INTRODUCTORY CONCEPTS OF R PROGRAMMING

Dr.M.Jayalakshmi
Assistant Professor (SG)
Department of Mathematics
School of Advanced Science
Vellore Institute of Technology
Vellore-14

Tamilnadu

September 20,2023

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

What is R?

- R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT& T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.
- R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis,classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity

- One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.
- R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form

The R environment

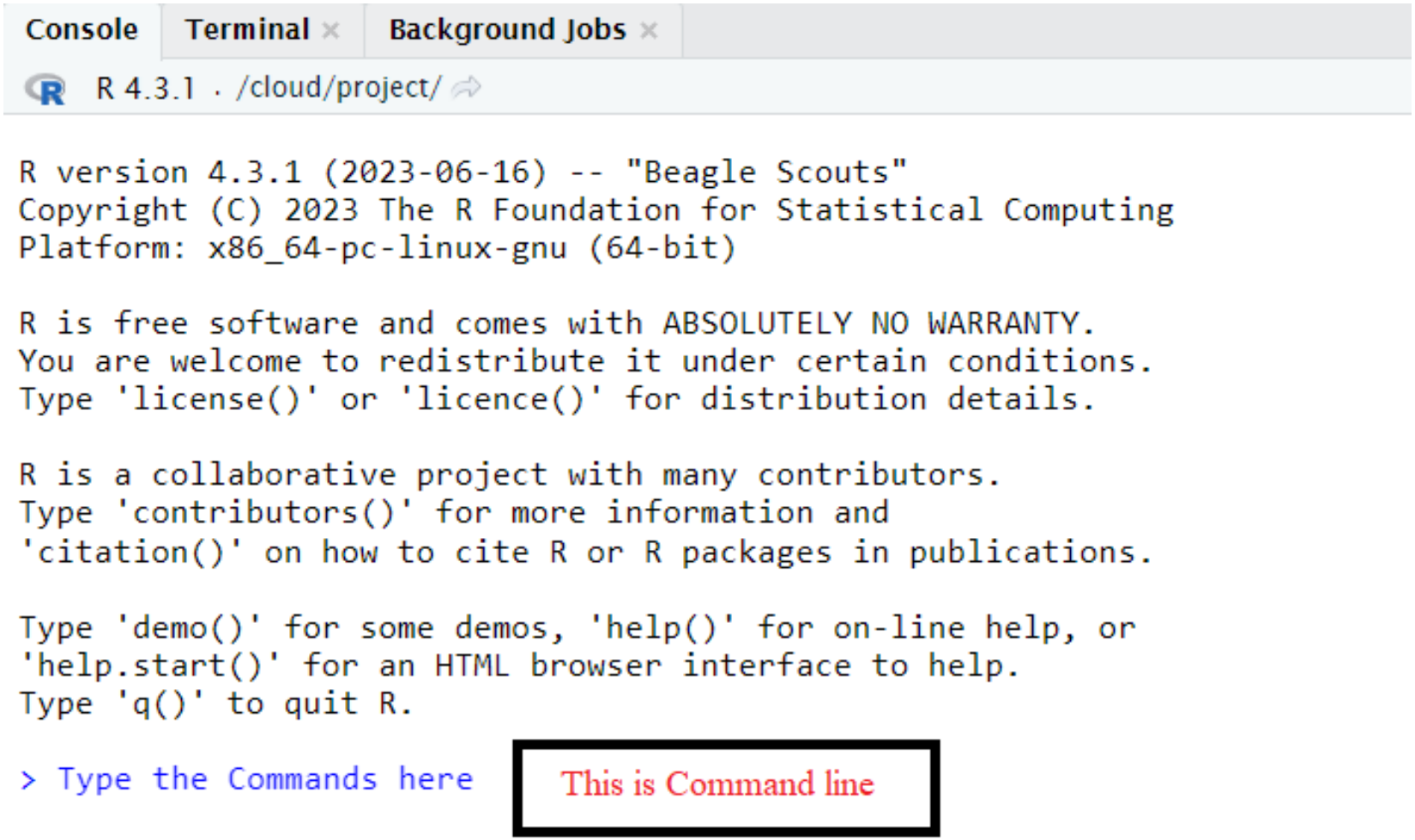
- R is an integrated suite of software facilities for data manipulation,calculation and graphical display
- It includes an effective data handling and storage facility
- A suite of operators for calculations on arrays, in particular matrices
- A large, coherent, integrated collection of intermediate tools for data analysis
- Graphical facilities for data analysis and display either on-screen or on hardcopy
- A well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

Why should you adopt R Programming Language?

- R, SAS, and SPSS are three statistical languages. Of these three statistical languages, R is the only an open source. SAS is the most important private software business in the world. SPSS is now overseen by IBM. R Programming is extensible and hence, R groups are noted for its energetic contributions. Lots of Rs typical features can be written in R itself and hence, R has gotten faster over time and serves as a glue language.

- **Installation**
  - R can be downloaded from one of the mirror sites in http://cran.r-project.org/mirrors.html. You should pick your nearest location.
- **Using External Data**
  - R offers plenty of options for loading external data, including Excel, Minitab and SPSS files. We have included a tutorial titled Data Import on the subject for the purpose.

What is Command line



- Execution of commands in R is not menu driven. (Not like clicking over buttons to get outcome)
- We need to type the commands.
- Single line and multi line commands are possible to write.
- When writing multi-line programs, it is useful to use a text editor rather than execute everything directly at the command line.

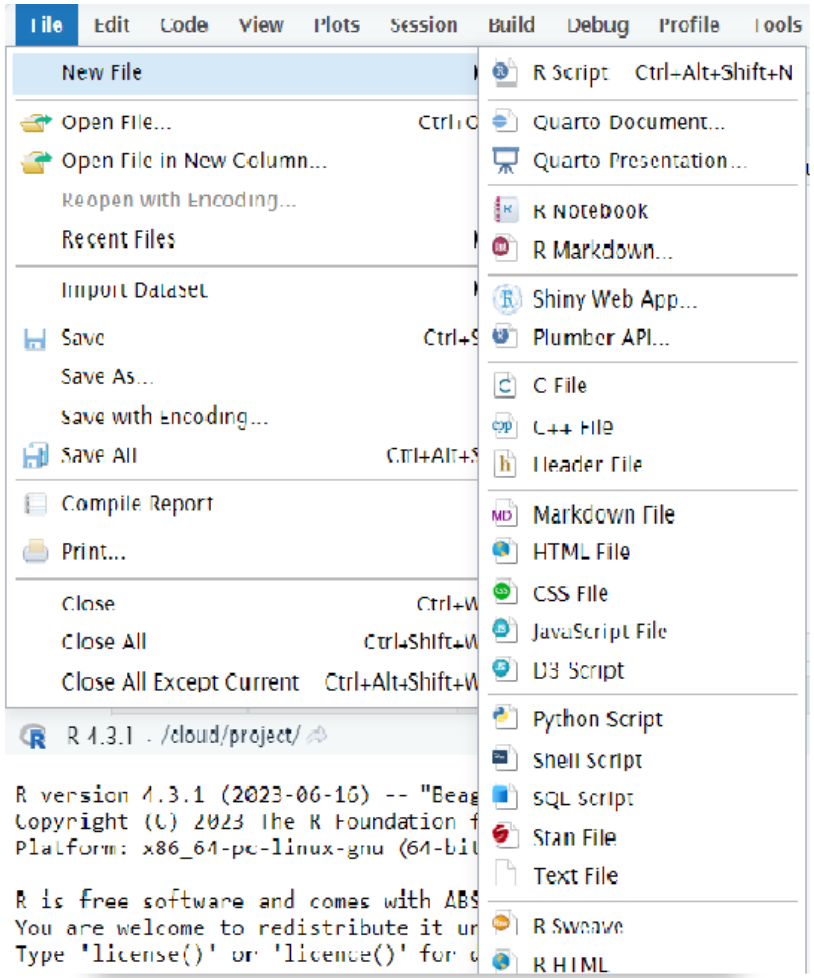
Command line versus scripts

At this point R will

open a window entitled

`Untitled-R Editor.`

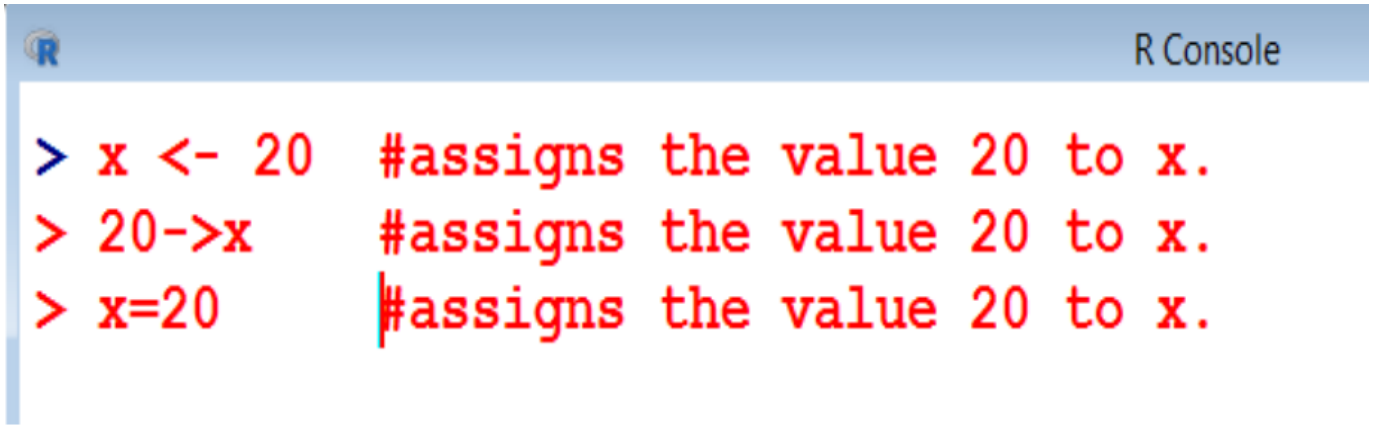
We may type and edit in this.



If we want to execute a line or a group of lines, just highlight them and press `Ctrl+R`.

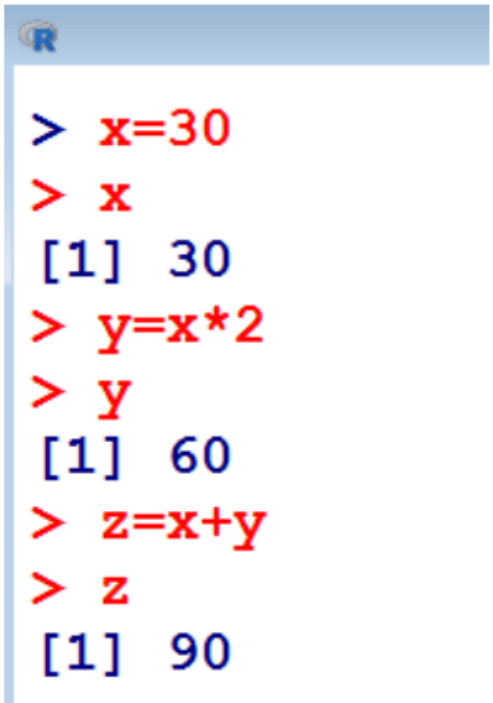
Basics

- $>$ is the prompt sign in R. you can enter numbers and perform calculations.
- The assignment operators are the left arrow with dash $<$- ,-$>$and equal sign $=$.



- > x = 30 #assigns the value 30 to x.
- > y = x * 2 #assigns the value 2*x to y.
- > z = x + y #assigns the value x + y to z.

Output:-



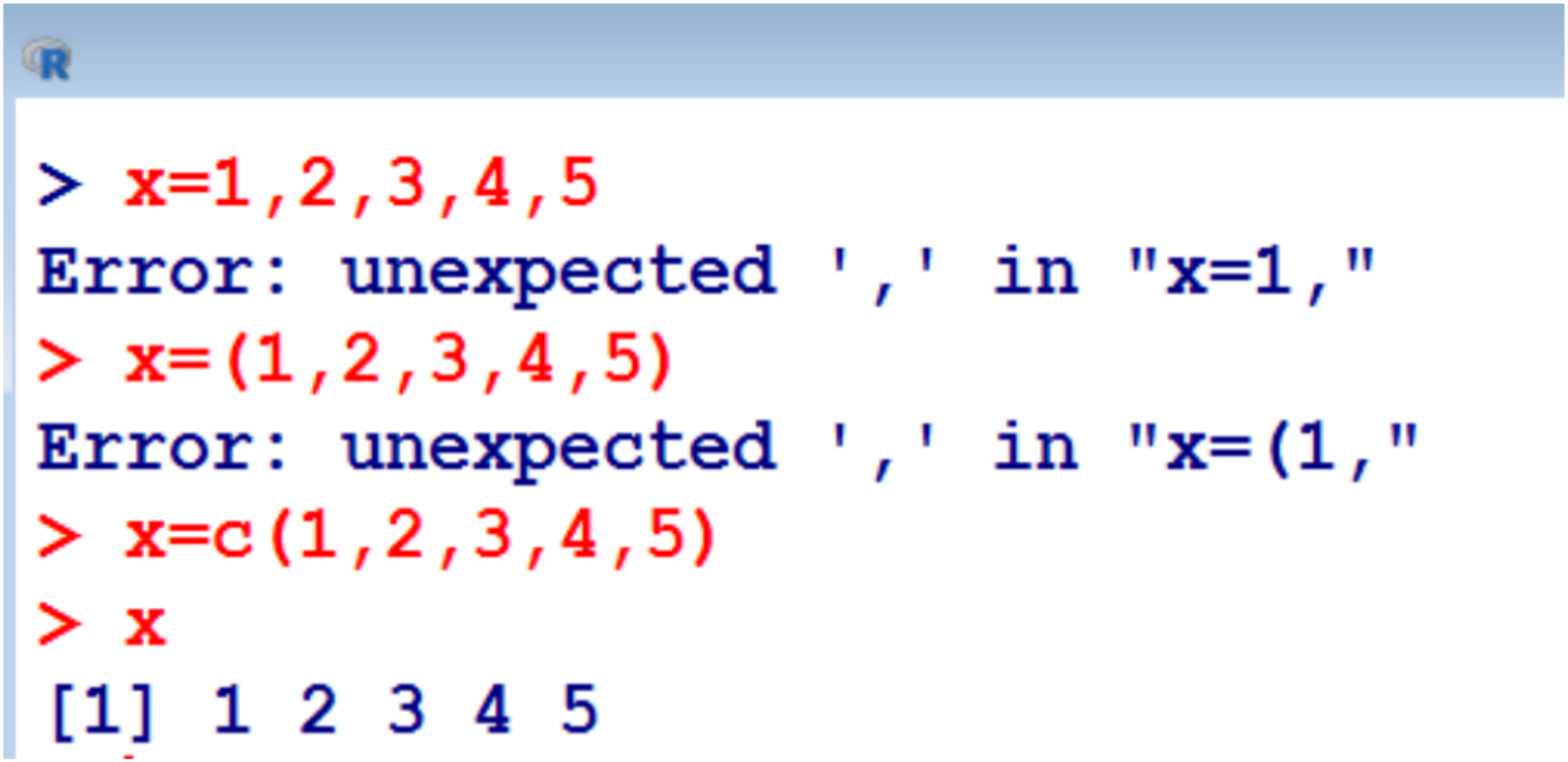
- # : The character # marks the beginning of a comment.All characters until the end of the line are ignored.

```
> # mu is the mean
> # x <- 20 is treated as comment only
```

- Capital and small letters are different.

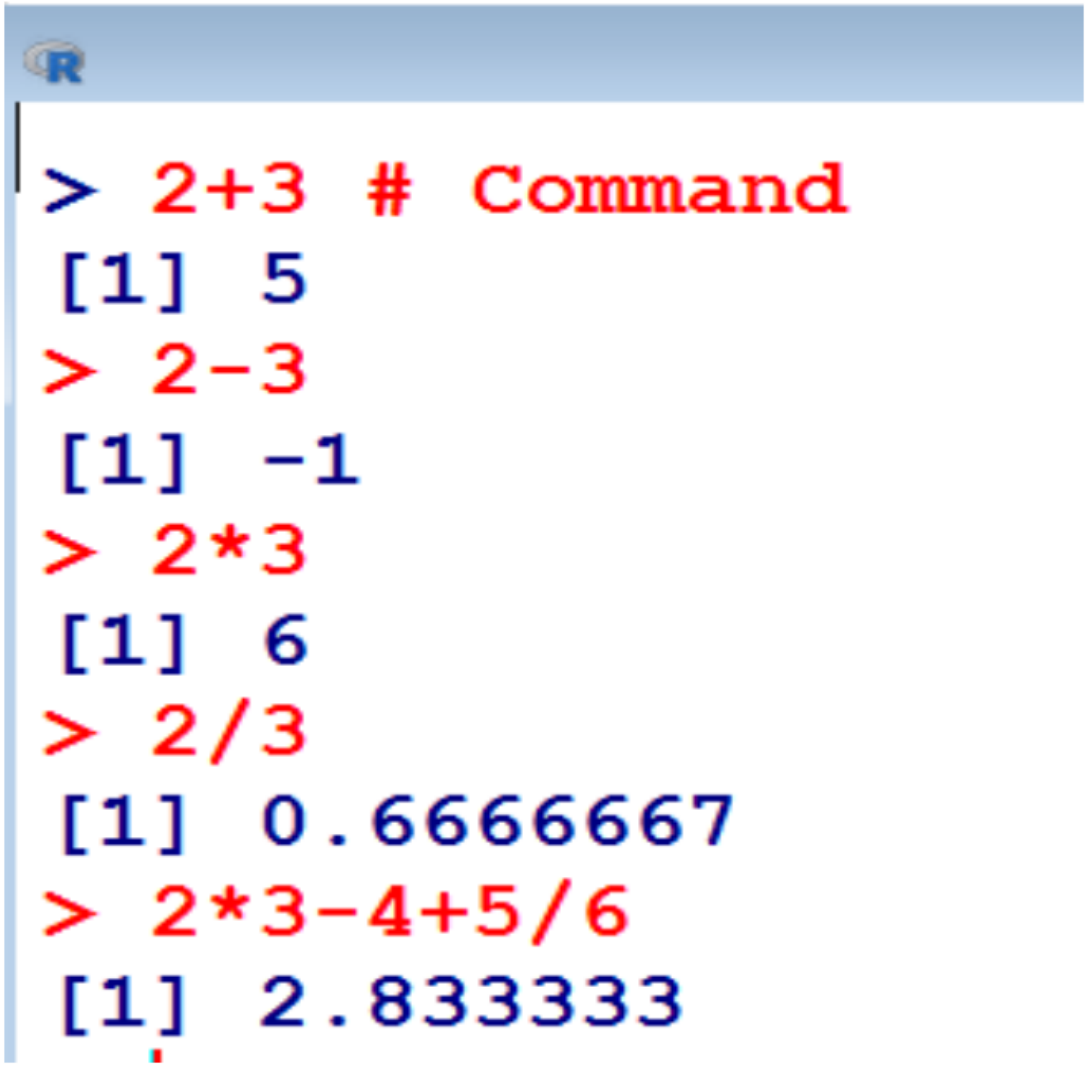
```
> x=20
> X
Error: object 'X' not found
> x
[1] 20
> X=30
> X
[1] 30
> x+X
[1] 50
```

- The command c(1,2,3,4,5) combines the numbers 1,2,3,4 and 5 to a vector.



```
> x=1,2,3,4,5
Error: unexpected ',' in "x=1,"
> x=(1,2,3,4,5)
Error: unexpected ',' in "x=(1,"
> x=c(1,2,3,4,5)
> x
[1] 1 2 3 4 5
```

R as a Calculator



```
> 2^3          # Command
 [1] 8
> 2**3         # Command
 [1] 8
> 2^0.5        # Command
 [1] 1.414214
> 2**0.5       # Command
 [1] 1.414214
> 2^-0.5       # Command
 [1] 0.7071068
> 2^3^2        # Command
 [1] 512
> (2^3)^2      # Command
 [1] 64
> 2^(3^2)      # Command
 [1] 512
```

R Console

```
> c(2,3,5,7)^2                # command: application to a vector
[1]  4  9 25 49
> c(2,3,5,7)^c(2,3)           # command: application to a vector
[1]   4  27  25 343
> c(1,2,3,4,5,6)^c(2,3,4)     # command: application to a vector
[1]    1    8   81   16  125 1296
> c(2,3,5,7)*3                # command: application to a vector
[1]  6  9 15 21
> c(2,3,5,7)*c(2,3)           # command: application to a vector
[1]  4  9 10 21
> c(1,2,3,4,5,6)*c(2,3,4)     # command: application to a vector
[1]  2  6 12  8 15 24
> c(2,3,5,7)*c(-2,-3,-5,8)# command: application to a vector
[1]  -4  -9 -25  56
> c(2,3,5,7) + 10           # command: application to a vector
[1] 12 13 15 17
```

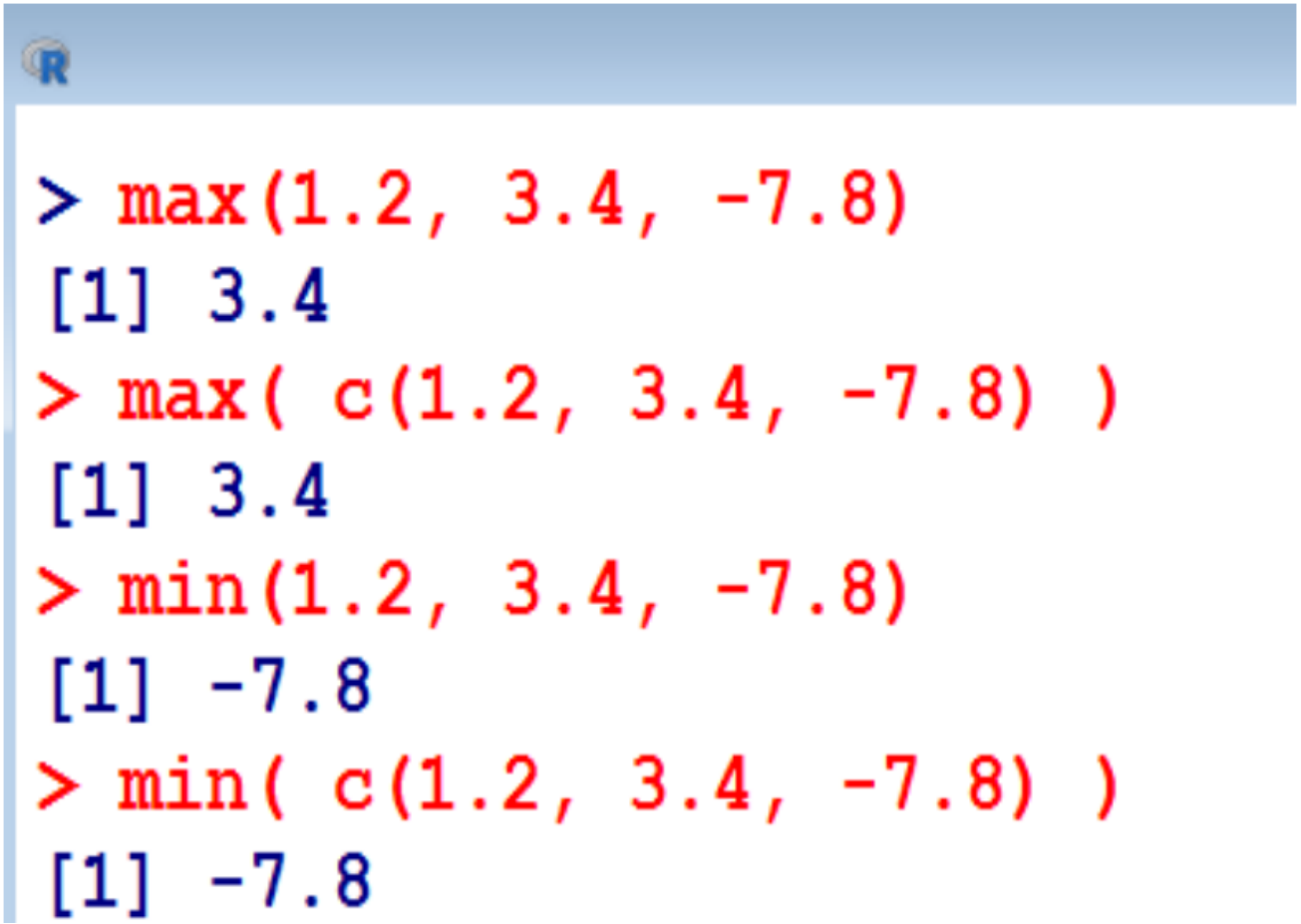
- **Integer Division: Division in which the fractional part (remainder) is discarded**

```
> c(2,3,5,7) %/% 2
[1] 1 1 2 3
> c(2,3,5,7) %/% c(2,3)
[1] 1 1 2 2
```

- **x mod y : modulo operation finds the remainder after division of one number by another**

```
> c(2,3,5,7) %% 2
[1] 0 1 1 1
> c(2,3,5,7) %% c(2,3)
[1] 0 0 1 1
```

Maximum: max & Minimum: min



```
> max(1.2, 3.4, -7.8)
[1] 3.4
> max( c(1.2, 3.4, -7.8) )
[1] 3.4
> min(1.2, 3.4, -7.8)
[1] -7.8
> min( c(1.2, 3.4, -7.8) )
[1] -7.8
```

Overview Over Further Functions

`abs()`	Absolute value
`sqrt()`	Square root
`round(), floor(), ceiling()`	Rounding, up and down
`sum(), prod()`	Sum and product
`log(), log10(), log2()`	Logarithms
`exp()`	Exponential function
`sin(), cos(), tan(), asin(), acos(), atan()`	Trigonometric functions
`sinh(), cosh(), tanh(), asinh(), acosh(), atanh()`	Hyperbolic functions

```
> abs(-4)
[1] 4
> abs(c(-1,-2,-3,4,5))
[1] 1 2 3 4 5
> sqrt(4)
[1] 2
> sqrt(c(4,9,16,25))
[1] 2 3 4 5
> sum(c(2,3,5,7))
[1] 17
> prod(c(2,3,5,7))
[1] 210
> round(1.23)
[1] 1
> round(1.83)
[1] 2
```

- log() function computes natural logarithms (Ln) for a number or vector. log10 computes common logarithms (Lg).log2 computes binary logarithms (Log2). log(x,b) computes logarithms with base b.

```
> log(10)
[1] 2.302585
> log10(10)
[1] 1
> log(9,base=3)
[1] 2
```

Complex functions

R Console

```
> var<-3+5i          # a variable with a complex value
> var
[1] 3+5i
> Re(var)            # real part
[1] 3
> Im(var)            # imaginary part
[1] 5
> Conj(var)          # complex conjugate
[1] 3-5i
> Mod(var)           # complex modulus
[1] 5.830952
> Arg(var)           # complex argument
[1] 1.030377
```

Matrix

- **Matrices are important objects in any calculation. A matrix is a rectangular array with *p* rows and *n* columns.**
- **Note:-**

➢ The parameter nrow defines the row number of a matrix.

➢ The parameter ncol defines the column number of a matrix.

➢ The parameter data assigns specified values to the matrix elements.

- **In R, a 4×2-matrix X can be created with a following command:**

```
> x <- matrix( nrow=4, ncol=2,data=c(1,2,3,4,5,6,7,8))
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

- **One can access a single element of a matrix with x[i,j]:**

```
> x[3,2]
[1] 7
```

- **In case, the data has to be entered row wise, then a 4×2-matrix X can be created with**

R Console

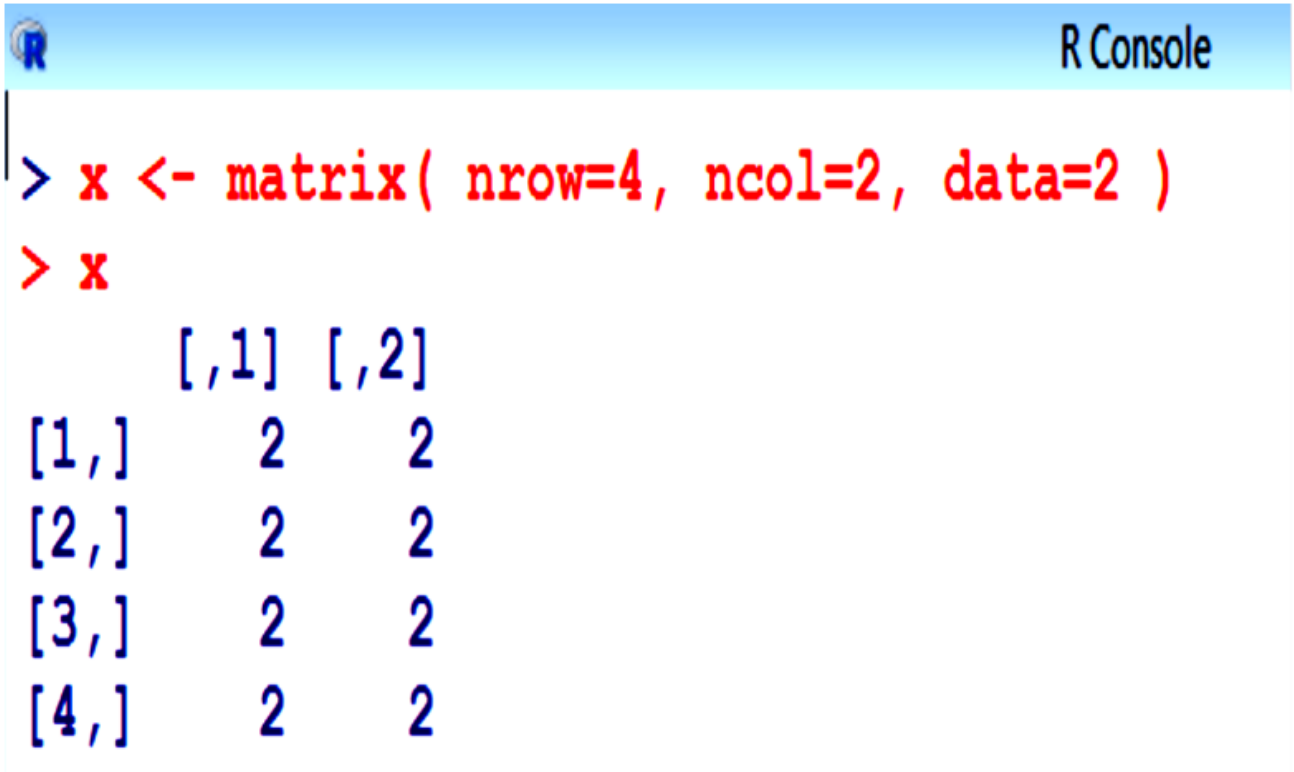
```
> x <- matrix( nrow=4, ncol=2,data=c(1,2,3,4,5,6,7,8),byrow=TRUE)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```

- **We can get specific properties of a matrix:**

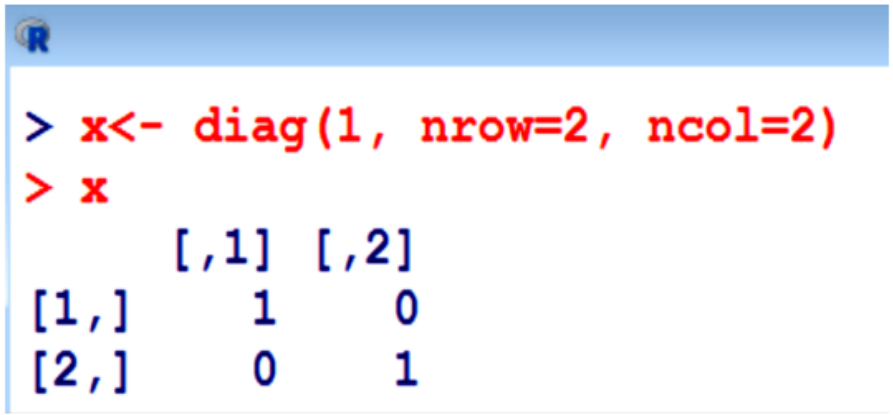
R Console

```
> dim(x)      #dimension of the matrix
[1] 4 2
> nrow(x)     #number of rows
[1] 4
> ncol(x)     #number of columns
[1] 2
> mode(x)     #Type of storage
[1] "numeric"
> attributes(x)  #Informs the dimension of the matix
$dim
[1] 4 2
```

- **Assigning a specified number to all matrix elements:**



- **Construction of a diagonal matrix, here the identity matrix of a dimension 2:**



```
> x<- diag(1, nrow=2, ncol=2)
> x
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

- Transpose of a matrix X: X'

R Console

```
> x <- matrix( nrow=4, ncol=2,data=c(1,2,3,4,5,6,7,8))
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> y<-t(x)
> y
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

- **Multiplication of a matrix with a constant**

```
> 4*x
     [,1] [,2]
[1,]    4    8
[2,]   12   16
[3,]   20   24
[4,]   28   32
```

- **Matrix multiplication: operator %*%**

```
> xt<-t(x)%*%x
> xt
     [,1] [,2]
[1,]   84  100
[2,]  100  120
```

- Cross product of a matrix X, X'X, with a function crossprod

```
> xt <- crossprod(x)
> xt
     [,1] [,2]
[1,]   84  100
[2,]  100  120
>
```

- Note: Command crossprod() executes the multiplication faster than the conventional method with t(x)%*%x

- **Addition and subtraction of matrices (of same dimensions!) can be executed with the usual operators + and –**

```
> x + 6*x
     [,1] [,2]
[1,]    7   14
[2,]   21   28
[3,]   35   42
[4,]   49   56
> 5*x - x
     [,1] [,2]
[1,]    4    8
[2,]   12   16
[3,]   20   24
[4,]   28   32
```

- **Access to rows, columns or submatrices:**

R Console

```
> x <- matrix( nrow=4, ncol=2,data=c(1,2,3,4,5,6,7,8),byrow=TRUE)
> x[2,]
[1] 3 4
> x[,2]
[1] 2 4 6 8
> x[,3]   #There is no third column
Error in x[, 3] : subscript out of bounds
> x[3:4, 1:2]
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```

- **Inverse of a matrix:**

  solve( ) finds the inverse of a positive definite matrix

R Console

```
> y<- matrix( nrow=2, ncol=2, byrow=T,data=c(83,100,100,110))
> y
     [,1] [,2]
[1,]   83  100
[2,]  100  110
> solve(y)
           [,1]       [,2]
[1,] -0.1264368  0.1149425
[2,]  0.1149425 -0.0954023
```

- 5x = 10, what's x?

```
> solve(5,10)
[1] 2
```

- Let's see two variables examples:
- 3x + 2y = 8 , x + y =2; What's x and y?

R Console

```
> a <- matrix(c(3,1,2,1),nrow=2,ncol=2)
> b <- matrix(c(8,2),nrow=2,ncol=1)
> solve(a,b)
     [,1]
[1,]    4
[2,]   -2
>
```

- **Eigen Values and Eigen Vectors:**

  eigen() finds the eigen values and eigen vectors of a positive definite matrix

R Console

```
> y
     [,1] [,2]
[1,]   83  100
[2,]  100  110
> eigen(y)
$values
[1] 197.407136  -4.407136

$vectors
           [,1]       [,2]
[1,] 0.6581085 -0.7529231
[2,] 0.7529231  0.6581085
```

- **Changing the matrix's elements**

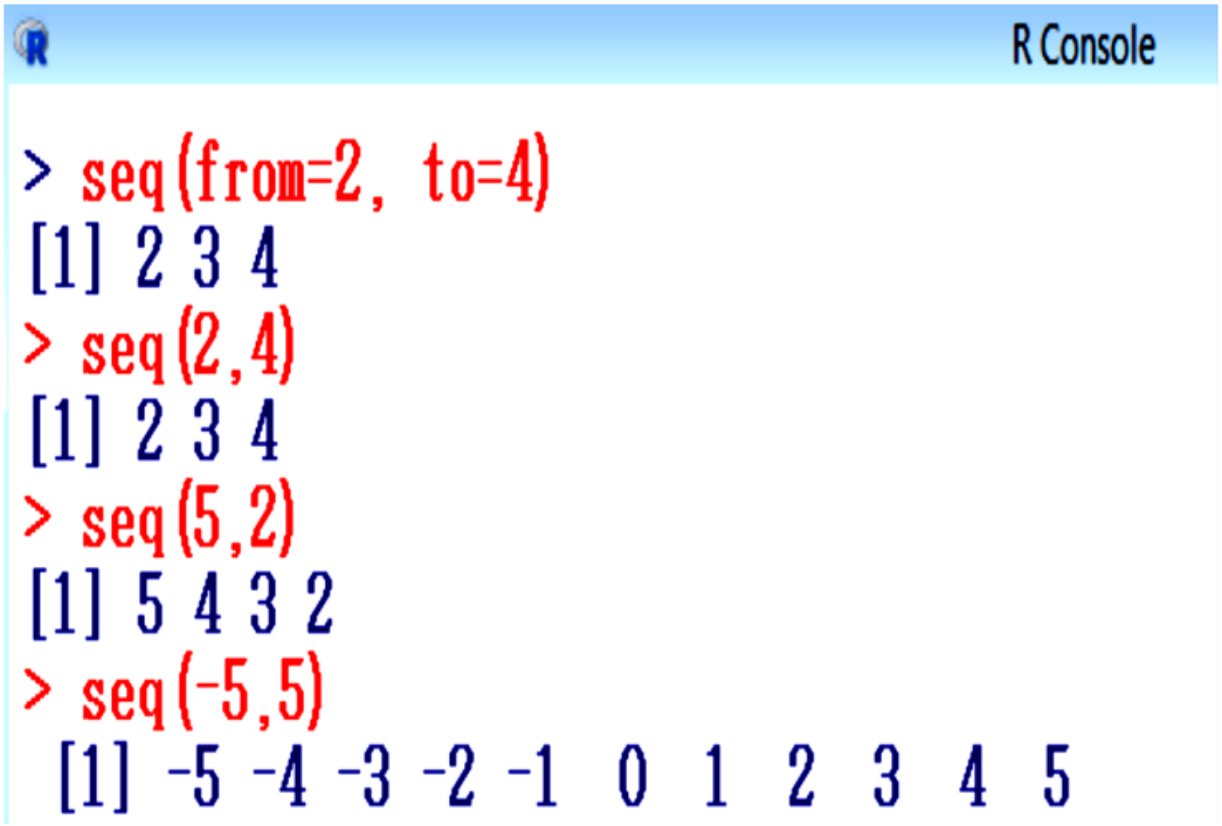
R Console

```
> mymatrix <- matrix(1:6,2,3,byrow=T)
> mymatrix
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> mymatrix <- rbind(mymatrix, 7:9)
> mymatrix
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> mymatrix <- cbind(mymatrix,6:8)
> mymatrix
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    6
[2,]    4    5    6    7
[3,]    7    8    9    8
```

R Console

```
> mymatrix[3,]<- 1:4          #changing an entire row
> mymatrix
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    6
[2,]    4    5    6    7
[3,]    1    2    3    4
> mymatrix[,4] <- 7:9         #changing the entire column
> mymatrix
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    7
[2,]    4    5    6    8
[3,]    1    2    3    9
> mymatrix <- mymatrix[-2,]   #deleting one row
> mymatrix
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    7
[2,]    1    2    3    9
> mymatrix <- mymatrix[,-4]   #deliting one column
> mymatrix
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

- **Sequences**
- A sequence is a set of related numbers, events, movements, or items that follow each other in a particular order. The regular sequences can be generated in R.



- **Sequence with constant increment and decrement:**

R Console

```
> seq(from=10, to=20, by=2)
[1] 10 12 14 16 18 20
> seq(from=20, to=10, by=-2)
[1] 20 18 16 14 12 10
> seq(3,-2, by=-0.5)
 [1]  3.0  2.5  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5 -2.0
> seq(to=10, length=10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(from=10, length=10)
 [1] 10 11 12 13 14 15 16 17 18 19
```

R Console

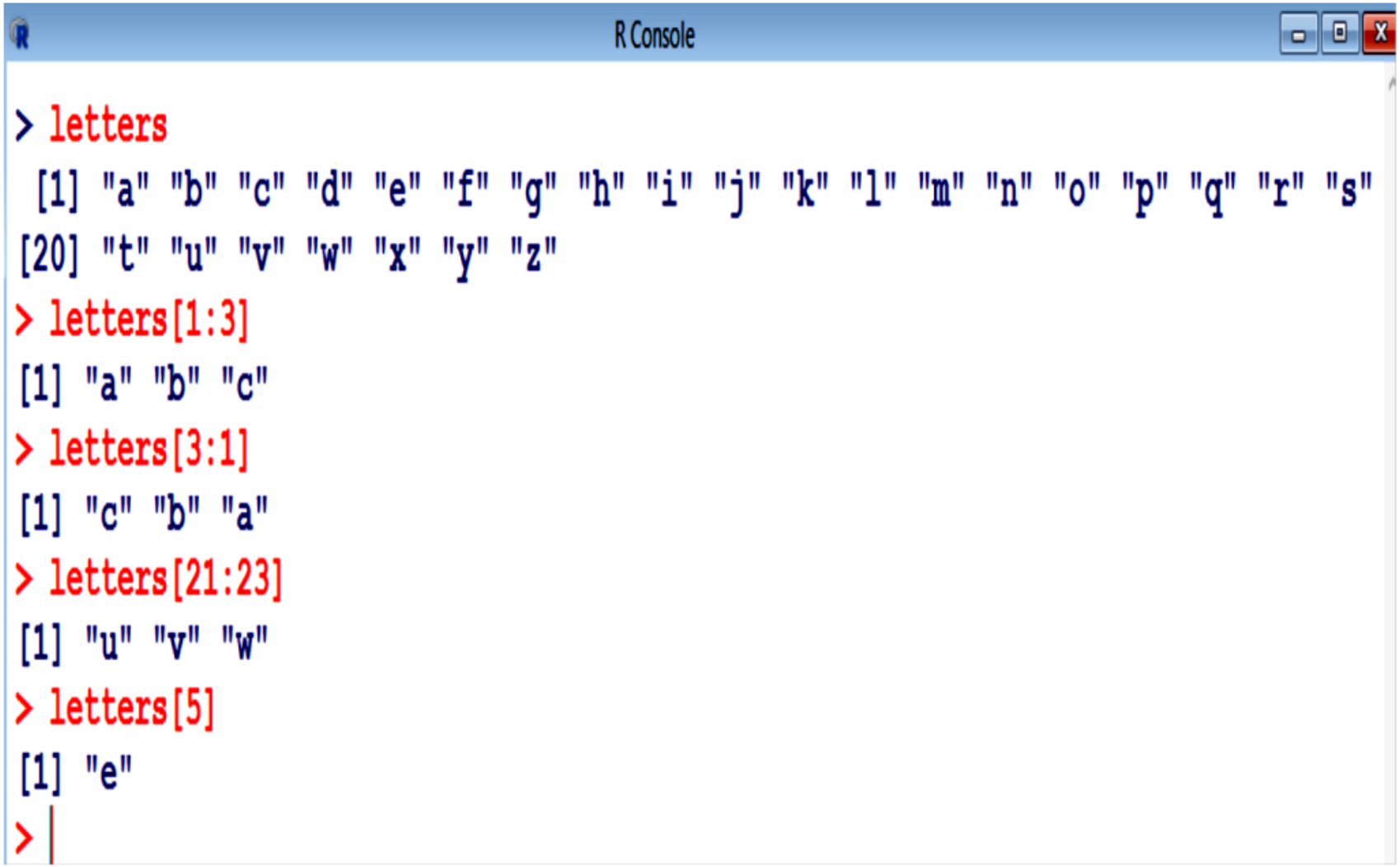
```
> seq(from=10, length=10, by=-2)
 [1] 10  8  6  4  2  0 -2 -4 -6 -8
> seq(from=10, length=5, by=-.2)
[1] 10.0  9.8  9.6  9.4  9.2
> x<-2
> seq(1, x, x/10)
[1] 1.0 1.2 1.4 1.6 1.8 2.0
> x<-50
> seq(0, x, x/10)
 [1]  0  5 10 15 20 25 30 35 40 45 50
```

- **Generating sequences of dates**

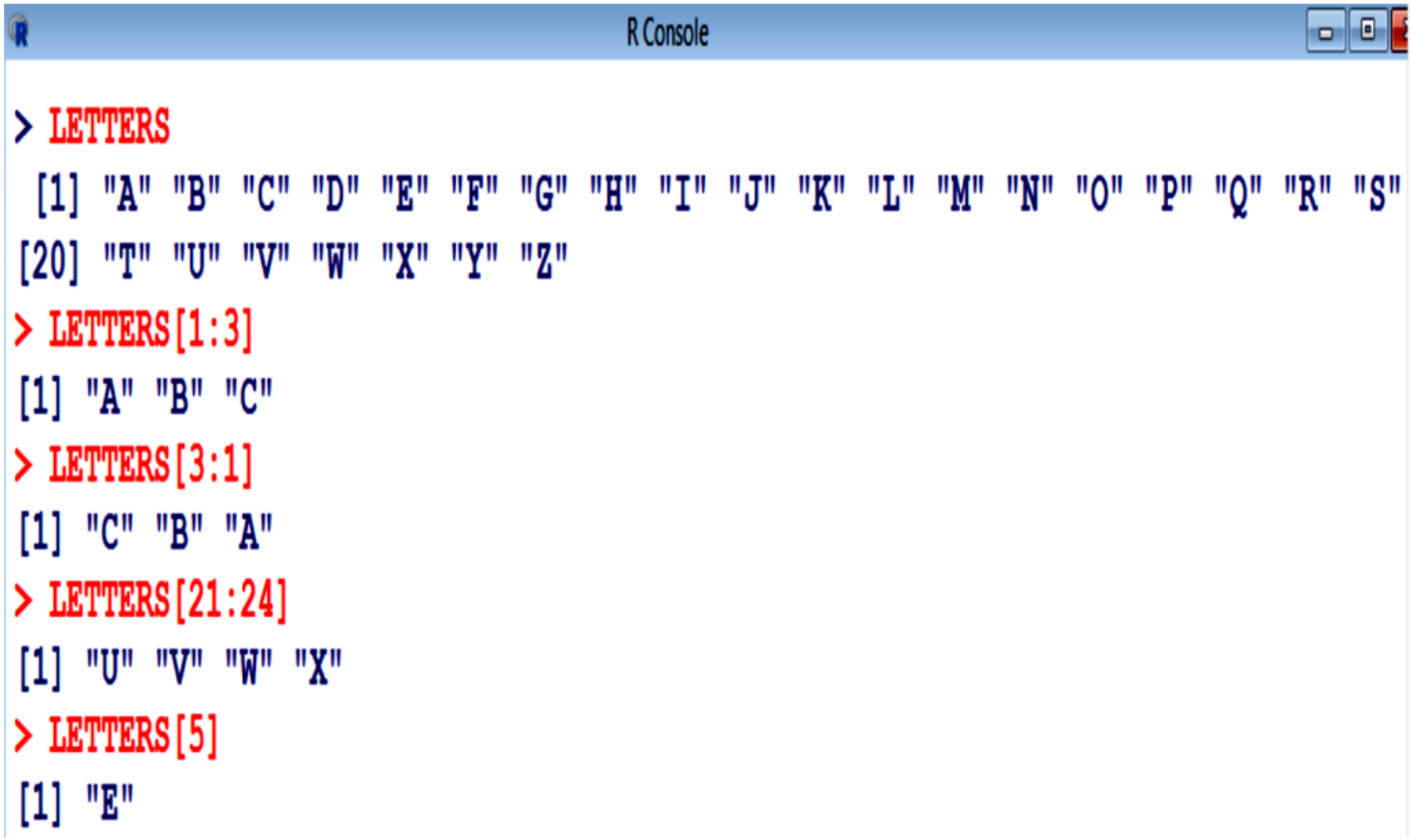
R Console

```
> seq(as.Date("2010-01-01"), as.Date("2017-01-01"),by="years")
[1] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01" "2014-01-01"
[6] "2015-01-01" "2016-01-01" "2017-01-01"
> seq(as.Date("2010-01-01"), as.Date("2017-11-30"),by="years")
[1] "2010-01-01" "2011-01-01" "2012-01-01" "2013-01-01" "2014-01-01"
[6] "2015-01-01" "2016-01-01" "2017-01-01"
> seq(as.Date("2017-11-30"), by = "days",length=6)
[1] "2017-11-30" "2017-12-01" "2017-12-02" "2017-12-03" "2017-12-04"
[6] "2017-12-05"
> seq(as.Date("2017-11-30"), by = "months",length=6)
[1] "2017-11-30" "2017-12-30" "2018-01-30" "2018-03-02" "2018-03-30"
[6] "2018-04-30"
> seq(as.Date("2017-11-30"), by = "years",length=6)
[1] "2017-11-30" "2018-11-30" "2019-11-30" "2020-11-30" "2021-11-30"
[6] "2022-11-30"
```

- **Generating sequences of letters-lower case alphabets**



- sequence of uppercase alphabets



- **Repeats-the command rep**

R Console

```
> rep(5.5, times=10)
 [1] 5.5 5.5 5.5 5.5 5.5 5.5 5.5 5.5 5.5 5.5
> rep(1:5,2)
 [1] 1 2 3 4 5 1 2 3 4 5
> rep(1:3,times=3)
[1] 1 2 3 1 2 3 1 2 3
> rep(1:3,each=3)
[1] 1 1 1 2 2 2 3 3 3
> rep(1:4,each=2,times=3)
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
> rep(1:4, 2:5)
 [1] 1 1 2 2 2 3 3 3 3 4 4 4 4 4
> rep(1:4,seq(from=2, to=8, by=2))
 [1] 1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4
```

R Console

```
> x <- matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> rep(x,4)
 [1] 1 3 2 4 1 3 2 4 1 3 2 4 1 3 2 4
```

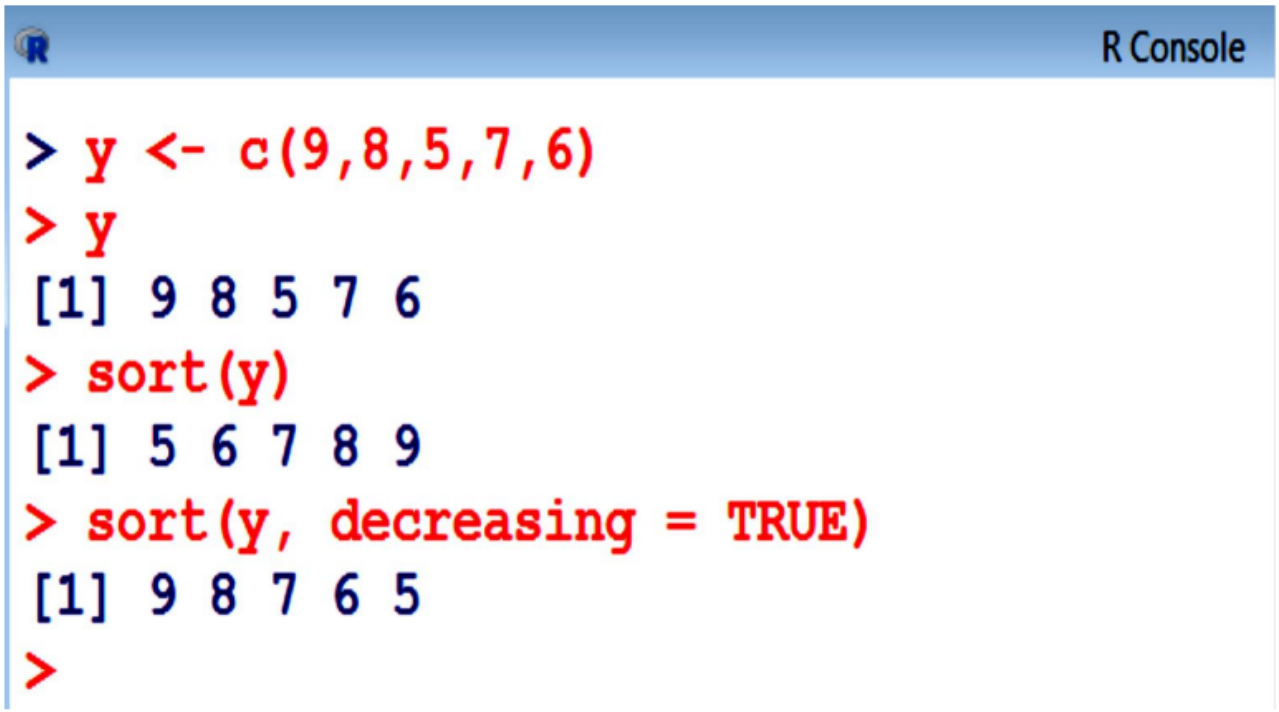
- **Repetition of characters**

R Console

```
> rep(c("a", "b", "c"), 2)
[1] "a" "b" "c" "a" "b" "c"
> rep(c("sas", "scope", "site"), 2)
[1] "sas"   "scope" "site"  "sas"   "scope" "site"
>
```

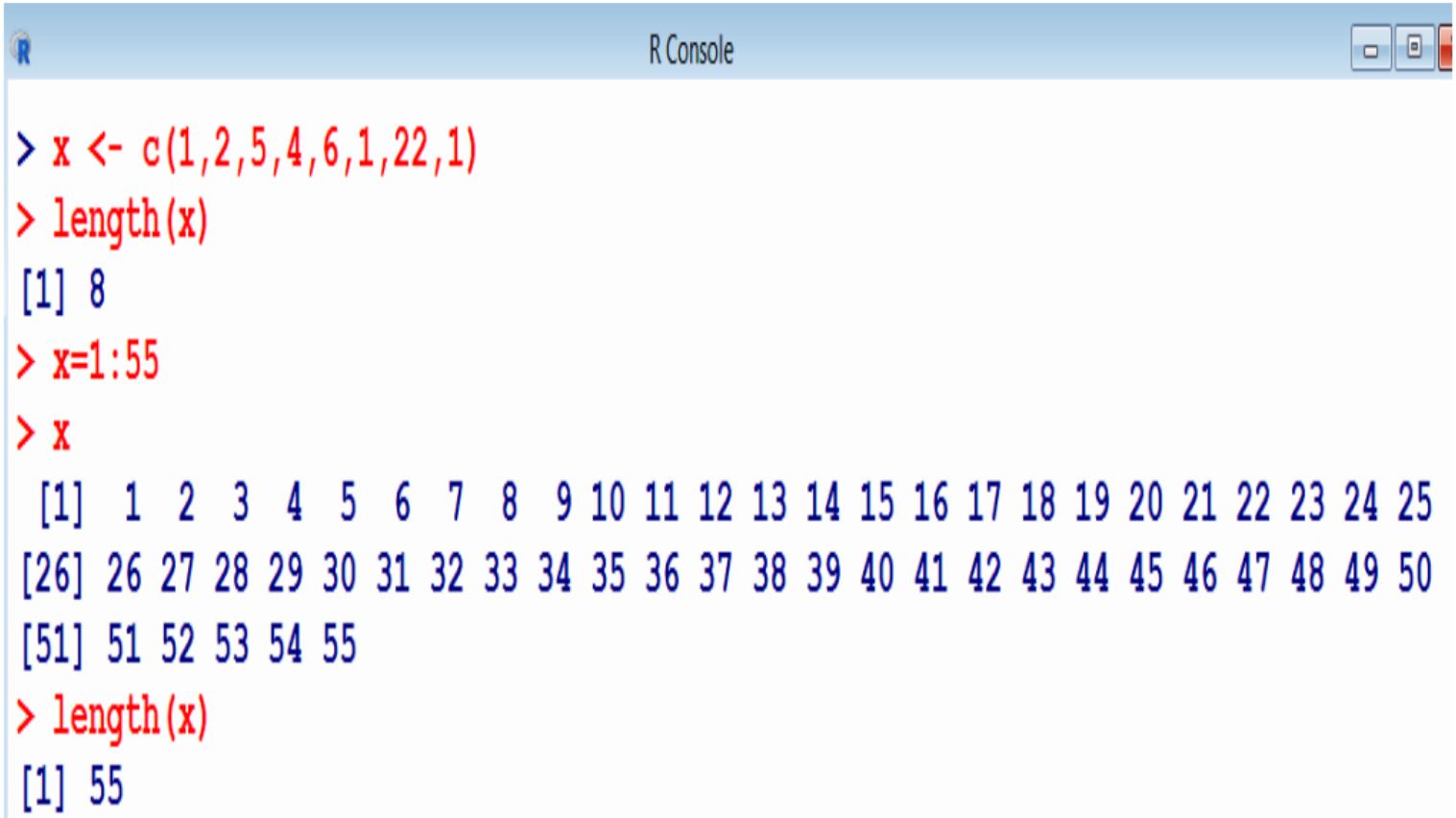
- Sorting

sort function sorts the values of a vector in ascending order (by default) or descending order.



- **Length**

length() function gets or sets the length of a vector (list) or other objects



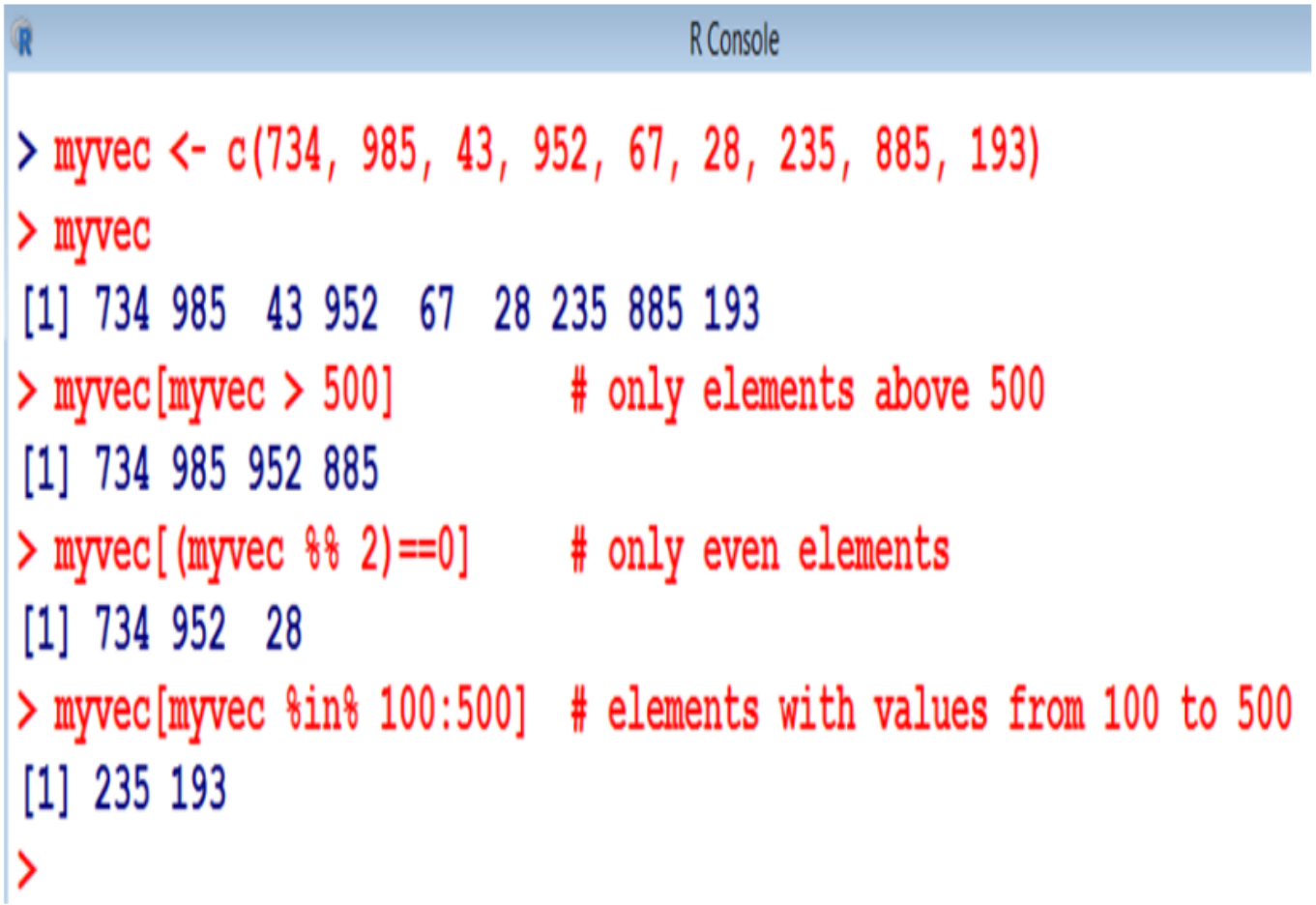
Data Structures in R

- **Extracting vector elements by the element index(es)**

R Console

```
> myvec <- c(734, 985, 43, 952, 67, 28, 235, 885, 193)
> myvec
[1] 734 985  43 952  67  28 235 885 193
> myvec[5]         # 5th element
[1] 67
> myvec[c(1,5,7)] # elements 1, 5 and 7
[1] 734  67 235
> myvec[-5]        # all but the 5th element
[1] 734 985  43 952  28 235 885 193
> myvec[-c(1,5,7)]# all but elements 1, 5 and 7
[1] 985  43 952  28 885 193
> myvec[4:6]       # elements 4 to 6
[1] 952  67  28
```

- Extracting vector elements by a logical expression



- set operations (union, intersection, asymmetric difference, equality and membership) on two vectors.
- Union() is not the same as concatenation c() because c() will duplicate values that are common to both vectors

R Console

```
> myvec1 <- c(3,6,7,8,12,23,94)
> myvec2 <- c(5,7,8,152,71,77)
> union(myvec1, myvec2)                 # set union
 [1]   3   6   7   8  12  23  94   5 152  71  77
> c(myvec1,myvec2)                      # diff b/w union() and c()
 [1]   3   6   7   8  12  23  94   5   7   8 152  71  77
> intersect(myvec1, myvec2)             # set intersection
[1] 7 8
> setdiff(myvec1, myvec2)               # set difference
[1]  3  6 12 23 94
>
```

• Missing data

R represents missing observations through the data value NA

We can detect missing values using is.na


```
> x <- NA          # assign NA to variable x
> is.na(x)         # is it missing?
[1] TRUE
```

- **Now try a vector to know if any value is missing?**

```
> x <- c(11, NA, 13)
> is.na(x)
[1] FALSE  TRUE FALSE
```

- **Logical Operators and Comparisons**

Operator	Executions
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Exactly equal to
!=	Not equal to
!	Negation (not)

- **TRUE and FALSE are reserved words denoting logical constants.**

Operator	Executions
`&, &&`	and
`|, ||`	or

Operator	Executions
`xor()`	either... or (exclusive)
`isTRUE(x)`	test if x is TRUE
`TRUE`	true
`FALSE`	false

R Console

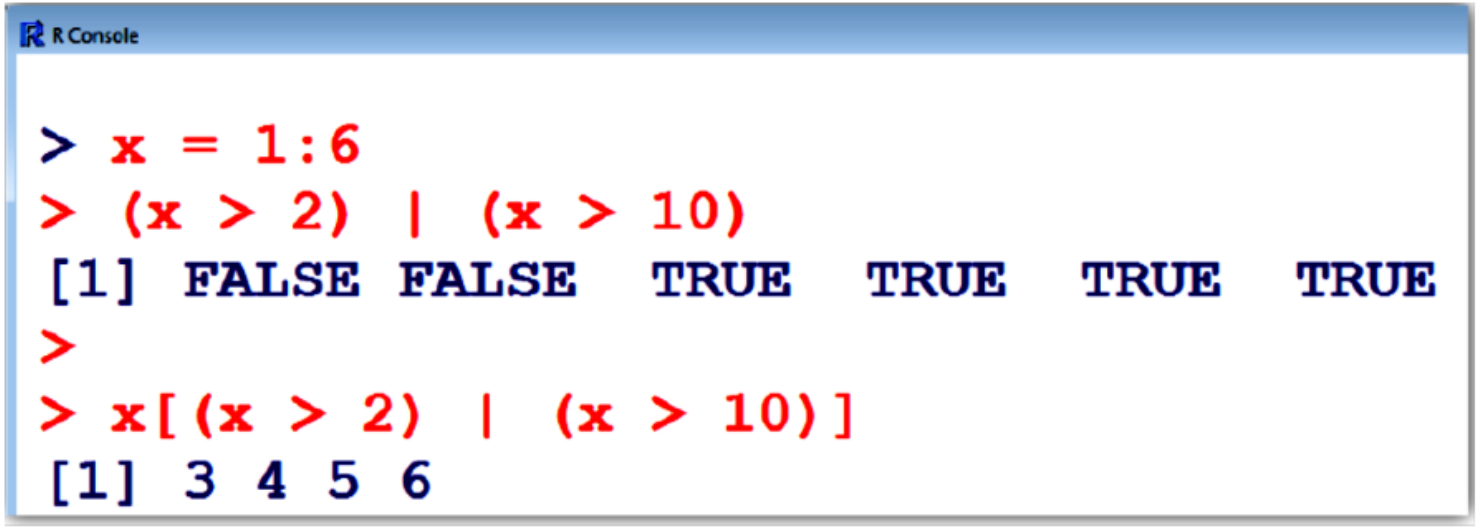
```
> 8 > 7
[1] TRUE
> 7 < 5
[1] FALSE
> isTRUE(8<6)    #Is 8 less than 6?
[1] FALSE
> isTRUE(8>6)    #Is 8 greater than 6?
[1] TRUE
```

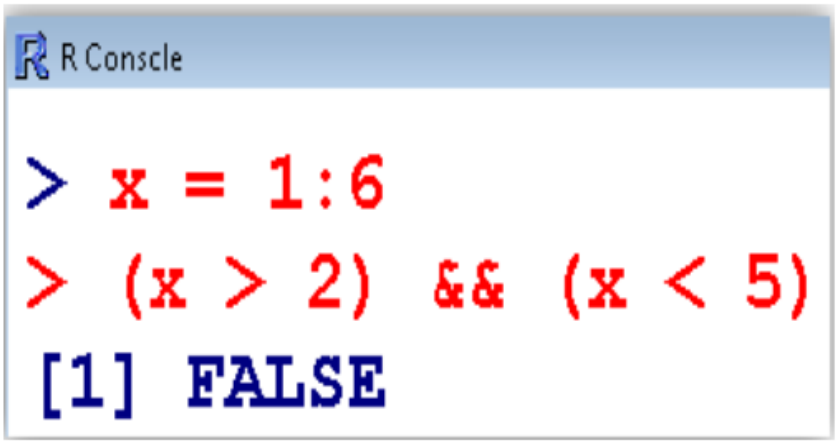
R Console

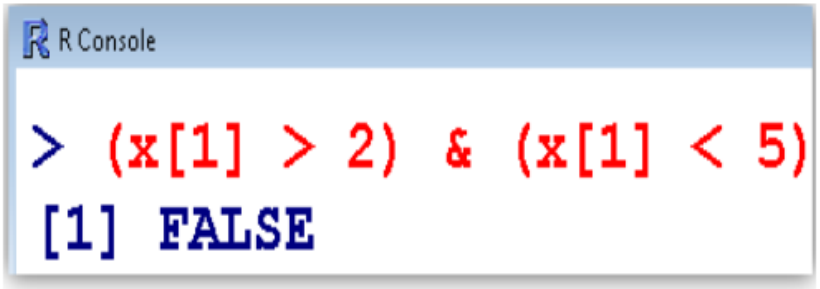
```
> x <- 5
> (x<10)&&(x > 2)      # && means AND
[1] TRUE
> (x < 10) || (x > 5) # || means OR
[1] TRUE
> (x > 10) || (x > 5)
[1] FALSE
```

R Console

```
> x = 10
> y = 20
> (x == 10) & (y == 20)  # == means exactly equal to
[1] TRUE
> (x == 10) & (y == 2)
[1] FALSE
> (x == 2) & (y == 3)
[1] FALSE
> (x == 1) & (y == 2)
[1] FALSE
> x = 1:6
> (x > 2) & (x < 5)
[1] FALSE FALSE  TRUE  TRUE FALSE FALSE
> x[(x > 2) & (x < 5)]
[1] 3 4
```

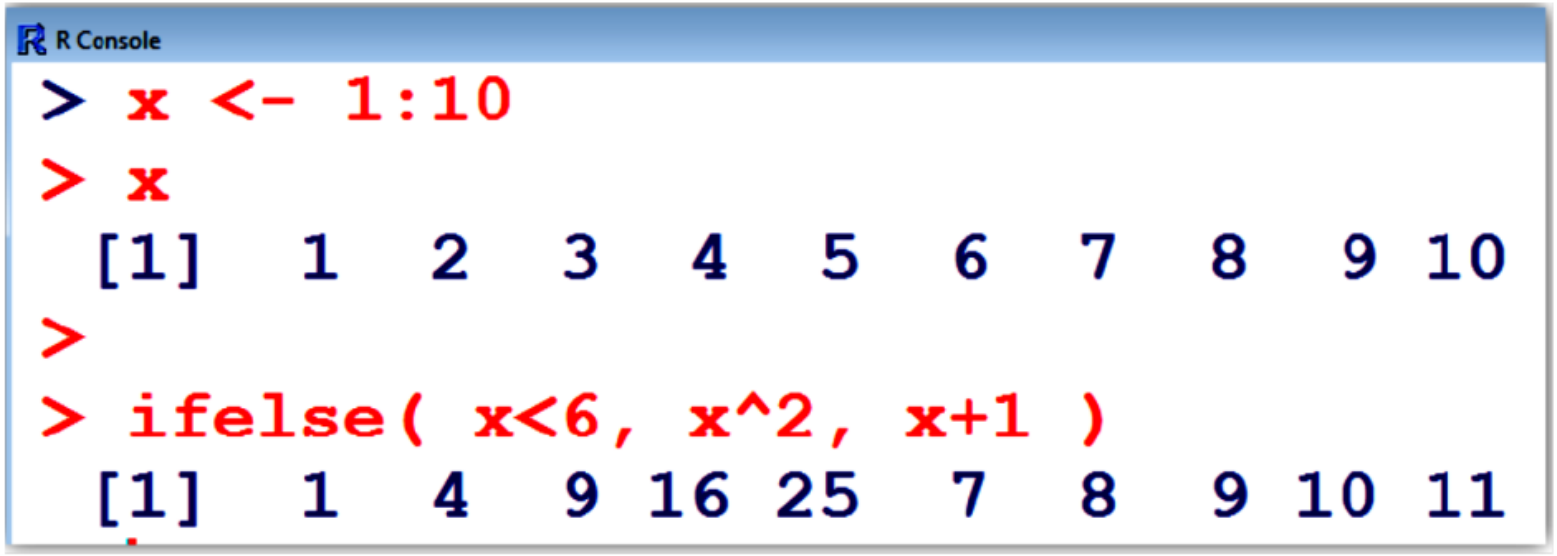

R Console
> x = 1:6
> (x > 2) | (x > 10)
[1] FALSE FALSE TRUE TRUE TRUE TRUE
>
> x[(x > 2) | (x > 10)]
[1] 3 4 5 6


R Conscle
> x = 1:6
> (x > 2) && (x < 5)
[1] FALSE


R Console
> (x[1] > 2) & (x[1] < 5)
[1] FALSE

- **Conditional execution**

  ifelse(test, yes, no)



- **Interpretation**

  If x < 6 (TRUE), then x = x2 (YES) .

  If x ≥ 6 (FALSE), then x = x + 1 (NO).

- The **for** loop
- **If the number of repetitions is known in advance a for() loop can be used.**

R Console

```
> for ( i in 1:5 ) { print( i^2 ) }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

R Console

```
> for ( i in c(2,4,6,7) ) { print( i^2 ) }
[1] 4
[1] 16
[1] 36
[1] 49
```

- **The while() loop**

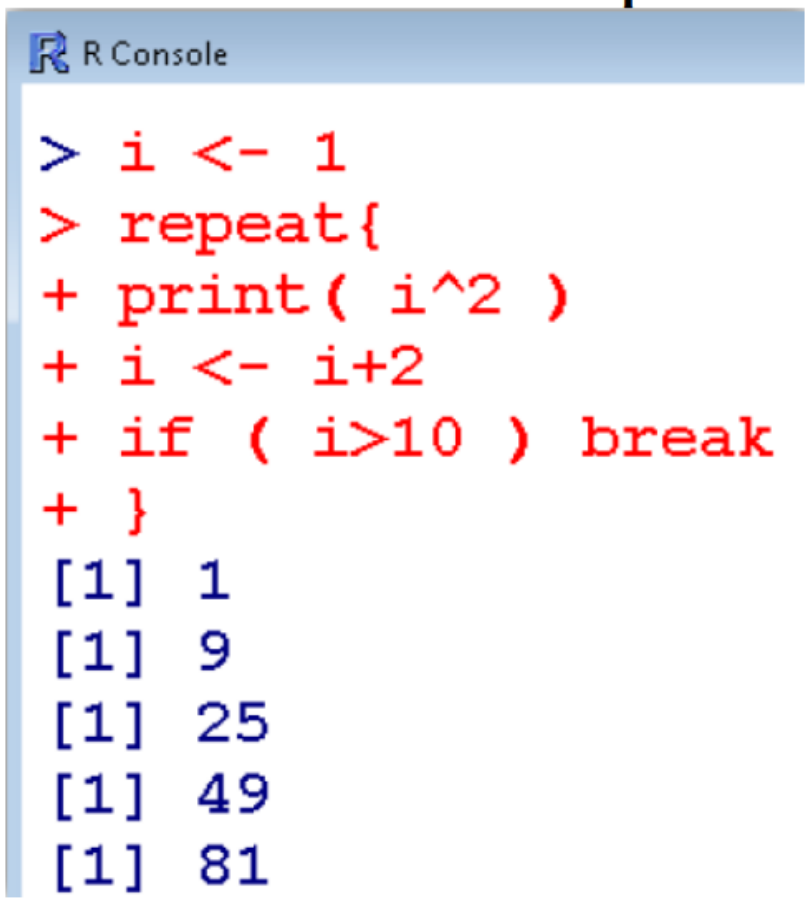
If the number of loops is not known in before, e.g. when an iterative algorithm to maximize a likelihood function is used, one can use a while() loop.

R Console

```
> i <- 1
> while (i<5) {
+ print(i^2)
+ i <- i+2
+ }
[1] 1
[1] 9
```

- The repeat loop

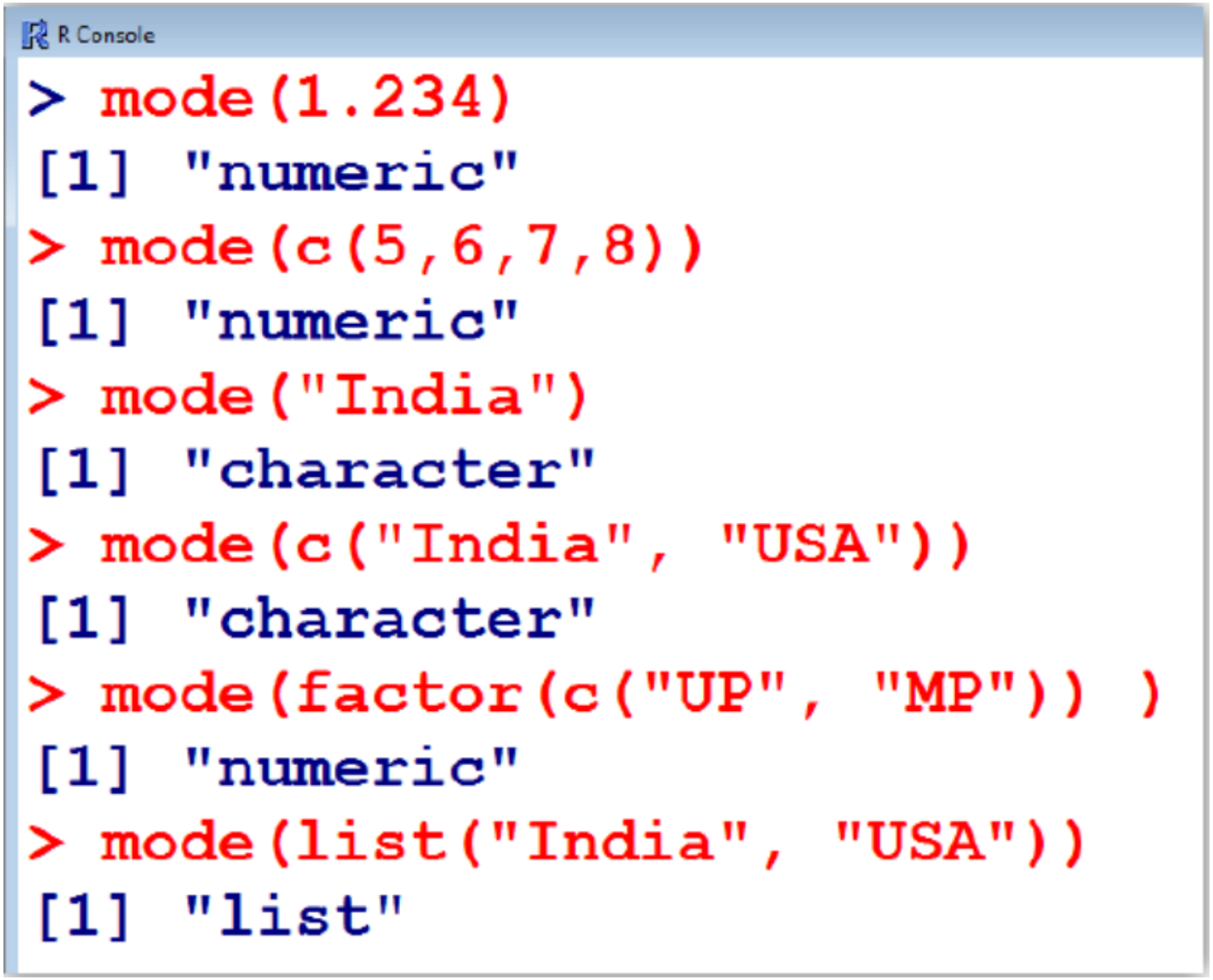
The repeat loop doesn't test any condition — in contrast to the while() loop — before entering the loop and also not during the execution of the loop.



Lists

Mode:

Object	Example	Mode
Number	`1.234`	numeric
Vector of numbers	`c(5, 6, 7, 8)`	numeric
Character string	`"India"`	character
Vector of character strings	`c("India", "USA")`	character
Factor	`factor(c("UP", "MP"))`	numeric
List	`list("India", "USA")`	list
Data frame	`data.frame(x=1:2, y=c("India", "USA"))`	list
Function	`print`	function


R Console
> mode(1.234)
[1] "numeric"
> mode(c(5,6,7,8))
[1] "numeric"
> mode("India")
[1] "character"
> mode(c("India", "USA"))
[1] "character"
> mode(factor(c("UP", "MP")))
[1] "numeric"
> mode(list("India", "USA"))
[1] "list"

Data Frames

- In a data frame, we can combine variables of equal length,with each row in the data frame containing observations on the same unit.
- Data frames contain complete data sets that are mostly created with other programs (spreadsheet-files, software SPSS-files, Excel-files etc.).
- Variables in a data frame may be numeric (numbers) or categorical (characters or factors).
- Package "MASS" describes functions and datasets to support Venables and Ripley, ``Modern Applied Statistics with S'' (4th edition 2002)

```
> library(MASS)
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
⋮	⋮	⋮	⋮	⋮	⋮
Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H

```
> library(MASS)
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
⋮	⋮	⋮	⋮	⋮	⋮
Rubens	18	13	17	17	G
Teniers	15	12	13	6	G
Van Dyck	15	10	17	13	G
Bourdon	10	8	8	4	H
Le Brun	16	16	8	16	H

R Console

```
> rownames(painters)
 [1] "Da Udine"       "Da Vinci"        "Del Piombo"
 [4] "Del Sarto"      "Fr. Penni"       "Guilio Romano"
 [7] "Michelangelo"   "Perino del Vaga" "Perugino"
[10] "Raphael"        "F. Zucarro"      "Fr. Salviata"
[13] "Parmigiano"     "Primaticcio"     "T. Zucarro"
[16] "Volterra"       "Barocci"         "Cortona"
[19] "Josepin"        "L. Jordaens"     "Testa"
[22] "Vanius"         "Bassano"         "Bellini"
[25] "Giorgione"      "Murillo"         "Palma Giovane"
[28] "Palma Vecchio"  "Pordenone"       "Tintoretto"
[31] "Titian"         "Veronese"        "Albani"
[34] "Caravaggio"     "Corregio"        "Domenichino"
[37] "Guercino"       "Lanfranco"       "The Carraci"
[40] "Durer"          "Holbein"         "Pourbus"
[43] "Van Leyden"     "Diepenbeck"      "J. Jordaens"
[46] "Otho Venius"    "Rembrandt"       "Rubens"
[49] "Teniers"        "Van Dyck"        "Bourdon"
```

R Console

```
> colnames(painters)
[1] "Composition" "Drawing"     "Colour"      "Expression"  "School"
```

R Console

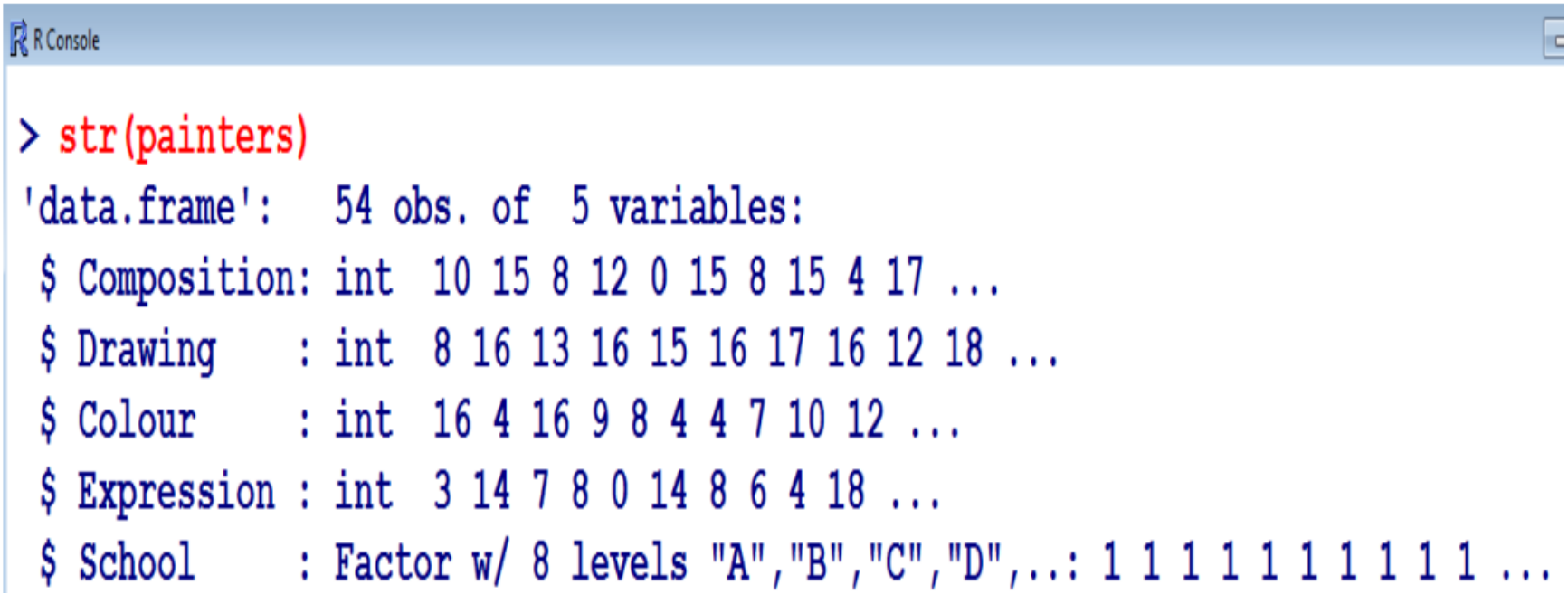
```
> x  <- 1:16
> y  <- matrix(x, nrow=4, ncol=4)
> z  <- letters[1:16]
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
>
> y
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
>
> z
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
[14] "n" "o" "p"
```

R Console

```
> datafr  <- data.frame(x, y, z)
> datafr
    x X1 X2 X3 X4 z
1   1  1  5  9 13 a
2   2  2  6 10 14 b
3   3  3  7 11 15 c
4   4  4  8 12 16 d
5   5  1  5  9 13 e
6   6  2  6 10 14 f
7   7  3  7 11 15 g
8   8  4  8 12 16 h
9   9  1  5  9 13 i
10 10  2  6 10 14 j
11 11  3  7 11 15 k
12 12  4  8 12 16 l
13 13  1  5  9 13 m
14 14  2  6 10 14 n
15 15  3  7 11 15 o
16 16  4  8 12 16 p
```

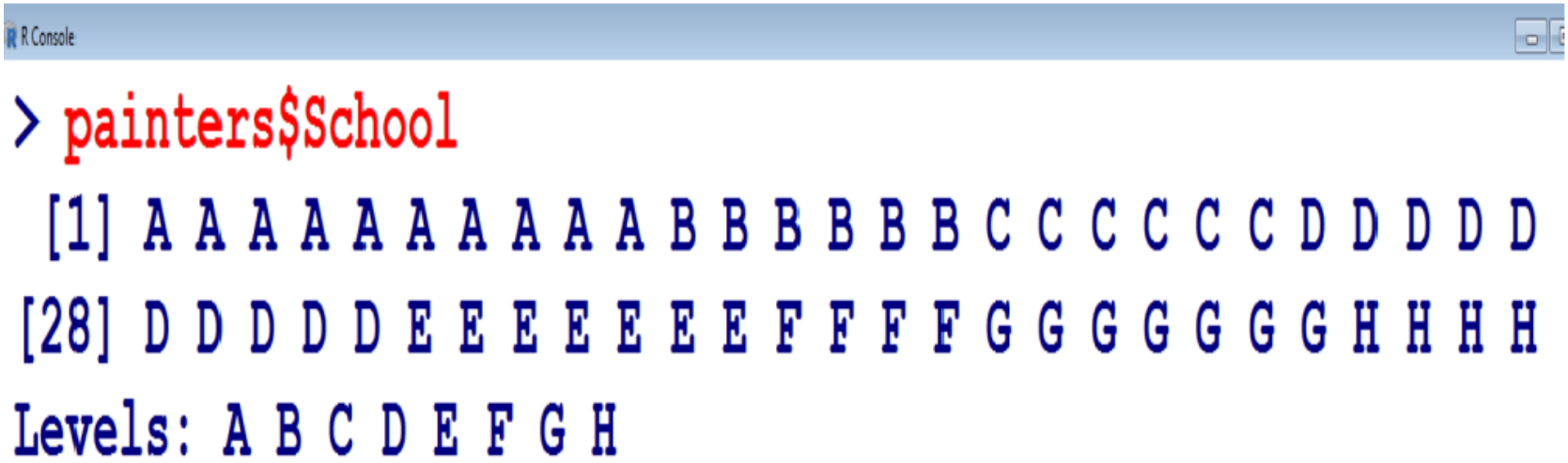
Structure of the data

The result of str gives the dimension as well as the name and type of each variable.



Extract a variable from data frame using $

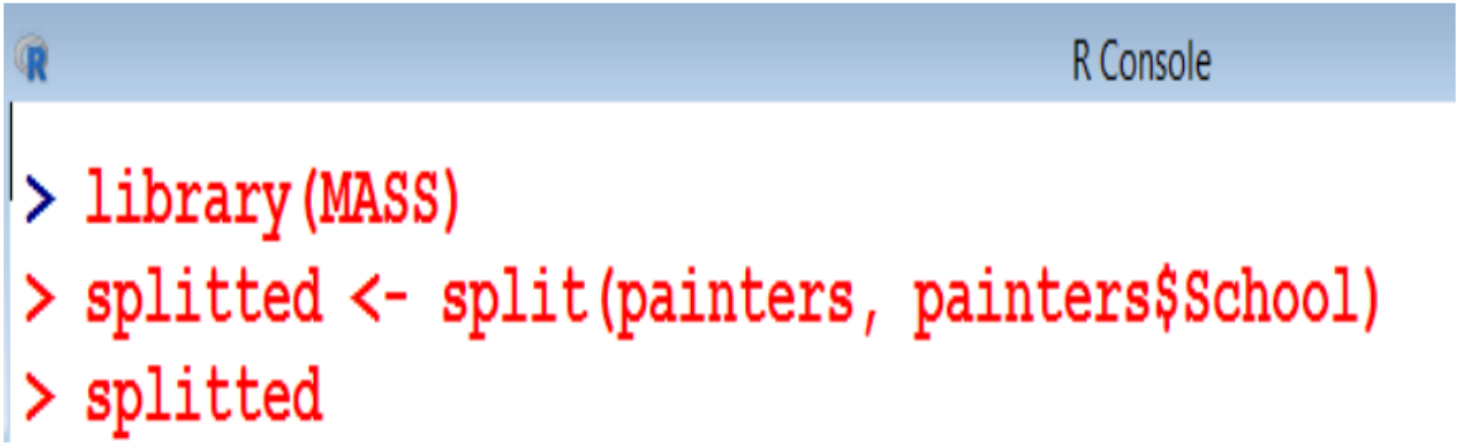
Suppose we want to extract information on variable School from the data set painters.



- Subsets of a data frame can be obtained with subset() or with the second equivalent command:

```
> library(MASS)
> subset(painters, School=='F')
           Composition Drawing Colour Expression School
Durer                8      10     10          8      F
Holbein              9      10     16         13      F
Pourbus              4      15      6          6      F
Van Leyden           8       6      6          4      F
```

- **The command split partitions the data set by values of a specific variable.This should preferably be a factor variable.**



- **Importing CSV and Tabular Data Files**

We can change the current working directory as follows:

- setwd("<location of the dataset>")

- Example

```
>setwd("C:\\Users\\admin\\Desktop\\")
>data=read.csv("stud.csv")
```

- Comma-separated values (CSV) files
- Data files have many formats and accordingly we have options for loading them.

```
>data=read.csv("C:\\Users\\admin\\Desktop\\Mokesh
\\stud.csv")
```

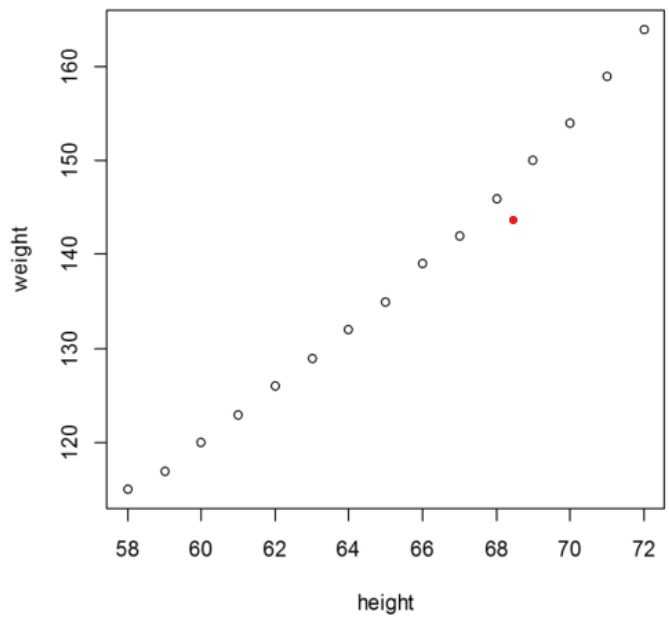
Or

```
>data=read.csv("C:/Users/admin/Desktop/Mokesh/stud
.csv")
```

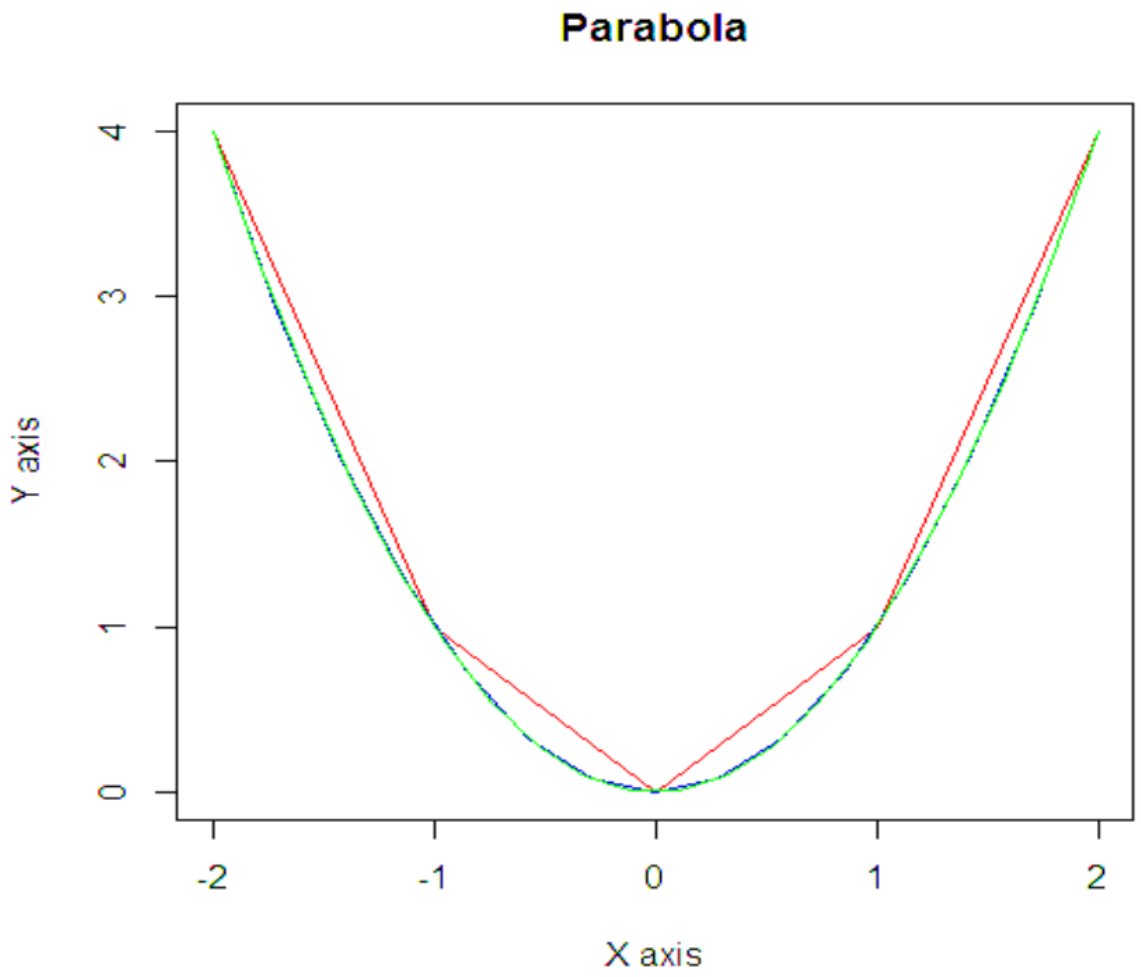
Graphics on R

- A simple plot plot(X) has each element of a discrete variable X ploted on the y-axis and the element's index on the x-axis

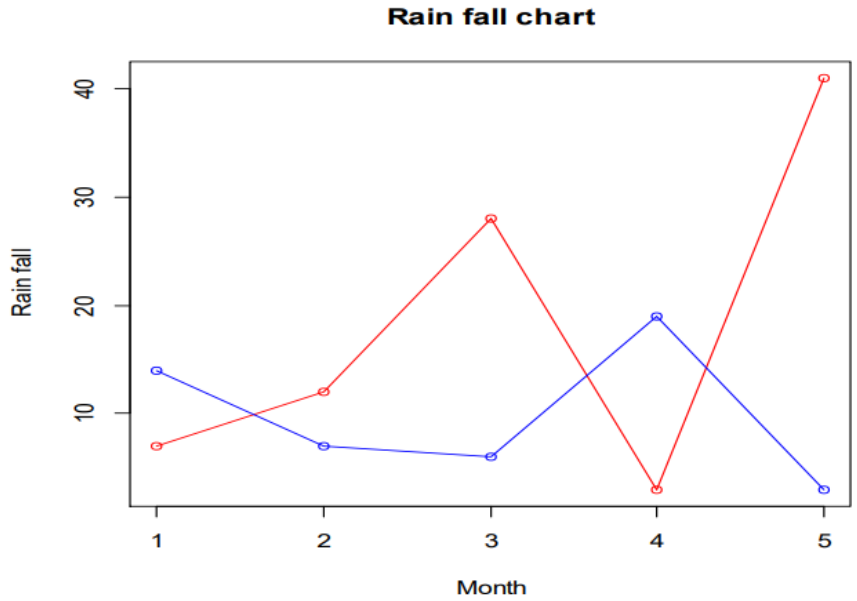
```
> # simple plot
> women   #inbuilt data set
   height weight
1      58    115
2      59    117
3      60    120
4      61    123
5      62    126
6      63    129
7      64    132
8      65    135
9      66    139
10     67    142
11     68    146
12     69    150
13     70    154
14     71    159
15     72    164
> plot(women)
```



```
> # function plot
> x = seq(-2,2)
> y = x^2
> # edgy graph!
> plot(x,y,type="l",xlab="X axis",ylab="Y axis",main="Parabola", col = "red")
> # better
> sp <- spline(x, y) # spline interpolation of data points
> lines(sp, col = "blue")
> # much better
> sp <- spline(x, y,n=20) # interpolation at n points spanning [xmin, xmax]
> lines(sp, col = "green")
```

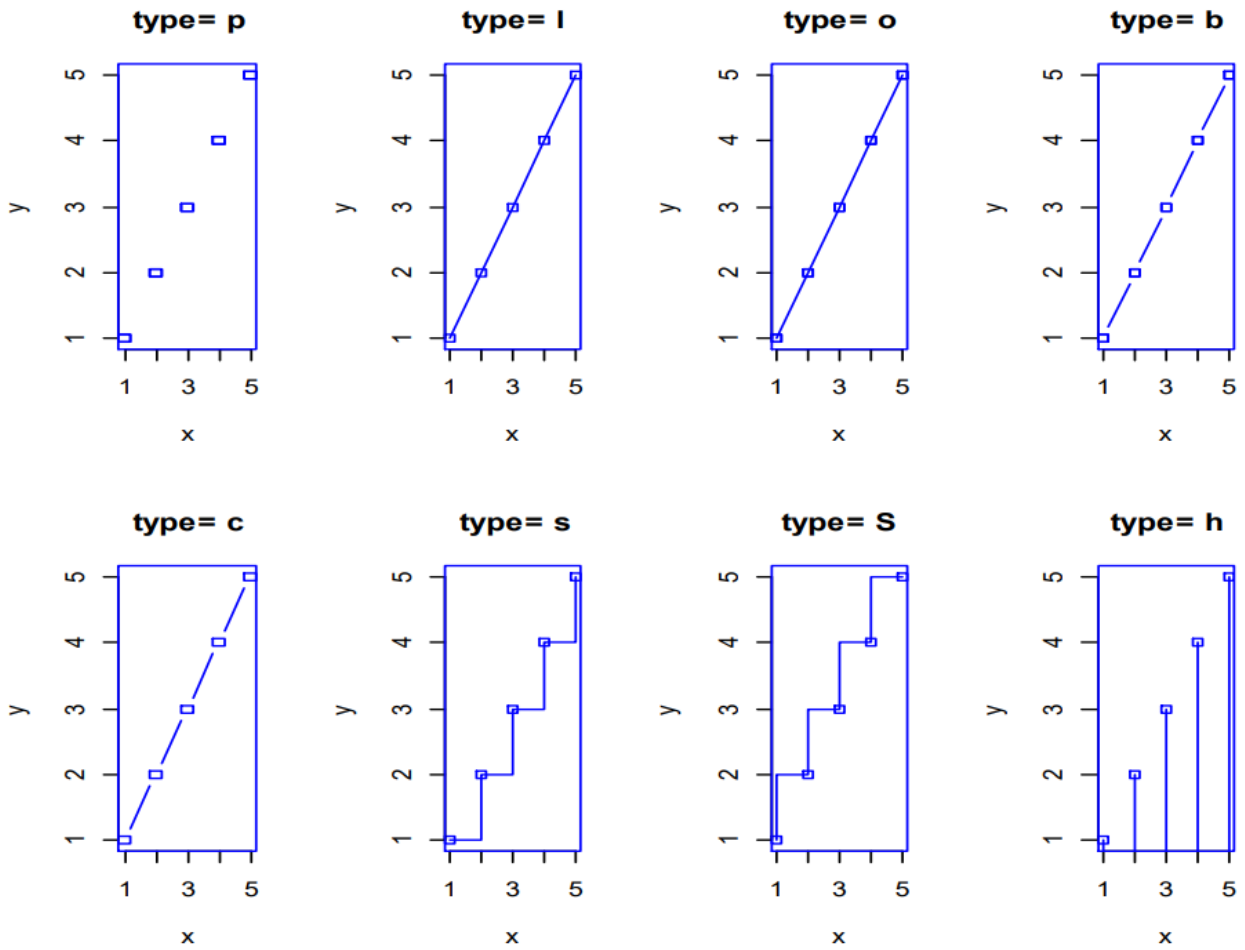

Parabola
Y axis
X axis
0
1
2
3
4
-2
-1
0
1
2

```
>v <- c(7,12,28,3,41)
>t <- c(14,7,6,19,3)
> plot(v,type = "o", col = "red", xlab = "Month", ylab =
"Rain  fall",main = "Rain fall chart")
>lines(t, type = "o", col = "blue")
```



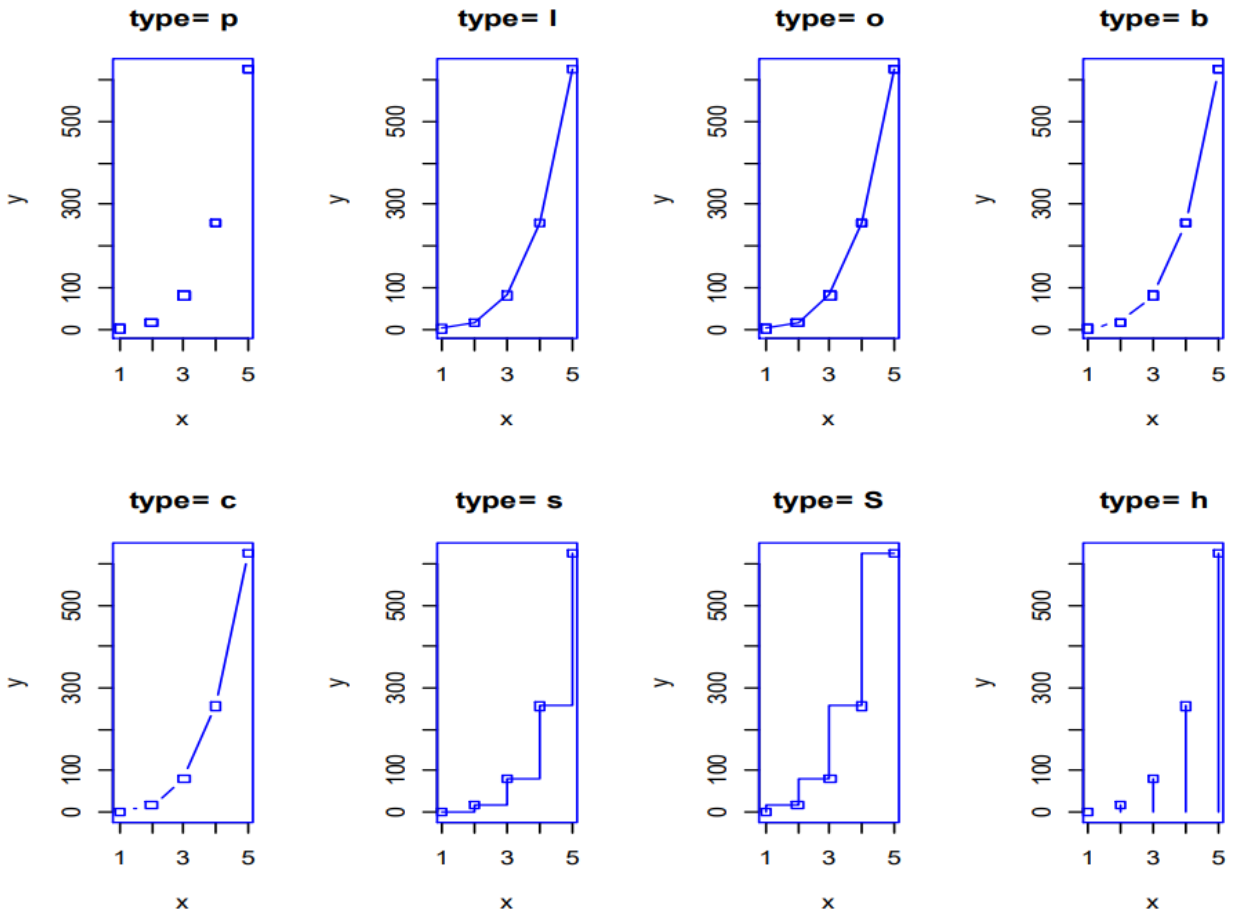
- **Line chart**
- A line chart is a simple plot with consecutive plots connected by lines

```
> x <- c(1:5)
> y <- x # create some data
> par(pch=22, col="blue") # plotting symbol and color
> par(mfrow=c(2,4)) # all plots on one page
> opts = c("p","l","o","b","c","s","S","h")
> for(i in 1:length(opts))
+ {
+ heading = paste("type=",opts[i])
+ plot(x, y, main=heading)
+ lines(x, y, type=opts[i])
+ }
```


type= p
type= l
type= o
type= b
type= c
type= s
type= S
type= h
y
x
1
2
3
4
5

R Console

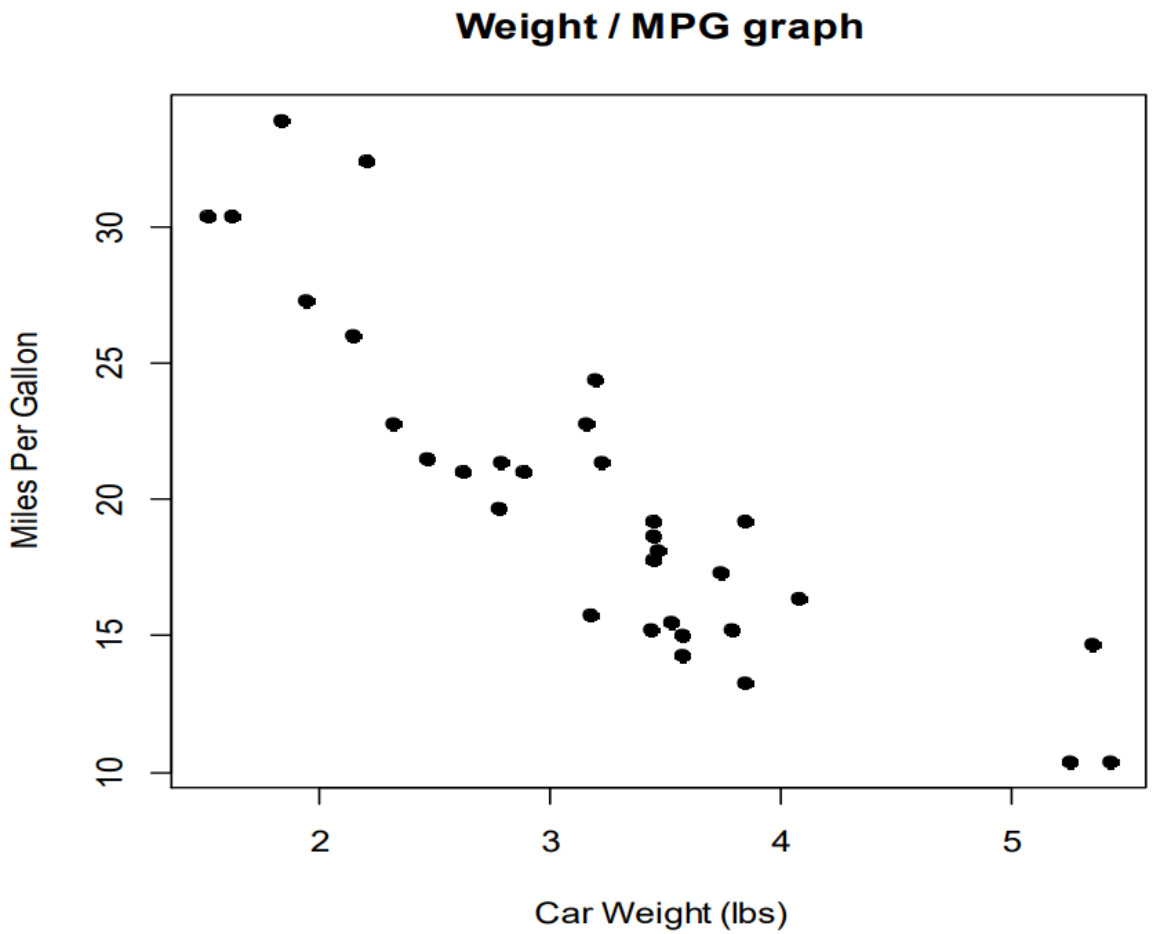
```
> x <- c(1:5)
> y <- x^4 # create some data
> par(pch=22, col="blue") # plotting symbol and color
> par(mfrow=c(2,4)) # all plots on one page
> opts = c("p","l","o","b","c","s","S","h")
> for(i in 1:length(opts))
+ {
+ heading = paste("type=",opts[i])
+ plot(x, y, main=heading)
+ lines(x, y, type=opts[i])
+ }
```


type= p
type= l
type= o
type= b
type= c
type= s
type= S
type= h
y
x
0
100
300
500
1
3
5

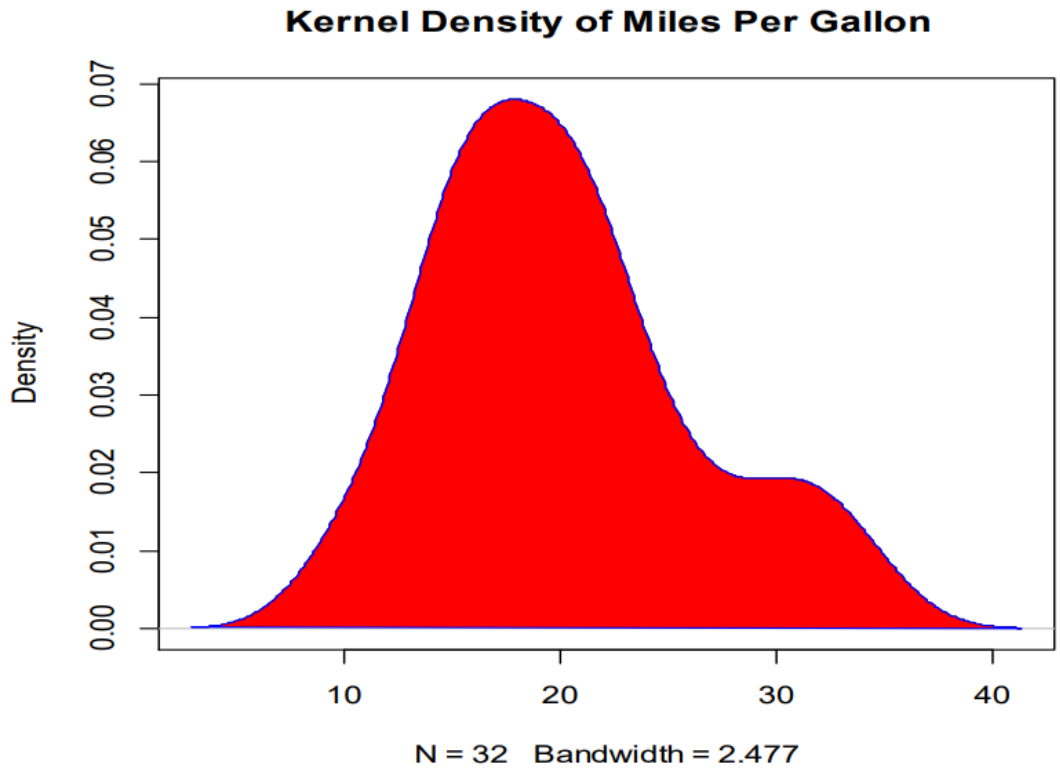
- **Scatterplot**

A scatterplot plot(X, Y) has each element of a variable Y ploted on the y-axis and the corresponding element for variable X on the x-axis

```
# scatterplot
>attach(mtcars)
>plot(wt, mpg, main="Weight / MPG graph",
xlab="Car Weight (lbs)", ylab="Miles Per Gallon",
pch=19)
```


Weight / MPG graph
Miles Per Gallon
10
15
20
25
30
2
3
4
5
Car Weight (lbs)

- **Kernel density plots**
- Kernel density plots nicely visualize the shape of a distribution. They can be better than histograms, even with normal curves because histograms are strongly affected by the number of bins used and by outliers.
- **# Kernel density plot**
- **>d <- density(mtcars$mpg) # kernel density estimates**
- **>plot(d)**
- **# Filled density plot**
- **>d <- density(mtcars$mpg)**
- **>plot(d, main="Kernel Density of Miles Per Gallon")**
- **>polygon(d, col="red", border="blue")**


Kernel Density of Miles Per Gallon
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
10
20
30
40
N = 32 Bandwidth = 2.477

- **boxplot(X) is a plot that, if X is a vector, the vector elements are the heights of the bars in the plot, if X is a matrix, the matrix columns are the heights of the bars in the plot, stacked after the first bar (column)**

- **If the argument beside=TRUE, then the values in each column are juxtaposed, not stacked.**

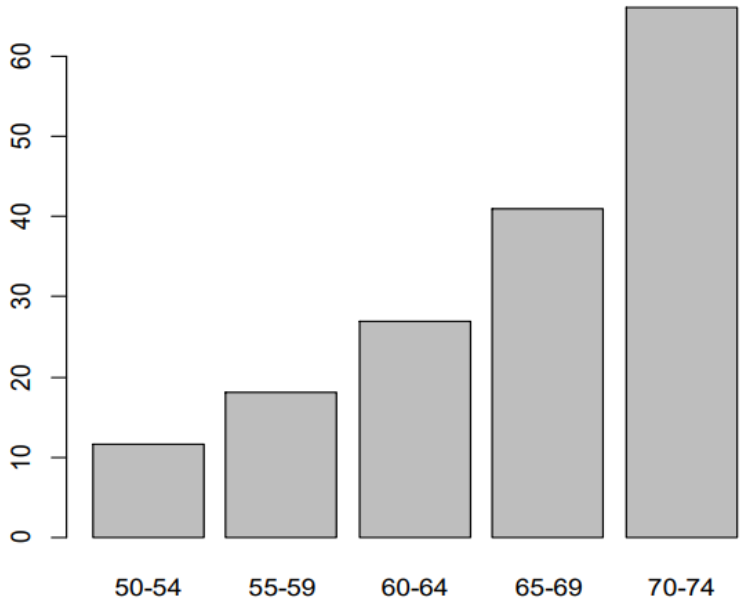
- **The argument horiz=TRUE creates an horizontal barplot.**

R Console

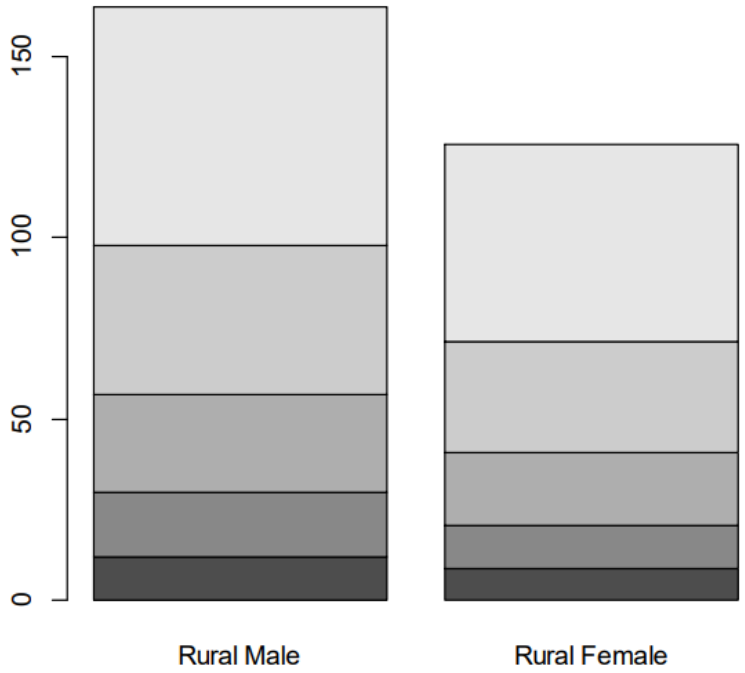
```
> VADeaths
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
> class(VADeaths)
[1] "matrix"
> dimnames(VADeaths)
[[1]]
[1] "50-54" "55-59" "60-64" "65-69" "70-74"

[[2]]
[1] "Rural Male"   "Rural Female" "Urban Male"   "Urban Female"
```

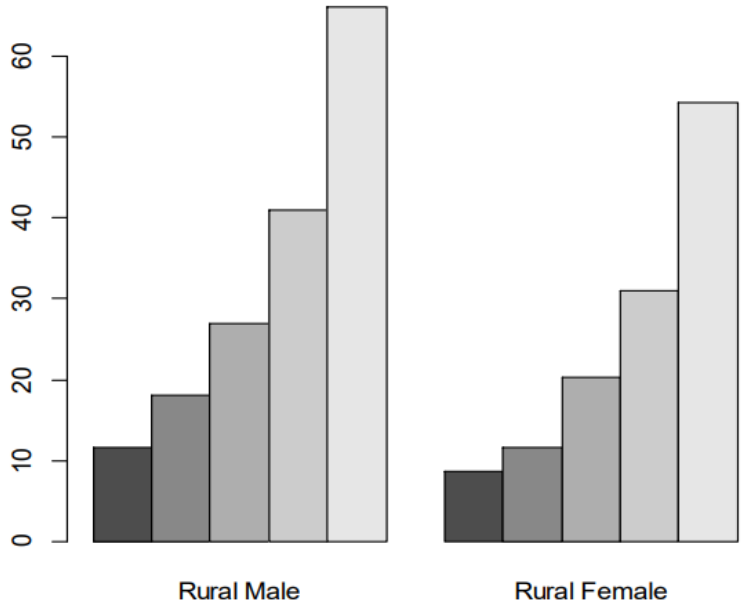
- > simple barplot
- > barplot (VADeaths[,"Rural Male"])



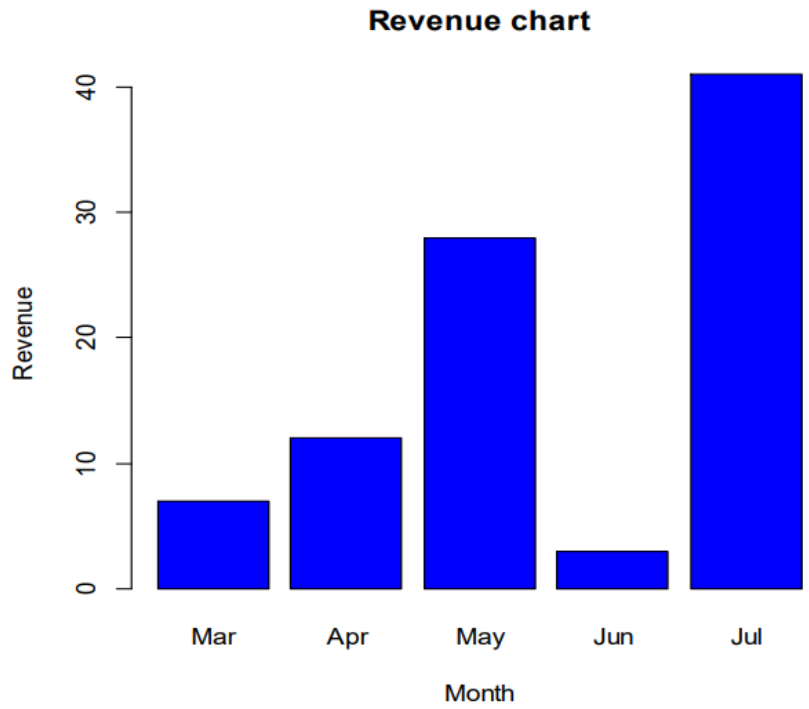
- # stacked barplots
- barplot(VADeaths[,c("Rural Male", "Rural Female")])



- > # juxtaposed barplots
- > barplot(VADeaths [,c("Rural Male", "Rural Female")],beside=T)

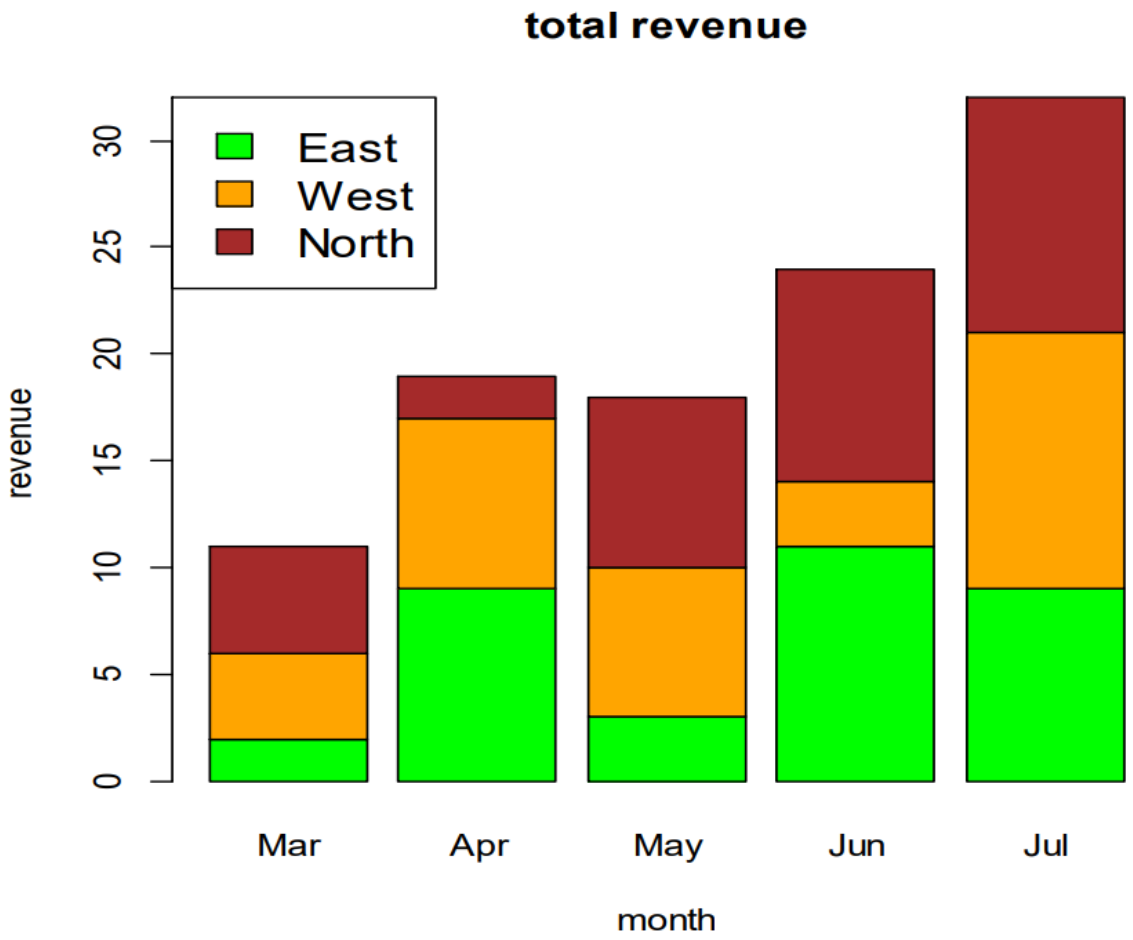


```
>H <- c(7,12,28,3,41)
>M <- c("Mar","Apr","May","Jun","Jul")
>barplot(H,names.arg = M,xlab = "Month",ylab =
"Revenue",col="blue",main = "Revenue chart")
```



Example :-

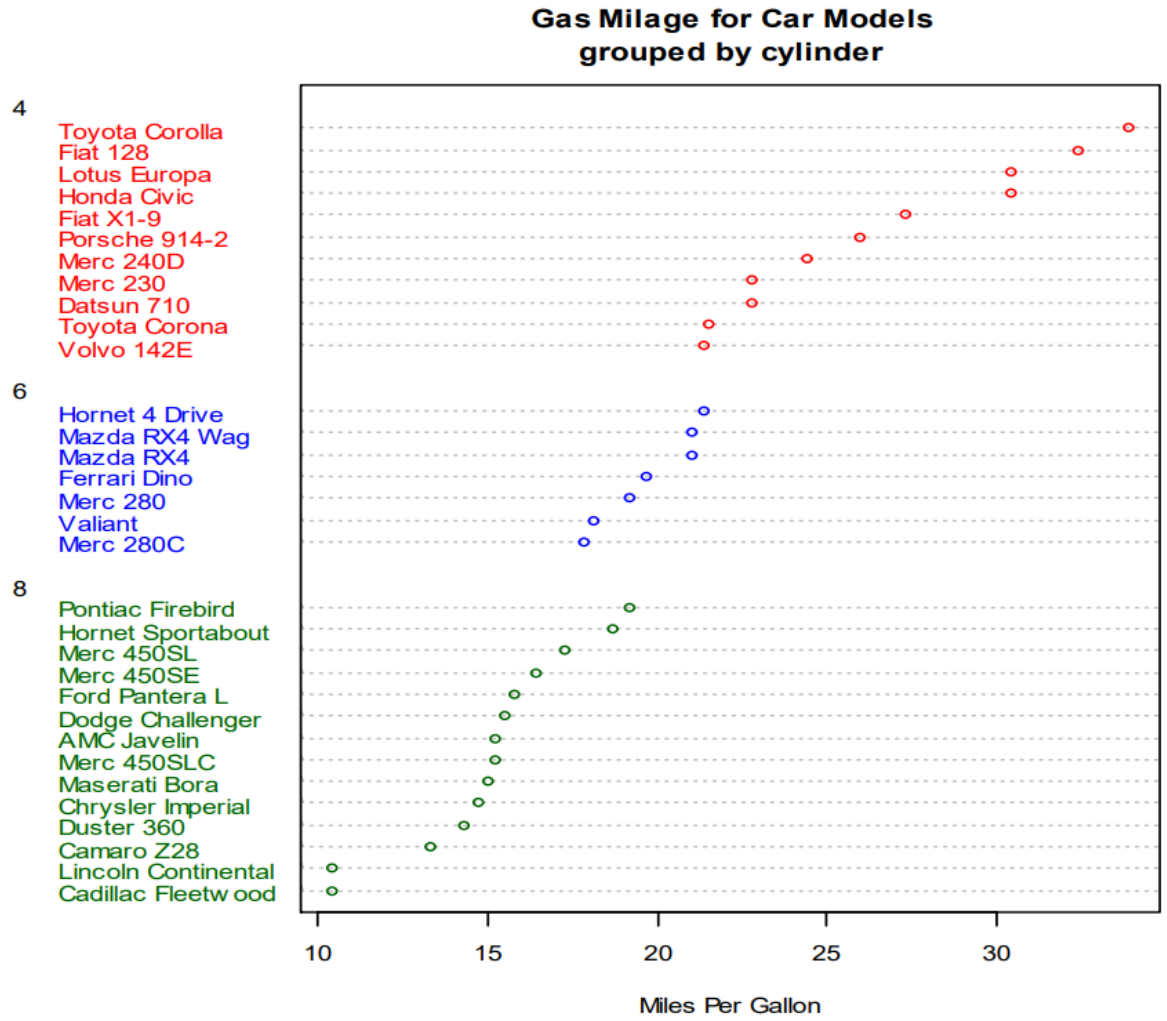
```
>colors <- c("green","orange","brown")
>months <- c("Mar","Apr","May","Jun","Jul")
>regions <- c("East","West","North")
>Values <-
matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11),nrow = 3,ncol
= 5,byrow =TRUE)
>barplot(Values,main = "total revenue",names.arg =
months,xlab = "month",ylab = "revenue",col=colors)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


total revenue
East
West
North
revenue
0
5
10
15
20
25
30
Mar
Apr
May
Jun
Jul
month

- # Simple Dotplot

>dotchart(mtcars$mpg,labels=row.names(mtcars),cex=.7, main="Gas Milage for Car Models",xlab="Miles Per Gallon")

- # Dotplot: Grouped Sorted and Colored
- # Sort by mpg, group and color by cylinder
- >x <- mtcars[order(mtcars$mpg),] # sort by mpg
- >x$cyl <- factor(x$cyl) # it must be a factor
- >x$color[x$cyl==4] <- "red"
- >x$color[x$cyl==6] <- "blue"
- >x$color[x$cyl==8] <- "darkgreen"
- >dotchart(x$mpg,labels=row.names(x),cex=.7,groups= x$cyl,main="Gas Milage for Car Models\ngrouped by cylinder",xlab="Miles Per Gallon",gcolor="black", color=x$color)

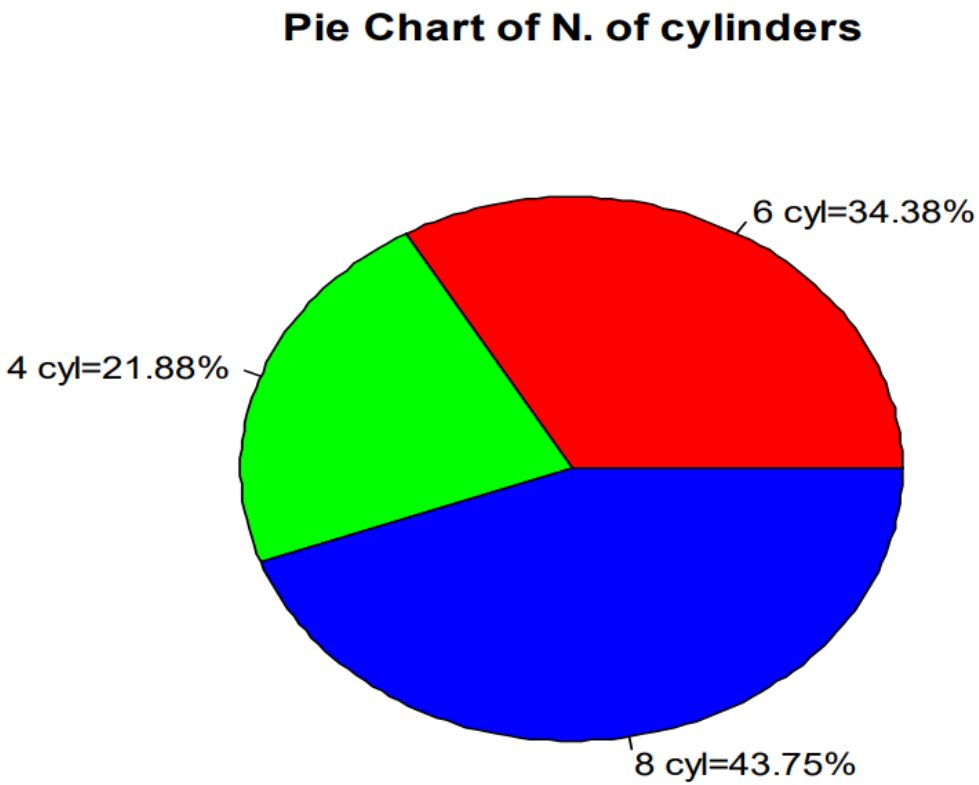

Gas Milage for Car Models
grouped by cylinder
4
Toyota Corolla
Fiat 128
Lotus Europa
Honda Civic
Fiat X1-9
Porsche 914-2
Merc 240D
Merc 230
Datsun 710
Toyota Corona
Volvo 142E
6
Hornet 4 Drive
Mazda RX4 Wag
Mazda RX4
Ferrari Dino
Merc 280
Valiant
Merc 280C
8
Pontiac Firebird
Hornet Sportabout
Merc 450SL
Merc 450SE
Ford Pantera L
Dodge Challenger
AMC Javelin
Merc 450SLC
Maserati Bora
Chrysler Imperial
Duster 360
Camaro Z28
Lincoln Continental
Cadillac Fleetw ood
10
15
20
25
30
Miles Per Gallon

- **Pie**
- pie(x) draws a circle (pie) cut into segments (slices), each slice represents a unique value from the elements of x and the sixe of the slice and the relative frequency of each unique value is represented by the size of t

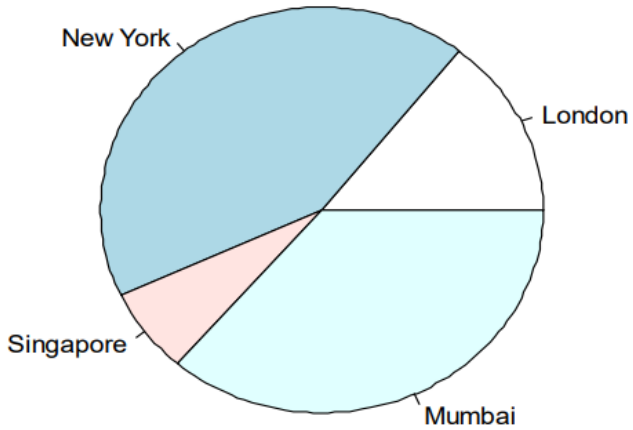
 # simple pie

```
>pie(unique(mtcars$cyl), labels = unique(mtcars$cyl), main="Pie Chart of N. of cylinders")                    # pie with percentages and colors
>with(mtcars, {
>n.cyl <- unique(cyl)
>percent.cyl <-round(table(cyl)/dim(mtcars)[1]*100,2)
>lbls <- paste(n.cyl," cyl=",percent.cyl,"%", sep="")
>pie(n.cyl, labels = lbls , main="Pie Chart of N. of cylinders", col=rainbow(length(lbls)))})
```

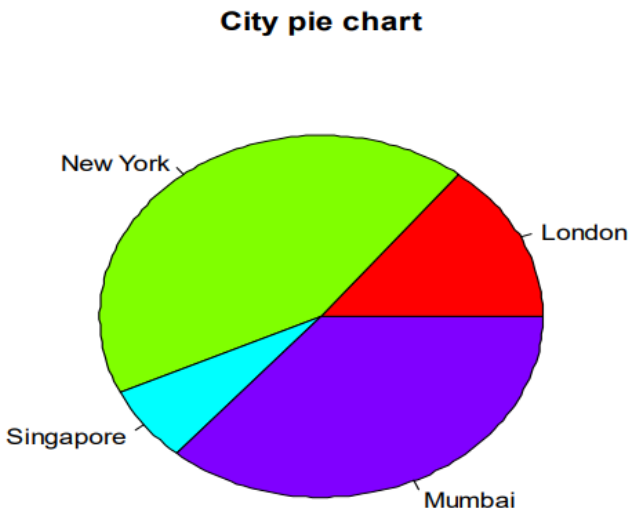
Ac


Pie Chart of N. of cylinders
6 cyl=34.38%
4 cyl=21.88%
8 cyl=43.75%

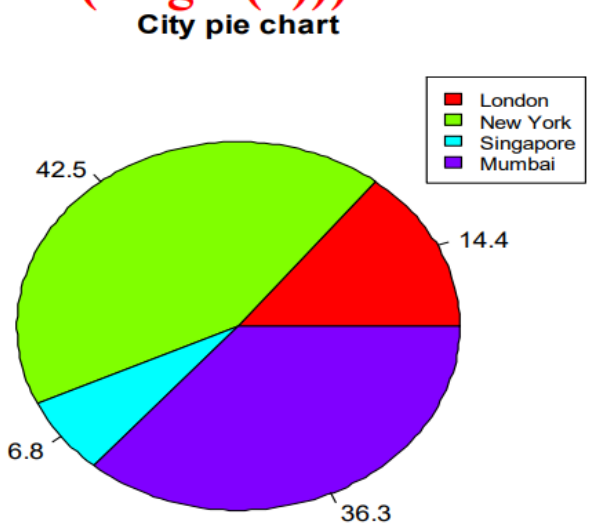
```
>x <- c(21, 62, 10, 53)
>labels <- c("London", "New York", "Singapore",
"Mumbai")
>pie(x,labels)
```



```
>x = c(21, 62, 10, 53)
>labels = c("London", "New York", "Singapore",
"Mumbai")
>pie(x, labels, main = "City pie chart", col =
rainbow(length(x)))
```

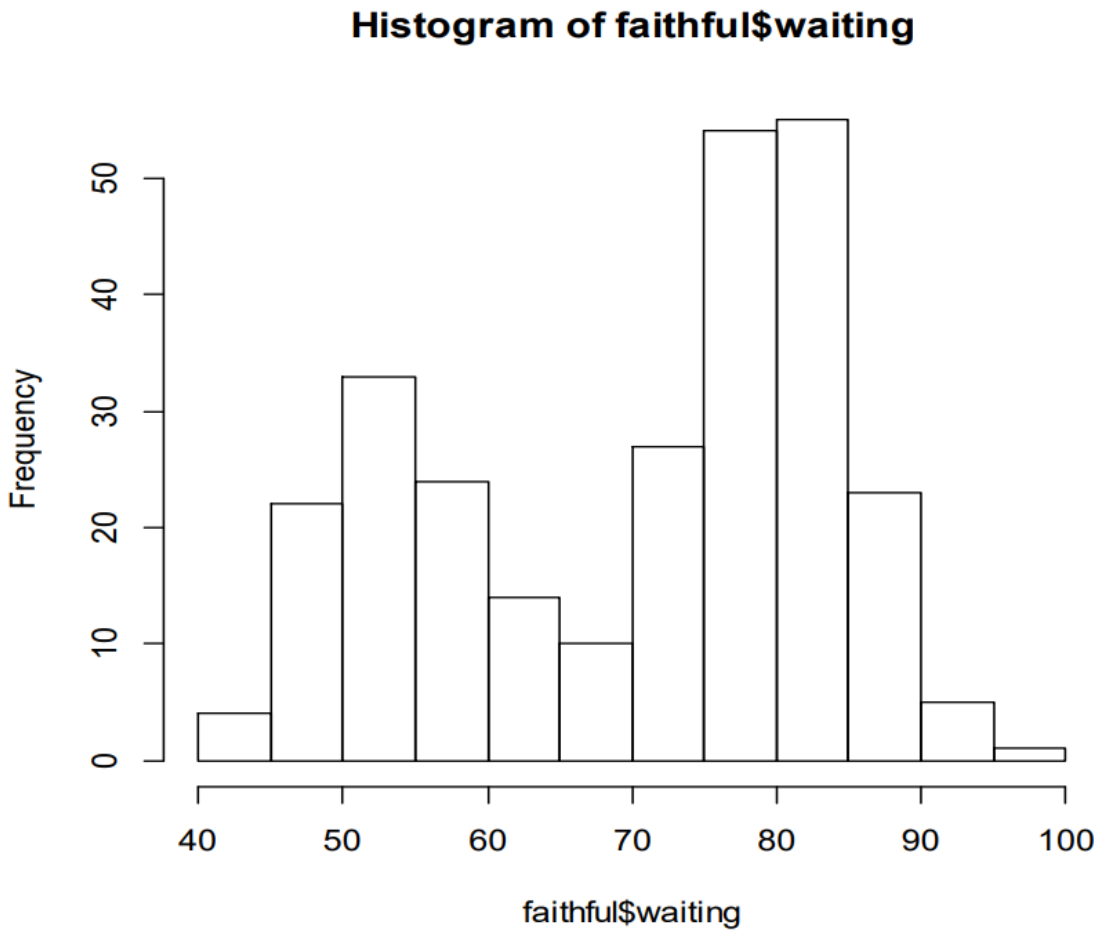


```
>x <- c(21, 62, 10,53)
>labels <- c("London","New York“ ,"Singapore“ ,"Mumbai" )
>piepercent<- round(100*x/sum(x), 1)
>pie(x, labels = piepercent, main = "City pie chart",col =
rainbow(length(x)))
>legend("topright", c("London","New York","Singapore",
"Mumbai"), cex = 0.8,fill = rainbow(length(x)))
```



- **histogram**
- hist(X) is an histogram, a bar plot with the frequencies of the values in X on the y-axis and the ranges of values on the x-axis
- A cumulative distribution curve is the proportion of X on the y-axis, up to the current position on the x-axis

- **> # simple histogram**
- **> hist(faithful$waiting)**

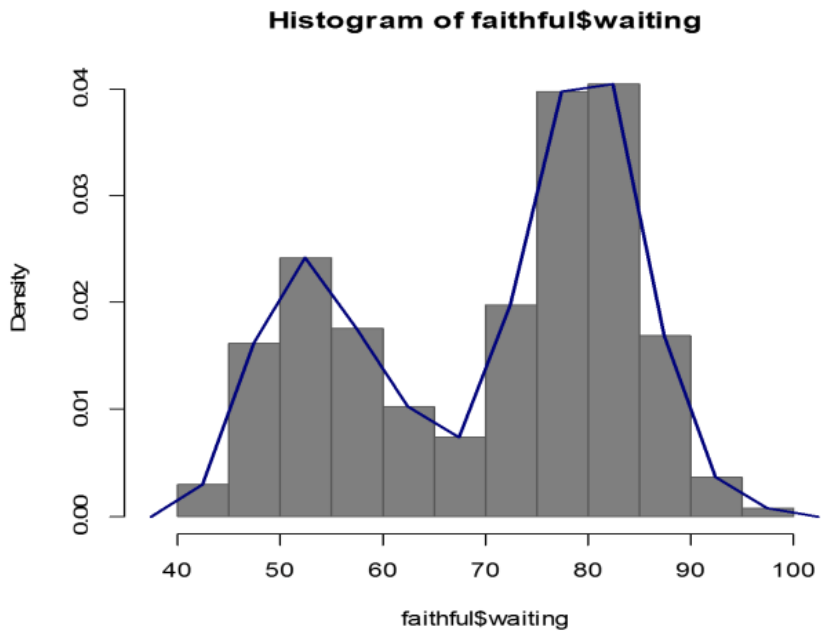

Histogram of faithful$waiting
Frequency
0
10
20
30
40
50
40
50
60
70
80
90
100
faithful$waiting

draw the histogram

>hist(faithful$waiting, prob =TRUE, xlim=range(xx) , border = "gray" , col="gray90")

adds the frequency polygon

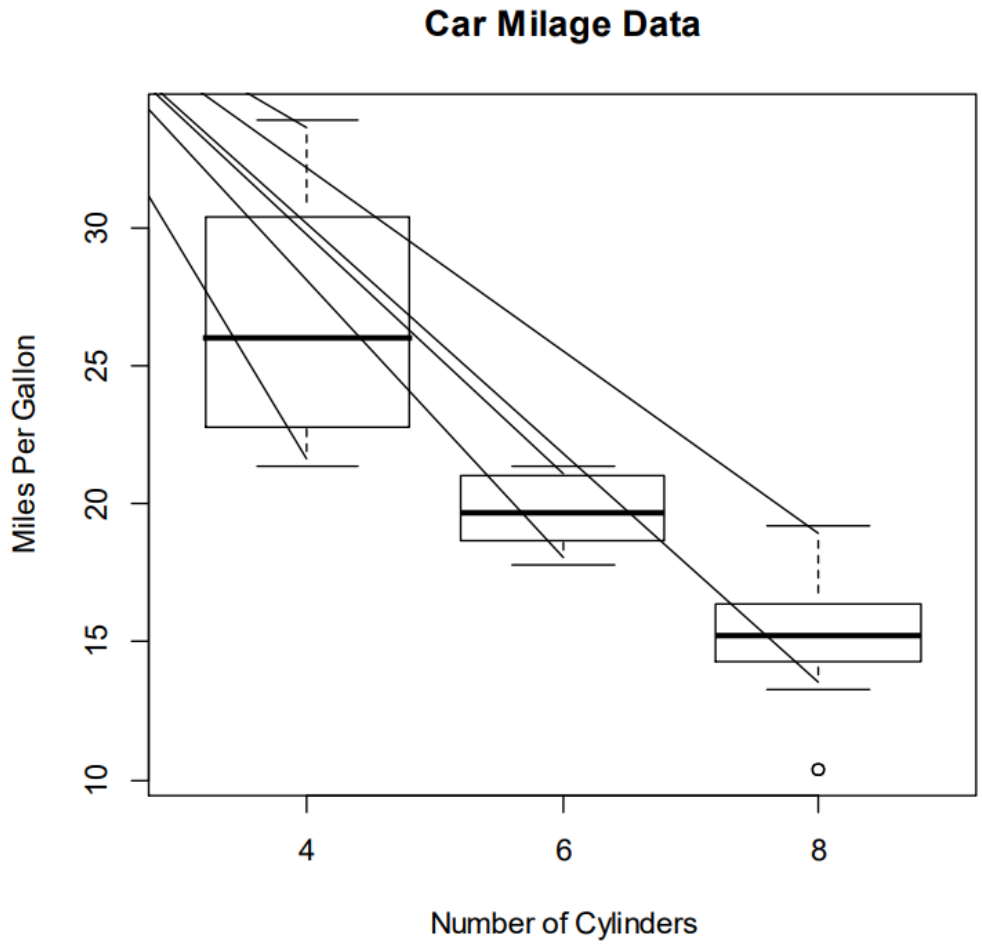
>lines(xx, yy, lwd=2, col = "royalblue")



- **boxplot**

boxplot(X) is a box-and-whisker plot with the values of variable X, this is an effective way to summarize larger datasets.

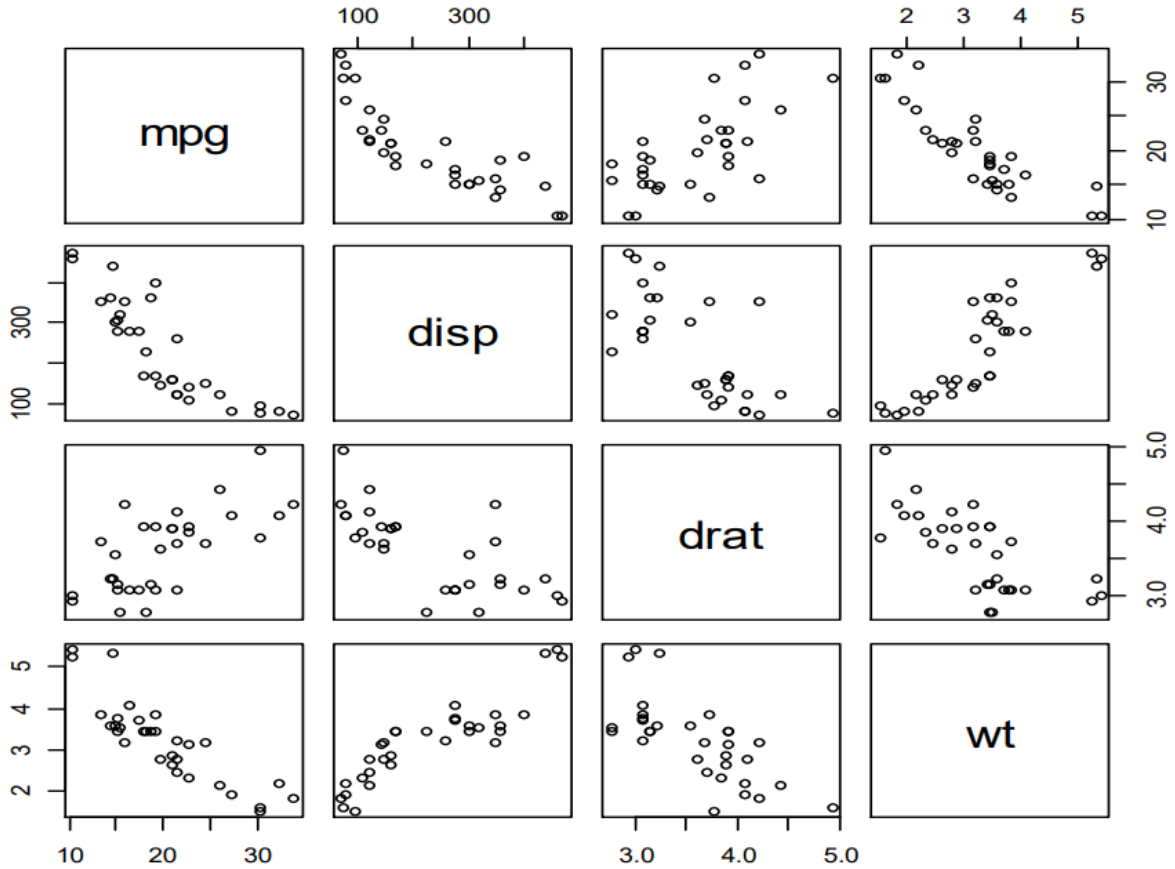
```
# Boxplot of MPG by Car Cylinders
> boxplot(mpg~cyl,data=mtcars, main="Car Milage
Data",xlab="Number of Cylinders", ylab="Miles Per
Gallon")
```


Car Milage Data
Miles Per Gallon
10
15
20
25
30
4
6
8
Number of Cylinders

- **Pairs**
- pairs() shows a matrix with all the scatterplots for the columns of variable X

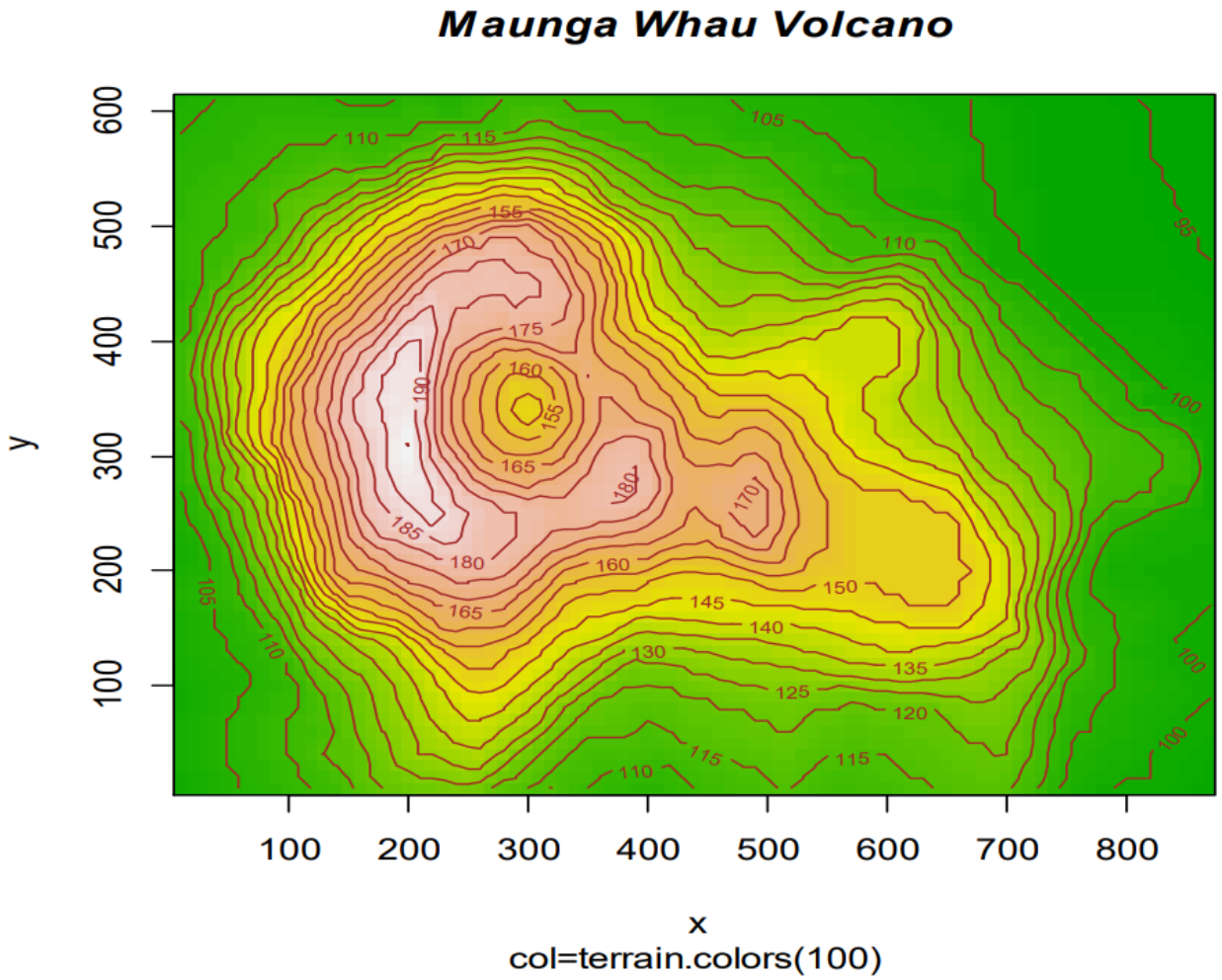
- **pairs(~mpg+disp+drat+wt,data=mtcars, main="Scatterplot Matrix MPG, Displacement,Rear axle ratio,Weight")**

Scatterplot Matrix MPG, Displacement,Rear axle ratio, Weight



- **Contour**
- contour(X,Y,Z) draws a contour plot, with vector X for the rows, vector Y for the columns and matrix X for the data

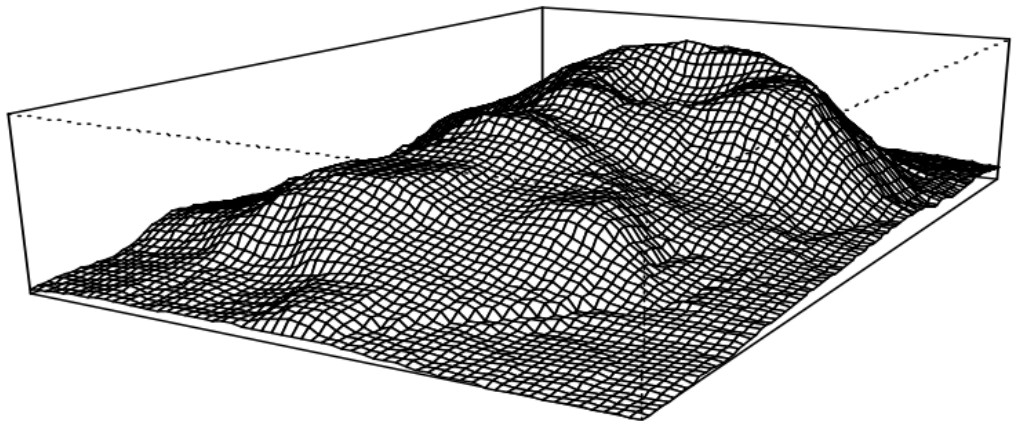
```
>x <- 10*(1:nrow(volcano)); x.at <- seq(100, 800, by=100)
>y <- 10*(1:ncol(volcano)); y.at <- seq(100, 600, by=100)
# Using Terrain Colors
>image(x, y, volcano, col=terrain.colors(100),axes=FALSE)
>contour(x, y, volcano, levels=seq(90, 200, by=5), add=TRUE,
col="brown")
>axis(1, at=x.at)
>axis(2, at=y.at)
>box()
>title(main="Maunga Whau Volcano", sub =
"col=terrain.colors(100)", font.main=4)
```


Maunga Whau Volcano
y
x
col=terrain.colors(100)
100
200
300
400
500
600
700
800

- **Persp**

persp(X,Y,Z) draws a 3d graph, with vector X for the rows, vector Y for the columns and matrix X for the data

```
## (2) Visualizing a simple DEM model
>z <- 2 * volcano # Exaggerate the relief
>x <- 10 * (1:nrow(z)) # 10 meter spacing (S to N)
>y <- 10 * (1:ncol(z)) # 10 meter spacing (E to W)
>persp(x, y, z, theta = 120, phi = 15, scale = FALSE, axes =
FALSE)
```



- **Tables**

Example:

```
> library(MASS)
>ships
> table(ships$type)
```

```
A B C D E
8 8 8 8 8
```

```
>table(ships$type,ships$year)
```

```
  60 65 70 75
A  2  2  2  2
B  2  2  2  2
C  2  2  2  2
D  2  2  2  2
E  2  2  2  2
```

Example :-

```
>library(MASS)
>USArrests
>table(USArrests[,3])
```

```
32 39 44 45 48 50 51 52 53 54 56 57 58 59 60 62 63 65 66 67 68 70 72 73 74 75
 1  1  2  2  2  1  1  1  1  1  1  1  1  1  2  1  1  1  4  2  1  2  2  1  1  1
77 78 80 81 83 85 86 87 89 91
 1  1  4  1  2  1  1  1  1  1
```

```
>table(cut(USArrests[,3],pretty(USArrests[,3])))

(30,40] (40,50] (50,60] (60,70] (70,80] (80,90] (90,100]
      2       7      10      12      11       7       1
```

Example :-

```
> airquality
> table(airquality[,4],airquality[,5])
>table(cut(airquality[,4],pretty(airquality[,4])),
  airquality[,5])
```

	5	6	7	8	9
(50,60]	8	0	0	0	0
(60,70]	16	2	0	0	7
(70,80]	6	18	3	11	14
(80,90]	1	8	25	15	5
(90,100]	0	2	3	5	4

Example :-

```
> library(MASS)
> cars
```

```
> head(cars)
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
5     8   16
6     9   10
```

```
> tail(cars)
   speed dist
45    23   54
46    24   70
47    24   92
48    24   93
49    24  120
50    25   85
```

```
> head(cars,3)
  speed dist
1     4    2
2     4   10
3     7    4
```

```
> tail(cars,3)
   speed dist
48    24   93
49    24  120
50    25   85
```

Thank You