ECON 172 - Section 2 : Introduction to R*

Spring 2019

GSIs: Madeline Duhon and Dennis Egger

1 Welcome to R

R is a programming language and environment for statistical computing and graphics which is free to use and runs on Windows, Unix and MacOS. To do analysis in R, one can either write one's own commands or take advantage of the huge number of pre-existing packages written by R users and freely available.

1.1 Getting Started in R

1.1.1 Using R on your local machine (optional)

R can be downloaded from https://cran.rstudio.com/.

We also recommend downloading RStudio, a user interface which makes R easier to use. It can be downloaded from https://www.rstudio.com/products/rstudio/download/.

If you run R on your local computer, you will need to install any packages you want to use (just one time), then load them into R's library for the current session. After that, it's easy to access any of the commands in the package (which are also generally extensively documented in a sort of user guide for the package).

1.1.2 Using R on your UC Berkeley's Datahub (recommended)

RStudio also runs on the UC Berkeley Datahub, so you don't need to install it locally on your computer. To start RStudio on Datahub, simply go to:

https://datahub.berkeley.edu/user/*YourCalCentralUsername*/rstudio/

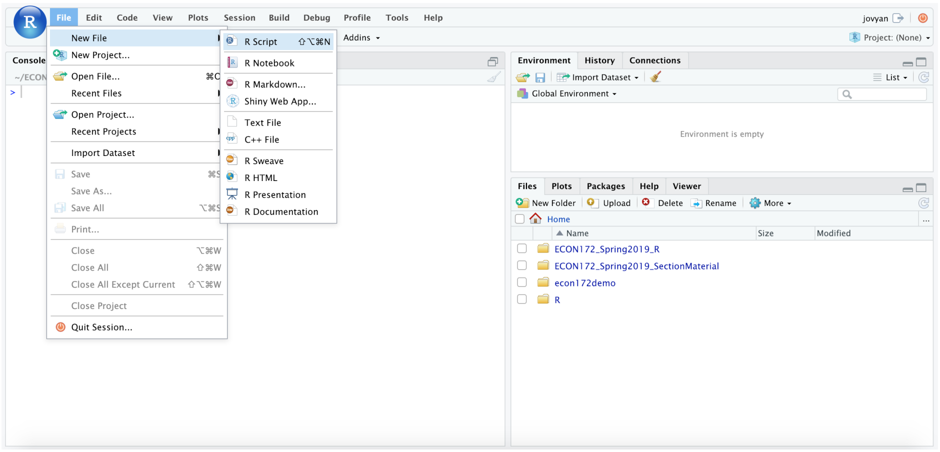
In our sections (and for the problem sets), we are going to show you this way of using RStudio. However, anything that we show you should also work on your local machine.

1.2 Getting Started in R

When we use R, we want to write and save our commands in a script. This will let us reproduce everything we do without having to retype our commands every time we work on a project. The first step is to open a script which saves with a .R extension, in RStudio as below:

*Thanks to Christina Brown and Isabelle Cohen, Ezequiel Garcia-Lembergman and Eric Hsu for earlier drafts of these notes.

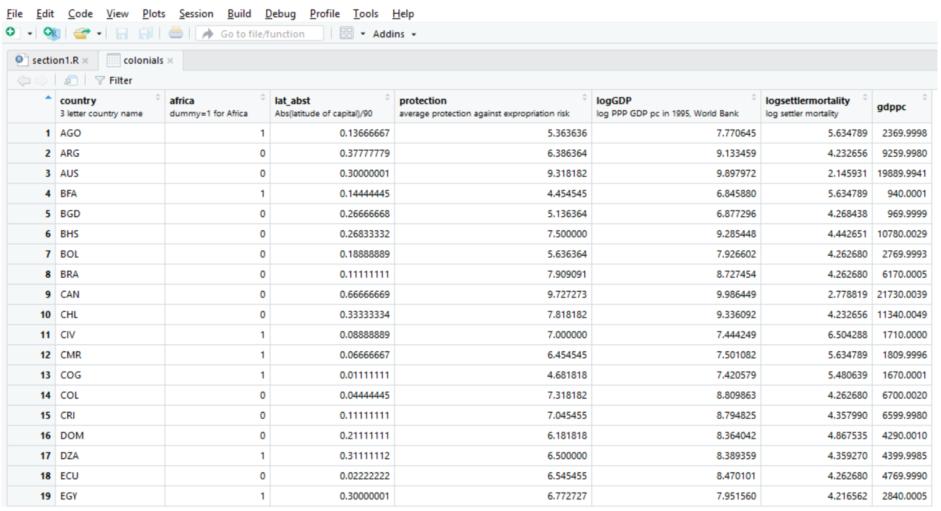
Figure 1: R Script



1.3 Loading Data

The first step in using R for statistical analysis is to store data in what R calls a "data frame," a matrix whose columns have different modes (like numeric, words, etc.) and each row is a unit of observation (i.e: a person or a country, etc.)

Figure 2: Dataframe in R



	country 3 letter country name	africa dummy=1 for Africa	lat_abst Abs(latitude of capital)/90	protection average protection against expropriation risk	logGDP log PPP GDP pc in 1995, World Bank	logsettlermortality log settler mortality	gdppc
1	AGO	1	0.13666667	5.363636	7.770645	5.634789	2369.9998
2	ARG	0	0.37777779	6.386364	9.133459	4.232656	9259.9980
3	AUS	0	0.30000001	9.318182	9.897972	2.145931	19889.9941
4	BFA	1	0.14444445	4.454545	6.845880	5.634789	940.0001
5	BGD	0	0.26666668	5.136364	6.877296	4.268438	969.9999
6	BHS	0	0.26833332	7.500000	9.285448	4.442651	10780.0029
7	BOL	0	0.18888889	5.636364	7.926602	4.262680	2769.9993
8	BRA	0	0.11111111	7.909091	8.727454	4.262680	6170.0005
9	CAN	0	0.66666669	9.727273	9.986449	2.778819	21730.0039
10	CHL	0	0.33333334	7.818182	9.336092	4.232656	11340.0049
11	CIV	1	0.08888889	7.000000	7.444249	6.504288	1710.0000
12	CMR	1	0.06666667	6.454545	7.501082	5.634789	1809.9996
13	COG	1	0.01111111	4.681818	7.420579	5.480639	1670.0001
14	COL	0	0.04444445	7.318182	8.809863	4.262680	6700.0020
15	CRI	0	0.11111111	7.045455	8.794825	4.357990	6599.9980
16	DOM	0	0.21111111	6.181818	8.364042	4.867535	4290.0010
17	DZA	1	0.31111112	6.500000	8.389359	4.359270	4399.9985
18	ECU	0	0.02222222	6.545455	8.470101	4.262680	4769.9990
19	EGY	1	0.30000001	6.772727	7.951560	4.216562	2840.0005

R can load data from many types of files, including .csv and .txt. R has its own type of data file, with a .rds or .Rdata extension, and can also load data from Stata, SPSS, and SAS (other statistics computing programs) with the appropriate commands. The .rds format allows for a single object to be saved at a time, like a single dataset. The .RData format allows for multiple objects to be saved at once. Either one is a great way to save your data in R format.

When we load a dataframe in R, we give it a name for reference, such as "mydata" or something more specific, like "colonials". We also need to tell R where to find the data. We can do that by specifying a full file path to its exact location of the data. We could also tell R to pick a specific directory, or folder, with the setwd (set working directory) command; if we do that, then we only need to tell R the name of the data for it to find it in that folder (or the rest of the file path, if necessary). **We strongly suggest setting the directory where you are going to work before starting.**

For this section, we will be using data from Robinson, Acemoglu and Johnson (2001). For your convenience, the following link imports this data directly into your Datahub repository:

`https://datahub.berkeley.edu/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Fdennistegger%2FECON172_Spring2019_SectionMaterial&urlpath=rstudio`.

Before we get started, you'll need to create a couple of folders in your Home directory in RStudio on datahub (in the lower right corner). Create a folder called "ECON172_Spring2019_R," and a subfolder called "Section_02". Copy over the files that you downloaded using the link above. All necessary files will also be available on bCourses if you prefer to upload the files directly.

As we get started, the first step is to set a working directory.

```
setwd("~/ECON172_Spring2019_R/Section_02/")
```

Next we want to open some data, and load the data into a dataframe. For the purposes of this section, we'll use the dataset "colonials" which is an example dataset that contains data on property rights index, gdp per capita and other variables for countries that were European colonies in the past.

```
library(haven) #this library allows you to load datasets in Stata format
colonials <- read_stata("colonials.dta")
```

Note that we tell R to name the dataframe "colonials" with the `<-` symbol. In general, the `<-` symbol means: assign something to something else. i.e: y `<-` x means assign x to y. You may notice that after the lines of "code" above, we included some explanation after the # symbol. # tells R that the following text is a comment, not code. It's great to include comments in your scripts, so that you don't forget why you wrote a particular line, or what you were doing in some section of the code. It is also good to allow someone else to replicate your work easily. When one is doing research it is a good practice to work in such a way that results are easy to replicate by anyone that has the code.

Since we are not using Stata in this class, let's remove what we just loaded from memory and instead import a .csv file with the same data.

```
remove(colonials)
colonials <- read.csv("colonials.csv")
```

Next we can visualize the database we just imported, and take a look at what variables are included.

```
View(colonials) ## view dataframe directly
colonials ## or print first 10 observations
names(colonials) ## print variable names
colonials_variables <- names(colonials) ## store list, assigned "colonials_variables"
colonials_variables ## take a look at this list
```

2 Analysis in R

2.1 Summary Statistics

Once a dataframe is loaded in R, it's very easy to run basic summary statistics. Unlike many other statistics programs, R provides fairly minimal output, and it's possible to store the results of a command without ever displaying them. For example, we may be interested in summary statistics for the GDP per capita of countries in the sample, which we can get by typing `summary(colonials$gdppc)`. This command instructs R to look in the dataframe listed before the `$` symbol (in this case, `colonials`), and to summarize the variable that follows the `$` symbol (in this case, `gdppc`).

```
summary(colonials$gdppc) ## just print
colonials_sumstats <- summary(colonials$gdppc) ## store summary, "colonials_variables"
colonials_sumstats ## take a look at this summary
```

Or, we may be interested specifically in the **mean** value of GDP per capita for countries in the sample specifically, which we can get by typing `mean(colonials$gdppc)`. As before, this command instructs R to look in the `colonials` dataframe and calculate the mean for the variable `gdppc`. We can store value resulting from this calculation, by using the `<-` symbol to store the results in a new object called `meangdp`, which can be by typing `meangdp` into the console, or used in another command or context later on.

```
mean(colonials$gdppc) ## just print
meangdp <- mean(colonials$gdppc) ## save as a variable, called "meangdp"
meangdp ## take a look at the value stored in of "meangdp"
```

Similar commands exist for standard deviation (`sd`), variance (`var`), minimum (`min`), maximum (`max`), median (`med`), range (`range`) and quantile (`quantile`).

2.2 Regression

Linear regression is also very straightforward to apply in R. The `lm` command, which comes preloaded, can be used for univariate or multivariate regression. Examples of how to run a regression of log GDP per capita on protection against expropriation and latitude are below:

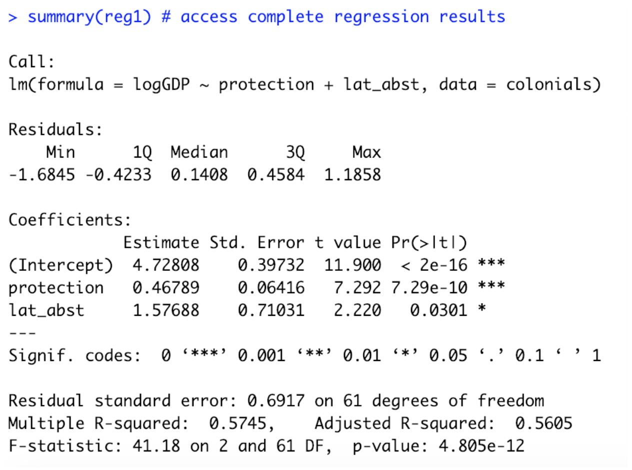
```
## Univariate regression of log gdp per capita on property rights index.
lm(logGDP ~ protection, data=colonials) # display regression results

## Multivariate, adding absolute latitude.
lm(logGDP ~ protection + lat_abst, data=colonials) # display regression results

## Multivariate regression, storing results
reg1 <- lm(logGDP ~ protection + lat_abst, data=colonials) # display regression results
```

In the last example, we store the results in the object `reg1`. If you try this in R, you'll notice the results don't display on the screen when you store the results, but you can access them by naming the object.

```
summary(reg1) # access complete regression results
```

Which gives the following output:

Figure 3: Regression Results



```
> summary(reg1) # access complete regression results

Call:
lm(formula = logGDP ~ protection + lat_abst, data = colonials)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.6845 -0.4233  0.1408  0.4584  1.1858 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  4.72808    0.39732  11.900  < 2e-16 ***
protection   0.46789    0.06416   7.292 7.29e-10 ***
lat_abst     1.57688    0.71031   2.220   0.0301 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.6917 on 61 degrees of freedom
Multiple R-squared:  0.5745,	Adjusted R-squared:  0.5605 
F-statistic: 41.18 on 2 and 61 DF,  p-value: 4.805e-12
```

2.3 Plotting

R also makes it easy to do basic plots. For example, one might want to create a scatterplot from two variables, and graph the regression line - or line of best fit - on the plot. This is easy to do with the plot command, which as a default plots the first variable listed on the horizontal axis, and the second on the vertical axis. Below are two ways of plotting protection against expropriation index and log of GDP per capita, one very simple and one with titles and labels. The second version also includes the `abline` command to add a regression line to the graph.

```
## A very basic scatterplot
plot(colonials$protection,colonials$logGDP)

## A scatterplot with titles
plot(colonials$protection,colonials$logGDP,
xlab="protection against expropriation", ylab="Log (GDP per capita)")
abline(lm(logGDP ~ protection , data=colonials)) # line of best fit
```

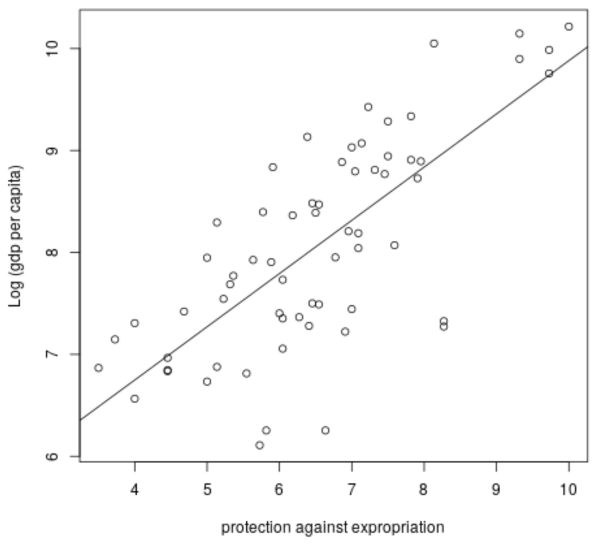
2.4 Exporting Results

These results and plots are great, but we may also want to export them for future use, for example to show to a professor as results in a problem set. Luckily, it's not necessary to take a screenshot of the graph, or even to export it from within RStudio, nor do we need to manually copy our estimates from regression tables.

Plots are very easy to store from RStudio. We can run code as shown below, which opens up a png file called Plot1, generates the graph, and closes and saves the graph.

```
png(file="Plot1.png")
plot(colonials$protection,colonials$logGDP,
xlab="protection against expropriation", ylab="Log (gdp per capita)")
abline(lm(logGDP ~ protection , data=colonials)) # line of best fit
dev.off()
```

Figure 4: Property Rights and Development



There are many packages in R that let one create very nice tables. We recommend stargazer. Take the last regression we ran, stored as `reg1` and say we want to export that to a table to be used in word or latex. To use stargazer, we would first make sure we have it installed on our computer, then add it to our library:

```
install.packages("stargazer")
library(stargazer)
```

We could then run following code:

```
## Default stargazer table
stargazer(reg1,out="Table 1.html",type="html")

## Stargazer table, customized
stargazer(reg1,out="Table 2.html",type="html",
title="Property rights and Development")

## Latex table
stargazer(reg1,out="Table 2.tex",
title="Property rights and Development",align=TRUE,
omit.stat=c("LL","ser","f","adj.rsq"),no.space=TRUE)
```

The last generates the following table:

Table 2: Property rights and Development

	Dependent variable:
	logGDP
protection	0.468***
	(0.064)
lat_abst	1.577**
	(0.710)
Constant	4.728***
	(0.397)
Observations	64
R^2	0.574
Note:	*p<0.1; **p<0.05; ***p<0.01

To save a table in word, you can use the html format, or type, as suggested by the first two tables, then simply open the table in your browser and copy it into word. Stargazer has extensive documentation (try `??stargazer` in R) and can also be used to make great summary statistics tables - try it out!

3 Resources for further study

UCLA has a number of excellent resources to help you learn more about how to use R, available at: http://www.ats.ucla.edu/stat/r/.

You can also find interactive lessons at Try R (http://tryr.codeschool.com/) for practice writing code.

4 R Commands: Cheat Sheet

Description	Command
Set up	
Install "stargazer" package	`install.packages("stargazer")`
Add "stargazer" to library	`library("stargazer")`
Search for "stargazer" in documentation	`??stargazer`
Get help for "plot"	`?plot`
Check working directory	`getwd()`
Change working directory	`setwd("/Users/myname/Documents")`
Import from csv	`mydata <- read.csv("/Users/myname/Documents/mydata.csv")`
Import from R (rds)	`mydata <- readRDS("/Users/myname/Documents/mydata.rds")`
Save R (rds) file	`save(mydata,file="/Users/myname/Documents/mydata.rds")`
Import from R (RData)	`load("Users/myname/Documents/mydata.RData")`
Save R (RData) file	`save(mydata,file="/Users/myname/Documents/mydata.RData")`
Save log	`savehistory(file="mylog.Rhistory")`
Examining dataset	
Sumarize data	`summary(mydata)`
Number of variables in dataset	`length(mydata)`
Observations of "x" in "mydata"	`length(mydata$x)`
View first ten observations	`head(mydata,n=10)`
View data	`View(mydata)`
Edit data	`Edit(mydata)`
Rename variable	`names(mydata)[names(mydata)=="old_name"] <- "new_name"`
Descriptive statistics	
Mean of variable "x"	`mean(mydata$x)`
Variance of variable "x"	`var(mydata$x)`
Standard deviation of variable "x"	`sd(mydata$x)`
Minimum value of variable "x"	`min(mydata$x)`
Maximum value of variable "x"	`max(mydata$x)`
Range of variable "x"	`range(mydata$x)`
Quantiles of variable "x"	`quantile(mydata$x)`
Range of variable "x"	`range(mydata$x)`
Mean by group	`tapply(mydata$x, mydata$group, mean)`
Analyzing data	
Univariate regression	`reg1<-lm(y ~ x, data=mydata)`
Multiivariate regression	`reg2<-lm(y ~ x1 + x2 + x3, data=mydata)`
Data plots	
Scatterplot	`plot(mydata$x,mydata$y)`
Add regression line to scatterplot	`abline(lm(y ~ x, data=mydata))`
Exporting results	

Saving plots	`png(file="Plot1.png")`
	`plot(mydata$x, mydata$y)`
	`abline(lm(y ~ x, data=mydata))`
	`dev.off()`

Stargazer for regression results, default	`stargazer(reg1,out="Table 1.html",type="html")`

Stargazer for regression results, custom	`stargazer(reg1,out="Table 2.html",type="html",`
	`title="Table 2",align=TRUE,`
	`omit.stat=c("LL","ser","f","adj.rsq"),`
	`no.space=TRUE)`

References

Robinson, James A., Daron Acemoglu and Simon Johnson. 2001. “The Colonial Origins of Comparative Development: An Empirical Investigation.” *American Economic Review* 91:1369–1401.