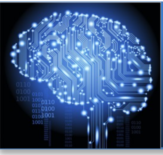


FIT1043 Lecture 8
Introduction to Data Science

Mahsa Salehi

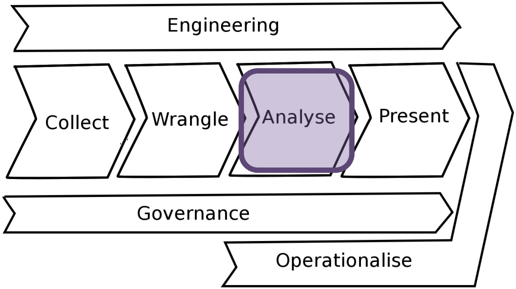
Faculty of Information Technology, Monash University

Semester 2, 2022

Unit Schedule

Week	Activities	Assignments
1	Overview of data science	Weekly Lecture/tutorial active participation assessment
2	Introduction to Python for data science	
3	Data visualisation and descriptive statistics	
4	Data sources and data wrangling	
5	Data analysis theory	Assignment 1
6	Regression analysis	
7	Classification and clustering	
8	Introduction to R for data science	
9	Characterising data and "big" data	Assignment 2
10	Big data processing	
11	Issues in data management	
12	Industry guest lecture	Assignment 3

Our Standard Value Chain



Last Week

This Week

Tools for data science

Outline

- Motivation to study R
- R data types
- Essential libraries
    - Wrangling
    - Exploration and analysis
    - Visualisation

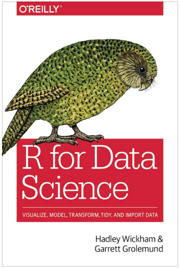
Learning Outcomes (Week 8)

By the end of this week you should be able to:

- Comprehend essentials for coding in R for data science
- Explain and interpret given R commands
- Apply R commands for data wrangling, visualisation, exploration and analysis

Introduction to R for Data Science

R for Data Science by H. Wickham and G. Grolemund



Top 6 Data Science Programming Languages for 2019

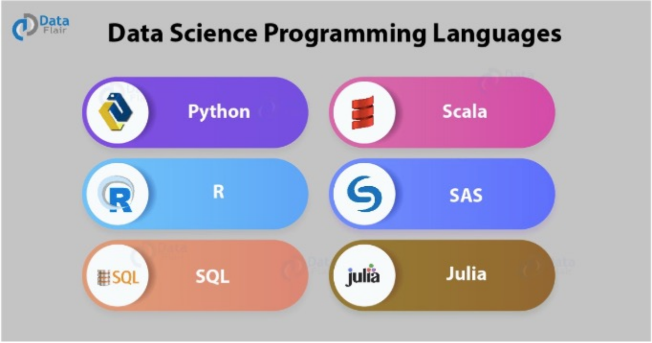


Image src: Data Flair

What is R?

- ➢ Data science preferred tools
- ➢ A language for **analysing** and **visualising** data
    - Interpreted (scripting) language, so no need to compile code
    - Designed by statisticians
    - Open-source
    - Very popular!

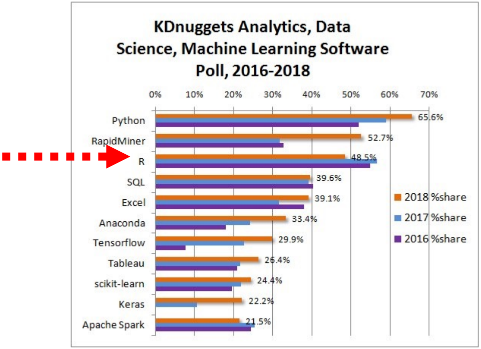


image src: kdnuggets.com

FLUX Question

- R Or Python

R Vs Python

Parameter	R	Python
Objective	Data analysis and statistics	Deployment and production
Flexibility	Easy to use available library	Easy to construct new models from scratch.
Important Packages and library	tydiverse, ggplot2, caret, zoo	pandas, scipy, scikit-learn, TensorFlow, caret
Disadvantages	Steep Learning curve Dependencies between library	Not as many specialized packages for statistical computing as R
Comparison	• Functional • More statistical support in general	• Object Oriented • More straightforward to do non-statistical tasks

Setting Up R Environment

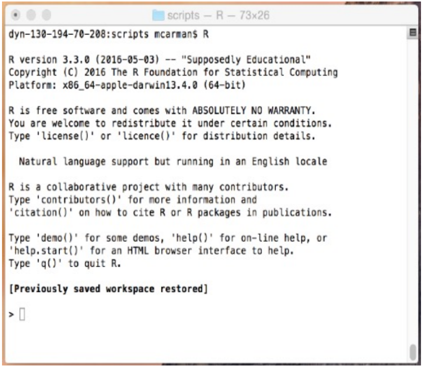
Installing R:

- Available for download from the R project
  - https://www.r-project.org/
- Get the **Rstudio IDE** (Integrated Development Environment) from:
  - https://www.rstudio.com/products/rstudio/
  - Both open source and commercial versions

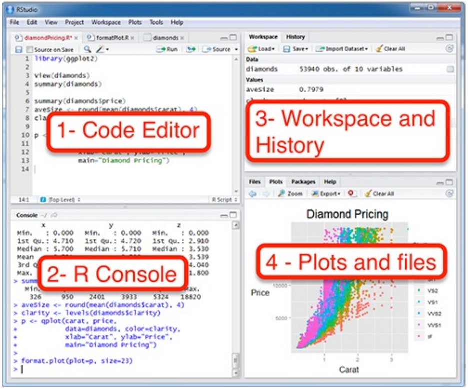
Running R:

- Either type “R” in a shell (Linux/MacOS)
- Or start R-Studio application (Windows/MacOS)

R Environment



```
dyn-130-194-70-208:scripts mcarman$ R

R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> 
```



```
library(ggplot2)

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
cla

p <


              main="Diamond Pricing")
```

```
diamonds      53940 obs. of 10 variables
avesize       0.7979
```

```
        x               y               z
Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
Median : 5.700   Median : 5.710   Median : 3.530
Mean                                     : 3.539
3rd Q                                    : 4.040
Max.                                     :31.800
> summ
  Min.                                      Max.
   326     950    2401    3933    5324   18820
> aveSize <- round(mean(diamonds$carat), 4)
> clarity <- levels(diamonds$clarity)
> p <- qplot(carat, price,
+            data=diamonds, color=clarity,
+            xlab="Carat", ylab="Price",
+            main="Diamond Pricing")
>
> format.plot(plot=p, size=23)
> 
```

1- Code Editor

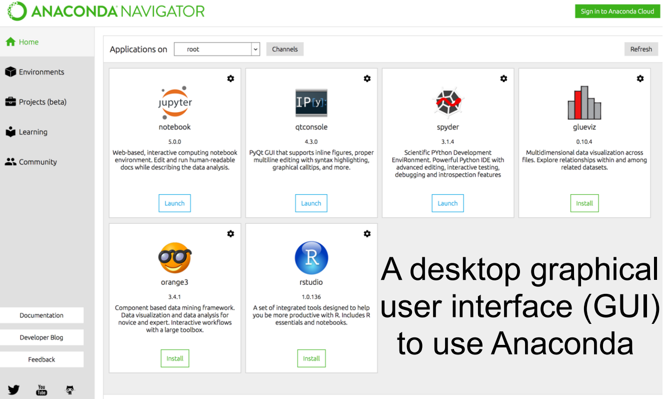
3- Workspace and History

2- R Console

4 - Plots and files

Anaconda Project

- Anaconda is a package manager, an environment manager, a Python/R data science distribution, and a collection of over 1,500+ open source packages. Anaconda is free and easy to install.



ANACONDA NAVIGATOR

Sign in to Anaconda Cloud

Home

Environments

Projects (beta)

Learning

Community

Documentation

Developer Blog

Feedback

Applications on root Channels Refresh

notebook
5.0.0
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

qtconsole
4.3.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch

spyder
3.1.4
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features
Launch

glueviz
0.10.4
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install

orange3
3.4.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install

rstudio
1.0.136
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

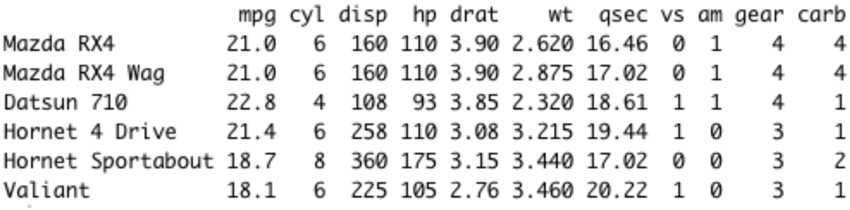
A desktop graphical user interface (GUI) to use Anaconda

Built-in Data Sets in R

- List available data set :
  > data()

- Load a built-in dataset
  > data(mtcars)

- Inspect the data set
  > head(mtcars, 6)

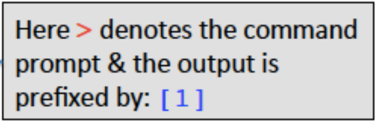


```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Basic R Syntax

- Compute mathematical expressions:

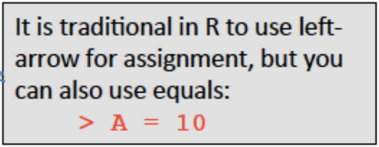
```
> 2^3+2
[1] 10
```



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

- Define variables and assign values:

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

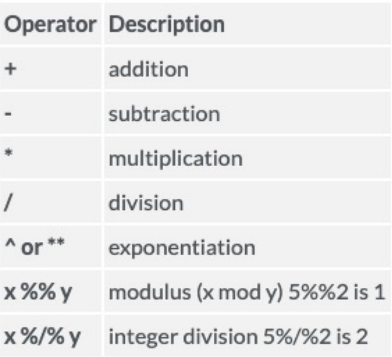


It is traditional in R to use left-arrow for assignment, but you can also use equals:

```
> A = 10
```

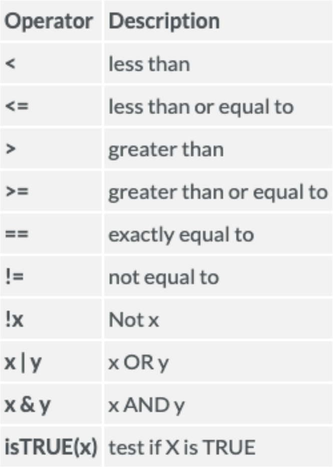
Operators

- Arithmetic Operators



Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

- Logical Operators



Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if X is TRUE

image src: Quick_R

If Else Condition

Syntax: *statement* would be executed if *expression* is *TRUE*

```
if(expression)
{
statement/s
}
```

Example:

```
> x <- 10
> if(x>0)
  {
  print(“This is Positive Number)
  }
[1] "This is Positive number"
```

For Loop

Syntax: statement would be executed n-times.

```
for(i in 1:n)
{
statement/s
}
```

Example:

```
> for(i in 1:3)
  {
  print(i^2)
  }
[1] 1
[1] 4
[1] 9
```

While Loop

Syntax:

```
while(condition)
{
statement/s
}
```

Example:

```
> i <- 1
> while (i <=6) {
  print(i*i)
  i = i+1
  }
  [1] 1
  [1] 4
  [1] 9
```

Break Statement

- **Break:** Stop the iteration and exit the loop.

Example:

```
> x <- 1:5
> for (i in x) {
    if (i == 3){
      break
    }
  print(i)
  }
[1] 1
[1] 2
```

Next Statement

- **Next:** Skip one step of the loop and jumps to the next cycle.

  Example:

```
> x <- 1:5
> for (i in x) {
      if (i == 3){
         next
      }
    print(i)
    }
 [1] 1
 [1] 2
 [1] 4
 [1] 5
```

Basic Data Types

- Numeric

```
> x <- 10.5
```

- Integer

```
> x <- as.integer(10.5)
```

- Complex

```
> x <- 1+2i
```

- Logical

```
> x <- TRUE
```

- Character

```
> x <- “Intro To R”
```

Basic Data Types(Cont.)

- Print the class name of y

```
> y <- 8
> class(y)
[1] "numeric”
```

- Is y an integer?

```
> is.integer(y)
[1] FALSE
```

- Change data type

```
 > as.character(y)
 [1] "8”
```

- Getting help

```
> help(c)
```

What is Vector?

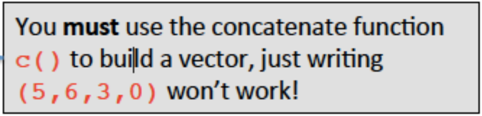
R has **c()** built in function which allows to store more than one value.

- Define a vector using the concatenate function:

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

- Concatenate function can be applied to vectors too:

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

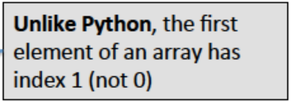


You **must** use the concatenate function `c()` to build a vector, just writing `(5,6,3,0)` won't work!

Accessing Vector Elements

- Accessing vector elements using position

```
> x <- c("Jan","Feb","Mar","April")
> y <- x[c(1,3,4)]
> print(y)
[1] ”Jan"  "Mar"  ”April”
```



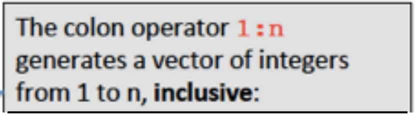
Unlike Python, the first element of an array has index 1 (not 0)

- Accessing vector elements using negative indexing

```
> t <- x[c(-1,-4)]
> print(t)
[1] "Feb" "Mar"
```

- Access range of values in vector

```
> x[1:3]
[1] "Jan" "Feb" "Mar"
```



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

Vector Arithmetic Operation

Operations can be performed on two vectors (**same length**) directly and are interpreted in an element-wise fashion.

- Create two vectors.

```
> v1 <- c(1,2,4,5,7,11)
> v2 <- c(12,4,3,8,1,21)
```

- Vector multiplication.

```
 > multi.result <- v1*v2
 > print(multi.result)
 [1]  12   8  12  40   7 231
```

FLUX Question

- How to check if a vector contains missing values?

What is Data Frame?

We can combine vectors together to form a table, called a “data frame”

- Create the data frame

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

Rename The Columns Of Data Frame

R has **names(df)** built in function which allows you to rename data frame columns

- Pass a vector of new names to the function

```
> names(myTable)<-c("Names", "Ages", "Heights")
> print(myTable)
  Names Ages Heights
1   Bill     76    1.55
2   Ted      82    1.69
3   Henry  104    1.49
4   Joan    78     1.57
```

Accessing Elements of Data Frame

- Number of rows in data frame

  ```
  > nrow(myTable)
  [1] 4
  ```

- Number of columns in data frame

  ```
  > ncol(myTable)
  [1] 3
  ```

- Dimension of data frame

  ```
  > dim(myTable)
  [1] 4 3
  ```

Get the Structure of the Data Frame

- Display the column names and data types

```
> str(myTable)
```

```
'data.frame':  4 obs. of  3 variables:
 $ Names  : chr  "Bill" "Ted" "Henry" "Joan"
 $ Ages   : num  76 82 104 78
 $ Heights: num  1.55 1.69 1.49 1.57
```

Summary Statistic

- Minimum value

  ```
  > min(myTable$Ages)
  [1] 76
  ```

- Average value

  ```
  > mean(myTable$Heights)
  [1] 1.575
  ```

- Standard deviation

  ```
  > sd(myTable$Heights)
  [1] 0.08386497
  ```

Summary of Data Frame

```
> summary(myTable)
```

```
    Names                 Ages          Heights
 Length:4           Min.   : 76.0   Min.   :1.490
 Class :character   1st Qu.: 77.5   1st Qu.:1.535
 Mode  :character   Median : 80.0   Median :1.560
                    Mean   : 85.0   Mean   :1.575
                    3rd Qu.: 87.5   3rd Qu.:1.600
                    Max.   :104.0   Max.   :1.690
```

Extracting Data From Data Frame

- Accessing column/s by name

  ```
  > myTable[“Ages"]
  ```

- Accessing multiple columns by name

  ```
  > myTable[c(“Names",“Ages")]
  ```

- Accessing columns by index

  ```
  > myTable[2]
  ```

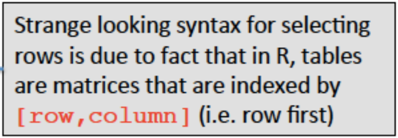
- Accessing multiple columns by index

  ```
  > myTable[c(1,2)]
  ```

Extracting Data From Data Frame(Cont.)

- Accessing first row and all the columns by appending comma

```
> myTable[1,]
  Names Ages Heights
1  Bill     76    1.55
```



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)

- Accessing a range of rows and all the columns

```
> myTable[2:4,]
  Names Ages Heights
2   Ted     82      1.69
3  Henry  104    1.49
4  Joan    78      1.57
```

Extracting Data From Data Frame(Cont.)

- Accessing particular cells by [row,column]

```
> myTable[1,2]
[1] 76
> myTable[3:4,2:3]
 Ages Heights
3  104   1.49
4  78     1.57
```

- Referring to a variable (a column) by using the $ syntax:

```
 > myTable$Ages
 [1] 76 82 104 78
 > myTable$Ages[3]
 [1] 104
```

Sorting Data in Data Frame

- Sort by Ages

  ```
  > newData <- myTable[order(myTable$Ages),]
  ```

- Sort by Ages (ascending)

  ```
  > newData <- myTable[order(myTable$Ages,
  decreasing = TRUE), ]
  ```

- Sort by Ages and Heights

  ```
  > newData <-
  myTable[order(myTable$Ages,myTable$Heights),]
  ```

Merging Data in Data Frame

merge(): Used to merge two data frames by common key variable/s

- Merge two data frames by ID

  ```
  > total <- merge(myTable, myTable2, by=”Names")
  ```

rbind() Used to join two data frames vertically(Must have same number of variables)

- Join two data frames

  ```
  > total <- rbind(myTable, myTable2)
  ```

Aggregating Data in Data Frame

- Aggregate data frame mtcars by cyl and vs, returning means for numeric variables

```
> aggData <- aggregate(mtcars, by=list(cyl,vs),
                                 FUN=mean)
> print(aggdata)
```

Getting & Setting Working Directory

Before reading/writing in R it is important to specify the location where we can find the respective file to read/write.

- Get the current working directory.

```
> getwd()
[1] " C:/Users/username/FolderName”
```

- Set current working directory.

```
> setwd(" D:/FolderName ")
```

Writing CSV File

R has a **write.csv()** built in function to write data into a CSV file.

- Write a data into csv file (file is in current working directory)

  ```
  > write.csv(myTable, “FileName.csv")
  ```

- Read a csv file (file is in other location)

  ```
  > write.csv(myTable, “D:/FolderName/FileName.csv")
  ```

Reading CSV File

R has a **read.csv()** built in function to read a CSV file.

- Read a csv file (file is in current working directory)

  ```
  > myData = read.csv(“FileName.csv")
  >print(myData)
  ```

- Read a csv file (file is in other location)

  ```
  > myData = read.csv(“D:/FolderName/FileName.csv")
  >print(myData)
  ```

Displaying Data

- 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:

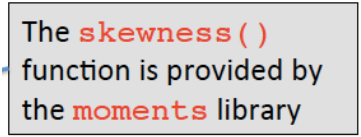
```
> head(myData)
...

> tail(myData)
 ...
```

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

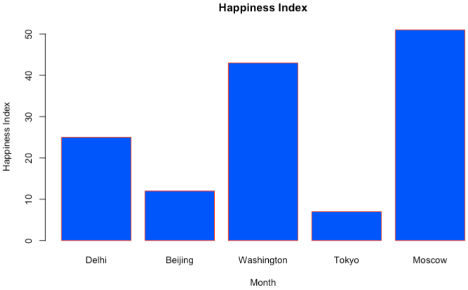
- Before loading a library for the first time you will need to install the package on your machine:

```
> install.packages(“moments”)
```

Visualising Data: Bar Chart

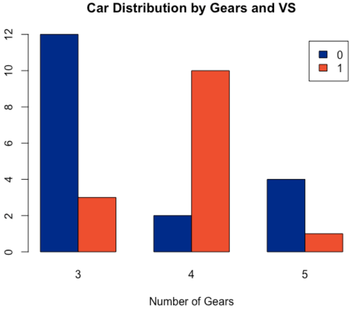
- Compare the value for categorical data using bar chart

```
> H <- c(25,12,43,7,51)
> M <- c("Delhi","Beijing","Washington","Tokyo","Moscow")
> barplot(H,xlab="Month",ylab="Happiness Index", col="blue",
            names.arg=M, main="Happiness Index",border="red")
```



Visualising Data: Group Bar Chart

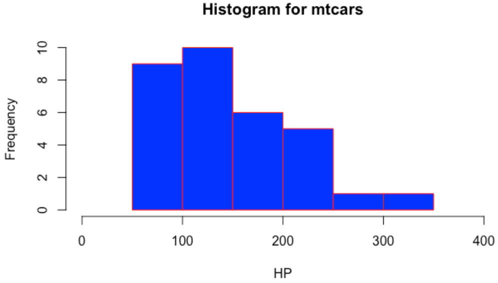
```
> counts <- table(mtcars$vs, mtcars$gear)
> barplot(counts, main="Car Distribution by Gears and VS", xlab =
            "Number of Gears", col=c("darkblue","red"),legend =
             rownames(counts),beside=TRUE)
```



Visualising Data: Histogram

Inspect the distribution of values for a particular variable by plotting it as a histogram

```
   > hist(mtcars$hp,main="Histogram for mtcars",xlab="HP",
border="red",col="blue",xlim=c(0,400))
```



Visualising Data: Boxplot

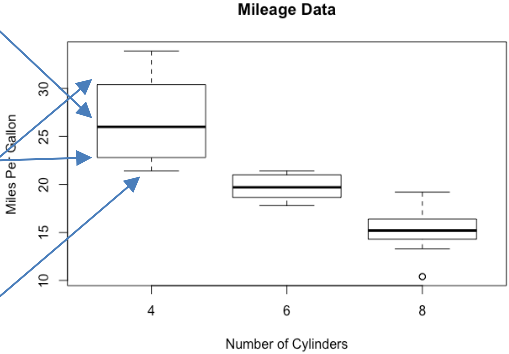
- Or its summary statistics by plotting it as a boxplot

```
> boxplot(mpg ~ cyl, data=mtcars, xlab="Number of
        Cylinders",ylab="Miles Per Gallon",main="Mileage Data")
```

Median value (half the data lies above and the other half below)

Upper & lower quartiles (25% of the data lies above/below these values and 50% between them)

Minimum value (or 1.5*InterQuartileRange below lower quartile)



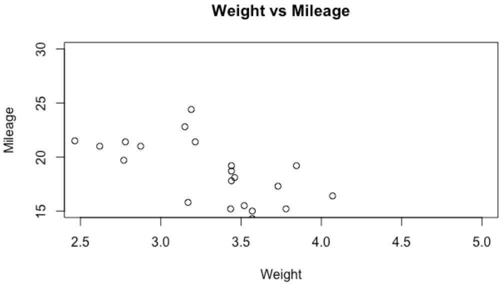
FLUX Question

How to find outliers in R?

Visualising Data: Scatter Plot

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

```
> input <- mtcars[,c('wt','mpg’)]
> plot(x=input$wt,y=input$mpg, xlab="Weight", ylab="Mileage",
        xlim=c(2.5,5), ylim=c(15,30), main="Weight vs Mileage")
```



Linear Regression

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

- Creating sample Data for height and weight

```
> height <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> weight <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
```

- Fitting a linear model in R is very simple

```
> fit <- lm(height~weight)
> print(fit)
Call:
lm(formula = height ~ weight)

Coefficients:
(Intercept)      weight
     61.380       1.415
```

Linear Regression(Cont.)

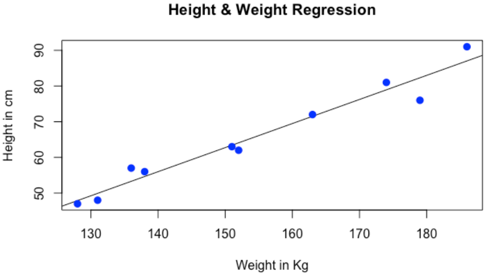
- Print out summary information regarding the fit (the slope, etc.)

```
> summary(fit)
Call:
lm(formula = height ~ weight)

Residuals:
      Min       1Q    Median       3Q      Max
  -6.0529  -2.4833   -0.0912   1.3774  10.0562

Coefficients:
            Estimate Std.   Error t   value     Pr(>|t|)
 (Intercept)  61.3803       7.2653     8.448   2.94e-05 ***
  weight        1.4153       0.1089    12.997   1.16e-06 ***
  ---
  Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

  Residual standard error: 4.712 on 8 degrees of freedom
  Multiple R-squared:  0.9548,          Adjusted R-squared:  0.9491
  F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

Visualize the Regression

- Plot the chart

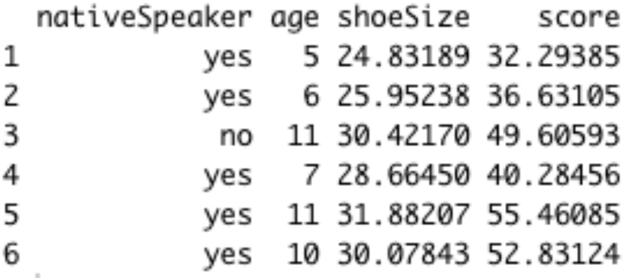
```
> plot(height, weight,col = "blue",main = "Height & Weight
      Regression”, abline(lm(weight ~ height)),cex = 1.3,pch= 16,
      xlab = "Weight in Kg",ylab = "Height in cm")
```



Decision Tree

- Install and load the party package.
  ```
  > install.packages("party")
  > library(party)
  ```

- Create the input data frame
  ```
  > inputData <- readingSkills[c(1:105),]
  > print(inputData)
  ```



```
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
```

Visualize The Decision Tree

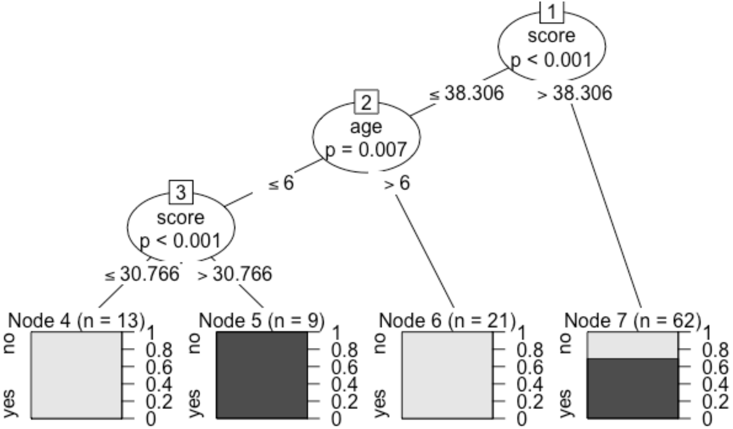
- Create the tree

```
> outputTree <- ctree( nativeSpeaker ~ age + shoeSize
                                   + score, data = inputData)
```

- Plot the tree

```
> plot(outputTree)
```

Visualize The Decision Tree(Cont.)



End Of Introduction

- You will work on a large data file in your tutorial/lab this week.
  - hourly ozone level readings across the US.

- There are MANY excellent R resources online if you'd like to learn more. For example:
  - lynda.com
  - datacamp.com