ECON 172 - Section 2 : Introduction to R

Dennis Egger and Maddie Duhon

9/3/2019

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.

Getting Started

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).

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://r.datahub.berkeley.edu/

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.

RMarkdown

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. What's even nicer about R is a file format called RMarkdown, such as the one you are currently working in. This file type allows you to combine text, mathematical equations (in a format called LaTex), code as well as dynamic output from this code, and automatically compile / output everything into a pdf document. (In RMarkdown language, this process is called "knitting" the document.)

Setting up the Environment

First, we need to set up the R environment, including installing all the required packages, loading all the required packages. This is done using R-code. The way to let RMarkdown know that the following output is a 'code chunk' is by enclosing the code chunk in triple quotations, as below. The code chunk indicator is followe by curly brackers, and r (indicating this code chunk is in the R language), a name for the code chunk (here "setup"), and some additional parameters. The parameter "results='hide' " and "message=FALSE" lets RMarkdown know that we will not want to include the output of or any messages generated by this code chunk in the pdf document (but the code will still run).

In your code, you can add comments, using # (beginning of the line) and ## (end of the line). These comments are not run as code, but will help your later self (and others reading your code) to understand what you were doing.

```r
knitr::opts_chunk$set(echo = TRUE) ## sets global option to include code chunks in-line

#install.packages("haven") ## only run once to install
library(haven) #this library allows you to load datasets in Stata format

#install.packages("tidyverse") ## only run once to install
library(tidyverse)

#install.packages("summarytools") ## only run once to install
library(summarytools)

#install.packages("stargazer") ## only run once to install
library(stargazer) ##This package is great for making tables in .html, .tex and many other formats.

#install.packages("broom") ## only run once to install
library(broom)
```

The code chunk specifies that, by default, code chunks are part of the pdf output. Next, it installs a few packages. You only need to run this installation once. Afterwards, the package will remain installed on your datahub. Next, the "library" command lets R know that you will want to use these packages in your code.

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.)

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.

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://r.datahub.berkeley.edu/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Fdennistegger%2FECON172_Spring2019_SectionMaterial&urlpath=rstudio%2F The folder should now appear as "ECON172_Spring2019_SectionMaterial" on your workspace (bottom right).

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. 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.

```r
## Load in data in csv form, from the folder just loaded into your directory.
## Important: Paths are always relative to the RMarkdown file location
##            Alternatively, specify your working directory using
##            knitr::opts_chunk$set(root.dir = "xxx")

colonials <- read.csv("../ECON172_Spring2019_SectionMaterial/Section_02_IntroToR/colonials.csv")
```

Next we can visualize the database we just imported, and take a look at what variables are included. To do so, it's easiest to click directly on the dataset in your Environment pane on the top right in RStudio.

```
## Let's take a look on the dataset.
colonials ## or print first 10 observations

## Let's take a look the variables in this dataset.
names(colonials) ## just print
colonials_variables <- names(colonials) ## store list, assigned "colonials_variables"
colonials_variables ## take a look at this list
```

Analysis in R

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). Note, the "results = 'markup' " bit tells RMarkdown to display the output of the code exactly as displayed in the R console.

```
## Let's look at some summary statistics.
## Here is the min, 25%-ile, median, mean, 75%-ile, and max for the variable gdppc
summary(colonials$gdppc) ## just print
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     450    1480    2835    5445    6968   27330
```

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

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     450    1480    2835    5445    6968   27330
```

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.

```
## Let's look specifically at mean GDP
mean(colonials$gdppc) ## just print
```

```
## [1] 5445.458
```

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

```
## [1] 5445.458
```

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

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:

$$Y_i = \alpha + \beta X_{1,i} + \gamma X_{2,i} + ... + e_i$$

(Note: The above mathematical expression was written in a typesetting environment called LaTex, and can easily be included in RMarkdown. For examples, see the Script.) 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
```

```
##
## Call:
## lm(formula = logGDP ~ protection, data = colonials)
##
## Coefficients:
## (Intercept)   protection
##      4.6604       0.5221
```

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

```
##
## Call:
## lm(formula = logGDP ~ protection + lat_abst, data = colonials)
##
## Coefficients:
## (Intercept)   protection     lat_abst
##      4.7281       0.4679       1.5769
```

```
## Storing results
reg1 <- lm(logGDP ~ protection, data=colonials) # display regression results
reg2 <- 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.

```
## Now access regresion results by:
summary(reg1) # access complete regression results
```

```
##
## Call:
## lm(formula = logGDP ~ protection, data = colonials)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8715 -0.4644  0.1683  0.4610  1.1413
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.66038    0.40851  11.408  < 2e-16 ***
## protection   0.52211    0.06119   8.533 4.72e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.7132 on 62 degrees of freedom
## Multiple R-squared:  0.5401, Adjusted R-squared:  0.5327
## F-statistic: 72.82 on 1 and 62 DF,  p-value: 4.724e-12
```

Tables

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 include the final table in the knitted pdf created by RMarkdown, you can use the following template. (Note, we want to use results='asis' and header=FALSE, in order for RMarkdown to compile the LaTex table into a pretty format when knitting the pdf):

```
stargazer(reg1, reg2,
          out="Table 1",type="latex",header=FALSE,  table.placement = "!h",
          title="Property rights and Development",align=TRUE,
          report = "vc*st", omit.stat=c("LL","ser","f","rsq","adj.rsq"),no.space=TRUE)
```

Table 1: Property rights and Development

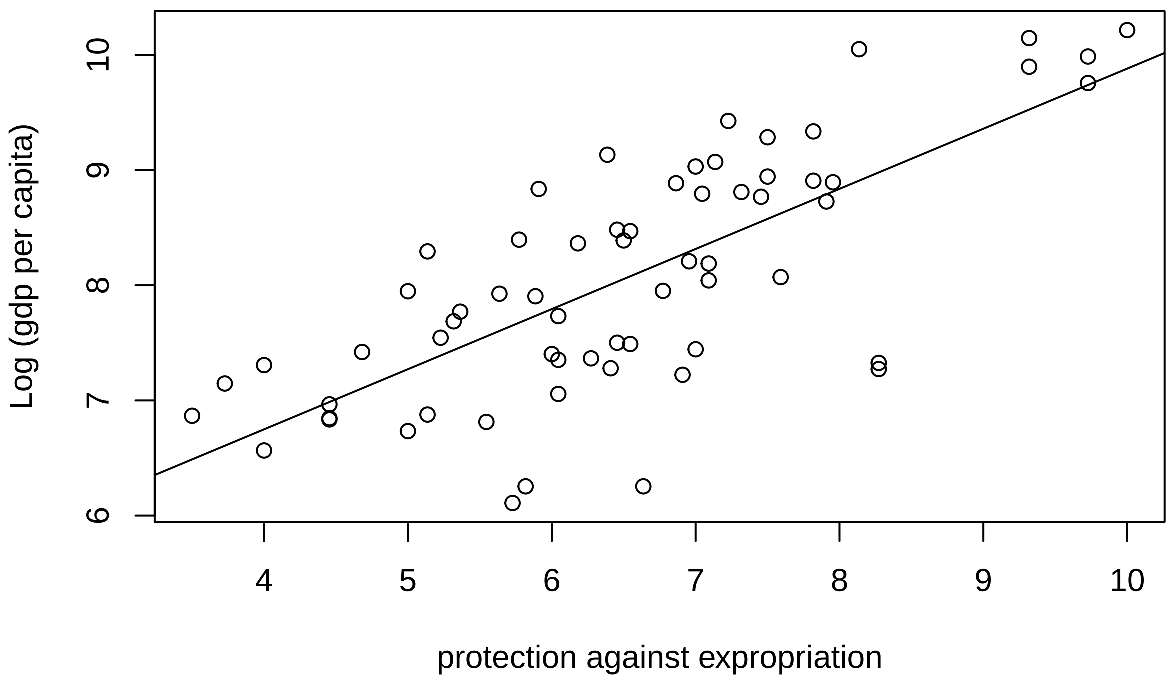
	Dependent variable:	
	logGDP	
	(1)	(2)
protection	0.522^{***}	0.468^{***}
	(0.061)	(0.064)
	$t = 8.533$	$t = 7.292$
lat_abst		1.577^{**}
		(0.710)
		$t = 2.220$
Constant	4.660^{***}	4.728^{***}
	(0.409)	(0.397)
	$t = 11.408$	$t = 11.900$
Observations	64	64
Note:	*p<0.1; **p<0.05; ***p<0.01	

The above command combines regression output saved in reg1 and reg2 into one table. Stargazer lets you specify title, alignment of coefficients inside columns and rows, which statistics to report (here, the variable names "v", coefficients "c", significance stars "*", standard errors"s" and t-statistics "t"). Moreover, you can tell it to omit statistics such as the log-likelihood "LL", the R2 "rsq", etc. You can also set the table placement in the final pdf (here, table.placement="H" means that the table will be placed directly after the code chunk). Stargazer has extensive documentation (try ??stargazer in R) and can also be used to make great summary statistics tables - try it out!

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.

```r
## A scatterplot with titles
reg2 <- lm(logGDP ~ protection , data=colonials)
plot(colonials$protection,colonials$logGDP,
     xlab="protection against expropriation", ylab="Log (gdp per capita)")
abline(lm(logGDP ~ protection , data=colonials)) # line of best fit
```



```r
## Remember that abline adds to the plot the regression line (linear fit line)
```

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.