R workshop: Data Manipulation and Graphics

Fernando E. Miguez

October 11, 2008

1 Preliminaries

Some important facts about R :

- Everything in R is an *object*
- Every object in R has a *class*
- We operate on objects using *functions*
- The *class* of an object determines how a function behaves when applied to it.

There are many function which are generic. This functions look the same to the user but behave differently depending on the class of the object we are manipulating.

Functions have arguments. For example,

```
?t.test
x <- rnorm(100)
res.t <- t.test(x)
class(res.t)
methods(class = "htest")
print(res.t)
## or simply
res.t
```

Probably, the two most useful generic functions are: `summary` are `plot`.

1.1 R resources

1.1.1 Website

R webpage: http://www.r-project.org

It is a good idea to read "An Introduction to R" from the website. Tip: see section 10.8 to learn how to customize your session.

1.1.2 Books

Data Manipulation with R. (2008). Phil Spector. Springer.

Lattice: Multivariate Data Visualization with R. (2008) Deepayan Sarkar. Springer.

Modern Applied Statistics with S. (2002) W. N. Venables and Brian D. Ripley Springer. 4th Edition.

R in Ecology

Ecological Models and Data in R. (2008) Benjamin Bolker. Princeton University Press.

Models for Ecological Data. (2007) Clark, J.S. Princeton University Press.

Ecological data models with R. (2007) Clark, J.S. Princeton University Press.

Hierarchical Modelling for the Environmental Sciences. (2006) Clark. J.S. and A. E. Gelfand (eds). Oxford University Press, Oxford, England.

1.2 Editors

If you will be using R fairly frequently you should use an editor.

- Windows: Tinn-R (free, recommended), RWinEdt.
- Mac: Good built-in Editor.
- Cross-platform: R commander (`Rcmdr` haven't tried it), emacs + ESS (the one I use, **steep** learing curve).

1.3 Other Software

GGobi: Multivariate Data Visualization.

Interactive and Dynamic Graphics for Data Analysis: With Examples Using R and GGobi. (2008) Dianne Cook and Deborah F. Swayne. Springer.

An introduction to multivariate statistics:
https://netfiles.uiuc.edu/miguez/www/Teaching.html

2 Data Manipulation

2.1 Loading data into R

There are many options to import and export data in R. Normally, we will only import and export text files with a `.txt` or `.csv` extension. For other more advanced options see "R Data Import/Export"

2.1.1 Data Formats

Data is typically stored in a variety of formats such as tab delimited, space delimited, comma separated. For simplicity I would recommend storing your data in a `.csv` format for input into R. To read the data in you can then use the `read.csv` function. To convert an Robject to a `.csv` file the `write.csv` function can be used.

2.2 Built-in Datasets

One nice feature of R is that it comes with built-in datasets. These datasets are useful to learn R and also to test analysis and graphics. For example, we can load a built-in data set called `trees` and read its documentation.

```
> data(trees)
> ?trees
## Looking at the begining of the object
> head(trees)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
```

```
## Looking at the end of the object
> tail(trees)
   Girth Height Volume
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
## Printing the first 6 rows
> trees[1:6,]
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
6  10.8     83   19.7
## Using the generic summary function on the data frame trees
> summary(trees)
     Girth           Height       Volume
 Min.   : 8.3   Min.   :63   Min.   :10.2
 1st Qu.:11.1   1st Qu.:72   1st Qu.:19.4
 Median :12.9   Median :76   Median :24.2
 Mean   :13.2   Mean   :76   Mean   :30.2
 3rd Qu.:15.2   3rd Qu.:80   3rd Qu.:37.3
 Max.   :20.6   Max.   :87   Max.   :77.0
## Examining the structure of the 'trees' object
> str(trees)
'data.frame': 31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

These are steps that I almost always go through before I start an analysis to ensure that I am looking at the correct data set and there are no apparent errors in data entry, units, etc.

Typically, some important tasks in data manipulation involve subsetting and merging data sets. There are many ways in which this can be achieved. Let us see a few.

```
## Selecting for printing rows 10, 20, 30
> trees[c(10,20,30),]
   Girth Height Volume
10  11.2     75   19.9
20  13.8     64   24.9
30  18.0     80   51.0
## Selecting a subset of the trees 'data frame'
## Only the first 10 observations
> trees.sub10 <- trees[1:10,]
> trees.sub10
   Girth Height Volume
1    8.3     70   10.3
2    8.6     65   10.3
3    8.8     63   10.2
4   10.5     72   16.4
5   10.7     81   18.8
6   10.8     83   19.7
7   11.0     66   15.6
8   11.0     75   18.2
9   11.1     80   22.6
10  11.2     75   19.9
## Selecting those observations with height larger than 76
> trees.sub76 <- trees[trees$Height > 76,]
> summary(trees.sub76)
     Girth           Height         Volume
 Min.   :10.7   Min.   :77.0   Min.   :18.8
 1st Qu.:12.1   1st Qu.:80.0   1st Qu.:25.8
 Median :14.2   Median :80.0   Median :34.5
 Mean   :14.9   Mean   :81.3   Mean   :40.3
 3rd Qu.:17.7   3rd Qu.:82.5   3rd Qu.:53.5
 Max.   :20.6   Max.   :87.0   Max.   :77.0
```

Let us now select a subset which includes trees with height larger than 76 **and** volume larger than 40. Notice the ampersand which combines both

logical conditions and produces a subset which satisfies both conditions (i.e. intersection set).

```
> trees.h76.v40 <- trees[trees$Height > 76 & trees$Volume > 40,]
> trees.h76.v40
   Girth Height Volume
25  16.3     77   42.6
26  17.3     81   55.4
27  17.5     82   55.7
28  17.9     80   58.3
29  18.0     80   51.5
30  18.0     80   51.0
31  20.6     87   77.0
## Try this for more information on logical operations
> ?">"
```

Another operation which is common in data manipulation is adding or removing columns from a `data.frame`.

```
## Adding an identifier to the trees data frame
> trees$ID <- 1:31
> head(trees)
## or
> trees[,4] <- 1:31
> head(trees)
## Notice that this time the column name is labelled 'V4' by default
## Let us remove this extra column
> trees <- trees[,-4]
```

This just involved adding a column. How about adding multiple columns or rows? For this we can use the `rbind` and `cbind` functions. Let us create a data set, save it and then read it back into R.

```
> set.seed(1234)
> Greeness <- rnorm(31,31,7)
> Ruggedness <- rnorm(31,50,5)
> etrees <- data.frame(Green=Greeness,Rug=Ruggedness)
> getwd()
# Will output the current working directory
```

```
> write.csv(etrees,file="eTrees.csv",row.names=F)
> rm("etrees")
> etrees <- read.csv("eTrees.csv",header=TRUE)
```

In the previous steps we have set a seed to initiate the random number generator. Then we generated two vectors using a normal distribution with different means and standard deviations. After that, we combined these vectors into a data frame and we wrote this object to a `.csv` object. We removed the object from the R work space and then we imported the data using the `read.csv` function. Now we can merge the `trees` data set with `etrees` using the `cbind` function (the `rbind` function will let you combine rows).

```
> trees.new <- cbind(trees,etrees)
> head(trees.new)
  Girth Height Volume Green  Rug
1   8.3     70   10.3  27.9 51.0
2   8.6     65   10.3  28.4 45.6
3   8.8     63   10.2  33.6 42.1
4  10.5     72   16.4  40.7 63.1
5  10.7     81   18.8  25.3 48.9
6  10.8     83   19.7  26.2 44.8
```

Let us say we want to know which row (observation) has the lowest height. We can use the `which.min` function (the `which.max` function will find the maximum).

```
## Using the which.min function
> which.min(trees.new$Height)
[1] 3
```

We know now that row 3 has the lowest value of height. We can now select this row or we could have combined these two operations into one.

```
> trees.new[which.min(trees.new$Height),]
  Girth Height Volume Green  Rug
3   8.8     63   10.2  33.6 42.1
```

Other useful functions for data manipulation are `split`, `merge`, and `reshape`.

3 Graphics

Now that we have mastered data manipulation, let us move to graphics. Making graphs in R is relatively easy if you do not want to change the defaults. Often changing some of the graphical parameters requires quite a bit of research. However, I believe that the flexibility of R graphics allows us to make highly sophisticated graphs which also take advantage of the available statistical functions. We can start by looking at the `trees` data set again.

```
pairs(trees, panel = panel.smooth, main = "trees data")
## Also try plot(trees)
```

With one line of code we were able to produce a fairly sophisticated plot, but changing some of the defaults might not be that easy (more on this latter).

Fig. 1 is only a simple demo of what can be accomplished in R, but now we go back to some basics. A simple scatterplot can be created in the following way (Fig. 2)

```
plot(Girth ~ Volume , data = trees)
## We can also add a title
title("Girth vs. Volume for the trees data set")
```

Let us change the point character (Fig. 3). Figure 4 shows some of the most commonly used point characters.

```
plot(Girth ~ Volume , data = trees, pch=16)
title("Girth vs. Volume for the trees data set")


## plot of point characters
stripplot(15:25,pch=15:25,xlab="",col="black",
      scales=list(tick.number=10),cex=2,aspect=0.1)
```

There are few options which can be clarified in the previous plot. First I am using a plotting function (`stripplot`) in the `lattice` package. Second I am specifying the from point character 15 to 25 as a sequence `15:25`. I am also supresing the labels on the axis by doing `xlab=""`. The option `col`

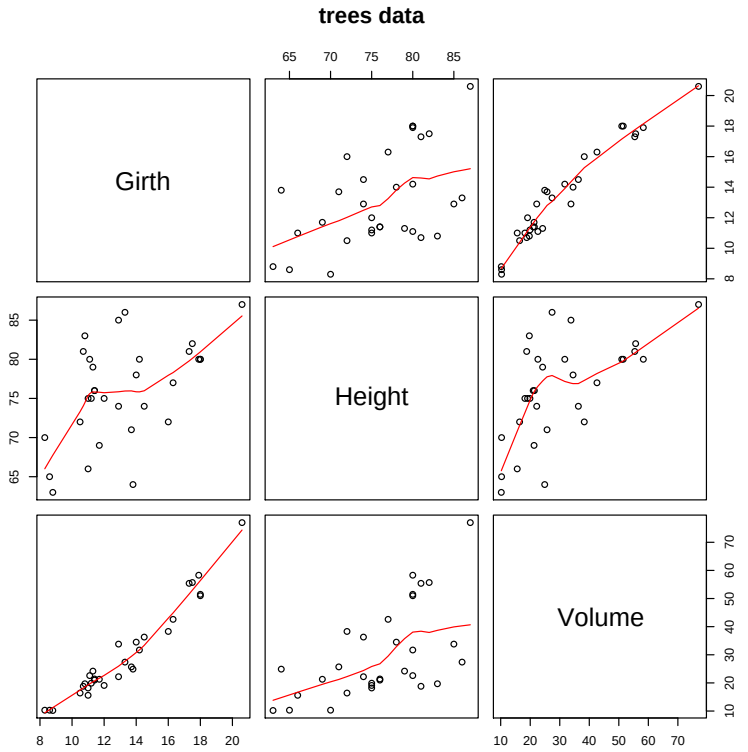

Figure 1: In this `pairs` plot the relationship between the three variables is appreciated. In addition, there is a line added using a locally weighted polynomial.

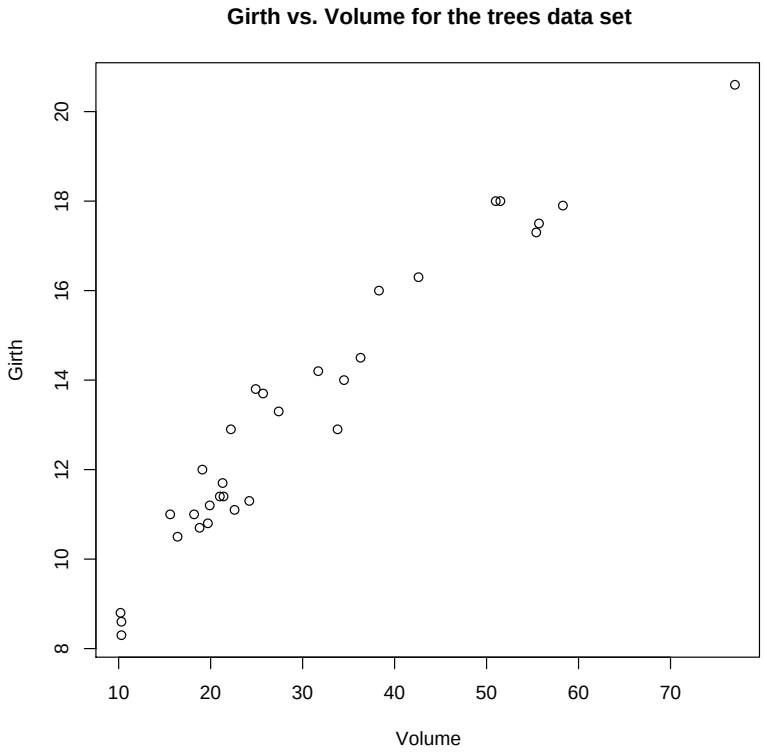

Figure 2: Scatterplot of the relationship between girth and volume for the `tree` data set

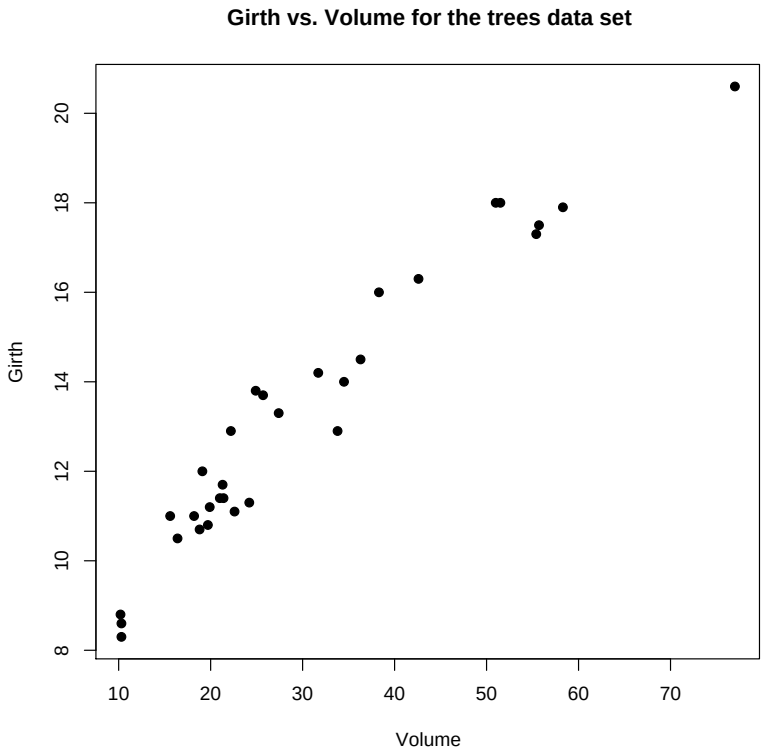

Figure 3: Scatterplot of the relationship between girth and volume for the `tree` data set. Now using a different point character.

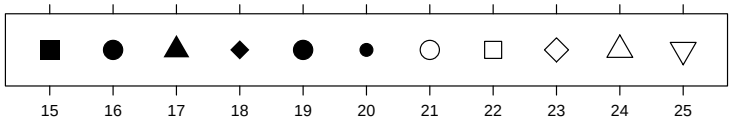

Figure 4: Illustration of the point characters between 15 and 25.

selects "black" as the color of choice (default for this function is "blue"). In addition the `scales` argument modifies the number of ticks on the x axis. The argument `cex=2` increases the character expansion by 2 and the aspect makes the plot more rectangular.

From now on I will concentrate on the `lattice` package. This is a package which is part of the recommended packages which is extremely powerful for graphical analysis of many variables. I will first show an example using a data set available in the `nlme` package (Fig. 5).

```
> library(nlme)
> data(Alfalfa)
> head(Alfalfa)
> xyplot(Yield ~ Date , data = Alfalfa)
## Want to change the color? try
xyplot(Yield ~ Date , data = Alfalfa, col = "purple", cex = 1.5)
```

The syntax of the plots in the `lattice` package allows for conditioning and grouping. First, we can condition by block using the pipe '|' (Fig. 7).

```
> xyplot(Yield ~ Date | Block , data = Alfalfa, cex = 1.5)
```

In the next step we can identify the 'groups' (varieties in this case) which is shown in Fig. 7.

```
> xyplot(Yield ~ Date | Block , type = "o", auto.key = TRUE,
           groups = Variety, data = Alfalfa, cex = 1.5)
```

Now to illustrate the use of density plots and scatterplot matrices we will use the `iris` dataset.

```
> data(iris)
> head(iris)
> densityplot(~Sepal.Length , data = iris)
## Or even better grouping by Species
> densityplot(~Sepal.Length , groups = Species ,
                       data = iris, auto.key=TRUE)

## Scatterplot matrix
> splom(~iris[,1:4] , groups = Species,
                       data = iris, auto.key=TRUE)
```

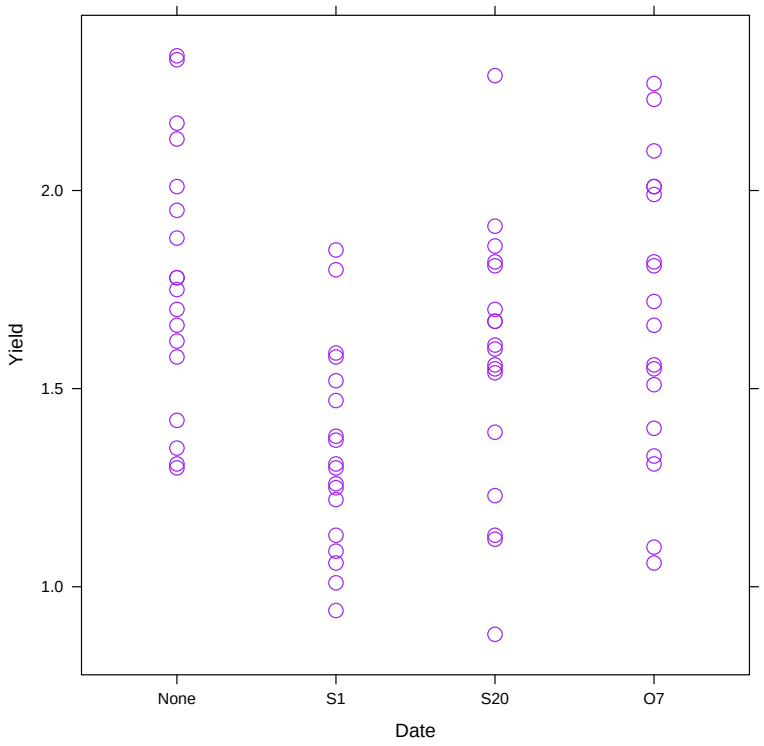

Figure 5: Scatterplot for the Alfalfa data set with Yield on the y axis and Date on the x axis.

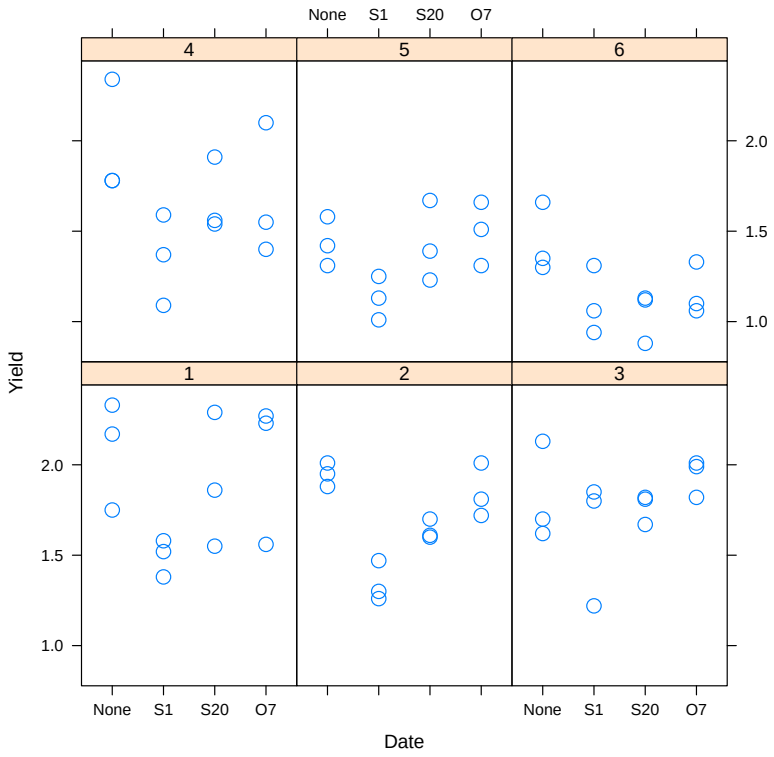

Figure 6: Scatterplot for the Alfalfa data set with Yield on the y axis and Date on the x axis, now conditioned by Block.

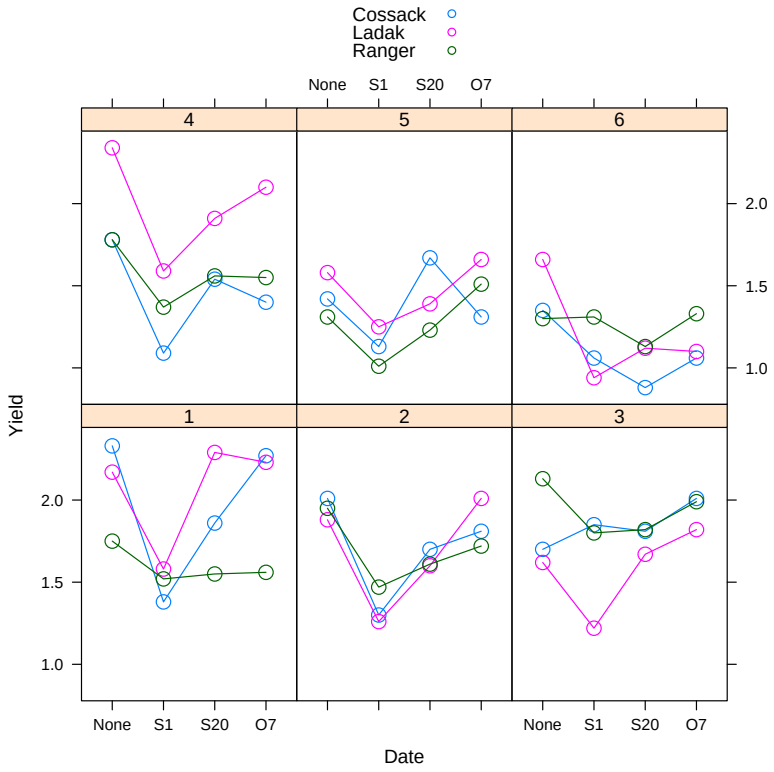

Figure 7: Scatterplot for the Alfalfa data set with Yield on the y axis and Date on the x axis, conditioned by Block and now with Varieties identified.

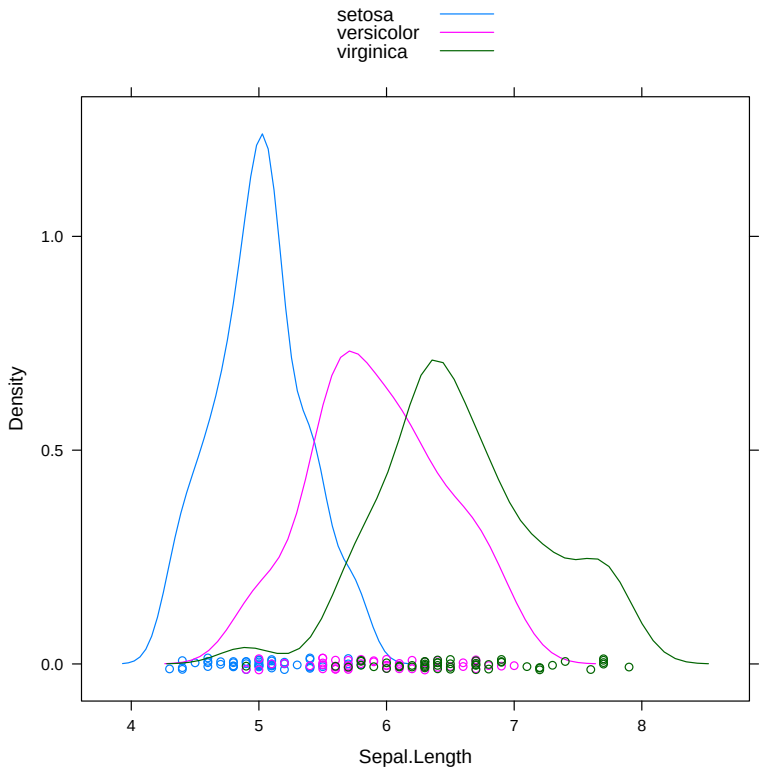

Figure 8: Density plot for `Sepal.Length` identified by `Species`

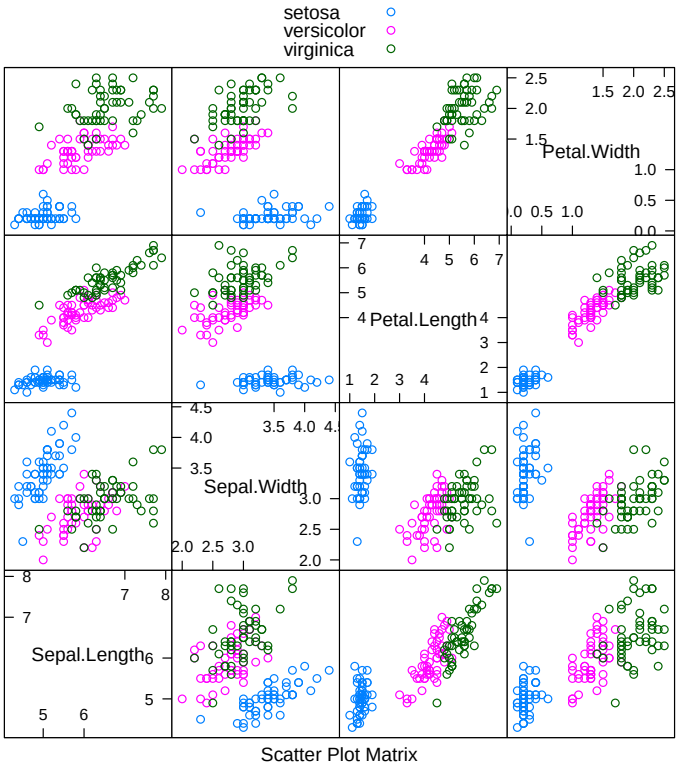

Figure 9: Scatterplot matrix for the iris dataset identified by `Species`

I will now show how to use the `bwplot`.

```
bwplot(yield ~ variety , data = barley,
       par.settings=list(box.rectangle=list(col="black"),
                         box.umbrella=list(col="black"),
                         plot.symbol=list(col="red",pch=4,cex=2)),
       aspect=0.7)
```

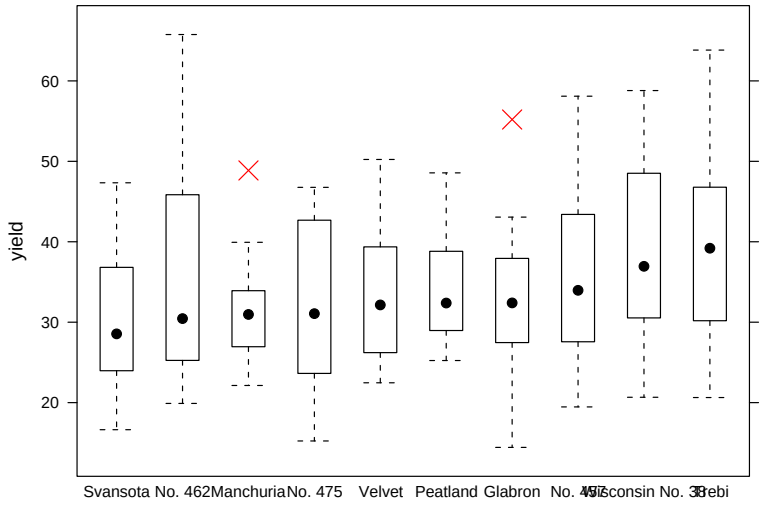

Figure 10: Box and wisker plot for the `barley` data set.

If we have extra time I will show how to do some simple statistical analysis such as regression, ANOVA and correlation.