SIPPS Coding Workshop (2023)

01 Setting up R and RStudio & Introduction

2023-06-14

Introduction

Welcome to the first SIPPS coding workshop!

In these workshops, we will teach you how to write programs in R that are useful for conducting research in psychology.

Programming involves writing precise instructions telling your computer how to transform some input into the output you want. When it comes to working with data in R, programming is how we tell R to load data, process it, plot it, fit models to it, and so on.

Introduction

Today, we will cover:

1. Logistics & structure of workshops
2. General coding advice
3. Installing R & RStudio
4. Finding your way around RStudio
5. Installing R packages
6. Relative and absolute file paths
7. Loading data into R

Logistics & structure

- The files for all lessons can be found on the SIPPS website.
- A week before most of the coding workshops, we will post videos to watch.
- A day or two before each workshop, we will sometimes post files to download for use during the workshop.
- During the workshops, you will work on coding challenge problems that illustrate key concepts, with assistance from the instructor(s).
- After the workshops, we will post an answer key for the coding challenge problems.

Logistics & structure

- After each workshop, we will ask you to fill out a **post-workshop survey** to help us figure out what is working or not working and improve the workshops going forward.
- If you want any additional help, you can reach out to the SIPPS instructors via email (contact info on the website) or via Slack.
- Any questions about logistics?

Coding Advice

- If you don't have much or any exposure to coding and/or the R language, *be patient with yourself*. Learning a programming language is like learning other languages, and it can take time to learn the syntax and semantics well enough to feel comfortable.
- **Do not be afraid to look for answers online.** Experienced programmers do this all the time. Learning how to search for answers to your coding programs is a key research skill.
  - Note: In these workshops, we do not recommend using ChatGPT to write yoru code. When learning to code, struggling to figure out the right answer is an important part of the learning process. In addition, ChatGPT is not always correct, and you need to build a sufficient knowledge base to be able to tell whether the code it writes is actually doing what you asked it to do.

Coding Advice

- **Save your work often!** RStudio can crash, and losing hours of work is not fun.
- If you want more resources outside of our workshop, there are a ton of free R tutorials online. See our website for a few that we recommend.

R & RStudio

- To use R in this workshop, you need **both** R and RStudio installed.
- Why are we downloading and installing two different files?
  - R and RStudio are different programs that do different things.

R & RStudio

- R is a *programming language*, like Python, Java, etc.
- Programming languages turn commands written in human-comprehensible scripts (which you will learn how to write!) into machine code that the computer can execute (which you will never see).
- Installing R allows your computer to understand and execute commands written in the R language.
- You can run commands in the R language in many ways, including (1) on the command line or (2) in an Integrated Development Environment (IDE).

R & RStudio

- RStudio is the most widely used IDE for R. Most people who spend their time writing R code do so in RStudio.
- **When you want to work with R, just open RStudio. In these workshops, you will never need to run R commands any other way.**
- As an IDE, RStudio requires R to run and it gives you a bunch of useful bonus tools that don't come standard with R.

Installing R & RStudio

To install R, visit the CRAN website to download the appropriate version of R for your operating system (Linux, Mac, or Windows). Run the installer contained within your downloaded file.

To install RStudio, visit the RStudio website to download the appropriate version of RStudio for your operating system. Again, run the installer that downloads to your computer.

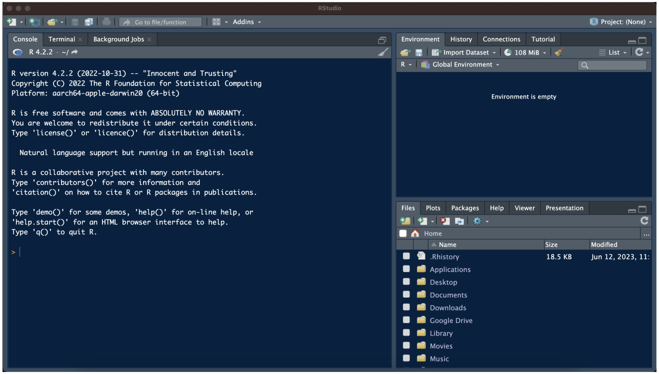
Installing R & RStudio

Has everyone been able to install R and RStudio? **These programs will be essential for all the coding workshops, so we want to make sure everyone has them installed.**

We will take a 5 minute break. Anyone who has questions or concerns about installing R or RStudio can join one of the breakout rooms, and we can try to troubleshoot.

RStudio Panes

When you open RStudio, by default you see the Console pane, the Environment pane, and the Files pane.

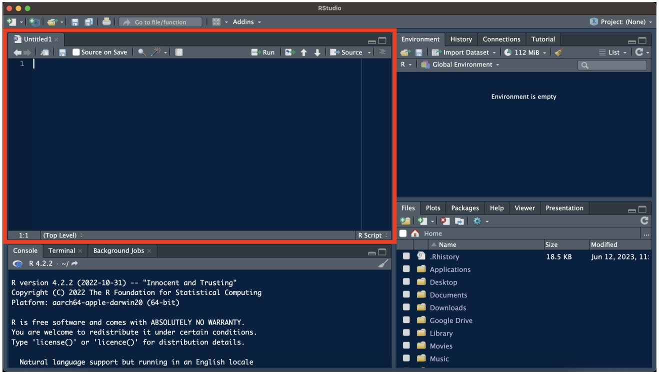


```
R version 4.2.2 (2022-10-31) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Name	Size	Modified
.Rhistory	18.5 KB	Jun 12, 2023, 11:
Applications		
Desktop		
Documents		
Downloads		
Google Drive		
Library		
Movies		
Music		

RStudio Panes

When you open a new script, you see the Source pane.

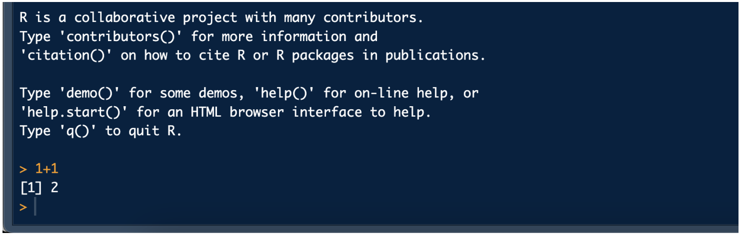


```
R version 4.2.2 (2022-10-31) -- "Innocent and Trusting"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale
```

Name	Size	Modified
.Rhistory	18.5 KB	Jun 12, 2023, 11:
Applications		
Desktop		
Documents		
Downloads		
Google Drive		
Library		
Movies		
Music		

Console Pane

If everything is working correctly, you can type R commands in the console (default position: lower left), and press `Enter` to execute those commands.



```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 1+1
[1] 2
>
```

Note: The console is a great place to try out new stuff or do one-off trouble-shooting, but it doesn't save your code. When writing any code you want to preserve, you always want to write scripts – that ensures both you and others can reproduce your analyses!

Environment/History Pane

In the Environment/History pane (default position: upper right), you can see some useful information about your current R session:

- *Environment*: This tab shows all of the objects you have created in your R session, which include data you have loaded, variables you have created, and functions you have defined.
- *History*: this is where you can see your command history of all the R commands you've run in this session. Use it wisely!

Files/Plots/Packages/Help Pane

This pane (default position: lower right) provides a wide variety of useful information:

- *Files*: This is a rudimentary file browser, in case you want to use this to click through your folders and open R scripts.
- *Plots*: If you render a graph, it appears in this tab.
- *Packages*: This is a list of all the packages (see below) you have installed. Any loaded packages have a checkmark next to them. (You can also load packages this way, but doing so in code is preferable; see below.)
- *Help*: You can search for and read documentation of any R function here. I spend a lot of time here!

Source Pane

- When using RStudio for research or work, you will spend most of your time in the source pane, which by default default appears in the upper left after you open a script file or start a new script file.
- This is where you **edit R script files.**
- R scripts files contain saved R code, which you can run as many times as you want.
- Writing your code in script files gives you records of the data processing and analyses you've written and allows both you and others to reproduce that processing and analysis.
- An R script file will always end in `.R`.

R project files

- When you are working in RStudio, you'll often be working with a particular set of code scripts, raw data, and other files that are all saved in one folder.
- An **R project file** tells RStudio "this folder is a place where I am doing work related to one complete project" and helps you keep your data analysis projects organized.
- You can create an R project file inside of a folder where you plan to do R work, or you can open an existing R project file in a folder you've already set up.
- *Note: In addition to helping organize your files, R projects also make it easier for multiple people to execute your code, which is very important when working on teams or posting your analyses on public repositories.*

R project files

At this point, we'd like you to follow along with this presentation in RStudio.

In the RStudio menu bar, please go to: `File` -> `Open File`, find the `coding-01.Rproj` file you downloaded, and open it.

The only difference you will notice, for now, is that the project name appears in the top right corner of the RStudio window.

RMarkdown Files

Next, in the RStudio menu bar, please go to: `File` -> `Open File`, find the `01_lesson.Rmd` file you downloaded, and open it.

If you scroll down to line 196, you will see this text!

RMarkdown Files

An .Rmd file is an **R markdown** file. R markdown files are useful teaching tools, because they allow you to combine regular text, such as this, with runnable code chunks, like the one below.

```
print("This is a code chunk!")
```

```
## [1] "This is a code chunk!"
```

In the .Rmd file you have open, you can click the green right-pointing arrow in the top right corner of the chunk below to run it.

We will be using R markdown files for all of our coding tutorials, but when you write scripts to analyze your data, you should generally use a regular R script file.

Installing Packages

Packages are bundles of functions in R, written by other programmers. Oftentimes, if there's something you want to do in R, there already exists a function to do it that you can find in a package.

For these workshops, you will need to install packages outside base R. Among these are: `psych`, `tidyverse`, `ggplot2`, `lme4`, `haven`, and `car`. Since these packages don't come with base R, you have to install them using the `install.packages` command.

Note: You only need to install packages **once**, so you generally do not need to incldue `install.packages` commands in your R scripts.

Installing Packages

Here's an example of how to use the function to install one package:

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

Now try installing `tidyverse` by modifying the code above:

```r
# Write your code in this block
```

Installing Packages

Here's an example of how to use the function to install one package:

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

Now try installing `tidyverse` by modifying the code above:

```r
# Write your code in this block
install.packages("tidyverse")
```

Installing Packages

You can also install multiple packages at once:

```r
install.packages(pkgs = c("ggplot2",
                          "lme4"))
```

Now try installing `haven` and `car` by modifying the code above:

```r
# Write your code in this block
```

Installing Packages

You can also install multiple packages at once:

```r
install.packages(pkgs = c("ggplot2",
                          "lmer"))
```

Now try installing `haven` and `car` by modifying the code above:

```r
# Write your code in this block
install.packages(pkgs = c("haven",
                          "car"))
```

Package Documentation

Packages come with documentation, which explains how to use the functions that they contain.

When working with a new package, particularly one you have found yourself, it is very important to read the documentation, so that you understand what the included functions are doing.

You can view the documentation for a package in the Help pane using the `help` function:

```
help(psych)
```

The documentation is also often available on **CRAN**.

Using Packages

If you want to use a function from a package you have installed, there are two ways to do it.

The first way, which is less common because it leads to longer lines of code that are harder to read, is to call the function you need and specify what package it's in when you call it.

Using Packages

For example, the `psych` package contains the `describe` function, which computes some descriptive statistics about an input vector. (We will go into more details about vectors in the next workshop!) You can call the `describe` function from the `psych` package as follows:

```
data <- c(1, 2, 3, 4, 5, 6)
psych::describe(data)
```

```
##    vars n mean   sd median trimmed  mad min max range sk
## X1    1 6  3.5 1.87    3.5     3.5 2.22   1   6     5
```

Using Packages

The mean_se function from the ggplot2 package also computes the mean if a vector, as well as values that are one standard error above and below the mean. Try calling it below with data as the argument:

```
# Add your code here
```

Using Packages

The mean_se function from the ggplot2 package also computes the mean if a vector, as well as values that are one standard error above and below the mean. Try calling it below:

```
# Add your code here
ggplot2::mean_se(data)
```

```
##     y     ymin     ymax
## 1 3.5 2.736237 4.263763
```

Loading Packages

- The more common way to use functions from a package is to *load* the packages into your active R session.
- This allows you to quickly and easily call the functions in those packages without writing the name of the package before the function.
- While you only have to install packages once, you have to load them in every R session that you want to use them.
- Generally, you should start your R scripts by loading all the packages you will need in that script.

Loading Packages

There are two ways to load packages:

- `library` loads a package and stops the script if the package isn't installed.
- `require` loads a package and outputs a warning but does not stop the script if the package isn't installed.

You almost always want to use `library` rather than `require`.

Loading Packages

The syntax for `library` is as follows, though the second is more common:

```r
# Argument to library() can be package name in a string
library("car")

# Or as a variable
library(tidyverse)
```

Loading Packages

After loading packages, we no longer need to include the package name when calling a function from the package:

```
library(psych)

describe(c(1, 2, 3, 4, 5, 6))
```

```
##    vars n mean   sd median trimmed  mad min max range sk
## X1    1 6  3.5 1.87    3.5     3.5 2.22   1   6     5
```

Loading Packages

- Usually, it is easier to use functions in a package by loading the package using `library` and then calling the functions without re-writing the package name over and over.
- However, sometimes we want to use the `package::function` syntax: Different packages can contain functions with the **same name**, which can lead to conflicts and unpredictable behavior depending on the order in which packages are loaded.
- The possibility of these conflicts is the reason R does not automatically load all your packages at the beginning of a session.
- In addition, specifying what packages your script requires as the beginning also helps other people run it without lots of trial and error.

Loading Data Into R

- Before we can start analyzing data in R, we need to understand how R interacts with files on your computer. This allows us to load data from data files.
- To load data into R, the data must be saved on your computer. (Some R packages allow you to scrape data from the internet from inside R. Even that data must be saved locally somewhere for R to load it.)
- Often, after manipulating data (e.g., cleaning it), you will want to export that data from R and save it as a file on your computer.
- In order to effectively load data into R and save the output of our analyses, we need to understand how R interacts with the file system on your computer.

The Working Directory

Every R session has a *working directory*, or a “home base” folder. Essentially, this where R starts looking when you try to load any files.

You can find out what folder is your working directory using the `getwd()` command as below.

```
getwd()
```

```
## [1] "/Users/cjmott/Library/CloudStorage/GoogleDrive-cjm
```

If you run this command on your own computer, you will get a **different output**. That’s because the folder structures on our computers are all different.

The Working Directory

- When you launch R by opening an R project file, R automatically sets your working directory to be the folder where this R project file is saved.
- That is why the getwd() displayed the path where the R project file was saved!
- Thus, if your R project file is in the same folder as your raw data, scripts, etc., then you can use relative file paths (see below) to more easily load those files.

File Paths

- Every single file and folder on your computer has an address.
- You can navigate these paths in your computer's built in file browser, but in order for R to be able to access these files, you have to tell R the address.
- You can think of your computer's main drive as a building, with only one front door, and a series of rooms (folders), which may contain stuff (files) or doors leading to more rooms (sub-folders).
  - You can walk through any series of connected doors to get to the room you're looking for.
- A file path is a set of directions of which rooms to walk through to get to a particular room (folder) or object (file) in a room.

File Paths

Because each room/folder on your computer has a name, file path directions look like the following:

- Mac/Linux: `"Folder name/Sub-folder name/File"` (note the forward slashes!)
- Windows: `"Folder name\Sub-folder name\File"` (note the backslashes this time!)
- **Note: Windows uses backslashes internally, but when loading files in R, only forward slashes work on all operating systems.**

There are two ways of specifying file paths: *absolute* and *relative*, which we'll get to next.

Absolute File Paths

- An **absolute file path** is a file's full address on your computer.
- An absolute path tells you how to get to a file starting from the **root folder,** which is essentially the one invisible giant folder holding literally everything saved on your computer.
  - In your computer-building, the root folder is the front door.
- If you give the directions to a room in your computer-building by *starting at the front entrance*, your computer can always teleport to your front entrance and follow those directions.

Absolute File Paths

The way you specify that a file path starts from the root folder (the front entrance to your computer) differs operating systems:

- Mac/Linux: Start your file path with *a single forward slash*, e.g. `"/Users/me/Documents/etc"`
- Windows: Start your file path with *the name of your drive*, e.g. `"C:/Users/me/My Documents/etc"`
- **Note: Recall that Windows uses backslashes internally, but R paths should be written with forward slashes in all operating systems**

Absolute File Paths

- Absolute file paths can be really long. Remember what `getwd()` output earlier? That's the absolute file path to where you save the documents for this workshop.
- Using absolute file paths also makes it harder for other people to run your code, because absolute file paths differ by operating system and often contain the account username.
- That's why we often want to use *relative* file paths.

Relative File Paths

- Relative file paths are directions to your files that assume you're starting in the current working directory.
- Relative file paths are shorter to type because you don't need to specify all the directions to get from the root folder to the current working directory.
- If you set the current working directory with an R Project file in a folder that contains all your code and data, then other people can run your code on their computer easily after loading the project!

Relative File Paths

To specify a relative path, you only need to start your path with file or folder name that exists in your current working directory.

Example:

- My current working directory is `plant foods`, containing a subfolder for `fruits` and a subfolder for `vegetables`.
- `plant foods` is inside a parent folder called `foods`.
- I'm looking for a file called `apple.jpg` inside the `fruits` folder.
- The relative file path is `fruits/apple.jpg` *with no forward-slash or drive name in front of it.*

Any path that does *not* start with a forward-slash or drive name is assumed to be a relative path, so R will start looking in your current working directory.

Relative File Paths

To ensure relative file paths in your code work properly, we recommend that you do not change your working directory during an active R session.

- R project files automatically set your working directory for you, so you don't need to set your working directory when you open up R if you open up an R project file.
- When you use R project files, you can open multiple R instances in different working directories if you need to work on multiple projects at once, so you shouldn't need to change your working directory inside one R session.
- Changing your working directory can lead to code errors if using relative file paths.

Special file path keywords

The following are special keywords you can use in file paths:

- Every OS:
  - single period `.`: The current working directory. A relative path like `"./fruits/apple.jpg"` is equivalent to `"fruits/apple.jpg"`.
  - double period `..`: The folder one level *up*. If the current working directory is `"plant foods/vegetables"`, I can go to the folder for fruits using the path `"../fruits"`. You can chain the double period to go backwards multiple folders, e.g., from `fruits`, `"/../../"` is `foods`.
- Mac/Linux only:
  - tilde `~`: This refers to your *home folder*. This is usually located at `"/Users/your_username"`. Your Documents folder, among other folders, is in your home folder, so the path `"~/Documents"` is equivalent to `"/Users/your_username/Documents"`.

Reading data into R

- One reason file paths are useful is that they allow you to read data (e.g., from an existing spreadsheet or text file) into R. In order to load such a file, you will need to specify the file path so R knows where the data is located.
- Let's say you want to load a CSV file containing data on Shakespeare's plays in R. This file is called `shakes.csv`, and it is located in the same folder as this R file.
- Note: when working with R, as well as most other statistical computing programs, it is *much* easier to work with `.csv` files than `.xls`, `.xlsx`, or files with other proprietary formats. If you have a choice about what format to download your data in, you should generally choose `.csv`.

Reading data into R

We're going to load the data into R using a function called `read_csv` (from the `tidyverse` package), but we need to make sure R can actually find the csv file of interest. This is where file paths come in!

If you would like to use an absolute file path, simply type out your entire computer path, all the way up to and including the `shakes.csv` file. You may want to use `getwd()` to get a sense of what the beginning of your absolute file path looks like, since it can be confusing. (This is a perfect time to use the Console!)

```
getwd()
read_csv("C:/Users/cjmot/Google Drive (Columbia)/SIPPS/Work
# replace everything except for shakes.csv with your own f
```

Reading data into R

If you would like to instead use a relative file path, then when pointing R to your file, you need only write out the path *relative* to this R file.

If you've opened this R file from an R Project file, the path will be set automatically.

You can also manually set it using the function `setwd()`. (Though we do not recommend doing this.)

```
setwd("C:/Users/cjmot/Google Drive (Columbia)/SIPPS/Worksho
read_csv("shakes.csv")
# replace everything except for shakes.csv with your own f
```

Reading data into R

- After running the code above, you saw a preview of the file in R. Sometimes this kind of quick preview is helpful!
- However, in order to do more with the data, you will want to save a copy of the file as an object in your R environment.
- Note: You can tell that the file has *not* yet been loaded into your environment by taking a look at the Environment tab in the upper right section of your RStudio window. Unless you were working in R before this workshop began, you will see a message noting that it is empty!

Loading the file as an R object

To load the file into your R environment, give it a name (e.g., `shakes`) and use the `<-` (the *assignment operator*) to save the data contained in the CSV into an R object with that name.

```r
shakes <- read_csv("shakes.csv")
```

Once you have done so, you will see this object listed in the Environment tab in the upper right section of your RStudio window.

Next week, you will learn a lot of things you can do with data loaded into the R Environment!

Knitting an R Markdown

- The last, but not least, cool thing about R Markdown files is that they can be exported as beautiful documents in a myriad of formats.
- This export process is called *knitting*, and should have a very obvious button at the top of the page called *Knit* with a knitting needle and ball of yarn. If you click the arrow directly to the right of the word knit, you'll see the options available to you.
- Today, we're going to knit to an html document, so click the arrow, then *Knit to HTML*. It should automatically open in a new window so you can immediately admire its beauty and grace, as well as save to a new file called `lesson.html` in your working directory.
- If you have edited the file paths in the code chunks above (which will raise errors on your computers because you don't have those file paths!), you should be able to knit this R markdown file right now