DSAI_M1

Leonhard Stransky

2023-09-19

Data Science "Einführung in relevante Softwareumgebungen''

Einführung:

Die Theorieeinheiten sollen als Hintergrundinformation zur Durchführung einer eigenen Analyse dienen. Zum Umgang mit R und RStudio oder Python ist der Link aus den Theorie Einheiten empfohlen. Als Vorlage für die Laborprotokolle steht euch im Kurs ein Template für R Markdown zur Verfügung.

Ziele:

Das Ziel ist es, DataCamp als Anleitung und R als Werkzeug zur Datenexploration und Visualisierung selbständig einsetzen und erste Ergebnisse interpretieren zu können.

Aufgabe 1(GK):

Erarbeite dir durch den Kurs Einführung in R in den Kapiteln 2 bis 4 die Grundlagen der Sprache Programmiersprache R und vertiefe dich dann in Visualisierung mit der Library 'ggplot' in Kapitel 5. Fasse zusätzlich die Inhalte von w3schools der Kapitel "Tutorial" und "Data Structures" (GK)

Führe dabei ein Protokoll in einem Markdown File über Informationen, Dokumentation und Codes als Nachschlagemöglichkeit, da dir der Kurs nur für 6 Monate zur Verfügung stehen wird.

2. Die R Sprache:

Übung1:

```
1/3 * ((1+3+5+7+2)/(3+5+4))
```

```
## [1] 0.5
```

```
exp(1)
```

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

```
sqrt(2)
```

```
## [1] 1.414214
```

```
8^(1/3)
```

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

```
log2(8)
```

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

Übung2:

1. Erzeugen Sie eine Sequenz von 0 bis 100 in 5-er Schritten.
2. Berechnen Sie den Mittelwert des Vektors [1, 3, 4, 7, 11, 2].
3. Lassen Sie sich die Spannweite x(max)-x(min) dieses Vektors ausgeben.
4. Berechnen Sie die Summe dieses Vektors.
5. Zentrieren Sie diesen Vektor.
6. Simulieren Sie einen Münzwurf mit der Funktion sample(). Tipp: nehmen Sie für Kopf 1 und für Zahl 0. Simulieren Sie 100 Münzwürfe.
7. Simulieren Sie eine Trick-Münze mit P!=0.5
8. Generieren Sie einen Vektor, der aus 100 Wiederholungen der Zahl 3 besteht.

```
seq(from = 0, to = 100, by = 5)
```

```
##  [1]   0   5  10  15  20  25  30  35  40  45  50  55  60  65  70  75  80  85  90
## [20]  95 100
```

```
mean(c(1,3,4,7,11,2))
```

```
## [1] 4.666667
```

```
b <- range(c(1,3,4,7,11,2))
b[2] - b[1]
```

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

```
sum(c(1,3,4,7,11,2))
```

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

```
scale(c(1,3,4,7,11,2), center = TRUE, scale = FALSE)
```

```
##             [,1]
## [1,] -3.6666667
## [2,] -1.6666667
## [3,] -0.6666667
## [4,]  2.3333333
## [5,]  6.3333333
```

```
## [6,] -2.6666667
## attr(,"scaled:center")
## [1] 4.666667
```

```
sample(c(0,1), size = 100, replace = TRUE)
```

```
##   [1] 0 1 0 1 0 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 1 1 0 0 0 1 1 0 0 1 1 0 1 1 1 0 1
##  [38] 1 1 0 0 1 0 0 0 0 0 1 1 1 0 1 1 1 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0
##  [75] 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1
```

```
sample(c(0,1,1,1,1), size = 100, replace = TRUE)
```

```
##   [1] 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 1 1
##  [38] 1 1 0 0 1 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1 1
##  [75] 0 1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1
```

```
rep(3, times = 100)
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

Übung3:

Wiederholen Sie einige der Beispiele von oben, aber speichern Sie diesmal die Resultate in neuen Variablen ab. Sie können die Variablen dann für weitere Operationen wieder verwenden.

```
## [1] 4.666667
```

```
## [1] 4.7
```

Übung4:

Tippen Sie in der Konsole scale(und drücken Sie TAB. Sie sehen, dass diese Funktion drei Argumente hat, x, center und scale. Schreiben Sie scale(vektor, scale = gefolgt von TAB. Sie sehen, dass scale = TRUE den default Wert TRUE hat.

Was sind die Argumente der Funktion round()? Hat eines der Argumente einen default Wert?

Schauen Sie im Help Viewer nach, was die Funktion rnorm() macht. Was sind die Argumente. Was bedeuten die default Werte?

Schauen Sie im Help Viewer nach, welche Argumente die seq() Funktion hat.

Was machen folgende Funktionsaufrufe?

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

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

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

```
##  [1]  1.000000  1.473684  1.947368  2.421053  2.894737  3.368421  3.842105
##  [8]  4.315789  4.789474  5.263158  5.736842  6.210526  6.684211  7.157895
## [15]  7.631579  8.105263  8.578947  9.052632  9.526316 10.000000
```

round:

Bsp: round(x, digits = 1)

x ist die Zahl oder der Vektor

digits ist die Anzahl der Dezimalstellen auf die gerundet werden soll

Die Verwendung von Digits ist optional und hat einen default wert von 0.

rnorm: Dies wird verwendet um Zufallszahlen aus einer Normalverteilung zu generieren.

Bsp: rnorm(n, mean = 0, sd = 1)

n ist die Anzahl der Zufallszahlen. (erfordert)

mean ist der Durchschnitt (Mittelwert) der Normalverteilung.

sd ist die Standardabweichung der Normalverteilung.

mean und sd sind optionale Argumente. mean hat als default wert 0 sund sd 1 (Dies enstpspricht einer Standardnormalverteilung)

seq: Dies wird verwendet um reguläre Sequenzen von Zahlen zu generieren.

Bsp: seq()´

seq(from, to, by, length.out, along.with, . . .)

from ist der Startwert der Sequenz. (optional) (default wert: 1)

to: Der Endwert der Sequenz. (optional) (default wert: 1)

by: Der Inkrementwert, um den die Sequenz erhöht wird. (optional) default wert wird automatisch berechnet, um die gewünschte Länge der Sequenz length.out zu erreichen.

length.out: Die gewünschte Länge der Sequenz. (optional) default wert ist NULL. Wenn length.out nicht angegeben wird, wird die Länge der Sequenz basierend auf den anderen Argumenten berechnet.

along.with: Übernimmt die Länge der Sequenz von einem anderen Vektor oder einer anderen Sequenz. (optional)

Character Vektors

```
letters[1:3]
```

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

```
text <- c("these are", "some strings")
text
```

```
## [1] "these are"    "some strings"
```

```
length(text)
```

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

```
vorname <- "Ronald Aylmer"
nachname <- "Fisher"
paste("Mein Name ist:", vorname, nachname, sep = " ")
```

```
## [1] "Mein Name ist: Ronald Aylmer Fisher"
```

Logical vectors

Logische Vektoren werden vor allem dazu benutzt, um numerische Vektoren zu indizieren, z.B. um alle positiven Elemente eines Vektors auszuwählen.

```
x <- rnorm(24)
x
```

```
##  [1] -2.98137975 -0.58848020  0.63474512 -0.88840363  1.05270702  0.27074613
##  [7]  0.53331336  1.45152935  0.73839878 -0.68153889 -2.23127207  1.02505559
## [13] -1.37897839 -1.47051238 -1.14275658  0.03673759  0.11147475  1.46694827
## [19]  0.50115812 -0.93985061 -0.49044433  0.70634814  0.50441673 -0.76797491
```

```
x > 0
```

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

```
mean(x)
```

```
## [1] -0.1886672
```

man kann auch bei dieses Vektors den Mittelwert bestimmen

Factors

```
geschlecht <- c("male", "female", "male", "male", "female")
geschlecht
```

```
## [1] "male"   "female" "male"   "male"   "female"
```

```
typeof(geschlecht)
```

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

```
geschlecht <- factor(geschlecht, levels = c("female", "male"))
geschlecht
```

```
## [1] male   female male   male   female
## Levels: female male
```

```
typeof(geschlecht)
```

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

Hier kann man sehen das durch das Anwenden der methode Faktor man verschhiedene level bestimmen kann –> so werden die Werte zu Integern und geschlecht zu der Klasse Faktor Mit factor() kann man die Reihenfolge genau bestimmen - es müssen aber alle Stufen explizit angegeben werden.

Lists

```
x <- list(1:3, "a", c(TRUE, FALSE, TRUE), c(2.3, 5.9))
x
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1]  TRUE FALSE  TRUE
##
## [[4]]
## [1] 2.3 5.9
```

```
x[2]
```

```
## [[1]]
## [1] "a"
```

```
b <- list(int = 1:3,
          string = "a",
          log = c(TRUE, FALSE, TRUE),
          double = c(2.3, 5.9))
b
```

```
## $int
## [1] 1 2 3
##
## $string
## [1] "a"
##
## $log
## [1]  TRUE FALSE  TRUE
##
## $double
## [1] 2.3 5.9
```

Data Frames

Nun kommen wir zu dem für uns wichtigsten Objekt in R, dem Data Frame. Datensätze werden in R durch Data Frames repräsentiert. Ein Data Frame besteht aus Zeilen (rows) und Spalten (columns). Technisch gesehen ist ein Data Frame eine Liste, deren Elemente gleichlange (equal-length) Vektoren sind.

```
library(dplyr)
```

```
##
## Attache Paket: 'dplyr'
```

```
## Die folgenden Objekte sind maskiert von 'package:stats':
##
##     filter, lag
```

```
## Die folgenden Objekte sind maskiert von 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
df <- tibble(geschlecht = factor(c("male", "female",
                                   "male", "male",
                                   "female")),
             alter = c(22, 45, 33, 27, 30))
df
```

```
## # A tibble: 5 x 2
##   geschlecht alter
##   <fct>      <dbl>
## 1 male          22
## 2 female        45
## 3 male          33
## 4 male          27
## 5 female        30
```

```
attributes(df)
```

```
## $class
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
##
## $row.names
## [1] 1 2 3 4 5
##
## $names
## [1] "geschlecht" "alter"
```

W3schools:

R Tutorial:

```
# This is a comment
"Hello World!"
```

R Comments:

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

```
name <- "John"
age <- 40

name   # output "John"
```

R Variables:

```
## [1] "John"
```

```
age    # output 40
```

```
## [1] 40
```

```
# Concatenate Elements:
text <- "awesome"

paste("R is", text)
```

```
## [1] "R is awesome"
```

```
# Multiple Variables:

# Assign the same value to multiple variables in one line:
var1 <- var2 <- var3 <- "Orange"

# Print variable values:
var1
```

```
## [1] "Orange"
```

```
var2
```

```
## [1] "Orange"
```

```
var3
```

```
## [1] "Orange"
```

```
# Variabel Names:

# Legal variable names:
```

```r
myvar <- "John"
my_var <- "John"
myVar <- "John"
MYVAR <- "John"
myvar2 <- "John"
.myvar <- "John"

# Illegal variable names:
# 2myvar <- "John"
# my-var <- "John"
# my var <- "John"
# _my_var <- "John"
# my_v@ar <- "John"
# TRUE <- "John"
```

R Data Types:

1. numeric - (10.5, 55, 787)
2. integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
3. complex - (9 + 3i, where "i" is the imaginary part)
4. character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
5. logical (a.k.a. boolean) - (TRUE or FALSE)

```r
# numeric
x <- 10.5
class(x)
```

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

```r
# integer
x <- 1000L
class(x)
```

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

```r
# complex
x <- 9i + 3
class(x)
```

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

```r
# character/string
x <- "R is exciting"
class(x)
```

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

```r
# logical/boolean
x <- TRUE
class(x)
```

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

```r

# R Numbers:
x <- 10.5    # numeric
y <- 10L     # integer
z <- 1i      # complex
```

```r
# Type Conversion:
as.numeric()
```

R Numbers:

```
## numeric(0)
```

```r
as.integer()
```

```
## integer(0)
```

```r
as.complex()
```

```
## complex(0)
```

```r
max(5, 10, 15)
```

R Math:

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

```r
min(5, 10, 15)
```

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

```r
sqrt(16)
```

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

```r
abs(-4.7) # Returns positive Value
```

```
## [1] 4.7
```

```r
ceiling(1.4) # Rounds the number up
```

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

```r
floor(1.4) # Rounds the number down
```

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

```r
str <- "Hello"
str # print the value of str
```

R Strings:

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

```r
# For multiple lines use ,
str <- "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."

str # print the value of str
```

```
## [1] "Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit,\nsed do eiusmod tempor incididunt\nut
```

```r
cat(str) # To set the line brakes at the same position as the code
```

```
## Lorem ipsum dolor sit amet,
## consectetur adipiscing elit,
## sed do eiusmod tempor incididunt
## ut labore et dolore magna aliqua.
```

```
nchar(str) # To get the length of a string
```

```
## [1] 123
```

```
# For finding characters and sequences
str <- "Hello World!"

grepl("H", str)
```

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

```
grepl("Hello", str)
```

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

```
grepl("X", str)
```

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

Escape Symbols:

1. \ Backslash
2. \ New Line
3. Carriage Return
4. Tab
5. Backspace

R Operators: Arithmetic Operators:

1. • Addition
2. • Subtraction
3. • Multiplication
4. / Division
5. Exponent
6. %% Modulus (Remainder from division)
7. %/% Integer Division

Assignment Operators:

```
my_var <- 3

my_var <<- 3

3 -> my_var

3 ->> my_var
```

```
my_var # print my_var
```

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

Comparison Operators:

1. == Equal
2. != Not equal
3. Greater than
4. < Less than
5. = Greater than or equal to
6. <= Less than or equal to

Logical Operators:

1. & Element-wise Logical AND operator. It returns TRUE if both elements are TRUE
2. && Logical AND operator - Returns TRUE if both statements are TRUE
3. Elementwise- Logical OR operator. It returns TRUE if one of the statement is TRUE
4. || Logical OR operator. It returns TRUE if one of the statement is TRUE.
5. ! Logical NOT - returns FALSE if statement is TRUE

Miscellaneous Operators:

1. : Creates a series of numbers in a sequence
2. %in% Find out if an element belongs to a vector
3. %*% Matrix Multiplication

```
str <- "Hello"
str # print the value of str
```

R Functions:

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

```
# For multiple lines use ,
str <- "Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."

str # print the value of str
```

```
## [1] "Lorem ipsum dolor sit amet,\nconsectetur adipiscing elit,\nsed do eiusmod tempor incididunt\nut
```

```
cat(str) # To set the line brakes at the same position as the code
```

```
## Lorem ipsum dolor sit amet,
## consectetur adipiscing elit,
```

```
## sed do eiusmod tempor incididunt
## ut labore et dolore magna aliqua.
```

```
nchar(str) # To get the length of a string
```

```
## [1] 123
```

```
# For finding characters and sequences
str <- "Hello World!"

grepl("H", str)
```

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

```
grepl("Hello", str)
```

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

```
grepl("X", str)
```

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

Escape Symbols:

1. \ Backslash
2. \ New Line
3. Ĉarriage Return
4. T̂ab
5. B̲ackspace

R Operators: Arithmetic Operators:

1. • Addition
2. • Subtraction
3. • Multiplication
4. / Division
5. Ê̂xponent
6. %% Modulus (Remainder from division)
7. %/% Integer Division

Assignment Operators:

```
my_function <- function() {
  print("Hello World!")
}

my_function() # call the function named my_function
```

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

R Data Structures:

```
# Vector of strings
fruits <- c("banana", "apple", "orange")

# Print fruits
fruits
```

R Vectors:

```
## [1] "banana" "apple"  "orange"
```

```
# Vector of numerical values
numbers <- c(1, 2, 3)

# Print numbers
numbers
```

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

```
# Vector with numerical values in a sequence
numbers <- 1:10

numbers
```

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

```
# Length of vectors:
fruits <- c("banana", "apple", "orange")

length(fruits)
```

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

```
# Sort a vector:
fruits <- c("banana", "apple", "orange", "mango", "lemon")
numbers <- c(13, 3, 5, 7, 20, 2)

sort(fruits)  # Sort a string
```

```
## [1] "apple"  "banana" "lemon"  "mango"  "orange"
```

```
sort(numbers) # Sort numbers
```

```
## [1]  2  3  5  7 13 20
```

```
# Accessing vectors:
fruits[1]
```

```
## [1] "banana"
```

```
# Change "banana" to "pear"
fruits[1] <- "pear"

# Print fruits
fruits
```

```
## [1] "pear"   "apple"  "orange" "mango"  "lemon"
```

```
# Repeat vectors:
repeat_each <- rep(c(1,2,3), each = 3)
```

```
repeat_each
```

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

```
numbers <- seq(from = 0, to = 100, by = 20)

numbers
```

```
## [1]   0  20  40  60  80 100
```

```
# List of strings
thislist <- list("apple", "banana", "cherry")

# Print the list
thislist
```

R Lists:

```
## [[1]]
## [1] "apple"
##
## [[2]]
## [1] "banana"
##
## [[3]]
## [1] "cherry"
```

```
# Access the list
thislist[1]
```

```
## [[1]]
## [1] "apple"
```

```
# Change item in list
thislist <- list("apple", "banana", "cherry")
thislist[1] <- "blackcurrant"

# Length of a list
length(thislist)
```

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

```
# Check if item exists
"apple" %in% thislist
```

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

```
# Add item to list
append(thislist, "orange")
```

```
## [[1]]
## [1] "blackcurrant"
##
## [[2]]
## [1] "banana"
##
## [[3]]
```

```
## [1] "cherry"
##
## [[4]]
## [1] "orange"
```

```
# Remove item from list
newlist <- thislist[-1]

# Range of indexes
(thislist)[2:5]
```

```
## [[1]]
## [1] "banana"
##
## [[2]]
## [1] "cherry"
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

```
# Loop through list
for (x in thislist) {
  print(x)
}
```

```
## [1] "blackcurrant"
## [1] "banana"
## [1] "cherry"
```

```
# Join two lists
list1 <- list("a", "b", "c")
list2 <- list(1,2,3)
list3 <- c(list1,list2)

list3
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] "b"
##
## [[3]]
## [1] "c"
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 2
##
## [[6]]
## [1] 3
```

```r
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix
thismatrix
```

R Matrices:

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

```r
# Access matrix items
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

thismatrix[1, 2]
```

```
## [1] "cherry"
```

```r
# Access more than one row
thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fig

thismatrix[c(1,2),]
```

```
##      [,1]     [,2]     [,3]
## [1,] "apple"  "orange" "pear"
## [2,] "banana" "grape"  "melon"
```

```r
# Access more than one column
thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fig

thismatrix[, c(1,2)]
```

```
##      [,1]     [,2]
## [1,] "apple"  "orange"
## [2,] "banana" "grape"
## [3,] "cherry" "pineapple"
```

```r
# Add rows and columns
newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry"))

# Print the new matrix
newmatrix
```

```
##      [,1]     [,2]        [,3]    [,4]
## [1,] "apple"  "orange"    "pear"  "strawberry"
## [2,] "banana" "grape"     "melon" "blueberry"
## [3,] "cherry" "pineapple" "fig"   "raspberry"
```

```r
#Remove the first row and the first column
thismatrix <- thismatrix[-c(1), -c(1)]

thismatrix
```

```
##      [,1]        [,2]
## [1,] "grape"     "melon"
## [2,] "pineapple" "fig"
```

```r
# Check if items exist
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)

"apple" %in% thismatrix
```

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

```r
# Matrix Length
length(thismatrix)
```

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

```r
# Loop trough list
for (rows in 1:nrow(thismatrix)) {
  for (columns in 1:ncol(thismatrix)) {
    print(thismatrix[rows, columns])
  }
}
```

```
## [1] "apple"
## [1] "cherry"
## [1] "banana"
## [1] "orange"
```

```r
# Combine two matrices
# Combine matrices
Matrix1 <- matrix(c("apple", "banana", "cherry", "grape"), nrow = 2, ncol = 2)
Matrix2 <- matrix(c("orange", "mango", "pineapple", "watermelon"), nrow = 2, ncol = 2)

# Adding it as a rows
Matrix_Combined <- rbind(Matrix1, Matrix2)
Matrix_Combined
```

```
##      [,1]     [,2]
## [1,] "apple"  "cherry"
## [2,] "banana" "grape"
## [3,] "orange" "pineapple"
## [4,] "mango"  "watermelon"
```

```r
# Adding it as a columns
Matrix_Combined <- cbind(Matrix1, Matrix2)
Matrix_Combined
```

```
##      [,1]     [,2]     [,3]     [,4]
## [1,] "apple"  "cherry" "orange" "pineapple"
## [2,] "banana" "grape"  "mango"  "watermelon"
```

```r

# Vector of strings
fruits <- c("banana", "apple", "orange")

# Print fruits
fruits
```

R Arrays:

```
## [1] "banana" "apple"  "orange"
```

```r
# Vector of numerical values
numbers <- c(1, 2, 3)

# Print numbers
numbers
```

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

```r
# Vector with numerical values in a sequence
numbers <- 1:10

numbers
```

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

```r
# Length of vectors:
fruits <- c("banana", "apple", "orange")

length(fruits)
```

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

```r
# Sort a vector:
fruits <- c("banana", "apple", "orange", "mango", "lemon")
numbers <- c(13, 3, 5, 7, 20, 2)

sort(fruits)  # Sort a string
```

```
## [1] "apple"  "banana" "lemon"  "mango"  "orange"
```

```r
sort(numbers) # Sort numbers
```

```
## [1]  2  3  5  7 13 20
```

```r
# Accessing vectors:
fruits[1]
```

```
## [1] "banana"
```

```r
# Change "banana" to "pear"
fruits[1] <- "pear"

# Print fruits
fruits
```

```
## [1] "pear"   "apple"  "orange" "mango"  "lemon"
```

```r
# Repeat vectors:
repeat_each <- rep(c(1,2,3), each = 3)

repeat_each
```

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

```r
numbers <- seq(from = 0, to = 100, by = 20)

numbers
```

```
## [1]   0  20  40  60  80 100
```

```r
# List of strings
thislist <- list("apple", "banana", "cherry")

# Print the list
thislist
```

R Lists:

```
## [[1]]
## [1] "apple"
##
## [[2]]
## [1] "banana"
##
## [[3]]
## [1] "cherry"
```

```r
# Access the list
thislist[1]
```

```
## [[1]]
## [1] "apple"
```

```r
# Change item in list
thislist <- list("apple", "banana", "cherry")
thislist[1] <- "blackcurrant"

# Length of a list
length(thislist)
```

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

```r
# Check if item exists
"apple" %in% thislist
```

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

```r
# Add item to list
append(thislist, "orange")
```

```
## [[1]]
## [1] "blackcurrant"
##
## [[2]]
## [1] "banana"
##
## [[3]]
## [1] "cherry"
##
## [[4]]
## [1] "orange"
```

```r
# Remove item from list
newlist <- thislist[-1]

# Range of indexes
(thislist)[2:5]
```

```
## [[1]]
## [1] "banana"
##
## [[2]]
## [1] "cherry"
##
## [[3]]
## NULL
##
## [[4]]
## NULL
```

```
# Loop through list
for (x in thislist) {
  print(x)
}
```

```
## [1] "blackcurrant"
## [1] "banana"
## [1] "cherry"
```

```
# Join two lists
list1 <- list("a", "b", "c")
list2 <- list(1,2,3)
list3 <- c(list1,list2)

list3
```

```
## [[1]]
## [1] "a"
##
## [[2]]
## [1] "b"
##
## [[3]]
## [1] "c"
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 2
##
## [[6]]
## [1] 3
```

```
# An array with one dimension with values ranging from 1 to 24
thisarray <- c(1:24)
thisarray
```

R Matrices:

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

```
# An array with more than one dimension
multiarray <- array(thisarray, dim = c(4, 3, 2))
```

```
multiarray
```

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

```
# Access Array
multiarray[2, 3, 2]
```

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

```
# Check if item exists
2 %in% multiarray
```

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

```
# Amount of rows
dim(multiarray)
```

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

```
# Array Length
length(multiarray)
```

```
## [1] 24
```

```
# Loop trough array
for(x in multiarray){
  print(x)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
## [1] 11
## [1] 12
## [1] 13
## [1] 14
## [1] 15
```

```
## [1] 16
## [1] 17
## [1] 18
## [1] 19
## [1] 20
## [1] 21
## [1] 22
## [1] 23
## [1] 24
```

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
Data_Frame
```

R Data Frames:

```
##   Training Pulse Duration
## 1 Strength   100       60
## 2  Stamina   150       30
## 3    Other   120       45
```

```
# Summarize Data
summary(Data_Frame)
```

```
##    Training             Pulse          Duration
##  Length:3           Min.   :100.0   Min.   :30.0
##  Class :character   1st Qu.:110.0   1st Qu.:37.5
##  Mode  :character   Median :120.0   Median :45.0
##                     Mean   :123.3   Mean   :45.0
##                     3rd Qu.:135.0   3rd Qu.:52.5
##                     Max.   :150.0   Max.   :60.0
```

```
# Access items
Data_Frame[1]
```

```
##   Training
## 1 Strength
## 2  Stamina
## 3    Other
```

```
Data_Frame[["Training"]]
```

```
## [1] "Strength" "Stamina"  "Other"
```

```
Data_Frame$Training
```

```
## [1] "Strength" "Stamina"  "Other"
```

```
# Add a new row
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))
```

```r
# Add a new column
New_col_DF <- cbind(Data_Frame, Steps = c(1000, 6000, 2000))

# Remove the first row and column
Data_Frame_New <- Data_Frame[-c(1), -c(1)]

# Data Frame length
length(Data_Frame)
```

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

```r
# Combining Data Frames
Data_Frame1 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame2 <- data.frame (
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
New_Data_Frame
```

```
##   Training Pulse Duration
## 1 Strength   100       60
## 2  Stamina   150       30
## 3    Other   120       45
## 4  Stamina   140       30
## 5  Stamina   150       30
## 6 Strength   160       20
```

```r

# Create a factor
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))

# Print the factor
music_genre
```

R Factors:

```
## [1] Jazz    Rock    Classic Classic Pop     Jazz    Rock    Jazz
## Levels: Classic Jazz Pop Rock
```

```r
# Factor length
length(music_genre)
```

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

```r
# Access Factors
music_genre[3]
```

```
## [1] Classic
## Levels: Classic Jazz Pop Rock
```

```r
# Chage item value
music_genre[3] <- "Pop"
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.