CHAPTER 4

GRAPHICS USING R

❖ OBJECTIVES

On completion of this Chapter you will be able to:

- understand what is Exploratory Data Analysis
- introduce about the main graphical packages
- draw pie charts using R
- draw scatter plots using R
- draw line plots using R
- draw histograms using R
- draw box plots using R
- draw bar plots using R
- know about other existing graphical packages

4.1. Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a visual based method used to analyse data sets and to summarize their main characteristics. Exploratory Data Analysis (EDA) shows how to use visualisation and transformation to explore data in a systematic way. EDA is an iterative cycle of the below steps:

1) Generate questions about data.
2) Search for answers by visualising, transforming, and modelling data.
3) Use what is learnt to refine questions and/or generate new questions.

Exploratory Data Analysis (EDA) is an approach for data analysis that employs a variety of techniques (mostly graphical) to:

1) Maximize insight into a data set
2) Uncover underlying structure
3) Extract important variables
4) Detect outliers and anomalies
5) Test underlying assumptions
6) Develop parsimonious models
7) Determine optimal factor settings.

4.2. Main Graphical Packages

The basic graphs in R can be drawn using the *base* graphics system. These have some limitations and they are overcome in the next level of graphics called the *grid* graphics system. This system allows to plot the points or lines in the place where desired. But, this does not allow us to draw a scatter plot. Hence, we go for the next level of plotting which the *lattice* graphics system is. In this system, the results of a plot can be saved. Also these scatter plots can contain multiple panels in which we can draw multiple graphs and compare them to each other. The next levels of graphs are the *ggplot2* graphics system. In this the "*gg*" stands for "*grammar of graphics*". This breaks down the graphs into many parts or chunks.

4.3. Pie Charts

In R the pie chart is created using the *pie()* function which takes positive numbers as vector input. The additional parameters are used to control labels, colour, title etc. The basic syntax for creating a pie-chart is as given below and the explanation of the parameters are also listed.

pie(x, labels, radius, main, col, clockwise)

x – numeric vector

labels – description of the slices

radius – values between [-1 to +1]

main – title of the chart

col – colour palette

clockwise – logical value – TRUE (Clockwise), FALSE (Anti Clockwise)

```
> x <- c(25, 35, 10, 5, 15)
> labels <- c("Rose", "Lotus", "Lilly", "Sunflower", "Jasmine")
> pie(x, labels = percent, main = "Flowers", col = rainbow(length(x)))
> legend("topright", c("Rose", "Lotus", "Lilly", "Sunflower", "Jasmine"),
                                        cex = 0.8, fill = rainbow(length(x)))
```

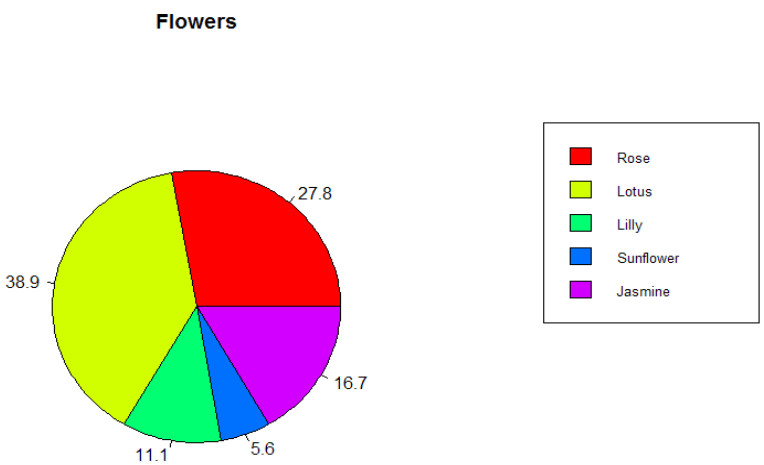

Figure 4.1 Pie Chart of Flowers

A 3D Pie Chart can be drawn using the package *plotrix* which uses the function *pie3D()*.

```
> install.packages("plotrix")
> library(plotrix)
> pie3D(x, labels = labels, explode = 0.1, main = "Flowers")
```

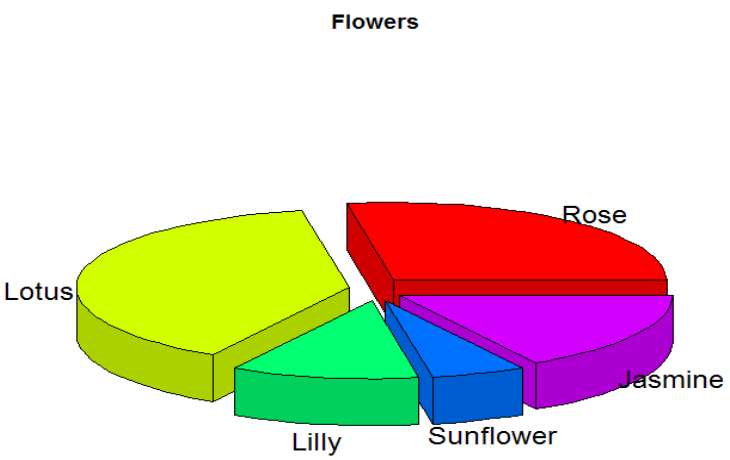

Figure 4.2 3-D Pie Chart of Flowers

4.4. Scatter Plots

Scatter plots are used for exploring the relationship between the two continuous variables. Let us consider the dataset *"cars"* that lists the "Speed and Stopping Distances of Cars". The basic scatter plot in the *base* graphics system can be obtained by using the *plot()* function as in *Fig. 4.3*. The below example compares if the speed of a car has effect on its stopping distance using the plot.

```
> colnames(cars)
[1] "speed" "dist"
> plot(cars$speed, cars$dist)
```

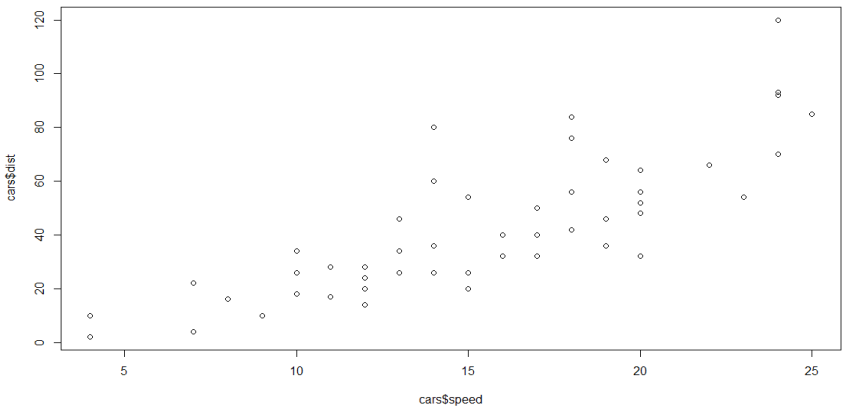

Figure 4.3 Basic Scatter Plot of Car Speed Vs Distance

This plot can be made more appealing and readable by adding colour and changing the plotting character. For this we use the arguments *col* and *pch* (can take the values between 1 and 25) in the *plot()* function as below. Thus the plot in *Fig. 4.4* shows that there is a strong positive correlation between the speed of a car and its stopping distance.

> *plot(cars$speed, cars$dist, col = "red", pch = 15)*

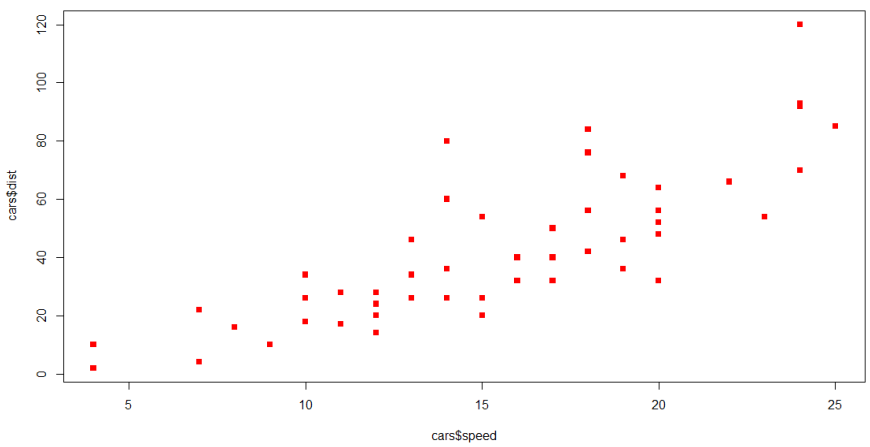

Figure 4.4 Coloured Scatter Plot of Car Speed Vs Distance

The *layout()* function is used to control the layout of multiple plots in the matrix. Thus in the example below multiple related plots are placed in a single figure as in *Fig. 4.5*.

```
> data(mtcars)
> layout(matrix(c(1,2,3,4), 2, 2, byrow = TRUE))
> plot(mtcars$wt, mtcars$mpg, col = "blue", pch = 17)
> plot(mtcars$wt, mtcars$disp, col = "red", pch = 15)
> plot(mtcars$mpg, mtcars$disp, col = "dark green", pch = 10)
> plot(mtcars$mpg, mtcars$hp, col = "violet", pch = 7)
```

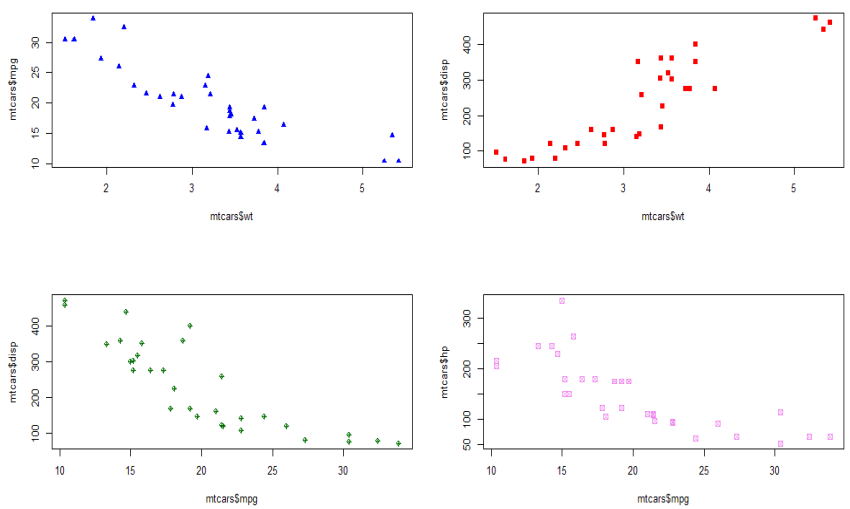

Figure 4.5 Layout of Multiple Scatter Plots

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatter plot matrix. We use *pairs()* function to create matrices of scatter plots as in *Fig. 4.6*. The basic syntax for creating scatter plot matrices in R is as below.

pairs(formula, data)

```
> pairs(~wt+mpg+disp+cyl,data = mtcars, main = "Scatterplot Matrix")
```

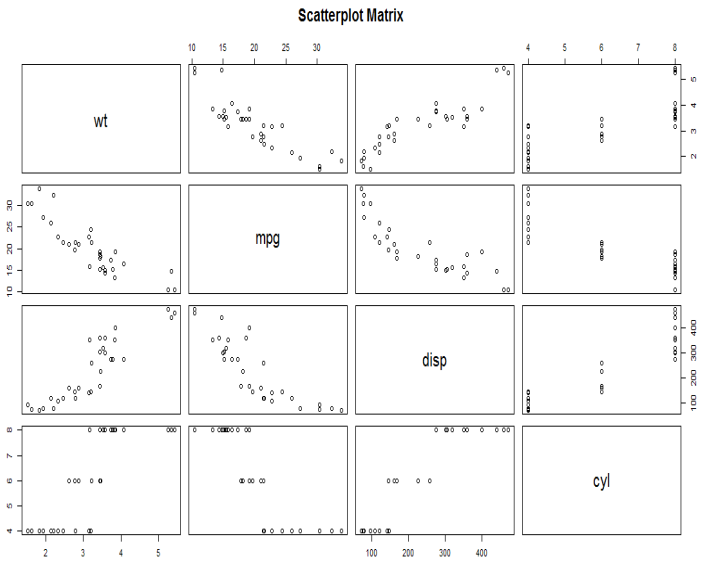

Figure 4.6 Scatter Plot Matrix Using pairs()

The *lattice* graphics system has equivalent of *plot()* function and it is *xyplot()*. This function uses a formula to specify the x and y variables (*yvar* ~ *xvar*) and a data frame argument. To use this function, it is required to include the *lattice* package.

```
> library(lattice)
> xyplot(mtcars$mpg ~ mtcars$disp, mtcars, col = "purple", pch = 7)
```

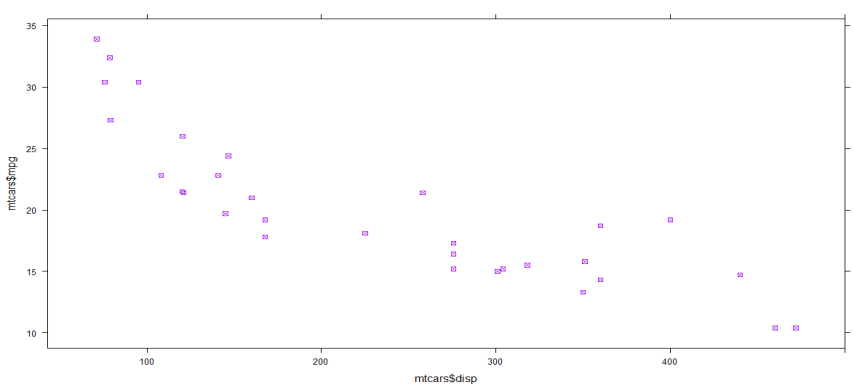

Figure 4.7 Scatter Plot Matrix Using xyplot()

Axis scales can be specified in the *xyplot()* using the *scales* argument and this argument must be a list. This list consists of the *name = value* pairs. If we mention *log = TRUE*, the log scales for the x and y axis are set as in *Fig. 4.8*. The scales list can take other arguments also like the *x* and *y* that sets the x and y axes respectively.

```
> xyplot(mtcars$mpg ~ mtcars$disp, mtcars, scales = list(log = TRUE),
                                                col = "red", pch = 11)
```

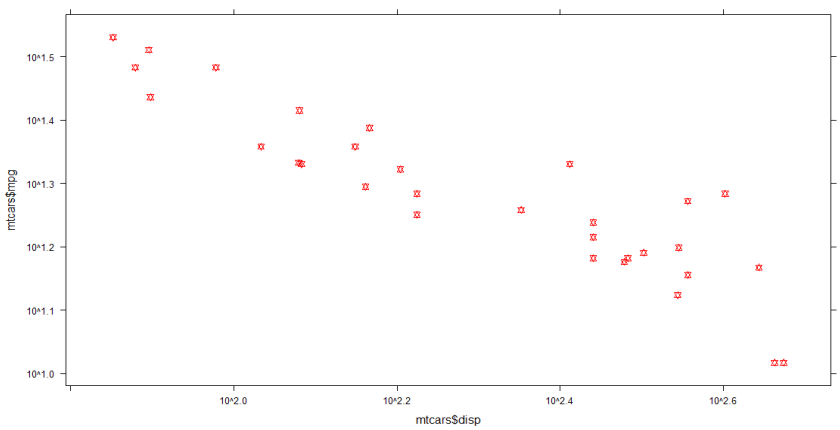

Figure 4.8 Scatter Plot Matrix with Axis Scales Using xyplot()

The data in the graph can be split based on one of the columns in the dataset namely *mtcars$carb*. This can be done by appending the pipe symbol (|) along with the column name used for splitting. The argument *relation = "same"* means that each panel shares the same axes. If the argument *alternating = TRUE*, axis ticks for each panel is drawn on alternating sides of the plot as in *Fig. 4.9*.

```
> xyplot(mtcars$mpg ~ mtcars$disp | mtcars$carb, mtcars,
        scales = list(log = TRUE, relation = "same", alternating = FALSE),
                                        layout = c(3, 2), col = "blue", pch = 14)
```

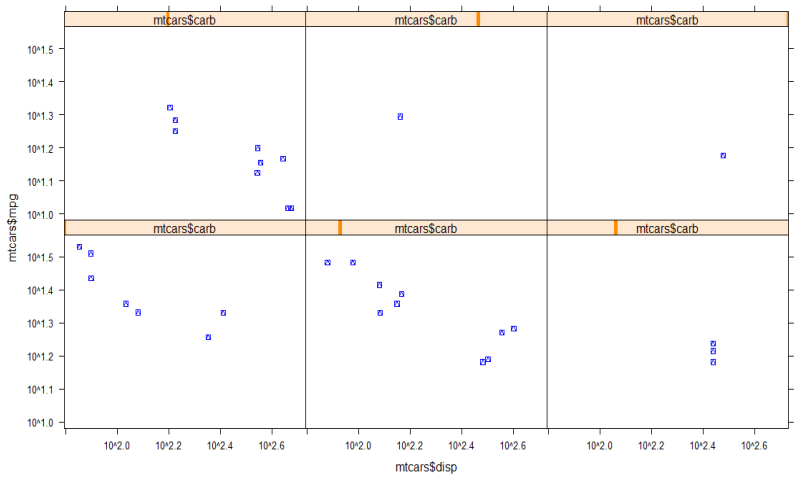

Figure 4.9 Scatter Plot Split on a Column

The lattice plots can be stored in variables and hence they can be further updated using the function update as below.

```
> graph1 <- xyplot(mtcars$mpg ~ mtcars$disp | mtcars$carb, mtcars,
                   scales = list(log = TRUE, relation = "same", alternating = FALSE),
                                                   layout = c(3, 2), col = "blue", pch = 14)
> graph2 <- update(graph1, col = "yellow", pch = 6)
```

In the *ggplot2* graphics, each plot is drawn with a call to the *ggplot()* function as in *Fig. 4.10*. This function takes a data frame as its first argument. The passing of data frame columns to the x and y axis is done using the *aes()* function which is used within the *ggplot()* function. The other aesthetics to the graph are then added using the *geom()* function appended with a "+" symbol to the *ggplot()* function.

```
> library(ggplot2)
> ggplot(mtcars, aes(mpg, disp)) +
                   geom_point(color = "purple", shape = 16, cex = 2.5)
```

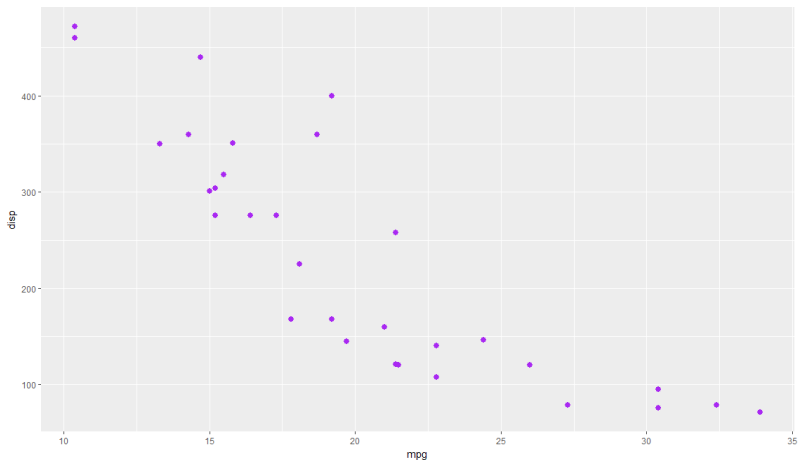

Figure 4.10 Scatter Plot Using ggplot()

The *ggplots* can also be split into several panels like the lattice plots as in *Fig. 4.11*. This is done using the function *facet_wrap()* which takes a formula of the column used for splitting. The function *theme()* is used to specify the orientation of the axis readings. The functions *facet_wrap()* and *theme()* are appended to the *ggplot()* function using the "+" symbol. The *ggplots* can be stored in a variable like the lattice plots and as usual wrapping the expression in parentheses makes it to auto print.

```
> (graph1 <- ggplot(mtcars, aes(mpg, disp)) +
                    geom_point(color = "dark green", shape = 15, cex = 3))
> (graph2 <- graph1 + facet_wrap(~mtcars$cyl, ncol = 3) +
                    theme(axis.text.x = element_text(angle = 90, hjust = 1)
```

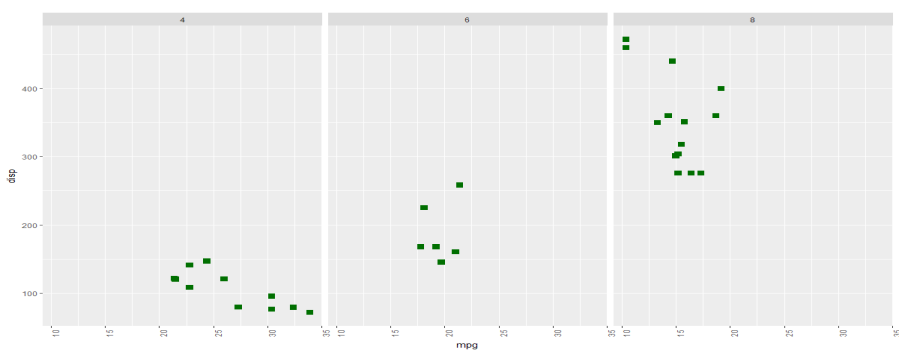

Figure 4.11 – Scatter Plot Split into Panels Using ggplot()

4.5. Line Plots

A line chart / line plot is a graph that connects a series of points by drawing line segments between them. Line charts are usually used in identifying the trends in data. The *plot()* function in R is used to create the line graph in *base* graphics as in *Fig. 4.12*. This function takes a vector of numbers as input together with few more parameters listed below.

plot(v, type, col, xlab, ylab)

v – numeric vector

type - takes value "p" (only points), or "l" (only lines) or "o" (both points and lines)

xlab – label of x-axis

ylab – label of y-axis

main - title of the chart

col - colour palette

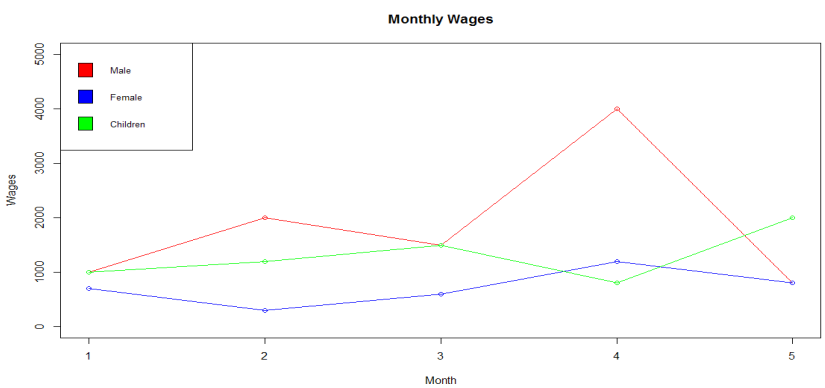

Figure 4.12 Line Plot Using Basic Graphics

> male <- c(1000, 2000, 1500, 4000, 800)

> female <- c(700, 300, 600, 1200, 800)

> child <- c(1000, 1200, 1500, 800, 2000)

> wages <- c("Male", "Female", "Children")

> color = c("red", "blue", "green")

```
> plot(male, type = "o", col = "red", xlab = "Month", ylab = "Wages",
                                    main = "Monthly Wages", ylim = c(0, 5000))
> lines(female, type = "o", col = "blue")
> lines(child, type = "o", col = "green")
> legend("topleft", wages, cex = 0.8, fill = color)
```

Line plots in the *lattice* graphics uses the *xyplot()* function as in *Fig. 4.13*. In this multiple lines can be creating using the "+" symbol in the formula where the x and the y axes are mentioned. The argument *type* = "*l*" is used to mention that it is a continuous line.

```
> xyplot(economics$pop + economics$unemploy ~ economics$date, economics, type = "l")
```

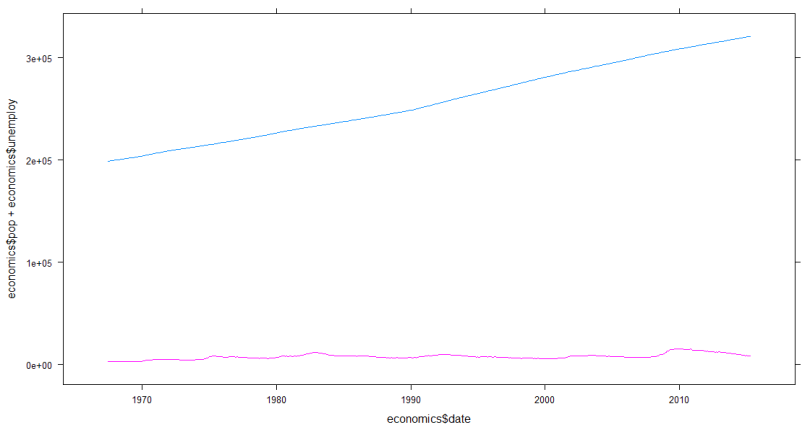

Figure 4.13 Line Plot Using Lattice Graphics

In the *ggplot2* graphics, the same syntax for scatter plots are used, except for the change of *geom_plot()* function with the *geom_line()* function as in *Fig. 4.14*. But, there need to be multiple *geom_line()* functions for multiple lines to be drawn in the graph.

```
> ggplot(economics, aes(economics$date)) + geom_line(aes(y = economics$pop)) +
                                    geom_line(aes(y = economics$unemploy))
```

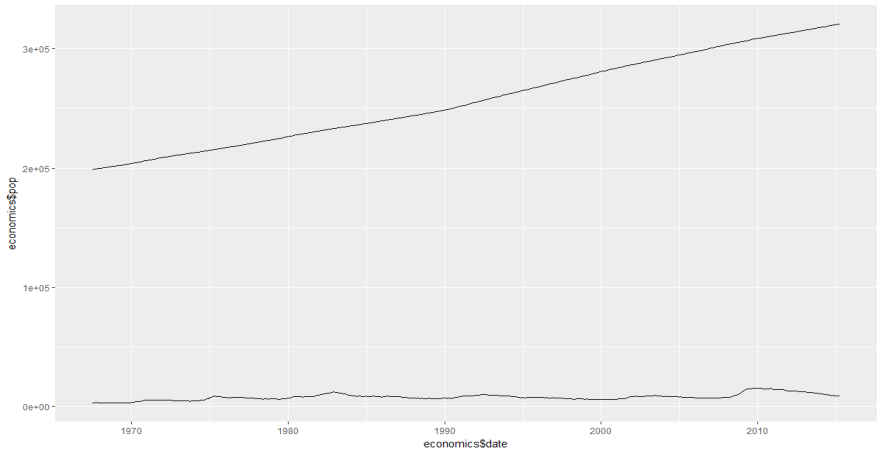

Figure 4.14 Line Plot Using ggplot2 Graphics

The plot in the Fig. 4.15 can be drawn without using multiple *geom_line()* functions also. This is possible using the function *geom_ribbon()* as mentioned below. This function plots not only the two lines, but also the contents in between the two lines.

```
> ggplot(economics, aes(economics$date, ymin = economics$unemploy,
                        ymax = economics$pop)) + geom_ribbon(color = "blue", fill = "white")
```

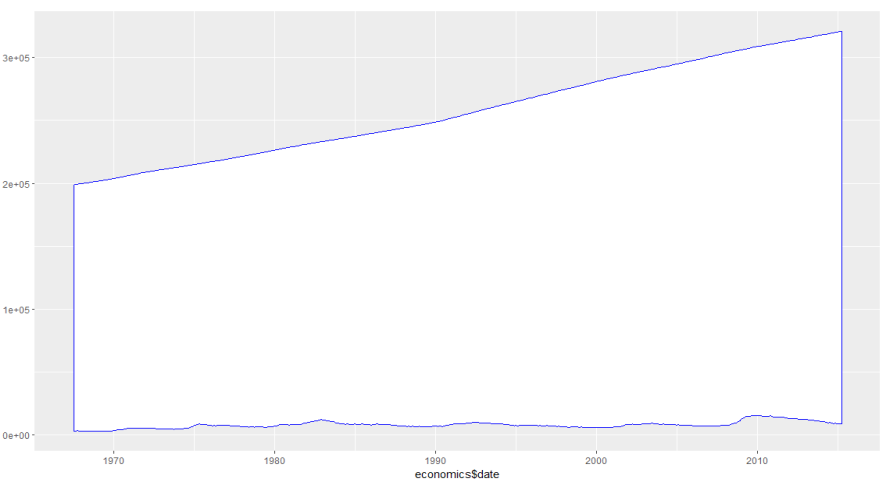

Figure 4.15 Line Plot Using geom_ribbon()

4.6. Histograms

Histograms represents the variable values frequencies, that are split into ranges. This is similar to bar charts, but histograms group values into continuous ranges. In R histograms in the *base* graphics are drawn using the function *hist()* as in the *Fig. 4.16*, that takes a vector of numbers as input together with few more parameters listed below.

hist(v, main, xlab, xlim, ylim, breaks, col, border)

v – numeric vector — *main - title of the chart*

col - colour palette — *border – border colour*

xlab – label of x-axis — *xlim – range of x-axis*

ylim – range of y-axis — *breaks – width of each bar*

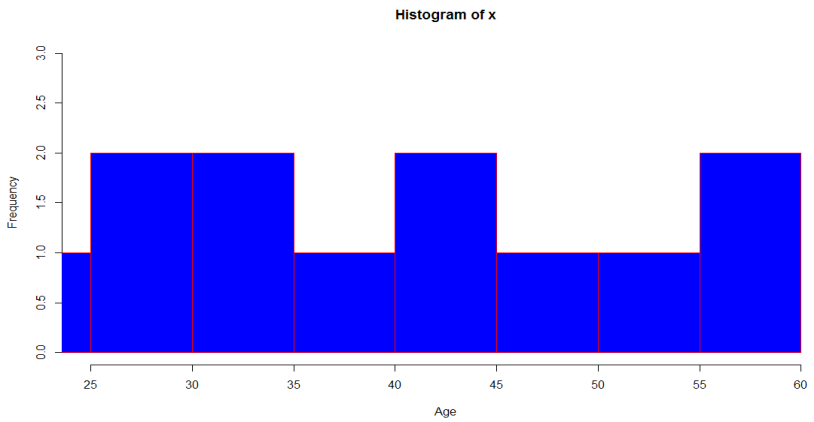

Figure 4.16 Histogram Using Base Graphics

```
> x <- c(45, 33, 31, 23, 58, 47, 39, 58, 28, 55, 42, 27)
> hist(x, xlab = "Age", col = "blue", border = "red", xlim = c(25, 60),
                                                    ylim = c(0, 3), breaks = 5)
```

The *lattice* histogram is drawn using the function *histogram()* as in *Fig. 4.17* and it behaves in the same way as the base ones. But it allows easy splitting of data into panels and saving plots as variables. The *breaks* argument behaves the same way as with *hist()*. The *lattice* histograms support counts, probability densities, and percentage y-axes via the *type* argument, which takes the string "*count*", "*density*", or "*percent*".

> *histogram(~ mtcars$mpg, mtcars, breaks = 10)*

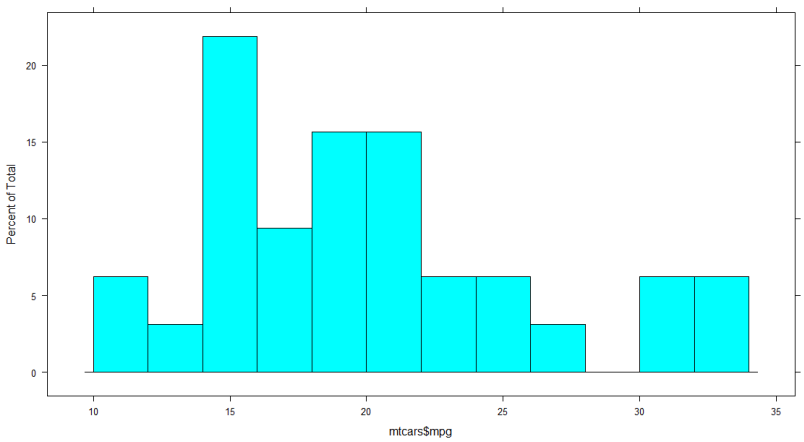

Figure 4.17 Histogram Using Lattice Graphics

The *ggplot* histograms are created by adding the function *geom_histogram()* to the *ggplot()* function as in *Fig. 4.18*. Bin specification is simple here, we just need to pass a numeric bin width to *geom_histogram()* function. It is possible to choose between counts and densities by passing the special names *..count..* or *..density..* to the y-aesthetic.

> *ggplot(mtcars, aes(mtcars$mpg, ..density..)) + geom_histogram(binwidth = 5)*

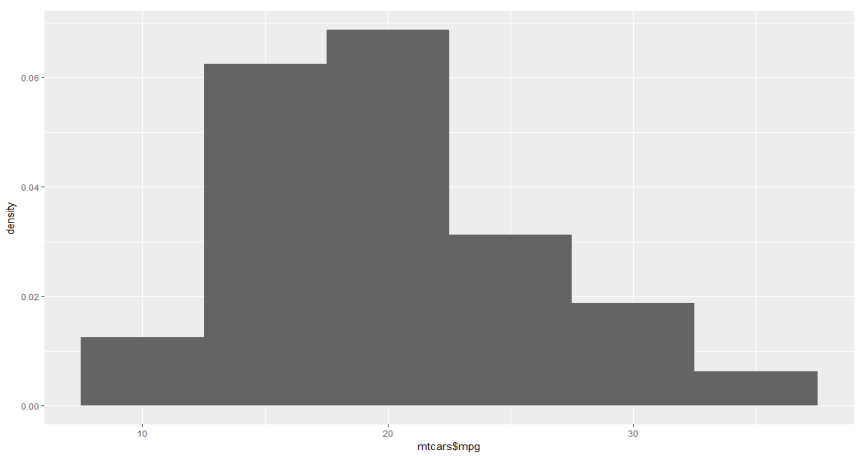

Figure 4.18 Histogram Using ggplot2 Graphics

4.7. Box Plots

The box plot divides the data into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data. This shows the data distribution by drawing the box plots. In R *base* graphics the box plot is created using the *boxplot()* function as in *Fig. 4.19*, which takes the following parameters. The parameters are used to give the data as a data frame, a vector or a formula, a logical value to draw a notch, a logical value to draw a box as per the width of the sample, give title of the chart, labels for the boxes. The basic syntax for creating a box-plot is as given below and the explanation of the parameters are also listed.

boxplot(x, data, notch, varwidth, names, main)

x – vector or a formula

data – data frame

notch – logical value (TRUE – draw a notch)

varwidth – logical value (TRUE – box width proportionate to sample size

names – labels printed under the boxes

main – title of the chart

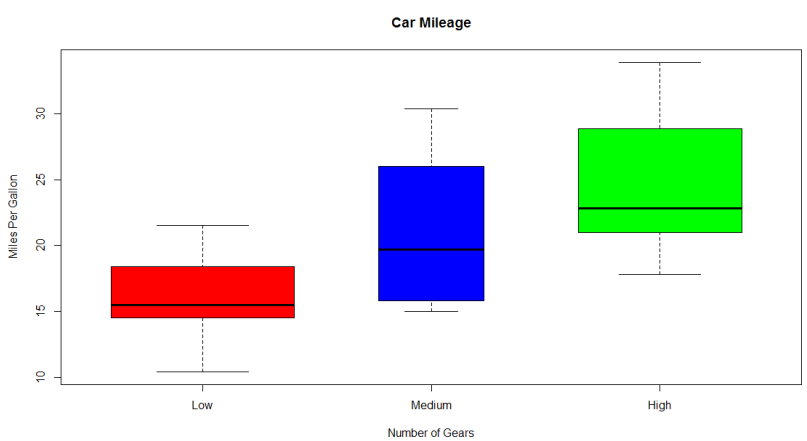

Figure 4.19 Box Plots Using Base Graphics

This type of plot is often clearer if we reorder the box plots from smallest to largest, in some sense. The *reorder()* function changes the order of a factor's levels, based upon some numeric score.

```
> boxplot(mpg ~ reorder(gear, mpg, median), data = mtcars,
                    xlab = "Number of Gears", ylab = "Miles Per Gallon",
                    main = "Car Mileage", varwidth = TRUE,
          col = c("red","blue", "green"), names = c("Low", "Medium", "High"))
```

In the *lattice* graphics the box plot is drawn using the function *bwplot()* as in *Fig. 4.20*.

```
> bwplot(mpg ~ reorder(gear, mpg, median), data = mtcars,
          xlab = "Number of Gears", ylab = "Miles Per Gallon",
          main = "Car Mileage", varwidth = TRUE,
          col = c("red","blue", "green"), names = c("Low", "Medium", "High"))
```

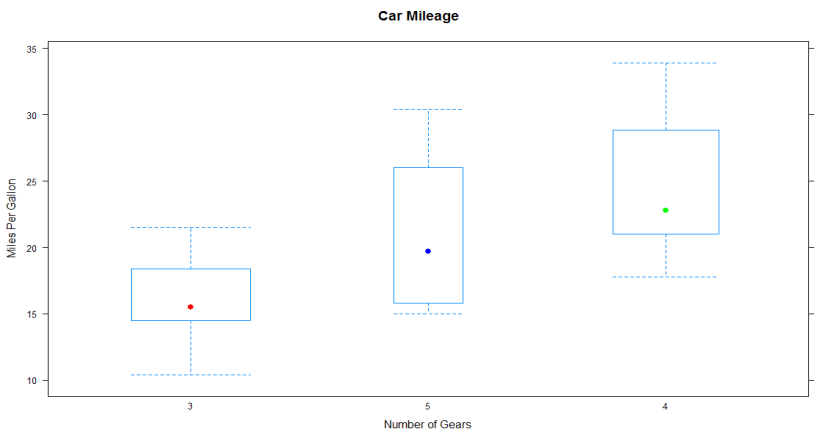

Figure 4.20 Box Plots Using Lattice Graphics

In the *ggplot2* graphics the box plot is drawn by adding the function *geom_boxplot()* to the function *ggplot()* as in *Fig. 4.21*.

```
> ggplot(mtcars, aes(reorder(gear, mpg, median), mpg)) + geom_boxplot()
```

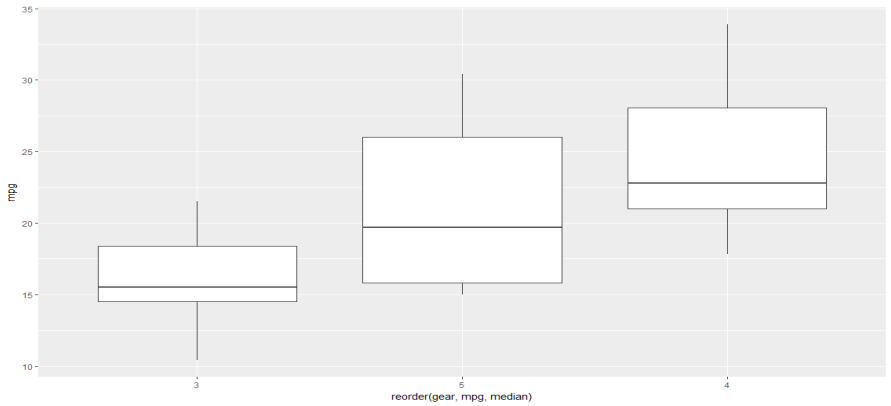

Figure 4.21 Box Plots Using ggplot2 Graphics

4.8. Bar Plots

Bar charts are the natural way of displaying numeric variables split by a categorical variable. In R *base* graphics the bar chart is created using the *barplot()* function as in *Fig. 4.22*, which takes a matrix or a vector of numeric values. The additional parameters are used to give labels to the X-axis, Y-axis, give title of the chart, labels for the bars and colours. The basic syntax for creating a bar-chart is as given below and the explanation of the parameters are also listed.

barplot(H, xlab, ylab, main, names.arg, col)

H – numeric vector or matrix

x-lab – label of x-axis

y-lab – label of y-axis

main - title of the chart

names.arg – vector of labels under each bar

col – colour palette

```
> x <- matrix(c(1000, 900, 1500, 4400, 800, 2100, 1700, 2900, 3800),
                                                    nrow = 3, ncol = 3)
> years <- c("2011", "2012", "2013")
> city <- c("Chennai", "Mumbai", "Kolkata")
```

```
> color <- c("red", "blue", "green")
> barplot(x, main = "Yearly Sales", names.arg = years, xlab = "Year",
                                        ylab = "Sales", col = color)
> legend("topleft", city, cex = 0.8, fill = color)
```

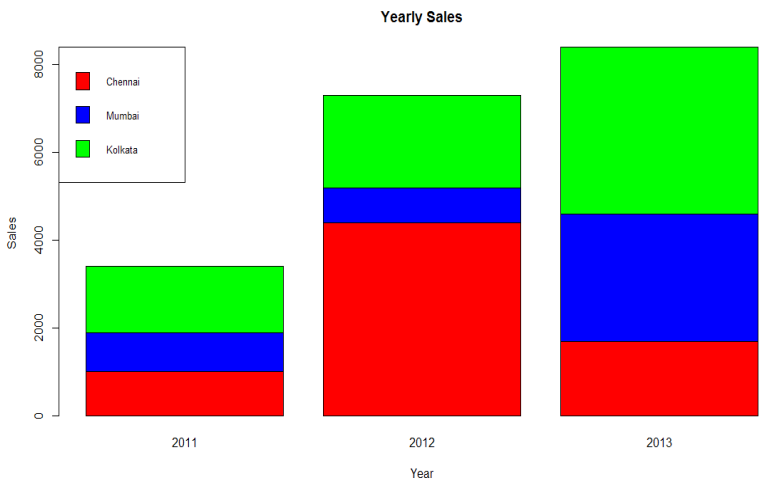

Figure 4.22 Vertical Bar Plot Using Base Graphics

By default the bars are vertical, but if we want horizontal bars, they can be generated with *horiz = TRUE* parameter as in *Fig. 4.23*. We can also do some fiddling with the plot parameters, via the *par()* function. The *las* parameter controls whether labels are horizontal, vertical, parallel, or perpendicular to the axes. Plots are usually more readable if you set *las = 1*, for horizontal. The *mar* parameter is a numeric vector of length 4, giving the width of the plot margins at the bottom/left/top/right of the plot.

```
> x <- matrix(c(1000, 900, 1500, 4400, 800, 2100, 1700, 2900, 3800), nrow = 3, ncol = 3)
> years <- c("2011", "2012", "2013")
```

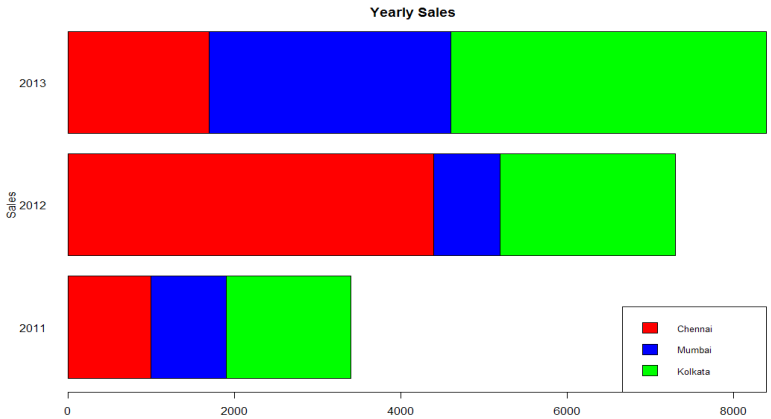

Figure 4.23 Horizontal Bar Plot Using Base Graphics

```
> city <- c("Chennai", "Mumbai", "Kolkata")
> color <- c("red", "blue", "green")
> par(las = 1, mar = c(3, 9, 1, 1))
> barplot(x, main = "Yearly Sales", names.arg = years,
                                    xlab = "Year", ylab = "Sales", col = color, horiz = TRUE)
> legend("bottomright", city, cex = 0.8, fill = color)
```

The *lattice* equivalent of the function *barplot()*, is the function *barchart()* as shown in *Fig. 4.24*. The formula interface is the same as those we saw with scatter plots, *yvar ~ xvar*.

```
> barchart(mtcars$mpg ~ mtcars$disp, mtcars)
```

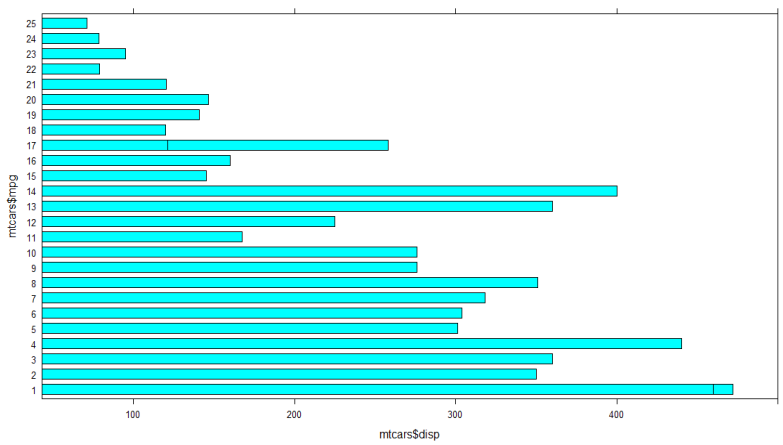

Figure 4.24 Horizontal Bar Plot Using Lattice Graphics

Extending this to multiple variables just requires a tweak to the formula, and passing *stack = TRUE* to make a stacked plot as in *Fig. 4.25*.

```
> barchart(mtcars$mpg ~ mtcars$disp + mtcars$qsec + mtcars$hp, mtcars,
                                                          stack = TRUE
```

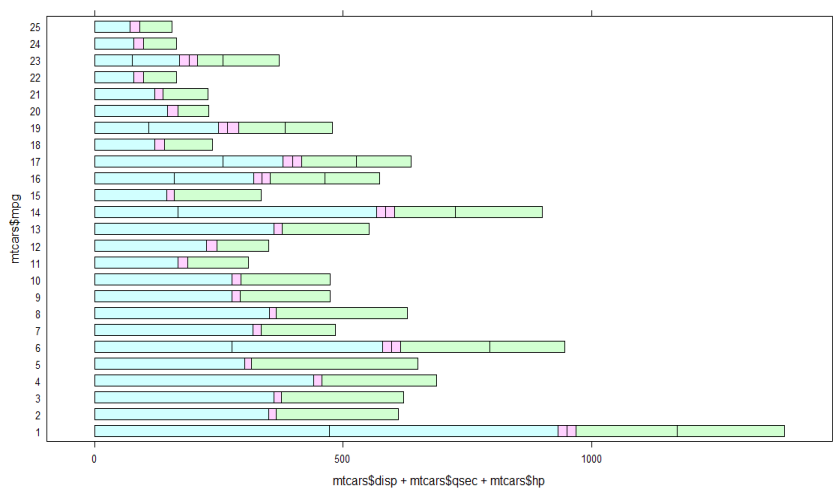

Figure 4.25 Horizontal Stacked Bar Plot Using Lattice Graphics

In the *ggplot2* graphics the bar chart is drawn by adding the function *geom_bar()* to the function *ggplot()* as in *Fig. 4.26*. Like *base*, *ggplot2* defaults to vertical bars; adding the function *coord_flip()* swaps this. We must pass the argument *stat = "identity"* to the function *geom_bar()*.

```
> ggplot(mtcars, aes(mtcars$mpg, mtcars$disp)) + geom_bar(stat = "identity") +
                                                              coord_flip()
```

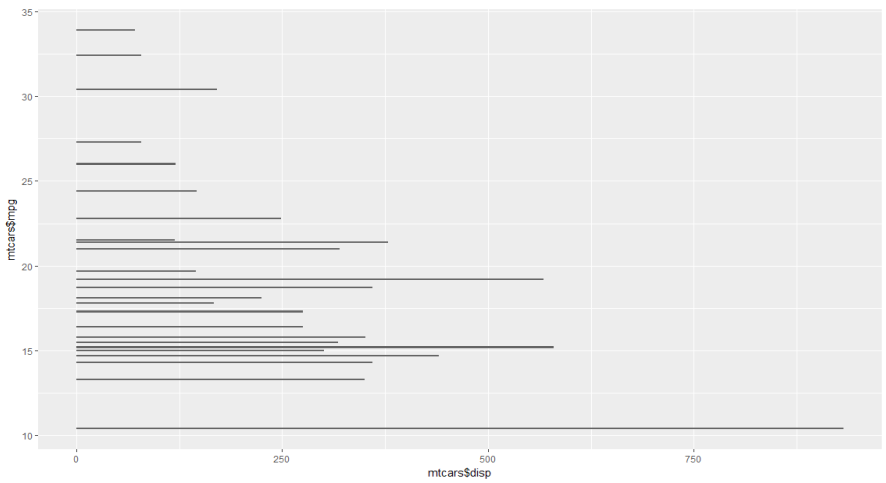

Figure 4.26 Horizontal Bar Plot Using ggplot2 Graphics

4.9. Other Graphical Packages

Package Name	Description
Vcd	Plots for visualizing categorical data, such as mosaic plots and association plots
Plotrix	Loads of extra plot types
latticeExtra	Extends the *lattice* package
GGally	Extend the *ggplot2* package
Grid	Provide access to the underlying framework of lattice and ggplot2 packages

Package Name	Description
gridSVG	Write grid-based plots to SVG files
Playwith	Allows pointing and clicking to interact with base or lattice plots
Iplots	Provides a whole extra system of plots with more interactivity
googleVis	Provides an R wrapper around Google Chart Tools, creating plots that can be displayed in a browser.
Rggobi	Provides an interface to GGobi package (for visualizing high-dimensional data)
GGobi	Open source visualization program for exploring high-dimensional data.
Rgl	Provides an interface to OpenGL for interactive 3D plots
Animation	Lets to make animated GIFs or SWF animations
rCharts	Provides wrappers to half a dozen JavaScript plotting libraries using lattice syntax.

❖ HIGHLIGHTS

- Exploratory Data Analysis (*EDA*) shows how to use visualisation and transformation to explore data in a systematic way.
- The main graphical packages are *base*, *lattice* and *ggplot*2.
- In R the pie chart is created using the *pie()* function.
- A 3D Pie Chart can be drawn using the package plotrix which uses the function *pie3D()*.
- The basic scatter plot in the base graphics system can be obtained by using the *plot()* function.
- We use the arguments *col* and *pch* (values between 1 and 25) in the *plot()* function to specify colour and plot pattern.
- The *layout()* function is used to control the layout of multiple plots in the matrix.

- We use *pairs()* function to create matrices of scatter plots.
- The *lattice* graphics system has equivalent of *plot()* function and it is *xyplot()*.
- In the *ggplot2* graphics, each plot is drawn with a call to the *ggplot()* function.
- The *ggplots* can also be split into several panels like the *lattice* plots and this is done using the function *facet_wrap()*.
- The *plot()* function in R is used to create the line graph in base graphics and the argument *type* = "*l*" is used to mention that it is a line.
- Line plots in the *lattice* graphics uses the *xyplot()* function and the argument *type* = "*l*" is used to mention that it is a line.
- The *ggplot2* scatter plot are created by adding the function *geom_line()* to the *ggplot()* function.
- In R histograms in the *base* graphics are drawn using the function *hist()*.
- The *lattice* histogram is drawn using the function *histogram()*.
- The *ggplot2* histograms are created by adding the function *geom_histogram()* to the *ggplot()* function.
- In R *base* graphics the box plot is created using the *boxplot()* function.
- In the *lattice* graphics the box plot is drawn using the function *bwplot()*.
- In the *ggplot2* graphics the box plot is drawn by adding the function *geom_boxplot()* to the function *ggplot()*.
- In R base graphics the bar chart is created using the *barplot()* function.
- The lattice equivalent of the function *barplot()*, is the function *barchart()*.
- In the *ggplot2* graphics the bar chart is drawn by adding the function *geom_bar()* to the function *ggplot()*.