Introduction to R

Saeed Saffari Saeed.Saffari@dal.ca

Winter 2025

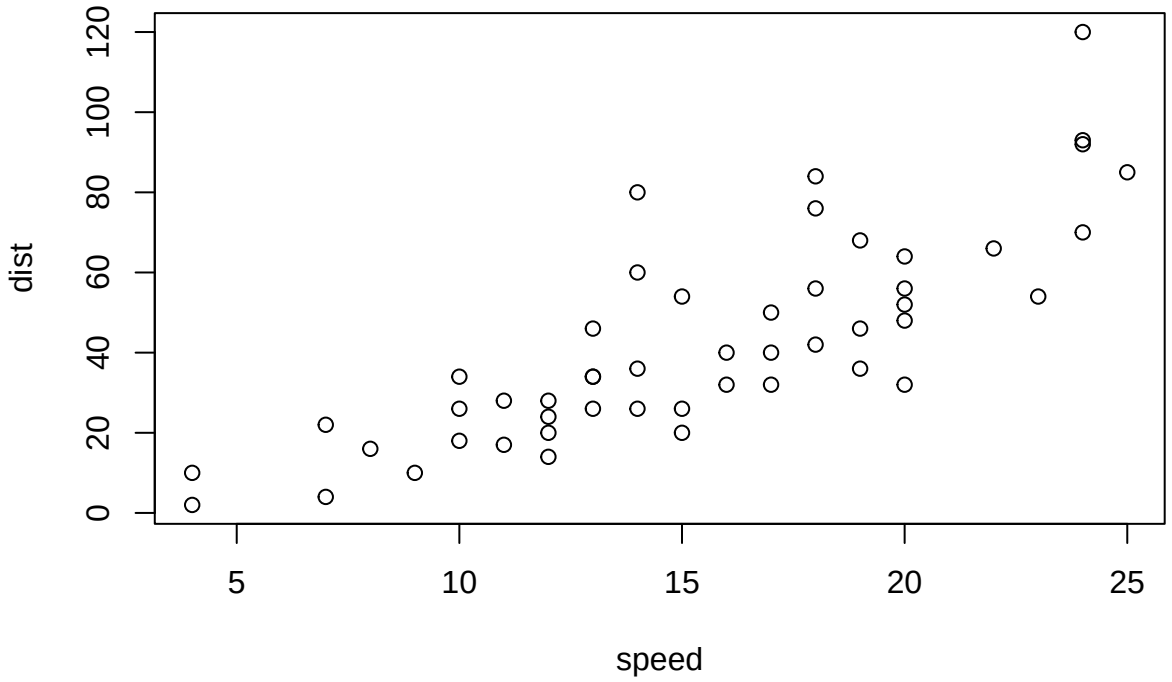
Contents

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).

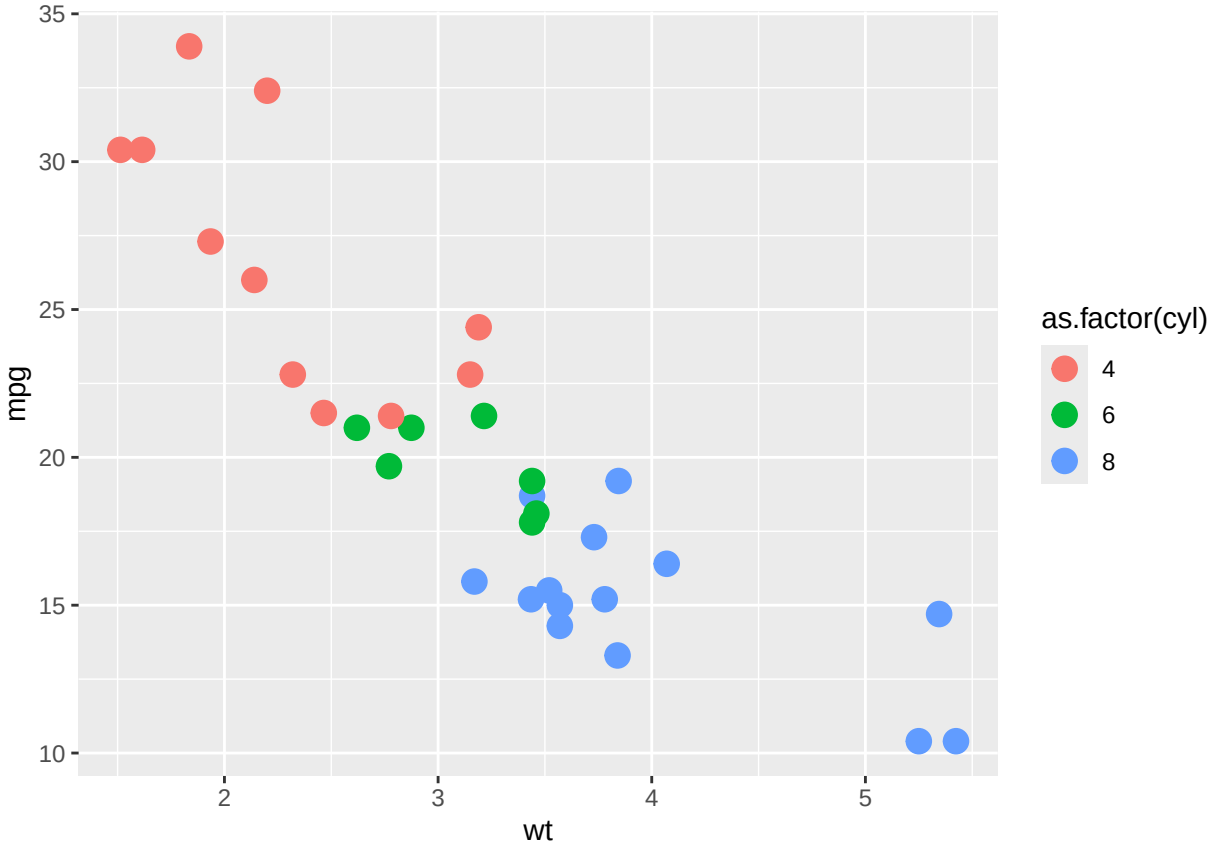
```
plot(cars)
```




```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
data("mtcars")
ggplot(data = mtcars, aes(x = wt, y= mpg)) +
  geom_point(aes(color=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:

$$\alpha = \beta = \frac{-\alpha \cdot \gamma^2}{\sqrt{\sigma^3}}$$

2 Basic of learning

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

2.1 How to Print

```
print('Hello world')
```

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

```
print("This is R programming Workshop!")
```

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

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

```r
18 + 4
```

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

```r
18 - 4
```

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

```r
18 * 4
```

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

```r
18 / 4
```

```
## [1] 4.5
```

```r
2 ** 3
```

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

```r
2 ^ 3
```

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

```r
18 %% 4
```

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

```r
18 %/% 4
```

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

```r
log(2)
```

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

```r
log10(2)
```

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

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

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

We can save values in variables:

```r
x <- 18
y = 4
z <- x + y
print(z)
```

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

```r
class(z)
```

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

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

```r
a = 5 + (4 - 3 * 2)**3 + 1
a = 10
a = a * 2
a = a - 5
a
```

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

2.3.1 Practice:

Convert a given temperature X degrees Celsius to Fahrenheit.

$$F = C \cdot \frac{9}{5} + 32$$

```
temp = 20
fahrenheit <- (temp * 9 / 5) + 32
print(fahrenheit)
```

```
## [1] 68
```

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
```

```
x > 2
```

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

```
x >= 2
```

```
## [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 line is comment!
r = 0.2 # interest rate
```

you can create 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(5)
x
```

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

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

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

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

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

```
x <- seq(1,10,2)
x
```

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

This work like *linspace* in numpy pakage in Python

```
x <- seq(1,10,length = 5)
x
```

```
## [1]  1.00  3.25  5.50  7.75 10.00
```

Replicate function:

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

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

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

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

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

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

2.5 Practice:

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

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

```
years = 2000:2025
growth_rate <- 0.03 # rate
GDP = 1000 * (1+growth_rate)**(years - 2000)
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
```

2.6 Loops

2.6.1 if , elif

```
#"R" + 2
```

```
age <- 15
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  3  years."
```

```
age <- 16
if (age <= 4){
  price = 0
} else if (age < 16){
  price = 50
} else {
#} else if (age >= 16){
  price = 100
}

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

```
## [1] "Your cost is $ 100"
```

2.6.2 for loops

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

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

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

```
## [1] 1
## [1] 4
## [1] 9
## [1] 16
## [1] 25
```

```
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
```

2.6.3 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
```

2.6.4 Practice:

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

```
"It's R"
```

```
## [1] "It's R"
```

```
x <- -5
if (x > 0) {
  print('Positive')
} else if (x < 0){
  print("Negative")
} else {
  print('Zero')
}
```

```
## [1] "Negative"
```

2.7 function

```
average = function(a,b,c){
  summ = a + b + c
  ave = summ / 3
  return(ave)
}
```

```r
average(34,12,-23) * 2
```

```
## [1] 15.33333
```

```r
average(23,56,98)
```

```
## [1] 59
```

```r
price_func = function(age){
if (age <= 4){
  price = 0
} else if (age < 16){
  price = 50
} else {
  price = 100
}
print(paste("Your cost is $", price))
}

price_func(18)
```

```
## [1] "Your cost is $ 100"
```

```r
for (i in seq(1:20)){
  #print(i)
  price_func(i)
}
```

```
## [1] "Your cost is $ 0"
## [1] "Your cost is $ 0"
## [1] "Your cost is $ 0"
## [1] "Your cost is $ 0"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 50"
## [1] "Your cost is $ 100"
## [1] "Your cost is $ 100"
## [1] "Your cost is $ 100"
## [1] "Your cost is $ 100"
## [1] "Your cost is $ 100"
```

2.7.1 Practice I:

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

```r
temp <- function(temps){
  fahrenheit <- (temps * 9 / 5) + 32
  print(fahrenheit)
```

```
}
temp(90)
```

```
## [1] 194
```

2.7.2 Practice II:

Create a function to calculate the **future value** K_n of an investment after `n` years with a given principal K and interest rate r.

The formula for compound interest is:

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

```
future_value <- function(k, r, n){
  k_n <- k * (1+r)^n # future value formula
  return(k_n)
}
future_value(1000, 0.05, 10)
```

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

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

```
## [1] 628.8946
```

2.7.3 Practice III:

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 create vectors is to use function `c()`.

```
x <- c(10.25, 3.5, 8.75, 23.15, 12)
x
```

```
## [1] 10.25  3.50  8.75 23.15 12.00
```

```
x <- c(10.25, 3.5, 8.75, 23.15, 12, 'a', 'b', "c")
x
```

```
## [1] "10.25" "3.5"   "8.75"  "23.15" "12"    "a"     "b"     "c"
```

```
class(x)
```

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

```
x <- c(1,2,3,4,5,6,7)
x
```

```
## [1] 1 2 3 4 5 6 7
```

```
y <- 1:7
y
```

```
## [1] 1 2 3 4 5 6 7
```

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(1.5, 3.25, 8.75, 13.15)
x
```

```
## [1]  1.50  3.25  8.75 13.15
```

```
length(x)
```

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

3.1 Vector Indexing

```
x <- c(10,45,30,50,35,50,80)
x
```

```
## [1] 10 45 30 50 35 50 80
```

```
x[1]
```

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

```
x[3]
```

```
## [1] 30
```

```
x[-2]
```

```
## [1] 10 30 50 35 50 80
```

```
x[3:6]
```

```
## [1] 30 50 35 50
```

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

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

```
length(x)
```

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

```
x[10]
```

```
## [1] NA
```

```
x
```

```
## [1] 10 45 30 50 35 50 80
```

```
x[2]
```

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

```
x[2] <- -8
x
```

```
## [1] 10 -8 30 50 35 50 80
```

```
x[10] = 20
x
```

```
##  [1] 10 -8 30 50 35 50 80 NA NA 20
```

```
x[-3] = 6
x
```

```
##  [1]  6  6 30  6  6  6  6  6  6  6
```

```
x
```

```
##  [1]  6  6 30  6  6  6  6  6  6  6
```

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

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

Use for loops for access to elemets of vectors

```
for (i in x){
  print(x*2)
}
```

```
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
```

```
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
##  [1] 12 12 60 12 12 12 12 12 12 12
```

3.2 Matching Operator

```
x <- c(10,45,30,50,35,50,80)
x
```

```
## [1] 10 45 30 50 35 50 80
```

```
35 %in% x
```

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

```
37 %in% x
```

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

```
y <- c(30, 37, 45)
y %in% x
```

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

3.3 Vector Arithmetic's

```
x <- c(10,45,30,50,35,50,80)
x
```

```
## [1] 10 45 30 50 35 50 80
```

```
x + 2
```

```
## [1] 12 47 32 52 37 52 82
```

```
x * 2
```

```
## [1]  20  90  60 100  70 100 160
```

```
sqrt(x)
```

```
## [1] 3.162278 6.708204 5.477226 7.071068 5.916080 7.071068 8.944272
```

```
x <- c(10,45,30,50)
y <- c(5,1,2,4)

x + y
```

```
## [1] 15 46 32 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)
x
```

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

```
length(x)
```

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

```
sum(x)
```

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

```
mean(x)
```

```
## [1] 33.75
```

```
prod(x)
```

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

```
rev(x)
```

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

```
sort(x)
```

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

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

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

3.5 Logical Vector

```
x <- c(10,45,30,50,35)
x
```

```
## [1] 10 45 30 50 35
```

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

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

```
x[y]
```

```
## [1] 45 35
```

```
x <- c(10,45,30,50,35)
x
```

```
## [1] 10 45 30 50 35
```

```
which(x>30)
```

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

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

```
## [1] 45 50 35
```

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(x)
x
```

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

```
table(x)
```

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

3.7 Mathematical Function in R

3.8 Random Number in R

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)
```

4 Matrix

4.1 Creat Matrix

Matrix are 2-dimensional vectors and dimensional attribute is of length 2 (rows and columns). We should to know that Matrix contain elements of same type.

4.2 Matrix diag

like `numpy.full` in python

like `numpy.diag` in python

for find the elements of diagonal of matrix:

4.3 Matrix Indexing

Indexing in R programming is similar to Python.

You can change values in matrix.

4.4 Matrix Specific Functions

4.5 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.

```
set.seed(123)
mat <- matrix(sample(1:100, 16), nrow = 4)
```

4.6 Practice II

Given a 3x3 matrix A of integers and a vector b = (30, 20, 15), solve the linear equation $A \cdot X = b$.

```
set.seed(123)
A <- matrix(sample(1:10, 9, replace = TRUE), nrow = 3)
b <- c(30, 20, 15)
```

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

5.2 List subset Operator

6 Dataframe

Dataframes are objects in R and used to store tabular data. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length. Dataframe can be created using `data.frame()` function or imported from various file types.

- 'read.table()"
- 'read.csv()"

6.1 Creating Dataframes

```
id <- c(101,102,103, 104, 105)
names <- c("Sanaz", "Saeed", "James", "Peter", "Emma")
scores <- c(98.45, 45.65, 78.79, 56.32, 87.23)
students <- data.frame(id, names, scores)
```

6.2 Dataframes Indexing

6.3 Dataframes `subset()` function for filtering

6.4 Dataframes `rbine()` and `cbind()`

add rows

add columns

6.5 Saving data in csv

6.6 Missing Data

In this part we find out how handle a missing data like `NA`.

This function is like `.isnull()` in python programming.

```
x <- c(10,4,NA,7,15,NaN)
```

Remove missing values

```
weather <- data.frame(
  id = c(101, 102, 103, 104, 105),
  temperature = c(25.8, 34.2, NA, 27.4, 20.5),
  wind = c(78, 59, 63, 40, 68),
  humidity = c(25, 45, 85, NA, 61)
)
```

7 Dplyr Package

You can download and install packages with install.packages("The name of package"). In this case, run `install.packages('dplyr')` to download and install `dplyr` package.

Also, you can import packages in R with `library()` function.

Data is imported in to dataframes using: `read.csv()`

`read.csv()` arguments:

- **file:** name of the file (mandatory argument)
- **header:** logical value (default is false)
- **sep:** separator (default is comma (`,`))

you can see head or tail of data with below function. It's work like `.head()` and `.tail()` in Pandas package in python.

like `.shape` in pandas package in python

like `.describe()` in pandas package in python for understand structure of data:

7.1 dplyr `select()` function

Select special columns

Select with number of columns:

Select with name of columns:

for get names of columns, you can use below function. This work like `.columns` in pandas package in Python.

Also you can select range of columns:

You can drop columns with use minus sign (`-`) in `select` function.

7.2 dplyr `filter()` function

dplyr `arrange()` function

7.3 dplyr `rename()` function

7.4 dplyr `mutate()` function

7.4.1 Practice:

Import the data and Create a new column called murder_rate that shows the murder rate (murders per million people)

7.5 dplyr `group_by()` function

7.6 dplyr Pipe Operator %>%

7.6.1 Practice:

Find the states with a population greater than 10 million and murders greater than 500. Display only the state, population, and murder columns.

8 Data Visualization with `dplyr`

8.1 Bar Graphs

Horizontsl Bar Graphs

8.2 Histogram

8.3 Scatter Plots

The default of `plot` functions is Scatter plot.

pch values

Values of pch are stored internally as integers.

Figure 1: Some values of pch

8.4 Line Graphs

`type` in `plot` function

- **"p"** for points,
- **"l"** for lines,
- **"b"** for both points and lines,
- **"c"** for empty points joined by lines,
- **"o"** for overplotted points and lines,
- **"s"** and **"S"** for stair steps,
- **"h"** for histogram-like vertical lines,
- **"n"** does not produce any points or lines.

8.5 Box plots

8.6 Multiple Plots in Layout

9 Regressions and Models

9.1 Simple Linear Regression

Linear regression is used to model the relationship between a dependent variable and one or more independent variables by fitting a line through the data.

9.2 Multiple linear regression

Multiple regression extends linear regression by adding more independent variables to better predict the dependent variable.

9.2.0.1 Practice: Add `region` (as a categorical variable) to the multiple regression model.

9.2.1 t-test (Comparing Means)

A t-test compares the means of two groups to determine if they are statistically different.

9.2.2 Correlation Test

A correlation test checks the strength and direction of the relationship between two variables.

9.2.2.1 Practice

9.2.3 ANOVA (Analysis of Variance)

ANOVA tests if there are significant differences between the means of three or more groups.

9.2.4 Regression Plots