Appendix B

Appendix B

File Paths

Throughout this book, we have primarily worked with data imported directly from the `hansardr` library, which conveniently provides structured debate data through a single command. However, in most digital humanities research, data rarely arrives in a pre-packed form. More often, it exists as files stored on our computer's hard drive or in organized project directories. To access this data, we need to know about file paths. One of the primary reasons to use file paths is to access and read data files, such as those saved in the `.csv` (comma-separated values) or `.json` (JavaScript Object Notation) format, to name just two popular examples.

Every dataset, transcript, or image used in research is stored somewhere on your computer and can only be accessed in a programming language by specifying its file path. We dedicated this appendix chapter to file paths because, although the concept is fundamental, it is also one of the most common sources of confusion for analysts with data.

On the surface, file paths seem simple, especially if you can see the file sitting on your Desktop or in your Home folder. However, they are conceptually difficult because they rely on multiple layers of abstraction that form an invisible structural path within your computer's file system.

To bridge this concept, we liken file paths to movement through a physical archive. To understand what we mean, imagine you are standing inside a vast archive. The building contains many rooms; within each room are rows of shelves, and on those shelves sit labeled boxes. To find a single document, one must know the exact route through this hierarchy — the room, the shelf, and the box — just as one needs a file path on a computer to locate a specific file within its folders.

A letter stored in Room 3, inside Box 27, within Folder 5, could be represented as a file path like this:

```
/archive/room3/box27/folder5/letter.txt
```

If file paths have posed challenges, that experience is entirely valid—there are many good reasons why they can be challenging. Here are just a few more:

Challenge 1. Computers Require Precise File Paths

Computers handle file paths "rigidly," meaning that they expect exact spelling or notations. A single misplaced character can prevent the system from locating the correct file. Humans, by contrast, tend to think of files more loosely–based on where things feel like they are located–using a conceptual model rather than an exact digital address.

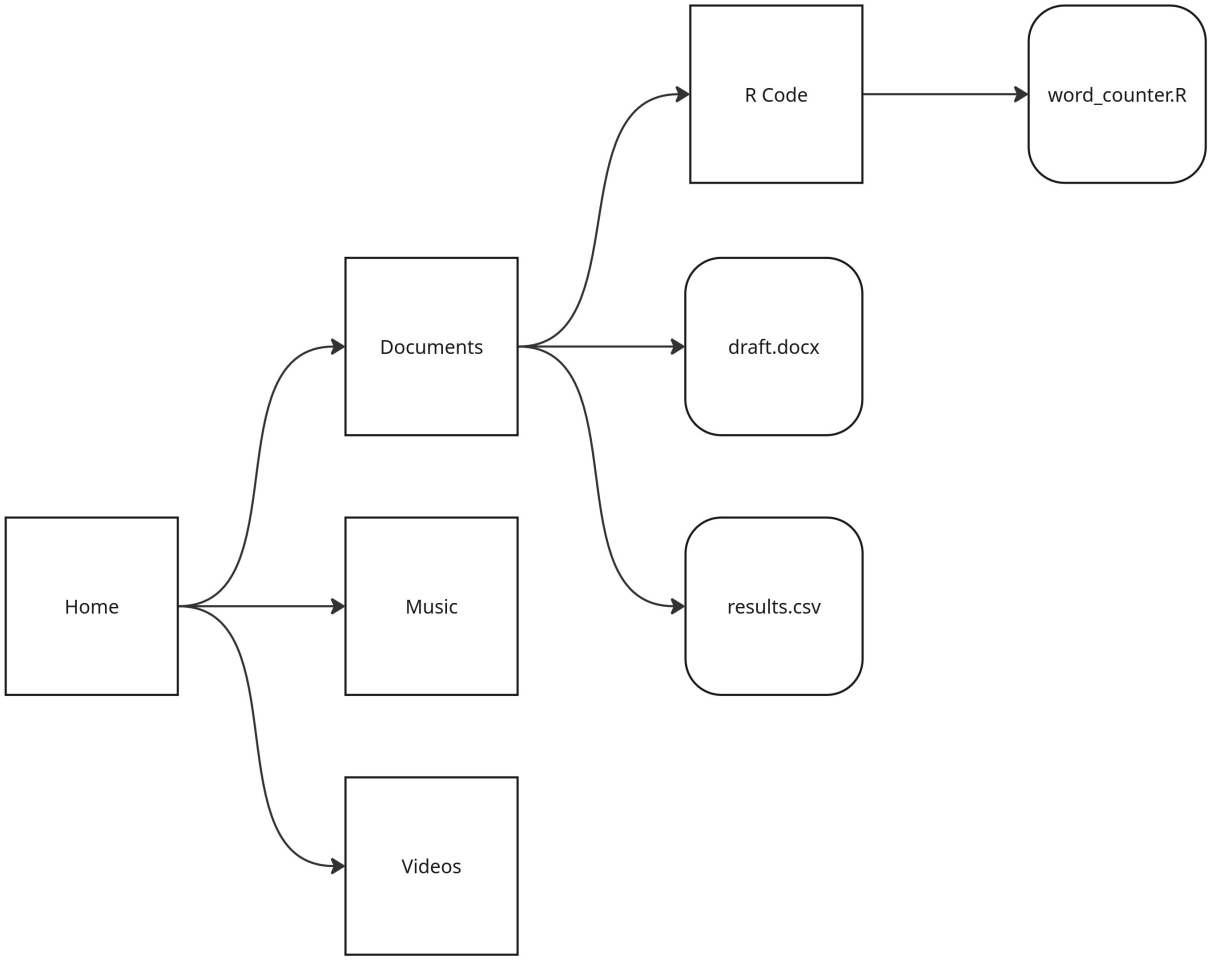
Challenge 2. File paths Encode Invisible Hierarchies

When a humanist thinks of data they might begin by imagining content such as letters, manuscripts, photographs, or datasets. But computers understand structure. They organize meaning spatially, through nested directories that shape how information can be found and related.

File directories are structured like trees, yet when we interact with our computers, we usually see only the leaves—the individual files and folders. The branching structure that connects them remains hidden, making

file paths conceptually difficult. When working in a programming language, we must mentally reconstruct this invisible hierarchy to navigate it effectively. Figure B.1 provides a visual representation of a hypothetical file hierarchy. To reach the file `word_counter.R` within this hypothetical hierarhcy, you would start from your Home directory, or the root of your computer. From there, you move into the Documents folder, which branches off from Home. Inside Documents, there's a subfolder called R Code, and inside that folder lives the file `word_counter.R`.

```
knitr::include_graphics("~/Repos/text-mining-for-historical-analysis/appendix_b/file_paths_are_trees.jpg
```



The route through this tree is represented by the following file path:

```
/Home/Documents/R Code/word_counter.R
```

However, this representation is not consistent across operating systems. Different systems use different conventions for denoting directories and paths—what works on one computer may not work on another. This inconsistency introduces the next challenge in working with file paths.

Challenge 3: File Paths Differ Across Operating Systems

Each operating system has its own way of writing and interpreting file paths. For instance, Windows uses backslashes (\) and drive letters (e.g., `C:\Documents\data.csv`), while macOS and Linux use forward slashes (`/`) and begin from a root directory (e.g., `/Users/username/Documents/data.csv`). In result, a file path that works on one computer may not work on another without adjustment.

Challenge 4: File Paths Depend on Context

If you run the same R code from a different folder, it may fail even though the file itself exists. This can feel confusing — as if the computer is behaving inconsistently — because the code worked perfectly a moment ago. What's really happening is that the computer's sense of "where you are" has changed. Since file paths are interpreted relative to the folder from which the code is executed, moving to a new location changes how the system searches for files.

In programming, two main types of file paths determine how a system locates files: absolute paths and relative paths.

Absolute paths begin from the root directory of the computer's file system and describe the complete route to the file, for example:

```
/home/steph/project/data/hansard_1850.csv
```

Relative paths begin from your "current working directory," that is, the folder where your code is being executed, and describe the file's location relative to that position.

```
data/hansard/1850/hansard_1850.csv
```

Different programming languages and systems use slightly different terminology to describe your location within the file hierarchy. The current working directory (abbreviated `cwd`), the present working directory (abbreviated `pwd`), and the working directory (abbreviated `wd`) refer to the same concept and are used interchangeably.

In R, you can see the folder from which R is accessing files by running the following command:

```
getwd()
```

```
## [1] "/home/stephbuongiorno/Repos/text-mining-for-historical-analysis/appendix_b"
```

If you run the same R code from a different folder, your working directory will change, and you may need to adjust your file paths accordingly to ensure the code can still locate the files.

File Paths, Reproducability, and Collaboration

As we have established, file paths can be complicated. Yet, they are important to master for ensuring reproducability and collaboration. In a collaborative environment, like a digital humanities lab, the directory structure on one researcher's computer is unlikely to match that of a collaborator. If your code contains file paths hard-coded to your own computer, it will fail when run on someone else's computer. Code that circulates between institutions, servers, and operating systems is especially subject to such a failure.

To avoid this problem, we recommend creating a project folder–a single, well-organized directory that contains all of the files, data, and scripts associated with a given research project. By treating this folder as the project's root and using relative paths (paths defined relative to that root), collaborators can re-run your code without needing to replicate your personal folder structure. This practice makes your project easier to share and maintain over time.

Programming languages also provide operating system–agnostic methods for handling file paths so that issues like the direction of slashes (forward `/` versus backward \) do not cause errors when sharing code.

For instance, in R, you can use the `file.path()` function to construct or join components of a file path. This function automatically uses the correct separator for your operating system so that your code will run across any operating system.

```
file_path <- file.path("data", "raw", "speeches.csv")
```

The resulting path is automatically be tailored to your operating system. We ran this code on a Linux computer, so the slashes appear in the forward (/) position.

```
file_path
```

```
## [1] "data/raw/speeches.csv"
```

Alternatively, when working collaboratively you might use the `here` library if you want your project to automatically locate files relative to the project's root directory, without manually setting or changing the working directory.

```
library(here)
```

```
## here() starts at /home/stephbuongiorno/Repos/text-mining-for-historical-analysis
```

```
relative_file_path<- here("data", "raw", "speeches.csv")
```

```
relative_file_path
```

```
## [1] "/home/stephbuongiorno/Repos/text-mining-for-historical-analysis/data/raw/speeches.csv"
```

Saving Visualizations to Your Computer for Analysis

Along with learning how to locate and read data into RStudio, another important reason to understand file paths is the ability to save visualizations directly to your computer's hard drive in bulk. This skill makes it possible to generate and store numerous visualizations for side-by-side comparison and interpretation, rather than producing one figure at a time and waiting for each to finish. We suggest that this workflow provides a more seamless and efficient experience and can aid interpretation by allowing researchers to identify patterns, contrasts, and trends across multiple visualizations without disturbance.

Throughout this book, we created visualizations one at a time, primarily for demonstration. We made this choice deliberately: generating several or dozens of visualizations and saving them directly to a hard drive without a clear understanding of the underlying file hierarchy can make visualizations difficult to locate. Visualizations that are saved onto a hard drive and forgotten about can quickly begin to consume storage space. Additionally, we did not want to spend readers' time trying to figure out where images were saved, which we feared could distract from the main focus of the chapter.

Nevertheless, the ability to save multiple visualizations programmatically is a valuable part of digital historical analysis. The following code demonstrates this process in a simple workflow for exploring how language changes across decades in the Hansard debates. For each decade, it loads the corresponding dataset, tokenizes the text into individual words, removes common stop words, and counts how often each remaining word appears. It then makes a bar chart showing the 20 most frequent words for that decade. It automatically saves one plot for each decade onto your computer's hard drive.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.1     v stringr   1.5.2
## v ggplot2   4.0.0     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 errors
```

```r
library(tidytext)
library(fs)

# function to process and plot top words for a given decade
# it takes the decade of interest and the name of the directory
# where the visualization will be saved
plot_decade <- function(decade, output_dir) {
  dataset_name <- paste0("hansard_", decade)
  data(list = dataset_name, package = "hansardr", envir = environment())
  hansard_data <- get(dataset_name)

  p <- hansard_data %>%
    unnest_tokens(word, text) %>%
    anti_join(stop_words, by = "word") %>%
    count(word, sort = TRUE) %>%
    slice_head(n = 20) %>%
    ggplot(aes(x = reorder(word, n), y = n)) +
    geom_col(fill = "steelblue") +
    coord_flip() +
    labs(
      title = paste("Top Words in Hansard Debates,", decade, "s"),
      x = "Word",
      y = "Count") +
    theme_minimal()

  outfile <- file.path(output_dir, paste0("hansard_", decade, "_topwords.png"))

  ggsave(outfile, plot = p, width = 6, height = 4, dpi = 300)

  message("Saved: ", outfile) }
```

```r
# define the decades of interest
decades <- c("1800", "1810", "1820")

# find your current working directory
# your visualizations will be saved here
output_dir <- getwd()

print(paste("Visualizations will be saved to:", output_dir))
```

```
## [1] "Visualizations will be saved to: /home/stephbuongiorno/Repos/text-mining-for-historical-analysis
```

```
for (decade in decades) {
  print(paste0("Processing ", decade))
  plot_decade(decade, output_dir) }
```

```
## [1] "Processing 1800"
```

```
## Saved: /home/stephbuongiorno/Repos/text-mining-for-historical-analysis/appendix_b/hansard_1800_topwo
```

```
## [1] "Processing 1810"
```

```
## Saved: /home/stephbuongiorno/Repos/text-mining-for-historical-analysis/appendix_b/hansard_1810_topwo
```

```
## [1] "Processing 1820"
```

```
## Saved: /home/stephbuongiorno/Repos/text-mining-for-historical-analysis/appendix_b/hansard_1820_topwo
```

We can use `list.files()` to see the newly created visualizations in their directory.

```
list.files()
```

```
## [1] "appendix_b.Rmd"             "file_paths_are_trees.jpg"
## [3] "hansard_1800_topwords.png" "hansard_1810_topwords.png"
## [5] "hansard_1820_topwords.png" "vis_1800"
## [7] "vis_1810"                   "vis_1820"
```

If you are having trouble locating the folder with your visualizations on your computer, you can run `getwd()` in R to display your current working directory. Copy the file path that appears in the console and paste it into your computer's file explorer (called "Finder" on macOS or "file browser" on Linux). This path represents the exact address that R is using—so opening it in your file explorer will take you to the same location your computer recognizes as the working directory.

Here is ours again:

```
getwd()
```

```
## [1] "/home/stephbuongiorno/Repos/text-mining-for-historical-analysis/appendix_b"
```

Advanced Tips

create a directory for each decade Put images into it

Again, we use `plot_decade()` which uses `ggsave()` to save the visualization to the hard drive.

```
# Define decades of interest
decades <- c("1800", "1810", "1820")

# Loop through each decade
for (decade in decades) {

  # Create a new directory for the current decade
```

```
  new_dir <- paste0("vis_", decade)

  # Create the directory if it doesn't exist
  dir_create(new_dir)
  print(paste("Created folder:", new_dir))

  # Print progress message
  print(paste("Processing", decade))

  # Set output directory for plot_decade() to save inside this folder

  # Use your plotting function
  plot_decade(decade, new_dir) }
```

```
## [1] "Created folder: vis_1800"
## [1] "Processing 1800"

## Saved: vis_1800/hansard_1800_topwords.png

## [1] "Created folder: vis_1810"
## [1] "Processing 1810"

## Saved: vis_1810/hansard_1810_topwords.png

## [1] "Created folder: vis_1820"
## [1] "Processing 1820"

## Saved: vis_1820/hansard_1820_topwords.png
```

If you use `list.files()` again you can see the new folders (starting with `vis_`) containing the new visualizations.

```
list.files()
```

```
## [1] "appendix_b.Rmd"            "file_paths_are_trees.jpg"
## [3] "hansard_1800_topwords.png" "hansard_1810_topwords.png"
## [5] "hansard_1820_topwords.png" "vis_1800"
## [7] "vis_1810"                  "vis_1820"
```

Conclusion

The study of file paths may seem far removed from the interpretive questions that animate digital history or text analysis, yet it reveals how interpretation depends on underlying structures. Each visualization, dataset, or code script in this book relies on invisible systems of organization that determine what can be accessed, reproduced, or shared. To understand file paths, then, is to understand how such digital order becomes the precondition for analytical order.

In this chapter, we demonstrated how to navigate those structures, since file paths are a lingering challenge that impede research. Distinguishing between absolute and relative paths, identifying the working directory,

constructing platform-independent routes with `file.path()` and `here()`, and saving visualizations programmatically form the groundwork for more ambitious research—projects that can be reproduced, shared, and scaled across collaborate teams.

In addition, using `ggsave()` we demonstrated how automation can make analysis more seamless. By saving visualizations directly to folders, we can generate multiple figures in succession and compare them side by side, rather than producing one at a time. This workflow supports interpretation, allowing researchers to trace linguistic or historical patterns across decades without the mental interruption of generating images one-by-one.