1. R Basic

Getting Started with R

- R as calculator
  - Arithmetic Operator

    +, -, *, /, %/%, %%, ^

  - Useful Mathematical Functions

Functions	Meaning
log(x)	log to base e of x
exp(x)	antilog of x (ex)
log10(x)	log to base 10 of x
sqrt(x)	square root of x
round(x,digit=0)	round the value of x to an integer
cos(x)	cosine of x in radians
sin(x)	sine of x in radians
tan(x)	tangent of x in radians
abs(x)	The absolute value of x

Getting Started with R

- Comparisons and Logical Expressions
  - Relational Operators

    ==, !=, >=, <=, >, < : producing the answer TRUE or FALSE
  - Logical operators
    - ! : logical NOT.

      !expression is TRUE if and only if expression is FALSE.
    - & : logical AND.

      expression & expression is TRUE if both expressions are TRUE.
    - | : logical OR.

      expression | expression is TRUE if at least one of the two expressions is TRUE.

Getting Started with R

```
> 5^3 * exp(-5)
[1] 0.8422434
> (0.9^5) * (0.1^5)
[1] 5.9049e-06
> 1 / (1*sqrt(2*pi)) * exp(-((3-1)^2)/2)
[1] 0.05399097
 > 4 %% 2 != 0
[1] FALSE
> cos(pi/2) == 0
[1] FALSE
> !( 3 <= 4 )
[1] FALSE
> 5 > 3 & 0 != 1
[1] TRUE
> 5 > 3 | 0 != 1
[1] TRUE
```

Getting Started with R

- Assignments
  - Assignment
    - Objects obtain values or functions in R by assignment, which is achieved by '<-' or '='.
  - Variable Names
    - case-sensitive.
    - should not begin with numbers or symbols.
    - should not contain blank spaces.
- Getting Help
  - Getting help in R
    - help(*topic*) : displays an help page for the command topic (= ?*topic*).

```
> x <- 5
> x
[1] 5
> y <- x^2 + 3
> y
[1] 28
```

Getting Started with R

- R Objects
  - data objects : vectors, factors, matrices (more generally arrays), lists, dataframes
  - function objects
- Mode of Data Objects
  - mode : what type of data the object contains

Data type	Description	Examples
logical	TRUE or FALSE	TRUE, FALSE
numeric	integer and double	5, -2, 3.14, pi, sqrt(2)
complex	complex number	2.1+3i, 5+0i
character	character string	'This is text', "5"

  - mode(*object*)
  - as.<type> (*object*) : Converting Data Types
  - is.<type> (*object*) : Checking Data Types, returns TRUE or FALSE

Getting Started with R

```
> a <- "Hello World“ ; a
[1] "Hello World"
> b <- 2 < 4 ; b
[1] TRUE
> mode(a)
[1] "character"
> mode(b)
[1] "logical"
> x <- TRUE
> as.numeric(x)
[1] 1
> as.character(x)
[1] "TRUE“
> as.numeric("3.14159")
[1] 3.14159
> a <- "3"
> is.numeric(a)
[1] FALSE
> is.logical(a)
[1] FALSE
```

Getting Started with R

- Special Values in R
  - NA(Not Available)
    - data that is not available
    - In general, any operation on an NA becomes an NA.
  - NaN(Not A Number)
    - an undefined result for a mathematical operation
    - In R, NaN implies NA.
  - Inf, -Inf
    - positive and negative infinity (everything outside a certain range)
  - Testing for Special Values
    - is.na(*x*) : tests for NA or NaN data in *x*
    - is.nan(*x*) : tests for NaN data in *x*
    - is.infinite(*x*) : tests if *x* is positive or negative infinity
    - To test for special values, never use == .

Getting Started with R

```
> 5*NA
[1] NA
> 0/0
[1] NaN
> 0*Inf
[1] NaN
> exp(710)
[1] Inf
> exp(-Inf)
[1] 0
> x <- NaN
> is.na(x)
[1] TRUE
> x == NA
[1] NA
```

Data Object - Vectors

- Denition
  - Vectors are variables with one or more values of the same type : logical, numeric, complex, character.
- Creating Vectors
  - c(*arguments*) : concatenates *arguments* to form a vector
  - seq(from=*val1*, to=*val2*) : creates a sequence from *val1* to *val2* , with step size 1 (or -1)
    - by=*val3* : ... , with step size *val3*
    - length=*val3* : ... , with length *val3*
  - *val1* : *val2* : seq(from= *val1*, to= *val2*)
  - rep(*x*, each=*val1*, times=*val2*) : each of *x* is repeated *val1* times, the whole series repeated *val2* times

Data Object - Vectors

```
> c(2, 5, 3, 7)
[1] 2 5 3 7
> c(1, "2")
[1] "1" "2"
> c(TRUE, 3)
[1] 1 3
> v1 <- c(1, 2, 3)
> v2 <- c(4, 5)
> v3 <- c( v1, v2, 6 )
> v3
[1] 1 2 3 4 5 6
> seq(from=1, to=10, by=3)
[1]  1  4  7 10
> seq(3, 7, length=10)
 [1] 3.000000 3.444444 3.888889 4.333333 4.777778 5.222222
 [7] 5.666667 6.111111 6.555556 7.000000
> 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, times=2)
 [1] 1 2 3 4 1 2 3 4
```

Data Object - Vectors

- Attributes of a Vector
  - Attributes of a Vector : mode, length, names
    - mode(*vector*) : mode of *vector*
    - length(*vector*) : number of elements in *vector*
    - names(*vector*) : elements' names in *vector*

```
> age <- c("Lee"=22, "Ryu"=25, "Kim"=20)
> age
Lee Ryu Kim
 22  25  20
> mode(age)
[1] "numeric"
> length(age)
[1] 3
> names(age)
[1] "Lee" "Ryu" "Kim"
```

Data Object - Vectors

- Reference Elements of a Vector
  - Subscripts of a Vector

```
vector [index vector]
```

Such index vector can be any of four distinct types below.

- A vector of positive integral quantities
- A vector of negative integral quantities
- A logical vector
- A vector of character strings (only for named vector)

  - Updating a Vector

```
vector [index vector] <- replacement values
```

Data Object - Vectors

```
> x <- c(12,15,13,17,11)
> x[3:5]
[1] 13 17 11
> x[2*(1:2)] <- 20
> x
[1] 12 20 13 20 11
> x[-(3:5)]
[1] 12 20
> v <- c(12,15,13,17,11)
> v[v > 12]
[1] 15 13 17
> v[v <= 16 & (v%%2) == 1]
[1] 15 13 11
> v[v > 12] <- 0
> v
[1] 12  0  0  0 11
> v[v == 0] <- c(5, 6, 7)
> v
[1] 12  5  6  7 11
```

Data Object - Vectors

```
> w <- c(1, 3, NA, 5)
> w2 <- w[!is.na(w)]
> w2
[1] 1 3 5
> fruit <- c("orange"=5, "banana"=10, "apple"=1, "peach"=20)
> lunch <- fruit[c("apple", "orange")]
> lunch
 apple orange
     1      5
```

Data Object - Vectors

- Vector Operations
  - Calculation of Vectors
    - Elementwise Operation
      - Standard operations (e.g. +, -, *, /, ^, log, exp, sin, cos, tan, sqrt, !=, <, >=, &, |, !) on vectors are element by element.
    - Recycling Rule
      - When the length of one vector is not the same as the length of the other vector, the shorter vectors are recycled until they match the length of the longest vector.

Data Object - Vectors

- Useful Commands on Vectors

Functions	Meaning
max(x), min(x)	maximum or minimum value in x
sum(x)	total of all the values in x
mean(x)	arithmetic average of the values in x
prod(x)	the product of all the values in x
median(x)	median value in x
var(x)	sample variance of x
sd(x)	sample standard deviation of x
cor(x,y)	sample correlation between x and y
quantile(x, prob=c(0, 0.25, 0.5, 0.75, 1))	sample quantiles corresponding to the given probabilities
sort(x, decreasing=FALSE)	a sorted version of x, in increasing order
which(x)	give the TRUE subscript of a set of logical vectors x
is.element(x,y)	performs set membership on x and y (=%in%)

Data Object - Vectors

```
> c(2, 5, 3) + c(4, 2, 7)
[1]  6  7 10
> c(3, 2) * c(2, 5, 3, 4)
[1]  6 10  9  8
> x1 <- c(3, 1, 4, 5, 9, 2, 6, 3, 7, 6)
> y1 <- c(2, NA, 4, 5, 9, NA, 6, 3, 7, 1)
> max(x1)
[1] 9
> max(y1, na.rm=TRUE)
[1] 9
> mean(x1)
[1] 4.6
> var(y1, na.rm=TRUE)
[1] 7.125
> sd(x1)
[1] 2.458545
> quantile(x1)
  0%  25%  50%  75% 100%
 1.0  3.0  4.5  6.0  9.0
> quantile(x1, prob=c(0, 0.2, 1))
  0%  20% 100%
 1.0  2.8  9.0
```

Data Object - Vectors

```
> tmp <- 0:10
> sum(tmp < 5)
[1] 5
> sum(tmp[tmp < 5])
[1] 10
> v <- c(13, 15, 11, 12, 19, 14, 18, 19)
> sort(v)
[1] 11 12 13 14 15 18 19 19
> mean( sort(v)[-c(1, length(v))] )
[1] 15.16667
> which(v <= 17)
[1] 1 2 3 4 6
> a <- c(1, 3, 4, 5, 8)
> b <- c(2, 3, 5, 10)
> is.element(a, b)
[1] FALSE  TRUE FALSE  TRUE FALSE
> a %in% b
[1] FALSE  TRUE FALSE  TRUE FALSE
```

Data Object - Matrices

- Denition
  - A matrix is a two-dimensional generalization of a vector.
  - The elements of a matrix must be of the same mode.
- Creating Matrices

  To create a matrix, use the command matrix().
  - matrix(data=*vector*): converting a *vector* into a matrix.
    - By default values are stored by columns and the number of columns=1.
    - We can specify the desired number of rows and columns using the argument nrow=value1 , ncol=value2 .
    - To store values by rows, use the optional argument byrow=TRUE.

  To combind vectors by rows or columns, use the command rbind() and cbind().
  - rbind(*arguments*) : binding together vectors in *arguments* row-wise
  - cbind(*arguments*) : binding together vectors in *arguments* column-wise

Data Object - Matrices

```
> matrix(1:8, nrow=4, ncol=2)
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
> matrix(c("a", "b", "c", "d", "e", "f"), ncol=3, byrow=TRUE)
     [,1] [,2] [,3]
[1,] "a"  "b"  "c"
[2,] "d"  "e"  "f"
> cbind(1:3, 5:7)
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
> rbind( matrix(c(T, F, F, F), 4, 3), matrix(2, 3, 3))
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    0    0    0
[3,]    0    0    0
[4,]    0    0    0
[5,]    2    2    2
[6,]    2    2    2
[7,]    2    2    2
```

Data Object - Matrices

```
> m <- matrix(data=1:8, nrow=4, ncol=2)
> m2 <- c(9, 10, 11, 12)
> cbind(m, m2)
          m2
[1,] 1 5  9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
```

Data Object - Matrices

- Attributes of a Matrix
  - Attributes of a Matrix
    - nrow(*matrix*) : gets the number of rows present in *matrix*
    - ncol(*matrix*) : gets the number of columns present in *matrix*
    - length(*matrix*) : gets or sets the number of values in *matrix*
    - mode(*matrix*) : gets or sets mode of *matrix*
    - rownames(*matrix*) : gets or sets row names in *matrix*
    - colnames(*matrix*) : gets or sets column names in *matrix*

Data Object - Matrices

```
> z <- matrix(1:10, 2, 5)
> nrow(z)
[1] 2
> ncol(z)
[1] 5
> mode(z)
[1] "numeric"
> length(z)
[1] 10
> rownames(z) <- c("R1", "R2")
> colnames(z) <- c("C1", "C2", "C3", "C4", "C5")
> z
   C1 C2 C3 C4 C5
R1  1  3  5  7  9
R2  2  4  6  8 10
```

Data Object - Matrices

- Reference Elements of a Matrix
  - Subsetting a Matrix

```
matrix [ row index, column index ]
```

    - Subsections of a matrix can be specified by giving the name of the matrix followed by index vectors in square brackets [].
    - Index vectors can be sequences of positive integers, negative integers, logical vectors or character vectors (only for named matrix), just like with vectors.
    - A blank means 'all of the rows' or 'all of the columns'.

  - Updating a Matrix

```
matrix [ row index, column index ] <- replacement values
```

Data Object - Matrices

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

Data Object - Matrices

```
> x <- matrix (1:12, 3, 4); x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> x[x[ , 2] != 4 & x[ , 4] != 12, ]
[1]  2  5  8 11
> x[ , x[ 1, ] %in% 1:5 ] <- 0
> x
     [,1] [,2] [,3] [,4]
[1,]    0    0    7   10
[2,]    0    0    8   11
[3,]    0    0    9   12
> x[1:2, c(2,4)] <- 21:24; x
     [,1] [,2] [,3] [,4]
[1,]    0   21    7   23
[2,]    0   22    8   24
[3,]    0    0    9   12
```

Data Object - Matrices

```
> Nmat <- matrix(1:20, ncol=5)
> rownames( Nmat ) <- c("r1", "r2", "r3", "r4")
> colnames( Nmat ) <-  c("c1", "c2", "c3", "c4", "c5")
> Nmat
   c1 c2 c3 c4 c5
r1  1  5  9 13 17
r2  2  6 10 14 18
r3  3  7 11 15 19
r4  4  8 12 16 20
> Nmat["r2", "c3"]
[1] 10
> Nmat[ Nmat[ ,"c2"] > 6 | Nmat[ ,"c4"] < 14, ]
   c1 c2 c3 c4 c5
r1  1  5  9 13 17
r3  3  7 11 15 19
r4  4  8 12 16 20
```

Data Object - Matrices

- Matrix Operations
  - Elementary and Summary Functions
    - The elementary (e.g. log, exp, sin, cos, sqrt) and summary functions (e.g. max, sum, mean) listed in the vector section work similarly with matri- ces.
  - Basic Operations on Matrices
    - Arithmetic, relational, logical operations(e.g. +, -, *, /, ^, !=, <, >=, &, |, !) on matrices is performed element by element. Therefore matrices must be conformable.
    - If a vector is used, recycling rule can be used.
  - Matrix Multiplication
    - Use the %*% operator. The matrices must be conformable.
    - If a vector is used in matrix multiplication, it will be coerced to either a row or column matrix to make the arguments conformable.
    - Using %*% on two vectors will return the inner product as a matrix not a scalar.

Data Object - Matrices

- Matrix Diagonals
  - diag(*x*): extracts or replaces the diagonal of a matrix, or constucts a diagonal matix
    - If *x* is a vector, it creates diagonal matrix with elements of *x* in the diagonal.
    - If *x* is a matrix, it returns a vector containing the elements of the diagonal.
    - If *x* is a scalar, this creates a *x* by *x* identity matrix.
- The "Apply" Mechanism
  - The function apply() can be used to compute row or column summaries.
    - apply(*matrix*, 1, *function*): computes row summaries of the type specified by *function* for the matrix specified by *matrix* .
    - apply(*matrix*, 2, *function*): computes column summaries, similarly.
  - It is also possible to use your own function directly.

Data Object - Matrices

- Other Useful Matrix Functions

Functions	Meaning
t(A)	transpose of A
det(A)	determinate of A
solve(A)	matrix inverse of A

```
> m1 <- matrix(1:6, nrow=3) ; m1
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> m2 <- matrix(16:11, nrow=3) ; m2
     [,1] [,2]
[1,]   16   13
[2,]   15   12
[3,]   14   11
> m1 + m2
     [,1] [,2]
[1,]   17   17
[2,]   17   17
[3,]   17   17
> m1 * m2
     [,1] [,2]
[1,]   16   52
[2,]   30   60
[3,]   42   66
```

Data Object - Matrices

```
> m1 >= m2
      [,1]  [,2]
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,] FALSE FALSE
> m2 < 15
      [,1] [,2]
[1,] FALSE TRUE
[2,] FALSE TRUE
[3,]  TRUE TRUE
> A <- matrix(1:4, nrow=2)
> B <- matrix(1, nrow=2, ncol=2)
> A %*% B
     [,1] [,2]
[1,]    4    4
[2,]    6    6
> Yvec <- 10:11
> A %*% Yvec
     [,1]
[1,]   43
[2,]   64
> Yvec %*% Yvec
     [,1]
[1,]  221
```

```
> m1                  > m2
     [,1] [,2]            [,1] [,2]
[1,]    1    4       [1,]   16   13
[2,]    2    5       [2,]   15   12
[3,]    3    6       [3,]   14   11
```

Data Object - Matrices

```
> diag(A)
[1] 1 4
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> diag(5:8)
     [,1] [,2] [,3] [,4]
[1,]    5    0    0    0
[2,]    0    6    0    0
[3,]    0    0    7    0
[4,]    0    0    0    8
> Appmat <- matrix (1:12, nrow=3, ncol=4)
> Appmat
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> apply(Appmat, 1, sum)
[1] 22 26 30
> apply(Appmat, 2, prod)
[1] 6  120  504 1320
```

Data Object - Matrices

```
> mat1 <- matrix(1:6, 2, 3)
> mat2 <- matrix(5:0, 2, 3)
> mat1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat2
     [,1] [,2] [,3]
[1,]    5    3    1
[2,]    4    2    0
> mat3 <- t(mat2) ; mat3
     [,1] [,2]
[1,]    5    4
[2,]    3    2
[3,]    1    0
> mat4 <- mat1 %*% mat3
> solve(mat4)
           [,1]       [,2]
[1,]  0.6666667 -0.4166667
[2,] -1.1666667  0.7916667
> mat5 <- mat3 %*% mat1
> det(mat5)
[1] 2.664535e-14
```

Data Object - Factors

- Denition
  - A factor is a special kind of vector object. We can tell R that a vector object is categorical data by making it a factor.
  - The set of all possible categories for a factor is called the levels of the factor.
- Creating Factors
  - factor(*vector*): creates a *vector* (character or numeric) as a factor
    - The factor stores the nominal values as a vector of integers in the range 1 to k (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the levels) mapped to these integers.
    - By default, R takes the levels to be the set of values occurring in the input vector, sorted into ascending order (either numerically or alphabetically).
    - levels=*levels vector* : to change the order of levels as you desire
    - labels=*labels vector* : to change the labels of the categories in the levels of a factor

Data Object - Factors

```
> x <- c("small", "medium", "large", "medium", "small")
> factor(x)
[1] small  medium large  medium small
Levels: large medium small
> z1 <- factor(x,levels=c("small","medium","large"))
> z1
[1] small  medium large  medium small
Levels: small medium large
> z2 <-factor(x,levels=c("small","medium","large"),
+ labels=c("S","M","L"))
> z2
[1] S M L M S
Levels: S M L
> z3 <- factor(x, labels=c("S","M","L") )
> z3
[1] L M S M L
Levels: S M L
```

Data Object - Factors

```
> data <- c(3,2,1,3,1,2,3,3,1,2,3,3,1)
> factor(data)
 [1] 3 2 1 3 1 2 3 3 1 2 3 3 1
Levels: 1 2 3
> factor(data, labels=c("I","II","III"))
 [1] III II  I   III I   II  III III I   II  III III I
Levels: I II III
> factor(data, levels=c(3,4,5))
 [1] 3    <NA> <NA> 3    <NA> <NA> 3    3    <NA> <NA> 3    3
[13] <NA>
Levels: 3 4 5
> factor(data, levels=c(0,1,2,3), labels=c("a","b","c","d"))
 [1] d c b d b c d d b c d d b
Levels: a b c d
```

Data Object - Factors

- Attributes of a Factor
  - Attributes of a Factor
    - mode(*factor*) : mode of factor
    - length(*factor*) : number of elements in factor
    - levels(*factor*) : the value of the levels of factor
- Reference Elements of a Factor
  - Generally follows the same rules as vector indexing.
- Useful Function for Utilizing Factors
  - Tabulation
    - table(*factor*) : To count the number of times each level occurs in factor.
    - table(*factor1*, *factor2*) : It is also possible to use table to count the number of times each combination of the levels of two factors (*factor1* and *factor2*) occurs. The resulting matrix is called a contingency table.

Data Object - Factors

```
> eyecol <- factor(c("hazel", "blue", "brown", "green", "blue",
"brown"))
> pain <- factor(c("low", "medium", "medium", "high", "medium",
"low"), levels=c("low", "medium", "high"))
> eyecol
[1] hazel blue  brown green blue  brown
Levels: blue brown green hazel
> pain
[1] low    medium medium high   medium low
Levels: low medium high
> mode( pain )
[1] "numeric"
> length( eyecol )
[1] 6
> levels( pain )
[1] "low"    "medium" "high"
```

Data Object - Factors

```
> table( pain )
pain
   low medium   high
     2      3      1
> table( eyecol, pain )
        pain
eyecol  low medium high
  blue    0      2    0
  brown   1      1    0
  green   0      0    1
  hazel   1      0    0`
```

Data Object - Lists

- **Definition**
  - A list is an object consisting of an ordered collection objects known as its components.
  - It is a general form of a vector, where the components don't need to be of the same type or dimension.
- **Creating Lists**
  - To create a list, use the command list().
  - list(*name_1=obj_1*, *name_2=obj_2*,..., *name_m =obj_m*): sets up a list of m components using *obj_1*, . . . , *obj_m* for the components and giving them names as specified by the argument names.
  - If these names are omitted, the components are numbered only.

Data Object - Lists

```
> L <- list( name=c("Chris","Barb"), age=c(25, 23),
+            haircolor=c("Brown", "Red"))
> L
$name
[1] "Chris" "Barb"

$age
[1] 25 23

$haircolor
[1] "Brown" "Red"
```

Data Object - Lists

```
> L2 <- list(c(62, 63, 67), matrix(1:12, nrow=3),
+            list( c("a", "b", "c"), TRUE ))
> L2
[[1]]
[1] 62 63 67

[[2]]
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

[[3]]
[[3]][[1]]
[1] "a" "b" "c"

[[3]][[2]]
[1] TRUE
```

Data Object - Lists

- Attributes of a List
  - Attributes of a List
    - mode(*list*) : returns "list"
    - length(*list*) : number of top-level components in *list*
    - names(*list*) : names of each top-level components in *list*

```
> LT <- list(name="fred", wife="mary", no.child=3,
+            child.ages=c(4, 7, 8))
> mode(LT)
[1] "list"
> length(LT)
[1] 4
> names(LT)
[1] "name"       "wife"       "no.child"   "child.ages"
```

Data Object - Lists

- Reference Elements of a List
  - Subsetting Lists

  To specify list components you can use one of the methods below.

```
List [[comp_index]]
List [[“comp_name”]]
List $ comp_name
```

    - Once you've specified the component, you can access parts of that component using the proper single [ ]or double [[ ]] branket method.
  - Updating a Matrix

```
List [[comp_index]]
List [[“comp_name”]] <- replacement values
List $ comp_name
```

Data Object - Lists

```
> L <- list( c(1,5,3), matrix(1:6, nrow=3), c("Hello", "world"))
> L
[[1]]
[1] 1 5 3

[[2]]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

[[3]]
[1] "Hello" "world"

> L[[1]]
[1] 1 5 3
> L[[2]][2,1]
[1] 2
```

Data Object - Lists

```
> L[[2]][1,1] <- 99
> L
[[1]]
[1] 1 5 3
[[2]]
     [,1] [,2]
[1,]   99    4
[2,]    2    5
[3,]    3    6
[[3]]
[1] "Hello" "world"
```

Data Object - Lists

```
> Ln <- list( v=c(1,5,3), m=matrix(1:6, nrow=3),
+             text=c("Hello", "world"))
> Ln
$v
[1] 1 5 3

$m
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

$text
[1] "Hello" "world"
> Ln$v
[1] 1 5 3
> Ln$m[2, 1]
[1] 2
> Ln[["text"]][-1]
[1] "world"
```

Data Object -DataFrames

- **Definition**
  - A dataframe is a special kind of a *list* which has a matrix-like structure where the columns can be of different types.
  - Each column (a vector or a factor object) is a *component* of the list and each component has the same length.
  - A dataframe is the fundamental data structure used by R 's statistical modeling functions.
- **Creating Dataframes**
  - To create a dataframe, use the data.frame() function.
  - data.frame(*name_1 =obj1*, *name_2 =obj2*,..., *name_m =obj_m*) : takes a number of objects *obj1,…, obj_m*, which are vectors or factors, returns a single object containing all objects as variables.

Data Object -DataFrames

```
> group <- data.frame(name=c("Hans", "Caro", "Lars", "Ines",
+                            "Samira", "Peter", "Sarah"),
+                     gender=c("male", "female", "male", "female",
+                              "female", "male", "female"),
+                     favourite.colour=c("green", "blue", "yellow",
+                                        "black", "yellow", "green",
+                                        "black"),
+                     income=c(800,1233,2400,4000,2899,1100,1900))
> group
    name gender favourite.colour income
1   Hans   male            green    800
2   Caro female             blue   1233
3   Lars   male           yellow   2400
4   Ines female            black   4000
5 Samira female           yellow   2899
6  Peter   male            green   1100
7  Sarah female            black   1900
```

Data Object -DataFrames

```
> str(group)
'data.frame':    7 obs. of  4 variables:
 $ name           : Factor w/ 7 levels "Caro","Hans",..: 2 1 4 3 6 5 7
 $ gender         : Factor w/ 2 levels "female","male": 2 1 2 1 1 2 1
 $ favourite.colour: Factor w/ 4 levels "black","blue",..: 3 2 4 1 4
 $ income         : num  800 1233 2400 4000 2899 ...
> summary(group)
     name       gender  favourite.colour     income
 Caro  :1   female:4   black :2         Min.   : 800
 Hans  :1   male  :3   blue  :1         1st Qu.:1166
 Ines  :1              green :2         Median :1900
 Lars  :1              yellow:2         Mean   :2047
 Peter :1                               3rd Qu.:2650
 Samira:1                               Max.   :4000
 Sarah :1
```

Data Object -DataFrames

- **Reading Data from the Files**
  - The simplest way to construct a dataframe is to use read.table() function to read an entire dataframe from an external file.

  - Important Options of read.table()

```
read.table("file", header=FALSE, sep="", ... )
```

    - file : the name of the file with a path name to read. To set the working directory, use setwd("*directory*") then you can avoid typing the file path.
    - header : logical. Does the first row contain column labels?
    - sep : the field separator character (e.g. "\t", ",")
    - na.string : a character vector of strings which are to be interpreted as NA values.
    - skip : number of lines to skip before reading data.

Data Object -DataFrames

```
> working.directory<-"C:/RExercise/"
> setwd( working.directory )
> data.life <- read.table("lifespan.csv", header=TRUE, sep=",",
+                         na.string=".")
> head( data.life )
  wghtcls smoker lifespan
1       3      0     50.3
2       3      0     52.8
3       3      1       NA
4       3      0     56.0
5       2      0     58.1
6       2      1     60.2
> data.ex <- read.table("example.csv", header=TRUE, sep=",", skip=7 )
> head( data.ex )
  ID      visit     trt PCS MCS    dr
1  1   3/5/2008 Placebo  88  71 Dr. A
2  1   9/7/2008 Placebo  67  68 Dr. A
3  1   9/7/2008 Placebo  67  68 Dr. A
4  2  6/19/2008    Drug  51  64 Dr. Y
5  2 12/22/2008    Drug  86  56 Dr. Y
6  3  4/11/2008 Placebo  85  59 Dr. K
```

Data Object -DataFrames

- Attributes of a Dataframe
  - Attributes of a Dataframe
    - mode(*dataframe*) : returns "list"
    - length(*dataframe*) : number of variables (columns) in *dataframe*
    - names(*dataframe*) : names of variables (columns) in *dataframe*
    - row.names(*dataframe*) : names of observations (rows) in *dataframe*

Data Object -DataFrames

```
> mode(data.life)
[1] "list"
> length(data.life)
[1] 3
> names(data.life)
[1] "wghtcls"  "smoker"   "lifespan"
> row.names(data.life)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
> rownames(data.life)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
> colnames(data.life)
[1] "wghtcls"  "smoker"   "lifespan"
> nrow(data.life)
[1] 12
> ncol(data.life)
[1] 3
```

Data Object -DataFrames

- **Subsetting Dataframes**

  Since a dataframe is similar to both a matrix and a list, it can be treated as either a matrix or a list when extracting components.

  - **A dataframe as a matrix** : you can think of a dataframe as a matrix with columns possibly of differing modes. It may be displayed in matrix form and its row and columns can be extracted using [] (matrix indexing conventions).
  - **A dataframe as a list** : each column of a dataframe is an component of the list and can be referenced using [[]] or $.

Data Object -DataFrames

```
> group
    name gender favourite.colour income
1   Hans   male            green    800
2   Caro female             blue   1233
3   Lars   male           yellow   2400
4   Ines female            black   4000
5 Samira female           yellow   2899
6  Peter   male            green   1100
7  Sarah female            black   1900
> group$income
[1]  800 1233 2400 4000 2899 1100 1900
> group$gender[2]
[1] female
Levels: female male
> group[, "income"]
[1]  800 1233 2400 4000 2899 1100 1900
> group[group$income==max(group$income), ]
  name gender favourite.colour income
4 Ines female            black   4000
> group$name
[1] Hans   Caro   Lars   Ines   Samira Peter  Sarah
Levels: Caro Hans Ines Lars Peter Samira Sarah
```

Data Object -DataFrames

```
> group[1,]
  name gender favourite.colour income
1 Hans   male            green    800
> group[, -2]
    name favourite.colour income
1   Hans            green    800
2   Caro             blue   1233
3   Lars           yellow   2400
4   Ines            black   4000
5 Samira           yellow   2899
6  Peter            green   1100
7  Sarah            black   1900
> group[group$name=="Hans",]
  name gender favourite.colour income
1 Hans   male            green    800
> group[group[["favourite.colour"]]=="yellow", ]
    name gender favourite.colour income
3   Lars   male           yellow   2400
5 Samira female           yellow   2899
```

```
> group
    name gender favourite.colour income
1   Hans   male            green    800
2   Caro female             blue   1233
3   Lars   male           yellow   2400
4   Ines female            black   4000
5 Samira female           yellow   2899
6  Peter   male            green   1100
7  Sarah female            black   1900
```

Data Object -DataFrames

```
> mean(group$income)
[1] 2047.429
> mean(group[group[["favourite.colour"]]=="yellow", "income"])
[1] 2649.5
> mean(group[group$favourite.colour %in% c("green", "blue", "black"),
"income"])
[1] 1806.6
```

```
> group
    name gender favourite.colour income
1   Hans   male            green    800
2   Caro female             blue   1233
3   Lars   male           yellow   2400
4   Ines female            black   4000
5 Samira female           yellow   2899
6  Peter   male            green   1100
7  Sarah female            black   1900
```

Testing and Coercing Data Objects

- Commands for Testing and Coercing R Data Objects
  - is.<*OBJ*>(*object*) : returns TRUE if *object* is <*OBJ*> and FALSE otherwise.
    - is.vector(*object*)
    - is.matrix(*object*)
    - is.factor(*object*)
    - is.list(*object*)
    - is.data.frame(*object*)

  - as.<*OBJ*>(*object*) : attempts to coerce *object* into <*OBJ*>.
    - as.vector(*object*)
    - as.matrix(*object*)
    - as.factor(*object*)
    - as.list(*object*)
    - as.data.frame(*object*)