R graphics and data manipulation

Mark Dunning, Mike Smith, Sarah Vowler

12 December 2014

About this course

- Common types of plot; What makes a good plot? (Sarah)
- Creating basic plots in R (Mark)
- Practical
- Customising a plot (Mark)
- Tea / Coffee / Biscuits
- Practical
- Manipulating data (Mark)
- Practical

What the course is not

- Introduction to R from scratch
- Advanced / specialised graphics
- ‘Programming’ in R

Other resources

- A course manual
- An R package; crukCIMisc
- An online support forum; bioinf-qa001/

Plotting Basics

Mark Dunning

12/12/2014

Introducing the plot function

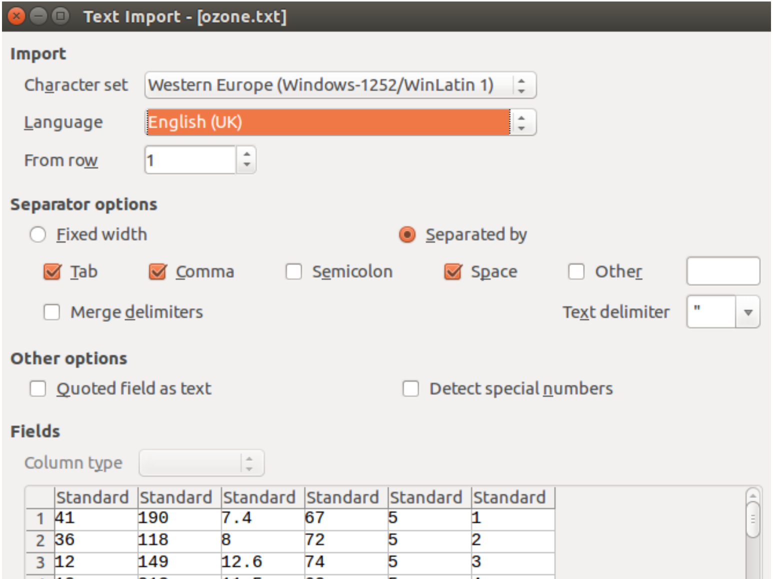
Introducing the dataset

Data describing weather conditions in New York City in 1973 were obtained from the supplementary data to *Biostatistics: A Methodology for the Health Sciences*

You can load these data into Excel for reference

Reading the data

Like other programs, we need to specify some details about the file when we read it in



Text Import - [ozone.txt]

Import

Character set: Western Europe (Windows-1252/WinLatin 1)

Language: English (UK)

From row: 1

Separator options

○ Fixed width ● Separated by

☑ Tab ☑ Comma □ Semicolon ☑ Space □ Other

□ Merge delimiters

Text delimiter: "

Other options

□ Quoted field as text □ Detect special numbers

Fields

Column type

	Standard	Standard	Standard	Standard	Standard	Standard
1	41	190	7.4	67	5	1
2	36	118	8	72	5	2
3	12	149	12.6	74	5	3

Reading the data

- File location
- Column *separator*
- Use column headings in file?
- Skip any rows?

Word of caution

Leo Tolstoy:

> *Happy families are all alike; every unhappy family is unhappy in its own way.*

Hadley Wickham:

> *Like families, tidy datasets are all alike but every messy dataset is messy in its own way*

`http://vimeo.com/33727555`

Reading the data

To import these data into R we use the `read.csv` function, which will create a *data-frame* representation

- Many examples of reading data given in the course manual

```
data <- read.csv("data/ozone.csv")
```

Reading the data

If we don't know where the file is located, we can use the `file.choose` function

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

Exploring the data

You should **always check** that the data have been imported correctly by previewing and checking the *dimensions*.

```
head(data)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

```
dim(data)
```

```
## [1] 153   6
```

Exploring the data

```r
summary(data)
```

```
##      Ozone          Solar.R         Wind            Temp
##  Min.   :  1.0   Min.   :  7   Min.   : 1.70   Min.   :5
##  1st Qu.: 18.0   1st Qu.:116   1st Qu.: 7.40   1st Qu.:7
##  Median : 31.5   Median :205   Median : 9.70   Median :7
##  Mean   : 42.1   Mean   :186   Mean   : 9.96   Mean   :7
##  3rd Qu.: 63.2   3rd Qu.:259   3rd Qu.:11.50   3rd Qu.:8
##  Max.   :168.0   Max.   :334   Max.   :20.70   Max.   :9
##  NA's   :37      NA's   :7
##      Month          Day
##  Min.   :5.00   Min.   : 1.0
##  1st Qu.:6.00   1st Qu.: 8.0
##  Median :7.00   Median :16.0
##  Mean   :6.99   Mean   :15.8
##  3rd Qu.:8.00   3rd Qu.:23.0
##  Max.   :9.00   Max.   :31.0
##
```

Data representation

The data are stored in a data frame. These are subset using square brackets `[]`

e.g. print rows 1 to 10 from the first column

```
data[1:10,1]
```

```
##  [1] 41 36 12 18 NA 28 23 19  8 NA
```

Data representation

We can get entire columns and rows by *omitting* the row or column index. The result is a `vector`

```
data[1,]
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
```

```
data[,1]
```

```
##   [1]  41  36  12  18  NA  28  23  19   8  NA   7  16  1
##  [18]   6  30  11   1  11   4  32  NA  NA  NA  23  45 11
##  [35]  NA  NA  NA  29  NA  71  39  NA  NA  23  NA  NA  2
##  [52]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA 135  49  3
##  [69]  97  97  85  NA  10  27  NA   7  48  35  61  79  6
##  [86] 108  20  52  82  50  64  59  39   9  16  78  35  6
## [103]  NA  44  28  65  NA  22  59  23  31  44  21   9  N
## [120]  76 118  84  85  96  78  73  91  47  32  20  23  2
## [137]   9  13  46  18  13  24  16  13  23  36   7  14
```

Data representation

The data frame is *not altered*

```
dim(data)
```

```
## [1] 153   6
```

```
data[1,]
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
```

```
dim(data)
```

```
## [1] 153   6
```

About NA

- You may have noticed some `NA` entries in the vector
- This is R's way of denoting *missing values*
- They can cause problems when we try and calculate averages. Most functions have an `na.rm` option
- Can also use `na.omit`

Thinking about the data

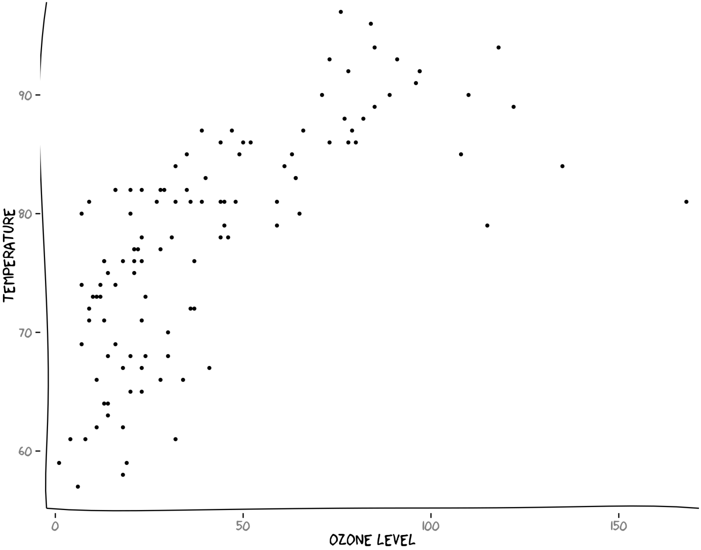
What variables do we have?

- Ozone, Wind, Temp (Continuous)
- Month, Day (Discrete)

What are we interested in?

- Trend
- Relationship

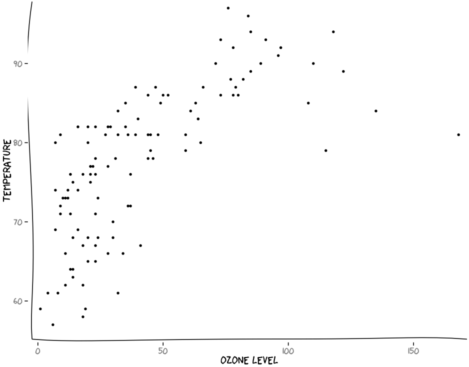
Thinking about the plot



Thinking about the plot

A figure consists of

- Data points; each defined by an x and y coordinate
- Axes; defining the range of the data and a label
- Title



Assignment to a variable

- We can extract named columns from a data frame using the $ operator
- The result is a `vector`

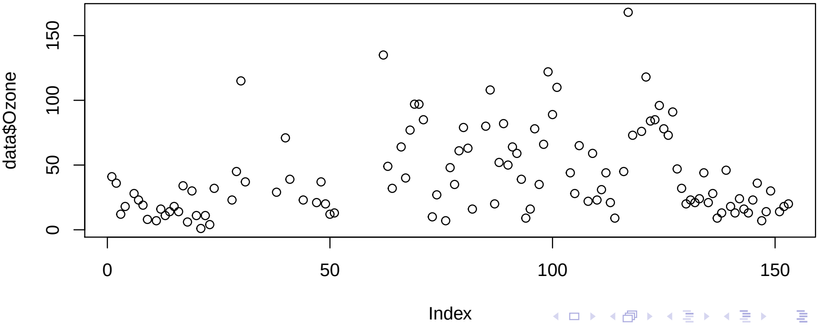
```
ozone <- data$Ozone
```

Scatter plots

Suppose we want to look at the change in Ozone level (continuous)

- `plot` is the general-purpose plotting function in R
- Given a *vector* it will plot the values in the vector on the **y** axis, and index on the **x** axis
- It will create axes and labels automatically

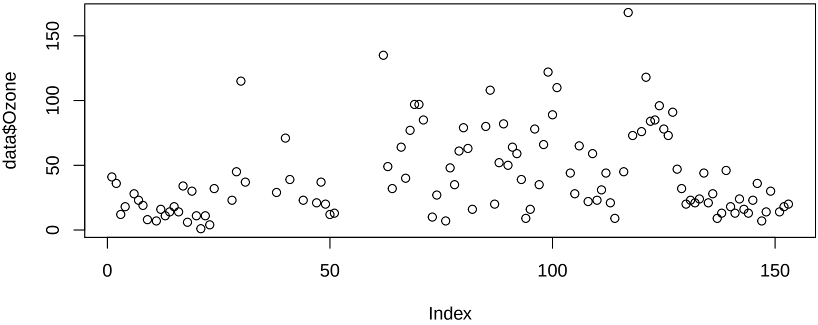
```
plot(data$Ozone)
```



Scatter plots

- We have 153 points on the plot
- Axis labels, points, title and colours can be altered (see later)

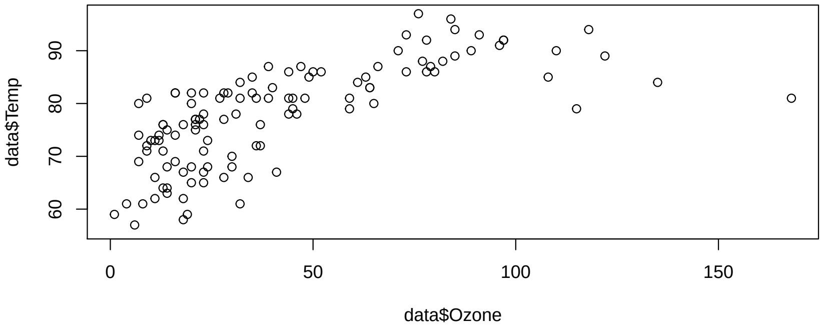
```
plot(data$Ozone)
```



Data visualisation

- Can plot one vector against another
- First *argument* is plotted on the x axis, second *argument* on the y axis

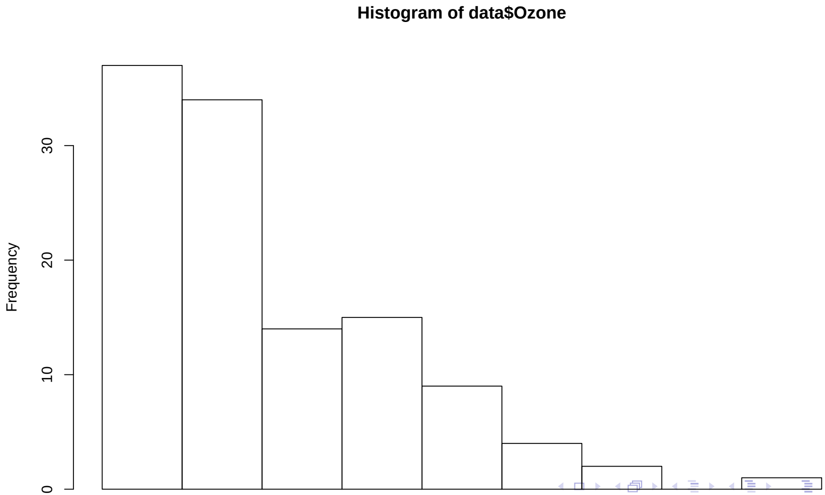
```
plot(data$Ozone,data$Temp)
```



Other ways of visualising a vector

If we were interested in the *distribution* of the data, we could use a histogram

```r
hist(data$Ozone)
```



Visualising Distributions

The dataset

We have made some observations of cell in different conditions

- Three different groups (categories) in the dataset
- Repeated measurements for each group
- Are the data distributed differently in the different groups?

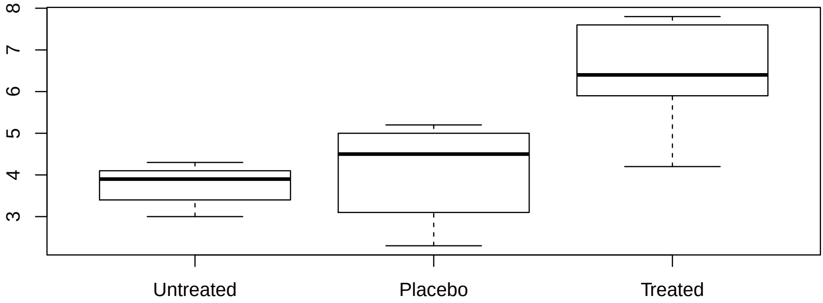
```r
data <- read.delim("data/plasma.txt")
data
```

```
##   Untreated Placebo Treated
## 1       3.4     2.3     4.2
## 2       4.3     5.2     7.8
## 3       3.0     4.5     5.9
## 4       3.9     3.1     6.4
## 5       4.1     5.0     7.6
```

The boxplot

If given a data frame, `boxplot` will summarize each column separately and construct the box from the quantiles. Again, the axes and labels are automatically decided

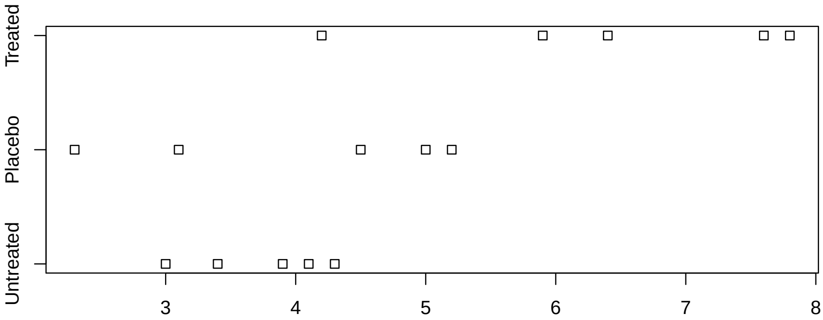
```
boxplot(data)
```



Plotting individual points

The stripchart or dotchart functions can be used to visualise individual points

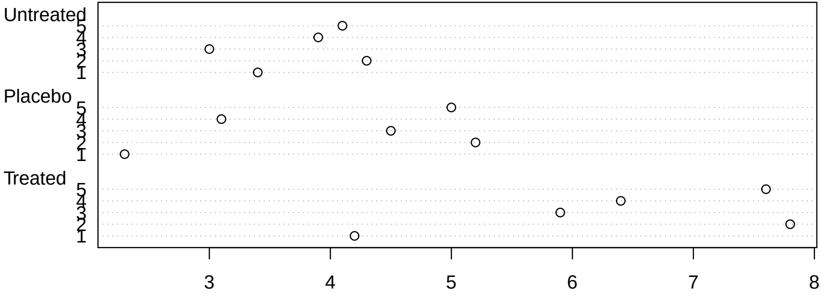
```
stripchart(data)
```



Plotting individual points

The stripchart or dotchart functions can be used to visualise individual points

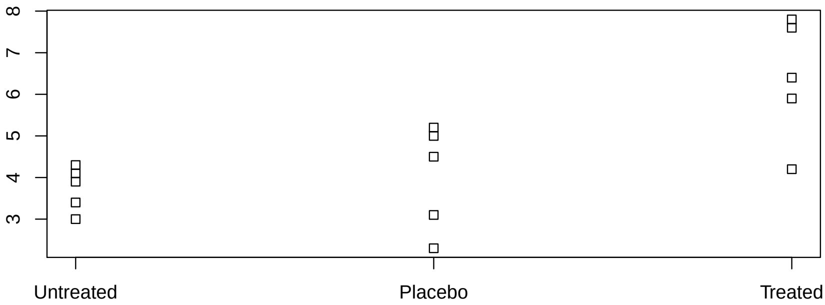
```
dotchart(as.matrix(data))
```



Plotting individual points

- `vertical = TRUE` ensures the plot is in the same orientation as the boxplot

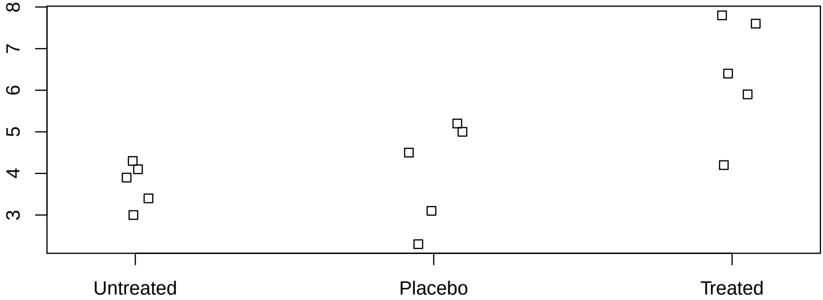
```
stripchart(data,vertical=TRUE)
```



Plotting individual points

- We can *stack* or *jitter* points if required

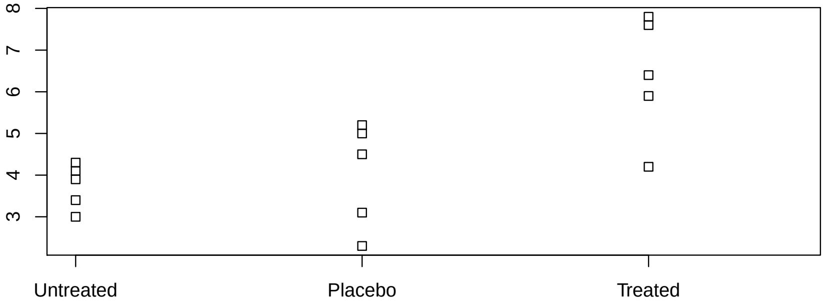
```
stripchart(data,vertical=TRUE,method="jitter")
```



Plotting individual points

- We can *stack* or *jitter* points if required

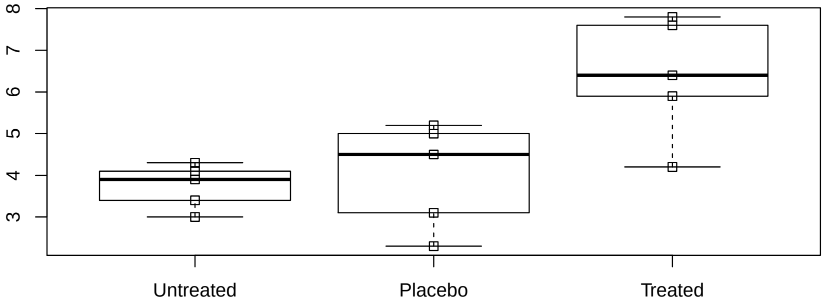
```
stripchart(data,vertical=TRUE,method="stack")
```



Overlaying points

- add=TRUE argument overlays the stripchart on an existing plot

```
boxplot(data)
stripchart(data,vertical=TRUE,add=TRUE)
```



Summarising the data

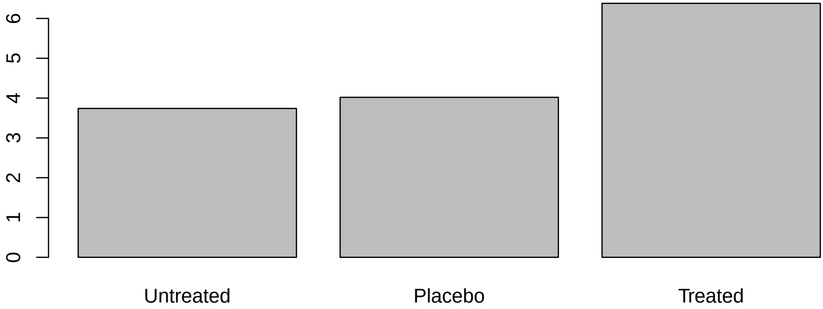
```
summary(data)
```

```
##    Untreated       Placebo        Treated    
##  Min.   :3.00   Min.   :2.30   Min.   :4.20  
##  1st Qu.:3.40   1st Qu.:3.10   1st Qu.:5.90  
##  Median :3.90   Median :4.50   Median :6.40  
##  Mean   :3.74   Mean   :4.02   Mean   :6.38  
##  3rd Qu.:4.10   3rd Qu.:5.00   3rd Qu.:7.60  
##  Max.   :4.30   Max.   :5.20   Max.   :7.80  
```

Bar plots

To display the data as a barplot, we need to compute the mean of each *column*. The `colMeans` function is convenient for this.

```
barplot(colMeans(data))
```



N.B. see also `rowMeans`, `colSums`, `rowSums`

Calculating error bars

To add *error bars* we need to calculate the standard deviations

```
sd(data$Untreated)
```

```
## [1] 0.532
```

```
sd(data$Placebo)
```

```
## [1] 1.264
```

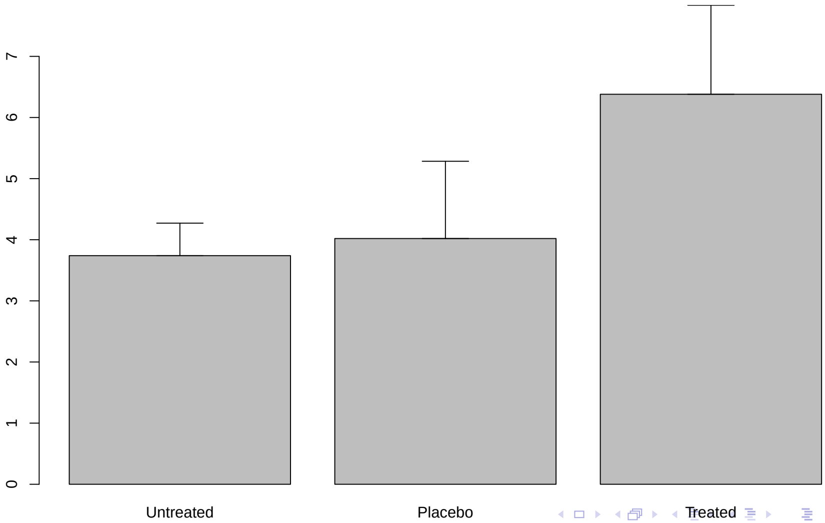
```
sd(data$Treated)
```

```
## [1] 1.457
```

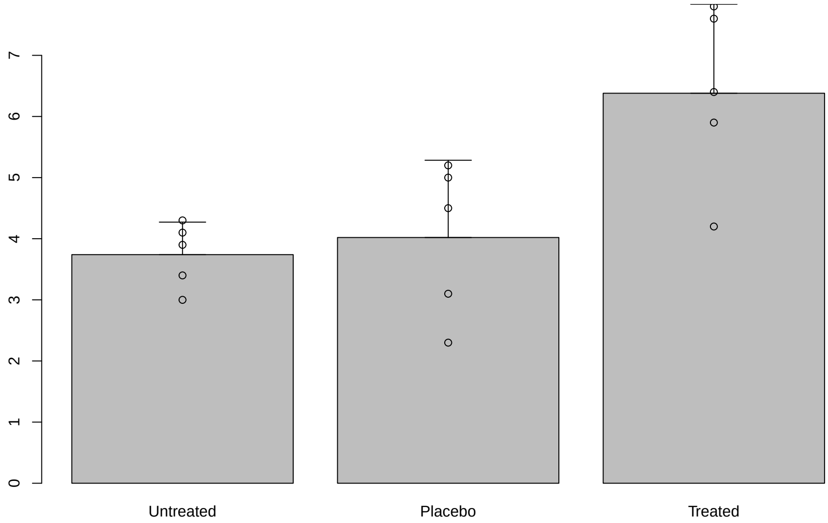
Adding error bars

Possible, but recall earlier discussion

```
dpPlot(data)
```



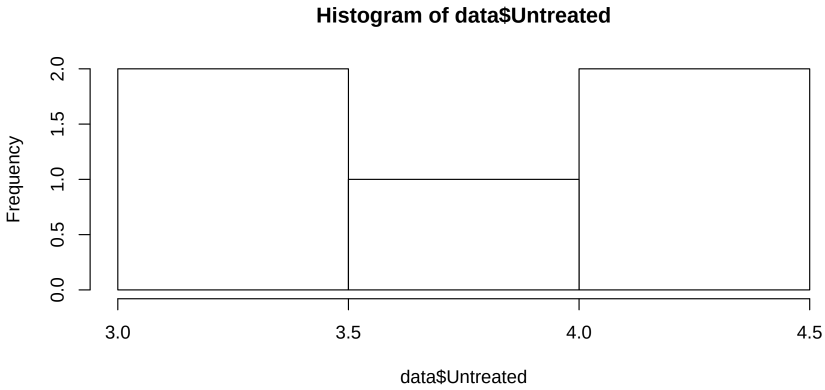
We can still overlay points



Histograms

You can also make a histogram - (not very useful in this case)

```
hist(data$Untreated)
```



About data formats

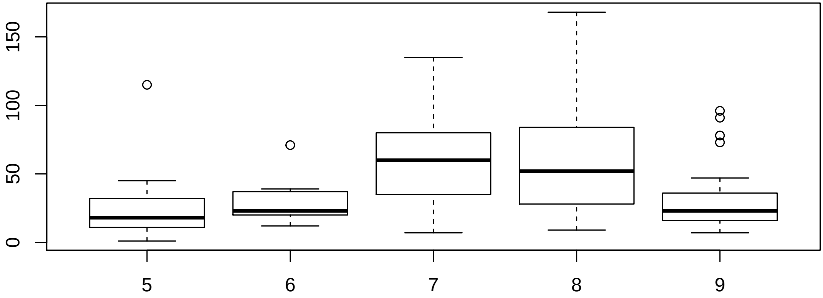
- We produce boxplots from data in this format
- Each group of interest is in a different column

```
data <- read.delim("data/plasma.txt")
data
```

```
##   Untreated Placebo Treated
## 1       3.4     2.3     4.2
## 2       4.3     5.2     7.8
## 3       3.0     4.5     5.9
## 4       3.9     3.1     6.4
## 5       4.1     5.0     7.6
```

About data formats

- Given what we know so far, what format should the data for this plot be in?



```
##   Month1 Month2 Month3
## 1     41     NA    135
## 2     36     NA     49
## 3     12     NA     32
```

A note about ‘long data’

- Recall our weather data
- We do not have separate columns for each month
- Ozone observations are *stacked* on top of each other
- There is an *indicator* variable to tell us the month
- This is know as ‘long data’

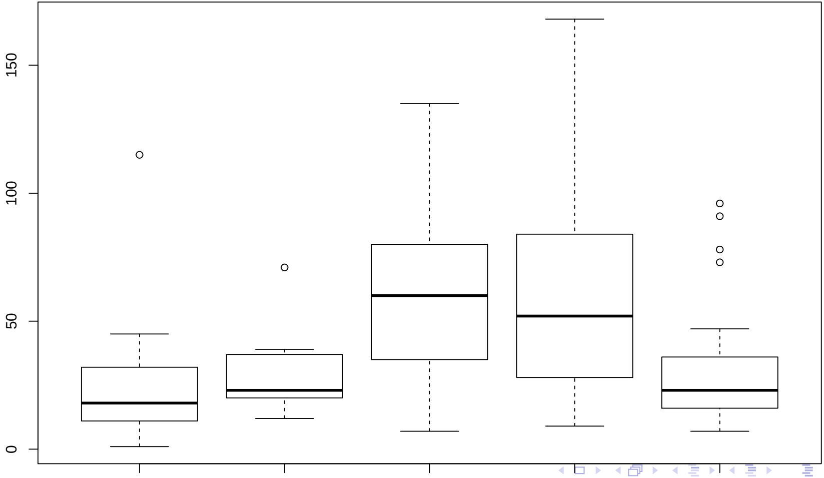
```r
data <- read.csv("data/ozone.csv")
head(data)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Boxplot of long data

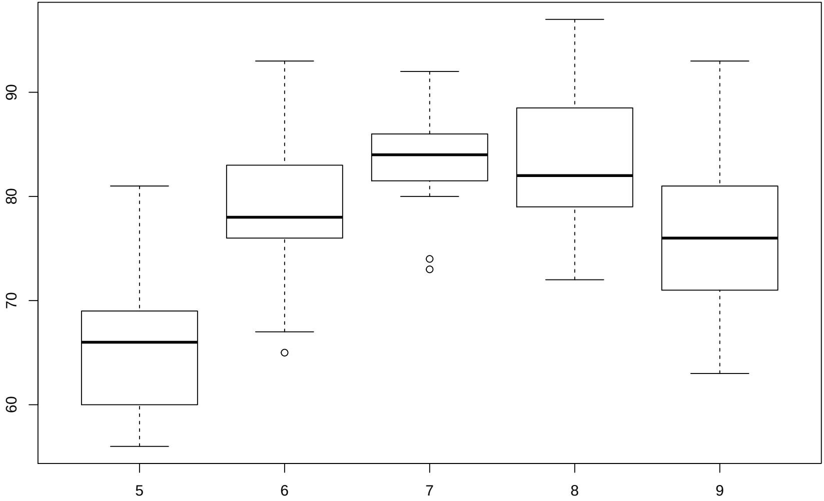
- Month is a variable in the data frame
- We use *formula* syntax with the ~ symbol. e.g. `y ~ x`

```
boxplot(data$Ozone~data$Month)
```



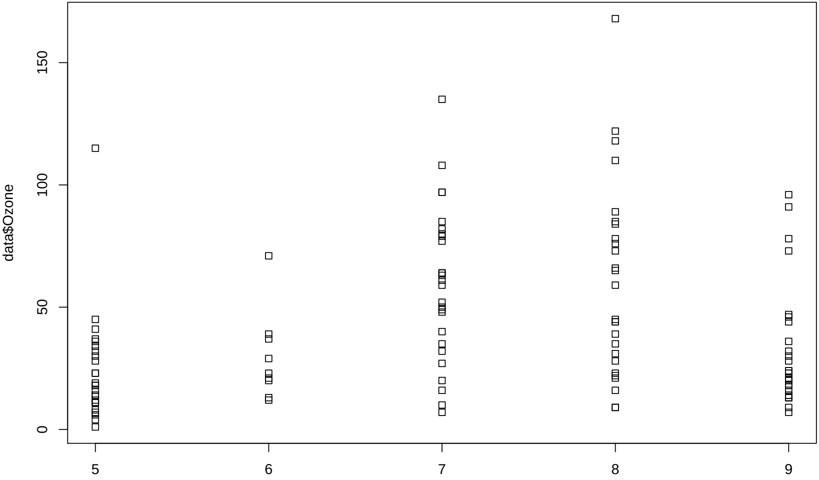
Boxplot of long data

```
boxplot(data$Temp~data$Month)
```



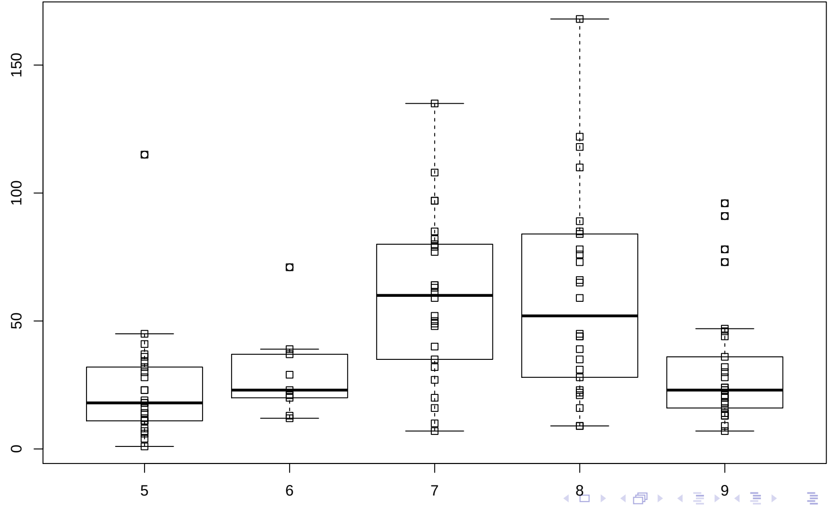
stripchart of long data

```
stripchart(data$Ozone~data$Month,vertical=TRUE)
```



Boxplot of long data

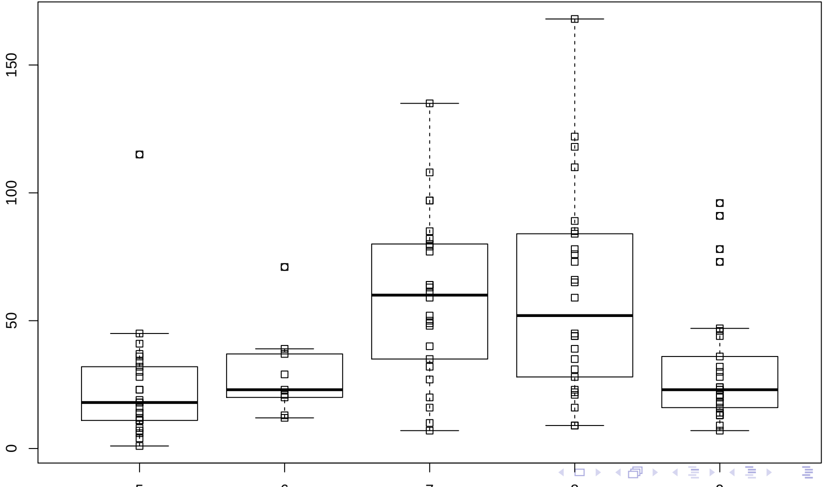
```
boxplot(data$Ozone~data$Month)
stripchart(data$Ozone~data$Month,vertical=TRUE,add=TRUE)
```



Boxplot of long data

- This is equivalent and a bit more concise

```
boxplot(Ozone~Month,data)
stripchart(Ozone~Month,data,vertical=TRUE,add=TRUE)
```



Count data

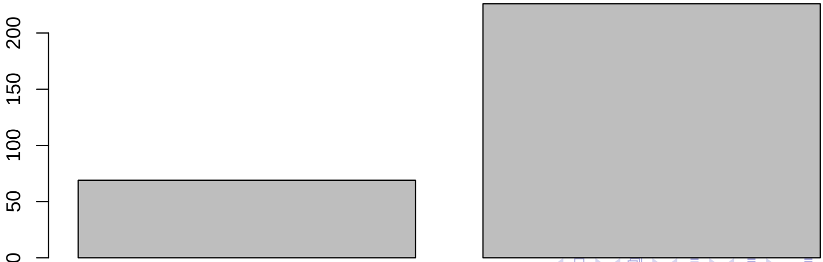
Making a barplot

- Often we have to make a table before constructing a bar plot

```
clinical <- read.delim("data/NKI295.pdata.txt")
table(clinical$ER)
```

```
##
## Negative Positive
##       69      226
```

```
barplot(table(clinical$ER))
```

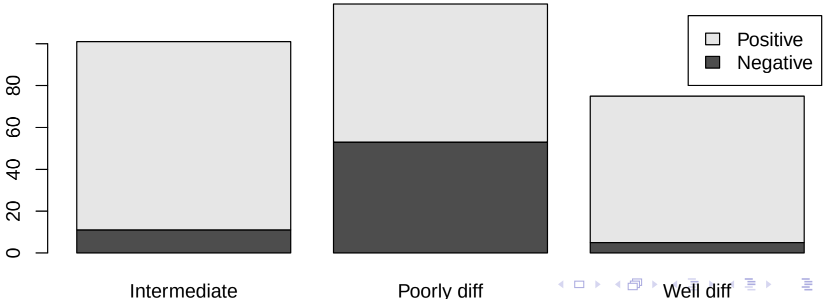


Stacking

```
counts <- table(clinical$ER,clinical$grade)
counts
```

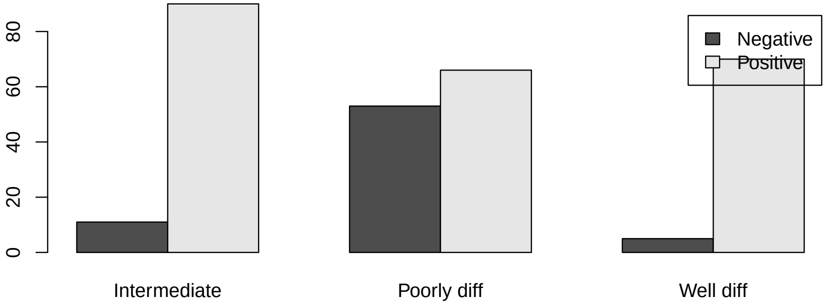
```
##
##              Intermediate Poorly diff Well diff
##   Negative             11          53         5
##   Positive             90          66        70
```

```
barplot(counts, legend = rownames(counts))
```



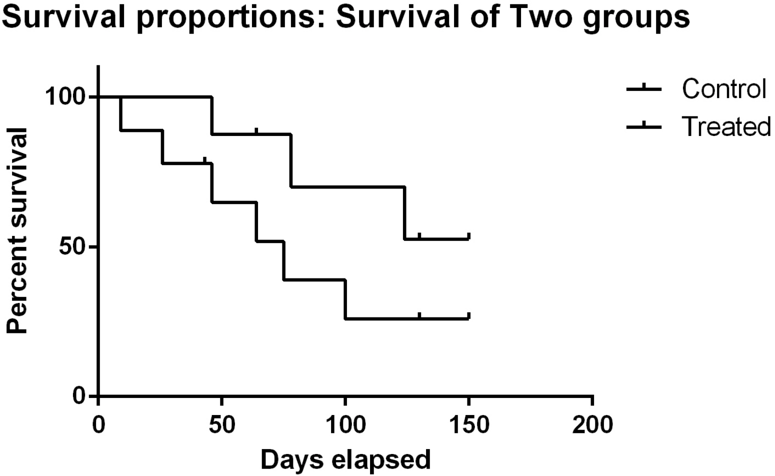
Grouping

```
counts <- table(clinical$ER,clinical$grade)
barplot(counts,beside=TRUE,legend=rownames(counts))
```



Curves

Survival curves



Survival curves

To perform a survival analysis we need the following pieces of information

- Time to Event
- Event (e.g. dead or alive)
- Group

Example data

```r
clinical <- read.delim("data/NKI295.pdata.txt")

Event <- clinical$event_death
Time <- clinical$survival.death.
Group <- clinical$ER
```

The survival package

```
library(survival)
```

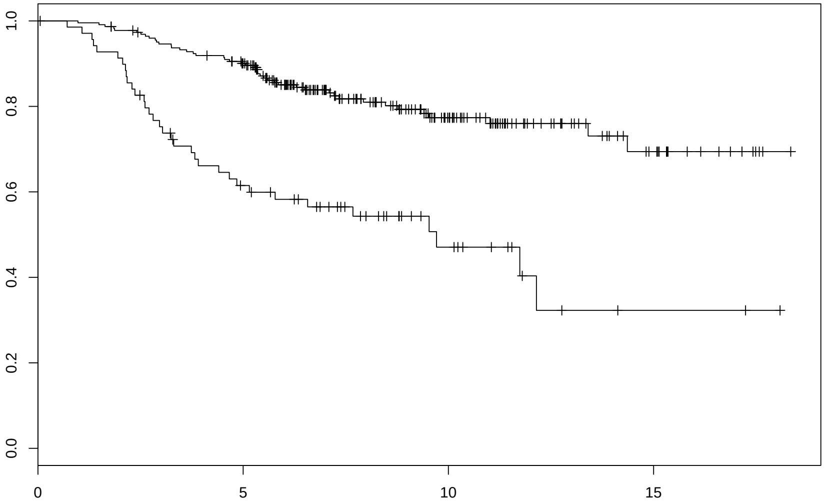
```
## Loading required package: splines
```

```
survData <- Surv(Time, Event)
survData[1:10]
```

```
##   [1] 12.997+ 11.157+ 10.138+  8.802+ 10.294+  5.804+  7.
##   [9]  8.233+  7.866+
```

Making the Survival curve

```
plot(survfit(survData ~ Group))
```



Survival data in Prism

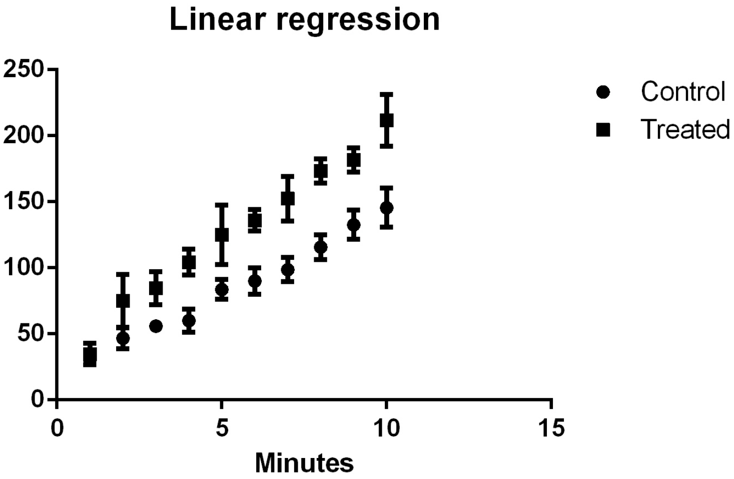
- Prism uses a special format to represent survival data
- See practical for details

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

```
##   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
```

Growth Curve

Goal is to produce following



Growth Curve

```r
data <- read.delim("PrimerExamples/Linear regression.txt")
head(data)
```

```
##   Minutes Control Control.1 Control.2 Treated Treated.1
## 1       1      34        29        28      31        29
## 2       2      38        49        53      61        NA
## 3       3      57        NA        55      78        99
## 4       4      65        65        50      93       111
## 5       5      76        91        84      NA       109
## 6       6      79        93        98     134       145
```

Procedure

- Gather columns together according to *group*
- Calculate avearge values for each time point
- Calculate a variability measurement (e.g. standard deviation)
- Plot averages with error bars
- Smooth curve through the points

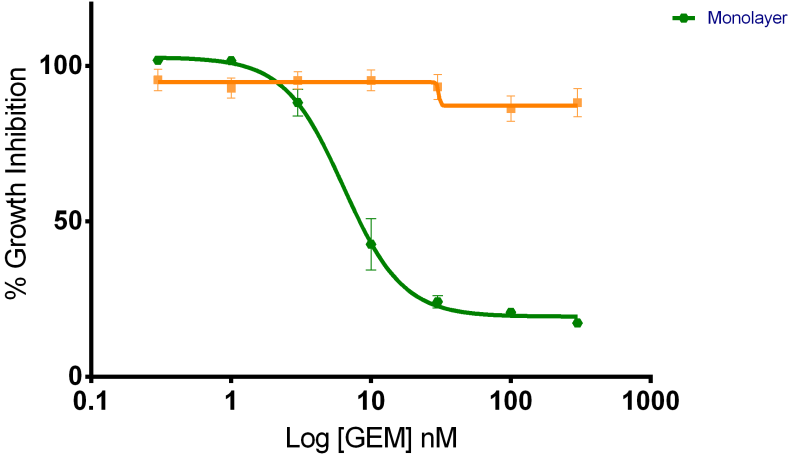
Shortcut

- We have implemented this in the `crukCIMisc` package that accompanies this course - `prismTimeSeries`
- See practical for example

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

Dose response

Goal is to produce following



Another shortcut

- Data are *similar* format as previous example
- see `prismDoseResponse` in `crukCIMisc`
- See package `drc` for more in-depth analysis
- `install.packages(drc)`

Break for practical

Customising a Plot

Mark Dunning

12/12/2014

Changing how a plot is created

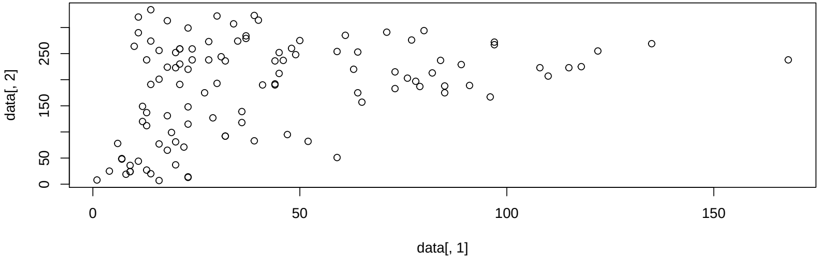
Specifying extra arguments to plot

- The `plot` function creates a very basic plot
- Many optional arguments can be specified See `?plot`
- Other plots e.g. `boxplot`, `hist`, `barplot` are special instances of `plot` so can accept the same arguments

Lets re-visit the ozone dataset

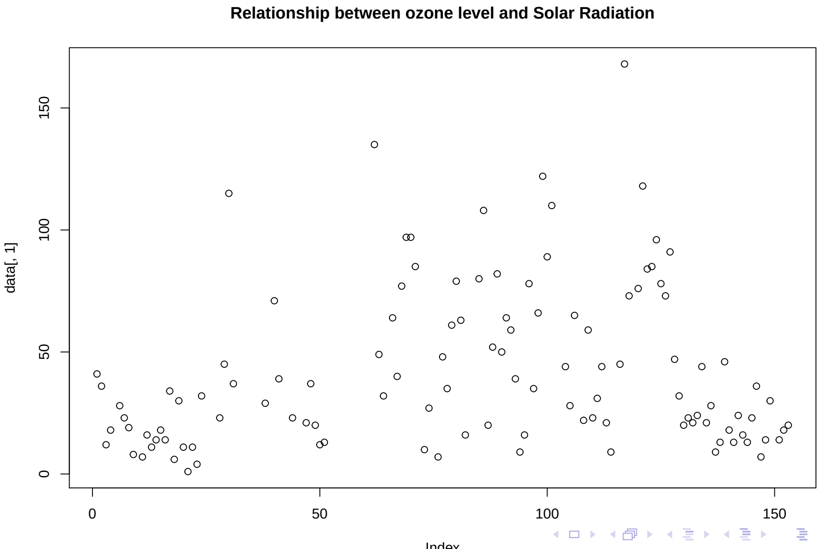
The default plots are ugly; No title, un-helpful labels, No colour

```
data <- read.csv("data/ozone.csv")
plot(data[,1],data[,2])
```



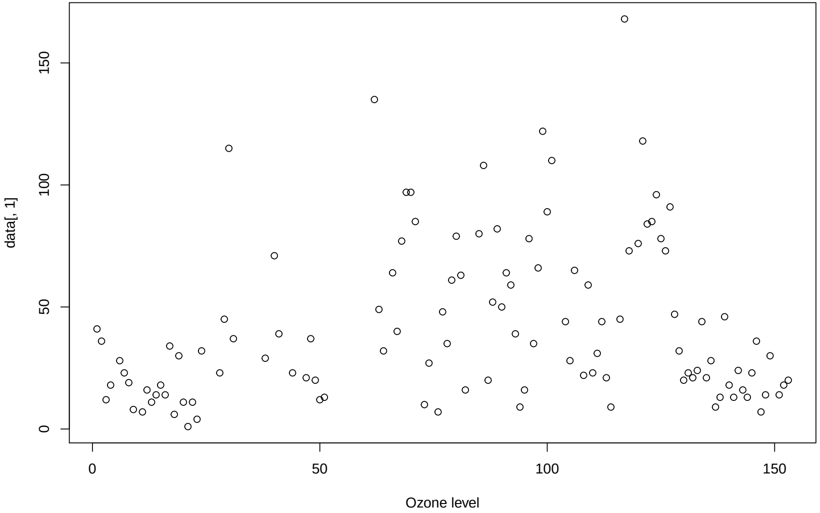
Adding a title

```
plot(data[,1],
     main="Relationship between ozone level and Solar Radia
```



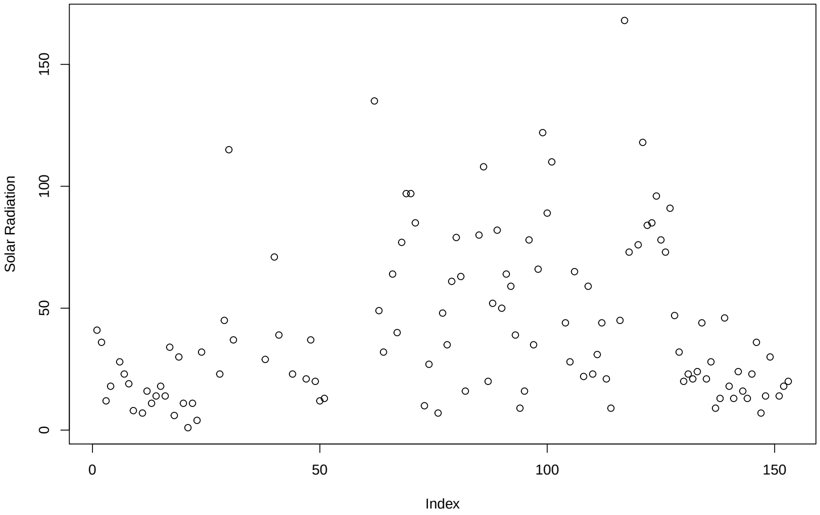
Axis labels

```r
plot(data[,1], xlab="Ozone level")
```



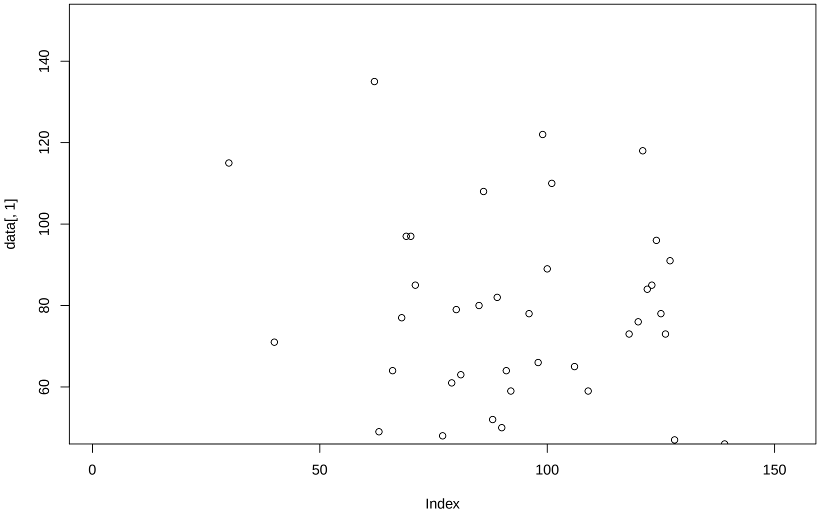
Axis labels

```
plot(data[,1], ylab="Solar Radiation")
```



Axis limits

```
plot(data[,1], ylim=c(50,150))
```



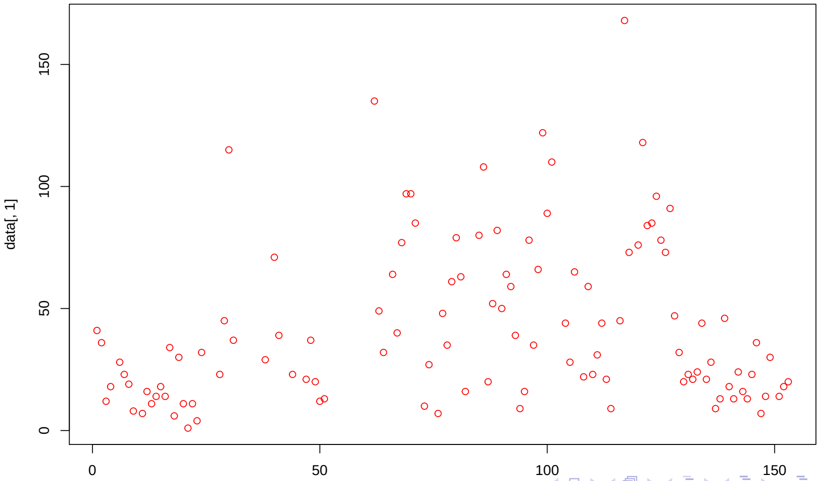
Defining a colour

- R can recognise various strings `"red"`, `"orange","green","blue","yellow"....`
- Or more exotic ones springgreen2, gray91, grey85, khaki3, maroon, darkred, mediumspringgreen, tomato3. . . .. See `colours()`.
- See `http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf`
- Can also use **R**ed **G**reen **Blue** , hexadecimal, values

Use of colours

Changing the `col` argument to `plot` changes the colour that the points are plotted in

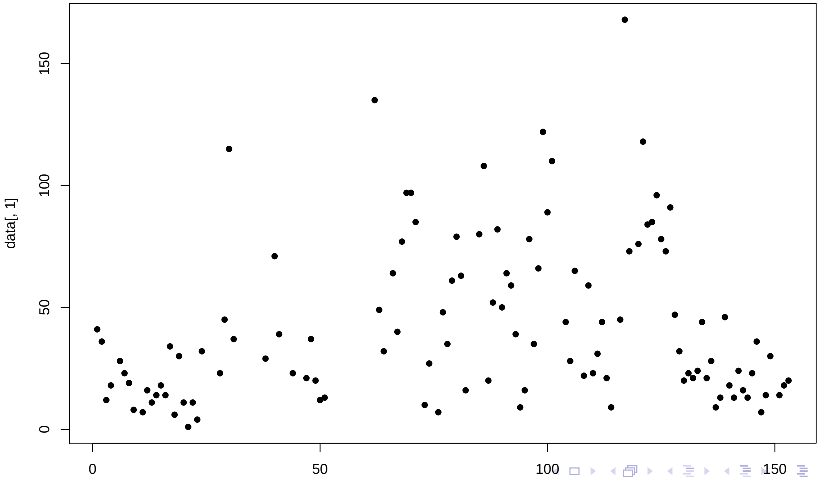
```
plot(data[,1],col="red")
```



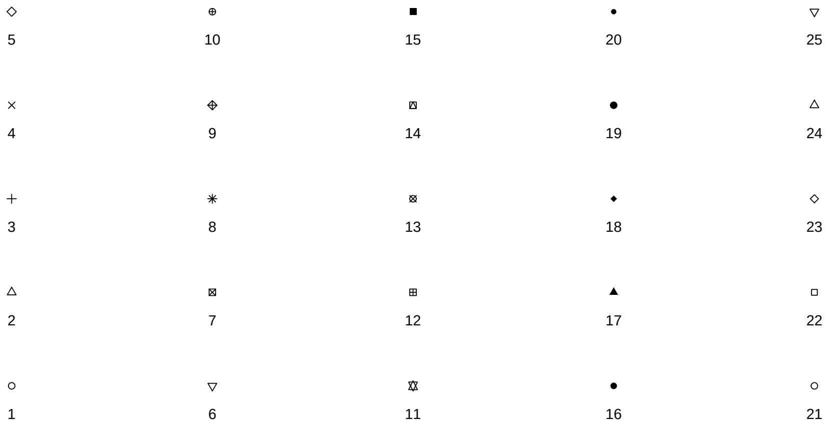
Plotting characters

- R can use a variety of *p*lotting *ch*aracters
- Each of which has a numeric *code*

```
plot(data[,1], pch=16)
```



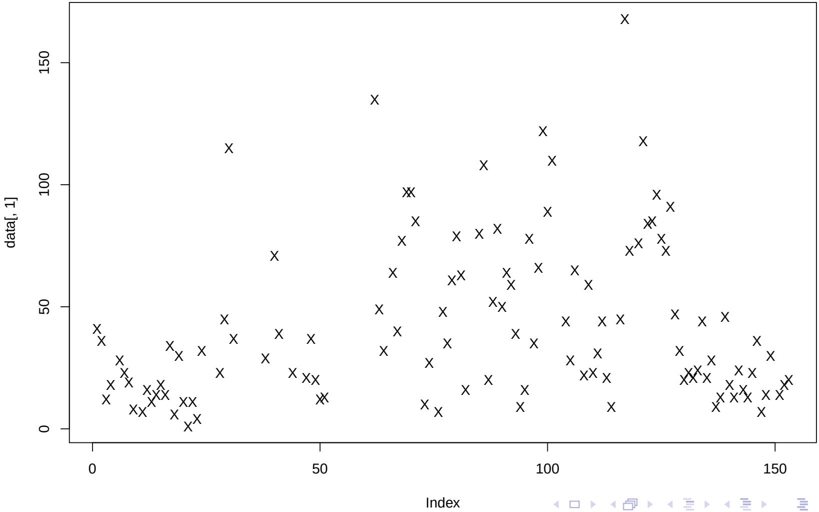
Plotting characters



Plotting characters

- Or you can specify a character

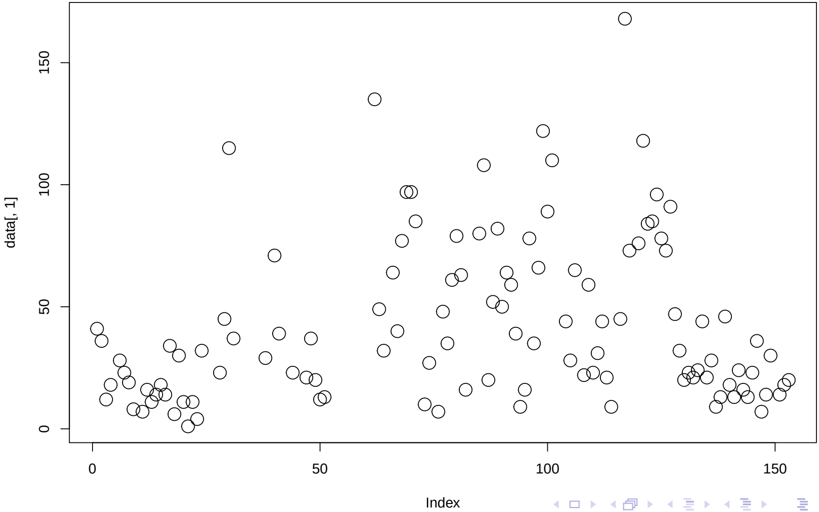
```
plot(data[,1], pch="X")
```



Size of points

Character **ex**pansion

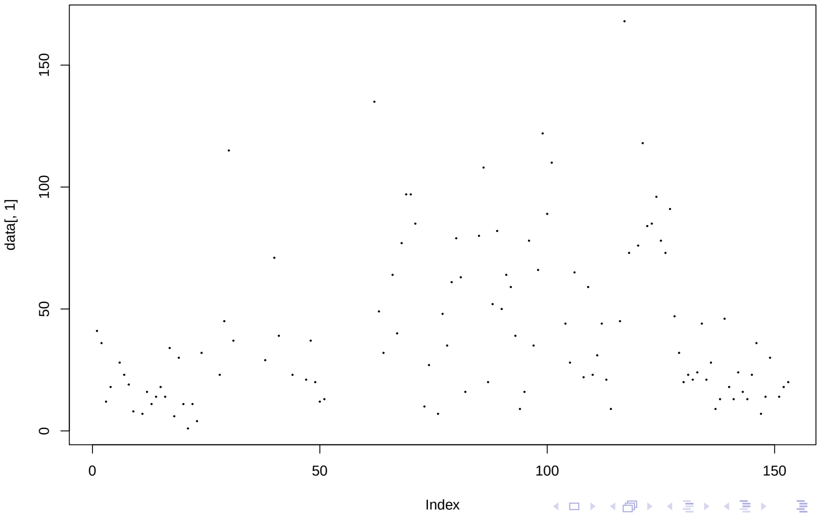
```
plot(data[,1], cex=2)
```



Size of points

Character **ex**pansion

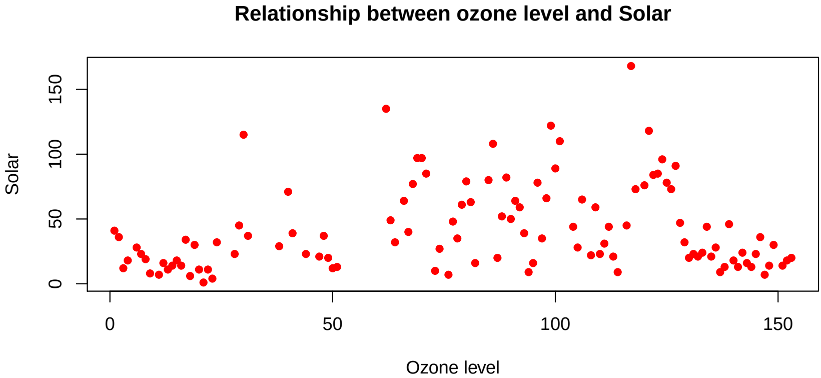
```
plot(data[,1], cex=0.2)
```



Multiple options at the same time

```r
plot(data[,1], pch=16,col="red",
     main="Relationship between ozone level and Solar",
     xlab="Ozone level",
     ylab="Solar")
```

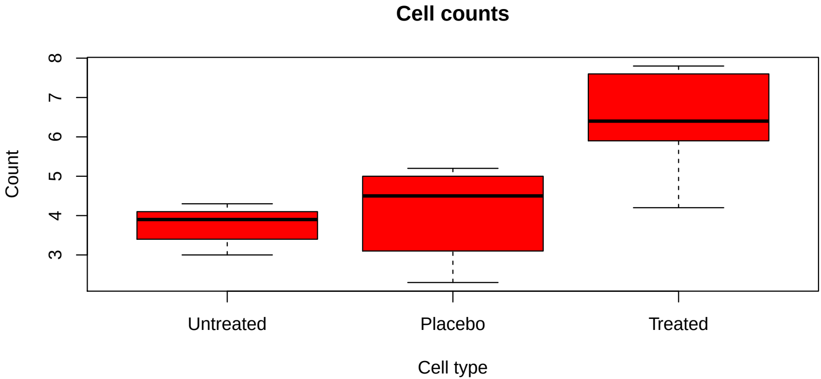
Multiple options at the same time



Applicable to other types of plot

```r
data <- read.delim("data/plasma.txt")
data
boxplot(data, main="Cell counts",xlab="Cell type",
        ylab="Count",col="red")
```

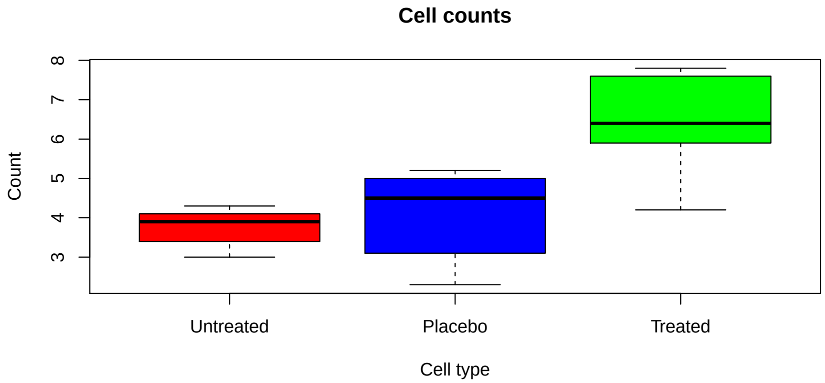
Applicable to other types of plot



What about multiple colours?

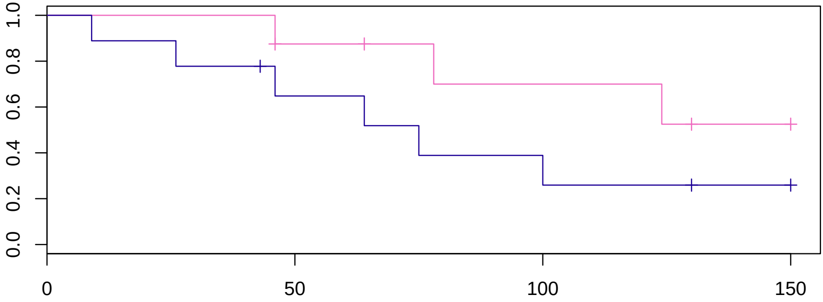
- The `col`, `pch` and `cex` arguments are *vectors*
- Previously we used a vector of length one that was *recycled*

```r
boxplot(data, main="Cell counts",xlab="Cell type",
        ylab="Count",col=c("red","blue","green"))
```



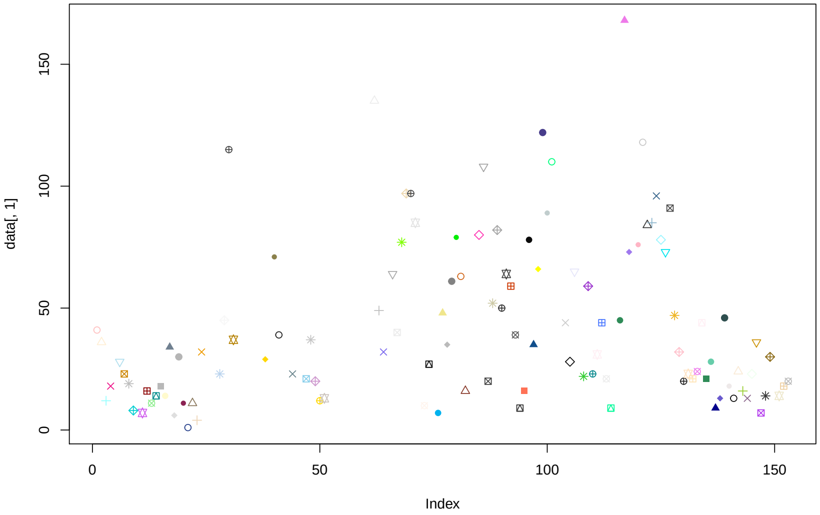
Applicable to other types of plot

```
plot(survfit(SurvData ~ Group),
     col=c(CRUKcol("Pink"),CRUKcol("Blue")))
```



Don't get carried away

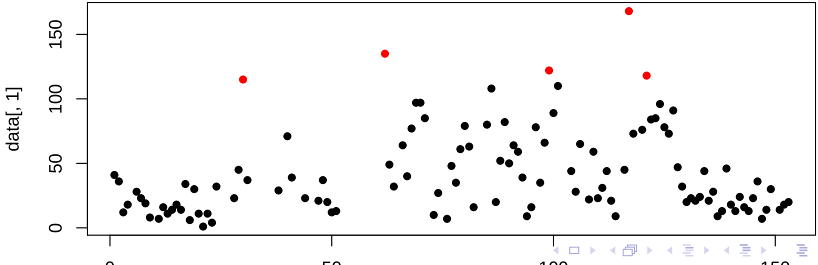
- Each point can have a unique colour, plotting character, size.



Can modify specific points

- Suppose we know that observations 117, 62, 99, 121 and 30 were the highest ozone level
- We may wish to plot them a different colour
- *a* Solution: Create a vector of colours the required length and modify the appropriate entries

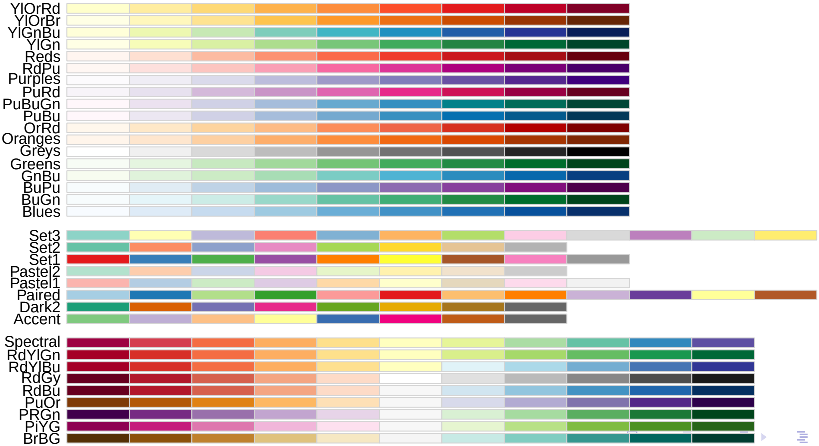
```
mycols <- rep("black", 153)
mycols[c(117,62,99,121,30)] <- "red"

plot(data[,1], pch=16, col=mycols)
```



Using a palette

- The RColorBrewer package has various ready-made colour schemes

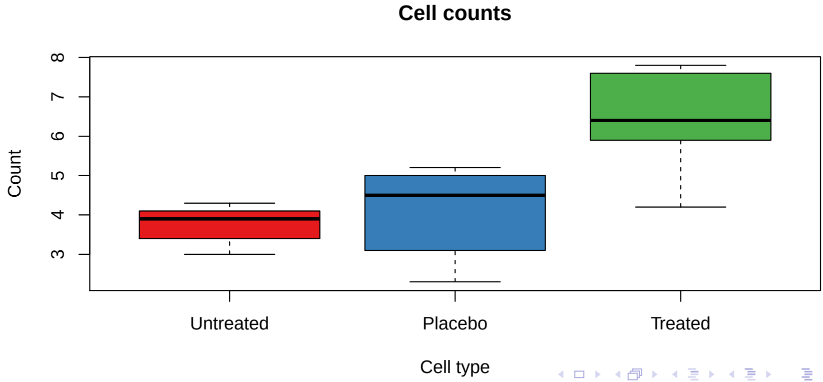
```
library(RColorBrewer)
display.brewer.all()
```



Creating a palette

- brewer.pal function creates a vector of the specified length comprising colours from the named palette

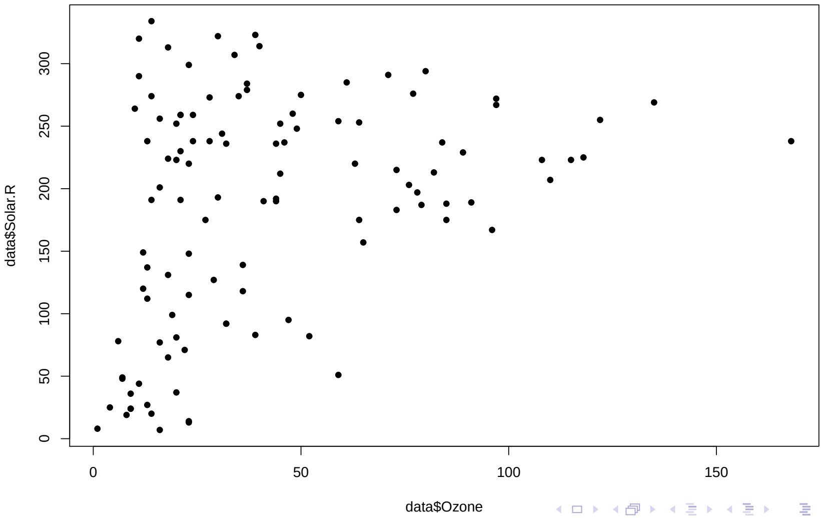
```
mypal <- brewer.pal(3, "Set1")
boxplot(data, main="Cell counts",xlab="Cell type",
        ylab="Count",col=mypal)
```



Modifying an existing plot

Initial plot

```
data <- read.csv("data/ozone.csv")
plot(data$Ozone, data$Solar.R,pch=16)
```

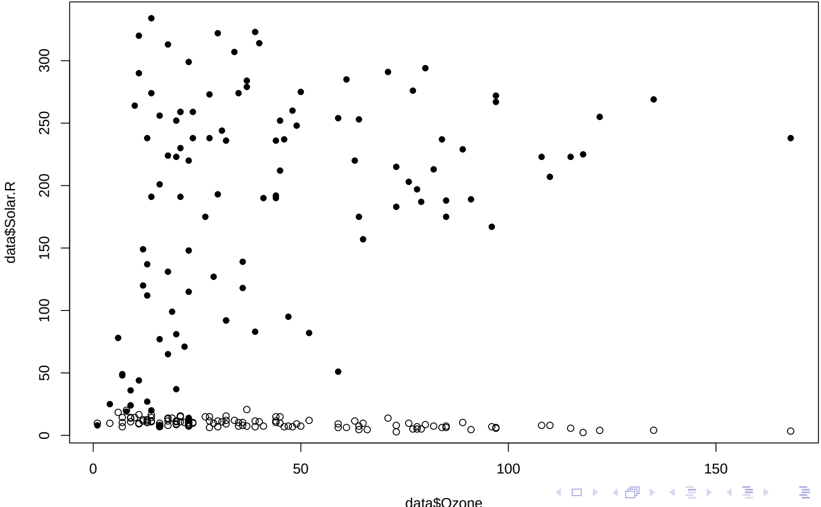


The points function

- `points` can be used to set of points to an *existing* plot
- it requires a vector of x and y coordinates
- Note that axis limits of the existing plot are not altered

Adding points

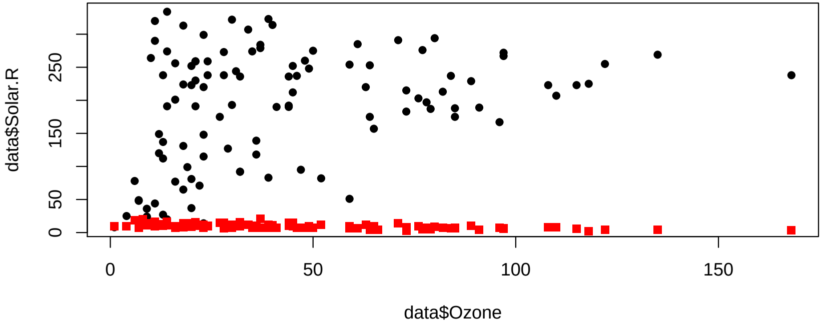
```r
data <- read.csv("data/ozone.csv")
plot(data$Ozone, data$Solar.R,pch=16)
points(data$Ozone, data$Wind)
```



Adding points

points can also use the pch, col arguments. Useful for distinguishing between variables

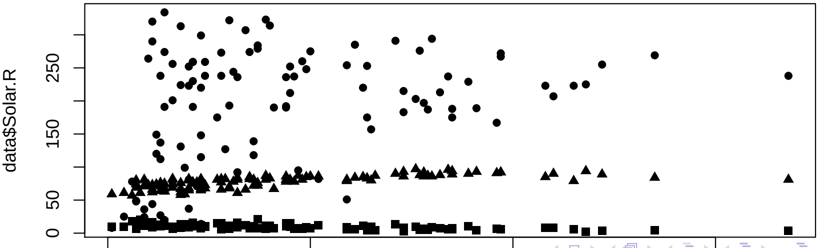
```
data <- read.csv("data/ozone.csv")
plot(data$Ozone, data$Solar.R,pch=16)
points(data$Ozone, data$Wind,pch=15,col="red")
```



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!
- A call to `plot` will start a new graphics window

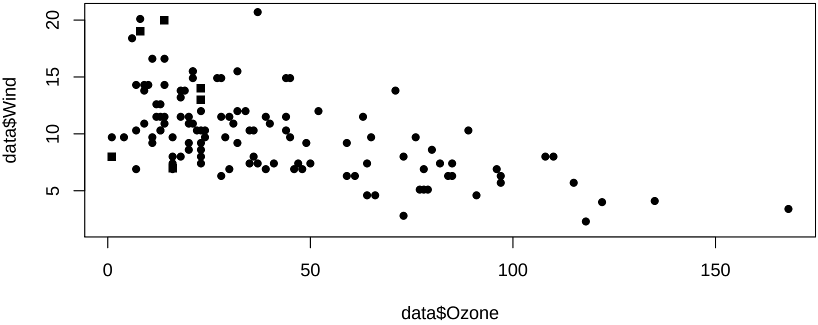
```
data <- read.csv("data/ozone.csv")
plot(data$Ozone, data$Solar.R,pch=16)
points(data$Ozone, data$Wind,pch=15)
points(data$Ozone, data$Temp,pch=17)
```



Adding points

- Be careful about the order in which you add points

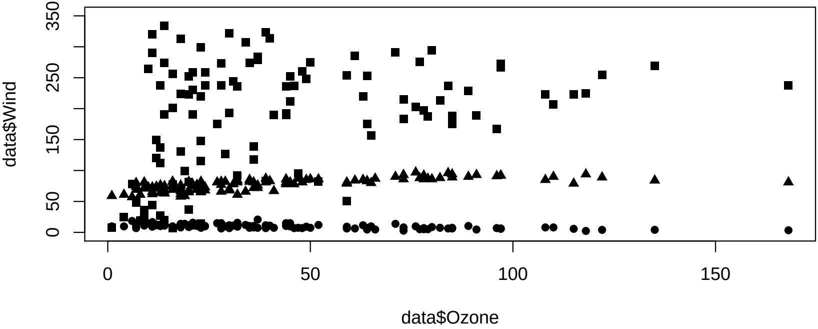
```
plot(data$Ozone, data$Wind,pch=16)
points(data$Ozone, data$Solar.R,pch=15)
points(data$Ozone, data$Temp,pch=17)
```



Adding points

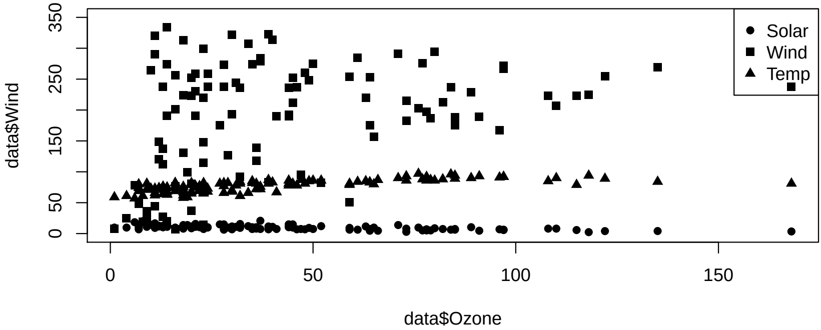
- Can define suitable axis limits in initial plot

```
plot(data$Ozone, data$Wind,pch=16,ylim=c(0,350))
points(data$Ozone, data$Solar.R,pch=15)
points(data$Ozone, data$Temp,pch=17)
```



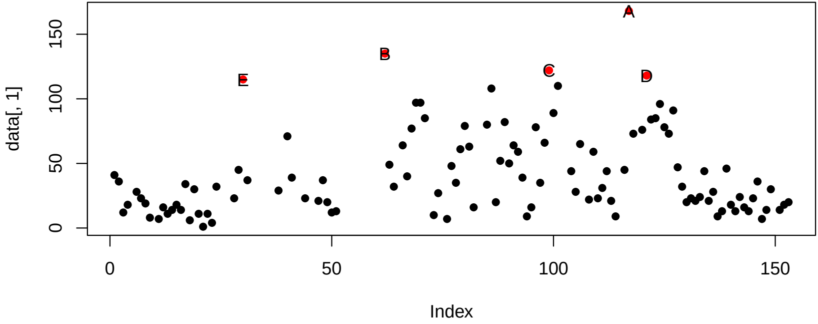
Adding a legend

```
plot(data$Ozone, data$Wind,pch=16,ylim=c(0,350))
points(data$Ozone, data$Solar.R,pch=15)
points(data$Ozone, data$Temp,pch=17)
legend("topright", legend=c("Solar","Wind","Temp"),
       col="black", pch=c(16,15,17))
```



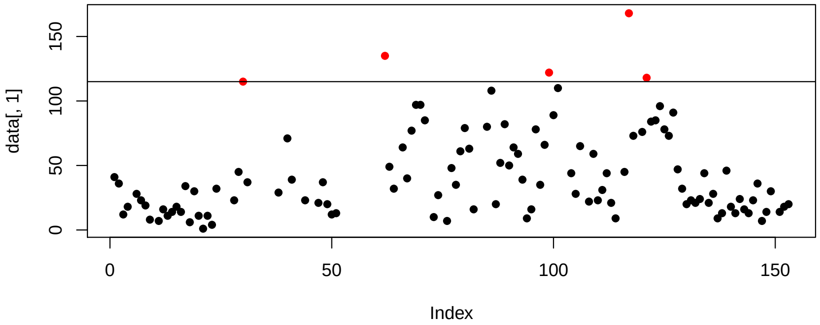
Adding text

```
mycols <- rep("black", 153)
mycols[c(117,62,99,121,30)] <- "red"

plot(data[,1], pch=16, col=mycols)
text(c(117,62,99,121,30), data[c(117,62,99,121,30),1],
     labels=LETTERS[1:5])
```



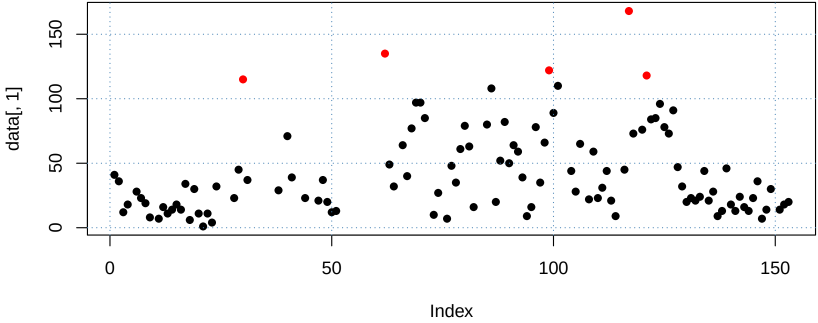
Adding lines

```
mycols <- rep("black", 153)
mycols[c(117,62,99,121,30)] <- "red"

plot(data[,1], pch=16, col=mycols)
abline(h = 115)
```



Adding lines

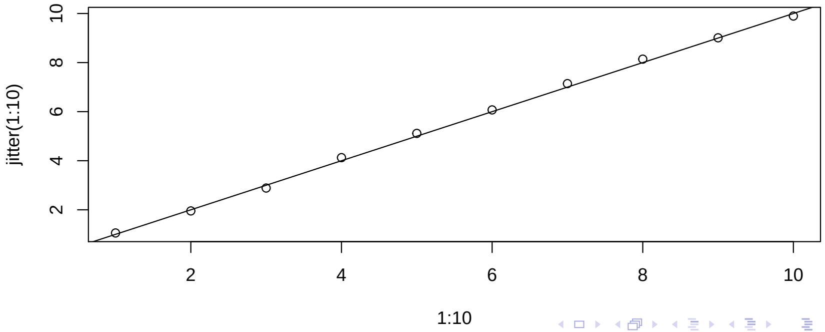
```
plot(data[,1], pch=16, col=mycols)
grid(col="steelblue")
```



Adding lines

- `abline` can take a gradient and intercept argument
- for `y = x` use `a=0` and `b=1`
- Can be used to draw a *line of best fit* in conjunction with a linear model

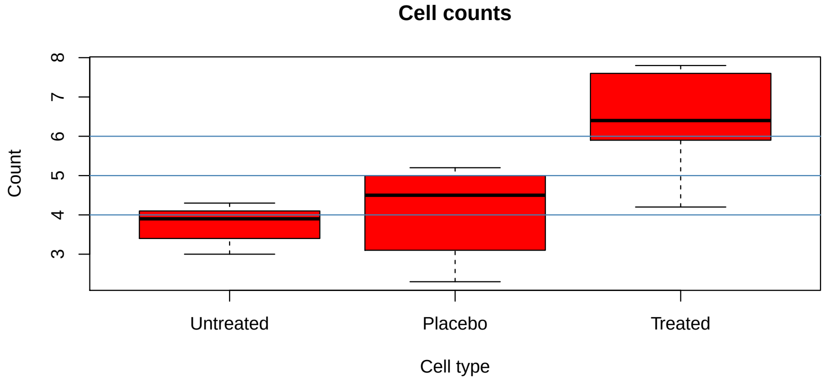
```r
plot(1:10, jitter(1:10))
abline(0,1)
```



Adding lines

Lines can also be added to other plots

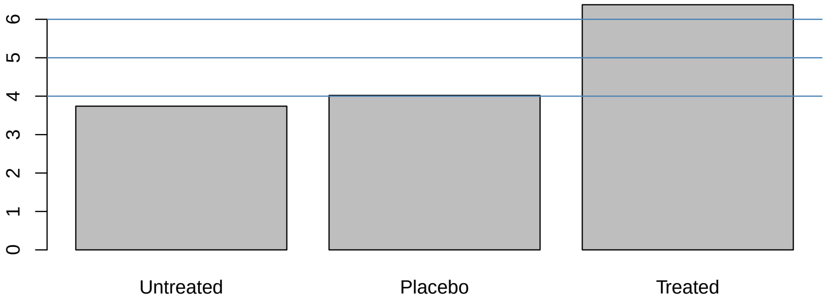
```
boxplot(data, main="Cell counts",xlab="Cell type",
        ylab="Count",col="red")
abline(h=c(4,5,6),col="steelblue")
```



Adding lines

Lines can also be added to other plots

```
barplot(colMeans(data))
abline(h=c(4,5,6),col="steelblue")
```



See also

- rect `example(rect)`
- polygon `example(polygon)`
- segments `example(segments)`

Plot layout options

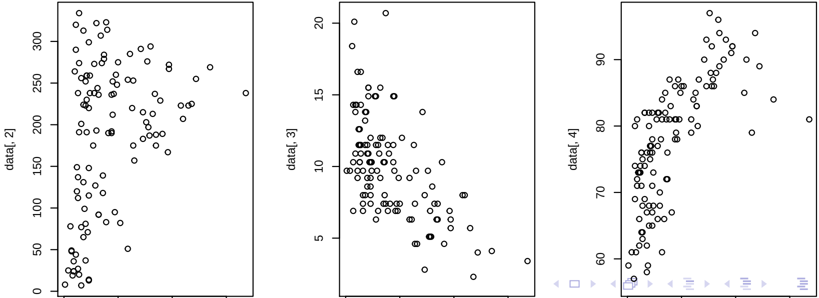
The `par` function

- Using the `par` function prior to creating a plot allows several plot defaults to be set
- ?par for details

Multiple figures

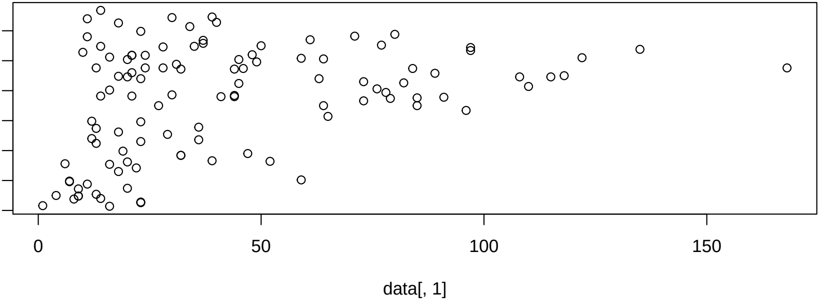
- We can have *multiple figures* per *row* using `mfrow`
- e.g. one row and three columns
- each new call to plot is added in a new panel
- see also `mfcol`

```
par(mfrow=c(1,3))
plot(data[,1],data[,2])
plot(data[,1],data[,3])
plot(data[,1],data[,4])
```



Margin size

- the `mar` vector specifies that amount of space around each edge of the plot
- `c(bottom, left, top, right)`



Exporting a plot

As a png

- png function prior to code to create plot
- file is created in your working directory (doesn't need to exist)
- `dev.off()` afterwards
- can also make jpeg in similar fashion

```
png("mycoolplot.png")
plot(data[,1],data[,2])
dev.off()
```

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

As a pdf

- As before, except use `pdf`

```r
pdf("mycoolplot.pdf")
plot(data[,1],data[,2])
dev.off()
```

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

As a pdf

- However, a pdf can have multiple pages
- Can *annotate* by program such as Photoshop
- Can specify dimensions, dpi etc

```r
pdf("mycoolmultipageplot.pdf")
plot(data[,1],data[,2])
plot(data[,1],data[,3])
dev.off()
```

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

Break for practical

Data Manipulation

Mark Dunning

12/12/2014

Data in R are not static

- We can add new variables and observations
- Re-order / sort the existing data
- Create subsets
- Create copies of our data
- Remove old copies using `rm`

Calculating new variables

```r
data <- read.csv("data/ozone.csv")
TempCelc <- (data$Temp - 32)/1.8
data$TempCelc <- TempCelc
head(data)
```

```
##   Ozone Solar.R Wind Temp Month Day TempCelc
## 1    41     190  7.4   67     5   1    19.44
## 2    36     118  8.0   72     5   2    22.22
## 3    12     149 12.6   74     5   3    23.33
## 4    18     313 11.5   62     5   4    16.67
## 5    NA      NA 14.3   56     5   5    13.33
## 6    28      NA 14.9   66     5   6    18.89
```

Appending columns

```
data <- read.csv("data/ozone.csv")
TempCelc <- (data$Temp - 32)/1.8
newdata <- data.frame(TempCelc,
                      MonthName = month.name[data$Month])
head(cbind(data, newdata))
```

```
##   Ozone Solar.R Wind Temp Month Day TempCelc MonthName
## 1    41     190  7.4   67     5   1    19.44       May
## 2    36     118  8.0   72     5   2    22.22       May
## 3    12     149 12.6   74     5   3    23.33       May
## 4    18     313 11.5   62     5   4    16.67       May
## 5    NA      NA 14.3   56     5   5    13.33       May
## 6    28      NA 14.9   66     5   6    18.89       May
```

Adding new observations

- We can add new rows (observations) to a dataset
- Useful if data are spread across multiple files
- Take care that columns are the same

```
newobs <- c(50, 140, 8, 67, 10,1,19.4)
data2 <- rbind(data,newobs)
tail(data2)
```

```
##     Ozone Solar.R Wind Temp Month Day
## 149    30     193  6.9   70     9  26
## 150    NA     145 13.2   77     9  27
## 151    14     191 14.3   75     9  28
## 152    18     131  8.0   76     9  29
## 153    20     223 11.5   68     9  30
## 154    50     140  8.0   67    10   1
```

Re-ordering and sorting

- At the moment, these data are in date-order

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

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

- We might want to know the hottest days

Re-ordering and sorting

```r
sort(data$Temp)
```

```
##   [1] 56 57 57 57 58 58 59 59 61 61 61 62 62 63 64 64 65
##  [24] 67 67 68 68 68 68 69 69 69 70 71 71 71 72 72 72 73
##  [47] 74 74 75 75 75 75 76 76 76 76 76 76 76 76 76 77 77
##  [70] 78 78 78 78 78 79 79 79 79 79 79 80 80 80 80 80 81
##  [93] 81 81 81 81 82 82 82 82 82 82 82 82 82 83 83 83 83
## [116] 85 85 85 85 86 86 86 86 86 86 86 87 87 87 87 87 88
## [139] 90 91 91 92 92 92 92 92 93 93 93 94 94 96 97
```

Re-ordering and sorting

```r
sort(data$Temp,decreasing = TRUE)
```

```
##   [1] 97 96 94 94 93 93 93 92 92 92 92 92 91 91 90 90 9
##  [24] 87 87 87 87 86 86 86 86 86 86 86 85 85 85 85 85 8
##  [47] 83 83 82 82 82 82 82 82 82 82 82 81 81 81 81 81 8
##  [70] 80 80 80 80 79 79 79 79 79 79 78 78 78 78 78 78 7
##  [93] 76 76 76 76 76 76 76 76 76 75 75 75 75 74 74 74 7
## [116] 72 72 71 71 71 70 69 69 69 68 68 68 68 67 67 67 6
## [139] 64 63 62 62 61 61 61 59 59 58 58 57 57 57 56
```

Re-ordering and sorting

- What is the difference between the output of sort and order?

```
tempOrder <- order(data$Temp, decreasing = TRUE)
length(tempOrder)
```

```
## [1] 153
```

```
tempOrder
```

```
##   [1] 120 122 121 123  42 126 127  43  69  70 102 125  7
##  [18]  71  99  68  89 119  39  41  80  98 128  85  88  9
##  [35]  36  63  81  86  97  35  62  65  79 129  61  66  6
##  [52]  78  84  87  95 105 143  29  64  74  77  83  92  9
##  [69]  45  59  76 106 130  30  37  46 107 109 116  32  5
##  [86]  47  52  60 108 113 136 150  31  51  53  54  55 11
## [103] 115 132 151   3  11  33  82  22  50  58  73 133
## [120] 145 149  10  12 147  14  19 142 153   1  28  34 14
## [137]  49  16 144 148   4  20   9  23  24   8  21  15  2
```

Re-ordering and sorting

- `sort` gives the *values* in sorted order
- `order` gives *indices*
- we can use the result of `order` to subset the data

```
tempOrder[1:5]
```

```
## [1] 120 122 121 123  42
```

```
data[tempOrder[1:5],]
```

```
##     Ozone Solar.R Wind Temp Month Day
## 120    76     203  9.7   97     8  28
## 122    84     237  6.3   96     8  30
## 121   118     225  2.3   94     8  29
## 123    85     188  6.3   94     8  31
## 42     NA     259 10.9   93     6  11
```

Writing a new file

- At this point, we might want to write our re-ordered data to a file

```
newData <- data[tempOrder,]
dim(newData)
```

```
## [1] 153   6
```

```
write.csv(newData, file="reorderedWeather.csv")
```

General Subsetting

- We have already seen how to subset using numeric indexes
- We can also subset using *logical* vectors
- i.e. a vector of `TRUE` or `FALSE` values

```
myvec <- 1:10
myvec
```

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

```
myvec[c(TRUE, TRUE, FALSE,FALSE,TRUE,TRUE,
        FALSE, FALSE,FALSE,TRUE)]
```

```
## [1]  1  2  5  6 10
```

General Subsetting

- The `TRUE` or `FALSE` values can be derived from a test such as
- <, >, `==`, `!=`
- Mulitple conditions can be tested using & (`and`) | (*or*)
- Also `is.na`, `is.infinite` and more. . . .

Adding points

- Suppose we are interested in days with Ozone level over 100
- Use the > function
- Get a TRUE or FALSE for every observation

```
data$Ozone > 100
```

```
##    [1] FALSE FALSE FALSE FALSE    NA FALSE FALSE FALSE FA
##   [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA
##   [23] FALSE FALSE    NA    NA    NA FALSE FALSE  TRUE FA
##   [34]    NA    NA    NA    NA FALSE    NA FALSE FALSE
##   [45]    NA    NA FALSE FALSE FALSE FALSE FALSE    NA
##   [56]    NA    NA    NA    NA    NA    NA  TRUE FALSE FA
##   [67] FALSE FALSE FALSE FALSE FALSE    NA FALSE FALSE
##   [78] FALSE FALSE FALSE FALSE FALSE    NA    NA FALSE  T
##   [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA
##  [100] FALSE  TRUE    NA    NA FALSE FALSE FALSE    NA FA
##  [111] FALSE FALSE FALSE FALSE    NA FALSE  TRUE FALSE
##  [122] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FA
```

Adding points

- Get the TRUE indices using the which function

```r
highOzone <- which(data$Ozone > 100)
highOzone
```

```
## [1]  30  62  86  99 101 117 121
```

Adding points

- Now do the subset

```
data[highOzone,]
```

```
##     Ozone Solar.R Wind Temp Month Day
## 30    115     223  5.7   79     5  30
## 62    135     269  4.1   84     7   1
## 86    108     223  8.0   85     7  25
## 99    122     255  4.0   89     8   7
## 101   110     207  8.0   90     8   9
## 117   168     238  3.4   81     8  25
## 121   118     225  2.3   94     8  29
```

- Could write this to a file if we wish. . . .

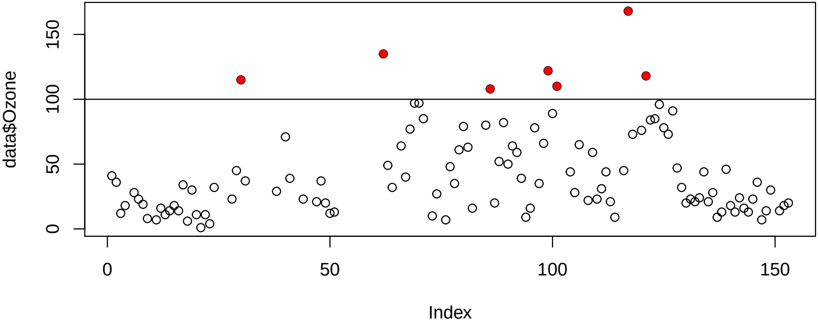
Adding points

- The `points` funtion is used to add points to an existing plot
- We need to give it a set of x and y coordinates
- The x values are the indices we've just computed.
- y values are obtained by subsetting the *Ozone* variable

```
newX <- highOzone
newY <- data$Ozone[newX]
```

Adding points

```
highOzone <- which(data$Ozone > 100)
plot(data$Ozone)
abline(h=100)
points(newX, newY,col="red",pch=16)
```



Subsetting by text

We now consider the clinical characteristics of a breast cancer cohort

```
clinical <- read.delim("data/NKI295.pdata.txt")
table(clinical$ER)
```

```
##
## Negative Positive
##       69      226
```

Subsetting by text

We might wish to know the identity of *ER negative* samples

- Note the double ==

```
clinical$ER == "Negative"
```

```
##   [1] FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE FA
##  [12] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FA
##  [23] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  T
##  [34] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FA
##  [45] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FA
##  [56]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FA
##  [67] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FA
##  [78] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FA
##  [89] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FA
## [100] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FA
## [111] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FA
## [122] FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FA
## [133]  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FA
```

Returning indices

```r
which(clinical$ER == "Negative")
```

```
##  [1]   3   4   7  13  19  22  28  31  32  33  35  38  50
## [18]  63  66  68  71  76  82  90  92  94  96  99 101 110
## [35] 128 133 135 137 139 143 144 145 148 151 153 160 167
## [52] 206 209 222 224 228 230 231 233 236 239 242 248 262
## [69] 293
```

Potential trap number 1.

```r
which(clinical$ER == "negative")
```

```
## integer(0)
```

Potential trap number 2.

```r
which(clinical$er == "Negative")
```

```
## integer(0)
```

Potential trap number 3.

```r
match("Negative", clinical$ER)
```

```
## [1] 3
```

Use in subsetting

```r
clinical[which(clinical$ER == "Negative"),]
```

```
##     sampleNames Label_Traing_and_Validation event_death
## 3      NKI295_7                    Training           0
## 4      NKI295_8                  Validation           0
## 7     NKI295_12                    Training           0
## 13    NKI295_28                    Training           0
## 19    NKI295_48                  Validation           1
## 22    NKI295_57                    Training           1
## 28    NKI295_71                    Training           1
## 31    NKI295_75                  Validation           1
## 32    NKI295_76                    Training           1
## 33   NKI295_103                  Validation           1
## 35   NKI295_109                    Training           0
## 38   NKI295_113                  Validation           1
## 50   NKI295_130                    Training           0
## 51   NKI295_131                  Validation           1
## 55   NKI295_135                  Validation           0
```

Alternative

- grep finds indices of all entries that match

```
clinical[grep("Negative",clinical$ER),]
```

```
##      sampleNames Label_Traing_and_Validation event_death
## 3       NKI295_7                    Training           0
## 4       NKI295_8                  Validation           0
## 7      NKI295_12                    Training           0
## 13     NKI295_28                    Training           0
## 19     NKI295_48                  Validation           1
## 22     NKI295_57                    Training           1
## 28     NKI295_71                    Training           1
## 31     NKI295_75                  Validation           1
## 32     NKI295_76                    Training           1
## 33    NKI295_103                  Validation           1
## 35    NKI295_109                    Training           0
## 38    NKI295_113                  Validation           1
## 50    NKI295_130                    Training           0
```

Match multiple strings

```
clinical[which(clinical$Fan.nearest.centroid %in%
                  c("Basal", "HER2")),]
```

```
##       sampleNames Label_Traing_and_Validation event_death
## 4      NKI295_8                  Validation           0
## 6     NKI295_11                  Validation           0
## 7     NKI295_12                    Training           0
## 19    NKI295_48                  Validation           1
## 22    NKI295_57                    Training           1
## 27    NKI295_62                  Validation           1
## 28    NKI295_71                    Training           1
## 31    NKI295_75                  Validation           1
## 32    NKI295_76                    Training           1
## 33   NKI295_103                  Validation           1
## 35   NKI295_109                    Training           0
## 37   NKI295_111                    Training           1
## 50   NKI295_130                    Training           0
## 51   NKI295_131                  Validation           1
```

Useful functions for manipulating text

- substr

```
substr(clinical$sampleNames,1,3)[1:5]
```

```
## [1] "NKI" "NKI" "NKI" "NKI" "NKI"
```

```
substr(clinical$sampleNames,1,3)[1:5] == "NKI"
```

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

Useful functions for manipulating text

- strtrim

```
strtrim(clinical$sampleNames,3)[1:5]
```

```
## [1] "NKI" "NKI" "NKI" "NKI" "NKI"
```

Useful functions for manipulating text

- strsplit

```
strsplit(as.character(clinical$sampleNames), "_")[[1]]
```

```
## [1] "NKI295" "4"
```

```
matrix(unlist(strsplit(as.character(clinical$sampleNames),
       ,ncol=2,byrow=TRUE)
```

```
##          [,1]     [,2]
##    [1,] "NKI295" "4"
##    [2,] "NKI295" "6"
##    [3,] "NKI295" "7"
##    [4,] "NKI295" "8"
##    [5,] "NKI295" "9"
##    [6,] "NKI295" "11"
##    [7,] "NKI295" "12"
##    [8,] "NKI295" "13"
```

Useful functions for manipulating text

Not an extensive list

- `toupper`, `tolower` - convert upper / lower case
- `gsub` - substitute text
- `paste` - combine text
- `intersect`, `setdiff` see which is in common

Combining data from files

- Now look at typical gene expression matrix
- Each row corresponds to a *gene*
- Each column is a *sample*

```
evalues <- read.delim("data/NKI295.exprs.txt")
dim(evalues)
```

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

```
evalues[1:5,1:5]
```

```
##    X NKI295_4 NKI295_6 NKI295_7 NKI295_8
## 1 16  -0.7130  0.23551   0.6052  -1.1407
## 2 17  -0.6884  0.18337   0.2555   1.0043
## 3 18  -0.5237 -0.03184   0.1948   0.5602
## 4 19  -2.7191 -1.30018  -2.0737  -1.7526
## 5 20  -0.8871 -1.02838  -0.3982  -1.4834
```

Clinical data

- Each row is a *sample*
- Each column is a different *clinical* variable
- Can have as many columns as you like
- e.g. first five rows in sample information are first five columns in expression matrix

```
clindata <- read.delim("data/NKI295.pdata.txt")
clindata[1:5,1:5]
```

```
##   sampleNames Label_Traing_and_Validation event_death
## 1    NKI295_4                    Training           0
## 2    NKI295_6                  Validation           0
## 3    NKI295_7                    Training           0
## 4    NKI295_8                  Validation           0
## 5    NKI295_9                    Training           0
##   Distant_metastasis_as_first_event.MCR. survival.death.
## 1                                      0          12.997
## 2                                      0          11.157
```

Matching-up the columns

- Good to check that columns of expression matrix match the clinical data

```
length(intersect(colnames(evalues), clindata[,1]))
```

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

```
setdiff(colnames(evalues), clindata[,1])
```

```
## [1] "X"
```

```
setdiff(clindata[,1],colnames(evalues))
```

```
## character(0)
```

```
all(clindata[,1] == colnames(evalues)[-1])
```

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

Matching-up the columns

- Find columns in the clinical data that match the clinical data

```r
neword <- match(clindata[,1], colnames(evalues))
evalues.reorder <- evalues[,neword]
```

Clinical data

- Can also go from clinical to gene expression matrix
- e.g. if we know what *rows* in the clinical matrix correspond to ER negatives, we will what *columns* they are in the gene expression matrix

```
which(clindata$ER == "Negative")[1:4]
```

```
## [1]  3  4  7 13
```

- Columns 3,4,7,13 are all ER negative samples

Gene annotation

- Each row is a *gene* in the experiment
- Each column is *annotation* about that gene
- e.g. Rows 1 to 5 in the annotation matrix tell us about rows 1 to 5 in the expression matrix

```
annodata <- read.delim("data/NKI295.fdata.txt")
annodata[1:5,1:5]
```

```
##   probeID  symbol
## 1      16   GREM2
## 2      17 ZNF280B
## 3      18
## 4      19     FGB
## 5      20  SCARA5
##                                                     des
## 1 Gremlin 2, cysteine knot superfamily, homolog (Xenopus
## 2                                       Zinc finger prot
## 3                                           MRNA, clone
```

Gene annotation

- e.g. row 1 in the expression matrix is the gene expression values for GREM2

```
annodata[1,]
```

```
##   probeID symbol
## 1      16   GREM2
##                                                     des
## 1 Gremlin 2, cysteine knot superfamily, homolog (Xenopus
##   ensg  unigene
## 1      Hs.98206
```

-

Example analyses

- Extract the expression values for a given gene
- Extract the arrays representing a particular clinical subgroup
- Plot gene expression against particular clinical variables
- Compare expression of one gene against another

Wrap-up

Things we didn't mention

- loops, if / else etc
- apply, lapply
- writing functions
- ggplot2 `http://ggplot2.org/`
- Bioconductor `http://bioconductor.org/`

Don't be a stranger!

- Email if you need help mark.dunning@cruk.cam.ac.uk
- Internal online support forum. Go to bioinf-qa001/ in web-browser
- `http://www.meetup.com/Cambridge-R-Users-Group-Meetup/`

Other references

- R cookbook `http://www.cookbook-r.com/`
- Quick-R `http://www.statmethods.net/`
- UC Riverside guide `http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual`
- Course Manual

Practice!

- Lots of example datasets are available online

`http://vincentarelbundock.github.io/Rdatasets/datasets.html`

Break for final practical