In-class Lab 1

ECON 4223 (Prof. Tyler Ransom, U of Oklahoma)

January 17, 2019

The purpose of this in-class lab is to familiarize yourself with R and RStudio. The lab should be completed in your group. To get credit, upload your .R script to the appropriate place on Canvas.

First steps

1. Open RStudio on yours or a group member's laptop
2. Click File > New File > R script.

You will see a blank section of screen in the top-left of your RStudio window. This is where you will write your first R script.

Console

The bottom-left of the screen has a tab called "Console". This is basically a very fancy calculator.

Try the calculator by typing something like

```
2+2
```

Or even something fancier like

```
sqrt(pi)
```

Packages

R makes extensive use of third-party packages. We won't get into the details right now, but for this class, you will need to install a few of these. Installing packages is quite easy. Type the following two lines of code at the very top of your script:

```
install.packages("tidyverse", repos='http://cran.us.r-project.org')
install.packages("skimr", repos='http://cran.us.r-project.org')
install.packages("wooldridge", repos='http://cran.us.r-project.org')
```

You've just installed two packages. Basically, you've downloaded them onto your computer. Just like with other software on your computer, you only need to do the installation once. However, you still need to tell R that you will be using the packages. Add the following two lines of code to your script (below the first two lines you wrote). Notice how there are no quotation marks inside the parenthese this time.

```
library(tidyverse)
library(skimr)
library(wooldridge)
```

Running a script

To execute the script, click on the word "Source" in the top-right corner of the top-left window pane. This will take what is in your script and automatically send it to the console (as if you typed it directly into the console)

To save the script, click on the disk icon at the top of your script pane (but not the disk icon at the very top of RStudio). Name your script `ICL1_XYZ.R` where `XYZ` are your initials.

Commenting

Now, put a hashtag (#) in front of the first two lines of code in your script, like so:

```
#install.packages("tidyverse")
#install.packages("skimr")
#install.packages("wooldridge")
```

The hashtag is how you tell R not to run the code in your script. This is known as "commenting" your code.

At the very top of your script, type the names of your group members with a hashtag in front.

From now on, add all of the code you see to your script.

Exploring data

Now that you've got some of the basics of R, let's look at some data!

Loading data

We're going to load a data set from the `wooldridge` package. The data set is called `wage1`.

```
df <- as_tibble(wage1)
```

What we did there was convert it to a `tibble`, which is a nice format for data sets (see Ch. 10 of Wickham and Grolemund (2017)). We called the converted tibble `df`, but you can call it whatever you want: `mydata`, `data123`, whatever.

Browsing

If you re-run your script (by clicking `Source` ... or better yet, click on the little arrow next to `Source` which opens a menu that you can click `Source with echo`), you'll see something new in the "Environment" window (top-right). It says `df` under the "Data" heading.

Double-click on `df` in the Environment window. This will show you your data in a format similar to an Excel spreadsheet. You can use this to easily scan the data to make sure things look reasonable.

Summary statistics

Let's look at the summary statistics of one of our variables. Suppose we want to know: What is the average years of education in our sample?

```
skim(df$educ)
```

The `$` means that we are looking at the `educ` column in the tibble `df`.

In your script, write the value of the Mean of `educ`, preceded by a comment (the hashtag symbol)

What fraction of the sample is composed of women?

```r
mean(df$female)
# or
skim(df) #then look at 'female' column
# or
skim(df$female)
```

Visualization

Suppose you want to visualize the entire distribution of education. You would use the following code:

```r
ggplot(df, aes(educ)) + geom_histogram(binwidth=1)+theme_classic()
```

In a comment, write the most common value of education (the mode of the distribution) below the code in your script.

Repeat the previous two code snippets, but this time use the `wage` variable instead of the `educ` variable.

Creating a new variable

Suppose you want to add a new variable to `df`. For example, the wage variable is expressed in 1976 dollars, and you want to know what the wage would be in today's dollars. (Note: the CPI implies that $1.00 in 1976 equals $4.53 today)

```r
df <- df %>% mutate(realwage=wage*4.53)
summary(df$realwage)
```

You can verify that `realwage` got added to `df` by clicking on the preview of `df` and scrolling all the way over to the right.

For more information on `mutate()`, see section 5.5 of Wickham and Grolemund (2017). You can also drop a variable by typing `df <- df %>% mutate(realwage=NULL)`

Dropping observations

Suppose we want to get rid of all men in our data. (For example, maybe we're doing research on women's labor force participation.) To do this, we use the `filter()` function.

To use `filter()`, you provide conditions for *keeping* a specific observation. Add the following to your script:

```r
summary(df$female)
df <- df %>% filter(female==1)
summary(df$female)
```

We told R to keep observations where `female` equals 1. You can see that, prior to the `filter()` statement, women were 48% of the data. Now, they are 100%. We can thus verify that `filter()` did what we asked it to.

Missing values

A common occurrence in cross-sectional observational data is missing values. For example, someone leaves blank a question in a survey. Or the wage of someone who is unemployed is not defined. In R, missing values are stored as `NA` (meaning "not applicable"). To drop `NA` observations, use the `is.na` condtion:

```r
summary(df$wage)
df <- df %>% filter(!is.na(wage))
summary(df$wage)
```

In this case, we want R to keep *non-missing* wage observations, so we put `!` in front. `!` is generally accepted notation for logical negation, i.e. `!TRUE` equals `FALSE`.

References

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media. http://r4ds.had.co.nz.