Chapter 1 - Introduction and Basics

M Affouf

January 2, 2018

Agenda

- Course overview
- Introduction to R, RStudio and R Markdown
- Programming basics

How this class will work

- No programming knowledge presumed
- Some prior stat assigned reading presumed.
- Basic stat definitions
- Hypothesis testing (t-tests, confidence intervals)
- Linear regression
- Class attendance is mandatory
- Class will be *very* cumulative

Mechanics

- One session a week:
- First half: loading materials, concepts, methods, examples
- Second half: Labs
- Class participation
- Weekly Assignment
- Final project (Last 3 weeks)

Mechanics

- **Attendance and Class participation** (20%)
- **Labs and assignments**: Each lecture has an **accompanying lab assignment** (40%).
- Late Assignments/homework **will not be accepted for credit**
- **Final project** (40%)
- You will write a report analysing a policy question using a publicly available data set

Course resources

- Google Drive will be used to post all course notes, labs, assignments, data and announcement
- The instructor GoogleDrive: `maths3141@gmail.com`, I will share a folder with your email
- Every student should create a folder in their Google Drive named: FDA_YourLastName and share it with `maths3141@gmail.com`
- All graded Labs/Assignemts should be uploaded to your FDA folder

highly recommended: - Garrett Grolemund and Hadley Wickham, *R for Data Science*

- Phil Spector, *Data Manipulation with R*
- Paul Teetor, *The R Cookbook*
- Winston Chang, *The R Graphics Cookbook*
- Norman Matloff, *The Art of R Programming: A Tour of Statistical Software Design*

Goal of this class

This class will teach you to use R to:

- Generate graphical and tabular data summaries

- Perform statistical analyses (e.g., hypothesis testing, regression modeling)

- Handle different data sources: Surveys, Financial, and text data

- Produce reproducible statistical reports using R Markdown

Why R?

- Free (open-source)
- Programming language (not point-and-click)
- Excellent graphics
- Offers broadest range of statistical tools
- Easy to generate reproducible reports
- Easy to integrate with other tools

The R Console

Basic interaction with R is through typing in the **console**

The R Console

- You type in commands, R gives back answers (or errors)
- Menus and other graphical interfaces are extras built on top of the console
- We will use **RStudio** in this class

1. Download R:
2. Then download RStudio: `http://www.rstudio.com/`

RStudio is an IDE for R

RStudio has 4 main windows ('panes'): - Source - Console - Workspace/History - Files/Plots/Packages/Help

- Use the **Console** pane to type or paste commands to get output from R
- To look up the help file for a function or data set, type `?function` into the Console
- E.g., try typing in `?mean`
- Use the `tab` key to auto-complete function and object names
- Use the **Source** pane to create and edit R and Rmd files
- The menu bar of this pane contains handy shortcuts for sending code to the **Console** for evaluation
- By default, any figures you produce in R will be displayed in the **Plots** tab
- Menu bar allows you to Zoom, Export, and Navigate back to older plots
- When you request a help file (e.g., `?mean`), the documentation will appear in the **Help** tab

RStudio: Panes overview

1. **Source** pane: create a file that you can save and run later
2. **Console** pane: type or paste in commands to get output from R
3. **Workspace/History** pane: see a list of variables or previous commands
4. **Files/Plots/Packages/Help** pane: see plots, help pages, and other items in this window.

Getting Started

- You should have the latest version of R installed!
- Open R Studio
- Files –> New –> R Script
- Save the blank R script as “day1.R” in a directory of your choosing
- Add a comment header

Commenting in Scripts

Add a comment header to day1.R :# is the comment symbol

```
####################
# Title: Demo R Script
# Author: M Affouf
# Date: 12/24/2017
# Purpose: Demonstrate comments in R
######################

# nothing to its right is evaluated

# this # is still a comment
### you can use many #'s as you want

# sometimes you have a really long comment,
#    like explaining what you are doing
#    for a step in analysis.
# Take it to another line
```

Explaining output on slides

In slides, a command (we'll also call them code or a code chunk) will look like this

```
print("I'm code")
```

```
## [1] "I'm code"
```

And then directly after it, will be the output of the code.
So `print("I'm code")` is the code chunk and [1] "I'm code" is the output.

R as a calculator

```
2 + 2
```

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

```
2 * 4
```

```
## [1] 8
```

```
2 ^ 3
```

```
## [1] 8
```

Note, when you type your command, R inherently thinks you want to print the result.

R as a calculator

- The R console is a full calculator
- Try to play around with it:
  - $+$, -, $/$, * are add, subtract, divide and multiply
  - ^ or ** is power
  - parentheses – (and) – work with order of operations

R as a calculator

```
2 + (2 * 3)^2
```

```
## [1] 38
```

```
(1 + 3) / 2 + 45
```

```
## [1] 47
```

R as a calculator

Try evaluating the following:

- `2 + 2 * 3 / 4 -3`
- `2 * 3 / 4 * 2`
- `2^4 - 1`

- You can create variables from within the R environment and from files on your computer
- R uses "=" or "<-" to assign values to a variable name
- Variable names are case-sensitive, i.e. X and x are different

```
x = 2 # Same as: x <- 2
x
```

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

```
x * 4
```

```
## [1] 8
```

```
x + 2
```

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

R variables

- The most comfortable and familiar class/data type for many of you will be `data.frame`
- You can think of these as essentially Excel spreadsheets with rows (usually subjects or observations) and columns (usually variables)
- Go to RStudio –> Tools –> Import Dataset –> From Web URL then paste

`http://data/Charm_City_Circulator_Ridership.csv`

R variables

- We can display the top of the data with `head`:

```
head(Charm_City_Circulator_Ridership)
```

```
       day       date orangeBoardings orangeAlightings ora
1   Monday 01/11/2010              877              1027
2  Tuesday 01/12/2010              777               815
3 Wednesday 01/13/2010             1203              1220
4 Thursday 01/14/2010             1194              1233
5   Friday 01/15/2010             1645              1643
6 Saturday 01/16/2010             1457              1524
  purpleBoardings purpleAlightings purpleAverage greenBoard
1              NA                NA            NA
2              NA                NA            NA
3              NA                NA            NA
4              NA                NA            NA
5              NA                NA            NA
6              NA                NA            NA
```

R variables

- `data.frames` are somewhat advanced objects in R; we will start with simpler objects;
- Here we introduce "1 dimensional" classes; these are often referred to as 'vectors'
- Vectors can have multiple sets of observations, but each observation has to be the same class.

```
class(x)
```

```
## [1] "numeric"
```

```
y = "hello world!"
print(y)
```

```
## [1] "hello world!"
```

```
class(y)
```

R variables

Try assigning your full name to an R variable called `name`

R variables

Try assigning your full name to an R variable called `name`

```
name = "Rober De Niro"
name
```

```
## [1] "Rober De Niro"
```

The 'combine' function

The function `c()` collects/combines/joins single R objects into a vector of R objects. It is mostly used for creating vectors of numbers, character strings, and other data types.

```
x <- c(1, 4, 6, 8)
x
```

```
## [1] 1 4 6 8
```

```
class(x)
```

```
## [1] "numeric"
```

The ‘combine’ function

Try assigning your first and last name as 2 separate character strings into a single vector called `name2`

The 'combine' function

Try assigning your first and last name as 2 separate character strings into a length-2 vector called `name2`

```
name2 = c("Robert","De Niro")
name2
```

```
## [1] "Robert"  "De Niro"
```

R variables

`length()`: Get or set the length of vectors (including lists) and factors, and of any other R object for which a method has been defined.

```
length(x)
```

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

```
y
```

```
## [1] "hello world!"
```

```
length(y)
```

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

R variables

What do you expect for the length of the `name` variable? What about the `name2` variable?

What are the lengths of each?

R variables

What do you expect for the length of the `name` variable? What about the `name2` variable?

What are the lengths of each?

```
length(name)
```

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

```
length(name2)
```

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

R variables

You can perform functions to entire vectors of numbers very easily.

```
x + 2
```

```
## [1]  3  6  8 10
```

```
x * 3
```

```
## [1]  3 12 18 24
```

```
x + c(1, 2, 3, 4)
```

```
## [1]  2  6  9 12
```

R variables

But things like algebra can only be performed on numbers.

```
> name2 + 4
[1] Error in name2 * 4 : non-numeric argument
 to binary operator
```

R variables

And save these modified vectors as a new vector.

```
y = x + c(1, 2, 3, 4)
y
```

```
## [1]  2  6  9 12
```

Note that the R object `y` is no longer "Hello World!" - It has effectively been overwritten by assigning new data to the variable

R variables

- You can get more attributes than just class. The function `str` gives you the structure of the object.

```
str(x)
```

```
##  num [1:4] 1 4 6 8
```

```
str(y)
```

```
##  num [1:4] 2 6 9 12
```

This tells you that `x` is a numeric vector and tells you the length.

Back to our data.frame example

- Let's see what the structure of our data.frame is:

```
str(Charm_City_Circulator_Ridership)
```

```
## 'data.frame':    1146 obs. of  15 variables:
##  $ day               : Factor w/ 7 levels "Friday","Monday
##  $ date              : Factor w/ 1146 levels "01/01/2011",
##  $ orangeBoardings   : int  877 777 1203 1194 1645 1457 83
##  $ orangeAlightings: int  1027 815 1220 1233 1643 1524 9
##  $ orangeAverage     : num  952 796 1212 1214 1644 ...
##  $ purpleBoardings   : int  NA NA NA NA NA NA NA NA NA NA
##  $ purpleAlightings: int  NA NA NA NA NA NA NA NA NA NA
##  $ purpleAverage     : num  NA NA NA NA NA NA NA NA NA NA
##  $ greenBoardings    : int  NA NA NA NA NA NA NA NA NA NA
##  $ greenAlightings   : int  NA NA NA NA NA NA NA NA NA NA
##  $ greenAverage      : num  NA NA NA NA NA NA NA NA NA NA
##  $ bannerBoardings   : int  NA NA NA NA NA NA NA NA NA NA
##  $ bannerAlightings: int  NA NA NA NA NA NA NA NA NA NA
```

Review

- Creating a new script
- Using R as a calculator
- Assigning values to variables
- Performing algebra on numeric variables

R Markdown

- R Markdown allows the user to integrate R code into a report
- When data changes or code changes, so does the report
- No more need to copy-and-paste graphics, tables, or numbers
- Creates **reproducible** reports
- Anyone who has your R Markdown (.Rmd) file and input data can re-run your analysis and get the exact same results (tables, figures, summaries)
- Can output report in HTML (default), Microsoft Word, or PDF
- This example shows an **R Markdown** (.Rmd) file opened in the Source pane of RStudio.
- To turn an Rmd file into a report, click the **Knit HTML** button in the Source pane menu bar
- The results will appear in a **Preview window**, as shown on the right
- You can knit into html (default), MS Word, and pdf format
- These lecture slides are also created in RStudio (R Presentation)
- To integrate R output into your report, you need to use R code

In-class exercise: Hello world!

1. Open **RStudio** on your machine
2. File > New File > R Markdown ...
3. Change `summary(cars)` in the first code block to `print("Hello world!")`
4. Click `Knit HTML` to produce an HTML file.
5. Save your Rmd file as `helloworld.Rmd`

Basics: the class in a nutshell

- Everything we'll do comes down to applying **functions** to **data**
- **Data**: things like 7, "seven", 7.000, the matrix $\begin{bmatrix} 7 & 7 & 7 \\ 7 & 7 & 7 \end{bmatrix}$
- **Functions**: things like log , $+$ (two arguments), $<$ (two), mod (two), `mean` (one)

  *A function is a machine which turns input objects (**arguments**) into an output object (**return value**), possibly with **side effects**, according to a definite rule*

Data building blocks

You'll encounter different kinds of data types

- **Booleans** Direct binary values: `TRUE` or `FALSE` in R
- **Integers**: whole numbers (positive, negative or zero)
- **Characters** fixed-length blocks of bits, with special coding; **strings** = sequences of characters
- **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like 1.87×10^6
- **Missing or ill-defined values**: `NA`, `NaN`, etc.

Operators (functions)

You can use R as a very, very fancy calculator

Command	Description
`+,-,*,\`	add, subtract, multiply, divide
`^`	raise to the power of
`%%`	remainder after division (ex: `8 %% 3 = 2`)
`( )`	change the order of operations
`log(), exp()`	logarithms and exponents (ex: `log(10) = 2.30`
`sqrt()`	square root
`round()`	round to the nearest whole number (ex: `round(`
`floor(), ceiling()`	round down or round up
`abs()`	absolute value

```
7 + 5 # Addition
```

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

Operators cont'd.

Comparisons are also binary operators; they take two objects, like numbers, and give a Boolean

```
7 > 5
```

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

```
7 < 5
```

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

```
7 >= 7
```

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

```
7 <= 5
```

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

Boolean operators

Basically "and" and "or":

```
(5 > 7) & (6*7 == 42)
```

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

```
(5 > 7) | (6*7 == 42)
```

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

(will see special doubled forms, && and ||, later)

More types

- `typeof()` function returns the type
- `is.`*foo*`()` functions return Booleans for whether the argument is of type *foo*
- `as.`*foo*`()` (tries to) "cast" its argument to type *foo* — to translate it sensibly into a *foo*-type value

Special case: `as.factor()` will be important later for telling R when numbers are actually encodings and not numeric values. (E.g., 1 = High school grad; 2 = College grad; 3 = Postgrad) ###

```
r   typeof(7)
## [1] "double"
r   is.numeric(7)
## [1] TRUE
r   is.na(7)
## [1] FALSE ###
r   is.character(7)
```

Variables

We can give names to data objects; these give us **variables**

A few variables are built in:

```
pi
```

```
## [1] 3.141593
```

Variables can be arguments to functions or operators, just like constants:

```
pi*10
```

```
## [1] 31.41593
```

```
cos(pi)
```

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

Assignment operator

Most variables are created with the **assignment operator**, <- or =

```
time.factor <- 12
time.factor
```

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

```
time.in.years = 2.5
time.in.years * time.factor
```

```
## [1] 30
```

The assignment operator also changes values:

```
time.in.months <- time.in.years * time.factor
time.in.months
```

```
## [1] 30
```

The workspace

What names have you defined values for?

```
ls()
```

```
## [1] "Charm_City_Circulator_Ridership" "name"
## [3] "name2"                           "time.factor"
## [5] "time.in.months"                  "time.in.years"
## [7] "x"                               "y"
```

Getting rid of variables:

```
rm("time.in.months")
ls()
```

```
## [1] "Charm_City_Circulator_Ridership" "name"
## [3] "name2"                           "time.factor"
## [5] "time.in.years"                   "x"
## [7] "y"
```

First data structure: vectors

- Group related data values into one object, a **data structure**
- A **vector** is a sequence of values, all of the same type
- `c()` function returns a vector containing all its arguments in order

```
students <- c("Sean", "Louisa", "Frank", "Farhad", "Li")
midterm <- c(80, 90, 93, 82, 95)
```

- Typing the variable name at the prompt causes it to display

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

Indexing

- `vec[1]` is the first element, `vec[4]` is the 4th element of `vec`

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
students[4]
```

```
## [1] "Farhad"
```

- `vec[-4]` is a vector containing all but the fourth element

```
students[-4]
```

```
## [1] "Sean"   "Louisa" "Frank"  "Li"
```

Vector arithmetic

Operators apply to vectors "pairwise" or "elementwise":

```
final <- c(78, 84, 95, 82, 91) # Final exam scores
midterm # Midterm exam scores
```

```
## [1] 80 90 93 82 95
```

```
midterm + final # Sum of midterm and final scores
```

```
## [1] 158 174 188 164 186
```

```
(midterm + final)/2 # Average exam score
```

```
## [1] 79 87 94 82 93
```

```
course.grades <- 0.4*midterm + 0.6*final # Final course gr
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

Pairwise comparisons

Is the final score higher than the midterm score?

```
midterm
```

```
## [1] 80 90 93 82 95
```

```
final
```

```
## [1] 78 84 95 82 91
```

```
final > midterm
```

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

Boolean operators can be applied elementwise:

```
(final < midterm) & (midterm > 80)
```

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

Functions on vectors

Command	Description
`sum(vec)`	sums up all the elements of `vec`
`mean(vec)`	mean of `vec`
`median(vec)`	median of `vec`
`min(vec)`, `max(vec)`	the largest or smallest element of `vec`
`sd(vec)`, `var(vec)`	the standard deviation and variance of

Functions on vectors

```
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

```
mean(course.grades) # mean grade
```

```
## [1] 86.8
```

```
median(course.grades)
```

```
## [1] 86.4
```

```
sd(course.grades) # grade standard deviation
```

```
## [1] 6.625708
```

More functions on vectors

```
sort(course.grades)
```

```
## [1] 78.8 82.0 86.4 92.6 94.2
```

```
max(course.grades) # highest course grade
```

```
## [1] 94.2
```

```
min(course.grades) # lowest course grade
```

```
## [1] 78.8
```

Referencing elements of vectors

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

Vector of indices:

```
students[c(2,4)]
```

```
## [1] "Louisa" "Farhad"
```

Vector of negative indices

```
students[c(-1,-3)]
```

```
## [1] "Louisa" "Farhad" "Li"
```

More referencing

`which()` returns the `TRUE` indexes of a Boolean vector:

```
course.grades
```

```
## [1] 78.8 86.4 94.2 82.0 92.6
```

```
a.threshold <- 90 # A grade = 90% or higher
course.grades >= a.threshold # vector of booleans
```

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

```
a.students <- which(course.grades >= a.threshold) # Applyi
a.students
```

```
## [1] 3 5
```

```
students[a.students] # Names of A students
```

```
## [1] "Frank" "Li"
```

Named components

You can give names to elements or components of vectors

```
students
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
names(course.grades) <- students # Assign names to the gra
names(course.grades)
```

```
## [1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
course.grades[c("Sean", "Frank","Li")] # Get final grades
```

```
##  Sean Frank    Li
##  78.8  94.2  92.6
```

Note the labels in what R prints; these are not actually part of the value

Useful RStudio tips

Keystroke	Description
<tab>	autocompletes commands and filenames, and lists arguments for functions. Highly useful!
<up>	cycle through previous commands in the console prompt