Introduction to R for Stata Users

Day 1

Laura Garcia Montoya

6/25/2019

0. Before we begin:

Instructor: Laura García Montoya
Email: lauragarciam@u.northwestern.edu

Downloading and Installing R: Go to: https://workshops.rcs.northwestern.edu/install/r/

Website with documents:
https://github.com/lauragarciamontoya/Introduction-to-R-for-Stata-Users

- Introduction
- How is this workshop going to work? R markdown Color Code
- Goals for this workshop:
  - Intro to R
  - Logic of objects
  - Main differences with Stata
  - Foundation to learn on your own

1. Introduction: What is R? and R Studio?

R

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.

RStudio

RStudio makes R easier to use. It is an integrated development environment for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

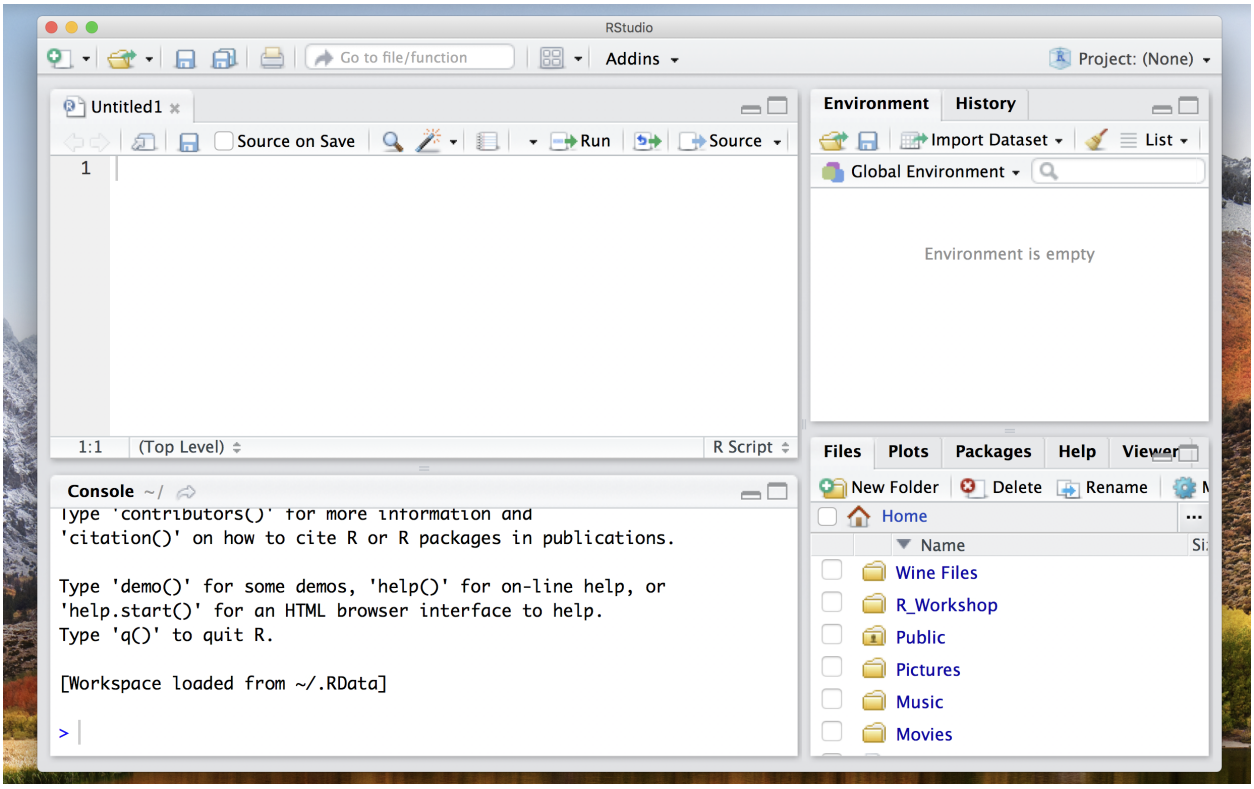
Why should I use R - if I already know Stata?

- IT IS FREE!!
- Much more internet documentation in R.

- Very flexible - easy to create functions/libraries/tools. R has a much more feature rich programming language.
- R has much better graphical tools (ggplot2)
- R can hold multiple data sets.
- R has better tooling to query other databases, parse JSON data, scrape web sites, etc. and has way more tools and packages - text analysis, machine learning.
- R markdown
- BUT, it is harder to learn.
- BUT, there are some things that are harder to do. For example, estimating robust standard errors is a pain. Coefficient plots.
- AND: YOU CAN USE BOTH! I USE THEM FOR DIFFERENT PURPOSES.

Using R Studio

Once you launch the R Studio App this is how it should look like:



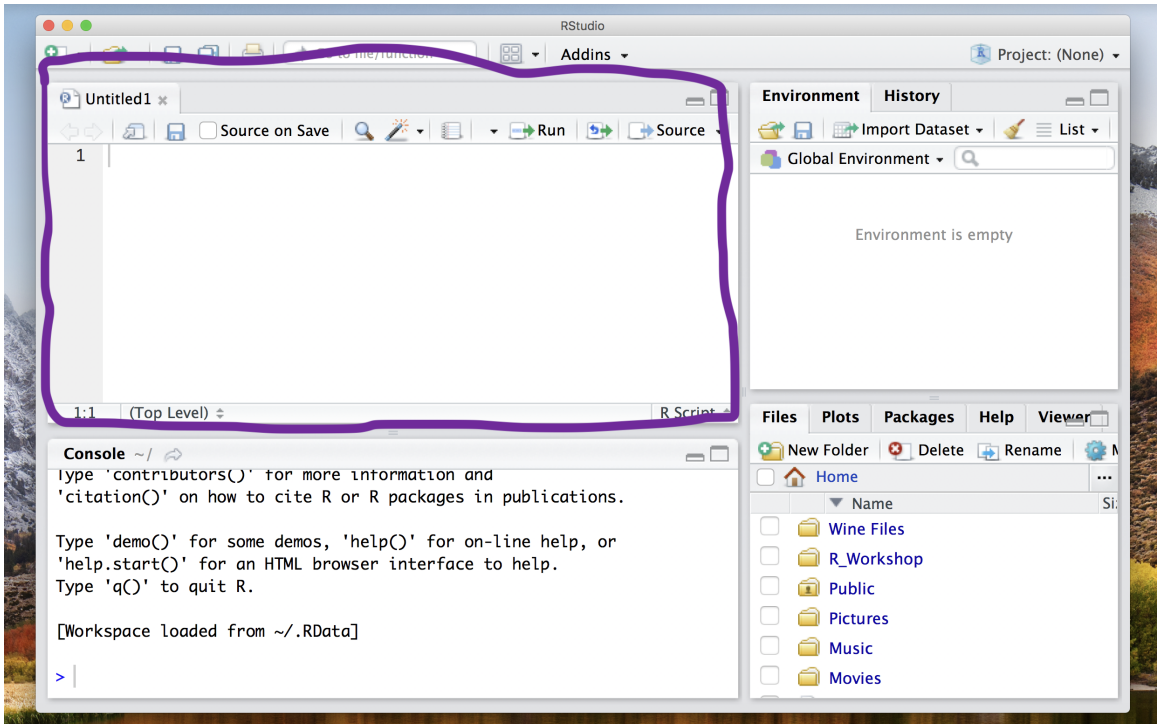
[STOP: ARE YOU ACTUALLY SEEING THIS?]

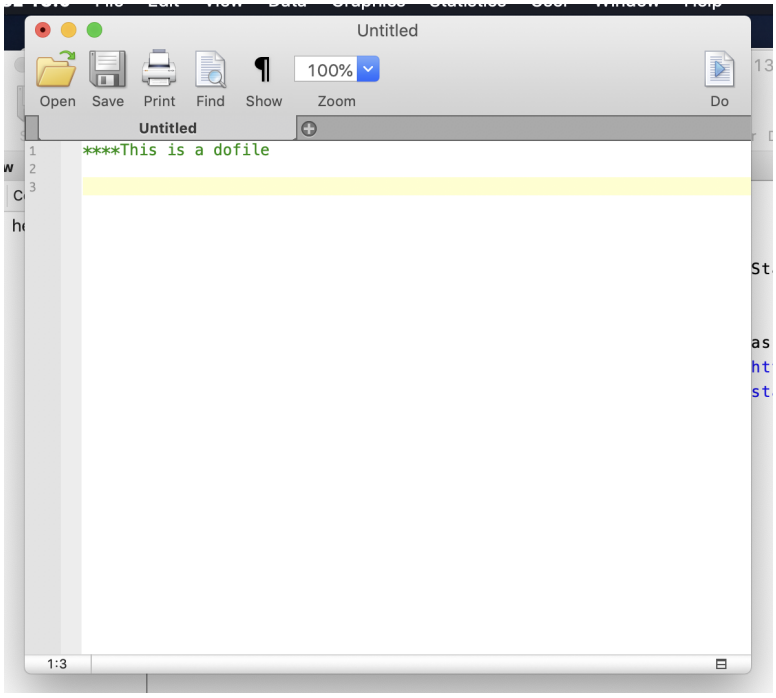
If you have not done so, type or copy in the console:

```
install.packages(c("tidyverse", "car","cowplot",
                   "foreign", "broom", "gapminder"),
                 repos="http://cran.rstudio.com")
```

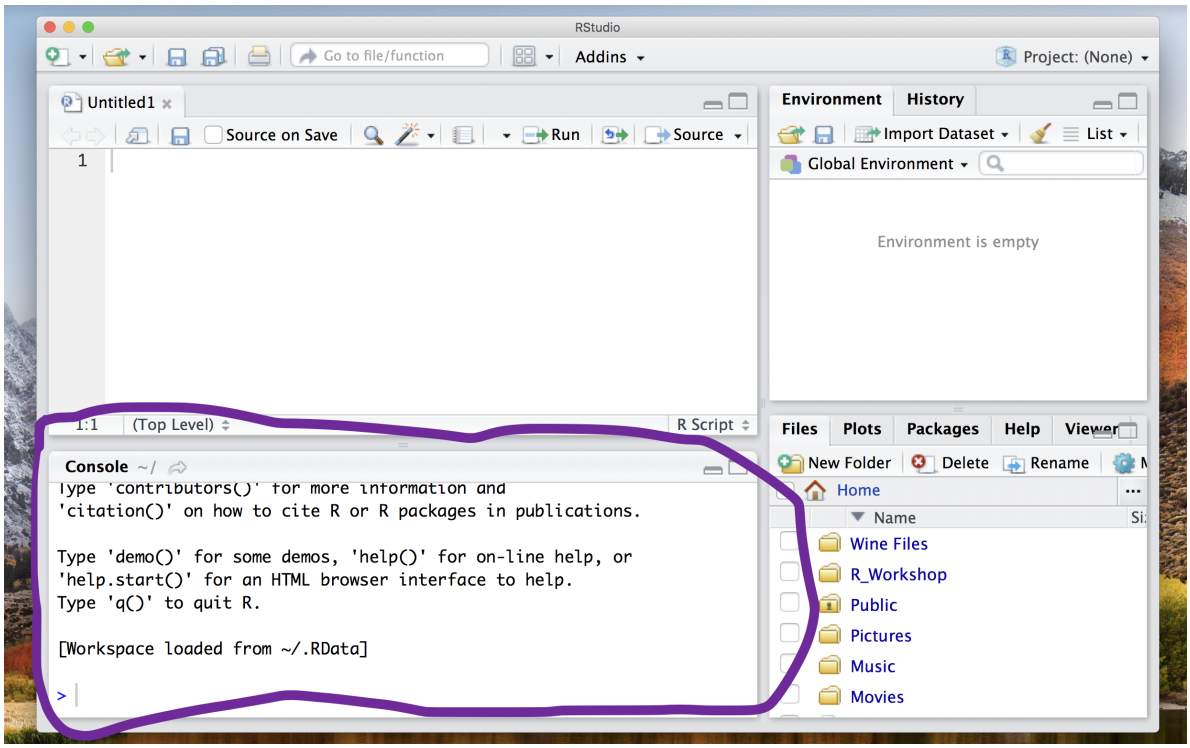
The parts

1. **The Editor:** The top left quadrant is the editor. This is where you write R code you want to save and use later. **In Stata**: This is the equivalent to the Do-file.

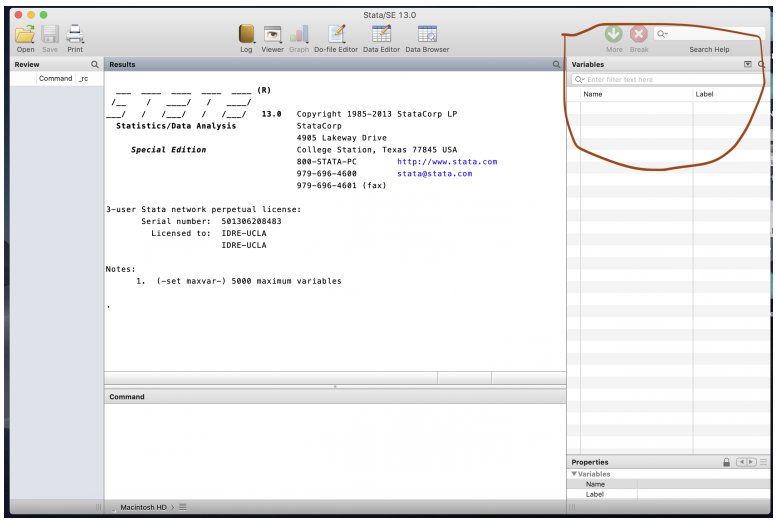




2. **The console:** You can find the console in the lower left quadrant of R Studio. The console is an interactive computer programming environment for R. It takes user inputs/commands, evaluates them, and returns the result to the user.



In Stata: This is the equivalent to both the command line and the results window in Stata.

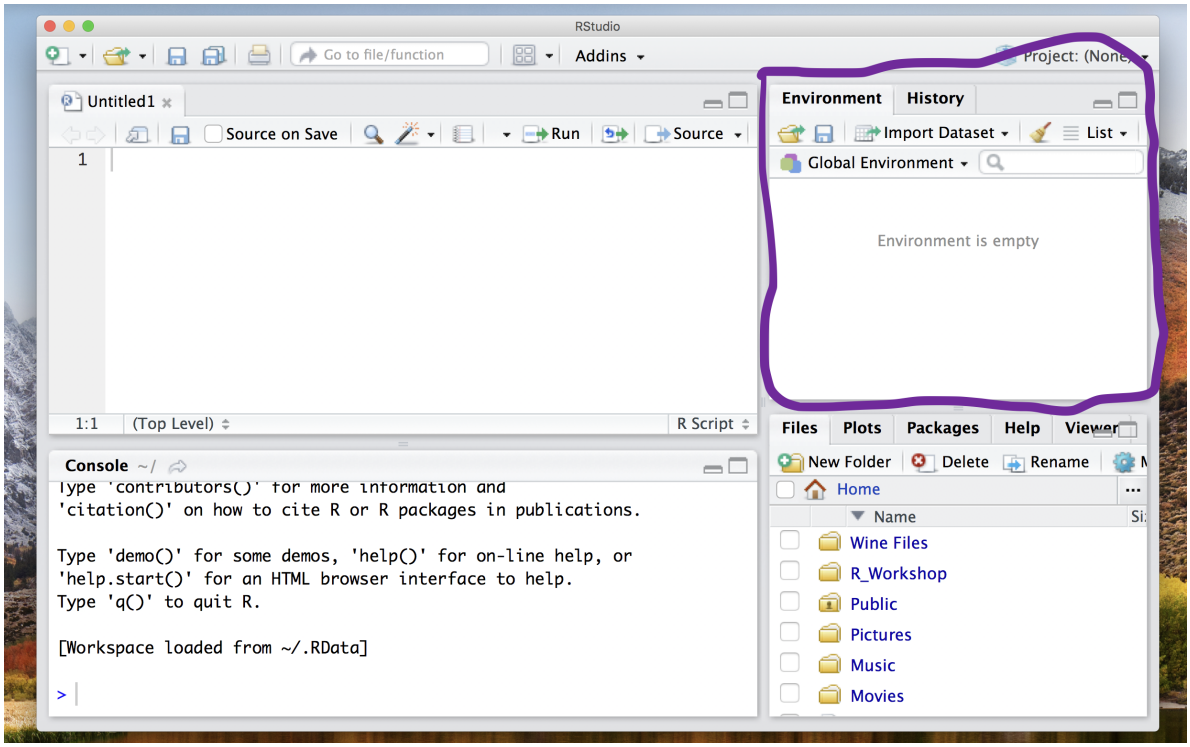


3. **Environment/History:** The upper right corner is composed by two tabs.

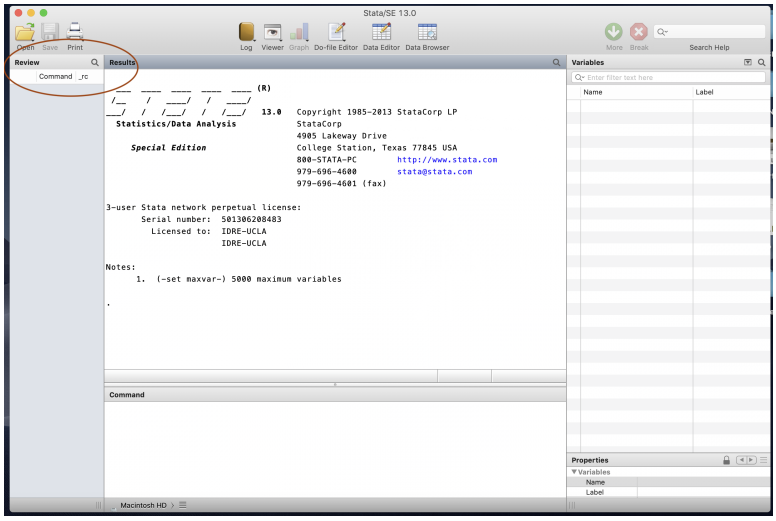
   Environment: Here you can see a list of all the objects you defined in the console. That is: functions, variables, datasets, values. You can also use it to import custom datasets manually and make them instantly available in the console.

   You can also inspect the environment of other packages you installed and loaded (more on packages at a later time). Play around with it - you can't break anything.

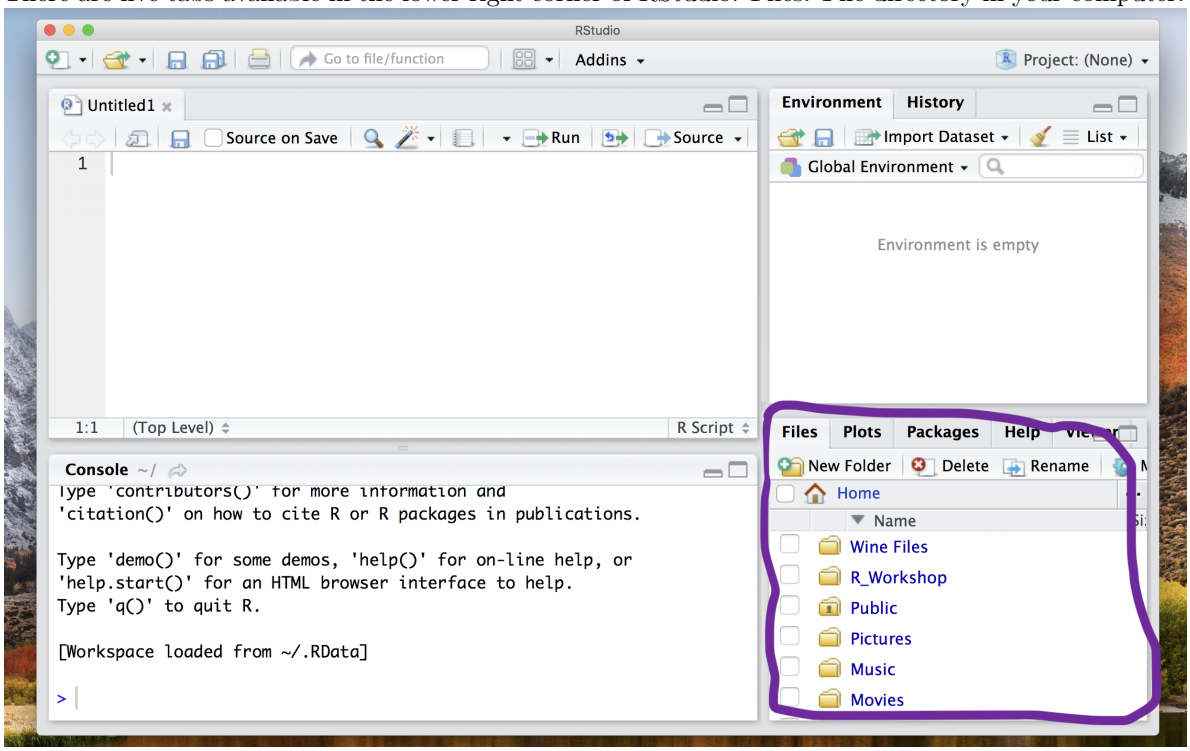
   In Stata: This does not have an equivalent in Stata because Stata is not object based. However, the variables windowm displays the variables in a similar way in which the Environment can display the variables inside a dataset.



   History: Lists every single console command you executed since the last project started. It is saved into a hidden .Rhistory file in your projects folder. You can choose not to save your environment after a session.
   In Stata: This is the equivalent to the Review window in Stata.



4. **Miscellaneous:**
There are five tabs available in the lower right corner of RStudio. Files: File directory in your computer.



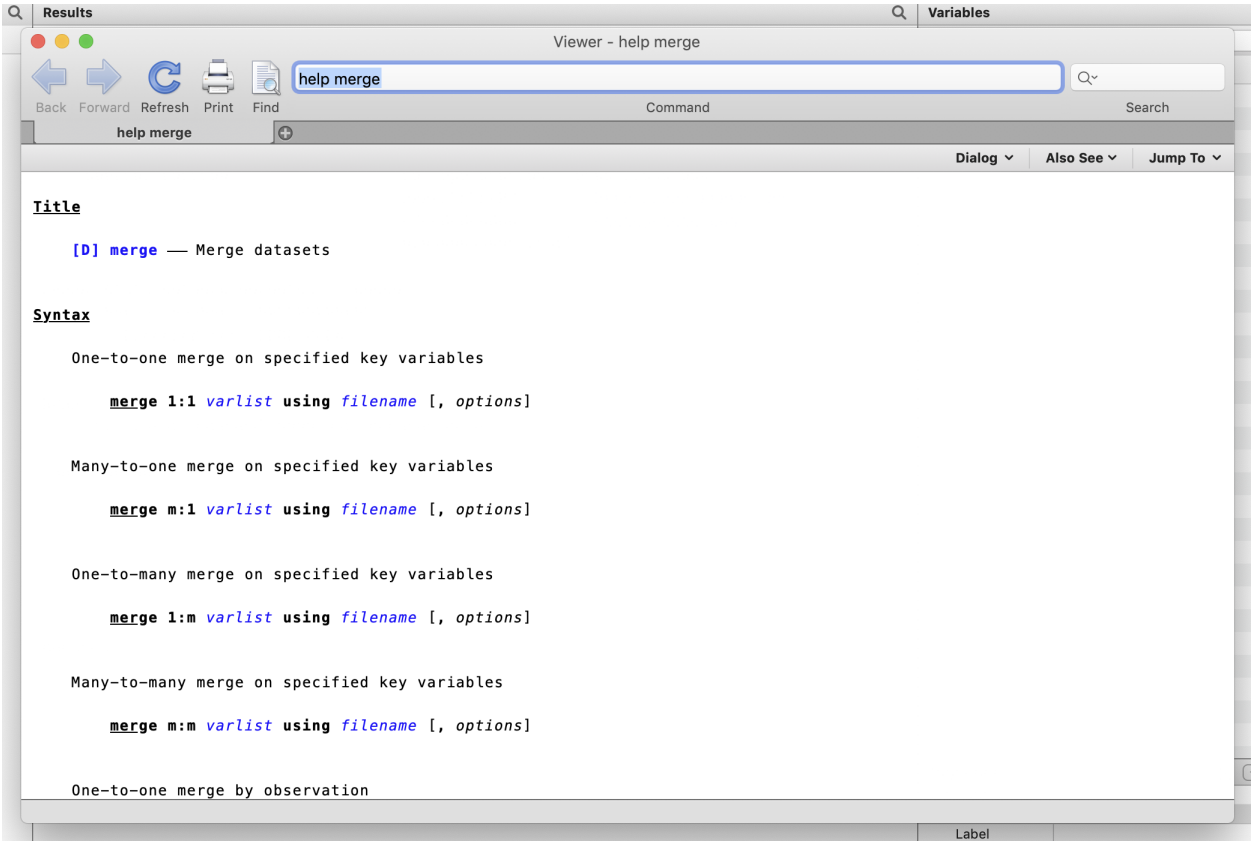
Plots: Displays the plots you produce, you can edit them and save them as images.

Packages: Useful to install and use packages. (WE WILL LEARN WHAT THIS MEANS SOON)

Viewer: Built-in browser. Not very used for the purposes of this workshop

Help: Contains the help directory and displays the help file when you ask for questions. This window is extremely useful as you will notice soon.

In Stata: The help window that pops up in Stata works very similar to the Help tab in R Studio.



[STOP: QUESTIONS SO FAR?]

Basic Functions in the console

Calculator: You can use the console as a calculator. Give it a try, you just need to type what you want to calculate.

The gray box with a different font you see below, means that this is code - R. The phrases preceded with # are the instruction and the rest has the format to be typed on the console.

In Stata: This use of the console is equivalent to typing "display" followed by functions in Stata.

```
#Add 2 plus 2
2+2
```

```
## [1] 4
```

```
#Multiply 2 and 3 together
2*3
```

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

```
#Calculate the log of 10.
log(10)
```

```
## [1] 2.302585
```

```
#2+2*(3+2)
2+2*(3+2)
```

```
## [1] 12
```

```
#What is the maximum number between 2,3,4?
max(2,3,4)
```

```
## [1] 4
```

Test logical statements: Using comparison and logical Operators you can test if statements are TRUE OR FALSE.

```
#Is 1 smaller than 2?
1<2
```

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

```
#2 is smaller than 1, right?
2<1
```

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

```
#1 is larger than 0 and 2 is larger than 0
1>0&2>0
```

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

```
#One is larger than 0 or -1 is larger than 0
1>0|-1>0
```

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

```
#One is smaller than 0 or -2 is smaller than 0
1<0 |-2<0
```

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

```
#One is smaller than 0 or 2 is smaller than 0
1<0|2<0
```

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

Getting Help / R documentation

Inside R: You can type ? in the console followed by the name of a function to get documentation of that function, the help file it will be displayed in the help window.

HELP: just type ? followed by text. In Stata: Equivalent to typing help or h in the Ccommand line.

?function_name opens the help for function_name(). If you know the name of the function, this will tell you how it works and how you can control its operation.[1]

In Google: "In R how to add new column in data.frame"

Stack Overflow: There is a large community on the internet asking and answering questions in R. When you google a problem your are facing with R you will most likely see stack overflow, it is a reliable source of answers.

There is a vast amount of resources online for R users, and this is one of its main advantages. Many people around the world are using and it is very likely that the same errors have occurred to more than one.

To keep in mind

- There are multiple ways to do almost everything in R
- Googling "how to ... in R" is necessary and very common for most people

Practice Exercises

1. Using the console
   a. Calculate square root of 109090
   b. Add 3 and 5 together.
   c. What number is larger: The log of 2000 or the square root of 51? (Try to do this in one line only.)
   d. What is the maximum number between: the square root of 200, seven times 2, and log of 3000. (Try to do this in one line in the console)

2. Google to find guidance/functions to: a. calculate standard deviation b. make a histogram of a variable c. import excel file

3. Download and save the documentation of the package ggplot2. (Hint: Use ?)

4. Find out what is wrong with the following line of code: install.package(car) (Hint: type it down on the console)

[STOP HERE, QUESTIONS?]

[1] Caveat: The documentation isn't always aimed at beginners since it requires terminology that you don't know YET. You can always use google to trouble shoot.

2. Introduction to Programming

Programming in R means writing commands or giving instructions you want your computer to execute. For now, we are going to give the commands directly in the console. Later on, we will learn how to use a script to write and run code. The advantages of using a script are being able to run your code more than once and replicating it in the future. The intuition is the same for Stata's do files. I don't anticipate problems here for you .

One could say that R is composed by **functions and objects**. Examples or objects include vectors, matrices, dataframes, figures. The concept *function* here means the same as in the Math Camp. In R, you use functions to create, transform or delete objects.

NOTE: The object based logic in R is very different from Stata - understanding what this difference means will get you a long way in learning R

Structure of the code:

Coding is then "writing" functions to create, transform, delete, combine objects.

Objects

Creating New Objects: Using "<-" in the console allows you to create and save objects. For example:

```
#Create the object "r" that is equal to "a".
r<-'a'

#Create the object "name" that is equal to "Laura Garcia Montoya"
name<-'Laura Garcia Montoya'
x<-2
y<-3
```

Names of objects can be anything as long as they start with a letter and do not contain spaces. Developing conventions to name objects is a good coding practice. For example separating words with `_'' or`.".

```
#Create a vector with the values of GDP for 4 countries.
country_gdp<-c(18,23,5,20)

#This means that the first country has a GDP of 1,
#the second a GDP of 23, the third one a GDP of 5 and the fourth a 20.
#(FAKE DATA - MEANINGLESS)
```

Example

```
#Create an object with name "age" assigning the value of your age in the console.
age<-29

#Then, type "age" in the console and see what you get.

#Create an object named "program" and assign the value.
#In this case we are dealing with text instead of numbers. (HINT Use "")
program<-"Political Science"
```

Types of Objects:

There are many types of objects. We will talk more about this later, but for now you should know objects could be: Single values, Vectors, Lists, Matrices, Data Frames.

OBJECTS: KEY DIFFERENCE WITH STATA

"In Stata, everything is an action. You regress one variable against another. In R, everything is an object." In Stata, the environment is the data and eveything else we do, we are doing to the data. Recode variables, add new variables, rename variables, sort the data, drop observations from the data. We sometimes "collapse" the dataset into a broader level of aggregation. One exception to this "action" based logic are the store estimates or scalar in locals that we save, but these are usually temporary.

In R, on the other hand, instead of *regressing* Y *on* BX, like you do in Stata, you create a linear model. Another example is the difference between tabulating a variable in Stata and creating a table in R. In Stata the commands are verbs, in R they are nouns.

This logic is what allows you in R to have multiple datasets open at the same time. Opening a dataset in R means that you are creating an object of the type dataframe. More on this later. [2]

Functions

Functions take multiple input objects and return output objects. We have seen some functions already. For example sqrt(), log(), max().

The logic of functions in very similar in both programs.

Common functions:

```
# To calculate mean, mean()
mean(c(2,3,4))
```

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

```
#To add values
sum(c(2,3,40000,3020))
```

```
## [1] 43025
```

```
#Print, prints the value of the object.
print(mean(c(2,3,4))) #Notice that with the correct use of () you can combine functions.
```

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

```
#Ifelse, evaluates a logical statement and returns a value for TRUE and a value for FALSE.
ifelse(2>3, 0, 1)
```

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

You can combine functions and objects:

```
#Creates an object "mean_gdp" using the function mean() using the object we created above "country_gdp"
mean_gdp<-mean(country_gdp)
```

Or a more ellaborate example:

```
#Use object `age' to calculate the year you were born.
year_born<-2018-age
print(year_born)
```

[2] Based on Blissett 2016 which can be found here http://rslblissett.com/wp-content/uploads/2016/09/RTutorial_160930.pdf

```
## [1] 1989
```

```
#Works for me since my true birthday is in february.

#But what if my true birthday was in October?
#We can use think of a more complicated function that uses conditionals.

#1. Create month of birth in numbers
month<-10

#2. Create function that assigns year I was born. Using ifelse function.

year_birth<-ifelse(month>9,2018-age+1, year_birth)
print(year_birth)
```

```
## [1] 1990
```

```
#Now, what if my birthday was tomorrow (Sept 20)?
day<-20
month<-9 #replace value of month from 9 to 10.

#We can combine ifelse(). ifelse() is a very useful function!
year_birth<-ifelse(month<9,2018-age,ifelse((month=9&day<20),2018-age,2018-age+1))
print(year_birth)
```

```
## [1] 1990
```

[STOP HERE: QUESTIONS?]

Packages

Not all the functions you will need are already programmed in R. But, there are many packages that have more specific functions. These can be found in packages that you can install and use in R. "An R package is a collection of functions, data, and documentation that extends the capabilities of base R. Using packages is key to the successful use of R." We talked about this last time.

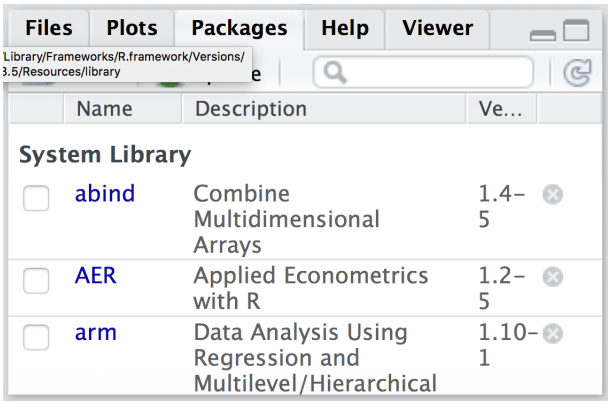
In Stata this logic is similar - ssc install.

Installing packages 1. To install package: type in the console the following line. ?install.packages.

Let's try installing package foreign.

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

Another option is to use the window "Packages"



\

2. Once the package is installed you need to load the package. Loading the package is necessary in each new R session in which you will use the package (unlike installation, which has to be done just once)

```
library(foreign)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

3. One source of complication is that two packages might have the same function. Depending on which one you load first R will mask the function for one of them. If this happens, you will get a warning and you can fix it on time.

Working directory

Where does the analysis live in your computer?

Working Directory: This is where R looks for files and this is also where it saves any file you want to save. You can identify the current working directory by typing in the console:

```
getwd()
```

Set Working Directory: You can also set a working directory by typing in the console:

Equivalent to Stata's *cd*

```
setwd("/Users/lauragarciamontoya/Box Sync/PhD/R Workshop/R for Stata Users Workshop") #your own
```

TIP: BE ORGANIZED: It is very easy to generate chaos in your computer (and brains) by not paying attention to the location of the files and making sure that you have a consistent system of organizing your files.

Note on File organization:In R you can have entire R projects, which are associated with R working directories. They work as folders in which you have workspace, history, source documents, objects, functions.

More information on projects here: https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects

Console ??? Script (how to use them)

The script is where you keep a record the "clean" code.

Best practices of coding and workflow - applies to Stata as well

Comments: Use # to comment your code. R will ignore it but you do want to make sure you keep track of what and why you write certain lines of code.

The same is done in Stata's do files using * for comments

```
#This part of the code creates variable of outcome
outcome<-c(1,2,3)

#This part of the code creates variable "new_name" equal to variable "outcome" + 2.
new_name<-outcome+2
```

Check: Check the object you just created, revise if the function works.

Formatting: Make sure you are organized in your scripts, if you get used to this good practice your life will be easier in the future.

To keep in mind

- R functions don't change the value of a variable or dataset
- if you just type sort(myvector) nothing will happen.
- Instead: myvector <- sort(myvector)

Practice

1. Create an object with the name "x":

Pick a number and Save it as x
Multiply x by 3
Take the log of the above
Subtract 4 from the above
Square the above

2. Install and load the package car

3. Write a line of code in which you:

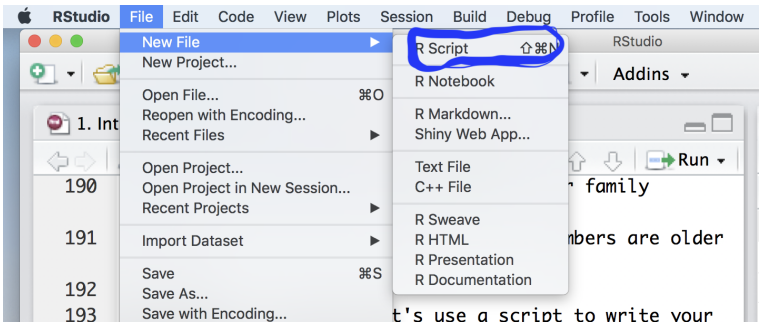
a. Create an object named "adult"
b. Assing the value of 1 if age>=18 and 0 otherwise. (ifelse() should be helpful)

4. Write code in your console such that:

a. You create an object with the ages of all the members of your family.
b. You calculate the average age of your family members.
c. You count how many of your family members are older than you.

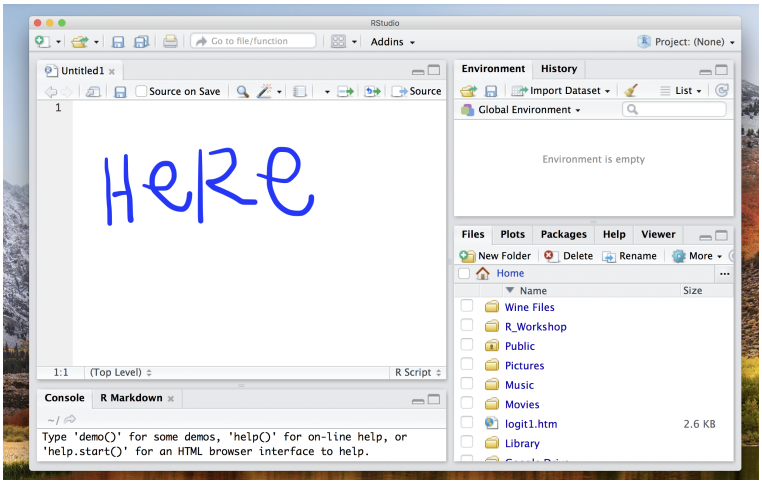
5. Repeat 3 - but now, let's use a script to write your lines of code. \

6. Open a new file - R script.



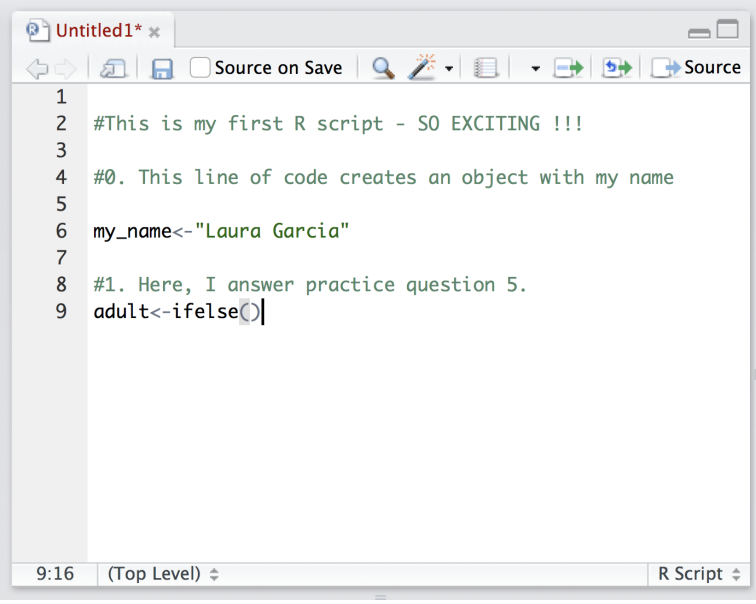
\

2. This is how it should look like:

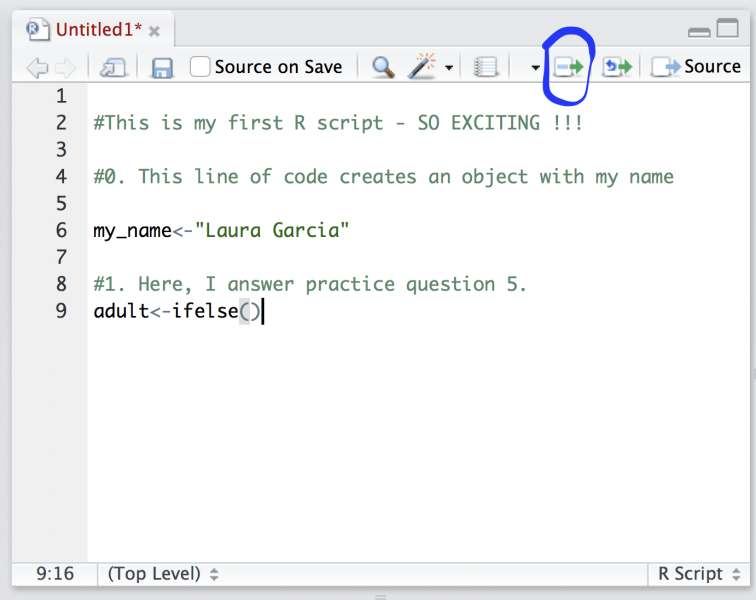


\

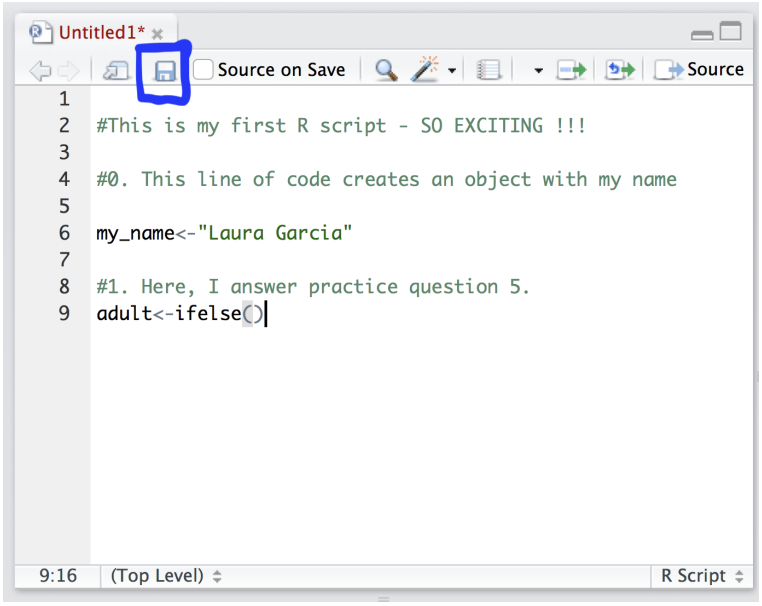
3. Just type your code, make sure you include comments.



4. Use the upper/right botton to *run* your code.



5. Make sure you save your script


Untitled1*
Source on Save
Source
1
2 #This is my first R script - SO EXCITING !!!
3
4 #0. This line of code creates an object with my name
5
6 my_name<-"Laura Garcia"
7
8 #1. Here, I answer practice question 5.
9 adult<-ifelse()
9:16
(Top Level)
R Script

\

[STOP HERE]

3. Vectors, matrices and data frames - INDEXING

3.1 Vectors

Vectors (also called arrays) are one possible type of object. You can create a vector with the "c()" function. "c" stands for combine. Vectors can only contain one type of values (text, numbers) In turn, the coding structure we used to create and name objects is the same that we used above.

Example

```
name_vector<-c('value1', 'value2', 'value3')
```

3.1.1 Structure of vectors:

```
a<-c(1:10)
a
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
y<-c(1:10)
y
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
z<-c(1,3,5,'a',6,5,3,5,7,8) #Notice that by including 'a', numbers became text.
z
```

```
##  [1] "1" "3" "5" "a" "6" "5" "3" "5" "7" "8"
```

3.1.2 Indexing Vectors

There are multiple ways to access or replace values in vectors or other data structures. The most common approach is to use *Indexing*. KEY 1: Brackets [ ] are used for indexing, whereas parentheses ( ) are used to call a function. KEY 2:Mastering the art of indexing early will be of great help in the learning curve of R. KEY 3: This is an important difference with STATA - key to understand logic of indexing.

a. Vector indexes (and all other indexes in R) start with 1, not 0:

```
x<-c('a', 'b', 'c', 'd', 'e')
x[1]
```

```
## [1] "a"
```

b. You can take slices of vectors: take

```
x[1:3]
```

```
## [1] "a" "b" "c"
```

c. Or exclude values with a negative sign:

```
x[-1]
```

```
## [1] "b" "c" "d" "e"
```

d. Elements are returned in the order that the indices are supplied:

```
x[c(5,1)]
```

```
## [1] "e" "a"
```

e. You can use a vector of integers or booleans to select from a vector as well:

```
x[x<'c']
```

```
## [1] "a" "b"
```

```
x[c(1,3,5)]
```

```
## [1] "a" "c" "e"
```

Practice: Creating vectors and indexing:

1. Create one vector named "months" which contains the months of the year.
2. Create one vector named "days" which contains the days of the week
3. Display all the months of the year
4. Indexing, display the first 6 months of the year
5. Indexing, display your favorite month of the year
6. Indexing, display all of the months of the year except your least favorite

3.1.3 Other useful functions

Get the length of a vector with function length:

```
length(x)
```

```
## [1] 5
```

You can check if a vector contains a certain value using %in%

```
'b' %in% x # Is value 'b' found in vector x?
```

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

Use function "which" to find all positions:

```
y <- c(1:3, 1:3)
which(y == 3) #In which positions can I find 3 in vector y?
```

```
## [1] 3 6
```

Or get the first position of one or more elements in a vector with the "match" function:

```
match(c('b', 'd', 'k'), x)
```

```
## [1]  2  4 NA
```

You can also name the elements of a vector with the function "names".

```
x<-1:5
names(x)<-c("Ohio","Illinois","Indiana","Michigan","Wisconsin")
```

Which allows you to select values from the vector using the names:

```
x["Ohio"]
```

```
## Ohio
##    1
```

```
x[c("Illinois", "Indiana")]
```

```
## Illinois  Indiana
##        2        3
```

3.1.4 Factors:

Factors are a special type of vector can be used for categorical variables:

```
colors<-c("red", "blue", "green", "red", "red", "blue")

colors_factors<-factor(colors)
```

They could also be converted into integers, not always useful but good to know it is possible:

```
colors_levels<-as.integer(colors_factors)
colors_levels
```

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

```
mean(colors_levels)
```

```
## [1] 2.166667
```

```
gender<-c("f","m","m","m","f")
gender
```

```
## [1] "f" "m" "m" "m" "f"
```

```
gender_levels<-as.integer(factor(gender))
gender_levels
```

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

3.2 Matrices:

Matrices are conformed by one or more vectors. KEY: all vectors in a matrix have to be of the same type, and lenght.

Practice

Use the help command in R to figure out how to create the following matrix

1	6
3	7
5	9

```
?matrix
```

[STOP HERE]

3.2.1 Creating matrices:

```
A<-matrix(c(1,2,3,11,12,13),nrow = 3,ncol = 2)

B<-matrix(c(0,1,4,10,15,20),nrow = 2,ncol = 3)

matrix(c('a', 'b', 'c', 'd'), nrow=2)
```

```
##      [,1] [,2]
## [1,] "a"  "c"
## [2,] "b"  "d"
```

```
y<-matrix(1:25, nrow=5, byrow=TRUE)

y
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12   13   14   15
## [4,]   16   17   18   19   20
## [5,]   21   22   23   24   25
```

```
y%*%y #Matrix multiplication
x<-1:5 #Create new variable
y%*%x
y^-1  #matrix inversion
y * -1
```

3.2.2 Indexing matrices:

The logic of indexing is similar to that of the vectors. However, here we have two dimensions that we care about: row and column.

Elements in a matrix can be identified indexing the following way:

```
y[1,1] #Element in row 1 column 1
```

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

```
y[1,] #Elements in Row 1
```

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

```
y[,1] #Elements in Column 1
```

```
## [1]  1  6 11 16 21
```

```
y[1:2,3:4]
```

```
##      [,1] [,2]
## [1,]    3    4
## [2,]    8    9
```

```
y[,c(1,4)]
```

```
##      [,1] [,2]
## [1,]    1    4
```

```
## [2,]    6    9
## [3,]   11   14
## [4,]   16   19
## [5,]   21   24
```

3.3 Lists:

Lists are a bit like complex vectors. An element of a list can hold any other object, including another list. You can keep multi-dimensional and ragged data in R using lists. Not useful now, but good to know they exist.

```
l1 <- list(1, "a", TRUE, 1+4i, c(1,3,4,6))
l1
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
##
## [[5]]
## [1] 1 3 4 6
```

3.3.1 Indexing lists

Elements of the list:

```
l1[1]
```

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

```
l1[5]
```

```
## [[1]]
## [1] 1 3 4 6
```

Elements that are part of an element of a list:

```
l1[[2]][1]
```

```
## [1] "a"
```

```
l1[[5]][2]
```

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

3.4 Data Frames

Data frames are the core data structure in R. A data frame is a list of named vectors with the particularity that all vectors have the same length. Columns are typically variables and rows are observations. Different columns can have different types of data:

3.4.1 Creating Data Frames

Using these commands we will create our own dataframe:

KEY: LENGHT OF THE VECTORS HAS TO BE THE SAME, but each vector can be different type.

```
id<-1:20 #creating a vector with number from 1:20
color<-c(rep("red", 3), rep("green",10), rep("blue", 7))
gender<-c(rep("f",10),rep("m",10))
score<-runif(20)

#Put together vectors in one data frame with the name df.
df<-data.frame(id, color, score,gender)
```

Instead of making individual objects first, we could do it all together:

```
df<-data.frame(id=1:20,
               color=c(rep("red", 3), rep("green",10), rep("blue", 7)),
               score=runif(20), gender=c(rep("f",10),rep("m",10))
)
```

We can now use some functions to explore what we have in our data frame:

```
#Table
table(df$color,df$gender)
```

```
##
##         f m
##   blue  0 7
##   green 7 3
##   red   3 0
```

```
table(df$color)
```

```
##
##  blue green   red
##     7    10     3
```

```
#Dimension
dim(df)
```

```
## [1] 20  4
```

```
#First 6 rows
head(df) #possible to adjust the number of rows
```

```
##   id color     score gender
## 1  1   red 0.1795844      f
## 2  2   red 0.8699835      f
## 3  3   red 0.2605568      f
## 4  4 green 0.2912591      f
## 5  5 green 0.2995723      f
## 6  6 green 0.9099495      f
```

```r
head(df,5)
```

```
##   id color     score gender
## 1  1   red 0.1795844      f
## 2  2   red 0.8699835      f
## 3  3   red 0.2605568      f
## 4  4 green 0.2912591      f
## 5  5 green 0.2995723      f
```

3.4.2 Indexing Data Frames:

Data frames can be indexed like matrices to retrieve the values.

```r
df[2,2] #Retrieves value in row 2 column 2.
```

```
## [1] red
## Levels: blue green red
```

```r
df[1,] #Retrieves row 1.
```

```
##   id color     score gender
## 1  1   red 0.1795844      f
```

```r
df[-(2:3),]#Retrieves all except rows 2 to 3.
```

```
##    id color      score gender
## 1   1   red 0.17958441      f
## 4   4 green 0.29125914      f
## 5   5 green 0.29957234      f
## 6   6 green 0.90994949      f
## 7   7 green 0.06999008      f
## 8   8 green 0.37956895      f
## 9   9 green 0.09355730      f
## 10 10 green 0.38494619      f
## 11 11 green 0.12133578      m
## 12 12 green 0.46683913      m
## 13 13 green 0.77589954      m
## 14 14  blue 0.69399006      m
## 15 15  blue 0.20915475      m
## 16 16  blue 0.20692839      m
## 17 17  blue 0.94190929      m
## 18 18  blue 0.11812371      m
## 19 19  blue 0.43564271      m
## 20 20  blue 0.31760070      m
```

Practice 1:

Index to retrieve the following values:

1. Third column
2. Rows 6 to 10 in column 3.
3. All values except those in column 2.
4. All values except those in row 1 and column 10.

3.4.3 Alternatives ways of Indexing in Data Frames:

Each of the vectors in a dataframe has a name. You can also use the names of the vector after a $ (instead of indexing):

```
df$color
```

```
##  [1] red   red   red   green green green green green green green green green
## [12] green green blue  blue  blue  blue  blue  blue  blue
## Levels: blue green red
```

Indexing into a data frame with a single integer or name of the column will give you the column(s) specified as a new data frame.

```
df['color']
```

```
##    color
## 1    red
## 2    red
## 3    red
## 4  green
## 5  green
## 6  green
## 7  green
## 8  green
## 9  green
## 10 green
## 11 green
## 12 green
## 13 green
## 14  blue
## 15  blue
## 16  blue
## 17  blue
## 18  blue
## 19  blue
## 20  blue
```

```
df[2:3]
```

```
##    color      score
## 1    red 0.17958441
## 2    red 0.86998346
## 3    red 0.26055684
## 4  green 0.29125914
## 5  green 0.29957234
## 6  green 0.90994949
## 7  green 0.06999008
## 8  green 0.37956895
## 9  green 0.09355730
## 10 green 0.38494619
## 11 green 0.12133578
## 12 green 0.46683913
## 13 green 0.77589954
## 14  blue 0.69399006
## 15  blue 0.20915475
## 16  blue 0.20692839
```

```
## 17  blue 0.94190929
## 18  blue 0.11812371
## 19  blue 0.43564271
## 20  blue 0.31760070
```

Instead of index numbers or names, you can also select values by using logical statements. This is usually done when the goal is to select rows.

```
df[df$color == "green",]
```

```
##    id color      score gender
## 4   4 green 0.29125914      f
## 5   5 green 0.29957234      f
## 6   6 green 0.90994949      f
## 7   7 green 0.06999008      f
## 8   8 green 0.37956895      f
## 9   9 green 0.09355730      f
## 10 10 green 0.38494619      f
## 11 11 green 0.12133578      m
## 12 12 green 0.46683913      m
## 13 13 green 0.77589954      m
```

Practice 2:

Describe in words what the followings commands mean.

```
df[df$score > .5,]
```

```
##    id color     score gender
## 2   2   red 0.8699835      f
## 6   6 green 0.9099495      f
## 13 13 green 0.7758995      m
## 14 14  blue 0.6939901      m
## 17 17  blue 0.9419093      m
```

```
df[df$score > .5 & df$color == "blue",]
```

```
##    id color     score gender
## 14 14  blue 0.6939901      m
## 17 17  blue 0.9419093      m
```

You can count number of observations by using logical statements and sum.

```
sum(df$score > .5)
```

```
## [1] 5
```

You can assign names to the rows of a data frame as well as to the columns, and then use those names for indexing and selecting data.

```
rownames(df)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20"
```

You can add columns or rows simply by assigning values to them.

```
df$weight<-c(rnorm(20))
```

rbind and cbind (for row bind and column bind) are two functions that can be useful when you want to combine a vector with a dataframe.

```
df$year<-1901:1920
df
```

```
##    id color      score gender      weight year
## 1   1   red 0.17958441      f -0.17764318 1901
## 2   2   red 0.86998346      f -0.04722422 1902
## 3   3   red 0.26055684      f -1.20702278 1903
## 4   4 green 0.29125914      f  1.88888415 1904
## 5   5 green 0.29957234      f  1.51226939 1905
## 6   6 green 0.90994949      f -0.84127657 1906
## 7   7 green 0.06999008      f -0.63037241 1907
## 8   8 green 0.37956895      f  0.23255271 1908
## 9   9 green 0.09355730      f  0.31058640 1909
## 10 10 green 0.38494619      f  2.47313741 1910
## 11 11 green 0.12133578      m  0.39978986 1911
## 12 12 green 0.46683913      m -1.61643816 1912
## 13 13 green 0.77589954      m -0.45241775 1913
## 14 14  blue 0.69399006      m -1.48532597 1914
## 15 15  blue 0.20915475      m  0.25857079 1915
## 16 16  blue 0.20692839      m -0.88683019 1916
## 17 17  blue 0.94190929      m -0.04407694 1917
## 18 18  blue 0.11812371      m -0.23000851 1918
## 19 19  blue 0.43564271      m -0.32547151 1919
## 20 20  blue 0.31760070      m -0.12563402 1920
```

```
year_plus_1<-1902:1921

df<-cbind(df,year_plus_1)
```

You could rename columns (variables) with function names()

```
names(df)[names(df)=='color'] <- 'rainbow'

names(df)
```

```
## [1] "id"          "rainbow"     "score"       "gender"      "weight"
## [6] "year"        "year_plus_1"
```

Create subsets using indexing/conditions

```
mysubset<-list(reds=df[df$rainbow == "red",])
```

or a subset of variables

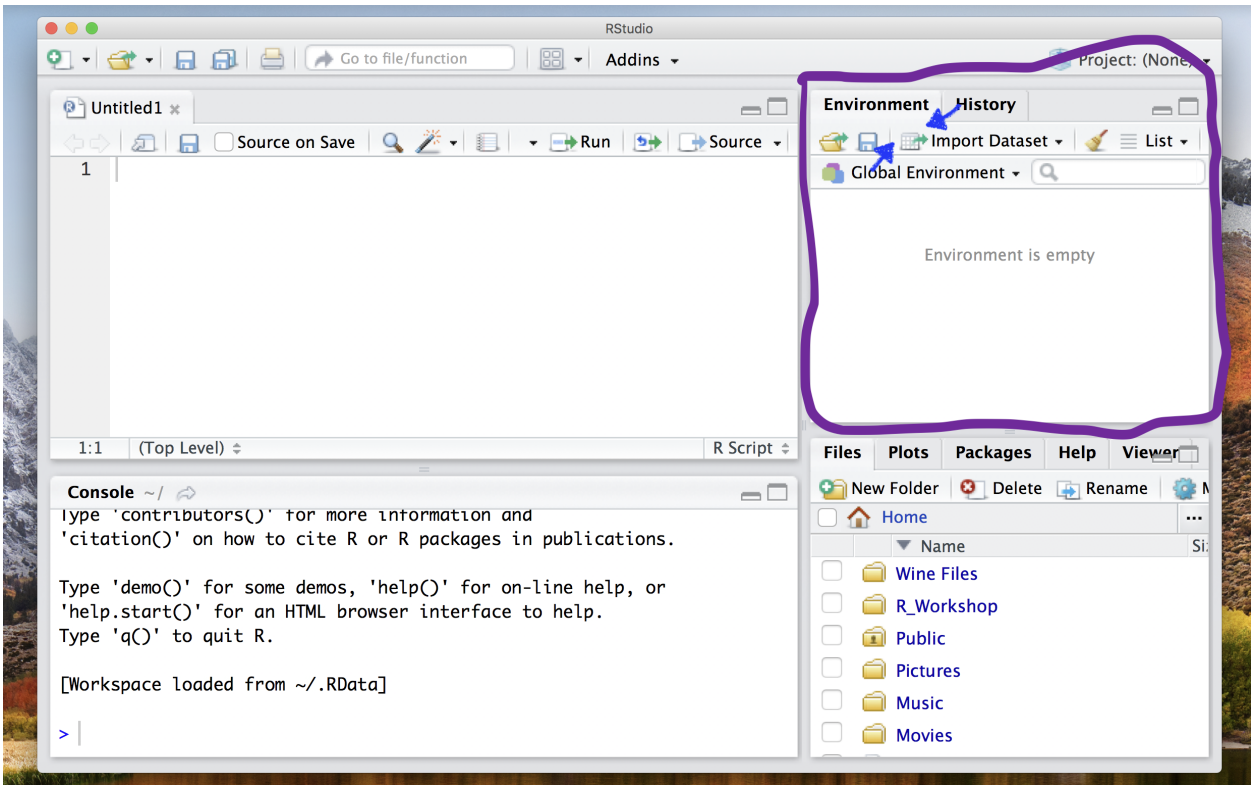
```
mysubset2<-cbind(df$rainbow,df$year)
```

3.4.3 Importing Datasets

Before we start, make sure you download and save the excel file "Example_data.xls" available on Github. Save it in a place that is easy to find later.

a) Using the environment window:

You can import to R any a dataset you previously saved in computer. It will allow you to access your file directory and to navigate in your computer until you find and select the document.



b) Using lines of code

This one is harder because it implies understanding of directory/path in your computer and in R. We will import the same dataset "Example_data.xls" Once you are comfortable with setwd() and how to type paths, this is the most efficient option. You can run a script that has a line to import the datasets.

```
##This will show you the current working directory.
getwd()
```

```
## [1] "/Users/lauragarciamontoya/Box Sync/PhD/R Workshop/R for Stata Users Workshop"
```

```
##Change it to your own working directory (varies for each person)
#setwd("/Users/lauragarciamontoya/Desktop")
library(readxl)
#imported_data_example<-("/Desktop/Example_data.xls") # SPECIFY THE PATH.
```

c) Datasets that are part of packages.

```
install.packages('gapminder', dep=TRUE, repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/4t/1yq3d5492qv72rfgn8v92mm40000gn/T//RtmpWOV6Kb/downloaded_packages
```

```
library(gapminder)
```

d) With a url - dataset that is in a website

```
site="http://faculty.wcas.northwestern.edu/~lmh735/nes2008.RData"
load(file=url(site))
ls()
```

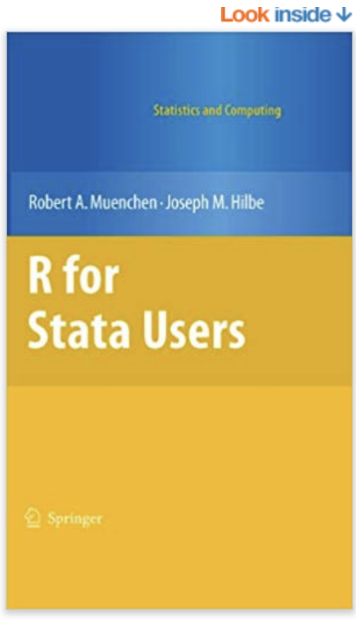
```
##  [1] "a"              "A"              "age"            "B"
##  [5] "color"          "colors"         "colors_factors" "colors_levels"
##  [9] "country_gdp"    "day"            "df"             "gender"
## [13] "gender_levels"  "id"             "l1"             "mean_gdp"
## [17] "month"          "mysubset"       "mysubset2"      "name"
## [21] "name_vector"    "nes08"          "new_name"       "outcome"
## [25] "program"        "r"              "score"          "site"
## [29] "x"              "y"              "year_birth"     "year_born"
## [33] "year_plus_1"    "z"
```

[STOP HERE: QUESTIONS?]

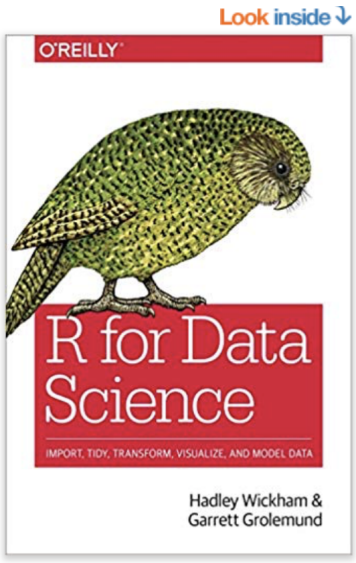
ADDITIONAL RESOURCES:

Books:

R for Stata Users (Muenchen and Hilbe)



R for Data Science: Import, Tidy, Transform, Visualize, and Model Data (Wickham and Grolemund)



LINKS TO USEFUL WEBSITES:

http://rslblissett.com/wp-content/uploads/2016/09/RTutorial_160930.pdf

https://github.com/nuitrcs/rworkshops

https://www.rstudio.com/resources/cheatsheets/

Focused on Stata to R

https://dlab.berkeley.edu/blog/quick-and-easy-way-turn-your-stata-knowledge-r-knowledge

https://dss.princeton.edu/training/RStata.pdf

https://github.com/EconometricsBySimulation/RStata/wiki/Dictionary:-Stata-to-R