Recommonded materials for R beginners

Kevin & Yalong

25/01/2019

Contents

Introduction to RStudio **5**
Step 1: Run RStudio . . . 5
Step 2: Install packages in RStudio . . . 6
Step 3: Copy and run code . . . 6
Step 4: Do statistics . . . 6
Step 5: Build a group of related vectors . . . 6
Step 6: Combine different vectors . . . 7
Step 7: Perform dataframe operation . . . 7
Step 8: Determine statistics . . . 8
Step 9: Display data . . . 8
Step 10: Determine equation for the line . . . 8
Step 11: Predict age . . . 9
Step 12: Lay down multiple plots . . . 9

Introduction to R programming **11**
Installation and basics . . . 11
Reference material . . . 11

Activity: Exploring & Visualising Data using R **13**
Data on Maps with R . . . 13

Introduction to RStudio

If you don't have RStudio (or R), please follow the instructions on: Introduction to R programming.

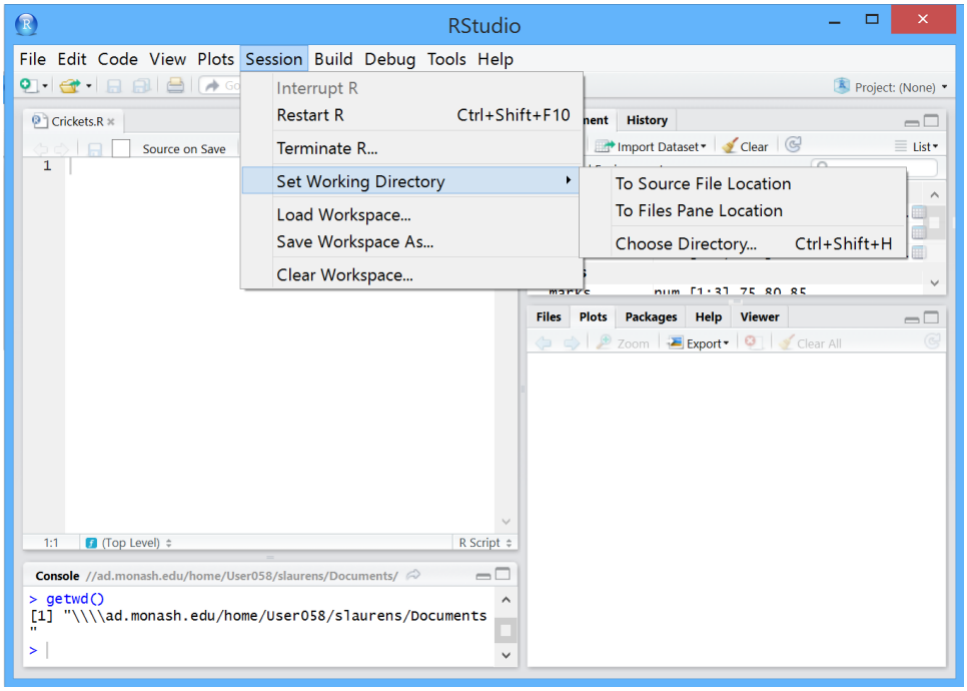
Step 1: Run RStudio

The four main windows in RStudio are shown below:

- top left; this is your **R script** file (currently 'Crickets.R', which can be saved and run, either in pieces or completely);
- bottom left; that is the **Console**, where R commands are run (currently showing getwd()), and output is shown;
- top right; that is the **Workspace and history**, you can see which data and values R has in its memory.
- bottom right; of that is where you can see Plots, install Packages, view your Files.

Working directory should be where your scripts (code files) and data files are.

- getwd() can print out your current working directory; **bottom right Files -> Go To Working Directory**
- use setwd() to change the directory you want. You can also do this by click at the **bottom right Files -> Navigate to your folder -> More -> Set As Working Directory**



Note that there are several options for coding in RStudio. You can:

1 Enter the code directly into an R script file and then run it from there (using select and C**TRL-R**), or 2. Copy and paste from the file to the Console, or 3. Write directly into the Console (then **ENTER** to run for the latter two options).

Try the last option, writing and testing your code in the Console, then when it's working and you want to save it, copy it into the script file above the Console (which can be saved).

Step 2: Install packages in RStudio

There are two different ways to install packages in RStudio:

- Install via R Console;
  - The **Console** is located at the **bottom left** section.
  - Type in the command (replace "*packageName*" with the package you want to install, like "*ggmap*").

```
install.packages("packageName")
```

- Press "return" key
- Install via RStudio GUI
  - Locate the "**Packages**" tab at the **bottom right** section and click.
  - Click the "Install" button and an install dialogue will pop out.
  - Write the package name in the text field and click install.

Step 3: Copy and run code

When dealing with data, data structures are important. R & Python have similar storage options and similar flexibility in usage.

Copy the code below to the Console and run:

```
scores <- c(75, 80, 85) # put the data in
scores # and get the data out
```

As an example of flexibility, we can run an operation on a structure without an explicit loop:

```
scores * 2
scores * scores
```

Step 4: Do statistics

So, let's do some statistics. The average of 75, 80, 85 is 80? Confirm with R:

```
mean(scores)
median(scores)
sd(scores)
```

Step 5: Build a group of related vectors

Doing more with data, instead of working with a single vector (scores), we can build a group of related vectors (Cricket):

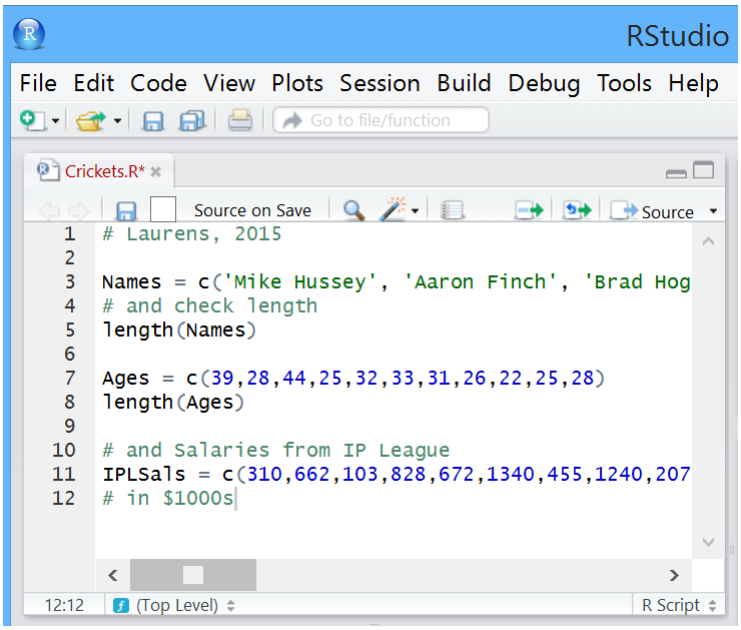
```
Names <- c("Mike Hussey", "Aaron Finch", "Brad Hogg", "Steve Smith", "George Bailey", "Mitchell Johnson
# and check length
length(Names)

Ages <- c(39,28,44,25,32,33,31,26,22,25,28)
length(Ages)
```

```
'Error: object 'ages' not found'
```

Oops, R is case sensitive!

```
length(Ages)
# and Salaries in 1000s from IP League
IPLSals <- c(310,662,103,828,672,1340,455,1240,207,1030,1140)
```

So, if the above worked, and you want to save it, copy it to your script file, add a comment (e.g. name, date), and save:



Step 6: Combine different vectors

For more powerful operations, we can combine different vectors (of the same length), with different data types, into a dataframe:

```
df <- data.frame(Names, Ages, IPLSals)
df # and display
```

Who wants to try this?

```
df * scores
```

Step 7: Perform dataframe operation

Dataframe operations, display rows and columns:

```
df[1, ]      #  row 1, all columns
df[, 1]      # column 1, all rows (or all rows, column 1)
df[2:3]      #
df[2:3, ]    # subtle difference
```

Step 8: Determine statistics

Statistics using dataframes:

```
# so what's the average age? Guess first, around 30?
mean(df[,2])
# or could do this:
mean(df[,"Ages"])
# or
mean(df$Ages)
# or this? mean(Ages)
```

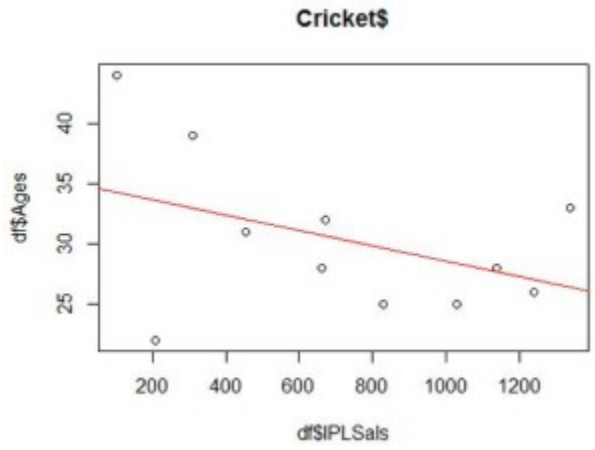
So what is the average IPLSal? The standard deviation? The variance?

Try var() and sd()

Step 9: Display data

Let's display data, adding a linear regression:

```
plot(df$Ages ~ df$IPLSals, main = "Cricket$")  #
abline(lm(df$Ages ~ df$IPLSals), col = "red")  # equation, fits to points
```



Step 10: Determine equation for the line

Nice, but what is the equation for the line? We can do it in stages:

```
fit <- lm(df$Ages ~ df$IPLSals)
fit
```

So the regression equation is: age = -0.006372 x salary + 34.899143

I.e. you can predict a player's age from his IPL cricket salary.

What is the predicted age for a $1M salary?

What is it for $0?

Step 11: Predict age

Let's try it the other way around, swapping the axes (i.e., swapping dependent and independent variables):

```
fit <- lm(df$IPLSals ~ df$Ages)
fit
```

What is the equation?

What is the estimated salary for a 20 year old?

```
# And plot it the other way round
plot(df$IPLSals ~ df$Ages, main = "Cricket$")
abline(fit, col="red")
```

Step 12: Lay down multiple plots

More plotting and scheming, you can lay down multiple plots on top of each other:

```
sorted <- sort(Ages) # just so we can see some other data
plot(Ages, IPLSals, xlab = "Ages", col = "red" )
par(new=TRUE) # adds new plot to old; redraws axes….
plot(sorted, IPLSals, xlab = "Ages", col = "green" )
```

Or add points to a plot:

```
plot(Ages, IPLSals, col = "blue" )
points(sorted, IPLSals, col = "red" )
```

Or you can just let R sort it out:

```
plot(df)
```

Introduction to R programming

The expectation is that you already know at least the basics of programming, but that you haven't programmed in R previously. So here you are introduced you to R; installing it; its major features and basic programming concepts; its specific capabilities for doing probability and statistics; and a brief guide to reference material. All of these are well-documented on the Internet, so mostly this submodule will just provide a path through some of this literature. If you read through the links, install and do the exercises, and complete this module's programming assignment, you will be well on your way with R programming. In subsequent modules we will freely illustrate data analysis concepts with R code.

R is free software that originally was modelled on the prior (rather expensive) statistical programming language and platform called S (see the R and S descriptions at wikipedia). Like much free software, students, researchers and academics around the world have made contributions to R, with the result that it has become quite a powerful statistical modeling programming language and is well worth learning.

Installation and basics

The first step is to read Torfs & Brauer's "A (Very) Short Introduction to R" [pdf]. It explains how to download and install R and RStudio from CRAN (the Comprehensive R Archive Network). (It explains this for Windows, but for Macs it's also completely straightforward; CRAN [website], by the way, stands for The Comprehensive R Archive Network and holds all the software and documentation you really need for R.) RStudio is a graphical user interface that make programming, testing and running R programs easier, so I recommend installing it. After installation, you should read through the rest of this introduction, doing the examples in R or RStudio (the "ToDos"). That will give you a decent start on programming, including R's control structures, data structures, graphing, file operations, etc.

The first step in using RStudio should always be to set your working directory to the directory where your source or data are, using the **setwd()** command in your console. If you make that your habit, you will avoid a lot of confusion. You can always use **getwd()** to confirm your location.

Reference material

When you have specific questions about what function to use for what, etc., there are a large number of good references for R. Here are some:

- The wikibooks' "R Programming" [website]. This includes quite a lot of easily accessed reference material. It also has introductory material and guides to statistical methods, such as clustering and maximum likelihood estimation.
- The CRAN manuals [website; pdf]. These include "An Introduction to R" [pdf, 100pp], installation details, a thorough language specification and the "R Reference Index", which is a 9 MB pdf comprehensive manual.

- Johnson's "RTips. Revival 2014!" [website; pdf] updates a document begun in 1999. This is (in pdf form) 72 pages of good guidance on using R for statistical tasks.

Activity: Exploring & Visualising Data using R

Data on Maps with R

You will need R and RStudio, you will also need to install the following libraries (and dependencies):

- ggmap
- ggplot2
- maps

If you are not sure how to do this and want to explore some basic knowledge about R.

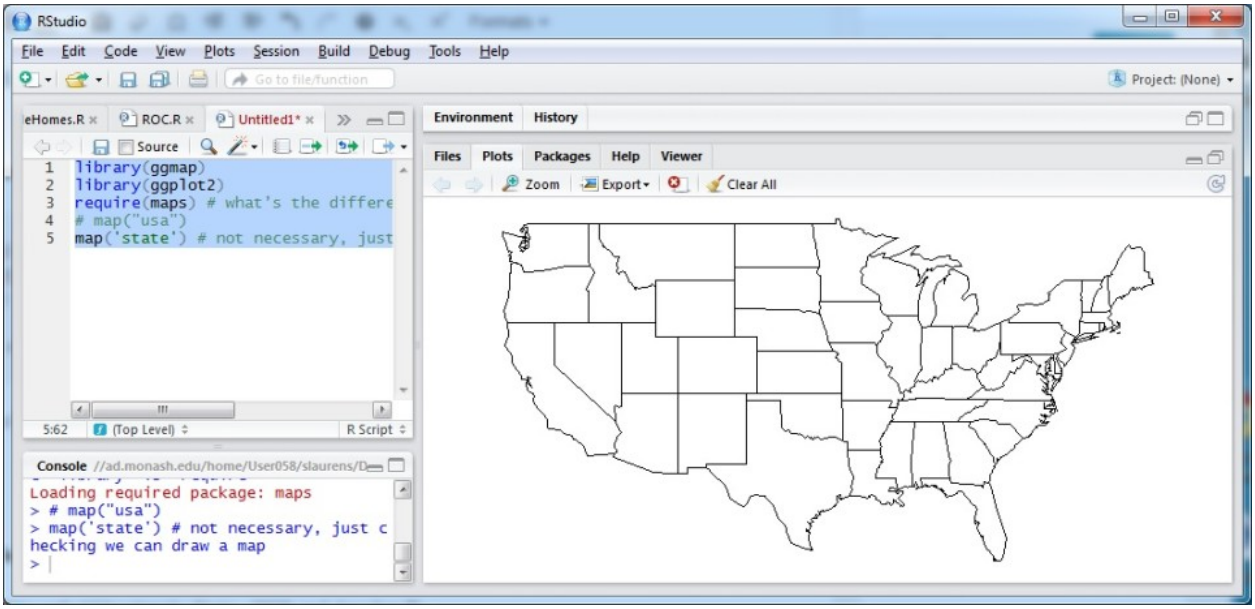
DO TAKE A LOOK AT:

- Introduction to RStudio
- Introduction to R programming
- [Torfs & Brauer's "A (Very) Short Introduction to R" [pdf]]http://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf

Step 1 Run RStudio, install libraries, try the following code (copy, paste, run):

```
library(ggmap)
library(ggplot2)
library(maps)
# or you can try e.g. require(maps)
# map("usa")
map('state') # not necessary, just checking we can draw a map
```

You should see a basic map of the US states:



If an error appear as "Plot region too large".

You just need to adjust the **right bottom** section of RStudio larger.

Step 2 Read the modified data file (state names & codes have been added).

Download the file at Household-heating-by-State-2008.csv, and put it in your working directory.

```
data <- read.csv("Household-heating-by-State-2008.csv", header=T)
head(data)
names(data)
```

Step 3 Names are a bit much, simplify the one we're interested in:

```
names(data)[4] <- "MobileHomes"
names(data)
```

Step 4 Now group the Mobile Home data by State, calculating the average:

```
ag <- aggregate(MobileHomes ~ States, FUN = mean, data = data)
```

Look at the paprameters of `aggregate`.

- `MobileHomes ~ States` means group the Mobile Home data by State
- `FUN = mean` means calculate their averages,

How many states should there be?

```
head(ag)
dim(ag)
```

Not going to worry about the first '#N/A' state for now, delete it if you like

Step 5 Get map data (built-in map data in ggplot2 package)

```
m.usa <- map_data("state") # we want the states
head(m.usa)
dim(m.usa) # more info than we need
```

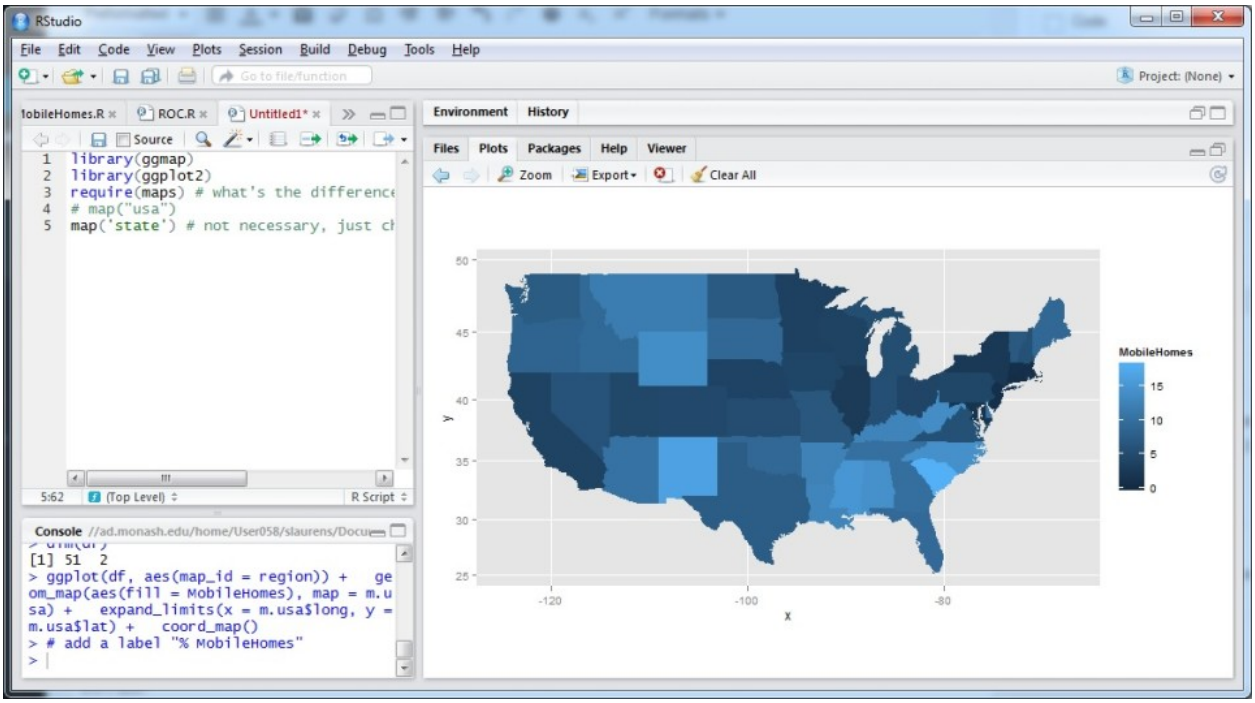
Step 6 Force our states to lowercase to match the map, keep also one column of data

```
df <- data.frame(region = tolower(ag$States),
                 MobileHomes = ag$MobileHomes, stringsAsFactors = F
                 )
dim(df)
```

50 states?

Step 7 Now plot it on the map (look at all those layers...)

```
# Create a empty canvas
ggplot(df, aes(map_id = region)) +
   # draw the grid
  expand_limits(x = m.usa$long, y = m.usa$lat) +
  # draw a us map, fill = MobileHomes means color the map according to MobileHomes property
  geom_map(aes(fill = MobileHomes), map = m.usa) +
  # fix the ratio of the x and y axises, to match a map
  coord_map()
```



QUESTION: Compare R with Tableau Public for data visualisation – **which do you prefer and why?** Consider also the data wrangling to prepare and process the data.