

*Jerid Francom*

---

# **Quantitative Text Analysis for Linguistics**

*Reproducible Research using R*



---

---

## ***Table of contents***

---

<b>Welcome</b>	<b>9</b>
License . . . . .	9
Credits . . . . .	10
Acknowledgements . . . . .	10
Build information . . . . .	10
<b>Preface</b>	<b>13</b>
<b>Preface</b>	<b>13</b>
Rationale . . . . .	13
Aims . . . . .	14
Approach . . . . .	15
Structure . . . . .	16
Book level . . . . .	16
Chapter level . . . . .	17
Resources . . . . .	18
Getting started . . . . .	18
R and IDEs . . . . .	19
R packages . . . . .	20
Git and GitHub . . . . .	21
Getting help . . . . .	22
Conventions . . . . .	23
Prose . . . . .	23
Code blocks . . . . .	23
Callouts . . . . .	24
To the instructor . . . . .	25
Basic Introduction . . . . .	25
Intermediate Introduction . . . . .	26
Advanced Introduction . . . . .	26
Activities . . . . .	26
Summary . . . . .	27
Questions . . . . .	28
<b>1 Text analysis in context</b>	<b>35</b>
1.1 Making sense of a complex world . . . . .	36
1.1.1 Heuristic Understanding . . . . .	36

1.1.2	Science to advance understanding . . . . .	37
1.2	Data analysis . . . . .	38
1.2.1	Emergence of data science . . . . .	38
1.2.2	Computing skills, statistical knowledge, and domain knowledge . . . . .	39
1.2.3	Applications of data science . . . . .	40
1.3	Language analysis . . . . .	40
1.4	Text analysis . . . . .	44
1.4.1	Approaches . . . . .	44
1.4.2	Implementation . . . . .	45
1.4.3	Applications . . . . .	45
	Summary . . . . .	48
	Activities . . . . .	49
	Questions . . . . .	50
<b>I</b>	<b>Orientation</b>	<b>31</b>
<b>2</b>	<b>Understanding data</b>	<b>57</b>
2.1	Data . . . . .	58
2.1.1	Populations . . . . .	58
2.1.2	Samples . . . . .	59
2.1.3	Corpora . . . . .	60
<b>II</b>	<b>Foundations</b>	<b>53</b>
2.2	Information . . . . .	72
2.2.1	Organization . . . . .	72
2.2.2	Transformation . . . . .	76
2.3	Documentation . . . . .	81
2.3.1	Data origin information . . . . .	81
2.3.2	Data dictionaries . . . . .	81
	Summary . . . . .	82
	Activities . . . . .	83
	Questions . . . . .	84
<b>3</b>	<b>Approaching analysis</b>	<b>87</b>
3.1	Descriptive assessment . . . . .	88
3.1.1	Information values . . . . .	89
3.1.2	Summaries . . . . .	91
3.2	Types of analysis . . . . .	106
3.2.1	Inferential data analysis . . . . .	107
3.2.2	Predictive data analysis . . . . .	112
3.2.3	Exploratory data analysis . . . . .	118
3.3	Reporting . . . . .	121
	Summary . . . . .	122
	Activities . . . . .	124
	Questions . . . . .	124

<b>4</b>	<b>Framing research</b>	<b>127</b>
4.1	Keys to strong research . . . . .	128
4.2	Connect . . . . .	128
4.2.1	Research area . . . . .	128
4.2.2	Research problem . . . . .	130
4.3	Findings . . . . .	132
4.3.1	Research aim . . . . .	132
4.3.2	Research question . . . . .	133
4.4	Blueprint . . . . .	134
4.4.1	Identify . . . . .	135
4.4.2	Plan . . . . .	137
4.4.3	Scaffold . . . . .	140
	Summary . . . . .	141
	Activities . . . . .	142
	Questions . . . . .	143
<b>5</b>	<b>Acquire data</b>	<b>149</b>
5.1	Downloads . . . . .	150
5.1.1	Manual . . . . .	150
5.1.2	Programmatic . . . . .	151
5.2	APIs . . . . .	161
5.2.1	Open access . . . . .	161
5.2.2	Authentication . . . . .	171
5.3	Web scraping . . . . .	177
5.3.1	A toy example . . . . .	177
5.3.2	A practical example . . . . .	180
5.3.3	Scaling up . . . . .	185
5.4	Documentation . . . . .	194
	Summary . . . . .	196
	Activities . . . . .	196
	Questions . . . . .	197
<b>6</b>	<b>Curate data(sets)</b>	<b>199</b>
6.1	Unstructured . . . . .	200
6.1.1	Orientation . . . . .	200
6.1.2	Tidy the data . . . . .	201
6.1.3	Write dataset . . . . .	205
6.1.4	Summary . . . . .	206
6.2	Structured . . . . .	208
6.2.1	Orientation . . . . .	209
6.2.2	Tidy the datasets . . . . .	211
6.2.3	Write dataset . . . . .	214
6.2.4	Summary . . . . .	215
6.3	Semi-structured . . . . .	216
6.3.1	Orientation . . . . .	216

6.3.2 Tidy the data . . . . .	220
6.3.3 Write datasets . . . . .	233
6.3.4 Summary . . . . .	234
6.4 Documentation . . . . .	235
Summary . . . . .	237
Activities . . . . .	238
Questions . . . . .	239
<b>7 Transform datasets</b>	<b>241</b>
7.1 Normalize . . . . .	242
7.2 Recode . . . . .	247
7.3 Generate . . . . .	255
7.4 Merge . . . . .	260
7.5 Documentation . . . . .	266
Summary . . . . .	268
Activities . . . . .	268
Questions . . . . .	269
<b>III Preparation</b>	<b>145</b>
<b>8 Exploration</b>	<b>275</b>
8.1 Orientation . . . . .	275
8.1.1 Research goals . . . . .	276
8.1.2 Approaches . . . . .	276
8.1.3 Workflow . . . . .	276
8.2 Analysis . . . . .	278
8.2.1 Descriptive analysis . . . . .	278
8.2.2 Unsupervised learning . . . . .	279
8.3 Summary . . . . .	280
Activities . . . . .	280
Questions . . . . .	281
<b>9 Prediction</b>	<b>283</b>
9.1 Orientation . . . . .	283
9.1.1 Research goals . . . . .	284
9.1.2 Approaches . . . . .	284
9.1.3 Workflow . . . . .	284
<b>IV Analysis</b>	<b>271</b>
9.2 Analysis . . . . .	291
9.2.1 Classification . . . . .	291
9.2.2 Regression . . . . .	293
9.3 Reporting . . . . .	294
9.4 Summary . . . . .	295
Activities . . . . .	295
Questions . . . . .	296

<i>Contents</i>	7
<b>10 Inference</b>	<b>299</b>
10.1 Orientation . . . . .	300
10.1.1 Research goals . . . . .	300
10.1.2 Approaches . . . . .	300
10.1.3 Workflow . . . . .	301
10.2 Analysis . . . . .	302
10.2.1 Categorical dependent variable . . . . .	303
10.2.2 Continuous dependent variable . . . . .	304
10.3 Reporting . . . . .	304
10.4 Summary . . . . .	304
Activities . . . . .	304
Questions . . . . .	305
10.5 Preparation . . . . .	306
10.6 Univariate analysis . . . . .	312
10.6.1 Categorical . . . . .	312
10.6.2 Continuous . . . . .	316
10.7 Bivariate analysis . . . . .	320
10.7.1 Categorical . . . . .	320
10.7.2 Continuous . . . . .	324
10.8 Multivariate analysis . . . . .	329
10.8.1 Categorical . . . . .	329
10.8.2 Continuous . . . . .	333
10.9 Summary . . . . .	338
<b>11 Reporting</b>	<b>343</b>
Questions . . . . .	346
<b>12 Collaboration</b>	<b>347</b>
Questions . . . . .	348
<b>References</b>	<b>349</b>
<b>References</b>	<b>349</b>
<b>V Communication</b>	<b>339</b>
<b>A Data</b>	<b>355</b>



---

# 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 large-scale language archives (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. Francom<sup>1</sup> is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

---

<sup>1</sup><https://francojc.github.io/>

---

## Credits

Font Awesome Icons<sup>2</sup> 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, (*add your name here!*). As always, any errors or omissions are my own.

---



---

## Build information

This textbook was built with the `quarto` package (Allaire 2022) and the `bookdown` package (Xie 2023a) for R. The source code for this book is available on GitHub<sup>3</sup>.

This version of the textbook was built with R version 4.3.0 (2023-04-21) on macOS Ventura 13.4 with the following packages:

---

package	version	source
dplyr	1.1.2	CRAN (R 4.3.0)
forcats	1.0.0	CRAN (R 4.3.0)
ggplot2	3.4.2	CRAN (R 4.3.0)
here	1.0.1	CRAN (R 4.3.0)
knitr	1.43	CRAN (R 4.3.0)
lubridate	1.9.2	CRAN (R 4.3.0)
purrr	1.0.1	CRAN (R 4.3.0)
qtalrkit	0.0.0.9000	Github (qtalr/qtalrkit@fd9391d00026e0455856ea345ee19036fa4d8257)
readr	2.1.4	CRAN (R 4.3.0)
readtext	0.90	CRAN (R 4.3.0)
rmarkdown	2.22	CRAN (R 4.3.0)
stringr	1.5.0	CRAN (R 4.3.0)

---

<sup>2</sup><https://fontawesome.com/>

<sup>3</sup><https://github.com/qtalr/book>

package	version	source
tadr	0.1.2	local (/Users/francojc/Documents/Academic/Research/Projects/1Active/tad/tadr)
tibble	3.2.1	CRAN (R 4.3.0)
tidyr	1.3.0	CRAN (R 4.3.0)
tidytext	0.4.1	CRAN (R 4.3.0)
tidyverse	2.0.0	CRAN (R 4.3.0)
webshot	0.5.4	CRAN (R 4.3.0)



---

## Preface

---

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 introduc-

tion 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 process<sup>4</sup>. 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.

---

<sup>4</sup>Research 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.

## 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.

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 1<sup>5</sup>.

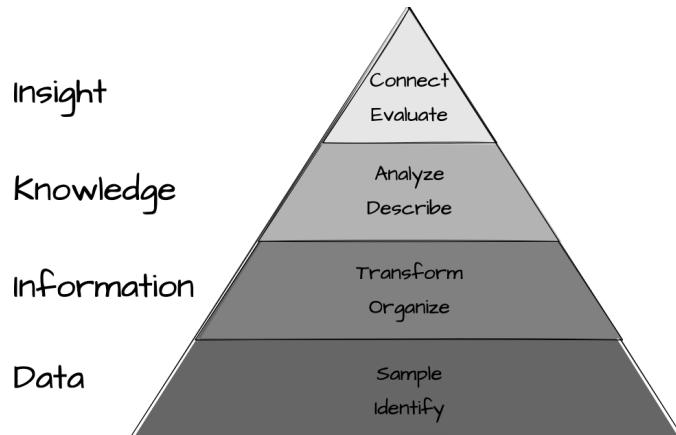


Figure 1: Data to Insight Hierarchy (DIKI)

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

<sup>5</sup>Adapted from Ackoff (1989) and Rowley (2007).

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 stages<sup>6</sup>. The chapter-level structure is consistent across chapters and can be seen in Table 0.2.

Table 0.2: 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	Identify the learning objectives for the chapter	Textbook	Introduction
Overview	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

---

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

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 chapter concludes 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 Resources<sup>7</sup>(qtalr website) that includes programming tutorials and demonstrations to enhance the reader's recognition of how programming strategies are implemented. Finally, a GitHub repository<sup>8</sup> 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 text<sup>9</sup>.

---

## 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.

---

<sup>7</sup><https://qtalr.github.io/qtalrkit/>

<sup>8</sup><https://github.com/qtalr>

<sup>9</sup>Errata for the textbook is found on the qtalr website.

## 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 Programming<sup>10</sup> website and is available for macOS, Linux, and Windows<sup>11</sup> 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 RStudio<sup>12</sup> or Visual Studio Code<sup>13</sup>, 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 cloud, a **remote environment**. There are a number of cloud-based options for working with R, including RStudio Cloud<sup>14</sup> and Microsoft Azure<sup>15</sup>. 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 Docker<sup>16</sup>, 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 project<sup>17</sup> and can be used locally or in the cloud.

There are trade-offs in terms of cost, convenience, and flexibility when choosing

---

<sup>10</sup><https://www.r-project.org/>

<sup>11</sup><https://cloud.r-project.org/>

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

<sup>13</sup><https://code.visualstudio.com/>

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

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

<sup>16</sup><https://www.docker.com/>

<sup>17</sup><https://rocker-project.org/>

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<sup>a</sup>
- Choosing and setting up an IDE<sup>b</sup>
- Working with R in remote and virtual environments<sup>c</sup>

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

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

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

## R packages

Throughout your R programming journey you will take advantage of code created by other R users in the form of packages. A package is a downloadable set of functions and/ or datasets which aim to accomplish a given cohesive set of related tasks. There are official R package repositories such as CRAN<sup>18</sup> (Comprehensive R Archive Network) and other packages are available on code-sharing repositories such as GitHub<sup>19</sup>.

### Consider this

The Comprehensive R Archive Network (CRAN) includes groupings of popular packages related to a given applied programming task called Task Views<sup>a</sup>. 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.

<sup>a</sup><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.

<sup>18</sup><https://cran.r-project.org/>

<sup>19</sup><https://github.com/>

Install the following packages from CRAN.

- **tidyverse** (Wickham 2023)
- **rmarkdown** (Allaire et al. 2023)
- **quarto** (Allaire 2022)
- **tinytex** (Xie 2023b)
- **devtools** (Wickham et al. 2022)
- **usethis** (Wickham 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<sup>a</sup>

---

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

## Git and GitHub

- Move the labs to qtalr and then create a starred list to link here

GitHub<sup>20</sup> 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 textbook<sup>21</sup> are shared on GitHub. To access and complete these exercises you will need to sign up for a (free) GitHub account<sup>22</sup> 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.

---

<sup>20</sup><https://github.com/>

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

<sup>22</sup>[https://github.com/signup?ref\\_cta=Sign+up&ref\\_loc=header+logged+out&ref\\_page=%2F&source=header-home](https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home)

**💡 Guides**

- Setting up Git and GitHub<sup>a</sup>

<sup>a</sup><https://qtalr.github.io/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! Experienced programmers and novices alike require support. Fortunately there is a very large community of programmers who have developed many official support resources and who actively contribute to unofficial discussion forums. Together these resources provide ample methods for overcoming any challenge.

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-Universe<sup>23</sup>.

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 Overflow<sup>24</sup>. There are also R-specific discussion forums such as RStudio Community<sup>25</sup>.

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.

<sup>23</sup><https://r-universe.dev/search/>

<sup>24</sup><https://stackoverflow.com/>

<sup>25</sup><https://community.rstudio.com/>

**↳ Guides**

- Creating reproducible examples<sup>a</sup>

<sup>a</sup><https://qtalr.github.io/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<sup>26</sup>**
  - Links to internal and external resources, footnotes, and citations including references to R packages when first introduced.

### Code blocks

More lengthy code will be presented in code blocks as seen below.

```
# 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 this code block. First, this code block 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 chunks 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/>.

## 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 questions<sup>27</sup>, collaborate with peers, and seek help from the ample resources available online when they encounter scope-limited programming problems.

---

### Activities

At this point you should have a working R environment with the core packages including `qtlrkit` installed. You should also have verified that you have a working Git environment and that you have a GitHub account. If you have not

---

<sup>27</sup>If 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.is<sup>28</sup> to facilitate reader questions and discussion.

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<sup>a</sup>

**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.

<sup>a</sup><https://github.com/qtalr/swirl>

### ❖ Recipe

**What:** Literate programming I<sup>a</sup>

**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.

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

### ⚠ Lab

**What:** Literate programming I<sup>a</sup>

**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.

<sup>a</sup><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

### Conceptual questions

1. What is the purpose of the textbook and what are the three skills it aims to scaffold?
  2. What are the key components of quantitative text analysis?
  3. What is the role of programmatic approaches in quantitative text analysis?
  4. What are the potential challenges involved in conducting quantitative text analysis and why are the gains worth the effort?
  5. What are some key resources available to support learning and conducting quantitative text analysis effectively?
  6. How does the structure of the textbook and associated resources work to support learning and proficiency in the areas of data literacy, research skills, and programming skills?
  7. What are the conventions used in the textbook to signal important concepts and questions to explore?
  8. How is the textbook designed to be accessible for both novice and seasoned practitioners in the area of quantitative text analysis?
  9. What are some key resources available to obtain support for learning and conducting quantitative text analysis effectively?
  10. What is the relationship between R and an IDE (e.g. RStudio, VS Code)?
  11. What is the relationship between R and a version control system (e.g. Git, GitHub)?
  12. What is the relationship between R and a package manager (e.g. CRAN, Bioconductor)?
-

- What is the goal of data science, and how has it become popular in recent years?
- Who is the intended readership for this textbook, and why is it designed to be accessible to a wide audience?
- What are the main components of each chapter, and how are they structured to support learning outcomes?
- Why is programming emphasized as an important skill for implementing text analysis techniques, and what language is used in this textbook?
- What resources are available to support the aims and approach of the textbook, and how are they helpful for readers?

### Technical exercises

1. Install the latest version of R by following the instructions for your operating system. <https://cran.r-project.org/>
2. Install an IDE
- RStudio: a popular Integrated Development Environment (IDE) for R. This will provide a user-friendly interface for writing, testing, and debugging R code. <https://rstudio.com/products/rstudio/download/>
- VS Code: a popular IDE for many programming languages. This will provide a user-friendly interface for writing, testing, and debugging R code. <https://code.visualstudio.com/download>
3. Install Git, a version control system that allows you to track changes to files and collaborate with others. <https://git-scm.com/downloads>
4. Create a GitHub account. <...>
5. Install the `tidyverse` package by running `install.packages("tidyverse")` in the R Console pane.
6. Install the `swirl` package by running `install.packages("swirl")` in the R Console pane.
7. Open RStudio and create a new project for this textbook. This will help you keep your code and files organized.
8. Fork the textbook repository to own GitHub repository and then clone it to your local machine. This will create a local copy of the textbook on your computer.
9. Open the textbook in RStudio and explore the structure of the project.
10. Open the `index.qmd` file and knit the document. This will render the textbook in HTML format.
11. Add, commit, and push your changes to the textbook to your GitHub repository.



— | — | —

# Part I

# Orientation



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

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.



# 1

---

## *Text analysis in context*

---

Science walks forward on two feet, namely theory and experiment...Sometimes it is one foot which is put forward first, sometimes the other, but continuous progress is only made by the use of both.

— Robert A. Millikan

### ▀ Outcomes

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

- Need to add more on how this chapter provides a foundation for the rest of the textbook.

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<sup>a</sup>

**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.

<sup>a</sup><https://github.com/qtalr/swirl>

## 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 Machine<sup>1</sup>.) 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 is<sup>2</sup>, it is still a small fragment of the written language that is produced on the web, just the web<sup>4</sup>. Consider that for a moment.

---

<sup>1</sup><https://web.archive.org/>

<sup>2</sup>As of 22 July 2021, there are 6,341,359 articles in the English Wikipedia<sup>3</sup> containing over 3.9 billion words occupying around 19 gigabytes of information.

<sup>4</sup>For reference, Common Crawl<sup>5</sup> has millions of gigabytes collected since 2008.

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 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 with precision and therefore we cannot 'trust' our intuitions, at least in an absolute sense.

**💡 Consider this**

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

### 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, modifying, and manipulating 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 sometime 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 and innovative uses of this technology the world wide web took flight. 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.

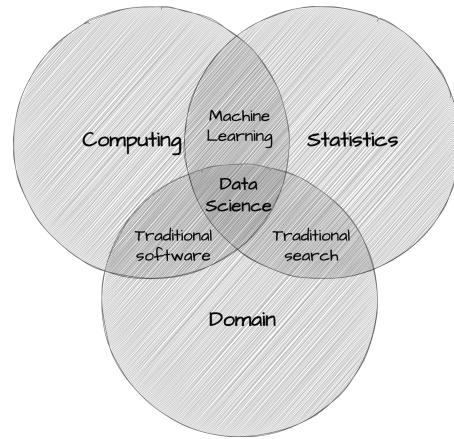


Figure 1.1: Data Science Venn Diagram adapted from Drew Conway<sup>6</sup>.

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.

- Add OpenAI's GPT to this list

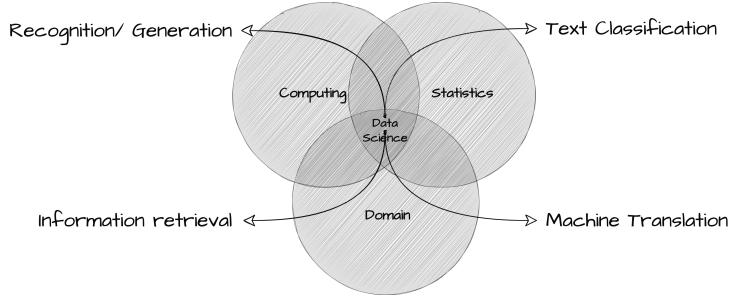


Figure 1.2: Well-known language applications

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, *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.*). The 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. 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.

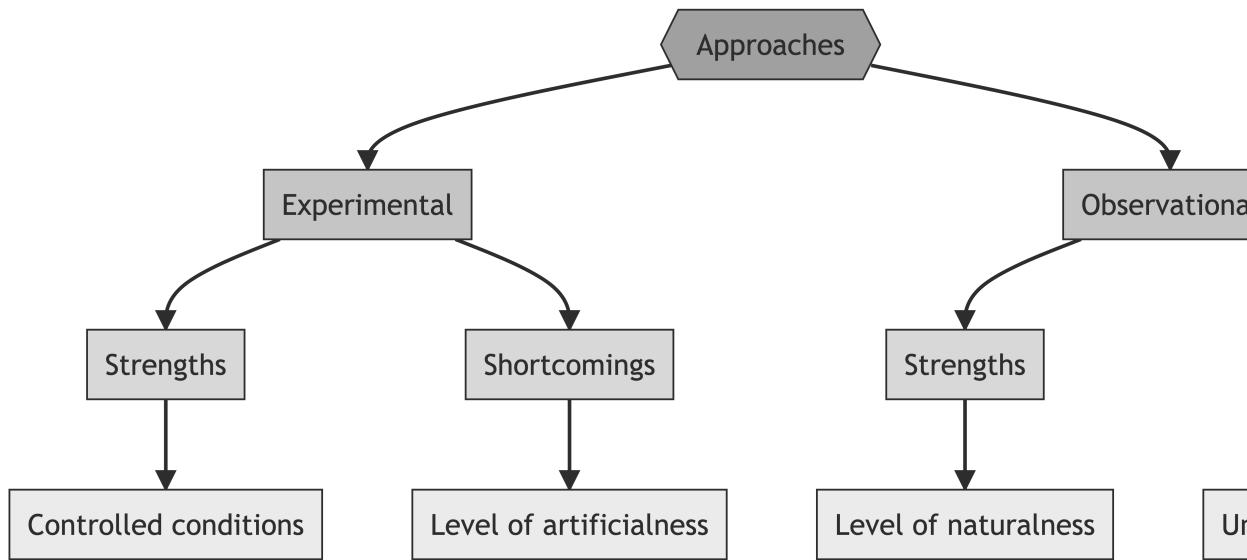


Figure 1.3: Experimental and observational data collection methods.

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. 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 ultimate goal for deriving knowledge and insight.

---

## 1.4 Text analysis

- Revise this section.

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-digestible 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 the data, **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.

For research aimed at predicting some target outcome variable, **predictive data analysis** (PDA) is employed. PDA is a supervised machine learning task that uses a set of features to predict a target outcome variable. The burden of evaluation is on the predictive power of the model.

For research aimed at explaining relationships between variables and the population from which the sample was drawn, **inferential data analysis** (IDA) is employed. IDA is a statistical learning task that uses a set of features to predict a target outcome variable. The burden of evaluation is on the explanatory power of the model.

In this way, 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 languages form 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 the most popular and widely used, particularly in linguistics. 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.* the text analysis methods we will cover are at play. These uses of text analysis are production-level applications aimed at performing practical tasks for consumers.

#### 💡 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 can be applied to a wide range of topics and research questions in both theoretical and applied subfields.

#### 📄 Case study

- ... theoretical examples ...
- morphology?
- syntax?

... applied examples ...

#### 📄 Case study

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 (i.e. as the result of), in argumentative essays. The authors used the Michigan Corpus of Upper-Level Student Papers (MICUSP) corpus using the argumentative essay section for L1-English and the Corpus of Ohio Learner and Teacher English (COLTE) for the L1-Chinese English essays. They found that L1-Chinese writers used more than 2 times as many bundle types than L1-English peers which they attribute to L1-Chinese writers attempt to avoid uncommon expressions and/or due to their lack of register awareness (conversation has more bundles than writing generally).

### Case study

Wulff, Stefanowitsch, and Gries (2007) explore differences between British and American English at the lexico-syntactic level in the *into-causative* construction (ex. ‘He tricked me into employing him.’). The analysis uses newspaper text (The Guardian and LA Times) and the findings suggest that American English uses this construction in verbal persuasion verbs whereas British English uses physical force verbs.

- What is the role of text analysis in linguistics research?
- Means to an end: text analysis as a tool for other language research methods (hypothesis generation, external valid data collection, linguistic annotation, methodological triangulation, etc. (Francom 2022))

... examples of text analysis to support other research methods ...

- End in itself: text analysis as the research method to gain a deeper understanding of language structure, function, variation, and acquisition, which can contribute to both theoretical (phonetics, morphology, syntax, semantics) and applied research (language acquisition, sociolinguistics, computational linguistics, psycholinguistics).

... examples of text analysis as the research method

- Add citations to a few examples of text analysis in linguistics research, instead of the abstracts?

... research from areas such as translation, stylistics, language variation, dialectology, psychology, psycholinguistics, political science, and sociolinguistics which highlights the diversity of fields and subareas which employ quantitative text analysis.

### 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 benefit from the use of text analysis?

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

### 🔥 Caution

#### Case study

Eisenstein et al. (2012) track the geographic spread of neologisms from city to city in the United States using Twitter data collected between 6/2009 and 5/2011. They only used tweets with geolocation data and then associated each tweet with a zip code using the US Census. The most populous metropolitan areas were used. They also used the demographics from these areas to make associations between lexical innovations and demographic attributes. From this analysis they are able to reconstruct a network of linguistic influence. One of the main findings is that demographically-similar cities are more likely to share linguistic influence. At the individual level, there is a strong, potentially stronger role of demographics than geographical location.

### 🔥 Caution

#### Case study

Voigt et al. (2017) explore potential racial disparities in officer respect in police body camera footage. The dataset is based on body camera footage from the Oakland Police Department during April 2014. A total of 981 stops by 245 different officers were included (black 682, white 299) and resulted in 36,738 officer utterances. The authors found evidence for racial disparities in respect but not formality of utterances, with less respectful language used with the black community members.

### 🔥 Caution

#### Case study

Jaeger and Snider (2007) use a corpus study to investigate the phe-

nomenon of syntactic persistence, the increased tendency for speakers to use a particular syntactic form over an alternate when the syntactic form is recently processed. The authors attempt to distinguish between two competing explanations for the phenomenon: (1) transient activation, where the increased tendency is short-lived and time-bound and (2) implicit learning, where the increased tendency is a reflect of learning mechanisms. The use of a speech corpora (Switchboard and spoken BNC) were used to avoid the artificialness that typically occurs in experimental settings. The authors investigated the ditransitive alternation (NP PP/ NP NP), voice alternation (active/ passive), and complementizer/ relativizer omission. In these alternations structural bias was established by measuring the probability for a verb form to appear in one of the two syntactic forms. Then the probability that that form (target) would change given previous exposure to the alternative form (prime) was calculated; what the authors called surprisal. Distance between the prime structure and the target verb were considered in the analysis. In these alternations, the less common structure was used in the target more often when it corresponded to the prime form (higher surprisal) suggesting that implicit learning underlies syntactic persistence effects.

### 🔥 Caution

#### Case study

Olohan (2008) investigate the extent to which translated texts differ from native texts. In particular the author explores the notion of explication in translated texts (the tendency to make information in the source text explicit in the target translation). The study makes use of the Translational English Corpus (TEC) for translation samples and comparable sections of the British National Corpus (BNC) for the native samples. The results suggest that there is a tendency for syntactic explication in the translational corpus (TEC) which is assumed to be a subconscious process employed unwittingly by translators.

## 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 programmatic skills but also enhance your appreciation for the richness of human language and its important role in our everyday lives.

---

## Activities

- Add summary of the goals and outcomes of these activities.

### \_recipe

**What:** Literate programming II<sup>a</sup>

**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.

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

### \_lab

**What:** Literate programming II<sup>a</sup>

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

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

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

## Questions

### Conceptual questions

1. Discriminate between quantitative and non-quantitative research.
  2. Identify research in an area of interest in linguistics that has taken a quantitative approach to text analysis.
  3. What are the benefits of reproducible research in data science?
  4. In your own words, define literate programming?
  5. What are the benefits of literate programming?
  6. What are the benefits of using R and Quarto for literate programming?
- What is the importance of science in making sense of a complex world?
  - How has the tool belt for scientific research and exploration changed in the 21st century?
  - What is text analysis and how is it used in various fields?
  - What are the three fundamental areas that this textbook aims to develop knowledge and skills in?
  - What are the benefits of learning text analysis beyond just improving data literacy, research skills, and programming skills?
  - How does the structure of the textbook lead to the development of knowledge and skills in data literacy, research skills, and programming skills?
  - How does text analysis contribute to our understanding of human language and its role in our everyday lives?

### Technical exercises

- Create various data types in R (e.g. vectors, matrices, data frames, lists, etc.) and explore their properties.
- 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.
- Fork a quantitative text analysis project on GitHub and clone it to your local machine. Run the code and explore the results.
- ...
- Explore the following resources and identify a quantitative text analysis project. Rpubs<sup>a</sup>, GitHub<sup>b</sup>, DataCamp<sup>c</sup>, Kaggle<sup>d</sup>, R-bloggers<sup>e</sup>.
- The repository “Text Mining and Sentiment Analysis of Twitter Data” located at <https://github.com/anantavijay/Text-Mining-and>

Sentiment-Analysis-of-Twitter-Data contains a real-world quantitative text analysis study with reproducible code in R. The repository contains code for sentiment analysis of Twitter data and for text-mining techniques such as tokenization, stemming, and sentiment analysis. The repository also contains an R script that can be used to reproduce the results of the analysis.

---

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

<sup>b</sup><https://github.com>

<sup>c</sup><https://datacamp.com>

<sup>d</sup><https://kaggle.com>

<sup>e</sup><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*

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

— Carly Fiorina

### Outcomes

- Analyze the distinct types of data and compare their differences.
- Define information and describe format it takes.
- Evaluate the importance of documentation in promoting reproducible research.

In this chapter, the groundwork is laid for deriving insights from data by focusing on 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 information, such as organization and transformation, are examined. The ‘tidy’ approach to organizing data is discussed, including structural characteristics and semantic attributes. Moreover, data transformation techniques are introduced, which can enhance the quality and usability of data in subsequent analysis steps. The importance of documentation in quantitative research is emphasized through addressing data origin and data dictionaries. Methods for documenting sources and processes are provided to ensure a comprehensive understanding of extracting valuable insights from data.

### Swirl lesson

**What:** Objects, Packages and functions<sup>a</sup>

**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.

<sup>a</sup><https://github.com/qtalr/swirl>

## 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 the 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 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 form 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 little information 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. 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. To attempt to align, or **balance**, the relative sizes of the samples for the strata is the second population-informed sampling strategy.

### 2.1.3 Corpora

A key feature of a sample is that it is purposely selected. Samples are not simply a collection or set of data from the population. Samples are rigorously selected with an explicit target population in mind. In text analysis, a purposely sampled collection of texts, of the type defined here, is known as a **corpus** (pl. corpora). For this same reason a set of texts or documents which have not been selected along a purposely selected sampling frame is not a corpus rather a collection of documents. The sampling frame, and therefore the populations modeled, in any given corpus most likely will vary and for this reason it is not a safe assumption that any given corpus is equally applicable for any and every research question. Corpus development (*i.e.* language sampling) is purposeful, and the characteristics of the corpus development process should be made explicit through documentation. Therefore vetting a corpus sample for its applicability to a research goal is a key step in that a research must take to ensure the integrity of the research findings.

#### 💡 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:

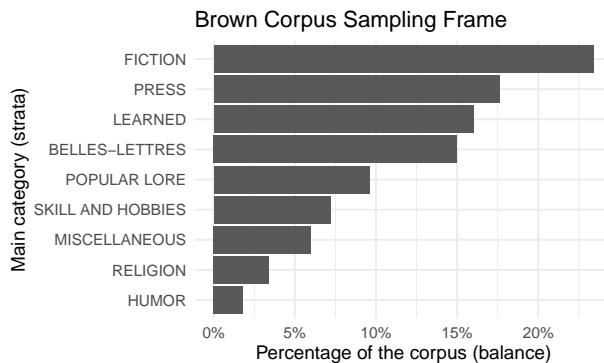


Figure 2.1: Overview of the sampling frame of the Brown Corpus.

Can you determine what language population this corpus aims to represent? What types of research might this corpus support or not support?

### 2.1.3.1 Types

#### *Reference*

With the notion of sampling frames in mind, some corpora are compiled with the aim to be of general purpose (general or **reference corpora**), and some with much more specialized sampling frames (**specialized corpora**). For example, the American National Corpus (ANC)<sup>1</sup> or the British National Corpus (BNC)<sup>2</sup> are corpora which aim to model (represent/ reflect) the general characteristics of a variety of the English language, the former of American English and the later British English. These are ambitious projects, and require significant investments of time in corpus design and then in implementation (and continued development) that are usually undertaken by research teams (Ädel 2020).

#### *Specialized*

Specialized corpora aim to represent more specific populations. The Santa Barbara Corpus of Spoken American English (SBCSAE)<sup>3</sup>, 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 writing<sup>4</sup>, computer-mediated communication (CMC)<sup>5</sup>, language use in specific regions of the world<sup>6</sup>, a country<sup>7</sup>, a region of a country<sup>8</sup>, etc.

#### Consider this

Grieve, Nini, and Guo (2018) compiled a 8.9 billion-word corpus of geo-tagged posts from Twitter between 2013-2014 in the United States. The authors provide a search interface<sup>a</sup> 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?

<sup>1</sup><https://www.anc.org/>

<sup>2</sup><http://www.natcorp.ox.ac.uk/>

<sup>3</sup><https://www.linguistics.ucsb.edu/research/santa-barbara-corpus>

<sup>4</sup><https://www.coventry.ac.uk/research/research-directories/current-projects/2015/british-academic-written-english-corpus-bawc/>

<sup>5</sup><https://www.clarin.eu/resource-families/cmc-corpora>

<sup>6</sup><http://ice-corpora.net/ice/index.html>

<sup>7</sup><https://www.wgtn.ac.nz/lals/resources/corpora-default/corpora-wsc>

<sup>8</sup><https://cesa.arizona.edu>

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

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 OPUS<sup>9</sup>. 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 Brown<sup>10</sup> and LOB<sup>11</sup> corpora.

The aim of the quantitative text researcher is to select the corpus or corpora (plural of corpus) which best aligns 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 language<sup>12</sup>, 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.

### 2.1.3.2 Sources

#### *Published*

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 and language used in natural settings that can be compiled by investigators into a corpus. Many organizations exist around the globe that provide access to corpora in browsable catalogs, or **repositories**. There are repositories dedicated to language research, in general, such as the Language Data Consortium<sup>13</sup> or for specific language domains, such as the language acquisition repository TalkBank<sup>14</sup>. It is always advisable to start looking for the available language data in a repository. The advantage of beginning your

<sup>9</sup><https://opus.nlpl.eu/>

<sup>10</sup><https://ota.bodleian.ox.ac.uk/repository/xmlui/handle/20.500.12024/0402>

<sup>11</sup><https://ota.bodleian.ox.ac.uk/repository/xmlui/handle/20.500.12024/0167>

<sup>12</sup><https://mtsamples.com/index.asp>

<sup>13</sup><https://www.ldc.upenn.edu/>

<sup>14</sup><http://talkbank.org/>

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. A standardized resource many times will be easier to interpret and evaluate for its appropriateness for a particular research project.

Repositories are by no means the only source of 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 (*e.g.* LDC) or via search interfaces (*e.g.* COCA). 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 GitHub<sup>15</sup>, Zenodo<sup>16</sup>, Re3data<sup>17</sup>, OSF<sup>18</sup>, *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.

\*

#### Custom-built

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 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, 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.

The second way to acquire data from the web is through an **Application**

---

<sup>15</sup><https://github.com/>

<sup>16</sup><https://zenodo.org/>

<sup>17</sup><http://www.re3data.org/>

<sup>18</sup><https://osf.io/>

**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.

### Dive deeper

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

### Consider this

Explore some of the resources listed on the qtalrkit compansion site<sup>a</sup> 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?

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

### *Ethical considerations*

Ethical practices in corpus data collection involve adhering to principles and guidelines that ensure researchers' work is used fairly, respectfully, and with integrity. A corpus represents a form of Intellectual Property (IP) and must be treated accordingly. Individual data resources often have their own 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 licenses, like those used by the American National Corpus, Wikipedia, and TalkBank, span from open to more restrictive uses, including requirements for attribution or prohibiting commercial use. 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.

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 is essential to demonstrate compliance with IP laws and ethical guidelines.

Attribution and citation are critical aspects of respecting intellectual property rights in corpus data collection. Researchers should follow established citation

guidelines, such as APA, MLA, or Chicago, to accurately credit original authors. Providing clear and consistent citations helps maintain transparency and ensures that authors receive appropriate recognition for their work. By prioritizing ethical practices, researchers can protect intellectual property rights and uphold the integrity of their research, fostering trust and credibility in their findings.

### 2.1.3.3 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, meta-data 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 the level of linguistic annotation.

The following is the file and folder structure for 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
```

...

---

**Block 2.1** Customers Query

---

```
SELECT * FROM Customers
```

---

Then we query the customers database (Block 2.1).

---

**Block 2.2** Toy corpus structure

---

```
falls between unstructured and structured data. This
↳ covers a wide range of data structures. For example, a
↳ *.csv* file is a tabular file format which is
↳ generally structured with a header row and data rows.
↳ The header row contains the column names and the data
↳ rows contain the values for each column. However, the
↳ data in each column is not strictly defined as part of
↳ the format.
```

---

In this toy 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 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 in 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 ... In either case, it is recommended to open the file in a text editor to inspect the structure of the file content to verify.

*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**. 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 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:

### Example 2.3. MASC data frame

```
# A tibble: 11 x 8
#> # ... with 8 variables:
#> #   title <chr>, date <dbl>, modality <fct>, domain <dbl>,
#> #   ref_num <dbl>, word <chr>, lemma <chr>, pos <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 Californ~ 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 vibration 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. The values in each column are of the same type (e.g. `<chr>`, `<dbl>`, `<fct>`). This structured format is designed to be easy to query and analyze.

**Semi-structured data** falls between unstructured and structured data. This covers a wide range of data structures. For example, a `.csv` file is a tabular file format which is generally structured with a header row and data rows. The header row contains the column names and the data rows contain the values for each column. However, the data in each column is not strictly defined as part of the format.

#### Example 2.4. MASC CSV

```

title,date,modality,domain,ref_num,word,lemma,pos
Hotel California,2008,Writing,General Fiction,0,>,>,NN
Hotel California,2008,Writing,General Fiction,1,Hotel,hotel,NNP
Hotel California,2008,Writing,General Fiction,2,California,california,NNP
Hotel California,2008,Writing,General Fiction,3,Fact,fact,NNP
Hotel California,2008,Writing,General Fiction,4,:,:,:,
Hotel California,2008,Writing,General Fiction,5,Sound,sound,NNP
Hotel California,2008,Writing,General Fiction,6,is,be,VBZ
Hotel California,2008,Writing,General Fiction,7,a,a,DT
Hotel California,2008,Writing,General Fiction,8,vibration,vibration,NN
Hotel California,2008,Writing,General Fiction,9,,.,.

```

On the other end of semi-structured range, a otherwise plain text file with part-of-speech tags appended to each word is lightly-structured.

#### Example 2.5. MASC POS

```

>/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

```

In the middle fall many file formats including `.xml` and `.json` which are hierarchical data formats.

#### Example 2.6. 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>
<node xml:id="penn-n10">
<link targets="seg-r17"/>
</node>
<a xml:id="penn-N65627" label="tok" ref="penn-n10" as="anc">
<fs>
<f name="base" value="sound"/>
<f name="msd" value="NNP"/>
<f name="string" value="Sound"/>
</fs>
</a>
<node xml:id="penn-n100">
<link targets="seg-r184"/>
</node>
<a xml:id="penn-N65655" label="tok" ref="penn-n100" as="anc">
<fs>
<f name="base" value="that"/>
<f name="msd" value="IN"/>
<f name="string" value="that"/>
</fs>
</a>

```

The format of semi-structured data is often influenced by characteristics of the data or reflect an author's individual preferences. For example, the following is a snippet from a `.utt` file from the SWDA corpus:

...

*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.

---

In corpora with more meta-data, a header may be appended to the top of each running text document or the meta-data may be contained in a separate file with appropriate coding to coordinate meta-data attributes with each text in the corpus.

Meta-data header sample from the Switchboard Dialog Act Corpus<sup>19</sup>.

When meta-data and/ or linguistic annotation increases in complexity it is common to structure each corpus document more explicitly with a markup language such as XML (Extensible Markup Language) or organize relationships between language and meta-data attributes in a database.

XML format for meta-data (and linguistic annotation) from the Brown Corpus<sup>20</sup>.

Although there has been a push towards standardization of corpus formats, most available resources display some degree of idiosyncrasy. Being able to parse the structure of a corpus is a skill that will develop with time. With more experience working with corpora you will become more adept at identifying how the data is stored and whether its content and format will serve the needs of your analysis.

---

<sup>19</sup>

<sup>20</sup>[http://www.nltk.org/nltk\\_data/](http://www.nltk.org/nltk_data/)

## 2.2 Information

Identifying an adequate corpus resource 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 useful and informative format. This is the process of converting a corpus into a **dataset** – a tabular representation of data as information which can be leveraged for analysis.

### 2.2.1 Organization

Data alone is not informative. Only through explicit organization of the data in a way that makes relationships explicit does the 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 text to make the relationships meaningful and useful for analysis.

- curation

For the running text in the Europarle Corpus, we know that there are files which are the source text (original) and files that correspond to the target text (translation). In `?@tbl-structure-europarle` we see that this text has been organized so that there are columns corresponding to the `type` and `sentence` with an additional `sentence_id` column to keep an index of how the sentences are aligned.

#### 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<sup>a</sup> as my personal preference. There are also alternatives<sup>b</sup>. 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.

<sup>a</sup>[https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/names.html#snake\\_case](https://bookdown.org/content/d1e53ac9-28ce-472f-bc2c-f499f18264a3/names.html#snake_case)

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

Other corpus resources are **semi-structured data** –that is, there are some characteristics which are structured, but other which are not.

The Switchboard Dialog Act Corpus is an example of a semi-structured resource. It has meta-data associated with each of the 1,155 conversations in the corpus. In `?@tbl-structure-swda` a language-relevant sub-set of the meta-data is associated with each utterance.

Relatively fewer resources are **structured datasets**. In these cases a high amount of meta-data and/ or linguistic annotation is included in the corpus. The format convention, however, varies from resource to resource. Some of the formats are programming general (.csv, .xml, .json, *etc.*) and others are resource specific (.cha, .utt, .prd, *etc.*). In `?@tbl-structure-brown` the XML version of the Brown Corpus is represented in tabular format. Note that along with other meta-data variables, it also contains a variable with linguistic annotation for grammatical category (`pos` part-of-speech) of each word.

### 2.2.1.1 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 *derived 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 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*.

### 2.2.1.2 Informational values

In a tidy dataset, each column represents a **variable** that measures a specific attribute with varying values. These values can be classified into two primary types: **categorical** and **numeric**. To further refine this classification, we can

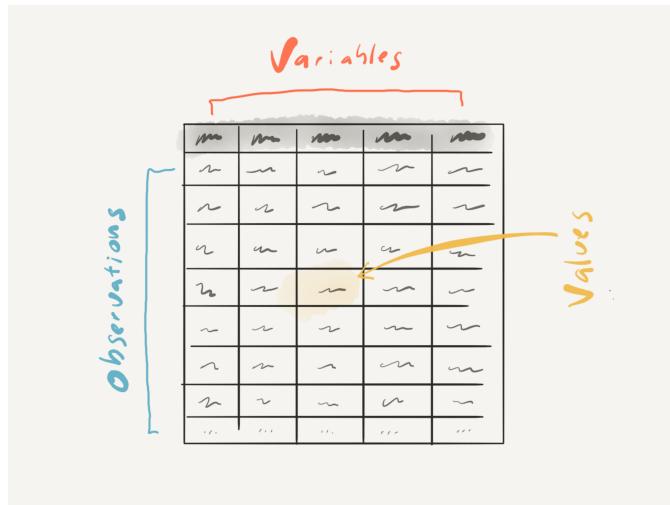


Figure 2.2: Visual summary of the tidy format.

identify four distinct types of values: nominal, ordinal, interval, and ratio. Collectively, these four types are referred to as **levels of measurement**. By categorizing values into either categorical or numeric groups, we can better understand how to approach and analyze the data within each column.

Categorical variables are qualitative measures and can be further broken down into **nominal** and **ordinal**. Numeric variables are quantitative measures and can be further broken down into **interval** and **ratio**. The informational values of these variables are summarized in Figure 2.3.

- Create a drawio diagram for this figure.
  - Provide examples of each of these types of levels of measurement.
- .... examples of levels of measurement ....

#### 2.2.1.3 Semantic value

Semantic value in a tidy dataset is derived from the association of this physical structure along the two dimensions of this rectangular format. First, each column is a variable which reflects measures for a particular attribute that can take on different values. In this way it ‘varies’.

In the Europarl Corpus dataset, in ???, for example, the `type` column measures the type of text, either `Source` or `Target`. Furthermore variables can contain measures which are either qualitative or quantitative, that is category- or scale-based.

Second, each row is an **observation** that contains all of the variables associ-

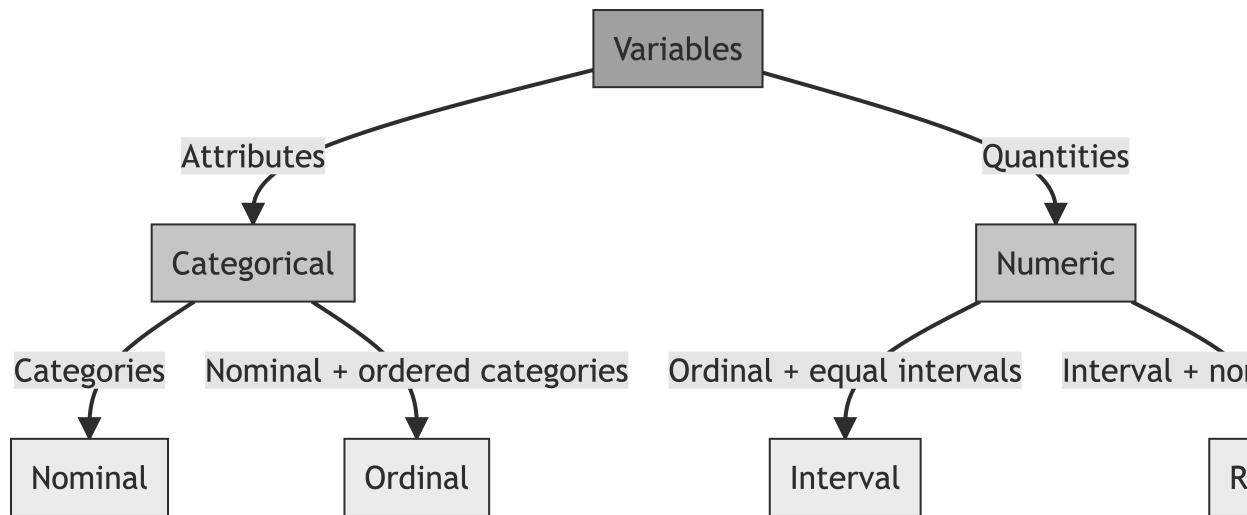


Figure 2.3: Informational values of variables.

ated with the primary **unit of observation**. The primary unit of observation is the variable that is the essential focus of the informational structure. In this same dataset the first observation contains the `type`, `sentence_id`, and the `sentence`. As this dataset is currently structured, the primary unit of investigation is the `sentence` as each of the other variables have measures that characterize each value of `sentence`.

The decision as to what the primary unit of observation is is fundamentally guided by the research question, and therefore highly specific to the particular research project. Say instead we wanted to focus on words instead of sentences. The dataset would need to be transformed such that a new variable (`words`) would be created and each row would index a word in the corpus.

The values for the variables `type` and `sentence_id` maintain the necessary description for each `word` to ensure the required semantic relationships to identify the particular attributes for each word observation. This dataset may seem redundant in that the values for `type` and `sentence_id` are repeated numerous times but this ‘redundancy’ makes the relationship between each variable associated with the primary unit of investigation explicit. This format makes a tidy dataset a versatile format for researchers to conduct analyses in a powerful and flexible way, as we will see throughout this textbook.

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. Data can be organized in many ways which do not make relationships between variables and observations explicit.

### 💡 Consider this

All tabular data does not have the ‘tidy’ format that I have described here. 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?

### 💡 Consider this

Consider the following dataset:

What is the primary unit of observation in this dataset? What are the variables? What are the values for each variable? What is the semantic value of this dataset?

## 2.2.2 Transformation

At this point have introduced the first step in data curation in which the original data is converted into a relational dataset (derived dataset) and highlighted the importance of this informational structure for setting the stage for data analysis. However, the primary derived dataset is often not the final organizational step before proceeding to statistical analysis. Many times, if not always, the derived dataset requires some manipulation or transformation to prepare the dataset for the specific analysis approach to be taken. This is another level of human intervention and informational organization, and therefore another step forward in our journey from data to insight and as such a step up in the DIKI hierarchy. Common types of transformations include cleaning variables (normalization), separating or eliminating variables (recoding), creating new variables (generation), or incorporating others datasets which integrate with the existing variables (merging). The results of these transformations build on and manipulate the derived dataset and produce an *analysis dataset*. Let’s now turn to provide a select set of examples of each of these transformations using the datasets we have introduced in this chapter.

### 2.2.2.1 Normalization

The process of normalization aims to *sanitize* the values within a variable or set of variables. This may include removing whitespace, punctuation, numerals, or special characters or substituting uppercase for lowercase characters, numerals for word versions, acronyms for their full forms, irregular or incorrect spelling for accepted forms, or removing common words (**stopwords**), etc.

On inspecting the Europarl dataset ([?@tbl-structure-europarl](#)) we will

see that there are sentence lines which do not represent actual parliament speeches. In `?@tbl-normalize-non-speech-identify-europarl` we see these lines.

A research project aiming to analyze speech would want to normalize this dataset removing these lines, as seen in `?@tbl-normalize-non-speech-remove-europarl`.

Another feature of this dataset which may require attention is the fact that the English lines include whitespace between possessive nouns.

This may affect another transformation process or subsequent analysis, so it may be a good idea to normalize these forms by removing the extra whitespace.

A final normalization case scenario involves changing converting all the text to lowercase. If the goal for the research is to count words at some point the fact that a word starts a sentence and by convention the first letter is capitalized will result distinct counts for words that are in essence the same (*i.e.* “In” vs. “in”).

Note that lowercasing text, and normalization steps in general, can come at a cost. For example, lowercasing the Europarl dataset sentences means we lose potentially valuable information; namely the ability to identify proper names (*i.e.* “Mr Kumar Ponnambalam”) and titles (*i.e.* “European Parliament”) directly from the orthographic forms. There are, however, transformation steps that can be applied which aim to recover ‘lost’ information in situations such as this and others.

### 2.2.2.2 Recoding

The process of recoding aims to *recast* the values of a variable or set of variables to a new variable or set of variables to enable more direct access. This may include extracting values from a variable, stemming or lemmatization of words, tokenization of linguistic forms (words, ngrams, sentences, *etc.*), calculating the lengths of linguistic units, removing variables that will not be used in the analysis, *etc.*

Words that we intuitively associate with a ‘base’ word can take many forms in language use. For example the word forms ‘investigation’, ‘investigation’, ‘investigate’, ‘investigated’, *etc.* are intuitively linked. There are two common methods that can be applied to create a new variable to facilitate the identification of these associations. The first is stemming. **Stemming** is a rule-based heuristic to reduce word forms to their stem or root form.

A few things to note here. First there are a number of stemming algorithms both for individual languages and distinct languages<sup>21</sup>. Second not all words can be stemmed as they do not have alternate morphological forms (*i.e.* “The”, “of”, *etc.*). This generally related to the distinction between closed-class (articles, prepositions, conjunctions, *etc.*) and open-class (nouns, verbs, adjectives, *etc.*) grammatical categories. Third the stem generated for those words that can be stemmed result in forms that are not words themselves. Nonetheless, stems can be very useful for more easily extracting a set of related word forms.

As an example, let’s identify all the word forms for the stem ‘investig’.

We can see from the results in `?@tbl-recoding-stemming-brown-search` that searching for `word_stems` that match ‘investig’ returns a set of stem-related forms. But it is worth noting that these forms cut across a number of grammatical categories. If instead you want to draw a distinction between grammatical categories, we can apply **lemmatization**. This process is distinct from stemming in two important ways: (1) inflectional forms are grouped by grammatical category and (2) the resulting forms are lemmas or ‘base’ forms of words.

To appreciate the difference between stemming and lemmatization, let’s compare a filter for `word_lemmas` which match ‘investigation’.

Only lemma forms of ‘investigate’ which are nouns appear. Let’s run a similar search but for the lemma ‘be’.

Again only words of the same grammatical category are returned. In this case the verb ‘be’ has many more inflectional forms than ‘investigate’.

Another form of recoding is to detect a pattern in the values of an existing variable and create a new variable whose values are the extracted pattern or register that the pattern occurs and/ or how many times it occurs. As an example, let’s count the number of disfluencies (‘uh’ or ‘um’) that occur in each utterance in `utterance_text` from the Switchboard Dialog Act Corpus. *Note I’ve simplified the dataset dropping the non-relevant variables for this example.*

One of the most common forms of recoding in text analysis is tokenization. **Tokenization** is the process of recasting the text into smaller linguistic units. When working with text that has not been linguistically annotated, the most feasible linguistic tokens are words, ngrams, and sentences. While word and sentence tokens are easily understandable, ngram tokens need some explana-

---

<sup>21</sup><https://snowballstem.org/algorithms/>

tion. An **ngram** is a sequence of either characters or words where  $n$  is the length of this sequence. The ngram sequences are drawn incrementally, so the bigrams (two-word sequences) for the sentence “This is an input sentence.” are:

this is, is an, an input, input sentence

We’ve already seen word tokenization exemplified with the Europarle Corpus in subsection Structure in `?@tbl-tidy-words-europarle`, so let’s create (word) bigram tokens for this corpus.

As I just mentioned, ngrams sequences can be formed of characters as well. Here are character trigram (three-character) sequences.

### 2.2.2.3 Generation

The process of generation aims to *augment* a variable or set of variables. In essence this aims to make implicit attributes explicit to that they are directly accessible. This often targeted at the automatic generation of linguistic annotations such as grammatical category (part-of-speech) or syntactic structure.

In the examples below I’ve added linguistic annotation to a target (English) and source (Spanish) example sentence from the Europarle Parallel Corpus. First, note the variables that are added to our dataset that correspond to grammatical category. In addition to the `type` and `sentence_id` we have an assortment of variables which replace the `sentence` variable. As part of the process of annotation the input text to be annotated `sentence` is tokenized `token` and indexed `token_id`. Then `upos` contains the Universal Part of Speech tags<sup>22</sup>, and a detailed list of features is included in `feats`. The syntactic annotation is reflected in the `token_id_source` and `syntactic_relation` variables. These variables correspond to the type of syntactic parsing that has been done, in this case Dependency Parsing (using the Universal Dependencies<sup>24</sup> framework). Another common syntactic parsing framework is phrase constituency parsing (Jurafsky and Martin 2020).

Now compare the English example sentence dataset in `?@tbl-generation-europarle-en-example` with the parallel sentence in Spanish. Note that the grammatical features are language specific. For example, Spanish has gender which is apparent when scanning the `feats` variable.

There is much more to explore with linguistic annotation, and syntactic pars-

<sup>22</sup>Descriptions of the UPOS tagset<sup>23</sup>

<sup>24</sup><https://universaldependencies.org/>

ing in particular, but at this point it will suffice to note that it is possible to augment a dataset with grammatical information automatically.

There are strengths and shortcomings with automatic linguistic annotation that a research should be aware of. First, automatic linguistic annotation provides quick access to rich and highly reliable linguistic information for a large number of languages. However, part-of-speech taggers and syntactic parsers are not magic. They are resources that are built by training a computational algorithm to recognize patterns in manually annotated datasets producing a language model. This model is then used to predict the linguistic annotations for new language (as we just did in the previous examples). The shortcomings of automatic linguistic annotation is first, not all languages have trained language models and second, the data used to train the model inevitably reflect a particular variety, register, modality, *etc*. The accuracy of the linguistic annotation is highly dependent on alignment between the language sampling frame of the trained data and the language data to be automatically annotated. Many (most) of the language models available for automatic linguistic annotation are based on language that is most readily available and for most languages this has traditionally been newswire text. It is important to be aware of these characteristics when using linguistic annotation tools.

#### 2.2.2.4 Merging

The process of merging aims to *join* a variable or set of variables with another variable or set of variables from another dataset. The option to merge two (or more) datasets requires that there is a shared variable that indexes and aligns the datasets.

To provide an example let's look at the Switchboard Dialog Act Corpus. Our existing, disfluency recoded, version includes the following variables.

It turns out that on the corpus website<sup>25</sup> a number of meta-data files are available, including files pertaining to speakers and the topics of the conversations.

The speaker meta-data for this corpus is in a the `caller_tab.csv` file and contains a `speaker_id` variable which corresponds to each speaker in the corpus and other potentially relevant variables for a language research project including `sex`, `birth_year`, `dialect_area`, and `education`.

Since both datasets contain a shared index, `speaker_id` we can merge these two datasets. The result is found in `?@tbl-merging-swda-speaker-added`.

---

<sup>25</sup><https://catalog.ldc.upenn.edu/docs/LDC97S62/>

In this example case the dataset that was merged was already in a structured format (.csv). Many corpus resources contain meta-data in stand-off files that are structured.

In some cases a researcher would like to merge information that does not already accompany the corpus resource. This is possible as long as a dataset can be created that contains a variable that is shared. Without a shared variable to index the datasets the merge cannot take place.

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 derived dataset and the ultimate goals for the analysis dataset.

### ⚠ Tip

Note in some cases we may convert our tidy tabular dataset to other data formats that may be required for some particular statistic approaches but at all times the relationship between the variables should be maintained in line with our research purpose. We will touch on examples of other types of data formats (*e.g.* Corpus and Document-Term Matrix (DTM) objects in R) when we dive into particular statistical approaches that require them later in the textbook.

---

## 2.3 Documentation

### 2.3.1 Data origin information

### 2.3.2 Data dictionaries

As we have seen in this chapter that acquiring 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 the creation/ selection of the corpus data, the description of the variables chosen from the corpus for the derived dataset, and the description of the variables created from the derived dataset for the analysis dataset.

For an existing corpus sample acquired from a repository (*e.g.* Switchboard Dialog Act Corpus<sup>26</sup>, Language Data Consortium), a research group (*e.g.* CEDEL<sup>27</sup>), or an individual researcher (*e.g.* SMS Spam Collection<sup>28</sup>), there is often documentation provided describing key attributes of the resource. This documentation should be included with the acquisition of the corpus and added to the research project. For a corpus that a researcher compiles themselves, they will need to generate this documentation.

The curation and transformation steps conducted on the original corpus data to produce the datasets should also be documented. The steps themselves can be included in the programming scripts as code comments (or in prose if using a literate programming strategy (*e.g.* R Markdown)). The structure of each resulting dataset should include what is called a **data dictionary**. This is a table which includes the variable names, the values they contain, and a short prose description of each variable (*e.g.* ACTIV-ES Corpus<sup>29</sup>).

---

## Summary

In this chapter we have focused on data and information –the first two components of DIKI Hierarchy. This process is visualized in Figure 2.4.

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 derived dataset which adopts the tidy dataset format where each column is a

---

<sup>26</sup><https://catalog.ldc.upenn.edu/LDC97S62>

<sup>27</sup><http://cedel2.learnercorpora.com/>

<sup>28</sup><https://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>

<sup>29</sup><https://osf.io/9jafz/>

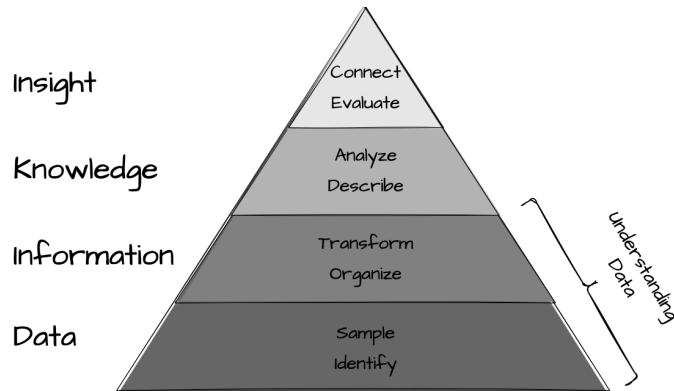


Figure 2.4: Understanding data: visual summary

variable, each row is an observation, and the intersection of columns and rows contain values. This derived dataset serves to establish the base informational relationships from which your research will stem.

The derived dataset will most likely require transformations including normalization, recoding, generation, and/ or merging to enhance the usefulness of the information to analysis. An analysis dataset is the result of this process.

Finally, documentation should be implemented at each stage of the analysis project process. Employing a programmatic approach establishes documentation of the implementation steps but the motivation behind the decisions taken and the content of the corpus data and datasets generated also need documentation to ensure transparent and reproducible research.

## Activities

- Add summary of the purpose, goals, outcomes of these activities

### \_recipe

**What:** Reading, inspecting, and writing data<sup>a</sup>

**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.

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

**Lab**

**What:** Reading, inspecting, and writing data<sup>a</sup>

**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.

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

## Questions

**Conceptual questions**

1. What is the difference between a population and a sample in linguistic research?
2. Why is it important to vet a corpus before using it in a research project?
3. What is a derived dataset in the context of linguistic research?
4. How does the tidy table format help with linguistic analysis?
5. What kinds of transformations may be performed on a derived dataset to enhance its usefulness for analysis?
6. What is an analysis dataset and why is it important in linguistic research?
7. Why is documentation important in the process of conducting linguistic analysis?
8. How does a programmatic approach enhance documentation in linguistic research?
9. Why is it important to document the motivation behind decisions taken in linguistic analysis projects?
10. How does documenting the corpus data and generated datasets contribute to transparent and reproducible research in linguistics?

To add:

- What is the difference between a variable and a value?
- Why is it important to identify the levels of measurement of variables in a dataset?
- What is the difference between the unit of analysis and the unit of observation?

- How does the unit of analysis and the unit of observation relate to the levels of measurement of variables in a dataset?
- In what ways can a dataset be considered a model of the world?
- To what extent does the sample represent the population? What implications does this have for the generalizability of the results of a research project?

**Technical questions**

1. Creating a Sample Corpus:
2. Writing a Corpus Documentation
3. Converting a Corpus to a Derived Dataset:
4. Writing a Data Dictionary
5. Transforming a Derived Dataset:
6. Merging Datasets:
7. Writing a dataset to disk
8. Consider (an example dataset) and its data dictionary, write a script to read the dataset, inspect it, and write it to disk.
9. Consider a dataset and its data dictionary what appears to be the unit of analysis and the unit of observation?



# 3

## *Approaching analysis*

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

— H.G. Wells

### Keys

- What is the role of statistics in data analysis?
- What is the importance of descriptive assessment in data analysis?
- In what ways are the main approaches to data analysis similar and different?

In this chapter I will build on the notions of data and information from the previous chapter. The aim of statistics in quantitative analysis is to uncover patterns in datasets. Thus statistics is aimed at deriving knowledge from information, the next step in the DIKI Hierarchy Figure 1. Where the creation of information from data involves human intervention and conscious decisions, as we have seen, deriving knowledge from information involves even more conscious subjective decisions on what information to assess, and what method to select to interrogate the information, and ultimately how to interpret the findings. The first step is to conduct a descriptive assessment of the information, both at the individual variable level and also between variables, the second is to interrogate the dataset either through exploratory, predictive, or inferential analysis methods, and the third is to interpret and report the findings.

### Interactive programming

**What:** Data visualization<sup>a</sup>

**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.

<sup>a</sup><https://github.com/lin380/swirl>

Table 3.1: First 10 observations of the BELC dataset for demonstration.

participant_id	age_group	sex	num_tokens	num_types	ttr
L02	10-year-olds	female	48	12	0.250
L05	10-year-olds	female	72	15	0.208
L10	10-year-olds	female	144	26	0.181
L11	10-year-olds	female	40	8	0.200
L12	10-year-olds	female	164	23	0.140
L16	10-year-olds	female	52	12	0.231
L22	10-year-olds	female	188	30	0.160
L27	10-year-olds	female	32	8	0.250
L28	10-year-olds	female	336	34	0.101
L29	10-year-olds	female	212	34	0.160

### 3.1 Descriptive assessment

A descriptive assessment of the dataset includes a set of diagnostic measures and tabular and visual summaries which provide researchers a better understanding of the structure of a dataset, prepare the researcher to make decisions about which statistical methods and/ or tests are most appropriate, and to safeguard against false assumptions (missing data, data distributions, etc.). In this section we will *EDIT: first cover the importance of understanding the informational value that variables can represent and then move to use this understanding to approach summarizing individual variables and relationships between variables.*

To ground this discussion I will introduce a new dataset. This dataset is drawn from the Barcelona English Language Corpus (BELC)<sup>1</sup>, which is found in the TalkBank repository<sup>2</sup>. I've selected the "Written composition" task from this corpus which contains writing samples from 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 many (but not all) participants up to four times over the course of seven years. In Table 3.1 I've included the first 10 observations from the dataset which reflects structural and transformational steps I've done so we start with a tidy dataset.

The entire dataset includes 79 observations from 36 participants. Each observation in the BELC dataset corresponds to an individual learner's composition. It includes which participant wrote the composition (`participant_id`), the

<sup>1</sup><https://slabank.talkbank.org/access/English/BELC.html>

<sup>2</sup><http://talkbank.org/>

age group they were part of at the time (`age_group`), their sex (`sex`), and the number of English words they produced (`num_tokens`), the number of unique English words they produced (`num_types`). The final variable (`ttr`) is the calculated ratio of number of unique words (`num_types`) to total words (`num_tokens`) for each composition. This is known as the Type-Token Ratio and it is a standard metric for measuring lexical diversity.

### 3.1.1 Information values

Understanding the informational value, or **level of measurement**, of a variable or set of variables is key to preparing for analysis as it has implications for what visualization techniques and statistical measures we can use to interrogate the dataset. There are two main levels of measurement a variable can take: categorical and continuous. **Categorical variables** reflect class or group values. **Continuous variables** reflect values that are measured along a continuum.

The BELC dataset contains three categorical variables (`participant_id`, `age_group`, and `sex`) and three continuous variables (`num_tokens`, `num_types`, and `ttr`). The categorical variables identify class or group membership; which participant wrote the composition, what age group they were in, and their biological sex. The continuous variables measure attributes that can take a range of values without a fixed limit and the differences between each value are regular. The number of words and number of unique words for each composition can range from 1 to  $n$  and the Type-Token Ratio being derived from these two variables is also continuous for the same reason. Furthermore, the differences between the each of values of these measures is on a defined interval, so for example a composition which has a word count (`num_tokens`) of 40 is exactly two times as large as a composition with a word count of 20.

The distinction between categorical and continuous levels of measurement, as mentioned above, are the main two and for some statistical approaches the only distinction that needs to be made to conduct an analysis. However, categorical and continuous can each be broken down into subcategories and for some descriptive and analytic purposes these distinctions are important. For categorical variables a distinction can be made between variables in which there is a structured relationship between the values and those in which there is not. *Nominal variables* contain values which are labels denoting the membership in a class in which there is no relationship between the labels. *Ordinal variables* also contain labels of classes, but in contrast to nominal variables, there is a relationship between the classes, namely one in which there is a precedence relationship or order. With this in mind, our categorical variables be sub-classified. There is no order between the values of `participant_id` and `sex` and they are therefore nominal whereas the values of `age_group` are ordered, each value refers to a sequential age group, and therefore it is ordinal.

Turning to continuous variables, another subdivision can be made which hinges on the existence of a non-arbitrary zero or not. *Interval variables* contain values in which the difference between the values is regular and defined, but the measure has an arbitrary zero value. A typical example of an interval variable is temperature measurements on the Fahrenheit scale. A value of 0 on this scale does not mean there is 0 temperature. *Ratio variables* have all the properties of interval variables but also include a non-arbitrary definition of zero. All of the continuous variables in the BELC dataset (`num_tokens`, `num_types`, and `ttr`) are ratio variables as a value of 0 would indicate the lack of this attribute.

An hierarchical overview of the relationship between the two main and four sub-types of levels of measurement appear in Figure 3.1.

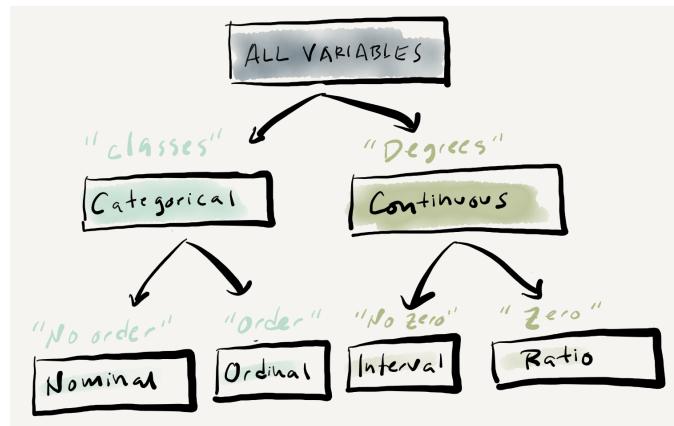


Figure 3.1: Levels of measurement graphic representation.

A few notes of practical importance; First, the distinction between interval and ratio variables is often not applicable in text analysis and therefore often treated together as continuous variables. Second, the distinction between ordinal and interval/continuous variables is not as clear cut as it may seem. Both variables contain values which have an ordered relationship. By definition the values of an ordinal variable do not reflect regular intervals between the units of measurement. But in practice interval/continuous variables with a defined number of values (say from a Likert scale used on a survey) may be treated as an ordinal variable as they may be better understood as reflecting class membership. Third, all continuous variables can be converted to categorical variables, but the reverse is not true. We could, for example, define a criterion for binning the word counts in `num_tokens` for each composition into ordered classes such as “low”, “mid”, “high”. On the other hand, `sex` (as it has been measured here) cannot take intermediate values on a unfixed range. The upshot is that variables can be down-typed but not up-typed. In most cases it is preferred to treat continuous variables as such, if the nature of the variable

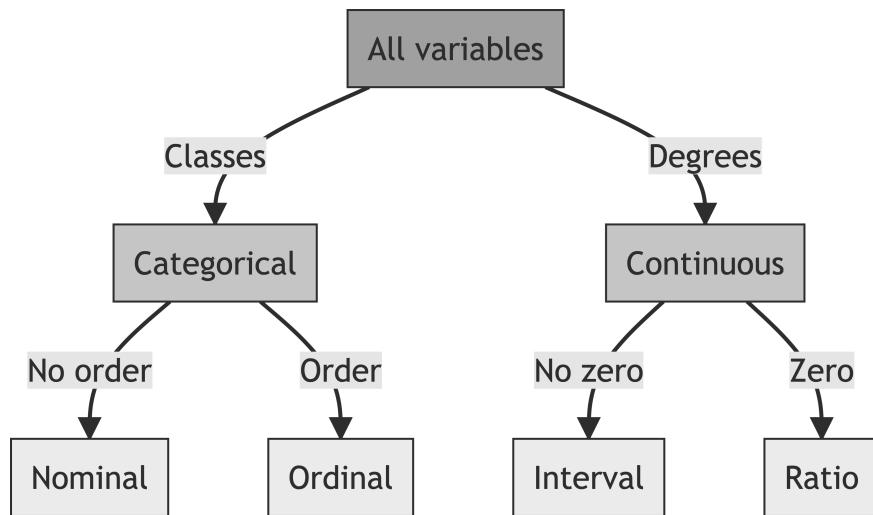


Figure 3.2: Levels of measurement graphic representation.

permits it, as the down-typing of continuous data to categorical data results in a loss of information –which will result in a loss of information and hence statistical power which may lead to results that obscure meaningful patterns in the data (R. Harald Baayen 2004).

### 3.1.2 Summaries

It is always key to gain insight into shape of the information through numeric, tabular and/ or visual summaries before jumping in to analytic statistical approaches. The most appropriate form of summarizing information will depend on the number and informational value(s) of our target variables. To get a sense of how this looks, let's continue to work with the BELC dataset and pose different questions to the data with an eye towards seeing how various combinations of variables are descriptively explored.

#### 3.1.2.1 Single variables

The way to statistically summarize a variable into a single measure is to derive a **measure of central tendency**. For a continuous variable the most common measure is the (arithmetic) *mean*, or average, which is simply the sum of all the values divided by the number of values. As a measure of central tendency, however, the mean can be less-than-reliable as it is sensitive to outliers which is to say that data points in the variable that are extreme

Table 3.2: Central tendency and dispersion measures for the continuous variables in the BELC dataset.

skim_variable	mean	sd	p0	p25	p50	p75	p100	iqr
num_tokens	264.911	175.611	4.000	116.000	220.00	360.000	740.00	244.000
num_types	40.253	22.801	1.000	22.000	38.00	54.000	97.00	32.000
ttr	0.167	0.032	0.101	0.144	0.16	0.182	0.25	0.039

relative to the overall distribution of the other values in the variable affect the value of the mean depending on how extreme the deviate. One way to assess the effects of outliers is to calculate a **measure of dispersion**. The most common of these is the *standard deviation* which estimates the average amount of variability between the values in a continuous variable. Another way to assess, or rather side-step, outliers is to calculate another measure of central tendency, the *median*. A median is calculated by sorting all the values in the variable and then selecting the value which falls in the middle of all the other values. A median is less sensitive to outliers as extreme values (if there are few) only indirectly affect the selection of the middle value. Another measure of dispersion is to calculate quantiles. A *quantile* slices the data in four percentile ranges providing a five value numeric summary of the spread of the values in a continuous variable. The spread between the first and third quantile is known as the Interquartile Range (IQR) and is also used as a single statistic to summarize variability between values in a continuous variable.

Below is a list of central tendency and dispersion scores for the continuous variables in the BELC dataset Table 3.2.

 Tip

The descriptive statistics returned above were generated by the `skimr` package.

In the above summary, we see the mean, standard deviation (sd), and the quantiles (the five-number summary, p0, p25, p50, p75, and p100). The middle quantile (p50) is the median and the IQR is listed last.

These are important measures for assessing the central tendency and dispersion and will be useful for reporting purposes, but to get a better feel of how a variable is distributed, nothing beats a visual summary. A boxplot graphically summarizes many of these metrics. In Figure 3.3 we see the same three continuous variables, but now in graphical form.

In a boxplot, the bold line is the median. The surrounding box around the median is the interquartile range. The extending lines above and below the IQR mark the largest and lowest value that is within 1.5 times either the

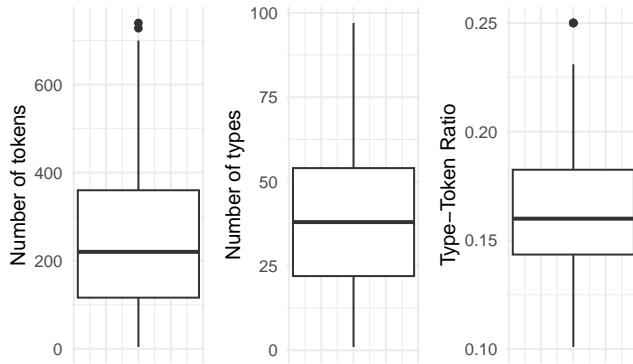


Figure 3.3: Boxplots for each of the continuous variables in the BELC dataset.

3rd (top of the box) or 1st (bottom of the box). Any values that fall outside, above or below, the extending lines are considered statistical outliers and are marked as dots.<sup>3</sup>

Boxplots provide a robust and visually intuitive way of assessing central tendency and variability in a continuous variable but this type of plot can be complemented by looking at the overall distribution of the values in terms of their frequencies. A histogram provides a visualization of the frequency (and density in this case with the blue overlay) of the values across a continuous variable binned at regular intervals.

In Figure 3.4 I've plotted histograms in the top row and density plots in the bottom row for the same three continuous variables from the BELC dataset.

#### Histograms and Boxplots

A histogram is a graphical representation of the frequency distribution of a dataset. It is created by dividing the data into intervals or “bins” and counting the number of data points that fall into each bin. The height of the bars in a histogram represents the frequency or count of data points in each bin. The shape of the histogram can give us insights into the distribution's central tendency, spread, and skewness.

A boxplot, on the other hand, is a graphical representation of the five-number summary of a dataset: the minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. It provides a visual summary of the dataset's central tendency, dispersion, and potential outliers. The

---

<sup>3</sup>Note that each of these three variables are to be considered separately here (vertically). Later we will see the use of boxplots to compare a continuous variable across levels of a categorical variable (horizontally).

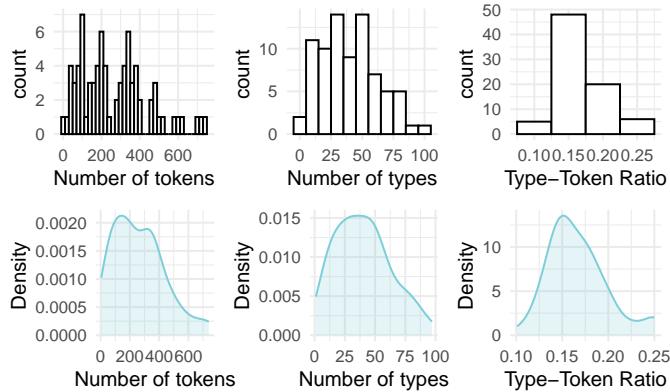


Figure 3.4: Histograms and density plots for the continuous variables in the BELC dataset.

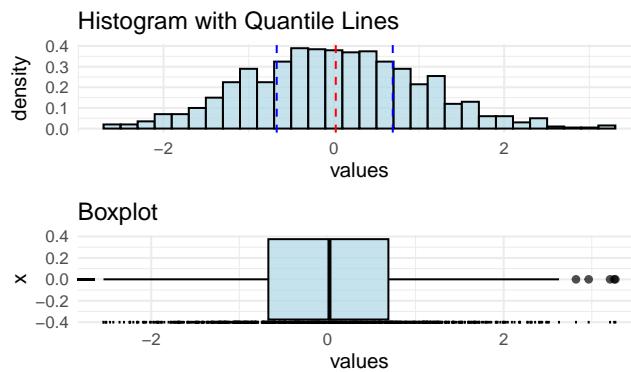


Figure 3.5: Histogram and boxplot for a simulated normal distribution.

box in a boxplot represents the interquartile range (IQR), which contains the middle 50% of the data. The whiskers extend from the box to the minimum and maximum values within  $1.5 * \text{IQR}$ , and any data points beyond the whiskers are considered potential outliers.

Relating histograms and boxplots:

- Both histograms and boxplots provide information about the central tendency and spread of the data. The peak(s) in a histogram can give us a sense of where the data is centered, similar to the median in a boxplot. The width of the histogram can indicate the spread of the data, akin to the IQR in a boxplot.
- The shape of a histogram can provide insights into the skewness of a

dataset. A histogram that is symmetric with a single peak can correspond to a roughly symmetric boxplot. A positively skewed histogram (with a long tail to the right) can correspond to a boxplot where  $Q_3 - Q_2 > Q_2 - Q_1$ , and a negatively skewed histogram (with a long tail to the left) can correspond to a boxplot where  $Q_2 - Q_1 > Q_3 - Q_2$ .

- While histograms focus on the frequency distribution of data points, boxplots focus on the data's quartiles and potential outliers. Both types of plots can be used in tandem to get a more comprehensive understanding of a dataset's distribution.

In summary, histograms and boxplots are related in that they both offer graphical representations of data distributions. They provide different perspectives on the data's central tendency, spread, and skewness, and when used together, they can offer a more complete understanding of a dataset.

Histograms provide insight into the distribution of the data. For our three continuous variables, the distributions happen not to be too strikingly distinct. They are, however, not the same either. When we explore continuous variables with histograms we are often trying to assess whether there is skew or not. There are three general types of skew, visualized in Figure 3.6.

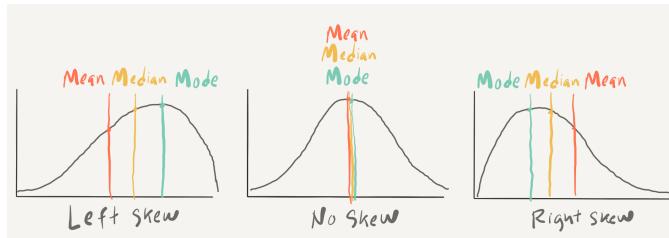


Figure 3.6: Examples of skew types in density plots.

In histograms/ density plots in which the distribution is either left or right, the median and the mean are not aligned. The *mode*, which indicates the most frequent value in the variable is also not aligned with the other two measures. In a left-skewed distribution the mean will be to the left of the median which is left of the mode whereas in a right-skewed distribution the opposite occurs. In a distribution with absolutely no skew these three measures are the same. In practice these measures rarely align perfectly but it is very typical for these three measures to approximate alignment. It is common enough that this distribution is called the Normal Distribution<sup>4</sup> as it is very common in real-world data.

Another and potentially more informative way to inspect the normality of a

<sup>4</sup>formally known as a Gaussian Distribution

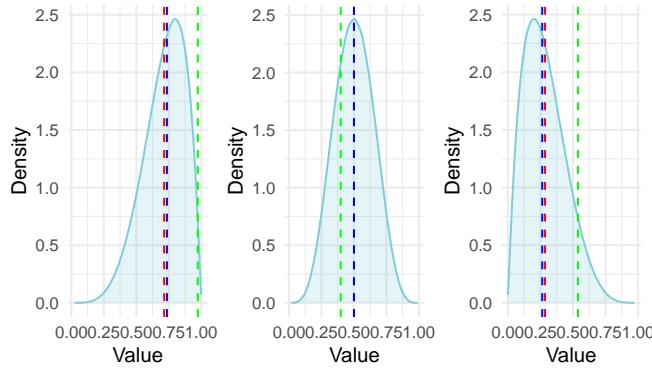


Figure 3.7: Examples of skew types in density plots.

distribution is to create Quantile-Quantile plots (QQ Plot). In Figure 3.8 I've created QQ plots for our three continuous variables. The line in each plot is the normal distribution and the more points that fall off of this line, the less likely that the distribution is normal.

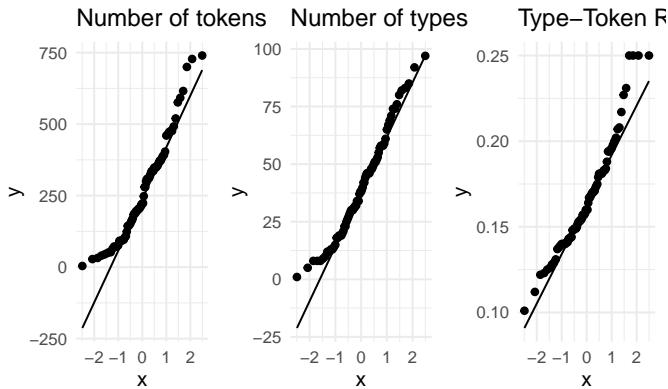


Figure 3.8: QQ Plots for the continuous variables in the BELC dataset.

A visual inspection can often be enough to detect non-normality, but in cases which visually approximate the normal distribution (such as these) we can perform the Shapiro-Wilk test of normality. This is an inferential test that compares a variable's distribution to the normal distribution. The likelihood that the distribution differs from the normal distribution is reflected in a *p*-value. A *p*-value below the .05 threshold suggests the distribution is non-normal. In Table 3.3 we see that given this criterion only the distribution of `num_types` is normally distributed.

Table 3.3: Results from Shapiro-Wilk test of normality for continuous variables in the BELC dataset.

variable	statistic	p_value
Number of tokens	0.942	0.001
Number of types	0.970	0.058
Type-Token Ratio	0.948	0.003

Downstream in the analytic analysis, the distribution of continuous variables will need to be taken into account for certain statistical tests. Tests that assume ‘normality’ are parametric tests, those that do not are non-parametric. Distributions which approximate the normal distribution can sometimes be transformed to conform to the normal distribution either by outlier trimming or through statistical procedures (e.g. square root, log, or inverse transformation), if necessary. At this stage, however, the most important thing is to recognize whether the distributions approximate or wildly diverge from the normal distribution.

Before we leave continuous variables, let’s consider another approach for visually summarizing a single continuous variable. The Empirical Cumulative Distribution Function, or *ECDF*, is a summary of the cumulative proportion of each of the values of a continuous variable over the domain of possible values. An ECDF plot can be useful in determining what proportion of the values fall above or below a certain percentage of the data.

In Figure 3.9 we see ECDF plots for our three continuous variables.

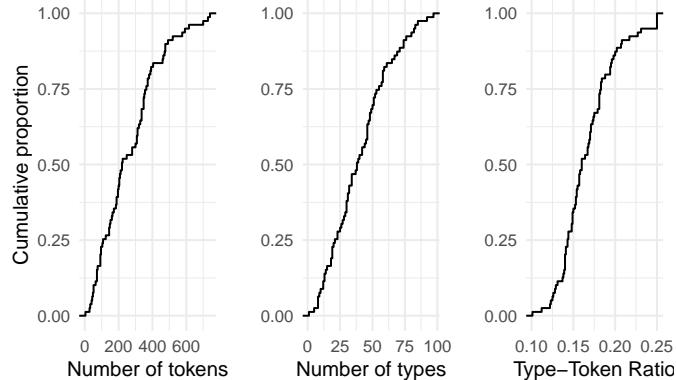


Figure 3.9: ECDF plots for the continuous variables in the BELC dataset.

Take, for example, the number of tokens (`num_tokens`) per composition. The ECDF plot tells us that 50% of the values in this variable are 56 words or

less. In the three variables plotted, the cumulative growth is quite steady. In some cases it is not. When it is not, an ECDF goes a long way to provide us a glimpse into key bends in the proportions of values in a variable.

Now let's turn to the descriptive assessment of categorical variables. For categorical variables, central tendency can be calculated as well but only a subset of measures given the reduced informational value of categorical variables. For nominal variables where there is no relationship between the levels the central tendency is simply the mode. The levels of ordinal variables, however, are relational and therefore the median, in addition to the mode, can also be used as a measure of central tendency. Note that a variable with one mode is unimodal, two modes, bimodal, and in variables that have two or more modes multimodal.

### Tip

To get numeric value of the median for an ordinal variable the levels of the variable will need to be numeric as well. Non-numeric levels can be recoded to numeric for this purpose if necessary.

Below is a list of the central tendency metrics for the categorical variables in the BELC dataset.

#### Variable type: factor

skim_variable	ordered	n_unique	top_counts
participant_id	FALSE	36	L05: 3, L10: 3, L11: 3, L12: 3
age_group	TRUE	4	10-: 24, 16-: 24, 12-: 16, 17-: 15
sex	FALSE	2	fem: 48, mal: 31

In practice when a categorical variable has few levels it is common to simply summarize the counts of each level in a table to get an overview of the variable. With ordinal variables with more numerous levels, the five-score summary (quantiles) can be useful to summarize the distribution. In contrast to continuous variables where a graphical representation is very helpful to get perspective on the shape of the distribution of the values, the exploration of single categorical variables is rarely enhanced by plots.

#### 3.1.2.2 Multiple variables

In addition to the single variable summaries (univariate), it is very useful to understand how two (bivariate) or more variables (multivariate) are related to add to our understanding of the shape of the relationships in the dataset. Just as with univariate summaries, the informational values of the variables frame our approach.

To explore the relationship between two continuous variables we can statistically summarize a relationship with a **coefficient of correlation** which is a

measure of **effect size** between continuous variables. If the continuous variables approximate the normal distribution *Pearson's r* is used, if not *Kendall's tau* is the appropriate measure. A correlation coefficient ranges from -1 to 1 where 0 is no correlation and -1 or 1 is perfect correlation (either negative or positive). Let's assess the correlation coefficient for the variables `num_tokens` and `ttr`. Since these variables are not normally distributed, we use Kendall's tau. Using this measure the correlation coefficient is  $-0.565$  suggesting there is a correlation, but not a particularly strong one.

Correlation measures are important for reporting but to really appreciate a relationship it is best to graphically represent the variables in a *scatterplot*. In Figure 3.10 we see the relationship between `num_tokens` and `ttr`.

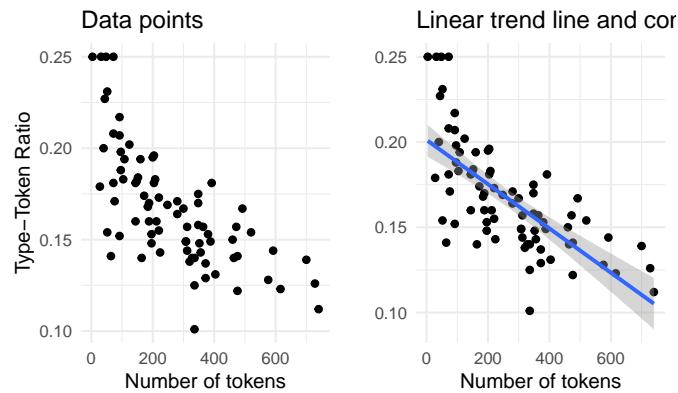


Figure 3.10: Scatterplot...

In both plots `ttr` is on the y-axis and `num_tokens` on the x-axis. The points correspond to the intersection between these variables for each single observation. In the left pane only the points are represented. Visually (and given the correlation coefficient) we can see that there is a negative relationship between the number of tokens and the Type-Token ratio: in other words, the more tokens a composition has the lower the Type-Token Ratio. In this case this trend is quite apparent, but in other cases it may not be. To provide an additional visual cue a trend line is often added to a scatterplot. In the right pane I've added a linear trend line. This line demarcates the optimal central tendency across the relationship, assuming a linear relationship. The steeper the line, or slope, the more likely the correlation is strong. The band, or ribbon, around this trend line indicates the **confidence interval** which means that real central tendency could fall anywhere within this space. The wider the ribbon, the larger the variation between the observations. In this case we see that the ribbon widens when the number of tokens is either low or high. This means that the trend line could be potentially be drawn either steeper (more strongly correlated) or flatter (less strongly correlated).

 Tip

In plots comparing two or more variables, the choice of which variable to plot on the x- and y-axis is contingent on the research question and/or the statistical approach. The language varies between statistical approaches: in inferential methods the x-axis is used to plot what is known as the dependent variable and the y-axis an independent variable. In predictive methods the dependent variable is known as the outcome and the independent variable a predictor. Exploratory methods do not draw distinctions between variables along these lines so the choice between which variable to plot along the x- and y-axis is often arbitrary.

Let's add another variable to the mix, in this case the categorical variable `sex`, taking our bivariate exploration to a multivariate exploration. Again each point corresponds to an observation where the values for `num_tokens` and `ttr` intersect. But now each of these points is given a color that reflects which level of `sex` it is associated with.

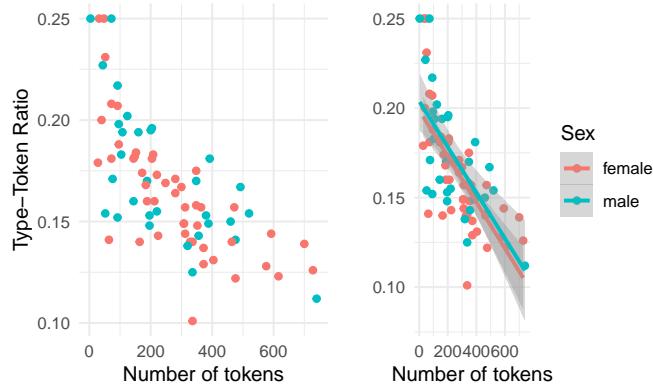


Figure 3.11: Scatterplot visualizing the relationship between `num_tokens` and `ttr`.

In this multivariate case, the scatterplot without the trend line is more difficult to interpret. The trend lines for the levels of `sex` help visually understand the variation of the relationship of `num_tokens` and `ttr` much better. But it is important to note that when there are multiple trend lines there is more than one slope to evaluate. The correlation coefficient can be calculated for each level of `sex` (i.e. ‘male’ and ‘female’) independently but the relationship between the each slope can be visually inspected and provide important information regarding each level’s relative distribution. If the trend lines are parallel (ignoring the ribbons for the moment), as it appears in this case, this suggests that the relationship between the continuous variables is stable

Table 3.4: Summary table for `tokens` by `age_group`.

<code>age_group</code>	<code>mean_num_tokens</code>	<code>sem</code>	<code>ci</code>
10-year-olds	111	14.8	24.3
12-year-olds	230	28.5	46.9
16-year-olds	327	24.6	40.4
17-year-olds	450	51.9	85.4

across the levels of the categorical variable, with males showing more lexical diversity than females declining at a similar rate. If the lines were to cross, or suggest that they would cross at some point, then there would be a potentially important difference between the levels of the categorical variable (known as an interaction). Now let's consider the meaning of the ribbons. Since the ribbons reflect the range in which the real trend line could fall, and these ribbons overlap, the differences between the levels of our categorical variable are likely not distinct. So at a descriptive level, this visual summary would suggest that there are no differences between the relationship between `num_tokens` and `ttr` for the distinct levels of `sex`.

Characterizing the relationship between two continuous variables, as we have seen is either performed through a correlation coefficient metric or visually. The approach for summarizing a bivariate relationship which combines a continuous and categorical variable is distinct. Since a categorical variable is by definition a class-oriented variable, a descriptive evaluation can include a tabular representation, with some type of summary statistic. For example, if we consider the relationship between `num_tokens` and `age_group` we can calculate the mean for `num_tokens` for each level of `age_group`. To provide a metric of dispersion we can include either the standard error of the mean (SEM) and/or the confidence interval (CI).

The SEM is a metric which summarizes variation based on the number of values and the CI, as we have seen, summarizes the potential range of in which the mean may fall given a likelihood criterion (usually the same as the *p*-value, .05).

Because we are assessing a categorical variable in combination with a continuous variable a table is an available visual summary. But as I have said before, a graphic summary is hard to beat. In the following figure (Figure 3.12) a barplot is provided which includes the means of `num_tokens` for each level of `age_group`. The overlaid bars represent the confidence interval for each mean score.

When CI ranges overlap, just as with ribbons in scatterplots, the likelihood that the differences between levels are 'real' is diminished.

To gauge the effect size of this relationship we can use *Spearman's rho* for rank-

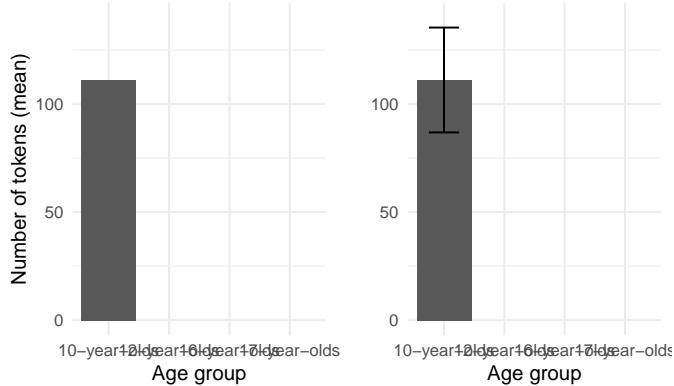


Figure 3.12: Barplot comparing the mean `num_tokens` by `age_group` from the BELC dataset.

based coefficients. The score is 0.708 indicating that the relationship between `age_group` and `num_tokens` is quite strong.<sup>5</sup>

Now, if we want to explore a multivariate relationship and add `sex` to the current descriptive summary, we can create a summary table, but let's jump straight to a barplot.

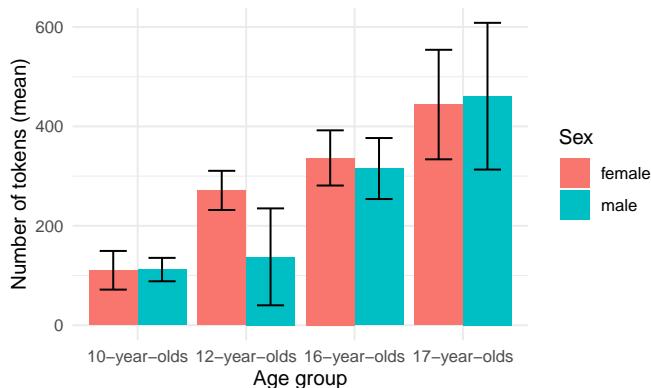


Figure 3.13: Barplot comparing the mean `num_tokens` by `age_group` and `sex` from the BELC dataset.

We see in Figure 3.13 that on the whole, there appears to be a general trend towards more tokens in a composition for more advanced learner levels. However, the non-overlap in CI bars for the '12-year-olds' for the levels of `sex` ('male'

<sup>5</sup>To calculate effect sizes for the difference between two means, *Cohen's d* is used.

and ‘female’) suggest that 12-year-old females may produce more tokens per composition than males – a potential divergence from the overall trend.

Barplots are a familiar and common visualization for summaries of continuous variables across levels of categorical variables, but a boxplot is another useful visualization of this type of relationship.

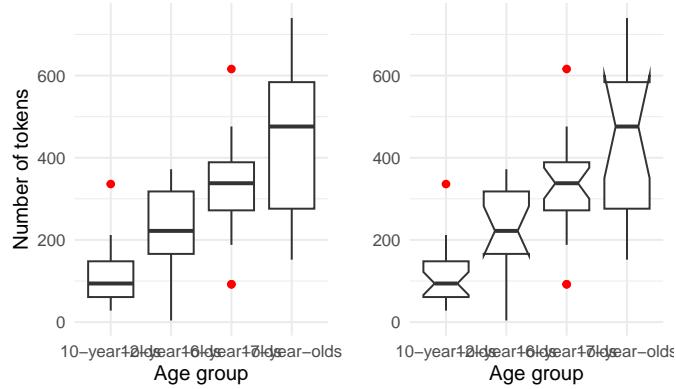


Figure 3.14: Boxplot of the relationship between `age_group` and `num_tokens` from the BELC dataset.

As seen when summarizing single continuous variables, boxplots provide a rich set of information concerning the distribution of a continuous variable. In this case we can visually compare the continuous variable `num_tokens` with the categorical variable `age_group`. The plot in the right pane includes ‘notches’. Notches represent the confidence interval, in boxplots this interval surrounds the median. When compared horizontally across levels of a categorical variable the overlap of notched spaces suggest that the true median may be within the same range. Additionally, when the confidence interval goes outside the interquartile range (the box) the notches hinge back to the either the 1st (lower) or the 3rd (higher) IQR range and suggests that the variability is high.

We can also add a third variable to our exploration. As in the barplot in Figure 3.13, the boxplot in Figure 3.15 suggests that there is an overall trend towards more tokens per composition as a learner advances in experience, except at the ‘12-year-old’ level where there appears to be a difference between ‘males’ and ‘females’.

Up to this point in our exploration of multiple variables we have always included at least one continuous variable. The central tendency for continuous variables can be summarized in multiple ways (mean, median, and mode) and when calculating means and medians, measures of dispersion are also provide helpful information summarize variability. When working with cat-



Figure 3.15: Boxplot of the relationship between `age_group`, `num_tokens` and `sex` from the BELC dataset.

Table 3.5: Contingency table for `age_group` and `sex`.

<code>sex/age_group</code>	10-year-olds	12-year-olds	16-year-olds	17-year-olds	Total
female	58% (14)	69% (11)	54% (13)	67% (10)	61% (48)
male	42% (10)	31% (5)	46% (11)	33% (5)	39% (31)
Total	100% (24)	100% (16)	100% (24)	100% (15)	100% (79)

egorical variables, however, measures of central tendency and dispersion are more limited. For ordinal variables central tendency can be summarized by the median or mode and dispersion can be assessed with an interquartile range. For nominal variables the mode is the only measure of central tendency and dispersion is not applicable. For this reason relationships between categorical variables are typically summarized using **contingency tables** which provide cross-variable counts for each level of the target categorical variables.

Let's explore the relationship between the categorical variables `sex` and `age_group`. In Table 3.5 we see the contingency table with summary counts and percentages.

As the size of the contingency table increases, visual inspection becomes more difficult. As we have seen, a graphical summary often proves more helpful to detect patterns.

In Figure 3.16 the left pane shows the counts. Counts alone can be tricky to evaluate and adjusting the barplot to account for the proportions of males to females in each group, as shown in the right pane, provides a clearer picture of the relationship. From these barplots we can see there were more females in the study overall and particularly in the 12-year-olds and 17-year-olds groups. To gauge the association strength between `sex` and `age_group` we can cal-

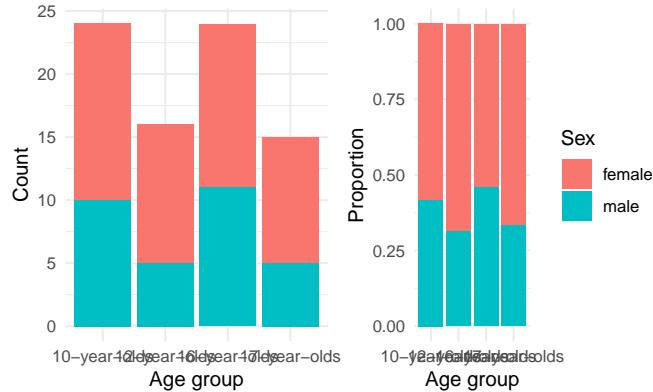


Figure 3.16: Barplot...

Table 3.6: Contingency table for `age_group`, `rank_tokens`, and `sex` (female).

rank_tokens/age_group	10-year-olds	12-year-olds	16-year-olds	17-year-olds	Total
low	27% (13)	10% (5)	4% (2)	6% (3)	48% (23)
mid	2% (1)	13% (6)	21% (10)	6% (3)	42% (20)
high	0% (0)	0% (0)	2% (1)	8% (4)	10% (5)
Total	29% (14)	23% (11)	27% (13)	21% (10)	100% (48)

culate *Cramer's V* which, in spirit, is like our correlation coefficients for the relationship between continuous variables. The Cramer's V score for this relationship is 0 which is low, suggesting that there is not a strong association between `sex` and `age_group` –in other words, the relationship is stable.

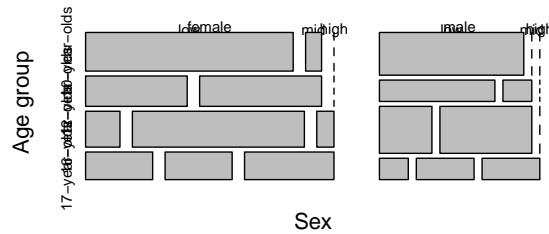
Let's look at a more complex case in which we have three categorical variables. Now the dataset, as is, does not have a third categorical variable for us to explore but we can recast the continuous `num_tokens` variable as a categorical variable if we bin the scores into groups. I've binned tokens into three score groups with equal ranges in a new variable called `rank_tokens`.

Adding a second categorical independent variable ups the complexity of our analysis and as a result our visualization strategy will change. Our numerical summary will include individual two-way cross-tabulations for each of the levels for the third variable. In this case it is often best to use the variable with the fewest levels as the third variable, in this case `sex`.

Contingency tables with this many levels are notoriously difficult to interpret. A plot that is often used for three-way contingency table summaries is a mosaic plot. In Figure 3.17 I have created a mosaic plot for the three categorical variables in the previous contingency tables.

Table 3.7: Contingency table for `age_group`, `rank_tokens`, and `sex` (male).

rank_tokens/age_group	10-year-olds	12-year-olds	16-year-olds	17-year-olds	Total
low	32% (10)	13% (4)	13% (4)	3% (1)	61% (19)
mid	0% (0)	3% (1)	23% (7)	6% (2)	32% (10)
high	0% (0)	0% (0)	0% (0)	6% (2)	6% (2)
Total	32% (10)	16% (5)	35% (11)	16% (5)	100% (31)

Figure 3.17: Mosaic plot for three categorical variables `age_group`, `rank_tokens`, and `sex` in the BELC dataset.

The mosaic plot suggests that the number of tokens per composition increase as the learner age group increases and that females show more tokens earlier.

In sum, a dataset is information but when the observations become numerous or complex they are visually difficult to inspect and understand at a pattern level. The descriptive methods described in this section are indispensable for providing the researcher an overview of the nature of each variable and any (potential) relationships between variables in a dataset. Importantly, the understanding derived from this exploration underlies all subsequent investigation and will counted on to frame your approach to analysis regardless of the research goals and the methods employed to derive more substantial knowledge.

### 3.2 Types of analysis

From identifying a target population, to selecting a data sample that represents that population, and then to structuring the sample into a dataset, the

goals of a research project inform and frame the process. So it will be unsurprising to know that the process of selecting an approach to analysis is also intimately linked with a researcher's objectives. 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 goals and how the general goals 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 to the would-be-researcher to identify these research approaches in the literature and to make appropriate decisions as to which approach their research should adopt.

### 3.2.1 Inferential data analysis

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, or top-down, approach to investigation in which every step in research stems from a 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. The aim is to infer conclusions about a certain relationship in the population based on a statistical evaluation of a (corpus) sample. So, if a researcher's aim is to draw conclusions that generalize, then, this is the analysis approach a researcher will take.

Given the fact that this approach aims at making claims that can be generalized to the larger population, the IDA approach has the most rigorous set of methodological restrictions. First and foremost of these is the fact that a testable hypothesis must be formulated *before* research begins. The hypothesis guides the collection of data, the organization of the data into a dataset and the transformation, selection of the variables to be used to address the hypothesis, and the interpretation of the results. To conduct an analysis and then draw a hypothesis which conforms to the results is known as "Hypothesis After Result is Known" (HARKing) (Kerr 1998) and this practice violates the principles of significance testing. A second key stipulation is that the reliability of the sample data, the corpus in text analysis, to provide evidence to test the hypothesis must be representative of the population. A corpus used in a study which is misaligned with the hypothesis undermines the ability of the researcher to make valid claims about the population. In essence, IDA is only as good as the primary data it is based on.

At this point, let me elaborate on the potentially counterintuitive nature of hypothesis formulation and testing. The IDA, or Null-Hypothesis Significance

Testing (NHST), paradigm is in fact approached by proposing two mutually exclusive hypotheses. The first is the **Alternative Hypothesis** ( $H_1$ ).  $H_1$  is a precise statement grounded in the previous literature outlining a predicted relationship (and in some cases the directionality of a relationship). This is the effect that the research aims to investigate. The second hypothesis is the **Null Hypothesis** ( $H_0$ ).  $H_0$  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.

So to provide an example consider a hypothetical study which is aimed at investigating the claim that men and women differ in terms of the number of questions they use in spontaneous conversations. The Alternative Hypothesis would 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. Formally:

$H_0$ : There is no difference between how men and women use questions in spontaneous conversations.

Note that stated in this way our hypothesis makes no prediction about the directionality of the difference between men and women, only that there is a difference. It is a likely scenario that a hypothesis will stake a claim on the direction of the difference. A directional hypothesis would look like this:

$H_1$ : Women use more questions than men in spontaneous conversations.

$H_0$ : There is no difference between how men and women use questions in spontaneous conversations or men use more questions than women.

A further aspect which may run counter to expectations is that the aim of hypothesis testing is not to find evidence in support of  $H_1$ , but rather the aim is to assess the likelihood that we can reliably reject  $H_0$ . The default assumption is that  $H_0$  is true until there is sufficient evidence to reject it and accept  $H_1$ , the *alternative*. 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.

Let's now turn to the identification of variables, the statistical interrogation of these variables, and the interpretation of the statistical results. First, since a clearly defined and testable hypothesis is at the center of the IDA approach, the variables are in some sense pre-defined. The goal of the researcher is to select data and curate that data to produce variables that are operationalized (practically measured) to test the hypothesis. A second consideration are the roles that the variables will play in the analysis. In standard IDA one variable will be the **dependent variable** and one or more variables will be **independent variables**. The dependent variable, sometimes referred to as the outcome or response variable, is the variable which contains the information which is predicted to depend on the information in the independent variable(s). It is the variable whose variation a research study seeks to explain. An independent variable, sometimes referred to as a predictor or explanatory variable, is a variable whose variation is predicted to explain the variation in the dependent variable.

Returning to our hypothetical study on the use of questions between men and women in spontaneous conversation, the frequency of questions used by each speaker would be our dependent variable and the biological sex of the speakers our independent variable. This is so because hypothesis ( $H_1$ ) states the proposition that a speaker's sex will predict the frequency of questions used.

In our hypothetical study we've identified two variables, one dependent and one independent. It is important keep in mind that there can be multiple independent 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.

Say we formulate a more complex relationship where the educational level of our speakers is also related to the number of questions. We can update our hypothesis to reflect such a scenario.

$H_1$ : Less educated women use more questions than men in spontaneous conversations.

$H_0$ : There is no difference between how men and women use questions in spontaneous conversations regardless of educational level, or more educated women use more questions than less educated women, or men use more questions than women.

The hypothesis we have described predicts what is known as an *interaction*; the relationship between our independent variables predict different variational patterns in the dependent variable. As you most likely can appreciate the more independent variables we include in our hypothesis, and by extension our analysis, the more difficult it becomes to interpret. Due to the increasing difficulty for interpretation, in practice, IDA studies rarely include more than two or three independent variables in the same analysis.

Independent 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 independent 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.

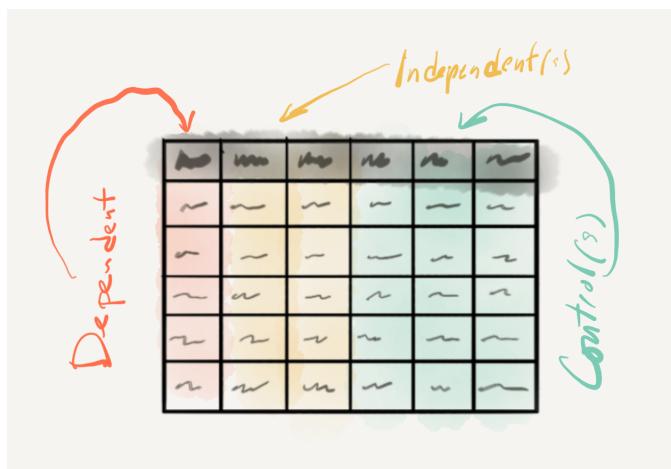


Figure 3.18: Variable roles in inferential analysis.

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 independent 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 Gries (2013) and Paquot and Gries (2020) for a more exhaustive description). Below I've summarized some common statistical scenarios

Table 3.8: Common monofactorial tests.

Variable roles		
Dependent	Independent	Test
Categorical	Categorical	Pearson's Chi-squared test
Continuous	Categorical	Student's t-Test
Continuous	Continuous	Pearson's correlation test

Table 3.9: Common multifactorial tests.

Variable roles			
Dependent	Independent	Control	Test
Categorical	<i>varied</i>	<i>varied</i>	Logistic regression
Continuous	<i>varied</i>	<i>varied</i>	Linear regression

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, non-independent variables, etc.

In Table 3.8 we see **monofactorial tests**, tests with only one independent variable.

Table 3.9 includes a listing of **multipfactorial tests**, tests with more than one independent and/ or control variables.

One key point to make before we turn to how to interpret the statistical results is concerns the use of the data in IDA. In contrast to the other two analysis methods we will cover, the data in IDA is only used once. That is to say, that the entire dataset is used a single time to statistically interrogate the relationship(s) of interest. The resulting confidence metrics (p-values, etc.) are evaluated and the findings are interpreted. The practice of running multiple tests until a statistically significant result is found is called “p-hacking” (Head et al. 2015) and like HARKing (described earlier) violates statistical hypothesis testing practice. For this reason it is vital to identify your statistical approach from the outset of your research project.

Now let’s consider how to approach interpreting the results from a statistical test. As I have now made reference to multiple times, the results of statistical procedure in hypothesis testing will result in a confidence metric. The most standard and widely used of these confidence metrics is the p-value. The p-value provides a probability that the results of our statistical test could be explained by the null hypothesis. When this probability crosses below the threshold of .05, the result is considered statistically significant, otherwise

we have a ‘null result’ (i.e. non-significant). However, this sets up a binary distinction that can be problematic. On the one hand what is one to do if a test returns a p-value of .051 or something ‘marginally significant’? According to standard practice these results would not be statistically significant. But it is important to note that a p-value is sensitive to the sample size. A small sample may return a non-significant result, but a larger sample size with the same underlying characteristics may very well return a significant result. On the other hand, if we get a statistically significant result, do we move on –case closed? As I just pointed out the sample size plays a role in finding statistically significant results, but that does not mean that the results are ‘important’ for even small effects in large samples can return a significant p-value.

It is important to underscore that the purpose of IDA is to draw conclusions from a dataset which are generalizable to the population. These conclusions require that there are rigorous measures to ensure that the results of the analysis do not overgeneralize (suggest there is a relationship when there is not one) and balance that with the fact that we don’t want to undergeneralize (miss the fact that there is an relationship in the population, but our analysis was not capable of detecting it). Overgeneralization is known as **Type I error** or false positive and undergeneralization is a **Type II error** or false negative.

For these reasons it is important to calculate the size and magnitude of the result to gauge the uncertainty of our result in standardized, sample size-independent way. This is performed by analyzing the **effect size** and reporting a **confidence interval (CI)** for the results. The wider the CI the more uncertainty surrounds our statistical result, and therefore the more likely that our significant p-value could be the result of Type I error. A non-significant p-value and large effect size could be the result of Type II error. In addition to vetting our p-value, the CI and effect size can help determine if a significant result is reliable and ‘important’. Together effect size and CIs aid in our ability to realistically interpret confidence metrics in statistical hypothesis testing.

### 3.2.2 Predictive data analysis

Predictive data analysis (PDA) is the first of the two types of statistical approaches we will cover that fall under **machine learning**. A branch of artificial intelligence (AI), machine learning aims to develop computer algorithms that can essentially learn patterns from data automatically. In the case of PDA, also known as **supervised learning**, the learning process is guided (supervised) by directing an algorithm to associate patterns in a variable or set of variables to single particular variable. The particular variable is analogous to some degree to a dependent variable in IDA, but in the machine learning literature this variable is known as the **target** variable. The other variable or (more often than not) variables are known as **features**. The goal of PDA is to develop a statistical generalization that can accurately predict the values of a target variable using the values of the feature variables. PDA can

be seen as a mix of deductive (top-down) and inductive (bottom-up) methods in that the target variable is determined by a research goal but the feature variables and choice of statistical method (algorithm) are not fixed and can vary depending on their usefulness in effectively predicting the target variable. PDA is a versatile method that often employed to derive intelligent action from data, but it can also be used for hypothesis generation and even hypothesis testing, under certain conditions. If a researcher's aim is to create model that can perform a language related task, explore association strength between a target variable and various types and combinations of features, or to perform emerging alternative approaches to hypothesis testing<sup>6</sup>, this is the analysis approach a researcher will take.

At this point let's consider some departures from the inferential data analysis (IDA) approach we covered in the last subsection that are important to highlight to orient our overview of PDA. First, while the cornerstone of IDA is the hypothesis, in PDA this is typically not the case. A research question which identifies a source of potential uncertainty in an area and outlines a strategy for addressing this uncertainty is sufficient groundwork to embark on an analysis. A second divergence, is the fact that the data is used in a very distinct way. In IDA the entire dataset is statistically interrogated once and only once. In PDA the dataset is (minimally) partitioned into a **training set** and a **test set**. The training set is used to train a statistical model and the test set is left to test the accuracy of the statistical model. The training set typically constitutes a larger portion of the data (typically around 75%) and serves as the test bed for iteratively applying one or more algorithms and/ or feature combinations to produce the most successful learning model. The test set is reserved for a final evaluation of the model's performance. Depending 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. In this scenario the proportions of the partitions vary, but a good rule of thumb is to reserve 60% of the data for training, 20% for development, and 20% for testing.

Let's now turn to the identification of variables, the statistical interrogation of these variables, and the interpretation of the statistical results. In IDA the variables (features) are pre-determined by the hypothesis and the informational values and number of these variables plays a significant role in selecting a statistical procedure (algorithm). Lacking a hypothesis, a PDA approach's main goal is to make accurate predictions on the target variable and is free to explore any number of features and feature combinations to that end. The target variable is the only variable which necessarily fixed and in this light pre-determined.

To give an example, let's consider a language task in which the goal is to take

---

<sup>6</sup>see Deshors and Gries (2016) and R. Harald Baayen (2011)

Table 3.10: First ten observations from the SMS Spam Collection (v.1)

sms_type	message
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to rec
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though
spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up
ham	Even my brother is not like to speak with me. They treat me like aids patent.
ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as yo
spam	WINNER!! As a valued network customer you have been selected to receivea £900 prize reward
spam	Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with

text messages (SMS) and develop a language model that predict if a message is spam or not. Minimally we would need data which includes individual text messages and each of these text message will need to be labeled as being either spam or legitimate messages ('ham' in this case). In Table 3.10 we see the first ten of 5574 observations from the SMS Spam Collection (v.1) dataset collected by Almeida, Gómez Hildago, and Yamakami (2011).

As it stands we have two variables; `sms_type` is clearly the target and `message` contain the full messages. The question is how best to transform the information in the `message` variable such that it will provide an algorithm useful information to predict each value of `sms_type`. Since the informational value of `sms_type` is categorical we will call the values **classes**. The process of deciding on how to transform the information in `message` into useful features is called **feature engineering** and it is a process which is much an art as a science. On the creative side of things it is often helpful to have a mixture of relevant domain knowledge and clever hacking skills to envision what features may work best. The more logistic side of things requires some knowledge about the strengths and weaknesses of various learning algorithms when dealing with certain number and informational value feature combinations.

Leaving the choice of algorithm aside, let's focus on feature engineering. Since each `message` value is a unique message, the chance that using `message` as it is, is not likely to help us make reliable predictions about the status of new message ('spam' or 'ham'). A simple first-pass approach to decomposing `message` to draw out similarities and distinctions between the classes may be to break each message into words. Now SMS messages are not your average type of text -there are many non-standard forms. So our definition of word may simply be character groupings broken apart by whitespace. To avoid confusion between our common-sense understanding of word and the types of character strings, it is often the case that language feature values are called **terms**. Other term

types may work better, n-grams, character sequences, stems/lemmas, or even combinations of these. Certain terms may be removed that are potentially uninformative either based on their class (stopwords, numerals, punctuation, etc.) or due to their distribution. The process of systematic isolation of terms which are more informative than others is called **dimensionality reduction** (Kowsari et al. 2019). With experience a researcher will become more adept at recognizing advantages and potential issues and alternative ways of approaching the creation of features but there is almost always some level of trial and error in the process. Feature engineering is very much an exploratory process. It is also iterative. You can try a set of features with an algorithm and produce a language model and test it on the training set – if it is accurate, great. If not, you can brainstorm some more – you are free to try further engineer the features trying new features or feature measures (term weights) and/ or change the learning algorithm.

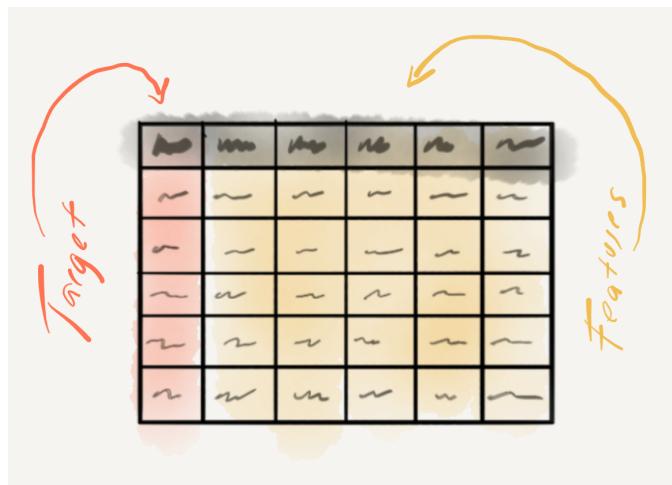


Figure 3.19: Variable roles in predictive analysis.

Let's now turn to some considerations to take into account when selecting a statistical algorithm. First, just as in IDA, variable informational value plays a role in algorithm selection, specifically the informational value of the target variable. If the target variable is categorical, then we are looking for a **classification** algorithm. If the target variable is continuous, we will employ a **regression** algorithm.<sup>7</sup> Some common classification algorithms are listed in Table 3.11.

Another consideration to take into account is the whether the researcher aims to go beyond simply using an algorithm to make accurate predictions, but

---

<sup>7</sup>The name regression can be a bit confusing given a very common classification algorithm is “Logistic Regression”.

Table 3.11: Some common supervised learning algorithms.

Classification	Regression
Logistic Regression	Linear Regression
Support Vector Machine	Support Vector Regression
Naïve Bayes Classifier	Poisson Regression
Neural Network	
Decision Tree	

also wants to understand how the algorithm made its predictions and what contribution features made in the process. There are algorithms that produce models that allow the researcher to peer into and understand its inner workings (e.g. logistic regression, naïve bayes classifiers, *inter alia*) and those that do not (e.g. neural networks, support vector machines, *inter alia*). Those that do not are called '**black-box**' algorithms. Neither type assures the best prediction accuracy. Important trade-offs need to be considered, however, if the best prediction comes from a black-box method, but the goal of the research is to understand the contribution of the features to the model's predictions.

Once we have identified our target variable, engineered a promising set of features, and selected an algorithm to employ that meets our research goals, it is now time to interrogate the dataset. The first step is to partition the dataset into a training and test set. The training set is the dataset we will use to try out different features and/ or algorithms with the aim of developing a model which can most accurately predict the target variable values in this training set. This is the second step and it's done by first training an algorithm to associate the features with the (actual) target values. Next, the resulting model is then applied to the same training data, yet with the target variable removed, or hidden, from the machine learner. The target values predicted by the model for each observation are compared to the actual target values. The more predicted and actual values for the target variable coincide, the more accurate the model. If the model shows high accuracy, then we are ready to move to evaluate this model on the test set (again removing the target variable). If the model accuracy is low, it's back to the drawing board either returning to feature engineering and/ or algorithm selection in hopes to improve model performance. In this way, the training data can be used multiple times, a clear divergence from standard IDA methods in which the data is interrogated and analyzed once and only once.

For all applications of PDA the interpretation of the prediction model includes some metric or metrics of accuracy comparing the extent to which the models predictions and the actual targets align. In cases in which the inner workings

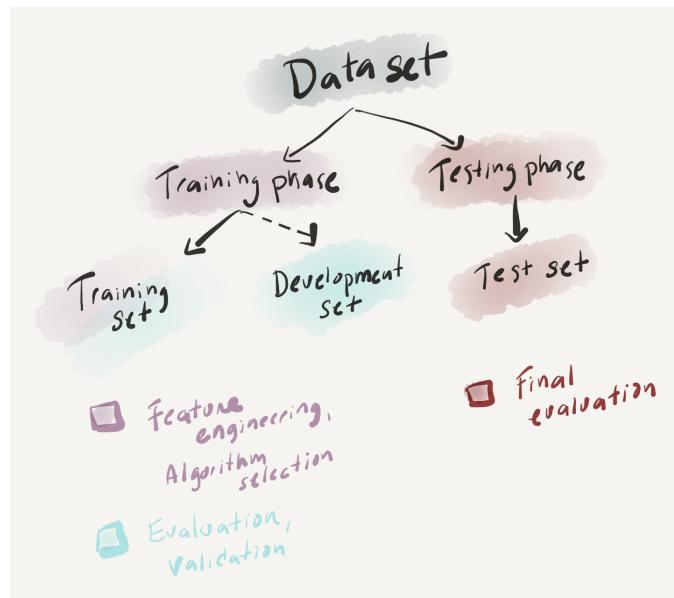


Figure 3.20: Phases in predictive analysis.

of the model are of interest, a researcher can dive into features and their contributions to the prediction model in an exploratory fashion according to the research goals. The exploration of features, then, varies, so at this time let's focus on the metrics of prediction accuracy.

The standard form for evaluating a model's performance differs between classification models (naive bayes) and regression models (linear regression). For classification models, a cross-tabulation of the predicted and actual classes results in a **contingency table** which can be used to calculate **accuracy** which is the sum of all the correctly predicted observations divided by the total number of observations in the test set. In addition to accuracy, there are various other measures which aim to assess a model's performance to gain more insight into the potential over- or under-generalization of the model (*Precision* and *Recall*). For regression models, differences between predicted and actual values can be assessed using a **coefficient of correlation** (typically  $R^2$ ). Again, more fine-grained detail about the model's performance can be calculated (*Root Mean Square Error*).

Another component worthy of consideration when evaluating a model's performance is how do we determine if the performance is actually good. One the one hand, accuracy rates into the 90+% range on the test set is usually a good sign that the model is performing well. No model will perform with perfect accuracy, however, and depending on the goal of the research particular error patterns may be more important, and problematic, than the overall pre-

diction accuracy. On the other hand, another eventuality is that the model performs very well on the training set but that on the test set (new data) the performance drops significantly. This is a sign that during the training phrase the machine learning algorithm learned nuances in the data ('noise') that obscure the signal pattern to be learned. This problem is called **overfitting** and to avoid it researchers iteratively run evaluations of the training data using resampling. 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.

### 3.2.3 Exploratory data analysis

The last of the three analysis types, exploratory data analysis (EDA) includes a wide range of methods whose objective is to identify structure in datasets using only the data itself. In this way, EDA is an inductive, bottom-up approach to data analysis, which does not make any formal assumptions about the relationship(s) between variables. EDA can be roughly broken into two subgroups of analysis. **Unsupervised learning**, like supervised learning (PDA), is a subtype of machine learning. However, unlike prediction, unsupervised learning does not include a target variable to guide associations. The second subgroup of EDA methods can be seen as a (more robust) extension of the **descriptive analysis methods** covered earlier in this chapter. Either through unsupervised learning or descriptive methods, EDA 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. These qualitative assessments may prove useful to generate hypotheses or to generate groupings to be used in predictive analyses. So, if a researcher's aim is to probe a dataset in order to explore potential relationships in an area where predictions and/ or hypotheses cannot be clearly made, this is the analysis approach to choose.

In contrast to both IDA and even PDA in which there are assumptions made about what relationship(s) to explore, EDA makes no such assumptions. Furthermore, given the exploratory nature of the process, EDA is not an approach which can in itself be used to make conclusive generalizations about the populations from which the (corpus) sample in which it is drawn. For IDA the fidelity of the sample and the process of selection of the variables is of utmost importance to ensure that the statistical results are reliably generalizable. Even in the case of PDA, the sample and variables selected are key to building a robust predictive model. However, in contrast to IDA, but similar to

PDA, EDA methods may reuse the data selecting different variables and/or methods as research goals dictate. If a machine learning approach to EDA is adopted, the dataset can be partitioned into training and test sets, in a similar fashion to PDA. And as with PDA, the training set is used for refining statistical measures and the test set is used to evaluate the refined measures. Although the evaluation results still cannot be used to generalize, the insight can be taken as stronger evidence that there is a potential relationship, or set of relationships, worthy of further study.

Another notable point of contrast concerns the interpretation of EDA results. Although quantitative in nature, exploratory methods involve a high level of human interpretation. Human interpretation is a part of each stage of data analysis, and each statistical approach, in general, but exploratory methods produce results that require associative thinking and pattern detection which is distinct from the other two analysis approaches, in particular, IDA.

Again, as we have done for the other two analysis approaches, let's turn to the process of variable identification, data interrogation, and interpretation methods. As in the case of PDA, EDA only requires a research goal. But in PDA, the research goal centered around predicting a target variable. In EDA, there is no such focus. The research goal may in fact be less defined and a researcher may consider various relationships in turn or simultaneously. The curation of the variables, however, does overlap in spirit to the process of **feature engineering** that we touched on for creating variables for predictive models. But in EDA the measure to gauge whether the engineered variables are good, is left to the qualitative evaluation of the researcher.

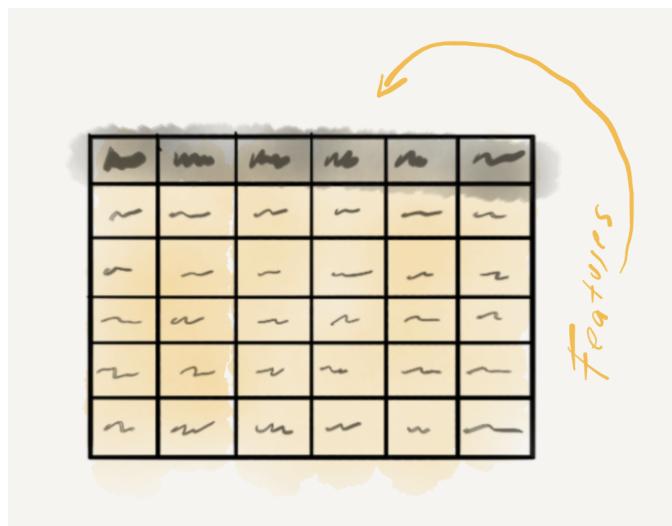


Figure 3.21: Variable roles in exploratory analysis.

Table 3.12: 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...

Table 3.13: Some common EDA analysis methods

Descriptive	Unsupervised learning
Term frequency analysis	Cluster analysis
Term keyness analysis	Topic Modeling
Collocation analysis	Dimensionality reduction

For illustrative purposes let's consider the State of the Union Corpus (SOTU) (Benoit 2020). The presidential addresses and a set of meta-data 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 3.12.

A dataset such as this one could be leveraged to explore many different types of research questions. Key to guiding the engineering of features, however, is to clarify from the outset of the research project what the entity of study is, or **unit of analysis**. In IDA and PDA approaches, the unit of analysis forms an explicit part of the research hypothesis or goal. In EDA the research question may have multiple fronts, which may be reflected in differing units of analysis. For example, based on the SOTU dataset, we could be interested in political rhetoric, language of particular presidents, party ideology, etc. 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 vary widely, between and within the same research project, but for instructive purposes we can draw a distinction between descriptive methods and unsupervised learning methods.

EDA leans heavily on visual representations of both descriptive and unsupervised learning methods. Visualizations enable humans to identify and ex-

trapolate associative patterns. Visualizations range from standard barplots and scatterplots to network graphs and dendrograms and more. Some sample visualizations based on the SOTU Corpus are found in Figure 3.22.

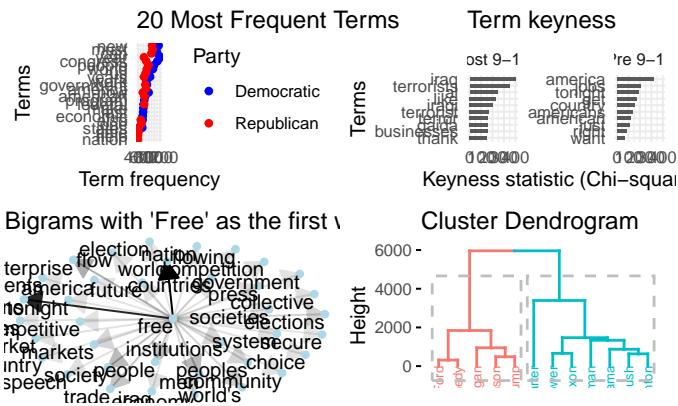


Figure 3.22: Sample visualizations from the SOTU Corpus (1946-2020).

Just as feature selection and analysis method, the interpretation of the results in EDA are much more varied than in the other analysis methods. EDA methods provide information which requires much more human intervention and associative interpretation. In this way, EDA can be seen as a quantitatively informed qualitative assessment approach. The results from one approach can be used as the input to another. Findings can lead to further exploration and probing of nuances in the data. 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.

### 3.3 Reporting

Much of the necessary reporting for an analysis features in prose as part of the write-up of a report or article. This will include descriptive summaries, a blueprint of the method(s) used, and the results. Descriptive summaries will often include assessments of individual variables and/or relationships between variables (central tendency, dispersion, association strength, etc.). Any procedures applied to diagnose or to correct the data should also be included in the final report. This information is key to helping readers assess the results from the analysis. A blueprint of the methods used will describe the variable selection process, how the variables were used in the statistical analysis, and any other information that is relevant for a reader to understand what was done and why it was done. Reporting results from an analysis will depend

on the type of analysis and the particular method(s) employed. For inferential analyses this will include the test statistic(s) ( $X^2$ ,  $R^2$ , etc.) and some measure of confidence ( $p$ -value, confidence interval, effect size). In predictive analyses accuracy results and related information will need to 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.

While a good article write-up will include the most vital information to understand the procedures taken in an analysis, there are many more details which do not traditionally appear in prose. If a research project was conducted programmatically, however, the programming files (scripts) used to generate the analysis can (and should) be shared. While the scripts themselves are highly useful for other researchers to consult and understand in fine-grained detail the steps that were taken, it is important to also recognize that the research project should be well documented –through organized project directory and file structure as well as through code commenting. This description and instructions on how to run the analysis form a **research compendium** which ensure that the research conducted is easily understood and able to be reproduced and/ or enhanced by other researchers.

## Summary

In this chapter we have focused on description and analysis –the third component of DIKI Hierarchy. This process is visually summarized in Figure 3.23.

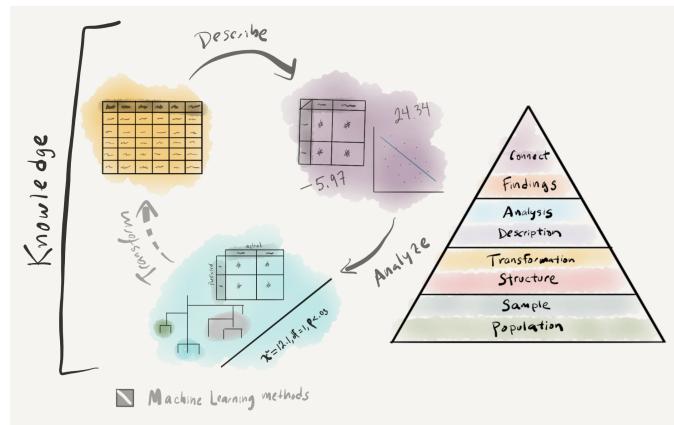


Figure 3.23: Approaching analysis: visual summary

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 descriptive 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 covered three data analysis types in this chapter: inferential, predictive, and exploratory. Each of these embodies very 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 discovering the extent to which a given relationship can be extrapolated from the data to provide a model of language that can accurately predict an outcome using new data. While many times predictive analysis is used to perform language tasks, it can also be a highly effective methodology for applying different algorithmic approaches and exploring relationships a target 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 deriving 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 reporting the metrics, procedures, and results from analysis. 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

### 💡 Recipe

**What:** Descriptive assessment of datasets<sup>a</sup>

**How:** Read Recipe 4 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).

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_4.html](https://lin380.github.io/tadr/articles/recipe_4.html)

### 💡 Lab

**What:** Descriptive assessment of datasets<sup>a</sup>

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

**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.

<sup>a</sup>[https://github.com/lin380/lab\\_4](https://github.com/lin380/lab_4)

## Questions

### 💡 Conceptual questions

- What are the key differences between descriptive and analytic statistics?
- What informational values can a variable take?
- 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 and which are continuous? What are the informational values of each variable? What are the measures of central tendency and dispersion for each variable?
- What type(s) of tables or plots are appropriate for summarizing a vari-

- able? What type(s) of tables or plots are appropriate for summarizing the relationship between two variables?
- In the following variables and information values, identify if the plots are appropriate for summarizing the relationship.
  - What are the key differences between inferential, predictive, and exploratory 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:
  - Given the following research questions, identify which type of analysis is most appropriate and why:
  - 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?
  -

**i** 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*

If we knew what it was we were doing, it would not be called research, would it?

— Albert Einstein

“The reproducibility of studies and the ability to follow up on the work of others is key for innovation in science and engineering.”

— Leland Wilkinson

### Keys

- What are the strategies for selecting a research area and identifying a research problem?
- How does a research problem and research aim frame the development of a research statement?
- What is a ‘research blueprint’ and how do the conceptual and practical steps involved in developing it aid the researcher as well as the 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 a 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

### 💡 Interactive programming

**What:** Version control<sup>a</sup>

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

**Why:** To ....

<sup>a</sup><https://github.com/lin380/swirl>

---

## 4.1 Keys to strong research

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 area of research is the first decision to make in terms of where to make a contribution to understanding. At this point, the aim is to identify a general area of interest where a researcher wants to derive insight. 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 Linguistics<sup>1</sup> (Brown 2005), A Practical Guide to Electronic Resources in the Humanities<sup>2</sup> (Dubnjakovic and Tomlin 2010), Routledge encyclopedia of translation technology<sup>3</sup> (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, Reddit<sup>4</sup> has a wide variety of active subreddits (r/LanguageTechnology<sup>5</sup>, r/Linguistics<sup>6</sup>, r/corpuslinguistics<sup>7</sup>, r/DigitalHumanities<sup>8</sup>, etc.). Twitter and Facebook also have interesting posts on linguistics and language-related fields worth following. Through one of these social media site you may find particular people that maintain a blog worth browsing. For example, I follow Julia Silge<sup>9</sup>, Rachel Tatman<sup>10</sup>, and Ted Underwood<sup>11</sup>, *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.

---

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

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

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

<sup>4</sup><https://www.reddit.com/>

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

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

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

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

<sup>9</sup><https://juliasilge.com/>

<sup>10</sup><http://www.rctatman.com/>

<sup>11</sup><https://tedunderwood.com/>

Table 4.2: A list of some linguistics journals.

Resource
<a href="https://www.euppublishing.com/loi/cor">Corpora</a>
<a href="https://www.degruyter.com/journal/key/CLLT/html">Corpus Linguistics and Linguistic Theory</a>
<a href="https://benjamins.com/catalog/ijcl">International Journal of Corpus Linguistics</a>
<a href="http://ijls.net/">International Journal of Language Studies</a>
<a href="https://www.cambridge.org/core/journals/journal-of-child-language">Journal of Child Language</a>
<a href="https://www.cambridge.org/core/journals/journal-of-linguistic-geography/all-issues">Journal of Linguistic Geography</a>
<a href="http://www.tandfonline.com/toc/njql20/current">Journal of Quantitative Linguistics</a>

#### 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 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 in conducting searches. These can be world-accessible resources (e.g. Google Scholar<sup>12</sup>) or limited-access resources that are provided through an academic institution (e.g. Linguistics and Language Behavior Abstracts<sup>13</sup>), ERIC<sup>14</sup>, PsycINFO<sup>15</sup>, etc.). Some organizations and academic institutions provide research guides<sup>16</sup> 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 the following tables 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 re-

<sup>12</sup><https://scholar.google.com/>

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

<sup>14</sup><https://eric.ed.gov/>

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

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

Table 4.3: A list of some humanities journals.

Resource	Description
<a href="http://www.digitalhumanities.org/dhq/">Digital Humanities Quarterly</a>	Digital Humanities
<a href="https://academic.oup.com/dsh">Digital Scholarship in the Humanities</a>	DSH or Digital
<a href="https://culturalanalytics.org/">Journal of Cultural Analytics</a>	Cultural Analytics

Table 4.4: A list of some computational linguistics journals.

Resource
<a href="https://direct.mit.edu/coli">Computational Linguistics</a>
<a href="http://lrec-conf.org/">LREC Conferences</a>
<a href="https://transacl.org/index.php/tacl/index">Transactions of the Association for Computational

search, but 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 Zotero<sup>17</sup><sup>18</sup>, Mendeley<sup>20</sup>, or BibDesk<sup>21</sup> 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 first<sup>22</sup>. Annotate these publications using highlighting features of the citation management software

<sup>17</sup><https://www.zotero.org/>

<sup>18</sup>Zotero Guide<sup>19</sup>

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

<sup>21</sup><https://bibdesk.sourceforge.io/>

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

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, theoretical, methodological, or appear along other lines. Note these aspects and continue to conduct more refine 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 Findings

### 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, evaluate, 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. Consider how your research statement relates the previous literature. Do you aim to test a hypothesis based on previous exploratory analyses? Are you looking to generate new knowledge in an (apparently) uncharted area? Etc.

In general, a problem statement which addresses a smaller, nuanced gap will tend to adopt similar research aims as the previous literature while a larger, more divergent gap will tend to adopt a distinct research aim. This is not a hard rule, but more of a heuristic, however, and it is important to be familiar with both the previous literature, the nature of different types of analysis, and the goals of the research to ensure that the research is best-positioned to generate findings that will contribute to the existing body of understanding in a principled way.

#### 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. The research statement can be seen as the content and the research aim as the form.

The form of a research question will vary based on 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 one of the hypotheses we previously sketched out in Chapter 3, leaving aside the implicit null hypothesis.

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 look something like this.

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

And a similar exploratory-based research question would take this form.

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.

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 men and women. 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 men and women 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 the previous examples, we identified the number of conversations as part of the research question. But in other cases a researcher may seek to understand other aspects of questions in spontaneous conversations (i.e type of question, features of questions, etc.). The unit of observation, however, would remain the same.

---

#### 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) the process of identifying the data, information, and analysis methods to be used and (2) the creation of a plan to structure and document the project.

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.

#### 4.4.1 Identify

Importance of identifying and documenting the key aspects required to conduct the research cannot be understated. On the one hand this process links concept to 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, on paper, may present challenges that may require modification or reevaluation of the viability of the project. It is not uncommon to encounter roadblocks or even dead-ends for moving a well-founded research question forward when considering the available data, a researcher's (current) technical and/ or research skills, and the given time frame for the project. In practice, the process of identifying the data, information, and methods of analysis are considered in tandem with the investigative work to develop a research aim and research question. In this subsection I will cover the main characteristics to consider when developing a research blueprint.

The first, and most important, part of establishing a research blueprint is to **identify a viable data source**. Regardless of how you find and access the data, it is essential to vet the corpus sample in light of the research question. 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 conversa-

tions. 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, there 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 meta-data is incomplete for the goals of the research, it is sometimes possible to merge meta-data from other sources.

The third step is to **identify a method of analysis**. The selection of the analysis approach that was part of the research aim 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. In inferential research, the number and information values of the variables to be analyzed will be of key importance (Gries 2013). The informational value of the dependent variable will again narrow the search for the appropriate method. The number of independent variables also plays an important role. For example, a study with a categorical dependent variable with a single categorical independent variable will lead the researcher to the Chi-squared test. A study with a continuous dependent variable with multiple independent variables will lead to linear regression. Another aspect of note for inference studies is the consideration of the distribution of continuous 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.

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 numeric prediction 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 algorithms have strengths and shortcomings when working large numbers and/ or types of features (Lantz 2013).

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.

In sum, the identification of the data, information, and analysis methods that will be used in the proposed research are key to ensuring the research is viable. Be sure to document this process in prose and describe the strengths and potential shortcomings of (1) the corpus data selected, (2) the information to be extracted for analysis, and (3) the analysis method(s) that are appropriate for the research aim and what the evaluation method will be. Furthermore, not every eventuality can be foreseen. It is helpful to include a description of aspects of this process which may pose challenges and to include potential contingency plans as part of this prose description.

#### 4.4.2 Plan

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).

##### 4.4.2.1 Principles of reproducible projects

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

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 data and derived 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 analysis file (script) should represent a particular, well-defined step in the research process.

5. Each analysis 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 analysis scripts should be tied together by a ‘master’ script that is used to coordinate the execution of all the analysis steps.
7. Everything should be documented. This includes 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 countless 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 workflowr<sup>23</sup> and ProjectTemplate<sup>24</sup> but there are many other resources for R included on the CRAN Task View for Reproducible Research<sup>25</sup>. There are many advantages to working with pre-existing frameworks for the savvy R programmer.

#### 4.4.2.2 Project template

In this textbook, however, I have developed a project template (available on GitHub<sup>26</sup>) which I believe simplifies and makes the process more transparent for beginning and intermediate R programmers, the directory structure is provided below.

```
#> ./project_template/
#> +- README.md
#> +- _pipeline.R
#> +- analysis
#> |   +- 1_acquire_data.Rmd
#> |   +- 2_curate_dataset.Rmd
#> |   +- 3_transform_dataset.Rmd
#> |   +- 4_analyze_dataset.Rmd
#> |   +- 5_generate_article.Rmd
#> |   +- _session-info.Rmd
#> |   +- _site.yml
#> |   +- index.Rmd
#> |   \-- references.bib
#> +- data
#> |   +- derived
#> |   \-- original
```

---

<sup>23</sup><https://jdblischak.github.io/workflowr/>

<sup>24</sup><http://projecttemplate.net/>

<sup>25</sup><https://cran.r-project.org/web/views/ReproducibleResearch.html>

<sup>26</sup>[https://github.com/lin380/project\\_template](https://github.com/lin380/project_template)

```
#> \-- output
#>     +-- figures
#>     \-- results
```

Let me now describe how this template structure aligns with the seven principles of quality reproducible research.

1. All files are plain text (e.g. `.R`, `.Rmd`, `.csv`, `.txt`, etc.).
2. There are three main directories `analysis/`, `data/`, and `output/`.
3. The `data/` directory contains sub-directories for `original` ('read-only') data and `derived` data.
4. The `analysis/` directory contains five scripts which are numbered to correspond with their sequential role in the research process.
5. Each of these analysis scripts are designed to be modular; input and output must be explicit and no intermediate objects are carried over to other analysis scripts. Dataset output should be written to and read from the `data/derived/` directory. Figures and statistical results should be written to and read from `output/figures/` and `output/results` respectively.
6. All of the analysis scripts, and therefore the entire project, are tied to the `_pipeline.R` script. To reproduce the entire project only this script need be run.
7. Documentation takes place at many levels. The `README.md` file is the first file that a researcher will consult. It contains a brief description of the project goals and how to reproduce the analysis. Analysis scripts use the Rmarkdown format (`.Rmd`). This format allows researchers to interleave prose description and executable code in the same script. This ensures that the rationale for the steps taken are described in prose, the code is made available to consult, and that code comments can be added to every line. The `_session-info.Rmd` script is merged with each analysis script to provide information about the computing environment and packages used to conduct each step analysis. As this is a template, no data or datasets appear. However, once data is acquired and that data is curated and transformed, documentation for these resources should be documented for each resource in a data dictionary along side the data(set) itself.

The aspects of the project template described in points 1-7 together form the backbone for reproducible research. This template, however, includes additional functionality to enhance efficient and communicable research. The `_pipeline.R` script executes the analysis scripts in the `analysis` directory, but as a side effect also produces a working website<sup>27</sup> and a journal-ready ar-

---

<sup>27</sup>[https://lin380.github.io/project\\_template\\_demo/](https://lin380.github.io/project_template_demo/)

ticle for publishing your analysis, results, and findings to the web in HTML<sup>28</sup> and PDF<sup>29</sup> format. The `index.Rmd` file is the splash page for the website and is a good place to house your pre-analysis investigative work including your research area, problem, aim, and question and to document your research blueprint including the identification of viable data resource(s), the key variables for the analysis, the analysis method, and the method of assessment. All Rmarkdown files provide functionality for citing and organizing references. The `references.bib` file is where references are stored and can be used to include citations that support your research throughout your project.

#### 4.4.3 Scaffold

This template will allow you to organize your research design and align it with implementation steps to conduct quality reproducible research. To set the structure for you to conduct your analysis, you will need to download or fork and clone this template from the GitHub repository and then make some adjustments to personalize this template for your research.

To create a local copy of this project template either:

1. Download and decompress the .zip file<sup>30</sup>
2. If you have git<sup>31</sup> installed on your machine and a GitHub account<sup>32</sup>, fork the repository<sup>33</sup> to your own GitHub account. Then open a terminal in the desired location and clone the repository<sup>34</sup>. If you are using RStudio, you can setup a new RStudio Project with the clone using the ‘New Project...’ dialog, choosing ‘Version Control’, and following the steps.

Before you begin configuring and adding your project-specific details to this template. Reproduce this project ‘as-is’ to confirm that it builds on your local machine.

In RStudio or in R session in a Terminal application, open the console in the root directory of the project. Then run:

```
source("_pipeline.R")
```

---

<sup>28</sup>[https://lin380.github.io/project\\_template\\_demo/5\\_generate\\_article.html](https://lin380.github.io/project_template_demo/5_generate_article.html)

<sup>29</sup>[https://lin380.github.io/project\\_template\\_demo/article.pdf](https://lin380.github.io/project_template_demo/article.pdf)

<sup>30</sup>[https://github.com/lin380/project\\_template/archive/refs/heads/main.zip](https://github.com/lin380/project_template/archive/refs/heads/main.zip)

<sup>31</sup><https://github.com/git-guides/install-git>

<sup>32</sup>[https://github.com/signup?ref\\_cta=Sign+up&ref\\_loc=header+logged+out&ref\\_page=%2F&source=header-home](https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home)

<sup>33</sup><https://docs.github.com/en/get-started/quickstart/fork-a-repo#forking-a-repository>

<sup>34</sup><https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/cloning-a-repository-from-github/cloning-a-repository#cloning-a-repository>

It will take some time to complete, when it does the prompt (>) in the console will return. Then navigate to and open `docs/index.html` in a browser.

Once you have confirmed that the project template builds, then you can begin to configure the template to reflect your project. There are a few files to consider first. These files are places where the title of your project should appear.

- `README.md`
- `_pipeline.R`
- `analysis/index.Rmd`

After updating these files, build the project again and make sure that the new changes appear as you would like them. You are now ready to start your research project!

---

## 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.

This chapter concludes the Orientation section of this textbook. At this point the fundamental characteristics of research are in place to move a project towards implementation. The next section, Preparation, aims to cover the acquisition, curation, and transformation of 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. Following the Preparation section our attention will turn to the implementation of the three analysis approaches we have covered: inference, prediction, and exploration. Throughout these next sections we will maintain our aim to develop methodological and communicable research by connecting our implementation process to reproducible programming strategies.

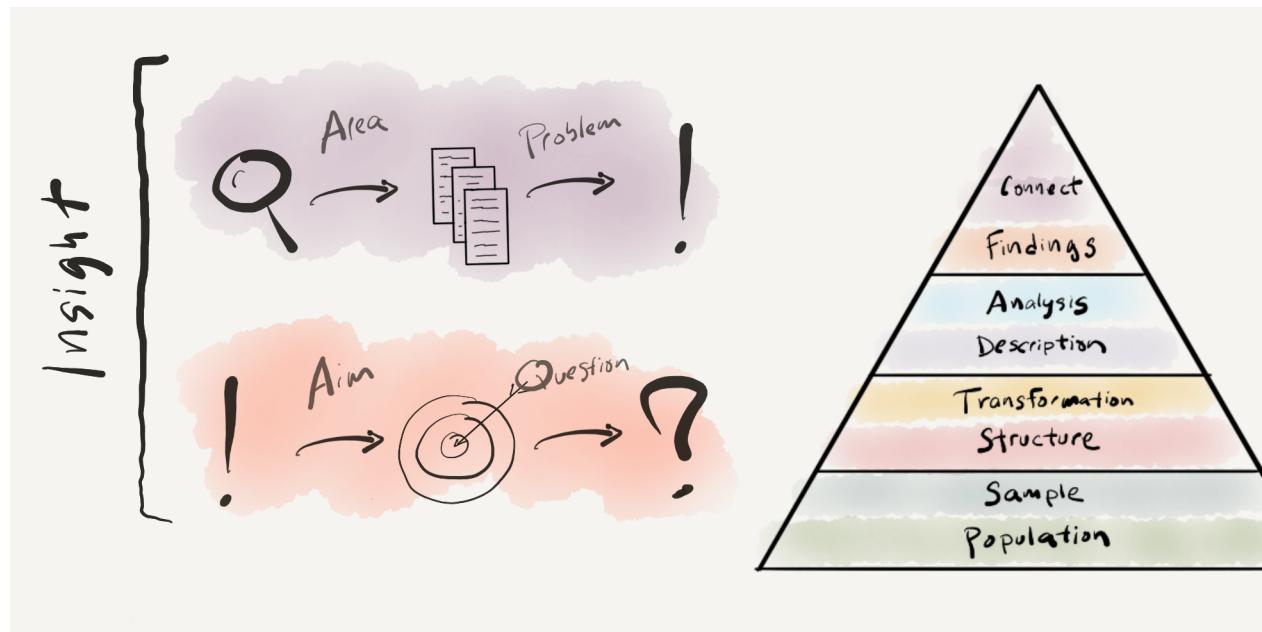


Figure 4.1: Framing research: visual summary

## Activities

### 💡 Recipe

**What:** Project management with Git, GitHub, and RStudio Cloud<sup>a</sup>  
**How:** Read Recipe 5 and participate in the Hypothes.is online social annotation.

**Why:** To learn how to use Git, GitHub, and RStudio to manage, store, and publish reproducible research projects.

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_5.html](https://lin380.github.io/tadr/articles/recipe_5.html)

### 💡 Lab

**What:** Project management with Git, GitHub, and RStudio Cloud<sup>a</sup>  
**How:** Clone, fork, and complete the steps in Lab 5.  
**Why:** To set up a GitHub account, fork and copy a GitHub repository

to RStudio Cloud, and use R, Git, and GitHub to manage, store, and publish changes to a reproducible research project.

<sup>a</sup>[https://github.com/lin380/lab\\_5](https://github.com/lin380/lab_5)

---

## Questions

### Conceptual questions

- What is the difference between a research question and a research hypothesis?
- What is the difference between a research design and a research plan?
- 

### Technical exercises

1. Matching research questions with data sources
2. Matching research questions with research designs
3. Preregistering a research project (?)
4. Propose a quantitative research topic (or question if possible).  
Support your topic with supporting literature. (?)



---

---

## Part III

# Preparation



At this point we will turn our attention to implementing the specifics outlined in our research blueprint. This section will group the components which concern the acquisition, curation, and transformation of data into a dataset which is prepared to be submitted to analysis. In each of these three 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 downloads, working with APIs, and webscraping. 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.



# 5

---

## *Acquire data*

---

The scariest moment is always just before you start.

– Stephen King

### Keys

- What are the most common strategies for acquiring corpus data?
- What programmatic steps can we take to ensure the acquisition process is reproducible?
- What is the importance of documenting data?

There are three main ways to acquire corpus data using R that I will introduce you to: **downloads**, **APIs**, and **web scraping**. In this chapter we will start by manually and programmatically downloading a corpus as it is the most straightforward process for the novice R programmer and typically incurs the least number of steps. Along the way I will introduce some key R coding concepts including control statements and custom functions. Next I will cover using R packages to interface with APIs, both open-access and authentication-based. APIs will require us to delve into more detail about R objects and custom functions. Finally acquiring data from the web via webscraping is the most idiosyncratic and involves both knowledge of the web, more sophisticated R skills, and often some clever hacking skills. I will start with a crash course on the structure of web documents (HTML) and then scale up to a real-world example. To round out the chapter we will cover the process of ensuring that our data is documented in such a way as to provide sufficient information to understand its key sampling characteristics and the source from which it was drawn.

### Swirl

**What:** Loops and vectorization<sup>a</sup>

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

**Why:** To ....

<sup>a</sup><https://github.com/lin380/swirl>

## 5.1 Downloads

### 5.1.1 Manual

The first acquisition method I will cover here is inherently non-reproducible from the standpoint that the programming implementation cannot acquire the data based solely on running the project code itself. In other words, it requires manual intervention. Manual downloads are typical for data resources which are not openly accessible on the public facing web. These can be resources that require institutional or private licensing (Language Data Consortium<sup>1</sup>, International Corpus of English<sup>2</sup>, BYU Corpora<sup>3</sup>, etc.), require authorization/registration (The Language Archive<sup>4</sup>, COW Corpora<sup>5</sup>, etc.), and/ or are only accessible via resource search interfaces (Corpus of Spanish in Southern Arizona<sup>6</sup>, Corpus Escrito del Español como L2 (CEDEL2)<sup>7</sup>, etc.).

Let's work with the CEDEL2 corpus (Lozano 2009) which provides a search interface and open access to the data through the search interface. The home-page can be seen in Figure 5.1.

Following the search/ download link you can find a search interface that allows the user to select the sub-corpus of interest. I've selected the subcorpus "Learners of L2 Spanish" and specified the L1 as English.

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 called `cedel2/`.

<sup>1</sup><https://www.ldc.upenn.edu/>

<sup>2</sup><http://ice-corpora.net/ice/>

<sup>3</sup><https://www.corpusdata.org/>

<sup>4</sup><https://archive.mpi.nl/tla/>

<sup>5</sup><https://www.webcorpora.org/>

<sup>6</sup><https://cesa.arizona.edu/>

<sup>7</sup><http://cedel2.learnercorpora.com/>



Figure 5.1: CEDEL2 Corpus homepage

```
data/
  derived
  original
  cedel2
  texts.csv
```

Other resources will inevitably include unique processes to obtaining the data, but in the end the data should be archived in the research structure in the `data/original/` directory and be treated as ‘read-only’.

### 5.1.2 Programmatic

There are many resources that provide corpus data is directly accessible for which programmatic approaches can be applied. Let’s take a look at how this works starting with the a sample from the Switchboard Corpus, a corpus of 2,400 telephone conversations by 543 speakers. First we navigate to the site with a browser and download the file that we are looking for. In this case I found the Switchboard Corpus on the NLTK data repository site<sup>8</sup>. More often than not this file will be some type of compressed archive file with an extension such as `.zip` or `.tz`, which is the case here. Archive files make downloading large single files or multiple files easy by grouping files and directories into one file. In R we can used the `download.file()` function from the base R library<sup>9</sup>. There are a number of **arguments** that a function may require or provide optionally. The `download.file()` function minimally requires two: `url` and

<sup>8</sup>[http://www.nltk.org/nltk\\_data/](http://www.nltk.org/nltk_data/)

<sup>9</sup>Remember base R packages are installed by default with R and are loaded and accessible by default in each R session.

Filename	L1	Age score (%)	Placement test	Proficiency	Proficiency (self-assessment)	Age of exposure to Spanish	Years studying Spanish	Stay abroad (months)	Task type	Medium
1 EN_WR_5_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_2K_2_2_TB	English	26	16.3	Lower beginner	2	18	2	0	2. Famous person	Written
4 EN_WR_8_16_3_1_KJ0	English	16	18.6	Lower beginner	2.5	14	3	0	1. Region where you live	Written

Figure 5.2: Search and download interface for the CEDEL2 Corpus

`destfile`. That is the file to download and the location where it is to be saved to disk.

```
# Download .zip file and write to disk
download.file(
  url =
  "https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/corpora/switc
  destfile = "../data/original/switchboard.zip"
)
```

As we can see looking at the directory structure for `data/` the `switchboard.zip` file has been downloaded.

```
data
  derived
  original
    switchboard.zip
```

Once an archive file is downloaded, however, the file needs to be ‘decompressed’ to reveal the file structure. To decompress this file we use the `unzip()` function with the arguments `zipfile` pointing to the `.zip` file and `exdir` specifying the directory where we want the files to be extracted to.

```
# Decompress .zip file and extract to our target directory
unzip(
  zipfile = "../data/original/switchboard.zip",
```

```
    exdir = "../data/original/"
)
```

The directory structure of `data/` now should look like this:

```
data
  derived
  original
    switchboard
      README
      discourse
      disfluency
      tagged
      timed-transcript
      transcript
    switchboard.zip
```

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 `switchboard.zip` 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 serve leading to unnecessary data transfer and server traffic. Let's tackle each of these issues in turn.

To avoid writing the `switchboard.zip` file to disk (long-term) we can use the `tempfile()` function to open a temporary holding space for the file. This space can then be used to store the file, unzip it, and then the temporary file will be destroyed. We assign the temporary space to an R object we will name `temp` with the `tempfile()` function. This object can now be used as the value of the argument `destfile` in the `download.file()` function. Let's also assign the web address to another object `url` which we will use as the value of the `url` argument.

```
# Create a temporary file space for our .zip file
temp <- tempfile()
# Assign our web address to `url`
url <-
  ↵ "https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/corpora/switchb
# Download .zip file and write to disk
download.file(url, temp)
```

 Tip

In the previous code I've used the values stored in the objects `url` and `temp` 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` as the value for the argument `zipfile` from the `unzip()` function. I've assigned our target directory path to `target_dir` and used it as the value for the argument `exdir` to prepare us for the next tweak on our approach.

```
# Assign our target directory to `target_dir`
target_dir <- "../data/original/"
# Decompress .zip file and extract to our target directory
unzip(
  zipfile = temp,
  exdir = target_dir
)
```

Our directory structure now looks like this:

```
data
  derived
  original
    switchboard
      README
      discourse
      disfluency
      tagged
      timed-transcript
      transcript
```

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 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.

To achieve this we need to introduce two new functions `if()` and `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. `if()` evaluates logical statements and processes subsequent code based on the logical value it is passed as an argument. Let's look at a toy example.

```
num <- 1
if (num == 1) {
  cat(num, "is 1")
} else {
  cat(num, "is not 1")
}
```

```
> 1 is 1
```

I assigned `num` to the value `1` and created a logical evaluation `num ==` whose result is passed as the argument to `if()`. If the statement returns `TRUE` then the code withining the first set of curly braces `{...}` is run. If `num == 1` is false, like in the code below, the code withining the braces following the `else` will be run.

```
num <- 2
if (num == 1) {
  cat(num, "is 1")
} else {
  cat(num, "is not 1")
}
```

```
> 2 is not 1
```

The function `if()` is one of various functions that are called **control statements**. Theses functions provide a lot of power to make dynamic choices as code is run.

Before we get back to our key objective to avoid downloading resources that we already have on disk, let me introduce another strategy to making code more powerful and ultimately more efficient and as well as more legible –the **custom function**. Custom functions are functions that the user writes to create a set of procedures that can be run in similar contexts. I've created a custom function named `eval_num()` below.

```
eval_num <- function(num) {
  if (num == 1) {
```

```
    cat(num, "is 1")
} else {
    cat(num, "is not 1")
}
}
```

Let's take a closer look at what's going on here. The function `function()` creates a function in which the user decides what arguments are necessary for the code to perform its task. In this case the only necessary argument is the object to store a numeric value to be evaluated. I've called it `num` because it reflects the name of the object in our toy example, but there is nothing special about this name. It's only important that the object names be consistently used. I've included our previous code (except for the hard-coded assignment of `num`) inside the curly braces and assigned the entire code chunk to `eval_num`.

We can now use the function `eval_num()` to perform the task of evaluating whether a value of `num` is or is not equal to 1.

```
> 1 is 1  
  
eval_num(num = 2)  
  
> 2 is not 1  
  
eval_num(num = 3)  
  
> 3 is not 1
```

I've put these coding strategies together with our previous code in a custom function I named `get_zip_data()`. There is a lot going on here. Take a look first and see if you can follow the logic involved given what you now know.

```
get_zip_data <- function(url, target_dir) {  
  # Function: to download and decompress a .zip file to a  
  # target directory  
  
  # Check to see if the data already exists  
  if (!dir.exists(target_dir)) { # if data does not exist,  
    # download/ decompress  
    cat("Creating target data directory \n") # print  
    # status message
```

```

dir.create(path = target_dir, recursive = TRUE,
    ↵ showWarnings = FALSE) # create target data
    ↵ directory
cat("Downloading data... \n") # print status message
temp <- tempfile() # create a temporary space for the
    ↵ file to be written to
download.file(url = url, destfile = temp) # download
    ↵ the data to the temp file
unzip(zipfile = temp, exdir = target_dir, junkpaths =
    ↵ TRUE) # decompress the temp file in the target
    ↵ directory
cat("Data downloaded! \n") # print status message
} else { # if data exists, don't download it again
    cat("Data already exists \n") # print status message
}
}

```

OK. You should have recognized the general steps in this function: the argument `url` and `target_dir` specify where to get the data and where to write the decompressed files, the `if()` statement evaluates whether the data already exists, if not (`!dir.exists(target_dir)`) then the data is downloaded and decompressed, if it does exist (`else`) then it is not downloaded.

### Tip

The prefixed `!` in the logical expression `dir.exists(target_dir)` returns the opposite logical value. This is needed in this case so when the target directory exists, the expression will return `FALSE`, not `TRUE`, and therefore not proceed in downloading the resource.

There are a couple key tweaks I've added that provide some additional functionality. For one I've included the function `dir.create()` to create the target directory where the data will be written. I've also added an additional argument to the `unzip()` function, `junkpaths = TRUE`. Together these additions allow the user to create an arbitrary directory path where the files, and only the files, will be extracted to on our disk. This will discard the containing directory of the `.zip` file which can be helpful when we want to add multiple `.zip` files to the same target directory.

A practical scenario where this applies is when we want to download data from a corpus that is contained in multiple `.zip` files but still maintain these files in a single primary data directory. Take for example the Santa Barbara Corpus<sup>10</sup>. This corpus resource includes a series of interviews in which there is

---

<sup>10</sup><http://www.linguistics.ucsb.edu/research/santa-barbara-corpus>

one .zip file, SBCorpus.zip which contains the transcribed interviews<sup>11</sup> and another .zip file, metadata.zip which organizes the meta-data<sup>12</sup> associated with each speaker. Applying our initial strategy to download and decompress the data will lead to the following directory structure:

```

data
  derived
  original
    SBCorpus
      TRN
      __MACOSX
        TRN
    metadata
      __MACOSX

```

By applying our new custom function `get_zip_data()` to the transcriptions and then the meta-data we can better organize the data.

```

# Download corpus transcriptions
get_zip_data(
  url =
    ↵ "http://www.linguistics.ucsb.edu/sites/secure.lsit.ucsb.edu.ling.d7/files/sitefil
    ↵
    target_dir = "../data/original/sbc/transcriptions/"
)

# Download corpus meta-data
get_zip_data(
  url =
    ↵ "http://www.linguistics.ucsb.edu/sites/secure.lsit.ucsb.edu.ling.d7/files/sitefil
    ↵
    target_dir = "../data/original/sbc/meta-data/"
)

data
  derived
  original
    sbc

```

<sup>11</sup><http://www.linguistics.ucsb.edu/sites/secure.lsit.ucsb.edu.ling.d7/files/sitefiles/research/SBC/SBCorpus.zip>

<sup>12</sup><http://www.linguistics.ucsb.edu/sites/secure.lsit.ucsb.edu.ling.d7/files/sitefiles/research/SBC/metadata.zip>

```
meta-data
transcriptions
```

If we add data from other sources we can keep them logical separate and allow our data collection to scale without creating unnecessary complexity. Let's add the Switchboard Corpus sample using our `get_zip_data()` function to see this in action.

```
# Download corpus
get_zip_data(
  url =
    ↵ "https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/packages/corpora/switc
    ↵
    target_dir = "../data/original/scs/"
)

data
derived
original
sbc
  meta-data
  transcriptions
scs
  README
  discourse
  disfluency
  tagged
  timed-transcript
  transcript
```

At this point we have what we need to continue to the next step in our data analysis project. But before we go, we should do some housekeeping to document and organize this process to make our work reproducible. We will take advantage of the `project-template` directory structure, seen below.

```
README.md
_pipeline.R
analysis
  1_acquire_data.Rmd
  2_curate_dataset.Rmd
  3_transform_dataset.Rmd
  4_analyze_dataset.Rmd
```

```
5_generate_article.Rmd  
_session-info.Rmd  
_site.yml  
index.Rmd  
references.bib  
data  
  derived  
  original  
    sbc  
    scs  
functions  
output  
  figures  
  results
```

First it is good practice to separate custom functions from our processing scripts. We can create a file in our `functions/` directory named `acquire_functions.R` and add our custom function `get_zip_data()` there.

### ⚠ Tip

Note that that the `acquire_functions.R` file is an R script, not an Rmarkdown document. Therefore code chunks that are used in `.Rmd` files are not used, only the R code itself.

We then use the `source()` function to read that function into our current script to make it available to use as needed. It is good practice to source your functions in the SETUP section of your script.

```
# Load custom functions for this project  
source(file = "../functions/acquire_functions.R")
```

In this section, to sum up, we've covered how to access, download, and organize data contained in .zip files; the most common format for language data found on repositories and individual sites. This included an introduction to a few key R programming concepts and strategies including using functions, writing custom functions, and controlling program flow with control statements. Our approach was to gather data while also keeping in mind the reproducibility of the code. To this end I introduced programming strategies for avoiding unnecessary web traffic (downloads), scalable directory creation, and data documentation.

**⚠ Tip**

The custom function `get_zip_data()` works with `.zip` files. There are many other compressed file formats (e.g. `.gz`, `.tar`, `.tgz`), however. In the R package `tadr` that accompanies this coursebook, a modified version of the `get_zip_data()` function, `get_compressed_data()`, extends the same logic to deal with a wider range of compressed file formats, including `.zip` files.

Explore this function's documentation (`?tadr::get_compressed_data()`) and/ or view the code (`tadr::get_compressed_data`) to better understand this function.

---

## 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 possible. For example, `gutenbergr`<sup>13</sup> provides access to Project Gutenberg, `rtweet`<sup>14</sup> to Twitter, and `Rfacebook`<sup>15</sup> to Facebook.<sup>16</sup>

### 5.2.1 Open access

Using R package interfaces, however, often requires some more knowledge about R objects and functions. Let's take a look at how to access data from Project Gutenberg through the `gutenbergr` package. Along the way we will touch upon various functions and concepts that are key to working with the R data types vectors and data frames including filtering and writing tabular data to disk in plain-text format.

To get started let's install and/ or load the `gutenbergr` package. If a package is not part of the R base library, we cannot assume that the user will have the package in their library. The standard approach for installing and then

<sup>13</sup><https://CRAN.R-project.org/package=gutenbergr>

<sup>14</sup><https://CRAN.R-project.org/package=rtweet>

<sup>15</sup><https://CRAN.R-project.org/package=Rfacebook>

<sup>16</sup>See Section @ref(sources) for a list of some other API packages.

loading a package is by using the `install.packages()` function and then calling `library()`.

```
install.packages("gutenbergr") # install `gutenbergr`  
#   ↵ package  
library(gutenbergr) # load the `gutenbergr` package
```

This approach works just fine, but luck has it that there is an R package for installing and loading packages! The `pacman`<sup>17</sup> package includes a set of functions for managing packages. A very useful one is `p_load()` which will look for a package on a system, load it if it is found, and install and then load it if it is not found. This helps potentially avoid using unnecessary bandwidth to install packages that may already exist on a user's system. But, to use `pacman` we need to include the code to install and load it with the functions `install.packages()` and `library()`. I've included some code that will mimic the behavior of `p_load()` for installing `pacman` itself, but as you can see it is not elegant, luckily it's only used once as we add it to the SETUP section of our master file, `_pipeline.R`.

```
# Load `pacman`. If not installed, install then load.  
if (!require("pacman", character.only = TRUE)) {  
  install.packages("pacman")  
  library("pacman", character.only = TRUE)  
}
```

Now that we have `pacman` installed and loaded into our R session, let's use the `p_load()` function to make sure to install/ load the two packages we will need for the upcoming tasks. If you are following along with the `project_template`, add this code within the SETUP section of the `1_acquire_data.Rmd` file.

```
# Script-specific options or packages  
pacman::p_load(tidyverse, gutenbergr)
```

### ⚠ Tip

Note that the arguments `tidyverse` and `gutenbergr` are comma-separated but not quoted when using `p_load()`. When using `install.packages()` to install, package names need to be quoted (character strings). `library()` can take quotes or no quotes, but only one package at a time.

---

<sup>17</sup><https://CRAN.R-project.org/package=pacman>

Project Gutenberg provides access to thousands of texts in the public domain. The `gutenbergr` package contains a set of tables, or **data frames** in R speak, that index the meta-data for these texts broken down by text (`gutenberg_metadata`), author (`gutenberg_authors`), and subject (`gutenberg_subjects`). I'll use the `glimpse()` function loaded in the `tidyverse`<sup>18</sup> package<sup>19</sup> to summarize the structure of these data frames.

```

glimpse(gutenberg_metadata) # summarize text meta-data

> Rows: 69,199
> Columns: 8
> $ gutenberg_id      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, ~
> $ title              <chr> "The Declaration of Independence of the United Sta~
> $ author              <chr> "Jefferson, Thomas", "United States", "Kennedy, Jo~
> $ gutenberg_author_id <int> 1638, 1, 1666, 3, 1, 4, NA, 3, 3, NA, 7, 7, 7, 8, ~
> $ language             <chr> "en", "en", "en", "en", "en", "en", "en", "en", "e~
> $ gutenberg_bookshelf <chr> "Politics/American Revolutionary War/United States~
> $ rights               <chr> "Public domain in the USA.", "Public domain in the~
> $ has_text             <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TR~

glimpse(gutenberg_authors) # summarize authors meta-data

> Rows: 21,323
> Columns: 7
> $ gutenberg_author_id <int> 1, 3, 4, 5, 7, 8, 9, 10, 12, 14, 16, 17, 18, 20, 2~
> $ author                <chr> "United States", "Lincoln, Abraham", "Henry, Patri~
> $ alias                 <chr> "U.S.A.", NA, NA, NA, "Dodgson, Charles Lutwidge", ~
> $ birthdate              <int> NA, 1809, 1736, 1849, 1832, NA, 1819, 1860, 1805, ~
> $ deathdate              <int> NA, 1865, 1799, 1931, 1898, NA, 1891, 1937, 1844, ~
> $ wikipedia              <chr> "https://en.wikipedia.org/wiki/United_States", "ht~
> $ aliases                <chr> NA, "United States President (1861-1865)/Lincoln, ~

glimpse(gutenberg_subjects) # summarize subjects meta-data

> Rows: 230,993
> Columns: 3
> $ gutenberg_id <int> 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, ~
> $ subject_type <chr> "lcsh", "lcsh", "lcc", "lcc", "lcsh", "lcsh", "lcc", "lcc~
> $ subject        <chr> "United States -- History -- Revolution, 1775-1783 -- Sou~
```

<sup>18</sup><https://CRAN.R-project.org/package=tidyverse>

<sup>19</sup>`tidyverse` is not a typical package. It is a set of packages: `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, and `tibble`. These packages are all installed/ loaded with `tidyverse` and form the backbone for the type of work you will typically do in most analyses.

 Tip

The `gutenberg_metadata`, `gutenberg_authors`, and `gutenberg_subjects` are periodically updated. To check to see when each data frame was last updated run:

```
attr(gutenberg_metadata, "date_updated")
```

To download the text itself we use the `gutenberg_download()` function which takes one required argument, `gutenberg_id`. The `gutenberg_download()` function is what is known as ‘vectorized’, that is, it can take a single value or multiple values for the argument `gutenberg_id`. Vectorization refers to the process of applying a function to each of the elements stored in a **vector** –a primary object type in R. A vector is a grouping of values of one of various types including character (`chr`), integer (`int`), double (`dbl`), and logical (`lg1`) and a data frame is a grouping of vectors. The `gutenberg_download()` function takes an integer vector which can be manually added or selected from the `gutenberg_metadata` or `gutenberg_subjects` data frames using the `$` operator (e.g. `gutenberg_metadata$gutenberg_id`).

Let’s first add them manually here as a toy example by generating a vector of integers from 1 to 5 assigned to the variable name `ids`.

```
ids <- 1:5 # integer vector of values 1 to 5
ids

> [1] 1 2 3 4 5
```

To download the works from Project Gutenberg corresponding to the `gutenberg_ids` 1 to 5, we pass the `ids` object to the `gutenberg_download()` function.

```
works_sample <- gutenberg_download(gutenberg_id = ids) #
  ↪ download works with `gutenberg_id` 1-5
glimpse(works_sample) # summarize `works` dataset

> Rows: 2,959
> Columns: 2
> $ gutenberg_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
> $ text <chr> "December, 1971 [Etext #1]", "", "", "The Project Gutenb~
```

Two attributes are returned: `gutenberg_id` and `text`. The `text` column contains values for each line of text (delimited by a carriage return) for each of the 5 works we downloaded. There are many more attributes available from the Project Gutenberg API that can be accessed by passing a character vector

of the attribute names to the argument `meta_fields`. The column names of the `gutenberg_metadata` data frame contains the available attributes.

```
names(gutenberg_metadata) # print the column names of the
                           ↴ `gutenberg_metadata` data frame

> [1] "gutenberg_id"          "title"           "author"
> [4] "gutenberg_author_id"   "language"        "gutenberg_bookshelf"
> [7] "rights"                "has_text"
```

Let's augment our previous download with the title and author of each of the works. To create a character vector we use the `c()` function, then, quote and delimit the individual elements of the vector with a comma.

```
# download works with `gutenberg_id` 1-5 including `title`
                           ↴ and `author` as attributes
works_sample <-
  gutenberg_download(
    gutenberg_id = ids,
    meta_fields = c("title", "author")
  ) #

glimpse(works_sample) # summarize dataset

> Rows: 2,959
> Columns: 4
> $ gutenberg_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
> $ text          <chr> "December, 1971 [Etext #1]", "", "", "The Project Gutenb~n
> $ title         <chr> "The Declaration of Independence of the United States of ~
> $ author        <chr> "Jefferson, Thomas", "Jefferson, Thomas", "Jefferson, Tho~
```

Now, in a more practical scenario we would like to select the values of `gutenberg_id` by some principled query such as works from a specific author, language, or subject. To do this we first query either the `gutenberg_metadata` data frame or the `gutenberg_subjects` data frame. Let's say we want to download a random sample of 10 works from English Literature (Library of Congress Classification, "PR"). Using the `dplyr::filter()` function (`dplyr` is part of the `tidyverse` package set) we first extract all the Gutenberg ids from `gutenberg_subjects` where `subject_type == "lcc"` and `subject == "PR"` assigning the result to `ids`.<sup>20</sup>

---

<sup>20</sup>See Library of Congress Classification<sup>21</sup> documentation for a complete list of subject codes.

Tip

The operators = and == are not equivalents. == is used for logical evaluation and = is an alternate notation for variable assignment (<-).

The `gutenberg_subjects` data frame does not contain information as to whether a `gutenberg_id` is associated with a plain-text version. To limit our query to only those English Literature works with text, we filter the `gutenberg_metadata` data frame by the ids we have selected in `ids` and the attribute has `text` in the `gutenberg_metadata` data frame.

```
# Filter for only those works that have text
ids_has_text <-
  filter(gutenberg_metadata,
         gutenberg_id %in% ids$gutenberg_id,
         has_text == TRUE)
glimpse(ids_has_text)

> Rows: 9,548
> Columns: 8
> $ gutenberg_id      <int> 11, 12, 13, 16, 20, 26, 27, 35, 36, 42, 43, 46, 58~
> $ title              <chr> "Alice's Adventures in Wonderland", "Through the L~
> $ author              <chr> "Carroll, Lewis", "Carroll, Lewis", "Carroll, Lewi~
> $ gutenberg_author_id <int> 7, 7, 7, 10, 17, 17, 23, 30, 30, 35, 35, 37, 17, 4~
> $ language            <chr> "en", "en", "en", "en", "en", "en", "en", "en", "e~
> $ gutenberg_bookshelf <chr> "Children's Literature", "Best Books Ever Listings~
> $ rights              <chr> "Public domain in the USA.", "Public domain in the~
> $ has_text            <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TR~
```

 Tip

A couple R programming notes on the code phrase `gutenberg_id %in% ids$gutenberg_id`. First, the `$` symbol in `ids$gutenberg_id` is the programmatic way to target a particular column in an R data frame. In this example we select the `ids` data frame and the column `gutenberg_id`, which is a integer vector. The `gutenberg_id` variable that precedes the `%in%` operator does not need an explicit reference to a data frame because the primary argument of the `filter()` function is this data frame (`gutenberg_metadata`). Second, the `%in%` operator logically evaluates whether the vector elements in `gutenberg_metadata$gutenberg_ids` are also found in the vector `ids$gutenberg_id` returning `TRUE` and `FALSE` accordingly. This effectively filters those ids which are not in both vectors.

As we can see the number of works with text is fewer than the number of works listed, 9900 versus 9548. Now we can safely do our random selection of 10 works, with the function `slice_sample()` and be confident that the ids we select will contain text when we take the next step by downloading the data.

```
set.seed(123) # make the sampling reproducible
ids_sample <- slice_sample(ids_has_text, n = 10) # sample
  ↵ 10 works
glimpse(ids_sample) # summarize the dataset

> Rows: 10
> Columns: 8
> $ gutenberg_id      <int> 10542, 10734, 60253, 13776, 7532, 67002, 16604, 28~
> $ title              <chr> "The Boats of the \"Glen Carrig\"\\r\\nBeing an acco~
> $ author              <chr> "Hodgson, William Hope", NA, "Orczy, Emmuska Orczy~
> $ gutenberg_author_id <int> 3260, NA, 45, NA, NA, 30, 3579, 6137, 797, 1865
> $ language             <chr> "en", "en", "en", "en", "en", "hu", "en", "en", "e~
> $ gutenberg_bookshelf <chr> "Horror", NA, NA, NA, NA, NA, "Humor", NA, NA
> $ rights               <chr> "Public domain in the USA.", "Public domain in the~
> $ has_text            <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TR~

works_pr <- gutenberg_download(gutenberg_id =
  ↵  ids_sample$gutenberg_id,
                                meta_fields = c("author",
                                ↵  "title"))
glimpse(works_pr) # summarize the dataset
```

```

works_pr <- gutenberg_download(gutenberg_id =
  ↵  ids_sample$gutenberg_id,
                                meta_fields = c("author",
                                ↵  "title"))
write_rds(works_pr, file =
  ↵  "data/acquire-data/gutenberg_works_pr.rds")

> Rows: 86,086
> Columns: 4
> $ gutenberg_id <int> 7532, 7532, 7532, 7532, 7532, 7532, 7532, 7532, 753~  

> $ text          <chr> "A BOOK OF OLD BALLADS", "", "Selected and with an Introd~  

> $ author        <chr> NA, N~  

> $ title         <chr> "A Book of Old Ballads -  

Volume 2", "A Book of Old Ballad~"

```

At this point we have data and could move on to processing this dataset in preparation for analysis. However, we are aiming for a reproducible workflow and this code does not conform to our principle of modularity: each subsequent step in our analysis will depend on running this code first. Furthermore, running this code as it is creates issues with bandwidth, as in our previous examples from direct downloads. To address modularity we will write the dataset to disk in **plain-text format**. In this way each subsequent step in our analysis can access the dataset locally. To address bandwidth concerns, we will devise a method for checking to see if the dataset is already downloaded and skip the download, if possible, to avoid accessing the Project Gutenberg server unnecessarily.

To write our data frame to disk we will export it into a standard plain-text format for two-dimensional datasets: a CSV file (comma-separated value). The CSV structure for this dataset will look like this:

```

works_pr |> head() |> format_csv() |> cat()

> gutenberg_id,text,author,title
> 7532,A BOOK OF OLD BALLADS,NA,A Book of Old Ballads - Volume 2
> 7532,,NA,A Book of Old Ballads - Volume 2
> 7532,Selected and with an Introduction,NA,A Book of Old Ballads -
Volume 2
> 7532,,NA,A Book of Old Ballads - Volume 2
> 7532,by,NA,A Book of Old Ballads - Volume 2
> 7532,,NA,A Book of Old Ballads - Volume 2

```

The first line contains the names of the columns and subsequent lines the observations. Data points that contain commas themselves (e.g. “Shaw, Bernard”)

are quoted to avoid misinterpreting these commas as delimiters in our data. To write this dataset to disk we will use the `reader::write_csv()` function.

```
write_csv(works_pr, file =
  ↵  "../data/original/gutenberg_works_pr.csv")
```

To avoid downloading dataset that already resides on disk, let's implement a similar strategy to the one used for direct downloads (`get_zip_data()`). I've incorporated the code for sampling and downloading data for a particular subject from Project Gutenberg with a control statement to check if the dataset file already exists into a function I named `get_gutenberg_subject()`. Take a look at this function below.

```
get_gutenberg_subject <- function(subject, target_file,
  ↵  sample_size = 10) {
  # Function: to download texts from Project Gutenberg
  ↵  with
  # a specific LCC subject and write the data to disk.

  pacman::p_load(tidyverse, gutenbergr) # install/load
  ↵  necessary packages

  # Check to see if the data already exists
  if(!file.exists(target_file)) { # if data does not
    ↵  exist, download and write
    target_dir <- dirname(target_file) # generate target
  ↵  directory for the .csv file
  dir.create(path = target_dir, recursive = TRUE,
    ↵  showWarnings = FALSE) # create target data
    ↵  directory
  cat("Downloading data... \n") # print status message
  # Select all records with a particular LCC subject
  ids <-
    filter(gutenberg_subjects,
      subject_type == "lcc", subject == subject) #
  ↵  select subject
  # Select only those records with plain text available
  set.seed(123) # make the sampling reproducible
  ids_sample <-
    filter(gutenberg_metadata,
      gutenberg_id %in% ids$gutenberg_id, # select
  ↵  ids in both data frames
```

```

    has_text == TRUE) |> # select those ids that
  ↵ have text
    slice_sample(n = sample_size) # sample N works
  # Download sample with associated `author` and `title`
  ↵ metadata
works_sample <-
  gutenberg_download(gutenberg_id =
    ↵ ids_sample$gutenberg_id,
    meta_fields = c("author",
    ↵ "title"))
  # Write the dataset to disk in .csv format
  write_csv(works_sample, file = target_file)
  cat("Data downloaded! \n") # print status message
} else { # if data exists, don't download it again
  cat("Data already exists \n") # print status message
}
}
}

```

Adding this function to our function script `functions/acquire_functions.R`, we can now source this function in our `analysis/1_acquire_data.Rmd` script to download multiple subjects and store them in on disk in their own file.

Let's download American Literature now (LCC code "PQ").

```

# Download Project Gutenberg text for subject 'PQ'
  ↵ (American Literature)
# and then write this dataset to disk in .rds format

# Select all records with a particular LCC subject
ids <-
  filter(gutenberg_subjects,
    subject_type == "lcc", subject == "PQ") # select
  ↵ subject
# Select only those records with plain text available
set.seed(123) # make the sampling reproducible
ids_sample <-
  filter(gutenberg_metadata,
    gutenberg_id %in% ids$gutenberg_id, # select ids
  ↵ in both data frames
    has_text == TRUE) |> # select those ids that have
  ↵ text
  slice_sample(n = sample_size) # sample N works
}

```

```

# Download sample with associated `author` and `title`
#   metadata
works_pq <-
  gutenberg_download(gutenberg_id =
    ids_sample$gutenberg_id,
    meta_fields = c("author", "title"))

write_rds(x = works_pq, file =
  "data/acquire-data/gutenberg_works_pq.rds")

# Download Project Gutenberg text for subject 'PQ'
#   (American Literature)
# and then write this dataset to disk in .csv format
get_gutenberg_subject(subject = "PQ", target_file =
  "../data/original/gutenberg/works_pq.csv")

```

Applying this function to both the English and American Literature datasets, our data directory structure now looks like this:

```

data
  derived
  original
    gutenberg
      works_pq.csv
      works_pr.csv
  sbc
    meta-data
    transcriptions
  scs
    README
    discourse
    disfluency
    documentation
    tagged
    timed-transcript
    transcript

```

### 5.2.2 Authentication

Some APIs and the R interfaces that provide access to them require authentication. This may either be through an interactive process that is mediated between R and the web service and/ or by visiting the developer website of

the particular API. In either case, there is an extra step that is necessary to make the connect to the API to access the data.

Let's take a look at the popular micro-blogging platform Twitter. The `rtweet` package (Kearney, Revilla Sancho, and Wickham 2023) provides access to tweets in various ways. To get started install and/or load the `rtweet` package.

```
pacman::p_load(rtweet) # install/load rtweet package
```

Now before a researcher can access data from Twitter with `rtweet`, an authentication token must be setup and made accessible<sup>22</sup>. After following the steps for setting up an authentication token and saving it, that token can be accessed with the `auth_as()` function.

```
auth_as(twitter_auth) # load the saved `twitter_auth`  
→ token
```

Now that we the R session is authenticated, we can explore a popular method for querying the Twitter API which searches tweets (`search_tweets`) posted in the recent past (6-9 days).

Let's look at a typical query using the `search_tweets()` function.

```
rt_latinx <-  
  search_tweets(q = "latinx", # query term  
                n = 100, # number of tweets desired  
                type = "mixed", # a mix of `recent` and  
                → `popular` tweets  
                include_rts = FALSE) # do not include RTs
```

Looking at the arguments in this function, we see I've specified the query term to be 'latinx'. This is a single word query but if the query included multiple words, the spaces between would be interpreted as the logical AND (only match tweets with all the individual terms). If one would like to include multi-word expressions, the expressions should be enclosed by single quotes (i.e. `q = "'spanish speakers' AND latinx"`). Another approach would be to include the logical OR (match tweets with either of the terms). Multi-word expressions can be included as in the previous case. Of note, hashtags are acceptable terms, so `q = "#latinx"` would match tweets with this hashtag.

The number of results has been set at '100', but this is the default, so I could have left it out. But you can increase the number of desired tweets. There are

---

<sup>22</sup><https://docs.ropensci.org/rtweet/articles/auth.html>

rate limits which cap the number of tweets you can access in a given 15-minute time period.

Another argument of importance is the `type` argument. This argument has three possible attributes `popular`, `recent`, and `mixed`. When the `popular` attribute is Twitter API will tend to return fewer tweets than specified by `n`. With `recent` or `mixed` you will most likely get the `n` you specified (note that `mixed` is a mix of `popular` and `recent`).

A final argument to note is the `include_rts` whose attribute is logical. If `FALSE` no retweets will be included in the results. This is often what a language researcher will want.

Now, once the `search_tweets` query has been run, there are a large number of variables that are included in the resulting data frame. Here's an overview of the names of the variables and the vector types for each variable.

The Twitter API documentation for the standard Search Tweets call<sup>23</sup>, which is what `search_tweets()` interfaces with has quite a few variables (35 to be exact). For many purposes it is not necessary to keep all the variables. Furthermore, since we will want to write a plain-text file to disk as part of our project, we will need to either convert or eliminate any of the variables that are marked as type `list`. The most common variable to convert is the `coordinates` variable, as it will contain the geolocation codes for those Twitter users' tweets captured in the query that have geolocation enabled on their device. It is of note, however, that using `search_tweets()` without specifying that only tweets with geocodes should be captured (`geocode =`) will tend to return very few, if any, tweets with geolocation information as the majority of Twitter users do not have geolocation enabled.

Let's assume that we want to keep all the variables that are not of type `list`. One option is to use `select()` and name each variable we want to keep. On the other hand we can use a combination of `select()` and negated `!where()` to select all the variables that are not lists (`is_list`). Let's do the later approach.

```
rt_latinx_subset <-
  rt_latinx |> # dataset
  select(!where(is_list)) # select all variables that are
  ↵ NOT lists

  rt_latinx_subset |> # subsetted dataset
  glimpse() # overview

> Rows: 100
```

---

<sup>23</sup><https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets>

Table 5.1: Variables and variable types returned from Twitter API via rtweet’s `search_tweets()` function.

created_at	character
id	double
id_str	character
full_text	character
truncated	logical
display_text_range	double
entities	list
metadata	list
source	character
in_reply_to_status_id	double
in_reply_to_status_id_str	character
in_reply_to_user_id	double
in_reply_to_user_id_str	character
in_reply_to_screen_name	character
geo	logical
coordinates	list
place	list
contributors	logical
is_quote_status	logical
retweet_count	integer
favorite_count	integer
favorited	logical
retweeted	logical
lang	character
possibly_sensitive	logical
quoted_status_id	double
quoted_status_id_str	character
quoted_status	list
text	character
favorited_by	logical
display_text_width	logical
retweeted_status	logical
quoted_status_permalink	logical
query	logical
possibly_sensitive_appealable	logical

```

> Columns: 30
> $ created_at
> $ id
> $ id_str
> $ full_text
> $ truncated
> $ display_text_range
> $ source
> $ in_reply_to_status_id
> $ in_reply_to_status_id_str
> $ in_reply_to_user_id
> $ in_reply_to_user_id_str
> $ in_reply_to_screen_name
> $ geo
> $ contributors
> $ is_quote_status
> $ retweet_count
> $ favorite_count
> $ favorited
> $ retweeted
> $ lang
> $ possibly_sensitive
> $ quoted_status_id
> $ quoted_status_id_str
> $ text
> $ favorited_by
> $ display_text_width
> $ retweeted_status
> $ quoted_status_permalink
> $ query
> $ possibly_sensitive_appealable

```

` "Sun Sep 26 17:38:06 +0000 2021", "Sun S~  
<dbl> 1.44e+18, 1.44e+18, 1.44e+18, 1.44e+18, ~  
<chr> "1442181701967302659", "1442196629801488~  
<chr> "If we call it Latinx Mass they can't ca~  
<lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE~  
<dbl> 57, 177, 166, 23, 261, 153, 202, 211, 57~  
<chr> "<a href=\"https://mobile.twitter.com\" ~  
<dbl> NA, NA, NA, 1.44e+18, NA, NA, NA, NA, 1.~  
<chr> NA, NA, NA, "1437436224042635269", NA, N~  
<dbl> NA, NA, NA, 4.26e+08, NA, NA, NA, NA, 2.~  
<chr> NA, NA, NA, "426159377", NA, NA, NA, NA, ~  
<chr> NA, NA, NA, "MorganStanley", NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, ~  
<int> 351, 124, 62, 0, 0, 0, 0, 0, 0, 0, 0, 1, ~  
<int> 3902, 898, 280, 0, 0, 0, 0, 0, 0, 7, 0, ~  
<lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE~  
<lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE~  
<chr> "en", "en", "es", "en", "en", "en"~  
<lgl> NA, FALSE, FALSE, FALSE, FALSE, FALSE, F~  
<dbl> NA, NA, NA, NA, 1.44e+18, NA, NA, NA, NA~  
<chr> NA, NA, NA, NA, "1442475408058830856", N~  
<chr> "If we call it Latinx Mass they can't ca~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~  
<lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, ~

Now we have the 30 variables which can be written to disk as a plain-text file. Let's go ahead and do this, but wrap it in a function that does all the work we've just laid out in one function. In addition we will check to see if the same query has been run, and skip running the query if the dataset is on disk.

```

write_search_tweets <-
  function(query, path, n = 100, type = "mixed",
  ↪ include_rts = FALSE) {
  # Function
  # Conduct a Twitter search query and write the results
  ↪ to a csv file

```

```
if(!file.exists(path)) { # check to see if the file
  ↵ already exists
  cat("File does not exist \n") # message

  library(rtweet) # to use Twitter API
  library(tidyverse) # to manipulate data

  auth_get() # get authentication token

  results <- # query results
  search_tweets(q = query, # query term
    n = n, # number of tweets desired
    ↵ (default 100)
    type = type, # type of query
    include_rts = include_rts) |> # to
    ↵ include RTs
  select(!where(is_list)) # remove list variables

  if(!dir.exists(dirname(path))) { # isolate directory
    ↵ and check if exists
    cat("Creating directory \n") # message

    dir.create(path = dirname(path), # isolate and
      ↵ create directory (remove file name)
      recursive = TRUE, # create embedded
      ↵ directories if necessary
      showWarnings = FALSE) # do not report
      ↵ warnings
  }

  write_csv(x = results, file = path) # write results
  ↵ to csv file
  cat("Twitter search results written to disk \n") # 
  ↵ message

} else {
  cat("File already exists! \n") # message
}
}
```

Let's run this function with the same query as above.

```
write_search_tweets(query = "latinx", path =
  ↴ "../data/original/twitter/rt_latinx.csv")
```

And the appropriate directory structure and file have been written to disk.

```
data/original/twitter/
  rt_latinx.csv
```

In sum, this subsection provided an overview to acquiring data from web service APIs through R packages. We took at closer look at the `gutenbergr` package which provides programmatic access to works available on Project Gutenberg and the `rtweet` package which provides authenticated access to Twitter. Working with package interfaces requires more knowledge of R including loading/ installing packages, working with vectors and data frames, and exporting data from an R session. We touched on these programming concepts and also outlined a method to create a reproducible workflow.

---

## 5.3 Web scraping

There are many resources available through manual and direct downloads from repositories and individual sites and R package interfaces to web resources with APIs, but these resources are relatively limited to the amount of public-facing textual data recorded on the web. In the case that you want to acquire data from webpages, R can be used to access the web programmatically through a process known as web scraping. The complexity of web scrapes can vary but in general it requires more advanced knowledge of R as well as the structure of the language of the web: HTML (Hypertext Markup Language).

### 5.3.1 A toy example

HTML is a cousin of XML (eXtensible Markup Language) and as such organizes web documents in a hierarchical format that is read by your browser as you navigate the web. Take for example the toy webpage I created as a demonstration in Figure 5.3.

The file accessed by my browser to render this webpage is `test.html` and in plain-text format looks like this:

```
<html>
  <head>
    <title>My website</title>
```

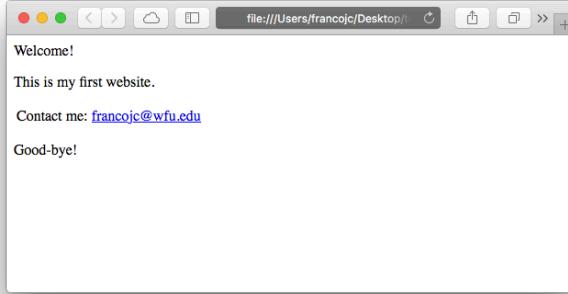


Figure 5.3: Example web page.

```
</head>
<body>
  <div class="intro">
    <p>Welcome!</p>
    <p>This is my first website. </p>
  </div>
  <table>
    <tr>
      <td>Contact me:</td>
      <td>
        <a href="mailto:francojc@wfu.edu">francojc@wfu.edu</a>
      </td>
    </tr>
  </table>
  <div class="conc">
    <p>Good-bye!</p>
  </div>
</body>
</html>
```

Each element in this file is delineated by an opening and closing tag, `<head></head>`. Tags are nested within other tags to create the structural hierarchy. Tags can take class and id labels to distinguish them from other tags and often contain other attributes that dictate how the tag is to behave when rendered visually by a browser. For example, there are two `<div>` tags in our toy example: one has the label `class = "intro"` and the other `class = "conc"`. `<div>` tags are often used to separate sections of a webpage that may require special visual formatting. The `<a>` tag, on the other hand, creates a web link. As part of this tag's function, it requires the attribute `href=` and a web protocol – in this case it is a link to an email address

`mailto:francojc@wfu.edu`. More often than not, however, the `href=` contains a URL (Uniform Resource Locator). A working example might look like this: `<a href="https://francojc.github.io/">My homepage</a>`.

The aim of a web scrape is to download the HTML file, parse the document structure, and extract the elements containing the relevant information we wish to capture. Let's attempt to extract some information from our toy example. To do this we will need the `rvest`<sup>24</sup>(Wickham 2022) package. First, install/load the package, then, read and parse the HTML from the character vector named `web_file` assigning the result to `html`.

```
pacman::p_load(rvest) # install/ load `rvest`  
  
html <- read_html(web_file) # read raw html and parse to  
#   ↳ xml  
html  
  
> {html_document}  
> <html>  
> [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...  
> [2] <body>\n      <div class="intro">\n          <p>Welcome!</p>\n          <p>This is ...
```

`read_html()` parses the raw HTML into an object of class `xml_document`. The summary output above shows that tags the HTML structure have been parsed into ‘elements’. The tag elements can be accessed by using the `html_elements()` function by specifying the tag to isolate.

```
html |>  
html_elements("div")  
  
> {xml_nodeset (2)}  
> [1] <div class="intro">\n          <p>Welcome!</p>\n          <p>This is my first web ...  
> [2] <div class="conc">\n          <p>Good-bye!</p>\n      </div>
```

Notice that `html_elements("div")` has returned both `div` tags. To isolate one of tags by its class, we add the class name to the tag separating it with a

..

```
html |>  
html_elements("div.intro")  
  
> {xml_nodeset (1)}  
> [1] <div class="intro">\n          <p>Welcome!</p>\n          <p>This is my first web ...
```

---

<sup>24</sup><https://CRAN.R-project.org/package=rvest>

Great. Now say we want to drill down and isolate the subordinate `<p>` nodes. We can add `p` to our node filter.

```
html |>
  html_elements("div.intro p")

> {xml_nodeset (2)}
> [1] <p>Welcome!</p>
> [2] <p>This is my first website. </p>
```

To extract the text contained within a node we use the `html_text()` function.

```
html |>
  html_elements("div.intro p") |>
  html_text()

> [1] "Welcome!"           "This is my first website. "
```

The result is a character vector with two elements corresponding to the text contained in each `<p>` tag. If you were paying close attention you might have noticed that the second element in our vector includes extra whitespace after the period. To trim leading and trailing whitespace from text we can add the `trim = TRUE` argument to `html_text()`.

```
html |>
  html_elements("div.intro p") |>
  html_text(trim = TRUE)

> [1] "Welcome!"           "This is my first website."
```

From here we would then work to organize the text into a format we want to store it in and write the results to disk. Let's leave writing data to disk for later in the chapter. For now keep our focus on working with `rvest` to acquire data from html documents working with a more practical example.

### 5.3.2 A practical example

With some basic understanding of HTML and how to use the `rvest` package, let's turn to a realistic example. Say we want to acquire lyrics from the online music website and database last.fm<sup>25</sup>. The first step in any web scrape is to investigate the site and page(s) we want to scrape to ascertain if there any licensing restrictions. Many, but not all websites, will include a plain text file `robots.txt`<sup>26</sup> at the root of the main URL. This file is declares which

---

<sup>25</sup><https://www.last.fm/>

<sup>26</sup><https://www.cloudflare.com/learning/bots/what-is-robots.txt/>

websites a ‘robot’ (including web scraping scripts) can and cannot access. We can use the `robotstxt` package to find out which URLs are accessible<sup>27</sup>.

```
pacman::p_load(robotstxt) # load/ install `robotstxt`  
  
paths_allowed(paths = "https://www.last.fm/") # check  
  ↳ permissions  
  
> [1] TRUE
```

The next step includes identifying the URL we want to target and exploring the structure of the HTML document. Take the following webpage I have identified, seen in Figure 5.4.

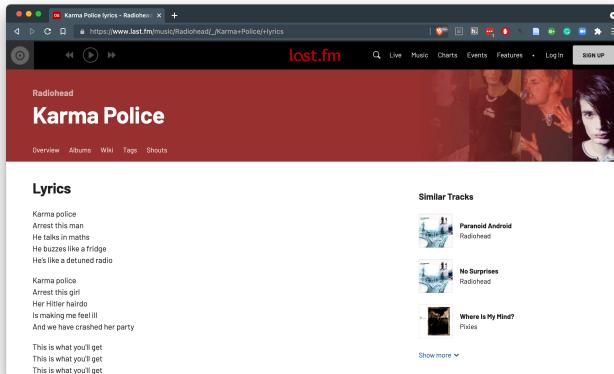


Figure 5.4: Lyrics page from last.fm

As in our toy example, first we want to feed the HTML web address to the `read_html()` function to parse the tags into elements. We will then assign the result to `html`.

```
# read and parse html as an xml object  
lyrics_url <-  
  ↳ "https://www.last.fm/music/Radiohead/_/Karma+Police/+lyrics"  
html <- read_html(lyrics_url) # read raw html and parse to  
  ↳ xml  
html  
  
> {html_document}
```

---

<sup>27</sup>It is important to check the paths of sub-domains as some websites allow access in some areas and not in others

```
> <html lang="en" class="no-js playbar-masthead-release-shim youtube-provider-not-ready">
> [1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
> [2] <body>\n<div id="initial-tealium-data" data-require="tracking/tealium-uta ...
```

At this point we have captured and parsed the raw HTML assigning it to the object named `html`. The next step is to identify the html elements that contain the information we want to extract from the page. To do this it is helpful to use a browser to inspect specific elements of the webpage. Your browser will be equipped with a command that you can enable by hovering your mouse over the element of the page you want to target and using a right click to select “Inspect” (Chrome) or “Inspect Element” (Safari, Brave). This will split your browser window vertical or horizontally showing you the raw HTML underlying the webpage.

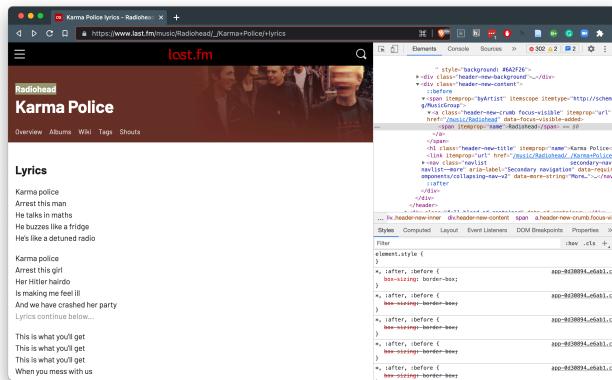


Figure 5.5: Using the “Inspect Element” command to explore raw html.

From Figure 5.5 we see that the element we want to target is contained within an `<a></a>` tag. Now this tag is common and we don’t want to extract every `a` so we use the class `header-new-crumb` to specify we only want the artist name. Using the convention described in our toy example, we can isolate the artist of the lyrics page.

```
html |>
  html_element("a.header-new-crumb")
> {html_node}
> <a class="header-new-crumb" itemprop="url" href="/music/Radiohead">
```

```
> [1] <span itemprop="name">Radiohead</span>
```

We can then extract the text with `html_text()`.

```
artist <-
  html |>
  html_element("a.header-new-crumb") |>
  html_text()
artist

> [1] "Radiohead"
```

Let's extract the song title in the same way.

```
song <-
  html |>
  html_element("h1.header-new-title") |>
  html_text()
song

> [1] "Karma Police"
```

Now if we inspect the HTML of the lyrics page, we will notice that the lyrics are contained in `<p></p>` tags with the class `lyrics-paragraph`.

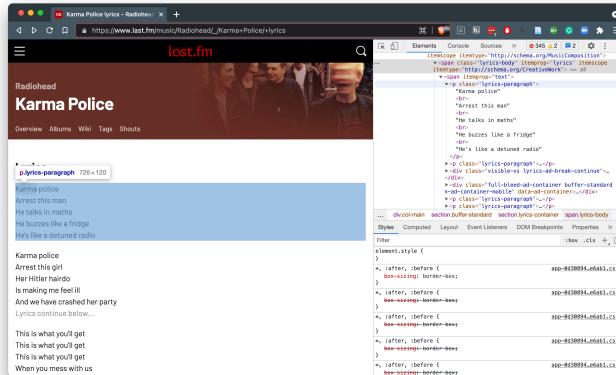


Figure 5.6: Using the “Inspect Element” command to explore raw html.

Since there are multiple elements that we want to extract, we will need to use the `html_elements()` function instead of the `html_element()` which only targets one element.

```

lyrics <-
  html |>
  html_elements("p.lyrics-paragraph") |>
  html_text()
lyrics

> [1] "Karma policeArrest this manHe talks in mathsHe buzzes like a fridgeHe's like a detu
> [2] "Karma policeArrest this girlHer Hitler hairdoIs making me feel illAnd we have crash
> [3] "This is what you'll getThis is what you'll getThis is what you'll getWhen you mess
> [4] "Karma policeI've given all I canIt's not enoughI've given all I canBut we're still
> [5] "This is what you'll getThis is what you'll getThis is what you'll getWhen you mess
> [6] "For a minute thereI lost myself, I lost myselfPhew, for a minute thereI lost myself
> [7] "For a minute thereI lost myself, I lost myselfPhew, for a minute thereI lost myself

```

At this point, we have isolated and extracted the artist, song, and lyrics from the webpage. Each of these elements are stored in character vectors in our R session. To complete our task we need to write this data to disk as plain text. With an eye towards a tidy dataset, an ideal format to store the data is in a CSV file where each column corresponds to one of the elements from our scrape and each row an observation. A CSV file is a tabular format and so before we can write the data to disk let's coerce the data that we have into tabular format. We will use the `tibble()` function here to streamline our data frame creation.<sup>28</sup> Feeding each of the vectors `artist`, `song`, and `lyrics` as arguments to `tibble()` creates the tabular format we are looking for.

```

tibble(artist, song, lyrics) |>
  glimpse()

> #> #> #> #> #> #> #>
> #> Rows: 7
> #> Columns: 3
> #> $ artist <chr> "Radiohead", "Radiohead", "Radiohead", "Radiohead", "Radiohead", "Radiohead", "Radiohead"
> #> $ song   <chr> "Karma Police", "Karma Police", "Karma Police", "Karma Police", "Karma Police", "Karma Police", "Karma Police"
> #> $ lyrics <chr> "Karma policeArrest this manHe talks in mathsHe buzzes like a f~

```

Notice that there are seven rows in this data frame, one corresponding to each paragraph in `lyrics`. R has a bias towards working with vectors of the same length. As such each of the other vectors (`artist`, and `song`) are replicated, or recycled, until they are the same length as the longest vector `lyrics`, which a length of seven.

For good documentation let's add our object `lyrics_url` to the data frame, which contains the actual web link to this page, and assign the result to `song_lyrics`.

---

<sup>28</sup>`tibble` objects are `data.frame` objects with some added extra bells and whistles that we won't get into here.

```
song_lyrics <- tibble(artist, song, lyrics, lyrics_url)
```

The final step is to write this data to disk. To do this we will use the `write_csv()` function.

```
write_csv(x = song_lyrics, path =
  ..../data/original/lyrics.csv")
```

### 5.3.3 Scaling up

At this point you may be think, ‘Great, I can download data from a single page, but what about downloading multiple pages?’ Good question. That’s really where the strength of a programming approach takes hold. Extracting information from multiple pages is not fundamentally different than working with a single page. However, it does require more sophisticated understanding of the web and R coding strategies, in particular **iteration**.

Before we get to iteration, let’s first create a couple functions to make it possible to efficiently reuse the code we have developed so far:

1. the `get_lyrics` function wraps the code for scraping a single lyrics webpage from last.fm.

```
get_lyrics <- function(lyrics_url) {
  # Function: Scrape last.fm lyrics page for: artist,
  #           song,
  # and lyrics from a provided content link.
  # Return as a tibble/data.frame

  cat("Scraping song lyrics from:", lyrics_url, "\n")

  pacman::p_load(tidyverse, rvest) # install/ load
  # package(s)

  url <- url(lyrics_url, "rb") # open url connection
  html <- read_html(url) # read and parse html as an xml
  # object
  close(url) # close url connection

  artist <-
    html |>
    html_element("a.header-new-crumb") |>
```

```
html_text()

song <-
  html |>
  html_element("h1.header-new-title") |>
  html_text()

lyrics <-
  html |>
  html_elements("p.lyrics-paragraph") |>
  html_text()

cat("...one moment ")

Sys.sleep(1) # sleep for 1 second to reduce server load

song_lyrics <- tibble(artist, song, lyrics, lyrics_url)

cat("... done! \n")

return(song_lyrics)
}
```

 Tip

The `get_lyrics` function includes all of the code developed previously, but also includes: (1) output messages (`cat()`), (2) a processing pause (`Sys.sleep()`), and (3) code to manage opening and closing web connections (`url()` and `close()`).

Points (1) and (2) will be useful when we iterate over this function to provide status messages and to reduce server load when processing multiple webpages from a web domain. (3) will be necessary to manage webpages that are non-existent. As we will see, we will generate url link to multiple song lyrics some of which will not be valid. To avoid errors that will stop the processing these steps have been incorporated here.

2. the `write_content` writes the webscraped data to our local machine, including functionality to create the necessary directory structure of the target file path we choose.

```

write_content <- function(content, target_file) {
  # Function: Write the tibble content to disk. Create the
  ↵   directory if
  # it does not already exist.

  pacman::p_load(tidyverse) # install/ load packages

  target_dir <- dirname(target_file) # identify target
  ↵   file directory structure
  dir.create(path = target_dir, recursive = TRUE,
  ↵   showWarnings = FALSE) # create directory
  write_csv(content, target_file) # write csv file to
  ↵   target location

  cat("Content written to disk!\n")
}

```

With just these two functions, we can take a lyrics URL from last.fm and scrape and write the data to disk like this.

```

lyrics_url <-
  ↵   "https://www.last.fm/music/Pixies/_/Where+Is+My+Mind%3F/+lyrics"

lyrics_url |>
  get_lyrics() |>
  write_content(target_file =
  ↵   "../data/original/lastfm/lyrics.csv")

data/original/lastfm/
lyrics.csv

```

Now we could manually search and copy URLs and run this function pipeline. This would be fine if we had just a few particular URLs that we wanted to scrape. But if we want to, say, scrape a set of lyrics grouped by genre. We would probably want a more programmatic approach. The good news is we can leverage our understanding of webscraping to scrape last.fm to harvest the information needed to create and store links to songs by genre. We can then pass these links to a pipeline, similar to the previous one, to scrape lyrics for many songs and store the results in files grouped by genre.

Last.fm provides a genres page where some of the top genres are listed and can be further explored.

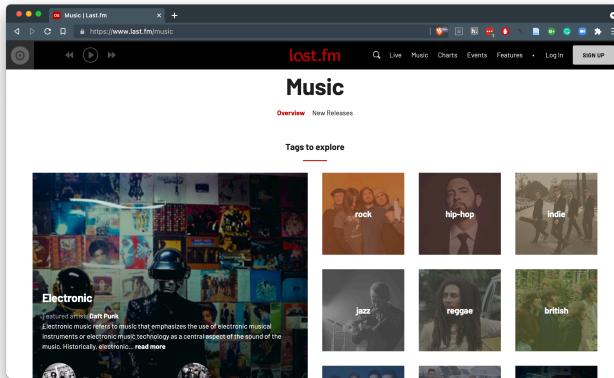


Figure 5.7: Genre page on last.fm

Diving into a particular genre, ‘rock’ for example, you will get a listing of the top tracks in that genre.

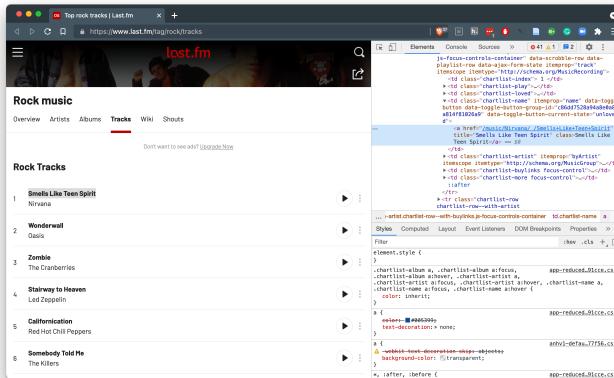


Figure 5.8: Tracks by genre list page on last.fm

If we inspect the HTML elements for the track names in Figure 5.8, we can see that a relative URL is found for the track. In this case, I have ‘Smells Like Teen Spirit’ by Nirvana highlighted in the inspector. If we follow this link to the track page and then to the lyrics for the track, you will notice that the relative URL on the track listings page has all the unique information. Only the web domain <https://www.last.fm> and the post-pended `/+lyrics` is missing.

So with this we can put together a function which gets the track listing for a

last.fm genre, scrapes the relative URLs for each of the tracks, and creates a full absolute URL to the lyrics page.

```
get_genre_lyrics_urls <- function(last_fm_genre) {
  # Function: Scrapes a given last.fm genre title for top
  # tracks in
  # that genre and then creates links to the lyrics pages
  # for these tracks

  cat("Scraping top songs from:", last_fm_genre, "genre:
       \n")

  pacman::p_load(tidyverse, rvest) # install/ load
  # packages

  # create web url for the genre listing page
  genre_listing_url <-
    paste0("https://www.last.fm/tag/", last_fm_genre,
          "/tracks")

  genre_lyrics_urls <-
    read_html(genre_listing_url) |> # read raw html and
    # parse to xml
    html_elements("td.chartlist-name a") |> # isolate the
    # track elements
    html_attr("href") |> # extract the href attribute
    paste0("https://www.last.fm", ., "/+lyrics") # join
    # the domain, relative artist path, and the
    # post-pended /+lyrics to create an absolute URL

  return(genre_lyrics_urls)
}
```

With this function, all we need is to identify the verbatim way last.fm lists the genres. For Rock, it is rock but for Hip Hop, it is hip+hop.

```
get_genre_lyrics_urls("hip+hop") |> # get urls for top
# hip hop tracks
head(n = 10) # only display 10 tracks

> Scraping top songs from: hip+hop genre:
> [1] "https://www.last.fm/music/Juzhin/_/Charlie+Conscience+(feat.+MMAIO)/+lyrics"
> [2] "https://www.last.fm/music/Juzhin/_/Railways/+lyrics"
> [3] "https://www.last.fm/music/Juzhin/_/Coming+Down/+lyrics"
```

```
> [4] "https://www.last.fm/music/Juzhin/_/Tupona/+lyrics"
> [5] "https://www.last.fm/music/Juzhin/_/Sakhalin/+lyrics"
> [6] "https://www.last.fm/music/Juzhin/_/3+Simple+Minutes/+lyrics"
> [7] "https://www.last.fm/music/Juzhin/_/Lost+Sense/+lyrics"
> [8] "https://www.last.fm/music/Juzhin/_/Wonderful/+lyrics"
> [9] "https://www.last.fm/music/Gina+Moryson/_/Vanilla+Smoothy+(Live)/+lyrics"
> [10] "https://www.last.fm/music/Juzhin/_/Flunk-Down+(Juzhin+Remix)/+lyrics"
```

So now we have a method to scrape URLs by genre and list them in a vector. Our approach, then, could be to pass these lyrics URLs to our existing pipeline which downloads the lyrics (`get_lyrics()`) and then writes them to disk (`write_content()`).

```
# Note: will not run
get_genre_lyrics_urls("hip+hop") |> # get lyrics urls for
  ↳ specific genre
    get_lyrics() |> # scrape lyrics url
      write_content(target_file =
        ↳ "../data/original/lastfm/hip_hop.csv") # write to
          ↳ disk
```

This approach, however, has a couple of problems. (1) our `get_lyrics()` function only takes one URL at a time, but the result of `get_genre_lyrics_urls()` will produce many URLs. We will be able to solve this with iteration using the `purrr`<sup>29</sup> package, specifically the `map()` function which will iteratively map each URL output from `get_genre_lyrics_urls()` to `get_lyrics()` in turn. (2) the output from our iterative application of `get_lyrics()` will produce a tibble for each URL, which then sets up a problem with writing the tibbles to disk with the `write_content()` function. To avoid this we will want to combine the tibbles into one single tibble and then send it to be written to disk. The `bind_rows()` function will do just this.

```
# Note: will run, but with occasional errors
get_genre_lyrics_urls("hip+hop") |> # get lyrics urls for
  ↳ specific genre
    map(get_lyrics) |> # scrape lyrics url
      bind_rows() |> # combine tibbles into one
        write_content(target_file =
          ↳ "../data/original/lastfm/hip_hop.csv") # write to
            ↳ disk
```

This preceding pipeline conceptually will work. However, on my test-

ing, it turns out that some of the URLs that are generated in the `get_genre_lyrics_urls()` do not exist on the site. That is, the song is listed but no lyrics have been added to the song site. This will mean that when the URL is sent to the `get_lyrics()` function, there will be an error when attempting to download and parse the page with `read_html()` which will halt the entire process. To avoid this error, we can wrap the `get_lyrics()` function in a function designed to attempt to download and parse the URL (`tryCatch()`), but if there is an error, it will skip it and move on to the next URL without stopping the processing. This approach is reflected in the `get_lyrics_catch()` function below.

```
# Wrap the `get_lyrics()` function with `tryCatch()` to
# skip URLs that have no lyrics

get_lyrics_catch <- function(lyrics_url) {
  tryCatch(get_lyrics(lyrics_url),
    error = function(e) return(NULL)) # no, URL,
    ↪   return(NULL)/ skip
}
```

Updating the pipeline with the `get_lyrics_catch()` function would look like this:

```
# Note: will run, but we can do better
get_genre_lyrics_urls("hip+hop") |> # get lyrics urls for
  ↪   specific genre
  map(get_lyrics_catch) |> # scrape lyrics url
  bind_rows() |> # combine tibbles into one
  write_content(target_file =
    ↪   "../data/original/lastfm/hip_hop.csv") # write to
    ↪   disk
```

This will work, but as we have discussed before one of this goals we have we acquiring data for a reproducible research project is to make sure that we are developing efficient code that will not burden site's server we are scraping from. In this case, we would like to check to see if the data is already downloaded. If not, then the script should run. If so, then the script does not run. Of course this is a perfect use of a conditional statement. To make this a single function we can call, I've wrapped the functions we created for getting lyric URLs from last.fm, scraping the URLs, and writing the results to disk in the `download_lastfm_lyrics()` function below. I also added a line to add a `last_fm_genre` column to the combined tibble to store the name of the genre we scraped (i.e. `mutate(genre = last_fm_genre)`).

```

download_lastfm_lyrics <- function(last_fm_genre,
← target_file) {
  # Function: get last.fm lyric urls by genre and write
  ← them to disk

  if(!file.exists(target_file)) {

    cat("Downloading data.\n")

    get_genre_lyrics_urls(last_fm_genre) |>
      map(get_lyrics_catch) |>
      bind_rows() |>
      mutate(genre = last_fm_genre) |>
      write_content(target_file)

  } else {
    cat("Data already downloaded!\n")
  }
}

```

Now we can call this function on any genre on the last.fm site and download the top 50 song lyrics for that genre (provided they all have lyrics pages).

```

# Scrape lyrics for 'pop'
download_lastfm_lyrics(last_fm_genre = "pop", target_file
←   = "../data/original/lastfm/pop.csv")

# Scrape lyrics for 'rock'
download_lastfm_lyrics(last_fm_genre = "rock", target_file
←   = "../data/original/lastfm/rock.csv")

# Scrape lyrics for 'hip hop'
download_lastfm_lyrics(last_fm_genre = "hip+hop",
←   target_file = "../data/original/lastfm/hip_hop.csv")

# Scrape lyrics for 'metal'
download_lastfm_lyrics(last_fm_genre = "metal",
←   target_file = "../data/original/lastfm/metal.csv")

```

Now we can see that our web scrape data is organized in a similar fashion to the other data we acquired in this chapter.

```

data/
  derived/
    original/
      cedel2/
        texts.csv
      gutenberg/
        works_pq.csv
        works_pr.csv
    lastfm/
      country.csv
      hip_hop.csv
      lyrics.csv
      metal.csv
      pop.csv
      rock.csv
    sbc/
      meta-data/
      transcriptions/
  scs/
    README
    discourse
    disfluency
    documentation/
    tagged
    timed-transcript
    transcript
  twitter/
    rt_latinx.csv

```

Again, it is important to add these custom functions to our `acquire_functions.R` script in the `functions/` directory so we can access them in our scripts more efficiently and make our analysis steps more succinct and legible.

In this section we covered scraping language data from the web. The `rvest` package provides a host of functions for downloading and parsing HTML. We first looked at a toy example to get a basic understanding of how HTML works and then moved to applying this knowledge to a practical example. To maintain a reproducible workflow, the code developed in this example was grouped into task-oriented functions which were in turn joined and wrapped into a function that provided convenient access to our workflow and avoided unnecessary downloads (in the case the data already exists on disk).

Here we have built on previously introduced R coding concepts and demonstrated various others. Web scraping often requires more knowledge of and

familiarity with R as well as other web technologies. Rest assured, however, practice will increase confidence in your abilities. I encourage you to practice on your own with other websites. You will encounter problems. Consult the R documentation in RStudio or online and lean on the R community on the web at sites such as Stack Overflow<sup>30</sup> *inter alia*.

## 5.4 Documentation

As part of the data acquisition process it is important include documentation that describes the data resource(s) that will serve as the base for a research project. For all resources the data should include as much information possible that outlines the sampling frame of the data (Ädel 2020). For a corpus sample acquired from a repository will often include documentation which will outline the sampling frame –this most likely will be the very information which leads a researcher to select this resource for the project at hand. It is important to include this documentation (HTML or PDF file) or reference to the documentation (article citation or link<sup>31</sup>) within the reproducible project’s directory structure.

In other cases where the data acquisition process is formulated and conducted by the researcher for the specific aims of the research (i.e. API and web scraping approaches), the researcher should make an effort to document those aspects which are key for the study, but that may also be of interest to other researchers for similar research questions. This will may include language characteristics such as modality, register, genre, etc., speaker/ writer characteristics such as demographics, time period(s), context of the linguistic communication, etc. and process characteristics such as the source of the data, the process of acquisition, date of acquisition, etc. However, it is important to recognize that each language sample and the resource from which it is drawn is unique. As a general rule of thumb, a researcher should document the resource as if this were a resource *they* were to encounter for the first time. To archive this information, it is standard practice to include a **README** file in the relevant directory where the data is stored.

```
data/
  derived/
  original/
    cedel2/
```

<sup>30</sup><https://stackoverflow.com/>

<sup>31</sup>Note that web links can change and it is often best to safeguard the documentation by downloading the HTML documentation page instead of linking

```
documentation/
texts.csv
gutenberg/
README.md
works_pq.csv
works_pr.csv
lastfm/
README.md
country.csv
hip_hop.csv
lyrics.csv
metal.csv
pop.csv
rock.csv
sbc/
meta-data/
transcriptions/
scs/
README
discourse
disfluency
documentation/
tagged
timed-transcript
transcript
twitter/
README.md
rt_latinx.csv
```

For both existing corpora and data samples acquired by the researcher it is also important to signal if there are conditions and/ or licensing restrictions that one should heed when using and potentially sharing the data. In some cases existing corpus data come with restrictions on data sharing. These can be quite restrictive and ultimately require that the corpus data not be included in publically available reproducible project or data can only be shared in a derived format. If this the case, it is important to document the steps to legally acquire the data so that a researcher can acquire their own license and take full advantage of your reproducible project.

In the case of data from APIs or web scraping, there too may be stipulations on sharing data. A growing number of data sources apply one of the available Creative Common Licenses<sup>32</sup>. Check the source of your data for more infor-

---

<sup>32</sup><https://creativecommons.org/about/cclicenses/>

mation and if you are a member of a research institution you will likely have a specialist<sup>33</sup> on Copyright and Fair Use<sup>34</sup>.

---

## Summary

In this chapter we have covered a lot of ground. On the surface we have discussed three 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 and working with those function in an iterative manner. 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 posts I will cover how to wrangle your raw data into a tidy dataset. This will include working with and incorporating meta-data as well as augmenting a dataset with linguistic annotations.

---

## Activities

### 💡 Recipe

**What:** Control statements, custom functions, and iteration<sup>a</sup>

**How:** Read Recipe 6 and participate in the Hypothes.is online social annotation.

**Why:** To increase your ability to produce effective, concise, and reproducible code. The three main areas we will cover are working with control statements, writing custom functions, and leveraging iteration. These programming strategies are often useful for acquiring data but, as we

---

<sup>33</sup><https://zsr.wfu.edu/digital-scholarship/copyright/>

<sup>34</sup><https://www.copyright.gov/fair-use/more-info.html>

will see, they are powerful concepts that can be used throughout a reproducible research project.

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_6.html](https://lin380.github.io/tadr/articles/recipe_6.html)

### 💡 Lab

**What:** Control statements, custom functions, and iteration<sup>a</sup>

**How:** Clone, fork, and complete the steps in Lab 6.

**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.

<sup>a</sup>[https://github.com/lin380/lab\\_6](https://github.com/lin380/lab_6)

---

## Questions

### 💡 Conceptual questions

1. ...
2. ...

### 💡 Technical exercises

1. ...
2. ...



# 6

## *Curate data(sets)*

The hardest bit of information to extract is the first piece.

— Robert Ferrigno

### Keys

- what are some of the formats that data can take?
- what R programming strategies are used to read these formats into tabular, tidy dataset structures?
- what is the importance of maintaining modularity between data and data processing in a reproducible research project?

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. As Acquired data can come in a wide variety of formats that depend largely on the richness of the metadata that is included, but also can reflect individual preferences. In this chapter we will consider three general types of 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 (see Chapter 4, section 4.2. The resulting dataset will be the base from which we will work to further transform the dataset such that it aligns with the analysis method(s) that we will implement. And as in previous implementation steps, we will discuss the important role of documentation.

### Swirl

**What:** Regular Expressions<sup>a</sup>

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

**Why:** To learn the basics of how to define search patterns to match strings, or characters, using Regular Expressions.

<sup>a</sup><https://github.com/lin380/swirl>

## 6.1 Unstructured

The bulk of text that is available in the wild is of the unstructured variety. Unstructured data is data that has not been organized to make the information contained within explicit. Explicit 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.

As an example of an unstructured source of text data, let's take a look at the Europarl Parallel Corpus<sup>1</sup>, as introduced in Chapter 2 “Understanding data”. This data contains parallel texts (source and translated documents) from the European Parliamentary proceedings for some 21 European languages. Here we will focus in on the translation from Spanish to English (Spanish-English).

### 6.1.1 Orientation

With the data downloaded into the `data/original/europarl/` directory we see that there are two files. One corresponding to the source language (Spanish) and one for the target language (English).

```
data/original/europarl/  
    europarl-v7.es-en.en  
    europarl-v7.es-en.es
```

Looking at the first 10 lines of the first file, we can see that this is running text.

```
> 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
```

<sup>1</sup><https://www.statmt.org/europarl/>

Table 6.1: Idealized structure for the Europarl Corpus dataset.

type	sentence_id	sentence
Source	1	...sentence from source language
Target	1	...sentence from target language

concerned, particularly those of the terrible storms, in the various countries of the European Union.

- > Please rise, then, for this minute's silence.
- > (The House rose and observed a minute's silence)
- > Madam President, on a point of order.
- > You will be aware from the press and television that there have been a number of bomb explosions and killings in Sri Lanka.
- > One of the people assassinated very recently in Sri Lanka was Mr Kumar Ponnambalam, who had visited the European Parliament just a few months ago.

The only meta information that we can surmise from these files is the fact that we know one is the source language and one is the target language and that each sentence is aligned (parallel) with the lines in the other file.

So with what we have we'd like to create a data frame that has the seen in Table 6.1.

### 6.1.2 Tidy the data

To create this dataset structure let's read the files with the `readtext()` function from `readtext` package and assign them to a meaningful variable.

```
# Read the Europarle files
europarl_en <- # English target text
  ↳ readtext::readtext("../data/original/europarl/europarl-v7.es-en.en",
  ↳ # path to the data
    verbosity = 0) # don't show warnings

europarl_es <- # Spanish source text
  ↳ readtext::readtext("../data/original/europarl/europarl-v7.es-en.es",
  ↳ # path to the data
    verbosity = 0) # don't show warnings
```

**⚠ Tip**

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.

Now there are a couple things to note about the `europarl_en` and `europarl_es` objects. If we inspect their structure, we will find that the dimensions of the data frame that is created is one row by two columns.

```
str(europarl_en) # inspect the structure of the object

#> Classes 'readtext' and 'data.frame': 1 obs. of 2 variables:
#> $ doc_id: chr "europarl-v7.es-en.en"
#> $ text : chr "Resumption of the session\nI declare resumed the
session of the European Parliament adjourned on Friday 17 Decem"|
--truncated--
```

**⚠ 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 columns are `doc_id` and `text`. `doc_id` is created by `readtext` to index each file that is read in. The `text` column is where the text appears. The fact that we only have one row means that all the text in the entire file is contained in one cell! We will want to break this cell up into rows for each sentence, but for now let's work with getting the columns to line up with our idealized dataset structure.

First let's change the type of data frame that we are working with to a tibble. This will make sure we don't accidentally print hundreds of lines to our R Markdown output and/ or the R Console. Then we will rename the `doc_id` column to `type` and change the value of that column to "Target" (for English) and "Source" (for Spanish).

```

europarl_target <-
  europarl_en |> # readtext data frame
  as_tibble() |> # convert to tibble
  rename(type = doc_id) |> # rename doc_id to type
  mutate(type = "Target") # change type value to 'Target'

europarl_source <-
  europarl_es |> # readtext data frame
  as_tibble() |> # convert to tibble
  rename(type = doc_id) |> # rename doc_id to type
  mutate(type = "Source") # change type value to 'Source'

```

We have two objects now, one corresponding to the ‘Source’ and the other the ‘Target’ parallel texts. Let’s now join these two datasets, one on top of the other –that is, by rows. We will use the `bind_rows()` function for this.

```

europarl <-
  bind_rows(europarl_target, europarl_source)

str(europarl) # inspect the structure of the object

#> #> tibble [2 x 2] (S3:tbl_df/tbl/data.frame)
#> $ type: chr [1:2] "Target" "Source"
#> $ text: chr [1:2] "Resumption of the session\nI declare resumed the session of the Eur

```

The `europarl` dataset now has 2 columns, as before, and 2 rows –each corresponding to the distinct language types (Source/ Target).

Remember our goal is to create a dataset structure with three columns `type`, `sentence_id`, and `sentence`. At the moment we have `type` and `text` –where `text` has all of the sentences in for each type in a cell. So we are going to want to break up the `text` column into sentences, group the sentences that are created by `type`, and then number these sentences so that they are aligned between the distinct types.

To break up the text into sentences we are going to turn to the `tidytext` package. This package has a extremely useful function `unnest_tokens()` which provides an effective way to break text into various units (see `?tidytext::unnest_tokens` for a full list of token types). Since I know from looking at the raw text that each sentence is on its own line, the best strategy to break the text into sentence units is to find a way to break each line into a new row in our dataset. To do this we need to use the `token = "regex"` (for Regular Expression) and use the `pattern = "\n"` which tells R to look for carriage returns to use as the breaking criterion.

```

europarl_sentences <-
  europarl |>
    tidytext::unnest_tokens(output = sentence, # new column
                           input = text, # column to find
                           ↵   text
                           token = "regex", # use a regular
                           ↵   expression to break up the
                           ↵   text
                           pattern = "\n", # break text by
                           ↵   carriage returns (returns
                           ↵   after lines)
                           to_lower = FALSE) # do not
                           ↵   lowercase the text

glimpse(europarl_sentences) # preview the structure

#> Rows: 3,926,375
#> Columns: 2
#> $ type      <chr> "Target", "Target", "Target", "Target", "Target", "Target", ...
#> $ sentence <chr> "Resumption of the session", "I declare resumed the session o~
```

 Tip

Regular Expressions are a powerful pattern matching syntax. They are used extensively in text manipulation and we will see them again and again. A good website to practice Regular Expressions is RegEx101<sup>a</sup>. You can also install the `regexplain` package in R to get access to a useful RStudio Addin<sup>b</sup>.

<sup>a</sup><https://regex101.com/>

<sup>b</sup><https://rstudio.github.io/rstudioaddins/>

Our new `europarl_sentences` object is a data frame with almost 4 million rows! The final step to get to our envisioned dataset structure is to add the `sentence_id` column which will be calculated by grouping the data by `type` and then assigning a row number to each of the sentences in each group.

```

europarl_sentences_id <-
  europarl_sentences |> # dataset
  group_by(type) |> # group by type
  mutate(sentence_id = row_number()) |> # add a row number
  ↵   for each sentence for each level of type
```

Table 6.2: First ten sentences in the Europarle Corpus curated dataset.

type	sentence_id	sentence
Source	1	Reanudación del período de sesiones
Target	1	Resumption of the session
Source	2	Declaro reanudado el período de sesiones del Parlamento Europeo, interrumpido el
Target	2	I declare resumed the session of the European Parliament adjourned on Friday 17 I
Source	3	Como todos han podido comprobar, el gran "efecto del año 2000" no se ha produci
Target	3	Although, as you will have seen, the dreaded 'millennium bug' failed to materialise,
Source	4	Sus Señorías han solicitado un debate sobre el tema para los próximos días, en el cu
Target	4	You have requested a debate on this subject in the course of the next few days, dur
Source	5	A la espera de que se produzca, de acuerdo con muchos colegas que me lo han pedi
Target	5	In the meantime, I should like to observe a minute's silence, as a number of Memb

```

  select(type, sentence_id, sentence) |> # select the
  #   relevant columns to keep
  ungroup() |> # ungroup by type
  arrange(sentence_id, type) # arrange the dataset

  europarle_sentences_id |>
    slice_head(n = 10) |>
    knitr::kable(booktabs = TRUE)

```

### 6.1.3 Write dataset

At this point we have the curated dataset (`europarle_sentences_id`) 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 (transform data) 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).

```

data/
  derived
  original

```

Since this dataset is derived by our work, we will add it to the `derived/` directory. I'll create a `europarle/` directory just to keep things organized.

```

# Write the curated dataset to disk
fs::dir_create(path = "../data/derived/europarle/") #
  ↵  create the europarle directory
write_csv(x = europarle_sentences_id, # object to write
          file =
            ↵  "../data/derived/europarle/europarle_curated.csv")
            ↵  # target file location/ name

```

This is how the directory structure under the derived/ directory looks now.

```

data/
  derived
    europarle
      europarle_curated.csv
  original
    europarle
      europarl-v7.es-en.en
      europarl-v7.es-en.es

```

#### 6.1.4 Summary

In this section we worked with unstructured data and looked at how to read the data into an R session and manipulate the data to form a tidy dataset with a few columns that we could derive based on the information we have about the corpus.

In our discussion we worked step by step to curate the Europarle Corpus, adding in intermediate steps for illustration purposes. However, in a more realistic case the code would most likely make more extensive use of piping (|>) to reduce the number of intermediate objects and make the code more legible. Below I've included a sample of what that code might look like.

```

# Read data and set up `type` column
europarle_en <-

  ↵  readtext::readtext("../data/original/europarle/europarl-v7.es-en.en",
  ↵  # path to the data
    verbosity = 0) |> # don't show
    ↵  warnings
  as_tibble() |> # convert to tibble
  rename(type = doc_id) |> # rename doc_id to type
  mutate(type = "Target") # change type value to 'Target'

```

```
europarl_es <-
  ↳  readtext::readtext("../data/original/europarl/europarl-v7.es-en.en",
  ↳  # path to the data
    ↳  verbosity = 0) |> # don't show
      ↳  warnings
  as_tibble() |> # convert to tibble
  rename(type = doc_id) |> # rename doc_id to type
  mutate(type = "Source") # change type value to 'Source'

# Join the datasets by rows
europarl <-
  bind_rows(europarl_en, europarl_es)

# Segment the `text` column into `sentence` units
europarl <-
  europarl |> # dataset
  tidytext::unnest_tokens(output = sentence, # new column
    input = text, # column to find
    ↳  text
    token = "regex", # use a regular
    ↳  expression to break up the
    ↳  text
    pattern = "\n", # break text by
    ↳  carriage returns (returns
    ↳  after lines)
    to_lower = FALSE) # do not
    ↳  lowercase the text

# Add `sentence_id` to each `type`
europarl <-
  europarl |> # dataset
  group_by(type) |> # group by type
  mutate(sentence_id = row_number()) |> # add a row number
    ↳  for each sentence for each level of type
  select(type, sentence_id, sentence) |> # select the
    ↳  relevant columns to keep
  ungroup() |> # ungroup by type
  arrange(sentence_id, type) # arrange the dataset

# Write the curated dataset to disk
```

Table 6.3: Example .csv file in R

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

```
fs::dir_create(path = ".../data/derived/europarle/") #
  ↪  create the europarle directory
write_csv(x = europarle_sentences_id, # object to write
          file =
            ↪ ".../data/derived/europarle/europarle_curated.csv")
            ↪ # target file location/ name
```

## 6.2 Structured

On the opposite side of the spectrum from unstructured data, structured data includes more metadata information –often much more. The association of metadata with the language to be analyzed means that the data has already been curated to some degree, therefore it is more apt to discuss structured data as a dataset. There are two 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 is curated in a fashion that makes sense for the current research. Let's look at each of these questions briefly and then get to a practical example.

There are file formats which are purposely designed for storing structured datasets. Some very common file types are .csv, .xml, .json, etc. The data within these files is explicitly organized. For example, in a .csv file, the dataset structure is represented by delimiting the columns and rows by commas.

```
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
```

When read into R, the .csv file format is converted to a data frame with the appropriate structure.

With an understanding of how the information is encoding into a file, 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. The curation

Table 6.4: ?(caption)

process that is reflected in a structured dataset may or may not initially align with the goals of our research either in terms of the type(s) of information or the unit of analysis of the structured dataset. The aim, then, is to take advantage of the information and curate it such that it does align.

As an example case of curating structured datasets, we will look at the song lyric datasets acquired from Last.fm in the previous chapter.

### 6.2.1 Orientation

The individual datasets from the Last.fm webscrape are found inside the `data/original/lastfm/` directory, and includes the `README.md` documentation file.

```
data/
  derived/
  original/
    lastfm/
      README.md
      country.csv
      hip_hop.csv
      lyrics.csv
      metal.csv
      pop.csv
      rock.csv
```

Let's take a look at the structure of one of genres from these set of lyrics to familiarize ourselves with the structure.

```
lf_country <- read_csv(file =
  "data/original/lastfm/country.csv") # read the csv
  file
lf_country |> # dataset
  slice_head(n = 10) |> # first 10 observations
knitr::kable(booktabs = TRUE) # print pretty table
```

We can see a couple important characteristics from this preview of the dataset. First, we see the columns include `artist`, `song`, `lyrics`, `lyrics_url`, and `genre`. Second, we see that for each son the lyrics are segmented across multiple rows.

Table 6.5: Example file from the Last.fm dataset of song lyrics.

artist	song	lyrics
Johnny Cash	Hurt	I hurt myself todayTo see if I still feelI focus on the painThe only thing that
Johnny Cash	Hurt	What have I becomeMy sweetest friendEveryone I know goes awayIn the end
Johnny Cash	Hurt	I wear this crown of thornsUpon my liar's chairFull of broken thoughtsI cann
Johnny Cash	Hurt	What have I becomeMy sweetest friendEveryone I know goes awayIn the end
Johnny Cash	Hurt	If I could start againA million miles awayI would keep myselfI would find a w
Johnny Cash	Ring of Fire	Love is a burning thingAnd it makes a fiery ringBound by wild desireI fell in
Johnny Cash	Ring of Fire	I fell into a burning ring of fireI went down, down, downAnd the flames went
Johnny Cash	Ring of Fire	I fell into a burning ring of fireI went down, down, downAnd the flames went
Johnny Cash	Ring of Fire	The taste of love is sweetWhen hearts like ours meetI fell for you like a child
Johnny Cash	Ring of Fire	Oh, but the fire went wild

 Tip

You may notice that in addition to the lyrics being separated by line, there appears to be an artifact from the original webscrape of this data which has individual lyric lines run in to the next. An example is the the lyrics "... hurt myself todayTo see if I still feelI focus...". We will address this issue when it comes time to normalize the dataset in the transform process.

Given the fact that each of these files will include a `genre` label, that means that we will be able to read in each of these files in one operation and the distinction between genres will be recoverable. The next thing to think about is how we want to curate the dataset for our purposes. That is, what should the base structure of our curated dataset look like?

Let's make the assumption that we want to have the columns `artist`, `song`, `lyrics`, and `genre`. The `lyrics_url` could be useful for documentation purposes, but for our text analysis it does not appear to be very relevant –so we will drop it. The second aspect concerns the observations. As it stands, the dataset the observations reflect the formatting of the website from which the lyrics were drawn. A potentially better organization would to have each observation correspond to all the lyrics for a single song. In this case we will want to collapse the current `lyrics` column's values into lyrics for the entire song –maintaining the other measure for each of the other columns.

With this structure in mind, we are shooting for an idealized structure such as the one below.

Table 6.6: Idealized structure for the Last.fm dataset.

artist	song	lyrics	genre
Johnny Cash	Hurt	...lyrics for the song...	country
Johnny Cash	Ring of Fire	...lyrics for the song...	country

### 6.2.2 Tidy the datasets

So our objectives are set, let's first read in all the files. To do this we will again use the `readtext()` function. But instead of reading one file at a time we will read all the files of interest (those with the `.csv` extension) in one go. The `readtext()` function allows for the use of 'wildcard' notation (\*) in the file(s) path to enable pattern matching.

So the files in the `data/original/lastfm/` directory look like this.

```
./data/original/lastfm/README.md
./data/original/lastfm/country.csv
./data/original/lastfm/hip_hop.csv
./data/original/lastfm/lyrics.csv
./data/original/lastfm/metal.csv
./data/original/lastfm/pop.csv
./data/original/lastfm/rock.csv
```

We want all the files, except the `REAME.md` file. To do this we want our path to look like this:

```
./data/original/lastfm/*.csv
```

The wildcard `*` replaces the genre names and this effectively only matches files ending in `.csv`.

Great, that will capture the files we are looking for but when working with `readtext()` we will need to set the `text_field` argument to the column that corresponds to the text in our dataset. That is the `lyrics` column. Let's go ahead and do this and convert the result to a tibble.

```
lastfm <-
  readtext(file = ".../data/original/lastfm/*.csv", # files
          to match using *.csv
          text_field = "lyrics") |> # text column from
          the datasets
  as_tibble() # convert to a tibble
```

```

glimpse(lastfm) # preview

#> Rows: 2,172
#> Columns: 6
#> $ doc_id      <chr> "country.csv.1", "country.csv.2", "country.csv.3", "country~
#> $ text         <chr> "I hurt myself todayTo see if I still feelI focus on the pa~
#> $ artist       <chr> "Johnny Cash", "Johnny Cash", "Johnny Cash", "Johnny Cash", ~
#> $ song          <chr> "Hurt", "Hurt", "Hurt", "Hurt", "Hurt", "Ring of Fire", "Ri~
#> $ lyrics_url   <chr> "https://www.last.fm/music/Johnny+Cash/_/Hurt/+lyrics", "ht~
#> $ genre         <chr> "country", "country", "country", "country", "country", "cou~

```

Looking at the preview of the data frame we now have in `lastfm` there are a couple things to note. First, we see that a column `doc_id` has been added. This column is used by `readtext()` to index the file from which the data was read. In our case since we already have sufficient information to index our dataset, we can drop this column. Next we see that the `lyrics` column has been renamed to `text`. This is because we set this as the `text_field` when we read in the files. We can easily rename this column, but we'll leave that for later.

Let's go ahead and drop the columns that we have decided will not figure in our curated dataset. We can use the `select()` function to either select those columns we want to keep or by using the `-` operator, identify the columns we want to drop. The decision of 'selecting' or 'deselecting' is usually one of personal choice and code succinctness. In this case, we are dropping two columns and keeping four, so let's deselect. I will assign the result to the same name as our current dataset, effectively overwriting that dataset.

```

lastfm <- # new dataset
  select(lastfm, # original dataset
        -doc_id, -lyrics_url) # drop these columns

glimpse(lastfm) # preview

#> Rows: 2,172
#> Columns: 4
#> $ text    <chr> "I hurt myself todayTo see if I still feelI focus on the painTh~
#> $ artist  <chr> "Johnny Cash", "Johnny Cash", "Johnny Cash", "Johnny Cash", "Jo~
#> $ song    <chr> "Hurt", "Hurt", "Hurt", "Hurt", "Hurt", "Ring of Fire", "Ring o~
#> $ genre   <chr> "country", "country", "country", "country", "country", "country~

```

Now let's work to collapse the lyrics in the `text` column by each distinct `artist`, `song`, and `genre` combination. We will use the `group_by()` function to create `artist` `genre` groupings and then use `summarize()` to create

Table 6.7: Sample lyrics from Last.fm dataset collapsed by artist, song, and genre.

artist	song	genre	lyrics
3 Doors Down	Here Without You	rock	A hundred days have made me olderSince the last time tha
3 Doors Down	Kryptonite	rock	Well I took a walk around the world to ease my troubled m
50 Cent	In Da Club	hip-hop	Go, go, go, go, goGo, shortyIt's your birthdayWe gon' p
a-ha	Take On Me	pop	Talking awayI don't know whatWhat to sayI'll say it anyway
ABBA	Dancing Queen	pop	You can dance, you can jiveHaving the time of your life, oo

a new column in which the `text` field is collapsed into all the song lyrics for this grouping. Inside the `summarize()` function we use `str_flatten()` with the argument `collapse = " "` to collapse each observation in `text` leaving a single whitespace between the observations (otherwise each line would then be joined contiguously to the next).

```

lastfm <-
  lastfm |> # dataset
  group_by(artist, song, genre) |> # grouping
  summarise(lyrics = str_flatten(text, collapse = " ")) |>
    # collapse text into the new column `lyrics`
    # (dropping `text`)
  ungroup() # unset the groupings

glimpse(lastfm) # preview

#> Rows: 199
#> Columns: 4
#> $ artist <chr> "3 Doors Down", "3 Doors Down", "50 Cent", "a-ha", "ABBA", "Aer~
#> $ song   <chr> "Here Without You", "Kryptonite", "In Da Club", "Take On Me", "~
#> $ genre  <chr> "rock", "rock", "hip-hop", "pop", "pop", "rock", "country", "co~
#> $ lyrics <chr> "A hundred days have made me olderSince the last time that I sa~
```

Let's take a look at the first 5 observations from this collapsed dataset.

```

lastfm |> # dataset
  slice_head(n = 5) |> # first 5 observations
  knitr::kable(booktabs = TRUE) # print pretty table
```

At this point, the only thing left to do to get this dataset to align with our idealized dataset structure is to organize the column ordering (using `select()`). I will also arrange the dataset alphabetically by `genre` and `artist` (using `arrange()`).

Table 6.8: Sample lyrics from curated Last.fm dataset.

artist	song	lyrics
Alan Jackson	Little Bitty	Have a little love on a little honeymoon You got a little dish an
Alan Jackson	Remember When	Remember when I was young and so were you And time stood
Brad Paisley	Mud on the Tires	I've got some big news The bank finally came through And I'm
Carrie Underwood	Before He Cheats	Right now, he's probably slow dancin' With a bleached-blond t
Dierks Bentley	What Was I Thinkin'	Becky was a beauty from south Alabama Her daddy had a hea

```

lastfm <-
  lastfm |> # original dataset
  select(artist, song, lyrics, genre) |> # order columns
  ↵  (and rename `text` to `lyrics`)
  arrange(genre, artist) # arrange rows by `genre` and
  ↵  `artist`

lastfm |> # curated dataset
  slice_head(n = 5) |> # first 5 observations
  knitr::kable(booktabs = TRUE) # print pretty table

```

### 6.2.3 Write dataset

We now have a curated dataset that we can write to disk. Again, as with the Europarl Corpus dataset we curated before, we will write this dataset to the `data/derived/` directory –effectively ensuring that it is clear that this dataset was created by our project work.

```

fs::dir_create(path = "../data/derived/lastfm/") # create
  ↵  lastfm subdirectory
write_csv(lastfm,
  file =
  ↵  "../data/derived/lastfm/lastfm_curated.csv")
  ↵  # write lastfm to disk and label as the
  ↵  curated dataset

```

And here's an overview of our new directory structure.

```

data/
  derived/
    lastfm/

```

```

lastfm_curated.csv
original/
lastfm/
README.md
country.csv
hip_hop.csv
lyrics.csv
metal.csv
pop.csv
rock.csv

```

#### 6.2.4 Summary

Again, to summarize, here is the code that will accomplish the steps we covered in this section on curating structured datasets.

```

# Read Last.fm lyrics and subset relevant columns
lastfm <-
  readtext(file = ".../data/original/lastfm/*.csv", # files
  ↵   to match using *.csv
    text_field = "lyrics") |> # text column from
    ↵   the datasets
  as_tibble() |> # convert to a tibble
  select(-doc_id, -lyrics_url) # drop these columns

# Collapse text by artist, song, and genre grouping
lastfm <-
  lastfm |> # dataset
  group_by(artist, song, genre) |> # grouping
  summarise(lyrics = str_flatten(text, collapse = " ")) |>
  ↵   # collapse text into the new column `lyrics`
  ↵   (dropping `text`)
  ungroup() # unset the groupings

# Order columns and arrange rows
lastfm <-
  lastfm |> # original dataset
  select(artist, song, lyrics, genre) |> # order columns
  ↵   (and rename `text` to `lyrics`)
  arrange(genre, artist) # arrange rows by `genre` and
  ↵   `artist`

```

```

# Write curated dataset to disk
fs::dir_create(path = "../data/derived/lastfm/") # create
  ↵ lastfm subdirectory
write_csv(lastfm,
  file =
    ↵ "../data/derived/lastfm/lastfm_curated.csv")
  ↵ # write lastfm to disk and label as the
  ↵ curated dataset

```

### 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.

As an example we will work with the The Switchboard Dialog Act Corpus (SDAC) which extends the Switchboard Corpus<sup>2</sup> with speech act annotation. **(ADD CITATION)**

 Tip

The SDAC dialogues (`swb1_dialogact_annot.tar.gz`) are available as a free download from the LDC<sup>a</sup>. To download, decompress, and organize this resource, follow the strategies discussed in “Acquire data” for Direct Downloads. The `tadr` package provides the `tadr::get_compressed_data()` function to accomplish this step.

<sup>a</sup><https://catalog.ldc.upenn.edu/docs/LDC97S62/>

#### 6.3.1 Orientation

The main directory structure of the SDAC data looks like this:

<sup>2</sup><https://catalog.ldc.upenn.edu/LDC97S62>

```

data/
  derived/
  original/
    sdac/
      README
      doc/
      sw00utt/
      sw01utt/
      sw02utt/
      sw03utt/
      sw04utt/
      sw05utt/
      sw06utt/
      sw07utt/
      sw08utt/
      sw09utt/
      sw10utt/
      sw11utt/
      sw12utt/
      sw13utt/

```

The `README` file contains basic information about the resource, the `doc/` directory contains more detailed information about the dialog annotations, and each of the following directories prefixed with `sw...` contain individual conversation files. Here's a peek at internal structure of the first couple directories.

```

README
doc
  manual.august1.html
sw00utt
  sw_0001_4325.utt
  sw_0002_4330.utt
  sw_0003_4103.utt
  sw_0004_4327.utt
  sw_0005_4646.utt

```

Let's take a look at the first conversation file (`sw_0001_4325.utt`) to see how it is structured.

```

>
*****  

>  

*****  

> *x* **  


```

```
> **x* Copyright (C) 1995 University of Pennsylvania **x*
> **x* **x*
> **x* The data in this file are part of a preliminary version of the
**x*
> **x* Penn Treebank Corpus and should not be redistributed. Any **x*
> **x* research using this corpus or based on it should acknowledge **x*
> **x* that fact, as well as the preliminary nature of the corpus. **x*
> **x* **x*
>
*****  

>
*****  

>
>
> FILENAME: 4325_1632_1519
> TOPIC#: 323
> DATE: 920323
> TRANSCRIBER: glp
> UTT_CODER: tc
> DIFFICULTY: 1
> TOPICALITY: 3
> NATURALNESS: 2
> ECHO_FROM_B: 1
> ECHO_FROM_A: 4
> STATIC_ON_A: 1
> STATIC_ON_B: 1
> BACKGROUND_A: 1
> BACKGROUND_B: 2
> REMARKS: None.
>
>
=====
>
>
> 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? /
```

There are few things to take note of here. First we see that the conversation files have a meta-data header offset from the conversation text by a line of

= characters. Second the header contains meta-information of various types. Third, the text is interleaved with an annotation scheme.

Some of the information may be readily understandable, such as the various pieces of meta-data in the header, but to get a better understanding of what information is encoded here let's take a look at the `README` file. In this file we get a birds eye view of what is going on. In short, the data includes 1155 telephone conversations between two people annotated with 42 'DAMSL' dialog act labels. The `README` file refers us to the `doc/manual.august1.html` file for more information on this scheme.

At this point we open the the `doc/manual.august1.html` file in a browser and do some investigation. We find out that 'DAMSL' stands for 'Discourse Annotation and Markup System of Labeling' and that the first characters of each line of the conversation text correspond to one or a combination of labels for each utterance. So for our first utterances we have:

```

o = "Other"
qw = "Wh-Question"
qy^d = "Declarative Yes-No-Question"
+ = "Segment (multi-utterance)"

```

Each utterance is also labeled for speaker ('A' or 'B'), speaker turn ('1', '2', '3', etc.), and each utterance within that turn ('utt1', 'utt2', etc.). There is other annotation provided withing each utterance, but this should be enough to get us started on the conversations.

Now let's turn to the meta-data in the header. We see here that there is information about the creation of the file: 'FILENAME', 'TOPIC', 'DATE', etc. The `doc/manual.august1.html` file doesn't have much to say about this information so I returned to the LDC Documentation<sup>3</sup> and found more information in the Online Documentation<sup>4</sup> section. After some poking around in this documentation I discovered that that meta-data for each speaker in the corpus is found in the `caller_tab.csv` file. This tabular file does not contain column names, but the `caller_doc.txt` does. After inspecting these files manually and comparing them with the information in the conversation file I noticed that the 'FILENAME' information contained three pieces of useful information delimited by underscores \_.

```
*****
```

---

<sup>3</sup><https://catalog.ldc.upenn.edu/docs/LDC97S62/>

<sup>4</sup><https://catalog.ldc.upenn.edu/docs/LDC97S62/>

Table 6.9: Idealized structure for the SDAC dataset.

doc_id	damsl_tag	speaker	turn_num	utterance_num	utterance_text	speaker_id
4325	o	A	1	1	Okay. /	1632
4325	qw	A	1	2	{D So, }	1632
4325	qy^d	B	2	1	[ [ I guess, +	1519

```

FILENAME: 4325_1632_1519
TOPIC#: 323
DATE: 920323
TRANSCRIBER: glp

```

The first information is the document id (4325), the second and third correspond to the speaker number: the first being speaker A (1632) and the second speaker B (1519).

In sum, we have 1155 conversation files. Each file has two parts, a header and text section, separated by a line of = characters. The header section contains a ‘FILENAME’ line which has the document id, and ids for speaker A and speaker B. The text section is annotated with DAMSL tags beginning each line, followed by speaker, turn number, utterance number, and the utterance text. With this knowledge in hand, let’s set out to create a tidy dataset with the following column structure:

### 6.3.2 Tidy the data

Let’s begin by reading one of the conversation files into R as a character vector using the `read_lines()` function from the `readr` package.

```

doc <-
  read_lines(file =
    " ../data/original/sdac/sw00utt/sw_0001_4325.utt") #
    ↪   read a single file as character vector

```

To isolate the vector element that contains the document and speaker ids, we use `str_detect()` from the `stringr` package. This function takes two arguments, a string and a pattern, and returns a logical value, TRUE if the pattern is matched or FALSE if not. We can use the output of this function, then, to subset the `doc` character vector and only return the vector element (line) that contains `digits_digits_digits` with a regular expression. The expression combines the digit matching operator `\d` with the `+` operator to match 1 or more contiguous digits. We then separate three groups of `\d+` with underscores `_`. The result is `\d+_\\d+_\\d+`.

```
doc[str_detect(doc, pattern = "\\\d+_\\\\d+_\\\\d+")] # isolate
  ↵ pattern
#> [1] "FILENAME:\\t4325_1632_1519"
```

 Tip

The `stringr` package has a handy function `str_view()` and `str_view_all()` which allow for interactive pattern matching. There is also an RStudio Addin with the `regeexplain` package which also can be very helpful for developing regular expression syntax.

The next step is to extract the three digit sequences that correspond to the `doc_id`, `speaker_a_id`, and `speaker_b_id`. First we extract the pattern that we have identified with `str_extract()` and then we can break up the single character vector into multiple parts based on the underscore `_`. The `str_split()` function takes a string and then a pattern to use to split a character vector. It will return a list of character vectors.

```
doc[str_detect(doc, "\\\d+_\\\\d+_\\\\d+")] |> # isolate
  ↵ pattern
  str_extract(pattern = "\\\d+_\\\\d+_\\\\d+") |> # extract the
    ↵ pattern
  str_split(pattern = "_") # split the character vector

#> [[1]]
#> [1] "4325" "1632" "1519"
```

A **list** is a special object type in R. It is an unordered collection of objects whose lengths can differ (contrast this with a data frame which is a collection of objects whose lengths are the same –hence the tabular format). In this case we have a list of length 1, whose sole element is a character vector of length 3 –one element per segment returned from our split. This is a desired result in most cases as if we were to pass multiple character vectors to our `str_split()` function we don't want the results to be conflated as a single character vector blurring the distinction between the individual character vectors. If we *would* like to conflate, or *flatten* a list, we can use the `unlist()` function.

```
doc[str_detect(doc, "\\\d+_\\\\d+_\\\\d+")] |> # isolate
  ↵ pattern
  str_extract(pattern = "\\\d+_\\\\d+_\\\\d+") |> # extract the
    ↵ pattern
  unlist(str_split(pattern = "_"))
```

```

str_split(pattern = "_") |> # split the character vector
unlist() # flatten the list to a character vector

#> [1] "4325" "1632" "1519"

```

Let's flatten the list in this case, as we have a single character vector, and assign this result to `doc_speaker_info`.

```

doc_speaker_info <-
  doc[str_detect(doc, "\\d+_\\d+_\\d+")] |> # isolate
  ↵ pattern
  str_extract(pattern = "\\d+_\\d+_\\d+") |> # extract the
  ↵ pattern
  str_split(pattern = "_") |> # split the character
  ↵ vector
  unlist() # flatten the list to a character vector

```

`doc_speaker_info` is now a character vector of length three. Let's subset each of the elements and assign them to meaningful variable names so we can conveniently use them later on in the tidying process.

```

doc_id <- doc_speaker_info[1] # extract by index
speaker_a_id <- doc_speaker_info[2] # extract by index
speaker_b_id <- doc_speaker_info[3] # extract by index

```

The next step is to isolate the text section extracting it from rest of the document. As noted previously, a sequence of `=` separates the header section from the text section. What we need to do is to index the point in our character vector `doc` where that line occurs and then subset the `doc` from that point until the end of the character vector. Let's first find the point where the `=` sequence occurs. We will again use the `str_detect()` function to find the pattern we are looking for (a contiguous sequence of `=`), but then we will pass the logical result to the `which()` function which will return the element index number of this match.

```

doc |>
  str_detect(pattern = "=+") |> # match 1 or more `=`
  which() # find vector index

#> [1] 31

```

So for this file 31 is the index in `doc` where the `=` sequence occurs. Now it is important to keep in mind that we are working with a single file from the `sdac/` data. We need to be cautious to not create a pattern that may be

matched multiple times in another document in the corpus. As the `=+` pattern will match `=`, or `==`, or `====`, etc. it is not implausible to believe that there might be a `=` character on some other line in one of the other files. Let's update our regular expression to avoid this potential scenario by only matching sequences of three or more `=`. In this case we will make use of the curly bracket operators `{}`.

```
doc |>
  str_detect(pattern = "={3,}") |> # match 3 or more `=`
  which() # find vector index

#> [1] 31
```

We will get the same result for this file, but will safeguard ourselves a bit as it is unlikely we will find multiple matches for `====`, `=====`, etc.

31 is the index for the `=` sequence, but we want the next line to be where we start reading the text section. To do this we increment the index by 1.

```
text_start_index <-
  doc |>
  str_detect(pattern = "={3,}") |> # match 3 or more `=`
  which() # find vector index
text_start_index <- text_start_index + 1 # increment index
  ↵ by 1
```

The index for the end of the text is simply the length of the `doc` vector. We can use the `length()` function to get this index.

```
text_end_index <- length(doc)
```

We now have the bookends, so to speak, for our text section. To extract the text we subset the `doc` vector by these indices.

```
text <- doc[text_start_index:text_end_index] # extract
  ↵ text
head(text) # preview first lines of `text`

#> [1] ""
#> [2] ""
#> [3] "o          A.1 utt1: Okay. /"
#> [4] "qw         A.1 utt2: {D So, }"
#> [5] ""
#> [6] "qy^d       B.2 utt1: [ [ I guess, +"
```

The text has some extra whitespace on some lines and there are blank lines as well. We should do some cleaning up before moving forward to organize the data. To get rid of the whitespace we use the `str_trim()` function which by default will remove leading and trailing whitespace from each line.

```
text <- str_trim(text) # remove leading and trailing
                        ↵ whitespace
head(text) # preview first lines of `text`  
  

#> [1] ""
#> [2] ""
#> [3] "o          A.1 utt1: Okay. /"
#> [4] "qw         A.1 utt2: {D So, }"
#> [5] ""
#> [6] "qy^d       B.2 utt1: [ [ I guess, +"
```

To remove blank lines we will use the a logical expression to subset the `text` vector. `text != ""` means return TRUE where lines are not blank, and FALSE where they are.

```
text <- text[text != ""] # remove blank lines
head(text) # preview first lines of `text`  
  

#> [1] "o          A.1 utt1: Okay. /"
#> [2] "qw         A.1 utt2: {D So, }"
#> [3] "qy^d       B.2 utt1: [ [ I guess, +"
#> [4] "+"        A.3 utt1: What kind of experience [ do you, + do you ] have, then with
#> [5] "+"        B.4 utt1: I think, ] + {F uh, } I wonder ] if that worked. /"
#> [6] "qy         A.5 utt1: Does it say something? /"
```

Our first step towards a tidy dataset is to now combine the `doc_id` and each element of `text` in a data frame.

```
data <- data.frame(doc_id, text) # tidy format `doc_id`  

                                ↵ and `text`
slice_head(data, n = 5) |> # preview first lines of `text`
knitr::kable(booktabs = TRUE)
```

With our data now in a data frame, its time to parse the `text` column and extract the damsl tags, speaker, speaker turn, utterance number, and the utterance text itself into separate columns. To do this we will make extensive use of regular expressions. Our aim is to find a consistent pattern that distinguishes each piece of information from other other text in a given row of `data$text` and extract it.

Table 6.10: First 5 observations of prelim data curation of the SDAC data.

doc_id	text
4325	o A.1 utt1: Okay. /
4325	qw A.1 utt2: {D So, }
4325	qy^d B.2 utt1: [ [ I guess, +
4325	+ A.3 utt1: What kind of experience [ do you, + do you ] have, then with child care? /
4325	+ B.4 utt1: I think, ] + {F uh, } I wonder ] if that worked. /

The best way to learn regular expressions is to use them. To this end I've included a link to the interactive regular expression practice website regex101<sup>5</sup>.

Open this site and copy the text below into the 'TEST STRING' field.

```

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? /
sd         B.6 utt1: I think it usually does. /
ad         B.6 utt2: You might try, {F uh, } /
h          B.6 utt3: I don't know, /
ad         B.6 utt4: hold it down a little longer, /

```

Now manually type the following regular expressions into the 'REGULAR EXPRESSION' field one-by-one (each is on a separate line). Notice what is matched as you type and when you've finished typing. You can find out exactly what the component parts of each expression are doing by toggling the top right icon in the window or hovering your mouse over the relevant parts of the expression.

```

^.+?\s
[AB]\.\d+
utt\d+
:.+$
```

As you can now see, we have regular expressions that will match the damsl tags, speaker and speaker turn, utterance number, and the utterance text. To

<sup>5</sup><https://regex101.com>

Figure 6.1: RegEx101

apply these expressions to our data and extract this information into separate columns we will make use of the `mutate()` and `str_extract()` functions. `mutate()` will take our data frame and create new columns with values we match and extract from each row in the data frame with `str_extract()`. Notice that `str_extract()` is different than `str_extract_all()`. When we work with `mutate()` each row will be evaluated in turn, therefore we only need to make one match per row in `data$text`.

I've chained each of these steps in the code below, dropping the original `text` column with `select(-text)`, and overwriting `data` with the results.

```
# Extract column information from `text`
data <-
  data |> # current dataset
  mutate(damsl_tag = str_extract(string = text, pattern =
    "^.+?\s")) |> # extract damsl tags
```

```

    mutate(speaker_turn = str_extract(string = text, pattern
    ↵   = "[AB]\\.\\d+")) |> # extract speaker_turn pairs
    mutate(utterance_num = str_extract(string = text,
    ↵   pattern = "utt\\d+")) |> # extract utterance number
    mutate(utterance_text = str_extract(string = text,
    ↵   pattern = ":+$")) |> # extract utterance text
    select(-text) # drop the `text` column

glimpse(data) # preview the data set

#> Rows: 159
#> Columns: 5
#> $ doc_id      <chr> "4325", "4325", "4325", "4325", "4325", "4325", ~
#> $ damsl_tag   <chr> "o ", "qw ", "qy^d ", "+ ", "+ ", "qy ", "sd ", "ad ", ~
#> $ speaker_turn <chr> "A.1", "A.1", "B.2", "A.3", "B.4", "A.5", "B.6", "B.6", ~
#> $ utterance_num <chr> "utt1", "utt2", "utt1", "utt1", "utt1", "utt1", "utt1", ~
#> $ utterance_text <chr> ": Okay. /", ": {D So, }", ": [ [ I guess, +", ": What~

```

 Tip

One twist you will notice is that regular expressions in R require double backslashes (\\\\") where other programming environments use a single backslash (\\").

There are a couple things left to do to the columns we extracted from the text before we move on to finishing up our tidy dataset. First, we need to separate the `speaker_turn` column into `speaker` and `turn_num` columns and second we need to remove unwanted characters from the `damsl_tag`, `utterance_num`, and `utterance_text` columns.

To separate the values of a column into two columns we use the `separate()` function. It takes a column to separate and character vector of the names of the new columns to create. By default the values of the input column will be separated by non-alphanumeric characters. In our case this means the . will be our separator.

```

data <-
  data |> # current dataset
  separate(col = speaker_turn, # source column
           into = c("speaker", "turn_num")) # separate
           ↵   speaker_turn into distinct columns: speaker
           ↵   and turn_num

```

```

glimpse(data) # preview the data set

#> Rows: 159
#> Columns: 6
#> $ doc_id      <chr> "4325", "4325", "4325", "4325", "4325", "4325", "4325", ~
#> $ damsl_tag   <chr> "o ", "qw ", "qy^d ", "+", "+", "qy ", "sd ", "ad ", ~
#> $ speaker      <chr> "A", "A", "B", "A", "B", "A", "B", "B", "B", "B", "B", ~
#> $ turn_num     <chr> "1", "1", "2", "3", "4", "5", "6", "6", "6", "6", "6", ~
#> $ utterance_num <chr> "utt1", "utt2", "utt1", "utt1", "utt1", "utt1", "utt1", "utt1", ~
#> $ utterance_text <chr> ": Okay. /", "{D So, }", "[ [ I guess, +", "What~

```

To remove unwanted leading or trailing whitespace we apply the `str_trim()` function. For removing other characters we matching the character(s) and replace them with an empty string ("") with the `str_replace()` function. Again, I've chained these functions together and overwritten `data` with the results.

```

# Clean up column information
data <-
  data |> # current dataset
  mutate(damsl_tag = str_trim(damsl_tag)) |> # remove
  ↵ leading/ trailing whitespace
  mutate(utterance_num = str_replace(string =
  ↵ utterance_num, pattern = "utt", replacement = ""))
  ↵ |> # remove 'utt'
  mutate(utterance_text = str_replace(string =
  ↵ utterance_text, pattern = ":\\s", replacement = ""))
  ↵ |> # remove ':'
  mutate(utterance_text = str_trim(utterance_text)) # trim
  ↵ leading/ trailing whitespace

glimpse(data) # preview the data set

#> Rows: 159
#> Columns: 6
#> $ doc_id      <chr> "4325", "4325", "4325", "4325", "4325", "4325", "4325", ~
#> $ damsl_tag   <chr> "o ", "qw ", "qy^d ", "+", "+", "qy ", "sd ", "ad ", "h", "ad~
#> $ speaker      <chr> "A", "A", "B", "A", "B", "A", "B", "B", "B", "B", "B", ~
#> $ turn_num     <chr> "1", "1", "2", "3", "4", "5", "6", "6", "6", "6", "6", ~
#> $ utterance_num <chr> "1", "2", "1", "1", "1", "1", "1", "2", "3", "4", "5", ~
#> $ utterance_text <chr> "Okay. /", "{D So, }", "[ [ I guess, +", "What kind of~

```

To round out our tidy dataset for this single conversation file we will connect the `speaker_a_id` and `speaker_b_id` with speaker A and B in our cur-

rent dataset adding a new column `speaker_id`. The `case_when()` function does exactly this: allows us to map rows of `speaker` with the value “A” to `speaker_a_id` and rows with value “B” to `speaker_b_id`.

```
# Link speaker with speaker_id
data <-
  data |> # current dataset
  mutate(speaker_id = case_when( # create speaker_id
    speaker == "A" ~ speaker_a_id, # speaker_a_id value
    ~ when A
    speaker == "B" ~ speaker_b_id # speaker_b_id value
    ~ when B
  ))
  glimpse(data) # preview the data set

#> Rows: 159
#> Columns: 7
#> $ doc_id      <chr> "4325", "4325", "4325", "4325", "4325", "4325", ~
#> $ damsl_tag   <chr> "o", "qw", "qy^d", "+", "+", "qy", "sd", "ad", "h", "ad~
#> $ speaker     <chr> "A", "A", "B", "A", "B", "A", "B", "B", "B", "B", ~
#> $ turn_num    <chr> "1", "1", "2", "3", "4", "5", "6", "6", "6", "6", ~
#> $ utterance_num <chr> "1", "2", "1", "1", "1", "1", "1", "2", "3", "4", "5", ~
#> $ utterance_text <chr> "Okay. /", "{D So, }", "[ [ I guess, +", "What kind of~
#> $ speaker_id   <chr> "1632", "1632", "1519", "1632", "1519", "1632", "1519", ~
```

We now have the tidy dataset we set out to create. But this dataset only includes one conversation file! We want to apply this code to all 1155 conversation files in the `sdac/` corpus. The approach will be to create a custom function which groups the code we’ve done for this single file and then iterative send each file from the corpus through this function and combine the results into one data frame.

Here’s the custom function with some extra code to print a progress message for each file when it runs.

```
extract_sdac_metadata <- function(file) {
  # Function: to read a Switchboard Corpus Dialogue file
  ~ and extract meta-data
  cat("Reading", basename(file), "...")

  # Read `file` by lines
  doc <- read_lines(file)
```

```

# Extract `doc_id`, `speaker_a_id`, and `speaker_b_id`
doc_speaker_info <-
  doc[str_detect(doc, "\d+\d+\d+")] |> # isolate
  ↵ pattern
  str_extract("\d+\d+\d+") |> # extract the pattern
  str_split(pattern = "_") |> # split the character
  ↵ vector
  unlist() # flatten the list to a character vector
  doc_id <- doc_speaker_info[1] # extract `doc_id`
  speaker_a_id <- doc_speaker_info[2] # extract
  ↵ `speaker_a_id`
  speaker_b_id <- doc_speaker_info[3] # extract
  ↵ `speaker_b_id`


# Extract `text`
text_start_index <- # find where header info stops
  doc |>
  str_detect(pattern = "{3,}") |> # match 3 or more `=`
  which() # find vector index

  text_start_index <- text_start_index + 1 # increment
  ↵ index by 1
  text_end_index <- length(doc) # get the end of the text
  ↵ section

  text <- doc[text_start_index:text_end_index] # extract
  ↵ text
  text <- str_trim(text) # remove leading and trailing
  ↵ whitespace
  text <- text[text != ""] # remove blank lines

  data <- data.frame(doc_id, text) # tidy format `doc_id`
  ↵ and `text`


# Extract column information from `text`
data <-
  data |>
  mutate(damsl_tag = str_extract(string = text, pattern
    ↵ = ".+?\s")) |> # extract damsl tags
  mutate(speaker_turn = str_extract(string = text,
    ↵ pattern = "[AB]\.\d+")) |> # extract
    ↵ speaker_turn pairs

```

```

    mutate(utterance_num = str_extract(string = text,
    ↵   pattern = "utt\\d+")) |> # extract utterance
    ↵   number
    mutate(utterance_text = str_extract(string = text,
    ↵   pattern = ".+$")) |> # extract utterance text
    select(-text)

    # Separate speaker_turn into distinct columns
    data <-
        data |> # current dataset
        separate(col = speaker_turn, # source column
            into = c("speaker", "turn_num")) # separate
            ↵   speaker_turn into distinct columns:
            ↵   speaker and turn_num

    # Clean up column information
    data <-
        data |>
        mutate(damsl_tag = str_trim(damsl_tag)) |> # remove
        ↵   leading/ trailing whitespace
        mutate(utterance_num = str_replace(string =
        ↵   utterance_num, pattern = "utt", replacement = ""))
        ↵   |> # remove 'utt'
        mutate(utterance_text = str_replace(string =
        ↵   utterance_text, pattern = ":\\s", replacement =
        ↵   "")) |> # remove ':'
        mutate(utterance_text = str_trim(utterance_text)) #
        ↵   trim leading/ trailing whitespace

    # Link speaker with speaker_id
    data <-
        data |> # current dataset
        mutate(speaker_id = case_when( # create speaker_id
            speaker == "A" ~ speaker_a_id, # speaker_a_id value
            ↵   when A
            speaker == "B" ~ speaker_b_id # speaker_b_id value
            ↵   when B
            ))
        cat(" done.\n")
        return(data) # return the data frame object
    }

```

As a sanity check we will run the `extract_sdac_metadata()` function on a the conversation file we were just working on to make sure it works as expected.

```

extract_sdac_metadata(file =
  ↵ ".../data/original/sdac/sw00utt/sw_0001_4325.utt") |>
  glimpse()

#> Reading sw_0001_4325.utt ... done.
#> Rows: 159
#> Columns: 7
#> $ doc_id      <chr> "4325", "4325", "4325", "4325", "4325", "4325", ~
#> $ damsl_tag   <chr> "o", "qw", "qy^d", "+", "+", "qy", "sd", "ad", "h", "ad~
#> $ speaker     <chr> "A", "A", "B", "A", "B", "A", "B", "B", "B", "B", ~
#> $ turn_num    <chr> "1", "1", "2", "3", "4", "5", "6", "6", "6", "6", ~
#> $ utterance_num <chr> "1", "2", "1", "1", "1", "1", "1", "2", "3", "4", "5", ~
#> $ utterance_text <chr> "Okay. /", "{D So, }", "[ [ I guess, +", "What kind of~
#> $ speaker_id   <chr> "1632", "1632", "1519", "1632", "1519", "1632", "1519", ~

```

Looks good!

So now it's time to create a vector with the paths to all of the conversation files. `fs::dir_ls()` interfaces with our OS file system and will return the paths to the files in the specified directory. We also add a pattern to match conversation files (`regexp = "\\utt$"`) so we don't accidentally include other files in the corpus. `recurse` set to `TRUE` means we will get the full path to each file.

```

sdac_files <-
  fs::dir_ls(path = ".../data/original/sdac/", # source
  ↵ directory
    recurse = TRUE, # traverse all
    ↵ sub-directories
    type = "file", # only return files
    regexp = "\\utt$") # only return files
    ↵ ending in .utt
  head(sdac_files) # preview file paths

...

```

```

.../data/original/sdac/sw00utt/sw_0001_4325.utt
.../data/original/sdac/sw00utt/sw_0002_4330.utt
.../data/original/sdac/sw00utt/sw_0003_4103.utt
.../data/original/sdac/sw00utt/sw_0004_4327.utt
.../data/original/sdac/sw00utt/sw_0005_4646.utt
.../data/original/sdac/sw00utt/sw_0006_4108.utt

```

o pass each conversation file in the vector of paths to our conversation files iteratively to the `extract_sdac_metadata()` function we use `map()`. This will apply the function to each conversation file and return a data frame for each.

`bind_rows()` will then join the resulting data frames by rows to give us a single tidy dataset for all 1155 conversations. Note there is a lot of processing going on here we have to be patient.

```
# Read files and return a tidy dataset
sdac <-
  sdac_files |> # pass file names
  map(extract_sdac_metadata) |> # read and tidy
    ↵ iteratively
  bind_rows() # bind the results into a single data frame
```

We now see that we have 223606 observations (individual utterances in this dataset).

```
glimpse(sdac) # preview complete curated dataset

#> Rows: 223,606
#> Columns: 7
#> $ doc_id      <chr> "4325", "4325", "4325", "4325", "4325", "4325", ~
#> $ damsl_tag   <chr> "o", "qw", "qy^d", "+", "+", "qy", "sd", "ad", "h", "ad~
#> $ speaker     <chr> "A", "A", "B", "A", "B", "A", "B", "B", "B", "B", "B", ~
#> $ turn_num    <chr> "1", "1", "2", "3", "4", "5", "6", "6", "6", "6", "6", ~
#> $ utterance_num <chr> "1", "2", "1", "1", "1", "1", "1", "2", "3", "4", "5", ~
#> $ utterance_text <chr> "Okay. /", "{D So, }", "[ [ I guess, +", "What kind of~
#> $ speaker_id   <chr> "1632", "1632", "1519", "1632", "1519", "1632", "1519", ~
```

### 6.3.3 Write datasets

Again as in the previous cases, we will write this dataset to disk to prepare for the next step in our text analysis project.

```
fs::dir_create(path = "../data/derived/sdac/") # create
  ↵ sdac subdirectory
write_csv(sdac,
  file = "../data/derived/sdac/sdac_curated.csv")
  ↵ # write sdac to disk and label as the
    ↵ curated dataset
```

The directory structure now looks like this:

```
data/
  derived/
```

```

sdac/
    sdac_curated.csv
original/
    sdac/
        README
        doc/
        sw00utt/
        sw01utt/
        sw02utt/
        sw03utt/
        sw04utt/
        sw05utt/
        sw06utt/
        sw07utt/
        sw08utt/
        sw09utt/
        sw10utt/
        sw11utt/
        sw12utt/
        sw13utt/

```

### 6.3.4 Summary

In this section we looked at semi-structured data. This type of data often requires the most work to organize into a tidy dataset. We continued to work with many of the R programming strategies introduced to this point in the coursebook. We also made more extensive use of regular expressions to pick out information from a semi-structured document format.

To round out this section I've provided a code summary of the steps involved to conduct the curation of the Switchboard Dialogue Act Corpus files. Note that I've added the `extract_sdac_metadata()` custom function to a file called `curate_functions.R` and sourced this file. This will make the code more succinct and legible here, as well in your own research projects.

```

# Source the `extract_sdac_metadata()` function
source("../functions/curate_functions.R")

# Get list of the corpus files (.utt)
sdac_files <-
  fs::dir_ls(path = "../data/original/sdac/", # source
             directory

```

```

    recurse = TRUE, # traverse all
    ↵   sub-directories
    type = "file", # only return files
    regexp = "\\*.utt$") # only return files
    ↵   ending in .utt

# Read files and return a tidy dataset
sdac <-
  sdac_files |> # pass file names
  map(extract_sdac_metadata) |> # read and tidy
  ↵   iteratively
  bind_rows() # bind the results into a single data frame

# Write curated dataset to disk
fs::dir_create(path = "../data/derived/sdac/") # create
  ↵   sdac subdirectory
write_csv(sdac,
  file = "../data/derived/sdac/sdac_curated.csv")
  ↵   # write sdac to disk and label as the
  ↵   curated dataset

```

---

## 6.4 Documentation

At this stage we again want to ensure that the data that we have derived is well-documented. Where in the data acquisition process the documentation was focused on the sampling frame, curated datasets require documentation that describes the structure of the now rectangular dataset and its attributes. This documentation is known as a **data dictionary**. At the curation stage this documentation often contains the following information (“How to Make a Data Dictionary” 2021):

- names of the variables (as they appear in the dataset)
- human-readable names for the variables
- short prose descriptions of the variables, including units of measurement (where applicable)

A data dictionary will take the format of a table and can be stored in a tabular-oriented file format (such as .csv). It is often easier to work with a spreadsheet to create this documentation. I suggest creating a .csv file with the basic structure of the documentation. You can do this however you choose, but I suggest using something along these lines as seen in the following custom function, `data_dic_starter()`.

Table 6.11: Data dictionary starter structure for the SDAC curated dataset.

variable_name	name	description
doc_id		
damsl_tag		
speaker		
turn_num		
utterance_num		
utterance_text		
speaker_id		

```
data_dic_starter <- function(data, file_path) {
  # Function:
  # Creates a .csv file with the basic information
  # to document a curated dataset

  tibble(variable_name = names(data), # column with
  ↳ existing variable names
  name = "", # column for human-readable names
  description = "") |> # column for prose description
  write_csv(file = file_path) # write to disk
}
```

Running this function in the R Console on the curated dataset (in this case the `sdac` dataset), will provide this structure.

The resulting .csv file can then be opened with spreadsheet software (such as MS Excel, Google Sheets, etc.) and edited.<sup>6</sup>

Save this file as a .csv file and replace the original starter file. Note that it is important to use a plain-text file format for the official documentation file and avoid proprietary formats to ensure open accessibility and future compatibility.<sup>7</sup>

Our `data/derived/` directory now looks like this.

```
data/
  derived/
```

<sup>6</sup>Note on RStudio Cloud you will need to download the .csv file and, after editing, upload the complete data dictionary file. Make sure to save the edited file as a .csv file.

<sup>7</sup>Although based on spreadsheets, Bromman and Woo (2018) outlines many of the best for good data organization regardless of the technology.

A	B	C	D	E	F	G
1	variable_name	name	description			
2	doc_id	Document ID	ID for each conversation document			
3	damsl_tag	DAMSL tag	DAMSL dialogue act annotation labels			
4	speaker	Speaker label	Label for each speaker in the conversation			
5	turn_num	Number of turns	Number of contiguous utterance turns for a given speaker			
6	utterance_num	Number of utterances	The cumulative number of utterances in the conversation			
7	utterance_text	Utterance text	The actual dialogue utterance. Includes disfluency annotation.			
8	speaker_id	Speaker ID	ID for each speaker			
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

```
sdac/
sdac_curated.csv
data_dictionary_sdac.csv
```

## 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 meta-information included. The results from our data curation resulted in a curated dataset that is saved separate from the original data to maintain modularity between what the data(set) looked like before

we intervene and afterwards. In addition to the code we use to derived the curated dataset's structure, we also include a data dictionary which documents the names of the variables and provides sufficient description of these variables so that it is clear what our dataset contains.

It is important to recognized that this curated dataset will form the base for the next step in our text analysis project and the last step in data preparation for analysis: dataset transformation. This last step in preparing data for analysis is to convert this curated dataset into a dataset that is directly aligned with the research aims (i.e. analysis method(s)) of the project. 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.

---

## Activities

### 💡 Recipe

**What:** Regular Expressions and reshaping datasets<sup>a</sup>

**How:** Read Recipe 7 and participate in the Hypothes.is online social annotation.

**Why:** To 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.

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_7.html](https://lin380.github.io/tadr/articles/recipe_7.html)

### 💡 Lab

**What:** Regular Expressions and reshaping datasets<sup>a</sup>

**How:** Clone, fork, and complete the steps in Lab 7.

**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.

<sup>a</sup>[https://github.com/lin380/lab\\_7](https://github.com/lin380/lab_7)

## Questions

### Conceptual questions

1. ...
2. ...

### Technical exercises

1. ...
2. ...



# 7

## Transform datasets

Nothing is lost. Everything is transformed.

— Michael Ende, The Neverending Story

### Keys

- What is the role of data transformation in a text analysis project?
- What are the general processes for preparing datasets for analysis?
- How do each of these general processes transform datasets?

In this chapter we turn our attention to the process of moving a curated dataset one step closer to analysis. Where in the process of curating data into a dataset the goal was to derive a tidy dataset that contained the main relational characteristics of the data for our text analysis project, the transformation step refines and potentially expands these characteristics such that they are more in line with our analysis aims. In this chapter I have grouped various transformation steps into four categories: normalization, recoding, generation, and merging. It is of note that these categories have been ordered and are covered separately for descriptive reasons. In practice the ordering of which transformation to apply before another is highly idiosyncratic and requires that the researcher evaluate the characteristics of the dataset and the desired results.

Furthermore, since in any given project there may be more than one analysis that may be performed on the data, there may be distinct transformation steps which correspond to each analysis approach. Therefore it is possible that there are more than one transformed dataset created from the curated dataset. This is one of the reasons that we create a curated dataset instead of deriving a transformed dataset from the original data. The curated dataset serves as a point of departure from which multiple transformational methods can derive distinct formats for distinct analyses.

Let's now turn to demonstrations of some common transformational steps using datasets with which we are now familiar.

### 💡 Swirl

**What:** Data manipulation<sup>a</sup>

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

**Why:** To learn more about data frame objects in R and how to manipulate them using the `dplyr` package.

<sup>a</sup><https://github.com/lin380/swirl>

## 7.1 Normalize

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 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.

### Europarle Corpus

Consider the curated Europarle Corpus dataset. I will read in the dataset. Since the dataset is quite large, I have also subsetted the dataset keeping only the first 1,000 observations for each of value of `type` for demonstration purposes.

```

europarle <- read_csv(file =
  "data/derived/europarle/europarle_curated.csv") |>
  # read curated dataset
  filter(sentence_id < 1001) # keep first 1000
  # observations for each type

glimpse(europarle)

#> Rows: 2,000
#> Columns: 3
#> $ type      <chr> "Source", "Target", "Source", "Target", "Source", "Target"~
#> $ sentence_id <dbl> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, ~
#> $ sentence   <chr> "Reanudación del período de sesiones", "Resumption of the ~

```

Simply looking at the first 14 lines of this dataset, we can see that if our goal

Table 7.1: Europarl Corpus curated dataset preview.

type	sentence_id	sentence
Source	1	Reanudación del período de sesiones
Target	1	Resumption of the session
Source	2	Declaro reanudado el período de sesiones del Parlamento Europeo, interrumpido el
Target	2	I declare resumed the session of the European Parliament adjourned on Friday 17 I
Source	3	Como todos han podido comprobar, el gran "efecto del año 2000" no se ha produci
Target	3	Although, as you will have seen, the dreaded 'millennium bug' failed to materialise,
Source	4	Sus Señorías han solicitado un debate sobre el tema para los próximos días, en el cu
Target	4	You have requested a debate on this subject in the course of the next few days, dur
Source	5	A la espera de que se produzca, de acuerdo con muchos colegas que me lo han pedi
Target	5	In the meantime, I should like to observe a minute's silence, as a number of Memb
Source	6	Invito a todos a que nos pongamos de pie para guardar un minuto de silencio.
Target	6	Please rise, then, for this minute's silence.
Source	7	(El Parlamento, de pie, guarda un minuto de silencio)
Target	7	(The House rose and observed a minute's silence)

is to work with the transcribed ('Source') and translated ('Target') language, there are lines which do not appear to be of interest.

`sentence_id` 1 appears to be title and `sentence_id` 7 reflects description of the parliamentary session. Both of these are artifacts that we would like to remove from the dataset.

To remove these lines we can turn to the programming strategies we've previously worked with. Namely we will use `filter()` to filter observations in combination with `str_detect()` to detect matches for some pattern that is indicative of these lines that we want to remove and not of the other lines that we want to keep.

Before we remove any lines, let's try craft a search pattern to identify these lines, and exclude the lines we will want to keep. Condition one is lines which start with an opening parenthesis (. Condition two is lines that do not end in standard sentence punctuation (., !, or ?). I've added both conditions to one `filter()` using the logical *OR* operator (`|`) to ensure that either condition is matched in the output.

```
# Identify non-speech lines
europarl |>
  filter(str_detect(sentence, "^\\(") |
    str_detect(sentence, "[^.!?]$")) |> # filter lines
    # that detect a match for either condition 1 or 2
```

Table 7.2: Non-speech lines in the Europarle dataset.

type	sentence_id	sentence
Source	110	(El Parlamento rechaza la propuesta por 164 votos a favor, 166 votos en contra y 7 abstenciones)
Target	93	(Parliament rejected the request) President.
Source	85	(Aplausos del grupo PSE)
Source	674	A5-0087/1999 del Sr. Jonckheer, en nombre de la Comisión de Asuntos Económicos
Source	134	Consejeros de seguridad para el transporte de mercancías peligrosas
Source	221	Transporte de mercancías peligrosas por carretera
Target	109	(Parliament rejected the request, with 164 votes for, 166 votes against and 7 abstentions)
Target	638	(The sitting was closed at 8.25 p.m.)
Target	675	A5-0087/1999 by Mr Jonckheer, on behalf of the Committee on Economic and Monetary Affairs
Target	293	Structural Funds - Cohesion Fund coordination

```

slice_sample(n = 10) |> # random sample of 10
  ↵ observations
knitr::kable(booktabs = TRUE)

```

Since this search appears to match lines that we do not want to preserve, let's move now to eliminate these lines from the dataset. To do this we will use the same regular expression patterns, but now each condition will have it's own `filter()` call and the `str_detect()` will be negated with a prefixed `!`.

```

europarl <-
  europarl |> # dataset
  filter(!str_detect(sentence, pattern = "^\\")) |> #
    ↵ remove lines starting with (
  filter(!str_detect(sentence, pattern = "[^.!?]$")) #
    ↵ remove lines not ending in ., !, or ?

```

Let's look at the first 14 lines again, now that we have eliminated these artifacts.

One further issue that we may want to resolve concerns the fact that there are whitespaces between possessive forms (i.e. “minute' s silence”). In this case we can employ `str_replace_all()` inside the `mutate()` function to overwrite the `sentence` values that match an apostrophe ' with whitespace (`\s`) before s.

```

europarl <-
  europarl |> # dataset

```

Table 7.3: Europarl Corpus non-speech lines removed.

type	sentence_id	sentence
Source	2	Declaro reanudado el período de sesiones del Parlamento Europeo, interrumpido el
Target	2	I declare resumed the session of the European Parliament adjourned on Friday 17 I
Source	3	Como todos han podido comprobar, el gran "efecto del año 2000" no se ha produci
Target	3	Although, as you will have seen, the dreaded 'millennium bug' failed to materialise,
Source	4	Sus Señorías han solicitado un debate sobre el tema para los próximos días, en el cu
Target	4	You have requested a debate on this subject in the course of the next few days, dur
Source	5	A la espera de que se produzca, de acuerdo con muchos colegas que me lo han pedi
Target	5	In the meantime, I should like to observe a minute's silence, as a number of Memb
Source	6	Invito a todos a que nos pongamos de pie para guardar un minuto de silencio.
Target	6	Please rise, then, for this minute's silence.
Source	8	Señora Presidenta, una cuestión de procedimiento.
Target	8	Madam President, on a point of order.
Source	9	Sabrá usted por la prensa y la televisión que se han producido una serie de explosio
Target	9	You will be aware from the press and television that there have been a number of b

```

mutate(sentence = str_replace_all(string = sentence,
                                  pattern = "'\\ss",
                                  replacement = "s")) #
  ↵   replace 's with
  ↵   `s

```

Now we have normalized text in the `sentence` column in the Europarl dataset.

### Last FM Lyrics

Let's look at another dataset we have worked with during this course-book: the Lastfm lyrics. Reading in the `lastfm_curated` dataset from the `data/derived/` directory we can see the structure for the curated structure.

```

lastfm <- read_csv(file =
  ↵   "../data/derived/lastfm/lastfm_curated.csv") # read in
  ↵   lastfm_curated dataset

```

There are a few things that we might want to clean out of the `lyrics` column's values. First, there are lines from the original webscrape where the end of one stanza runs into the next without whitespace between them (i.e. "honeymoonYou"). These reflect contiguous end-new line segments where stanzas

Table 7.4: Last fm lyrics dataset preview with one artist/ song per genre and the `lyrics` text truncated at 200 characters for display purposes.

artist	song	lyrics
Alan Jackson	Little Bitty	Have a little love on a little honeymoonYou got a little dish and you
50 Cent	In Da Club	Go, go, go, go, goGo, shortyIt's your birthdayWe gon' party like
Black Sabbath	Paranoid	Finished with my woman'Cause she couldn't help me with my mindI
a-ha	Take On Me	Talking awayI don't know whatWhat to sayI'll say it anywayToday i
3 Doors Down	Here Without You	A hundred days have made me olderSince the last time that I saw yo

were joined in the curation process. Second, we see that there are what appear to be backing vocals which appear between parentheses (i.e. “(Take On Me)”).

In both cases we will use `mutate()`. With contiguous end-new line segments we will use `str_replace_all()` inside and for backing vocals in parentheses we will use `str_remove_all()`.

The pattern to match for end-new lines from the stanzas will use some regular expression magic. The base pattern includes finding a pair of lowercase-uppercase letters (i.e. “nY”, in “honeymoonYou”). For this we can use the pattern `[a-z] [A-Z]`. To replace this pattern using the lowercase letter then a space and then the uppercase letter we take advantage of the grouping syntax in regular expressions (...). So we add parentheses around the two groups to capture like this `([a-z])([A-Z])`. In the replacement argument of the `str_replace_all()` function we then specify to use the captured groups in the order they appear `\1` for the lowercase letter match and `\2` for the uppercase letter match.

Now, I've looked more extensively at the `lyrics` column and found that there are other combinations that are joined between stanzas. Namely that ', !, ,, ), ?, and I also may precede the uppercase letter. To make sure we capture these possibilities as well I've updated the regular expression to `([a-z'!,.])?([A-Z])`.

Now to remove the backing vocals, the regex pattern is `\((.+?)\)` –match the parentheses and everything within the parentheses. The added ? after the + operator is what is known as a ‘lazy’ operator. This specifies that the .+ will match the minimal string that is enclosed by the trailing ). If we did not include this then we would get matches that span from the first parenthesis ( all the way to the last, which would match real lyrics, not just the backing vocals.

Putting this to work let's clean the `lyrics` column.

Table 7.5: Last fm lyrics with cleaned lyrics...

artist	song	lyrics
Alan Jackson	Little Bitty	Have a little love on a little honeymoon You got a little dish and you
50 Cent	In Da Club	Go, go, go, go, go Go, shorty It's your birthday We gon' party like
Black Sabbath	Paranoid	Finished with my woman' Cause she couldn't help me with my mind
a-ha	Take On Me	Talking away I don't know what What to say I'll say it anyway Today
3 Doors Down	Here Without You	A hundred days have made me older Since the last time that I saw you

```

lastfm <-
  lastfm |> # dataset
  mutate(lyrics =
    str_replace_all(string = lyrics,
      pattern =
        "([a-z'!,.]?[I])([A-Z])", #
        ↵   find contiguous end/ new
        ↵   line segments
      replacement = "\\\1 \\\2")) |> #
        ↵   replace with whitespace
        ↵   between
  mutate(lyrics = str_remove_all(lyrics, "\\(.+?\\)")) #
    ↵   remove backing vocals (Take On Me)
  
```

Now given the fact that songs are poems, there are many lines that are not complete sentences so there is no practical way to try to segment these into grammatical sentence units. So in this case, this seems like a good stopping point for normalizing the lastfm dataset.

## 7.2 Recode

Normalizing text can be seen as an extension of dataset curation to some extent in that the structure of the dataset is maintained. In both the Europarl and Lastfm cases 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.

**Switchboard Dialogue Act Corpus**

Table 7.6: 20 randomly sampled lines of the SDAC curated dataset.

doc_id	damsl_tag	speaker	turn_num	utterance_num	utterance_text
3697	sd	B	74	2	[ I'm, + I'm ] planning on it. /
3813	qh	A	29	3	all I can think of is if you don't keep [ a rea
3214	sv	B	48	2	{C and } you never read it again -
2967	b	B	46	1	Yeah. /
4733	sv	A	73	1	{D You know, } I just feel they do better th
3052	sd( $\hat{q}$ )	A	44	2	{C and } the wife said "these are not his sla
2537	sv	B	62	6	[ [ that, + the, ] + the ] absolute refusal to
3121	%	B	116	2	{C and, } -/
2324	sd	B	22	5	{C and } it'll convert every <noise> measur
3750	sd	B	35	2	{C and, } {F uh, } in the afternoon, {D you
4723	b	B	54	1	Right. /
2602	aa	B	2	1	Okay, /
2441	+	A	141	1	- {D you know, } /
2372	b	B	106	1	Huh-uh. /
3080	bh	B	100	1	{F Oh, } is that right. /
3777	sv	A	101	3	{D well, } a couple of the, {F uh, } ones Dis
2064	b	A	69	1	Yeah. /
2515	b	B	48	1	Okay. /
4032	b	A	55	1	Uh-huh. /
2691	+	A	99	1	# and # had their house on the market dow

The Switchboard Dialogue Act Corpus dataset that was curated in the previous chapter contains a number of variables describing conversations between speakers of American English.

Let's read in this dataset and take a closer look.

```
sdac <- read_csv(file =
  ↪ ".../data/derived/sdac/sdac_curated.csv") # read
  ↪ curated dataset
```

Among a number of metadata variables, curated dataset includes the `utterance_text` column which contains dialogue from the conversations interleaved with a disfluency annotation scheme<sup>1</sup>.

Let's drop a few variables from our dataset to rein in our focus. I will keep the `doc_id`, `speaker_id`, and `utterance_text`.

---

<sup>1</sup><https://staff.fnwi.uva.nl/r.fernandezrovira/teaching/DM-materials/DFL-book.pdf>

Table 7.7: First 10 lines of the simplified SDAC curated dataset.

doc_id	speaker_id	utterance_text
4325	1632	Okay. /
4325	1632	{D So, }
4325	1519	[ [ I guess, +
4325	1632	What kind of experience [ do you, + do you ] have, then with child care? /
4325	1519	I think, ] + {F uh, } I wonder ] if that worked. /
4325	1632	Does it say something? /
4325	1519	I think it usually does. /
4325	1519	You might try, {F uh, } /
4325	1519	I don't know, /
4325	1519	hold it down a little longer, /

```
sdac_simplified <-
  sdac |> # dataset
  select(doc_id, speaker_id, utterance_text) # columns to
  ↵ retain
```

In this disfluency annotation system, there are various conventions used for non-sentence elements. If say, for example, a researcher were to be interested in understanding the use of filled pauses ('uh' or 'uh'), the aim would be to identify those lines where the {F ...} annotation is used around the utterances 'uh' and 'um'.

To do this we turn to the `str_count()` function. This function will count the number of matches found for a pattern. We can use a regular expression to identify the pattern of interest which is all the instances of {F followed by either `uh` or `um`. Since the disfluencies may start an utterance, and therefore be capitalized we need to formulate a regular expression which allows for either `U` or `u` for each disfluency type. The result from each disfluency match will be added to a new column. To create a new column we will wrap each `str_count()` with `mutate()` and give the new column a meaningful name. In this case I've opted for `uh` and `um`.

```
sdac_disfluencies <-
  sdac_simplified |> # dataset
  mutate(uh = str_count(utterance_text, "\\\{F [Uu]h")) |>
  ↵ # match {F Uh or {F uh}
  mutate(um = str_count(utterance_text, "\\\{F [Uu]m")) #
  ↵ match {F Um or {F um}}
```

Table 7.8: First 20 lines of SDAC dataset with counts for the disfluencies ‘uh’ and ‘um’.

doc_id	speaker_id	utterance_text	uh	um
4325	1632	Okay. /	0	0
4325	1632	{D So, }	0	0
4325	1519	[ [ I guess, +	0	0
4325	1632	What kind of experience [ do you, + do you ] have, then with child care? /	0	0
4325	1519	I think, ] + {F uh, } I wonder ] if that worked. /	1	0
4325	1632	Does it say something? /	0	0
4325	1519	I think it usually does. /	0	0
4325	1519	You might try, {F uh, } /	1	0
4325	1519	I don't know, /	0	0
4325	1519	hold it down a little longer, /	0	0
4325	1519	{C and } see if it, {F uh, } -/	1	0
4325	1632	Okay <beep>. /	0	0
4325	1632	«long pause» {D Well, }	0	0
4325	1519	Okay /	0	0
4325	1519	[ I, +	0	0
4325	1632	Does it usually make a recording or s-, /	0	0
4325	1519	{D Well, } I ] don't remember. /	0	0
4325	1519	It seemed like it did, /	0	0
4325	1519	{C but } <laughter> it might not. /	0	0
4325	1519	[ I guess + -	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.

```
sdac_disfluencies <-
  sdac_disfluencies |> # dataset
  pivot_longer(cols = c("uh", "um"), # columns to convert
               names_to = "filler", # column for the
               # column names (i.e. filler types)
               values_to = "count") # column for the
               # column values (i.e. counts)
```

### Last fm

In the previous example, we used a matching approach to extract information embedded in one column of the dataset and recoded the dataset to maintain the fidelity between the particular unit of observation and the other metadata.

Another common approach for recoding datasets in text analysis projects involves recoding linguistic units as smaller units; a process known as tokenization.

Let’s return to the `lastfm` object we normalized earlier in the chapter to see the various ways one can choose to tokenize linguistic information.

In the current `lastfm` dataset, the unit of observation is the lyrics for the entire artist, song, and genre combination. If, however, we would like to change the unit to say words, we would like each word used to appear on its own row, while still maintaining the other relevant attributes associated with each word.

The `tidytext` package includes a very useful function `unnest_tokens()` which allows us to tokenize some textual input into smaller linguistic units. The ‘unnest’ part of the the function name refers to the process of extracting the unit of interest while maintaining the other relevant attributes. Let’s see this in action.

Table 7.9: First 20 lines of SDAC dataset with tidy format for **fillers** as the unit of observation.

doc_id	speaker_id	utterance_text	filler
4325	1632	Okay. /	uh
4325	1632	Okay. /	um
4325	1632	{D So, }	uh
4325	1632	{D So, }	um
4325	1519	[ [ I guess, +	uh
4325	1519	[ [ I guess, +	um
4325	1632	What kind of experience [ do you, + do you ] have, then with child care? /	uh
4325	1632	What kind of experience [ do you, + do you ] have, then with child care? /	um
4325	1519	I think, ] + {F uh, } I wonder ] if that worked. /	uh
4325	1519	I think, ] + {F uh, } I wonder ] if that worked. /	um
4325	1632	Does it say something? /	uh
4325	1632	Does it say something? /	um
4325	1519	I think it usually does. /	uh
4325	1519	I think it usually does. /	um
4325	1519	You might try, {F uh, } /	uh
4325	1519	You might try, {F uh, } /	um
4325	1519	I don't know, /	uh
4325	1519	I don't know, /	um
4325	1519	hold it down a little longer, /	uh
4325	1519	hold it down a little longer, /	um

Table 7.10: Last fm dataset with normalized lyrics.

artist	song	lyrics
Alan Jackson	Little Bitty	Have a little love on a little honeymoon You got a little dish and you
50 Cent	In Da Club	Go, go, go, go, go Go, shorty It's your birthday We gon' party like
Black Sabbath	Paranoid	Finished with my woman' Cause she couldn't help me with my mind
a-ha	Take On Me	Talking away I don't know what What to say I'll say it anyway Today
3 Doors Down	Here Without You	A hundred days have made me older Since the last time that I saw you

Table 7.11: First 10 observations for lastfm dataset tokenized by words.

artist	song	genre	word
Alan Jackson	Little Bitty	country	have
Alan Jackson	Little Bitty	country	a
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	love
Alan Jackson	Little Bitty	country	on
Alan Jackson	Little Bitty	country	a
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	honeymoon
Alan Jackson	Little Bitty	country	you
Alan Jackson	Little Bitty	country	got

```
lastfm |> # dataset
  unnest_tokens(output = word, # column for tokenized
    ↪ output
      input = lyrics, # input column
      token = "words") |> # tokenize unit type
  slice_head(n = 10) |> # preview first 10 lines
  kable(booktabs = TRUE)
```

We can see from the output, each word appears on a separate line in the order of appearance in the input text (`lyrics`). Furthermore, the output is in tidy format as each of the words is still associated with the relevant attribute values (`artist`, `song`, and `genre`). By default the tokenized text output is lowercased and the original text input column is dropped. These can be overridden, however, if desired.

In addition to ‘words’, the `unnest_tokens()` function provides easy access to a number of common tokenized units including ‘characters’, ‘sentences’, and ‘paragraphs’.

```
lastfm |> # dataset
  unnest_tokens(output = character, # column for tokenized
    ↪ output
      input = lyrics, # input column
      token = "characters") |> # tokenize unit
        ↪ type
  slice_head(n = 10) |> # preview first 10 lines
  kable(booktabs = TRUE)
```

Table 7.12: First 10 observations for lastfm dataset tokenized by characters.

artist	song	genre	character
Alan Jackson	Little Bitty	country	h
Alan Jackson	Little Bitty	country	a
Alan Jackson	Little Bitty	country	v
Alan Jackson	Little Bitty	country	e
Alan Jackson	Little Bitty	country	a
Alan Jackson	Little Bitty	country	l
Alan Jackson	Little Bitty	country	i
Alan Jackson	Little Bitty	country	t
Alan Jackson	Little Bitty	country	t
Alan Jackson	Little Bitty	country	l

The other two built-in options ‘sentences’ and ‘paragraphs’ depend on punctuation and/ or line breaks to function, so in this particular dataset, these options will not work given the particular characteristics of the `lyrics` variable.

There are even other options which allow for the creation of sequences of linguistic units. Say we want to tokenize our lyrics into two-word sequences, we can specify the `token` as ‘ngrams’ and then add the argument `n = 2` to reflect we want two-word sequences.

```
lastfm |>
  unnest_tokens(output = bigram, # column for tokenized
    ↵   output
      input = lyrics, # input column
      token = "ngrams", # tokenize unit type
      n = 2) |> # size of word sequences
  slice_head(n = 10) |> # preview first 10 lines
  kable(booktabs = TRUE)
```

The ‘n’ in ‘ngram’ refers to the number of word-sequence units we want to tokenize. Two-word sequences are known as ‘bigrams’, three-word sequences ‘trigrams’, and so on.

Table 7.13: First 10 observations for lastfm dataset tokenized by bigrams

artist	song	genre	bigram
Alan Jackson	Little Bitty	country	have a
Alan Jackson	Little Bitty	country	a little
Alan Jackson	Little Bitty	country	little love
Alan Jackson	Little Bitty	country	love on
Alan Jackson	Little Bitty	country	on a
Alan Jackson	Little Bitty	country	a little
Alan Jackson	Little Bitty	country	little honeymoon
Alan Jackson	Little Bitty	country	honeymoon you
Alan Jackson	Little Bitty	country	you got
Alan Jackson	Little Bitty	country	got a

---

### 7.3 Generate

In the process of recoding a dataset the transformation of the dataset works with information that is already explicit. The process of generation, however, aims to make implicit information explicit. 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.

Table 7.14: First 10 lines in English from the normalized SDAC dataset.

type	sentence_id	sentence
Target	2	I declare resumed the session of the European Parliament adjourned on Friday 17 L
Target	3	Although, as you will have seen, the dreaded 'millennium bug' failed to materialise,
Target	4	You have requested a debate on this subject in the course of the next few days, dur
Target	5	In the meantime, I should like to observe a minute's silence, as a number of Membe
Target	6	Please rise, then, for this minute's silence.
Target	8	Madam President, on a point of order.
Target	9	You will be aware from the press and television that there have been a number of b
Target	10	One of the people assassinated very recently in Sri Lanka was Mr Kumar Ponnamb
Target	11	Would it be appropriate for you, Madam President, to write a letter to the Sri Lan
Target	12	Yes, Mr Evans, I feel an initiative of the type you have just suggested would be ent

Let's take a look at annotation some of the language from the Europarle dataset we normalized.

```
europarl |>
  filter(type == "Target") |>
  slice_head(n = 10) |>
  kable(booktabs = TRUE)
```

We will use the cleanNLP package to do our linguistic annotation. The annotation process depends on the pre-trained language models. There is a list of the models available to access<sup>2</sup>. The `load_model_udpipe()` custom function below downloads the specified language model and initialized the `udpipe` engine (`cnlp_init_udpipe()`) for conducting annotations.

```
load_model_udpipe <- function(model_lang) {
  # Function
  # Download and load the specified udpipe language model

  cnlp_init_udpipe(model_lang) # to download the model, if
  # not downloaded
  base_path <- system.file("extdata", package = "cleanNLP")
  # get the base path
  model_name <- # extract the model_name
  base_path |> # extract the base path
  dir() |> # get the directory
```

<sup>2</sup><https://github.com/bnosac/udpipe#pre-trained-models>

```

stringr::str_subset(pattern = paste0("^", model_lang))
↪ # extract the name of the model

model_path <- udpipe::udpipe_load_model(file =
↪ file.path(base_path, model_name, fsep = "/")) # create
↪ the path to the downloaded model stored on disk
return(model_path)
}

```

In a test case, let's load the 'english' model to annotate a sentence line from the Europarle dataset to illustrate the basic workflow.

```

eng_model <- load_model_udpipe("english") # load and
↪ initialize the language model, 'english' in this case.

eng_annotation <-
europarle |> # dataset
filter(type == "Target" & sentence_id == 6) |> # select
↪ English and sentence_id 6
cnlp_annotate(text_name = "sentence", # input text
↪ (sentence)
doc_name = "sentence_id") # specify the
↪ grouping column (sentence_id)

glimpse(eng_annotation) # preview structure

#> List of 2
#> $ token    : tibble [11 x 11] (S3: tbl_df/tbl/data.frame)
#>   ..$ doc_id      : num [1:11] 6 6 6 6 6 6 6 6 6 6 ...
#>   ..$ sid        : int [1:11] 1 1 1 1 1 1 1 1 1 1 ...
#>   ..$ tid        : chr [1:11] "1" "2" "3" "4" ...
#>   ..$ token       : chr [1:11] "Please" "rise" "," "then" ...
#>   ..$ token_with_ws: chr [1:11] "Please " "rise" ", " "then" ...
#>   ..$ lemma       : chr [1:11] "please" "rise" "," "then" ...
#>   ..$ upos        : chr [1:11] "INTJ" "VERB" "PUNCT" "ADV" ...
#>   ..$ xpos        : chr [1:11] "UH" "VB" "," "RB" ...
#>   ..$ feats       : chr [1:11] NA "Mood=Imp|VerbForm=Fin" NA "PronType=Dem" ...
#>   ..$ tid_source   : chr [1:11] "2" "0" "2" "10" ...
#>   ..$ relation     : chr [1:11] "discourse" "root" "punct" "advmod" ...
#> $ document: tibble [1 x 2] (S3: tbl_df/tbl/data.frame)
#>   ..$ type   : chr "Target"
#>   ..$ doc_id: num 6
#> - attr(*, "class")= chr [1:2] "cnlp_annotation" "list"

```

Table 7.15: Annotation information for a single English sentence from the Europarle dataset.

doc_id	sid	tid	token	token_with_ws	lemma	upos	xpos	feats
6	1	1	Please	Please	please	INTJ	UH	NA
6	1	2	rise	rise	rise	VERB	VB	Mood=Imp VerbForm=Fin
6	1	3	,	,	,	PUNCT	,	NA
6	1	4	then	then	then	ADV	RB	PronType=Dem
6	1	5	,	,	,	PUNCT	,	NA
6	1	6	for	for	for	ADP	IN	NA
6	1	7	this	this	this	DET	DT	Number=Sing PronType=Dem
6	1	8	minute	minute	minute	NOUN	NN	Number=Sing
6	1	9	's	's	's	PART	POS	NA
6	1	10	silence	silence	silence	NOUN	NN	Number=Sing
6	1	11	.	.	.	PUNCT	.	NA

We see that the structure returned by the `cnlp_annotate()` function is a list. This list contains two data frames (tibbles). One for the tokens (and their annotation information) and the document (the metadata information). We can inspect the annotation characteristics for this one sentence by targeting the `$tokens` data frame. Let's take a look at the linguistic annotation information returned.

There is quite a bit of information which is returned from `cnlp_annotate()`. First note that the input sentence has been tokenized by word. Each token includes the token, lemma, part of speech (`upos` and `xpos`), morphological features (`feats`), and syntactic relationships (`tid_source` and `relation`). It is also key to note that the `doc_id`, `sid` and `tid` maintain the relational attributes from the original dataset – and therefore maintains our annotated dataset in tidy format.

Let's now annotate the same sentence from the Europarle corpus for the Source ('Spanish') and note the similarities and differences.

```
spa_model <- load_model_udpipe("spanish") # load and
  ↵ initialize the language model, 'spanish' in this case.

spa_annotation <-
  europarle |> # dataset
  filter(type == "Source" & sentence_id == 6) |> # select
    ↵ Spanish and sentence_id 6
```

Table 7.16: Annotation information for a single Spanish sentence from the Europarl dataset.

doc_id	sid	tid	token	token_with_ws	lemma	upos	xpos	feats
6	1	1	Invito	Invito	Invito	VERB	NA	Gender=Masc Number=Sing
6	1	2	a	a	a	ADP	NA	NA
6	1	3	todos	todos	todo	PRON	NA	Gender=Masc Number=Plur
6	1	4	a	a	a	ADP	NA	NA
6	1	5	que	que	que	SCONJ	NA	NA
6	1	6	nos	nos	yo	PRON	NA	Case=Acc,Dat Number=Plur
6	1	7	pongamos	pongamos	pongar	VERB	NA	Mood=Ind Number=Plur Pe
6	1	8	de	de	de	ADP	NA	NA
6	1	9	pie	pie	pie	NOUN	NA	Gender=Masc Number=Sing
6	1	10	para	para	para	ADP	NA	NA
6	1	11	guardar	guardar	guardar	VERB	NA	VerbForm=Inf
6	1	12	un	un	uno	DET	NA	Definite=Ind Gender=Masc N
6	1	13	minuto	minuto	minuto	NOUN	NA	Gender=Masc Number=Sing
6	1	14	de	de	de	ADP	NA	NA
6	1	15	silencio	silencio	silencio	NOUN	NA	Gender=Masc Number=Sing
6	1	16	.	.	.	PUNCT	NA	NA

```
cnlp_annotate(text_name = "sentence", # input text
              ↵ (sentence)
              doc_name = "sentence_id") # specify the
              ↵ grouping column (sentence_id)
```

For the Spanish version of this sentence, we see the same variables. However, the **feats** variable has morphological information which is specific to Spanish—notably gender and mood.

### ⚠ Tip

The rsyntax package (Welbers and van Atteveldt 2022) can be used to recode and extract patterns from the output from automatic linguistic annotations using cleanNLP. See the documentation for more information<sup>a</sup>.

<sup>a</sup><https://github.com/vanatteveldt/rsyntax>

Table 7.17: First 10 observations for `lastfm_words` dataset.

artist	song	genre	word
Alan Jackson	Little Bitty	country	have
Alan Jackson	Little Bitty	country	a
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	love
Alan Jackson	Little Bitty	country	on
Alan Jackson	Little Bitty	country	a
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	honeymoon
Alan Jackson	Little Bitty	country	you
Alan Jackson	Little Bitty	country	got

## 7.4 Merge

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. To merge datasets it is required that the datasets share one or more attributes. With a common attribute two datasets can be joined to coordinate the attributes of one dataset with the other effectively adding attributes and one dataset with extended information. Another approach is to join datasets with the goal of filtering one of the datasets given the matching attribute.

Let's see this in practice. Take the `lastfm` dataset. Let's tokenize the dataset into words, using `unnest_tokens()` such that our unit of observation is words.

```
lastfm_words <-
  lastfm |> # dataset
  unnest_tokens(output = "word", # output column
                input = "lyrics", # input column
                token = "words") # tokenized unit (words)

lastfm_words |> # dataset
  slice_head(n = 10) |> # first 10 observations
  kable(booktabs = TRUE)
```

Consider the `get_sentiments()` function which returns words which have been classified as ‘positive’- or ‘negative’-biased, if the lexicon is set to ‘bing’ (Hu and Liu 2004).

```

sentiments_bing <-
  tidytext::get_sentiments(lexicon = "bing") # get 'bing'
  ↵ lexicon from get_sentiments

sentiments_bing |>
  slice_head(n = 10) # preview first 10 observations

#> # A tibble: 10 x 2
#>   word      sentiment
#>   <chr>     <chr>
#> 1 2-faces    negative
#> 2 abnormal    negative
#> 3 abolish     negative
#> 4 abominable  negative
#> 5 abominably  negative
#> 6 abominate   negative
#> 7 abomination negative
#> 8 abort       negative
#> 9 aborted     negative
#> 10 aborts     negative

```

Since the `sentiments_bing` dataset and the `lastfm_words` dataset both share a column `word` (which has the same type of values) we can join these two datasets. The `sentiments_bing` dataset has 6786 unique words. Let's check how many distinct words our `lastfm_words` dataset has.

```

lastfm_words |> # dataset
distinct(word) |> # find unique words
nrow() # count distinct rows/ words

#> [1] 4614

```

One thing to note is that the `sentiments_bing` dataset does not include function words, that is words that are associated with closed-class categories (pronouns, determiners, prepositions, etc.) as these words do not have semantic content along the lines of positive and negative. So many of the words that appear in the `lastfm_words` will not be matched. Other thing to note is that the `sentiments_bing` lexicon will undoubtedly have words that do not appear in the `lastfm_words` and vice versa.

If we want to keep all the words in the `lastfm_words` and add the sentiment information for those words that do match in both datasets, we can use the `left_join()` function. `lastfm_words` will be the dataset on the 'left' and therefore all rows in this dataset will be retained.

Table 7.18: First 10 observations for the `lastfm_words` `sentiments_bing`‘ left join.

artist	song	genre	word	sentiment
Alan Jackson	Little Bitty	country	have	NA
Alan Jackson	Little Bitty	country	a	NA
Alan Jackson	Little Bitty	country	little	NA
Alan Jackson	Little Bitty	country	love	positive
Alan Jackson	Little Bitty	country	on	NA
Alan Jackson	Little Bitty	country	a	NA
Alan Jackson	Little Bitty	country	little	NA
Alan Jackson	Little Bitty	country	honeymoon	NA
Alan Jackson	Little Bitty	country	you	NA
Alan Jackson	Little Bitty	country	got	NA

```
left_join(lastfm_words, sentiments_bing) |>
  slice_head(n = 10) |> # first 10 observations
  kable(booktabs = TRUE)
```

So we see that quite a few of the words from `lastfm_words` are not matched. To focus in on those words in `lastfm_words` that do match, we’ll run the same join operation and filter for rows where `sentiment` is not empty (i.e. that there is a match in the `sentiments_bing` lexicon).

```
left_join(lastfm_words, sentiments_bing) |>
  filter(sentiment != "") |> # return matched sentiments
  slice_head(n = 10) |> # first 10 observations
  kable(booktabs = TRUE)
```

Let’s turn to another type of join: an anti-join. The purpose of an anti-join is to eliminate matches. This makes sense for a quick and dirty approach to removing function words (i.e. those grammatical words with little semantic content). In this case we use the `get_stopwords()` function to get the dataset. We’ll specify English as the language and we’ll use the default lexicon (‘Snowball’).

```
english_stopwords <-
  get_stopwords(language = "en") # get English stopwords
  ↵   from the Snowball lexicon

english_stopwords |>
  slice_head(n = 10) # preview first 10 observations
```

Table 7.19: First 10 observations for the `lastfm_words` sentiments\_bing‘ left join.

artist	song	genre	word	sentiment
Alan Jackson	Little Bitty	country	love	positive
Alan Jackson	Little Bitty	country	well	positive
Alan Jackson	Little Bitty	country	well	positive
Alan Jackson	Little Bitty	country	well	positive
Alan Jackson	Little Bitty	country	smile	positive
Alan Jackson	Little Bitty	country	well	positive
Alan Jackson	Little Bitty	country	well	positive
Alan Jackson	Little Bitty	country	well	positive
Alan Jackson	Little Bitty	country	smile	positive
Alan Jackson	Little Bitty	country	good	positive

```
#> # A tibble: 10 x 2
#>   word      lexicon
#>   <chr>    <chr>
#> 1 i        snowball
#> 2 me       snowball
#> 3 my       snowball
#> 4 myself   snowball
#> 5 we       snowball
#> 6 our      snowball
#> 7 ours     snowball
#> 8 ourselves snowball
#> 9 you      snowball
#> 10 your    snowball
```

Now if we want to eliminate stopwords from our `lastfm_words` dataset we use `anti_join()`. All the observations in the `lastfm_words` where there is not a match in `english_stopwords` will be returned.

```
anti_join(lastfm_words, english_stopwords) |>
  slice_head(n = 10) |>
  kable(booktabs = TRUE)
```

We can also merge datasets that we generate in our analysis or that we import from other sources. This can be useful when there are cases in which a corpus has associated metadata that is contained in files separate from the corpus itself. This is the case for the Switchboard Dialogue Act Corpus.

Our existing, disfluency recoded, version includes the following variables.

Table 7.20: First 10 observations in `lastfm_words` after filtering for English stopwords.

artist	song	genre	word
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	love
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	honeymoon
Alan Jackson	Little Bitty	country	got
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	dish
Alan Jackson	Little Bitty	country	got
Alan Jackson	Little Bitty	country	little
Alan Jackson	Little Bitty	country	spoon

```
sdac_disfluencies |> # dataset
  slice_head(n = 10) # preview first 10 observations

> # A tibble: 10 x 5
>   doc_id speaker_id utterance_text           filler count
>   <dbl>     <dbl> <chr>                  <chr>  <int>
> 1 4325      1632 Okay. /                   uh      0
> 2 4325      1632 Okay. /                   um      0
> 3 4325      1632 {D So, }                 uh      0
> 4 4325      1632 {D So, }                 um      0
> 5 4325      1519 [ [ I guess, +          uh      0
> 6 4325      1519 [ [ I guess, +          um      0
> 7 4325      1632 What kind of experience [ do you, + do you ] ~ uh
> 8 4325      1632 What kind of experience [ do you, + do you ] ~ um
> 9 4325      1519 I think, ] + {F uh, } I wonder ] if that work~ uh
> 10 4325     1519 I think, ] + {F uh, } I wonder ] if that work~ um
```

The online documentation page<sup>3</sup> provides a key file `caller_tab.csv` which contains speaker metadata information. Included in this `.csv` file is a column `caller_no` which contains the `speaker_id` we currently have in the `sdac_disfluencies` dataset. Let's read this file into our R session renaming `caller_no` to `speaker_id` to prepare to join these datasets.

```
sdac_speaker_meta <-
```

<sup>3</sup><https://catalog.ldc.upenn.edu/docs/LDC97S62/>

```

read_csv(file =
  ↪ "https://catalog.ldc.upenn.edu/docs/LDC97S62/caller_tab.csv",
  ↪
  col_names = c("speaker_id", # changed from
  ↪ `caller_no``,
    "pin",
    "target",
    "sex",
    "birth_year",
    "dialect_area",
    "education",
    "ti",
    "payment_type",
    "amt_pd",
    "con",
    "remarks",
    "calls_deleted",
    "speaker_partition"))

glimpse(sdac_speaker_meta)

#> Rows: 543
#> Columns: 14
#> $ speaker_id      <dbl> 1000, 1001, 1002, 1003, 1004, 1005, 1007, 1008, 1010~
#> $ pin              <dbl> 32, 102, 104, 5656, 123, 166, 274, 322, 445, 461, 57~
#> $ target            <chr> "N", "N", "N", "N", "Y", "N", "N", "N", "N", "Y~
#> $ sex               <chr> "FEMALE", "MALE", "FEMALE", "MALE", "FEMALE", "FEMAL~
#> $ birth_year        <dbl> 1954, 1940, 1963, 1947, 1958, 1956, 1965, 1939, 1932~
#> $ dialect_area       <chr> "SOUTH MIDLAND", "WESTERN", "SOUTHERN", "NORTH MIDLA~
#> $ education          <dbl> 1, 3, 2, 2, 2, 2, 1, 1, 2, 2, 1, 2, 2, 3, 3, 2, 3~
#> $ ti                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
#> $ payment_type        <chr> "CASH", "GIFT", "GIFT", "NONE", "GIFT", "GIFT", "CAS~
#> $ amt_pd              <dbl> 15, 10, 11, 7, 11, 22, 20, 3, 11, 9, 25, 9, 1, 16, 1~
#> $ con                 <chr> "N", "N", "N", "Y", "N", "Y", "N", "Y", "N", "N", "N~
#> $ remarks             <dbl> NA, ~
#> $ calls_deleted        <dbl> 2, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0~
#> $ speaker_partition <chr> "DN2", "XP", "XP", "DN2", "XP", "ET", "DN1", "DN1", ~

```

Now to join the `sdac_disfluencies` and `sdac_speaker_meta`. Let's turn to `left_join()` again as we want to retain all the observations (rows) from `sdac_disfluencies` and add the columns for `sdac_speaker_meta` where the `speaker_id` column values match.

```

sdac_disfluencies <-
  left_join(sdac_disfluencies, sdac_speaker_meta) # join
  ↵   by `speaker_id` 

glimpse(sdac_disfluencies)

#> Rows: 447,212
#> Columns: 18
#> $ doc_id      <dbl> 4325, 4325, 4325, 4325, 4325, 4325, 4325, 4325, 4325~ 
#> $ speaker_id   <dbl> 1632, 1632, 1632, 1632, 1519, 1519, 1632, 1632, 1519~ 
#> $ utterance_text <chr> "Okay. /", "Okay. /", "{D So, }", "{D So, }", "[ [~ 
#> $ filler        <chr> "uh", "um", "uh", "um", "uh", "um", "uh", "um", "uh"~ 
#> $ count         <int> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0~ 
#> $ pin            <dbl> 7713, 7713, 7713, 7713, 775, 775, 7713, 7713, 775, 7~ 
#> $ target         <chr> "N", "N~ 
#> $ sex             <chr> "FEMALE", "FEMALE", "FEMALE", "FEMALE", "FEMALE", "F~ 
#> $ birth_year     <dbl> 1962, 1962, 1962, 1962, 1971, 1971, 1962, 1962, 1971~ 
#> $ dialect_area    <chr> "WESTERN", "WESTERN", "WESTERN", "WESTERN", "SOUTH M~ 
#> $ education       <dbl> 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1~ 
#> $ ti              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~ 
#> $ payment_type    <chr> "CASH", "CASH", "CASH", "CASH", "CASH", "CASH", "C~ 
#> $ amt_pd          <dbl> 10, 10, 10, 10, 4, 4, 10, 10, 4, 4, 10, 10, 4, 4, 4~ 
#> $ con              <chr> "Y", "Y~ 
#> $ remarks          <dbl> NA, ~ 
#> $ calls_deleted    <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~ 
#> $ speaker_partition <chr> "UNC", "UNC", "UNC", "UNC", "UNC", "UNC", "UNC", "UNC", "UN~
```

Now there are some metadata columns we may want to keep and others we may want to drop as they may not be of importance for our analysis. I'm going to assume that we want to keep `sex`, `birth_year`, `dialect_area`, and `education` and drop the rest.

```

sdac_disfluencies <-
  sdac_disfluencies |> # dataset
  select(doc_id:count, sex:education) # subset key columns
```

## 7.5 Documentation

Documentation of the transformed dataset is just as important as the curated dataset. Therefore we use the same process as covered in the previous chapter. First we write the transformed dataset to disk and then we work to provide

Table 7.21: First 10 observations for the `sdac_disfluencies` dataset with speaker metadata.

doc_id	speaker_id	utterance_text	filler
4325	1632	Okay. /	uh
4325	1632	Okay. /	um
4325	1632	{D So, }	uh
4325	1632	{D So, }	um
4325	1519	[ [ I guess, +	uh
4325	1519	[ [ I guess, +	um
4325	1632	What kind of experience [ do you, + do you ] have, then with child care? /	uh
4325	1632	What kind of experience [ do you, + do you ] have, then with child care? /	um
4325	1519	I think, ] + {F uh, } I wonder ] if that worked. /	uh
4325	1519	I think, ] + {F uh, } I wonder ] if that worked. /	um

a data dictionary for this dataset. I've included the `data_dic_starter()` custom function to apply to our dataset(s).

```
data_dic_starter <- function(data, file_path) {
  # Function:
  # Creates a .csv file with the basic information
  # to document a curated dataset

  tibble(variable_name = names(data), # column with
         ← existing variable names
         name = "", # column for human-readable names
         description = "") |> # column for prose description
  write_csv(file = file_path) # write to disk
}
```

Let's apply our function to the `sdac_disfluencies` dataset using the R console (not part of our project script to avoid overwriting our documentation!).

```
data_dic_starter(data = sdac_disfluencies, file_path =
  ↵  "../data/derived/sdac/sdac_disfluencies_data_dictionary.csv")

data/derived/
  sdac/
    data_dictionary_sdac.csv
    sdac_curated.csv
```

```
sdac_disfluencies.csv  
sdac_disfluencies_data_dictionary.csv
```

Open the `data_dictionary_sdac_disfluencies.csv` file in spreadsheet software and add the relevant description of the dataset.

## 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. There are four general types of transformation steps: normalization, recoding, generation, and merging. In any given research project some or all of these steps will be employed –but not necessarily in the order presented in this chapter. Furthermore there may also be various datasets generated in at this stage each with a distinct analysis focus in mind. In any case it is important to write these datasets to disk and to document them according to the principles that we have established in the previous chapter.

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 inference, prediction, and/ or exploratory methods.

## Activities

### 💡 Recipe

**What:** Dataset manipulation: tokenization and joining datasets<sup>a</sup>

**How:** Read Recipe 8 and participate in the Hypothes.is online social annotation.

**Why:** To work with to 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.

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_8.html](https://lin380.github.io/tadr/articles/recipe_8.html)

 Lab

**What:** Dataset manipulation: tokenization and joining datasets<sup>a</sup>

**How:** Clone, fork, and complete the steps in Lab 8.

**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.

---

<sup>a</sup>[https://github.com/lin380/lab\\_8](https://github.com/lin380/lab_8)

---

## Questions

 Conceptual questions

1. ...
2. ...

 Technical exercises

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*

---

... Quote ...

### Keys

- ...

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 unsupervised learning or descriptive 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.

### Swirl

**What:** Unsupervised Learning<sup>a</sup>

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

**Why:** To ...

<sup>a</sup><https://github.com/lin380/swirl>

---

### 8.1 Orientation

The aim of this section is to provide an overview of 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 modeling, and vector space modeling. It will also include use cases of these methods.

### **8.1.1 Research goals**

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 Approaches**

- There is no fixed outcome variable rather there is only a set of predictors or covariates.
- The data is mutable, meaning that the data can be changed or modified as needed to address the research question.

#### **8.1.2.1 Analysis types**

There are two main types of exploratory 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 or both of these approaches can be used to explore a dataset.

### **8.1.3 Workflow**

Prerequisites: - A working research question - A dataset which aligns with the research question or hypothesis in terms of its sampling frame and the variables it contains or can be derived from the text - A set of preliminary interests and/ or linguistic variables to explore in the dataset that align with the research question

Process: - Identify and extract the variables of interest in the dataset - Interrogate the dataset using descriptive analysis and/ or unsupervised learning

### 8.1.3.1 Identify

- With the research question in mind, identify the variables of interest in the dataset
- Identify the linguistic variables that can be derived from the text (i.e. linguistic units: words, n-grams, sentences, etc.)
- Consider the operational measures of the variables derived from the text (i.e. frequency, dispersion, co-occurrence, etc.)
- Consider the other variables in the dataset that may be target for grouping or filtering the dataset (i.e. speaker information, document information, linguistic unit information, etc.)

### 8.1.3.2 Interrogate

Descriptive analysis:

- Frequency analysis
- Co-occurrence analysis

Unsupervised learning:

- Clustering
- Topic modeling
- Word embedding

### 8.1.3.3 Interpret

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

### 8.2.1 Descriptive analysis

#### 8.2.1.1 Frequency analysis

Frequency analysis is a descriptive method that counts the number of times a linguistic unit (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 linguistic units that are distinctive to a particular group or sub-group in the dataset.

- Raw frequency (counting)
  - Linguistic units (characters, words, n-grams, sentences, etc.)
  - Raw frequency (absolute frequency)
  - Zipf's law (rank-frequency)
  - Issues with raw frequency ( $f$ ):
    - \* Incomparable across sub-corpora, corpora, and time
- Term frequency (normalization)
  - Relative frequency (proportional frequency)
  - Makes samples (more) comparable (divergence of corpus sizes can influence validity of comparison)
  - Issues with term frequency ( $tf$ ):
    - \* Does not reveal the distribution of terms across a text or set of texts
    - \* Does not highlight distinctive words
      - One way to address this is to filter out 'stop words' (i.e. the, a, an, etc.)
- Adjusted frequency (relevance)
  - Dispersion (distribution)
    - \*  $idf$  (inverse document frequency)
    - \* ... Juilland's  $D$  (equal/ non-equal parts)
    - \* Gries'  $DP$  Deviation of Proportions (DP)
  - Weights
    - \*  $tf - idf$  (term frequency-inverse document frequency)
      - Distinctive words (Used to identify the most relevant keywords in a given text.)
      - Issues with  $tf - idf$ : Does not reveal the context of terms, Does not reveal the relationship between terms
    - \* weighted log odds (`tidylo` package<sup>1</sup>)
      - Distinctive words (Used to identify the most relevant keywords in a given text.)

<sup>1</sup><https://juliasilge.r-universe.dev/articles/tidylo/tidylo.html>

- Advantage over  $tf-idf$ : does a better job dealing with different combinations of words and documents having different counts.

### 8.2.1.2 Co-occurrence analysis

- Concordance
  - KWIC (keyword in context) This section will discuss keyword in context (KWIC) analyses, which is used to identify meaningful keywords in a given text. It will discuss various ways to analyse a text and extract keywords, as well as discuss various practical applications of KWIC in linguistics.
- Collocation
  - PMI (pointwise mutual information)
  - Dice's coefficient
  - ...

## 8.2.2 Unsupervised learning

### 8.2.2.1 Clustering

This section will discuss clustering techniques, which are used to partition data into clusters based on similarity. It will discuss various approaches to clustering, such as k-means and hierarchical clustering, as well as discuss their use cases in linguistics.

- K-means (pre-defined number of clusters)
- Hierarchical clustering (dendrogram)

### 8.2.2.2 Dimensionality reduction

- PCA (principal component analysis)
- MDS (multidimensional scaling)

### 8.2.2.3 Topic modeling

This section will discuss topic modeling techniques, which are used to identify and group semantically similar topics in unstructured data. It will discuss various approaches to topic modelling, such as Latent Dirichlet Allocation (LDA), and discuss their applications in linguistics.

- LDA (latent Dirichlet allocation)
- LSA (latent semantic analysis)

### 8.2.2.4 Word embedding

This section will discuss word embedding techniques, which are used to represent words in a vector space. It will discuss various approaches to word

embedding, such as Word2Vec and GloVe, and discuss their applications in linguistics.

- Word2vec (skip-gram)
  - GloVe (global vectors for word representation)
- 

### 8.3 Summary

Exploratory data analysis is a set of methods that can be used to explore a dataset and to identify new questions and new variables of interest. The methods can be used to describe a dataset and to identify linguistic units that are distinctive to a particular group or sub-group in the dataset. The methods can also be used to identify semantically similar topics in unstructured data. The results of exploratory analysis can be used to inform the development of a hypothesis or to inform the design of a machine learning model.

---

### Activities

#### 💡 Recipe

**What:** Exploratory methods: descriptive and unsupervised learning analysis methods<sup>a</sup>

**How:** Read Recipe 10 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.

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_11.html](https://lin380.github.io/tadr/articles/recipe_11.html)

#### 💡 Lab

**What:** Exploratory Data Analysis<sup>a</sup>

**How:** Clone, fork, and complete the steps in Lab 9.

**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.

<sup>a</sup>[https://github.com/lin380/lab\\_11](https://github.com/lin380/lab_11)

## Questions

### **i** 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?

### **i** Technical exercises

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*

---

... Quote ...

### Keys

- ...

In this chapter I present an introduction to approaches to data analysis known as machine learning, specifically supervised learning. In a nutshell, the aim of supervised learning is to leverage a potential relationship between a target or outcome variable and a set of other variables (features) derived from text to create a statistical generalization (model) that can accurately predict the values of the target variable using the values of the feature variables. We consider practical tasks as well as theoretical applications of the statistical learning in text analysis highlighting the standard workflow for building predictive models, testing and evaluating models, working to improve model performance, and how to interpret and report findings.

### Swirl

**What:** Supervised Learning<sup>a</sup>

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

**Why:** ...

<sup>a</sup><https://github.com/lin380/swirl>

---

### 9.1 Orientation

The aim of this section is to introduce the reader to the concept of supervised learning and to provide a brief overview of the workflow for building predictive models for text analysis.

### 9.1.1 Research goals

Supervised learning is a type of machine learning that 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 them to analyze language data to identify patterns or trends in language use, verify hypotheses, and prescribe actions. Supervised machine learning is an active area of research in linguistics, with many potential applications and areas for further exploration.

Predictive analyses are more inductive than exploratory analyses, which are more deductive. This means that we are more interested in the relationship between a particular outcome variable and a set of predictor variables than we are in the relationship between the predictor variables themselves, as we would be in an exploratory analysis. In this sense, we have a particular outcome in mind from the outset. On the other hand, the input variables are mutable, meaning that they can be changed to see how they affect the outcome –which points to the exploratory aspect of predictive analyses.

### 9.1.2 Approaches

- Outcome variable and any number of predictor variables
- Predictor variables are features derived from text and are mutable.

#### 9.1.2.1 Analysis types

There are two main types of supervised machine learning algorithms: classification, which is used to predict a categorical outcome such as the genre of a text, and regression, which is used to predict a continuous outcome such as the sentiment of a text.

- Supervised learning
  - Classification
    - \* Categorical outcome variable
  - Regression
    - \* Continuous outcome variable

### 9.1.3 Workflow

Prerequisites: - A working research question or hypothesis - A dataset which aligns with the research question or hypothesis in terms of its sampling frame and the variables it contains or can be derived from the text and a target variable to be predicted. - A set of preliminary features to be derived from the text that are used to predict the target variable

### 9.1.3.1 Identify

Data cleaning and feature extraction are the first steps in the process of preparing data for supervised machine learning.

In order to use supervised machine learning, linguists must first identify measurable properties of the text use as the input variables or features that are most likely to produce a model that performs well (i.e. that when used make accurate predictions). Once the feature types are identified, the data is processed to clean any extraneous elements and format the structure of the dataset given the requirements of the algorithm that will be used in subsequent steps.

### 9.1.3.2 Interrogate

Model training is the next step towards building a predictive model.

In this step, the data is split into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate the model's performance. The testing set is reserved and not used to train the model, so that the model's performance can be evaluated on data that it has not seen before.

The model is then trained on the training data and evaluated on the testing data. The results are then evaluated and the hyperparameters of the model may be adjusted to optimize its performance.

and the hyperparameters of the model may be adjusted to optimize its performance.

Hyperparameters are variables that are set prior to running a machine learning algorithm whose values influence the final result. In supervised machine learning, hyperparameters are typically used to control the learning process such as the learning rate, momentum, and batch size.

Some applications of supervised machine learning in linguistics include text classification, part-of-speech tagging, and language identification. Supervised machine learning is an active area of research in linguistics, with many potential applications and areas for further exploration.

### 9.1.3.3 Interpret

To either evaluate the training or testing set, the model is used to make predictions on the data in the set. The predictions are then compared to the actual values of the target variable in the set to evaluate the model's performance. So how is the model's performance evaluated?

*Classification*

Table 9.1: A labeled confusion matrix

	Predicted positive	Predicted negative
Actual positive	TP	FP
Actual negative	FN	TN

Table 9.2: Confusion matrix for a hypothetical model's performance on a test set

	Predicted positive	Predicted negative
Actual positive	100	10
Actual negative	20	50

For classification, there are a number of metrics that can be used to evaluate the performance of a model, including accuracy, precision, recall, and F1 score. 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 9.1.

Now let's fill this confusion matrix with hypothetical values, as seen in Table 9.2 to see how the metrics are calculated.

- Accuracy is defined as the proportion of correct predictions made by the model.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \quad (9.1)$$

The number of correct predictions is the sum of true positives and true negatives. So in our case this is  $100 + 50 = 150$ . The total number of predictions is the sum of all four cells in the confusion matrix, so in our case this is  $100 + 10 + 20 + 50 = 180$ . So the accuracy of our hypothetical model is  $150/180 = 0.833$ .

- Precision is defined as the proportion of positive predictions that are correct.

$$\text{Precision} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{false positives}} \quad (9.2)$$

- Recall is defined as the proportion of actual positives that are correctly identified.

$$\text{Recall} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{false negatives}} \quad (9.3)$$

- The F1 score is the harmonic mean of precision and recall.

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (9.4)$$

Area under the curve (AUC) is the area under the ROC (Receiver Operating Characteristic) curve, which is a graph of true positives (TPR) and false positives (FPR). The AUC is a measure of the model's performance across all possible classification thresholds. The AUC is a number between 0 and 1, where 0.5 represents a model that is no better than random guessing, and 1 represents a perfect model.

### Feature importance

- parallel coordinate visualization

In a supervised text classification task, you can use parallel coordinate plots to visualize the distribution of class labels across different feature dimensions. This can help identify which features are most informative for distinguishing between classes and inform feature selection or dimensionality reduction techniques.

```
# Libraries
library(tidyverse)
library(tidytext)

# Faux data
data <- data.frame(
  text = c("I love cats", "Cats are amazing", "I hate
  ↵ dogs", "Dogs are annoying"),
  class = c("positive", "positive", "negative",
  ↵ "negative")
)

# Tokenize data
tokenized_data <- data %>%
  unnest_tokens(word, text) %>%
  count(class, word) %>%
  ungroup() %>%
  spread(class, n, fill = 0)

# Normalize the term frequencies
normalized_data <-
  tokenized_data %>%
  mutate(
```

```

positive = positive / sum(positive),
negative = negative / sum(negative)
) %>%
gather(class, frequency, -word)

# Generate the parallel coordinate plot
ggplot(normalized_data, aes(x = class, y = frequency,
                           group = word, color = word)) +
  geom_line() +
  theme_minimal() +
  labs(
    title = "Parallel Coordinate Visualization",
    subtitle = "Text Classification Model Using Words as
               Features",
    x = "Class",
    y = "Normalized Term Frequency"
  )

```

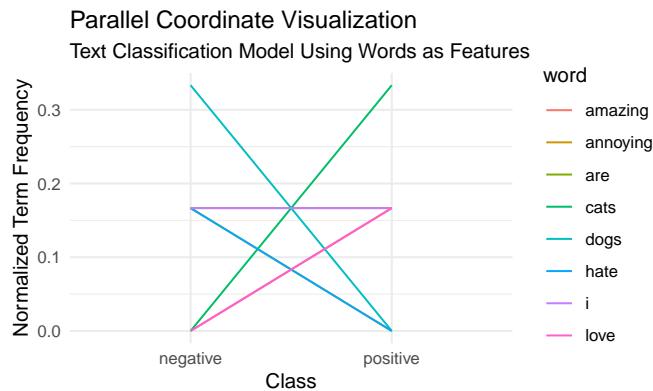


Figure 9.1: A parallel coordinate plot showing the distribution of class labels across different feature dimensions

We can look at the parallel coordinate plot in Figure 9.1 to see that the words “cats” and “love” are more common in the positive class, while the words “dogs” and “hate” are more common in the negative class. This suggests that these words are good features for distinguishing between the two classes.

### Regression

For regression, the most common metric is the root mean squared error (RMSE). The RMSE is the square root of the mean of the squared differences

Table 9.3: A table showing the differences between mean squared error, root mean squared error, and mean absolute error

error	formula	description
MSE	$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	The average of the squared differences between the predicted values and the actual values.
RMSE	$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$	The square root of the mean squared error.
MAE	$\frac{1}{n} \sum_{i=1}^n  y_i - \hat{y}_i $	The average of the absolute differences between the prediction and the actual data.

between the predicted values and the actual values. The lower the RMSE, the better the model fits the data.

Supervised machine learning algorithms for regression are typically evaluated using measures of error such as mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE). MSE is used to measure the average of the squares of the errors, MAE is the average of the absolute differences between the prediction and the actual data, and RMSE is the square root of the mean squared error. For each of these statistics, the lower the value, the better the model fits the data. The differences between these statistics are shown in Table 9.3.

The main advantages of using MSE, RMSE, and MAE are that they are all on the same scale as the dependent variable, and they are all differentiable, which makes them useful for optimization algorithms. MSE is the most commonly used metric for regression, but RMSE and MAE are also used. MSE is more sensitive to outliers than RMSE and MAE, so it is more useful when the data has outliers. RMSE and MAE are more useful when the data does not have outliers.

Plot the actual and predicted values to see how well the model fits the data.

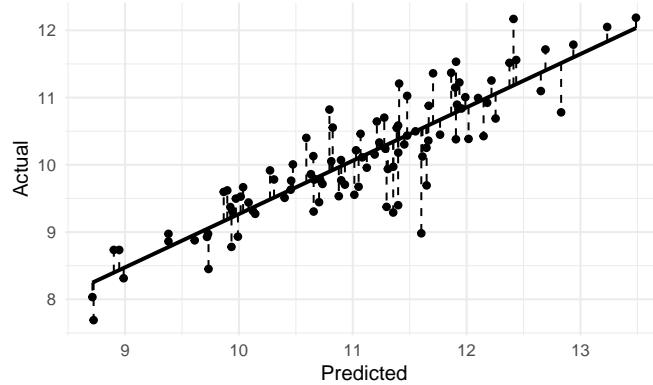


Figure 9.2: A plot of the actual and predicted values for a regression model

We can now apply our error metrics to the `results` data to see how well the model fits the data.

```
#/
# Calculate the error metrics for the `results` data
results |>
  mutate(error = actual - predicted) |> # calculate the
  ~ error
  summarise(
    mse = mean(error^2), # calculate the MSE
    rmse = sqrt(mse), # calculate the RMSE
    mae = mean(abs(error)), # calculate the MAE
    n = n()
  ) |> # calculate the number of observations
  mutate(
    mse = mse * 1 / n, # multiply by 1/n to get the MSE
    ~ for n observations
    rmse = rmse * 1 / n, # multiply by 1/n to get the RMSE
    ~ for n observations
    mae = mae * 1 / n
  ) # multiply by 1/n to get the MAE for n observations

> # A tibble: 1 x 4
>       mse     rmse     mae     n
>   <dbl>   <dbl>   <dbl> <int>
> 1 0.0113 0.0106 0.00947    100
```

Formula for calculating the MSE:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (9.5)$$

So we can implement this in R subtracting the actual values from the predicted values, squaring the differences, then taking the mean of all the squared differences, and finally multiplying by  $1/n$  to get the MSE for  $n$  observations.

```
results |> # use the `results` data
  mutate(error = actual - predicted) |> # calculate the
  ~ error
  summarise(
    mse = mean(error^2), # calculate the MSE
    n = n()
  ) |> # calculate the number of observations
```

```

mutate(mse = mse * 1 / n) |> # multiply by 1/n to get
  ↵ the MSE for n observations
  pull(mse) # pull the MSE value out of the tibble

> [1] 0.0113

```

Formula for calculating the RMSE:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (9.6)$$


---

## 9.2 Analysis

Recap and introduction to the structure of the analysis subsection.

- Introduce an algorithm
- Build a model
  - Preprocessing (tokenization, lemmatization, etc.)
  - Feature extraction (TF-IDF, word embeddings, etc.)
  - Model selection (logistic regression, SVM, etc.)
  - Model training
- Evaluate (and adjust) the model on the training data
  - Cross-validation
  - Evaluation metrics
  - Compare to null and/ or other models
  - Adjust the model (hyperparameters, regularization, etc.) as necessary
- Evaluate the model on the test data
  - Evaluation metrics
  - Evaluate feature importance
  - Evaluate the features of correct and incorrect predictions

### 9.2.1 Classification

We will first start with classification which is by far the most common text analysis approach in supervised machine learning. Again, classification is the task of predicting a categorical variable from a set of features. The features we use will be derived from the text but can take many forms. For example, we can use the raw text, the word counts, the TF-IDF values, or the word embeddings. We also will take into account the number of features we use. There is a trade-off, however, to the number of features: a) the more features we use, the more complex the model will be, and the more likely it will overfit the training data and b) the less features we use, the less complex the model

will be, and the more likely it will underfit the training data. To find the optimal number of features we can use a technique called cross-validation.

The most common text classification algorithms are logistic regression, k-nearest neighbors, Naive Bayes, and support vector machines. We will start with logistic regression and k-nearest neighbors as they are the simplest to understand and implement. We will then move on to Naive Bayes and support vector machines as they are more complex and require more explanation.

Building a null model for classification we simply predict the most common class in the training data. This makes sense as we have seen earlier, with categorical data the central tendency is estimated by the mode –i.e. the most common value.

### 9.2.1.1 K-nearest neighbors

K-nearest neighbors is a simple supervised machine learning method for classification. It is a non-parametric method, which means that it does not make any assumptions about the underlying distribution of the data. It is a lazy learning method, which means that it does not learn a discriminant function from the training data but instead stores the training data. It is a distance-based method, which means that it uses a distance metric to find the  $k$  nearest neighbors to a new observation. It is a simple method, which means that it is easy to understand and implement.

### 9.2.1.2 Logistic regression

Logistic regression is a supervised machine learning method for classification. It is a parametric method, which means that it makes assumptions about the underlying distribution of the data. It is an iterative method, which means that it uses an iterative algorithm to find the optimal parameters. To avoid overfitting it uses regularization such as ridge regression or lasso regression. These regularization methods penalize the model for having too many parameters. However, how does one know what the optimal number of parameters is? This is where cross-validation comes in.

### 9.2.1.3 Naive Bayes

Naive Bayes is a supervised machine learning method for classification. It is a parametric method, which means that it makes assumptions about the underlying distribution of the data. It is a probabilistic method, which means that it uses Bayes' theorem to calculate the probability of a class given the predictor variables. It is a generative method, which means that it learns the joint probability distribution of the predictor variables and the outcome variable. It is a simple method, which means that it makes the assumption that the predictor variables are independent of each other. This assumption is called the naive assumption. Now this assumption does not theoretically

hold for language data as words are not independent of each other. However, in practice, Naive Bayes' models still perform well on many text classification tasks.

#### 9.2.1.4 Decision trees

Decision trees for text classification are a supervised machine learning method for classification. They are a non-parametric method, which means that they do not make any assumptions about the underlying distribution of the data. They are a greedy method, which means that they use a greedy algorithm to find the optimal split of the predictor variables. They are a simple method, which means that they are easy to understand and implement.

In practical terms using decision trees for text classification can be very useful as they are easy to interpret. For example, we can see which words are most important for the classification of a text. However, they are prone to overfitting the training data. To avoid this we can use a technique called bagging. Bagging is a technique that uses multiple decision trees to make a prediction. The prediction is then the mode of the predictions of the individual decision trees. This is called a random forest.

### 9.2.2 Regression

In supervised machine learning regression tasks contrast to classification tasks as the outcome variable is continuous. A typical example outside of language would be to predict the price of a house given the number of bedrooms, the number of bathrooms, the size of the house, etc. For language this means that the labeled outcome variable is a number, not a class. For example, we can predict the number of words in a text given the number of characters in the text, the number of sentences in the text, etc. Other applications of regression in text analysis are sentiment analysis (where the outcome is a scalar value) and topic modeling (where the outcome is a probability distribution over topics).

#### 9.2.2.1 Linear regression

Linear regression can be used to predict a continuous outcome variable from a set of features. It is a parametric method, which means that it makes assumptions about the underlying distribution of the data. It is an iterative method, which means that it uses an iterative algorithm to find the optimal parameters. To avoid overfitting it uses regularization such as ridge regression or lasso regression. These regularization methods penalize the model for having too many parameters. However, how does one know what the optimal number of parameters is? This is where cross-validation comes in.

### 9.2.2.2 Decision trees (regression)

Decision trees for regression are a supervised machine learning method for regression. They are a non-parametric method, which means that they do not make any assumptions about the underlying distribution of the data. They are a greedy method, which means that they use a greedy algorithm to find the optimal split of the predictor variables. They are a simple method, which means that they are easy to understand and implement.

### 9.2.2.3 Neural networks

Neural networks are a supervised machine learning method for regression and classification. They are a non-parametric method, which means that they do not make any assumptions about the underlying distribution of the data. They are an iterative method, which means that they use an iterative algorithm to find the optimal parameters. They are a complex method, which means that they are difficult to understand and implement. However, they are very powerful and can be used to solve a wide range of problems. However, they are expensive to train and require a lot of data. It is often the case that a simpler method will perform just as well as a neural network in certain contexts.

---

## 9.3 Reporting

When reporting the results of a supervised machine learning model it is important to report the evaluation metrics that are most relevant to the problem at hand. For example, if the problem is to predict a continuous outcome, then the most relevant evaluation metric is the mean squared error (MSE). It is often useful to report the root mean squared error (RMSE) as well.

However, if the problem is to predict the class, then the most relevant evaluation metric is the accuracy. Other evaluation metrics that are often reported are the precision, recall, and F1-score. It can also be useful to report the confusion matrix. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives.

If the research goal is focused on prediction accuracy, then these statistics are the most relevant. But in other cases, the supervised learning algorithm is a means to gauge a relationship between the outcome and the predictor variables, namely to gauge the most important predictor variables. The model can then be used to identify those predictor variables that support accurate predictions and even to identify those predictor variables that do not support accurate predictions. Note, however, that some supervised learning algorithms are not able to identify the most important predictor variables. For example, neural networks are not able to identify the most important predictor variables.

These ‘black box’ algorithms may lead to accurate predictions, but they do not provide any insight into the underlying relationship between the outcome and the predictor variables.

---

## 9.4 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<sup>a</sup>

**How:** Read Recipe 10 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.

---

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_10.html](https://lin380.github.io/tadr/articles/recipe_10.html)

### 💡 Lab

**What:** Predictive Data Analysis<sup>a</sup>

**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.

<sup>a</sup>[https://github.com/lin380/lab\\_10](https://github.com/lin380/lab_10)

## Questions

### Conceptual questions

1. What is the difference between a continuous 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?

### Technical exercises

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 continuous 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*

People generally see what they look for, and hear what they listen for.

— Harper Lee, *To Kill a Mockingbird*

### Keys

- what are the three main types of inferential analysis approaches?
- how does the informational value of the dependent variable relate to the statistical approach adopted?
- how do descriptive, statistical, and evaluative steps work together to produce the reliable results?

In this chapter we consider approaches to deriving knowledge from information which can be generalized to the population from which the data is sampled. This process is known as statistical inference. The discussion here implements descriptive assessments, statistical tests, and model evaluation procedures for a series of contexts which are common in the analysis of corpus-based data. The chapter is structured into three main sections which correspond to the number of variables included in the statistical procedure. Each of these sections includes a subsection dedicated to the informational value of the dependent variable; the variable whose variation is to be explained.

For this discussion two datasets will be used as the base to pose various questions to submit for interrogation. It is of note that the questions in the subsequent sections are posited to highlight various descriptive, statistic, and evaluation procedures and do not reflect the standard approach to hypothesis testing which assumes that the null and alternative hypotheses are developed at the outset of the research project.

The process for each inferential data analysis in this section will include three steps: (1) descriptive assessment, (2) statistical interrogation, and (3) evaluation of the results.

 Swirl

**What:** Significance testing<sup>a</sup>

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

**Why:** To learn how to implement various statistical tests in common significance testing scenarios.

<sup>a</sup><https://github.com/lin380/swirl>

## 10.1 Orientation

The aim of this section is to provide an overview of inferential data analysis (IDA) and to introduce the three main types of inferential analysis approaches that are most common in text analysis research.

### 10.1.1 Research goals

The goal of IDA is to detect, explain, and generalize. The relationship to detect is predetermined by the hypothesis. In this situation the researcher will have identified an outcome variable (often known as a dependent variable) and often a predictor or set of predictor variables (independent variables) that are directly linked to the hypothesis in a way that the results of the statistical analysis allows to either confirm that null hypothesis or reject it. Explaining the results of the statistical analysis is key to summarizing the findings and how they relate to the hypothesis. To the extent that the text sample used is representative of the population from which the data is sampled, the results can be generalized to the population. IDA is not an exploratory endeavor and as such that analysis is performed on the data in a much more conservative manner than is the case in exploratory data analysis (EDA) or predictive data analysis (PDA).

### 10.1.2 Approaches

- The dependent variable and predictor variables are fixed (tied to hypothesis)
- Descriptive statistics and visualizations (plots or tables) are used to summarize the data and provide a preliminary assessment of the data
- Inferential statistics are used to test the hypothesis

### 10.1.2.1 Analysis types

The type of analysis that is performed depends most heavily on the informational value of the dependent variable. The informational value of the dependent variable is determined by the type of data that is collected. Secondly, the number and informational types of the independent variables (predictor variables) also play a role in determining the type of analysis that is performed.

- Categorical dependent variable
  - Descriptive statistics
    - \* Frequency
    - \* Proportion
    - \* Confidence intervals
  - Inferential statistics
    - \* Chi-square
    - \* Logistic regression
- Continuous dependent variable
  - Descriptive statistics
    - \* Mean
    - \* Standard deviation
    - \* Confidence intervals
  - Inferential statistics
    - \* Correlation
    - \* Linear regression

### 10.1.3 Workflow

Prerequisites: - A testable hypothesis (covering the outcome space, i.e. null and alternative hypotheses). - A data set that aligns with the population targeted to generalize to. - A operationalized dependent and independent variable(s) that are tied to the hypothesis.

#### 10.1.3.1 Identify

- Identify the informational value of the dependent and independent variable(s).
- Assess the distribution of the independent and dependent variables with the appropriate descriptive statistics and visualizations.
- Choose an appropriate statistical test based on the informational value and distribution of the dependent and independent variables.
- Apply transformations to the data as needed to meet the assumptions of the statistical tests.

#### 10.1.3.2 Interrogate

- Apply the appropriate statistical test to the data:
  - Categorical dependent variable

- \* Chi-square (dependent and one independent variable)
- \* Logistic regression (dependent and one or more independent variables)
- Continuous dependent variable
  - \* Correlation (dependent and one independent variable)
  - \* Linear regression (dependent and one or more independent variables)
- Assess the results of the statistical test (p-value, confidence intervals, effect size)

#### 10.1.3.3 Interpret

Review the results of the statistical test and interpret the results in the context of the hypothesis.

---

## 10.2 Analysis

Recap and introduction to the structure of the analysis subsection.

- Categorical dependent variable
  - Categorical/ continuous independent variable(s)
    - \* Descriptive assessment
      - 0-1 categorical independent variable: Tables summary statistics (contingency table)
      - Continuous independent variable(s): Plots and summary statistics (boxplots, histograms, etc.)
    - \* Statistical interrogation
      - Chi-square (dependent and one independent variable)
      - Logistic regression (dependent and one or more independent variables)
    - \* Evaluation of results
      - p-value
      - Confidence intervals
      - Effect size
  - Continuous dependent variable
    - Continuous/ categorical independent variable(s)
      - \* Descriptive assessment
        - Tables, plots, and summary statistics
      - \* Statistical interrogation
        - Correlation
        - Linear regression
      - \* Evaluation of results
        - p-value

- Confidence intervals
- Effect size

### 10.2.1 Categorical dependent variable

#### 10.2.1.1 Chi-square

Chi-square tests can be used for frequencies of categorical variables. The chi-square goodness of fit test is used to test whether the observed frequencies of a categorical variable match the expected frequencies of a single variable. The chi-square test of independence is used to test whether the observed frequencies of two categorical variables are independent of each other. For hypothesis tests that include more than two variables, the chi-square test is not appropriate. Instead, logistic regression is used.

#### 10.2.1.2 Logistic regression

Logistic regression can handle more than one independent variable, and these variables need not be categorical. The dependent variable is a binary variable, and the independent variables can be continuous or categorical. The logistic regression model is used to predict the probability of the dependent variable being 1. The logistic regression model is used to test whether the independent variables are associated with the dependent variable.

Using the `infer` package, a logistic regression can be performed using the `specify()` and `generate()` functions. The `specify()` function is used to specify the dependent and independent variables. The `generate()` function is used to generate the model. The `calculate()` function is used to calculate the p-value and confidence intervals.

```
library(infer) # for inferential statistics

# load the `dative` data from the `languageR` package
data(package = "languageR", data = "dative")

# specify the dependent `RealizationOfRecipient` and
# independent variables `LengthOfTheme` and
# `AnimacyOfTheme` for a logistic regression model
log_reg <-
  dative %>%
  specify(RealizationOfRecipient ~ LengthOfTheme +
    AnimacyOfTheme) %>%
  generate(response = "logistic", type = "response") %>%
  calculate(stat = "p.value", order = "descending")
```

### 10.2.2 Continuous dependent variable

#### 10.2.2.1 Correlation

#### 10.2.2.2 Linear regression

---

### 10.3 Reporting

---

### 10.4 Summary

---

## Activities

#### 💡 Recipe

**What:** Statistical inference: prep, assess, interrogate, evaluate, and report<sup>a</sup>

**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.

<sup>a</sup>[https://lin380.github.io/tadr/articles/recipe\\_9.html](https://lin380.github.io/tadr/articles/recipe_9.html)

#### 💡 Lab

**What:** Statistical inference: prep, assess, interrogate, evaluate, and report<sup>a</sup>

**How:** Clone, fork, and complete the steps in Lab 9.

**Why:** To gain experience working with coding strategies to prepare, assess, interrogate, evaluate, and report results from an inferential data analysis, practice transforming datasets and visualizing relationships, and implement organizational strategies for organizing and reporting results in a reproducible fashion.

<sup>a</sup>[https://github.com/lin380/lab\\_9](https://github.com/lin380/lab_9)

---

## Questions

**i** Conceptual questions

1. What is the difference between a descriptive and inferential analysis?
2. What are the steps involved in conducting a descriptive analysis?
3. What are the steps involved in conducting an inferential analysis?
4. What are the steps involved in preparing data for inferential analysis?
5. What are the steps involved in conducting a statistical interrogation?
6. What are the steps involved in evaluating the results of an inferential analysis?
7. What are the steps involved in reporting the results of an inferential analysis?
8. Would word frequency differences between two groups of words be better assessed using a t-test or a chi-squared distribution?
9. Would word lengths between two groups of words be better assessed using a t-test or a chi-squared distribution?
10. What type of visualization would be best for exploring the relationship between two categorical variables?
11. What type of visualization would be best for exploring the relationship between two non-categorical variables?
12. What type of visualization would be best for exploring the relationship between a categorical and a non-categorical variable?
13. What is the role of confidence intervals in inferential data analysis? How is this similar or different to the role of p-values?
14. What is the role of effect size in inferential data analysis? How is this similar or different to the role of p-values?

**i** Technical exercises

1. Use the lm() function to create a linear model, assess the summary statistics, and evaluate the results.
2. Use the glm() function to assess the relationship between two categorical variables and evaluate the results.

3. Apply a chi-squared distribution to explore categorical dependent variables and evaluate the results.
4. Calculate correlation coefficients between two non-categorical variables and evaluate the results.
5. Read in a dataset and transform it to prepare it for inferential analysis.
6. Decide which type of visualization is most appropriate for the dataset and then implement it using ggplot2.
7. Use the effectsize() function to calculate effect size and confidence intervals.

## 10.5 Preparation

At this point let's now get familiar with the datasets and prepare them for analysis. The first dataset to consider is the `dative` dataset. This dataset can be loaded from the languageR package (R. H. Baayen and Shafeai-Bajestan 2019).

```
dative <-  
  languageR::dative |> # load the `dative` dataset  
  as_tibble() # convert the data frame to a tibble object  
  
glimpse(dative) # preview structure  
  
#> Rows: 3,263  
#> Columns: 15  
#> $ Speaker <fct> NA, NA,~  
#> $ Modality <fct> written, written, written, written, wr~  
#> $ Verb <fct> feed, give, give, give, offer, give, pay, bring~  
#> $ SemanticClass <fct> t, a, a, a, c, a, t, a, a, a, a, t, a, c, a,~  
#> $ LengthOfRecipient <int> 1, 2, 1, 1, 2, 2, 2, 1, 1, 1, 2, 2, 1, 1,~  
#> $ AnimacyOfRec <fct> animate, animate, animate, animate, anim~  
#> $ DefinOfRec <fct> definite, definite, definite, definite, definit~  
#> $ PronomOfRec <fct> pronominal, nonpronominal, nonpronominal, prono~  
#> $ LengthOfTheme <int> 14, 3, 13, 5, 3, 4, 4, 1, 11, 2, 3, 3, 5, 2, 8,~  
#> $ AnimacyOfTheme <fct> inanimate, inanimate, inanimate, ina~  
#> $ DefinOfTheme <fct> indefinite, indefinite, definite, indefinite, d~  
#> $ PronomOfTheme <fct> nonpronominal, nonpronominal, nonpronominal, no~  
#> $ RealizationOfRecipient <fct> NP, NP, NP, NP, NP, NP, NP, NP, NP,~  
#> $ AccessOfRec <fct> given, given, given, given, given, given, given~  
#> $ AccessOfTheme <fct> new, new, new, new, new, new, new, accessi~
```

Table 10.1: ?(caption)

From `glimpse()` we can see that this dataset contains 3,263 observations and 15 columns.

The R Documentation can be consulted using `?dative` in the R Console. The description states:

Data describing the realization of the dative as NP or PP in the Switchboard corpus and the Treebank Wall Street Journal collection.

For a bit more context, a dative is the phrase which reflects the entity that takes the recipient role in a ditransitive clause. In English, the recipient (dative) can be realized as either a noun phrase (NP) as seen in (1) or as a prepositional phrase (PP) as seen in (2) below.

1. They give [you NP] a drug test.
2. They give a drug test [to you PP].

Together these two syntactic options are known as the Dative Alternation.

The observational unit for this dataset is `RealizationOfRecipient` variable which is either ‘NP’ or ‘PP’. For the purposes of this chapter I will select a subset of the key variables we will use in the upcoming analyses and drop the others.

```
dative <-
  dative |> # dataset
  select(RealizationOfRecipient, Modality,
    ~ LengthOfRecipient, LengthOfTheme) |> # select key
    ~ variables
  janitor::clean_names() # normalize variable names
```

realization_of_recipient	modality	length_of_recipient	length_of_theme
NP	written	1	14
NP	written	2	3
NP	written	1	13
NP	written	1	5
NP	written	2	3
NP	written	2	4
NP	written	2	4
NP	written	1	1
NP	written	1	11
NP	written	1	2

Table 10.2: Data dictionary for the `dative` dataset.

variable_name	name	description
realization_of_recipient	Realization of Recipient	A factor with levels NP and PP coding the realization
modality	Language Modality	A factor with levels *spoken*, *written*.
length_of_recipient	Length of Recipient	A numeric vector coding the number of words comprising the recipient
length_of_theme	Length of Theme	A numeric vector coding the number of words comprising the theme

In Table 10.2 I've created a data dictionary describing the variables in our new `dative` dataset based on the variable descriptions in the `languageR::dative` documentation.

The second dataset that we will use in this chapter is the `sdac_disfluencies` dataset that we worked to derived in the previous chapter. Let's read in the dataset and preview the structure.

```
sdac_disfluencies <-
  read_csv(file =
    ↪  "../data/derived/sdac/sdac_disfluencies.csv") # read
    ↪  transformed dataset

glimpse(sdac_disfluencies) # preview structure

#> Rows: 447,212
#> Columns: 9
#> $ doc_id      <dbl> 4325, 4325, 4325, 4325, 4325, 4325, 4325, 4325, 4325, 4~
#> $ speaker_id   <dbl> 1632, 1632, 1632, 1632, 1519, 1519, 1632, 1632, 1519, 1~
#> $ utterance_text <chr> "Okay. /", "Okay. /", "[D So, ]", "[D So, ]", "[ [ I ~
#> $ filler        <chr> "uh", "um", "uh", "um", "uh", "um", "uh", "um", "uh", "~
#> $ count         <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
#> $ sex            <chr> "FEMALE", "FEMALE", "FEMALE", "FEMALE", "FEMALE", "FEMA~
#> $ birth_year     <dbl> 1962, 1962, 1962, 1971, 1971, 1962, 1962, 1971, 1971, 1~
#> $ dialect_area   <chr> "WESTERN", "WESTERN", "WESTERN", "WESTERN", "SOUTH MIDL~
#> $ education       <dbl> 2, 2, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1~
```

We prepared a data dictionary that reflects this transformed dataset. Let's read that file and then view it in Table 10.3.

```
sdac_disfluencies_dictionary <- read_csv(file =
  ↪  "../data/derived/sdac/sdac_disfluencies_data_dictionary.csv")
  ↪  # read data dictionary
```

For our analysis purposes we will reduce this dataset, as we did for the `dative` dataset, retaining only the variables of interest for the upcoming analyses.

Table 10.3: Data dictionary for the `sdac_disfluencies` dataset.

variable_name	name	description
doc_id	Document ID	Unique identifier for each conversation file.
speaker_id	Speaker ID	Unique identifier for each speaker in the corpus.
utterance_text	Utterance Text	Transcribed utterances for each conversation. Includes disfluency annotations.
filler	Filler	Filler type either uh or um.
count	Count	Number of fillers for each utterance.
sex	Sex	Sex for each speaker either male or female.
birth_year	Birth Year	The year each speaker was born.
dialect_area	Dialect Area	Region from the US where the speaker spent first 10 years.
education	Education	Highest educational level attained: values 0, 1, 2, 3, and 9.

Table 10.4: First 10 observations of simplified `sdac_disfluencies` dataset.

speaker_id	filler	count	sex	birth_year	education
1632	uh	0	FEMALE	1962	2
1632	um	0	FEMALE	1962	2
1632	uh	0	FEMALE	1962	2
1632	um	0	FEMALE	1962	2
1519	uh	0	FEMALE	1971	1
1519	um	0	FEMALE	1971	1
1632	uh	0	FEMALE	1962	2
1632	um	0	FEMALE	1962	2
1519	uh	1	FEMALE	1971	1
1519	um	0	FEMALE	1971	1

```
sdac_disfluencies <-
  sdac_disfluencies |> # dataset
  select(speaker_id, filler, count, sex, birth_year,
    ~ education) # select key variables
```

Let's preview this simplified `sdac_disfluencies` dataset.

Now the `sdac_disfluencies` dataset needs some extra transformation to better prepare it for statistical interrogation. On the one hand the variables `birth_year` and `education` are not maximally informative. First it would be more ideal if `birth_year` would reflect the age of the speaker at the time of the conversation(s) and furthermore the coded values of `education` are not explicit as far what the numeric values refer to.

The second issue has to do with preparing the `sdac_disfluencies` dataset

for statistical analysis. This involves converting our column types to the correct vector types for statistical methods. Specifically we need to convert our categorical variables to the R type ‘factor’ (fct). This includes of our current variables which are character vectors, but also the `speaker_id` and `education` which appear as numeric but do not reflect a continuous variables; one is merely a code which uniquely labels each speaker and the other is an ordinal list of educational levels.

This will be a three step process, first we will normalize the `birth_year` to reflect the age of the speaker, second we will convert all the relevant categorical variables to factors, and third we will convert the `education` variable to a factor adding meaningful labels for the levels of this factor.

Consulting the online manual for this corpus<sup>1</sup>, we see that the recording date for these conversations took place in 1992, so we can simply subtract the `birth_year` from 1992 to get each participant’s age. We’ll rename this new column `age` and drop the `birth_year` column.

```
sdac_disfluencies <-
  sdac_disfluencies |> # dataset
  mutate(age = (1992 - birth_year)) |> # calculate age
  select(-birth_year) # drop `birth_year` column
```

Now let’s convert all the variables which are character vectors. We can do this using the the `factor()` function; first on `speaker_id` and then, with the help of `mutate_if()`, to all the other variables which are character vectors.

```
sdac_disfluencies <-
  sdac_disfluencies |> # dataset
  mutate(speaker_id = factor(speaker_id)) |> # convert
    ↵ numeric to factor
  mutate_if(is.character, factor) # convert all character
    ↵ to factor
```

We know from the data dictionary that the `education` column contains four values (0, 1, 2, 3, and 9). Again, consulting the corpus manual we can see what these values mean.

EDUCATION	COUNT
<hr/>	
0	14 less than high school

<sup>1</sup>[https://catalog.ldc.upenn.edu/docs/LDC97S62/swb1\\_manual.txt](https://catalog.ldc.upenn.edu/docs/LDC97S62/swb1_manual.txt)

1	39	less than college
2	309	college
3	176	more than college
9	4	unknown

So let's convert `education` to a factor adding these descriptions as factor level labels. The function `factor()` can take an argument `labels =` which we can manually assign the label names for the factor levels in the order of the factor levels. Since the original values were numeric, the factor level ordering defaults to ascending order.

```
sdac_disfluencies <-
  sdac_disfluencies |> # dataset
  mutate(education = factor(education,
    labels = c("less than high
      ↵ school", # value 0
      "less than
        ↵ college", #
        ↵ value 1
      "college", # value
        ↵ 2
      "more than
        ↵ college", #
        ↵ value 3
      "unknown")) # ↵ value 9
```

So let's take a look at the `sdac_disfluencies` dataset we've prepared for analysis.

```
glimpse(sdac_disfluencies)

#> Rows: 447,212
#> Columns: 6
#> $ speaker_id <fct> 1632, 1632, 1632, 1632, 1519, 1519, 1632, 1632, 1519, 1519, ~
#> $ filler      <fct> uh, um, uh, ~
#> $ count       <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, ~
#> $ sex         <fct> FEMALE, FEMALE, FEMALE, FEMALE, FEMALE, FEMALE, FEMALE, FEM~
#> $ education   <fct> college, college, college, college, less than college, less~
#> $ age          <dbl> 30, 30, 30, 30, 21, 21, 30, 30, 21, 21, 30, 30, 21, 21, 21, ~
```

Now the datasets `dative` and `sdac_disfluencies` are ready to be statistically interrogated.

## 10.6 Univariate analysis

In what follows I will provide a description of inferential data analysis when only one variable is to be interrogated. This is known as a univariate analysis, or one-variable analysis. We will consider a case when the variable is categorical and the other continuous.

### 10.6.1 Categorical

As an example of a univariate analysis where the variable used in the analysis is categorical we will look at the `dative` dataset. In this analysis we may be interested in knowing whether the recipient role in a ditransitive construction is realized more as an ‘NP’ or ‘PP’.

#### Descriptive assessment

The `realization_of_recipient` variable contains the relevant information. Let’s take a first look using the `skimr` package.

```
dative |> # dataset
  select(realization_of_recipient) |> # select the
    ↵ variable
  skim() |> # get data summary
  yank("factor") # only show factor-oriented information
```

#### Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
realization_of_recipient	0	1	FALSE	2	NP: 2414, PP: 849

The output from `skim()` produces various pieces of information that can be helpful. On the one hand we get diagnostics that tell us if there are missing cases (`NA` values), what the proportion of complete cases is, if the the factor is ordered, how many distinct levels the factor has, as well as the level counts.

Looking at the `top_counts` we can see that of the 3,263 observations, in 2,414 the dative is expressed as an ‘NP’ and 849 as ‘PP’. Numerically we can see that there is a difference between the use of the alternation types. A visualization is often helpful for descriptive purposes in statistical analysis. In this particular case, however, we are considering a single categorical variable with only two levels (values) so a visualization is not likely to be more informative than the numeric values we have already obtained. But for demonstration purposes and to get more familiar with building plots, let’s create a visualization.

```
dative |> # dataset
  ggplot(aes(x = realization_of_recipient)) + # mapping
  geom_bar() + # geometry
  labs(x = "Realization of recipient", y = "Count") # 
    ↵   labels
```

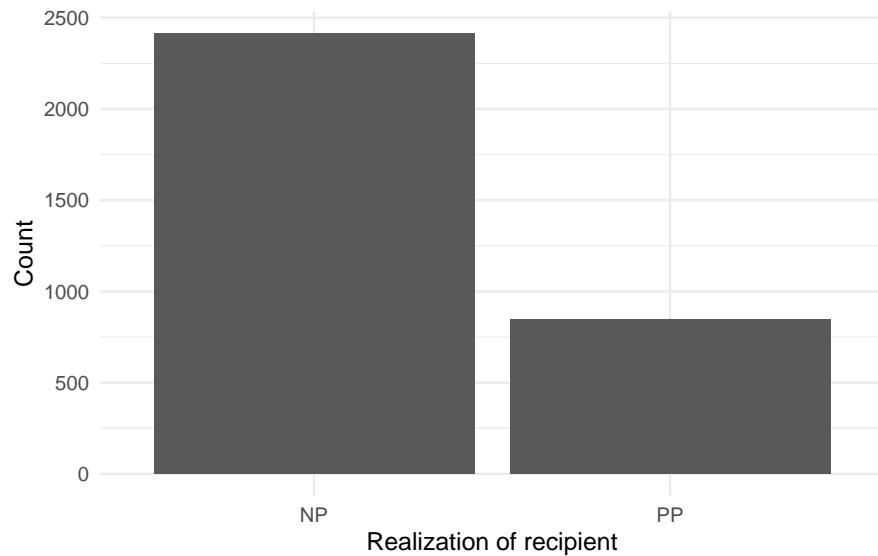


Figure 10.1: Barplot visualizing the realization of recipient

The question we want to address, however, is whether this numerical difference is in fact a statistical difference.

### Statistical interrogation

To statistically assess the distribution for a categorical variable, we will turn to the Chi-squared test. This test aims to gauge whether the numerical differences between 'NP' and 'PP' counts observed in the data is greater than what would be expected by chance. Chance in the case where there are only two possible outcome levels is 50/50. For our particular data where there are 3,263 observations half would be 'NP' and the other half 'PP' –specifically 1631.5 for each.

To run this test we first will need to create a cross-tabulation of the variable. We will use the `xtabs()` function to create the table.

```
ror_table <-
```

```

xtabs(formula = ~ realization_of_recipient, # formula
      ↵ selecting the variable
      data = dative) # dataset

ror_table # preview

#> realization_of_recipient
#>   NP    PP
#> 2414  849

```

No new information here, but the format (i.e. an object of class ‘table’) is what is important for the input argument for the `chisq.test()` function we will use to run the test.

```

c1 <- chisq.test(x = ror_table) # apply the chi-squared
                                ↵ test to `ror_table`

c1 # preview the test results

#>
#> Chi-squared test for given probabilities
#>
#> data: ror_table
#> X-squared = 751, df = 1, p-value <2e-16

```

The preview of the `c1` object reveals the main information of interest including the Chi-squared statistic, the degrees of freedom, and the *p*-value (albeit in scientific notation). However, the `c1` is an ‘htest’ object and includes a number of other pieces information about the test.

```

names(c1) # preview column names

#> [1] "statistic" "parameter" "p.value"     "method"      "data.name" "observed"
#> [7] "expected"   "residuals"  "stdres"

```

For our purposes let’s simply confirm that the *p*-value is lower than the standard .05 threshold for statistical significance.

```

c1$p.value < .05 # confirm p-value below .05

#> [1] TRUE

```

Other information can be organized in a more readable format using the broom package’s `augment()` function.

```

c1 |> # statistical result
augment() # view detailed statistical test information

#> # A tibble: 2 x 6
#>   realization_of_recipient .observed .prop .expected .resid .std.resid
#>   <fct>                 <int> <dbl>    <dbl>    <dbl>    <dbl>
#> 1 NP                   2414  0.740    1632.   19.4     27.4
#> 2 PP                   849   0.260    1632.  -19.4    -27.4

```

Here we can see the observed and expected counts and the proportions for each level of `realization_of_recipient`. We also get additional information concerning residuals, but we will leave these aside.

### Evaluation

At this point we may think we are done. We have statistically interrogated the `realization_of_recipient` variable and found that the difference between ‘NP’ and ‘PP’ realization in the datives in this dataset is statistically significant. However, we need to evaluate the size (‘effect size’) and the reliability of the effect (‘confidence interval’). The `effectsize` package provides a function `effectsize()` that can provide us both the effect size and confidence interval.

```

effects <-
  effectsize(c1) # evaluate effect size and generate a
  ↪ confidence interval (fei type given 2x1 contingency
  ↪ table)

effects # preview effect size and confidence interval

#> Fei |      95% CI
#> -----
#> 0.48 | [0.45, 1.00]
#>
#> - Adjusted for uniform expected probabilities.
#> - One-sided CIs: upper bound fixed at [1.00].

```

`effectsize()` recognizes the type of test results in `c1` and calculates the appropriate effect size measure and generates a confidence interval. Since the effect statistic (“Fei”) falls between the 95% confidence interval this suggests the results are reliably interpreted (chances of Type I (false positive) or Type II (false negative) are low).

Now, the remaining question is to evaluate whether the significant result here is a strong effect or not. To do this we can pass the effect size measure to the `interpret_r()` function.

```
interpret_r(effects$Fei) # interpret the effect size
#> [1] "very large"
#> (Rules: funder2019)
```

Turns out we have a strong effect; the realization of dative alternation heavily favors the ‘NP’ form in our data. The potential reasons why are not considered in this univariate analysis, but we will return to this question later as we add independent variables to the statistical analysis.

### 10.6.2 Continuous

Now let’s turn to a case when the variable we aim to interrogate is non-categorical. For this case we will turn to the `sdac_disfluencies` dataset. Specifically we will aim to test whether the use of fillers is normally distributed across speakers.

 Tip

This is an important step when working with numeric dependent variables as the type of distribution will dictate decisions about whether we will use parametric or non-parametric tests if we consider the extent to which an independent variable (or variables) can explain the variation of the dependent variable.

Since the dataset is currently organized around fillers as the observational unit, I will first transform this dataset to sum the use of fillers for each speaker in the dataset.

```
sdac_speaker_fillers <-
  sdac_disfluencies |> # dataset
  group_by(speaker_id) |> # group by each speaker
  summarise(sum = sum(count)) |> # add up all fillers used
  ungroup() # remove grouping parameter
```

#### Descriptive assessment

Let’s perform some descriptive assessment of the variable of interest `sum`. First let’s apply the `skim()` function and retrieve just the relevant numeric descriptors with `yank()`. One twist here, however, is that I’ve customized the `skim()` function using the `skim_with()` to remove the default histogram and add the Interquartile Range (IQR) to the output. This new `skim` function `num_skim()` will take the place of `skim()`.

Table 10.5: First 10 observations of `sdac_speaker_fillers` dataset.

speaker_id	sum
155	28
1000	45
1001	264
1002	54
1004	45
1005	129
1007	0
1008	27
1010	2
1011	54

```

num_skim <-
  skim_with(numeric = sf1(hist = NULL, # remove hist skim
                         iqr = IQR)) # add IQR
  ↵   to skim

sdac_speaker_fillers |> # dataset
  select(sum) |> # variable of interest
  num_skim() |> # get custom data summary
  yank("numeric") # only show numeric-oriented information

```

#### Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	iqr
sum	0	1	87.1	108	0	16	45	114	668	98

We see here that the mean use of fillers is 87.1 across speakers. However, the standard deviation and IQR are large relative to this mean which indicates that the dispersion is quite large, in other words this suggests that there are large differences between speakers. Furthermore, since the median (p50) is smaller than the mean, the distribution is right skewed.

Let's look a couple visualizations of this distribution to appreciate these descriptives. A histogram will provide us a view of the distribution using the counts of the values of `sum` and a density plot will provide a smooth curve which represents the scaled distribution of the observed data.

```

p1 <-
  sdac_speaker_fillers |> # dataset
  ggplot(aes(sum)) + # mapping

```

```

geom_histogram() + # geometry
labs(x = "Fillers", y = "Count")

p2 <-
  sdac_speaker_fillers |> # dataset
  ggplot(aes(sum)) + # mapping
  geom_density() + # geometry
  geom_rug() + # visualize individual observations
  labs(x = "Fillers", y = "Density")

p1 + p2 + plot_annotation("Filler count distributions.")

```

Filler count distributions.

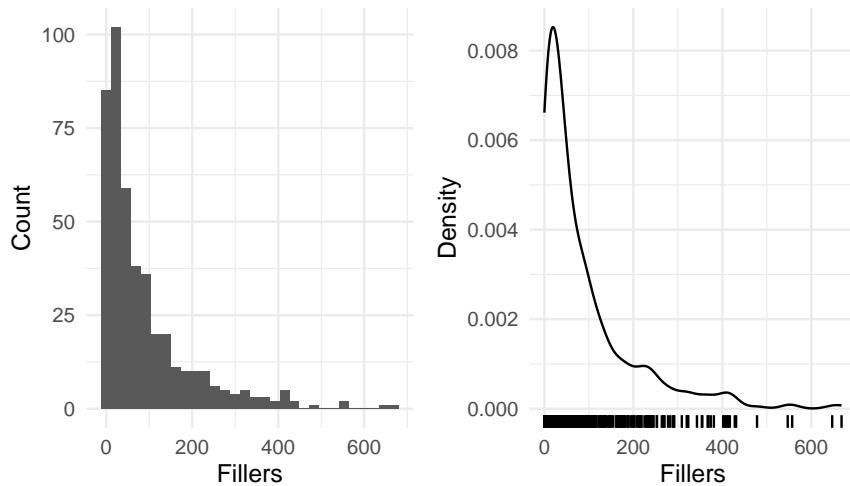


Figure 10.2: ?(caption)

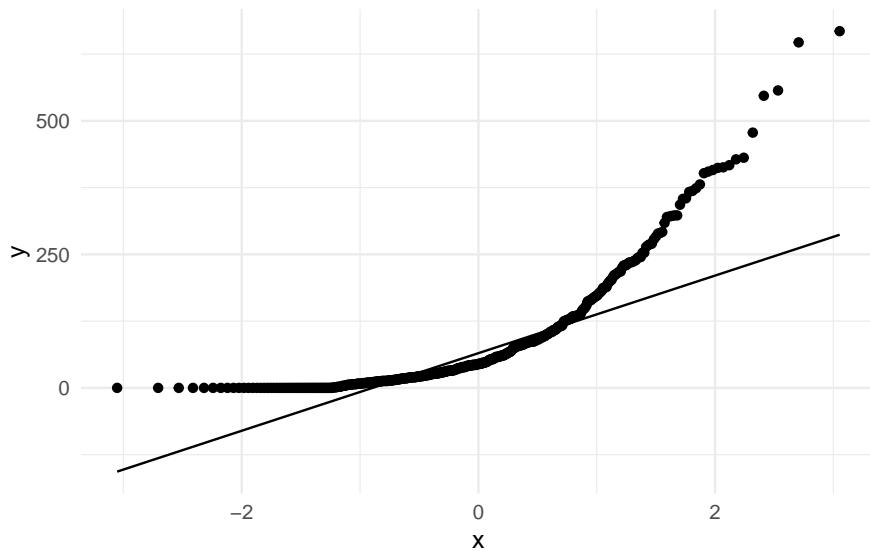
From the plots in Figure 10.2 we see that our initial intuitions about the distribution of `sum` are correct. There is large dispersion between speakers and the data distribution is right skewed.

 Tip

Note that I've used the `patchwork` package for organizing the display of plots and including a plot annotation label.

Since our aim is to test for normality, we can generate a Quantile-Quantile plots (QQ Plot).

```
sdac_speaker_fillers |> # dataset
  ggplot(aes(sample = sum)) + # mapping
  stat_qq() + # calculate expected quantile-quantile
  # distribution
  stat_qq_line() # plot the qq-line
```



Since many points do not fall on the expected normal distribution line we have even more evidence to support the notion that the distribution of `sum` is non-normal.

### Statistical interrogation

Although the descriptives and visualizations strongly suggest that we do not have normally distributed data let's run a normality test. For this we turn to the `shapiro.test()` function which performs the Shapiro-Wilk test of normality. We pass the `sum` variable to this function to run the test.

```
s1 <- shapiro.test(sdac_speaker_fillers$sum) # apply the
# normality test to `sum`

s1 # preview the test results

#>
#> Shapiro-Wilk normality test
#>
#> data: sdac_speaker_fillers$sum
```

```
#> W = 0.8, p-value <2e-16
```

As we saw with the results from the `chisq.test()` function, the `shapiro.test()` function produces an object with information about the test including the *p*-value. Let's run our logical test to see if the test is statistically significant.

```
s1$p.value < .05 #
```

```
#> [1] TRUE
```

### Evaluation

The results from the Shapiro-Wilk Normality Test tell us that the distribution of `sum` is statistically found to differ from the normal distribution. So in this case, statistical significance suggests that `sum` cannot be used as a parametric dependent variable. For our aims this is all the evaluation required. Effect size and confidence intervals are not applicable.

It is of note, however, that the expectation that the variable `sum` would conform to the normal distribution was low from the outset as we are working with count data. Count data, or frequencies, are in a strict sense not continuous, but rather discrete –meaning that they are real numbers (whole numbers which are always positive). This is a common informational type to encounter in text analysis.

---

## 10.7 Bivariate analysis

A more common scenario in statistical analysis is the consideration of the relationship between two-variables, known as bivariate analysis.

### 10.7.1 Categorical

Let's build on our univariate analysis of `realization_of_recipient` and include an explanatory, or independent variable which we will explore to test whether it can explain our earlier finding that 'NP' datives are more common than 'PP' datives. The question to test, then, is whether modality explains the distribution of the `realization_of_recipient`.

#### Descriptive assessment

Both the `realization_of_recipient` and `modality` variables are categorical, specifically nominal as we can see by using `skim()`.

Table 10.6: Contingency table for `realization_of_recipient` and `modality`.

realization_of_recipient/modality	spoken	written	Total
NP	79% (1859)	61% (555)	74% (2414)
PP	21% (501)	39% (348)	26% (849)
Total	100% (2360)	100% (903)	100% (3263)

```
dative |>
  select(realization_of_recipient, modality) |> # select
  ↵ key variables
  skim() |> # get custom data summary
  yank("factor") # only show factor-oriented information
```

**Variable type: factor**

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
realization_of_recipient	0	1	FALSE	2	NP: 2414, PP: 849
modality	0	1	FALSE	2	spo: 2360, wri: 903

For this reason measures of central tendency are not applicable and we will turn to a contingency table to summarize the relationship. The `janitor` package has a set of functions, the primary function being `tabyl()`. Other functions used here are to adorn the contingency table with totals, percentages, and to format the output for readability.

```
dative |>
  tabyl(realization_of_recipient, modality) |> #
  ↵ cross-tabulate
  adorn_totals(c("row", "col")) |> # provide row and
  ↵ column totals
  adorn_percentages("col") |> # add percentages to the
  ↵ columns
  adorn_pct_formatting(rounding = "half up", digits = 0)
  ↵ |> # round the digits
  adorn_ns() |> # add observation number
  adorn_title("combined") |> # add a header title
  kable(booktabs = TRUE) # pretty table
```

To gain a better appreciation for this relationship let's generate a couple plots one which shows cross-tabulated counts and the other calculated proportions.

```

p1 <-
  dative |> # dataset
  ggplot(aes(x = realization_of_recipient, fill =
    ↵ modality)) + # mappings
  geom_bar() + # geometry
  labs(y = "Count", x = "Realization of recipient") #
    ↵ labels

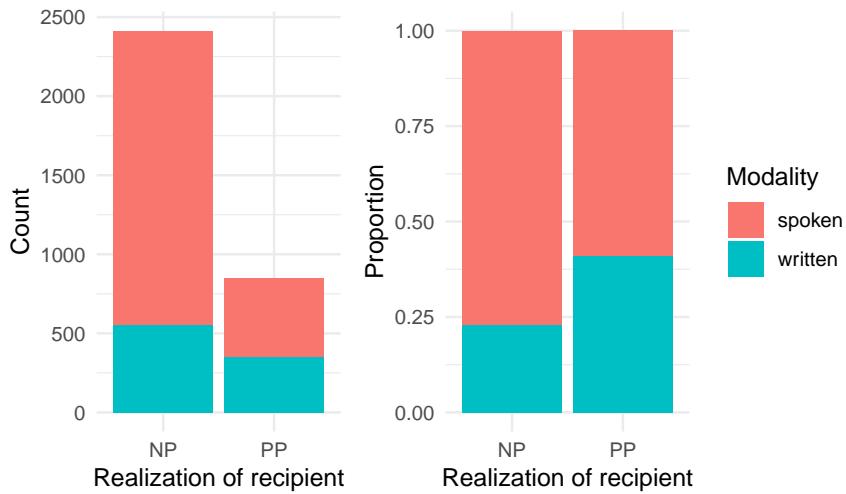
p2 <-
  dative |> # dataset
  ggplot(aes(x = realization_of_recipient, fill =
    ↵ modality)) + # mappings
  geom_bar(position = "fill") + # geometry, with fill for
    ↵ proportion plot
  labs(y = "Proportion", x = "Realization of recipient",
    ↵ fill = "Modality") # labels

p1 <- p1 + theme(legend.position = "none") # remove legend
  ↵ from left plot

p1 + p2 + plot_annotation("Relationship between
  ↵ Realization of recipient and Modality.")

```

Relationship between Realization of recipient and Modality.



Looking at the count plot (in the left pane) we see that large difference between the realization of the dative as an 'NP' or 'PP' obscures to some degree our

ability to see to what degree modality is related to the realization of the dative. So, a proportion plot (in the right pane) standardizes each level of `realization_of_recipient` to provide a more comparable view. From the proportion plot we see that there appears to be a trend towards more use of ‘PP’ than ‘NP’ in the written modality.

### Statistical interrogation

Although the proportion plot is visually helpful, we use the raw counts to statistically analyze this relationship. Again, as we are working with categorical variables, now for a dependent and independent variable, we use the Chi-squared test. And as before we need to create the cross-tabulation table to pass to the `chisq.test()` to perform the test.

```
ror_mod_table <-  
  xtabs(formula = ~ realization_of_recipient + modality, #  
    ↪ formula  
    data = dative) # dataset  
  
c2 <- chisq.test(ror_mod_table) # apply the chi-squared  
  ↪ test to `ror_mod_table`  
  
c2 # # preview the test results  
  
#>  
#> Pearson's Chi-squared test with Yates' continuity correction  
#>  
#> data: ror_mod_table  
#> X-squared = 101, df = 1, p-value <2e-16  
  
c2$p.value < .05 # confirm p-value below .05  
  
#> [1] TRUE
```

We can preview the result and provide a confirmation of the *p*-value. This evidence suggests that there is a difference between the distribution of dative realization according to modality.

We can also see more details about the test.

```
c2 |> # statistical result  
  augment() # view detailed statistical test information  
  
#> # A tibble: 4 x 9  
#>   realization_of_~1 modal~2 .obse~3 .prop .row.~4 .col.~5 .expe~6 .resid .std.~7  
#>   <fct>           <fct>     <int>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
```

```
#> 1 NP           spoken     1859 0.570   0.770   0.788   1746.   2.71   10.1
#> 2 PP           spoken     501  0.154   0.590   0.212   614.  -4.56  -10.1
#> 3 NP           written    555  0.170   0.230   0.615   668.  -4.37  -10.1
#> 4 PP           written    348  0.107   0.410   0.385   235.   7.38   10.1
#> # ... with abbreviated variable names 1: realization_of_recipient, 2: modality,
#> # 3: .observed, 4: .row.prop, 5: .col.prop, 6: .expected, 7: .std.resid
```

### Evaluation

Now we want to calculate the effect size and the confidence interval to provide measures of assurance that our finding is robust.

```
effects <- effectsize(c2) # evaluate effect size and
                           ↪ generate a confidence interval

effects # preview effect size and confidence interval

#> Cramer's V (adj.) |      95% CI
#> -----
#> 0.18                | [0.15, 1.00]
#>
#> - One-sided CIs: upper bound fixed at [1.00].
```

```
interpret_r(effects$Cramers_v) # interpret the effect size

#> [1] "small"
#> (Rules: funder2019)
```

We get effect size and confidence interval information. Note that the effect size, reflected by Cramer's V, for this relationship is weak. This points out an important aspect to evaluation of statistical tests. The fact that a test is significant does not mean that it is meaningful. A small effect size suggests that we should be cautious about the extent to which this significant finding is robust in the population from which the data is sampled.

#### 10.7.2 Continuous

For a bivariate analysis in which the dependent variable is not categorical, we will turn to the `sdac_disfluencies` dataset. The question we will pose to test is whether the use of fillers is related to the type of filler ('uh' or 'um').

##### Descriptive assessment

The key variables to assess in this case are the variables `count` and `filler`. But before we start to explore this relationship we will need to transform the dataset such that each speaker's use of the levels of `filler` is summed. We

Table 10.7: First 10 observations from `sdac_fillers` dataset.

	speaker_id	filler	sum
155	uh	28	
155	um	0	
1000	uh	37	
1000	um	8	
1001	uh	262	
1001	um	2	
1002	uh	34	
1002	um	20	
1004	uh	30	
1004	um	15	

will use `group_by()` to group `speaker_id` and `filler` combinations and then use `summarize()` to then sum the counts for each filler type for each speaker

```
sdac_fillers <-
  sdac_disfluencies |> # dataset
  group_by(speaker_id, filler) |> # grouping parameters
  summarize(sum = sum(count)) |> # summed counts for each
    ↵   speaker-filler combination
  ungroup() # remove the grouping parameters
```

Let's preview this transformation.

Let's take a look at them together by grouping the dataset by `filler` and then using the custom `skim` function `num_skim()` for the numeric variable `count`.

```
sdac_fillers |> # dataset
  group_by(filler) |> # grouping parameter
  num_skim() |> # get custom data summary
  yank("numeric") # only show numeric-oriented information
```

#### Variable type: numeric

skim_variable	filler	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	iqr
sum	uh	0	1	71.4	91.5	0	14	39	91	661	77
sum	um	0	1	15.7	31.0	0	0	4	16	265	16

We see here that the standard deviation and IQR for both ‘uh’ and ‘um’ are relatively large for the respective means (71.4 and 15.7) suggesting the distribution is quite dispersed. Let's take a look at a boxplot to visualize the counts in `sum` for each level of `filler`.

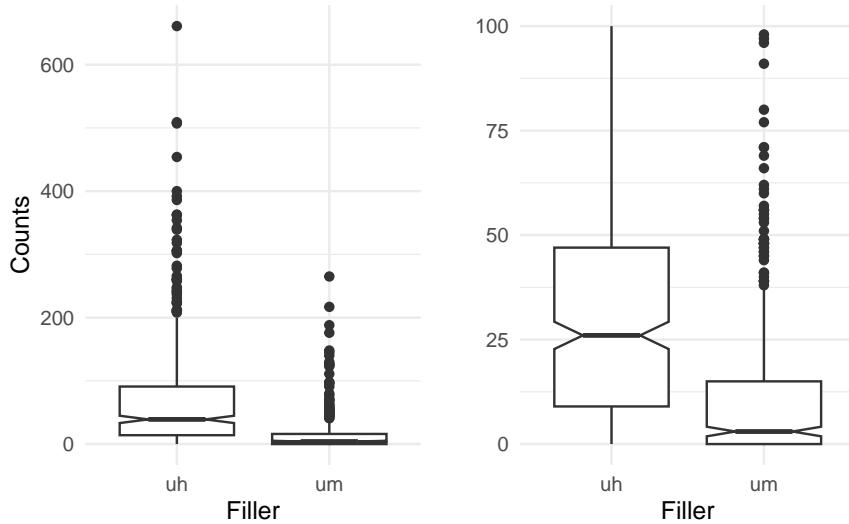
```

p1 <-
  sdac_fillers |> # dataset
  ggplot(aes(x = filler, y = sum)) + # mappings
  geom_boxplot(notch = TRUE) + # geometry
  labs(x = "Filler", y = "Counts") # labels

p2 <-
  sdac_fillers |> # dataset
  ggplot(aes(x = filler, y = sum)) + # mappings
  geom_boxplot(notch = TRUE) + # geometry
  ylim(0, 100) + # scale the y axis to trim outliers
  labs(x = "Filler", y = "") # labels

p1 + p2

```



In the plot in the left pane we see a couple things. First, it appears that there is in fact quite a bit of dispersion as there are quite a few outliers (dots) above the lines extending from the boxes. Recall that the boxes represent the first and third quartile, that is the IQR and that the notches represent the confidence interval. Second, when we compare the boxes and their notches we see that there is little overlap (looking horizontally). In the right pane I've zoomed in a bit trimming some outliers to get a better view of the relationship between the boxes. Since the overlap is minimal and in particular the notches do not overlap at all, this is a good indication that there is a significant trend.

From the descriptive statistics and the visual summary it appears that the

filler ‘uh’ is more common than ‘um’. It’s now time to submit this to statistical interrogation.

### Statistical interrogation

In a bivariate (and multivariate) analysis where the dependent variable is non-categorical we apply Linear Regression Modeling (LM). The default assumption of linear models, however, is that the dependent variable is normally distributed. As we have seen our variable `sum` does not conform to the normal distribution. We know this because of our tests in the univariate case, but as mentioned at the end of that section, we are working with count data which by nature is understood as discrete and not continuous in a strict technical sense. So instead of using the linear model for our regression analysis we will use the Generalized Linear Model (GLM) (R. Harald Baayen 2008; Gries 2013).

The function `glm()` implements generalized linear models. In addition to the formula (`sum ~ filler`) and the dataset to use, we also include an appropriate distribution family for the dependent variable. For count and frequency data the appropriate family is the “Poisson” distribution.

```
m1 <-  
  glm(formula = sum ~ filler, # formula  
       data = sdac_fillers, # dataset  
       family = "poisson") # distribution family  
  
summary(m1) # preview the test results  
  
#>  
#> Call:  
#> glm(formula = sum ~ filler, family = "poisson", data = sdac_fillers)  
#>  
#> Deviance Residuals:  
#>     Min      1Q  Median      3Q      Max  
#> -11.95   -5.61   -3.94    0.80   41.99  
#>  
#> Coefficients:  
#>             Estimate Std. Error z value Pr(>|z|)  
#> (Intercept)  4.26794   0.00564    757   <2e-16 ***  
#> fillerum     -1.51308   0.01327   -114   <2e-16 ***  
#> ---  
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
#>  
#> (Dispersion parameter for poisson family taken to be 1)  
#>  
#> Null deviance: 72049  on 881  degrees of freedom  
#> Residual deviance: 55071  on 880  degrees of freedom
```

```
#> AIC: 58524
#>
#> Number of Fisher Scoring iterations: 6
```

Let's focus on the coefficients, specifically for the 'fillerum' line. Since our factor `filler` has two levels one level is used as the reference to contrast with the other level. In this case by default the first level is used as the reference. Therefore the coefficients we see in 'fillerum' are 'um' in contrast to 'uh'. Without digging into the details of the other parameter statistics, let's focus on the last column which contains the *p*-value. A convenient aspect of the `summary()` function when applied to regression model results is that it provides statistical significance codes. In this case we can see that the contrast between 'uh' and 'um' is significant at  $p < .001$  which of course is lower than our standard threshold of .05.

Therefore we can say with some confidence that the filler 'uh' is more frequent than 'um'.

### Evaluation

Given we have found a significant effect for `filler`, let's look at evaluating the effect size and the confidence interval. Again, we use the `effectsize()` function. We then can preview the `effects` object. Note that effect size of interest is in the second row of the coefficient (`Std_Coefficient`) so we subset this column to extract only the effect coefficient for the `filler` contrast.

```
effects <- effectsize(m1) # evaluate effect size and
                           ↵ generate a confidence interval

effects # preview effect size and confidence interval

#> # Standardization method: refit
#>
#> Parameter | Std. Coef. |         95% CI
#> -----
#> (Intercept) |      4.27 | [ 4.26,   4.28]
#> fillerum    |     -1.51 | [-1.54, -1.49]
#>
#> - Response is unstandardized.

interpret_r(effects$Std_Coefficient[2]) # interpret the
                                         ↵ effect size

#> [1] "very large"
#> (Rules: funder2019)
```

The coefficient statistic falls within the confidence interval and the effect size is strong so we can be confident that our findings are reliable given this data.

---

## 10.8 Multivariate analysis

The last case to consider is when we have more than one independent variable we want to use to assess their potential relationship to the dependent variable. Again we will consider a categorical and non-categorical dependent variable. But, in this case the implementation methods are quite similar, as we will see.

### 10.8.1 Categorical

For the categorical multivariate case we will again consider the `dative` dataset and build on the previous analyses. The question to be posed is whether modality in combination with the length of the recipient (`length_of_recipient`) together explain the distribution of the realization of the recipient (`realization_of_recipient`).

#### Descriptive assessment

Now that we have three variables, there is more to summarize to get our descriptive information. Luckily, however, the same process can be applied to three (or more) variables using the `group_by()` function and then passed to `skim()`. In this case we have two categorical variables and one numeric variable. So we will group by both the categorical variables and then pass the numeric variable to the custom `num_skim()` function –pulling out only the relevant descriptive information for numeric variables with `yank()`.

```
dative |> # dataset
  select(realization_of_recipient, modality,
    ~ length_of_recipient) |> # select key variables
  group_by(realization_of_recipient, modality) |> #
    ~ grouping parameters
  num_skim() |> # get custom data summary
  yank("numeric") # only show numeric-oriented information
```

#### Variable type: numeric

skim_variable	realization_of_recipient	modality	n_missing	complete_rate	mean	sd	p0
length_of_recipient	NP	spoken	0	1	1.14	0.60	1
length_of_recipient	NP	written	0	1	1.95	1.59	1
length_of_recipient	PP	spoken	0	1	2.30	2.04	1
length_of_recipient	PP	written	0	1	4.75	4.10	1

There is much more information now that we are considering multiple independent variables, but if we look over the measures of dispersion we can see that the median and the IQR are relatively similar to their respective means suggesting that there are fewer outliers and relatively little skew.

Let's take a look at a visualization of this information. Since we are working with a categorical dependent variable and there is one non-categorical variable we can use a boxplot. The addition here is to include a `color` mapping which will provide a distinct box for each level of modality ('written' and 'spoken').

```
p1 <-
  dative |> # dataset
  ggplot(aes(x = realization_of_recipient, y =
    ~ length_of_recipient, color = modality)) + # mappings
  geom_boxplot(notch = TRUE) + # geometry
  labs(x = "Realization of recipient", y = "Length of
    recipient (in words)", color = "Modality") # labels

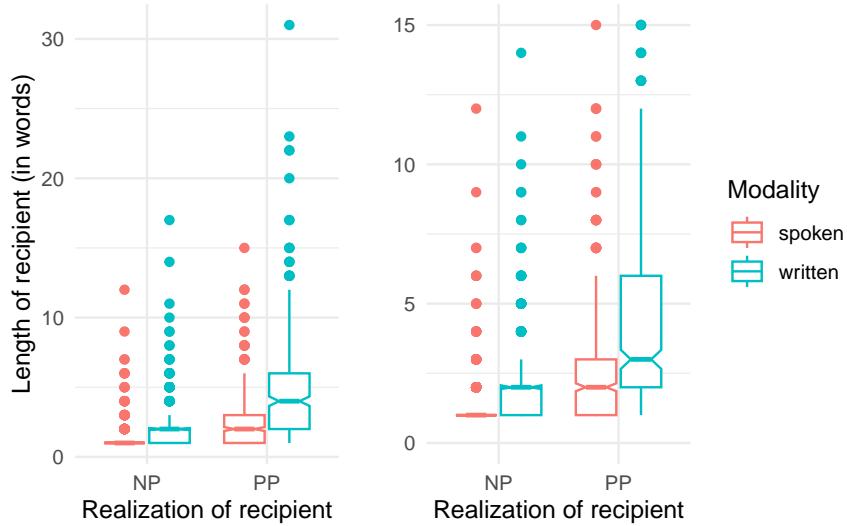
p2 <-
  dative |> # dataset
  ggplot(aes(x = realization_of_recipient, y =
    ~ length_of_recipient, color = modality)) + # mappings
  geom_boxplot(notch = TRUE) + # geometry
  ylim(0, 15) + # scale the y axis to trim outliers
  labs(x = "Realization of recipient", y = "", color =
    "Modality") # labels

p1 <- p1 + theme(legend.position = "none") # remove the
  ~ legend from the left pane plot

p1 + p2
```

In the left pane we see the entire visualization including all outliers. From this view it appears that there is a potential trend that the length of the recipient is larger when the realization of the recipient is 'PP'. There is also a potential trend for modality with written language showing longer recipient lengths overall. The pane on the right is scaled to get a better view of the boxes by scaling the y-axis down and as such trimming the outliers. This plot shows more clearly that the length of the recipient is longer when the recipient is realized as a 'PP'. Again, the contrast in modality is also a potential trend, but the boxes (of the same color), particularly for the spoken modality overlap to some degree.

So we have some trends in mind which will help us interpret the statistical interrogation so let's move there next.



### Statistical interrogation

Once we involve more than two variables, the choice of statistical method turns towards regression. In the case that the dependent variable is categorical, however, we will use Logistic Regression. The workhorse function `glm()` can be used for a series of regression models, including logistic regression. The requirement, however, is that we specify the family of the distribution. For logistic regression the family is "binomial". The formula includes the dependent variable as a function of our other two variables, each are separated by the + operator.

```
m1 <- glm(formula = realization_of_recipient ~ modality +
  ↪   length_of_recipient, # formula
  ↪   data = dative, # dataset
  ↪   family = "binomial") # distribution family

summary(m1) # preview the test results

#>
#> Call:
#> glm(formula = realization_of_recipient ~ modality + length_of_recipient,
#>       family = "binomial", data = dative)
#>
#> Deviance Residuals:
#>      Min        1Q    Median        3Q       Max
#>
```

```
#> -4.393 -0.598 -0.598 0.132 1.924
#>
#> Coefficients:
#>                               Estimate Std. Error z value Pr(>|z|)
#> (Intercept)              -2.3392    0.0797 -29.35   <2e-16 ***
#> modalitywritten       -0.0483    0.1069   -0.45     0.65
#> length_of_recipient    0.7081    0.0420   16.86   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#> Null deviance: 3741.1  on 3262  degrees of freedom
#> Residual deviance: 3104.7  on 3260  degrees of freedom
#> AIC: 3111
#>
#> Number of Fisher Scoring iterations: 5
```

The results from the model again provide a wealth of information. But the key information to focus on is the coefficients. In particular the coefficients for the independent variables `modality` and `length_of_recipient`. What we notice, is that the *p*-value for `length_of_recipient` is significant, but the contrast between ‘written’ and ‘spoken’ for `modality` is not. If you recall, we used this same dataset to explore `modality` as a single independent variable earlier –and it was found to be significant. So why now is it not? The answer is that when multiple variables are used to explain the distribution of a measure (dependent variable) each variable now adds more information to explain the dependent variable –each has its own contribution. Since `length_of_recipient` is significant, this suggests that the explanatory power of `modality` is weak, especially when compared to `length_of_recipient`. This make sense as we saw in the earlier model the fact that the effect size for `modality` was not strong and that is now more evident that the `length_of_recipient` is included in the model.

### Evaluation

Now let’s move on and gauge the effect size and calculate the confidence interval for `length_of_recipient` in our model. We apply the `effectsize()` function to the model and then use `interpret_r()` on the coefficient of interest (which is in the fourth row of the `Std_Coefficients` column).

```
effects <- effectsize(m1) # evaluate effect size and
                           ↳ generate a confidence interval

effects # preview effect size and confidence interval
```

```
#> # Standardization method: refit
#>
#> Parameter | Std. Coef. | 95% CI
#> -----
#> (Intercept) | -1.03 | [-1.15, -0.92]
#> modalitywritten | -0.05 | [-0.26, 0.16]
#> length_of_recipient | 1.46 | [1.30, 1.64]
#>
#> - Response is unstandardized.

    interpret_r(effects$Std_Coefficient[4]) # interpret the
    ↳ effect size

#> [1] NA
#> (Rules: funder2019)
```

We see we have a coefficient that falls within the confidence interval and the effect size is large. So we can saw with some confidence that the length of the recipient is a significant predictor of the use of ‘PP’ as the realization of the recipient in the dative alternation.

### 10.8.2 Continuous

The last case we will consider here is when the dependent variable is non-categorical and we have more than one independent variable. The question we will pose is whether the type of filler and the sex of the speaker can explain the use of fillers in conversational speech.

We will need to prepare the data before we get started as our current data frame `sdac_fillers` has filler and the sum count for each filler grouped by speaker—but it does not include the `sex` of each speaker. The `sdac_disfluencies` data frame does have the `sex` column, but it has not been grouped by speaker. So let’s transform the `sdac_disfluencies` summarizing it to only get the `speaker_id` and `sex` combinations. This should result in a data frame with 441 observations, one observation for each speaker in the corpus.

```
sdac_speakers_sex <-
  sdac_disfluencies |> # dataset
  distinct(speaker_id, sex) # summarize for distinct
  ↳ `speaker_id` and `sex` values
```

Let’s preview the first 10 observations form this transformation.

Great, now we have each `speaker_id` and `sex` for all 441 speakers. One thing

Table 10.8: First 10 observations of the `sdac_speakers_sex` data frame.

speaker_id	sex
155	NA
1000	FEMALE
1001	MALE
1002	FEMALE
1004	FEMALE
1005	FEMALE
1007	FEMALE
1008	FEMALE
1010	MALE
1011	FEMALE

to note, however, is that speaker ‘155’ does not have a value for `sex` –this seems to be an error in the metadata that we will need to deal with before we proceed in our analysis. Let’s move on to join our new `sdac_speakers_sex` data frame and the `sdac_fillers` data frame.

Now that we have a complete dataset with `speaker_id` and `sex` we will now join this dataset with our `sdac_fillers` dataset effectively adding the column `sex`. We want to keep all the observations in `sdac_fillers` and add the column `sex` for observations that correspond between each data frame for the column `speaker_id` so we will use a left join with the function `left_join()` with the `sdac_fillers` dataset on the left.

```
sdac_fillers_sex <-
  left_join(sdac_fillers, sdac_speakers_sex) # join
```

Now let’s preview the first observations in this new `sdac_fillers_sex` data frame.

At this point let’s drop this speaker from the `sdac_speakers_sex` data frame.

```
sdac_fillers_sex <-
  sdac_fillers_sex |> # dataset
  filter(speaker_id != "155") # drop speaker_id 155
```

We are now ready to proceed in our analysis.

### Descriptive assessment

The process by now should be quite routine for getting our descriptive statis-

Table 10.9: First 10 observations of the `sdac_fillers_sex` data frame.

speaker_id	filler	sum	sex
155	uh	28	NA
155	um	0	NA
1000	uh	37	FEMALE
1000	um	8	FEMALE
1001	uh	262	MALE
1001	um	2	MALE
1002	uh	34	FEMALE
1002	um	20	FEMALE
1004	uh	30	FEMALE
1004	um	15	FEMALE

tics: select the key variables, group by the categorical variables, and finally pull the descriptives for the numeric variable.

```
sdac_fillers_sex |> # dataset
  select(sum, filler, sex) |> # select key variables
  group_by(filler, sex) |> # grouping parameters
  num_skim() |> # get custom data summary
  yank("numeric") # only show numeric-oriented information
```

#### Variable type: numeric

skim_variable	filler	sex	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	q4
sum	uh	FEMALE	0	1	63.22	76.5	0	12.0	39.0	81.8	100.0
sum	uh	MALE	0	1	78.74	102.6	0	15.2	37.5	101.5	100.0
sum	um	FEMALE	0	1	22.38	36.3	0	1.0	9.0	28.0	39.0
sum	um	MALE	0	1	9.92	24.2	0	0.0	1.0	8.0	10.0

Looking at these descriptives, it seems like there is quite a bit of variability for some combinations and not others. In short, it's a mixed bag. Let's try to make sense of these numbers with a boxplot.

```
p1 <-
  sdac_fillers_sex |> # dataset
  ggplot(aes(x = filler, y = sum, color = sex)) + #
    # mappings
  geom_boxplot(notch = TRUE) + # geometry
  labs(x = "Filler", y = "Counts", color = "Sex") # labels
```

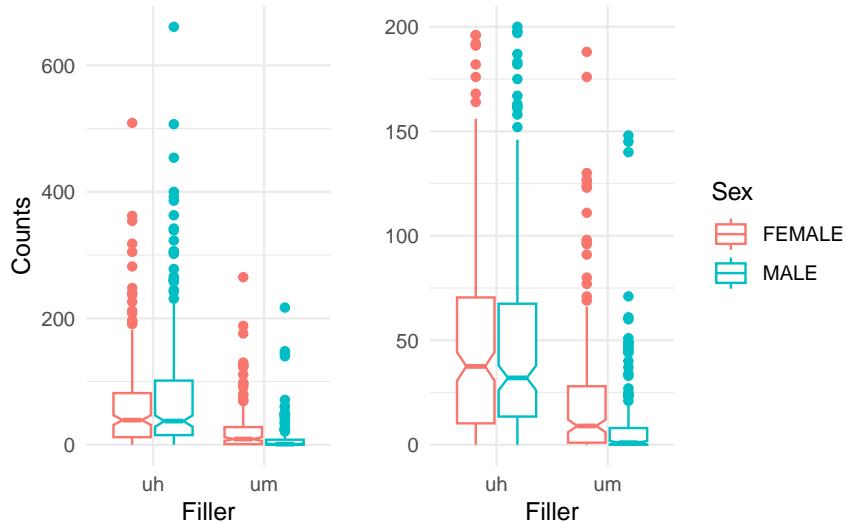
```

p2 <-
  sdac_fillers_sex |> # dataset
  ggplot(aes(x = filler, y = sum, color = sex)) + #
    ↵ mappings
  geom_boxplot(notch = TRUE) + # geometry
  ylim(0, 200) + # scale the y axis to trim outliers
  labs(x = "Filler", y = "", color = "Sex") # labels

p1 <- p1 + theme(legend.position = "none") # drop the
    ↵ legend from the left pane plot

p1 + p2

```



We can see that ‘uh’ is used more than ‘um’ overall. But that whereas men and women use ‘uh’ in similar ways, women use more ‘um’ than men. This is known as an interaction. So we will approach our statistical analysis with this in mind.

### Statistical interrogation

We will again use a generalized linear model with the `glm()` function to conduct our test. The distribution family will be the same has we are again using the `sum` as our dependent variable which contains discrete count values. The formula we will use, however, is new. Instead of adding a new variable to our independent variables, we will test the possible interaction between `filler` and `sex` that we noted in the descriptive assessment. To encode an interac-

tion the `*` operator is used. So our formula will take the form `sum ~ filler * sex`. Let's generate the model and view the summary of the test results as we have done before.

```
m1 <-
  glm(formula = sum ~ filler * sex, # formula
      data = sdac_fillers_sex, # dataset
      family = "poisson") # distribution family

summary(m1) # preview the test results

#>
#> Call:
#> glm(formula = sum ~ filler * sex, family = "poisson", data = sdac_fillers_sex)
#>
#> Deviance Residuals:
#>    Min      1Q  Median      3Q     Max
#> -12.55   -6.21   -3.64    1.08   40.60
#>
#> Coefficients:
#>             Estimate Std. Error z value Pr(>|z|)
#> (Intercept)  4.14660   0.00876  473.2  <2e-16 ***
#> fillerum     -1.03827   0.01714   -60.6  <2e-16 ***
#> sexMALE      0.21955   0.01145    19.2  <2e-16 ***
#> fillerum:sexMALE -1.03344   0.02791   -37.0  <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for poisson family taken to be 1)
#>
#> Null deviance: 71956 on 879 degrees of freedom
#> Residual deviance: 53543 on 876 degrees of freedom
#> AIC: 56994
#>
#> Number of Fisher Scoring iterations: 6
```

Again looking at the coefficients we see something new. First we see that there is a row for the `filler` contrast and the `sex` contrast but also the interaction between `filler` and `sex` ('`fillerum:sexMALE`'). All rows show significant effects. It is important to note that when an interaction is explored and it is found to be significant, the other simple effects, known as main effects ('`fillerum`' and '`sexMALE`'), are ignored. Only the higher-order effect is considered significant.

Now what does the '`fillerum:sexMALE`' row mean. It means that there is an interaction between `filler` and `sex`. the directionality of that interaction

should be interpreted using our descriptive assessment, in particular the visual boxplots we generated. In sum, women use more ‘um’ than men or stated another way men use ‘um’ less than women.

### Evaluation

We finalize our analysis by looking at the effect size and confidence intervals.

```

effects <- effectsize(m1) # evaluate effect size and
                           ↳ generate a confidence interval

effects # preview effect size and confidence interval

#> # Standardization method: refit
#>
#> Parameter      | Std. Coef. |         95% CI
#> -----
#> (Intercept)    |      4.15 | [ 4.13,  4.16]
#> fillerum       |     -1.04 | [-1.07, -1.00]
#> sexMALE        |      0.22 | [ 0.20,  0.24]
#> fillerum:sexMALE |     -1.03 | [-1.09, -0.98]
#>
#> - Response is unstandardized.

interpret_r(effects$Std_Coefficient[4]) # interpret the
                                           ↳ effect size

#> [1] "very large"
#> (Rules: funder2019)

```

We can conclude, then, that there is a strong interaction effect for `filler` and `sex` and that women use more ‘um’ than men.

## 10.9 Summary

In this chapter we have discussed various approaches to conducting inferential data analysis. Each configuration, however, always includes a descriptive assessment, statistical interrogation, and an evaluation of the results. We considered univariate, bivariate, and multivariate analyses using both categorical and non-categorical dependent variables to explore the similarities and differences between these approaches.

— | — | —

## 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

## *Reporting*

..quote..

— Author

### Keys

- ...

In this chapter, we first examine the role of research presentations in academic and professional settings, highlighting the benefits of presenting one's work as a means to develop and refine ideas and conclusions. We then discuss the purpose of a research document, focusing on how its structure effectively conveys the project's rationale, goals, procedures, results, and findings. Additionally, we explore the integration of citations, references, figures, and tables in research documents using Quarto. Lastly, we investigate the application of field-specific and publishing house formats to a variety of document formats, such as Word, PDF, HTML, and ePub.

### Swirl

**What:** Rendering Quarto<sup>a</sup>

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

**Why:** To ...

<sup>a</sup><https://github.com/lin380/swirl>

### 1. Introduction

- Importance of clear and effective communication in research
  - Enhances understanding of the research
  - Facilitates collaboration and interdisciplinary work
- Overview of the chapter content
  - Research presentations
  - Research document structure
  - Citations and references
  - Figures and tables

- Field-specific and publishing house formats
- 2. Research Presentations
  - Importance of research presentations in academia and professional contexts
    - Sharing research findings
    - Building a professional reputation
  - Benefits of presenting research work
    - Developing and refining ideas and conclusions through feedback and discussion
    - Receiving feedback and suggestions from peers and experts
      - \*Conducting peer review can help ensure that the research is conducted appropriately and transparently. Peer review can also identify potential errors or biases in the research design or analysis, and can provide suggestions for improving the study's reproducibility
    - Enhancing communication and presentation skills
    - Building professional network and fostering collaboration
  - Key elements of an effective research presentation
    - Structure and organization: logical flow, clear sections, concise points
    - Visual design and aesthetics: use of color, fonts, visuals, and layout
    - Delivery and engagement: pacing, tone, body language, and audience interaction
  - Using R for creating research presentations
    - RMarkdown and its integration with presentation formats (e.g., xaringan, slidify)
    - Incorporating data visualization, tables, and other R-generated content seamlessly
  - Tips for delivering engaging and memorable presentations
    - Practice and preparation
    - Storytelling and relatability
    - Addressing diverse audience backgrounds
- 3. Research Documents
  - Purpose of a research document
    - Communicate research methodology and findings
    - Contribute to the body of knowledge
  - Benefits of writing a research document
    - Clarifying and organizing thoughts and ideas
    - Providing a comprehensive record of research work
    - Enhancing writing and argumentation skills
    - Contributing to the body of knowledge in a field
    - Establishing professional credibility and visibility
  - Components of a research document
    - Title: concise and informative

- Abstract: brief summary of the research
  - Introduction: background, research question, and objectives
  - Methodology: description of the data, tools, and analysis techniques
  - Results: presentation of the findings
  - Discussion: interpretation and implications of the results
  - Conclusion: summary and future directions
  - References: list of cited sources
  - Tips for organizing and structuring research documents
    - Outlining the main sections
    - Maintaining logical flow and consistency
    - Using clear and concise language
4. Citations and References
- Importance of proper citation and referencing
    - Acknowledge the work of others
    - Demonstrate the foundation of your research
  - Different citation styles (APA, MLA, Chicago, etc.)
    - Overview of common styles
    - Choosing the appropriate style for your discipline
  - Using R and Quarto for managing citations and references
    - Integration with reference managers (e.g., Zotero, Mendeley)
    - Automating citation formatting
  - Integration of citations and references in research documents
    - In-text citations
    - Reference list or bibliography
5. Figures and Tables
- Importance of visual representations in research reporting
    - Aid in understanding complex data
    - Enhance the readability of the document
  - Creating and customizing figures and tables using R and Quarto
    - ggplot2 for creating visualizations
    - kable and gt for generating tables
  - Guidelines for labeling, formatting, and presenting figures and tables
    - Clear and informative titles and labels
    - Consistent formatting
    - Appropriate use of color, fonts, and layout
    - Proper referencing and integration in the text
    - Accessibility considerations for diverse readers
6. Field-specific and Publishing House Formats
- Importance of adhering to field-specific and publisher guidelines

- Consistency and professionalism
  - Meeting submission requirements for journals, conferences, and other venues
  - Overview of common format requirements
    - Manuscript formatting: margins, line spacing, headings, etc.
    - Citation and reference style
    - Figure and table formatting
  - Using R and Quarto to apply formatting guidelines
    - Customizing document templates
    - Applying and managing style files
  - Exporting research documents to various formats
    - Word: .docx files for collaboration and editing
    - PDF: .pdf files for sharing and printing
    - HTML: web-based documents for online publication
    - ePub: e-book format for digital reading
7. Conclusion
- Recap of the main topics covered in the chapter
  - Importance of clear and effective communication in research reporting
  - Encouragement to practice and refine reporting skills through various research projects

---

## Questions

**i** Conceptual questions

1. ...
2. ...

**i** Technical exercises

1. ...
2. ...

# 12

---

## *Collaboration*

---

..quote..

— Author

### Keys

- ...

Whether for other researchers or for your future self, creating research that is well-documented and reproducible is a fundamental part of conducting modern scientific inquiry. In this chapter we will emphasize the importance of this endeavor and outline strategies for ensuring your research project is reproducible. This will include directory and file structure, key documentation files as well as how to effectively use existing software resources and frameworks for publishing your research (either for private use, journal requirements, or general public consumption) on popular repositories such as GitHub and Open Science Framework (OSF).

### Swirl

**What:** Compiling Research Projects<sup>a</sup>

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

**Why:** To ...

---

<sup>a</sup><https://github.com/lin380/swirl>

Outline...

This is what we will cover in this chapter: ...

## Questions

### Conceptual questions

1. ...
2. ...

### Technical exercises

1. ...
2. ...

---

## 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.
- Allaire, JJ. 2022. *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://CRAN.R-project.org/package=rmarkdown>.
- Almeida, Tiago A, José María Gómez Hildago, and Akebo Yamakami. 2011. “Contributions to the Study of SMS Spam Filtering: New Collection and Results.” In *Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG’11)*, 4. Mountain View, CA.
- Baayen, R. Harald. 2004. “Statistics in Psycholinguistics: A Critique of Some Current Gold Standards.” *Mental Lexicon Working Papers* 1 (1): 1–47.
- . 2008. *Analyzing Linguistic Data: A Practical Introduction to Statistics Using r*. Cambridge Univ Pr.
- . 2011. “Corpus Linguistics and Naive Discriminative Learning.” *Revista Brasileira de Linguística Aplicada* 11 (2): 295–328.
- Baayen, R. H., and Elnaz Shafaei-Bajestan. 2019. *languageR: Analyzing Linguistic Data: A Practical Introduction to Statistics*. <https://CRAN.R-project.org/package=languageR>.
- 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>.
- Benoit, Kenneth. 2020. *Quanteda.corpora: A Collection of Corpora for Quanteda*. [http://github.com/quanteda/quanteda.corpora](https://github.com/quanteda/quanteda.corpora).
- Broman, Karl W., and Kara H. Woo. 2018. “Data Organization in Spreadsheets.” *The American Statistician* 72 (1): 2–10. <https://doi.org/10.1080/00031305.2017.1375989>.
- 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>.
- 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*.
- Gandrud, Christopher. 2015. *Reproducible Research with r and r Studio*<sup>1</sup>. Second edition. CRC Press.

<sup>1</sup><https://www.ncbi.nlm.nih.gov/pubmed/17811671>

- 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. 2013. *Statistics for Linguistics with r. A Practical Introduction*. 2nd revise.
- Grieve, Jack, Andrea Nini, and Diansheng Guo. 2018. “Mapping Lexical Innovation on American Social Media.” *Journal of English Linguistics* 46 (4): 293–319.
- Head, Megan L., Luke Holman, Rob Lanfear, Andrew T. Kahn, and Michael D. Jennions. 2015. “The Extent and Consequences of p-Hacking in Science.” *PLOS Biology* 13 (3): e1002106. <https://doi.org/10.1371/journal.pbio.1002106>.
- 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>.
- “How to Make a Data Dictionary.” 2021. *OSF Guides*. <https://help.osf.io/hc/en-us/articles/360019739054-How-to-Make-a-Data-Dictionary>.
- Hu, Minqing, and Bing Liu. 2004. “Mining and Summarizing Customer Reviews.” In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 168–77.
- 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).
- Jurafsky, Daniel, and James H. Martin. 2020. *Speech and Language Processing*.
- Kearney, Michael W., Lluís Revilla Sancho, and Hadley Wickham. 2023. *Rtweet: Collecting Twitter Data*. <https://CRAN.R-project.org/package=rtweet>.
- Kerr, Norbert L. 1998. “HARKing: Hypothesizing After the Results Are Known.” *Personality and Social Psychology Review* 2 (3): 196–217.
- Kloumann, IM, CM Danforth, KD Harris, and CA Bliss. 2012. “Positivity of the English Language.” *PloS One*.

- Kowsari, Kamran, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendum, Laura E. Barnes, and Donald E. Brown. 2019. "Text Classification Algorithms: A Survey." *Information* 10 (4): 150. <https://doi.org/10.3390/info10040150>.
- 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.
- Lantz, Brett. 2013. *Machine Learning with r*. Birmingham: Packt Publishing.
- Lewis, Michael. 2004. *Moneyball: The Art of Winning an Unfair Game*. WW Norton & Company.
- 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.
- Marwick, Ben, Carl Boettiger, and Lincoln Mullen. 2018. "Packaging Data Analytical Work Reproducibly Using r (and Friends)." *The American Statistician* 72 (1): 80–88.
- 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>.
- Olohan, Maeve. 2008. "Leave It Out! Using a Comparable Corpus to Investigate Aspects of Explicitation in Translation." *Cadernos de Tradução*, 153–69.
- Paquot, Magali, and Stefan Th. Gries, eds. 2020. *A Practical Handbook of Corpus Linguistics*. Switzerland: Springer.
- Roediger, H. L. L., and K. B. B McDermott. 2000. "Distortions of Memory." *The Oxford Handbook of Memory*, 149–62.
- 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.
- 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>.
- 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.

- Welbers, Kasper, and Wouter van Atteveldt. 2022. *Rsyntax: Extract Semantic Relations from Text by Querying and Reshaping Syntax*. <https://CRAN.R-project.org/package=rsyntax>.
- Wickham, Hadley. 2014. “Tidy Data.” *Journal of Statistical Software* 59 (10). <https://doi.org/10.18637/jss.v059.i10>.
- . 2022. *Rvest: Easily Harvest (Scrape) Web Pages*. <https://CRAN.R-project.org/package=rvest>.
- . 2023. *Tidyverse: Easily Install and Load the Tidyverse*. <https://CRAN.R-project.org/package=tidyverse>.
- Wickham, Hadley, Jennifer Bryan, Malcolm Barrett, and Andy Teucher. 2023. *Usethis: Automate Package and Project Setup*. <https://CRAN.R-project.org/package=usethis>.
- Wickham, Hadley, Jim Hester, Winston Chang, and Jennifer Bryan. 2022. *Devtools: Tools to Make Developing r Packages Easier*. <https://CRAN.R-project.org/package=devtools>.
- 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://CRAN.R-project.org/package=bookdown>.
- . 2023b. *Tinytex: Helper Functions to Install and Maintain TeX Live, and Compile LaTeX Documents*. <https://github.com/rstudio/tinytex>.



A

---

*Data*

---

...



---

---

## ***Index***

---

data science, 38  
dataset, 72  
Experimental data, 41  
Observational data, 41  
population, 58  
Reproducible research, 39  
research question, 133  
sample, 59  
sample size, 59  
Text analysis, 44