Introduction to R

Introduction to Econometrics W3412

Begin

Download R from the Comprehensive R Archive Network (CRAN) by choosing a location close to you. Students are also recommended to download RStudio, a free and open-source user interface for R. RStudio is easier to use than the default RGui, especially for beginners. This introduction will refer to some features of RStudio, but all commands described should also work in RGui.

Create a folder in the My Documents folder called "r_recitation". Download auto.dta and example1.R from Courseworks into this folder. We will save all of our output into this folder.

Introduction to R

R is a programming environment that can be used to perform data analysis, data management, and graphics. This introduction should allow you to complete basic tasks. More detailed tutorials can be found at the following websites:

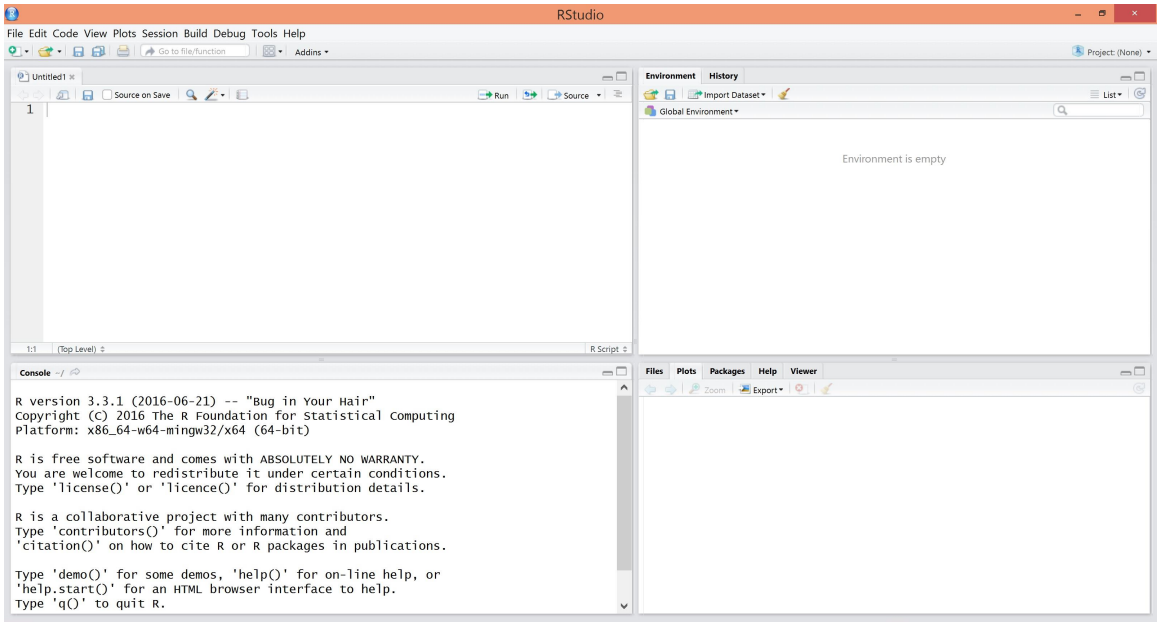
- http://data.princeton.edu/R/
- https://www.rochester.edu/college/psc/thestarlab/help/rcourse/R-Course.pdf
- http://www.ats.ucla.edu/stat/r/
- https://www.r-bloggers.com/how-to-learn-r-2/
- https://cran.r-project.org/manuals.html

This introduction borrows from these various tutorials.

1 Basics

1.1 Opening R

When you open R, you will see the *R Console*, where you can type in commands. The *R Console* is interactive; after you type in commands, R displays the results of your commands in the same window.



Opening RStudio instead, you will see two windows with multiple tabs, in addition to the *Console*. Under the *Environment* tab, you can view objects in the global environment. Under the *History* tab, you can view your past commands and re-use them. Using different tabs, you can open files and R scripts (*Files*), view plots (*Plots*), install and load packages (*Packages*), view help files (*Help*), and view local web contents (*Viewer*).

Open a new *R script* by going to (File > New script) in RGui, or (File > New File > R Script) in RStudio. You can type your commands here and save them as a separate file. More on this later.

1.2 The R Console

In the *Console*, type

```
2+2
```

R works as a calculator when you type in a mathematical expression. R evaluates the expression and shows the results in the *Console*.

The results of a calculation may be assigned to a named object. Type

```
x <- pi^2
```

```
x
```

You see that the object x now refers to the value of π squared.

In the *Console*, you can use the up and down arrows to access earlier commands. Note that R commands are case-sensitive. You can clear the *Console* by pressing Ctrl + l.

1.3 Writing commands

In R, you can construct commands by calling *functions*. R comes with a set of pre-installed functions, and you can install/load additional packages to access other named functions. All functions have names and take arguments in parentheses:

```
function(argument1,argument2,...)
```

For instance, the following command asks R to evaluate the square root of 2 and print the value in a sentence:

```
cat("y is",sqrt(2))
```

Commands can be separated by a semi-colon(;) or by starting a new line. The comment operator # makes R ignore everything written after it (in the current line).

1.4 Help

To get more information about any specific named function, such as `solve`, you can type:

```
help(solve)
```

or `?solve`

In some cases, you may not know the function names for the tasks you want to do. For example, you may want to calculate column means. You may search for a keyword in the documentations of installed packages by typing:

```
help.search("mean")
```

or `??mean`

Alternatively, you can get a list of all function names containing the search term:

```
apropos("mean")
```

You can also launch the overall R help page by typing:

```
help.start()
```

2 In Class Example

Let's do an example to practice using R.

2.1 Loading Data

2.1.a Directory

Type `getwd` to find R's current directory. This is the directory from which R opens or saves files if you do not specify an alternate directory.

You can change directories by typing `setwd(dir).`

Change the directory to the r_recitation folder:

```
setwd("D:/My Documents/r_recitation/")
```

From now on, we will use and save files in the r_recitation folder.

2.1.b Loading Packages

As mentioned above, various pre-written functions are available through additional packages. You can install and load the packages needed for your specific purpose. For example, we want to open auto.dta, but R does not know out of the box how to open a data set written in the Stata format. It turns out there is a package for opening data sets generated by other statistical programs. To install this package, type:

```
install.packages("foreign")
```

To use the functions in this package, you have to tell R to load the package's library each time you launch an R session:

```
library("foreign")
```

2.1.c Load Data

Clear all objects in memory by typing:

```
rm(list=ls(all=TRUE))
```

Load auto.dta using the following command:

```
auto=read.dta("auto.dta")
```

You can use files that are not in your current directory if you specify where the file is located.

Note that you can open datasets written using R's `save` function with `load()`; and open csv files with `read.csv()`. See help for other file formats.

2.2 Data Management and Analysis

2.2.a Describe the Data

To describe the data, type:

```
str(auto, give.attr=FALSE)
```

R will print the number of observations and variables. Then for each variable, the variable type and the first few observations are printed according to how the data is sorted.

In RStudio, you can also view this information by clicking on the small arrow next to the `auto` object under the *Environment* tab.

2.2.b Look at the Data

To look at the *make* of the first 5 observations, sorted by *make*, type:

```
head(auto, n=5)
```

In RStudio (only), you can view the dataset by clicking on the `auto` object under the *Environment* tab, or by typing: `view(auto)`

2.2.c Summarize the Data

To obtain summary statistics of the data, type:

```
summary(auto)
```

R will give the means, quartile values, minimum, maximum, and the number of missing values for all the numeric variables in the data. For example, the mean *mpg* is 21.3. For categorical data, R will give the frequency of the first few most frequent levels.

Note: *make* is a character variable (string). The variable *rep78* has 69 observations because the values of *rep78* are missing for some observations. R denotes missing with a "NA" for both numeric and character variables.

In order to obtain a more detailed summary of the variable *mpg*, install/load "psych" package and type:

```
describe(auto$mpg)
```

This command tells R to also report the standard deviation, skewness, kurtosis, etc. Since the skewness is positive, we know that *mpg* has a long right tail.

Note: to access variables in the data frame `auto`, we used the symbol $. We can also attach this data frame to the R search path so that we can refer to the variables in `auto` simply by their names:

```
attach(auto)
```

To summarize *mpg* for only foreign cars, type:

```
summary(subset(mpg,foreign=="Foreign"))
```

Note that we selected a subset of the data by a **logical statement** `(foreign==1)`:

The symbols for logical operators include:

Symbol	Meaning
`>`	Strictly greater
`<`	Strictly less
`>=`	Greater or equal
`<=`	Less or equal
`==`	Equal
`!=`	Not equal

To summarize *mpg* for domestic cars, type:

```
summary(subset(mpg,foreign!="Foreign"))
```

Alternatively, you could have applied summary functions to the data frame, broken down by group. For example, load "plyr" package and type:

```
ddply(auto, c("foreign"), summarise, N = length(mpg), mean = mean(mpg), sd = sd(mpg))
```

2.2.d Vectors, Matrices and Arrays

R is an object oriented language, which makes it great for working with vectors, matrices and arrays (more so than Stata). R can hold multiple data frames as well as many different-sized arrays as objects in memory. While this introduction does not go into detail on this topic, it is useful to learn how to work with arrays in R. You may refer to Chapter 3 and Chapter 4 in Peter Haschke's course notes.

2.2.e Graphs

To see the empirical distribution of *mpg*, type:

```
hist(mpg)
```

The graph shows us that *mpg* has a long right tail.

You can save this graph by typing:

```
jpeg("histogram_mpg.jpeg")
hist(mpg)
dev.off()
```

Make a scatterplot of *mpg* vs. *weight* by typing:

```
plot1 <- ggplot ( data = auto )
plot1 <- plot1 + geom_point( aes( x = weight , y = mpg ) )
```

The first line initiates a ggplot object. Then the second lines adds the scatter plot as a layer using the `geom_point` funciton. Other aspects of the fuction, such as labels, can be added as additional layers. The resulting scatterplot shows heavier cars have lower *mpg*.

2.2.f Generating Variables

The variable *weight* is in pounds. Let's generate a variable for weight in 1000s of pounds.

```
weightdiv1000 <- weight/1000
```

Summarize the variables *weight* and *weightdiv1000* to see that the variable was created correctly.

Note that some mathematical expressions in R include:

Symbol	Expression	Symbol	Expression
+	Add	^	power
-	Substract	sqrt()	square root
*	Multiply	exp()	exponential
/	Divide	log()	natural log

Generate a scalar object equal to the sample mean of *mpg*:

```
mpg_bar <- mean(mpg)
```

Some simple data manipulation can be done as follows (but there are also other ways):

Command	Description
df <- subset(df, select = -c(x,y))	drop variables named x and y from data frame 'df'
df <- subset(df, select = c(x,y))	drop every variable except x and y from 'df'
rm(x)	remove object x
x <- y	replace object x with object y
names(df)[names(df) == 'x'] <- 'y'	rename variable x "y" in 'df'

2.2.g Test of the Null Hypothesis

Suppose you wanted to test whether foreign and domestic cars have the same mean mpg.

The null hypothesis is: $H_0 : \mu_{domestic} - \mu_{foreign} = 0$

The alternative hypothesis is: $H_1 : \mu_{domestic} - \mu_{foreign} \neq 0$

Test this hypothesis using Student's t-test in R:

```
t.test(mpg ~ foreign, auto, var.equal=TRUE)
```

Due to the very small p-value, we reject the null hypothesis that foreign and domestic cars have the same mean mpg.

2.2.h Linear Regression

Do a linear regression of *mpg* on *weight*:

```
model1 <- lm( mpg ~ weight )
summary(model1)
```

You should see the following output:

```
Call:
lm(formula = mpg ~ weight)

Residuals:
    Min      1Q  Median      3Q     Max
-6.9593 -1.9325 -0.3713  0.8885 13.8174

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 39.4402835  1.6140031   24.44   <2e-16 ***
weight      -0.0060087  0.0005179  -11.60   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.439 on 72 degrees of freedom
Multiple R-squared:  0.6515,    Adjusted R-squared:  0.6467
F-statistic: 134.6 on 1 and 72 DF,  p-value: < 2.2e-16
```

Note that you can now obtain the fitted values for *mpg*:

```
mpghat <- fitted(model1)
```

and the residuals of the regression:

```
ehat <- residuals(model1)
```

which will be stored in the new objects *mpghat* and *ehat*.

Here's what your regression output means:

In the first section (**Call**), the estimated model is shown.

In the second section (**Residuals**), the minimum, maximum and quartile values of the estimated residuals are shown.

In the third section (**Coefficients**), there is the table for the regression coefficients.

```
    [a]          [b]         [c]       [d]       [e]    [f]
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 39.4402835  1.6140031   24.44   <2e-16 ***
weight      -0.0060087  0.0005179  -11.60   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

[a] This column shows the the predictor variables (*Intercept* and *weight*). The first row (*Intercept*) represents the constant or intercept.

[b] **Estimate** - These are the values for the regression equation for predicting the dependent variable from the independent variable(s). The regression equation is presented in many different ways, for example: $Y_{predicted} = b_0 + b_1 * x_1$

The column of estimates provides the values for b_0 and b_1 for this equation.

The coefficient on *weight* is -.0060087. So for every one pound **increase** in weight, a - .0060087 unit **decrease** in *mpg* is predicted, holding all other variables constant.

[c] **Std. Error** - These are the standard errors associated with the coefficients.

[d] **t value** - These are the t-statistics used in testing whether a given coefficient is significantly different from zero.

[e] **Pr(>| t |)** - This column shows the 2-tailed p-values used in testing the null hypothesis that the coefficient (parameter) is 0. Using an alpha of 0.05: the coefficient for *weight* is significantly different from 0 because its p-value is 0.000, which is smaller than 0.05; the constant (*Intercept*) is also significantly different from 0 at the 0.05 alpha level.

[f] The stars indicate whether the coefficients are significant at different levels of alpha. As indicated by **Signif. codes** below, '***' indicate significance at the 0.001 level, '**' at 0.01, '*' at 0.05, and '.' at 0.1.

In the last section, there are descriptions of overall model fit. To better understand these, let's first look at the Analysis of Variance (ANOVA) Table:

```
anova <- anova(model1)
```

```
          [g]      [h]     [i]      [j]
                    Df  Sum Sq Mean Sq F value    Pr(>F)
weight       1 1591.99 1591.99  134.62 < 2.2e-16 ***
Residuals 72  851.47   11.83
```

[g] The total variance can be partitioned into the variance that can be explained by the independent variables (Model) and the variance that is not explained by the independent variables (the Residual, sometimes called Error).

[h] **Df** - These are the degrees of freedom associated with the sources of variance. The total variance has $N - 1$ degrees of freedom. The Model degrees of freedom corresponds to the number of coefficients estimated minus 1. Including the intercept, there are 2 coefficients, so the model has $2 - 1 = 1$ degree of freedom. The Residual degrees of freedom is the DF Total minus the DF Model, $73 - 1 = 72$.

[i] **Sum Sq** - These are the Sum of Squares associated with the sources of variance: independent varaibles (Model), and Residuals.

[j] **Mean Sq** - These are the Mean Squares, the Sum of Squares divided by their respective Df.

Now we can interpret the descriptions of ovall model fit.

```
[k] Residual standard error: 3.439 on 72 degrees of freedom
[l] Multiple R-squared:  0.6515, [m]Adjusted R-squared:  0.6467
[n] F-statistic: 134.6 on 1 and 72 DF[o] p-value: < 2.2e-16
```

[k] **Residual standard error** - This is the standard deviation of the error term, and is the square root of the Mean Squares of the Residual ($\sqrt{(11.83)} = 3.439$).

[l] **Multiple R-squared** - This is the proportion of variance in the dependent variable (mpg) which can be explained by the independent variable ($weight$). This is an overall measure of the strength of association.

[m] **Adjusted R-squared** - This is an adjustment of the R-squared that penalizes the addition of extraneous predictors to the model. Adjusted R-squared is computed using the formula $1 - ((1 - Rsq)((N - 1)/(N - k - 1))$ where k is the number of predictors.

[n] **F-statistic** - F-statistic is the Mean Squares of the Model (1591.99) divided by the Mean Square of the Residuals (11.83), yielding F=134.6. The Model and Residual degrees of freedom are also reported.

[o] **p-value** - This is the p-value associated with the above F-statistic. It is used in testing the null hypothesis that all of the model coefficients are 0.

2.2.i Save

Save the data to a new dataset named auto2. Alternatively you can overwrite auto.dta by using the second command.

```
save(auto, file = "auto2.Rdata" )
write.dta(auto, file = "auto.dta" )
```

2.2.j Exit R

Exit R (without saving the workspace) by typing:

```
q(save="no")
```

2.2.k Writing R Scripts

Instead of typing your commands one by one, you can write a script and run all those commands at once (like Stata's do file).

A sample script that corresponds to the In Class Example is provided on courseworks (r_example1.R). You have already downloaded this file. This file was created and can be edited in a text editor.

When you save this file, make sure you change the extension to *.R. It will not work if it has a different extension.

Open a new script. From the R Editor, open (r_example1.R) and review it. You can execute the entire script by typing:

```
source('r_example1.r')
```

```
source('r_example1.r', print.eval=TRUE)
```

The first command executes the commands without printing the results, while the second prints the results.

2.2.1 Compiling Notebook

You may want to keep a permanent record of your results (like Stata's log file). You can do this in RGui using the `sink()` function. RStudio provides a neat way to do this using a script. Open r_example1.R in RStudio, and go to (File > Compile Notebook). All the commands in the script are executed and the results are printed into a HTML, pdf or Word document.