A Comprehensive Guide to Data Visualisation in R for Beginners

towardsdatascience.com/a-guide-to-data-visualisation-in-r-for-beginners-ef6d41a34174

Parul Pandey

February 4, 2019

An overview of the R visualisation capabilities.

The world today is filled with data and it becomes imperative that we analyse it properly to gain meaningful insights. Data Visualisation is a vital tool that can unearth possible crucial insights from data. If the results of an analysis are not visualised properly, it will not be communicated effectively to the desired audience.

In this tutorial, we will learn how to analyze and display data using R statistical language. We will begin with basic plots and move on to more advanced ones later in the article.

Table of Contents

Pre-requisites

A basic familiarity with R and its syntax will get you started easily.

Introduction to R

Overview

R is a language and environment for statistical computing and graphics. R is also extremely flexible and easy to use when it comes to creating visualisations. One of its capabilities is to produce good quality plots with minimum codes.

Installation

We shall briefly go over the steps required to install R :

1. Go to the R homepage and select **CRAN**.
   CRAN is an acronym for — Comprehensive R Archive Network. It's the collection of sites which carry R Distributions, Packages and documentation.
2. Select the location which is nearest to you.
3. Download and Install R depending upon your OS.

Alternatively, you can use RStudio over the base R GUI.

Startup

After R has been downloaded and installed, you can launch it from your Applications folder(MacOS) or Desktop Icon(Windows). Just type in the following commands to check if R has been installed properly and running.

```
> 1
1
> ‘hello World’
hello world
```

Loading in the Data

Datasets can either be built-in or can be loaded from external sources in R.

> **Built-in datasets** refer to the datasets already provided within R. We shall be using one such dataset called the **air quality** dataset, which pertains to the daily air quality measurements in New York from May to September 1973. This dataset consists of more than 100 observations on 6 variables i.e. **Ozone**(mean parts per billion), **Solar.R**(Solar Radiation), **Wind**(Average wind speed), **Temp**(maximum daily temperature in Fahrenheit), **Month**(month of observation) and **Day**(Day of the month)

To load the built-in dataset into the R type the following command in the console:

```
data(airquality)
```

> In case of an **External data** source (CSV, Excel, text, HTML file etc.), simply set the folder containing the data as the **working directory** with the **setwd()** command.

```
setwd(path of the folder where the file is located)
```

Now, load the file with the help of the **read** command. In this case, data is in the form of a CSV file named **airquality.csv** which can be downloaded from here

```
airquality = read.csv('airquality.csv',header=TRUE, sep=",")
```

The above code reads the file **airquality.csv** into a data frame **airquality**. **Header=TRUE** specifies that the data includes a header and **sep=”,”** specifies that the values in data are separated by commas.

Data Exploration

Once the data has been loaded into the workspace, it is time to explore it to get an idea about its structure.

> **str(data)**
> It displays the internal structure of an R object and gives a quick overview of the rows and columns of the dataset.

```
'data.frame': 111 obs. of  6 variables:
$ Ozone  : int  41 36 12 18 23 19 8 16 11 14 ...
$ Solar.R: int  190 118 149 313 299 99 19 256 290 274 ...
$ Wind   : num  7.4 8 12.6 11.5 8.6 13.8 20.1 9.7 9.2 10.9 ...
$ Temp   : int  67 72 74 62 65 59 61 69 66 68 ...
$ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
$ Day    : int  1 2 3 4 7 8 9 12 13 14 ... `
```

head(data,n) and **tail(data,n)**
The head outputs the top **n** elements in the dataset while the tail method outputs the bottom **n**.

```
head(data, n=3)
Ozone Solar.R Wind Temp Month Day
1    41      190  7.4   67      5   1
2    36      118  8.0   72      5   2
3    12      149 12.6   74      5   3

tail(data, n=3)
   Ozone Solar.R Wind Temp Month Day
109    14      191 14.3   75      9  28
110    18      131  8.0   76      9  29
111    20      223 11.5   68      9  30
```

summary(data)
The summary method displays descriptive statistics for every variable in the dataset, depending upon the type of the variable.

```
summary(data)

Ozone          Solar.R          Wind            Temp           Month            Day
 Min.   :  1.0   Min.   :  7.0   Min.   : 2.30   Min.   :57.00   Min.   :5.000   Min.
: 1.00
 1st Qu.: 18.0   1st Qu.:113.5   1st Qu.: 7.40   1st Qu.:71.00   1st Qu.:6.000   1st
Qu.: 9.00
 Median : 31.0   Median :207.0   Median : 9.70   Median :79.00   Median :7.000
Median :16.00
 Mean   : 42.1   Mean   :184.8   Mean   : 9.94   Mean   :77.79   Mean   :7.216   Mean
:15.95
 3rd Qu.: 62.0   3rd Qu.:255.5   3rd Qu.:11.50   3rd Qu.:84.50   3rd Qu.:9.000   3rd
Qu.:22.50
 Max.   :168.0   Max.   :334.0   Max.   :20.70   Max.   :97.00   Max.   :9.000   Max.
:31.00
```

We can see at a glance the **mean**, **median**, **max** and the **quartile values** of the variables.

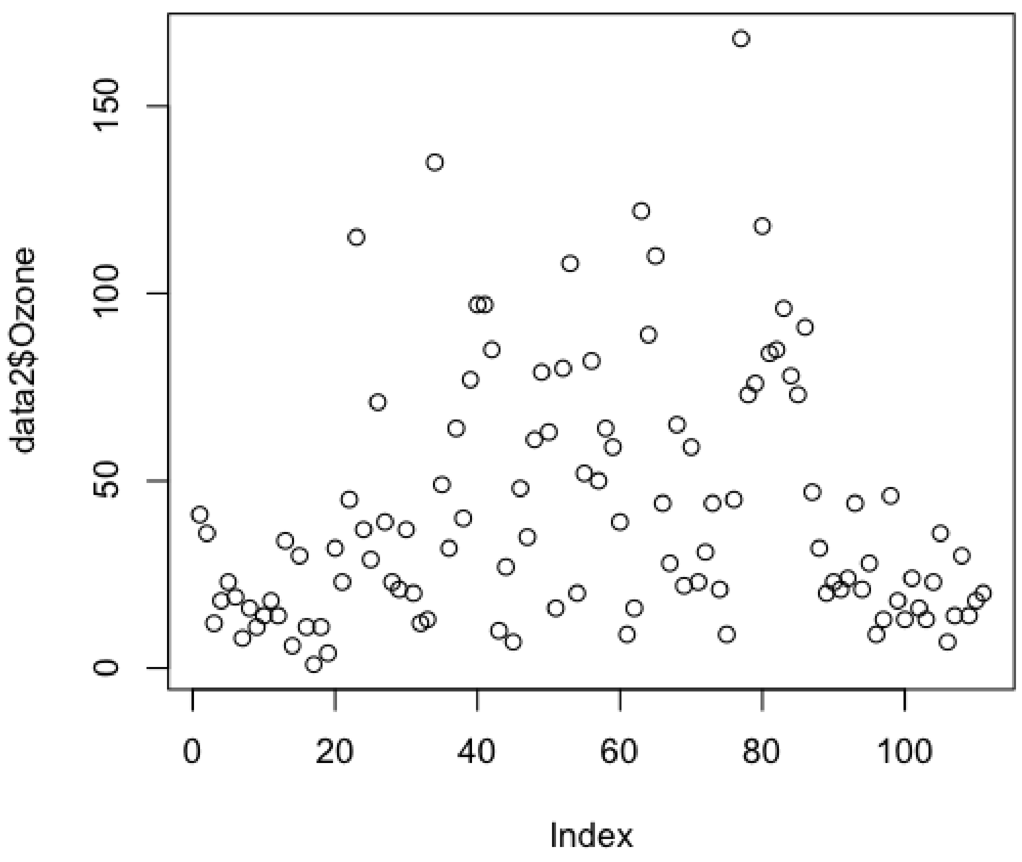
Getting Started with Basic Plots

The **graphics** package is used for plotting **base** graphs like scatter plot, box plot etc. A complete list of functions with help pages can be obtained by typing : `library(help = "graphics")`.

The plot() function

The `plot()` function is a kind of a generic function for plotting of **R** objects.

```
plot(airquality$Ozone)
```

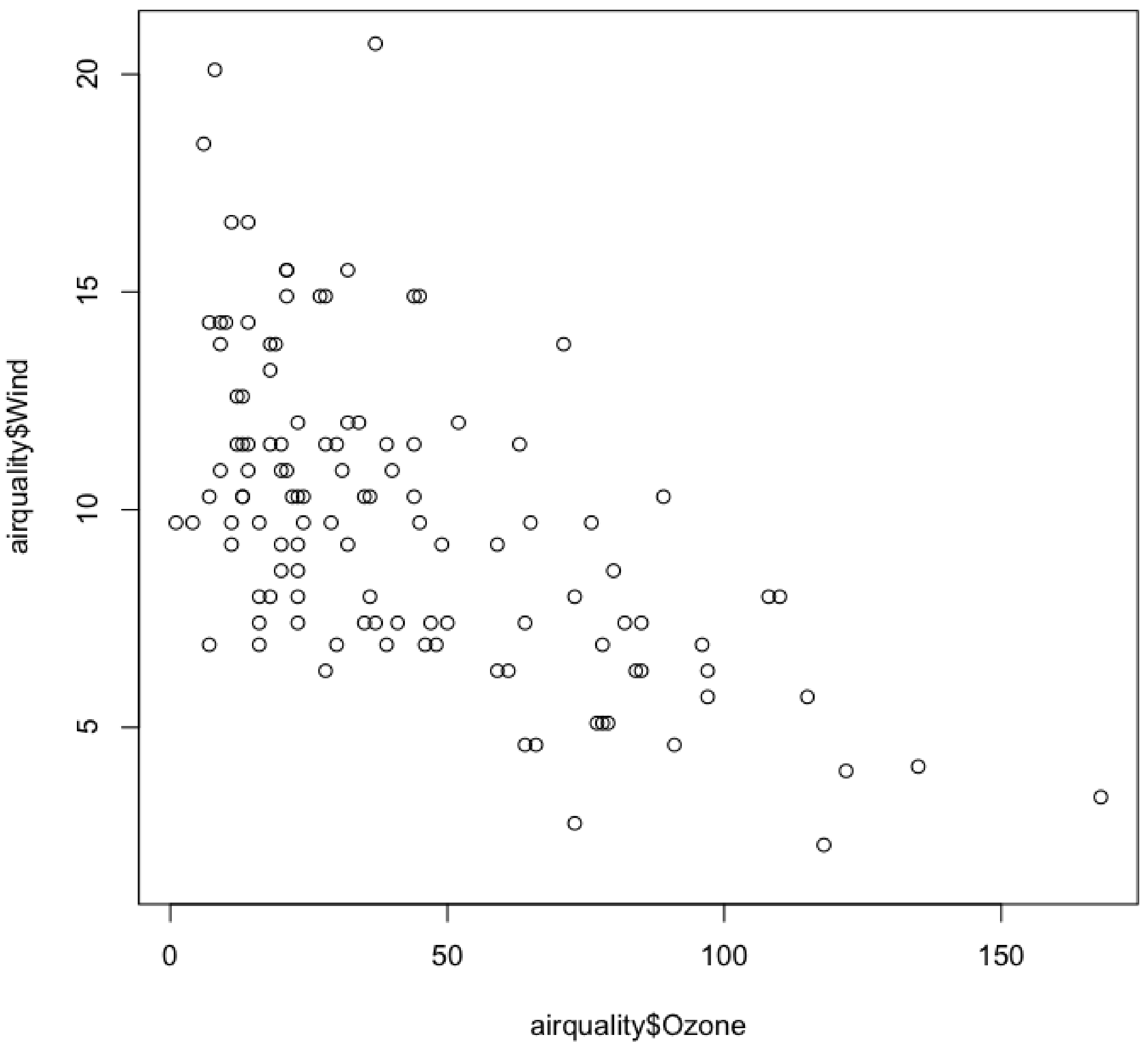


Scatter Plot

We get a **scatter/dot** plot here wherein each dot represents the value of the **Ozone** in **mean parts per billion**.

Let us now plot a graph between the Ozone and Wind values to study the relationship between the two.

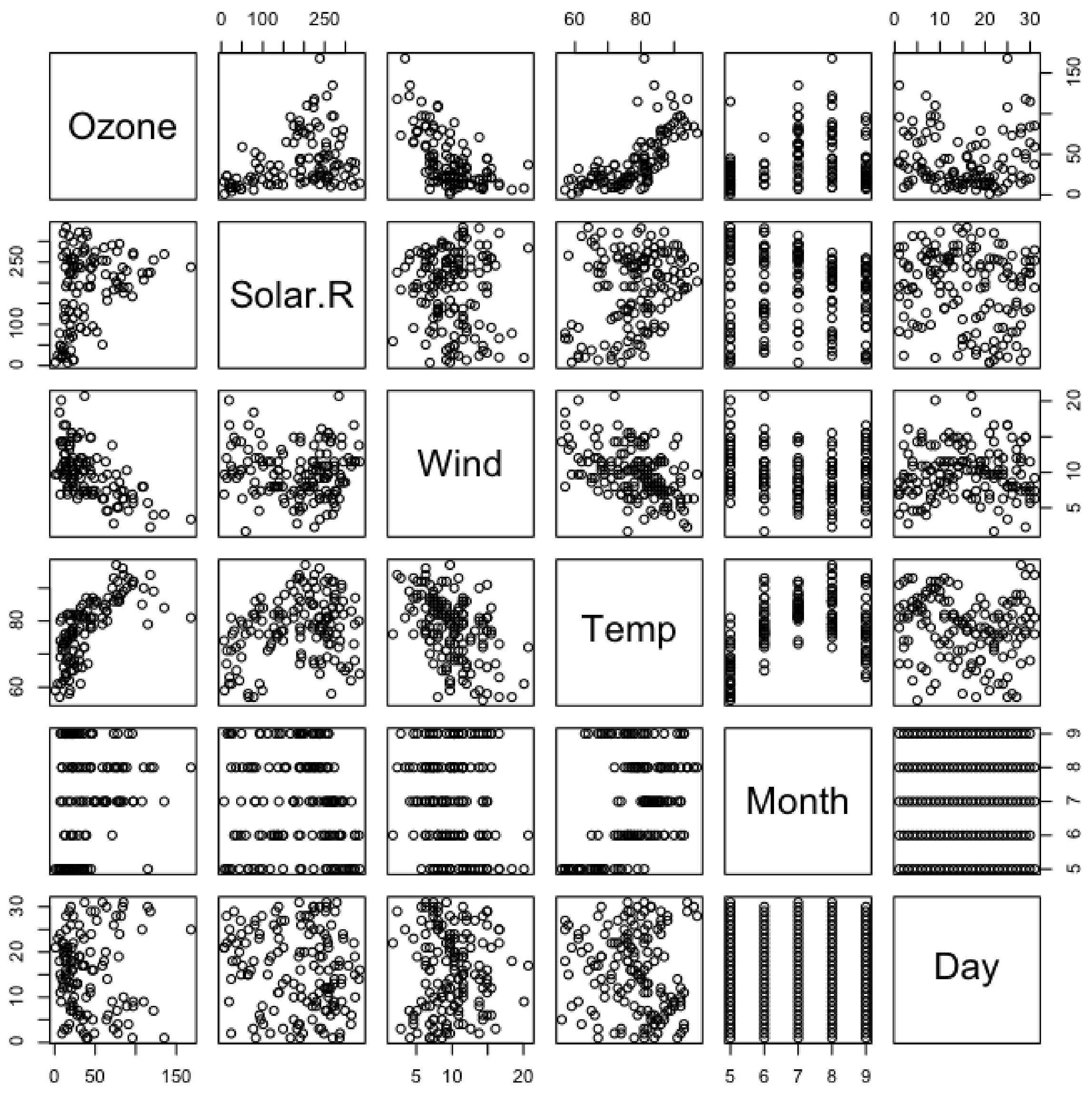
```
plot(airquality$Ozone, airquality$Wind)
```



The plot shows that Wind and Ozone values have a somewhat **negative correlation**.

> What happens when we use plot command the with the entire dataset without selecting any particular columns?

```
plot(airquality)
```



We get a matrix of scatterplots which is a correlation matrix of all the columns. The plot above instantly shows that:

- The level of Ozone and Temperature is correlated positively.
- Wind speed is negatively correlated to both Temperature and Ozone level.

We can quickly discover the relationship between variables by merely looking at the plots drawn between them.

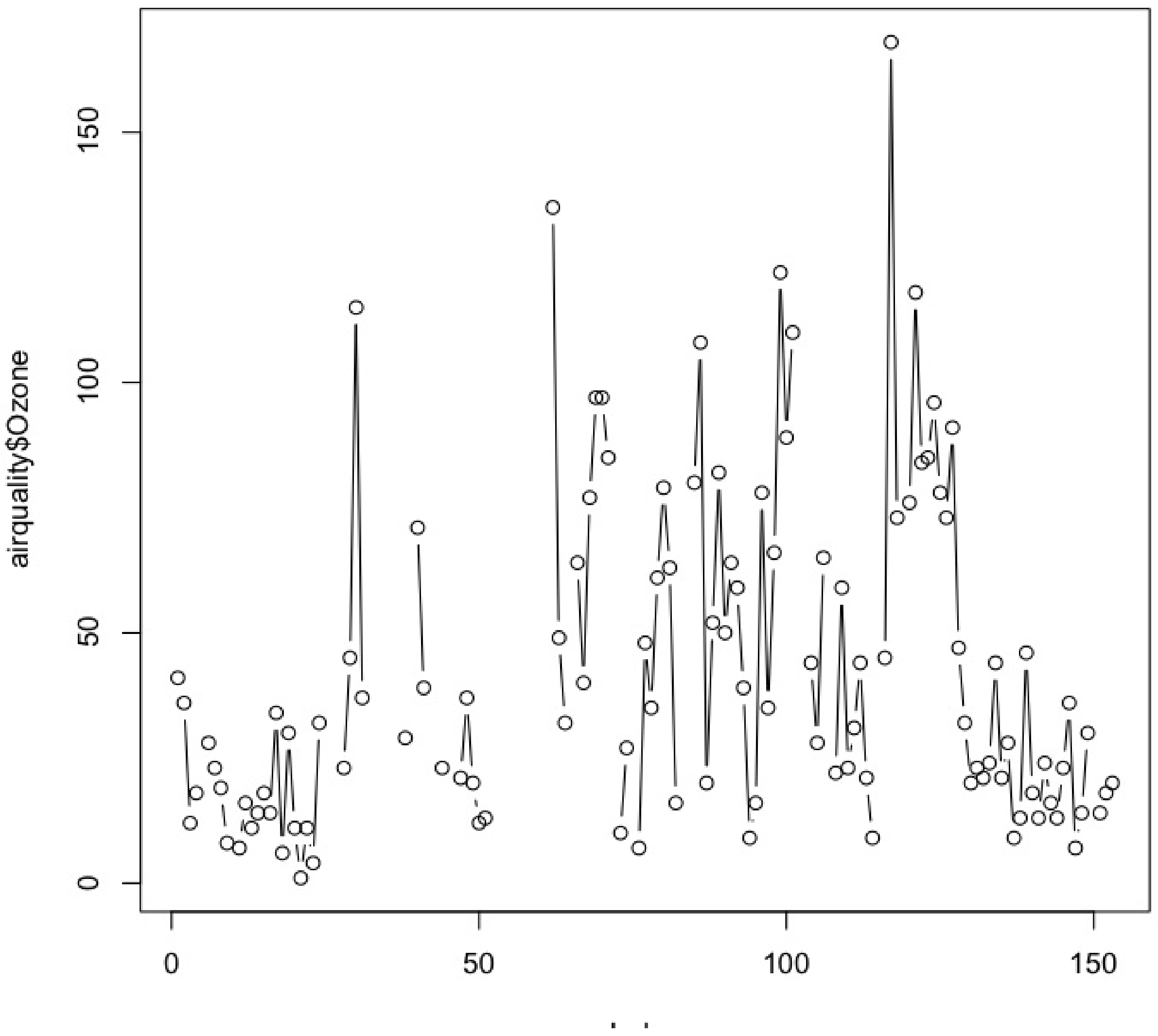
Using arguments with the plot() function

We can easily style our charts by playing with the arguments of the `plot()` function.

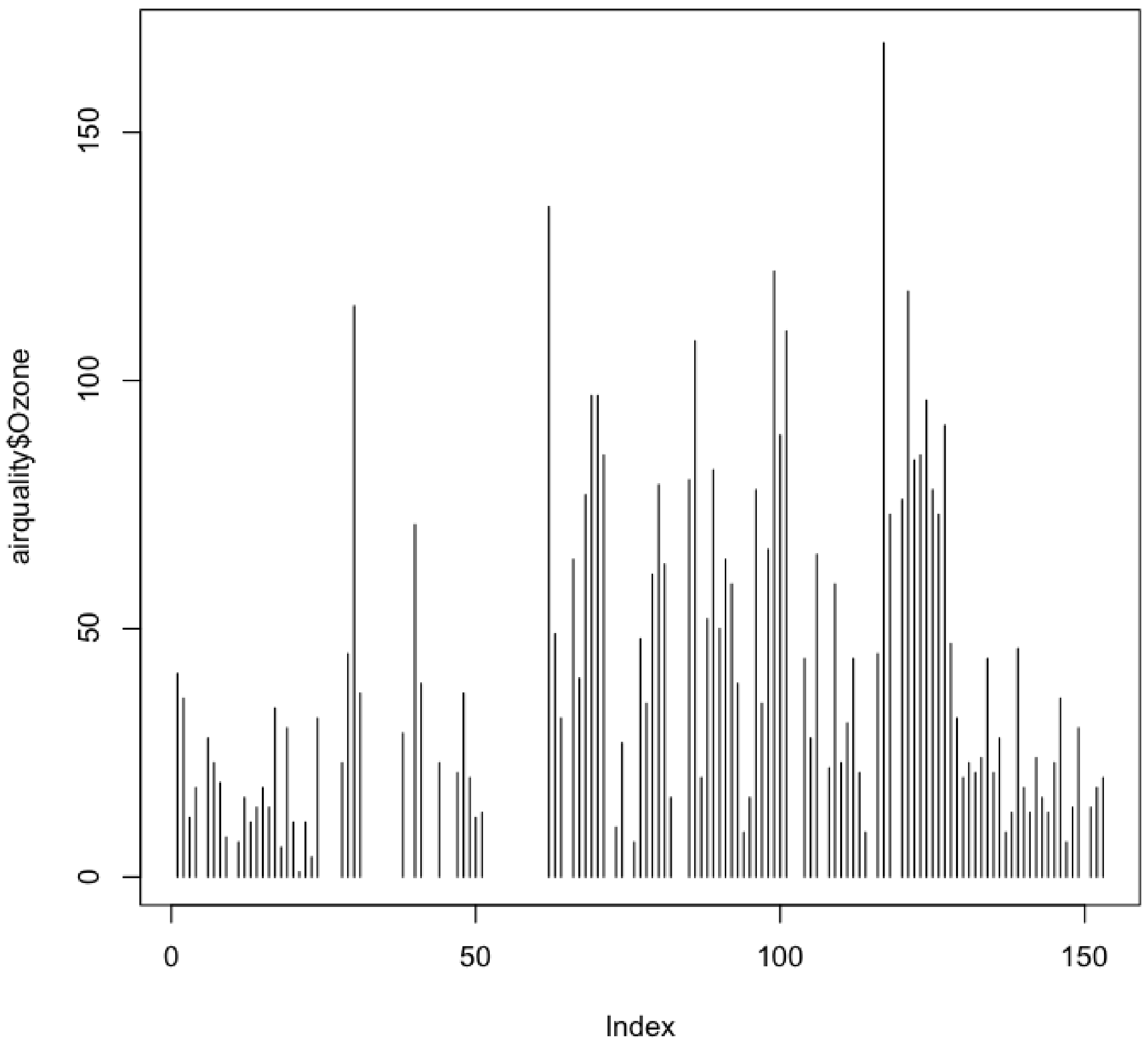
type argument

The plot function has an argument called `type` which can take in values like **p: points**, **l: lines**,**b: both** etc. This decides the shape of the output graph.

```
# points and lines
 plot(airquality$Ozone, type= "b")
```



```
# high density vertical lines.
 plot(airquality$Ozone, type= "h")
```

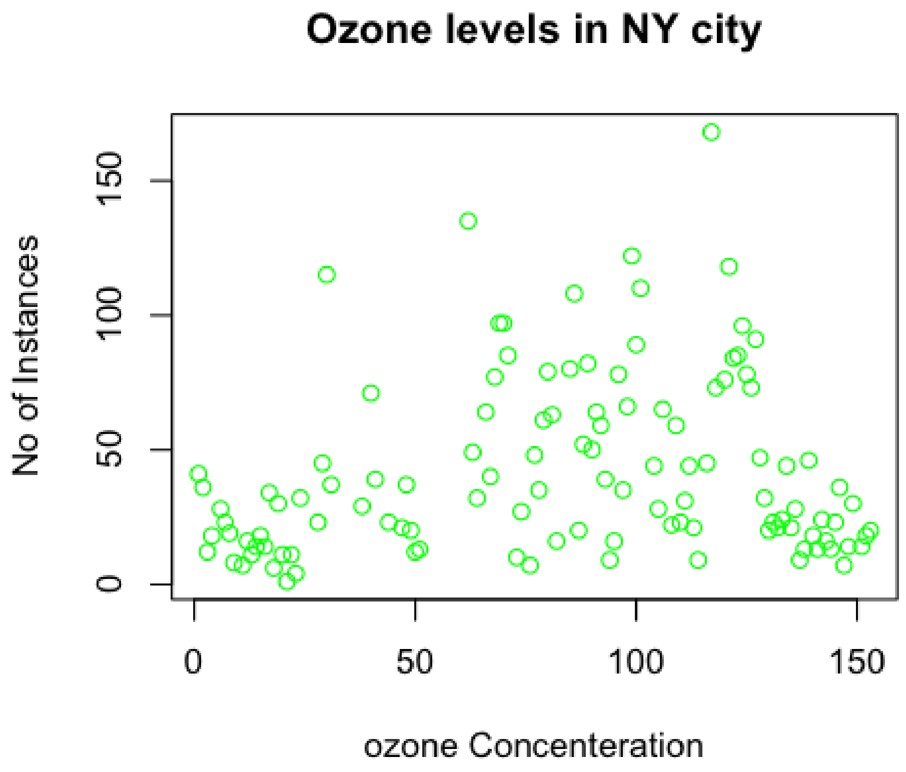


You can read more about the `plot()` command by typing `?plot()` in the console.

Labels and Titles

We can also label the X and the Y axis and give a title to our plot. Additionally, we also have the option of giving color to the plot.

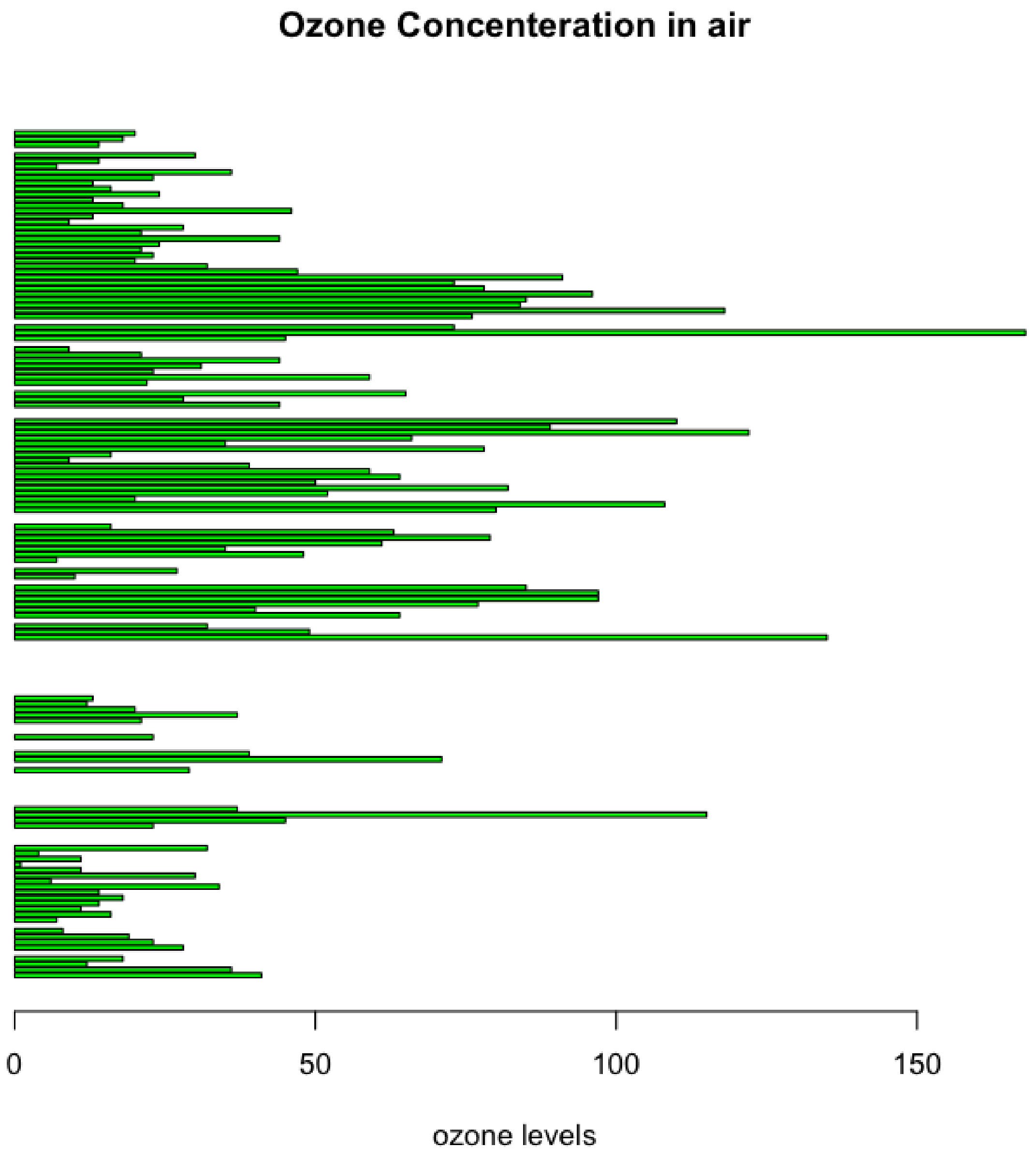
```
plot(airquality$Ozone, xlab = ‘ozone Concentration’, ylab = ‘No of Instances’, main
= ‘Ozone levels in NY city’, col = ‘green’)
```



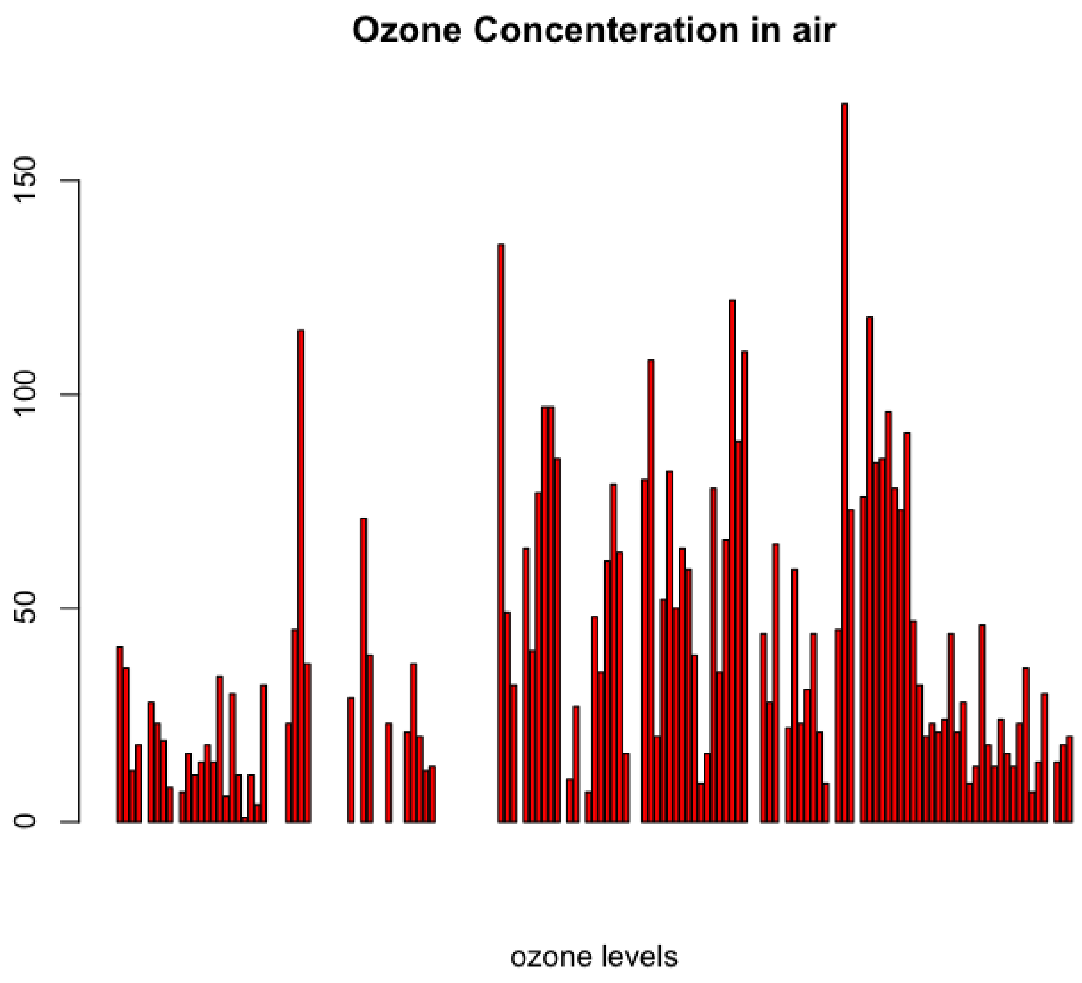
2.Barplot

In a bar plot, data is represented in the form of rectangular bars and the length of the bar is proportional to the value of the variable or column in the dataset. Both horizontal, as well as a vertical bar chart, can be generated by tweaking the **horiz** parameter.

```
# Horizontal bar plot
barplot(airquality$Ozone, main = 'Ozone Concenteration in air',xlab = 'ozozne
levels', col='green',horiz = TRUE)
```



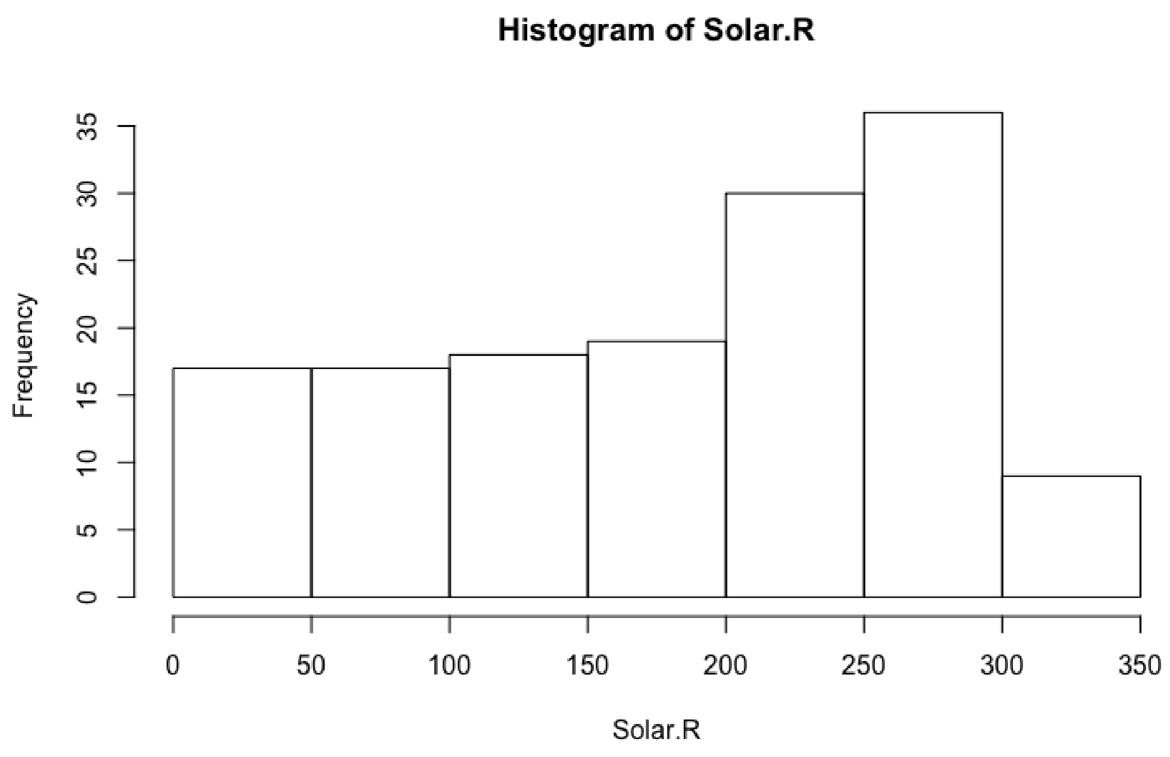
```
# Vertical bar plot
barplot(airquality$Ozone, main = 'Ozone Concenteration in air',xlab = 'ozone
levels', col='red',horiz = FALSE)
```



3.Histogram

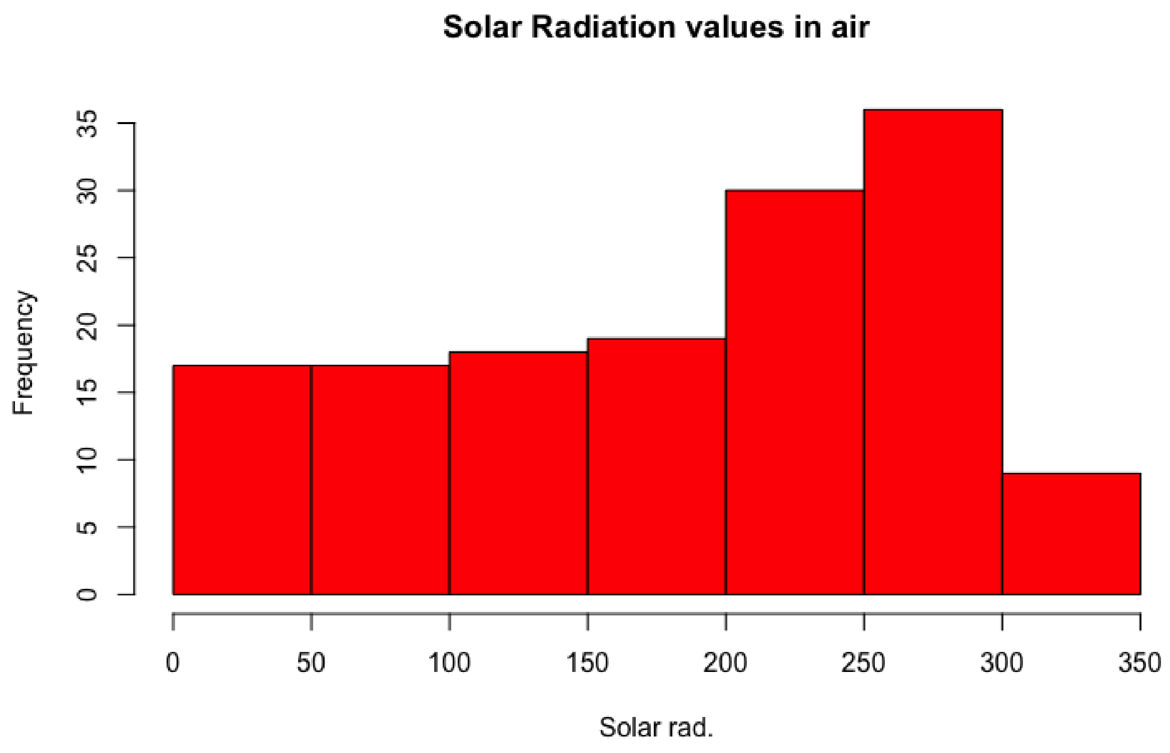
A histogram is quite similar to a bar chart except that it groups values into continuous ranges. A histogram represents the frequencies of values of a variable bucketed into ranges.

```
hist(airquality$Solar.R)
```



We get a histogram of the **Solar.R** values. By giving an appropriate value for the **color** argument, we can obtain a coloured histogram as well.

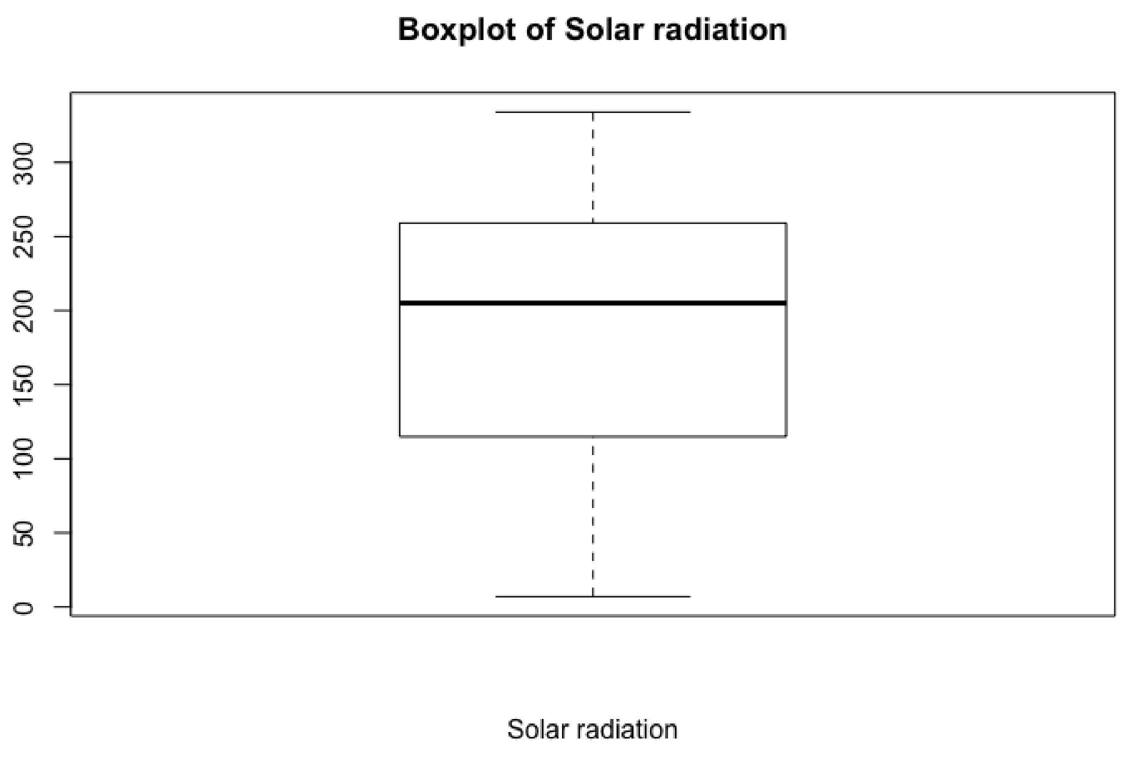
```
hist(Solar.R, main = 'Solar Radiation values in air',xlab = 'Solar rad.', col='red')
```



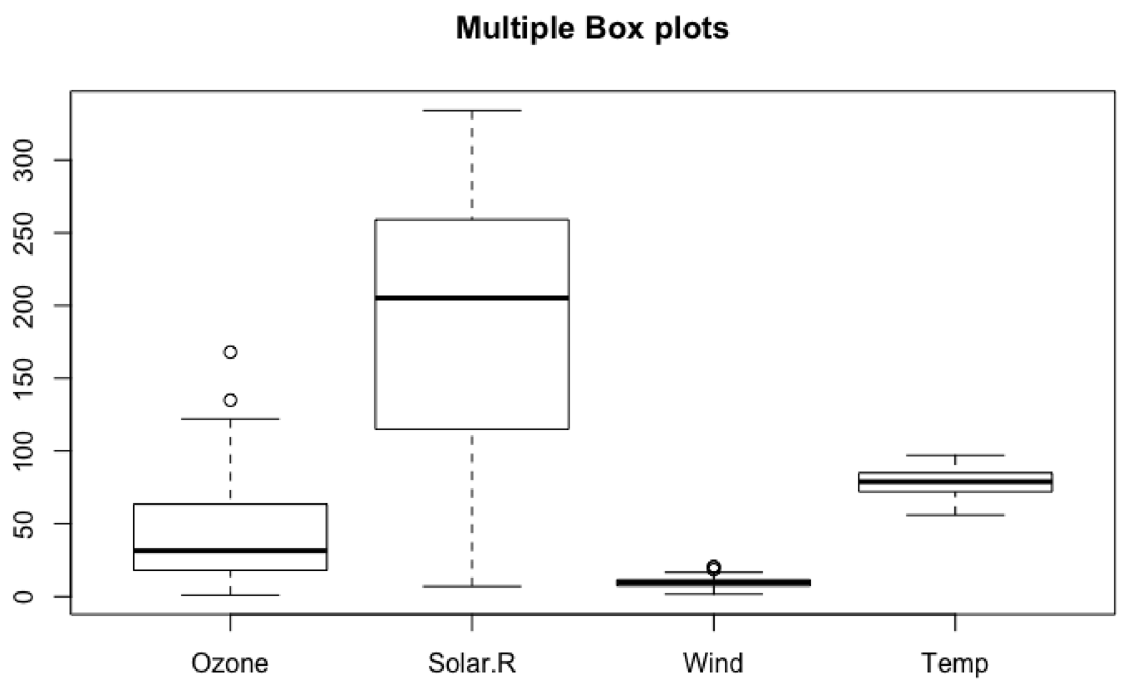
4.Boxplot

We have seen how the `summary()` command in R can display the descriptive statistics for every variable in the dataset. Boxplot does the same albeit graphically in the form of quartiles. It is again very straightforward to plot a boxplot in R.

```
#Single box plot
boxplot(airquality$Solar.R)
```



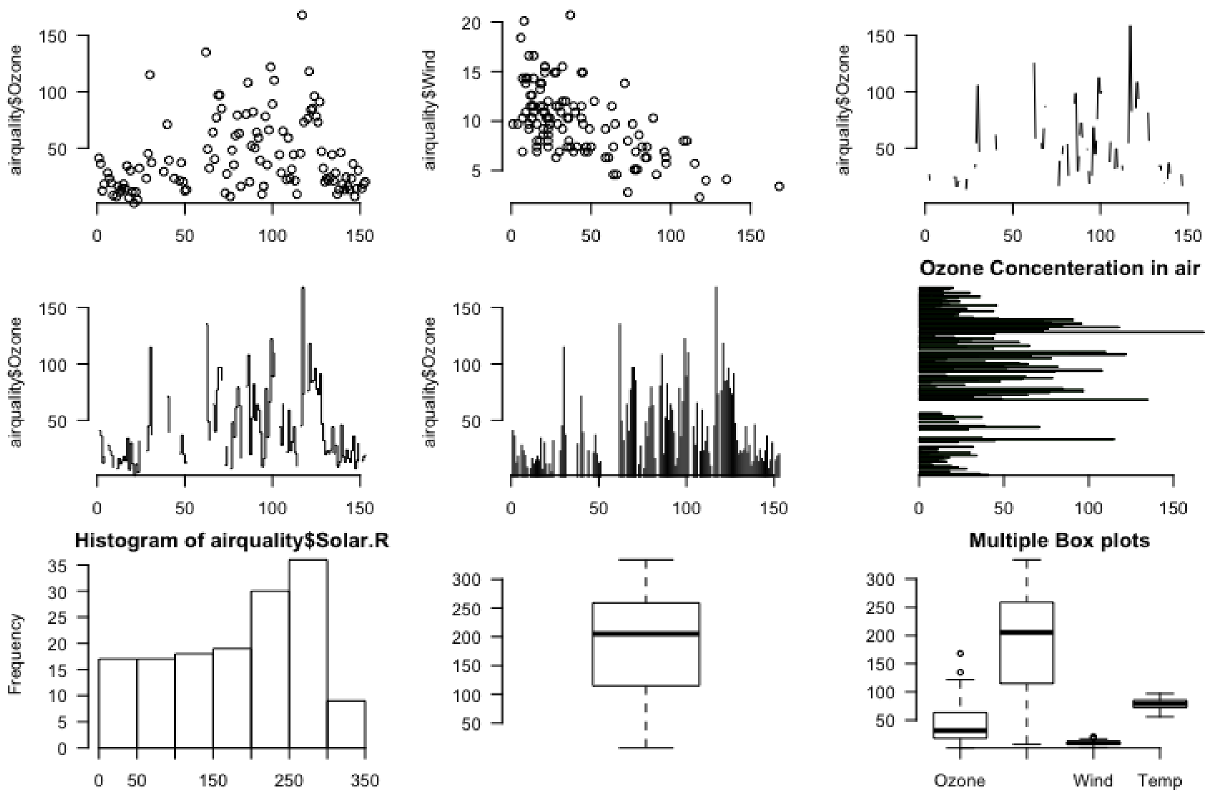
```
# Multiple box plots
boxplot(airquality[,0:4], main='Multiple Box plots')
```



5. Grid of Charts

There is a very interesting feature in R which enables us to plot multiple charts at once. This comes in very handy during the EDA since the need to plot multiple graphs one by one is eliminated.
For drawing a grid, the first argument should specify certain attributes like the margin of the grid(`mar)` , no of rows and columns(`mfrow`), whether a border is to be included(`bty`) and position of the labels(`las` : 1 for horizontal, `las` : 0 for vertical).

```
par(mfrow=c(3,3), mar=c(2,5,2,1), las=1, bty="n")
plot(airquality$Ozone)
plot(airquality$Ozone, airquality$Wind)
plot(airquality$Ozone, type= "c")
plot(airquality$Ozone, type= "s")
plot(airquality$Ozone, type= "h")
barplot(airquality$Ozone, main = 'Ozone Concenteration in air',xlab = 'ozozne
levels', col='green',horiz = TRUE)
hist(airquality$Solar.R)
boxplot(airquality$Solar.R)
boxplot(airquality[,0:4], main='Multiple Box plots'
```



> Note: You can use function documentation to know more about a given function by typing `? plot name` . Also, `example(plot)` runs the demo of the plot directly in the console.

Visualisation libraries in R

R comes equipped with sophisticated visualisation libraries having great capabilities. Let us have a closer look at some of the commonly used ones.

In this section, we will use the built-in mtcars dataset to show the uses of the various libraries. This dataset has been extracted from the 1974 *Motor Trend* US magazine.

Lattice Graphs

Lattice package is essentially an improvement upon the R Graphics package and is used to visualize multivariate data. Lattice enables the use of ***trellis graphs***. Trellis graphs exhibit the relationship between variables which are dependent on one or more variables. Let us start by installing and loading the package.

```
# Installing & Loading the package
install.package("lattice")
library(lattice)

#Loading the dataset
attach(mtcars)
```

The **attach** function attaches the database to the **R** search path so the objects in the database can be accessed by simply giving their names. (See `?attach()` for more details)

```
# Exploring the dataset

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
```

Before proceeding with the working of the lattice package, let us do a little pre-processing of the data. There are two columns in our mtcars dataset namely **gear** and **cyl** which are categorical in nature. We need to factorize them to make them more meaningful.

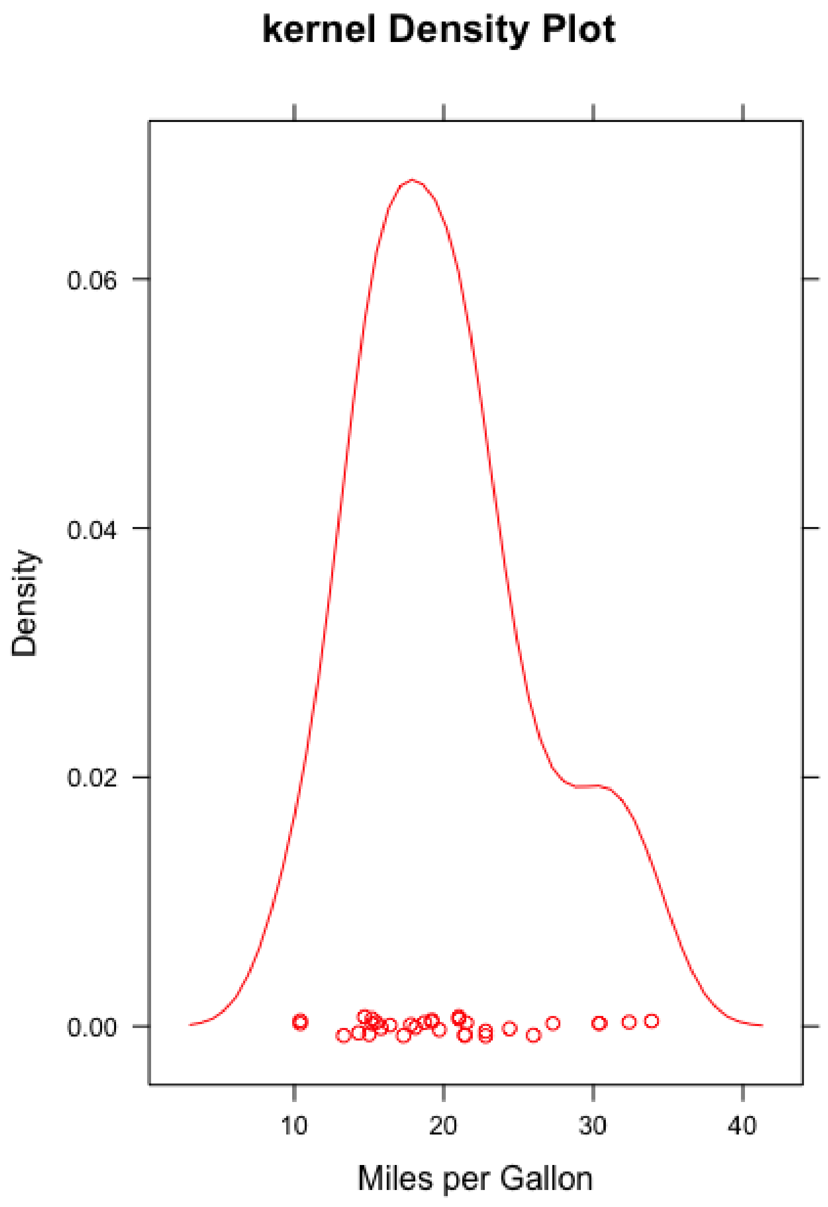
```
gear_factor<-factor(gear,levels=c(3,4,5),
labels=c("3gears","4gears","5gears"))

cyl_factor <-factor(cyl,levels=c(4,6,8),
labels=c("4cyl","6cyl","8cyl"))
```

Note: since we have attached the dataset mtcars, we do not need to specify `mtcars$gear` *or* `mtcars$cyl` *.*

Now let us see how we can use the **lattice** package to create some basic plots in R.

Kernel density plots

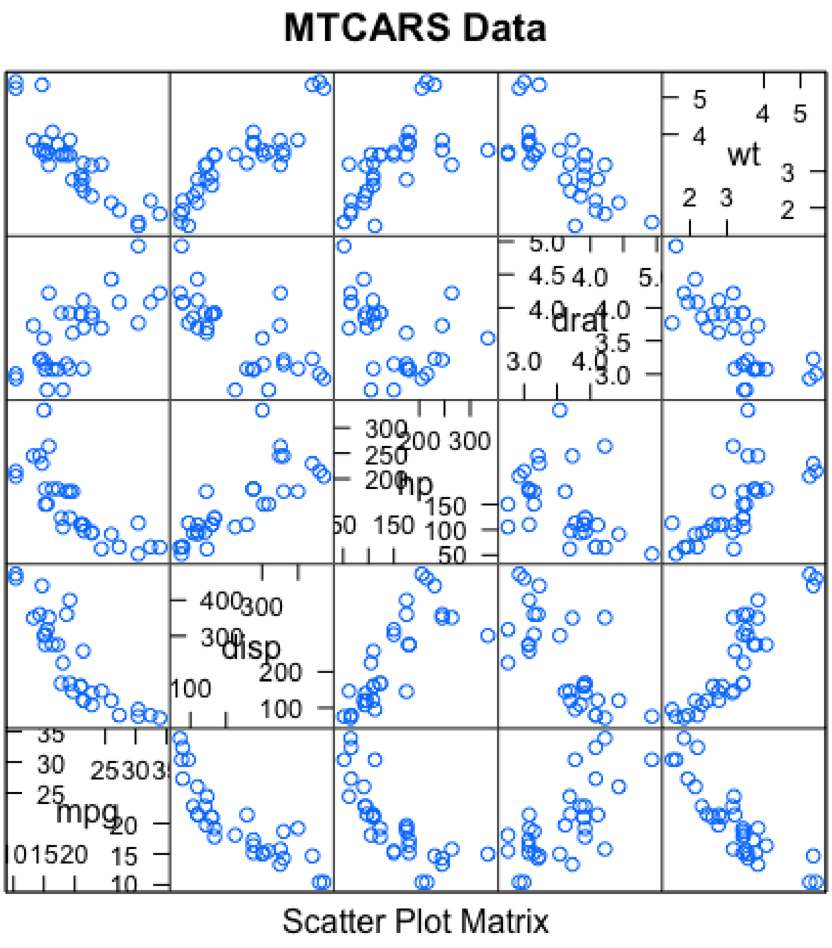
```
densityplot(~mpg, main="Density Plot",  xlab="Miles per Gallon")
```



It is very straightforward to use the lattice library. One simply needs to plug in the columns for which the plot is desired.

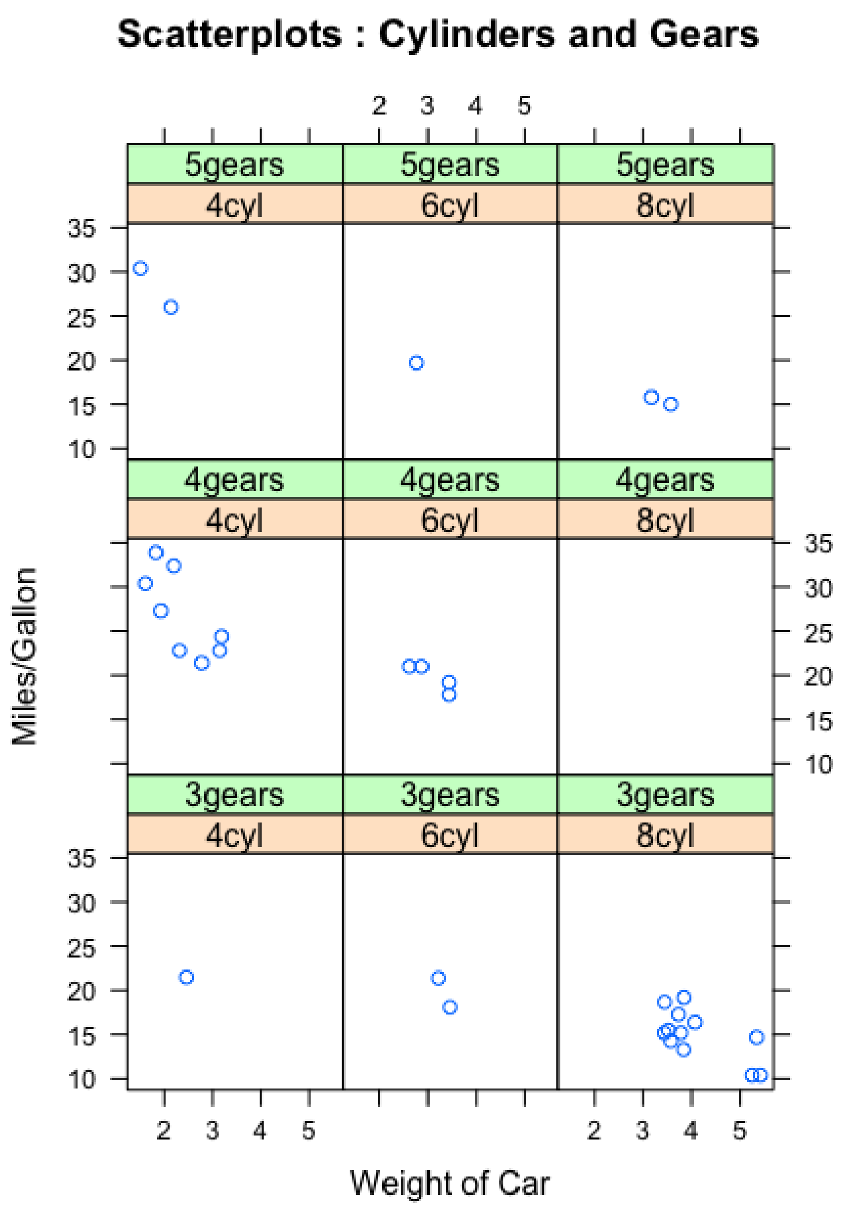
scatterplot matrix

```
splom(mtcars[c(1,3,4,5,6)], main="MTCARS Data")
```



Scatterplots depicting a combination of two factors

```
xyplot(mpg~wt|cyl_factor*gear_factor,
main="Scatterplots : Cylinders and Gears",
ylab="Miles/Gallon", xlab="Weight of Car")
```



2. ggplot2

The ggplot2 package is one of the most widely used visualisation packages in R. It enables the users to create sophisticated visualisations with little code using the **Grammar of Graphics**. The Grammar of Graphics is a general scheme for data visualization which breaks up graphs into semantic components such as scales and layers.

The popularity of ggplot2 has increased tremendously in recent years since it makes it possible to create graphs that contain both univariate and multivariate data in a very simple manner.

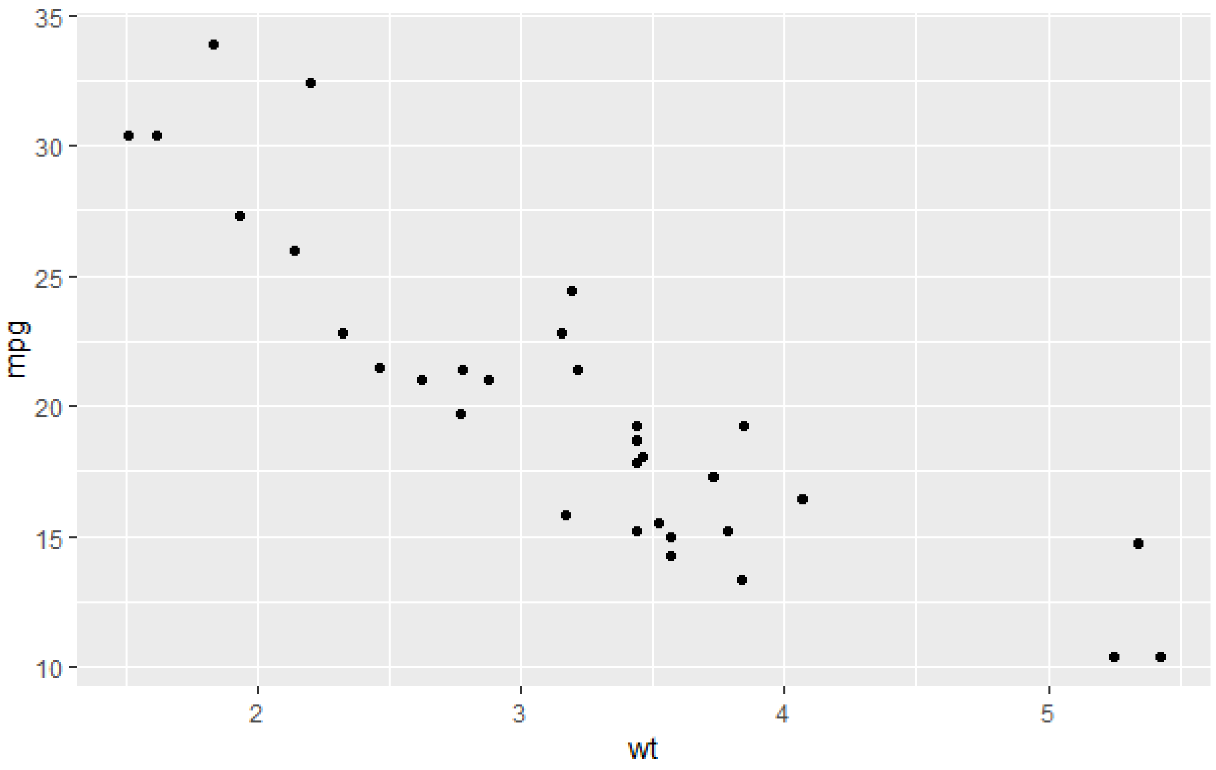
```
#Installing & Loading the package

install.package(“ggplot2”)
library(ggplot2)

#Loading the dataset
attach(mtcars)
```

```
# create factors with value labels

mtcars$gear <- factor(mtcars$gear,levels=c(3,4,5),
labels=c(“3gears”, ”4gears”, ”5gears”))
mtcars$am <- factor(mtcars$am,levels=c(0,1),
labels=c(“Automatic”,”Manual”))
mtcars$cyl <- factor(mtcars$cyl,levels=c(4,6,8),
labels=c(“4cyl”,”6cyl”,”8cyl”))
```

Let us create a few plots to understand the capability of ggplot2

Scatter Plots

```
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg)) + geom_point()
```

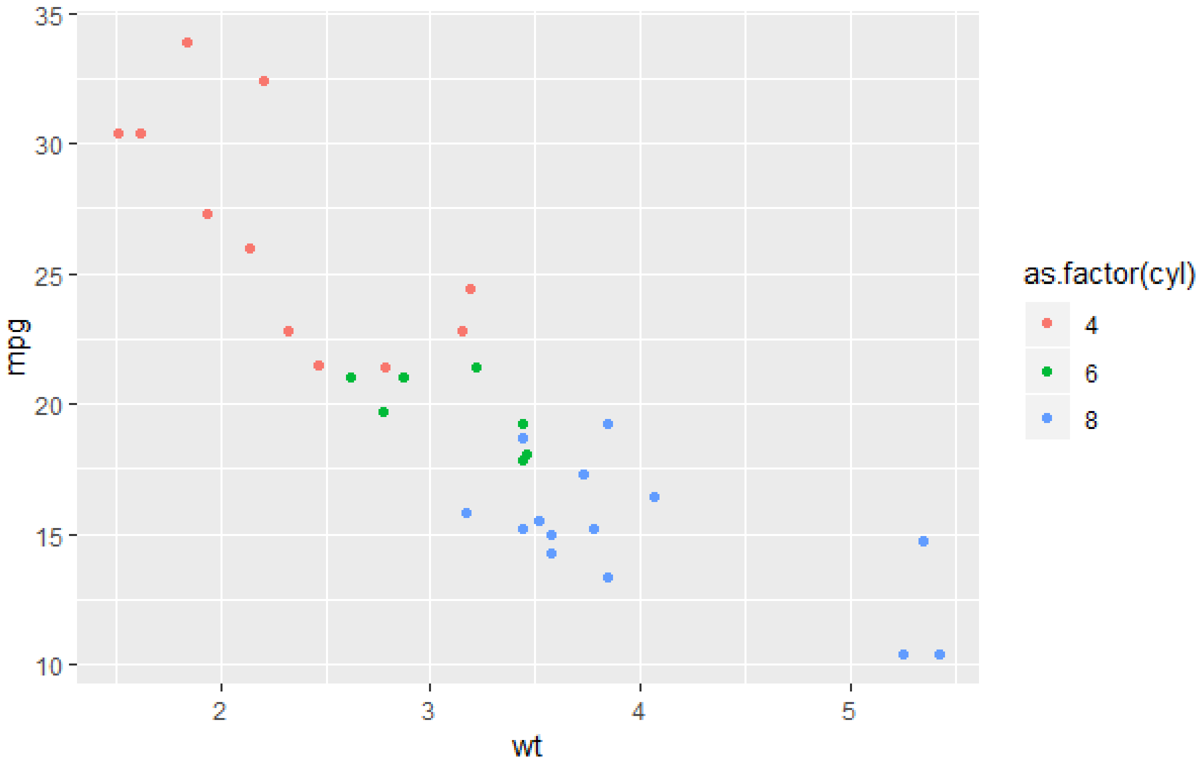


`geom_point()` is used to create scatterplots and geom can have many variations like `geom_jitter()` , `geom_count()` etc

Styling scatter plots by factor

We know that the dataset mtcars consists of certain variables which are in the form of factors. We can utilise this property to split our dataset

```
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, color = as.factor(cyl))) +
geom_point()
```

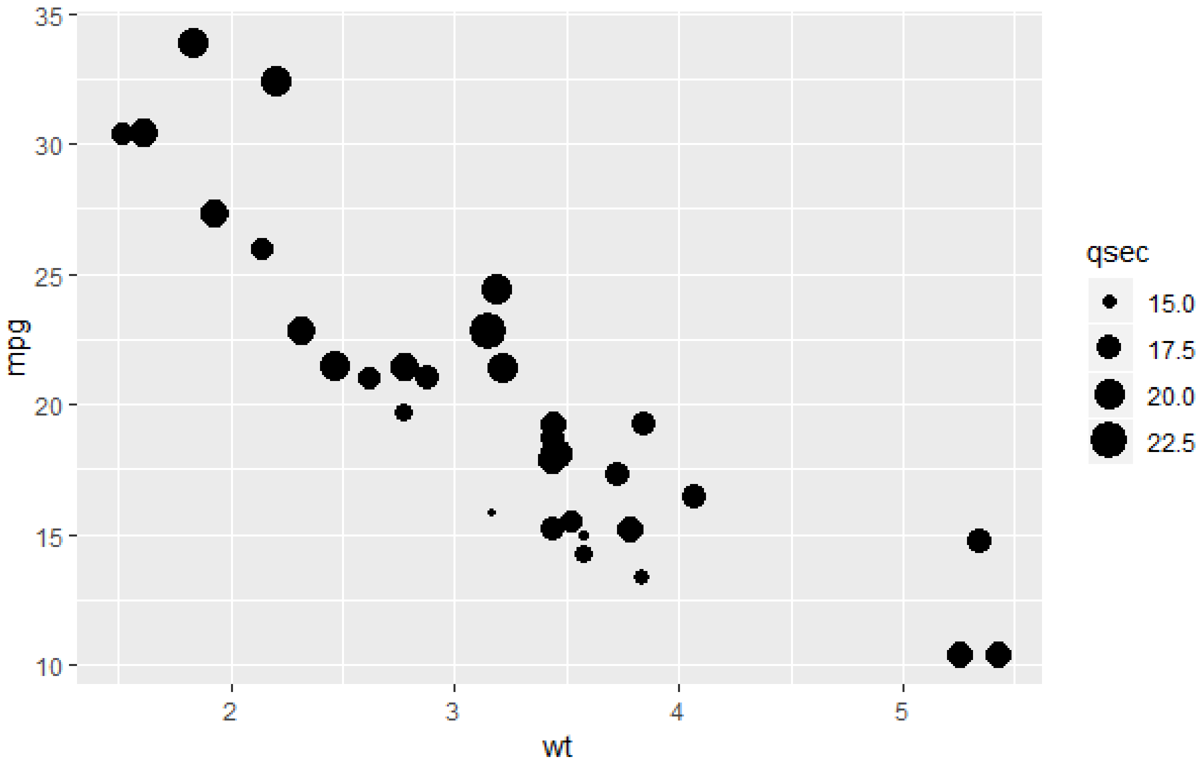


The color parameter is used to differentiate between different factor level of the cyl variable.

Styling scatter plots by size

Another useful feature of ggplot2 is that it can be styled according to the size of the attributes.

```
ggplot(data = mtcars, mapping = aes(x = wt, y = mpg, size = qsec)) + geom_point()
```

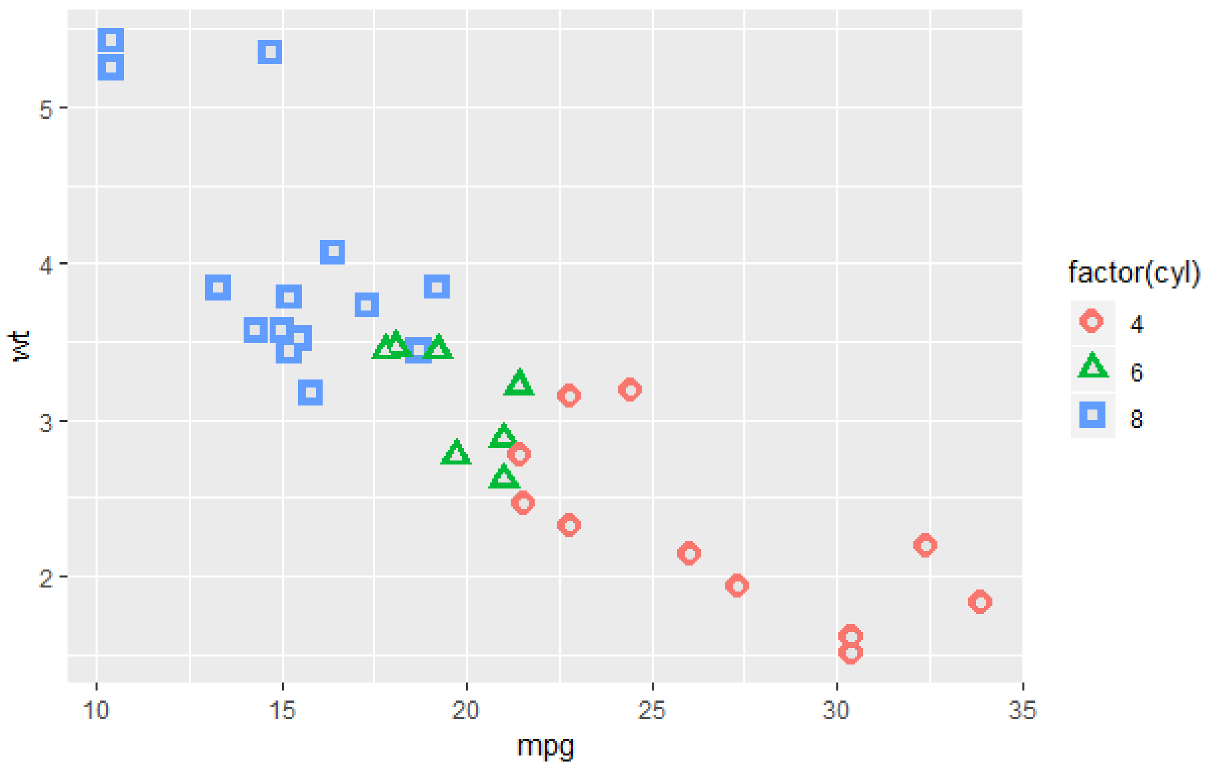


In the above example, the value of `qsec` indicates the acceleration which decides the size of the points.

Different symbols for different sizes

With ggplot2, one can also create unique and interesting shapes by layering multiple points of different sizes

```
p  <-  ggplot(mtcars,aes(mpg, wt, shape  =  factor(cyl)))
  p + geom_point(aes(colour  =  factor(cyl)), size  =  4) + geom_point(colour  =
"grey90", size  =  1.5)
```



3. Plotly

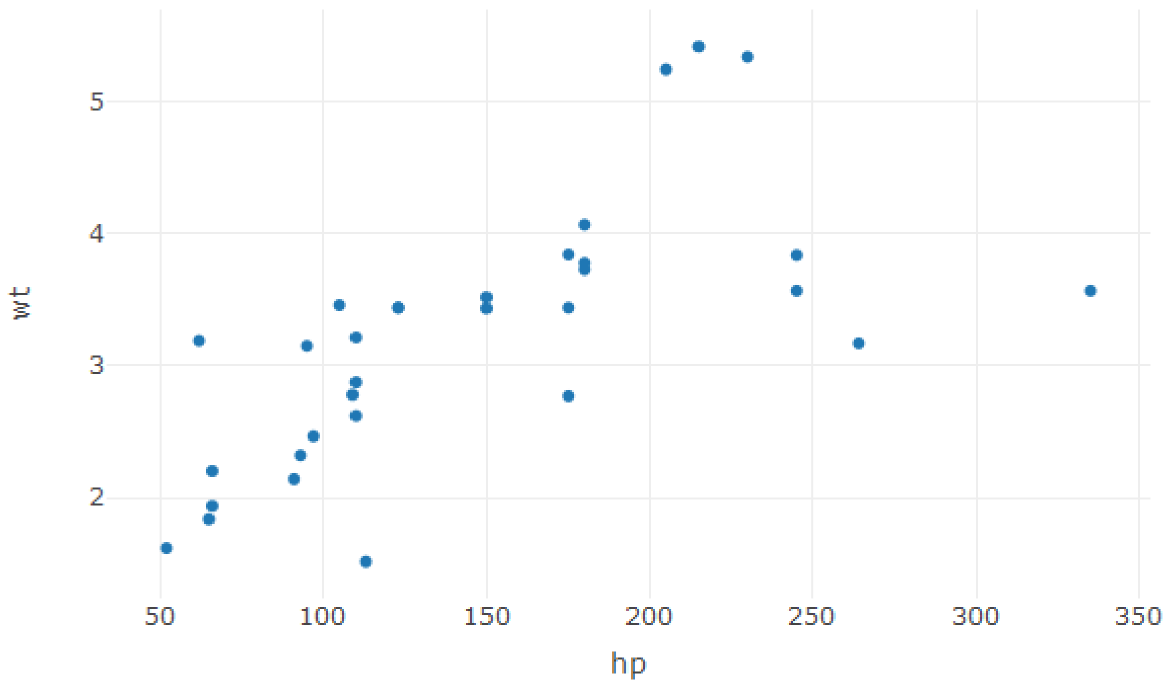
Plotly is an R package that creates interactive web-based graphs via the open source JavaScript graphing library plotly.js. It can easily translate the ‘ggplot2’ graphs to web-based versions also.

```
#Installing & Loading the package

install.package(“plotly”)
 library(plotly)
```

Let us now see how we can utilise plotly to create interactive visualisations. We will be working with the same mtcars dataset that used in the lattice graphs demonstration.

Basic Scatter Plot

```
p <- plot_ly(data = mtcars, x = ~hp, y = ~wt)
p
```

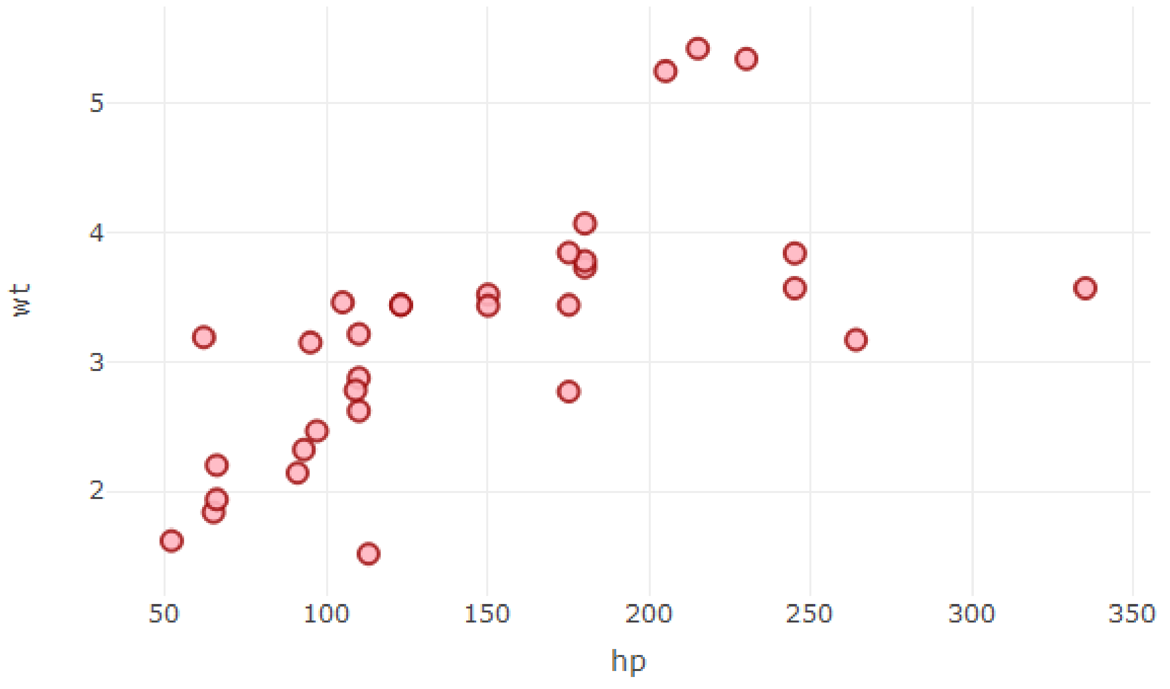


The plot above can also be exported in form of a web page to keep its interactiveness intact.

Styled Scatter Plot

The scatter plot can be styled by giving in the appropriate color codes.

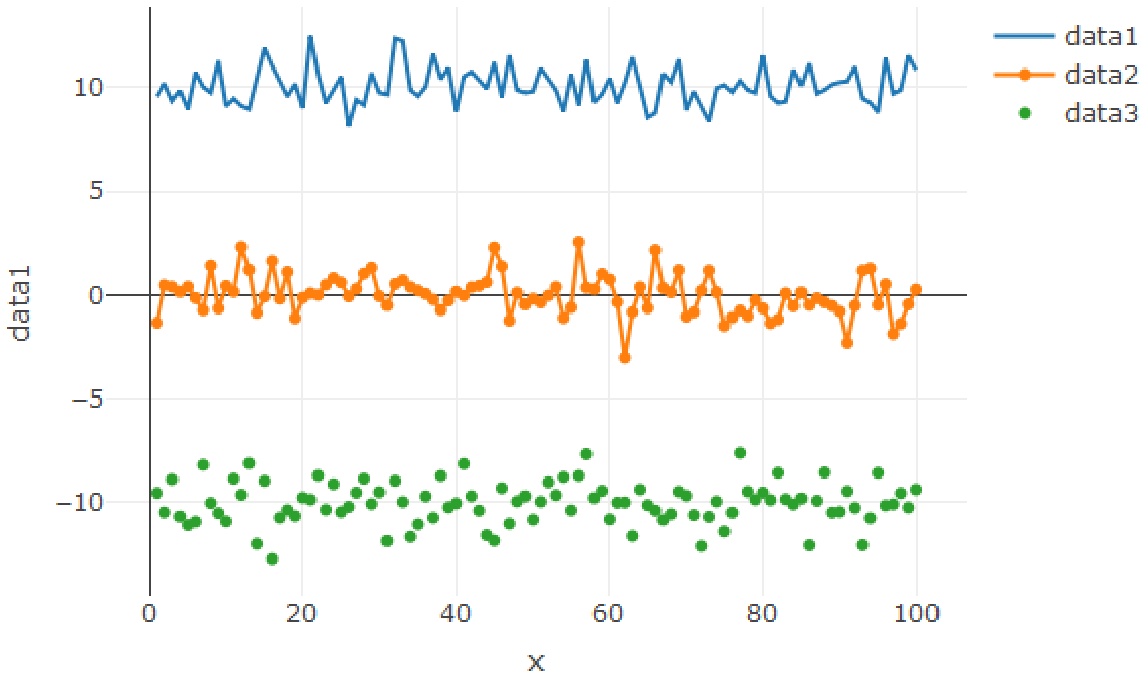
```
p <- plot_ly(data = mtcars, x = ~hp, y = ~wt, marker = list(size = 10, color =
'rgba(255, 182, 193, .9)', line = list(color = 'rgba(152, 0, 0, .8)', width = 2)))

p
```



Markers and Lines

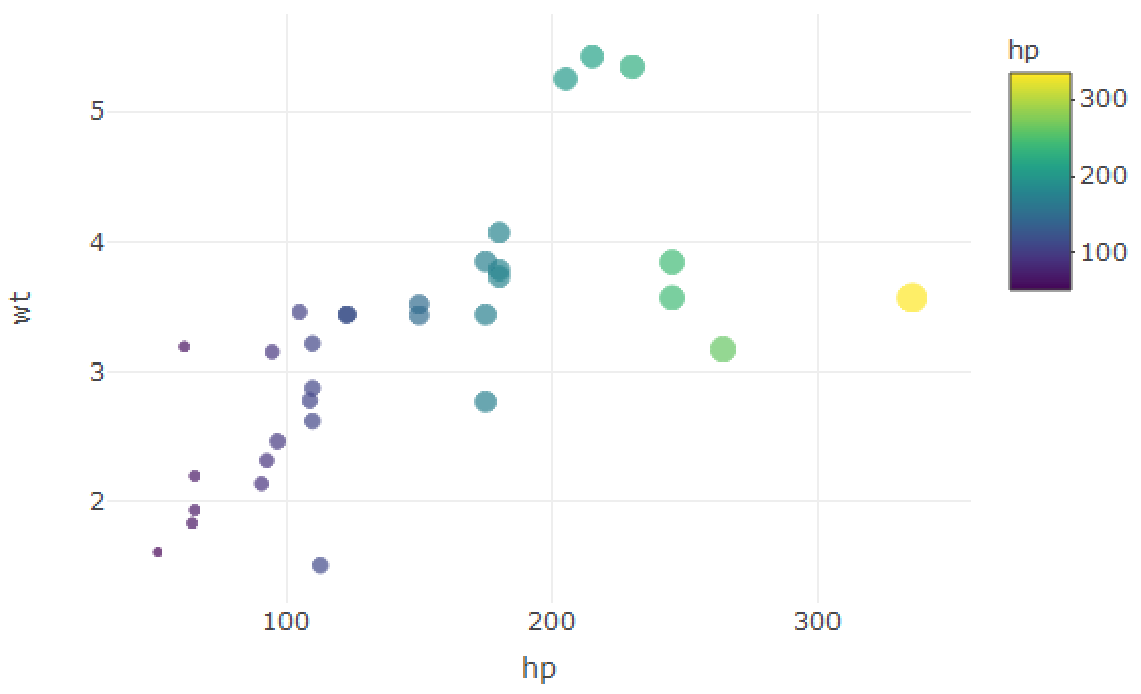
It is also possible to plot markers and lines in the same graph, with plotly. Here we will create an arbitrary data frame to showcase this feature.

```
data1 <- rnorm(100, mean = 10)
data2 <- rnorm(100, mean = 0)
data3 <- rnorm(100, mean = -10)
x <- c(1:100)

data <- data.frame(x, data1, data2, data3)

p <- plot_ly(data, x = ~x)%>%

add_trace(y = ~data1, name = ‘data1’,mode = ‘lines’)%>%
add_trace(y = ~data2, name = ‘data2’, mode = ‘lines+markers’)%>%
add_trace(y = ~data3, name = ‘data3’, mode = ‘markers’)
```



Adding Color and Size Mapping

```
p <- plot_ly(data = t~csa, xy =~hp, y = ~wt,pcolor = ~hp, size = ~hp )
p
```



Although this is not a complete list of the packages used for visualisation in R, these should be enough to get you started.

Visualising Geographical data in R

Geographic data (**Geo data**) relates to the location-based data. It primarily deals with describing objects with respect to their relationship in space. The data is usually stored in the form of coordinates. It makes more sense to be able to see a state or a country in the form of a map as it gives a more realistic overview. In the section below, we will briefly outline the capabilities of R in terms of geographical data visualisation.

Geographical maps

We will be working with a sample superstore dataset of the **ABC company**. The dataset consists of locations of their stores in the US. Let's load in the data and check out its columns.

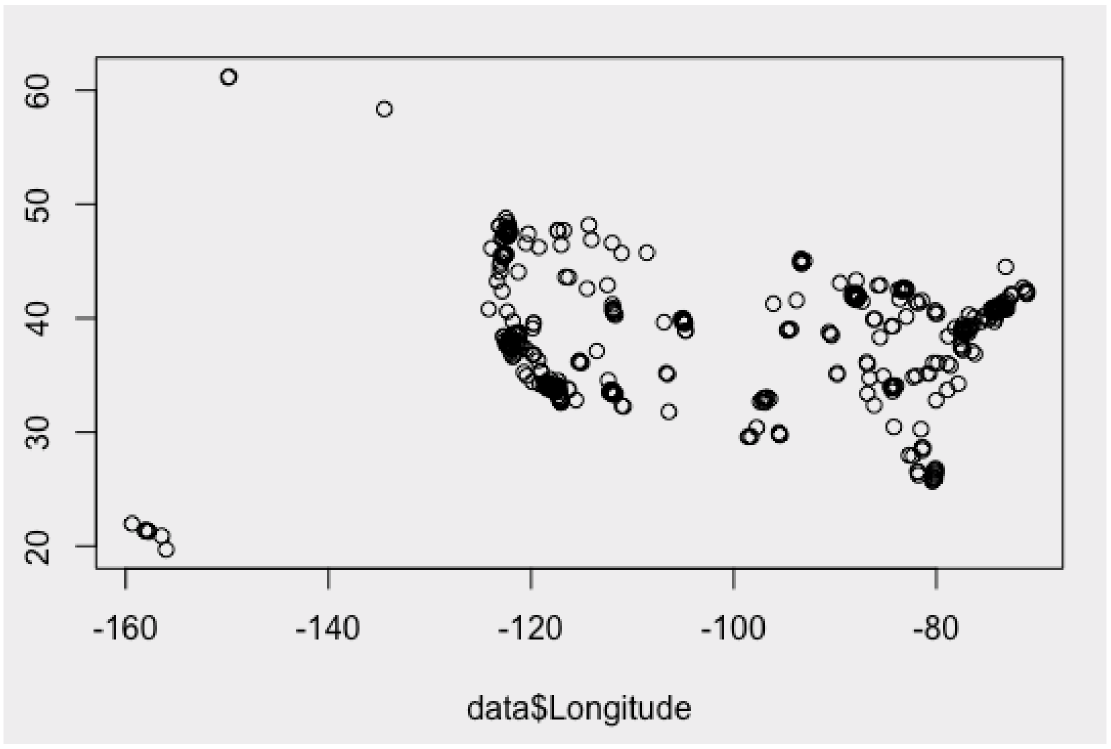
```
data <- read.csv('ABC_locations.csv', sep=",")
```

```
head(data)
                    Address       City   State   Zip.Code Latitude  Longitude
1  1205 N. Memorial Parkway Huntsville Alabama 35801-5930 34.74309  -86.60096
2      3650 Galleria Circle     Hoover Alabama 35244-2346 33.37765  -86.81242
3    8251 Eastchase Parkway Montgomery Alabama      36117 32.36389  -86.15088
4 5225 Commercial Boulevard     Juneau  Alaska 99801-7210 58.35920 -134.48300
5      330 West Dimond Blvd  Anchorage  Alaska 99515-1950 61.14327 -149.88422
6          4125 DeBarr Road  Anchorage  Alaska 99508-3115 61.21081 -149.80434
```

plot() function

We will create a crude map by simply the Latitude and the Longitude column.

```
plot(data$Longitude,data$Latitude)
```



The output isn't an exact map but it does give a faint outline of the US boundary.

map() function

maps package is very useful and pretty straightforward when it comes to plotting the geographical data.

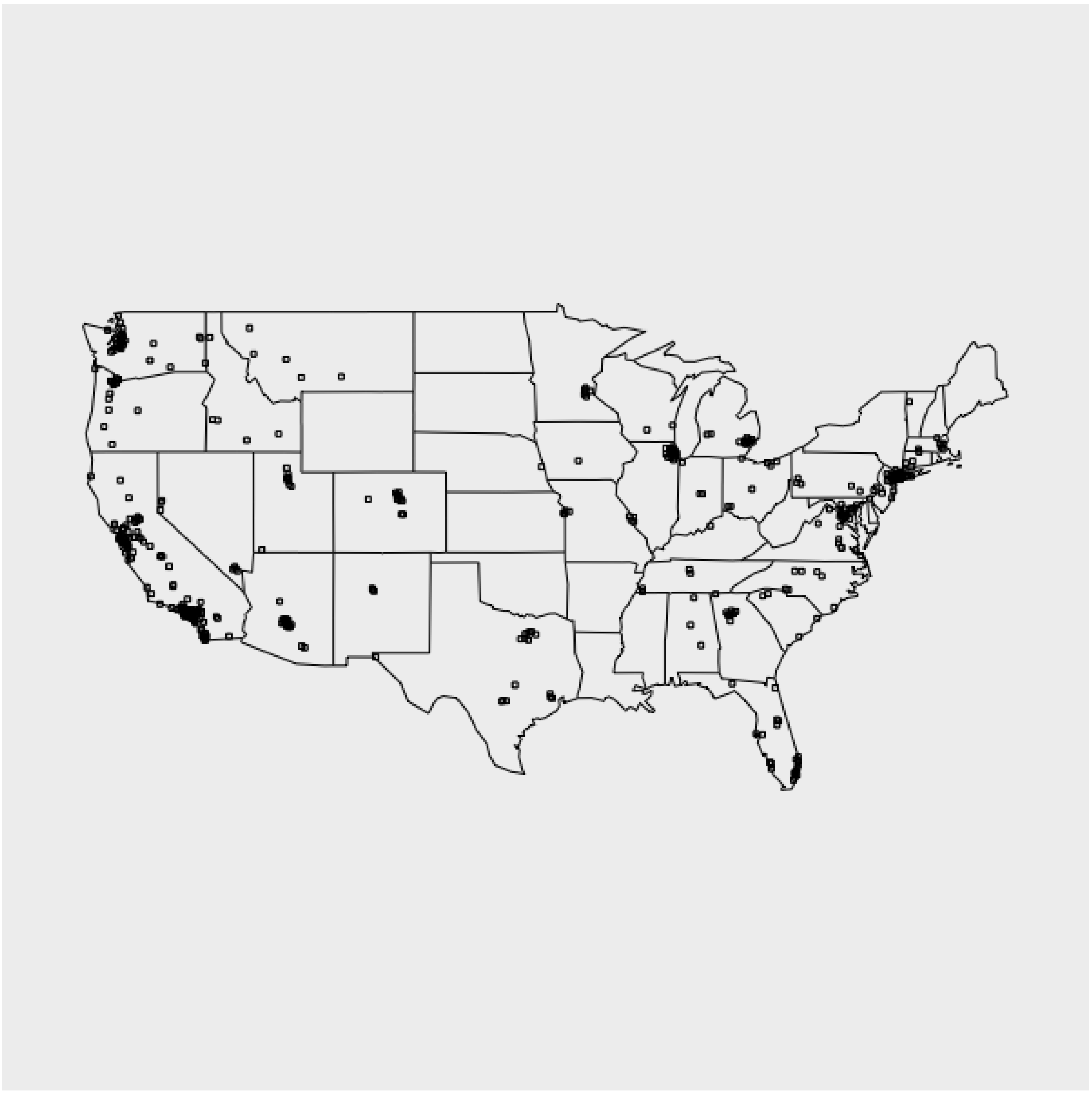
```
# Install package
install.packages("maps", dependencies=TRUE)

# Loading the installed maps package
library(maps)
```

Using the map() function to plot a base map of the US

```
map(database="state")
```

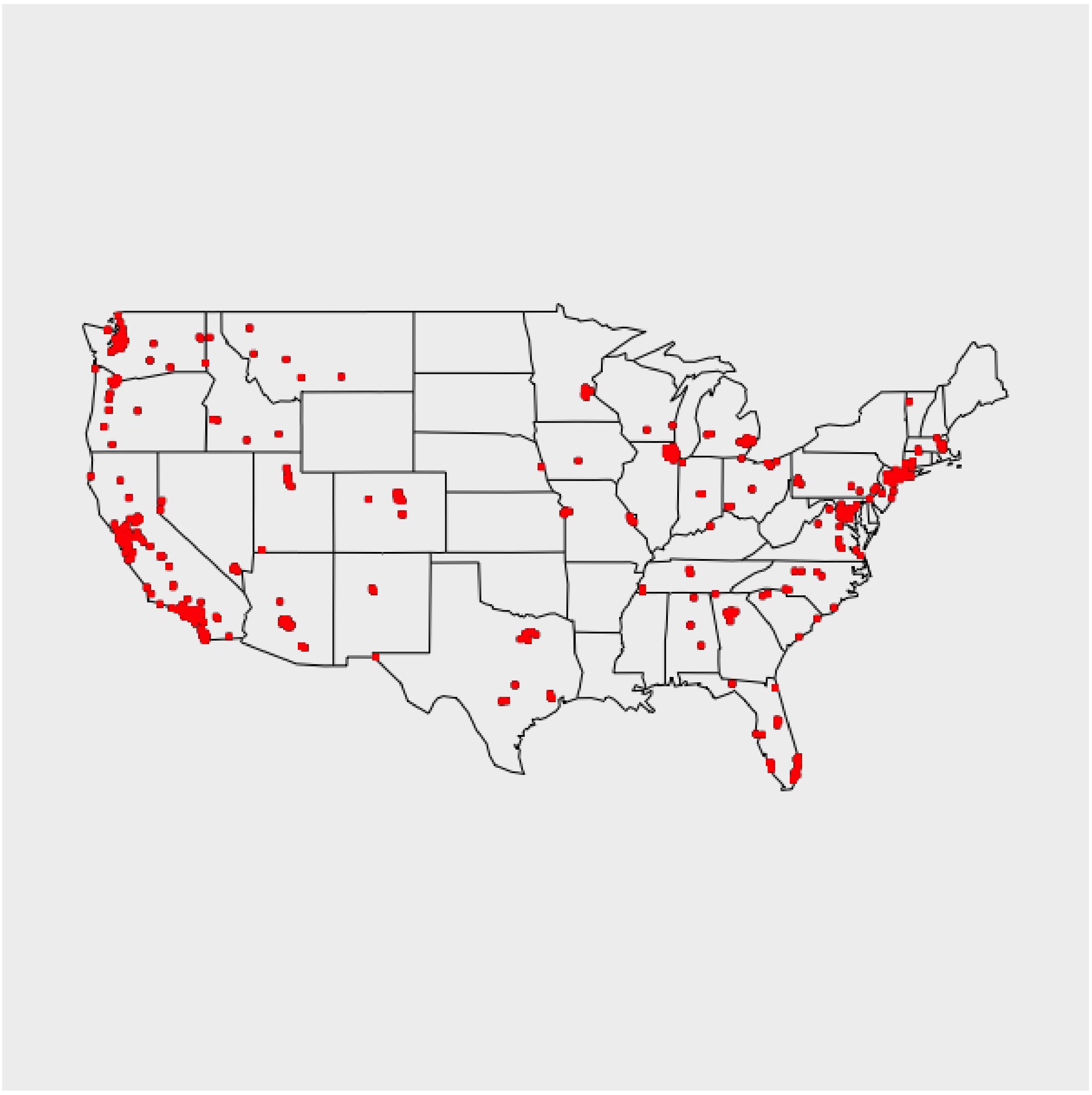
Building a point map on top of the base map using symbols() function

```
symbols(data$Longitude, data$Latitude, squares =rep(1, length(data$Longitude)),
inches=0.03, add=TRUE)
```



Giving the symbols a color

```
symbols(data$Longitude, data$Latitude,bg = 'red', fg = 'red', squares =rep(1,
length(data$Longitude)), inches=0.03, add=TRUE)
```



The commands used with the map function are kind of self-explanatory. However, you can read more about it on their documentation page.

The geographical data visualisation holds a lot of importance where the data consists of locations. One can easily visualise the exact places and areas and convey a better picture.