BASIC FUNCTIONS
&
DATA VISUALISATION TOOLS
IN R- LANGUAGE

Prepared By
Lijin S Reji
Student
Beach Theory

Under the Guidance of
Dr. Jacob Thomas Koickal
CEO
Beach Theory

1. The function help.start() opens a browser interface to help information, manuals, and helpful links. It may take practice, and time, to learn to navigate the wealth of information that is on offer.
2. The function RSiteSearch() initiates (assuming a live internet connection) a search of R manuals and help pages, and of the R-help mailing list archives, for key words or phrases. The argument restrict allows some limited targeting of the search. See help(RSiteSearch) for details
3. # - used to make comments
4. ctrl+L clears the console
5. ctrl + Enter – to excecute a command in top left box
6. ctrl + D – to quit

VARIABLES , FACTORS, LEVELS

All column in a data set are variables

some variables are numeric while some others are factors.

A factor is stored internally as a numeric vector with values 1, 2, 3, . . . , k. The value k is the number of levels. The levels are character strings.

```
> str(rainforest)
'data.frame':   65 obs. of  7 variables:
 $ dbh    : num  6 23 20 23 24 5 5 8 10 8 ...
 $ wood   : num  NA 353 208 445 590 14 10 31 59 30 ...
 $ bark   : num  NA NA NA NA NA NA NA NA NA NA ...
 $ root   : num  6 135 NA NA NA 2 NA NA NA 6 ...
 $ rootsk : num  0.3 13 NA NA NA 2.4 NA NA NA 1 ...
 $ branch : num  NA 35 41 50 NA NA NA NA NA 4 ...
 $ species: Factor w/ 4 levels "Acacia mabellae",..: 1 1 1 1 1 1 1 1 1 1 ...

> x <- c(rep("male", 100), rep("female", 102))
> levels(x)
NULL                 ### character vector has no levels
> x <- factor(x)
> levels(x)
[1] "female" "male"
> x <- factor(x, levels = c("male", "female"))   ## to change order of factors
> levels(x)
[1] "male"   "female"
```

RANDOM NORMAL DISTRIBUTION

7. rnorm(100,10,3)- 100 numbers generated in random distribution with mean 10 and sd 3. [rnorm(100)- default mean 0 ,sd 1

8. TO SEE WHICH ALL PACKAGES ARE ATTACHED AND BASIC INFO
   sessionInfo()

```
> library(xlsx)
Loading required package: rJava
Loading required package: xlsxjars

> sessionInfo()
R version 3.2.3 (2015-12-10)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
 [1] LC_CTYPE=en_IN.UTF-8          LC_NUMERIC=C                  LC_TIME=en_IN.UTF-8
 [4] LC_COLLATE=en_IN.UTF-8        LC_MONETARY=en_IN.UTF-8       LC_MESSAGES=en_IN.UTF-8
 [7] LC_PAPER=en_IN.UTF-8          LC_NAME=en_IN.UTF-8           LC_ADDRESS=en_IN.UTF-8
[10] LC_TELEPHONE=en_IN.UTF-8      LC_MEASUREMENT=en_IN.UTF-8
LC_IDENTIFICATION=en_IN.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] xlsx_0.5.7     xlsxjars_0.6.1 rJava_0.9-8

loaded via a namespace (and not attached):
[1] tools_3.2.3
```

TO REPEAT rep()

```
> rep(c(1,2,3), 4)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
> rep(c(1,2,3), 4)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

TAKING SUBSET FROM A DATA FRAME BAESD ON A VARIABLE OF FACTOR TYPE WITH DIFFERENT LEVELS

```
install.packages("DAAG")
library(DAAG)
> head(rainforest)
   dbh wood bark root rootsk branch         species
27   6   NA   NA    6    0.3     NA Acacia mabellae
61  23  353   NA  135   13.0     35 Acacia mabellae
62  20  208   NA   NA     NA     41 Acacia mabellae
63  23  445   NA   NA     NA     50 Acacia mabellae
65  24  590   NA   NA     NA     NA Acacia mabellae
80   5   14   NA    2    2.4     NA Acacia mabellae
```

```
nbranch <- subset(rainforest, species == "Acacia mabellae")$branch
> nbranch
 [1] NA 35 41 50 NA NA NA NA NA  4 30 13 10 17 46 92
```

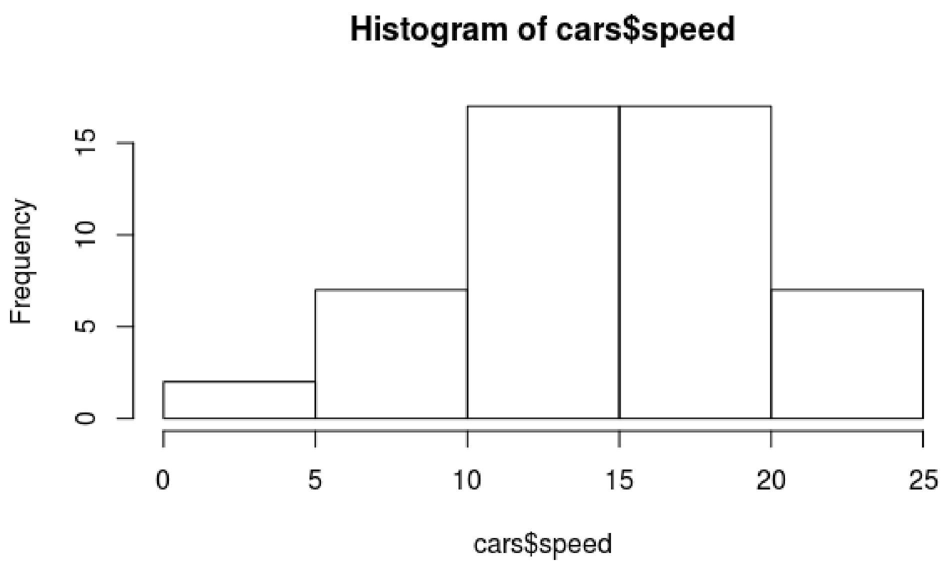
DIRECTORY SETTINGS

9. getwd ()– to know the current working directory
10. setwd(“~/path”) - to change directory while using r in ubuntu
11. list.files() - will list files in current drectory

HISTOGRAM

12. hist()- plots histogram, frequency distribution of individual objects, each vertical bars are called bins, by default – 9 bins,
13. hist(name, breaks = 15, main = “title”, col = “colours”, xlab = “x axis heading”) – histogram with 15 bins

```
> hist(cars$speed)
```



SAVING AS PDF

14. pdf(“name.pdf”)- convert the graphical to pdf but does not save it
15. dev.off() - close graphical ouyput file and saves it in current directory

NEW R SCRIPT

16. ctrl+shift+n- new R script
17. use top left box to make and save R scripts
18. source(“name.R”) - will call the .R script and excecute

SEARCHING FILES & PACKAGES

19. file.exists(" name with extension") - check if the file exists in current directory
20. ls() or objects() - will show the vectors and data frames in the workspace/enveronment only, no files will be shown.
21. search() - will search for packages which are right now loaded

22. FINDING FUNCTION- help.search()

```
> help.search("sort")  ## list functions with 'sort' as a title or as an alias in the
                          help page
```

MAKING A VECTOR

23. x <- c(1 ,2 , 4) - creatng a vector x using combine function
24. q<-c(x,x,4);q or (q<-c(x,x,4)) – creating q and then printing q in one step

```
> (dd <- c(1,0,8,0,10)) ## equivalent to assignmet & print(dd)
[1]  1  0  8  0 10
```

PRINTING

25. x or print (x) – both are same

CALLING ELEMENTS

26. x[3] – using unique operator [] to call third element of vector x
27. x[2:4] – call elements from second position to fourth

MEAN

28. mean(x) – find mean- numerical measure of central location of data

```
> nbranch
 [1] NA 35 41 50 NA NA NA NA NA  4 30 13 10 17 46 92
> mean(nbranch) ## when NA is not ommitted
[1] NA
> mean(nbranch, na.rm = T)   ## when NA is ommitted
[1] 33.8
```

STANDARD DEVIATION

29. sd(x) – standard deviation of x

PRE – INSTALLED DATA SETS IN R

30. data() - to know the data sets allready in R

    To see what other data sets come bundled with R, we can use the data() command to obtain a list of data sets along with a short description of each. If we modify the data from a data set, we can reload it by providing the name of the data set in question as an input parameter to the data() command, for example, data(iris) reloads the iris data set.

31. ? or help() – used to get information

32. attach()- to call coloumns of a data frame by direct name

```
> attach(Cars93.summary)
> abbrev
[1] C  L  M  Sm Sp V
Levels: C L M Sm Sp V
> Min.passengers
[1] 4 6 4 4 2 7
> detach(Cars93.summary)
```

33. Nile – prints data in Nile
34. length () - prints length
35. mode () - print data type

PASTE & STRING SPLITTING

36. paste("li","j") - outputs "li j"- used to combine words
37. strsplit(u,"space")- split character with spaces in between eg "li" "j"

SAVING A FILE

38. save (a, file = "a.csv")

READING CSV

39. data1<-read.csv(file.choose() , header = T) OR WE CAN USE
40. data<- read.csv(file = "~/famili.csv", header = T)

READING ANY FILE USING READ.TABLE

41. `data2 <- read.table(file = "~/family.csv", header = TRUE , sep = ",")` "," is for .csv and "\t" is for .txt or tsv

REMOVING DATA rm()

42. rm(data1) - removes data

TO READ TAB DELIMITED FILE TXT

43. read.delim() - to read txt file

LOADING A DATA SET eg: IRIS

data(iris)

OPERATIONS IN DATA SETS

44. dim(data) – dimensions – rows columns

45. head(iris)

```
> head(iris, n=3)
  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
```

Collectively, the sepal length, sepal width,
petal length, and petal width are referred to as **features, attributes, predictors, dimensions, or independent variables** in literature

Similarly, the species column in the data frame is what we are trying to predict with our model, and so it is referred to as the **dependent variable, output, or target**

Each row in the data frame corresponding
to a single data point is referred to as an **observation,** though it typically involves
observing the values of a number of features.

46. tail(iris)
47. iris[c(3 , 4 , 7),] -read rows 3,4,7- without coma space – 3,4,7 coloumns are taken default
48. iris[3:5,] -
49. iris[-(3:148),]
50. names(iris)- coloumn names

VIEW DATA

51. view(data1)
52. importing file directly from console is same that as the combination of – read.XXX functio and view function

STRUCTURE OF DATA

53. str(data1) gives number of observations for number of variables= 1 observation of 4 variables- family

SUMMARY

54. summary(iris)

```
> summary(cars)
     speed           dist       
 Min.   : 4.0   Min.   :  2.00  
 1st Qu.:12.0   1st Qu.: 26.00  
 Median :15.0   Median : 36.00  
 Mean   :15.4   Mean   : 42.98  
 3rd Qu.:19.0   3rd Qu.: 56.00  
 Max.   :25.0   Max.   :120.00  
```

FIX DATA- EDIT COLOUMN TITLE

55. fix(data1) – edit the data frame

INSTALLING PACKAGES

56. sudo apt-get install r-cran-xml - to install XMLpackages if normal installing of packages fails via R consloe fails – type this in Terminal
57. for installing Rcurl install `sudo apt-get install libcurl4-openssl-dev` in Terminal then try – install.packages(“RCurl”) in R
58. if Rcurl does not work for installing httr package initially type -sudo apt-get install libssl-dev- in terminal. The type- install.package (“httr”) -in R console

SIMPLE WEBSCRAPING (FIRSTPROGRAM.R)

59. webscrapping for websites that are not full HTML -

```
library(XML)
library(httr)
url<-"http://en.wikipedia.org/wiki/Brazil_national_football_t
eam"
tabs <- GET(url)
tabs <- readHTMLTable(rawToChar(tabs$content),
stringAsFactors = F)
```

60. reading XML files

```
library(XML)
library(methods)
data3<- xmlToDataFrame("~/simple.xml")
```

61. reading HTML data from websites which are pure HTML eg:

```
library(XML)
lijin <- readHTMLTAble (“http://www.w3schools.com/colors/color_tryit.asp?hex=000000” ,
which = 1)
```

62. to read excel files (.xlsx)-

sudo apt-get install r-can-rjava - install rjava packagevia terminal
install.packages(“xlsx”) - install package in R
library(xlsx)
data<-read.xlsx(“~/family.xlsx”, sheetIndex = 1)

63. reading data from a url where there is only data:

```
sampleurl <- "https://archive.ics.uci.edu/ml/machine-learning
databases/iris/iris.data"
> setwd("~/datascience")
> local<- file.path("data", "demo.data")
> download.file(smapleurl, local)
> data3<- read.table("~/datascience/demo.data", header = T, sep = ",")
> View(data3)
```

64. **variance-** shows how the values are diferrent from the mean value for one factor or column larger value of variance ,larger seperation between the values and the values are widely seperated in that way- `var(iris$sepallength)`

65. **covariance** between two factors- the covariance shows how the different factors are inter related, anegative covariance value will indicate an inverse relation while a positive value will indicate a direct prportional relation.- `cov(x,y)`

66. **corelation coefficient** is the measure of how the data is linearly related, if the value of cor coefficient is close to one the values are positively linearly related- `cor(x,y)`

67. **apropos(“med”)**- to search if we dont know the exact name of the function

```
> apropos("sort")
[1] "is.unsorted"         ".rs.sortCompletions" "sort"                "sort.default"
[5] "sortedXyData"        "sort.int"            "sort.list"           "sort.POSIXlt"
```

68. **class(x)**- print the class of x that is the data type of x

69. **is.integer(x)**- is x an intiger- returns TRUE or FALSE

70. **y <-as.integer (3)** - to assign y integer data type , also can be used for coercing decimal values ie as.integer(3.12) gives a value of only 3 as output.

71. **as.complex(1)** - 1+0i

TO COMBINE SENTENCES

72. `sprintf("%s has %d rupees", "lijn", 10000)`
[1] "lijn has 10000 rupees"
sprintf funtion can be used to write sentence using place holders as in C

TO DERIVE SENTENCES

73. substr("lijin has 100 rupees", start = 2, stop = 10)
[1] "ijin has "
substr () can be used to extract a group of words from a full sentence

TO SUBSTITUTE SOME WORDS FROM A SENTENCE

74. sub("little", "BIG","mary has a little lamb")
[1] "mary has a BIG lamb"
sub() function can be used to replace one string with another string

MATRIX CREATION

75. creating a matrix

```
A<- matrix(data = c(1,2,3,4,5,6), nrow = 3, ncol = 2, byrow = T)
> A
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

76. for giving names to rows and coloumns

```
dimnames(A)<- list(
+ c("row1","row2","row3"),
+ c("co1","col2"))
> A
     co1 col2
row1   1    2
row2   3    4
row3   5    6
```

77. To take the **trnspose of a matrix**, use t() function:

```
>A
     co1 col2
row1   1    2
row2   3    4
row3   5    6
> t(A)
     row1 row2 row3
co1     1    3    5
col2    2    4    6
```

COMBINE ROWS AND COLOUMNS

78. cbind(A,B) - to bind matrix A and B by coloumn, A and B has equal number of rows
79. rbind(A,B) - vice versa of cbind
80. c(A) - will deconstruct matrix A -

```
>A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

DECONSTRUCTION OF A MATRIX

```
> c(A)
[1] 1 4 2 5 3 6
```

81. nrow(mtcars)
82. ncol(mtcars)
83. head(mtcars)

VECTOR SLICE

84. mtcars[[9]] or mtcars[["am"]] or mtcars[,"am"] or mtcars$am ($ sign cannot be used for calling rows)- all calls 9th coloumn VECTOR ONLY NOT SLICE of mtcars data set
85. for VECTOR SLICES – mtcars[1] or mtcars["mg"] or mtcars[c("mg","cc")]

ROW SLICE

86. mtcars[1,] or mtcars["name1",] - will give first row slice – adding coma will be used for calling rows-
87. mtcars[c(1,2),] or mtcars [c("name1","name2"),] can both be used for row slicing more than 1 row

88. ## Logical indexing for calling rows-

```
>L <- mtcars$am ==0  #L logical vector with all 0 true and 1 false in "am" variable
>mtcars[L,]  # calling a data frame with all "am" values = 0.
> mtcars[L,]$mpg # calling a vector of variable "mpg" with "am"  values = 0
```

FREQUENCY DISTRIBUTION

89. table(painters$school) - this can be used to find the frequency distribution of qualitative data only – the data which cannot be measured in numbers- cbind() function can be used to prind the data in coloumn format, for quantitative continous data, we have to convert that to descrete data

```
> head(tinting)
  case id  age sex tint target    it  csoa   agegp
1    1  1 22.4   f   no  hicon 26.00 46.80 younger
2    1  1 22.4   f   lo  hicon 32.24 37.44 younger
3    1  1 22.4   f   hi  hicon 27.04 42.64 younger
4    1  1 22.4   f   no  locon 17.68 41.60 younger
5    1  1 22.4   f   lo  locon 20.80 37.44 younger
6    1  1 22.4   f   hi  locon 26.00 40.56 younger

> table(tinting$sex)

 f  m 
91 91 
> table(Sex=tinting$sex, Agegp= tinting$agegp)
   Agegp
Sex younger older
  f      63    28
  m      28    63
> table(Sex=tinting$sex, Agegp= tinting$agegp, Tint = tinting$tint, Target = tinting$target)
, , Tint = no, Target = locon

   Agegp
Sex younger older
  f       9     4
  m       4     9

, , Tint = lo, Target = locon

   Agegp
Sex younger older
  f       9     4
  m       4     9

, , Tint = hi, Target = locon
```

```
   Agegp
Sex younger older
  f       9     4
  m       4     9

, , Tint = no, Target = hicon

   Agegp
Sex younger older
  f      18     8
  m       8    18

, , Tint = lo, Target = hicon

   Agegp
Sex younger older
  f       9     4
  m       4     9

, , Tint = hi, Target = hicon

   Agegp
Sex younger older
  f       9     4
  m       4     9
```

relative frequency distribution

- percentage of each distributed variables out of total frequency - frquency distribution/ nrow(painters) - the values can be rounded off to 2 digits by – old = options(digits=1)- use cbind()

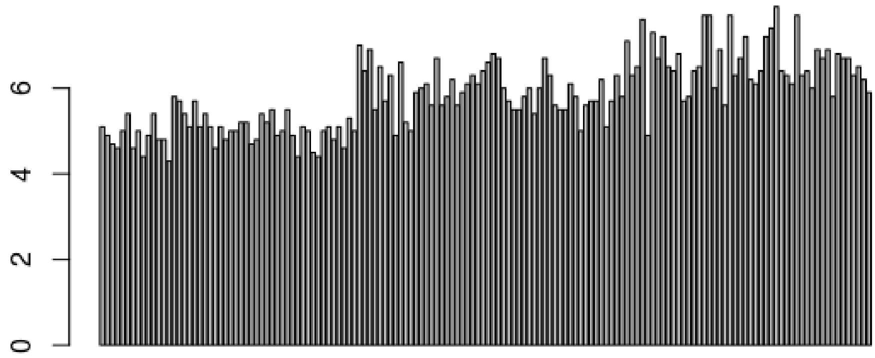
TO ROUND OFF DECIMAL NUMBERS- OLD

old = options(digits=1

BAR AND PIE PLOT

90. barplot(school.freq) – gives the bar plot of frequency distribution colour can be given to each bar by – barploat (school.freq, col= colour)- here colours is another variable- colour<- c("red","blue", "green", "yellow", "brown", "black", "grey", "white")

barplot(sl)   sepel length of iris



91. pie(schoo.freq, col = colour)- for pie chart

QUALITATIVE DATA

92. to classify a data into diffterent data frame based the value of some qualitative data variable- 1) we have to make another variable which contains only the values of this qualitative data variable- c_school <- painters$school 2) we have to make a logical vector whose value are true only when the value of the “some qualitative data variable under study” is available – B_school<- c_school==“C” 3)make another data frame having school value “C” - C_school <- painters [B_school,] 4)find mean by mean(C_school$composition)

TAPPLY

93. tapply(painters$composition, painters$school , mean)- will give mean of composition based on different schools in one step

RANGE

94. range(faithful$eruption)- gives the range

```
> fossil_fuel
  year carbon
1 1800      8
2 1850     54
3 1900    534
4 1950   1630
5 2000   6611

> range(fossil_fuel$carbon)
[1]    8 6611
```

MAKING A SEQUENCE

95. seq(1.5, 5.5, by=0.5)- will give a sequence 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5

QUANTITATIVE DATA

96. to find the frequency distribution of quantitative variable 1) find range- range(). 2) make a sequence based on this range - seq(1.5, 5.5, by = 0.5). 3)cut the quantitative based on this sequence in to varoious intervals- cut(A, B, right = F) 4)use table function on the resultant variable - table(A) 5) to plot - hist(B, right = F)

CUMULATIVE DISTRIBUTION FREQUENCY

97. cumulative frequency - cumsum(table(A))- apply cumsum () function on frequency distribution table
98. cum.eruption <- c(0, cumsum(table(eruption.cut)))
    > plot(cutting, cum.eruption, main = "faithful eruption", xlab = "duration in mnutes", ylab = "cumulative eruption"

PROBABILITY DENSITY FUNCTION

1. > plot(density(eruption))
   density is the probability density function

TO JOIN DOTS IN A PLOT WITH LINES

lines(cutting, cum.eruption)

RELATIVE CUMULATIVE DISTRIBUTION= CUMULATIVE DISTRIBUTION/nrow(dataset)

ecdf function- to make relative cumulative distribution funtion in one step

ecdf(faithful$eruptions)

STEM AND LEAF PLOT

stem(eruption)

SCATTER PLOT

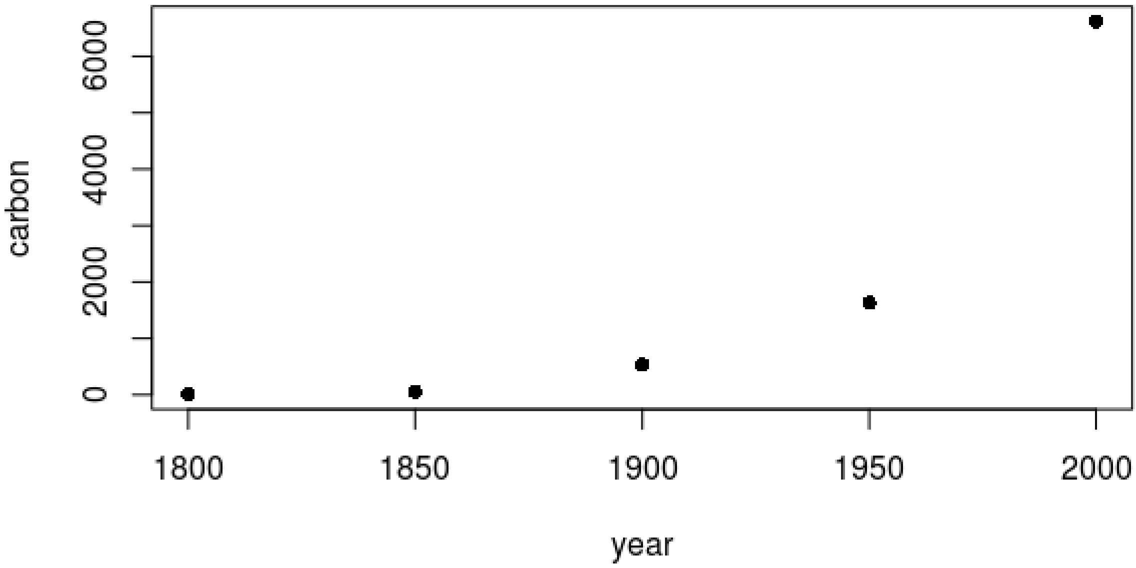
same as that of plot()- it is called so since it is the plot of two random variable

```
carbon <- c(8, 54, 534, 1630, 6611)
year <- c(1800, 1850, 1900, 1950, 2000)
plot(year, carbon, pch = 16) OR plot(carbon ~ year)
```

The construct
Carbon ˜Year
is a graphics formula. The
plot()
function interprets
this formula to mean “Plot
Carbon
as a function of
Year
” or “Plot
Carbon
on the
y
-axis against
Year
on the
x
-axis”.

The setting
pch=16

(where
pch
is “plot character”) gives a solid black dot



LINEAR REGRESSION MODEL

```
lm(waiting ~ eruption)
```

TO DRAW LINE IN SCATTER PLOT based on linear regression model

```
abline(lm(waiting ~ eruption))
```

WITH AND WITHIN

both are used as a substitude of $ function, but mainly done when more than 1 operation has to be done on a data frame.

with(data, plot()) - is used to operations directly on the data frame and we cannot s tore the resultant on the variable as the out put of that variable will be null.
within() is used when we have to store the resultant operations on a variable

In repeated computations with the same data frame, it is tiresome to keep repeating the name of the data frame. The function with() is often helpful in this connection.

```
> with(Cars93.summary, c(mean(Min.passengers), median(Max.passengers)))
[1] 4.5 6.0
```

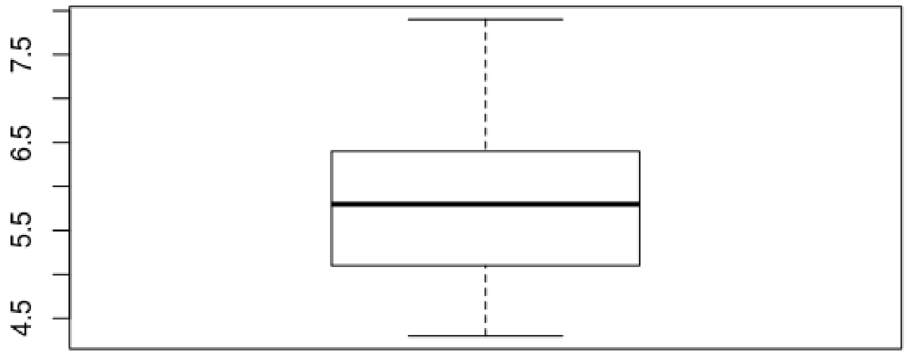
AGGREGATE

The aggregate() function yields a data frame that has the mean or value of another specified function for each combination of factor levels.

```
data("iris")
 head(iris)
iris_av <- with(iris, aggregate(Sepal.Length, by= list(Species), mean))
iris_av

 Group.1     x
1     setosa 5.006
2 versicolor 5.936
3  virginica 6.588
```

BOX PLOT

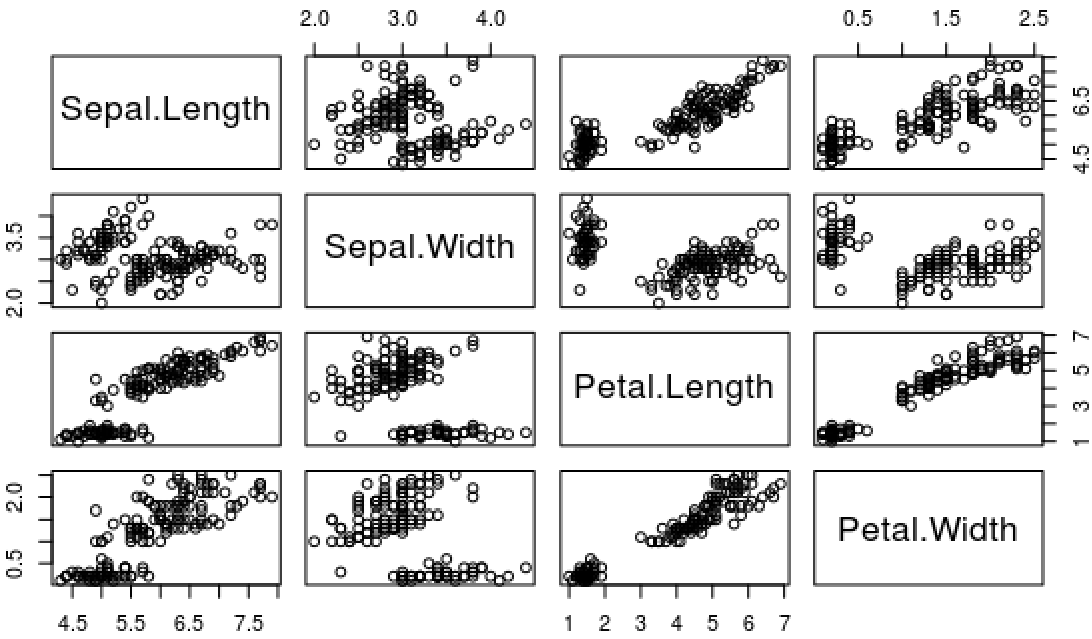
```
boxplot(desity)
```




```
> sl.b<- boxplot(sl)
> summary(sl.b)
       Length Class  Mode
stats 5      -none- numeric
n     1      -none- numeric
conf  2      -none- numeric
out   0      -none- numeric
group 0      -none- numeric
names 1      -none- character
> names(sl.b)
[1] "stats" "n"     "conf"   "out"    "group" "names"
> sl.b$stats
      [,1]
[1,]  4.3
[2,]  5.1
[3,]  5.8
[4,]  6.4
[5,]  7.9
> sl.b$stats[1]
[1] 4.3
> sl.b$stats[3]
[1] 5.8
```

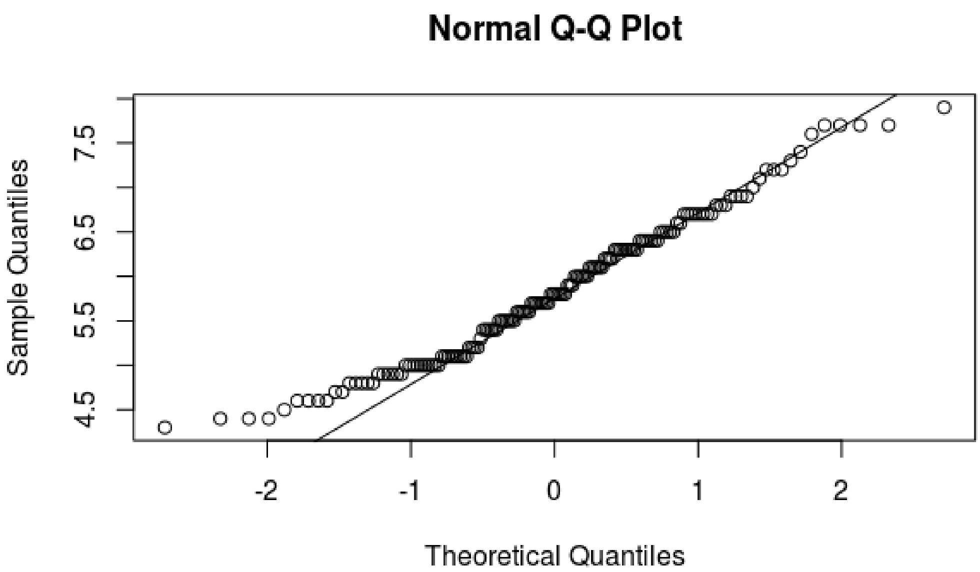
PAIRS - to see scatter plotting in matrix format-

```
pairs(iris[,1:4])
```



QQ plot- used to check normality- normally distributed or not ie Gaussian

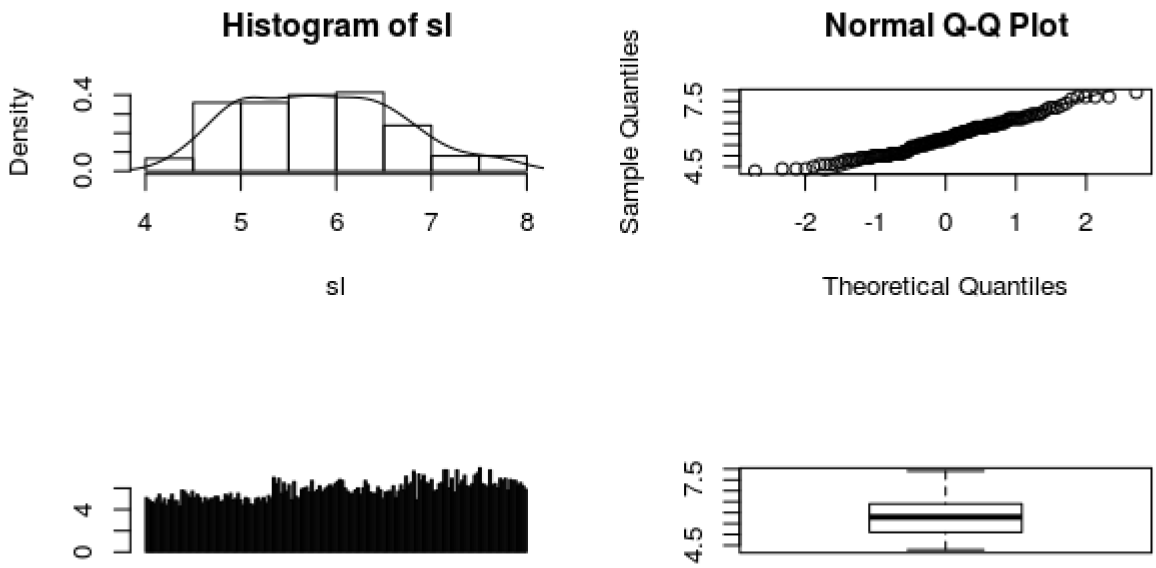
```
qqnorm(sl)
qqline(sl)
```



PAR - to see multiple types of plot at same console

```
par(mfrow=c(2,2))
hist(sl, freq = F)
lines(sl.d)
qqnorm(sl)
qqline(sl)
barplot(sl)
boxplot(sl)
```

par(mfrow=c(1,1))- this line of code is to make it back to view one plot at a time



MAKING OUR OWN FUNCTION

```
function( arglist ) expr
return(value)

eucl_dist <- function(x1, x2) sqrt(sum(x1-x2)^2)
```

TO APPLY FUNCTION ON A DATA SET

```
apply(X, MARGIN, FUN, ...)
distance <- apply(iris_features, 1, function(x) eucl_dist(x, new_sample))
```

TO SORT A DATA SET

distance_sort <- sort(distance, index.return = T)
the index.return
parameter set to TRUE , so that we also get back the indexes of the row numbers in our iris data frame corresponding to each distance computed.

```
> fourcities <- c("trivandrum", "chennai", "bangalore", "hyderabad")
```

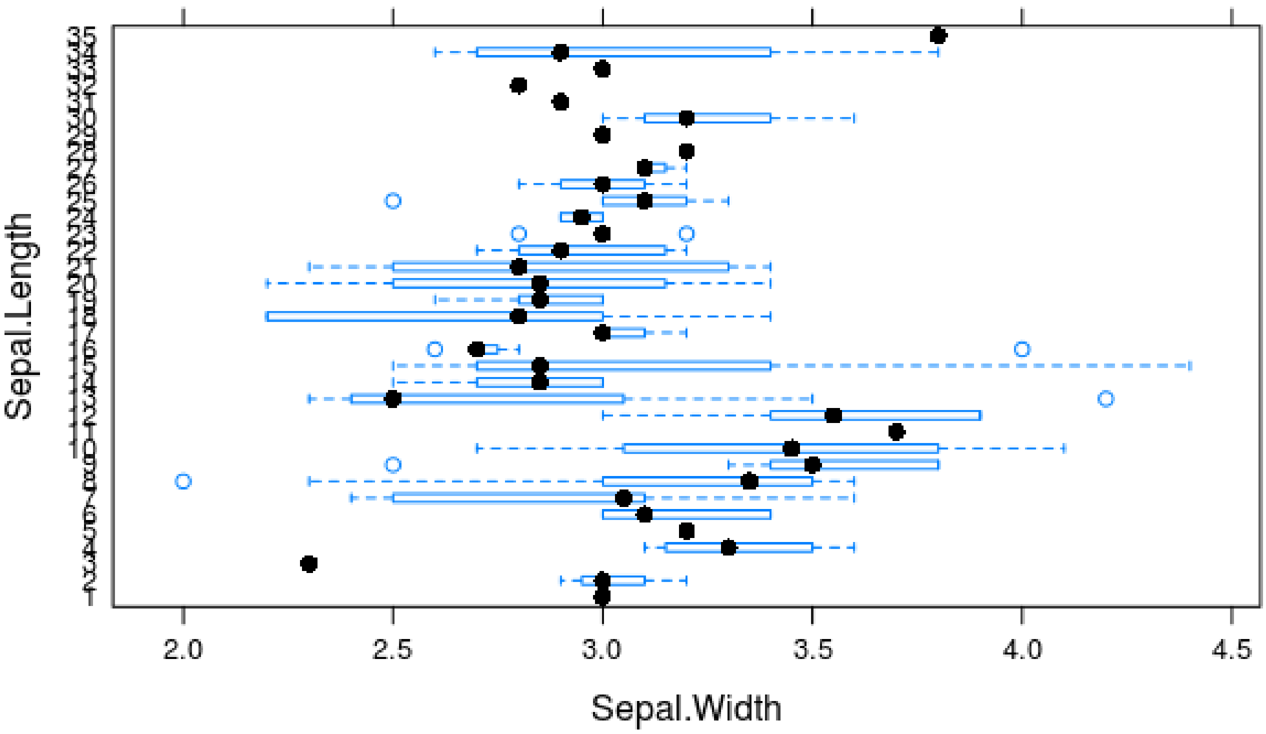
```
> sort(fourcities)
[1] "bangalore"  "chennai"    "hyderabad"  "trivandrum"

> sort(fourcities, index.return = T)
$x
[1] "bangalore"  "chennai"    "hyderabad"  "trivandrum"

$ix
[1] 3 2 4 1
```

FINDING NUMBER OF CHARACTERS nchar()

```
> fourcities <- c("trivandrum", "chennai", "bangalore", "hyderabad")
> nchar("hyderabad")
[1] 9
> nchar(fourcities)
[1] 10  7  9  9
```

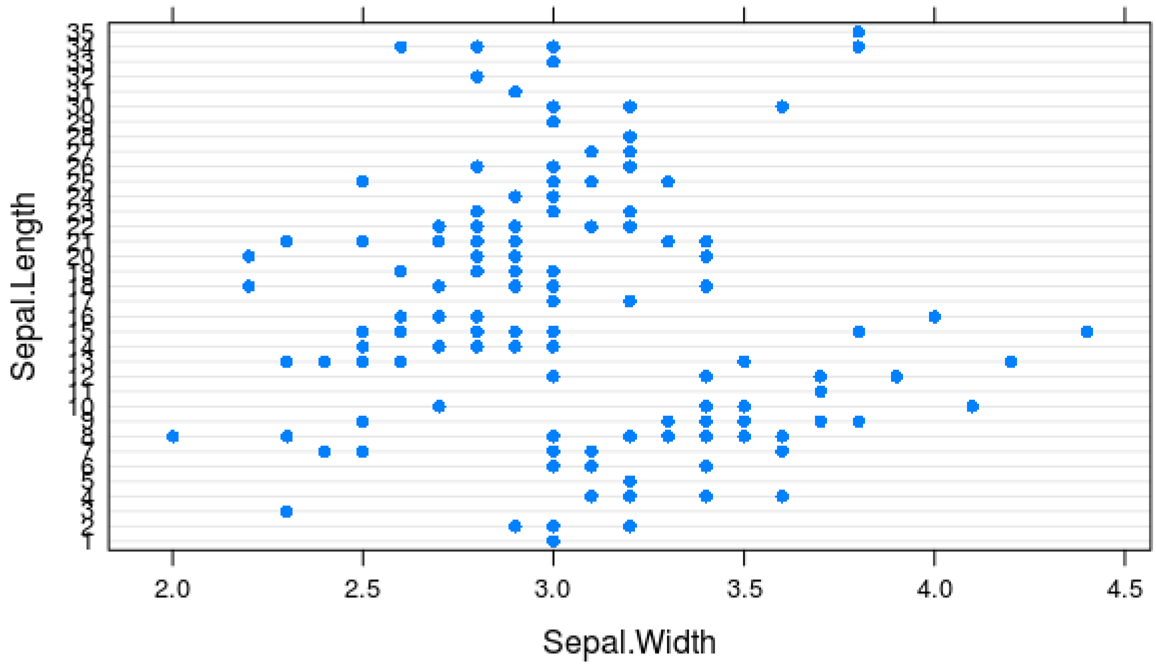
BOX AND WHISKERS PLOT

```
library("lattice")
bwplot(Sepal.Length~Sepal.Width, data = iris)
```



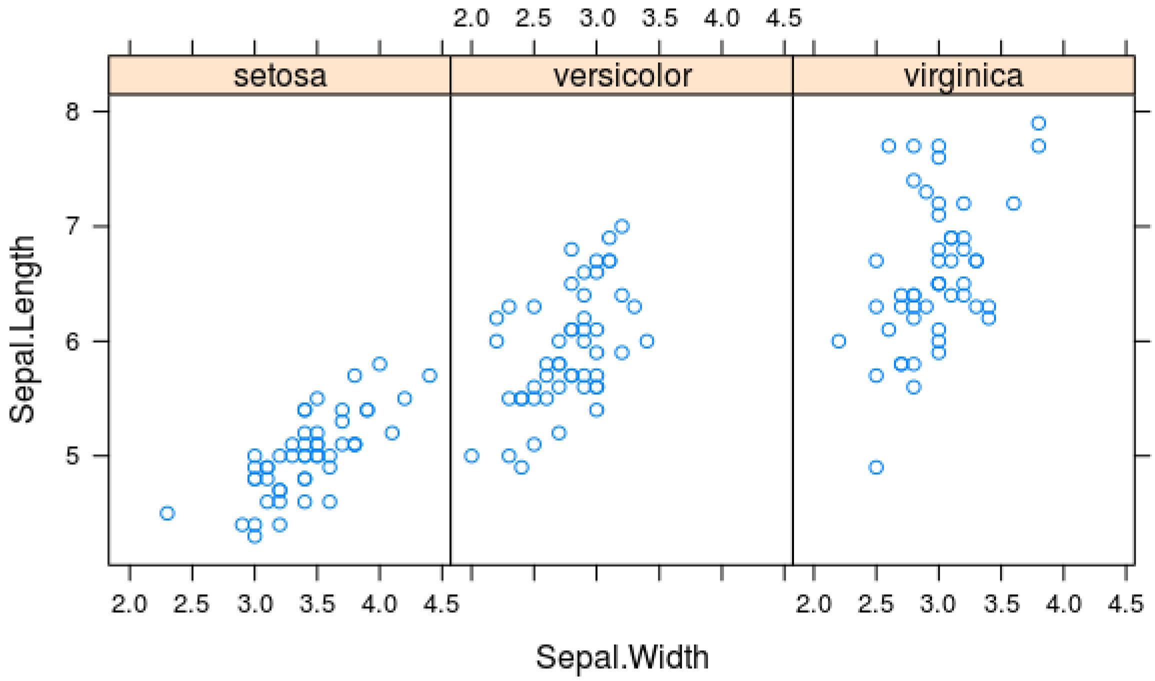
DOT PLOT

```
library("lattice")
dotplot(Sepal.Length~Sepal.Width, data = iris)
```



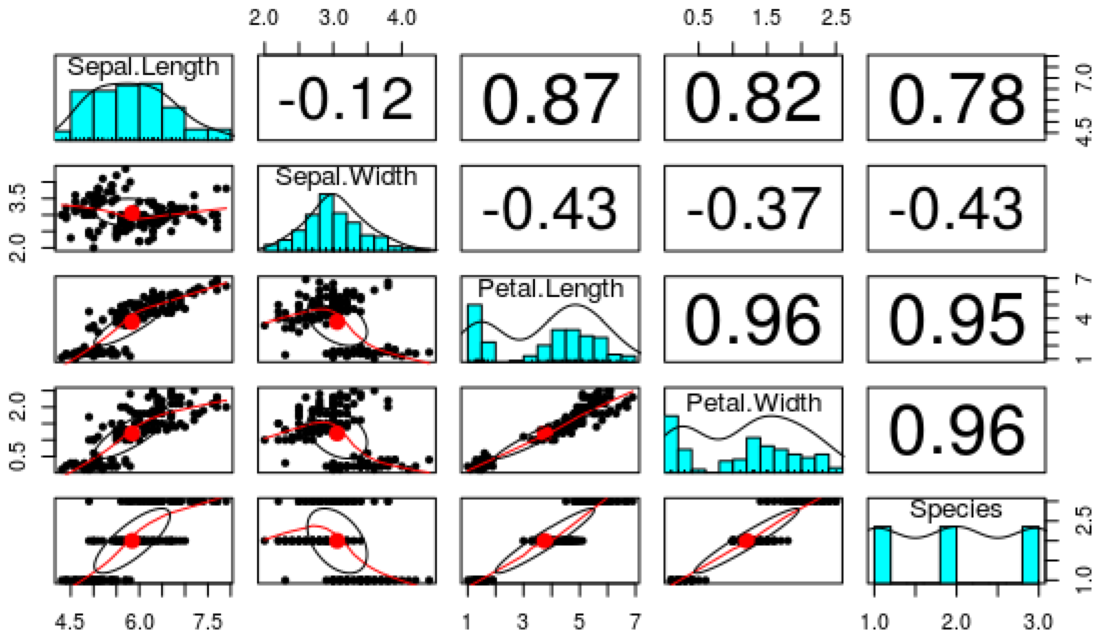
XY PLOT- we can add third variable in XY PLOT

```
 library("lattice")
xyplot(Sepal.Length~Sepal.Width|Species, data = iris)
```



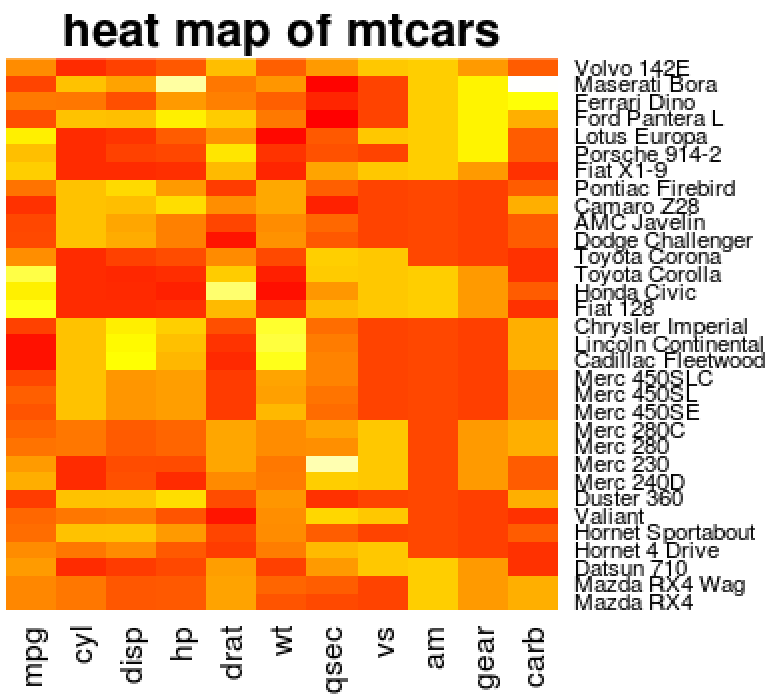
PAIRS.PANELS

```
library("psych")
pairs.panels(iris)
```



HEAT MAP- can only be used when data is converted to matrix format.

```
Install.packages(“ggplot2”)
library(ggplot2)
heatmap(as.matrix(mtcars), scale = "column", col = heat.colors(256)
        , main = "heat map of mtcars", Rowv = NA, Colv = NA)
```



TIME SERIES – to make data periodic, as in the cases of data which has dates, years
ts() - ts will make data in the form of a time frame

```
> jobs <- seq(950,1070, by = 5)
> jobs
 [1]  950  955  960  965  970  975  980  985  990  995 1000 1005 1010 1015 1020 1025 1030
1035
[19] 1040 1045 1050 1055 1060 1065 1070
> jobs <- ts(jobs, start = 1975, frequency = 12)
> jobs
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1975  950  955  960  965  970  975  980  985  990  995 1000 1005
1976 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055 1060 1065
1977 1070
```

use window() to extract subset from a time series

```
> subjobs <- window(jobs, start = 1975.25, end = 1976.75)
> subjobs
      Jan  Feb  Mar  Apr  May  Jun  Jul  Aug  Sep  Oct  Nov  Dec
1975                 965  970  975  980  985  990  995 1000 1005
1976 1010 1015 1020 1025 1030 1035 1040 1045 1050 1055
```

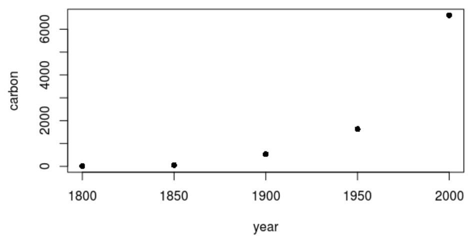
DATA FRAME

99. df <- data.frame(a,b,c) – used to make dta frames with other vectors
100. mtcars[1,2] - element in first row second coloumn
101. mtcars ["mazda","cyl"]

    making a data frame

```
    carbon <- c(8, 54, 534, 1630, 6611)
year <- c(1800, 1850, 1900, 1950, 2000)
fossil_fuel <- data.frame(year=year, carbon = carbon)
plot (carbon~year, data = fossil_fuel, pch = 16)
```

```
> fossil_fuel
  year carbon
1 1800      8
2 1850     54
3 1900    534
4 1950   1630
5 2000   6611
```

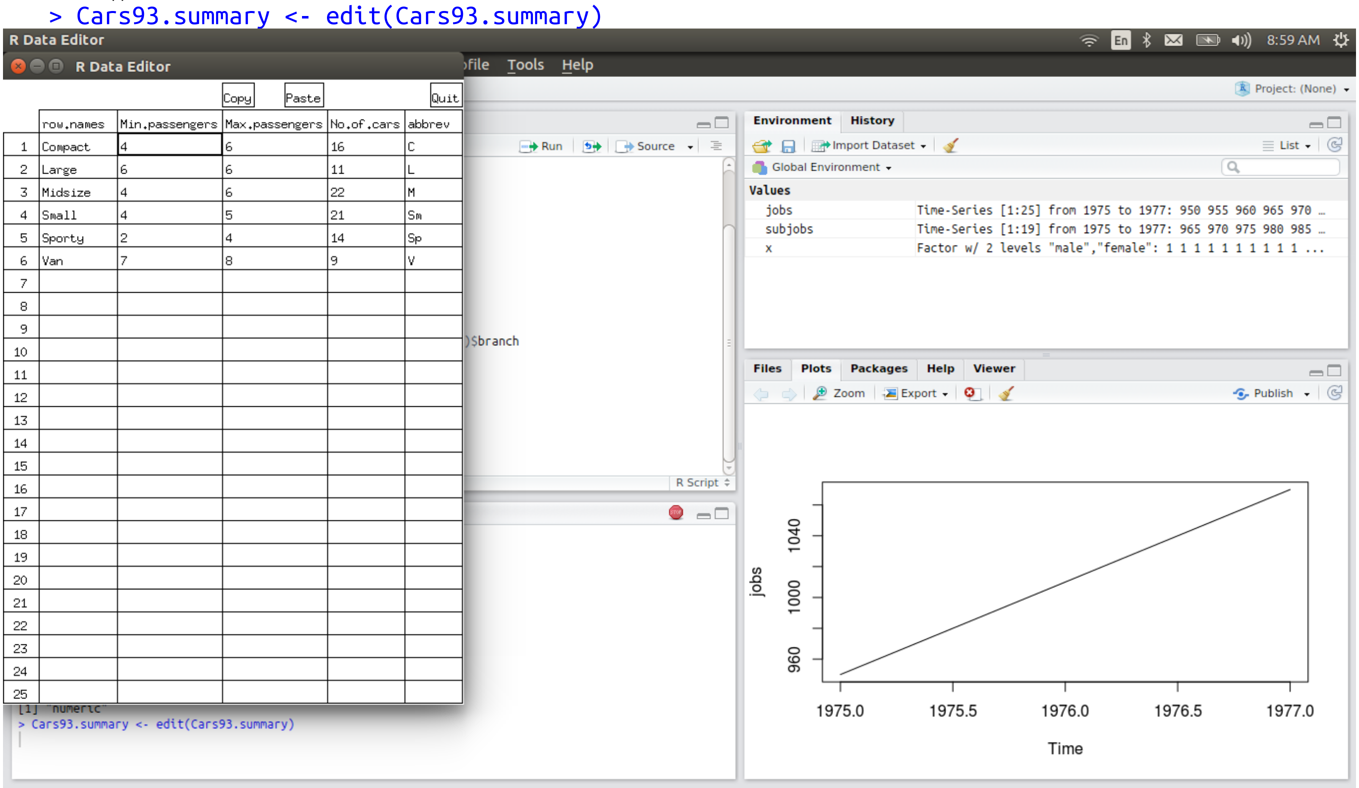


class() can be used to check the type of factor in a data frame

```
> Cars93.summary
         Min.passengers Max.passengers No.of.cars abbrev
Compact               4              6         16      C
Large                 6              6         11      L
Midsize               4              6         22      M
Small                 4              5         21     Sm
Sporty                2              4         14     Sp
Van                   7              8          9      V
> class(Cars93.summary$abbrev)
[1] "factor"
> class(Cars93.summary$Min.passengers)
[1] "numeric"
```

edit() can be used to edit the data frame

```
> Cars93.summary <- edit(Cars93.summary)
```



To know the names of rows and coloumns of a data frame

row-- rownames() or row.names()

coloumn - colnames or names

```
> Cars93.summary
         Min.passengers Max.passengers No.of.cars abbrev
Compact               4              6         16      C
Large                 6              6         11      L
Midsize               4              6         22      M
Small                 4              5         21     Sm
Sporty                2              4         14     Sp
Van                   7              8          9      V

> rownames(Cars93.summary)
[1] "Compact" "Large"   "Midsize" "Small"   "Sporty"  "Van"
> colnames(Cars93.summary)
[1] "Min.passengers" "Max.passengers" "No.of.cars"     "abbrev"
> row.names(Cars93.summary)
[1] "Compact" "Large"   "Midsize" "Small"   "Sporty"  "Van"
> names(Cars93.summary)
[1] "Min.passengers" "Max.passengers" "No.of.cars"     "abbrev"
```

The functions names() (or colnames()) and rownames() can also be used to assign new names

before changing

```
> names(Cars93.summary)
[1] "Min.passengers" "Max.passengers" "No.of.cars"     "abbrev"
```

after changing

```
> names(Cars93.summary)[3] <- "numofcars"
> names(Cars93.summary)
[1] "Min.passengers" "Max.passengers" "numofcars"      "abbrev"
```

calling rows and coloumns in a data frame

```
Cars93.summary[4, 2]
Cars93.summary[1:3, 2:3] # Rows 1-3 and columns 2-3
Cars93.summary[, 2:3] # Columns 2-3 (all rows)
```

```
Cars93.summary[, c("No.of.cars", "abbrev")] # Cols 2-3, by name
Cars93.summary[, -c(2,3)] # omit columns 2 and 3

> Cars93.summary
         Min.passengers Max.passengers numofcars abbrev
Compact               4              6        16      C
Large                 6              6        11      L
Midsize               4              6        22      M
Small                 4              5        21     Sm
Sporty                2              4        14     Sp
Van                   7              8         9      V
> Cars93.summary[2,3]
[1] 11
> Cars93.summary[1:2,2:3]
         Max.passengers numofcars
Compact               6        16
Large                 6        11
> Cars93.summary[,2:3]
         Max.passengers numofcars
Compact               6        16
Large                 6        11
Midsize               6        22
Small                 5        21
Sporty                4        14
Van                   8         9
> Cars93.summary[1:2,]
         Min.passengers Max.passengers numofcars abbrev
Compact               4              6        16      C
Large                 6              6        11      L


> Cars93.summary[,c("Max.passengers", "numofcars")]
         Max.passengers numofcars
Compact               6        16
Large                 6        11
Midsize               6        22
Small                 5        21
Sporty                4        14
Van                   8         9
> Cars93.summary[,-c(2,3)]
         Min.passengers abbrev
Compact               4      C
Large                 6      L
Midsize               4      M
Small                 4     Sm
Sporty                2     Sp
Van                   7      V
```

The subset() function offers an alternative way to extract rows and columns

Use the argument select to specify a subset of columns

```
> Cars93.summary
         Min.passengers Max.passengers numofcars abbrev
Compact               4              6        16      C
Large                 6              6        11      L
Midsize               4              6        22      M
Small                 4              5        21     Sm
Sporty                2              4        14     Sp
Van                   7              8         9      V

> subset(Cars93.summary, subset = c(T,F,T,F,F,F))  ## by default it selects rows
         Min.passengers Max.passengers numofcars abbrev
Compact               4              6        16      C
Midsize               4              6        22      M
```

Use of the subscript notation to extract a column, as in Cars93.summary[, 1], returns a vector. By contrast, extraction of the raw Cars93.summary[1,] returns a data frame, necessary because this allows different elements (columns) to retain their existing classes

MAKING LIST- list is a collection of R objects

102. V<-list(A,B) - can be used to make vector of by combining diffent kind of vector
103. attach(V) – attach list eg- V to R so that we will only call the name of vectors only
104. detach (V) – vice versa of R

```
> USACanadapop <- list( USAcities = c("NY", "LA", "Chicago"),
+                       Canadacities = c("orlando", "montreal"),
+                       population = c(USA = 120, Canada = 145))
> USACanadapop
$USAcities
[1] "NY"      "LA"      "Chicago"

$Canadacities
[1] "orlando"  "montreal"

$population
   USA Canada
   120    145
```

STACK- used to convert different coloumns of a data frame in to another data frame of two coloumns whose first coloumn has the stacked values of all the columns of initial data frame and second coloumn is a factor with levels as names of coloumns of initial data frame

```
> Cars93.summary
         Min.passengers Max.passengers numofcars abbrev
Compact               4              6        16      C
Large                 6              6        11      L
Midsize               4              6        22      M
Small                 4              5        21     Sm
Sporty                2              4        14     Sp
Van                   7              8         9      V

> cars <- stack(Cars93.summary, select = 1:4)
Warning message:
In stack.data.frame(Cars93.summary, select = 1:4) :
  non-vector columns will be ignored
> head(cars)
  values            ind
1      4 Min.passengers
2      6 Min.passengers
3      4 Min.passengers
4      4 Min.passengers
5      2 Min.passengers
6      7 Min.passengers
> str(cars)
'data.frame':   18 obs. of  2 variables:
 $ values: num  4 6 4 4 2 7 6 6 6 5 ...
 $ ind   : Factor w/ 3 levels "Max.passengers",..: 2 2 2 2 2 2 1 1 1 1 ...
```

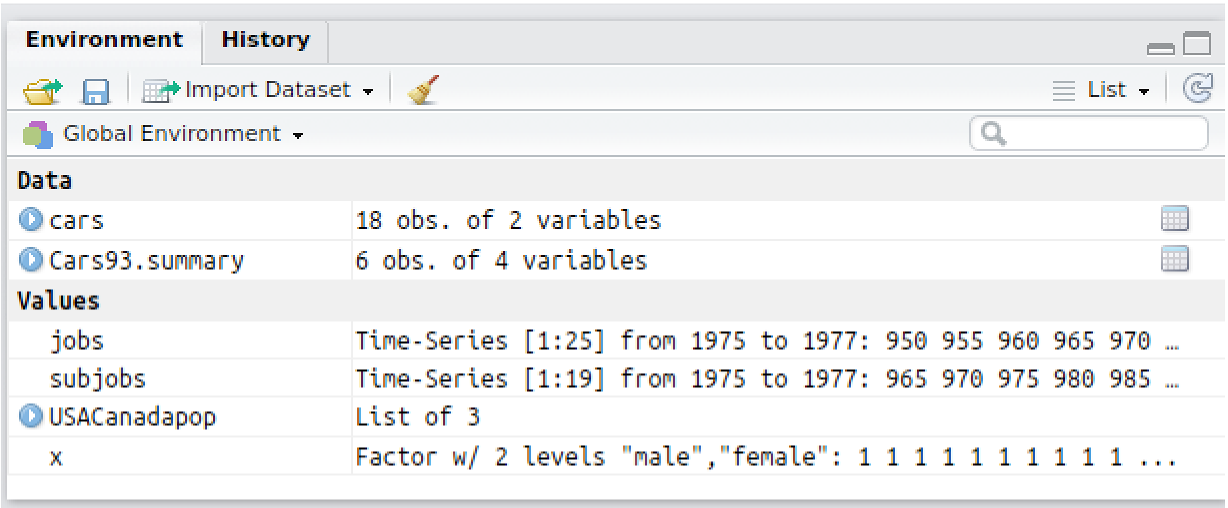
unstack will remove stacked format

```
> unstack(cars)
  Max.passengers Min.passengers numofcars
1              6              4        16
2              6              6        11
3              6              4        22
4              5              4        21
5              4              2        14
6              8              7         9
```

SOME IMPORTANT FUNCTIONS

all()- Given a set of logical vectors, are all of the values true?

```
 > all(jobs)
[1] TRUE
Warning message:
In all(jobs) : coercing argument of type 'double' to logical
> all(x)
Error in Summary.factor(c(1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L, 1L,  :
  ‘all’ not meaningful for factors
```

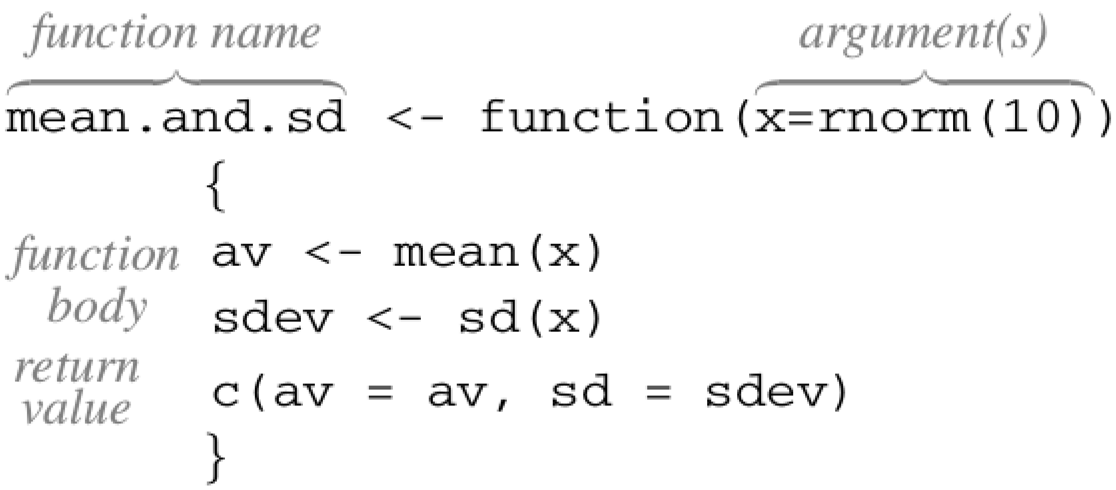


SAPPLY - it can be used to apply mean or median or value of a function on all the coloumns of a data frame together (except factors)

```
> head(iris)
  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
> sapply(iris[ ,-5], range)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]          4.3         2.0          1.0         0.1
[2,]          7.9         4.4          6.9         2.5
```

CREATING MY OWN FUNCTION

```
> mean.and.sd <- function(x){
+   Mean <- mean(x)
+   SD <- sd(x)
+   c(MN = Mean, Sdev = SD)
+ }
> mean.and.sd(dd)
      MN     Sdev
3.800000 4.816638
```



IF ELSE STATEMENTS-

```
> if(mean(dd) > median(dd)) print("mean > meadian") else print("mean < = median")
[1] "mean > meadian"
> mean(dd)
[1] 3.8
> median(dd)
[1] 1
```

FINDING VALUES IN A GIVEN VECTOR -- %in%

```
> y <- rep(1:4, each = 2)
> y
[1] 1 1 2 2 3 3 4 4
> y [y %in% c(2,3)]
[1] 2 2 3 3
> y [y %in% c(2)]
[1] 2 2
> y [y %in% 2]
[1] 2 2
> z <- "lijin is happy"
> z [z %in% "happy"]
character(0)
> a <- z [z %in% "happy"]
> a
character(0)
```

MATCHING the values of a vector

```
> y
[1] 1 1 2 2 3 3 4 4
> match(y, c(2,3))
[1] NA NA  1  1  2  2 NA NA
> match(y, c(2,3), nomatch = 0) ## arguemnt nomatch will make NA values 0
[1] 0 0 1 1 2 2 0 0
> match(z, "happy")
[1] NA
> match(z, "lijin is happy")
[1] 1
> b <- strsplit(z, " ")
> b
[[1]]
[1] "lijin" "is"    "happy"
> match(b, c("is", "happy"))
[1] NA
```

IDENTIFICATION OF ROWS THAT HAS MISSING VALUES AND OMIT THE SAME-

complete.cases()

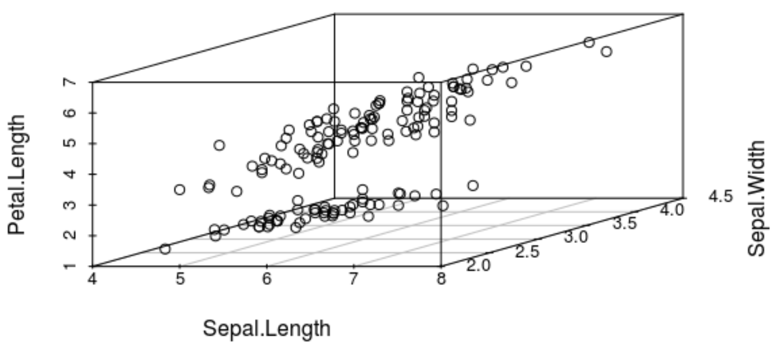
```
> science [!complete.cases(science), ]   ## science is data set in DAAG package
    State PrivPub school class  sex like Class
671   ACT  public     19     1 <NA>    5  19.1
672   ACT  public     19     1 <NA>    5  19.1
```

The function na.omit() omits any rows that contain missing values. For example,

```
> dim(science)
[1] 1385 7
> Science <- na.omit(science)
> dim(Science)
[1] 1383 7
```

SCATTER PLOT 3D

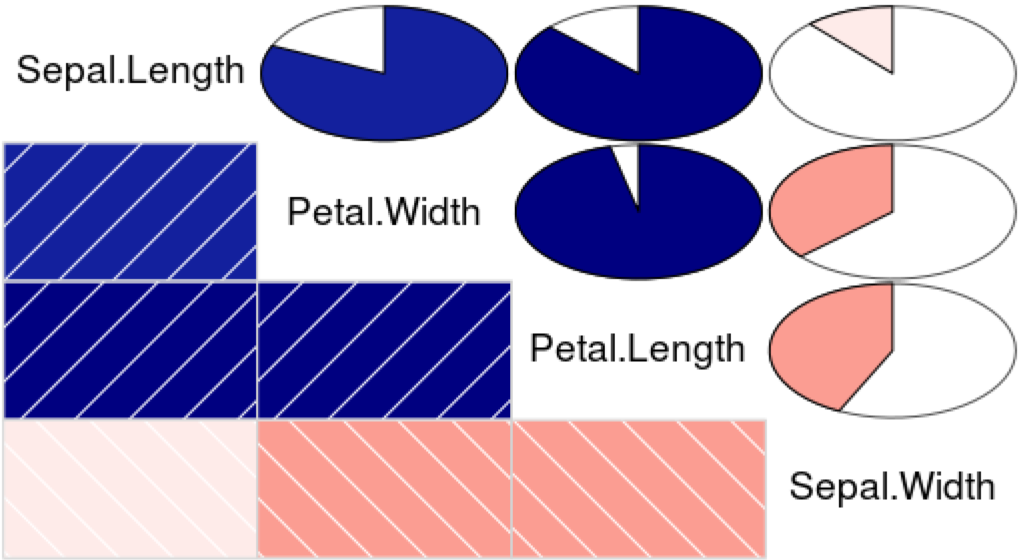
```
install.packages("scatterplot3d")

library(scatterplot3d)

scatterplot3d(iris[1:3])
```



CORRELOGRAM OR CORRGRAM

```
> cor(iris[,1:4])
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```
library(corrgram)

> corrgram(iris, order = T, lower.panel = panel.shade, upper.panel = panel.pie,
+          text.panel = panel.txt)
```



JITTER - USED TO ADD NOICE TO THE DATA

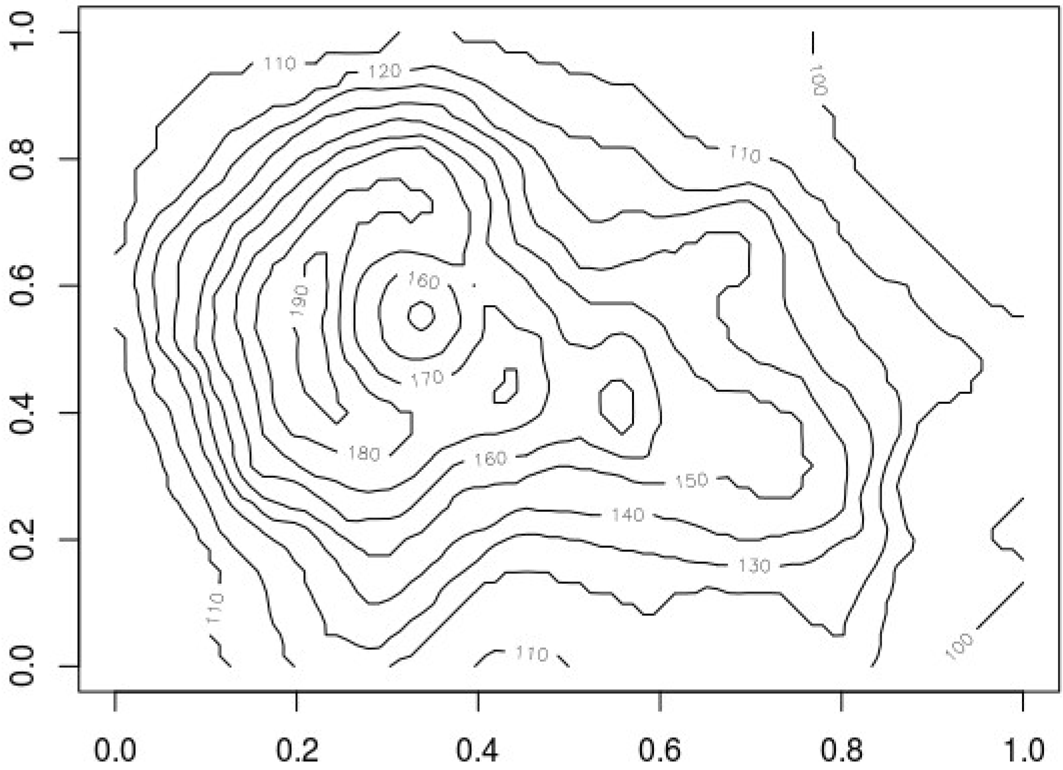
```
> dd
[1]  1  0  8  0 10
> jitter(dd)
[1]  1.19015523  0.07055692  7.91184899  0.06337822 10.19998752
> jitter(dd, factor = 2)
[1]  1.0757836 -0.2615765  8.3119029 -0.0171281  9.8966233
> jitter(dd, factor = 2, amount = 2)
[1] -0.1065645  1.0107661  6.9332359 -0.2859592 11.1876223
```

CONTOUR PLOT- same as contour lines used in maps

```
> str(volcano)
 num [1:87, 1:61] 100 101 102 103 104 105 105 106 107 108 ...
```

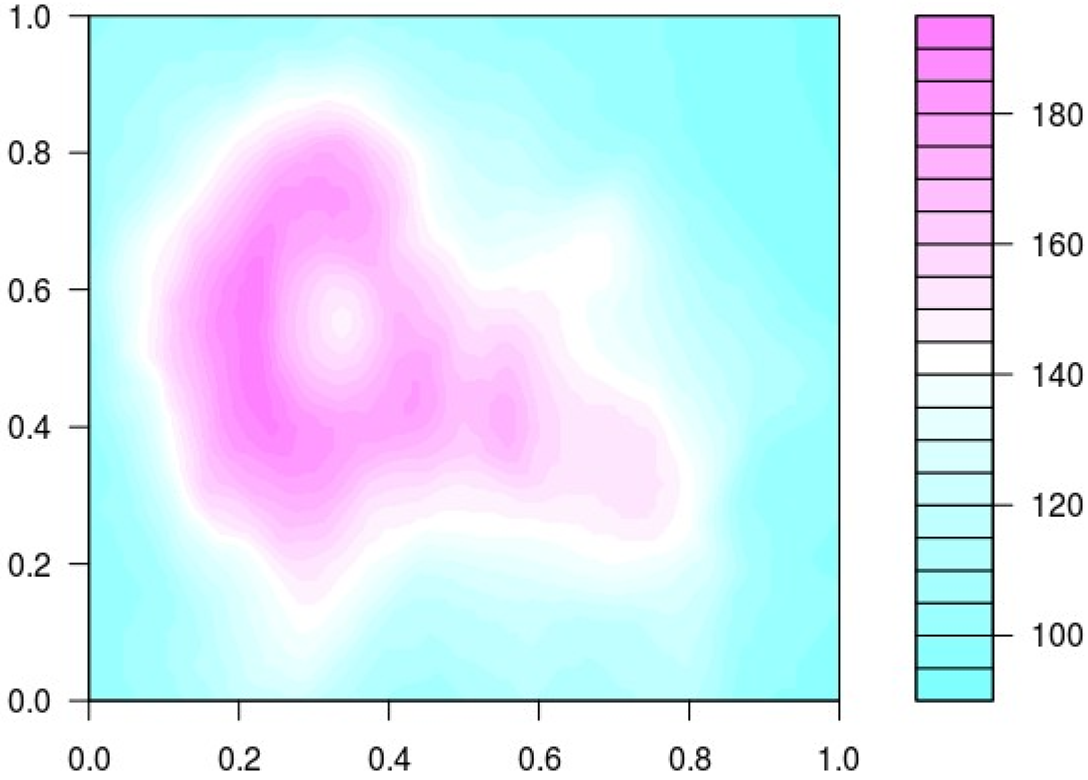
matrix with 87 rows and 61 columns, rows corresponding to grid lines running east to west and columns to grid lines running south to north

```
> contour(volcano)
```



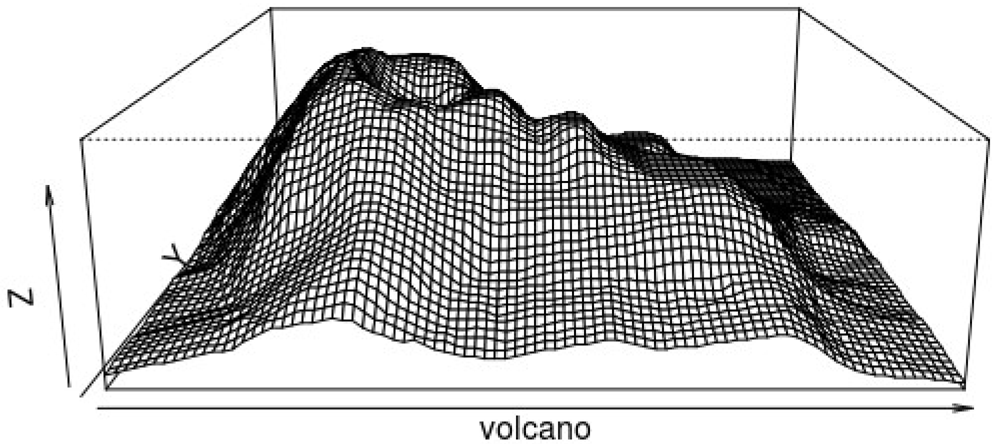
FILLED CONTOUR- uses colors in a contour map to indicate height difference

```
> filled.contour(volcano)
```



3D- PERSPECTIVE PLOT-

```
> persp(volcano, expand = 0.3)
```



LEVEL PLOT- SIMILAR TO CONTOUR PLOT- instead of lines we use colours

```
sl <- iris$Sepal.Length

sw <- iris$Sepal.Width

pw <- iris$Petal.Width

levelplot(pw ~ sl*sw, data = iris)  ### pw as function of sw and sl
```