MONASH University

FIT1043 Introduction to Data Science

Week 8: Introduction to R for data science

Ts. Dr. Sicily Ting

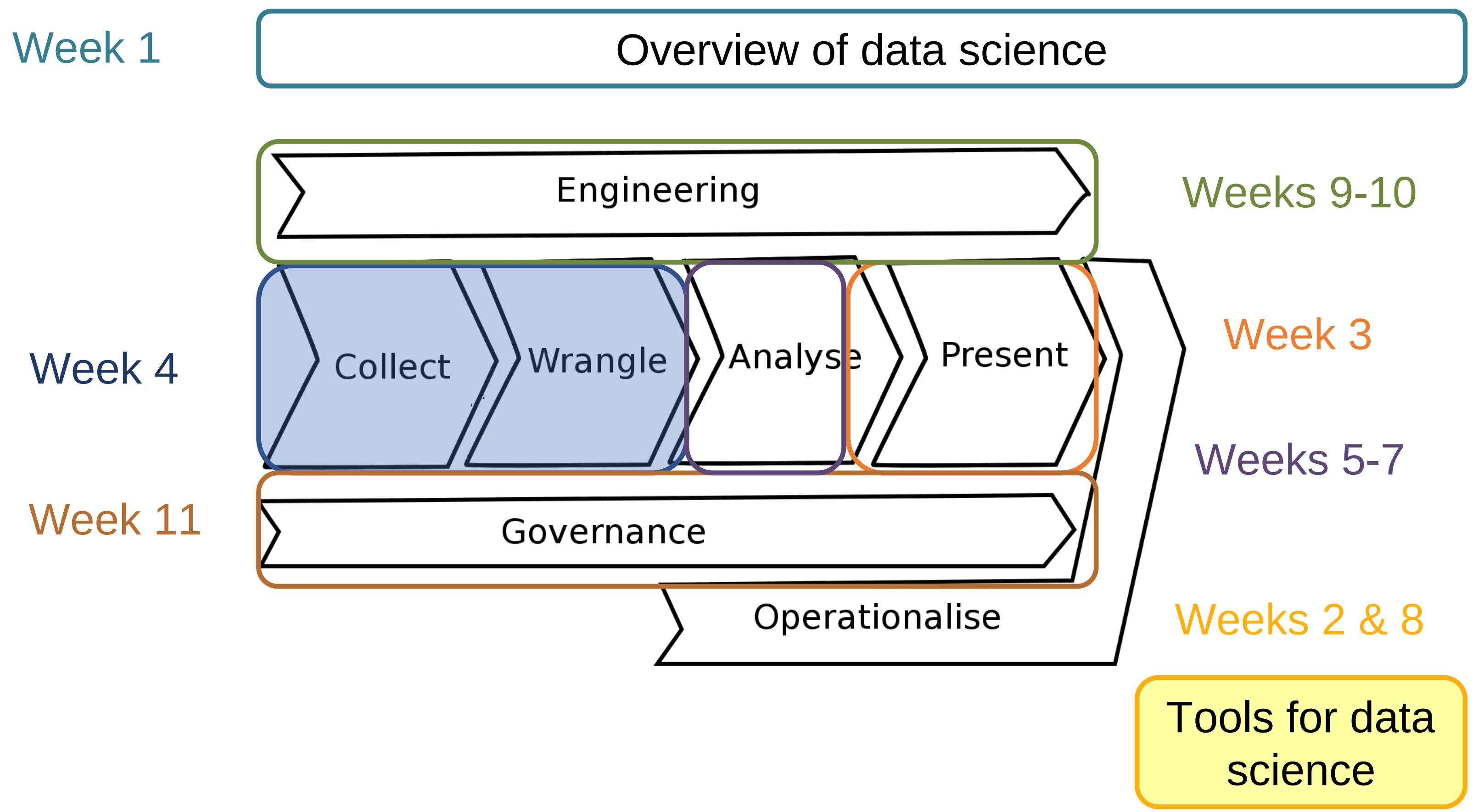
School of Information Technology
Monash University Malaysia

With materials from Wray Buntine, Mahsa Salehi

Week 7 Coverage
Classification
Clustering

MONASH University

Week	Activities	Assignments
1	Overview of data science	
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	Assignment 2
9	Characterising data and "big" data	
10	Big data processing	
11	Issues in data management	Assignment 3
12	Industry guest lecture	


Week 1
Overview of data science
Engineering
Weeks 9-10
Week 4
Collect
Wrangle
Analyse
Present
Week 3
Weeks 5-7
Week 11
Governance
Operationalise
Weeks 2 & 8
Tools for data science

Week 8 Outline

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

MONASH University

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

MONASH University

Data Science Programming Languages

https://www.analyticsinsight.net/top-10-data-science-programming-languages-for-2020/

Python

Python holds a special place among all other programming languages. It is an object-oriented, open-source, flexible and easy to learn a programming language and has a rich set of libraries and tools designed for data science. Also, Python has a huge community base where developers and data scientists can ask their queries and answer queries of others. Data science has been using Python for a long time and it is expected to continue to be the top choice for data scientists and developers.

R

R is a very unique language and has some really interesting features which aren't present in other languages. These features are very important for data science applications. Being a vector language, R can do many things at once, functions can be added to a single vector without putting it in a loop. As the power of R is being realized, it is finding use in a variety of other places, starting from financial studies to genetics and biology and medicine.

SQL

SQL (Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system. As the role of a data scientist is to turn raw data into actionable insights, therefore they primarily use SQL for data retrieval. To be an effective data scientist, they must know how to wrangle and extract data from the databases using SQL language.

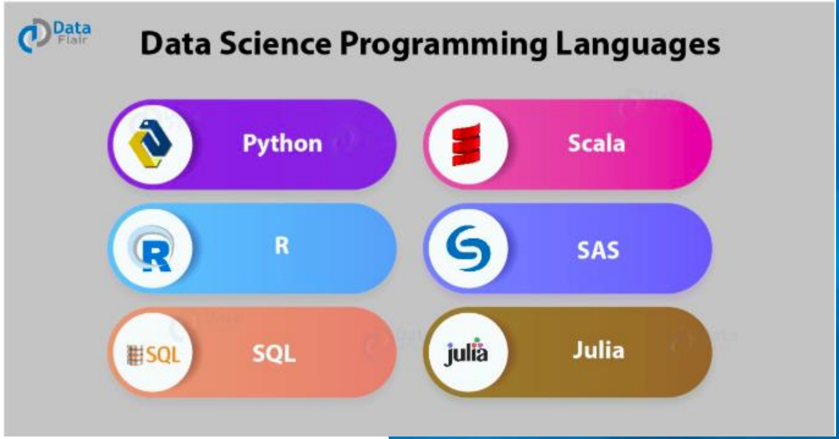


Image source: Data Flair

What is R?

https://towardsdatascience.com/top-programming-languages-for-ai-engineers-in-2020-33a9f16a80b0

- R

R was created by **Ross Ihaka** and **Robert Gentleman** with the **first version being launched in 1995**. Currently being maintained by the R Development Core Team, R is the implementation of S programming language and aids in developing statistical software and data analysis.

The qualities that are making R a good fit for AI programming among developers are:

> - *The fundamental feature of R being good at crunching huge numbers puts it in a better position than Python with its comparatively unrefined NumPy package.*
>
> - *With R, you can work on various paradigms of programming such as functional programming, vectorial computation and object-oriented programming.*

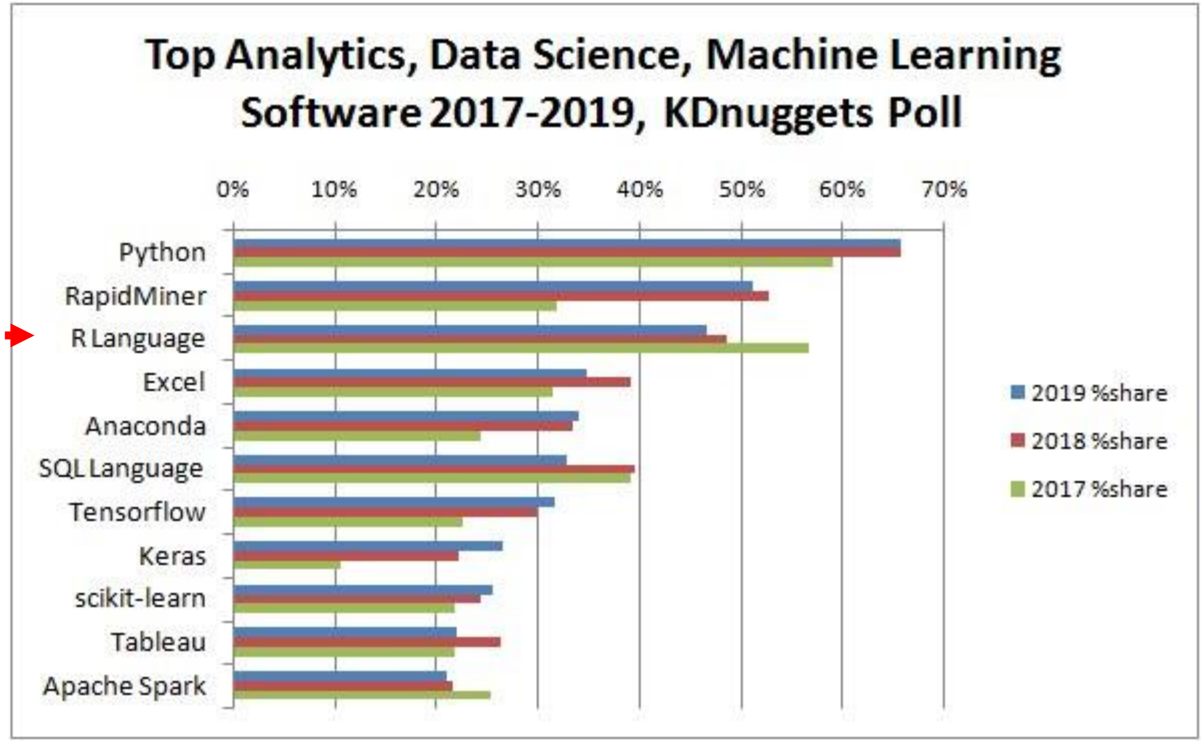
MONASH University

What is R?

https://www.kdnuggets.com/2019/05/poll-top-data-science-machine-learning-platforms.html

A language for analysing and visualising data

- Interpreted (scripting) language, so no need to compile code
- Designed by statisticians
- Open-source
- Very popular!



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	Tidyverse, ggplot2, caret, zoo	pandas, scipy, scikit-learn, TensorFlow, caret
Disadvantages	Slow, High Learning curve, Dependencies between library	Not as many specialized packages for statistical computing as R
Comparison	• Functional • More data analysis built-in • More statistical support in general	• Object Oriented • Relies on packages • More straightforward to do non-statistical tasks

Note: R is mainly used for statistical analysis while Python provides a more general approach to data science.

Setting Up R Environment

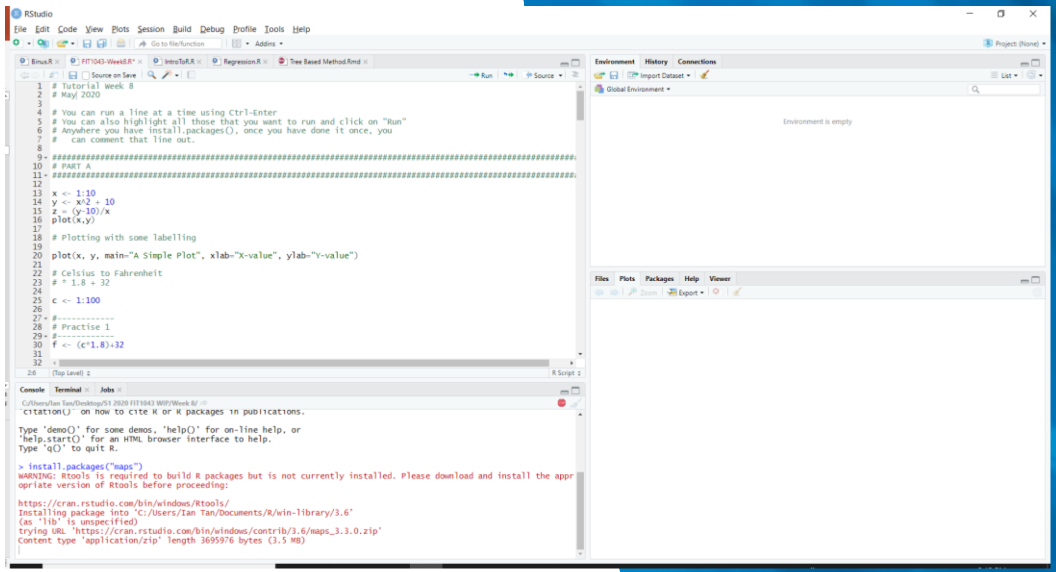
Installing R

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

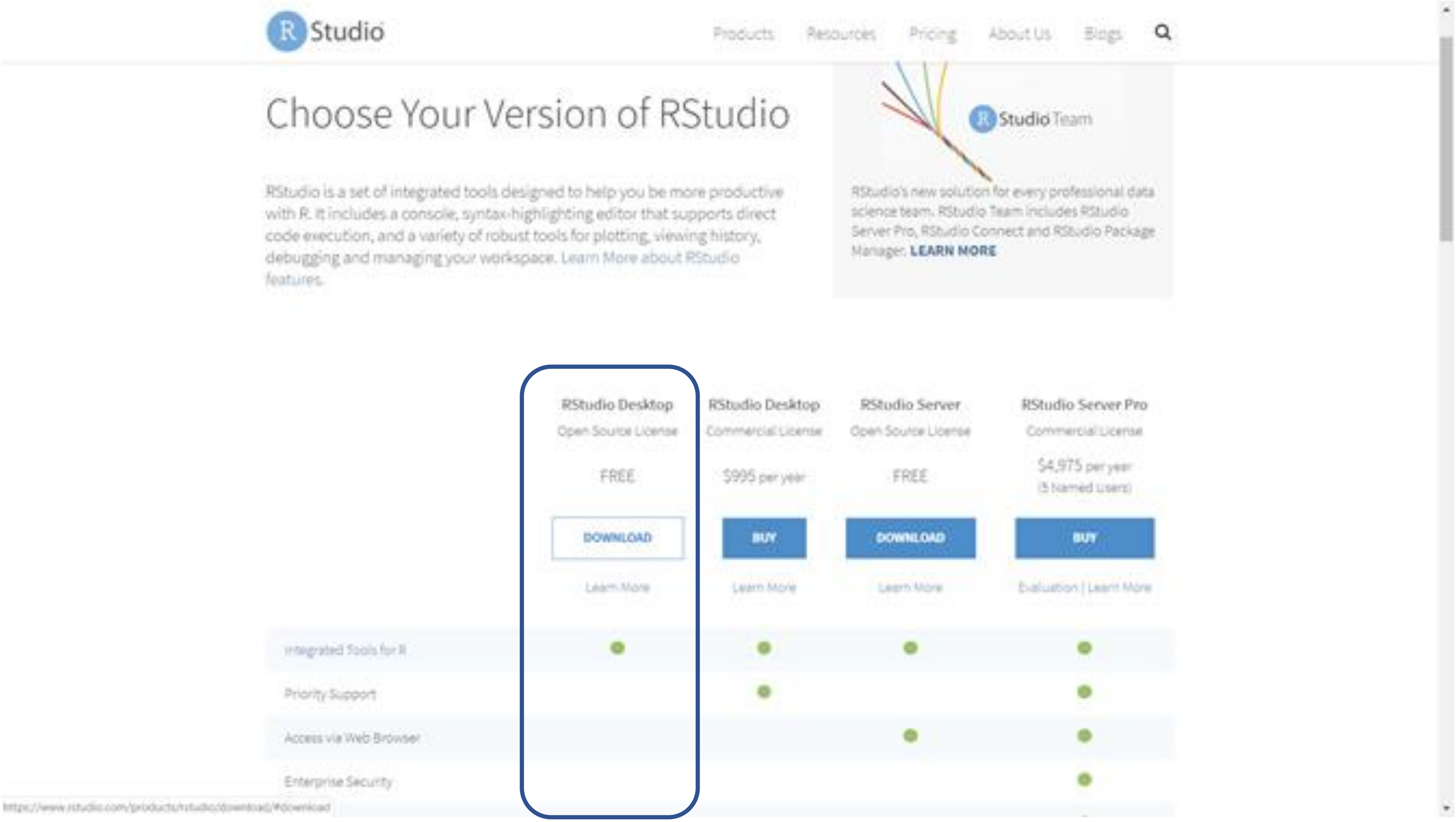
- Install it from Anaconda navigator

Running R

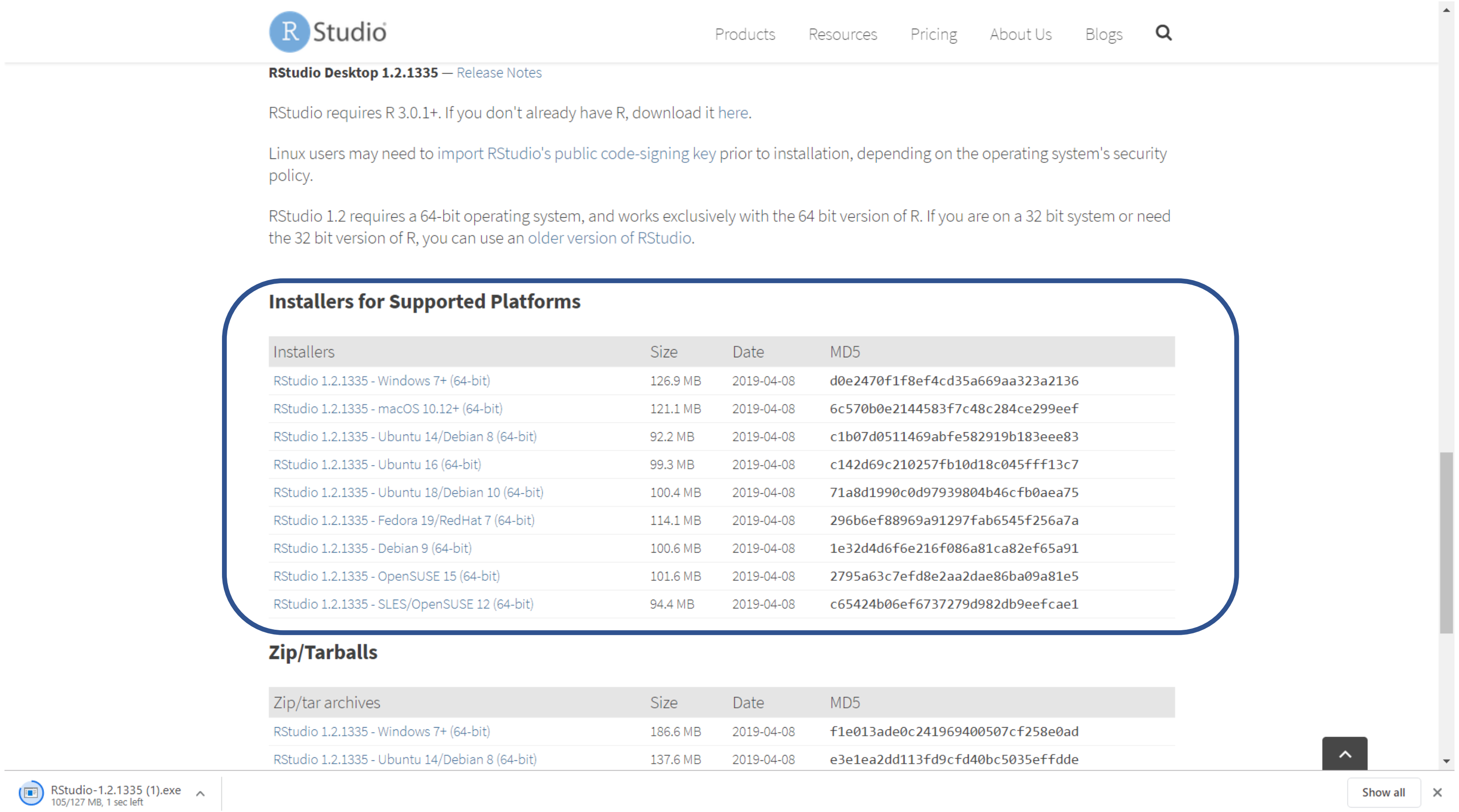
- Either type “R” in a shell (Linux/MacOS)
- Or start the R console or R-Studio application (Windows/MacOS)



Setting Up R Environment



Setting Up R Environment


R Studio

Products Resources Pricing About Us Blogs

RStudio Desktop 1.2.1335 — Release Notes

RStudio requires R 3.0.1+. If you don't already have R, download it here.

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

RStudio 1.2 requires a 64-bit operating system, and works exclusively with the 64 bit version of R. If you are on a 32 bit system or need the 32 bit version of R, you can use an older version of RStudio.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+ (64-bit)	126.9 MB	2019-04-08	d0e2470f1f8ef4cd35a669aa323a2136
RStudio 1.2.1335 - macOS 10.12+ (64-bit)	121.1 MB	2019-04-08	6c570b0e2144583f7c48c284ce299eef
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	92.2 MB	2019-04-08	c1b07d0511469abfe582919b183eee83
RStudio 1.2.1335 - Ubuntu 16 (64-bit)	99.3 MB	2019-04-08	c142d69c210257fb10d18c045fff13c7
RStudio 1.2.1335 - Ubuntu 18/Debian 10 (64-bit)	100.4 MB	2019-04-08	71a8d1990c0d97939804b46cfb0aea75
RStudio 1.2.1335 - Fedora 19/RedHat 7 (64-bit)	114.1 MB	2019-04-08	296b6ef88969a91297fab6545f256a7a
RStudio 1.2.1335 - Debian 9 (64-bit)	100.6 MB	2019-04-08	1e32d4d6f6e216f086a81ca82ef65a91
RStudio 1.2.1335 - OpenSUSE 15 (64-bit)	101.6 MB	2019-04-08	2795a63c7efd8e2aa2dae86ba09a81e5
RStudio 1.2.1335 - SLES/OpenSUSE 12 (64-bit)	94.4 MB	2019-04-08	c65424b06ef6737279d982db9eefcae1

Zip/Tarballs

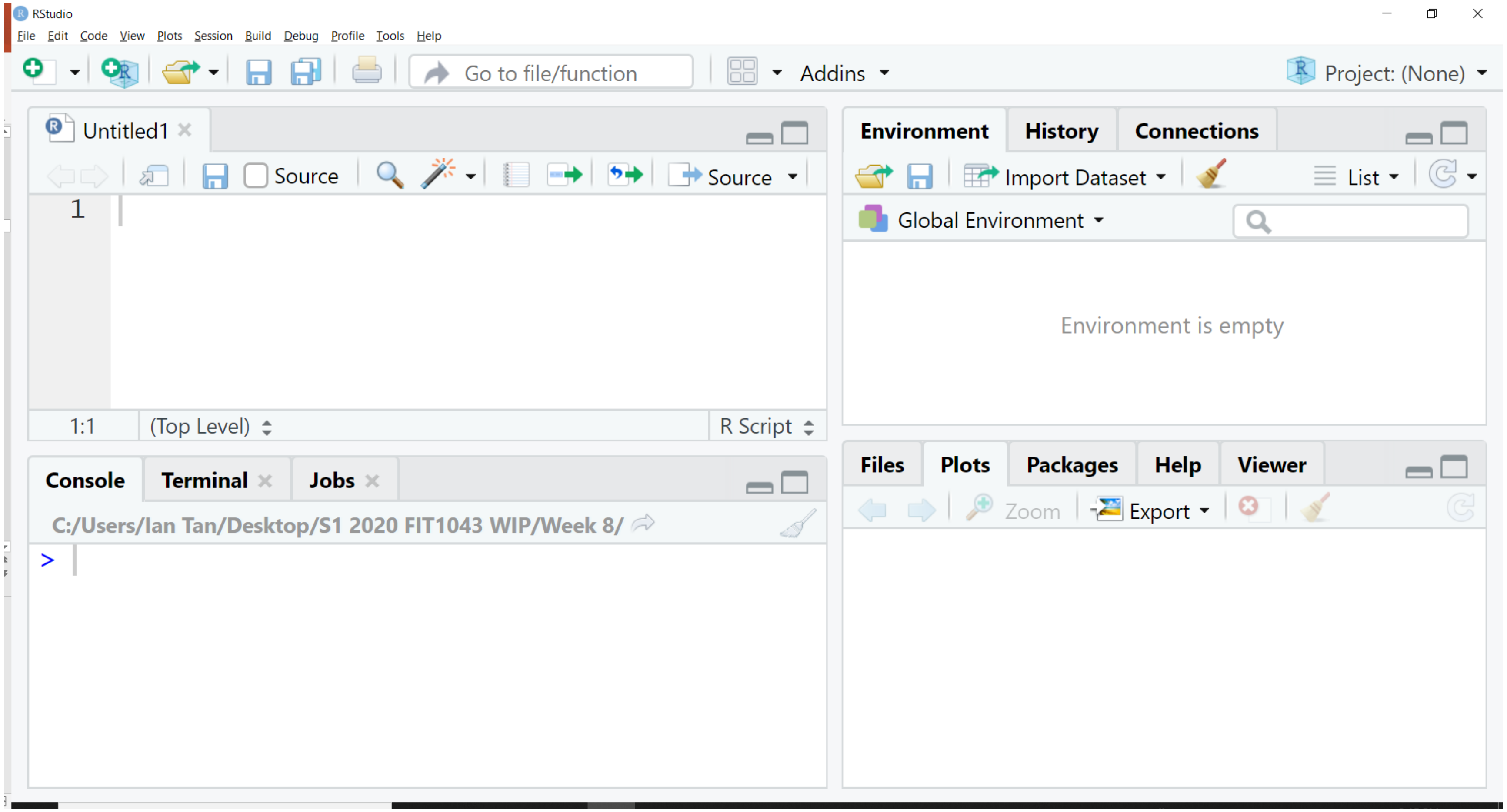
Zip/tar archives	Size	Date	MD5
RStudio 1.2.1335 - Windows 7+ (64-bit)	186.6 MB	2019-04-08	f1e013ade0c241969400507cf258e0ad
RStudio 1.2.1335 - Ubuntu 14/Debian 8 (64-bit)	137.6 MB	2019-04-08	e3e1ea2dd113fd9cfd40bc5035effdde

RStudio-1.2.1335 (1).exe
105/127 MB, 1 sec left

Show all

MONASH University

Setting Up R Environment



R Basics

Basic R Syntax

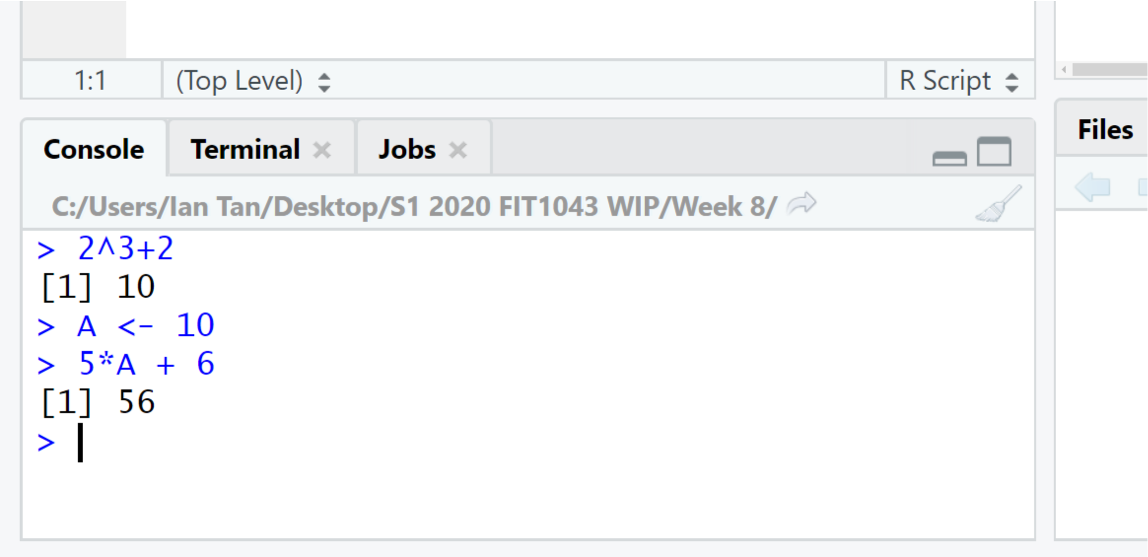
Here > denotes the command prompt & the output is prefixed by: [1]

- Compute mathematical expressions:

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

- 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
```

MONASH University

Basic Data Types

Numeric (can be integer or floating numder)

```
> x <- 10.5
```

Integer

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

Complex (with real and imagery part)

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

Logical (True / False)

```
> x <- TRUE
```

Character

```
> x <- "Intro To R"
```

Basic Data Types

- 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)
```

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 source: Quick_R

If-Else Conditions

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 Loops

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 Loops

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
```

R Data Wrangling and Exploration

MONASH University

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:**

MONASH University

Vector Arithmetic Operations

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
```

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 a Data Frame

R has name(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
```

Data Frame Audit

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   : 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
```

Summary Statistics

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              eights
 Bill :1   Min.    : 76.0   Min.    :1.490
 Henry:1   1st Qu.: 77.5   1st Qu.:1.535
 Joan :1   Median : 80.0   Median :1.560
 Ted  :1   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

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

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("Ages"),]
```

Sort by ages and heights

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

Sort by ages (ascending) and heights (descending)

```
> newData <- mtcars[order("Ages", "Heights", decreasing = TRUE), ]
```

** Order is only for Numeric value

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(dataframeA, dataframeB, by="ID")
```

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

Join two data frames

```
> total <- rbind(dataframeA, dataframeB)
```

R Working Environment

MONASH
University

Built-in Data Sets in R

There are several built-in data sets within the R environment

List available data set :

```
> data()
```

Load a built-in dataset

```
> data(mtcars)
```

Aggregating Data in Data Frame

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

```
> attach(mtcars)
> aggData <- aggregate(mtcars, by=list(cyl,vs), FUN=mean, na.rm=TRUE)
> print(aggdata)
> myData <- mtcars
> detach(mtcars)
```

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)
```

Inspect the data set

```
> head(myData, 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

Getting & Setting Working Directory

Before reading/writing in R it is important to specify the location where we can find the respective file to be 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(myData, "FileName.csv")
```

Read a csv file (file is in other location)

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

Reading a 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)

```
> newData = read.csv("FileName.csv")
> print(newData)
```

Read a csv file (file is in other location)

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

Loading Libraries

Libraries are lists of functions that are not available in R by default.

Loading a library

```
> library(moments)
> skewness(myData$mpg)
```

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")
```

R Data Visualisation

Bar Charts
Histograms
Box Plots
Scatter Plots

MONASH University

Bar Chart

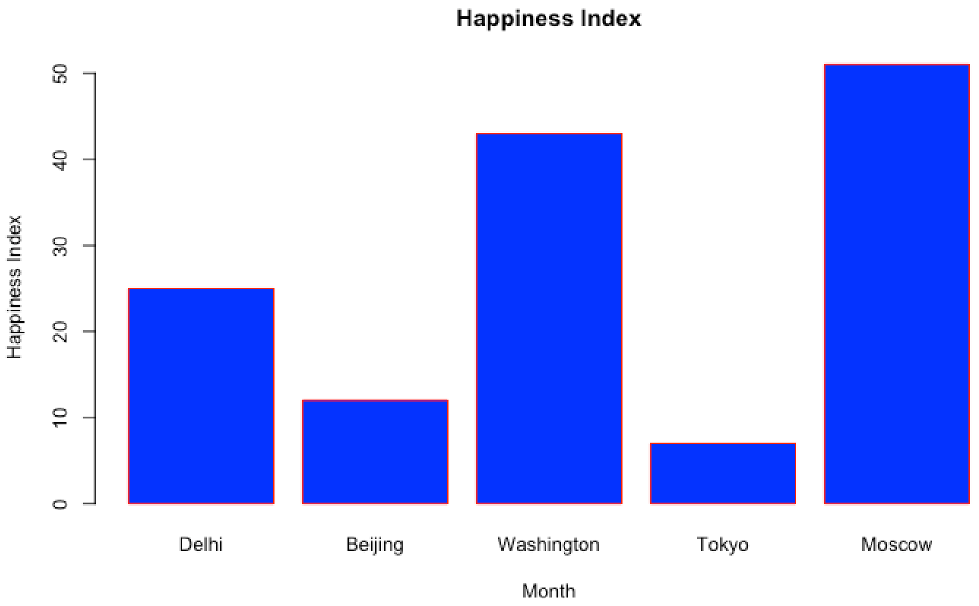
Compare the value for categorical data using bar chart

```
> 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")
```

- Syntax:
    - barplot(H,xlab,ylab,main, names.arg,col)

- Description of Parameters

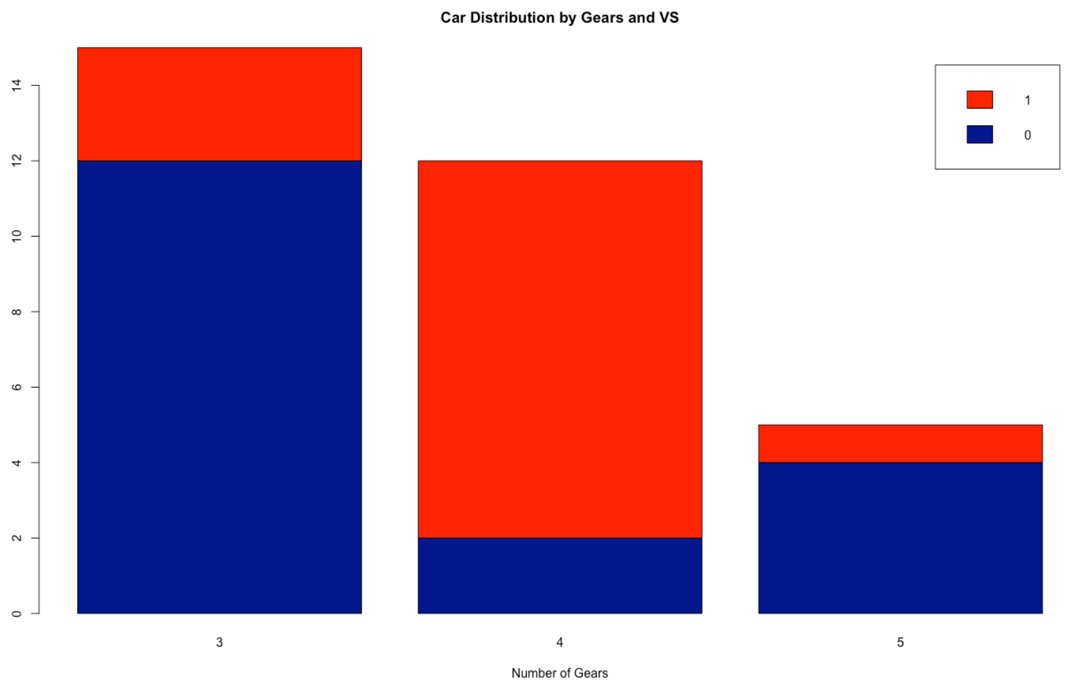
H is a vector or matrix containing numeric values used in bar chart.
xlab is the label for x axis.
ylab is the label for y axis.
main is the title of the bar chart.
names.arg is a vector of names appearing under each bar.
col is used to give colors to the bars in the graph.



MONASH University

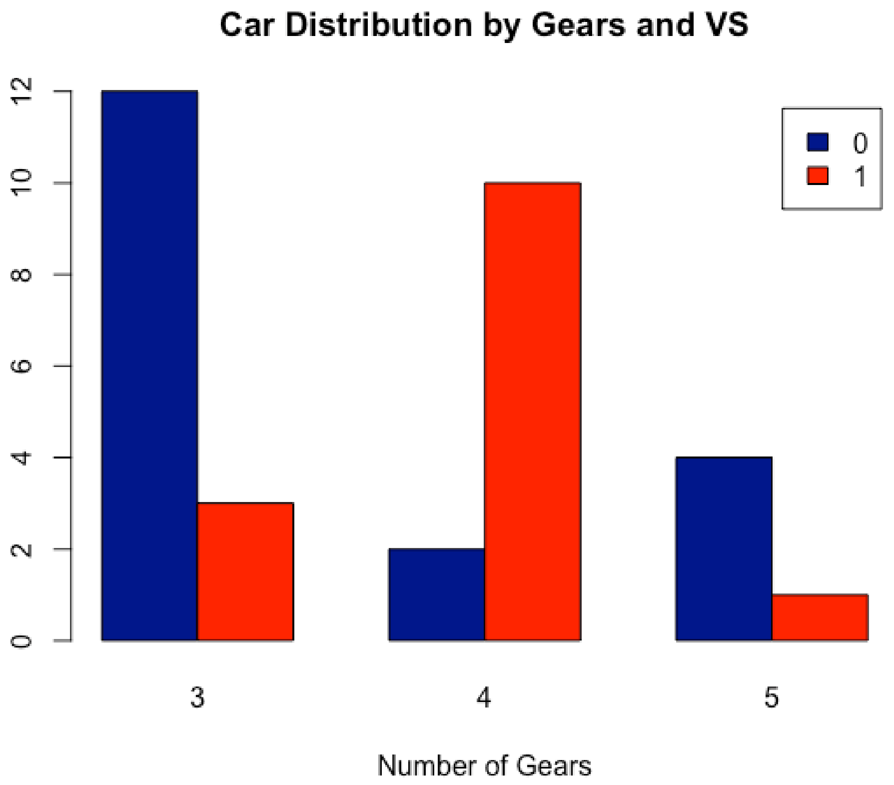
Stacked Bar Chart

```
> 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))
```



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)
```



Histograms

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

```
> hist(AirPassengers, main="Histogram for Air Passengers",
       xlab = "Passengers",border = "red", col = "blue",
       xlim = c(100,700), breaks =5)
```

- Syntax:
  - hist(v,main,xlab,xlim,ylim,breaks,col,border)
- Description of Parameters

v is a vector containing numeric values used in histogram.

main indicates title of the chart.

xlab is used to give description of x-axis.

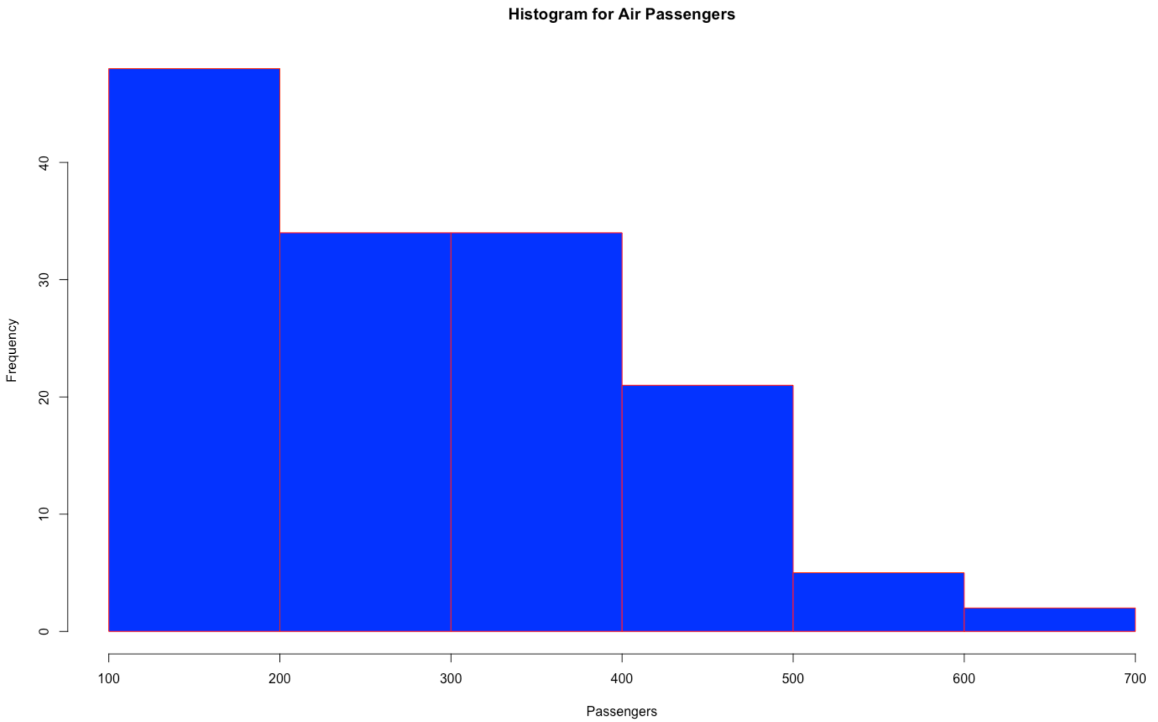
xlim is used to specify the range of values on the x-axis.

ylim is used to specify the range of values on the y-axis.

breaks is nothing but number of bins.

col is used to set color of the bars.

border is used to set border color of each bar.



Boxplots

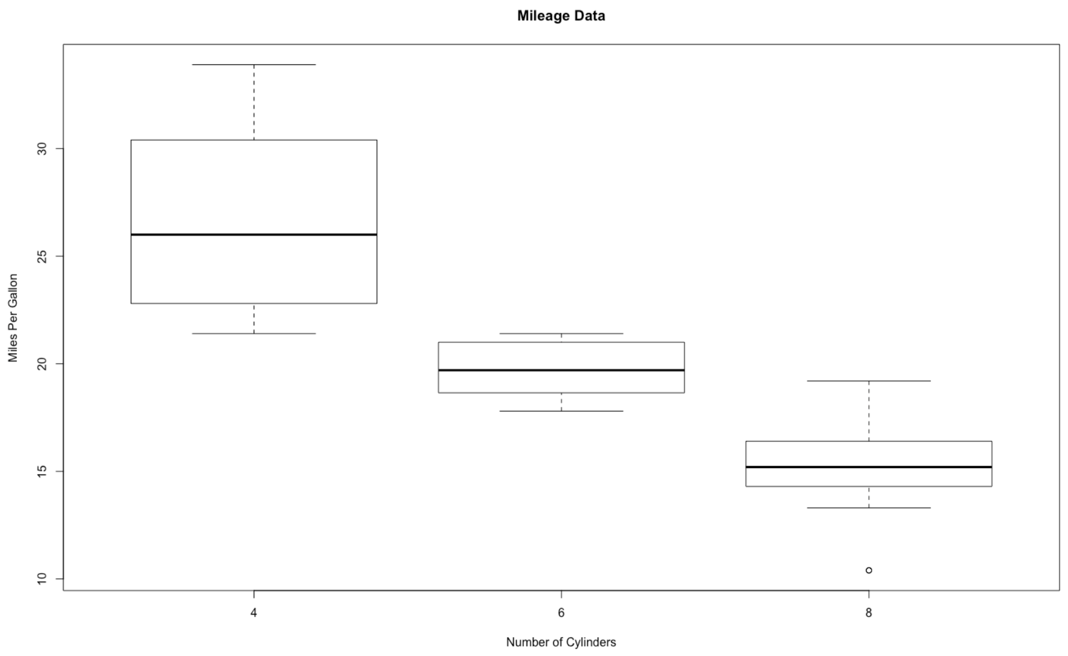
Or its summary statistics by plotting it as a boxplot

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

- Syntax:
  - boxplot(x,data,notch,varwidth,names,main)

- Description of Parameters

x is a vector or a formula.
data is the data frame.
notch is a logical value. Set as TRUE to draw a notch.
varwidth is a logical value. Set as true to draw width of the box proportionate to the sample size.
names are the group labels which will be printed under each boxplot.
main is used to give a title to the graph.



Boxplots

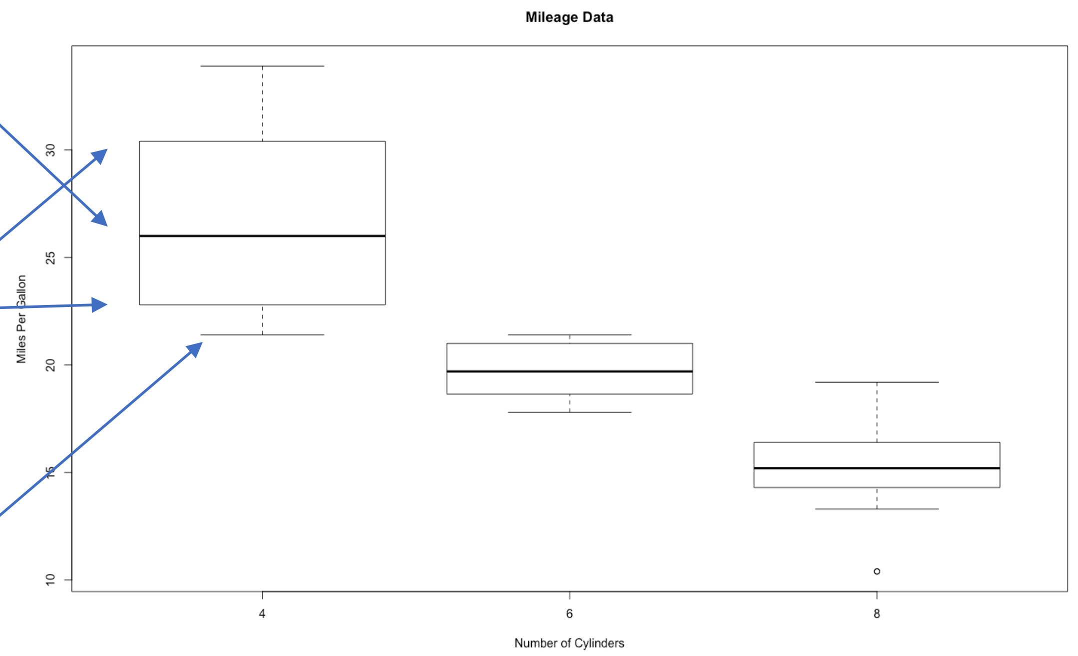
Or its summary statistics by plotting it as a boxplot

```
> 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)

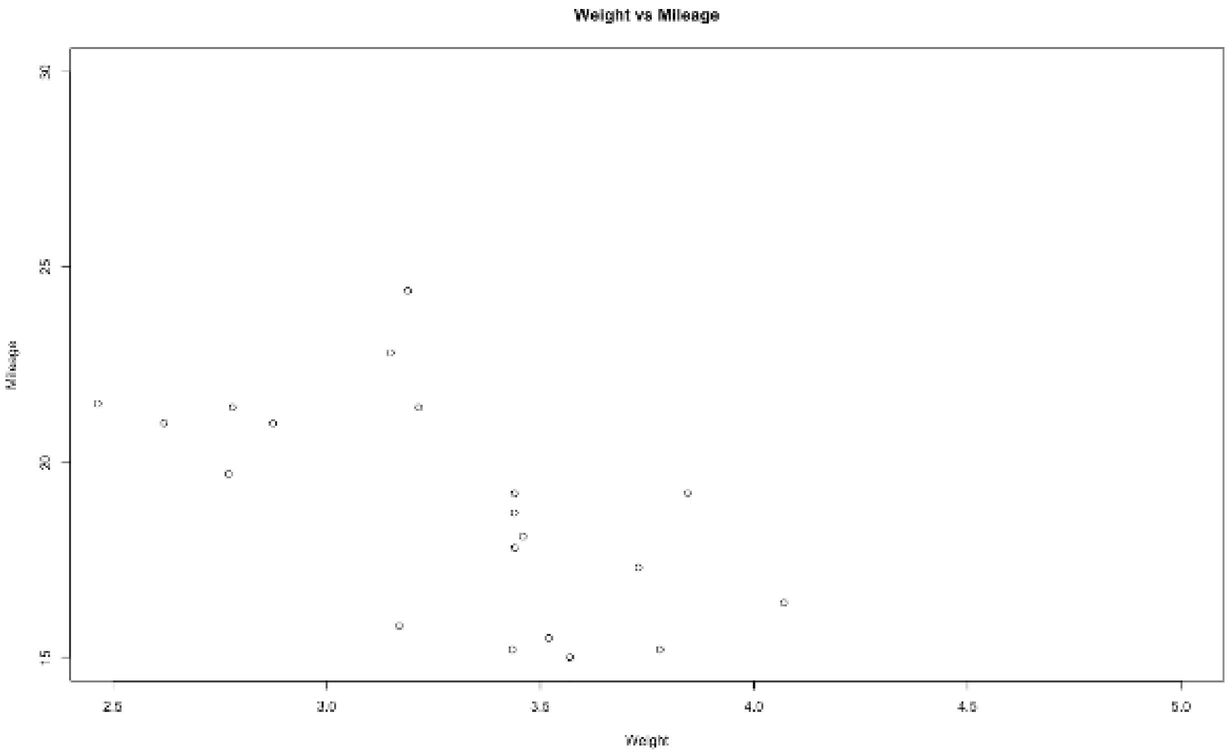
Minimum value (or 1.5*InterQuartileRange below lower quartile)



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 with R

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
```

MONASH University

Linear Regression

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

```
> summary(fit)
```

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
```

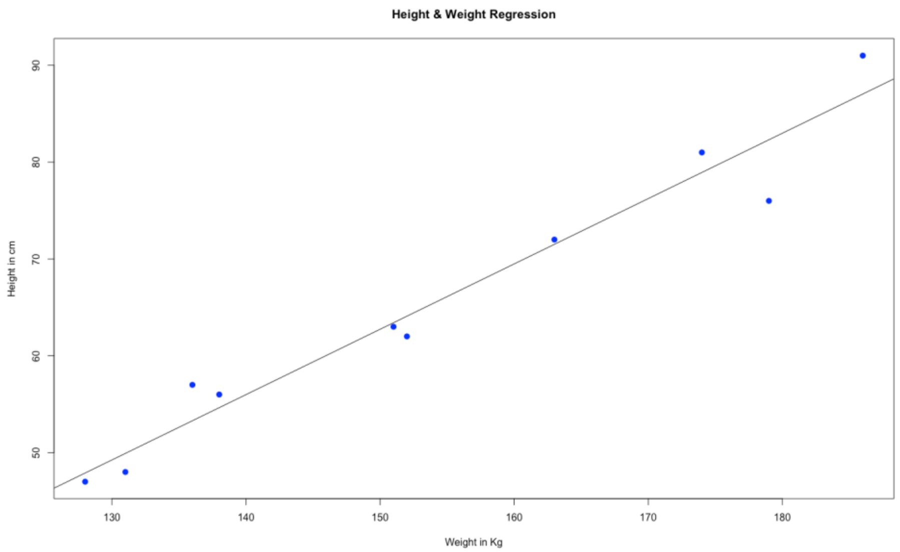
Linear Regression Visualisation

Give the chart file a name, this will plot it to a file instead if in RStudio

```
> png(file = "linearregression.png")
```

Plot the chart

```
> plot(height, weight, col = "blue", main = "Height & Weight
      Regression", abline(lm(weight ~ height)),
      cex = 0.8, pch= 16, xlab = "Weight in Kg",
      ylab = "Height in cm")
```



MONASH University

Notes

If you plot it to a file, the system will changed the output automatically.

```
> png(file = "linearregression.png")
```

To reset it back, use the command

```
> dev.off()
```

You can run it a few times until you get the

```
null device
```

MONASH University

Decision Trees with R

MONASH University

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
```

Visualise the Decision Tree

Give the chart file a name (try this after you have seen it on the RStudio bottom right pane)

```
> png(file = "decision_tree.png")
```

Create the tree.

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

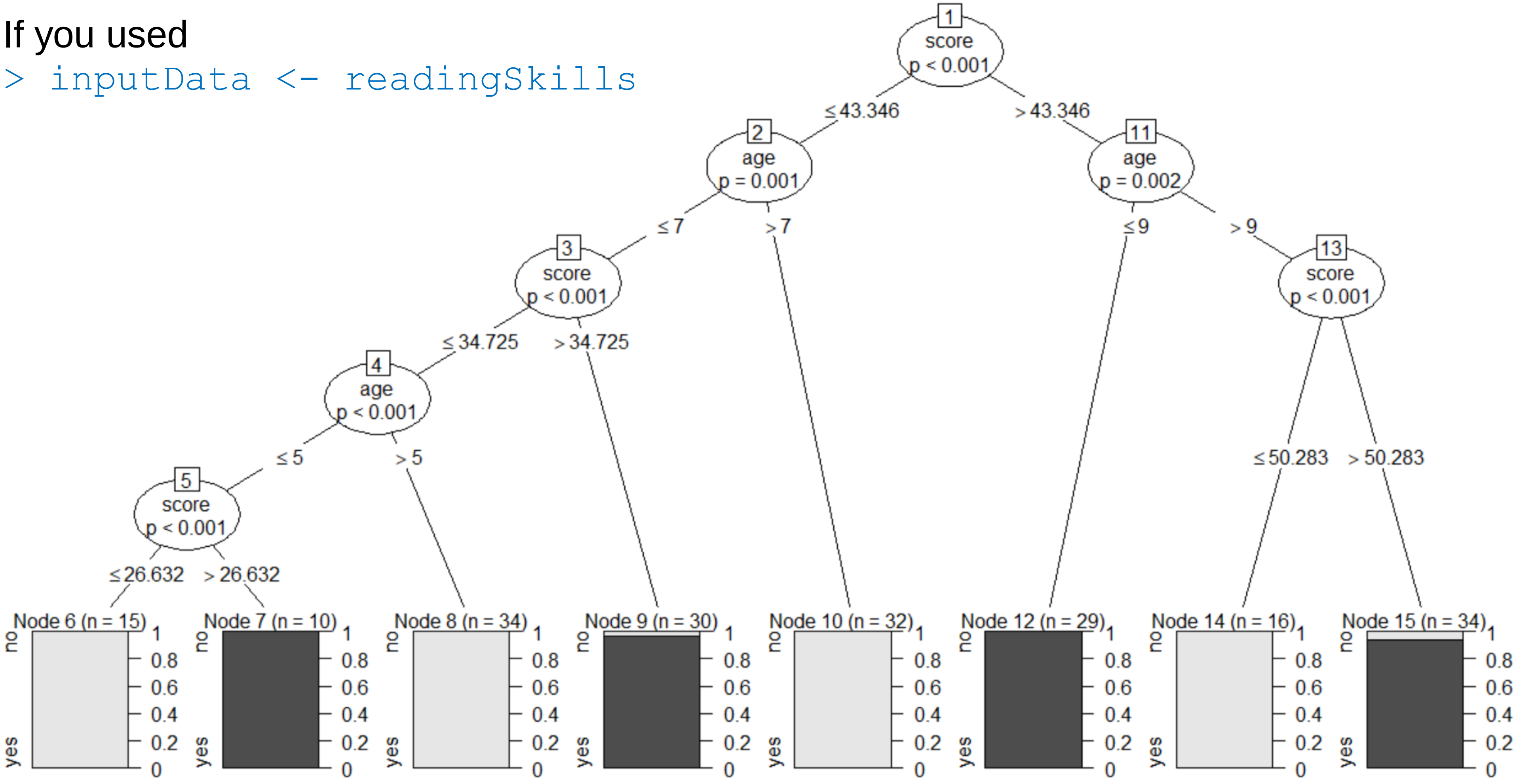
Plot the tree

```
> plot(outputTree)
```

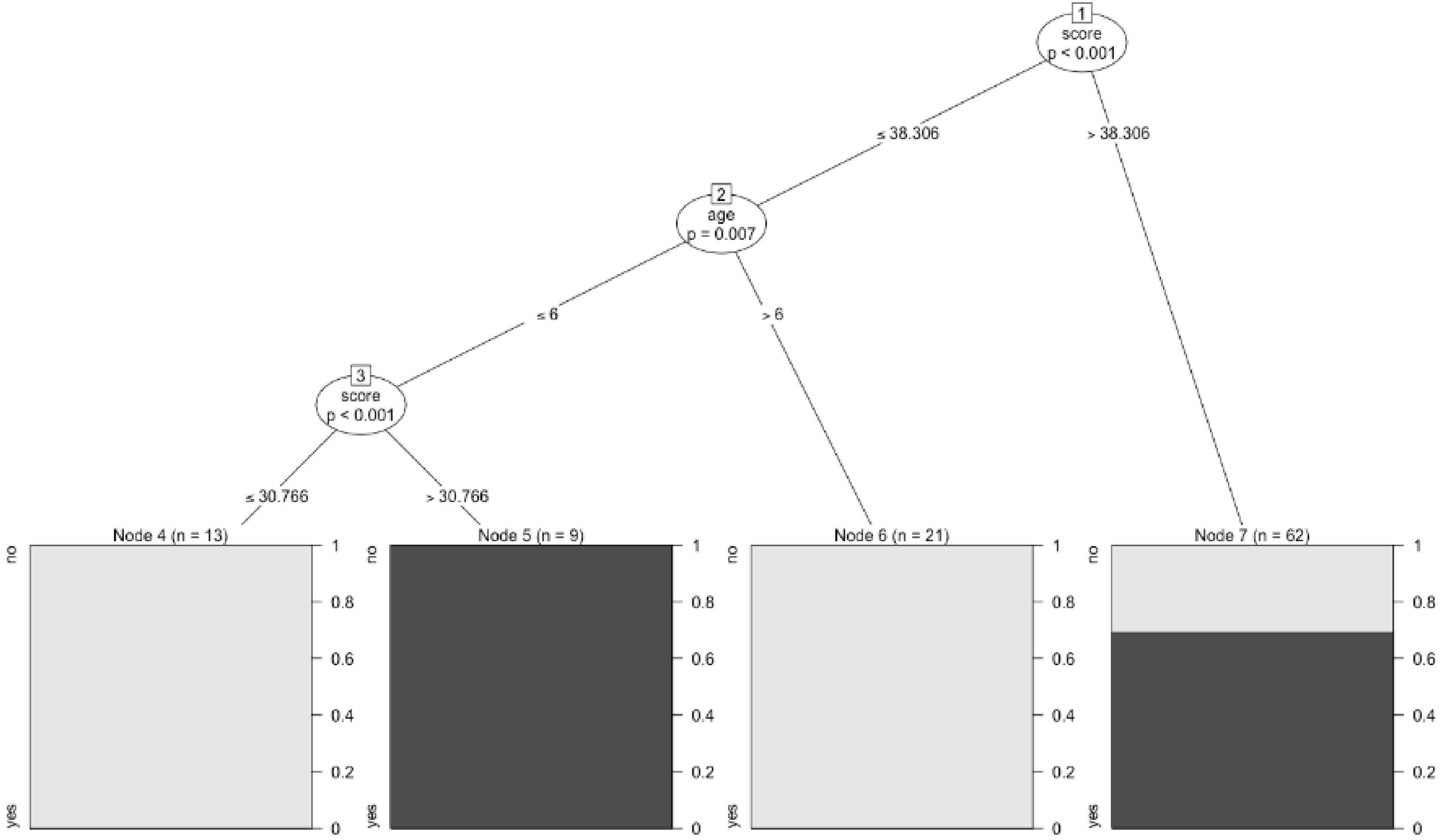
Visualise the Decision Tree
Visualisation

If you used

```
> inputData <- readingSkills
```



Visualise the Decision Tree



Recap: 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

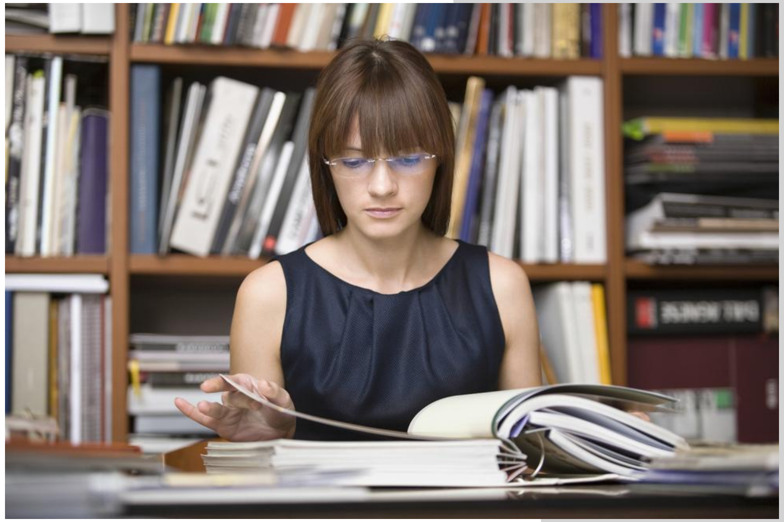
MONASH University

Home Activities

Suggested Activities for the week

Online Materials

Comprehensive courses on Datacamp to get your started, but for this Unit, just the Introduction to R would be sufficient.



Books (Articles)

Peng, Roger D. *R programming for data science*. Leanpub, 2016. (This is STILL free! ☺)

Zuur, Alain, Elena N. Ieno, and Erik Meesters. *A Beginner's Guide to R*. Springer Science & Business Media, 2009.

MONASH University

Tutorials Week 8

Installation of RStudio

R walk-through