet.html>

⁴<https://www.ids-mannheim.de/digspra/kl/projekte/korpora/>

⁵<https://www.tnc.org.tr/>

⁶<https://www.linguistics.ucsb.edu/research/santa-barbara-corpus>

⁷<https://www.coventry.ac.uk/research/research-directories/current-projects/2015/british-academic-written-english-corpus-bawe/>

⁸<https://www.clarin.eu/resource-families/cmc-corpora>

⁹<http://ice-corpora.net/ice/index.html>

¹⁰<https://www.wgtn.ac.nz/lals/resources/corpora-default/corpora-wsc>

¹¹<https://cesa.arizona.edu>

Grieve, Nini, and Guo (2018) compiled a 8.9 billion-word corpus of geotagged posts from Twitter between 2013-2014 in the United States. The authors provide a search interface^a to explore relationship between lexical usage and geographic location. Explore this corpus searching for terms related to slang (“hella”, “wicked”), geographical (“mountain”, “river”), meteorological (“snow”, “rain”), and/ or any other term types. What types of patterns do you find? What are the benefits and/ or limitations of this type of data and/ or interface?

^a<https://isogloss.shinyapps.io/isogloss/>

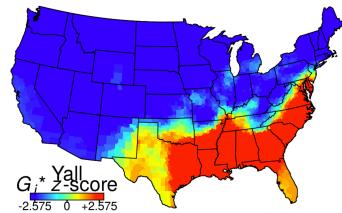


Figure 2.2: Example distribution of the term ‘Ya’ll’ the Word Mapper project.

Another set of specialized corpora are resources which aim to compile texts from different languages or different language varieties for direct or indirect comparison. Corpora that are directly comparable, that is they include source and translated texts, are called **parallel corpora**. Parallel corpora include different languages or language varieties that are indexed and aligned at some linguistic level (*i.e.* word, phrase, sentence, paragraph, or document), see OPUS¹². Corpora that are compiled with different languages or language varieties but are not directly aligned are called **comparable corpora**. The comparable language or language varieties are sampled with the same or similar sampling frame, for example Brown¹³ and LOB¹⁴ corpora.

The aim of the quantitative text researcher is to select the corpus, or corpora, which best align with the purpose of the research. For example, a general corpus such as the American National Corpus may be better suited to address a question dealing with the way American English works, but this general resource may lack detail in certain areas, such as medical language¹⁵, that may be vital for a research project aimed at understanding changes in medical terminology. Furthermore, a researcher studying spoken language might collect a corpus of transcribed conversations from a particular community or region, such as the SBCSAE. While this would not include every possible spoken utterance produced by members of that group, it could be considered a representative sample of the population of speech in that context.

Sources

Published

¹²<https://opus.nlpl.eu/>

¹³<https://ota.bodleian.ox.ac.uk/repository/xmlui/handle/20.500.12024/0402>

¹⁴<https://ota.bodleian.ox.ac.uk/repository/xmlui/handle/20.500.12024/0167>

¹⁵<https://mtsamples.com/index.asp>

The most common source of data used in contemporary quantitative research is the internet. On the web an investigator can access corpora published for research purposes. Many organizations exist around the globe that provide access to published corpora in browsable catalogs, or **repositories**. There are repositories dedicated to language research, in general, such as the Language Data Consortium¹⁶ or that specialize in specific domains, such as the spoken language repository TalkBank¹⁷. It is always advisable to start looking for the available language data in a repository. The advantage of beginning your data search in repositories is that a repository, especially those geared towards the linguistic community, will make identifying language corpora faster than through a general web search. Furthermore, repositories often require certain standards for corpus format and documentation for publication.

Repositories are by no means the only source of published corpora on the web. Researchers from around the world provide access to corpora and datasets on their own sites or through data sharing platforms. Corpora of various sizes and scopes will often be accessible on a dedicated homepage or appear on the homepage of a sponsoring institution. These resources may be available for download or via search interfaces. Finding these resources is often a matter of doing a web search with the word ‘corpus’ and a list of desired attributes, including language, modality, register, *etc.*

As part of a general movement towards reproducibility, more corpora are available on **data sharing platforms** such as GitHub¹⁸, Zenodo¹⁹, Re3data²⁰, OSF²¹, *etc.* These platforms enable researchers to securely store, manage, and share data with others. Support is provided for various types of data, including documents and code, and as such they are a good place to look as they often include reproducible research projects as well.

Develop

Language corpora prepared by researchers and research groups listed on repositories or hosted by the researchers themselves is often the first place to look for data. The web, however, contains a wealth of language and language-related data that can be accessed by researcher to compile their own corpus. There are two primary ways to attain language data from the web. The first is through an **Application Programming Interface** (API). APIs are, as the title suggests, programming interfaces which allow access, under certain conditions, to information that a website or database accessible via the web contains.

The second, more involved, way to acquire data from the web is through the process of web scraping. **Web scraping** is the process of harvesting data from the public-facing web. Language texts may be found on sites as uploaded files, such as pdf or doc (Word) documents,

¹⁶<https://www.ldc.upenn.edu/>

¹⁷<http://talkbank.org/>

¹⁸<https://github.com/>

¹⁹<https://zenodo.org/>

²⁰<http://www.re3data.org/>

²¹<https://osf.io/>

or found displayed as the primary text of a site. Given the wide variety of documents uploaded and language behavior recorded daily on news sites, blogs and the like, compiling a corpus has never been easier. Having said that, how the data is structured and how much data needs to be retrieved can pose practical obstacles to collecting data from the web, particularly if the approach is to acquire the data by manually instead of automating the task.

Dive deeper

The process of corpus development is a topic in and of itself. For a more in-depth discussion of the process, see Ädel (2020).

Consider this

Explore some of the resources listed on the qtalrkit compansion site^a and consider their sampling frames. Can you think of a research question or questions that this resource may be well-suited to support research into? What types of questions would be less-than-adequate for a given resource?

^a<https://qtalr.github.io/qtalrkit/articles/guide-5.html>

Ethical considerations

Just because data is available on the web does not mean it is free to use. Repositories, APIs, and individual data resources often have licensing agreements and terms of use, ranging from public domain to proprietary licenses. Public domain licenses, such as those found in Project Gutenberg, allow anyone to use the data for any purpose. Creative Commons licenses²², like those used by the American National Corpus, Wikipedia, and TalkBank, span from public domain to more restrictive uses, including requirements for attribution or prohibiting commercial use. Even more restrictive licenses, such as those for the Corpus of Contemporary American English and the British National Corpus, may require a fee to access and use the data, even for research purposes.

Respecting intellectual property rights is crucial when working with corpus data. Violating these rights can lead to legal and ethical issues, including lawsuits, fines, and damage to one's professional reputation. To avoid these problems, researchers must ensure they have the necessary permissions to use copyrighted works in their corpora. Obtaining permissions involves contacting the author or publisher and requesting consent to use their work for research purposes. Documenting all obtained permissions and providing attribution and/ or citation is essential respecting the intellectual property rights of others.

²²<https://creativecommons.org/about/cclicenses/>

Formats

Whether you are using a published corpus or developing your own, it is important to understand how the data you want to work with is formatted. When referring to the format of a corpus, this includes the folder and file structure, the file types, the internal structure of the files themselves, and how file content is encoded electronically.

Folder and file structure

Some corpus resources are contained in a single file, such as a spreadsheet or a text file, but more often than not a corpus will be comprised of multiple files and folders. The folder and file structure will reflect the organization of the corpus and may include sub-folders for different types or groupings of data. In addition to the corpus data itself, metadata and documentation will often be included in the corpus folder structure. The corpus data may be grouped by language, modality, register, or other attributes such as types of linguistic annotation.

To illustrate, in Example 2.1 we have the file and folder structure of a toy corpus.

Example 2.1. Toy corpus structure

```
corpus/
└── documentation/
    ├── README.md
    └── LICENSE
└── metadata/
    └── speakers.csv
└── data/
    ├── spoken/
    │   ├── inter-09-a.xml
    │   ├── inter-09-b.xml
    │   ├── convo-09-a.xml
    │   ├── ...
    └── written/
        ├── essay-09-a.xml
        ├── essay-09-b.xml
        ├── respo-09-a.xml
        ├── ...

```

In this example, we have a corpus folder with three sub-folders: *documentation/*, *metadata/*, and *data/*. The *data/* folder contains two sub-folders: *spoken/* and *written/*. Each folder contains the relevant data files.

Where a single file is easy to download from the web, a corpus with a more complex folder structure can be more difficult to access. For that reason, many corpus resources are packaged into and made into a single compressed file. **File compression** has two benefits: it preserves the folder structure in a format which is contained in a single file and it also reduces the overall storage size. Common file compression formats are *.zip* and *.tar.gz*. So a compressed corpus file for the example above may be named something like *corpus.zip* or *corpus.tar.gz*. To access the original data within a compressed file, one must use a decompression tool or software to extract the contents after downloading it.

File types

In our toy corpus example, you may have noticed that each of the filenames appear with either *.md*, *.csv*, *.xml*, or nothing appended. These are examples of **file extensions**. File extensions are a short sequence of characters, usually preceded by a period (.) which are used to indicate the type or format of file. File extensions help both users and software programs to identify the content and purpose of a file.

⚠ Warning

If you are working on your own desktop computer, you may not see the file extensions. This is because the file explorer is configured to hide them by default. To see the file extensions, you will need to change the settings in your file explorer. Use a search engine to find instructions for your operating system.

In addition to those listed above, other file extensions often encountered when working with data for text analysis include *.txt*, *.pdf*, *.docx*, *.xlsx*, *.json*, and *.html*. Common file extensions will often be associated with specific software programs on your computer, especially those which are directly associated with proprietary software such as *.docx* for Microsoft Word or *.xlsx* for Microsoft Excel. However, many file extensions are not directly associated with any specific software program and can be opened and edited with any text editor.

It is important to note that file extensions are helpful conventions, but they are not a guarantee of the file type or structure of the file content. Furthermore, corpus developers may create their own file extensions to signal the unique structure of their data. For example, the *.utt* file extension used in the Switchboard Dialogue Act Corpus (SWDA) or the *.cha* extension used for TalkBank resource transcripts signal project-specific structuring. In either case, it is recommended to open the file in a text editor to inspect the structure of the file content to confirm the file structure before processing the data contained therein.

File content

The internal structure of the content of corpus data files is an important aspect of any corpus both in terms of what data is included and how to approach accessing and processing the data. A corpus may include various types of linguistic (*e.g.* part of speech, syntactic structure, named

entities, *etc.*) or non-linguistic (*e.g.* source, dates, speaker information, *etc.*) attributes. These attributes are known as **metadata**, or data about data. As a general rule, files which include more metadata tend to be more internally structured. Internal file structure refers to the degree to which the content is easy to query and analyze by a computer. Let's review characteristics of the three main types of file structure types and associate common file extensions that files in each have.

Unstructured data is data which does not have a machine-readable internal structure. This is the case for plain text files (*.txt*), which are simply a sequence of characters. For example, in Example 2.2 we see a snippet of a plain text file from the the Manually Annotated Sub-Corpus of American English (MASC) (Ide et al. 2008):

Example 2.2. MASC plain text

```
>Hotel California

Fact: Sound is a vibration. Sound travels as a mechanical wave through a medium,
    ↵ and in space, there is no
medium. So when my shuttle malfunctioned and the airlocks didn't keep the air
    ↵ in, I heard nothing. After the
first whoosh of the air being sucked away, there was lightning, but no thunder.
    ↵ Eyes bulging in
panic, but no screams. Quiet and peaceful, right? Such a relief to never again
    ↵ hear my crewmate Jesse natter
about his girl back on Earth and that all-expenses-paid vacation-for-two she won
    ↵ last time he was on leave. I
swore, if I ever had to see a photo of him in a skimpy bathing suit again,
    ↵ giving the camera a cheesy thumbs-up
from a lounge chair on one of those white sandy beaches, I'd kiss a monkey.
    ↵ Metaphorically, of course.
```

Other examples of files which often contain unstructured data include *.pdf* and *.docx* files. While these file types may contain data which appears structured to the human eye, the structure is not designed to be machine-readable. As such the data would typically be read into R as a vector of **character strings**. It is possible to perform only the most rudimentary queries on this type of data, such as string matches. For anything more informative, it is necessary to further process this data.

On the other end of the spectrum, **structured data** is data which conforms to a tabular format in which elements in tables and relationships between tables are defined. This makes querying and analyzing easy and efficient. Relational databases (*e.g.* MySQL, PostgreSQL, etc.) are designed to store and query structured data. The data frame object in R is also a structured data format. In each case, the data is stored in a tabular format in which each row

represents a single observation and each column represents a single attribute whose values are of the same type.

In Example 2.3 we see an example of an R data frame object which overlaps with the data in the plain text file above in Example 2.2:

Example 2.3. MASC data frame

	title	date	modality	domain	ref_num	word	lemma	pos
	<chr>	<dbl>	<fct>	<chr>	<dbl>	<chr>	<chr>	<chr>
1	Hotel California	2008	Writing	General Fiction	0 >	>	NN	
2	Hotel California	2008	Writing	General Fiction	1	Hotel	hotel	NNP
3	Hotel California	2008	Writing	General Fiction	2	Cali...	cali...	NNP
4	Hotel California	2008	Writing	General Fiction	3	Fact	fact	NNP
5	Hotel California	2008	Writing	General Fiction	4 :	:	:	
6	Hotel California	2008	Writing	General Fiction	5	Sound	sound	NNP
7	Hotel California	2008	Writing	General Fiction	6	is	be	VBZ
8	Hotel California	2008	Writing	General Fiction	7	a	a	DT
9	Hotel California	2008	Writing	General Fiction	8	vibr...	vibr...	NN
10	Hotel California	2008	Writing	General Fiction	9 .	.	.	
11	Hotel California	2008	Writing	General Fiction	10	Sound	sound	NNP

Here we see that the data is stored in a tabular format with each row representing a single observation (`word`) and each column representing a single attribute. Internally, R applies a schema to ensure the values in each column are of the same type (*e.g.* `<chr>`, `<dbl>`, `<fct>`, *etc.*). This structured format is designed to be easy to query and analyze and as such is the primary format for data analysis in R.

 **Tip**

It is conventional to work with column names for datasets in R using the same conventions that are used for naming objects. It is a matter of taste which convention is used, but I have adopted snake case^a as my personal preference (*e.g.* `ref_num`). There are also alternatives^b. Regardless of the convention you choose, it is good practice to be consistent. It is also of note that the column names should be balanced for meaningfulness and brevity. This brevity is of practical concern but can be somewhat opaque. For questions into the meaning of the column and its values consult the resource's dataset documentation, consult Section 2.3.

^ahttps://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/names.html#snake_case

^b<https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/names.html>

Semi-structured data falls between unstructured and structured data. This covers a wide range of file structuring approaches. For example, a otherwise plain text file with part-of-speech tags appended to each word is minimally structured (Example 2.4).

Example 2.4. MASC plain text with part-of-speech tags

```
>/NN Hotel/NNP California/NNP Fact/NNP :/: Sound/NNP is/VBZ a/DT vibration/NN
↪  ./ Sound/NNP travels/VBZ as/IN a/DT mechanical/JJ wave/NN through/IN a/DT
↪  medium/NN ,/, and/CC in/IN space/NN ,/, there/EX is/VBZ no/DT medium/NN .
↪  So/RB when/WRB my/PRP$ shuttle/NN malfunctioned/JJ and/CC the/DT
↪  airlocks/NNS did/VBD n't/RB keep/VB the/DT air/NN in/IN ,/, I/PRP heard/VBD
↪  nothing/NN ./. After/IN the/DT
```

Towards the more structured end of semi-structured data, many file formats including *.xml* and *.json* contain highly structured, hierarchical data. For example, in Example 2.5 shows a snippet from a *.xml* file from the MASC corpus.

Example 2.5. MASC XML

```
<a xml:id="penn-N65571" label="tok" ref="penn-n0" as="anc">
↪
<fs>
  <f name="base" value="&gt;"/>
  <f name="msd" value="NN"/>
  <f name="string" value="&gt;"/>
</fs>
</a>
<node xml:id="penn-n1">
  <link targets="seg-r1"/>
</node>
<a xml:id="penn-N65599" label="tok" ref="penn-n1" as="anc">
  <fs>
    <f name="base" value="hotel"/>
    <f name="msd" value="NNP"/>
    <f name="string" value="Hotel"/>
  </fs>
</a>
```

The format of semi-structured data is often influenced by characteristics of the data or reflect an author's individual preferences. It is sometimes the case that data will be semi-structured in a less-standard format. For example, the SWDA corpus includes a *.utt* file extension for files which contain utterances annotated with dialogue act tags.

Example 2.6. SWDA .utt file

```
o          A.1 utt1: Okay. /
qw          A.1 utt2: {D So, }

qy^d          B.2 utt1: [ [ I guess, +
+          A.3 utt1: What kind of experience [ do you, + do you ] have, then with
↪  child care? /

+          B.4 utt1: I think, ] + {F uh, } I wonder ] if that worked. /
qy          A.5 utt1: Does it say something? /
```

Whether standard or not, semi-structured data is often designed to be machine-readable. As with unstructured data, the ultimate goal is to convert the data into a structured format and augment the data where necessary to prepare it for a particular research analysis.

File encoding

The last aspect to consider about corpus formats is **file encoding**. For a computer to display and process text characters, it must be encoded in a way that the computer can understand (*i.e.* 1's and 0's). Historically, character encoding schemes were developed to represent characters from specific character script sets (*e.g.* ASCII only includes characters from the English alphabet). However, as the need for a consistent and more inclusive way to encode characters from multiple languages and scripts became apparent, the Unicode standard, Unicode Transformation Format (UTF), was developed in the early 1990s. UTF encodings (UTF-8, UTF-16, and UTF-32) are now the most common way to encode text data and modern computers typically use them by default. Although other more script-specific encoding schemes can still be found in older data (*e.g.* ISO-8859, Windows-1252, Shift JIS).

When working with corpus data, it is important to know if the encoding scheme used for the data is compatible with your computing environment's default (most likely UTF). If it is not, you will need to convert the data to a compatible encoding scheme. Rest assured, there is support in R for converting between different encoding schemes if the need arises.

2.2 Information

Identifying an adequate corpus resource, in terms of content, licensing, and formatting, for the target research question is the first step in moving a quantitative text research project forward. The next step is to select the components or characteristics of this resource that are relevant

for the research and then move to organize the attributes of this data into a more informative format. This is the process of converting corpus data into a **dataset** – a tabular representation of particular attributes of the data as the basis for generating information. Once the data represented as dataset, it is often manipulated and transformed adjusting and augmenting the data such that it better aligns with the research question and the analytical approach.

2.2.1 Organization

Data alone is not informative. Only through explicit organization of the data in a way that makes relationships and meaning explicit does data become information. In this form, our data is called a dataset. This is a particularly salient hurdle in text analysis research. Many textual sources are unstructured or semi-structured, that is relationships that will be used in the analysis have yet to be purposefully drawn and organized from the data.

Tidy Data

The selection of the attributes from a corpus and the juxtaposition of these attributes in a relational format, or dataset, that converts data into information is known as **data curation**. The process of data curation minimally involves creating a base dataset, or *curated dataset*, which establishes the main informational associations according to philosophical approach outlined by Wickham (2014).

In this work, a **tidy dataset** refers both to the structural (physical) and informational (semantic) organization of the dataset. Physically, a tidy dataset is a tabular data structure, illustrated in Figure 2.3, where each *row* is an observation and each *column* is a variable that contains measures of a feature or attribute of each observation. Each cell where a given row-column intersect contains a *value* which is a particular attribute of a particular observation for the particular observation-feature pair also known as a *data point*.

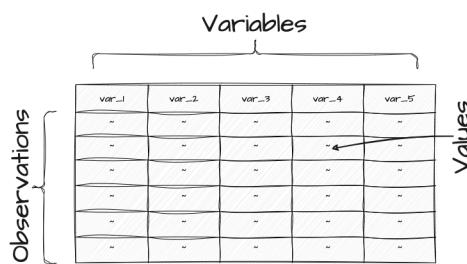


Figure 2.3: Visual summary of the tidy format.

In terms of semantics, columns and rows both contribute to the informational value of the dataset. Let's start with columns. In a tidy dataset, each column is a variable, an attribute

Table 2.1: MASC dataset variables.

title	modality	date	ref_num	word	pos	num_letters
Hotel California	Writing	2008	0	>	NN	1
Hotel California	Writing	2008	1	Hotel	NNP	5
Hotel California	Writing	2008	2	California	NNP	10
Hotel California	Writing	2008	3	Fact	NNP	4
Hotel California	Writing	2008	4	:	:	1
Hotel California	Writing	2008	5	Sound	NNP	5
Hotel California	Writing	2008	6	is	VBZ	2
Hotel California	Writing	2008	7	a	DT	1
Hotel California	Writing	2008	8	vibration	NN	9
Hotel California	Writing	2008	9	.	.	1

that can take on a number of values. Although variables vary in terms of values, they do not in type. A variable is of one and only one informational type. Statistically speaking, informational types are defined as **levels of measurement**, a classification system used to semantically distinguish between types of variables. There are four levels (or types) in this system: nominal, ordinal, interval, and ratio.

In practice, however, text analysis researchers often group these levels into three main informational types: categorical, ordinal, and numeric (S. T. Gries 2021). What do these informational types represent? **Categorical data** is for labeled data or classes that answer the question “what?” **Ordinal data** is categorical data with rank order that answers the question “what order?” **Numeric data** is ordinal data with equal intervals between values that answers the question “how much or how many?”

Let’s look at an example of a tidy dataset. Using the criteria just described, let’s see if we can identify the informational values (categorical, ordinal, or numeric) of the variables that appear in a snippet from the MASC corpus in dataset form in Table 2.1.

We have seven variables listed as headers for each of the columns. We could go one-by-one left-to-right but let’s take another tack. Instead, let’s identify all those variables that cannot be numeric –these are all the non-numeral variables: `title`, `modality`, `word`, and `pos`. The question to ask of these variables is whether they represent an order or rank. Since titles, modalities, words, and parts-of-speech are not ordered values, they are all categorical.

Now in relation to `date`, `ref_num`, and `num_letters`. All three are numerals, so they could be numeric. But they could also be numeral representations of ordinal data. Before we can move forward, we need to make sure we understand what each variable means and how it is measured, or **operationalized**. The variable name and the values can be helpful in this respect. `date` is what it sounds like, a date, and is operationalized as a year in the Gregorian calendar. And `num_letters` seems quite descriptive as well, number of letters, appearing as a

letter count. But in some cases it may be opaque as to what is being measured by the variable name alone, for example `ref_num`, and one will have to refer to the dataset documentation. In this case `ref_num` is a reference number operationalized as a unique identifier for each word per document in the corpus.

With this in mind, let's return to the question of whether `date`, `ref_num`, and `num_letters` are numeric or ordinal. Starting with the trickiest one, `date`, we can ask the question to identify numeric data: "how much or how many?". In the case of `date`, the answer is neither. A date is a point in time, not a quantity. So `date` is not numeric. But it does provide information about order. Hence, `date` is ordinal. `ref_num` is also ordinal because the question "what order?" can be asked of it. Finally, `num_letters` is numeric because it answers the question "how many?".

Let's turn to the second semantic value of a tidy dataset. In a tidy dataset, each row is an observation. But an observation of what? This depends on what the unit of observation is. That sounds circular, but it's not. The **unit of observation** is simply the primary entity that is being observed or measured (Sedgwick 2015). Even without context, it can often be identified in a dataset by looking at the level of specificity of the variable values and asking what each variable describes. When one variable appears to be the most individualized and other variables appear to describe that variable, then the most individualized variable is likely the unit of observation of the dataset, *i.e.* the meaning of each observation.

Applying these strategies to the Table in 2.1, we can see that each observation at its core is a word. We see that the values of each observation are the attributes of each word. `word` is the most individualized variable and the `pos` (part-of-speech), `num_letters`, and `ref_num` all describe the word.

The other variables `title`, `modality`, and `date` are not direct attributes of the word. Instead, they are attributes of the document in which the word appears. Together, however, they all provide information about the word.

💡 Consider this

Data can be organized in many ways. It is important to make clear that data in tabular format in itself does not constitute a dataset, in the tidy sense we will be using. Can you think of examples of tabular information that would not be in a tidy format? What would be the implications of this for data analysis?

As we round out this section on data organization, it is important to stress that the purpose of curation is to represent the corpus data in an informative, tidy format. A curated dataset serves as a reference point making relationships explicit, enabling more efficient querying, and paving the way for further processing before analysis. In the subsequent section, we will highlight common approaches to modifying the curated dataset, either row-wise or column-wise, to make it more amenable to the particular aims of a given analysis.

2.2.2 Transformation

At this point have introduced the first step creating a dataset ready for analysis, data curation. However, a curated dataset is rarely the final organizational step before proceeding to statistical analysis. Many times, if not always, the curated dataset requires **data transformation** to derive or generate new data for the dataset. This process may incur row-wise (observation) or column-wise (variable) level changes, as illustrated in Figure 2.4.

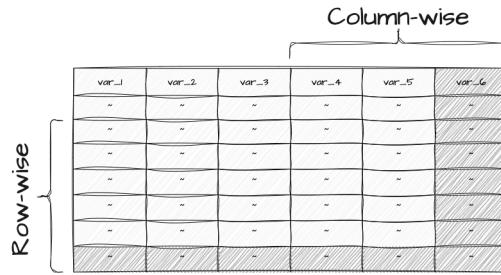


Figure 2.4: Visualization of row-wise and column-wise transformation operations on a dataset.

The results build on and manipulate the curated dataset to produce a *derived dataset*. While there is typically one curated dataset that serves as the base organizational dataset, there may be multiple derived datasets, each aligning with the informational needs of specific analyses in the research project.

In what follows, we will discuss the most common types of data transformation: text normalization, variable recoding, text tokenization, variable generation, and observation/ variable merging. Note, however, that the order in which these transformations are applied in a given research project is not fixed and will vary depending on the dataset and the research question(s) to be addressed.

Text normalization

The process of text normalization aims to prepare and standardize text. It is often a preliminary step in data transformation processes which include variables with text. The aim is to convert the text into a uniform format to reduce unwanted variation and noise.

Let's take a toy dataset, in Table 2.2, as an example starting point. In this dataset, we have two variables, `text_id` and `text`. It only has one observation.

Table 2.2: A toy dataset with two variables, `text_id` and `text`.

text_id	text
1	It's a beautiful day in the US, and our group decided to visit the famous Grand Canyon. As we reached the destination, Jane said, "I can't believe we're finally here!" The breathtaking view left us speechless; indeed, it was a sight to behold. During our trip, we encountered tourists from different countries, sharing stories and laughter. For all of us, this experience will be cherished forever.

The types of transformations we apply will depend on the specific needs of the project, but can include those found in Table 2.3.

Table 2.3: Common text normalization tasks

Task name	Relevant example	Typical purpose
Lowercasing	"Text" to "text"	Minimizing case sensitivity in subsequent analysis
Removal of Punctuation and Special Characters	"Hello, World!" to "Hello World"	Removing non-alphanumeric characters that may not carry semantic value
Adjustment of Forms	"colour" to "color", "it's" to "it is", "1" to "one"	Standardizing variations in spelling, contractions, and numeric forms to a common format

These transformations are column-wise operations, meaning they preserve the number of rows in the dataset. They also preserve the number of columns, but *do* change the values of the variables. These tasks should be applied with an understanding of how the changes will impact the analysis. For example, lowercasing can be useful for reducing differences between words that are otherwise identical, yet differ in case due to word position in a sentence ("The" versus "the"). However, lowercasing can also be problematic if the case of the word carries semantic value, such as in the case of "US" (United States) and "us" (first person plural pronoun).

Let's be conservative and only apply lowercasing to our toy dataset as seen in Table 2.4.

Table 2.4: A toy dataset with two variables, `text_id` and `text`, where the text has been lower-cased.

text_id	text
1	it's a beautiful day in the us, and our group decided to visit the famous grand canyon. as we reached the destination, jane said, "i can't believe we're finally here!" the breathtaking view left us speechless; indeed, it was a sight to behold. during our trip, we encountered tourists from different countries, sharing stories and laughter. for all of us, this experience will be cherished forever.

When text normalization steps are motivated and applied with foresight they serve to enhance the quality of the data and improves the reliability of subsequent transformation steps.

Variable recoding

Recoding is the process of transforming the values of one or more variables into new values which are more amenable to analysis. The aim is to simplify complex variables, making it easier to identify patterns and trends relevant for the research question. This is a column-wise operation which can be applied to categorical or numeric variables.

Let's return to the MASC dataset and demonstrate recoding of categorical and numeric variables. In Table 2.1 the `pos` variable whose values represent the part-of-speech (POS) of each token in the text. The measure is a POS tag from the Penn Treebank tagset (Marcus, Santorini, and Marcinkiewicz 1993). This tagset makes twelve major and 45 minor grammatical class distinctions. In an analysis that aims to explore only major class distinctions, it would be useful to recode the `pos` variable into major classes only (*i.e.* noun, pronoun, adjective, verb, adverb, *etc.*) to facilitate queries, summaries, and visualizations.

Table 2.5: A toy dataset with three variables, `text_id`, `pos`, `major_pos`, where the `pos` variable has been recoded into major grammatical classes `major_pos`.

title	modality	date	ref_num	word	pos	major_pos	num_letters
Hotel California	Writing	2008	0	>	NN	noun	1
Hotel California	Writing	2008	1	Hotel	NNP	noun	5
Hotel California	Writing	2008	2	California	NNP	noun	10
Hotel California	Writing	2008	3	Fact	NNP	noun	4
Hotel California	Writing	2008	4	:	:	punctuation	1
Hotel California	Writing	2008	5	Sound	NNP	noun	5
Hotel California	Writing	2008	6	is	VBZ	verb	2
Hotel California	Writing	2008	7	a	DT	determiner	1
Hotel California	Writing	2008	8	vibration	NN	noun	9

title	modality	date	ref_num	word	pos	major_pos	num_letters
Hotel California	Writing	2008	9	.	.	punctuation	1

In Table 2.5, the `pos` variable has been recoded into major grammatical classes. The `major_pos` variable is a categorical variable with 12 levels, one for each major grammatical class in the Penn Treebank tagset. While the demonstration here demonstrates the simplification of a categorical variable, recoding can also be used to transliterate categorical variables. Continuing with the theme of POS tags, the `pos` variable could be recoded into a different tagset, such as the Universal Dependencies tagset (Nivre et al. 2016).

Now, let's look at recoding the numeric variable `num_letters`. This variable represents the number of letters in each token. In the MASC dataset, the `num_letters` variable is a numeric variable with a range of values from 1 to 21. In some analyses, it may be useful to recode this variable into discrete categories, or bins, such as short, medium, and long words.

Table 2.6: The MASC dataset with the `num_letters` variable recoded into three categories: short, medium, and long words in `word_length`.

title	modality	date	ref_num	word	pos	major_pos	num_letters	word_length
Hotel California	Writing	2008	0	>	NN	noun	1	short
Hotel California	Writing	2008	1	Hotel	NNP	noun	5	medium
Hotel California	Writing	2008	2	California	NNP	noun	10	long
Hotel California	Writing	2008	3	Fact	NNP	noun	4	medium
Hotel California	Writing	2008	4	:	:	punctuation	1	short
Hotel California	Writing	2008	5	Sound	NNP	noun	5	medium
Hotel California	Writing	2008	6	is	VBZ	verb	2	short
Hotel California	Writing	2008	7	a	DT	determiner	1	short
Hotel California	Writing	2008	8	vibration	NN	noun	9	long
Hotel California	Writing	2008	9	.	.	punctuation	1	short

In Table 2.6 the variable `word_length` appears with the values `short`, `medium`, and `long`. This is now a categorical variable of type ordinal. Of note, is that the operational definition of used to create these word length bins should be made explicit in the documentation of the dataset.

In sum, recoding is a useful data transformation technique that can be used to simplify complex variables, making it easier to identify patterns and trends relevant for the research question.

Text tokenization

A text-oriented transformation step is **text tokenization**. This process involves adapting the text such that it reflects the target linguistic unit that will be used in the analysis. This is a row-wise operation expanding the number of rows, if the linguistic unit is smaller than the original variable, or reducing the number of rows, if the linguistic unit is larger than the original variable. At its core, tokenization is the process which enables the quantitative analysis of text.

Text variables can be tokenized at any linguistic level. To illustrate, consider our toy dataset from Table 2.4. We can tokenize the text at the sentence level, in Table 2.7, by splitting the text at the period followed by a space. This results in a dataset with four observations, one for each sentence in the original text.

Table 2.7: A toy dataset with two variables, `text_id` and `sentence`, where the text has been tokenized at the sentence level.

text_id	sentence
1	it's a beautiful day in the us, and our group decided to visit the famous grand canyon
1	as we reached the destination, jane said, "i can't believe we're finally here!" the breathtaking view left us speechless; indeed, it was a sight to behold
1	during our trip, we encountered tourists from different countries, sharing stories and laughter
1	for all of us, this experience will be cherished forever.

It is important to make explicit what the operationalization of our linguistic unit is as common terms such as sentence, word, *etc.* can be defined in different ways. For example, the sentence tokenization above is based on the assumption that sentences are separated by a period followed by a space. This is a suitable definition for this text, but likely will not be for other English text or for other languages/ writing scripts. For words, a very simple operationalization is to use whitespace separation (*e.g.* "I cannot believe it." – ["I", "cannot", "believe", "it."]). However, this approach does not handle punctuation marks (*e.g.* ["it."]) or contractions (*e.g.* ["can't"]). A more sophisticated operationalization will be necessary for these, and possibly other, cases.

Another important token unit is the n -gram. Words or characters can be grouped into contiguous sequences with a moving window of a certain size n . Single unit windows are referred to as unigrams, two units as bigrams, three units as trigrams, and so on. Let's tokenize our toy dataset at the bigram level for words using a simple whitespace separation for words, as seen in Table 2.8.

Table 2.8: A toy dataset with two variables, `text_id` and `bigram`, where the text has been tokenized at the bigram word level.

text_id	word
1	it's a
1	a beautiful
1	beautiful day
1	day in
1	in the
1	the us
1	us and
1	and our
1	our group
1	group decided

In Table 2.8 we see that the first bigram is “it's a” –the first two words (based on whitespace separation) in the text. The second bigram is “a toy” –the second and third words in the text. This continues to the end of the text. N -gram tokenization can be useful to capture context that would otherwise be lost from tokenizing words or characters at the unigram level.

Up to this point our tokens have been surface forms. That is, they are the actual words or characters as they appear in the text. However, we may want to reduce the tokens to their base form, removing their inflectional forms. This is known as **lemmatization**. For example, the word “run” is the lemma of the words “running”, “runs”, and “ran”. Let's lemmatize the third sentence in our toy dataset. For comparison, `word` and `lemma` are shown side-by-side in Table 2.9.

Table 2.9: A toy dataset with two variables, `text_id` and `word`, where the text has been tokenized at the unigram word level and lemmatized.

text_id	word	lemma
1	during	during
1	our	our
1	trip	trip
1	we	we
1	encountered	encounter

text_id	word	lemma
1	tourists	tourist
1	from	from
1	different	different
1	countries	country
1	sharing	share
1	stories	story
1	and	and
1	laughter	laughter

Case study

Inflectional family size is the number of inflectional forms for a given word and can be calculated from a corpus by counting the number of surface forms for each lemma in the corpus (Kostić, Marković, and Baucal 2003). Baayen, Feldman, and Schreuder (2006) found that words with larger inflectional family size are associated with faster word recognition times in lexical processing tasks.

Together tokenization and lemmatization are powerful tools for transforming text. If our dataset contains more robust linguistic annotation or that annotation can be generated (see Section 2.2.2), this information can also be leveraged to tokenize language into a format that is easier to explore and quantify in an analysis.

Variable generation

The process of variable generation aims to augment existing variables or create new ones, and as such it is a column-wise operation. Generation can include applying calculations or extracting relevant information from existing variables or enhancing text variables with linguistic annotation. Simplifying a bit, generation helps make implicit attributes explicit. The results of this process enables direct access during analysis to features that were otherwise hidden or difficult to access.

Let's highlight a some common calculation and extraction examples that generate variables. First, let's look at the calculation of measures. In text analysis, measures are often used to describe the properties of a document or linguistic unit. For example, the number of words in a corpus document, the lengths of sentences, the number of clauses in a sentence, *etc.*. In turn, these measures can be used to calculate other measures, such as lexical diversity or syntactic complexity measures.

In terms of extraction, the goal is to distill relevant information from existing variables. For example, extracting the year from a date variable, or extracting the first name from a full name variable. In text analysis, extraction is often used to extract information from text

Table 2.10: A MASC sample document in dataset tokenized into sentences.

text_id	sentence_id	sentence
1	1	>Hotel California Fact: Sound is a vibration.
1	2	Sound travels as a mechanical wave through a medium, and in space, there is no me
1	3	So when my shuttle malfunctioned and the airlocks didn't keep the air in, I heard no
1	4	After the first whoosh of the air being sucked away, there was lightning, but no thun
1	5	Eyes bulging in panic, but no screams.

variables. Say we have a dataset with a variable containing conversation utterances. We may want to extract some characteristic from those utterances and capture their occurrence in a new variable.

But what if we want to extract linguistic information from a text variable that is not explicitly present in the text? This is where linguistic annotation comes in. Linguistic annotation is the process of enriching text with linguistic information, such as morphological features, part-of-speech tags, syntactic structure, *etc..* This can be done manually by linguist coders and/ or done using natural language processing (NLP) tools, many of which are available in R (see Chapter 7).

To illustrate the process of generating linguistic annotation with existing tools, I will use the plain text version of the MASC. In Table 2.10, the text has been organized into a dataset and tokenized into sentences. The `text_id` variable is a unique identifier for each document, and the `sentence_id` variable is a unique identifier for each sentence.

Applying a pre-trained model from the Universal Dependencies (UD)²³²⁴ project, we can generate linguistic annotation for each token in the MASC.

Table 2.11: Automatic linguistic annotation for grammatical category and syntactic structure for an example English sentence from the MASC.

doc_id	sentence	token	itbken	xpos	features	syntactic_relation
1	After	4	1	IN	NA	mark
1	the	4	2	DT	Definite=Def PronType=Art	det
1	first	4	3	JJ	Degree=Pos NumType=Ord	amod
1	whoosh	4	4	NN	Number=Sing	nsubj:pass
1	of	4	5	IN	NA	case
1	the	4	6	DT	Definite=Def PronType=Art	det
1	air	4	7	NN	Number=Sing	nmod

²³<https://universaldependencies.org/>

²⁴The Universal Dependency project is an effort to develop cross-linguistically consistent treebank annotation for many languages. The project has developed a set of annotation guidelines and a set of tools for generating linguistic annotation. The project has also developed a set of pre-trained models for many languages.

doc_id	sentence_id	token_id	token	xpos	features	syntactic_relation
1	4	8	being	VBG	VerbForm=Ger	aux:pass
1	4	9	sucked	VBN	Tense=Past VerbForm=Part Voice=Pass	advcl
1	4	10	away	RB	NA	advmod
1	4	11	,	,	NA	punct
1	4	12	there	EX	NA	expl
1	4	13	was	VBD	Mood=Ind Number=Sing Person=3 Tense=Past VerbForm=Fin	Part
1	4	14	lightning	NN	Number=Sing	nsubj
1	4	15	,	,	NA	punct
1	4	16	but	CC	NA	cc
1	4	17	no	DT	NA	det
1	4	18	thunder	NN	Number=Sing	conj
1	4	19	.	.	NA	punct

The annotated dataset now includes the key variables `xpos` (Penn treebank tags), `features` (morphological features), and `syntactic_relation`. The results of this process can then be further transformed as need be to fit the needs of the analysis.

A word of caution: automated linguistic annotation offers rapid access to abundant and highly dependable linguistic data for numerous languages. However, linguistic annotation tools are not infallible. They are tools developed by training computational algorithms to identify patterns in previously annotated and verified datasets, resulting in a language model. This model is then employed to predict linguistic annotations for new language data (as seen in Table 2.11). The accuracy of the linguistic annotation heavily relies on the congruence between the language sampling framework of the trained data and the language data set to be automatically annotated.

Observation/ variable merging

The processing of merging datasets is a transformation step which can be row-wise or column-wise. Row-wise merging is the process of combining datasets by appending observations from one dataset to another. Column-wise merging is the process of combining datasets by appending variables from one dataset to another. In either case, merging provides a way to enrich a dataset by incorporating additional information.

To merge in row-wise manner the datasets involved in the process must have the same variables and variable types. This process is often referred to as **concatenating datasets**. It can be thought of as stacking datasets on top of each other to create a larger dataset. Remember, having the same variables and variable types is not the same as having the same values.

Take, for example, a case when a corpus resource contains data for two populations. In the course of curating and transforming the datasets it may make more sense to work with

the datasets separately. However, when it comes time to analyze the data, it may be more convenient to work with the datasets as a single dataset. In this case, the datasets can be concatenated to create a single dataset.

To illustrate, consider the toy datasets in Table 2.12 and Table 2.13.

Table 2.12: Toy dataset of written text data.

participant_id	text_id	modality	text
P1	T1	Written	Technology has revolutionized our lives in many ways. It has made communication easier, faster, and more efficient.
P3	T3	Written	Climate change is a pressing issue that affects everyone on Earth. We must take immediate action to reduce our carbon footprint.
P5	T5	Written	Education is the key to personal and societal growth. Investing in quality education will lead to a brighter future for all.

Table 2.13: Toy dataset of spoken text data.

participant_id	text_id	modality	text
P2	T2	Spoken	Hello, my name is X. I am a software engineer working at XYZ company.
P4	T4	Spoken	Hi, I'm X, and I work as a project manager. My main responsibility is to ensure that projects are completed on time and within budget.
P6	T6	Spoken	Hi, my name is X, and I'm a teacher. I teach English at a local high school.

These datasets, in Table 2.12 and Table 2.13, contain the same variables and variable types, but different observations –one in which the sample contains written language and the other spoken. Conveniently, they can be concatenated to create a single dataset that contains all of the observations, as seen in Table 2.14.

Table 2.14: Toy dataset of written and spoken text data concatenated.

participant_id	text_id	modality	text
P1	T1	Written	Technology has revolutionized our lives in many ways. It has made communication easier, faster, and more efficient.
P3	T3	Written	Climate change is a pressing issue that affects everyone on Earth. We must take immediate action to reduce our carbon footprint.
P5	T5	Written	Education is the key to personal and societal growth. Investing in quality education will lead to a brighter future for all.
P2	T2	Spoken	Hello, my name is X. I am a software engineer working at XYZ company.
P4	T4	Spoken	Hi, I'm X, and I work as a project manager. My main responsibility is to ensure that projects are completed on time and within budget.
P6	T6	Spoken	Hi, my name is X, and I'm a teacher. I teach English at a local high school.

Merging datasets can be performed in a column-wise manner as well. In this process, the datasets need not have the exact same variables and variable types, rather it is required that the datasets share a common variable of the same informational type that can be used to index the datasets. This process is often referred to as **joining datasets**.

Corpus resources often include metadata in stand-off annotation format. That is, the metadata is not embedded in the corpus files, but rather is stored in a separate file. The metadata and corpus files will share a common variable which is used to join the metadata with the corpus files.

To exemplify, here's another toy dataset that shares the `participant_id` index with the previous dataset in Table 2.14 and includes the variables `native_speaker_eng`, `age`, and `gender`:

Table 2.15: Toy dataset of participant data with a shared variable `participant_id` to index the datasets.

participant_id	native_speaker_eng	age	gender
P1	Yes	28	M
P2	No	35	M
P3	Yes	42	F
P4	No	26	F

participant_id	native_speaker_eng	age	gender
P5	Yes	31	M
P6	No	39	F

This dataset provides additional information about each participant, such as their English native speaker status, age, and gender.

Since the two datasets share the `participant_id` variable, we can merge them to create a new dataset that combines the information from both datasets, as we see in Table 2.16.

Table 2.16: Joining variables from two datasets based on a shared index variable.

participant_id	text_id	modality	text	native_speaker_eng	age	gender
P1	T1	Written	Technology has revolutionized our lives in many ways. It has made communication easier, faster, and more efficient.	Yes	28	M
P3	T3	Written	Climate change is a pressing issue that affects everyone on Earth. We must take immediate action to reduce our carbon footprint.	Yes	42	F
P5	T5	Written	Education is the key to personal and societal growth. Investing in quality education will lead to a brighter future for all.	Yes	31	M
P2	T2	Spoken	Hello, my name is X. I am a software engineer working at XYZ company.	No	35	M
P4	T4	Spoken	Hi, I'm X, and I work as a project manager. My main responsibility is to ensure that projects are completed on time and within budget.	No	26	F
P6	T6	Spoken	Hi, my name is X, and I'm a teacher. I teach English at a local high school.	No	39	F

Joining datasets is a powerful tool for enriching a dataset with additional column-wise information. It is important to note that merging datasets can also remove information in a row-wise manner. For example, when merging two datasets with a shared variable, it is possible to remove observations that do not have a match one of the two datasets. This process effectively filters out observations not shared between the two datasets. On the other hand, an anti-join explicitly removes observations that are shared between the two datasets.

Dive deeper

In some analyses, it may be useful to remove words with little semantic value, such as articles, prepositions, and conjunctions or words that are very common in the language. These are known as stopwords. There are various predefined lists of stopwords for different languages available on the web and through R in the `stopwords` package (Benoit, Muhr, and Watanabe 2021). Anti-joining a stopword list with a dataset of word tokens is often used to remove stopwords from the dataset.

However, it is important to note the criteria used to determine which words are considered stopwords in a particular resource may not fit a researcher's needs or the characteristics of the data. Learn more about approaches to identifying stopwords in Kaur and Buttar (2018).

In sum, the transformation steps described here collectively aim to produce higher quality datasets that are relevant in content and structure to submit to analysis. The process may include one or more of the previous transformations but is rarely linear and is most often iterative. It is typical to do some normalization then generation, then recoding, and then return to normalizing, and so forth. This process is highly idiosyncratic given the characteristics of the curated dataset and the ultimate goal for the derived dataset(s).

2.3 Documentation

As we have seen in this chapter, acquiring corpus data and converting that data into information involves a number of conscious decisions and implementation steps. As a favor to ourselves, as researchers, and to the research community, it is crucial to document these decisions and steps. This makes it both possible to retrace our own steps and also provides a guide for future researchers that want to reproduce and/ or build on your research. A programmatic approach to quantitative research helps ensure that the implementation steps are documented and reproducible but it is also vital that the decisions that are made are documented as well. This includes data origin information for the acquired corpus data and data dictionaries for the curated and derived datasets.

2.3.1 Data origin

Data acquired from corpus resources should be accompanied by information about the **data origin**. Table 2.17 provides a list of the types of information that should be included in the data origin information.

Table 2.17: Data origin information.

Information	Description
Resource name	Name of the corpus resource.
Data source	URL, DOI, <i>etc.</i>
Data sampling frame	Language, language variety, modality, genre, <i>etc.</i>
Data collection date(s)	The date or date range of the data collection.
Data format	Plain text, XML, HTML, <i>etc.</i>
Data schema	Relationships between data elements: files, folders, <i>etc.</i>
License	CC BY, CC BY-NC, <i>etc.</i>
Attribution	Citation information for the data source.

For many corpus resources, the corpus documentation will include all or most of this information as part of the resource download or documented online. If this information is not present in the corpus resource or you compile your own, it is important to document this information yourself. This information can be documented in file, such as a plain text file or spreadsheet, that is included with the corpus resource.

2.3.2 Data dictionaries

The process of organizing the data into a dataset, curation, and modifications to the dataset in preparation for analysis, transformation, each include a number of project-specific decisions. These decisions should be documented.

On the one hand each dataset that is created should have a **data dictionary** file. A data dictionary is a document, usually in a spreadsheet format, that describes the variables in a dataset. The key information that should be included in a data dictionary is provided in Table 2.18.

Table 2.18: Data dictionary information.

Information	Description
Variable name	The name of the variable as it appears in the dataset, <i>e.g.</i> <code>participant_id</code> , <code>modality</code> , <i>etc.</i>
Readable variable name	A human-readable name for the variable, <i>e.g.</i> ‘Participant ID’, ‘Language modality’, <i>etc.</i>
Variable type	The type of information that the variable contains, <i>e.g.</i> ‘categorical’, ‘ordinal’, <i>etc.</i>
Variable description	A prose description expanding on the readable name and can include measurement units, allowed values, <i>etc.</i>

Organizing this information in a tabular format, such as a spreadsheet, can make it easy for others to read and understand your data dictionary.

On the other hand, the data curation and transformation steps should be documented in the code that is used to create the dataset. This is one of the valuable features of a programmatic approach to quantitative research. The transparency of this documentation is enhanced by using **literate programming** strategies to intermingling prose descriptions and code the steps in the same, reproducible document.

By providing a comprehensive data dictionary and using a programmatic approach to data curation and transformation, you ensure that others can easily understand and work with your dataset, facilitating collaboration and reproducibility.

Summary

In this chapter we have focused on data and information –the first two components of DIKI Hierarchy. This process is visualized in Figure 2.5.

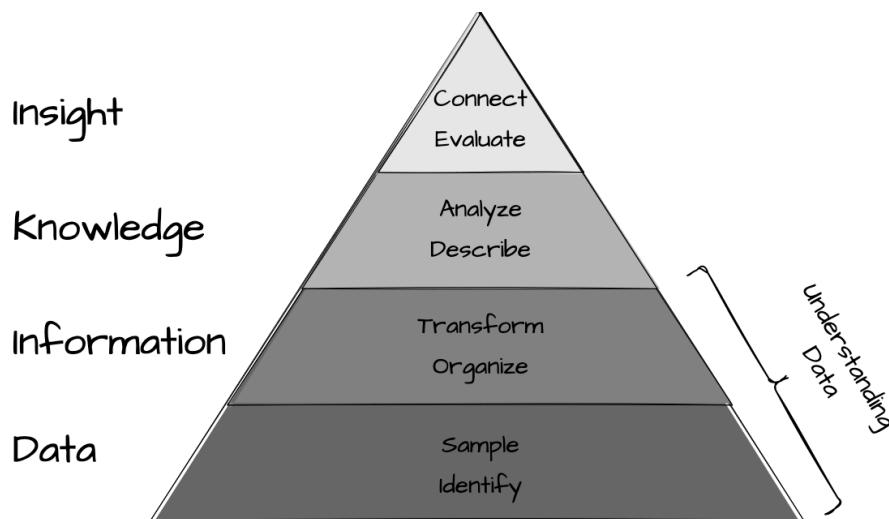


Figure 2.5: Understanding data: visual summary

First a distinction is made between populations and samples, the latter being a intentional and subjective selection of observations from the world which attempt to represent the population of interest. The result of this process is known as a corpus. Whether developing a corpus or selecting an existing a corpus it is important to vet the sampling frame for its applicability and viability as a resource for a given research project.

Once a viable corpus is identified, then that corpus is converted into a curated dataset which adopts the tidy dataset format where each column is a variable, each row is an observation, and

the intersection of columns and rows contain values. This curated dataset serves to establish the base informational relationships from which your research will stem.

The curated dataset will most likely require transformations which may include normalization, tokenization, recoding, generation, and/ or merging to enhance the usefulness of the information to analysis. A derived dataset or set of datasets will be the result from this process.

Finally, documentation should be implemented at the acquisition, curation, and transformation stages of the analysis project process. The combination of data origin, data dictionary, and literate programming files establishes documentation of the data and implementation steps to ensure transparent and reproducible research.

Activities

In the following activities you will learn how to read, inspect, and write data and datasets in R using reproducible strategies.

_recipe

What: Reading, inspecting, and writing data^a

How: Read Recipe 2 and participate in the Hypothes.is online social annotation.

Why: To use literate programming in Quarto to work with R coding strategies for reading, inspecting, and writing datasets.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-2.html>

Lab

What: Reading, inspecting, and writing data^a

How: Clone, fork, and complete the steps in Lab 2.

Why: To read datasets from packages and from plain-text files, inspect and report characteristics of datasets, and write datasets to plain-text files.

^a<https://github.com/qtalr/lab-2>

Questions

Conceptual questions

- What is the difference between a population and a sample?
- Why is it important to vet a corpus before using it in a research project?
- What is a curated dataset in the context of linguistic research?

- What is the difference between a variable, an observation, and a value?
- Why is it important to identify the informationasl types of variales in a dataset?
- What kinds of transformations may be performed on a curated dataset to enhance its usefulness for analysis?
- What is an transformed dataset and why is it important in linguistic research?
- Why is documentation important in the process of conducting linguistic analysis?
- How does a programmatic approach enhance documentation in linguistic research?
- How does documenting the corpus data and the curated and derived datasets contribute to transparent and reproducible research in linguistics?

❖ Technical questions

- Creating a sample corpus.
- Writing a corpus documentation.
- Converting a corpus to a derived dataset.
- Writing a data dictionary.
- Transforming a derived dataset.
- Merging datasets.
- Writing a dataset to disk.
- Consider (an example dataset) and its data dictionary, write a script to read the dataset, inspect it, and write it to disk.
- Consider a dataset and its data dictionary what appears to be the unit of analysis and the unit of observation?

3 Approaching analysis



Draft

Ready for review.

Statistical thinking will one day be as necessary for efficient citizenship as the ability to read and write.

— H.G. Wells



Outcomes

- Recall the fundamental concepts and principles of statistics in data analysis.
- Articulate the roles of diagnostic, analytic, and interpretive statistics in quantitative analysis.
- Compare the similarities and differences between analytic approaches to data analysis.

In this chapter I will build on the notions of data and information from the previous chapter. The aim of analysis is to derive knowledge from information, the next step in the DIKI Hierarchy. Where the creation of information from data involves human intervention and conscious decisions, as we have seen, deriving knowledge from information involves another level of intervention. The goal is to break down complex information into simpler components which are more readily interpretable. In what follows, we will cover the main steps in the process of analysis. The first is to inspect the data to ensure its quality and understand its characteristics. The second is to interrogate the data to uncover patterns and relationships and interpret the findings. To conclude this chapter I will outline methods to and the importance of communicating the analysis results and procedure in a transparent and reproducible manner.



Lessons

What: Data visualization^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: To explore data visually in text and in graphics.

Table 3.1: Data dictionary for the BELC dataset.

variable	name	description	variable_t
part_id	Participant ID	Unique identifier for each participant	categorical
sex	Participant's sex	Sex of the participant	categorical
group	Time group	Longitudinal group to which the participant belongs	ordinal
month_age	Participant's age in months	Age of the participant in months	numeric
utt_id	Utterance ID	Unique identifier for each utterance	numeric
word_id	Word ID	Unique identifier for each word within an utterance	numeric
word	Word	The word spoken by the participant	categorical
lemma	Word lemma	Base form of the word	categorical
pos	Part of speech	Grammatical category of the word	categorical

^a<https://github.com/qtalr/lessons>

3.1 Diagnose

The purpose of diagnostic measures is to inspect your data to ensure its quality and understand its characteristics. There are two primary types of diagnostic measures: verification and description. Verification methods are applied to catch missing or erroneous data while descriptive methods are used to gain a better understanding of the data. Although treated in two separate sections, in practice these methods are complementary and are often addressed in tandem.

To ground this discussion I will introduce a new dataset. This dataset is drawn from the Barcelona English Language Corpus (BELC) (Muñoz 2006), which is found in the TalkBank repository. I've selected the "Written composition" task from this corpus which contains 80 writing samples from 36 second language learners of English at different ages. Participants were given the task of writing for 15 minutes on the topic of "Me: my past, present and future". Data was collected for participants from one to three times over the course of seven years (at 10, 12, 16, and 17 years of age).

In Table 3.1 we see the data dictionary for the BELC dataset which reflects structural and transformational steps I've done so we start with a tidy dataset with `word` as the unit of observation.

The data dictionary provides a easily accessible overview of the dataset. This includes a human-readable mapping from variable names to variable descriptions. Further, it provides information about the type of variable (e.g., categorical, ordinal, numeric). As we will see the informational type of variables is key to diagnostic measures, as well as all other components of analysis.

Table 3.2: Summary output for missing values in the BELC dataset.

variable	type	n_missing	complete_rate
part_id	character	0	1.000
sex	character	0	1.000
group	character	0	1.000
word	character	0	1.000
lemma	character	79	0.985
pos	character	23	0.996
month_age	numeric	0	1.000
utt_id	numeric	0	1.000
word_id	numeric	0	1.000

3.1.1 Verify

Although a dataset has undergone curation and transformation, it is still important to verify the data. This is a process of checking the data to ensure that it is accurate and complete. In the case that it is not, consideration should be given to how to address the issues.

The most basic and usually the first step is to check for **missing data** to ensure that all necessary data points are present. In Table 3.2, there are missing values for the `lemma` and `pos` variables in the BELC dataset.

There are two primary approaches to dealing with missing data: deletion and recoding. Since these missing values account for only 1.5% and 0.4% of the data respectively, we might be safe to remove these observations. Another approach is to recode the missing values by either applying a unique value for missing values (e.g., `NULL`) or by imputing values. Imputing values is usually done by replacing missing values with some middle-of-the-road value (e.g., mean, median, mode), but other, more nuanced approaches are possible.

Dive deeper

For more information on missing data, see the  in this book.

In either case, it is important to consider the implications of missing data for the analysis. For example, if the missing data is not at random or include a sizeable portion of the values of interest, then the analysis may be biased.

Value coding schemes, annotation errors, or other issues may result in **anomalies** in the data. These are values that are unusual, unexpected, or inconsistent with the rest of the data or effect the treatment of the data for the particular analysis to be performed.

For categorical variables, this may include values that are not expected or are not in the set of values that are expected. A summary of the values for a given variable can be used as a first step to identify anomalies. In Table 3.3, we see the minimum and maximum number

Table 3.3: Summary output for categorical variables in the BELC dataset.

variable	min_chars	max_chars	num_unique
part_id	3	3	36
sex	4	6	2
group	2	2	4
word	1	20	913
lemma	1	20	774
pos	1	9	38

of characters and the number of unique values for each categorical variable in the BELC dataset.

From our knowledge of the data, we can gauge whether these values are expected. For example, `sex` has two values; likely corresponding to some coding of ‘male’ and ‘female’. The variable `part_id` has 36 distinct values, which is expected since there are 36 participants and `group` has four, corresponding to the longitudinal time groups. It is also possible to gauge the expected values for `lemma` as we know that these are the base words and should be less than the number of words in the dataset.

Further verification of the categorical variables is need, of course. This may include aggregating the data to see the distribution of values and/ or checking the values against the documentation.

Let’s now consider numeric variables. Numeric variables, by their very nature, do not lend themselves to the same type of summary used for categorical variables (*i.e.* character lengths, number of unique values, or aggregation) to detect anomalies. For numeric variables there are two types of anomalies that we will consider: outliers and errors in coding. **Outliers** are anomalies that are extreme values that are not representative of the great majority of the data points. To determine what is extreme, we need to consider the distribution of the data, that is, the range of values and the frequency of values. It is rarely the case that we can eyeball the distribution of the data based on raw values. Instead, a combination of summary statistics and visualizations are used to determine the distribution of the data. For this reason, the detection of outliers is often carried out as part of the descriptive assessment of the data, as we will see in Section 3.1.2.

On the other hand, coding anomalies are values that are not expected or are not in the set of values that are expected. These can sometimes be detected by visual inspection of the data. For example, in Table 3.4, we see the first 10 observations for each variable in the BELC dataset.

Leaving `month_age` aside, we see that the other two numeric variables `utt_id` and `word_id` index utterances and words respectively. However, in contrast to `part_id` which is a categorical variable as it serves as a unique identifier for each participant, these variables are numeric as they serve to not only index utterances and words but also to provide a measure of how many

Table 3.4: First 10 observations for variables in the BELC dataset.

part_id	sex	group	month_age	utt_id	word_id	word	lemma	pos
L01	female	T2	153	0	0	I	I	pro:sub
L01	female	T2	153	0	1	was	be	cop
L01	female	T2	153	0	2	born	born	adj
L01	female	T2	153	0	3	in	in	prep
L01	female	T2	153	0	4	Barcelona	Barcelona	n:prop
L01	female	T2	153	0	5	and	and	coord
L01	female	T2	153	0	6	I	I	pro:sub
L01	female	T2	153	0	7	live	live	v
L01	female	T2	153	0	8	in	in	prep
L01	female	T2	153	0	9	Barcelona	Barcelona	n:prop

Table 3.5: First 10 observations of the reconfigured BELC dataset.

essay_id	part_id	sex	group	tokens	types	ttr	prop_l2
E1	L01	female	T2	79	46	0.582	0.987
E2	L02	female	T1	18	18	1.000	0.667
E3	L02	female	T3	101	53	0.525	1.000
E4	L05	female	T1	20	17	0.850	0.900
E5	L05	female	T3	158	80	0.506	0.987
E6	L05	female	T4	184	94	0.511	0.995
E7	L07	male	T3	98	60	0.612	1.000
E8	L07	male	T4	134	84	0.627	0.978
E9	L10	female	T1	38	28	0.737	0.974
E10	L10	female	T3	118	74	0.627	1.000

utterances or words have been produced. Seen in this light, `0` for the first value of `utt_id` and `word_id` is unexpected. To adjust for this, we can add `1` to each value of these variables.

3.1.2 Describe

The goal of descriptive statistics is to summarize the data in order to understand and prepare the data for the analysis approach to be performed. This is accomplished through a combination of statistic measures and/ or tabular or graphic summaries. The choice of descriptive statistics is guided by the type of data, as well as the question(s) being asked of the data.

To that end, let's consider a reconfiguration of the BELC dataset, in Table 3.5, which will provide a more illustrative dataset.

In this new configuration, the unit of observation is now `essay_id`. Each of the following variable are attributes or measures of this variable. The new variables in this dataset are aggregates of the previous BELC dataset: `tokens` is the number of total words, `types` is the number of unique words, `ttr` is the ratio of unique words to total words. This is known as the Type-Token Ratio and it is a standard metric for measuring lexical diversity. Finally, the proportion of L2 words (English) to the total words (`tokens`) is provided in `prop_l2`.

In descriptive statistics, there are four basic questions that are asked of each of the variables in the dataset. Each correspond to a different type of descriptive measure.

1. Central Tendency: Where do the data points tend to be located?
2. Dispersion: How spread out are the data points?
3. Distribution: What is the overall shape of the data points?
4. Interdependence: How are these data points related to other data?

Central tendency

The central tendency is measure which aims to summarize the data points in a variable as the most representative, middle or most typical value. There are three common measures of central tendency: the mode, mean and median. Each differ in how they summarize the data points.

The **mode** is the value, or values, that appears most frequently in a set of values. If there are multiple values with the highest frequency, then the variable is said to be multimodal. The most versatile of the central tendency measures as it can be applied to all levels of measurement, although the mode is not often used for numeric variables as it is not as informative as other measures.

The more common measures for numeric variables are the mean and the median. The **mean** is a summary statistic calculated by summing all the values and dividing by the number of values. The **median** is calculated by sorting all the values in the variable and then selecting the middle value. Given that the mean and median are calculated differently, they will not always yield the same result. Differences that appear between the mean and median will be of interest to us later in this chapter.

Dispersion

The mean, median, and mode provide summary information where data points tend to be located. However, they do not provide us with any understanding as to how representative this value is. To provide this context, the spread of the values around the central tendency, or **dispersion**, is calculated.

For categorical variables, the spread is framed in terms of how balanced the values are across the levels. One way to do this is to calculate the (normalized) entropy. **Entropy** is a measure

Table 3.6: Central tendency and dispersion of the variables in the BELC dataset

(a) Categorical variables			(b) Numeric variables			
variable	top_counts		norm_entropy	mean	median	sd
essay_id	E1: 1, E10: 1, E11: 1, E12: 1		tokens	0.0767.62	56.50	44.20
part_id	L05: 3, L10: 3, L11: 3, L12: 3		types	0.9831.85	38.50	23.03
sex	fem: 48, mal: 32		ttr	0.9710.68	0.66	0.13
group	T1: 25, T3: 24, T2: 16, T4: 15		prop	0.29810.96	0.99	0.10
						iqr

of uncertainty. The more balanced the values are across the levels, the higher the entropy. The less balanced the values are across the levels, the lower the entropy. Normalized entropy scores range from 0 to 1, with 0 indicating that all the values are the same and 1 indicating that all the values are different.

The most common measure of dispersion for numeric variables is the **standard deviation**. The standard deviation is calculated by taking the square root of the variance. The **variance** is the average of the squared differences from the mean. So, more succinctly, the standard deviation is a measure of the spread of the values around the mean. Where the standard deviation is anchored to the mean, the **interquartile range** (IQR) is tied to the median. The median represents the sorted middle of the values, in other words the 50th percentile. The IQR is the difference between the 75th percentile and the 25th percentile. Again, just as the mean and the median, the standard deviation and the IQR are calculated in different ways, they are not always the same.

Let's now consider the relevant central tendency and dispersion of the variables in the BELC dataset in Table 3.6.

In Table 3.6a we see the measures for categorical variables. The `top_counts` variable gives us a short list of the most frequent levels of the variable. From `top_count` we can gather whether the variable has one mode or is multimodal. Both `essay_id` and `part_id` have the same most frequent value for the levels listed. On the other hand, `sex` and `group` have a single mode. We can also appreciate the dispersion of these variables based on the `norm_entropy` of each variable. `essay_id` is completely balanced across the levels, so it has a normalized entropy of 1. The other variables are not as balanced, but still quite balanced as the normalized entropy is close to 1.

In Table 3.6b the numeric variables have a column for the mean, median, standard deviation, and IQR for each. The variable `tokens` has a larger difference between the mean and median than the other variables and the standard deviation is relatively large suggesting that the values are more spread out around the mean. In the case of `ttr` the mean and median are quite close and the standard deviation is relatively small suggesting that the values are more tightly clustered around the mean.

Table 3.7: Standardized central tendency and dispersion of numeric variables

variable	mean	median	sd	iqr
tokens	0	-0.25	1	1.39
types	0	-0.15	1	1.37
ttr	0	-0.19	1	1.14
prop_l2	0	0.25	1	0.27

When interpreting these summary values, it is important to only directly compare column-wise. That is, focusing only on a single variable, not across variables. Each variable, as is, is measured on a different scale and only relative to itself can we make sense of the values.

However, we can transform the central tendency and dispersion scores for numeric variables to make them more comparable by standardizing the scale of the values. **Standardization** is a scale-based transformation that changes the scale of the values to a common scale, or *z-scores*. It involves two separate transformations: centering and scaling. **Centering** is a transformation that subtracts the mean or median from each value. The result is a mean and median of zero. **Scaling** is a transformation that divides each value by the standard deviation or IQR.

In Table 3.7, we see the same summary statistics as in Table 3.6b, but the values have been standardized for the mean and standard deviation. The mean is now zero and the standard deviation is one. This allows us to compare the median and IQR of the variables more directly.

One more caveat to keep in mind is that we need to be mindful of the nature of the data being standardized and what the standardized values mean. For example, the variables `tokens` and `types` were originally counts. But the standardized values are not interpretable as counts, they are now on a different scale –specifically a z-score scale. In the same way since the `ttr` and `prop_l2` variables were originally proportions, the standardized values are also not interpretable as proportions. One additional twist, however, is that the original scales for these pairs of variables were not the same: `tokens` and `types` were counts, but `ttr` and `prop_l2` were proportions. So, even though the standardized values are on the same scale, they are not directly comparable.

Beyond comparing central tendency and dispersion across variables, standardization is useful for analytic statistics to mitigate the influence of variables with large values. In some cases, the statistical method will require standardization of variables before analysis.

Distributions

Summary statistics of the central tendency and dispersion of a variable provide a sense of the most representative value and how spread out the data is around this value. However, to

Table 3.8: Frequency table for the variable `sex`.

sex	frequency	proportion
female	48	0.6
male	32	0.4

gain a more comprehensive understanding of the variable, it is key to consider the frequencies of all the data points. The **distribution** of a variable is the pattern or shape of the data that emerges when the frequencies of all data points are considered. This can reveal patterns that might not be immediately apparent from summary statistics alone. Understanding the frequency and distribution of data points is vital as it informs subsequent choices of statistical analysis and evaluative methods, ensuring they are appropriate for the specific characteristics of the data.

When assessing the distribution of categorical variables, we can use a frequency table or bar plot. A **frequency table** is a useful method to display the frequency and proportion of each level in a categorical variable in a clear and concise manner. In Table 3.8 we see the frequency table for the variable `sex`.

A **bar plot** is a type of plot where the x-axis is a categorical variable and the y-axis is the frequency of the values. The frequency is represented by the height of the bar. The variables can be ordered by frequency, alphabetically, or some other order. Figure 3.1 is a bar chart for the variables `sex`, `group`, and `part_id`, ordered alphabetically.

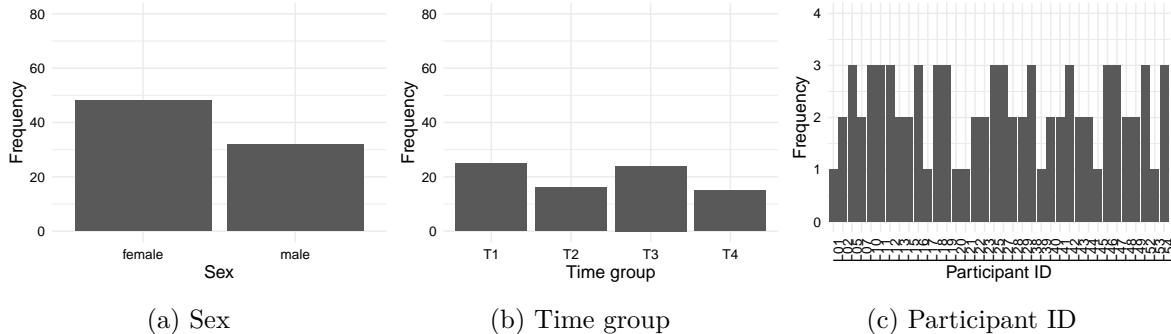


Figure 3.1: Bar plots for categorical variables `sex`, `group`, `part_id` in the BELC dataset.

So for a frequency table or barplot, we can see the frequency of each level of a categorical variable. This gives us some knowledge about the BELC dataset: there are more girls in the dataset, more essays appear in first and third time groups, and the number of essays written by each participant is scattered from one to three. If we were to see any clearly loopsided categories, this would be a sign of imbalance in the data and we would need to consider how this might impact our analysis.

💡 Consider this

The goal of descriptive statistics is to summarize the data in a way that is meaningful and interpretable. With this in mind, compare the frequency table in 3.8 and bar plot in 3.1a. Does one provide a more interpretable summary of the data? Why or why not? Are there any other ways you might communicate this distribution more effectively?

For numeric variables, understanding the distribution is more complex, and also more important. In essence, however, we are assessing two things: the appearance of outliers in relation to and the overall shape of the distribution.

Now, a frequency table, as in Table 3.8, does not summarize the distribution of a numeric variable in a concise, readily human-consumable format. Instead, the distribution of a numeric variable is best understood visually.

The most common visualizations of the distribution of a numeric variable are histograms and density plots. **Histograms** are a type of bar plot where the x-axis is a numeric variable and the y-axis is the frequency of the values falling within a determined range of values, or bins. The frequency of values within each bin is represented by the height of the bars. **Density plots** are a smoothed version of histograms. The y-axis of a density plot is the probability of the values. When frequent values appear closely together, the plot line is higher. When the frequency of values is lower or more spread out, the plot line is lower. An example of these plots is show in Figure 3.2 for the variable `tokens`.

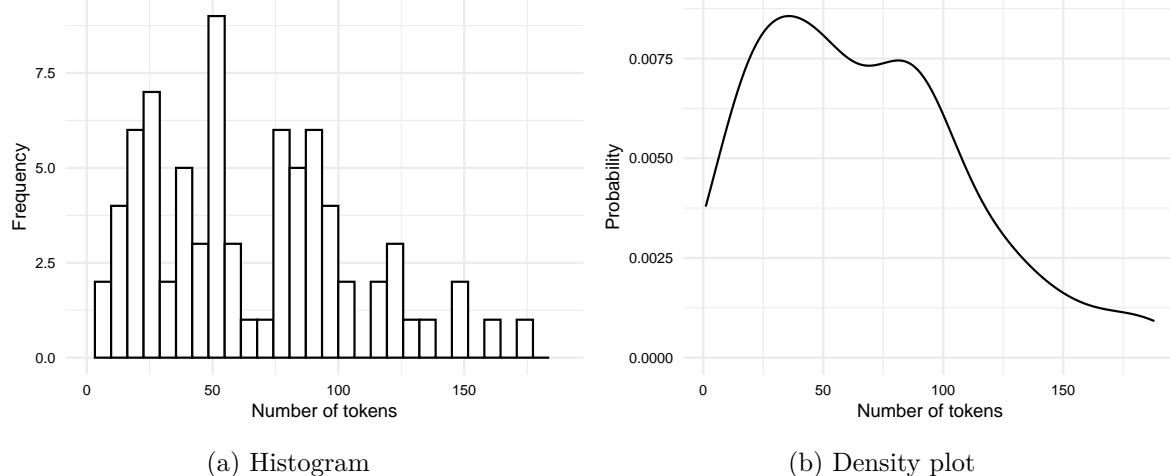


Figure 3.2: Distribution plots for the variable `tokens`.

Both the histogram in Figure 3.2a and the density plot in Figure 3.2b show the distribution of the variable `tokens` in slightly different ways which translate into trade-offs in terms of interpretability.

The histogram shows the frequency of the values in bins. The number of bins and/ or binwidth can be changed for more or less granularity. A rough grain histogram shows the general shape of the distribution, but it is difficult to see the details of the distribution. A fine grain histogram shows the details of the distribution, but it is difficult to see the general shape of the distribution. The density plot shows the general shape of the distribution, but it hides the details of the distribution. Given this trade-off, it is often useful explore outliers with histograms and the overall shape of the distribution with density plots.

In Figure 3.3 we see histograms for the variables `tokens`, `types`, and `ttr`.

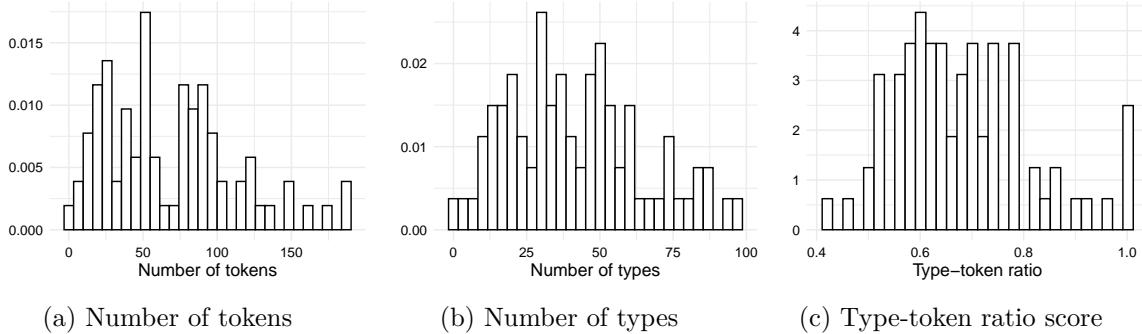


Figure 3.3: Histograms for numeric variables `tokens`, `types`, and `ttr`.

Focusing on the details captured in the histogram we are better able to detect potential outliers. Outliers can reflect valid values that are simply extreme or they can reflect something erroneous in the data. To distinguish between these two possibilities, it is important to know the context of the data. Take, for example, Figure 3.3c. We see that there is a bin near the value 1.0. Given that the type-token ratio is a ratio of the number of types to the number of tokens, it is unlikely that the type-token ratio would be exactly 1.0 as this would mean that every word in an essay is unique. Another, less dramatic, example is the bin to the far right of Figure 3.3a. In this case, the bin represents the number of tokens in an essay. An uptick in the number of essays with a large number of tokens is not surprising and would not typically be considered an outlier. On the other hand, consider the bin near the value 0 in the same plot. It is unlikely that a true essay would have 0, or near 0, words and therefore a closer look at the data is warranted.

It is important to recognize that outliers contribute undue influence to overall measures of central tendency and dispersion. To appreciate this, let's consider another helpful visualization called a **boxplot**. A boxplot is a visual representation which aims to represent the central tendency, dispersion, and distribution of a numeric variable in one plot.

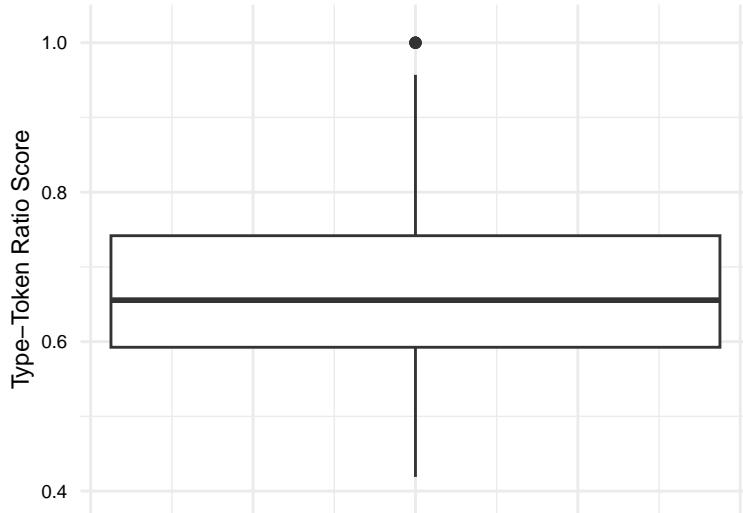


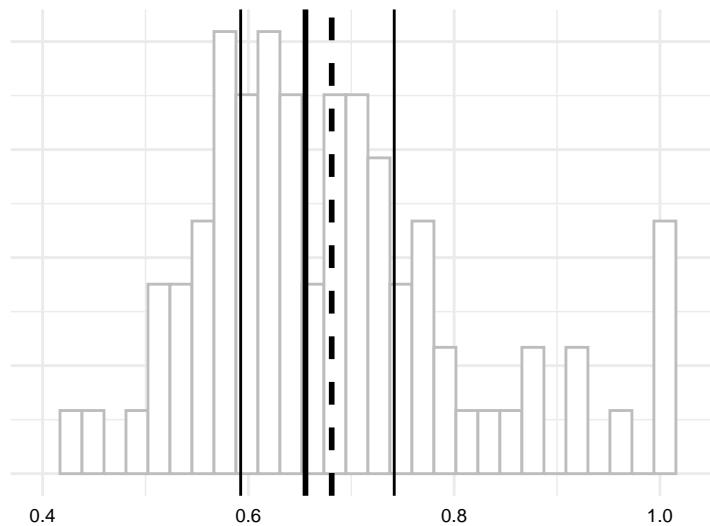
Figure 3.4: Boxplot for the variable `ttr`.

In Figure 3.4 we see a boxplot for `ttr` variable. The box in the middle of the plot represents the interquartile range (IQR) which is the range of values between the first quartile and the third quartile. The solid line in the middle of the box represents the median. The lines extending from the box are called ‘whiskers’ and provide the range of values which are within 1.5 times the IQR. Values outside of this range are plotted as individual points.

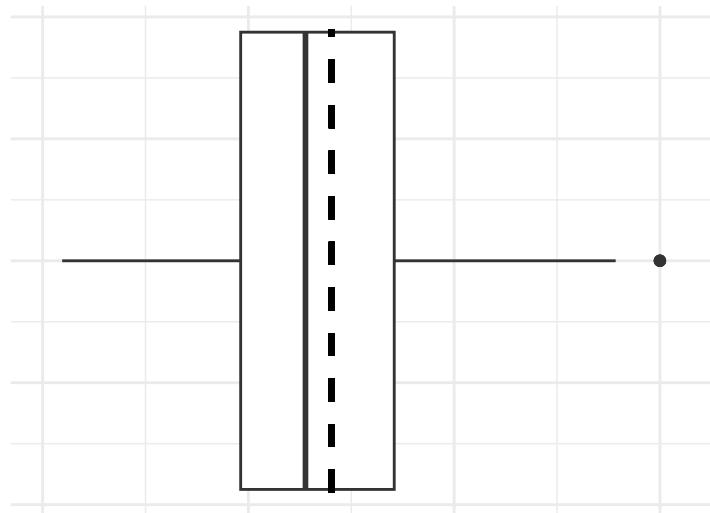
Now let’s consider boxplots from another angle. In Figure 3.5b I’ve plotted the boxplot horizontally, right below the histogram in Figure 3.5a. In this view, we can see that a boxplot is a simplified histogram augmented with central tendency and dispersion statistics. While histograms focus on the frequency distribution of data points, boxplots focus on the data’s quartiles and potential outliers.

I’ve added a dashed line in Figure 3.5a and Figure 3.5b to signal the mean in this set of plots, but it is not typically included. I include the dashed line to make a point: the mean is more sensitive to outliers than the median. As I pointed out in Section 3.1.2, the mean is the sum of all values divided by the number of values. If there are extreme values, the mean will be pulled in the direction of the extreme values. The median, however, is the middle value and a few extreme values have less effect. So, when central tendency is reported, if there is a sizeable difference between the mean and the median, measures of dispersion will be larger and the direction of the difference can be used to infer the presence of outliers.

Returning to outliers, it is important to address them to safeguard the accuracy of the analysis. There are two main ways to address outliers: 1) transform the data and 2) eliminate observations with outliers (**trimming**). Trimming is more extreme as it removes data but can be the best approach for true outliers. Transforming the data is an approach to mitigating



(a) Histogram



(b) Boxplot (horizontal)

Figure 3.5: Histogram and boxplot for the variable `ttr`.

the influence of extreme but valid values. **Transformation** involves applying a mathematical function to the data which changes the scale and/ or shape of the distribution, but does not remove data nor does it change the relative order of the values.

In Figure 3.8, we see two boxplots. Figure 3.6a is the original `ttr` data and Figure 3.6b reflects the data trimmed to remove outliers. In this case, we have removed essays with a type-token ratio of 1.

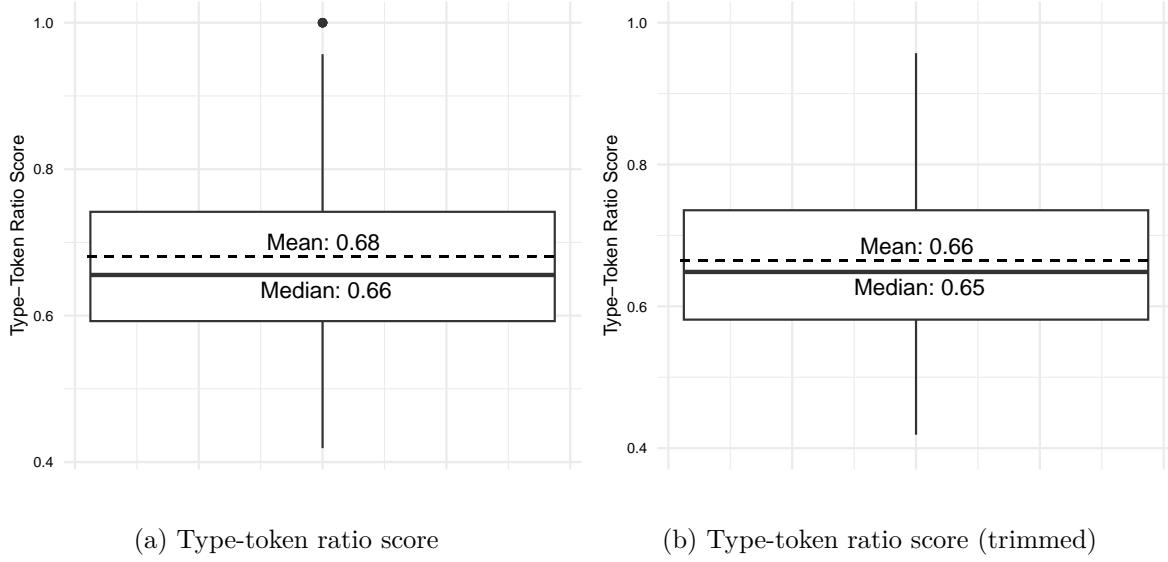


Figure 3.6: Boxplots for `ttr` before and after trimming.

We can now appreciate the relatively larger effect that the outliers had on the mean value of the `ttr` variable. As outliers are removed as the difference between the mean and median will become smaller.

The exploration the data points with histograms and boxplots has helped us to identify outliers. Now we turn to the question of the overall shape of the distribution. The key question is whether the observed distribution of each variable approximates the Normal Distribution, or not.

The **Normal Distribution** is a theoretical distribution where the values are symmetrically dispersed around the central tendency (mean/ median). In terms we can now understand, this means that the mean and median are the same. The Normal Distribution is important because many statistical tests assume that the data distribution is normal or near normal.

Stepping away from our BELC dataset, I've created simulated data that fit normal and non-normal, or skewed, distributions. I present each of these distributions as density plots with mean and median line overlays in Figure 3.7.

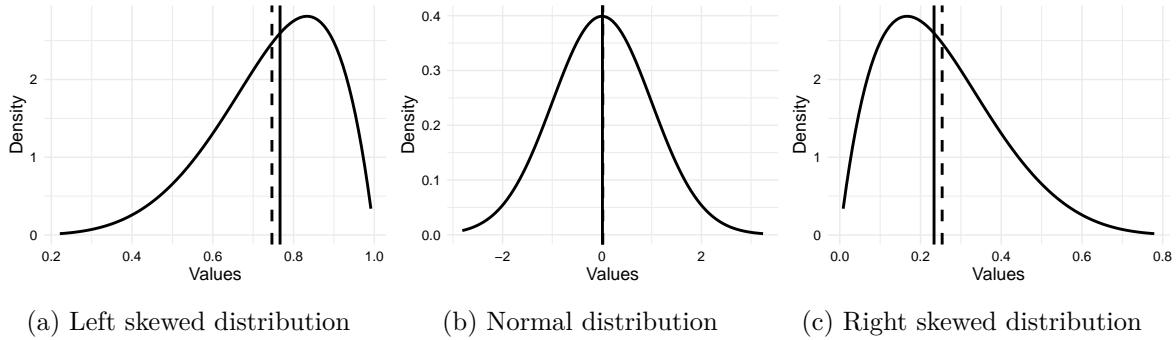


Figure 3.7: Mean and median for normal and skewed distributions.

A **Normal Distribution**, illustrated in Figure 3.7b, is a distribution where the values are symmetrically dispersed around the central tendency (mean/ median). This means that in a theoretical distribution that the mean and median are the same. The Normal Distribution is also known as the **Gaussian Distribution** or the **Bell Curve**, for the hallmark bell shape of the distribution. In this distribution, extreme values are less likely than values near the center.

A **skewed distribution** is not a specific type of distribution but rather a characteristic than many distributions can exhibit where the values are not symmetrically dispersed around the central tendency. A distribution in which values tend to disperse to the left of the central tendency is **left skewed** as in Figure 3.7a and dispersion to the right is **right skewed** as in Figure 3.7c.

Data that are normally, or near-normally distributed are often analyzed using parametric tests while data that exhibit a skewed distributed are often analyzed using non-parametric tests. Divergence from normality is not a binary distinction. Rather, it is a matter of degree. A visual inspection is usually sufficient for experienced researchers to determine whether a distribution is normal or skewed. However, for those who are less experienced or if you want to be more precise, there are two primary measures which can help ascertain the degree to which a distribution is normal: skewness and kurtosis. **Skewness** is a measure of the degree to which a distribution is asymmetrical. **Kurtosis** is a measure of the degree to which a distribution is peaked.

In Table 3.10 I provide the skewness and kurtosis scores for our simulated distributions along with central tendency measures for context.

All things distribution are matters of degree, so there are no hard and fast rules for determining whether a distribution is normal or skewed. However, there are some general guidelines that can be used to determine the degree to which a distribution is normal or skewed, as shown in Table 3.10.

Table 3.9: Skewness and kurtosis for normal and skewed distributions.

distribution	mean	median	histogram	skewness	kurtosis
Left skew	0.746	0.767		-0.711	3.27
Normal	0.016	0.009		0.065	2.93
Right skew	0.254	0.233		0.711	3.27

Table 3.10: Rules of thumb for skewness and kurtosis scores.

(a) Skewness scores

(b) Kurtosis scores

Score Range	Evaluation	Score Range	Evaluation
-0.5 to 0.5	Approximately symmetric	< 3	Less peaked than normal
-1 to -0.5 or 0.5 to 1	Moderately skewed	Equal to 3	Normal peak
< -1 or > 1	Highly skewed	> 3	More peaked than normal

⭐ Dive deeper

Another approach for visually summarizing a single numeric variable is the Empirical Cumulative Distribution Function, or *ECDF*. An ECDF plot is a summary of the cumulative proportion of each of the values of a numeric variable. In addition to providing insight into the distribution of a variable, ECDF plots can be useful in determining what proportion of the values fall above or below a certain percentage of the data.

The question is which type of distribution does each numeric variable in the BELC dataset fit? Comparing the variables `ttr`, `types` and `prop_l2` in Figure 3.8 to the three distributions in Figure 3.7, we see that all three numeric variables in the BELC dataset are skewed to some degree.

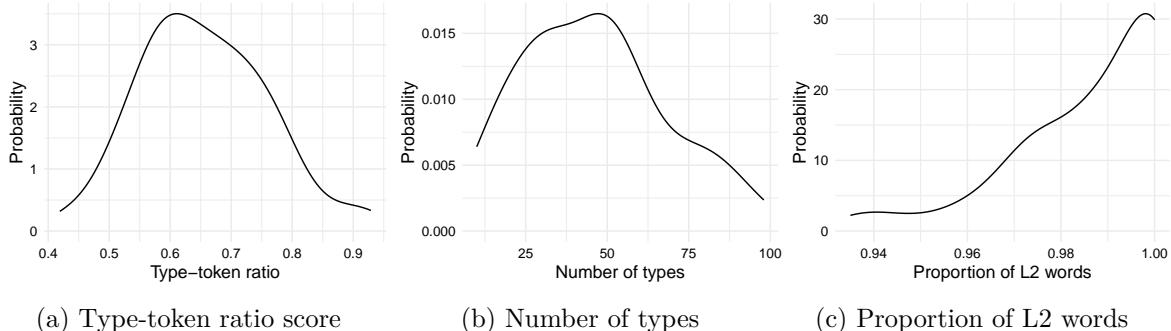


Figure 3.8: Histogram/ Density plots for numeric variables in the BELC dataset.

Table 3.11: Skewness and kurtosis for numeric variables in the BELC dataset.

distribution	mean	median	histogram	skewness	kurtosis
ttr	0.655	0.648		0.319	2.90
types	46.044	46.000		0.407	2.45
tokens	75.338	77.000		0.669	2.98
prop_l2	0.986	0.990		-1.273	4.13

Figure 3.8a for `ttr` has some right skewing but not as much as `types` in Figure 3.8b. `prop_l2` in Figure 3.8c is the most skewed of the three variables. As mentioned earlier, skewed distributions can take many forms, some are more skewed than others.

To view statistics on our three variables in Figure 3.8, we can calculate the skewness and kurtosis.

Given the characteristics of the numeric variables in the BELC dataset, although none of them are perfectly normal, but only `prop_l2` is highly skewed. Therefore, if we intend to use these variables ‘as-is’ in statistical measures or tests, we now know whether to choose parametric or non-parametric alternatives.

In the case that a variable is highly skewed, it is often useful to attempt transform the variable to reduce the skewness. In contrast to scale-based transformations (*e.g.* centering and scaling), shape-based transformations change the scale and the shape of the distribution. The most common shape-based transformation is the logarithmic transformation. The **logarithmic transformation** (log-transformation) takes the log (typically base 10) of each value in a variable. The log-transformation is useful for reducing the skewness of a variable as it compresses large values and expands small values. If the skewness is due to these factors, the log-transformation can help.

It is important to note, however, that if scale-based transformations are to be applied to a variable, they should be applied after the log-transformation as the log of negative values is undefined.

Interdependence

We have covered the first three of the four questions we are interested in asking in a descriptive analysis. The fourth, and last, question is whether there is mutual dependence between variables. If so, what is the directionality and how strong is the dependence? Knowing the answers to these questions will help frame our approach to analysis.

To assess interdependence, the number and information types of the variables under consideration are important. Let’s start by considering two variables. If we are working with

Table 3.13: Contingency tables for categorical variable `sex` and ordinal variable `group` in the BELC dataset.

(a) Counts				(b) Percentages			
group	female	male	Total	group	female	male	Total
T1	7	9	16	T1	43.75%	56.25%	100.00%
T2	11	4	15	T2	73.33%	26.67%	100.00%
T3	13	10	23	T3	56.52%	43.48%	100.00%
T4	9	5	14	T4	64.29%	35.71%	100.00%
Total	40	28	68	Total	58.82%	41.18%	100.00%

two variables, we are dealing with a **bivariate** relationship. Given there are three informational types (categorical, ordinal, and numeric), there are six logical bivariate combinations: categorical-categorical, categorical-ordinal, categorical-numeric, ordinal-ordinal, ordinal-numeric, and numeric-numeric.

The directionality of a relationship will take the form of a tabular or graphic summary depending on the informational value of the variables involved. In Table 3.12, we see the appropriate summary types for each of the six bivariate combinations.

Table 3.12: Appropriate summary types for different combinations of variable types.

	Categorical	Ordinal	Numeric
Categorical	Contingency table	Contingency table/ Bar plot	Pivot table/ Boxplot
Ordinal	-	Contingency table/ Bar plot	Pivot table/ Boxplot
Numeric	-	-	Scatterplot

Let's first start with the combinations that include a categorical or ordinal variable. Categorical and ordinal variables reflect measures of class-type information, with add meaningful ranks to ordinal variables. To assess a relationship with these variable types, a table is always a good place to start. When combined together, a contingency table is the appropriate table. A **contingency table** is a cross-tabulation of two class-type variables, basically a two-way frequency table. This means that three of the six bivariate combinations are assessed with a contingency table: categorical-categorical, categorical-ordinal, and ordinal-ordinal.

In Table 3.13 we see contingency tables for the categorical variable `sex` and ordinal variable `group` in the BELC dataset.

A contingency table may include only counts, as in Table 3.13a, or may include proportions or percentages in an effort to normalize the counts and make them more comparable, as in Table 3.13b.

Table 3.14: Pivot table for the relationship between `group` and `tokens` in the BELC dataset.

group	mean_tokens
T1	35.4
T2	62.5
T3	85.0
T4	118.9

It is sometimes helpful to visualize a contingency table as a bar plot when there are a larger number of levels in either or both of the variables. Again, looking at the relationship between `sex` and `group`, we see that we can plot the counts or the proportions. In Figure 3.9, we see both.

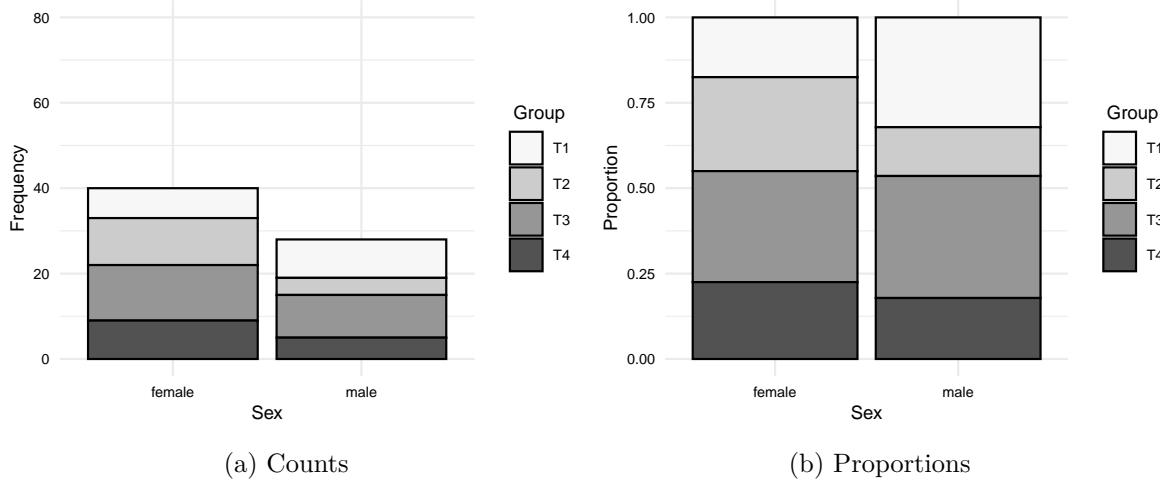


Figure 3.9: Bar plots for the relationship between `sex` and `group` in the BELC dataset.

To summarize and assess the relationship between a categorical or an ordinal variable and a numeric variable, we cannot use a contingency table. Instead, this type of relationship is best summarized in a table using a summary statistic in a **pivot table**. A pivot table is a table in which a class-type variable is used to group a numeric variable by some summary statistic appropriate for numeric variables, *e.g.* mean, median, standard deviation, *etc.*

In Table 3.14, we see a pivot table for the relationship between `group` and `tokens` in the BELC dataset. Specifically, we see the mean number of tokens by group.

We see that the mean number of tokens increases from Group T1 to T4, which is consistent with the idea that the students in the higher groups are writing longer essays.

Although a pivot table may be appropriate for targeted numeric summaries, a visualization is often more informative for assessing the dispersion and distribution of a numeric variable by a categorical or ordinal variable. There are two main types of visualizations for this type of

relationship: a boxplot and a **violin plot**. A violin plot is a visualization that summarizes the distribution of a numeric variable by a categorical or ordinal variable, adding the overall shape of the distribution, much as a density plot does for histograms.

In Figure 3.10, we see both a boxplot and a violin plot for the relationship between `group` and `tokens` in the BELC dataset.

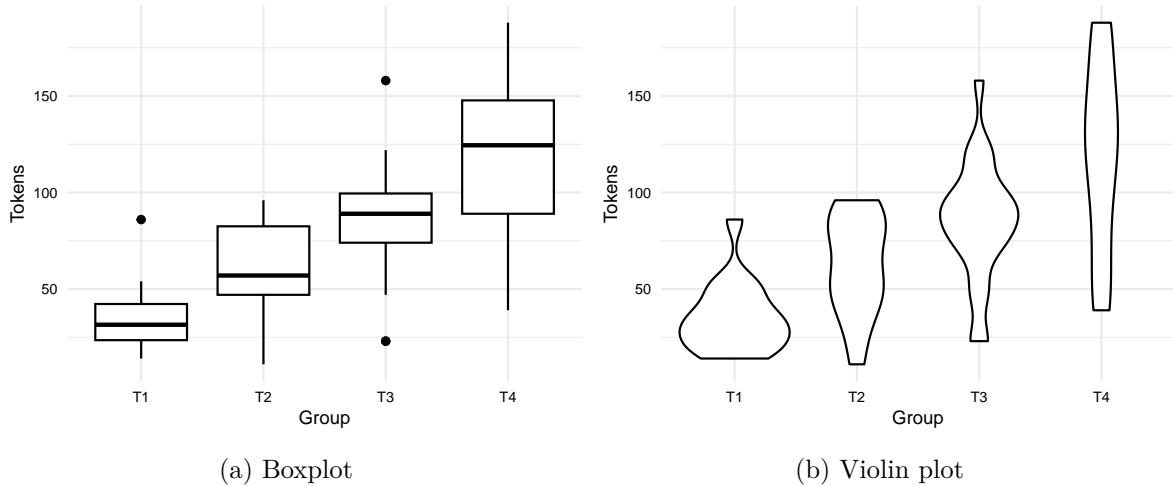


Figure 3.10: Boxplot and violin plot for the relationship between `group` and `tokens` in the BELC dataset.

From the boxplot in Figure 3.10a, we see that the general trend towards more tokens used by students in higher groups. But we can also appreciate the dispersion of the data within each group looking at the boxes and whiskers. On the surface it appears that the data for groups T1 and T3 are closer to each other than groups T2 and T4, in which there is more variability within these groups. Furthermore, we can see outliers in groups T1 and T3, but not in groups T2 and T4. From the violin plot in Figure 3.10b, we can see the same information, but we can also see the overall shape of the distribution of tokens within each group. In this plot, it is very clear that group T4 includes a wide range of token counts.

The last bivariate combination is numeric-numeric. To summarize this type of relationship a scatterplot is used. A **scatterplot** is a visualization that plots each data point as a point in a two-dimensional space, with one numeric variable on the x-axis and the other numeric variable on the y-axis. Depending on the type of relationship you are trying to assess, you may want to add a trend line to the scatterplot. A trend line is a line that summarizes the overall trend in the relationship between the two numeric variables. To assess the extent to which the relationship is linear, a straight line is drawn which minimizes the distance between the line and the points.

In Figure 3.11, we see a scatterplot and a scatterplot with a trend line for the relationship between `ttr` and `types` in the BELC dataset.

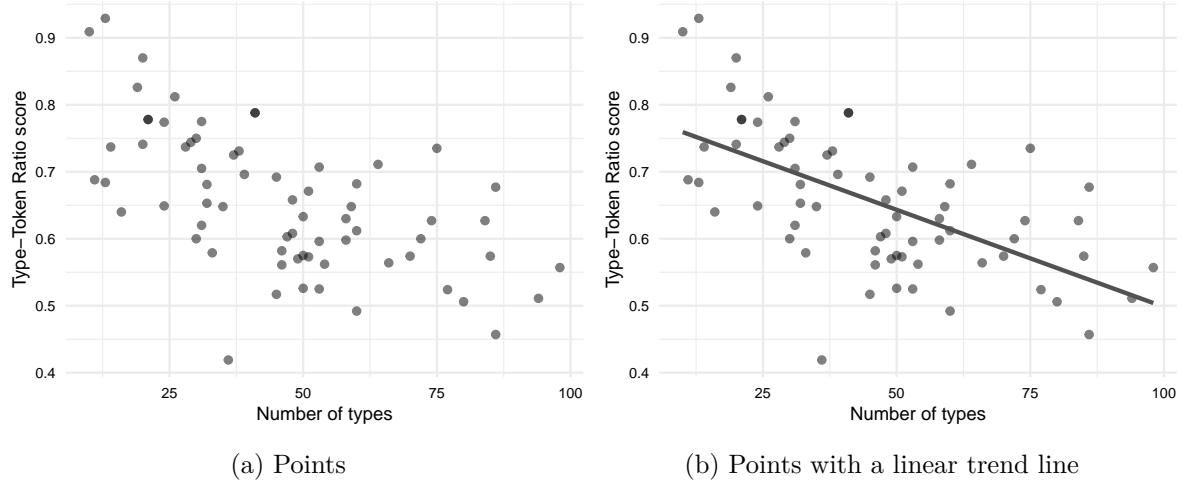


Figure 3.11: Scatter plot for the relationship between `ttr` and `types` in the BELC dataset.

We see that there is an apparent positive relationship between these two variables, which is consistent with the idea that as the number of types increases, the type-token ratio increases. In other words, as the number of unique words increases, so does the lexical diversity of the text. Since we are evaluating a linear relationship, we are assessing the extent to which there is a **correlation** between `ttr` and `types`. A correlation simply means that as the values of one variable change, the values of the other variable change in a consistent manner.

Once a sense of the directionality of a relationship can be established, the next step is to gauge the relative strength, or association. **Association** refers to any relationship in which there is a dependency between two variables. Quantitative measures of association, in combination with tabular and visual summaries, can provide a more complete picture of the relationship between two variables.

There are a number of measures of association, depending on the types of variables being assessed and, for numeric variables, whether the distribution is normal (parametric) or non-normal (non-parametric), as seen in Table 3.15.

Table 3.15: Measures of association or correlation strength for different combinations of variable types.

	Categorical	Ordinal	Numeric Non-parametric	Numeric Parametric
Categorical	Chi-square χ^2 , Cramér's V	Goodman and Kruskal's γ	Rank biserial	Point-biserial
	-	Kendall's τ	Correlation	Correlation
Ordinal	-	-	Kendall's τ	Pearson's r
Numeric Non-parametric	-	-	Kendall's τ	Pearson's r

Table 3.16: Gamma for the relationship between `sex` and `group` in the BELC dataset.

Parameter1	Parameter2	r	CI	CI_low	CI_high
sex.female	group.T1	-0.381	0.95	-0.568	-0.157
sex.female	group.T2	0.389	0.95	0.167	0.575
sex.male	group.T1	0.381	0.95	0.157	0.568
sex.male	group.T2	-0.389	0.95	-0.575	-0.167

Categorical	Ordinal	Numeric <i>Non-parametric</i>	Numeric <i>Parametric</i>
Numeric <i>Parametric</i>	-	-	Pearson's <i>r</i>

Association measures often are expressed as a number between -1 and 1, where 0 indicates no association, -1 indicates a perfect negative association, and 1 indicates a perfect positive association. The closer the number is to 0, the weaker the association. The closer the number is to -1 or 1, the stronger the association. Association statistics are often accompanied by a **confidence interval** (CI), which is a range of values that is likely to contain the true value of the association in the population. The confidence interval is expressed as a percentage, such as 95%, which means that if we were to repeat the study 100 times, 95 of those studies would produce a confidence interval that contains the true value of the association in the population. If the range between the lower and higher bounds of the confidence interval contains 0, then the association is likely no different than chance.

Given these measures and interpretations, let's consider the different types of bivariate relationships we have seen so far in the BELC dataset. The first interdependence we explored involved the categorical variable `sex` and the ordinal variable `group`. This relationship may not be of primary interest to a study on L2 writing, but it is a good example of how to assess the strength of an association between a categorical and ordinal variable. Furthermore, it could be the case that we want to assess whether we have widely unbalanced female/ male proportions in our time groups.

Using Table 3.15, we see that we can use Goodman and Kruskal's γ (gamma) to assess the strength of the association between these two variables. The measures of association in Table 3.16 suggest that the proportion of male participants is higher in group T1 and lower in group T2. However, these associations are moderately strong, as the gamma value is near ± 0.4 .

When paired with Figure 3.9 we can appreciate that groups T1 and T2 have contrasting proportions of females to males and that groups T3 and T4 are more closely proportioned. This observation should be considered when approaching statistical analyses in which categorical variables required (near) equal proportions of categories.

Table 3.17: Pearson's r for the relationship between `group` and `tokens` in the BELC dataset.

Parameter1	Parameter2	r	CI	CI_low	CI_high
group.T1	tokens	-0.520	0.95	-0.675	-0.322
group.T2	tokens	-0.160	0.95	-0.384	0.082
group.T3	tokens	0.161	0.95	-0.080	0.385
group.T4	tokens	0.521	0.95	0.322	0.675

Table 3.18: Pearson's r for the relationship between `ttr` and `types` in the BELC dataset.

Parameter1	Parameter2	r	CI	CI_low	CI_high
ttr	types	-0.606	0.95	-0.738	-0.43

Now let's take a look at a more interesting relationship, the one between the ordinal variable `group` and the numeric variable `tokens`. Since we determined that `tokens` was near normally distributed, we can choose the parametric version of our association measure, Pearson's r . The measures of association in Table 3.17 suggest that there is a negative association between group T1 and a positive one between group T4 and `tokens`, which is consistent with the idea that as the group number increases, the number of tokens increases. These associations are moderate to strong, as the Pearson's r values are near ± 0.5 . However, the other groups (T2 and T3) have very weak associations with `tokens` and the CI includes 0, which means that the association is likely no different than chance.

These association measures suggest that there is a relationship between `group` and `tokens`, but that the relationship is not the same for all groups. This may be due to a number of factors, such as the number of participants in each group, the effect of outliers within particular levels, etc. or may simply underscore that the relationship between `group` and `tokens` is not linear. What we do with this information will depend on our research aims. Whatever the case, we can use these measures to inform our next steps, as we will see in the next section.

Finally, let's look at the relationship between the numeric variables `ttr` and `types`. Since we determined both `ttr` and `types` are normally distributed, we can choose the parametric version of our association measure, Pearson's r . The measures of association in Table 3.18 suggest that there is a negative association between `ttr` and `types`, which is consistent with the idea that as the number of types increases, the type-token ratio decreases. This association is strong, as Pearson's r value is near 0.6.

Before moving on to the next section, it is important to remember that through the process of diagnostic measures, we gain a thorough understanding of our data's characteristics and quality, preparing us for the next step in our analysis. However, remember that these measures do not exist in isolation. The decisions we make at this stage, from handling missing data to understanding the distribution of our variables, can have significant implications on our subsequent analysis. So, this initial step of data analysis deserves our careful attention and scrutiny.

3.2 Analyze

Having ensured that our dataset is clean, valid, and thoroughly understood, we can proceed to the next key stage of our data analysis process - employing analytic methods. The goal of analysis, generally, is to generate knowledge from information. The type of knowledge generated and the process by which it is generated, however, differ and can be broadly grouped into three analysis types: exploratory, predictive, and inferential.

In this section I will provide an overview of how each of these analysis types are tied to research aims and how the general purpose of each type affect: (1) how to *identify* the variables of interest, (2) how to *interrogate* these variables, and (3) how to *interpret* the results. I will structure the discussion of these analysis types moving from the least structured (inductive) to most structured (deductive) approach to deriving knowledge from information with the aim to provide enough information for you to identify these research approaches in the literature and to make appropriate decisions as to which approach your research should adopt.

3.2.1 Explore

In **Exploratory Data Analysis (EDA)**, we use a variety of methods to identify patterns, trends, and relations within and between variables. The goal of EDA is uncover insights in an inductive, data-driven manner. That is to say, that we do not enter into EDA with a fixed hypothesis in mind, but rather we explore intuition, probe anecdote, and follow hunches to identify patterns and relationships and to evaluate whether and why they are meaningful. We are admittedly treading new or unfamiliar terrain letting the data guide our analysis. This means that we can use and reuse the same data to explore different angles and approaches adjusting our methods and measures as we go. In this way, EDA is an iterative, meaning generating process.

In line with the investigative nature of EDA, the identification of variables of interest is a discovery process. We most likely have a intuition about the variables we would like to explore, but we are able to adjust our variables as need be to suit our research aims. When the identification and selection of variables is open, the process is known as **feature engineering**. A process that is much an art as a science, feature engineering leverages a mixture of relevant domain knowledge, intuition, and trial and error to identify features that serve to best represent the data and to best serve the research aims. Furthermore, the roles of features in EDA are fluid –no variable has a special status, as seen in Figure 3.12. We will see that in other types of analysis, some or all the roles of the variables are fixed.

Table 3.19: First ten addresses from the SOTU Corpus.

president	date	delivery	party	addresses
Truman	1946-01-21	written	Democratic	To the Congress of the United States: A quarter...
Truman	1947-01-06	spoken	Democratic	Mr. President, Mr. Speaker, Members of the Cong...
Truman	1948-01-07	spoken	Democratic	Mr. President, Mr. Speaker, and Members of the ...
Truman	1949-01-05	spoken	Democratic	Mr. President, Mr. Speaker, Members of the Cong...
Truman	1950-01-04	spoken	Democratic	Mr. President, Mr. Speaker, Members of the Cong...
Truman	1951-01-08	spoken	Democratic	Mr. President, Mr. Speaker, Members of the Cong...
Truman	1952-01-09	spoken	Democratic	Mr. President, Mr. Speaker, Members of the Cong...
Truman	1953-01-07	written	Democratic	To the Congress of the United States: I have th...
Eisenhower	1953-02-02	spoken	Republican	Mr. President, Mr. Speaker, Members of the Eigh...
Eisenhower	1954-01-07	spoken	Republican	Mr. President, Mr. Speaker, Members of the Eigh...

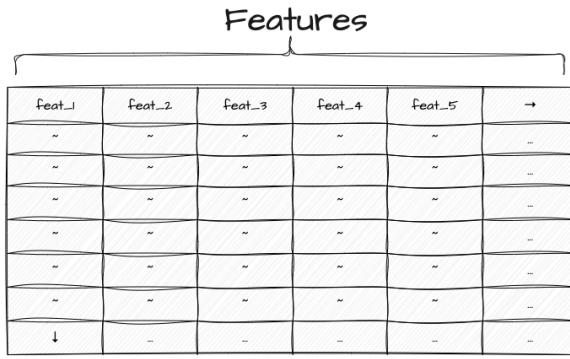


Figure 3.12: Roles of variables in EDA.

For illustrative purposes let's consider the State of the Union Corpus (SOTU) (Benoit 2020). The presidential addresses and a set of metadata variables are included in the corpus. I've subsetted this corpus to only include U.S. presidents since 1946. A tabular preview of the first 10 addresses (truncated for display) can be found in Table 3.19.

A dataset such as this one could serve as a starting point to explore many different types of research questions. In order to maintain research coherence so our efforts to not careen into a free-for-all, we need to tether our feature engineering to a unit of analysis that is relevant to the research question. A **unit of analysis** is the entity that we are interested in studying. Not to be confused with the unit of observation, which is the entity that we are able to observe and measure (Sedgwick 2015).

To demonstrate the distinction, let's look consider different approaches to analyzing the SOTU dataset. For example, the unit of analysis could be the language of particular presidents, party ideology, or political rhetoric in general and the unit of observation could be individual words, phrases, sentences, etc. In some cases the unit of analysis and the unit of observation are the

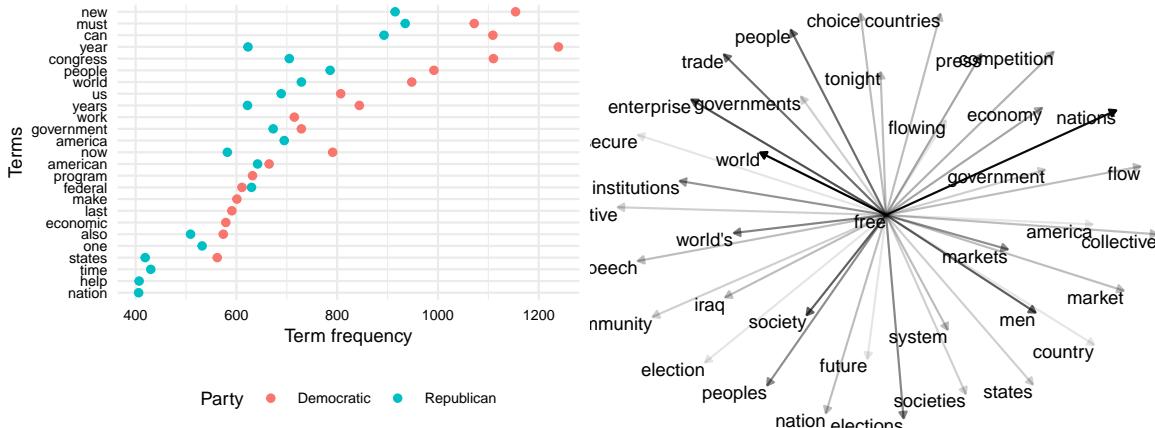
Table 3.20: Some common EDA methods

Descriptive methods	Unsupervised learning methods
Frequency analysis	Cluster analysis
Keyness analysis	Topic Modeling
Co-occurrence analysis	Vector Space Models

same. For example, if we were interested in potential changes use of the word “terrorist” over time in SOTU addresses, the unit of analysis and the unit of observation would be the same –individual addresses. So, depending on the perspective we are interested in investigating, the choice of how to approach engineering features to gain insight will vary.

By the same token, approaches for interrogating the dataset can differ significantly, between research projects and within the same project, but for instructive purposes, let's draw a distinction between descriptive methods and unsupervised learning methods, as seen in Table 3.20.

The first group, **descriptive methods** can be seen as a (more robust) extension of the descriptive statistics covered earlier in this chapter including statistic, tabular, and visual techniques. For example, a frequency analysis of the SOTU dataset could be used to identify the most common words used by U.S. political parties in their addresses, in Figure 3.13a, or a co-occurrence analysis could be used to identify the most common words that appear after the term “free”, in Figure 3.13b, in the dataset.

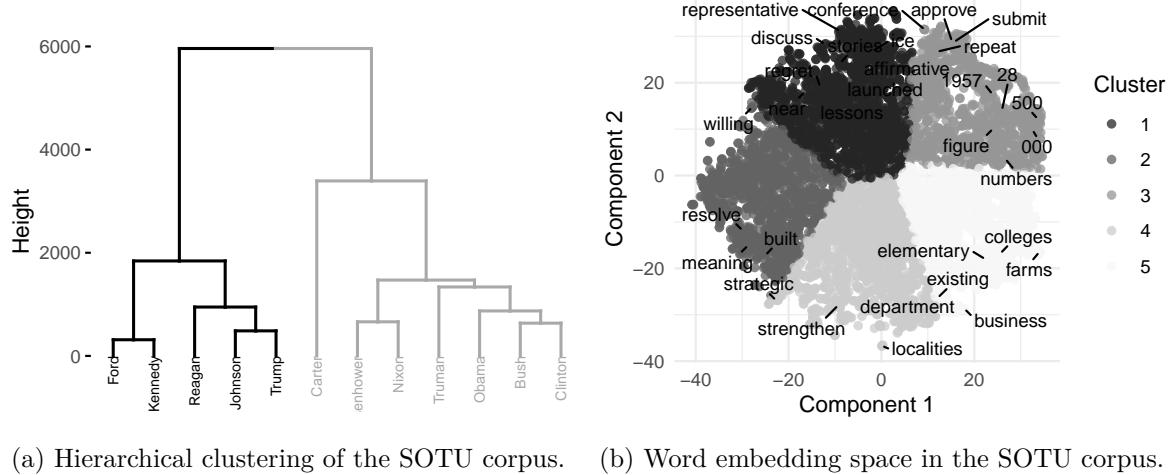


(a) Frequency analysis of the 20 most frequent terms by party.
(b) Co-occurrence analysis of the terms that appear after the term 'free'.

Figure 3.13: Example of descriptive methods applied to the SOTU dataset.

The second group, **unsupervised learning**, is a subtype of machine learning in which an algorithm is used to find patterns within and between variables in the data without any guidance (supervision). In this way, the algorithm, or machine learner, is left to make connections

and associations wherever they may appear in the input data. If we were interested in finding word-use continuities and discontinuities between presidents, we could use a clustering algorithm, seen in Figure 3.14a. Or if we wanted to uncover themes ...  [ADD: modify plot] we could use a vector space model, as in Figure 3.14b.



(a) Hierarchical clustering of the SOTU corpus. (b) Word embedding space in the SOTU corpus.

Figure 3.14: Example of unsupervised learning methods applied to the SOTU dataset.

Either through descriptive, unsupervised learning methods, or a combination of both, EDA employs quantitative methods to summarize, reduce, and sort complex datasets in order to provide the researcher novel perspective to be qualitatively assessed. Exploratory methods produce results that require associative thinking and pattern detection. Speculative as they are, the results from exploratory methods can be highly informative and lead to new insight and inspire further study in directions that may not have been expected.

3.2.2 Predict

Predictive Data Analysis (PDA) employs a variety of techniques to examine and evaluate the association strength between a variable or set of variables, with a specific focus on predicting a target variable. The aim of PDA is to construct models that can accurately forecast future outcomes, using either data-driven or theory-driven approaches. In this process, **supervised learning** methods, where the machine learning algorithm is guided (supervised) by a target outcome variable, are used. This means we don't begin PDA with a completely open-ended exploration, but rather with an objective - accurate predictions. However, the path to achieving this objective can be flexible, allowing us freedom to adjust our models and methods. Unlike EDA, where the entire dataset can be reused for different approaches, PDA requires a portion of the data to be reserved for evaluation, enhancing the validity of our predictive models. Thus, PDA is an iterative process that combines the flexibility of exploratory analysis with the rigor of confirmatory analysis.

Table 3.21: Data dictionary of the ENNTT corpus.

variable	name	variable_type	description
session_id	Session ID	categorical	Unique identifier for each session
seq_speaker_id	Sequential Speaker ID	ordinal	Unique numeric identifier for each speaker
state	State	categorical	Country of the session speaker
language	Language	categorical	Original language in which the sentence was uttered
type	Type	categorical	Category of the speaker: natives, nonnatives, or translators
text	Text	categorical	Text spoken in the session

There are two types of variables in PDA: the outcome variable and the predictor variables, or features. The **outcome variable** is the variable that the researcher is trying to predict. It is the only variable that is necessarily fixed as part of the research question. The features are the variables that are used to predict the outcome variable. An overview of the roles of these variables in PDA is shown in Figure 3.15.

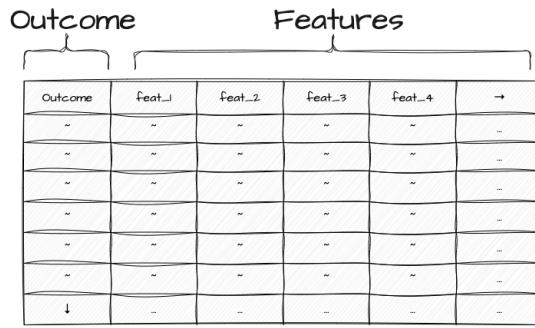


Figure 3.15: Roles of variables in PDA.

Feature selection can be either data-driven or theory-driven. Data-driven features are those that are engineered to enhance predictive power, while theory-driven features are those that are selected based on theoretical relevance.

Let's consider the Europarl corpus of native, non-native and translated texts (ENNTT) (Nisioi et al. 2016). This is a monolingual English corpus of translated and non-translated texts from the European Parliament.

Now depending on our research question, we will have a different outcome variable. If we want to examine the potential linguistic differences between native and non-native speakers, we will select our outcome variable to be the `type` (natives/ nonnatives). The features selected to use to predict `type` depend on our research question. If our research is guided by data, we will choose features that are specifically designed to boost the ability to predict. On the other hand, if our research is steered by theory, we will opt for features that are chosen due to their theoretical significance. In either case, the original dataset will likely need to be transformed.

The approach to interrogating the dataset includes three main steps: feature engineering, model selection, and model evaluation. We've discussed feature engineering, so what is model selection and model evaluation? And how do we go about performing these steps?

Model selection is the process of choosing a machine learning algorithm and set of features that produces the best prediction accuracy for the outcome variable. To refine our approach such that we arrive at the best combination of algorithm and features, we need to train our machine learner on a variety of combinations and evaluate the accuracy of each. We don't want to train and evaluate on the same data, as this would be cheating, and likely would not produce a model that generalizes well to new data. Instead, we split our data into two sets: a training set and a test set. The **training set** is used to train the machine learner, while the **test set** is used to evaluate the accuracy of the model¹. The larger portion of the data, from 60% to 80%, is used for training, while the remaining portion is used for testing.

The elephant in the room is, what type of machine learning algorithm do I use? Well, there are many different types of machine learning algorithms, each with their own strengths and weaknesses. The first rough cut is to decide what type of outcome variable we are predicting: categorical or numeric. If the outcome variable is categorical, we are performing a **classification** task, and if the outcome variable is numeric, we are performing a **regression** task. As we see in Table 3.22, there are various algorithms that can be used for each task.

Table 3.22: Some common supervised learning algorithms used in PDA.

Classification	Regression	Learner type
Logistic Regression	Linear Regression	Interpretable
Decision Tree	Regression Tree	Interpretable
Support Vector Machine	Support Vector Regression	Black box
Multilayer Perceptron	Multilayer Perceptron	Black box

I've included a column in Table 3.22 that characterizes a second consideration which is whether we want an interpretable model or a black box model. When talking about whether a model is interpretable or not, we are not referring to the evaluation of the accuracy of the model. Rather, we are referring to the inner workings of the model itself that allow us to understand how the model is making its predictions. An **interpretable model** is one that can be understood and explored by humans, while a **black box model** is one whose inner workings are not trivially unraveled. The advantage of an interpretable model is that it researchers can go beyond evaluating prediction accuracy and probe feature-outcome associations. On the other hand, if the goal is to simply boost prediction accuracy, interpretability may not be a concern.

¹Depending on the application and the amount of available data, a third *development set* is sometimes created as a pseudo test set to facilitate the testing of multiple approaches on data outside the training set before the final evaluation on the test set is performed.

Finally, there are a number of algorithm-specific strengths and weaknesses to be considered in the process of model selection. These hinge on characteristics of the data, such as the size of the dataset, the number of features, the type of features, and the expected type of relationships between features or on computing resources, such as the amount of time available to train the model or the amount of memory available to store the model.

Model evaluation is the process of assessing the accuracy of the model on the test set, which is a proxy for how well the model will generalize to new data. Model evaluation is performed quantitatively by calculating the accuracy of the model on the training, to develop the model, and ultimately, the test set. The accuracy of a model is calculated by comparing the predicted values to the actual values. For the results of classification tasks, this results in a contingency table, known as a confusion matrix. A **confusion matrix** juxtaposes predicted and actual values allowing various metrics to be calculated, for example in Table 3.23.

Table 3.23: Confusion matrix for the utterance type classification task.

	Predicted: natives	Predicted: nonnatives
Actual: natives	26294 (90% of 29215)	2921 (10% of 29215)
Actual: nonnatives	730 (10% of 7304)	6574 (90% of 7304)

Since regression tasks predict numeric values, the accuracy of the model is calculated by comparing the difference between the predicted and actual values.

It is important to note that whether the accuracy metrics are good is to some degree qualitative judgment. For example, classification accuracy overall may be relatively high, but the model may be performing poorly on one of the classes. In this case, the model may not be useful for the task at hand, despite the overall accuracy.

In the end, PDA offers a versatile path to discover data-driven insights, to probe theory-driven associations, or even simply to perform tasks that are too complex or time-consuming for humans to perform.

3.2.3 Infer

The most commonly recognized of the three data analysis approaches, **Inferential data analysis (IDA)** is the bread-and-butter of science. IDA is a deductive, theory-driven approach in which all aspects of analysis stem from a pre-determined premise, or hypothesis, about the nature of a relationship in the world and then aims to test whether this relationship is statistically supported given the evidence. Since the goal is to infer conclusions about a certain relationship in the population based on a statistical evaluation of a (corpus) sample, the representativeness of the sample is of utmost importance. Furthermore, the use of the data is limited to the scope of the hypothesis –that is, the data cannot be used for exploratory purposes.

Table 3.24: Data dictionary of the SWDA dataset.

variable	name	description	variable_type
doc_id	Document ID	Unique identifier for each document	numeric
speaker_id	Speaker ID	Unique identifier for each speaker	numeric
sex	Speaker Gender	Gender of the speaker	character
damsl_tag	DAMSL Tag	A tag indicating the dialogue act of the utterance	character
utterance_text	Utterance Text	The text of the speaker's utterance	character

The selection of variables and the roles they play in the analysis are determined by the hypothesis. In a nutshell, a **hypothesis** is a formal statement about the state of the world. This statement is theory-driven meaning that it is predicated on previous research. We are not exploring or examining relationships, rather we are testing a specific relationship. In practice, however, we are in fact proposing two mutually exclusive hypotheses. The first is the **Alternative Hypothesis**, or H_1 . This is the hypothesis I just described –the statement grounded in the previous literature outlining a predicted relationship. The second is the **Null Hypothesis**, or H_0 . This is the flip-side of the hypothesis testing coin and states that there is no difference or relationship. Together H_1 and H_0 cover all logical outcomes.

To connect hypotheses to variable selection and variable roles, let's consider a study in which a researcher is investigating the claim that men and women differ in terms of the number of questions they use in spontaneous conversations. The unit of analysis is individuals (i.e. men and women) and the unit of observation is (spontaneous) conversations.

A dataset based on the Switchboard Dialog Act Corpus (SWDA) (University of Colorado Boulder 2008), seen in Table 3.24, aligns well with this investigation. It is a large collection of transcribed telephone conversations between strangers. The dataset includes gender information for each participant and dialog act annotation for each utterance, including a range of question types.

The Alternative Hypothesis may be formulated in this way:

H_1 : Men and women differ in the frequency of the use of questions in spontaneous conversations.

The Null Hypothesis, then, would be a statement describing the remaining logical outcomes. Specifically:

H_0 : Men and women do *not* differ in the frequency of the use of questions in spontaneous conversations.

Now, in standard IDA one variable is the dependent variable and one or more variables are predictor variables. The **dependent variable**, sometimes referred to as the outcome or response variable, is the variable which contains the information which is hypothesized to depend on the information in the predictor variable(s). It is the variable whose variation a research study seeks to explain. A **predictor variable**, sometimes referred to as a independent

or explanatory variable, is a variable whose variation is hypothesized to explain the variation in the dependent variable.

Returning to our hypothetical study and the hypotheses presented, we can identify the variables in our study and map them to their roles. The frequency of questions used by each speaker would be our dependent variable and the biological sex of the speakers our predictor variable. This is so because H_1 states the proposition that a speaker's sex will predict the frequency of questions used. The next step would be to operationalize what we mean by 'frequency of questions' and then transform the dataset to reflect this definition.

In our hypothetical study we've identified two variables, one dependent and one predictor. It is important to keep in mind that there can be multiple predictor variables in cases where the dependent variable's variation is predicted to be related to multiple variables. This relationship would need to be explicitly part of the original hypothesis, however. Due to the increasing difficulty for interpretation, in practice, IDA studies rarely include more than two or three predictor variables in the same analysis.

Predictor variables add to the complexity of a study because they are part of our research focus, specifically our hypothesis. It is, however, common to include other variables which are not of central focus, but are commonly assumed to contribute to the explanation of the variation of the dependent variable. Let's assume that the background literature suggests that the age of speakers also plays a role in the number of questions that men and women use in spontaneous conversation. Let's also assume that the data we have collected includes information about the age of speakers. If we would like to factor out the potential influence of age on the use of questions and focus on the particular predictor variables we've defined in our hypothesis, we can include the age of speakers as a **control variable**. A control variable will be added to the statistical analysis and documented in our report but it will not be included in the hypothesis nor interpreted in our results.

We can now see in Figure 3.16 the variables roles assigned to variables in a hypothesis-driven study.

Table 3.25: Common monofactorial tests used in IDA.

Variable roles		Test
Dependent	Predictor	
Categorical	Categorical	Pearson's Chi-squared test
Numeric	Categorical	Student's t-Test
Numeric	Numeric	Pearson's correlation test

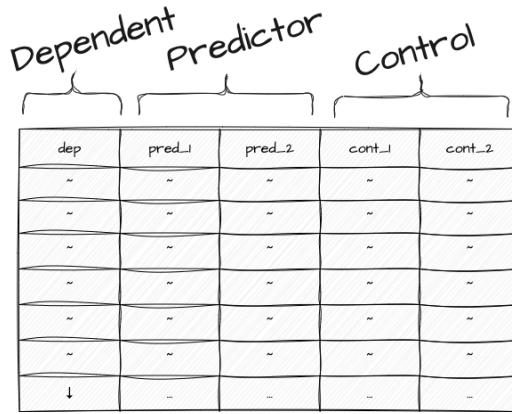


Figure 3.16: Roles of variables in IDA.

At this point let's look at the main characteristics that need to be taken into account to statistically interrogate the variables we have chosen to test our hypothesis. The type of statistical test that one chooses is based on (1) the informational value of the dependent variable and (2) the number of predictor variables included in the analysis. Together these two characteristics go a long way in determining the appropriate class of statistical test, but other considerations about the distribution of particular variables (i.e. normality), relationships between variables (i.e. independence), and expected directionality of the predicted effect may condition the appropriate method to be applied.

As you can imagine, there are a host of combinations and statistical tests that apply in particular scenarios, too many to consider in given the scope of this coursebook (see S. Th. Gries (2013) and Paquot and Gries (2020) for a more exhaustive description). Below I've summarized some common statistical scenarios and their associated tests which focus on the juxtaposition of informational values and the number of variables, leaving aside alternative tests which deal with non-normal distributions, ordinal variables, *etc.*

In Table 3.25 we see **monofactorial tests**, tests with only one predictor variable.

Table 3.26 includes a listing of **multifactorial tests**, tests with more than one predictor and/or control variables.

Table 3.26: Common multifactorial tests used in IDA.

Variable roles			
Dependent	Predictor	Control	Test
Categorical	<i>varied</i>	<i>varied</i>	Logistic regression
Numeric	<i>varied</i>	<i>varied</i>	Linear regression

IDA relies heavily on quantitative evaluation methods to draw conclusions that can be generalized to the target population. It is key to understand that our goal in hypothesis testing is not to find evidence in support of H_1 , but rather to assess the likelihood that we can reliably reject H_0 . The metric used to determine if there is sufficient evidence is based on the probability that given the nature of the relationship and the characteristics of the data, the likelihood of there being no difference or relationship is low. The threshold for likelihood has traditionally been summarized in the p -value statistic. In the Social Sciences, a p -value lower than .05 is considered *statistically significant* which when interpreted correctly means that there is more than a 95% chance that the observed relationship would not be predicted by H_0 . Note that we are working in the realm of probability, not in absolutes, therefore an analysis that produces a significant result does not prove H_1 is correct or that H_0 is incorrect, for that matter. A margin of error is always present. For this reason, other metrics such as effect size and confidence intervals are also used to interpret the results of statistical tests.

3.3 Communicate

Conducting research should be enjoyable and personally rewarding but the effort you have invested and knowledge you have generated should be shared with others. Whether part of a blog, presentation, journal article, or for your own purposes it is important to document your analysis results and process in a way that is informative and interpretable. This enhances the value of your work, allowing others to learn from your experience and build on your findings.

3.3.1 Report

The most widely recognized form of communicating research is through a report. A report is a narrative of your analysis, including the research question, the data you used, the methods you applied, and the results you obtained. We are both reporting our findings and documenting our process to inform others of what we did and why we did it but also to invite readers to evaluate our findings for themselves. The scientific process is a collaborative one and evaluation by peers is a key component of the process.

The audience for your report will determine the level of detail and the type of information you will need to include in your report but there are some common elements to reference in

any report. First, the research question and/ or hypothesis should be clearly stated and the motivation for the question should be explained. This will help the reader understand the context of the analysis and the importance of the results. Second, diagnostic procedures to verify or describe the data should be explained. This may include anomaly correction, missing data, data transformation, etc. and/ or descriptive summaries of the data including assessments of individual variables (central tendency, dispersion, distribution) and/ or relationships between variables (association strength). Third, a blueprint of the methods used will describe the variable selection process, how the variables are operationalized, what analysis methods are employed, and how the variables are used in the statistical analysis. Fourth, the results from the analysis are reported. Reporting details will depend on the type of analysis and the particular method(s) employed. For inferential analyses this will include the test statistic(s) and some measure of confidence. In predictive analyses, accuracy results will be reported. For exploratory analyses, the reporting of results will vary and often include visualizations and metrics that require more human interpretation than the other analysis types. Finally, the results are interpreted in light of the research question and/ or hypothesis. This will include a discussion of the limitations of the analysis and a discussion of the implications of the results for future research.

3.3.2 Document

While a good report will include the most vital information to understand the procedures, results, and findings of an analysis, there is much more information generated in the course of an analysis which does not traditionally appear in prose. If a research project is conducted programmatically, however, data, code, and documentation can be made available to others as part of the communication process. Increasingly, researchers are sharing their data and code as part of the publication process. This allows others to reproduce the analysis and verify the results contributing to the collaborative nature of the scientific process.  [CITATION]

Together, data, code, and documentation form a **research compendium**. As you can imagine the research process can quickly become complex and unwieldy as the number of files and folders grows. If not organized properly, it can be difficult to find the information you need. Furthermore, if not documented, decisions made in the course of the analysis can be difficult or impossible to trace. For this reason it is recommendable to follow a set of best practices for organizing and documenting your research compendium.

We will have more to say about this in the next chapter but for now it will suffice to point to some key elements in a research compendium. First, the data used in the analysis should be saved as a separate file(s). As a given research project progresses to analysis, the data may be transformed and manipulated to best fit the needs of the analysis. Preserving the data at each stage adds to the complete picture of the data from collection to analysis. Second, since you are working programmatically, you can share your precise analysis step-by-step in code form. This allows others to reproduce your analysis and verify your results. Including code comments provides additional information to communicate the steps taken and your thought

process. Finally, a codebook documents any additional information that helps understand the research better. This will often include guides for installing software and running the code to reproduce the analysis and an overview of the aims of the scripts and the contents of the data and datasets.

Summary

In this chapter we have focused on description and analysis –the third component of DIKI Hierarchy. This process is visually summarized in Figure 3.17.

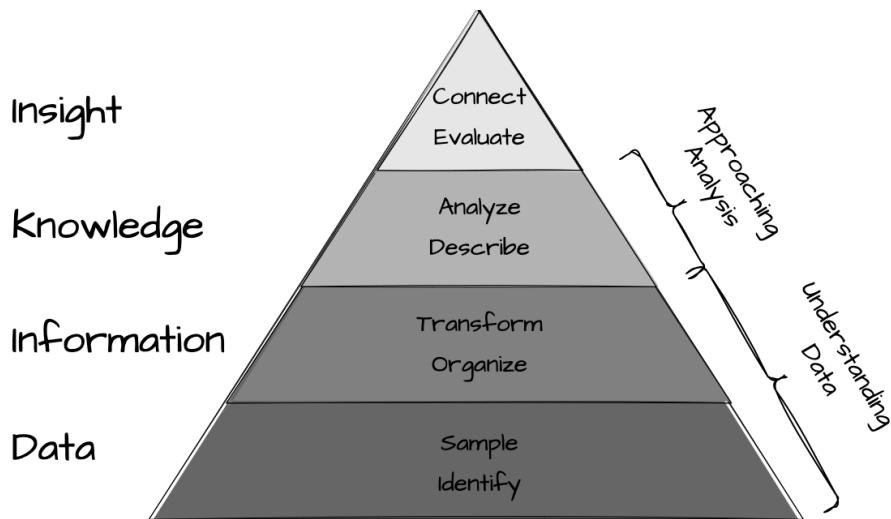


Figure 3.17: Approaching analysis: visual summary

Building on the strategies covered in Chapter 2 “Understanding data” to derive a rich relational dataset, in this chapter we outlined key points in approaching analysis. The first key step in any analysis is to perform a diagnostic assessment of the individual variables and relationships between variables. To select the appropriate descriptive measures we covered the various informational values that a variable can take. In addition to providing key information for reporting purposes, descriptive measures are important to explore so the researcher can get a better feel for the dataset before conducting an analysis.

We outlined three data analysis types in this chapter: exploratory, predictive, and inferential. Each of these embodies distinct approaches to deriving knowledge from data. Ultimately the choice of analysis type is highly dependent on the goals of the research. Inferential analysis is centered around the goal of testing a hypothesis, and for this reason it is the most highly structured approach to analysis. This structure is aimed at providing the mechanisms to draw conclusions from the results that can be generalized to the target population. Predictive

analysis has a less-ambitious but at times more relevant goal of examining the extent to which a given relationship can be established from the data to provide a model of language that can accurately predict an outcome using new data. This methodology is highly effective for applying different algorithmic approaches and examining relationships between an outcome variable and various configurations of variables. The ability to explore the data in multiple ways, is also a key strength of employing an exploratory analysis. The least structured and most variable of the analysis types, exploratory analyses are a powerful approach to generating knowledge from data in an area where clear predictions cannot be made.

I rounded out this chapter with a short description of the importance of communicating the analysis process and results. Reporting, in its traditional form, is documented in prose in an article. This reporting aims to provide the key information that a reader will need to understand what was done, how it was done, and why it was done. This information also provides the necessary information for reader's with a critical eye to understand the analysis in more detail. Yet even the most detailed reporting in a write-up still leaves many practical, but key, points of the analysis obscured. A programming approach provides the procedural steps taken that when shared provide the exact methods applied. Together with the write-up, a research compendium which provides the scripts to run the analysis and documentation on how to run the analysis forms an integral part of creating reproducible research.

Activities

-  Add description of outcomes ...

Recipe

What: Descriptive assessment of datasets^a

How: Read Recipe 3 and participate in the Hypothes.is online social annotation.

Why: To explore appropriate methods for summarizing variables in datasets given the number and informational values of the variable(s).

^a<https://qtalr.github.io/qtalrkit/articles/recipe-3.html>

Lab

What: Descriptive assessment of datasets^a

How: Clone, fork, and complete the steps in Lab 3.

Why: To identify and apply the appropriate descriptive methods for a vector's informational value and to assess both single variables and multiple variables with the appropriate statistical, tabular, and/ or graphical summaries.

^a<https://github.com/qtalr/lab-3>

Questions

Conceptual questions

- What are the key differences between assessment and analysis?
- What are the potential measures of central tendency and dispersion for a variable? Does it depend on the informational value of the variable?
- Consider the following variables: X = number of children, Y = number of siblings, Z = number of siblings who are older than the participant. Which of these variables are categorical, ordinal, numeric? What are the measures of central tendency and dispersion for each variable?
- What type(s) of tables or plots are appropriate for summarizing a variable? What type(s) of tables or plots are appropriate for summarizing the relationship between two variables?
- In the following variables and informational values, identify if the plots are appropriate for summarizing the relationship.
 ...
- What are the key differences between exploratory, predictive, and inferential analysis?
- How do the goals of the research influence the choice of analysis type?
- Given the following research questions, identify which type of analysis is most appropriate and why.
 ...
- How are the results of inferential, predictive, and exploratory analysis evaluated?
- Research compendia are an important part of reproducible research. What are the key components of a research compendium? What are the benefits of sharing a research compendium?

Technical exercises

- Create a contingency table for the following variables:
- Create a plot for the following variables:
- Report these tables and plots with a short interpretation of what they show.
- ...

4 Framing research



Draft

Ready for review.

Thus, the task is, not so much to see what no one has seen yet; but to think what nobody has thought yet, about that what everybody sees.

— Arthur Schopenhauer



Outcomes

- Identify a research area and problem by listing key strategies and describing their contribution towards research identification.
- Explain the significance of a well-framed research question in guiding the overall research project.
- Comprehend how the conceptual and practical steps involved in developing a research blueprint aid not only the researcher but also the broader scientific community.

At this point in this part of the textbook, we have covered Data, Information, and Knowledge from the Data to Insight Hierarchy. The goal has been to provide an orientation to the main building blocks of doing text analysis. Insight is the last component of the hierarchy. However, in practical terms, it is the first step to address in an research project as goals of a research project influence all subsequent steps.

In this chapter we discuss how to frame research, that is how to position your research project's findings to contribute insight to understanding of the world. We will cover how to connect with the literature, selecting a research area and identifying a research problem, and how to design research best positioned to return relevant findings that will connect with this literature, establishing a research aim and research question. We will round out this chapter with a guide on developing a research blueprint —a working plan to organize the conceptual and practical steps to implement the research effectively and in a way that supports communicating the research findings and the process by which the findings were obtained.

Table 4.1: Characteristics of research (Cross, 2006).

Characteristic	Description
Purposive	Based on identification of an issue or problem worthy and capable of investigation
Inquisitive	Seeking to acquire new knowledge
Informed	Conducted from an awareness of previous, related research
Methodical	Planned and carried out in a disciplined manner
Communicable	Generating and reporting results which are feasible and accessible by others

>_ Swirl lesson

What: Version control^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why:  To

^a<https://github.com/qtalr/lessons>

4.1 Frame

Together a research area, problem, aim and question and the research blueprint that forms the conceptual and practical scaffolding of the project ensure from the outset that the project is solidly grounded in the main characteristics of good research. These characteristics, summarized by Cross (2006), are found in Table 4.1.

With these characteristics in mind, let's get started with the first component to address – connecting with the literature.

4.2 Connect

4.2.1 Research area

The first decision to make in the research process is to identify a research area. A **research area** is a general area of interest where a researcher wants to derive insight and make a contribution to understanding. For those with an established research trajectory in language, the area of research to address through text analysis will likely be an extension of their prior work. For others, which include new researchers or researcher's that want to explore new areas of language research or approach an area through a language-based lens, the choice of area may be less obvious. In either case, the choice of a research area should be guided by a desire to contribute something relevant to a theoretical, social, and/ or practical matter of personal

interest. Personal relevance goes a long way to developing and carrying out **purposive** and **inquisitive** research.

So how do we get started? The first step is to reflect on your own areas of interest and knowledge, be it academic, professional, or personal. Language is at the heart of the human experience and therefore found in some fashion anywhere one seeks to find it. But it is a big world and more often than not the general question about what area to explore language use is sometimes the most difficult. To get the ball rolling, it is helpful to peruse disciplinary encyclopedias or handbooks of linguistics and language-related academic fields (e.g. Encyclopedia of Language and Linguistics¹ (Brown 2005), A Practical Guide to Electronic Resources in the Humanities² (Dubnjakovic and Tomlin 2010), Routledge encyclopedia of translation technology³ (Chan 2014))

A more personal, less academic, approach is to consult online forums, blogs, *etc.* that one already frequents or can be accessed via an online search. For example, Reddit⁴ has a wide variety of active subreddits (r/LanguageTechnology⁵, r/Linguistics⁶, r/corpuslinguistics⁷, r/DigitalHumanities⁸, *etc.*). Twitter and Facebook also have interesting posts on linguistics and language-related fields worth following. Through one of these social media sites you may find particular people that maintain a blog worth browsing. For example, I follow Julia Silge⁹, Rachel Tatman¹⁰, and Ted Underwood¹¹, *inter alia*. Perusing these resources can help spark ideas and highlight the kinds of questions that interest you.

Regardless of whether your inquiry stems from academic, professional, or personal interest, try to connect these findings to academic areas of research. Academic research is highly structured and well-documented and making associations with this network will aid in subsequent steps in developing a research project.

4.2.2 Research problem

Once you've made a rough-cut decision about the area of research, it is now time to take a deeper dive into the subject area and jump into the literature. This is where the rich structure of disciplinary research will provide aid to traverse the vast world of academic knowledge and

¹<https://www.sciencedirect.com/referencework/9780080448541/encyclopedia-of-language-and-linguistics>

²<https://www.sciencedirect.com/book/9781843345978/a-practical-guide-to-electronic-resources-in-the-humanities>

³<https://www.routledgehandbooks.com/doi/10.4324/9781315749129>

⁴<https://www.reddit.com/>

⁵<https://www.reddit.com/r/LanguageTechnology/>

⁶<https://www.reddit.com/r/linguistics/>

⁷<https://www.reddit.com/r/corpuslinguistics/>

⁸<https://www.reddit.com/r/DigitalHumanities/>

⁹<https://juliasilge.com/>

¹⁰<http://www.rctatman.com/>

¹¹<https://tedunderwood.com/>

Table 4.2: A list of some linguistics journals.

Resource
Corpora
Corpus Linguistics and Linguistic Theory
International Journal of Corpus Linguistics
International Journal of Language Studies
Journal of Child Language
Journal of Linguistic Geography
Journal of Quantitative Linguistics

Table 4.3: A list of some humanities journals.

Resource	Description
Digital Humanities Quarterly	Digital Humanities
Digital Scholarship in the Humanities	DSH or Digital Scholarship
Journal of Cultural Analytics	Cultural Analytics

identify a research problem. A **research problem** highlights a particular topic of debate or uncertainty in existing knowledge which is worthy of study.

Surveying the relevant literature is key to ensuring that your research is **informed**, that is, connected to previous work. Identifying relevant research to consult can be a bit of a ‘chicken or the egg’ problem –some knowledge of the area is necessary to find relevant topics, some knowledge of the topics is necessary to narrow the area of research. Many times the only way forward is to jump into conducting searches. These can be world-accessible resources (*e.g.* Google Scholar¹²) or limited-access resources that are provided through an academic institution (*e.g.* Linguistics and Language Behavior Abstracts¹³), ERIC¹⁴, PsycINFO¹⁵, *etc.*). Some organizations and academic institutions provide research guides¹⁶ to help researcher’s access the primary literature.

Another avenue to explore are journals dedicated to areas in which linguistics and language-related research is published. In Table 4.2, Table 4.3, and Table 4.4, I’ve listed a number of highly visible journals in linguistics, digital humanities, and computational linguistics.

To explore research related to text analysis it is helpful to start with the (sub)discipline name(s) you identified in when selecting your research area, more specific terms that occur to you or key terms from the literature, and terms such as ‘corpus study’ or ‘corpus-based’. The results from first searches may not turn out to be sources that end up figuring explicitly in your research, but

¹²<https://scholar.google.com/>

¹³<https://about.proquest.com/en/products-services/llba-set-c>

¹⁴<https://eric.ed.gov/>

¹⁵<https://www.ebsco.com/products/research-databases/apa-psycinfo>

¹⁶<https://guides.zsr.wfu.edu/linguistics>

Table 4.4: A list of some computational linguistics journals.

Resource
Computational Linguistics
LREC Conferences
Transactions of the Association for Computational Lin

it is important to skim these results and the publications themselves to mine information that can be useful to formulate better and more targeted searches. Relevant information for honing your searches can be found throughout an academic publication (article or book). However, pay particular attention to the abstract, in articles, and the table of contents, in books, and the cited references. Abstracts and tables of contents often include discipline-specific jargon that is commonly used in the field. In some articles there is even a short list of key terms listed below the abstract which can be extremely useful to seed better and more precise search results. The references section will contain relevant and influential research. Scan these references for publications which appear to narrow in on topic of interest and treat it like a search in its own right.

Once your searches begin to show promising results it is time to keep track and organize these references. Whether you plan to collect thousands of references over a lifetime of academic research or your aim is centered around one project, software such as Zotero¹⁷¹⁸, Mendeley²⁰, or BibDesk²¹ provide powerful, flexible, and easy-to-use tools to collect, organize, annotate, search, and export references. Citation management software is indispensable for modern research –and often free!

As your list of relevant references grows, you will want to start the investigation process in earnest. Begin skimming (not reading) the contents of each of these publications, starting with the most relevant first²². Annotate these publications using highlighting features of the citation management software to identify: (1) the stated goal(s) of the research, (2) the data source(s) used, (3) the information drawn from the data source(s), (4) the analysis approach employed, and (5) the main finding(s) of the research as they pertain to the stated goal(s). Next, in your own words, summarize these five key areas in prose adding your summary to the notes feature of the citation management software. This process will allow you to efficiently gather and document references with the relevant information to guide the identification of a research problem, to guide the formation of your problem statement, and ultimately, to support the literature review that will figure in your project write-up.

From your preliminary annotated summaries you will undoubtedly start to recognize overlapping and contrasting aspects in the research literature. These aspects may be topical,

¹⁷<https://www.zotero.org/>

¹⁸Zotero Guide¹⁹

²⁰<https://www.mendeley.com/reference-management/reference-manager>

²¹<https://bibdesk.sourceforge.io/>

²²Or what appears to be most relevant. This may change as you start to take a closer look.

theoretical, methodological, or appear along other lines. Note these aspects and continue to conduct more refined searches, annotate new references, and monitor for any emerging patterns of uncertainty or debate (gaps) which align with your research interest(s). When a promising pattern takes shape, it is time to engage with a more detailed reading of those references which appear most relevant highlighting the potential gap(s) in the literature.

At this point you can focus energy on more nuanced aspects of a particular gap in the literature with the goal to formulate a problem statement. A **problem statement** directly acknowledges a gap in the literature and puts a finer point on the nature and relevance of this gap for understanding. This statement reflects your first deliberate attempt to establish a line of inquiry. It will be a targeted, but still somewhat general, statement framing the gap in the literature that will guide subsequent research design decisions.

4.3 Define

4.3.1 Research aim

With a problem statement in hand, it is now time to consider the goal(s) of the research. A **research aim** frames the type of inquiry to be conducted. Will the research aim to explore, examine, or explain? In other words, will the research seek to uncover novel relationships, assess the potential strength of a particular relationship, or test a particular relationship? As you can appreciate, the research aim is directly related to the analysis methods we touched upon in Chapter 3.

To gauge how to frame your research aim, reflect on the literature that led you to your problem statement and the nature of the problem statement itself. If the gap at the center of the problem statement is a lack of knowledge, your research aim may be exploratory. If the gap concerns a conjecture about a relationship, then your research may take a predictive approach. When the gap points to the validation of a relationship, then your research will likely be inferential in nature. Before selecting your research aim it is also helpful to consult the research aims of the primary literature that led you to your research statement.

Typically, a problem statement addressing a subtle, specific issue tends to adopt research objectives similar to prior studies. In contrast, a statement focusing on a broader, more distinct issue is likely to have unique research goals. Yet, this is more of a guideline than a strict rule.

It's crucial to understand both the existing literature and the nature of various types of analyses. Being clear about your research goals is important to ensure that your study is well-placed to produce results that add value to the current understanding in an informed manner.

4.3.2 Research question

The next step in research design is to craft the research question. A **research question** is clearly defined statement which identifies an aspect of uncertainty and the particular relationships that this uncertainty concerns. The research question extends and narrows the line of inquiry established in the research statement and research aim. To craft a research question, we can use the research statement for the content and the research aim for the form.

Form

The form of a research question will vary based on the research aim, which as I mentioned, is intimately connected to the analysis approach. For inferential-based research, the research question will actually be a statement, not a question. This statement makes a testable claim about the nature of a particular relationship –*i.e.* asserts a hypothesis.

For illustration, let's return to the hypothesis (H_1) we previously sketched out in Chapter 3, leaving aside the implicit null hypothesis, seen in Example 4.1.

Example 4.1. Women use more questions than men in spontaneous conversations.

For predictive- and exploratory-based research, the research question is in fact a question. A reframing of the example hypothesis for a predictive-based research question might take the form seen in Example 4.2.

Example 4.2. Can the number of questions used in spontaneous conversations predict if a speaker is male or female?

And a similar exploratory-based research question might take the form seen in Example 4.3.

Example 4.3. Do men and women differ in terms of the number of questions they use in spontaneous conversations?

The central research interest behind these hypothetical research questions is, admittedly, quite basic. But from these simplified examples, we are able to appreciate the similarities and differences between the forms of research statements that correspond to distinct research aims.

Content

In terms of content, the research question will make reference to two key components. First, is the unit of analysis. The **unit of analysis** is the entity which the research aims to investigate. For our three example research aims, the unit of analysis is the same, namely *speakers*. Note, however, that the current unit of analysis is somewhat vague in the example research questions. A more precise unit of analysis would include more information about the population from which the speakers are drawn (*i.e.* English speakers, American English speakers, American English speakers of the Southeast, *etc.*).

The second key component is the unit of observation. The **unit of observation** is the primary element on which the insight into the unit of analysis is derived and in this way constitutes the essential organizational unit of the dataset to be analyzed. In our examples, the unit of observation, again, is unchanged and is *spontaneous conversations*. Note that while the unit of observation is key to identify as it forms the organizational backbone of the research, it is very common for the research to derive variables from this unit to provide evidence to investigate the research question.

In examples 4.1, 4.2, and 4.3, we identified the number of conversations as part of the research question. Later in the research process it will be key to operationalize this variable. For example, will the number of conversations be the total number of conversations in the dataset or will it be the average number of conversations per speaker? These are important questions to consider as they will influence variable selection, statistical choices, and ultimately the interpretation of the results.

4.4 Blueprint

Efforts to craft a research question are a very important aspect of developing purposive, inquisitive, and informed research (returning to Cross's characteristics of research). Moving beyond the research question in the project means developing and laying out the research design in a way such that the research is **Methodical** and **Communicable**. In this textbook, the method to achieve these goals is through the development of a research blueprint. The blueprint includes two components: (1) laying out a conceptual plan and (2) deriving the organizational scaffolding that will support the implementation of the research.

As Ignatow and Mihalcea (2017) point out:

Research design is essentially concerned with the basic architecture of research projects, with designing projects as systems that allow theory, data, and research methods to interface in such a way as to maximize a project's ability to achieve its goals [...]. Research design involves a sequence of decisions that have to be taken in a project's early stages, when one oversight or poor decision can lead to results that are ultimately trivial or untrustworthy. Thus, it is critically important to

think carefully and systematically about research design before committing time and resources to acquiring texts or mastering software packages or programming languages for your text mining project.

In what follows, I will cover the main aspects of developing a research blueprint. I will start with the conceptual plan and then move on to the organizational scaffolding.

4.4.1 Plan

Importance of establishing a feasible research design from the outset and documenting the key aspects required to conduct the research cannot be understated. On the one hand this process links a conceptual plan to a tangible implementation. In doing so, a researcher is better-positioned to conduct research with a clear view of what will be entailed. On the other hand, a promising research question may present unexpected challenges once a researcher sets about to implement the research. This is not uncommon to encounter issues that require modification or reevaluation of the viability of the project. However, a well-documented research plan will help a researcher to identify and address many of these challenges at the conceptual level before expending effort on the implementation.

Let's now consider the main aspects of developing a research plan: identifying data source(s), key variables, and analysis methods. Before we do, however, it is important to reiterate the importance of a research question or hypothesis before moving forward in research planning. The research question or hypothesis is the central component of the research plan. It guides every step which follows from data selection to interpretation of the analysis results. Furthermore, a well-founded research question is based on a solid literature review from which can provide helpful guidance at key choice points in the research process.

First, **identify a viable data source**. Viability includes the accessibility of the data, availability of the data, and the content of the data. If a purported data source is not accessible and/ or is has stringent restrictions on its use, then it is not a viable data source. If a data source is accessible and available, but does not contain the building blocks needed to address the research question, then it is not a viable data source. In the case that research is inferential in nature, the sampling frame of the corpus is of primary importance as the goal is to generalize the findings to a target population. A corpus resource should align, to the extent feasible, with this target population. For predictive and exploratory research, the goal to generalize a claim is not central and for this reason the there is some freedom in terms of how representative a corpus sample is of a target population. Ideally, a researcher will find and be able to model a language population of target interest. Since the goal, however, is not to test a hypothesis, but rather to explore particular or evaluate potential relationships, either in an exploratory or predictive fashion, the research can often continue with the stipulation that the results are interpreted in the light of the characteristics of the available corpus sample.

The second step is to **identify the key variables** need to conduct the research are and then ensure that this information can be derived from the corpus data. The research question will

reference the unit of analysis and the unit of observation, but it is important at this point to then pinpoint what the key variables will be. If the unit of observation is spontaneous conversations. The question as to what aspects of these conversations will be used in the analysis. In the research questions presented in this chapter, we will want to envision what needs to be done to derive a variable which measures the number of questions in each of the conversations. In other research, their may be features that need to be extracted, recoded, and/ or generated to address the research question. Other variables of importance may be non-linguistic in nature. In cases where there the metadata is incomplete for the goals of the research, it is sometimes possible to merge metadata from other sources.

The third step is to **identify a method of analysis** to interrogate the dataset. The selection of the analysis approach that was part of the research aim (*i.e.* explore, predict, or infer) and then the research question goes a long way to narrowing the methods that a researcher must consider. But there are a number of factors which will make some methods more appropriate than others.

Exploratory research is the least restricted of the three types of analysis approaches. Although it may be the case that a research will not be able to specify from the outset of a project what the exact analysis methods will be, an attempt to consider what types of analysis methods will be most promising to provide results to address the research question goes a long way to steering a project in the right direction and grounding the research. As with the other analysis approaches, it is important to be aware of what the analysis methods available and what type of information they produce in light of the research question.

For predictive-based research, the informational value of the target variable is key to deciding whether the prediction will be a classification task or a regression task. This has downstream effects when it comes time to evaluate and interpret the results. Although the feature engineering process in predictive analyses means that the features do not need to be specified from the outset and can be tweaked and changed as needed during an analysis, it is a good idea to start with a basic sense of what features most likely will be helpful in developing a robust predictive model. Furthermore, while the number and informational values of the features (predictor variables) are not as important to selecting a prediction method (algorithm) as they are in inferential analysis methods, it is important to recognize that particular algorithms have strengths and shortcomings when working large numbers and/ or types of features (Lantz 2013).

In inferential research, the number and information values of the variables to be analyzed will be of key importance (S. Th. Gries 2013). The informational value of the dependent variable will again narrow the search for the appropriate method. The number of predictor variables also plays an important role. For example, a study with a categorical dependent variable with a single categorical predictor variable will lead the researcher to the Chi-squared test. A study with a numeric dependent variable with multiple predictor variables will lead to linear regression. Another aspect of note for inference studies is the consideration of the distribution of numeric variables –a normal distribution will use a parametric test where a non-normal distribution will use a non-parametric test. These details need not be nailed down at this

point, but it is helpful to have them on your radar to ensure that when the time comes to analyze the data, the appropriate steps are taken to test for normality and then apply the correct test.

The last of the main components of the research plan concerns the **interpretation and evaluation of the results**. This step brings the research plan full circle connecting the research question to the methods employed. It is important to establish from the outset what the criteria will be to evaluate the results. This is in large part a function of the relationship between the research question and the analysis method. For example, in exploratory research, the results will be evaluated qualitatively in terms of the associative patterns that emerge. Predictive and inferential research leans more heavily on quantitative metrics in particular the accuracy of the prediction or the strength of the relationship between the dependent and predictor variable(s), respectively. However, these quantitative metrics require qualitative interpretation to determine whether the results are meaningful in light of the research question.

To summarize these planning steps, I've created a checklist in Table 4.5.

Table 4.5: Research Plan Checklist

Steps	Description
Research Question or Hypothesis	Formulate a research question or hypothesis based on a thorough review of existing literature including references. This will guide every subsequent step from data selection to interpretation of results.
Data Source(s)	Identify viable data source(s) and vet the sample data in light of the research question. Consider to what extent the goal is to generalize findings to a target population, and ensure that the corpus aligns as much as feasible with this target.
Key Variables	Determine the key variables needed for the research, define how they will be operationalized, and ensure they can be derived from the corpus data. Additionally, identify any features that need to be extracted, recoded or generated.
Analysis Method	Choose an appropriate method of analysis to interrogate the dataset. This choice should be in line with your research aim (<i>e.g.</i> , exploratory, predictive, inferential). Be aware of what each method can offer and how it addresses your research question.
Interpretation & Evaluation	Establish criteria to interpret and evaluate the results. This will be a function of the relationship between the research question and the analysis method.

In addition to addressing the steps outlined in Table 4.5, it is also important to document the strengths and shortcomings of the research plan including the data source(s), the information to be extracted from the data, and the analysis methods. If there are potential shortcomings,

which there most often are, sketch out contingency plans to address these shortcomings. This will help buttress your research and ensure that your time and effort is well-spent.

Together the information collected from this process will serve to guide the research and provide a solid foundation for the research write-up. Furthermore, you may consider pre-registering your research project to ensure that your plans are well-documented and to provide a timestamp for your research. Pre-registration can also be a helpful way to get feedback on your research plan from colleagues and experts in the field. Popular pre-registration platforms include Open Science Framework²³ and Center for Open Science²⁴.

4.4.2 Scaffold

The next step in creating a research blueprint is to consider how to physically implement your project. This includes how to organize files and directories in a fashion that both provides the researcher a logical and predictable structure to work with but also ensures that the research is **Communicable**. On the one hand, communicable research includes a strong write-up of the research, but, on the other hand, it is also important that the research is reproducible. Reproducibility strategies are a benefit to the researcher (in the moment and in the future) as it leads to better work habits and to better teamwork and it makes changes to the project easier. Reproducibility is also of benefit to the scientific community as shared reproducible research enhances replicability and encourages cumulative knowledge development (Gandrud 2015).

There are a set of guiding principles to accomplish these goals (Gentleman and Temple Lang 2007; Marwick, Boettiger, and Mullen 2018), seen in Example 4.4.

Example 4.4. Reproducible Research Principles

1. All files should be plain text which means they contain no formatting information other than whitespace.
2. There should be a clear separation between the data, method, and output of research. This should be apparent from the directory structure.
3. A separation between original, derived, and analysis data should be made. Original data should be treated as ‘read-only’. Any changes to the original data should be justified, generated by the code, and documented (see point 6).
4. Each project file (script) should represent a particular, well-defined step in the research process.
5. Each project script should be modular –that is, each file should correspond to a specific goal in the analysis procedure with input and output only corresponding to this step.
6. All project scripts should be tied together by a ‘main’ script that is used to coordinate the execution of all the project steps.

²³<https://osf.io/>

²⁴<https://www.cos.io/initiatives/prereg>

7. Everything should be documented. This includes data collection, data preprocessing, analysis steps, script code comments, data description in data dictionaries, information about the computing environment and packages used to conduct the analysis, and detailed instructions on how to reproduce the research.

These seven principles can be physically implemented in numerous ways. In recent years, there has been a growing number of efforts to create R packages and templates to quickly generate the scaffolding and tools to facilitate reproducible research. Some notable R packages include workflowr²⁵ (Blischak, Carbonetto, and Stephens 2023), ProjectTemplate²⁶ (White 2023), and targets²⁷ (Landau 2023), but there are many other resources for R included on the CRAN Task View for Reproducible Research²⁸. There are many advantages to working with pre-existing frameworks for the savvy R programmer including the ability to quickly generate a project scaffold, to efficiently manage changes to the project, and to buy in to a common framework that is supported by a community of developers.

On the other hand, these frameworks can be a bit daunting for the novice R programmer. At the most basic level, a project can implement the seven principles outlined above by creating a directory structure and a set of files manually.

Example 4.5. Minimal Project Framework

```
project/
├── data/
│   ├── analysis/
│   ├── derived/
│   └── original/
├── output/
│   ├── figures/
│   ├── reports/
│   ├── results/
│   └── tables/
├── code/
│   └── ...
└── _main.R
└── README
```

In Example 4.5, I provide a minimal framework that aligns with the reproducible research principles. Let me now make the connections between the principles and this project structure.

²⁵<https://jdblischak.github.io/workflowr/>

²⁶<http://projecttemplate.net/>

²⁷<https://github.com/ropensci/targets>

²⁸<https://cran.r-project.org/web/views/ReproducibleResearch.html>

The *project/* directory is composed of three main sections: *data/*, *output/*, and *code/* corresponding to the input, output, and code, respectively.

The *data/* section is divided into three subsections:

- *analysis/* for storing data used to perform analysis
- *derived/* for housing data produced in curation and transformation steps
- *original/* for keeping the original ‘read-only’ data

The *output/* section contains four subsections:

- *figures/* for visualizations produced as part of the project
- *reports/* for the resulting reports (e.g. article, presentation, blog post, etc.)
- *results/* for statistical results from the analysis
- *tables/* for summary tables

The *code/* directory houses the code, with the *_main.R* file at the root of the project orchestrating the execution of the scripts that conduct the project steps (*1-acquire-data.qmd*, *2-curate-dataset.qmd*, *3-transform-datasets.qmd*, and *4-analyze-datasets.qmd*).

Lastly, the *README* file provides a description of the project and instructions on how to reproduce the research.

The project structure in Example 4.5 meets the minimal structural requirements for reproducible research but can be augmented in more sophisticated ways to support more functionality, as we will see in Chapter 12. One enhancement that I highly recommend is the use of literate programming, in the form of Quarto documents, to serve as the main project scripts. This facilitates the combination of executable code and prose documentation for each project step in a single, modular file.

Summary

The aim of this chapter is to provide the key conceptual and practical points to guide the development of a viable research project. Good research is purposive, inquisitive, informed, methodological, and communicable. It is not, however, always a linear process. Exploring your area(s) of interest and connecting with existing work will help couch and refine your research. But practical considerations, such as the existence of viable data, technical skills, and/or time constraints, sometimes pose challenges and require a researcher to rethink and/or redirect the research in sometimes small and other times more significant ways. The process of formulating a research question and developing a viable research plan is key to supporting viable, successful, and insightful research. To ensure that the effort to derive insight from data is of most value to the researcher and the research community, the research should strive to be methodological and communicable adopting best practices for reproducible research.

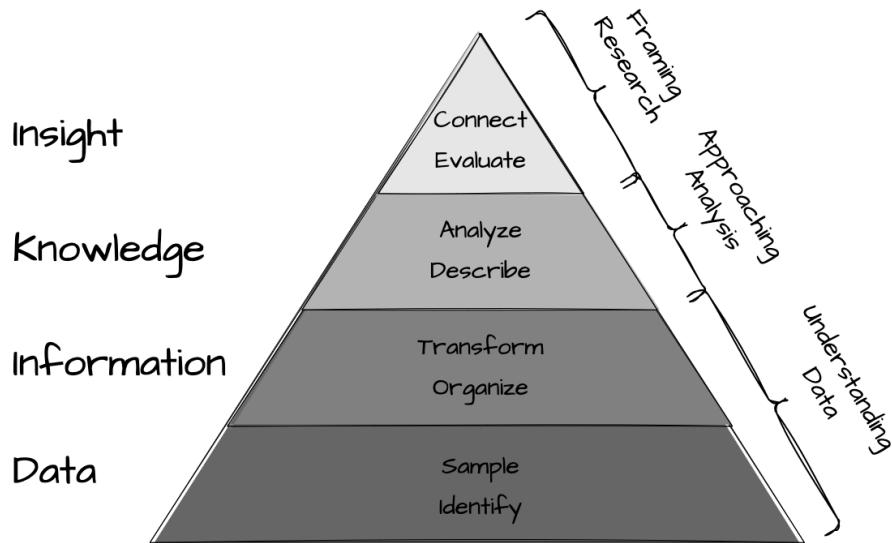


Figure 4.1: Framing research: visual summary

This chapter concludes the Foundations part of this textbook. At this stage our overview of fundamental characteristics of research are in place to move a project towards implementation, as seen in Figure 4.1. From this point forward we will integrate your conceptual knowledge and emerging R programming skills as we cover common scenarios encountered when conducting reproducible research with real-world data.

The next part, Preparation, aims to cover R coding strategies to acquire, curate, and transform data in preparation for analysis. These are the first steps in putting a research blueprint into action and by no coincidence the first components in the Data to Insight Hierarchy. Without further ado, let's get started!

Activities

-  Add description of outcomes ...

Recipe

What: Project management^a

How: Read Recipe 4 and participate in the Hypothes.is online social annotation.

Why:  To learn how to use ... reproducible research projects.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-4.html>

LAB

What: Project management^a

How: Clone, fork, and complete the steps in Lab 4.

Why: ↗ To ... a reproducible research project.

^a<https://github.com/qtalr/lab-4>

Questions

Conceptual questions

- What are the key characteristics of good research as described in this chapter?
- What are some strategies researchers can use to identify potential research areas and problems to investigate?
- For each strategy, describe how it contributes to research that is purposive, inquisitive, and informed.
- Why is framing a clear, focused research question or hypothesis important? Briefly explain how the research question guides the overall research process.
- What is the difference between the unit of analysis and the unit of observation? How do these concepts relate to the research question?
- What does it mean to operationalize a variable? Why is this important?
- The process of developing a research blueprint involves both conceptual planning and practical implementation steps. Explain how going through this process not only aids the individual researcher, but also the research community.
- Describe the main aspects of developing a research plan.
- Explain why it is important for research to be methodological and reproducible. What are some challenges researchers may face in achieving this?

↗ Technical exercises

- Matching research questions with data sources
- Matching research questions with research plans
- Preregistering a research project (?)
- Propose a quantitative research topic (or question if possible). Support your topic with supporting literature. (?)
- ...

Part III

Preparation

At this point we begin our journey to implement the research blueprint. As such, the content will be more focused on the practical steps to bring a plan to fruition integrating our conceptual understanding of the research process from the previous chapters with our emerging programming skills developed in lessons, recipes, and labs.

This part, Preparation, will address data acquisition, curation, and transformation steps. The goal of data preparation is to create a dataset which is ready for analysis. In each of these three upcoming chapters, I will outline some of the main characteristics to consider in each of these research steps and provide authentic examples of working with R to implement these steps. In Chapter 5 this includes the most common strategies for acquiring data: downloads and APIs. In Chapter 6 we turn to organize data into rectangular, or ‘tidy’, format. Depending on the data or dataset acquired for the research project, the steps necessary to shape our data into a base dataset will vary, as we will see. In Chapter 7 we will work to manipulate curated datasets to create datasets which are aligned with the research aim and research question. This often includes normalizing values, recoding variables, and generating new variables as well as and sourcing and merging information from other datasets with the dataset to be submitted for analysis.

Each of these chapters will cover the necessary documentation to trace our steps and provide a record of the data preparation process. Documentation serves to inform the analysis and interpretation of the results and also forms the cornerstone of reproducible research.

5 Acquire data



Draft

Ready for review.

The scariest moment is always just before you start.

– Stephen King



Outcomes

- Identify common strategies for acquiring corpus data.
- Describe how to organize and document data acquisition to support reproducibility.
- Recall R programming concepts and strategies relevant to acquiring data.

As we start down the path to executing our research blueprint, our first step is to acquire the primary data that will be employed in the project. This chapter covers three widely-used strategies for acquiring corpus data: downloads and APIs (Application Programming Interfaces). We will encounter various file formats and folder structures in the process and we will address how to effectively organize our data for subsequent processing. Crucial to our efforts is the process of documenting our data. We will learn to provide data origin information to ensure key characteristics of the data and its source are documented.

Along the way we will explore R coding concepts including control statements and custom functions relevant to the task of acquiring data. By the end of this chapter, you will not only be adept at acquiring data from diverse sources but also capable of documenting it comprehensively, enabling you to replicate the process in the future.



Swirl lesson

What: Control Statements, Custom Functions, Vectorization^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: To recognize the logic behind code that can make dynamic choices and to recall how functions serve to produce efficient, reusable, and more legible code.

^a<https://github.com/qtalr/lessons>

5.1 Downloads

The most common and straightforward method for acquiring corpus data is through direct downloads. In a nutshell, this method involves navigating to a website, locating the data, and downloading it to your computing environment. In some cases access to the data requires manual intervention and in others the process can be implemented programmatically. The data may be contained in a single file or multiple files. The files may be compressed or uncompressed. The data may be hierarchically organized or not. Each resource will have its own unique characteristics that will influence the process of acquiring the data. In this section we will work through a few examples to demonstrate the general process of acquiring data through downloads.

5.1.1 Manual

In contrast to the other data acquisition methods we will cover in this chapter, **manual downloads** require human intervention. This means that manual downloads are non-reproducible in a strict sense and require that we keep track of and document our procedure. It is a very common for research projects to acquire data through manual downloads as many data resources require some legwork before they are accessible for downloading. These can be resources that require institutional or private licensing and fees (Language Data Consortium¹, International Corpus of English², BYU Corpora³, *etc.*), require authorization/ registration (The Language Archive⁴, COW Corpora⁵, *etc.*), and/ or are only accessible via resource search interfaces (Corpus of Spanish in Southern Arizona⁶, Corpus Escrito del Español como L2 (CEDEL2)⁷, *etc.*).

Let's take a look at how to acquire data from a resource that requires manual intervention. The resource we will use is the Corpus Escrito del Español como L2 (CEDEL2)⁸ (Lozano 2009), a corpus of Spanish learner writing. It includes L2 writing from students with a variety of L1 backgrounds. For comparative purposes it also includes native writing for Spanish, English, and several other languages.

The CEDEL2 corpus is a freely available resource, but to access the data you must first use a search interface to select the relevant characteristics of the data of interest. Following the search/ download link you can find a search interface that allows the user to select the subcorpus and filter the results by a set of attributes, seen in Figure 5.1.

¹<https://www.ldc.upenn.edu/>

²<http://ice-corpora.net/ice/>

³<https://www.corpusdata.org/>

⁴<https://archive.mpi.nl/tla/>

⁵<https://www.webcorpora.org/>

⁶<https://cesa.arizona.edu/>

⁷<http://cedel2.learnercorpora.com/>

⁸<http://cedel2.learnercorpora.com/>

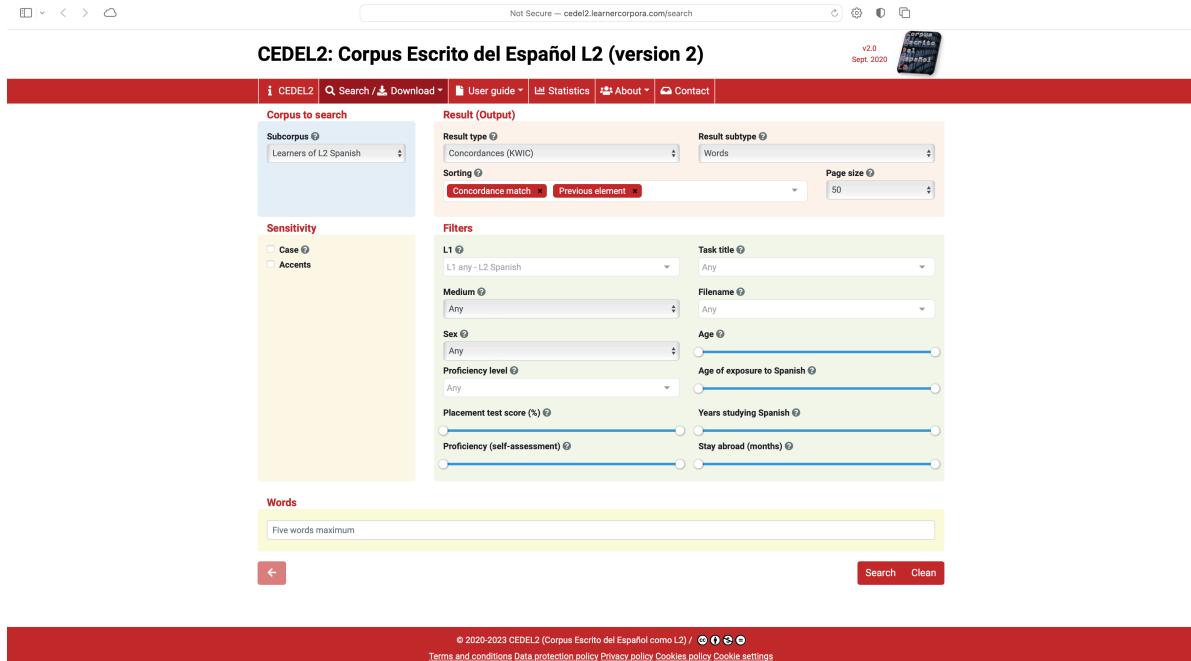


Figure 5.1: Search and download interface for the CEDEL2 Corpus

For this example let's assume that we want to acquire data to use in a study comparing the use of the Spanish preterite and imperfect past tense aspect in written texts by English L1 learners of Spanish to native Spanish speakers. To acquire data for such a project, we will first select the subcorpus "Learners of L2 Spanish". We will set the results to provide full texts and filter the results to "L1 English - L2 Spanish". Additionally, we will set the medium to "Written". This will provide us with a set of texts for the L2 learners that we can use for our study. The search parameters and results are shown in Figure 5.2.

The screenshot shows the CEDEL2 search interface. At the top, there are tabs for 'CEDEL2', 'Search / Download', 'User guide', 'Statistics', 'About', and 'Contact'. The 'Search / Download' tab is active. The main area is titled 'CEDEL2: Corpus Escrito del Español L2 (versión 2)'. On the left, there is a sidebar for 'Corpus to search' with a dropdown for 'Subcorpus' set to 'Learners of L2 Spanish'. Below this is a 'Sensitivity' section with checkboxes for 'Case' and 'Accents'. The main search area has 'Result (Output)' set to 'Texts', 'Sorting' set to 'L1', 'Placement test score (%)', 'Age', and 'Filename', with a 'Page size' of 50. The 'Filters' section includes dropdowns for 'L1' (set to 'L1 English - L2 Spanish'), 'Task title' (set to 'Any'), 'Medium' (set to 'Written'), 'Filename' (set to 'Any'), 'Sex' (set to 'Any'), 'Age' (set to 'Any'), 'Proficiency level' (set to 'Any'), 'Placement test score (%)' (set to 'Any'), 'Years studying Spanish' (set to 'Any'), 'Proficiency (self-assessment)' (set to 'Any'), and 'Stay abroad (months)' (set to 'Any'). Below the filters is a table of search results with 39 rows. The table columns are: Filename, L1, Age, Placement test score (%), Proficiency, Proficiency (self-assessment), Age of exposure to Spanish, Years studying Spanish, Stay abroad (months), Task title, and Medium. The first three rows are shown below:

Filename	L1	Age	Placement test score (%)	Proficiency	Proficiency (self-assessment)	Age of exposure to Spanish	Years studying Spanish	Stay abroad (months)	Task title	Medium
1 EN_WR_6_20_3_1_CJB	English	20	14	Lower beginner	2.25	17	3	0	1. Region where you live	Written
2 EN_WR_7_16_2_6_RM	English	16	16.3	Lower beginner	2.5	15	2	0	6. Recent trip	Written
3 EN_WR_7_26_2_2_TB	English	26	16.3	Lower beginner	2	18	2	0	2. Famous person	Written

Figure 5.2: Search results for the CEDEL2 Corpus

The ‘Download’ link now appears for this search criteria. Following this link will provide the user a form to fill out. This particular resource allows for access to different formats to download (Texts only, Texts with metadata, CSV (Excel), CSV (Others)). I will select the ‘CSV (Others)’ option so that the data is structured for easier processing downstream when we work to curate the data in our next processing step. Then I will choose to save the CSV in the *data/original/* directory of my project and create a sub-directory named *cedel2/*, as seen in Example 5.1.

Example 5.1. Download CEDEL2 L2 Spanish Learners data

```
data/
└── analysis/
└── derived/
└── original/
    └── cedel2/
        └── cedel2-l1-english-learners.csv
```

Note that the file is named *cedel2-l1-english-learners.csv* to reflect the search criteria used to acquire the data. In combination with other data documentation, this will help us to maintain transparency.

Now, after downloading the L2 learner and the native speaker data into the appropriate directory, we move on to the next processing step, right? Not so fast! Imagine we are working on a project with a collaborator. How will they know where the data came from? What if we need to come back to this data in the future? How will we know what characteristics we used to filter the data? The directory and filenames may not be enough. To address these questions we need to document the origin of the data, and in the case of data acquired through manual downloads, we need to document the procedures we took to acquire the data to the best of our ability.

As discussed in Section 2.3.1, all acquired data should be accompanied by a data origin file. The majority of this information can typically be identified on the resource’s website and/or the resource’s documentation. In the case of the CEDEL2 corpus, the corpus homepage provides most of the information we need.

Structurally, data documentation files should be stored close to the data they describe. So for our data origin file this means adding it to the `data/original/` directory. Naming the file in a transparent way is also important. I’ve named the file `cedel2_do.csv` to reflect the name of the corpus, the meaning of the file as data origin with `*_do`, and the file extension `.csv`* to reflect the file format. CSV files reflect tabular content. It is not required that data origin files are tabular, but it makes it easier to read and display them in literate programming documents.

Tip

There are many ways to create and edit CSV files. You can use a spreadsheet program like MS Excel or Google Sheets, a text editor like Notepad or TextEdit, or a code editor like RStudio or VS Code. The `qtairkit` package provides a convenient function, `create_data_origin()` to create a CSV file with the data origin boilerplate structure. This CSV file then can be edited to add the relevant information in any of the above mentioned programs.

Using a spreadsheet program is the easiest method for editing tabular data. The key is to save the file as a CSV file, and not as an Excel file, to maintain our adherence to the principle of using open formats for reproducible research.

In Table 5.1, I’ve created a data origin file for the CEDEL2 corpus.

Given this is a manual download we also need to document the procedure used to retrieve the data in prose. The script in the `code/` directory that is typically used to acquire the data is not used to programmatically retrieve data in this case. However, to keep things predictable we will use this file to document the download procedure. I’ve created a Quarto file named `1_acquire_data.qmd` in the `code/` directory of my project.

A glimpse at the directory structure of the project at this point is seen in Example 5.2.

Example 5.2. Project structure for the CEDEL2 corpus data acquisition

Table 5.1: Data origin file for the CEDEL2 corpus

attribute	description
Resource name	CEDEL2: Corpus Escrito del Español como L2.
Data source	http://cedel2.learnercorpora.com/ , https://doi.org/10.1177/02676583211050522
Data sampling frame	Corpus that contains samples of the language produced from learners of Spanish as
Data collection date(s)	2006-2020.
Data format	CSV file. Each row corresponds to a writing sample. Each column is an attribute of
Data schema	A CSV file for L2 learners and a CSV file for native speakers.
License	CC BY-NC-ND 3.0 ES
Attribution	Lozano, C. (2022). CEDEL2: Design, compilation and web interface of an online cor

```

project/
└── code/
    ├── 1_acquire_data.qmd
    └── ...
└── data/
    ├── analysis/
    ├── derived/
    └── original/
        ├── cedel2_do.csv
        └── cedel2/
            ├── cedel2-l1-english-learners.csv
            └── cedel2-native-spanish-speakers.csv
└── output/
    ├── figures/
    ├── reports/
    ├── results/
    └── tables/
└── README.md
└── _main.R

```

In the *1_acquire_data.qmd* file I've added example sections to display the data origin CSV file as a table and to document the data download procedures, as seen in File 5.1.

The output from *1_acquire_data.qmd* will contain a table displaying the data origin file and a prose section documenting the data acquisition process. This will provide a transparent record of the data acquisition process for future reference.

Manually downloading other resources will inevitably include unique processes for obtaining the data, but in the end the data should be archived in the research structure in the *data/o-*

riginal/ directory and documented in the appropriate places. The acquired data is treated as ‘read-only’, meaning it is not modified in any way. This gives us a transparent starting point for subsequent steps in the data preparation process.

5.1.2 Programmatic

There are many resources that provide corpus data that is directly accessible for which programmatic downloads can be applied. A **programmatic download** is a download in which the process can be automated through code. Thus, this is a reproducible process. The data can be acquired by anyone with access to the necessary code.

In this case, and subsequent data acquisition procedures in this chapter, we use the *1_acquire_data.qmd* Quarto file to its full potential intermingling prose, code, and code comments to execute and document the download procedure. In File 5.2, I’ve added example sections to display example boilerplate structure for a programmatic data acquisition and documentation.

To illustrate how this works to conduct a programmatic download, we will work with the Switchboard Dialog Act Corpus (SWDA) (University of Colorado Boulder 2008). The version that we will use is found on the Linguistic Data Consortium under the Switchboard-1 Release 2 Corpus⁹. The corpus and related documentation are linked on the catalog page <https://catalog.ldc.upenn.edu/docs/LDC97S62/>.

From the documentation we learn that the corpus contains transcripts for 1155 5-minute two-way telephone conversations among English speakers for all areas of the United States. The speakers were given a topic to discuss and the conversations were recorded. The corpus metadata and annotations for sociolinguistic and discourse features.

The SWDA was referred to in Section 3.2.3 to support our toy hypothesis that men and women differ in the frequency of the use of questions in spontaneous conversations. This corpus, as you can image, could support a wide range of interesting research questions. Let’s assume we are following research conducted by Tottie (2011) to explore the use of filled pauses such as “um” and “uh” and traditional sociolinguistic variables such as sex, age, and education in spontaneous speech by American English speakers.

With this goal in mind, let’s get started writing the code to download and organize the data in our project directory. First we need to identify the URL (Uniform Resource Locator) for the data that we want to download. More often than not this file will be some type of compressed archive file with an extension such as *.zip* (Zipped file), *.tar* (Tarball file), or *tar.gz* (Gzipped tarball file), which is the case for the SWDA corpus. Compressed files make downloading multiple files easy by grouping files and directories into one file.

⁹https://catalog.ldc.upenn.edu/docs/LDC97S62

💡 Consider this

You may be wondering what the difference between *.zip*, *.tar*, and *.tar.gz* files are. The *.zip* file format is the most common. It groups files and directories into one file (archives) and compresses them to reduce the size of the file in one step when the file is created. The *.tar* file format is used to archive files and folders, it does not perform compression. Gzipping performs the compression to the *.tar* file resulting in a file with the *.tar.gz* extension. Notably the *.gz* compression is highly efficient for large files. Take the *swda.tar.gz* file for example. It has a compressed file size of 4.6 MB, but when uncompressed it is 16.9 MB. This is a 73% reduction in file size.

In R we can use the `download.file()` function from base R. The `download.file()` function minimally requires two arguments: `url` and `destfile`. These correspond to the file to download and the location where it is to be saved to disk. To break out the process a bit, I will assign the URL and destination file path to variables and then use the `download.file()` function to download the file.

Example 5.3.

```
# URL to SWDA corpus compressed file
file_url <-
  "https://catalog.ldc.upenn.edu/docs/LDC97S62/swb1_dialogact_annot.tar.gz"

# Relative path to project/data/original directory
file_path <- "../data/original/swda.tar.gz"

# Download SWDA corpus compressed file
download.file(url = file_url, destfile = file_path)
```

⚠ Warning

Note that the `file_path` variable in Example 5.3 is a relative path to the *data/original/* directory. This is because the *1_acquire_data.qmd* file that we are using for this code is located in the *code/* directory and the *data/* directory is a sibling directory to the *code/* directory.

It is also possible to use an absolute path to the *data/original/* directory. I will have more to say about the advantages and disadvantages of relative and absolute paths in reproducible research in Chapter 12.

As we can see looking at the directory structure, in Example 5.4, the *swda.tar.zip* file has been added to the *data/original/* directory.

Example 5.4. Downloaded SWDA corpus compressed file

```
data/
└── analysis/
└── derived/
└── original/
    └── swda.tar.zip
```

Once a compressed file is downloaded, however, the file needs to be ‘decompressed’ to reveal the directory structure and files. To decompress this `.tar.gz` file we use the `untar()` function with the arguments `tarfile` pointing to the `.tar.gz` file and `exdir` specifying the directory where we want the files to be extracted to. Again, I will assign the arguments to variables. Then we can decompress the file using the `untar()` function.

Example 5.5.

```
# Relative path to the compressed file
tar_file <- "../data/original/swda.tar.gz"

# Relative path to the directory to extract to
extract_to_dir <- "../data/original/swda/"

# Decompress .zip file and extract to our target directory
untar(tar_file, extract_to_dir)
```

The directory structure of `data/` in Example 5.6 now shows the `swda.tar.gz` file and the `swda` directory that contains the decompressed directories and files.

Example 5.6.

```
data/
└── analysis/
└── derived/
└── original/
    └── swda/
        ├── README
        ├── doc/
        ├── sw00utt/
        ├── sw01utt/
        ├── sw02utt/
        ├── sw03utt/
        └── sw04utt/
```

```
|   └── sw05utt/
|   └── sw06utt/
|   └── sw07utt/
|   └── sw08utt/
|   └── sw09utt/
|   └── sw10utt/
|   └── sw11utt/
|   └── sw12utt/
|   └── sw13utt/
└── swda.tar.gz
```

At this point we have acquired the data programmatically and with this code as part of our workflow anyone could run this code and reproduce the same results. The code as it is, however, is not ideally efficient. Firstly the *swda.tar.gz* file is not strictly needed after we decompress it and it occupies disk space if we keep it. And second, each time we run this code the file will be downloaded from the remote server. This leads to unnecessary data transfer and server traffic and will overwrite the data if it already exists in our project directory which could be problematic if the data changes on the remote server. Let's tackle each of these issues in turn.

To avoid writing the *swda.tar.gz* file to disk (long-term) we can use the `tempfile()` function to open a temporary holding space for the file in the computing environment. This space can then be used to store the file, decompress it, and then the temporary file will automatically be deleted. We assign the temporary space to an R object we will name `temp_file` with the `tempfile()` function. This object can now be used as the value of the argument `destfile` in the `download.file()` function.

Example 5.7.

```
# URL to SWDA corpus compressed file
file_url <-
  "https://catalog.ldc.upenn.edu/docs/LDC97S62/swb1_dialogact_annot.tar.gz"

# Create a temporary file space for our .tar.gz file
temp_file <- tempfile()

# Download SWDA corpus compressed file
download.file(file_url, temp_file)
```

Tip

In Example 5.7, I've used the values stored in the objects `file_url` and `temp_file` in the `download.file()` function without specifying the argument names –only providing the names of the objects. R will assume that values of a function map to the ordering of the arguments. If your values do not map to ordering of the arguments you are required to specify the argument name and the value. To view the ordering of objects hit `tab` after entering the function name or consult the function documentation by prefixing the function name with `?` and hitting `enter`.

At this point our downloaded file is stored temporarily on disk and can be accessed and decompressed to our target directory using `temp_file` as the value for the argument `tarfile` from the `untar()` function. I've assigned our target directory path to `extract_to_dir` and used it as the value for the argument `exdir`.

Example 5.8.

```
# Assign our target directory to `extract_to_dir'  
extract_to_dir <- "../data/original/swda/"  
  
# Decompress .tar.gz file and extract to our target directory  
untar(tarfile = temp_file, exdir = target_dir)
```

Our directory structure in Example 5.8 is the same as in Example 5.6, minus the `swda.tar.gz` file.

The second issue I raised concerns the fact that running this code as part of our project will repeat the download each time. Since we would like to be good citizens and avoid unnecessary traffic on the web and avoid potential issues in overwriting data, it would be nice if our code checked to see if we already have the data on disk and if it exists, then skip the download, if not then download it.

The desired functionality we've described can be implemented using the `if()` function. The `if()` function is one of a class of functions known as control statements. **Control statements** allow us to control the flow of our code by evaluating logical statements and processing subsequent code based on the logical value it is passed as an argument.

So in this case we want to evaluate whether the data directory exists on disk. If it does then skip the download, if not, proceed with the download. In combination with `else` which provides the ‘if not’ part of the statement, we have the following logical flow in Example 5.9.

Example 5.9.

```
if (DIRECTORY_EXISTS) {  
  # Do nothing  
} else {  
  # Download data  
}
```

We can simplify this statement by using the `!` operator which negates the logical value of the statement it precedes. So if the directory exists, `!DIRECTORY_EXISTS` will return `FALSE` and if the directory does not exist, `!DIRECTORY_EXISTS` will return `TRUE`. In other words, if the directory does not exist, download the data. This is shown in Example 5.10.

Example 5.10.

```
if (!DIRECTORY_EXISTS) {  
  # Download data  
}
```

Now, to determine if a directory exists in our project directory we will turn to the `fs` package (Hester, Wickham, and Csárdi 2023). The `fs` package provides a set of functions for interacting with the file system, including `dir_exists()`. `dir_exists()` takes a path to a directory as an argument and returns the logical value, `TRUE`, if that directory exists, and `FALSE` if it does not.

We can use this function to evaluate whether the directory exists and then use the `if()` function to process the subsequent code based on the logical flow we set out in Example 5.10. Applied to our project, the code will look like Example 5.11.

Example 5.11.

```
# Load the `fs` package  
library(fs)  
  
# URL to SWDA corpus compressed file  
file_url <-  
  "https://catalog.ldc.upenn.edu/docs/LDC97S62/swb1_dialogact_annot.tar.gz"  
  
# Create a temporary file space for our .tar.gz file  
temp_file <- tempfile()  
  
# Assign our target directory to `extract_to_dir`
```

```

extract_to_dir <- "../data/original/swda/"

# Check if our target directory exists
# If it does not exist, download the file and extract it
if (!dir_exists(extract_to_dir)) {
  # Download SWDA corpus compressed file
  download.file(file_url, temp_file)

  # Decompress .tar.gz file and extract to our target directory
  untar(tarfile = temp_file, exdir = extract_to_dir)
}

```

The code in Example 5.11 is added to the *1_acquire_data.qmd* file we introduced in File 5.2. When this file is run, the SWDA corpus data will be downloaded and extracted to our project directory. If the data already exists, the download will be skipped, just as we wanted.

Before we move on, we need to make sure to create and add the appropriate information to the data origin file. To make this easier, the *qtalrkit* package includes a function, *create_data_origin()*, to create a data origin file template in CSV format. This function takes the path for the desired file. In the SWDA Corpus case, this might be something like: *../data/original/swda_do.csv*. The function only needs to be run once and does not need to be part of the reproducible workflow.

Running the code in Example 5.12 at the console will create the file. Open it in your preferred text or spreadsheet editor to add the appropriate information.

Example 5.12.

```

# Load the `qtalrkit` package
library(qtalrkit)

# Create a data origin file template
create_data_origin("../data/original/swda_do.csv")

```

Our complete project structure for the SWDA corpus data acquisition is shown in Example 5.13.

Example 5.13. Project structure for the SWDA corpus data acquisition

```

project/
  └── code/

```

```
|- 1_acquire_data.qmd
|- ...
+- data/
  +- analysis/
  +- derived/
  +- original/
    +- swda_do.csv
    +- swda/
      +- README
      +- doc/
      +- sw00utt/
      +- sw01utt/
      +- sw02utt/
      +- sw03utt/
      +- sw04utt/
      +- sw05utt/
      +- sw06utt/
      +- sw07utt/
      +- sw08utt/
      +- sw09utt/
      +- sw10utt/
      +- sw11utt/
      +- sw12utt/
      +- sw13utt/
+- output/
  +- figures/
  +- reports/
  +- results/
  +- tables/
+- README.md
+- _main.R
```

Great, we've successfully acquired and documented the SWDA Corpus data. We've leveraged R to automate the download and extraction of the data, depending on the existence of the data in our project directory. But you may be asking yourself, "Can't I just navigate to the corpus page and download the data manually myself?" The simple answer is, "Yes, you can." The more nuanced answer is, "Yes, but consider the trade-offs."

The following scenarios highlight the some advantages to automating the process. If you are acquiring data from multiple files, it can become tedious to document the manual process for each file such that it is reproducible. It's possible, but it's error prone. Now, if you are collaborating with others, you will want to share this data with them. It is very common

to find data that has limited restrictions for use in academic projects, but the most common limitation is redistribution. This means that you can use the data for your own research, but you cannot share it with others. If you plan on publishing your project to a repository, like GitHub, to share the data as part of your reproducible project, you would be violating the terms of use for the data. By including the programmatic download in your project, you can ensure that your collaborators can easily and effectively acquire the data themselves and that you are not violating the terms of use.

5.2 APIs

A convenient alternative method for acquiring data in R is through package interfaces to web services. These interfaces are built using R code to make connections with resources on the web through **Application Programming Interfaces** (APIs). Websites such as Project Gutenberg, Twitter, Facebook, and many others provide APIs to allow access to their data under certain conditions, some more limiting for data collection than others. Programmers (like you!) in the R community take up the task of wrapping calls to an API with R code to make accessing that data from R convenient, and of course reproducible.

Dive deeper

Many, many web services provide API access. These APIs span all kinds of data, from text to images to video to audio. Visit the Public APIs website^a to explore the diversity of APIs available.

ROpenSci maintains a curated list of R packages that provide access to data from web services. Visit the ROpenSci website^b to explore the packages available.

^a<https://publicapis.io/>

^b<https://ropensci.org/packages/data-access/>

In addition to popular public APIs, there are also APIs that provide access to repositories and databases which are of particular interest to linguists. For example, Wordbank¹⁰ provides access to a large collection of child language corpora through the `wordbankr` package (Braginsky 2022), and Glottolog¹¹, World Atlas of Language Structures¹² (WALS), and PHOIBLE¹³ provide access to large collections of language metadata that can be accessed through the `lingtypology` package (Moroz 2023).

Let's work with an R package that provides access to the TalkBank¹⁴ database. The TalkBank project  [CITATION] contains a large collection of spoken language corpora from

¹⁰<http://wordbank.stanford.edu/>

¹¹<https://glottolog.org/>

¹²<https://wals.info/>

¹³<https://phoible.org/>

¹⁴<https://talkbank.org/>

various contexts: conversation, child language, multilinguals, *etc.* Resource information, web interfaces, and links to download data in various formats can be found by perusing individual resources linked from the main page. However, the **TBDBr** package (Kowalski and Cavanaugh 2022) provides convenient access to data using R once a data resource is identified.

The CABNC (Albert, de Ruiter, and de Ruiter 2015) contains the demographically sampled portion¹⁵ of the spoken portion of the British National Corpus (BNC) (Leech 1992).

Useful for a study aiming to research spoken British English, either in isolation or in comparison to American English (SWDA).

First, we need to install and load the **TBDBr** package. The `install.package()` and `library()` functions work just great for this. An alternative is to use the `pacman` package (Rinker and Kurkiewicz 2019) to install and load the package in one step with the function `p_load()`, as shown in Example 5.14.

Example 5.14.

```
# Install and load the TBDBr package
pacman::p_load(TBDBr)
```

The **TBDBr** package provides a set of common `get*()` functions for acquiring data from the TalkBank corpus resources. These include:

- `getParticipants()`
- `getTranscripts()`
- `getTokens()`
- `getTokenTypes()`
- `getUtterances()`

For each of these function the first argument is `corpusName`, which is the name of the corpus resource as it appears in the TalkBank database. The second argument is `corpora`, which takes a character vector describing the path to the data. For the CABNC, these arguments are "ca" and `c("ca", "CABNC")` respectively. To determine these values, **TBDBr** provides the `getLegalValues()` interactive function which allows you to interactively select the repository name, corpus name, and transcript name (if necessary).

Another important aspect of these function is that they return data frame objects. Since we are accessing data that is in a structured database, this makes sense. However, we should always check the documentation for the object type that is returned by function to be aware of how to work with the data.

Let's start by retrieving the utterance data for the CABNC and preview the data frame it returns using `glimpse()`.

¹⁵http://www.natcorp.ox.ac.uk/docs/URG/BNCdes.html#body.1_div.1_div.5_div.1

Example 5.15.

```
# Set corpus_name and corpus_path
corpus_name <- "ca"
corpus_path <- c("ca", "CABNC")

# Get utterance data
utterances <-
  getUtterances(
    corpusName = corpus_name,
    corpora = corpus_path)

> [1] "Fetching data, please wait..."
> [1] "Success!"

# Preview the data
glimpse(utterances)

> Rows: 235,901
> Columns: 10
> $ filename <list> "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", ~
> $ path      <list> "ca/CABNC/KB0/KB0RE000", "ca/CABNC/KB0/KB0RE000", "ca/CABNC~
> $ utt_num   <list> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1~
> $ who       <list> "PS002", "PS006", "PS002", "PS006", "PS002", "PS006", "PS00~
> $ role      <list> "Unidentified", "Unidentified", "Unidentified", "Unidentifi~
> $ postcodes <list> <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NU~
> $ gems      <list> <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NU~
> $ utterance <list> "You enjoyed yourself in America", "Eh", "did you", "Oh I c~
> $ startTime <list> "0.208", "2.656", "2.896", "3.328", "5.088", "6.208", "8.32~
> $ endTime   <list> "2.672", "2.896", "3.328", "5.264", "6.016", "8.496", "9.31~
```

Inspecting the output from Example 5.15, we see that the data frame contains 235901 observations and 10 variables.

The summary provided by `glimpse()` also provides other useful information. First, we see the data type of each variable. Interestingly, the data type for each variable in the data frame is `list`. Being that a list is two-dimensional data type, like a data frame, we have list-type data in each value. This is known as a **nested structure**. We will see, and create, nested structures later, but for now it will suffice to say that we would like to ‘unnest’ these lists and reveal the list-contained vector types at the data frame level.

To do this we will pass the `utterances` data frame to the `unnest()` function from the `tidyverse` package (Wickham, Vaughan, and Girlich 2023). `unnest()` takes a data frame and a vector of variable names to unnest, `cols = c()`. To unnest all variables, we will use the `everything()` function from `dplyr` (Wickham, François, et al. 2023) to select all variables. We will use the result to overwrite the `utterances` object with the unnested data frame.

Example 5.16.

```
# Unnest the data frame
utterances <-
  utterances |>
  unnest(cols = everything())

# Preview the data
glimpse(utterances)
```

```
> Rows: 235,901
> Columns: 10
> $ filename <chr> "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", ~
> $ path      <chr> "ca/CABNC/KB0/KB0RE000", "ca/CABNC/KB0/KB0RE000", "ca/CABNC/~
> $ utt_num   <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17~
> $ who       <chr> "PS002", "PS006", "PS002", "PS006", "PS002", "PS006", "PS002~
> $ role      <chr> "Unidentified", "Unidentified", "Unidentified", "Unidentifie~
> $ postcodes <lgl> NA, ~
> $ gems       <lgl> NA, ~
> $ utterance  <chr> "You enjoyed yourself in America", "Eh", "did you", "Oh I co~
> $ startTime <chr> "0.208", "2.656", "2.896", "3.328", "5.088", "6.208", "8.32"~
> $ endTime   <chr> "2.672", "2.896", "3.328", "5.264", "6.016", "8.496", "9.312~
```

The output from Example 5.16 shows that the variables are now one-dimensional vector types.

Returning to the information about our data frame from `glimpse()`, the second thing to notice is we get a short preview of the values for each variable. There are a couple things we can gleen from this. One is that we can confirm or clarify the meaning of the variable names by looking at the values. The other thing to consider is whether the values show any patterns that may be worthy of more scrutiny. For example, various variables appear to contain the same values for each observation. For a variable like `filename`, this is expected as the first values likely correspond to the same file. However, for the variables `postcodes` and `gems` the values are 'NA'. This suggests that these variables may not contain any useful information and we may want to remove them later.

For now, however, we want to acquire and store the data in its original form (or as closely as possible). So now, we have acquired the utterances data and have it in our R session as a data frame. To store this data in a file, we will first need to consider the file format. Data frames are tabular, so that gives us a few options. Since we are working in R, we could store this data as an R object, in the form of an RDS file. An RDS file is a binary file that can be read back into R as an R object. This is a good option if we want to store the data for use in R, but not if we want to share the data with others or use it in other software. Another option is to store the data as a spreadsheet file, such as XSLX (MS Excel). This may make viewing and editing the contents more convenient, but it depends on the software available to you and others. A third, more viable option, is to store the data as a CSV file. CSV files are plain text files that can be read and written by most software. This makes CSV files one of the most popular for sharing tabular data. For this reason, we will store the data as a CSV file.

The `readr` package (Wickham, Hester, and Bryan 2023) provides the `write_csv()` function for writing data frames to CSV files. The first argument is the data frame to write, and the second argument is the path to the file to write. Note, however, that the directories in the path we specify need to exist. If they do not, we will get an error. In this case, I would like to write the file `utterances.csv` to the `../data/original/cabnc/` directory. The original project structure does not contain a `cabnc/` directory, so I need to create one. To do this, I will use `dir_create()` from the `fs` package (Hester, Wickham, and Csárdi 2023).

Example 5.17.

```
# Create the target directory
dir_create("../data/original/cabnc/")

# Write the data frame to a CSV file
write_csv(utterances, "../data/original/cabnc/utterances.csv")
```

Chaining the steps covered in Examples 5.15, 5.16, and 5.17, we have a succinct and legible code to acquire, adjust, and write utterances from the CABNC in Example 5.18.

Example 5.18.

```
# Set corpus_name and corpus_path
corpus_name <- "ca"
corpus_path <- c("ca", "CABNC")

# Create the target directory
dir_create("../data/original/cabnc/")

# Get utterance data
```

```

getUtterances(
  corpusName = corpus_name,
  corpora = corpus_path
) |>
  unnest(cols = everything()) |>
  write_csv("../data/original/cabnc/utterances.csv")

```

If our goal is just to acquire utterances, then we are done acquiring data and we move on to the next step. However, if we want to acquire other datasets from the CABNC, say participants, tokens, *etc.*, then we can either repeat the steps in Example 5.18 for each data type, or we can write a function to do this for us. A function serves us to make our code more legible and reusable for the CABNC, and since the TalkBank data is structured similarly across corpora, we can also use the function to acquire data from other corpora, if need be.

To write a function, we need to consider the following:

1. What is the name of the function?
2. What arguments does the function take?
3. What functionality does the function provide?
4. Does the function have optional arguments?
5. How does the function return the results?

Taking each in turn, the name of the function should be descriptive of what the function does. In this case, we are acquiring and writing data from Talkbank corpora. A possible name is `get_talkbank_data()`. The required arguments of the the `get*()` functions will definitely figure in our function. In addition, we will need to specify the path to the directory to write the data.

Example 5.19.

```

get_talkbank_data <- function(corpus_name, corpus_path, target_dir) {
  # ...
}

```

The next thing to consider is what functionality the function provides. In this case, we want to acquire and write data from Talkbank corpora. We can start by leveraging the code steps in Example 5.18, making some adjustments to the code replacing the hard-coded values with the function arguments and adding code to create the target file name based on the `target_dir` argument.

Example 5.20.

```

get_talkbank_data <- function(corpus_name, corpus_path, target_dir) {

  # Create the target directory
  dir_create(target_dir)

  # Set up file path name
  utterances_file <- path(target_dir, "utterances.csv")

  # Acquire data and write to file
  getUtterances(corpusName = corpus_name, corpora = corpus_path) |>
    unnest(cols = everything()) |>
    write_csv(utterances_file)
}

```

Before we address the obvious feature missing, which is the fact that this function in Example 5.20 only acquires and writes data for utterances, let's consider some functionality which would make this function more user-friendly.

What if the data is already acquired? Do we want to overwrite it, or should the function skip the process for files that already exist? By skipping the process, we can save time and computing resources. If the files are periodically updated, then we might want to overwrite existing files. To achieve this functionality we will use an `if()` statement to check if the file exists. If it does, then we will skip the process. If it does not, then we will acquire and write the data.

Example 5.21.

```

get_talkbank_data <- function(corpus_name, corpus_path, target_dir) {

  # Create the target directory
  dir_create(target_dir)

  # Set up file path name
  utterances_file <- path(target_dir, "utterances.csv")

  # If the file does not exist, then...
  # Acquire data and write to file
  if(!file_exists(utterances_file)) {
    getUtterances(corpusName = corpus_name, corpora = corpus_path) |>
      unnest(cols = everything()) |>
      write_csv(utterances_file)
}

```

```
  }  
}
```

We can also add functionality to Example 5.21 to force overwrite existing files, if need be. To do this, we will add an optional argument to the function, `force`, which will be a logical value. We will set the default to `force = FALSE` to preserve the existing functionality. If `force = TRUE`, then we will overwrite existing files. Then we add another condition to the `if()` statement to check if `force = TRUE`. If it is, then we will overwrite existing files.

Example 5.22.

```
get_talkbank_data <- function(corpus_name, corpus_path, target_dir, force =  
  FALSE) {  
  
  # Create the target directory  
  dir_create(target_dir)  
  
  # Set up file path name  
  utterances_file <- path(target_dir, "utterances.csv")  
  
  # If the file does not exist, then...  
  # Acquire data and write to file  
  if(!file_exists(utterances_file) | force) {  
    getUtterances(corpusName = corpus_name, corpora = corpus_path) |>  
    unnest(cols = everything()) |>  
    write_csv(utterances_file)  
  }  
}
```

From this point, we add the functionality to acquire and write the other data available from Talkbank corpora, such as participants, tokens, *etc.* This involves adding additional file path names and `if()` statements to check if the files exist surrounding the processing steps to Example 5.22. It may be helpful to perform other input checks, print messages, *etc.* for functions that we plan to share with others. I will leave these enhancements as an exercise for the reader.

Before we leave the topic of functions, let's consider where to put functions after we write them. Here are a few options:

1. In the same script as the code that uses the function.
2. In a separate script, such as *functions.R*.

3. In a package, which is loaded by the script that uses the function.

The general heuristic for choosing where to put functions is to put them in the same script as the code that uses them if the function is only used in that script. If the function is used in multiple scripts or the function or number of functions clutters the readability of the code, then put it in a separate script. If the function is used in multiple projects, then put it in an R package.

Dive deeper

If you are interested in learning more about writing functions, check out the Writing Functions chapter^a in the R for Data Science^b book.

If you find yourself writing functions that are useful for multiple projects, you may want to consider creating an R package. R packages are a great way to share your code with others. If you are interested in learning more about creating R packages, check out the R Packages book^c by Hadley Wickham and Jenny Bryan.

^a<https://r4ds.had.co.nz/functions.html>

^b<https://r4ds.had.co.nz/>

^c<https://r-pkgs.org/>

In this case, we will put the function in a separate file, *functions.R*, in the same directory as the other code files as in Example 5.25.

Example 5.23.

```
code/
  └── 1_acquire_data.qmd
  └── ...
  └── functions.R
```

Tip

Note that that the *functions.R* file is an R script, not a Quarto document. Therefore code blocks that are used in *.qmd* files are not used, only the R code and code comments.

To include this, or other functions in in the R session of the code file that uses them, use the `source()` function, as seen in Example 5.24.

Example 5.24.

```
# Source functions
source("functions.R")
```

Given the utility of this function to my projects and potentially others', I've included the `get_talkbank_data()` function in the `qtalrkit` package. You can view the source code by calling the function without parentheses `()`, or on the `qtalrkit` GitHub repository.

After running the `get_talkbank_data()` function, we can see that the data has been acquired and written to the `data/original/cabnc/` directory.

Example 5.25.

```
data/
└── analysis
└── derived
└── original
    └── cabnc
        ├── participants.csv
        ├── token_types.csv
        ├── tokens.csv
        ├── transcripts.csv
        └── utterances.csv
```

Add comments to your code in `1-acquire-data.Rmd` and create and complete the data origin documentation file for this resource, and the acquisition is complete.

Summary

In this chapter we have covered a lot of ground. On the surface we have discussed a few methods for acquiring corpus data for use in text analysis. In the process we have delved into various aspects of the R programming language. Some key concepts include writing custom functions, control statements, and applying functions iteratively. We have also considered topics that are more general in nature and concern interacting with data found on the internet.

Each of these methods should be approached in a way that is transparent to the researcher and to would-be collaborators and the general research community. For this reason the documentation of the steps taken to acquire data are key both in the code and in human-facing documentation.

At this point you have both a bird's eye view of the data available on the web and strategies on how to access a great majority of it. It is now time to turn to the next step in our data analysis project: data curation. In the next chapter, I will cover how to wrangle your raw data into a tidy dataset.

Activities

-  Add description of outcomes

Recipe

-  update

What: Collecting and documenting data^a

How: Read Recipe 5 and participate in the Hypothes.is online social annotation.

Why: To understand common strategies to collect, organize, and document data using effective, concise, and reproducible code. You will see how control statements, writing custom functions, and leveraging iteration works to these goals. These programming strategies are often useful for acquiring data but, as we will see, they are powerful concepts that can be used throughout a reproducible research project.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-5.html>

Lab

-  update

What: Control statements, custom functions, and iteration^a

How: Clone, fork, and complete the steps in Lab 5.

Why: To gain experience working with coding strategies such as control statements, custom functions, and iteration, practice working with direct downloads and API interfaces to acquire data, and implement organizational strategies for organizing data in reproducible fashion.

^a<https://github.com/qtalr/lab-5>

Questions

-  create conceptual and technical questions

Conceptual questions

- ...
- For many resources, information to describe the data origin is found on the resource's website. Visit the XXX resource and complete the data origin information.

Technical exercises

- ...
- ...

File 5.1 1-acquire-data.qmd: Acquire data file

```
---
```

```
title: "Acquire data"
format: html
---
```

```
## Overview
```

```
The goal of this script is to acquire and document data for this project from
↳ the CEDEL2 corpus. The acquired data will be stored in the
↳ `data/original/cedel2/` directory.
```

```
## Data origin
```

```
To document the origin of the data we created a file named `cedel2_do.csv` in the
↳ `data/original/` directory. This file contains the following information:
```

```
```{r}
```

```
#| label: tbl-cedel2-data-origin
#| tbl-cap: "Data origin file for the CEDEL2 corpus"
#| echo: false
```

```
Display data origin file
```

```
readr::read_csv("../data/original/cedel2_do.csv") |> knitr::kable()
```
## Download procedures
```

```
The process to acquire data from the CEDEL2 corpus involved the following steps:
```

```
L2 Spanish Learners:
```

1. Navigate to the [CEDEL2 Corpus](<http://cedel2.learnercorpora.com/search>)
 ↳ search interface
2. Select the subcorpus "Learners of L2 Spanish"
3. Set the results to provide full texts
4. Filter the results to "L1 English - L2 Spanish"
5. Set the medium to "Written"
6. Download the data in CSV format
7. Save the CSV file to the `data/original/cedel2/` directory as
 ↳ `cedel2-l1-english-learners.csv`

File 5.2 1-acquire-data.qmd: Acquire data file

```
---
```

```
title: "Acquire data"
format: html
---
```

```
## Overview
```

```
The goal of this script is to ...
```

```
## Data origin
```

```
To document the origin of the data we created a file named ...
```

```
## Download procedures
```

```
```{r}
#| label: setup
```

```
Load libraries
library(tidyverse)
````
```

```
```{r}
... additional code here to acquire data ...
````
```

```
... and so on
```

6 Curate datasets



Draft

Ready for review.

The hardest bit of information to extract is the first piece.

— Robert Ferrigno



Outcomes

- Describe the importance of data curation in text analysis
- Recognize the different types of data formats
- Associate the types data formats with the appropriate R programming techniques to curate the data

In this chapter we will now look at the next step in a text analysis project: data curation. That is, the process of converting the original data we acquire to a tidy dataset. Acquired data can come in a wide variety of formats. These formats tend to signal the richness of the metadata that is included in the file content. In this chapter we will consider three general types of content formats: (1) unstructured data, (2) structured data, and (3) semi-structured data. Regardless of the file type and the structure of the data, it will be necessary to consider how to curate a dataset such that the structure reflects the basic the unit of analysis that we wish to investigate. The resulting dataset will form the base from which we will work to further transform the dataset such that it aligns with the unit(s) of observation required for the analysis method(s) that we will implement. Once the dataset is curated, we will create a data dictionary that describes the dataset and the variables that are included in the dataset for transparency and reproducibility.



Swirl lesson

What: Pattern Matching and Manipulate Datasets^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: To familiarize yourself with the basics of using the pattern matching syntax Regular Expressions and the `dplyr` package to manipulate datasets.

^a<https://github.com/qtalr/lessons>

6.1 Unstructured

The bulk of text ever created is of the unstructured variety. Unstructured data is data that has not been organized to make the information contained within machine-readable. Remember that text in itself is not information. Only when given explicit context does text become informative. The explicit contextual information that is included with data is called metadata. Metadata can be linguistic or non-linguistic in nature. So for unstructured data there is little to no metadata directly associated with the data. This information needs to be added or derived for the purposes of the research, either through manual inspection or (semi-)automatic processes. For now, however, our job is just to get the unstructured data into a structured format with a minimal set of metadata that we can derive from the resource.

6.1.1 Reading data

Let's consider some of the common file formats which contain unstructured data, such as TXT, PDF, and DOCX, and how to read these formats into an R session.

It is very common for unstructured data resources to include data that is in TXT files. These are the simplest of file formats which contain no formatting or explicit metadata. Many times TXT files have the *.txt* extension, but it is not required. There are many ways to read TXT files into R and many packages that can be used to do so. For example, using the `readr` package, we can choose to read the entire file into a single vector of character strings with `read_file()` or read the file by lines with `read_lines()` in which each line is a character string in a vector.

Less commonly used in prepared data resources, PDF and DOCX files are also formats for unstructured data. These formats are often more complex than TXT files as they contain formatting and embedded metadata. However, these attributes are primarily for visual presentation and not for machine-readability. Therefore, we need to use packages and functions that can extract the text content from these files and potentially some of the metadata. For example, using the `readtext` package (Benoit and Obeng 2023), we can read the text content from PDF and DOCX files into a single vector of character strings with `readtext()`.

Whether in TXT, PDF, or DOCX format, the resulting data structure will require further processing to convert the data into a tidy dataset. For example, we may need to split the data into multiple columns or rows, or extract metadata from the text content.

6.1.2 Orientation

As an example of curating an unstructured source of corpus data, let's take a look at the Europarl Parallel Corpus¹ (Koehn 2005). This corpus contains parallel texts (source and translated documents) from the European Parliamentary proceedings between 1996-2011 for some 21 European languages.

Let's assume we selected this corpus data because we are interested in researching Spanish to English translations. After consulting the corpus website, downloading the compressed file, and inspecting the decompressed structure, we have the file structure seen in Example 6.1.

Example 6.1.

```
project/
└── code/
    ├── 1-acquire-data.qmd
    ├── 2-curate-data.qmd
    └── ...
└── data/
    ├── analysis/
    ├── derived/
    └── original/
        ├── europarl_do.csv
        └── europarl/
            ├── europarl-v7.es-en.en
            └── europarl-v7.es-en.es
└── output/
    ├── figures/
    ├── reports/
    ├── results/
    └── tables/
└── README.md
└── _main.R
```

The *europarl_do.csv* file contains the data origin information documented as part of the acquisition process. The contents are seen in Table 6.1.

Now let's get familiar with the corpus directory structure and the files. In Example 6.1, we see that there are two corpus files, *europarl-v7.es-en.es* and *europarl-v7.es-en.en*, that contain the source and target language texts, respectively. The file names indicate that the files contain Spanish-English parallel texts. The *.es* and *.en* extensions indicate the language of the text.

¹<https://www.statmt.org/europarl/>

Table 6.1: Data origin: Europarl Corpus

| attribute | description |
|-------------------------|--|
| Resource name | Europarl Parallel Corpus |
| Data source | < https://www.statmt.org/europarl/ > |
| Data sampling frame | Spanish transcripts from the European Parliament proceedings |
| Data collection date(s) | 1996-2011 |
| Data format | TXT files with '.es' for source (Spanish) and '.en' for target (English) files. |
| Data schema | Line-by-line unannotated parallel text |
| License | See: < https://www.europarl.europa.eu/legal-notice/en/ > |
| Attribution | Please cite the paper: Koehn, P. 2005. 'Europarl: A Parallel Corpus for Statistical Machine Translation' |

Looking at the beginning of the *.es* and *.en* files, in File 6.1 and File 6.2, we see that the files contain a series of lines in either the source or target language.

File 6.1 data/original/europarl/europarl-v7.es-en.es: Spanish source text

Reanudación del periodo de sesiones

Después de una pausa del período de sesiones del Parlamento Europeo interrumpido el
 ↳ viernes 16 de diciembre, los pasados y reitero a los Señores diputados que
 ↳ han sido vittimas de catastrofes naturales verdaderamente terribles.

We can clearly appreciate that the data is unstructured. That is, there is no explicit metadata associated with the data. The data is just a series of lines. The only information that we can surmise from structure of the data is that the texts are line-aligned and that the data in each file corresponds to source and target languages.

Now, before embarking on a data curation process, it is recommended to define the structure of the data *curta* we want to create. This is the “idealized structure” of the data. For a curated dataset there are two considerations. First, we want to maintain the original structure of the data as much as possible. This provides a link to the original data and gives us a default dataset structure from which we can derive other structures that fit our analysis needs. Second, we want a *tidy* structure to the data that makes it easier to work with. This is the ‘tidy’ part

Resumption of the session

I declare resumed the session of the European Parliament adjourned on Friday 17

↪ December 1999, and I would like once again to wish you a happy new year in
↪ the hope that you enjoyed a pleasant festive period.

Although, as you will have seen, the dreaded 'millennium bug' failed to

↪ materialise, still the people in a number of countries suffered a series of
↪ natural disasters that truly were dreadful.

You have requested a debate on this subject in the course of the next few days,

↪ during this part-session.

In the meantime, I should like to observe a minute's silence, as a number of

↪ Members have requested, on behalf of all the victims concerned, particularly
↪ those of the terrible storms, in the various countries of the European Union.

of the data curation process. The aim is to imbue the dataset with as much of the metadata the data resource makes available.

Given what we know about the data, we can define the idealized structure of the data as seen in Table 6.2.

Table 6.2: Idealized structure for the europarl Corpus dataset.

| doc_id | type | line_id | line |
|--------|--------|---------|------------------------------|
| 1 | Source | 1 | ...line from source language |
| 2 | Source | 2 | ... |
| 3 | Source | 3 | ... |
| 4 | Target | 1 | ...line from target language |
| 5 | Target | 2 | ... |
| 6 | Target | 3 | ... |

The dataset structure in Table 6.2 has four columns. The first column, `doc_id` is a unique identifier for each line in the corpus. `type`, indicates whether the line is from the source or target language. The third column, `line_id`, is a unique identifier for each line for each type. The last column, `line`, contains the text of the line, and maintains the structure of the original data. The observations are lines.

Our task now is to develop code that will read the original data and render the idealized structure as a curated dataset we will write to the `data/derived/` directory. The code we

develop will be added to the *2-curate-data.qmd* file. And finally, the dataset will be documented with a data dictionary file.

6.1.3 Tidy the data

To create the idealized dataset structure in Table 6.2, let's start by reading the files into R. We will use the `readtext()` function from the `readtext` package. `readtext()` is a versatile function that can read many different types of text files (e.g. *.txt*, *.csv*, *xml*, etc.). It can also read multiple files at once using wildcard matching. We will use it to read the *.es* and *.en* files by passing the path to the directory where the files are located, and then use the `*` wildcard to read all the files in the directory.

```
# Load package
library(readtext)

# Read Europarl files .es and .en
europarl_docs_tbl <-
  readtext("../data/original/europarl/*") |> # read in files
  tibble() # convert to tibble
```

Example 6.2.

In Example 6.2, the `readtext()` function reads all the files in the *../data/original/europarl/* directory and returns a tibble.

⚠ Warning

The `readtext()` function can read many different types of file formats, from structured to unstructured. However, it depends in large part on the extension of the file to recognize what algorithm to use when reading a file. In this particular case, the Europarl files do not have a typical extension (they have *.en* and *.es*). The `readtext()` function will treat them as plain text (*.txt*), but it will throw a warning message. To suppress the warning message you can add the `verbosity = 0` argument or set `readtext_options(verbosity = 0)`.

Let's inspect the `europarl_docs_tbl` object with the `str()` function, in Example 6.3.

Example 6.3.

```
# Preview data
str(europarl_docs_tbl)
```

```

> tibble [2 x 2] (S3: tbl_df/tbl/data.frame)
> $ doc_id: chr [1:2] "europarl-v7.es-en.en" "europarl-v7.es-en.es"
> $ text : chr [1:2] "Resumption of the session\nI declare resumed the session of the European Parliament"

```

We see that the output from Example 6.3 is a tibble with two columns, `doc_id` and `text`. The `doc_id` column contains the name of the file from which the text was read. The `text` column contains the text of the file. There are two observations, one for each file.

 **Tip**

Note that the `str()` function from base R is similar to `glimpse()`. However, `glimpse()` will attempt to show you as much data as possible. In this case since our column `text` is a very long character vector it will take a long time to render. I've chosen the `str()` function as it will automatically truncate the data.

The fact that we only have one row for each file means that all the text in each file is contained in one cell! We want to break these cells up into rows for each line, as they appear in the original data. You may be wondering why the `readtext()` function did not do this for us, after all the original data was already separated into lines. The reason is that the `readtext()` function chose to read the files as plain text, and plain text does not have any structure. The line breaks, however, are still there, they are just represented as a special character, `\n`. Comparing Example 6.4 and Example 6.5, we can see that the text is a single long character vector, and that the line breaks are represented as `\n`.

Example 6.4.

```

# Preview first 50 characters
europarl_docs_tbl |>
  pull(text) |>
  str_trunc(50) |>
  str_view()

> [1] | Resumption of the session
>     | I declare resumed the...
> [2] | Reanudación del periodo de sesiones
>     | Declaro rea...

```

Example 6.5.

```

# Preview first 50 raw characters
europarl_docs_tbl |>

```

```
pull(text) |>
  str_trunc(50) |>
  str_view(use_escapes = TRUE)
```

```
> [1] | Resumption of the session\nI declare resumed the...
> [2] | Reanudaci\u00f3n del per\u00f3ximo estado de sesiones\nDeclaro rea...
```

In Example 6.4, we can see a truncated version of the text as it is in a printed format. The `str_trunc()` function from the `stringr` package (Wickham 2022) truncates the text to the first 50 characters. The `str_view()` function from the same package allows us to see the text in a viewer pane.

In Example 6.5, we can see the raw text, that is, the text as it is stored in the computer. The `use_escapes = TRUE` argument tells the `str_view()` function to show the special characters as they are stored in the computer. This includes the `\n` line-feed character, which represents a line break.



Tip

The `str_view()` with the argument `use_escapes = TRUE` also allows you to see other special characters, such as `\t` (tab) and `\r` (carriage return) as well as Unicode characters, such as `\u00f3n` (ó), `\u00ed` (í), etc.. These characters are not visible in the printed version of the text, but they are there in the raw text.

We can see that the text is a single long character vector, and that the line breaks are represented as `\n`. So our goal is to split the text for each file into lines creating a new row for each line created.

To do this we will use another function from the `stringr` package, `str_split()`, whose function is to split a character vector into smaller character vectors based on some splitting criteria. In Example 6.6, we use the `str_split()` function to split the text into lines based on the `\n` character and assign it to the new column `lines` using `mutate()`. Since we will not need the `text` column anymore, we can use `select()` to drop it.

Example 6.6.

```
# Split text into lines
europarl_lines_tbl <-
  europarl_docs_tbl |>
  mutate(lines = str_split(text, "\n")) |>
  select(-text)
```

```
# Preview
europarl_lines_tbl

> # A tibble: 2 × 2
>   doc_id          lines
>   <chr>         <list>
> 1 europarl-v7.es-en.en <chr [1,965,734]>
> 2 europarl-v7.es-en.es <chr [1,965,734]>
```

Previewing the output in Example 6.6, we can see that the `lines` column contains a list of character vectors. Why is this so? `str_split()` takes a character vector and splits it, as we know. It is a vectorized function, meaning that it can take a vector of character vectors and split each one into subsegments. Since any given input character vector can have a different number of subsegments, the number of subsegments in each file could be different. The ideal object for such data is a list. So `str_split()` returns a list of character vectors.

The list of character vectors, `lines` in our case, is still associated with the respective `doc_id` value. However, we want to create a new row for each line in the list. To do this, we will use the `unnest()` function from the `tidyverse` package (Wickham, Vaughan, and Girlich 2023). The `unnest()` function takes a list column and creates a new row for each element in the list. We use the `unnest()` function to create a new row for each line in the `lines` column. We can see the output from Example 6.7.

Example 6.7.

```
# Create a new row for each line
europarl_lines_tbl <-
  europarl_lines_tbl |>
  unnest(lines)

# Preview
europarl_lines_tbl |>
  slice_head(n = 5)

> # A tibble: 5 × 2
>   doc_id          lines
>   <chr>         <chr>
> 1 europarl-v7.es-en.en Resumption of the session
> 2 europarl-v7.es-en.en I declare resumed the session of the European Parliament~
> 3 europarl-v7.es-en.en Although, as you will have seen, the dreaded 'millennium~
```

```
> 4 europarl-v7.es-en.en You have requested a debate on this subject in the cours~  
> 5 europarl-v7.es-en.en In the meantime, I should like to observe a minute' s si~
```

Remember that the data in the Europarl corpus is aligned, meaning that each line in the source file is aligned with a line in the target file. So we need to make sure that the number of lines in each file is the same. We can do this by grouping the data by `doc_id` and then counting the number of lines in each file. We can do this using the `group_by()` and `count()` functions from the `dplyr` package (Wickham, François, et al. 2023). We can see the output in Example 6.8.

Example 6.8.

```
# Count the number of lines in each file  
europarl_lines_tbl |>  
  group_by(doc_id) |>  
  count()  
  
> # A tibble: 2 × 2  
> # Groups: doc_id [2]  
>   doc_id           n  
>   <chr>          <int>  
> 1 europarl-v7.es-en.en 1965734  
> 2 europarl-v7.es-en.es 1965734
```

The output of Example 6.8 shows that the number of lines in each file is the same. This is good. If the number of lines in each file was different, we would need to figure out why and fix it.

We now have our `lines` column and the associated observations for our idealized dataset, in Table 6.2. Let's now leverage the existing `doc_id` to create the `type` column. The goal is to assign the value of `type` according to the `doc_id` value. Specifically, when `doc_id` is 'europarl-v7.es-en.es' type should be 'Source' and when `doc_id` is 'europarl-v7.es-en.en' type should be 'Target'.

We can do this using the `case_when()` function from the `dplyr` package (Wickham, François, et al. 2023). The `case_when()` function takes a series of conditions and assigns a value based on the first condition that is met. In this case, we will use `mutate()` to create the `type` column and then use the `doc_id` column to create the conditions and the `type` column to assign the values from `case_when()`. We will store the output in a new object called `europarl_lines_type_tbl`, as seen in Example 6.9.

Example 6.9.

```

# Create `type` column
europarl_lines_type_tbl <-
  europarl_lines_tbl |>
  mutate(type = case_when(
    doc_id == "europarl-v7.es-en.es" ~ "Source",
    doc_id == "europarl-v7.es-en.en" ~ "Target"
  ))

# Preview dataset
glimpse(europarl_lines_type_tbl)

```

```

> Rows: 3,931,468
> Columns: 3
> $ doc_id <chr> "europarl-v7.es-en.en", "europarl-v7.es-en.en", "europarl-v7.es-
> $ lines  <chr> "Resumption of the session", "I declare resumed the session of ~
> $ type   <chr> "Target", "Target", "Target", "Target", "Target", "Target", "Ta-

```

The preview output from Example 6.9 shows us that we now have three columns, `doc_id`, `lines`, and `type`. The `type` column has been created and the values have been assigned according to the `doc_id` values.

We can now overwrite the `doc_id` column with a unique identifier for each line. A numeric identifier makes sense. We can do this by using the `mutate()` function to assign `doc_id` a sequential number with `row_number()`, which increments by 1 for each row. We will store the output in a new object called `europarl_lines_type_id_tbl`, as seen in Example 6.10.

Example 6.10.

```

# Create new `doc_id` column
europarl_lines_type_id_tbl <-
  europarl_lines_type_tbl |>
  mutate(doc_id = row_number())

# Preview dataset
glimpse(europarl_lines_type_id_tbl)

```

```

> Rows: 3,931,468
> Columns: 3
> $ doc_id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, ~
> $ lines  <chr> "Resumption of the session", "I declare resumed the session of ~
> $ type   <chr> "Target", "Target", "Target", "Target", "Target", "Target", "Ta-

```

The preview output from Example 6.10 shows us that we now have the same three columns, `doc_id`, `lines`, and `type`. However, the values in the `doc_id` column now reflect a unique identifier for each line.

The last step to get to our envisioned dataset structure is to add the `line_id` column which will be calculated by grouping the data by `type` and then assigning a row number to each of the lines in each group. We use the `group_by()` function to perform the grouping as seen in Example 6.11.

Example 6.11.

```
# Create `line_id` column
europarl_lines_type_id_line_id_tbl <-
  europarl_lines_type_id_tbl |>
  group_by(type) |>
  mutate(line_id = row_number()) |>
  ungroup()

# Preview dataset
glimpse(europarl_lines_type_id_line_id_tbl)

> Rows: 3,931,468
> Columns: 4
> $ doc_id  <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, ~
> $ lines    <chr> "Resumption of the session", "I declare resumed the session of~
> $ type     <chr> "Target", "Target", "Target", "Target", "Target", "Target", "T~
> $ line_id <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, ~
```

The preview output from Example 6.11 shows us that we now have the desired four columns, `doc_id`, `lines`, `type`, and `line_id`.

Before we get to writing the dataset to disk, let's organize it in a way that the columns are in the order we want and the rows are sorted in a way that makes sense.

Reordering the columns involves using the `select()` function and list the order of the columns. To sort by columns, we will use `arrange()` and specify the columns to order by. We store the results in a more legible object, `europarl_curated_tbl`, as seen in Example 6.12.

Example 6.12.

```
# Reorder columns and sort rows
europarl_curated_tbl <-
```

```

europarl_lines_type_id_line_id_tbl |>
  select(doc_id, type, line_id, lines) |>
  arrange(line_id, type, doc_id)

# Preview dataset
glimpse(europarl_curated_tbl)

> Rows: 3,931,468
> Columns: 4
> $ doc_id <int> 1965735, 1, 1965736, 2, 1965737, 3, 1965738, 4, 1965739, 5, 19~
> $ type    <chr> "Source", "Target", "Source", "Target", "Source", "Target", "S~
> $ line_id <int> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, ~
> $ lines   <chr> "Reanudación del periodo de sesiones", "Resumption of the sess~

```

The preview output from Example 6.12 shows us that we now have the desired four columns, `doc_id`, `type`, `line_id`, and `lines`. The rows are sorted by `line_id`, `type`, and `doc_id`. This gives us a dataset that reads left to right from document to line oriented attributes and top to bottom by source-target line pairs.

6.1.4 Write dataset

At this point we have the curated dataset (`europarl_curated_tbl`) in a tidy format. This dataset, however, is only in the current R session. We will want to write this dataset to disk so that in the next step of the text analysis workflow (transformation) we will be able to start work on this dataset and make changes as needed to fit our analysis needs.

We will leverage the project directory structure which has distinct directories for `original/` and `derived/` data(sets), seen in Example 6.13.

Example 6.13.

```

data/
|— analysis/
|— derived/
└— original/
  |— europarl_do.csv
  └— europarl/
    |— europarl-v7.es-en.es
    └— europarl-v7.es-en.en

```

Since this is a tabular, tidy dataset we have various options for the file type to write. Many of these formats are software-specific, such as `*.xlsx` for Microsoft Excel, `*.sav` for SPSS, `*.dta` for Stata, and `*.rds` for R. We will use the `*.csv` format since it is a common format that can be read by many software packages. We will use the `write_csv()` function from the `readr` package to write the dataset to disk.

Now the question is where to save our CSV file. Since our `europarl_curated_tbl` dataset is derived by our work, we will add it to the `derived/` directory. I'll create a `europarl/` directory with `dir_create()` just to keep things organized.

Example 6.14.

```
# Create the europarl/ directory
dir_create(path = "../data/derived/europarl/")

# Write the curated dataset to disk
write_csv(
  x = europarl_curated_tbl,
  file = "../data/derived/europarl/europarl_curated.csv"
)
```

After running the code in Example 6.14, the directory structure under the `derived/` directory should look like Example 6.15.

Example 6.15.

```
data/
|__ analysis/
|__ derived/
|  |__ europarl/
|  |  |__ europarl_curated.csv
|__ original/
  |__ europarl
    |__ europarl-v7.es-en.en
    |__ europarl-v7.es-en.es
```

The final step, as always, is to document the dataset. For datasets the documentation is a data dictionary, as discussed in Section 2.3.2. As with data origin files, you can use spreadsheet software to create and/ or edit the data dictionary.

In the `qtlrkit` package we have a function, `create_data_dictionary()` that will generate the scaffolding for a data dictionary. The function takes two arguments, `data` and `file_path`. It reads the dataset columns and provides a template for the data dictionary.

Table 6.3: Data dictionary for the `europarl_curated_tbl` dataset.

| variable | name | variable_type | description |
|----------|---------------|---------------|---|
| doc_id | Document ID | numeric | Unique identification number for each document |
| type | Document Type | categorical | Type of document; either 'Source' (Spanish) or 'Target' (English) |
| line_id | Line ID | numeric | Unique identification number for each line in each document type |
| lines | Lines | categorical | Content of the lines in the document |

⭐ Dive deeper

The `create_data_dictionary()` function provides a rudimentary data dictionary template by default. However, you can take advantage of OpenAI's text generation models to generate a more detailed data dictionary for you to edit. To do this create an OpenAI account^a and an API key^b and add this key to your R environment (`Sys.setenv(OPENAI_API_KEY = "sk...")`). Then you can specify the model you would like to use in the function with the `model =` argument. For example, `model = "gpt-3.5-turbo"` will use the GPT-3.5 Turbo model.

^a<https://platform.openai.com/signup>

^b<https://platform.openai.com/account/api-keys>

Example 6.16.

```
# Create the data dictionary
create_data_dictionary(
  data = europarl_curated_tbl,
  file_path <- "../data/derived/europarl/europarl_curated_dd.csv"
)
```

An example of the data dictionary for the `europarl_curated_tbl` dataset is shown in Table 6.3.

6.2 Structured

Structured data already reflects the physical and semantic structure of a tidy dataset. This means that the data is already in a tabular format and the relationships between columns and rows are already well-defined. Therefore the heavy lifting of curating the data is already done. There are two remaining questions, however, that need to be taken into account. One, logistical question, is what file format the dataset is in and how to read it into R. And the second, more research-based, is whether the data may benefit from some additional curation and documentation to make it more amenable to analysis and more understandable to others.

6.2.1 Reading data

Let's consider some common formats for structured data and how to read them into R. First, we will consider R-native formats, such as package datasets and RDS files. Then will consider non-native formats, such as relational databases and datasets produced by other software. Finally, we will consider software agnostic formats, such as CSV.

R and some R packages provide structured datasets that are available for use directly within R. For example, the `languageR` package (**R-languageR?**) provides the `dative` dataset, which is a dataset containing the realization of the dative as NP or PP in the Switchboard corpus and the Treebank Wall Street Journal collection. The `janeaustenr` package (Silge 2022) provides the `austen_books` dataset, which is a dataset of Jane Austen's novels. Package datasets are loaded into an R session using either the `data()` function, if the package is loaded, or the `::` operator, if the package is not loaded. For example, `data(dative)` or `languageR::dative`.

Dive deeper

To explore the available datasets in a package, you can use the `data(package = "package_name")` function. For example, `data(package = "languageR")` will list the datasets available in the `languageR` package. You can also explore all the datasets available in the loaded packages with the `data()` function using no arguments. For example, `data()`.

R also provides a native file format for storing R objects, the RDS file. Any R object, including data frames, can be written from an R session to disk by using the `write_rds()` function from `readr`. The `.rds` files will be written to disk in a binary format that is not human-readable, which is not ideal for transparent data sharing. However, the files and the R objects can be read back into an R session using the `read_rds()` function with all the attributes intact, such as vector types, factor levels, *etc.*

R provides a suite of tools for importing data from non-native structured sources such as databases and datasets from software such as SPSS, SAS, and Stata. For instance, if you are working with data stored in a relational database such as MySQL, PostgreSQL, or SQLite, you can use the `DBI` package (R Special Interest Group on Databases (R-SIG-DB), Wickham, and Müller 2022) to connect to the database and the `dbplyr` package (Wickham, Girlich, and Ruiz 2023) to query the database using the SQL language. Files from SPSS (`.sav`), SAS (`.sas7bdat`), and Stata (`.dta`) can be read into R using the `haven` package (Wickham, Miller, and Smith 2023).

Software agnostic file formats include delimited files, such as CSV, TSV, *etc.*. These file formats lack the robust structural attributes of the other formats, but balance this shortcoming by storing structured data in more accessible, human-readable format. Delimited files are plain text files which use a delimiter, such as a comma (,), tab (\t), or pipe (|), to separate the columns and rows. For example, a CSV file is a delimited file where the columns and rows are separated by commas, as seen in Example 6.17.

Example 6.17.

```
column_1,column_2,column_3
row 1 value 1,row 1 value 2,row 1 value 3
row 2 value 1,row 2 value 2,row 2 value 3
```

Given the accessibility of delimited files, they are a common format for sharing structured data in reproducible research. It is not surprising, then, that this is the format which we have chosen for the derived datasets in this book.

6.2.2 Orientation

With an understanding of the various structured formats, we can now turn to considerations about how the original dataset is structured and how that structure is to be used for a given research project. As an example, we will work with the CABNC datasets acquired in Chapter 5. The structure of the original dataset is shown in Example 6.18.

Example 6.18.

```
data/
└── analysis/
└── derived/
└── original/
    └── cabnc_do.csv
    └── cabnc/
        ├── participants.csv
        ├── token_types.csv
        ├── tokens.csv
        ├── transcripts.csv
        └── utterances.csv
```

In addition to other important information, the data origin file *cabnc_do.csv* shown in Table 6.4 informs us the the datasets are related by a common variable.

The CABNC datasets are structured in a relational format, which means that the data is stored in multiple tables that are related to each other. The tables are related by a common column or set of columns, which are called a keys. A key is used to join the tables together to create a single dataset. There are two keys in the CABNC datasets, `filename` and `who`. Each variable corresponds to recording- and/ or participant-oriented datasets.

Table 6.4: Data origin: CABNC datasets

| attribute | description |
|-------------------------|--|
| Resource name | CABNC. |
| Data source | < https://ca.talkbank.org/access/CABNC.html >, <doi:10.21415/T55Q5R> |
| Data sampling frame | Over 400 British English Speakers from across the UK stratified age, gender, social |
| Data collection date(s) | 1992. |
| Data format | CSV Files |
| Data schema | The recordings are linked by ‘filename’ and the participants are linked by ‘who’. |
| License | CC BY NC SA 3.0 |
| Attribution | Saul Albert, Laura E. de Ruiter, and J.P. de Ruiter (2015) CABNC: the Jeffersonian |

Now, let’s envision a scenario in which we want to organize a dataset that can be used in a study that aims to investigate the relationship between speaker demographics and utterances. An ideal dataset would contain information about speakers and their utterances. In their original format, the CABNC datasets separate information about utterances and speakers in separate tables, `cabnc_utterances` and `cabnc_participants`, respectively. The idealized dataset, then, will combine the variables from each of these tables into a single dataset.

6.2.3 Tidy the dataset

With our idealized dataset in mind, let’s start the process of curation by reading the relevant datasets into an R session. Since we are working with CSV files will will use the `read_csv()` function, as seen in Example 6.19.

Example 6.19.

```
# Read the relevant datasets
cabnc_utterances <-
  read_csv("data/cabnc/original/utterances.csv")
cabnc_participants <-
  read_csv("data/cabnc/original/participants.csv")
```

The next step is to inspect the structure of the datasets. We can use the `glimpse()` function for this task.

Example 6.20.

```
# Preview the structure of the datasets
glimpse(cabnc_utterances)
```

```

> Rows: 235,901
> Columns: 10
> $ filename <chr> "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", ~
> $ path      <chr> "ca/CABNC/KB0/KB0RE000", "ca/CABNC/KB0/KB0RE000", "ca/CABNC/~
> $ utt_num   <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17~
> $ who       <chr> "PS002", "PS006", "PS002", "PS006", "PS002", "PS006", "PS002~
> $ role      <chr> "Unidentified", "Unidentified", "Unidentified", "Unidentifie~
> $ postcodes <lgl> NA, ~
> $ gems      <lgl> NA, ~
> $ utterance  <chr> "You enjoyed yourself in America", "Eh", "did you", "Oh I co~
> $ startTime <dbl> 0.208, 2.656, 2.896, 3.328, 5.088, 6.208, 8.320, 8.480, 10.2~
> $ endTime   <dbl> 2.67, 2.90, 3.33, 5.26, 6.02, 8.50, 9.31, 11.23, 14.34, 15.9~

```

```
glimpse(cabnc_participants)
```

```

> Rows: 6,190
> Columns: 13
> $ filename <chr> "KB0RE004", "KB0RE004", "KB0RE004", "KB0RE006", "KB0RE006", ~
> $ path      <chr> "ca/CABNC/0missing/KB0RE004", "ca/CABNC/0missing/KB0RE004", ~
> $ who       <chr> "PS008", "PS009", "KB0PSUN", "PS007", "PS008", "PS009", "KB0~
> $ name      <chr> "John", "Gethyn", "Unknown_speaker", "Alan", "John", "Gethyn~
> $ role      <chr> "Unidentified", "Unidentified", "Unidentified", "Unidentifie~
> $ language   <chr> "eng", "eng", "eng", "eng", "eng", "eng", "eng", "eng~
> $ monthage  <dbl> 481, 481, 13, 949, 481, 481, 13, 637, 565, 13, 637, 565, 13, ~
> $ age        <chr> "40;01.01", "40;01.01", "1;01.01", "79;01.01", "40;01.01", "~~
> $ sex        <chr> "male", "male", "male", "male", "male", "male", "male", "mal~
> $ numwords   <dbl> 28, 360, 156, 1610, 791, 184, 294, 93, 3, 0, 128, 24, 0, 150~
> $ numutts   <dbl> 1, 9, 27, 7, 5, 7, 6, 5, 1, 0, 11, 6, 0, 110, 74, 96, 12, 1, ~
> $ avgutt    <dbl> 28.00, 40.00, 5.78, 230.00, 158.20, 26.29, 49.00, 18.60, 3.0~
> $ medianutt <dbl> 28, 39, 5, 84, 64, 9, 3, 15, 3, 0, 9, 3, 0, 7, 6, 4, 3, 12, ~

```

From visual inspection of the output of Example 6.20 we can see that there are common variables in both datasets. It is also possible to intersect the datasets to see which variables are common using the `intersect()` function to intersect the column names of each data frame with the `names()` function, as seen in Example 6.21.

Example 6.21.

```
# Find the common variables
common_vars <-
```

```

intersect(
  names(cabnc_utterances),
  names(cabnc_participants)
)

# Print the common variables
common_vars

```

> [1] "filename" "path" "who" "role"

Using both visual inspection and column name intersection, we can make sure that the variable names and the values are consistent across the datasets. For example, if the variable `filename` in one dataset is called `file_name` in another dataset, then we will need to rename the variable in one of the datasets so that the variable names are consistent. Furthermore, if the values in the `filename` variable are not consistent across the datasets, then we will need to make the values consistent, if possible, before joining the datasets.

In this case, the variable names and values are consistent across the datasets. Therefore, we can join the datasets together using the `left_join()` function from the `dplyr` package, as seen in Example 6.22. This function will take the dataset on the left (`x =`) and join it to the dataset on the right (`y =`). The `by` argument specifies the variables to join on. The choice of which dataset to put on the left usually depends on which dataset has the most detailed information. In this case, the `cabnc_utterances` dataset has more detailed information about the utterances so we will put this dataset on the left.

Example 6.22.

```

# Join the datasets
cabnc_tbl <-
  left_join(
    x = cabnc_utterances,
    y = cabnc_participants,
    by = common_vars
  )

# Preview the dimensions of the joined dataset
dim(cabnc_tbl)

```

> [1] 235901 19

The result of Example 6.22 should produce a dataset with the same number of observations as the `cabnc_utterances` dataset, seen in Example 6.20, and a combined number of unique variables from both datasets, that is 23 minus the four common variables, seen in Example 6.21. The output of `dim()` confirms this.

Now we will consider the variables that will be useful for future analysis. Since we are creating a curated dataset, the goal will be to retain as much information as possible from the original datasets. There are cases, however, in which there may be variables that are not informative and thus, will not prove useful for any analysis. These removable variables tend to be of one of two types: variables which show no variation across observations and variables where the information is redundant.

Let's get a high-level summary of the variables in the dataset. We can use the `skim()` function from the `skimr` package (Waring et al. 2022) to get a summary of the variables in the dataset².

Example 6.23.

```
# Load package
library(skimr)

# Summarize the variables in the dataset
skim(cabnc_tbl)
```

We see from the `skim()` output in Table 6.5b, that the variables `postcodes` and `gems` have no values. Therefore we can remove these variables from the dataset. On the other hand, in Table 6.5a, the variables `role` and `language` have the same value for every observation as the output shows `n_unique` is 1. We can also remove these variables from the dataset.

Another set of variables of potential interest are the `startTime` and `endTime` variables, in Table 6.5c. These each have a completion rate of less than 100%, specifically 84%. On first blush, it would seem that we would go ahead and remove them. However, we should pause and consider that these variables may still be of some use. At the curation stage, however, it is best to err on the side of caution and leave them in the dataset. We will decide later whether to keep or remove them as we explore the dataset further in the research.

Another consideration is whether we have variables that are clearly redundant. For example, the variables `age` and `monthage` are both measures of age, stated in different measures, looking back at Example 6.20. As such, we can remove one of these variables from the dataset. In this case, we will remove the `age` variable as it is the least straightforward to interpret.

Another potentially redundant set of variables are `who` and `name` –both of which are speaker identifiers. The `who` variable is a unique identifier, but there may be some redundancy with the

²Note I've modified the output of `skim()` for display purposes.

Table 6.5: Summary of variables in the CABNC dataset

(a) ?(caption)

| variable | complete_rate | n_unique |
|-----------|---------------|----------|
| filename | 1 | 2020 |
| path | 1 | 2020 |
| who | 1 | 568 |
| role | 1 | 1 |
| utterance | 1 | 174414 |
| name | 1 | 269 |
| language | 1 | 1 |
| age | 1 | 83 |
| sex | 1 | 2 |

Categorical variables

(b) ?(caption)

| variable | complete_rate | mean |
|-----------|---------------|------|
| postcodes | 0 | NA |
| gems | 0 | NA |

Logical variables

(c) ?(caption)

| variable | complete_rate | mean |
|-----------|---------------|---------|
| utt_num | 1.000 | 231.67 |
| startTime | 0.841 | NA |
| endTime | 0.841 | NA |
| monthage | 1.000 | 448.61 |
| numwords | 1.000 | 1387.47 |
| numutts | 1.000 | 156.04 |
| avgutt | 1.000 | 9.13 |
| medianutt | 1.000 | 6.06 |

Numeric variables

`name` variable, that is there may be two speakers with the same name. We can check this by looking at the number of unique values in the `who` and `name` variables from the `skim()` output in Table 6.5a. `who` has 568 unique values and `name` has 269 unique values. This suggests that there are multiple speakers with the same name.

Another way to explore this is to look at the number of unique values in the `who` variable for each unique value in the `name` variable. We can do this using the `group_by()` and `summarize()` functions from the `dplyr` package. For each value of `name`, we will count the number of unique values in `who` and then sort the results in descending order.

Example 6.24.

```
cabnc_tbl |>
  group_by(name) |>
  summarize(n = unique(who) |> length()) |>
  arrange(desc(n))

> # A tibble: 269 x 2
>   name                n
>   <chr>              <int>
> 1 None                 59
> 2 Unknown_speaker      57
> 3 Group_of_unknown_speakers 10
> 4 Chris                 9
> 5 David                 9
> 6 Margaret                8
> 7 Ann                   7
> 8 John                   7
> 9 Alan                   6
> 10 Jackie                  5
> # i 259 more rows
```

It is good that we performed the check in Example 6.24 beforehand. In addition to speakers with the same name, such as ‘Chris’ and ‘David’, we also have multiple speakers with generic codes, such as ‘None’ and ‘Unknown_speaker’. It is clear that `name` is redundant and we can safely remove it from the dataset.

Another redundant variable is the `path` variable. This variable is not more informative than the `filename`, for our purposes. We will remove the `path` variable from the dataset too.

In all, we will remove the following variables from the dataset: `postcodes`, `gems`, `role`, `language`, `age`, `name`, and `path`. To drop variables from a data frame we can use the `select()` function

in combination with the `-` operator. The `-` operator tells the `select()` function to drop the variables that follow it.

Example 6.25.

```
# Drop variables
cabnc_tbl <-
  cabnc_tbl |>
  select(-postcodes, -gems, -role, -language, -age, -name, -path)

# Preview the dataset
glimpse(cabnc_tbl)
```

```
> Rows: 235,901
> Columns: 12
> $ filename <chr> "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", ~
> $ utt_num <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17~
> $ who <chr> "PS002", "PS006", "PS002", "PS006", "PS002", "PS006", "PS002~
> $ utterance <chr> "You enjoyed yourself in America", "Eh", "did you", "Oh I co~
> $ startTime <dbl> 0.208, 2.656, 2.896, 3.328, 5.088, 6.208, 8.320, 8.480, 10.2~
> $ endTime <dbl> 2.67, 2.90, 3.33, 5.26, 6.02, 8.50, 9.31, 11.23, 14.34, 15.9~
> $ monthage <dbl> 721, 601, 721, 601, 721, 601, 721, 601, 721, 601, 721, 601, ~
> $ sex <chr> "female", "male", "female", "male", "female", "male", "femal~
> $ numwords <dbl> 759, 399, 759, 399, 759, 399, 759, 399, 759, 399, 759, 399, ~
> $ numutts <dbl> 74, 64, 74, 64, 74, 64, 74, 64, 74, 64, 74, 64, 74, 2, 74, 6~
> $ avgutt <dbl> 10.26, 6.23, 10.26, 6.23, 10.26, 6.23, 10.26, 6.23, 10.26, 6~
> $ medianutt <dbl> 7, 5, 7, 5, 7, 5, 7, 5, 7, 1, 7, 5, 7, 5, 7, 5, ~
```

Now we have a dataset with 12 informative variables which describe the utterances for each recording. There are variables about the recordings, such as the `filename`, about the speakers, such as the `sex` and `who`, and about the utterances, such as the `utterance`, `utt_num`, *etc.*. Let's organize the columns to read left to right from most general to most specific. Again we turn to the `select()` function, this time including the variables in the order we want them to appear in the dataset. We will take this opportunity to rename some of the variable names so that they are more informative.

Example 6.26.

```
# Rename variables
cabnc_tbl <-
  cabnc_tbl |>
```

```

select(
  doc_id = filename,
  utt_num,
  utt_start = startTime,
  utt_end = endTime,
  utterance,
  part_id = who,
  part_age = monthage,
  part_sex = sex,
  num_words = numwords,
  num_utts = numutts,
  avg_utt_len = avgutt,
  median_utt_len = medianutt
)

# Preview the dataset
glimpse(cabnc_tbl)

```

```

> Rows: 235,901
> Columns: 12
> $ doc_id      <chr> "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE000", "KB0RE0~
> $ utt_num      <dbl> 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1~
> $ utt_start    <dbl> 0.208, 2.656, 2.896, 3.328, 5.088, 6.208, 8.320, 8.480, ~
> $ utt_end      <dbl> 2.67, 2.90, 3.33, 5.26, 6.02, 8.50, 9.31, 11.23, 14.34, ~
> $ utterance    <chr> "You enjoyed yourself in America", "Eh", "did you", "Oh~
> $ part_id      <chr> "PS002", "PS006", "PS002", "PS006", "PS002", "PS006", "~
> $ part_age     <dbl> 721, 601, 721, 601, 721, 601, 721, 601, 721, ~
> $ part_sex     <chr> "female", "male", "female", "male", "female", "male", "~
> $ num_words    <dbl> 759, 399, 759, 399, 759, 399, 759, 399, 759, 399, 759, ~
> $ num_utts     <dbl> 74, 64, 74, 64, 74, 64, 74, 64, 74, 64, 74, 64, 74, 2, ~
> $ avg_utt_len  <dbl> 10.26, 6.23, 10.26, 6.23, 10.26, 6.23, 10.26, 6.23, 10.~
> $ median_utt_len <dbl> 7, 5, 7, 5, 7, 5, 7, 5, 7, 5, 7, 1, 7, 5, 7, 5, 7, 5, 7, 5, 7~

```

The variable order is organized after running Example 6.26. Now let's sort the rows by `doc_id` and `utt_num` so that the utterances are in order. The `arrange()` function takes a data frame and a list of variables to sort by, in the order they are listed.

Example 6.27.

```

# Sort rows
cabnc_tbl <-
  cabnc_tbl |>
  arrange(doc_id, utt_num)

# Preview the dataset
cabnc_tbl |>
  slice_head(n = 10)

```

| doc_id | utt_num | utt_start | utt_end | utterance | part_id | part_age | part_sex |
|---|---------|-----------|---------|------------------|---------|----------|----------|
| 1 KB0RE000 | 0 | 0.208 | 2.67 | You enjoyed you~ | PS002 | 721 | female |
| 2 KB0RE000 | 1 | 2.66 | 2.90 | Eh | PS006 | 601 | male |
| 3 KB0RE000 | 2 | 2.90 | 3.33 | did you | PS002 | 721 | female |
| 4 KB0RE000 | 3 | 3.33 | 5.26 | Oh I covered a ~ | PS006 | 601 | male |
| 5 KB0RE000 | 4 | 5.09 | 6.02 | Oh very good ye~ | PS002 | 721 | female |
| 6 KB0RE000 | 5 | 6.21 | 8.50 | Er saw Mary and~ | PS006 | 601 | male |
| 7 KB0RE000 | 6 | 8.32 | 9.31 | Yes you did | PS002 | 721 | female |
| 8 KB0RE000 | 7 | 8.48 | 11.2 | in fact the who~ | PS006 | 601 | male |
| 9 KB0RE000 | 8 | 10.3 | 14.3 | Oh very nice ve~ | PS002 | 721 | female |
| 10 KB0RE000 | 9 | 14.3 | 16.0 | It is horrible ~ | PS006 | 601 | male |
| # i 4 more variables: num_words <dbl>, num_utts <dbl>, avg_utt_len <dbl>, | | | | | | | |
| # median_utt_len <dbl> | | | | | | | |

Applying the sorting in Example 6.27, we can see that the utterances are now our desired order. We have now completed the curation of the structured dataset aiming to provide a base for further analysis into the recorded utterances of speakers from the CABNC corpus.

6.2.4 Write dataset

Let's now write this dataset to disk. Again, since this is a dataset we have created, we will store the dataset in the *data/derived/* directory. To keep the CABNC separate from any other datasets that we may need for our research project, I will create a subdirectory called *cabnc/* to store the dataset. Into this directory we can write a CSV file with our `cabnc_tbl` data frame with the `write_csv()` function, as seen in Example 6.28.

Example 6.28.

Table 6.6: Data dictionary example for the curated CABNC dataset.

| variable | name | variable_type | description |
|----------------|--------------------------|---------------|---|
| doc_id | Document ID | categorical | Unique identifier for each document |
| utt_num | Utterance Number | ordinal | Sequential number assigned to each utterance |
| utt_start | Utterance Start Time | numeric | Timestamp indicating the start time of an utterance |
| utt_end | Utterance End Time | numeric | Timestamp indicating the end time of an utterance |
| utterance | Utterance Text | categorical | Textual content of an utterance |
| part_id | Participant ID | categorical | Unique identifier for each participant |
| part_age | Participant Age | numeric | Age of the participant, measured in months |
| part_sex | Participant Sex | categorical | Sex of the participant. (male/ female) |
| num_words | Number of Words | numeric | Total number of words in utterances in the document |
| num_utts | Number of Utterances | numeric | Total number of utterances in the document |
| avg_utt_len | Average Utterance Length | numeric | Average length of an utterance in the document |
| median_utt_len | Median Utterance Length | numeric | Median length of an utterance in the document |

```
# Create cabnc directory
dir_create(path = "../data/derived/cabnc")

# Write dataset to disk
cabnc_tbl |>
  write_csv(path = "../data/derived/cabnc/cabnc_curated.csv")
```

The final step is to create a data dictionary file for this dataset. We can do this using the `create_data_dictionary()` function, passing the data frame object name and a path to the directory where we want to store the data dictionary. The `create_data_dictionary()` function will create a CSV file with a data dictionary template in the specified directory.

Example 6.29.

```
# Create data dictionary
create_data_dictionary(
  data = cabnc_tbl,
  file_path = "../data/derived/cabnc_curated_dd.csv"
)
```

After running the code in Example 6.29, we can open the `cabnc_curated_dd.csv` file in the `data/derived/cabnc/` directory and fill in the data dictionary template. An example of an edited data dictionary is shown in Table 6.6.

And the final directory structure for the `data/` directory is shown in Example 6.30.

Example 6.30.

```
data/
└── analysis/
└── derived/
    ├── cabnc_curated_dd.csv
    └── cabnc/
        └── cabnc_curated.csv
└── original/
    ├── cabnc_do.csv
    └── cabnc/
        ├── participants.csv
        ├── token_types.csv
        ├── tokens.csv
        ├── transcripts.csv
        └── utterances.csv
```

6.3 Semi-structured

At this point we have discussed curating unstructured data and structured datasets. Between these two extremes falls semi-structured data. And as the name suggests, it is a hybrid between unstructured and structured data. This means that there will be important structured metadata included with unstructured elements. The file formats and approaches to encoding the structured aspects of the data vary widely from resource to resource and therefore often requires more detailed attention to the structure of the data and often includes more sophisticated programming strategies to curate the data to produce a tidy dataset.

6.3.1 Reading data

The file formats associated with semi-structured data include a wide range. These include file formats conducive to more structured-leaning data, such as XML, HTML, and JSON, and file formats with more unstructured-leaning data, such as annotated TXT files. Annotated TXT files may in fact appear with the *.txt* extension, but may also appear with other, sometimes resource-specific, extensions, such as *.utt* for the Switchboard Dialogue Act Corpus or *.cha* for the CHILDES corpus annotation files, for example.

The more structured file formats use standard conventions and therefore can be read into an R session with format-specific functions. Say, for example, we are working with data in a JSON file format, as in File 6.3. We can read the data into an R session with the `read_json()`

function from the `jsonlite` package (Ooms 2023). For XML and HTML files, the `rvest` package provides the `read_xml()` and `read_html()` functions.

Semi-structured data in TXT files can be read either as an entire file with `read_file()` or line-by-line with `read_lines()`. The choice of which approach to take depends on the structure of the data. If the data structure is line-based, then `read_lines()` often makes more sense than `read_file()`. However, in some cases, the data may be structured in a way that requires the entire file to be read into an R session and then subsequently parsed.

6.3.2 Orientation

To provide an example of the curation process using semi-structured data, we will work with the ENNTT corpus, introduced in Section 3.2.2. Let's look at the directory structure for the ENNTT corpus in Example 6.31.

Example 6.31.

```
data/
└── analysis/
└── derived/
└── original/
    └── enntt_do.csv
    └── enntt/
        ├── natives.dat
        ├── natives.tok
        ├── nonnatives.dat
        ├── nonnatives.tok
        ├── translations.dat
        └── translations.tok
```

We now inspect the data origin file for the ENNTT corpus, `enntt_do.csv`, in Table 6.7.

According to the data origin file, there are two important file types, `.dat` and `.tok`. The `.dat` files contain annotations and the `.tok` files contain the actual text. Let's inspect the first couple of lines in the `.dat` file for the native speakers, `nonnatives.dat`, in File 6.4.

We see that the `.dat` file contains annotations for various session and speaker attributes. The format of the annotations is XML-like. XML is a form of markup language, such as YAML, JSON, etc. **Markup languages** are used to annotate text with additional information about the structure, meaning, and/ or presentation of text. In XML, structure is built up by nesting of nodes. The nodes are named with tags, which are enclosed in angle brackets, `<` and `>`. Nodes are opened with `<TAG>` and closed with `</TAG>`. In Example 6.32 we see an example of a simple XML file structure.

Table 6.7: Data origin file for the ENNTT corpus.

| attribute | description |
|-------------------------|---|
| Resource name | Europarl corpus of Native, Non-native and Translated Texts - ENNTT |
| Data source | https://github.com/senisioi/enntt-release |
| Data sampling frame | English, European Parliament texts, transcribed discourse, political genre |
| Data collection date(s) | Not specified in the repository |
| Data format | .tok, .dat |
| Data schema | *.tok files contain the actual text; *.dat files contain the annotations corresponding |
| License | Not specified. Contact the authors for more information. |
| Attribution | Nisioi, S., Rabinovich, E., Dinu, L. P., & Wintner, S. (2016). A corpus of native, no |

Example 6.32.

```

<?xml version="1.0" encoding="UTF-8"?>
<book category="fiction">
  <title lang="en">The Catcher in the Rye</title>
  <author>J.D. Salinger</author>
  <year>1951</year>
</book>

```

In Example 6.32 there are four nodes, three of which are nested inside of the `<book>` node. The `<book>` node in this example is the root node. XML files require a root node. Nodes can also have attributes, such as the `category` attribute in the `<book>` node, but they are not required. Furthermore, XML files also require a declaration, which is the first line in Example 6.32. The declaration specifies the version of XML used and the encoding.

So the `.dat` file is not strict XML, but is similar in that it contains nodes and attributes. An XML variant you are likely familiar with, HTML, has more relaxed rules than XML. HTML is a markup language used to annotate text with information about the organization and presentation of text on the web that does not require a root node or a declaration –much like our `.dat` file. So suffice it to say that the `.dat` file can safely be treated as HTML.

And the `.tok` file for the native speakers, `nonnatives.tok`, in File 6.5, shows the actual text for each line in the corpus.

In a study in which we are interested in contrasting the language of natives and non-natives, we will want to combine the `.dat` and `.tok` files for these groups of speakers and then join these datasets into one.

The question is what attributes we want to include in the curated dataset. Given the research focus, we will not need the `LANGUAGE` or `NAME` attributes. We may want to modify the attribute names so they are a bit more descriptive.

An idealized version of the dataset based on this criteria is shown in Table 6.8.

Table 6.8: Idealized version of the ENNTT corpus.

| session_id | speaker_id | state | type | session_set | text |
|-------------|------------|--------|------------|-------------|---|
| ep-05-11-17 | 96779 | Poland | non-native | 184 | The Commission is following with interest the planned construction of a nuclear power plant in Akkuyu , Turkey and recognises the importance of ensuring that the construction of the new plant follows the highest internationally accepted nuclear safety standards . |
| ... | ... | ... | ... | ... | ... |

6.3.3 Tidy the data

Now that we have a better understanding of the corpus data and our target curated dataset structure, let's work to extract and organize the data from the native and non-native files into one dataset.

The general approach we will take is, for native and then non-natives, to read in the *.dat* file as an HTML file and then extract the line nodes and their attributes combining them into a data frame. Then we'll read in the *.tok* file as a text file and then combine the two into a single data frame. After producing a native and non-native data frame, we will combine them into one data frame to write and document as our curated dataset.

Starting with the natives, we read in the *.dat* file as an XML file with the `read_html()` function and then extract the line nodes with the `html_elements()` function as in Example 6.33.

Example 6.33.

```
# Load packages
library(rvest)

# Read in *.dat* file as HTML
ns_dat_lines <-
  read_html("../data/original/enntt/natives.dat") |>
  html_elements("line")

# Inspect
class(ns_dat_lines)
```

```
typeof(ns_dat_lines)
length(ns_dat_lines)
```

```
> [1] "xml_nodeset"
> [1] "list"
> [1] 116341
```

When can see that the `ns_dat_lines` object is a special type of list, `xml_nodeset` which contains 116,341 line nodes. Let's now jump out of sequence and read in the `.tok` file as a text file, in Example 6.34, again by lines using `read_lines()`, and compare the two to make sure that our approach will work.

Example 6.34.

```
# Read in *.tok* file by lines
ns_tok_lines <-
  read_lines("data/enntt/original/natives.tok")
```

```
# Inspect
class(ns_tok_lines)
typeof(ns_tok_lines)
length(ns_tok_lines)
```

```
> [1] "character"
> [1] "character"
> [1] 116341
```

We do, in fact, have the same number of lines in the `.dat` and `.tok` files. So we can proceed with extracting the attributes from the line nodes and combining them with the text from the `.tok` file.

Let's start by listing the attributes of the first line node in the `ns_dat_lines` object. We use the `html_attrs()` function to get the attribute names and the values, as in Example 6.35.

Example 6.35.

```
# List attributes of first line node
ns_dat_lines[[1]] |>
  html_attrs()
```

```

>           state          mepid      language      name
> "United Kingdom"      "2099"      "EN" "Evans, Robert J"
>   seq_speaker_id      session_id
>           "2"      "ep-00-01-17"

```

No surprise here, these are the same attributes we saw in the `.dat` file preview in File 6.4. At this point, it's good to make a plan on how to associate the attribute names with the column names in our curated dataset.

- `session_id` = `session_id`
- `speaker_id` = `MEPID`
- `state` = `state`
- `session_seq` = `seq_speaker_id`

We can do this one attribute at a time using the `html_attr()` function and then combine them into a data frame with the `tibble()` function as in Example 6.36.

Example 6.36.

```

# Extract attributes from first line node
session_id <- ns_dat_lines[[1]] |> html_attr("session_id")
speaker_id <- ns_dat_lines[[1]] |> html_attr("mepid")
state <- ns_dat_lines[[1]] |> html_attr("state")
session_seq <- ns_dat_lines[[1]] |> html_attr("seq_speaker_id")

# Combine into data frame
tibble(session_id, speaker_id, state, session_seq)

```

```

> # A tibble: 1 x 4
>   session_id speaker_id state      session_seq
>   <chr>      <chr>      <chr>      <chr>
> 1 ep-00-01-17 2099      United Kingdom 2

```

The results from Example 6.36 show that the attributes have been extracted and mapped to our idealized column names, but this would be tedious to do for each line node. A function to extract attributes and values from a line and add them to a data frame would help simplify this process. The function in Example 6.37 does just that.

Example 6.37.

```

# Function to extract attributes from line node
extract_datAttrs <- function(line_node) {
  session_id <- line_node |> html_attr("session_id")
  speaker_id <- line_node |> html_attr("mepid")
  state <- line_node |> html_attr("state")
  session_seq <- line_node |> html_attr("seq_speaker_id")

  tibble(session_id, speaker_id, state, session_seq)
}

```

It's a good idea to test out the function to verify that it works as expected. We can do this by passing the various indices to the `ns_dat_lines` object to the function as in Example 6.38.

Example 6.38.

```

# Test function
extract_datAttrs(ns_dat_lines[[1]])

```

```

> # A tibble: 1 x 4
>   session_id speaker_id state      session_seq
>   <chr>       <chr>      <chr>      <chr>
> 1 ep-00-01-17 2099     United Kingdom 2

```

```
extract_datAttrs(ns_dat_lines[[20]])
```

```

> # A tibble: 1 x 4
>   session_id speaker_id state      session_seq
>   <chr>       <chr>      <chr>      <chr>
> 1 ep-00-01-17 1309     United Kingdom 40

```

```
extract_datAttrs(ns_dat_lines[[100]])
```

```

> # A tibble: 1 x 4
>   session_id speaker_id state      session_seq
>   <chr>       <chr>      <chr>      <chr>
> 1 ep-00-01-18 4549     United Kingdom 28

```

Looks like the `extract_dat_attrs()` function is ready for prime-time. Let's now apply it to all of the line nodes in the `ns_dat_lines` object using the `map_dfr()` function from the `purrr` package as in Example 6.39.

Example 6.39.

```
# Extract attributes from all line nodes
ns_dat_attrs <-
  ns_dat_lines |>
  map_dfr(extract_dat_attrs)

# Inspect
glimpse(ns_dat_attrs)
```

```
> Rows: 116,341
> Columns: 4
> $ session_id  <chr> "ep-00-01-17", "ep-00-01-17", "ep-00-01-17", "ep-00-01-17"~
> $ speaker_id   <chr> "2099", "2099", "2099", "4548", "4548", "4541", "4541", "4~
> $ state        <chr> "United Kingdom", "United Kingdom", "United Kingdom", "Uni~
> $ session_seq  <chr> "2", "2", "2", "4", "4", "12", "12", "12", "12", "12", "12~
```

➊ Dive deeper

The `map*()` functions from the `purrr` package are a family of functions that apply a function to each element of a vector, list, or data frame. The `map_dfr()` function is a variant of the `map()` function that returns a data frame that is the result of row-binding the results, hence `_dfr`.

We can see that the `ns_dat_attrs` object is a data frame with 116,341 rows and 4 columns, just as we expected. We can now combine the `ns_dat_attrs` data frame with the `ns_tok_lines` vector to create a single data frame with the attributes and the text. This is done with the `mutate()` function assigning the `ns_tok_lines` vector to a new column named `text` as in Example 6.40.

Example 6.40.

```
# Combine attributes and text
ns_dat <-
  ns_dat_attrs |>
  mutate(text = ns_tok_lines)
```

```

# Inspect
glimpse(ns_dat)

> Rows: 116,341
> Columns: 5
> $ session_id  <chr> "ep-00-01-17", "ep-00-01-17", "ep-00-01-17", "ep-00-01-17"~
> $ speaker_id   <chr> "2099", "2099", "2099", "4548", "4548", "4541", "4541", "4~
> $ state        <chr> "United Kingdom", "United Kingdom", "United Kingdom", "Uni~
> $ session_seq  <chr> "2", "2", "2", "4", "4", "12", "12", "12", "12", "12", "12~
> $ text         <chr> "You will be aware from the press and television that ther~

```

This is the data for the native speakers. We can now repeat this process for the non-native speakers, *or* we can create a function to do it for us. Let's explore the later option.

This is the data for the native speakers. We can now repeat this process for the non-native speakers, or we can create a function to do it for us. Let's explore the later option.

I will name this function `combine_dat_tok()` and it will take two arguments: `dat_file` and `tok_file`. The `dat_file` argument will be the path to the `.dat` file and the `tok_file` argument will be the path to the `.tok` file. The function first reads both the files into R as lines. Then it extracts the attributes from the `.dat` file using the `extract_datAttrs()` function we created earlier. Finally, it combines the attributes with the text from the `.tok` file and returns a data frame.

Note that there is at least one tweak we need to make to our existing code base if we plan on combining the native and non-native data. As our function stands, it will return a data frame with the same column names for both the native and non-native data. We will need to add a column to which we will indicate what type of data it is (native or non-native). This is where the `type` column we planned earlier comes in. There are multiple ways to do this. We could add a column to the data frame before we return it, or we could add a column to the data frame after we return it. I will opt for the former and embed it inside of our function `combine_dat_tok()`.

Example 6.41.

```

# Load packages
library(rvest) # for reading/parsing HTML
library(purrr) # for mapping functions
library(fs) # for working with files

# Function to combine *.dat* and *.tok* files
combine_dat_tok <- function(dat_file, tok_file) {

```

```

# Read in files by lines
dat <-
  read_html(dat_file) |>
  html_nodes("line")
tok <- read_lines(tok_file)

# Create function to extract attributes
extract_datAttrs <- function(line_node) {
  session_id <- line_node |> html_attr("session_id")
  speaker_id <- line_node |> html_attr("mepid")
  state <- line_node |> html_attr("state")
  session_seq <- line_node |> html_attr("seq_speaker_id")

  tibble(session_id, speaker_id, state, session_seq)
}

# Apply function to all line nodes
datAttrs <-
  dat |>
  map_dfr(extract_datAttrs)

# Combine attrs and text into data frame
dataset <-
  datAttrs |>
  bind_cols(text = tok)

# Add type column with value of file name (w/o extension)
dataset <-
  dataset |>
  mutate(type = path_file(dat_file) |> path_ext_remove())

# Return data frame
return(dataset)
}

```

Apply the `combine_dat Tok()` function to read in the data for the native speakers and the non-native speakers in a few lines of code, as in Example 6.42.

Example 6.42.

```

# Native speakers
ns_dat <-
  combine_dat_tok(
    dat_file = "../data/original/enntt/natives.dat",
    tok_file = "../data/original/enntt/natives.tok"
  )
# Preview
glimpse(ns_dat)

# Non-native speakers
nns_dat <-
  combine_dat_tok(
    dat_file = "../data/original/enntt/nonnatives.dat",
    tok_file = "../data/original/enntt/nonnatives.tok"
  )
# Preview
glimpse(nns_dat)

```

```

> Rows: 116,341
> Columns: 6
> $ session_id  <chr> "ep-00-01-17", "ep-00-01-17", "ep-00-01-17", "ep-00-01-17"~
> $ speaker_id   <chr> "2099", "2099", "2099", "4548", "4548", "4541", "4541", "4~
> $ state        <chr> "United Kingdom", "United Kingdom", "United Kingdom", "Uni~
> $ session_seq  <chr> "2", "2", "2", "4", "4", "12", "12", "12", "12", "12", "12~
> $ text          <chr> "You will be aware from the press and television that ther~
> $ type          <chr> "natives", "natives", "natives", "natives", "natives", "na~

> Rows: 29,734
> Columns: 6
> $ session_id  <chr> "ep-00-01-18", "ep-00-01-18", "ep-00-01-18", "ep-00-01-18"~
> $ speaker_id   <chr> "653", "653", "653", "653", "653", "653", "653", "6~
> $ state        <chr> "Belgium", "Belgium", "Belgium", "Belgium", "Belgium", "Be~
> $ session_seq  <chr> "157", "157", "157", "157", "157", "157", "157", "157", "1~
> $ text          <chr> "The Commission is following with interest the planned con~
> $ type          <chr> "nonnatives", "nonnatives", "nonnatives", "nonnatives", "n~

```

The last step to curate the dataset is to combine the native and non-native data into a single data frame. We can do this using the `bind_rows()` function from the `dplyr` package, as essentially we are just stacking the rows from each data frame on top of each other.

```
# Combine native and non-native data
enntt_tbl <-
  bind_rows(ns_dat, nns_dat)
```

The `enntt_tbl` data frame is the curated dataset. We can perform some data checks to make everything looks good and then proceed to writing and documenting the dataset.

💡 Consider this

What data checks could we perform to ensure that the data is in the format we expect? What are some of the things we should be looking for?

6.3.4 Write dataset

As we have done for the other curated datasets, we will write our curated dataset to a `.csv` file. We will use the `write_csv()` function and name the file appropriately, as in Example 6.43.

Example 6.43.

```
# Create directory
dir_create("../data/derived/enntt")

# Write dataset to *.csv* file
write_csv(enntt_tbl, "../data/derived/enntt/enntt_curated.csv")
```

Document the dataset in a data dictionary using the `create_data_dictionary()` function from the `qtalrkit` package, as in Example 6.44.

Example 6.44.

```
# Load package
library(qtalrkit)

# Create data dictionary
create_data_dictionary(
  data = enntt_tbl,
  file_path = "../data/derived/enntt_curated_dd.csv"
)
```

Table 6.9: Data dictionary for the curated ENNTT dataset

| variable | name | variable_type | description |
|-------------|------------------|---------------|---|
| session_id | Session ID | categorical | Unique identifier for each session |
| speaker_id | Speaker ID | categorical | Unique identifier for each speaker |
| state | State | categorical | Name of the state or country the session is linked to |
| session_seq | Session Sequence | ordinal | Sequence number in the session |
| text | Text | categorical | Text transcript of the session |
| type | Type | categorical | The type of the speaker, whether native or nonnative |

After editing the data dictionary file to include the appropriate information, we will have something similar to Table 6.9.

The final step is to clean up and comment the code added to the *2-curate-datasets.qmd* file. The final directory structure is seen in Example 6.45.

Example 6.45.

```

project/
  └── code/
    |   └── 1-acquire-data.qmd
    |   └── 2-curate-data.qmd
    |   └── ...
  └── data/
    |   └── analysis/
    |   └── derived/
    |       |   └── enntt_curated_dd.csv
    |       └── enntt/
    |           └── enntt_curated.csv
    └── original/
        |   └── enntt_do.csv
        └── enntt/
            └── natives.dat
            └── natives.tok
            └── nonnatives.dat
            └── nonnatives.tok
            └── translations.dat
            └── translations.tok
  └── output/
    |   └── figures/
    |   └── reports/
    |   └── results/

```

```
|   └── tables/
|   └── README.md
└── _main.R
```

Summary

In this chapter we looked at the process of structuring data into a dataset. This included a discussion on three main types of data –unstructured, structured, and semi-structured. The level of structure of the original data(set) will vary from resource to resource and by the same token so will the file format used to support the level of metadata included. The results from data curation results in a dataset that is saved separate from the original data to maintain modularity between what the data(set) looked like before we intervene and afterwards. Since there can be multiple analysis approaches applied the original data in a research project, this curated dataset serves as the point of departure for each of the subsequent datasets derived from the transformational steps. In addition to the code we use to derived the curated dataset's structure, we also include a data dictionary which documents the curated dataset.

Activities

-  Add description of outcomes

Recipe

What: Organizing and documenting datasets^a

How: Read Recipe 6 and participate in the Hypothes.is online social annotation.

Why: To rehearse methods for deriving tidying datasets to use a the base for further project-specific purposes. We will explore how regular expressions are helpful in developing strategies for matching, extracting, and/ or replacing patterns in character sequences and how to change the dimensions of a dataset to either expand or collapse columns or rows.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-6.html>

Lab

What: Pattern Matching and Manipulate Datasets^a

How: Clone, fork, and complete the steps in Lab 6.

Why: To gain experience working with coding strategies reshaping data using tidyverse functions and regular expressions, to practice reading/ writing data from/ to disk, and

to implement organizational strategies for organizing and documenting a dataset in reproducible fashion.

^a<https://github.com/qtalr/lab-6>

Questions

Conceptual questions

1. ...
2. ...

Technical questions

1. ...
2. ...

File 6.3 data.json: Example JSON file

```
{  
  "data": [  
    {"speaker": "Alice",  
     "age": 27,  
     "sex": "Female",  
     "word_form": "running",  
     "lemma_form": "run"},  
    {"speaker": "Bob",  
     "age": 35,  
     "sex": "Male",  
     "word_form": "jumped",  
     "lemma_form": "jump"},  
    {"speaker": "Charlie",  
     "age": 42,  
     "sex": "Male",  
     "word_form": "singing",  
     "lemma_form": "sing"},  
    {"speaker": "Diane",  
     "age": 32,  
     "sex": "Female",  
     "word_form": "walked",  
     "lemma_form": "walk"}  
  ]  
}
```

File 6.4 `../data/original/enntt/nonnatives.dat`: Example `.dat` file for the non-native speakers.

```
<LINE STATE="Poland" MEPID="96779" LANGUAGE="EN" NAME="Danuta Hübner,"  
  ↵ SEQ_SPEAKER_ID="184" SESSION_ID="ep-05-11-17"/>  
<LINE STATE="Poland" MEPID="96779" LANGUAGE="EN" NAME="Danuta Hübner,"  
  ↵ SEQ_SPEAKER_ID="184" SESSION_ID="ep-05-11-17"/>
```

File 6.5 `../data/original/enntt/nonnatives.tok`: Example `.tok` file for the non-native speakers.

The Commission is following with interest the planned construction of a nuclear
power plant in Akkuyu , Turkey and recognises the importance of ensuring
that the construction of the new plant follows the highest internationally
accepted nuclear safety standards .

According to our information , the decision on the selection of a bidder has not
been taken yet .

7 Transform datasets



Draft

Ready for review.

Nothing is lost. Everything is transformed.

— Michael Ende, The Neverending Story



☞ to update the learning outcomes

- Understand the role of data transformation in a text analysis project.
- Identify the main types of transformations used to prepare datasets for analysis.
- Recognize the importance of planning and documenting the transformation process.

In this chapter, we will focus on transforming a curated dataset to refine and possibly expand its relational characteristics to align with our research. I will approach the transformation process by breaking it down into five subcategories: text normalization, variable recoding, text tokenization, variable generation, and observation/ variable merging. These categories are not sequential but may occur in any order based on the researcher's evaluation of the dataset characteristics and the desired outcome.



What: Reshape dataset rows, Reshape dataset columns^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: ... ☞

^a<https://github.com/qtalr/lessons>

7.1 Normalization

The process of normalizing datasets in essence is to sanitize the values of variable or set of variables such that there are no artifacts that will contaminate subsequent processing. It

Table 7.1: Data dictionary for the Europarl Corpus.

| variable | name | variable_type | description |
|----------|---------------|---------------|---|
| doc_id | Document ID | numeric | Unique identification number for each document |
| type | Document Type | categorical | Type of document; either 'Source' (Spanish) or 'Target' (English) |
| line_id | Line ID | numeric | Unique identification number for each line in each document type |
| lines | Lines | categorical | Content of the lines in the document |

may be the case that non-linguistic metadata may require normalization but more often than not linguistic information is the most common target for normalization as text often includes artifacts from the acquisition process which will not be desired in the analysis.

7.1.1 Orientation

To explore some of the strategies of normalization, we will look at the Europarl Corpus. As we are working towards transforming a curated dataset, we will start by getting oriented to the dataset. Following the reproducible research principles, I will assume that the curated dataset and its associated data dictionary are in the *data/derived/* directory, as seen in Example 7.1.

Example 7.1.

```
data/
└── analysis/
└── derived/
    ├── europarl_curated_dd.csv
    └── europarl/
        └── europarl_curated.csv
└── original/
```

The contents of the data dictionary for this dataset appears in Table 7.1.

This dataset contains transcribed source language (Spanish) and translated target language (English) from the proceedings of the European Parliament. The unit of observation is the `lines` variable whose values are lines of dialog.

Let's read in the dataset CSV file with `read_csv()` and inspect the first lines of the dataset with `slice_head()` in Example 7.2.

Example 7.2.

```

# Read in the dataset
europarl_curated_tbl <-
  read_csv(file = "../data/derived/europarl_curated.csv")

# Preview the first 10 lines
europarl_curated_tbl |>
  slice_head(n = 10)

```

```

> # A tibble: 10 x 4
>   doc_id type  line_id lines
>   <dbl> <chr> <dbl> <chr>
> 1 1965735 Source     1 "Reanudación del periodo de sesiones"
> 2     1 Target      1 "Resumption of the session"
> 3 1965736 Source     2 "Declaro reanudado el periodo de sesiones del Parlame~"
> 4     2 Target      2 "I declare resumed the session of the European Parlia~"
> 5 1965737 Source     3 "Como todos han podido comprobar, el gran \'efecto de~"
> 6     3 Target      3 "Although, as you will have seen, the dreaded 'millen~"
> 7 1965738 Source     4 "Sus Señorías han solicitado un debate sobre el tema ~"
> 8     4 Target      4 "You have requested a debate on this subject in the c~"
> 9 1965739 Source     5 "A la espera de que se produzca, de acuerdo con mucho~"
> 10    5 Target      5 "In the meantime, I should like to observe a minute' ~"

```

Simply looking at the first 10 lines of this dataset gives us a clearer sense of the dataset structure, but, in terms of normalization procedures we might apply, it is likely not sufficient. We want to get a sense of any potential inconsistencies in the dataset, in particular in the `lines` variable. Since this is a large dataset with 3931468 observations, we will need to explore the dataset in manageable chunks. The `slice_sample()` function will allow us to randomly sample a subset of the dataset of a certain number of observations specified by the `n =` argument, as seen in Example 7.3.

Example 7.3.

```

# Randomly sample 5 observations
europarl_curated_tbl |>
  slice_sample(n = 5)

> # A tibble: 5 x 4
>   doc_id type  line_id lines
>   <dbl> <chr> <dbl> <chr>
> 1 1979130 Source     13396 Nosotros no apostamos aquí, en este Parlamento, por lo~

```

```

> 2 3322228 Source 1356494 (SK) Señor Presidente, me he abstenido de votar porque~  

> 3 140652 Target 140652 It is these last two, however, together with Finland, ~  

> 4 2145661 Source 179927 Son las mujeres de esos mismos países: Egipto, Somalia~  

> 5 2269815 Source 304081 Permitanme añadir que, en nuestro último llamamiento p~
```

We should run the code in Example 7.3 multiple times to get a sense of the variation in the dataset.

Tip

R functions which return samples (*e.g.* `slice_sample()`) are generated using pseudo-random number generators. These generators are initialized with a seed value. You can control the seed value R uses to generate the random numbers by using the `set.seed()` function and setting the seed value to a number of your choice. It is important to note that setting a seed only affects the subsequent random number generation.

Using `set.seed()` is useful when you want to ensure that the same random numbers are generated each time you run the code. This is particularly helpful when you want to reproduce results in a report or other document.

In the case of the Europarl corpus dataset, it may be useful to see the source and target lines in the same sample. Do do this, we can first sample from the `line_id` variable and then filter the `europarl_curated_tbl` with the `filter()` function and the `%in%` operator to select the lines that match the sampled `line_id` values, as seen in Example 7.4.

Example 7.4.

```

# Randomly sample 5 line_id values
line_id_sample_vec <-
  europarl_curated_tbl |>
  distinct(line_id) |>
  slice_sample(n = 5) |>
  pull(line_id)

# Select the lines that match the sampled line_id values
europarl_curated_tbl |>
  filter(line_id %in% line_id_sample_vec)
```

```

> # A tibble: 10 x 4
>   doc_id type   line_id lines
>   <dbl> <chr>   <dbl> <chr>
> 1 1992525 Source   26791 La sociedad del conocimiento es algo más, y también p~
> 2 26791 Target    26791 The knowledge society is more than that, and it may a~
```

```

> 3 2247038 Source 281304 Especialmente en la industria de alta tecnología pued-
> 4 281304 Target 281304 Particularly in the technical and hi-tech industry, t-
> 5 2325587 Source 359853 - (IT) Señora Presidenta, Comisario, Señorías, quiero-
> 6 359853 Target 359853 . (IT) Madam President, Commissioner, ladies and gent-
> 7 2573895 Source 608161 En consecuencia, nuestro Grupo ha votado en contra.
> 8 608161 Target 608161 Consequently, our group voted against.
> 9 2581569 Source 615835 Al margen de ello, es todo el club de los seis, esos ~
> 10 615835 Target 615835 Beyond that, it is the whole club of six, these count-

```

After running the code in Example 7.3 and Example 7.4 multiple times, I identified a number of artifacts that we will want to consider addressing. These are included in Table 7.2.

Table 7.2: Characteristics of the Europarl Corpus dataset that may require normalization.

| Description | Examples | Concern |
|--------------------------|---|--|
| Non-speech annotations | (Abucheos), (A4-0247/98),
(The sitting was opened at
09:00) | Not of interest for our
analysis |
| Inconsistent whitespace | 5 % , , , , Palacio' s | May be problematic for
tokenization |
| Non-sentence punctuation | - | May be problematic for
tokenization |
| Abbreviations | Mr., Sr., Mme., Mr, Sr, Mme,
Mister, Señor, Madam | May be problematic for
tokenization |
| Text case | The, the, White, white | May be problematic for
tokenization |

The first three items in Table 7.2 are relatively straightforward. Non-speech annotations most likely are not relevant for our research. Inconsistent whitespace and non-sentence punctuation may be problematic for tokenization that depends on whitespace and punctuation regularities.

The other two considerations are more contingent on our research aims. The existence of various forms for the same word, abbreviated and unabridged, introduces variability may not be of interest for our analysis. Secondly, common conventions for capitalization in prose can introduce unwanted variability. If we leave the text as is, the tokens `The` and `the` will be treated as distinct. If we convert the text to lowercase, the tokens `White` and `white` will be treated as the same, even if `White` corresponds to a proper noun (*e.g. White House*).

These observations provide us a roadmap for the normalization process. For demonstration, let's focus only on a couple of these cases: removing parliamentary session descriptions and extra whitespace.

7.1.2 Application

Identifying our normalization goals is an important first step. The next step is to identify the procedures that will accomplish these goals. The majority of text normalization procedures can be accomplished with the `stringr` package (Wickham 2022). This package provides a number of functions for manipulating text. The workhorse functions we will use for our tasks are the `str_remove()` and `str_replace()` functions. As these functions give us the ability to remove or replace text. Our task is to identify the patterns we want to remove or replace.

Before we modify any lines, let's try craft a search pattern to identify the text of interest. This is done to avoid over- or under-generalizing the search pattern. If we are too general, we may end up removing or replacing text that we want to keep. If we are too specific, we may not remove or replace all the text we want to remove or replace.

Let's start by identifying non-parliamentary speech. Two functions from the `stringr` package come in handy here: `str_detect()` and `str_extract()`. `str_detect()` detects a pattern in a character vector and returns a logical vector, `TRUE` if the pattern is detected and `FALSE` if it is not. `str_extract()` extracts the text in a character vector that matches a pattern.

`str_detect()` pairs well with the `filter()` function to return observations that match a pattern in a character vector. `str_extract()` pairs well with the `mutate()` function to create a new variable which contains character vector that match a pattern.

Let's start with the `str_detect()` function. We will use this function to identify the lines that contain the parliamentary session descriptions. From the examples above, we can see that these instances are wrapped with parentheses (and). The text within the parentheses can vary, so we need a Regular Expression to do the heavy lifting. To start out we can match any one or multiple characters with `.+`. But it is important to recognize the `+` (and also the `*`) operators are 'greedy', meaning that if there are multiple matches, the longest match will be returned. In this case, we want to match the shortest match. To do this we can use the `?` operator to make the `+` operator 'lazy'. This will match the shortest match.

Our test code appears in Example 7.5.

Example 7.5.

```
# Identify non-speech lines
europarl_curated_tbl |>
  filter(str_detect(lines, "\\(.+?\\)")) |>
  slice_sample(n = 10)

> # A tibble: 10 x 4
>   doc_id type  line_id lines
>   <dbl> <chr> <dbl> <chr>
```

```

> 1 3225772 Source 1260038 (PT) Señor Presidente, quisiera plantear dos pregunta~  

> 2 3715842 Source 1750108 (El Parlamento decide la devolución a la Comisión)  

> 3 1961715 Target 1961715 (Parliament adopted the resolution)  

> 4 1429470 Target 1429470 27, originally Greens/EFA amendment in FEMM); binding~  

> 5 51632 Target 51632 Question No 8 by (H-0376/00):  

> 6 2482671 Source 516937 La Comisión propone proporcionar a las Agencias nacio~  

> 7 1059628 Target 1059628 (The President cut off the speaker)  

> 8 1507254 Target 1507254 in writing. - (LT) I welcomed this document, because ~  

> 9 2765325 Source 799591 (Aplausos)  

> 10 2668536 Source 702802 Las preguntas que, por falta de tiempo, no han rec~
```

The results from Example 7.5 show that we have identified the lines that contain at least one of the parliamentary session description annotations. A more targeted search to identify specific instances of the parliamentary session descriptions can be accomplished adding the `str_extract()` function as seen in Example 7.6.

Example 7.6.

```

# Extract non-speech fragments
europarl_curated_tbl |>
  filter(str_detect(lines, "\\(.+?\\)")) |>
  mutate(non_speech = str_extract(lines, "\\(.+?\\)")) |>
  slice_sample(n = 10)

> # A tibble: 10 x 5
>   doc_id type  line_id lines          non_speech
>   <dbl> <chr> <dbl> <chr>          <chr>
> 1 3225772 Source 1260038 (PT) Señor Presidente, quisiera plantear d~ (PT)
> 2 3715842 Source 1750108 (El Parlamento decide la devolución a la C~ (El Parla~
> 3 1961715 Target 1961715 (Parliament adopted the resolution) (Parliame~
> 4 1429470 Target 1429470 27, originally Greens/EFA amendment in FEM~ (para. 53)
> 5 51632 Target 51632 Question No 8 by (H-0376/00): (H-0376/0~
> 6 2482671 Source 516937 La Comisión propone proporcionar a las Age~ (correspo~
> 7 1059628 Target 1059628 (The President cut off the speaker) (The Pres~
> 8 1507254 Target 1507254 in writing. - (LT) I welcomed this documen~ (LT)
> 9 2765325 Source 799591 (Aplausos) (Aplausos)
> 10 2668536 Source 702802 Las preguntas que, por falta de tiempo,~ (Véase el~
```

The results from Example 7.6 show that we have identified the lines that contain parliamentary session description annotations and extracted this text –or have we? What if a given line contains more than one parliamentary session description annotation? It turns

out that `str_extract()` only returns the first match. To return all matches we can use the `str_extract_all()` function. Let's try again, in Example 7.7.

Example 7.7.

```
# Extract non-speech fragments
europarl_curated_tbl |>
  filter(str_detect(lines, "\\(.+?\\)")) |>
  mutate(non_speech = str_extract_all(lines, "\\(.+?\\)")) |>
  slice_sample(n = 10)

> # A tibble: 10 × 5
>   doc_id type  line_id lines          non_speech
>   <dbl> <chr> <dbl> <chr>          <list>
> 1 3225772 Source 1260038 (PT) Señor Presidente, quisiera plantear d~ <chr [1]>
> 2 3715842 Source 1750108 (El Parlamento decide la devolución a la C~ <chr [1]>
> 3 1961715 Target 1961715 (Parliament adopted the resolution) <chr [1]>
> 4 1429470 Target 1429470 27, originally Greens/EFA amendment in FEM~ <chr [1]>
> 5 51632 Target 51632 Question No 8 by (H-0376/00): <chr [1]>
> 6 2482671 Source 516937 La Comisión propone proporcionar a las Age~ <chr [2]>
> 7 1059628 Target 1059628 (The President cut off the speaker) <chr [1]>
> 8 1507254 Target 1507254 in writing. - (LT) I welcomed this documen~ <chr [1]>
> 9 2765325 Source 799591 (Aplausos) <chr [1]>
> 10 2668536 Source 702802 Las preguntas que, por falta de tiempo,~ <chr [1]>
```

OK, that might not be what you expected. The `str_extract_all()` function returns a list of character vectors. This is because for any given line in `lines` there may be a different number of matches. To maintain the data frame as rectangular, a list is returned for each value of `non_speech`. We could expand the list into a data frame with the `unnest()` function from the `tidyverse` package if our goal were to work with these matches. But that is not our aim. Rather, we want to know if we have multiple matches per line. Note that the information provided for the `non_speech` column by the tibble object tells us that we have some lines with multiple matches, as we can see in line 6 of our small sample. So good thing we checked!

Let's now remove these parliamentary session description annotations from each line in the `lines` column. We turn to `str_remove_all()`, a variant of `str_remove()`, that, as you expect, will remove multiple matches in a single line. We will use the `mutate()` function to overwrite the `lines` column with the modified text. The code is seen in Example 7.8.

Example 7.8.

```
# Remove non-speech fragments
europarl_curated_tbl <-
  europarl_curated_tbl |>
  mutate(lines = str_remove_all(lines, "\\\(.+?\\\)"))
```

I recommend spot checking the results of this normalization step by running the code in Example 7.5 again, if nothing appears we've done our job.

When you are content with the results, drop the observations that have no text in the `lines` column given the entire line was non-speech. This can be done with the `is.na()` function and the `filter()` function as seen in Example 7.9.

Example 7.9.

```
# Drop empty lines
europarl_curated_tbl <-
  europarl_curated_tbl |>
  filter(!is.na(lines))
```

The second item of business to address is the extra whitespace we observed in Table 7.2. If we consider the extra whitespace cases, we can categorize them into two types: multiple spaces that should be a single space (*e.g.*) and single spaces that occur within a word (*e.g.* 5 % , or Palacio' s).

To deal with multiple spaces, we can turn to the `str_replace_all()` function. This function will replace a pattern with a replacement string for every pattern match. In this case, we want to replace multiple spaces with a single space. We can use the `\s+` pattern to match one or more spaces and then replace it with `\s` or a single whitespace character " ".

Before we apply this normalization step, let's assess how many instances of multiple spaces we have in the dataset. We can use the `str_count()` function to count the number of matches for a pattern. The pattern we want needs to be a bit more precise than `\s+`, because this matches one or more. We want to match *two* or more. Using the regular expression operator `{,}` we can specify our pattern to be `\s{2,}`, *i.e.* two or more contiguous whitespaces. Let's count and sum all these matches. The code is seen in Example 7.10.

Example 7.10.

```
# Count multiple spaces
europarl_curated_tbl |>
```

```

  mutate(multiple_spaces = str_count(lines, "\\s{2,}")) |>
  summarize(total_multiple_spaces = sum(multiple_spaces, na.rm = TRUE))

> # A tibble: 1 × 1
>   total_multiple_spaces
>   <int>
> 1 130628

```

⚠ Warning

You may be wondering what the extra parameter `na.rm = TRUE` is doing in the `sum()` function. This parameter tells R to ignore values that are `NA` (not available). This is important because if we don't ignore `NA` values, the `sum()` function will return `NA` if there are any `NA` values in the vector. The code in Example 7.10 will return `NA` when the `\\s{2,}` doesn't match for a given line. This is because the `str_count()` function returns `NA` when there are no matches. If we don't ignore these `NA` values, the `sum()` function will return `NA` for the entire dataset.

The results from Example 7.10 show that we have over 130k instances of multiple spaces. Let's replace these with a single space. The code is seen in Example 7.11.

Example 7.11.

```

# Remove multiple spaces
europarl_curated_tbl <-
  europarl_curated_tbl |>
  mutate(lines = str_replace_all(lines, "\\s{2,}", "\\s"))

```

To check our work, we can run the code in Example 7.10 again. We should see that there are no more instances of multiple spaces.

Now let's turn to the single spaces that occur within a word. We can use the `str_replace_all()` function again to replace these single spaces with no space –but is that what we want? Probably not, we want *most* of the single spaces to remain, otherwise we would have one very, very long string on each line.

Instead, we want to narrow our scope and focus in on whitespace that occurs in particular contexts. One context we can focus on is the single quote ', as in `Palacio'`. In this use, the single quote is an apostrophe in a possessive form, but we might want to see if other forms, such as contractions, that also may have this extra whitespace. Let's check using a regular expression that matches a character string `\\w+` with single quote ' at the end followed by whitespace `\\s`. To give some more context I will add a character string `\\w+` after the

whitespace. Using the `str_extract_all()` function we can extract all the matches for this pattern. It returns a list, so we `unnest()` the list. Then we can pull the vector of matches and list the unique matches to get a sense of the context we are working with. The code is seen in Example 7.12.

Example 7.12.

```
# Match apostrophe followed by whitespace
europarl_curated_tbl |>
  slice_sample(n = 1000) |>
  mutate(quote_whitespace = str_extract_all(lines, "\w+'\\s\\w+")) |>
  unnest(quote_whitespace) |>
  pull(quote_whitespace) |>
  unique()

> [1] "Frontiers' regulation" "capital' s"           "People' s"
> [4] "communitarization' of" "years' standing"      "citizens' initiative"
> [7] "EU' s"                      "expulsion' of"      "Members' assistants"
> [10] "Europe' s"                  "Commission' s"      "Parliament' s"
> [13] "Mugabe' s"
```

In Example 7.12 we can see that we have many singular possessive forms we want to amend and other forms that we may want to keep –in particular when the single quote is part of a quote or a plural or irregular singular possessive. From various runs of this code, it looks safe to remove whitespace between the single quote and the `s` in the possessive form. We need to be careful not to remove whitespace in other contexts where the single quote is followed by `s`. To do this we can use the word boundary pattern `\b` after the `s` in our pattern to ensure that the `s` is not part of a following word. The code is seen in Example 7.13.

Example 7.13.

```
# Remove whitespace after apostrophe
europarl_transformed_tbl <-
  europarl_curated_tbl |>
  mutate(lines = str_replace_all(lines, "'\\s\\b", "'s"))
```

To check our work, we can run the code in Example 7.12 again. We should see that there are no more instances of whitespace after the single quote in possessive forms. Once we are satisfied with our work, we can continue with subsequent transformation procedures using the

Table 7.3: Data dictionary for the Switchboard dialog Act Corpus.

| variable | name | description |
|------------|----------------|---|
| doc_id | Document ID | Unique identifier for each document |
| speaker_id | Speaker ID | Unique identifier for each speaker |
| sex | Sex | Gender of the speaker |
| education | Education | Level of education of the speaker (1 = no high school, 2 = high school degree, 3 = college degree, 4 = post-graduate) |
| birth_year | Birth Year | Year of birth of the speaker |
| utt_id | Utterance ID | Unique identifier for each utterance |
| utt_text | Utterance Text | Text of the utterance |
| damsl_tag | DAMSL Tag | Tag indicating the communicative function of the utterance |

`europarl_transformed_tbl` data, or save it to a CSV file, create a data dictionary, and move on to the next transformation procedure.

Normalization goals will vary from dataset to dataset but the procedures often follow a similar line of attack to those outlined in this section. There are cases, however, in which normalization procedures are more easily accomplished after subsequent transformation steps or need to be post-poned to further the goals of other transformation steps. For example, standardizing abbreviated forms may be more easily accomplished after tokenization when each token is a word. Another example is the case of case conversion. Even if we are not directly interested in the case differences between words, certain generation procedures, Named Entity Recognition (NER) for example, may use case information to identify the names of people, locations, organizations, *etc.*. In these cases, it may be better to leave the case as is until after the generation step.

7.2 Recoding

Normalizing text can be seen as an extension of dataset curation to some extent in that the structure of the dataset is maintained. In the Europarl case, we saw this to be true. In the case of recoding, and other transformational steps, the aim will be to modify the dataset structure either by rows, columns, or both. Recoding processes can be characterized by the creation of structural changes which are derived from values in variables effectively recasting values as new variables to enable more direct access in our analyses.

7.2.1 Orientation

The Switchboard dialog Act Corpus corpus contains a number of variables describing conversations between speakers of American English. A subset of this dataset provides a good example of the recoding process. The data dictionary for this dataset appears in Table 7.3.

The data dictionary gives us a sense of the variables in the dataset. Let's read in the dataset and preview the first 10 lines to get a sense of the values in the dataset, as in Example 7.14.

Example 7.14.

```
# Read in the dataset
swda_curated_tbl <-
  read_csv("data/derived/swda/swda_curated.csv")

# Preview the first 10 lines
swda_curated_tbl |>
  slice_head(n = 10)
```



```
> # A tibble: 10 × 8
>   doc_id speaker_id sex     education birth_year utt_id utt_text           damsl_tag
>   <dbl>     <dbl> <chr>     <dbl>     <dbl> <dbl> <chr>           <chr>
> 1 4325      1632 Female     2       1962      1 Okay. /          o
> 2 4325      1632 Female     2       1962      2 {D So, }        qw
> 3 4325      1519 Female     1       1971      3 [ [ I guess, + qy^d
> 4 4325      1632 Female     2       1962      4 What kind of ~ +
> 5 4325      1519 Female     1       1971      5 I think, ] + ~ +
> 6 4325      1632 Female     2       1962      6 Does it say s~ qy
> 7 4325      1519 Female     1       1971      7 I think it us~ sd
> 8 4325      1519 Female     1       1971      8 You might try~ ad
> 9 4325      1519 Female     1       1971      9 I don't know,~ h
> 10 4325     1519 Female    1       1971     10 hold it down ~ ad
```

Considering the data dictionary and the preview of the `swda_curated_tbl` dataset, we observe a number of metadata variables, such as `doc_id`, `speaker_id`, `sex`, `education`, `birth_year`, `utt_id`, `utt_text`, and `damsl_tag`.

Most of these variables and their values are readily interpretable. However, the `damsl_tag` variable and the annotation scheme that appears interleaved with the dialog in `utt_text` may require a bit more explanation. If we consult the data origin file and/ or the corpus website, we see that the `damsl_tag` is a utterance-level annotation which indicates the dialog act type (*e.g.* statement, question, backchannel, *etc.*). The annotation interleaved with the dialog in `utt_text` is a disfluency annotation scheme¹. This scheme includes annotation for non-sentence elements such as filled pauses (*e.g.* {F uh}), discourse markers (*e.g.* {D well}), repetitions/ restarts (*e.g.* [I think + I believe]), among others.

¹<https://staff.fnwi.uva.nl/r.fernandezrovira/teaching/DM-materials/DFL-book.pdf>

Let's assume that we are interested in understanding the use of filled pauses in the Switchboard dialog. Tottie (2011) investigates the relationship between speakers' use of filled pauses `uh` and `um` and their socio-demographic background (sex, socio-economic status, and age) in British English. An American English comparison would be insightful. To do this, however, we will need to recode some of the variables in the dataset.

In this case, we will use `education` as a proxy for socio-economic status. As is, the values are numeric and are not maximally transparent. We can recode these values to be more interpretable. In the corpus documentation, the values for `education` are described in Table 7.4.

Table 7.4: Values for the `education` variable in the Switchboard dialog Act Corpus.

| Value | Description |
|-------|-----------------------|
| 0 | Less Than High School |
| 1 | Less Than College |
| 2 | College |
| 3 | More Than College |
| 9 | Unknown |

To derive a variable which reflects the age of each speaker, we will use the `birth_year` variable. This variable is a numeric value which indicates the year of birth for each speaker. We can derive a new variable `age` by subtracting the `birth_year` from the year of recordings, 1992.

Together `sex`, `education`, and `age` will provide us with the socio-demographic information we need to investigate the relationship between speakers' use of filled pauses and their socio-demographic background. The last component we need to derive is the use of filled pauses. To do this we will need to extract the filled pauses from the `utt_text` variable. We also need to consider what we mean by 'use'. In this case, we will operationalize the use of filled pauses as the number of times a filled pause is used per utterance.

7.2.2 Application

The plan to transform the `swda_curated_tbl` dataset is established. Now we need to implement the plan. We will start by recoding the `education` variable. Specifically, we want to map the numeric values to the descriptions in Table 7.4.

To do this we will use the `case_when()` function from the `dplyr` package. This function allows us to specify a series of conditions and the values to return if the condition is met. `case_when()` evaluates the conditions and `mutate()` writes the variable, in this case overwrites it, as seen in Example 7.15.

Example 7.15.

```

# Recode education
swda_curated_tbl <-
  swda_curated_tbl |>
  mutate(
    education = case_when(
      education == 0 ~ "Less Than High School",
      education == 1 ~ "Less Than College",
      education == 2 ~ "College",
      education == 3 ~ "More Than College",
      education == 9 ~ "Unknown"
    )
  )
# Preview the first 10 lines
swda_curated_tbl |>
  slice_head(n = 10)

```

```

> # A tibble: 10 x 8
>   doc_id speaker_id sex     education      birth_year utt_id utt_text damsl_tag
>   <dbl>     <dbl> <chr>   <chr>          <dbl>   <dbl> <chr>   <chr>
> 1 4325      1632 Female College      1962      1 Okay. / o
> 2 4325      1632 Female College      1962      2 {D So, } qw
> 3 4325      1519 Female Less Than Coll~ 1971      3 [ [ I g~ qy^d
> 4 4325      1632 Female College      1962      4 What ki~ +
> 5 4325      1519 Female Less Than Coll~ 1971      5 I think~ +
> 6 4325      1632 Female College      1962      6 Does it~ qy
> 7 4325      1519 Female Less Than Coll~ 1971      7 I think~ sd
> 8 4325      1519 Female Less Than Coll~ 1971      8 You mig~ ad
> 9 4325      1519 Female Less Than Coll~ 1971      9 I don't~ h
> 10 4325     1519 Female Less Than Coll~ 1971     10 hold it~ ad

```

To create the `age` variable, all we need to do is subtract the `birth_year` from the year of recording, 1992. Again we will use `mutate()` to create the `age` variable. The values are created by a subtraction operation. Since we will not need the `birth_year` variable afterwards, we will drop it from the dataset. The code is seen in Example 7.16.

Example 7.16.

```

# Recode age
swda_curated_tbl <-
  swda_curated_tbl |>

```

```

  mutate(age = 1992 - birth_year) |>
  select(-birth_year)

# Preview the first 10 lines
swda_curated_tbl |>
  slice_head(n = 10)

> # A tibble: 10 x 8
>   doc_id speaker_id sex   education      utt_id utt_text    damsl_tag  age
>   <dbl>      <dbl> <chr> <chr>      <dbl> <chr>      <chr>    <dbl>
> 1 4325      1632 Female College      1 Okay. /      o      30
> 2 4325      1632 Female College      2 {D So, }    qw      30
> 3 4325      1519 Female Less Than College 3 [ [ I gues~ qy^d 21
> 4 4325      1632 Female College      4 What kind ~ + 30
> 5 4325      1519 Female Less Than College 5 I think, ]~ + 21
> 6 4325      1632 Female College      6 Does it sa~ qy 30
> 7 4325      1519 Female Less Than College 7 I think it~ sd 21
> 8 4325      1519 Female Less Than College 8 You might ~ ad 21
> 9 4325      1519 Female Less Than College 9 I don't kn~ h 21
> 10 4325     1519 Female Less Than College 10 hold it do~ ad 21

```

Our final recoding step is to derive the frequency of filled pauses per utterance. In other words, we want to match the ‘uh’ and ‘um’ and return the number of matches for each utterance. There are a number of ways to do this. We could use an approach which applies the `str_extract_all()` function, which returns a list of matches, and then `unnest()` the list and count the number of matches. An alternative approach is to use the `str_count()` function to count the number of matches for a pattern. The later approach is more efficient for our purposes.

In either case, a pattern to match these annotations is needed. As always with pattern matching, we need to craft an expression that is as specific as possible to avoid over- or under-matching. We know from our observation and the corpus documentation that all filled pauses are wrapped by the `{F ...}` annotation. We can use this to our advantage. Before we jump into counting the filled pauses, let’s test a regular expression that matches the entire `{F ...}` annotation. An expression like `{F.*}` might be tempting, but this will be problematic for two reasons. First, since the `{` and `}` are regular expression operators we will need to escape them with the `\` convention. Second, the `*` operator is greedy, meaning that it will match the longest possible string. So if in a given utterance there are multiple filled pauses, the `*` operator will match all of them at once, not individually. To avoid this, we can use the `?` operator to make the `*` operator lazy. With these considerations in mind, we can move forward with `\{F.*?\}` as our expression.

We will send each utterance to the `str_extract_all()` function to match the filled pauses. This function returns a list of matches, so we will need to `unnest()` the list to get a vector of matches. Afterwards we will apply the `count()` function to summarize the number of matches for each match variation. The code is seen in Example 7.17.

Example 7.17.

```
# Test filled pause pattern
swda_curated_tbl |>
  mutate(
    matches = str_extract_all(
      utt_text,
      "\\\{F.*?\\\}")
  ) |>
  unnest(matches) |>
  count(matches)

> # A tibble: 120 x 2
>   matches      n
>   <chr>     <int>
> 1 "{F 0h, }"     1
> 2 "{F uh, }"     1
> 3 "{F \"0h, }"     1
> 4 "{F ((0h)) }"     1
> 5 "{F ((0h, }"     1
> 6 "{F ((Uh }"     1
> 7 "{F ((Uh)), }"     1
> 8 "{F ((Uh, }"     1
> 9 "{F + ] Um, }"     1
> 10 "{F + ] um, }"    1
> # i 110 more rows
```

The result from our test indicates that the `\\\{F .*\?\\\}` pattern matches a wide variety of filled pause annotations, 120 to be exact! We can see that there are a number of filled pause annotations that we are not be interested in, *e.g.* `{F 0h}`. Furthermore, the ‘uh’ and ‘um’ we are interested in sometimes include more annotation structure, *e.g.* `{F +] Um, }`.

A more sophisticated pattern is needed. When faced with a pattern matching task such as this, I find it helpful to start with a simple pattern and then add complexity as needed. This is an iterative process. To speed up the process, I often extract the matches and use an online tool for developing regular expressions, such as regex101.com².

²<https://regex101.com/>

With a (monster) regular expression that matches each of the filled pauses we are interested in, and only those filled pauses, we can move forward with counting the number of matches for each utterance.

To do this we will use the `str_count()` function from the `stringr` package. This function counts the number of matches for a pattern in a character vector. We will use `mutate()` to create new variables `uh` and `um` which will contain the counts for the filled pauses `uh` and `um`, respectively. The code is seen in Example 7.18.

Example 7.18.

```
# Recode filled pauses
swda_curated_tbl <-
  swda_curated_tbl |>
  mutate(
    uh = str_count(
      utt_text,
      "\\\{F\\\\s+[(+\\\\s\\\\])*\\u\\h[\\\\)\\\\s,\\\\.\\\\?-]*\\\\}"
    ),
    um = str_count(
      utt_text,
      "\\\{F\\\\s+[(+\\\\s\\\\])*\\u\\m[\\\\)\\\\s,\\\\.\\\\?-]*\\\\}"
    )
  )
# Preview the first 10 lines
swda_curated_tbl |>
  slice_head(n = 10)

> # A tibble: 10 x 10
>   doc_id speaker_id sex   education utt_id utt_text damsl_tag   age   uh   um
>   <dbl>     <dbl> <chr>     <dbl> <chr>   <chr>     <dbl> <int> <int>
> 1 4325      1632 Fema~ College      1 Okay. / o      30     0     0
> 2 4325      1632 Fema~ College      2 {D So, } qw     30     0     0
> 3 4325      1519 Fema~ Less Tha~    3 [ [ I g~ qy^d    21     0     0
> 4 4325      1632 Fema~ College      4 What ki~ +     30     0     0
> 5 4325      1519 Fema~ Less Tha~    5 I think~ +    21     1     0
> 6 4325      1632 Fema~ College      6 Does it~ qy    30     0     0
> 7 4325      1519 Fema~ Less Tha~    7 I think~ sd    21     0     0
> 8 4325      1519 Fema~ Less Tha~    8 You mig~ ad    21     1     0
> 9 4325      1519 Fema~ Less Tha~   9 I don't~ h     21     0     0
> 10 4325     1519 Fema~ Less Tha~  10 hold it~ ad    21     0     0
```

Now we have two new columns, `uh` and `um` which indicate how many times the relevant pattern was matched for a given utterance. By choosing to focus on disfluencies, however, we have made a decision to change the unit of observation from the utterance to the use of filled pauses (`uh` and `um`). This means that as the dataset stands, it is not in tidy format –where each observation corresponds to the observational unit. When datasets are misaligned in this particular way, there are in what is known as ‘wide’ format. What we want to do, then, is to restructure our dataset such that each row corresponds to the unit of observation –in this case each filled pause type.

To convert our current (wide) dataset to one where each filler type is listed and the counts are measured for each utterance we turn to the `pivot_longer()` function. This function creates two new columns, one in which the column names are listed and one for the values for each of the column names. The code is seen in Example 7.19.

Example 7.19.

```
# Tidy filled pauses
swda_curated_tbl <-
  swda_curated_tbl |>
  pivot_longer(
    cols = c("uh", "um"),
    names_to = "filler",
    values_to = "count"
  )
# Preview the first 10 lines
swda_curated_tbl |>
  slice_head(n = 10)

> # A tibble: 10 x 10
>   doc_id speaker_id sex   education    utt_id utt_text damsl_tag   age filler
>   <dbl>      <dbl> <chr>   <chr>      <dbl> <chr>   <chr>      <dbl> <chr>
> 1  4325       1632 Female College      1 Okay. / o      30 uh
> 2  4325       1632 Female College      1 Okay. / o      30 um
> 3  4325       1632 Female College      2 {D So, } qw     30 uh
> 4  4325       1632 Female College      2 {D So, } qw     30 um
> 5  4325       1519 Female Less Than Co~  3 [ [ I g~ qy^d   21 uh
> 6  4325       1519 Female Less Than Co~  3 [ [ I g~ qy^d   21 um
> 7  4325       1632 Female College      4 What ki~ +    30 uh
> 8  4325       1632 Female College      4 What ki~ +    30 um
> 9  4325       1519 Female Less Than Co~  5 I think~ +   21 uh
> 10 4325       1519 Female Less Than Co~  5 I think~ +   21 um
> # i 1 more variable: count <int>
```

Now we have a transformed dataset that is in tidy format. Each row corresponds to a filled pause type and the number of times it was used in a given utterance. It also includes the key socio-demographic variables we are interested in.

💡 Consider this

As confident as we may be in our recoding process, it is always a good idea to perform some data checks to ensure that the recoding process was successful. In the process, we can also gain some insight into the data. Considering the structure in the transformed Switchboard dialog Act Corpus dataset, what are some data checks we might want to perform? What are some insights we might gain from these data checks?

Finalize the transformation process by writing the dataset to a CSV file, create a data dictionary, and review and comment your code in the transformation script.

7.3 Tokenization

Another common transformation process that is particularly relevant for text analysis is tokenization. Tokenization is the process of segmenting units of language into components relevant for the research question. This includes breaking text in curated datasets into smaller units, such as words, *n*-grams, sentences, *etc.* or combining text into larger units relative to the original text.

The process of tokenization is fundamentally row-wise. By scaling the text units up or down, we change the unit of observation. It is important both for the research and the text processing to operationalize our language units. For example, while it may appear obvious to you what ‘word’ or ‘sentence’ means, a computer, and your reproducible research, needs a definition. This can prove trickier than it seems. For example, in English, we can segment text into words by splitting on whitespace. This works fairly well but there are some cases where this is not ideal. For example, in the case of contractions, such as *don't*, *won't*, *can't*, *etc.* the apostrophe is not a whitespace character. If we want to consider these contractions as separate words, then we need to consider a different tokenization strategy.

💡 Consider this

Consider the following paragraph:

“As the sun dipped below the horizon, the sky was set ablaze with shades of orange-red, illuminating the landscape. It’s a sight Mr. Johnson, a long-time observer, never tired of. On the lakeside, he’d watch with friends, enjoying the ever-changing hues—especially those around 6:30 p.m.—and reflecting on nature’s grand display. Even in the half-light, the water’s glimmer, coupled with the echo of distant laughter, created a timeless scene. The so-called

Table 7.5: Data dictionary for the CABNC Corpus.

| variable | name | description | variable_type |
|----------|----------------|---|---------------|
| doc_id | Document ID | Unique identifier for each document | string |
| part_id | Part ID | Identifier for the part within a document | string |
| sex | Sex | Gender of the person | string |
| age | Age | Age of the person | numeric |
| utt_id | Utterance ID | Identifier for each utterance | numeric |
| utt_text | Utterance Text | Text of the utterance | string |

‘magic hour’ was indeed magical, yet fleeting, like a well-crafted poem; it was the essence of life itself.”

What text conventions would pose issues for word tokenization based on a whitespace criterion?

Furthermore, tokenization strategies can vary between languages. For German words are often compounded together, meaning many ‘words’ will not be captured by the whitespace convention. Whitespace may not even be relevant for word tokenization in written languages, such as Chinese. The take home message is there is no one-size-fits-all tokenization strategy.

7.3.1 Orientation

Let’s look at a curated dataset from the CABNC Corpus to explore tokenization. The data dictionary for this dataset appears in Table 7.5.

The CABNC dataset contains a number of variables describing conversations between speakers of British English. Now let’s look at the dataset and preview the first 10 lines to get a sense of the values in the dataset, as in Example 7.20.

Example 7.20.

```
# Read in the dataset
cabnc_curated_tbl <-
  read_csv("data/derived/cabnc/cabnc_curated.csv")

# Preview the first 10 lines
cabnc_curated_tbl |>
  slice_head(n = 10)
```

```

> # A tibble: 10 x 6
>   doc_id  part_id sex     age  utt_id utt_text
>   <chr>    <chr>  <chr>  <dbl>  <dbl>  <chr>
> 1 KB0RE000 PS002 female  721      0 You enjoyed yourself in America
> 2 KB0RE000 PS006 male    601      1 Eh
> 3 KB0RE000 PS002 female  721      2 did you
> 4 KB0RE000 PS006 male    601      3 Oh I covered a nice trip yes
> 5 KB0RE000 PS002 female  721      4 Oh very good yeah
> 6 KB0RE000 PS006 male    601      5 Er saw Mary and Andrew and
> 7 KB0RE000 PS002 female  721      6 Yes you did
> 8 KB0RE000 PS006 male    601      7 in fact the whole family was together f-
> 9 KB0RE000 PS002 female  721      8 Oh very nice very nice yes It 's horrib-
> 10 KB0RE000 PS006 male   601      9 It is horrible is n't

```

Considering the data dictionary and the preview of the `cabnc_curated_tbl` dataset, we observe a number of metadata variables, such as `doc_id`, `part_id`, `sex`, `age`, and `utt_id` and the utterances in `utt_text`.

Let's assume that we are performing an exploratory analysis with this dataset and would like to consider the use of words and word sequences (*n*-grams). In this case we will be deriving multiple datasets with different units of observation.

7.3.2 Application

In the `cabnc_curated_tbl` data frame the `utt_text` variable contains the text we want to tokenize. We will start by tokenizing the text into words. Abstracting away from some of the metadata variables, if we envision what this should look like we might imagine something like Table 7.6.

Table 7.6: Example of tokenizing the `utt_text` variable into words.

| doc_id | utt_id | utt_word |
|----------|--------|----------|
| KB0RE000 | 0 | You |
| KB0RE000 | 0 | enjoyed |
| KB0RE000 | 0 | yourself |
| KB0RE000 | 0 | in |
| KB0RE000 | 0 | America |

Comparing Table 7.6 to the first line of the output of Example 7.14, we can see that we want to segment the words in the `utt_text` and then have each segment appear as a separate observation, retaining the relevant metadata variables.

Before we work with tokenizing text in a data frame, let's start with a character vector to get a sense of how tokenization works and what we will need to do to achieve the output in Table 7.6. Let's start with a character vector which contains the first three utterances from `cabnc_curated_tbl`. I will use `slice_head(n = 3)` and then `pull()` to extract the `utt_text` character vector in Example 7.21.

Example 7.21.

```
# Pull a character vector
cabnc_utts_chr <-
  cabnc_curated_tbl |>
  slice_head(n = 3) |>
  pull(utt_text)

# Preview the character vector
cabnc_utts_chr
```



```
> [1] "You enjoyed yourself in America" "Eh"
> [3] "did you"
```

We have the first three utterances in `cabnc_utts_chr`. Now we can tokenize the utterances into words using the `tokenize_words()` function from the `tokenizers` package (Mullen 2022). It's only required argument is a character vector, as seen in Example 7.22.

Example 7.22.

```
# Load package
library(tokenizers)

# Tokenize the utterances into words
cabnc_utts_chr |>
  tokenize_words()
```



```
> [[1]]
> [1] "you"      "enjoyed"   "yourself" "in"       "america"
>
> [[2]]
> [1] "eh"
>
> [[3]]
> [1] "did" "you"
```

The output shows that we get a list of length 3, one for each utterance. Each list element contains a character vector with different lengths based on the number of tokens created from the original utterance.

Now if we add the `tokenize_words()` function to a `mutate()` call, we can create a new variable with the tokenized utterances. However, the value for each observation will be a list. To expand the character vectors within each list into separate observations, we can use the `unnest()` function on the new variable. I will assign the result to `cabnc_unigrams_tbl` as we will have one-word tokens, as seen in Example 7.23.

Example 7.23.

```
# Tokenize the utterances into words
cabnc_unigrams_tbl <-
  cabnc_curated_tbl |>
  mutate(utt_words = tokenize_words(utt_text)) |>
  unnest(cols = utt_words)

# Preview the first 10 lines
cabnc_unigrams_tbl |>
  slice_head(n = 10)
```

```
> # A tibble: 10 x 7
>   doc_id  part_id sex     age utt_id utt_text          utt_words
>   <chr>    <chr>  <chr>   <dbl> <dbl>  <chr>          <chr>
> 1 KB0RE000 PS002 female    721     0 You enjoyed yourself in Ameri~ you
> 2 KB0RE000 PS002 female    721     0 You enjoyed yourself in Ameri~ enjoyed
> 3 KB0RE000 PS002 female    721     0 You enjoyed yourself in Ameri~ yourself
> 4 KB0RE000 PS002 female    721     0 You enjoyed yourself in Ameri~ in
> 5 KB0RE000 PS002 female    721     0 You enjoyed yourself in Ameri~ america
> 6 KB0RE000 PS006 male     601     1 Eh                  eh
> 7 KB0RE000 PS002 female    721     2 did you           did
> 8 KB0RE000 PS002 female    721     2 did you           you
> 9 KB0RE000 PS006 male     601     3 Oh I covered a nice trip yes   oh
> 10 KB0RE000 PS006 male    601     3 Oh I covered a nice trip yes  i
```

The `cabnc_unigrams_tbl` dataset is in the format we want, each row corresponds to a word token and each column contains the relevant metadata. The `tokenizers` package includes a variety of tokenization functions. For example, we can use the `tokenize_ngrams()` function to create n -grams. The `tokenize_ngrams()` function takes a character vector and a value for n and returns a list of n -grams. Other functions can tokenize character n -grams, sentences,

paragraphs, lines, or even allow you to specify a custom tokenization function with a regular expression.

As we have seen the `tokenize_*`() set of functions take a character vector and return a list of tokens. And if we are working with a data frame, we can then work to expand the list into separate observations. This is a common pattern in text analysis. So common, in fact, that the `tidytext` package (Robinson and Silge 2023) includes a function, `unnest_tokens()` that wraps the `tokenize_*` functions and expand the list of tokens into separate observations in one step. The tokenization types available are `character`, `word`, `ngram`, `sentence`, `regex`, and `skip_ngram`. We will use the `word` tokenization type to recreate the `cabnc_unigrams_tbl` dataset, as seen in Example 7.24.

Example 7.24.

```
# Load package
library(tidytext)

# Tokenize the utterances into words
cabnc_unigrams_tbl <-
  cabnc_curated_tbl |>
  unnest_tokens(
    output = utt_word,
    input = utt_text,
    token = "words"
  )
# Preview the first 10 lines
cabnc_unigrams_tbl |>
  slice_head(n = 10)
```

```
> # A tibble: 10 x 6
>   doc_id  part_id sex      age  utt_id utt_word
>   <chr>    <chr> <chr>    <dbl> <dbl> <chr>
> 1 KB0RE000 PS002 female    721     0 you
> 2 KB0RE000 PS002 female    721     0 enjoyed
> 3 KB0RE000 PS002 female    721     0 yourself
> 4 KB0RE000 PS002 female    721     0 in
> 5 KB0RE000 PS002 female    721     0 america
> 6 KB0RE000 PS006 male     601     1 eh
> 7 KB0RE000 PS002 female    721     2 did
> 8 KB0RE000 PS002 female    721     2 you
> 9 KB0RE000 PS006 male     601     3 oh
> 10 KB0RE000 PS006 male    601     3 i
```

The `utt_word` column in the outputs from both Example 7.23 and Example 7.24 are identical. One thing to note, however, is that the original `utt_text` variable is dropped in the `unnest_tokens()` approach. This one of a few default values for parameters which include `drop = TRUE` (dropping the original text variable) and `to_lower = TRUE`. In fact the `token = "words"` parameter is not needed as it is the default. We can change these defaults as we see fit.

➊ Dive deeper

The approach using either `tokenize_*`() or `unnest_tokens()` approaches tokenization from a segmentation point of view. That is, the context separating our tokens is tarterd and is removed. In some cases it may be more feasible to turn the approach around and instead target the tokens to extract. The `str_extract_all()` can be used for this purpose. Note, however, the former approach is often more effective and effecient. The later requires a regular expression, or set of, which can be tricky to develop for some tokenization tasks.

As we create derived datasets to explore, let's also create bigram tokens. We can do this by changing the `token` parameter to "ngrams" and specifying the value for n with the `n` parameter. I will assign the result to `cabnc_bigrams_tbl` as we will have two-word tokens, as seen in Example 7.25.

Example 7.25.

```
# Tokenize the utterances into bigrams
cabnc_bigrams_tbl <-
  cabnc_curated_tbl |>
  unnest_tokens(
    output = utt_bigram,
    input = utt_text,
    token = "ngrams",
    n = 2
  )
# Preview the first 10 lines
cabnc_bigrams_tbl |>
  slice_head(n = 10)

> # A tibble: 10 x 6
>   doc_id  part_id sex      age  utt_id  utt_bigram
>   <chr>    <chr>  <chr>  <dbl> <dbl>  <chr>
> 1 KB0RE000 PS002  female    721      0 you enjoyed
> 2 KB0RE000 PS002  female    721      0 enjoyed yourself
> 3 KB0RE000 PS002  female    721      0 yourself in
> 4 KB0RE000 PS002  female    721      0 in america
```

```

> 5 KB0RE000 PS006 male 601 1 <NA>
> 6 KB0RE000 PS002 female 721 2 did you
> 7 KB0RE000 PS006 male 601 3 oh i
> 8 KB0RE000 PS006 male 601 3 i covered
> 9 KB0RE000 PS006 male 601 3 covered a
> 10 KB0RE000 PS006 male 601 3 a nice

```

The two-word token sequences for each utterance appear as observations in the `cabnc_bigrams_tbl` dataset. You may notice that in row 5 the value for `utt_bigram` is NA. This is because the utterance only contains one word. The `unnest_tokens()` function will not create a token for a sequence that does not exist.

If we don't want to lose this information, we can modify the original `utt_text` variable to include a placeholder, some symbol that we will use to denote that a single word utterance is present. Another strategy which accomplishes the first goal and may enrich the bigram dataset is to add a start and end token to each utterance. This will allow us to identify the first and last word in each utterance. In either case we can turn to the `str_c()` function which will allow us to concatenate strings. In Example 7.26, I will add a start and end token to each utterance and then tokenize the utterances into bigrams.

Example 7.26.

```

# Prepend and append (x) char to `utt_text`
cabnc_derived_tbl <-
  cabnc_curated_tbl |>
  mutate(utt_text = str_c("x", utt_text, "x", sep = " "))
  )
# Tokenize the utterances into bigrams
cabnc_bigrams_tbl <-
  cabnc_derived_tbl |>
  unnest_tokens(
    output = utt_bigram,
    input = utt_text,
    token = "ngrams",
    n = 2
  )
# Preview the first 10 lines
cabnc_bigrams_tbl |>
  slice_head(n = 10)

```

```

> # A tibble: 10 x 6
>   doc_id  part_id sex      age  utt_id  utt_bigram

```

```

>     <chr>     <chr>     <chr>   <dbl>   <dbl> <chr>
> 1 KB0RE000 PS002 female    721      0 x you
> 2 KB0RE000 PS002 female    721      0 you enjoyed
> 3 KB0RE000 PS002 female    721      0 enjoyed yourself
> 4 KB0RE000 PS002 female    721      0 yourself in
> 5 KB0RE000 PS002 female    721      0 in america
> 6 KB0RE000 PS002 female    721      0 america x
> 7 KB0RE000 PS006 male     601      1 x eh
> 8 KB0RE000 PS006 male     601      1 eh x
> 9 KB0RE000 PS002 female    721      2 x did
> 10 KB0RE000 PS002 female   721      2 did you

```

⚠ Warning

In Example 7.26 I used `x` as the start and end token. My thought process was that `x` will be a unique token that will not be present in the utterances. This is a good strategy, but it is not foolproof. Another, more meaningful strategy may be to encode the start and end with `#` and `$`, respectively. In this case, however, this option is not ideal as the `unnest_tokens()` and `tokenize_ngrams()` functions strip punctuation by default. This means that the `#` and `$` will be removed.

The most common tokenization strategy is to segment text into smaller units, often words. However, there are times when we may want to segment text into larger units, effectively collapsing over rows. For example, if we are working with a curated dataset which is tokenized by words, we may want to segment the text into sentences. A couple considerations are in order, however. First, we need to be clear about what we mean by ‘sentence’ and how we will segment the text into sentences. In some cases key cues for sentence boundaries, such as sentential punctuation have been stripped from the text, making a simple definition of a sentence difficult or impossible to be performed computationally. Second, we also need to be clear how we will handle the metadata variables. In other words, when we collapse over rows, we need to be aware and intentional about how we group these new units of observation. For example, if we collapse the `cabnc_unigrams_tbl` dataset into sentences, will we group the sentences by `doc_id` or `part_id` or some other combination of metadata variables?

Let’s see how we might collapse text rows into larger units using the original curated CABNC data frame `cabnc_curated_tbl`. Refer back to the output in Example 7.20. The values in `utt_text` are utterances, which may map to sentences in some cases, but often not. Regardless, there is no sentential punctuation to help identify sentences, even across utterance observations. Furthermore, it may not even make sense to discuss sentences in the context of spoken language.

A unit that may make more sense is utterances per speaker per document. Note this context ‘per speaker per document’. In effect this is a grouping parameter for our approach to collapsing the text in `utt_text`. With a goal in mind we can turn to the `group_by()` function to group

our dataset and then `summarize()` to collapse the text in `utt_text` with the `str_c()` and the parameter `collapse = " "`. The code is seen in Example 7.27.

Example 7.27.

```
# Collapse utterances by speaker by document
cabnc_utterances_tbl <-
  cabnc_curated_tbl |>
  group_by(doc_id, part_id) |>
  summarize(utt_text = str_c(utt_text, collapse = " ")) |>
  ungroup()

# Preview the first 10 lines
cabnc_utterances_tbl |>
  slice_head(n = 10)

> # A tibble: 10 × 3
>   doc_id  part_id utt_text
>   <chr>    <chr>   <chr>
> 1 KB0RE000 KB0PSUN Hmm Hello
> 2 KB0RE000 PS002  You enjoyed yourself in America did you Oh very good yeah Y~
> 3 KB0RE000 PS006  Eh Oh I covered a nice trip yes Er saw Mary and Andrew and ~
> 4 KB0RE001 KB0PSUN Neck of lamb 's off Hello Morning
> 5 KB0RE001 PS005  You can tape me by all means but they probably wo n't like ~
> 6 KB0RE001 PS007  They want to know what spoken English is like Er now what d~
> 7 KB0RE002 PS003  No Was born in Brockly I was n't born sorry about that my f~
> 8 KB0RE002 PS007  You 're not Welsh speaking at all are you Mm mm but you are~
> 9 KB0RE003 PS006  It 's all supposed to be anonymous anyway Right now do you ~
> 10 KB0RE003 PS007 I do n't think I I 'm likely to say anything use use down a~
```

Now we have a data frame where the utterances for each speaker for each document are collapsed into a single observation. Yet by collapsing the utterances in this way, we have lost the speaker metadata `sex` and `age`. We can add this information by one of two approaches. The first is to simply add `sex` and `age` to the grouping parameter in `group_by()`. Since these variables are descriptors of `part_id` they will not change the result of our collapsing the text, but will be retained in the resulting output. The second approach is to use a join operation to add the metadata back to the collapsed dataset. I won't go into this approach here as we will cover joins head-on in Section 7.5.

7.4 Generation

The process of generation involves the addition of information to a dataset. This differs from the previous transformation procedures in that normalization, recoding, and tokenization involve manipulating, classifying, and/ or deriving information based on characteristics explicit in a dataset. Instead, generation involves deriving new information based on characteristics implicit in a dataset.

The most common type of operation involved in the generation process is the addition of linguistic annotation. This process can be accomplished manually by a researcher or research team or automatically through the use of pre-trained linguistic resources and/ or software. Ideally the annotation of linguistic information can be conducted automatically.

There are important considerations, however, that need to be taken into account when choosing whether linguistic annotation can be conducted automatically. First and foremost has to do with the type of annotation desired. Information such as part of speech (grammatical category) and morpho-syntactic information are the the most common types of linguistic annotation that can be conducted automatically.

Second, the degree to which the resource that will be used to annotate the linguistic information is aligned with the language variety and/or register is also a key consideration. As noted, automatic linguistic annotation methods are contingent on pre-trained resources. The language and language variety used to develop these resources may not be available for the language under investigation, or if it does, the language variety and/ or register may not align. The degree to which a resource does not align with the linguistic information targeted for annotation is directly related to the quality of the final annotations. To be clear, no annotation method, whether manual or automatic is guaranteed to be perfectly accurate.

7.4.1 Orientation

As an example, we'll posit that we are conducting translation research. Specifically, we will set up an investigation into the effect of translation on the syntactic simplification of text. The basic notion is that when translators translate text from one language to another, they subconsciously simplify the text, compared to native texts (Liu and Afzaal 2021).

To assess this hypothesis, we will need to identify comparable translated and native texts. The ENNTT corpus contains native and translated English. The texts are drawn from European Parliament proceedings ensuring that the texts are comparable in terms of register.

The data dictionary for the curated native dataset appears in Table 7.7.

Both the natives and translations datasets have the same variables. For the purposes of this investigation, the `text` and `type` variables are the most relevant. A variable to index the text, such as `doc_id`, will be useful as we move forward and will be added to the dataset.

Table 7.7: Data dictionary for the curated native ENNTT dataset.

| variable | name | variable_type | description |
|-------------|------------------|---------------|---|
| session_id | Session ID | categorical | Unique identifier for each session |
| speaker_id | Speaker ID | categorical | Unique identifier for each speaker |
| state | State | categorical | The country or region the speaker is from |
| session_seq | Session Sequence | ordinal | The order in which the session occurred |
| text | Text | categorical | The spoken text during the session |
| type | Type | categorical | The type of speaker. Natives in this dataset. |

The next step is to operationalize what we mean by syntactic simplification. There are many measures of syntactic complexity (Szmrecsanyi 2004). For our purposes, let's focus on two: number of T-units and sentence length (in words). Length is straightforward to calculate after word tokenization but a T-unit is a bit more involved. A T-unit is a main clause and all of its subordinate clauses. To calculate the number of T-units, we will need to identify the main clauses and their subordinate clauses.

An idealized transformed dataset for this investigation would look something like Table 7.8.

Table 7.8: Idealized transformed dataset for the syntactic simplification investigation.

| doc_id | type | t_units | text |
|--------|------------|---------|---|
| 1 | translated | 5 | I am happy right now. |
| 2 | translated | 11 | I think that John believes that Mary is a good person. |
| 3 | native | 2 | She thinks I am not happy right now. |
| 4 | native | 4 | Although she knew that the weather was bad, Mary decided to go for a walk, hoping that she would feel better. |

To identify the main clauses and their subordinate clauses, we will need to annotate the ENNTT texts with syntactic information. Specifically, we will need to identify and count the main clauses and their subordinate clauses.

7.4.2 Application

As fun as it would be to hand-annotate the ENNTT corpus, we will instead turn to automatic linguistic annotation. Specifically, we will use the `udpipe` package (Wijffels 2023) which provides an interface for annotating text using pre-trained models from the Universal Dependencies³ (UD) project (Nivre et al. 2020). The UD project is an effort to develop cross-linguistically consistent treebank annotation for a variety of languages.

³<https://universaldependencies.org/>

Our first step, then, is to peruse the available pre-trained models for the languages we are interested in and selected the most register-aligned models. The models, model names, and licensing information are documented in the `udpipe` package and can be accessed by running `?udpipe::udpipe_download_model()` in the R console. For illustrative purposes, the `english` treebank model from the <https://github.com/bnosac/udpipe.models.ud> repository which is released under the CC-BY-SA license⁴. This model is trained on various sources including news, Wikipedia, and web data of various genres.

Let's set the stage by providing an overview of the annotation process.

1. Load the `udpipe` package.
2. Select the pre-trained model to use and the directory where the model will be stored in your local environment.
3. Prepare the dataset to be annotated (if necessary). This includes ensuring that the dataset has a column of text to be annotated and a grouping column. By default, the names of these columns are expected to be `text` and `doc_id`, respectively. The `text` column needs to be a character vector and the `doc_id` column needs to be a unique index for each text to be annotated.
4. Annotate the dataset. The result returns a data frame.

Steps 3 and 4 are repeated for the `enntt_natives_curated` and the `enntt_translations_curated` datasets. For brevity, I will only show the code for the dataset for the natives. Additionally, I will subset the dataset to 10,000 randomly selected lines for both dataset, as in Example 7.28 for the natives. Syntactic annotation is a computationally expensive operation and the natives and translations datasets contain 116,341 and 738,597 observations, respectively.

Example 7.28.

```
# Subset the natives ENNTT dataset
enntt_natives_tbl <-
  enntt_natives_curated_tbl |>
  slice_sample(n = 10000)
```

👉 Tip Computational Performance

In your own research computationally expensive cannot be avoided, but it can be managed. One strategy is to work with a subset of the data until your code is working as expected. Once you are confident that your code is working as expected, then you can scale up to the full dataset.

⁴<https://creativecommons.org/licenses/by-sa/4.0/>

If you are using Quarto, you can use the `cache: true` metadata field in your code blocks to cache the results of computationally expensive code blocks. This will allow you to run your code once and then use the cached results for subsequent runs.

Parallel processing is another strategy for managing computationally expensive code. Some packages, such as `udpipe`, have built-in support for parallel processing. Other packages, such as `tidytext`, do not. In these cases, you can use the `future` package (Bengtsson 2023) to parallelize your code.

With the `enntt_natives_tbl` object, let's execute steps 1-4, as seen in Example 7.29.

Example 7.29.

```

# Load package
library(udpipe)

# Model and directory
model <- "english"
model_dir <- "../data/"

# Prepare the dataset to be annotated
enntt_natives_preped_tbl <-
  enntt_natives_tbl |>
  mutate(doc_id = row_number()) |>
  select(doc_id, text)

# Annotate the dataset
enntt_natives_ann <-
  udpipe(
    x = enntt_natives_preped_tbl,
    object = model,
    model_dir = model_dir
  ) |>
  tibble()

# Preview
glimpse(enntt_natives_ann)

```

```

> $ sentence_id  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
> $ sentence      <chr> "It is extremely important that action is taken to ensur~
> $ start        <int> 1, 4, 7, 17, 27, 32, 39, 42, 48, 51, 58, 63, 66, 73, 77, ~
> $ end          <int> 2, 5, 15, 25, 30, 37, 40, 46, 49, 56, 61, 64, 71, 75, 82 ~
> $ term_id      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1 ~
> $ token_id     <chr> "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", ~
> $ token        <chr> "It", "is", "extremely", "important", "that", "action", ~
> $ lemma         <chr> "it", "be", "extremely", "important", "that", "action", ~
> $ upos          <chr> "PRON", "AUX", "ADV", "ADJ", "SCONJ", "NOUN", "AUX", "VE ~
> $ xpos          <chr> "PRP", "VBZ", "RB", "JJ", "IN", "NN", "VBZ", "VBN", "TO" ~
> $ feats         <chr> "Case=Nom|Gender=Neut|Number=Sing|Person=3|PronType=Prs" ~
> $ head_token_id <chr> "4", "4", "4", "0", "8", "8", "8", "4", "10", "8", "13", ~
> $ dep_rel       <chr> "expl", "cop", "advmod", "root", "mark", "nsubj:pass", " ~
> $ deps          <chr> NA, ~
> $ misc          <chr> NA, ~

```

There is quite a bit of information which is returned from `udpipe()`. Note that the input lines have been tokenized by word. Each token includes the `token`, `lemma`, part of speech (`upos` and `xpos`), morphological features (`feats`), and syntactic relationships (`head_token_id` and `dep_rel`). The `token_id` keeps track of the token's position in the sentence and the `sentence_id` keeps track of the sentence's position in the original text. In the case of the Europarl dataset, most values of `lines` are just one sentence, but there are some cases where the `lines` variable contains multiple sentences in which the `sid` will be incremented. Finally, the `doc_id` column and its values correspond to the `doc_id` in the `enntt_natives_tbl` dataset.

The number of variables in the `udpipe()` annotation output is quite overwhelming. However, these attributes come in handy for manipulating, extracting, and plotting information based on lexical and syntactic patterns. See the dependency tree in Figure 7.1 for an example of the syntactic information that can be extracted from the `udpipe()` annotation output.

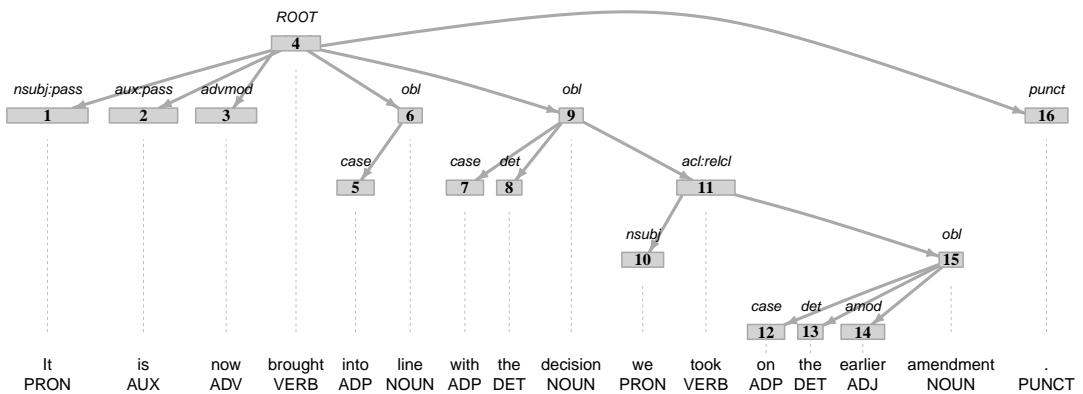


Figure 7.1: Plot of the syntactic tree for a sentence in the ENNTT natives dataset.

➊ Dive deeper

The plot in Figure 7.1 was created using the `rsyntax` package ([R-rsyntax?](#)). In addition to creating dependency tree plots, the `rsyntax` package can be used to extract syntactic patterns from the `udpipe()` annotation output. See the documentation for more information^a.

^a<https://github.com/vanatteveldt/rsyntax>

In Figure 7.1 we see the syntactic tree for a sentence in the ENNTT natives dataset. Each node is labeled with the `token_id` which provides the linear ordering of the sentence. Above the nodes the `dep_relation`, or dependency relationship label is provided. These labels are based on the UD project's dependency relations⁵. We can see that the 'ROOT' relation is at the top of the tree and corresponds to the verb 'brought'. 'ROOT' relations mark predicates in the sentence. Not seen in the example tree, 'cop' relation is a copular, or non-verbal predicate and should be included. These are the key syntactic pattern we will use to identify main clauses for T-units. Now we need to identify the subordinate clauses. In the UD project's listings, the relations 'ccomp' (clausal complement), 'xcomp' (open clausal complement), and 'acl:relcl' (relative clause), as seen in Figure 7.1) are subordinate clauses.

To calculate the number of T-units and words per sentence we turn to the `dplyr` package. We will use the `group_by()` function to group the dataset by `doc_id` and `sentence_id` and then use the `summarize()` function to calculate the number of T-units and words per sentence, where a T-unit is the combination of the sum of main clauses and sum of subordinate clauses. The code is seen in Example 7.30.

⁵<https://universaldependencies.org/u/dep/index.html>

Example 7.30.

```
# Calculate the number of T-units and words per sentence
enntt_natives_ann_tunits_words_tbl <-
  enntt_natives_ann |>
  group_by(doc_id, sentence_id) |>
  summarize(
    main_clauses = sum(dep_rel %in% c("ROOT", "cop")),
    subord_clauses = sum(dep_rel %in% c("ccomp", "xcomp", "acl:relcl")),
    t_units = main_clauses + subord_clauses,
    word_len = n()
  ) |>
  ungroup()

# Preview
glimpse(enntt_natives_ann_tunits_words_tbl)
```

```
> Rows: 10,199
> Columns: 6
> $ doc_id      <chr> "1", "10", "100", "1000", "10000", "1001", "1002", "100~
> $ sentence_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
> $ main_clauses <int> 1, 0, 0, 0, 2, 0, 0, 1, 1, 0, 1, 1, 0, 2, 0, 1, 1, 1~
> $ subord_clauses <int> 3, 2, 1, 0, 1, 2, 1, 1, 0, 2, 1, 0, 3, 2, 0, 4, 2, 1, 1~
> $ t_units      <int> 4, 2, 1, 0, 3, 2, 1, 2, 1, 2, 2, 1, 4, 2, 2, 4, 3, 2, 2~
> $ word_len     <int> 21, 25, 27, 15, 40, 43, 29, 23, 13, 30, 33, 9, 68, 35, ~
```

A quick spot check of some sentences calculations `enntt_natives_ann_tunits_words_tbl` dataset against the `enntt_natives_ann` is good to ensure that the calculation is working as expected. In Figure 7.2 we see a sentence that has a word length of 13 and a T-unit value of 5.

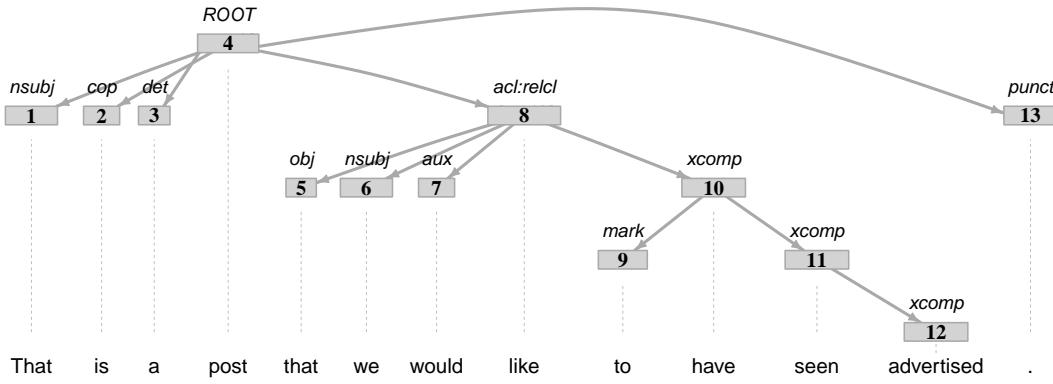


Figure 7.2: Sentence with a word length of 13 and a T-unit value of 5.

Now we can drop the intermediate columns we created to calculate our key syntactic complexity measures using `select()` to indicate those that we do want to keep. I will assign the result to `enntt_natives_syn_comp` as we will be working with syntactic complexity measures for the native texts, as seen in Example 7.31.

Example 7.31.

```
# Select columns
enntt_natives_syn_comp <-
  enntt_natives_ann_tunits_words_tbl |>
  select(doc_id, sentence_id, t_units, word_len)
```

Now we can repeat the process for the ENNTT translated dataset. I will assign the result to `enntt_translations_syn_comp`. The next step is to join the sentences from the annotated data frames into our datasets so that we have the information we set out to generate for both datasets. Then we will concatenate both the `enntt_natives_syn_comp` and `enntt_translations_syn_comp` datasets into a single dataset. Both the join and the concatenation will be covered in the next section.

7.5 Merging

One final class of transformations that can be applied to curated datasets to enhance their informativeness for a research project is the process of merging two or more datasets. There

are two primary types of merging: joins and concatenation. **Joins** can be row- or column-wise operations that combine datasets based on a common attribute or set of attributes. **Concatenation** is exclusively a row-wise operation that combines datasets that share the same attributes.

Of the two types of merges, joins are the most powerful and sometimes more difficult to understand. When two datasets are joined at least one common variable must be shared between the two datasets. The common variable(s) are referred to as **keys**. The keys are used to match observations in one dataset with observations in another dataset by serving as an index.

There are a number of join types. The most common are left, full, semi, and anti. The type of join determines which observations are retained in the resulting dataset. Let's see this in practice. First, let's create two datasets to join with a common variable **key**, as seen in Example 7.32.

Example 7.32.

```
a_tbl <- tibble(
  key = c(1, 2, 3, 5, 8),
  a = letters[1:5]
)

b_tbl <- tibble(
  key = c(1, 2, 4, 6, 8),
  b = letters[6:10]
)

a_tbl
b_tbl

> # A tibble: 5 x 2
>   key     a
>   <dbl> <chr>
> 1     1 a
> 2     2 b
> 3     3 c
> 4     5 d
> 5     8 e
> # A tibble: 5 x 2
>   key     b
>   <dbl> <chr>
> 1     1 f
> 2     2 g
> 3     4 h
> 4     6 i
> 5     8 j
```

The **a_tbl** and the **b_tbl** datasets share the **key** variable, but the values in the **key** variable are not identical. The two datasets share values 1, 2, and 8. The **a_tbl** dataset has values 3 and 5 in the **key** variable and the **b_tbl** dataset has values 4 and 6 in the **key** variable.

If we apply a left join to the **a_tbl** and **b_tbl** datasets, the result will be a dataset that retains all of the observations in the **a_tbl** dataset and only those observations in the **b_tbl** dataset that have a match in the **a_tbl** dataset. The result is seen in Example 7.33.

Example 7.33.

```
left_join(x = a_tbl, y = b_tbl, by = "key")
```

```
> # A tibble: 5 x 3
>   key a     b
>   <dbl> <chr> <chr>
> 1     1 a     f
> 2     2 b     g
> 3     3 c     <NA>
> 4     5 d     <NA>
> 5     8 e     j
```

Now, if the key variable has the same name, R will recognize and assume that this is the variable to join on and we don't need the `by =` argument, but if there are multiple potential key variables, we use `by =` to specify which one to use.

A full join retains all observations in both datasets, as seen in Example 7.34.

Example 7.34.

```
full_join(x = a_tbl, y = b_tbl)
```

```
> # A tibble: 7 x 3
>   key a     b
>   <dbl> <chr> <chr>
> 1     1 a     f
> 2     2 b     g
> 3     3 c     <NA>
> 4     5 d     <NA>
> 5     8 e     j
> 6     4 <NA>  h
> 7     6 <NA>  i
```

Left and full joins maintain or increase the number of observations. On the other hand, semi and anti joins aim to decrease the number of observations. A semi join retains only those observations in the left dataset that have a match in the right dataset, as seen in Example 7.35.

Example 7.35.

```
semi_join(x = a_tbl, y = b_tbl)
```

```
> # A tibble: 3 x 2
>   key a
>   <dbl> <chr>
> 1     1 a
> 2     2 b
> 3     8 e
```

And an anti join retains only those observations in the left dataset that do not have a match in the right dataset, as seen in Example 7.36.

Example 7.36.

```
anti_join(x = a_tbl, y = b_tbl)
```

```
> # A tibble: 2 x 2
>   key a
>   <dbl> <chr>
> 1     3 c
> 2     5 d
```

Of these join types, the left join and the anti join are some of the most common to encounter in research projects.

::: {.callout} Consider this

In addition to datasets that are part of an acquired resource or derived from a corpus resource, there are also a number of datasets that are included in R packages that are particularly relevant for text analysis. For example, the `tidytext` package includes `sentiments` and `stop_words` datasets. The `lexicon` package (Rinker 2019) includes large number of datasets that include sentiment lexicons, stopword lists, contractions, and more.

Consider the `lexicon::key_contractions` dataset. This dataset includes a list of common contractions and their expanded forms.

What needs to be done to join these two datasets? What are the keys? What type of join should be used? What is the resulting dataset?

Table 7.9: Common contractions and expanded forms.

| contraction | expanded |
|-------------|------------|
| 'cause | because |
| 'tis | it is |
| 'twas | it was |
| ain't | am not |
| aren't | are not |
| can't | can not |
| could've | could have |
| it's | it is |

Table 7.10: Example tokenized dataset with contractions.

| doc_id | sent_id | token_id | token |
|--------|---------|----------|---------|
| 1 | 1 | 1 | I |
| 1 | 1 | 2 | can't |
| 1 | 1 | 3 | believe |
| 1 | 1 | 4 | that |
| 1 | 1 | 5 | it's |
| 1 | 1 | 6 | not |
| 1 | 1 | 7 | butter |
| 1 | 1 | 8 | ! |

7.5.1 Orientation

With this in mind, let's return to our syntactic simplification investigation. Recall that we started with two curated ENNTT datasets: the natives and translations. We manipulated these datasets subsetting them to 10,000 randomly selected lines, prepped them for annotation by adding a `doc_id` column and dropping all columns except `text`, and then annotated them using the `udpipe` package. We then calculated the number of T-units and words per sentence.

These steps produced two datasets for both the natives and for the translations. The first dataset for each is the annotated data frame. The second is the data frame with the syntactic complexity measures we calculated. The annotated data frames are named `enntt_natives_ann` and `enntt_translations_ann`. The data frames with the syntactic complexity measures are named `enntt_natives_syn_comp` and `enntt_translations_syn_comp`.

In the end, we want a dataset that looks something like Table 7.11.

Table 7.11: Idealized merged dataset for the syntactic simplification investigation.

| doc_id | type | t_units | word_len | text |
|--------|-------------|---------|----------|--|
| 1 | natives | 1 | 5 | I am happy right now. |
| 2 | translation | 3 | 11 | I think that John believes that Mary is a good person. |

To create this unified dataset, we will need to apply joins and concatenation. First, we will join the prepped datasets with the annotated datasets. Then we will concatenate the two resulting datasets.

7.5.2 Application

Let's start by joining the annotated datasets (`enntt_natives_ann` and `enntt_translations_ann`) with the datasets with the syntactic complexity calculations (`enntt_natives_syn_comp` and `enntt_translations_syn_comp`). In these joins, we can see that the prepped and calculated datasets share a couple variables, `doc_id` and `sentence_id`, in Example 7.37.

Example 7.37.

```
# Preview datasets to join
enntt_natives_ann |>
  slice_head(n = 3)

> # A tibble: 3 x 17
>   doc_id paragraph_id sentence_id sentence      start    end term_id token_id token
>   <chr>      <int>      <int> <chr>      <int> <int> <int> <chr>   <chr>
> 1 1           1          1 It is extr~      1     2     1 1      It
> 2 1           1          1 It is extr~      4     5     2 2      is
> 3 1           1          1 It is extr~      7    15     3 3      extr~
> # i 8 more variables: lemma <chr>, upos <chr>, xpos <chr>, feats <chr>,
> #   head_token_id <chr>, dep_rel <chr>, deps <chr>, misc <chr>

enntt_natives_syn_comp |>
  slice_head(n = 3)

> # A tibble: 3 x 4
>   doc_id sentence_id t_units word_len
>   <chr>      <int>    <int>    <int>
> 1 1           1        4       21
> 2 10          1        2       25
> 3 100         1        1       27
```

The `doc_id` and `sentence_id` variables are both keys that we will use to join the datasets. The reason being that if we only use one of the two we will not align the two datasets at the sentence level. Only the combination of `doc_id` and `sentence_id` isolates the sentences for which we have syntactic complexity measures. Beyond having common variable (or variables in our case), we must also ensure that join key variables are of the same vector type in both data frames and that we are aware of any differences in the values. From the output in Example 7.37, we can see that the `doc_id` and `sentence_id` variables aligned in terms of vector type; `doc_id` is

character and `sentence_id` is integer in both data frames. If they happened not to be, there types would need to be adjusted.

Now, we need to check for differences in the values. We can do this by using the `setequal()` function. This function returns `TRUE` if the two vectors are equal and `FALSE` if they are not. If the two vectors are not equal, the function will return the values that are in one vector but not the other. So if one has `10001` and the other doesn't we will get `FALSE`. Let's see this in practice, as seen in Example 7.38.

Example 7.38.

```
# Check for differences in the values
setequal(
  enntt_natives_ann$doc_id,
  enntt_natives_syn_comp$doc_id
)
```

```
> [1] TRUE
```

```
setequal(
  enntt_natives_ann$sentence_id,
  enntt_natives_syn_comp$sentence_id
)
```

```
> [1] TRUE
```

So the values are the same. The final check is to see if the vectors are of the same length. We know the values are the same, but we don't know if the values are repeated. We do this by simply comparing the length of the vectors, as seen in Example 7.39.

Example 7.39.

```
# Check for differences in the length
length(enntt_natives_ann$doc_id) ==
  length(enntt_natives_syn_comp$doc_id)
```

```
> [1] FALSE
```

```

length(enntt_natives_ann$sentence_id) ==
length(enntt_natives_syn_comp$sentence_id)

> [1] FALSE

```

So they are not the same length. Using the `nrow()` function, I can see that the annotated dataset has 264124 observations and the calculated dataset has 10199 observations. The annotation data frames will have many more observations due to the fact that the unit of observations is word tokens. The calculated data frames' unit of observation is the sentence.

To appreciate the difference in the number of observations, let's look at the first 10 observations of the natives annotated frame for just the columns of interest, as seen in Example 7.40.

Example 7.40.

```

# Preview the annotated dataset
enntt_natives_ann |>
  select(doc_id, sentence_id, sentence, token) |>
  slice_head(n = 10)

> # A tibble: 10 × 4
>   doc_id sentence_id sentence                                token
>   <chr>     <int> <chr>
> 1 1           1 It is extremely important that action is taken to e~ It
> 2 1           1 It is extremely important that action is taken to e~ is
> 3 1           1 It is extremely important that action is taken to e~ extr~
> 4 1           1 It is extremely important that action is taken to e~ impo~
> 5 1           1 It is extremely important that action is taken to e~ that
> 6 1           1 It is extremely important that action is taken to e~ acti~
> 7 1           1 It is extremely important that action is taken to e~ is
> 8 1           1 It is extremely important that action is taken to e~ taken
> 9 1           1 It is extremely important that action is taken to e~ to
> 10 1          1 It is extremely important that action is taken to e~ ensu~

```

The annotated data frames have a lot of redundancy in for the join variables and the `sentence` variable that we want to add to the calculated data frames. We can reduce the redundancy by using the `distinct()` function from the `dplyr` package. In this case we want all observations where `doc_id`, `sentence_id` and `sentence` are distinct. We then select these variables with `distinct()`, as seen in Example 7.41.

Example 7.41.

```

# Reduce annotated data frames to unique sentences
enntt_natives_ann_distinct <-
  enntt_natives_ann |>
  distinct(doc_id, sentence_id, sentence)

enntt_translations_ann_distinct <-
  enntt_translations_ann |>
  distinct(doc_id, sentence_id, sentence)

```

We now have two datasets that are ready to be joined with the calculated datasets. The next step is to join the two. We will employ a left join where the syntactic complexity data frames are on the left and the join variables will be both the `doc_id` and `sentence_id` variables. The code is seen in Example 7.42.

Example 7.42.

```

# Join the native datasets
enntt_natives_transformed_tbl <-
  left_join(
    x = enntt_natives_syn_comp,
    y = enntt_natives_ann_distinct,
    by = c("doc_id", "sentence_id")
  )

# Preview
glimpse(enntt_natives_transformed_tbl)

```

```

> Rows: 10,199
> Columns: 5
> $ doc_id      <chr> "1", "10", "100", "1000", "10000", "1001", "1002", "1003", ~
> $ sentence_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
> $ t_units     <int> 4, 2, 1, 0, 3, 2, 1, 2, 1, 2, 2, 1, 4, 2, 2, 4, 3, 2, 2, 1~
> $ word_len    <int> 21, 25, 27, 15, 40, 43, 29, 23, 13, 30, 33, 9, 68, 35, 16, ~
> $ sentence    <chr> "It is extremely important that action is taken to ensure ~

```

```

# Join the translations datasets
enntt_translations_transformed_tbl <-
  left_join(
    x = enntt_translations_syn_comp,

```

```
  y = enntt_translations_ann_distinct,  
  by = c("doc_id", "sentence_id")  
)  
  
# Preview  
glimpse(enntt_translations_transformed_tbl)
```

The two data frames now have the same columns and we are closer to our final dataset. The next step is to move toward concatenating the two datasets. Before we do that, we need to do some preparation. First, and most important, we need to add a `type` column to each dataset. This column will indicate whether the sentence is a native or a translation. The second is that our `doc_id` does not serve as a unique identifier for the sentences. Only in combination with `sentence_id` can we uniquely identify a sentence.

So our plan will be to add a `type` column to each dataset specifying the values for all the observations in the respective dataset. Then we will concatenate the two datasets. Note, if we combine them before, distinguishing the type will be more difficult. After we concatenate the two datasets, we will add a `doc_id` column that will serve as a unique identifier for the sentences and drop the `sentence_id` column. OK, that's the plan. Let's execute it in Example 7.43.

Example 7.43.

```
# Add a type column
enntt_natives_transformed_tbl <-
  enntt_natives_transformed_tbl |>
  mutate(type = "natives")

enntt_translations_transformed_tbl <-
  enntt_translations_transformed_tbl |>
  mutate(type = "translations")

# Concatenate the datasets
enntt_transformed_tbl <-
```

```

bind_rows(
  enntt_natives_transformed_tbl,
  enntt_translations_transformed_tbl
)

# Overwrite the doc_id column with a unique identifier
enntt_transformed_tbl <-
  enntt_transformed_tbl |>
  mutate(doc_id = row_number()) |>
  select(doc_id, type, t_units, word_len, text = sentence)

# Preview
glimpse(enntt_transformed_tbl)

```

```

> Rows: 20,591
> Columns: 5
> $ doc_id  <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18~
> $ type     <chr> "natives", "natives", "natives", "natives", "natives", "nativ~
> $ t_units  <int> 4, 2, 1, 0, 3, 2, 1, 2, 1, 2, 2, 1, 4, 2, 2, 4, 3, 2, 2, 1, 1~
> $ word_len <int> 21, 25, 27, 15, 40, 43, 29, 23, 13, 30, 33, 9, 68, 35, 16, 53~
> $ text     <chr> "It is extremely important that action is taken to ensure tha~

```

The output of Example 7.43 now looks like Table 7.11. We have a dataset that has the syntactic complexity measures for both the natives and the translations. We can now write this dataset to disk and document it in the data dictionary.

Summary

In this chapter we covered the process of transforming datasets. The goal is to manipulate the curated dataset to make it align better for analysis. We covered various types of transformation procedures from text normalization to data frame merges. In any given research project some or all of these steps will be employed –but not necessarily in the order presented in this chapter. It is not uncommon to mix procedures as well. The etiology of the transformation is as unique as the data that you are working with.

Since you are applying techniques that have a significant factor on the shape and contents of your dataset(s) it is important to perform data checks to ensure that the transformations are working as expected. You may not catch everything, and some things may not be caught until later in the analysis process, but it is important to do as much as you can as early as you can.

In line with the reproducible research principles, it is important to write the transformed dataset to disk and to document it in the data dictionary. This is especially important if you are working with multiple datasets. Good naming conventions also come into play. Choosing descriptive names is so easily overlooked by your present self but so welcomed by your future self.

 more on concluding this part and moving to the next part

This chapter concludes the section on data/ dataset preparation. The next section we turn to analyzing datasets. This is the stage where we interrogate the datasets to derive knowledge and insight either through exploratory, predictive, or inferential methods.

Activities

 Add description of outcomes

Recipe

What: Transforming and documenting datasets^a

How: Read Recipe 7 and participate in the Hypothes.is online social annotation.

Why: To work with the primary types of transformations, tokenization and joins. Tokenization is the process of recasting textual units as smaller textual units. The process of joining datasets aims to incorporate other datasets to augment or filter the dataset of interest.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-7.html>

Lab

What: Dataset manipulation: tokenization and joining datasets^a

How: Clone, fork, and complete the steps in Lab 7.

Why: To gain experience working with coding strategies for transforming datasets using tidyverse functions and regular expressions, practice reading/ writing data from/ to disk, and implement organizational strategies for organizing and documenting a dataset in reproducible fashion.

^a<https://github.com/qtalr/lab-7>

Questions

 Conceptual questions

1. ...
2. ...

 Technical questions

1. ...
2. ...

Part IV

Analysis

In this section we turn to the analysis of datasets, the evaluation of results, and the interpretation of the findings. We will outline the three main types of statistical analyses: Exploratory Data Analysis (EDA), Predictive Data Analysis (PDA), and Inferential Data Analysis (IDA). Each of these analysis types have distinct, non-overlapping aims and therefore should be determined from the outset of the research project and included as part of the research blueprint. The aim of this section is to establish a clearer picture of the goals, methods, and value of each of these approaches.

8 Exploration

 Draft

Ready for review.

The data speaks for itself, but only if we are willing to listen.

— Nate Silver

Outcomes

- Identify when an exploratory data analysis approach is the best fit for a given research project.
- Describe the fundamental methods of descriptive analysis and unsupervised learning, recognizing their strengths in revealing patterns and summarizing data.
- Interpret the basic insights gained from data summarization and pattern recognition, considering how these insights could guide further questions or research.

In this chapter, we examine a wide range of strategies for deriving insight from data in cases where the researcher does not start with a preconceived hypothesis or prediction, but rather the researcher aims to uncover patterns and associations from data allowing the data to guide the trajectory of the analysis. The chapter outlines two main branches of exploratory data analysis: 1) descriptive analysis which statistically and/ or visually summarizes a dataset and 2) unsupervised learning which is a machine learning approach that does not assume any particular relationship between variables in a dataset. Either through descriptive or unsupervised learning methods, exploratory data analysis employs quantitative methods to summarize, reduce, and sort complex datasets and statistically and visually interrogate a dataset in order to provide the researcher novel perspective to be qualitatively assessed.

Lessons

What: Matrices, Exploratory Visualization^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: Learn how to work with matrices to store and analyze numeric data using `quanteda`

and to further your understanding of graphically representing data using `ggplot2` and other packages for more advanced plotting.

^a<https://github.com/qtalr/lessons>

8.1 Orientation

The aim of this section is to provide an overview of exploratory data analysis (EDA). We will delve into various descriptive methods, such as frequency analysis and co-occurrence analysis, which are fundamental tools in linguistic research. However, our exploration won't stop there. We will also integrate modern exploratory methods from unsupervised learning approaches, including clustering, dimensionality reduction, and vector space modeling. This may sound overwhelming, but I will strive to keep explanations clear and concise, ensuring their practicality and relevance to your linguistic inquiries is apparent. To this end, we will provide real-world examples to exemplify the applicability of these methodologies.

8.1.1 Research goal

As discussed in Section 3.2.1 and Section 4.3.1, the goal of exploratory data analysis is to discover, describe, and posit new hypotheses. The researcher does not start with a preconceived hypothesis or prediction, but rather the researcher aims to uncover patterns and associations from data allowing the data to guide the trajectory of the analysis. This analysis approach is best-suited for research where the literature on a research question is limited, or where the researcher is interested in exploring a new research question.

Since the researcher does not start with a preconceived hypothesis, the researcher is not able to test a hypothesis and generalize to a population, but rather the researcher is able to describe the data and provide a new perspective to be qualitatively assessed. This is achieved through an iterative and inductive process of data exploration, where the researcher uses quantitative methods to summarize, reduce, and sort complex datasets and statistically and visually interrogate a dataset letting the data guide the analysis.

8.1.2 Approach

The approach to exploratory data analysis is iterative and inductive. To reign in the analysis, however, it is important to have a research question to guide the analysis. The research question will often be broad and exploratory in nature, but it will provide a framework for the analysis including the unit of analysis and sometimes the units of observation. Yet the units of observation can be modified as needed to address the research question. Furthermore, the methods applied to the data can evolve as the research unfolds. The researcher may start with

a descriptive analysis and then move to an unsupervised learning approach, or vice versa. The researcher may also pivot the approach to explore new questions and new variables. Ultimately, the researcher is guided by the data and the research question, but the researcher is not bound by a preconceived hypothesis or prediction.

With a research question and relevant data in hand, we can look to conduct the analysis. The general workflow for exploratory data analysis is shown in Table 8.1.

Table 8.1: Workflow for exploratory data analysis

| Step | Name | Description |
|------|--------------------|--|
| 1 | Identify | Consider the research question and identify variables of potential interest to provide insight into our question. |
| 2 | Inspect | Check for missing data, outliers, <i>etc.</i> and check data distributions and transform if necessary. |
| 3 | Interrogate | Submit the selected variables to descriptive (frequency, keyword, co-occurrence analysis, <i>etc.</i>) or unsupervised learning (clustering, dimensionality reduction, vector spacing modeling, <i>etc.</i>) methods to provide quantitative measures to evaluate. |
| 4 | Interpret | Evaluate the results and determine if they are valid and meaningful to respond to the research question. |
| 5 | (Optional) Iterate | Repeat steps 1-4 as new questions emerge from your interpretation. |

Let's elaborate on each of these steps. First, we want to consider our research question and identify the variables of potential interest to provide insight to our question. Starting with a transformed dataset means that much of the data preparation has already been done, but we may need to further transform the data, either up front or as we explore the data. In text analysis, this often includes identifying and extracting the linguistic variables of interest, such as words, *n*-grams, sentences, *etc.* Depending on the annotation scheme, other linguistic variables may be of interest, such as part-of-speech tags, syntactic dependencies, semantic roles, *etc.*

We may also want to consider the operational measures of the variables derived from the text, such as frequency, dispersion, co-occurrence, keyness, *etc.* We may also want to consider the other variables in the dataset that may be target for grouping or filtering the dataset, such as speaker information, document information, linguistic unit information, *etc.*

During or after extracting and operationalizing the variables of interest, we want to inspect the dataset to ensure the quality of the data and understand its characteristics. This may include checking for missing data, checking for outliers, checking for errors, checking for inconsistencies, *etc.* We may also want to inspect the distribution of the variables of interest to understand their characteristics. Summary statistics and visualizations, such as those covered in Section 3.1, are useful for inspecting the dataset and also provide a foundation for interrogating the dataset.

Once we have identified the variables of interest and inspected the dataset, we can interrogate the dataset using descriptive analysis and/ or unsupervised learning. Descriptive analysis is a set of methods that statistically and/ or visually summarizes a dataset. Descriptive analysis can be used to describe a dataset and to identify linguistic units (frequency analysis) or co-occurring (co-occurrence analysis) units that are distinctive to a particular group or sub-group in the dataset. Unsupervised learning is a machine learning approach that does not assume any particular relationship between variables in a dataset. It can be used to identify groupings (clustering) in the data including patterning of linguistic units, identifying semantically similar topics (topic modeling), and estimating word context relationships (vector space modeling).

Exploratory methods will produce a set of statistical and/ or visual results. The researcher must interpret these results to determine if they are meaningful and if they provide a new perspective on the research question. Many times the results from one method will lead to new questions which can be explored with other methods. In some cases, the results may not be meaningful and the researcher may need to return to the data preparation stage to modify the dataset or the variables of interest. As the aim of exploratory analysis is just that, to explore, the researcher can pivot the approach to explore new questions and new variables. Ultimately, what is meaningful is determined by the researcher in the light of the research question and the potential insight obtained from the results.

8.2 Analysis

In this section will discuss exploratory data analysis (EDA) for linguists, with a focus on descriptive methods such as frequency analysis and co-occurrence analysis, as well as unsupervised learning approaches such as clustering, topic modelling, and word embedding. To ground the discussion, we will use the the Manually Annotated Sub-Corpus (MASC) of the American National Corpus. The data dictionary for the `masc_transformed` dataset is shown in Table 8.2.

We will work with the MASC as our dataset to approach a task, more than a question. The task will be to identify relevant materials for an English Language Learner (ELL) textbook. This will involve multiple research questions and allow us to illustrate some very fundamental concepts that will emerge across text analysis research.

First, I'll read in the dataset and only keep the variables that will pertain to our task, dropping the `description` and `domain` variables, and preview the dataset in Example 8.1.

Table 8.2: Data dictionary for the MASC dataset.

| variable | name | variable_type | description |
|-------------|----------------|---------------|---|
| doc_id | Document ID | numeric | Unique identifier for each document |
| description | Description | categorical | Description of the content of the document |
| modality | Modality | categorical | The form in which the document is presented (written or spoken) |
| genre | Genre | categorical | The category or type of the document |
| domain | Domain | categorical | The subject or field to which the document belongs |
| term_num | Term Number | numeric | Index number term per document |
| term | Term | categorical | Individual word forms in the document |
| lemma | Lemma | categorical | Base or dictionary form of the term |
| pos | Part of Speech | categorical | Grammatical category of the term (modified PENN Treebank) |

Example 8.1.

```
# Read and subset the MASC dataset
masc_tbl <-
  read_csv("../data/masc/masc_transformed.csv") |>
  select(-description, -domain)

# Preview the MASC dataset
masc_tbl |>
  slice_head(n = 5)

> # A tibble: 5 x 7
>   doc_id modality genre   term_num term      lemma      pos
>   <dbl> <chr>    <chr>     <dbl> <chr>    <chr>    <chr>
> 1     1 Written  Letters      0 December  december  NNP
> 2     1 Written  Letters      1 1998    1998     CD
> 3     1 Written  Letters      2 Your     your     PRP$
> 4     1 Written  Letters      3 contribution contribution NN
> 5     1 Written  Letters      4 to       to       T0
```

From the output in Example 8.1, we should note a couple of things. First the `doc_id` is treated as numeric `<dbl>` and it is not a quantitative variable –we should change this vector type to `<chr>`. Second, at some point in our analysis we may need to recode some of the character variables to factor variables as analysis methods may require this.

Example 8.2.

Table 8.3: Summary of the MASC dataset.

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---------------|-----------|---------------|-----|-----|-------|----------|------------|
| doc_id | 0 | 1 | 1 | 3 | 0 | 392 | 0 |
| modality | 0 | 1 | 6 | 7 | 0 | 2 | 0 |
| genre | 0 | 1 | 4 | 12 | 0 | 18 | 0 |
| term | 25 | 1 | 1 | 99 | 0 | 39470 | 0 |
| lemma | 4 | 1 | 1 | 99 | 0 | 28008 | 0 |
| pos | 0 | 1 | 2 | 5 | 0 | 39 | 0 |

```
# Change doc_id to character
masc_tbl <-
  mascot |>
  mutate(doc_id = as.character(doc_id))
```

To get a better sense of distribution of the dataset, let's use `skim()` from the `skimr` package to generate a summary of the dataset. In particular, let's just focus on the character variables by using `yank("character")`, as seen in Example 8.3.

Example 8.3.

```
# Load package
library(skimr)

# Generate summary of the MASC dataset
masc_tbl_skm <-
  mascot |>
  skim()

# Pull character variables
masc_tbl_skm |>
  yank("character") |>
  kable()
```

Looking at Table 8.3, we see that there are 392 documents, two modalities, 18 genres, over 30k unique terms (which are words), over 28k lemmas (word base forms), and 39 distinct part-of-speech tags.

8.2.1 Descriptive analysis

Descriptive analysis includes common techniques such as frequency analysis to determine the most frequent words or phrases, dispersion analysis to see how terms or topics are distributed throughout a document or corpus, keyword analysis to identify distinctive terms, and/ or co-occurrence analysis to see what terms tend to appear together.

Using the MASC dataset, we will entertain questions such as:

- What are the most common terms a beginning ELL should learn?
- Are there term differences between spoken and written discourses that should be emphasized?
- What are the most common verb particle constructions?

Along the way, we will introduce some fundamental concepts in text analysis such as tokens and types and frequency, dispersion, and co-occurrence measures. In addition, we will apply various descriptive analysis techniques and visualizations to explore the dataset and identify new questions and new variables of interest.

Frequency analysis

At its core, frequency analysis is a descriptive method that counts the number of times a linguistic unit, or term, (*i.e.* word, *n*-gram, sentence, *etc.*) occurs in a dataset. The results of frequency analysis can be used to describe the dataset and to identify terms that are linguistically distinctive or distinctive to a particular group or sub-group in the dataset.

Raw frequency

Let's consider what the most common words in the MASC dataset are as a starting point to making inroads on our task by identifying relevant vocabulary for an ELL textbook.

In the `masc_tbl` data frame we have the linguistic unit `term` which corresponds to the word-level annotation of the MASC. The `lemma` corresponds to the base form of each term, for words with inflectional morphology, the lemma is the word sans the inflection (*e.g.* is - be, are - be). For other words, the `term` and the `lemma` will be the same (*e.g.* the - the, in - in). These two variables pose a choice point for us: do we consider words to be the actual forms or the base forms? There is an argument to be made for both. In this case I will operationalize our linguistic unit as the `lemma` variable, as this will allow us to group words with inflectional morphology together.

To perform a basic word frequency analysis, we start by using the `count()` function from the `dplyr` package to count the number of times each lemma occurs in the dataset. We'll sort by the most frequent lemmas, as seen in Example 8.4.

Example 8.4.

```
# Lemma count, sorted
masc_tbl |>
  count(lemma, sort = TRUE)

> # A tibble: 28,009 x 2
>   lemma      n
>   <chr> <int>
> 1 ,        27113
> 2 .        26258
> 3 the     26137
> 4 be       19466
> 5 to       13548
> 6 and      12528
> 7 of       12005
> 8 a        10480
> 9 in       8374
> 10 i       7783
> # i 27,999 more rows
```

The output of this frequency tabulation in Example 8.4 is a data frame with two columns: `lemma` and `n`. The `lemma` column contains the unique lemmas in the dataset, and the `n` column contains the frequency of each lemma. The data frame is sorted in descending order by the frequency of lemmas. Now the result includes over 28,000 rows –which corresponds to the number of unique lemmas in the dataset.

At this point, it is important to define a few key concepts that are fundamental to working with text. First, a **term** is a defined linguistic unit extracted from a corpus. In our dataset, the terms are words, such as ‘the’, ‘houses’, ‘are’. A lemma is an annotated recoding of words which represent the uninflected base form of a word. In either case, the term or lemma is an instance of a linguistic unit. These instances are called **tokens**. When we count the number of times a term or lemma occurs in a dataset, we are counting the number of tokens (`n`), such as in Example 8.4. Now, the list of unique linguistic units is a list of **types** (`lemma`). By definition, then, there will always be at least as many tokens as types, but more often than not (many) more tokens than types.

Our first pass at calculating lemma frequency in Example 8.4 should bring something else to our attention. As we can see among the most frequent lemmas are non-words such as `,`, and `..`. As you can imagine, given the conventions of written and transcriptional language, these types are very frequent. For a frequency analysis focusing on words, however, we should probably remove them. Thinking ahead, there may also be other non-words that we want to

remove, such as symbols, numbers, *etc.* Let's take a look at Figure 8.1, where I've counted the part-of-speech tags `pos` in the dataset to see what other non-words we might want to remove.

Example 8.5.

```
# Part-of-speech tags
masc_tbl |>
  count(pos) |>
  ggplot(
    aes(x = reorder(pos, desc(n)), y = n)
  ) +
  geom_col() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(x = "Part-of-speech tags", y = "Token frequency")
```

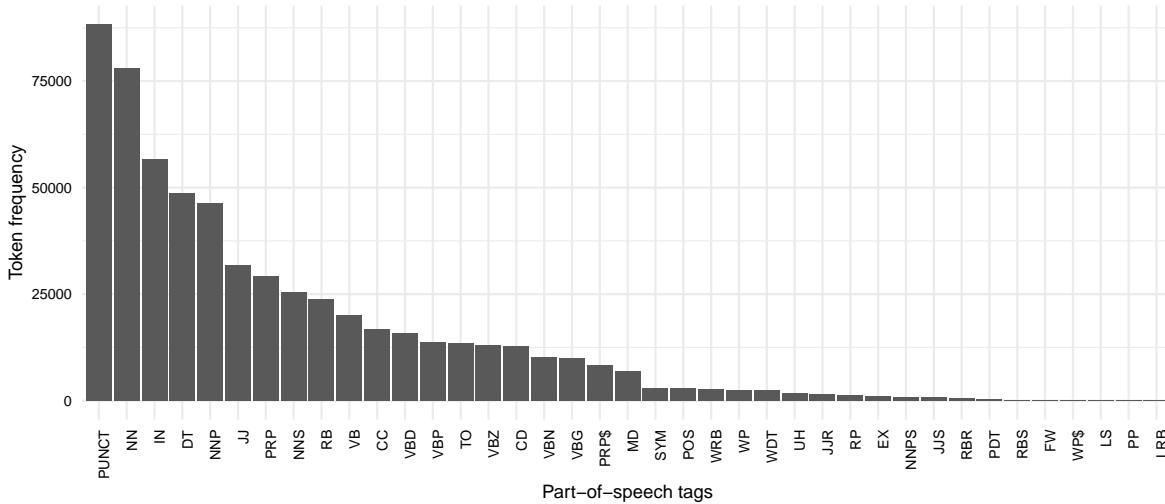


Figure 8.1: Part-of-speech tags in the MASC dataset.

Consulting the PENN Tagset online¹, we can see that the `pos` variable includes a number of non-words or other elements to exclude including:

- ‘CD’ - Cardinal number
- ‘FW’ - Foreign word
- ‘LS’ - List item marker
- ‘SYM’ - Symbol

¹<https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html>

This modified tagset has grouped the punctuation tags into a single tag, ‘PUNCT’.

We can use this information to remove lemmas that are tagged with either of these values. We can do this by filtering the data frame to only include lemmas that are not tagged with the `pos` values listed above, as seen in Example 8.6.

Example 8.6.

```
# Filter out lemmas with PUNCT or SYM for pos
masc_tbl <-
  masc_tbl |>
  filter(!(pos %in% c("CD", "FW", "LS", "SYM", "PUNCT")))

# Lemma count, sorted (again)
masc_tbl |>
  count(lemma, sort = TRUE)

> # A tibble: 26,164 x 2
>   lemma     n
>   <chr> <int>
> 1 the     26137
> 2 be      19466
> 3 to      13548
> 4 and     12528
> 5 of      12005
> 6 a       10461
> 7 in      8374
> 8 i       7783
> 9 that    7082
> 10 you    5276
> # i 26,154 more rows
```

Now we are only viewing the most frequent words in the dataset, which reduces the number of observations to around 26k. Let’s now explore the frequency distribution of the tokens. In Figure 8.2, I’ve created three plots which include: 1) all the types, 2) the top 100 types, and 3) the top 10 types in the dataset.

```
# [ ] consider how to present the 'all types' plot better, more concisely

# Plot lemma count for all types
masc_tbl |>
```

```
count(lemma) |>
arrange(desc(n)) |>
ggplot(aes(x = reorder(lemma, desc(n)), y = n)) +
geom_col() +
labs(x = "Types", y = "Token frequency") +
theme(axis.text.x = element_blank())

# Plot lemma count for top 100 types
masc_tbl |>
count(lemma) |>
arrange(desc(n)) |>
slice_head(n = 100) |>
ggplot(aes(x = reorder(lemma, desc(n)), y = n)) +
geom_col() +
labs(x = "Types", y = "Token frequency") +
theme(axis.text.x = element_text(angle = 90, hjust = 1.3))

# Plot lemma count for top 10 types
masc_tbl |>
count(lemma) |>
arrange(desc(n)) |>
slice_head(n = 10) |>
ggplot(aes(x = reorder(lemma, desc(n)), y = n)) +
geom_col() +
labs(x = "Types", y = "Token frequency") +
theme(axis.text.x = element_text(angle = 65, hjust = 1.3))
```

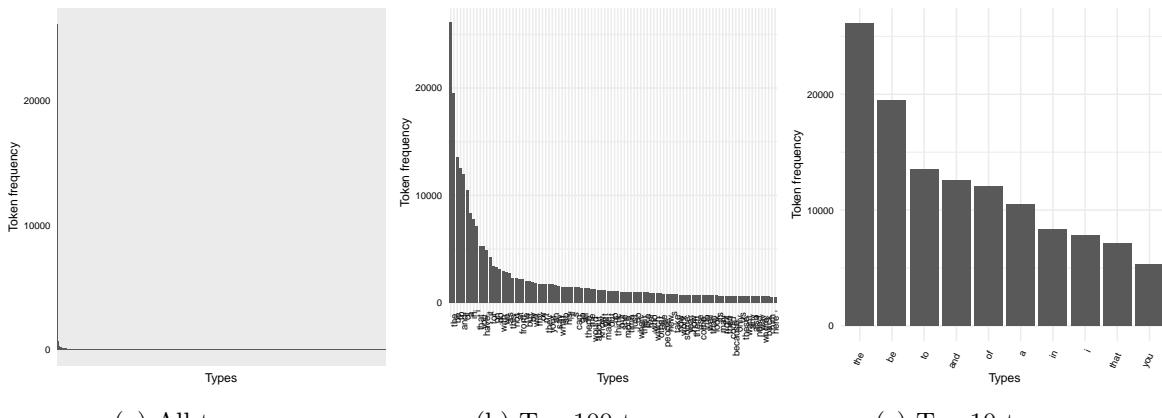


Figure 8.2: Frequency plots of tokens in the MASC dataset

The distributions we see in Figure 8.2 are highly right-skewed (in Figure 8.2a in a very extreme way!). This is typical of natural language distributions, notably documented by George Kingsley Zipf (Zipf 1949). This type of distribution approaches the theoretical Zipf distribution. A Zipf (or Zipfian) distribution is characterized by the fact that the frequency of any word is inversely proportional to its rank in the frequency table. In other words, the most frequent word occurs approximately twice as often as the second most frequent word, three times as often as the third most frequent word, and so on.

As we can see, our distributions do not follow the Zipf distribution exactly. This is because the Zipf distribution is a theoretical distribution, and the actual distribution of words in a corpus is affected by various sampling factors, including the size of the corpus. The larger the corpus, the closer the distribution will be to the Zipf distribution.

➊ Dive deeper

As stated above, Zipfian distributions are typical of natural language and are observed at various linguistic levels. This is because natural language is a complex system, and complex systems tend to exhibit Zipfian distributions. Other examples of complex systems that exhibit Zipfian distributions include the size of cities, the frequency of species in ecological communities, the frequency of links in the World Wide Web, *etc.*

The observation captured in the Zipf distribution is key to understanding quantitative text analysis. It demonstrates that most of the types in a corpus occur (relatively) infrequently, while a small number of types occur very frequently. In fact, if we calculate the cumulative frequency of the lemmas in the `masc_tbl` data frame, we can see that the top 10 types account for over 20% of the lemmas used in the dataset – by 100 types that increases to over 40%, as seen in Example 8.7.

Example 8.7.

```
# Calculate cumulative frequency
lemma_cumul_freq <-
  masc_tbl |>
  count(lemma) |>
  arrange(desc(n)) |>
  mutate(cumulative = cumsum(n)) |>
  mutate(percent = cumulative / sum(n))

lemma_cumul_freq |>
  slice_head(n = 2000) |>
  ggplot(aes(x = reorder(lemma, desc(n)), y = percent)) +
  geom_col() +
  geom_vline(xintercept = 10, linetype = "dashed") +
```

```

geom_vline(xintercept = 100, linetype = "dashed") +
# annotate("text", x = 10+10, y = 0.5, label = "10 lemmas") +
# annotate("text", x = 100+10, y = 0.5, label = "100 lemmas") +
scale_y_continuous(labels = scales::percent, limits = c(0, 1)) +
labs(x = "Types", y = "Cumulative frequency percent") +
theme(axis.text.x = element_blank())

```

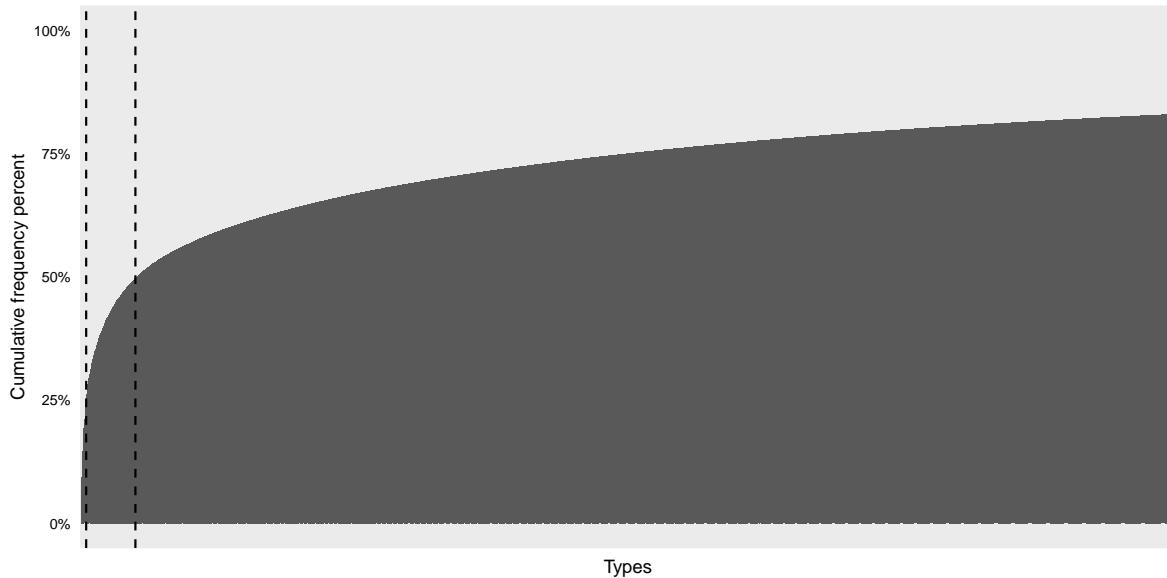


Figure 8.3: Cumulative frequency of lemmas in the MASC dataset

If we look at the types that appear within the first 100 most frequent, you can likely also appreciate another thing about language use. Let's list the top 100 types in Example 8.8.

Example 8.8.

```

# Top 100 types
lemma_cumul_freq |>
  slice_head(n = 100) |>
  pull(lemma) |>
  matrix(ncol = 10, byrow = TRUE) |>
  kable(col.names = NULL)

```

For the most part, the most frequent words are not content words, but rather function words (*e.g.* determiners, prepositions, pronouns, auxiliary verbs). Function words include a closed

Table 8.4: Top 100 lemma types in the MASC dataset.

| | | | | | | | | | |
|-------|-------|-------|-------|-------|--------|---------|------|-------|-------|
| the | be | to | and | of | a | in | i | that | you |
| have | it | for | on | do | with | we | as | this | not |
| at | from | he | but | by | will | my | or | n't | they |
| your | an | say | what | so | his | if | 's | can | go |
| all | there | me | would | about | know | get | make | out | up |
| think | our | she | more | time | just | no | when | their | like |
| her | who | which | other | see | people | new | s | take | now |
| work | some | year | how | them | use | come | into | well | than |
| look | its | may | right | then | could | because | only | us | these |
| want | any | also | need | way | where | back | him | here | ' |

class of relatively few words that are used to express grammatical relationships between content words. It then is no surprise that they are the comprise many of the most frequent words in a corpus.

Another key observation is that among the most frequency content words (*e.g.* nouns, verbs, adjectives, adverbs) are words that are quite semantically generic –that is, they are words that are used in a wide range of contexts and take a wide range of meanings. Take for example the adjective ‘good’. It can be used to describe a wide range of nouns, such as ‘good food’, ‘good people’, ‘good times’, *etc.* A sometimes near-synonym of ‘good’, for example ‘good student’, is the word ‘studious’. Yet, ‘studious’ is not as frequent as ‘good’ as it is used to describe a narrower range of nouns, such as ‘studious student’, ‘studious scholar’, ‘studious researcher’, *etc.* In this way, ‘studious’ is more semantically specific than ‘good’.

💡 Consider this

Based on what you now know about the expected distribution of words in a corpus, what if your were asked to predict what the most frequency English word used is in each U.S. State? What would you predict? How confident would you be in your prediction? What if you were asked to predict what the most frequency word used is in the language of a given country? What would you want to know before making your prediction?

So common across corpus samples, in some analyses these usual suspects of the most common words are considered irrelevant and are filtered out. In our ELL materials task, however, we might exclude them for this simple fact that it will be a given that we will teach these words given their grammatical importance. If we want to focus on the most common content words, we can filter out the function words.

One approach to filtering out these words is to use a pre-determined list of **stopwords**. The **tidytext** package includes a data frame **stop_words** of stopword lexicons for English. We can select a lexicon from **stop_words** and use **anti_join()** to filter out the words that appear in the

Table 8.5: Frequency of tokens in the MASC dataset after filtering out stopwords

| | | | | | | | | | |
|---------|----------|-----------|---------|---------|---------|---------|---------|-------------|-----------|
| n't | 's | make | time | people | work | year | back | ' | find |
| give | day | thing | jack | man | yeah | good | call | world | president |
| state | question | service | change | life | leave | subject | set | long | place |
| write | show | child | end | feel | hand | system | school | information | part |
| group | follow | run | support | today | point | read | provide | uh | send |
| turn | include | talk | fact | & | live | put | word | number | start |
| law | case | company | money | great | open | home | city | issue | woman |
| job | american | important | result | book | hear | sparrow | house | problem | um |
| america | walk | family | begin | country | date | face | friend | report | move |
| order | head | id | watch | form | program | market | week | area | figure |

word variable from the lemma variable in the `masc_tbl` data frame. In Example 8.9, I perform this filtering and then re-run the frequency analysis for the top 100 lemmas.

Example 8.9.

```
# Load package
library(tidytext)

# Select stopword lexicon
stopwords <-
  stop_words |>
  filter(lexicon == "SMART")

# Filter out stop words
anti_join(
  x = masc_tbl,
  y = stopwords,
  by = c("lemma" = "word")
) |>
  count(lemma, sort = TRUE) |>
  slice_head(n = 100) |>
  pull(lemma) |>
  matrix(ncol = 10, byrow = TRUE) |>
  as_tibble() |>
  kable(col.names = NULL)
```

The resulting list in Table 8.5 paints a different picture of the most frequent words in the dataset. The most frequent words are now content words, and included in most frequent words are more semantically specific words.

Eliminating words in this fashion, however, may not always be the best approach. Available lists of stopwords vary in their contents and are determined by other researchers for other potential uses. We may instead opt to create our own stopword list that is tailored to the task, or we may opt to use a statistical approach based on their distribution in the dataset using a combination of frequency and dispersion measures, as we will see in [the next section.]

For our case, however, we have another strategy to apply. Since our task is to identify relevant vocabulary, beyond the fundamental function words in English, we can use the part-of-speech tags to reduce our dataset to just the content words, that is nouns, verbs, adjectives, and adverbs. We need to consult the Penn Tagset again, to ensure we are selecting the correct tags. I will assign this data frame to `masc_content_tbl` to keep it separate from our main data frame `masc_tbl`, seen in Example 8.10.

Example 8.10.

```
# Penn Tagset for content words
# Nouns: NN, NNS,
# Verbs: VB, VBD, VBG, VBN, VBP, VBZ
# Adjectives: JJ, JJR, JJS
# Adverbs: RB, RBR, RBS

content_pos <- c("NN", "NNS", "VB", "VBD", "VBG", "VBN", "VBP", "VBZ", "JJ",
  "JJR", "JJS", "RB", "RBR", "RBS")

# Select content words
masc_content_tbl <-
  masc_tbl |>
  filter(pos %in% content_pos)
```

We now have reduced the number of observations by 50% focusing on the content words. We are getting closer to identifying the vocabulary that we want to include in our ELL materials, but we will need some more tools to help us identify the most relevant vocabulary.

Dispersion

Dispersion is a measure of how evenly distributed a linguistic unit is across a dataset. This is a key concept in text analysis, as important as frequency. It is important to recognize that frequency and dispersion are measures of different characteristics. We can have two words that occur with the same frequency, but one word may be more evenly distributed across a dataset than the other. Depending on the researcher's aims, this may be an important distinction to make. For our task, it is likely the case that we want to capture words that are well-dispersed across the dataset as words that have a high frequency and a low dispersion

tend to be connected to a particular context, whether that be a particular genre, a particular speaker, a particular topic, *etc.* In other research, aim may be the reverse; to identify words that are highly frequent and highly concentrated in a particular context to identify words that are distinctive to that context.

This a wide variety of measures that can be used to estimate the distribution of types across a dataset. Let's focus on three measures: document frequency (df), inverse document frequency (idf), and Gries' Deviation of Proportions (dp).

The most basic measure is **document frequency** (df). This is the number of documents in which a type appears at least once. For example, if a type appears in 10 documents, then the document frequency is 10. This is a very basic measure, but it is a good starting point.

A nuanced version of document frequency is **inverse document frequency** (idf). This measure takes the total number of documents and divides it by the document frequency. This results in a measure that is inversely proportional to the document frequency. That is, the higher the document frequency, the lower the inverse document frequency. This measure is often log-transformed to spread out the values.

One thing to consider about df and idf is that neither takes into account the length of the documents in which the type appears nor the spread of the type within each document. To take these factors into account, we can use Gries' Deviation of Proportions (dp) measure (S. T. Gries 2023, 87–88). The dp measure is calculated as the difference between the proportion of a tokens in a document and tokens in the corpus. The metric can be subtracted from 1 to create a normalized measure of dispersion ranging between 0 and 1, with lower values being more dispersed.

Let's consider how these measures differ with three scenarios:

Imagine a type with a token frequency of 100 appears in each of the 10 documents in a corpus.

A. Each of the documents is 100 words long. The type appears 10 times in each document. B. Each of the documents is 100 words long. But now the type appears once in 9 documents and 91 times in 1 document. C. Nine of the documents constitute 99% of the corpus. The type appears once in each of the 9 documents and 91 times in the 10th document.

Scenario A is the most dispersed, scenario B is less dispersed, and scenario C is the least dispersed. Yet, the type's df and idf scores will be the same. But the dp score will reflect increasing concentration of the type from A to B to C. You may wonder why we would want to use df or idf at all. The answer is some combination of the fact that they are computationally less expensive to calculate, they are widely used (especially idf), and/ or in many practical situations they often highly correlated with dp .

So for our task we will use dp as our measure of dispersion. The `qtlrkit` package includes the `calc_type_metrics()` function which calculates, among other metrics, the dispersion metrics

df , idf , and/ or dp . Let's select dp and assign the result to `masc_lemma_disp`, as seen in Example 8.11.

Example 8.11.

```
# Load package
library(qtalrkit)

# Calculate deviance of proportions (DP)
masc_lemma_disp <-
  masc_content_tbl |>
  calc_type_metrics(
    type = lemma,
    documents = doc_id,
    dispersion = "dp"
  ) |>
  arrange(dp)

# Preview
masc_lemma_disp |>
  slice_head(n = 10)
```

```
> # A tibble: 10 x 3
>   type     n     dp
>   <chr> <dbl> <dbl>
> 1 be     19231 0.123
> 2 have   5136  0.189
> 3 not    2279  0.240
> 4 make   1149  0.266
> 5 other   882  0.270
> 6 more   1005  0.276
> 7 take   769  0.286
> 8 only   627  0.286
> 9 time   931  0.314
> 10 see    865  0.327
```

We would like to identify lemmas that are frequent and well-dispersed. But an important question arises, what is the threshold for frequency and dispersion that we should use to identify the lemmas that we want to include in our ELL materials?

💡 Consider this

You may be wondering why the Inverse Document Frequency is, in fact, the inverse of the document counts, instead of just a count of the documents that each type appears in. The *idf* is a very common measure in machine learning that is used in combination with (term) frequency to calculate the $tf - idf$ (term frequency-inverse document frequency) measure. That is, the product of the frequency of a term and the inverse document frequency of the term. This serves as a weighting measure that lowers the $tf - idf$ score for terms that are frequent across documents and increases the $tf - idf$ score for terms that are infrequent across documents.

Consider what types will end up with a high or a low $tf - idf$ score. What use(s) could this measure have for distinguishing between types in a corpus?

Hint: consider the earlier discussion of stopword lists.

There are statistical approaches to identifying natural breakpoints, including clustering, but a visual inspection is often good enough for practical purposes. Let's create a density plot to see if there is a natural break in the distribution of our dispersion measure, as seen in Figure 8.4.

Example 8.12.

```
# Density plot of dp
masc_lemma_disp |>
  ggplot(aes(x = dp)) +
  geom_density() +
  scale_x_continuous(breaks = seq(0, 1, .1)) +
  labs(x = "Deviation of Proportions")
```

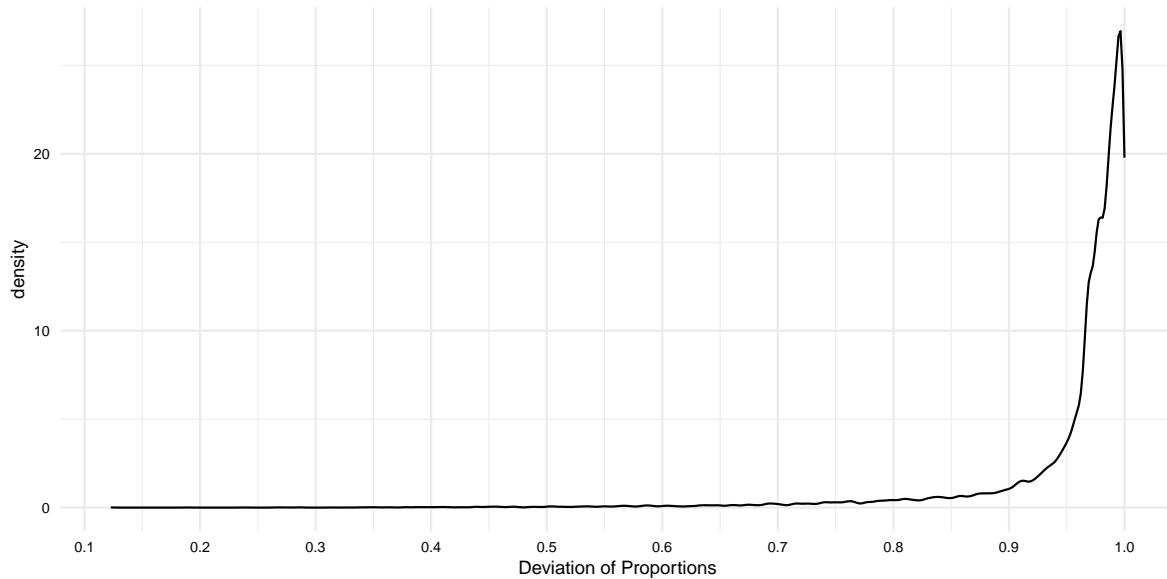


Figure 8.4: Density plot of Deviation of Proportions for lemmas in the MASC dataset

What we are looking for is an elbow in the distribution of dispersion measures. In Figure 8.4, we can see that there is distinctive bend in the distribution between .85 and .97. We can split the difference and use this as a threshold to filter out lemmas that are less dispersed. In Example 8.13, I filter out lemmas that have a dispersion measure less than .91. Then in Table 8.6, I preview the top and bottom 50 lemmas in the dataset.

Example 8.13.

```

# Filter for lemmas with dp <= .91
masc_lemma_disp_thr <-
  masc_lemma_disp |>
  filter(dp <= .91) |>
  arrange(desc(n))

# Preview top
masc_lemma_disp_thr |>
  slice_head(n = 50) |>
  pull(type) |>
  matrix(ncol = 10, byrow = TRUE) |>
  kable(col.names = NULL)

# Preview bottom
masc_lemma_disp_thr |>

```

Table 8.6: Frequency of tokens in the MASC dataset after filtering out lemmas with a Deviation of Proportions less than .91

| (a) ?(caption) | | | | | | | | | |
|----------------|-------|------|------|------|-------|------|--------|-------|------|
| be | have | do | not | n't | say | go | know | get | make |
| think | more | just | time | so | other | see | people | take | now |
| work | year | come | use | well | look | then | right | only | want |
| also | way | need | back | here | new | find | give | thing | tell |
| t | first | help | day | many | man | ask | very | much | even |

Top 50 lemmas

| (b) ?(caption) | | | | | | | | | |
|----------------|------------|-----------|------------|------------|------------|--------------|-----------|-------|--|
| fortunately | ignorance | liability | brave | summarize | liberty | wound | nostalgic | accid | |
| dump | sting | tuition | unleash | blur | going | devote | shy | proto | |
| instrument | mainstream | awaken | prosperous | resistance | awkward | alright | proximity | presi | |
| triumph | wildly | hook | buzz | absurd | afterwards | evolutionary | sandy | rethi | |
| harsh | dismiss | fetch | presume | qualify | eve | envy | interfere | stric | |

Bottom 50 lemmas

```
slice_tail(n = 50) |>
  pull(type) |>
  matrix(ncol = 10) |>
  kable(col.names = NULL)
```

We now have a good candidate list of common vocabulary that is spread well across the corpus.

Relative frequency

Gauging frequency and dispersion across the entire corpus is a good starting point for any frequency analysis, but it is often the case that we want to compare the frequency and dispersion of linguistic units across corpora or sub-corpora.

In the case of the MASC dataset, for example, we may want to compare metrics across the two modalities or the various genres. Simply comparing frequency counts across these sub-corpora is not a good approach, and can be misleading, as the sub-corpora may vary in size. For example, if one sub-corpus is twice as large as another sub-corpus, then, all else being equal, the frequency counts will be twice as large in the larger sub-corpus. This is why we use relative frequency measures, which are normalized by the size of the sub-corpus.

💡 Consider this

A variable in the MASC dataset that has yet to be used is the `pos` part-of-speech variable. How could we use this variable to refine our frequency and dispersion analysis of lemma types?

Hint: consider lemma forms that may be tagged with different parts-of-speech.

To normalize the frequency of linguistic units across sub-corpora, we can use the **relative frequency** measure. This is the frequency of a linguistic unit divided by the total number of linguistic units in the sub-corpus. This bakes in the size of the sub-corpus into the measure. The notion of relative frequency is key to all research working with text, as it is the basis for the statistical approach to text analysis where comparisons are made.

There are some field-specific terms that are used to refer to relative frequency measures. For example, in information retrieval and Natural Language Processing, the relative frequency measure is often referred to as the **term frequency**. In corpus linguistics, the relative frequency measure is often modified slightly to include a constant (*e.g.* $rf * 100$) which is known as the **observed relative frequency**. Although the observed relative frequency per number of tokens is not strictly necessary, it is often used to make the values more interpretable as we can now talk about an observed relative frequency of 1.5 as a linguistic unit that occurs 1.5 times per 100 linguistic units.

Let's consider how we might compare the frequency and dispersion of lemmas across the two modalities in the MASC dataset, spoken and written. To make this a bit more interesting and more relevant, let's add the `pos` variable to our analysis. The intent, then, will be to identify lemmas tagged with particular parts of speech that are particularly indicative of each of the modalities.

We can do this by collapsing the `lemma` and `pos` variables into a single variable, `lemma_pos`, with the `str_c()` function, as seen in Example 8.14.

Example 8.14.

```
# Collapse lemma and pos into type
masc_content_tbl <-
  masc_content_tbl |>
  mutate(lemma_pos = str_c(lemma, pos, sep = "_"))

# Preview
masc_content_tbl |>
  slice_head(n = 10)

> # A tibble: 10 × 8
```

```

>   doc_id modality genre  term_num term          lemma          pos  lemma_pos
>   <chr>  <chr>    <chr>     <dbl> <chr>          <chr>          <chr> <chr>
> 1 1     Written  Letters      3 contribution contribution NN  contributio~
> 2 1     Written  Letters      7 mean          mean          VB  mean_VB
> 3 1     Written  Letters      8 more          more          JJR  more_JJR
> 4 1     Written  Letters     12 know          know          VB  know_VB
> 5 1     Written  Letters     15 help          help          VB  help_VB
> 6 1     Written  Letters     17 see           see           VB  see_VB
> 7 1     Written  Letters     19 much          much          JJ  much_JJ
> 8 1     Written  Letters     21 contribution contribution NN  contributio~
> 9 1     Written  Letters     22 means          mean          VBZ mean_VBZ
> 10 1    Written  Letters    25 'm            be            VBP be_VBP

```

Now this will increase the number of lemma types in the dataset as we are now considering lemmas where the same lemma form is tagged with different parts-of-speech.

Getting back to calculating the frequency and dispersion of lemmas in each modality, we can use the `calc_type_metrics()` function with `lemma_pos` as our type argument. We will, however, need to apply this function to each sub-corpus independently and then concatenate the two data frames. This function returns a (raw) frequency measure by default, but we can specify the `frequency` argument to `rf` to calculate the relative frequency of the linguistic units as in Example 8.15.

Example 8.15.

```

# Calculate relative frequency
# Spoken
masc_spoken_metrics <-
  masc_content_tbl |>
  filter(modality == "Spoken") |>
  calc_type_metrics(
    type = lemma_pos,
    documents = doc_id,
    frequency = "rf",
    dispersion = "dp"
  ) |>
  mutate(modality = "Spoken") |>
  arrange(desc(n))

# Written
masc_written_metrics <-
  masc_content_tbl |>

```

```

filter(modality == "Written") |>
calc_type_metrics(
  type = lemma_pos,
  documents = doc_id,
  frequency = "rf",
  dispersion = "dp"
) |>
mutate(modality = "Written") |>
arrange(desc(n))

# Concatenate spoken and written metrics
masc_metrics <-
bind_rows(masc_spoken_metrics, masc_written_metrics)

# Preview
masc_metrics |>
slice_head(n = 10)

```

```

> # A tibble: 10 x 5
>   type      n      rf      dp modality
>   <chr>    <dbl>    <dbl>    <dbl> <chr>
> 1 be_VBZ    2612  0.0489  0.0842 Spoken
> 2 be_VBP    1282  0.0240  0.111  Spoken
> 3 be_VBD    1020  0.0191  0.301  Spoken
> 4 n't_RB    829   0.0155  0.139  Spoken
> 5 have_VBP  766   0.0143  0.152  Spoken
> 6 do_VBP    728   0.0136  0.180  Spoken
> 7 be_VB     655   0.0123  0.147  Spoken
> 8 not_RB    638   0.0119  0.137  Spoken
> 9 just_RB   404   0.00756 0.267  Spoken
> 10 so_RB    387   0.00725 0.357  Spoken

```

With the `rf` measure, we are now in a position to compare ‘apples to apples’, as you might say. We can now compare the relative frequency of lemmas across the two modalities. Let’s preview the top 10 lemmas in each modality, as seen in Example 8.16.

Example 8.16.

```

# Preview top 10 lemmas in each modality
masc_metrics |>

```

```

group_by(modality) |>
  slice_max(n = 10, order_by = rf) |>
  ungroup()

> # A tibble: 20 x 5
>   type      n      rf      dp modality
>   <chr>    <dbl>   <dbl>   <dbl> <chr>
> 1 be_VBZ    2612  0.0489  0.0842 Spoken
> 2 be_VBP    1282  0.0240  0.111  Spoken
> 3 be_VBD    1020  0.0191  0.301  Spoken
> 4 n't_RB    829   0.0155  0.139  Spoken
> 5 have_VBP  766   0.0143  0.152  Spoken
> 6 do_VBP    728   0.0136  0.180  Spoken
> 7 be_VB     655   0.0123  0.147  Spoken
> 8 not_RB    638   0.0119  0.137  Spoken
> 9 just_RB   404   0.00756 0.267  Spoken
> 10 so_RB    387   0.00725 0.357  Spoken
> 11 be_VBZ   4745  0.0248  0.230  Written
> 12 be_VBD   3317  0.0173  0.366  Written
> 13 be_VBP   2617  0.0137  0.237  Written
> 14 be_VB    1863  0.00974 0.218  Written
> 15 not_RB   1640  0.00858 0.259  Written
> 16 have_VBP 1227  0.00642 0.290  Written
> 17 n't_RB   905   0.00473 0.540  Written
> 18 have_VBD 859   0.00449 0.446  Written
> 19 have_VBZ 777   0.00406 0.335  Written
> 20 say_VBD   710   0.00371 0.609  Written

```

We can appreciate, now, that there are similarities and a few differences between the most frequent lemmas for each modality. First, there are similar lemmas in written and spoken modalities, such as ‘be’, ‘have’, and ‘not’. Second, the top 10 include verbs and adverbs. Now we are looking at the most frequent types, so it is not surprising that we see more in common than not. However, looking close we can see that contracted forms are more frequent in the spoken modality, such as ‘isn’t’, ‘don’t’, and ‘can’t’ and that ordering of the verb tenses differs to some degree. Whether these are important distinctions for our task is something we will need to consider.

We can further cull our results by filtering out lemmas that are not well-dispersed across the sub-corpora. Although it may be tempting to use the threshold we used earlier, we should consider that the size of the sub-corpora are different and the distribution of the dispersion measure may be different. With this in mind, we need to visualize the distribution of the dispersion measure for each modality, as seen in Figure 8.5.

```
# Density plot of dp by modality
masc_metrics |>
  ggplot(aes(x = dp)) +
  geom_density(alpha = .5) +
  scale_x_continuous(breaks = seq(0, 1, .1)) +
  labs(x = "Deviation of Proportions", y = "Density") +
  facet_wrap(~ modality, ncol = 2, scales = "free_x")
```

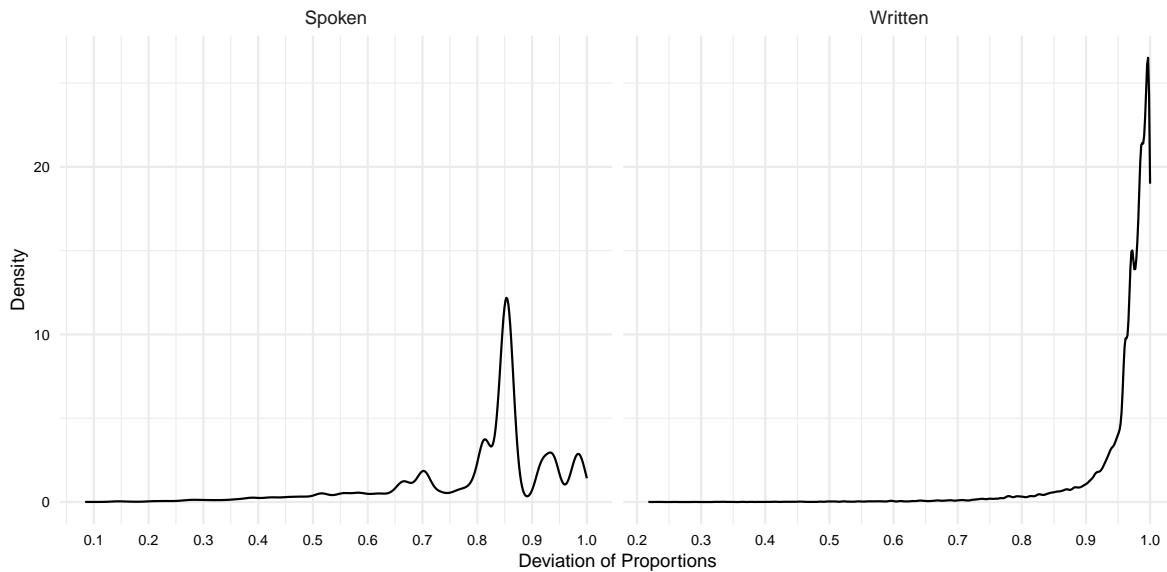


Figure 8.5: Density plot of Deviation of Proportions for lemmas in the MASC dataset by modality

As expected, the density plots point to different thresholds for each modality. The written subcorpus follows closely with the previous distribution, but the spoken subcorpus has more than one bend in the distribution. Why are there multiple peaks in the density plot? It points to some level of inconsistency in the spoken data, either potentially some level of context-dependent language use (genres, topics, speech styles) or it could be due to the fact that the spoken subcorpus' size is too small to provide a reliable distribution.

In any case, we can estimate the threshold for the spoken corpus making use of the largest peak in the distribution as the reference point. With this approach in mind, let's maintain the .91 threshold for the written subcorpus and use a .79 threshold for the spoken subcorpus. Let's filter out lemmas that have a dispersion measure less than .91 for the written subcorpus and less than .79 for the spoken subcorpus, as seen in Example 8.17.

Example 8.17.

```
# Filter for lemmas with
# dp <= .91 for written and
# dp <= .79 for spoken
masc_metrics_thr <-
  masc_metrics |>
  filter(
    (modality == "Written" & dp <= .91) |
    (modality == "Spoken" & dp <= .79)
  ) |>
  arrange(desc(rf))
```

Filtering the less-dispersed types reduces the dataset from 33637 to 4860 observations. This will provide us with a more succinct list of common and well-dispersed lemmas that are used in each modality.

As much as the frequency and dispersion measures can provide us with a good starting point, it does not provide an understanding of what types are more indicative of a particular sub-corpus, modality subcorpora in our case. We can do this by calculating the log odds ratio of each lemma in each modality.

The **log odds ratio** is a measure that quantifies the difference between the frequencies of a type in two corpora or sub-corpora. In spirit and in name, it compares the odds of a type occurring in one corpus versus the other. The values range from negative to positive infinity, with negative values indicating that the type is more frequent in the first corpus and positive values indicating that the lemma is more frequent in the second corpus. The magnitude of the value indicates the strength of the association.

The `tidylo` package provides a convenient function `bind_log_odds()` to calculate the log odds ratio, and a weighed variant, for each type in each sub-corpus. Let's use this function to calculate the log odds ratio for each lemma in each modality, as seen in Example 8.18.

Example 8.18.

```
# Load package
library(tidylo)

# Calculate log odds ratio
masc_metrics_thr <-
  masc_metrics_thr |>
  bind_log_odds(
    set = modality,
```

```

        feature = type,
        n = n,
        unweighted = TRUE
    )

# Preview (ordered by log_odds)
# Spoken
masc_metrics_thr |>
    slice_max(n = 10, order_by = log_odds)

> # A tibble: 10 x 7
>   type              n      rf      dp modality log_odds log_odds_weighted
>   <chr>         <dbl>  <dbl>  <dbl> <chr>      <dbl>      <dbl>
> 1 understanding_NN 45 0.000843 0.649 Spoken  0.955  6.41
> 2 meeting_NNS     42 0.000786 0.702 Spoken  0.955  6.19
> 3 testimony_NN    36 0.000674 0.785 Spoken  0.955  5.73
> 4 administration_NN 34 0.000637 0.650 Spoken  0.955  5.57
> 5 trial_NN        33 0.000618 0.769 Spoken  0.955  5.49
> 6 governor_NN     28 0.000524 0.597 Spoken  0.955  5.05
> 7 intelligent_JJ   28 0.000524 0.777 Spoken  0.955  5.05
> 8 faith_NN         27 0.000506 0.524 Spoken  0.955  4.96
> 9 walk_VBZ         27 0.000506 0.761 Spoken  0.955  4.96
> 10 gun_NNS        26 0.000487 0.577 Spoken  0.955  4.87

# Written
masc_metrics_thr |>
    slice_min(n = 10, order_by = log_odds)

> # A tibble: 10 x 7
>   type              n      rf      dp modality log_odds log_odds_weighted
>   <chr>         <dbl>  <dbl>  <dbl> <chr>      <dbl>      <dbl>
> 1 correct_JJ       13 0.0000680 0.900 Written -0.461  -3.75
> 2 mean_VBP        38 0.000199 0.776 Written -0.411  -5.00
> 3 president_NN    37 0.000194 0.840 Written -0.406  -4.81
> 4 board_NN         27 0.000141 0.830 Written -0.405  -4.10
> 5 argument_NN     24 0.000126 0.798 Written -0.356  -3.07
> 6 meeting_NN      44 0.000230 0.852 Written -0.343  -3.90
> 7 question_NN     80 0.000418 0.630 Written -0.330  -4.94
> 8 say_VBN         20 0.000105 0.767 Written -0.325  -2.42
> 9 read_VBN        10 0.0000523 0.895 Written -0.325  -1.71

```

```
> 10 cut_NN          9 0.0000471 0.879 Written      -0.325          -1.62
```

The distinctive terms in each modality from Example 8.18 may not jibe with your intuition, and that's understandable. This is likely because we are comparing sub-corpora of different sizes and with different document lengths. The log odds ratio is a measure that is sensitive to these differences.

The second measure produced by `bind_log_odds()` function, is the weighted log odds ratio. This measure provides a more robust and interpretable measure for comparing term frequencies across corpora, especially when term frequencies are low or when corpora are of different sizes. The weighting (or standardization) also makes it easier to identify terms that are particularly distinctive or characteristic of one corpus over another. Note that the weighted measure's interpretation is slightly different than the log odds's. The larger positive values in each corpus indicate that the type is more indicative of that (sub-)corpus, and the larger negative values indicate that the type is less indicative.

Let's imagine we would like to extract the most indicative verbs for each modality using the weighted log odds as our measure. We can do this with a little regular expression magic. Let's use the `str_subset()` function to filter for lemmas that start with 'V' and then use `slice_max()` to extract the top 10 most indicative verb lemmas, as seen in Example 8.19.

Example 8.19.

```
# Preview (ordered by log_odds_weighted)
# Spoken and written
masc_metrics_thr |>
  group_by(modality) |>
  filter(str_detect(type, "_V")) |>
  slice_max(n = 10, order_by = log_odds_weighted) |>
  select(-n, -log_odds) |>
  ungroup()

> # A tibble: 20 x 5
>   type          rf      dp modality log_odds_weighted
>   <chr>      <dbl>  <dbl> <chr>            <dbl>
> 1 be_VBZ      0.0489  0.0842 Spoken          14.0
> 2 do_VBP      0.0136  0.180  Spoken          10.3
> 3 be_VBP      0.0240  0.111  Spoken          8.66
> 4 think_VBP   0.00655 0.259  Spoken          8.32
> 5 have_VBP    0.0143  0.152  Spoken          8.00
> 6 know_VBP    0.00528 0.260  Spoken          7.03
> 7 go_VBG      0.00534 0.207  Spoken          6.47
```

| | | | | |
|--------------------|----------|-------|---------|------|
| > 8 do_VBD | 0.00603 | 0.321 | Spoken | 5.97 |
| > 9 mean_VBP | 0.00247 | 0.543 | Spoken | 5.94 |
| > 10 do_VB | 0.00455 | 0.207 | Spoken | 5.61 |
| > 11 don_VB | 0.000361 | 0.839 | Written | 4.04 |
| > 12 doe_VBZ | 0.000350 | 0.871 | Written | 3.98 |
| > 13 walk_VBD | 0.000319 | 0.790 | Written | 3.80 |
| > 14 associate_VBN | 0.000303 | 0.777 | Written | 3.70 |
| > 15 reply_VBD | 0.000293 | 0.838 | Written | 3.64 |
| > 16 develop_VBG | 0.000288 | 0.812 | Written | 3.60 |
| > 17 require_VBN | 0.000272 | 0.793 | Written | 3.50 |
| > 18 fall_VBD | 0.000267 | 0.757 | Written | 3.47 |
| > 19 meet_VB | 0.000241 | 0.729 | Written | 3.30 |
| > 20 regard_VBG | 0.000225 | 0.823 | Written | 3.19 |

Note that the log odds are larger for the spoken modality than the written modality. This indicates that these types are more strongly indicative of the spoken modality than the types in the written modality are indicative of the written modality. This is not surprising, as the written modality is typically more diverse in terms of lexical usage than the spoken modality, where the terms tend to be repeated more often, including verbs.

Co-occurrence analysis

Moving forward on our task, we have a good idea of the general vocabulary that we want to include in our ELL materials and can identify lemma types that are particularly indicative of each modality. Another useful approach to complement our analysis is to identify words that co-occur with our target lemmas –in particular verbs. In English it is common for verbs to appear with a preposition or adverb, such as ‘give up’, ‘look after’. These ‘phrasal verbs’ form a semantic unit that is distinct from the verb alone.

In cases such as this, we are aiming to do a co-occurrence analysis. Co-occurrence analysis is a set of methods that are used to identify words that appear in close proximity to a target type.

An exploratory, primarily qualitative, approach is to display the co-occurrence of words in a Keyword in Context (KWIC). This is a table that displays the target word in the center of the table and the words that appear before and after the target word. This is a useful approach for spot identifying collocations of a target word or phrase.

The `quanteda` package includes a function `kwic()` that can be used to create a KWIC table. It does require some transformation to the data, however. We need to collapse the `lemma` column into a single string for each document from the original transformed dataset, `masc_tbl`. Then we can apply the `corpus()` and then `tokens` function to create a `quanteda` `tokens` object. Then we can apply the `kwic()` function to create a KWIC table.

Example 8.20.

```
# Collapse lemma, pos into a single string
masc_text_tbl <-
  masc_tbl |>
  mutate(lemma_pos = str_c(lemma, pos, sep = "_")) |>
  group_by(doc_id, modality, genre) |>
  summarize(text = str_c(lemma_pos, collapse = " ")) |>
  ungroup()

# Load package
library(quanteda)

masc_corpus <-
  masc_text_tbl |>
  corpus(
    text_field = "text",
    docid_field = "doc_id"
  )

masc_corpus |>
  tokens() |>
  kwic(
    pattern = phrase("*_V* *_IN*"),
    window = 3
  ) |>
  as_tibble() |>
  select(docname, pre, keyword, post) |>
  slice_sample(n = 10)

> # A tibble: 10 x 4
>   docname pre                         keyword      post
>   <chr>   <chr>                      <chr>        <chr>
> 1 231    the_DT bio-sphere_JJ there_RB be_VBZ in_IN   this_DT whole_J~
> 2 169    ataturk_NNP rise_VBD to_T0  power_VB on_IN  a_DT wave_NN of~
> 3 219    1980s_NN warren_NNP this_DT result_VBD in_IN  persecution_NN ~
> 4 183    well_RB since_IN it_PRP   look_VBZ like_IN i_PRP be_VBP al~
> 5 332    . _NN when_WRB            register_VBG by_IN phone_NN please~
> 6 150    allow_VBP doctor_NNS to_T0 decide_VB for_IN themselves_PRP ~
> 7 5     ' s_POS counselor_NNS   feel_VBD that_IN his_PRP $ perso~
> 8 266    transfer_NN agent_NNS to_T0 stay_VB out_IN of_IN it_PRP th~
> 9 158    censorship_NN process_NN that_WDT go_VBZ into_IN put_VBG book_NN~
```

```
> 10 181      and_CC be_VBZ clearly_RB      recline_VBG in_IN  a_DT hospital_N~
```

A straightforward quantitative way to explore co-occurrence is to set the unit of observation to an n -gram of terms. An n -gram is a sequence of n words. For example, a 2-gram is a sequence of two words, a 3-gram is a sequence of three words, and so on. Then, the frequency and dispersion metrics can be calculated for each n -gram.

In general, deriving n -grams from a corpus is a straightforward process. The `tidytext` package includes a function `unnest_tokens()` that can be used to create n -grams from a corpus. The function can take a single column of untokenized text or a tokenized column in combination with a variable to use as the grouping variable. In the `masc_tbl` dataset, we have tokenized text in the `lemma` column and a grouping variable in the `doc_id` column. We can use the `unnest_tokens()` function to create a new data frame with a row for each n -gram in each document, as seen in Example 8.21.

Example 8.21.

```
# Load package
library(tidytext)

# Create bigrams
masc_tbl |>
  unnest_tokens(
    output = bigrams,
    input = lemma,
    token = "ngrams",
    n = 2,
    collapse = "doc_id"
  ) |>
  slice_head(n = 10)
```

```
> # A tibble: 10 x 2
>   doc_id bigrams
>   <chr>  <chr>
> 1 1      december your
> 2 1      your contribution
> 3 1      contribution to
> 4 1      to goodwill
> 5 1      goodwill will
> 6 1      will mean
> 7 1      mean more
> 8 1      more than
```

```
> 9 1      than you
> 10 1     you may
```

The result of this operation can be joined with the original dataset using the `doc_id` as the key variable. Then we can calculate the frequency and dispersion metrics for each n -gram in each modality. However, this approach is not ideal for our task. The reason is that we are interested in identifying n -grams that include verbs. The `unnest_tokens()` function does not allow us to filter the n -grams by part-of-speech.

Another, more informative approach is to create a new variable that combines the lemma and part-of-speech for each observation before using `unnest_tokens()` to generate the bigrams. We can use the `str_c()` function from the `stringr` package to join the `lemma` and `pos` columns into a single string, so that we have a variable `lemma_pos` with the lemma and part-of-speech joined by an underscore.

One consideration that we need to take for our goal to identify verb particle constructions, is how we ultimately want to group our `lemma_pos` values. This is particularly important given the fact that our `pos` tags for verbs include information about the verb's tense and person. This means that a verb in a verb particle bigram, such as 'look after', will be represented by multiple `lemma_pos` values, such as 'look_VB', 'look_VBP', 'look_VBD', and 'look_VBG'. If we want this level of detail, we just proceed as described above. However, if we want to group the verb particle bigrams by a single verb value, we need to recode the `pos` values for verbs. We can do this with the `case_match()` function from the `dplyr` package.

In Example 8.22, I recode the `pos` values for verbs to 'V' and then join the `lemma` and `pos` columns into a single string.

Example 8.22.

```
# Collapse lemma into a single string
masc_lemma_pos_tbl <-
  masc_tbl |>
  mutate(pos = case_when(
    str_detect(pos, "^V") ~ "V",
    TRUE ~ pos
  )) |>
  group_by(doc_id) |>
  mutate(lemma_pos = str_c(lemma, pos, sep = "_")) |>
  ungroup()

# Preview
masc_lemma_pos_tbl |>
  slice_head(n = 10)
```

```

> # A tibble: 10 x 8
>   doc_id modality genre  term_num term      lemma      pos lemma_pos
>   <chr>  <chr>    <chr>     <dbl> <chr>     <chr>      <chr> <chr>
> 1 1     Written  Letters      0 December  december  NNP  december_NNP
> 2 1     Written  Letters      2 Your      your      PRP$  your_PRP$
> 3 1     Written  Letters      3 contribution contribution NN  contributio~
> 4 1     Written  Letters      4 to       to       T0  to_TO
> 5 1     Written  Letters      5 Goodwill  goodwill  NNP  goodwill_NNP
> 6 1     Written  Letters      6 will     will     MD  will_MD
> 7 1     Written  Letters      7 mean     mean     V  mean_V
> 8 1     Written  Letters      8 more     more     JJR  more_JJR
> 9 1     Written  Letters      9 than     than     IN  than_IN
> 10 1    Written  Letters     10 you     you     PRP  you_PRP

```

```

masc_lemma_pos_tbl |>
  unnest_tokens(
    output = bigrams,
    input = lemma_pos,
    token = "ngrams",
    n = 2,
    to_lower = FALSE,
    collapse = "doc_id"
  ) |>
  filter(str_detect(bigrams, "_V.*_IN")) |>
  calc_type_metrics(
    type = bigrams,
    documents = doc_id,
    frequency = "rf",
    dispersion = "dp"
  ) |>
  arrange(desc(rf))

```

```

> # A tibble: 4,512 x 4
>   type          n      rf      dp
>   <chr>      <dbl>  <dbl>  <dbl>
> 1 be_V in_IN      359  0.0259  0.366
> 2 be_V that_IN     210  0.0151  0.504
> 3 look_V at_IN     157  0.0113  0.528
> 4 say_V that_IN     134  0.00966 0.580
> 5 be_V on_IN       120  0.00865 0.523
> 6 talk_V about_IN    114  0.00821 0.622

```

```

> 7 be_V of_IN      110 0.00793 0.503
> 8 know_V that_IN 98 0.00706 0.605
> 9 think_V that_IN 90 0.00649 0.643
> 10 be_V about_IN 76 0.00548 0.608
> # i 4,502 more rows

```

We have identified and derived frequency and dispersion metrics for n – *grams* that include verb particle construction candidates. Yet, there is a problem with this approach. The problem is that the n – *grams* are not necessarily verb particle constructions in the sense that they form a semantic unit. Second, frequency and dispersion metrics are not necessarily the best measures for identifying the co-occurrence relationship between the verb and the particle. In other words, just because a two-word sequence is frequent and well-dispersed does not mean that the two words form a semantic unit.

To address these issues, we can use a statistical measures to estimate collocational strength between two words. A **collocation** is a sequence of words that co-occur more often than would be expected by chance. The most common measure of collocation is the **pointwise mutual information** (PMI) measure. The PMI measure is calculated as the log ratio of the observed frequency of two words co-occurring to the expected frequency of the two words co-occurring. The expected frequency is calculated as the product of the frequency of each word. The PMI measure is a log ratio, so the values range from negative to positive infinity, with negative values indicating that the two words co-occur less often than would be expected by chance and positive values indicating that the two words co-occur more often than would be expected by chance. The magnitude of the value indicates the strength of the association.

Let's calculate the PMI for all the bigrams in the MASC dataset. We can use the `calc_assoc_metrics()` function from `qtalrkit`. We need to specify the `association` argument to `pmi` and the `type` argument to `bigrams`, as seen in Example 8.23.

Example 8.23.

```

masc_lemma_pos_assoc <-
  masc_lemma_pos_tbl |>
  calc_assoc_metrics(
    doc_index = doc_id,
    token_index = term_num,
    type = lemma_pos,
    association = "pmi"
  )

# Preview
masc_lemma_pos_assoc |>

```

```

arrange(desc(pmi)) |>
slice_head(n = 10)

> # A tibble: 10 x 4
>   x                 y                 n     pmi
>   <chr>              <chr>            <dbl>  <dbl>
> 1 #Christian_NN    bigot_NN          1  12.4
> 2 #FAIL_NN          phenomenally_RB  1  12.4
> 3 #NASCAR_NN        #indycar_NN      1  12.4
> 4 #PALM_NN          merchan_NN       1  12.4
> 5 #Twitter_NN       #growth_NN        1  12.4
> 6 #college_NN       #jobs_NN          1  12.4
> 7 #education_NN     #teaching_NN      1  12.4
> 8 #faculty_NN       #cites_NN         1  12.4
> 9 #fb_NN             siebel_NNP       1  12.4
> 10 #glitchmyass_NN  reps_NNP         1  12.4

```

One caveat to using the PMI measure is that it is sensitive to the frequency of the words. If the words in a bigram pair are infrequent, and especially if they only occur once, then the PMI measure will be inflated. To mitigate this issue, we can apply a frequency threshold to the bigrams before calculating the PMI measure. Let's filter out bigrams that occur less than 10 times, as seen in Example 8.24.

Example 8.24.

```

# Filter for bigrams that occur >= 10 times
masc_lemma_pos_assoc_thr <-
  masc_lemma_pos_assoc |>
  filter(n >= 10) |>
  arrange(desc(pmi))

# Preview
masc_lemma_pos_assoc_thr |>
  slice_head(n = 10)

> # A tibble: 10 x 4
>   x                 y                 n     pmi
>   <chr>              <chr>            <dbl>  <dbl>
> 1 pianista_NN      irlandesa_NN    10  10.0
> 2 costa_NNP         rica_NNP        10  9.95

```

```

> 3 nanowrimo_NNP novel_NNP      12  9.87
> 4 bin_NN      laden_NNP      11  9.79
> 5 osama_NNP   bin_NN       11  9.79
> 6 bin_NNP     ladin_NNP      11  9.70
> 7 los_NNP     angeles_NNP    11  9.64
> 8 chilean_JJ   seabass_NNS   13  9.64
> 9 novel_NNP   ch_NNP       12  9.58
> 10 st_NNP     zip_NNP      10  9.52

```

Now we are in a position to identify verb particle constructions. We can filter for bigrams that include a verb and a particle and that have a PMI measure greater than 0, as seen in Example 8.25.

Example 8.25.

```

# Filter for bigrams that include a verb and a particle
# and that have a PMI measure greater than 0
masc_verb_part_assoc <-
  masc_lemma_pos_assoc_thr |>
  filter(str_detect(x, "_V")) |>
  filter(str_detect(y, "_IN")) |>
  filter(pmi > 0) |>
  arrange(x)

# Preview
masc_verb_part_assoc |>
  slice_head(n = 10)

```

```

> # A tibble: 10 x 4
>       x             y       n     pmi
>   <chr>        <chr> <dbl> <dbl>
> 1 account_V     for_IN    17  3.74
> 2 acknowledge_V that_IN   13  3.62
> 3 act_V         as_IN    14  2.96
> 4 agree_V       with_IN   45  3.21
> 5 agree_V       that_IN   14  1.94
> 6 appear_V      on_IN    12  1.98
> 7 appear_V      in_IN    24  1.76
> 8 argue_V       that_IN   20  3.26
> 9 arrive_V      at_IN    18  3.39
> 10 arrive_V     in_IN    10  1.48

```

We can clean up the results a bit by removing the part-of-speech tags from the `x` and `y` variables, up our minimum PMI value, and create a network plot to visualize the results, as seen in Figure 8.6.

```
# Clean up results
masc_verb_part_assoc_plot <-
  masc_verb_part_assoc |>
  filter(pmi >= 2) |>
  mutate(
    x = str_remove(x, "_V.*"),
    y = str_remove(y, "_IN")
  )

# Create an association network plot
# `x` and `y` are the nodes
# `pmi` is the edge weight

library(igraph)
library(ggraph)

masc_verb_part_assoc_plot |>
  graph_from_data_frame() |>
  ggraph(layout = "nicely") +
  geom_edge_link(aes(color = pmi),
    alpha = 0.8,
    edge_width = 0.8,
    arrow = grid::arrow()
  ) +
  geom_node_point(color = "black") +
  geom_node_text(aes(label = name), repel = TRUE) +
  scale_edge_color_gradient(low = "grey90", high = "grey20") +
  theme_void()
```

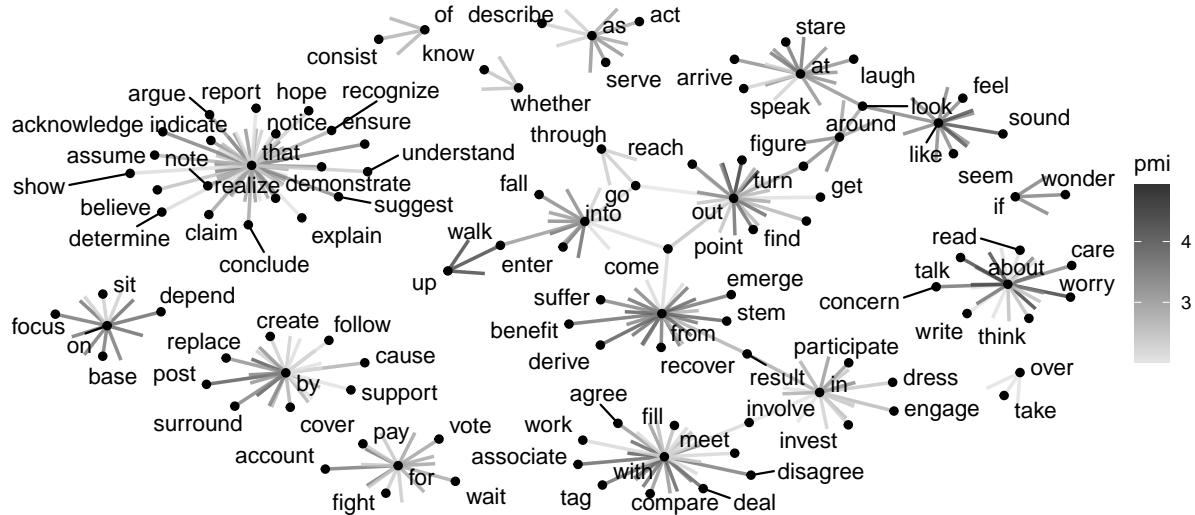


Figure 8.6: Network plot of verb particle constructions in the MASC dataset

From this plot, and from the underlying data, we can explore verb particle constructions. We could go further and apply our co-occurrence methods to each modality separately, if we wanted to identify verb particle constructions that are distinctive to each modality. We could also apply our co-occurrence methods to other parts-of-speech, such as adjectives and nouns, to identify collocations of these parts-of-speech. There is much more to explore with co-occurrence analysis, but this should give you a good idea of the types of questions that can be addressed with co-occurrence analysis.

8.2.2 Unsupervised learning

Aligned in purpose with descriptive approaches, unsupervised learning approaches to exploratory data analysis are used to identify patterns in the data from an algorithmic perspective. Common methods in text analysis include principle component analysis, clustering, and vector space modeling.

We will continue to use the MASC dataset as we develop materials for our ELL textbook to illustrate unsupervised learning methods. In the process, we will explore the following questions:

- Can we identify and group documents based on linguistic features or co-occurrence patterns of the data itself?
 - Do the groups of documents map to the labels in the dataset?
 - Can we estimate the semantics of words based on their co-occurrence patterns?

Through these questions we will build on our knowledge of frequency, dispersion, and co-occurrence analysis and introduce concepts and methods associated with machine learning.

Clustering

Clustering is a unsupervised learning technique that can be used to group similar items in the text data, helping to organize the data into distinct categories and discover relationships between different elements in the text. The main steps in the procedure includes identifying the relevant linguistic features to use for clustering, representing the features in a way that can be used for clustering, and applying a clustering algorithm to the data. However, it is important to consider the strengths and weaknesses of the clustering algorithm for a particular task and how the results will be evaluated.

In our ELL textbook task, we may very well want to explore the similarities and/ or differences between the documents based on the distribution of linguistic features. This provides us a view to evaluate to what extent the labels in the dataset (modality and genre) map to the distribution of linguistic features. Based on this evaluation, we may want to consider re-labeling the documents, collapsing labels, or even adding new labels.

Enter clustering. Instead of relying entirely on the labels in the MASC dataset, we can let the data itself say something about how related the documents are. Yet, a pivotal question is what features should we use, otherwise known as **feature selection**. We could use terms or lemmas, but we may want to consider other features, such as parts-of-speech or some co-occurrence patterns. We are not locked into using one criterion, and we can perform clustering multiple times with different features, but we should consider the implications of our feature selection for our interpretation of the results.

Another key question is what clustering algorithm to use. Again, we are not married to one algorithm, and we can perform clustering multiple times with different algorithms, but not all algorithms are created equal. Some algorithms are better suited for certain types of data and certain types of tasks. For example, **Hierarchical clustering** is a good choice when we are not sure how many clusters we want to identify, as it does not require us to specify the number of clusters from the outset. However, it is not a good choice when we have a large dataset, as it can be computationally expensive compared to some other algorithms. **K-means clustering**, on the other hand, is a good choice when we want to identify a pre-defined number of clusters, and the aim is to gauge how well the data fit the clusters. These two clustering techniques, therefore complement each other with Hierarchical clustering being a good choice for initial exploration and K-means clustering being a good choice for targeted evaluation.

With these considerations in mind, let's start by identifying the linguistic features that we want to use for clustering. Imagine that among the various features that we are interested in associating documents, we consider lemma use and part-of-speech use. However, we need to operationalize what we mean by 'use'. In machine learning, this process is known as **feature engineering**. Since we aim to compare documents it is logical for us to use the document-normalized features. So in both lemma and part-of-speech tags, we will use the relative frequency. An additional operation that we can apply to the lemma feature is to weight the relative frequency by the dispersion of the lemma. This will give us a measure of the

distinctiveness of the lemma in the document. A common implementation of this approach is to use the $tf - idf$ measure, which is the product of the relative frequency and the inverse document frequency.

Each of these engineered feature sets represents a different aspect of the lexical nature of the documents. The relative frequency of lemmas represents the lexical diversity of the documents, the dispersion-weighted $tf - idf$ of lemmas represents the distinctiveness of the lemmas in the documents, and the relative frequency of part-of-speech tags represents the grammatical diversity of the documents (Petrenz and Webber 2011). //FIXME CITATIONS

The next question to address in any analysis is how to represent the features. In our case, we want to represent the features in each document. In machine learning, the most common way to represent features is in a matrix. In our case, we want to create a matrix with the documents in the rows and the features in the columns. The values in the matrix will be the operationalization of lexical use in each document for each of our three candidate measures. This configuration is known as a **document-term matrix** (DTM).

To recast a data frame into a DTM, we can use the `cast_dtm()` function from the `tidytext` package. This function takes a data frame with a document identifier, a feature identifier, and a value for each observation and casts it into a matrix. Operations such as normalization are easily and efficiently performed in R on matrices, so initially we can cast a frequency table of lemmas and part-of-speech tags into a matrix and then normalize the matrix by documents. For the $tf - idf$ measure we use the `bind_tf_idf()` function from the `tidytext` package. This function takes a DTM and calculates the $tf - idf$ measure for each feature in each document. This is a normalized measure, so we do not need to normalize the matrix by documents. Let's see how this works with the MASC dataset in Example 8.26.

Example 8.26.

```
# Load packages
library(tidytext)

# Create a document-term matrix of lemmas
masc_lemma_dtm <-
  masc_tbl |>
  count(doc_id, lemma) |>
  cast_dtm(doc_id, lemma, n) |>
  as.matrix()

# Create a document-term matrix of part-of-speech tags
masc_pos_dtm <-
  masc_tbl |>
  count(doc_id, pos) |>
```

```

cast_dtm(doc_id, pos, n) |>
  as.matrix()

# Create a document-term matrix of tf-idf weighted lemmas
masc_lemma_tfidf_dtm <-
  masc_tbl |>
  count(doc_id, lemma) |>
  bind_tf_idf(doc_id, lemma, n) |>
  cast_dtm(doc_id, lemma, tf_idf) |>
  as.matrix()

```

Note preview the a subset of the contents of a matrix, such as in Example 8.26, we use bracket syntax [] instead of the `head()` function. Let's take a look at the first 5 rows and 5 columns of the matrices, as seen in Example 8.27.

Example 8.27.

```

# Preview
masc_lemma_dtm[1:5, 1:5]

```

```

>      Terms
> Docs  'd 's M. a account
> 1    1 1 1 15      1
> 10   0 0 0 7      0
> 100  0 0 0 0      0
> 101  0 0 0 2      0
> 102  0 0 0 1      0

```

```

masc_pos_dtm[1:5, 1:5]

```

```

>      Terms
> Docs  CC DT EX IN JJ
> 1    14 35 1 44 27
> 10   11 38 0 39 18
> 100  0 2 0 2 3
> 101  3 16 0 23 7
> 102  20 29 0 34 20

```

```
masc_lemma_tfidf_dtm[1:5, 1:5]
```

```
>      Terms
> Docs      'd      's      M.      a account
> 1  0.0547 0.00341 0.241 0.006898 0.0391
> 10 0.0000 0.00000 0.000 0.003467 0.0000
> 100 0.0000 0.00000 0.000 0.000000 0.0000
> 101 0.0000 0.00000 0.000 0.001050 0.0000
> 102 0.0000 0.00000 0.000 0.000479 0.0000
```

Now we can normalize the lemma and pos matrices by documents. We can do this by dividing each feature count by the total count in each document. This is a row-wise transformation, so we can use the `rowSums()` function from base R to calculate the total count in each document. Then each count divided by its row's total count, as seen in Example 8.28.

Example 8.28.

```
# Normalize lemma and pos matrices by documents
masc_lemma_dtm <-
  mascot_lemma_dtm / rowSums(masc_lemma_dtm)

masc_pos_dtm <-
  mascot_pos_dtm / rowSums(masc_pos_dtm)
```

There are two concerns to address before we can proceed with clustering. First, clustering algorithm performance tends to degrade with the number of features. If we consider either the relative frequency of lemmas or the dispersion-weighted $tf - idf$ of lemmas, we are looking at over 25k features! Second, clustering algorithms perform better with more informative features. That is to say, features that are more distinct across the documents provide better information for deriving useful clusters.

We can address both of these concerns by reducing the number of features and increasing the informativeness of the features. To accomplish this is to use **dimensionality reduction**. Dimensionality reduction is a set of methods that are used to reduce the number of features in a dataset while retaining as much information as possible. The most common method for dimensionality reduction is **principle component analysis** (PCA). PCA is a method that transforms a set of correlated variables into a set of uncorrelated variables, known as principle components. The principle components are ordered by the amount of variance that they explain in the data. The first principle component explains the most variance, the second principle component explains the second most variance, and so on.

We can apply PCA to each of these features and assess how well the features account for the variation in the data. We can then use the features that account for the most variation in the data for clustering. The `prcomp()` function from base R can be used to perform PCA. Let's apply PCA to each of our candidate feature matrices, as seen in Example 8.29.

Example 8.29.

```
# Set seed for reproducibility
set.seed(123)

# Apply PCA to each feature matrix
masc_lemma_pca <-
  masc_lemma_dtm |>
  prcomp()

masc_pos_pca <-
  masc_pos_dtm |>
  prcomp()

masc_lemma_tfidf_pca <-
  masc_lemma_tfidf_dtm |>
  prcomp()
```

We can visualize the amount of variance explained by each principle component with a scree plot. The `fviz_eig()` function from the `factoextra` package can be used to create a scree plot. The `fviz_eig()` function takes the output of the `prcomp()` function as its argument and an argument `ncp = 10` to specify the number of principle components to include in the plot. Let's create a scree plot for each of our candidate feature matrices, as seen in Figure 8.7.

```
# Load package
library(factoextra)

# Scree plot: lemma relative frequency
fviz_eig(masc_lemma_pca, ncp = 10)

# Scree plot: lemma dispersion-weighted tf-idf
fviz_eig(masc_lemma_tfidf_pca, ncp = 10)

# Scree plot: part-of-speech relative frequency
fviz_eig(masc_pos_pca, ncp = 10)
```

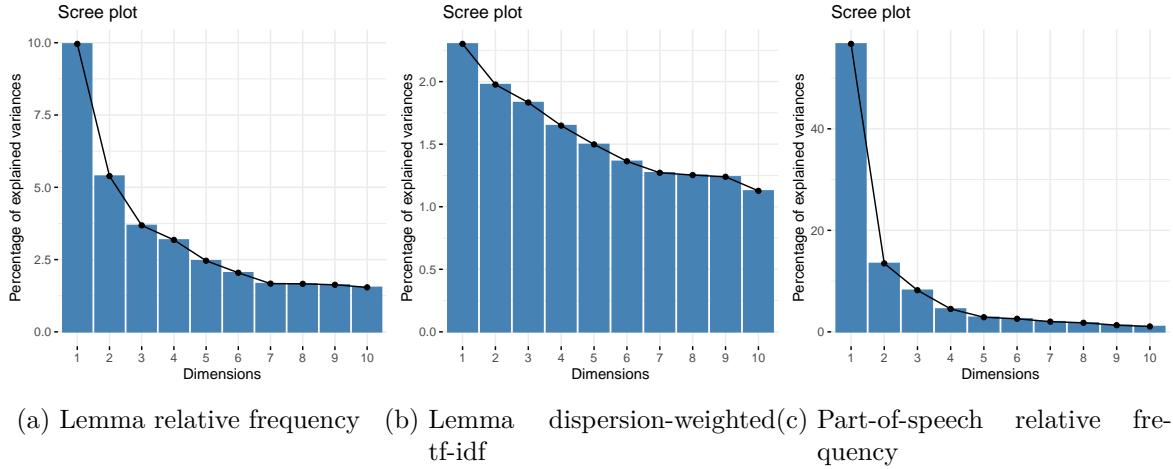


Figure 8.7: Scree plot of the principle components of the MASC dataset

From Figure 8.7, we can see the plots are different. All trend toward less variance explained as the number of dimensions increase. But for our purposes, we are interested in the largest variance explained with the fewest dimensions. To that end, the Figure 8.7c plot is the most reduced and most informative. The amount of variance explained is over 30% for the first dimension alone, however, the variance explained decreases between 4 and 5 dimensions. This is a good indication that we should use 4 dimensions for our clustering algorithm.

To calculate the amount of variance explained by each principle component we can square the standard deviations of the principle components and divide by the sum of the squared standard deviations. Let's calculate the amount of variance explained by each principle component for each of our candidate feature matrices, as seen in Example 8.30.

Example 8.30.

```
# Calculate variance explained for first 3 principle components
# lemma
masc_lemma_pca_var <-
  masc_lemma_pca$sdev^2 / sum(masc_lemma_pca$sdev^2) * 100

sum(masc_lemma_pca_var[1:4])
```

```
> [1] 22.2
```

```
# pos
masc_pos_pca_var <-
```

```
masc_pos_pca$sdev^2 / sum(masc_pos_pca$sdev^2) * 100  
sum(masc_pos_pca_var[1:4])
```

```
> [1] 82.9
```

```
# lemma tf-idf  
masc_lemma_tfidf_pca_var <-  
  masc_lemma_tfidf_pca$sdev^2 / sum(masc_lemma_tfidf_pca$sdev^2) * 100  
sum(masc_lemma_tfidf_pca_var[1:4])
```

```
> [1] 7.76
```

Combining the findings in the Scree plots and the variance explained calculations, we can see that the first four principle components of the part-of-speech features account for a good proportion of the variance. Therefore, all else being equal, we should use the part-of-speech features. As with all things exploratory, however, it is important to consider the implications of our feature selection for our interpretation of the results. In this case, the part-of-speech features approximate grammatical diversity of the documents, more so than lexical diversity. This means that the clusters that we identify will be based on a particular measure of grammatical diversity of the documents. If, for example, we want to identify clusters based on the lexical diversity of the documents, we may opt to use the lemma features, or some other operationalized measure of lexical diversity.

Before we leave PCA, let's also take a look at the principle components themselves. The `get_pca_var()` function from the `factoextra` package can be used to extract the principle components from the output of the `prcomp()` function. The `get_pca_var()` function takes the output of the `prcomp()` function as its argument and an argument `ncp = 10` to specify the number of principle components to include in the plot. Let's create a plot of the first five principle components for the part-of-speech data, as seen in Figure 8.8.

```
# Load package  
library(factoextra)  
  
# Plot principle components: pos  
masc_pos_pca |>  
  fviz_contrib(  
    choice = "var",
```

```

    axes = 1:4,
    top = 20
)

```

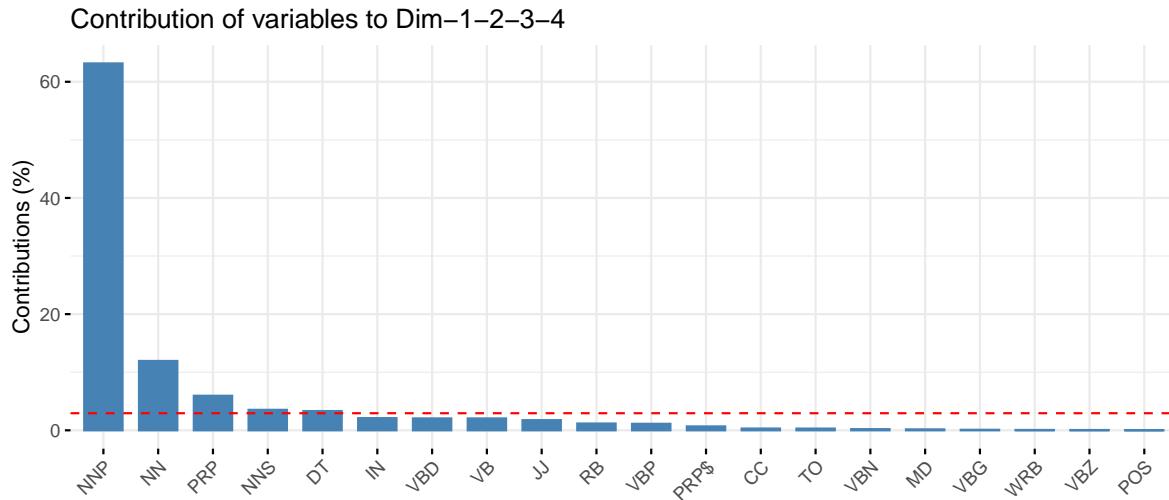


Figure 8.8: Feature contributions to the PCA of the MASC dataset

From Figure 8.8, we can see that the four principle components are dominated by the relative frequency of nouns, personal pronouns, prepositions, and determiners. This information can help us better understand the results of the clustering algorithm.

Now that we have identified the features that we want to use for clustering and we have represented the features in a way that can be used for clustering, we can apply a clustering algorithm to the data. For Hiearchical clustering, we can use the `hclust()` function from base R. The `hclust()` function takes a distance matrix as its argument and an argument `method = "average"` to specify the average linkage method. The average linkage method takes the average of the dissimilarities between all pairs in two clusters. It is less sensitive to outliers compared to other methods. Let's apply the clustering algorithm to the part-of-speech features, as seen in Example 8.31.

Example 8.31.

```

# Extract first 4 principle components
masc_pos_pca_pc <-
  masc_pos_pca$x[, 1:4]

# Create distance matrix

```

```

masc_pos_dist <-
  masc_pos_pca_pc |>
  dist(method = "manhattan")

# Apply the clustering algorithm
masc_pos_hc <-
  masc_pos_dist |>
  hclust(method = "average")

# Visualize
masc_pos_hc |> fviz_dend(show_labels = FALSE)

```

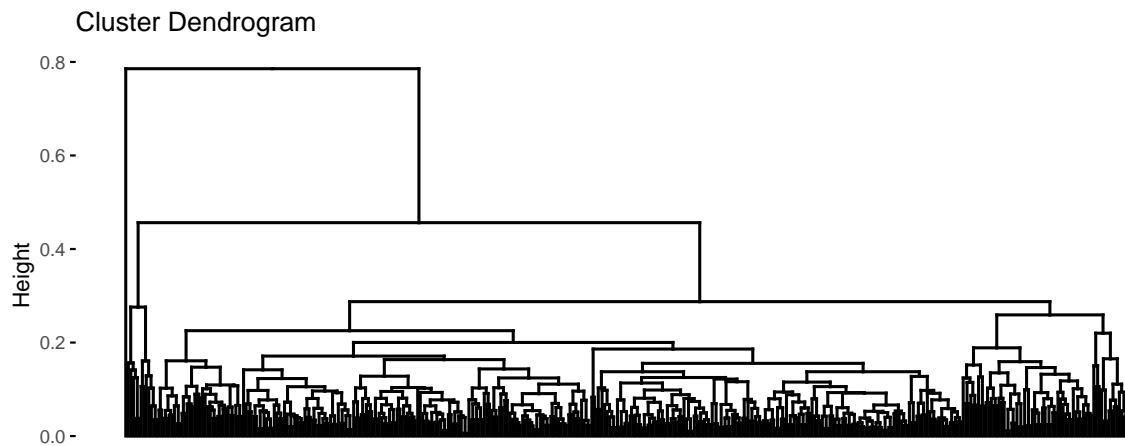


Figure 8.9: Hierarchical clustering of the MASC dataset

Since we are exploring the usefulness of the 18 genre labels used in the MASC dataset we have a good idea of how many clusters we want to start with. This is a good case to employ the K-means clustering algorithm. In K-means clustering, we specify the number of clusters that we want to identify. For each cluster number, a random center is generated. Then each observation is assigned to the cluster with the nearest center. The center of each cluster is then recalculated based on the distribution of the observations in the cluster. This process is iterates either a pre-defined number of times, or until the centers converge (*i.e.* observations stop switching clusters).

We can use the `kmeans()` function from base R to apply the K-means clustering algorithm. The `kmeans()` function takes the matrix of features as its first argument and the number of clusters as its second argument. We can specify the number of clusters with the `centers` argument.

The `kmeans()` function also takes an argument `nstart` to specify the number of random starts. The K-means algorithm is sensitive to the initial starting points, so it is a good idea to run the algorithm multiple times with different starting points. The `nstart` argument specifies the number of random starts. The default value is 1, but we can increase this to 10 or 20 to increase the likelihood of finding a good solution.

Our goal, then, will be to assess how well this number of clusters fits the data. If it does not fit the data well, we can try a different number of clusters. We can then compare the results of the clustering with the genre labels to see how well the clusters map to the labels and make adjustments to the way we group the labels as necessary.

Let's start with 18 clusters, assuming the target of the number of genres in the MASC dataset. We can apply the K-means clustering algorithm to the part-of-speech features, as seen in Example 8.32.

Example 8.32.

```
# Extract first 4 principle components
masc_pos_pca_pc <-
  masc_pos_pca$x[, 1:4]

# K-means clustering
masc_pos_kmeans <-
  masc_pos_pca_pc |>
  kmeans(
    centers = 18,
    nstart = 25,
    iter.max = 20
  )
```

The `factoextra` package provides `fviz_cluster()` function for visualizing the results of clustering algorithms. The `fviz_cluster()` function takes the output of the `kmeans()` function as its first argument and the matrix of features as its second argument. The `fviz_cluster()` function can be used to visualize the clusters in the data, as seen in Figure 8.10.

```
# Visualize
masc_pos_kmeans |> # output of kmeans()
  fviz_cluster(
    data = masc_pos_pca_pc, # matrix of features
    ellipse.type = "norm",
    ellipse.level = 0.95,
    geom = "point",
```

```

  pointsize = 1,
  palette = "grey",
  ggtheme = theme_qtalr()
)

```

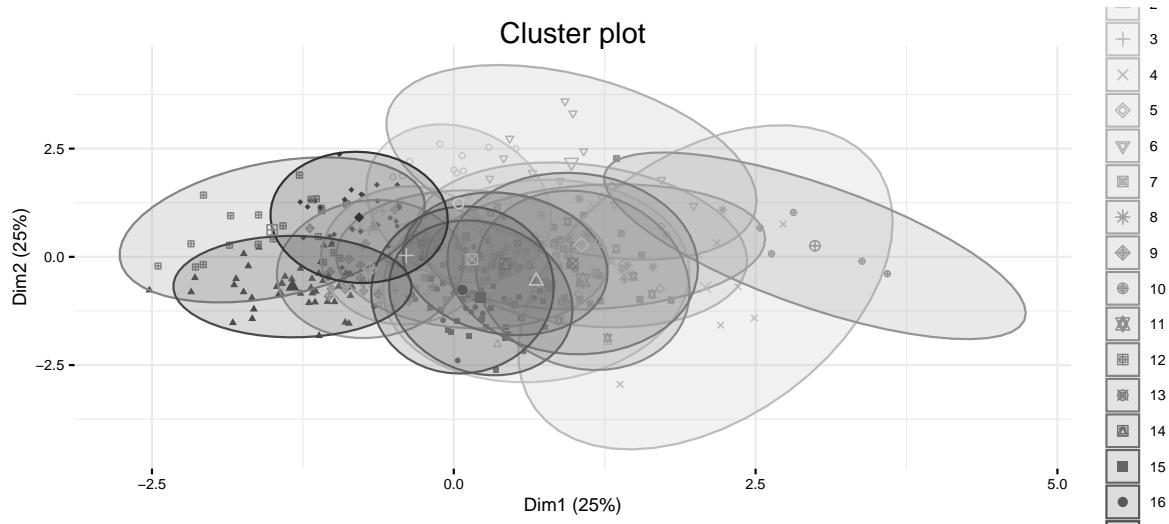


Figure 8.10: K-means clustering of the MASC dataset

The ellipses in a k-means plot represent the 95% confidence interval for each cluster. The ellipses are based on the multivariate normal distribution of the data in each cluster. The size and shape of the ellipses tell us about the variance of the data in each cluster. The larger the ellipse the greater the dispersion of values. A wide ellipse suggests high within cluster variability. The distance between the clusters also tells us about the similarity between the clusters. The closer the clusters are to each other, the more similar they are. The further the clusters are from each other, the more dissimilar they are.

So in Figure 8.10, we see a mix of shapes and sizes. This suggests that some clusters are more homogeneous than others. We also see separation between the large wide clusters towards the top of the plot and the smaller, more circular clusters towards the center. With 18 clusters, we have a lot of clusters, so it is difficult to interpret the results as there is a large amount of overlap. In sum, 18 clusters is likely not an optimal number for this clustering approach.

We could run the code in Example 8.32 for different values for k and plot each in turn. But a more effective way to determine the optimal number of clusters is to plot the within-cluster sum of squares (WSS) for a range of values for k . The WSS is the sum of the squared distance between each observation and its cluster center. With a plot of the WSS for a range of values for k , we can identify the value for k where the WSS begins to level off. This is known as the

elbow method. The elbow method is a heuristic, so it is not always clear where the elbow is. However, it is a good starting point for identifying the optimal number of clusters.

Again, the `factoextra` package has us covered. The `fviz_nbclust()` function can be used to plot the WSS for a range of values for k . The `fviz_nbclust()` function takes the `kmeans()` function as its first argument and the matrix of features as its second argument. The `fviz_nbclust()` function also takes arguments `method = "wss"` to specify the WSS method and `k.max = 20` to specify the maximum number of clusters to plot. Let's plot the WSS for a range of values for k , as seen in Figure 8.11.

```
# Determine the optimal number of clusters
masc_pos_pca_pc |>
  fviz_nbclust(
    FUNcluster = kmeans,
    method = "wss", # method
    k.max = 20,
    nstart = 25,
    iter.max = 20
  )
```

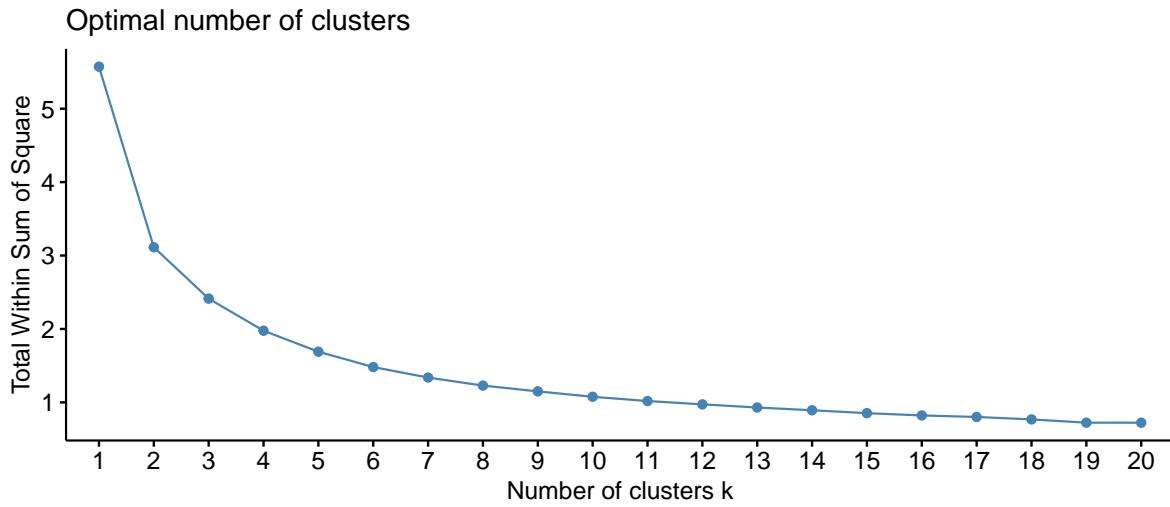


Figure 8.11: Elbow method for k-means clustering of the MASC dataset

It is clear that there is significant gains in cluster fit from 1 to 4 clusters, but the gains begin to level off after 5-7 clusters. Now we can skip ahead and try 4 clusters, as seen in Example 8.33.

Example 8.33.

```

# Set seed for reproducibility
set.seed(123)

# K-means: for 5 clusters
masc_pos_kmeans_fit <-
  masc_pos_pca_pc |>
  kmeans(
    centers = 4,
    nstart = 25,
    iter.max = 20
  )

# Visualize
masc_pos_kmeans_fit |>
  fviz_cluster(
    data = masc_pos_pca_pc,
    ellipse.type = "norm",
    ellipse.level = 0.95,
    geom = "point",
    pointsize = 1,
    palette = "grey",
    ggtheme = theme_qtalr()
  )

```

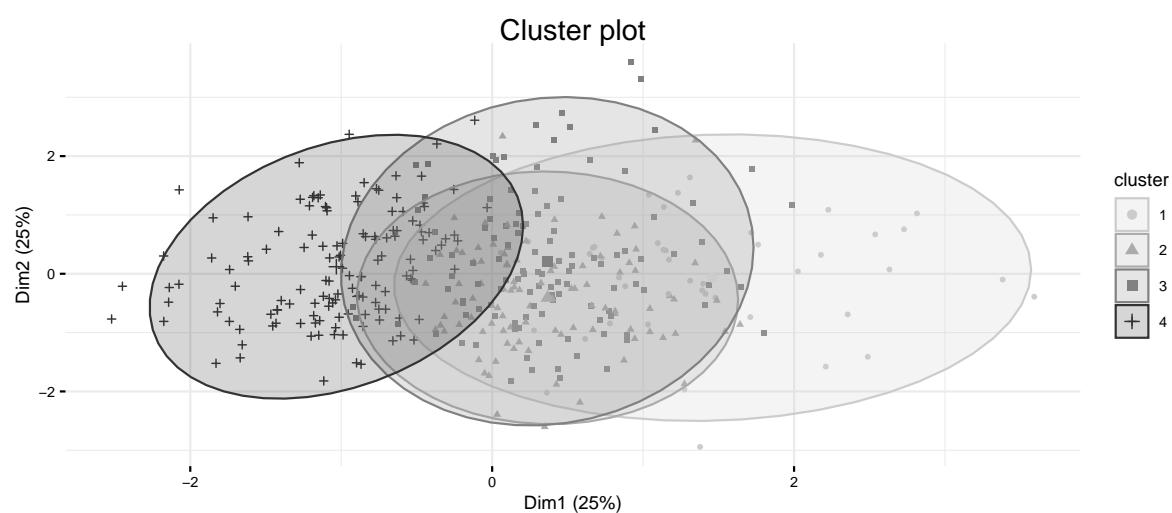


Figure 8.12: K-means clustering of the MASC dataset with 4 clusters

The results are much more interpretable with 4 clusters. We can see that the clusters are more homogeneous and more distinct from each other, in particular clusters 1 and 2, and are generally similar in shape and size. However, there is still some overlap between the clusters, in particular for clusters 3 and 4. We expect there to be noise as we have paired down the number of features from over 25k to 4, so this is a good working solution.

From this point we can join document-cluster pairings produced by the k-means algorithm with the original dataset. We can then explore the clusters in terms of the original features. We can also explore the clusters in terms of the original labels. Let's join the cluster assignments to the original dataset, as seen in Example 8.34.

Example 8.34.

```
# Organize k-means clusters into a tibble
masc_pos_cluster_tbl <-
  tibble(
    doc_id = names(masc_pos_kmeans_fit$cluster),
    cluster = masc_pos_kmeans_fit$cluster
  )

# Join cluster assignments to original dataset
masc_cluster_tbl <-
  masc_tbl |>
  left_join(
    masc_pos_cluster_tbl,
    by = "doc_id"
  )

# Preview
masc_cluster_tbl |>
  slice_head(n = 5)
```

```
> # A tibble: 5 x 8
>   doc_id modality genre  term_num term      lemma      pos   cluster
>   <chr>   <chr>    <chr>     <dbl> <chr>     <chr>     <chr>   <int>
> 1 1       Written  Letters      2 Your      your     PRP$      4
> 2 1       Written  Letters      3 contribution contribution NN      4
> 3 1       Written  Letters      4 to        to       T0       4
> 4 1       Written  Letters      6 will      will     MD       4
> 5 1       Written  Letters      7 mean      mean     VB       4
```

We now see that the cluster assignments from the k-means algorithm have been joined to the original dataset. We can now explore the clusters in terms of the original features. For example,

let's look at the distribution of the clusters across modality first, as seen in Example 8.35. To do this, we first need to reduce our dataset to the distinct combinations of modality, genre, and cluster. Then, we can use the `janitor` package's `tabyl()` function to provide formatted percentages.

Example 8.35.

```
# Load package
library(janitor)

# Reduce to distinct combinations of modality, genre, and cluster
masc_meta_tbl <-
  masc_cluster_tbl |>
  distinct(modality, genre, cluster)

# Tabulate: cluster by modality
masc_meta_tbl |>
  tabyl(cluster, modality) |>
  adorn_percentages("row") |>
  adorn_pct_formatting(digits = 1)

>   cluster Spoken Written
>     1     0.0% 100.0%
>     2     0.0% 100.0%
>     3     9.1%  90.9%
>     4    30.8%  69.2%
```

From Example 8.35, we can see that the clusters are not evenly distributed across the modalities. In particular, cluster 1 is where the great majority of the spoken modality appears. We can also appreciate that there may be some written genres that are more similar to spoken genres than other written genres. This may be something we could like to explore further.

Let's dive into genres and limit our analysis to the written modality, as seen in Example 8.36. To do this, we first need to filter the dataset to the written modality. In addition to the row-wise percentages which capture the proportion of the genre that contributes to each cluster, let's also consider the column-wise percentages to consider how each genre is distributed across the clusters.

Example 8.36.

```
# Tabulate: cluster by genre for written modality
masc_meta_tbl |>
  filter(modality == "Written") |>
  tabyl(cluster, genre) |>
  adorn_percentages("row") |>
  adorn_pct_formatting(digits = 1)
```

```
> cluster  Blog Email Essay Fiction Fictlets Government Jokes Journal Letters
>       1  0.0% 33.3%  0.0%    0.0%    0.0%      0.0% 33.3%  0.0%    0.0%
>       2 10.0% 10.0% 10.0%    0.0%    0.0%      10.0%  0.0% 10.0% 10.0%
>       3 10.0% 10.0% 10.0%    0.0%    0.0%      10.0% 10.0% 10.0% 10.0%
>       4 11.1% 11.1% 11.1% 11.1% 11.1%      0.0% 11.1% 11.1% 11.1%
> Newspaper Non-fiction Technical Travel Guide Twitter
>       33.3%      0.0%      0.0%      0.0%      0.0%
>       10.0%     10.0%     10.0%     10.0%      0.0%
>       10.0%      0.0%      0.0%     10.0%     10.0%
>       11.1%      0.0%      0.0%      0.0%      0.0%
```

```
# Tabulate: cluster by genre for written modality
masc_meta_tbl |>
  filter(modality == "Written") |>
  tabyl(cluster, genre) |>
  adorn_percentages("col") |>
  adorn_pct_formatting(digits = 1)
```

```
> cluster  Blog Email Essay Fiction Fictlets Government Jokes Journal Letters
>       1  0.0% 25.0%  0.0%    0.0%    0.0%      0.0% 33.3%  0.0%    0.0%
>       2 33.3% 25.0% 33.3%    0.0%    0.0%      50.0%  0.0% 33.3% 33.3%
>       3 33.3% 25.0% 33.3%    0.0%    0.0%      50.0% 33.3% 33.3% 33.3%
>       4 33.3% 25.0% 33.3% 100.0% 100.0%      0.0% 33.3% 33.3% 33.3%
> Newspaper Non-fiction Technical Travel Guide Twitter
>       25.0%      0.0%      0.0%      0.0%      0.0%
>       25.0%     100.0%    100.0%     50.0%      0.0%
>       25.0%      0.0%      0.0%     50.0%    100.0%
>       25.0%      0.0%      0.0%      0.0%      0.0%
```

On the other hand, looking at the written genres, we see that there are some genres which are grouped entirely in cluster 1. In other words, these genres, such as ‘Fictlets’ and ‘Jokes’, align with spoken genres in terms of their grammatical diversity. This is an interesting finding

that we may want to explore further. Furthermore, it may be of interest to explore individual documents in genres which have a significant proportion of documents in cluster 1. There are too many possibilities to explore here but this is a good example of how exploratory data analysis can be used to identify new questions and new variables of interest.

💡 Consider this

Given the cluster assignments derived using the distribution of part-of-speech tags, what other relationships between the clusters and the original features could one explore? What are the limitations of this approach? What are the implications of this approach for the interpretation of the results?

Vector space models

In our discussion of clustering, we targeted associations between documents based on the distribution of linguistic features. We now turn to targeting associations between linguistic features based on their distribution across documents. The technique we will introduce is known as **vector space modeling**. Vector space modeling aims to represent linguistic features as numerical vectors which reflect the various linguistic contexts in which the features appear. Together these vectors form a feature-context space in which features with similar contextual distributions are closer together.

An interesting property of vector space models is that are able to capture semantic and/ or syntactic relationships between features based on their distribution. In this way, vector space modeling can be seen as an implementation of the **distributional hypothesis** –that words that appear in similar linguistic contexts tend to have similar meanings (Harris 1954). As Firth (1957) states “you shall know a word by the company it keeps”.

📄 Case study

Garg et al. (2018) quantify and compare gender and ethnic stereotypes over time using word embeddings. The authors explore the temporal dynamics of stereotypes using word embeddings as a quantitative measure of bias. The data used includes word embeddings from the Google News dataset for contemporary analysis, as well as embeddings from the COHA and Google Books datasets for historical analysis. Additional validation is done using embeddings from the New York Times Annotated Corpus. Several word lists representing gender, ethnicity, and neutral words are collated for analysis. The main finding is that language reflects and perpetuates cultural stereotypes, and the analysis shows consistency in the relationships between embedding bias and external metrics across datasets over time. The results also highlight the impact of historical events, such as the women’s movement of the 1960s, on the encoding of stereotypes.

Let’s assume in our textbook project we are interested in gathering information about English’s expression of the semantic concepts of manner and motion. For learners of English, this can be

an area of difficulty as languages differ in how these semantic properties are expressed. English is a good example of a “satellite-framed” language, that is that manner and motion are often encoded in the same verb with a particle encoding the motion path (“rush out”, “climb up”). Other languages such as Spanish, Turkish, and Japanese are “verb-framed” languages, that is that motion but not manner is encoded in the verb (“salir corriendo”, “koşarak çıkmak”, “ ”).

We can use vector space modeling to represent the distribution of verbs in the MASC dataset and then target the concepts of manner and motion to then explore how English encodes these concepts. The question will be what will our features be. They could be terms, lemmas, pos tags, etc. Or they could be some combination. Considering the task at hand which we will ultimately want to know something about verbs, it makes sense to include the part of speech information in combination with either the term or the lemma.

If we include term and pos then we have a feature for every morphological variant of the term (e.g. house_VB, housed_VBD, housing_VBG). This can make the model more sizeable than it needs to be. If we include lemma and pos then we have a feature for every lemma with a distinct grammatical category (e.g. house_NN, house_VB). Note that as the pos tags are from the Penn tagset, many morphological variants appear in the tag itself (e.g. house_VB, houses_VBZ, housing_VBG). This is a good example of how the choice of features can impact the size of the model. In our case, it is not clear that we need to include the morphological variants of the verbs, so we will use lemma and a simplified pos as our features.

To engineer these features we will need to simplify the tags. We will conflate Penn tagset distinctions between nouns, verbs, adjectives, and adverbs. This will give us a feature for every lemma with a distinct grammatical category (e.g. house_NOUN, house_VERB), and no more than that. To do this we can apply the `case_when()` function from the `dplyr` package. The `case_when()` function takes a series of logical statements and returns a value based on the first logical statement that is `TRUE`. Let’s conflate the Penn tagset distinctions between nouns, verbs, adjectives, and adverbs, as seen in Example 8.37.

Example 8.37.

```
# Conflate Penn tagset into Universal-like tagset
masc_tbl <-
  mascot_tbl |>
  mutate(xpos = case_when(
    pos %in% c("NN", "NNS", "NNP", "NNPS") ~ "NOUN",
    pos %in% c("VB", "VBD", "VBG", "VBN", "VBP", "VBZ") ~ "VERB",
    pos %in% c("JJ", "JJR", "JJS") ~ "ADJ",
    pos %in% c("RB", "RBR", "RBS") ~ "ADV",
    TRUE ~ pos # keep other tags
  ))
```

```
# Lemma + pos
masc_tbl <-
  mascot_tbl |>
  mutate(lemma_pos = str_c(lemma, xpos, sep = "_"))

# Preview
masc_tbl |> glimpse()
```

```
> Rows: 439,539
> Columns: 9
> $ doc_id    <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", ~
> $ modality  <chr> "Written", "Written", "Written", "Written", "Written", "Written", ~
> $ genre     <chr> "Letters", "Letters", "Letters", "Letters", "Letters", "Letters", ~
> $ term_num  <dbl> 2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, ~
> $ term      <chr> "Your", "contribution", "to", "will", "mean", "more", "than", ~
> $ lemma     <chr> "your", "contribution", "to", "will", "mean", "more", "than", ~
> $ pos        <chr> "PRP$", "NN", "TO", "MD", "VB", "JJR", "IN", "PRP", "MD", "VB", ~
> $ xpos       <chr> "PRP$", "NOUN", "TO", "MD", "VERB", "ADJ", "IN", "PRP", "MD", "VB", ~
> $ lemma_pos <chr> "your_PRP$", "contribution_NOUN", "to_TO", "will_MD", "mean_IN", ~
```

When VSM is applied to words, it is known as **word embedding**. To calculate word embeddings there are various algorithms that can be used (BERT, word2vec, GloVe, etc.) The most common algorithm is **word2vec** (Mikolov et al. 2013). Word2vec is a neural network-based algorithm that learns word embeddings from a large corpus of text. In the word2vec algorithm the researcher can choose to learn embeddings from a Continuous Bag of Words (CBOW) or a Skip-gram model. The CBOW model predicts a target word based on the context words. The Skip-gram model predicts the context words based on the target word. The CBOW model is faster to train and is better for frequent words. The Skip-gram model is slower to train and is better for infrequent words.

Another consideration to take into account is the size of the corpus used to train the model. VSM provide more reliable results when trained on larger corpora. The MASC dataset is relatively small. We've simplified our features in order to have a smaller vocabulary in hopes to offset this limitation to a degree. But the choice of either CBOW or Skip-gram can also help to offset this limitation. CBOW can be better for smaller corpora as it aggregates context information.

To implement the word2vec algorithm on our lemma + pos features, we will use the `word2vec` package. The `word2vec()` function takes a text file and uses it to train the vector representations. To prepare the MASC dataset for training, we will need to write the lemma + pos features to a text file as a single character string. We can do this by first collapsing the

`lemma_pos` variable into a single string for each document using the `str_c()` function from the `stringr` package. Then we can use the `writeLines()` function to write the string to a text file. Let's prepare the MASC dataset for training, as seen in Example 8.38.

Example 8.38.

```
# Prepare data for word2vec training
masc_tbl |>
  summarize(text = str_c(lemma_pos, collapse = " ")) |>
  pull(text) |>
  writeLines(
    con = "../data/derived/masc_word2vec.txt"
  )
```

With our `masc_word2vec.txt` file, we read in, apply the word2vec algorithm using the `word2vec` package, and write the model to disk. By default, the `word2vec()` function applies the CBOW model, with 50 dimensions, a window size of 5, and a minimum word count of 5. We can change these parameters as needed, but let's apply the default algorithm to the text file splitting features by sentence punctuation, as seen in Example 8.39.

Example 8.39.

```
# Load package
library(word2vec)

# Training word2vec model
masc_model <-
  word2vec(
    x = "../data/derived/masc_word2vec.txt",
    split = c(" ", ".?!" ),
  )

# Write model to disk
write.word2vec(
  masc_model,
  file = "../data/derived/masc_word2vec.bin"
)

# Set seed for reproducibility
set.seed(1234)
```

```

# Load package
library(word2vec)

# Train word2vec model
masc_model <-
  word2vec(
    x = "data/masc_word2vec.txt",
    split = c(" ", ".?!" ),
  )

# Write model to disk
write.word2vec(
  mascot_model,
  file = "data/masc_word2vec.bin"
)

```

Writing the model to disk is important as it allows us to read the model in without having to retrain it. In cases where the corpus is large, this can save a lot of computational time.

Now that we have a trained model, we can read it in with the `read.vectors()` function from the `wordVectors` package.

Example 8.40.

```

# Load package
library(wordVectors)

# Read word2vec model
masc_model <-
  read.vectors(
    filename = "../data/derived/masc_word2vec.bin"
  )

```

The `read.vectors()` function returns a matrix where each row is a term in the model and each column is a dimension in the vector space, as seen in Example 8.41.

Example 8.41.

```
# Inspect
dim(masc_model)
```

```
> [1] 5892 50
```

```
# Preview
masc_model[1:5, 1:5]
```

```
> A VectorSpaceModel object of 5 words and 5 vectors
>           [,1]   [,2]   [,3]   [,4]   [,5]
> map_VERB      -1.487  1.510 -0.662 -1.2129 -0.6458
> KGCUBS10_NOUN -0.387  1.751 -0.603  0.6270 -0.1965
> MICKEY3_NOUN   0.675  1.035 -0.516  1.1289 -0.0784
> amenity_NOUN   -0.311  0.682 -1.225  0.0541 -0.8392
> transmission_NOUN -0.837 -0.354 -1.193 -0.5576 -0.7866
> attr(,".cache")
> <environment: 0x7fceb0832e48>
```

The row-wise vector in the model is the vector representation of each feature. The notion is that these values can now be compared with other terms to explore distributional relatedness. We can extract specific features from the matrix using the [] operator.

As an example, let's compare the vectors for noun-verb pairs for the lemmas 'run' and 'walk'. To do this we extract these features from the model. To appreciate the relatedness of these features it is best to visualize them. We can do this by first reducing the dimensionality of the vectors using principal components analysis (PCA). We can then plot the first two principle components, as seen in Figure 8.13.

Example 8.42.

```
# Extract vectors
word_vectors <-
  masc_model[c("run_VERB", "walk_VERB", "run_NOUN", "walk_NOUN"), ] |>
  as.matrix()

pca <-
  word_vectors |>
  scale() |>
  prcomp()
```

```

pca_tbl <-
  as_tibble(pca$x[, 1:2]) |>
  mutate(word = rownames(word_vectors))

pca_tbl |>
  ggplot(aes(x = PC1, y = PC2, label = word)) +
  geom_point() +
  ggrepel::geom_text_repel()

```

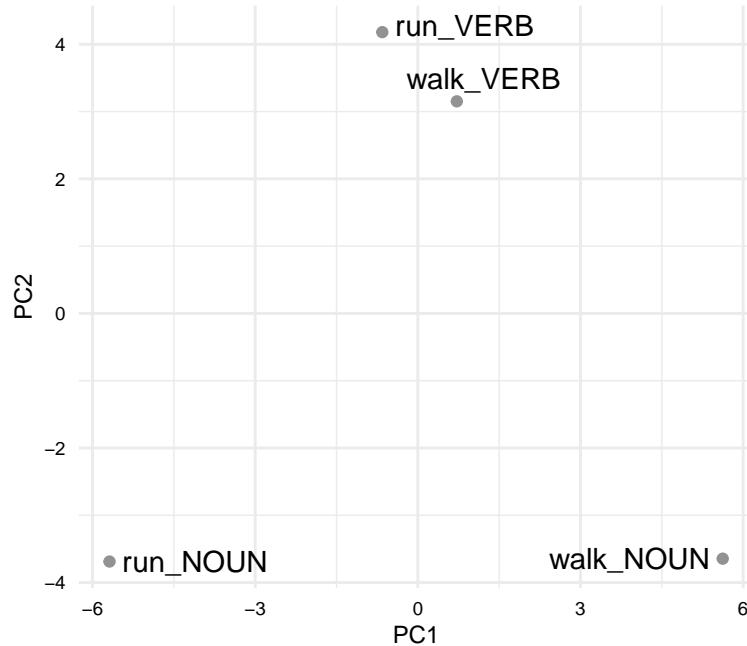


Figure 8.13: Similarity between ‘run’ and ‘walk’ in the MASC dataset

From Figure 8.13, we can see that each of these features occupies a distinct position in the reduced vector space. But on closer inspection, we can see that there is a relationship between the lemma pairs. Remember that PCA reduces the dimensionality of the data by identifying the dimensions that capture the greatest amount of variance in the data. This means that of the 50 dimensions in the model, the PC1 and PC2 correspond to orthogonal dimensions that capture the greatest amount of variance in the data. If we look along PC1, we can see that there is a distinction between part-of-speech. Looking along PC1, we see some pariyty between lemma meanings. Given these features, we can see that meaning and grammatical category can be captured in the vector space.

An interesting property of vector space models is that we can build up a dimension of meaning by adding vectors that are expected to approximate that meaning. For example, we can add the vectors for typical motion verbs to create a vector for motion-similarity and one for manner-similarity. We can then compare the feature vectors for all verbs and assess their motion-similarity and manner-similarity.

To do this let's first subset the model to only include verbs, as in Example 8.43. We will also remove the part-of-speech tags from the rownames of the matrix as they are no longer needed.

Example 8.43.

```
# Filter to verbs
verbs <- str_subset(rownames(masc_model), ".*_VERB")
verb_vectors <- masc_model[verbs, ]

# Remove part-of-speech tags
rownames(verb_vectors) <-
  verb_vectors |>
  rownames() |>
  str_replace_all("_VERB", "")

# Inspect
dim(verb_vectors)
```

```
> [1] 1115 50
```

```
# Preview
verb_vectors[1:5, 1:5]
```

```
> A VectorSpaceModel object of 5 words and 5 vectors
>      [,1] [,2] [,3] [,4] [,5]
> map     -1.4875 1.510 -0.662 -1.2129 -0.646
> whip     0.0496 0.919 -1.020  0.3087  0.350
> enroll   -0.6251 1.620 -1.878  0.0861 -0.428
> tuck     0.5610 1.054 -1.331 -0.5298  1.598
> suppress 0.3824 1.928 -1.750 -0.3229 -0.643
> attr(".cache")
> <environment: 0x7fcf05391490>
```

We now have `verb_vectors` which includes the vector representations for all verbs 1115 in the MASC dataset. Next, let's seed the vectors for motion-similarity and manner-similarity and calculate the vector 'closeness' to the motion and manner seed vectors with the `closest_to()` function from the `wordVectors()` package.

Example 8.44.

```
# Add vectors for motion-similarity and manner-similarity
motion <-
  c("go", "come", "leave", "arrive", "enter", "exit", "depart", "return")

motion_similarity <-
  verb_vectors |> closest_to(motion, n = Inf)

# Preview
motion_similarity |> glimpse()

> Rows: 1,115
> Columns: 2
> $ word           <chr> "tie", "approach", "enter", "step", "race", "ru-
> $ `similarity to motion` <dbl> 0.803, 0.803, 0.786, 0.784, 0.781, 0.770, 0.769~

manner <-
  c("run", "walk", "jump", "crawl", "swim", "fly", "drive", "ride")

manner_similarity <-
  verb_vectors |> closest_to(manner, n = Inf)

# Preview
manner_similarity |> glimpse()

> Rows: 1,115
> Columns: 2
> $ word           <chr> "walk", "drop", "pull", "climb", "step", "throw-
> $ `similarity to manner` <dbl> 0.885, 0.877, 0.864, 0.860, 0.859, 0.842, 0.840~
```

The `motion_similarity` and `manner_similarity` data frames each contain all the verbs with a corresponding closeness measure. We can join these two data frames by feature to create a single data frame with the motion-similarity and manner-similarity measures, as seen in Example 8.45.

Example 8.45.

```

# Join motion-similarity and manner-similarity
manner_motion_similarity <-
  manner_similarity |>
  inner_join(motion_similarity)

# Preview
manner_motion_similarity |> glimpse()

> Rows: 1,115
> Columns: 3
> $ word           <chr> "walk", "drop", "pull", "climb", "step", "throw"
> $ `similarity to manner` <dbl> 0.885, 0.877, 0.864, 0.860, 0.859, 0.842, 0.840~
> $ `similarity to motion` <dbl> 0.743, 0.751, 0.623, 0.753, 0.784, 0.582, 0.709~

```

The result of Example 8.45 is a data frame with the motion-similarity and manner-similarity measures for all verbs in the MASC dataset. We can now visualize the distribution of motion-similarity and manner-similarity measures, as seen in Figure 8.14.

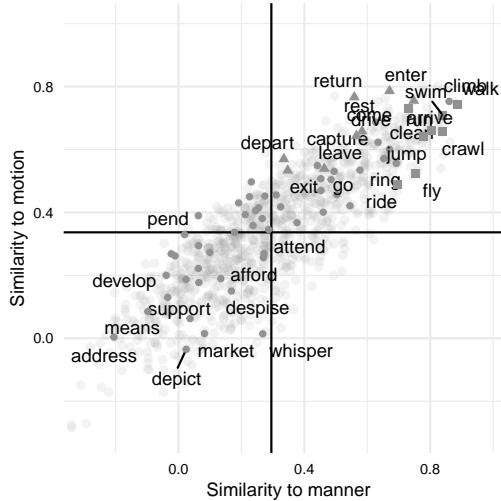


Figure 8.14: Motion-similarity and manner-similarity of verbs in the MASC dataset

From Figure 8.14, we can see that the manner-similarity is plotted on the x-axis and the motion-similarity on the y-axis. I've added horizontal and vertical lines to break the scatterplot into quadrants –the top-right corresponding to high manner- and motion-similarity and the bottom-left corresponding to low manner- and motion-similarity. This captures the majority

of the verbs in the dataset. The verbs in the top-left quadrant have high motion-similarity but lower manner similarity, and verbs in the bottom-right quadrant have high manner-similarity but lower motion-similarity.

I've randomly sampled 50 verbs from the dataset and plotted them as text labels. I've also plotted the motion and manner seed vectors as triangle and box points, respectively. We can see that motion- and manner-similarity seed verbs are found in the top-left quadrant together, showing that they are semantically related. Verbs in the other quadrants are either lower in motion- or manner-similarity, or both. From a qualitative point of view it appears that many of the verbs coincide with intuition. Some, however, less so. This is to be expected to some degree as the model is trained on a relatively small corpus. All in all, this serves as an example of how vector space modeling can be used to explore semantic relationships between linguistic features.

8.3 Summary

In this chapter we surveyed a range of methods for uncovering insights from data, particularly when we do not have a predetermined hypothesis. We broke the chapter discussion along the two central branches of exploratory data analysis: descriptive analysis and unsupervised learning. Descriptive analysis offers statistical or visual summaries of datasets through frequency, dispersion, and co-occurrence measures, while unsupervised learning utilizes machine learning techniques to uncover patterns without predefining variable relationships. Here we covered a few unsupervised learning methods including clustering, dimensionality reduction, and vector space modeling. Through either descriptive or unsupervised learning methodologies, we probe questions in a data-driven fashion and apply methods to summarize, reduce, and sort complex datasets. This in turn facilitates novel, quantitative perspectives that can subsequently be evaluated qualitatively, offering us a robust approach to exploring and generating research questions.

Activities

Recipe

What: Exploratory methods: descriptive and unsupervised learning analysis methods^a

How: Read Recipe 8 and participate in the Hypothes.is online social annotation.

Why: To illustrate how to prepare a dataset for descriptive and unsupervised machine learning methods and evaluate the results for exploratory data analysis.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-8.html>

💻 Lab

What: Exploratory Data Analysis^a

How: Clone, fork, and complete the steps in Lab 8.

Why: To gain experience working with coding strategies to prepare, feature engineer, explore, and evaluate results from exploratory data analyses, practice transforming datasets into new object formats and visualizing relationships, and implement organizational strategies for organizing and reporting results in a reproducible fashion.

^a<https://github.com/qtalr/lab-8>

Questions

✍ Conceptual questions

1. What is exploratory data analysis?
2. How can exploratory data analysis be used to uncover patterns and associations?
3. Describe the workflow of exploratory data analysis?
4. What are the advantages and disadvantages of descriptive analysis?
5. What are the advantages and disadvantages of unsupervised learning?
6. What is the difference between supervised and unsupervised learning?
7. How does exploratory data analysis differ from traditional hypothesis testing?

✍ Technical questions

1. Write a function in R to conduct a hierarchical cluster analysis on a dataset.
2. Implement a k-means algorithm in R to identify clusters within a dataset.
3. Implement a Principal Component Analysis (PCA) algorithm in R to identify patterns and associations within a dataset.
4. Write a function in R to produce a descriptive summary of a dataset.
5. Conduct a correlation analysis in R to identify relationships between variables in a dataset.
6. Load a dataset into R and conduct a frequency analysis on the dataset.
7. Load a dataset into R and conduct a keyword in context analysis on the dataset.
8. Load a dataset into R and conduct a keyword analysis on the dataset.
9. Load a dataset into R and conduct a sentiment analysis on the dataset.
10. Load a dataset into R and conduct a topic modelling analysis on the dataset.

9 Prediction



Draft

Ready for review.

All models are wrong, but some are useful.

— George E.P. Box

▀ Outcomes

- Identify the research goals of predictive data analysis
- Describe the workflow for predictive data analysis
- Recognize quantitative and qualitative methods for evaluating predictive models

In this chapter, I introduce supervised learning as an approach to data analysis, specifically focusing on its applications in text analysis. Supervised learning aims to establish a relationship between a target (or outcome) variable and a set of feature variables derived from text data. By leveraging this relationship, statistical generalizations (models) can be created to accurately predict values of the target variable based on the values of the feature variables. Throughout the chapter, we explore practical tasks and theoretical applications of statistical learning in text analysis. We also cover the standard workflow for building predictive models, testing and evaluating model performance, improving model accuracy, and interpreting and reporting findings.

➤_ Lessons

What: Supervised Learning^a

How: In the R Console pane load `swirl`, run `swirl()`, and follow prompts to select the lesson.

Why: ... 🔧

^a<https://github.com/qtalr/lessons>

9.1 Orientation

In this section, I introduce the concept of supervised learning and provide an overview of the workflow for building and evaluating predictive models for text analysis. First we will discuss the research goals that are typically addressed using supervised learning, contrasting them with the goals of exploratory and inferential analysis. Next, we will discuss the approaches that are typically used to address these goals, including the types of data structures and algorithms that are used. Finally, we will discuss the workflow for building predictive models, including the steps for preparing data, training and testing models, and evaluating and reporting results.

9.1.1 Research goal

Predictive data analysis (PDA) is a powerful analysis method for linguists and other researchers interested in making predictions about new or future data based on patterns in existing data. As discussed in Section 3.2.2 and Section 4.4.1, PDA is a type of supervised learning, which means that it involves training a model on a labeled dataset where the input data and desired output are both provided. The model is able to make predictions or classifications based on the input data by learning the relationships between the input and output data. Supervised machine learning is an important tool for linguists studying language and communication, as it allows us to analyze language data to identify patterns or trends in language use, verify hypotheses, and prescribe actions.

In contrast to EDA, PDA does require that we have a particular goal in mind from the outset. This goal is to predict a fixed outcome variable based on a set of predictor variables. However, PDA can be applied with distinct aims in mind: exploratory-oriented or hypothesis-driven prediction. In exploratory-oriented prediction, the goal is to examine potential relationships between an outcome and a mutable set of predictor variables. The results of this type of analysis can be used to generate new insight and questions. In hypothesis-driven prediction, in contrast, the goal is to use predictive data analysis to test a hypothesis about the relationship between the outcome and a pre-defined set of predictor variables. In this case, the results of the analysis is used to explain or infer a relationship between the outcome and predictor variables.

Dive deeper

- MuPDAR approach (Multifactorial Prediction and Deviation Analysis with Regressions) (Deshors and Gries 2016; S. Th. Gries and Deshors 2014) and
 - Training on group
 - Testing on another group (or groups)
- Discriminant analysis (Baayen 2011)

It is important to establish which aim is being taken, as this will influence the approach that is taken.

9.1.2 Approach

The approach to conducting predictive analysis shares some commonalities with exploratory data analysis (Section 8.1.2) (as well as inferential analysis Chapter 10), but there are also some key differences. Let's look at the workflow in Table 9.1 and then discuss these commonalities and differences.

Table 9.1: Workflow for predictive data analysis

| Step | Name | Description |
|------|-----------------------|---|
| 1 | Identify | Consider the research question and aim and identify relevant variables |
| 2 | Initial split | Split the data into representative training and testing sets |
| 3 | Integrate | Apply variable selection and engineering procedures |
| 4 | Inspect | Inspect the data to ensure that it is in the correct format and that the training and testing sets are representative of the data |
| 5 | Interrogate | Train and evaluate the model on the training set, adjusting models or hyperparameters as needed, to produce a final model |
| 6 | Interpret | Interpret the results of the final model in light of the research question or hypothesis |
| 7 | (Optional)
Iterate | Repeat steps 3-6 to selecting new variables |

Focusing on the overlap with other analysis methods, we can see some fundamental steps such as identifying relevant variables, inspecting the data, interrogating the data, and interpreting the results. And if our research aim is exploratory in nature, iteration may also be a part of the workflow. These steps highlight the importance conducting methodologic and communicable research, as discussed in Section 4.1.

There are two main differences, however, between the PDA and the EDA workflow we discussed in Chapter 8. The first, reflected the majority of the steps in the workflow, is that PDA requires partitioning the data into training and testing sets. As discussed in Section 3.2.2, the training set is used to develop the model, and the testing set is used to evaluate the model's performance. This strategy is used to ensure that the model is robust and generalizes well to new data. It is well known, and makes intuitive sense, that using the same data to develop and evaluate a model likely will not produce a model that generalizes well to new data. This is because the model will have potentially conflated the nuances of the data ('the noise') with any real

Table 9.2: A labeled confusion matrix

| | Actual positive | Actual negative |
|--------------------|-----------------|-----------------|
| Predicted positive | TP | FP |
| Predicted negative | FN | TN |

trends ('the signal') and therefore will not be able to generalize well to new data. This is called overfitting and by holding out a portion of the data for testing, we can evaluate the model's performance on data that it has not seen before and therefore get a more accurate estimate of the generalizable trends in the data.

Another procedure to avoid the perils of overfitting, is to use resampling methods as part of the model evaluation on the training set. Resampling is the process of repeatedly drawing samples from the training set and evaluating the model on each sample. The two most common resampling methods are **bootstrapping** (resampling with replacement) and **cross-validation** (resampling without replacement). The performance of these multiple models are summarized and the error between them is assessed. The goal is to minimize the performance differences between the models while maximizing the overall performance. These measures go a long way to avoiding overfitting and therefore maximizing the chance that the training phase will produce a model which is robust at the testing phase.

The second difference, not reflected in the workflow but inherent in predictive analysis, is that PDA requires a fixed outcome variable. This means that the outcome variable must be defined from the outset and cannot be changed during the analysis. Furthermore, the informational nature of the outcome variable will dictate the what type of algorithm we choose to interrogate the data and how we will evaluate the model's performance. If the outcome is categorical in nature, we will use a **classification algorithm** (*e.g.* logistic regression, naive bayes, *etc.*). Classification evaluation metrics include accuracy, precision, recall, and F1 score which can be derived from and visualized in a cross-tabulation of the predicted and actual outcome values.

To understand these measures it is helpful to consider a confusion matrix, which is a table that describes the performance of a classification model on data for which the true values are known. The confusion matrix is a two-by-two matrix that shows the number of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN), as seen in Table 9.2.

If the outcome is numeric in nature, we will use a **regression algorithm** (*e.g.* linear regression, support vector regression, *etc.*). Since the difference between prediction and actual values is numeric, metrics that quantify numerical differences, such as root mean square error (RMSE) or R^2 , are used to evaluate the model's performance.

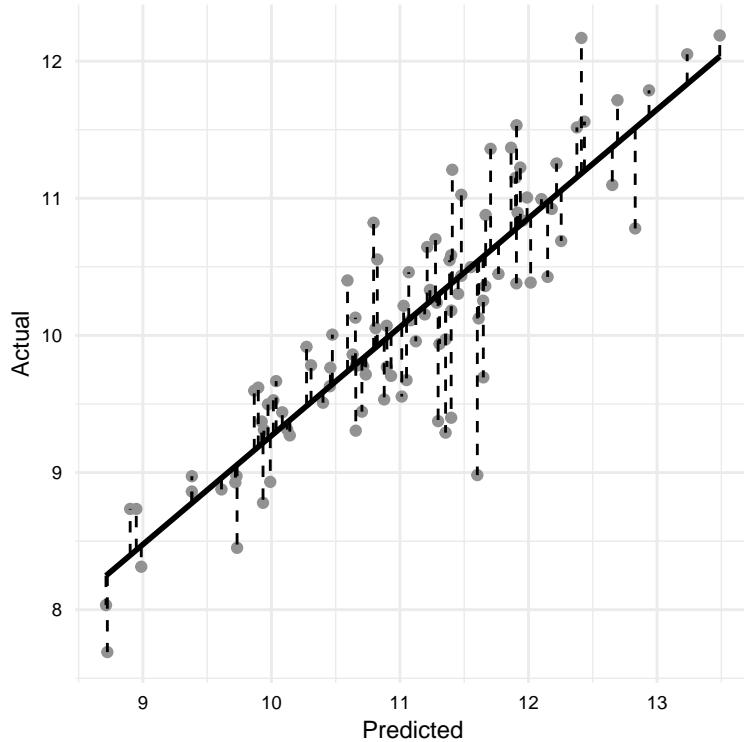


Figure 9.1: A plot of the actual and predicted values for a regression model

The evaluation of the model is quantitative on the one hand, but it is also qualitative in that we need to consider the implications of the model's performance in light of the research question or hypothesis. Furthermore, depending on our research question we may be interested in exploring the features that are most important to the model's performance. This is called **feature importance** and can be derived from the model's coefficients or weights. Notably, however, some of the most powerful models in use today, such as deep neural networks, are not easily interpretable and therefore feature importance is not easily derived. This is something to keep in mind when considering the research question and the type of model that will be used to address it.

9.2 Analysis

In this section we now turn to the practical application of predictive data analysis. The discussion will be separated into classification and regression tasks, as model selection and evaluation procedures differ between the two. For each task, we will frame a research goal and work through the process of building a predictive model to address that goal. Along the

Table 9.3: Data dictionary for the CEDEL2 corpus

| variable | name | variable_type | description |
|-----------------|-----------------|---------------|---|
| doc_id | Document ID | numeric | Unique identifier for each document |
| subcorpus | Subcorpus | categorical | The subcorpus to which the document belongs ('Learn') |
| placement_score | Placement Score | numeric | The score obtained by the document author in a placement test |
| proficiency | Proficiency | ordinal | The level of language proficiency of the document author |
| text | Text | character | The written text provided by the document author |

way we will cover concepts and methods that are common to both classification and regression tasks and specific to each.

To frame our analyses, we will posit research aimed at identifying language usage patterns in second language use, one for a classification task and one for a regression task. Our first research question will be to identify potential salient differences in Spanish language use between natives and L1 English learners (categorical). Our second research question will be to gauge the extent to which the L1 English learners' Spanish language placement test scores (numeric) can be predicted based on their language use.

We will use data from the CEDEL2 corpus¹. We will include a subset of the variables from this data that are relevant to our research questions. The data dictionary for this dataset is seen in Table 9.3.

We will be using the `tidymodels` framework in R to perform this analysis. `tidymodels` is a metapackage, much like `tidyverse`, that provides a consistent interface for modeling and machine learning. Some key packages unique to `tidymodels` are `recipes`, `parsnip`, `workflows`, and `tune`. `recipes` includes functions for preprocessing and engineering features. `parsnip` provides a consistent interface for specifying modeling algorithms. `workflows` allows us to combine recipes and models into a single pipeline. Finally, `tune` give us the ability to evaluate and tune hyperparameters of models.

Since we are using text data, we will also be using the `textrecipes` package which makes various functions available for preprocessing text and extracting and engineering features.

Let's go ahead and do the setup, loading the necessary packages and data. This is seen in Example 9.1.

Example 9.1.

```
# Load packages
library(tidymodels)  # modeling metapackage
library(textrecipes) # text preprocessing
```

¹See Appendix A.3 for more information on the CEDEL2 corpus.

```

library(janitor)      # data inspection

# Set global options
tidymodels_prefer()  # prefer tidymodels functions over other functions with
# the same name

# Read in the dataset
cedel2_df <-
  read_csv("../data/cedel2/cedel2_df.csv")

# Preview
cedel2_df |> glimpse()

```

```

> Rows: 2,957
> Columns: 5
> $ doc_id      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, ~
> $ subcorpus   <chr> "Learner", "Learner", "Learner", "Learner", ~
> $ placement_score <dbl> 14.0, 16.3, 16.3, 18.6, 18.6, 18.6, 20.9, 20.9, ~
> $ proficiency <chr> "Lower beginner", "Lower beginner", "Lower beginner", ~
> $ text         <chr> "Yo vivo es Atlanta, Georgia. Atlanta es muy grande ciu~
```

The results from Example 9.1 show that we have five variables and 2957 observations. The primary variables for the classification task are the `subcorpus` variable and the `text` variable. The `placement_score` variable is the outcome variable for the regression task.

9.2.1 Text classification

The goal of this analysis is to classify texts as either native or learner based on the writing samples. This is a binary classification problem. We will approach this problem from an exploratory perspective, and therefore our aim is to derive features from the text that best distinguish between the two classes.

Let's modify the data frame to include only the variables we need for this analysis. In the process, we will rename the `subcorpus` variable to `outcome` to reflect that it is the outcome variable and convert it to a factor vector to meet requirements of the modeling functions we will use in our analysis. This is seen in Example 9.2.

Example 9.2.

```

# Select variables and factor outcome
cedel2_cls_df <-
  cedel2_df |>
  select(outcome = subcorpus, proficiency, text) |>
  mutate(outcome = factor(outcome))

# Preview
cedel2_cls_df |> glimpse()

```

```

> Rows: 2,957
> Columns: 3
> $ outcome      <fct> Learner, Learner, Learner, Learner, Learner, Lear-
> $ proficiency <chr> "Lower beginner", "Lower beginner", "Lower beginner", "Low-
> $ text         <chr> "Yo vivo es Alanta, Georgia. Atlanta es muy grande ciudad.~
```

To view the distribution of the outcome variable between the two levels we can use the `tabyl()` function from the `janitor` package, as seen in Example 9.3.

Example 9.3.

```

# View the distribution of the outcome variable
cedel2_cls_df |>
  tabyl(outcome) |>
  adorn_pct_formatting(digits = 1)

```

```

> outcome      n percent
> Learner 1906  64.5%
> Native 1051  35.5%
```

So a little less than two-thirds of the texts are from learners. It is important to gauge the distribution of the outcome variable to see if it is balanced or imbalanced. The classes need not be perfectly balanced, but if they are wildly imbalanced it can cause problems for the model.

So in step 1 of our workflow (from Table 9.1), we need to identify the features that we will use to classify the texts. There may be many features that we could use. These could be features derived from raw text (*e.g.* characters, words, n-grams, *etc.*), feature vectors (*e.g.* word embeddings), or meta-linguistic features (*e.g.* part-of-speech tags, syntactic parses, or semantic features) that have been derived from these through manual or automatic annotation.

Let's start simple and use a bag-of-words approach to classify texts where words are the features. This is a simple approach that is often used as a baseline for more complex models. If it doesn't work well, we can try something else.

This provides us the linguistic unit we will use but we still need to decide how to represent these words. Do we use raw token counts? Do we use normalized frequencies? Do we use some type of weighting scheme? These are questions that we need to consider as we embark on this analysis. Since we are exploring we can use trial-and-error or consider the implications of each approach and choose the one that best fits our research question –or both.

As a first pass, let's use raw token counts with our word features.

With our features identified, we can move on to step 2 of our workflow and split the data into training and testing sets. We make the splits to our data at this point to draw a line in the sand between the data we will use to train the model and the data we will use to test the model. A typical approach in supervised machine learning is to allocate around 75-80% of the data to the training set and the remaining 20-25% to the testing set, depending on the number of observations. We have 2957 observations in our data set, so we can allocate 80% of the data to the training set and 20% of the data to the testing set.

In Example 9.4, we will use the `initial_split()` function from the `rsample` package to split the data into training and testing sets. The `initial_split()` function takes a data frame and a proportion and returns a `split` object which contains the training and testing sets. We will use the `strata` argument to stratify the data by the `outcome` variable. This will ensure that the training and testing sets have the same proportion of native and learner texts.

Example 9.4.

```
# Split the data into training and testing sets
cedel2_cls_split <-
  initial_split(
    data = cedel2_cls_df,
    prop = 0.8,
    strata = outcome
  )

# Create training set
cedel2_cls_train <- training(cedel2_cls_split) # 80% of data

# Create testing set
cedel2_cls_test <- testing(cedel2_cls_split) # 20% of data
```

A confirmation of the distribution of the data across the training and testing sets as well as a break down of the outcome variable can be seen in Example 9.5.

Example 9.5.

```
# View the distribution of the outcome variables
cedel2_cls_train |>
  tabyl(outcome) |>
  adorn_totals("row") |>
  adorn_pct_formatting(digits = 1)
```

```
> outcome      n percent
> Learner 1524  64.5%
> Native   840  35.5%
> Total    2364 100.0%
```

```
cedel2_cls_test |>
  tabyl(outcome) |>
  adorn_totals("row") |>
  adorn_pct_formatting(digits = 1)
```

```
> outcome      n percent
> Learner 382  64.4%
> Native   211  35.6%
> Total    593 100.0%
```

We can see that the split was successful. The training and testing sets have very similar proportion of native and learner texts.

We are now ready to create a ‘recipe’, step 3 in our analysis. A recipe is a set of instructions or blueprint which specify the outcome variable and the feature variable and determines how to preprocess and engineer the feature variables.

We will use the `recipe()` function from the `recipes` package to create the recipe. The `recipe()` function minimally takes a formula and a data frame and returns a `recipe` object. The formula specifies the outcome variable (y) and the feature variable(s) ($x_1..x_n$). For example $y \sim x$ can be read as “ y is a function of x ”. In our particular case, we will use the formula `outcome ~ text` to specify that the outcome variable is the `outcome` variable and the feature variable is the `text` variable. The code is seen in Example 9.6.

Example 9.6.

```

# Create a recipe
cedel2_cls_rec <-
  recipe(
    outcome ~ text,
    data = cedel2_cls_train
  )

# Preview
cedel2_cls_rec

```

The recipe object at this moment contains just one instruction, what the variables are and what their relationship is.

Tip

R formulas are a powerful way to specify relationships between variables and are used extensively in data modeling including exploratory, predictive, and inferential analysis. The basic formula syntax is $y \sim x$ where y is the outcome variable and x is the feature variable. The formula syntax can be extended to include multiple feature variables, interactions, and transformations. For more information on R formulas, see R for Data Science^a.

^a<https://r4ds.github.io/bookclub-tmwr/r-formula-syntax.html>

The `recipes` package provides a wide range of `step_*`() functions which can be applied to the recipe to specify how to engineer the variables in our recipe call. These include functions to scale (*e.g.* `step_center()`, `step_scale()`, *etc.*) and transform (*e.g.* `step_log()`, `step_pca()`, *etc.*) numeric variables, and functions to encode (*e.g.* `step_dummy()`, `step_labelencode()`, *etc.*) categorical variables.

These step functions are great when we have selected the variables we want to use in our model and we want to engineer them in a particular way. In our case, however, we need to derive features from the text in the `text` column of datasets before we engineer them. To ease this process, the `textrecipes` package provides a number of step functions for preprocessing text data. These include functions to tokenize (*e.g.* `step_tokenize()`), remove stop words (*e.g.* `step_stopwords()`), and to derive meta-features (*e.g.* `step_lemma()`, `step_stem()`, *etc.*) ². Furthermore, there are functions to engineer features in ways that are particularly relevant to text data, such as feature frequencies and weights (*e.g.* `step_tf()`, `step_tfidf()`, *etc.*) and token filtering (*e.g.* `step_tokenfilter()`).

²Note that functions for meta-features require more sophisticated text analysis software to be installed on the computing environment (*e.g.* `spacyr` for `step_lemma()`, `step_pos()`, *etc.*). See the `textrecipes` package documentation for more information.

So let's build on our basic recipe `cedel2_cls_rec` by adding steps relevant to our task. To extract our features, we will use the `step_tokenize()` function to tokenize the text into words. The default behavior of the `step_tokenize()` function is to tokenize the text into words, but other token units can be derived and various options can be added to the function call (as the `tokenizers` package is used under the hood). Adding the `step_tokenize()` function to our recipe is seen in Example 9.7.

Example 9.7.

```
# Add step to tokenize the text
cedel2_cls_rec <-
  cedel2_cls_rec |>
  step_tokenize(text)

# Preview
cedel2_cls_rec
```

The recipe object now contains two instructions, one for the outcome variable and one for the feature variable. The feature variable instruction specifies that the text should be tokenized into words.

We now need to consider how to engineer the word features. If we add `step_tf()` we will get a matrix of token counts by default. We also have the option to add `step_tfidf()` to get a matrix of term frequencies weighted by inverse document frequency, which effectively down weights words that are common across all documents.

We decided in step 1 that we will start with raw token counts, so we will add `step_tf()` to our recipe. This is seen in Example 9.8.

Example 9.8.

```
# Add step to tokenize the text
cedel2_cls_rec <-
  cedel2_cls_rec |>
  step_tf(text)

# Preview
cedel2_cls_rec
```

To make sure things are in order and that the recipe performs as expected, we can use the functions `prep()` and `bake()` to inspect the recipe. The `prep()` function takes a recipe object and a data frame and returns a `prep` object. The `prep` object contains the recipe and the data

frame with the feature variables engineered according to the recipe. The `bake()` function takes a `prep` object and an optional new dataset to apply the recipe to. If we only want to see the application to the training set, we can use the `new_data = NULL` argument.

In Example 9.9, we use the `prep()` function to create a `prep` object and then use the `bake()` function to create a data frame with the feature variables. We can then inspect the data frame to see if the recipe performed as expected.

Example 9.9.

```
# Create a prep object
cedel2_cls_prep <-
  prep(
    cedel2_cls_rec,
    training = cedel2_cls_train
  )

# Create a data frame with the feature variables
cedel2_cls_bake <-
  bake(
    cedel2_cls_prep,
    new_data = NULL
  )

# Preview
cedel2_cls_bake |> dim()
```

```
> [1] 2364 37907
```

```
cedel2_cls_bake[
  c(1000:1001, 2000:2001),      # selected rows
  c(1, 1000:1001, 20000:20001) # selected columns
]
```

```
> # A tibble: 4 x 5
>   outcome tf_text_actuación tf_text_actuaciones tf_text_inservibles
>   <fct>          <int>          <int>          <int>
> 1 Learner          0            0            0
> 2 Learner          0            0            0
> 3 Native           0            0            0
```

```
> 4 Native          0          0          0
> # i 1 more variable: tf_text_inside <int>
```

The resulting engineered features data frame has 2364 observations and 37907 variables. The first variable is the outcome variable and the remaining variables are the engineered features. We can see that the recipe performed as expected and that the feature variables are the token counts for each word in the text.



Tip

When applying tokenization and feature engineering steps to text data the result is often contained in a matrix object. In the the `recipes` package a data frame with a matrix-like structure is returned.

But we should pause. We have a lot of features! One for every single word in the entire corpus. This is an unwieldy number of features for a model and it is likely that many of these features are not useful for our classification task. Furthermore, the more features we have, the more chance these features capture the nuances of these particular writing samples, thus, we are the more likely we are to overfit the model. All in all, we need to reduce the number of features.

Our domain knowledge about language can be of help us decide on how to approach reducing the feature set. On the one hand, we know that most tokens occur rarely in any sizable corpus. And the lower the frequency, the more likely that the token may reflect nuances of the speaker or the content of particular texts rather than generalizable trends. On the other hand, we know that a relatively small number of the most frequent words quickly account for a large proportion of the tokens in a corpus.

So just with these two considerations, we can see that we can filter out many infrequent words and likely have quite a few viable features. Let's start with an arbitrary threshold of the top 1,000 words by frequency. We can use the `step_tokenfilter()` function to filter out the top 1,000 words by frequency. This particular step needs to be applied before the `step_tf()` step, so we will add it to our recipe before the `step_tf()` step. This is seen in Example 9.10.

Example 9.10.

```
# Rebuild recipe with tokenfilter step
cedel2_cls_rec <-
  recipe(
    outcome ~ text,
    data = cedel2_cls_train
  ) |>
  step_tokenize(text) |>
  step_tokenfilter(text, max_tokens = 1000) |>
  step_tf(text)
```

```

# Prep and bake
cedel2_cls_prep <-
  prep(
    cedel2_cls_rec,
    training = cedel2_cls_train
  )

cedel2_cls_bake <-
  bake(
    cedel2_cls_prep,
    new_data = NULL
  )

# Preview
cedel2_cls_bake |> dim()

```

```
> [1] 2364 1001
```

```
cedel2_cls_bake[1:5, 1:10]
```

```

> # A tibble: 5 x 10
>   outcome tf_text_10 tf_text_2 tf_text_3 tf_text_4 tf_text_5 tf_text_a
>   <fct>     <int>     <int>     <int>     <int>     <int>     <int>
> 1 Learner      0         0         0         0         0         0
> 2 Learner      0         0         0         0         0         2
> 3 Learner      0         1         0         0         0         5
> 4 Learner      0         0         0         0         0         0
> 5 Learner      0         0         0         0         0         2
> # i 3 more variables: tf_text_abandonado <int>, tf_text_abuela <int>,
> #   tf_text_abuelos <int>

```

We now have a manageable set of features. Only during the interrogation step will we know if they are useful.

👉 Tip

The `prep()` and `bake()` functions are useful for inspecting the recipe and the engineered features, but they are not required to build a recipe. When a recipe is added to a workflow, the `prep()` and `bake()` functions are called automatically as part of the process.

There is one last step we should add to our recipe which concerns the skewed nature of word frequency distributions. The magnitude of the token counts will be highly skewed with a few words having very high counts and most words having relatively low counts, even for the top 1,000 words. This skew can be problematic for some models, so we will add a step to log normalize the token counts. This will not change the rank order, only the magnitude of the differences. Note that we use `all_predictors()` to apply the log transformation to all the word features, as seen in Example 9.11.

Example 9.11.

```
# Add log step to recipe
cedel2_cls_rec <-
  cedel2_cls_rec |>
  step_log(all_predictors(), offset = 1) # add 1 to avoid log(0)

cedel2_cls_rec
```

We are now ready to turn our attention to step 5 of our workflow, interrogating the data. In this step, we will first select a classification algorithm, then add this algorithm and our recipe to a workflow object. We will then use the workflow object to train and evaluate the model on the training set.

There are many classification algorithms to choose from with their own strengths and shortcomings. In Table 9.4, we list some of the most common classification algorithms and their characteristics.

Table 9.4: Classification algorithms

| Algorithm | Strengths | Shortcomings |
|-------------------------|--|--|
| Logistic regression | Simple, interpretable, fast | Assumes linear relationship between features and outcome |
| Naive Bayes | Simple, interpretable, fast | Assumes independence between features |
| Decision trees | Nonlinear relationships, interpretable | Prone to overfitting |
| Random forest | Nonlinear relationships, interpretable | Prone to overfitting |
| Support vector machines | Nonlinear relationships, interpretable | Prone to overfitting |
| Neural networks | Nonlinear relationships, fast | Prone to overfitting, difficult to interpret |

In the process of selecting an algorithm, simple, computationally efficient, and interpretable models are preferred over complex, computationally expensive, and uninterpretable models, all things being equal. Only if the performance of the simple model is not good enough should we move on to a more complex model.

With this end mind, we will start with a simple logistic regression model to see how well we can classify the texts in the training set with the features we have engineered. We will use the `logistic_reg()` function from the `parsnip` package to specify the logistic regression model. We then select the implementation engine (`glm` General Linear Model) and the mode of the model (`classification`). The implementation engine is the software that will be used to fit the model. The mode is the type of model, either classification or regression. The code is seen in Example 9.12.

Example 9.12.

```
# Create a model specification
cedel2_cls_spec <-
  logistic_reg() |>
  set_engine("glm") |>
  set_mode("classification")

# Preview
cedel2_cls_spec

> Logistic Regression Model Specification (classification)
>
> Computational engine: glm
```

In the `parsnip` package, model specifications are separate from model fitting. This allows us to specify the model once and then fit the model with different data sets. Furthermore, different algorithms will have different hyperparameters that can be tuned. The code in Example 9.12 uses the default hyperparameters for the logistic regression model.

Tip

You can retrieve the list of potential engines for a given model specification with the `show_engines()` function. For example,

```
show_engines(logistic_reg())
```

```
# A tibble: 7 × 2
  engine    mode
```

```

<chr>    <chr>
1 glm      classification
2 glmnet   classification
3 LiblineaR classification
4 spark    classification
5 keras    classification
6 stan    classification
7 brulee   classification

```

Now we will combine our recipe and model specification into a workflow object. The workflow object will allow us to train and evaluate the model on the training set. We will use the `workflow()` function from the `workflows` package to combine the recipe and model specification into a workflow object. The code is seen in Example 9.13.

Example 9.13.

```

# Create a workflow
cedel2_cls_wf <-
  workflow() |>
  add_recipe(cedel2_cls_rec) |>
  add_model(cedel2_cls_spec)

# Preview
cedel2_cls_wf

> == Workflow =====
> Preprocessor: Recipe
> Model: logistic_reg()
>
> -- Preprocessor -----
> 4 Recipe Steps
>
> * step_tokenize()
> * step_tokenfilter()
> * step_tf()
> * step_log()
>
> -- Model -----
> Logistic Regression Model Specification (classification)
>
> Computational engine: glm

```

The workflow contains all the preprocessing and the model-specific parameters we've selected. A workflow object at this stage in the analysis is not trained. It is a blueprint for a model. We can use the `fit()` function to apply the workflow to the training data to train the model and update our workflow object with the trained model. The `fit()` function takes a workflow and a data frame and returns an updated workflow object. The code is seen in Example 9.14.

Example 9.14.

```
# Fit the workflow
cedel2_cls_wf_fit <-
  fit(
    cedel2_cls_wf,
    data = cedel2_cls_train
  )
```

The workflow object now contains the trained model. But things did not go off without a hitch. We receive two warning messages when fitting the workflow to the training data:

```
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Warning messages are a sign that something may be amiss with the features we have chosen, the model we have selected, or both! Warning messages can sometimes be cryptic and steeped in the jargon of the algorithm. When in doubt it is best to consult the documentation of the algorithm and/ or seek help from the R community.

The first warning, the ‘algorithm did not converge’, means that given the data and the model, the algorithm could not estimate a stable solution. The reason or reasons for this can be varied, but it typically due to extreme outliers, collinearity, or a small sample size. Collinearity is when two or more features are highly correlated. Given we are using words as features, it is likely that there is collinearity in the data. We will need to address this.

The second warning is ‘fitted probabilities numerically 0 or 1 occurred’. This means that the model is overfitting the data, (i.e. there are one or more features that perfectly predict the outcome). This very well could be related to the first warning, as a small set of features may be perfectly predicting the outcome in a way that will likely not generalize to new data.

The upshot is that this model specification and/ or the feature engineering needs to be changed. So we step back in the workflow and make changes that will cascade downstream.

It turns out that a regularized logistic regression model often addresses the issues this model is experiencing. We will use the `glmnet` engine to fit a regularized logistic regression model. It

is a more complex approach, but it can often mitigate the issues of collinearity and overfitting by penalizing features that are highly correlated and by shrinking the coefficients of features that are not useful for predicting the outcome.

We will build a new model specification using the `logistic_reg()` function and the `glmnet` engine. This time we will use hyperparameters in our `logistic_reg()` call to specify the penalty we want to use and the strength of the penalty. The penalty we will use is the lasso (L1). We now need to decide what value to use for the strength of the penalty. We either experiment with different values one by one or we can implement a range of values in tandem and then select the best value. We will use the latter approach.

The `tune` package provides a number of functions for selecting, or ‘tuning’, hyperparameters. The first is the `tune()` function which we add as the argument of the hyperparameter we want to tune in the model specification, as seen in Example 9.15.

Example 9.15.

```
# Create a model specification for tuning
cedel2_cls_spec_tune <-
  logistic_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet") |>
  set_mode("classification")

# Create a workflow
cedel2_cls_wf_tune <-
  workflow() |>
  add_recipe(cedel2_cls_rec) |>
  add_model(cedel2_cls_spec_tune)
```

We now have a workflow that includes the `tune()` function as a placeholder for a range of values for the penalty hyperparameter. We use the `grid_regular()` function from the `dials` package to specify a grid of values for the penalty hyperparameter. Let’s choose a random set of 10 values, as seen in Example 9.16.

➊ Dive deeper

The hyperparameters `penalty` and `mixture` control which type(s) of regularization to apply and if there is a mixing of the types. In the model specification in Example 9.15, the `penalty` is set to be tuned and the `mixture` is set to 1, which means that only a lasso penalty will be applied.

See the documentation for the `parsnip::logistic_reg()` function for more information.

Example 9.16.

```

# Create a grid of values for the penalty hyperparameter
cedel2_cls_grid <-
  grid_regular(
    penalty(),
    levels = 10
  )

# Preview
cedel2_cls_grid

> # A tibble: 10 × 1
>       penalty
>   <dbl>
> 1 0.0000000001
> 2 0.0000000129
> 3 0.000000167
> 4 0.00000215
> 5 0.0000278
> 6 0.000359
> 7 0.000464
> 8 0.00599
> 9 0.0774
> 10 1

```

The 10 values chosen to be in the grid range from nearly zero to 1, where 0 indicates no penalty and 1 indicates a strong penalty.

Now to perform the tuning and arrive at an optimal value for `penalty` we need to create a tuning workflow. We do this by calling the `tune_grid()` function using our tuning model specification workflow, a resampling object, and our hyperparameter grid and returns a `tune_grid` object.

Now, a resampling object is not something we've seen yet. Resampling is a strategy that allows us to generate multiple training and testing sets from a single data set –in this case the training data we split at the outset. Each variation of the training and testing sets is called a fold. Which is why this type of resampling is called k-fold cross-validation. The `vfold_cv()` function from the `rsample` package takes a data frame and a number of folds and returns a `vfold_cv` object. We will use 10 folds and include the model specification and the hyperparameter grid in the `tune_grid()` function call. The code is seen in Example 9.17.

Example 9.17.

```

# Set seed for reproducibility
set.seed(123)

# Create a resampling object
cedel2_cls_vfold <-
  vfold_cv(
    cedel2_cls_train,
    v = 10
  )

# Tune the model
cedel2_cls_tune <-
  tune_grid(
    cedel2_cls_wf_tune,
    resamples = cedel2_cls_vfold,
    grid = cedel2_cls_grid
  )

# Preview
cedel2_cls_tune

```

```

> # Tuning results
> # 10-fold cross-validation
> # A tibble: 10 x 4
>   splits          id    .metrics      .notes
>   <list>         <chr> <list>        <list>
> 1 <split [2127/237]> Fold01 <tibble [20 x 5]> <tibble [0 x 3]>
> 2 <split [2127/237]> Fold02 <tibble [20 x 5]> <tibble [0 x 3]>
> 3 <split [2127/237]> Fold03 <tibble [20 x 5]> <tibble [0 x 3]>
> 4 <split [2127/237]> Fold04 <tibble [20 x 5]> <tibble [0 x 3]>
> 5 <split [2128/236]> Fold05 <tibble [20 x 5]> <tibble [0 x 3]>
> 6 <split [2128/236]> Fold06 <tibble [20 x 5]> <tibble [0 x 3]>
> 7 <split [2128/236]> Fold07 <tibble [20 x 5]> <tibble [0 x 3]>
> 8 <split [2128/236]> Fold08 <tibble [20 x 5]> <tibble [0 x 3]>
> 9 <split [2128/236]> Fold09 <tibble [20 x 5]> <tibble [0 x 3]>
> 10 <split [2128/236]> Fold10 <tibble [20 x 5]> <tibble [0 x 3]>

```

The `cedel2_cls_tune` object contains the results of the tuning for each fold. We can see the results of the tuning for each fold by calling the `collect_metrics()` function on the `cedel2_cls_tune` object, as seen in Example 9.18.

Example 9.18.

```
# Collect the results of the tuning
cedel2_cls_tune_metrics <-
  collect_metrics(cedel2_cls_tune)

# Visualize metrics
cedel2_cls_tune |> autoplot()
```

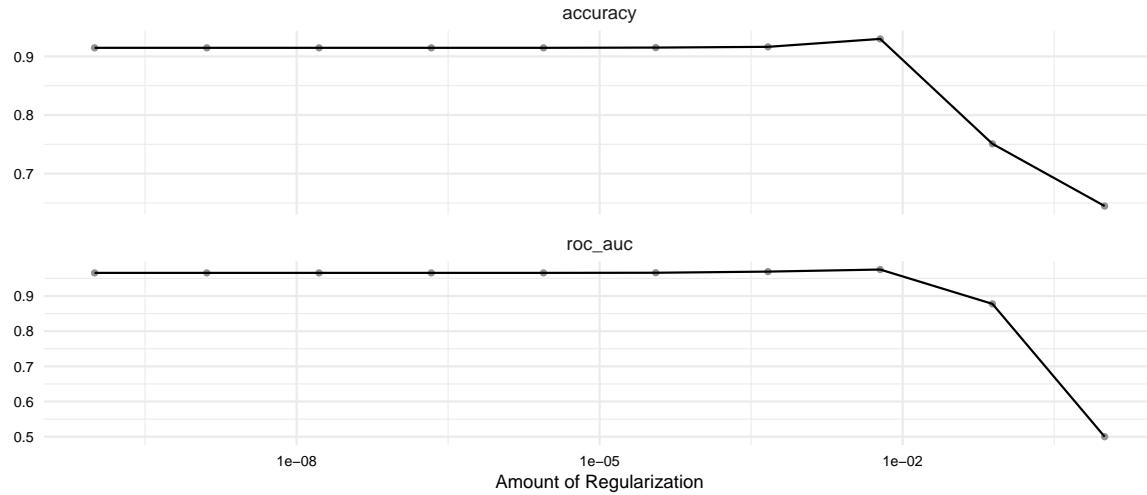


Figure 9.2: Metrics for each fold of the tuning process.

The most common metrics for model performance in classification are accuracy and the area under the receiver operating characteristic curve (ROC-AUC). Accuracy is the proportion of correct predictions. The ROC-AUC is a measure of the trade-off between sensitivity and specificity. Sensitivity is the proportion of true positives that are correctly identified. Specificity is the proportion of true negatives that are correctly identified. The ROC-AUC is a measure of how well the model can distinguish between the two classes.

In the plot of the metrics, we can see that the many of the penalty values performed similarly, with a drop off in performance at the higher values. Conveniently, the `show_best()` function from the `tune` package takes a `tune_grid` object and returns the best performing hyperparameter values. The code is seen in Example 9.19.

Example 9.19.

```

# Show the best performing hyperparameter value
cedel2_cls_tune |>
  show_best(metric = "roc_auc")

> # A tibble: 5 x 7
>   penalty .metric .estimator  mean   n std_err .config
>   <dbl> <chr> <chr>     <dbl> <int>  <dbl> <chr>
> 1 0.00599  roc_auc binary    0.975   10  0.00395 Preprocessor1_Model08
> 2 0.000464  roc_auc binary    0.969   10  0.00420 Preprocessor1_Model07
> 3 0.0000359  roc_auc binary    0.966   10  0.00411 Preprocessor1_Model06
> 4 0.0000000001  roc_auc binary    0.966   10  0.00415 Preprocessor1_Model01
> 5 0.00000000129  roc_auc binary    0.966   10  0.00415 Preprocessor1_Model02

```

We can make this selection programmatically by using the `select_best()` function. This function needs a metric to select by. We will use the ROC-AUC and select the best value for the penalty hyperparameter. The code is seen in Example 9.20.

Example 9.20.

```

# Select the best performing hyperparameter value
cedel2_cls_best <-
  select_best(cedel2_cls_tune, metric = "roc_auc")

# Preview
cedel2_cls_best

> # A tibble: 1 x 2
>   penalty .config
>   <dbl> <chr>
> 1 0.00599 Preprocessor1_Model08

```

All of that to tune a hyperparameter! Now we can update the model specification and workflow with the best performing hyperparameter value using the previous `cedel2_cls_wf_tune` workflow and the `finalize_workflow()` function. The `finalize_workflow()` function takes a workflow and the selected parameters and returns an updated `workflow` object, as seen in Example 9.21.

Example 9.21.

```

# Update model specification
cedel2_cls_wf_lasso <-
  cedel2_cls_wf_tune |>
  finalize_workflow(cedel2_cls_best)

cedel2_cls_wf_lasso

```

```

> == Workflow =====
> Preprocessor: Recipe
> Model: logistic_reg()
>
> -- Preprocessor -----
> 4 Recipe Steps
>
> * step_tokenize()
> * step_tokenfilter()
> * step_tf()
> * step_log()
>
> -- Model -----
> Logistic Regression Model Specification (classification)
>
> Main Arguments:
>   penalty = 0.00599484250318942
>   mixture = 1
>
> Computational engine: glmnet

```

Our model specification and the workflow are updated with the parameters.

Let's now return to fitting the workflow to the training set as we did before for the vanilla logistic regression model. As a reminder we are still working in step 5 of our workflow, interrogating the data. We identified and addressed potential issues in the model specification, leaving the feature selection the same.

We again fit the workflow to the training set, as seen in Example 9.22.

Example 9.22.

```

# Fit the workflow
cedel2_cls_wf_lasso_fit <-

```

```

fit(
  cedel2_cls_wf_lasso,
  data = cedel2_cls_train
)

```

No warnings, so that is a good sign! –Or at least a better sign than before.

Let's evaluate the performance of the model on the training data. The `predict()` function takes a trained model specification and a data frame and returns a data frame with the predicted outcome. We can join these predicted outcomes with the actual outcomes in the training data. The `metrics()` function takes a data frame with the actual and predicted outcomes and returns a data frame with the metrics for the model. The code is seen in Example 9.23.

Example 9.23.

```

# Evaluate workflow
cedel2_cls_lasso_fit_preds <-
  bind_cols(
    cedel2_cls_train,
    predict(cedel2_cls_wf_lasso_fit, cedel2_cls_train)
  )

# Calculate accuracy
cedel2_cls_lasso_fit_preds |>
  metrics(truth = outcome, estimate = .pred_class)

> # A tibble: 2 x 3
>   .metric  .estimator .estimate
>   <chr>    <chr>        <dbl>
> 1 accuracy  binary     0.970
> 2 kap       binary     0.934

```

Again, no warnings, so we have a functioning model. It also has a high accuracy, but we know that this is not the most reliable metric for the robustness of the model on new data. Similar to what we did to tune the hyperparameters, we can use cross-validation to gauge the variability of the model. The `fit_resamples()` function takes a workflow and a resampling object and returns metrics for each fold. The code is seen in Example 9.24.

Example 9.24.

```
# Cross-validate workflow
cedel2_cls_lasso_cv <-
  cedel2_cls_wf_lasso |>
  fit_resamples(
    resamples = cedel2_cls_vfold,
    control = control_resamples(save_pred = TRUE)
  )
```

We want to aggregate the metrics across the folds to get a sense of the variability of the model. The `collect_metrics()` function takes the results of a cross-validation and returns a data frame with the metrics.

Example 9.25.

```
# Collect metrics
cedel2_cls_lasso_cv |>
  collect_metrics()
```

```
> # A tibble: 2 x 6
>   .metric  .estimator  mean    n std_err .config
>   <chr>    <chr>     <dbl> <int>   <dbl> <chr>
> 1 accuracy  binary     0.930    10  0.00503 Preprocessor1_Model1
> 2 roc_auc   binary     0.975    10  0.00395 Preprocessor1_Model1
```

The accuracy has dropped, but is still very high. From these metrics it appears we have a good candidate model. In many cases, however, there may be further room for improvement. A good next step to in these cases is to evaluate the model errors and see if there are any patterns that can be addressed before evaluating the model on the test set.

For classification tasks, a good place to start is to visualize the confusion matrix to assess false negatives and false positives. The `conf_mat_resampled()` function takes a `fit_resamples` object and returns a table (`tidy = FALSE`) with the confusion matrix for the aggregated folds. We can pass this to the `autoplot()` function to plot as in Example 9.26.

Example 9.26.

```
# Plot confusion matrix
cedel2_cls_lasso_cv |>
  conf_mat_resampled(tidy = FALSE) |>
  autoplot(type = "heatmap")
```

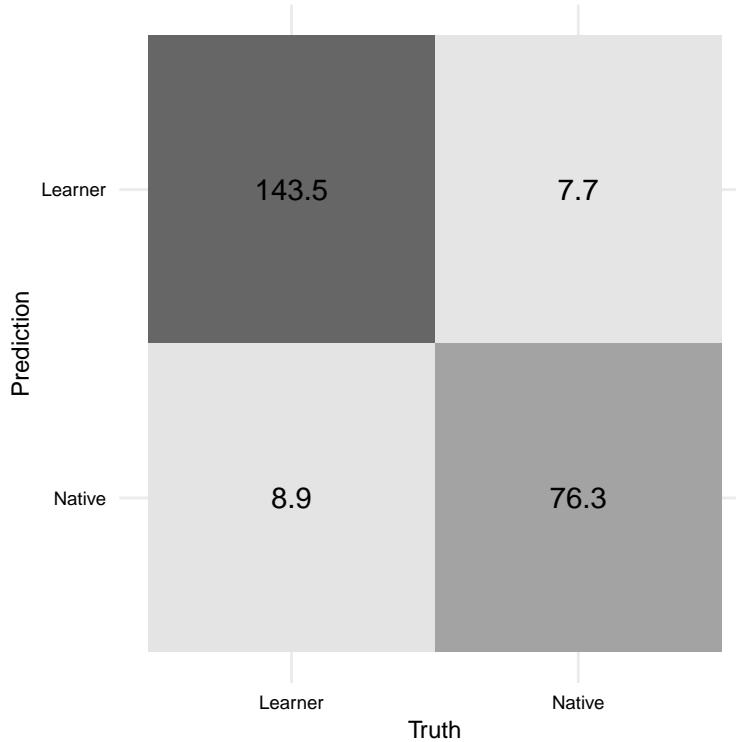


Figure 9.3: Confusion matrix for the aggregated folds of the cross-validation.

The top left to bottom right diagonal contains the true positives and true negatives. The top right to bottom left diagonal contains the false positives and false negatives. The convention is speak of one class being the positive class and the other class being the negative class. In our case, we will consider the positive class to be the ‘learner’ class and the negative class to be the ‘natives’ class.

- interpretation has changed, with log-transformation

We can see that there are more learners falsely predicted to be natives than the other way around. This may be due to the fact that there are simply more learners than natives in the data set or this could signal that there are some learners that are more similar to natives than other learners. Clearly this can’t be the entire explanation as the model is not perfect, even some natives are classified falsely as learners! But may be an interesting avenue for further exploration. Maybe these are learners that are more advanced or have a particular style of writing that is more similar to natives.

Another perspective often applied to evaluate a model is the ROC curve. The ROC curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) for different classification thresholds. To produce the ROC curve, we pass the resampled fit to the

`collect_predictions()` function. We then pass this result to `roc_curve()` function. But we want the results grouped by each fold, so we use `group_by(id)`. Finally, we can pass this to the `autoplot()` function to plot as in Example 9.27.

Example 9.27.

```
# Plot ROC curve
cedel2_cls_lasso_cv |>
  collect_predictions() |>
  group_by(id) |>
  roc_curve(truth = outcome, .pred_Native) |>
  autoplot()
```

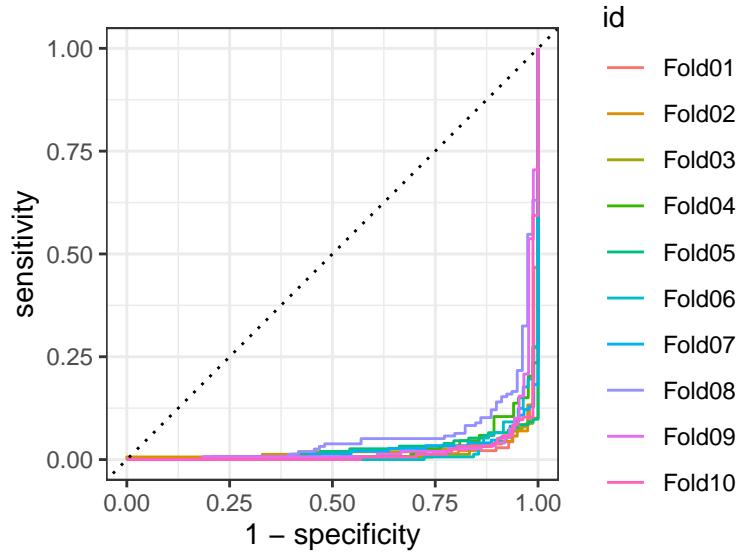


Figure 9.4: ROC curve for the aggregated folds of the cross-validation.

All signs point to a robust model.

We are now ready to move on to step 6, evaluating the model on the test set. We will use the `predict()` function to predict the outcome on the test set. The `metrics()` function takes a

data frame with the actual and predicted outcomes and returns a data frame with the metrics for the model. The code is seen in Example 9.28.

Example 9.28.

```
# Evaluate model on test set
cedel2_cls_lasso_fit_preds_test <-
  bind_cols(
    cedel2_cls_test,
    predict(cedel2_cls_wf_lasso_fit, cedel2_cls_test)
  )

# Calculate accuracy
cedel2_cls_lasso_fit_preds_test |>
  metrics(truth = outcome, estimate = .pred_class)

> # A tibble: 2 × 3
>   .metric   .estimator .estimate
>   <chr>     <chr>        <dbl>
> 1 accuracy  binary      0.924
> 2 kap       binary      0.834
```

The accuracy is as good as the training set, even a tad higher. This is a good sign that the model is robust as it performs well on both training and test sets. We can evaluate the confusion matrix on the test set as well. The code is seen in Example 9.29.

Example 9.29.

```
# Plot confusion matrix
cedel2_cls_lasso_fit_preds_test |>
  conf_mat(truth = outcome, estimate = .pred_class) |>
  autoplot(type = "heatmap")
```

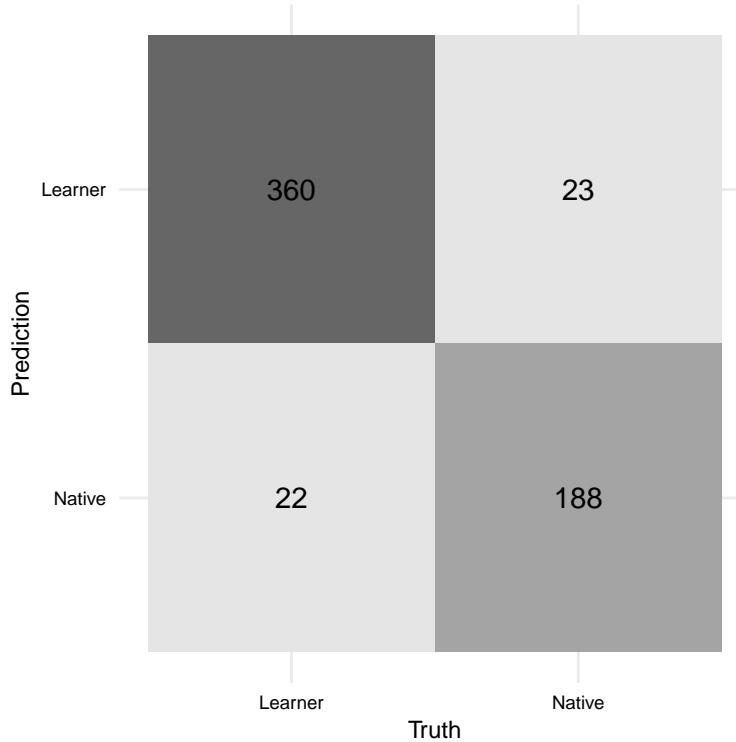


Figure 9.5: Confusion matrix for the test set.

On the test set the false instances look similar to the distribution on the training set, with the exception that learners are more likely to be falsely predicted to be natives than the other way around.

We can inspect the errors on the test set by filtering the data frame to only include the false instances. I will then select the columns with the actual outcome, the predicted outcome, the proficiency level, and the text and separate the predicted outcome to inspect them separately, as seen in Example 9.30.

Example 9.30.

```
# Inspect errors
cedel2_cls_lasso_fit_preds_test |>
  filter(outcome != .pred_class) |>
  select(outcome, .pred_class, proficiency, text)
```

```
> # A tibble: 45 x 4
```

```

>   outcome .pred_class proficiency      text
>   <fct>   <fct>      <chr>          <chr>
> 1 Learner Native      Upper beginner   "Un dia, El niño estaba durmiendo cua-
> 2 Learner Native      Upper beginner   "Mi opinión de la educación de biling-
> 3 Learner Native      Lower intermediate "La película "Solas" contiene muchos ~
> 4 Learner Native      Upper intermediate "A la semana pasada, yo vi la películ-
> 5 Learner Native      Lower advanced    "El año pasado fui a EEUU para 3 sema-
> 6 Learner Native      Lower advanced    "Es una pregunta que lleva mucho tiem-
> 7 Learner Native      Lower advanced    "En el video, podemos ver que el niño-
> 8 Learner Native      Lower advanced    "Actualmente estoy viviendo y trabaja-
> 9 Learner Native      Lower advanced    "Podría escribir un libro de mis pla-
> 10 Learner Native     Lower advanced    "Este mes pasado, yo viaje a la bella-
> # i 35 more rows

```

```

# Inspect learners falsely predicted to be natives
cedel2_cls_lasso_fit_preds_test |>
  filter(outcome == "Learner", .pred_class == "Native") |>
  select(outcome, .pred_class, proficiency, text) |>
  count(proficiency, sort = TRUE)

```

```

> # A tibble: 5 x 2
>   proficiency      n
>   <chr>          <int>
> 1 Lower advanced   10
> 2 Upper advanced   8
> 3 Upper beginner   2
> 4 Lower intermediate 1
> 5 Upper intermediate 1

```

- Majority of misclassified learners are advanced, which could be expected as they are more similar to natives. There are some beginners that are misclassified as natives, which is interesting, and not expected. But all models are wrong, but some are useful –George Box.
- Still an open question as to why some natives are classified as learners.

We can inspect the estimates for the features in the model to gain some insight into what features are most important for predicting the outcomes. The `extract_fit_parsnip()` function takes a trained model specification and returns a data frame with the estimated coefficients for each feature. The code is seen in Example 9.31.

Example 9.31.

```

# Extract estimates
cedel2_cls_wf_lasso_fit |>
  extract_fit_parsnip() |>
  tidy()

> # A tibble: 1,001 x 3
>   term            estimate  penalty
>   <chr>          <dbl>    <dbl>
> 1 (Intercept)    -1.44    0.00599
> 2 tf_text_10      0        0.00599
> 3 tf_text_2       0        0.00599
> 4 tf_text_3       0        0.00599
> 5 tf_text_4       0        0.00599
> 6 tf_text_5       0        0.00599
> 7 tf_text_a       0.332   0.00599
> 8 tf_text_abandonado 0        0.00599
> 9 tf_text_abuela  0        0.00599
> 10 tf_text_abuelos 0        0.00599
> # i 991 more rows

```

The estimates are the log odds of the outcome. In a binary classification task, the log odds of the outcome is the log of the probability of the outcome divided by the probability of the other outcome. In our case, the reference outcome is “Learner”, so negative log-odds indicate that the feature is associated with the “Learner” outcome and positive log-odds indicate that the feature is associated with the “Native” outcome.

The estimates are in log-odds, so we need to exponentiate them to get the odds. The odds are the probability of the outcome divided by the probability of the other outcome. The probability of the outcome is the odds divided by the odds plus one. The code is seen in Example 9.32.

Example 9.32.

```

# Calculate probability
cedel2_cls_wf_lasso_fit |>
  extract_fit_parsnip() |>
  tidy() |>
  mutate(probability = exp(estimate) / (exp(estimate) + 1))

> # A tibble: 1,001 x 4

```

```

>   term          estimate  penalty  probability
>   <chr>        <dbl>    <dbl>    <dbl>
> 1 (Intercept) -1.44    0.00599  0.191
> 2 tf_text_10    0        0.00599  0.5
> 3 tf_text_2     0        0.00599  0.5
> 4 tf_text_3     0        0.00599  0.5
> 5 tf_text_4     0        0.00599  0.5
> 6 tf_text_5     0        0.00599  0.5
> 7 tf_text_a     0.332   0.00599  0.582
> 8 tf_text_abandonado 0        0.00599  0.5
> 9 tf_text_abuela 0        0.00599  0.5
> 10 tf_text_abuelos 0       0.00599  0.5
> # i 991 more rows

```

So just looking at the snippet of the features returned from Example 9.32, we can see that the features ‘a’ and ‘abandonado’ are slightly associated with the “Native” outcome and the other features are neutral (**probability** = 0.5).

A quick way to extract the most important features for predicting the each outcome is to use the `vi()` function from the `vip` package. It takes a trained model specification and returns a data frame with the most important features. The code is seen in Example 9.33.

Example 9.33.

```

# Load package
library(vip)

conflicted::conflicts_prefer(vip::vi)

# Extract important features
var_importance_df <-
  cedel2_cls_wf_lasso_fit |>
  vi()

# Preview
var_importance_df

> # A tibble: 1,000 x 3
>   Variable        Importance Sign
>   <chr>          <dbl> <chr>
> 1 tf_text_ahí     8.83  POS
> 2 tf_text_sorpresa 8.43  POS

```

```

> 3 tf_text_juan           7.91 POS
> 4 tf_text_todavia        7.45 NEG
> 5 tf_text_encuentran     6.72 NEG
> 6 tf_text_diferencia      6.64 POS
> 7 tf_text_eeuu            6.34 NEG
> 8 tf_text_última          6.26 POS
> 9 tf_text_favorito        6.25 NEG
> 10 tf_text_único          6.07 POS
> # i 990 more rows

```

The `Variable` column contains each feature (with the feature type and corresponding variable `tf_text_`), `Importance` provides the absolute log-odds value, and the `Sign` column indicates whether the feature is associated with the “NEG” (“Learner”) or the “POS” (“Native”) outcome. We can recode the `Variable` and `Sign` columns to make them more interpretable and exponentiate the log-odds to get probabilities and then plot them using `ggplot()`, as in Example 9.34.

Example 9.34.

```

# Recode variable and sign
var_importance_df <-
  var_importance_df |>
  mutate(
    Feature = str_remove(Variable, "tf_text_"),
    Outcome = case_when(Sign == "NEG" ~ "Learner", Sign == "POS" ~ "Native"),
    Importance = exp(Importance) / (exp(Importance) + 1)
  ) |>
  select(Outcome, Feature, Importance)

# Plot
var_importance_df |>
  slice_max(Importance, n = 50) |>
  ggplot(aes(x = reorder(Feature, Importance), y = Importance)) +
  geom_point() +
  coord_flip() +
  facet_wrap(~ Outcome, scales = "free_y") +
  labs(x = NULL, y = "Importance", fill = NULL) +
  theme_minimal()

```

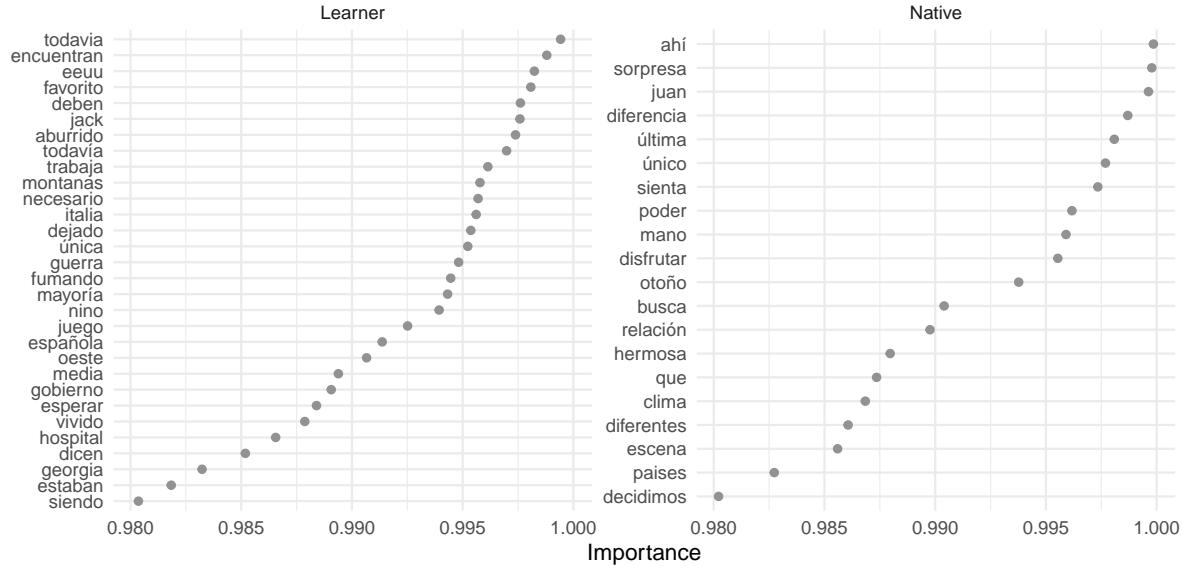


Figure 9.6: Most important features for predicting the outcome.

We can inspect Figure 9.6, and qualitatively assess what these features may be telling us about the differences between the learners and the natives.

In this section we've build a text classifier using a regularized logistic regression model. We've tuned the hyperparameters to arrive at a robust model that performs well on both the training and test sets. We've also evaluated the model errors and inspected the most important features for predicting the outcome.

The `tidymodels` package provides a framework for building and evaluating supervised machine learning models that is modular in nature. This means, that we can quickly and easily change the model specification, the features, feature engineering, and the hyperparameters to arrive at a robust model. If the model is not robust, we can change models in the model specification. If the features are not robust, we can change the recipe. If the model is overfitting, we can tune the hyperparameters.

9.2.2 Text regression

We will now turn our attention to the second task in this section, text regression. In this task, we will use the same original dataset as in the classification task, but we will predict the placement score based on the learner writing samples. Let's start by extracting the observations (only learners) and the relevant variables from the original data set. The code is seen in Example 9.35.

Example 9.35.

```
# Extract observations and relevant variables
cedel2_reg <-
  cedel2_df |>
  filter(proficiency != "Native") |>
  select(outcome = placement_score, proficiency, text)

# Preview
cedel2_reg |> glimpse()
```

```
> Rows: 1,906
> Columns: 3
> $ outcome    <dbl> 14.0, 16.3, 16.3, 18.6, 18.6, 18.6, 20.9, 20.9, 20.9, 20.9~
> $ proficiency <chr> "Lower beginner", "Lower beginner", "Lower beginner", "Low~
> $ text        <chr> "Yo vivo es Alanta, Georgia. Atlanta es muy grande ciudad.~
```

In this task, our outcome variable is numeric so we do not need, or want, to convert it to a factor. Our predictor variable `text` is the same as before. We have already weighed the options for feature engineering and decided to use the term frequency method (raw counts) for the top 1,000. Since we are setting up the same recipe as before, essentially, we can use the same code as before.

Let's first move to step 2, initial split. We will use the `initial_split()` function again, but this time we will not need to stratify the data as we are not predicting a categorical variable. The code is seen in Example 9.36.

Example 9.36.

```
# Set seed for reproducibility
set.seed(123)

# Split data
cedel2_reg_split <-
  initial_split(cedel2_reg, prop = 0.8)

# Training set
cedel2_reg_train <-
  training(cedel2_reg_split)

# Test set
```

```
cedel2_reg_test <-
  testing(cedel2_reg_split)
```

The training set has 1524 observations and the test set has 382 observations.

Now we can create the recipe to set up the variable relations, select the features, and engineer the features. The code is seen in Example 9.37.

Example 9.37.

```
# Create a recipe
cedel2_reg_rec <-
  recipe(outcome ~ text, data = cedel2_reg_train) |>
  step_tokenize(text) |>
  step_tokenfilter(text, max_tokens = 1000) |>
  step_tf(text) |>
  step_log(all_predictors(), offset = 1)

# Preview
cedel2_reg_rec
```

At this point we would inspect our recipe to make sure that it looks as expected and gauge the number of features. But since are using the same recipe as before, we can skip this step.

We can now proceed to interrogate the data. As before we will want to start with a simple model and then build up to more complex models. Let's consider some common algorithms for regression tasks in Table 9.5.

Table 9.5: Regression algorithms

| Algorithm | Strengths | Shortcomings |
|-------------------|--|--|
| Linear regression | Simple, interpretable, fast | Assumes linear relationship between features and outcome |
| Decision trees | Nonlinear relationships, interpretable | Prone to overfitting |
| Random forest | Nonlinear relationships, interpretable | Prone to overfitting |
| Neural networks | Nonlinear relationships, fast | Prone to overfitting, difficult to interpret |

Let's start with a with a linear regression model in mind for our model specification. But let's also consider what we learned in our first attempt to build a logistic regression model using word frequencies. We learned that the model was overfitting the training data and we needed to use a regularized model to reduce the variance of the model. So let's start with a regularized linear regression model.

The `linear_reg()` function, just as the `logistic_reg()` function, provides arguments for the regularization hyperparameters when the `glmnet` computational engine is used. Let's tune the `penalty` hyperparameter in the same way as before. The code for the process is seen in Example 9.38.

Example 9.38.

```
# Create model specification
cedel2_reg_spec_lasso <-
  linear_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet") |>
  set_mode("regression")

# Create workflow
cedel2_reg_wf_lasso <-
  workflow() |>
  add_recipe(cedel2_reg_rec) |>
  add_model(cedel2_reg_spec_lasso)

# Create tuning grid
cedel2_reg_grid <-
  grid_regular(penalty(), levels = 10)

# Set seed for reproducibility
set.seed(123)

# Create tuning workflow
cedel2_reg_tune <-
  tune_grid(
    cedel2_reg_wf_lasso,
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    grid = cedel2_reg_grid,
    control = control_resamples(save_pred = TRUE)
  )

# Select best parameter
```

```

chosen_penalty <-
  cedel2_reg_tune |>
  select_best(metric = "rmse")

# Update workflow
cedel2_reg_final_wf_lasso <-
  cedel2_reg_wf_lasso |>
  finalize_workflow(chosen_penalty)

cedel2_reg_final_wf_lasso

```

```

> == Workflow =====
> Preprocessor: Recipe
> Model: linear_reg()
>
> -- Preprocessor -----
> 4 Recipe Steps
>
> * step_tokenize()
> * step_tokenfilter()
> * step_tf()
> * step_log()
>
> -- Model -----
> Linear Regression Model Specification (regression)
>
> Main Arguments:
>   penalty = 1
>   mixture = 1
>
> Computational engine: glmnet

```

Now we fit this model to the training data and evaluate the performance using cross-validation. The code is seen in Example 9.39.

Example 9.39.

```

# Cross-validated workflow
cedel2_reg_cv_lasso <-
  cedel2_reg_final_wf_lasso |>

```

```

fit_resamples(
  resamples = vfold_cv(cedel2_reg_train, v = 10),
  control = control_resamples(save_pred = TRUE)
)

# Collect metrics
cedel2_reg_cv_lasso |>
  collect_metrics()

> # A tibble: 2 x 6
>   .metric .estimator  mean     n std_err .config
>   <chr>   <chr>     <dbl> <int>   <dbl> <chr>
> 1 rmse    standard    13.3     10   0.348 Preprocessor1_Model1
> 2 rsq     standard     0.659     10   0.0222 Preprocessor1_Model1

```

Now, the RMSE estimate is 13.3. RMSE is expressed in the same units as the outcome variable. In this case, the outcome variable is the placement test score percent. So the RMSE is 13.3 percentage points. The R^2 (rsq) is 0.659. This means that the model explains 66% of the variance in the outcome variable. Taken together, this isn't the best model.

But how good or bad is it? This is where we can use the null model to compare the model to. The null model is a model that predicts the mean of the outcome variable. We can use the `null_model()` function to create a null model and submit it to cross-validation. The code is seen in Example 9.40.

Example 9.40.

```

# Create null model
null_model <-
  null_model() |>
  set_engine("parsnip") |>
  set_mode("regression")

# Cross-validate null model
null_cv <-
  workflow() |>
  add_recipe(cedel2_reg_rec) |>
  add_model(null_model) |>
  fit_resamples(
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    metrics = metric_set(rmse)

```

```

)
# Collect metrics
null_cv |>
  collect_metrics()

```

```

> # A tibble: 1 × 6
>   .metric .estimator  mean     n std_err .config
>   <chr>   <chr>     <dbl> <int>   <dbl> <chr>
> 1 rmse    standard    22.6     10    0.206 Preprocessor1_Model1

```

Our the word features perform better than the null model which means that it is picking up on some signal in the data.

Let's visualize the distribution of the predictions and the errors from our word features model to see if there are any patterns of interest. We can use the `collect_predictions()` function to extract the predictions of the cross-validation and plot the true outcome agains the predicted outcome using `ggplot()`, as in Example 9.58.

Example 9.41.

```

# Visualize predictions
cedel2_reg_cv_lasso |>
  collect_predictions() |>
  ggplot(aes(outcome, .pred, shape = id)) +
  geom_point(alpha = 0.5, position = position_jitter(width = 0.5)) +
  geom_smooth(method = "lm", se = FALSE, linewidth = 0.5) + # trend for each fold
  labs(
    x = "Truth",
    y = "Predicted score",
    shape = "Fold"
)

```

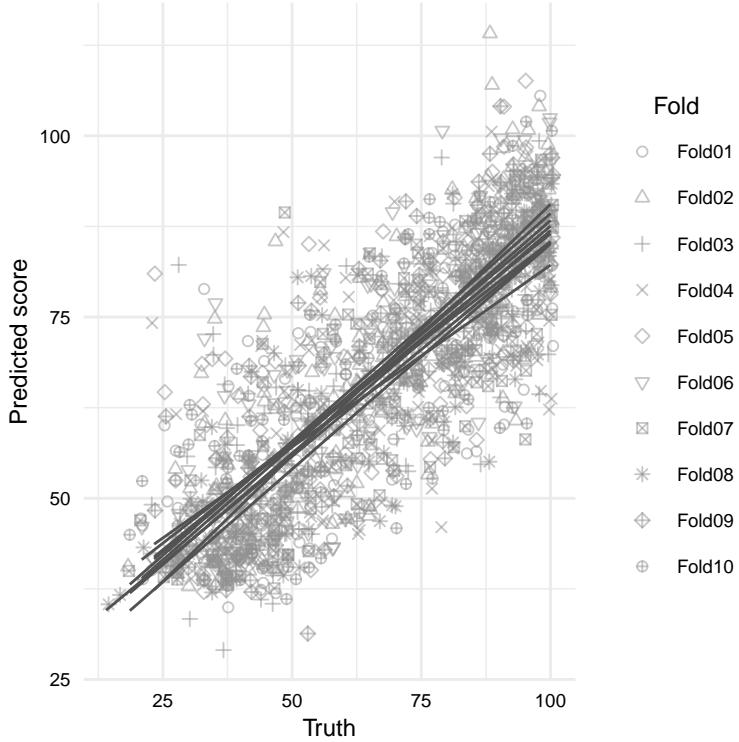


Figure 9.7: Distribution of the RMSE for the cross-validated linear regression model.

From this plot, we see data points for each predicted and truth value pair for each of the ten folds. There is a trend line for each fold which shows the linear relationship between the predicted and truth values for each fold. The trend lines are more similar than different, which is a good sign that the model is not wildly overfitting the training data. Looking closer, however, we can see the errors. Some are noticeably distant from the linear trend lines, *i.e.* outliers, in particular for test scores in the higher and lower ranges. There also seems to be a pattern in that the model predicts lower scores for higher scores and higher scores for lower scores. We can see this by the overplotted points in the higher and lower ranges.

If the R^2 value is in the ballpark, this means that somewhere around 40% of the variation is not explained by the frequency of the top 1,000 words. This is not surprising, as there are many other factors that contribute to the proficiency level of a text.

We have a model that is performing better than the null model, but it is not performing well enough to be very useful. We will need to update the model specification and/ or the features to try to improve the model fit. Let's start with the model. There are many different model specifications we could try, but we will likely need to use a more complex model specification to capture the complexity that we observed in the errors from the previous model.

Let's try a decision tree model. **Decision trees** are non-linear models that are able to model non-linear relationships and interactions between the features and the outcome and tend to be less influenced by outliers. These are all desirable characteristics. Decision trees, however, can be prone to overfitting.

Along the spectrum of model complexity, decision trees are more complex than linear regression models, but less complex than other models such as neural networks. Furthermore, decision trees are interpretable, which is a nice feature for an exploratory-oriented analysis.

To implement a new model in `tidymodels`, we need to create a new model specification and a new workflow. We will use the `decision_tree()` function from the `parsnip` package to create the model specification. The `decision_tree()` function takes no arguments and returns a `decision_tree` object. We create the new model specification in Example 9.42.

Example 9.42.

```
# Create model specification
cedel2_reg_spec_tree <-
  decision_tree() |>
  set_engine("rpart") |>
  set_mode("regression")

cedel2_reg_spec_tree

> Decision Tree Model Specification (regression)
>
> Computational engine: rpart
```

We now have a new model specification. We can now create a new workflow. We will use the same recipe as before, but we will change the model specification to `cedel2_prof_spec_tree`. We add this to a new workflow and fit the workflow to the training data. The code is seen in Example 9.43.

Example 9.43.

```
# Create workflow
cedel2_reg_wf_tree <-
  workflow() |>
  add_recipe(cedel2_reg_rec) |>
  add_model(cedel2_reg_spec_tree)

# Set seed for reproducibility
```

```

set.seed(123)

# Cross-validated workflow
cedel2_reg_cv_tree <-
  cedel2_reg_wf_tree |>
  fit_resamples(
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    control = control_resamples(save_pred = TRUE)
  )

```

We can now evaluate the performance of the model using cross-validation. The code is seen in Example 9.44.

Example 9.44.

```

# Collect metrics
cedel2_reg_cv_tree |>
  collect_metrics()

> # A tibble: 2 × 6
>   .metric .estimator   mean     n std_err .config
>   <chr>   <chr>     <dbl> <int>   <dbl> <chr>
> 1 rmse    standard    15.1     10   0.404 Preprocessor1_Model1
> 2 rsq     standard    0.554     10   0.0215 Preprocessor1_Model1

```

The metrics for the vanilla decision tree are similar to the regularized linear regression model. The RSME is 15.1 and the R^2 is 0.554. Yet, if we compare the standard error between the two models, we can see that the decision tree model has a larger standard error. This means that the decision tree model is more prone to overfitting the training data.

To minimize overfitting, we can tune hyperparameters of the model. Regularization is one way to reduce overfitting, as we saw with the regularized linear regression model. Another way to reduce overfitting is to reduce the complexity of the model. This can be done by reducing the number of features or by reducing the number of splits in the decision tree, known as **pruning**.

Another approach is to implement a random forest model. A **random forest** is an ensemble model that combines multiple decision trees to make a prediction. A random forest is a type of ensemble model which means that it combines multiple models to make a prediction. In addition to multiple decision trees, random forests also perform random feature selection. This

helps to reduce the correlation between the decision trees and thus reduces the variance of the model.

Let's try a random forest model. We will use the `rand_forest()` function from the `parsnip` package to create the model specification. The `rand_forest()` function takes no arguments and returns a `rand_forest` object. We will select the `ranger` engine and add the `importance` argument to ensure that we can extract feature importance if this model proves to be useful. We create the new model specification in Example 9.45.

Example 9.45.

```
# Create model specification
cedel2_reg_spec_rf <-
  rand_forest() |>
  set_engine("ranger", importance = "impurity") |>
  set_mode("regression")

cedel2_reg_spec_rf
```

```
> Random Forest Model Specification (regression)
>
> Engine-Specific Arguments:
>   importance = impurity
>
> Computational engine: ranger
```

We can now update the workflow to use the new model specification. The code is seen in Example 9.46.

Example 9.46.

```
# Update workflow
cedel2_reg_wf_rf <-
  cedel2_reg_wf_tree |>
  update_model(cedel2_reg_spec_rf)

# Set seed for reproducibility
set.seed(123)

# Cross-validated workflow
cedel2_reg_cv_rf <-
```

```

cedel2_reg_wf_rf |>
  fit_resamples(
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    control = control_resamples(save_pred = TRUE)
  )

```

We can now evaluate the performance of the model using cross-validation. The code is seen in Example 9.47.

Example 9.47.

```

# Collect metrics
cedel2_reg_cv_rf |>
  collect_metrics()

> # A tibble: 2 x 6
>   .metric .estimator   mean     n std_err .config
>   <chr>   <chr>     <dbl> <int>  <dbl> <chr>
> 1 rmse    standard    13.1     10  0.305 Preprocessor1_Model1
> 2 rsq     standard    0.680     10  0.0133 Preprocessor1_Model1

```

The random forest model performs better than the decision tree model and the regularized linear regression model. The RSME is 13.1 and the R^2 is 0.68. We also see that the standard error is the lowest of the models we have tried so far. This means that the random forest model is less prone to overfitting the training data.

Before we settle on this model, let's try one more model, a support vector machine (SVM). A **support vector machine** is a supervised machine learning model that can be used for both classification and regression. It is a non-parametric method, which means that it does not make any assumptions about the underlying distribution of the data. It is also a method which can be better suited for high-dimensional data where many values are zero, that is, sparse data, which is often the case with text data.

Let's again specify the model. We will use the `svm_linear()` function from the `parsnip` package to create the model specification and we will use the `LiblineaR` computational engine. We create the new model specification in Example 9.48.

Example 9.48.

```
# Create model specification
cedel2_reg_spec_svm <-
  svm_linear() |>
  set_engine("LiblineaR") |>
  set_mode("regression")

cedel2_reg_spec_svm
```

```
> Linear Support Vector Machine Model Specification (regression)
>
> Computational engine: LiblineaR
```

Now let's update the workflow to use the new model specification. The code is seen in Example 9.49.

Example 9.49.

```
# Update workflow
cedel2_reg_wf_svm <-
  cedel2_reg_wf_rf |>
  update_model(cedel2_reg_spec_svm)

# Set seed for reproducibility
set.seed(123)

# Cross-validated workflow
cedel2_reg_cv_svm <-
  cedel2_reg_wf_svm |>
  fit_resamples(
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    control = control_resamples(save_pred = TRUE)
  )
```

Let's evaluate the cross-validation performance of the model. The code is seen in Example 9.50.

Example 9.50.

```

# Collect metrics
cedel2_reg_cv_svm |>
  collect_metrics()

> # A tibble: 2 x 6
>   .metric .estimator  mean     n std_err .config
>   <chr>   <chr>     <dbl> <int>   <dbl> <chr>
> 1 rmse    standard    14.7     10   0.353 Preprocessor1_Model1
> 2 rsq     standard     0.625     10   0.0152 Preprocessor1_Model1

```

This SVR is not performing better than the linear regression and random forest models and the standard error is not particularly low. So we will not pursue this model further.

So in summary, we've tried four different model specifications. The regularized linear regression model, the decision tree model, the random forest model, and the support vector machine model. The random forest model performed the best. But there stands to be improvement, but it looks as if we may need to update the features.

In the recipe we have used until this point we have limited the word features to 1,000 and used the raw counts of the words. The rationale for this was that we wanted to limit the number of features to reduce the complexity of the model while still capturing the most important features. But we may have limited the features too much, at least now that we are comparing features between the same population (learners). A potentially more informative feature would be the TF-IDF score. This score is a measure of how important a word is to a document in a collection or corpus. Specifically, it is the product of the term frequency and the inverse document frequency. The more dispersed a feature is across the corpus, the lower the TF-IDF score. The more concentrated a feature is across the corpus, the higher the TF-IDF score, and the more informative the feature is for distinguishing between documents.

So we can use the `step_tfidf()` function to update the recipe. The code is seen in Example 9.51.

Example 9.51.

```

# Create a recipe
cedel2_reg_rec <-
  recipe(outcome ~ text, data = cedel2_reg_train) |>
  step_tokenize(text) |>
  step_tokenfilter(text, max_tokens = 1000) |>
  step_tfidf(text)

```

```
# Preview
cedel2_reg_rec
```

👉 Tip

Note that the log-transformation is not necessary when using TF-IDF scores as normalization is built into the TF-IDF calculation.

Another change we can explore is to increase the number of features. But how many features should we use? We can turn to the `tune()` function to help us answer this question. Let's add the `tune()` function to the recipe and tune the `max_tokens` hyperparameter. The code is seen in Example 9.52.

Example 9.52.

```
# Create a recipe
cedel2_reg_rec <-
  recipe(outcome ~ text, data = cedel2_reg_train) |>
  step_tokenize(text) |>
  step_tokenfilter(text, max_tokens = tune()) |>
  step_tfidf(text)

# Preview
cedel2_reg_rec
```

Let's tune the `max_tokens` with the random forest model specification that we created earlier. We will create a tuning workflow, which will tune the `max_tokens` hyperparameter on the training data using cross-validation. The code is seen in Example 9.53.

Example 9.53.

```
# Create tuning workflow
cedel2_tune_wf <-
  workflow() |>
  add_recipe(cedel2_reg_rec) |>
  add_model(cedel2_reg_spec_rf)

cedel2_tune_wf
```

```
> == Workflow =====
```

```

> Preprocessor: Recipe
> Model: rand_forest()
>
> -- Preprocessor -----
> 3 Recipe Steps
>
> * step_tokenize()
> * step_tokenfilter()
> * step_tfidf()
>
> -- Model -----
> Random Forest Model Specification (regression)
>
> Engine-Specific Arguments:
>   importance = impurity
>
> Computational engine: ranger

```

The `tune_grid()` function takes a workflow, a resampling method, a tuning grid, and a set of metrics. The tuning grid is a set of hyperparameters and their values. The `tune_grid()` function will tune the hyperparameters on the training data using cross-validation. The code is seen in Example 9.54. This is a computationally intensive process, as it will train a 10 folds, for each hyperparameter value. Each fold in the random forest is 500 trees, so this will train 5,000 trees for each hyperparameter value! So that is 30,000 trees in total. This code will take some time to run.

Example 9.54.

```

# Create tuning grid
cedel2_tune_grid <-
  grid_regular(max_tokens(range = c(500, 3500)), levels = 5)

cedel2_tune_grid

> # A tibble: 5 x 1
>   max_tokens
>   <int>
> 1      500
> 2     1250
> 3     2000
> 4     2750
> 5     3500

```

```

# Set seed for reproducibility
set.seed(123)

# Create tuning workflow
cedel2_tune <-
  tune_grid(
    cedel2_tune_wf,
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    grid = cedel2_tune_grid,
    metrics = metric_set(rmse, rsq),
    control = control_resamples(save_pred = TRUE)
  )

cedel2_tune

```

```

> # Tuning results
> # 10-fold cross-validation
> # A tibble: 10 × 5
>   splits          id   .metrics      .notes      .predictions
>   <list>         <chr> <list>        <list>        <list>
> 1 <split [1371/153]> Fold01 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 2 <split [1371/153]> Fold02 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 3 <split [1371/153]> Fold03 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 4 <split [1371/153]> Fold04 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 5 <split [1372/152]> Fold05 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 6 <split [1372/152]> Fold06 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 7 <split [1372/152]> Fold07 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 8 <split [1372/152]> Fold08 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 9 <split [1372/152]> Fold09 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>
> 10 <split [1372/152]> Fold10 <tibble [10 × 5]> <tibble [0 × 3]> <tibble>

```

Just as we did for the `penalty` hyperparameter, we can visualize the performance of the model as a function of the `max_tokens` hyperparameter. The code is seen in Example 9.55.

Example 9.55.

```

# Visualize tuning results
cedel2_tune |>
  collect_metrics() |>
  ggplot(aes(max_tokens, mean)) +

```

```

geom_point() +
geom_line() +
scale_x_continuous(
  breaks = scales::pretty_breaks(n = 10),
  labels = scales::comma
) +
facet_wrap(~ .metric, scales = "free_y", ncol = 1) +
labs(
  x = "Max tokens"
)

```

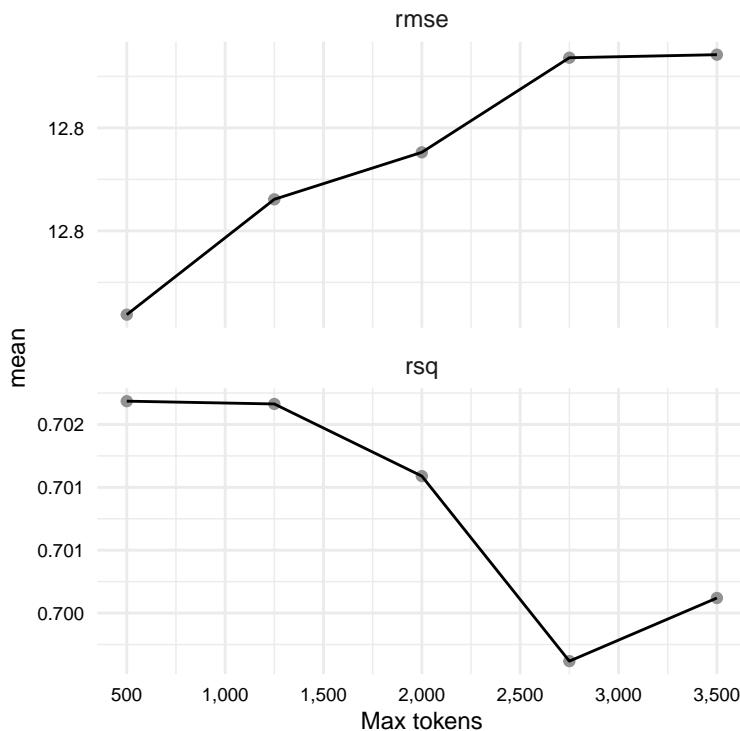


Figure 9.8: Performance of the random forest model as a function of the `max_tokens` hyperparameter.

We added a range of maximum token values to the tuning grid to see if there is a point of diminishing returns. We can see that the performance of the model actually appears to improve, not with more tokens, but instead with less. Our lowest value of 500 tokens appears to perform the best. This is interesting, as it suggests that there are a small number of words that are most informative for predicting the placement test score. It also suggests that

individual words, on the whole, are not very informative for predicting the placement test score.

We can now select the best hyperparameter value. The code is seen in Example 9.56.

Example 9.56.

```
# Select best parameter
chosen_max_tokens <-
  cedel2_tune |>
  select_best(metric = "rmse")

# Update workflow
cedel2_reg_final_rf <-
  cedel2_tune_wf |>
  finalize_workflow(chosen_max_tokens)

cedel2_reg_final_rf

> == Workflow =====
> Preprocessor: Recipe
> Model: rand_forest()
>
> -- Preprocessor -----
> 3 Recipe Steps
>
> * step_tokenize()
> * step_tokenfilter()
> * step_tfidf()
>
> -- Model -----
> Random Forest Model Specification (regression)
>
> Engine-Specific Arguments:
>   importance = impurity
>
> Computational engine: ranger
```

We can now fit the model to the training data and evaluate the performance using cross-validation. The code is seen in Example 9.57.

Example 9.57.

```

# Cross-validated workflow
cedel2_reg_cv_rf <-
  cedel2_reg_final_rf |>
  fit_resamples(
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    control = control_resamples(save_pred = TRUE)
  )

# Collect metrics
cedel2_reg_cv_rf |>
  collect_metrics()

```

```

> # A tibble: 2 x 6
>   .metric .estimator  mean     n std_err .config
>   <chr>   <chr>     <dbl> <int>   <dbl> <chr>
> 1 rmse    standard    12.8     10   0.221 Preprocessor1_Model1
> 2 rsq     standard     0.695    10   0.0137 Preprocessor1_Model1

```

We've improved the model, but not very much. We can see that the RMSE is 13.1 and the R^2 is 0.68. The standard error is NA. This is the lowest standard error we have seen so far, but it is still not as low as we would like.

Let's now visualize the distribution of the predictions and the errors from our word features model to see if there are any patterns of interest. We can use the `collect_predictions()` function to extract the predictions of the cross-validation and plot the true outcome against the predicted outcome using `ggplot()`, as in Example 9.58.

Example 9.58.

```

# Visualize predictions
cedel2_reg_cv_rf |>
  collect_predictions() |>
  ggplot(aes(outcome, .pred, shape = id)) +
  geom_point(alpha = 0.5, position = position_jitter(width = 0.5)) +
  geom_smooth(method = "lm", se = FALSE) + # trend for each fold
  labs(
    x = "Truth",
    y = "Predicted score",
    shape = "Fold"
  )

```

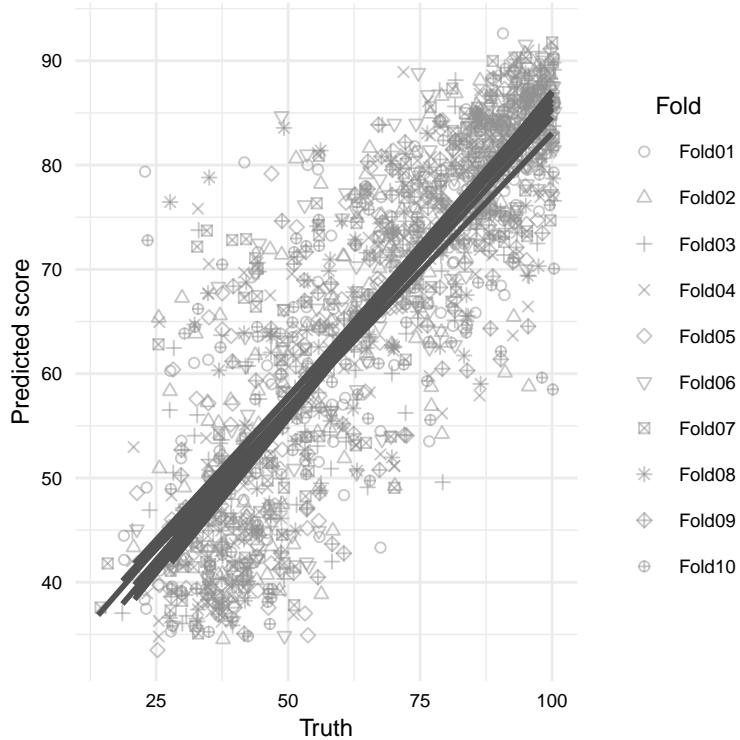


Figure 9.9: Distribution of the RMSE for the cross-validated linear regression model.

There appears to be more cohesion in the predictions, overall. The errors however seem visually to be larger for the lower scores.

At this point we can either consider this model to be good enough or we can try to improve it further. Let's take one more shot at improving the features by including bigrams. To include words (unigrams) and bigrams, we can modify the `step_tokenize()` function in our recipe. We add the `token = "ngrams"` argument and specify the options as `list(n = 2, n_min = 1)`. This will create unigrams and bigrams. The code is seen in Example 9.59.

Example 9.59.

```
# Create a recipe
cedel2_reg_rec <-
  recipe(outcome ~ text, data = cedel2_reg_train) |>
  step_tokenize(text, token = "ngrams", options = list(n = 2, n_min = 1)) |>
  step_tokenfilter(text, max_tokens = tune()) |>
  step_tfidf(text)
```

```
# Preview
cedel2_reg_rec
```

Since the features have changed, we won't assume that our tuning of the `max_tokens` is still valid. So we will tune the `max_tokens` hyperparameter again, as in Example 9.52 through Example 9.57. For the sake of brevity, we will not show the code here. We will simply show the results.

```
# Create workflow
cedel2_reg_wf_rf <-
  workflow() |>
  add_recipe(cedel2_reg_rec) |>
  add_model(cedel2_reg_spec_rf)

# Create tuning grid
cedel2_tune_grid <-
  grid_regular(max_tokens(range = c(500, 3500)), levels = 5)

# Set seed for reproducibility
set.seed(123)

# Create tuning workflow
cedel2_tune <-
  tune_grid(
    cedel2_reg_wf_rf,
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    grid = cedel2_tune_grid,
    metrics = metric_set(rmse, rsq),
    control = control_resamples(save_pred = TRUE)
  )

# Select best parameter
chosen_max_tokens <-
  cedel2_tune |>
  select_best(metric = "rmse")

# Update workflow
cedel2_reg_final_rf <-
  cedel2_reg_wf_rf |>
  finalize_workflow(chosen_max_tokens)
```

```

# Cross-validated workflow
cedel2_reg_cv_rf <-
  cedel2_reg_final_rf |>
  fit_resamples(
    resamples = vfold_cv(cedel2_reg_train, v = 10),
    control = control_resamples(save_pred = TRUE)
  )

# Collect metrics
estimates <-
  cedel2_reg_cv_rf |>
  collect_metrics() |>
  pull(3)

```

The tuning of `max_tokens` selected 2,000 features as the optimal number. The cross-validation of the training dataset produced an RMSE of 12.8 and an R^2 of 0.698. The standard error is NA. The upshot is that by including bigrams we have not improved, or worsened, the model. Including the unigrams and bigrams together may be more informative for the exploration of the features.

We could continue to try to improve the model, but at this point we have a model that is performing better than the null model and is performing better than the other models we have tried. So we will consider this model to be good enough.

Let's now fit the 1-2 gram model to the testing data and evaluate the performance on the testing set. The code is seen in Example 9.60.

Example 9.60.

```

# Fit model on training data
cedel2_reg_final_rf_fit <-
  cedel2_reg_final_rf |>
  fit(cedel2_reg_train)

# Predict on testing data
cedel2_reg_final_rf_pred <-
  bind_cols(
    cedel2_reg_test,
    predict(cedel2_reg_final_rf_fit, cedel2_reg_test)
  )

# Evaluate performance

```

```
cedel2_reg_final_rf_pred |>
  metrics(truth = outcome, estimate = .pred)
```

```
> # A tibble: 3 x 3
>   .metric .estimator .estimate
>   <chr>   <chr>       <dbl>
> 1 rmse    standard     12.9
> 2 rsq     standard     0.694
> 3 mae     standard     9.84
```

We can now visualize the feature importance of the model. The code is seen in Example 9.61.

Example 9.61.

```
# Extract predictions
cedel2_reg_final_rf_fit |>
  vip::vi(scale = TRUE)  |>
  mutate(Variable = str_replace(Variable, "^tfidf_text_", "")) |>
  slice_max(Importance, n = 20) |>
  # reorder variables by importance
  ggplot(aes(reorder(Variable, Importance), Importance)) +
  geom_point() +
  coord_flip() +
  labs(
    x = "Feature",
    y = "Importance"
  )
```

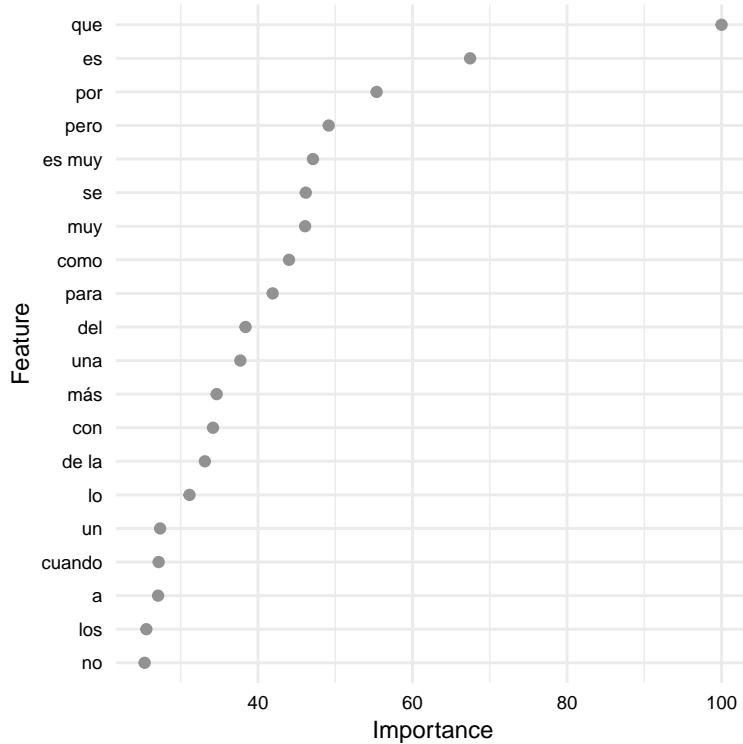


Figure 9.10: Feature importance of the random forest model.

9.3 Summary

In this chapter we have learned about supervised machine learning. We have learned about the different types of supervised machine learning methods and how they can be used to predict and classify. We have also learned about the different types of data structures that are used in supervised machine learning. Finally, we have learned about the different types of evaluation metrics that are used to evaluate the performance of supervised machine learning models.

Activities

Recipe

What: Predictive models: prep, train, test, and evaluate^a

How: Read Recipe 9 and participate in the Hypothes.is online social annotation.

Why: To illustrate some common coding strategies for preparing datasets for inferential

data analysis, as well as the steps conduct descriptive assessment, statistical interrogation, and evaluation and reporting of results.

^a<https://qtalr.github.io/qtalrkit/articles/recipe-8.html>

█ Lab

What: Predictive Data Analysis^a

How: Clone, fork, and complete the steps in Lab 10.

Why: To gain experience working with coding strategies to prepare, feature engineer, train and test a predictive model, and evaluate results from a predictive data analysis, practice transforming datasets into new object formats and visualizing relationships, and implement organizational strategies for organizing and reporting results in a reproducible fashion.

^a<https://github.com/qtalr/lab-9>

Questions

❖ Conceptual questions

1. What is the difference between a numeric and a categorical variable?
2. What is the difference between a regression and a classification model?
3. What is the difference between a training set and a testing set?
4. What is the difference between a hyperparameter and a parameter?
5. What is the difference between a supervised and an unsupervised machine learning model?
6. What advantages and disadvantages do supervised machine learning models have over traditional methods of text analysis?
7. What are some potential applications of supervised machine learning in linguistics?

❖ Applied questions

1. Write a program to build a classification model which uses a set of collected text features to predict a target variable.
2. Use the classification model to classify a series of documents and assess the accuracy of the model.
3. Develop a regression model which uses text features to predict a numeric target variable.

4. Create a text mining application to analyze a large body of text and discover correlations between variables.
5. Use a clustering algorithm to discover clusters in a large dataset, and create a visualization to present the identified clusters.
6. Analyze the structure of a text corpus and identify patterns in word usage and feature distributions.
7. Build a predictive model using text as an input and binary or categorical outcomes as the target.
8. Develop a natural language processing application which classifies text into predefined categories using a supervised learning algorithm.
9. Use a supervised learning algorithm to build a predictive model which classifies a set of unseen texts into predefined categories.
10. Develop a web application which allows users to easily explore a set of text documents, visualize the content of the documents, and generate predictive models from the text.

10 Inference



Caution

Under development.

Part V

Communication

In this section I cover the steps in presenting the findings of the research both as a research document and as a reproducible research project. Both research documents and reproducible projects are fundamental components of modern scientific inquiry. On the one hand a research document provides readers a detailed summary of the main import of the research study. On the other hand making the research project available to interested readers ensures that the scientific community can gain insight into the process implemented in the research and thus enables researchers to vet and extend this research to build a more robust and verifiable research base.

11 Reports



Caution

Under development.

12 Collaboration



Under development.

References

- Ackoff, Russell L. 1989. "From Data to Wisdom." *Journal of Applied Systems Analysis* 16 (1): 3–9.
- Ädel, Annelie. 2020. "Corpus Compilation." In *A Practical Handbook of Corpus Linguistics*, edited by Magali Paquot and Stefan Th. Gries, 3–24. Switzerland: Springer.
- Albert, Saul, Laura E. de Ruiter, and J. P. de Ruiter. 2015. "CABNC: The Jeffersonian Transcription of the Spoken British National Corpus." TalkBank.
- Allaire, JJ. 2023. *Quarto: R Interface to Quarto Markdown Publishing System*. <https://github.com/quarto-dev/quarto-r>.
- Allaire, JJ, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, et al. 2023. *Rmarkdown: Dynamic Documents for r*. <https://github.com/rstudio/rmarkdown>.
- Baayen, R. Harald. 2011. "Corpus Linguistics and Naive Discriminative Learning." *Revista Brasileira de Linguística Aplicada* 11 (2): 295–328.
- Baayen, R. Harald, L. B. Feldman, and R. Schreuder. 2006. "Morphological Influences on the Recognition of Monosyllabic Monomorphemic Words." *Journal of Memory and Language* 55: 290–313. <https://doi.org/10.1016/j.jml.2006.03.008>.
- Bao, Wang, Ning Lianju, and Kong Yue. 2019. "Integration of Unsupervised and Supervised Machine Learning Algorithms for Credit Risk Assessment." *Expert Systems with Applications* 128 (August): 301–15. <https://doi.org/10.1016/j.eswa.2019.02.033>.
- Bengtsson, Henrik. 2023. *Future: Unified Parallel and Distributed Processing in r for Everyone*. <https://future.futureverse.org>.
- Benoit, Kenneth. 2020. *Quanteda.corpora: A Collection of Corpora for Quanteda*. <http://github.com/quanteda/quanteda.corpora>.
- Benoit, Kenneth, David Muhr, and Kohei Watanabe. 2021. *Stopwords: Multilingual Stopword Lists*. <https://github.com/quanteda/stopwords>.
- Benoit, Kenneth, and Adam Obeng. 2023. *Readtext: Import and Handling for Plain and Formatted Text Files*. <https://github.com/quanteda/readtext>.
- Blischak, John, Peter Carbonetto, and Matthew Stephens. 2023. *Workflowr: A Framework for Reproducible and Collaborative Data Science*. <https://github.com/workflowr/workflowr>.
- Braginsky, Mika. 2022. *Wordbankr: Accessing the Wordbank Database*. <https://langcog.github.io/wordbankr/>.
- Bresnan, Joan. 2007. "A Few Lessons from Typology." *Linguistic Typology* 11 (1): 297–306.
- Brown, Keith. 2005. *Encyclopedia of Language and Linguistics*. Vol. 1. Elsevier.
- Buckheit, Jonathan B., and David L. Donoho. 1995. "Wavelab and Reproducible Research." In *Wavelets and Statistics*, 55–81. Springer.

- Bychkovska, Tetyana, and Joseph J. Lee. 2017. “At the Same Time: Lexical Bundles in L1 and L2 University Student Argumentative Writing.” *Journal of English for Academic Purposes* 30 (November): 38–52. <https://doi.org/10.1016/j.jeap.2017.10.008>.
- Campbell, Lyle. 2001. “The History of Linguistics.” In *The Handbook of Linguistics*, edited by Mark Aronoff and Janie Rees-Miller, 81–104. Blackwell Handbooks in Linguistics. Blackwell Publishers.
- Carmi, Elinor, Simeon J. Yates, Eleanor Lockley, and Alicja Pawluczuk. 2020. “Data Citizenship: Rethinking Data Literacy in the Age of Disinformation, Misinformation, and Malinformation.” *Internet Policy Review* 9 (2).
- Chambers, John M. 2020. “S, r, and Data Science.” *Proceedings of the ACM on Programming Languages* 4 (HOPL): 1–17. <https://doi.org/10.1145/3386334>.
- Chan, Sin-wai. 2014. *Routledge Encyclopedia of Translation Technology*. Routledge.
- Conway, Lucian Gideon, Laura Janelle Gornick, Chelsea Burfeind, Paul Mandella, Andrea Kuenzli, Shannon C. Houck, and Deven Theresa Fullerton. 2012. “Does Complex or Simple Rhetoric Win Elections? An Integrative Complexity Analysis of u.s. Presidential Campaigns.” *Political Psychology* 33 (5): 599–618. <https://doi.org/10.1111/j.1467-9221.2012.00910.x>.
- Cross, Nigel. 2006. “Design as a Discipline.” *Designerly Ways of Knowing*, 95–103.
- “Data Never Sleeps 7.0 Infographic.” 2019. <https://www.domo.com/learn/infographic/data-never-sleeps-7>.
- Deshors, Sandra C, and Stefan Th. Gries. 2016. “Profiling Verb Complementation Constructions Across New Englishes.” *International Journal of Corpus Linguistics*. 21 (2): 192–218.
- Desjardins, Jeff. 2019. “How Much Data Is Generated Each Day?” *Visual Capitalist*.
- Donoho, David. 2017. “50 Years of Data Science.” *Journal of Computational and Graphical Statistics* 26 (4): 745–66. <https://doi.org/10.1080/10618600.2017.1384734>.
- Dubnjakovic, Ana, and Patrick Tomlin. 2010. *A Practical Guide to Electronic Resources in the Humanities*. Elsevier.
- Eisenstein, Jacob, Brendan O’Connor, Noah A Smith, and Eric P Xing. 2012. “Mapping the Geographical Diffusion of New Words.” *Computation and Language*, 1–13. <https://doi.org/10.1371/journal.pone.0113114>.
- Firth, John R. 1957. *Papers in Linguistics*. Oxford University Press.
- Francom, Jerid. 2022. “Corpus Studies of Syntax.” In *The Cambridge Handbook of Experimental Syntax*, edited by Grant Goodall, 687–713. Cambridge Handbooks in Language and Linguistics. Cambridge University Press.
- . 2023. *Qtalrkit: Quantitative Text Analysis for Linguists Resource Kit*. <https://github.com/qtalr/qtalrkit>.
- Gandrud, Christopher. 2015. *Reproducible Research with r and r Studio*¹. Second edition. CRC Press.

¹<https://www.ncbi.nlm.nih.gov/pubmed/17811671>

- Garg, Nikhil, Londa Schiebinger, Dan Jurafsky, and James Zou. 2018. “Word Embeddings Quantify 100 Years of Gender and Ethnic Stereotypes.” *Proceedings of the National Academy of Sciences* 115 (16): E3635–44. <https://doi.org/10.1073/pnas.1720347115>.
- Gentleman, Robert, and Duncan Temple Lang. 2007. “Statistical Analyses and Reproducible Research.” *Journal of Computational and Graphical Statistics* 16 (1): 1–23.
- Gilquin, Gaëtanelle, and Stefan Th Gries. 2009. “Corpora and Experimental Methods: A State-of-the-Art Review.” *Corpus Linguistics and Linguistic Theory* 5 (1): 1–26. <https://doi.org/10.1515/CLLT.2009.001>.
- Gomez-Uribe, Carlos A., and Neil Hunt. 2015. “The Netflix Recommender System: Algorithms, Business Value, and Innovation.” *ACM Transactions on Management Information Systems (TMIS)* 6 (4): 1–19.
- Gries, Stefan Th. 2021. *Statistics for Linguistics with r*. De Gruyter Mouton.
- . 2023. “Statistical Methods in Corpus Linguistics.” In *Readings in Corpus Linguistics: A Teaching and Research Guide for Scholars in Nigeria and Beyond*, 78–114.
- Gries, Stefan Th. 2013. *Statistics for Linguistics with r. A Practical Introduction*. 2nd revise.
- Gries, Stefan Th., and Sandra C. Deshors. 2014. “Using Regressions to Explore Deviations Between Corpus Data and a Standard/Target: Two Suggestions.” *Corpora* 9 (1): 109–36. <https://doi.org/10.3366/cor.2014.0053>.
- Grieve, Jack, Andrea Nini, and Diansheng Guo. 2018. “Mapping Lexical Innovation on American Social Media.” *Journal of English Linguistics* 46 (4): 293–319.
- Harris, Zellig S. 1954. “Distributional Structure.” *Word* 10 (2-3): 146–62. <https://doi.org/10.1080/00437956.1954.11659520>.
- Hay, Jennifer. 2002. “From Speech Perception to Morphology: Affix Ordering Revisited.” *Language* 78 (3): 527–55.
- Hester, Jim, Hadley Wickham, and Gábor Csárdi. 2023. *Fs: Cross-Platform File System Operations Based on Libuv*. <https://fs.r-lib.org>.
- Hicks, Stephanie C., and Roger D. Peng. 2019. “Elements and Principles for Characterizing Variation Between Data Analyses.” arXiv. <https://doi.org/10.48550/arXiv.1903.07639>.
- Ide, Nancy, Collin Baker, Christiane Fellbaum, Charles Fillmore, and Rebecca Passonneau. 2008. “MASC: The Manually Annotated Sub-Corpus of American English.” In *6th International Conference on Language Resources and Evaluation, LREC 2008*, 2455–60. European Language Resources Association (ELRA).
- Ignatow, Gabe, and Rada Mihalcea. 2017. *An Introduction to Text Mining: Research Design, Data Collection, and Analysis*. Sage Publications.
- Jaeger, T Florian, and Neal Snider. 2007. “Implicit Learning and Syntactic Persistence: Surprisal and Cumulativity.” *University of Rochester Working Papers in the Language Sciences* 3 (1).
- Kaur, Jashanjot, and P. Kaur Buttar. 2018. “A Systematic Review on Stopword Removal Algorithms.” *International Journal on Future Revolution in Computer Science & Communication Engineering* 4 (4): 207–10.
- Kloumann, IM, CM Danforth, KD Harris, and CA Bliss. 2012. “Positivity of the English Language.” *PloS One*.

- Koehn, P. 2005. “Europarl: A Parallel Corpus for Statistical Machine Translation.” *MT Summit X*, 12–16.
- Kostić, Aleksandar, Tanja Marković, and Aleksandar Baucal. 2003. “Inflectional Morphology and Word Meaning: Orthogonal or Co-Implicative Cognitive Domains?” In *Morphological Structure in Language Processing*, edited by R. Harald Baayen and Robert Schreuder, 1–44. De Gruyter Mouton. <https://doi.org/10.1515/9783110910186.1>.
- Kowalski, John, and Rob Cavanaugh. 2022. *TBDBr: Easy Access to TalkBankDB via r API*. <https://github.com/TalkBank/TalkBankDB-R>.
- Krathwohl, David R. 2002. “A Revision of Bloom’s Taxonomy: An Overview.” *Theory into Practice* 41 (4): 212–18.
- Kross, Sean, Nick Carchedi, Bill Bauer, and Gina Grdina. 2020. *Swirl: Learn r, in r*. <http://swirlstats.com>.
- Kucera, H, and W N Francis. 1967. *Computational Analysis of Present Day American English*. Brown University Press Providence.
- Landau, William Michael. 2023. *Targets: Dynamic Function-Oriented Make-Like Declarative Pipelines*. <https://docs.ropensci.org/targets/>.
- Lantz, Brett. 2013. *Machine Learning with r*. Birmingham: Packt Publishing.
- Leech, Geoffrey. 1992. “100 Million Words of English: The British National Corpus (BNC),” no. 1991: 1–13.
- Lewis, Michael. 2004. *Moneyball: The Art of Winning an Unfair Game*. WW Norton & Company.
- Liu, Kanglong, and Muhammad Afzaal. 2021. “Syntactic Complexity in Translated and Non-Translated Texts: A Corpus-Based Study of Simplification.” Edited by Diego Raphael Amancio. *PLOS ONE* 16 (6): e0253454. <https://doi.org/10.1371/journal.pone.0253454>.
- Lozano, Cristóbal. 2009. “CEDEL2: Corpus Escrito Del Español L2.” *Applied Linguistics Now: Understanding Language and Mind/La Lingüística Aplicada Hoy: Comprendiendo El Lenguaje y La Mente*. Almería: Universidad de Almería, 197–212.
- Magueresse, Alexandre, Vincent Carles, and Evan Heetderks. 2020. “Low-Resource Languages: A Review of Past Work and Future Challenges.” arXiv. <https://arxiv.org/abs/2006.07264>.
- Manning, Christopher. 2003. “Probabilistic Syntax.” In *Probabilistic Linguistics*, edited by Bod, Jennifer Hay, and Jannedy, 289–341. Cambridge, MA: MIT Press.
- Marcus, Mitchell P, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. “Building a Large Annotated Corpus of English: The Penn Treebank.” *Computational Linguistics* 19 (2): 313–30.
- Marwick, Ben, Carl Boettiger, and Lincoln Mullen. 2018. “Packaging Data Analytical Work Reproducibly Using r (and Friends).” *The American Statistician* 72 (1): 80–88.
- Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. “Distributed Representations of Words and Phrases and Their Compositionality.” In *Advances in Neural Information Processing Systems*, 3111–19.
- Moroz, George. 2023. *Lingtypology: Linguistic Typology and Mapping*. <https://CRAN.R-project.org/package=lingtypology>.

- Mosteller, Frederick, and David L Wallace. 1963. “Inference in an Authorship Problem.” *Journal of the American Statistical Association* 58 (302): 275–309. <https://www.jstor.org/stable/2283270>.
- Mullen, Lincoln. 2022. *Tokenizers: Fast, Consistent Tokenization of Natural Language Text*. <https://docs.ropensci.org/tokenizers/>.
- Muñoz, Carmen, ed. 2006. *Age and the Rate of Foreign Language Learning*. 1st ed. Vol. 19. Second Language Acquisition Series. Clevedon: Multilingual Matters.
- Nisioi, Sergiu, Ella Rabinovich, Liviu P. Dinu, and Shuly Wintner. 2016. “A Corpus of Native, Non-Native and Translated Texts.” In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*. Portorož, Slovenia: European Language Resources Association (ELRA).
- Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D Manning, Ryan McDonald, et al. 2016. “Universal Dependencies V1: A Multilingual Treebank Collection.” *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*, 1659–66. <https://doi.org/>?
- Nivre, Joakim, Marie-Catherine De Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. “Universal Dependencies V2: An Evergrowing Multilingual Treebank Collection.” *arXiv Preprint arXiv:2004.10643*. <https://arxiv.org/abs/2004.10643>.
- Olohan, Maeve. 2008. “Leave It Out! Using a Comparable Corpus to Investigate Aspects of Explicitation in Translation.” *Cadernos de Tradução*, 153–69.
- Ooms, Jeroen. 2023. *Jsonlite: A Simple and Robust JSON Parser and Generator for r*. <https://jeroen.r-universe.dev/jsonlite> <https://arxiv.org/abs/1403.2805>².
- Paquot, Magali, and Stefan Th. Gries, eds. 2020. *A Practical Handbook of Corpus Linguistics*. Switzerland: Springer.
- Petrenz, Philipp, and Bonnie Webber. 2011. “Stable Classification of Text Genres.” *Computational Linguistics* 37 (2): 385–93. https://doi.org/10.1162/COLI_a_00052.
- R Special Interest Group on Databases (R-SIG-DB), Hadley Wickham, and Kirill Müller. 2022. *DBI: R Database Interface*. <https://dbi.r-dbi.org>.
- Riehemann, Susanne Z. 2001. “A Constructional Approach to Idioms and Word Formation.” PhD thesis, Stanford.
- Rinker, Tyler. 2019. *Lexicon: Lexicons for Text Analysis*. <https://github.com/trinker/lexicon>.
- Rinker, Tyler, and Dason Kurkiewicz. 2019. *Pacman: Package Management Tool*. <https://github.com/trinker/pacman>.
- Robinson, David, and Julia Silge. 2023. *Tidytext: Text Mining Using Dplyr, Ggplot2, and Other Tidy Tools*. <https://github.com/juliasilge/tidytext>.
- Roediger, H. L. L., and K. B. B McDermott. 2000. “Distortions of Memory.” *The Oxford Handbook of Memory*, 149–62.

²<https://jeroen.r-universe.dev/jsonlite%0A><https://arxiv.org/abs/1403.2805>

- Rowley, Jennifer. 2007. “The Wisdom Hierarchy: Representations of the DIKW Hierarchy.” *Journal of Information Science* 33 (2): 163–80. <https://doi.org/10.1177/0165551506070706>.
- Saxena, Shweta, and Manasi Gyanchandani. 2020. “Machine Learning Methods for Computer-Aided Breast Cancer Diagnosis Using Histopathology: A Narrative Review.” *Journal of Medical Imaging and Radiation Sciences* 51 (1): 182–93.
- Sedgwick, Philip. 2015. “Units of Sampling, Observation, and Analysis.” *BMJ (Online)* 351 (October): h5396. <https://doi.org/10.1136/bmj.h5396>.
- Silge, Julia. 2022. *Janeaustenr: Jane Austen’s Complete Novels*. <https://github.com/juliasilge/janeaustenr>.
- Szmrecsanyi, Benedikt. 2004. “On Operationalizing Syntactic Complexity.” In *Le Poids Des Mots. Proceedings of the 7th International Conference on Textual Data Statistical Analysis. Louvain-La-Neuve*, 2:1032–39.
- Talarico, Jennifer M., and David C. Rubin. 2003. “Confidence, Not Consistency, Characterizes Flashbulb Memories.” *Psychological Science* 14 (5): 455–61. <https://doi.org/10.1111/1467-9280.02453>.
- Tottie, Gunnell. 2011. “Uh and Um as Sociolinguistic Markers in British English.” *International Journal of Corpus Linguistics* 16 (2): 173–97.
- University of Colorado Boulder. 2008. “Switchboard Dialog Act Corpus. Web Download.” Linguistic Data Consortium.
- Voigt, Rob, Nicholas P. Camp, Vinodkumar Prabhakaran, William L. Hamilton, Rebecca C. Hetey, Camilla M. Griffiths, David Jurgens, Dan Jurafsky, and Jennifer L. Eberhardt. 2017. “Language from Police Body Camera Footage Shows Racial Disparities in Officer Respect.” *Proceedings of the National Academy of Sciences* 114 (25): 6521–26.
- Waring, Elin, Michael Quinn, Amelia McNamara, Eduardo Arino de la Rubia, Hao Zhu, and Shannon Ellis. 2022. *Skimr: Compact and Flexible Summaries of Data*. <https://docs.ropensci.org/skimr/>.
- White, John Myles. 2023. *ProjectTemplate: Automates the Creation of New Statistical Analysis Projects*. <http://projecttemplate.net>.
- Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (10). <https://doi.org/10.18637/jss.v059.i10>.
- . 2022. *Stringr: Simple, Consistent Wrappers for Common String Operations*. <https://stringr.tidyverse.org>.
- . 2023. *Tidyverse: Easily Install and Load the Tidyverse*. <https://tidyverse.tidyverse.org>.
- Wickham, Hadley, Jennifer Bryan, Malcolm Barrett, and Andy Teucher. 2023. *Usethis: Automate Package and Project Setup*. <https://usethis.r-lib.org>.
- Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. *Dplyr: A Grammar of Data Manipulation*. <https://dplyr.tidyverse.org>.
- Wickham, Hadley, Maximilian Girlich, and Edgar Ruiz. 2023. *Dbplyr: A Dplyr Back End for Databases*. <https://dbplyr.tidyverse.org/>.
- Wickham, Hadley, Jim Hester, and Jennifer Bryan. 2023. *Readr: Read Rectangular Text Data*. <https://readr.tidyverse.org>.

- Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2022. *Devtools: Tools to Make Developing r Packages Easier*. <https://devtools.r-lib.org/>.
- Wickham, Hadley, Evan Miller, and Danny Smith. 2023. *Haven: Import and Export SPSS, Stata and SAS Files*. <https://haven.tidyverse.org>.
- Wickham, Hadley, Davis Vaughan, and Maximilian Girlich. 2023. *Tidyr: Tidy Messy Data*. <https://tidyr.tidyverse.org>.
- Wijffels, Jan. 2023. *Udpipe: Tokenization, Parts of Speech Tagging, Lemmatization and Dependency Parsing with the UDPipe 'NLP' Toolkit*. <https://bnosac.github.io/udpipe/en/index.html>.
- Wulff, S, A Stefanowitsch, and Stefan Th. Gries. 2007. “Brutal Brits and Persuasive Americans.” *Aspects of Meaning*.
- Xie, Yihui. 2023a. *Bookdown: Authoring Books and Technical Documents with r Markdown*. <https://github.com/rstudio/bookdown>.
- . 2023b. *Tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>.
- Zipf, George Kingsley. 1949. *Human Behavior and the Principle of Least Effort*. Oxford, England: Addison-Wesley Press.

A Data

A.1 ANC

- ANC

A.2 BNC

- Brown Corpus

A.3 CABNC

- CABNC

A.4 CEDEL2

- CEDEL2

A.5 ENNTT

- ENNTT
- Europarl
- Federalist Papers (LOC)
- SOTU
- SWDA
- ...

B Feedback

Thank you for taking the time to read through this book. I really value your opinion and I would love to hear your thoughts as I continue to make progress.

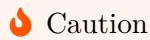
Where to review

Chapters that are ready for feedback will appear with the following callout.



Ready for review.

Those that are not ready will appear with the following callout.



Under development.

In a few cases a chapter will be ready for review, but I'll still be working on the exercises, callouts, *etc.* In those cases, the items that are still under development will be marked with the  icon.

What to look for

As you read over the draft, I'd appreciate your feedback in the following areas:

1. Clarity and Comprehensibility

I'd love to know if you think the content is clear and easy to understand. Do you think the concepts are broken down enough? Are the examples helpful? If anything seems too jargon-y or confusing, definitely let me know.

2. Consistency

It's pretty important to keep things smooth. So, keep an eye out for any inconsistent writing styles, terminology, or layout. If something seems off, I'd appreciate it if you point it out.

3. Relevance

Does the material match the current standards and knowledge? Will it be of interest to linguists? If something feels outdated or irrelevant, don't hesitate to mention it.

4. Engagement

I'm not looking to drop a boring read on people. So, as you're going through it, think about whether it holds your interest. Maybe the prose needs more life, the examples need to be more diverse, or the exercises could be more or less challenging. If you have any ideas, I'm all ears.

How to submit feedback

Depending on your preference, you can submit feedback in one of three ways:

- [hypothes.is](https://hypothes.is/groups/wppaYKxy/qtal-feedback)¹

This is the easiest way to submit feedback. Join the “qtal_feedback” annotation group and just highlight the text you want to comment on and click the “Annotate” button. You can also add comments to the right sidebar.

- GitHub issues²

This book is hosted on GitHub, so you can submit feedback directly through the issues page for the repository. Just click the “New issue” button and fill out the form. You'll need a GitHub account to do this.

- Email me at francojc@wfu.edu³

If you'd rather not use the other options, you can always email me directly. Just make sure to try to include references to the specific parts of the book you're referring to. A link or section number will do.

Thank yous!

I want to thank you beforehand for your willingness to help me out. I really appreciate it. I also want to thank you in print. Please give me the name you would like to appear in the Acknowledgements section. If you'd rather not be acknowledged in the final version of the book, please let me know.

¹<https://hypothes.is/groups/wppaYKxy/qtal-feedback>

²<https://github.com/qtalr/book/issues>

³<mailto:francojc@wfu.edu>

Index

data science, 31
dataset, 58
experimental data, 34
observational data, 34
population, 45
reproducible research, 32
research question, 122
sample, 45
 representativeness, 45
sample size, 46
Text analysis, 36