# Unit -II

# R Programming

# Index

- - Data structures in R
- Data types
- Arrays, Tables
- Matrices: operations
- Lists: operations
- Data frames: creation, factors, reading.

# Data Frames

- Data Frames are data displayed in a format as a table.
- Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical.
- However, each column should have the same type of data.
- Use the data.frame() function to create a data frame:

- **Example**

```
# 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
```

# Summarize the Data

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

Data_Frame

summary(Data_Frame)
```

# Access Items

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

Data_Frame[1]

Data_Frame[["Training"]]

Data_Frame$Training
```

# Add Rows

- Use the rbind() function to add new rows in a Data Frame:

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

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

# Print the new row
New_row_DF
```

# Add Columns

- Use the cbind() function to add new columns in a Data Frame:

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

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

# Print the new column
New_col_DF
```

# Remove Rows and Columns

- Use the c() function to remove rows and columns in a Data Frame:

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

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

# Print the new data frame
Data_Frame_New
```

# Amount of Rows and Columns

- Use the dim() function to find the amount of rows and columns in a Data Frame:

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

dim(Data_Frame)
```

We can also use the ncol() function to find the number of columns and nrow() to find the number of rows:

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

ncol(Data_Frame)
nrow(Data_Frame)
```

# Data Frame Length

- Use the length() function to find the number of columns in a Data Frame (similar to ncol()):

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

length(Data_Frame)
```

# Combining Data Frames

- Use the rbind() function to combine two or more data frames in R vertically:

```
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
```

And use the cbind() function to combine two or more data frames in R horizontally:

```
Data_Frame3 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame4 <- data.frame (
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)
New_Data_Frame1
```

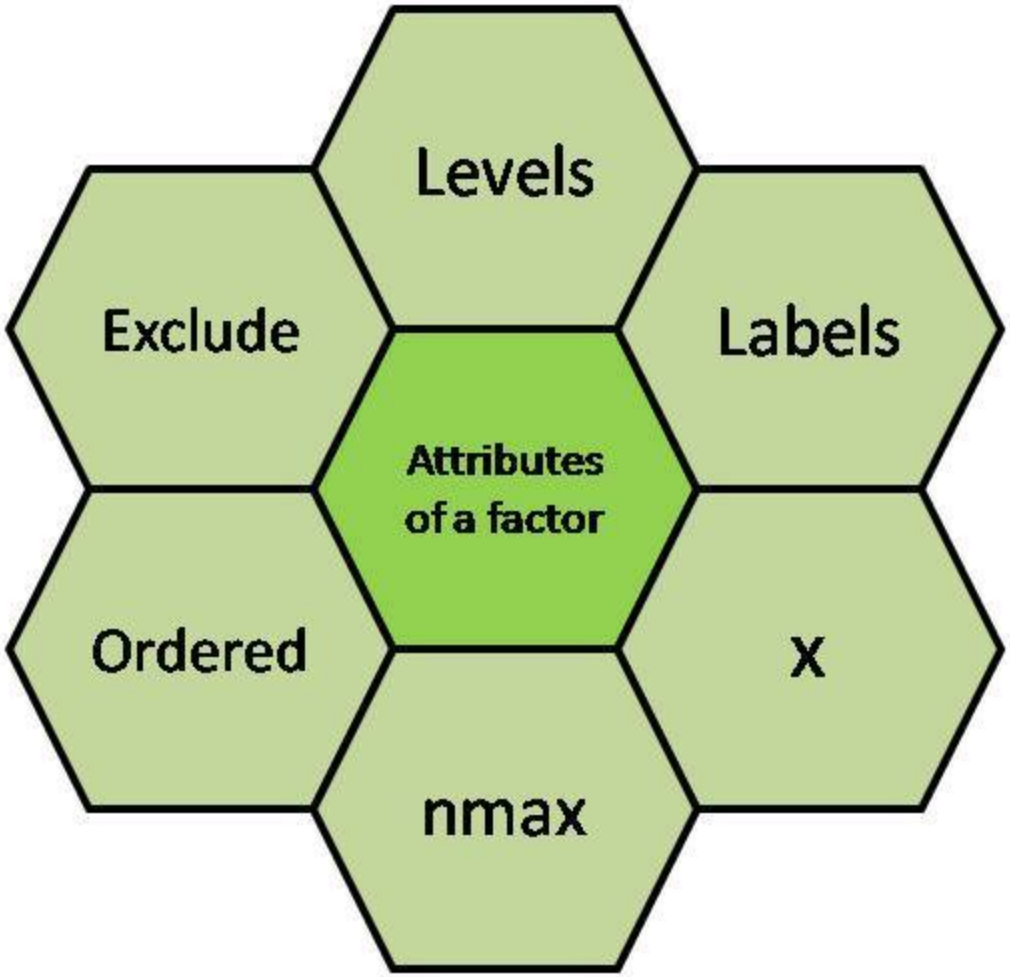
# R Factors

- **Factors** are data structures which are implemented to categorize the data or represent categorical data and store it on multiple levels.

  They can be stored as integers with a corresponding label to every unique integer. Though factors may look similar to character vectors, they are integers and care must be taken while using them as strings.
  The factor accepts only a restricted number of distinct values. For example, a data field such as gender may contain values only from female, male or transgender.
- In the above example, all the possible cases are known beforehand and are predefined. These distinct values are known as levels. After a factor is created it only consists of levels that are by default sorted alphabetically.

# Attributes of a Factor

- **x:** It is the vector which needs to be converted into a factor.
- **Levels:** It is a set of distinct values which are given to the input vector x.
- **Labels:** It is a character vector corresponding to the number of labels.
- **Exclude:** This will mention all the values you want to exclude.
- **Ordered:** This logical attribute decide whether the levels are ordered.
- **nmax:** It will decide the upper limit for the maximum number of levels.


Levels
Exclude
Labels
Attributes
of a factor
Ordered
x
nmax

# Creating a Factor

- The command used to create or modify a factor in R language is – **factor()** with a vector as input.
  The two steps to creating a factor are:
- Creating a vector
- Converting the vector created into a factor using function factor()

# Factors

- Factors are used to categorize data. Examples of factors are:

1. Demography: Male/Female
2. Music: Rock, Pop, Classic, Jazz
3. Training: Strength, Stamina

To create a factor, use the factor() function and add a vector as argument:

- **Example**

```
music_genre <- factor(c("Jazz", "Rock", "Classic",
   "Classic", "Pop", "Jazz", "Rock", "Jazz"))


music_genre
```

```
# Creating a vector
x<-c("female", "male", "male", "female")
print(x)

# Converting the vector x into a factor named
   gender
gender<-factor(x)
print(gender)
```

# Checking for a Factor

- Function**is.factor()** is used to check whether the variable is a factor and returns “TRUE” if it is a factor.

```
gender <- factor(c("female", "male", "male",
  "female"));
print(is.factor(gender))
```

- Function **class()** is also used to check whether the variable is a factor and if true returns “factor”.

```
gender <- factor(c("female", "male", "male",
   "female"));
class(gender)
```

# Accessing elements of a Factor

- Like we acess elements of a vector, same way we acess the elements of a factor. If gender is a factor then gender[i] would mean acessing i th element in the factor.

```
gender <- factor(c("female", "male", "male",
  "female"));
gender[3]
```

- More than one element can be accessed at a time.

```
gender <- factor(c("female", "male", "male",
  "female"));
gender[c(2, 4)]
```

# Modification of a Factor

- After a factor is formed, its components can be modified but the new values which needs to be assigned must be in the predefined level.

```
gender <- factor(c("female", "male", "male",
   "female"  ));
gender[2]<-"female"
gender
```

- For selecting all the elements of the factor gender except ith element, gender[-i] should be used.
  So if you want to modify a factor and add value out of predefines levels, then first modify levels.

```
gender <- factor(c("female", "male", "male", "female"
  ));

# add new level
levels(gender) <- c(levels(gender), "other")
gender[3] <- "other"
gender
```

# Factors in Data Frame

- On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

```
# Create the vectors for data frame. height <-
  c(132,151,162,139,166,147,122) weight <-
  c(48,49,66,53,67,52,40) gender <-
  c("male","male","female","female","male","fe
  male","male") # Create the data frame.
  input_data <-
  data.frame(height,weight,gender)
  print(input_data) # Test if the gender column
  is a factor. print(is.factor(input_data$gender))
  # Print the gender column so see the levels.
  print(input_data$gender)
```

# Arrays

- Arrays are essential data storage structures defined by a fixed number of dimensions. Arrays are used for the allocation of space at contiguous memory locations. Uni-dimensional arrays are called vectors with the length being their only dimension. Two-dimensional arrays are called matrices, consisting of fixed numbers of rows and columns. Arrays consist of all elements of the same data type. Vectors are supplied as input to the function and then create an array based on the number of dimensions.

# Creating an Array

- An array in R can be created with the use of **array()** function. List of elements is passed to the array() functions along with the dimensions as required.
- **Syntax:**
- array(data, dim = (nrow, ncol, nmat), dimnames=names)

# Uni-Dimensional Array

- A vector is a uni-dimensional array, which is specified by a single dimension, length. A Vector can be created using '**c()**' function. A list of values is passed to the c() function to create a vector.

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
print (vec1)

# cat is used to concatenate
# strings and print it.
cat ("Length of vector : ", length(vec1))
```

# Multi-Dimensional Array

- A two-dimensional matrix is an array specified by a fixed number of rows and columns, each containing the same data type. A matrix is created by using **array()** function to which the values and the dimensions are passed.

```
# arranges data from 2 to 13
# in two matrices of dimensions 2x3
arr = array(2:13, dim = c(2, 3, 2))
print(arr)
```

- Vectors of different lengths can also be fed as input into the **array()** function. However, the total number of elements in all the vectors combined should be equal to the number of elements in the matrices. The elements are arranged in the order in which they are specified in the function.

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)

# elements are combined into a single vector,
# vec1 elements followed by vec2 elements.
arr = array(c(vec1, vec2), dim = c(2, 3, 2))
print (arr)
```

# Accessing arrays

- The arrays can be accessed by using indices for different dimensions separated by commas. Different components can be specified by any combination of elements' names or positions.

```
vec <- c(1:10)

# accessing entire vector
cat ("Vector is : ", vec)

# accessing elements
cat ("Third element of vector is : ", vec[3])
```

# Accessing entire matrices

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")
arr = array(c(vec1, vec2), dim = c(2, 3, 2),
            dimnames = list(row_names,
                            col_names, mat_names))

# accessing matrix 1 by index value
print ("Matrix 1")
print (arr[,,1])

# accessing matrix 2 by its name
print ("Matrix 2")
print(arr[,,"Mat2"])
```

# Accessing specific rows and columns of matrices

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")
arr = array(c(vec1, vec2), dim = c(2, 3, 2),
        dimnames = list(row_names,
                        col_names, mat_names))

# accessing matrix 1 by index value
print ("1st column of matrix 1")
print (arr[, 1, 1])

# accessing matrix 2 by its name
print ("2nd row of matrix 2")
print(arr["row2",,"Mat2"])
```

# Accessing elements individually

```
vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)
vec2 <- c(10, 11, 12)
row_names <- c("row1", "row2")
col_names <- c("col1", "col2", "col3")
mat_names <- c("Mat1", "Mat2")
arr = array(c(vec1, vec2), dim = c(2, 3, 2),
    dimnames = list(row_names, col_names, mat_names))

# accessing matrix 1 by index value
print ("2nd row 3rd column matrix 1 element")
print (arr[2, "col3", 1])

# accessing matrix 2 by its name
print ("2nd row 1st column element of matrix 2")
print(arr["row2", "col1", "Mat2"])
```