Introduction to R

Saeed Saffari Saeed.Saffari@dal.ca

Winter 2026

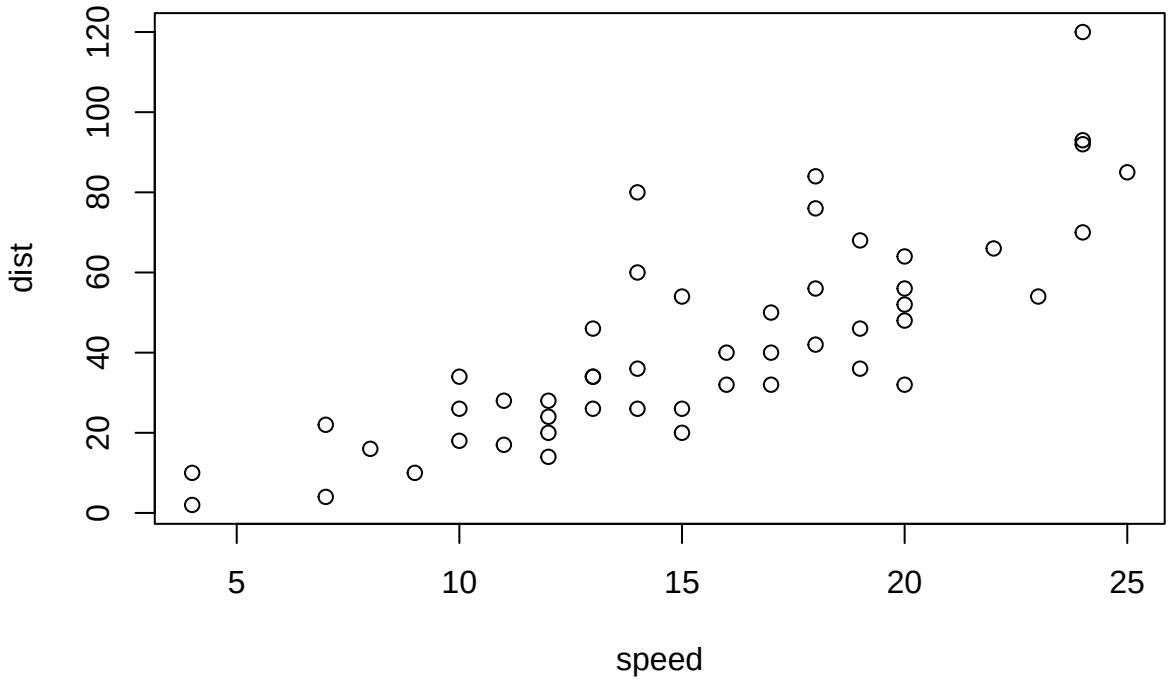
Contents

1 Introduction

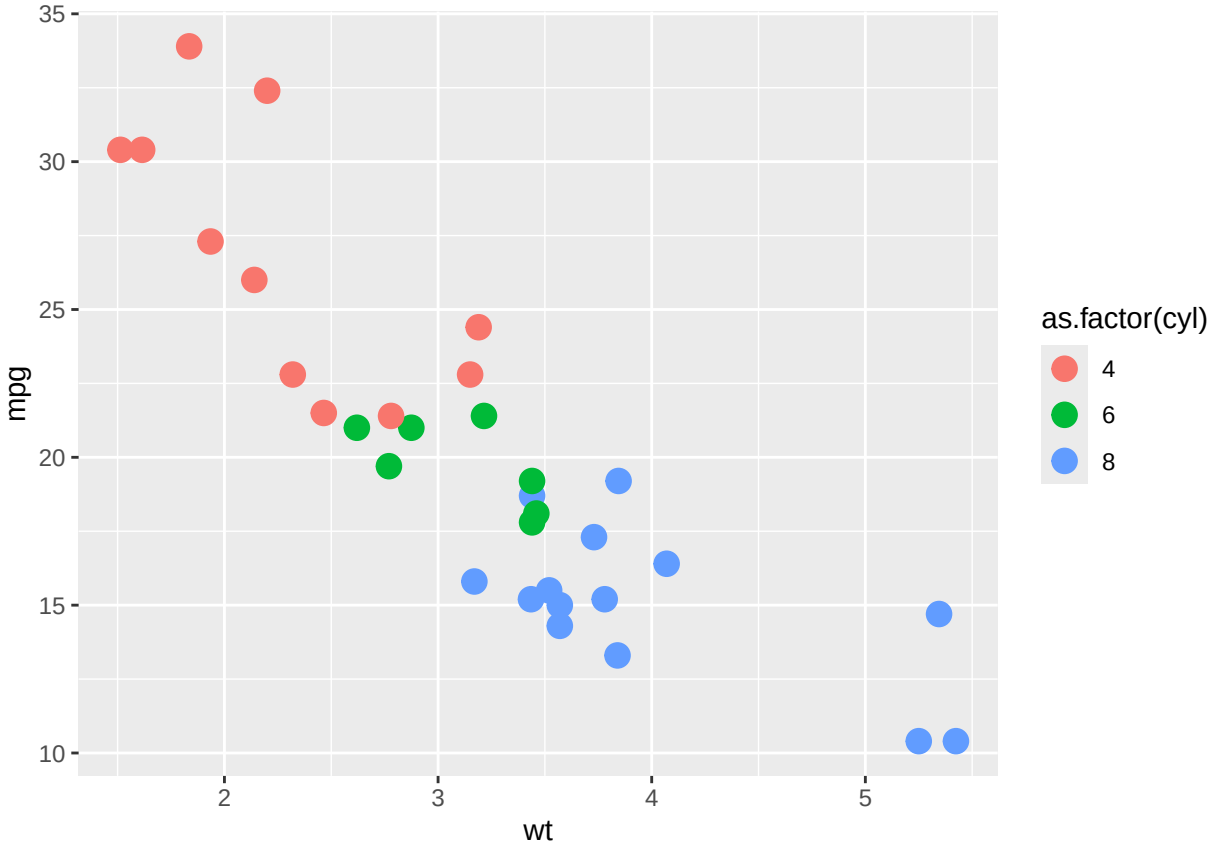
This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter* (on Mac) or *Ctrl+Shift+Enter* (on Windows).

```
plot(cars)
```




```
library(ggplot2)
ggplot(data=mtcars, aes(x=wt, y=mpg)) +
  geom_point(aes(colour = as.factor(cyl)), size=4)
```



Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I* (on Mac) or *Ctrl+Alt+I* (on Windows).

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

You can download and install packages with `install.packages("The name of package")`.

1.1 Markdown:

$$\alpha = \beta = \frac{-\alpha \times \theta^3}{\sqrt{2 \times \theta}}$$

2 Basic of learning

In this part we talk about basic elements of R programming.

2.1 How to Print

```
print('This is R programming course!')
```

```
## [1] "This is R programming course!"
```

2.2 Data Types (Classes)

In R, data types (or classes) define the kind of data stored in variables. Here are the most common data types in R:

Class	Description	Example	Code Example
`numeric`	Represents decimal or whole numbers	3.14, 42, -7.8	`x <- 3.14; class(x)`
`character`	Represents text strings	"Hello", "R Programming"	`z <- "Hello"; class(z)`
`logical`	Represents Boolean values (TRUE or FALSE)	TRUE, FALSE	`is_valid <- TRUE; class(is_valid)`
`complex`	Represents complex numbers	2+3i, 1-4i	`c <- 2 + 3i; class(c)`
`list`	Represents a collection of different types	A list of numbers, text	`lst <- list(1, "Hello", TRUE); class(lst)`

```
print(class(3.14))
```

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

```
print(class('R programming'))
```

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

```
print(class(TRUE))
```

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

2.3 Arithmetic Operators

Symbol	Task Performed
+	Addition
-	Subtraction
/	division
*	multiplication
**	to the power of
^	to the power of
%%	modulus
%/%	floor division

```
18 + 4
```

```
## [1] 22
```

```r
18 - 4
```

```
## [1] 14
```

```r
18 * 4
```

```
## [1] 72
```

```r
10 / 2
```

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

```r
2 ** 3
```

```
## [1] 8
```

```r
9 ** 0.5
```

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

```r
log(2)
```

```
## [1] 0.6931472
```

```r
log10(2)
```

```
## [1] 0.30103
```

```r
14 %% 4
```

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

```r
14 %/% 4
```

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

```r
5 + (4 - 3 * 2)**3 + 1
```

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

In R programming, code runs line by line, with only the last assignment determining the final value of a variable.

```r
x <- 18
x <- 10
x <- x * 2 - 3
x
```

```
## [1] 17
```

We can save values in variables:

```r
x <- 18
class(x)
```

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

```r
a = 'R programming'
class(a)
```

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

2.4 Relational Operators

Symbol	Task Performed
<-	Assignment
=	Assignment
assign()	Assignment
==	True, if it is equal
!=	True, if not equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to

```
z <- 10
y = 6
assign('x', 2)

x < y
```

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

```
x >= y
```

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

```
x != y
```

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

```
x == y
```

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

```
b = 2
x <= b
```

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

```
x > 1 & y < 10
```

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

```
x > 1 & y > 10
```

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

```
x > 1 | y > 10
```

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

you can use below command to get special values:

```
x <- pi
x
```

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

```
e = exp(1)
e
```

```
## [1] 2.718282
```

```
x = letters
x
```

```
##  [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"
```

```
x <- LETTERS
x
```

```
##  [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"
```

```
x <- month.name
x
```

```
##  [1] "January"   "February"  "March"     "April"     "May"       "June"
##  [7] "July"      "August"    "September" "October"   "November"  "December"
```

```
x <- month.abb
x
```

```
##  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

you can write comment with # :

```
# This is line is comment!

r = 0.2    # interest rate
```

you can creat sequence numbers with below command:

This work like *arange* in numpy pakage in Python

```
x <- 1:10
x
```

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

```
x <- 1:10 * 2
x
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
x <- seq(from=1, to=9, by=3)
x
```

```
## [1] 1 4 7
```

```
x <- seq(1, 10, by=4)
x
```

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

This work like *linspace* in numpy pakage in Python

```
x <- seq(1, 10, length=3)
x
```

```
## [1]  1.0  5.5 10.0
```

Replicate function:

```
x <- 1:3
x
```

```
## [1] 1 2 3
```

```r
y <- rep(x, time=5)
y
```

```
##  [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```r
y <- rep(x, each=5)
y
```

```
##  [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

2.4.1 Practice:

Simulate GDP growth from the year 2000 to 2025 with an annual growth rate of 3% starting from 1000 units.

$$GDP_t = 1000 \cdot (1 + r)^n$$

```r
years = 0:25
growth_rate = 0.03 # 3% annual rate
GDP <- 1000 * (1+growth_rate)^years
print(GDP)
```

```
##  [1] 1000.000 1030.000 1060.900 1092.727 1125.509 1159.274 1194.052 1229.874
##  [9] 1266.770 1304.773 1343.916 1384.234 1425.761 1468.534 1512.590 1557.967
## [17] 1604.706 1652.848 1702.433 1753.506 1806.111 1860.295 1916.103 1973.587
## [25] 2032.794 2093.778
```

```r
years = 2000:2025
growth_rate = 0.03 # 3% annual rate
GDP <- 1000 * (1+growth_rate)^(years-2000)
print(GDP)
```

```
##  [1] 1000.000 1030.000 1060.900 1092.727 1125.509 1159.274 1194.052 1229.874
##  [9] 1266.770 1304.773 1343.916 1384.234 1425.761 1468.534 1512.590 1557.967
## [17] 1604.706 1652.848 1702.433 1753.506 1806.111 1860.295 1916.103 1973.587
## [25] 2032.794 2093.778
```

2.5 Loops

2.5.1 if , elif

```r
age <- 14

if (age >= 18){
  print('You are old enough to vote!')
} else {
  print('You can NOT vote yet!')
  print(paste('You can will vote after', 18 - age, "years."))
}
```

```
## [1] "You can NOT vote yet!"
## [1] "You can will vote after 4 years."
```

```r
age = 16

if (age <= 4){
  price = 0
```

```
} else if (age < 16){
  price = 5
} else {
#} else if (age >= 16){
  price = 10
}

print(paste("Your cost is $", price,'.'))
```

```
## [1] "Your cost is $ 10 ."
```

2.5.2 for loops

```
for (i in 1:5){
  print(i**i)

}
```

```
## [1] 1
## [1] 4
## [1] 27
## [1] 256
## [1] 3125
```

```
z = 0
for (i in 1:10){
  z = z + i
  print(z)
}
```

```
## [1] 1
## [1] 3
## [1] 6
## [1] 10
## [1] 15
## [1] 21
## [1] 28
## [1] 36
## [1] 45
## [1] 55
```

2.5.3 while loops

```
i <- 1

while (i <= 10){
  print(i)
  i <- i + 1
}
```

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

```
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

Use break and next in loops

```
for (i in 1:10){
  if (i==5){
    break
  }
  print(i)
}
```

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

```
for (i in 1:10){
  if (i==5){
    next
  }
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

2.5.4 Practice:

Write a conditional statement to check whether the variable is positive, negative, or zero and print the appropriate message.

```
x = 0
if (x > 0){
  print("Positive")
} else if (x < 0){
  print('negative')
} else {
  print("Zero")
}
```

```
## [1] "Zero"
```

2.6 function

```
average = function(a, b, c){
  summ = a + b + c
```

```r
  ave = summ / 3
  return(ave)
}

average(34, 13, -21)
```

```
## [1] 8.666667
```

$$K_n = K \times (1 + r)^n$$

```r
future_value = function(k, r, n){
  k_n = k * (1+r)^n
  return(k_n)
}
```

```r
future_value(1000, 0.05, 10)
```

```
## [1] 1628.895
```

```r
future_value(500, 0.10, 5)
```

```
## [1] 805.255
```

2.6.1 Practice I:

Write a function `temp()` that converts a temperature in Celsius to Fahrenheit.

$$F = (C \times \frac{9}{5}) + 32$$

2.6.2 Practice II:

Write a function in R that takes a number as input and returns whether the number is even or odd.

3 Vetors

The most common way to creat vectors is to use function `c()`.

```
x <- c(10.25, 3.5, 8.75, 25, 12)
x
```

```
## [1] 10.25  3.50  8.75 25.00 12.00
```

```
x <- c('a', "b", "C")
x
```

```
## [1] "a" "b" "C"
```

```
x <- c(10.25, 3.5, 8.75, 25, 12, "A", 'b', "C")
x
```

```
## [1] "10.25" "3.5"   "8.75"  "25"    "12"    "A"     "b"     "C"
```

```
class(x)
```

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

Join vectors

```
x <- c(10,20,30,40)
y <- c(3.5, 4.75)

z <- c(x, y)
z
```

```
## [1] 10.00 20.00 30.00 40.00  3.50  4.75
```

You can find *lenght* of vectors with *lenght()* function:

```
x <- c(10.25, 3.5, 8.75, 25, 12)
x
```

```
## [1] 10.25  3.50  8.75 25.00 12.00
```

```
length(x)
```

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

```
length(z)
```

```
## [1] 6
```

3.1 Vector Indexing

```
x
```

```
## [1] 10.25  3.50  8.75 25.00 12.00
```

```
for (i in x){
  print(i)
}
```

```
## [1] 10.25
## [1] 3.5
## [1] 8.75
## [1] 25
## [1] 12
```

Use for loops for access to elemets of vectors

```
for (i in 1:length(x)){
  print(x[i])
}
```

```
## [1] 10.25
## [1] 3.5
## [1] 8.75
## [1] 25
## [1] 12
```

Use index:

```
x
```

```
## [1] 10.25  3.50  8.75 25.00 12.00
```

```
x[2]
```

```
## [1] 3.5
```

```
x[-2]
```

```
## [1] 10.25  8.75 25.00 12.00
```

```
x[1:3]
```

```
## [1] 10.25  3.50  8.75
```

```
x[c(1, 3, 4)]
```

```
## [1] 10.25  8.75 25.00
```

```
x[2] = -18
```

```
x
```

```
## [1]  10.25 -18.00   8.75  25.00  12.00
```

```
x[-3]
```

```
## [1]  10.25 -18.00  25.00  12.00
```

```
x[-3] = 0
x
```

```
## [1] 0.00 0.00 8.75 0.00 0.00
```

```
x <- c(10.25, 3.5, 8.75, 25, 12)
x
```

```
## [1] 10.25  3.50  8.75 25.00 12.00
```

```
y <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
x[y]
```

```
## [1] 10.25  8.75 25.00
```

3.2 Matching Operator

```
x <- c(10,45,30,56,40,80)
x
```

```
## [1] 10 45 30 56 40 80
```

```
30 %in% x
```

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

```
37 %in% x
```

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

```
y <- c(30, 37, 40)
y %in% x
```

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

3.3 Vector Arithmetic's

```
x
```

```
## [1] 10 45 30 56 40 80
```

```
x + 2
```

```
## [1] 12 47 32 58 42 82
```

```
x = x * 2
x
```

```
## [1]  20  90  60 112  80 160
```

```
sqrt(x)
```

```
## [1]  4.472136  9.486833  7.745967 10.583005  8.944272 12.649111
```

```
x <- c(10,45,30,50)
y <- c(5,1,3,4)

x + y
```

```
## [1] 15 46 33 54
```

```
z <- c(10,20,30)
x + z
```

```
## Warning in x + z: longer object length is not a multiple of shorter object
## length
```

```
## [1] 20 65 60 60
```

3.4 Vector Methods

```
x <- c(10,45,30,50)
length(x)
```

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

```
sum(x)
```

```
## [1] 135
```

```
prod(x)
```

```
## [1] 675000
```

```
rev(x)
```

```
## [1] 50 30 45 10
```

```
sort(x, decreasing = TRUE)
```

```
## [1] 50 45 30 10
```

```
x <- c(10,45,30,50)
y <- c(5,1,3,4)

x %*% y
```

```
##      [,1]
## [1,]  385
```

3.5 Logical Vector

```
x
```

```
## [1] 10 45 30 50
```

```
y <- x > 30 & x < 50
y
```

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

```
x[y]
```

```
## [1] 45
```

```
x[which((x>30))]
```

```
## [1] 45 50
```

3.6 Factors

- Used to represent categorical data
- Treated as integer vector, having a label
- Factors are self describing

```
x <- c('Male', "Female", "Male", 'Male', "Female")
x
```

```
## [1] "Male"   "Female" "Male"   "Male"   "Female"
```

```
x <- factor(c('Male', "Female", "Male", 'Male', "Female"))
x
```

```
## [1] Male   Female Male   Male   Female
## Levels: Female Male
```

```
table(x)
```

```
## x
## Female   Male
##      2      3
```

3.7 Mathematical Function in R

```
x <- c(4.235, -3.548, 5.324, 7.892)
x
```

```
## [1]  4.235 -3.548  5.324  7.892
```

```
abs(x)
```

```
## [1] 4.235 3.548 5.324 7.892
```

```
ceiling(x)  # next integer
```

```
## [1]  5 -3  6  8
```

```
floor(x)  # smaller integer
```

```
## [1]  4 -4  5  7
```

```
round(x)
```

```
## [1]  4 -4  5  8
```

```
round(x, digits = 2)
```

```
## [1]  4.24 -3.55  5.32  7.89
```

```
sqrt(abs(x))
```

```
## [1] 2.057912 1.883614 2.307379 2.809270
```

```
log10(abs(x))
```

```
## [1] 0.6268534 0.5499836 0.7262380 0.8971871
```

3.8 Random Number in R

```
x <- rnorm(1000)
#x
```

```
mean(x)
```

```
## [1] -0.06296946
```

```
sd(x)
```

```
## [1] 0.9763518
```

3.9 Practice:

Given a list of students (`"Alice", "Bob", "Charlie", "David", "Eve"`) and their corresponding scores (`85, 92, 78, 55, 88`), extract the names and scores of students who passed (`score >= 60`). Also, calculate the mean score of the students who passed.

```
students <- c("Alice", "Bob", "Charlie", "David", "Eve")
scores <- c(85, 92, 78, 55, 88)

passed <- scores >= 60

scores[passed]
```

```
## [1] 85 92 78 88
```

```
students[passed]
```

```
## [1] "Alice"   "Bob"     "Charlie" "Eve"
```

```
mean(scores[passed])
```

```
## [1] 85.75
```

4 Matrix

4.1 Creat Matrix

Matrix are 2-dimentsional vectors and Dimensianal attribute is of lenght 2 (rows and columns). We should to know that Matrix contain elemts of same type.

```
m = matrix(nrow = 2, ncol = 3)
m
```

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

```
dim(m)
```

```
## [1] 2 3
```

```
m <- matrix(c(1,2,3,4,5,6))
m
```

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

```
m <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
m
```

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

```
m <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3, byrow = TRUE)
m
```

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

```
m <- matrix(seq(from = 1, to = 40, by = 2), nrow = 4, ncol = 5, byrow = TRUE)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]   11   13   15   17   19
## [3,]   21   23   25   27   29
## [4,]   31   33   35   37   39
```

```
dim(m)
```

```
## [1] 4 5
```

```
nrow(m)
```

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

```
ncol(m)
```

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

```
length(m)
```

```
## [1] 20
```

4.2 Matrix diag

like `numpy.full` in python

```
m <- matrix(4, 3,3)
m
```

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

like `numpy.diag` in python

```
m <- diag(1, 3,3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

```
diag(4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```
diag(1:5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    0    0    0
## [2,]    0    2    0    0    0
## [3,]    0    0    3    0    0
## [4,]    0    0    0    4    0
## [5,]    0    0    0    0    5
```

for find the elements of diagonal of matrix:

```
m <- matrix(seq(from = 1, to = 40, by = 2), nrow = 4, ncol = 5, byrow = TRUE)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]   11   13   15   17   19
## [3,]   21   23   25   27   29
## [4,]   31   33   35   37   39
```

```
diag(m)
```

```
## [1]  1 13 25 37
```

4.3 Matrix: Naming Rows & Columns

```
m <- matrix(seq(from = 1, to = 40, by = 2), nrow = 4, ncol = 5, byrow = TRUE)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]   11   13   15   17   19
## [3,]   21   23   25   27   29
## [4,]   31   33   35   37   39
```

```
rownames(m) = c('A', "b", 'F', 'S')
colnames(m) = c('B', "W", 'X', 'S', "L")
```

```
m
```

```
##    B  W  X  S  L
## A  1  3  5  7  9
## b 11 13 15 17 19
## F 21 23 25 27 29
## S 31 33 35 37 39
```

4.4 Matrix Indexing

Indexing in R programming is similar to Python.

```
m <- matrix(seq(from = 1, to = 40, by = 2), nrow = 4, ncol = 5, byrow = TRUE)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]   11   13   15   17   19
## [3,]   21   23   25   27   29
## [4,]   31   33   35   37   39
```

```
m[1,] # for get single row
```

```
## [1] 1 3 5 7 9
```

```
m[,3] # for get single column
```

```
## [1]  5 15 25 35
```

```
m[2,3]
```

```
## [1] 15
```

```
m[2,2:4]
```

```
## [1] 13 15 17
```

```
m[1:3,2:4]
```

```
##      [,1] [,2] [,3]
## [1,]    3    5    7
## [2,]   13   15   17
## [3,]   23   25   27
```

```
m[,-2]
```

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

```
## [1,]    1    5    7    9
## [2,]   11   15   17   19
## [3,]   21   25   27   29
## [4,]   31   35   37   39
```

```
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]   11   13   15   17   19
## [3,]   21   23   25   27   29
## [4,]   31   33   35   37   39
```

You can change values in matrix.

```
m[2,3] = 0
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]   11   13    0   17   19
## [3,]   21   23   25   27   29
## [4,]   31   33   35   37   39
```

4.5 Matrix: `rbine()` and `cbind()` functions

You can combine matrices with `rbine()` and `cbind()` functions.

At first, we want to combine the matrices from the row.

```
A <- matrix(c(1,2,3,4,5,6,8,9,1) , nrow=3, ncol=3, byrow=TRUE)
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    8    9    1
```

```
B <- rbind(A, c(10,11,12))
B
```

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

After that, we want to combine the matrices from the columns.

```
C <- cbind(A, c(10,11,12))
C
```

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

Relational Operators in Matrics:

```
A <- matrix(c(1,2,3,4,5,6,8,9,1) , nrow=3, ncol=3, byrow=TRUE)
B <- matrix(c(3,1,2,4,2,1,5,1,2), nrow=3, ncol=3, byrow=TRUE)
```

```r
A
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    8    9    1
```

```r
B
```

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

```r
A + B
```

```
##      [,1] [,2] [,3]
## [1,]    4    3    5
## [2,]    8    7    7
## [3,]   13   10    3
```

```r
A - B
```

```
##      [,1] [,2] [,3]
## [1,]   -2    1    1
## [2,]    0    3    5
## [3,]    3    8   -1
```

```r
A * B
```

```
##      [,1] [,2] [,3]
## [1,]    3    2    6
## [2,]   16   10    6
## [3,]   40    9    2
```

```r
A / B
```

```
##           [,1] [,2] [,3]
## [1,] 0.3333333  2.0  1.5
## [2,] 1.0000000  2.5  6.0
## [3,] 1.6000000  9.0  0.5
```

```r
A %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   26    8   10
## [2,]   62   20   25
## [3,]   65   27   27
```

Like `numpy.transpose()` or `.T` in python

```r
A <- matrix(c(1,2,3,4,5,6,8,9,1,4,2,3) , nrow=3, ncol=4, byrow=TRUE)

t(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    1
## [2,]    2    6    4
## [3,]    3    8    2
## [4,]    4    9    3
```

4.6 Matrix Specific Functions

```
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    8    9
## [3,]    1    4    2    3
```

```
rowSums(A)
```

```
## [1] 10 28 10
```

```
colSums((A))
```

```
## [1]  7 12 13 16
```

```
rowMeans((A))
```

```
## [1] 2.5 7.0 2.5
```

```
colMeans(A)
```

```
## [1] 2.333333 4.000000 4.333333 5.333333
```

4.6.1 Practice I

Create a 4x4 matrix of random integers between 1 and 100.

- Print the matrix.
- Calculate the row-wise sum and column-wise mean.
- Check if the matrix is symmetric by comparing it to its transpose.

5 Lists

5.1 Creat list

`Lists` are also collecting of data and another kind of data storage. Lists can contain elemnts of any type of R object and these elements of list don't need be same type. You can creat list by using `list()` function.

Creat list with vectors

```
classno <- c(101,102,103)
name <- c("Sanaz", "Saeed", "Sarah")
scores <- c(98.45, 45.65, 78.79)
```