Graphics - Part II:
Basic Graphics Continued

Statistics 135

Autumn 2005

Basic Graphics

- Pie charts

Don't Use

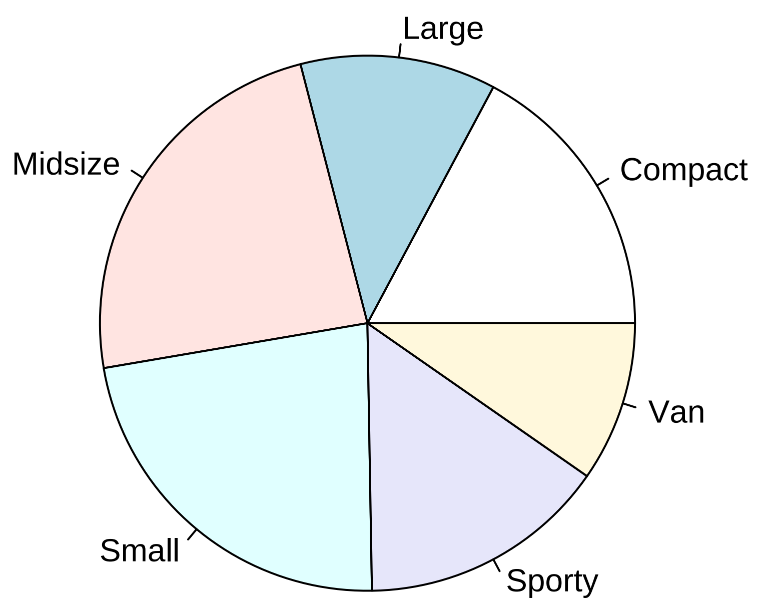
> Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data. (From `help(pie)`

However if you need to

```
Usage:

     pie(x, labels = names(x), edges = 200, radius = 0.8,
         density = NULL, angle = 45, col = NULL,
         border = NULL, lty = NULL, main = NULL, ...)
```

```
Arguments:

       x: a vector of positive quantities. The values in
          'x' are displayed as the areas of pie slices.

  labels: a vector of character strings giving names for
          the slices. For empty or NA labels, no pointing
          line is drawn either.
```

```
pie(table(Type))
```



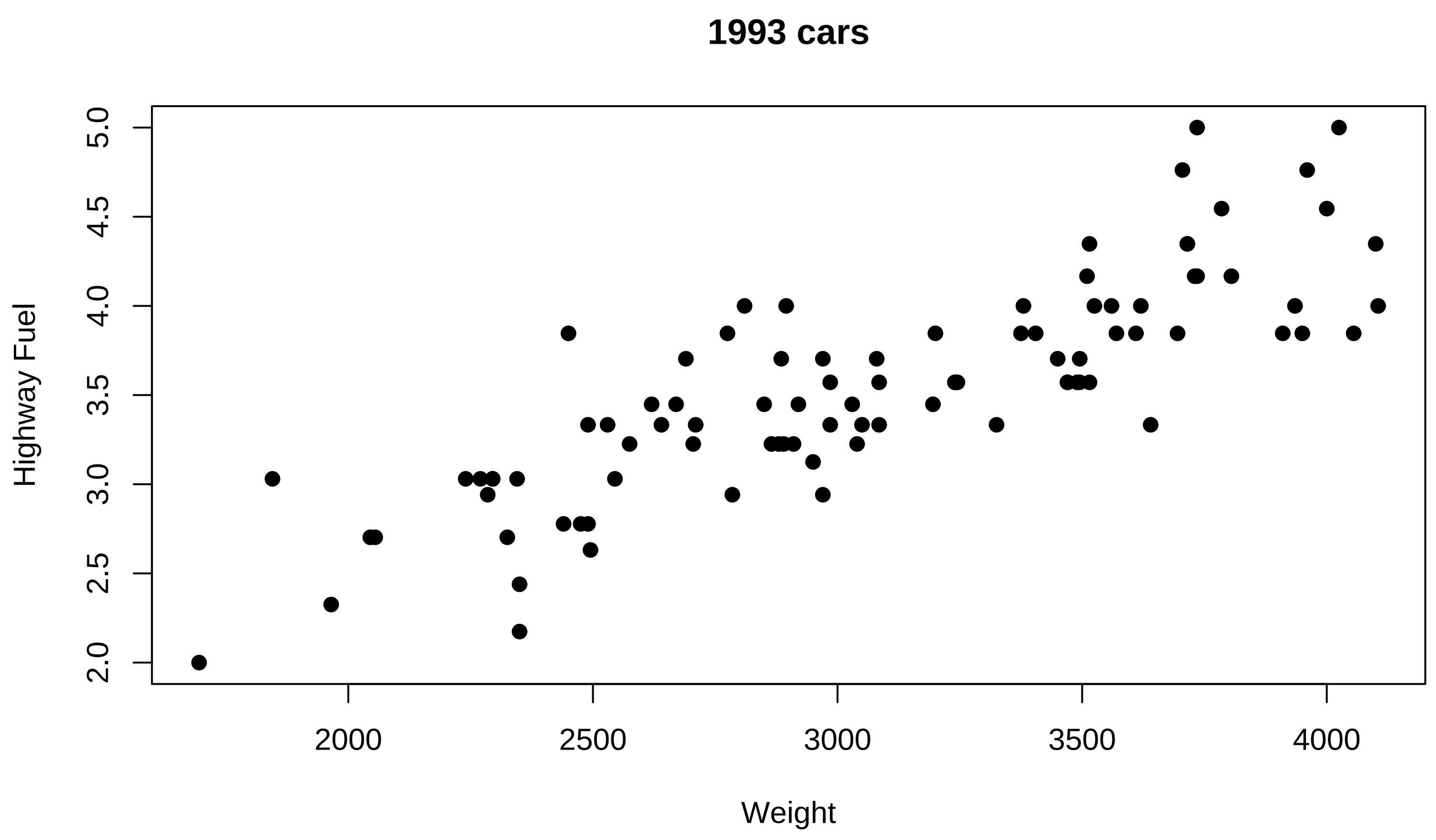
• Scatterplots

One of the many uses of the `plot` function.

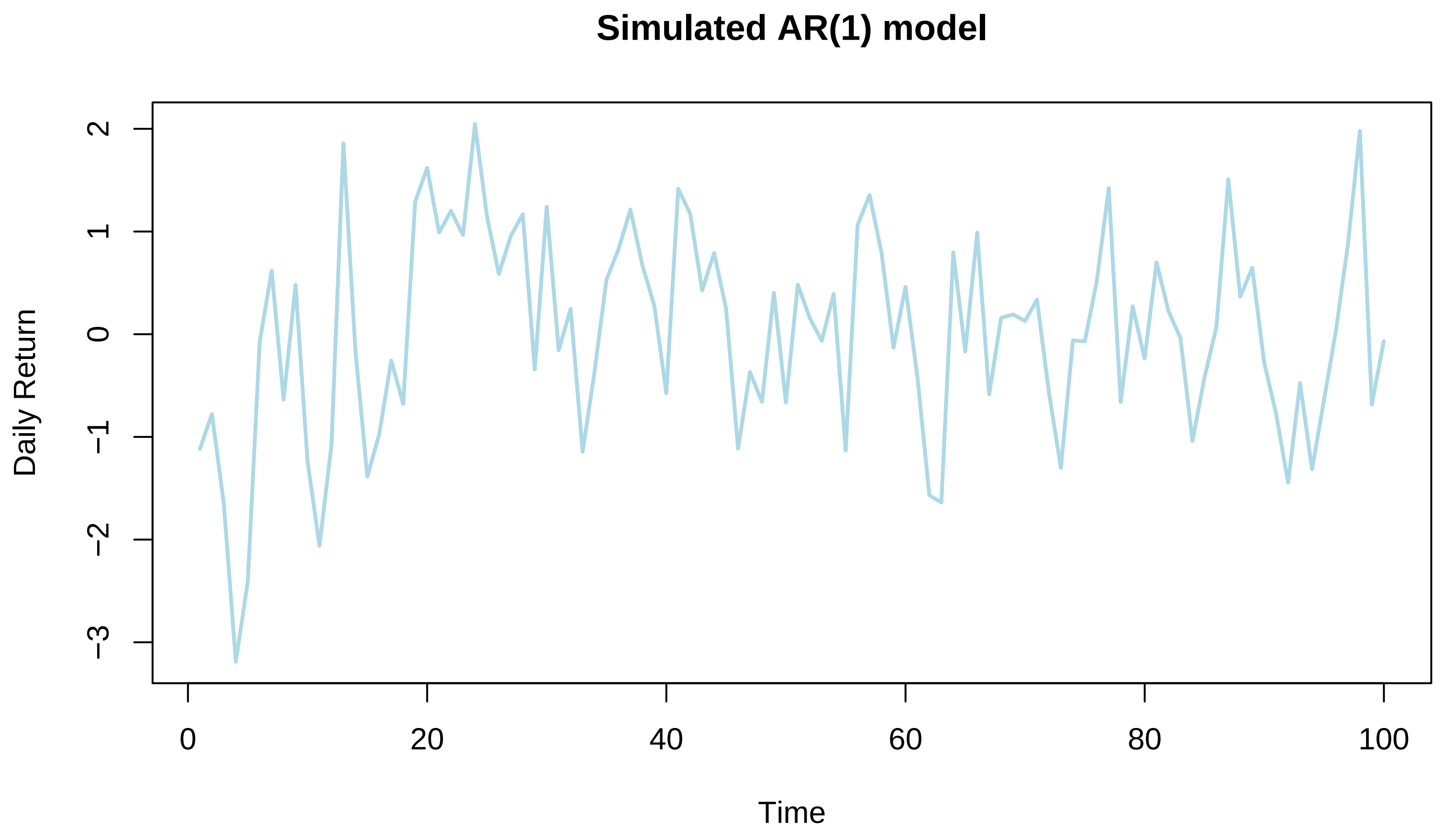
```
plot(x, y, ...)

Arguments:

       x: the coordinates of points in the plot. Alternatively,
          a single plotting structure, function or _any R object
          with a 'plot' method_ can be provided.

       y: the y coordinates of points in the plot, _optional_
          if 'x' is an appropriate structure.

     ...: graphical parameters can be given as arguments to
          'plot'.  Many methods will also accept the following
          arguments:
```

```
type: what type of plot should be drawn.  Possible types are

        *  ’"p"’ for *p*oints,

        *  ’"l"’ for *l*ines,

        *  ’"b"’ for *b*oth,

        *  ’"c"’ for the lines part alone of ’"b"’,

        *  ’"o"’ for both "*o*verplotted",

        *  ’"h"’ for "*h*istogram" like (or "high-density")
           vertical lines,

        *  ’"s"’ for stair *s*teps,

        *  ’"S"’ for other *s*teps, see _Details_ below,
        * '"n"' for no plotting.

     All other 'type's give a warning or an error; using,
     e.g., 'type = "punkte"' being equivalent to
     'type = "p"' for S compatibility.
```

There are many other options available for the default method. See `help(plot.default)` for more info (I'll also talk about some of them later when I discuss the `par` function). We've seen some examples of scatterplots already. Here are a couple more.

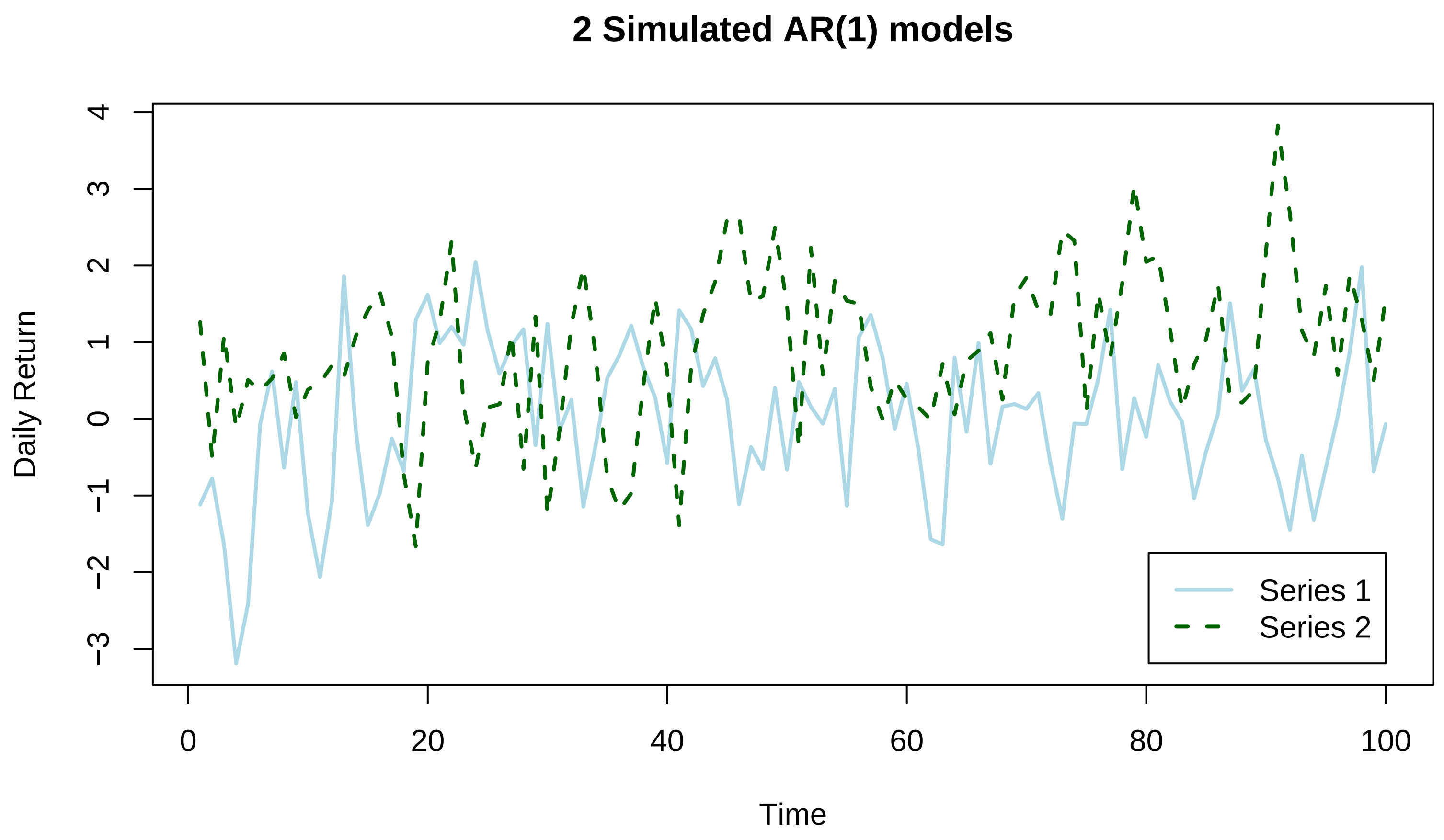


```
par(mar=c(5,4,4,1)+0.1, mfrow=c(1,1))
plot(Weight,HighFuel, pch=16, xlab="Weight", ylab="Highway Fuel",
  main="1993 cars")
```



```
par(mar=c(5,4,4,1)+0.1, mfrow=c(1,1))
plot(times,ret, type="l", xlab="Time", ylab="Daily Return",
  main="Simulated AR(1) model", col="lightblue", lwd=2)
```

In **S**, it is easy to build complicated plots, by adding pieces together. One example, is

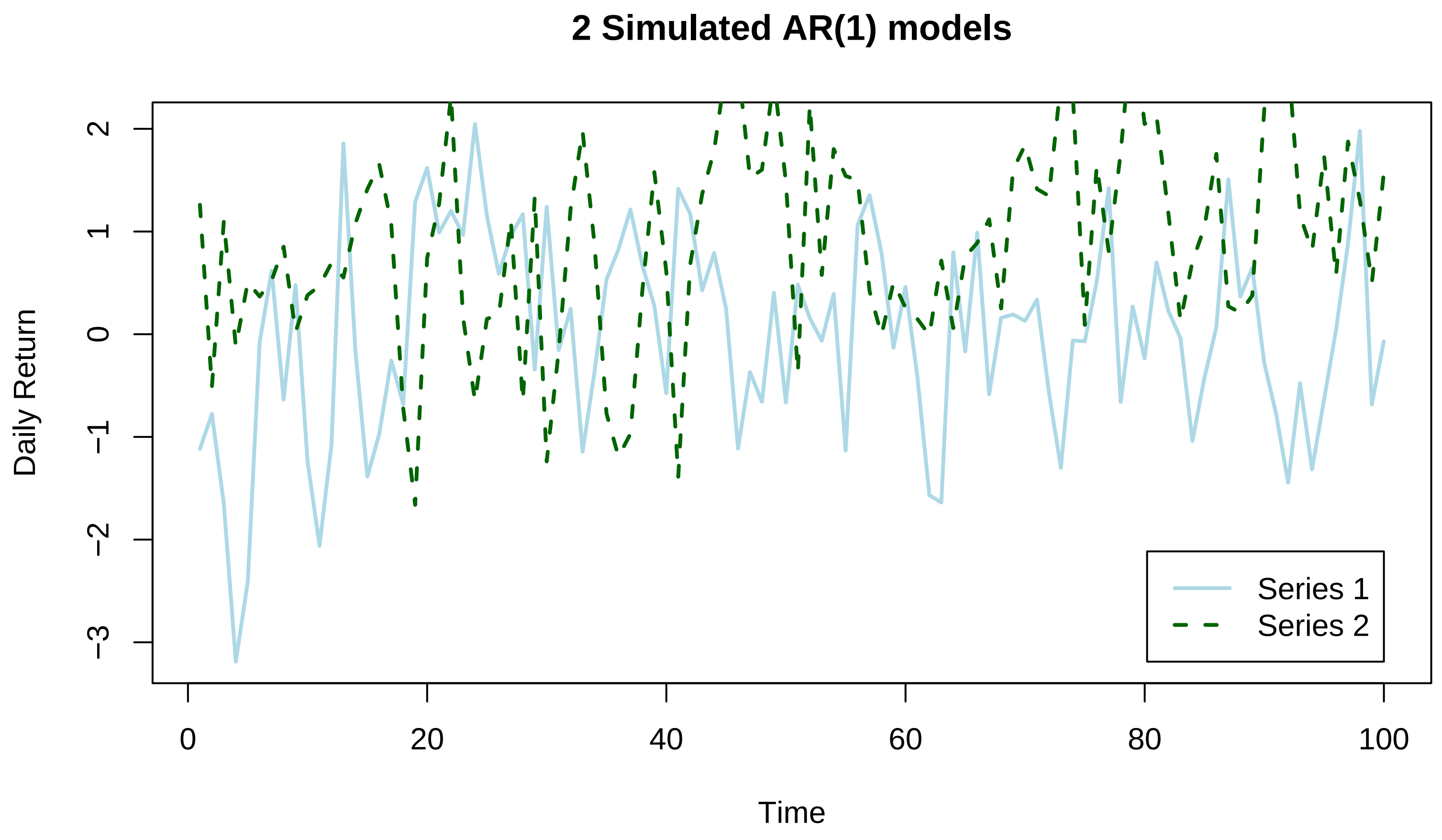


The commands for this plot are as follows.

```
plot(times, ret, type="l", xlab="Time", ylab="Daily Return",
  main="2 Simulated AR(1) models", col="lightblue", lwd=2,
  ylim=c(min(c(ret,ret2)), max(c(ret, ret2))))
lines(times, ret2, col="darkgreen", lty=2, lwd=2) legend(100,
min(c(ret,ret2)), c("Series 1", "Series 2"),
  xjust=1, yjust=0, col=c("lightblue", "darkgreen"),
  lty=1:2, lwd=2)
```

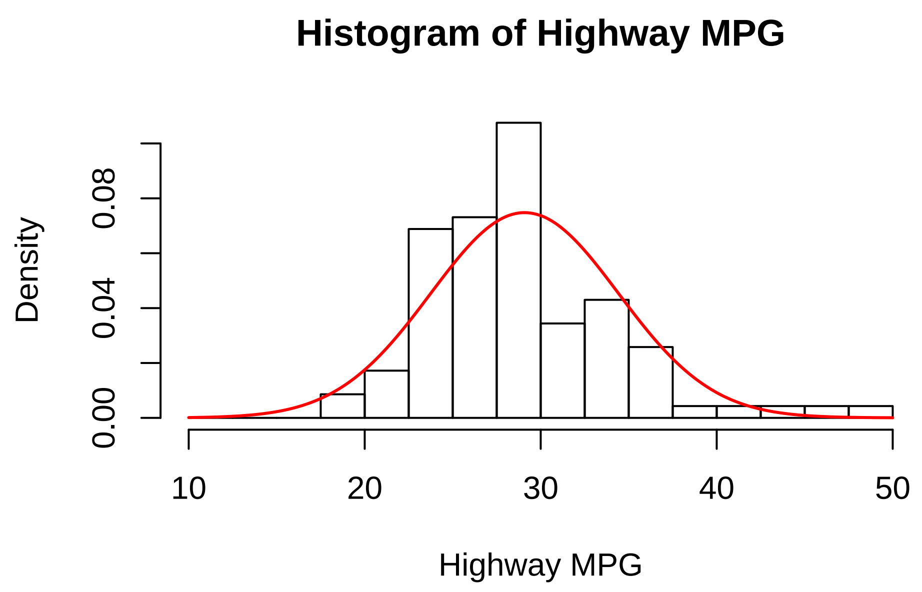
Note that when combining different elements together, you need to make sure that they fit.

When originally putting the figure together, parts of series 2 were clipped from the plot since the higher values were outside the range of series 1.



In the better version, I adjusted the range of the y-axis to match the range of both series. If this is not done, **S** bases the ranges of the x and y axes based on the inputted data.

In addition to adding elements to a graph created by `plot`, it can be done to other plot types such as histograms or boxplots. For example, a plot of a line can be superimposed onto a histogram.

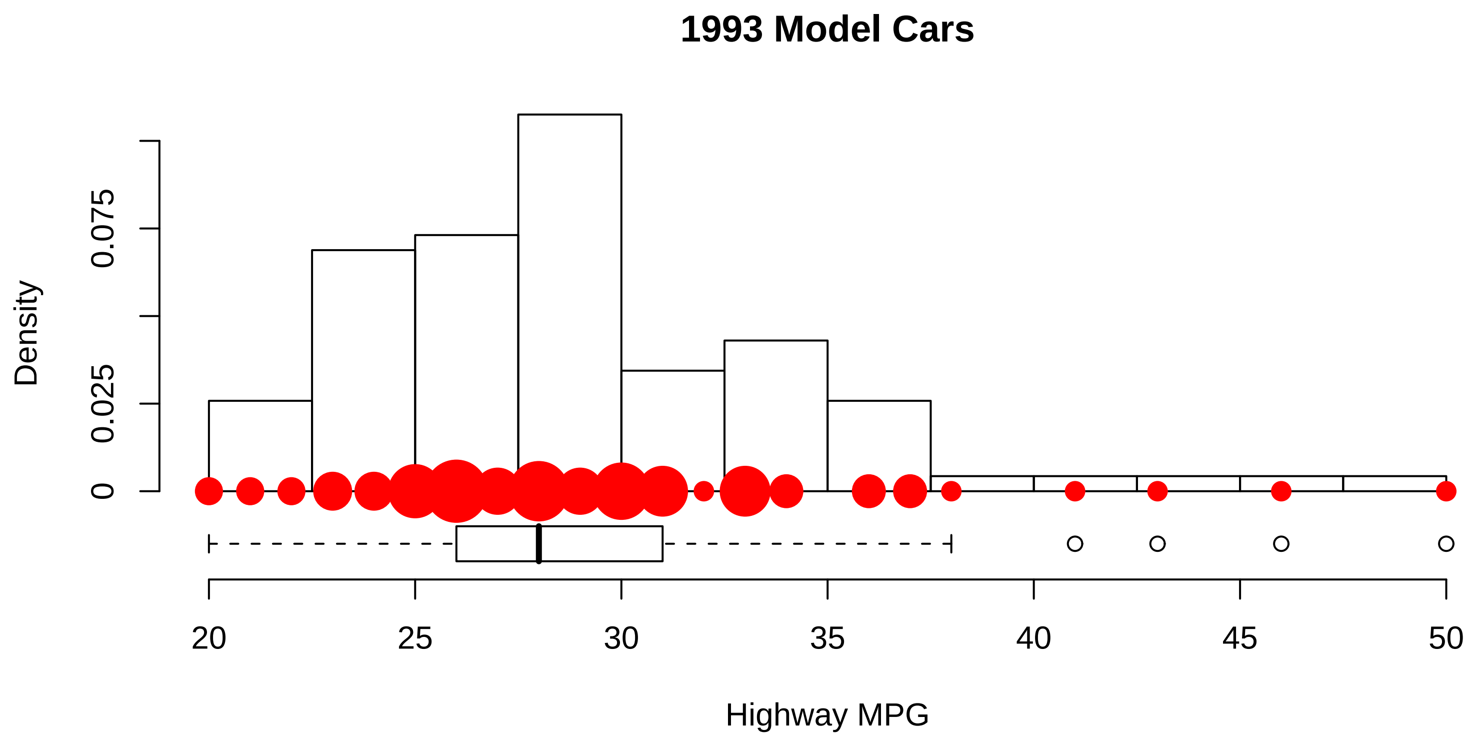


In the plot on the right, using the `prob=T` option in the `hist` function makes it much easier to get the height of the normal curve with the same mean and standard deviation as the data correct. Otherwise, you need to figure out the scale factor which depends on the number of observations in the vector and the width of each bin.

```
hist(HighMPG, prob=T, breaks=seq(10,50,by=2.5), xlab="Highway MPG",
  main="Histogram of Highway MPG", xlim=c(10,50))
xpts <- (100:500)/10 ; ypts <- dnorm(xpts, mean(HighMPG), sqrt(var(HighMPG)))
lines(xpts, ypts, lwd=1.5, col=2)
```

In the previous examples, a single high-level plotting function (e.g. plot, histogram, boxplot, barplot, etc) has been combined with one or more lower-level plotting function (`points`, `lines`, `abline`, etc).

A high-level plotting function is one that, as a default, creates a new figure. A low-level function add elements to the current figure. It is possible however, to combine two or more, high-level plots into a single figure. One example is the following, where a histogram is combined with a box-plot. The key element is the `add=T` option of most high level plot functions. The default for these functions is `add=F`. The option is not available in `plot` so if it is to be combined with another high-level plot function, call `plot` first. For a low-level plot function, while the option doesn't exist, it always acts like `add=T`.



```
highmpg<-cbind(table(cars93$HighMPG),sort(unique(cars93$HighMPG)))
hist(HighMPG, breaks=seq(20,50,by=2.5), prob=T, xlab="Highway MPG",
  main="1993 Model Cars", ylim=c(-0.02,0.11), yaxt="n")
symbols(highmpg[,2], 0*highmpg[,2], circles=sqrt(highmpg[,1]*0.05),
  add=T, inches=F, fg=2, bg=2)
boxplot(HighMPG, add=T, axes=F, at=-0.015, boxwex=0.02, horizontal=T)
axis(2,(0:4)*0.025, (0:4)*0.025)
```

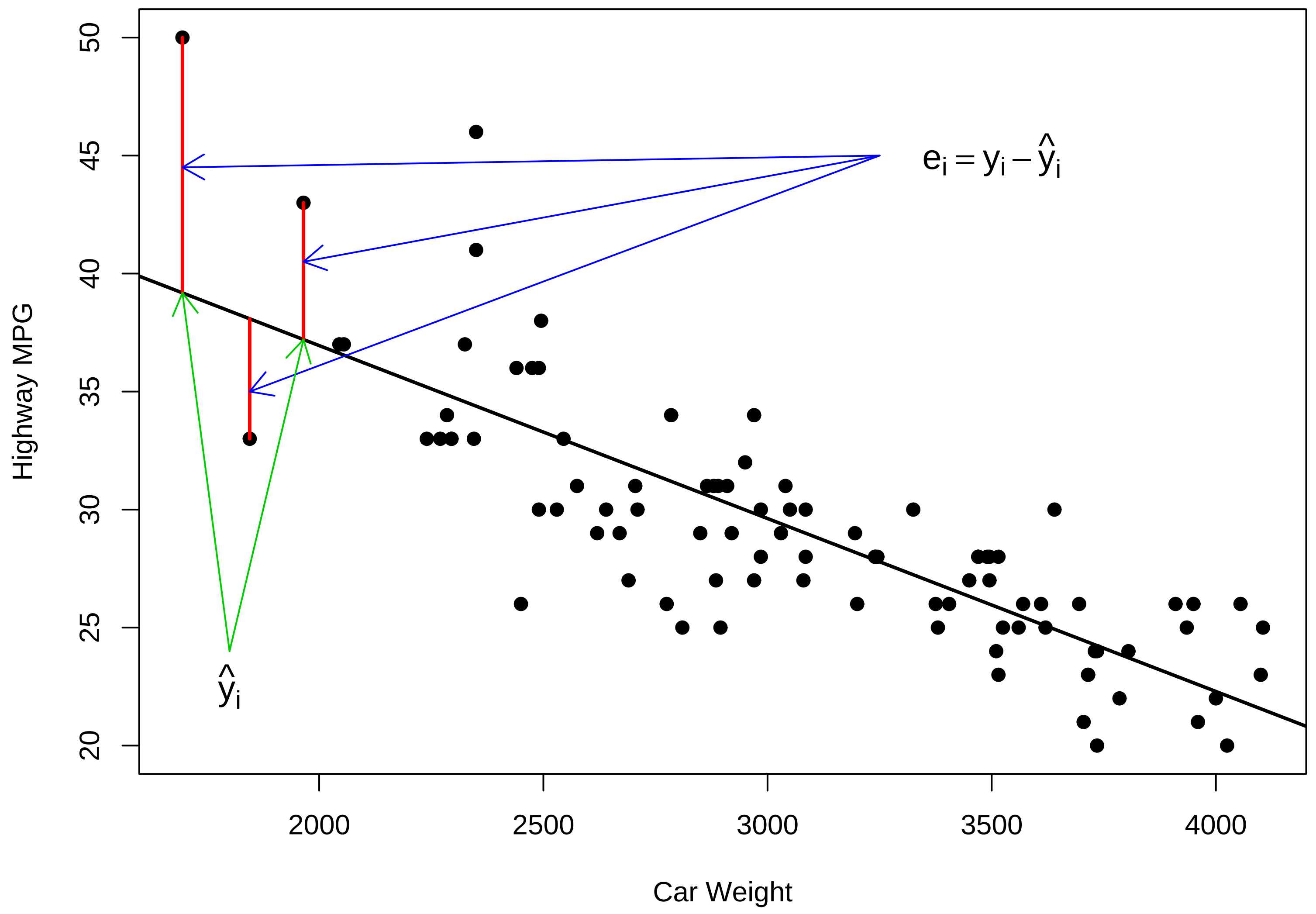
Where possible, try to use low-level plot functions to add elements to a figure. Usually there are less options to set when working with low-level plot functions, making thing easier. Also a later example will show how you can get into trouble but using `plot` twice in a single figure.

It is also possible to annotate figures. Useful functions for doing this are `legend`, `text` (add text at desired locations), `segments` (draws line segments), and `arrows` (draws arrows from one point to another).

An example, combining many of these elements is the following.

One other point to note on with this plot, is that the plot points were indicted with a formula, not vectors of the x and y coordinates, as was done in one of the boxplot examples.

```
carfit <- lsfit(Weight, HighMPG)
weights <- c(1695, 1845, 1965)
xmat <- matrix(c(1,1,1,weights), byrow=F, ncol=2)
carfitted <- xmat%*%matrix(carfit$coef, ncol=1)
plot(HighMPG ~ Weight, cars93, pch=16, xlab="Car Weight",
  ylab="Highway MPG")
abline(carfit, lwd=2)
segments(weights,obs,weights,carfitted,col=2, lwd=2)
arrows(3250,45,1695,44.5,col=4,length=0.15)
arrows(3250,45,1965,40.5,col=4,length=0.15)
arrows(3250,45,1845,35,col=4,length=0.15)
arrows(1800,24,1695,carfitted[1],col=3,length=0.15)
arrows(1800,24,1965,carfitted[3],col=3,length=0.15)
text(3500,45,expression(e[i] == y[i] - hat(y)[i]),cex=1.25)
text(1800,22.5,expression(hat(y)[i]),cex=1.25)
```


$e_i = y_i - \hat{y}_i$
$\hat{y}_i$
Highway MPG
Car Weight
20
25
30
35
40
45
50
2000
2500
3000
3500
4000

Plot Options

Many plotting options can be set within the plotting functions. Examples seen so far have been

- `xlab`, `ylab`: Axis labels
- `main`: Plot title
- `xlim`, `ylim`: Axis limits
- `pch`: Plotting symbols (see `help(points)` for more details

Now these options are from high-level plotting functions. Many are available for all of these functions (like `main`). Others may be more command specific (`pch` which only is appropriate for `plot` - though used in some low-level functions as well).

There are a wide range of other plotting options, while available to the functions discussed so far, can be used in other ways and are not specific to certain functions. These additions options are part of the `par` function.

There are two types of options that can be set via `par`.

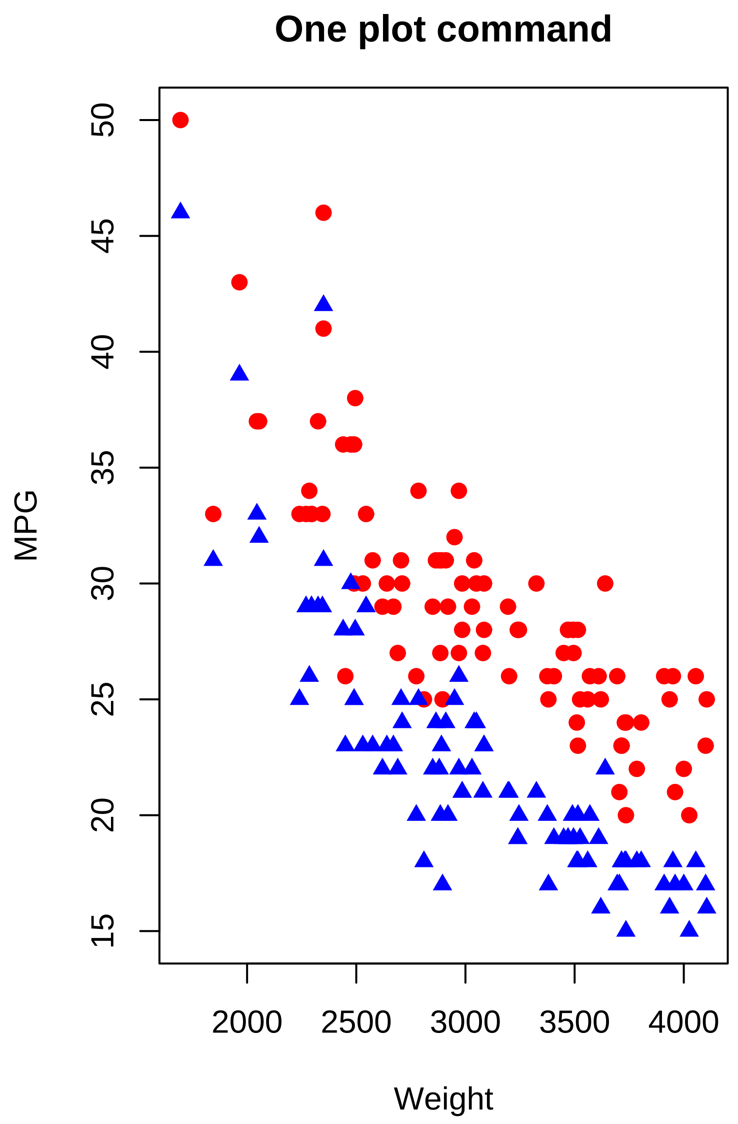
- Global plot features
  - Plot layout - `mfrow` or `mfcol` (sets number of rows and columns
  - Margins - `mar` or `mai`
- Item specific features
  - Colour - `col`, `fg`, or `bg`
  - Line types - `lty`
  - Line widths - `lwd`
  - Fonts - `cex` (sets font size), `font` (sets type face), or `family` (font family). Note that `cex` (and other `par` parameters) has methods for axis, lab, main, and sub for more specific adjustment of features.
  - Axes - `xaxp`, `xaxs`, `xaxt`, `xlog`. Similar for y axis.

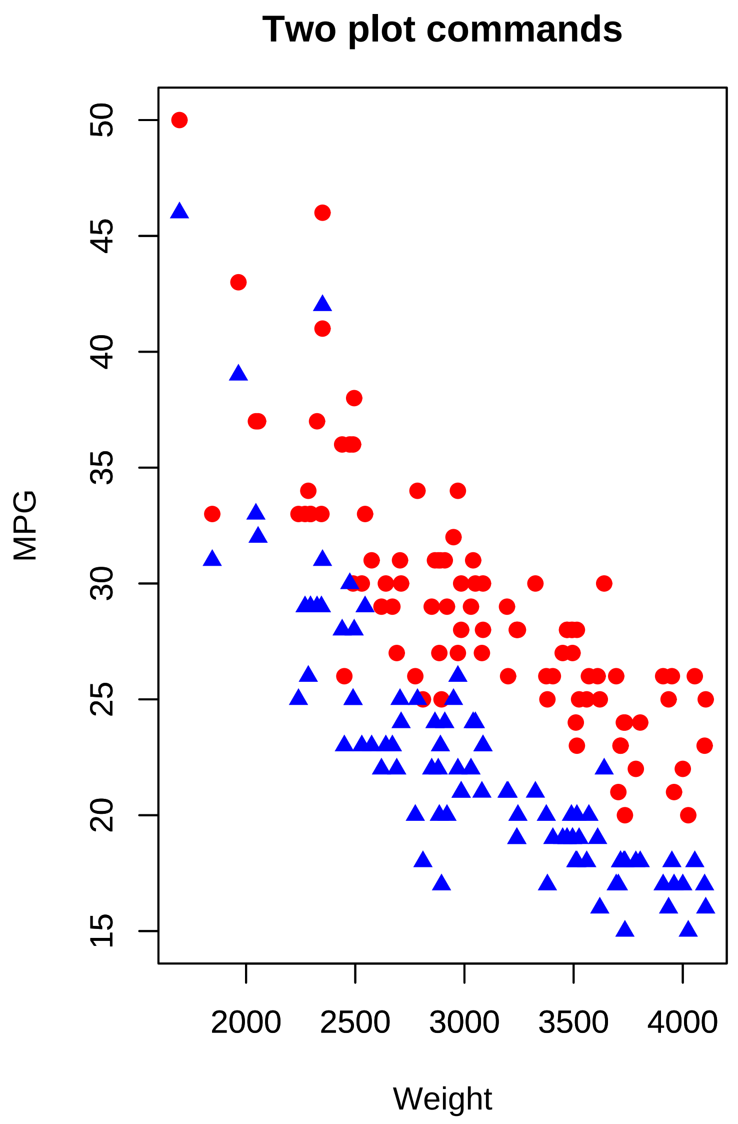
The global plot features must be set before plotting by the `par` command.

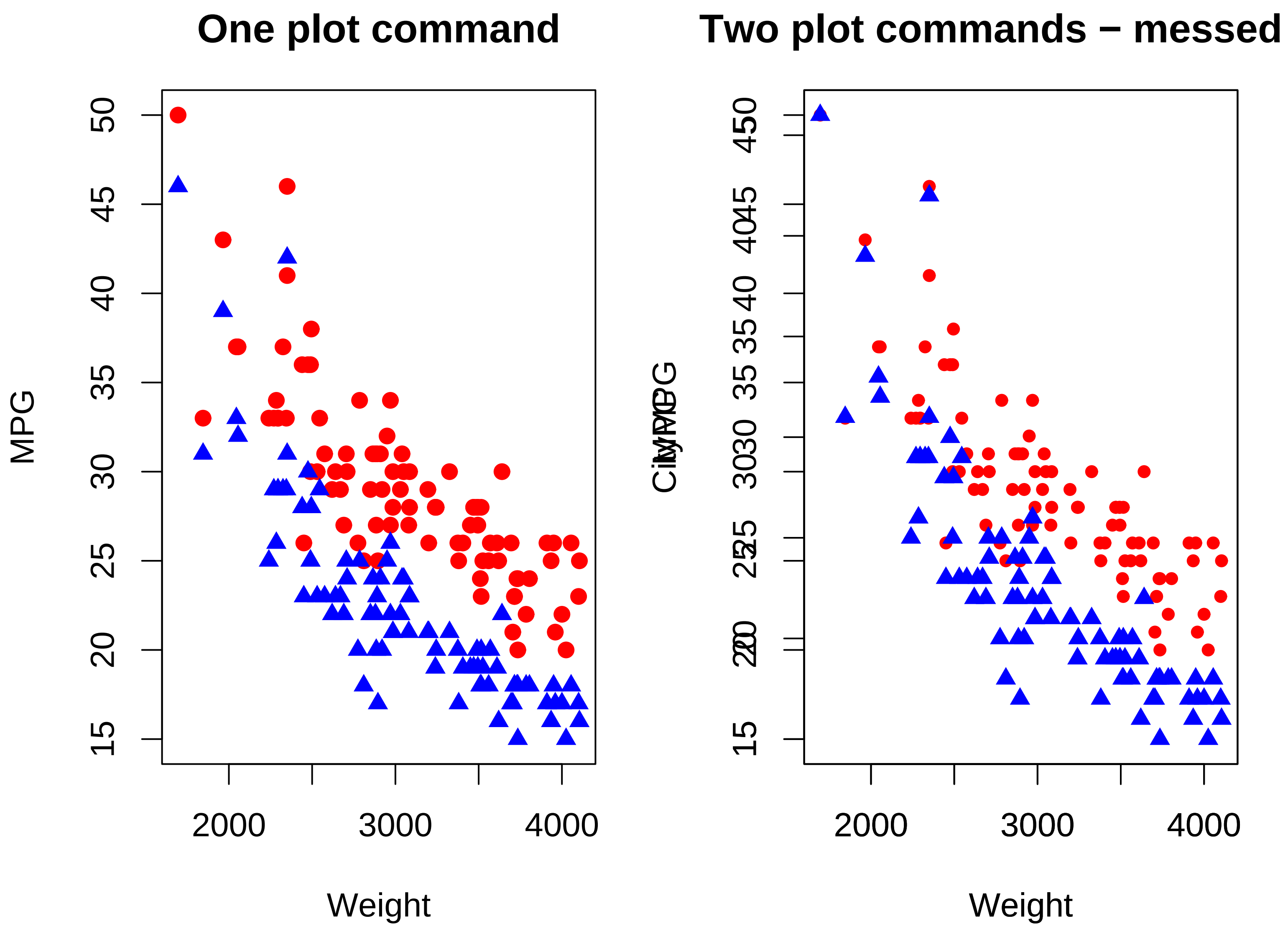
The item specific features may be set within a plotting function, or can be set by the `par` function to reset a default.

For example,

```
postscript('../lowvshigh.eps', horiz=F, width=9,height=6)
par(mfrow=c(1,2), pty="m", mar=c(4,4,3,1)+0.1)

plot(Weight,HighMPG, pch=16, col=2, xlab="Weight", ylab="MPG",
  main="One plot command", ylim=c(15,50))
points(Weight,CityMPG, pch=17, col=4)  # col="blue"
plot(Weight,HighMPG, pch=16, col=2, xlab="Weight", ylab="MPG",
  main="Two plot commands", ylim=c(15,50))
par(new=T) plot(Weight,CityMPG, pch=17, col=4, xlab="", ylab="",
  main="", ylim=c(15,50))
dev.off()
```


One plot command
MPG
Weight
15
20
25
30
35
40
45
50
2000
2500
3000
3500
4000


Two plot commands
MPG
Weight
15
20
25
30
35
40
45
50
2000
2500
3000
3500
4000


One plot command
Two plot commands − messed
MPG
City MPG
Weight
Weight
2000
3000
4000
15
20
25
30
35
40
45
50

```
postscript('../lowvshigh2.eps', horiz=F, width=9,height=6)
par(mfrow=c(1,2), pty="m", mar=c(4,4,3,1)+0.1)
par(cex=1.25) # increase text and symbols by 25%

plot(Weight,HighMPG, pch=16, col=2, xlab="Weight", ylab="MPG",
  main="One plot command", ylim=c(15,50))
points(Weight,CityMPG, pch=17, col=4)
plot(Weight,HighMPG, pch=16, col=2, xlab="Weight", ylab="MPG",
  main="Two plot commands - messed up", ylim=c(15,50), cex=0.75)
par(new=T)
plot(Weight,CityMPG, pch=17, col=4)
dev.off()
```

To see what all the par settings are give the `par()`. Specific setting can also be observed by passing the desired option to the function.

```
> par("cex", "mfrow")
$cex [1] 1

$mfrow [1] 1 2

> par()
$xlog [1] FALSE

$ylog [1] FALSE

$adj [1] 0.5

$ann [1] TRUE

$ask [1] FALSE
```

blah, blah, blah finally getting to the end

```
$xaxt [1] "s"

$xpd [1] FALSE

$yaxp [1] 15 50  7

$yaxs [1] "r"

$yaxt [1] "s"
```