Exercises on Graphics and Data Manipulation in R

Mark Dunning, Mike Smith and Sarah Vowler *

Last Document revision: April 8, 2015

Contents

Introduction

The data that you need for this practical are available at

https://sharepoint.cri.camres.org/sites/bioinformatics/Public/GraphicsAndDataManipulation/CourseData.zip

Please download this zip file and extract to your computer. You should then change your working directory in RStudio to point to the directory that you have just created. Session → Change Working directory.

To accompany this course, we have created a small number of utility and plotting functions and bundled them as a R package. You can install the latest version of this package by the following commands in

*Acknowledgements: Thanks to Fran Richards for supplying example data and figures for the practical

R:

```
install.packages("devtools")
library(devtools)
install_github(repo = "crukCIMisc", username = "markdunning")
library(crukCIMisc)
```

At the end of some subsections you may find some Optional Extensions. You can attempt these if you have time in the practical sessions, or on your own after the course.

1 Plotting basics

This section will provide you with practice on reading several different file formats into R

1.1 Weather Data

The dataset introduced in the slides concerns weather conditions in New York City from the summer of 1973. [1].

Exercise: Read the ozone data into your Rstudio. Practice using the 'file.choose' function to locate the file on your hard drive

```
myfile <- file.choose()
ozone <- read.csv(myfile)
```

Exercise: Verify that the dimensions and first few lines are as we expect. HINT: use the `dim` and `head` functions.

Exercise: Make sure you know how to extract the Wind column from the data frame by i) selecting using the column number ii) selecting using the column name. Verify that you get the same answer.

Exercise: Make a scatter plot with index on the x-axis and Wind speed on the y-axis

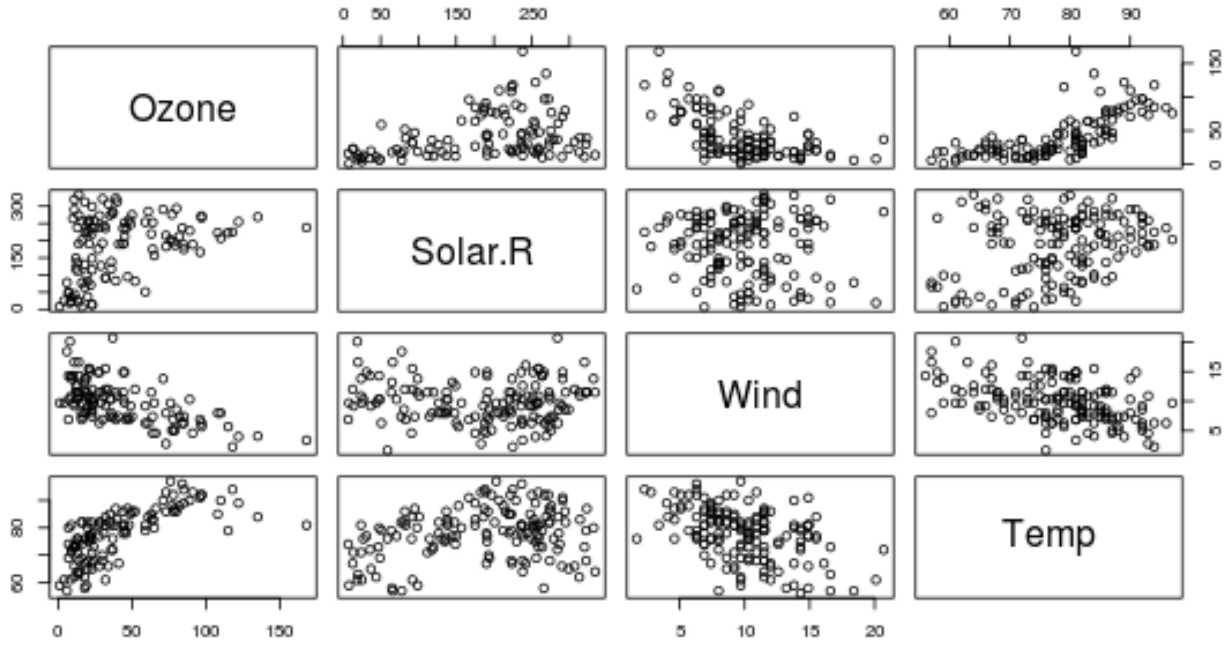
Exercise: Make a scatter plot to compare Wind speed and temperature

The `plot` function in R is flexible and will try to guess the most appropriate type of plot based on the data that you give it. If given a data frame with numeric data, it will make pairwise scatter plots of all variables. In our case, this will allow us to compare all combinations of variables on the same plot.

Exercise: Say we want to compare Ozone, Solar Radiation, Wind and Temperature variables only. Create a new data frame that consists of only these columns and save as a variable.

Exercise: Use the plot function on your new data frame. It should look something like the following;

[1] More details are available (http://faculty.washington.edu/heagerty/Books/Biostatistics/DATA/ozonedoc.txt



[Optional Extensions]

The `cor` function can be used to calculate correlations between variables. If given a data frame, it will calculate all pairwise correlations.

```
cor(ozone[,1:4],use="c")
```

We can also test the significance of the association between variables using `cor.test`

```
cor.test(ozone[,1],ozone[,4])
```

The function `lm` can be used to fit a linear relationship between two variables. The procedure for fitting is as follows.

```
mod <- lm(ozone[,4] ~ ozone[,1])
mod
```

1.2 Life Expectancy data

Exercise: Data describing the life expectancy of males and females born in particular years are given in the file 'UKLifeExpectancy.tsv' [2]. What function do you think you would use to read these data? Using your chosen function, read these data into RStudio and check the dimensions and first few lines of the data frame.

[2] These data come from a Guardian blog of 8th June 2011 http://www.theguardian.com/news/datablog/2011/jun/08/life-expectancy-uk-data-health

HINT: You should get 243 rows and 4 columns.

Exercise: Plot how the Male life expectancy rate changes over the years (given in the `Age` column)

Exercise: Plot the relationship between Male and Female Life Expectancy as a scatter plot. e.g. Male on the x axis and Female on the y axis.

Exercise: Visualise the Male life expectancy as a barplot. Which plot (scatter or barplot) displays the data in a better way?

```
barplot(life$Male.babies)
```

Exercise: Similarly, the following is a valid plot in R. Comment on whether the trends in the data are better displayed as a barplot or scatter plot. N.B. the `t` function here is transpose and used to reshape the data in the correct dimensions for the barplot.

```
barplot(as.matrix(t(life[,c(2,3)])),beside=TRUE)
```

[Optional Extensions]

Later in the course we will describe how lines, points and other annotations can be added to a plot. One of the functions for doing this is `abline` which can draw a straight line given slope and intercept arguments.

Exercise: Re-plot the relationship between male and female life expectancy and plot a straight line with intercept 0 and gradient 1.

1.3 Survival Data

For this section, you will need to use a couple of functions in the 'crukCIMisc' package.

```
library(crukCIMisc)
```

Exercise: Read the example file `data/Two groups.txt` into R

```
svdata <- read.delim("data/Two groups.txt")
head(svdata)
```

```
##   Days.elapsed Control Treated
## 1           46       1      NA
## 2           46       0      NA
## 3           64       0      NA
## 4           78       1      NA
## 5          124       1      NA
## 6          130       0      NA
```

You should see that the Time variable required for survival analysis can be taken from the first column of the file, and that the Group and Event vectors are encoded in the second and third columns.

Exercise: Use the `extractSurvivalEvent` function to get the Event vector from the second and third

columns of the data matrix. There is only one argument to this function; a data matrix that contains columns that we want to extract event information from.

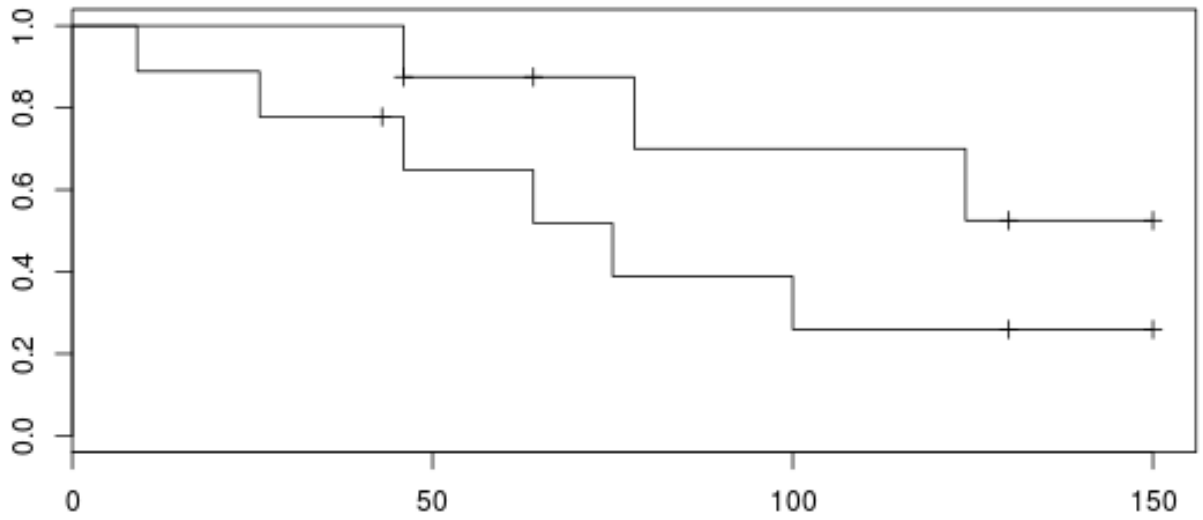
```
Event <- extractSurvivalEvent(svdata[,2:3])
```

Exercise: Use the `extractSurvivalGroups` function to get the Group vector from the data matrix

```
Group <- extractSurvivalGroups(svdata[,2:3])
```

Exercise: Now save the first column of the data matrix as the Time vector and proceed to the survival analysis with the *survival* package

```
library(survival)
Time <- svdata[,1]
SurvData <- Surv(Time, Event)
plot(survfit(SurvData ~ Group))
```



[Optional Extensions]

Exercise: Repeat the steps for the example dataset `data/Three groups.txt`. You will need to decide what column contains the Time data, and which columns include Event and Group information.

2 Plot options

In this part of the practical we will revisit some of the same datasets from the previous section, but introduce different ways in which we can customise and extend the basic plots.

2.1 Life expectancy data

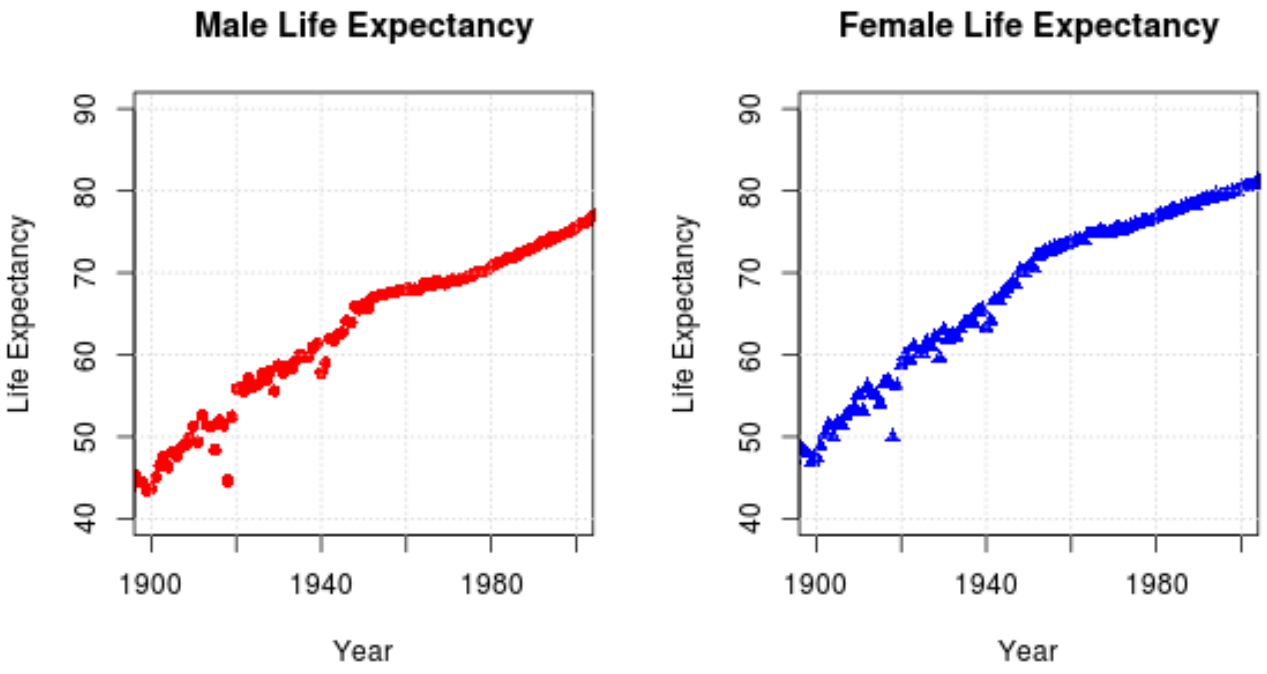
Exercise: Read the UK life expectancy data and plot the female life expectancy on the y axis against year on the x axis. Then overlay the Male life expectancy data using the `points` function. Choose different colours for male and female.

Exercise: Add a legend in the top-left corner using the `legend` function.

Exercise: We decide that we are only interested in the 20th century (i.e. years 1900 to 2000). Create a new plot that only displays years in this range by specifying an appropriate value for the `xlim` argument.

The easiest way to combine several plots on the same page is to use the `par` function. `par` is used to pre-specify many plotting options [3] by a series of named arguments, the most-common of which are the plot layout and margins. The argument to change the layout is `mfrow` which has to be a vector in the form `c(rows, columns)` to form a plot layout with the specified number of *rows* and *columns*. e.g. `par(mfrow = c(rows,columns))`. The order in which plots are created will fill up the page in the required layout configuration.

Exercise: Try and replicate the plot shown below. You will need to use the `par` function to set the layout of the plot to have one row and two columns. Also take care to make sure that you have the same y-axis in both plots.



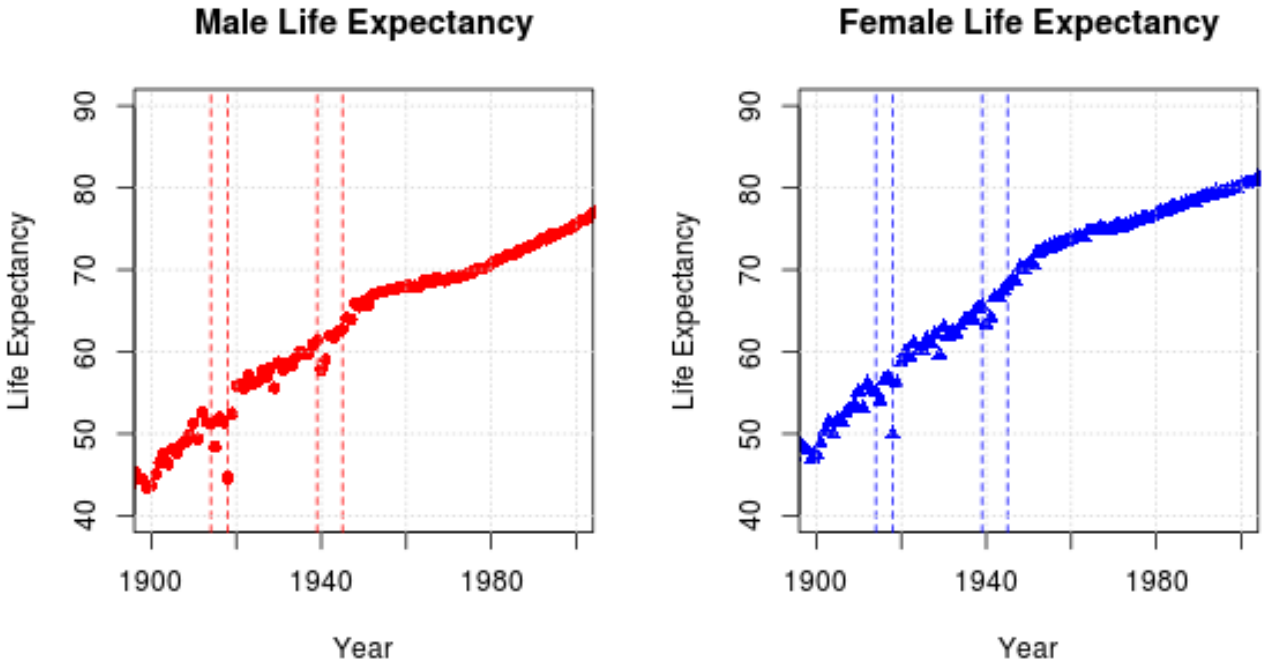
The life expectancy data also contains annotations about years in the 20th century when the two World Wars occurred and would presumably have an influence on the data.

[3] see ?par for details

```
##      Age Male.babies Female.babies Annotations
## 74  1914       51.29         55.13  WW1 starts
## 78  1918       44.64         49.91    WW1 ends
## 99  1939       61.37         65.60  WW2 starts
## 105 1945       62.73         67.95    WW2 ends
```

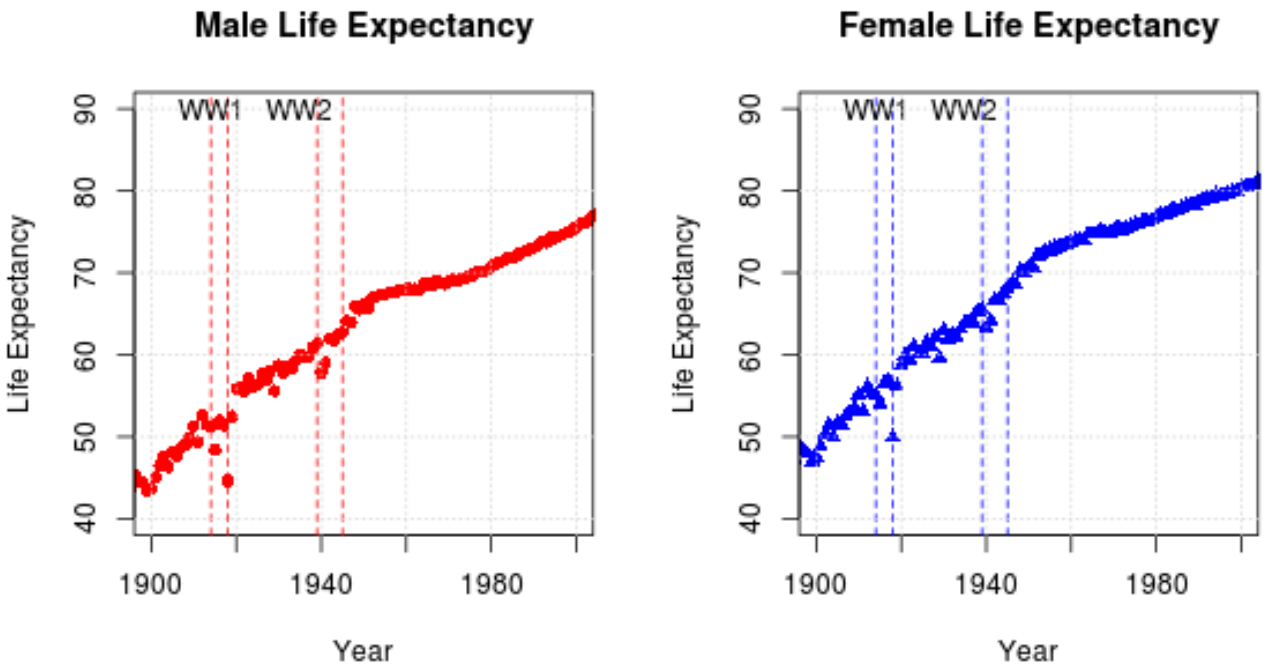
The `abline` can be used to add horizontal and vertical lines to an existing plot. As always, you can find out more information about the function by doing `?abline`. It can take slope and intercept values, plot the result of a linear model fit (outside the scope of today), or plot horizontal and vertical lines by setting the `h` and `v` arguments. The appearance of the line can be modified by the `lty` and `lwd` arguments. For instance, specifying `lty = 2` creates a dotted line.

Exercise: Use `abline` to add vertical lines to indicate periods in which the two world wars took place



The `text` function (`?text`) is another function that can modify an existing plot. In a similar manner to `points`, it has arguments to specify the `x` and `y` coordinates at which text will be written. It also needs a `labels` argument which can be used to specify the text to be written to the plot.

Exercise: Add text annotations to indicate when World War I and World War II started



[Optional Extensions]

Other arguments to the `text` function include the option to rotate the text (`srt`) and use a different font (`font`).

Exercise: See if you can rotate the text by 45 degrees and use a bold font

The `mtext` function allows text to be written in margins around the plot, rather than inside.

Exercise: Use the `mtext` function to write the text labels above the plot.

2.2 Weather data

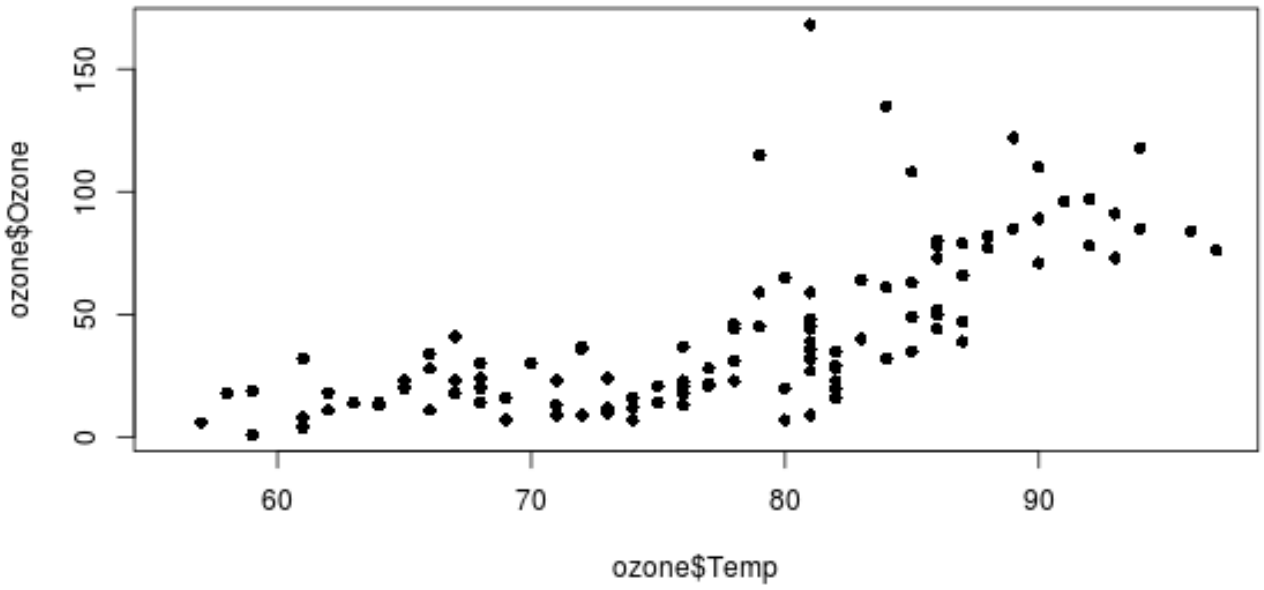
Exercise: Read the Weather data back into R. As seen in the slides, use the formula annotation to produce a boxplot with the temperature on the y axis, and month on the x axis.

Exercise: Repeat the plot with a different colour for each box. First create a vector with five valid colour names (i.e. the name appears in the output of `colours()`) and use this as the col argument to `plot`.

Exercise: Now, load the *RColorBrewer* package and see what palettes are available using `display.brewer.all()`. Use the `brewer.pal` function to create a palette of length 5 from one of the available options and use this palette to colour the boxplot.

We will now re-visit the scatter plot of Ozone level and Temperature

```
plot(ozone$Temp,ozone$Ozone,pch=16)
```



Lets consider the steps required to colour each point according to the month that the observations were made. Our goal is to produce a vector of length ??, where each item in the vector is the colour to be used to plot the corresponding point.

Step 1: Using the table function on the Month variable, you will see how many observations are present for each month

```
table(ozone$Month)
```

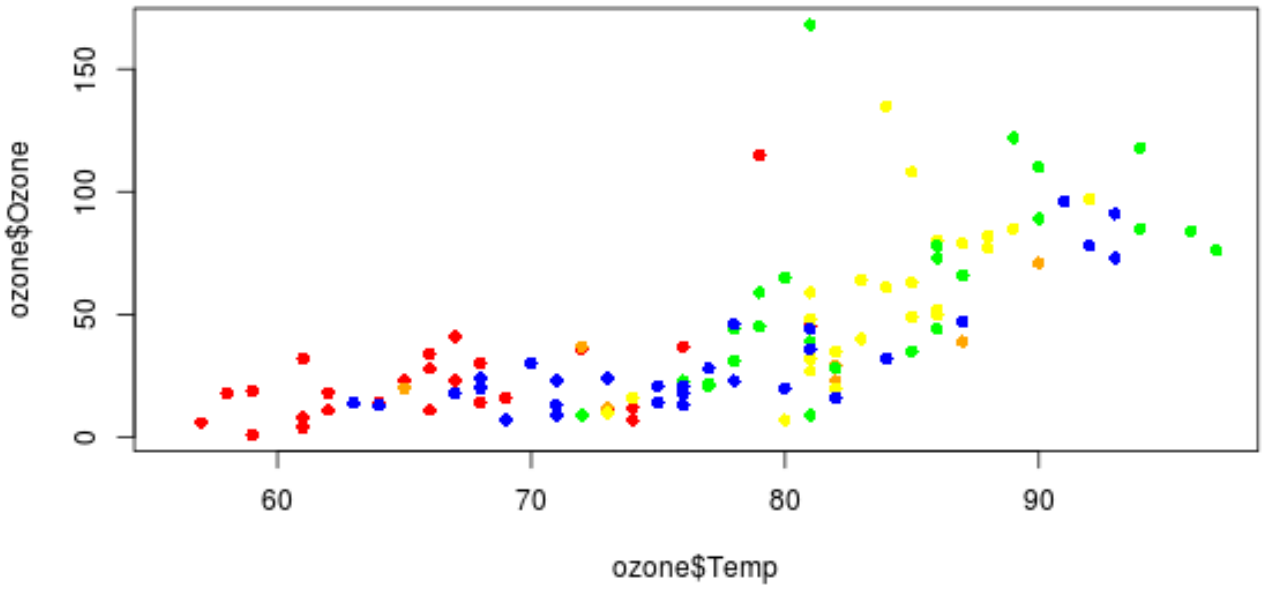
```
##
##  5  6  7  8  9
## 31 30 31 31 30
```

For simplicity, lets say we want to use `"red"`, `"orange"`, `"yellow"`, `"green"` and `"blue"` to represent the colours for each month.

Exercise: Create a vector of the string `"red"` repeated 31 times and assign it to a variable. These will be the colours that data points for the first month will be plotted in. HINT Use the `rep` for this task.

Exercise: Repeat the same exercise to create colour vectors of the appropriate length for all the other months

Exercise: Now combine all your vectors together and use this as the col argument when creating the scatter plot. Check that the length of your combined vector is ??. You should get the following plot;



[Optional Extensions]

Exercise: Create a legend in the top-left corner of the plot

Exercise: Rather than using numbers to represent month, use abbreviated month names stored in the built-in `month.abb` vector

Exercise: Check out the help page for `rep` (`?rep`), and in particular the `times` argument. See if you can construct the vector of colours in a more efficient manner.

3 Data Manipulation - Worked Examples

The following examples to demonstrate data manipulation are a bit more involved and are presented as walkthrough. Please feel free to type the code as it appears, but make sure you understand what is going on at each stage.

3.1 Calculating new variables

Now lets return to the Life Expectancy data. Previously we visualised the life expectancy recorded for each year to look for trends. However, we will now create a new variable that will allow us to split the data for the 20th century by *decade*.

Exercise: Read the Life Expectancy data into R

```
life <- read.delim("data/UKLifeExpectancy.tsv")
```

You should now be familiar with subsetting a data frame using numerical indices and the `[]` notation (i.e `life[1:10,1]` to get the first 10 rows from the first column of the `life` data frame. But what if you didn't know in advance what rows you wanted to extract? We can compare numeric values using `>`, `<`, `==` and return a logical vector. This logical vector can then be used to subset a data frame. In our particular example, we want observations where the `Age` variable is between 1900 and 2001.

Exercise: Create a logical vector that returns `TRUE` or `FALSE` depending on whether `Age` variable in a given row is after 1900

```
vec1 <- life$Age > 1900
```

Exercise: Create a second logical vector that returns `TRUE` or `FALSE` depending on whether `Age` variable in a given row is before 2001. Now use the `&` operator to combine this vector with the vector you calculated in the previous exercise

```
vec2 <- life$Age < 2001

vec1 & vec2
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
##  [15] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
##  [29] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
##  [43] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
##  [57] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
##  [71]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
##  [85]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
##  [99]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
## [113]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
## [127]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
## [141]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRU
## [155]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
## [169] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
## [183] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
## [197] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
## [211] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
## [225] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALS
## [239] FALSE FALSE FALSE FALSE FALSE
```

Exercise: Now subset the life expectancy data using the combined vector. You should get a data frame with 100 rows.

```
centdata <- life[vec1 & vec2,]
dim(centdata)
```

```
## [1] 100   4
```

Note that we didn't have to store the logical vectors in `vec1` and `vec2` and could have done the subsetting in one go; the following is also valid. However, when you're learning R, you might find it useful to bring tasks down into subtasks in this fashion.

```
centdata <- life[life$Age > 1900 & life$Age < 2001,]
dim(centdata)

## [1] 100   4
```

Here we will use the fact that each year is represented by four digits (we will check this first), and the third digit will represent the number of the decade.

Exercise: Create a new variable that captures the decade and use the `table` and `unique` functions to check that it is correct.

```
nchar(centdata$Age)

##   [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [43] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
##  [85] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

decade <- substr(centdata$Age, 3, 3)
table(decade)

## decade
##  0  1  2  3  4  5  6  7  8  9
## 10 10 10 10 10 10 10 10 10 10

unique(decade)

##  [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
```

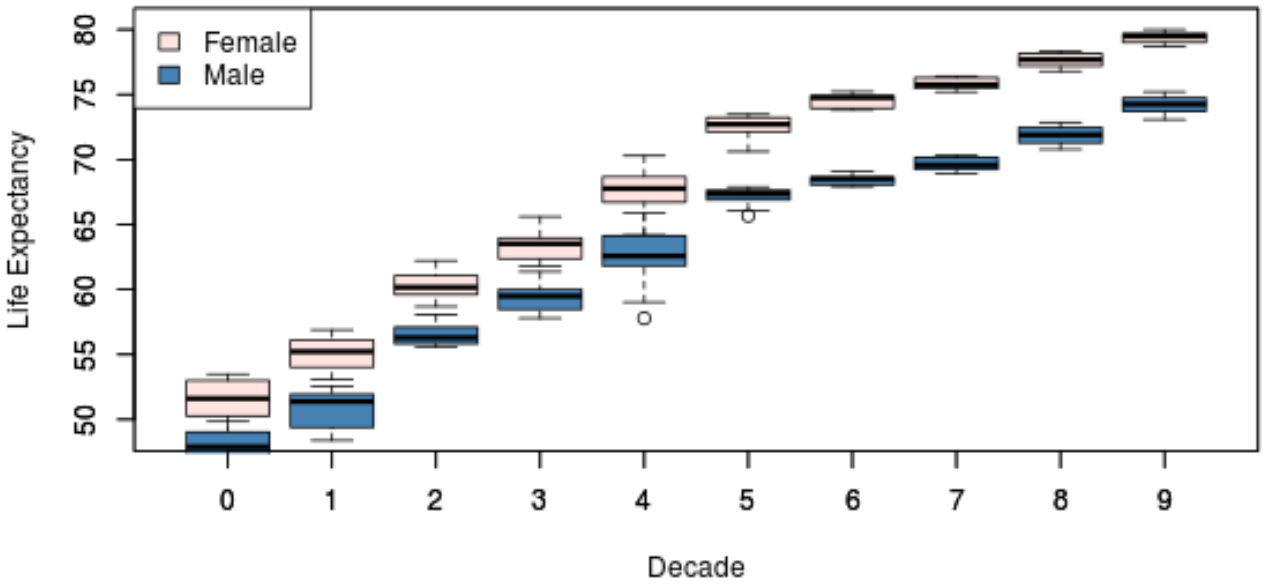
We now have a vector which is the same length as the number of observations in the data frame and can be used to describe different groups in the data. Thus, we can use the $\sim$ notation to create a boxplot with life expectancy on the y-axis, and decade on the x-axis.

Exercise: Create a boxplot that shows the increase in Female life expectancy for each decade

```
boxplot(centdata$Female.babies ~ decade,col="mistyrose")
```

Boxes to show the increase in Male life expectancy over the same period can be added in the following manner.

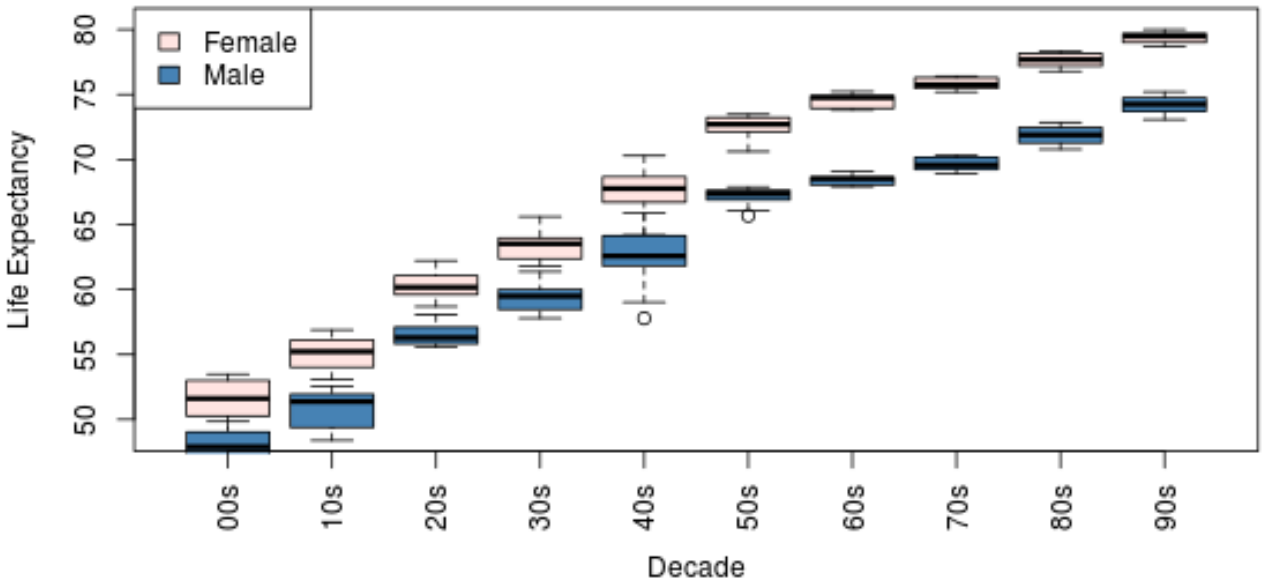
```
boxplot(centdata$Female.babies ~ decade,col="mistyrose",xlab="Decade",
        ylab="Life Expectancy")
boxplot(centdata$Male.babies ~ decade,col="steelblue",add=TRUE)
legend("topleft", fill=c("mistyrose", "steelblue"),legend=c("Female","Male"))
```



[Optional Extensions]

Exercise: Specify the `axes=FALSE` option to the boxplot function to supress the plotting of the axis. Then use the `axis` function to create an axis with labels 00s, 10s, 20s, 30s etc rather than 0,1,2,...

```
boxplot(centdata$Female.babies ~ decade,col="mistyrose",xlab="Decade",
        ylab="Life Expectancy",axes=FALSE)
boxplot(centdata$Male.babies ~ decade,col="steelblue",add=TRUE,axes=FALSE)
declabel <- unique(paste0(decade, "0s"))
axis(side=1, at = 1:10,labels = declabel,las=2)
axis(2)
box()
legend("topleft", fill=c("mistyrose", "steelblue"),legend=c("Female","Male"))
```



3.2 Combining data from multiple files

In this section we will consider the published data from the NKI breast cancer series. This is a series of 295 breast cancer patients that have been used to identify and validate various gene expression signatures. Typically, such datasets are spread over various files which give sample and feature annotation, and the actual gene expression values themselves.

Exercise: Read the files for the dataset in the following manner

```
emat <- read.delim("data/NKI295.exprs.txt",stringsAsFactors=FALSE)
fmat <- read.delim("data/NKI295.fdata.txt")
pmat <- read.delim("data/NKI295.pdata.txt")
```

Exercise: Check the dimensions of the expression matrix and sample annotation. What do each row and column in the two matrices correspond to?

```
dim(emat)
```

```
## [1] 24481   296
```

```
dim(pmat)
```

```
## [1] 295  21
```

Exercise: Use the == operator to verify that each column in the expression matrix (excluding the first column) corresponds to each row in the sample information matrix

```
colnames(emat)[-1] == pmat$sampleNames
```

Exercise: Similarly, verify that each row in the feature information corresponds to the first column of the expression matrix. You may wish to use the `all` to check that you get TRUE for every position

```
all(emat[,1] == fmat[,1])
```

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

3.3 Subsetting the patients

A common task is to create a subset of the data relating to patients that satisfy various clinical parameters. To achieve this, we rely on the fact that the sample identifiers are the column names for the expression matrix and also appear as a column in the sample information matrix.

Exercise: Find the sample names of the ER negative patients. You will first need to identify which column in the phenotypic data holds the ER status of each patient.

```
which(pmat$ER == "Negative")
```

```
##  [1]   3   4   7  13  19  22  28  31  32  33  35  38  50  51  55  56  61  63  66  68  71
## [22]  76  82  90  92  94  96  99 101 110 112 117 124 127 128 133 135 137 139 143 144 145
## [43] 148 151 153 160 167 170 171 172 178 206 209 222 224 228 230 231 233 236 239 242 248
## [64] 262 267 273 289 291 293
```

```
erNegSamples <- pmat$sampleNames[which(pmat$ER == "Negative")]
```

Exercise: Now match these sample names to the columns in the expression matrix to get a vector of column indices. Since each sample should be unique, it is Ok to use the `match` function. Use the column indices to create a subset of the expression matrix that describes only ER negative samples.

```
erNegMatix <- emat[,match(erNegSamples, colnames(emat))]
```

If we wish, we can write this data matrix to a file.

Exercise: Export the data for the ER negative patients as a comma-separated file

```
write.csv(erNegMatix, file="erNegativeExpression.csv")
```

3.4 Retrieving the data for a particular gene

Various functions can be used to see if a defined string (or set of strings) appears in a larger vector.

Exercise: Find the entry in the gene annotation matrix that corresponds to the gene symbol "ESR1". How would you do this using `grep`, `match`, `==`?

```
grep("ESR1", fmat$symbol)
```

```
## [1] 18889
```

```
match("ESR1", fmat$symbol)
```

```
## [1] 18889
```

```
which(fmat$symbol =="ESR1")
```

```
## [1] 18889
```

```
which(fmat$symbol %in% "ESR1")
```

```
## [1] 18889
```

Exercise: What is the probe ID for the ESR1 gene?

Exercise: Match the probe ID that you just found to relevant row in the expression matrix. Remember that probe IDs are in the first column of the expression matrix.

```
match("18904", emat[,1])
```

```
## [1] 18889
```

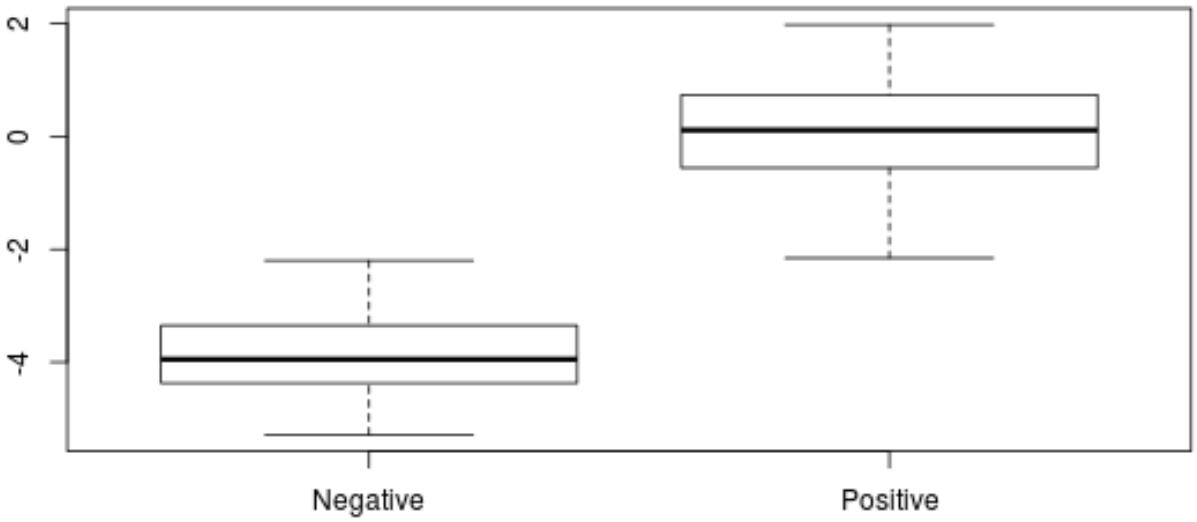
```
erVals <- emat[18889,-1]
```

3.5 Associating gene expression with clinical variables

We can now associate the expression of a particular gene with various clinical factors.

Exercise: Make a boxplot of ESR1 expression against ER status

```
boxplot(as.numeric(erVals) ~ pmat$ER)
```



We can test the signficance of the association using the `t.test` function.

```
t.test(as.numeric(erVals) ~ pmat$ER)
```

```
##
##  Welch Two Sample t-test
##
## data:  as.numeric(erVals) by pmat$ER
## t = -36.6006, df = 146.102, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.124197 -3.701624
## sample estimates:
## mean in group Negative mean in group Positive
##            -3.86215423             0.05075594
```

[Optional Extensions]

The pam50 gene signature has been widely-used to classify breast cancer into different subtypes. Here we show to retrieve the gene expression values relating to these genes [4].

```
pam50Genes <- read.csv("data/pam50Genes.csv")
head(pam50Genes)
```

```
##    probe probe.centroids EntrezGene.ID
## 1 ACTR3B          ACTR3B         57180
```

[4] this table was extracted from the *genefu* Bioconductor package

```
## 2   ANLN            ANLN         54443
## 3   BAG1            BAG1           573
## 4   BCL2            BCL2           596
## 5  BIRC5           BIRC5           332
## 6  BLVRA           BLVRA           644
```

First we match the gene symbols from our gene list to the gene annotation table for our dataset

```
match(pam50Genes[,1], fmat$symbol)
```

```
##  [1]  6202 22332  5212 22930 21356 23482 14961 16562 22112 22111    NA  1514 15355 18113
## [15] 16268  7054  6409 18889  5656 21705  6600 23531 10457  7915 14823    NA 22114 21288
## [29] 21293 19100   649 14097 12212  1146 16716 13269  1339    NA 23010 10228   615  1679
## [43]  4144 17607   285  5044   509 20563 10189  8950
```

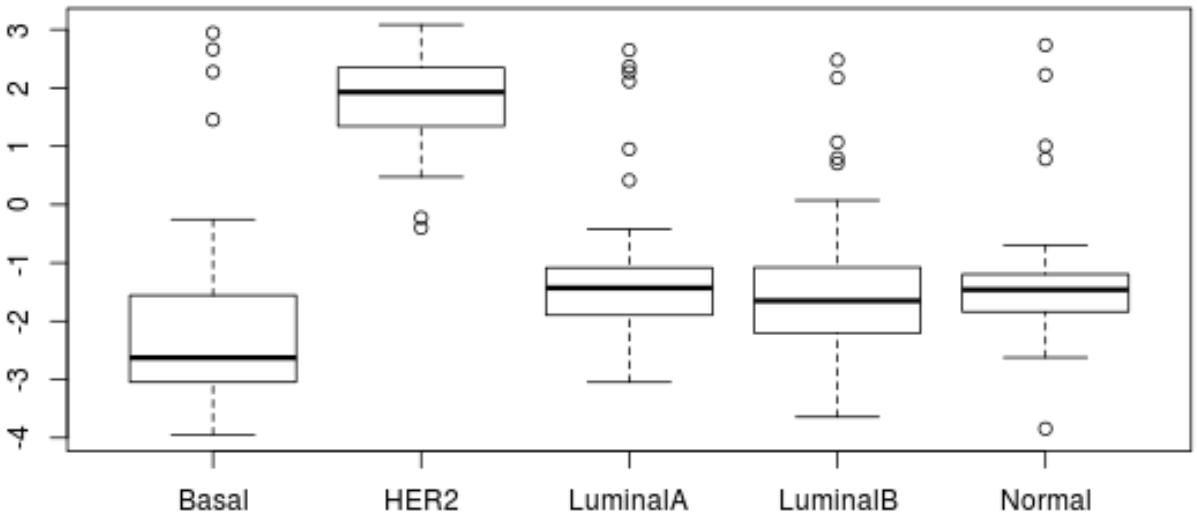
However, we notice that some genes are not found in our data. We can use the `na.omit` function to exclude these NA values from the index vector.

```
pam50Anno <- fmat[na.omit(match(pam50Genes[,1], fmat$symbol)),]
pam50Probes <- fmat[na.omit(match(pam50Genes[,1], fmat$symbol)),1]
```

We can then just match up the probes to the first column of the expression matrix and subset. For completeness, we can assign the rownames based on the gene name

```
pam50Data <- emat[match(pam50Probes,emat[,1]),-1]
rownames(pam50Data) <- pam50Anno$symbol
```

```
boxplot(as.numeric(pam50Data["ERBB2",]) ~ pmat$Fan.nearest.centroid)
```


3
2
1
0
-1
-2
-3
-4
Basal
HER2
LuminalA
LuminalB
Normal