

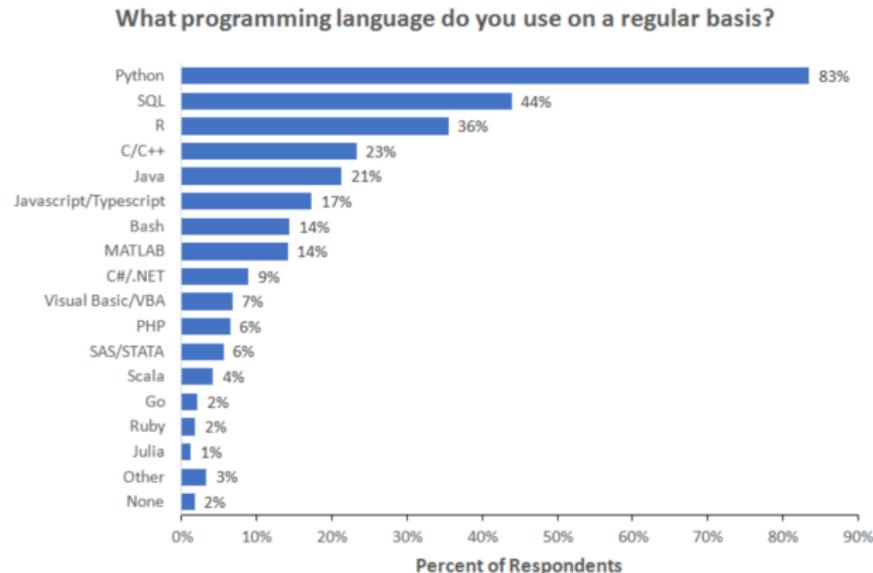
Unit 1: R Software

What is R?

- R is the most popular software used in statistical analysis.
- It is flexible and programmable.
- It provides the user with total control.
- It works from a command line.

R: Statistical programming language

- R is among the most popular languages for ‘data science’.



Note: Data are from the 2018 Kaggle Machine Learning and Data Science Survey. You can learn more about the study here: <http://www.kaggle.com/kaggle/kaggle-survey-2018>. A total of 18827 respondents answered the question.

R: Statistical programming language

• R is used by corporations:

Top Tier Companies using R

The following is a list of top brands or large organizations using R.

1. Facebook – For behavior analysis related to status updates and profile pictures.
2. Google – For advertising effectiveness and economic forecasting.
3. Twitter – For data visualization and semantic clustering
4. Microsoft – Acquired Revolution R company and use it for a variety of purposes.
5. Uber – For statistical analysis
6. Airbnb – Scale data science.
7. IBM – Joined R Consortium Group
8. ANZ – For credit risk modeling
9. HP
10. Ford
11. Novartis
12. Roche
13. New York Times – For data visualization
14. McKinsey
15. BCG
16. Bain

Financial Institutions

It includes major US and European Banks, Insurance Companies and Other financial institutions using R.

1. American Express
2. ANZ
3. Bank of America
4. Barclays Bank
5. Bazaj allianz Insurance
6. Bharti Axa insurance
7. Blackrock
8. Citibank
9. Dun & Bradstreet
10. Fidelity
11. HSBC
12. JP Morgan
13. KeyBank
14. Lloyds Banking
15. RBS
16. Standard Chartered
17. UBS
18. Wells Fargo
19. Goldman Sachs
20. Morgan Stanley
21. PNC Bank
22. Citizens Bank
23. Fifth Third Bank

Analytics and Consulting Companies using R

The below list comprises of niche analytics companies as well as consulting companies providing analytics or market research services.

1. A.T. Kearney
2. AbsolutData
3. AC Nielsen
4. Accenture
5. Bain & Company
6. Booz Allen Hamilton
7. Capgemini
8. Convergents
9. Deloitte Consulting
10. Evaluateserve
11. EXL
12. EY
13. Fractal Analytics
14. Gartner
15. Genpact
16. IBM
17. KPMG
18. Latent View
19. Manthan Systems
20. McKinsey & Company
21. Mu Sigma
22. PricewaterhouseCoopers
23. SIBIA Analytics
24. Simplify360
25. SmartCube
26. Target
27. The Boston Consulting Group
28. Tiger Analytics
29. Tower Watson
30. WNS
31. ZS Associate

<https://www.r-bloggers.com/companies-using-r/>

R: Statistical programming language

- R is used for journalism.

How to create BBC style graphics
Make a line chart
Make a multiple line chart
Make a bar chart
Make a stacked bar chart
Make a grouped bar chart
Make a dumbbell chart
Make a histogram
Make changes to the legend
Make changes to the axes
Add annotations
Work with small multiples
Do something else entirely

BBC Visual Journalism data team's cookbook for R graphics

Last updated: 2019-01-16

How to create BBC style graphics

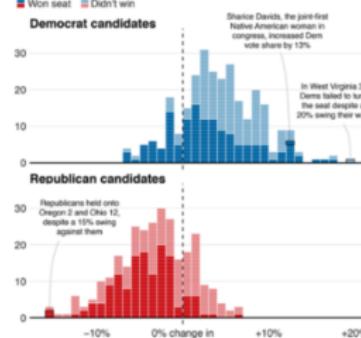
At the BBC data team, we have developed an R package and an R cookbook to make the process of creating publication-ready graphics in our in-house style using R's ggplot2 library a more reproducible process, as well as making it easier for people new to R to create graphics.

The cookbook below should hopefully help anyone who wants to make graphics like these:

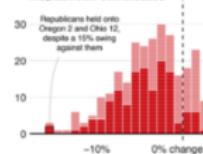
Blue wave

■ Won seat ■ Didn't win

Democrat candidates



Republican candidates

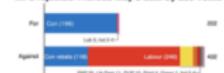


Where penalties are saved

World Cup shootout misses and saves, 1982-2014



MPs rejected Theresa May's deal by 230 votes



Earnings vary across virus even within subjects

Impact on men's earnings relative to the average degree



Source: AP, 19:01 ET

Source: Opta

We'll get to how you can put together the various elements of these graphics, but let's get the admin out of the way first...

Getting R and RStudio

- You can download your own copy from [R Project \(CRAN\)](http://www.r-project.org/) homepage at <http://www.r-project.org/>.
- The introductory tutorial for [R](http://cran.r-project.org/doc/manuals/R-intro.pdf) can be found at <http://cran.r-project.org/doc/manuals/R-intro.pdf>.
- Once [R](#) is installed, then [RStudio](https://posit.co/download/rstudio-desktop/) can be downloaded from <https://posit.co/download/rstudio-desktop/>.

Getting started with R

- We can work with R through its **console** as shown below:



```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"  
Copyright (C) 2020 The R Foundation for Statistical Computing  
Platform: x86_64-apple-darwin17.0 (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.
```

```
[R.app GUI 1.72 (7847) x86_64-apple-darwin17.0]
```

```
[History restored from /Users/katherinedavies/.Rapp.history]
```

```
>
```

- The other option, which is becoming increasingly popular, is through **RStudio**.

RStudio

- RStudio is an Integrated Development Environment which makes using R a bit smoother.
- RStudio can be thought of as the “dashboard” that most people use to “drive” R.

R: Engine



<http://moderndive.com/>

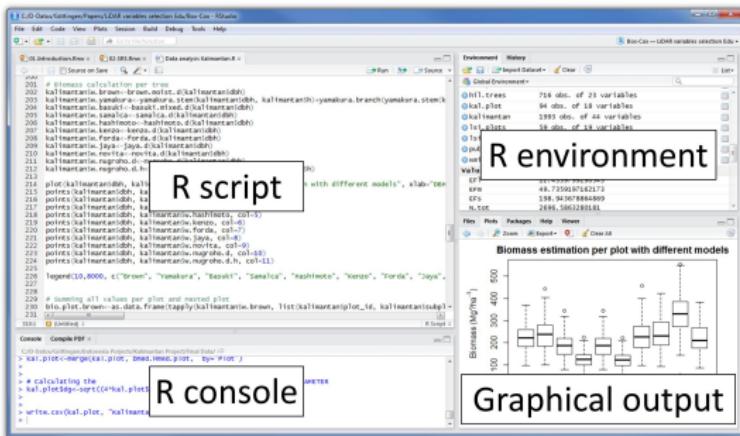
RStudio: Dashboard



RStudio

- In RStudio, there are 4 panels;

- Top left: R script
- Bottom left: R console
- Top right: R environment
- Bottom right: Graphical output



<https://www.analyticsvidhya.com/blog/2016/02/complete-tutorial-learn-data-science-scratch/>

R coding at a glance

- R code can be comprised of a variety of things, such as functions, calculations and plots.
- It is recommended to always create your code in an **R Script file** (an .R file), and in addition, to comment your code (explain what you are doing) using **#**; these comments are for you (and others to read) but **R** will ignore.
- In **R**, you would have the script file open and copy and paste into **R**; in **RStudio**, you would also open the script file (and it would appear in the top left panel) and highlight the lines of code you want to execute and click **Run**.

R syntax

- In both platforms, the **command line** is indicated by a `>`; this is where we tell **R** what to do (give it commands).
- On the command line, to redo a previous command, you can use the up arrow key (\blacktriangle or \uparrow).
- Here is a simple example of code on the command line:

```
>  
> simple.command <- "R welcomes 2150 students!"  
> print(simple.command)  
[1] "R welcomes 2150 students!"
```

where we first defined a **string** using `<-` – which means **assign** (note, an `=` can be used instead), and then we asked R to **print** the string.

- Here's another simple example where we define two variables and ask R to perform some simple mathematical calculations with them:

```
> x <- 8  
> y <- 2  
> x+y  
[1] 10  
> x-y  
[1] 6  
> x/y  
[1] 4  
> x*y  
[1] 16
```

- Basically R can be used as a calculator!

R objects and classes

- When something is created in R, it is an **object**, such as a **number**, **vector** or **matrix**.
- R **objects** are divided by **classes**; the main types of classes are **string**, **numeric**, **logical**, **matrix**, **data frame** and **lists**.
- We already saw what a string looked like; a **string** consists of *characters* and needs to have quotation marks around it (to tell R it is a string).
- numeric** is used for objects consisting of **numbers**, e.g. 1.26, 4 or 1e5.
- There are multiple classes within numeric itself, with a common one being **integer**.

R objects

- A **vector** in R (which can be in the numeric, character, or logical class) consists of a sequence of **components** which are of the same class, e.g. the numbers 2, 3 and 7; we use square brackets to define or access elements of vectors.
- A **matrix** in R is a generalization of a vector in that it contains many elements of the same type but they are arranged in **rows** and **columns**; a vector is a special case of a matrix in that it is a matrix with one row (or column). Like a vector, we use square brackets to define or access elements of matrices.

R objects

- A **data frame** is a two-dimensional array-like structure which is separated into columns each of equal length; each column is typically associated with a variable and the data on these variables can be of different forms (numeric, character, etc).
- **Lists** are R structures which contain objects of *different* types and even lengths; we use double square brackets to define and access elements of lists. Furthermore, lists can be nested, i.e., an element of a list can be a list itself.

R objects

- Oftentimes R objects are defined (and are empty) and then we fill them in (though this isn't always the case, for instance with imported data).

```
> a.vector <- double(5)      > a.matrix <- matrix(nrow=2, ncol=2)
> a.vector[1] <- 1          > a.matrix[1,1] <- 4
> a.vector[2] <- 1.4        > a.matrix[1,2] <- 6
> a.vector[3] <- 2          > a.matrix[2,1] <- 3
> a.vector[4] <- 4          > a.matrix[2,2] <- 4
> a.vector[5] <- 6.3        > a.matrix
                           [,1] [,2]
> a.vector
[1] 1.0 1.4 2.0 4.0 6.3  [1,]    4    6
                           [2,]    3    4
```

R objects

- Sometimes **loops** (to be discussed soon!) are used to fill in objects.
- Let us use a loop to fill in the elements of a vector:

```
> test.vector <- double(5)
> for(i in 1:5){test.vector[i] <- i^2}
> test.vector
[1] 1 4 9 16 25
```

and of a matrix:

```
> test.matrix <- matrix(nrow=2,ncol=2)
> for(i in 1:2){
+ for(j in 1:2){
+ test.matrix[i,j] <- i+j}}
> test.matrix
[,1] [,2]
[1,]    2    3
[2,]    3    4
```

R objects

- Here is a list composed of different types of objects:

```
> test.list <- list()
> test.list[[1]] <- a.vector
> test.list[[2]] <- simple.command
> test.list[[3]] <- matrix(1:4,nrow=2,byrow=T)
> test.list
[[1]]
[1] 1.0 1.4 2.0 4.0 6.3

[[2]]
[1] "R welcomes 2150 students!"

[[3]]
 [,1] [,2]
[1,]    1    2
[2,]    3    4
```

R objects

- The function `class` can be used to determine an object's class:

```
> test.matrix
     [,1] [,2]
[1,]    2    3
[2,]    3    4
> class(test.matrix)
[1] "matrix" "array"
> test.vector
[1]  1   4   9  16  25  36  49  64  81 100
> class(test.vector)
[1] "numeric"
> simple.command
[1] "R welcomes 2150 students!"
> class(simple.command)
[1] "character"
```

- Notice this though:

```
> test.vector2 <- seq(1:5)
> test.vector2
[1] 1 2 3 4 5
> class(test.vector2)
[1] "integer"
```

Built-in functions

- In addition to basic mathematical calculations, you have already seen some built-in commands/functions in R, such as `print` and `class`; in fact, R has many built-in functions to carry out simple tasks.
- Here are some commonly other used functions that you should familiarize yourself with:

<code>length</code>	<code>dim</code>	<code>which</code>
<code>t</code>	<code>rep</code>	<code>abs</code>
<code>cbind</code>	<code>rbind</code>	<code>rm</code>
<code>sqrt</code>	<code>log</code>	<code>exp</code>
<code>sum</code>	<code>mean</code>	<code>median</code>
<code>min</code>	<code>max</code>	<code>seq</code>
<code>sort</code>	<code>order</code>	<code>round</code>

- To see how they work, type a question mark and the name of the function, i.e., `?round`.

Some simple commands

```
> data <- c(1,4,0,2,8)
> mean(data)
[1] 3
> min(data)
[1] 0
> max(data)
[1] 8
> data[1]+data[4]
[1] 3
> sort(data)
[1] 0 1 2 4 8
> order(data)
[1] 3 1 4 2 5
> data^2
[1] 1 16 0 4 64
`
```

*Note the difference between `sort` and `order`!

Libraries and packages

- A **package** in R is just a collection of functions, usually with some connecting theme.
- Packages are stored in **libraries**; there are packages for almost everything:
 - `numDeriv` - Methods for calculating (usually) accurate numerical first and second order derivatives
 - `mgcv` - Generalized Additive Models
 - `survival` - Tools for survival analysis
 - `extraDist` - Provides functions dealing with special probability distributions
- R comes with a standard set of packages. Others packages need to be downloaded (installed); this is done in RStudio in the bottom right on the **Packages** tab or using `install.packages("packagename")`. Packages are then loaded using the command `library`, i.e., `library(numDeriv)`.

Reading data

- One of the main uses of R is **data analysis**; **exploratory data analysis** will be the focus of Unit 2, but first we need to learn how to get the data into R.
- Data can come from a variety of places; let's start with some data already in **R**, called **built-in** data sets.
- To see the list of these built-in data sets, simply type **data()**.
- To get a summary of a particular data set, for instance, **mtcars**, simply type **summary(mtcars)**.
- These data sets are data frames; you can access, manipulate and explore all the components of the data set; we will look at this more in Unit 2.

Importing data

- Data can also be **imported** from the internet, your computer (or some other local directory) into **R** .
- Let us start with importing a data set, in the form of an excel file, from a directory.
- Suppose you have an Excel file. To import it in **R** , you can use the drop down menus: **File - Import Dataset - From Excel**.
- Note that **R** may need to install or update packages to do this.

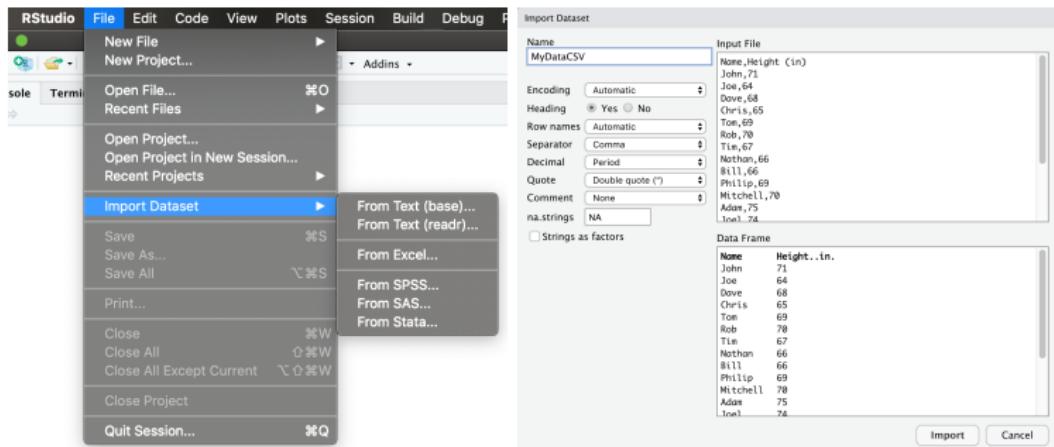
Importing data

- Now suppose you have what is called a **csv** file, i.e., a comma separated file.
- The procedure for importing this is similar to an excel file.
- Here is a fake data set viewed inTextEdit and beside it, in Excel:

Name, Height (in)	1	Name	Height (in)
John, 71	2	John	71
Joe, 64	3	Joe	64
Dave, 68	4	Dave	68
Chris, 65	5	Chris	65
Tom, 69	6	Tom	69
Rob, 70	7	Rob	70
Tim, 67	8	Tim	67
Nathan, 66	9	Nathan	66
Bill, 66	10	Bill	66
Philip, 69	11	Philip	69
Mitchell, 70	12	Mitchell	70
Adam, 75	13	Adam	75
Joel, 74	14	Joel	74
Matthew, 67	15	Matthew	67
Cole, 66	16	Cole	66
Henry, 68	17	Henry	68
Derek, 70	18	Derek	70
Riley, 72	19	Riley	72
Steve, 69	20	Steve	69

Importing data

- We can import using the dropdown menu (in RStudio) by selecting **File - Import Dataset - From Text (base)**, for instance, and then a box will show you how it is being imported and you click **Import**.



- Or we can import from the command line:
> `mydata <- read.csv(file="~/Desktop/MyDataCSV.csv", header=TRUE)`
- Note "header=TRUE" tells R that the top row has column names.

Importing data

- Suppose a website contains a data set you would like to import into R.
- The simplest method is to use the function `read.table`:

```
> web.data <- read.table(url("http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat"))
```

- Alternatively, the function `fread` from the package `data.table` can be used:

```
> library(data.table)
> web.data<- fread('http://www.stats.ox.ac.uk/pub/datasets/csb/ch11b.dat')
```

- This information is from :

[https://www.r-bloggers.com/
getting-data-from-an-online-source/.](https://www.r-bloggers.com/getting-data-from-an-online-source/)

Entering data

- We can enter data directly into R :

```
> Height <- c(71,64,68,65,69,70,67,66,66,69,70,75,74,67,66,68,70,72,69)
> Name <- c('John', 'Joe', 'Dave', 'Chris','Tom','Rob','Tim','Nathan','Bill','Phillip','Mitchell','Adam','Joel','Matthew','Cole','Henry','Derek','Riley','Steve')
> mydataframe <- data.frame(Name,Height)
> summary(mydataframe)
   Name           Height
Length:19      Min.   :64.00
Class :character 1st Qu.:66.50
Mode  :character Median :69.00
                  Mean   :68.74
                  3rd Qu.:70.00
                  Max.   :75.00
```

Augmenting a data frame

- We can also add to this data frame, say Weight, in two ways:

```
> Weight <- c(180,150,165,175,180,165,200,195,180,163,178,169,184,170,162,150,181,190,176)
> MyNewDataframe <- data.frame(Name,Weight,Height)
> summary(MyNewDataframe)
```

Name	Weight	Height
Length:19	Min. :150.0	Min. :64.00
Class :character	1st Qu.:165.0	1st Qu.:66.50
Mode :character	Median :176.0	Median :69.00
	Mean :174.4	Mean :68.74
	3rd Qu.:180.5	3rd Qu.:70.00
	Max. :200.0	Max. :75.00

```
> MyNewDataframe <- data.frame(mydataframe,Weight)
> summary(MyNewDataframe)
```

Name	Height	Weight
Length:19	Min. :64.00	Min. :150.0
Class :character	1st Qu.:66.50	1st Qu.:165.0
Mode :character	Median :69.00	Median :176.0
	Mean :68.74	Mean :174.4
	3rd Qu.:70.00	3rd Qu.:180.5
	Max. :75.00	Max. :200.0

- When all the elements of a data set are numbers, matrices are commonly used to organize and store it.
- We will get more practice with data sets in Unit 2.

Exporting data and saving images

- Depending on the format you'd like to export the data as, there are various ways to do this.
- The function `write.table` takes a matrix or data frame and exports it as a .txt to the working directory. The function `write.csv` exports as a .csv. Examples:

```
write.table(mydata, "testexport.txt")
write.csv(mydata, "testexport.csv")
```

- There is also the function `write.xlsx` as part of the package `xlsx` in order to export as a spreadsheet.
- To save an image in R, there are various functions such as `jpeg` and `pdf()`; when using such functions, make sure to always type `dev.off()` after you create the plot, to close the device.
- In RStudio, simply go to the output panel and click **Export**.

Writing a function

- An **R function** is an object containing instructions for particular purpose and can be reused; it typically has **4** main pieces:
 - (1) Name
 - (2) Arguments: inputs to the function
 - (3) Body: does the work of the function
 - (4) Output: what is returned once the function is executed
- Arguments of functions can be assigned **default** values. If you do not specify a value for an argument, it will use the default values.

Writing and calling a function

- Let's make a function to find the **mean** of a list of numbers:

```
Name          Arguments  
↓             ↓  
> my.mean = function(x){  
+   mysum = sum(x)  
+   n = length(x) }  
+   return(mysum/n) ← Output  
+ }  
> my.mean(c(1,2,3,4,5))  
[1] 3
```

The diagram highlights parts of the R code:

- Name**: Points to the variable name `my.mean`.
- Arguments**: Points to the argument `x` in the function definition.
- Body**: Brackets the assignment statement `mysum = sum(x)` and the calculation `n = length(x)`.
- Output**: An arrow points from the `return` statement to the final output value `3`.

More on functions

- We can write a function which calls another function:

```
> diffs <- function(x=c(1,2)){  
+ return(x[2]-x[1])  
+ }  
> new.fun <- function(x=c(1,2)){  
+ temp <- diffs(x)  
+ return(exp(temp))  
+ }  
> new.fun(c(6,3))  
[1] 0.04978707
```

- The function `new.fun` takes a vector of length 2 (see the default), applies the function `diffs` then applies the built-in function `exp`.

Logical operators

- The standard logical operators in R are `&`, `|` and `!`, which correspond to `and`, `or` and `not`, respectively.
- In addition, and in conjunction with these operators, the following are commonly used:

<code><</code> (less than)	<code><=</code> (less than or equal to)
<code>></code> (greater than)	<code>>=</code> (greater than or equal to)
<code>==</code> (equal to)	<code>!=</code> (not equal to)

Conditionals

- Conditionals are expressions which tell R to compute something depending on whether a **condition** is satisfied or not.
- The commonly used conditionals are **if** and **ifelse**.
- We can use conditionals on the command line directly:

```
> ifelse(1<2,3,2)
[1] 3
> if(1<2){w<-3}else{w<-2}
> w
[1] 3
```

or inside functions:

```
> cond.function <- function(x){
+ if(x>0){print("Non-negative number")}else{print("Negative number")}
+ }
> cond.function(-2)
[1] "Negative number"
> cond.function(2)
[1] "Non-negative number"
```

Loops

- Loops are way to **automate** repetition of steps/calculations.
- There are primarily two types of loops: **for** and **while**.
- In a **for** loop, a sequence is supplied and the statement/function is evaluated/executed on each item in the sequence; the loop is exited once the last item is reached.
- In a **while** loop, the loop is only entered if some **condition** is satisfied, and if so, the statement/function is evaluated/executed; once executed, the condition needs to be **re-evaluated** and if not satisfied, the loop terminates and exits (otherwise, it repeats).

Examples of loops

- A **for** loop:

```
> test.loop <- double(10)
> for(i in 1:10){
+ test.loop[i] <- i^2}
> test.loop
[1] 1 4 9 16 25 36 49 64 81 100
```

- A **while** loop:

```
> i<-1
> while(i<10){
+ print(i)
+ i <- i+1}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

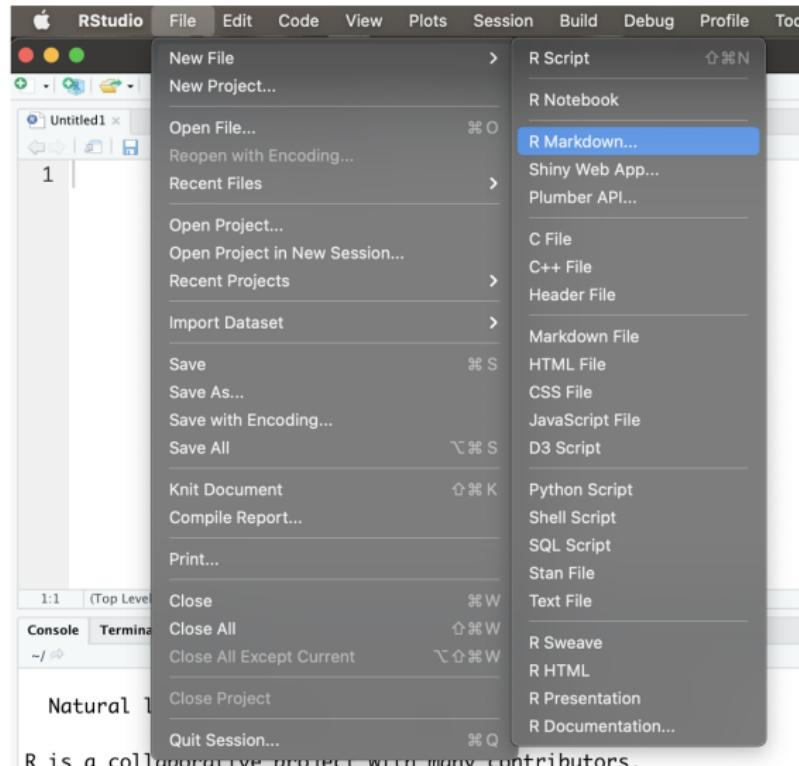
A word of warning

- When you work in R (the console directly or using RStudio), you are working in a workspace.
- When you save your workspace, an .RData file is created in your working directory and all the objects (variables, functions, etc) will be stored in that file.
- When you re-open R (or RStudio), and you are using the same working directory, it will open whatever workspace was used last; so if the last workspace was saved, it will open that workspace.
- It is therefore highly recommended to not save your workspace when closing R (or RStudio)!

- RMarkdown is a relatively new tool for [R](#) users; it was introduced in 2004 but it has been within the last few years that it has really become popular.
- RMarkdown is only available when using [RStudio](#).
- With a single RMarkdown file (which are saved as .Rmd) you can save and execute code as well as generate reports in various formats; common formats for the output are html and pdf.
- To use RMarkdown, you may need to install packages, such as: [knitr](#) and [rmarkdown](#).

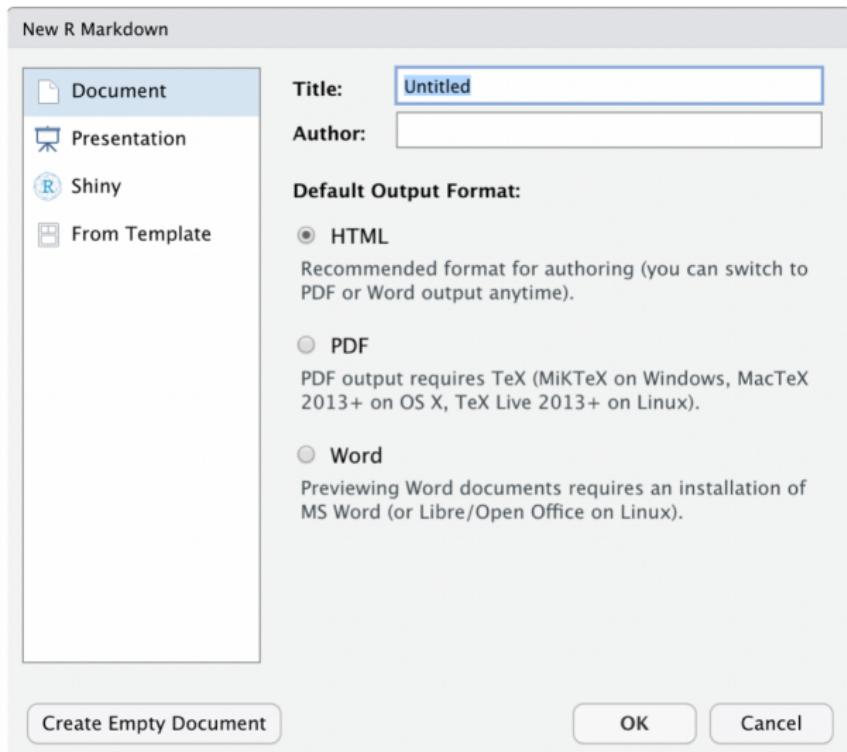
Creating an RMarkdown (.Rmd) file

Step 1: Creating an .Rmd file



Creating an RMarkdown file

Step 2: Choosing format and name of output



In this course, our output format will be PDF.

RMarkdown syntax

- Under the automated information about the file, you can simply start typing text.
- Headers can be inserted by typing `#`, i.e., `# Section 1`.
- There are ways to highlight or emphasize text:
 - Italics: `*italic*`
 - Bold: `**bold**`
- We can also make lists:

- Unordered:

Unordered list

- * Item 1
- * Item 2
 - + sub-item 2a
 - + sub-item 2b
- * Item 3

- Ordered:

Ordered list

1. Item 1
2. Item 2
3. Item 3
 - + sub-item 3a
 - + sub-item 3b

Code chunks in RMarkdown

- Executable **R** code can be embedded into the .Rmd file.
- To insert **R** code (often referred to as **code chunks**), you must type three back tick marks, r, the curly brackets, the code and then three more tick marks:

```
```{r echo=TRUE}
mean(c(2,5,8))
```
```

- Note that **echo=TRUE** means we want our code displayed in the output file.

Plots and figures in RMarkdown

- We can name a code chunk and we can have R generate **figures** for display (or not) in the output file.

```
```{r Inserting a plot, echo=FALSE}
mydata <- read.csv(file "~/Desktop/MyDataCSV.csv", header=TRUE)
hist(mydata$Height,xlab="Height")
```

- If there is a local image (for which you can determine the path) to be inserted, simply type:

![Caption](filepath)

- Notice the difference between the two ways to produce figures in your output file: one is R code (a code “chunk”) and one is pure RMarkdown syntax.

# Tables in RMarkdown

- **Tables** can be made in various ways, many of which rely on **R** functions from various **R** packages such as:
  - knitr
  - xtable
  - stargazer
  - tables
- Tables can also be created manually.
- Some tables look better in certain formats (pdf versus html, for instance).

# Tables in RMarkdown

- Making a simple table from raw data using RMarkdown syntax:

Name	Faculty	Years in Position
Chris Sanders	Medicine	5
Suzanne Clay	Arts	10
Joe Smith	Science	8
Ross Watson	Engineering	18

- The output pdf file displays the table as:

Name	Faculty	Years in Position
Chris Sanders	Medicine	5
Suzanne Clay	Arts	10
Joe Smith	Science	8
Ross Watson	Engineering	18

# Tables in RMarkdown

- Loading `knitr` and using its function `kable`:

Here is a sample table

```
```{r loading mtcars data set and making a table, results="asis"}  
library(knitr)  
kable(mtcars[1:4,], caption="A simple table")  
```
```

- And here's how it looks in a pdf file:

Table 2: A simple table

|                | 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    |

# How to knit

- To produce the output file (pdf, or whatever you chose), you need to [knit](#); do not pick up some yarn and knitting needles!



- In [RStudio](#) simply click the [Knit](#) button at the top; if the output file is a pdf, it will open in the appropriate software; if the output file is html, the file typically is opened immediately for viewing.
- Assuming the necessary figures and other files are available, anyone who has the .Rmd file can [reproduce](#) the same output using RMarkdown.

# Other R and RMarkdown resources

- Besides using help files in R, the internet is full of suggestions and help using R.
- A quick **internet search** of any issue you're having is likely to find a solution (though it may not be a good one).
- The official R Short Reference Card contains some basic useful functions.
- There are also many online resources for RMarkdown, including videos, cheat sheets and examples.