Introduction to Solving Biological Problems Using R - Day 2

Mark Dunning, Suraj Menon, and Aiora Zabala. Original material by Robert Stojnić, Laurent Gatto, Rob Foy John Davey, Dávid Molnár and Ian Roberts

Last modified: 14 Dec 2015

true

Day 2 Schedule

1. Further customisation of plots
2. Statistics
3. Data Manipulation Techniques
4. Programming in R
5. Further report writing

1. Further customisation of plots

Recap

- We have seen how to use `plot` , `boxplot` , `hist` etc to make simple plots
- These come with arguments that can be used to change the appearance of the plot
    - `col` , `pch`
    - `main` , `xlab` , `ylab`
    - etc….
- We will now look at ways to modify the plot appearance after it has been created
- Also, how to export the graphs

The painter’s model

- R employs a painter’s model to construct it’s plots
- Elements of the graph are added to the canvas one layer at a time, and the picture built up in levels.
- Lower levels are obscured by higher levels,
    - allowing for blending, masking and overlaying of objects.

Example data

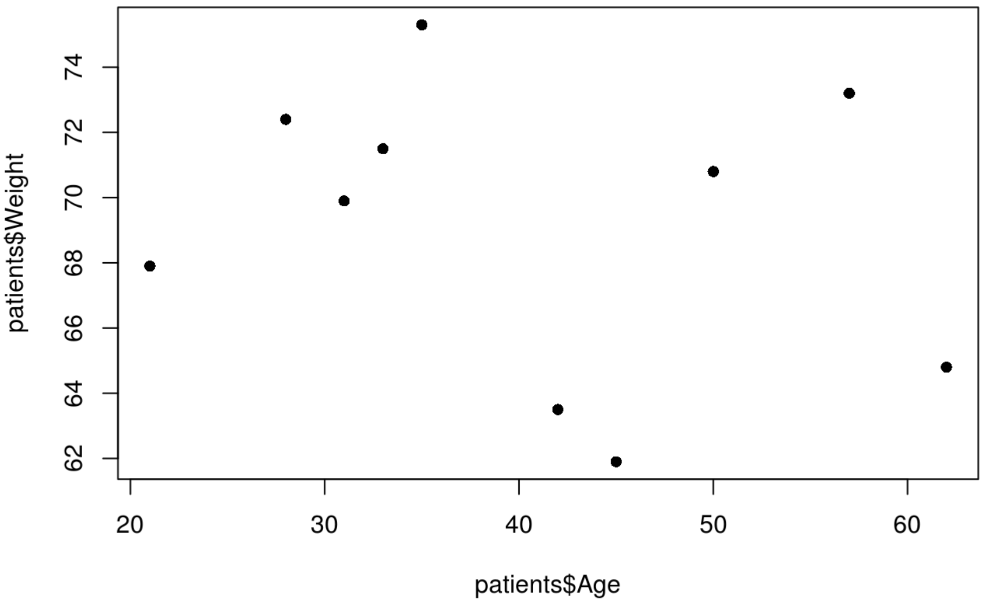
- We will re-use the patients data from yesterday

```
age    <- c(50, 21, 35, 45, 28, 31, 42, 33, 57, 62)
weight <- c(70.8, 67.9, 75.3, 61.9, 72.4, 69.9, 63.5,
71.5, 73.2, 64.8)
firstName <- c("Adam", "Eve", "John", "Mary", "Peter",
"Paul", "Joanna", "Matthew", "David", "Sally")
secondName <- c("Jones", "Parker", "Evans", "Davis",
"Baker","Daniels", "Edwards", "Smith", "Roberts", "Wilson")

consent <- c(TRUE, TRUE, FALSE, TRUE, FALSE, FALSE,
FALSE, TRUE, FALSE, TRUE)

sex <- c("Male", "Female", "Male", "Female", "Male", "Male"
,
"Female", "Male", "Male", "Female")

patients <- data.frame(First_Name = firstName,
                       Second_Name = secondName,
                       Full_Name = paste(firstName, secondN
ame),
                       Sex = factor(sex),
                       Age = age,
                       Weight = weight,
                       Consent = consent,
                       stringsAsFactors = FALSE)
```

Initial plot

- Recall our patients dataset from yesterday
  - we might want to display other characteristics on the plot
    - e.g. gender of individual

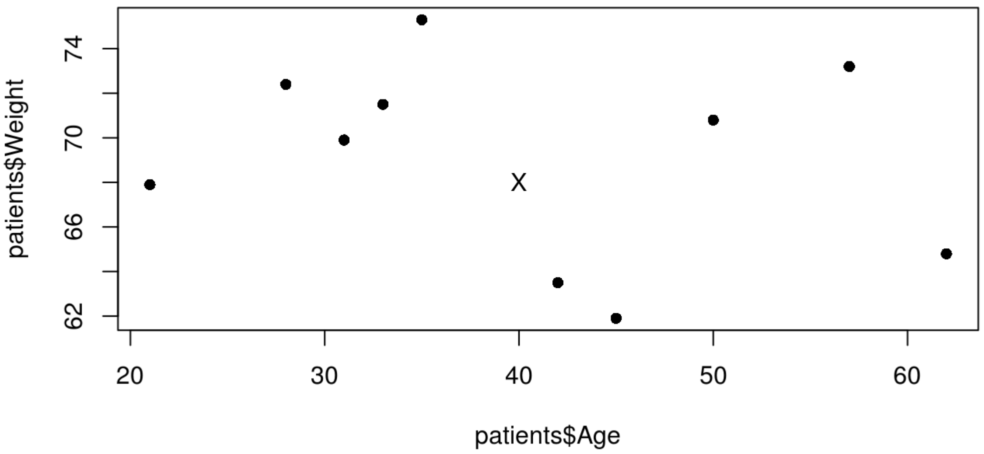
```
plot(patients$Age, patients$Weight,pch=16)
```



The points function

- `points` can be used to set of points to an *existing* plot
- It requires a vector of x and y coordinates
    - these do not have to be the same length as the number of points in the initial plot
        - hence we can use `points` to highlight observations
        - or add a set of new observations
- Note that axis limits of the existing plot are not altered

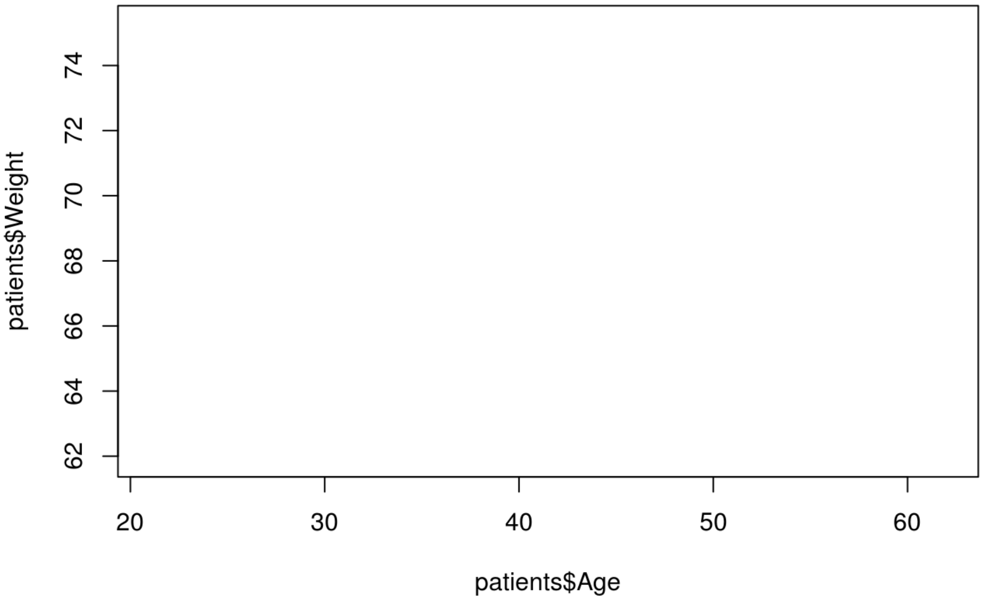
```
plot(patients$Age, patients$Weight,pch=16)
points(40,68,pch="X")
```



Creating a blank plot

- Often it is useful to create a blank ‘canvas’ with the correct labels and limits

```
plot(patients$Age, patients$Weight,type="n")
```



Adding points to differentiate gender

- Selecting males using the `==` comparison we saw yesterday
  - gives a `TRUE` or `FALSE` value
  - can be used to index the data frame
  - which means we can get the relevant Age and Weight values

```
males <- patients$Sex == "Male"
males
```

```
##  [1]  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TR
UE FALSE
```

```
patients[males,]
```

```
##   First_Name Second_Name     Full_Name  Sex Age Weight C
onsent
## 1       Adam       Jones    Adam Jones Male  50   70.8
  TRUE
## 3       John       Evans    John Evans Male  35   75.3
 FALSE
## 5      Peter       Baker   Peter Baker Male  28   72.4
 FALSE
## 6       Paul     Daniels  Paul Daniels Male  31   69.9
 FALSE
## 8    Matthew       Smith Matthew Smith Male  33   71.5
  TRUE
## 9      David     Roberts David Roberts Male  57   73.2
 FALSE
```

```
patients[males,"Age"]
```

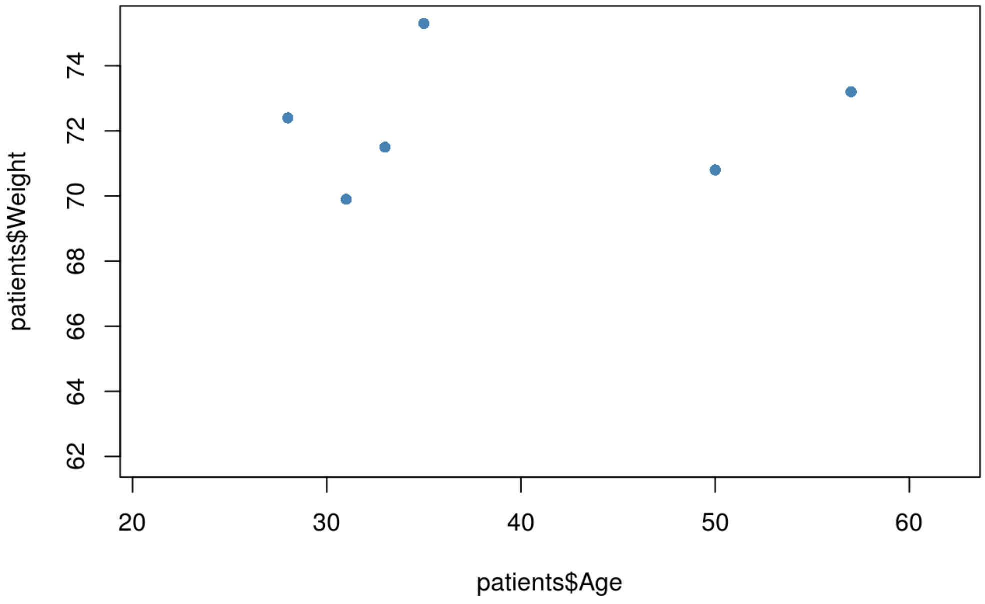
```
## [1] 50 35 28 31 33 57
```

```
patients[males,"Weight"]
```

```
## [1] 70.8 75.3 72.4 69.9 71.5 73.2
```

Adding points to differentiate gender

```
plot(patients$Age, patients$Weight,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,c
ol="steelblue")
```



Adding points to differentiate gender

```
females <- patients$Sex == "Female"
females
```

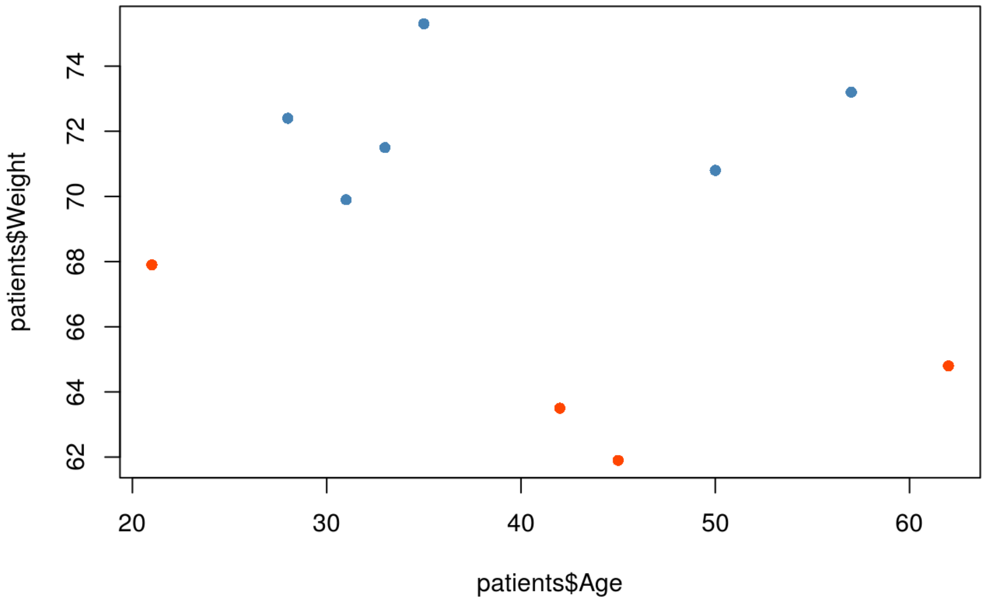
```
##  [1] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FAL
SE  TRUE
```

```
patients[females,]
```

```
##    First_Name Second_Name      Full_Name    Sex Age Weig
ht Consent
## 2         Eve      Parker     Eve Parker Female  21   67
.9    TRUE
## 4        Mary       Davis     Mary Davis Female  45   61
.9    TRUE
## 7      Joanna     Edwards Joanna Edwards Female  42   63
.5   FALSE
## 10      Sally      Wilson   Sally Wilson Female  62   64
.8    TRUE
```

Adding points to differentiate gender

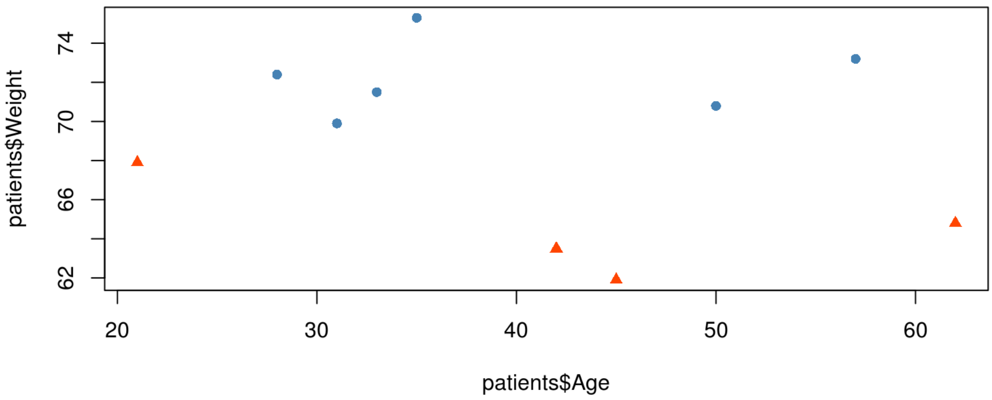
```
plot(patients$Age, patients$Weight,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,c
ol="steelblue")
points(patients$Age[females], patients$Weight[females],pch=
16,col="orangered1")
```



Adding points

- Each set of points can have a different colour and shape
- Axis labels and title and limits are defined by the plot
- You can add points ad-nauseum. Try not to make the plot cluttered!
- Once you've added points to a plot, they cannot be removed
- A call to `plot` will start a new graphics window
  - or typing `dev.off()`

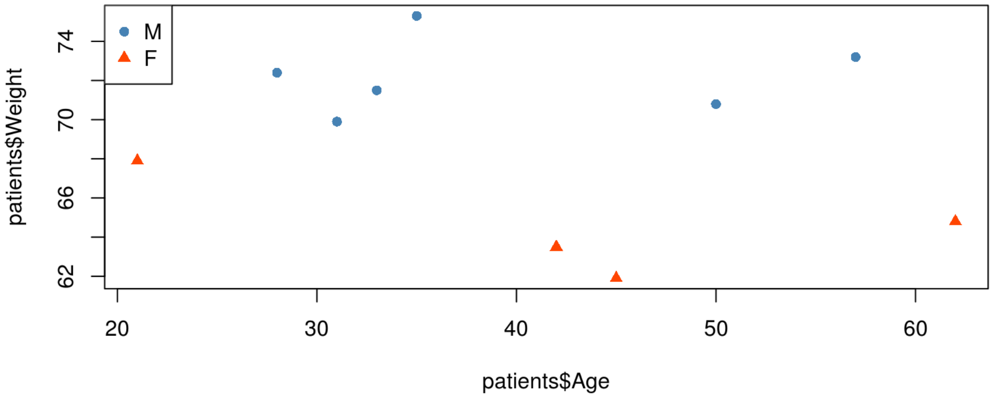
```
plot(patients$Age, patients$Weight,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,c
ol="steelblue")
points(patients$Age[females], patients$Weight[females],pch=
17,col="orangered1")
```



Adding a legend

- Should also add a legend to help interpret the plot
  - use the `legend` function
  - can give x and y coordinates where legend will appear
  - also recognises shortcuts such as ***topleft*** and ***bottomright***…

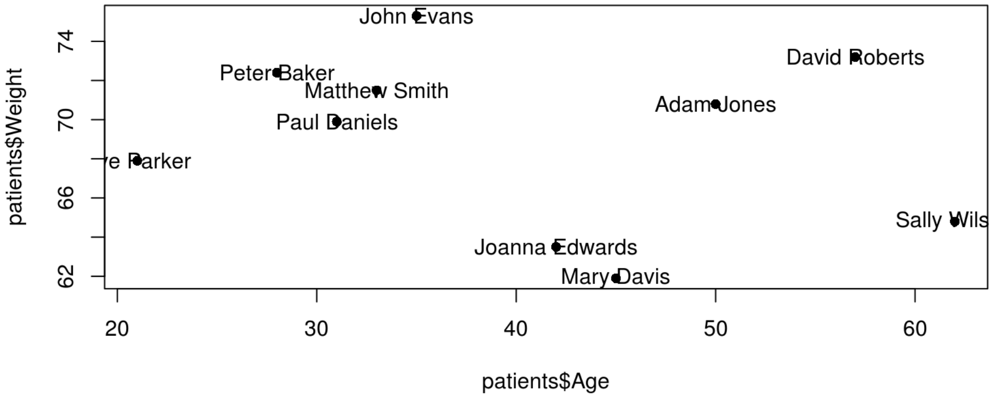
```
plot(patients$Age, patients$Weight,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,c
ol="steelblue")
points(patients$Age[females], patients$Weight[females],pch=
17,col="orangered1")
legend("topleft", legend=c("M","F"),
       col=c("steelblue","orangered1"), pch=c(16,17))
```



Adding text

- Text can also be added to a plot in a similar manner
  - the `labels` argument specifies the text we want to add

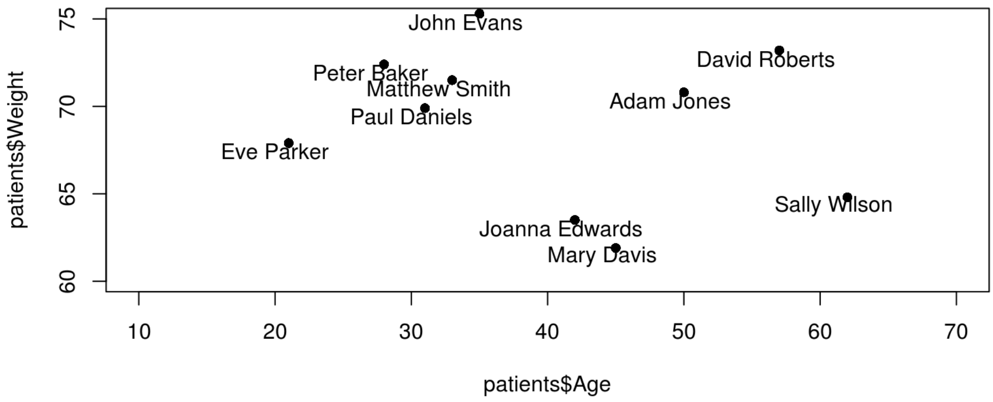
```
plot(patients$Age, patients$Weight,pch=16)
text(patients$Age, patients$Weight,labels=patients$Full_Nam
e)
```



Adding text

- Can alter the positions so they don’t interfere with the points of the graph

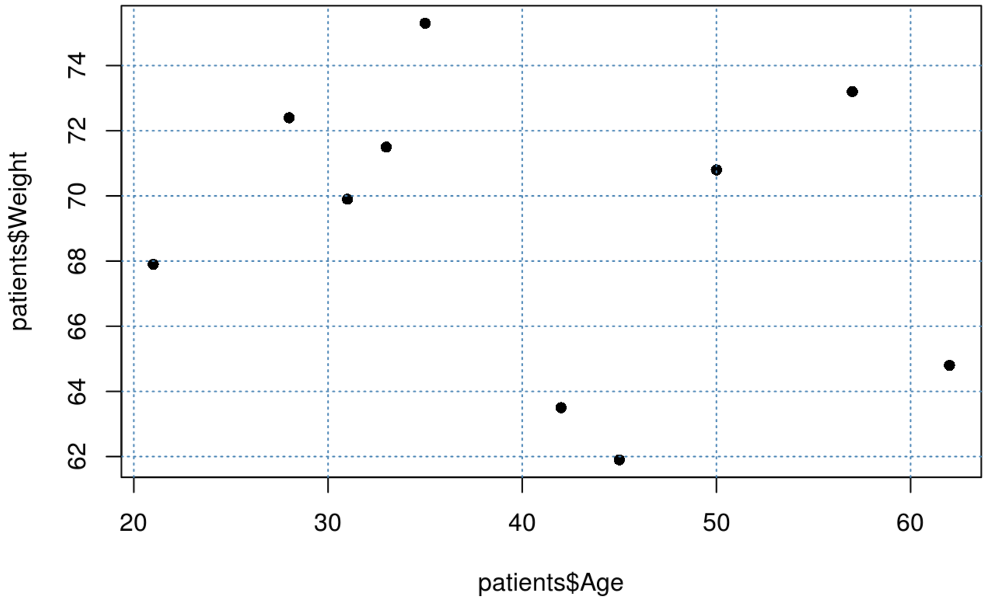
```
plot(patients$Age, patients$Weight,pch=16,xlim=c(10,70),yli
m=c(60,75))
text(patients$Age-1, patients$Weight-0.5,labels=patients$Fu
ll_Name)
```



Adding lines

- To aid our interpretation, it is often helpful to add guidelines
    - `grid()` is one easy way of doing this.

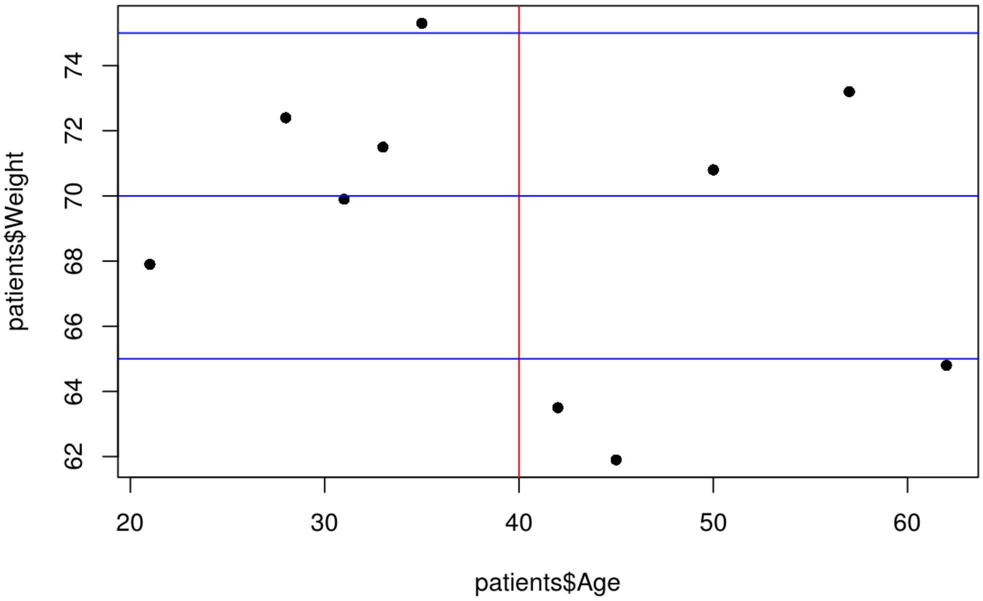
```
plot(patients$Age, patients$Weight,pch=16)
grid(col="steelblue")
```



Adding lines

- Can also add lines that intersect the axes
    - `v =` for vertical lines
    - `h=` for horizontal
    - can specify multiple lines in a vector

```
plot(patients$Age, patients$Weight,pch=16)
abline(v=40,col="red")
abline(h=c(65,70,75),col="blue")
```

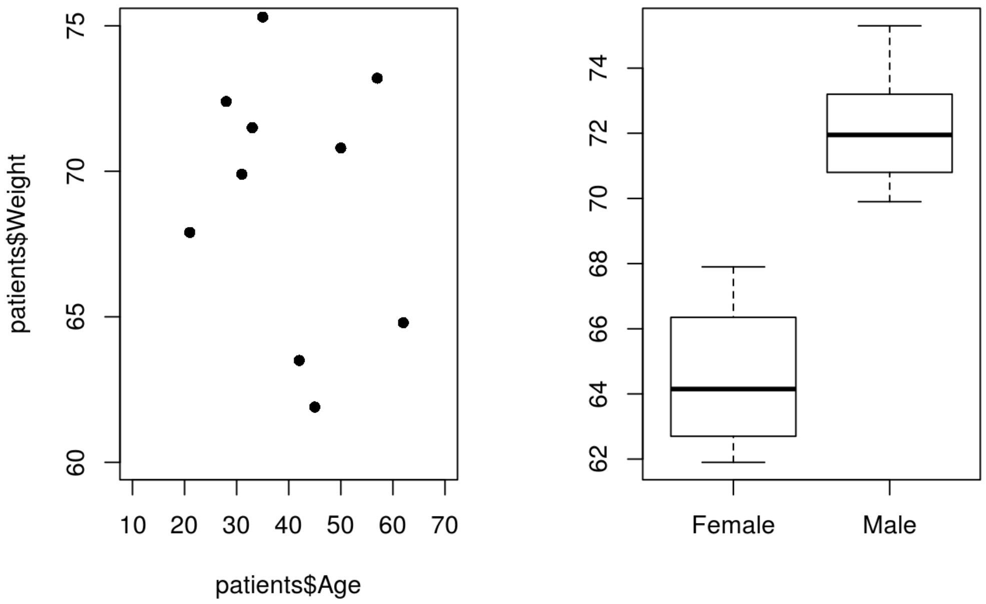


Plot layouts

- The `par` function can be used specify the appearance of a plot
- The settings persist until the plot is closed with `dev.off`
- `?par` and scroll to ***graphical parameters***
- One example is `mfrow`
    - “multiple figures per row”
    - needs to be a vector of rows and columns
        - e.g. a plot with one row and two columns `par(mfrow=c(1,2))`
        - don’t need the same kind of plot in each cell

Plot layouts

```
par(mfrow=c(1,2))
plot(patients$Age, patients$Weight,pch=16,xlim=c(10,70),yli
m=c(60,75))
boxplot(patients$Weight~patients$Sex)
```



- see also `mar` for setting the margins
  - `par(mar=c(...))`

Exporting graphs from RStudio

- Easiest option to to use the Export button from the Plots panel
- Otherwise, use the `pdf` function
  - you will see that the plot does not appear in RStudio

```
pdf("ExampleGraph.pdf")
plot(rnorm(1:10))
```

- You need to use the `dev.off` to stop printing graphs to the pdf and ‘close’ the file
  - allows you to create a pdf document with multiple pages

```
dev.off()
```

- pdf is a good choice for publication as they can be imported into photoshop, inkscape etc

Exporting graphs from RStudio

- To save any graph you have created to a pdf, repeat the code you used to create the plot with `pdf(..)` before and `dev.off()` afterwards
  - you can have as many lines of code in-between as you like

```
pdf("mygraph.pdf")
plot(patients$Age, patients$Weight,pch=16)
abline(v=40,col="red")
abline(h=c(65,70,75),col="blue")
dev.off()
```

```
## png
##   2
```

Exporting graphs from RStudio

- We can specify the dimensions of the plot, and other properties of the file (`?pdf`)

```
pdf("ExampleGraph.pdf",width=10,height=10)
plot(rnorm(1:10))
dev.off()
```

```
## png
##   2
```

- Other formats can be created
  - e.g. ***png***, or others `?png`
  - more appropriate for email, presentations, web page

```
png("ExampleGraph.png")
plot(rnorm(1:10))
dev.off()
```

```
## png
##   2
```

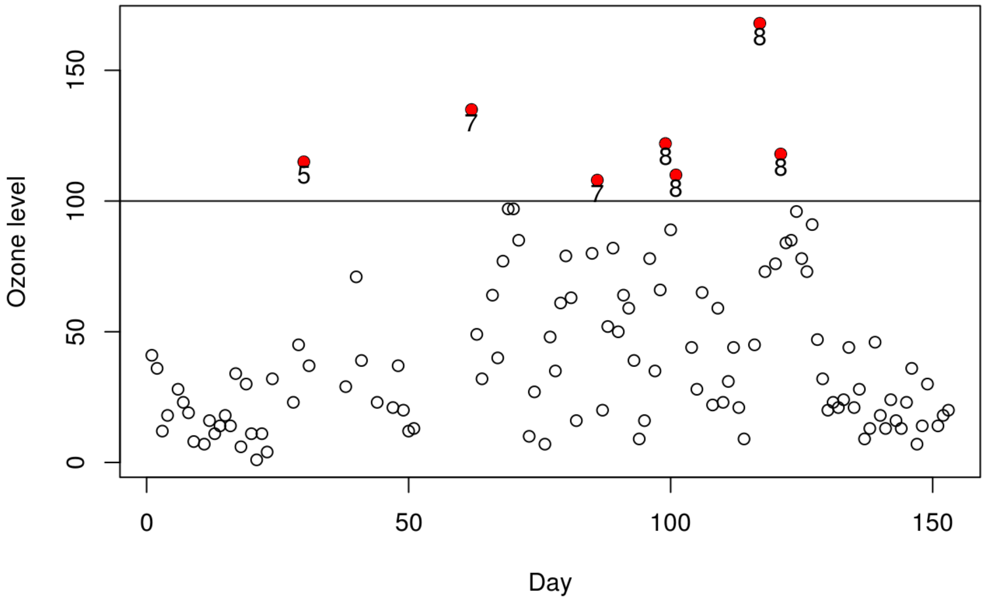
Exercise: exercise5.Rmd

- Return to the weather data from yesterday

```
weather <- read.csv("ozone.csv")
```

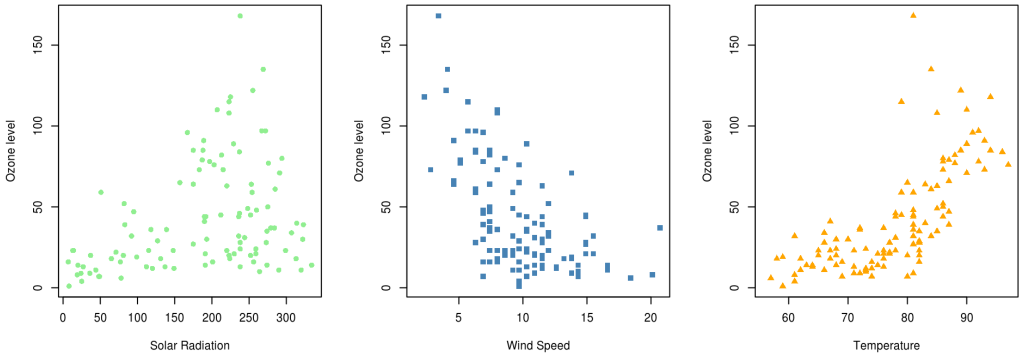
- Make a scatter plot of all observations of Ozone level
  - i.e. with the y axis being the Ozone variable, and x-axis being the row index
- Highlight any days in the study which had Ozone level > 100
- Indicate which month these days with high ozone-level belong to

Target Graph



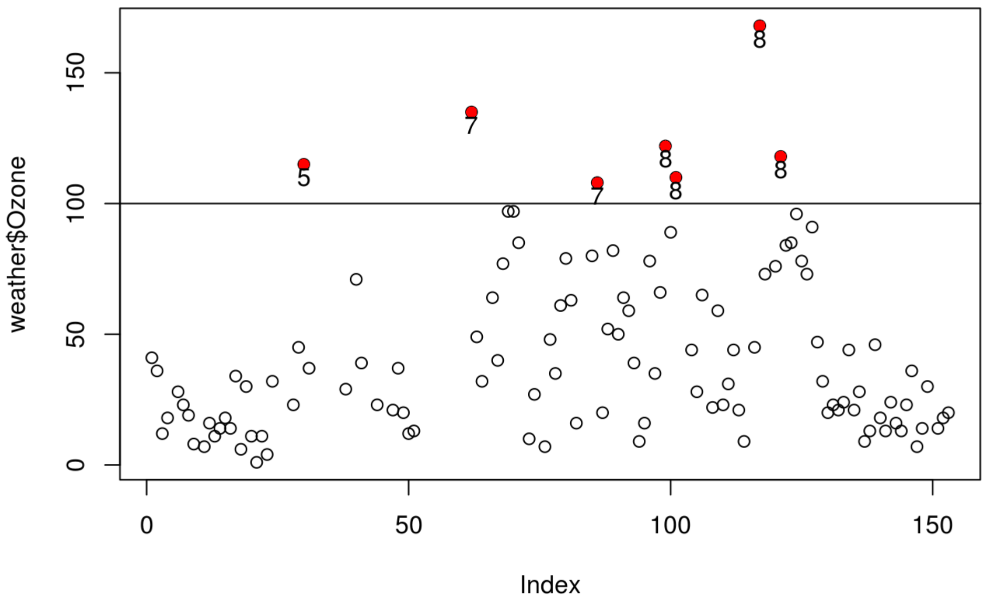
Exercise: exercise5.Rmd

- Using the `par` function, create a layout with three columns
- Plot Ozone versus Solar Radiation, Wind Speed and Temperature on separate graphs
    - use different colours and plotting characters on each plot
- Save the plot to a pdf
- HINT: Create the graph first in RStudio, then when you're happy with it, use the `pdf` function to save to a file



Solution

```
plot(weather$Ozone)
abline(h=100)
highO <- which(weather$Ozone > 100)
points(highO, weather$Ozone[highO],col="red",pch=16)
text(highO,weather$Ozone[highO]-5, labels=weather$Month[hig
hO])
```



Solution

```
pdf("ozoneCorrelations.pdf")
par(mfrow=c(1,3))
plot(weather$Solar.R,weather$Ozone,pch=16,col="lightgreen",
ylab="Ozone level",xlab="Solar Radiation")
plot(weather$Wind,weather$Ozone, pch=15,col="steelblue",yla
b="Ozone level", xlab="Wind Speed")
plot(weather$Temp,weather$Ozone,pch=17,col="orange", ylab="
Ozone level",xlab="Temperature")
dev.off()
```

If the graph looks a bit stretched…

```
pdf("ozoneCorrelations.pdf",width=10,height = 6)
par(mfrow=c(1,3))
plot(weather$Solar.R,weather$Ozone,pch=16,col="lightgreen",
ylab="Ozone level",xlab="Solar Radiation")
plot(weather$Wind,weather$Ozone, pch=15,col="steelblue",yla
b="Ozone level", xlab="Wind Speed")
plot(weather$Temp,weather$Ozone,pch=17,col="orange", ylab="
Ozone level",xlab="Temperature")
dev.off()
```

2. Statistics

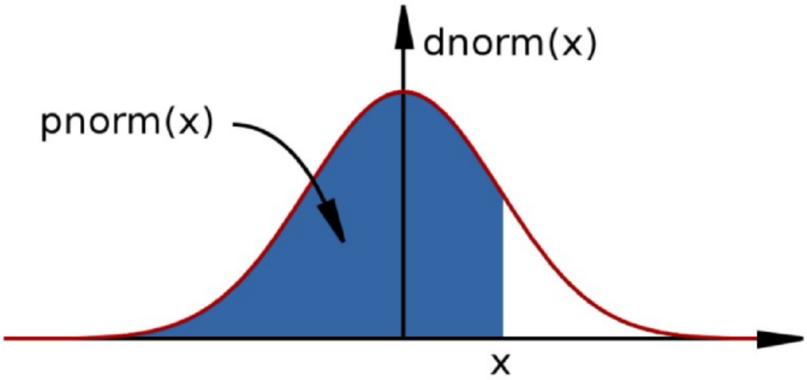
Built-in support for statistics

- R is a statistical programming language
    - Classical statistical tests are built-in

- Statistical modeling functions are built-in
- Regression analysis is fully supported
- Additional mathematical packages are available (`MASS` , Waves, sparse matrices, etc)

Distribution functions

- Most commonly used distributions are built-in, functions have stereotypical names, e.g. for normal distribution
  - `pnorm` - cumulative distribution for x
  - `qnorm` - inverse of pnorm (from probability gives x)
  - `dnorm` - distribution density
  - `rnorm` - random number from normal distribution



- available for variety of distributions: `punif` (uniform), `pbinom` (binomial), `pnbinom` (negative binomial), `ppois` (poisson), `pgeom` (geometric), `phyper` (hyper-geometric), `pt` (T distribution), pf (F distribution)

Distribution functions

- 10 random values from the Normal distribution with mean 10 and standard deviation 5

```
rnorm(10, mean=10, sd=5)
```

- The probability of drawing 10 from this distribution

```
dnorm(10, mean=10, sd=5)
```

```
## [1] 0.07978846
```

```
dnorm(100, mean=10, sd=5)
```

```
## [1] 3.517499e-72
```

Distribution functions (continued)

- The probability of drawing a value smaller than 10

```
pnorm(10, mean=10, sd=5)
```

```
## [1] 0.5
```

- The inverse of `pnorm`

```
qnorm(0.5, mean=10, sd=5)
```

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

- How many standard deviations for statistical significance?

```
qnorm(0.95, mean=0, sd=1)
```

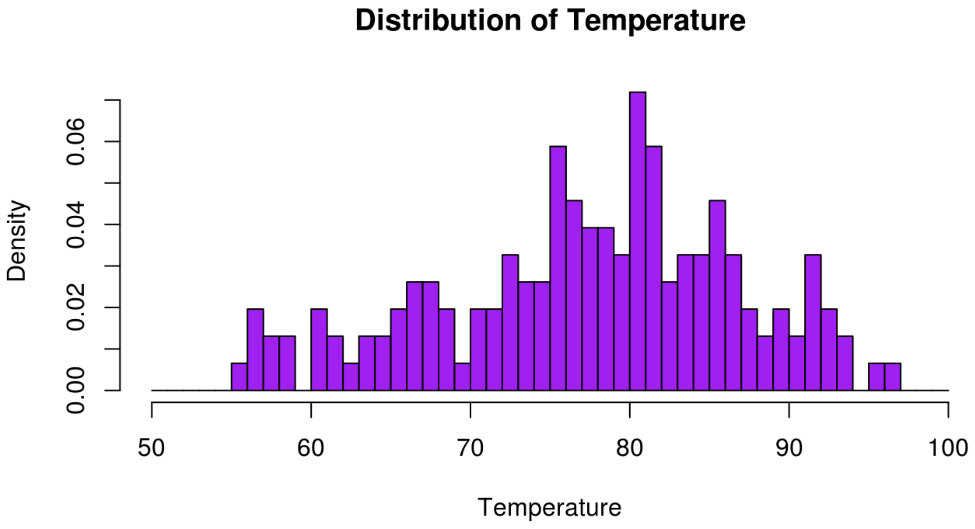
```
## [1] 1.644854
```

Example

Recall our histogram of temperature from yesterday

- the data look to be roughly normally-distributed
- an assumption we rely on for various statistical tests

```
hist(weather$Temp,col="purple",xlab="Temperature",
     main="Distribution of Temperature",breaks = 50:100,freq=FALSE)
```



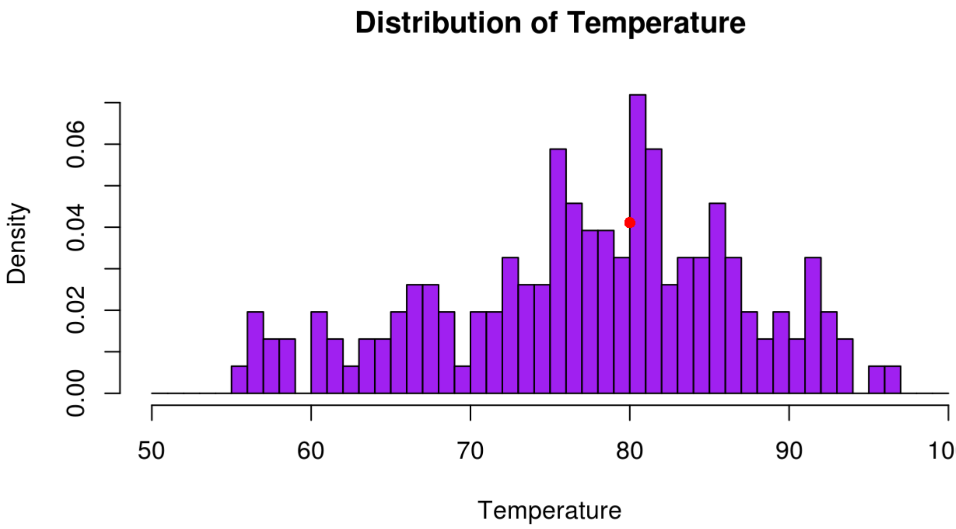
Create a normal distribution curve

If our data are normally-distributed, we can calculate the probability of drawing particular values.

- e.g. a temperature of 80
- we can overlay this on the histogram using `points` as we just saw

```
tempMean <- mean(weather$Temp)
tempSD <- sd(weather$Temp)

dnorm(80, mean=tempMean,sd=tempSD)
hist(weather$Temp,col="purple",xlab="Temperature",
     main="Distribution of Temperature",breaks = 50:100,fre
q=FALSE)
points(80, dnorm(80, mean=tempMean,sd=tempSD),col="red",pch
=16)
```

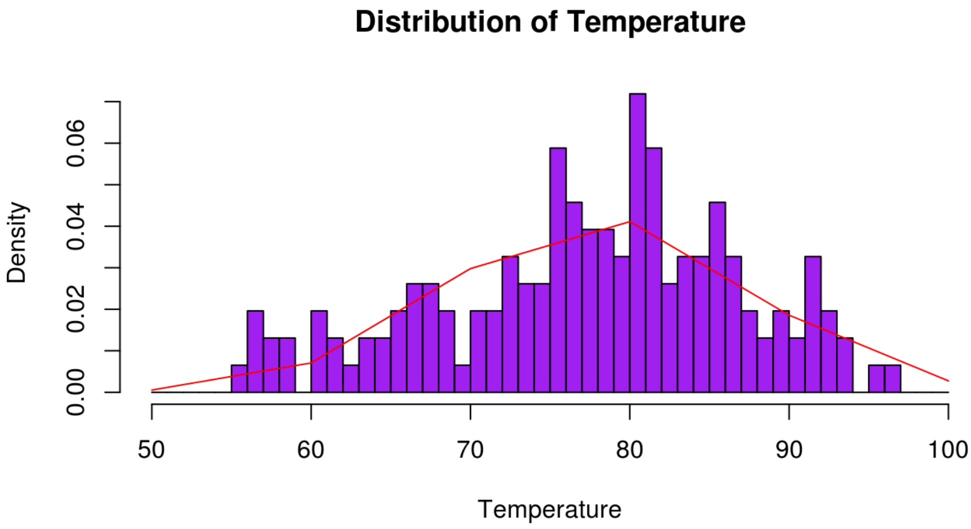
```
## [1] 0.04110626
```



Create a normal distribution curve

- We can repeat the calculation for a vector of values
  - remember that functions in R are often ***vectorized***
  - use `lines` in this case rather than `points`

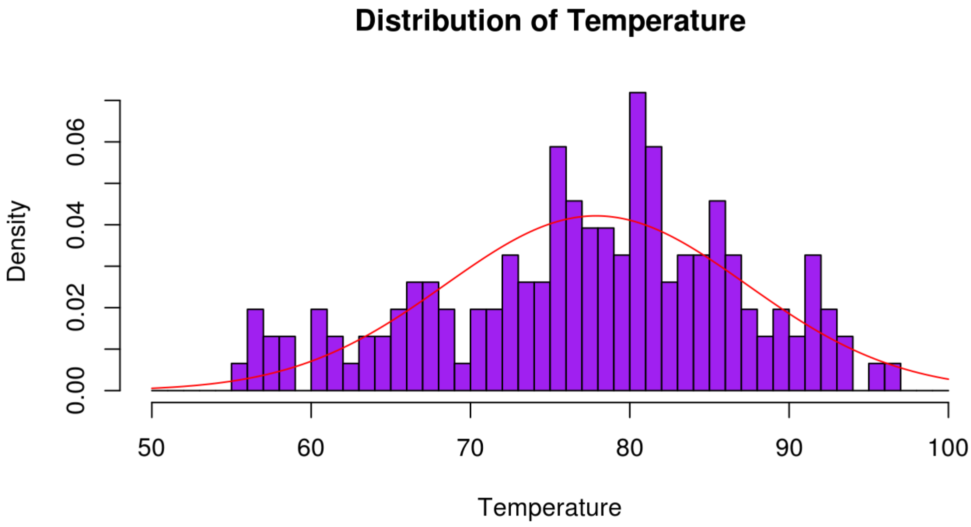
```
xs <- c(50,60,70,80,90,100)
ys <- dnorm(xs, mean=tempMean,sd=tempSD)
lines(xs,ys,col="red",pch=16)
```



Create a normal distribution curve

- For a smoother curve, use a longer vector
  - we can generate x values using the `seq` function

```
xs <- seq(50,100,length.out = 10000)
ys <- dnorm(xs, mean=tempMean,sd=tempSD)
lines(xs,ys,col="red",pch=16)
```



Simple testing

- If we want to compute the probability of observing a particular temperature, from the same distribution we can use the standard formula to calculate a t statistic:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{(n)}}$$

```
t <- (tempMean - 50) / (tempSD / sqrt(length(weather$Temp))
)
t
```

```
## [1] 36.43696
```

- or use the `t.test` function to compute the statistic and corresponding p-value

```
t.test(weather$Temp,mu=50)
```

```
##
##  One Sample t-test
##
## data:  weather$Temp
## t = 36.437, df = 152, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 50
## 95 percent confidence interval:
##  76.37051 79.39420
## sample estimates:
## mean of x
##  77.88235
```

Two sample tests: Basic data analysis

- Comparing 2 variances
    - Fisher’s F test

```
var.test()
```

- Comparing 2 sample means with normal errors
    - Student’s t test

```
t.test()
```

- Comparing 2 means with non-normal errors
    - Wilcoxon’s rank test

```
wilcox.test()
```

Two sample tests: Basic data analysis

- Comparing 2 proportions
    - Binomial test

```
prop.test()
```

- Correlating 2 variables

  - Pearson’s / Spearman’s rank correlation

```
cor.test()
```

- Testing for independence of 2 variables in a contingency table
  - Chi-squared / Fisher’s exact test

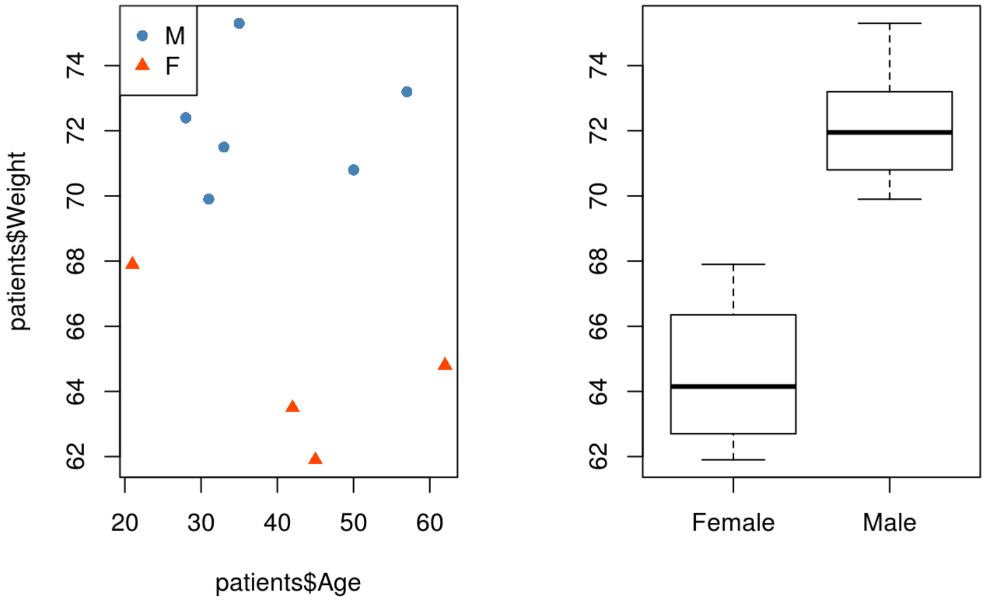
```
chisq.test();fisher.test()
```

Statistical tests in R

- Bottom-line: Pretty much any statistical test you care to name will probably be in R
  - this is not supposed to be a statistics course (sorry!)
  - none of them are particular harder than others to use
  - the difficulty is deciding which test to use
    - whether the assumptions of the test are met etc
  - consult your local statistician if not sure
  - some good references
    - Simple R eBook (https://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf)
    - Elements of Statistical Learning eBook (http://statweb.stanford.edu/~tibs/ElemStatLearn/download.html)

Example analysis

- We have already seen that men in our Patients dataset tend to be heavier than women
  - we can test this formally in R



Test variance assumption

```
var.test(patients$Weight~patients$Sex)
```

```
##
##  F test to compare two variances
##
## data:  patients$Weight by patients$Sex
## F = 1.759, num df = 3, denom df = 5, p-value = 0.5417
## alternative hypothesis: true ratio of variances is not e
qual to 1
## 95 percent confidence interval:
##   0.2265757 26.1830147
## sample estimates:
## ratio of variances
##           1.759041
```

Perform the t-test

```
t.test(patients$Weight~patients$Sex,var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
## data:  patients$Weight by patients$Sex
## t = -5.4584, df = 8, p-value = 0.0006027
## alternative hypothesis: true difference in means is not
equal to 0
## 95 percent confidence interval:
##  -10.893759  -4.422908
## sample estimates:
## mean in group Female   mean in group Male
##             64.52500             72.18333
```

- This function can be tuned in various ways
  - assumed equal variances, or not (and use Welch's correction)
  - deal with parired samples
  - two-sided, or one-sided p-value
  - as usual: `?t.test`

Linear regression: Basic data analysis

- Linear modeling is supported by the function `lm()`
  - `example(lm)` the output assumes you know a fair bit about the subject
- `lm` is really useful for plotting lines of best fit to XY data in order to determine intercept, gradient & Pearson's correlation coefficient
  - This is very easy in R

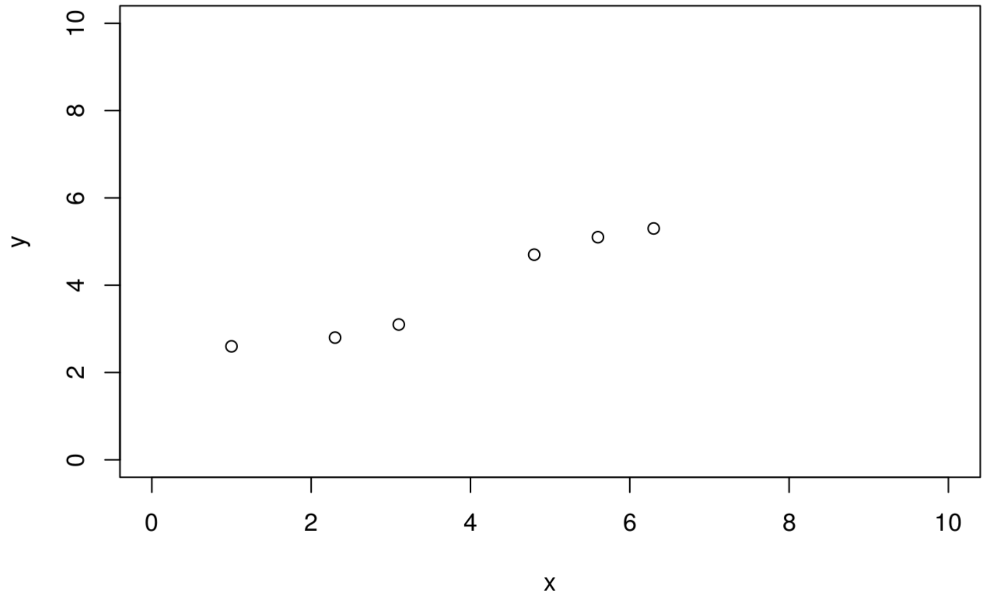
- Three steps to plotting with a best fit line

1. Plot XY scatter-plot data
2. Fit a linear model
3. Add bestfit line data to plot with `abline()` function

Typical linear regression analysis: Basic data analysis

- The ~ (***tilde***) is used to define a ***formula***; i.e. “y is given by x”

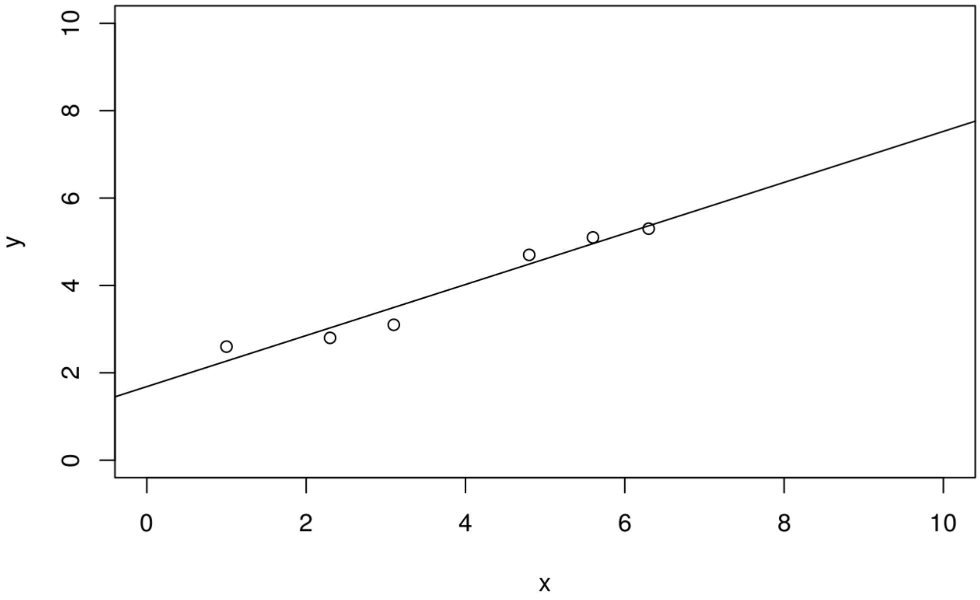
```
x <- c(1, 2.3, 3.1, 4.8, 5.6, 6.3)
y <- c(2.6, 2.8, 3.1, 4.7, 5.1, 5.3)
plot(x,y, xlim=c(0,10), ylim=c(0,10))
```



Typical linear regression analysis: Basic data analysis

The ~ is used to define a formula; i.e. “y is given by x” - Take care about the order of x and y in the plot and lm expressions

```
plot(x,y, xlim=c(0,10), ylim=c(0,10))
myModel <- lm(y~x)
abline(myModel)
```



In-depth summary

```
summary(myModel)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##        1        2        3        4        5        6
##  0.33159 -0.22785 -0.39520  0.21169  0.14434 -0.06458
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.68422    0.29056   5.796   0.0044 **
## x            0.58418    0.06786   8.608   0.0010 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
 ' ' 1
##
## Residual standard error: 0.3114 on 4 degrees of freedom
## Multiple R-squared:  0.9488, Adjusted R-squared:  0.936
## F-statistic:  74.1 on 1 and 4 DF,  p-value: 0.001001
```

Typical linear regression analysis: Basic data analysis

- Get the coefficients of the fit from:

```
coef(myModel)
```

```
## (Intercept)           x 
##   1.6842239   0.5841843
```

```
resid(myModel)
```

```
##           1           2           3           4 
  5           6 
##  0.33159186 -0.22784770 -0.39519512  0.21169160  0.14434
418 -0.06458482
```

```
fitted(myModel)
```

```
##        1        2        3        4        5        6 
## 2.268408 3.027848 3.495195 4.488308 4.955656 5.364585
```

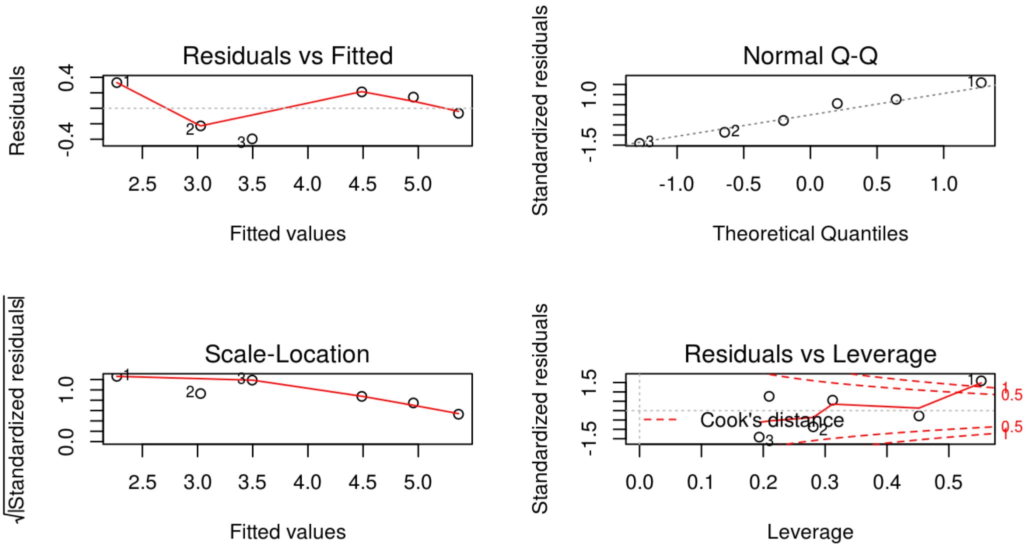
```
names(myModel)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "ra
nk"         
##  [5] "fitted.values" "assign"        "qr"            "df
.residual" 
##  [9] "xlevels"       "call"          "terms"         "mo
del"
```

Diagnostic plots of the fit

- Get QC of fit from

```
par(mfrow=c(2,2))
plot(myModel)
```



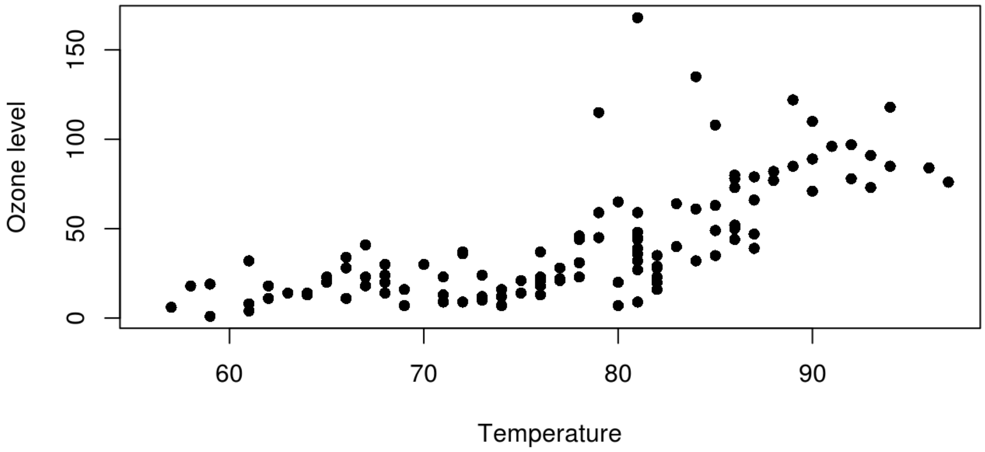
Modelling formulae

- R has a very powerful formula syntax for describing statistical models
- Suppose we had two explanatory variables `x` and `z` and one response variable `y`
- We can describe a relationship between, say, `y` and `x` using a tilde `~`, placing the response variable on the left of the tilde and the explanatory variables on the right:
    - `y~x`
- It is very easy to extend this syntax to do multiple regressions, ANOVAs, to include interactions, and to do many other common modelling tasks. For example

```
y~x       #If x is continuous this is linear regression
y~x       #If x is categorical, this is ANOVA
y~x+z     #If x and z are continuous, this is multiple regression
y~x+z     #If x and z are categorical, this is two-way ANOVA
y~x+z+x:z # : is the symbol for the interaction term
y~x*z     # * is a shorthand for x+z+x:z
```

Exercise: exercise6.Rmd

- There are suggestions that Ozone level could be influenced by Temperature



- Perform a linear regression analysis to assess this
    - fit the linear model and print a summary of the output
    - plot the two variables and overlay a best-fit line

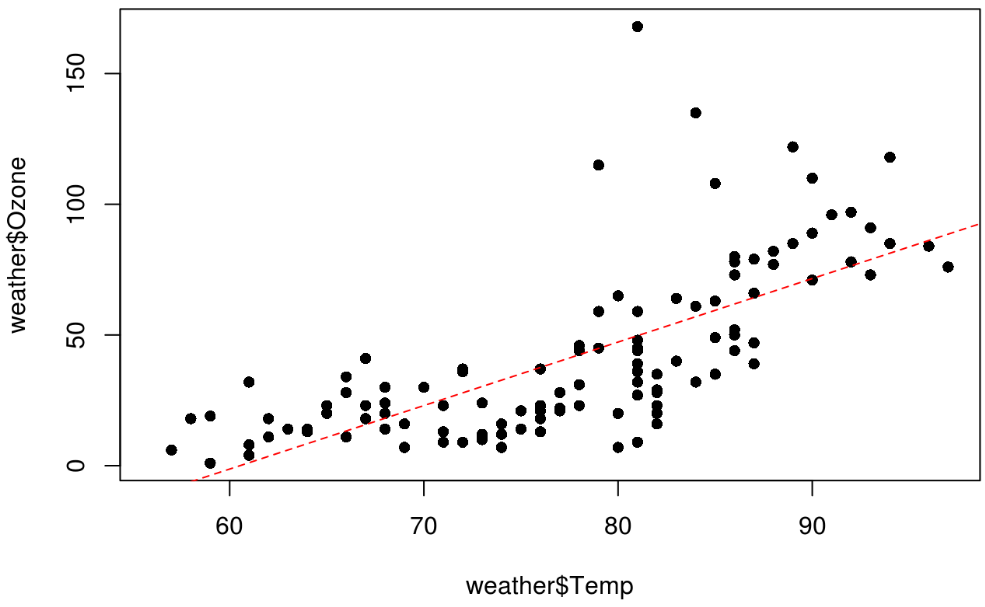
Solution: solution-exercise6.Rmd

```
mod1 <- lm(weather$Ozone~weather$Temp)
summary(mod1)
```

```
##
## Call:
## lm(formula = weather$Ozone ~ weather$Temp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -40.729 -17.409  -0.587  11.306 118.271
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -146.9955    18.2872  -8.038 9.37e-13 ***
## weather$Temp    2.4287     0.2331  10.418  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1
 ' ' 1
##
## Residual standard error: 23.71 on 114 degrees of freedom
##   (37 observations deleted due to missingness)
## Multiple R-squared:  0.4877, Adjusted R-squared:  0.4832

## F-statistic: 108.5 on 1 and 114 DF,  p-value: < 2.2e-16
```

Solution

```
plot(weather$Temp, weather$Ozone,pch=16)
abline(mod1,col="red",lty=2)
```



3. Data Manipulation Techniques

Motivation

- So far we have been lucky that all our data have been in the same file
    - this is not usually the case
    - dataset may be spread over several files
        - This takes longer, and is harder, than many people realise
    - We need to combine before doing an analysis

Combining data from multiple sources: Gene Clustering Example

- R has powerful functions to combine heterogeneous data sources into a single data set
- Gene clustering example data
    - gene expression values in ***gene.expression.txt***
    - gene information in ***gene.description.txt***
    - patient information in ***cancer.patients.txt***
- A breast cancer dataset with numerous patient characteristics
    - we will concentrate on ***ER status*** (positive / negative)
    - what genes show a statistically-significant different change between ER groups?

Peek at the data

```
evals <- read.delim("gene.expression.txt",stringsAsFactors
= FALSE)
evals[1:5,1:5]
dim(evals)
```

```
##                  NKI_4 NKI_6  NKI_7  NKI_8  NKI_9
## Contig56678_RC -0.261 0.346  0.047 -1.140 -0.110
## AF026004       -0.064 0.040 -0.165 -0.031  0.330
## AB033049       -0.307 0.046 -0.139  0.036 -0.154
## AB033050        0.582 0.216  0.091 -0.186 -0.156
## AB033086       -2.000 0.102 -0.016 -0.358  0.153
```

```
## [1] 498 337
```

- 498 rows and 337 columns
- One row for each gene
    - rows are named according to particular technology used to make measurement
    - the names of each row can be returned by `rownames(evals)` ; giving a vector
- One column for each patient
    - the names of each column can be returned by `colnames(evals)` ; giving a vector

Peek at the data

```
genes <- read.delim("gene.description.txt",stringsAsFactors
 = FALSE)
head(genes)
```

```
##                         probe HUGO.gene.symbol Chromosom
e     Start
## Contig56678_RC Contig56678_RC            THSD4      chr1
5  71433788
## AF026004             AF026004            CLCN2       chr
3 184063973
## AB033049             AB033049          ANKRD50       chr
4 125585207
## AB033050             AB033050            ZMIZ1      chr1
0  80828792
## AB033086             AB033086           NLGN4X       chr
X   5808083
## NM_003008           NM_003008            SEMG2      chr2
0  43850010
```

```
dim(genes)
```

```
## [1] 498   4
```

- 498 rows and 4 columns
- One for for each gene
- Includes mapping between manufacturer ID and Gene name

Peek at the data

```
subjects <- read.delim("cancer.patients.txt")
head(subjects)
```

```
##        samplename age er grade
## NKI_4       NKI_4  41  1     3
## NKI_6       NKI_6  49  1     2
## NKI_7       NKI_7  46  0     1
## NKI_8       NKI_8  48  0     3
## NKI_9       NKI_9  48  1     3
## NKI_11     NKI_11  37  1     3
```

```
dim(subjects)
```

```
## [1] 337   4
```

- One for each patient in the study
- Each column is a different characteristic of that patient
    - e.g. whether a patient is ER positive or negative

```
table(subjects$er)
```

```
## 
##   0   1 
##  88 249
```

Ordering and sorting

To get a feel for these data, we will look at how we can subset and order

- R allows us to do the kinds of filtering, sorting and ordering operations you might be familiar with in Excel
- For example, if we want to get information about patients that are ER negative
    - these are indicated by an entry of ***0*** in the `er` column

```
subjects$er==0
```

```
## [1] "FALSE" "FALSE" "..."   "FALSE" "FALSE"
```

Ordering and sorting

We can do the comparison within the square brackets

- Remembering to include a `,` to index the columns as well
- Best practice to create a new variable and leave the original data frame untouched

```
erNegPatients <- subjects[subjects$er==0,]
head(erNegPatients)
```

```
##        samplename age er grade
## NKI_7       NKI_7  46  0     1
## NKI_8       NKI_8  48  0     3
## NKI_12     NKI_12  46  0     3
## NKI_24     NKI_24  49  0     3
## NKI_28     NKI_28  40  0     3
## NKI_44     NKI_44  53  0     3
```

Ordering and sorting

Sorting is supported by the `sort` function

- given a vector, it will return a sorted version of the same length

```
sort(erNegPatients$grade)
```

```
##  [1] "1"   "1"   "1"   "1"   "1"   "2"   "2"   "2"   "2"
   "2"   "..."
## [12] "3"   "3"   "3"   "3"   "3"   "3"   "3"   "3"   "3"
```

- but this is not useful in all cases
    - we have lost the extra information that we have about the patients

Ordering and sorting

- Instead, we can use `order`
- Given a vector, `order` will give a set of numeric values which will give an ordered version of the vector
    - default is smallest —> largest

```
myvec <- c(9,10,4,3,8,5,6,2,1,7)
myvec
```

```
##  [1]  9 10  4  3  8  5  6  2  1  7
```

```
order(myvec)
```

```
##  [1]  9  8  4  3  6  7 10  5  1  2
```

- i.e. number in position 9 is the smallest, number in position 8 is the second smallest

```
myvec[9]
```

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

```
myvec[8]
```

```
## [1] 2
```

Ordering and sorting

- We can use the result of `order` to perform a subset of our original vector
- The result is an ordered vector

```
myvec.ord <- myvec[order(myvec)]
myvec.ord
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

- Implication: We can use `order` on a particular column of a data frame, and use the result to sort all the rows

Ordering and sorting

- Here we order the `age` column and use the result to re-order the rows in the data frame

```
erNegPatientsByAge <- erNegPatients[order(erNegPatients$age
),]
head(erNegPatientsByAge)
```

```
##         samplename age er grade
## NKI_330    NKI_330  26  0     3
## NKI_57      NKI_57  28  0     3
## NKI_230    NKI_230  28  0     3
## NKI_90      NKI_90  29  0     3
## NKI_48      NKI_48  30  0     3
## NKI_86      NKI_86  30  0     3
```

Ordering and sorting

- can change the behaviour of `order` to be Largest —-> Smallest

```
erNegPatientsByAge <- erNegPatients[order(erNegPatients$age
,decreasing = TRUE),]
head(erNegPatientsByAge)
```

```
##         samplename age er grade
## NKI_96      NKI_96  62  0     3
## NKI_93      NKI_93  61  0     3
## NKI_119    NKI_119  54  0     3
## NKI_44      NKI_44  53  0     3
## NKI_75      NKI_75  52  0     3
## NKI_76      NKI_76  52  0     2
```

- we can write the result to a fie if we wish

```
write.table(erNegPatientsByAge,file="erNegativeSubjectsByAg
e.txt",sep="\t")
```

Exercise: exercise7.Rmd

- Imagine we want to know information about chromosome 8 genes that have been measured.
- create a new data frame containing information on genes on Chromosome 8
- order the rows in this data frame according to start position, and write the results to a file

Solution: solution-exercise7.Rmd

```
chr8Genes <- genes[genes$Chromosome=="chr8",]
head(chr8Genes)
```

```
##                         probe HUGO.gene.symbol Chromosom
e     Start
## Contig29827_RC Contig29827_RC            FUT10       chr
8  33228344
## NM_003046           NM_003046           SLC7A2       chr
8  17396286
## Contig55940_RC Contig55940_RC            CYHR1       chr
8 145675315
## NM_004133           NM_004133            HNF4G       chr
8  76452203
## NM_004374           NM_004374            COX6C       chr
8 100890223
## AF052142             AF052142            NCALD       chr
8 102698770
```

```
chr8GenesOrd <-chr8Genes[order(chr8Genes$Start),]
head(chr8GenesOrd)
```

```
##                          probe HUGO.gene.symbol Chromosom
e    Start
## NM_004745           NM_004745           DLGAP2       chr
8  1449569
## NM_018941           NM_018941             CLN8       chr
8  1711870
## AL117604             AL117604             DLC1       chr
8 12940872
## NM_003046           NM_003046           SLC7A2       chr
8 17396286
## Contig58301_RC Contig58301_RC           SLC7A2       chr
8 17396286
## NM_000662           NM_000662             NAT1       chr
8 18067618
```

```
write.table(chr8GenesOrd,"chromosome8.gene.info.txt",sep="\
t")
```

Retrieving data for a particular gene

- Gene `ESR1` is known to be hugely-different between ER positive and negative patient
    - let’s check that this is evident in our dataset
    - if not, something has gone wrong!
- First step is to locate this gene in our dataset

Character matching in R

- we have already seen various ways of comparing numeric values
    - ==, >, <
    - each of which returns a vector of logical values
    - == will also work with text

```
LETTERS
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"
 "N" "O" "P" "Q"
## [18] "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
```

```
"A" == LETTERS
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
SE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FAL
SE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE
```

Character matching in R

- `match` and `grep` are often used to find particular matches
    - CAUTION: by default, match will only return the ***first*** match!

```
match("D", LETTERS)
```

```
## [1] 4
```

```
grep("F", rep(LETTERS,2))
```

```
## [1]  6 32
```

```
match("F", rep(LETTERS,2))
```

```
## [1] 6
```

Retrieving data for a particular gene

- find the name of the ID that corresponds to gene ***ESR1***
    - mapping between IDs and genes is in the ***genes*** data frame
        - ID in first column, gene name in the second
- save this ID as a variable

```
ind <- match("ESR1", genes$HUGO.gene.symbol)
genes[ind,]
```

```
##               probe HUGO.gene.symbol Chromosome     Star
t
## NM_000125 NM_000125             ESR1       chr6 15212881
4
```

```
probe <- genes[ind,1]
probe
```

```
## [1] "NM_000125"
```

Retrieving data for a particular gene

Now, find which row in our expression matrix is indexed by this ID

- recall that the rownames of the expression matrix are the probe IDs
- save the expression values as a variable

```
match(probe,rownames(evals))
```

```
## [1] 384
```

```
evals[match(probe,rownames(evals)),1:10]
```

```
##            NKI_4 NKI_6  NKI_7 NKI_8 NKI_9 NKI_11 NKI_12
NKI_13 NKI_14
## NM_000125 -0.007 0.074 -0.767 -0.82 -0.18 -0.296     NA
-0.163  0.059
##           NKI_17
## NM_000125 -0.035
```

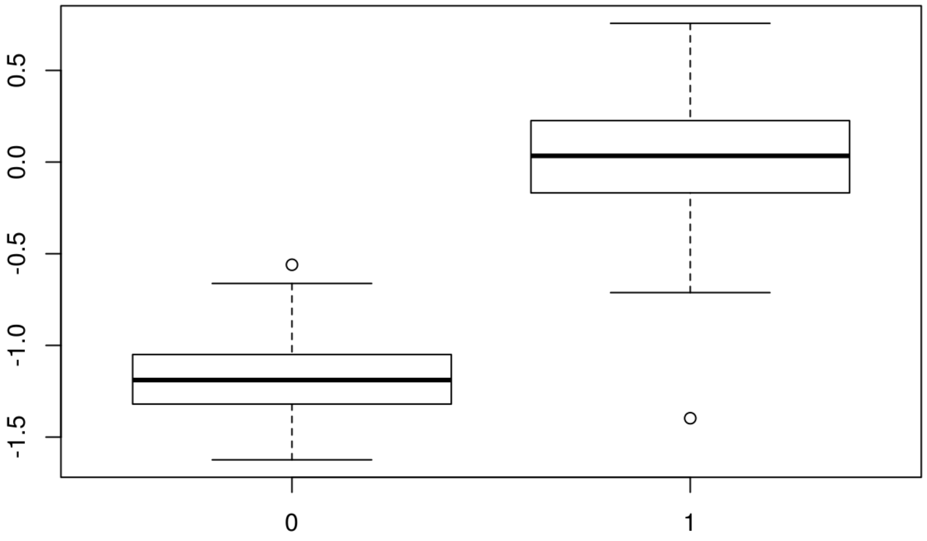
```
genevals <- evals[match(probe,rownames(evals)),]
```

Relating to patient characteristics

We have numeric expression values and want to visualise them against our categorical data

- use a boxplot, for example

```
boxplot(as.numeric(genevals)~factor(subjects$er))
```



Relating to patient characteristics

- the p-value is also encouraging

```
t.test(as.numeric(genevals)~factor(subjects$er))
```

```
##
##  Welch Two Sample t-test
##
## data:  as.numeric(genevals) by factor(subjects$er)
## t = -38.746, df = 205.88, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not
equal to 0
## 95 percent confidence interval:
##  -1.246953 -1.126198
## sample estimates:
## mean in group 0 mean in group 1
##     -1.17388506      0.01269076
```

Complete script

esr1Example.Rmd

```
genes <- read.delim("gene.description.txt")
subjects <- read.delim("cancer.patients.txt")
evals <- read.delim("gene.expression.txt",stringsAsFactors
= FALSE)

ind <- match("ESR1", genes[,2])
probe <- genes[ind,1]
genevals <- evals[match(probe,rownames(evals)),]
boxplot(as.numeric(genevals)~factor(subjects$er))
t.test(as.numeric(genevals)~factor(subjects$er))
```

Exercise: exercise8.Rmd

Repeat the same steps we performed for the gene ESR1, but for GATA3

- Try and make as few changes as possible from the ESR1 script
- Can you see why making a markdown document is useful for analysis?

4. Programming in R

Motivation

From the previous exercise, you should see how we can easily adapt our markdown scripts

- e.g. ESR1 versus GATA3
- But what if we want to analyse many genes?
- it would be tedious to create a new markdown document for every gene
- ……and prone to error too

Introducing loops

- Many programming languages have ways of doing the same thing many times, perhaps changing some variable each time. This is called *looping*
- Loops are not used in R so often, because we can usually achieve the same thing using vector calculations
- For example, to add two vectors together, we do not need to add each pair of elements one by one, we can just add the vectors

```
x<- 1:10
y <- 11:20
x+y
```

- But there are some situations where R functions can not take vectors as input. For example, `t.test` will only test one gene at a time
- What if we wanted to test multiple genes?

Introducing loops

- We could do this:

```
t.test(evals[1,]~factor(subjects$er))
t.test(evals[2,]~factor(subjects$er))
```

- But this will be boring to type, difficult to change, and prone to error
- As we are doing the same thing multiple times, but with a different index each time, we can use a **loop** instead

Loops: Commands and flow control

- R has two basic types of loop
  - a `for` loop: run some code on every value in a vector
  - a `while` loop: run some code while some condition is true
    - *hardly ever used!*

`for`

```
for(i in 1:10){
  print(i)
}
```

`while`

```
i <- 1
while ( i <= 10 ) {
   print(i)
   i <- i + 1
}
```

Loops: Commands and flow control

- Here's how we might use a `for` loop to test the first 10 genes

```
for (i in 1:10) {
   t.test(as.numeric(evals[i,])~factor(subjects$er))
}
```

- This is exactly the same as:

```
i <- 1
t.test(evals[i,]~factor(subjects$er))
i <- 2
t.test(evals[i,]~factor(subjects$er))
i <- 3 ......
```

Storing results

However, this for loop is doing the calculations but not storing the results

- the output of `t.test` is an object with data placed in different slots
  - the `names` of the object tells us what data we can retrieve, and what name to use
  - N.B it is a “list” object

```
t <- t.test(as.numeric(evals[1,])~factor(subjects$er))
names(t)
```

```
## [1] "statistic"   "parameter"   "p.value"     "conf.int"
    "estimate"
## [6] "null.value"  "alternative" "method"      "data.name
"
```

```
t$statistic
```

```
##         t
## -20.12546
```

Storing results

- When using a loop, we often create an empty “dummy” variable
- This is used store the results at each stage of the loop

```
stats <- NULL
for (i in 1:10) {
   tmp <- t.test(as.numeric(evals[i,])~factor(subjects$er))
   stats[i] <- tmp$statistic
}
stats
```

```
##  [1] -20.1254643  -1.7973581  -9.2625540  -3.3080720   0
.7512869
##  [6]  -0.6220547  -0.2596520  -4.1309155  -1.7027881 -16
.1224377
```

Practical application

Previously we have identified probes on chromosome 8

- Lets say that we want to do a t-test for each gene on chromosome 8

```
head(chr8GenesOrd)
```

```
##                         probe HUGO.gene.symbol Chromosom
e    Start
## NM_004745          NM_004745           DLGAP2       chr
8  1449569
## NM_018941          NM_018941             CLN8       chr
8  1711870
## AL117604            AL117604             DLC1       chr
8 12940872
## NM_003046          NM_003046           SLC7A2       chr
8 17396286
## Contig58301_RC Contig58301_RC           SLC7A2       chr
8 17396286
## NM_000662          NM_000662             NAT1       chr
8 18067618
```

- The first step is to extract the expression values for chromosome 8 genes from our expression matrix, which has expression values for all genes
- We can use the `match` function to tell us which rows in the matrix correspond to chromosome 8 genes

```
match(chr8GenesOrd$probe,rownames(evals))
```

```
##  [1] 215 494 161   8 481 461 140 478   7  87 256 139 449
 128 138 176 201
## [18]  77
```

```
chr8Expression <- evals[match(chr8GenesOrd$probe,rownames(e
vals)),]
dim(chr8Expression)
```

```
## [1]  18 337
```

Exercise: exercise9.Rmd

- Create a for loop to perform to test if the expression level of each gene on chromosome 8 is significantly different between ER positive and negative samples
- Store the ***p-value*** from each individual test

Solution

```
pvals <- NULL
for (i in 1:18) {
   tmp <- t.test(as.numeric(chr8Expression[i,])~factor(subj
ects$er))
   pvals[i] <- tmp$p.value
}
pvals
```

```
##  [1] 5.464153e-03 2.408701e-01 5.842811e-05 6.611391e-05
 2.590922e-57
##  [6] 2.564435e-69 9.382548e-01 7.555477e-01 7.955434e-01
 2.088048e-01
## [11] 2.695280e-01 5.440249e-01 3.764754e-02 2.297528e-37
 2.077849e-04
## [16] 2.188104e-03 1.340043e-12 2.169950e-08
```

Conditional branching: Commands and flow control

- Use an `if` statement for any kind of condition testing
- Different outcomes can be selected based on a condition within brackets

```
if (condition) {
... do this ...
} else {
... do something else ...
}
```

- `condition` is any logical value, and can contain multiple conditions.
    - e.g. `(a == 2 & b < 5)`, this is a compound conditional argument

Other conditional tests

- There are various tests that can check the type of data stored in a variable
    - these tend to be called `is.`
        - try *tab-complete* on `is.`

```
is.numeric(10)
```

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

```
is.numeric("TEN")
```

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

```
is.character(10)
```

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

- `is.na` is useful for seeing if an `NA` value is found
  - cannot use `== NA` !

```
match("foo", genes[,2])
```

```
## [1] NA
```

```
is.na(match("foo", genes[,2]))
```

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

Conditional branching: Commands and flow control

- Using the `for` loop we wrote before, we could add some code to plot the expression of each gene
  - a boxplot would be ideal
- However, we might only want plots for genes with a “significant” pvalue
- Here’s how we can use an `if` statement to test for this
  - for each iteration of the the loop
    - test if the p-value from the test is below 0.05 or not
    - if the p-value is less than 0.05 make a boxplot
    - if not, do nothing

```
pdf("Chromosome8Genes.pdf")
pvals <- NULL
for (i in 1:18) {
   tmp <- t.test(as.numeric(chr8Expression[i,])~factor(subjects$er))
   pvals[i] <- tmp$p.value
   if(tmp$p.value < 0.05){
     boxplot(as.numeric(chr8Expression[i,])~factor(subjects$er),main=chr8Genes$HUGO.gene.symbol[i])
   }
}
pvals
dev.off()
```

Code formatting avoids bugs!

Compare:

```
f<-26
while(f!=0){
print(letters[f])
f <- f-1}
```

to:

```
f <- 26
while( f != 0 ){
   print(letters[f])
   f <- f-1
}
```

- The code between brackets `{}` *always* is *indented*, this clearly separates what is executed once, and what is run multiple times
- Trailing bracket `}` always alone on the line at the same indentation level as the initial bracket `{`
- Use white spaces to divide the horizontal space between units of your code, e.g. around assignments, comparisons

5. Report Writing

Creating a markdown file from scratch

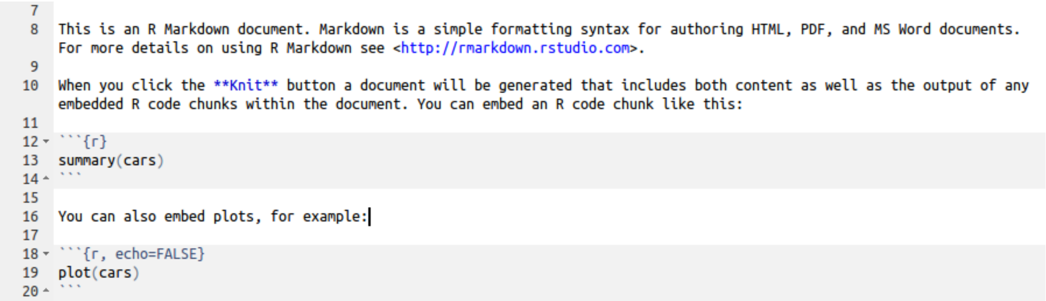
File - > New File - > R Markdown

- Choose ‘Document’ and the default output type (HTML)
- A new tab is created in RStudio
- The header allows you to specify a Page title, author and output type

```
---
title: "Untitled"
author: "Mark Dunning"
date: "18/08/2015"
output: html_document
---
```

Format of the file

- **Lines 8 - 10** Plain text description
- **Lines 12 - 14** An R code ‘chunk’
- **Lines 18 to 20** Another code chunk, this time producing a plot



~~~

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r}
summary(cars)
```

You can also embed plots, for example:

```{r, echo=FALSE}
plot(cars)
```
~~~

- Pressing the ***Knit HTML*** button will create the report
  - Note that you need to 'save' the markdown file before you will see the compiled report in your working directory

Text formatting

See **? - >** ***Markdown Quick Reference*** in RStudio

- Enclose text in * to format in *italics*
- Enclose text in ** to format in **bold**
- *** for ***bold italics***
- ` to format like `code`
- $ to include equations: $e = mc^2$
- > quoted text:

> To be or not to be

- see Markdown Quick Reference for more
  - adding images
  - adding web links
  - tables

Not quite enough for a reproducible document

- Minimally, you should record what version of R, and the packages you used.
- Use the `sessionInfo()` function
  - e.g. for the version of R I used to make the slides

```
sessionInfo()
```

```
## R version 3.2.2 (2015-08-14)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.2 LTS
## 
## locale:
##  [1] LC_CTYPE=en_GB.UTF-8       LC_NUMERIC=C              

##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_GB.UTF-8    

##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_GB.UTF-8   

##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C                 

##  [9] LC_ADDRESS=C               LC_TELEPHONE=C            

## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C       

## 
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  me
thods   base     
## 
## loaded via a namespace (and not attached):
##  [1] magrittr_1.5        formatR_1.2.1       parallel_3.
2.2    
##  [4] tools_3.2.2         htmltools_0.2.6     yaml_2.1.13

##  [7] Biobase_2.30.0      stringi_1.0-1       rmarkdown_0
.8.1   
## [10] knitr_1.11          BiocGenerics_0.16.1 stringr_1.0
.0     
## [13] digest_0.6.8        evaluate_0.8
```

Defining chunks

- It is not great practice to have one long, continuous R script
- Better to break-up into smaller pieces; '*chunks*'
- You can document each chunk separately
- Easier to catch errors
- The characteristics of each chunk can be modified
  - You might not want to print the R code for each chunk
  - or the output
  - etc

Chunk options

Code chunks are encapsulated between backticks. Options for the chunk can be put inside the curly brackets `{....}`

```
'''{r}
my code here.....
'''
```

- It’s a good idea to name each chunk
  - Easier to track-down errors
- We can display R code, but not run it
  - `eval=FALSE`
- We can run R code, but not display it
  - `echo=FALSE`
  - e.g. setting display options
- Suppress warning messages
  - `warning=FALSE`

Chunk options: eval

- Sometimes we want to format code for display, but not execute
  - we want to show the code for how we read our data, but want our report to compile quickly

```
'''{r,eval=FALSE}
data <- read.delim("path.to.my.file")
'''
```

Chunk options: echo

- Might want to load some data from disk
  - e.g. the R object from reading the data in the previous slide

    ```
    '''{r echo=FALSE}
    load("mydata.rda")
    '''
    ```

- Your P.I. wants to see your results, but doesn’t really want to know about the R code that you used

Chunk options: results

- Some code or functions might produce lots of output to the screen that we don’t need

```
for(i in 1:100){
  print(i)
}
```

Chunk options: message and warning

- Loading an R package will sometimes print messages and / or warnings to the screen
    - not always helpful in a report

```
'''{r}
library(DESeq)
'''
```

```
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
##
## The following objects are masked from 'package:parallel'
:
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEv
alQ,
##     clusterExport, clusterMap, parApply, parCapply, parL
apply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following objects are masked from 'package:stats':
##
##     IQR, mad, xtabs
##
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbi
nd,
##     colnames, do.call, duplicated, eval, evalq, Filter,
Find, get,
##     grep, grepl, intersect, is.unsorted, lapply, lengths
, Map,
##     mapply, match, mget, order, paste, pmax, pmax.int, p
min,
##     pmin.int, Position, rank, rbind, Reduce, rownames, s
apply,
##     setdiff, sort, table, tapply, union, unique, unlist,
 unsplit
##
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("p
kgname")'.
##
## Loading required package: locfit
## locfit 1.5-9.1    2013-03-22
## Loading required package: lattice
##     Welcome to 'DESeq'. For improved performance, usabil
ity and
##     functionality, please consider migrating to 'DESeq2'
.
```

Chunk options: message and warning

- Using `message=FALSE` and `warning=FALSE`

```
'''{r message=FALSE,warning=FALSE}
library(DESeq)
'''
```

- Could also need `suppressPackageStartupMessages`

Chunk options: cache

- `cache=TRUE` will stop certain chunks from being evaluate if their code does not change
- speeds-up the compilation of the document
    - we don’t want to reload our dataset if we’ve only made a tiny change downstream

```
'''{r echo=FALSE,cache=TRUE}
load("mydata.rda")
'''
```

Running R code from the main text

- We can add R code to our main text, which gets evaluated
    - make sure we always have the latest figures, p-values etc

```
.....the sample population consisted of  'r table(gender)[1
]' females and 'r table(gender)[2]' males.....
```

…..the sample population consisted of 47 females and 50 males…..

```
.....the p-value of the t-test is 'r pval', which indicates
 that.....
```

…..the p-value of the t-test is 0.05, which indicates that…..

- We call this “in-line” code

Running R code from the main text

- Like the rest of our report these R statements will get updated each time we compile the report

```
.....the sample population consisted of  'r table(gender)[1
]' females and 'r table(gender)[2]' males.....
```

…..the sample population consisted of 41 females and 54 males…..

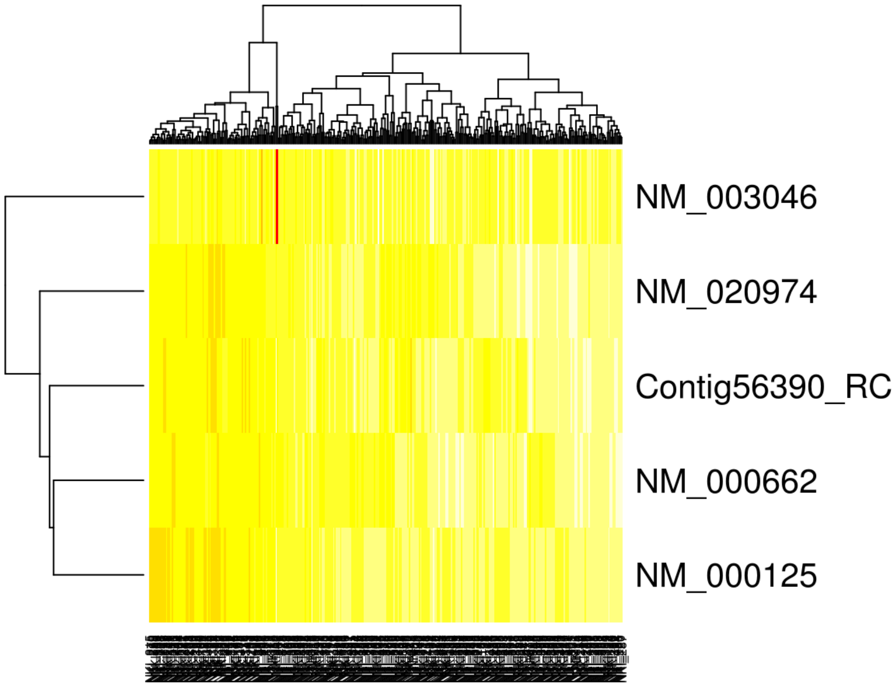
```
.....the p-value of the t-test is 'r pval', which indicates
 that.....
```

…..the p-value of the t-test is 0.1, which indicates that…..

Making a heatmap

- A heatmap is often used to visualise how the expression level of a set of genes vary between conditions
- Making the plot is actually quite straightforward
    - providing you have processed the data appropriately!
    - here, we use `na.omit` to ensure we have no `NA` values

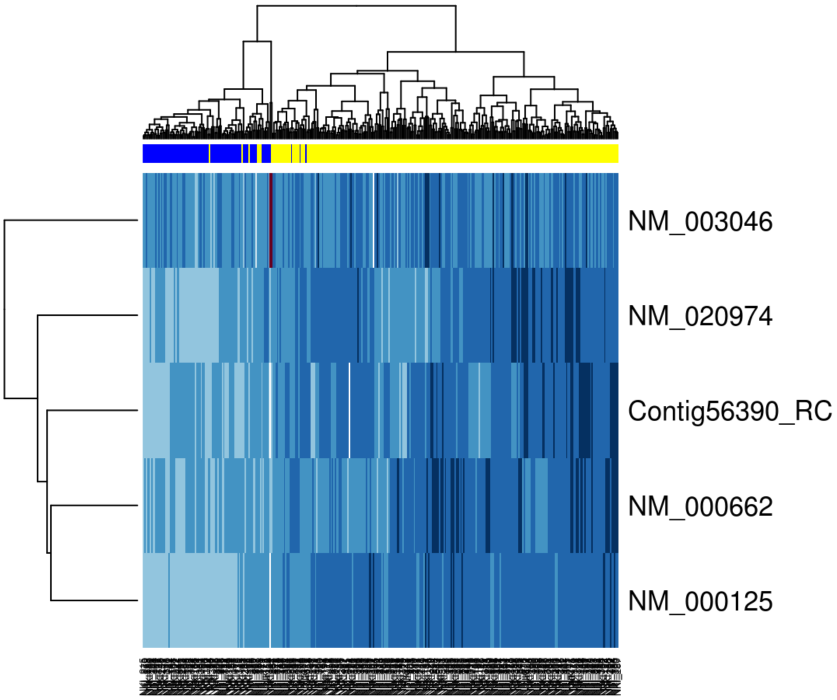
```
genelist <- c("ESR1", "NAT1", "SUSD3","SLC7A2" ,"SCUBE2")
probes <- na.omit(genes[match(genelist,genes[,2]),1])
exprows <- match(probes,rownames(evals))

heatmap(as.matrix(evals[exprows,]),)
```



Heatmap adjustments

- We can provide a colour legend for the samples
- Adjust colour of cells

```
library(RColorBrewer)
sampcol <- rep("blue", ncol(evals))
sampcol[subjects$er == 1 ] <- "yellow"
rbPal <- brewer.pal(10, "RdBu")
heatmap(as.matrix(evals[exprows,]),ColSideColors = sampcol,
col=rbPal)
```



- see also
  - `heatmap.2` from `library(gplots)` ; `example(heatmap.2)`
  - `heatmap.plus` from `library(heatmap.plus)` ; `example(heatmap.plus)`

Exercise

This analysis is recorded in `exercise10.Rmd` .

- Use “in-line” R code to report how many patients were involved in the study
- Hide the code chunk used to produce the plot (`echo=FALSE`)
- Cache the code chunk used to read the raw data (`cache=TRUE`)

End of Course

Wrap-up

- Thanks for your attention
- Practice, practice, practice
  - ….& persevere
- Need inspiration? R code is freely-availabe, so read other peoples’ code!
  - Read blogs (http://www.r-bloggers.com/)
  - Follow the forums (http://stackoverflow.com/questions/tagged/r)
  - Download datasets (http://vincentarelbundock.github.io/Rdatasets/datasets.html) to practice with
  - Bookmark some reference (https://en.wikibooks.org/wiki/R_Programming) guides
  - on twitter @rstudio, @Rbloggers, @RLangTip