# A very brief Introduction to R for Data Science

Slides by: Mark Carman

# What is R?

- A language for **analysing** and **visualising** data
  - interpreted (scripting) language, so no need to compile code
  - designed by statisticians
  - open-source
  - very popular!

- Alternatives:
  - Python, Matlab, SAS, …

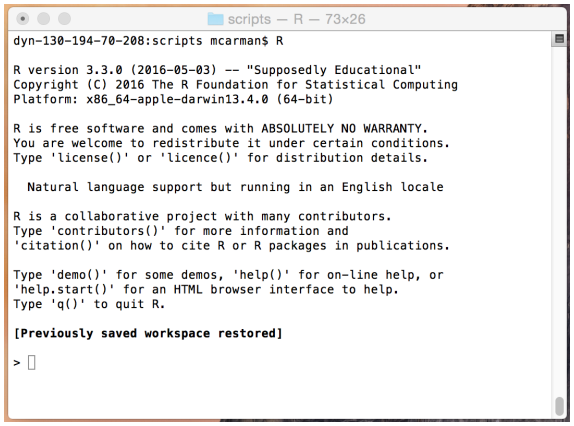
# Getting started

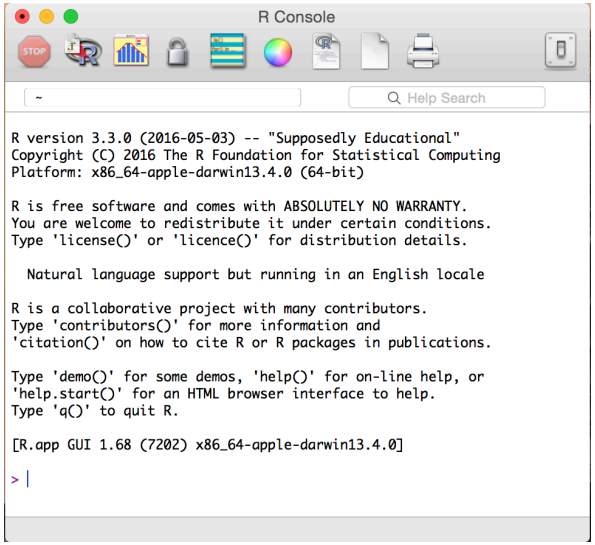
Installing R:

- available for download from the R project website:
  - http://www.r-project.org/
- or get the Rstudio IDE (Integrated Development Environment) from:
  - http://www.rstudio.com/products/rstudio/
  - both open source and commercial versions

Running R:

- either type “R” in a shell (Linux/MacOS)
- or start the R console or R-Studio application (Windows/MacOS)



# Basic R Syntax

- Compute mathematical expressions:

```
> 2^3+2
[1] 10
```

Here > denotes the command prompt & the output is prefixed by: [1]

- Define variables and assign values:

```
> A <- 10
> 5*A +6
[1] 56
```

It is traditional in R to use left-arrow for assignment, but you can also use equals:

```
> A = 10
```

- Define a vector using the concatenate function:

```
> B <- c(5,6,3,0)
> B
[1] 5 6 3 0
```

You **must** use the concatenate function `c()` to build a vector, just writing `(5,6,3,0)` won't work!

- Concatenate function can be applied to vectors too:

```
> B <- c(B,c(1,2))
> B
[1] 5 6 3 0 1 2
```

# Vectors in R

- Access element in a vector:

```
> B <- c(5,6,3,0,1,2)
> B[1]
[1] 5
```

**Unlike Python**, the first element of an array has index 1 (not 0)

- Access range of values in vector:

```
> B[1:3]
[1] 5 6 3
```

The colon operator `1:n` generates a vector of integers from 1 to n, **inclusive**:

```
> 1:3
[1] 1 2 3
```

- Operations can be performed on vectors directly, and are interpreted in an element-wise fashion:

```
> B^2
[1] 25 36 9 0 1 4
> B + 2
[1] 7 8 5 2 3 4
```

Singletons are automatically expanded to vectors. Vectors must be of same length

- e.g. `B[1:3]+B[1:2]` would return an error

# Data Frames

- We can combine vectors together to form a table, called a "data frame" in R:

```
> names <- c("Bill","Ted","Henry","Joan")
> ages <- c(76,82,104,78)
> heights <- c(1.55,1.69,1.49,1.57)
> my_table <- data.frame(names,ages,heights)
> my_table
  names ages heights
1  Bill   76    1.55
2   Ted   82    1.69
3 Henry  104    1.49
4  Joan   78    1.57
```

- We can then select columns either by name or index:

```
> my_table['names']
  names
1  Bill
2   Ted
3 Henry
4  Joan
> my_table[1]
  names
1  Bill
2   Ted
3 Henry
4  Joan
```

To select multiple columns, just combine them as a vector:

```
> my_table[c(1,3)]
  names heights
1  Bill    1.55
2   Ted    1.69
3 Henry    1.49
4  Joan    1.57
```

# Data Frames (cont.)

- Selecting row(s) by appending a comma:

```
> my_table[1,]
  names ages heights
1  Bill   76    1.55
> my_table[2:4,]
  names ages heights
2   Ted   82    1.69
3 Henry  104    1.49
4  Joan   78    1.57
```

Strange looking syntax for selecting rows is due to fact that in R, tables are matrices that are indexed by `[row,column]` (i.e. row first)

- Select particular cells by [row,column]:

```
> my_table[1,2]
[1] 76
> my_table[3:4,2:3]
  ages heights
3  104    1.49
4   78    1.57
```

- We often also refer to a variable (a column) by using the $ syntax:

```
> my_table$ages
[1]  76  82 104  78
> my_table$ages[3]
[1] 104
```

# Loading & Saving data

- Loading data from a CSV file

```
> my_data <- read.table("my_data.csv",header=TRUE,sep=",")
```

- Call the str function to see the column names and data types in the resulting table

```
> str(my_table)
'data.frame':4 obs. of  3 variables:
 $ names  : Factor w/ 4 levels "Bill","Henry",..: 1 4 2 3
 $ ages   : num  76 82 104 78
 $ heights: num  1.55 1.69 1.49 1.57
```

- If a file is big, we don't want to print it all out, just to have a look at it. Instead we can inspect the first/last lines of the table

```
> head(my_data)
...
> tail(my_data)
...
```

- Saving results to a CSV file

```
> write.csv(my_data,file="my_data.csv")
```

# Summary Statistics

We can easily compute lots of summary statistics:

- such as minimum and average values:

```
> min(my_table$ages)
[1] 76
> mean(my_table$height)
[1] 1.575
```

- standard deviations:

```
> sd(my_table$height)
[1] 0.08386497
```

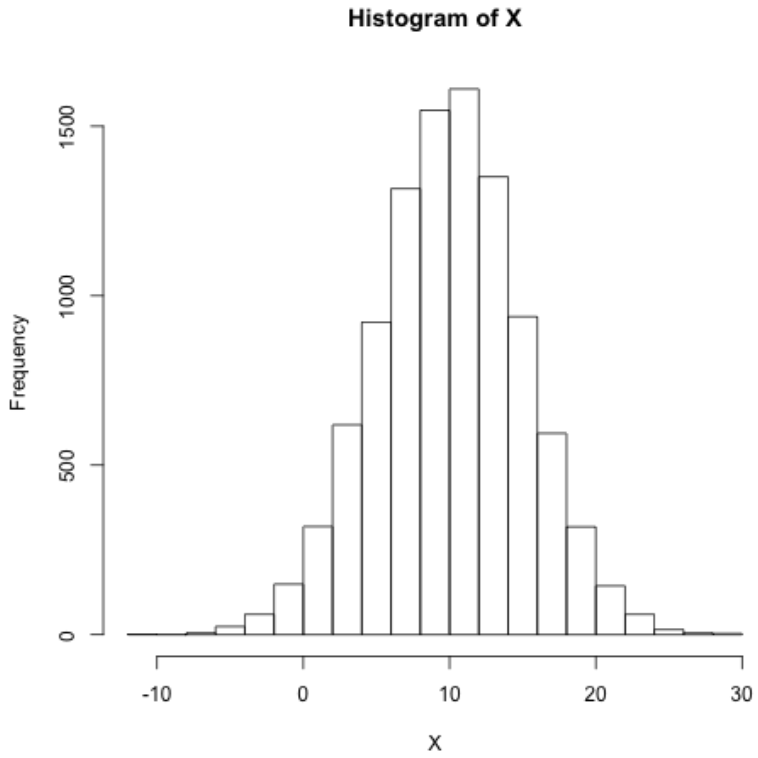
- or medians and quartiles:

```
> summary(my_table$height)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.490   1.535   1.560   1.575   1.600   1.690
```

# Visualising data: Histogram

- We can inspect the distribution of values for a particular variable by plotting it as a histogram

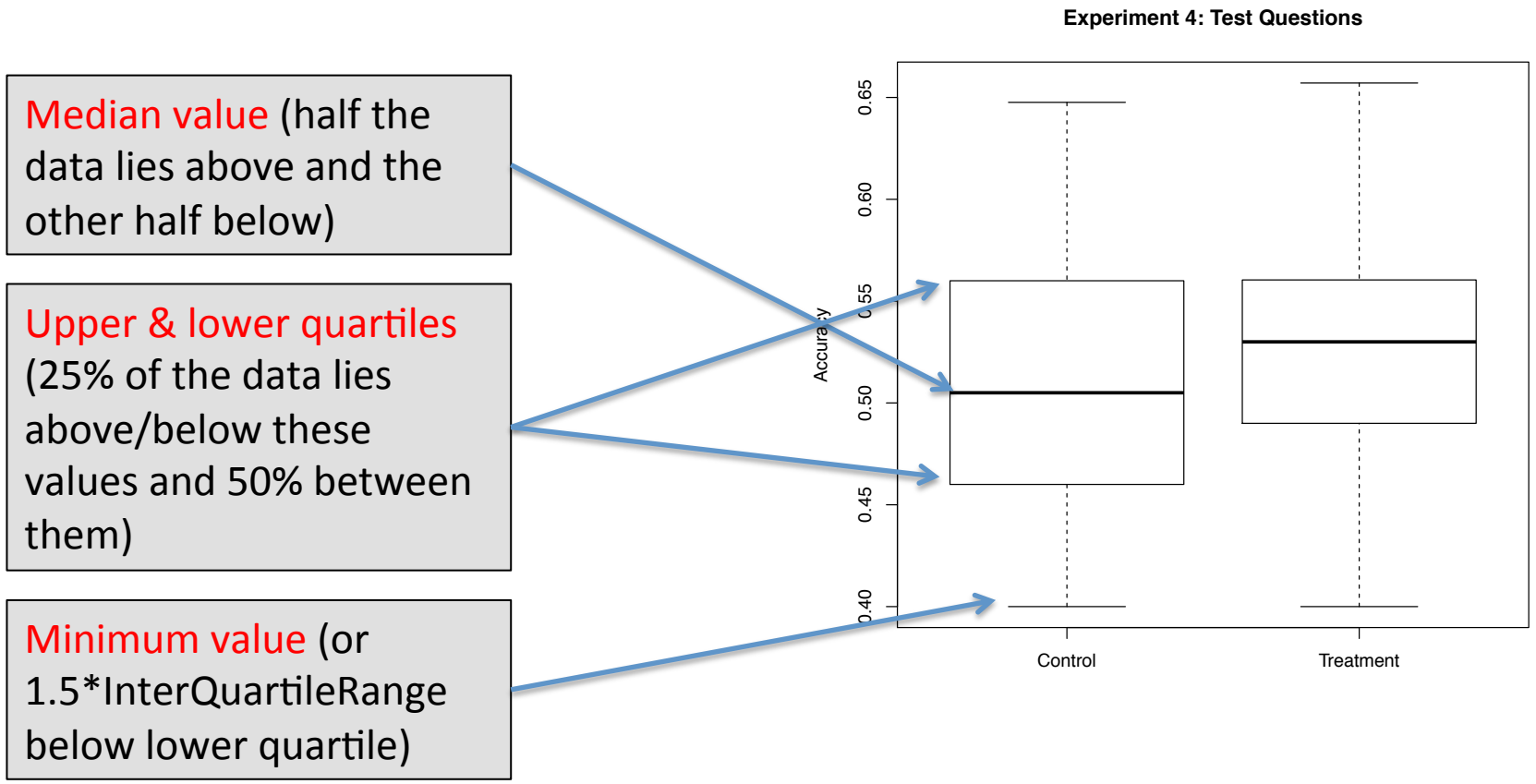
```
> hist(X,data=my_data)
```



# Visualising data: boxplot

- or its summary statistics by plotting it as a boxplot

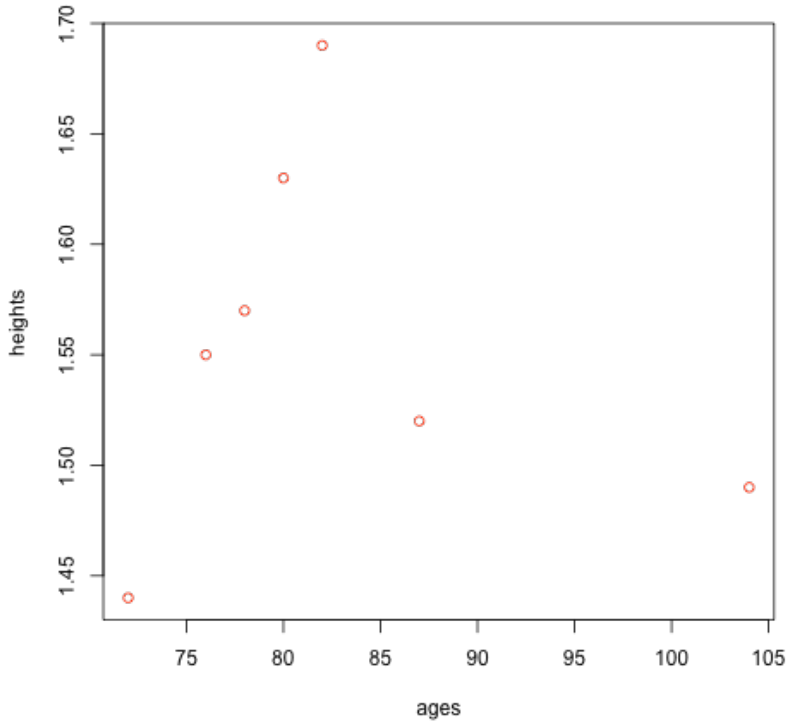
```
> boxplot(Accuracy~Group,data=my_data)
```



# Visualising Data: Scatter

- Or the variation of one variable against another by plotting data as a scatterplot

```
> plot(heights~ages,data=my_table,col="red")
```



# Fitting a Linear Model

Often we'd like to see if there exists a linear trend relationship between two variables.

- Fitting a linear model in R is very simple

```
> fit <- lm(height~age,data=myData)
```

- Print out summary information regarding the fit (the slope, etc.)

```
> summary(fit)
```

- Add the line of best fit to a graph

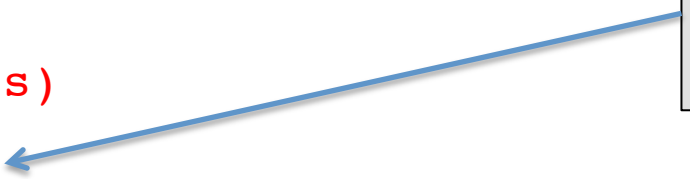
```
> abline(fit,col='red')
```

# Loading libraries

- Libraries are lists of functions that are not available in R by default.
- Loading a library

```
> library(moments)
> skewness(data)
```

The `skewness()` function is provided by the `moments` library



- Before loading a library for the first time you will need to install the package on your machine:

```
> install.packages("moments")
```

# End Of Introduction

- We'll be experimenting with R in this week's tutorial
- There are MANY excellent R resources online if you'd like to learn more. For example:
  - lynda.com
  - datacamp.com