R recap

Mark Dunning; mark 'dot' dunning 'at' cruk.cam.ac.uk

Last modified: 16 Mar 2016

Pre-amble

In this session we will review some of the basic features of the R language, before proceeding more-complicated workflows required for the analysis of NGS, and other high-throughput data.

We recommend using the RStudio GUI for this course.

Getting help with R

R has an in-built help system. At the *console*, you can type `?` followed by the name of a function. This will bring-up the documentation for the function; which includes the expected inputs (*arguments*), the output you should expect from the function and some use-cases.

```
?mean
```

More-detailed information on particular packages is also available (see below)

R packages

The **Packages** tab in the bottom-right panel of RStudio lists all packages that you currently have installed. Clicking on a package name will show a list of functions that available once that package has been loaded. The `library` function is used to load a package and make it's functions / data available in your current R session. *You need to do this every time you load a new RStudio session.*

```
library(beadarray)
```

There are functions for installing packages within R. If your package is part of the main **CRAN** repository, you can use `install.packages`

We will be using the `wakefield` R package in this practical. To install it, we do.

```
install.packages("wakefield")
```

Bioconductor packages have their own install script, which you can download from the Bioconductor website

```
source("http://www.bioconductor.org/biocLite.R")
biocLite("affy")
```

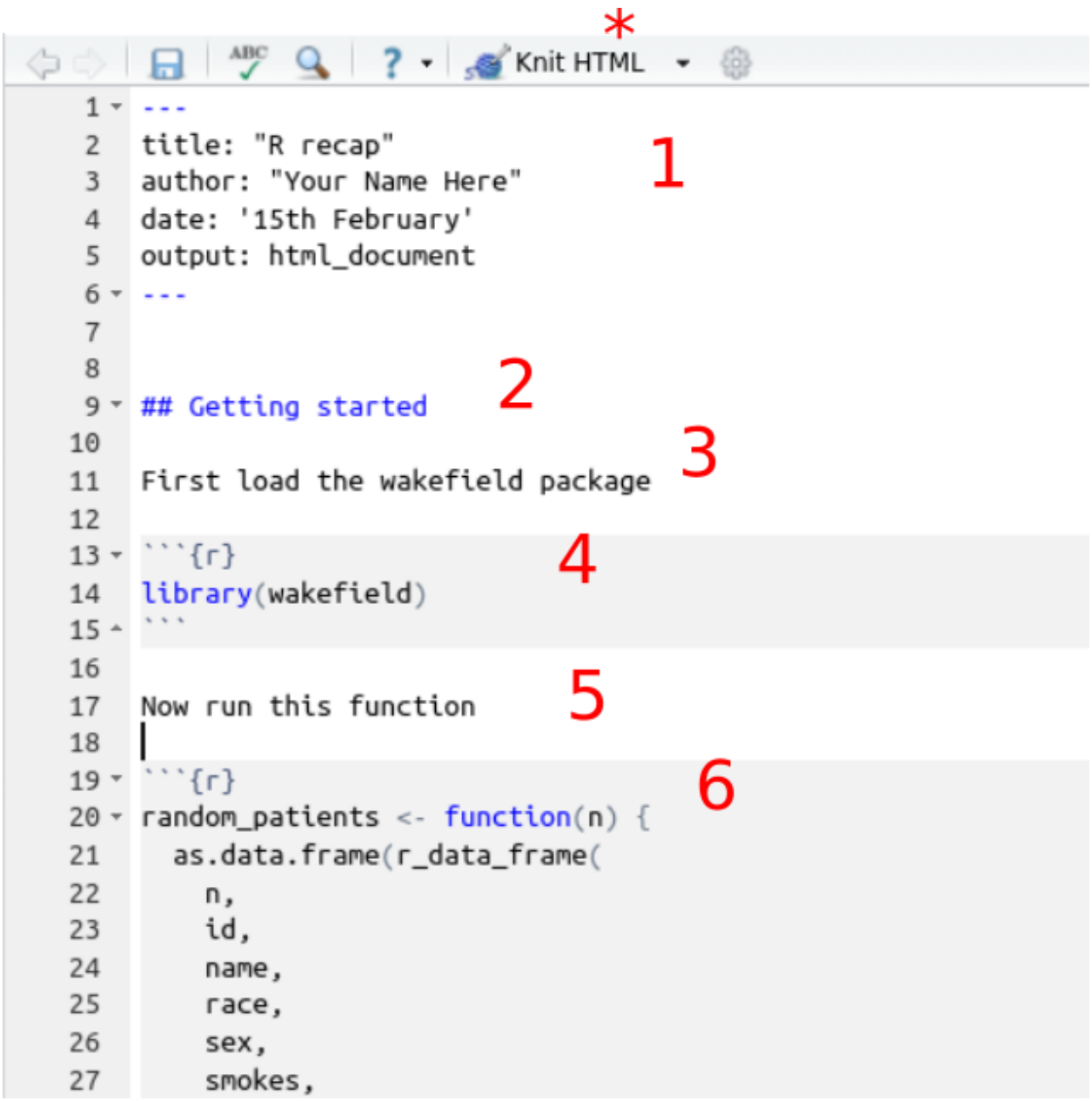
A package may have several *dependancies*; other R packages from which it uses functions or data types (re-using code from other packages is strongly-encouraged). If this is the case, the other R packages will be located and installed too.

So long as you stick with the same version of R, you won't need to repeat this install process.

About the R markdown format

Aside from teaching you about RNA-seq and ChIP-seq analysis, we also hope to teach you how to work in a reproducible manner. The first step in this process is to master the R markdown format.

Open the file `session2-template.Rmd` in Rstudio now.....



1. Header information
2. Section heading
3. Plain text
4. R code to be run
5. Plain text
6. R code to be run

Each line of R code can be executed in the R console by placing the cursor on the line and pressing `CTRL + ENTER`. You can also highlight multiple lines of code. NB. You do not need to highlight to the backtick (“‘) symbols. Hitting the knit button (*) will run all R code in order and (providing there are no errors!) you will get a PDF or HTML document. The resultant document will contain all the plain text you wrote, the R code, and any outputs (including graphs, tables etc) that R produced. You can then distribute this document to have a reproducible account of your analysis.

How to use the template

- Change your name, add a title and date in the header section
- Add notes, explanations of code etc in the white space between code chunks. You can add new lines with `ENTER`. Clicking the `?` next to the `Knit HTML` button will give more information about how to format this text. You can introduce **bold** and *italics* for example.
- Some code chunks are left blank. These are for you to write the R code required to answer the questions
- You can try to knit the document at any point to see how it looks

The Practical

Getting started

We are going to explore some of the basic features of R using some patient data; the kind of data that we might encounter in the wild. However, rather than using real-life data we are going to make some up. There is a package called `wakefield` that is particularly convenient for this task.

```
library(wakefield)
```

Various patient characteristics can be generated. The following is a function that uses the package to create a *data frame* with various clinical characteristics. The number of patients we want to simulate is an argument.

Don’t worry about what the function does, you can just paste the following into the R console, or highlight it in the the markdown template and press `CTRL + ENTER` to run.

```
random_patients <- function(n) {
  as.data.frame(r_data_frame(
    n,
    id,
    name,
    race,
    sex,
    smokes,
    height,
    birth(random = TRUE, x = NULL, start = Sys.Date() - 365 * 45, k = 365*2,by = "1 days"),
    state,
    pet,
    grade_level(x=1:3),
    died,
    normal(name="Count"),
    date_stamp)
  )
}
```

We can now use the `random_patients` function to generate a data frame of fictitious patients

```
patients <- random_patients(100)
```

In Rstudio , you can view the contents of this data frame in a tab.

```
View(patients)
```

Q. What are the dimensions of the data frame?

Q. What columns are available?

*** HINT: see the `dim`, `ncol`, `nrow` and `colnames` functions

```
## [1] 10 13
```

```
##  [1] "ID"          "Name"        "Race"        "Sex"         "Smokes"
##  [6] "Height"      "Birth"       "State"       "Pet"         "Grade_Level"
## [11] "Died"        "Count"       "Date"
```

Q. Can you think of two ways to access the Names of the patients?

Q. What type of object is returned?

```
##  [1] "Britt"  "Martin" "Young"  "Deon"   "Juan"   "Devon"  "Adam"
##  [8] "Cary"   "Yong"   "Clyde"
```

```
##  [1] "Britt"  "Martin" "Young"  "Deon"   "Juan"   "Devon"  "Adam"
##  [8] "Cary"   "Yong"   "Clyde"
```

We can access the columns of a data frame by either

- knowing the column index
- knowing the column name

By column name is recommended, unless you can guarentee the columns will also be in the same order

TOP TIP: Use auto-complete with the key to get the name of the column correct

A vector (1-dimensional) is returned, the length of which is the same as the number of rows in the data frame. The vector could be stored as a variable and itself be subset or used in further calculations

```
peeps <- patients$Name
peeps
```

```
##  [1] "Britt"  "Martin" "Young"  "Deon"   "Juan"   "Devon"  "Adam"
##  [8] "Cary"   "Yong"   "Clyde"
```

```
length(peeps)
```

```
## [1] 10
```

```
nchar(peeps)
```

```
##  [1] 5 6 5 4 4 5 4 4 4 5
```

```
substr(peeps,1,3)
```

```
##  [1] "Bri" "Mar" "You" "Deo" "Jua" "Dev" "Ada" "Car" "Yon" "Cly"
```

The `summary` function is a useful way of summarising the data containing in each column. It will give information about the *type* of data (remember, data frames can have a mixture of numeric and character columns) and also an appropriate summary. For numeric columns, it will report some stats about the distribution of the data. For categorical data, it will report the different *levels*.

```
summary(patients)
```

```
##       ID                 Name                  Race       Sex
##  Length:10          Length:10          White    :7   Male  :6
##  Class :character   Class :character   Hispanic :2   Female:4
##  Mode  :character   Mode  :character   Black    :1
##                                        Asian    :0
##                                        Bi-Racial:0
##                                        Native   :0
##                                        (Other)  :0
##    Smokes            Height         Birth                    State
##  Mode :logical   Min.   :65.0   Min.   :1971-09-01   New York    :2
##  FALSE:7         1st Qu.:67.0   1st Qu.:1971-10-12   Pennsylvania:2
##  TRUE :3         Median :68.0   Median :1972-05-26   Colorado    :1
##  NA's :0         Mean   :69.4   Mean   :1972-04-07   Georgia     :1
##                  3rd Qu.:71.0   3rd Qu.:1972-07-04   Indiana     :1
##                  Max.   :77.0   Max.   :1973-02-02   Missouri    :1
##                                                      (Other)     :2
##     Pet    Grade_Level    Died             Count
##  Dog  :5   1:2         Mode :logical   Min.   :-1.0275
##  Cat  :3   2:2         FALSE:4         1st Qu.:-0.3792
##  None :2   3:6         TRUE :6         Median : 0.2978
##  Bird :0               NA's :0         Mean   : 0.4072
##  Horse:0                               3rd Qu.: 0.9062
##                                        Max.   : 2.4957
##
##       Date
```

```
##  Min.   :2015-05-16
##  1st Qu.:2015-08-23
##  Median :2015-10-16
##  Mean   :2015-10-22
##  3rd Qu.:2016-01-08
##  Max.   :2016-02-16
##
```

Q. Can you identify

which columns contain numerical data?

which columns contain categorical data?

which columns contain logical (`TRUE` or `FALSE`) values?

Subsetting

A data frame can be subset using square brackes `[]` placed after the name of the data frame. As a data frame is a two-dimensional object, you need a *row* and *column* index, or vector indices.

Q. Make sure you can understand the behaviour of the following commands

---

```
patients[1,2]
patients[2,1]
patients[c(1,2,3),1]
patients[c(1,2,3),c(1,2,3)]
```

Note that the data frame is not altered we are just seeing what a subset of the data looks like and not changing the underlying data. If we wanted to do this, we would need to create a new variale.

```
patients
```

```
##    ID   Name     Race    Sex Smokes Height      Birth        State  Pet
## 1  01  Britt    White   Male   TRUE     71 1972-06-21    Wisconsin  Cat
## 2  02 Martin    White   Male  FALSE     68 1973-02-02     Colorado None
## 3  03  Young    White Female  FALSE     67 1971-10-04 Pennsylvania  Dog
## 4  04   Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia  Cat
## 5  05   Juan Hispanic   Male   TRUE     65 1972-04-30     New York  Dog
## 6  06  Devon    White   Male   TRUE     71 1972-06-29     Missouri  Dog
## 7  07   Adam    Black Female  FALSE     68 1972-10-23     New York  Dog
## 8  08   Cary    White   Male  FALSE     66 1971-09-01      Indiana None
```

```
## 9  09   Yong    White Female  FALSE     74 1971-09-12         Ohio  Cat
## 10 10  Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania  Dog
##    Grade_Level  Died      Count       Date
## 1            3  TRUE  2.0429659 2015-05-16
## 2            2 FALSE -0.3968310 2015-07-16
## 3            3  TRUE  0.9995400 2015-08-16
## 4            1  TRUE -0.3263563 2015-09-16
## 5            3 FALSE  0.1008862 2015-10-16
## 6            3 FALSE  2.4956672 2015-10-16
## 7            3  TRUE -0.9369304 2015-12-16
## 8            3  TRUE -1.0275402 2016-01-16
## 9            1 FALSE  0.6261153 2016-02-16
## 10           2  TRUE  0.4947157 2016-02-16
```

Should we wish to see all rows, or all columns, we can neglect either the row or column index

---

Q. Make sure you can understand the behaviour of the following commands

---

```
patients[1,]
```

```
##   ID  Name  Race  Sex Smokes Height      Birth     State Pet Grade_Level
## 1 01 Britt White Male   TRUE     71 1972-06-21 Wisconsin Cat           3
##   Died    Count       Date
## 1 TRUE 2.042966 2015-05-16
```

```
patients[,1]
```

```
##  [1] "01" "02" "03" "04" "05" "06" "07" "08" "09" "10"
```

```
patients[,c(1,2)]
```

```
##    ID   Name
## 1  01  Britt
## 2  02 Martin
## 3  03  Young
## 4  04   Deon
## 5  05   Juan
## 6  06  Devon
## 7  07   Adam
## 8  08   Cary
## 9  09   Yong
## 10 10  Clyde
```

Q. How can we view all information about the first six patients?

*** HINT `head` is commonly-used to give a snapshot of a data frame. Otherwise, you can use the `[row,column]` notation.

```
##   ID   Name     Race    Sex Smokes Height      Birth        State  Pet
## 1 01  Britt    White   Male   TRUE     71 1972-06-21    Wisconsin  Cat
## 2 02 Martin    White   Male  FALSE     68 1973-02-02     Colorado None
## 3 03  Young    White Female  FALSE     67 1971-10-04 Pennsylvania  Dog
## 4 04   Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia  Cat
## 5 05   Juan Hispanic   Male   TRUE     65 1972-04-30     New York  Dog
## 6 06  Devon    White   Male   TRUE     71 1972-06-29     Missouri  Dog
##   Grade_Level  Died      Count       Date
## 1           3  TRUE  2.0429659 2015-05-16
## 2           2 FALSE -0.3968310 2015-07-16
## 3           3  TRUE  0.9995400 2015-08-16
## 4           1  TRUE -0.3263563 2015-09-16
## 5           3 FALSE  0.1008862 2015-10-16
## 6           3 FALSE  2.4956672 2015-10-16
```

Rather than selecting rows based on their *numeric* index (as in the previous example) we can use what we call a *logical test*. This is a test that gives either a `TRUE` or `FALSE` result. When applied to subsetting, only rows with a `TRUE` result get returned.

For example we could compare the `Count` variable to zero. The result is a *vector* of `TRUE` or `FALSE`; one for each row in the data frame

```
patients$Count < 0
```

```
##  [1] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE
```

This R code can be put inside the square brackets.

```
patients[patients$Count<0, ]
```

```
##   ID   Name     Race    Sex Smokes Height      Birth    State  Pet
## 2 02 Martin    White   Male  FALSE     68 1973-02-02 Colorado None
## 4 04   Deon Hispanic   Male  FALSE     77 1972-07-06  Georgia  Cat
## 7 07   Adam    Black Female  FALSE     68 1972-10-23 New York  Dog
## 8 08   Cary    White   Male  FALSE     66 1971-09-01  Indiana None
##   Grade_Level  Died      Count       Date
## 2           2 FALSE -0.3968310 2015-07-16
## 4           1  TRUE -0.3263563 2015-09-16
## 7           3  TRUE -0.9369304 2015-12-16
## 8           3  TRUE -1.0275402 2016-01-16
```

If we wanted to know about the patients that had died, we could do;

```
deceased <- patients[patients$Died == TRUE,]
deceased
```

```
##    ID  Name     Race    Sex Smokes Height      Birth        State  Pet
## 1  01 Britt    White   Male   TRUE     71 1972-06-21    Wisconsin  Cat
## 3  03 Young    White Female  FALSE     67 1971-10-04 Pennsylvania  Dog
## 4  04  Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia  Cat
## 7  07  Adam    Black Female  FALSE     68 1972-10-23     New York  Dog
## 8  08  Cary    White   Male  FALSE     66 1971-09-01      Indiana None
## 10 10 Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania  Dog
##    Grade_Level Died      Count       Date
## 1            3 TRUE  2.0429659 2015-05-16
## 3            3 TRUE  0.9995400 2015-08-16
## 4            1 TRUE -0.3263563 2015-09-16
## 7            3 TRUE -0.9369304 2015-12-16
## 8            3 TRUE -1.0275402 2016-01-16
## 10           2 TRUE  0.4947157 2016-02-16
```

In fact, this is equivalent

```
deceased <- patients[patients$Died,]
```

The test of equality == also works for text

```
patients[patients$Race == "White",]
```

```
##    ID   Name  Race    Sex Smokes Height      Birth        State  Pet
## 1  01  Britt White   Male   TRUE     71 1972-06-21    Wisconsin  Cat
## 2  02 Martin White   Male  FALSE     68 1973-02-02     Colorado None
## 3  03  Young White Female  FALSE     67 1971-10-04 Pennsylvania  Dog
## 6  06  Devon White   Male   TRUE     71 1972-06-29     Missouri  Dog
## 8  08   Cary White   Male  FALSE     66 1971-09-01      Indiana None
## 9  09   Yong White Female  FALSE     74 1971-09-12         Ohio  Cat
## 10 10  Clyde White Female  FALSE     67 1971-11-08 Pennsylvania  Dog
##    Grade_Level  Died      Count       Date
## 1            3  TRUE  2.0429659 2015-05-16
## 2            2 FALSE -0.3968310 2015-07-16
## 3            3  TRUE  0.9995400 2015-08-16
## 6            3 FALSE  2.4956672 2015-10-16
## 8            3  TRUE -1.0275402 2016-01-16
## 9            1 FALSE  0.6261153 2016-02-16
## 10           2  TRUE  0.4947157 2016-02-16
```

Q. Can you create a data frame of dog owners?

```
##    ID  Name     Race    Sex Smokes Height      Birth        State Pet
```

```
## 3  03 Young    White Female  FALSE     67 1971-10-04 Pennsylvania Dog
## 5  05  Juan Hispanic   Male   TRUE     65 1972-04-30     New York Dog
## 6  06 Devon    White   Male   TRUE     71 1972-06-29     Missouri Dog
## 7  07  Adam    Black Female  FALSE     68 1972-10-23     New York Dog
## 10 10 Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania Dog
##    Grade_Level  Died      Count       Date
## 3            3  TRUE  0.9995400 2015-08-16
## 5            3 FALSE  0.1008862 2015-10-16
## 6            3 FALSE  2.4956672 2015-10-16
## 7            3  TRUE -0.9369304 2015-12-16
## 10           2  TRUE  0.4947157 2016-02-16
```

There are a couple of ways of testing for more than one text value. The first uses an *or* | statement. i.e. testing if the value of `Pet` is `Dog` *or* the value is `Cat`.

The `%in%` function is a convenient function for testing which items in a vector correspond to a defined set of values.

```
patients[patients$Pet == "Dog" | patients$Pet == "Cat",]
```

```
##    ID  Name     Race    Sex Smokes Height      Birth        State Pet
## 1  01 Britt    White   Male   TRUE     71 1972-06-21    Wisconsin Cat
## 3  03 Young    White Female  FALSE     67 1971-10-04 Pennsylvania Dog
## 4  04  Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia Cat
## 5  05  Juan Hispanic   Male   TRUE     65 1972-04-30     New York Dog
## 6  06 Devon    White   Male   TRUE     71 1972-06-29     Missouri Dog
## 7  07  Adam    Black Female  FALSE     68 1972-10-23     New York Dog
## 9  09  Yong    White Female  FALSE     74 1971-09-12         Ohio Cat
## 10 10 Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania Dog
##    Grade_Level  Died      Count       Date
## 1            3  TRUE  2.0429659 2015-05-16
## 3            3  TRUE  0.9995400 2015-08-16
## 4            1  TRUE -0.3263563 2015-09-16
## 5            3 FALSE  0.1008862 2015-10-16
## 6            3 FALSE  2.4956672 2015-10-16
## 7            3  TRUE -0.9369304 2015-12-16
## 9            1 FALSE  0.6261153 2016-02-16
## 10           2  TRUE  0.4947157 2016-02-16
```

```
patients[patients$Pet %in% c("Dog","Cat"),]
```

```
##    ID  Name     Race    Sex Smokes Height      Birth        State Pet
## 1  01 Britt    White   Male   TRUE     71 1972-06-21    Wisconsin Cat
## 3  03 Young    White Female  FALSE     67 1971-10-04 Pennsylvania Dog
## 4  04  Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia Cat
## 5  05  Juan Hispanic   Male   TRUE     65 1972-04-30     New York Dog
## 6  06 Devon    White   Male   TRUE     71 1972-06-29     Missouri Dog
## 7  07  Adam    Black Female  FALSE     68 1972-10-23     New York Dog
## 9  09  Yong    White Female  FALSE     74 1971-09-12         Ohio Cat
## 10 10 Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania Dog
##    Grade_Level  Died      Count       Date
## 1            3  TRUE  2.0429659 2015-05-16
## 3            3  TRUE  0.9995400 2015-08-16
```

```
## 4            1  TRUE -0.3263563 2015-09-16
## 5            3 FALSE  0.1008862 2015-10-16
## 6            3 FALSE  2.4956672 2015-10-16
## 7            3  TRUE -0.9369304 2015-12-16
## 9            1 FALSE  0.6261153 2016-02-16
## 10           2  TRUE  0.4947157 2016-02-16
```

Similar to *or*, we can require that both tests are `TRUE` by using an *and* `&` operation. e.g. to look for white males.

```
patients[patients$Race == "White" & patients$Sex =="Male",]
```

```
##   ID   Name  Race  Sex Smokes Height      Birth     State  Pet Grade_Level
## 1 01  Britt White Male   TRUE     71 1972-06-21 Wisconsin  Cat           3
## 2 02 Martin White Male  FALSE     68 1973-02-02  Colorado None           2
## 6 06  Devon White Male   TRUE     71 1972-06-29  Missouri  Dog           3
## 8 08   Cary White Male  FALSE     66 1971-09-01   Indiana None           3
##    Died     Count       Date
## 1  TRUE  2.042966 2015-05-16
## 2 FALSE -0.396831 2015-07-16
## 6 FALSE  2.495667 2015-10-16
## 8  TRUE -1.027540 2016-01-16
```

Q. Can you create a data frame of deceased patients with a 'count' < 0

```
##   ID Name     Race    Sex Smokes Height      Birth    State  Pet
## 4 04 Deon Hispanic   Male  FALSE     77 1972-07-06  Georgia  Cat
## 7 07 Adam    Black Female  FALSE     68 1972-10-23 New York  Dog
## 8 08 Cary    White   Male  FALSE     66 1971-09-01  Indiana None
##   Grade_Level Died      Count       Date
## 4           1 TRUE -0.3263563 2015-09-16
## 7           3 TRUE -0.9369304 2015-12-16
## 8           3 TRUE -1.0275402 2016-01-16
```

Ordering and sorting

A vector can be returned in sorted form using the `sort` function.

```
sort(peeps)
```

```
##  [1] "Adam"   "Britt"  "Cary"   "Clyde"  "Deon"   "Devon"  "Juan"
##  [8] "Martin" "Yong"   "Young"
```

```
sort(patients$Count,decreasing = TRUE)
```

```
##  [1]  2.4956672  2.0429659  0.9995400  0.6261153  0.4947157  0.1008862
##  [7] -0.3263563 -0.3968310 -0.9369304 -1.0275402
```

However, if we want to sort an entire data frame a different approach is needed. The trick is to use `order`. Rather than giving a sorted set of *values*, it will give sorted *indices*.

```
patients[order(patients$Count),]
```

```
##    ID   Name     Race    Sex Smokes Height      Birth        State  Pet
## 8  08   Cary    White   Male  FALSE     66 1971-09-01      Indiana None
## 7  07   Adam    Black Female  FALSE     68 1972-10-23     New York  Dog
## 2  02 Martin    White   Male  FALSE     68 1973-02-02     Colorado None
## 4  04   Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia  Cat
## 5  05   Juan Hispanic   Male   TRUE     65 1972-04-30     New York  Dog
## 10 10  Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania  Dog
## 9  09   Yong    White Female  FALSE     74 1971-09-12         Ohio  Cat
## 3  03  Young    White Female  FALSE     67 1971-10-04 Pennsylvania  Dog
## 1  01  Britt    White   Male   TRUE     71 1972-06-21    Wisconsin  Cat
## 6  06  Devon    White   Male   TRUE     71 1972-06-29     Missouri  Dog
##    Grade_Level  Died      Count       Date
## 8            3  TRUE -1.0275402 2016-01-16
## 7            3  TRUE -0.9369304 2015-12-16
## 2            2 FALSE -0.3968310 2015-07-16
## 4            1  TRUE -0.3263563 2015-09-16
## 5            3 FALSE  0.1008862 2015-10-16
## 10           2  TRUE  0.4947157 2016-02-16
## 9            1 FALSE  0.6261153 2016-02-16
## 3            3  TRUE  0.9995400 2015-08-16
## 1            3  TRUE  2.0429659 2015-05-16
## 6            3 FALSE  2.4956672 2015-10-16
```

```
patients[order(patients$Sex),]
```

```
##    ID   Name     Race    Sex Smokes Height      Birth        State  Pet
## 1  01  Britt    White   Male   TRUE     71 1972-06-21    Wisconsin  Cat
## 2  02 Martin    White   Male  FALSE     68 1973-02-02     Colorado None
## 4  04   Deon Hispanic   Male  FALSE     77 1972-07-06      Georgia  Cat
## 5  05   Juan Hispanic   Male   TRUE     65 1972-04-30     New York  Dog
## 6  06  Devon    White   Male   TRUE     71 1972-06-29     Missouri  Dog
## 8  08   Cary    White   Male  FALSE     66 1971-09-01      Indiana None
## 3  03  Young    White Female  FALSE     67 1971-10-04 Pennsylvania  Dog
## 7  07   Adam    Black Female  FALSE     68 1972-10-23     New York  Dog
## 9  09   Yong    White Female  FALSE     74 1971-09-12         Ohio  Cat
## 10 10  Clyde    White Female  FALSE     67 1971-11-08 Pennsylvania  Dog
##    Grade_Level  Died      Count       Date
## 1            3  TRUE  2.0429659 2015-05-16
## 2            2 FALSE -0.3968310 2015-07-16
## 4            1  TRUE -0.3263563 2015-09-16
## 5            3 FALSE  0.1008862 2015-10-16
## 6            3 FALSE  2.4956672 2015-10-16
## 8            3  TRUE -1.0275402 2016-01-16
## 3            3  TRUE  0.9995400 2015-08-16
```

```
## 7            3  TRUE -0.9369304 2015-12-16
## 9            1 FALSE  0.6261153 2016-02-16
## 10           2  TRUE  0.4947157 2016-02-16
```

A final point on data frames is that we can export them out of R once we have done our data processing.

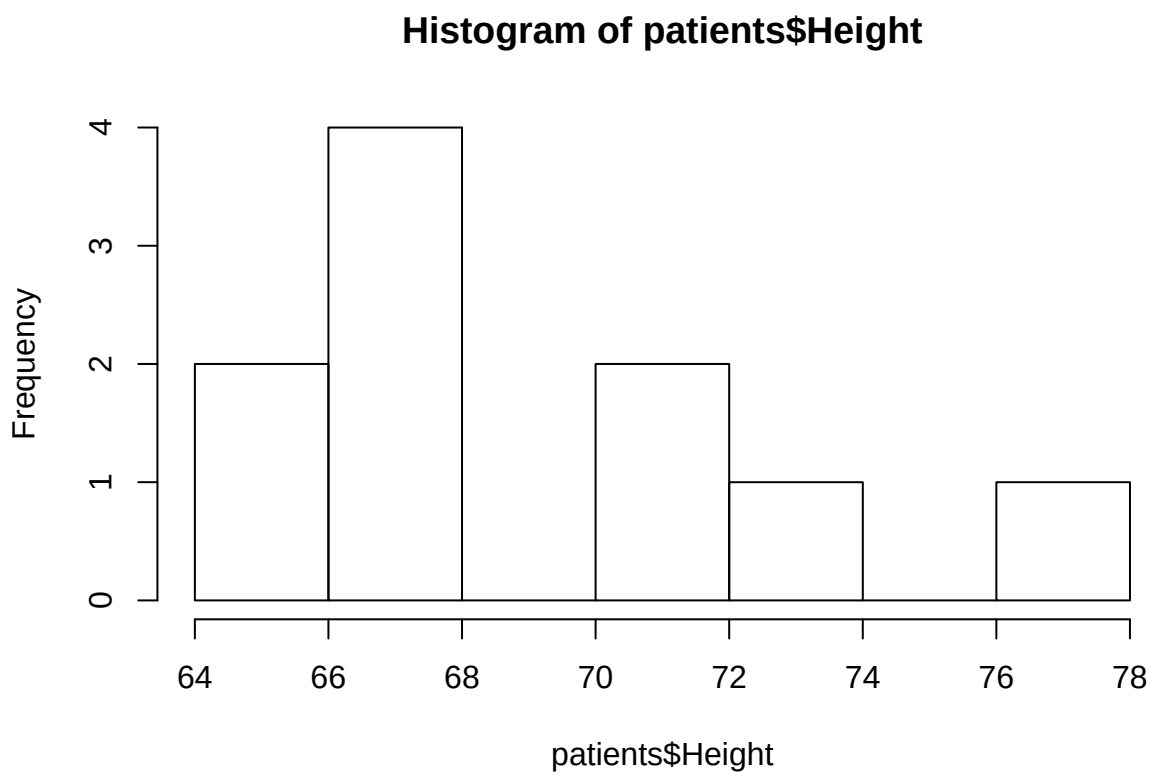
```
countOrder <- patients[order(patients$Count),]
write.csv(countOrder, file="patientsOrderedByCount.csv")
```

Simple plotting

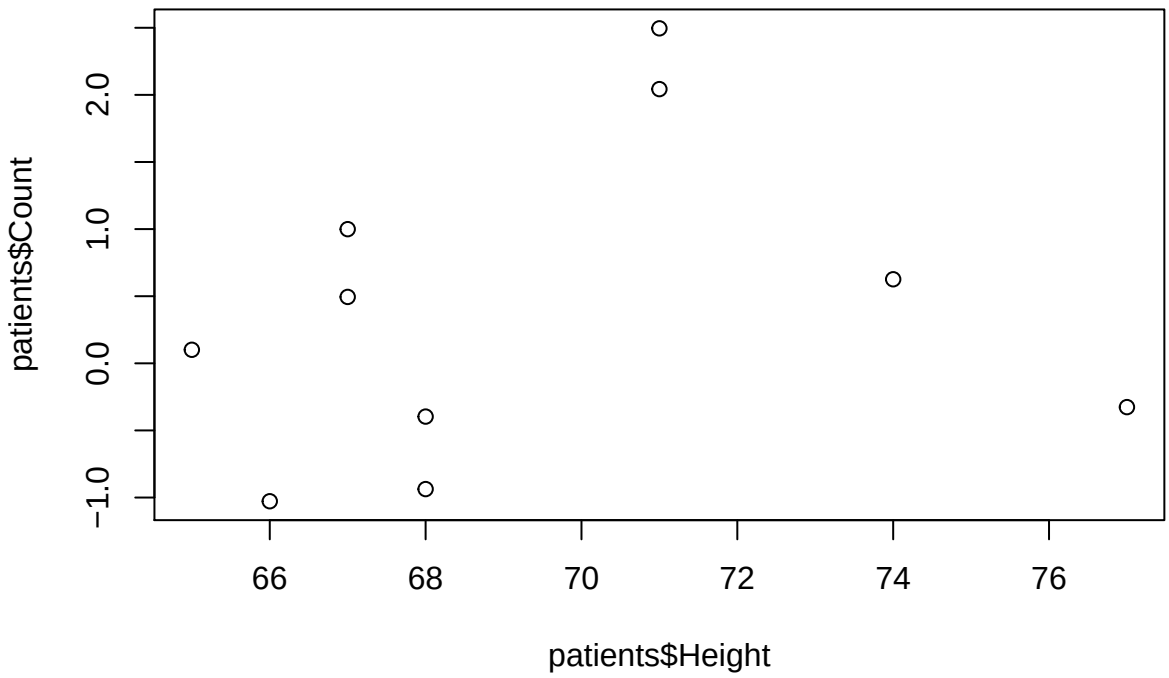
Various simple plots are supported in the *base* distribution of R (what you get automatically when you download R). In the course, we will show how some of these plots can be used to inform us about the quality of NGS data, and to visualise our results.

Plotting is discussed in greater length on our introductory R course and a useful reference is the Quick-R page.

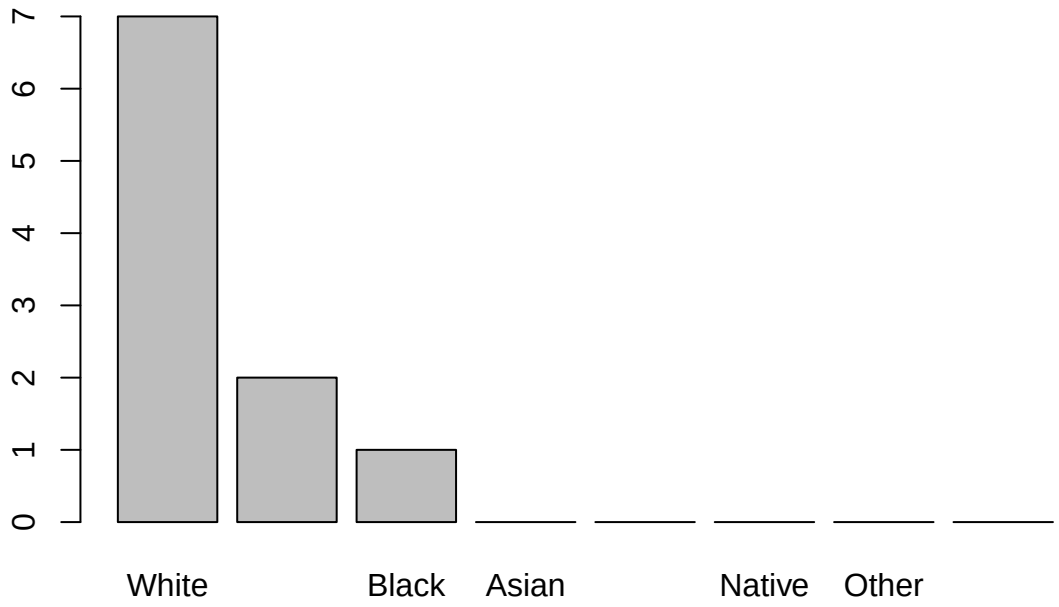
```
hist(patients$Height)
```



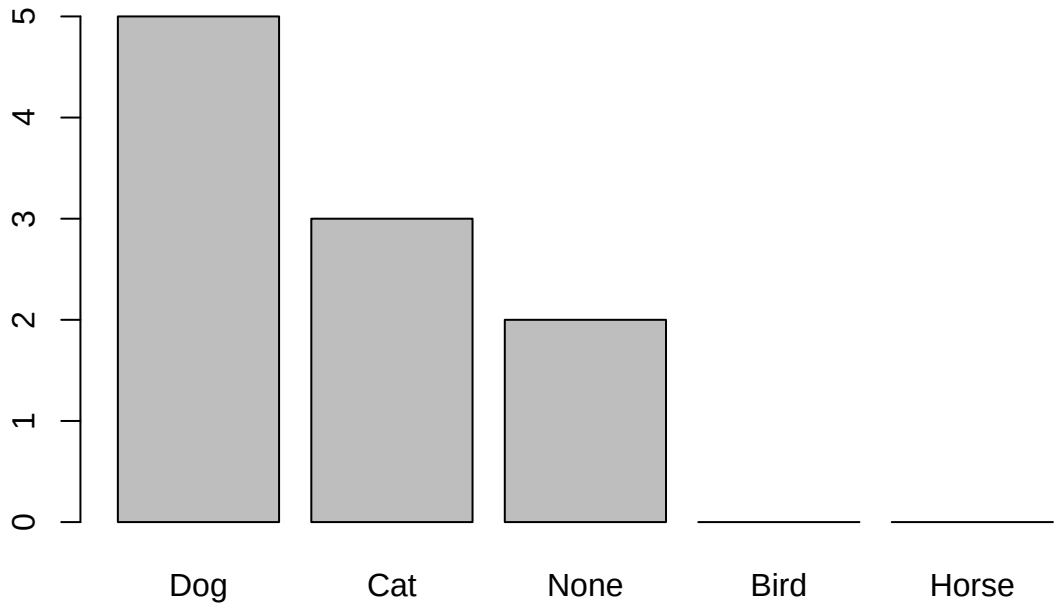

```
plot(patients$Height,patients$Count)
```



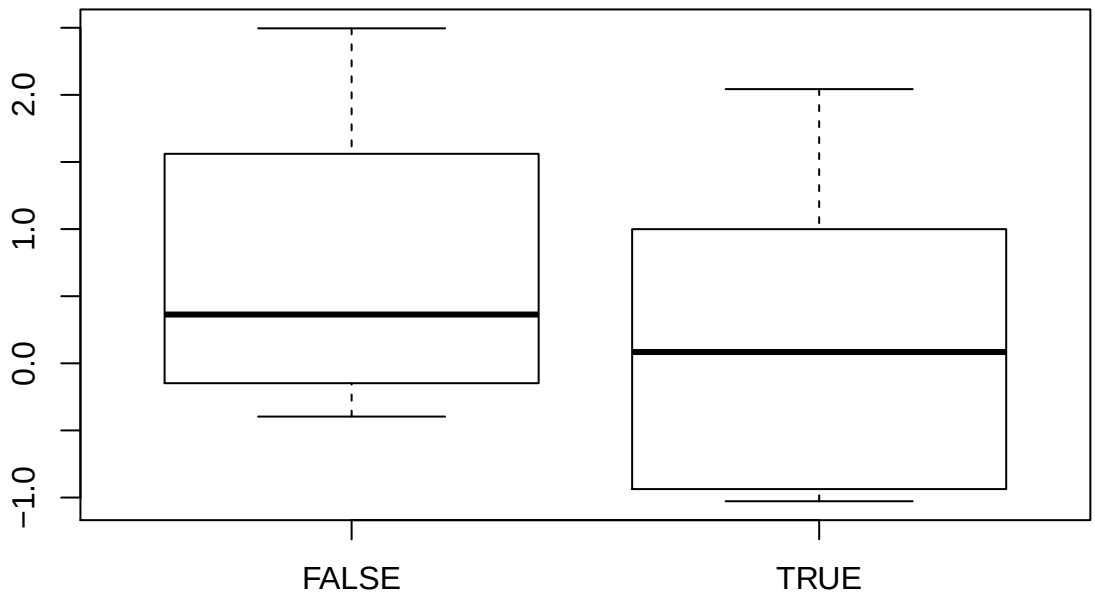

```
barplot(table(patients$Race))
```




```
barplot(table(patients$Pet))
```

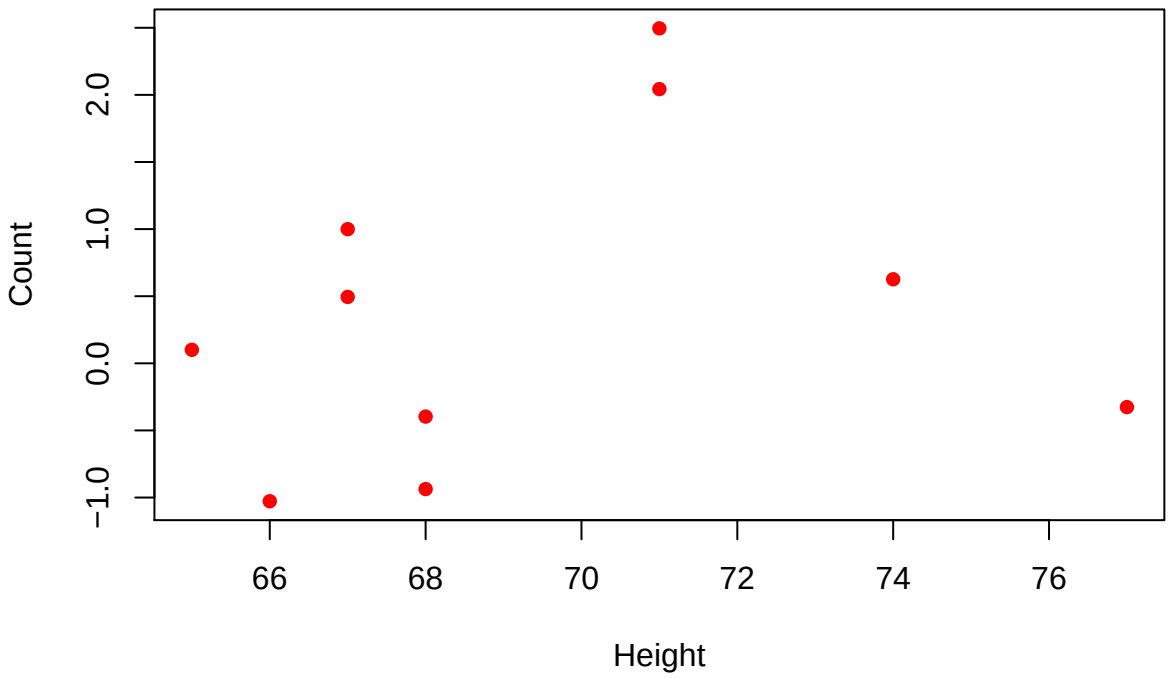



```
boxplot(patients$Count ~ patients$Died)
```

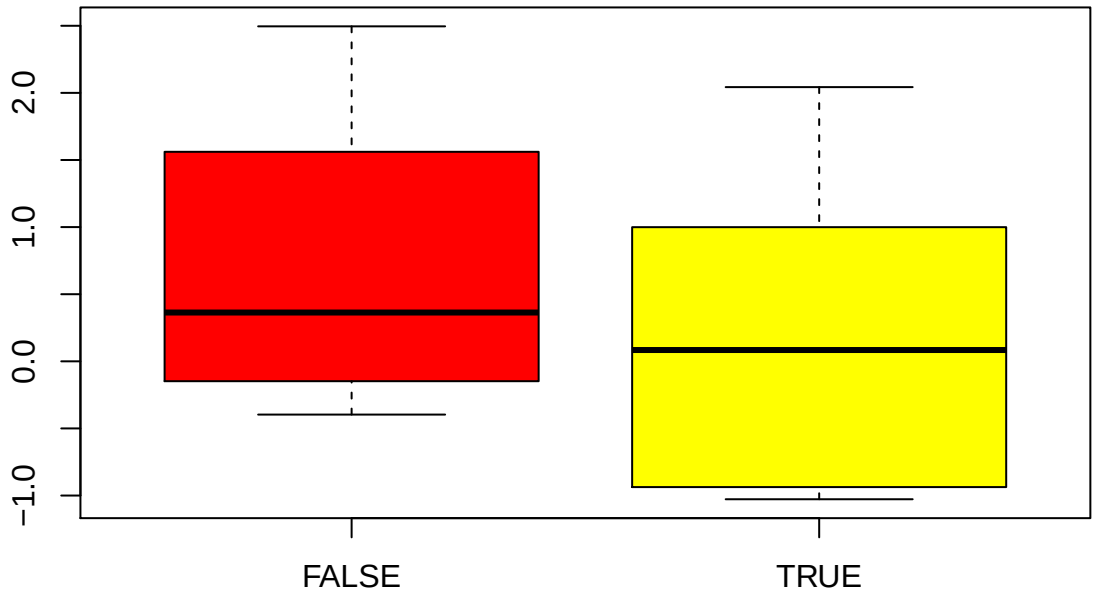


Lots of customisations are possible to enhance the appaerance of our plots; colour, labels, axes, legends

```
plot(patients$Height,patients$Count,pch=16,
     col="red",xlab="Height",
     ylab="Count")
```




```
boxplot(patients$Count ~ patients$Died,col=c("red","yellow"))
```



Plots can be exported by the ***Plots*** tab in RStudio, or by calling the `pdf` or `png` functions which will write the plot to a file

```
png("myLittlePlot.png")
barplot(table(patients$Pet))
dev.off()
```

```
## pdf
##   2
```