Course Code: CSE3190

Course Title: Fundamentals of Data Analysis

Lab sheet- 4

Experiment No 4: Demonstrate Data Cleaning and Preprocessing using R Programming

Level1: **Illustrate** how to Handle Missing Data in R by Identifying missing values and imputing missing values using mean, median, or other methods. Level2: **Demonstrate** Data Transformation using R Programming and perform Standardizing and normalizing data and Log-transformations and scaling.

Data cleaning is the process of identifying and fixing errors or inconsistencies in a dataset to make it accurate and reliable for analysis. It includes handling missing data, removing duplicates, correcting mistakes, resolving inconsistencies, and standardizing formats, ultimately transforming raw data into a refined, ready-to-use form.

- **Basic Steps in Data Cleaning Using R**

***Loading the Data Set**

```
data <- read.csv("C:/Users/User/Downloads/dataframe.csv",header = TRUE, sep = ",")
```

#2.Inspecting the dataset

 # Information about the dataframe

```
str(data)
```

#names of the columns present in dataset

```
names(data)
```

#To check the first few rows of dataset

head(data)

#To check the summary of the dataset

summary(data)

 # Number of rows and columns

dim(data)

#to know the number of rows in a dataset

nrow (data)

#to know the number of cols in a dataset

ncol(data)

 # Check the data type of each column

sapply(data, class)

 # Check the class (data frame, matrix, etc.) of the dataset

class(data)

 # Open a spreadsheet-like view of the dataset in RStudio

View(data)

#3. Checking for Missing values

#to check for missing values in the entire dataset

sum(is.na(data))

#to check for number of missing values per column

colSums(is.na(data))

#to check for number of missing values per row

rowSums(is.na(data))

 # Check if there are any missing values in the dataset

anyNA(data)

#4. Handling Missing data

#removes missing values in the dataset

na.omit(data)

#remove the data in sepcific value using the command

data[!is.na(data$column_name),]

#cheching if the all the rows or columns in a datframe have no missing value

complete.cases(data)

#Illustrate how to Handle Missing Data in R by Identifying missing values and imputing missing values using mean, median, or other methods.

#Imputing Missing Values with the Mean

#For numeric columns, a common method is to

Calculate Mean for all numerical attributes (excluding model name and vs)

mean_values <- sapply(data[, 2:11], mean) # 2:11 selects columns 2 to 11 (numerical attributes)

 # Print the results

print("Mean Values:")

print(mean_values)

 # Replace missing values in a numeric column (e.g., column_name) with the mean of the column

data$column_name[is.na(data$ column_name)] <- mean(data$ column_name, na.rm = TRUE)

 # Calculate Median for all numerical attributes (excluding model name and vs)

median_values <- sapply(data[, 2:11], median) print("\nMedian Values:")

print(median_values)

 # Replace missing values in a numeric column (e.g., ' column_name **') with the median of the column**

data$ column_name [is.na(data$ column_name)] <- median(data$ column_name, na.rm = TRUE)

 # Calculate Mode for all numerical attributes (excluding model name and vs)

library(modeest)

mode_values <- sapply(data[, 2:11], mfv)

print(mode_values)

 # Calculate the mode of a categorical variable

d1 <- c("A", "B", "A", "C", "A")

mode_value <- mfv(d1)#most frequent function(mfv)

print(mode_value)

 # Replace missing values in a categorical column (e.g., 'column_name') with the mode

data$ **column_name** [is.na(data$ **column_name**)] <- get_mode(data$ **column_name**)

#Impute Missing Values with a Constant Value

#You may want to replace missing values with a constant value, such as 0 or a placeholder string.

 # Replace missing values in the dataset with a specific value (e.g., 0 for numeric, 'Unknown' for categorical)

data$ **column_name** [is.na(data$ **column_name**)] <- 0

data$ **column_name** [is.na(data$ **column_name**)] <- 'NAN'

 # Replace missing values based on a custom condition

data$ **column_name** [is.na(data$ **column_name**)] <- ifelse(data$ **column_name** > 50000, 40, 30)

#Recoding in R

 # Sample dataset

```
sample1 <- data.frame(
  Age = c(25, 40, 17, 55, 70),
  Income = c(35000, 50000, 20000, 75000, 100000),
  Gender = c("M", "F", "M", "F", "M")
)
```

 # View the dataset

```
print(sample1)
```

#Recoding Age into Age Groups: We'll recode the Age variable into three categories: "Minor," "Adult," and "Senior,"

#using the ifelse() function.

 # Recode 'Age' into age groups

```
sample1$AgeGroup <- ifelse(sample1$Age < 18, "Minor",
                    ifelse(sample1$Age <= 60, "Adult", "Senior"))
```

 # Recode 'Income' into income categories

```
sample1$IncomeCategory <- ifelse(sample1$Income < 30000, "Low",
                          ifelse(sample1$Income <= 60000, "Medium", "High"))
```

 # Recode 'Gender' using factor

```
sample1$Gender <- factor(sample1$Gender, levels = c("M", "F"), labels = c("Male", "Female"))
```

 # View the updated dataset

 # Use 'within()' to modify the data frame and 'cut()' to categorize 'Age'

```
sample1 <- within(sample1, {
  AgeCategory <- cut(Age, breaks = c(0, 18, 60, 100), labels = c("Minor", "Adult", "Senior"))
})
print(sample1)
```

* **Data Transformation in R**: Standardization, Normalization, Log-Transformations, and Scaling

Data transformation is a crucial step in data preprocessing, especially when dealing with algorithms that are sensitive to the scale of input data. Below is an explanation and demonstration of how to perform standardization, normalization, log-transformation, and scaling in R.

1. Standardization (Z-score Normalization)

Standardization transforms the data so that it has a mean of 0 and a standard deviation of 1. This ensures that the features have the same scale, making the data suitable for algorithms that rely on the magnitude of the data, such as linear regression or SVM.

```
# data
data <- data.frame(
  Age = c(25, 40, 17, 55, 70),
  Income = c(35000, 50000, 20000, 75000, 100000)
)

# Standardization (Z-score normalization)
data_standardized <- as.data.frame(scale(data))

# View standardized data
print(data_standardized)
```

2. Normalization (Min-Max Scaling)

Normalization scales data between a fixed range, commonly [0,1]. This transformation is particularly useful when the range of values is uneven across features, as it brings all features to the same scale without distorting their relationships.

```
# Normalization scales data to [0,1] range
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}
```

```
# Apply normalization to all columns
data_normalized <- as.data.frame(lapply(data, normalize))


# View normalized data
print(data_normalized)
```

3. Log-Transformation

Log-transformation compresses the range of the data, particularly useful when there are large outliers or a skewed distribution. It reduces the impact of extreme values by transforming them logarithmically.

```
# Log-transform the 'Income' column
data_log_transformed <- data
data_log_transformed$Income <- log(data$Income)


# View log-transformed data
print(data_log_transformed)
```

4. Scaling (Mean Centering)

Scaling is the process of shifting data such that the mean of the transformed data becomes zero. This is particularly useful for techniques like **Principal Component Analysis (PCA)**, which rely on mean-centered data to compute principal components.

```
# Scaling (mean-centering) without altering standard deviation
data_scaled <- as.data.frame(scale(data, center = TRUE, scale = FALSE))


# View scaled data
print(data_scaled)
```