Using R with RStudio & Tidyverse

Arend M Kuyper

## Introduction

The purpose of this chapter is to provide you with an introduction to the free statistical software R along with RStudio and the tidyverse packages. While R can have a steep learning curve and be intimidating to new users, especially those new to coding, RStudio and the tidyverse packages make R much more accessible. Beyond improving accessibility, these tools are intentionally designed to make users more productive with R and improve the reproducibility of their work. Not surprisingly, this chapter is focused on the “how to” of foundational data work. We will highlight and demonstrate essential best practices for using R with RStudio and the tidyverse for quantitative research. However, don’t let this chapter’s focus on first steps of data work deceive, R and its ecosystem of extension packages allow for the implementation of all the statistical techniques found in this book and much, much more.

## R

R is a powerful and highly flexible statistical analysis tool. It provides a wide array of statistical techniques and methods while also providing highly developed graphics capabilities. R’s ability to create publication-quality graphics has been, and continues to be, one of its greatest strengths. It does all of this free of charge with a dedicated community of developers.

The R project is a GNU project, which means it is a free software, mass collaboration project. Knowing that R is open source and is actively developed and maintained through mass collaboration provides important context for users concerning its basic structure and potential resources. R can be considered as being made up of two 2 parts:

1. the base R system that is downloaded from the Comprehensive R Archive Network, also known as CRAN, and
2. a large ecosystem of extension packages, sometimes called libraries.

The base R system is actively maintained and updated for various operating systems by the R Core Team. Typically there is 1 major update along with 2 minor updates per year. Having an active release schedule like this is critical for the success of open source software. This ensures up-to-date compatibility with operating systems and signals to users that it won’t be abandoned.

While the base R system adequately covers most statistical needs and functionality, it is arguably the large ecosystem of extension packages that has contributed to R’s growth. Packages can provide implementations of methodologies not currently in base R, improve the usability of R, or provide tools that allow you to do non-statistical tasks (e.g. sending emails or building websites). In particular, the tidyverse packages have been very influential and have made working with R significantly more accessible. We will take a closer look at the tidyverse packages later in this chapter.

As of this writing, the CRAN package repository features over 21,000 contributed packages. The CRAN repository checks all packages for compatibility and expects the packages to be maintained, which means users can expect packages from the CRAN repository to work with R. This work is done by a network of volunteers, called the CRAN team, and it is a testament to the size and dedication of the R community that this is possible. Though CRAN is the primary R package repository, users can find packages through:

* CRAN-like repositories such as BioConductor and R-forge;
* GitHub and BitBucket;
* Personal websites.

While packages outside of the CRAN repository aren’t vetted by the CRAN team for compatibility, they can be very useful. They may implement cutting edge statistical techniques or provide tools for more bespoke analyses. Going through the CRAN submission process can be daunting, time intensive, and restrictive so it is not uncommon to find very useful packages not hosted on CRAN.

### Using R

A common roadblock for many new R users is that it requires the users to write code or commands. This can be a significant hurdle for many, but there are several free software options that make working with R much more user-friendly. The most popular being RStudio, which we will discuss in more detail later on in the chapter. The need to write code or commands isn’t removed, but it is made much more intuitive and accessing help is made easier. Using R and having to write code is a net positive for increasing the reproducibility of research, at least for computational and analysis work.

The value of learning to write R code is significantly enhanced by following best practices for coding and setting up workflows. When users are first learning it can seem unnecessary to follow such advice, but it is important to avoid developing bad and inefficient habits. RStudio and the tidyverse are specifically designed to guide users to follow and implement best practices. We will be highlighting and demonstrating some of these best practices in the following sections, but readers wanting more guidance should see the **Suggested further readings**.

## RStudio

RStudio is an integrated development environment (IDE) designed to make working with R more accessible and productive. While R comes with its own graphical user interface (GUI), it simply was not designed with a wide range of users in mind. So, it is common that R be paired with some other open source software such as RStudio, R Commander, Deducer, jupyter notebooks, vscode, or positron. RStudio is by far the most widely used and known. A significant portion of R’s growth in usage can be reasonably attributed to RStudio. It has become synonymous with R. RStudio can be downloaded for free from <https://posit.co/download/rstudio-desktop/>.

When RStudio is first opened, there are 4 panes as seen in [Figure 1](#fig-rstudio-layout). Sometimes the source pane is missing, but that is easily remedied by opening a new R script (.R file): **File > New File > R Script**.

1. The **Source pane** is where you can edit and save R scripts, which are essentially text files containing R code. This is where most of the data analysis work happens and should be documented.
2. The **Console pane** is used to write short interactive R commands.
3. The **Environment pane** displays temporary R objects as created during that R session. It also contains the useful history tab.
4. The **Output pane** displays the plots, tables, or HTML outputs of executed code along with saved files. This pane also includes the packages and help tabs which are especially useful since the first is for managing and installing packages and the second is setup to help access documentation.

|  |
| --- |
| Figure 1: RStudio’s basic layout |

### Prepare for success

Before starting to work with data, there are 2 best practices that should be discussed:

1. Adjusting a few RStudio default options to improve long-term reproducibility of your data analysis work.
2. Use RStudio projects to improve organization and collaboration, which includes with your future self.

By default, workspaces will load everything that you had been working on previously, from .Rdata files. While this might sound harmless or even desirable, it actually creates a workflow that could easily lead to ghost or zombie objects. That is, objects that are not reproducible because we may have ran code out of order ot altered code and forgot to re-run it to update things. By being automatically saved we might not catch this error until it is way too late. So, to develop R scripts that are complete and self-contained records of the data work we need to make a few adjustments. In RStudio, set this via **Tools > Global Options**, uncheck “Restore .RData into Workspace at Startup” and choose **Never** on the “Save workspace to .RData on exit” as seen in [Figure 2](#fig-rstudio-wrk-options).

|  |
| --- |
| Figure 2: RStudio’s Globol Options: Workspace Best Practice |

The larger and more sprawling research and analysis work becomes, the more important it is to be organized. Meaning, it is important to have a “home base” of operations where everything for your data analysis or research work is self contained. By using RStudio projects it becomes straightforward to organize your work. You can think of an RStudio project as being a home directory/folder that will ultimately contain everything for your data analysis project. RStudio projects provide a solid workflow that will serve you well in the future:

* When starting a data analysis project (or any work in R), create a new RStudio project,
* Keep all data files there; organized in a data sud-directory.
* Keep all scripts there; edit them, run them in bits or as a whole. Naming them sequentially (e.g. 0-loading-data.R, 1-inspecting-data.R, etc) is recommended.
* Save your outputs (plots and cleaned data) there. Organized into appropriate sub-directories or folders

By using an RStudio project, everything you need is in one place, and cleanly separated from all the other projects that you are working on. The also make collaboration easier. The folders could be maintained on a shared drive so or version control software like git can be integrated into the projects. If used appropriately, then we should be able to simply zip/compress the RStudio project folder and share it with anyone else.

To create a new project in the RStudio, use the **File > New Project**. In the New Project wizard that pops up, select **New Directory**, then **New Project**. Pick a name for the project, for example “cwift-examples” (name of project for examples provided later in this chapter), and then click the **Create Project** button. This will launch a new RStudio Project inside a new folder called “cwift-examples”. The name of the project should appear in the top-right hand corner of Rstudio as seen in [Figure 3](#fig-cwift-examples-project). A good piece of advice, if it says “Project: (None)”, then don’t do any work. Set up a miscellaneous project for scratch work and musings. Always be working in a project.

## Tidyverse

The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures. (*Tidyverse*, 2024)

This generally makes the tidyverse syntax easier to read and implement, especially for those new to coding and data analytic work. It makes chaining together data processing steps intuitive and easier than in other programming languages. The core tidyverse packages are those most likely needed for everyday data analyses and are all loaded by the meta {tidyverse} package. As of {tidyverse} 2.0, the following 9 packages are included in the core tidyverse:

* {ggplot2} is a system for declaratively creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.
* {dplyr} provides a grammar of data manipulation, providing a consistent set of verbs that solve the most common data manipulation challenges.
* {tidyr} provides a set of functions that help you get to tidy data. Tidy data is data with a consistent form: in brief, every variable goes in a column, and every column is a variable.
* {readr} provides a fast and friendly way to read rectangular data (like csv, tsv, and fwf).
* {purrr} enhances R’s functional programming toolkit by providing a complete and consistent set of tools for working with functions and vectors.
* {tibble} is a modern re-imagining of the data frame, keeping what time has proven to be effective, and throwing out what it has not.
* {stringr} provides a cohesive set of functions designed to make working with strings as easy as possible.
* {forcats} provides a suite of useful tools that solve common problems with factors (categorical variables).
* {lubridate} provides a set of tools to make it easier to work with dates and times in R, which can be difficult and inconsistent in base R.

To install the tidyverse packages, go to **output pane > Packages > Install**. The package install wizard will pop-up and enter “tidyverse” as the name of the page to download. Ensure that “Install dependencies” is checked, then click **Install**. Alternatively, we could have ran install.packages("tidyverse") in the console. Not only will we get the 9 core tidyverse packages, but all of the packages they are dependent on will also be installed.

Beyond the core tidyverse packages, we will be using several additional packages as we work through a few data explorations in the next section. Packages such as tidycenus (tools to obtain census data in tidy format), {janitor} (tools for cleaning data), {sf} (tools for making maps), and {skimr} (tools to quickly skim data). Packages like {tidycensus} and {janitor} are often described as tidyverse-adjacent because they are built to work with the tidyverse and therefore adhere to the shared design philosophies.

The tidyverse’s influence on the R package ecosystem has been transformative. It has laid a foundation upon which developers in a large open-source community can collaborate and build tools that are much more accessible (e.g. tidyverse-adjacent packages). Developers can build their packages to leverage all existing tools and benefits of the tidyverse ecosystem which helps to alleviate dependency issues and compatibility with current and future features. Ultimately, this means users will be able to pick up new packages much more efficiently because of the shared design philosophies and standards.

## Data Exploration Examples

Time to see R with RStudio and tidyverse in action. We will providing highlights from a short exploration of educational data that has a spatial or geographical component. The objective is to demonstrate the usefulness and flexibility of using these tools for data work. For those interested in seeing more details and/or wanting to work through the code directly, then please visit the GitHub repository (<https://github.com/akuyper/cwift-examples>) for this data exploration. Regardless, we encourage readers to explore the GitHub repository because it is a demonstration of best practices when working with data.

Before taking a closer look at the data and exploration work, let’s peak at the layout of the project and discuss what we see in [Figure 3](#fig-cwift-examples-project):

* It is clear we are working in a project because the project name, “cwift-examples”, is visible in the top-right corner.
* The R script in the Source Pane is well organized. It has comments to help guide humans through the code, it starts with a preamble that loads the necessary packages and data, and we see indentation and white/negative spaces being used to make the code more readable — also makes it easier to debug. It follows The tidyverse style guide (Wickham, 2024). As a best practice, a consistent style guide should always be used with a data project.
* The Environment Pane contains items created by running the code.
* The Console Pane has remnants of code being ran (it is pushed from the source and executed/ran in the console).
* The Output Pane is displaying the Files tab and we can see the project well organized with appropriate sub-directories and file names.

|  |
| --- |
| Figure 3: Layout overview for the cwift-examples project |

A closer look at the **Source Pane: R Script** in [Figure 3](#fig-cwift-examples-project) reveals that the |> is being used repeatedly. This is the pipe operator and the tidyverse use it to build data pipelines (i.e. take a dataset and perform a series of actions on it). Code writing and readability is generally easier when using the pipe.

Consider [Figure 4](#fig-piping), it provides a simple example to demonstrate the difference between using pipe or not. We have a dataset, called my\_data, and we want to sequentially apply actions h(), g(), and f(), That is we want to take my\_data, *and then* apply h(), *and then* apply g(), *and then* finally apply f(). Replacing the *and then*’s with |> naturally leads to the piping example in [Figure 4](#fig-piping). Non-piping requires that we work from the inside out and doesn’t really match how we naturally organize the work.

There is another pipe operator, %>%, that is commonly used because it came with the tidyverse and pre-dated |>. They generally work the same so it is best to pick one and stick with it as a matter practice. |> is commonly called the native pipe because it was added to base R. So ,|> is always available in R. We don’t have to load the tidyverse or more specifically the maggrittr package to have a pipe operator.

|  |
| --- |
| Figure 4: Non-piping or piping in R |

### The data

We will be using the 2021 American Community Survey Comparable Wage Index for Teachers (ACS-CWIFT or CWIFT) data from the Education Demographic and Geographic Estimates (EDGE) Program (*Comparable Wage Index for Teachers (CWIFT)*, 2024).

The Comparable Wage Index (CWI) is an index that was initially created by the National Center for Education Statistics (NCES) to facilitate comparison of educational expenditures across locales (principally school districts, or local educational agencies—LEAs) or states (state educational agencies— SEAs). The CWI is a measure of the systematic, regional variations in the wages and salaries of college graduates who are not PK-12 educators as determined by reported occupational category. It can be used by researchers to adjust district-level finance data at different levels in order to make better comparisons across geographic areas. (Cornman et al., 2019)

The basic idea behind any CWI is that workers will demand higher wages in areas with high costs of living or where commonly sought after local amenities such as a good climate, low crime rates, and museums are lacking. This means, it should be possible to measure geographic variation in the cost of hiring teachers (PK-12 educators) by observing systematic, regional variations in the wages of comparable workers who are not PK-12 educators. Put another way, say accountants, nurses, data analyst, and software engineering all earn 20% more than the national average for their professions in San Francisco, then it is reasonable to expect that the cost of hiring teachers in San Francisco would also be 20% more than the national average for teachers.

A core motivation for the developing CWIFT estimates is to enable more meaningful comparisons concerning teacher expenditures across school districts. That is, we can use the CWIFT to normalize dollar amounts and make them comparable by dividing the dollar amounts by the district-level CWIFT, which are already normalized to the national average wage.

Suppose we wanted to make comparisons between salary expenditure data from the Elementary and Secondary Information (ELSI) system for the 2019-20 school year. The total current expenditures on salary per pupil for the Los Angeles Unified School District was $6,137 and was $5,433 in Palm Beach County School District (Florida). So it appears Los Angeles Unified is spending $704 more on salary per pupil. This is obviously misleading comparison since the cost of living is very different betwen these two areas. When adjust for their difference in purchasing power using their 2021 CWIFT estimates, Los Angeles Unified is $5,369 ($6,137 / 1.143) and Palm Beach County is $5,601 ($5,433 / 0.970). The story has flipped and Palm Beach effectively spent $232 more per pupil than Los Angeles Unified.

Another useful application of the CWIFT is to adjust state aid to a school district based on differences in wages due to geographical variations. Suppose a state, say South Dakota, is considering a program intended to provide an additional $100 per pupil in wages. An extra $100 per pupil does not go as far in different locations around the state (i.e. geographic variations in the cost of education). The 2021 CWIFT for Harrisburg, SD is 0.895, or about 10% lower than the national average; the 2021 CWIFT for Pierre, SD is 0.739, or approximately 26% lower than the national average. To receive the same increase in purchasing power as a $100 increase in the Pierre School District, the Harrisburg School District would need to receive $121.11 ($100\*(0.895 / 0.739)).

|  |
| --- |
| Figure 5: The first 15 observations in the state CWIFT dataset |

### dplyr: data wrangling

Demo(s) with comments and a few tips for best practices

|  |
| --- |
| Figure 6: Demonstration of various plots for examining state CWIFT estimates: (A) Approximate 95% confidence intervals for state CWIFT estimates (B) Density plot for state CWIFT estimates (C) Boxplot for state CWIFT estimates, (D) A density dot plot for state CWIFT estimates |

|  |
| --- |
| Figure 7: Demonstration of two choropleth/heat maps for examining state CWIFT estimates: (A) US state map with shading according to CWIFT estimates (B) State bins map with shading according to CWIFT estimates |

### ggplot2: data visualization

Demo(s) with comments and a few tips for best practices

|  |
| --- |
| Figure 8: Demonstration of a grouped boxplot to examine county CWIFT estimates |

|  |
| --- |
| Figure 9: Demonstration of a choropleth/heat map to examine county CWIFT estimates |

|  |
| --- |
| Figure 10: Demonstration of arranging plots to make comparisons between county CWIFT estimates within states. County maps with shading according to CWIFT estimates for (A) California, high variance with most counties above 1 for CWTFT (B) South Dakota, low variance with most values below 1 for CWIFT (C) Maryland, high variance with most values below 1 for CWIFT |

|  |
| --- |
| Figure 11: Demonstration of two choropleth/heat maps created when examining state CWIFT estimates: (A) US state map with shading according to CWIFT estimates (B) State bins map with shading according to CWIFT estimates |

## Comment on AI & R coding

Generative artificial intelligence, based on large language models (LLMs), looks to be a transformational assistive technology. It has the potential to supercharge productivity and is well on its way to becoming a standard tool in a researcher’s tool kit. All facets of the research process are being impacted, which includes writing R code and conducting data analyses. There are several experimental packages or addins to leverage LLMs directly in RStudio: {chattr}, {tidychatmodels}, {gptstudio} and GitHub’s Copilot to name a few. Copilot is explicitly for code-completion and generation.

As far as coding and conducting data analyses, these tools have made it possible to simply provide prompts that contain basic guidance and in return we receive annotated code with explanations. They are capable of providing code exploratory data analyses. It is almost too good to be true, and it can be. Without sufficient knowledge of the R syntax and available tools like the tidyverse, it becomes difficult to judge the quality of the output or even if the output is worthwhile. So, for R-novices these tools can be a potential trap. Like all tools they need to be used wisely. A good starting point would be to use them as debuggers of your own code or asking them to improve upon your code. For more experienced R users, they can be used to generate starter/skeleton code. This can help cut down on some of the tedious coding work, but it requires a level of discernment that only comes with experience.

Before we jump right to using these tools in research, we need to be aware of data security and ethical concerns. We must be careful with the data these tools have access to. In fact, usage of LLMs may represent breaches of data use agreements and/or violation of privacy for study participants. We suggest a default practice of never exposing experimental data to AI tools, unless cleared in advanced. To be clear, AI tools can still be used to generate code, you just shouldn’t include access to your data in the prompts.

Bottom line. AI tools can be very useful, epecially for coding. But they must be used with caution when handling confidential or private data.

## Conclusion

## Research essentials (250-300 words)

Other than R scripts, Quarto (.qmd) or R Markdown (.Rmd) documents could be used to document the work. While beyond the scope chapter, we encourge you to

version control git & git hub

## Questions for further investigation (50-100 words)

How does getting setup compare to other statistical software you may have used? If setup was an issue, have you searched for any free workshops or materials online? Try using R with RStudio and the tidyverse by playing with a data project of your own and/or by downloading the data exploration examples RStudio project from its GitHub webpage (<https://github.com/akuyper/cwift-examples>).

## Suggested further reading (50-100 words/3 resources & why)

Hadley Wickham Danielle Navarro & Pedersen (2024)

Hadley Wickham Mine Çetinkaya-Rundel & Grolemund (2024)

(Wickham, 2024)

(Wickham et al., 2019)

## References

*Comparable wage index for teachers (CWIFT)*. (2024). NCES Education Demographic; Geographic Estimates (EDGE); <https://nces.ed.gov/programs/edge/Economic/TeacherWage>.Accessed: 2024-08-21

Cornman, S. Q., Nixon, L. C., Spence, M. J., Taylor, L. L., & Geverdt, D. E. (2019). *Education demographic and geographic estimates (EDGE) program: American community survey comparable wage index for teachers (ACS-CWIFT)* (NCES 2018130). U.S. Department of Education; National Center for Education Statistics.Retrieved 2024-08-20 from http://nces.ed.gov/pubsearch/

Hadley Wickham, Danielle Navarro, & Pedersen, T. L. (2024). *ggplot2: Elegant graphics for data analysis (3e)*. <https://ggplot2-book.org/>.Accessed: 2024-08-20

Hadley Wickham, Mine Çetinkaya-Rundel, & Grolemund, G. (2024). *R for data science (2e)*. <https://r4ds.hadley.nz/>.Accessed: 2024-08-20

*Tidyverse*. (2024). <https://www.tidyverse.org/>.Accessed: 2024-08-15

Wickham, H. (2024). *The tidyverse style guide*. <https://style.tidyverse.org/index.html>.Accessed: 2024-09-03

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. <https://doi.org/10.21105/joss.01686>