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 Sep 2015

true

Day 2 Schedule

1. Further customisation of plots
2. Statistics
3. Data Manipulation Techniques
4. 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
- Set your working directory to ***Day_2_scripts***

The painter’s model

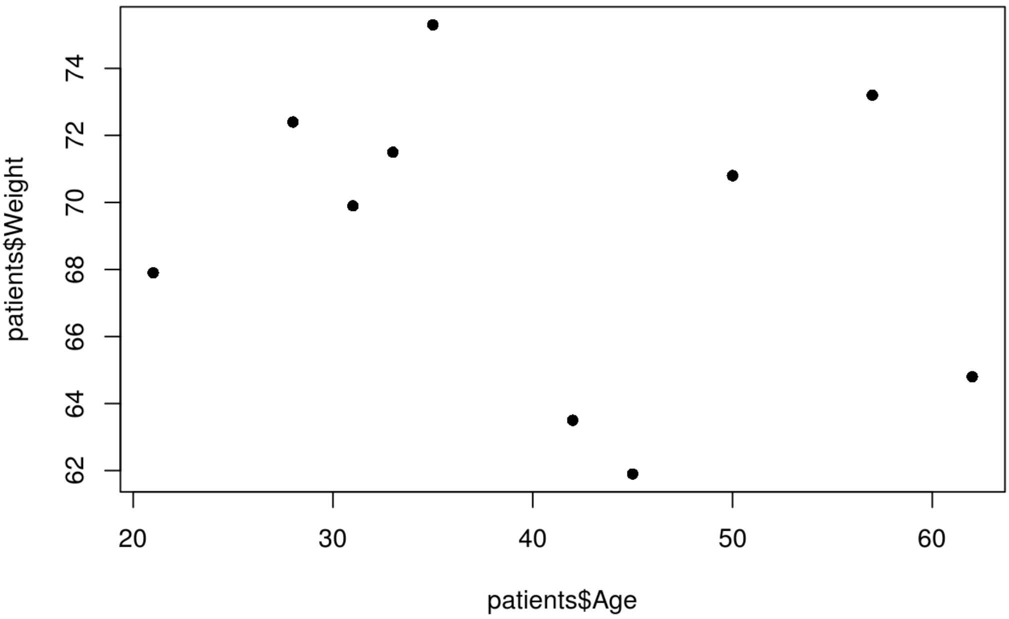
- R employs a painter’s model to construct it’s plots
- Elements of the graph are added to the canvas one layer at a time, and the picture built up in levels.
- Lower levels are obscured by higher levels,
    - allowing for blending, masking and overlaying of objects.

Initial plot

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

```
source("../Day_1_scripts/1.2_patients.R")
```

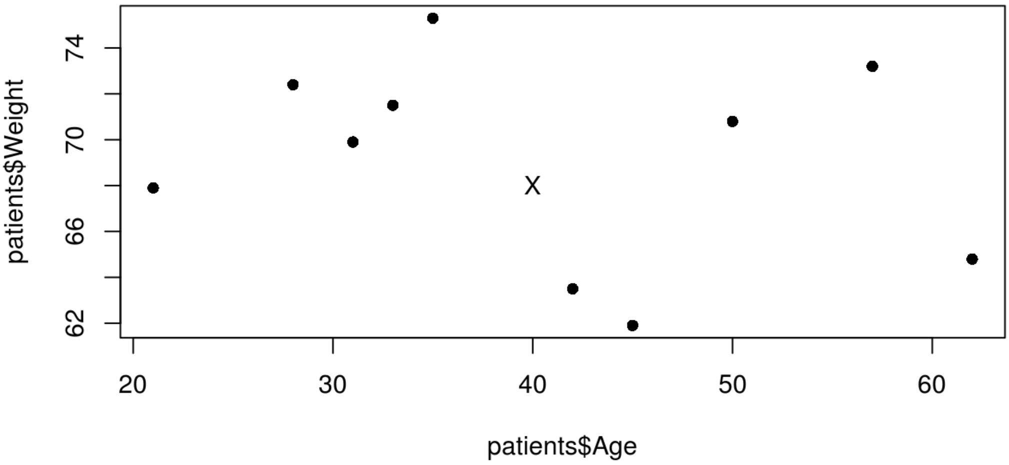
```
plot(patients$Age, patients$Weight,pch=16)
```



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

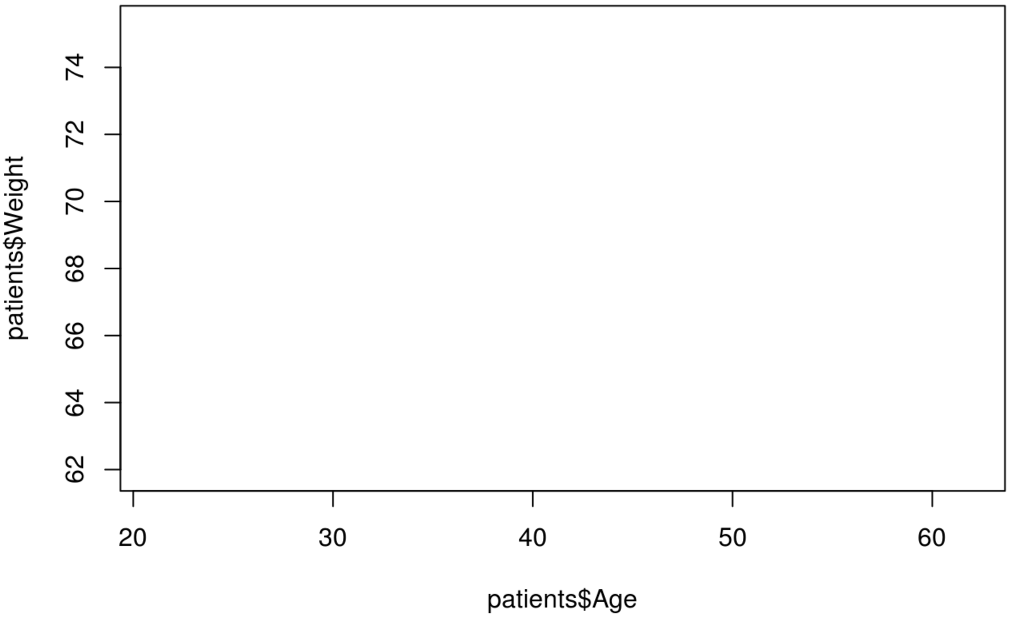
```
plot(patients$Age, patients$Weight,pch=16)
points(40,68,pch="X")
```



Creating a blank plot

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

```
plot(patients$Age, patients$Weight,pch=16,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  TRUE FALS
E
```

```
patients[males,]
```

```
##   First_Name Second_Name     Full_Name  Sex Age Weight Consent
## 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"]
```

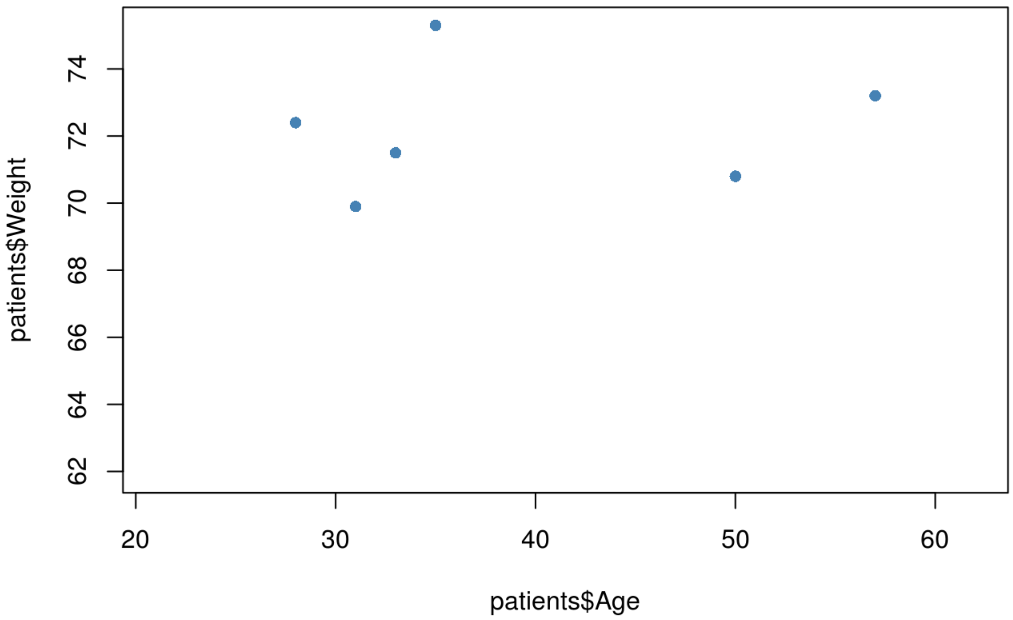
```
## [1] 50 35 28 31 33 57
```

```
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,pch=16,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,col="ste
elblue")
```



Adding points to differentiate gender

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

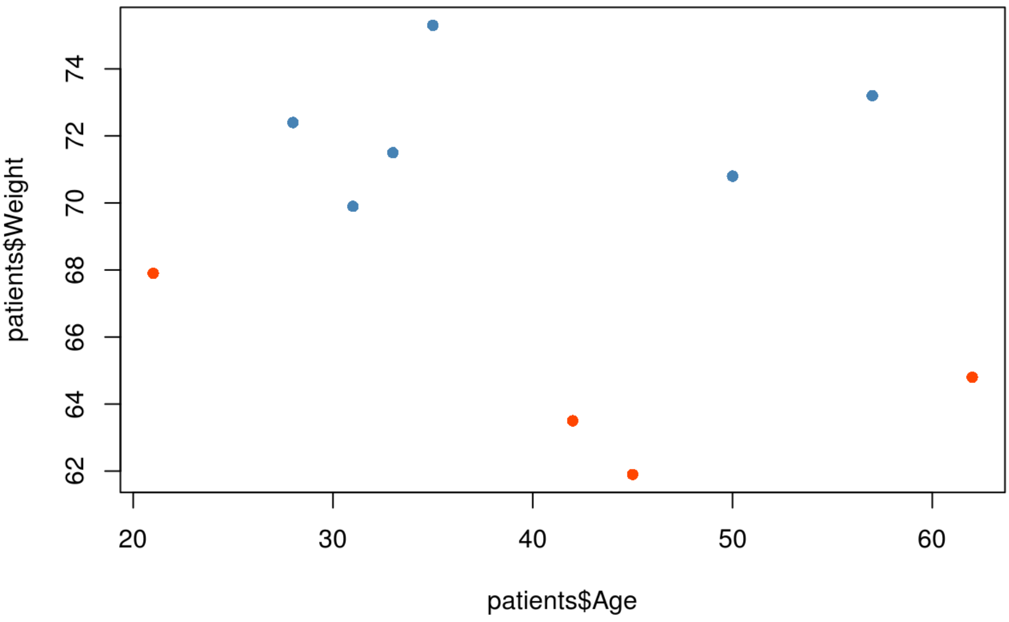
```
##  [1] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRU
E
```

```
patients[females,]
```

```
##    First_Name Second_Name      Full_Name    Sex Age Weight Cons
ent
## 2         Eve      Parker     Eve Parker Female  21   67.9     T
RUE
## 4        Mary       Davis     Mary Davis Female  45   61.9     T
RUE
## 7      Joanna     Edwards Joanna Edwards Female  42   63.5    FA
LSE
## 10      Sally      Wilson   Sally Wilson Female  62   64.8     T
RUE
```

Adding points to differentiate gender

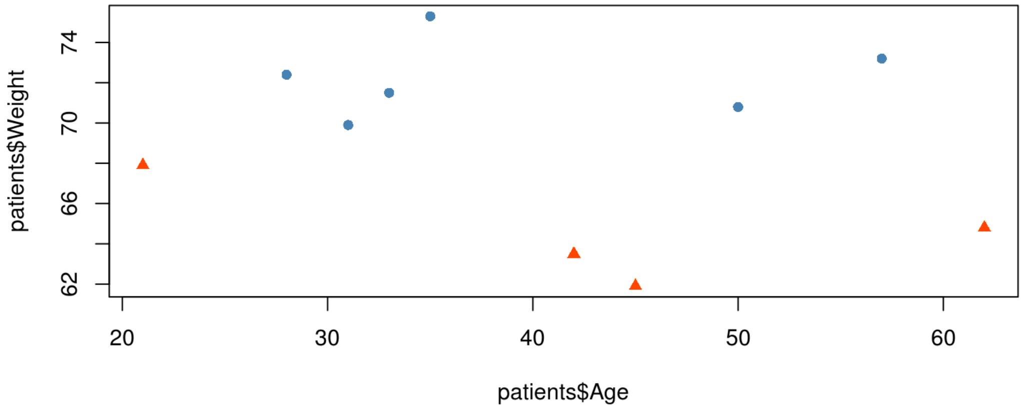
```
plot(patients$Age, patients$Weight,pch=16,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,col="ste
elblue")
points(patients$Age[females], patients$Weight[females],pch=16,col=
"orangered1")
```



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()`

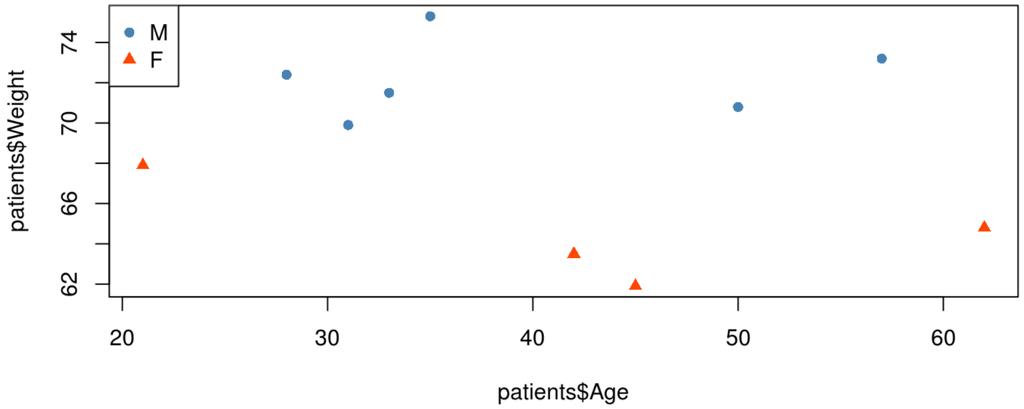
```
plot(patients$Age, patients$Weight,pch=16,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,col="steelblue")
points(patients$Age[females], patients$Weight[females],pch=17,col="orangered1")
```



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,pch=16,type="n")
points(patients$Age[males], patients$Weight[males],pch=16,col="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))
```

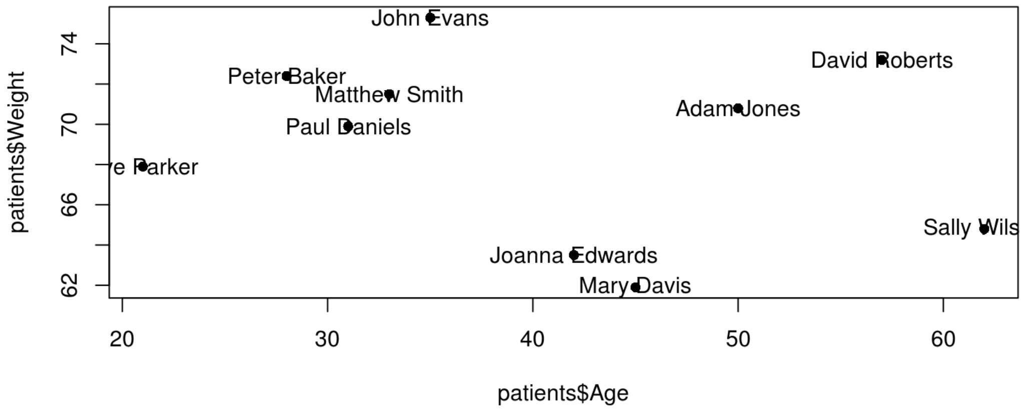


Adding text

- text can also be added to a plot in a similar manner

- the `labels` argument specifies the text we want to add

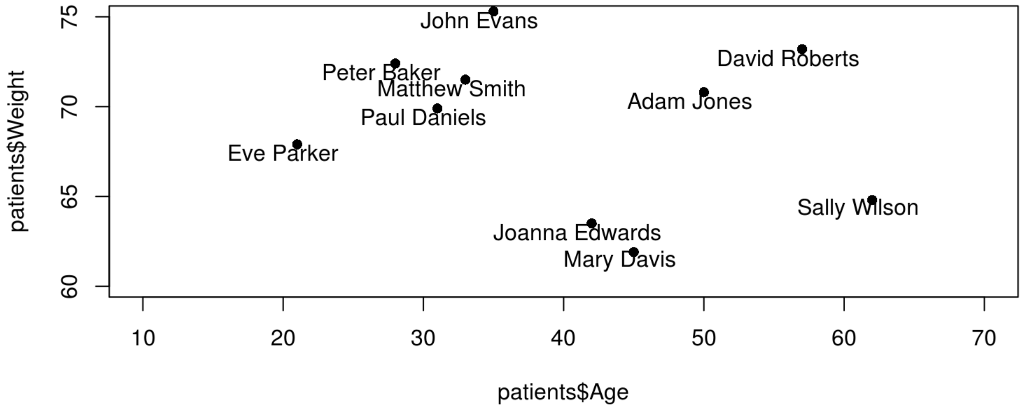
```
plot(patients$Age, patients$Weight,pch=16)
text(patients$Age, patients$Weight,labels=patients$Full_Name)
```



Adding text

- can alter the positions so they don't interfere with the points of the graph

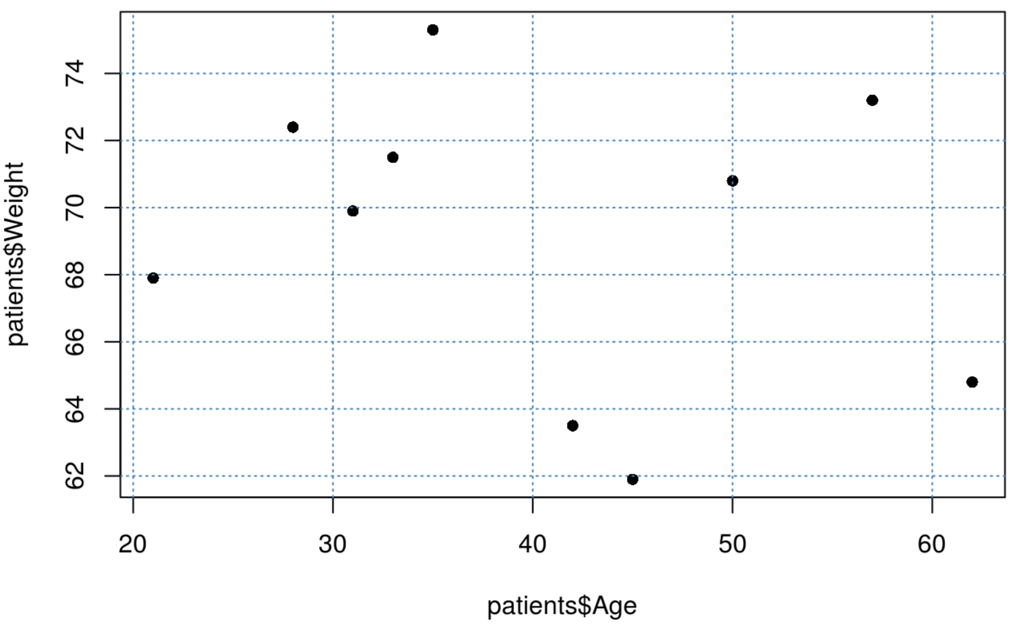
```
plot(patients$Age, patients$Weight,pch=16,xlim=c(10,70),ylim=c(60,
75))
text(patients$Age-1, patients$Weight-0.5,labels=patients$Full_Name
)
```



Adding lines

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

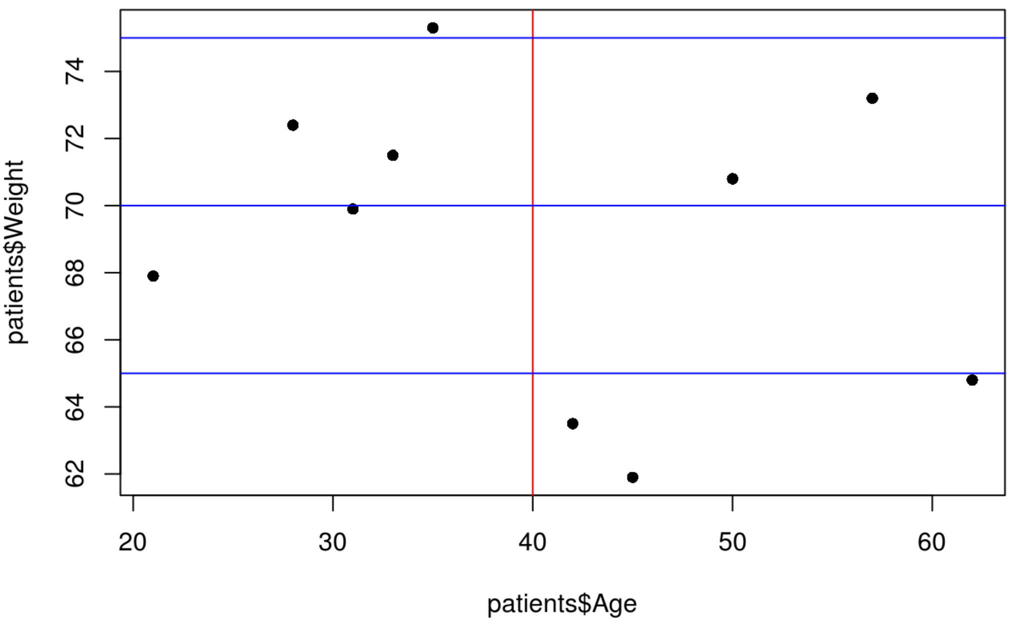
```
plot(patients$Age, patients$Weight,pch=16)
grid(col="steelblue")
```



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")
```

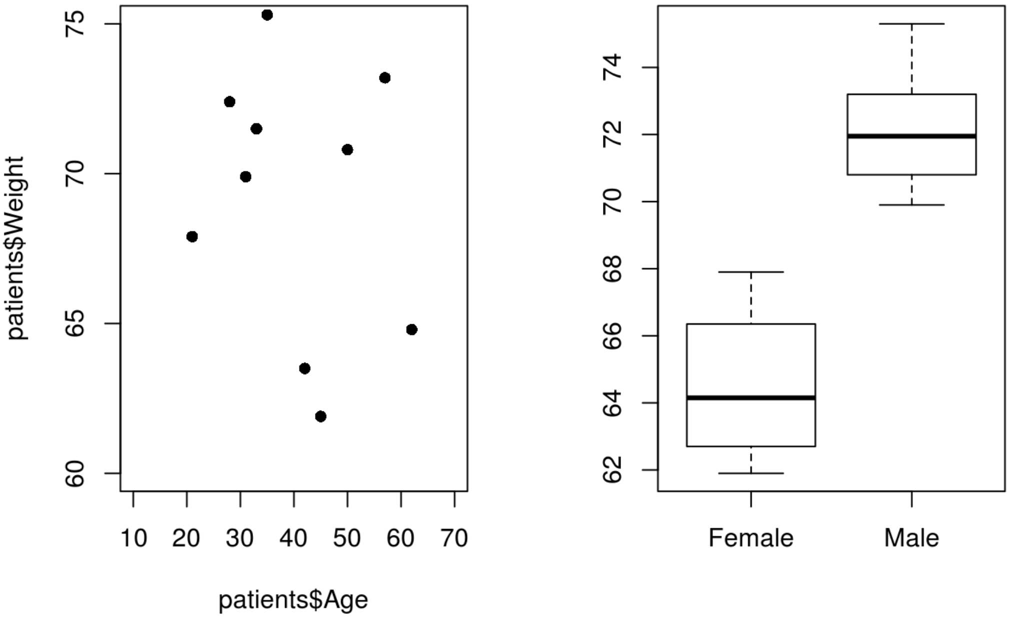


Plot layouts

- 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),ylim=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
```

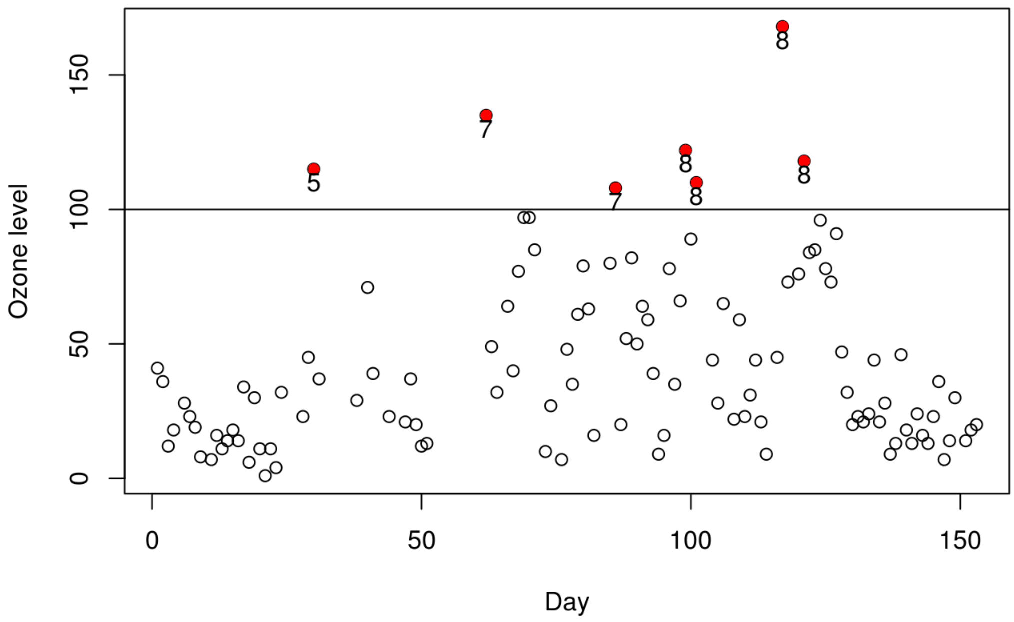
Exercise

- Return to the weather data from yesterday

```
weather <- read.csv("../Day_1_scripts/ozone.csv")
```

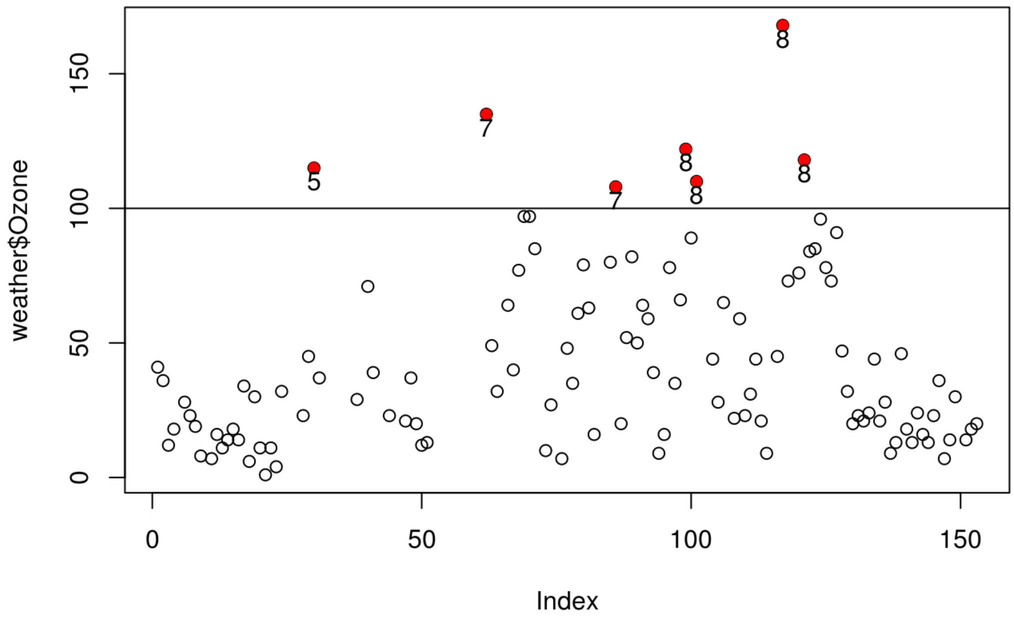
- Make a scatter plot of all daily observations of Ozone level
- Highlight any days which had Ozone level > 100
- Indicate which month the days with high ozone-level belong to

Target Graph



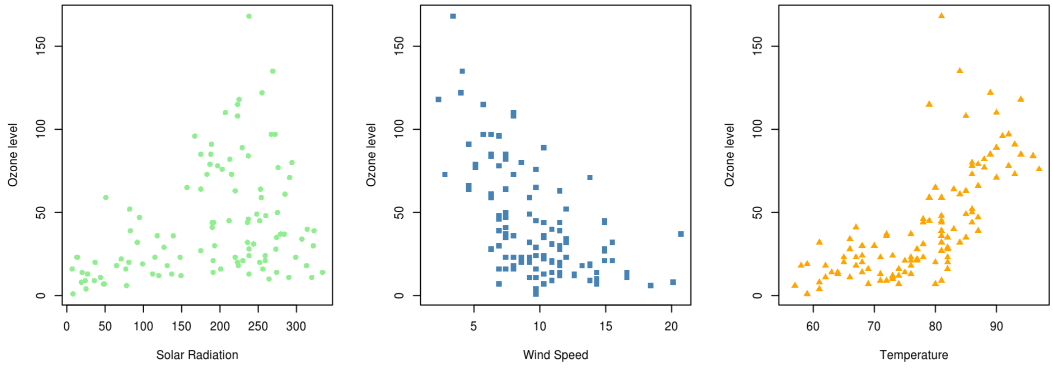
Solution

```
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[highO])
```



Exercise

- 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

```
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",ylab="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",ylab="Ozone level", xlab="Wind Speed")
plot(weather$Temp,weather$Ozone,pch=17,col="orange", ylab="Ozone level",xlab="Temperature")
dev.off()
```

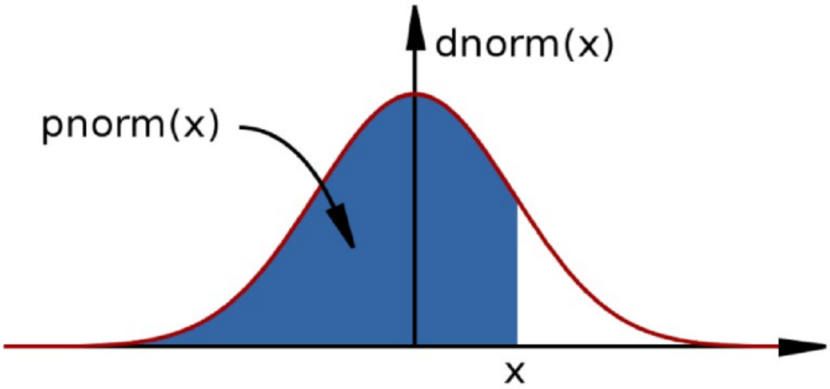
2. Statistics

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` (hypergeometric), `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)
```

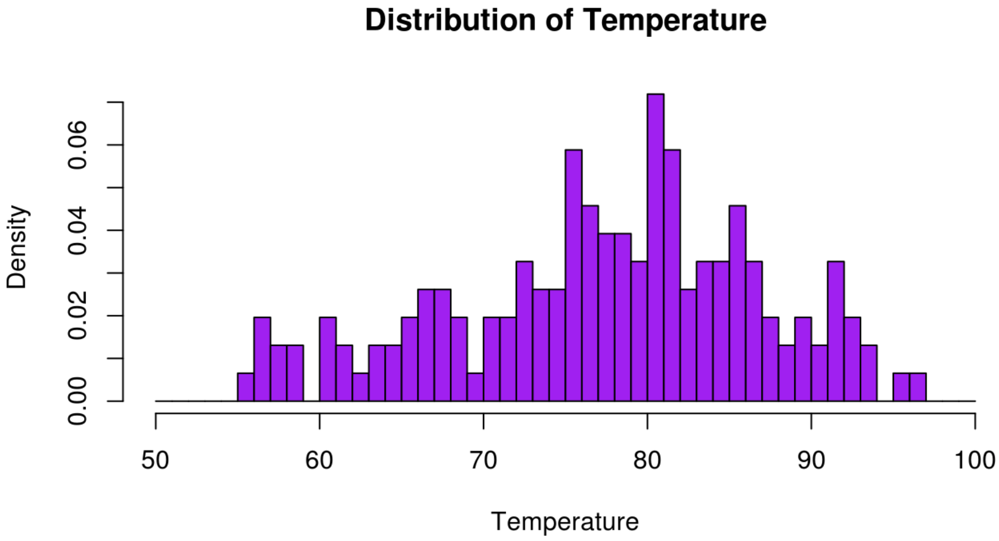
```
## [1] 1.644854
```

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
)
```



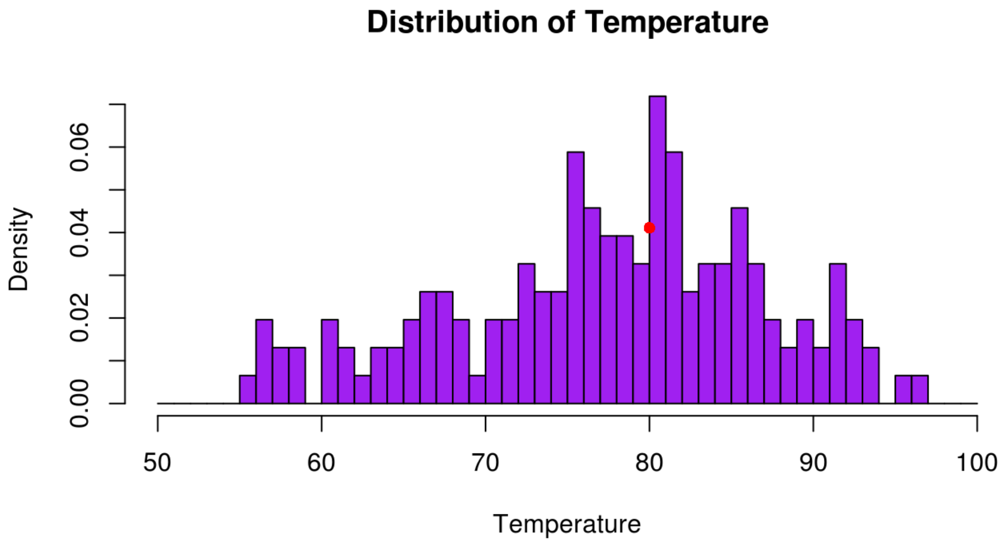
Create a normal distribution curve

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

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

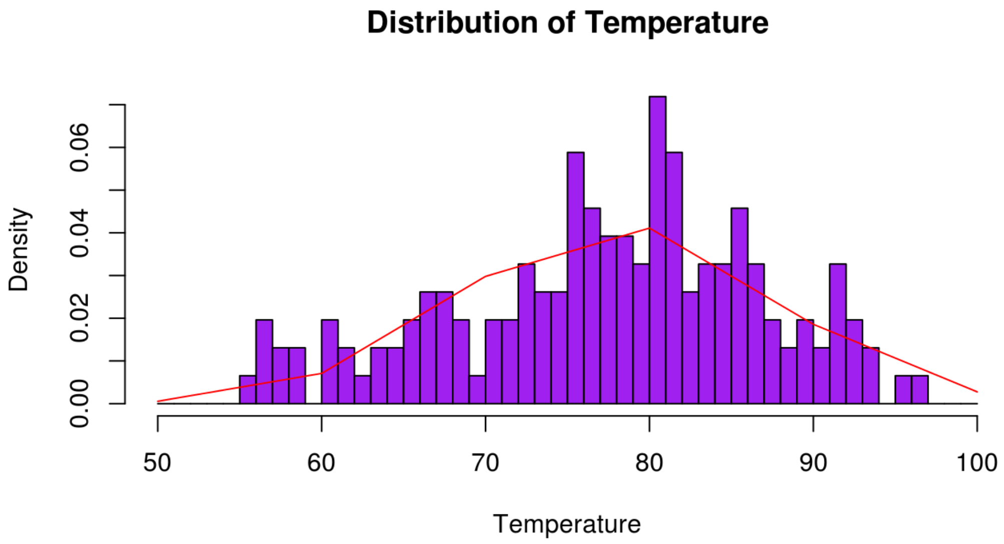
```
## [1] 0.04110626
```



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`

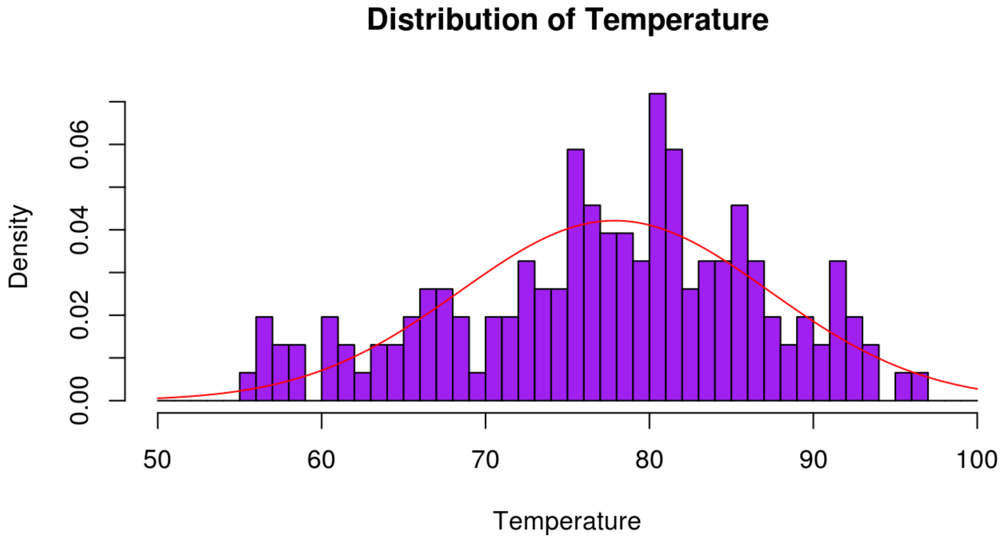
```
xs <- c(50,60,70,80,90,100)
ys <- dnorm(xs, mean=tempMean,sd=tempSD)
lines(xs,ys,col="red",pch=16)
```



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

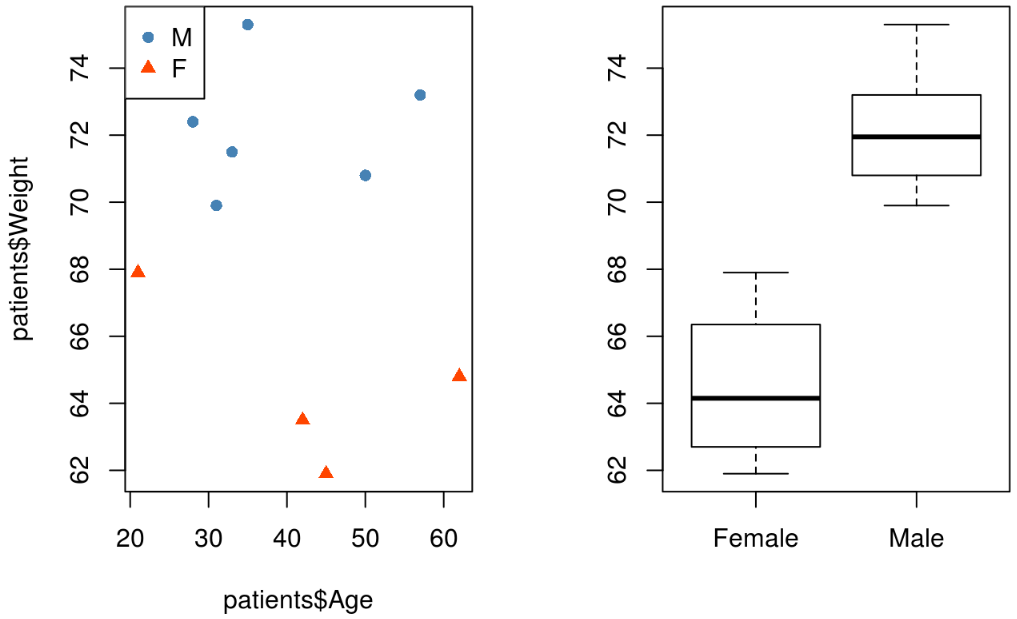
```
chisq.test();fisher.test()
```

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 equal 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 t
o 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

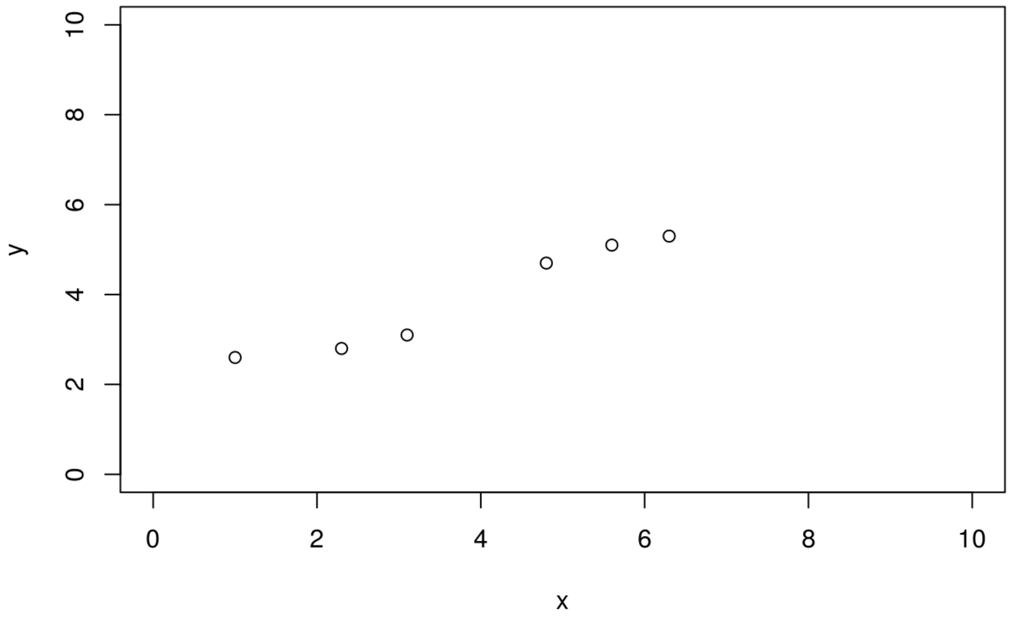
- Linear modeling is supported by the function `lm()`
    - `example(lm)` the output assumes you know a fair bit about the subject
- `lm` is really useful for plotting lines of best fit to XY data in order to determine intercept, gradient & Pearson’s correlation coefficient
    - This is very easy in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

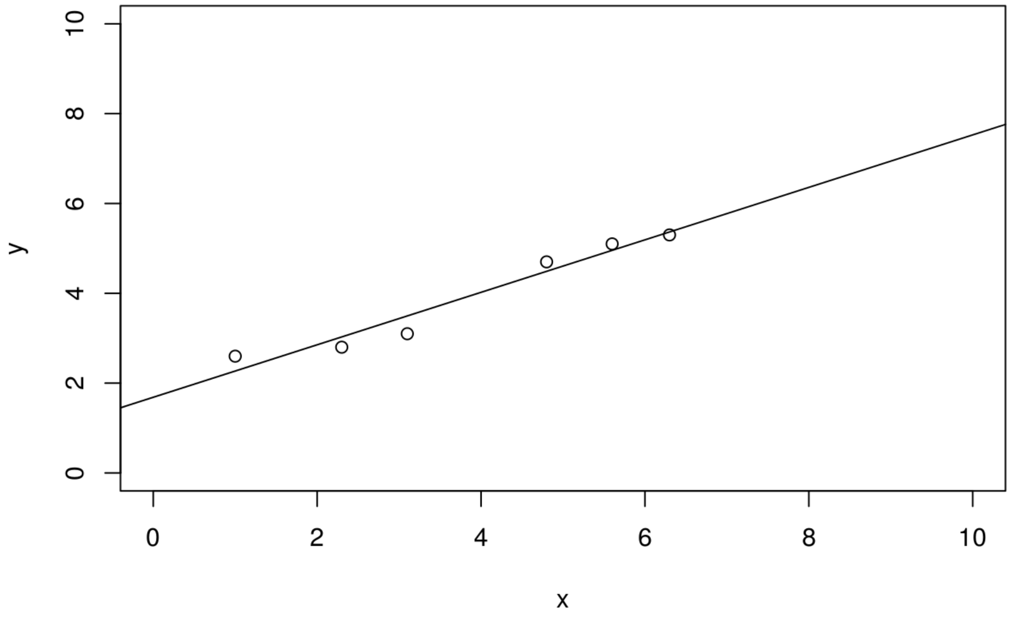
```
x <- c(1, 2.3, 3.1, 4.8, 5.6, 6.3)
y <- c(2.6, 2.8, 3.1, 4.7, 5.1, 5.3)
plot(x,y, xlim=c(0,10), ylim=c(0,10))
```



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  0.14434418 -0.
06458482
```

```
fitted(myModel)
```

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

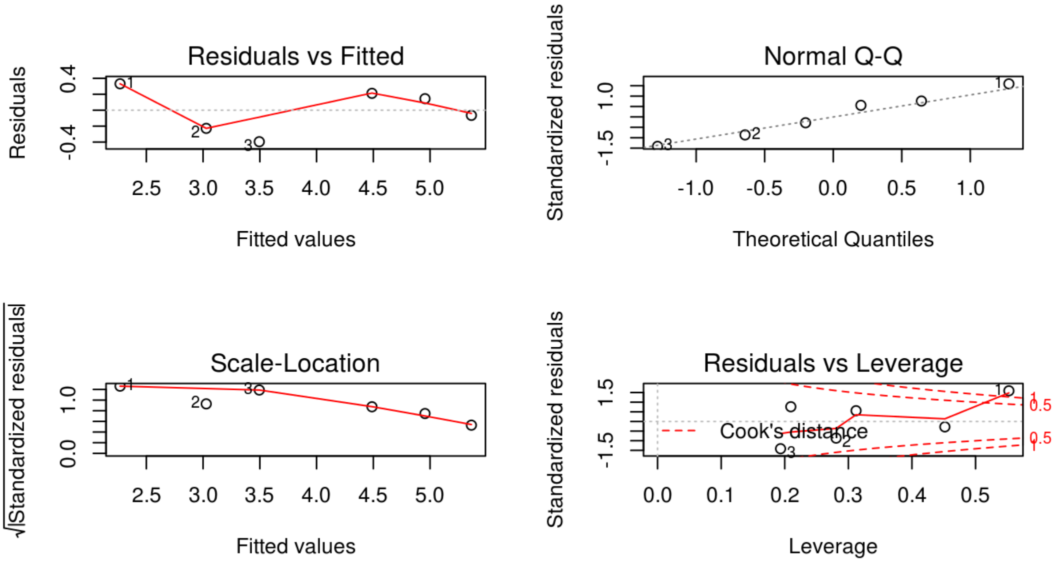
```
names(myModel)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"         
##  [5] "fitted.values" "assign"        "qr"            "df.residu
al"
##  [9] "xlevels"       "call"          "terms"         "model"
```

Diagnostic plots of the fit

- Get QC of fit from

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



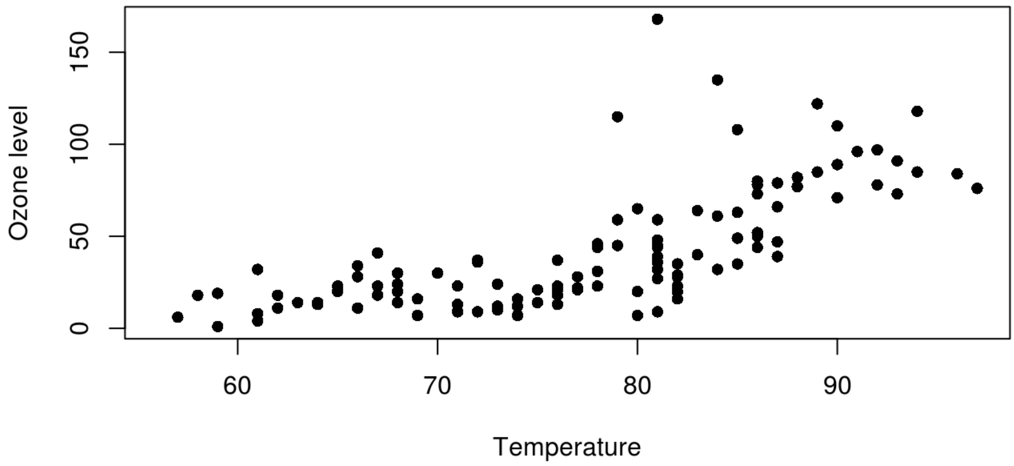
Modelling formulae

- 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

- 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

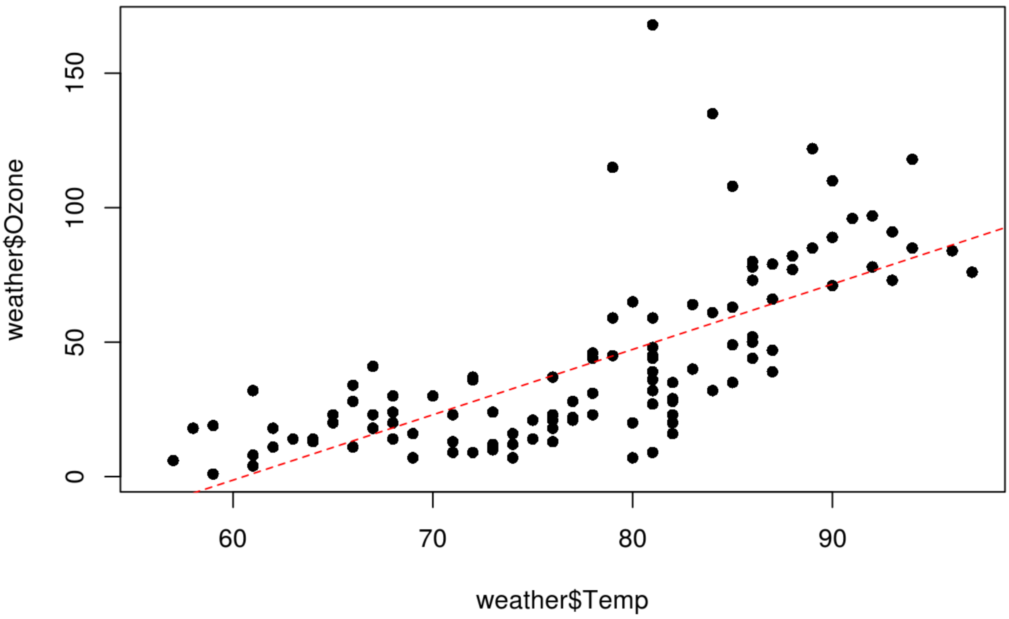
Solution

```
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
  - A dataset may be spread over severall 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 ***Day_2_scripts/gene.expression.txt***
  - patient information in ***Day_2_scripts/gene.description.txt***
  - gene information in ***Day_2_scripts/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?

Analysis sketch

- Read the relevant files
- Do a sanity check to see if a known gene exhibits a change between groups
- Test a list of candidate genes to see if they show a significant change
- Make heatmaps of our genes

The R scripting language

- A script is a series of instructions that when executed sequentially automates a task
    - A script is a good solution to a repetitive problem
- The art of good script writing is
    - understanding exactly what you want to do
    - expressing the steps as concisely as possible
    - making use of error checking
    - including descriptive comments
- R is a powerful scripting language, and embodies aspects found in most standard programming environments
    - procedrual statements
    - loops
    - functions
    - conditional branching
- Scripts may be written in any standard text editor, e.g. notepad, gedit, kate
    - we will use RStudio

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
## AF155654         0.050 0.075  0.039 -0.099 -0.098
## Contig56678_RC -0.261 0.346  0.047 -1.140 -0.110
## Contig36647_RC  0.848 0.600 -0.984  0.077 -0.150
## Contig48552_RC  0.144 0.160 -0.055  0.064  0.192
## AB033037         0.228 0.008 -0.019  0.004  0.232
```

```
## [1] 550 337
```

- 550 rows and 337 columns
- One row for each gene
    - rows are named according to particular technology used to make measurement
- One column for each patient

Peek at the data

```
genes <- read.delim("gene.description.txt")
head(genes)
dim(genes)
```

```
##                    probe HUGO.gene.symbol
## AA017147_RC AA017147_RC             <NA>
## AA024493_RC AA024493_RC             <NA>
## AA043429_RC AA043429_RC         C17orf27
## AA470152_RC AA470152_RC             <NA>
## AA485939_RC AA485939_RC             <NA>
## AA490225_RC AA490225_RC             <NA>
```

```
## [1] 24481     2
```

- 24481 rows and 2 columns
- One for for each gene
    - more rows than genes we have measurements for
- Includes mapping between manufacturer ID and Gene name

Peek at the data

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

```
##         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_337    NKI_337  29  1     1
## NKI_48      NKI_48  30  0     3
```

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

- One for 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
```

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 FALSE FALS
E FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E 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[,2])
genes[ind,]
```

```
##               probe HUGO.gene.symbol
## NM_000125 NM_000125             ESR1
```

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

```
## [1] NM_000125
## 24481 Levels: AA017147_RC AA024493_RC AA043429_RC AA470152_RC .
.. zero_barcode
```

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] 428
```

```
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
```

```
evals[428,1:10]
```

```
##            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
```

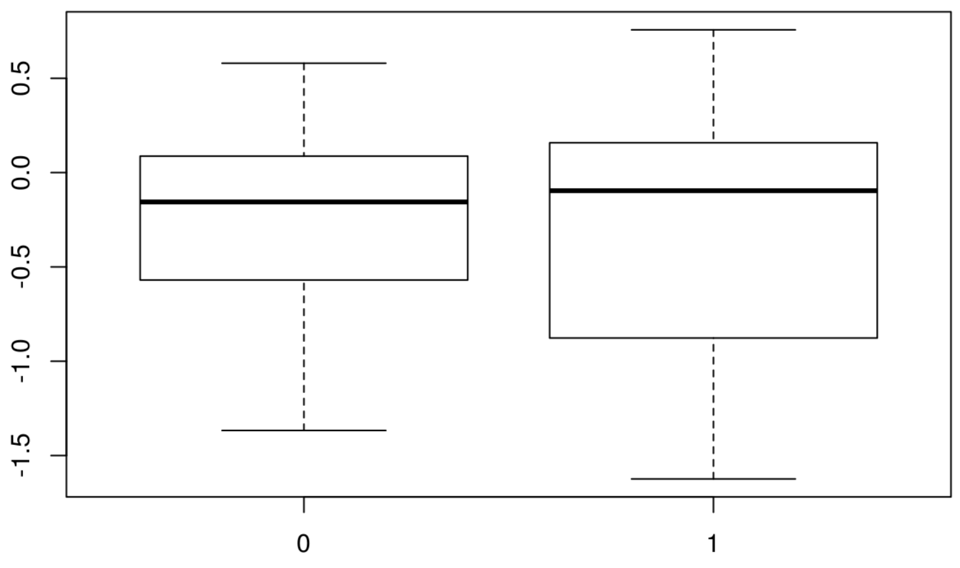
```
genevals <- evals[match(probe,rownames(evals)),]
```

Relating to patient characteristics

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

- use a boxplot, for example
- something doesn’t look correct though….

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



Relating to patient characteristics

- the p-value is not encouraging either
  - what could have gone wrong?

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

```
##
##  Welch Two Sample t-test
##
## data:  as.numeric(genevals) by factor(subjects$er)
## t = 0.21768, df = 171.94, p-value = 0.8279
## alternative hypothesis: true difference in means is not equal t
o 0
## 95 percent confidence interval:
##  -0.1219951  0.1522385
## sample estimates:
## mean in group 0 mean in group 1
##      -0.2833864      -0.2985081
```

The right patients, but not neccessarily in the right order!

- the problem is that our expression values are not in the same order as the clinical data
  - a common occurence, sadly :/

```
colnames(evals)[1:10]
```

```
##  [1] "NKI_4"  "NKI_6"  "NKI_7"  "NKI_8"  "NKI_9"  "NKI_11" "NKI
_12"
##  [8] "NKI_13" "NKI_14" "NKI_17"
```

```
subjects$samplename[1:10]
```

```
##  [1] NKI_330 NKI_57  NKI_230 NKI_90  NKI_337 NKI_48  NKI_56  NK
I_85
##  [9] NKI_86  NKI_92
## 337 Levels: NKI_100 NKI_102 NKI_103 NKI_104 NKI_106 NKI_107 ...
 NKI_99
```

```
colnames(evals)[1:10] == subjects$samplename[1:10]
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
E
```

Re-ordering both matrices

- match returns a vector of the positions of (first) matches of its first argument in its second.
- recall that we can use a vector to subset, or order, the columns of a data frame

```
match(subjects$samplename,colnames(evals))[1:10]
```

```
##  [1] 270  35 181  57 277  32  34  53  54  59
```

- the output indicates that column 270 of the expression matrix represents the same data as row 1 of the `subjects` matrix
- column 35 of the expression matrix represents the same data as row 2 of the `subjects` matrix
- etc….

Re-ordering both matrices

- we can use this vector to index our expression matrix
  - remembering to omit the row index
- sanity check to make sure we've done the job
- match the `probe` index as before
  - we haven't changed the rows so this should be the same

```
evals <- evals[,match(subjects$samplename,colnames(evals))]
genevals <- evals[match(probe,rownames(evals)),]
colnames(evals)[1:10] == subjects$samplename[1:10]
```

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

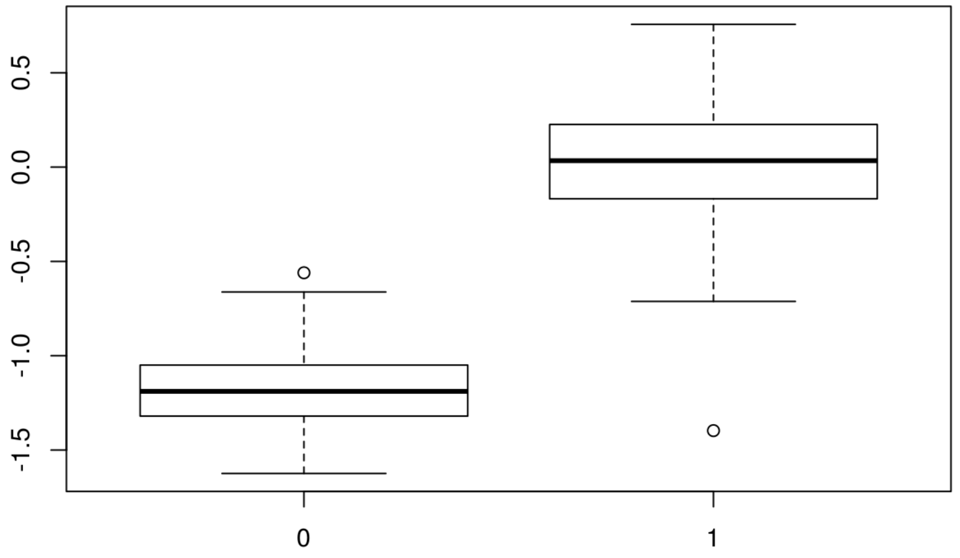
```
all(colnames(evals) == subjects$samplename)
```

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

Try again

- Better!

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



Try again

- Much better

```
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 t
o 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

`gene-analysis.R`

```
genes <- read.delim("gene.description.txt")
subjects <- read.delim("cancer.patients.txt")
evals <- read.delim("gene.expression.txt",stringsAsFactors = FALSE
)
evals <- evals[,match(subjects$samplename,colnames(evals))]

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))
```

(short) Exercise

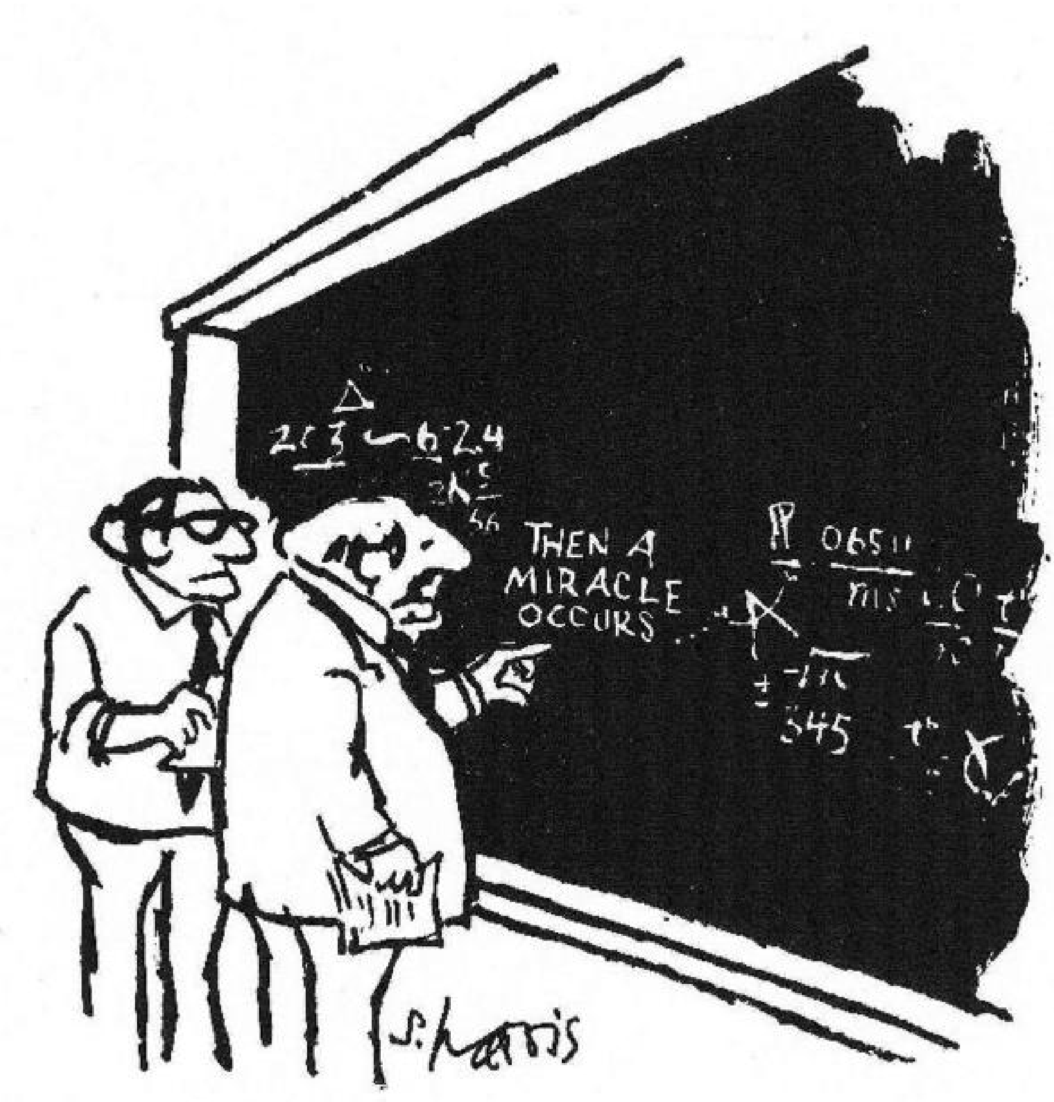
- make sure that you can run the script, and can understand it
- modify the script to analyse `NAT1`

There are a number of issues that we will address this afternoon

- How can we handover the results of our analysis to our collaborators
    - copy and paste the graphs into a word document!!?
    - …..what happens if the data change?
- Imagine if we have a list of 100 genes to test
    - what a pain it would be to change the script for each gene

4. Report Writing

Principles of Reproducible Research



"I think you should be more explicit here in step two."

Sidney Harris - New York Times

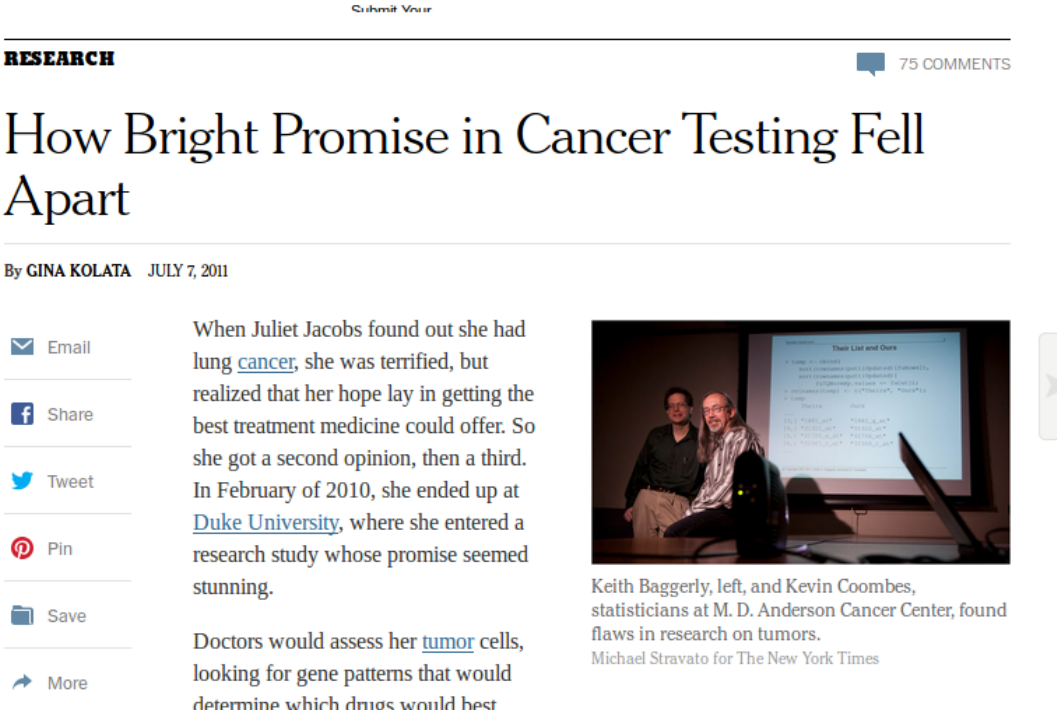
Why should we do reproducible research?

Five selfish reasons - Florian Markowetz Blog (https://scientificbsides.wordpress.com/2015/07/15/five-selfish-reasons-for-working-reproducibly/) and slides (http://f1000research.com/slides/4-207)

1. Avoid disaster
2. Easier to write papers
3. Easier to talk to reviewers
4. Continuity of your work in the lab
5. Reputation

It is a hot topic at the moment

- Statisticians at MD Anderson tried to reproduce results from a Duke paper and unintentionally unravelled a web of incompetence and skullduggery
  - as reported in the ***New York Times***



RESEARCH

How Bright Promise in Cancer Testing Fell Apart

By GINA KOLATA JULY 7, 2011

When Juliet Jacobs found out she had lung cancer, she was terrified, but realized that her hope lay in getting the best treatment medicine could offer. So she got a second opinion, then a third. In February of 2010, she ended up at Duke University, where she entered a research study whose promise seemed stunning.

Doctors would assess her tumor cells, looking for gene patterns that would determine which drugs would best

Keith Baggerly, left, and Kevin Coombes, statisticians at M. D. Anderson Cancer Center, found flaws in research on tumors.
Michael Stravato for The New York Times

Hear the full account

- Very entertaining talk from Keith Baggerly in Cambridge, December 2010

The Importance of Reproducible Research in High-Throug…

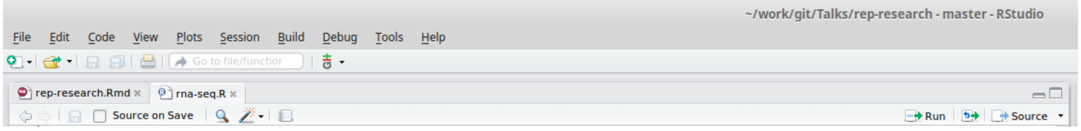
What can we do about it?

- Use scripts / R
- Use version control
- Document early
- Document everything
- Write comments and explanations
- Automatically-generate your plots, tables, etc from the data
    - always ensure that you have the latest version

Simple example in RStudio

- Lets take gene-analysis.R

- Use the Compile Notebook button in RStudio
- Take an R script and turn into HTML, PDF or even Word
- All code will be displayed and the outputs printed
- A compiled report will be generated in your working directory



What is going on?

- The `knitr` package is being used convert the R script into ‘markdown’ format, which it then compiles into the output of your choosing
- `knitr` is distributed with RStudio
    - `knitr` is the modern-day equivalent of `Sweave`
- markdown is a easy-to-read, easy-to-write text format often used to write HTML, readme files etc
- the following should create the file `gene-analysis.Rmd` in the Day_2_scripts folder

```
library(knitr)
spin(hair="gene-analysis.R",knit=FALSE)
```

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
    - Lets add this to the gene-analysis.R script and see what happens

```
sessionInfo()
```

```
## R version 3.2.0 (2015-04-16)
## 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  methods  
 base
##
## loaded via a namespace (and not attached):
##  [1] magrittr_1.5    formatR_1.2     tools_3.2.0     htmltools_
0.2.6
##  [5] yaml_2.1.13     stringi_0.5-5   rmarkdown_0.6.1 knitr_1.10
.5
##  [9] stringr_1.0.0   digest_0.6.8    evaluate_0.7
```

Create a markdown file from scratch

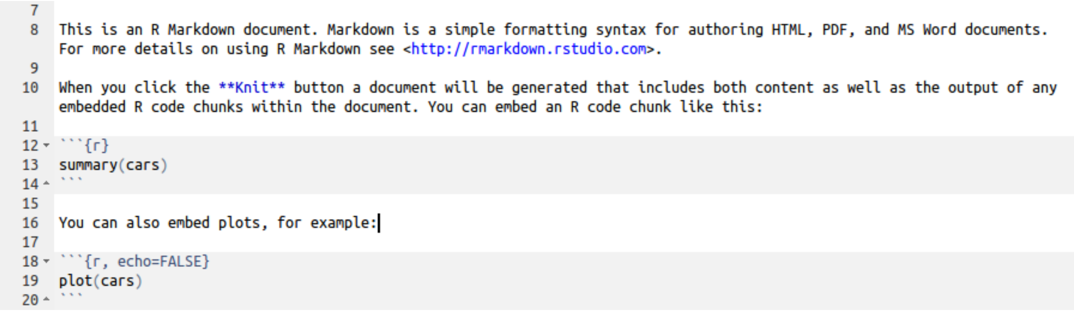
File - > New File - > R Markdown

- 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



~~~~
 7
 8   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>.
 9
10   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:
11
12   ```{r}
13   summary(cars)
14   ```
15
16   You can also embed plots, for example:
17
18   ```{r, echo=FALSE}
19   plot(cars)
20   ```
~~~~

- 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

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, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
##
## The following object is masked from 'package:stats':
##
##     xtabs
##
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, as.vector, cbind,
##     colnames, do.call, duplicated, eval, evalq, Filter, Find, g
et,
##     intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, ran
k,
##     rbind, Reduce, rep.int, rownames, sapply, 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("pkgname"
)'.
##
## Loading required package: locfit
## locfit 1.5-9.1    2013-03-22
## Loading required package: lattice
##     Welcome to 'DESeq'. For improved performance, usability 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

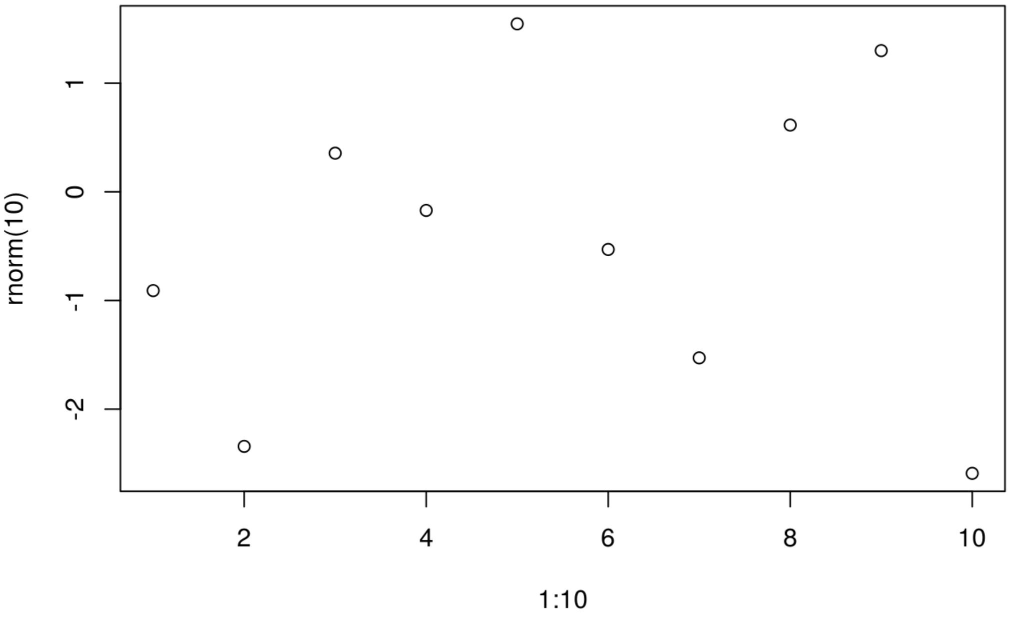
- `cache=TRUE` will stop certain chunks from being evaluate if their code does not change
- speeds-up the compilation of the document
    - we don’t want to reload our dataset if we’ve only made a tiny change downstream

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

Including plots

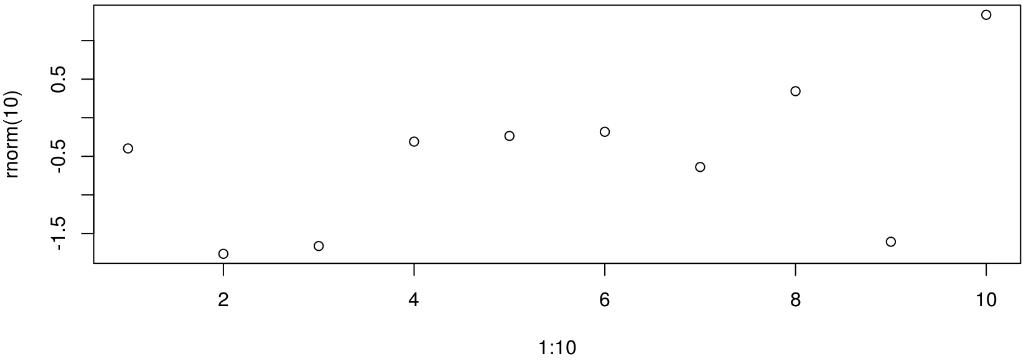
- Use a plotting function (`plot` , `boxplot` , `hist` etc) will include the plot at the relevant point in the document

```
'''{r}
plot(1:10, rnorm(10))
'''
```



Control over plots

```
'''{r fig.height=2,fig.align='right', fig.height=4,fig.width=9}
plot(1:10, rnorm(10))
'''
```



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]' fema
les 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]' fema
les 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…..

References

- Useful reference:
    - Reproducible Research in R and RStudio
        - http://christophergandrud.github.io/RepResR-RStudio/ (http://christophergandrud.github.io/RepResR-RStudio/)
        - Useful exercise is to compile the book from the source code (https://github.com/christophergandrud/Rep-Res-Book)
    - Implementing Reproducible Research (https://osf.io/s9tya/wiki/home/)
    - FYI This course was generated from R markdown files
        - see our github repository (https://github.com/cambiotraining/r-intro)

Exercise

- Create a new markdown file that will report your analysis of the expression of ESR1 in breast cancer
    - File -> New File -> R Markdown
- Create separate code chunks for each stage of the analysis (copy code from gene-analysis.R if you wish)
    - Reading the data
    - Re-order the gene expression matrix

    - Extract data for ESR1
    - Produce a boxplot
    - Perform the t-test

Exercise

Once you have a report that you are happy with

- 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`)
- Hide code used to perform the t-test and the output of the test. However, use “in-line” R code to report the p-value of the t-test
- Solution: *esr1-analysis.Rmd*

We will now see how to modify the report to analyse multiple genes

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"    "es
timate"
## [6] "null.value"  "alternative" "method"      "data.name"
```

```
t$statistic
```

```
##         t
## -10.60817
```

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] -10.6081734 -20.1254643 -17.0940420  -2.4298943  -1.855833
7
##  [6]  -4.5176798  -0.4136376  -0.2214786  -1.5956797  -1.405284
3
```

Modifying code to work on a gene-list

- What if we want to test and arbitrary gene list
- Supply gene list as a vector
- Add extra steps to locate correct row in matrix

```
stats <- NULL

for(i in 1:length(genelist)){

  probe <- genes[match(genelist[i],genes[,2]),1]
  j <- match(probe,rownames(evals))

  tmp <- t.test(as.numeric(evals[j,])~factor(subjects$er))
  stats[i] <- tmp$statistic

}
```

Example

```
stats <- NULL
genelist <- c("ESR1", "NAT1", "SUSD3","SLC7A2" ,"SCUBE2")

for(i in 1:length(genelist)){

  probe <- genes[match(genelist[i],genes[,2]),1]
  j <- match(probe,rownames(evals))

  tmp <- t.test(as.numeric(evals[j,])~factor(subjects$er))
  stats[i] <- tmp$statistic

}
stats
```

```
## [1] -38.74628 -22.75125 -20.88450 -21.31987 -20.97641
```

Introducing error-checking

- what happens when we try and run this?
- should see an error as ***foo*** and ***bar*** are not defined in our gene table
    - the error message is often obtuse
    - we can anticipate this kind of error in advance

```
stats <- NULL
genelist <- c("foo", "bar")

for(i in 1:length(genelist)){

  probe <- genes[match(genelist[i],genes[,2]),1]
  j <- match(probe,rownames(evals))

  tmp <- t.test(as.numeric(evals[j,])~factor(subjects$er))
  stats[i] <- tmp$statistic

}
stats
```

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 might want to warn the user if we can't find a particular gene with the pattern we are searching for.
- Here's how we can use an `if` statement to test for this
    - only run the t-test if we can successfully map the gene
        - i.e if we cannot map the probe, then the probes vector would be `NA`
        - only proceed if the `probes` vector is not `NA`

```
stats <- NULL

for(i in 1:length(genelist)){

  probes <- genes[match(genelist[i],genes[,2]),1]
  j <- match(probes,rownames(evals))

  if(!is.na(probes)){
   tmp <- t.test(as.numeric(evals[j,])~factor(subjects$er))
   stats[i] <- tmp$statistic
  } else message("Could not find gene ",genelist[i])

}
```

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

Extracting multiple genes

```
genelist <- c("ESR1", "NAT1", "SUSD3","SLC7A2" ,"SCUBE2","foo")
stats <- NULL
for(i in 1:length(genelist)){

  probes <- genes[match(genelist[i],genes[,2]),1]
  j <- match(probes,rownames(evals))

  if(!is.na(probes)){
   tmp <- t.test(as.numeric(evals[j,])~factor(subjects$er))
   stats[i] <- tmp$statistic

  }
  else message("Could not find gene ", genelist[i])

}
```

```
## Could not find gene foo
```

```
stats
```

```
## [1] -38.74628 -22.75125 -20.88450 -21.31987 -20.97641
```

Adding plots

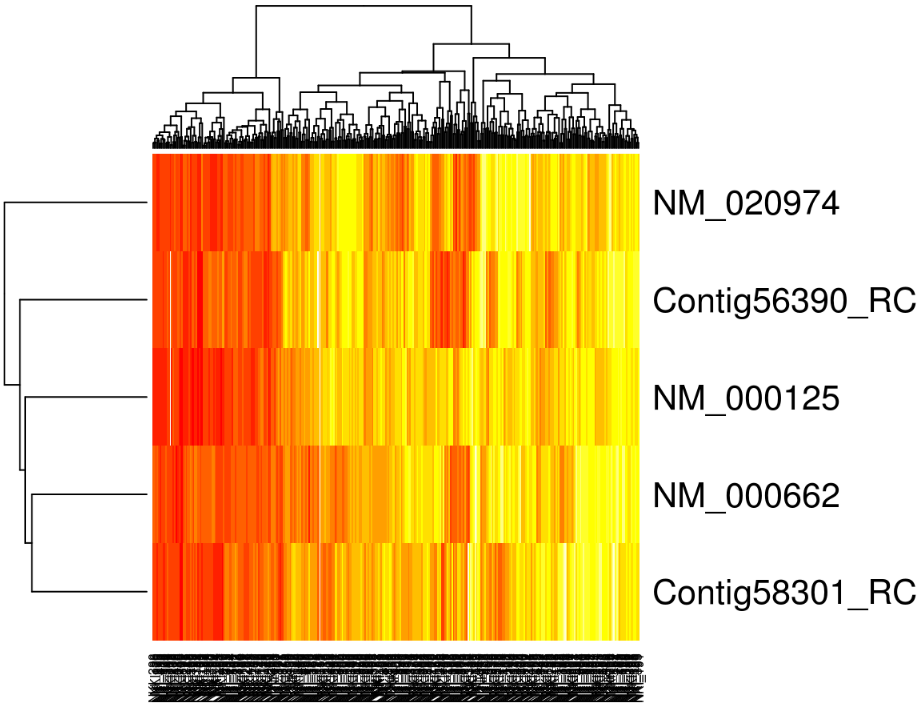
- The number of lines within the loop can be as long as you like *but preferably not too long*
  - we can easily add boxplots to our code

```
genelist <- c("ESR1", "NAT1", "SUSD3","SLC7A2" ,"SCUBE2")
stats <- NULL
for(i in 1:length(genelist)){

  probes <- genes[match(genelist[i],genes[,2]),1]
  j <- match(probes,rownames(evals))

  if(length(probes)>0){
   tmp <- t.test(as.numeric(evals[j,])~factor(subjects$er))
   stats[i] <- tmp$statistic
   boxplot(as.numeric(evals[j,])~factor(subjects$er),main=genelist
[i],xlab="ER Status")
  }

}
stats
```

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!

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

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

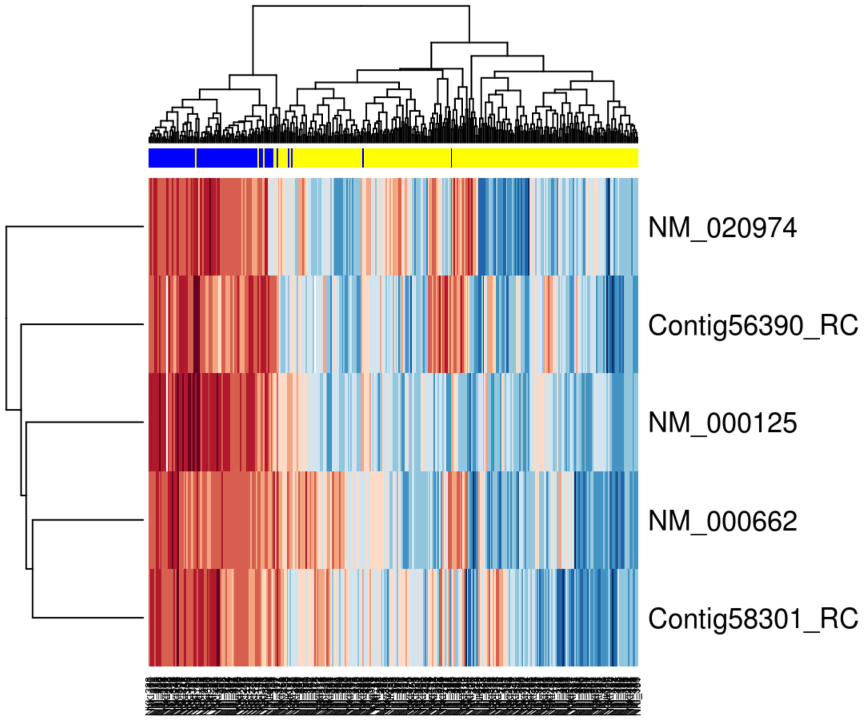


Heatmap adjustments

- 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=rbP
al)
```



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

Compiling the report

- See the markdown document genelist-analysis.Rmd

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