Computer Specifications and Setup

Before beginning any analysis, we can make sure the computing environment is ready for the tasks ahead. This chapter introduces the basic setup required to follow the examples in the book, including installing essential R packages and managing the computational resources needed to work with large-scale textual datasets.

The setup and tools introduced here reflect one of the central challenges of the book itself. Our aim is to teach how to perform text mining at scale, an approach that, at the time of writing, remains inaccessible and largely unwritten about in the practice of digital history. We hope to prepare readers with the knowledge to handle the practical constraints involved in working with large historical corpora.

Because historical datasets vary widely in size—and because readers will be working on many different types of computers or have access to vastly different computational resources—this chapter also offers guidance for managing limited memory or processing power. When necessary, readers can load smaller samples of the data to practice the methods. These sampled outputs may differ from the full visualizations and results presented in the book, but they will still allow readers to follow the workflow and understand the techniques.

Installing Packages

The purpose of a package is to add new tools and functions to R.

Many packages can be installed directly from the Comprehensive R Archive Network (CRAN), a central repository where R packages are reviewed, stored, and distributed. To use a function from another package, we must first install the package with `install.packages()`, and then load it with `library()`. We install the packages here so the libraries load smoothly later, without extra setup or interruptions.

If RStudio prompts you to restart R while installing a package, click "Yes" to allow the restart, and then re-run the installation command so the package loads into the fresh session.

Packages only have to be installed once, but they need to be reloaded for each new R session using `library()`.

First we can install our primary data science packages that serve as the core of our data processing approach.

```r
# For core data science and visualization packages, central to this book
install.packages("tidyverse")

# For tokenizing, cleaning, and computing tf-idf
install.packages("tidytext")

# For high-performance tools to work with large data tables
install.packages("data.table")

# For a comprehensive text analysis toolkit
# (e.g., keywords-in-context, tokenization, document-feature matrices)
install.packages("quanteda")

# For handling and manipulating dates and times
# (useful for organizing historical data chronologically)
install.packages("lubridate")
```

```r
# For stemming and lemmatization
# (reducing words to their dictionary or root forms for cleaner analysis)
install.packages("textstem")
```

We can also install packages that give us tools for creating more visualizations:

```r
# For creating word clouds from text data
install.packages("wordcloud")

# For formatting numbers, colors, and axes in visualizations
install.packages("scales")

# For ridge plots (density plots stacked by category)
install.packages("ggridges")

# For non-overlapping text labels in ggplot2
install.packages("ggrepel")

# For accessible and colorblind-friendly palettes for visualizations
install.packages("viridis")
```

Packages for document formatting and communicating data in tables:

```r
# The core engine for rendering R Markdown documents
install.packages("knitr")

# For creating elegantly formatted tables in HTML and PDF
install.packages("kableExtra")

# For arranging multiple tables, plots, or graphics on a page
install.packages("gridExtra")
```

Packages for webscraping and data extraction:

```r
# Tools for scraping data from websites
install.packages("rvest")

# Tools for working with and parsing XML or HTML documents
install.packages("xml2")

# Install tools for accessing and working with U.S. Department of Justice datasets
install.packages("usdoj")
```

While many packages can be installed from CRAN, many more specialty, research-oriented packages—such as those used in this book—are hosted on GitHub instead. Packages hosted exclusively on GitHub may be there because they change rapidly, support specialized workflows, or, as in our case, include larger datasets that CRAN is not designed to support.

When a package is hosted on GitHub, it can be installed in several ways. Fpr this book, we try to make the installation of our packages as easy as possible while working on as many computers as possible. To do this we provide special installation scripts that automate downloading and installing the required files. These scripts are accessed using `source()`, which tells R to fetch and run the installation code directly from the provided URL. For example:

```r
source("https://raw.githubusercontent.com/Democracy-Lab/hansardr/master/tools/install_hansardr.R")
library(hansardr)

source("https://raw.githubusercontent.com/Democracy-Lab/tmha.data/main/tools/install_tmha.data.R")
library(tmha.data)
```

In RStudio, you can verify whether a package is already installed by checking the Packages tab in the lower-right panel. This tab displays all packages currently available in your R library. If a package appears in that list, it is already installed; if not, you can install it using `install.packages()`. You can also load an installed package by checking the box next to its name in this tab, which is equivalent to calling `library()` in your script.

```r
install.packages("text2vec")  # Install text2vec
```

```r
library("httr")
library("fs")
library("pdftools")
```

Loading Tools

Once we have installed a package, we can test the installation of any of these packages using the `library()` function like so:

```r
library(hansardr)
```

If the code returns an error such as `Error in library(hansardr) : there is no package called 'hansardr'`, it means the package has not been installed. If no error appears, the package has loaded successfully and its functions are ready to use.

Understanding Package Versions

Sometimes package version issues can cause code to fail. If a package is outdated, it may not behave as we expect. We can check the version of any installed package with:

```r
packageVersion("hansardr")
```

```
## [1] '1.0.0'
```

Just like we installed a package, we can also update a package using `install.packages()`:

```r
install.packages("tidytext")
```

Or we can update all of our packages at once. This step might take awhile to complete depending on the number of packages we have installed.

```r
update.packages()
```

Handling Resources

Readers will be working on many different kinds of computers, some with more or less memory and processing power. Large historical datasets can sometimes cause R to slow down or crash, especially on older computers. This section outlines a few strategies for managing resources.

To help ensure accessibility, all of the code in this book has been tested in multiple classroom settings. We demonstrate workflows that will run on a MacBook with 8 GB of RAM and roughly 10 GB of free hard-drive space. The examples have been designed to run within these constraints.

However, if you encounter additional issues—such as code running slowly, long wait times, or working on a machine with limited resources—we provide several strategies for adapting the workflows in this book to smaller or less powerful computers.

One simple approach to resource management is to remove objects from memory when they are no longer needed. To demonstrate this, we load the 1820s from the `hansardr` library.

```r
# This will give us access to the Hansard corpus
library(hansardr)

# Load just the debates from the 1820s
data("hansard_1820")
```

We can use `rm()` to remove dataframes we no longer need. While `rm()` removes the data from R, it does not clear the amount of computer memory the data is consuming. Instead, we can follow wih `gc()` to clear space and reclaim memory that can be used again for other computations.

```r
# Remove the debates from R
rm(hansard_1820)

# Reclaim computer memory
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  587308 31.4    1316790 70.4   707688 37.8
## Vcells 1119983  8.6    8388608 64.0  1974117 15.1
```

Another strategy is to work with a sample of the data rather than the full dataset. This allows you to practice the methods without loading everything into memory. For example, you might limit the dataset to a smaller set of speakers or select only a few years from the larger decade-long corpus. In the following code, we combine the Hansard debates with their associated metadata—specifically the dates on which each speech was given—so that we can filter the corpus to include only a few of the years we want to work with.

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.6
## v forcats   1.0.1     v stringr   1.6.0
## v ggplot2   4.0.1     v tibble    3.3.0
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.1.0
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
# Load just the debates from the 1820s
data("hansard_1820")

data("debate_metadata_1820")

merged_hansard_data <- hansard_1820 %>%
  left_join(debate_metadata_1820, by = "sentence_id")

# Keep only the years 1820-1822 instead of the full decade
hansard_sample <- merged_hansard_data %>%
  filter(speechdate >= as.Date("1820-01-01"),
         speechdate <= as.Date("1822-12-31"))
```

While taking a random sample of a data frame can improve performance, random sampling is generally discouraged because it may produce a distorted or incomplete representation of the historical record. In learning situations where the dataset cannot be loaded due to limited computational resources, a random sample can be used as a temporary workaround. In such cases, the resulting outputs may not be historically representative and may differ from the and visualizations presented in the book.

```r
# Randomly sample and keep only 10% of the dataframe
hansard_sample <- hansard_1850 %>%
  sample_frac(0.10)
```

Finalizing Setup

With these tools installed and your environment ready, you now have everything you need to begin exploring large-scale historical text data. Every computer has limitations and each chapter will provide strategies for working effectively with the Hansard data regardless of hardware. In the following chapters, we turn from setup to analysis, beginning our journey into the methods of text mining at scale.