Appendix A

Appendix A

Other Essential R Packages for Computational Historical Analysis

If you have programmed before, you might appreciate having a "cheat sheet" that outlines the key functions and concepts from popular R libraries used throughtout this book as well as for applications beyond this book. This overview can help experienced programmers quickly orient themselves to R's ecosystem and idioms, espeically if they are coming from Python, JavaScript, or another language.

If you are new to programming, however, we encourage you to begin with Chapter 1: Counting Top Words Over Time. This chapter introduces you to R step-by-step, starting with the basics of loading data and summarizing it in plain language. You'll learn core programming concepts and be introduced to significant libraries, and key functions within these libraries, as they arise in the context of historical inquiry, like identifying the most frequently used words in political speeches.

data.table
`data.table` extends the functionality of `data.frame()` for high-performance data manipulation. It is especially well-suited for working with large datasets due to its speed and efficient memory usage. Unlike tibbles or data frames, data.table performs many operations by reference, which avoids copying and significantly improves performance on large datasets. Unlike the tidyverse, which emphasizes readability through piping and clearly named functions, `data.table` uses an in-place modification approach with a consistent `[i, j, by]` syntax for subsetting, transforming, and grouping data. Key functions and syntax patterns include:

- `data.table()`: creates a data table, similar to how `data.frame()` creates a data frame.
- `[i]`: filters rows based on a condition (e.g., `DT[x > 5]`).
- `[ , j]`: applies expressions or transformations to columns (e.g., `DT[ , mean(x)]`).
- `[ , := ]`: adds or updates columns by reference without copying the data (e.g., `DT[ , new_col := x + y]`).
- `[ , .(summary1 = mean(x), summary2 = sum(y))]`: returns multiple summary statistics using `.()` to bundle results.
- `[ , .N, by = group]`: counts the number of rows by group, where `.N` is a built-in variable for row count.

dplyr
`dplyr` is a tidyverse package designed for fast and consistent data processing It provides a set of core functions that operate on data frames or tibbles, and allow analysts to express data transformations in a readable, step-by-step way. Some of the most commonly used functions include:

- `mutate()`: adds new variables to a data frame while preserving existing ones.
- `transmute()`: creates new variables and drops all existing ones not explicitly included.
- `select()`: chooses specific variables (columns) based on their names.
- `filter()`: extracts rows that meet specified logical conditions.
- `group_by()`: groups data by one or more variables, allowing subsequent operations to be performed within each group.
- `arrange()`: reorders rows based on the values of one or more variables.

`dplyr` also supports the magrittr pipe operator `%>%`, which lets analysts chain multiple operations together in a linear, readable form without creating intermediate objects. This can make complex data transformations easier to follow and debug.

ggplot2
`ggplot2` is a tidyverse package for creating data visualizations based on *The Grammar of Graphics* (Wilkinson, 2005), a systematic framework for describing and building graphics. Rather than using a single function to generate an entire plot, `ggplot2` constructs plots by layering components and mapping data variables to visual aesthetics. The plotting process begins with the `ggplot()` function, which initializes the plot object. Aesthetic mappings—such as x, y, color, or fill—are defined using `aes()`. Layers are then added to the plot using functions such as:

- `geom_point()`: creates scatter plots.
- `geom_bar()`: creates bar charts
- `geom_line()`: creates line graphs
- `geom_histogram()`: creates histograms
- `geom_boxplot()`: creates boxplots
- `facet_wrap()` and `facet_grid()`: creates subplots based on categories
- `labs()`: customizes axis labels, titles, and legends
- `theme()`: modifies the visual appearance of the plot

Each step is combined using the `+` operator.

httr
`httr` is an R package that simplifies working with HTTP requests, making it easier to interact with web APIs. An HTTP request is a standardized way for a client (like R) to communicate with a server over the internet, often to retrieve, submit, update, or delete data. A web API (Application Programming Interface) is a structured set of rules that allows software applications to communicate with each other, often over the web. Many APIs follow a RESTful architecture, which uses standard HTTP methods (like `GET` and `POST`) to access resources using URLs. The `httr` package provides an R interface for sending these requests, handling user authentication, and parsing the data returned by the server. It offers key functions such as:

- `GET()`: sends an HTTP GET request to retrieve data from a specified URL.
- `POST()`: sends an HTTP POST request, typically used to submit data to a server.
- `content()`: extracts and parses the content of a response, often returned in JSON, text, or HTML.
- `status_code()`: returns the HTTP status code from a response, which helps determine if the request succeeded (e.g., 200 OK) or failed.
- `add_headers()`: includes custom headers in a request, such as API keys or content types.
- `authenticate()` adds login credentials for APIs that require basic authentication.
- `modify_url()`: constructs or adjusts URLs by appending paths or query parameters.

jsonlite
`jsonlite` provides a way to convert between R objects and JSON (JavaScript Object Notation), a common text-based format for storing and exchanging structured data. JSON is commonly used in web development and APIs due to its readability and compatibility across programming languages. `jsonlite` provides functions for engaging web data, APIs, and interoperability with other programming environments within R. It is tidyverse-friendly and supports deeply nested structures (e.g., data that contains multiple levels of hierarchy, such as lists within lists or records within records), automatic conversion of data frames, and streaming large datasets. Key functions include:

- `fromJSON()`: reads JSON data from a file or URL and converts it into R objects (typically lists or data frames).
- `toJSON()`: converts R objects to JSON strings.

- `stream_in()`: reads large JSON files in chunks (useful for newline-delimited JSON), allowing efficient memory use.
- `stream_out()`: writes large JSON datasets to a file line-by-line for use with systems expecting newline-delimited JSON.

lubridate

`lubridate` is part of the tidyverse ecosystem but must be loaded separately using `library(lubridate)`. It provides a consistent set of tools for working with dates and times in R, which can otherwise be error-prone and difficult to manage using R's base functions. In this way, `lubridate` supports temporal analyses across a variety of applications, including historical analysis, ongitudinal studies, time series, and event tracking. It recognizes a wide range of date formats and allows users to convert character strings into date-time objects using functions like:

- `ymd()`, `mdy()`, and `dmy()`: parses dates in year-month-day, month-day-year, or day-month-year formats.
- `ymd_hms()`: parses full date-time strings.
- `year()`, `month()`, `day()`, `hour()`, `minute()`, and `second()`: extracts or modifies components of a date-time object.
- `now()` and `today()`: retrieves the current time or date.
- `interval()` and `duration()`: measures spans of time.

pdftools

The `pdftools` package is particularly helpful for accessing digitized archival documents, extracting structured data from reports, or preparing PDFs for text mining. It provides tools to extract and manipulate text content found in PDF files. PDF (Portable Document Format) is a widely used file format designed to preserve the layout of documents across different platforms. However, extracting information from PDFs can be challenging due to their complex structure. `pdftools` converts PDFs into plain text, that is, unformatted character data that can be further processed or analyzed in R. Key functions include:

- `pdf_text()`: extracts text from each page of a PDF as a character vector, with one element per page.
- `pdf_info()`: retrieves metadata from a PDF file, such as author, title, number of pages, and creation date.
- `pdf_render_page()`: renders a PDF page to a bitmap image, useful for visual inspection or OCR.

plotly

`plotly` creates interactive, web-based visualizations. While not part of the tidyverse, it integrates well with tidyverse data structures like tibbles and complements `ggplot2` by adding interactivity such as hover labels, zooming, and clickable elements. Key functions include:

- `plot_ly()`: builds interactive plots from scratch by mapping data to visual attributes like x, y, type, and color.
- `ggplotly()`: converts a static ggplot object into an interactive plotly visualization.
- `layout()`: customizes axis labels, titles, legends, and plot dimensions.
- `subplot()`: combines multiple plotly graphs into a single layout.
- `add_trace()`: adds additional layers or data series to an existing plotly object.

quanteda

`quanteda` is a powerful R package for natural language processing (NLP), designed for managing, transforming, and analyzing large collections of text. It is known for its speed and efficient memory handling, especially when working with sparse and high-dimensional data common in text analysis. Unlike tidyverse packages, which are built around data frames and rectangular data, `quanteda` uses specialized text objects—-such as the corpus, tokens, and document-feature matrix (dfm)—-to represent and process text data more efficiently. These objects support operations like tokenization or keyword search without the need to reshape tabular data.

While tidyverse workflows typically emphasize row-by-row or column-by-column transformations, `quanteda` works at the level of documents and features (e.g., words). Key functions in `quanteda` include:

- `corpus()`: creates a corpus object from raw text data.
- `tokens()`: splits text into tokens (e.g., words, phrases), with options for removing punctuation, numbers, or stopwords.
- `dfm()`: creates a document-feature matrix (DFM), a table of word counts across documents used for most quantitative text analysis.
- `kwic()`: performs keyword-in-context searches to examine how specific terms appear in context.
- `textstat_frequency()`: generates statistical summaries such as word frequencies or lexical diversity.
- `tokens_ngrams()`: constructs multi-word expressions (e.g., bigrams, trigrams) for phrase-level analysis.

reader
`readr` is a tidyverse package that streamlines the process of importing and exporting rectangular data. It is significantly faster and more consistent than base R functions like `read.csv()` or `read.table()`, and it returns tibbles by default, which integrate smoothly with other tidyverse tools. Key functions include:

- `read_csv()`: reads comma-separated values into a tibble.
- `read_tsv()`: reads tab-separated values.
- `read_delim()`: handles files with custom delimiters.
- `write_csv()`: exports tibbles to CSV files.
- `write_tsv()`: exports tibbles to tab-separated files.
- `read_lines()`: reads a file line by line, useful for text files.

readxl and writexl
When working with spreadsheet data, `readxl` and `writexl` provide alternatives to base R. readxl enables fast reading of Excel files (`.xls` and `.xlsx`), and it imports data as tibbles. `writexl`, on the other hand, allows writing tibbles or data frames to Excel files. These packages support collaboration with non-programmatic users, such as collaborators who use Excel. Key functions include:

- `read_excel()`: reads Excel files into R as tibbles; supports both `.xls` and `.xlsx` formats.
- `excel_sheets()`: lists the sheet names in an Excel file, useful for previewing or iterating through multiple sheets.
- `write_xlsx()`: writes one or more data frames or tibbles to an `.xlsx` file and supports named lists of data frames to create multi-sheet workbooks.

spacyr
`spacyr` is an R wrapper for the `spaCy` Python library. A wrapper is a tool or package that provides a simplified interface to another tool, often written in a different language. In this case, `spacyr` is an R wrapper fora library written in Python—-allowing analysts to use `spaCy`'s abilities directly in R without needing to write Python code. It can integrate smoothly with `quanteda`, particularly for use with `quanteda` corpora, and allows for linguistically rich analysis of text beyond basic token counts. It is especially useful for tasks requiring grammatical understanding or entity recognition. Key functions include:

- `spacy_initialize()`: sets up the connection between R and the underlying spaCy Python environment.
- `spacy_parse()`: performs tokenization, part-of-speech tagging, lemmatization, and syntactic dependency parsing. It returns a structured data frame with one row per token.

stringr
`stringr` is a tidyverse package that simplifies and standardizes string manipulation in R. It provides a consistent set of functions for detecting, extracting, modifying, and formatting strings, all of which follow a clear naming convention (starting with `str_`). Some of the most commonly used functions include:

- `str_detect()`: determines whether a pattern is present within a string.
- `str_replace()`: replaces the first instance of a pattern within a string.
- `str_replace_all()`: replaces all instances of a pattern within a string.
- `str_sub()`: extracts or replaces substrings using character positions.
- `str_to_lower()` and `str_to_upper()`: convert strings to lowercase or uppercase.
- `str_trim()` removes leading and trailing whitespace.
- `str_c()` concatenates strings together, similar to `paste()` in base R.
- `str_split()` splits a string into pieces based on a specified delimiter.

tibble
`tibble` is a modern alternative to base R's `data.frame()` and serves as the standard data structure in the tidyverse. Tibbles display only the first 10 rows and the columns that fit on screen, never convert strings to factors by default, and support columns with complex or list-like data. Tibbles integrate with `dplyr`, `ggplot2`, and the rest of the tidyverse. Key functions include:

- `tibble()`: creates a new tibble from vectors or variables.
- `as_tibble()`: converts existing data frames or lists into tibbles.
- `enframe()`: transforms named vectors into two-column tibbles, useful for reshaping key-value data.
- `deframe()`: converts a two-column tibble back into a named vector.
- `glimpse()`: provides a compact, horizontal view of the data, useful for quickly inspecting variable types and contents.

tidyverse
The tidyverse (Wickham et al., 2019) is a collection of R packages that reimagines base R's syntax to support clearer analysis. It introduces a grammar that emphasizes readability and expressiveness, making code easier to write, understand, and debug. The tidyverse is especially well-suited for working with rectangular data—tables where each column is a variable and each row is an observation. When the tidyverse is loaded using `library(tidyverse)`, it automatically loads several core packages, including `ggplot2` for data visualization, `dplyr` for data manipulation, `tidyr` for reshaping data, `readr` for reading rectangular data, `purrr` for functional programming, and `tibble` for modern, consistent data frames. These packages are designed to work seamlessly together, using standardized outputs, consistent naming conventions, and the pipe operator `%>%` to support readable, step-by-step workflows. This integration allows users to focus on the logic of their analysis rather than the technical details of individual functions.

viridis
`viridis` provides perceptually uniform color scales that are easy to interpret, colorblind-friendly, and print well in grayscale. Originally developed for scientific visualization, viridis is especially effective in contexts where consistent contrast and accessibility are important. In `ggplot2`, `viridis` palettes can be applied using:

- `scale_fill_viridis_c()` for continuous color scales, where colors transition smoothly across a range of numeric values—such as temperature, frequency, or time.
- `scale_fill_viridis_d()` for discrete color scales, where each distinct category—such as political party or document type—is assigned a separate, easily distinguishable color.
- `scale_color_viridis_c()` and `scale_color_viridis_d()` for applying continuous or discrete color scales to lines and points, respectively.