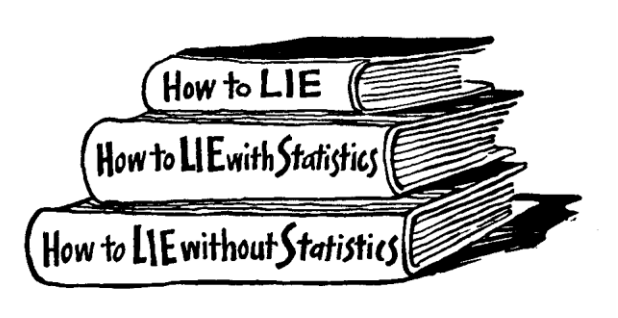
MAT2001-STATISTICS FOR ENGINEERS-EMBEDDED LAB
R-BASICS-LAB-I & LAB-2

"There are three kinds of lies: lies, damned lies, and statistics." (B.Disraeli)



What is R?

- Introduction to 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, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

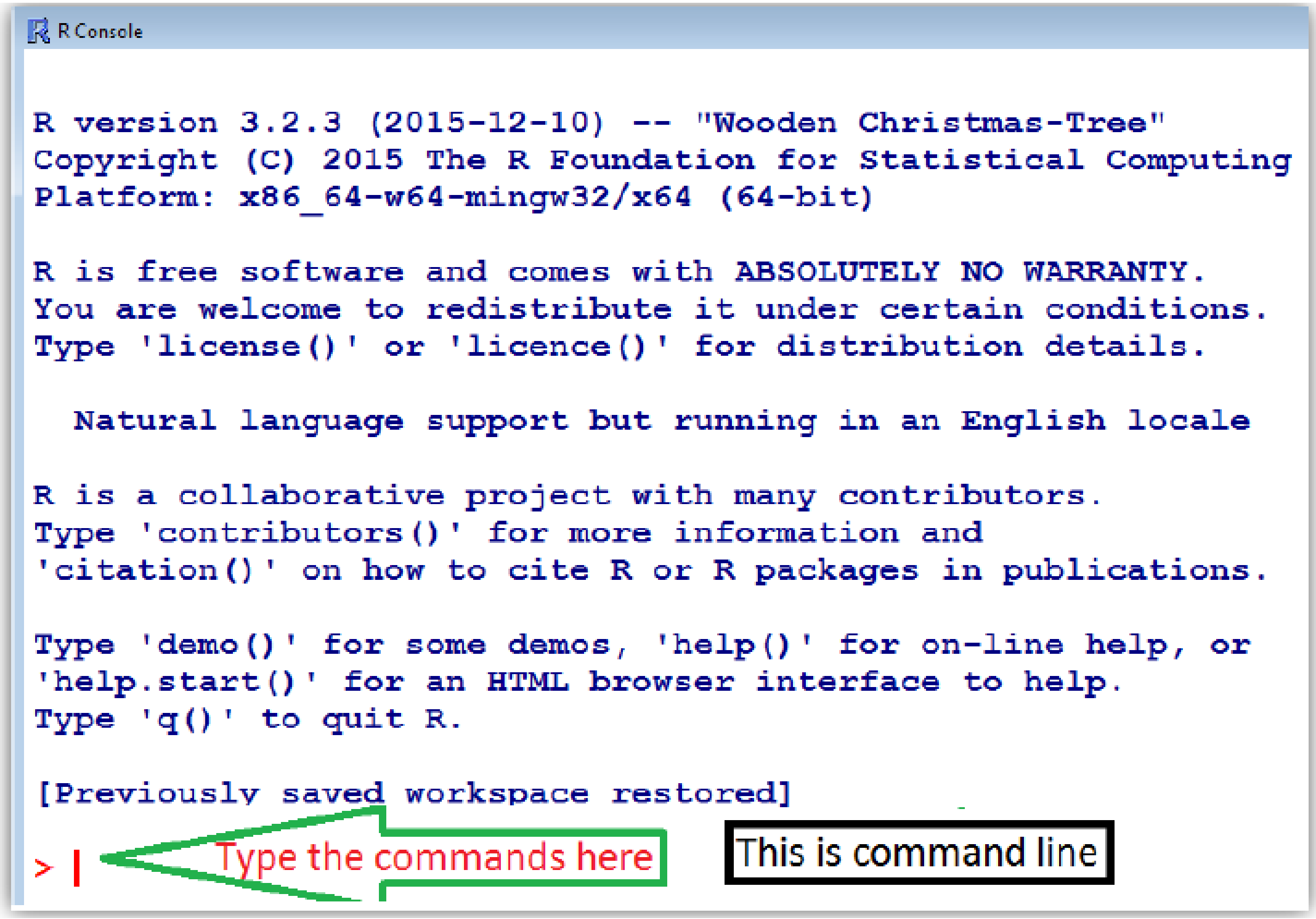
Why should you adopt R Programming Language?

- 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.

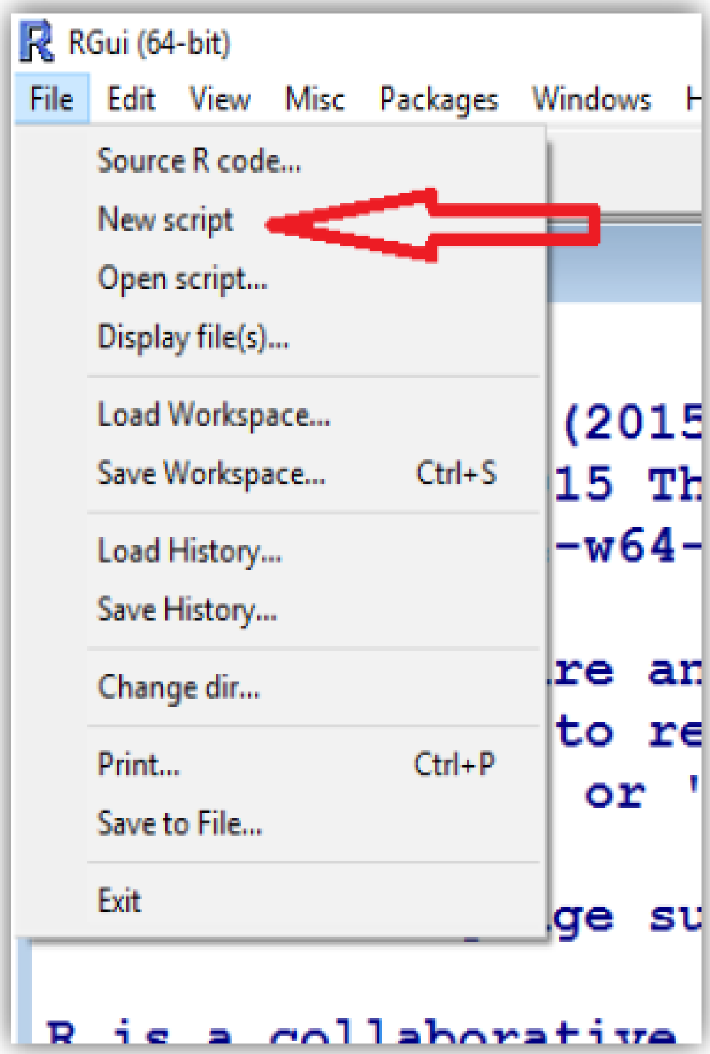
Command Line vs Scripts

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

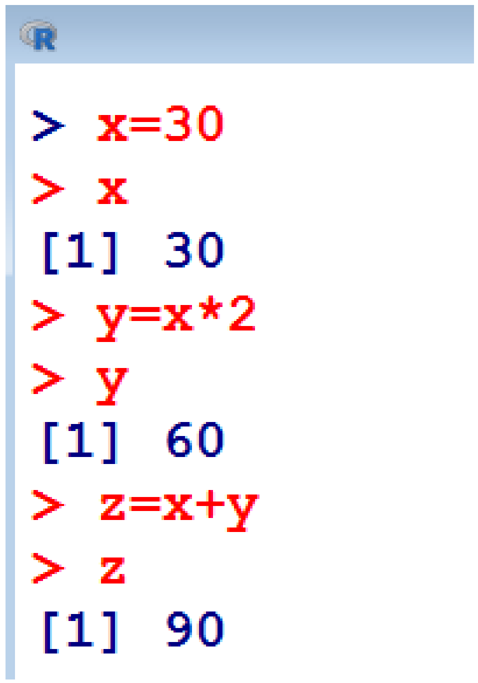
- > is the prompt sign in R. you can enter numbers and perform calculations.
- The assignment operators are the left arrow with dash <- ,->and equal sign =.

R Console

```
> x <- 20   #assigns the value 20 to x.
> 20->x     #assigns the value 20 to x.
> x=20      #assigns the value 20 to x.
```

- > 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] 5
> 2-3
 [1] -1
> 2*3
 [1] 6
> 2/3
 [1] 0.6666667
> 2*3-4+5/6
 [1] 2.833333
```

```
> 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:**

R Console

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

- **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.

R Console

```
> seq(from=2, to=4)
[1] 2 3 4
> seq(2,4)
[1] 2 3 4
> seq(5,2)
[1] 5 4 3 2
> seq(-5,5)
 [1] -5 -4 -3 -2 -1  0  1  2  3  4  5
>
```

- **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**

R Console

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
> letters[1:3]
[1] "a" "b" "c"
> letters[3:1]
[1] "c" "b" "a"
> letters[21:23]
[1] "u" "v" "w"
> letters[5]
[1] "e"
>
```

- **sequence of uppercase alphabets**

R Console

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"
> LETTERS[1:3]
[1] "A" "B" "C"
> LETTERS[3:1]
[1] "C" "B" "A"
> LETTERS[21:24]
[1] "U" "V" "W" "X"
> LETTERS[5]
[1] "E"
>
```

- **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.

R Console

```
> y <- c(9,8,5,7,6)
> y
[1] 9 8 5 7 6
> sort(y)
[1] 5 6 7 8 9
> sort(y, decreasing = TRUE)
[1] 9 8 7 6 5
>
```

- **Length**

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

R Console

```
> x <- c(1,2,5,4,6,1,22,1)
> length(x)
[1] 8
> x=1:55
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
[51] 51 52 53 54 55
> length(x)
[1] 55
```

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

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[myvec > 500]          # only elements above 500
[1] 734 985 952 885
> myvec[(myvec %% 2)==0]      # only even elements
[1] 734 952  28
> myvec[myvec %in% 100:500]  # elements with values from 100 to 500
[1] 235 193
>
```

- 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

R Console

```
> 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 Console

```
> 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)

R Console

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
> ifelse( x<6, x^2, x+1 )
 [1]  1  4  9 16 25  7  8  9 10 11
```

- **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.

R Console

```
> i <- 1
> repeat{
+ print( i^2 )
+ i <- i+2
+ if ( i>10 ) break
+ }
[1] 1
[1] 9
[1] 25
[1] 49
[1] 81
```

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)

R Console

```
> 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.

R Console

```
> str(painters)
'data.frame':   54 obs. of  5 variables:
 $ Composition: int  10 15 8 12 0 15 8 15 4 17 ...
 $ Drawing    : int  8 16 13 16 15 16 17 16 12 18 ...
 $ Colour     : int  16 4 16 9 8 4 4 7 10 12 ...
 $ Expression : int  3 14 7 8 0 14 8 6 4 18 ...
 $ School     : Factor w/ 8 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

- **Extract a variable from data frame using $**

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

```
R Console
> painters$School
 [1] A A A A A A A A A A B B B B B B C C C C C C D D D D D
[28] D D D D D E E E E E E E F F F F G G G G G G G H H H H
Levels: A B C D E F G H
```

- **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.

R Console

```
> library(MASS)
> splitted <- split(painters, painters$School)
> splitted
```

- **Practice**

1. x=23, y=x^2, z=y^3+x^2

2. 2+3-4*6/3-2^3/4+7-6*2**3/2+2-4+5 ?

3. -478/365-12^3/47-6**2**3-27^-(1/3)-7^6*3

4. c(1,2,3,4)*c(1,2,2,4)^c(2,1,3,2)**c(1,2,3,3)-c(2,3,402653185,262145) ?

5. c(2,3,4,5)^c(2,3)+c(12,23,14,25)^c(3,2)-c(5,6,7,8)*c(2,3)?

- max(c(62,83,44,75)^-c(9,-3)) / min(c(52,62,71,85) ^ c(2,3)) -prod(c(1,2,1,2) ^c(1,2)) +max(c(12,13,14,15) ^ c(2,3)) ?

- X1 <- c(123,258,318,624), X2 <- sqrt(X1^3)+X1/X1^2-X1**(1/2)

- X<-matrix(nrow=3,ncol=3,data= c(10,20,30,40,50,60,70,80,90),byrow=F) then X[,2]?

- X<-matrix(nrow=3, ncol=3, data = c(10,20,30,40,50,60,70,80,90) , byrow=F) then X[2:3,2:3] ?

- **x = 33:53** then specifies the outcomes of the following statements:
  x[(x > 28) & (x < 59)] and x[(x > 28) | | (x < 59)]?

- **sqrt(abs(seq(-6,6, by = 3)))**?

- **letter[1:3]**?

- **LETTERS[18:12]**

- **rep(2:5,4)**

- **rep(70:65, times=3)?**
- **rep(20:25, each=3)?**
- **x <- matrix(nrow=2, ncol=2, data=1:4, byrow=T)**
  rep(x, each=3)?
- **sort(c(20,50, 10, 30, 90,70, 80), decreasing = FALSE)**?
- **sort (c(20,50, 10, 30, 90,70, 80), increasing = TRUE)?**

- **order(c(20,50, 10, 30, 90,70, 80), decreasing = FALSE)**?

- **mode(c(1, 2, "3", 8+9, "7+9", 6.7, 110*45))**?

- **x <- c(10, 75, 20, 35, 30, 40, 180, 50, 60, 27, 70, 67, 80, 50, 39, 120)**

>x[(x>50)]?
>x[(x - 20 > 40)]
>x[(x^2 + 10 > 50)]

- **factor(c(rep("male",2), rep("female", 3)))**?