In-Class Lab 1

ECON 425 (Justin Heflin, West Virginia University)

January 18, 2023

The purpose of this in-class lab is to familiarize yourself with R and RStudio. The lab may be completed as a group, but each student should turn in their own work. To get credit, upload your .R script to the appropriate place on eCampus ("In-Class Labs' ' folder).

First steps

1. Open RStudio on your laptop
2. Click File > New File > R script.
3. Save the script: Click File > Save As… > Name your script `ICL1_XYZ.R` where `XYZ` are your initials > Select the location where you want your script to be saved.

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 three lines of code at the very top of your script:

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

You've just installed three 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 three lines you wrote). Notice how there are no quotation marks inside the parentheses this time.

```
library(tidyverse)
library(modelsummary)
library(wooldridge)
```

Commenting

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

```
#install.packages("tidyverse")
#install.packages("modelsummary")
#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 your name with a hashtag in front.

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

Objects

We will often want to store results of calculations to resue them later. For this, we can work with basic **objects**. An object has a name and a content. We can freely choose the name of an object given certain rules. Objects have to start with a letter (lowercase or capital) and include only letters, numbers, and even some special characters such as "." and ''_''. *R is case senstitive, so x and X are different object names.*

The content of an object is assigned using `<-` which is supposed to resemble an arrow and is simply typed as the two characters ''less than" and ''minus". In order to assign the value 30 to the object y, type:

```
#generate object y
y <- 30
print(y); y
```

Typically, printing things in R is done with the `print()` function. However, if one is working in RStudio, just call the actual variable will print it to the console just the same.

A new object with the name `y` is created and has the value 30. From now on, we can use `y` in our calculations. We can perform various mathematical operations on our object as well. For instance,

```
#We can square the object y
y^2

#Add a number to it
y + 3
```

In your script, write the values you get from your two previous lines of code, preceded by a comment (the hashtag symbol)

Now, let's define another object `z` with the value of 5.

```
#Generate object z
z <- 5

#Generate a new object using our first two objects (y & z)
x <- y + z
```

A list of all currently defined objects can be obtained using `ls()`. In RStudio, it is also shown in the "Workspace" window (top right panel).

Vectors

Often times, for statistical calculations, we need to work with data sets including many numbers. The simplest way we can collect many numbers is called a vector in *R* terminology. To define a vector, we can collect different values using `c(value1, value2, value3,...)`. All the operators and functions used above can be used for vectors. Then they are applied to each of the elements separately.

```r
#Define a vector named b with the following numbers: 1, 2, 5, 8, 9, 12
b <- c(1, 2, 5, 8, 9, 12)

h <- b + 3
print(h); h
```

```
## [1]  4  5  8 11 12 15
```

```
## [1]  4  5  8 11 12 15
```

```r
k <- h + b

sqrt(k)
```

```
## [1] 2.236068 2.645751 3.605551 4.358899 4.582576 5.196152
```

```r
#When working with data in a data-set, it is sometimes helpful to know the number of
#elements, (numbers/observations) in a column/vector.
#For that, we can use the length() function

length(b)
```

```
## [1] 6
```

Special Type of Vectors

The contents of *R* do not need to be numeric. There are also `character` vectors. For handling character vectors, the contents simply need to be enclosed in quotation marks.

```r
states <- c("West Virginia", "Kentucky", "Virginia")
states
```

```
## [1] "West Virginia" "Kentucky"      "Virginia"
```

In your script, define a vector using three cities or counties of your choosing. Name the vector "Econ_425".

Another useful type of vector, is `logical` vectors. Essentially, this is a `TRUE` or `FALSE` where each element can only take on one of those two values.

```r
w <- c(5048, 4610, 4069, 4039, 3548, 2754, 2022) #NFL passing yards
sum(w) #The total number of passing yards by those seven quarterbacks
```

```
## [1] 26090
```

```r
q <- w<4000 | w >= 3500 #the "|" symbol means either is true or both are true
q
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```r
v <- w<4100 & w>= 3500 #the "&" symbol means both are true
v
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
```

Many economic variables of interest have a qualitative rather than a quantitative interpretation. They only take a finite set of values and the outcomes do not necessarily have a numerical meaning. Rather, they represent `qualitative` information. Think about gender, military rank, grade, major, marital status, product type or brand. In some of these examples, the order of the outcomes has a natural interpretation (such as grades), whereas others it does not (such as product type or brand).

Let's look at a specific example, end of semester survey, where you are asked to rate each professor on a scale between 1 and 5. Where 1 = "Strongly Disagree", 2 = "Disagree", 3 = "Neutral", 4 = "Agree", 5 = "Strongly Disagree". Or you could think about a company asking its customers to rate a product on a scale between 1 (="bad), 2 (="okay"), and 3 (="good").

We can store the responses of the ten customers in terms of the numbers 1, 2, and 3 in a vector. Now we could work directly with these numbered responses, but often, it is convenient to use so-called `factors`. One advantage is we can attach labels to the outcomes. Given a vector `x` with a finite set of values, a new factor `xf` can be generated using the command

```r
#Original ratings
x <- c(1, 2, 3, 3, 2, 1, 2, 2, 3, 1)
x
```

```
##  [1] 1 2 3 3 2 1 2 2 3 1
```

```r
xf <- factor(x, labels=c("bad", "okay", "good"))
xf
```

```
##  [1] bad  okay good good okay bad  okay okay good bad
## Levels: bad okay good
```

Naming & Indexing Vectors

The elements of a vector can be named which can increase the readability of the output. Given a vector `vec` and a string vector `namevec` of the same length, the names are attached to the vector elements using the following code

```r
names(vec) <- namevec
```

If we want to access a single element or a subset from a vector, we can work with indices. They are written in square brackets next to the vector name. For example, `myvector[4]` returns the 4^{th} element of `myvector` and `myvector[6] <- 8` changes the 6^{th} element to take the value 8.

```
#Create a vector of points per game of top five scorers in the NBA
points_per_game <- c(34, 33.5, 32, 30.9, 30.8)

#Create a string vector of player names
player_names <- c("Luka", "Joel", "Giannis", "Shai", "Jayson")

#Assign player names to vector and display vector
names(points_per_game) <- player_names
points_per_game
```

```
##    Luka    Joel Giannis    Shai  Jayson
##    34.0    33.5    32.0    30.9    30.8
```

```
#Indices by number
points_per_game[3]
```

```
## Giannis
##      32
```

```
points_per_game[2:4]
```

```
##    Joel Giannis    Shai
##    33.5    32.0    30.9
```

```
#Indices by name
points_per_game["Luka"]
```

```
## Luka
##   34
```

```
#Logical indices
points_per_game[points_per_game>=32]
```

```
##    Luka    Joel Giannis
##    34.0    33.5    32.0
```

##Matrices Matrices are important tools for econometric analyses. Let's look at three different ways to define a matrix object from scratch:

- **matrix(vec, nrow=m)** takes the numbers stored in vector *vec* and put them into a matrix with *m* rows.
- **rbind(r1, r2,…)** takes the vectors *r1, r2,…*, which should have the same length as the rows of a matrix
- **cbind(c1, c2,…)** takes the vectors *c1, c2,…*, which should have the same length as the columns of a matrix

```
#Generate matrix B from one vector with values
vec_b <- c(2, -9, 0, 3, 4, -3)

B <- matrix(vec_b, nrow = 2)
B
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    4
## [2,]   -9    3   -3
```

```
#Generate matrix B from two vectors corresponding to rows
r1 <- c(2, 0, 4)
r2 <- c(-9, 3, -3)

B <- rbind(r1, r2)
B
```

```
##    [,1] [,2] [,3]
## r1    2    0    4
## r2   -9    3   -3
```

```
#Generate matrix B from three vectors corresponding to columns
c1 <- c(2, -9)
c2 <- c(0, 3)
c3 <- c(4, -3)

B <- cbind(c1, c2, c3)
B
```

```
##      c1 c2 c3
## [1,]  2  0  4
## [2,] -9  3 -3
```

```
#Giving names to rows and columns
colnames(B) <- c("Year", "Tax Revenue", "Population")
rownames(B) <- c("West Virginia", "Virginia")
B
```

```
##               Year Tax Revenue Population
## West Virginia    2           0          4
## Virginia        -9           3         -3
```

```
#Indexing for extracting elements, still using the B matrix created above
#Allows us to access a subset of matrix elements, returns the element in row 2, column 1
B[2,1] #the first number (2) refers to the row number,
```

```
## [1] -9
```

```
#the second number refers (1) refers to the column number.

B[,2] #this line returns everything stored in both rows for the second column
```

```
## West Virginia      Virginia
##             0             3
```

```
B[,c(1,3)] #this line returns everything in both rows for the first and third column
```

```
##                Year Population
## West Virginia    2          4
## Virginia        -9         -3
```

```
B[2,] #returns a vector consisting of the elements in row 2, all columns
```

```
##        Year Tax Revenue  Population
##          -9           3          -3
```