Introduction to 

Surya Lamichhane

16 January 2024

Contents

Secttion 1 (Introduction to R)

Learning objectives

By the end of this lession, you will be able to

- Define R programming Languages and its use in data analytics
- Utilities various operators, objects, and data structures in R
- Subset data from R objects
- Perform various operations on R objects

Overview and History

R is a programming language and an environment for statistical computing, data visualization and statistical analysis.

- R is an open-source tool and is freely available to the public
- The R language has been developed as a successor of S
- Limitation of the S language was that it was only available in a commericial package, S-PLUS
- R was developed by Ross Ihaka and Robert Gentleman in the Department of Statistics at the University of Auckland in 1991
- In 1993 the first announcement of R was made to the public
- Ross's and Robert's experience developing R is documented in a 1996 paper in the Journal of Computational and Graphical Statistics
- All users have access to source code which enables users to study, modify, and redistribute the source code
- R has a worldwide repository system – CRAN
- The **Comprehensive R Archive Network (CRAN)**, is a network of sites that acts as the primary web service distributing R sources

Download and Install R

R works on every platform such as Windows, Mac OS X, and Linux systems. You can directly download R using the link **downlad R** from CRAN. If you want to watch a step-by-step tutorial on how to install R for Mac or Windows, you can check the following links:

- **Quick guide to install R for Windows**,
- **Quick guide to install R for Mac**,

Integrated Development Environment(IDE's)

An Integrated Development Environment, or IDE is a software that makes easier to write and work with packages. There are several IDEs available for R; however, RStudio is the most common and beginner-friendly for R users.

RStudio

- RStudio is a language-specific IDE built specially for R although it supports other languages. You can download Rstudio using the link **install RStudio**,
- Quick guide for the Rstudio can be found in **learn Rstudio basis**
- **R-markdown quick tour**.

Figure 1: Top data analytics tools

Top Programming Languages for data analytics and Data Science Why R is suitable for Data Analytics

- R is a free software
- Several algorithms for complex statistical models and Machine learnings are available
- R has some great visualization tools to tell the story about the data
- R code recogonized by other software such as SAS, Python
- R can handle semi-structured data and has built-in algorithms
- All users can define their customized algorithms in R and develop their own algorithms and packages

Use of R in Industry

R is one of the popular languages for a vast majority of Data Analysts and Data Scientists for:

- Data Visualization
- Data Manipulation
- Data Analysis and modeling
- Machine learning
- Deep learning

R is widely used in Academic research. Top companies such as Facebook, Google, Microsoft, also use R for various proposes.

A scenario of when to use R?

You have joined as an associate data scientist at Walmart. You required to create, store, and manage the daily sales data collected of a particular store. Since this data has to be statistically analyzed, the preferred tool to do so is R. Approach: To create and manage data in R, you need to understand the various data types, data structures, and operations in R.

```
You have joined as an associate data scientist at  Walmart. You required to create, store,
and manage the daily sales data collected of a particular store.
Since this data has to be statistically analyzed, the preferred tool to do so is R.
Approach: To create and manage data in R, you need to understand the various data types,
data structures, and operations in R.
```

Getting started with R Console vs. Editor

- The console window is a place where you run the command and you get the results; however, writing long program and fixing the potential error just by using it is really hard.
- R has text editor window which is very useful to write the programs (called R script), and save those codes for later.
- We can easily run the code line written in R editor by using **command + enter** in Macbook, and **ctrl + R** in window.

R Variables

Variables are containers for storing data values. R is a case-sensitive language, so while assigning a variable in R we need to consider the following:

- a combination of letters, digits, periods and underscores
- start with a letter or a period
- Reserved words in R cannot be used as variables
- Once assigned a value to the variable, R stored that value under that variable name until we quit R
- We can remove such a variable from R environment, by using *rm( )* function.

```r
#Good examples are
var_name1 = 1
.var_name = 5
var1 = 7
```

```r
#bad examples are
1var_name = 1
var@name = 5
var^ = 7
.12var = 3
```

```r
#bad examples are
TRUE = 1
FALSE = 5
T = 7
F = 4
```

- Try to give some meaningful name for the variable that you created

```r
first_prime_number = 2
print(first_prime_number)
```

```
## [1] 2
```

```r
value_of_pi = pi
print(value_of_pi)
```

```
## [1] 3.141593
```

Comment in R

Comments can be used to explain R code, and it make the code more understandable. It can also be used to prevent execution when testing alternative code.

Comments starts with a #. When executing code, R will ignore anything that starts with #.

- Always try to explain your code and procedure using comments.

```r
x = 7
y = 5
r = x %% y # results is the remainder while dividing 7 by 5
print(r)
```

```
## [1] 2
```

R Operators

R can handle different mathematical and logical operations. Operators in R can mainly be classified into the following categories.

- Assignment operators: To assign a variable for a given value(s) we use either of symbols = , <- and ->
- Arithmetic operators: Addition (+), subtraction (-), multiplication (*), division (/), exponentiation (^), modulus (%%), integer division (%/%)
- Relational operators: Less than (<), greater than (>), Less than or equal to (<=) , greater than or equal to (>=), equals (==), not equal (!=)
- Logical operators:
  ! Logical NOT, & Element-wise logical AND && Logical AND, | Element-wise logical OR, || Logical OR

Examples

```r
var1 = 27 # equal assignment
var2 <- 32 # left assignment
37 -> x # right assignment
```

Assignment operators The most common assignment operators are '=' and '<-'.

Arithmetic operators

1. Addition

```r
val1 <- 5
val2 <- 20
sum_two_val <- val1 + val2
print(sum_two_val)
```

```
## [1] 25
```

2. Subtraction

```r
val1 <- 15
val2 <- 20
diff_val <- val1 - val2
print(diff_val)
```

```
## [1] -5
```

3. Multiplication

```r
val1 <- 5
val2 <- 2
prod_val <- val1 * val2
print(prod_val)
```

```
## [1] 10
```

4. Division

```r
val1 <- 45
val2 <- 9
divide_val <- val1/val2
print(divide_val)
```

```
## [1] 5
```

5. Exponentiation

```r
x <- 4
print(x^3) # print x raise to power 3
```

```
## [1] 64
```

6. Modulus

The modulo operation of integers a and b. "a mod b" returns the remainder after dividing a by b. It is represented by 'a %% b' in R.

```r
int1 <- 25
int2 <- 7
remainder <- int1 %% int2
print(remainder) # the result is the remainder
```

```
## [1] 4
```

7. Integer division

Integer division of a and b is denoted by 'a %/%' in R, and it returns the quotient of the division.

```r
int1 <- 25
int2 <- 7
quotient <- int1 %/% int2
print(quotient) # the result is quotient
```

```
## [1] 3
```

Relational operators

```
operator    name            use
<         less than     compares two sides and returns TRUE if first object is less than second
<=        less or equal
>         greater than
!=        is not eqal   returns true if tow values are not equal
&         and           combines the correspondinng logical values of two vectors
&&        and           combines two logical values (Less useful in R)
|         or            combines the correspondinng logical values of two vectors
||        or            combines two logical values (Less useful in R)
```

Boolean Algebra

```
!TRUE = FALSE, !FALSE = TRUE
TRUE & TRUE = TRUE
TRUE & FALSE = FALSE
FALSE & FALSE = FALSE
TRUE | TRUE = TRUE
TRUE | FALSE = TRUE
FALSE | FALSE = FALSE
```

Examples

```r
x = c(1,3,4,8,4)
y = c(1,4,3,8,2)
x < y
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

```r
!(x < y)
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE
```

```r
x + 1 <= y
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

```r
(x < y) & (x + 1 <= y)
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

```r
(x < y) | (x + 1 <= y)
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE
```

Data Types in R

Our goal is to store the data in R and perform above mentioned operations of data objects, so first we need to know the data types in R. R has five basic classes of objects. We can find the data type in R by using the functions 'class( )', or 'typeof( )', (or checking by is.character( ), is.numeric( ) etc).

1. Character

The data type is a strings of letters, numbers, and other symbols or any value enclosed within single or double quotes.

```r
# example 1
string1 = "Hello world!"
print(string1)
```

```
## [1] "Hello world!"
```

```r
typeof(string1)
```

```
## [1] "character"
```

```r
class(string1)
```

```
## [1] "character"
```

```r
# example 2
Prog_languages = c('R', 'Python', 'Java', 'C', 'C+')
print(Prog_languages)
```

```
## [1] "R"      "Python" "Java"   "C"      "C+"
```

2. Numeric

Data object contains real numbers has numeric class.

```r
# example 1
num1 = 2.5
print(class(num1))
```

```
## [1] "numeric"
```

```r
# example 2
GPA_5students = c(3, 3.5, 3.8, 3.9, 4)
print(class(GPA_5students))
```

```
## [1] "numeric"
```

3. Integer

Data object contains only integer values.

```r
# example 1
val1 = 2L
print(class(val1))
```

```
## [1] "integer"
```

4. Complex

Complex number of type of a + b i, where $i = \sqrt{(-1)}$, an imaginary unit.

```r
complex_num <- 17 + 2i
print(class(complex_num))
```

```
## [1] "complex"
```

5. Logical

Logical values, usually used for comparisons: TRUE, FALSE, and NA (NA represents missing value or a value that does not exist).

```r
# example 1
num1 = 7
num2 = 8
compare1n2 = num1 == num2
print(compare1n2)
```

```
## [1] FALSE
```

```r
print(class(compare1n2 ))
```

```
## [1] "logical"
```

```r
# example 1
vec1 = c(3,7,5)
temp = (vec1 < 7)
print(class(temp))
```

```
## [1] "logical"
```

Data objects in R

The data values are stored in data objects. There are 6 types of objects in R Programming. They include vector, list, matrix, array, factor, and data frame. Vectors are one of the basic R programming data objects.

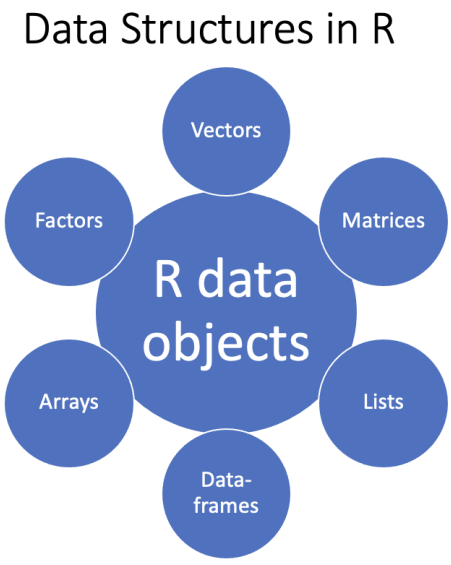


Figure 2: R-objects

1. Vectors

- Vectors are the most basic objects in R
- The c( ) function is used to create vectors of objects
- A vector can only store objects of the same class type data

Numeric vector

```
#  vector has numeric value
vector1 <- c(5, 2.1, 3, 9)
print(class(vector1))
```

```
## [1] "numeric"
```

Integer vector

```
#  vector has numeric value
vector2 = 1:5
print(class(vector2))
```

```
## [1] "integer"
```

Character vector

```
#  vector has character values
vector3 = c("John", "Arkansas", "A", "93.5")
print(class(vector3))
```

```
## [1] "character"
```

Logical vector

```
#  vector has logical values
vector4 = c(FALSE, TRUE, F, T)
print(class(vector4))
```

```
## [1] "logical"
```

Mixing data in a vector

When a vector is created with different types of objects (mixing objects), the data type of the objects is implicitly converted or coerced. New vector has all data values have same class.

```
#  mixing numeric value with integer
vector5 = c(2L, 4.1, 3.5)
print(class(vector5))
```

```
## [1] "numeric"
```

```
#  mixing numeric value with a character
vector6 = c("A", 4.1, 3.5)
print(class(vector6))
```

```
## [1] "character"
```

```r
#  mixing numeric value with logical value
vector7 = c(TRUE, FALSE,  4.1, 3.5)
print(class(vector7))
```

```
## [1] "numeric"
```

Typecasting in R

Typecasting is process of changing the data type of a variable. R Objects can be converted explicitly from one class to another using the "as." function.

```r
num_vec = c(1.2, 0.5, 2.1, 5.3, 7)
int_vec = as.integer(num_vec)
print(int_vec)
```

```
## [1] 1 0 2 5 7
```

```r
logic_vec = as.logical(int_vec)
print(logic_vec)
```

```
## [1]  TRUE FALSE  TRUE  TRUE  TRUE
```

Information about vectors

- three crucial attributes of vectors are: length, class and names
- length( ) function return the number of entries in the vector
- class( ) return type of the vector
- names ( ) return name of the vector if assigned

```r
student_info = c(Name = "John", State = "Arkansas", Grade = "A", Score = 93.5)
names(student_info)
```

```
## [1] "Name"  "State" "Grade" "Score"
```

```r
length(student_info)
```

```
## [1] 4
```

```r
class(student_info)
```

```
## [1] "character"
```

Note: If you want to remove the name of a variable, you can type names(vec_name) = NULL.

```r
student_info = c(Name = "John", State = "Arkansas", Grade = "A", Score = 93.5)
names(student_info) = NULL
print(student_info) # no names printed in the result
```

```
## [1] "John"     "Arkansas" "A"        "93.5"
```

Note: To assign name of a variable, you can type names(vec_name) = c(name vector that you required) as follows .

```r
names(student_info) = c('Name', 'State', "Grade", 'Score')
print(student_info) # names printed in the result
```

```
##       Name      State      Grade      Score
##     "John" "Arkansas"        "A"     "93.5"
```

2. Matrices

- Matrices are two dimensional array of numbers, it is simply a collection of numerical vectors
- Matrices in R can be created using the matrix( ) function
- matrix function has arguments matrix(data, nrow, ncol, byrow, dimnames)
- data: vector that inputs to construct a matrix
- nrow: number of rows of the matrix
- ncol: number of columns of the matrix
- byrow = TRUE if you want to fill data into matrix by row otherwise byrow = FALSE
- dimnames: Names vector for each dimension enclosed as a list

Example 1: Empty matrix with 4 rows and 3 columns

```r
Empty_matrix = matrix(nrow = 4, ncol = 3)
print(Empty_matrix)
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
## [4,]   NA   NA   NA
```

```r
dat = 1:12
matrix1 = matrix(data = dat, nrow = 4, ncol = 3)
print(matrix1)
```

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

Construct matrix row wise by using the " byrow = TRUE "

```r
dat = 1:12
matrix1 = matrix(data = dat, nrow = 4, ncol = 3, byrow = TRUE)
print(matrix1)
```

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

Matrix can be constructed by assigning the dim to the given vector

```r
dat = 1:12
dim(dat) = c(3, 4)
print(dat)
```

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

Matrix algebra review

- Dimension of matrices

The dimension (or size) of a matrix A is the number of rows follows by the number of columns of A. In R, the dimension of a Matrix A is obtained as dim(A), where the first entry is the number of rows and second entry represents as number of columns.

```r
cars_data = mtcars
dim(cars_data)
```

```
## [1] 32 11
```

- Number of rows and columns can be computed using the functions nrow( ) and ncol( ) respectively.

```r
# number of rows of the car data
nrow(cars_data)
```

```
## [1] 32
```

```r
# number of columns of the car data
ncol(cars_data)
```

```
## [1] 11
```

- Addition/subtraction

Two matrices can be added if and only if they have same dimension, and the sum is the sum the corresponding entries of two matrices.

```r
A = matrix(sample(20), nrow = 4, ncol = 5)
B = matrix(1:20, nrow = 4, ncol = 5)
dim(A) == dim(B)
```

```
## [1] TRUE TRUE
```

```r
print(A+B)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   17   25   24   23   21
## [2,]   21   20   15   15   29
## [3,]    6    9   28   24   27
## [4,]   22   15   25   22   32
```

```
print(A-B)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   15   15    6   -3  -13
## [2,]   17    8   -5  -13   -7
## [3,]    0   -5    6   -6  -11
## [4,]   14   -1    1  -10   -8
```

- Multiply a matrix by a number

Multiplication of a matrix A by a number k is denoted by k A (i.e. each entry of A is multiplied by k), it is computed in R by $k * A$.

```
A = matrix(sample(6), nrow = 3, ncol = 2)
5*A
```

```
##      [,1] [,2]
## [1,]   30    5
## [2,]   20   25
## [3,]   15   10
```

- Matrix products

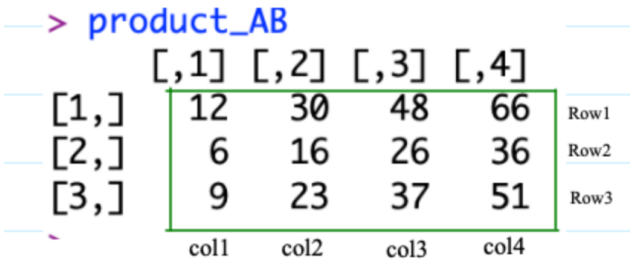
The product AB of two matrices A and B can be defined if number of columns of the first matrix A is equals the number of rows of the second matrix B (i.e. nrow(A) == ncol(B)). The $(ij)_{th}$ entry of the product AB is the sum of the product of the ith row of A and jth column of the matrix B.

a. If A is of size $m \times p$ and B is of size $p \times n$, then the product AB is of size $m \times n$.
b. The matrix product AB in R can be computed using A%*%B.

```
A = matrix(sample(6), nrow = 3, ncol = 2)
B = matrix(1:8, nrow = 2, ncol = 4)
ncol(A) == nrow(B)
```

```
## [1] TRUE
```

```
# prosduct matrix AB
product_AB = A %*% B
print(product_AB)
```



cbind( ) and rbind( ) functions

We can create matrices by binding columns and rows.

a. rbind( ): Binds vectors to create rows of the matrix
b. cbind( ): Binds vectors to create columns of the matrix

Example: There are there plants p1, p2, p3 and they produced two types of products A and B. Plant p1 produced 10 units of A and 7 units of B, plant p2 produced 8 units of A and 9 units of B, and plant p3 produced 6 units of A and 10 units of B. The selling price of A and B per unit are $200 and $180 respectively. What is the the revenue of each plant the company can obtain by selling from these two products?

```
p1 = c(10, 7)
p2 = c(8, 9)
p3 = c(6, 10)
prod_mat = cbind( p1, p2, p3)
colnames(prod_mat) = cbind("p1", "p2", "p3") #  assign column names
rownames(prod_mat) = c("price A", "price B") #  assign row names
print(prod_mat)
```

```
##         p1 p2 p3
## price A 10  8  6
## price B  7  9 10
```

```
price = c("price A" = 200, "price B" = 180)
rev = price %*% prod_mat
print(rev)
```

```
##        p1   p2   p3
## [1,] 3260 3220 3000
```

- Elementwise matrix multiplication

Element wise multiplication of two matrices A and B in R can be computed by $A * B$.

```
A = matrix(sample(6), nrow = 3, ncol = 2)
B = matrix(c(3,5,7, 1, 3, 2), nrow = 3, ncol = 2)
A * B
```

```
##      [,1] [,2]
## [1,]    6    6
## [2,]   15   12
## [3,]    7   10
```

- Transpose of a matrix

A matrix that is obtained by interchanging its rows and columns. For a given matrix A, its transpose is denoted by A^t or A' or A^T.

Note: The transpose of a matrix in R can be computed by t( )

```r
print(A)
```

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

```r
A_t = t(A)
print(A_t)
```

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

3. Arrays

Arrays are also the arrangement of vectors similar to matrices.

- Matrices have only rows and columns (two dimensional arrays)
- An array can have more than two dimensions
- Arrays are created using the array( ) function: *array(data, dim, dimnames)*.

```r
array1 <- array(1:24, dim = c(2,4,3) )
print(array1)
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    9   11   13   15
## [2,]   10   12   14   16
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]   17   19   21   23
## [2,]   18   20   22   24
```

4. Factors

Levels are the unique categorical values of the factor data

```r
# creating a factor data with responses as yes or no and creating levels automatically
response <- factor(c("yes", "yes", "no", "no", "yes", "yes", "yes"))
response
```

```
## [1] yes yes no  no  yes yes yes
## Levels: no yes
```

```
# creating a factor data with responses with predefined levels
response <- factor(c("agree", "disagree", "agree", "strongly agree", "disagree", "agree"),
                   levels = c("agree", "disagree", "neither agree nor disagree", "strongly agree", "str
response
```

```
## [1] agree          disagree       agree          strongly agree disagree
## [6] agree
## 5 Levels: agree disagree neither agree nor disagree ... strongly disagree
```

5. Dataframes

Dataframes are the most commonly used data objects in R for data analysis. Dataframe can take any type of vectors (including numeric, factor, character) in which the length of every element needs to be the same.

dataframes can be constructed by using functions data.frame( )

```
id <- c('A11', 'B12', 'C13', 'D14')
age <- c(21, 23, 20, 19)
response <- factor(c('yes', 'no', 'no', 'yes'))
df <- data.frame(id,age, response)
print(df)
```

```
##    id age response
## 1 A11  21      yes
## 2 B12  23       no
## 3 C13  20       no
## 4 D14  19      yes
```

6. Lists

List is another type of object in R programming. List can contain heterogeneous data types such as vectors with different length, matrices or another lists.

A list can be created using the list( ) function.

```
lst <- list( "A", 23.5, TRUE, c('John', 'Alex'), matrix(1:4,
nrow = 2, ncol =2))
print(lst)
```

```
## [[1]]
## [1] "A"
##
## [[2]]
## [1] 23.5
##
## [[3]]
## [1] TRUE
##
## [[4]]
```

```
## [1] "John" "Alex"
##
## [[5]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

Naming the data objects in R Objects in R can have names for individual elements, rows, and columns and list element.

a. Vector's name We can assign the name of elements of a vector in two ways:

- by adding value to the names attribute of a vector
- element's name can also be assigned while creating vector

```
# assign name for a given unnamed vector of top five states
pop_in_millons <-  c(39.03, 30.03, 22.24 ,19.68, 12.97)
top_five_state <- c("California", "Texas", "Florida", "New York", "Pennsylvania")
names(pop_in_millons) <- top_five_state
print(pop_in_millons)
```

```
##   California        Texas      Florida     New York Pennsylvania
##        39.03        30.03        22.24        19.68        12.97
```

```
# remove names
names(pop_in_millons) <- NULL
print(pop_in_millons)
```

```
## [1] 39.03 30.03 22.24 19.68 12.97
```

b. Row names and column names of a matrix can be created by using function *rownames( )* and *colnames( )*.

```
p1 = c(10, 7)
p2 = c(8, 9)
p3 = c(6, 10)
prod_mat = cbind( p1, p2, p3)
print(prod_mat)
```

```
##      p1 p2 p3
## [1,] 10  8  6
## [2,]  7  9 10
```

```
colnames(prod_mat) = cbind("p1", "p2", "p3") #  assign column names
rownames(prod_mat) = c("price A", "price B") #  assign row names
print(prod_mat)
```

```
##         p1 p2 p3
## price A 10  8  6
## price B  7  9 10
```

c. We can assign the name of list elements

```
student_info <- list(name = "Alex", ID = '1004392', course = c("Math-4225", "History-4145", "STAT 4101L
print(student_info)
```

```
## $name
## [1] "Alex"
##
## $ID
## [1] "1004392"
##
## $course
## [1] "Math-4225"    "History-4145" "STAT 4101L"
##
## $state
## [1] "TX"
```

Subsetting Data in R

Subsetting is retrieving parts of a data for specific purposes. There are a number of way of extracting subsets of R objects.

```
[ ]: square bracket with desired index
    - returns an object of the same class as original
    - by specifying vector of idices or names, it can be used to select more than one element
[[ ]]: double square bracket
    - returns the list or dataframe elements
$: dolar operator
    - returns the list or dataframe elements by name
    - works as similar as [[]]
```

a. Subset of vectors

```
vec = c(2, 5, 9, 20, 35, 12, 7)
#print first element of the vector
print(vec[1])
```

```
## [1] 2
```

```
# select 2nd, 3rd and 7th element
print(vec[c(2, 3, 7)])
```

```
## [1] 5 9 7
```

```
# print elements > 10
print(vec[vec > 10])
```

```
## [1] 20 35 12
```

b. Subset of Matrices

- A subset of a matrix can be derived by assigning the desired row number(s) and column number(s).
- submat_A = A[‘row position vector’, ‘column position vector’]
- submat_A = A[‘row name vector’, ‘column name vector’]
- A[, ‘column position vector’], all rows selected for the specified columns A[‘row position vector’,] , all columns selected for the specified rows

```
A = matrix(1:20, nrow = 5, ncol = 4, byrow = TRUE)
print(A)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
## [4,]   13   14   15   16
## [5,]   17   18   19   20
```

```
# print entry of fist row and first column
A11 = A[1, 1]
```

```
# print sub-matrix with row number 2,3,4 and columns 1,3, 4
sub_mat = A[2:4, c(1,3,4)]
print(sub_mat)
```

```
##      [,1] [,2] [,3]
## [1,]    5    7    8
## [2,]    9   11   12
## [3,]   13   15   16
```

Using row names and column names

```
mtcars = mtcars[1:5, 1:4]
print(head(mtcars))
```

```
##                   mpg cyl disp  hp
## Mazda RX4         21.0   6  160 110
## Mazda RX4 Wag     21.0   6  160 110
## Datsun 710        22.8   4  108  93
## Hornet 4 Drive    21.4   6  258 110
## Hornet Sportabout 18.7   8  360 175
```

```
sub_mat_mtcars = mtcars[c('Mazda RX4', 'Hornet 4 Drive') , c('mpg', 'hp')]
print(sub_mat_mtcars)
```

```
##                mpg  hp
## Mazda RX4      21.0 110
## Hornet 4 Drive 21.4 110
```

c. Subset of list objects

- To print data of list element list_name[[‘desired index’]] or list_name[[‘element’s name’]] or list_name$element_name.

- To print multiple list elements, we can use list_name[‘indices’]

```
print(student_info)
```

```
## $name
## [1] "Alex"
##
## $ID
## [1] "1004392"
##
## $course
## [1] "Math-4225"    "History-4145" "STAT 4101L"
##
## $state
## [1] "TX"
```

```
# print data of first list element with position
print(student_info[[1]])
```

```
## [1] "Alex"
```

```
# print data of first list element with name
print(student_info[["name"]])
```

```
## [1] "Alex"
```

```
# print data of first list element with name
print(student_info$name)
```

```
## [1] "Alex"
```

```
# print first two list elements
print(student_info[1:2])
```

```
## $name
## [1] "Alex"
##
## $ID
## [1] "1004392"
```

Missing values

- Missing values in R are represented by NA
- Undefined mathematical operations, such as division by zero, are represented by NaN (Not a Number)
- is.na( ) and is.nan( ) functions are used to test for NA and NaN respectively
- An NaN value is also an NA, but an NA value is not an NaN value

```
vec = c(10, 50, NaN, 100, 30, NA, 12)
is.na(vec)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE
```

```r
vec = c(10, 50, NaN, 100, 30, NA, 12)
is.nan(vec)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

Let's replace the missing value by the mean of the remaining values.

```r
vec = c(10, 50, NaN, 100, 30, NA, 12)
vec[is.na(vec)] = mean(vec, na.rm = TRUE)
print(vec)
```

```
## [1]  10.0  50.0  40.4 100.0  30.0  40.4  12.0
```