Graphics

R uses two different graphics systems, the **base** or **traditional** graphics system and the **grid** graphics system. The traditional graphics system includes **high-level** plotting functions (that produce complete plots) as well as **low-level** plotting functions (that produce simple graphical output such as lines, dots, text etc.), wheras **grid** only provides the latter, i.e., there are no functions for producing complete plots. But there are packages building on grid, namely **lattice** and **ggplot2**, that provide such high-level plotting functions, too.

High-level plotting functions

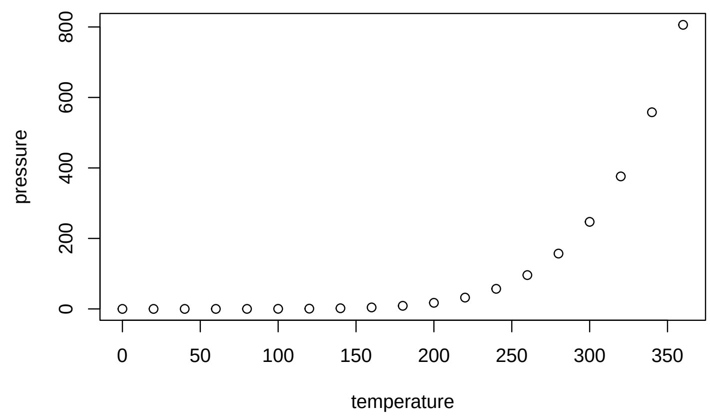
`plot()`

The traditional graphics functions are provided by the `graphics` package, which is automatically loaded in a standard installation of R. The generic `plot()` function is the most important function to produce high-level graphics. On the one hand, this means that the same data can be specified in various ways to give the same output:

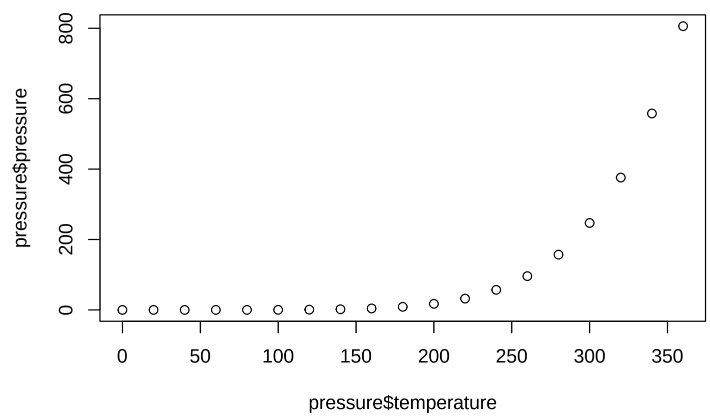
```
# vapor pressure of mercury as a function of temperature
head(pressure)
```

```
##   temperature pressure
## 1           0   0.0002
## 2          20   0.0012
## 3          40   0.0060
## 4          60   0.0300
## 5          80   0.0900
## 6         100   0.2700
```

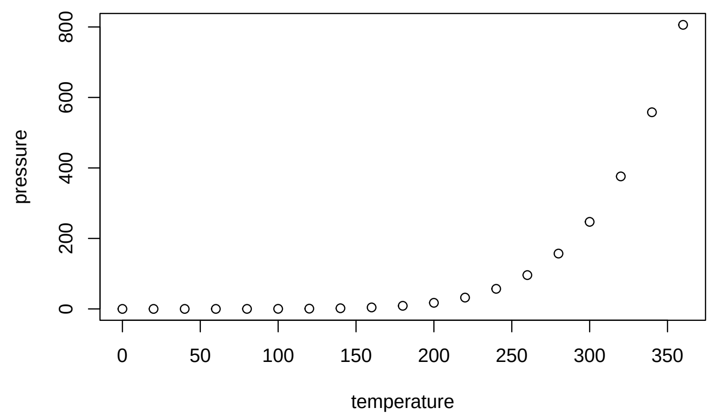
```
plot(pressure)
```




```r
plot(pressure$temperature, pressure$pressure)
```




```r
# formula interface, read "pressure depends on temperature":
plot(pressure ~ temperature, data = pressure)
```

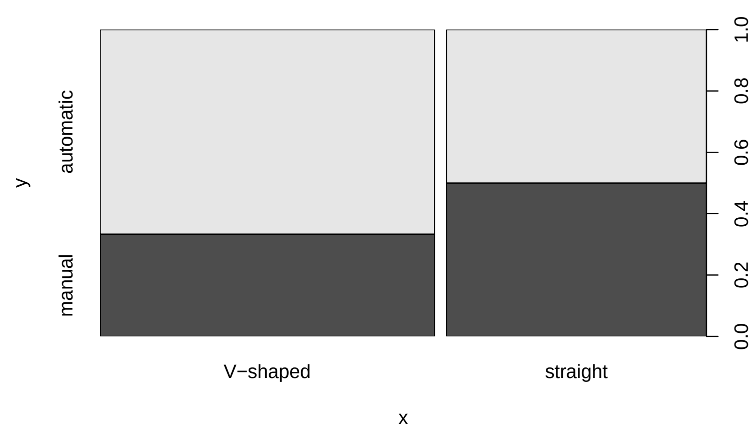


On the other hand, other types of data will lead to different output, i.e., types of plots. In the above example, both variables were numeric and thus, a scatterplot was produced. Let's use the `mtcars` data set to show this:

```
data(mtcars)
head(mtcars)
```

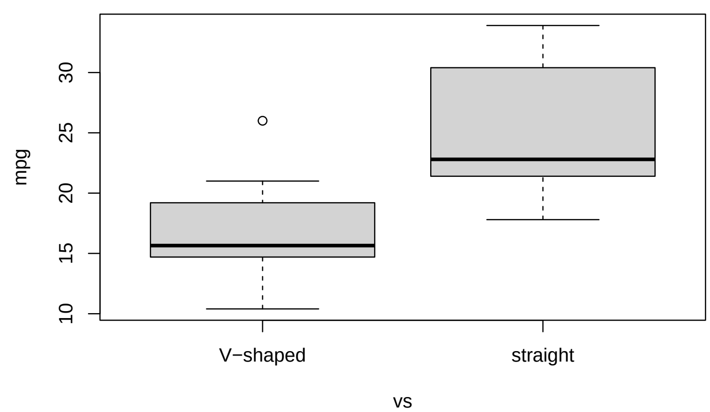
```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
# Coercing vs and am to factors:
mtcars$vs <- factor(mtcars$vs, labels = c("V-shaped", "straight"))
mtcars$am <- factor(mtcars$am, labels = c("automatic", "manual"))
plot(mtcars$vs, mtcars$am)
```



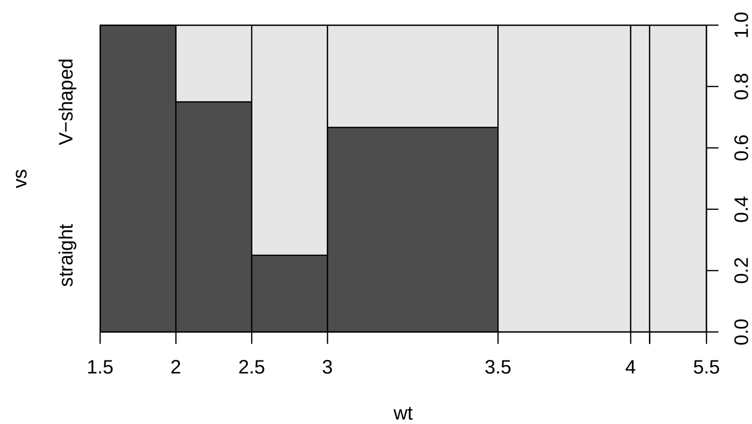
We produced a spine plot now. Accordingly, we can try to mix numeric (metric) and categorical variables:

```
plot(mpg ~ vs, mtcars)
```



Here, we got a boxplot.

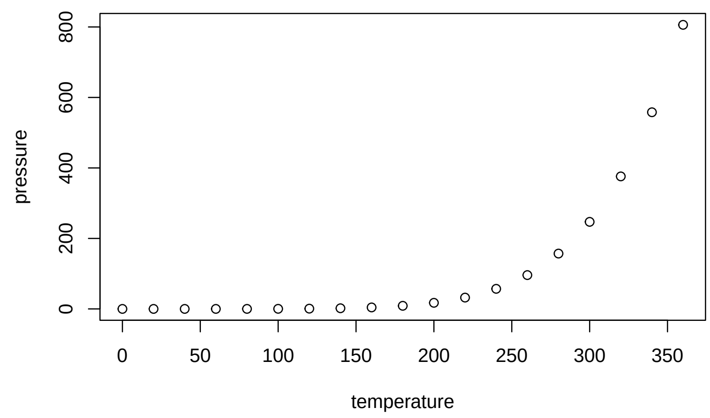
```
plot(vs ~ wt, mtcars)
```



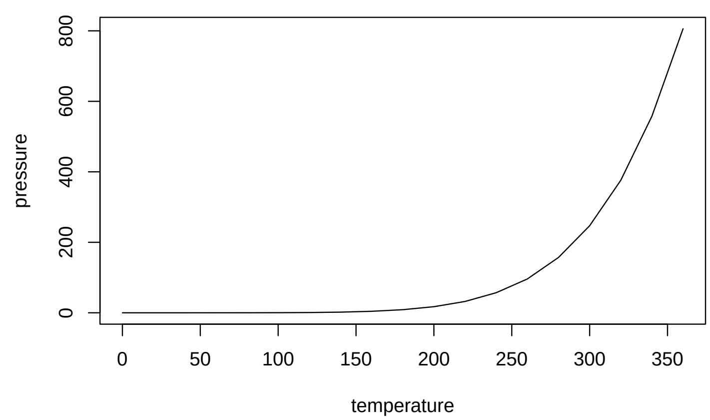
And finally, a spinogram if the dependent variable is categorical and the independent variable is metric.

Let's get back to the initial example to get variations on the basic scatterplot using the `type` argument:

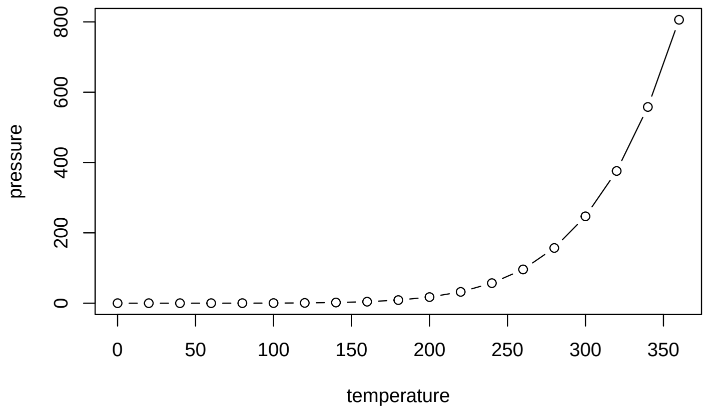
```
plot(pressure ~ temperature, data = pressure, type = "p") # points
```



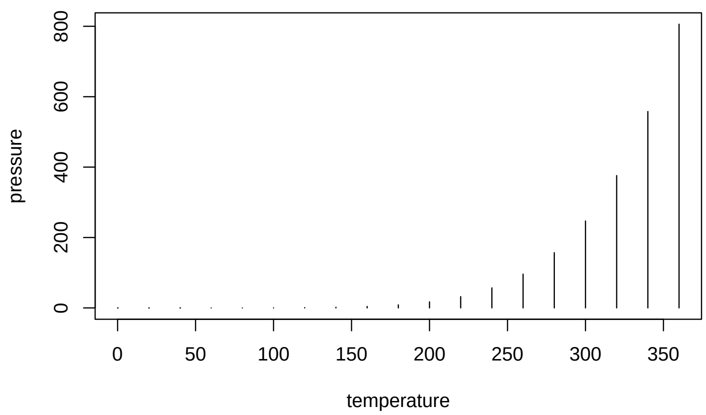

```
plot(pressure ~ temperature, data = pressure, type = "l") # lines
```



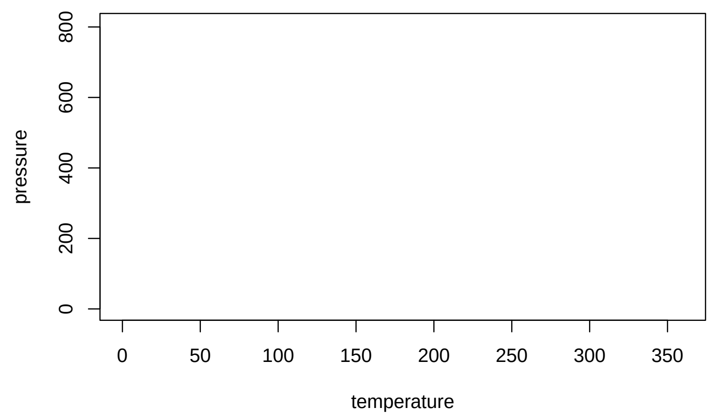

```
plot(pressure ~ temperature, data = pressure, type = "b") # both (points and lines)
```




```
plot(pressure ~ temperature, data = pressure, type = "h") # histogram-like vertical lines
```




```
plot(pressure ~ temperature, data = pressure, type = "n") # no points or lines
```



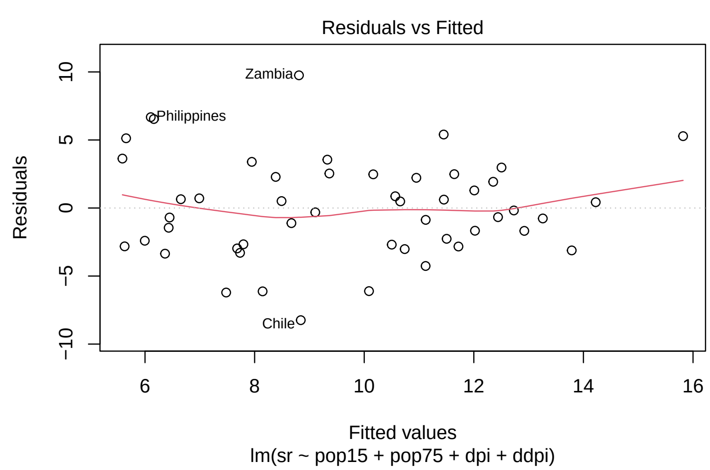
There are also specialized plots for other object types, e.g., objects containing results from statistical computation. As an example, consider the following linear model:

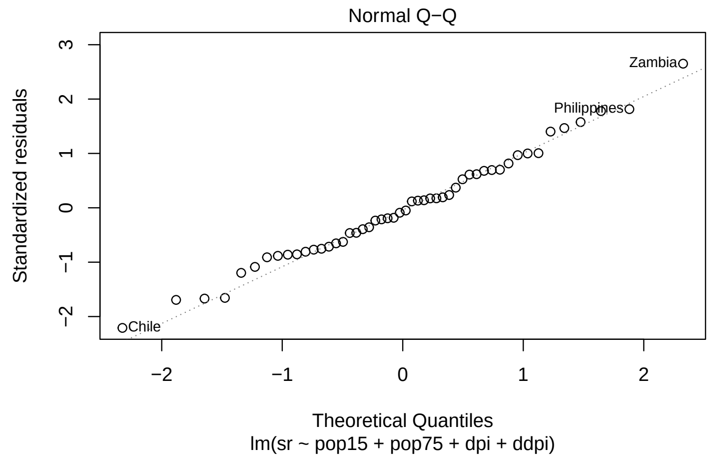
```
lmfit <- lm(sr ~ pop15 + pop75 + dpi + ddpi, data = LifeCycleSavings)
```

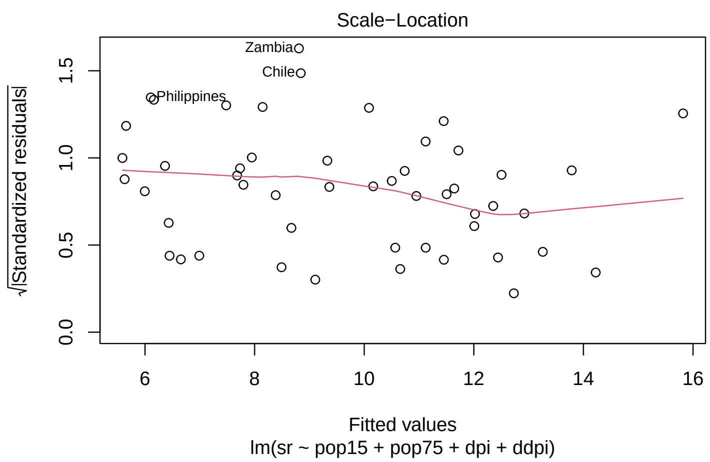
Here, we tried to explain the savings ratio (aggregate personal saving divided by disposable income) by per-capita disposable income, the percentage rate of change in per-capita disposable income, and two demographic variables: the percentage of population less than 15 years old and the percentage of the population over 75 years old.

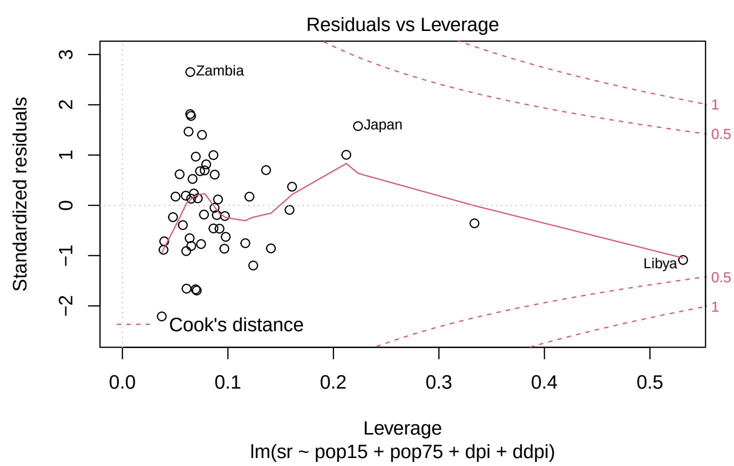
Applying `plot()` on the `lm`-object produces several regression diagnostics plots:

```
plot(lmfit)
```



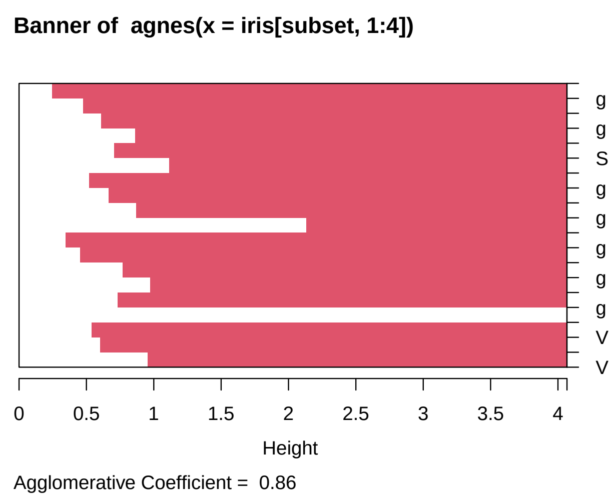


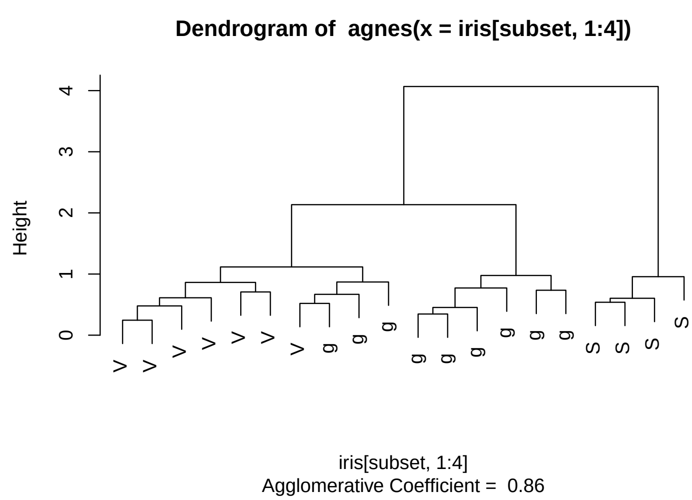




As a second example, consider an agglomerative cluster analysis on the famous iris data set (`?iris`), performed with function `agnes` from the `cluster` package:

```r
library(cluster)
subset <- sample(1:150, 20)
cS <- as.character(Sp <- iris$Species[subset])
cS[Sp == "setosa"] <- "S"
cS[Sp == "versicolor"] <- "V"
cS[Sp == "virginica"] <- "g"
ai <- agnes(iris[subset, 1:4])

plot(ai, labels = cS)
```



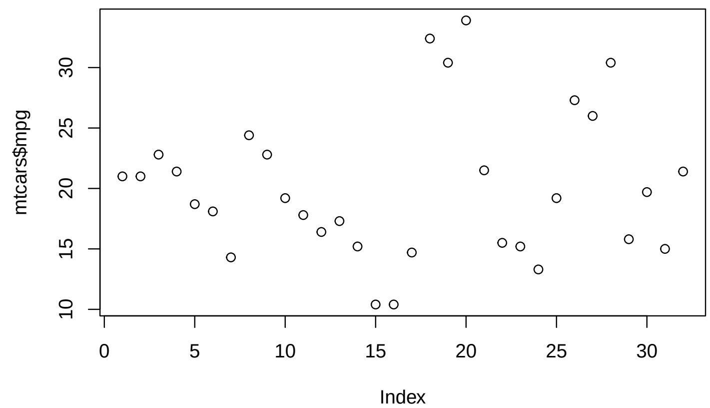


Here, a banner plot and a dendrogram make sense because they are useful in interpreting the cluster solution and evaluating its quality.

Plots of single variables

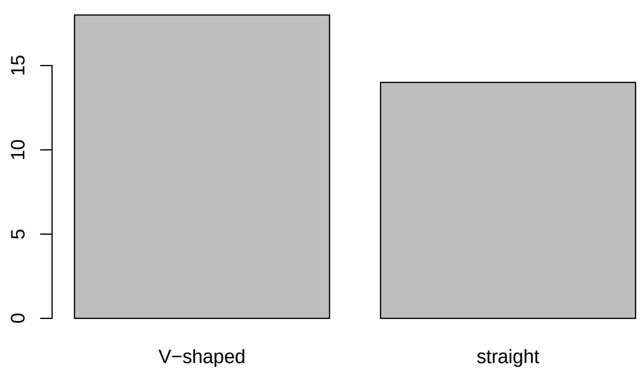
`plot()` also accepts single variables as input. For numeric variables, we get a scatterplot of the numeric values as a function of their indices:

```
plot(mtcars$mpg)
```



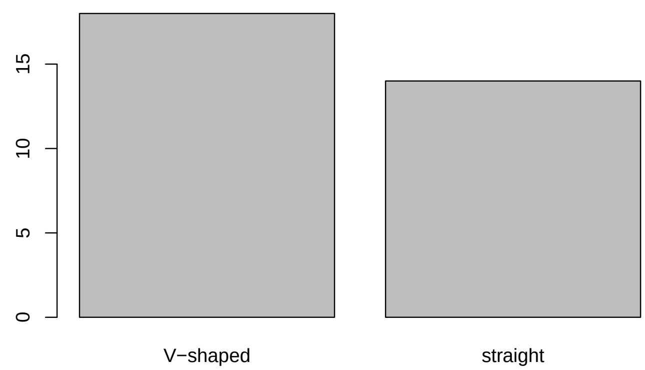
And for categorical variables, a simple bar chart is produced:

```
plot(mtcars$vs)
```



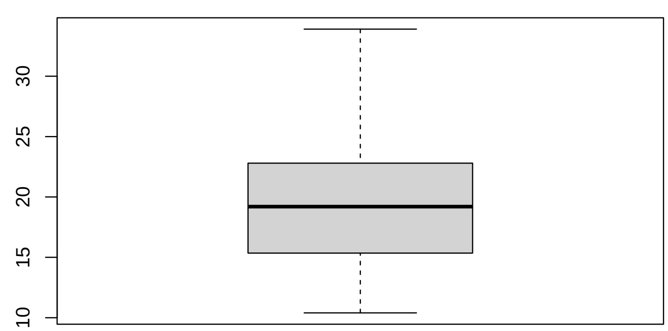
We can also call function `barplot()` explicitely to get the same result (in this case, we have to provide a numeric vector of bar heights, i.e., the absolute or relative frequencies):

```
barplot(table(mtcars$vs))
```

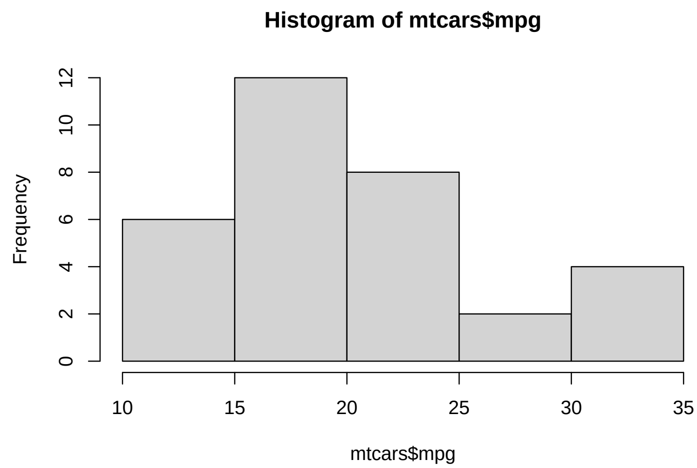


There are alternatives to these plots. For metric variables, boxplots, histograms, stripcharts ("1D-Scatterplot"), and stem-and-leaf plots make sense:

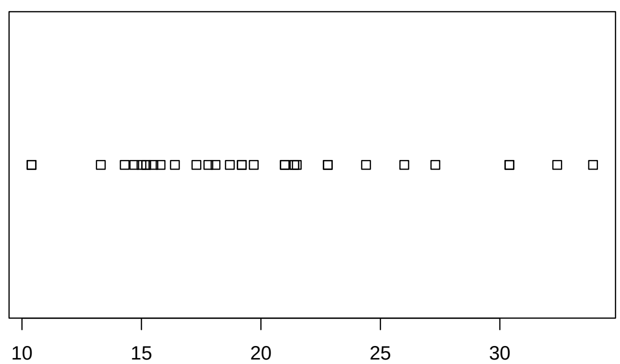
```
boxplot(mtcars$mpg)
```




```
hist(mtcars$mpg)
```




```
stripchart(mtcars$mpg)
```

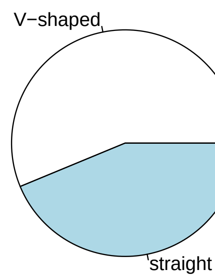



```
stem(mtcars$mpg)
```

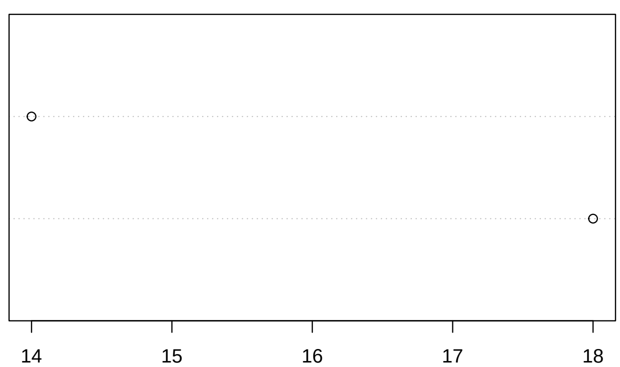
```
##
##   The decimal point is at the |
##
##   10 | 44
##   12 | 3
##   14 | 3702258
##   16 | 438
##   18 | 17227
##   20 | 00445
##   22 | 88
##   24 | 4
##   26 | 03
##   28 |
##   30 | 44
##   32 | 49
```

For a categorical variable, we could use pie or dot charts (again, we have to provide a table):

```
pie(table(mtcars$vs))
```



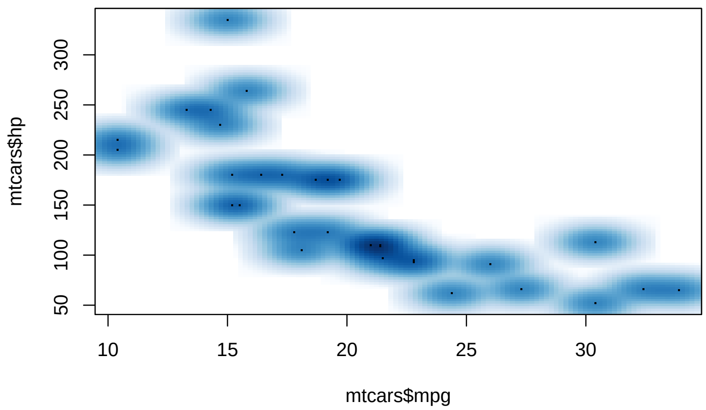

```
dotchart(as.numeric(table(mtcars$vs)))
```



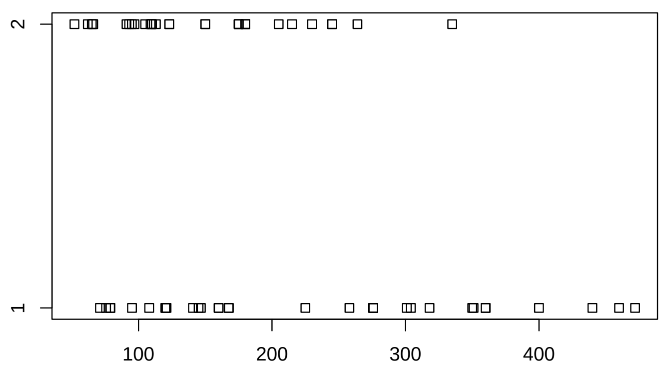
Plots of two variables

Apart from the default plot we get when providing two variables, there are many other plots that can be produced using the following functions:

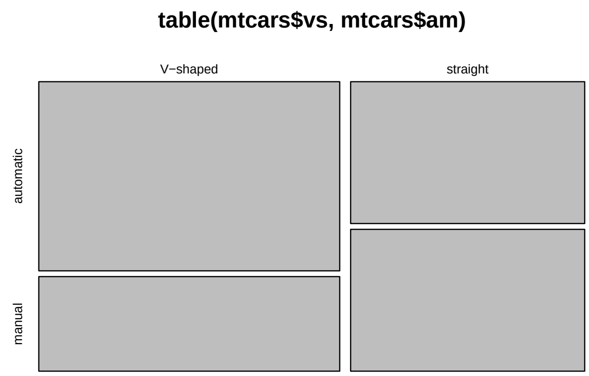
```
smoothScatter(mtcars$mpg, mtcars$hp) # Smooth scatterplot
```




```
stripchart(list(mtcars$disp, mtcars$hp)) # Strip chart
```



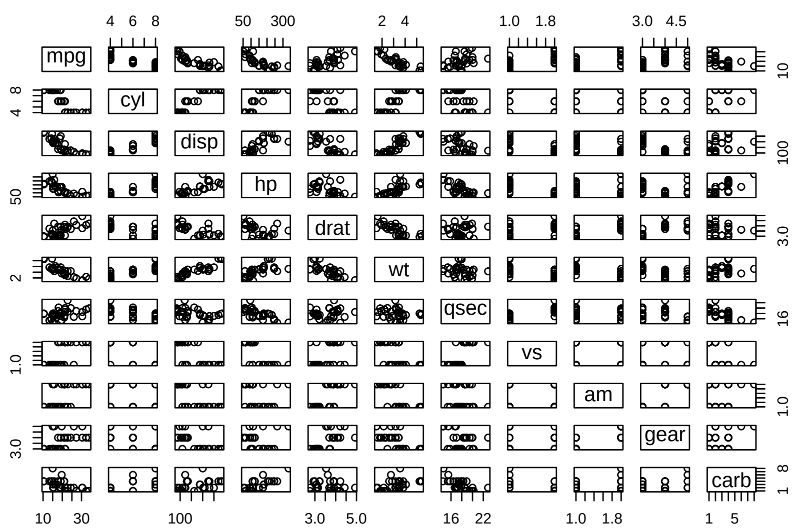

```
mosaicplot(table(mtcars$vs, mtcars$am)) # Mosaic plot
```



Plots of many variables

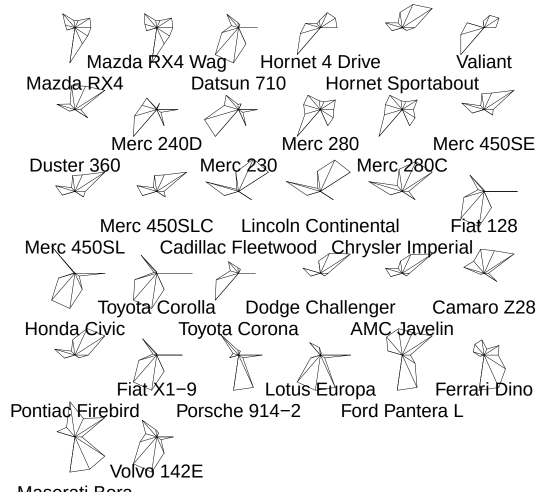
`plot()` produces a scatterplot matrix when all variables are numeric:

```
plot(mtcars)
```



The same is done by function `pairs()`. The stars plot is one example of the so-called *small multiples* technique, where many small plots are produced on one single page:

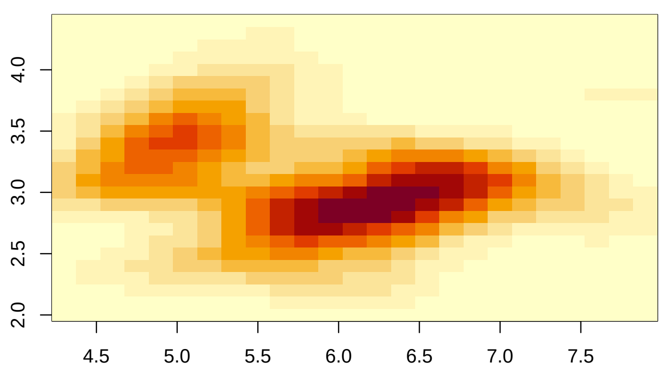
```
stars(mtcars)
```



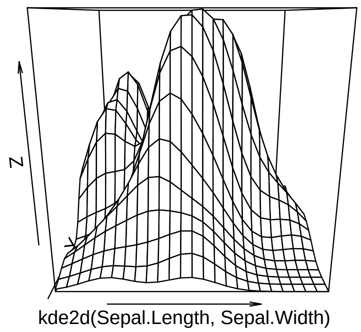
Here, every variable determines the length of the arms of each star.

There are some special functions when we have exactly three variables (more precisely, two variables plus a two-dimensional density estimation, giving us x-, y-, and z-coordinates):

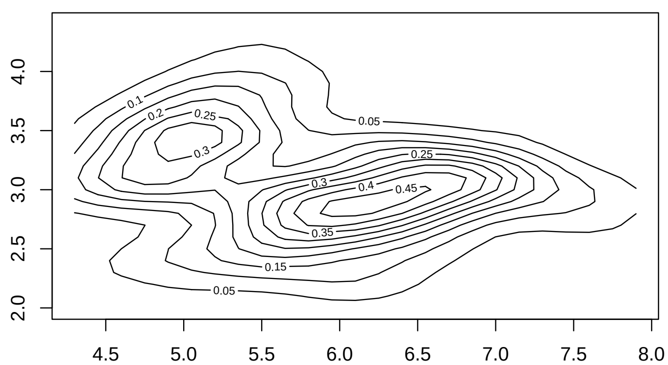
```
library(MASS) # for kde2d
attach(iris)

image(kde2d(Sepal.Length, Sepal.Width)) # colored image (spatial data)
```



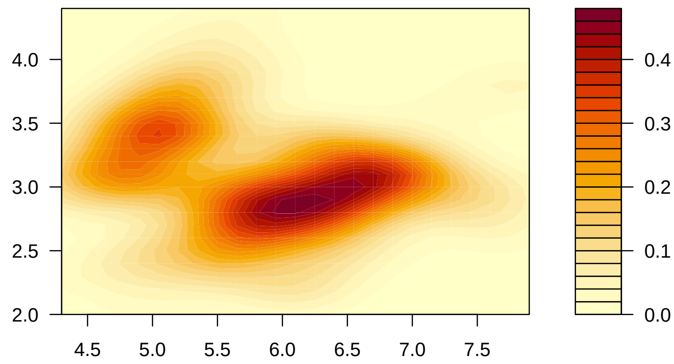

```
persp(kde2d(Sepal.Length, Sepal.Width)) # perspective plot
```




```r
contour(kde2d(Sepal.Length, Sepal.Width)) # contour plot
```




```r
filled.contour(kde2d(Sepal.Length, Sepal.Width)) # filled contour plot
```

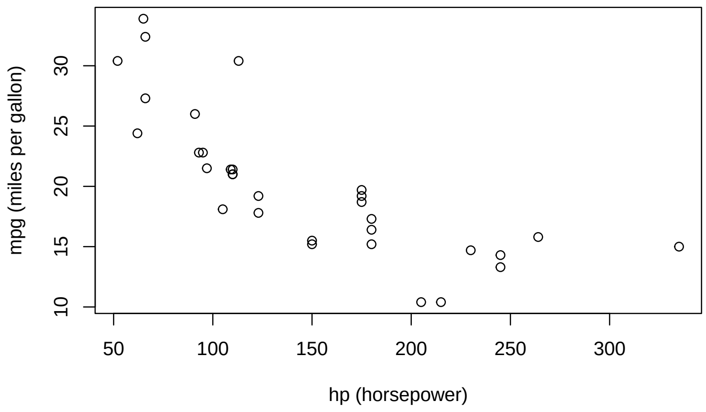



```
detach(iris)
```

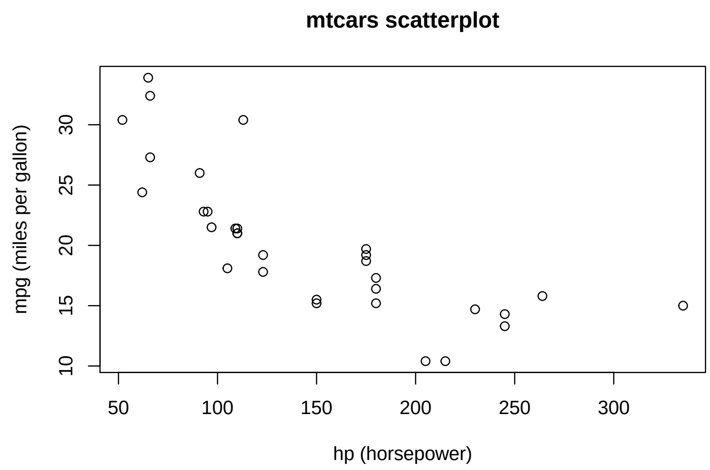
Arguments to graphics functions

There are many arguments like `xlab` or `main` that are standard, i.e., many high-level graphics functions will accept them.

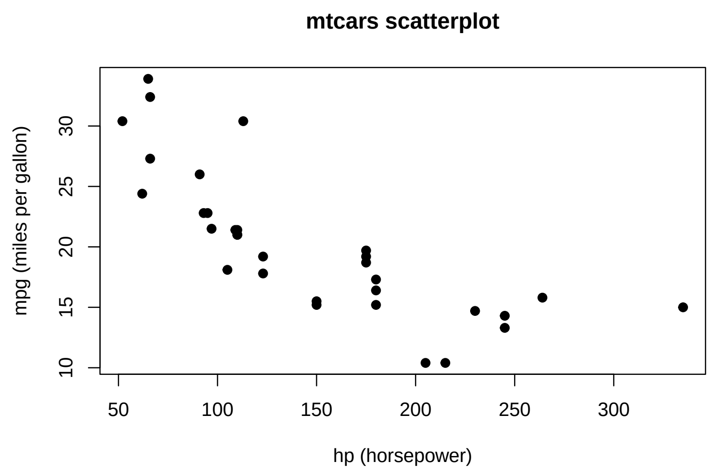
```
attach(mtcars)

# change x and y labels
plot(hp, mpg, xlab = "hp (horsepower)",
     ylab = "mpg (miles per gallon)")
```



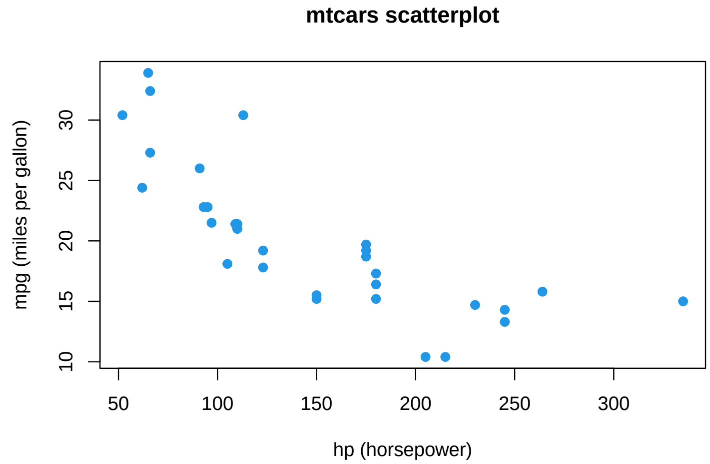

```
# add title
plot(hp, mpg, xlab = "hp (horsepower)",
     ylab = "mpg (miles per gallon)",
     main = "mtcars scatterplot")
```



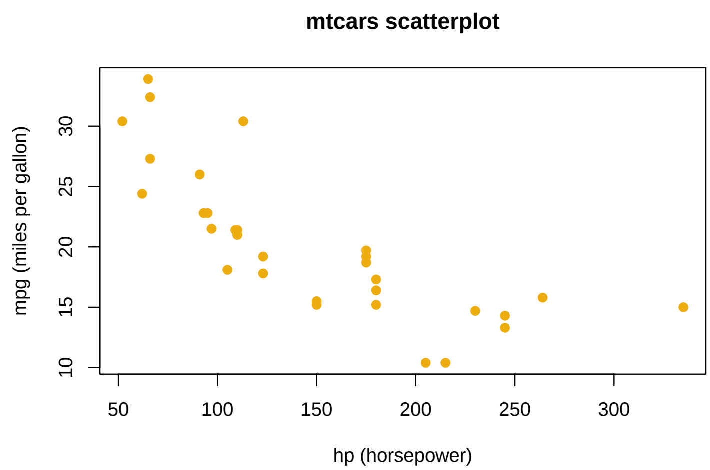

```r
# change plotting symbol
plot(hp, mpg, xlab = "hp (horsepower)",
     ylab = "mpg (miles per gallon)",
     main = "mtcars scatterplot", pch = 19)
```




```r
# change color (blue)
plot(hp, mpg, xlab = "hp (horsepower)",
     ylab = "mpg (miles per gallon)",
     main = "mtcars scatterplot", pch = 19, col = 4)
```




```r
# use color name
plot(hp, mpg, xlab = "hp (horsepower)",
     ylab = "mpg (miles per gallon)",
     main = "mtcars scatterplot", pch = 19, col = "darkgoldenrod2")
```

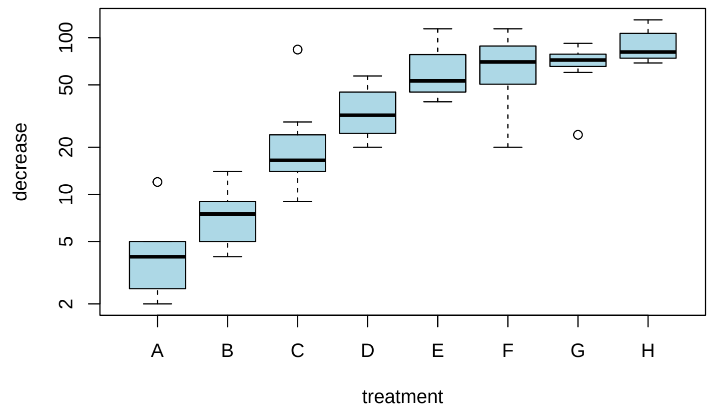



```r
detach(mtcars)
```

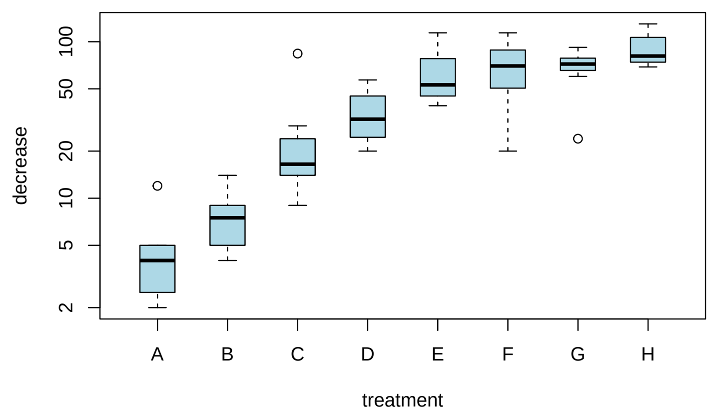
```r
colors() #list of available named colors
```

Other arguments are specific to a certain plot type, e.g., the `boxwex` argument that specifies the width of the boxes when producing a boxplot:

```r
boxplot(decrease ~ treatment, data = OrchardSprays, log = "y", col = "light blue")
```




```
boxplot(decrease ~ treatment, data = OrchardSprays, log = "y", col = "light blue", boxwex = 0.5)
```



The traditional graphics system has its strength in the creation of static plots and only provides limited functionality for interacting with graphical output, although there are some functions like `locator()` (returning the coordinates) or `identify()` that adds labels to data points:

```
plot(mtcars$hp, mtcars$mpg)
locator()
```

Customization

The so-called "graphics state" contains a large number of settings for each traditional graphics device. The main function to access the graphics state is via the `par()` function:

```
par()
```

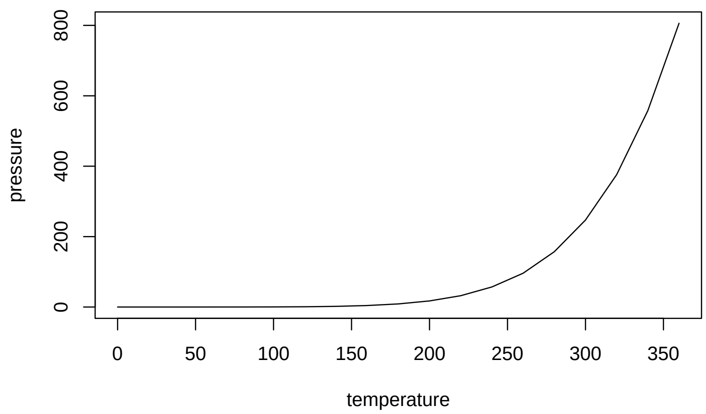
We can query for specific settings:

```r
par(c("col", "lty"))
```

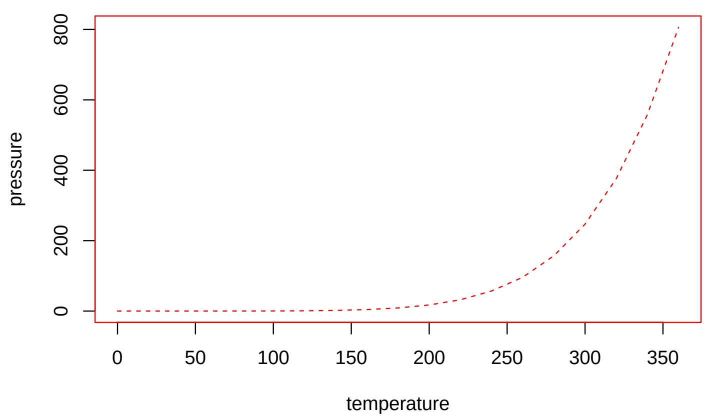
```
## $col
## [1] "black"
##
## $lty
## [1] "solid"
```

We can modify these settings by specifying a value:

```r
plot(pressure, type = "l")
```




```r
par(c(col = "red", lty = "dashed"))
plot(pressure, type = "l")
```



Because this has a persistent effect, we return to the default values:

```
par(c(col = "black", lty = "solid"))
```

We consult `par` for all the high-level traditional graphics state settings:

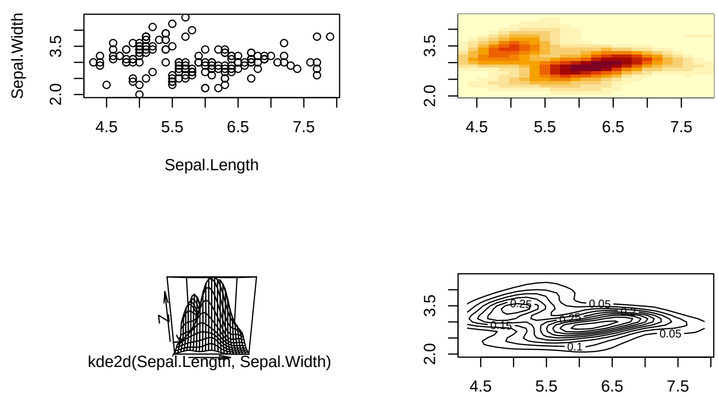
```
?par
```

Some important settings (lines, text, axes) will be introduced in the exercises for this chapter.

Multiple plots

There are various ways to place multiple plots on a single page. Using the traditional graphics state, we use `mfrow` and `mfcol` to define plot regions:

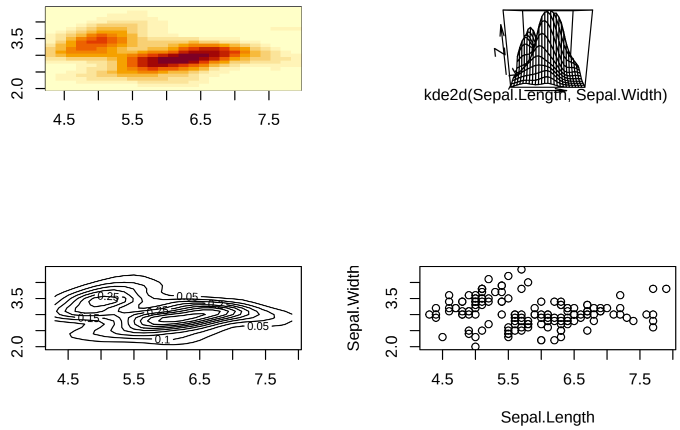
```
attach(iris)
par(mfrow = c(2, 2))

plot(Sepal.Length, Sepal.Width)
image(kde2d(Sepal.Length, Sepal.Width))
persp(kde2d(Sepal.Length, Sepal.Width))
contour(kde2d(Sepal.Length, Sepal.Width))
```




```
par(mfrow = c(1, 1))
```

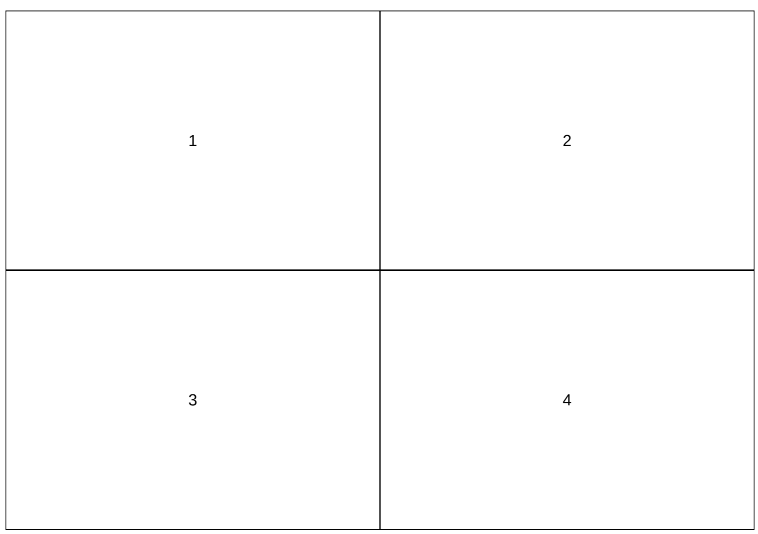
This only allows us to create figure regions of *equal* sizes. If we want to create multiple regions of *unequal* sizes, we have to use `layout()`. Of course, this command can also be used to create regions of equal sizes:

```
layout(rbind(c(1, 2),
             c(3, 4)))

image(kde2d(Sepal.Length, Sepal.Width))
persp(kde2d(Sepal.Length, Sepal.Width))
contour(kde2d(Sepal.Length, Sepal.Width))
plot(Sepal.Length, Sepal.Width)
```



We can check the regions created using `layout.show()`:

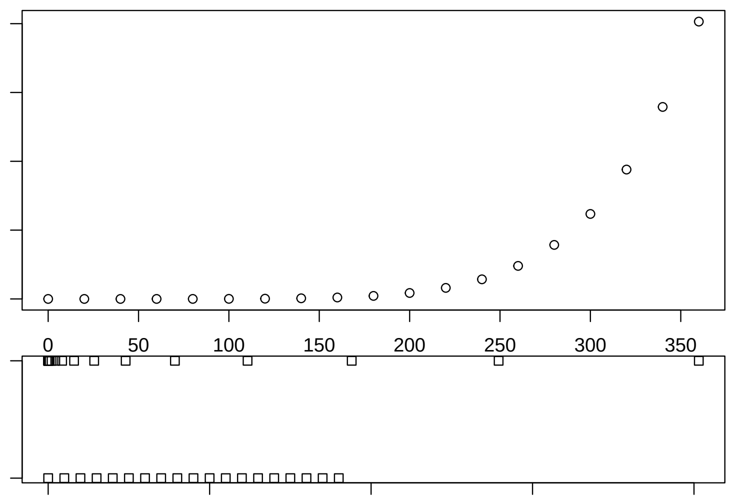
```
layout(rbind(c(1, 2),
             c(3, 4)))
layout.show(4)
```




```
layout(matrix(c(1, 2)), heights = c(2, 1))
layout.show(2)
```

It may be necessary to specify some additional arguments (margins, apect ratio, gaps between figures) to get a good-looking graph:

```
layout(matrix(c(1, 2)), heights = c(2, 1))
par(mar = rep(1, 4))
plot(pressure)
stripchart(pressure)
```

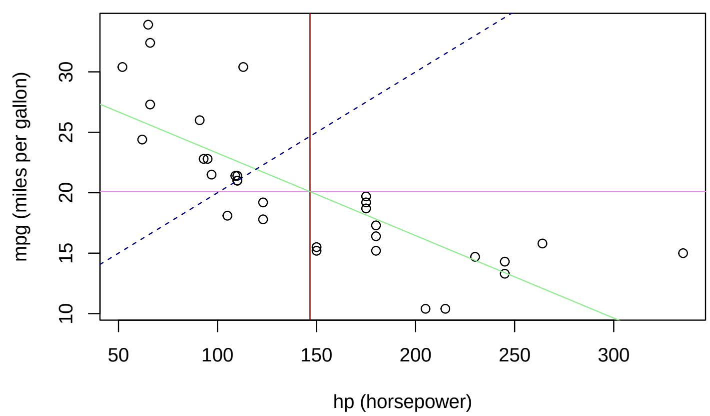



```
par(mar = c(5, 4, 4, 2) + 0.1)
```

Low-level plotting functions

Modifying the output from high-level graphics funtcions sometimes does not suffice, but **plot annotation** may be needed. A standard example is adding a regression line to a given scatterplot:

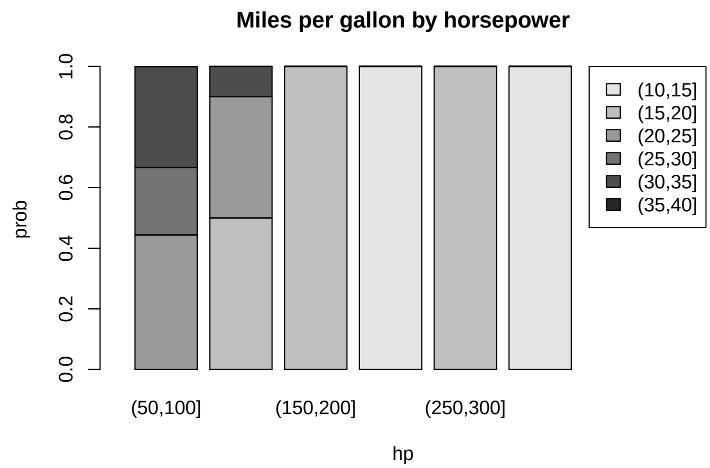
```
attach(mtcars)

plot(hp, mpg, xlab = "hp (horsepower)", ylab = "mpg (miles per gallon)")
abline(h = mean(mpg), col = "violet")
abline(v = mean(hp), col = "darkred")
abline(lsfit(hp, mpg), col = "lightgreen")
abline(a = 10, b = 0.1, lty = 2, col = "darkblue")
```




```
detach(mtcars)
```

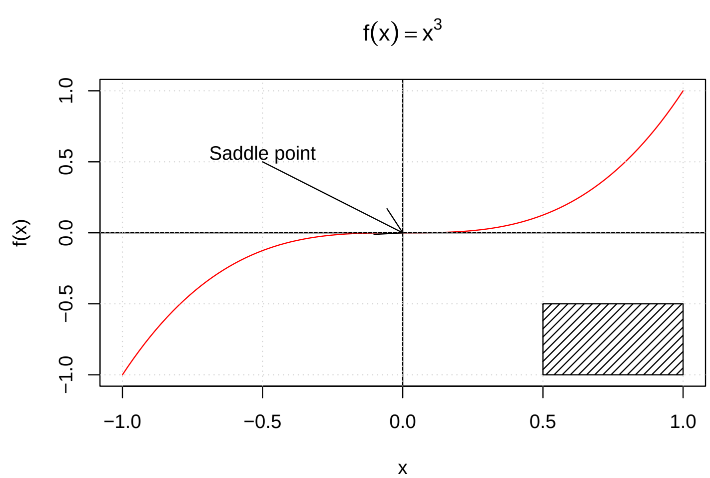
Another example - adding a legend:

```
mpg.cat.5 <- cut(mtcars$mpg, breaks = seq(10, 40, 5))
hp.cat.50 <- cut(mtcars$hp, breaks = seq(50, 350, 50))
mpg.hp <- table(mpg.cat.5, hp.cat.50)
mpg.hp.rel <- round(prop.table(mpg.hp, 2), digits = 3)

barplot(mpg.hp.rel, yaxt = "n", xlab = "hp", ylab = "prob",
        col = c("grey90", "grey75", "grey60", "grey45", "grey30", "grey15"),
        main = "Miles per gallon by horsepower",
        xlim = c(0, 9)) #stacked bar chart (100% stacked)
legend("topright", legend = rownames(mpg.hp.rel), col = "black",
       fill = c("grey90", "grey75", "grey60", "grey45", "grey30", "grey15"))
axis(2, at = seq(0, 1, 0.2))
```



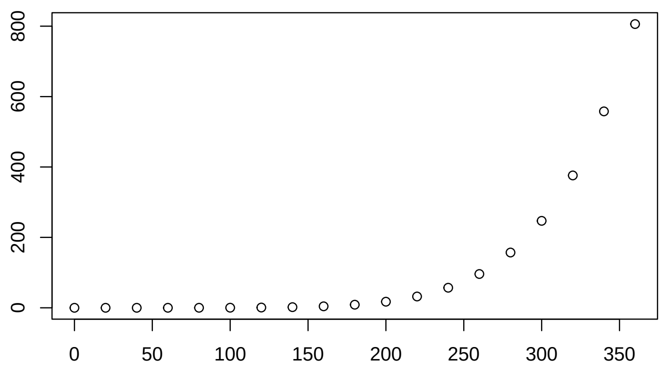
The following example gives an idea of the possibilities of plot annotation:

```
f = function(x) x ^ 3
curve(f, from = -1, to = 1, col = "red",
      main = expression(f(x) == x^3))

## A bit of cosmetics:
abline(h = 0, v = 0) ## X- and Y-axis
grid()

## Special labeling:
arrows(x0 = -0.5, y0 = 0.5, x1 = 0, y1 = 0)
text(x = -0.5, y = 0.55, labels = "Saddle point")

## Other graphics commands, e.g., lines() or rect():
rect(xleft = 0.5, ybottom = -1, xright = 1, ytop = -0.5, density = 20)
```



A simple plot from scratch

```
plot.new() # starts a new, completely blank plot
plot.window(range(pressure$temperature), # set scales of the axes
            range(pressure$pressure))
plot.xy(pressure, type = "p") # draw data symbols at data locations
box() # craw a rectangle around the plot region
axis(1) # add x axis
axis(2) # add y axis
```



To get additional examples, please type

```
demo(graphics)
```

or

```
example(barplot)
example(pie)# etc.
```

Grid graphics

`grid` has to be loaded in order to use the provided functions.

```
library(grid)
```

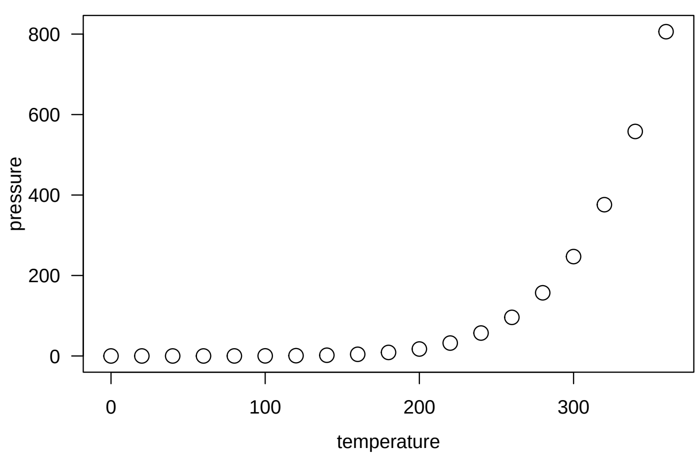
It provide extensive online documentation in a series of vignettes, available via the vignette function:

```
vignette(package = "grid")
```

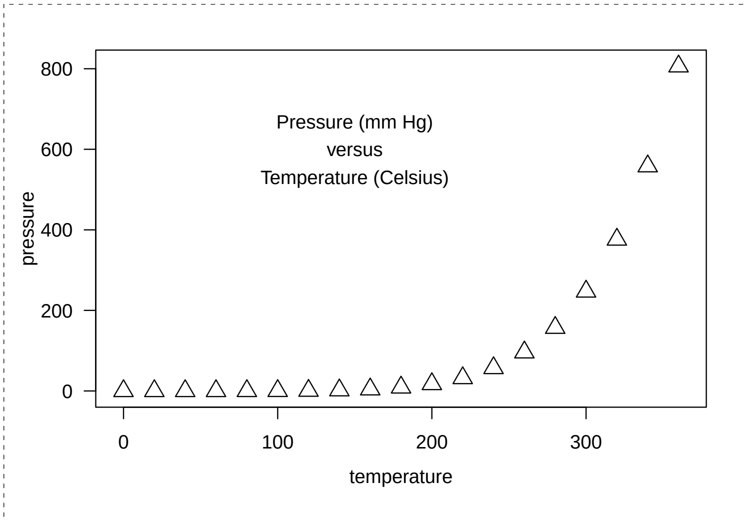
Example

A viewport is a rectangular region that provides a context for drawing. Objects created by the `viewport()` function are only descriptions of a drawing context. A viewport object must be pushed onto the viewport tree before it has any effect on drawing. We reproduce the pressure data scatterplot of notebook 8:

```
# creating and pushing two viewports:
pushViewport(plotViewport(c(5, 4, 2, 2))) # convenience function for producing a viewport

# produce a viewport with x- and y-axed scaled according to the data:
pushViewport(dataViewport(pressure$temperature,
                    pressure$pressure,
                    name = "plotRegion"))

# use graphical primitives to add plot elements:
# points are added to the most recent viewport, an object
# called "dataSymbols" is created
grid.points(pressure$temperature, pressure$pressure,
            name = "dataSymbols")
grid.rect() # add rectangle
grid.xaxis() # add x-axis
grid.yaxis() # add y-axis
grid.text("temperature", y = unit(-3, "line")) # add annotation
grid.text("pressure", x = unit(-3, "line"), rot = 90) # add annotation
```



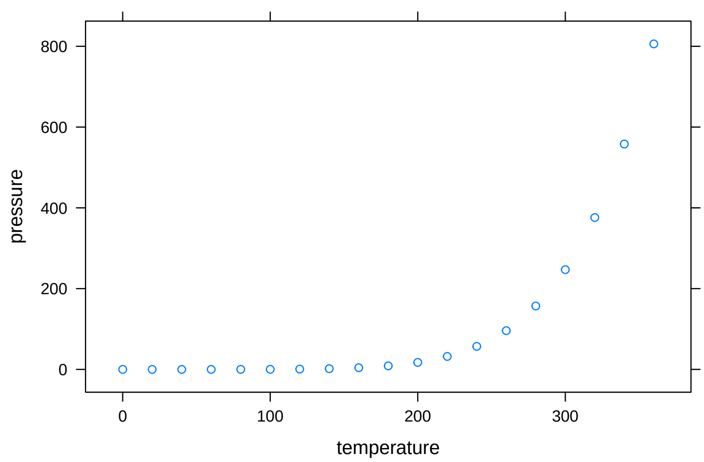

```
grid.edit("dataSymbols", pch = 2) # change symbols (use object name)
upViewport(2) # navigate back to working within the entire page
grid.rect(gp = gpar(lty = "dashed")) # draw a dashed rectangle
downViewport("plotRegion") # go back to the plot region to add text
grid.text("Pressure (mm Hg)\nversus\nTemperature (Celsius)",
          x = unit(150, "native"), y = unit(600, "native"))
```



Lattice

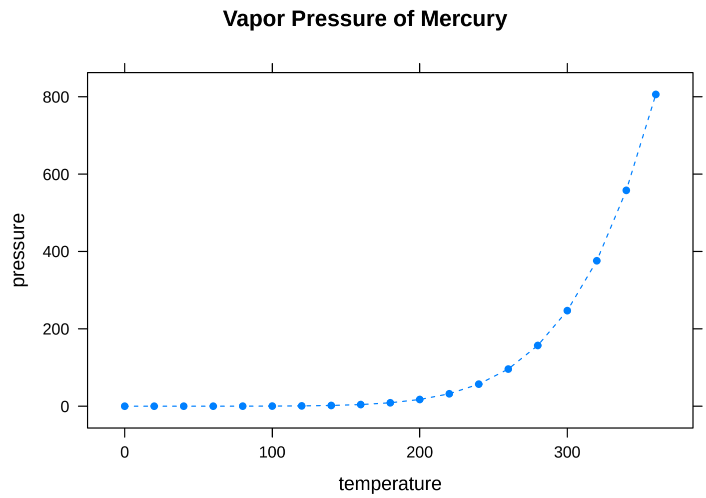
`lattice` is a high-level graphics system built on top of `grid`. We can load it to reproduce the above plot once again. The function that produces a scatterplot using `lattice` is called `xyplot`:

```
library(lattice)
xyplot(pressure ~ temperature, pressure)
```



Of course, we can change the appearance of a plot using arguments in `lattice`, too:

```
xyplot(pressure ~ temperature, pressure, type = "o",
       pch = 16, lty = "dashed",
       main = "Vapor Pressure of Mercury")
```

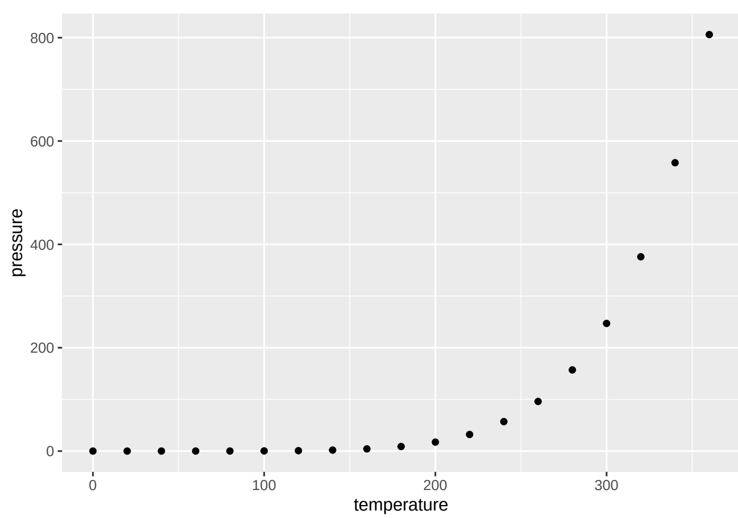


There are many other equivalents, too (`barchart()` for bar charts in lattice, `bwplot()` for boxplots etc.).

ggplot2

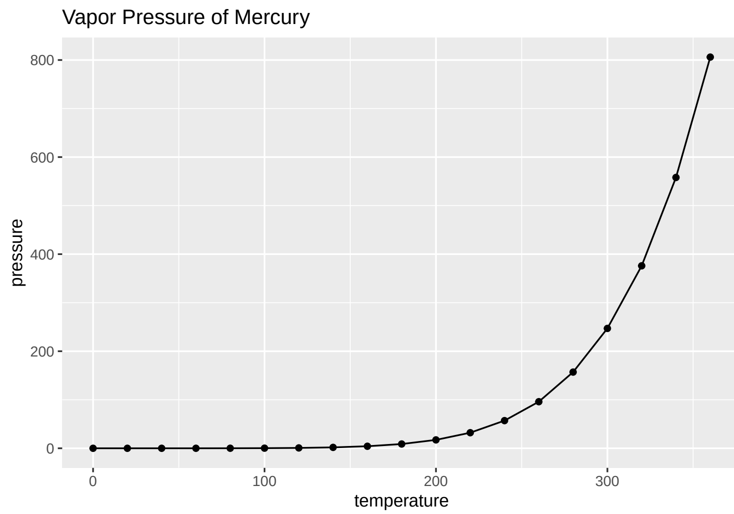
Another quite widely-used graphics package that is building on grid is `ggplot2`. Again, we reproduce the already-known scatterplot:

```
library(ggplot2)
qplot(temperature, pressure, data = pressure)
```



For customization purposes, users have to get acquainted with the `geom` argument that is unique to `ggplot2`:

```
qplot(temperature, pressure, data = pressure,
      main = "Vapor Pressure of Mercury",
      geom = c("point", "line"))
```


Vapor Pressure of Mercury
pressure
800
600
400
200
0
0
100
200
300
temperature