An Organic R Textbook

Ian Cero, PhD MStat

2021

About

This is textbook created during live discussion with the Squeglia Research Group of the Medical University of South Carolina.

Use the **index on the left** to navigate from topic to topic.

Getting Started

Under most circumstances, getting started with R is a straightforward process of downloading and installing a few components. In what follows, we'll talk about what those components are, and the order in which you'll want to install them.

NOTE: the order of installation matters, so please be careful to follow the instructions in the order given below.

Overview of the R ecosystem

Most of the time you are using R for data analysis, you'll want to remember that you are working with a whole ecosystem of analysis tools. Understanding the different roles these tools serve in your project will help you keep track of the best way to use them - and hopefully make your R experience more intuitive.

The **R ecosystem** you'll be using for data analysis generally consists of three parts:

- The **R language**, which is a coding language (like Java or Python) that was optimized for talking to computers about statistical problems. When you download and install "R" (step 1, below), you are teaching your computer how to "speak" that R language.
- The **RStudio Integrated Development Environment (IDE)** is a program that you will use to make it easier to talk to your computer in the R language. Think of R as a language and RStudio as a chat app that has a bunch of features (e.g., your contacts list, spell check) that make the chat experience faster and easier for you.
- **R Packages** are collections of code that other people have written to make R perform particular tasks, usually around a them. For example, there are packages for making R perform new types of analyses, but also for streamlining data cleaning. You can download these packages with R's `install.packages()` command, so that your computer can use them

too. Think of packages like special tricks you are teaching your computer. Once it learns the trick (i.e., installs the package) it can do that new trick with R over and over again, making youre live a lot easier.

Installation

Step 1 - Download and install the R language

The first step to a functioning R ecosystem on your computer is to install the R language on your computer. It's freely available at the Comprehensive R Archive Network (CRAN), which is an acronym you'll see a lot as we go forward. CRAN is just a group of programmers in charge of maintaining and updating the R language.

To install R, go to `https://cran.r-project.org/`. Then at the very top of the page, choose the installer that is right for your operating system (i.e., Windows, macOS, Linux).

HINT: Depending on your operating system, the downloads page can be kind of intimidating. What you are looking for is the most updated version of R, which as of today (2021-12-01) is R 4.1.2. If you find that you want something to take you through the process at a more step-by-step pace, this tutorial (`https://www.datacamp.com/community/tutorials/installing-R-windows-mac-ubuntu`) should have an answer for each operating system.

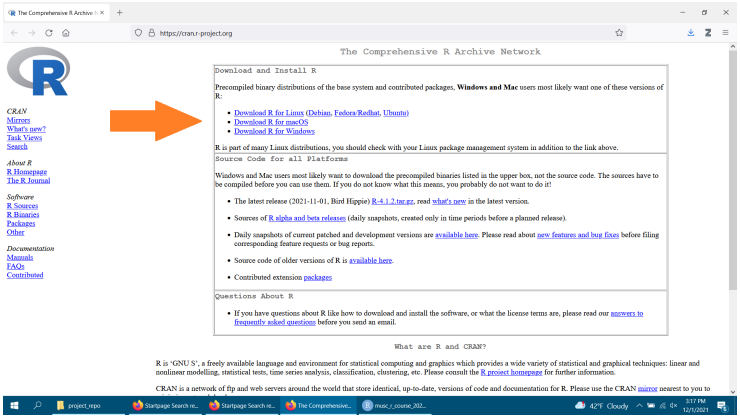

Figure 1: https://cran.r-project.org/

Step 2 - Download and install the RStudio IDE

Several years ago, writing code in R was especially difficult because there was so much to keep track of and it was all hidden behind the code. The RStudio IDE fixed that for us by allowing us to continue coding in R, but this time with a collection of useful windows that keep track of what's happening in our code (e.g., what datasets do we have loaded? what plots have we generated?).

After you installed R, installing the RStudio IDE should be fairly straightforward. Just go to their Downloads page (`https://www.rstudio.com/products/rstudio/download/`) and choose the **Desktop Version**

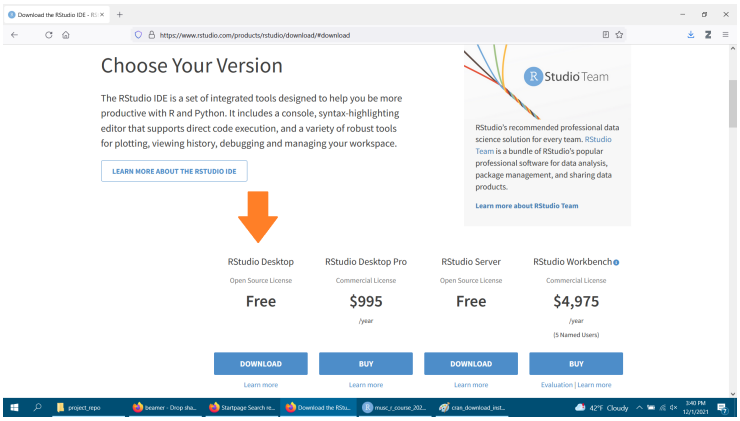

Figure 2: https://www.rstudio.com/products/rstudio/download/

NOTE: make sure you finish step 1 first! This will allow you to save several steps linking R and RStudio. This is because if R is installed first, RStudio will do the linking for you automatically.

Step 3 - Install the `tidyverse` package (optional)

Now that both R and RStudio are installed, let's open RStudio and install some packages.

1. Once you have Rstudio open, you should see several windows. Find the Console window.
2. Inside that window, type `install.packages('tidyverse')` and press ENTER.
    - R is case-sensitive, so make sure to type (or copy/paste) the command exactly.

- This should start an installation process that takes a few minutes (no more than 10) and will install a package you will use basically every time you program in R - so it's very useful to have.
- If you get an error message while installing, don't worry! That's pretty common and you've probably still done everything right. Just remind me in class and we will make sure to troubleshoot it for you.

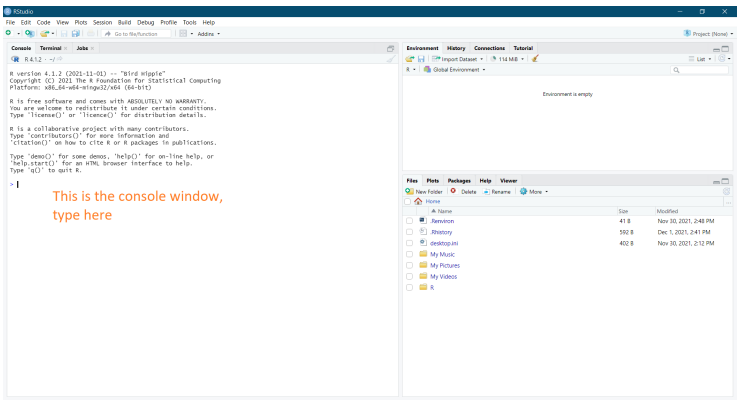

Figure 3: The RStudio IDE

A tour of RStudio

If R is a language, RStudio is a chat program that makes it easier to talk to your computer using that language. It includes multiple windows that help you keep track of the different parts of the conversation.

Although there are lots of tabs scattered throughout the overal RStudio application, there are generally 3 that we will use every day.

The console

Shown in the left half (or sometimes lower left quarter) of the screen. The console is where you can talk to R live. Everything you enter into the console happens right away, which makes it really useful for quick calculations.

The environment

The Environment tab in the top right quadrant shows you every object you currently have imported into R. This is especially useful for keeping track of

what you named your datasets (and whether your datasets even made it into R in the first place).

The lower right pane

There are many tabs in the lower right pane and you'll use most of them on a daily basis. The files tab shows you all the files in your current working directory (the file that R is paying attention to right now). The plots pane shows your plots, assuming you haven't told R to send them somewhere else. Lastly, the help pane will show you R's (very useful) help documentation, anytime you put a `?` in front of a command (e.g., `?lm()` brings up the help file for the `lm()` command).

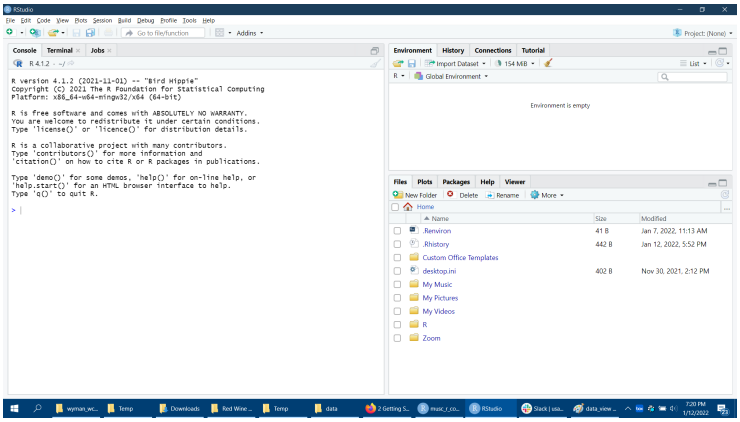

Figure 4: The RStudio IDE

Your very first analysis

To give us a roadmap for our future work in R, we'll start with a basic analysis here. For demonstration purposes, we'll be doing a basic analysis of red wine and checking whether its chemical properties predict how well it's rated by professional tasters.

Step 1 - download the data

The first step is simply to download the data, **which can be downloaded here.**

You might be asked to create a Kaggle account or to log in with Google. Don't worry though, it's totally free.

Step 2 - Make an RStudio Project

Now that we have our data, we need a place to store it - along with all the other important things we'll be working on, like our code and analysis output. The best option is to create an RStudio Project, which is a special kind of folder that RStudio knows to keep track of. RStudio projects have a number of advantages, but for now all you need to know is that they make it easier to keep track of your data.

To make a project...

1. Navigate to File » New Project (sometimes this takes a few seconds to load after you click on it)
2. Select New Directory
3. Select New Project
4. In the Directory Name text box, write the name for your project. In this case, a good name might be something like "Red Wine Practice."
    - Note, you can change the directory you want your project folder too, but it's not necessary for this example.
    - Leave all the remaining boxes (Create git repository, Use renv with this project) **unchecked**.
5. Click Create Project

With your project now created, you should now see "Red Wine Practice" (or whatever you named your project in the top of your RStudio application window). Moreover, if you look to the Files pane on the lower right, you should see a file called `Red Wine Practice.Rproj`. Lastly, you should notice that your working directory is now called "Red Wine Practice."

You can double-check this by typing `getwd()` (short for "get working directory") into the R Console on the bottom left and hit ENTER.

Step 3 - Get the data into your project folder

The quickest way to get your data into your project folder, is simply to copy/paste the `winequality-red.csv` you downloaded in Step 1 into your Red Wine Practice folder.

Where is that practice folder? Again, you can get the full path for your project folder simply by typing `getwd()` into the R console on the lower left and hitting ENTER.

To check whether your copy/paste operation worked, your can type `list.files()` into the R console. If it worked, you should see it listed along with your `Red Wine Practice.Rproj`

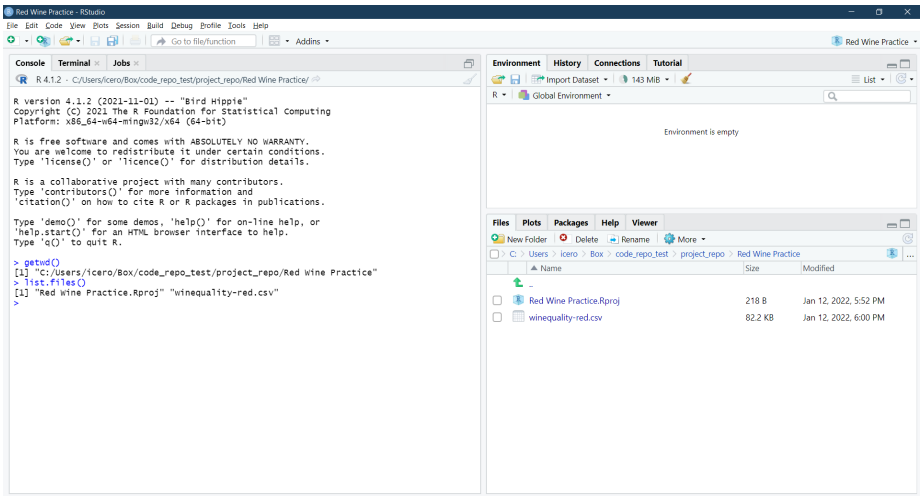

Figure 5: Checking the copy/paste operation worked with ‘list.files()‘

Step 4 - Open an Rmarkdown Notebook

We need a place to type our R commands, plus some notes to ourselves. The best way to do both of those things at the same time as an Rmarkdown notebook.

1. Navigate to File » New file » R Notebook
    - **NOTE**: You might get a message asking if you want to install some packages. Press OK / Yes - you *do* want to install them.
2. With the new notebook file opened, press CTRL+S to save the file under a different name. You can use whatever name you want. For this example, I will use `main.Rmd` to remind myself this is my main analysis file.
3. Inside your newly saved file, change the title from “R Notebook” to something more descriptive like “My first wine analysis.”
4. Lastly, RStudio gave us a bunch of boilerplate code. We won’t need that today, so delete everything below the second `---` at the top of the page, right under `output: html_notebook`. Your final document should then look something like the following.

Step 5 - Create a space to code

Rmarkdown documents have whitespace and greyspace. Whitespace is where you type notes to yourself. Greyspace is where you type R commands. We call these greyspaces **code blocks**.

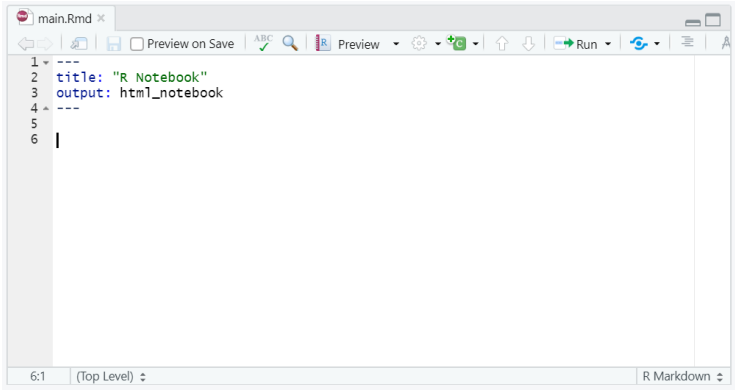

Figure 6: What your Rmarkdown file should look like before we start coding

1. Anywhere below line 4, type a note to yourself like "This is where I imported the data."
2. Move your curser to a line below that note you just wrote (e.g., line 8). Then, press CTRL+ALT+I. This will create a grey codeblock.

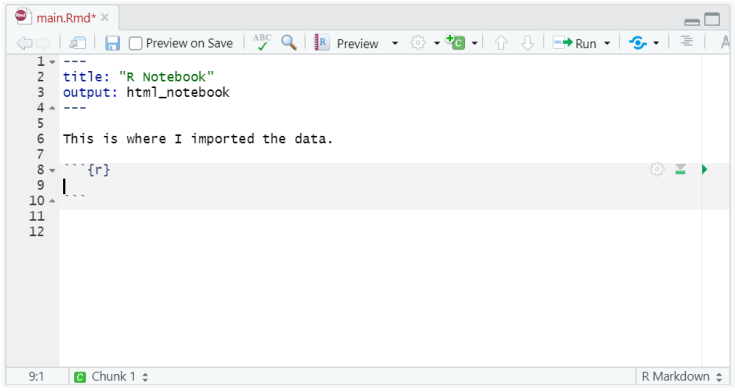

Figure 7: Adding a note to yourself and a grey code block

Step 6 - Write a command to import the data

Importing data involves four things: the name of the datafile, a command to read the data into R, the "assignment operator" (written as `<-`), and the name you want R to call the imported data when you reference it later. Fortunately, this *sounds* much more complicated than it is.

For now, just copy and paste the following commmand into the middle of your grey code block (for me, that is line 9).

```r
my_data <- read.csv('winequality-red.csv')
```

This is R code. It can be translated into English, but it's a little clunky. Also, generally code works from right to left. So, you would read this sentence as something like "Take the file called `winequality-red.csv`, read it into R, then call whatever comes out of that process `my_data`."

To get the R code to run, put your cursor inside the grey code block and press CTRL+SHIFT+ENTER. This runs all the code inside that block.

Step 7 - Look inside the data

Make a new code block a few lines down from the last one. Then type just the name of your imported data and run the block. You should see a preview of your dataset. To scroll through the other variables in the dataset, press the right-facing triangle on the right side of the preview.

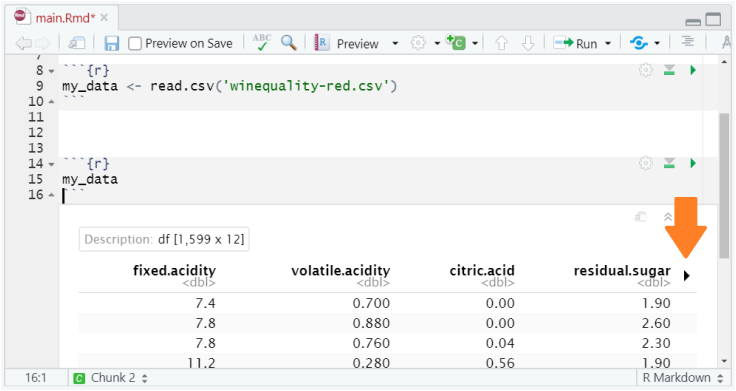

Figure 8: Preview of the data

Step 8 - Make a histogram

I think I'm most interested in the final variable in the dataset, `quality`. In this case, that is the quality of the wine - rated by professionals on a scale of 1 to 10. Let's see what the distribution looks like. For that we can use the `hist()` command (short for "histogram").

But what should we give our `hist()` command? We unfortunately can't give it the whole dataset. After all, we only want a histogram of one variable. How do we specify that variable? We use the "selection operator," which we write as a `$`-symbol, like below.

```
hist(my_data$quality)
```

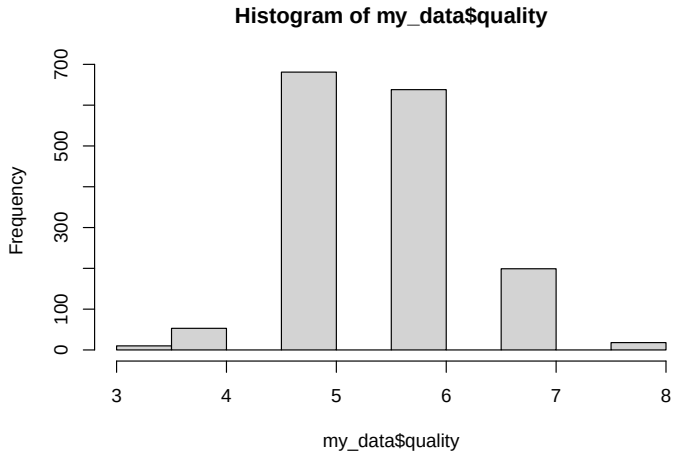

Figure 9: Histogram of wine quality ratings

Step 9 - Run a regression

Now that we have a sense of what our outcome variable (`quality`) looks like, let's see if we can investigate some of the chemical characteristics in wine associated with that outcome.

The ones that stick out to me are `pH` (acidity) and `alcohol` content because they seem like things that would really affect the taste. Let's use those as our predictors

To run a regression, we need just a few things (out of order): - The `lm()` command, which is short for "linear model." This is how R will know we want a regression. - Our data, named `my_data` - A regression formula, which tells R what the outcome variable and it's predictors are - The assignment operator again (`<-`), which tells us where to store the results - A name for where to store the results

Putting all of that together looks like this. One you have it all typed in (or copy/pasted), run the whole block with CTRL+SHIFT+ENTER.

```
my_results <- lm(
  formula = quality ~ pH + alcohol,
  data = my_data)
```

Step 10 - Get a summary of your results

You may have noticed that in Step 9, your results didn't show up anywhere after you ran your regression. That's because R stored them in `my_results`, just like it stored the outcome of the `read.csv()` command in `my_data`.

This often surprises people who come from other software packages, but that's okay. Our results are still easy to get. We just need to ask R for a summary of them.

```
summary(my_results)
```

```
##
## Call:
## lm(formula = quality ~ pH + alcohol, data = my_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7153 -0.4066 -0.1105  0.5076  2.4584
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.42581    0.38742  11.424  < 2e-16 ***
## pH          -0.85011    0.11571  -7.347 3.23e-13 ***
## alcohol      0.38617    0.01676  23.036  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6989 on 1596 degrees of freedom
## Multiple R-squared:  0.252,  Adjusted R-squared:  0.2511
## F-statistic: 268.9 on 2 and 1596 DF,  p-value: < 2.2e-16
```

This gives us a basic regression table with all of the same information you are used to. Under the *Coefficients* heading, we see the b-values / slopes (Estimate column), their standard errors, t-values, and p-values.

Next to the p-values, we see stars reminding us that our p-values are significant at the $< .001$ level. In fact, our p-values are so small, they have to be shown in scientific notation ("3.23e-13"). These can be treated as basically zero.

Lastly, at the very bottom, we also see the usual R-squared values (here, .252), our F-statistic, and degrees of freedom - everything we need to create a publication-ready regression table.

A Look forward

Over the last 10 steps, we ran through a basic version of essentially every R analysis you are likely to conduct in the future. This is example thus contains a useful workflow you will likely want to recreate in your future work:

- Create a project to store everything
- Import the data
- Visualize / explore the data
- Run main analysis model
- Summarize results

Although the data and models you use might be more complicated as your time in R progresses, it's helpful to remember that everything your are doing typically reduces to these fundamental steps.