Experiment - 01

Aim: To explore the descriptive statistics on the given dataset

Theory:

1. Introduction to Descriptive Statistics

Descriptive statistics summarize and describe the key features of a dataset, providing an overview of its structure and distribution. They include measures of central tendency (mean, median, and mode) and measures of variability (variance, standard deviation, range, IQR).

- **Measures of Central Tendency**:
  - **Mean**: The average of all data points. It is sensitive to outliers.
  - **Median**: The middle value when data is ordered. It is robust to outliers.
  - **Mode**: The most frequently occurring value.
- **Measures of Variability**:
  - **Variance**: Measures the spread of data around the mean in squared units.
  - **Standard Deviation**: The square root of variance; indicates how much data deviates from the mean.
  - **IQR (Interquartile Range)**: Difference between the 3rd and 1st quartiles (Q3 - Q1), robust against outliers.
  - **Coefficient of Variation (CV)**: Relative measure of variability calculated as standard deviation divided by the mean.

2. Measures of Shape

These include skewness and kurtosis, which describe the distribution's symmetry and peakedness.

- **Skewness**: Indicates the symmetry of data:
  - **Negative Skew**: The distribution has a long tail on the left side. The data values are concentrated on the right, and extremely small values pull the mean downward.
    Relationship: Mean < Median < Mode.
  - **Zero Skew**: The distribution is symmetrical, appearing balanced on both sides of the central value. It often resembles a bell shape.
    Relationship: Mean = Median = Mode.
  - **Positive Skew**: The distribution has a long tail on the right side. The data values are concentrated on the left, and extremely large values pull the mean upward.
    Relationship: Mean > Median > Mode.
- **Kurtosis**: Reflects the sharpness of the peak in data:
  - High kurtosis: Distinct peak and heavy tails.
  - Low kurtosis: Flat peak and lighter tails.

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load the Iris dataset
data = load_iris()
iris_df = pd.DataFrame(data.data, columns=data.feature_names)
iris_df['species'] = data.target

# Print basic information
print(iris_df.shape)
print(iris_df.head())
print(iris_df.info())
print(iris_df.isnull().sum())
print(iris_df.describe())

# Calculate mean, median, and mode
mean = iris_df['sepal length (cm)'].mean()
print("\nMean:", mean)

median = iris_df['sepal length (cm)'].median()
print("Median:", median)

mode = iris_df['sepal length (cm)'].mode()
print("Mode:", mode)

# Distribution plot
sns.histplot(iris_df['sepal length (cm)'], bins=10, kde=True,
color='blue')
plt.title("Distribution Plot of Sepal Length (cm)")
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Frequency")
plt.legend(labels=['sepal length (cm)'])
plt.show()

print(" ")
# Boxplot
sns.boxplot(x=iris_df['sepal length (cm)'], color='green')
plt.title("Boxplot of Sepal Length (cm)")
plt.xlabel("Sepal Length (cm)")
plt.show()
```

```
# Calculate other statistics
print("Min:", iris_df['sepal length (cm)'].min())
print("Max:", iris_df['sepal length (cm)'].max())
print("Range:", iris_df['sepal length (cm)'].max() - iris_df['sepal
length (cm)'].min())
print("Variance:", iris_df['sepal length (cm)'].var())
print("Standard Deviation:", iris_df['sepal length (cm)'].std())

# Interquartile Range (IQR)
Q1 = iris_df['sepal length (cm)'].quantile(0.25)
Q2 = iris_df['sepal length (cm)'].quantile(0.5)
Q3 = iris_df['sepal length (cm)'].quantile(0.75)
IQR = Q3 - Q1

print("Q1:", Q1)
print("Q2 (Median):", Q2)
print("Q3:", Q3)
print("IQR:", IQR)

# Skewness and Kurtosis
print("Skewness:", iris_df['sepal length (cm)'].skew())
print("Kurtosis:", iris_df['sepal length (cm)'].kurt())
```

Output:

Basic Information: shape, head, info, isnull, describe

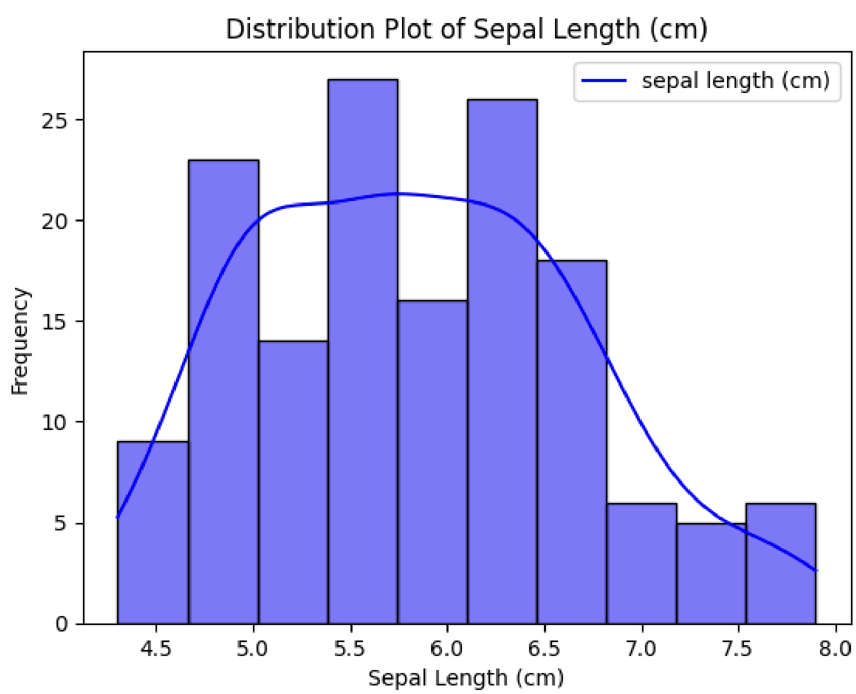
```
(150, 5)
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  species
0                5.1               3.5                1.4               0.2        0
1                4.9               3.0                1.4               0.2        0
2                4.7               3.2                1.3               0.2        0
3                4.6               3.1                1.5               0.2        0
4                5.0               3.6                1.4               0.2        0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   species            150 non-null    int64
dtypes: float64(4), int64(1)
```

```
None
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
dtype: int64
       sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)     species
count         150.000000        150.000000         150.000000        150.000000  150.000000
mean            5.843333          3.057333           3.758000          1.199333    1.000000
std             0.828066          0.435866           1.765298          0.762238    0.819232
min             4.300000          2.000000           1.000000          0.100000    0.000000
25%             5.100000          2.800000           1.600000          0.300000    0.000000
50%             5.800000          3.000000           4.350000          1.300000    1.000000
75%             6.400000          3.300000           5.100000          1.800000    2.000000
max             7.900000          4.400000           6.900000          2.500000    2.000000
```

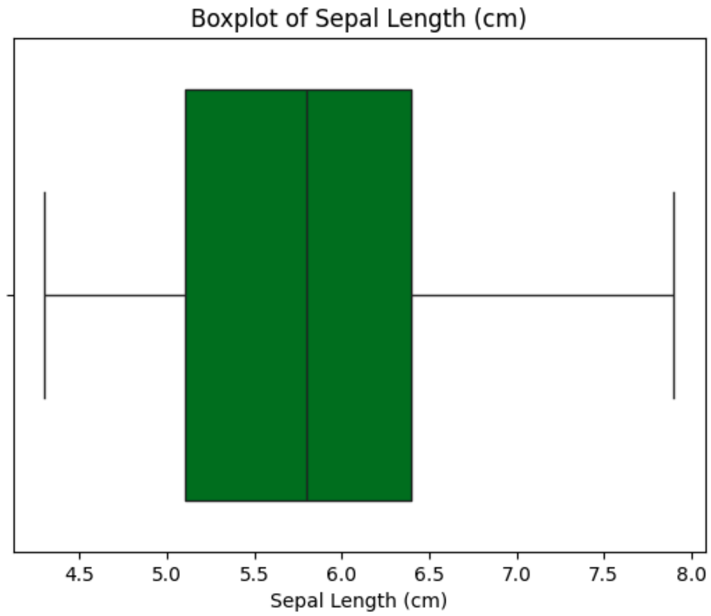
Mean, Median and Mode for the column "Sepal Length"

```
Mean: 5.843333333333334
Median: 5.8
Mode: 0    5.0
Name: sepal length (cm), dtype: float64
```

Distribution plot for the column "Sepal Length"



Boxplot for the column “Sepal Length”



Measures of dispersion or variability

```
Min: 4.3
Max: 7.9
Range: 3.6000000000000005
Variance: 0.6856935123042505
Standard Deviation: 0.8280661279778629
Q1: 5.1
Q2 (Median): 5.8
Q3: 6.4
IQR: 1.3000000000000007
Skewness: 0.3149109566369728
Kurtosis: -0.5520640413156395
```

Conclusion: Hence, we performed descriptive analysis on the iris dataset

Experiment - 02

Aim: To apply Data Cleaning techniques

Theory: Data Cleaning using Pandas in Python is the most important task that a data science professional should do. Wrong or bad-quality data can be detrimental to processes and analysis. Clean data will ultimately increase overall productivity and permit the very best quality information in decision-making.

Data Cleaning Cycle: It is the method of analyzing, distinguishing, and correcting untidy, raw data. Python Pandas Data Cleaning involves filling in missing values, handling outliers, and distinguishing and fixing errors in the dataset. Meanwhile, the techniques used for data cleaning in data science using Python might vary in step with different types of datasets. In this tutorial, we will learn how to clean data using pandas.

Signs of an untidy dataset

We have to take a closer look to find common signs of a messy dataset. These common signs are as follows:-

• **Missing numerical data:** Missing numerical data needs to be identified and addressed. Either they need to be deleted or replaced with a suitable test statistic.

• **Untidy data:** Untidy dataset can contain multiple problems. They prevent us from transforming the messy dataset into a clean dataset that is suitable for analysis.

• **Unexpected data values:** Mismatched data types of a column and data values can cause potential problems. They need to be investigated and solved.

• **Inconsistent column names:** Column names contain inconsistent capitalizations and bad characters. They need to be addressed properly.

• **Outliers:** Outliers need to be detected. They pose potential problems that need to be investigated and removed.

• **Duplicate rows and columns:** Duplicate rows and columns make data redundant. They can bias an analysis. Hence, they need to be found and dropped.

Code:

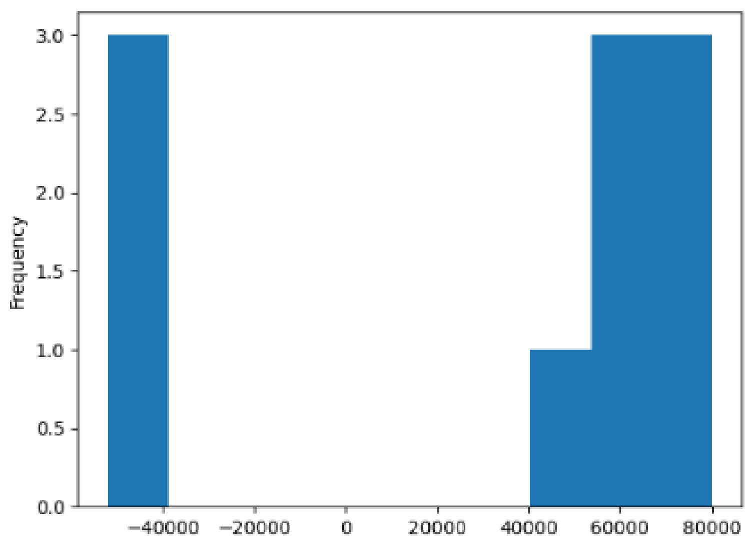
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

data = "/content/sample_data/fictitious_dataset.csv"
df = pd.read_csv(data)

print(df.shape)
print(df.head())
print(df.tail())
print(df.info())
print(df.dtypes)
print(df.describe)
print(df.columns)
```

```
(10, 9)
   id age_sex  height_cm  weight_kg  income_usd   bmi  hours_sleep  exercise_hours_weekly city
0   1    25_M        175         70       50000  23.5            6                    5.0   NY
1   2    34_F        160         55      -45000  21.1            7                    NaN   LA
2   3    29_M        180         82       60000  25.3            5                    4.0   TX
3   4    42_F        158         50       55000  20.1            8                    6.0   SF
4   5    31_M        172         76       70000  26.1            6                    NaN  CHI
   id age_sex  height_cm  weight_kg  income_usd   bmi  hours_sleep  exercise_hours_weekly city
5   6    27_F        165         59      -52000  21.7            7                    3.0  MIA
6   7    38_M        178         88       80000  28.4            4                    7.0  SEA
7   8    50_F        155         48       62000  19.9            9                    2.0  DEN
8   9    22_M        182         85       73000  27.0            3                    8.0  HOU
9  10    45_F        159         52      -49000  20.5           10                    NaN  BOS


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     10 non-null     int64  
 1   age_sex                10 non-null     object 
 2   height_cm              10 non-null     int64  
 3   weight_kg              10 non-null     int64  
 4   income_usd             10 non-null     int64  
 5   bmi                    10 non-null     float64
 6   hours_sleep            10 non-null     int64  
 7   exercise_hours_weekly  7 non-null      float64
 8   city                   10 non-null     object 
dtypes: float64(2), int64(5), object(2)
memory usage: 852.0+ bytes
```

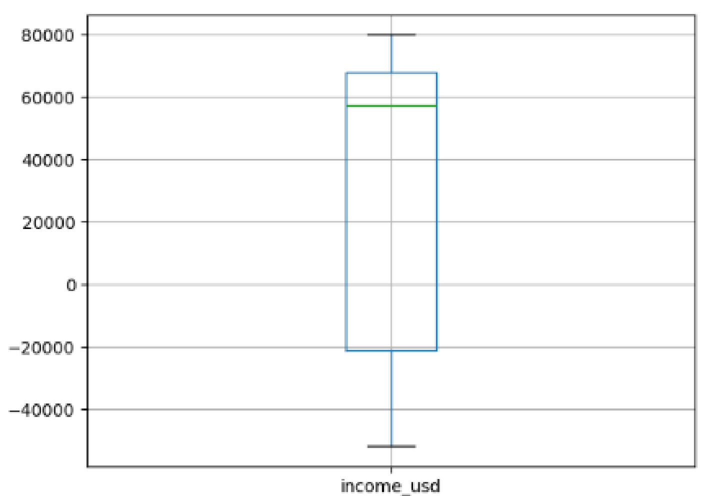
```
id                        int64
age_sex                  object
height_cm                 int64
weight_kg                 int64
income_usd                int64
bmi                     float64
hours_sleep               int64
exercise_hours_weekly   float64
city                     object
dtype: object
```

```
<bound method NDFrame.describe of    id age_sex  height_cm  weight_kg  income_usd   bmi  hours_sleep  exercise_hours_weekly city
0   1    25_M      175        70       50000     23.5      6             5.0            NY
1   2    34_F      160        55      -45000     21.1      7             NaN            LA
2   3    29_M      180        82       60000     25.3      5             4.0            TX
3   4    42_F      158        50       55000     20.1      8             6.0            SF
4   5    31_M      172        76       70000     26.1      6             NaN           CHI
5   6    27_F      165        59      -52000     21.7      7             3.0           MIA
6   7    38_M      178        88       80000     28.4      4             7.0           SEA
7   8    50_F      155        48       62000     19.9      9             2.0           DEN
8   9    22_M      182        85       73000     27.0      3             8.0           HOU
9  10    45_F      159        52      -49000     20.5     10             NaN           BOS>
Index(['id', 'age_sex', 'height_cm', 'weight_kg', 'income_usd', 'bmi', 'hours_sleep',
       'exercise_hours_weekly', 'city'],
      dtype='object')
```

```
df['income_usd'].plot(kind='hist')
plt.show()
```




```
df.boxplot(column='income_usd')
plt.show()
```



```
df[['age','sex']] = df.age_sex.str.split("_", expand = True)
df.drop(['age_sex'], axis=1, inplace=True)
df = df[['id', 'age', 'sex', 'height_cm', 'weight_kg', 'income_usd',
'bmi', 'hours_sleep',
       'exercise_hours_weekly', 'city']]

df['income_usd'].replace(-45000, 45000, inplace=True)
df['income_usd'].replace(-52000, 52000, inplace=True)
df['income_usd'].replace(-49000, 49000, inplace=True)

df.isnull().sum()
mean = df['exercise_hours_weekly'].mean()
df['exercise_hours_weekly'].fillna(value = mean, inplace=True)
df
```

	id	age	sex	height_cm	weight_kg	income_usd	bmi	hours_sleep	exercise_hours_weekly	city
0	1	25	M	175	70	50000	23.5	6	5.0	NY
1	2	34	F	160	55	45000	21.1	7	5.0	LA
2	3	29	M	180	82	60000	25.3	5	4.0	TX
3	4	42	F	158	50	55000	20.1	8	6.0	SF
4	5	31	M	172	76	70000	26.1	6	5.0	CHI
5	6	27	F	165	59	52000	21.7	7	3.0	MIA
6	7	38	M	178	88	80000	28.4	4	7.0	SEA
7	8	50	F	155	48	62000	19.9	9	2.0	DEN
8	9	22	M	182	85	73000	27.0	3	8.0	HOU
9	10	45	F	159	52	49000	20.5	10	5.0	BOS

Conclusion: Hence, we performed data cleaning on a fictitious dataset using Pandas

Experiment - 03

Aim: To explore the inferential statistics t-test on the given dataset

Theory: Inferential statistics allows us to make conclusions about a population based on a sample of data. One of the key methods used in inferential statistics is hypothesis testing, which helps us determine if observed differences between groups are statistically significant.
A t-test is a statistical test used to compare the means of two independent groups to determine whether the observed difference is due to chance or a significant factor.

The t-test is used to determine whether the mean age of passengers who survived is significantly different from those who did not.

Null Hypothesis ($H0$): There is no significant difference in the average age between passengers who survived and those who did not.
Alternative Hypothesis ($H1$): There is a significant difference in the average age between the two groups.

There are different types of t-tests used in statistical analysis:

1. **Independent (Unpaired) T-Test** – Compares means between two unrelated groups.
1. **Paired T-Test** – Compares means within the same group before and after a condition.
2. **One-Sample T-Test** – Compares the mean of a single group against a known population mean.

For this experiment, we use an **Independent T-Test** since we are comparing two separate groups:

- **Passengers who survived (Survived = 1)**
- **Passengers who did not survive (Survived = 0)**

Code:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd

df = pd.read_csv(".../Titanic-Dataset.csv")
df.shape
display(df.head())

# Drop missing values in the 'Age' column
df = df.dropna(subset=['Age'])
```

```
# Split data into Survived and Not Survived groups
survived_age = df[df["Survived"] == 1]["Age"]
not_survived_age = df[df["Survived"] == 0]["Age"]

# Perform independent t-test
t_stat, p_value = stats.ttest_ind(survived_age, not_survived_age,
equal_var=False)

# Print the results
print("\nT-Test Results:")
print(f"T-Statistic: {t_stat:.4f}")
print(f"P-Value: {p_value:.4f}")

# Interpret the results
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant
difference in average age between survivors and non-survivors.")
else:
    print("Fail to reject the null hypothesis: No significant
difference in average age between survivors and non-survivors.")
```

Output:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
T-Test Results:
T-Statistic: -2.0460
P-Value: 0.0412
Reject the null hypothesis: There is a significant difference in average age between survivors and non-survivors.
```

Conclusion: Hence, we performed inferential statistics t-test on the given dataset.

Experiment - 04

Aim: To perform Data Visualization techniques

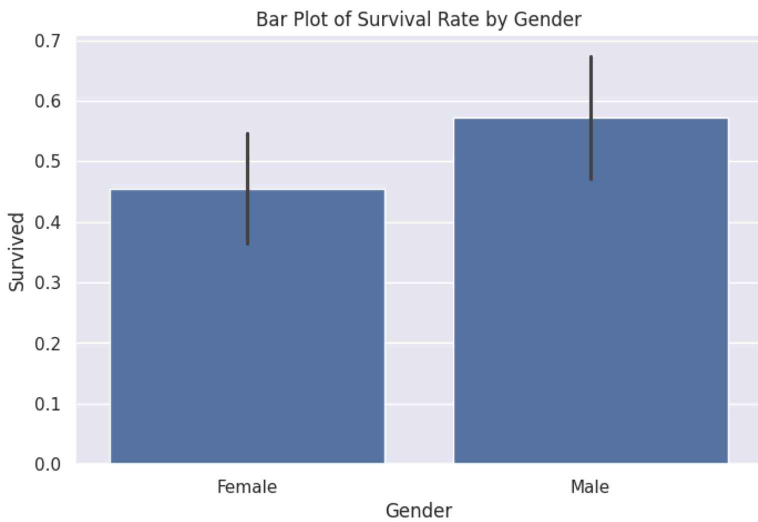
Theory: Data visualization is the graphical representation of information and data. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data. Additionally, it provides an excellent way for employees or business owners to present data to non-technical audiences without confusion.
Our eyes are drawn to colors and patterns. We can quickly identify red from blue, and squares from circles. Data visualization is another form of visual art that grabs our interest and keeps our eyes on the message. When we see a chart, we quickly see trends and outliers.

Data visualization techniques are ways to represent data visually to make it easier to understand. Some techniques include:

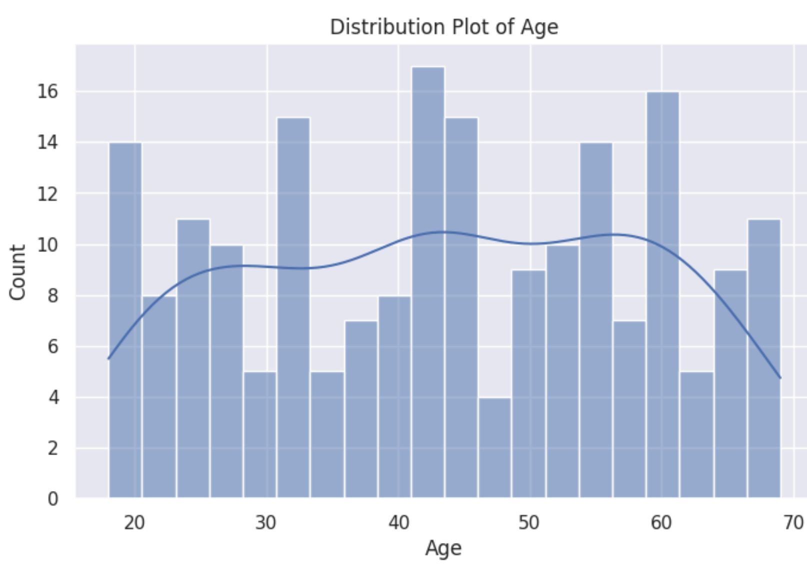
- **Bar plots:** A bar plot uses rectangular bars to represent data categories, with bar length or height proportional to their values. It compares discrete categories, with one axis for categories and the other for values.
- **Histograms:** Histogram is a type of graphical representation used in statistics to show the distribution of numerical data. It looks somewhat like a bar chart, but unlike bar graphs, which are used for categorical data, histograms are designed for continuous data, grouping it into logical ranges which are also known as "bins."
- **Box plots:** Box Plot is a graphical method to visualize data distribution for gaining insights and making informed decisions. Box plot is a type of chart that depicts a group of numerical data through their quartiles.
- **Count plots:** The countplot is used to represent the occurrence(counts) of the observation present in the categorical variable. It uses the concept of a bar chart for the visual depiction.
- **Scatter plots:** A scatter plot (aka scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between variables.
- **Pie Charts:** A pie chart is a type of graph representing data in a circular form, with each slice of the circle representing a fraction or proportionate part of the whole. All slices of the pie add up to make the whole equaling 100 percent.
- **Line plots:** A line plot is a type of graph that displays data points along a number line. It is basically useful to provide a clear and concise representation of trends, patterns, and changes that occur over time.

Code & Output:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd
import kagglehub

path = kagglehub.dataset_download("himelsarder/road-accident-survival-dataset")
df = pd.read_csv(path + "/accident.csv")
df.shape
df.head()

plt.figure(figsize=(8,5))
sns.barplot(x="Gender", y="Survived", data=df)
plt.title("Bar Plot of Survival Rate by Gender")
plt.show()
```

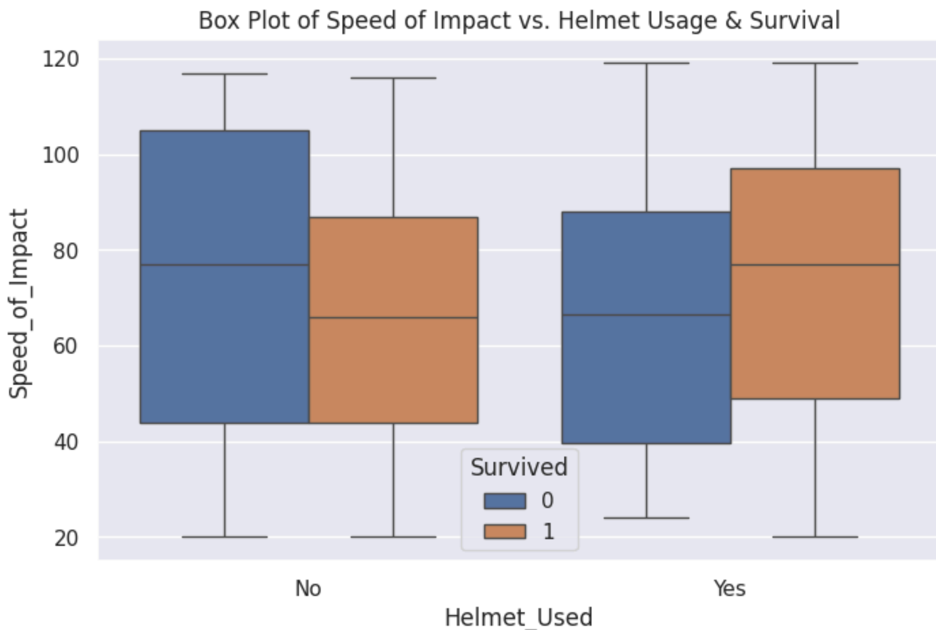



```
plt.figure(figsize=(8,5))
sns.histplot(df["Age"], bins=20, kde=True)
plt.title("Distribution Plot of Age")
plt.show()
```

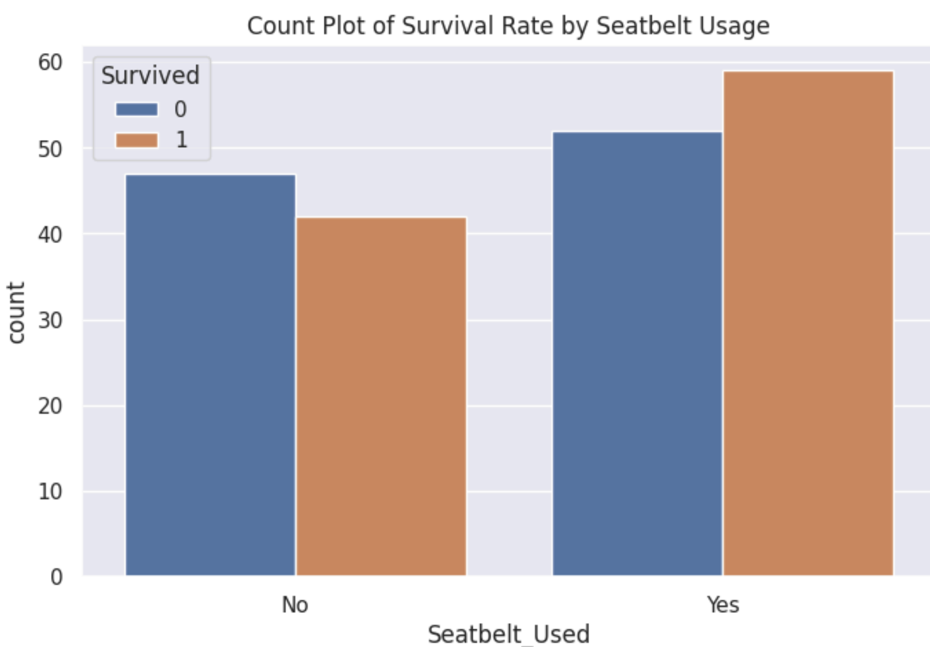



```
plt.figure(fig
```

```
sns.boxplot(x="Helmet_Used", y="Speed_of_Impact", hue="Survived", data=df)
plt.title("Box Plot of Speed of Impact vs. Helmet Usage & Survival")
plt.show()
```

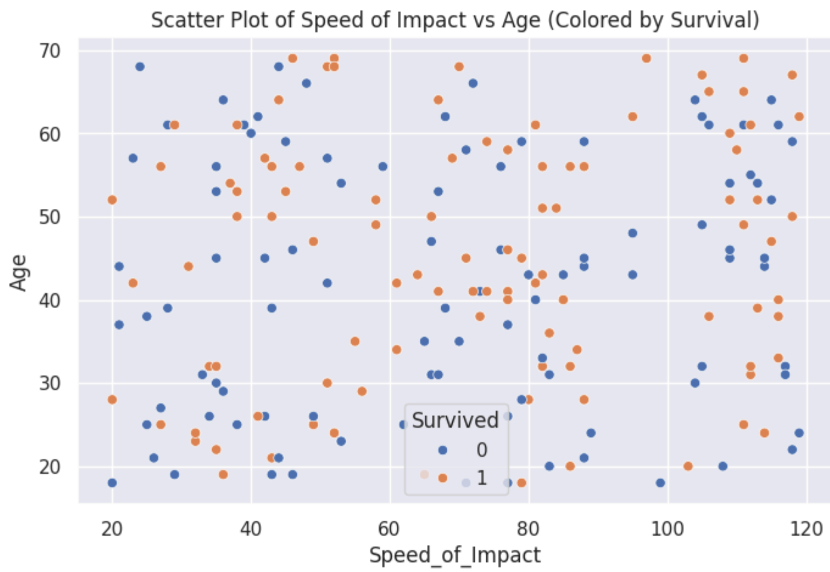



```
plt.figure(figsize=(8, 5))
sns.countplot(x="Seatbelt_Used", hue="Survived", data=df)
plt.title("Count Plot of Survival Rate by Seatbelt Usage")
plt.show()
```

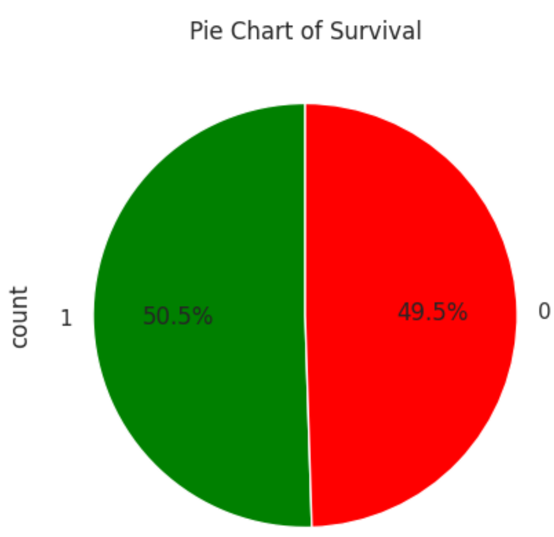



```
plt.figure(figsize=(8, 5))
```

```
sns.scatterplot(x="Speed_of_Impact", y="Age", hue="Survived", data=df)
plt.title("Scatter Plot of Speed of Impact vs Age (Colored by Survival)")
plt.show()
```




```
plt.figure(figsize=(8, 5))
counts = df['Survived'].value_counts()
counts.plot(kind='pie', autopct='%1.1f%%', startangle=90,
colors=['green','red'])
plt.title('Pie Chart of Survival')
plt.show()
```



Conclusion: Hence, we performed data visualization techniques on a Road Accidents dataset

Experiment - 05

Aim: To implement Linear Regression and evaluate the performance evaluation metrics

Theory: Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

There is one dependent or output variable which represents the Advertising data and is denoted by y. We want to build a linear relationship between these variables. This linear relationship can be modelled by mathematical equation of the form:- $Y = \beta 0 + \beta 1 * X$ ------------- (1)

In this equation, X and Y are called independent and dependent variables respectively,

$\beta 1$ is the coefficient for independent variable and

$\beta 0$ is the constant term.

$\beta 0$ and $\beta 1$ are called parameters of the model.

For simplicity, we can compare the above equation with the basic line equation of the form:-

$$y = ax + b \qquad \text{----------------- (2)}$$

We can see that

slope of the line is given by, $a = \beta 1$, and

intercept of the line by $b = \beta 0$.

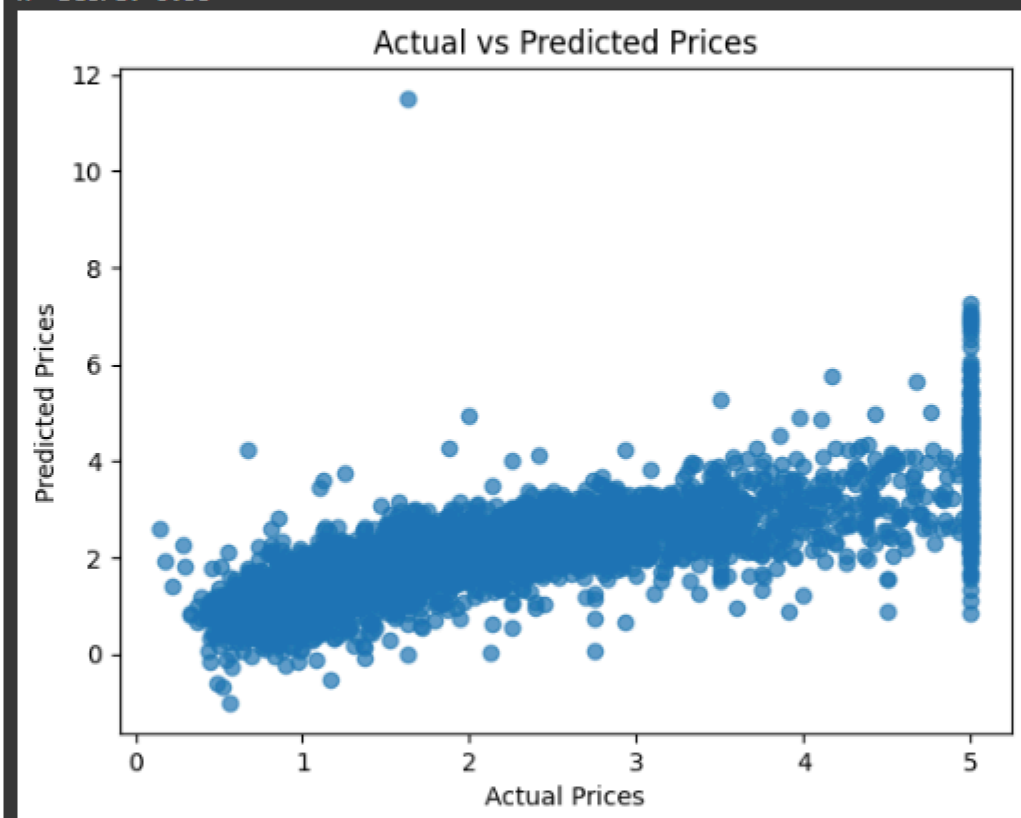
In this Simple Linear Regression model, we want to fit a line which estimates the linear relationship between X and Y. So, the question of fitting reduces to estimating the parameters of the model $\beta 0$ and $\beta 1$.

Code & Output:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error,
r2_score
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import MinMaxScaler  # Import MinMaxScaler

# Load California housing dataset
housing = fetch_california_housing()
df = pd.DataFrame(housing.data, columns=housing.feature_names)
df['PRICE'] = housing.target  # Add target variable

# Define features & target
X = df.drop(columns=['PRICE'])  # Features
y = df['PRICE']  # Target variable

# Split into train & test sets (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Create and fit MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train the model with scaled data
model = LinearRegression()
model.fit(X_train_scaled, y_train)

# Predictions with scaled data
y_pred = model.predict(X_test_scaled)

# Performance Metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Print results
print(f"Mean Absolute Error: {mae:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
```

```
print(f"Root Mean Squared Error: {rmse:.2f}")
print(f"R² Score: {r2:.2f}")

# Plot actual vs predicted prices
plt.scatter(y_test, y_pred, alpha=0.7)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted Prices")
plt.show()
)
```

```
Mean Absolute Error: 0.53
Mean Squared Error: 0.56
Root Mean Squared Error: 0.75
R² Score: 0.58
```



Conclusion: Hence, we implemented Linear Regression and evaluated the performance evaluation metrics

Experiment - 06

Aim: To implement SMOTE techniques to generate synthetic data to solve the problem of class imbalance

Theory: SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them. SMOTE synthesises new minority instances between existing minority instances.

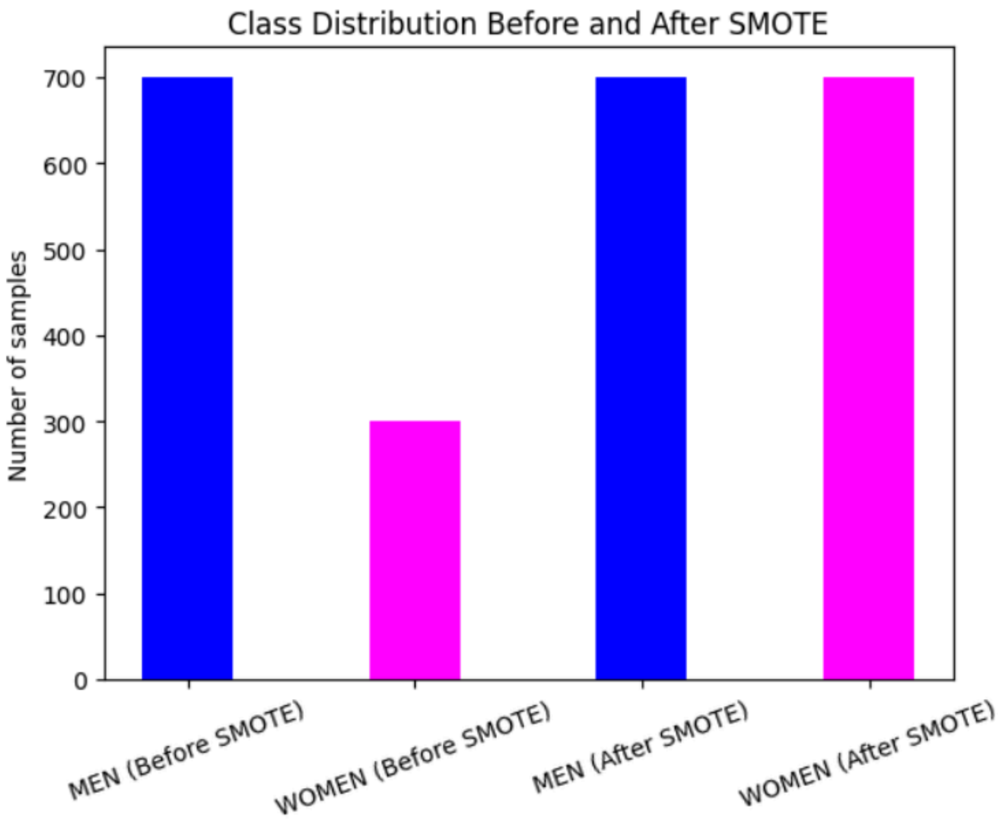
These synthetic training records are generated by randomly selecting one or more of the k-nearest neighbors for each example in the minority class. After the oversampling process, the data is reconstructed and several classification models can be applied for the processed data.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE

# Step 1: Create an imbalanced dataset
X, y = make_classification(n_classes=2, class_sep=2, weights=[0.7,
0.3],n_informative=3, n_redundant=1, flip_y=0,n_features=5,
n_clusters_per_class=1, n_samples=1000, random_state=42)

# Apply SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Class distributions
before_counts = Counter(y)
after_counts = Counter(y_resampled)

# Set positions for bars
x_labels = ["MEN (Before SMOTE)", "WOMEN (Before SMOTE)", "MEN (After
SMOTE)", "WOMEN (After SMOTE)"]
x_positions = np.arange(len(x_labels))  # Generate equally spaced
positions

# Plot the bars
plt.bar(x_positions[:2], before_counts.values(), color=['blue',
'fuchsia'], width=0.4, label="Before SMOTE")
```

```
plt.bar(x_positions[2:], after_counts.values(), color=['blue',
'fuchsia'], width=0.4, label="After SMOTE")

# Adjust x-axis labels
plt.xticks(x_positions, x_labels, rotation=20)  # Rotate labels for
better readability
plt.ylabel("Number of samples")
plt.title("Class Distribution Before and After SMOTE")
plt.show()
```

Output:



Conclusion: Hence, we successfully implemented SMOTE to solve the problem of imbalanced datasets

Experiment - 07

Aim: To implement outlier detection using Density-Based method.

Theory:

Density-Based Outlier Detection: Density-based methods identify outliers based on the density of data points in their neighborhood. A point is considered an outlier if it resides in a low-density region, significantly differing from the dense clusters.

These methods are particularly useful for:

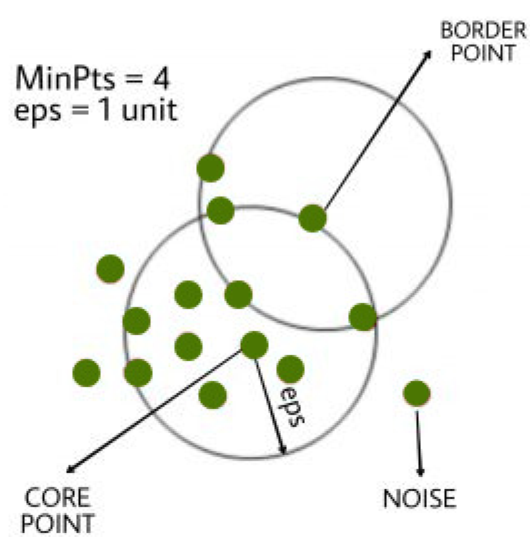
- Handling non-linear distributions of data.
- Detecting global as well as local outliers.
- Working well with high-dimensional data.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise): DBSCAN is a density-based clustering algorithm that also identifies outliers as points that do not belong to any cluster. It relies on two parameters:

- **ε (epsilon):** Defines the radius of a neighborhood around a point.
- **minPts:** The minimum number of points required to form a dense region.

A point is classified as:

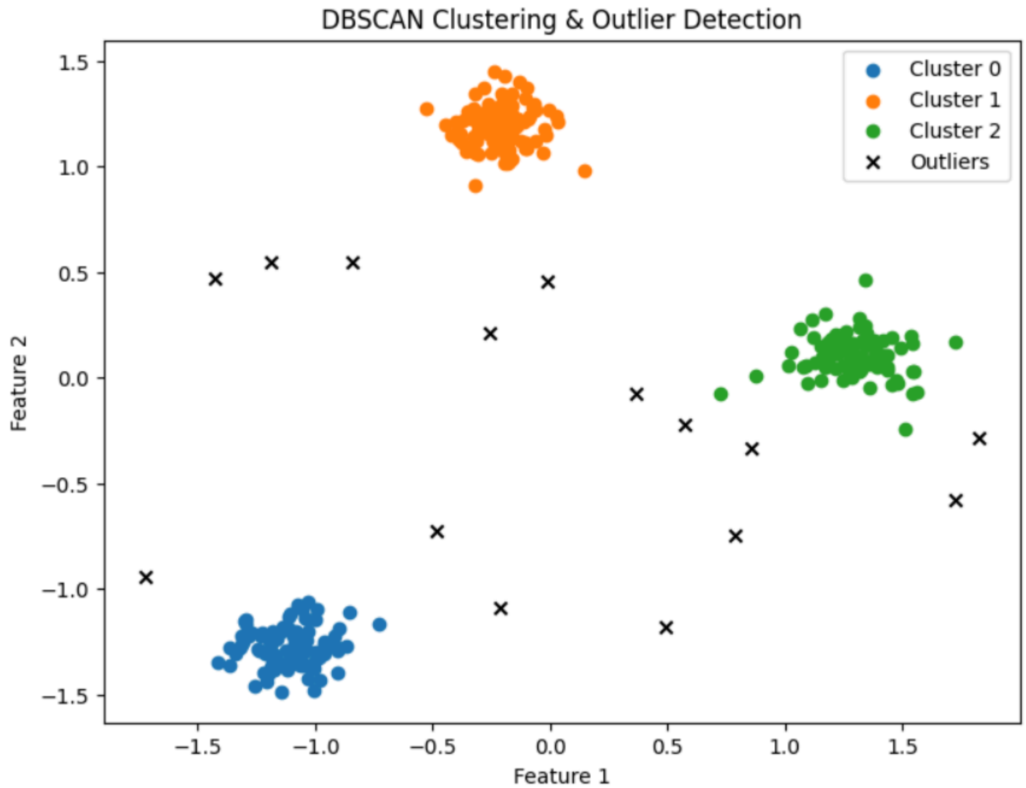
- **Core point:** If it has at least `minPts` within radius `ε`.
- **Border point:** If it has fewer than `minPts` neighbors but is in a core point's neighborhood.
- **Outlier (Noise point):** If it does not satisfy the above conditions.



Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generate synthetic data with clusters and some noise
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.6,
random_state=42)
X = np.vstack([X, np.random.uniform(low=-10, high=10, size=(20, 2))])
# Add noise points

# Standardize features (DBSCAN is sensitive to scale)
X = StandardScaler().fit_transform(X)

# Apply DBSCAN
dbscan = DBSCAN(eps=0.3, min_samples=5)
labels = dbscan.fit_predict(X)

# Identify outliers (DBSCAN labels them as -1)
outliers = X[labels == -1]

# Plot results
plt.figure(figsize=(8,6))
unique_labels = set(labels)
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k']
for label in unique_labels:
    if label == -1:
        # Outliers in black
        plt.scatter(outliers[:, 0], outliers[:, 1], c='black',
marker='x', label='Outliers')
    else:
        plt.scatter(X[labels == label, 0], X[labels == label, 1],
label=f'Cluster {label}')

plt.legend()
plt.title("DBSCAN Clustering & Outlier Detection")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

Output:



Conclusion: Hence, we successfully implemented outlier detection using Density-Based method

Experiment - 08

Aim: To implement Time series decomposition and moving averages method of trend.

Theory:
Time Series Analysis: A **time series** is a sequence of observations recorded over time. Analyzing time series data helps identify trends, seasonality, and irregular components, which is useful in forecasting and decision-making.

There are two primary methods:

1. **Time Series Decomposition** (Breaking a time series into its components)
2. **Moving Averages** (Smoothing a time series to observe trends)

Time series decomposition breaks down a time series into the following components:

1. **Trend (Tt):** The long-term movement in the data, indicating an overall increase or decrease.
2. **Seasonality (St):** Periodic fluctuations in the data that occur at regular intervals (e.g., daily, monthly, yearly).
3. **Residual (Rt):** The irregular variations or noise in the data that cannot be explained by trend or seasonality.

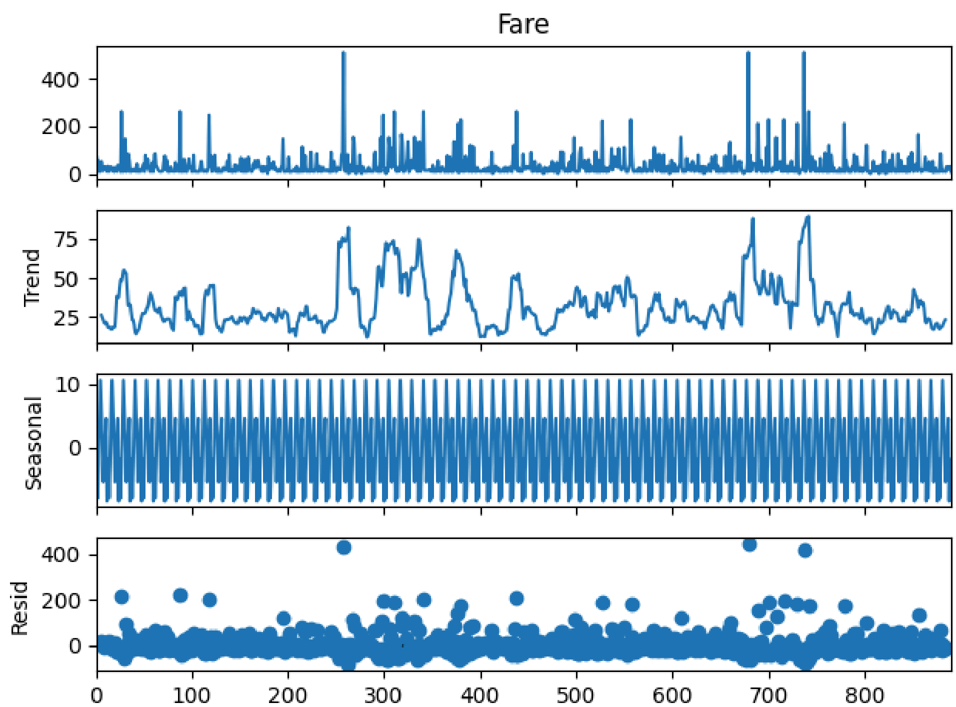
Moving Average Method of Trend: The Moving Averages Method of Trend is a fundamental technique in time series analysis used to smooth out short-term fluctuations and highlight the underlying long-term trend. It works by averaging a set number of past observations over a sliding window, effectively reducing noise and making patterns more visible.

There are different types of moving averages:

- **Simple Moving Average (SMA):** The unweighted average of previous data points over a fixed period.
- **Weighted Moving Average (WMA):** Assigns more weight to recent observations for a better reflection of recent trends.
- **Exponential Moving Average (EMA):** Gives exponentially decreasing weights to older data, making it more responsive to recent changes.

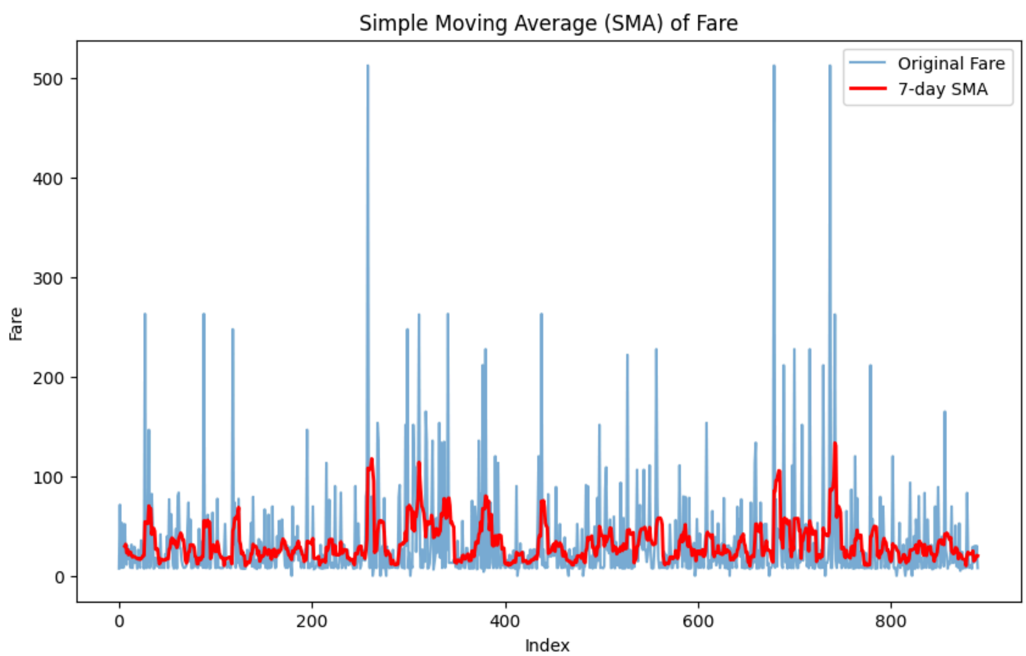
Code for Time Series Decomposition:

```
import pandas as pd
import kagglehub
from statsmodels.tsa.seasonal import seasonal_decompose

path = kagglehub.dataset_download("yasserh/titanic-dataset")
titanic = pd.read_csv(os.path.join(path, "Titanic-Dataset.csv"))
titanic['Fare'] = pd.to_numeric(titanic['Fare'], errors='coerce')
titanic['Index'] = pd.RangeIndex(start=0, stop=len(titanic), step=1)
titanic.set_index('Index', inplace=True)
decomposition = seasonal_decompose(titanic['Fare'], model='additive',
period=12)  # Adjust period as needed
decomposition.plot()
plt.show()
```



Code for Moving Averages:

```
import pandas as pd
import matplotlib.pyplot as plt
titanic['Fare'] = pd.to_numeric(titanic['Fare'], errors='coerce')
titanic['SMA'] = titanic['Fare'].rolling(window=7).mean()
plt.figure(figsize=(10, 6))
plt.plot(titanic['Fare'], label='Original Fare', alpha=0.6)
plt.plot(titanic['SMA'], label='7-day SMA', color='red', linewidth=2)
plt.legend()
plt.title('Simple Moving Average (SMA) of Fare')
plt.xlabel('Index')
plt.ylabel('Fare')
plt.show()
```



Conclusion: Hence, we successfully implemented outlier detection using the Density-Based method.