# Graphics - Part I: Basic Graphics

Statistics 135

Autumn 2005

# Basic Graphics

Want to focus today on standard graphics in **S**.

Look at standard plots such as histograms, stem & leaf, scatter plots, box plots, bar charts, etc. (Welcome back to Stat 100)

As part of this, we need to discuss **S**graphics devices - where the figure will get plotted. Need to consider on screen and file possibilities.

Next time we will discuss trellis graphics, which can be extremely useful for exploratory data analysis.

# Common Plots

• Stem & Leaf - A simple summary for a single quantitative variable.

This is one of the few (maybe only) graphic which is text based. Most other graphics in **S** are high resolution. Not often used, except as a quick and dirty summary by paper and pencil.

```
stem(x, scale = 1, width = 80, atom = 1e-08)
```

```
Arguments:

      x: a numeric vector.

  scale: This controls the plot length.

  width: The desired width of plot.
```

```
> stem(EngSize)

  The decimal point is at the |

  1 | 02333
  1 | 55555556666688888889
  2 | 000012222222222333333444
  2 | 5555888
  3 | 000000000002233444
  3 | 5588888888
  4 | 3
  4 | 56669
  5 | 0
  5 | 77
```

The line "4 | 3" corresponds to a single observation rounded to 4.3.

```
> stem(EngSize,scale=2)
  The decimal point is 1 digit(s) to the left of the |
  10 | 0
  12 | 0000
  14 | 0000000
  16 | 00000
  18 | 00000000
  20 | 00000
  22 | 0000000000000000
  24 | 0000000
  26 |
  28 | 000
  30 | 00000000000
  32 | 0000
  34 | 00000
  36 |
  38 | 00000000
  40 |
  42 | 0
  44 | 0
  46 | 000
  48 | 0
  50 | 0
  52 |
  54 |
  56 | 00
```

- Histograms

Usually used instead of stem & leaf plots.

```
hist(x, breaks = "Sturges", freq = NULL, probability = !freq,
          include.lowest = TRUE, right = TRUE,
          density = NULL, angle = 45, col = NULL, border = NULL,
          main = paste("Histogram of" , xname),
          xlim = range(breaks), ylim = NULL,
          xlab = xname, ylab,
          axes = TRUE, plot = TRUE, labels = FALSE,
          nclass = NULL, ...)
```

```
Arguments:

       x: a vector of values for which the histogram is desired.
```

breaks: one of:

* a vector giving the breakpoints between histogram cells,

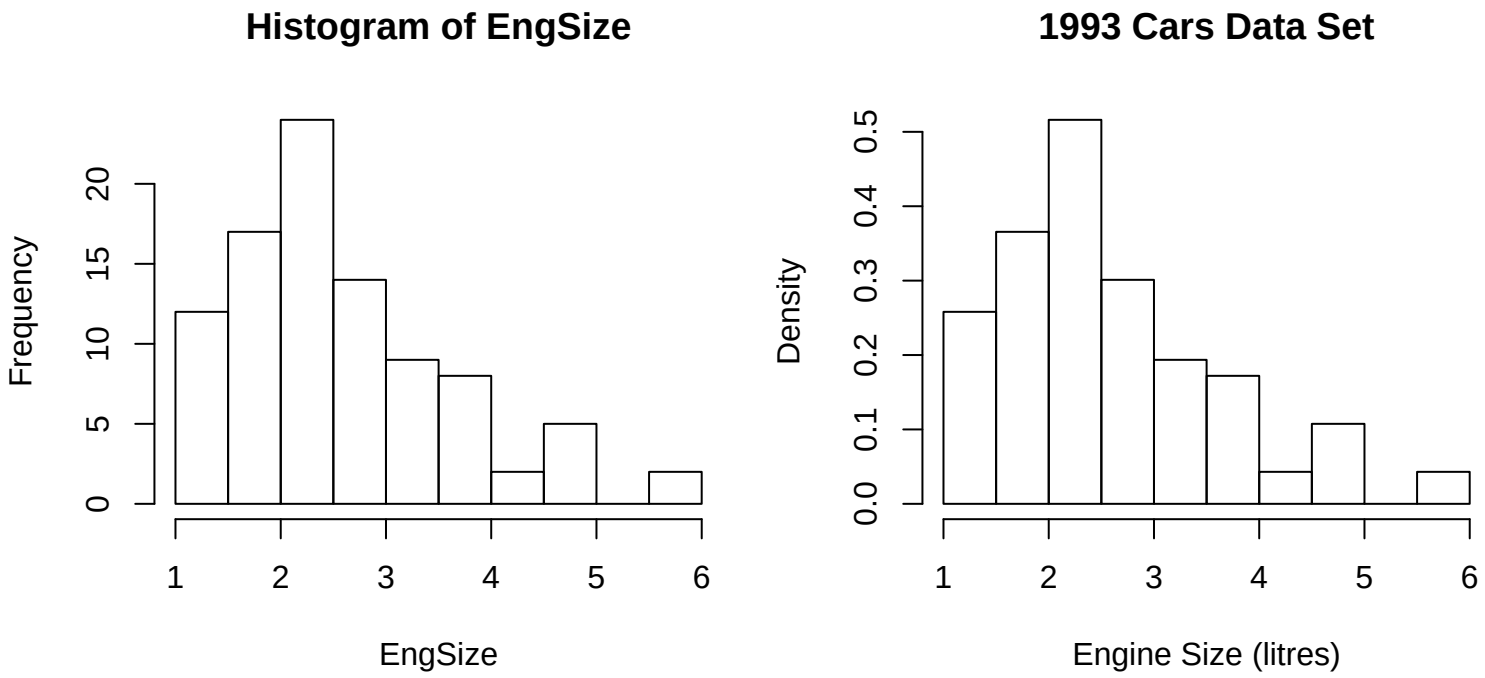
* a single number giving the number of cells for the histogram,

* a character string naming an algorithm to compute the number of cells (see Details),

* a function to compute the number of cells.

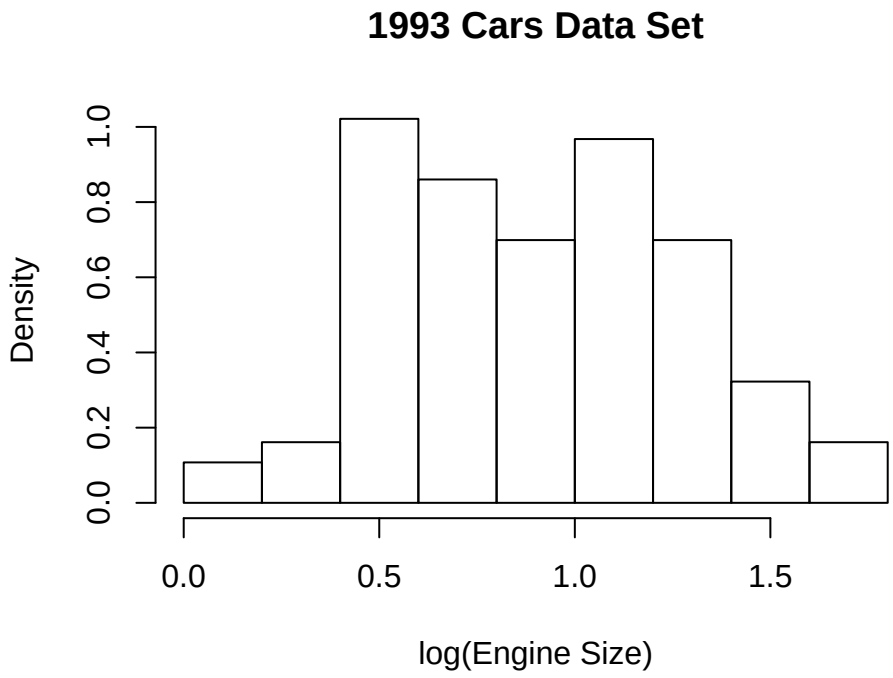
In the last three cases the number is a suggestion only.

```
    freq: logical; if 'TRUE', the histogram graphic is a
          representation of frequencies, the 'counts' component
          of the result; if 'FALSE', probability densities,
          component 'density', are plotted (so that the histogram
          has a total area of one).  Defaults to 'TRUE' _iff_
          'breaks' are equidistant (and 'probability' is not
          specified).

probability: an _alias_ for '!freq', for S compatibility.
```




```
par(mfrow=c(1,2))        # want 1 row, 2 columns in figure layout

# left plot
hist(EngSize)

# right plot
hist(EngSize, prob=T, main="1993 Cars Data Set", xlab="Engine Size (litres)")
```

It is possible to use functions of variables in plotting commands. You don't need to save a transformed version of the variable on plot with that. In this case, you will often want to set the axis labels.




```
par(mar=c(5,4,4,1)+0.1)
par(mfrow=c(1,1))
hist(log(EngSize), prob=T, main="1993 Cars Data Set",
  xlab="log(Engine Size)")
```

- Box plots

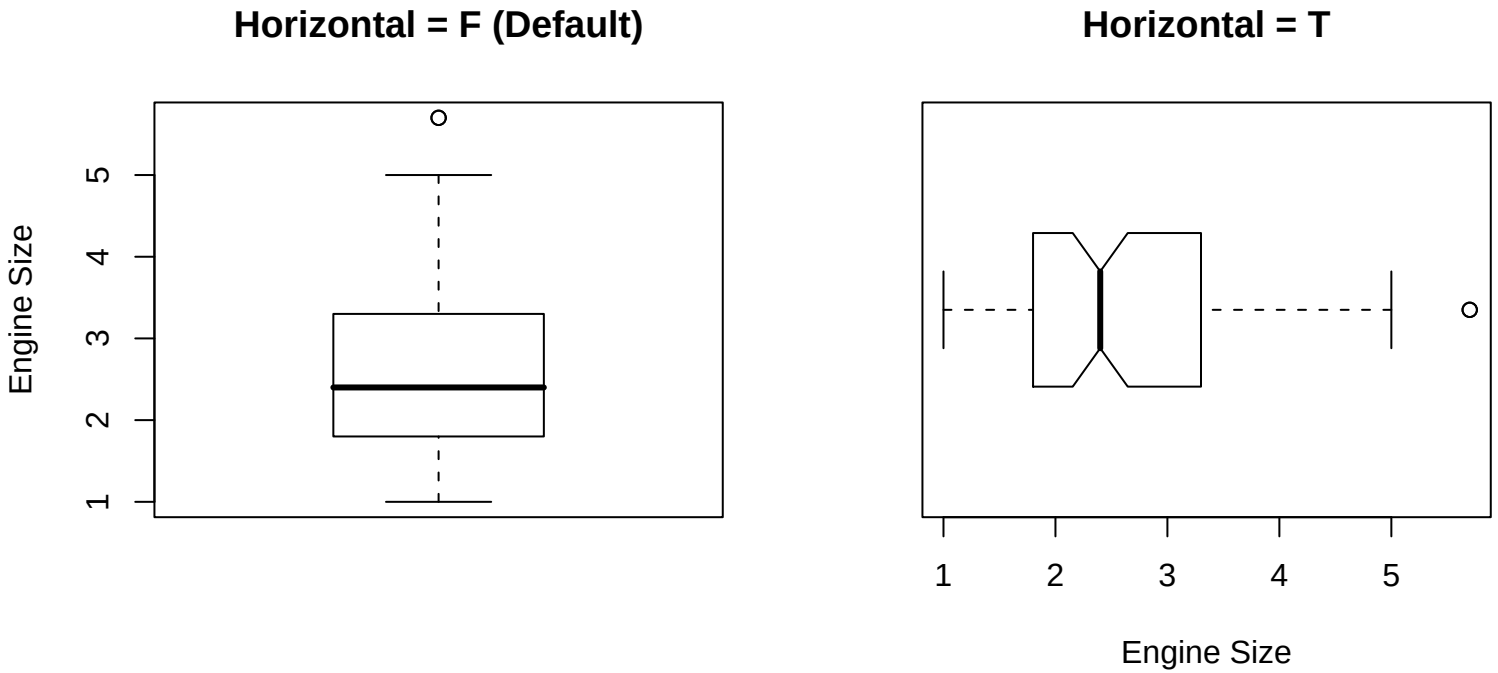
For the `boxplot` function, there are two version of the function, one of more use for looking at a single variable, and the other for looking a variable stratified by the levels of a categorical variable. Lets consider the first case.

```
boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE,
            notch = FALSE, outline = TRUE, names, plot = TRUE,
            border = par("fg"), col = NULL, log = "",
            pars = list(boxwex = 0.8, staplewex = 0.5,
            outwex = 0.5), horizontal = FALSE, add = FALSE,
            at = NULL)
```

Arguments:

x: for specifying data from which the boxplots are to be produced. Either a numeric vector, or a single list containing such vectors. Additional unnamed arguments specify further data as separate vectors (each corresponding to a component boxplot).

```
          'NA's are allowed in the data.

horizontal: logical indicating if the boxplots should be
          horizontal; default 'FALSE' means vertical boxes.

   names: group labels which will be printed under each boxplot.

      at: numeric vector giving the locations where the boxplots
          should be drawn, particularly when 'add = TRUE';
          defaults to '1:n' where 'n' is the number of boxes.
```

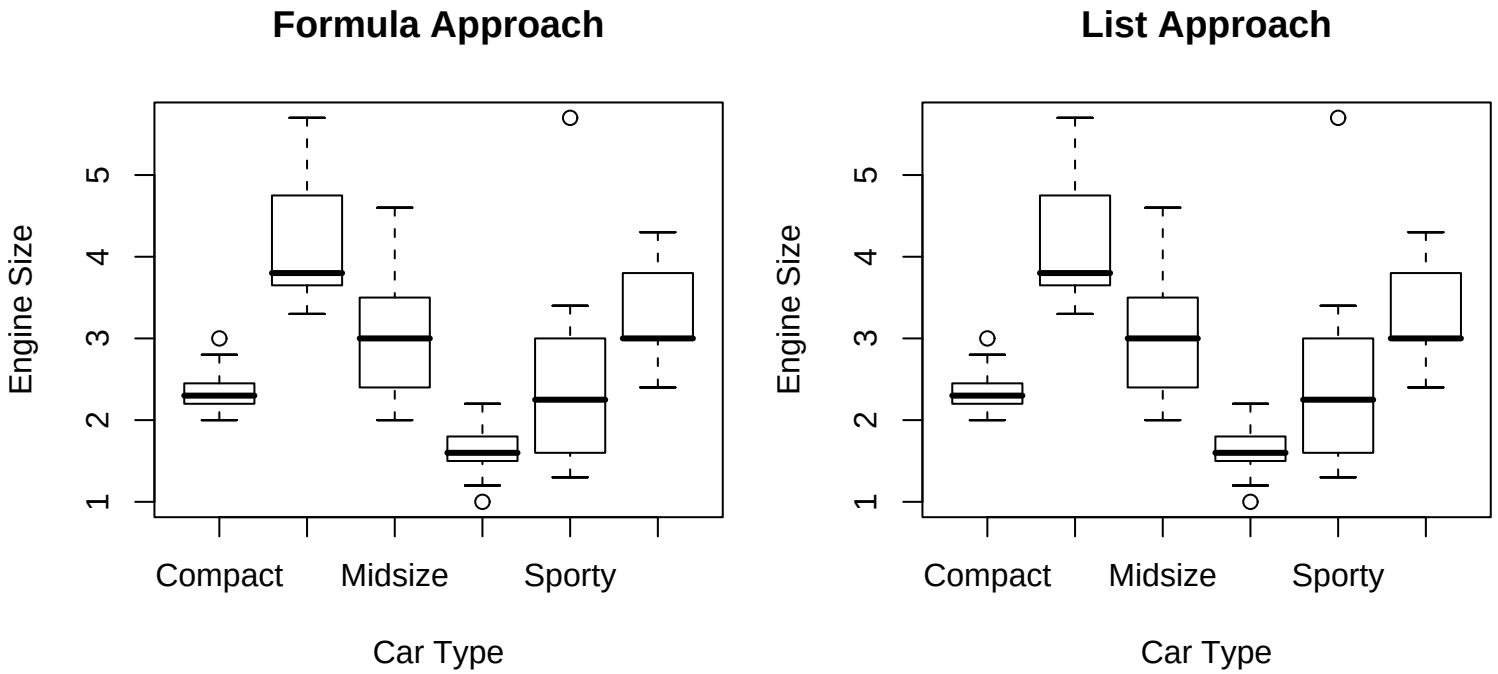



```
par(mfrow=c(1,2))
boxplot(EngSize, main="Horizontal = F (Default)", ylab="Engine Size")
boxplot(EngSize, main="Horizontal = T", xlab="Engine Size",
  horizontal=T, notch=T)
```

While the previous approach can be used to compare different groups, the section version of the `boxplot` function is more natural. It uses the idea of a formula, or model, we will see many times more in the course.

```
boxplot(formula, data = NULL, ..., subset, na.action = NULL)
```

```
Arguments:

 formula: a formula, such as 'y ~ grp', where 'y' is a numeric
          vector of data values to be split into groups according
          to the grouping variable 'grp' (usually a factor).

    data: a data.frame (or list) from which the variables in
          'formula' should be taken.

  subset: an optional vector specifying a subset of observations
          to be used for plotting.
```




```
# create a list splitting the EngSize by the Different Car Types
EngSize.Type <- split(EngSize, Type)
par(mar=c(5,4,4,1)+0.1, mfrow=c(1,2))
boxplot(EngSize ~ Type, data=cars93, main="Formula Approach",
  ylab="Engine Size", xlab="Car Type")
boxplot(EngSize.Type, main="List Approach", ylab="Engine Size",
  xlab="Car Type")
```

- Bar charts

Often the best way to plot categorical data

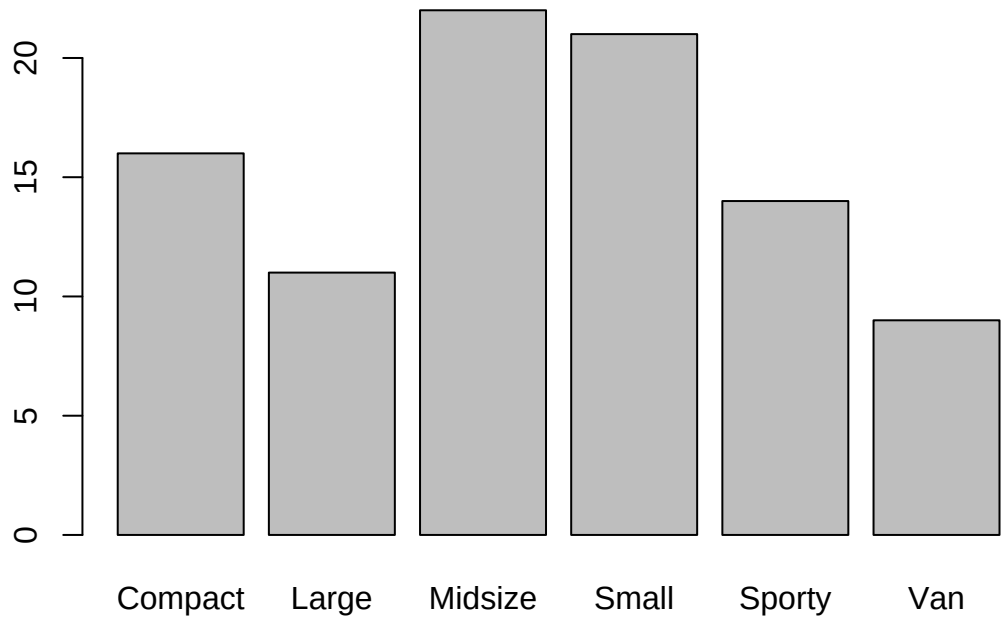
```
barplot(height, width = 1, space = NULL,
              names.arg = NULL, legend.text = NULL, beside = FALSE,
              horiz = FALSE, density = NULL, angle = 45,
              col = NULL, border = par("fg"),
              main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
              xlim = NULL, ylim = NULL, xpd = TRUE,
              axes = TRUE, axisnames = TRUE,
              cex.axis = par("cex.axis"),
              cex.names = par("cex.axis"), inside = TRUE,
              plot = TRUE, axis.lty = 0, offset = 0, ...)
```

Arguments:

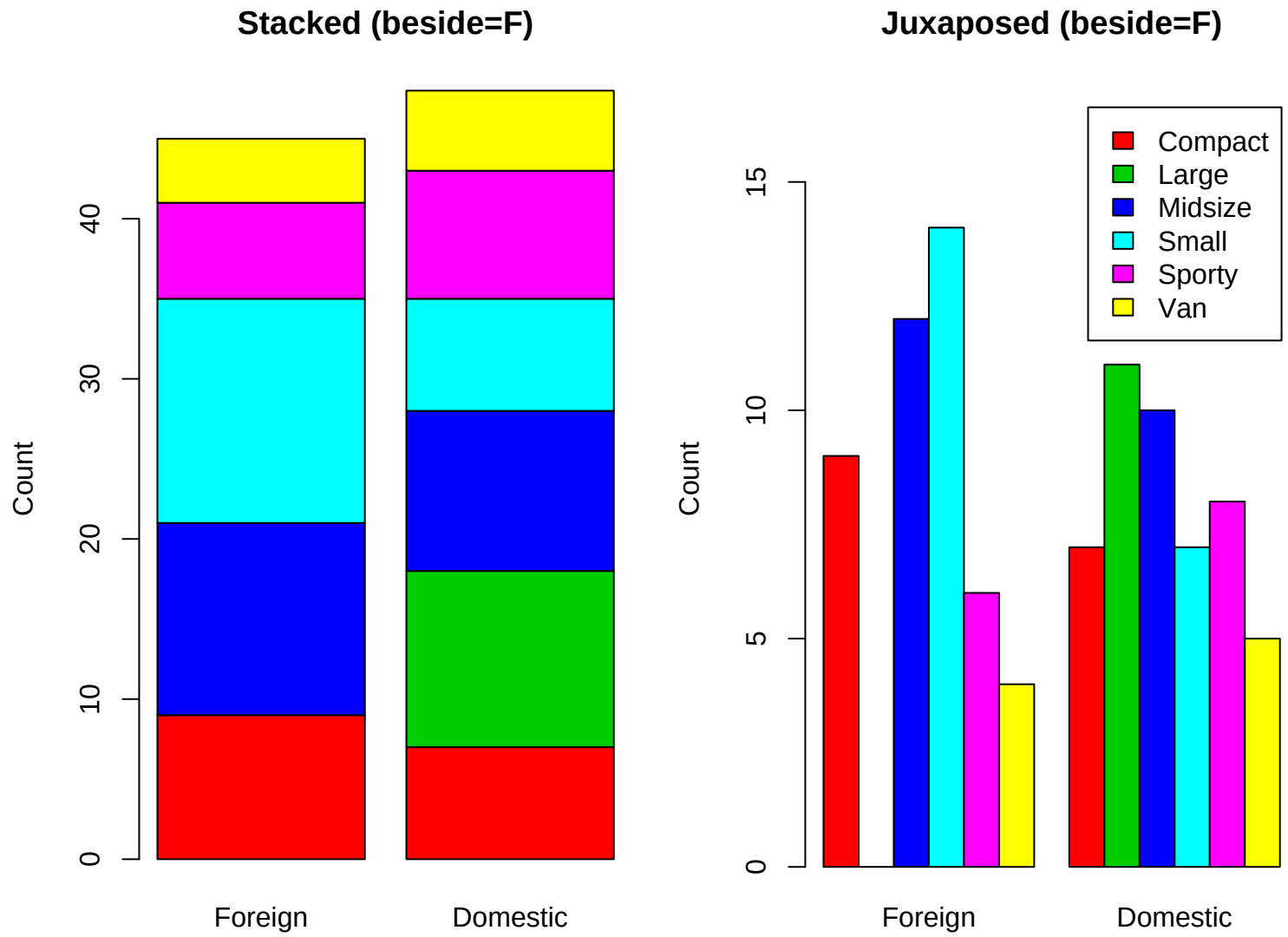
height: either a vector or matrix of values describing the bars which make up the plot. If 'height' is a vector, the plot consists of a sequence of rectangular bars with heights given by the values in the vector. If 'height' is a matrix and 'beside' is 'FALSE' then each bar of the plot corresponds to a column of 'height', with the values in the column giving the heights of stacked "sub-bars" making up the bar. If 'height' is a matrix and 'beside' is 'TRUE', then the values in each column are juxtaposed rather than stacked.

names.arg: a vector of names to be plotted below each bar or group of bars. If this argument is omitted, then the names are taken from the 'names' attribute of 'height' if this is a vector, or the column names if it is a matrix.

```
legend.text: a vector of text used to construct a legend for
        the plot, or a logical indicating whether a legend
        should be included.  This is only useful when 'height'
        is a matrix. In that case given legend labels should
        correspond to the rows of 'height'; if 'legend.text'
        is true, the row names of 'height' will be used as
        labels if they are non-null.

    col: a vector of colors for the bars or bar components. By
        default, grey is used if 'height' is a vector, and a
        gamma-corrected grey palette if 'height' is a matrix.
```




```
> table(Type)
Type
Compact   Large Midsize   Small  Sporty     Van
     16      11      22      21      14       9
> barplot(table(Type))
```


Stacked (beside=F)
Count
0
10
20
30
40
Foreign
Domestic
Juxaposed (beside=F)
Count
0
5
10
15
Compact
Large
Midsize
Small
Sporty
Van
Foreign
Domestic

```
td <- table(Type,Domestic)

par(mar=c(5,4,4,1)+0.1, mfrow=c(1,2))
barplot(td, ylab="Count", main="Stacked (beside=F)",
  names.arg=c("Foreign", "Domestic"),col=2:7)
barplot(td, ylab="Count", main="Juxaposed (beside=F)",
  names.arg=c("Foreign", "Domestic"),col=2:7,
  beside=T, ylim=c(0,17), legend.text=T)
```

- 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

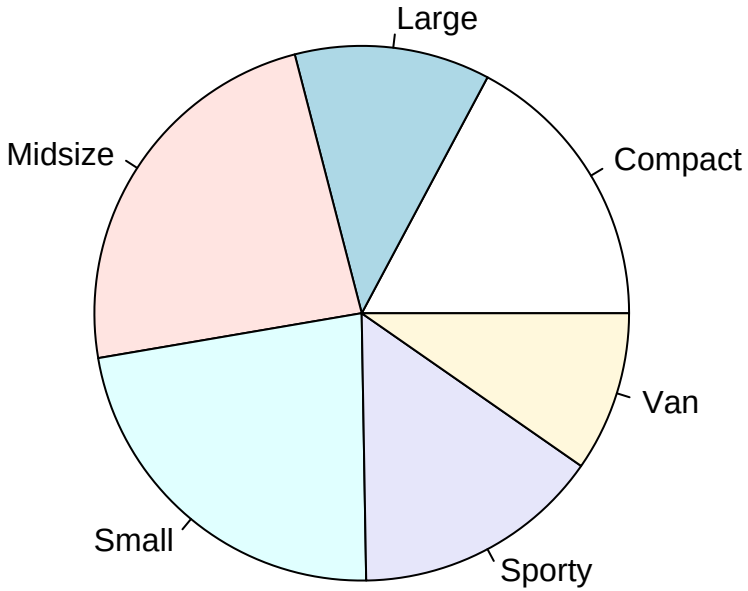
Usage:

```
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.

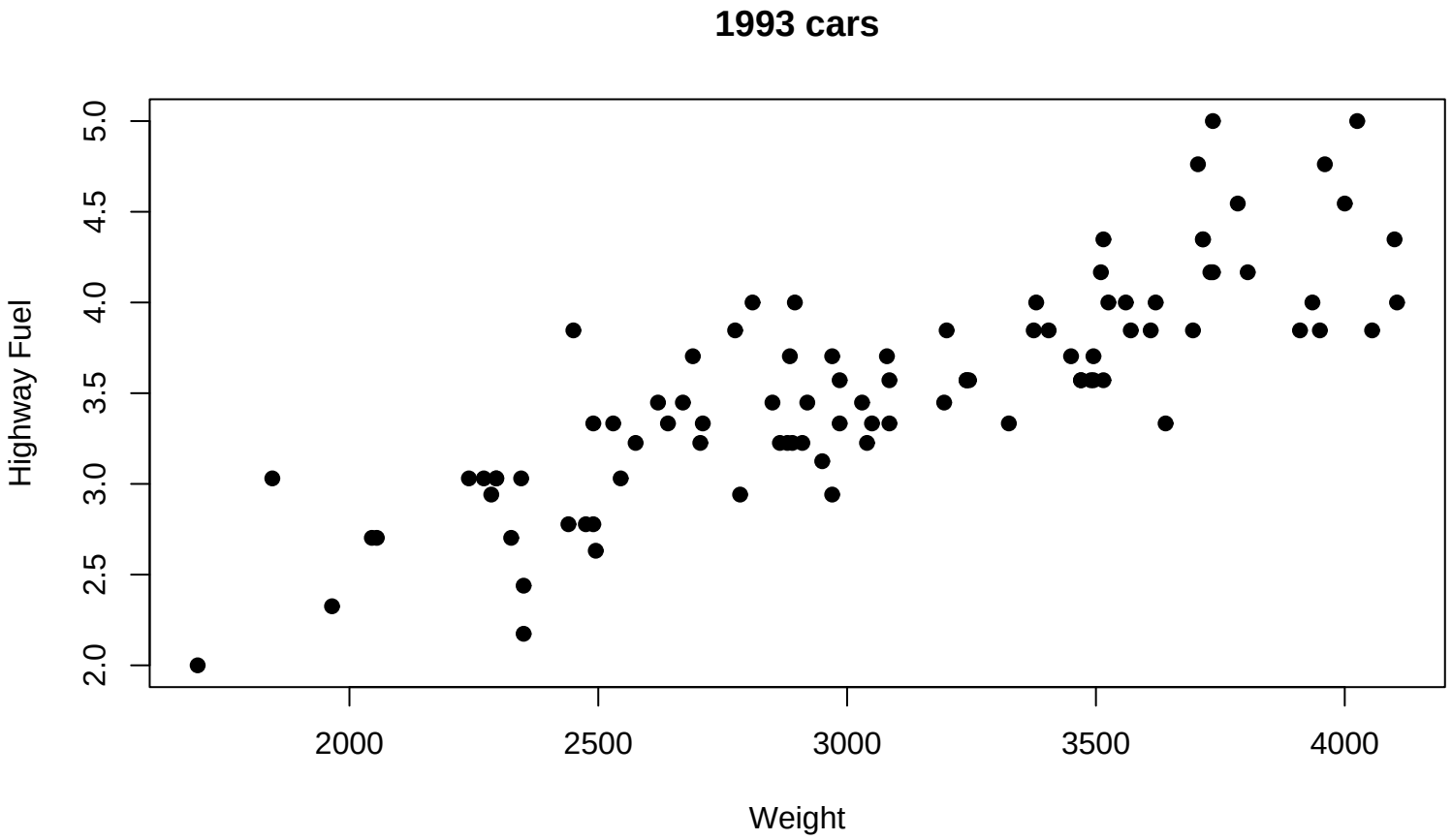
```
plot(x, y, ...)
```

```
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:
```

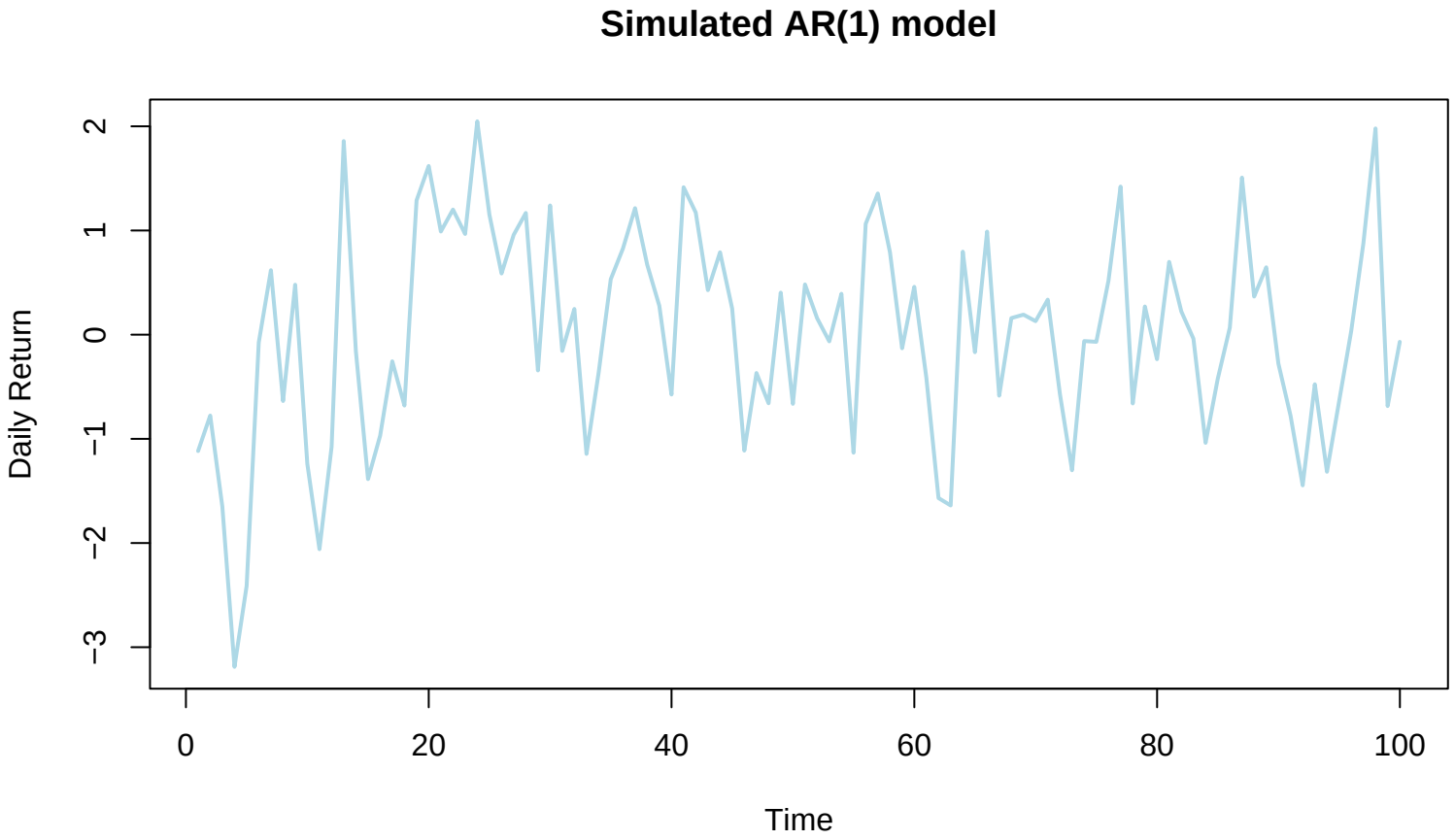
```
type: what type of plot should be drawn.  Possible types are

        *  '"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.

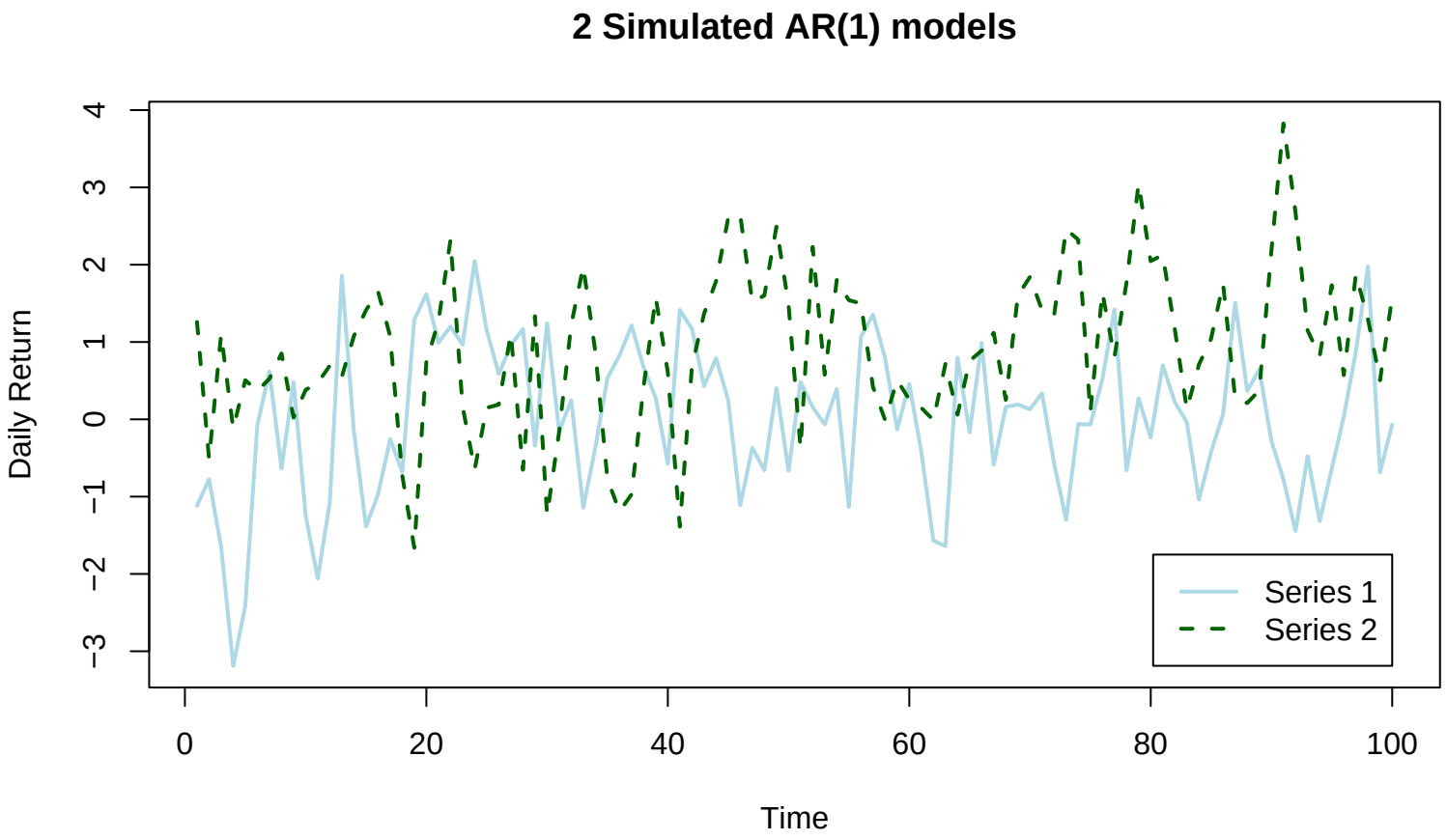



```
par(mar=c(5,4,4,1)+0.1, mfrow=c(1,1))
plot(Weight,HighFuel, pch=16, xlab="Weight", ylab="Highway Fuel",
  main="1993 cars")
```




```
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

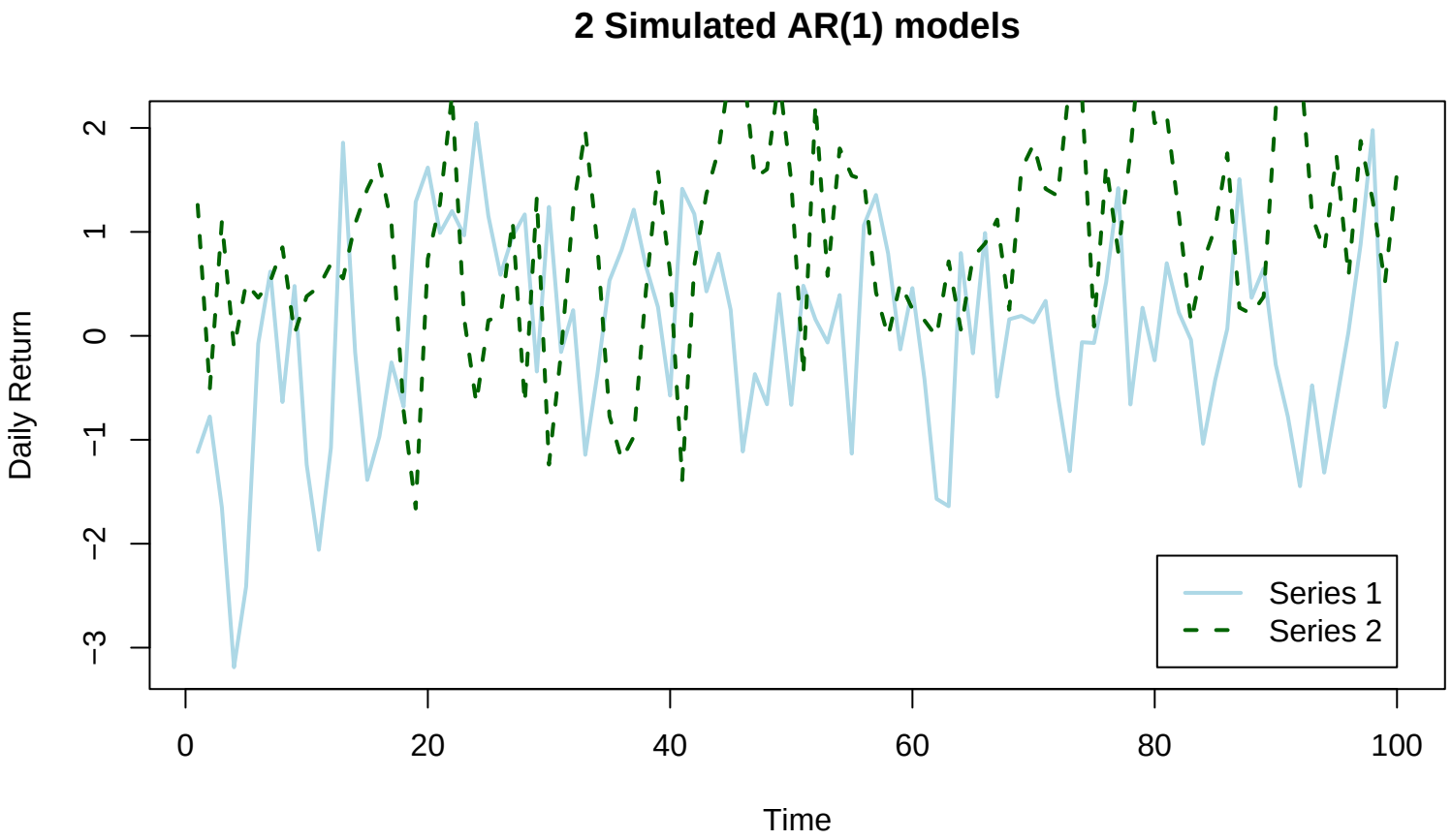


The commands for this plot are as follows.

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