Introduction to R

Economics 3818, Fall 2018

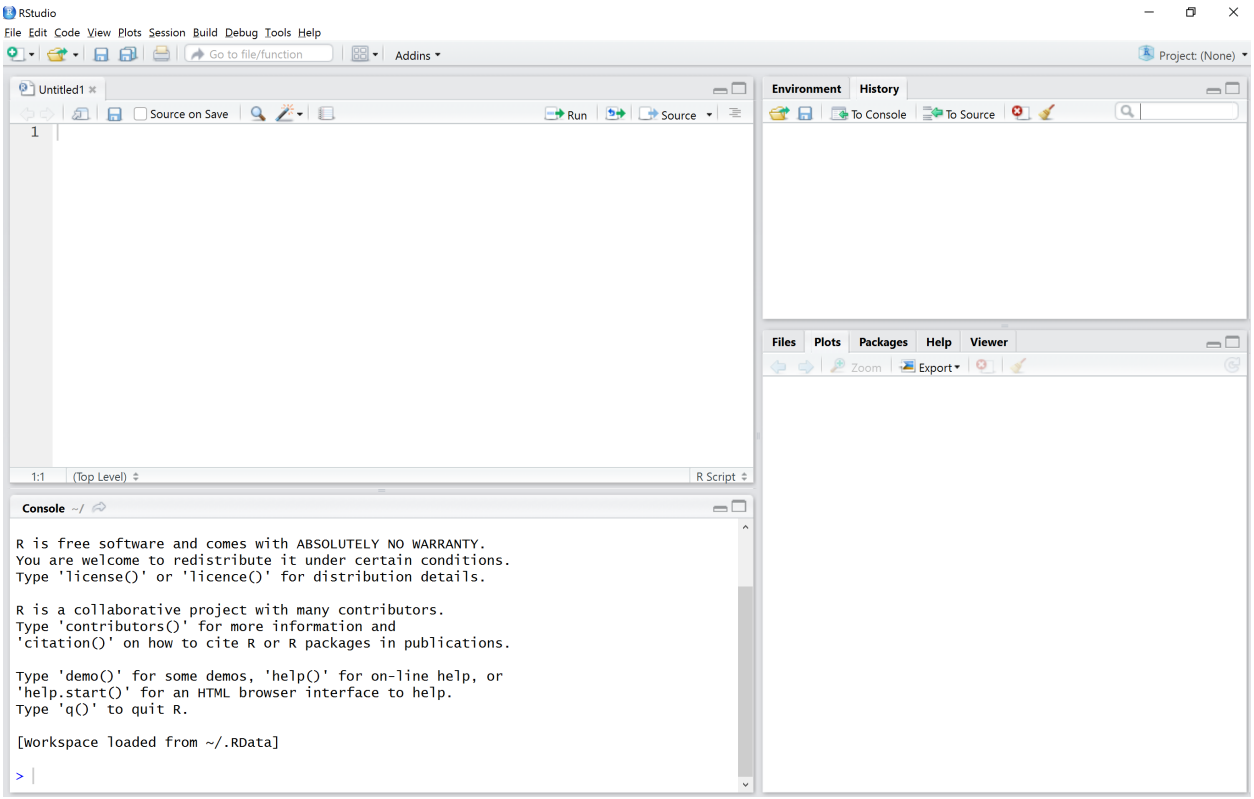
Installation

R is a free statistical programming language used frequently in industry. We will use the user interface Rstudio to simplify some of the work flow.

First, go to the following clickable address [https://cran.rstudio.com/] to download the base distribution for the appropriate operating system. Run the downloaded file and follow the installation instructions.

Next, install Rstudio for the appropriate operating system from the following address [https://www.rstudio.com/products/rstudio/download/]. You want the free desktop version. Again download the file and follow the installation instructions.

You should be now be able to open Rstudio, see something like this, and be set to go!



The Rstudio interface

Rstudio is broken up into four different panes. The top right is where you can access data created within your current environment and the history of what you have done. The bottom right allows you to browse files, look at plots, and read help files. The bottom left hosts the console, where any commands executed by R display their output and where you can run commands line by line. The top left is where you can write scripts. Scripts allow you to save a series of commands and run them at a later date or all at once.

Some R command basics

For now, we will only use the console. Later on we'll write scripts.

R as a calculator

We can use R just like any calculator, note that the order of operation matters! If you type they grey lines into the console, you should get the number returned.

```
2+3
```

```
[1] 5
```

```
(2+4)/7
```

```
[1] 0.8571429
```

```
2+4/7
```

```
[1] 2.571429
```

```
2^4
```

```
[1] 16
```

We can even use the modulus operator which returns the remainder after division.

```
14%%3
```

```
[1] 2
```

R as a functional programming language

For more complicated operations, R acts a functional programming language. For example here we take find $\sqrt{4}$ and $4!$.

```
sqrt(4)
```

```
[1] 2
```

```
factorial(4)
```

```
[1] 24
```

Help in R

Fortunately, R has detailed and thorough internal documentation. If you have any question on what a function does, or what type of arguments it can take, you can always type in `help("function")` or `?funtion` into the console. The details should show up in the bottom right pane. Go ahead and try out the command below.

```
help("mean")
```

You should notice in the help pane that the function `mean()` can take more than one argument. The additional arguments that can be passed through a function makes R very flexible.

Creating varaibles

A lot of time we want to save the value of some calculation, to be used later. We do this by assigning values to variables. This can be done with the `<-` or `=` operator. Here I create one variable called x and one called y.

```
x<-2
y=5
```

If you look in the upper right hand corner, you should see these variables in the environment tab. Now R will recognize these values as the number that you have assigned them.

```
x+y
```

```
[1] 7
```

Note that capitalization matters!

```
x+Y
```

```
Error in eval(expr, envir, enclos): object 'Y' not found
```

We can name variables anything we want.

```
buffalo <- 10; water <- 4; last <- 7;
nonsense <- buffalo*water - last;
nonsense
```

```
[1] 33
```

Non-scalar values

Vectors

Usually (almost always), we are interested in a bunch of numbers. R can hold non scalar values in vectors, matrices, lists, and data frames. The following creates two numeric vectors of equal length. The `c()` is the concatenate function, linking the numbers in a sequence.

```
x <- c(1,2,4,4,5,5)
y <- c(4,5,3,4,3,2)
x;y;
```

```
[1] 1 2 4 4 5 5
```

```
[1] 4 5 3 4 3 2
```

It is possible to index elements from the vector. Suppose I want to look at the second number in the `x` vector and the fourth number in the `y` vector, then take their product.

```
x[2];y[4];
```

```
[1] 2
```

```
[1] 4
```

```
x[2]*y[4];
```

```
[1] 8
```

A lot of standard operators work with vectors on an element by element basis. For example

```
x+y
```

```
[1] 5 7 7 8 8 7
```

```
sqrt(x)
```

```
[1] 1.000000 1.414214 2.000000 2.000000 2.236068 2.236068
```

We can perform a lot of functions on vector.

```
sum(x)
```

```
[1] 21
```

```
mean(x)
```

```
[1] 3.5
```

```
median(x)
```

```
[1] 4
```

```
sd(x)
```

```
[1] 1.643168
```

```
summary(x)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    2.50    4.00    3.50    4.75    5.00
```

```
IQR(x)
```

```
[1] 2.25
```

```
min(x)
```

```
[1] 1
```

```
max(x)
```

```
[1] 5
```

The : operator

Sometimes you are interested in a sequence of numbers that have a constant difference, such as `4,5,6,7,8`. In such a scenario you can use the `:` operator to create a vector of this sequence.

```
seq <- 1:4;
seq;
```

```
[1] 1 2 3 4
```

Matrices

Matrices are very useful, but we won't use them that much here. For example, here I make a six element vector into a 2 row by 3 column matrix.

```
mat <- matrix(x, nrow=2, ncol=3)
mat
```

```
     [,1] [,2] [,3]
[1,]    1    4    5
[2,]    2    4    5
```

R allows for a number of matrix operations including inverse, transpose, and Kronecker product.

Non-numeric values

R can also accept string and logical arguments. Note that when I assign the value to x, R forgets that it used to be a vector. `mode()` helps figure whether objects are strings (character), numeric, or logical.

```
x<-"hello world"
y<-TRUE
mode(x)
```

```
[1] "character"
```

```
mode(y)
```

```
[1] "logical"
```

Combining different data types within the same vector doesn't work like you want. Here the vector z becomes character although y is logical.

```
z<-c(x,y)
mode(z)
```

```
[1] "character"
```

Logical vector as an index

Suppose we are interested in a subset of a vector that meets some condition. For example, we have a vector of all multiples of 3 between 3 and 24, inclusive.

```
x<-3*1:8;
```

We are interested in all of the even numbers within this vector. We can create a logical statement for when this is true. Notice for logical statements we use two equal signs instead of one.

```
x%%2==0;
```

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

We can use the logical vector as an index to return all

```
x[x%%2==0]
```

```
[1]  6 12 18 24
```

Data frames

Data frames are very useful and is one of the strengths or R. It allows for you to store values of different types (string, logical, numeric) within the same object. Additionally, it is formatted in a way that is intuitive with data. Each row is a single observation and each column is a variable. Here I create a string vector and put it into a data frame with our vectors x and y.

```
x<-c(1,2,3,2,4,3)
y<-c(2,3,1,5,6,4)
z<-c("Jan", "Feb","March","April","May","June")
df <- data.frame(month = z, price=x, quantity=y)
df
```

```
  month price quantity
1   Jan     1        2
2   Feb     2        3
```

```
3 March     3        1
4 April     2        5
5   May     4        6
6  June     3        4
```

Data frames are very easy to subset. For a data frame called `df` and a variable called `month`, I can access the variable month by typing `df$month`

```
df$month
```

```
[1] Jan   Feb   March April May   June
Levels: April Feb Jan June March May
```

```
summary(df$price)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.0     2.0     2.5     2.5     3.0     4.0
```

Suppose I want to know what the price is in March. I can use a logical index that satisfies what I am looking for.

```
df$price[df$month=="March"]
```

```
[1] 3
```

Alternatively

```
df[df$month=="March","price"]
```

```
[1] 3
```

Also, there is a great function called `subset()` that you are welcome to look into.

Using actual Data

Finally, let's load some data from the web. I want to first clean my work space using the `rm()` command. Supplying `list=ls()` as an argument will delete all your vectors, matrices, dataframes, and lists.

```
rm(list=ls())
```

Loading the data

The data are on my personal webpage as a **c**omma **s**eperated **v**alues (.csv) file. This file type is a simple and common way to store data, readable by most statistical applications.
The data come from Harrison and Rubinfeld (1978), a paper that tries to quantify the value of clean air. The details on the variables can be found here: [http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.names]

We will use the `read.csv()` function with the data file URL. This function can also be used to load data that is stored locally, on your personal computer or network. Make sure that you are connected to the internet for this.

```
housing_df<-read.csv("https://mattbutner.github.io/data/housing_df.csv")
```

From here we can summarize the data, makes plots, run hypothesis tests, etc.

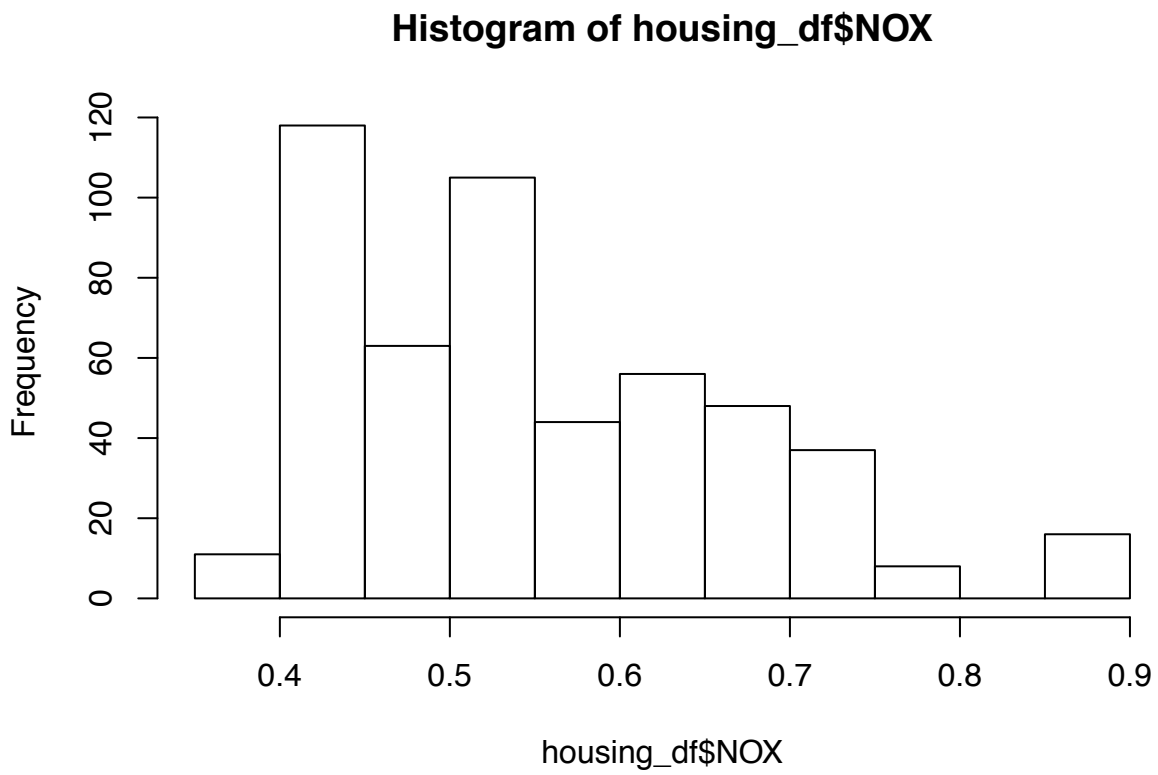
```
summary(housing_df$MEDV)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   5.00   17.02   21.20   22.53   25.00   50.00
```

Plotting Data

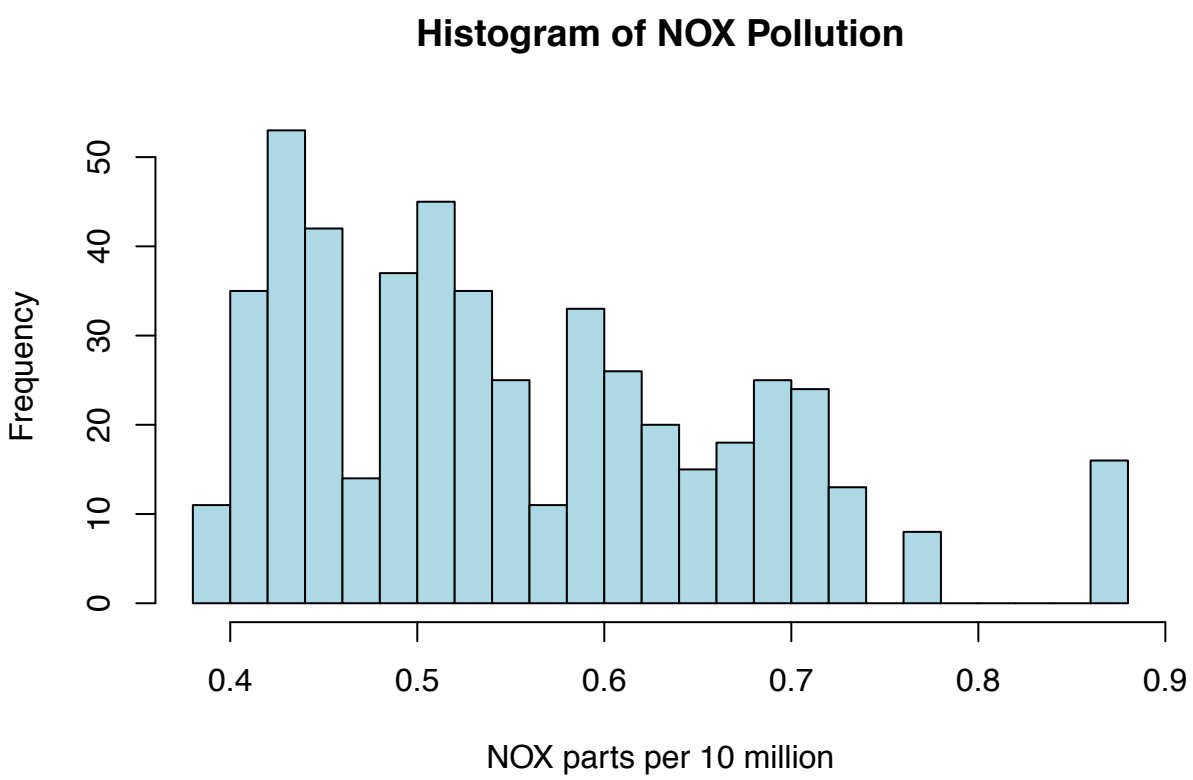
There are a number of ways to plot data in R. The base distribution has the functions `plot`, `hist`, and `pie`. Because R is opensource, there are free packages you can install in R to make more sophisticated graphics. ggplot2 is probably the most well known visualization package. We'll just use `hist` for now

```
hist(housing_df$NOX);
```



We can add arguments to make this look better.

```
hist(housing_df$NOX, main = "Histogram of NOX Pollution",
     xlab = "NOX parts per 10 million", col="lightblue", breaks=20);
```



References

Harrison, David, and Daniel L. Rubinfeld. "Hedonic housing prices and the demand for clean air." Journal of environmental economics and management 5.1 (1978): 81-102.