

Lecture 2: Working with RStudio and RMarkdown

Contents

Why R?	1
Getting Started	1
Saving your work	3
RMarkdown files	4
Running code in RMarkdown files	6
Loading our first dataset	7
Exploratory Data Analysis	8
Conclusion	10
Acknowledgements	10
Learning Objectives	

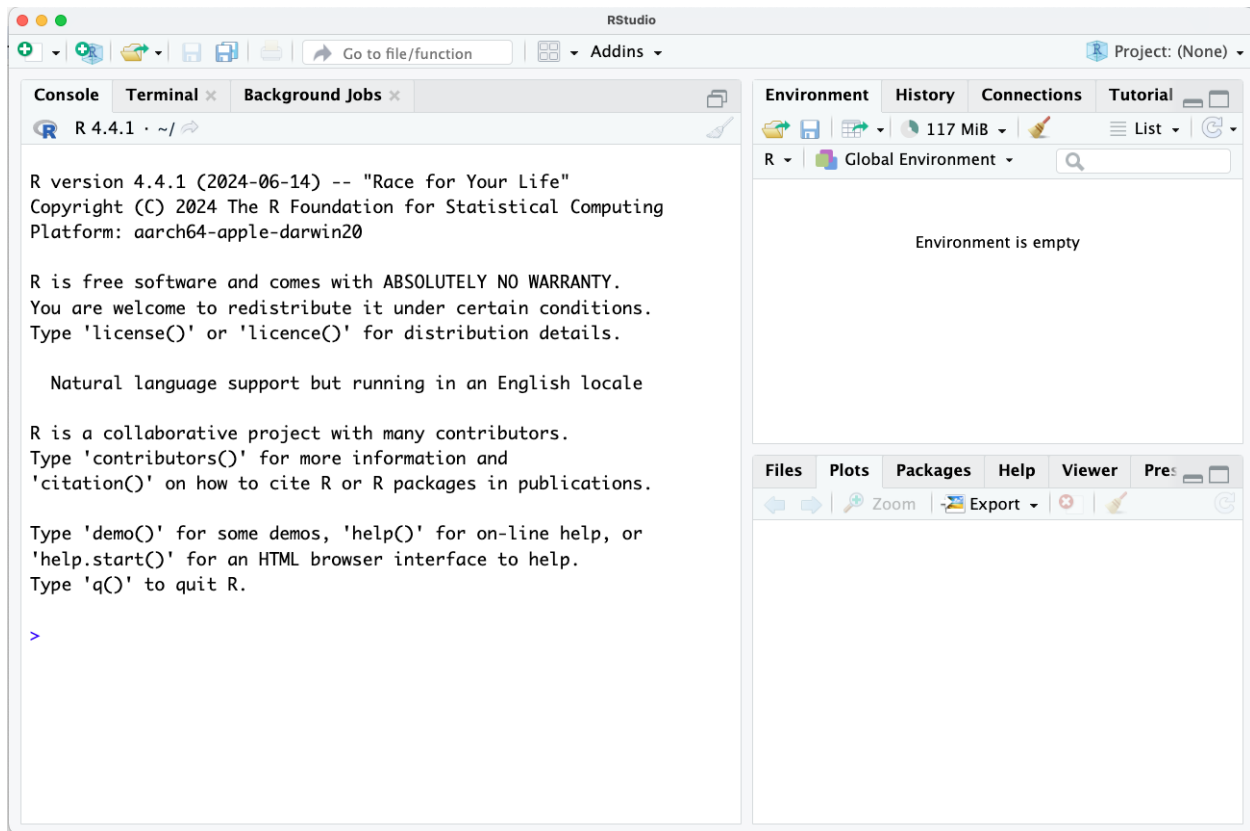
- Become comfortable with the RStudio interface.
- Understand how to record written explanations and code chunks in an RMarkdown file and `knit` the file into a report.
- Run some fun, basic commands in R.
- Load our first dataset into R: use this to review basic concepts about exploratory data analysis.

Why R?

R is an incredibly popular programming language for data analysis. It is used by statisticians but also by scientists in biology, psychology, economics, chemistry, etc. R is fully featured and very powerful, and unlike Stata or SaaS, it is *free and open source*. This means that you can continue to use it for free after you finish this class, and beyond. Additionally, R is *extensible*. When a statistician develops a new method, or when a scientist decides that they need a specific tool to analyze their specific data, they (or someone who wants to use it in R) will often implement the method as a collection of functions in R called a *package*. Other individuals who want to use the method or conduct a similar type of data analysis can download the package and use these functions without needing to rewrite all of the code themselves. The Comprehensive R Archive Network, or **CRAN**, contains over 13,000 contributed packages.

Getting Started

Hopefully you were able to download R and RStudio by following the instructions on GLOW. If so, go ahead and launch RStudio. You should see a window that looks like the image shown below.



The panel on the lower left is where the action happens. It's called the *console*. Everytime you launch RStudio, it will have the same text at the top of the console telling you the version of R that you're running. Below that information you will see the symbol `>`. This is called the *prompt*: it is a request for a command. Initially, interacting with R is all about typing commands into the console and interpreting the output.

The panel in the upper right contains your *environment*, which will show you all of your named variables and datasets once you create/load them. You can also view a *history* of all commands you have previously entered in the console.

Any plots that you generate will show up in the panel in the lower right corner. This is also where you can browse your files, access help, manage packages, etc.

Loading Packages

R is an open-source programming language, meaning that users can contribute packages that make our lives easier, and we can use them for free. For many labs in the future we will use the following R packages:

- The suite of **tidyverse** packages: for data wrangling and data visualization
- **openintro**: for data and custom functions with the OpenIntro resources
- **tinytex**: you may need this in order to knit an RMarkdown document into a PDF. For example, you need this if you have a Mac and have never installed tex.

Hopefully you were able to install these packages using `install.packages()` using the pre-class instructions. Note that you can check to see which packages (and which versions) are installed by inspecting the *Packages* tab in the lower right panel of RStudio. If you do not see `tidyverse`, `openintro`, and `tinytex` in this panel, it means that you need to run the following code from the pre-class instructions.

```
install.packages("tidyverse")
install.packages("openintro")
install.packages("tinytex")
tinytex::install_tinytex()
```

Next, regardless of what you completed before class, you need to load these packages in your working environment. We do this with the `library` function. Note that you need to **load** the packages every time you restart RStudio, whereas you only need to **install** the packages once ever. Note that `tinytex` is a special case package that we do not need to load: we needed it to make sure we had a Tex installation on our computer, which RMarkdown will need in the background in order to knit to PDFs.

```
library(tidyverse)
library(openintro)
```

Interacting with R in the Console

The most basic way to interact with RStudio is to use the console as a calculator. Using the console, you can perform basic arithmetic operations and you can assign values to variables.

Anything that you type after the `>` (the prompt) will be executed as R code. For example, type `5+3` after the prompt and then press enter (return). You should see your answer right away. If instead you type `x <- 5+3` and then enter, the console will no longer print out 8, but it will save your value of `x`. Note that `x` now appears in your *environment*. Now type `x` in the console and hit enter; verify that it prints the appropriate value.

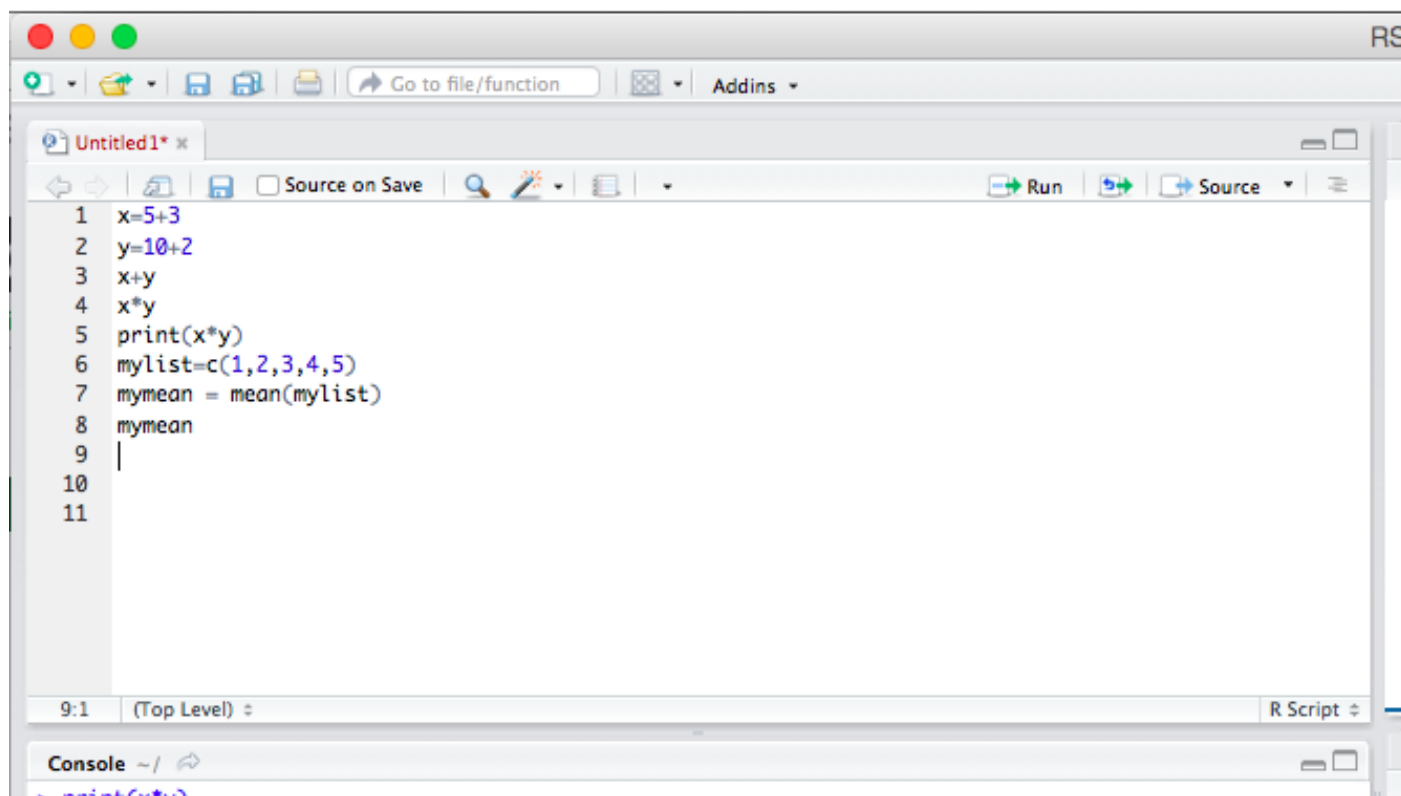
```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
```

```
> 5+3
[1] 8
> x <- 5+3
> x
[1] 8
> |
```

Saving your work

Anything that you could ever want to accomplish in R can be done directly in the console. However, if you are writing commands that you might want to save for later or share with others, it is better to write these inside a file, rather than in the console. For example, if you are working on an assignment, you probably want to be able to access your code again later to check your work.

The most basic type of file is an R Script file. You can create one by going to File, New File, R Script. It should show up in the upper left panel of your screen. This file works just like the console, but instead of running a command every time you click enter, you can write several lines at once without running them. If your cursor is on a certain line, clicking “run” in the top right corner will execute just that line. Clicking “source” will execute all lines at once. Experiment by typing code such as the following into your R Script file. Make sure you are comfortable with the difference between running an individual line and running the whole file.



RMarkdown files

RScript files are great for code, but they are not great for communicating your ideas to others. An RMarkdown file lets you intersperse *chunks* of R code with chunks of text and output the result as a nicely formatted document. For all homework assignments in this course, you will use RMarkdown to create documents that intersperse code with answers to written discussion questions. We will typically post the .Rmd file corresponding to the actual homework assignment, and early in the semester you can use these files as *templates*, where all you need to do is fill in your answers.

For today’s tutorial, we will make an RMarkdown document from scratch. In your RStudio application, select File, New File, R Markdown. If prompted, select PDF as the default output format. Also, if prompted

to install any packages, please do so! You will know that you were successful if you end up seeing an example file that looks like this:

```
1 ---
2 title: "Untitled"
3 output: pdf_document
4 date: "2024-09-09"
5 ---
6
7 ```{r setup, include=FALSE}
8 knitr::opts_chunk$set(echo = TRUE)
9 ```
10
11 ## R Markdown
12
13 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more
14 on using R Markdown see <http://rmarkdown.rstudio.com>.
15
16 When you click the Knit button a document will be generated that includes both content as well as the output of R code chunks
17 within the document. You can embed an R code chunk like this:
18
19 ```{r cars}
20 summary(cars)
```

Anything written in this document on plain white background is interpreted as text (write here as you would write in a Microsoft word document), and any code written on gray background is interpreted as R code (write here as you would write in the R console, with one command per line). We call the gray areas **code chunks**.

Note that, when used on white background, the # symbol creates titles and headers that show up in large font in the output document. We use text such as **### Exercise 1** to label the exercises. When used inside of a code chunk, the # symbol in R creates a code **comment**. This can be used to write regular text inside of a code chunk. Any text written in a code chunk after the # symbol is ignored; it is not run as R code. We use comments to leave you messages inside of lab and HW templates, such as “write your answer here”.

Knitting an RMarkdown document means turning the input file (.Rmd) into a nicely formatted output file (.pdf in this case). To knit your current template document, select **knit** from the buttons at the top of the RMarkdown file. A nicely formatted document will open up in a new window.

In the next few minutes, modify your example document so that it looks something like this one:

```
Source Visual
1 ---
2 title: "Lecture 2"
3 author: "Anna Neufeld"
4 output: pdf_document
5 date: "2024-09-10"
6 ---
7
8 ```{r setup, include=FALSE}
9 knitr::opts_chunk$set(echo = TRUE)
10 ```
11
12 ### Getting started
13
14 This is my first RMarkdown document. I want to see if I can create a PDF document that contains
15
16 ```{r}
17 5+8
18 ```
19
20 When I knit this, hopefully my PDF document will contain the printed number 13!
21
22 ### More advanced
23
24 Let's try one more chunk~
25
26 ```{r}
27 x = 5+7
28 y = 10-3
29 x-y
30 ```
31
32 What do you see when you knit the PDF document?
33
34 ### Bonus
35
36 What happens if you remove the line ``x=5+7`` from the code chunk above? Can you still knit?
```

Running code in RMarkdown files

You will notice that just typing the code in the gray chunk did not produce any sort of answer for us. To see the answer to this math expression, we need to **run** this code.

```
```{r}
5+8
```
```

First note that all code in your chunks is automatically run each time you knit your document. Knit your template document now to try this out! Did it print your answer?

You can knit your R Markdown file every time you write code, but sometimes the knitting process takes a few minutes and you just want to run a few lines at a time to make sure they are working. To run chunks without knitting, you can use either the Run button on the chunk (green sideways triangle) or you can highlight the code and click Run on the top right corner of the R Markdown editor. While it is good to run individual chunks to test out ideas and explore, we also strongly recommend that you **knit your document** each time you finish an exercise. The knitting process can sometimes lead to errors. Students often run in to trouble if they wait to try knitting their document until right before the assignment deadline. Knitting after each exercise ensures that you catch errors as you go.

When an R Markdown document is knitting, it only has access to the variables that are created inside of the R Markdown document itself. The R Markdown document does not get to use the environment in the upper right hand corner of your screen while it is knitting. The R Markdown document only gets to see the variables that are defined inside of the document. To illustrate this concept, delete the line `x <- 5+7` from your sample document.

```
x <- 5+7
y <- 10-3
x-y
```

Can you still run the chunk locally? What happens if you try to knit your document?

It is also important in an R Markdown document that all variables are defined in order. You will notice that exercise 5 contains two code chunks in your template. What happens if you add `x <- 5+7` back into your sample document, but you add it to its own code chunk. Can you knit the document now?

Loading our first dataset

While using R as a calculator was a fun warmup, in this class we are typically going to use R to analyze datasets. With this in mind, we need to learn how to load our dataset into R. There are several ways that we will do this in this class.

- Many datasets are built into packages. These are extremely easy to load. As long as you have the package installed, you just load the package and then access the relevant dataset with the `data()` function. This is what you will be doing on HW1!
- Most commonly, we will load data in from an Excel file or a .csv file. This is probably the way that many of you will load your data for your final projects.
- Sometimes, we will load our datasets directly from a URL. This can be easier and quicker than up-loading/downloading files.
- Sometimes (later in the semester) we will generate our own data!

At the start of the semester, we will always give you very clear instructions on how to load the required dataset. Eventually, for your projects, you will need to know how to troubleshoot and load the data yourself!

Loading the “Getting to Know You” dataset

Over the weekend, 67 Stat 202 students filled out the “getting to know you” survey. Google forms lets me export the survey responses into a google sheet / Excel spreadsheet. I removed the names, pronouns, and email addresses from the spreadsheet, and I also removed the answers to the “free response” questions (e.g. “Why are you taking this class”). I changed the column names for readability. Finally, I saved the resulting spreadsheet as a .csv.

I posted the anonymized data on GLOW as a .csv file, and I also posted it on my github page. There are two ways that you can load the data. The easiest is to run this code:

```
form_data <- read.csv("https://anna-neufeld.github.io/stat202_tutorials/Week1/form_data_anonymous.csv")
```

The slightly more cumbersome (but useful!) way involves downloading the .csv document from GLOW and importing it using the `import dataset` button. We will go over this way later in the semester, as storing a file locally can lead to come more cumbersome knitting issues.

Do you now have an object called `form_data` in your environment? What happens if you click on this dataset?

Exploratory Data Anlysis

Goals:

- Review the basic language of data analysis.
- Review some challenges of data cleaning.
- Get exposed to the following functions:

- `select()`
- `filter()`
- `group_by()`
- `summarize()`
- `ggplot()`
- `geom_histogram()`
- `geom_boxplot()`

Keywords

Click on `form_data` in your *environment*. You should see something like this:

| | X | year | previous_mathstat | previous_stat | previous_r |
|----|----|------------|-------------------|-------------------------------------|--|
| 1 | 1 | Junior | 0 | Other high school statistics course | No, I have no programming experience! |
| 2 | 2 | Sophomore | 2 | Other high school statistics course | No, but I have taken other programming courses. |
| 3 | 3 | Senior | 5+ | I took | No, but I have programmed in other languages for fu... |
| 4 | 4 | Senior | 2 | Stat 101 at Williams | Yes! I have taken a course where we used R. |
| 5 | 5 | Senior | 1 | No formal course | Yes! I have taken a course where we used R. |
| 6 | 6 | Sophomore | 0 | AP Statistics | No, I have no programming experience! |
| 7 | 7 | Junior | 1 | Stat 101 at Williams | Yes! I have taken a course where we used R. |
| 8 | 8 | Senior | 1 | Stat 101 at Williams | Yes! I have taken a course where we used R. |
| 9 | 9 | Senior | 5+ | No formal course | No, but I have taken other programming courses. |
| 10 | 10 | First year | 0 | AP Statistics | No, I have no programming experience! |

With a neighbor, discuss the following questions:

- What are the **observational units** in this dataset. How many are there?
- What are some examples of **quantitative variables** in this dataset?
- What are some examples of **categorical variables** in this dataset?
 - Bonus: are any of these **ordinal variables**?

- Do you see any issues with the data?

When we do exploratory data analysis, we want to explore **univariate** summaries of our individual variables. We also want to explore **multivariate** relationships between our variables. We want to do this visually, but we may also want to compute some summary statistics! The type of visual and numerical summary depends on the type of variable! We will dive into some examples below.

One quantitative variable

Let's use "sleep_hours_summer," which stores the recorded answers to the question "How many hours do you sleep per night, on average, during the summer?," as our first example of a quantitative variable.

As a first step, we should always **look** at the data. Run the following command in your console to view this variable. What do you notice?

```
form_data$sleep_hours_summer
```

Since not everyone input their answer as a number, R is treating this as a categorical variable. This means that we will not be able to do things like compute the mean! Here are some takeaways from this finding:

- Design your surveys well! I let this happen by giving you a free response question.
- You now have a choice to make as a researcher about how to deal with this.
- Here, we are going to make an easy and yet potentially problematic choice.

Because I would really love to work with a numerical variable, we are just going to throw out all students whose data cannot be automatically converted to a numerical variable. Let's discuss the following three lines of code:

```
as.numeric(form_data$sleep_hours_summer)
is.na(as.numeric(form_data$sleep_hours_summer))
!is.na(as.numeric(form_data$sleep_hours_summer))
```

The following code first *mutates* our dataset, meaning that it adds a new variable. This new variable is named `sleep_hours_summer_numeric` and stores the output of `as.numeric(sleep_hours_summer)`.

```
form_data_clean <- form_data %>%
  mutate(sleep_hours_summer_numeric = as.numeric(sleep_hours_summer)) %>%
  filter(!is.na(sleep_hours_summer_numeric))
```

One categorical variable

One of the categorical variables in our dataset is `dining_hall`, which is from the question about which dining hall students prefer. A good way to visualize this data is with a bar plot.

```
ggplot(data=form_data_anonymous, aes(x=dining_hall))+geom_bar()
```

On your own, make a bar plot of another categorical variable in the dataset. Describe what you see!

One quantitative, one categorical

Variables in R can be more than just numbers! If there is time left in section, play around with the following lines of code in your console. You do not need to turn anything in for this part of the assignment, and if you don't get to it during section you do not need to worry about it.

First, let's explore what happens when we work with text in R. Does it make sense to add two variables when they are not numbers? Discuss!

```
x <- "cat"
y <- "dog"
x*y
```

In statistics, we often are not working with one number at a time. We are often working with lists of numbers; datasets! We can make our first small dataset in R using the `c()` function, which binds elements together in a list. Discuss the following lines of code as a group.

```
data <- c(1,5,7,8,9,10,12,13)
mean(data)
data+5
mean(data+5)
data[3]
```

Conclusion

Today, we performed some basic calculations in R. The primary purpose was to give you practice making and knitting an RMarkdown document. Next week, we will start working with real datasets in R using the **tidyverse** packages.

Acknowledgements

Much of the content in this tutorial was adopted from an OpenIntro lab.

This is a product of OpenIntro that is released under a Creative Commons Attribution-ShareAlike 3.0 Unported. This lab was adapted for OpenIntro by Andrew Bray and Mine Çetinkaya-Rundel from a lab written by Mark Hansen of UCLA Statistics.

This tutorial uses the OpenIntro formatting template.