MONASH University

MONASH
BUSINESS
SCHOOL

Getting Started with R

High Dimensional Data Analysis

Anastasios Panagiotelis
Lecture 2

Basics

The R project

- In this unit we use the software package R.
- R is one of the the most popular software tool for professionals in the fields of Business Analytics/Data Science.

History of R

- R is based on earlier statistical software called 'S' which was developed at Bell Labs in the 1970s.
- R was initially developed in the early 1990s by two academics at the University of Auckland, Ross Ihaka and Robert Gentlemen.
- R has grown substantially since then and is now supported the not-for-profit R Foundation

R is Free Software

- R is free in two ways
  - It doesn't cost any money.
  - All of the source code of R is available meaning R can be customised, modified, and most importantly extended.
- R is part of a big Free Software project known as the GNU Project

Downloading R and R Studio

- R can be downloaded here
- Exact details of installing R will depend on whether you use Windows, Mac or Linux.
- A great tool for both new and experienced users of R is *RStudio*.
- It can be downloaded here free of cost.
- If you haven't done so already try to download R and R Studio.

Ways to use R

- To keep track of your workflow use a **script**:
  - Open a new script by typing Ctrl+Shift+N
  - Run a single line of code by pressing Ctrl+Enter
  - Run a whole script by pressing Ctrl+Shift+S or Ctrl+Shift+Enter
- You can save scripts to run them anytime.
- Scripts allow you to keep analysis **replicable** which is important in research and business.

Variables in R

- In R everything is stored in a **variable**.
- Here the word variable has a slightly different meaning to the usual statistical meaning.
- In R, think of these as little boxes with names on them.
- We can put a number into these boxes, or words or matrices or entire blocks of data or even other boxes.

Assigning Variables

- How to store the number 1 in a variable `a` and the number 2 in a variable `b`?

```
a<-1
b=2
```

- Note that you can use either `<-` or `=` to assign variables.

Seeing results

To see what is stored in a variable

```
print(b)
```

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

```
str(b)
```

```
##  num 2
```

Character variables

- Text can also be stored in a variable.
- Try to store your own name in a variable called *name*.

```
name<-'Anastasios'
str(name)
```

```
##  chr "Anastasios"
```

- Use apostrophes so that R does not look for a variable called `Anastasios`.

Workspace

- All variables are kept in the **workspace**. You can see what is in your workspace by using the command

```
ls()
```

```
## [1] "a"    "b"    "name"
```

Clear Workspace

- You can clear the workspace using

```
rm(list=ls())
```

- If you try ls() again the workspace will be empty. In RStudio you can also see all the variables in the *Environment* tab.
- It is good practice to start every script with this command so that you do not accidentally use data from a different project.

Working directory

- If you read data stored on your computer, or if you save plots or data then the concept of a **working directory** is important.
- To check your working directory type

```
getwd()
```

- To change the Working directory use `setwd`

```
setwd("/home/anastasios/Documents")
```

Basic arithmetic in R

- Basic arithmetic is fairly simple. Try `a+b`. Also we will put this in a new variable called `z`.

```
z<-a+b
str(z)
```

```
##  num 3
```

- To subtract use `-`, to multiply use `*`, to divide `/` and to take powers use `^`.

Functions in R

- Apart from very simple arithmetic, variables in R are manipulated using a **function**.
- The input goes in parentheses, while the output is assigned to a new variable.
- For example the function *sqrt* is used to take the square root.

```
rootb<-sqrt(b)
str(rootb)
```

```
##  num 1.41
```

Garbage in/Garbage out

What happens when you take a square root of something that is not a number?

```
rootname<-sqrt(name)
```

```
## Error in sqrt(name): non-numeric argument
```

Many if not most of the mistakes you make in R occur because you enter the incorrect type of input in a function.

Getting Help

- If you aren't sure what a function does, use R help. The easiest way is to simply use the `?`

```
?sqrt
```

- If you want to do something and do not know the name of the relevant function you can search using `??`.
- Find a function to do logarithms using

```
??logarithms
```

Comments

Anything after a `#` will not be executed by R.

```
a<-1 # Set the variable a to 1
#x<-4 This line is not executed
str(a)
```

```
##  num 1
```

```
str(x)
```

```
## Error in str(x): object 'x' not found
```

Vectors

In stats we have many observations for each variable. The function `c()` stores these in a **vector**. Suppose we have the following data:

Names	Drink	Consumption	Satisfaction
Andrew	Coke	50	5
Boris	Pepsi	40	4
Cathy	Coke	25	4
Diana	7Up	0	3

Manually inputting data

First let's create the variable Consumption

```r
Consumption<-c(50,40,25,0)
print(Consumption)
```

```
## [1] 50 40 25  0
```

```r
str(Consumption)
```

```
##  num [1:4] 50 40 25 0
```

Put values of drink into a variable.

Solution

The solution is

```
Drink<-c('Coke','Pepsi','Coke','7Up')
print(Drink)
```

```
## [1] "Coke"  "Pepsi" "Coke"  "7Up"
```

```
str(Drink)
```

```
##  chr [1:4] "Coke" "Pepsi" "Coke" "7Up"
```

Vector

These variables are example of a **vector**. Sometimes when we apply a function to a vector, we apply the function to each element.

```
logcons<-log(Consumption)
str(logcons)
```

```
##  num [1:4] 3.91 3.69 3.22 -Inf
```

Vectors

Other functions take a vector as an input and return a single number as the output

```
meancons<-mean(Consumption)
str(meancons)
```

```
##  num 28.8
```

Inf and NaN

The values `Inf` and `-Inf` refer to positive and negative infinity. The value `NaN` stands for not a number and indicates an error.

```
log(-1)
```

```
## Warning in log(-1): NaNs produced
```

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

It is important to distinguish `NaN` from `NA`. The latter is used for missing data.

Data Frame

- It is tedious to manually enter large datasets in this way. You will usually import data from an external file.
- There are many ways to import data. For files with the `.rds` extension it is easy

```
Beer<-readRDS("Beer.rds")
```

- Get the location of the file right. You can also open a file through the *file* tab

Data Frame

- There is only one variable here, the variable *Beer*. However this is a very special case of variable known as a **Data Frame**.
- A data frame contains other variables. For example *alcohol* content can be accessed via.

```
str(Beer$alcohol)
```

```
##  num [1:35] 3.7 4.1 4.2 4.3 2.9 2.3 4.2 4..
```

Lists

Another object common in R is known as a **list**.
A list can contain completely different variables.

```r
alist<-list(w=name, x=Drink, y=Beer)
```

elements of lists are accessed using `[[]]` or `$`

```r
alist[[1]]
```

```
## [1] "Anastasios"
```

Packages

R Packages

- One of the biggest advantages of R is the use of add-on packages, which are are easily downloaded from an online repository called **CRAN**. Using a package involves two steps:
  - Downloading and installing the package using the function `install.package`.
  - Load the package using `library` function
- Both these steps can also be done in RStudio through the *Packages* tab.
- Try install and load the R package `ggplot2`

Package Documentation

- By downloading the package you also download all of the help documentation.

```r
install.packages('ggplot2')
```

To load the package

```r
library(ggplot2)
```

Graphics in R

- Three different ways to do graphs in R
  - Base graphics do not require any packages
  - Trellis graphics (using package lattice)
  - ggplot2
- In this unit you will mostly be given instruction on using `ggplot2`, however if you have learnt base graphics in another unit and prefer this, then you can use it.
- There are many resources for learning `ggplot2`, including some that are free online.

MT cars dataset

To demonstrate `ggplot2` we use the dataset `mpg`, which contains information on the fuel efficiency of different cars. This can be loaded into R using the command

```
data(mpg)
```

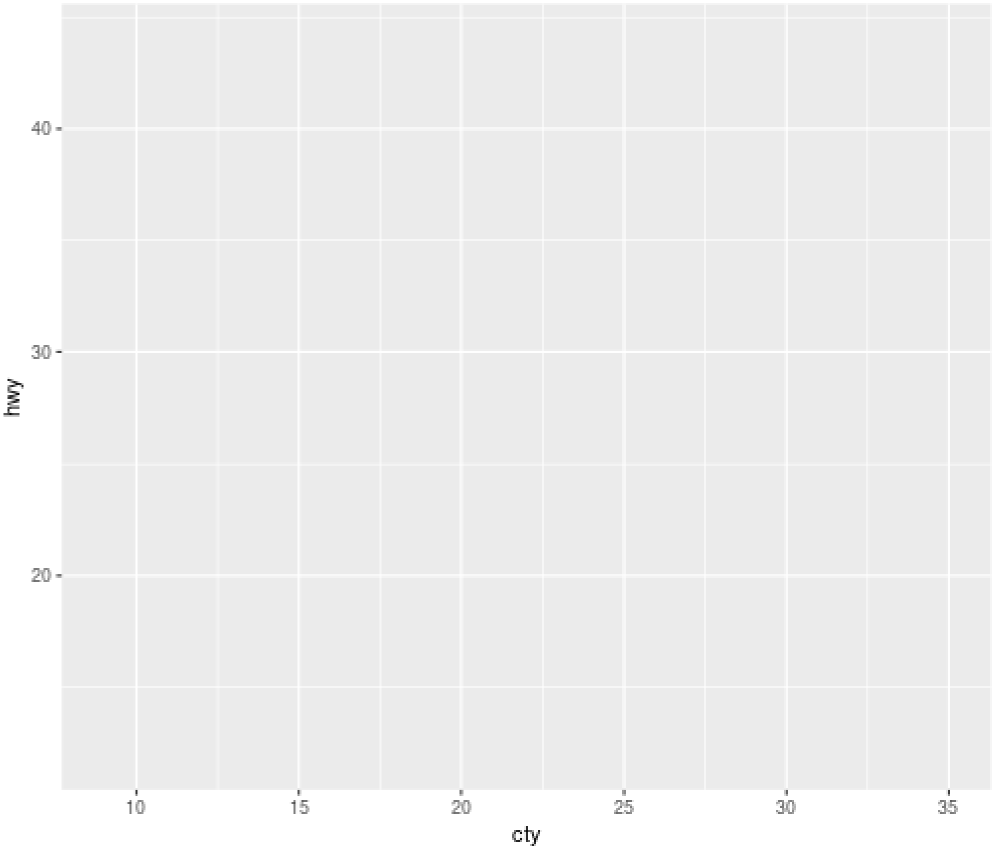
It is data that comes together with `ggplot2`.

ggplot object

- To make a plot, the first task is to create a ggplot object.
- You need to specify the data frame and the **aesthetic**.
- In a 2D plot we can compare two variables
- To start, think of the aesthetic as the x and y variable.
- Consider a plot to compare the fuel efficiency of a car in the city and on the highway.

Number of Cylinders v MPG

```
ggplot(mpg,aes(x=cty,y=hwy))
```

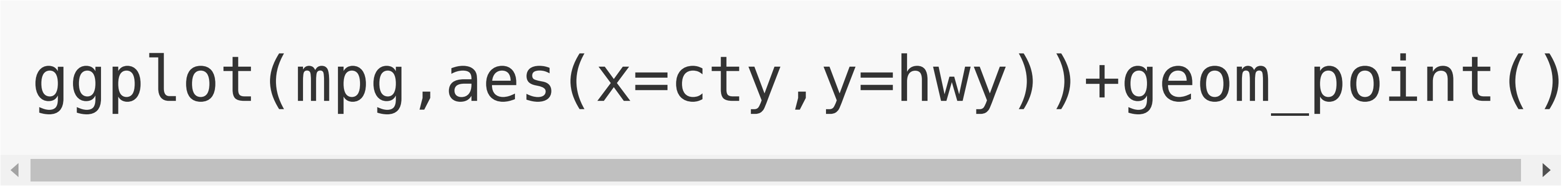


Geometry

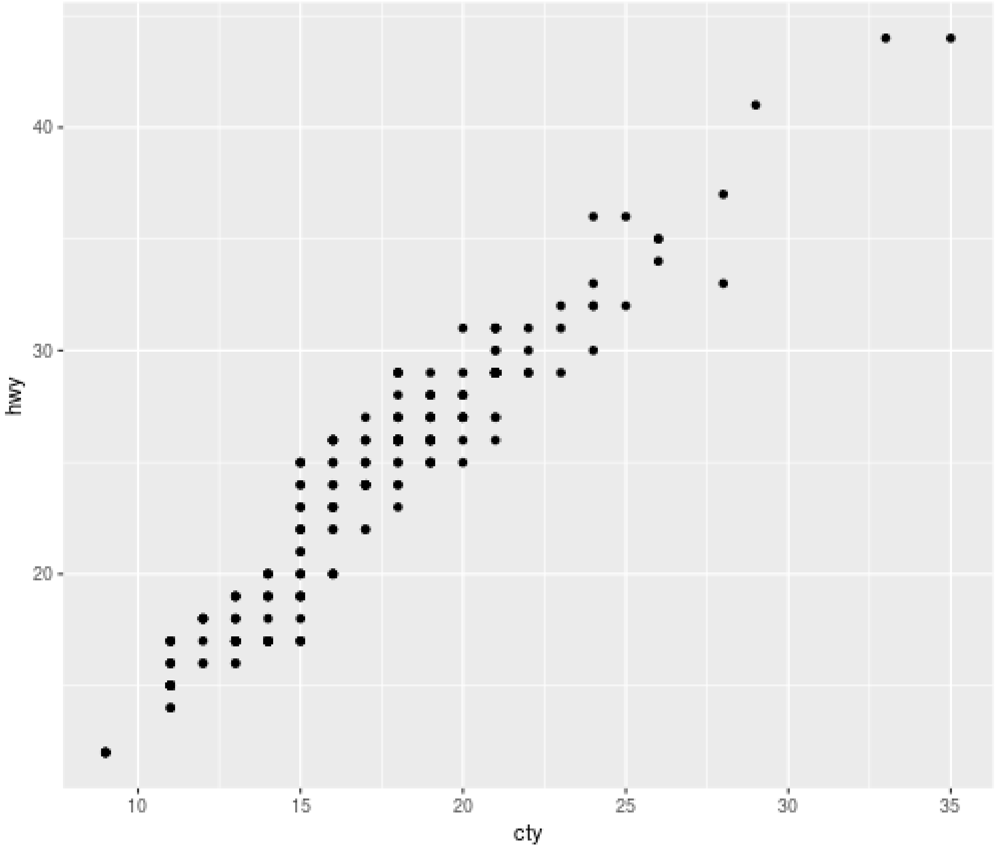
- This should produce some axes and labels but there is no plot yet.
- To produce a plot we need to tell R what type of plot we want.
- In the language of ggplot this is called a **geometry**
- For a scatter plot the geometry is

  `geom_point()`.

Scatterplot in R



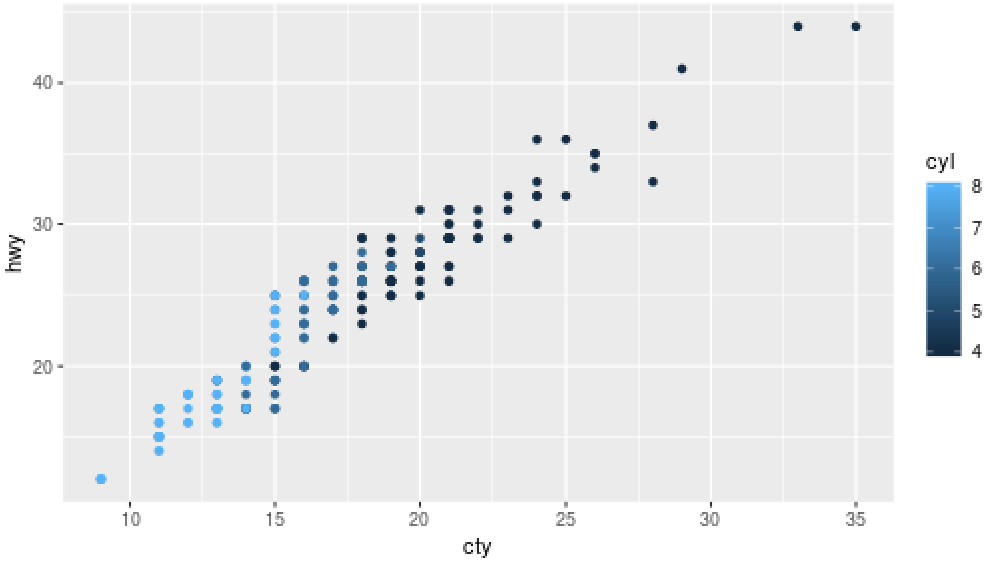
```
ggplot(mpg,aes(x=cty,y=hwy))+geom_point()
```



Other aesthetics

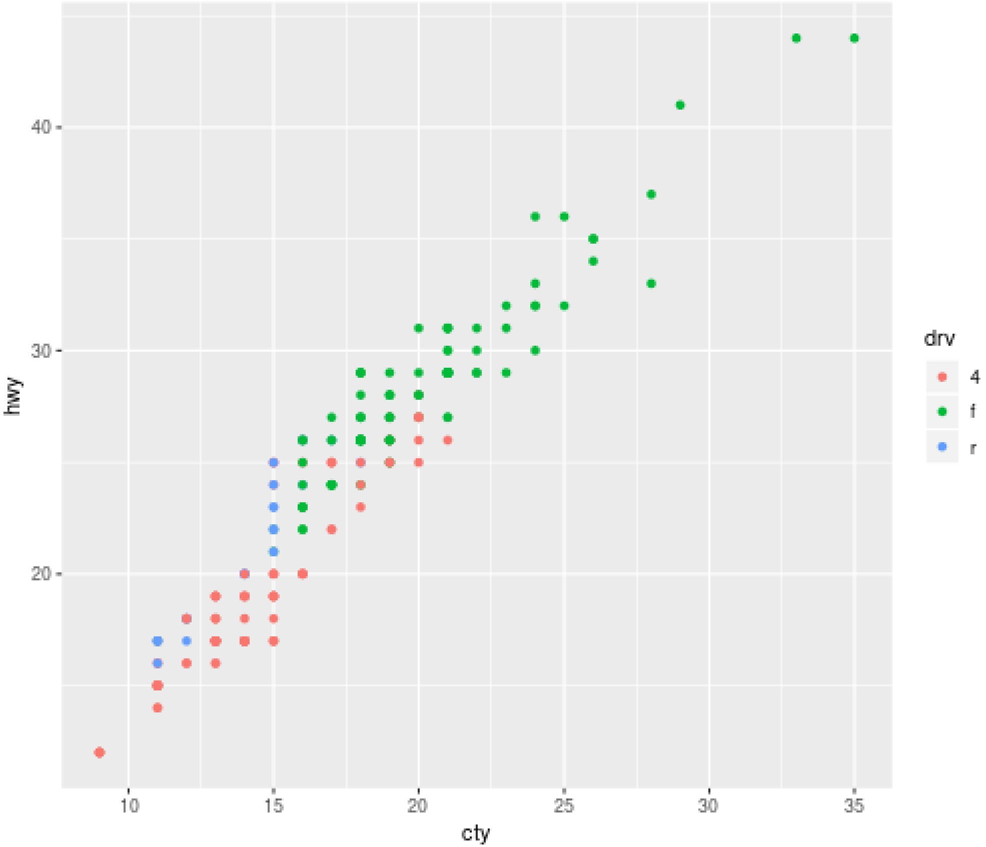
We can think of colour as a third aesthetic

```
ggplot(mpg,aes(x=cty,y=hwy,col=cyl))+geom
```



With a categorical variable

```
ggplot(mpg,aes(x=cty,y=hwy,col=drv))+geom
```



Quicker plots

A useful function in `ggplot2` is to use `qplot` which will try to guess the plot you want. Try these examples

```
qplot(x=cty,y=hwy,data=mpg)
qplot(x=cty,data=mpg)
```

Exporting graphics

- You can export graphics using the *Export* tab in the *Plot* tab in Rstudio.
- Many different file formats are available. As an alternative you can do the following:

```
pdf('myplot.pdf')
qplot(x=cty,data=mpg)
dev.off()
```

- Other file formats such an png or jpeg can be used instead of pdf.

Data Manipulation

- There are several ways to manipulate data, but a particularly useful and easy package to use is called **dplyr**.
- We can exclude observations using the `filter` function.
- To really understand how to use this function it helps to know about **logical operators** (try `?Logic`) and **relational operators** (try `?Comparison`).
- We will do a few simple examples here

Using dplyr

To create a new data frame that only includes 4 wheel drives

```r
library(dplyr)
mpg_4wd<-filter(mpg,drv=='4')
```

To exclude all 4 wheel drives

```r
mpg_no4wd<-filter(mpg,drv!='4')
```

Two conditions

Suppose we only want to consider cars that are 4 wheel drives **and** can drive more than 15 miles per gallon on the highway

```
mpg_4wd_hwyg15<-filter(mpg,(drv=='4')&(hw
```

Or those that are either 4 wheel drives **or** can drive less than 15 miles per gallon in the city

```
mpg_4wd_ctyl15<-filter(mpg,(drv=='4')|(ct
```

Without dplyr

- This sort of data manipulation can be done without `dplyr` but is more verbose.
- For example the last line would be

```
mpg_4wd_cty15<-mpg[((mpg$drv=='4')|(mpg$
```

- Both give the same result, use whichever you prefer.

Summarise Fuction

- Suppose we want the mean and standard deviation of the (filtered) data.

```
mpg_4wd_cty15<-filter(mpg,(drv=='4')|(ct
mean_sd_hwy<-summarise(mpg_4wd_cty15,mea
mean_sd_hwy
```

```
## # A tibble: 1 x 2
##   `mean(hwy)` `sd(hwy)`
##         <dbl>     <dbl>
## 1          19      3.91
```

Pipes

- Pipes from the *magrittr* package make this easier.

```
filter(mpg,(drv=='4')|(cty<15))%>%
  summarise(mean(hwy),sd(hwy))%>%
  print
```

```
## # A tibble: 1 x 2
##   `mean(hwy)` `sd(hwy)`
##         <dbl>     <dbl>
## 1          19      3.91
```

Conclusion

- This lecture has given you a foundation in R
- You can use R to do much more including collecting data off the web, cleaning the data, fitting models to the data, creating web applications, or even creating documents (these slides were created in RStudio).
- This can be daunting, but remember the best thing about R is that there are lots of ways to teach yourself R.