STAT 210
Applied Statistics and Data Analysis: Graphics in R
Basics

Joaquín Ortega

joaquin.ortegasanchez@kaust.edu.sa

Graphics in R

Graphics in R

Main graphical packages in R:

- base
- lattice by Deepayan Sarkar
- ggplot2 by Hadley Wickham
- other packages such as shiny and plotly.

We will only consider the base package (although some of the plots in the previous presentation were made with lattice and ggplot2)

Graphics in R

We start with a demo of the graphical capabilities of `R` using the functions `example` and `demo`:

```r
example(plot)
example(par)
example("palette")

demo(graphics)
demo(persp)
demo(image)
demo(colors)
```

`plot`

plot

plot() is the standard function for plotting in R.

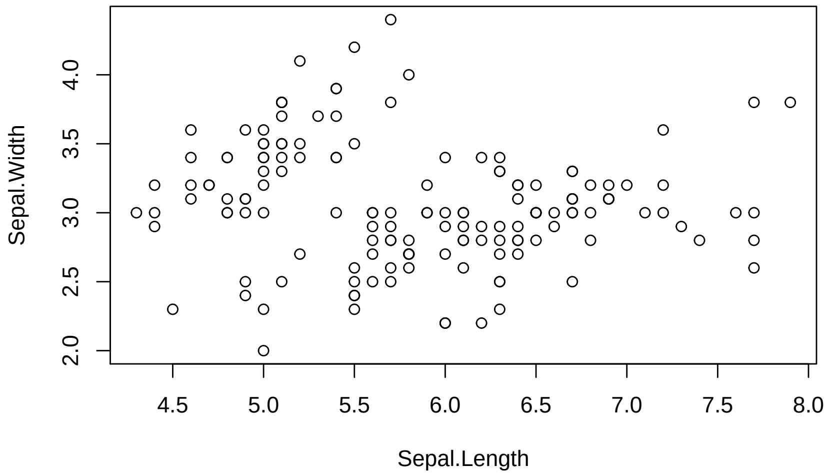
What you get depends on the object that holds the data, the mode of the data, and the syntax you use.

Let's see some examples using again the data set iris.

```r
attach(iris)
```

plot

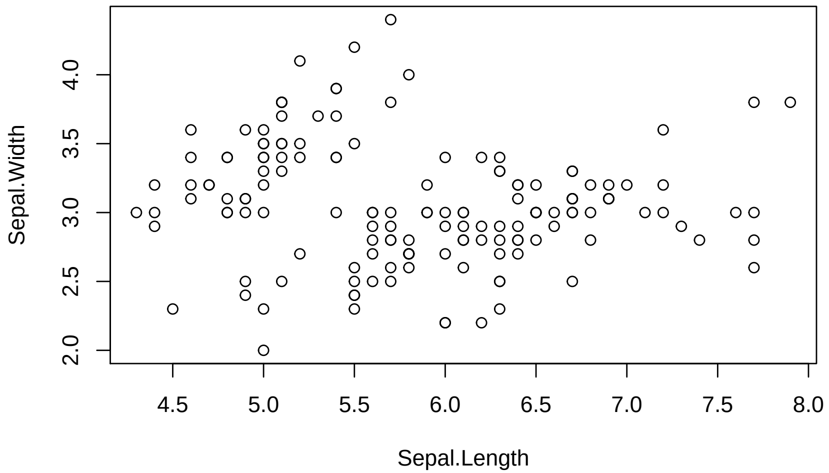
```
plot(Sepal.Length, Sepal.Width)
```



plot

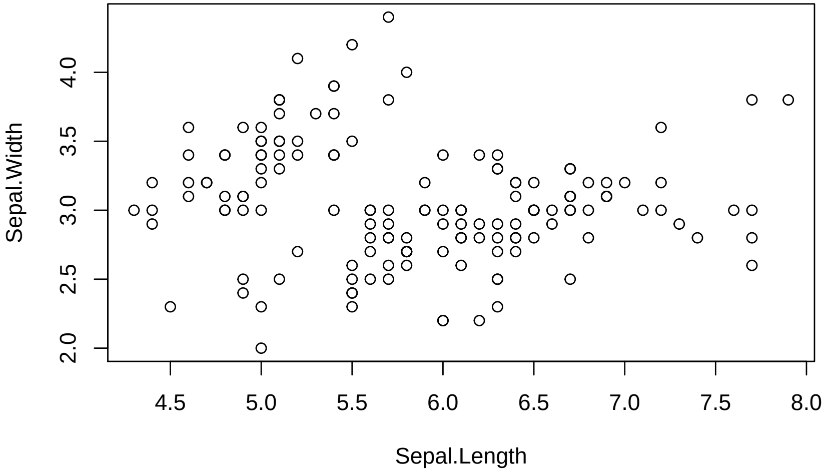
The same result is obtained with the following commands

```
plot( Sepal.Width ~ Sepal.Length)
```



plot

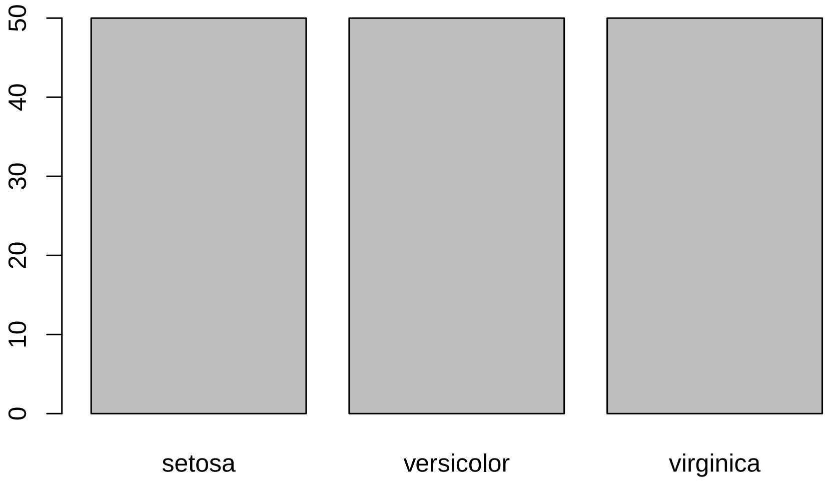
```r
plot(~ Sepal.Length + Sepal.Width)
```



plot

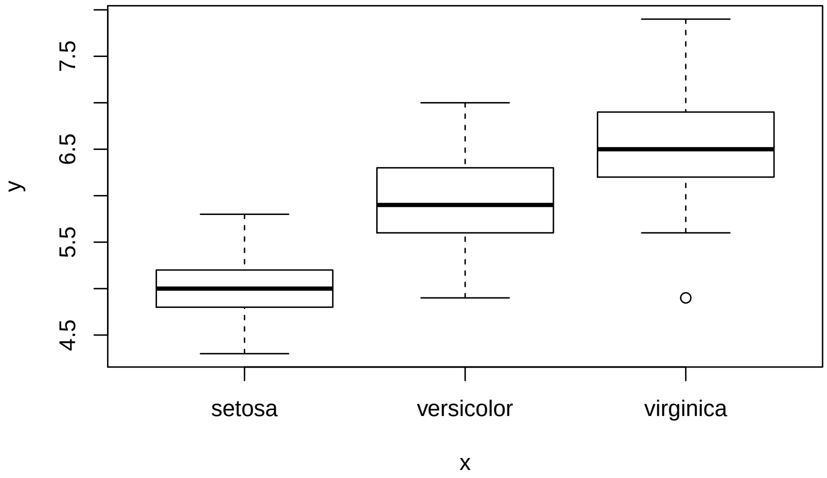
If the argument to plot is a factor, then

```r
plot(Species)
```



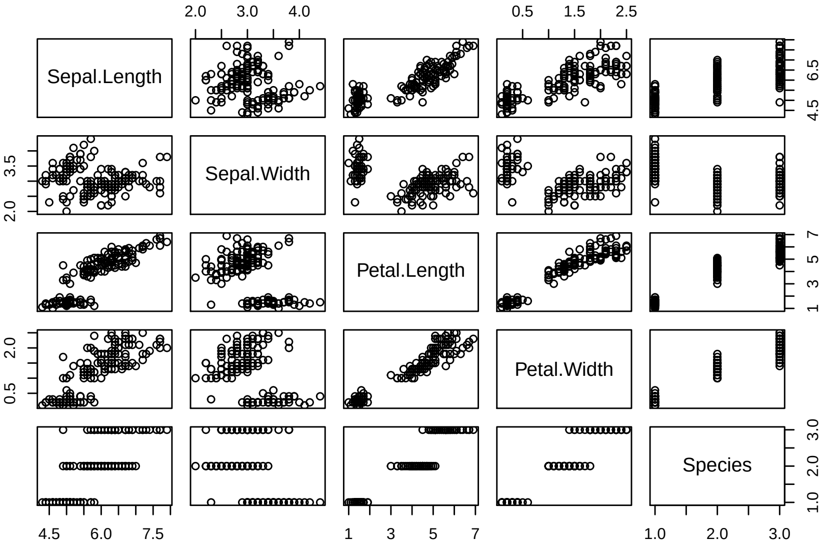
plot

```r
plot(Species, Sepal.Length)
```



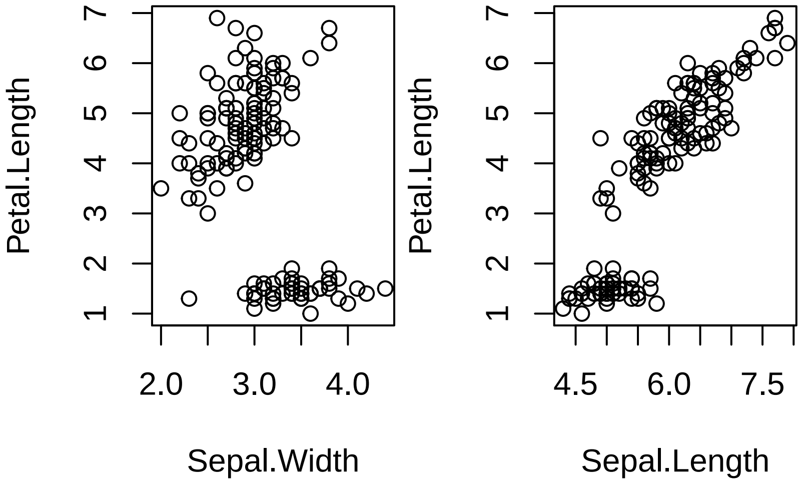
plot

```
plot(iris)
```



plot

```r
plot(Petal.Length ~ Sepal.Width + Sepal.Length)
```



plot

The `type` option determines the type of plot to be produced. The options are listed in the table.

Option	Value
`type = 'p'`	Plots points, is the default option
`type = 'l'`	Plots lines.
`type = 'b'`	Plots points joined by lines.
`type = 'o'`	Points and lines are superimposed.
`type = 'h'`	Vertical lines.
`type = 's'`	Step function, continuous from right.
`type = 'S'`	Step function, continuous from left.
`type = 'n'`	Does not draw the graph but keeps the dimensions

plot

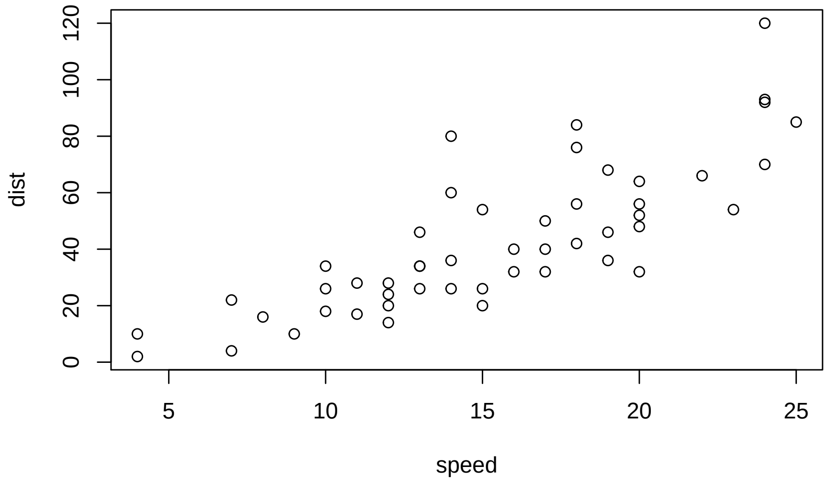
To show the effect that these options have on the plot we will use the `cars` data set, that has the speed and stopping distance for 50 cars, measured in the 1920's.

```
str(cars)
```

```
## 'data.frame':    50 obs. of  2 variables:
##  $ speed: num  4 4 7 7 8 9 10 10 10 11 ...
##  $ dist : num  2 10 4 22 16 10 18 26 34 17 ...
```

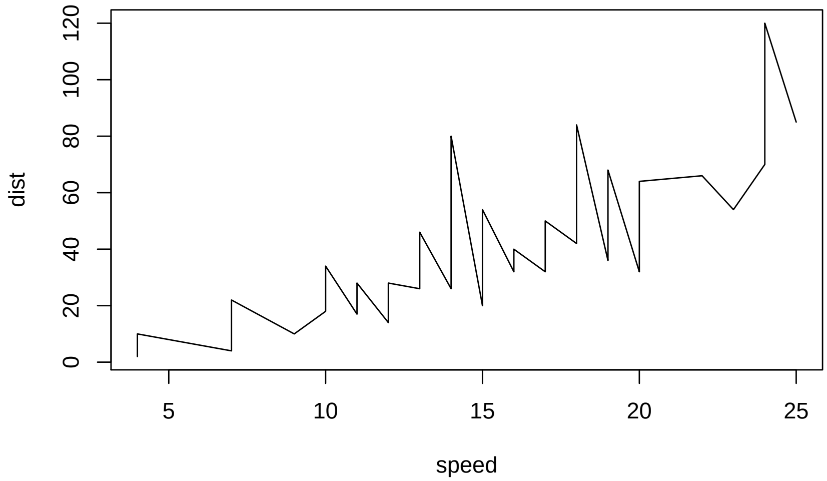
plot

```r
plot(cars, type='p')
```



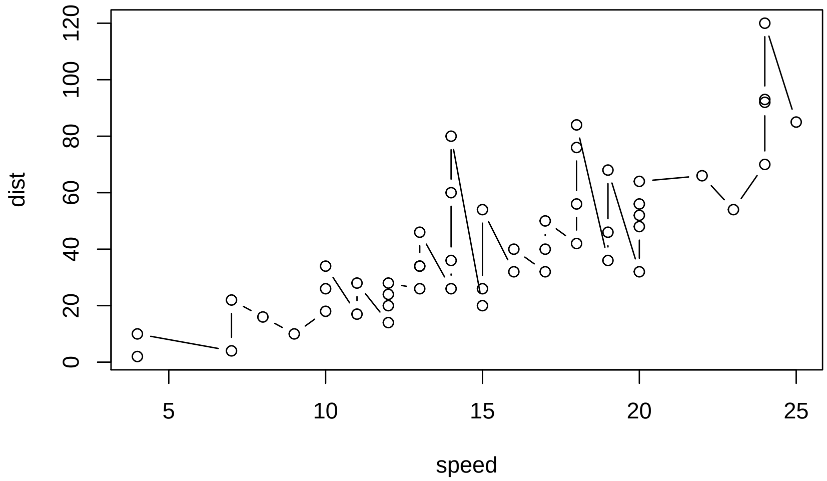
plot

```r
plot(cars, type='l')
```



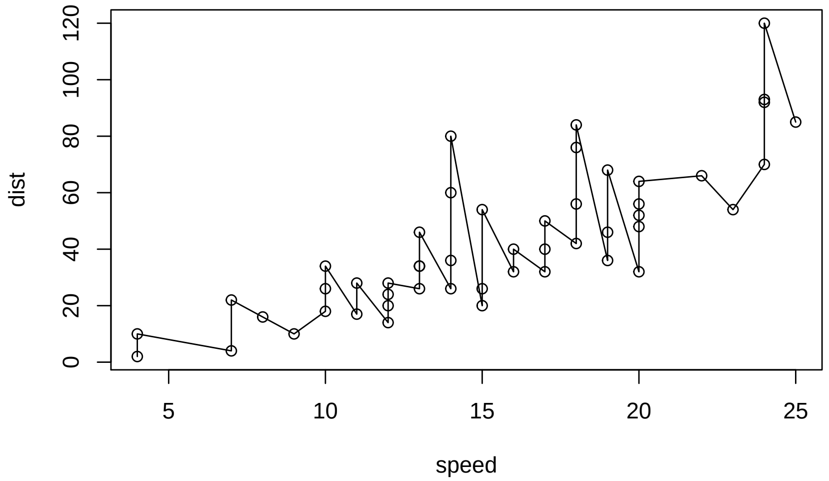
plot

```r
plot(cars, type='b')
```



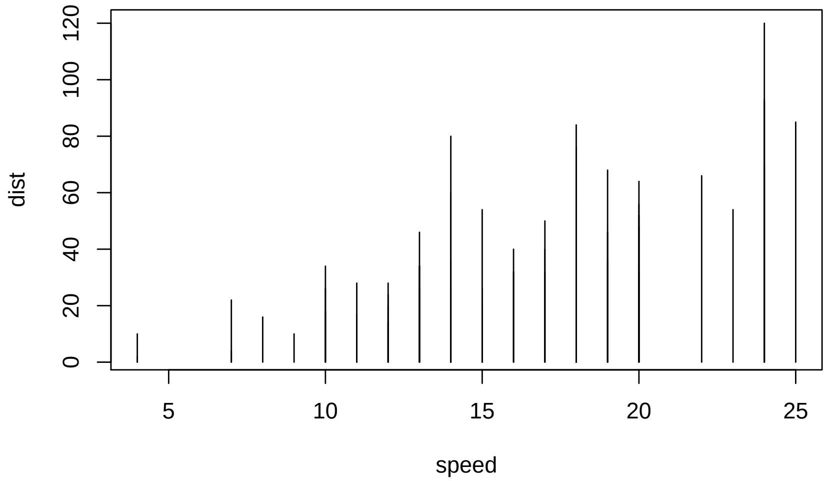
plot

```r
plot(cars, type='o')
```



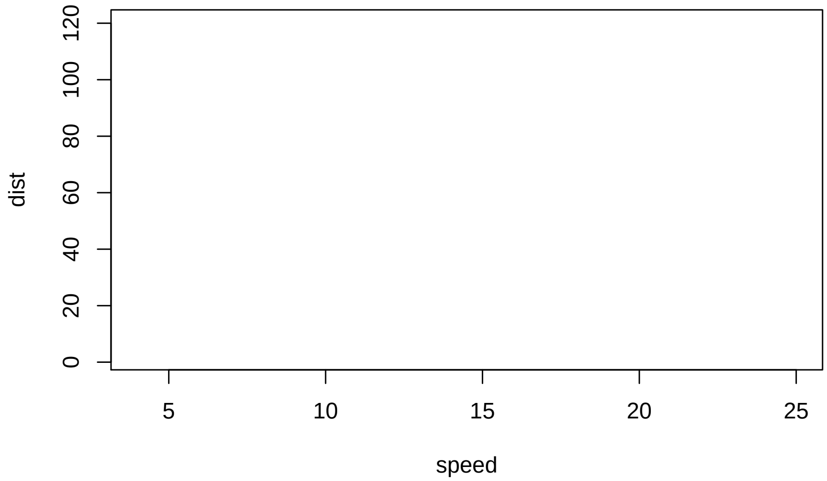
plot

```r
plot(cars, type='h')
```



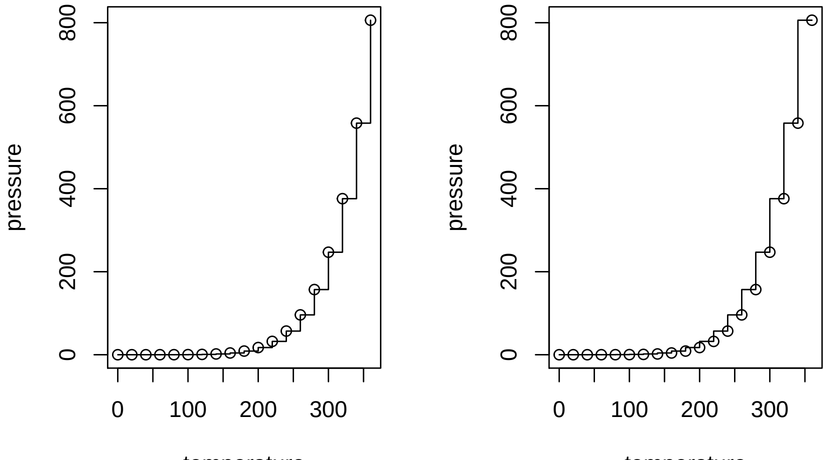
plot

```r
plot(cars, type='n')
```



plot

```
par(mfrow=c(1,2))
plot(pressure,type='s')
points(pressure)
plot(pressure,type='S')
points(pressure)
```

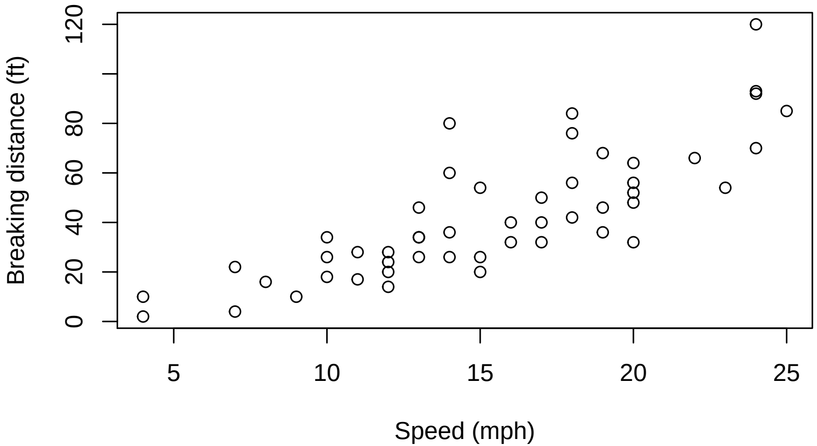


Options in `plot`

xlab and ylab

The options `xlab` and `ylab` give customized labels for the axes. By default, the labels come from the names of the variables in the data object.

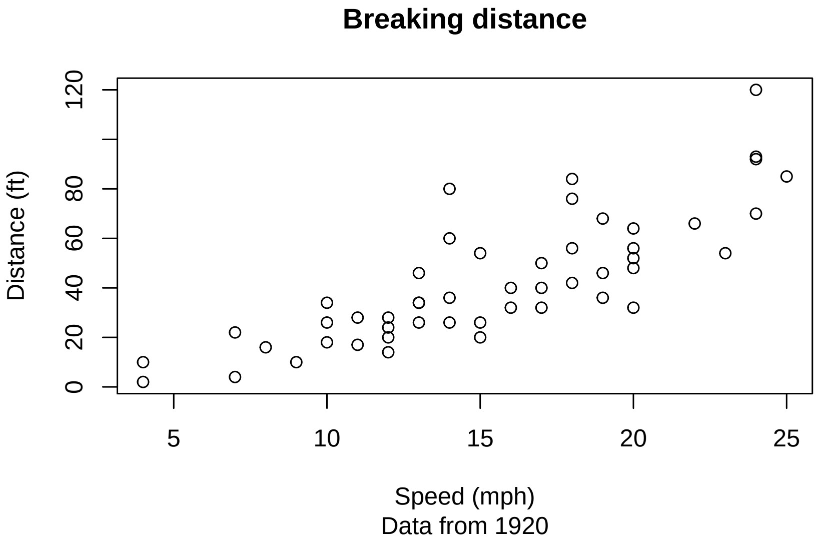
```r
plot(cars, xlab = 'Speed (mph)', ylab = 'Breaking distance (ft)')
```



main and sub

These options give a title and subtitle to the graph.

```
plot(cars, xlab = 'Speed (mph)', ylab = 'Distance (ft)',
     main = 'Breaking distance', sub = 'Data from 1920')
```



`xlim` and `ylim`

These options control the exact range of values on the x and y-axes, respectively.

The value is a vector with two components, indicating the endpoints of the range of values to be plotted.

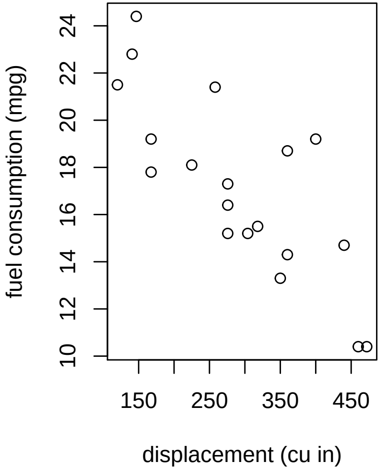
To show how to use them, let's use the data set `mtcars`, that has information on fuel consumption for cars.

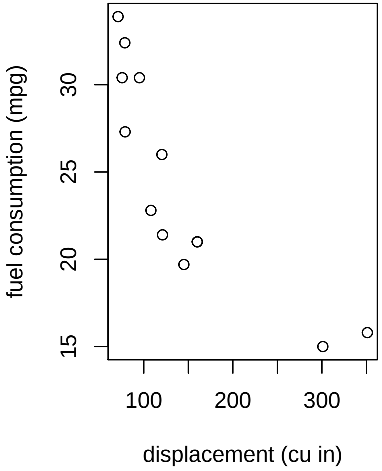
We want to plot `mpg` against `disp` in separate charts according to the value of `am`, which has two values , 0 for automatic and 1 for manual.

We divide the graphic window in two sections. In the next video we explain how to do this.

xlim and ylim

```r
attach(mtcars); par(mfrow=c(1,2))
plot(mpg[am==0] ~ disp[am==0], xlab = 'displacement (cu in)',
     ylab = 'fuel consumption (mpg)')
plot(mpg[am==1] ~ disp[am==1], xlab = 'displacement (cu in)',
     ylab = 'fuel consumption (mpg)')
```





xlim and ylim

Observe that the scales in both axes are different. We are going to use xlim and ylim to make them equal.

We start by getting the range of values for mpg and disp using the function range.

```
(r1 <- range(mpg))
```

```
## [1] 10.4 33.9
```

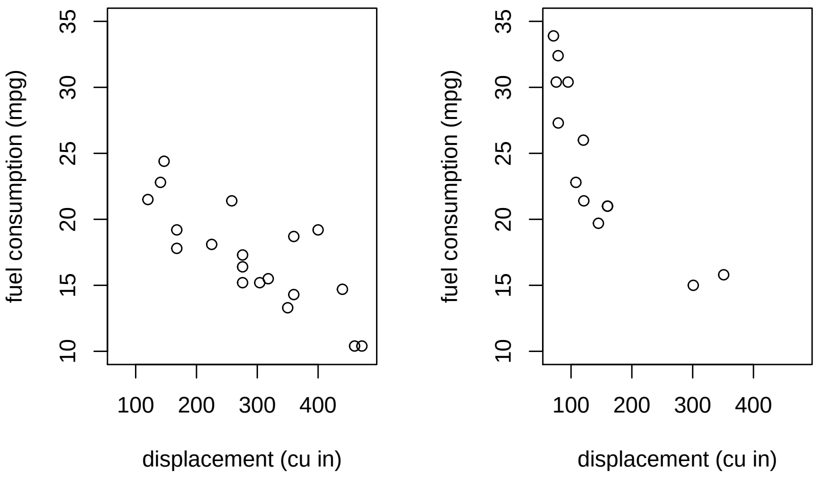
```
(r2 <- range(disp))
```

```
## [1]  71.1 472.0
```

We will set the limits for the y-axis to 10 and 35, and for the x-axis to 70 and 480. This guarantees that all the points will be included.

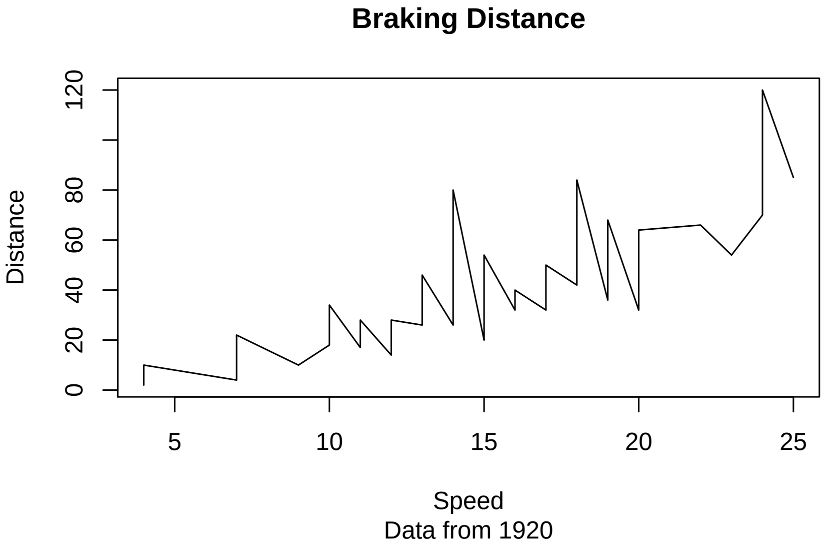
xlim and ylim

```
par(mfrow=c(1,2))
plot(mpg[am==0] ~ disp[am==0], xlab = 'displacement (cu in)',
     ylab = 'fuel consumption (mpg)', ylim = c(10, 35),xlim = c(70, 480))
plot(mpg[am==1] ~ disp[am==1], xlab = 'displacement (cu in)',
     ylab = 'fuel consumption (mpg)', ylim = c(10, 35), xlim = c(70, 480))
```



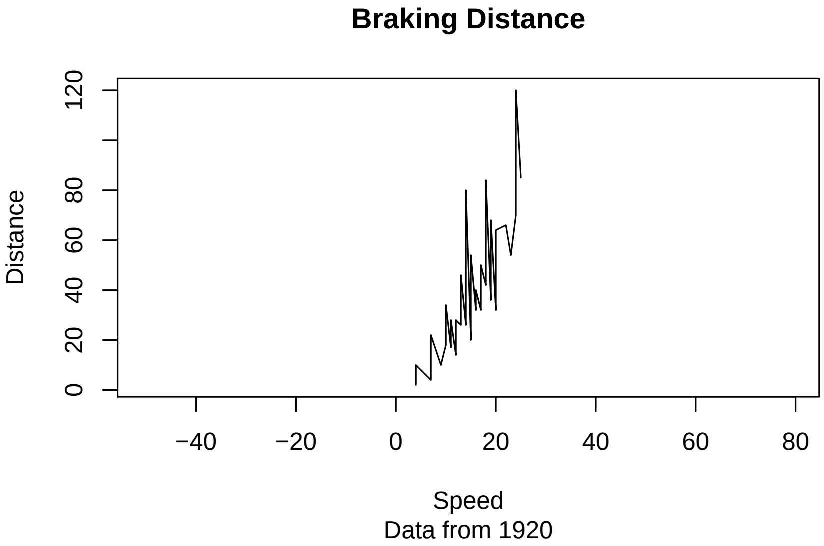
asp

```
plot(cars,type='l',xlab='Speed',ylab='Distance',
main='Braking Distance',sub='Data from 1920')
```



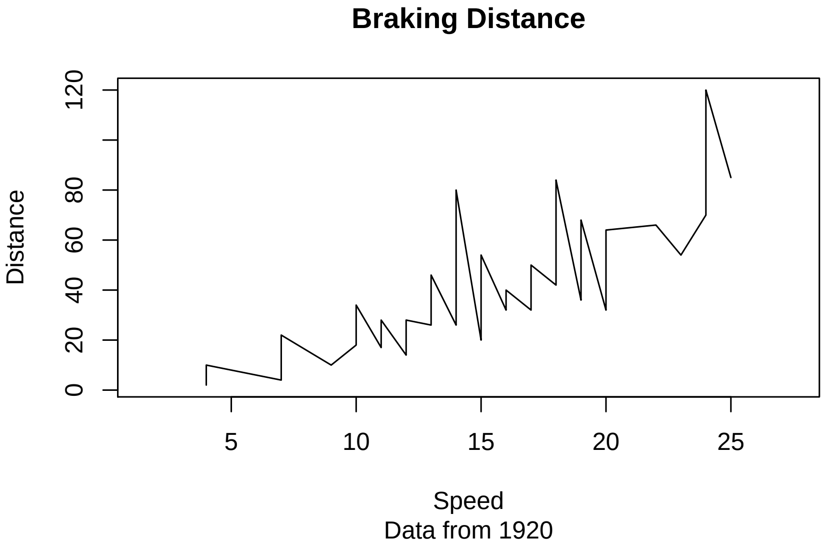
asp

```
plot(cars,type='l',xlab='Speed',ylab='Distance',
main='Braking Distance',sub='Data from 1920', asp=0.5)
```



asp

```r
plot(cars,type='l',xlab='Speed',ylab='Distance',
main='Braking Distance',sub='Data from 1920', asp=0.1)
```



`lty` and `lwd`

These two options control the type of line and the width, respectively, for the lines in the plot. There are six standard types of line.

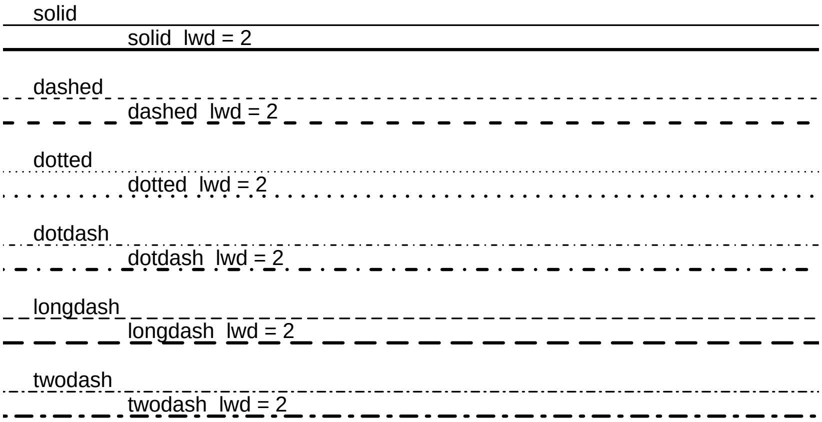


Figure 1: The six different line types, each in two different widths

`pch`

The parameter `pch` controls the shape of points in scatterplots and dotplots.

There are 26 standard shapes, numbered from 0 to 25, and 1 is the default.

The following graph presents all the standard shapes.

For shapes 0 to 20, the option ‘col’ controls the color. For shapes 21 to 25, the filling color can be chosen with the option `bg` while `col` controls the border color.

`pch`

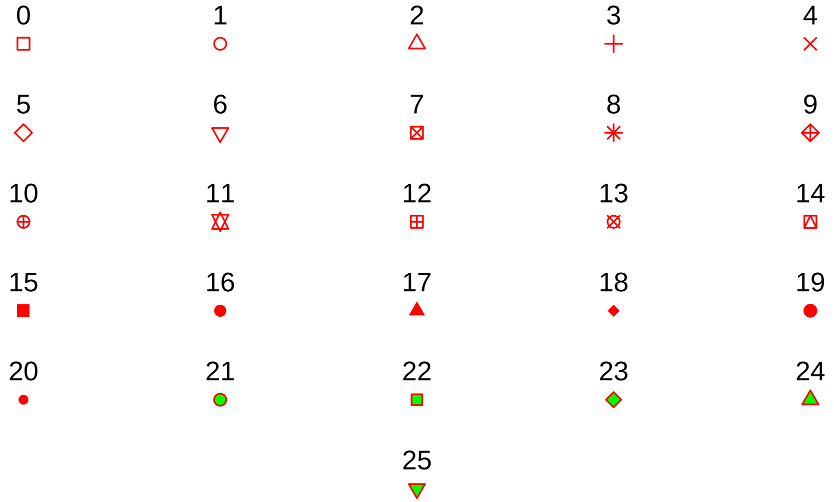


Figure 2: Standard point symbols and their codes

Color

The option `col` controls the color of the plotting symbol or line.

Colors can be specified in different ways. Perhaps the simplest is with the color name within quotation marks (e.g., 'blue' or 'green').

There are hundreds of named colors in `R`. A list is produced by the function `colors()`.

A pdf with the named colors can be found in http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf and a cheatsheet in https://www.nceas.ucsb.edu/sites/default/files/2020-04/colorPaletteCheatsheet.pdf.

Color

Also, eight particular colors can be set by numbers.

These form the color palette that can be accessed by numbers. The command `palette()` lists the colors in the active palette.

Numbers wrap around after 8, so 9 is equal to 1. These colors are

```
## [1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
## [8] "gray"
```

1 black
2 red
3 green3
4 blue
5 cyan
6 magenta
7 yellow
8 grey

Figure 3: Colors by numbers in R

Color

The colors in the palette can be modified using the function `palette()` with an argument. One easy way to do this is to use `rainbow()` to create a new selection of colors.

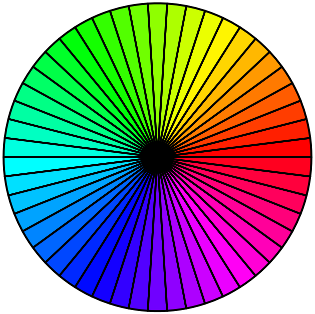


Figure 4: Palette of colors obtained with the function rainbow(50)