The Seychelles-Oxford Partnership Statistics Handbook

The Republic of Seychelles Ministry of Health and University of Oxford

2023-02-07

Contents

Welcome **5**

1 Introduction **7**

2 Installing R and RStudio **9**
2.1 Install R 9
2.2 Install RStudio 10
2.3 Install Rtools 10
2.4 Install Git for Windows 10

3 Introduction to R and RStudio **11**
3.1 What is R? 11
3.2 Why use R? 12
3.3 What is RStudio 13

4 Basics of R **15**
4.1 Summary 51

5 Objects in R **53**
5.1 Summary 73

6 Extending R with packages **75**
6.1 Summary 83

7 Writing functions **85**

8 Introduction to git and GitHub **97**
8.1 All about git 97

8.2 All about GitHub 97

9 Cloning a GitHub repository 103

9.1 Copy the repository URL of the repository you want to clone from GitHub 104
9.2 Go to RStudio and create new project 105
9.3 Choose Version Control 106
9.4 Select Git 107
9.5 Setup repository settings 108

10 Commit and push to GitHub 111

10.1 Click on `Commit` in the Git tab on RStudio 112
10.2 Getting changes saved and push to GitHub 113
10.3 Initiate a pull request 114

11 Participating in an R/RStudio project 119

11.1 Clone the project to your local machine 119
11.2 Create a new branch from the main branch 120
11.3 Code and make changes to your branch 122
11.4 Commit and push your changes and initiate a pull request . . 122
11.5 Merge pull request 123

12 Initiating an R/RStudio project 125

12.1 Create a new project in RStudio 126
12.2 Structure/organise your new project appropriately 132
12.3 Start coding 133
12.4 Next steps 133

13 Scientific workflows 135

13.1 Step 1: Create a new RStudio project 138
13.2 Step 2: Create an R file called `packages.R` 138
13.3 Step 3: Create placeholder directories for different components of workflow 140
13.4 Step 4: Create the target script file (`_targets.R`) 140
13.5 Step 5: Edit the `_targets.R` script file 141

Welcome

This is the handbook for the **Seychelles-Oxford Partnership** project on statistical training to strengthen evidence-based research capacity in the Seychelles.

FINIS CORONAT OPVS

Chapter 1

Introduction

Findings generated by international research can be widely applicable and relevant to informing and influencing national health policies and decisions (e.g. policies and guidelines relating to COVID-19). However, the most effective and meaningful policies are invariably borne from locally-driven research addressing contextualised health and health system problems.

The Seychelles is one of the few Sub-Saharan African countries with efficient, accurate and comprehensive data collection and disaggregation, across a variety of health metrics within the public health sector. However, critical skills and health research capacity to analyse this data and build a local evidence-base with which to inform policy and improve population health, are severely lacking. This shortcoming risks compromising efforts to meet national strategic health priorities and achieve SDG health targets.

To date, this skills gap has been bridged by outsourcing data analysis to international organisations and/or hiring foreign consultants. This has come at a significant risk of encouraging "parachute" research, whose objectives and findings have not necessarily been applicable to the local context nor aligned with national health agendas.

Recently, driven in part by the vital role data analysis has played and continues to play in the COVID-19 response locally, this critical shortage of data analysis skills has been recognised by the Seychelles Ministry of Health (MOH) as an acute problem, in need of urgent action.

We propose tackling this skills gap and building locally sustainable health research capacity by leveraging the vast research experience and skills of Oxford research partners to train and upskill data analysts from the Ministry of Health (MOH) Seychelles.

A *‘tailor-made’* curriculum will be developed and designed based on specific and current research-policy needs already identified by MOH partners. These key areas have been chosen based on impact, priority and what can be feasibly delivered in a 1-month period of intense training.

The topics covered by the training include:

1) Data analysis of health system performance indicators, particularly in health financing - cost-benefit and cost-effectiveness analyses.
2) Training in advanced statistical methods to support ongoing priority research and policy projects, specifically:
    (i) statistical methods for ongoing national vaccine effectiveness studies for COVID-19;
    (ii) training in survival analyses for specific diseases (i.e. statistical methods for cancer survival analyses); and,
    (iii) analyses in national mortality trends and surveillance.

The training will be delivered through both online coursework to be completed by all trainees over a 3-week period, followed by one week of intensive on-site training.

This handbook has been developed in support of the **Seychelles-Oxford Partnership Statistics Training** project which aims to strengthen local health research capacity in Seychelles through intensive training in health economic and statistical methods. This collaboration will build capacity for evidence-based health policy-making to better respond to local health challenges. In the long term, this initiative seeks to develop a culture of research in Small Island Developing States (SIDS), often under-represented in global heath research. It is therefore a step towards addressing this imbalance and promoting equitable participation and contribution of SIDS to global health research efforts.

Chapter 2

Installing R and RStudio

Steps in installing R and RStudio (and other required software) for Windows machines

Following is a simple guide to installing R and RStudio and other required software on Windows machines for the specific purpose of the Seychelles Ministry of Health training.

2.1 Install R

Important that R is installed first. R is the main software and is needed for RStudio to work properly. R should always be installed first.

Go to https://cran.r-project.org

Click on link that says Download R for Windows

Then click on the link that says install R for the first time

Then click on Download R-4.2.0 for Windows. This will start the download process.

Once downloaded, go to the exe file in your Downloads folder, double-click and follow all the install prompts, selecting recommended options all the time

2.2 Install RStudio

This step requires that step 1 has been done and was successful.

Go to https://www.rstudio.com/products/rstudio/download/

Select the download specific for your machine. If Windows - use this link to download - https://download1.rstudio.org/desktop/windows/RStudio-2022.02.2-485.exe

Once downloaded, double-click on exe file downloaded to your Downloads folder and then follow all install prompts, always selecting recommended options.

2.3 Install Rtools

For the things that I will be teaching you, we will need to expand the installation of R by installing the Rtools software from R.

The download link is https://cran.r-project.org/bin/windows/Rtools/rtools42/rtools.html if you have installed R version 4.2.0

Once you have downloaded the exe file, double-click on the exe file and follow all install prompts. Choose all the recommended options.

2.4 Install Git for Windows

For the things that I will be teaching you, we will need to install Git for Windows.

If your machine is 64-bit machine (most likely the case if your computer is new and running Windows 10 or later) - use this download link - https://github.com/git-for-windows/git/releases/download/v2.36.0.windows.1/Git-2.36.0-64-bit.exe

If your machine is 32-bit machine - use this link - https://github.com/git-for-windows/git/releases/download/v2.36.0.windows.1/Git-2.36.0-32-bit.exe

Once you have downloaded the exe file, double-click it and then follow all install prompts. Choose all recommended options.

Chapter 3

Introduction to R and RStudio

3.1 What is R?

R is a system for data manipulation, calculation, and graphics. It provides:

- Facilities for data handling and storage
- A large collection of tools for data analysis
- Graphical facilities for data analysis and display
- A simple but powerful programming language

R is often described as an environment for working with data. This is in contrast to a *package* which is a collection of very specific tools. R is not strictly a statistics system but a system that provides many classical and modern statistical procedures as part of a broader data-analysis tool. This is an important difference between R and other statistical systems. In R a statistical analysis is usually performed as a series of steps with intermediate results being stored in objects. Systems such as SPSS and SAS provide copious output from (e.g.) a regression analysis whereas R will give minimal output and store the results of a fit for subsequent interrogation or use with other R functions. This means that R can be tailored to produce exactly the analysis and results that you want rather than produce an analysis designed to fit all situations.

R is a language based product. This means that you interact with R by typing commands such as:

```
table(SEX, LIFE)
```

rather than by using menus, dialog boxes, selection lists, and buttons. This may seem to be a drawback but it means that the system is considerably more flexible than one that relies on menus, buttons, and boxes. It also means that every stage of your data management and analysis can be recorded and edited and re-run at a later date. It also provides an audit trail for quality control purposes.

`R` is available under UNIX (including Linux), the Macintosh operating system OS X, and Microsoft Windows. The method used for starting `R` will vary from system to system. On UNIX systems you may need to issue the `R` command in a terminal session or click on an icon or menu option if your system has a windowing system. On Macintosh systems `R` will be available as an application but can also be run in a terminal session. On Microsoft Windows systems there will usually be an icon on the Start menu or the desktop.

3.2 Why use R?

`R` is an open source system and is available under the *GNU general public license* (GPL) which means that it is available for free but that there are some restrictions on how you are allowed to distribute the system and how you may charge for bespoke data analysis solutions written using the `R` system. Details of the general public license are available from http://www.gnu.org/copyleft/gpl.html.

`R` is available for download from http://www.r-project.org/.

This is also the best place to get extension packages and documentation. You may also subscribe to the `R` mailing lists from this site. `R` is supported through mailing lists. The level of support is at least as good as for commercial packages. It is typical to have queries answered in a matter of a few hours.

Even though `R` is a free package it is more powerful than most commercial packages. Many of the modern procedures found in commercial packages were first developed and tested using `R` or **S-Plus** (the commercial equivalent of `R`).

3.3 What is RStudio

RStudio is an **integrated development environment (IDE)** for R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.

RStudio is available in open source and commercial editions and runs on the desktop (Windows, Mac, and Linux) or in a browser connected to RStudio Server or RStudio Workbench (Debian/Ubuntu, Red Hat/CentOS, and SUSE Linux).

Chapter 4

Basics of R

In this exercise we will use `R` to read a dataset and produce some descriptive statistics, produce some charts, and perform some simple statistical inference. The aim of the exercise is for you to become familiar with `R` and some basic `R` functions and objects.

The first thing we will do, after starting `R`, is issue a command to retrieve an example dataset:

```r
fem <- read.table("data/fem.dat", header = TRUE)
```

This command illustrates some key things about the way `R` works.

We are instructing `R` to assign (using the `<-` operator) the output of the `read.table()` function to an object called `fem`.

The `fem` object will contain the data held in the file `fem.dat` as an `R` data.frame object:

```r
class(fem)
```

```
## [1] "data.frame"
```

You can inspect the contents of the `fem` data.frame (or any other `R` object) just by typing its name:

```
fem
```

```
##   ID AGE IQ ANX DEP SLP SEX LIFE    WT
## 1  1  39 94   2   2   2   1    1  2.23
## 2  2  41 89   2   2   2   1    1  1.00
## 3  3  42 83   3   3   2   1    1  1.82
## 4  4  30 99   2   2   2   1    1 -1.18
## 5  5  35 94   2   1   1   1    2 -0.14
## 6  6  44 90  NA   1   2   2    2  0.41
```

Note that the `fem` object is built from other objects. These are the named vectors (columns) in the dataset:

```
names(fem)
```

```
## [1] "ID"   "AGE"  "IQ"   "ANX"  "DEP"  "SLP"  "SEX"  "LIFE" "WT"
```

The `[1]` displayed before the column names refers to the numbered position of the first name in the output. These positions are known as indexes and can be used to refer to individual items. For example:

```
names(fem)[1]
```

```
## [1] "ID"
```

```
names(fem)[8]
```

```
## [1] "LIFE"
```

```
names(fem)[2:4]
```

```
## [1] "AGE" "IQ"  "ANX"
```

The data consist of 118 records:

```
nrow(fem)
```

```
## [1] 118
```

each with nine variables:

```
ncol(fem)
```

```
## [1] 9
```

for female psychiatric patients.

The columns in the dataset are:

ID	Patient ID
AGE	Age in years
IQ	IQ score
ANX	Anxiety (1=none, 2=mild, 3=moderate, 4=severe)
DEP	Depression (1=none, 2=mild, 3=moderate or severe)
SLP	Sleeping normally (1=yes, 2=no)
SEX	Lost interest in sex (1=yes, 2=no)
LIFE	Considered suicide (1=yes, 2=no)
WT	Weight change (kg) in previous 6 months

The first ten records of the `fem` data.frame are:

```
##    ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39  94   2   2   2   1    1  2.23
## 2   2  41  89   2   2   2   1    1  1.00
## 3   3  42  83   3   3   2   1    1  1.82
## 4   4  30  99   2   2   2   1    1 -1.18
## 5   5  35  94   2   1   1   1    2 -0.14
## 6   6  44  90  NA   1   2   2    2  0.41
## 7   7  31  94   2   2  NA   1    1 -0.68
## 8   8  39  87   3   2   2   1    2  1.59
## 9   9  35 -99   3   2   2   1    1 -0.55
## 10 10  33  92   2   2   2   1    1  0.36
```

You may check this by asking `R` to display all columns of the first ten records in the `fem` data.frame:

```
fem[1:10, ]
```

```
##    ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 1   1  39  94   2   2   2   1    1  2.23
## 2   2  41  89   2   2   2   1    1  1.00
## 3   3  42  83   3   3   2   1    1  1.82
## 4   4  30  99   2   2   2   1    1 -1.18
## 5   5  35  94   2   1   1   1    2 -0.14
## 6   6  44  90  NA   1   2   2    2  0.41
## 7   7  31  94   2   2  NA   1    1 -0.68
## 8   8  39  87   3   2   2   1    2  1.59
## 9   9  35 -99   3   2   2   1    1 -0.55
## 10 10  33  92   2   2   2   1    1  0.36
```

The space after the comma is optional. You can think of it as a *placeholder* for where you would specify the indexes for columns you wanted to display. For example:

```
fem[1:10,2:4]
```

displays the first ten rows and the second, third and fourth columns of the `fem` data.frame:

```
##    AGE  IQ ANX
## 1   39  94   2
## 2   41  89   2
## 3   42  83   3
## 4   30  99   2
## 5   35  94   2
## 6   44  90  NA
## 7   31  94   2
## 8   39  87   3
## 9   35 -99   3
## 10  33  92   2
```

`NA` is a special value meaning *not available* or *missing*.

You can access the contents of a single column by name:

```
fem$IQ
```

```
##   [1]  94  89  83  99  94  90  94  87 -99  92  92  94  91  86  90 -99  91  82
##  [19]  86  88  97  96  95  87 103 -99  91  87  91  89  92  84  94  92  96  96
##  [37]  86  92 102  82  92  90  92  88  98  93  90  91 -99  92  92  91  91  86
##  [55]  95  91  96 100  99  89  89  98  98 103  91  91  94  91  85  92  96  90
##  [73]  87  95  95  87  95  88  94 -99 -99  87  92  86  93  92 106  93  95  95
##  [91]  92  98  92  88  85  92  84  92  91  86  92  89 -99  96  97  92  92  98
## [109]  91  91  89  94  90  96  87  86  89 -99
```

```
fem$IQ[1:10]
```

```
##  [1]  94  89  83  99  94  90  94  87 -99  92
```

The `$` sign is used to separate the name of the data.frame and the name of the column of interest. Note that `R` is case-sensitive so that `IQ` and `iq` are ***not*** the same.

You can also access rows, columns, and individual cells by specifying row and column positions. For example, the `IQ` column is the third column in the `fem` data.frame:

```
fem[ ,3]
```

```
##   [1]  94  89  83  99  94  90  94  87 -99  92  92  94  91  86  90 -99  91
##  [19]  86  88  97  96  95  87 103 -99  91  87  91  89  92  84  94  92  96
##  [37]  86  92 102  82  92  90  92  88  98  93  90  91 -99  92  92  91  91
##  [55]  95  91  96 100  99  89  89  98  98 103  91  91  94  91  85  92  96
##  [73]  87  95  95  87  95  88  94 -99 -99  87  92  86  93  92 106  93  95
##  [91]  92  98  92  88  85  92  84  92  91  86  92  89 -99  96  97  92  92
## [109]  91  91  89  94  90  96  87  86  89 -99
```

```
fem[9, ]
```

```
##   ID AGE  IQ ANX DEP SLP SEX LIFE    WT
## 9  9  35 -99   3   2   2   1    1 -0.55
```

```
fem[9,3]
```

```
## [1] -99
```

There are missing values in the `IQ` column which are all coded as **-99**. Before proceeding we must set these to the special `NA` value:

```
fem$IQ[fem$IQ == -99] <- NA
```

The term inside the square brackets is also an index. This type of index is used to refer to subsets of data held in an object that meet a particular condition. In this case we are instructing `R` to set the contents of the `IQ` variable to `NA` if the contents of the `IQ` variable is **-99**.

Check that this has worked:

```
fem$IQ
```

```
##   [1]  94  89  83  99  94  90  94  87  NA  92  92  94  91  86  90  NA  91  82
##  [19]  86  88  97  96  95  87 103  NA  91  87  91  89  92  84  94  92  96  96
##  [37]  86  92 102  82  92  90  92  88  98  93  90  91  NA  92  92  91  91  86
##  [55]  95  91  96 100  99  89  89  98  98 103  91  91  94  91  85  92  96  90
##  [73]  87  95  95  87  95  88  94  NA  NA  87  92  86  93  92 106  93  95  95
##  [91]  92  98  92  88  85  92  84  92  91  86  92  89  NA  96  97  92  92  98
## [109]  91  91  89  94  90  96  87  86  89  NA
```

We can now compare the groups who have and have not considered suicide. For example:

```
by(fem$IQ, fem$LIFE, summary)
```

Look at the help for the `by()` function:

```
help(by)
```

Note that you may use `?by` as a shortcut for `help(by)`.

The `by()` function applies another function (in this case the `summary()` function) to a column in a data.frame (in this case `fem$IQ`) split by the value of another variable (in this case `fem$LIFE`).

It can be tedious to always have to specify a data.frame each time we want to use a particular variable. We can fix this problem by 'attaching' the data.frame:

```
attach(fem)
```

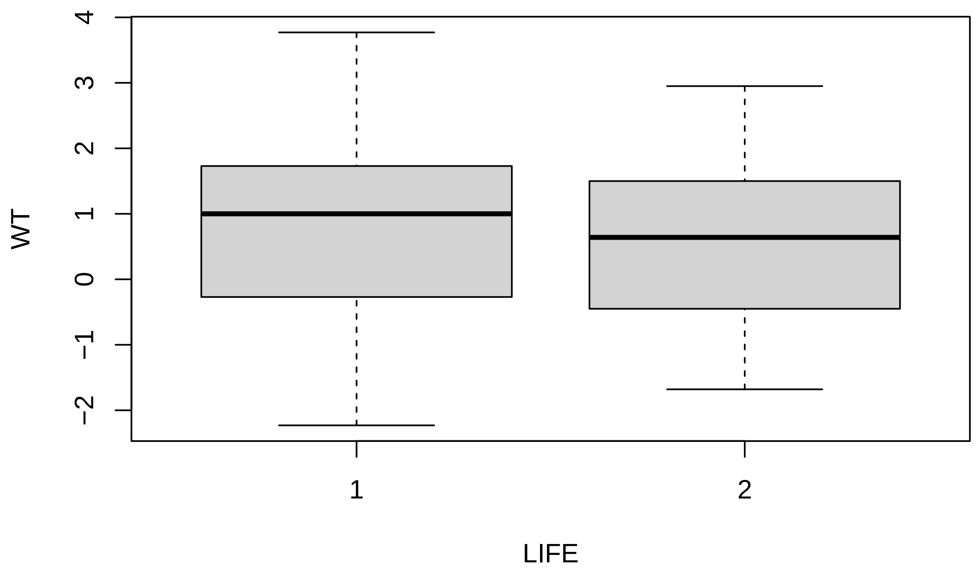
We can now refer to the columns in the `fem` data.frame without having to specify the name of the data.frame. This time we will produce summary statistics for `WT` by `LIFE`:

```
by(WT, LIFE, summary)
```

```
## LIFE: 1
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
## -2.2300 -0.2700  1.0000  0.7867  1.7300  3.7700       4 
## ------------------------------------------------------------
## LIFE: 2
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
## -1.6800 -0.4500  0.6400  0.6404  1.5000  2.9500       7
```

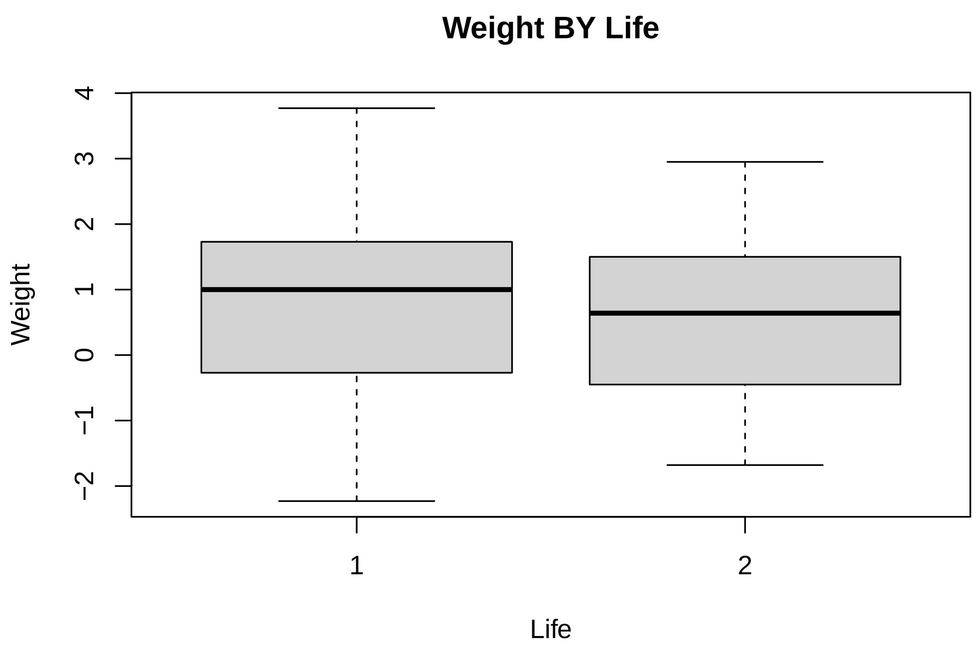
We can view the same data as a box and whisker plot:

```
boxplot(WT ~ LIFE)
```



We can add axis labels and a title to the graph:

```
boxplot(WT ~ LIFE,
        xlab = "Life",
        ylab = "Weight",
        main = "Weight BY Life")
```

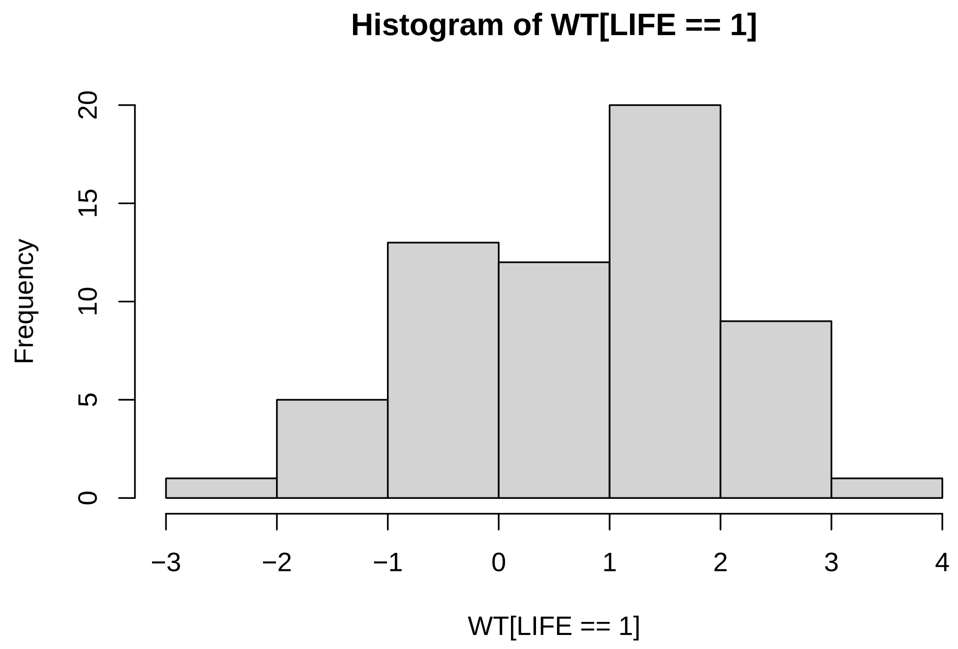


A more descriptive title might be “Weight Change BY Considered Suicide”.

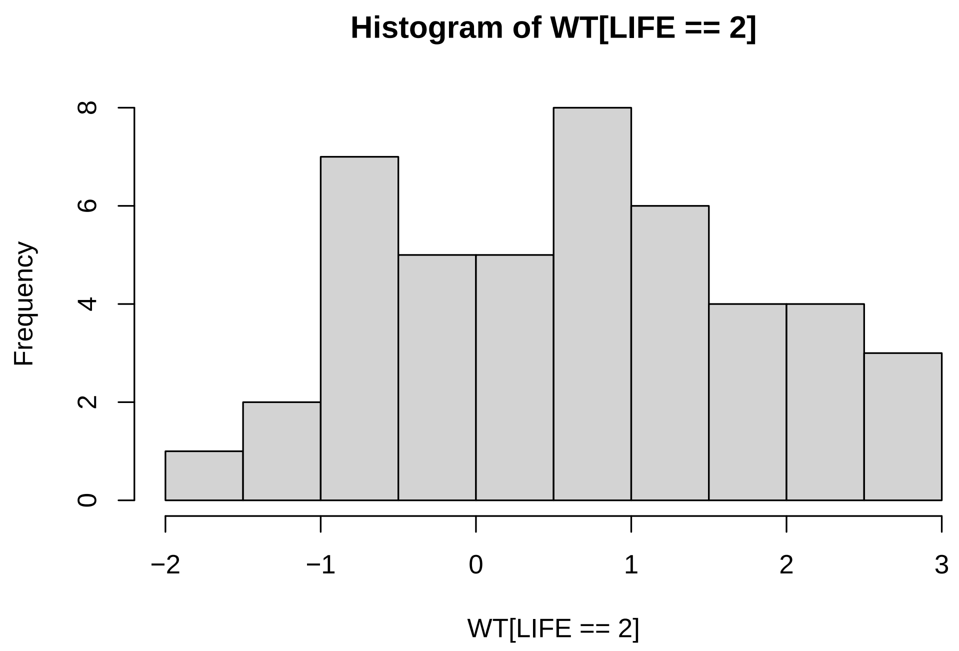
The groups do not seem to differ much in their medians and the distributions appear to be reasonably symmetrical about their medians with a similar spread of values.

We can look at the distribution as histograms:

```
hist(WT[LIFE == 1])
```

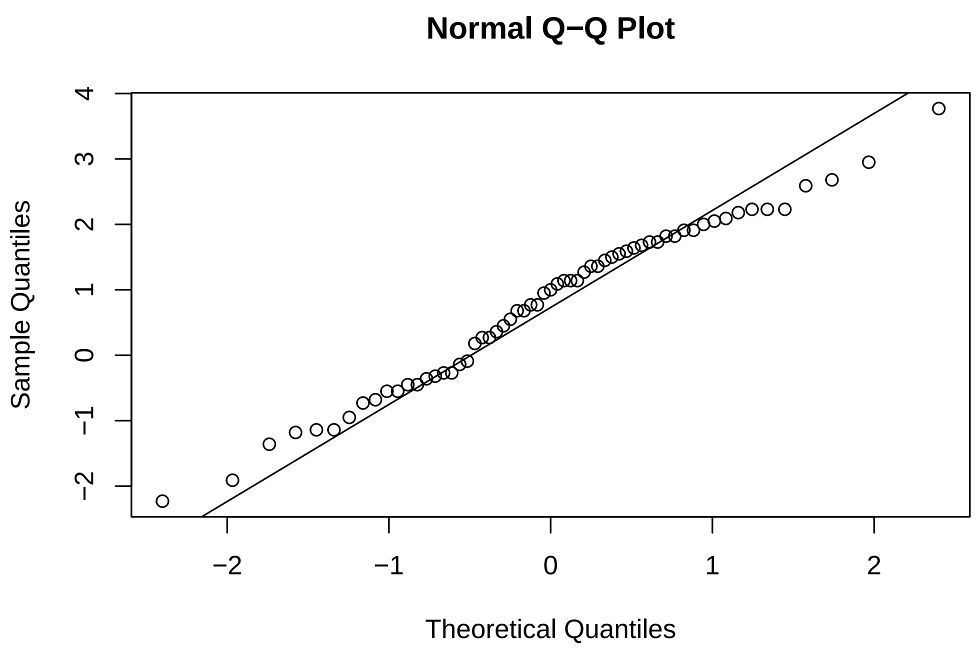


```
hist(WT[LIFE == 2])
```

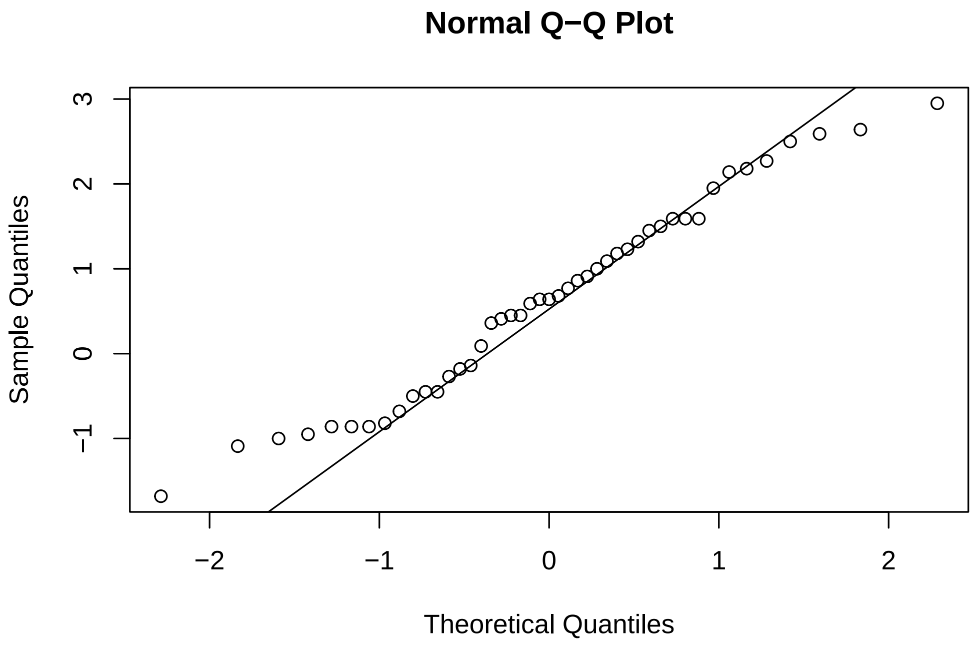

Histogram of WT[LIFE == 2]
Frequency
0
2
4
6
8
−2
−1
0
1
2
3
WT[LIFE == 2]

and check the assumption of normality using quantile-quantile plots:

```
qqnorm(WT[LIFE == 1])
qqline(WT[LIFE == 1])
```



```
qqnorm(WT[LIFE == 2])
qqline(WT[LIFE == 2])
```


Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2
3

or by using a formal test:

```
shapiro.test(WT[LIFE == 1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  WT[LIFE == 1]
## W = 0.98038, p-value = 0.4336
```

```
shapiro.test(WT[LIFE == 2])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  WT[LIFE == 2]
## W = 0.97155, p-value = 0.3292
```

Remember that we can use the `by()` function to apply a function to a data.frame, including statistical functions such as `shapiro.test()`:

```
by(WT, LIFE, shapiro.test)
```

```
## LIFE: 1
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.98038, p-value = 0.4336
##
## ------------------------------------------------------------
## LIFE: 2
##
##  Shapiro-Wilk normality test
##
## data:  dd[x, ]
## W = 0.97155, p-value = 0.3292
```

We can also test whether the variances differ significantly using *Bartlett's test* for the homogeneity of variances:

```
bartlett.test(WT, LIFE)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  WT and LIFE
## Bartlett's K-squared = 0.32408, df = 1, p-value = 0.5692
```

There is no significant difference between the two variances.

Many functions in `R` have a *formula interface* that may be used to specify multiple variables and the relations between multiple variables. We could have used the formula interface with the `bartlett.test()` function:

```
bartlett.test(WT ~ LIFE)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  WT by LIFE
## Bartlett's K-squared = 0.32408, df = 1, p-value = 0.5692
```

Having checked the normality and homogeneity of variance assumptions we can proceed to carry out a `t-test`:

```
t.test(WT ~ LIFE, var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  WT by LIFE
## t = 0.59869, df = 104, p-value = 0.5507
## alternative hypothesis: true difference in means between group 1 and group 2 is no
## 95 percent confidence interval:
##  -0.3382365  0.6307902
## sample estimates:
## mean in group 1 mean in group 2
##       0.7867213       0.6404444
```

There is no evidence that the two groups differ in weight change in the previous six months.

We could still have performed a `t-test` if the variances were not homogenous by setting the **var.equal** parameter of the `t.test()` function to **FALSE**:

```
t.test(WT ~ LIFE, var.equal = FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  WT by LIFE
## t = 0.60608, df = 98.866, p-value = 0.5459
## alternative hypothesis: true difference in means between group 1 and group
## 95 percent confidence interval:
##  -0.3326225  0.6251763
## sample estimates:
## mean in group 1 mean in group 2
##       0.7867213       0.6404444
```

or performed a non-parametric test:

```
wilcox.test(WT ~ LIFE)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  WT by LIFE
## W = 1488, p-value = 0.4622
## alternative hypothesis: true location shift is not equal to 0
```

An alternative, and more general, non-parametric test is:

```
kruskal.test(WT ~ LIFE)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  WT by LIFE
## Kruskal-Wallis chi-squared = 0.54521, df = 1, p-value = 0.4603
```

We can use the `table()` function to examine the differences in depression between the two groups:

```
table(DEP, LIFE)
```

```
##    LIFE
## DEP  1  2
##   1  0 26
##   2 42 24
##   3 16  1
```

The two distributions look very different from each other. We can test this using a chi-square test on the table:

```
chisq.test(table(DEP, LIFE))
```

```
##
##  Pearson's Chi-squared test
##
## data:  table(DEP, LIFE)
## X-squared = 43.876, df = 2, p-value = 2.968e-10
```

Note that we passed the output of the `table()` function directly to the `chisq.test()` function. We could have saved the table as an object first and then passed the object to the `chisq.test()` function:

```
tab <- table(DEP, LIFE)
chisq.test(tab)
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = 43.876, df = 2, p-value = 2.968e-10
```

The `tab` object contains the output of the `table()` function:

```
class(tab)
```

```
## [1] "table"
```

```
tab
```

```
##    LIFE
## DEP  1  2
##   1  0 26
##   2 42 24
##   3 16  1
```

We can pass this table object to another function. For example:

```
fisher.test(tab)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  tab
## p-value = 1.316e-12
## alternative hypothesis: two.sided
```

When we are finished with the tab object we can delete it using the `rm()` function:

```
rm(tab)
```

You can see a list of available objects using the `ls()` function:

```
ls()
```

```
## [1] "fem"
```

This should just show the `fem` object.

We can examine the association between loss of interest in sex and considering suicide in the same way:

```
tab <- table(SEX, LIFE)
tab
```

```
##    LIFE
## SEX  1  2
##   1 58 38
##   2  5 12
```

```
fisher.test(tab)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  tab
## p-value = 0.03175
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   1.080298 14.214482
## sample estimates:
## odds ratio
##   3.620646
```

Note that with a two-by-two table the `fisher.test()` function produces an estimate of, and confidence intervals for, the odds ratio. Again, we will delete the `tab` object:

```
rm(tab)
```

We could have performed the Fisher exact test without creating the tab object by passing the output of the `table()` function directly to the `fisher.test()` function:

```
fisher.test(table(SEX, LIFE))
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  table(SEX, LIFE)
## p-value = 0.03175
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##   1.080298 14.214482
## sample estimates:
## odds ratio
##   3.620646
```

Choose whichever method you find easiest but remember that it is easy to save the results of any function for later use.

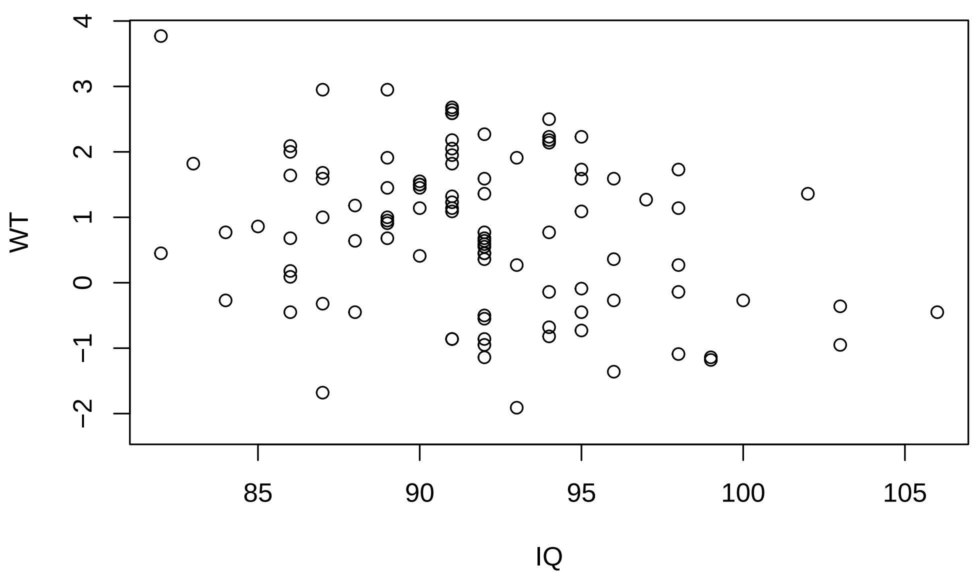
We can explore the correlation between two variables using the `cor()` function:

```
cor(IQ, WT, use = "pairwise.complete.obs")
```

```
## [1] -0.2917158
```

or by using a scatter plot:

```
plot(IQ, WT)
```



and by a formal test:

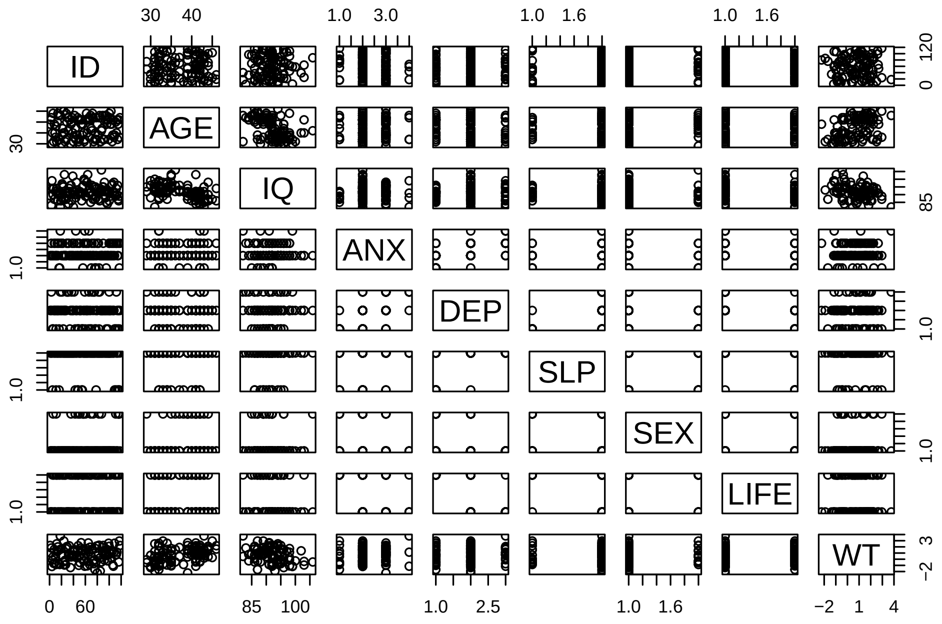
```
cor.test(IQ, WT)
```

```
##
##  Pearson's product-moment correlation
##
## data:  IQ and WT
## t = -3.0192, df = 98, p-value = 0.003231
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.4616804 -0.1010899
## sample estimates:
##        cor
## -0.2917158
```

With some functions you can pass an entire data.frame rather than a list of variables:

```
cor(fem, use = "pairwise.complete.obs")
pairs(fem)
```

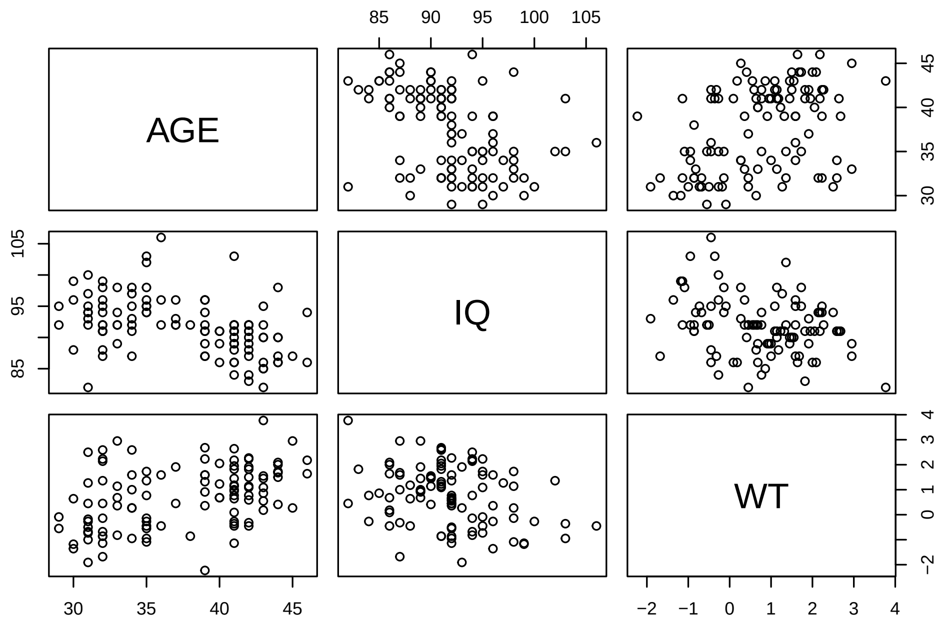
```
##                ID         AGE            IQ         ANX          DEP
## ID    1.00000000  0.03069077  0.0370598672 -0.02941825 -0.0554147209
## AGE   0.03069077  1.00000000 -0.4345435680  0.06734300 -0.0387049246
## IQ    0.03705987 -0.43454357  1.0000000000 -0.02323787 -0.0001307404
## ANX  -0.02941825  0.06734300 -0.0232378691  1.00000000  0.5437946347
## DEP  -0.05541472 -0.03870492 -0.0001307404  0.54379463  1.0000000000
## SLP  -0.07268743  0.02606547  0.0812993104  0.22317875  0.5248724551
## SEX   0.08999634  0.10609216 -0.0536558660 -0.21062493 -0.3058422258
## LIFE -0.05604349 -0.10300193 -0.0915396469 -0.34211268 -0.6139017253
## WT    0.02640131  0.41574411 -0.2917157832  0.11817532  0.0233742465
##                SLP         SEX        LIFE           WT
## ID   -0.072687434  0.08999634 -0.05604349  0.026401310
## AGE   0.026065468  0.10609216 -0.10300193  0.415744109
## IQ    0.081299310 -0.05365587 -0.09153965 -0.291715783
## ANX   0.223178752 -0.21062493 -0.34211268  0.118175321
## DEP   0.524872455 -0.30584223 -0.61390173  0.023374247
## SLP   1.000000000 -0.29053971 -0.35186578 -0.009259774
## SEX  -0.290539709  1.00000000  0.22316967 -0.027826514
## LIFE -0.351865775  0.22316967  1.00000000 -0.058605326
## WT   -0.009259774 -0.02782651 -0.05860533  1.000000000
```



The output can be a little confusing particularly if it includes categorical or record identifying variables. To avoid this we can create a new object that contains only the columns we are interested in using the column binding cbind() function:

```
newfem <- cbind(AGE, IQ, WT)
cor(newfem, use = "pairwise.complete.obs")
pairs(newfem)
```

```
##            AGE         IQ         WT
## AGE  1.0000000 -0.4345436  0.4157441
## IQ  -0.4345436  1.0000000 -0.2917158
## WT   0.4157441 -0.2917158  1.0000000
```



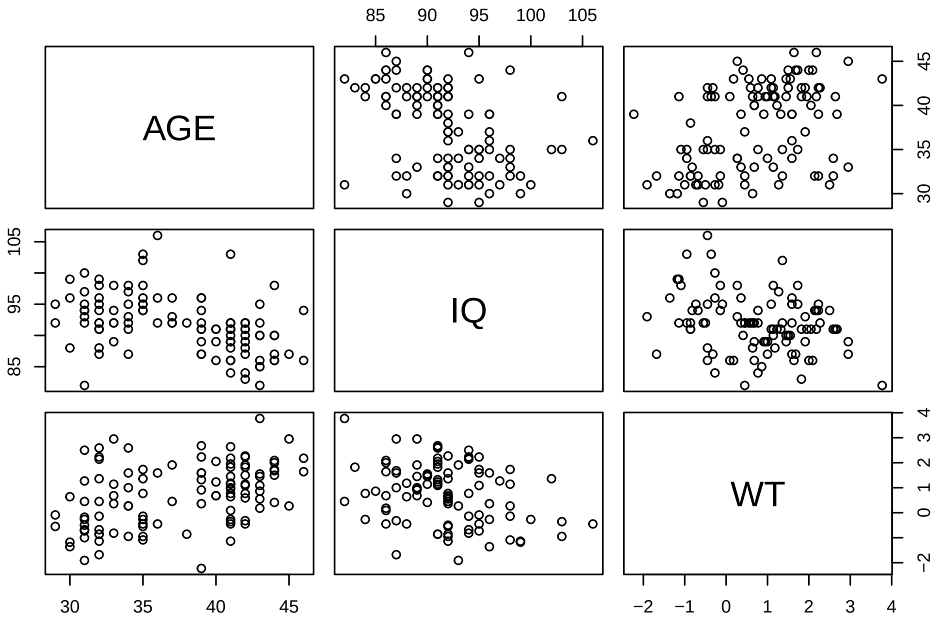
When we have finished with the **newfem** object we can delete it:

```
rm(newfem)
```

There was no real need to create the `newfem` object as we could have fed the output of the `cbind()` function directly to the `cor()` or `pairs()` function:

```
cor(cbind(AGE, IQ, WT), use = "pairwise.complete.obs")
pairs(cbind(AGE, IQ, WT))
```

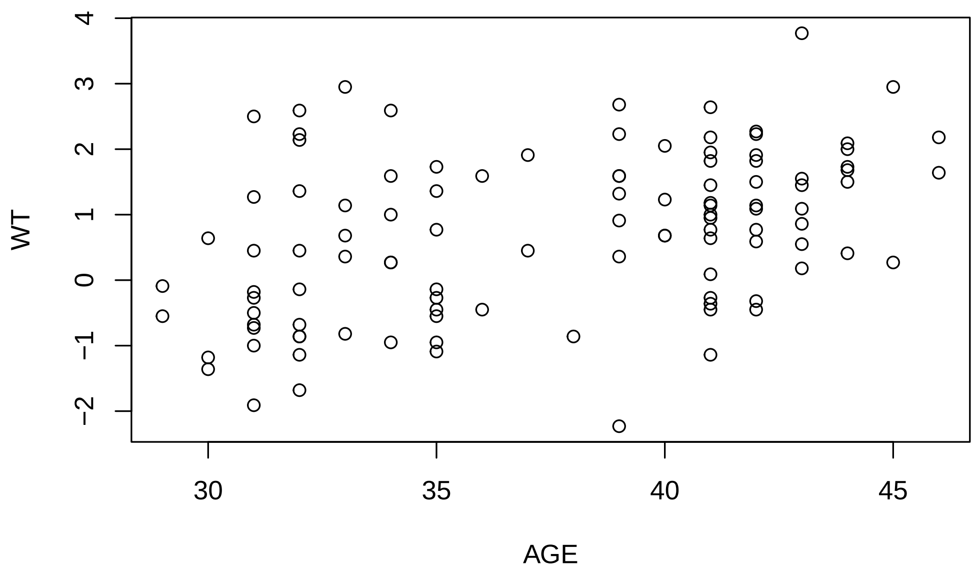
```
##            AGE         IQ         WT
## AGE  1.0000000 -0.4345436  0.4157441
## IQ  -0.4345436  1.0000000 -0.2917158
## WT   0.4157441 -0.2917158  1.0000000
```



It is, however, easier to work with the `newfem` object rather than having to retype the `cbind()` function. This is particularly true if you wanted to continue with an analysis of just the three variables.

The relationship between `AGE` and `WT` can be plotted using the `plot()` function:

```
plot(AGE, WT)
```



WT
4
3
2
1
0
−1
−2
30
35
40
45
AGE

And tested using the `cor()` and `cor.test()` functions:

```
cor(AGE, WT, use = "pairwise.complete.obs")
```

```
## [1] 0.4157441
```

```
cor.test(AGE, WT)
```

```
##
##  Pearson's product-moment correlation
##
## data:  AGE and WT
## t = 4.6841, df = 105, p-value = 8.457e-06
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2452434 0.5612979
## sample estimates:
##       cor
## 0.4157441
```

Or by using the linear modelling `lm()` function:

```
summary(lm(WT ~ AGE))
```

```
##
## Call:
## lm(formula = WT ~ AGE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.10678 -0.85922 -0.05453  0.71434  2.70874
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.25405    0.85547  -3.804  0.00024 ***
## AGE          0.10592    0.02261   4.684 8.46e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.128 on 105 degrees of freedom
##   (11 observations deleted due to missingness)
## Multiple R-squared:  0.1728, Adjusted R-squared:  0.165
## F-statistic: 21.94 on 1 and 105 DF,  p-value: 8.457e-06
```

We use the `summary()` function here to extract summary information from the output of the `lm()` function.

It is often more useful to use `lm()` to create an object:

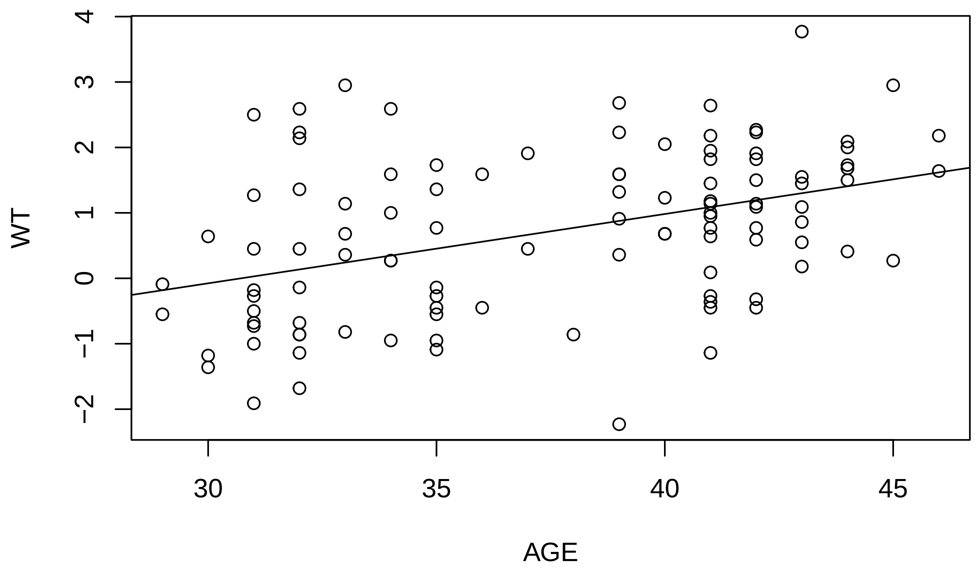
```
fem.lm <- lm(WT ~ AGE)
```

And use the output in other functions:

```
summary(fem.lm)
```

```
##
## Call:
## lm(formula = WT ~ AGE)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.10678 -0.85922 -0.05453  0.71434  2.70874
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.25405    0.85547  -3.804  0.00024 ***
## AGE          0.10592    0.02261   4.684 8.46e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.128 on 105 degrees of freedom
##   (11 observations deleted due to missingness)
## Multiple R-squared:  0.1728, Adjusted R-squared:  0.165
## F-statistic: 21.94 on 1 and 105 DF,  p-value: 8.457e-06
```

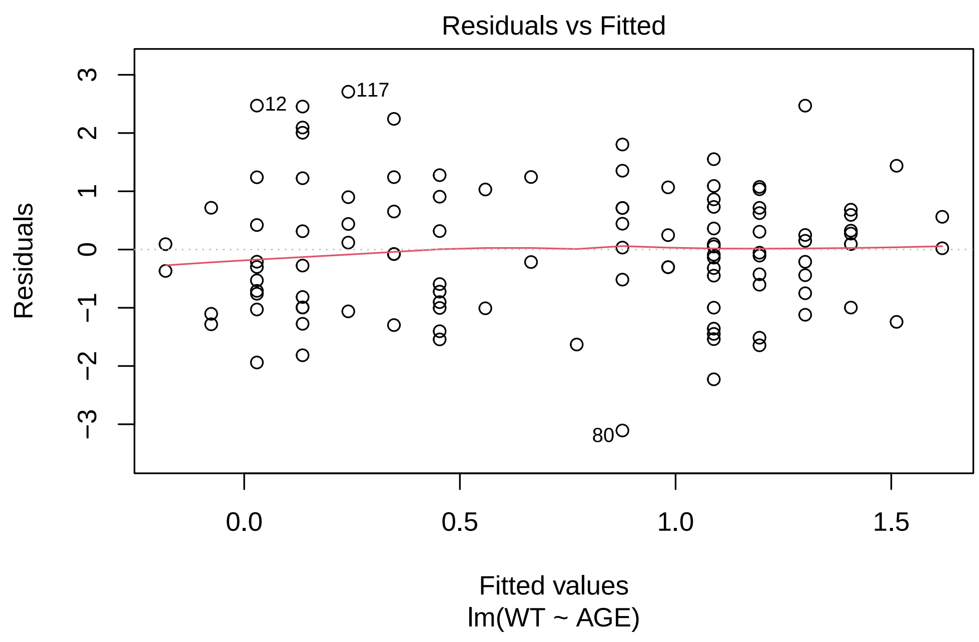
```
plot(AGE, WT)
abline(fem.lm)
```

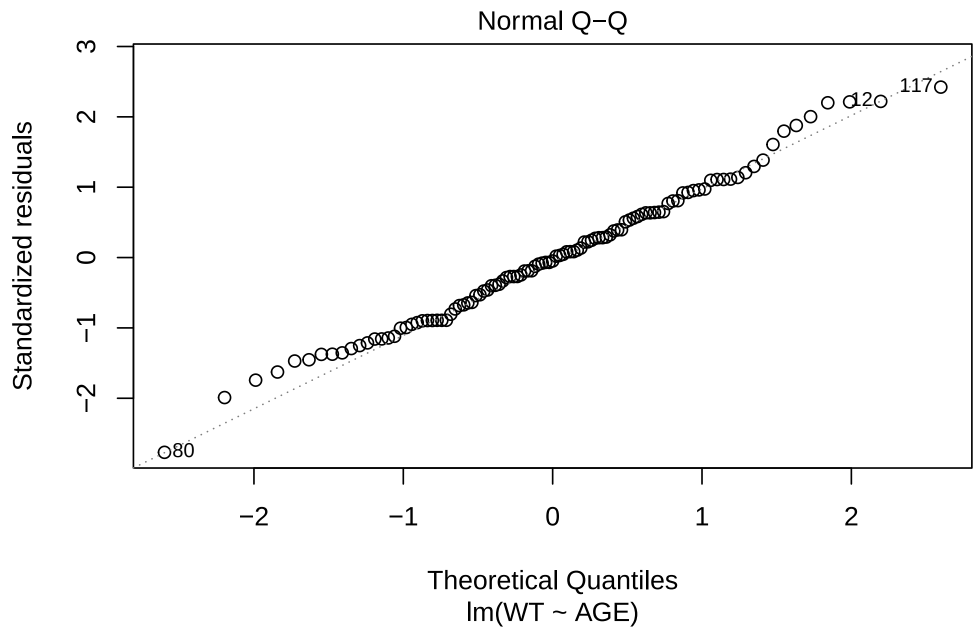


In this case we are passing the intercept and slope information held in the `fem.lm` object to the `abline()` function which draws a regression line. The `abline()` function adds to an existing plot. This means that you need to keep the scatter plot of `AGE` and `WT` open before issuing the `abline()` function call.

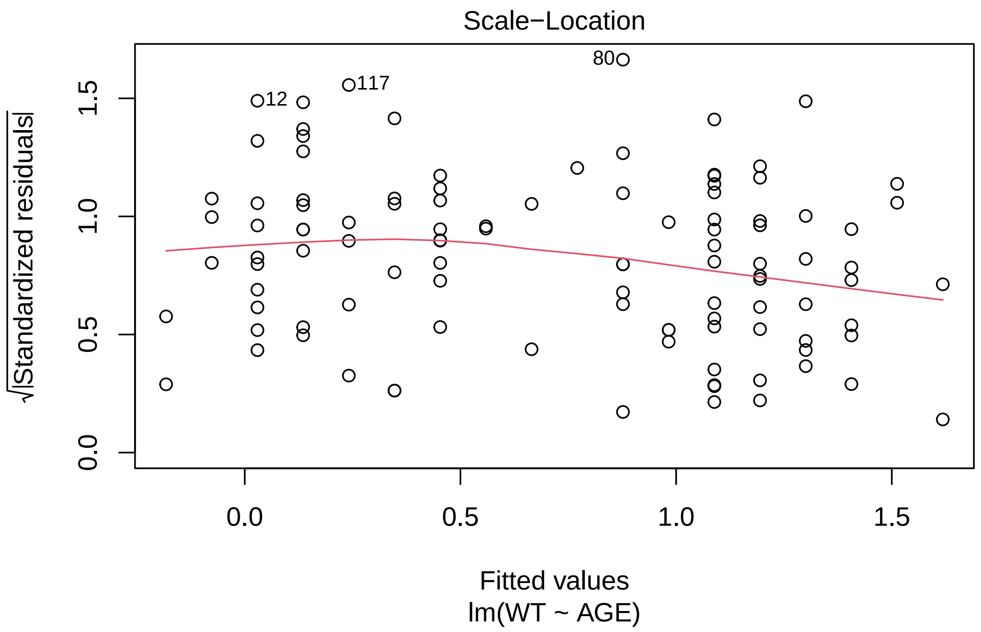
A useful function to apply to the `fem.lm` object is `plot()` which produces diagnostic plots of the linear model:

```
plot(fem.lm)
```

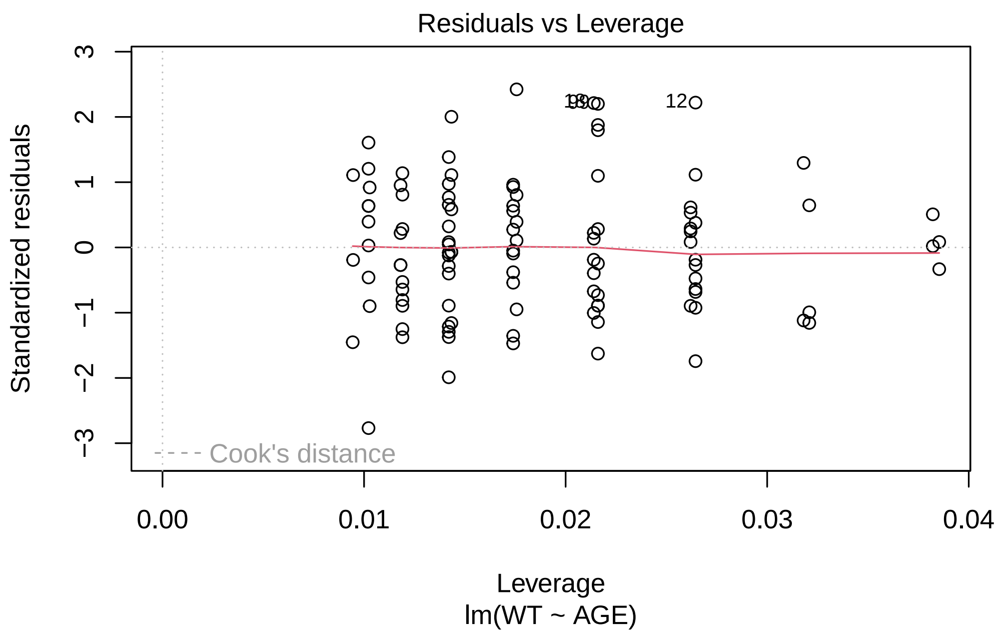




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(WT ~ AGE)
80
12
117



Scale−Location
√|Standardized residuals|
Fitted values
lm(WT ~ AGE)
80
117
12

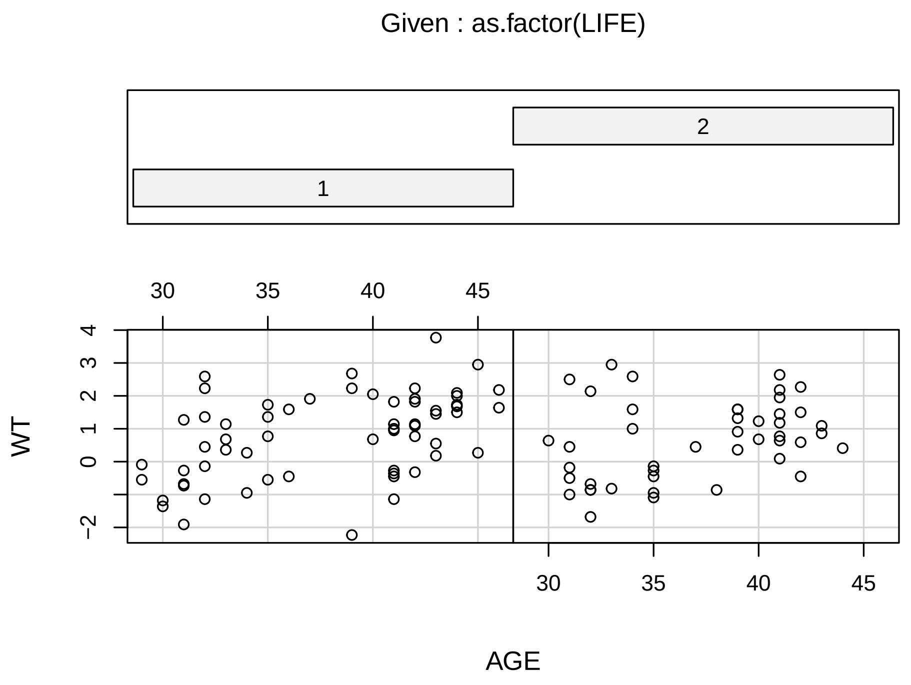

Residuals vs Leverage
Standardized residuals
Leverage
lm(WT ~ AGE)
Cook's distance
12
0.00
0.01
0.02
0.03
0.04
3
2
1
0
−1
−2
−3

Objects created by the `lm()` function (or any of the modelling functions) can use up a lot of memory so we should remove them when we no longer need them:

```
rm(fem.lm)
```

It might be interesting to see whether a similar relationship exists between `AGE` and `WT` for those who have and have not considered suicide. This can be done using the `coplot()` function:

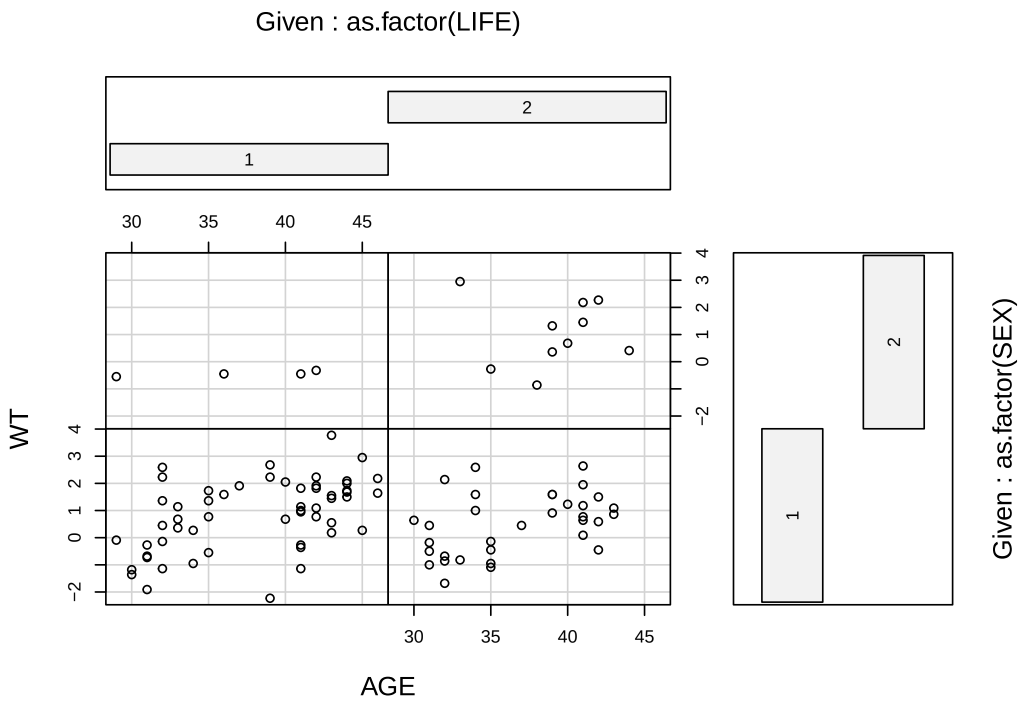
```
coplot(WT ~ AGE | as.factor(LIFE))
```



```
##
##  Missing rows: 21, 22, 31, 43, 44, 45, 69, 81, 101, 104, 114, 115
```

The two plots looks similar. We could also use `coplot()` to investigate the relationship between `AGE` and `WT` for categories of both `LIFE` and `SEX`:

```
coplot(WT ~ AGE | as.factor(LIFE) * as.factor(SEX))
```



```
##
##  Missing rows: 12, 17, 21, 22, 31, 43, 44, 45, 66, 69, 81, 101, 104, 105, 114, 115
```

although the numbers are too small for this to be useful here.

We used the `as.factor()` function with the `coplot()` function to ensure that `R` was aware that the `LIFE` and `SEX` columns hold categorical data.

We can check the way variables are stored using the `data.class()` function:

```
data.class(fem$SEX)
```

```
## [1] "numeric"
```

We can ‘apply’ this function to all columns in a data.frame using the `sapply()` function:

```
sapply(fem, data.class)
```

```
##        ID       AGE        IQ       ANX       DEP       SLP       SEX
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "nume
##        WT
## "numeric"
```

The `sapply()` function is part of a group of functions that apply a specified function to data objects:

Function(s)	**Applies a function to . . .**
`apply()`	rows and columns of matrices, arrays, and tables
`lapply()`	components of lists and data.frames
`sapply()`	components of lists and data.frames
`mapply()`	components of lists and data.frames
`tapply()`	subsets of data

Related functions are `aggregate()` which compute summary statistics for subsets of data, `by()` which applies a function to a data.frame split by factors, and `sweep()` which applies a function to an array.

The parameters of most `R` functions have default values. These are usually the most used and most useful parameter values for each function. The `cor.test()` function, for example, calculates *Pearson’s product moment correlation coefficient* by default. This is an appropriate measure for data from a bivariate normal distribution. The `DEP` and `ANX` variables contain ordered data. An appropriate measure of correlation between `DEP` and `ANX` is *Kendall’s tau*. This can be obtained using:

```
cor.test(DEP, ANX, method = "kendall")
```

```
##
##  Kendall's rank correlation tau
##
## data:  DEP and ANX
## z = 5.5606, p-value = 2.689e-08
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##       tau
## 0.4950723
```

Before we finish we should save the `fem` data.frame so that next time we want to use it we will not have to bother with recoding the missing values to the special `NA` value. This is done with the `write.table()` function:

```
write.table(fem, file = "newfem.dat", row.names = FALSE)
```

Everything in `R` is either a function or an object. Even the command to quit `R` is a function:

```
q()
```

When you call the `q()` function you will be asked if you want to save the workspace image. If you save the workspace image then all of the objects and functions currently available to you will be saved. These will then be automatically restored the next time you start `R` in the current working directory.

For this exercise there is no need to save the workspace image so click the **No** or **Don't Save** button (GUI) or enter `n` when prompted to save the workspace image (terminal).

4.1 Summary

- `R` is a functional system. Everything is done by calling functions.
- `R` provides a large set of functions for descriptive statistics, charting, and statistical inference.
- Functions can be chained together so that the output of one function is the input of another function.
- `R` is an object oriented system. We can use functions to create objects that can then be manipulated or passed to other functions for subsequent analysis.

Chapter 5

Objects in R

In this exercise we will explore how to manipulate `R` objects and how to write functions that can manipulate and extract data and information from `R` objects and produce useful analyses.

Before we go any further we should start `R` and retrieve a dataset:

```
salex <- read.table("data/salex.dat", header = TRUE, na.strings = "9")
```

Missing values are coded as 9 throughout this dataset so we can use the `na.strings` parameter of the `read.table()` function to replace all 9's with the special `NA` code when we retrieve the dataset. Check that this works by examining the `salex` data.frame:

```
salex
names(salex)
```

```
##   ILL HAM BEEF EGGS MUSHROOM PEPPER PORKPIE PASTA RICE LETTUCE TOMATO COLESLAW
## 1   1   1    1    1        1      1       2     2    2       2      2        2
## 2   1   1    1    1        2      2       1     2    2       2      1        2
## 3   1   1    1    1        1      1       1     1    1       1      2        2
## 4   1   1    1    1        2      2       2     2    2       1      1        2
## 5   1   1    1    1        1      1       1     1    1       1      1        1
## 6   1   1    1    1        2      2       2     2    2       2      1        1
##   CRISPS PEACHCAKE CHOCOLATE FRUIT TRIFLE ALMONDS
## 1      2         2         2     2      2       2
## 2      2          2          2     2      2       2
## 3      1          2          1     2      2       2
## 4      2          2          1     2      2       2
## 5      2          2          1     2      1       2
## 6      1          2          1     2      2       2
```

```
##  [1] "ILL"        "HAM"        "BEEF"       "EGGS"       "MUSHROOM"   "PEPPER"
##  [7] "PORKPIE"    "PASTA"      "RICE"       "LETTUCE"    "TOMATO"     "COLESLAW"
## [13] "CRISPS"     "PEACHCAKE" "CHOCOLATE" "FRUIT"      "TRIFLE"     "ALMONDS"
```

This data comes from a food-borne outbreak. On Saturday 17th October 1992, eighty-two people attended a buffet meal at a sports club. Within fourteen to twenty-four hours, fifty-one of the participants developed diarrhoea, with nausea, vomiting, abdominal pain and fever.

The columns in the dataset are as follows:

ILL	Ill or not-ill
HAM	Baked ham
BEEF	Roast beef
EGGS	Eggs
MUSHROOM	Mushroom flan
PEPPER	Pepper flan
PORKPIE	Pork pie
PASTA	Pasta salad
RICE	Rice salad
LETTUCE	Lettuce
TOMATO	Tomato salad
COLESLAW	Coleslaw
CRISPS	Crisps
PEACHCAKE	Peach cake
CHOCOLATE	Chocolate cake
FRUIT	Tropical fruit salad
TRIFLE	Trifle
ALMONDS	Almonds

Data is available for seventy-seven of the eighty-two people who attended the sports club buffet. All of the variables are coded 1=yes, 2=no.

We can use the `attach()` function to make it easier to access our data:

```
attach(salex)
```

The two-by-two table is a basic epidemiological tool. In analysing data from a food-borne outbreak collected as a retrospective cohort study, for example, we would tabulate each exposure (suspect foodstuffs) against the outcome (illness) and calculate risk ratios and confidence intervals. `R` has no explicit function to calculate risk ratios from two-by-two tables but we can easily write one ourselves.

The first step in writing such a function would be to create the two-by-two table. This can be done with the `table()` function. We will use a table of `HAM` by `ILL` as an illustration:

```
table(HAM, ILL)
```

This command produces the following output:

```
##    ILL
## HAM  1  2
##   1 46 17
##   2  5  9
```

We can manipulate the output directly but it is easier if we instruct `R` to save the output of the `table()` function in an object:

```
tab <- table(HAM, ILL)
```

The `tab` object contains the output of the `table()` function:

```
tab
```

```
##    ILL
## HAM  1  2
##   1 46 17
##   2  5  9
```

As it is stored in an object we can examine its contents on an item by item basis.

The `tab` object is an object of class `table`:

```
class(tab)
```

```
## [1] "table"
```

We can extract data from a table object by using indices or row and column co-ordinates:

```
tab[1,1]
tab[1,2]
tab[2,1]
```

```
## [1] 46
```

```
## [1] 17
```

```
## [1] 5
```

The numbers in the square brackets refer to the ***position*** (as row and column co-ordinates) of the data item in the table ***not*** the ***values*** of the variables. We can extract data using the values of the row and column variables by enclosing the index values in double quotes (“). For example:

```
tab["1","1"]
```

```
## [1] 46
```

The two methods of extracting data may be combined. For example:

```
tab[1,"1"]
```

```
## [1] 46
```

We can calculate a risk ratio using the extracted data:

```
(tab[1,1]/(tab[1,1]+tab[1,2]))/(tab[2,1]/(tab[2,1]+tab[2,2]))
```

Which returns a risk ratio of

```
## [1] 2.044444
```

This is a tedious calculation to have to type in every time you need to calculate a risk ratio from a two-by-two table. It would be better to have a function that calculates and displays the risk ratio automatically. Fortunately, `R` allows us to do just that.

The `function()` function allows us to create new functions in `R`:

```
tab2by2 <- function(exposure, outcome) {}
```

This creates an empty function called `tab2by2` that expects two parameters called `exposure` and `outcome`. We could type the whole function in at the `R` command prompt but it is easier to use a text editor:

```
fix(tab2by2)
```

This will start an editor with the empty `tab2by2()` function already loaded. We can now edit this function to make it do something useful:

```r
function(exposure, outcome)
  {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  print(tab)
  print(rr)
  }
```

Once you have made the changes shown above, check your work, save the file, and quit the editor. Before proceeding we should examine the `tab2by2()` function to make sure we understand what the function will do:

- The first line defines `tab2by2` as a function that expects to be given two parameters which are called `exposure` and `outcome`.
- The body of the function (i.e. the work of the function) is enclosed within curly brackets (`{}`).
- The first line of the body of the function creates a table object (`tab`) using the variables specified when the `tab2by2()` function is called (these are the parameters `exposure` and `outcome`).
- The next line creates four new objects (called `a`, `b`, `c`, and `d`) which contain the values of the four cells in the two-by-two table.
- The following line calculates the risk ratio using the objects `a`, `b`, `c`, and `d` and stores the result of the calculation in an object called `rr`.
- The final two lines print the contents of the `tab` and `rr` objects.

Let's try the `tab2by2()` function with our test data:

```
tab2by2(HAM, ILL)
```

```
##         outcome
## exposure  1  2
##        1 46 17
##        2  5  9
## [1] 2.044444
```

The `tab2by2()` function displays a table of `HAM` by `ILL` followed by the risk ratio calculated from the data in the table.

Try producing another table:

```
tab2by2(PASTA, ILL)
```

```
##         outcome
## exposure  1  2
##        1 25  3
##        2 26 23
## [1] 1.682692
```

Have a look at the `R` objects available to you:

```
ls()
```

```
## [1] "fem"     "salex"   "tab"     "tab2by2"
```

Note that there are no `a`, `b`, `c`, `d`, or `rr` objects.

Examine the `tab` object:

```
tab
```

```
##    ILL
## HAM  1  2
##   1 46 17
##   2  5  9
```

This is the table of `HAM` by `ILL` that you created earlier ***not*** the table of `PASTA` by `ILL` that was created by the `tab2by2()` function.

The `tab`, `a`, `b`, `c`, `d`, and `rr` objects in the `tab2by2()` function are local to that function and do not change anything outside of that function. This means that the `tab` object inside the function is independent of any object of the same name outside of the function.

When a function completes its work, all of the objects that are local to that function are automatically removed. This is useful as it means that you can use object names inside functions that will not interfere with objects of the same name that are stored elsewhere. It also means that you do not clutter up the `R` workspace with temporary objects.

Just to prove that `tab` in the `tab2by2()` function exists only in the `tab2by2()` function we can delete the tab object from the `R` workspace:

```
rm(tab)
```

Now try another call to the `tab2by2()` function:

```
tab2by2(FRUIT, ILL)
```

```
##         outcome
## exposure  1  2
##        1  1  4
##        2 49 22
## [1] 0.2897959
```

Now list the `R` objects available to you:

```
ls()
```

```
## [1] "fem"     "salex"   "tab2by2"
```

Note that there are no `tab`, `a`, `b`, `c`, `d`, or `rr` objects.

The `tab2by2()` function is very limited. It only displays a table and calculates and displays a simple ratio. A more useful function would also calculate and display a confidence interval for the risk ratio. This is what we will do now. Use the `fix()` function to edit the `tab2by2()` function:

```
fix(tab2by2)
```

We can now edit this function to calculate and display a 95% confidence interval for the risk ratio.

```
function(exposure, outcome) {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci.rr <- exp(log(rr) - 1.96 * se.log.rr)
  uci.rr <- exp(log(rr) + 1.96 * se.log.rr)
  print(tab)
  print(rr)
  print(lci.rr)
  print(uci.rr)
}
```

Once you have made the changes shown above, check your work, save the file, and quit the editor. We should test our revised function:

```
tab2by2(EGGS, ILL)
```

which produces the following output:

```
##         outcome
## exposure  1  2
##        1 40  6
##        2 10 20
## [1] 2.608696
## [1] 1.553564
## [1] 4.38044
```

The function works but the output could be improved. Use the `fix()` function to edit the `tab2by2()` function:

```
function(exposure, outcome) {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci.rr <- exp(log(rr) - 1.96 * se.log.rr)
  uci.rr <- exp(log(rr) + 1.96 * se.log.rr)
  print(tab)
  cat("\nRR :", rr,
      "\n95% CI :", lci.rr, uci.rr, "\n")
}
```

Once you have made the changes shown above, save the file and quit the editor.

Now we can test our function again:

```
tab2by2(EGGS, ILL)
```

Which produces the following output:

```
##         outcome
## exposure  1  2
##        1 40  6
##        2 10 20
##
## RR : 2.608696
## 95% CI : 1.553564 4.38044
```

The `tab2by2()` function displays output but does not behave like a standard `R` function in the sense that you cannot save the results of the `tab2by2()` function into an object:

```
test2by2 <- tab2by2(EGGS, ILL)
```

```
##         outcome
## exposure  1  2
##        1 40  6
##        2 10 20
##
## RR : 2.608696
## 95% CI : 1.553564 4.38044
```

displays output but does not save anything in the `test2by2` object:

```
test2by2
```

```
## NULL
```

The returned value (`NULL`) means that `test2by2` is an empty object. We will not worry about this at the moment as the `tab2by2()` function is good-enough for our current purposes. In Exercise 6 we will explore how to make our own functions behave like standard `R` functions.

We will now add the calculation of the odds ratio and its 95% confidence interval to the `tab2by2()` function using the `fix()` function.

There are two ways of doing this. We could either calculate the odds ratio from the table and use (e.g.) the method of Woolf to calculate the confidence interval:

```
or <- (a / b) / (c / d)
se.log.or <- sqrt(1 / a + 1 / b + 1 / c + 1 / d)
lci.or <- exp(log(or) - 1.96 * se.log.or)
uci.or <- exp(log(or) + 1.96 * se.log.or)
cat("\nOR     :", or,
    "\n95% CI :", lci.or, uci.or, "\n")
```

or use the output of the `fisher.test()` function:

```
ft <- fisher.test(tab)
cat("\nOR     :", ft$estimate,
    "\n95% CI :", ft$conf.int, "\n")
```

Note that we can refer to components of a function's output using the same syntax as when we refer to columns in a data.frame (e.g. `ft$estimate` to examine the estimate of the odds ratio from the `fisher.test()` function stored in the object `ft`).

The names of elements in the output of a standard function such as `fisher.test()` can be found in the documentation or the help system. For example:

```
help(fisher.test)
```

Output elements are listed under the `Value` heading.

Revise the `tab2by2()` function to include the calculation of the odds ratio and the 95% confidence interval. The revised function will look something like this:

```
function(exposure, outcome) {
  tab <- table(exposure, outcome)
  a <- tab[1,1]
  b <- tab[1,2]
  c <- tab[2,1]
  d <- tab[2,2]
  rr <- (a / (a + b)) / (c / (c + d))
  se.log.rr <- sqrt((b / a) / (a + b) + (d / c) / (c + d))
  lci.rr <- exp(log(rr) - 1.96 * se.log.rr)
  uci.rr <- exp(log(rr) + 1.96 * se.log.rr)
  or <- (a / b) / (c / d)
  se.log.or <- sqrt(1 / a + 1 / b + 1 / c + 1 / d)
  lci.or <- exp(log(or) - 1.96 * se.log.or)
  uci.or <- exp(log(or) + 1.96 * se.log.or)
  ft <- fisher.test(tab)
  cat("\n")
  print(tab)

  cat("\nRelative Risk     :", rr,
      "\n95% CI            :", lci.rr, uci.rr, "\n")

  cat("\nSample Odds Ratio :", or,
      "\n95% CI            :", lci.or, uci.or, "\n")

  cat("\nMLE Odds Ratio    :", ft$estimate,
      "\n95% CI            :",  ft$conf.int, "\n\n")
}
```

Once you have made the changes shown above, check your work, save the file, and quit the editor.

Test the `tab2by2()` function when you have added the calculation of the odds ratio and its 95% confidence interval.

Now that we have a function that will calculate risk ratios and odds ratios with confidence intervals from a two- by-two table we can use it to analyse the `salex` data:

```
tab2by2(HAM, ILL)
tab2by2(BEEF, ILL)
tab2by2(EGGS, ILL)
tab2by2(MUSHROOM, ILL)
tab2by2(PEPPER, ILL)
tab2by2(PORKPIE, ILL)
tab2by2(PASTA, ILL)
tab2by2(RICE, ILL)
tab2by2(LETTUCE, ILL)
tab2by2(TOMATO, ILL)
tab2by2(COLESLAW, ILL)
tab2by2(CRISPS, ILL)
tab2by2(PEACHCAKE, ILL)
tab2by2(CHOCOLATE, ILL)
tab2by2(FRUIT, ILL)
tab2by2(TRIFLE, ILL)
tab2by2(ALMONDS, ILL)
```

```
##
##         outcome
## exposure  1  2
##        1 46 17
##        2  5  9
##
## Relative Risk     : 2.044444
## 95% CI            : 0.9964841 4.194501
##
## Sample Odds Ratio : 4.870588
## 95% CI            : 1.428423 16.60756
##
## MLE Odds Ratio    : 4.75649
## 95% CI            : 1.22777 20.82921
```

```
##
##         outcome
## exposure  1  2
##        1 45 22
##        2  6  4
##
## Relative Risk     : 1.119403
## 95% CI            : 0.6568821 1.907592
##
## Sample Odds Ratio : 1.363636
## 95% CI            : 0.3485746 5.334594
##
## MLE Odds Ratio    : 1.357903
## 95% CI            : 0.2547114 6.428414
```

```
##
##         outcome
## exposure  1  2
##        1 40  6
##        2 10 20
##
## Relative Risk     : 2.608696
## 95% CI            : 1.553564 4.38044
##
## Sample Odds Ratio : 13.33333
## 95% CI            : 4.240168 41.92706
##
## MLE Odds Ratio    : 12.74512
## 95% CI            : 3.762787 50.05419
```

```
##
##         outcome
## exposure  1  2
##        1 24  6
##        2 25 19
##
## Relative Risk     : 1.408
## 95% CI            : 1.028944 1.926697
##
```

```
## Sample Odds Ratio : 3.04
## 95% CI            : 1.037274 8.909506
##
## MLE Odds Ratio    : 2.995207
## 95% CI            : 0.9421008 10.7953

##
##         outcome
## exposure  1  2
##        1 24  3
##        2 23 22
##
## Relative Risk     : 1.73913
## 95% CI            : 1.26876 2.383882
##
## Sample Odds Ratio : 7.652174
## 95% CI            : 2.013718 29.07844
##
## MLE Odds Ratio    : 7.448216
## 95% CI            : 1.861728 44.12015

##
##         outcome
## exposure  1  2
##        1 21  9
##        2 29 17
##
## Relative Risk     : 1.110345
## 95% CI            : 0.8044752 1.532509
##
## Sample Odds Ratio : 1.367816
## 95% CI            : 0.5113158 3.659032
##
## MLE Odds Ratio    : 1.362228
## 95% CI            : 0.4636016 4.190667

##
##         outcome
## exposure  1  2
##        1 25  3
##        2 26 23
##
## Relative Risk     : 1.682692
## 95% CI            : 1.255392 2.255433
##
## Sample Odds Ratio : 7.371795
## 95% CI            : 1.964371 27.66451
##
## MLE Odds Ratio    : 7.195422
## 95% CI            : 1.829867 42.07488
```

```
##
##         outcome
## exposure  1  2
##        1 28  4
##        2 23 22
##
## Relative Risk     : 1.711957
## 95% CI            : 1.250197 2.344268
##
## Sample Odds Ratio : 6.695652
## 95% CI            : 2.017327 22.22335
##
## MLE Odds Ratio    : 6.532868
## 95% CI            : 1.852297 29.84928
```

```
##
##         outcome
## exposure  1  2
##        1 28  1
##        2 23 25
##
## Relative Risk     : 2.014993
## 95% CI            : 1.488481 2.727744
##
## Sample Odds Ratio : 30.43478
## 95% CI            : 3.826938 242.041
##
```

```
## MLE Odds Ratio    : 29.32825
## 95% CI            : 4.161299 1284.306

##
##         outcome
## exposure  1  2
##        1 29  9
##        2 22 17
##
## Relative Risk     : 1.352871
## 95% CI            : 0.974698 1.877771
##
## Sample Odds Ratio : 2.489899
## 95% CI            : 0.9347213 6.632562
##
## MLE Odds Ratio    : 2.459981
## 95% CI            : 0.8467562 7.558026

##
##         outcome
## exposure  1  2
##        1 29  3
##        2 21 23
##
## Relative Risk     : 1.89881
## 95% CI            : 1.366876 2.63775
##
## Sample Odds Ratio : 10.5873
## 95% CI            : 2.806364 39.9417
##
## MLE Odds Ratio    : 10.26269
## 95% CI            : 2.600771 60.35431

##
##         outcome
## exposure  1  2
##        1 21 10
##        2 30 16
##
```

```
## Relative Risk     : 1.03871
## 95% CI            : 0.7529065 1.433004
##
## Sample Odds Ratio : 1.12
## 95% CI            : 0.4258139 2.945888
##
## MLE Odds Ratio    : 1.118358
## 95% CI            : 0.3858206 3.340535

##
##         outcome
## exposure  1  2
##        1  2  2
##        2 49 24
##
## Relative Risk     : 0.744898
## 95% CI            : 0.27594 2.010846
##
## Sample Odds Ratio : 0.4897959
## 95% CI            : 0.06497947 3.691936
##
## MLE Odds Ratio    : 0.4947099
## 95% CI            : 0.03393887 7.209143

##
##         outcome
## exposure  1  2
##        1 12  2
##        2 38 24
##
## Relative Risk     : 1.398496
## 95% CI            : 1.045064 1.871456
##
## Sample Odds Ratio : 3.789474
## 95% CI            : 0.7791326 18.43089
##
## MLE Odds Ratio    : 3.733535
## 95% CI            : 0.7318646 37.28268
```

```
##
##         outcome
## exposure  1  2
##        1  1  4
##        2 49 22
##
## Relative Risk     : 0.2897959
## 95% CI            : 0.04985828 1.684408
##
## Sample Odds Ratio : 0.1122449
## 95% CI            : 0.01185022 1.06318
##
## MLE Odds Ratio    : 0.1157141
## 95% CI            : 0.002240848 1.256134

##
##         outcome
## exposure  1  2
##        1 19  5
##        2 32 21
##
## Relative Risk     : 1.311198
## 95% CI            : 0.9718621 1.769016
##
## Sample Odds Ratio : 2.49375
## 95% CI            : 0.8067804 7.708156
##
## MLE Odds Ratio    : 2.465794
## 95% CI            : 0.7363311 9.778463

##
##         outcome
## exposure  1  2
##        1  3  3
##        2 38 19
##
## Relative Risk     : 0.75
## 95% CI            : 0.3300089 1.7045
##
```

```
## Sample Odds Ratio : 0.5
## 95% CI            : 0.09203498 2.716358
##
## MLE Odds Ratio    : 0.505905
## 95% CI            : 0.06170211 4.141891
```

Make a note of any positive associations (i.e. with a risk ratio > 1 with a 95% confidence intervals that does not include one). We will use these for the next exercise when we will use logistic regression to analyse this data.

Save the `tab2by2()` function:

```
save(tab2by2, file = "tab2by2.r")
```

We can now quit `R`:

```
q()
```

For this exercise there is no need to save the workspace image so click the **No** or **Don't Save** button (GUI) or enter `n` when prompted to save the workspace image (terminal).

5.1 Summary

- `R` objects contain information that can be examined and manipulated.
- `R` can be extended by writing new functions.
- New functions can perform simple or complex data analysis.
- New functions can be composed of parts of existing function.
- New functions can be saved and used in subsequent `R` sessions.
- Objects defined within functions are local to that function and only exist while that function is being used. This means that you can re-use meaningful names within functions without them interfering with each other.

Chapter 6

Extending R with packages

R has no built-in functions for survival analysis but, because it is an extensible system, survival analysis is available as an add-in package. You can find a list of add-in packages at the R website.

http://www.r-project.org/

Add-in packages are installed from the Internet. There are a series of R functions that enable you to download and install add-in packages.

The `survival` package adds functions to R that enable it to analyse survival data. This package may be downloaded and installed using `install.packages("survival")` or from the `Packages` or `Packages & Data` menu if you are using a GUI version of R.

Packages are loaded into R as they are needed using the `library()` function. Start R and load the `survival` package:

```
library(survival)
```

Before we go any further we should retrieve a dataset:

```
ca <- read.table("data/ca.dat", header = TRUE)
attach(ca)
```

The columns in this dataset on the survival of cancer patients in two different treatment groups are as follows:

time	Survival or censoring time (months)
status	Censoring status (1=dead, 0=censored)
group	Treatment group (1 / 2)

We next need to create a `survival` object from the `time` and `status` variables using the `Surv()` function:

```
response <- Surv(time, status)
```

We can then specify the model for the survival analysis. In this case we state that survival (`response`) is dependent upon the treatment `group`:

```
ca.surv <- survfit(response ~ group)
```

The `summary()` function applied to a `survfit` object lists the survival probabilities at each time point with 95% confidence intervals:

```
summary(ca.surv)
```

```
## Call: survfit(formula = response ~ group)
##
##                 group=1
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     8     22       1    0.955  0.0444       0.8714        1.000
##     9     21       1    0.909  0.0613       0.7966        1.000
##    13     19       1    0.861  0.0744       0.7270        1.000
##    14     17       1    0.811  0.0856       0.6591        0.997
##    18     16       1    0.760  0.0940       0.5963        0.968
##    19     15       1    0.709  0.1005       0.5373        0.936
##    21     14       1    0.659  0.1053       0.4814        0.901
##    23     13       1    0.608  0.1087       0.4282        0.863
##    30     10       1    0.547  0.1136       0.3643        0.822
##    31      9       1    0.486  0.1161       0.3046        0.776
##    32      8       1    0.426  0.1164       0.2489        0.727
##    34      7       1    0.365  0.1146       0.1971        0.675
##    48      5       1    0.292  0.1125       0.1371        0.621
##    56      3       1    0.195  0.1092       0.0647        0.585
##
```

```
##                  group=2
##  time n.risk n.event survival std.err lower 95% CI upper 95% CI
##     4     24       1   0.9583  0.0408      0.88163        1.000
##     5     23       2   0.8750  0.0675      0.75221        1.000
##     6     21       1   0.8333  0.0761      0.69681        0.997
##     7     20       1   0.7917  0.0829      0.64478        0.972
##     8     19       2   0.7083  0.0928      0.54795        0.916
##     9     17       1   0.6667  0.0962      0.50240        0.885
##    11     16       1   0.6250  0.0988      0.45845        0.852
##    12     15       1   0.5833  0.1006      0.41598        0.818
##    21     12       1   0.5347  0.1033      0.36614        0.781
##    23     11       1   0.4861  0.1047      0.31866        0.742
##    27     10       1   0.4375  0.1049      0.27340        0.700
##    28      9       1   0.3889  0.1039      0.23032        0.657
##    30      8       1   0.3403  0.1017      0.18945        0.611
##    32      7       1   0.2917  0.0981      0.15088        0.564
##    33      6       1   0.2431  0.0930      0.11481        0.515
##    37      5       1   0.1944  0.0862      0.08157        0.464
##    41      4       2   0.0972  0.0650      0.02624        0.360
##    43      2       1   0.0486  0.0473      0.00722        0.327
##    45      1       1   0.0000     NaN           NA           NA
```

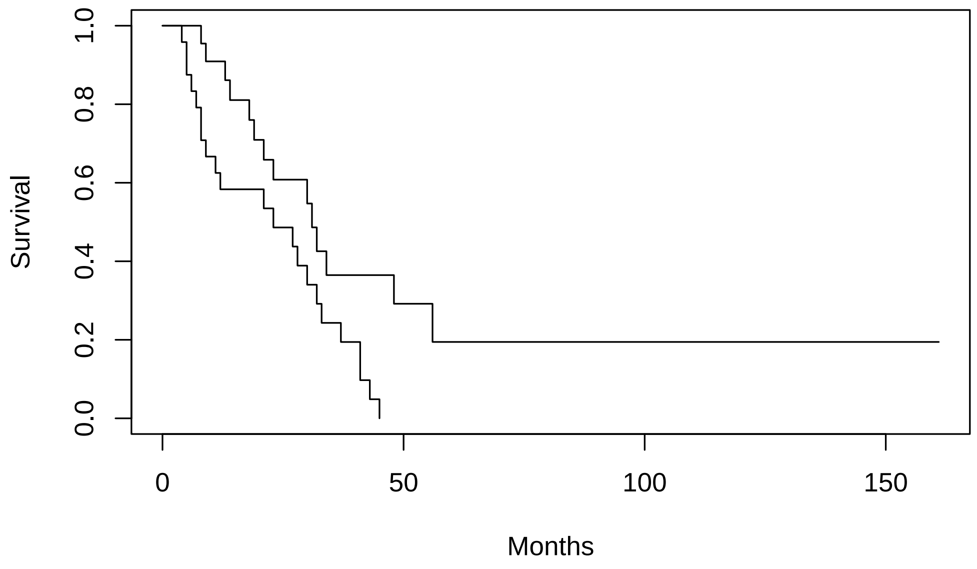
Printing the `ca.surv` object provides another view of the results:

```
ca.surv
```

```
## Call: survfit(formula = response ~ group)
##
##          n events median 0.95LCL 0.95UCL
## group=1 22     14     31      21      NA
## group=2 24     22     23      11      37
```

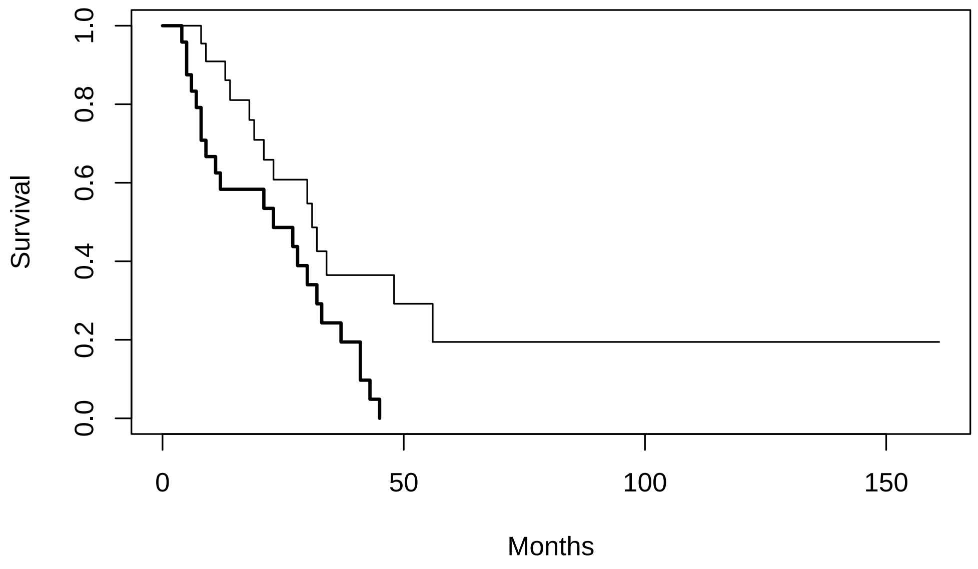
The `plot()` function with a `survfit` object displays the survival curves:

```
plot(ca.surv, xlab = "Months", ylab = "Survival")
```



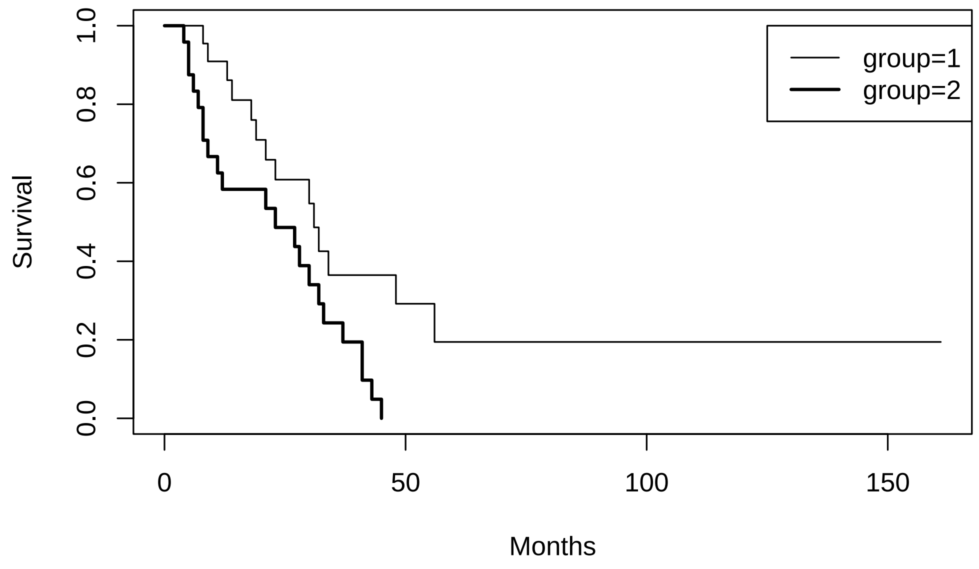
We can make it easier to distinguish between the two lines by specifying a width for each line using thelwd parameter of the `plot()` function:

```
plot(ca.surv, xlab = "Months", ylab = "Survival", lwd = c(1, 2))
```



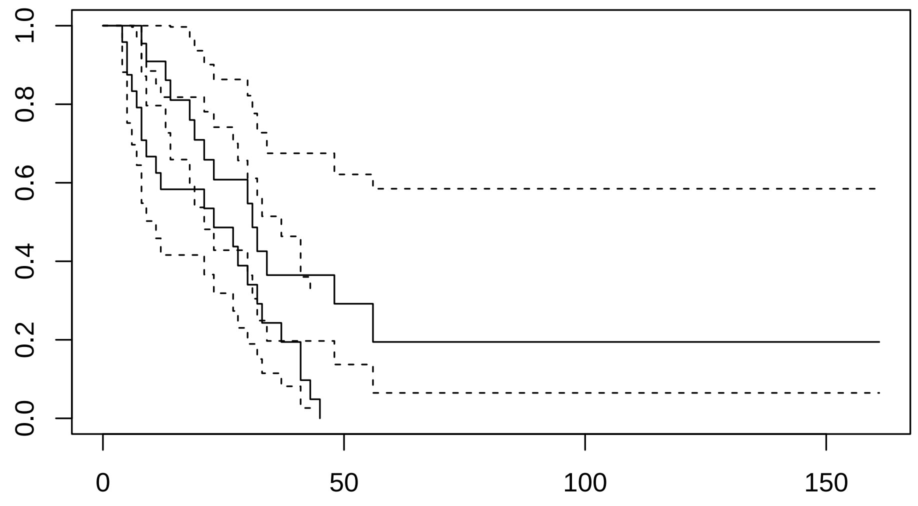
It would also be useful to add a legend:

```
legend(125, 1, names(ca.surv$strata), lwd = c(1, 2))
```



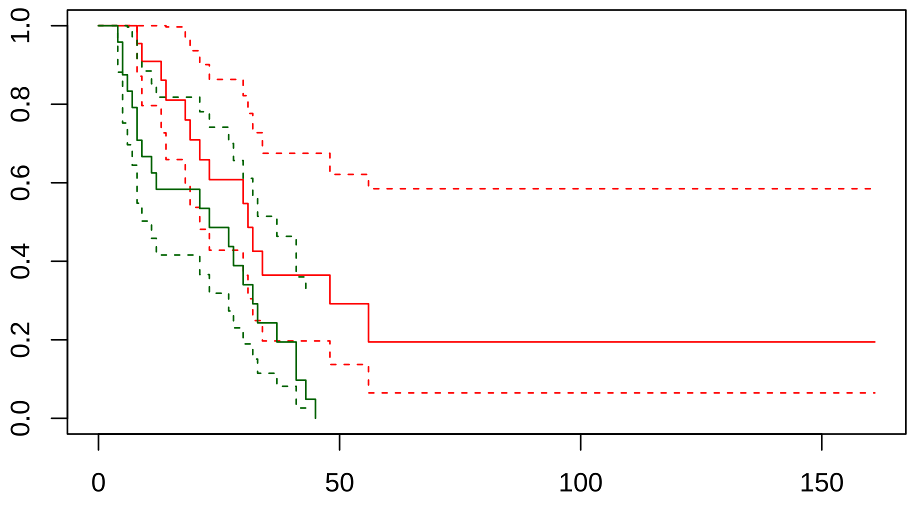
If there is only one survival curve to plot then plotting a `survfit` object will plot the survival curve with 95% confidence limits. You can specify that confidence limits should be plotted when there is more than one survival curve but the results can be disappointing:

```
plot(ca.surv, conf.int = TRUE)
```



Plots can be improved by specifying different colours for each curve:

```r
plot(ca.surv, conf.int = TRUE, col = c("red", "darkgreen"))
```



We can perform a formal test of the two survival times using the `survdiff()` function:

```
survdiff(response ~ group)
```

```
## Call:
## survdiff(formula = response ~ group)
##
##          N Observed Expected (O-E)^2/E (O-E)^2/V
## group=1 22       14     21.1      2.38      6.26
## group=2 24       22     14.9      3.36      6.26
##
##  Chisq= 6.3  on 1 degrees of freedom, p= 0.01
```

We can now quit `R`:

```
q()
```

For this exercise there is no need to save the workspace image so click the **No** or **Don't Save** button (GUI) or enter `n` when prompted to save the workspace image (terminal).

6.1 Summary

- `R` can be extended by adding additional packages. Some packages are included with the standard `R` installation but many others are available and may be downloaded from the Internet.
- You can find a list of add-in packages at the `R` website: http://www.r-project.org/
- Packages may also be downloaded and installed from this site using the `install.packages()` function or from the **Packages** or **Packages & Data** menu if you are using a GUI version of `R`.
- Packages are loaded into `R` as they are needed using the `library()` function. You can use the `search()` function to display a list of loaded packages and attached data.frames.

Chapter 7

Writing functions

In the earlier stages of this training, we worked a lot with data on weight and height to calculate body mass index (see https://oxfordihtm.github.io/seychelles-statistics-training/week1-day2.html). As a refresher, body mass index is calculated as follows:

$$\text{Body mass index} \ = \ \frac{weight\ (kgs)}{height\ (m)^2}$$

For this topic on writing functions in R, we will continue using BMI as an example to explore and demonstrate how we can create our own functions in R.

Let's say for example that you have been doing a research on children aged 11 years and older in 3 schools and you have collected the following data:

School 1516

```
school1516
```

```
##     school sex ageMonths weight height
## 427   1516   1       138   24.5  126.0
## 428   1516   1       150   28.3  136.3
## 429   1516   1       162   32.2  143.5
## 430   1516   1       162   32.7  143.5
## 431   1516   1       150   28.6  137.0
## 432   1516   2       138   26.5  134.0
## 433   1516   1        150   29.9  139.2
## 434   1516   1        150   30.0  139.5
## 435   1516   1        162   34.0  148.0
## 436   1516   1        138   25.4  135.7
## 437   1516   1        150   32.3  143.0
## 438   1516   2        174   38.3  153.5
## 439   1516   2        162   41.6  151.0
## 440   1516   1        150   30.7  145.0
## 441   1516   2        186   46.8  155.2
## 442   1516   1        186   46.6  163.4
## 443   1516   1        150   33.5  145.5
## 444   1516   1        186   47.0  164.0
## 445   1516   1        174   41.1  159.5
## 446   1516   2        162   39.1  152.2
## 447   1516   2        174   40.9  155.5
## 448   1516   2        162   39.7  153.0
## 449   1516   2        162   40.9  153.2
## 450   1516   1        150   34.2  147.5
## 451   1516   2        150   41.8  149.4
## 452   1516   1        138   28.0  141.5
## 453   1516   1        138   30.0  142.0
## 454   1516   1        138   33.1  142.0
## 455   1516   1        186   46.1  167.5
## 456   1516   1        150   36.2  149.0
## 457   1516   2        162   47.4  156.0
## 458   1516   1        150   30.3  150.2
## 459   1516   2        150   36.4  152.1
## 460   1516   2        150   36.4  155.0
## 461   1516   2        150   44.1  155.0
## 462   1516   2        162   42.3  160.1
## 463   1516   2        179   50.4  163.5
## 464   1516   1        150   37.6  155.0
## 465   1516   2        138   36.0  154.5
## 466   1516   2        138   46.1  156.0
```

School 1522

```
school1522
```

```
##     school sex ageMonths weight height
## 646   1522   1       203   30.6  140.5
## 647   1522   1       174   30.8  140.0
## 648   1522   1       162   29.3  136.3
## 649   1522   1       150   24.0  132.0
## 650   1522   1       150   28.1  132.1
## 651   1522   2       150   27.2  134.9
## 652   1522   1       162   34.2  139.2
## 653   1522   1       150   25.5  134.2
## 654   1522   1       138   24.6  129.0
## 655   1522   1       174   36.4  147.5
## 656   1522   1       150   28.7  137.5
## 657   1522   1       186   45.8  155.6
## 658   1522   1       174   36.3  151.6
## 659   1522   1       150   31.0  139.5
## 660   1522   1       138   29.0  134.3
## 661   1522   1       179   38.3  155.5
## 662   1522   2       138   31.3  138.4
## 663   1522   1       162   36.5  148.8
## 664   1522   1       155   36.8  145.2
## 665   1522   1       138   28.3  136.8
## 666   1522   1       138   26.8  137.3
## 667   1522   2       138   32.6  141.4
## 668   1522   2       138   31.9  143.0
## 669   1522   1       174   42.6  160.7
## 670   1522   2       198   57.8  158.0
## 671   1522   2       162   43.9  153.5
## 672   1522   2       150   35.1  150.6
## 673   1522   2       186   52.6  159.6
## 674   1522   2       150   45.1  152.8
## 675   1522   2       138   34.6  147.2
## 676   1522   2       150   45.3  153.1
## 677   1522   1       186   51.8  170.2
## 678   1522   2       150   57.1  154.2
## 679   1522   2       138   33.5  149.2
## 680   1522   1       150   36.3  154.1
## 681   1522   1       174   44.0  169.1
## 682   1522   2       150   44.5  158.3
## 683   1522   2       150   51.5  159.1
## 684   1522   2       138   47.4  157.8
## 685   1522   2       138   36.8  158.5
## 686   1522   2       138   52.0  161.0
```

School 1525

```
school1525
```

```
##     school sex ageMonths weight height
## 752   1525   1       186   26.2    137
## 753   1525   1       186   32.7    138
## 754   1525   1       150   25.9    130
## 755   1525   1       162   30.4    137
## 756   1525   2       138   24.4    129
## 757   1525   2       138   23.8    130
## 758   1525   1       150   26.1    133
## 759   1525   1       150   26.4    135
## 760   1525   1       174   35.1    148
## 761   1525   1       162   28.7    142
## 762   1525   1       150   28.0    136
## 763   1525   1       174   34.0    149
## 764   1525   1       186   40.6    155
## 765   1525   2       150   35.8    142
## 766   1525   1       150   35.4    140
## 767   1525   2       138   27.8    137
## 768   1525   2       138   28.2    137
## 769   1525   2       138   29.7    139
## 770   1525   2       138   30.9    139
## 771   1525   1       138   28.2    137
## 772   1525   2       138   26.2    140
## 773   1525   2       138   26.6    140
## 774   1525   1       138   27.2    138
## 775   1525   2       138   27.0    141
## 776   1525   1       150   31.3    145
## 777   1525   2       162   33.9    152
## 778   1525   2       162   42.0    153
## 779   1525   2       185   38.3    157
## 780   1525   2       138   31.0    145
## 781   1525   2       138   32.3    145
## 782   1525   1       139   35.1    144
## 783   1525   2       150   36.4    152
## 784   1525   2       138   32.7    147
## 785   1525   1       174   44.9    166
## 786   1525   2       138   32.2    148
## 787   1525   2       138   36.4    148
## 788   1525   1       138   31.4    146
## 789   1525   2       138   45.0    149
## 790   1525   2       162   49.4    160
## 791   1525   2       138   34.3    150
## 792   1525   1       138   30.0    148
## 793   1525   2       150   37.0    156
## 794   1525   2       162   52.2    165
## 795   1525   2       138   42.9    158
```

In this dataset, the units of the height measurement is in centimetres.

Using what we have learned earlier on calculating BMI using R, I can perform the following R commands to get the BMI for each child in each of the schools:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2

## Calculate BMI for children in school 1516
school1522$weight / (school1516$height / 100) ^ 2

## Calculate BMI for children in school 1516
school1525$weight / (school1516$height / 100) ^ 2
```

Because the commands are repetitive, I can easily copy and paste my initial line of code to calculate BMI for children in school 1516 and then just change the object names accordingly to calculate the BMI for children in the two other schools.

When I run these lines of code, I get the following results:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2
```

```
##  [1] 15.43210 15.23333 15.63695 15.87976 15.23789 14.75830 15.43095 15.4160
##  [9] 15.52228 13.79349 15.79539 16.25481 18.24481 14.60166 19.42954 17.4534
## [17] 15.82409 17.47472 16.15550 16.87903 16.91463 16.95929 17.42632 15.7196
## [25] 18.72730 13.98444 14.87800 16.41539 16.43128 16.30557 19.47732 13.4308
## [33] 15.73414 15.15088 18.35588 16.50280 18.85363 15.65036 15.08153 18.9431
```

```
## Calculate BMI for children in school 1516
school1522$weight / (school1516$height / 100) ^ 2
```

```
## Warning in school1522$weight/(school1516$height/100)^2: longer object lengt
## not a multiple of shorter object length
```

```
##  [1] 19.27438 16.57903 14.22865 11.65487 14.97150 15.14814 17.65012 13.1036
##  [9] 11.23083 19.76704 14.03492 19.43787 15.92035 14.74435 12.03967 14.3448
## [17] 14.78490 13.57079 14.46527 12.21679 11.08343 13.92627 13.59168 19.5805
## [25] 25.89564 21.92561 17.40726 26.08609 16.07485 15.58488 18.61440 22.9609
## [33] 24.68185 13.94381 15.10926 17.16604 16.64656 21.43600 19.85735 15.1216
## [41] 32.75384
```

```
## Calculate BMI for children in school 1516
school1525$weight / (school1516$height / 100) ^ 2
```

```
## Warning in school1525$weight/(school1516$height/100)^2: longer object lengt
## not a multiple of shorter object length
```

```
##  [1] 16.50290 17.60176 12.57755 14.76284 13.00016 13.25462 13.46983 13.5661
##  [9] 16.02447 15.58555 13.69260 14.42986 17.80624 17.02735 14.69670 10.4121
## [17] 13.32058 11.04253 12.14611 12.17362 10.83529 11.36315 11.58914 12.4102
## [25] 14.02307 16.93116 20.82920 18.99425 11.04923 14.54889 14.42308 16.1347
## [33] 14.13479 18.68887 13.40271 14.20099 11.74611 18.73049 20.69522 14.0943
## [41] 18.89645 19.91636 25.34934 20.83308
```

The calculation for the BMI of children in school 1516 seems to have completed without issues and a vector of BMI results have been produced. However, for school 1522 and school 1525, there is a warning saying:

```
## Warning in school1522$weight/(school1516$height)^2: longer object length
## of shorter object length
```

Although a result has been provided, the warning gives me an indication that someting is not quite right with my calculation and when I inspect further, I notice that in my formula for school 1522 and for school 1525, my denominator is still using data for school 1516 and this is most likely what is causing the warning message.

So, to correct this I go back to my lines of code and edit the denominators for school 1522 and school 1525 as follows:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2

## Calculate BMI for children in school 1516
school1522$weight / (school1522$height / 100) ^ 2

## Calculate BMI for children in school 1516
school1525$weight / (school1525$height / 100) ^ 2
```

which gives me:

```
## Calculate BMI for children in school 1516
school1516$weight / (school1516$height / 100) ^ 2
```

```
##  [1] 15.43210 15.23333 15.63695 15.87976 15.23789 14.75830 15.43095 15.41604
##  [9] 15.52228 13.79349 15.79539 16.25481 18.24481 14.60166 19.42954 17.45347
## [17] 15.82409 17.47472 16.15550 16.87903 16.91463 16.95929 17.42632 15.71962
## [25] 18.72730 13.98444 14.87800 16.41539 16.43128 16.30557 19.47732 13.43083
## [33] 15.73414 15.15088 18.35588 16.50280 18.85363 15.65036 15.08153 18.94313
```

```
## Calculate BMI for children in school 1516
school1522$weight / (school1522$height / 100) ^ 2
```

```
##  [1] 15.50132 15.71429 15.77161 13.77410 16.10277 14.94669 17.65012 14.15908
##  [9] 14.78277 16.73082 15.18017 18.91674 15.79459 15.92991 16.07852 15.83937
## [17] 16.34076 16.48493 17.45479 15.12217 14.21653 16.30492 15.59978 16.49597
## [25] 23.15334 18.63150 15.47594 20.65000 19.31656 15.96837 19.32626 17.88178
## [33] 24.01416 15.04898 15.28626 15.38741 17.75817 20.34543 19.03550 14.64837
## [41] 20.06095
```

```
## Calculate BMI for children in school 1516
school1525$weight / (school1525$height / 100) ^ 2
```

```
##  [1] 13.95919 17.17076 15.32544 16.19692 14.66258 14.08284 14.75493 14.4856
##  [9] 16.02447 14.23329 15.13841 15.31463 16.89906 17.75441 18.06122 14.8116
## [17] 15.02477 15.37188 15.99296 15.02477 13.36735 13.57143 14.28271 13.5808
## [25] 14.88704 14.67278 17.94182 15.53816 14.74435 15.36266 16.92708 15.7548
## [33] 15.13258 16.29409 14.70051 16.61797 14.73072 20.26936 19.29687 15.2444
## [41] 13.69613 15.20381 19.17355 17.18475
```

I now do not get the warning message and the expected length of BMI values for each school has now been produced.

From this short example above, we realise how tedious a task it is to type in the code above every time we need to calculate BMI. Also, it becomes even challenging to debug issues with the code because we have to review and edit (as needed) each iteration of the calculation to see where it may have gone wrong (especially when doing a cut and paste approach).

It would be better (and easier) to have a function that calculates and displays the BMI values automatically. Fortunately, `R` allows us to do just that.

The `function()` function allows us to create new functions in `R` with the following generic syntax:

```
function_name <- function(argument1, argument2, ...) {
  ## Your code here
}
```

Using this template/generic syntax, we apply it to create a function called `calculate_bmi` as follows:

```
calculate_bmi <- function(weight, height) {
  weight / height ^ 2
}
```

We now have a function for calculating and outputing BMI values.

Let us now test it with our 3 sets of data:

School 1516

```
calculate_bmi(
  weight = school1516$weight,
  height = school1516$height / 100
)
```

```
##  [1] 15.43210 15.23333 15.63695 15.87976 15.23789 14.75830 15.43095 15.41604
##  [9] 15.52228 13.79349 15.79539 16.25481 18.24481 14.60166 19.42954 17.45347
## [17] 15.82409 17.47472 16.15550 16.87903 16.91463 16.95929 17.42632 15.71962
## [25] 18.72730 13.98444 14.87800 16.41539 16.43128 16.30557 19.47732 13.43083
## [33] 15.73414 15.15088 18.35588 16.50280 18.85363 15.65036 15.08153 18.94313
```

School 1522

```
calculate_bmi(
  weight = school1522$weight,
  height = school1522$height / 100
)
```

```
##  [1] 15.50132 15.71429 15.77161 13.77410 16.10277 14.94669 17.65012 14.15908
##  [9] 14.78277 16.73082 15.18017 18.91674 15.79459 15.92991 16.07852 15.83937
## [17] 16.34076 16.48493 17.45479 15.12217 14.21653 16.30492 15.59978 16.49597
## [25] 23.15334 18.63150 15.47594 20.65000 19.31656 15.96837 19.32626 17.88178
## [33] 24.01416 15.04898 15.28626 15.38741 17.75817 20.34543 19.03550 14.64837
## [41] 20.06095
```

School 1525

```
calculate_bmi(
  weight = school1525$weight,
  height = school1525$height / 100
)
```

```
##  [1] 13.95919 17.17076 15.32544 16.19692 14.66258 14.08284 14.75493 14.48560
##  [9] 16.02447 14.23329 15.13841 15.31463 16.89906 17.75441 18.06122 14.81166
## [17] 15.02477 15.37188 15.99296 15.02477 13.36735 13.57143 14.28271 13.58081
## [25] 14.88704 14.67278 17.94182 15.53816 14.74435 15.36266 16.92708 15.75485
## [33] 15.13258 16.29409 14.70051 16.61797 14.73072 20.26936 19.29687 15.24444
## [41] 13.69613 15.20381 19.17355 17.18475
```

In our example here, the `calculate_bmi()` function helped a little bit in making the code to calculate BMI for each student in each school more efficient. But the efficiency that functions provide become more evident when you need to make more complex operations. For example, what if you need to get the mean BMI for students in each school? Without a function, we will have to do the following script for each school:

School 1516

```
## Calculate BMI for children in school 1516
bmi_school1516 <- school1516$weight / (school1516$height / 100) ^ 2

## Get the mean BMI for children in school 1516
mean_bmi_school1516 <- mean(bmi_school1516)

mean_bmi_school1516
```

```
## [1] 16.28491
```

School 1522

```
## Calculate BMI for children in school 1522
bmi_school1522 <- school1522$weight / (school1522$height / 100) ^ 2

## Get the mean BMI for children in school 1522
mean_bmi_school1522 <- mean(bmi_school1522)

mean_bmi_school1522
```

```
## [1] 16.89955
```

School 1525

```
## Calculate BMI for children in school 1525
bmi_school1525 <- school1525$weight / (school1525$height) ^ 2

## Get the mean BMI for children in school 1525
mean_bmi_school1525 <- mean(bmi_school1525)

mean_bmi_school1525
```

```
## [1] 0.001564695
```

As the operations/calculations we want to perform become more complex, the copy and paste method becomes more and more tedious. With the function approach, we can use the following:

```
calculate_mean_bmi <- function(weight, height) {
  bmi <- weight / height ^ 2
```

```r
  mean_bmi <- mean(bmi)

  return(mean_bmi)
}
```

Applying the function to the datasets, we get:

School 1516

```r
calculate_mean_bmi(
  weight = school1516$weight,
  height = school1516$height / 100
)
```

```
## [1] 16.28491
```

School 1522

```r
calculate_mean_bmi(
  weight = school1522$weight,
  height = school1522$height / 100
)
```

```
## [1] 16.89955
```

School 1525

```r
calculate_mean_bmi(
  weight = school1525$weight,
  height = school1525$height / 100
)
```

```
## [1] 15.64695
```

Chapter 8

Introduction to git and GitHub

8.1 All about git

`git` is a version control system for software development. It allows developers to keep track of changes made to their code and collaborate with other developers on a project. `git` also allows for easy rollbacks and branch management. It is widely used in the software industry and is considered one of the best version control systems available.

`git` was developed by Linus Torvalds in 2005. He created `git` as a replacement for the proprietary version control system he was using at the time. The development of `git` was driven by the need for a distributed version control system, which allows multiple developers to work on a project simultaneously, without the need for a central server. Linus Torvalds is also known for creating the Linux operating system kernel.

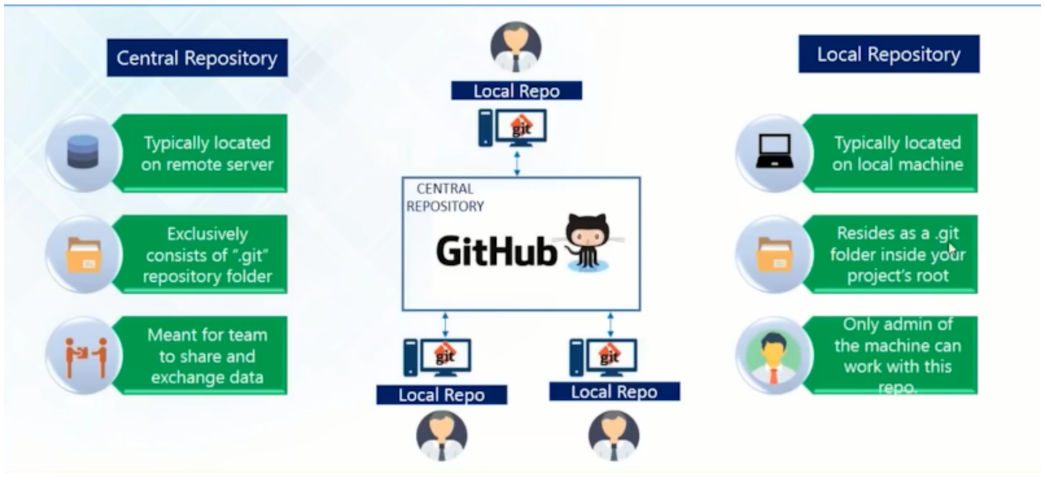
To use `git` in your machines, you will need to install it as described in the previous chapter (Chapter 2).

8.2 All about GitHub

GitHub is a web-based platform that provides hosting for software development and a community of developers to collaborate, share and learn from each other. It is built on top of `git`, which is the version control system used for managing and tracking changes to the code. Developers can use GitHub to store and manage their code, collaborate with other developers, and track and manage issues and bugs. It also provides tools for code review, project management, and documentation. It is widely used by developers and organizations to host and share code, as well as to build and maintain open-source software.

GitHub is not a software you need to install. Rather it is a remote or cloud-based server that holds its users' code versioned using the `git` version control system and to which a user's local, git-versioned code syncs/communicates with.

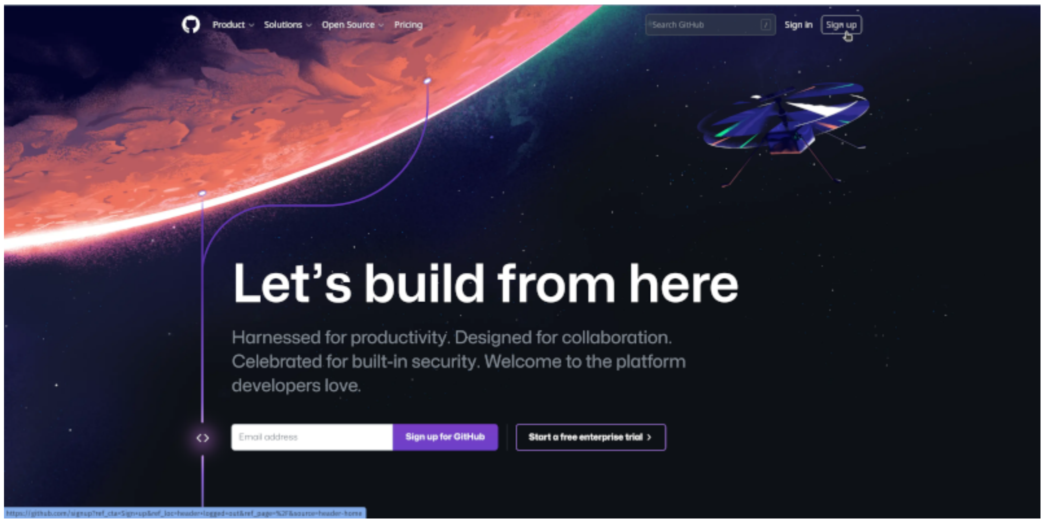
A good illustration of the `git` and `GitHub` relationship can be viewed below:


Central Repository
Typically located on remote server
Exclusively consists of ".git" repository folder
Meant for team to share and exchange data
Local Repo
git
CENTRAL REPOSITORY
GitHub
Local Repo
Local Repo
Local Repository
Typically located on local machine
Resides as a .git folder inside your project's root
Only admin of the machine can work with this repo.

To use GitHub, you will need create an account at https://github.com.

8.2.1 Creating a GitHub account

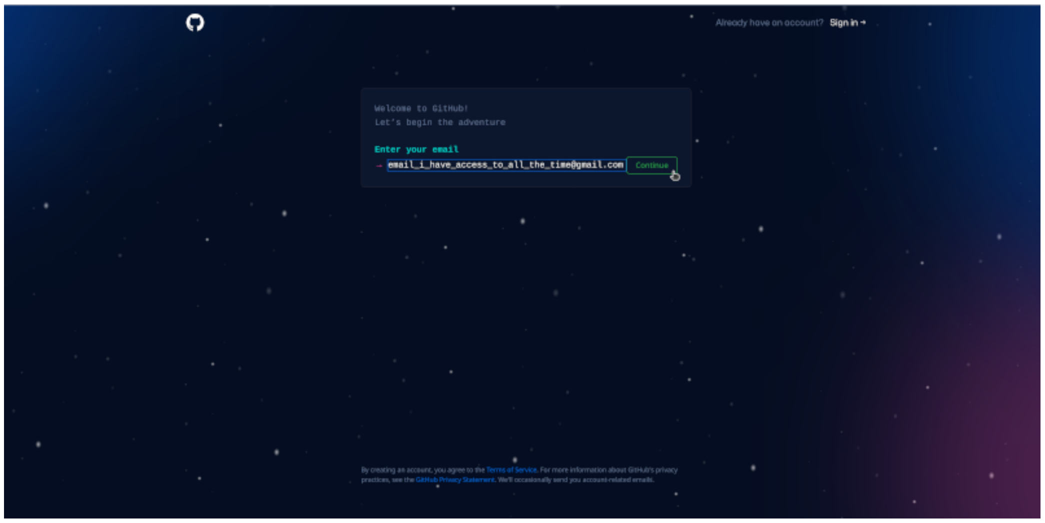
1. Head to https://github.com
2. On the upper right hand corner of the page, click on **Sign-up** button



Let's build from here

Harnessed for productivity. Designed for collaboration. Celebrated for built-in security. Welcome to the platform developers love.

3. You will be then prompted to provide an email address to register your account with.



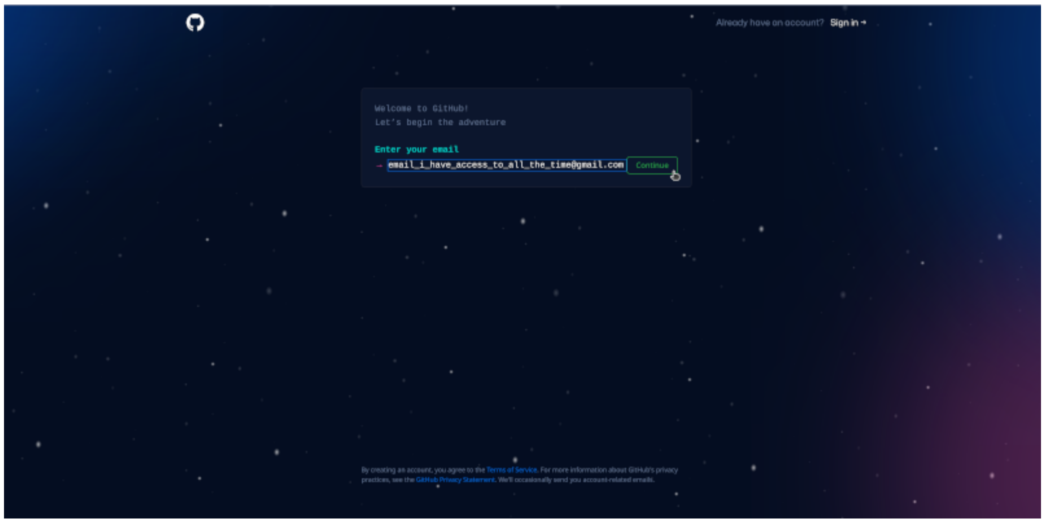
Welcome to GitHub!
Let's begin the adventure

Enter your email
→ email_i_have_access_to_all_the_time@gmail.com

By creating an account, you agree to the Terms of Service. For more information about GitHub's privacy practices, see the GitHub Privacy Statement. We'll occasionally send you account-related emails.

With regard to the email address to use for creating a GitHub account, a best practice recommendation is to use an email address that you will have access to all the time. Email addresses such as those for school (if you are a student) or for your current work may not always be the best email address to use as these email addresses tend to be time-limited (i.e., you lose the email address once you graduate or once you leave your current work).

4. You will then be prompted for a password.



Welcome to GitHub!
Let's begin the adventure

Enter your email
→ email_i_have_access_to_all_the_time@gmail.com

By creating an account, you agree to the Terms of Service. For more information about GitHub's privacy practices, see the GitHub Privacy Statement. We'll occasionally send you account-related emails.

5. Then follow all other prompts after this including confirmation of your email and creating a GitHub username.

With regard to creating/selecting a GitHub username, following are some best practice recommendations:

- Incorporate your actual name! People like to know who they're dealing with. Also makes your username easier for people to guess or remember.
- Pick a username you will be comfortable revealing to your future boss and/or to future collaborators.
- Shorter is better than longer.
- Be as unique as possible in as few characters as possible. In some settings GitHub auto-completes or suggests usernames.
- Make it timeless. Don't highlight your current university, employer, or place of residence.
- Avoid the use of upper vs. lower case to separate words. We highly recommend all lowercase. GitHub treats usernames in a case insensitive way, but using all lowercase is kinder to people doing downstream regular expression work with usernames, in various languages. A better strategy for word separation is to use a hyphen `-`.
- You can change your username later, but better to get this right the first time.

https://help.github.com/articles/changing-your-github-username/

https://help.github.com/articles/what-happens-when-i-change-my-username/

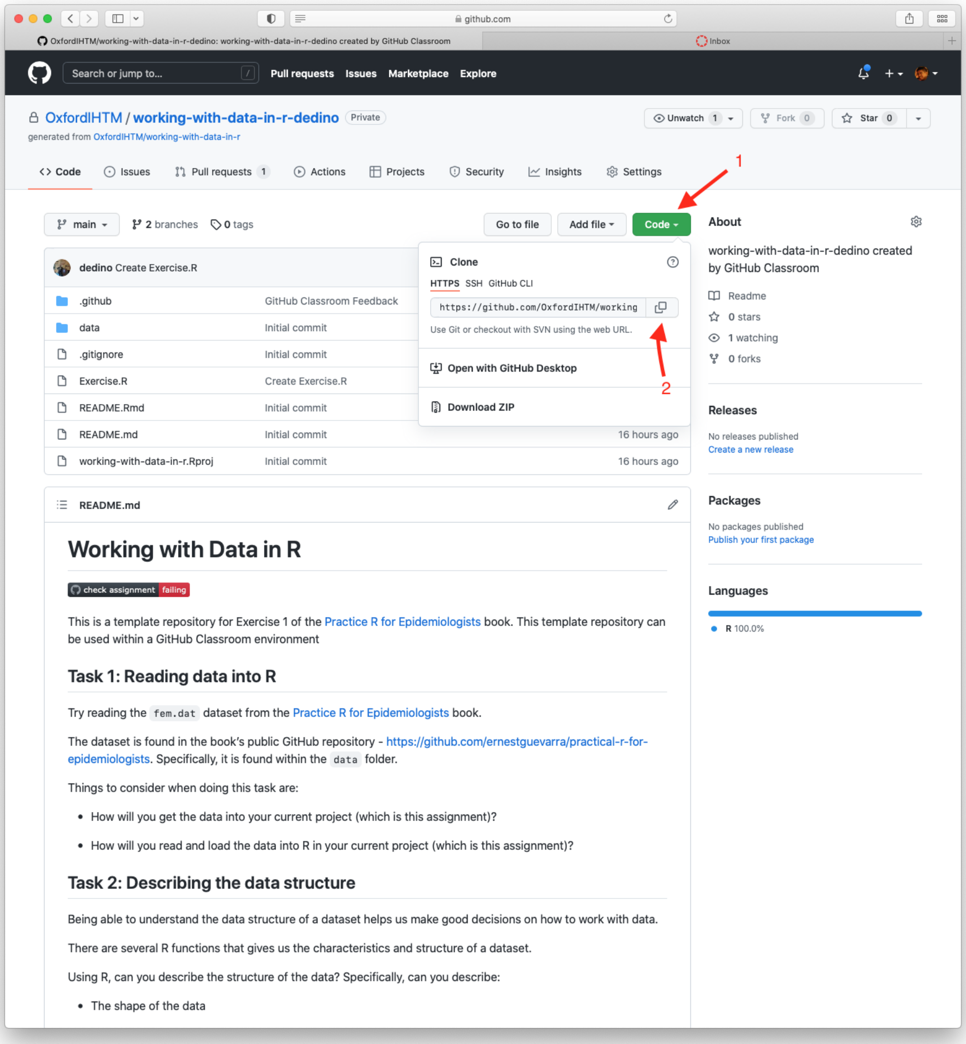
6. Once you have created a GitHub account, please do share your username to the lecture seriese presenter so he can add your username to the Oxford IHTM GitHub account and add you to the Oxford IHTM GitHub Classroom.

Chapter 9

Cloning a GitHub repository

The following steps show how to clone a GitHub repository into your local computer using RStudio. This tutorial is a summary of the the instructions described here - https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository.

9.1 Copy the repository URL of the repository you want to clone from GitHub



dedino Create Exercise.R

.github	GitHub Classroom Feedback	
data	Initial commit	
.gitignore	Initial commit	
Exercise.R	Create Exercise.R	
README.Rmd	Initial commit	
README.md	Initial commit	16 hours ago
working-with-data-in-r.Rproj	Initial commit	16 hours ago

README.md

Working with Data in R

This is a template repository for Exercise 1 of the Practice R for Epidemiologists book. This template repository can be used within a GitHub Classroom environment

Task 1: Reading data into R

Try reading the `fem.dat` dataset from the Practice R for Epidemiologists book.

The dataset is found in the book's public GitHub repository - https://github.com/ernestguevarra/practical-r-for-epidemiologists. Specifically, it is found within the `data` folder.

Things to consider when doing this task are:

- How will you get the data into your current project (which is this assignment)?
- How will you read and load the data into R in your current project (which is this assignment)?

Task 2: Describing the data structure

Being able to understand the data structure of a dataset helps us make good decisions on how to work with data.

There are several R functions that gives us the characteristics and structure of a dataset.

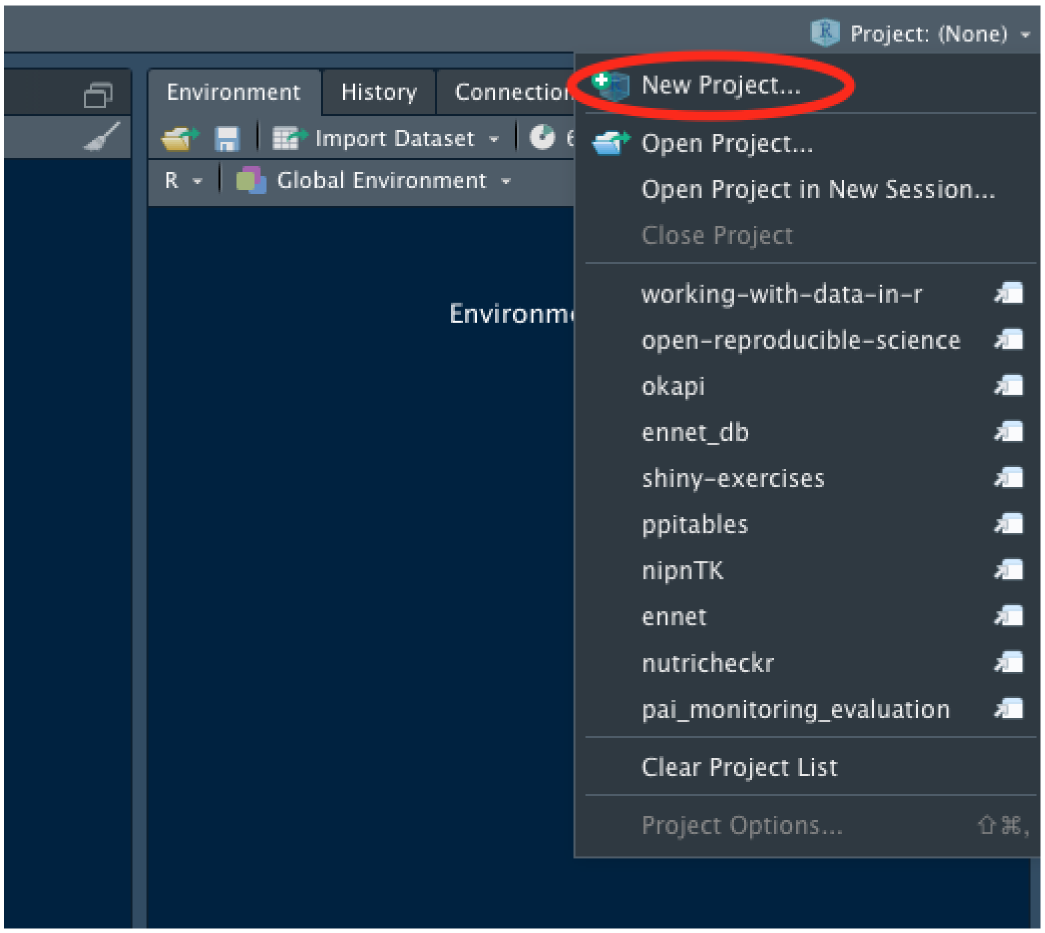
Using R, can you describe the structure of the data? Specifically, can you describe:

- The shape of the data

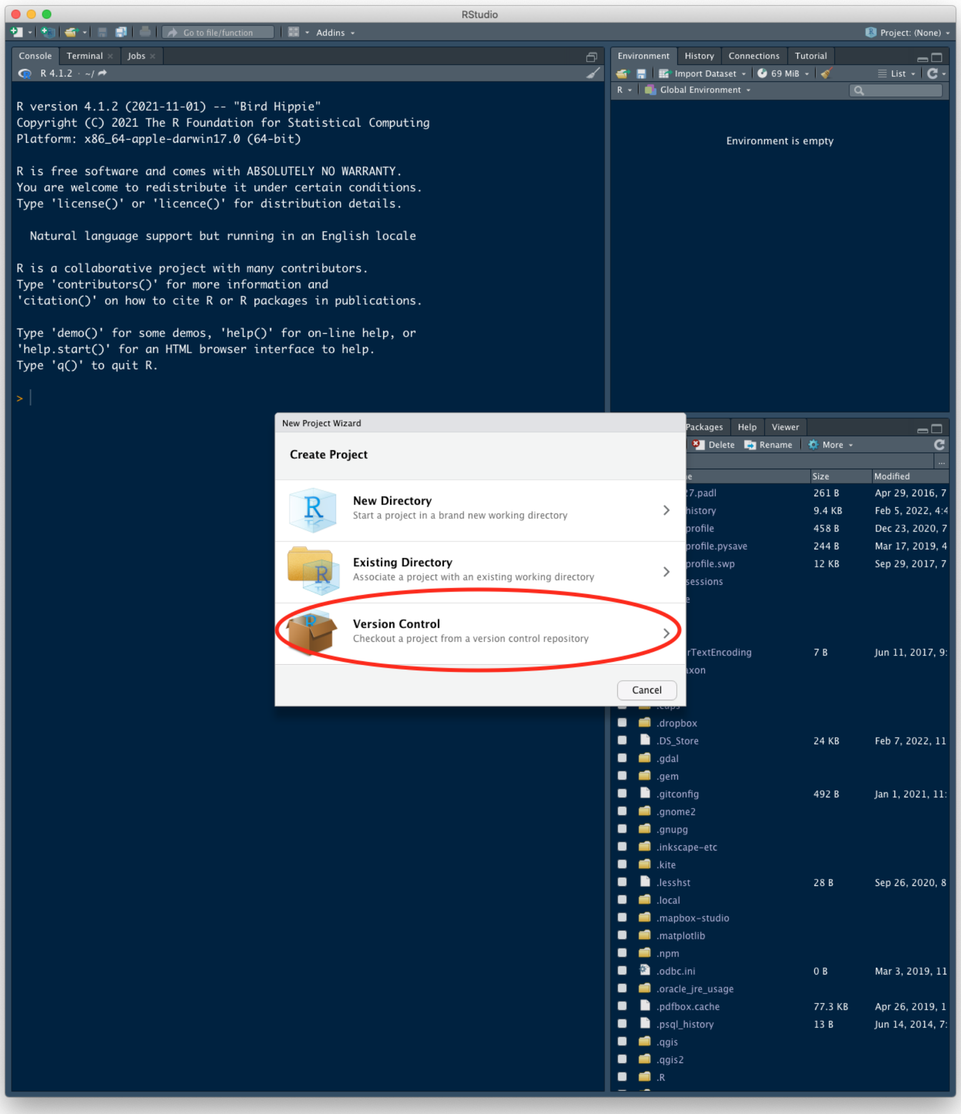
7.1.1 Go to the GitHub page of the repository you own and click on the green button that is labeled code

7.1.2 Click on the copy to clipboard icon to copy the repository URL.

9.2 Go to RStudio and create new project



9.3 Choose Version Control



```
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 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.

>
```

New Project Wizard

Create Project

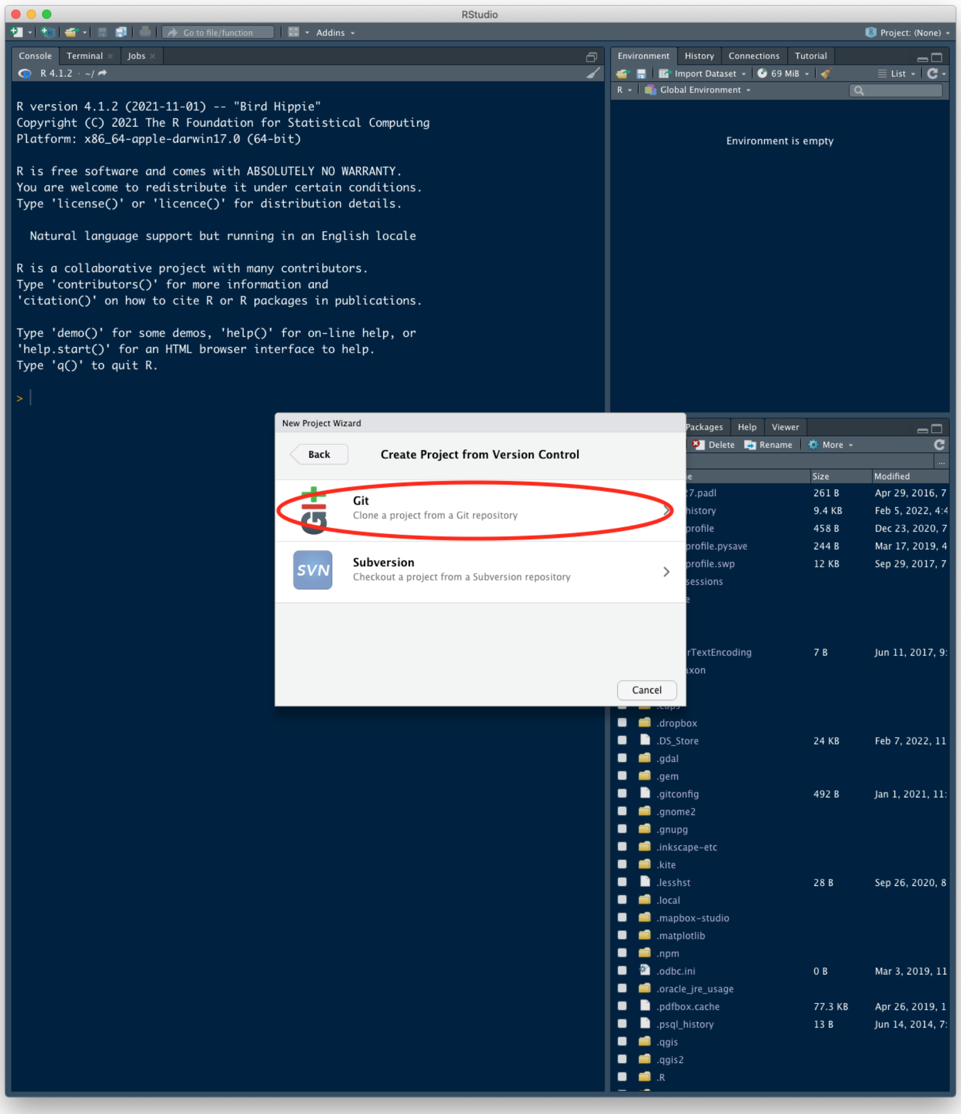
New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Cancel

9.4 Select Git



```
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 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.

>
```

Environment is empty

New Project Wizard

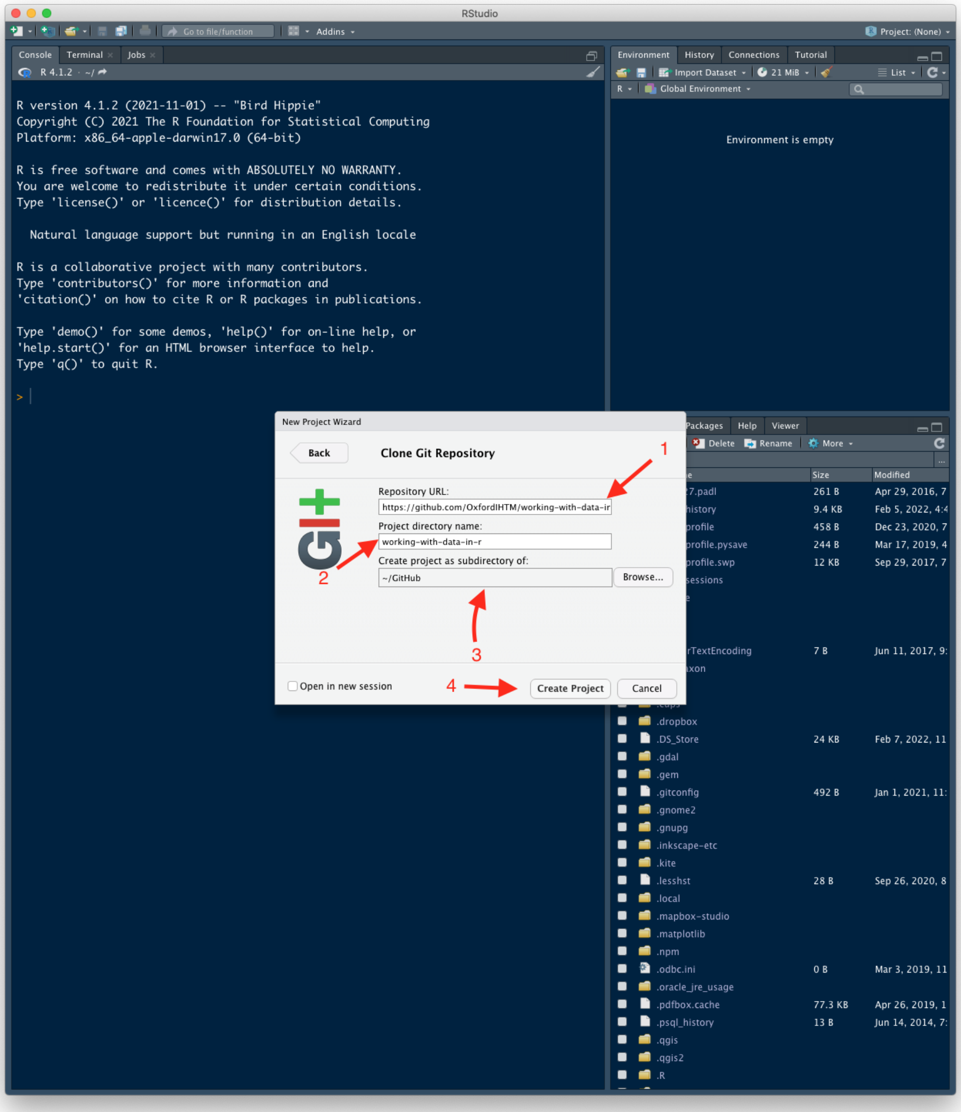
Create Project from Version Control

Git
Clone a project from a Git repository

Subversion
Checkout a project from a Subversion repository

```
e                      Size      Modified
7.padl                 261 B     Apr 29, 2016, 7
history                9.4 KB    Feb 5, 2022, 4:4
profile                458 B     Dec 23, 2020, 7
profile.pysave         244 B     Mar 17, 2019, 4
profile.swp            12 KB     Sep 29, 2017, 7
sessions
e

rTextEncoding          7 B       Jun 11, 2017, 9:
axon

.dropbox
.DS_Store              24 KB     Feb 7, 2022, 11
.gdal
.gem
.gitconfig             492 B     Jan 1, 2021, 11:
.gnome2
.gnupg
.inkscape-etc
.kite
.lesshst               28 B      Sep 26, 2020, 8
.local
.mapbox-studio
.matplotlib
.npm
.odbc.ini              0 B       Mar 3, 2019, 11
.oracle_jre_usage
.pdfbox.cache          77.3 KB   Apr 26, 2019, 1
.psql_history          13 B      Jun 14, 2014, 7:
.qgis
.qgis2
.R
```

9.5 Setup repository settings



```
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 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.

>
```

New Project Wizard

Clone Git Repository

Repository URL: https://github.com/OxfordIHTM/working-with-data-ir

Project directory name: working-with-data-in-r

Create project as subdirectory of: ~/GitHub

Open in new session

	Size	Modified
7.padl	261 B	Apr 29, 2016, 7
history	9.4 KB	Feb 5, 2022, 4:4
profile	458 B	Dec 23, 2020, 7
profile.pysave	244 B	Mar 17, 2019, 4
profile.swp	12 KB	Sep 29, 2017, 7
sessions		
rTextEncoding	7 B	Jun 11, 2017, 9:
xon		
.dropbox		
.DS_Store	24 KB	Feb 7, 2022, 11
.gdal		
.gem		
.gitconfig	492 B	Jan 1, 2021, 11:
.gnome2		
.gnupg		
.inkscape-etc		
.kite		
.lesshst	28 B	Sep 26, 2020, 8
.local		
.mapbox-studio		
.matplotlib		
.npm		
.odbc.ini	0 B	Mar 3, 2019, 11
.oracle_jre_usage		
.pdfbox.cache	77.3 KB	Apr 26, 2019, 1
.psql_history	13 B	Jun 14, 2014, 7:
.qgis		
.qgis2		
.R		

7.5.1 Paste the repository URL you copied earlier.

7.5.2 The Project directory name should be specified already after you paste the repository URL. Use the suggested directory name.

7.5.3 Browse for the directory on your local computer where you want to save the files for the specified project.

7.5.4 Click on the `Create Project` button/icon.

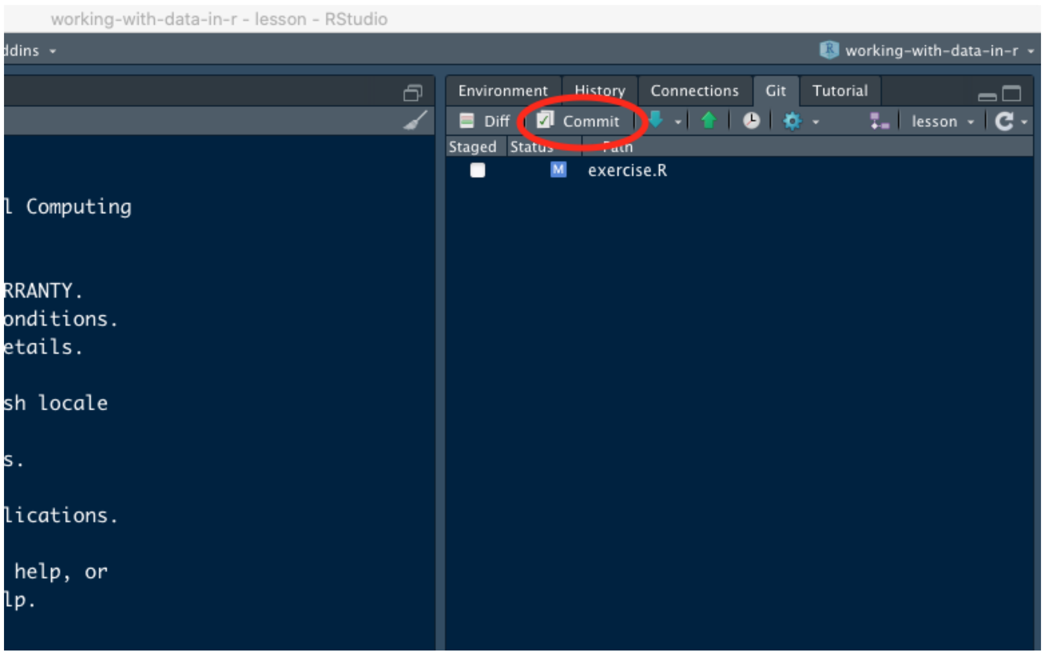
You will now have the GitHub repository in your local computer.

Chapter 10

Commit and push to GitHub

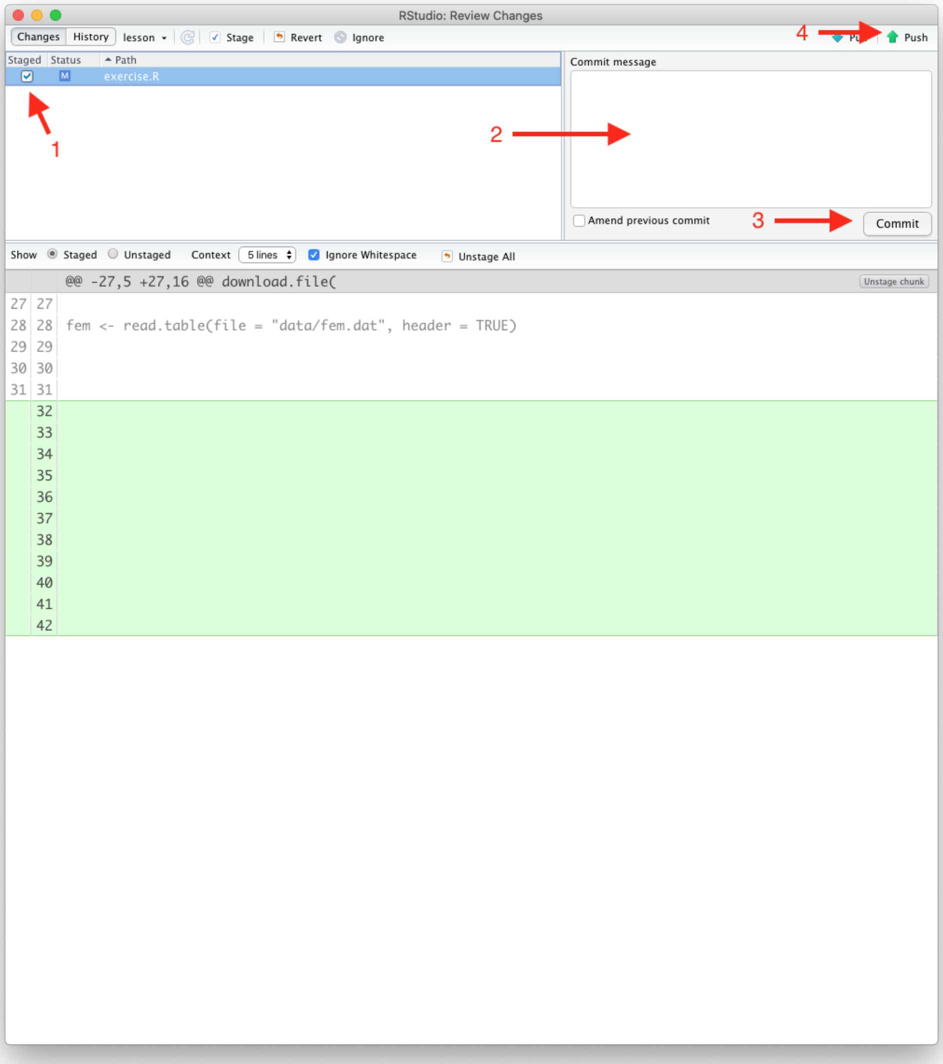
Following are the steps to take in committing your changes in RStudio and pushing them to GitHub.

10.1 Click on `Commit` in the Git tab on RStudio



```
l Computing

RRANTY.
onditions.
etails.

sh locale

s.

lications.

 help, or
lp.
```

10.2 Getting changes saved and push to GitHub



```
@@ -27,5 +27,16 @@ download.file(
fem <- read.table(file = "data/fem.dat", header = TRUE)
```

10.2.1 Tick the box beside the file that has changed to stage the changes.

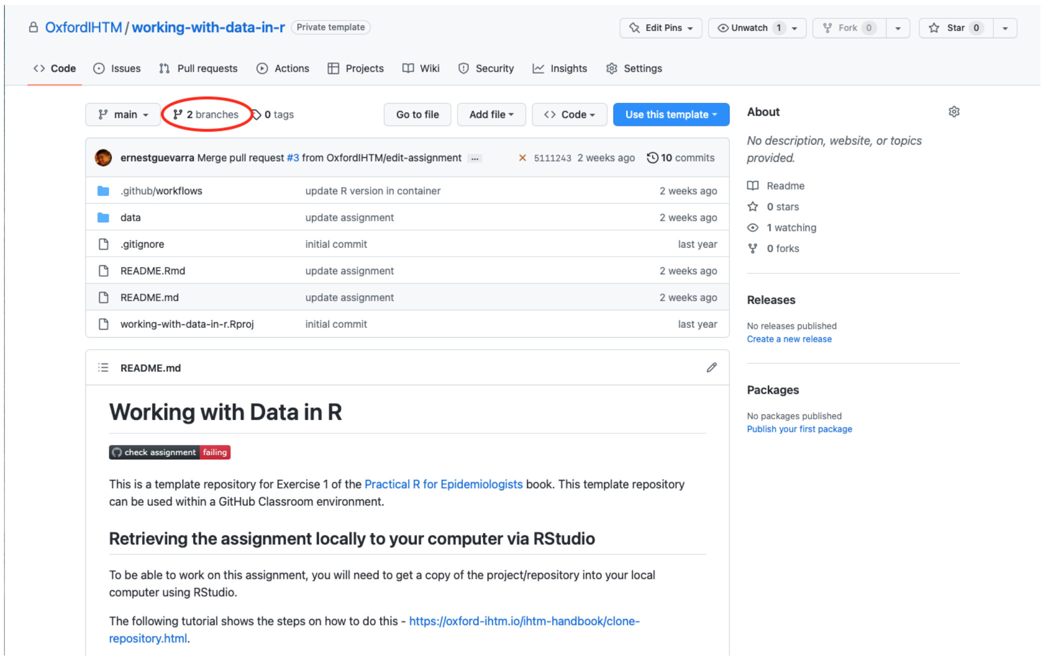
10.2.2 Write a commit message in the `Commit message` dialog box. In the commit message, describe the changes that you made.

10.2.3 Click on the `Commit` button.

10.2.4 Click on the `Push` button.

10.3 Initiate a pull request

10.3.1 Click on the **branches** link from your repository



ernestguevarra Merge pull request #3 from OxfordIHTM/edit-assignment ... × 5111243 2 weeks ago 10 commits

.github/workflows	update R version in container	2 weeks ago
data	update assignment	2 weeks ago
.gitignore	initial commit	last year
README.Rmd	update assignment	2 weeks ago
README.md	update assignment	2 weeks ago
working-with-data-in-r.Rproj	initial commit	last year

README.md

Working with Data in R

check assignment failing

This is a template repository for Exercise 1 of the Practical R for Epidemiologists book. This template repository can be used within a GitHub Classroom environment.

Retrieving the assignment locally to your computer via RStudio

To be able to work on this assignment, you will need to get a copy of the project/repository into your local computer using RStudio.

The following tutorial shows the steps on how to do this - https://oxford-ihtm.io/ihtm-handbook/clone-repository.html.

About

No description, website, or topics provided.

Readme
0 stars
1 watching
0 forks

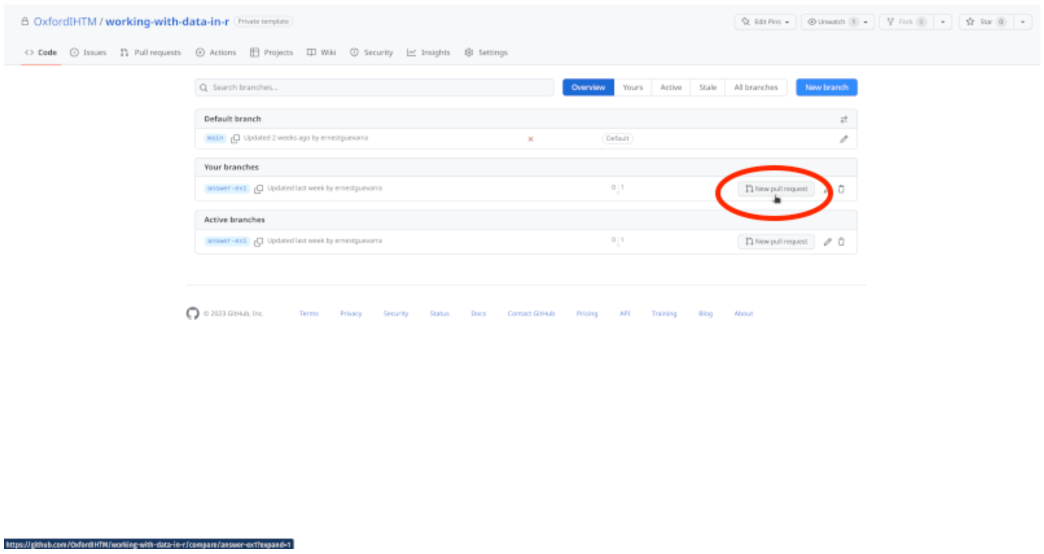
Releases

No releases published
Create a new release

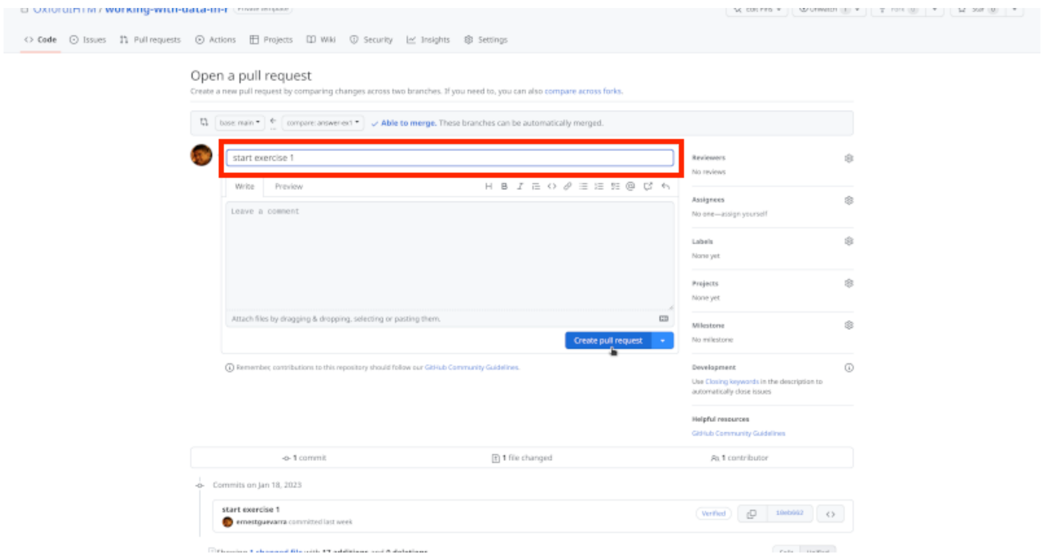
Packages

No packages published
Publish your first package

10.3.2 Click on the **Make pull request** link on the appropriate branch

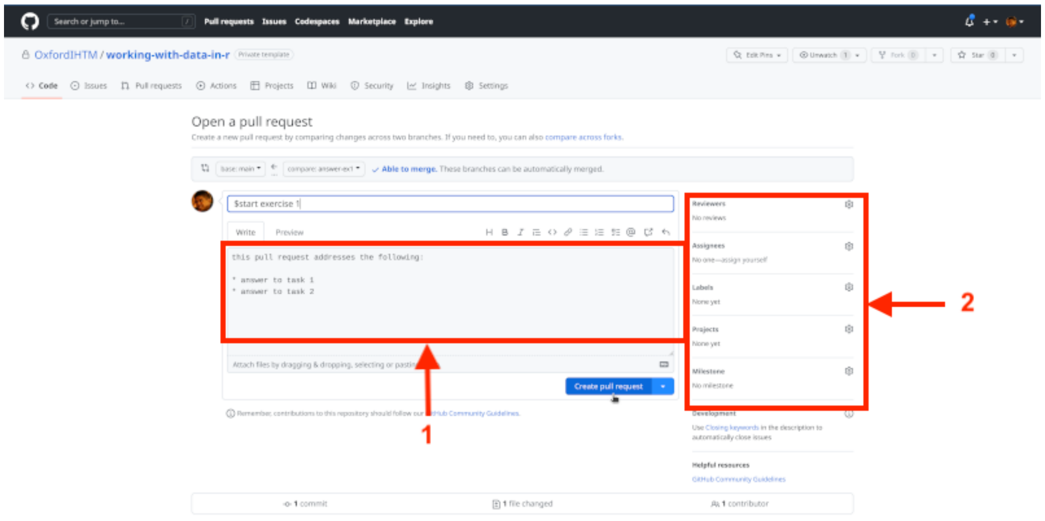


10.3.3 Enter a title for your pull request



Make the titel as short but as informative as possible

10.3.4 Add further description about the pull request (optional) and then click on **Create pull request** button


OxfordIHTM / working-with-data-in-r
Open a pull request
Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.
base: main
compare: answer-ex1
Able to merge. These branches can be automatically merged.
Write
Preview
Create pull request
Reviewers
No reviews
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
1
2

$start exercise 1

```
this pull request addresses the following:

* answer to task 1
* answer to task 2
```

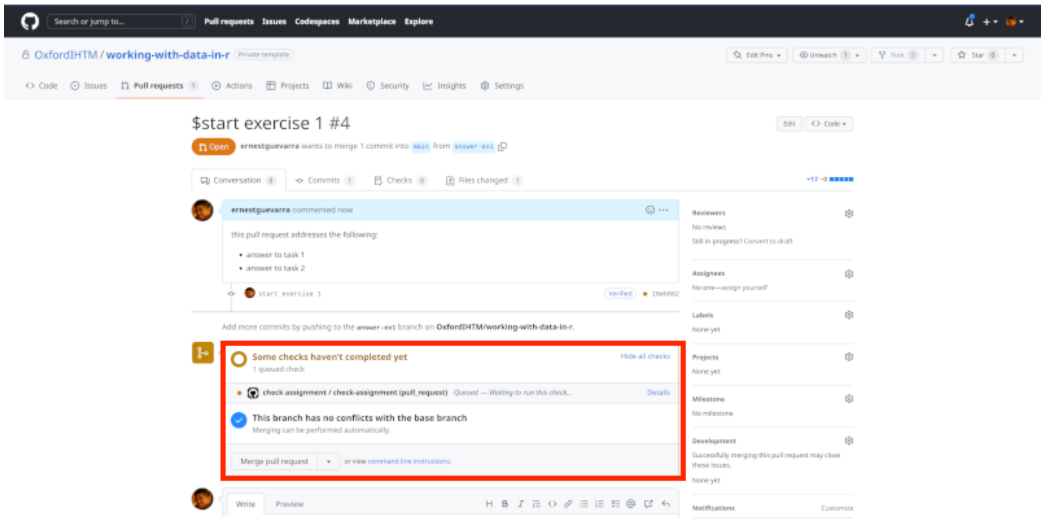
If you think more information will help the reviewer navigate through the changes you have made, use the comment box to add more details. This comments box can interpret Markdown syntax so you can format your text accordingly.

On the right hand side of the pull request page, you can set a specific reviewer for your pull request (recommended). Also, given that you are making this pull request, assign this pull request to you so you are notified of the progress of this pull request.

Then finally click on the **Create pull request** button

10.3.5 Wait for review

If the project has automated checks included, you will see that these checks will get initiated.



$start exercise 1 #4

Open ernestguevarra wants to merge 1 commit into `main` from `answer-ex1`

ernestguevarra commented now

this pull request addresses the following:

- answer to task 1
- answer to task 2

`start exercise 1` Verified 10eb662

Add more commits by pushing to the answer-ex1 branch on OxfordIHTM/working-with-data-in-r.

Some checks haven't completed yet
1 queued check

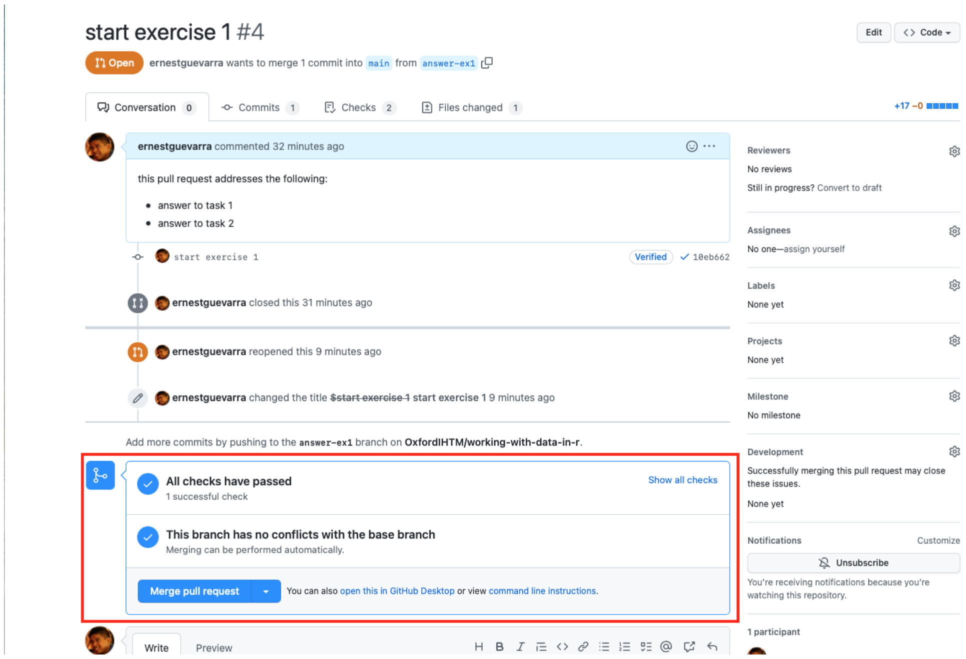
check assignment / check-assignment (pull_request) *Queued — Waiting to run this check...*

This branch has no conflicts with the base branch
Merging can be performed automatically.

or view command line instructions.

No reviews
Still in progress? Convert to draft
No one—assign yourself
None yet
None yet
No milestone
Successfully merging this pull request may close these issues.
None yet

If there are no issues with the code, the automated checks should show that all checks have passed.



start exercise 1 #4

Open ernestguevarra wants to merge 1 commit into `main` from `answer-ex1`

ernestguevarra commented 32 minutes ago

this pull request addresses the following:

- answer to task 1
- answer to task 2

`start exercise 1` Verified ✓ 10eb662

ernestguevarra closed this 31 minutes ago

ernestguevarra reopened this 9 minutes ago

ernestguevarra changed the title ~~$start exercise 1~~ start exercise 1 9 minutes ago

Add more commits by pushing to the answer-ex1 branch on OxfordIHTM/working-with-data-in-r.

All checks have passed
1 successful check

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

No reviews
Still in progress? Convert to draft
No one—assign yourself
None yet
None yet
No milestone
Successfully merging this pull request may close these issues.
None yet
You're receiving notifications because you're watching this repository.
1 participant

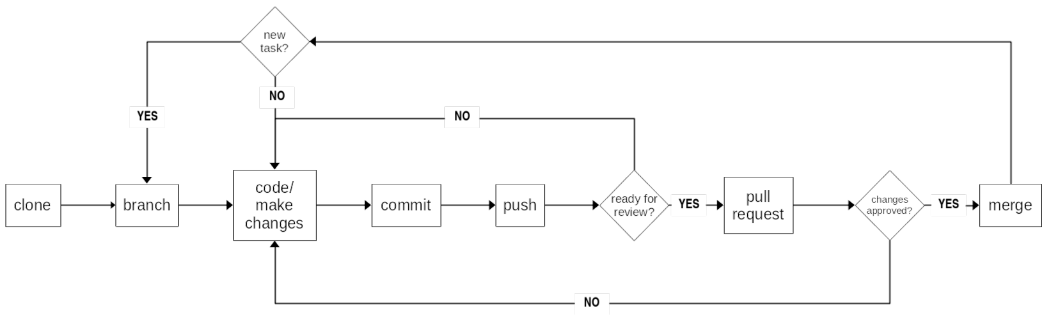
Wait for reviewer's feedback/comments. If reviewer request's changes, make changes to your code and then commit and push again (as above). If your project has automated checks, this will get triggered again within the same pull request. Your reviewer will be notified of the changes you have made and should review your work again. Once reviewer approves changes, you can then merge your work to the main branch.

Chapter 11

Participating in an R/RStudio project

Following are the general steps to take when participating in an existing R/RStudio project that has been initiated and led by someone else.

The following diagram illustrates the steps in this process:



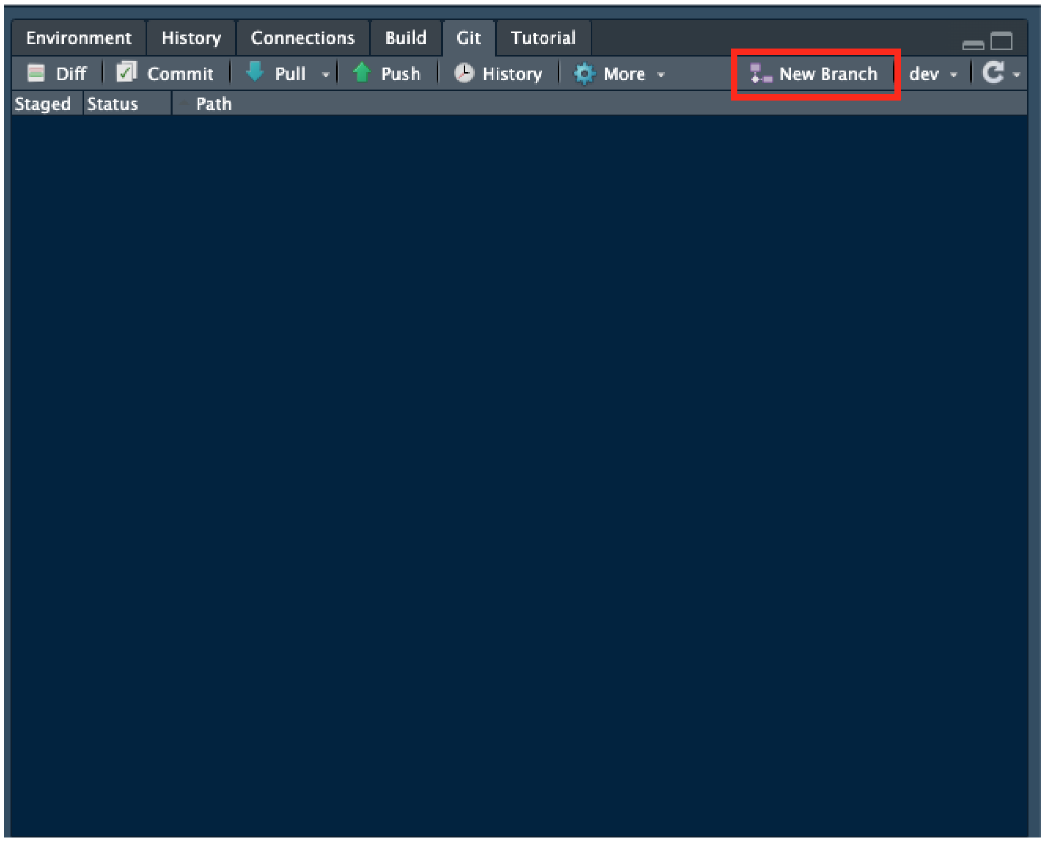
11.1 Clone the project to your local machine

Cloning a project to your local machine is described in Chapter 9

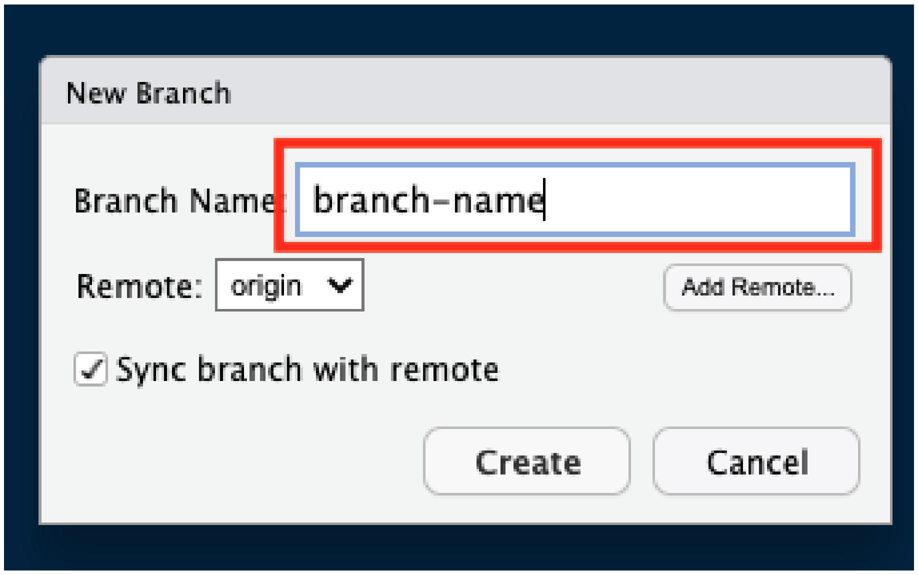
11.2 Create a new branch from the main branch

Before making any changes to the project, create a new branch as follows:

11.2.1 Click on New Branch



11.2.2 Name the new branch



Branch Name: branch-name

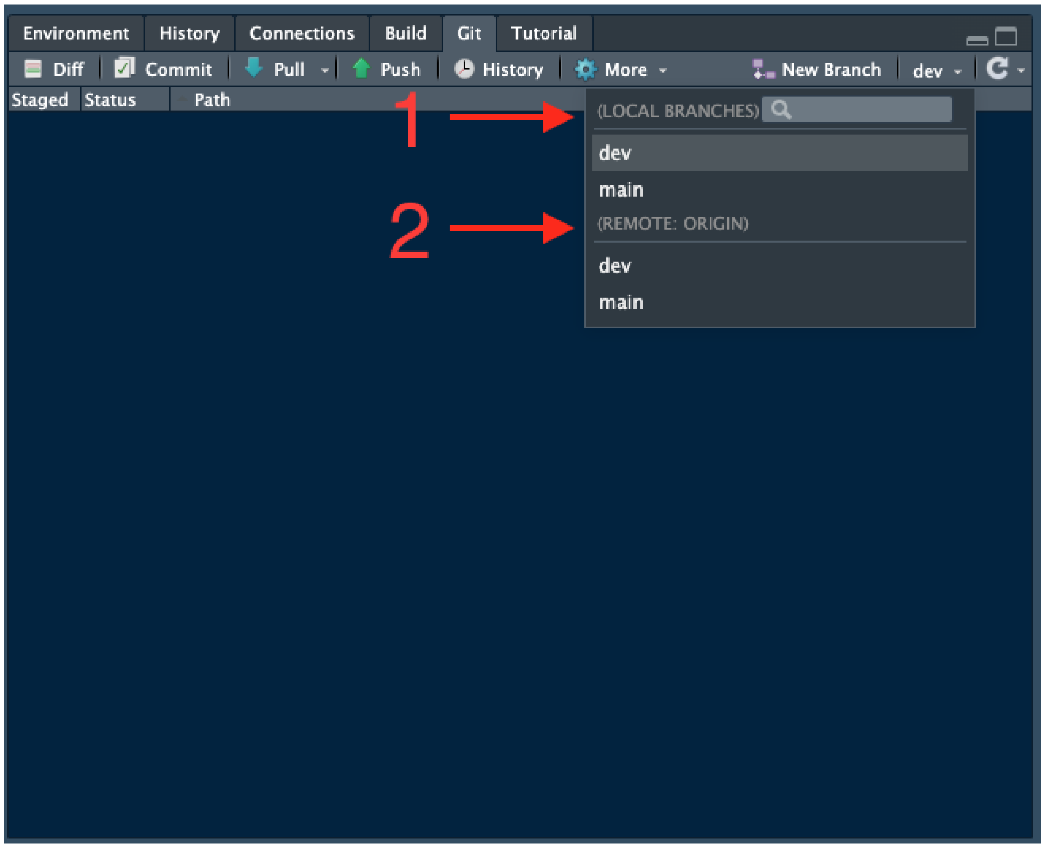
Remote: origin

✓ Sync branch with remote

Name the branch uniquely. The best way to name a branch will be based on how the team/person you are working with prefers to name branches. Some would like the branch name to succinctly describe the type of change that is being made. Some may ask you to name your branch with your username. Some may ask you to name your branch using coded values.

Once named, click on **Create**

You will now see the new branch in the list of branches



11.3 Code and make changes to your branch

Start coding and implement the changes you want to make or the changes that your collaborator/s asked you to make.

11.4 Commit and push your changes and initiate a pull request

After making changes, you should **commit and push** your changes. This process is described in Chapter 10. Your code and your changes do not have to be complete already for your to commit and push changes. It is good practice to commit and push frequently (at least once a day usually at the end of your coding session). See this as similar to saving your work at multiple stages.

Once your code and the changes you want to make are complete (and ideally that they are working correctly on your local machine), and that you are ready to have your work reviewed, you can now make a **pull request**. This process is also described in Chapter 10.

11.5 Merge pull request

Once your chosen reviewer has seen your work, they may ask you to make changes based on what they see with your code. If so, then start coding again on the same branch and address the reviewers comments, *commit** those changes and *push **the changes to your remote repository. Your changes will push into the same existing open** pull request** that is waiting approval. The reviewer can then view your changes and make the necessary feedback.

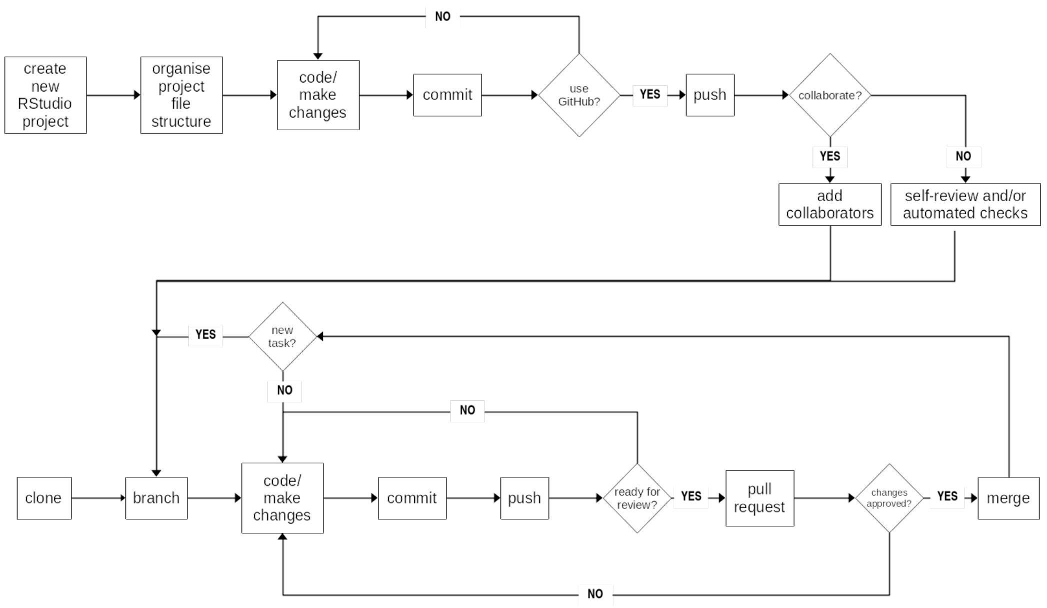
Once reviewer approves your changes, they may either merge your pull request themselves or they may let you know in their feedback that they are happy with your changes and that you can now merge your pull request. If so, then click on the **Merge pull request** button.

Your changes have now been integrated into the main branch of the project.

Chapter 12

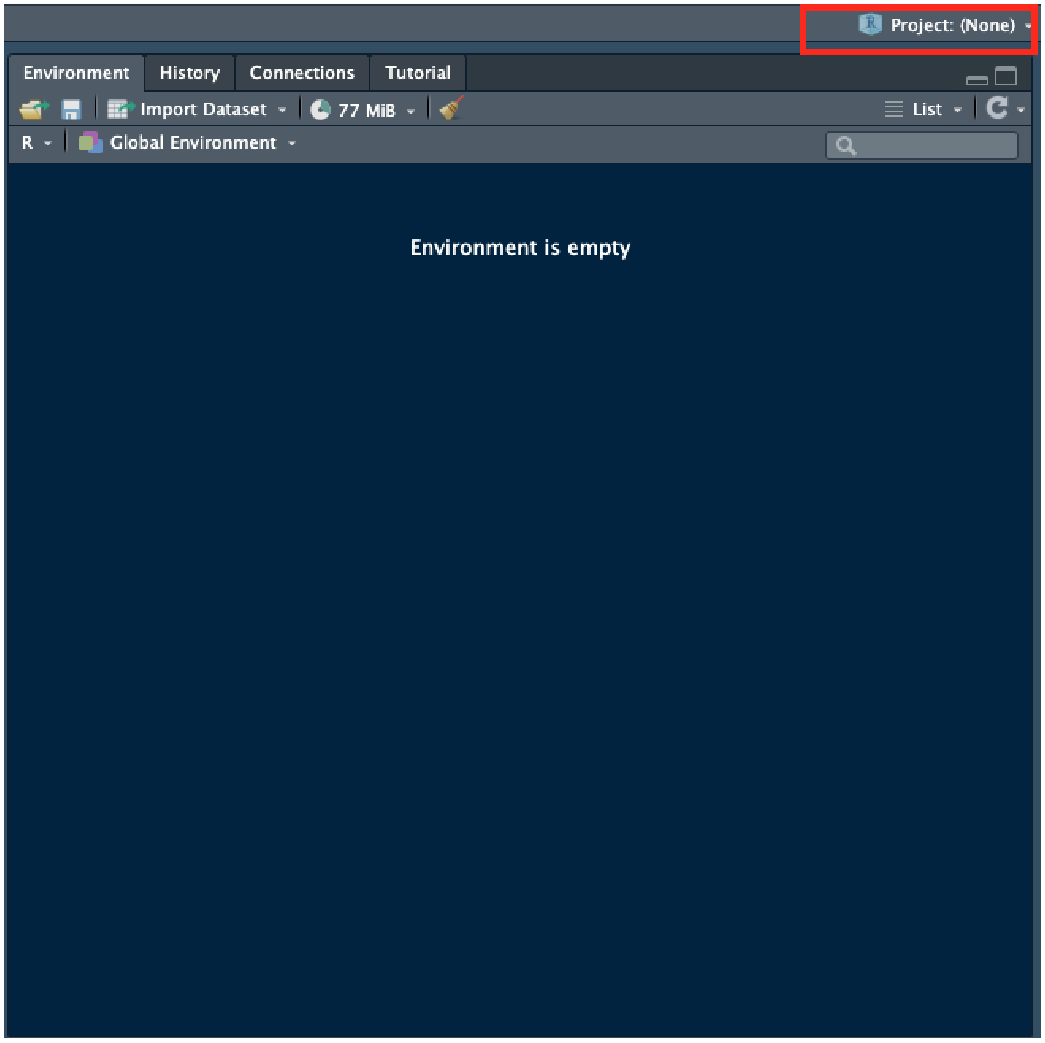
Initiating an R/RStudio project

Following is a diagram of the steps in initiating your own R/RStudio project.

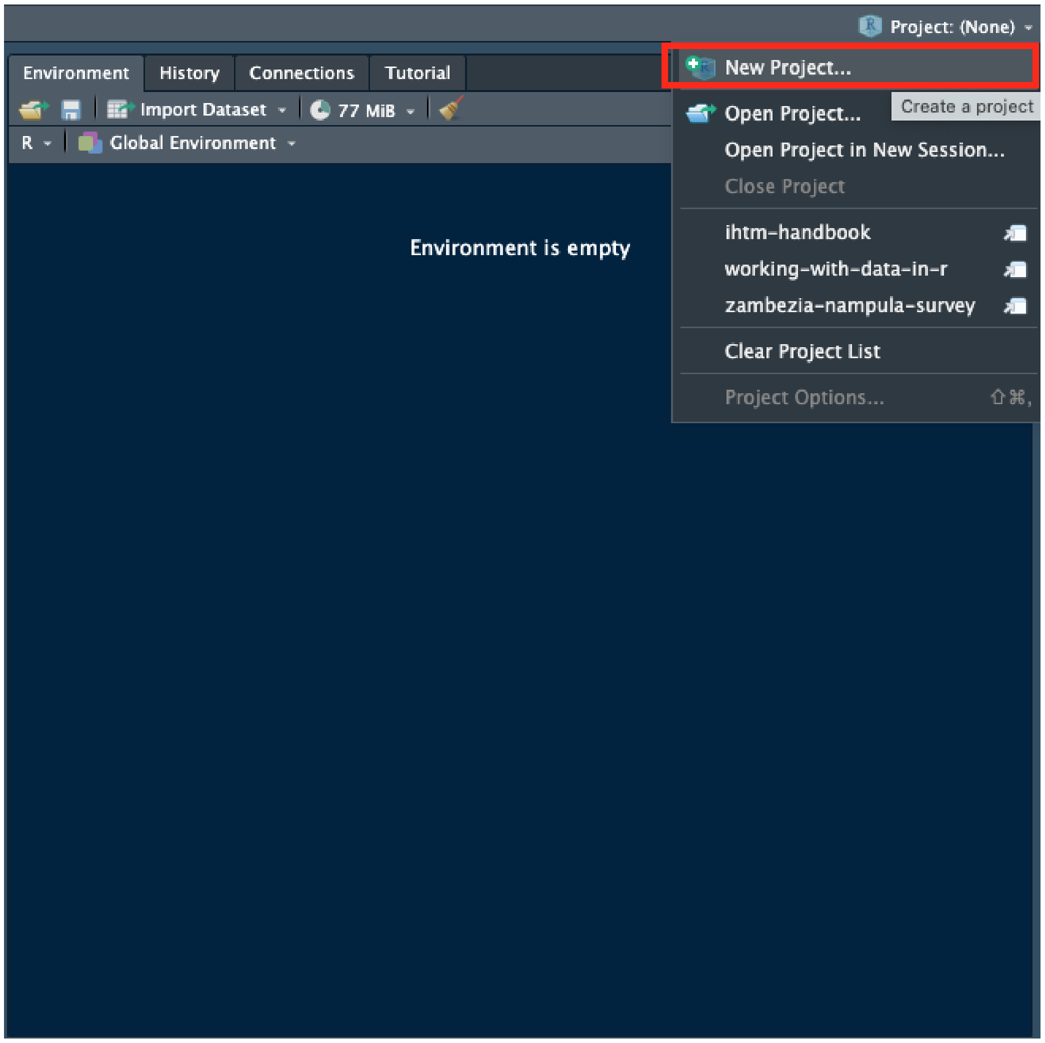


12.1 Create a new project in RStudio

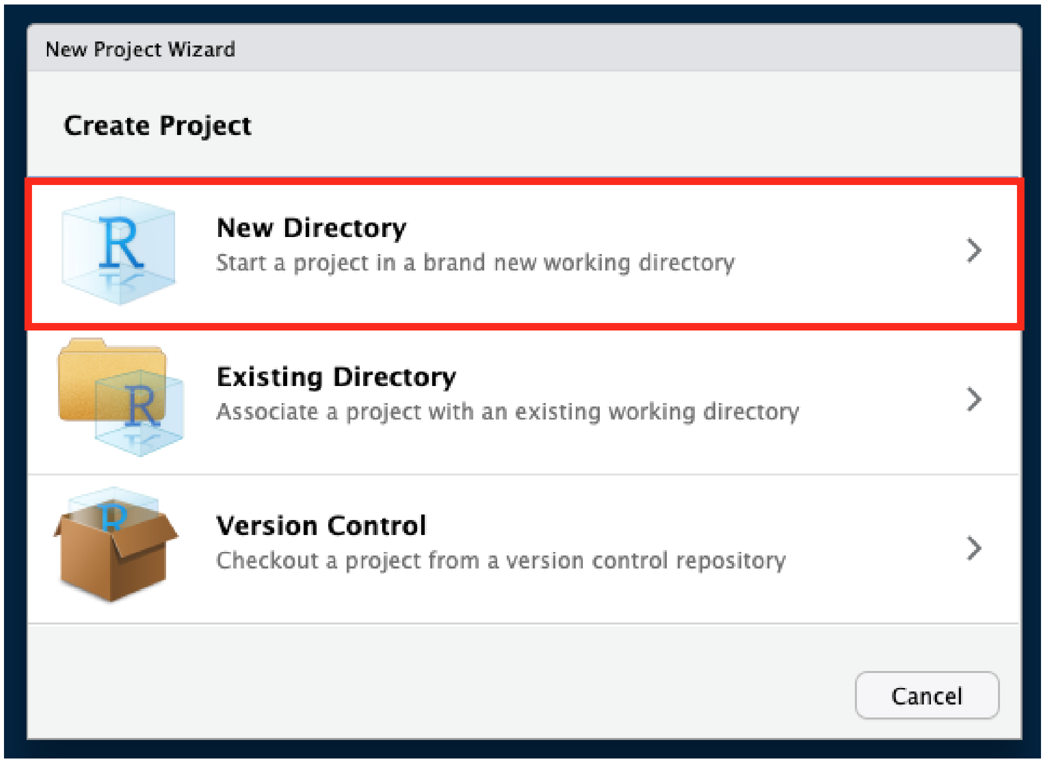
12.1.1 Click on New Project button on RStudio



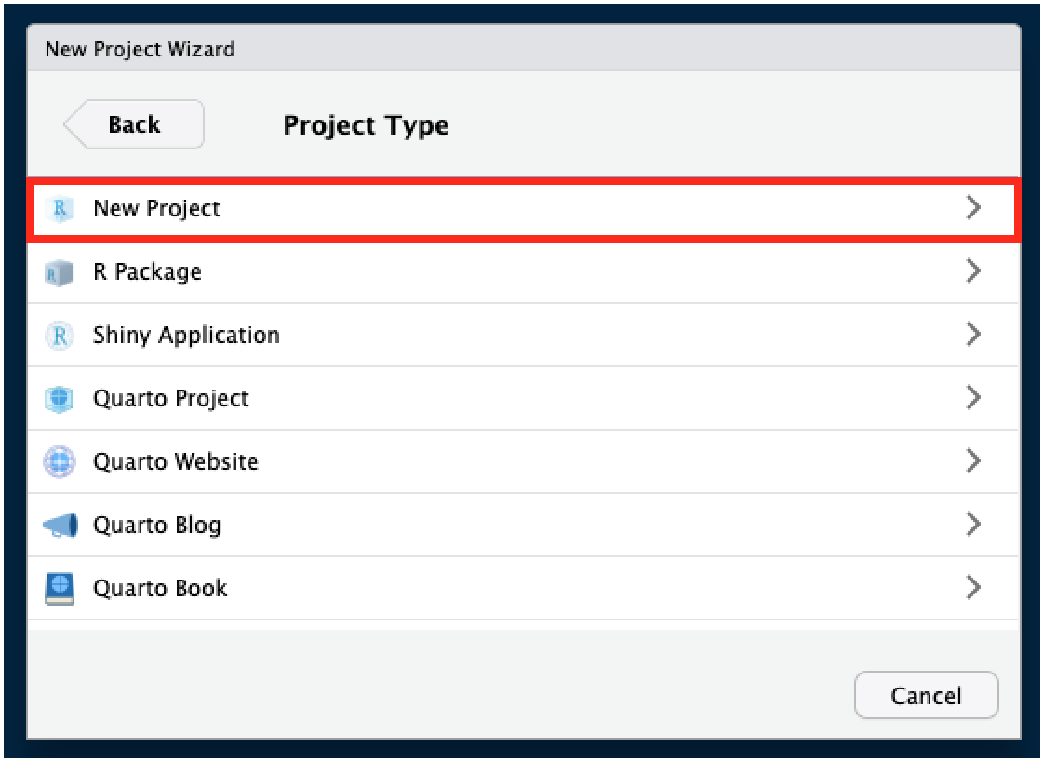
Environment is empty


Project: (None)
Environment
History
Connections
Tutorial
Import Dataset
77 MiB
R
Global Environment
Environment is empty
New Project...
Create a project
Open Project...
Open Project in New Session...
Close Project
ihtm-handbook
working-with-data-in-r
zambezia-nampula-survey
Clear Project List
Project Options...

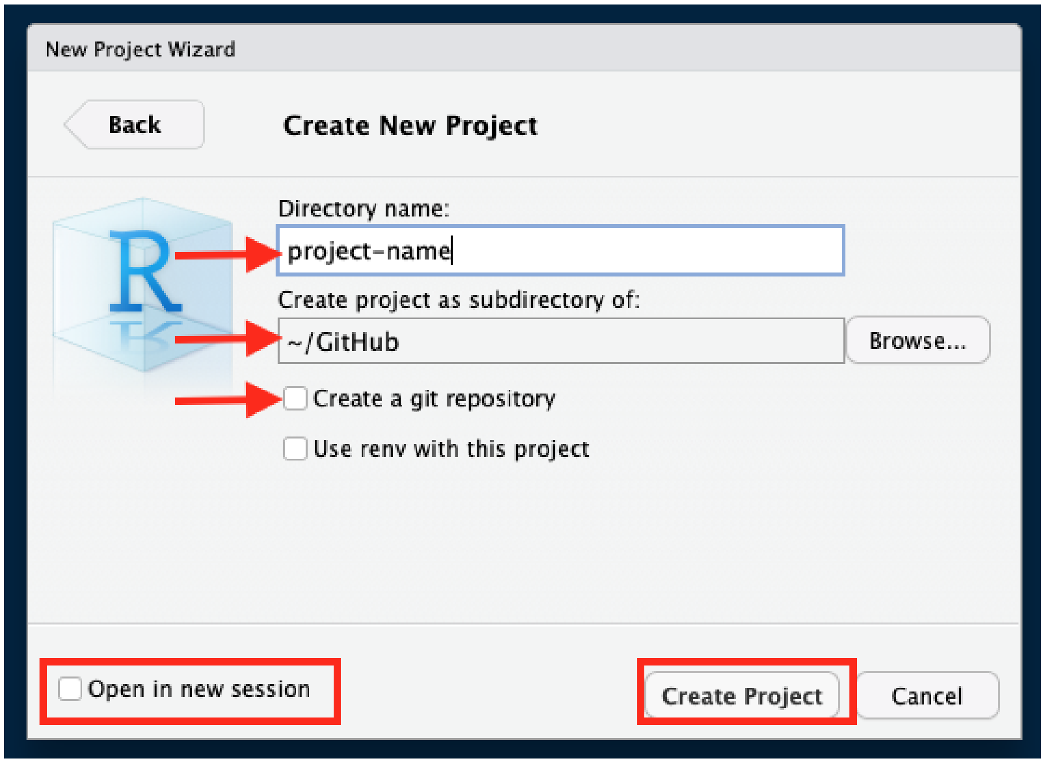
12.1.2 Create a New Directory


New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

12.1.3 Select New Project as project type



12.1.4 Specify details for new project



Directory name: project-name

Create project as subdirectory of: ~/GitHub

☐ Create a git repository

☐ Use renv with this project

12.4.1 Specify a project name. Best practices for naming a project are:

- Make sure that name is succinct (as short as possible while at the same time descriptive of the project);
- Don't use spaces for your project name. If you need to separate words, use a **dash** or an **underscore**;
- Avoid using capital letters.

12.4.2 Specify a directory/location in your local machine where to place the directory of your new project

12.4.3 Decide whether to use git to version this project

Here you can decide whether you want to use `git` to version your project. Remember that using `git` doesn't mean you have to use GitHub. `git` is software installed in your local machine and it versions what you have on your local machine. You don't need `GitHub` or any other similar service to version your code with `git` in your local machine.

I would recommend that you tick this option for any new project you create so that you can version your work in your local machine even if you don't want or decide not to use `GitHub` or any other remote `git` service.

12.4.4 Do you want to open a new session

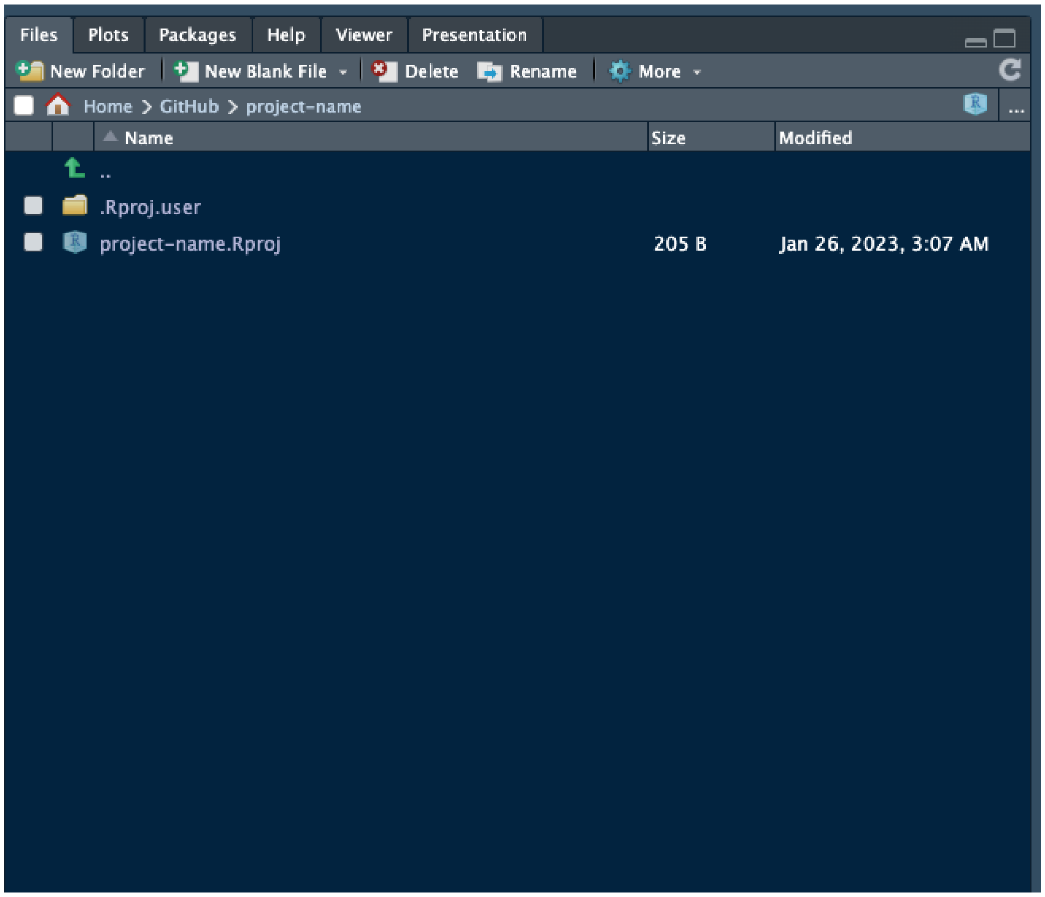
This is by default unticked and will open the new project within the existing RStudio session (if any). This means that if you have an existing RStudio session with another project that you are working on, that project will be closed and the new project you are creating will open in the existing RStudio session.

If you need your existing RStudio session and the project within it to remain open alongside the new project you are creating, tick this box/option.

12.4.4 Click on **Create New Project**

Once you click on **Create New Project**, you will now see the new project open in RStudio.

You will also see something like below witin the file explorer pane of RStudio.



Name	Size	Modified
..		
.Rproj.user		
project-name.Rproj	205 B	Jan 26, 2023, 3:07 AM

12.2 Structure/organise your new project appropriately

Project organisation is vital because:

- supports productivity because the different components of the project are placed in directories where they should be;
- enables clarity in communicating project structure;
- facilitates collaboration.

Organising an R project can be user- and project-dependent but there are generally accepted project organising structure that is common to most well-organised projects. Below is an example:

```
.
|-- my-project
    |-- data
    |-- output
        |-- figures
    |-- R
    |-- my-project.Rproj
    |-- analysis_workflow.R
    |-- README.md
```

12.3 Start coding

This will include creating bespoke R functions (as required) and creating an Rscript for the step-by-step processes in your scientific workflow.

12.4 Next steps

The next steps will depend on whether you will use `git` and `GitHub` for versioning your project and whether or not you will work on your project as a solo scientiest or work and collaborate with other scientists.

Chapter 13

Scientific workflows

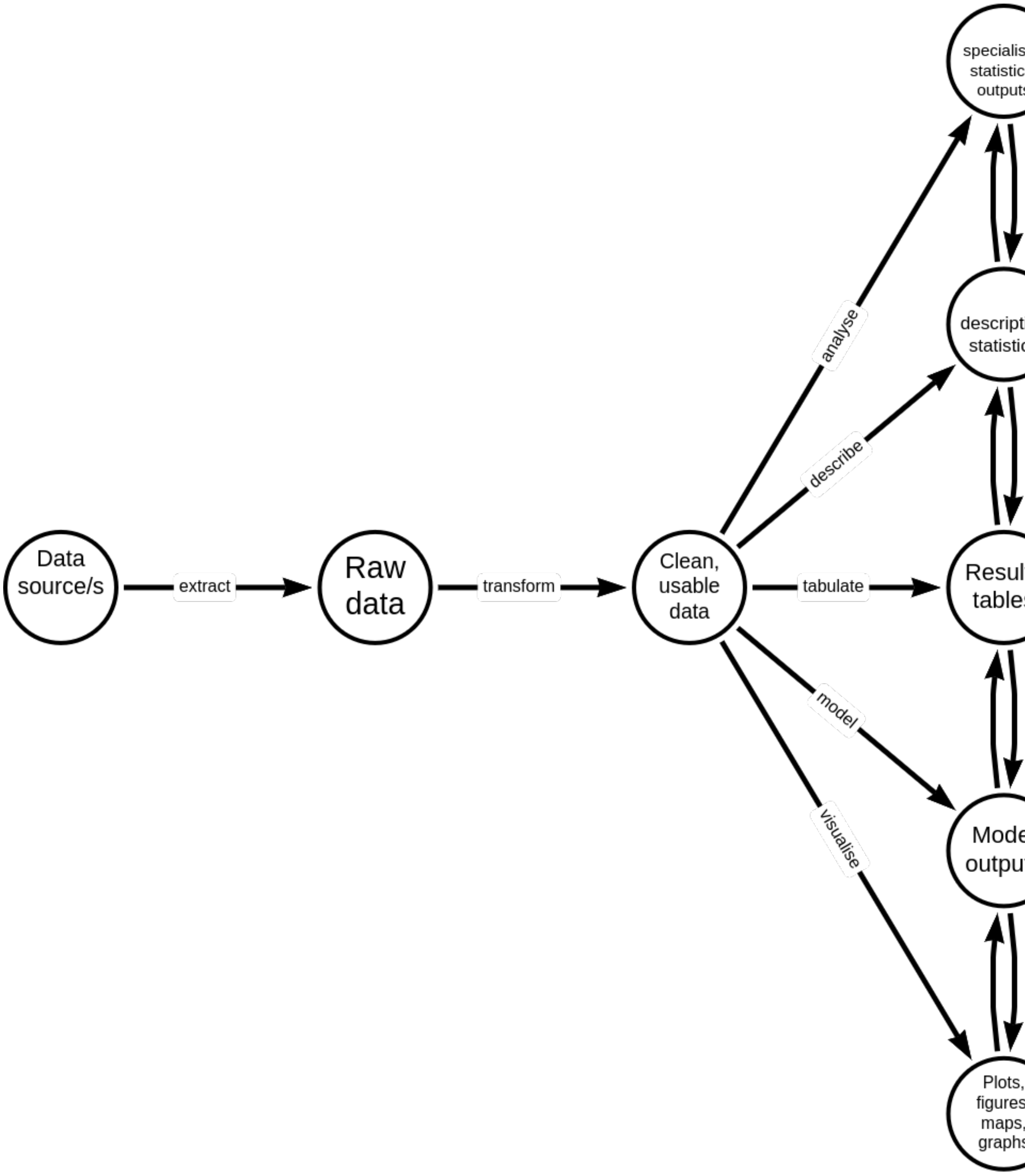
At this point, you would have written your own R code and R scripts and saved these within an R file (`.R`).

By now, you would have also appreciated how extensible R is through built-in packages and/or through functions that you have created yourself.

So far, in the examples that we have worked on, the operations and the problems have been quite straightforward. But from your own experience dealing with your own data, real world data is far from straightforward and far from simple. Complexity is almost always a given.

R's scripting capability and R's extensibility are its main characteristics that make R a good tool for creating robust scientific workflows particularly for complex data and research projects.

A typical scientific workflow would have the following steps:


Data source/s
extract
Raw data
transform
Clean, usable data
analyse
describe
tabulate
model
visualise
specialis statistic outputs
descripti statistic
Resul tables
Mode outpu
Plots, figures maps, graphs

In general, an R script should reflect the different steps outlined above. Hence, an R script of a scientific workflow would tend to look like this:

```
## Load libraries


## Retrieve and read data


## Process data


## Analyse data

### Descriptive analysis

### Statistical tests

### Model specifications


## Outputs

### Tabulation of results

### Model outputs

### Plots


## Report
```

In this chapter, we will go through a step-by-step walkthrough of how to build a robust scientific workflow in R. A robust worklow is one that is **portable** i.e., not dependent on hardware and software and instead can be run on almost any machine with very minimal, if any, additional setup or configuration required, and one that is **reproducible** i.e., can be run over and over again without issues, providing the expected results with the same data or providing updated results with new and/or updated data.

13.1 Step 1: Create a new RStudio project

- Open RStudio
- Click on the `File` option in the RStudio menu. In the dropdown menu, select `New Project`
- In the menu window, select `New directory` option.
- In the next menu window, select `New project` option.
- In the next menu window, enter the following details:
  - Name of the project - important to make the project name as short as possible but descriptive of the project you are creating; don't use spaces, instead use dash (or underscore) and avoid using capital letters;
  - Select the directory in your computer in which you want to save the project in. Click on `Browse` to open your computers file manager and navigate to the directory you want to save your project in;
  - Tick the selection box to make this project a git repository (whilst this is not necessary, this is highly recommended especially if you are collaborating with others);
  - Tick the selection box to enable `renv` in this project (this is what mainly contribute to the portability of your project); and,
  - Click on `Create project`

13.2 Step 2: Create an R file called `packages.R`

- Click on the `File` option in the RStudio menu. In the dropdown menu, select `New File` and then in the next dropdown menu, select `R script`.
- A new tab will open in your text editor pane of RStudio (upper left pane) with the name *Untitled1*. Save this file by clicking on the disk

icon on the text editor menu or do a keyboard shortcut with `CTRL + s`. Give this empty R script the filename `packages.R`.

- You should now see a file in the main directory/root directory of your project named `packages.R`
- Add code in the `packages.R` file specifying the packages you will be using in this project. There will be standard packages that we will always use with this type of workflow. So a template/generic `packages.R` file will contain the following:

```
################################################################################
#
#'
#' General packages needed for a targets workflow
#'
#
################################################################################

library(targets)
library(tarchetypes)
library(here)
library(rmarkdown)
library(knitr)
library(kableExtra)
library(dplyr)
library(openxlsx)
library(ggplot2)


################################################################################
#
#'
#' Add other packages that will be used in the project below
#'
#
################################################################################
```

13.3 Step 3: Create placeholder directories for different components of workflow

- In the lower right pane of RStudio (the file manager pane), find the menu button labelled `Folder`.
- Give this new folder the label of `R`. This filder will hold all bespoke functions that we will create to use for this project workflow;
- Repeat these steps to create new folders with the following labels:
  - `data` - This folder will hold any data that we retrieve as part of this workflow.
  - `outputs` - This folder will hold all our workflow outputs such as plots/figures, tables (in Excel or CSV files), HTML and/or Word and/or PDF outputs
  - `reports` - This folder will hold all our RMarkdown report (`.Rmd`) files
  - `docs` - This folder will hold any of our deployed outputs such as HTML report, dashboard, etc.

These are placeholder directories which we will populate as we work through the workflow for this project.

13.4 Step 4: Create the target script file (`_targets.R`)

The next task is to create a {targets} script file (`_targets.R`) which is the file that will define the workflow that we will be creating.

Create the file by:

- Clicking on File –> New File –> R Script in RStudio.
- A new tab will show in your Source window on the top left quadrant of your RStudio screen. This tab will usually be called `Untitled1`.
- Save this file first and change its name to `_targets.R`. Make sure to save it in the current project directory.

- You know that you were successful in doing this once you see a file called `_targets.R` in the File system window in the lower right quadrant of your RStudio screen.

13.5 Step 5: Edit the `_targets.R` script file

Now, the next step is to edit your script file by adding sets of R code that does the following:

- Loads the packages required
- Loads custom functions (if any)
- Defines individual targets using `tar_targets` function
- Ends with a list of targets objects

A basic `{targets}` workflow will look like this:

```r
## Load libraries ---------------------------------------------------------------
library(targets)



## Load custom functions --------------------------------------------------------
for (f in list.files("R", full.names = TRUE)) source (f)
for (f in list.files(here::here("R"), full.names = TRUE)) source (f)


## Create targets and list targets objects -------------------------------------

```