Lab
Practical – 07

Title: - Introduction to R Graphics and Data Preprocessing

Aim: - To perform data preprocessing using R programming.

Lab Objectives: -

Students will understand following R programming concepts:

I. Importing dataset
II. Handling the Missing Data
III. Encoding Categorical Data
IV. Splitting the Dataset into the Training and Test sets
V. Feature Scaling **Description: -**

Data Preprocessing in R

Data preprocessing is the initial phase of Machine Learning where data is prepared for machine learning models.
This part is crucial and needs to be performed properly and systematically.
If not, we will end up building models that are not accurate for their purpose.
Prerequisites
- To perform data preprocessing, you should have the following:
  - RStudio installed on your computer.
  - Packages 'caTools', 'tidyverse', readr, dplyr, ggplot2 installed.

▷ **Steps in data preprocessing**
- Step 1: Importing the Dataset ○
Step 2: Handling the Missing Data ○
Step 3: Encoding Categorical Data.
- Step 4: Splitting the Dataset into the Training and Test sets
- Step 5: Feature Scaling **I.**

Importing the dataset

▷ **Loading data from csv file**

```
dfdata<-read.csv("data.csv")
dfdata
```

You can get a count of the number of records with the nrow() function

```
nrow(dfdata)
```

You can get other information about dataset using following functions

```
dim(dfdata)
names(dfdata)
rownames(dfdata)
```

▹ **Selecting specific columns**

- It will often be the case that you don't need all the columns in the data that you import.
- The dplyr package includes a select() function that can be used to limit the fields in the data frame.

```
dfdata = select(dfdata,'Country','Age','Purchased')
View(dfdata)
```

▹ **Renaming Columns**

- You may also want to rename columns to make them more reader friendly or perhaps simplify the names.
- The select() function can be used to do this as well.
- You simply pass in the new name of the column followed by an equal sign and then the old column name.

```
dfdata = select(dfdata,'country'='Country','age'='Age','Purchased')
View(dfdata)
```

▹ **Filtering a dataset**

- In addition to limiting the columns that are part of a data frame, it's also common to subset or filter the rows using a where clause.
- Filtering the dataset enables you to focus on a subset of the rows instead of the entire dataset.
- The dplyr package includes a filter() function that supports this capability.

```
dfdata1=filter(dfdata,Country=='France')
View(dfdata1)
```

- You can also include multiple expressions in a filter() function.

```
dfdata2=filter(dfdata,country=='France',age<=40)
View(dfdata2)
```

II. Step 2: Handling the missing data

From the dataset, the Age and Salary column report missing data.

Before implementing our machine learning models, this problem needs to be solved, otherwise it will cause a serious problem to our machine learning models.

D Therefore, it's our responsibility to ensure this missing data is eliminated from our dataset using the most appropriate technique.

Here are two techniques we can use to handle missing data:

- ○ Delete the observation reporting the missing data:
  - ■ This technique is suitable when dealing with big datasets and with very few missing values i.e. deleting one row from a dataset with thousands of observations can not affect the quality of the data.
  - ■ When the dataset reports many missing values, it can be very dangerous to use this technique.
  - ■ Deleting many rows from a dataset can lead to the loss of crucial information contained in the data.
  - ■ To ensure this does not happen, we make use of an appropriate technique that has no harm to the quality of the data.
- ○ Replace the missing data with the average of the feature in which the data is missing:
  - ■ This technique is the best way so far to deal with the missing values.
  - ■ Many statisticians make use of this technique over that of the first one.

If you want to check if a value is missing, you must use the function is.na:is.na(NA)

To get the total number of NAs present in the dataset

```
sum(is.na(dfdata))
```

The following function can be used to replace the NA with the column mean for all the numeric columns. The numeric columns are identified by the sapply(data, is.numeric) function sapply(dfdata, is.numeric)

If you want just to ignore the NA values, there is often a parameter for specifying this

```
sum(dfdata$age,na.rm = TRUE)
```

D Let's start by replacing the missing data in the Age column with the mean of that column.

```
dfdata$age <- ifelse(is.na(dfdata$age),ave(dfdata$age, FUN = function(x) mean(x, na.rm =
TRUE)), dfdata$age)
```

What does the code above really do?

The above code blocks check for missing values in the age and salary columns and update the missing cells with the column-wise average.

- dfdata$age : Selects the column in the dataset specified after $
- is.na(dfdata$age): This method returns true for all the cells in the specified column with no values.
- ave(dfdata$age, FUN = function(x) mean(x, na.rm = 'TRUE')): This method calculates the average of the column passed as argument.

III. Step 3: Encoding categorical data

Encoding refers to transforming text data into numeric data.

Encoding Categorical data simply means we are transforming data that fall into categories into numeric data.

In our dataset, the Country column is Categorical data with 3 levels i.e. France, Spain, and Germany.

The purchased column is Categorical data as well with 2 categories, i.e. YES and NO.

The machine models we built on our dataset are based on mathematical equations and it only take numbers in those equations.

Keeping texts of a categorical variable in the equation can cause some troubles to the machine learning models and this why we encode those variables.

To transform a categorical variable into numeric, we use the factor() function.

```
dfdata$country = factor(dfdata$country,

               levels = c('France','Spain','Germany'),

               labels = c(1.0, 2.0 , 3.0 ))
```

Our country names were successfully replaced with numbers.
We do the same for the purchased column.

```
dfdata$Purchased = factor(dfdata$Purchased,levels = c('No', 'Yes'),labels = c(0, 1))
```

IV. Step 4: Splitting the dataset into the training and test set

In machine learning, we split data into two parts:

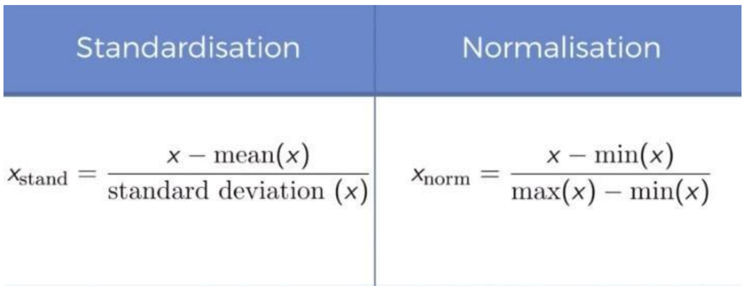
- Training set: The part of the data that we implement our machine learning model on.

○ Test set: The part of the data that we evaluate the performance of our machine learning model on.

D Using our dataset, let's split it into the training and test sets.
To begin with, we first load the required library.
D Install package caTools install.packages("caTools")
library(caTools)# required library for data splition
set.seed(123)
split = sample.split(dfdata$Purchased, SplitRatio = 0.8)# returns true if observation goes to the Training set and false if observation goes to the test set. #Creating the training set and test set separately training_set = subset(dfdata, split == TRUE) test_set = subset(dfdata, split == FALSE) training_set test_set

V. Step 5: Feature scaling

D It's a common case that in most datasets, features also known as inputs, are not on the same scale.
Many machine learning models are Euclidian distant-based.
It happens that, the features with the large units dominate those with small units when it comes to calculation of the Euclidian distance and it will be as if those features with small units do not exist.
To ensure this does not occur, we need to encode our features so that they all fall in the range between -3 and 3.
There are several ways we can use to scale our features.
The most used one is the standardization and normalization technique.
The normalization technique is used when the data is normally distributed while standardization works with both normally distributed and the data that is not normally distributed.
The formula for these two techniques is shown below.



Standardisation	Normalisation
$x_{\text{stand}} = \dfrac{x - \text{mean}(x)}{\text{standard deviation }(x)}$	$x_{\text{norm}} = \dfrac{x - \min(x)}{\max(x) - \min(x)}$

D Now, let's scale both the training set and test set of our dataset separately.
Here is how we achieve this:

training_set[, 2] = scale(training_set[, 2])

test_set[, 2] = scale(test_set[, 2])

training_set test_set

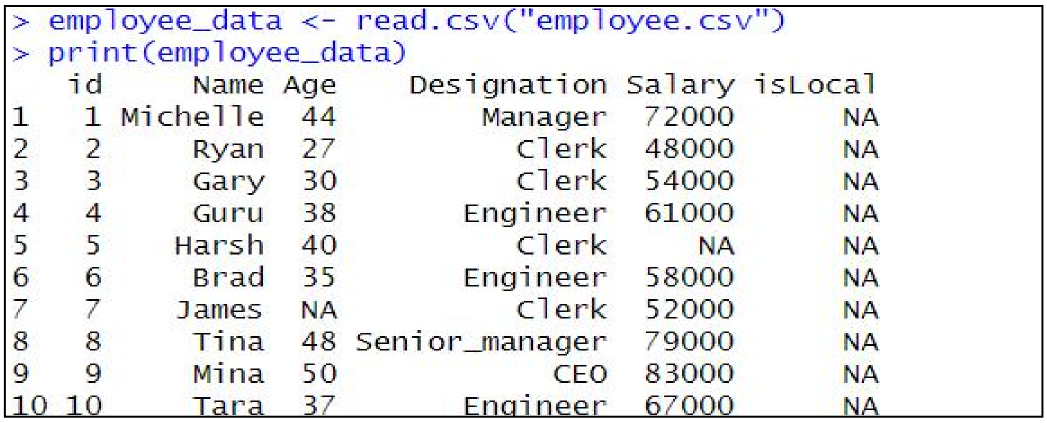
Our training and test set were successfully scaled.

Note that in our code we specified the columns to be scale.

Exercises

1. Import employee.csv file and perform following –

employee_data <- read.csv("employee.csv")
print(employee_data)

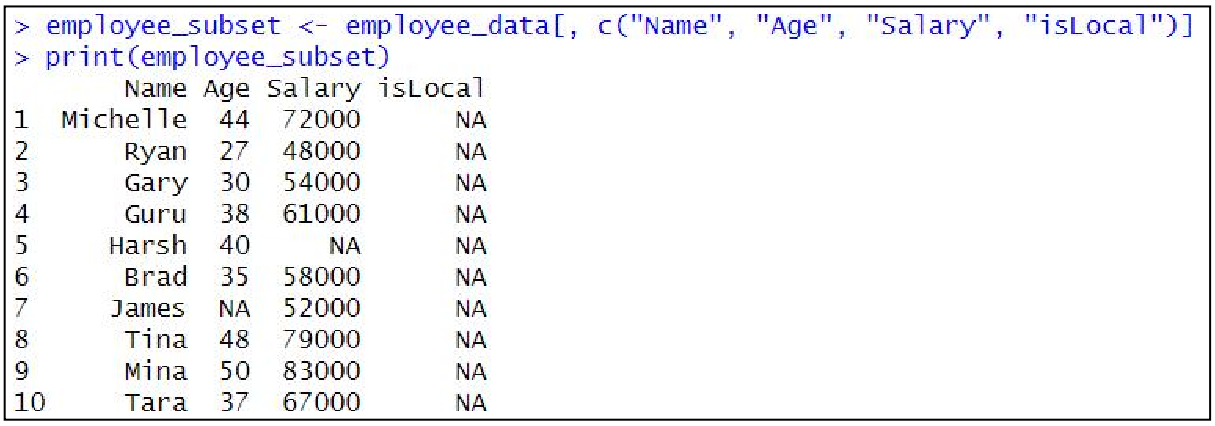


```
> employee_data <- read.csv("employee.csv")
> print(employee_data)
   id     Name Age     Designation Salary isLocal
1   1 Michelle  44         Manager  72000      NA
2   2     Ryan  27           Clerk  48000      NA
3   3     Gary  30           Clerk  54000      NA
4   4     Guru  38        Engineer  61000      NA
5   5    Harsh  40           Clerk     NA      NA
6   6     Brad  35        Engineer  58000      NA
7   7    James  NA           Clerk  52000      NA
8   8     Tina  48 Senior_manager  79000      NA
9   9     Mina  50             CEO  83000      NA
10 10     Tara  37        Engineer  67000      NA
```

1. Extract only following columns "Name", "Age", "Salary", "isLocal" into dataframe "employee_subset"

employee_subset <- employee_data[, c("Name", "Age", "Salary", "isLocal")]print(employee_subset)

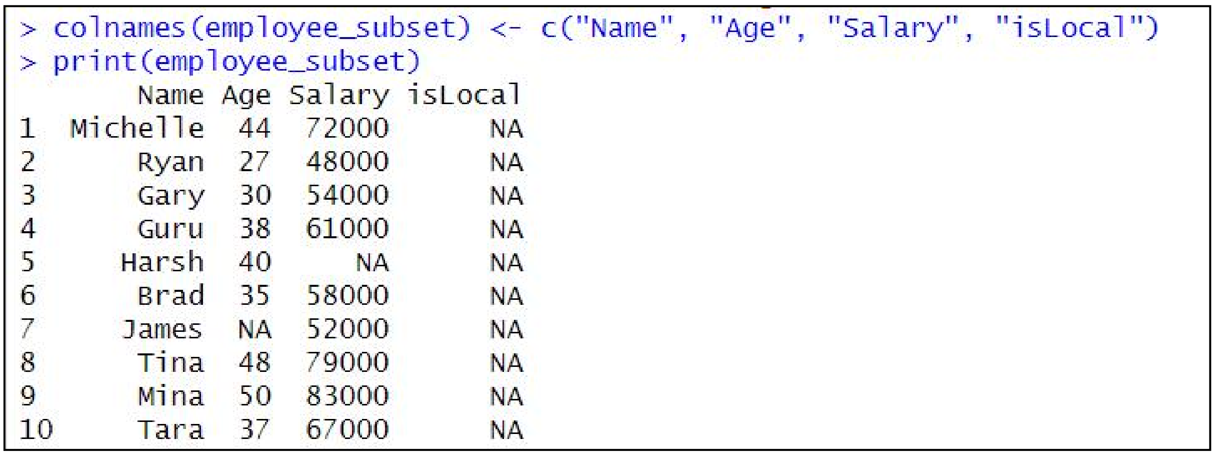
Lab



```
> employee_subset <- employee_data[, c("Name", "Age", "Salary", "isLocal")]
> print(employee_subset)
      Name Age Salary isLocal
1  Michelle  44  72000      NA
2      Ryan  27  48000      NA
3      Gary  30  54000      NA
4      Guru  38  61000      NA
5     Harsh  40     NA      NA
6      Brad  35  58000      NA
7     James  NA  52000      NA
8      Tina  48  79000      NA
9      Mina  50  83000      NA
10     Tara  37  67000      NA
```

2. Rename the following columns ""Name", "Age", "Designation", "Salary", "isLocal" from employee_subset dataframe

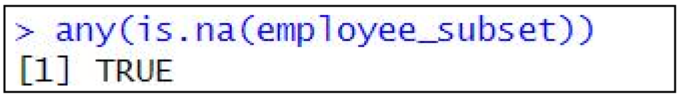
colnames(employee_subset) <- c("Name", "Age", "Salary", "isLocal")
print(employee_subset)



```
> colnames(employee_subset) <- c("Name", "Age", "Salary", "isLocal")
> print(employee_subset)
      Name Age Salary isLocal
1  Michelle  44  72000      NA
2      Ryan  27  48000      NA
3      Gary  30  54000      NA
4      Guru  38  61000      NA
5     Harsh  40     NA      NA
6      Brad  35  58000      NA
7     James  NA  52000      NA
8      Tina  48  79000      NA
9      Mina  50  83000      NA
10     Tara  37  67000      NA
```

2. Check if a value is missing in employee_subset
Code:

any(is.na(employee_subset))

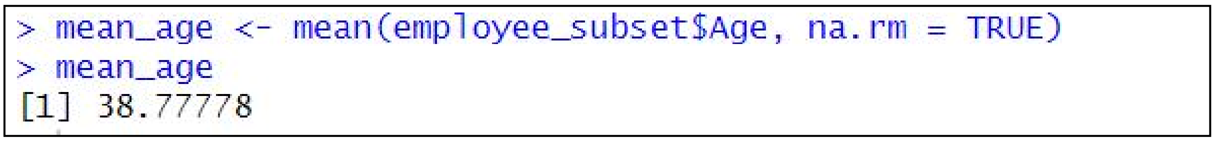


```
> any(is.na(employee_subset))
[1] TRUE
```

Lab

4. Calculate the mean of Age and Salary column in employee_subset
Mean Age:

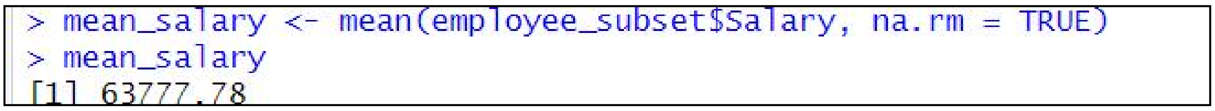
```
mean_age <- mean(employee_subset$Age, na.rm = TRUE)
mean_age
```



```
> mean_age <- mean(employee_subset$Age, na.rm = TRUE)
> mean_age
[1] 38.77778
```

Mean Salary:

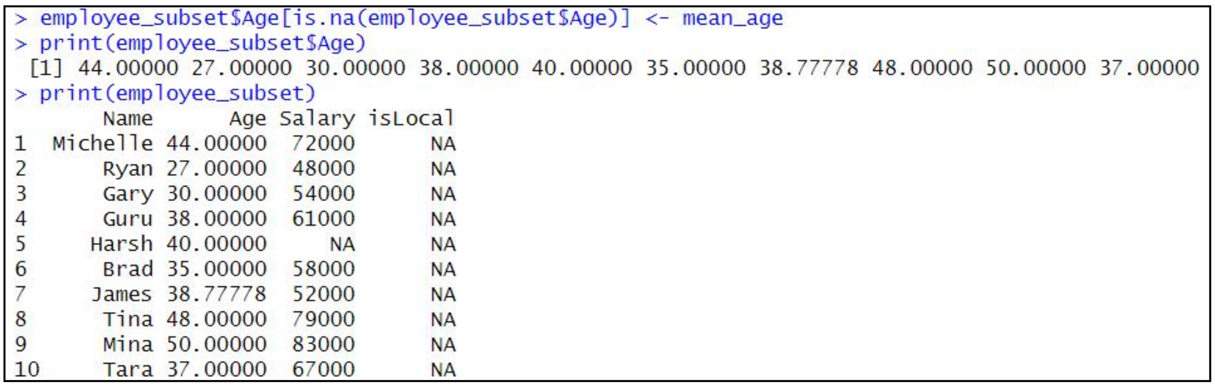
```
mean_salary <- mean(employee_subset$Salary, na.rm = TRUE)
mean_salary
```



```
> mean_salary <- mean(employee_subset$Salary, na.rm = TRUE)
> mean_salary
[1] 63777.78
```

5. Replace missing values by mean of that variable/column
Replace missing column Age:

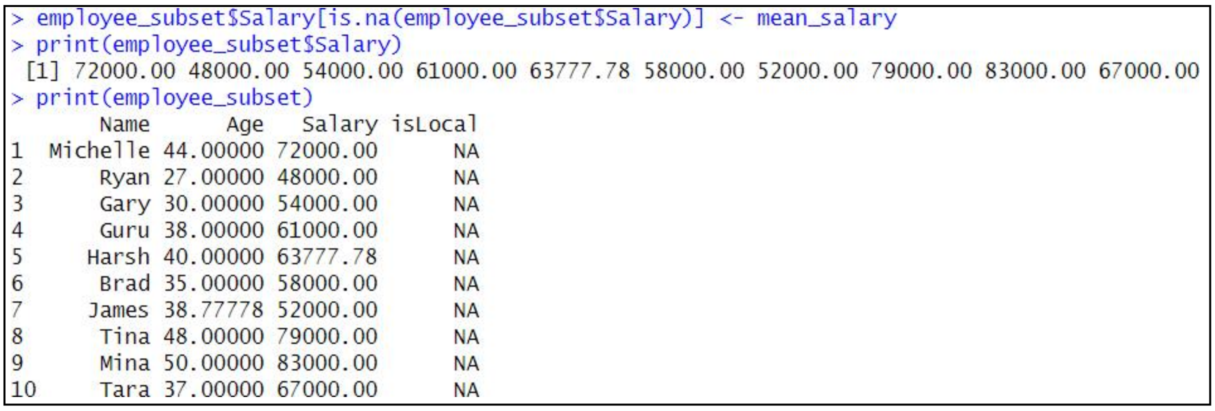
```
employee_subset$Age[is.na(employee_subset$Age)] <- mean_age
print(employee_subset$Age)
print(employee_subset)
```



```
> employee_subset$Age[is.na(employee_subset$Age)] <- mean_age
> print(employee_subset$Age)
 [1] 44.00000 27.00000 30.00000 38.00000 40.00000 35.00000 38.77778 48.00000 50.00000 37.00000
> print(employee_subset)
       Name      Age Salary isLocal
1  Michelle 44.00000  72000      NA
2      Ryan 27.00000  48000      NA
3      Gary 30.00000  54000      NA
4      Guru 38.00000  61000      NA
5     Harsh 40.00000     NA      NA
6      Brad 35.00000  58000      NA
7     James 38.77778  52000      NA
8      Tina 48.00000  79000      NA
9      Mina 50.00000  83000      NA
10     Tara 37.00000  67000      NA
```

Replace missing column Salary:

```
employee_subset$Salary[is.na(employee_subset$Salary)] <- mean_salary
print(employee_subset$Salary)
 print(employee_subset)
```



```
> employee_subset$Salary[is.na(employee_subset$Salary)] <- mean_salary
> print(employee_subset$Salary)
 [1] 72000.00 48000.00 54000.00 61000.00 63777.78 58000.00 52000.00 79000.00 83000.00 67000.00
> print(employee_subset)
       Name      Age   Salary isLocal
1  Michelle 44.00000 72000.00      NA
2      Ryan 27.00000 48000.00      NA
3      Gary 30.00000 54000.00      NA
4      Guru 38.00000 61000.00      NA
5     Harsh 40.00000 63777.78      NA
6      Brad 35.00000 58000.00      NA
7     James 38.77778 52000.00      NA
8      Tina 48.00000 79000.00      NA
9      Mina 50.00000 83000.00      NA
10     Tara 37.00000 67000.00      NA
```