Module 1: R Basics

Contents

1. INTRODUCTION

The goal of this module is to introduce R, the statistical software that we will use for this course. R is a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. While we hope you appreciate the ease and possible rigor of analysis in R, you should know that it is not the only statistical software capable of conducting the analysis presented in this course; other examples of statistical packages include SAS and STATA. However, technical support will be provided only for R in this course**. Don't be scared of R; it is one of the easiest statistical packages available and we are there to support you!**

The objective of this module is to provide an overview of R and enable you to perform some basic operations in R. For more systematic basic- and intermediate-level training in R, you may access the following free online resources available from Princeton University:

- ✓ http://data.princeton.edu/R/
- ✓ http://dss.princeton.edu/training/

UCLA also offers free online instruction material for statistical analysis using R (https://stats.idre.ucla.edu/r/),and the D-lab at UC Berkeley frequently offers free courses on R (http://dlab.berkeley.edu/).

Also, while we will cover some basic tools of statistical analysis, this course assumes a prior level of competency in the statistical theories related to hypothesis testing and regression models. If you wish to review these subjects, we recommend taking one of the free online courses offered by UC Berkeley (http://webcast.berkeley.edu/), Princeton University (http://dss.princeton.edu/training/), MIT (http://ocw.mit.edu/index.htm), or your university.

Data processing, or preparing datasets for analysis, is an important precursor to data analysis. While we will use simplified datasets for pedagogical purposes, in real life "painful" datasets are prevalent. For this reason, some of the problem sets in this and other modules will expose you to "real-life datasets" and offer tips to deal with common problems such as missing data, non-response and attrition, an over-abundance of variables, and multiple datasets. We will cover some basic data processing and management tasks in this module, and then will offer tips and guidance as required in subsequent modules.

How to use this learning guide: We recommend first going through the learning guide quickly to assess how conversant you are with the concepts, tools and methods described. If you are not conversant with the presented material, then we recommend reading through the learning guide carefully and completing the short exercises.

At the end of this session, you should be able to:

- ✓ Understand the setup and basic components of R
- ✓ Perform basic data processing and management operations in R

2. RSET UP

2.1 Obtaining R and R Studio

R is open source and is available for free to everyone! You can download it from CRAN for Windows here, Mac OS here. After download and installation, R can be accessed through the command line interface by typing the command R. The industry standard and common best practice to use R is through R Studio, an integrated development environment (IDE) which provides a Graphical User Interface (GUI). It includes a console, a syntax-highlighting editor as well as tools for plotting, history, debugging and workspace management. R Studio can be downloaded and installed from here.

2.2 R Studio Interface

Once R Studio is installed, it can be started from the Program Menu of your PC or by double-clicking the R Studio shortcut that may be available on your desktop. The R Studio interface has 5 windows, as shown in Figure 2 and described below:

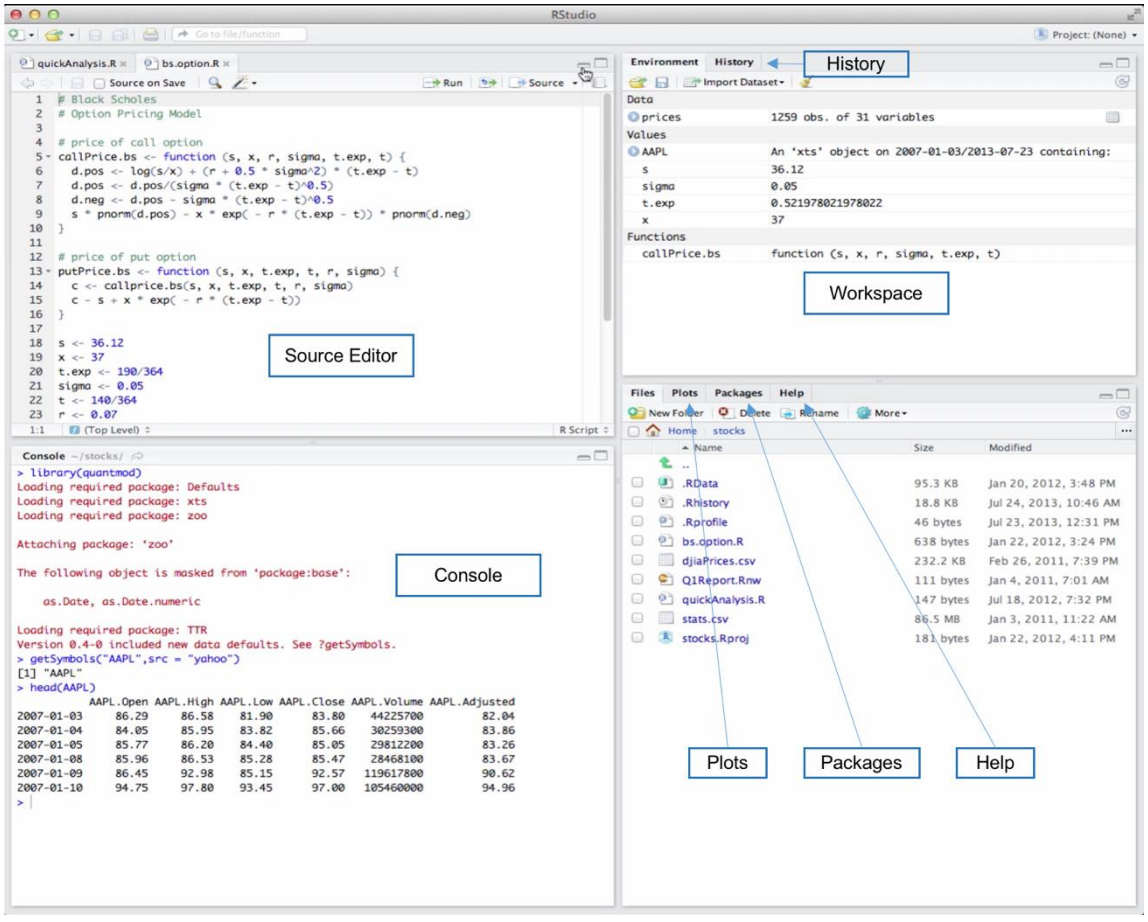


Figure 1. R Studio main window

- ✓ Console: where you can interactively run R commands and immediately see the output.
- ✓ Source Editor: where you can write R scripts. You can run the whole script or one line at a time and check the outputs in the console
- ✓ Workspace: where you can view objects in the global environment. This where you can see the data you have read and variables you have created. You can click on the objects to view them in the Data Viewer
- ✓ Searchable Command History: where can view history in context, search it and reuse your code.
- ✓ Plots pane: where you can view your plots and export them from
- ✓ Packages Window: Package manager to view, search, install and update packages
- ✓ Integrated R Help: Documentation for different commands, packages and functions in R

Each of the four panes can be minimized while not being used or resized according to your needs.

2.3 R Help

Like most software, the best way to learn R is to practice and experiment with it. There are two main ways to get help in R Studio, type `?<function/package name>` or `help(<function/package name>)` in the console. Alternatively, you can also search for the topic/command/function in the Integrated R help tab shown above. Figure 2 is the screenshot of the documentation for the lm ("`help(lm)`") function in the in-built stats package, when run on the console.

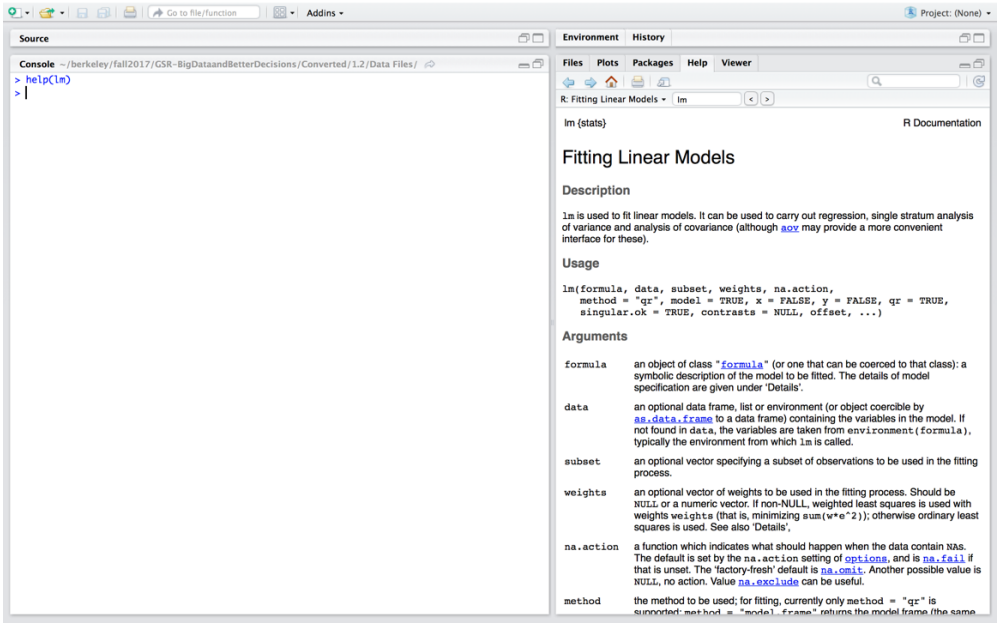


Figure 2. Help file for the lm function

Calling help opens a page (this depends on the operating system) with general information on the first line such as the name of the package where is (are) the documented function(s) or operators. Then comes a title followed by sections which give detailed information on the description, usage, arguments, details and examples. Examples are useful for beginners to get started with new commands.

The help In html format can be called by typing help.start(). A search with keywords is possible with this html help using the search bar while in the Help tab.

Exercise: One of the first things you will be learning to do is to set your working directory using the setwd function
Using help commands we learned earlier lookup the help page of setwd and use it to set your working directory to the location of MyFirstData.csv

Answer Key: `?setwd`
`setwd("< full or relative path of location>")`

Exercise: Explore the Packages tab and look through the pre-installed packages.
Install ggplot2, a graphics system for that we will be using for visualizing data

Answer Key: `Use the Install Button to install the package or use the command install.packages("ggplot2")`

2.4 R Scripts

A key advantage of statistical software like R is clear documentation of its analysis steps and their reproducibility. Therefore, although R users can either issue commands directly on the console or write commands in the source editor for running all at once and for reproduction and reuse of code.

One often runs commands by directly typing and executing though the console or by using the by writing the commands in the editor and executing them all at once or line by line. To execute a command in the source editor, select the command and click on the Run icon on the top left of the source editor. The keyboard shortcut to run commands is Cmd + Enter (Mac) or Ctrl + Enter (Windows)

While writing an R script, it is important to include comments to describe and document the assumptions and decisions that went into your code or to explain the analysis steps. A comment in R begins with a # and nothing happens on executing a comment line. Example:

```
# Clearing workspace
rm(list = ls())
```

The above line has a comment that describes the command. The rm function is used to remove objects from the workspace and takes in the name of the object or a list of objects as its argument. Tip : Use the help command `?rm or help(rm)` to learn more about the rm command. `ls()` lists all the objects in the workspace which we pass as an argument to the rm function, thereby clearing the workspace.

Other good practices to be aware of while writing R scripts are to set working directory and load the packages required for the script on top of the script.

- Working Directory: The working directory is the folder that contains the location of the file/files you want to read into R. As seen earlier, the working directory can be set using the `setwd()` command
- Load Packages : In addition to many in built commands, R also has many packages which are repositories of useful functions that can be installed and used as needed. A list of already installed packages can be viewed in the Packages

tab and new packages can be installed using the Install button or using the command `install.packages`. Once a package is installed, it should be loaded to be used in the session. This is done using the `library` or `require` functions. Once the packages are loaded, the functions defined within them can be used in the R session.

```
# Loading required packages
require(ggplot2)
```

The list of packages available by CRAN can be found here - https://cran.r-project.org/web/packages/available_packages_by_name.html

2.5 Opening and Saving Datasets

For reading and writing in files, R uses the working directory. To find this directory, the command `getwd()` (get working directory) can be used, and the working directory can be changed with `setwd` (as seen above). It is necessary to give the path to a file if it is not in the working directory.

Reading Data into R

R can read data stored in text (ASCII) files with the following functions: read.table (which has several variants, see below), scan and read.fwf. R can also read files in other formats (Excel, SAS, SPSS, . . .), and access SQLtype databases, but the functions needed for this are not in the base package. These functionalities are very useful for a more advanced use of R, but we will restrict here to reading files in ASCII format. The function read.table has for effect to create a data frame, and so is the main way to read data in tabular form. R also has its own data file formats with extensions .rdata and .rda which can be read and written using the functions `load` and `save`. For most of our class, we will be reading CSV (Comma-Separated Values) files into R and working on them.

For reading "rectangular data", where the data fits nicely into a rectangle of rows and columns, the function `read.table` is often used. It takes in the name of the file as the first argument and also has an argument for the separator (sep=), among others. Check out the documentation and examples for the `read.table` function. For CSV files, we use a variant of this function called read.csv in the following way.

```
?read.table
MyFirstData <- read.csv('MyFirstData.csv', header = TRUE)
```

This loads the data into a data frame named MyFirstData and the first row of the file will be the column names. We will see more on how to use data in data frames in the subsequent sections.

Exercise: Once you execute the `read.csv` command, click on the MyFirstData data frame that appears in the workspace and look through the data when it opens in the data viewer. Sometimes when our data is huge, it is helpful to get an idea of the structure of the data in a concise way. Execute the following command and observe the results

```
str(MyFirstData)
```

What information does R provide about the dataset? Also try
`summary(MyFirstData)`
What can we understand about the data from both these commands?

Writing Data

Often while doing analysis on data, you will need to create and save datasets derived from original datasets by subsetting them and/or creating derived columns etc. You can save a dataset from R into txt files or other formats using the `write.table, write.csv` and other functions.

Tips

- ✓ Use the console to try out new or unfamiliar commands and interpreting the results before including them in your script
- ✓ Instead of always writing the entire file path when you save or open data, especially if you frequently change the folder in which the data is stored, it is often more convenient to set working directory to the one with your data files and reference them by name. Fun fact – the working directory can also be set from the Session menu on the menu bar of R Studio
- ✓ Include the following two lines at the start of your script file. These help to start your script with a clean workspace and console

```
rm(list=ls())
cat('\014')
```

- ✓ In large datasets with thousands of variables, browsing for variables can be difficult. The `colnames` function can be used to list the column names of the dataframe. Try `colnames(MyFirstData)`
- ✓ Always comment your scripts for easy understanding and reuse in the future!

3. DATA PROCESSING

3.1 Creating Data

Regular sequences

- ✓ A regular sequence of integers, for example from 1 to 30, can be generated with the following command. The resulting vector x has 30 elements. <- is called the assign operator in R and is used to assign values/objects to variables. The : opera

```
x <- 1:30
```

- ✓ The function seq can generate sequences of real numbers as follows:

```
x_seq <- seq(1, 5, 0.5)
```

where the first number indicates the beginning of the sequence, the second one the end, and the third one the increment to be used to generate the sequence.

- ✓ One can also type directly the values using the function c:

```
x_vector <- c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
```

- ✓ It is also possible, if one wants to enter some data on the keyboard, to use the function scan with simply the default options:
  `z <- scan()`

- ✓ The function rep creates a vector with all its elements identical:
  `rep(1, 30)`

- ✓ The function gl (*generate levels*) is very useful because it generates regular series of factors. The usage of this function is gl(k, n) where k is the number of levels (or classes), and n is the number of replications in each level. Two options may be used: length to specify the number of data produced, and labels to specify the names of the levels of the factor. Examples:

```
> gl(3, 5)
> gl(3, 5, length=30)
> gl(2, 6, label=c("Male", "Female"))
```

Random sequences

Law	function
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor(F)	`rf(n, df1, df2)`
Pearson ($\chi 2$)	`rchisq(n,df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistic	`rwilcox(nn, m, n), rsignrank(nn, n)`

It is useful in statistics to be able to generate random data, and R can do it for a large number of probability density functions. These functions are of the form `rfunc(n, p1, p2, ...)`, where `func` indicates the probability distribution, n the number of data generated, and p1, p2, ... are the values of the parameters of the distribution. The above table gives the details for each distribution, and the possible default values.

Most of these functions have counterparts obtained by replacing the letter r with d, p or q to get, respectively, the probability density (`dfunc (x, ...)`), the cumulative probability density (p*func* `(x, ...)`), and the value of quantile (q*func* `(p, ...)`, with $0 < p < 1$). The last two series of functions can be used to find critical values or *P* -values of statistical tests. For instance, the critical values for a two-tailed test following a normal distribution at the 5% threshold are:

```
> qnorm(0.025)
```

```
> qnorm(0.975)
```

For the one-tailed version of the same test, either qnorm(0.05) or 1 - qnorm(0.95) will be used depending on the form of the alternative hypothesis.
The P -value of a test, say $\chi^2 = 3.84$ with $df = 1$, is:

```
> 1 - pchisq(3.84, 1)
```

Objects

- ✓ *Vector*. The function vector, which has two arguments mode and length, creates a vector which elements have a value depending on the mode specified as argument: 0 if numeric, FALSE if logical, or "" if charac- ter. The following functions have exactly the same effect and have for single argument the length of the vector: numeric(), logical(), and character().

- ✓ *Factor*. A factor includes not only the values of the corresponding categorical variable, but also the different possible levels of that variable (even if they are not present in the data). The function factor creates a factor with the following options:

```
factor(x, levels = sort(unique(x), na.last = TRUE),
       labels = levels, exclude = NA, ordered = is.ordered(x))
```

  levels specifies the possible levels of the factor (by default the unique values of the vector x), labels defines the names of the levels, exclude the values of x to exclude from the levels, and ordered is a logical argument specifying whether the levels of the factor are ordered. Recall that x is of mode numeric or character. Some examples follow.

```
> factor(1:3)
> factor(1:3,
levels=1:5)
> factor(1:3, labels=c("A", "B", "C"))
> factor(1:5,
exclude=4)
```

- ✓ *Data frame*. We have seen that a data frame is created implicitly by the function read.table; it is also possible to create a data frame with the function data.frame. The vectors so included in the data frame must be of the same length, or if one of the them is shorter, it is "recycled" a whole number of times:

```
> x <- 1:4; n <- 10; M <- c(10, 35); y <- 2:4
> data.frame(x,
n)
> data.frame(x,    M)
>  data.frame(x,   y) # this should throw an error
```

  If a factor is included in a data frame, it must be of the same length than the vector(s). It is possible to change the names of the columns with, for instance, data.frame(A1=x, A2=n). One can also give names to the rows with the option row.names which must be, of course, a vector of mode character and of length equal to the number of lines of the data frame. Finally, note that data frames have an attribute dim similarly to matrices.

- ✓ *List.* A list is created in a way similar to data frames with the function list. There is no constraint on the objects that can be included. In contrast to data.frame(), the names of the objects are not taken by default; taking the vectors x and y of the previous example:

```
> L1 <- list(x, y); L2 <- list(A=x, B=y)
> L1
> L2
```

Other objects like matrices, time series, expressions etc. also exist in R but are not in the scope of this course.

Converting between objects –
The reader has surely realized that the differences between some types of objects are small;
it is thus logical that it is possible to convert an object from a type to another by changing some of its attributes. Such a conversion will be done with a function of the type `as.something`.
The result of a conversion depends obviously of the attributes of the converted object.
Genrally, conversion follows intuitive rules.
For the conversion of modes, the following table summarizes the situation.

Conversion to	Function	Rules
numeric	`as.numeric`	`FALSE` → 0
		`TRUE` → 1
		`"1"`, `"2"`, ... → 1, 2, ...
		`"A"`, ... → `NA`
logical	`as.logical`	0 → `FALSE`
		other numbers → `TRUE`
		`"FALSE"`, `"F"` → `FALSE`
		`"TRUE"`, `"T"` → `TRUE`
		other characters → `NA`
character	`as.character`	1, 2, ... → `"1"`, `"2"`, ...
		`FALSE` → `"FALSE"`
		`TRUE` → `"TRUE"`

3.2 Typical Operators

Operators					
Arithmetic		Comparison		Logical	
+	addition	<	lesser than	! x	logical NOT
-	subtraction	>	greater than	x & y	logical AND
*	multiplication	<=	lesser than or equal to	x && y	id.
/	division	>=	greater than or equal to	x | y	logical OR
^	power	==	equal	x || y	id.
%%	modulo	!=	different	xor(x, y)	exclusive OR
%/%	integer division				

Exercise: Type `(347 * 821) ^ 0.5` on the console. Did you get 533.74807? This command is for quick on-screen calculations. You are able to use results from regression analysis or summary statistics in such calculations, but that will be covered in a future module.

3.3 Commands to Understand the Data

✓ **str**

This function is used to describe the structure of the dataset. The basic format of the function is:

```
str(object, ...)
```

For a data frame, str gives the number of rows and columns, and a sample of each columns' values along with the datatype.

```
> str(MyFirstData)
'data.frame':   1200 obs. of  8 variables:
 $ villid   : int  1001106 1001106 1001106 1001106 1001106 1001106 1001106 1001106 1001106 1001
106 ...
 $ hogid    : Factor w/ 1200 levels "0101103002.0639",..: 10 11 12 13 14 15 16 17 18 19 ...
 $ D_HH     : int  1 1 1 1 1 1 1 1 1 ...
 $ IncomeLab: int  NA NA NA 3200 NA 4320 4800 NA NA 3200 ...
 $ famsize  : int  6 6 6 5 5 5 5 6 6 3 ...
 $ agehead  : int  29 43 43 25 40 40 39 45 42 22 ...
 $ sexhead  : Factor w/ 2 levels "Female","Male": 1 2 2 1 2 2 2 1 2 2 ...
 $ pov_HH   : Factor w/ 2 levels "no pobre","pobre": 2 1 1 2 1 1 1 2 1 2 ...
```

Figure 6. Output of str function

✓ **summary**

This function provides the basic descriptive statistics and frequencies in the object.

```
summary(object)
```

This is often not only used for data frames but also for model objects to understand the model coefficients and significance values

✓ **nrow**

This function returns the number of rows in your object which is a vector, array or data frame.

```
nrow(MyFirstData)
```

✓ **colnames**

This function returns the column names of the dataset as a character vector. Column names can also be changed by accessing the relevant item in the list and assigning it to the new name. Once we learn about accessing data in the next section, we will learn to do this

```
colnames(MyFirstData)
```

3.4 Accessing Data

The indexing system is an efficient and flexible way to access selectively the elements of an object; it can be either numeric or logical. To access, for example, the third value of a vector x, we just type x[3] which can be used either to extract or to change this value:

```
> x <- 1:5
> x[3]
> x[3] <- 20
> x
```

The index itself can be a vector of mode numeric:

```
> i <- c(1, 3)
> x[i]
```

If x is a matrix or a data frame, the value of the i th line and j th column is accessed with x[i, j]. To access all values of a given row or column, one has simply to omit the appropriate index (without forgetting the comma!)

Exercise : Access the third row and fifth column of MyFirstData

```
MyFirstData[3,5]
```

It may be useful to keep in mind that indexing is made with square brackets, while parentheses are used for the arguments of a function:

```
> x(1)
Error: couldn't find function "x"
```

Indexing can also be used to suppress one or several rows or columns using negative values. For example, `x[-1, ]` will suppress the first row, while `x[-c(1, 15), ]` will do the same for the 1st and 15th rows. Using the matrix defined above:

Logical indexing can also be used with data frames, but with caution since different columns of the data frame may be of different modes.

```
> MyFirstData[20, -1] # this will return the 20th row without the first column
```

For matrices and data frames, *colnames* and *rownames* are labels of the columns and rows, respectively. They can be accessed either with their respective functions, or with dimnames which returns a list with both vectors.

```
> X <- matrix(1:4, 2)
>  rownames(X) <-  c("a", "b")
>  colnames(X) <-  c("c", "d")
> dimnames(X)
```

If the elements of an object have names, they can be extracted by using them as indices. Actually, this should be termed 'subsetting' rather than 'extraction' since the attributes of the original object are kept. For instance, if a data frame DF contains the variables x, y, and z, the command DF["x"] will return a data frame with just x; DF[c("x", "y")] will return a data frame with both variables. This works with lists as well if the elements in the list have names.

To extract a vector or a factor from a data frame, on can use the operator $ (e.g., DF$x). For instance, to access the agehead column in MyFirstData, we use `MyFirstData$agehead` and to find the mean agehead we use `mean(MyFirstData$agehead)` This syntax can be extended to creating a new column as well.

3.5 Commonly used functions

`sum(x)`	sum of the elements of x
`prod(x)`	product of the elements of x
`max(x)`	maximum of the elements of x
`min(x)`	minimum of the elements of x
`which.min(x)`	returns the index of the smallest element of x
`which.max(x)`	returns the index of the greatest element of x
`range(x)`	id. than c(min(x), max(x))
`length(x)`	number of elements in x
`mean(x)`	mean of the elements of x
`median(x)`	median of the elements of x
`var(x) or cov(x)`	variance of the elements of x (calculated on $n - 1$); if x is a matrix or a data frame, the variance-covariance matrix is calculated
`cor(x)`	correlation matrix of x if it is a matrix or a data frame (1 if x is a vector)
`var(x, y) or cov(x, y)`	covariance between x and y, or between the columns of x and those of y if they are matrices or data frames
`cor(x, y)`	linear correlation between x and y, or correlation matrix if they are matrices or data frames
`round(x, n)`	rounds the elements of x to n decimals
`rev(x)`	reverses the elements of x
`sort(x)`	sorts the elements of x in increasing order; to sort in decreasing order: rev(sort(x))
`rank(x)`	ranks of the elements of x
`log(x, base)`	computes the logarithm of x with base base
`scale(x)`	if x is a matrix, centers and reduces the data; to center only use the option center=FALSE, to reduce only scale=FALSE (by default center=TRUE, scale=TRUE)
`pmin(x,y,...)`	a vector which *i*th element is the minimum of x[i], y[i], . . .
`pmax(x,y,...)`	id. for the maximum
`cumsum(x)`	a vector which *i*th element is the sum from x[1] to x[i]
`cumprod(x)`	id. for the product
`cummin(x)`	id. for the minimum
`cummax(x)`	id. for the maximum
`match(x, y)`	returns a vector of the same length than x with the elements of x which are in y (NA otherwise)
`which(x == a)`	returns a vector of the indices of x if the comparison operation is true (TRUE), in this example the values of i for which x[i] == a (the argument of this function must be a variable of mode logical)
`choose(n, k)`	computes the combinations of k events among n repetitions = $n!/[(n-k)!k!]$

`na.omit(x)`	suppresses the observations with missing data (NA) (suppresses the corresponding line if x is a matrix or a data frame)
`na.fail(x)`	returns an error message if x contains at least one NA
`unique(x)`	if x is a vector or a data frame, returns a similar object but with the duplicate elements suppressed
`table(x)`	returns a table with the numbers of the differents values of x (typically for integers or factors)
`table(x, y)`	contingency table of x and y
`subset(x, ...)`	returns a selection of x with respect to criteria (..., typically com- parisons: x$V1 < 10); if x is a data frame, the option select gives the variables to be kept (or dropped using a minus sign)
`sample(x, size)`	resample randomly and without replacement size elements in the vector x, the option replace = TRUE allows to resample with replace- ment
`aggregate(forumla, data, FUN)`	aggregate based on the function
`ifelse(condition, yes, no)`	the condition is evaluated and yes denotes the value returns when the condition is true and no denotes the value for when the condition is false

Exercise: create a new variable named poor_male which is 1 if the household is categorized as poor (pov_HH==1) and the head of the household is male (sexhead is Male), and 0 otherwise. What is the frequency poor males?

Answer :

```
?ifelse
MyFirstData$poor_male <- ifelse(MyFirstData$pov_HH == 'pobre' &
MyFirstData$sexhead == 'Male', 1, 0)
?table
table(MyFirstData$poor_male)
```

4. VISUALIZING THE DATA

Visual representations of data often help researchers form hypotheses, identify problems in the data distribution, select appropriate analysis methods, and present the results in easier and more appealing formats. R has excellent capabilities to produce figures, charts or tables, but it could take you significant time before you learn to apply them. Here, in addition to the plotting functions available in the in-built *graphics* package in R, we will also learn about a widely used graphics system for R called *ggplot2*.

4.1 Introduction to ggplot2

ggplot2 is based on the grammar of graphics, the idea that you can build every graph from the same few components: a data set, a set of geoms—visual marks that represent data points, and a coordinate system. This cheat sheet by R Studio and this tutorial from Harvard are excellent resources to understand and use ggplot2. Here we will describe the basic building blocks of the library and its usage.

The basic idea of the grammar of graphics is this - independently specify plot building blocks and combine them to create just about any kind of graphical display you want. Building blocks of a graph include:

- data
- aesthetic mapping
- geometric object
- statistical transformations
- scales
- coordinate system
- position adjustments
- faceting

The essential building blocks that are necessary are the data, the aesthetic mapping and the geometric object which we will describe below.

Aesthetic Mapping

In ggplot land aesthetic means "something you can see". Examples include

- position (i.e., on the x and y axes)
- color ("outside" color)
- fill ("inside" color)
- shape (of points)
- linetype
- size

Each type of geom accepts only a subset of all aesthetics–refer to the geom help pages to see what mappings each geom accepts. Aesthetic mappings are set with the `aes()` function.

Geometic Objects (geom)

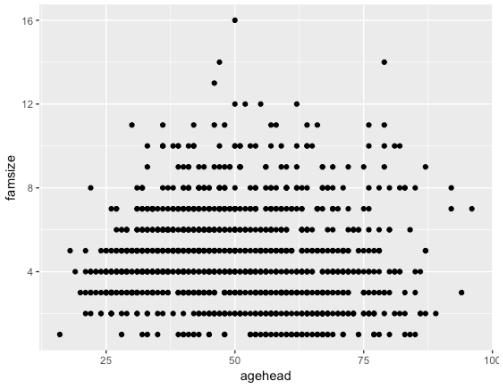
Geometric objects are the actual marks we put on a plot. Examples include:

- points (`geom_point`, for scatter plots, dot `plots`, etc)
- lines (`geom_line`, for time series, trend lines, etc)
- boxplot (`geom_boxplot`, for, well, boxplots!)

A plot must have at least one geom; there is no upper limit. You can add a geom to a plot using the + operator

Example of simple *scatter plot* of agehead and famsize using

```
ggplot(MyFirstData, aes(x = agehead, y=famsize)) + geom_point()
```



For the following plots, we will look at the base graphics way to construct the plot as well as constructing them using ggplot2

4.2 Histograms

A histogram is a graphical bar representation of a variable, in which the height of each bar is proportional to the frequency with which the values represented by that bar appear in the data.

Base graphics

```
hist(famsize, col="blue")
```

ggplot2

```
ggplot(MyFirstData, aes(x = famsize)) + geom_histogram(fill = "blue")
```

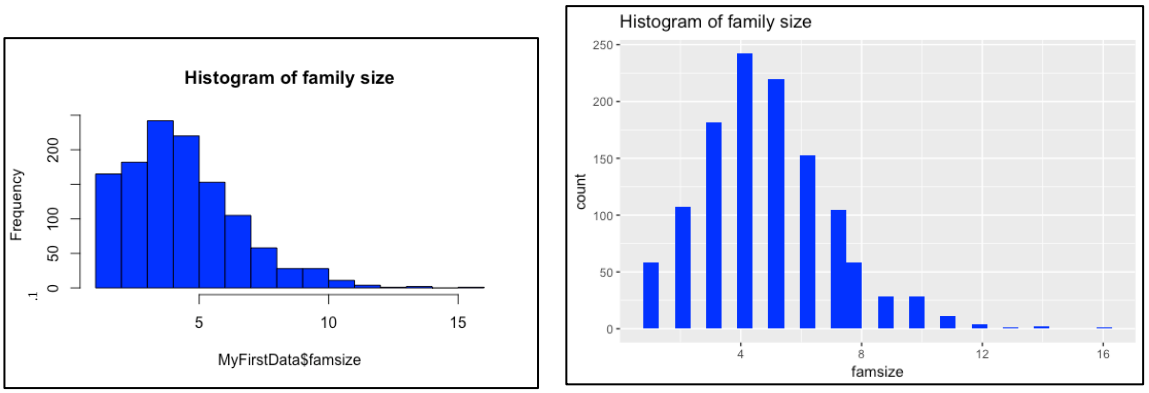


Figure 12. Output of hist and ggplot functions for histogram

4.3 Kernel Densities

Kernel densities are an alternative and popular way of visualizing variables' distributions. The main advantage of kernel densities over histograms is that rather than giving equal weight to each bar, the kernel gives more weight to data that are close to the point of reference and gives you a smoother line plot. The usual syntax for density plots is,

Base graphics

```
plot(density(MyFirstData$IncomeLab, na.rm = TRUE), main = "Density of Income")
```

ggplot2

```
ggplot(MyFirstData, aes(x = famsize)) + geom_histogram(fill = "blue")
```

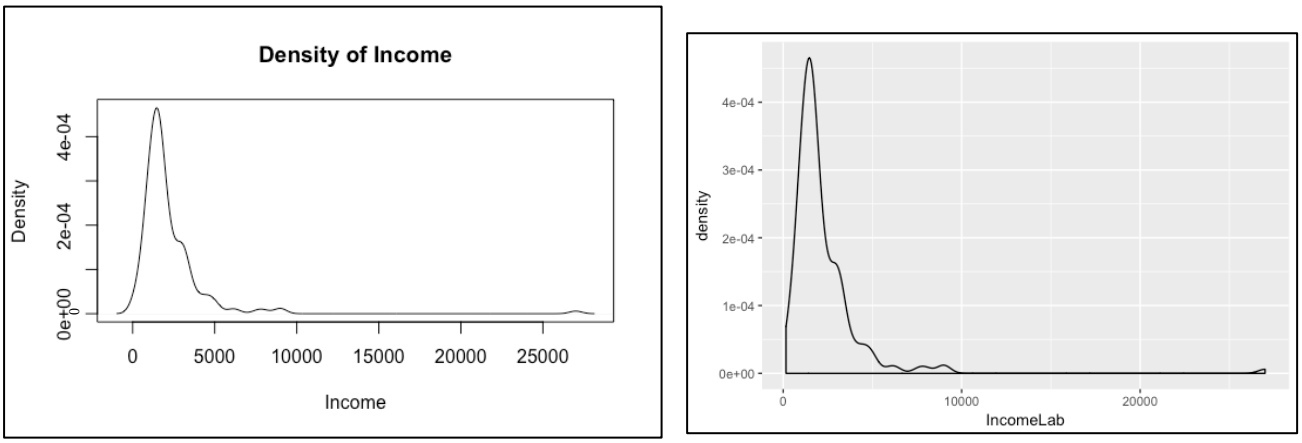


Figure 13. Output of density plots - first using base graphics and the next using ggplot2

The kernel density plot becomes smoother as the bandwidth increases, but the fit to the data is poorer compared to the smaller bandwidth.

5. BASIC REGRESSION ANALYSIS

At its core, regression analysis is a tool that estimates the relationship between an outcome (or dependent) variable and a set of predictor variables (or independent variables, also referred to as covariates). Although the covariates can be transformed by log, power or other non-linear transformation, the regression model is itself linear and additive as follows.

$$Y_i = \beta_0 + \beta_1 X_i + \sum_j (\beta_j K_j)_i + \varepsilon_i$$

The coefficient β_1 indicates the associated change in output *Y* for a unit change in input *X* when all other variables *K* are held constants. To provide unbiased estimates of the β coefficients, called marginal effects in econometrics, several key regression model assumptions must hold true. When these assumptions are not initially met, more complex regression models and other methods may have to be used. In these cases, applying regression analysis calls for considerable

skills and theory-based knowledge. In this module, we only demonstrate how regressions models are specified in R and explain the output.

In R, the `lm()`, or "linear model," function can be used to create a simple regression model.

The lm function accepts a number of arguments (use the `help(lm)` to explore). The following list explains the two most commonly used parameters.

- formula: describes the model

  Note that the formula argument follows a specific format. For simple linear regression, this is "YVAR ~ XVAR" where YVAR is the dependent, or predicted, variable and XVAR is the independent, or predictor, variable.

- data: the variable that contains the dataset

Figure 14 is a screenshot of the following regression command.

```
model <- lm(IncomeLab ~ D_HH+famsize+agehead+sexhead, data =
MyFirstData)
summary(model)
```

```
> model <- lm(IncomeLab ~ D_HH+famsize+agehead+sexhead, data = MyFirstData)
> summary(model)

Call:
lm(formula = IncomeLab ~ D_HH + famsize + agehead + sexhead,
    data = MyFirstData)

Residuals:
    Min      1Q  Median      3Q     Max
-2447.3 -1020.8  -390.3   443.2 24017.9

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3018.2879   774.3589   3.898  0.00014 ***
D_HH        -1190.0918   414.8902  -2.868  0.00465 **
famsize       130.8700    80.3182   1.629  0.10509
agehead       -10.3495    12.1909  -0.849  0.39711
sexheadMale    -0.7972   432.0480  -0.002  0.99853
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2351 on 169 degrees of freedom
  (1026 observations deleted due to missingness)
Multiple R-squared:  0.06183,   Adjusted R-squared:  0.03963
F-statistic: 2.785 on 4 and 169 DF,  p-value: 0.0283
```

Figure 14. Output of lm command

The output of regression command uses standard notations found in any econometric or statistics text book. Although R^2 is important in regression models used for predictions, the most relevant output for our statistical inference is:

- ✓ As a rule of thumb, "p-value" should be 0.05 or lower. This result implies that an F-test that tests joint significance of all model coefficients is statistically significantly different from a zero vector at the five percent level.

- ✓ The three columns “Std Error”, “t value” and “Pr>|t|” denote the standard error of the coefficient, the t-value for testing the significance against the null hypothesis that the coefficient is 0, and p-value is the statistical significance level for a two sided/tailed t-test.

The lm function also provides additional results that are not displayed in the regression results, like plots and predictions. Try `plot(model)` and explore the help page of the `predict` function to learn more.

6. HYPOTHESIS TESTING

Hypothesis testing is a fundamental part of any statistical analysis. We specify a null hypothesis, which specifies a state of the world that we would like to see if our data provides evidence against, and an alternative hypothesis. For example, the null hypothesis in context of an impact evaluation of a social impact project could be:

H0: The intervention has no effect on household income levels
H1: The intervention changes the household income levels

The null hypothesis is either rejected or not rejected based on statistical significance tests. Please note, we never “accept” the alternative hypothesis in statistical parlance.

Student t-test is one of the oldest most powerful statistical tests to assess whether the difference in two means is statistically significant (different from 0). In fact, if you are able to design a randomized control trial and achieve both perfect covariate baseline balance and perfect compliance with your study design, then a t-test (or its variants) is all that you may need to estimate the impacts of the trial. In R, t-tests are done using the t.test function

```
# independent 2-group t-test
t.test(y~x) # where y is numeric and x is a binary factor

# independent 2-group t-test
t.test(y1,y2) # where y1 and y2 are numeric

# paired t-test
t.test(y1,y2,paired=TRUE) # where y1 & y2 are numeric

# one sample t-test
t.test(y,mu=3) # Ho: mu=3
```

You can use the **var.equal = TRUE** option to specify equal variances and a pooled variance estimate. You can use the **alternative="less"** or **alternative="greater"** option to specify a one tailed test.

Figure 15 is the output of the following t-test command:

```
> t.test(MyFirstData$IncomeLab ~ MyFirstData$D_HH, var.equal = TRUE)

        Two Sample t-test

data:  MyFirstData$IncomeLab by MyFirstData$D_HH
t = 2.7327, df = 172, p-value = 0.006938
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  305.1836 1892.7926
sample estimates:
mean in group 0 mean in group 1
       3044.894        1945.906
```

Figure 15. Output of t.test function

7. BIBLIOGRAPHY/FURTHER READING

1. R for beginners – Emmanuel Paradis https://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
2. R Learning Modules - https://stats.idre.ucla.edu/r/modules/
3. Harvard ggplot2 tutorial - http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html
4. Ggplot2 Cheat Sheet - https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf
5. R Studio Cheat Sheets - https://www.rstudio.com/resources/cheatsheets/