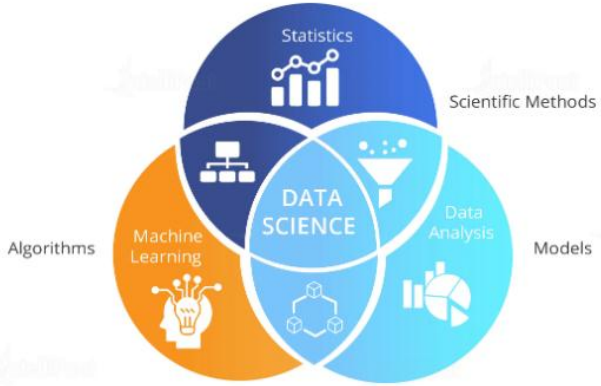
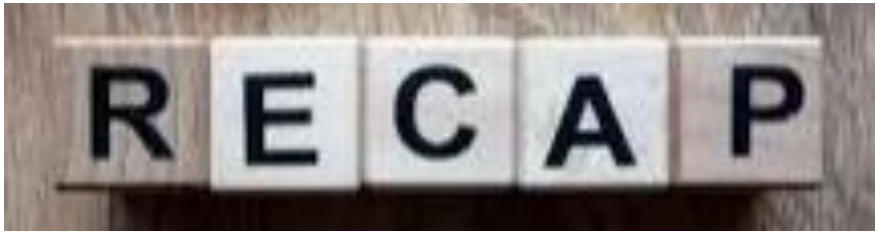
Workshop on Data Science using R



A. Bamila Virgin Louis, Asst.Prof
Department of Computer Science and Engineering

St.Xavier's Catholic College of Engineering
Chunkankadai, Nagercoil, Kanyakumari District



Control structures

Functions

String

Subsetting R Objects

There are three operators that can be used to extract subsets of R objects.

- [ ]operator -returns an object of the same class as the original. It can be used to select multiple elements of an object
- [[ ]] operator is used to extract elements of a list or a data frame. It can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame.
- $ operator is used to extract elements of a list or data frame by literal name. Its semantics are similar to that of [[ ]].

Subsetting a Vector

- Vectors can be subsetted using the [ ] operator.

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1]    ## Extract the first element
[1] "a"
> x[2]    ## Extract the second element
[1] "b"
```

- To extract multiple elements of a vector we can use an integer sequence.

```
> x[1:4]
[1] "a" "b" "c" "c"
```

- We can specify any arbitrary integer vector.

```
> x[c(1, 3, 4)]
[1] "a" "c" "c"
```

Subsetting a Vector

- We can also pass a logical sequence to the [ ] operator to extract elements of a vector that satisfy a given condition

```
> u <- x > "a"
> u
[1] FALSE  TRUE  TRUE  TRUE  TRUE FALSE
> x[u]
[1] "b" "c" "c" "d"
```

- We can subset the vector directly with the logical expression.

```
> x[x > "a"]
[1] "b" "c" "c" "d"
```

Subsetting a Matrix

```
> x <- matrix(1:6, 2, 3)
> x
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
> x[1, 2]
[1] 3
> x[2, 1]
[1] 2
```

```
> x[1, ]   ## Extract the first row
[1] 1 3 5
> x[, 2]   ## Extract the second column
[1] 3 4
```

Subsetting Lists

```
> x<-list(num=1:4,avg=0.6)
> x
$num
[1] 1 2 3 4

$avg
[1] 0.6
```

- The [[ ]] operator can be used to extract *single* elements from a list.

```
> x[[1]]
[1] 1 2 3 4
```

- The [[ ]] operator can also use named indices

```
> x[["avg"]]
[1] 0.6
> x$"avg"
[1] 0.6
> x$avg
[1] 0.6
```

Subsetting Nested Elements of a List

- The [[ ]] operator can take an integer sequence to extract a nested element of a list.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
>
> ## Get the 3rd element of the 1st element
> x[[c(1, 3)]]
[1] 14
>
> ## Same as above
> x[[1]][[3]]
[1] 14
>
> ## 1st element of the 2nd element
> x[[c(2, 1)]]
[1] 3.14
```

Extracting Multiple Elements of a List

- The [ ] operator can be used to extract *multiple* elements from a list.

```
> x <- list(num = 1:4, point = 0.6, text = "hello")
> x[c(1,3)]
$num
[1] 1 2 3 4

$text
[1] "hello"
```

- Note that x[c(1, 3)] is NOT the same as x[[c(1, 3)]].

```
> x[[c(1,3)]]
[1] 3
```

Vectorized Operations

- Arithmetic operations on two vectors.

```
> x <- 1:4
> y <- 6:9
> z <- x + y
> z
[1]  7  9 11 13
```

```
> x - y
[1] -5 -5 -5 -5
> x * y
[1]  6 14 24 36
> x / y
[1] 0.1666667 0.2857143 0.3750000 0.4444444
```

- Logical comparisons

```
> x
[1] 1 2 3 4
> x > 2
[1] FALSE FALSE  TRUE  TRUE
```

```
> x >= 2
[1] FALSE  TRUE  TRUE  TRUE
> x < 3
[1]  TRUE  TRUE FALSE FALSE
> y == 8
[1] FALSE FALSE  TRUE FALSE
```

Vectorized Matrix Operations

- Matrix operations are also vectorized.
- We can do element-by-element operations on matrices without having to loop over every element.

```
> x <- matrix(1:4, 2, 2)
> y <- matrix(rep(10, 4), 2, 2)
> x
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> y
     [,1] [,2]
[1,]   10   10
[2,]   10   10
```

```
> ## element-wise multiplication
> x * y
     [,1] [,2]
[1,]   10   30
[2,]   20   40
>
> ## element-wise division
> x / y
     [,1] [,2]
[1,]  0.1  0.3
[2,]  0.2  0.4
>
> ## true matrix multiplication
> x %*% y
     [,1] [,2]
[1,]   40   40
[2,]   60   60
```

Reading Data

- read.table, read.csv, for reading tabular data
- readLines, for reading lines of a text file
- source, for reading in R code files (inverse of dump)
- dget, for reading in R code files (inverse of dput)

Writing Data

- write.table, for writing tabular data to text files (i.e. CSV) or connections
- writeLines, for writing character data line-by-line to a file or connection
- dump, for dumping a textual representation of multiple R objects
- dput, for outputting a textual representation of an R object

The read.table() function

- file, the name of a file, or a connection
- header, logical indicating if the file has a header line
- sep, a string indicating how the columns are separated
- colClasses, a character vector indicating the class of each column in the dataset
- nrows, the number of rows in the dataset. By default read.table() reads an entire file.
- comment.char, a character string indicating the comment character. This defalts to "#". If there are no commented lines in your file, it’s worth setting this to be the empty string "".
- skip, the number of lines to skip from the beginning

Using Textual Formats for Storing Data

- The dump() and dput() functions are useful because the resulting textual format is editable, and in the case of corruption, potentially recoverable.
- Unlike writing out a table or CSV file, dump() and dput() preserve the *metadata,* so that another user doesn't have to specify it all over again.
- For example, we can preserve the class of each column of a table or the levels of a factor variable.

Using dput() and dget()

- the dput() output is in the form of R code and that it preserves metadata like the class
-
```
> y <- data.frame(a = 1, b = "a")
> dput(y)
structure(list(a = 1, b = "a"), class = "data.frame", row.names = c(NA,
-1L))
```
- The output of dput() can also be saved directly to a file.

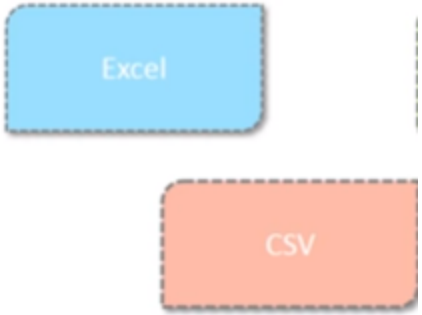
```
> ## Send 'dput' output to a file
> dput(y, file = "y.R")
> ## Read in 'dput' output from a file
> new.y <- dget("y.R")
> new.y
  a b
1 1 a
```

dump() and source()

- Multiple objects can be deparsed at once using the dump function and read back in using source.

```
> y<-data.frame(a=1L,b="a")
> x<-"Hello"
> dump(c("x", "y"), file = "data.R")
> rm(x, y)
> source("data.R")
> str(y)
'data.frame':   1 obs. of  2 variables:
 $ a: int 1
 $ b: chr "a"
> x
[1] "Hello"
> str(x)
 chr "Hello"
> y
  a b
1 1 a
```

Interfaces to the Outside World



- Data are read in using *connection* interfaces.

Connections can be made to

- file, opens a connection to a file
- url, opens a connection to a webpage

connections are powerful tools: navigate files or other external objects (DB, txt file, Web service API)

File Connections

- Connections to text files can be created with the file() function.

```
> str(file)
function (description = "", open = "", blocking = TRUE, encoding = getOption("encoding"),
    raw = FALSE, method = getOption("url.method", "default"))
```

- description is the name of the file
- open is a code indicating what mode the file should be opened in

  "r" open file in read only mode

  "w" open a file for writing (and initializing a new file)

  "a" open a file for appending

  "rb", "wb", "ab" reading, writing, or appending in binary mode (Windows)

File connections

- In practice, we often don't need to deal with the connection interface directly as many functions for reading and writing data just deal with it in the background.

```
> ## Create a connection to 'sample.txt'
> con <- file("sample.txt")
> ## Open connection to 'sample.txt' in read-only mode
> open(con, "r")
> ## Read from the connection
> data <- read.csv(con)
> ## Close the connection
> close(con)
```

which is the same as

```
> data <- read.csv("sample.txt")
```

Getting and Setting the Working Directory

```
> print(getwd())
[1] "C:/Users/User/Documents"

> setwd("C:/Users/User/Desktop")
```

CSV Files

- The csv file is a text file in which the values in the columns are separated by a comma.

```
Id,Name,Gender,Dept,Profit
102,Lancia,Female,IT,300
204,Christon,Male,Operations,280
105,Amali,Female,IT,320
303,Helena,Female,Finance,400
207,Nestricia,Female,Operations,310
403,Jeswin,Male,HR,260
405,Jino,Male,HR,280
106,Angel,Female,IT,310
208,Infanto,Male,Operations,330
409,Shane,Male,Finance,430
```

```
> data<-read.csv("input.csv")
> data
    Id      Name Gender       Dept Profit
1  102    Lancia Female         IT    300
2  204  Christon   Male Operations    280
3  105     Amali Female         IT    320
4  303    Helena Female    Finance    400
5  207 Nestricia Female Operations    310
6  403    Jeswin   Male         HR    260
7  405      Jino   Male         HR    280
8  106     Angel Female         IT    310
9  208   Infanto   Male Operations    330
10 409     Shane   Male    Finance    430
```

Analyzing the CSV File

- By default the **read.csv()** function gives the output as a data frame.

```
> data<-read.csv("input.csv")
> print(is.data.frame(data))
[1] TRUE
> print(ncol(data))
[1] 4
> print(nrow(data))
[1] 10
```

```
> #get maximum profit
> prof<-max(data$Profit)
> print(prof)
[1] 430
```

```
> #get persorn detail having maximum profit
> value<-subset(data,Profit==max(Profit))
> print(value)
    Id  Name Gender    Dept Profit
10 409 Shane   Male Finance    430
```

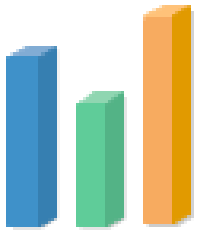
Writing into a CSV File

```
> maxvalue<-subset(data,Profit>360)
> write.csv(maxvalue,"output.csv")
> newdata<-read.csv("output.csv")
> print(newdata)
   X  Id   Name Gender    Dept Profit
1  4 303 Helena Female Finance    400
2 10 409  Shane   Male Finance    430
```

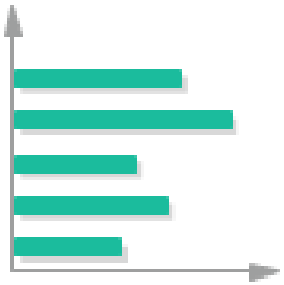
```
> minvalue<-subset(data,Profit<360)
> write.csv(minvalue,"output2.csv")
> otherdata<-read.csv("output2.csv")
```

```
> print(otherdata)
  X  Id      Name Gender       Dept Profit
1 1 102    Lancia Female         IT    300
2 2 204  Christon   Male Operations    280
3 3 105     Amali Female         IT    320
4 5 207 Nestricia Female Operations    310
5 6 403    Jeswin   Male         HR    260
6 7 405      Jino   Male         HR    280
7 8 106     Angel Female         IT    310
8 9 208   Infanto   Male Operations    330
```

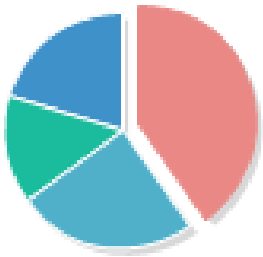
Data Visualization



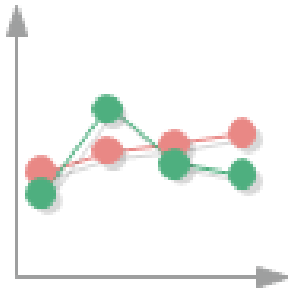
Column



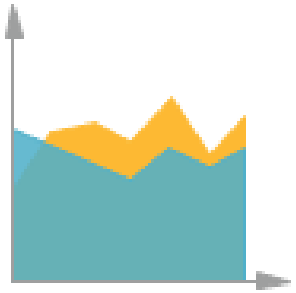
Bar



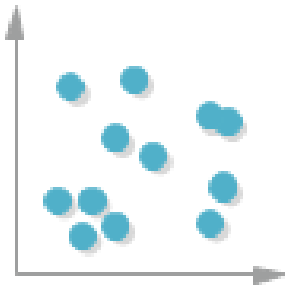
Pie



Line



Area



Scatter

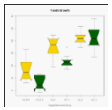
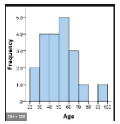
Charts & Graphs

R Programming language has numerous libraries to create charts and graphs.

- Pie Charts: Compare parts of a whole 

- Bar graphs: Compare things between different groups or track changes over time 

- Scatter plot: Show how much one variable affected by others 

- Boxplot: Summarize data from multiple sources and display result in single graph. 
- Histogram: Plot frequency of score occurrences in a continuous data set. 
- Line graphs: Tracks changes over short or long period of time. 

Pie-chart

- R Programming language has numerous libraries to create charts and graphs.
- A pie-chart is a representation of values as slices of a circle with different colors.
- pie chart is created using the **pie()** function which takes positive numbers as a vector input.

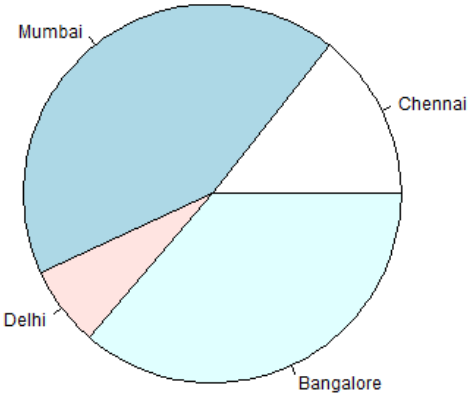
```
pie(x, labels, radius, main, col, clockwise)
```

- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise or anti clockwise.

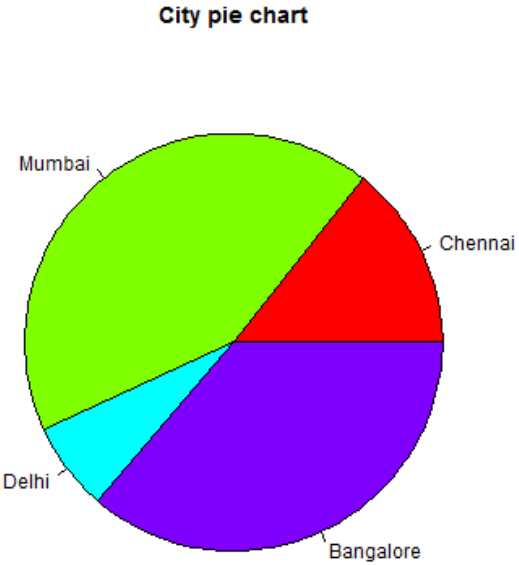
Simple pie-chart

- Create and save the pie chart in the current R working directory.

```
> # Create data for the graph.
> x <- c(21, 62, 10, 53)
> labels <- c("Chennai", "Mumbai", "Delhi", "Bangalore")
>
> # Give the chart file a name.
> png(file = "city.jpg")
>
> # Plot the chart.
> pie(x,labels)
>
> # Save the file.
> dev.off()
```

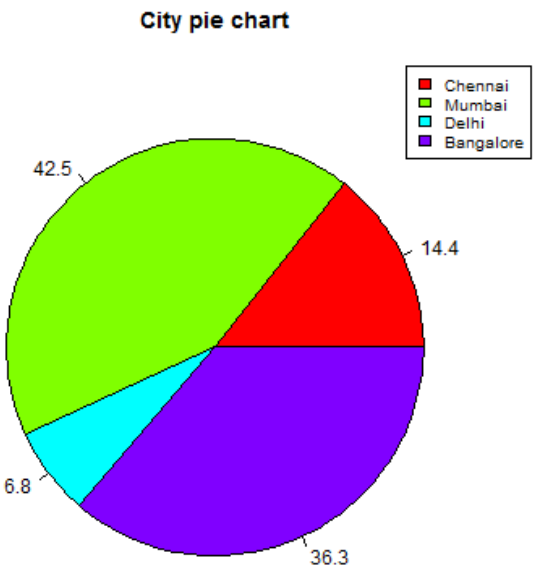


Pie Chart Title and Colors




```
> # Create data for the graph.
> x <- c(21, 62, 10, 53)
> labels <- c("Chennai", "Mumbai", "Delhi", "Bangalore")
>
> # Give the chart file a name.
> png(file = "city_title_colours.jpg")
>
> # Plot the chart with title and rainbow color pallet.
> pie(x, labels, main = "City pie chart", col = rainbow(length(x)))
>
> # Save the file.
> dev.off()
```

Slice Percentages and Chart Legend




```
> # Create data for the graph.
> x <-  c(21, 62, 10,53)
> labels <-  c("Chennai", "Mumbai", "Delhi", "Bangalore")
>
> piepercent<- round(100*x/sum(x), 1)
>
> # Give the chart file a name.
> png(file = "city_percentage_legends.jpg")
>
> # Plot the chart.
> pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
> legend("topright", c("Chennai", "Mumbai", "Delhi", "Bangalore"), cex = 0.8,
+    fill = rainbow(length(x)))
>
> # Save the file.
> dev.off()
```

3D Pie Chart

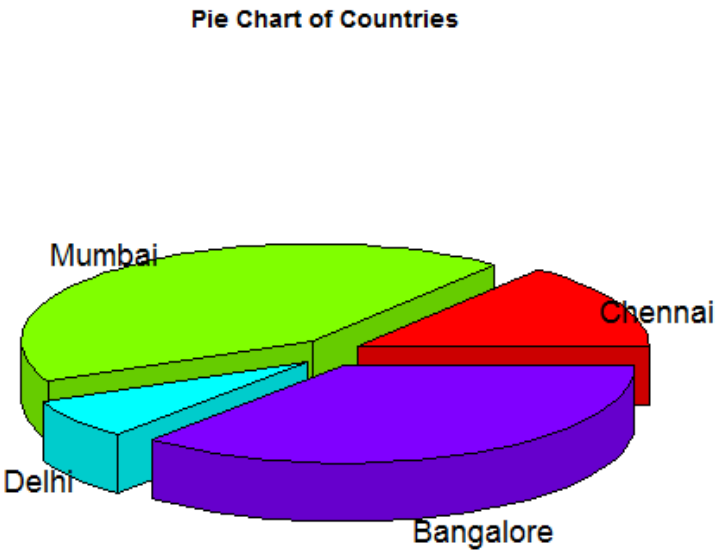
- Need plotrix package for 3D function

```
> install.packages("plotrix")
Installing package into 'C:/Users/User/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cran.asia/bin/windows/contrib/4.0/plotrix_3.7-8.zip'
Content type 'application/zip' length 1134284 bytes (1.1 MB)
downloaded 1.1 MB

package 'plotrix' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\User\AppData\Local\Temp\Rtmp8uxFN8\downloaded_packages
```

3D Pie Chart




```
> # Get the library.
> library(plotrix)
>
> # Create data for the graph.
> x <-  c(21, 62, 10,53)
> lbl <-  c("Chennai", "Mumbai", "Delhi", "Bangalore")
>
> # Give the chart file a name.
> png(file = "3d_pie_chart.jpg")
>
> # Plot the chart.
> pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")
```

Bar chart

- A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable.
- R uses the function **barplot()** to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

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

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

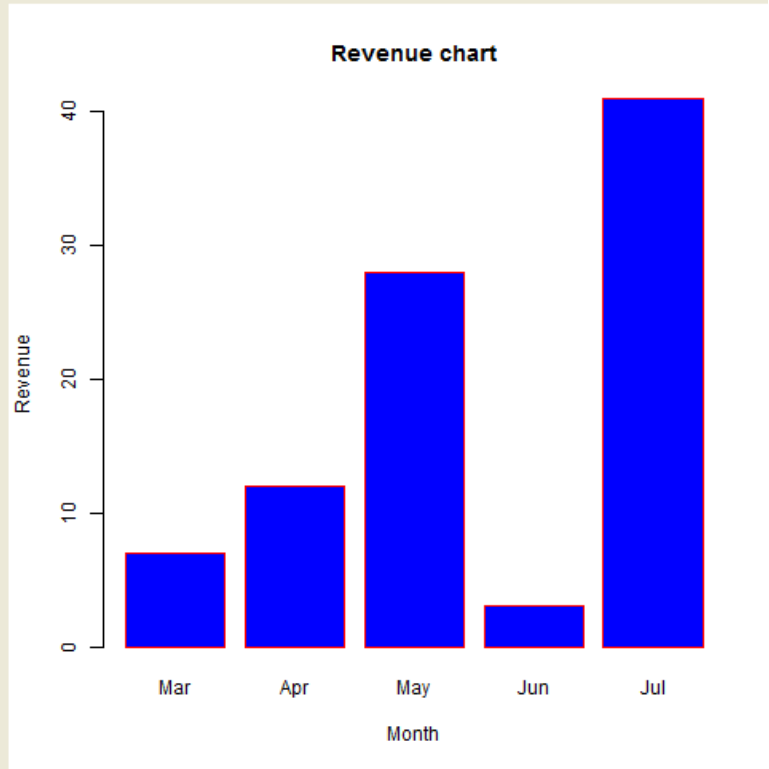
Bar chart

```
> # Create the data for the chart
> H <- c(7,12,28,3,41)
>
> # Give the chart file a name
> png(file = "barchart.png")
>
> # Plot the bar chart
> barplot(H)
>
> # Save the file
> dev.off()
```

```
> # Create the data for the chart
> H <- c(7,12,28,3,41)
>
>
> # Plot the bar chart
> barplot(H)
```

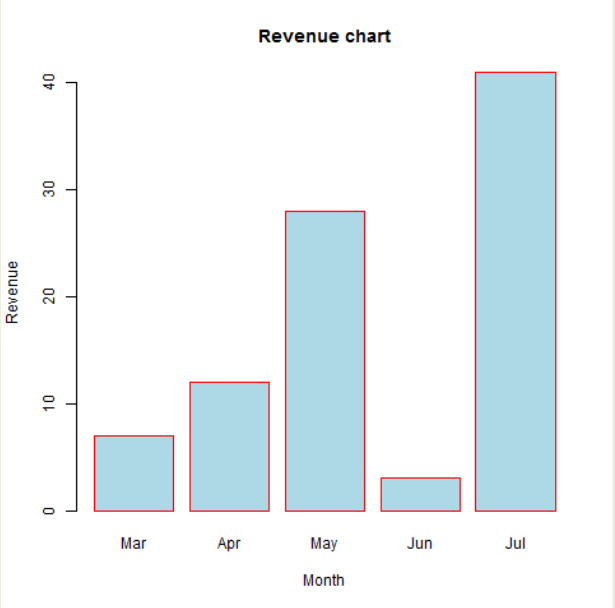
Bar chart

```
> # Create the data for the chart
> H <- c(7,12,28,3,41)
> M <- c("Mar","Apr","May","Jun","Jul")
>
> # Give the chart file a name
> png(file = "barchart_months_revenue.png")
>
> # Plot the bar chart
> barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",
+         main="Revenue chart",border="red")
>
> # Save the file
> dev.off()
```



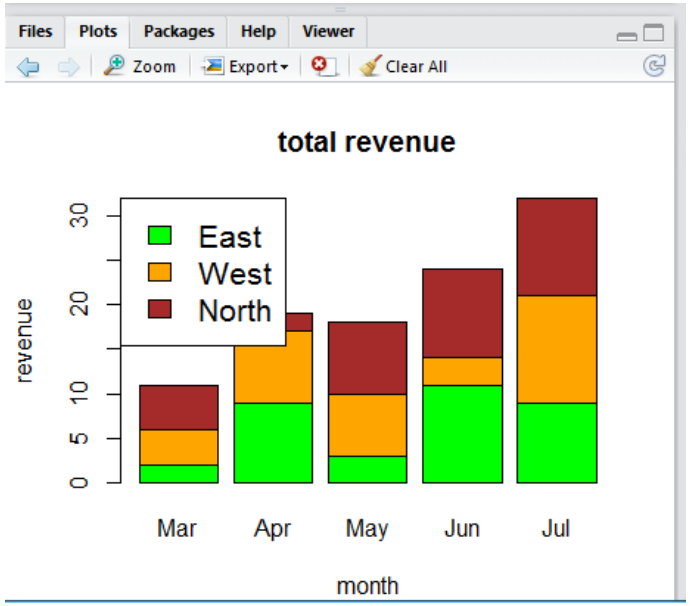
Bar chart

```
Console ~/
> # Create the data for the chart
> H <- c(7,12,28,3,41)
> M <- c("Mar","Apr","May","Jun","Jul")
>
> # Give the chart file a name
> png(file = "barchart_months_revenue.png")
>
> # Plot the bar chart
> barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="light blue
",
+         main="Revenue chart",border="red")
>
> # Save the file
> dev.off()
```



Group Bar Chart and Stacked Bar Chart

```
> # Create the input vectors.
> colors = c("green","orange","brown")
> months <- c("Mar","Apr","May","Jun","Jul")
> regions <- c("East","West","North")
>
> # Create the matrix of the values.
> Values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11), nrow = 3,
ncol = 5, byrow = TRUE)
>
> # Create the bar chart
> barplot(Values, main = "total revenue", names.arg = months, xlab
= "month", ylab = "revenue", col = colors)
>
> # Add the legend to the chart
> legend("topleft", regions, cex = 1.3, fill = colors)
```



We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

R - Boxplots

- Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles.
- This graph represents the minimum, maximum, median, first quartile and third quartile in the data set.
- It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

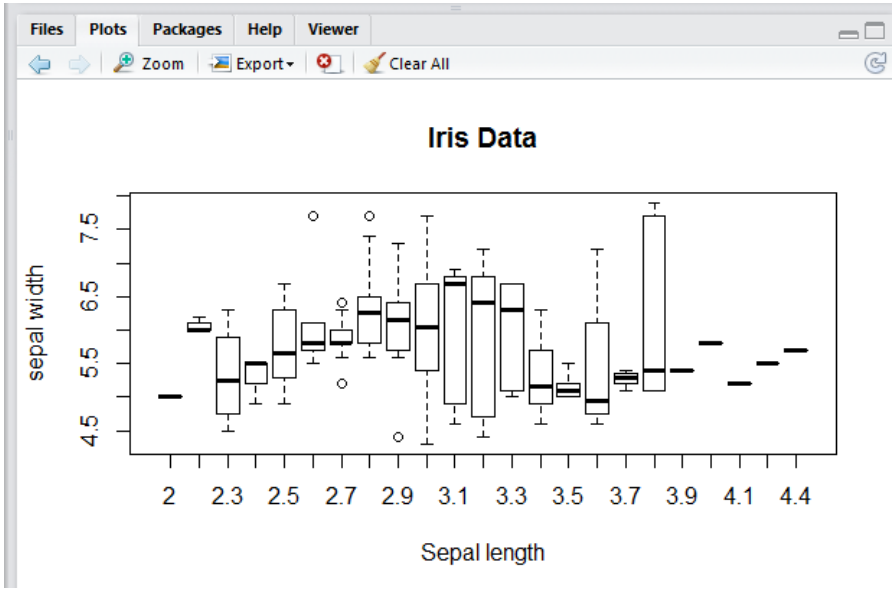
```
boxplot(x, data, notch, varwidth, names, main)
```

- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Data sets

```
> input<-iris[]
> print(head(input))
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

```
> # Plot the chart.
> boxplot(Sepal.Length~Sepal.Width, data = iris, xlab = "Sepal length",
+         ylab = "sepal width", main = "Iris Data")
```



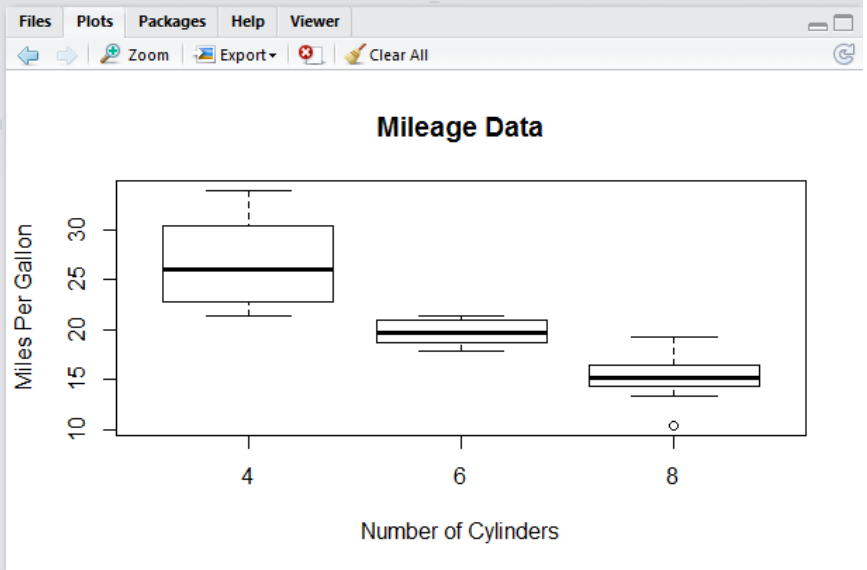
Data sets

```
> i<-mtcars[]
> print(i)
                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4          21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag      21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive     21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant            18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360         14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
Merc 280           19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
Merc 280C          17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
Merc 450SE         16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
Merc 450SL         17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
Merc 450SLC        15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4  8 460.0 215 3.00 5.424 17.82  0  0    3    4
Chrysler Imperial  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic        30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla     33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
Dodge Challenger   15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
AMC Javelin        15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
Camaro Z28         13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
Pontiac Firebird   19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
Fiat X1-9          27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2      26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
Lotus Europa       30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Ford Pantera L     15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
Ferrari Dino       19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
Maserati Bora      15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Boxplots

```
> input <- mtcars[,c('mpg','cyl')]
> print(head(input))
                   mpg cyl
Mazda RX4         21.0   6
Mazda RX4 Wag     21.0   6
Datsun 710        22.8   4
Hornet 4 Drive    21.4   6
Hornet Sportabout 18.7   8
Valiant           18.1   6
```

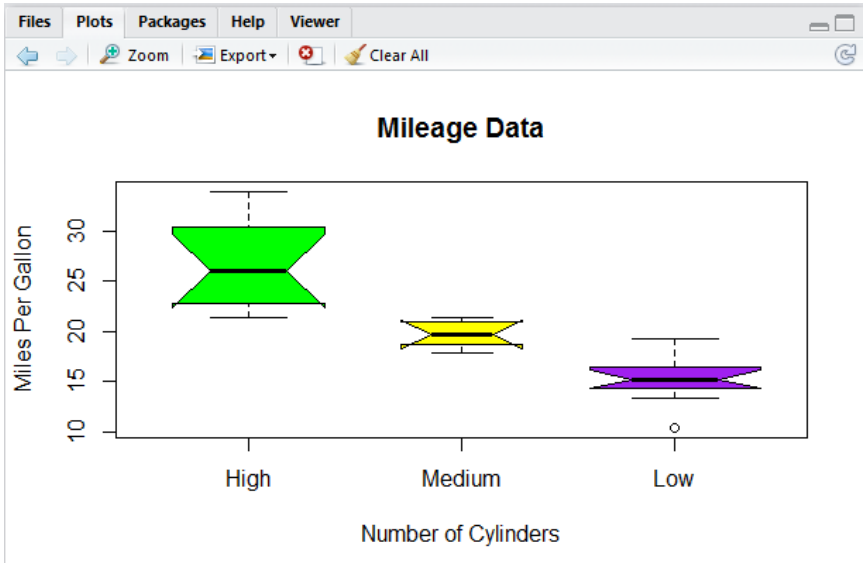
```
> # Plot the chart.
> boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
+         ylab = "Miles Per Gallon", main = "Mileage Data")
> # Plot the chart.
```



Boxplot with Notch

- We can draw boxplot with notch to find out how the medians of different data groups match with each other.
- The below script will create a boxplot graph with notch for each of the data group.

```
> # Plot the chart.
> boxplot(mpg ~ cyl, data = mtcars,
+         xlab = "Number of Cylinders",
+         ylab = "Miles Per Gallon",
+         main = "Mileage Data",
+         notch = TRUE,
+         varwidth = TRUE,
+         col = c("green","yellow","purple"),
+         names = c("High","Medium","Low")
+ )
```



R - Histograms

- A histogram represents the frequencies of values of a variable bucketed into ranges.
- Histogram is similar to bar chat but the difference is it groups the values into continuous ranges.
- Each bar in histogram represents the height of the number of values present in that range.
- R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.
- A simple histogram is created using input vector, label, col and border parameters.

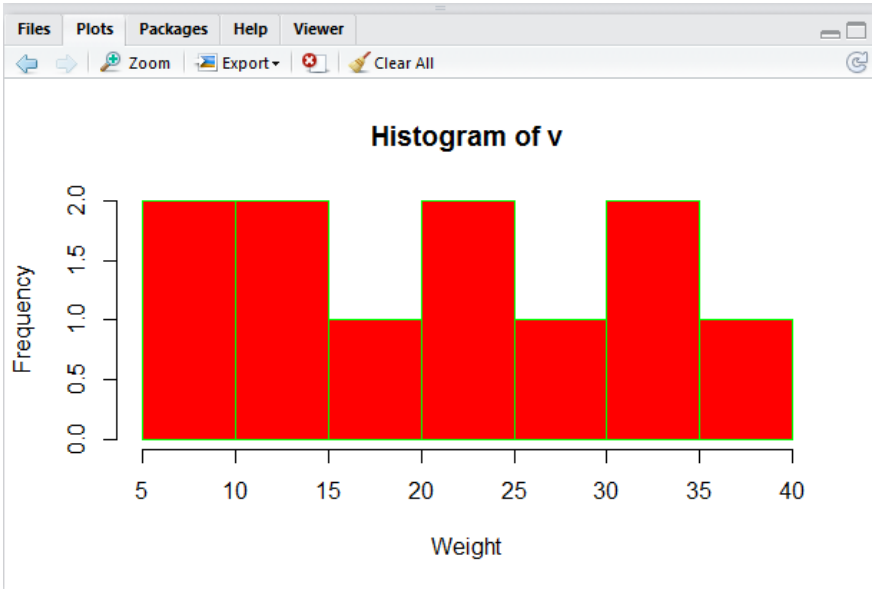
R - Histograms

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

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

R - Histograms

```
> # Create data for the graph.
> v <-  c(9,13,21,8,36,22,12,28,31,33,19)
>
> # Create the histogram.
> hist(v,xlab = "Weight",col = "red",border = "green")
```



R - Line Graphs

- A line chart is a graph that connects a series of points by drawing line segments between them.
- These points are ordered in one of their coordinate (usually the x-coordinate) value.
- Line charts are usually used in identifying the trends in data.
- The **plot()** function in R is used to create the line graph.

R - Line Graphs

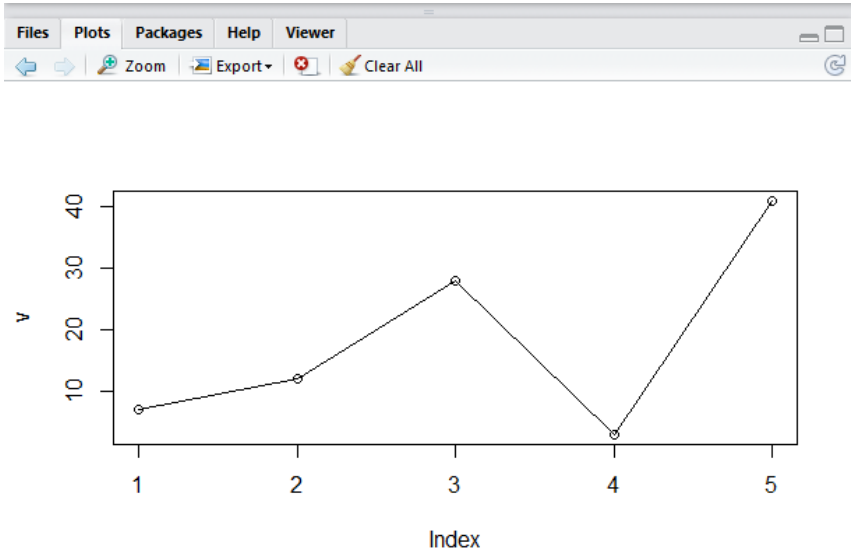
```
plot(v,type,col,xlab,ylab)
```

- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

R - Line Graphs

- A simple line chart is created using the input vector and the type parameter as "O".

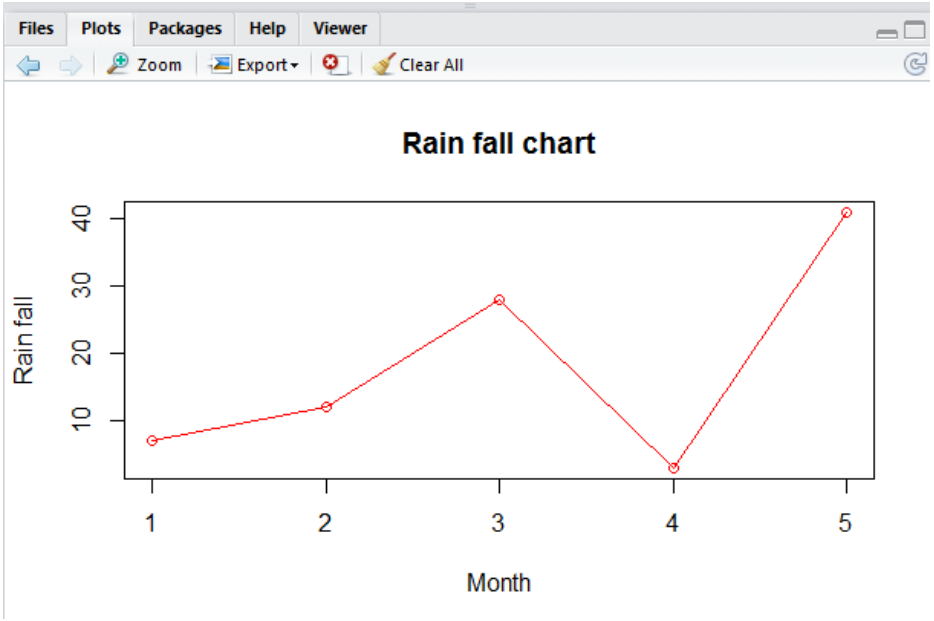
```
> # Create the data for the chart.
> v <- c(7,12,28,3,41)
> # Plot the bar chart.
> plot(v,type = "o")
```



Line Chart Title, Color and Labels

- The features of the line chart can be expanded by using additional parameters.

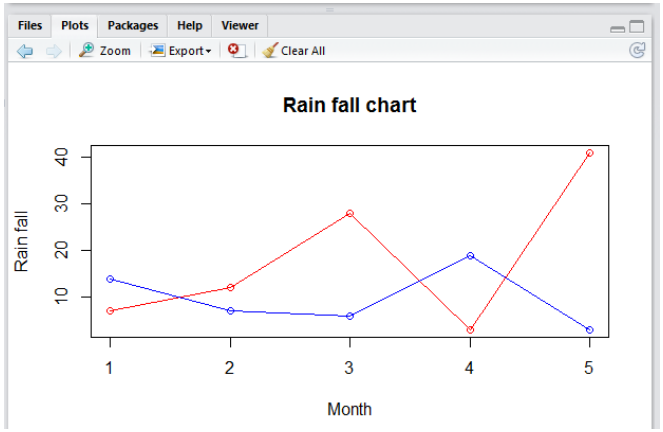
```
> # Plot the bar chart.
> plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",
+      main = "Rain fall chart")
```



Multiple Lines in a Line Chart

- More than one line can be drawn on the same chart by using the **lines()**function.

```
> # Create the data for the chart.
> v <- c(7,12,28,3,41)
> t <- c(14,7,6,19,3)
>
> # Give the chart file a name.
> png(file = "line_chart_2_lines.jpg")
>
> # Plot the bar chart.
> plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",
+      main = "Rain fall chart")
>
> lines(t, type = "o", col = "blue")
>
> # Save the file.
> dev.off()
```



R - Scatterplots

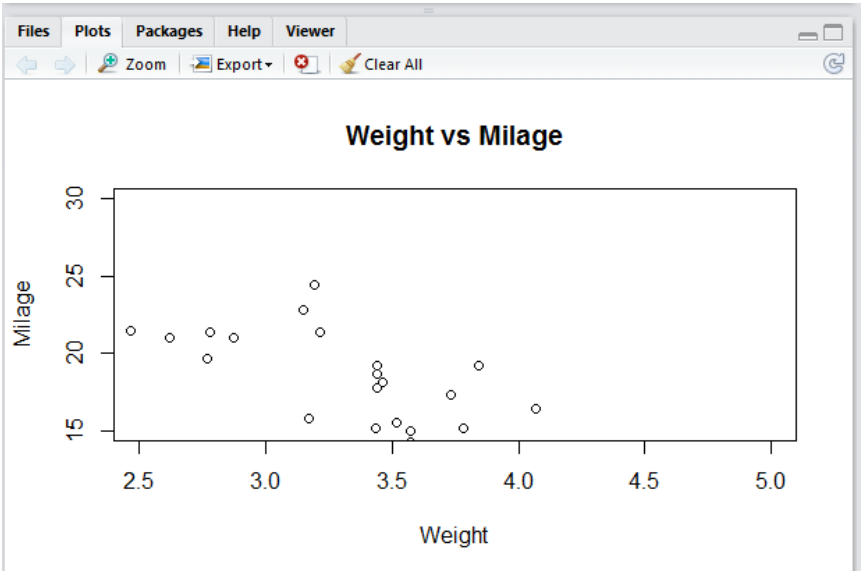
- Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables.
- One variable is chosen in the horizontal axis and another in the vertical axis.

```
plot(x, y, main, xlab, ylab, xlim, ylim, axes)
```

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

R - Scatterplots

```
> # Get the input values.
> input <- mtcars[,c('wt','mpg')]
>
> # Give the chart file a name.
> png(file = "scatterplot.png")
>
> # Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
> plot(x = input$wt,y = input$mpg,
+      xlab = "Weight",
+      ylab = "Milage",
+      xlim = c(2.5,5),
+      ylim = c(15,30),
+      main = "Weight vs Milage"
+ )
>
> # Save the file.
> dev.off()
```



Scatterplot Matrices

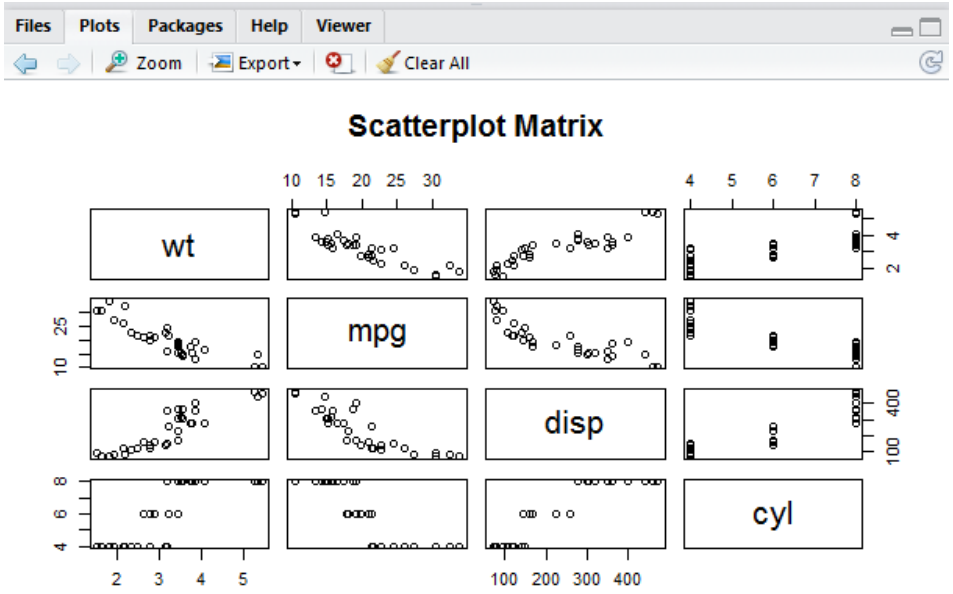
- When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix.
- We use **pairs()** function to create matrices of scatterplots.

```
pairs(formula, data)
```

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

Scatterplot Matrices

```
> # Give the chart file a name.
> png(file = "scatterplot_matrices.png")
>
> # Plot the matrices between 4 variables giving 12 plots.
>
> # One variable with 3 others and total 4 variables.
>
> pairs(~wt+mpg+disp+cyl,data = mtcars,
+       main = "Scatterplot Matrix")
>
> # Save the file.
> dev.off()
```



Distributions in the stats package

- The full list of standard distributions available can be seen using ?distribution.
- Density, cumulative distribution function, quantile function and random variate generation for many standard probability distributions are available in the stats package.

The functions are named in the form

- density/mass function- dxxx
- cumulative distribution function- pxxx
- quantile function qxxx
- random variate generation – rxxx.

Random number generation

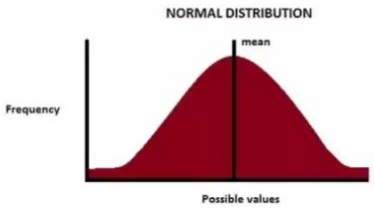
- R has functions to generate a random number from many standard distribution like uniform distribution, binomial distribution, normal distribution etc.
- Functions that generate random deviates start with the letter r.
- For example, runif() generates random numbers from a uniform distribution and rnorm() generates from a normal distribution.

Uniform Distribution

- A **uniform distribution** is one in which all values are equally likely within a range (and impossible beyond that range).


```
> runif(1)     # generates 1 random number
[1] 0.8628323
> runif(4)     # generates 4 random number
[1] 0.1116042 0.3979250 0.7474841 0.3881345
> runif(3, min=5, max=10)     # define the range between 5 and 10
[1] 6.546255 6.566184 5.151348
```

Normal Distribution



- Normal Distribution is a specific distribution that is bell shaped. Its exact nature is defined by its mean and standard deviation.
- We need to specify the number of samples to be generated.
- We can also specify the mean and standard deviation of the distribution.
- If not provided, the distribution defaults to 0 mean and 1 standard deviation.

```
> rnorm(1)     # generates 1 random number
[1] -0.320998
> rnorm(3)     # generates 3 random number
[1]  1.0704188  0.4302561 -0.8626655
> rnorm(3, mean=10, sd=2)     # provide our own mean and standard deviation
[1] 10.07420 10.85347 11.93559
```

Normal Distribution

normaldistrtable.R

Source on Save Run Source

```r
n = floor(rnorm(50, 5, 60))
print('List of random numbers in normal distribution:')
print(n)
t = table(n)
print("Count occurrences of each value:")
print(t)
```

Console Terminal

```
>
> source('~/.active-rstudio-document')
[1] "List of random numbers in normal distribution:"
 [1]   75  -34   62   51   69    2    6  -36  -22   35   47  164   54  -37   86   -7
[17]   43   49   48   15  -45 -100 -113   58  -45  102  124  -97   -7   94  -13   66
[33]  -81   41    5   18   46  -36  -39   19   97   17   90   64   34    5   52  -40
[49]  -12  -13
[1] "Count occurrences of each value:"
n
-113 -100  -97  -81  -45  -40  -39  -37  -36  -34  -22  -13  -12   -7    2    5
   1    1    1    1    2    1    1    1    2    1    1    2    1    2    1    2
   6   15   17   18   19   34   35   41   43   46   47   48   49   51   52   54
   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
  58   62   64   66   69   75   86   90   94   97  102  124  164
   1    1    1    1    1    1    1    1    1    1    1    1    1
>
```

Normal Distribution- Barplot

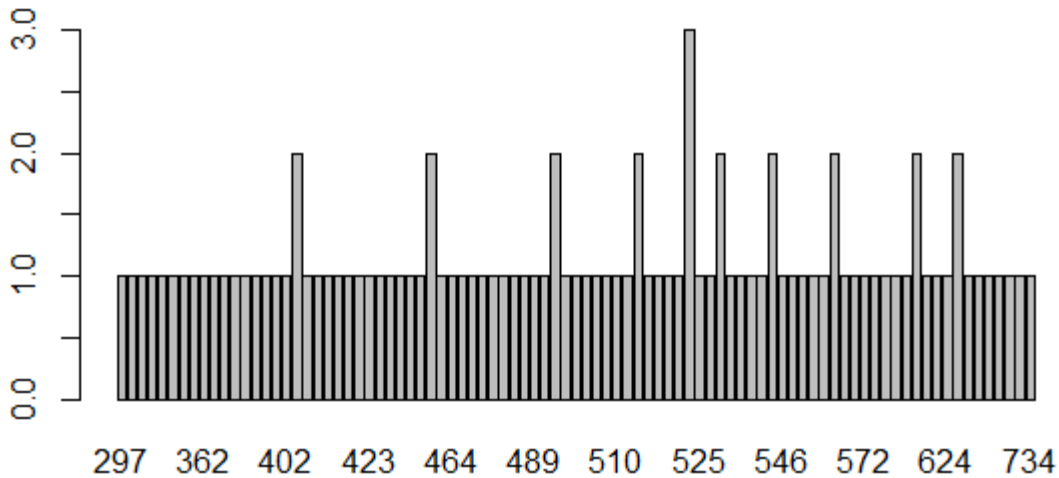
normaldistrtable.R | Untitled1*

Source on Save | Run | Source

```
n = floor(rnorm(100, 500, 100))
t = table(n)
barplot(t)
print(t)
```

Files | Plots | Packages | Help | Viewer

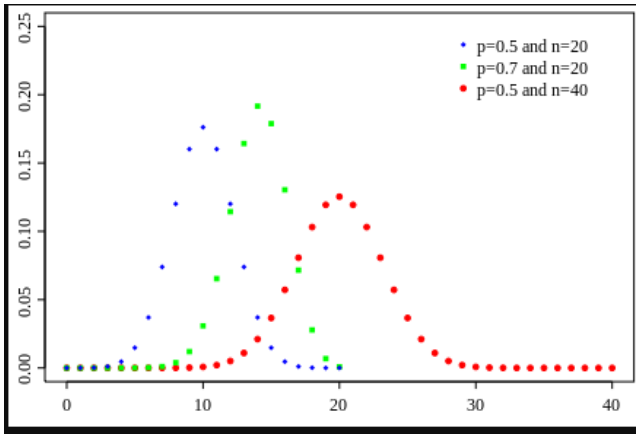
Zoom | Export | Publish



Binomial distribution

- The **binomial** is a type of **distribution** that has two possible outcomes (the prefix “bi” means two, or twice).
- A **binomial distribution** can be thought of as simply the probability of a SUCCESS or FAILURE outcome in an experiment or survey that is repeated multiple times.

```
dbinom(x, size, prob)
pbinom(x, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)
```



x is a vector of numbers.
p is a vector of probabilities.
n is number of observations.
size is the number of trials.
prob is the probability of success of each trial.

Binomial distribution

normaldistrtable.R | marksbarplot.R* | Untitled1*

Source on Save | Run | Source

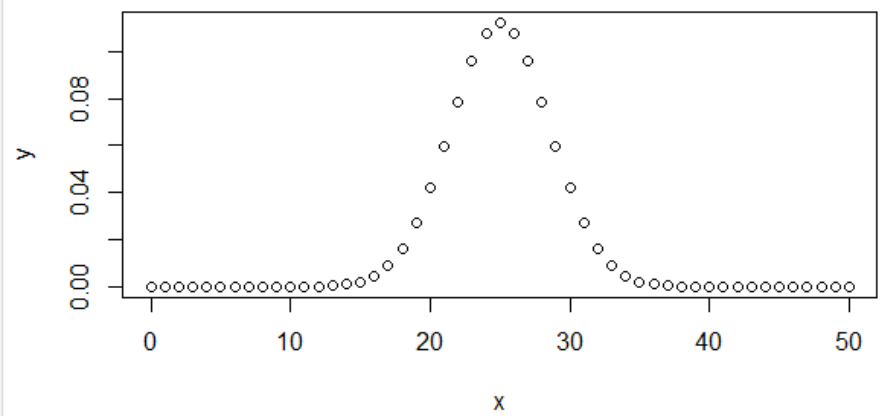
```r
# Create a sample of 50 numbers which are incremented by 1.
x <- seq(0,50,by = 1)

# Create the binomial distribution.
y <- dbinom(x,50,0.5)


# Plot the graph for this sample.
plot(x,y)
```

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish



R - Linear Regression

- Regression analysis is a very widely used statistical tool to establish a relationship model between two variables.
- One of these variable is called predictor variable whose value is gathered through experiments.
- The other variable is called response variable whose value is derived from the predictor variable.

R - Linear Regression

- The general mathematical equation for a linear regression is

$$y = ax + b$$

- **y** is the response variable.
- **x** is the predictor variable.
- **a** and **b** are constants which are called the coefficients.
- In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

Steps to Establish a Regression

- A simple example of regression is predicting weight of a person when his height is known. To do this we need to have the relationship between height and weight of a person.

The steps to create the relationship is –

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called **residuals**.
- To predict the weight of new persons, use the **predict()** function in R

lm() Function

- This function creates the relationship model between the predictor and the response variable.

```
lm(formula,data)
```

- **formula** is a symbol presenting the relation between x and y.
- **data** is the vector on which the formula will be applied.

```
> # Apply the lm() function.
> relation <- lm(y~x)
>
> print(relation)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -38.4551       0.6746
```

Get the Summary of the Relationship

```
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
>
> # Apply the lm() function.
> relation <- lm(y~x)
>
> print(summary(relation))

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3002 -1.6629  0.0412  1.8944  3.9775

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06
```

predict() Function

- The basic syntax for predict() in linear regression is

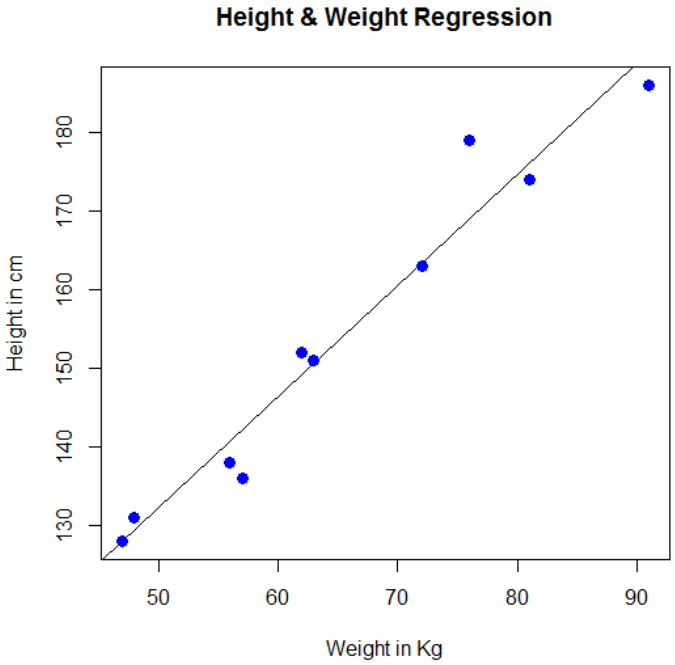
```
predict(object, newdata)
```

- **object** is the formula which is already created using the lm() function.
- **newdata** is the vector containing the new value for predictor variable.

```
> # The predictor vector.
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
>
> # The resposne vector.
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
>
> # Apply the lm() function.
> relation <- lm(y~x)
>
> # Find weight of a person with height 170.
> a <- data.frame(x = 170)
> result <-  predict(relation,a)
> print(result)
       1
76.22869
```

Visualize the Regression Graphically

```
> # Create the predictor and response variable.
> x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
> y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
> relation <- lm(y~x)
>
> # Give the chart file a name.
> #png(file = "linearregression.png")
>
> # Plot the chart.
> plot(y,x,col = "blue",main = "Height & Weight Regression",
+ abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm$
>
> # Save the file.
> #dev.off()
```



Summary

Day1

- Overview, installation, Basics
- Variables, R- Data type
- Operators

Day2

- Control structures
- Functions
- String

Day3

- Subsetting R objects, File connection
- R- charts / graphs, Statistical functions

Thank you