Simple plots in R

David Jorgensen

R plot function

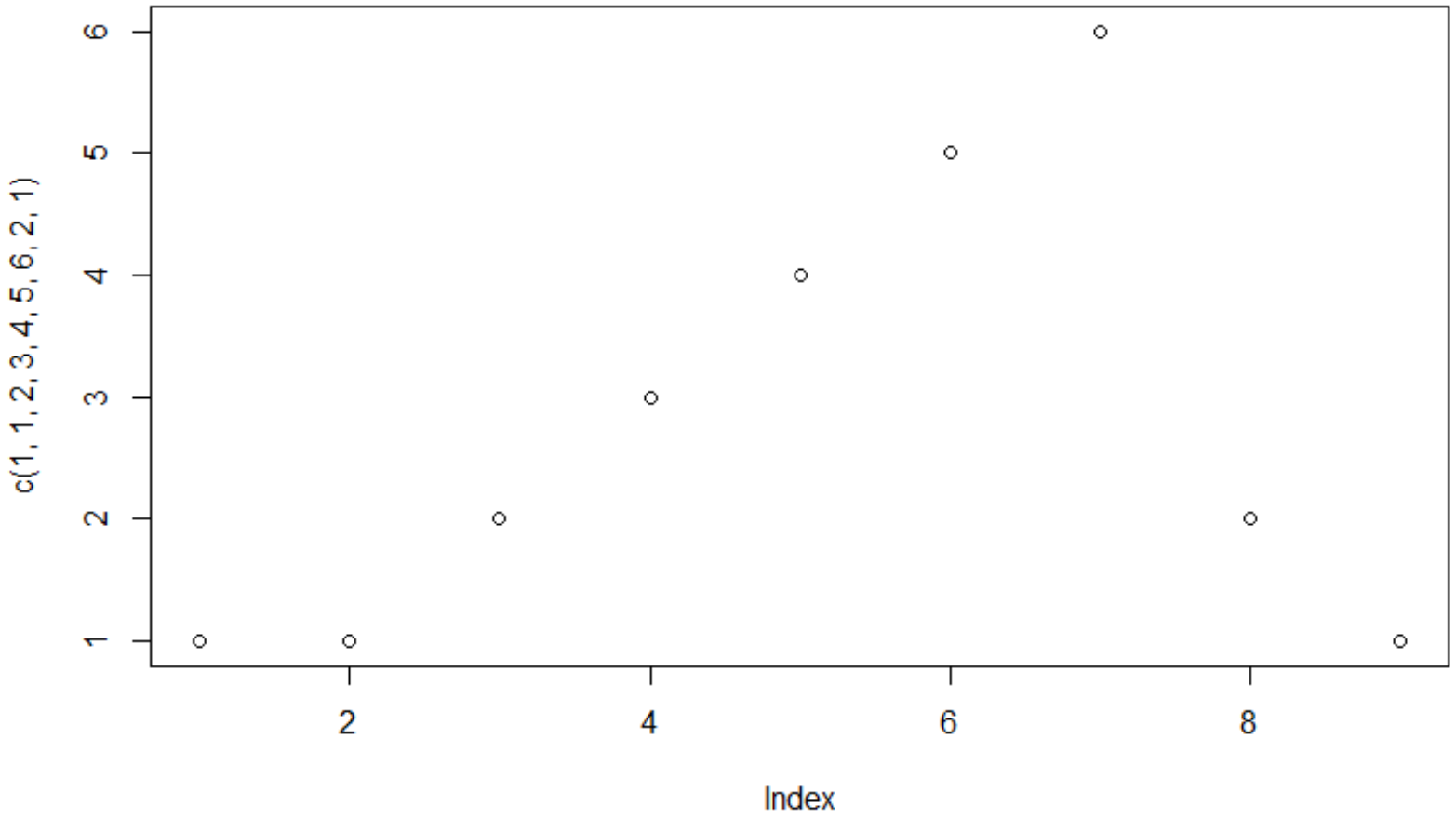
- The most used plotting function in R is **plot()**
- It can produce many types of plots depending on the data passed to the function

Simple case – plotting a vector

Make a vector c(1,1,2,3,4,5,6,2,1) and pass it to the plot function in R

What do you get?

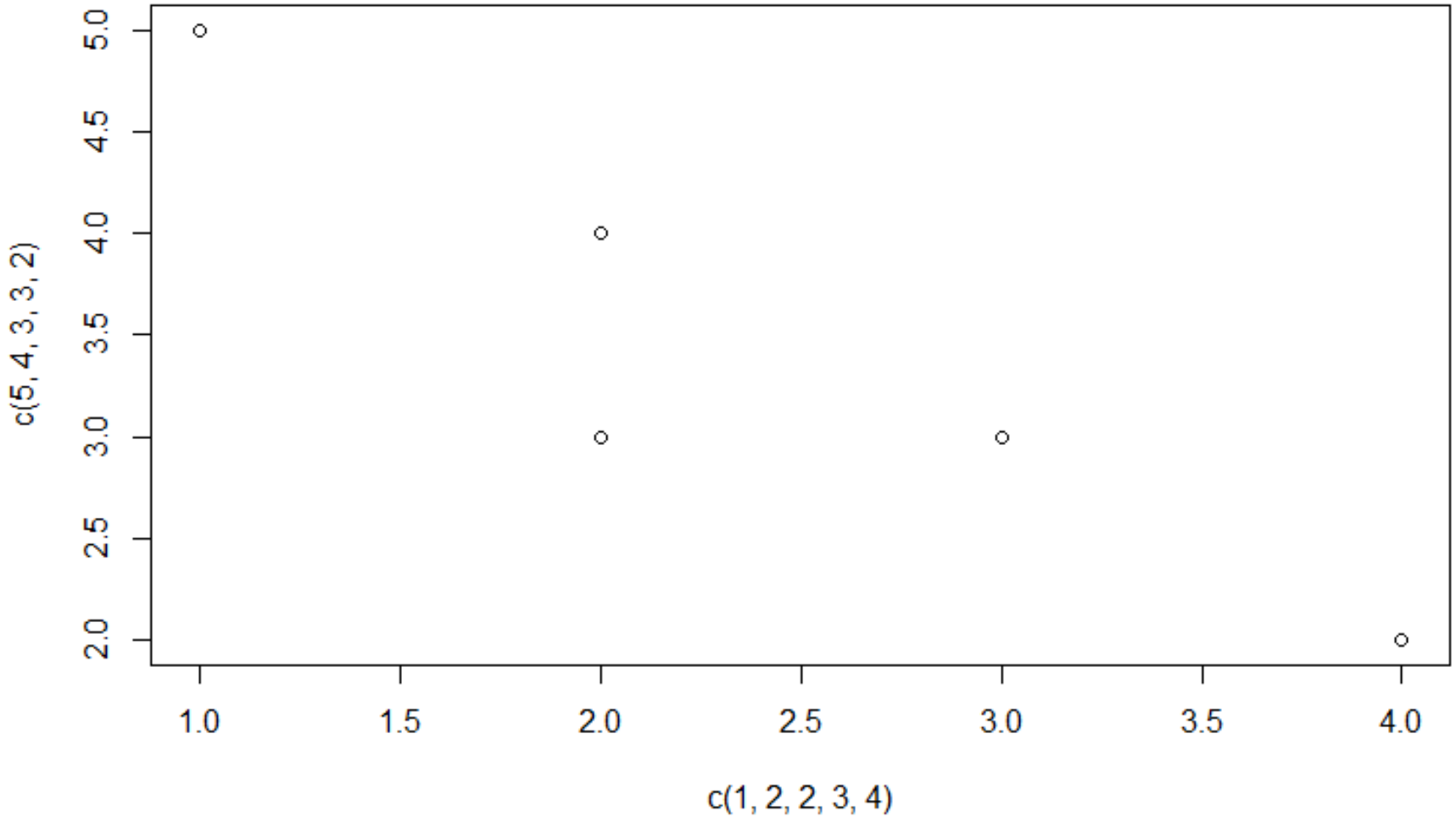
Should look like this:



Plotting 2 vectors:

Try `plot(c(1,2,2,3,4), c(5,4,3,3,2))`

You should get a plot of one vector against the other where the position of each number determines the number it is plotted against:


c(5, 4, 3, 3, 2)
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
c(1, 2, 2, 3, 4)

Creating some data to plot

We will use the seq() and sin() functions to create a sine curve between -3 and 3 which we can plot.

Create an object named x with a sequence from -3 to 3 with increments of 0.1

```
x <- seq(-3,3,0.1)
```

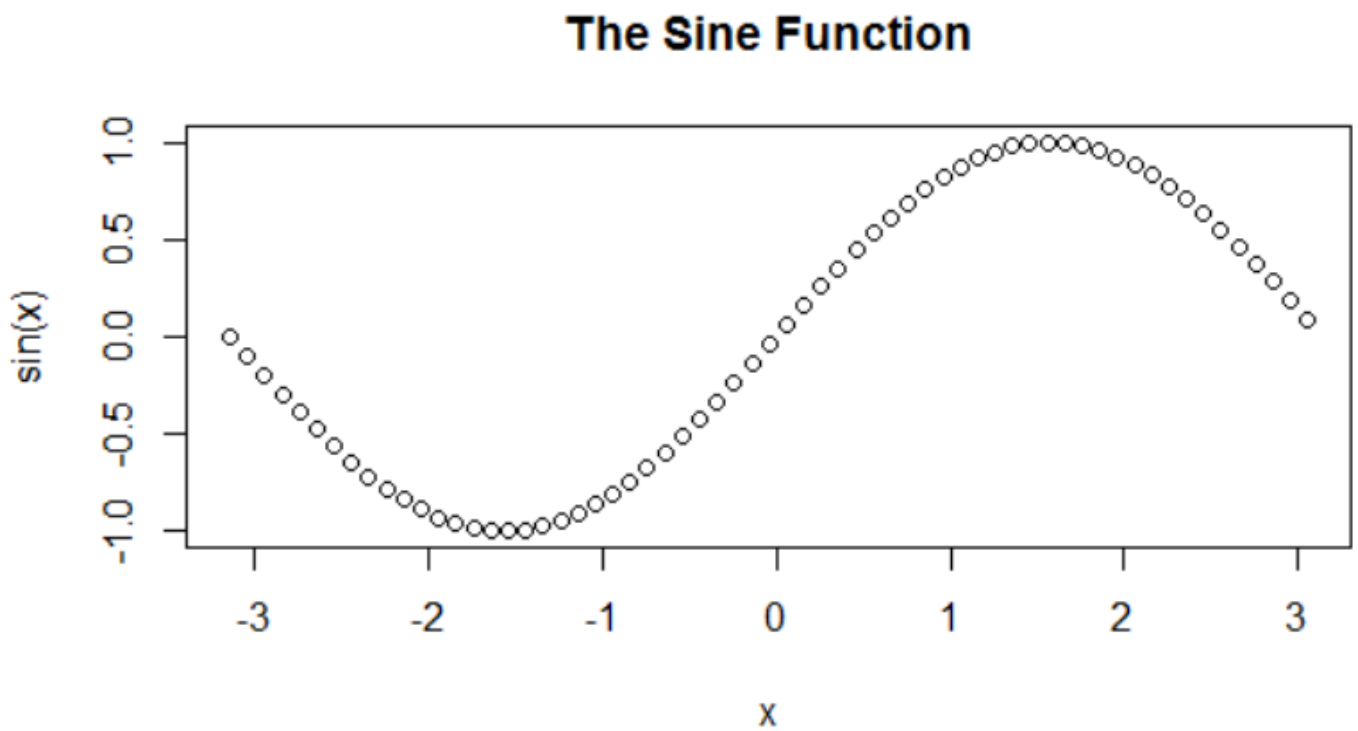
We will now plot a sine curve at these points using plot:

```
plot(x, sin(x))
```

Adding Titles and Labeling Axes

We can add a title to our plot with the parameter `main`. Similarly, `xlab` and `ylab` can be used to label the x-axis and y-axis respectively.

```
plot(x, sin(x),
main="The Sine Function",
ylab="sin(x)")
```



Changing plot type

We can see above that the plot is of circular points and black in color. This is the default color.

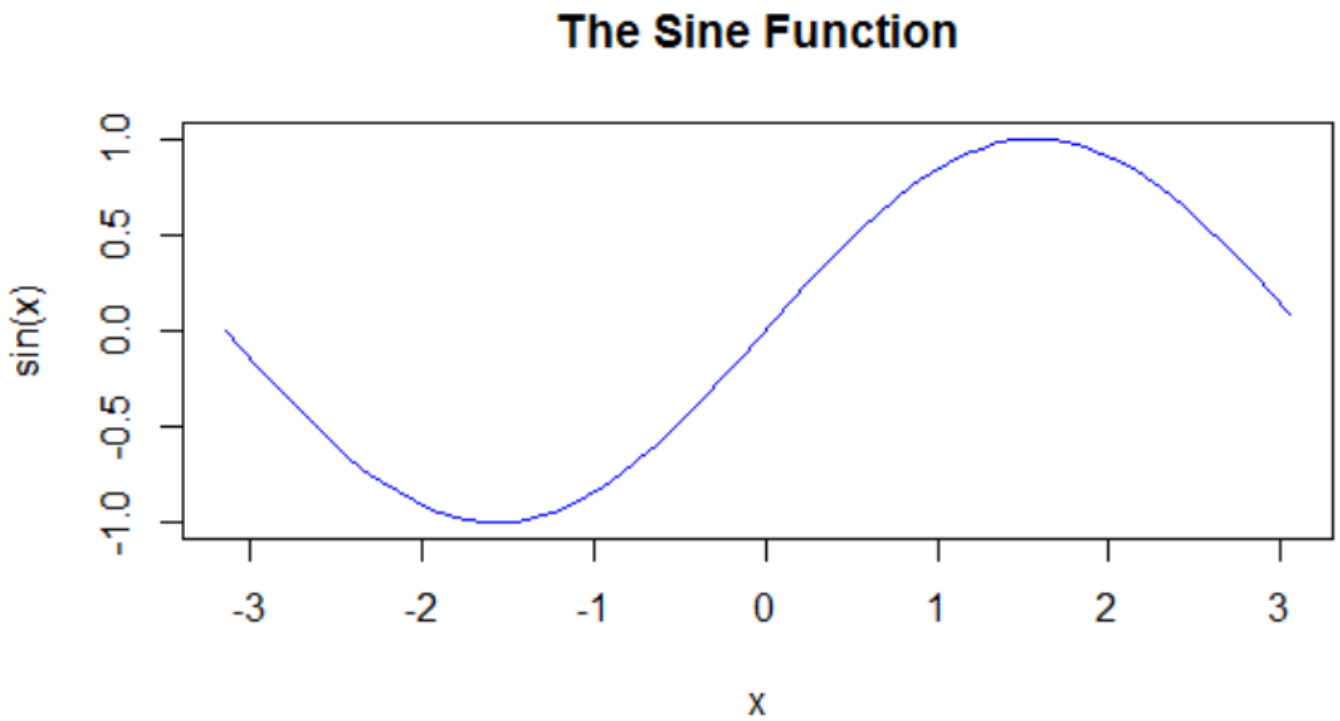
We can change the plot type with the argument `type`. It accepts the following strings and has the given effect.

```
"p" - points
"l" - lines
"h" - histogram-like vertical lines
"n" - does not produce any points or lines
```

Try changing your plot to a line:

Similarly, we can define the color using `col`.

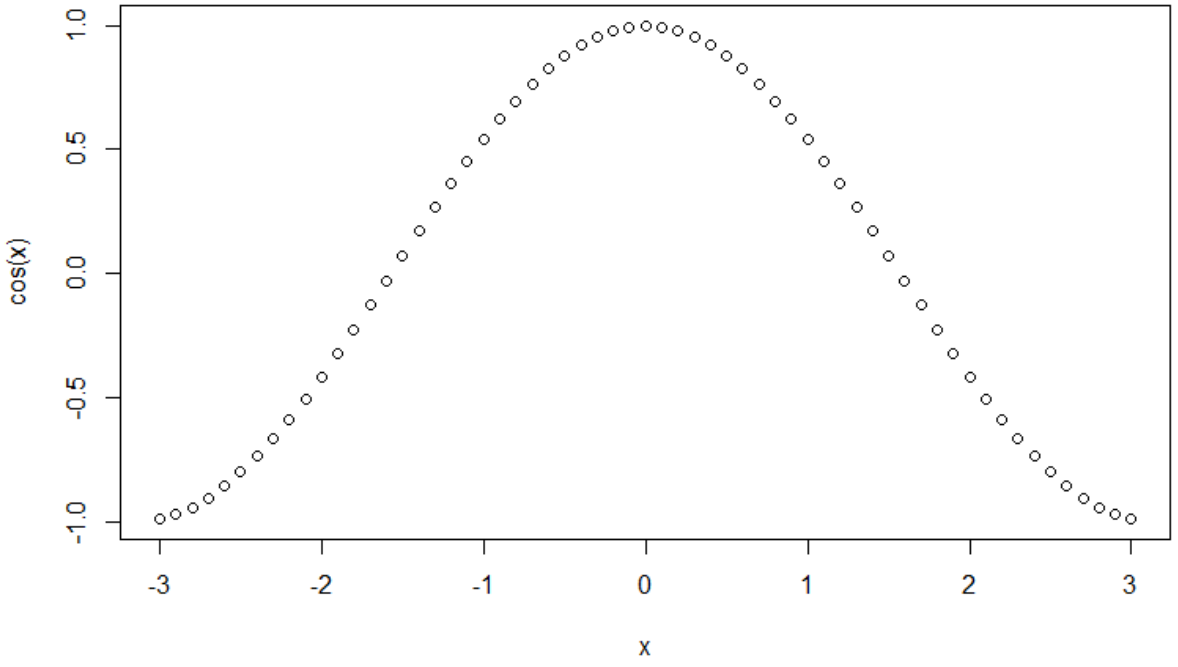
```
plot(x, sin(x),
main="The Sine Function",
ylab="sin(x)",
type="l",
col="blue")
```



Plotting multiple lines

Calling plot multiple times will write over the current plot in the plots window:

```
plot(x, sin(x))

plot(x, cos(x))
```

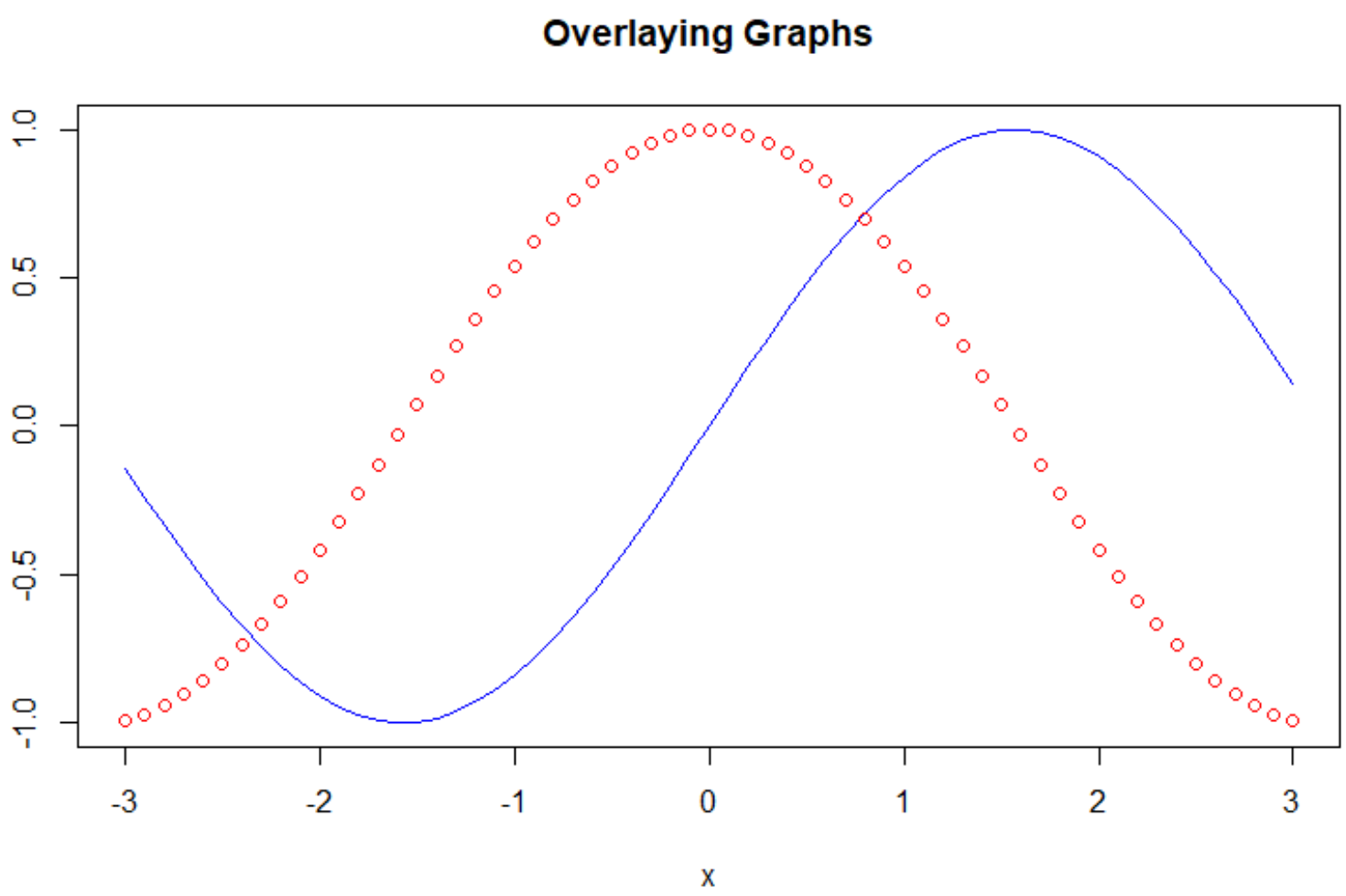


Plotting multiple lines

- We can use `lines()` rather than the plot function to add additional lines we want to plot to the graph:

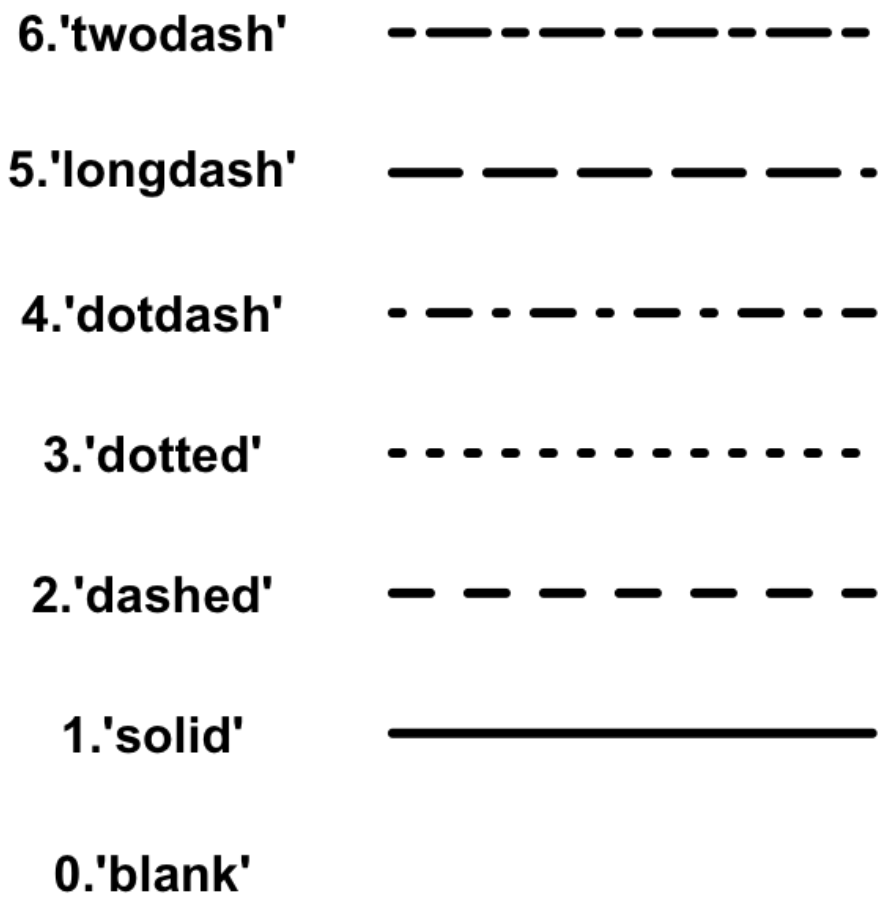
```
plot(x, sin(x),
main="Overlaying Graphs",
ylab="", type="l", col="blue")

lines(x,cos(x), col="red")
```

- We can use points in a similar way to plot points over an existing plot, try the above code with `points()` rather than `lines()`


Overlaying Graphs
1.0
0.5
0.0
-0.5
-1.0
-3
-2
-1
0
1
2
3
x

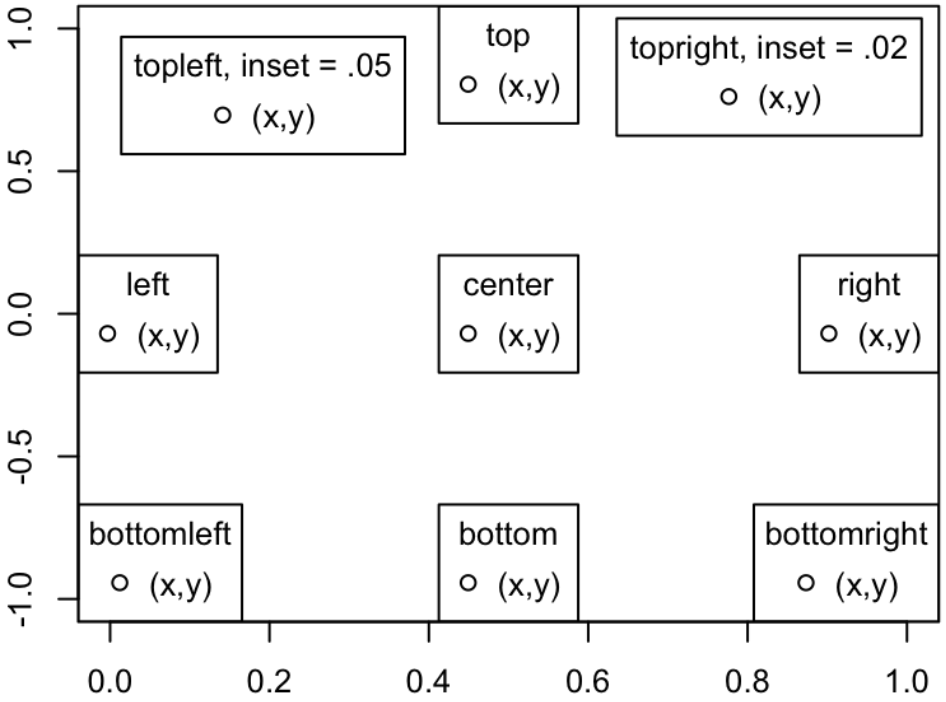
Changing the linetype

You can change the type of line plotted using the `lty` argument in lines or plot, try changing your added points to a line and plotting with a different linetype.



Add a legend to the plot

```
plot(x, sin(x),
main="Overlaying Graphs",
ylab="", type="l", col="blue")

lines(x,cos(x), col=“red”,
lty=2)

legend(“topleft”,
legend=c(“sin(x)”,“cos(x)”),
col=c(“blue”, “red”),
lty=c(1,2)
```

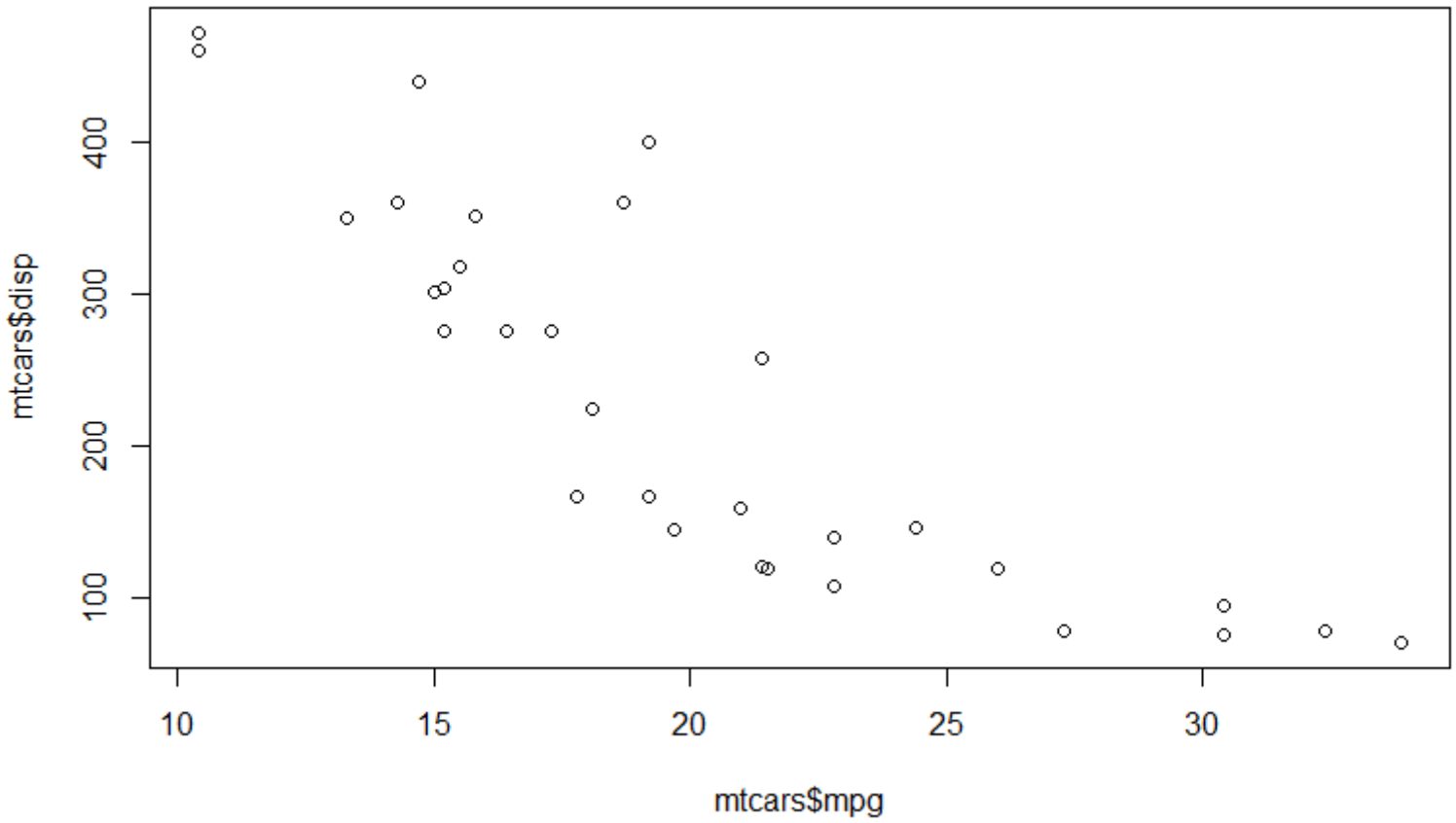


More plotting functions

- We will use the `mtcars` sample dataset in R to try out some more plotting functions.
- Run `View(mtcars)` to see the data in Rstudio

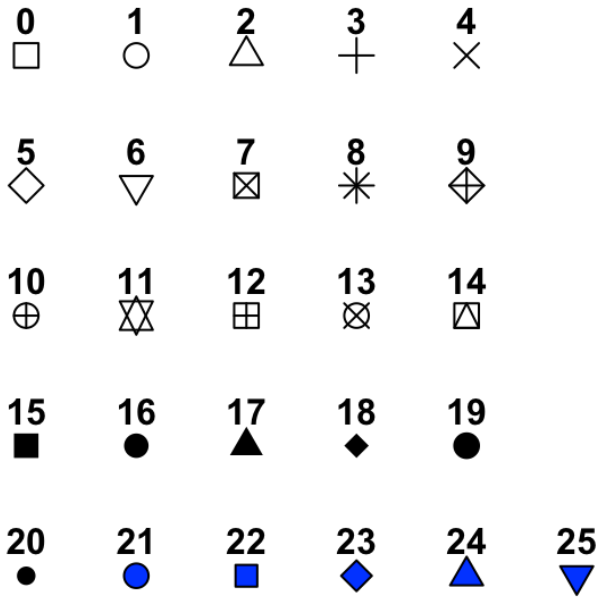
Changing point shape

Plot `mtcars$mpg` against `mtcars$disp` in R, it should look like this:



Changing point shape

we can change the shape of the points using the `pch` argument in the plot function, the point shapes are specified by numbers:

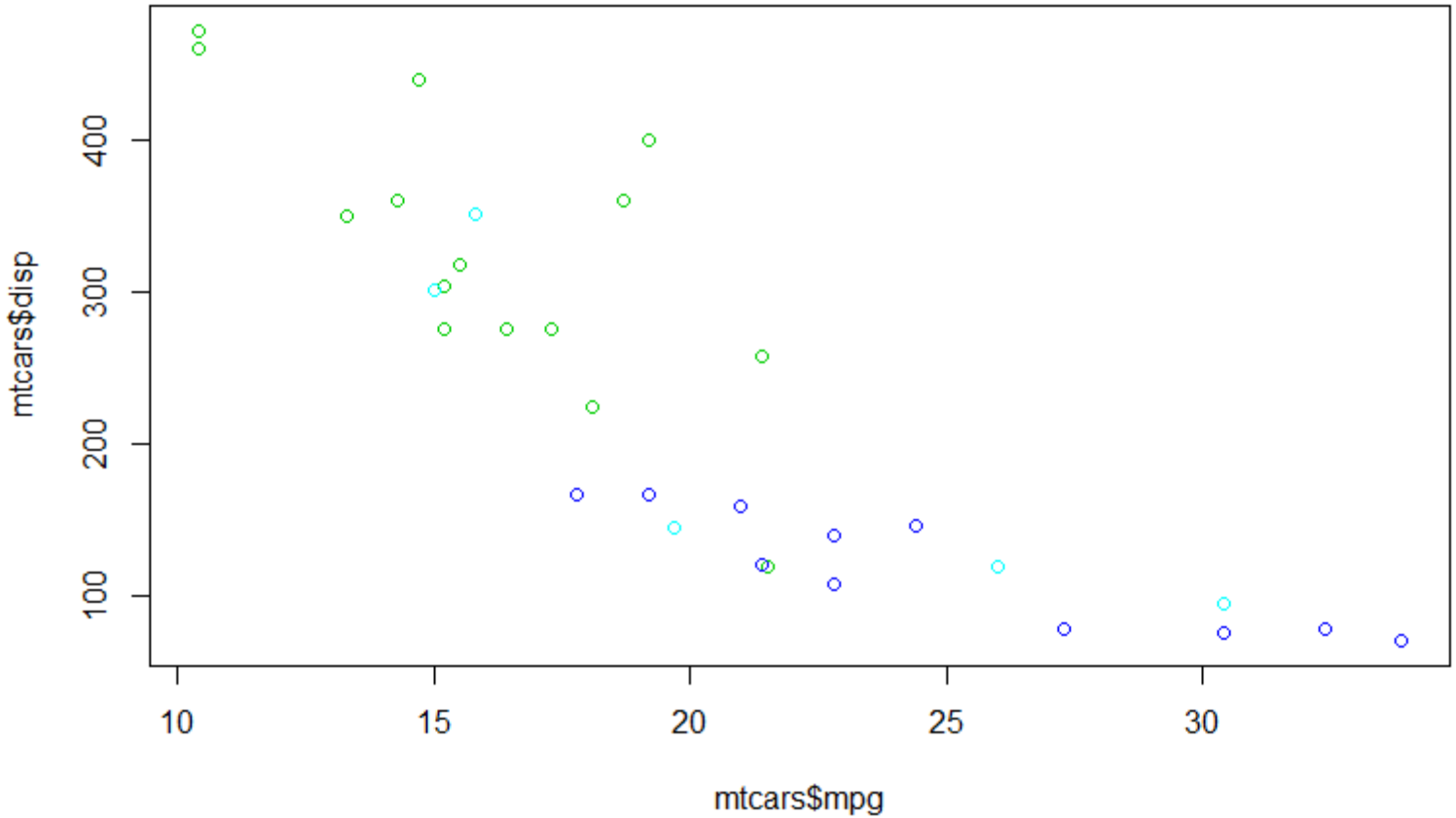


Try changing the shape of the points in your data

Colour by a factor

Using the col argument we can colour the points by other columns in our data, for example:

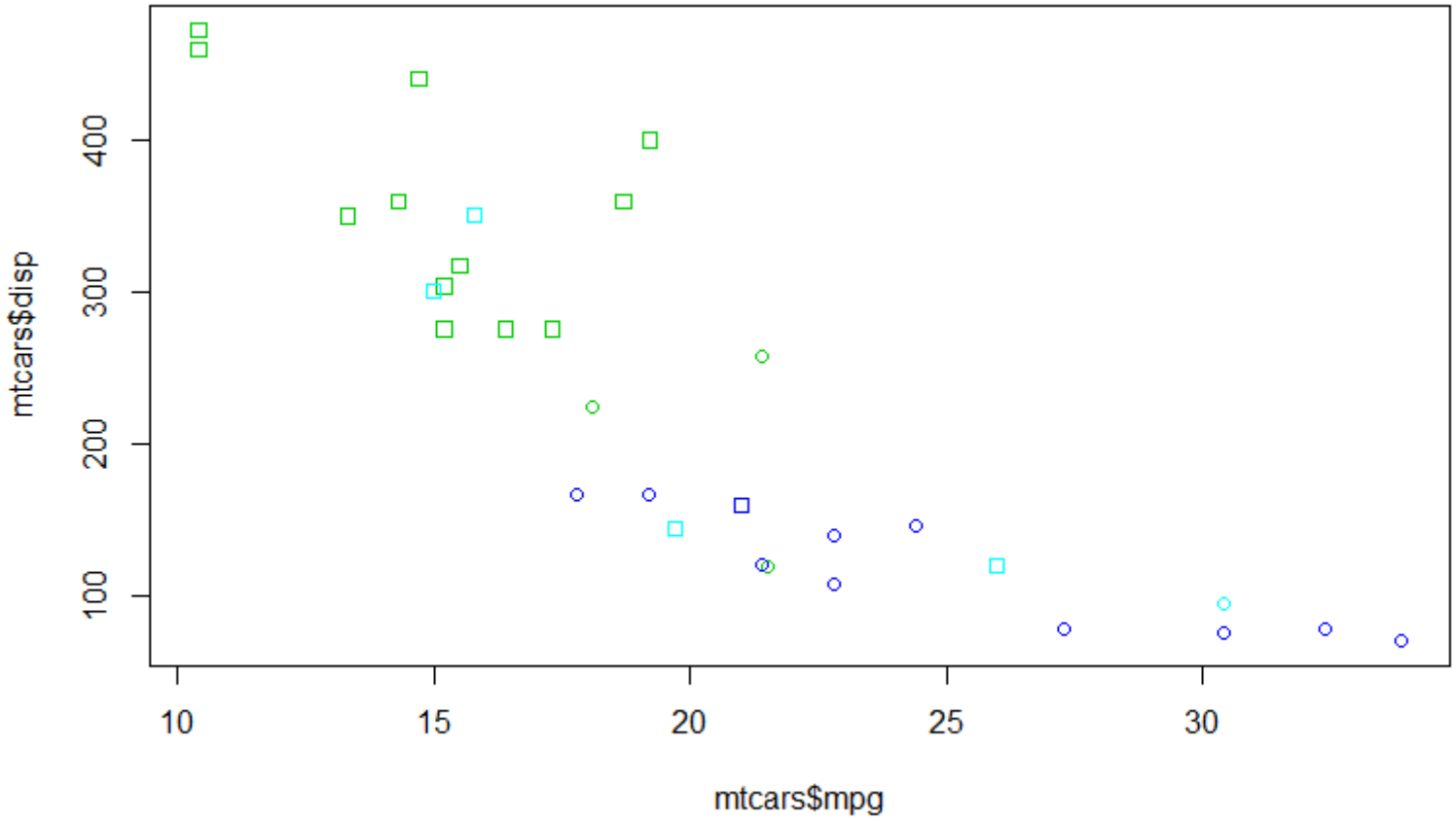
```
plot(mtcars$mpg, mtcars$disp, col=mtcars$gear)
```



Shape by a factor

```
plot(mtcars$mpg, mtcars$disp, col=mtcars$gear)
```

Use the pch and col arguments to produce a plot coloured by gear with point shapes from vs:



Label the plot

Use xlab and ylab to label your axes miles per gallon and displacement

hist()

- The `hist` function can be used to plot simple histograms of the frequency of a parameter in your dataset or a vector.
- Choose a column in mtcars and plot it with the `hist` function
- Hist automatically bins your data in to groups with similar values which are then summed in to a single bar. To change the number or position of these bins we can use the breaks argument in hist:

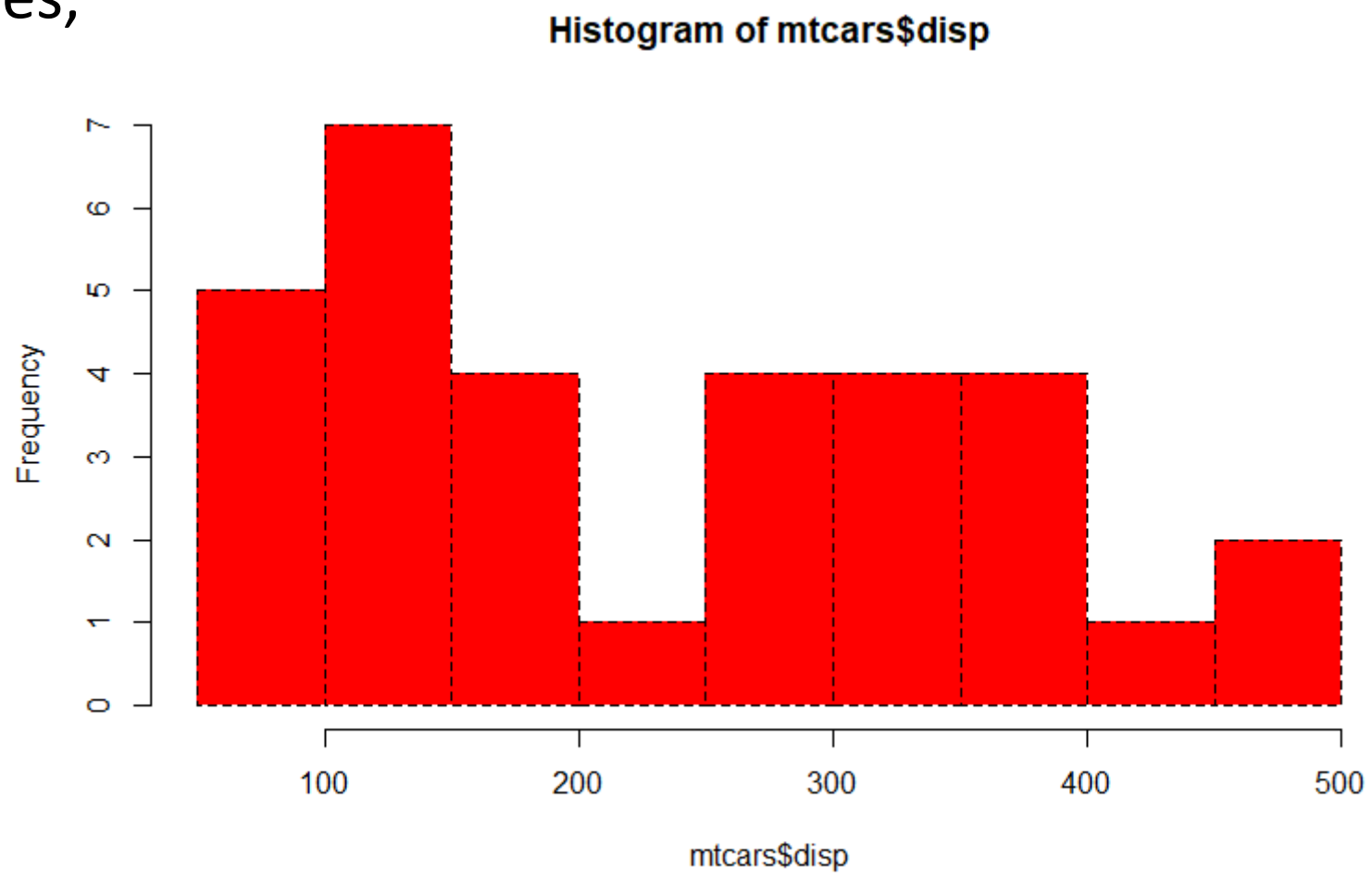
```
hist(mtcars$drat, breaks=10)
```

```
hist(mtcars$drat, breaks=c(min(mtcars$drat),4.0,4.5, 5.0,
max(mtcars$drat))))
```

Try running this code in R, see how changing the values in the breaks argument changes the position of the break points

Arguments to hist()

Using col and lty plot a histogram of mtcars$disp with red bars and dotted lines,



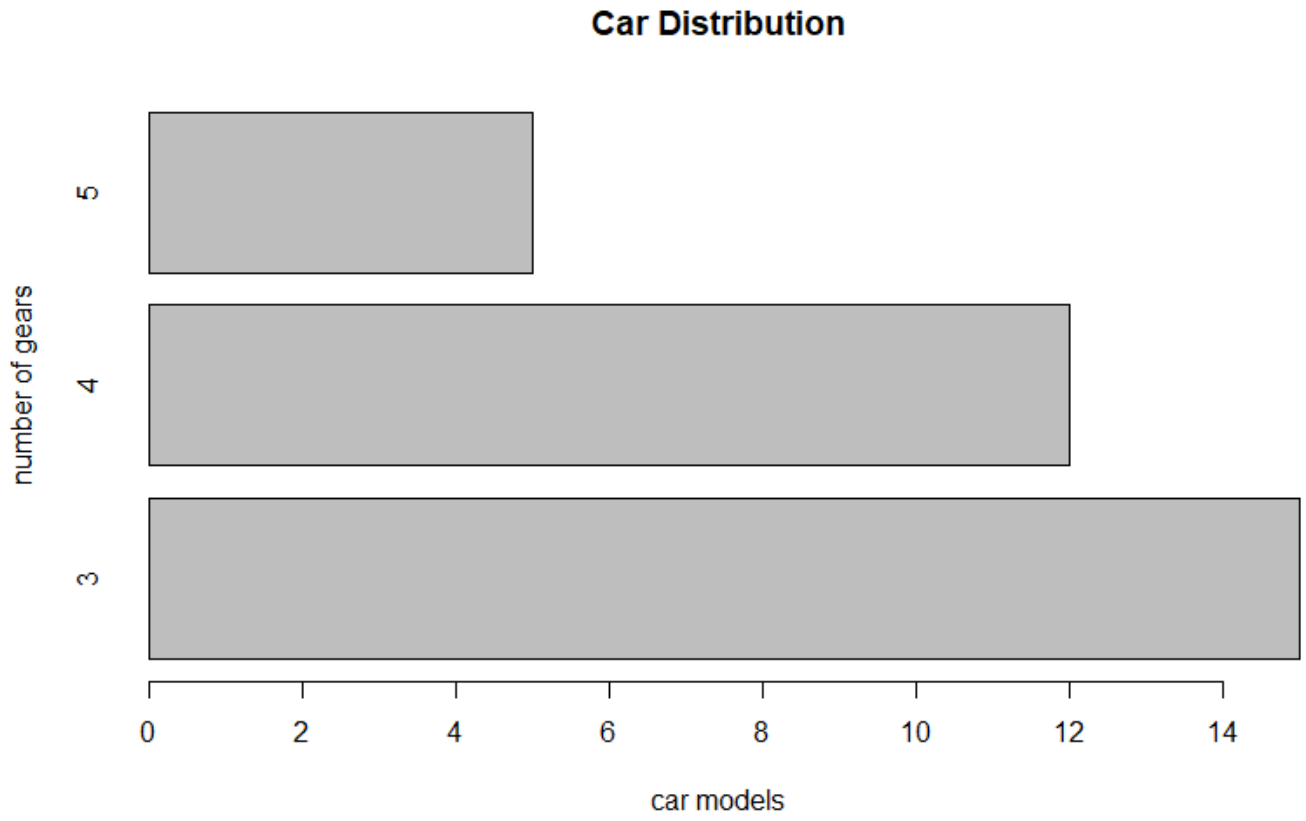
Barplots

We will make a simple barplot of the counts of the number of cars with different numbers of gears:

```
counts <- table(mtcars$gear)
barplot(counts, main="Car Distribution",
        xlab="Number of Gears")
```

Barplots

We can change this to a horizontal bar plot using the argument horiz=TRUE, add this to your bar plot, remember to change the x axis label as we are switching the axes.



Stacked barplots

Calculate counts by gears and vs using `table()` as before, separate the two columns of interest with a comma.

We can then plot a stacked barplot of gears coloured by vs

```
barplot(counts, main="Car Distribution",
  xlab="Number of Gears", col=c("darkblue","red"),
  legend = rownames(counts))
```

Change this stacked barplot to a grouped barplot using the argument beside=TRUE

Boxplots

Boxplots are used to show a central estimate and range of data, in R a standard boxplot shows the median and IQR of the data with whiskers extending 1.5x IQR further out

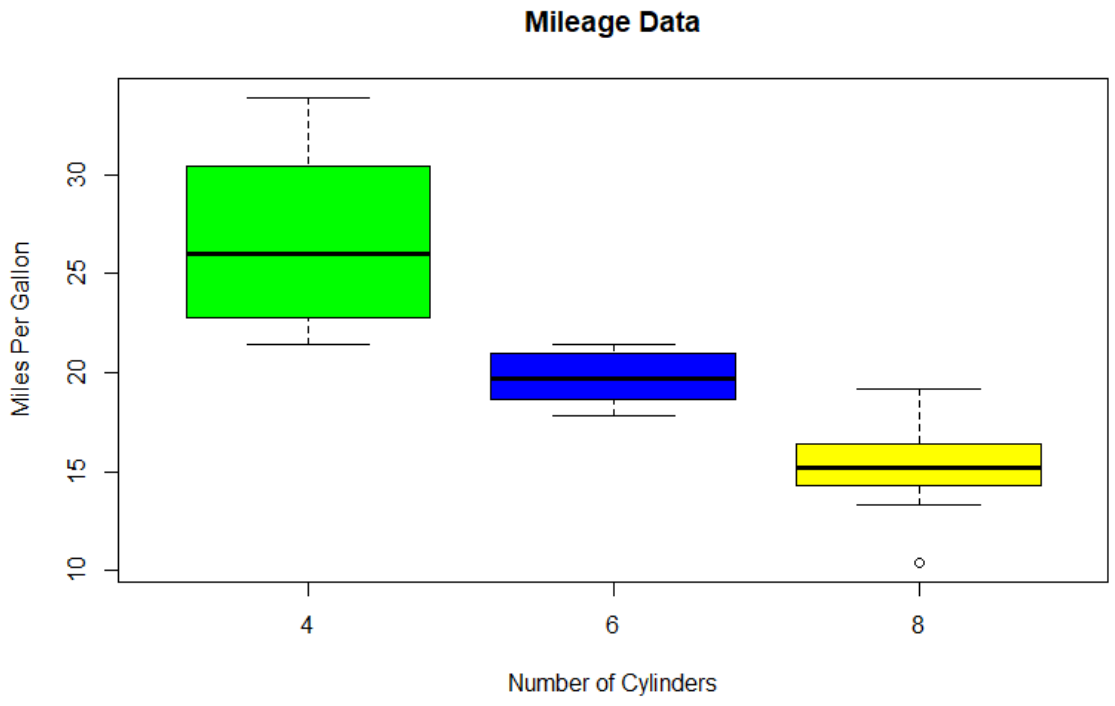
Plot a boxplot of mpg against cyl:

```
boxplot(mpg ~ cyl, data = mtcars,
 xlab = "Number of Cylinders",
 ylab = "Miles Per Gallon",
 main = "Mileage Data")
```

This also introduces a different syntax you can use for plotting by specifying your data using the data argument and using ~ to show which data columns you want to plot against eachother

Boxplots

Change the colour of you boxplots using col to set them to green, blue and yellow (Hint: you can use c() with col)



Multiple plots together

We can adjust the plotting area in R using the par() function.

par(mfrow=c(2,3) gives a plot area for 6 plots arranged in 2 rows and 3 columns.

Write your own code to plot 2 of the plots you have made previously side by side (2 columns). You need to run par before you start plotting.

Saving plots to file

In Rstudio you can save plots that are shown in the plot window to file by clicking export and selecting a file type.

You can also save plots directly in R to be more accurate with the final size and resolution. R supports PNG, JPG, SVG and PDF plotting directly from the console

Saving plots to file

To save to file you will need to give a file name within the png function:

```
png(“plot.png”)
```

Followed by your plot:

```
boxplot(mpg ~ cyl, data = mtcars,
 xlab = "Number of Cylinders",
 ylab = "Miles Per Gallon",
 main = "Mileage Data")
```

And then:

```
dev.off()
```

Saving plots to file

Within this function you can specify the size and resolution of your plot with `width`, `height` and `res`. The default width and height are in pixels. This can be changed with the `units` argument to `“in”` (inches) `“cm”` centimetres or `“mm”` millimetres.

Plotting some polio data

The built in plot functions in R are good for quickly visualising data.

We will load in the non-polio AFP data from 2015 to mid 2019 and use this to demonstrate some plotting

Importing the data

```
library(dplyr)
library(dataPakistan)

data_name <- system.file(package = "dataPakistan", "extdata") %>%
dir(full.names = TRUE) %>% .[5]

NPAFP <- readxl::read_excel(data_name, sheet = 1) %>% as.data.frame()
```

YOU WILL GET WARNINGS, THESE ARE OK TO IGNORE!

OR

Use the built in Rstudio file browser to select the file if you have saved the data from the memory stick

It is in data/ExcelData

The `table()` function

Table will calculate the number of times each character string or number occurs in a column of data or simple vector.

Try: `table(c(2012,2012,2013,2014))`

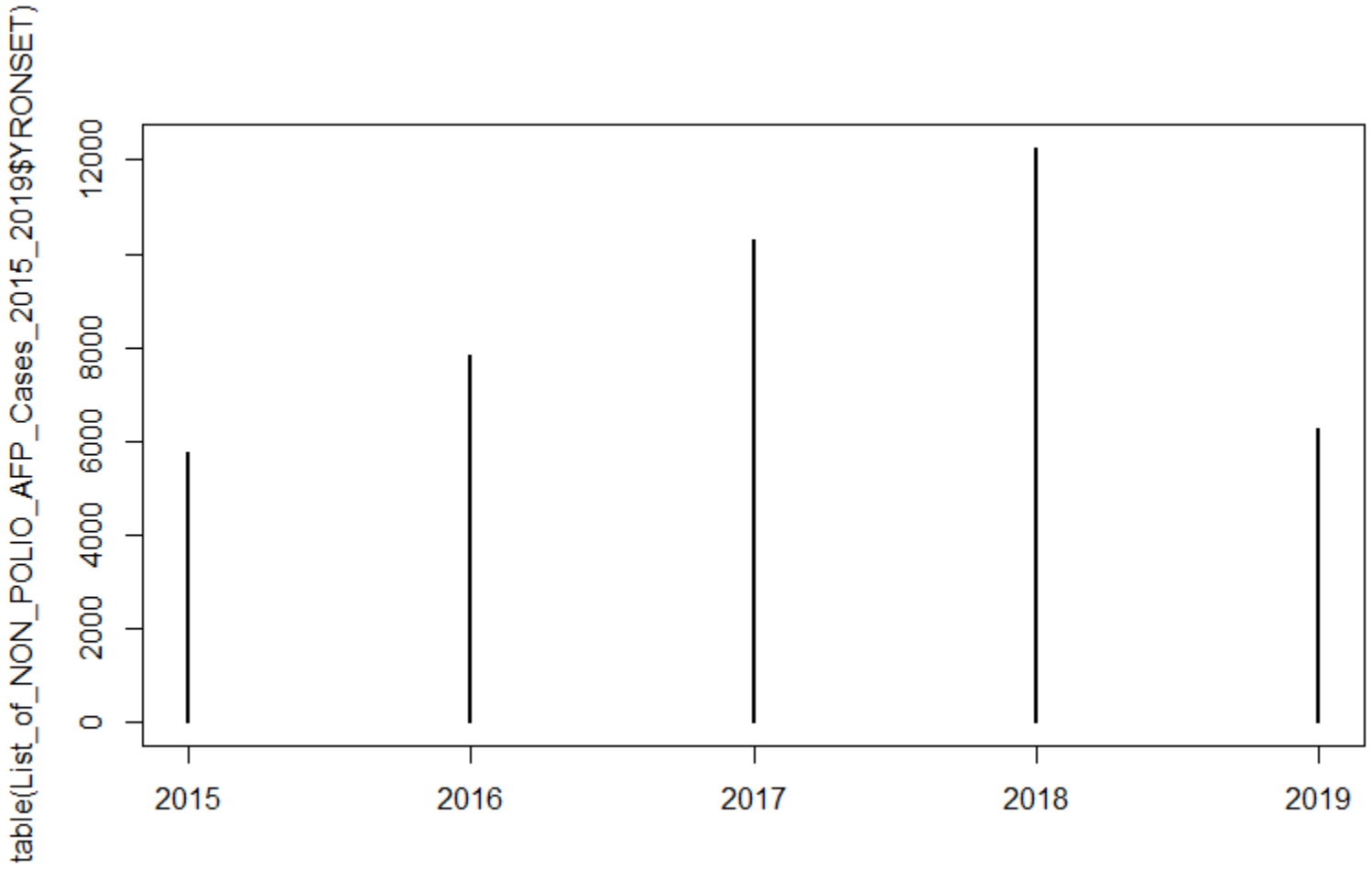
You will receive a table with the counts of each of these years.

We will now apply this to the years column in the NPAFP dataset

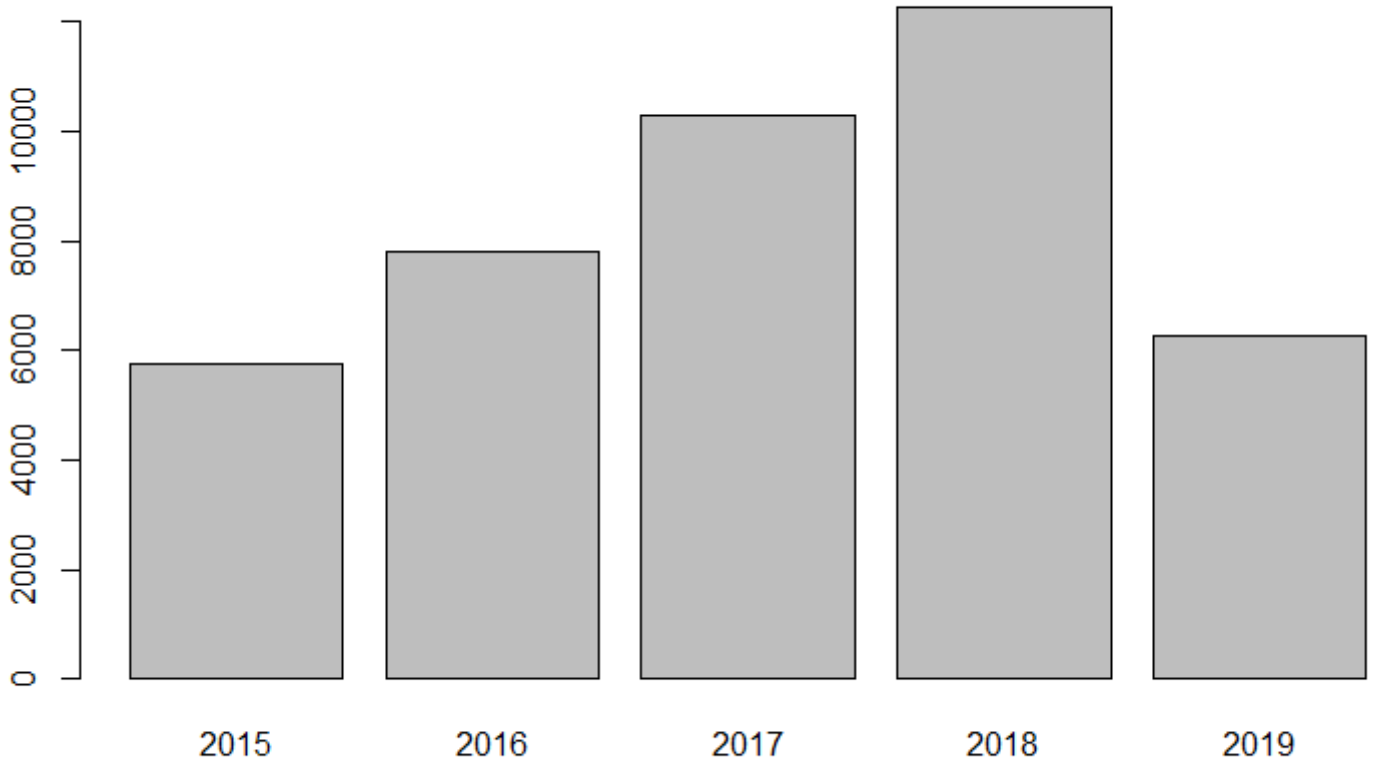
```
table(NPAFP$YRONSET)
```

Save this table to an object NPAFP_YR using <- or =

If we plot this object using `plot()` we will get the following output:

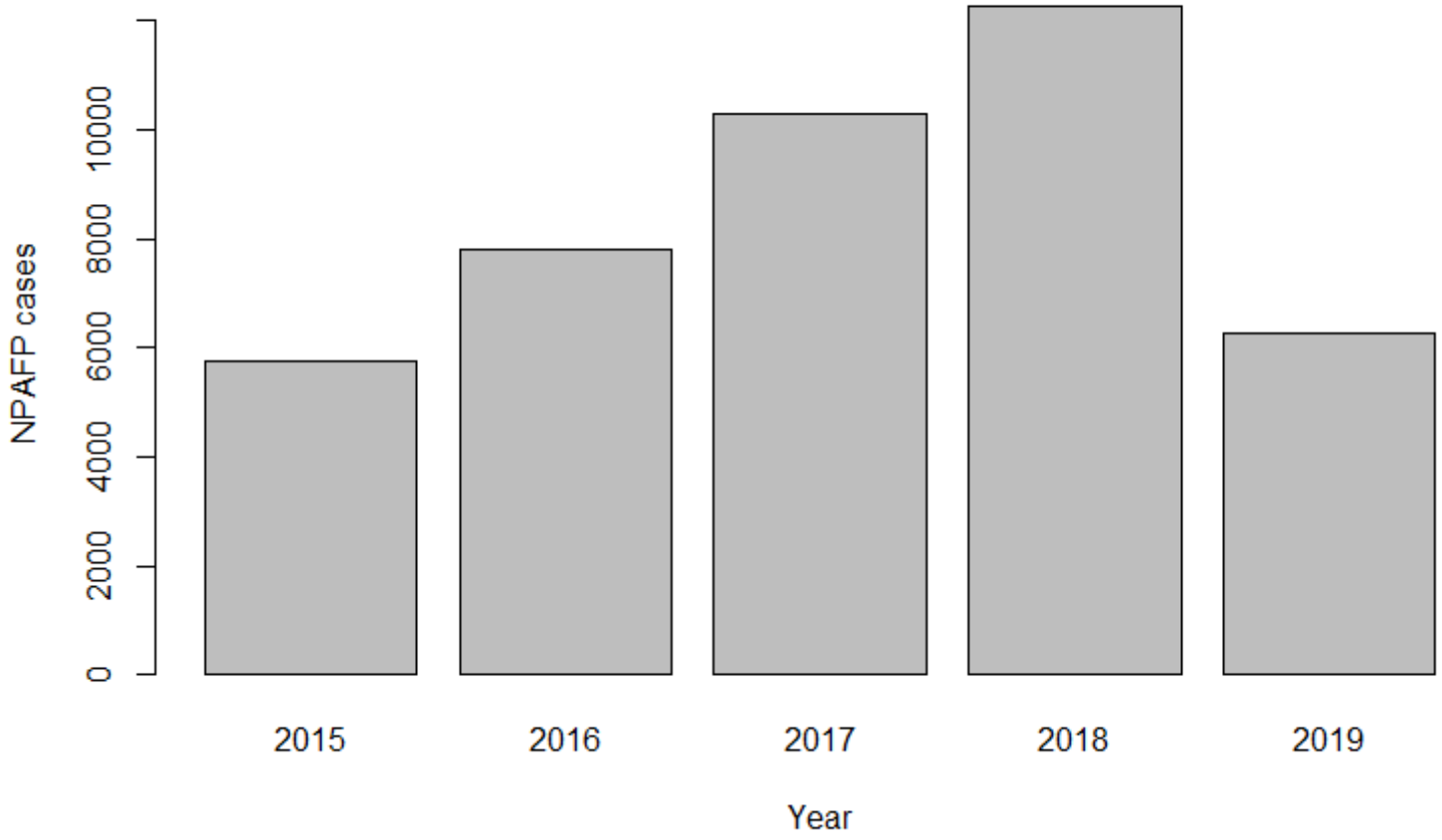


This is a simple barplot created using the `plot()` function. To make this look nicer we can change `plot()` for `barplot()`



Add x and y axis labels to your plot using `xlab=` and `ylab=` as we saw earlier.

```
barplot(NPAFP_YR, xlab="Year", ylab="NPAFP cases")
```



Subsetting data to plot

Making subsets of your data in R is straightforward, remember referencing inside data frames:

```
data[row,column]
```

And using == to determine if something is equal

Subsetting data to plot

Looking at your data file with View() you will see that there is a PROVINCE column in this data. To return all of the unique values in this column we can use the function `unique()`:

```
unique(NPAFP$PROVINCE)
```

This returns all of the province names included in this dataset.

This is useful as the dataset is long and scrolling through to see which provinces are included would be laborious.

Subsetting data to plot

If we want to select data from only one of these provinces, for example AJK, we can use the name of the province and run the following command. We will save this to a new object NPAFP_AJK:

Still need this comma, leaving the space after the comma blank will keep all columns

```
NPAFP_AJK<-NPAFP[NPAFP$PROVINCE==“AJK”,]
```

All rows where the column province is equal to AJK

Try extracting a different province from the dataset, use one of the names given by `unique(NPAFP$PROVINCE)`

Plotting a barchart for AJK

Use the `table()` function to create year totals for AJK in the same way as for the full dataset.

Now make the barplot for AJK by modifying your code from plotting the full dataset.