Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**

EXPERIMENT NO:1

Aim: Introduction to Data science and Data preparation using Pandas steps.

Theory: Data Science is an interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from structured and unstructured data. It combines expertise from domains like mathematics, statistics, computer science, and business to analyze and interpret data for decision-making.

Key Components of Data Science:

What is Data Science?

- Data Collection: Gathering data from various sources like databases, APIs, web scraping, or surveys.
- Data Preparation: Cleaning, transforming, and organizing raw data to make it suitable for analysis.
- Exploratory Data Analysis (EDA): Visualizing and summarizing data to understand patterns, relationships, and distributions.
- Model Building: Using statistical or machine learning algorithms to solve problems or make predictions.
- Model Deployment: Integrating models into real-world applications for practical use.
- Insights and Reporting: Interpreting results to drive decision-making.

What is Data Preparation?

Data preparation is the process of cleaning, transforming, and organizing raw data into a format that can be used for analysis or modeling. It is a crucial step in the data science workflow as raw data is often messy, incomplete, or inconsistent.

Steps in Data Preparation Using Pandas

1) **Load Data:** Data can be loaded into Pandas from various formats like CSV, Excel, or JSON using:

```
import pandas as pd
df = pd.read_csv('data.csv')  # Load data from a CSV file
```

2) **Description of the Dataset:** Use Pandas to get an overview of the dataset:

```
print(df.info())  # Dataset structure, data types, and memory usage
print(df.describe())  # Statistical summary of numeric columns
```

3) **Drop Columns:** Remove irrelevant or unnecessary columns that do not contribute to the analysis.

```
df = df.drop(['column_name'], axis=1)  # Drop specific columns
```

4) Handle Missing Data:
Drop rows or columns with excessive missing values:

```
df = df.dropna()  # Drop rows with missing values
```

Fill missing data with meaningful values:

```
df['column_name'].fillna(df['column_name'].mean(), inplace=True)  # Replace missing values
with mean
```

5) Create Dummy Variables: Convert categorical data into numerical format using one-hot encoding:

```
df = pd.get_dummies(df, columns=['categorical_column'], drop_first=True)
```

6) Detect and Handle Outliers: Outliers can be identified manually by visualizing distributions (e.g., boxplots) or using statistical methods like the Interquartile Range (IQR):

```
Q1 = df['column_name'].quantile(0.25)
Q3 = df['column_name'].quantile(0.75)
IQR = Q3 - Q1
outlier_mask = (df['column_name'] < (Q1 - 1.5 * IQR)) | (df['column_name'] > (Q3 + 1.5 *
IQR))
df_no_outliers = df[~outlier_mask]
```

Standardization and Normalization: Scale data to ensure features are on a similar range:

Standardization: Transform data to have a mean of 0 and standard deviation of 1.
Normalization: Scale values between 0 and 1.
Examples of both methods are provided in the earlier section.

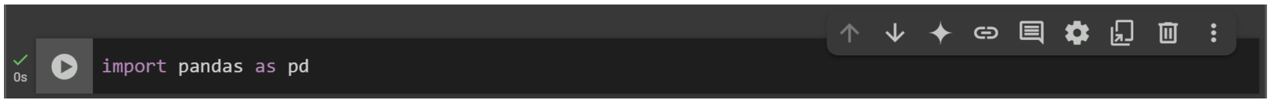
Importance of Data Preparation

- Improves Data Quality: Ensures data is clean, consistent, and free of errors.
- Boosts Model Performance: Preprocessed data helps machine learning models perform better and converge faster.
- Reduces Bias: Identifies and removes inconsistencies, outliers, and missing values to prevent skewed results.
- Facilitates Interpretability: Organized and clean data is easier to understand and visualize.
- Data preparation is often the most time-consuming yet critical step in a data science project. Properly prepared data leads to more accurate insights and successful outcomes.

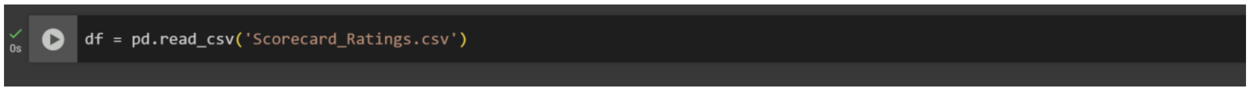
Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**

Performed Experiment:

Step 1: Firstly import Pandas Library as pd an then Load data in Pandas using pd.read_csv.

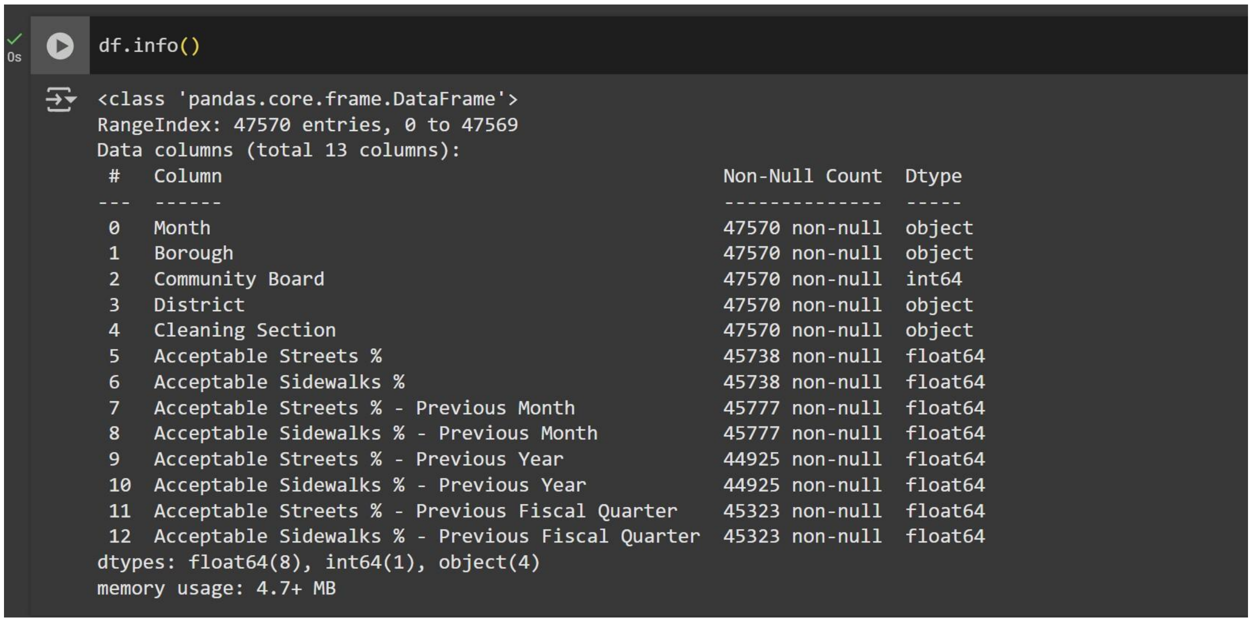


```
import pandas as pd
```



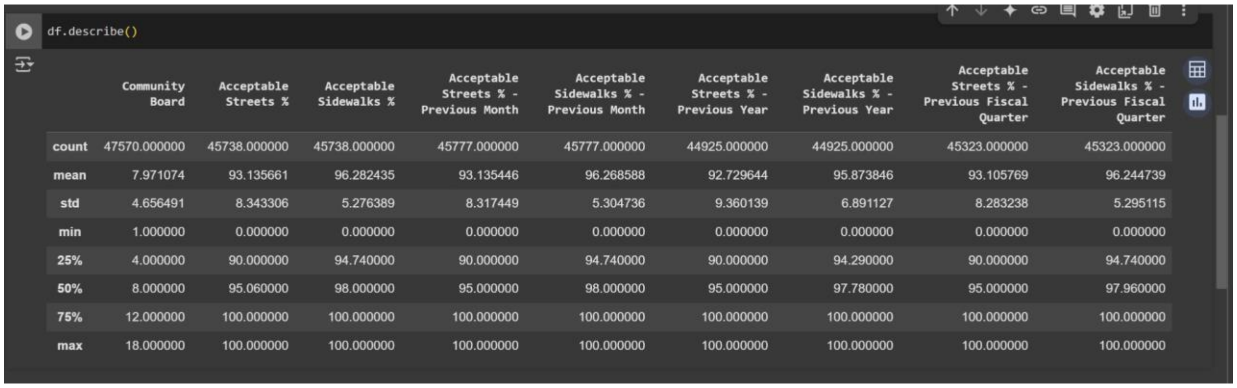
```
df = pd.read_csv('Scorecard_Ratings.csv')
```

Step2: Get Description of the Dataset by using following 2 commands df.info() -> Get basic information about the dataset df.describe() -> Summary statistics of the dataset



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 47570 entries, 0 to 47569
Data columns (total 13 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   Month                                         47570 non-null  object
 1   Borough                                       47570 non-null  object
 2   Community Board                               47570 non-null  int64
 3   District                                      47570 non-null  object
 4   Cleaning Section                              47570 non-null  object
 5   Acceptable Streets %                          45738 non-null  float64
 6   Acceptable Sidewalks %                        45738 non-null  float64
 7   Acceptable Streets % - Previous Month         45777 non-null  float64
 8   Acceptable Sidewalks % - Previous Month       45777 non-null  float64
 9   Acceptable Streets % - Previous Year          44925 non-null  float64
 10  Acceptable Sidewalks % - Previous Year        44925 non-null  float64
 11  Acceptable Streets % - Previous Fiscal Quarter  45323 non-null  float64
 12  Acceptable Sidewalks % - Previous Fiscal Quarter  45323 non-null  float64
dtypes: float64(8), int64(1), object(4)
memory usage: 4.7+ MB
```

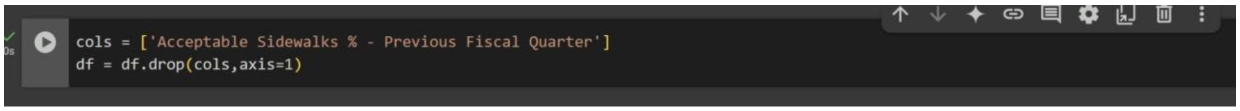


```
df.describe()
```

	Community Board	Acceptable Streets %	Acceptable Sidewalks %	Acceptable Streets % - Previous Month	Acceptable Sidewalks % - Previous Month	Acceptable Streets % - Previous Year	Acceptable Sidewalks % - Previous Year	Acceptable Streets % - Previous Fiscal Quarter	Acceptable Sidewalks % - Previous Fiscal Quarter
count	47570.000000	45738.000000	45738.000000	45777.000000	45777.000000	44925.000000	44925.000000	45323.000000	45323.000000
mean	7.971074	93.135661	96.282435	93.135446	96.268588	92.729644	95.873846	93.105769	96.244739
std	4.656491	8.343306	5.276389	8.317449	5.304736	9.360139	6.891127	8.283238	5.295115
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	4.000000	90.000000	94.740000	90.000000	94.740000	90.000000	94.290000	90.000000	94.740000
50%	8.000000	95.060000	98.000000	95.000000	98.000000	95.000000	97.780000	95.000000	97.960000
75%	12.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
max	18.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000

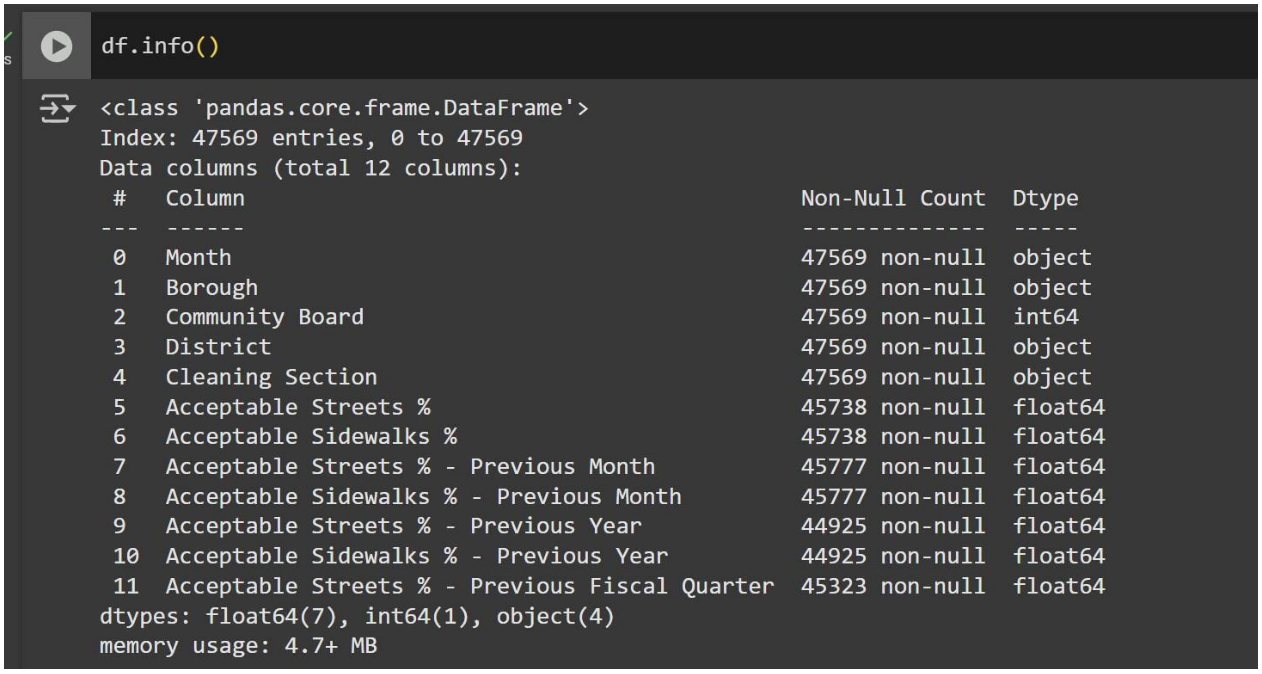
Step 3: Drop Columns that aren’t useful. From Our Dataset we are dropping the “Acceptable Sidewalks” column

Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**



```
cols = ['Acceptable Sidewalks % - Previous Fiscal Quarter']
df = df.drop(cols,axis=1)
```

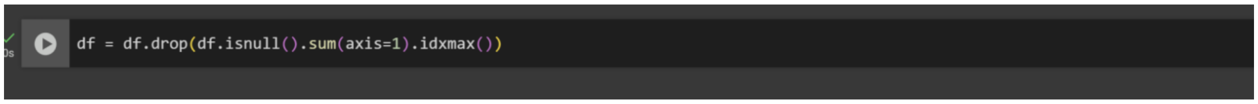
We can see that it returned total 9 columns as it dropped the column Acceptable Sidewalks



```
df.info()
```

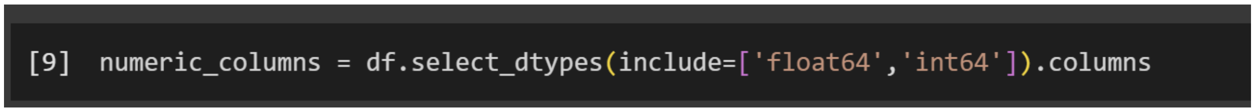
```
<class 'pandas.core.frame.DataFrame'>
Index: 47569 entries, 0 to 47569
Data columns (total 12 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   Month                                        47569 non-null  object
 1   Borough                                      47569 non-null  object
 2   Community Board                              47569 non-null  int64
 3   District                                     47569 non-null  object
 4   Cleaning Section                             47569 non-null  object
 5   Acceptable Streets %                         45738 non-null  float64
 6   Acceptable Sidewalks %                       45738 non-null  float64
 7   Acceptable Streets % - Previous Month        45777 non-null  float64
 8   Acceptable Sidewalks % - Previous Month      45777 non-null  float64
 9   Acceptable Streets % - Previous Year         44925 non-null  float64
 10  Acceptable Sidewalks % - Previous Year       44925 non-null  float64
 11  Acceptable Streets % - Previous Fiscal Quarter  45323 non-null  float64
dtypes: float64(7), int64(1), object(4)
memory usage: 4.7+ MB
```

Step 4: Drop row with maximum missing values. df.isnull().sum(axis=1) -> Computes the number of missing values (NaN) for each row. .idxmax() -> Returns the index of row with max. no. of missing value

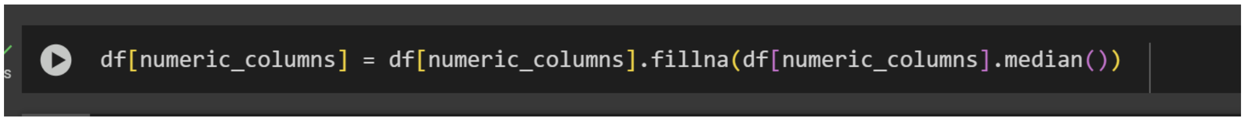


```
df = df.drop(df.isnull().sum(axis=1).idxmax())
```

Step 5: Taking care of missing data. We can fill the empty numeric values with mode or median or mean. Below we had filled it with median. Firstly we had fetched the numeric values and then using .fillna().median we had filled it.

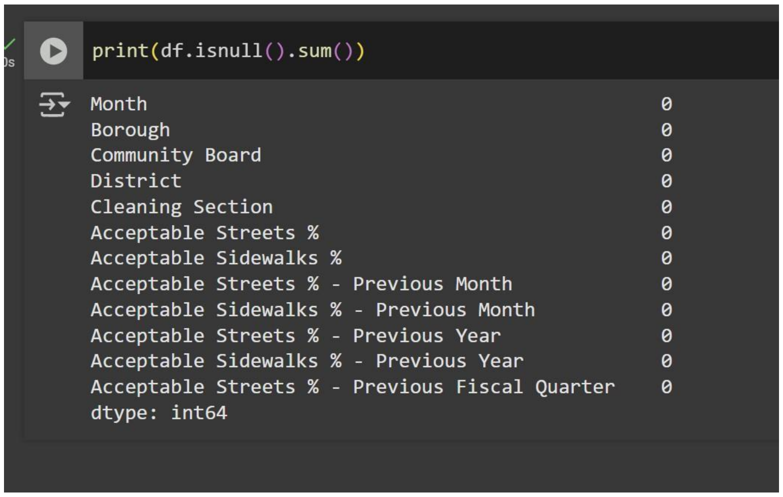


```
[9] numeric_columns = df.select_dtypes(include=['float64','int64']).columns
```



```
df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].median())
```

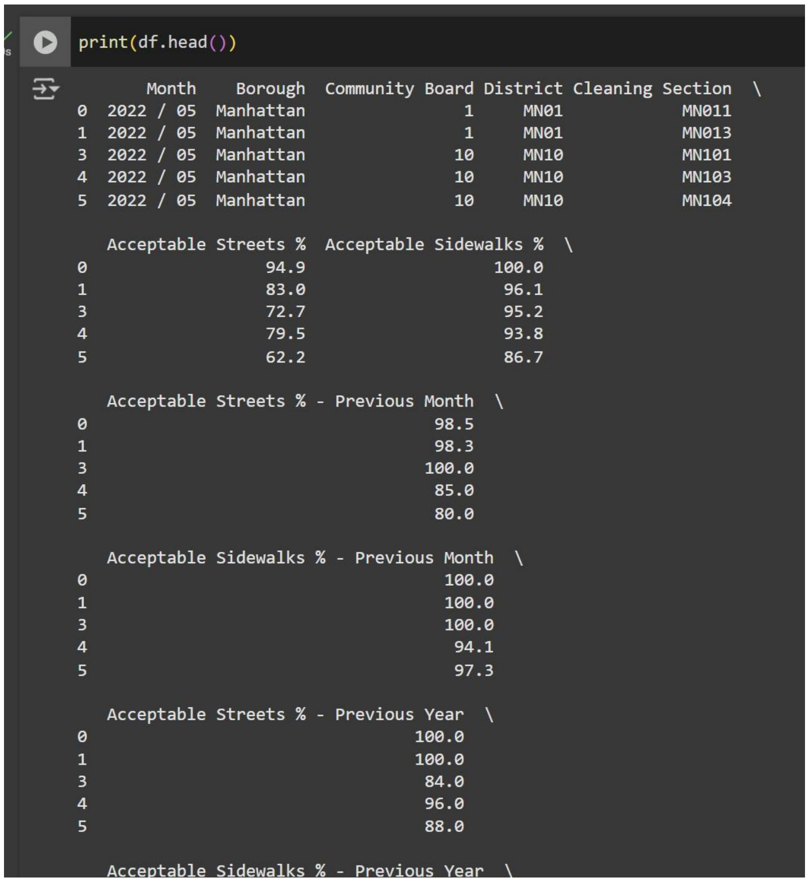
We can see that all the columns which had empty are filled. As they returned the sum 0



```
print(df.isnull().sum())
```

```
Month                                           0
Borough                                         0
Community Board                                 0
District                                        0
Cleaning Section                                0
Acceptable Streets %                            0
Acceptable Sidewalks %                          0
Acceptable Streets % - Previous Month           0
Acceptable Sidewalks % - Previous Month         0
Acceptable Streets % - Previous Year            0
Acceptable Sidewalks % - Previous Year          0
Acceptable Streets % - Previous Fiscal Quarter  0
dtype: int64
```

df.head() returns starting 5 values

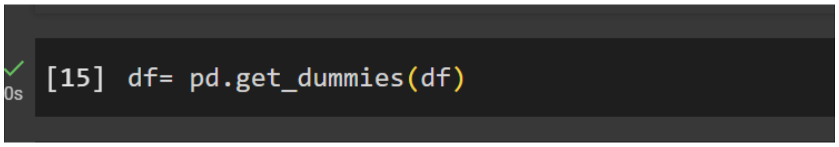


```
print(df.head())
```

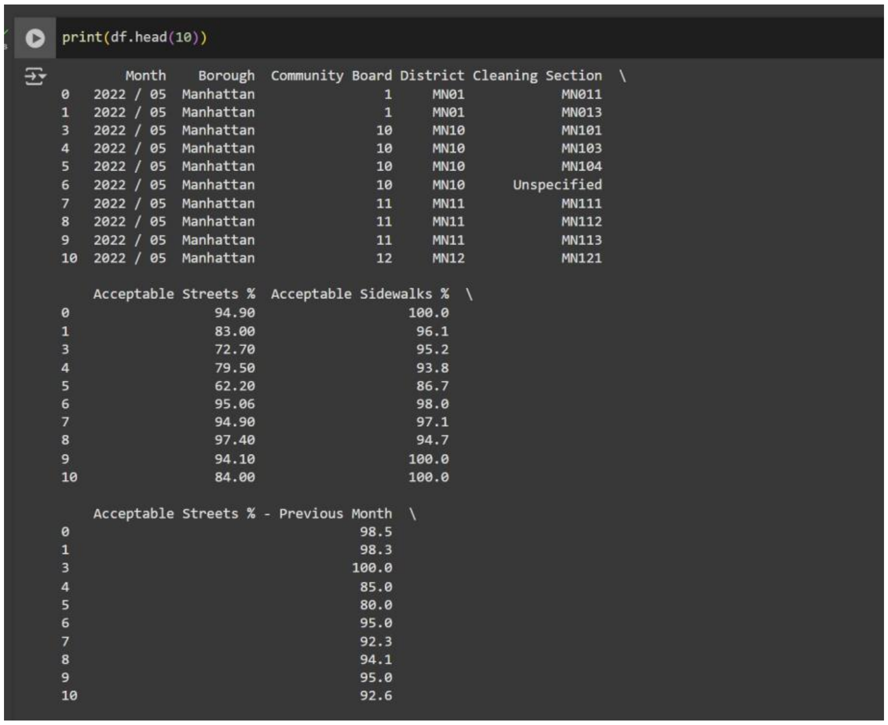
```
      Month    Borough  Community Board District Cleaning Section  \
0  2022 / 05  Manhattan                1     MN01            MN011
1  2022 / 05  Manhattan                1     MN01            MN013
3  2022 / 05  Manhattan               10     MN10            MN101
4  2022 / 05  Manhattan               10     MN10            MN103
5  2022 / 05  Manhattan               10     MN10            MN104

   Acceptable Streets %  Acceptable Sidewalks %  \
0                  94.9                   100.0
1                  83.0                    96.1
3                  72.7                    95.2
4                  79.5                    93.8
5                  62.2                    86.7

   Acceptable Streets % - Previous Month  \
0                                   98.5
1                                   98.3
3                                  100.0
4                                   85.0
5                                   80.0

   Acceptable Sidewalks % - Previous Month  \
0                                    100.0
1                                    100.0
3                                    100.0
4                                     94.1
5                                     97.3

   Acceptable Streets % - Previous Year  \
0                                 100.0
1                                 100.0
3                                  84.0
4                                  96.0
5                                  88.0

   Acceptable Sidewalks % - Previous Year  \
```

Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**

Step 6: Create dummy variables. By using the below commands separate columns are created for each unique value in a column



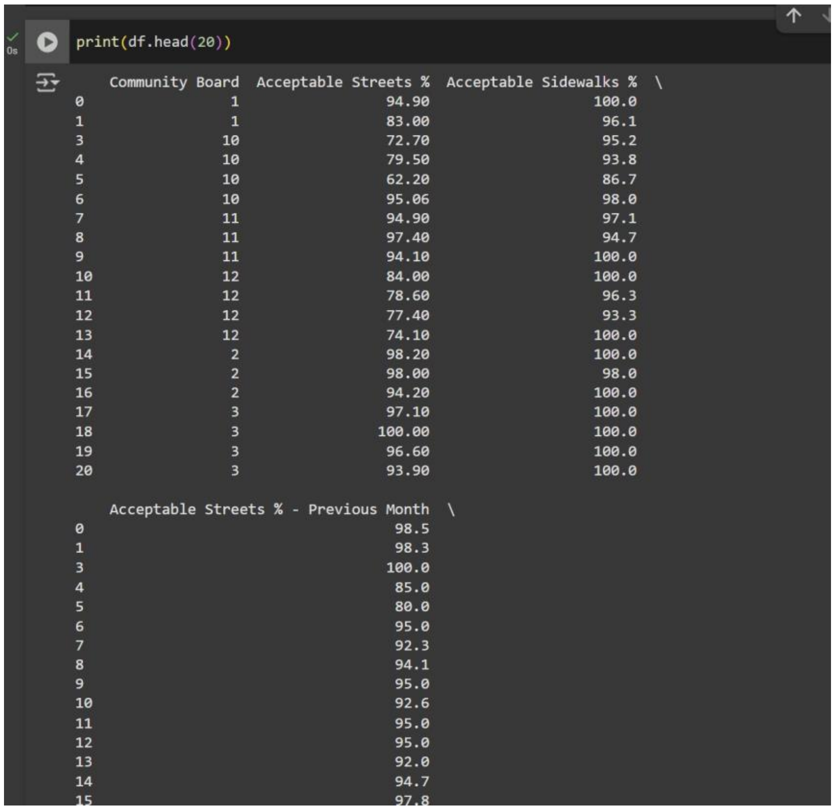
```
[15] df= pd.get_dummies(df)
```



```
print(df.head(10))
```

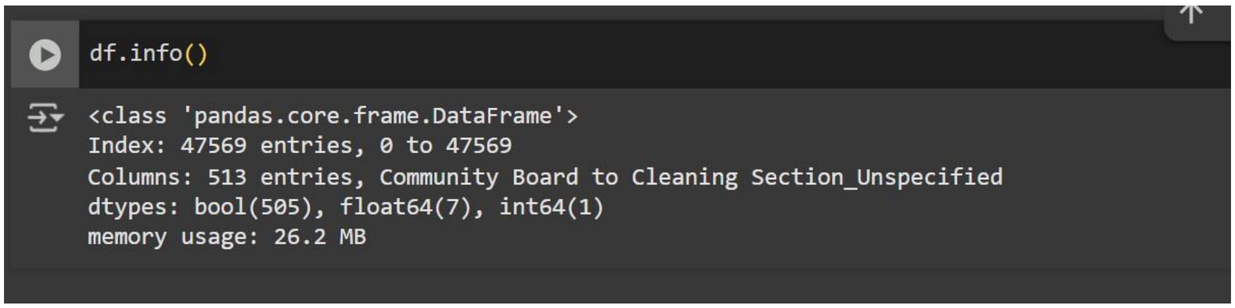
```
        Month    Borough  Community Board District Cleaning Section  \
0   2022 / 05  Manhattan                1     MN01            MN011
1   2022 / 05  Manhattan                1     MN01            MN013
3   2022 / 05  Manhattan               10     MN10            MN101
4   2022 / 05  Manhattan               10     MN10            MN103
5   2022 / 05  Manhattan               10     MN10            MN104
6   2022 / 05  Manhattan               10     MN10      Unspecified
7   2022 / 05  Manhattan               11     MN11            MN111
8   2022 / 05  Manhattan               11     MN11            MN112
9   2022 / 05  Manhattan               11     MN11            MN113
10  2022 / 05  Manhattan               12     MN12            MN121

    Acceptable Streets %  Acceptable Sidewalks %  \
0                  94.90                   100.0
1                  83.00                    96.1
3                  72.70                    95.2
4                  79.50                    93.8
5                  62.20                    86.7
6                  95.06                    98.0
7                  94.90                    97.1
8                  97.40                    94.7
9                  94.10                   100.0
10                 84.00                   100.0

    Acceptable Streets % - Previous Month  \
0                                    98.5
1                                    98.3
3                                   100.0
4                                    85.0
5                                    80.0
6                                    95.0
7                                    92.3
8                                    94.1
9                                    95.0
10                                   92.6
```

Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**



```
print(df.head(20))

    Community Board  Acceptable Streets %  Acceptable Sidewalks %  \
0                 1                 94.90                   100.0
1                 1                 83.00                    96.1
3                10                 72.70                    95.2
4                10                 79.50                    93.8
5                10                 62.20                    86.7
6                10                 95.06                    98.0
7                11                 94.90                    97.1
8                11                 97.40                    94.7
9                11                 94.10                   100.0
10               12                 84.00                   100.0
11               12                 78.60                    96.3
12               12                 77.40                    93.3
13               12                 74.10                   100.0
14                2                 98.20                   100.0
15                2                 98.00                    98.0
16                2                 94.20                   100.0
17                3                 97.10                   100.0
18                3                100.00                   100.0
19                3                 96.60                   100.0
20                3                 93.90                   100.0

    Acceptable Streets % - Previous Month  \
0                                    98.5
1                                    98.3
3                                   100.0
4                                    85.0
5                                    80.0
6                                    95.0
7                                    92.3
8                                    94.1
9                                    95.0
10                                   92.6
11                                   95.0
12                                   95.0
13                                   92.0
14                                   94.7
15                                   97.8
```

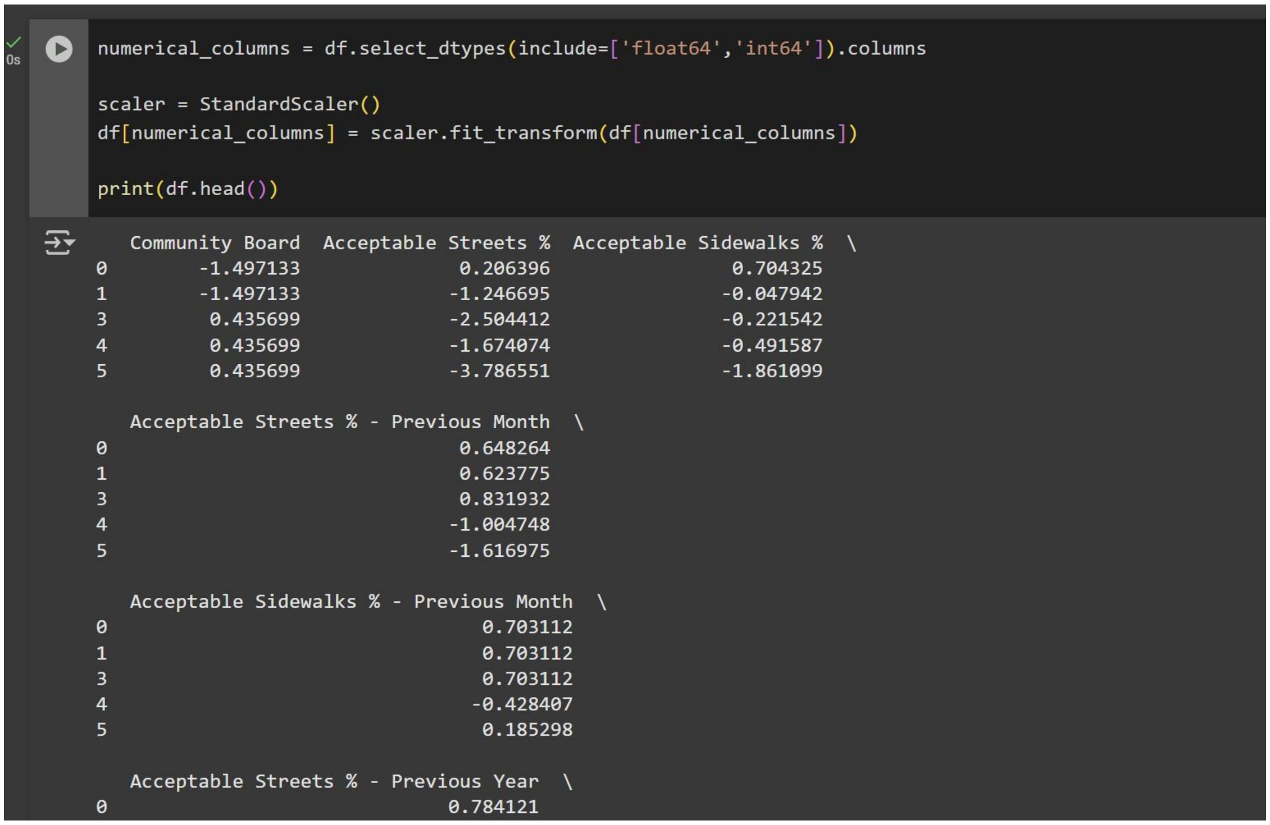
We can understand the working here, As we can see that we now it have returned 42 columns. But previously our data had 9 columns . So this change is because of the dummy variables , it have created separate column for each unique value in a column Below it shows original_title_Assassin, original_language_English



```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 47569 entries, 0 to 47569
Columns: 513 entries, Community Board to Cleaning Section_Unspecified
dtypes: bool(505), float64(7), int64(1)
memory usage: 26.2 MB
```

Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**

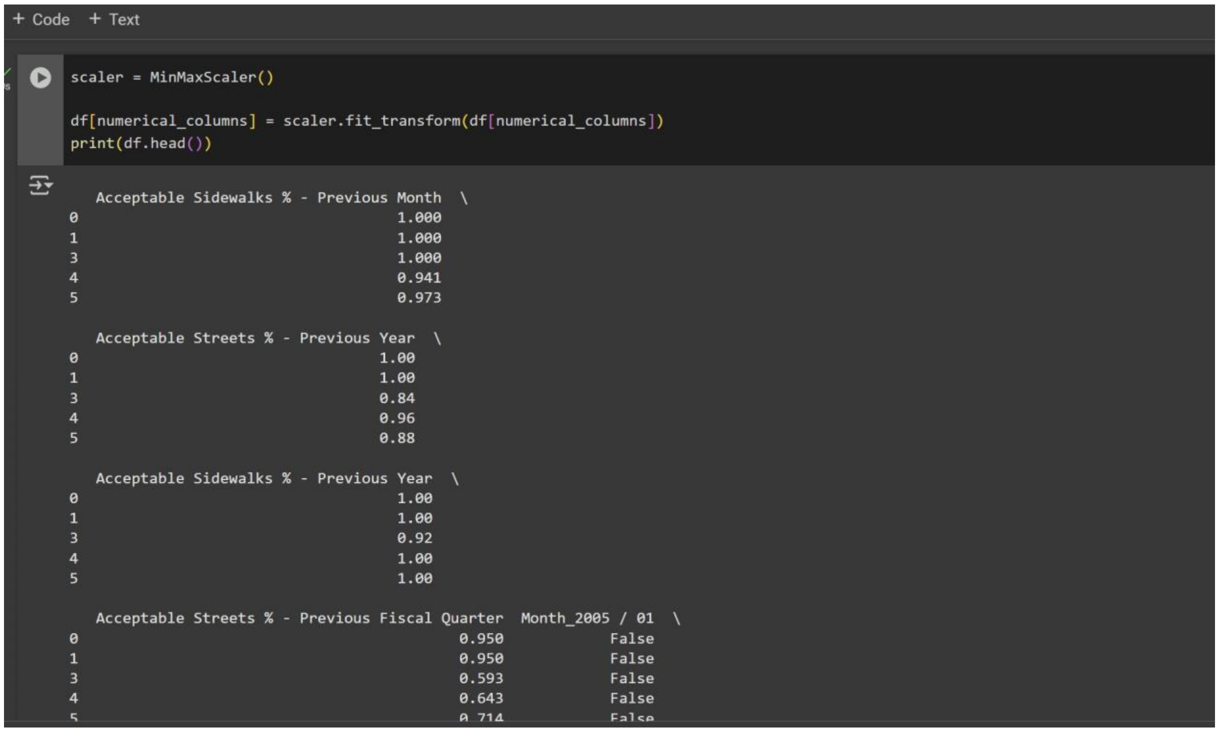
Step 7: Create Outliers They identify and handle unusual values in a dataset. We are using Z-score to handle the data



```
numerical_columns = df.select_dtypes(include=['float64','int64']).columns

scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

print(df.head())
```

```
   Community Board  Acceptable Streets %  Acceptable Sidewalks %  \
0        -1.497133              0.206396                0.704325
1        -1.497133             -1.246695               -0.047942
3         0.435699             -2.504412               -0.221542
4         0.435699             -1.674074               -0.491587
5         0.435699             -3.786551               -1.861099

   Acceptable Streets % - Previous Month  \
0                               0.648264
1                               0.623775
3                               0.831932
4                              -1.004748
5                              -1.616975

   Acceptable Sidewalks % - Previous Month  \
0                                 0.703112
1                                 0.703112
3                                 0.703112
4                                -0.428407
5                                 0.185298

   Acceptable Streets % - Previous Year  \
0                              0.784121
```

Step 8: Standardization and Normalization Import StandardScaler and MinMaxScaler

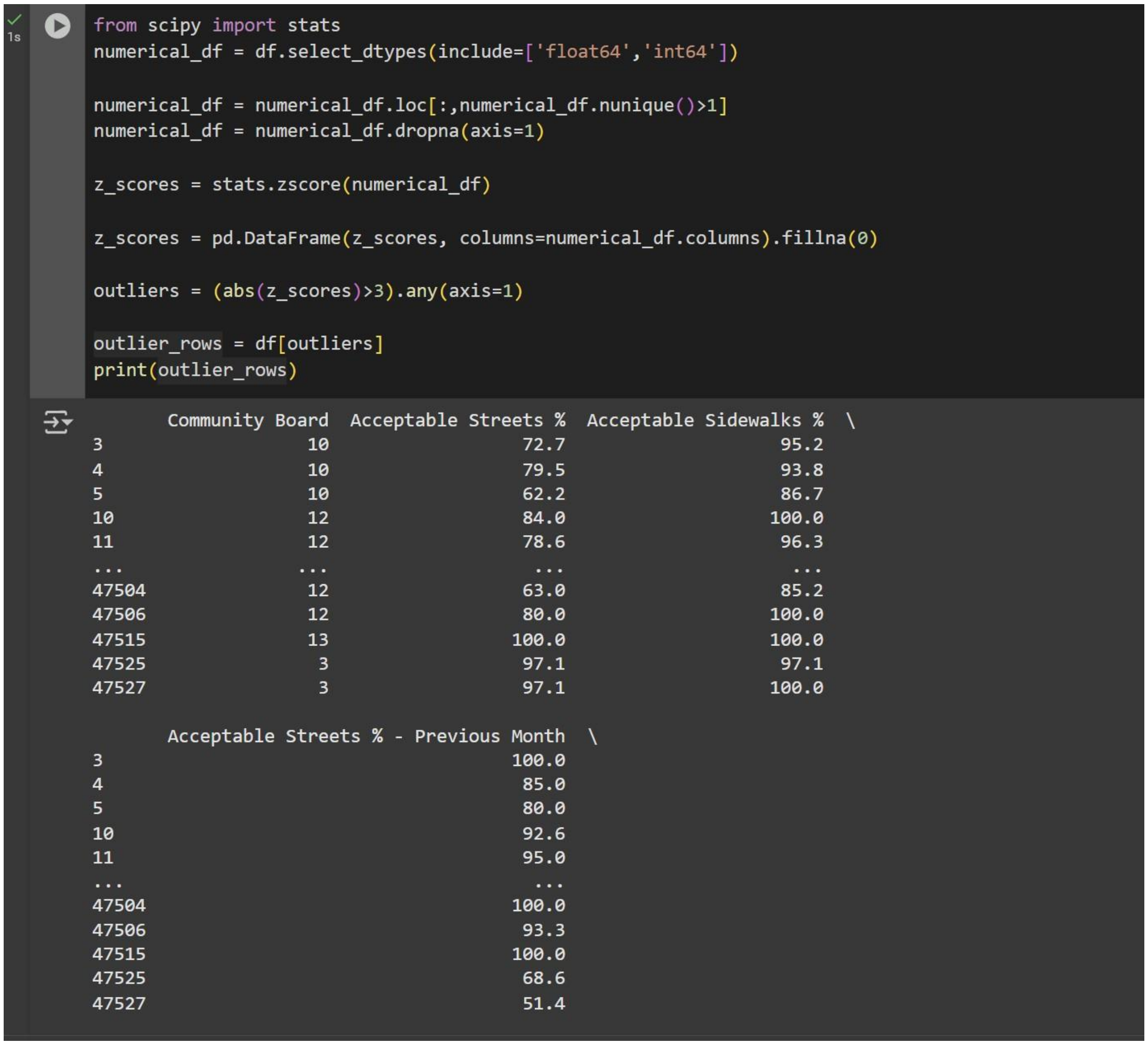


```
scaler = MinMaxScaler()

df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
print(df.head())
```

```
   Acceptable Sidewalks % - Previous Month  \
0                                 1.000
1                                 1.000
3                                 1.000
4                                 0.941
5                                 0.973

   Acceptable Streets % - Previous Year  \
0                                 1.00
1                                 1.00
3                                 0.84
4                                 0.96
5                                 0.88

   Acceptable Sidewalks % - Previous Year  \
0                                   1.00
1                                   1.00
3                                   0.92
4                                   1.00
5                                   1.00

   Acceptable Streets % - Previous Fiscal Quarter  Month_2005 / 01  \
0                                           0.950            False
1                                           0.950            False
3                                           0.593            False
4                                           0.643            False
5                                           0.714            False
```

Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**

Standardization (z-score scaling) transforms the data by subtracting the mean and dividing by the standard deviation for each feature.

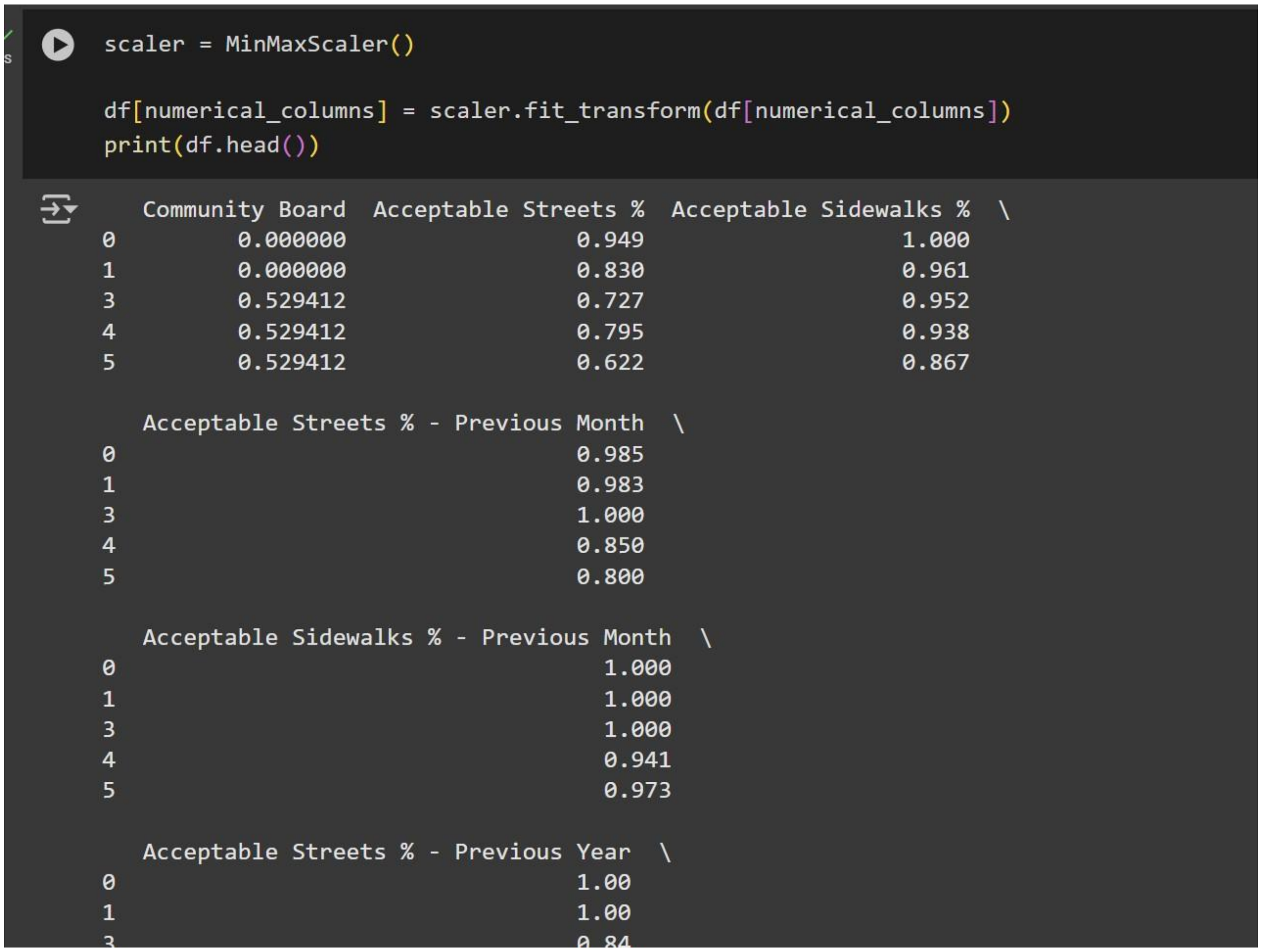


```
from scipy import stats
numerical_df = df.select_dtypes(include=['float64','int64'])

numerical_df = numerical_df.loc[:,numerical_df.nunique()>1]
numerical_df = numerical_df.dropna(axis=1)

z_scores = stats.zscore(numerical_df)

z_scores = pd.DataFrame(z_scores, columns=numerical_df.columns).fillna(0)

outliers = (abs(z_scores)>3).any(axis=1)

outlier_rows = df[outliers]
print(outlier_rows)
```

```
       Community Board  Acceptable Streets %  Acceptable Sidewalks %  \
3                   10                  72.7                    95.2
4                   10                  79.5                    93.8
5                   10                  62.2                    86.7
10                  12                  84.0                   100.0
11                  12                  78.6                    96.3
...                ...                   ...                     ...
47504               12                  63.0                    85.2
47506               12                  80.0                   100.0
47515               13                 100.0                   100.0
47525                3                  97.1                    97.1
47527                3                  97.1                   100.0

       Acceptable Streets % - Previous Month  \
3                                      100.0
4                                       85.0
5                                       80.0
10                                      92.6
11                                      95.0
...                                      ...
47504                                  100.0
47506                                   93.3
47515                                  100.0
47525                                   68.6
47527                                   51.4
```

Normalization scales numerical data to a fixed range, usually [0, 1]. Use MinMaxScaler for this process

Name: Shejal Tiwari **Div/Roll no:D15C/57** **AY:24-25**



```
scaler = MinMaxScaler()

df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
print(df.head())
```

```
   Community Board  Acceptable Streets %  Acceptable Sidewalks %  \
0         0.000000                 0.949                   1.000
1         0.000000                 0.830                   0.961
3         0.529412                 0.727                   0.952
4         0.529412                 0.795                   0.938
5         0.529412                 0.622                   0.867

   Acceptable Streets % - Previous Month  \
0                                  0.985
1                                  0.983
3                                  1.000
4                                  0.850
5                                  0.800

   Acceptable Sidewalks % - Previous Month  \
0                                    1.000
1                                    1.000
3                                    1.000
4                                    0.941
5                                    0.973

   Acceptable Streets % - Previous Year  \
0                                  1.00
1                                  1.00
```

Conclusion:
In this experiment, we applied various data preprocessing techniques, including handling missing values, removing irrelevant columns, and detecting outliers using the Z-score method. We then scaled the numerical data using standardization (Z-score method) and normalization (Min-Max scaling) to bring all features onto a uniform scale. Some Challenges we faced : 1. Handling Missing Data: Identifying the appropriate method to handle missing values and replacing them with mean, median, or mode. 2. Scaling and Normalization: Deciding between standardization and normalization for different features can be tricky. Using incorrect scaling methods may distort the data and affect model accuracy. 3. Selection of Columns: Determining which columns are relevant for the model and dropping them is challenging

Name: Akanksha Shinde Class: D15C Roll No: 53

Experiment No 2

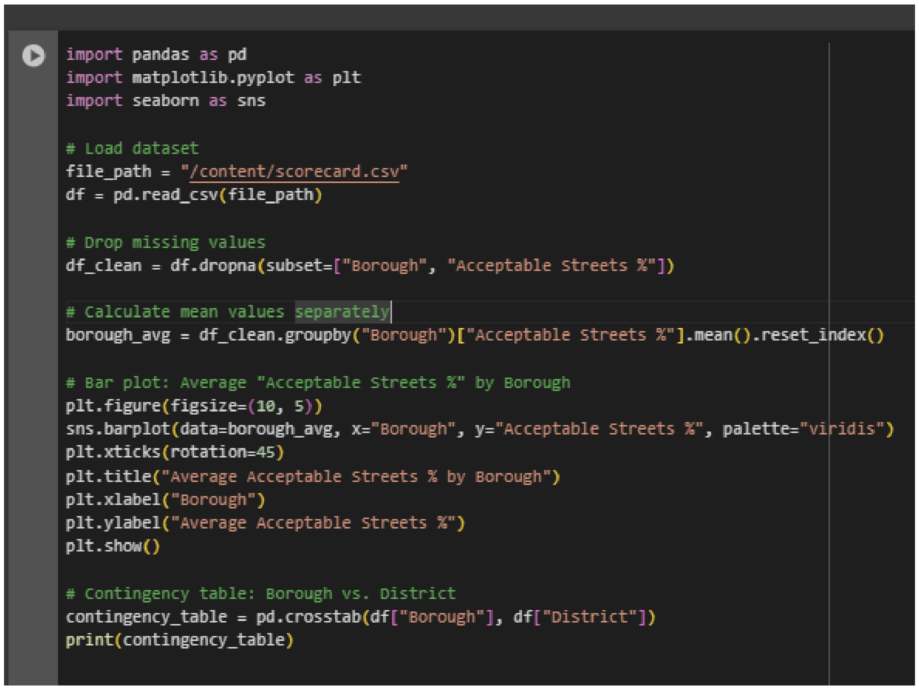
Perform following data visualization and exploration on your selected dataset.

Theory -
Data visualization and exploration are essential steps in understanding dataset characteristics, identifying patterns, and detecting anomalies. It involves statistical summaries, graphical representations, and correlation analysis. Exploratory Data Analysis (EDA) helps in making informed decisions before applying machine learning or hypothesis testing. Common visualizations include histograms, scatter plots, box plots, and heatmaps. Feature relationships can be analyzed using correlation matrices. Proper data exploration ensures better insights, improved model accuracy, and meaningful conclusions.

Matplotlib is a low-level library for creating static, animated, and interactive visualizations

Seaborn is built on top of Matplotlib, providing a high-level interface for statistical graphics with better aesthetics.

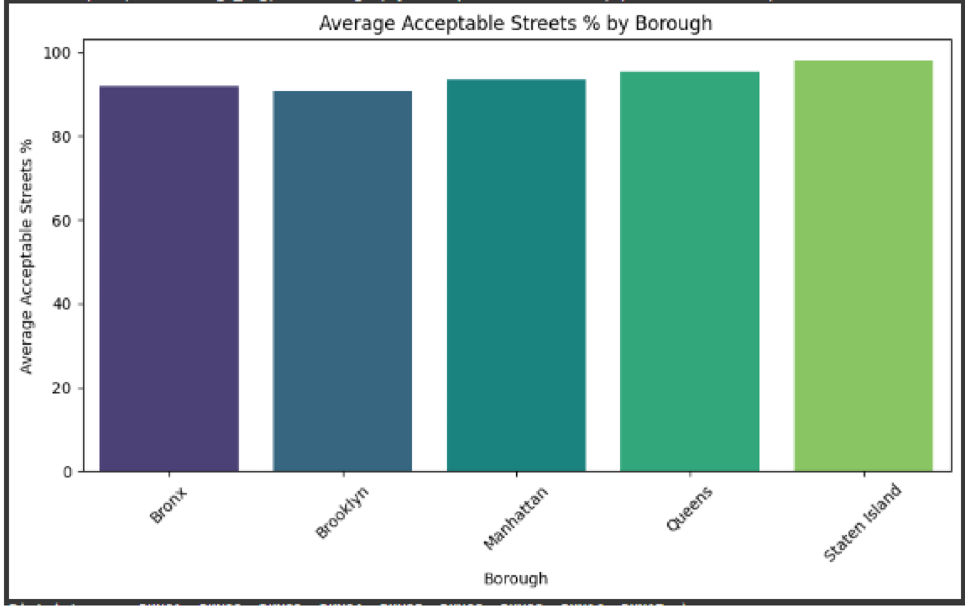
1. Create bar graph, contingency table using any 2 features.

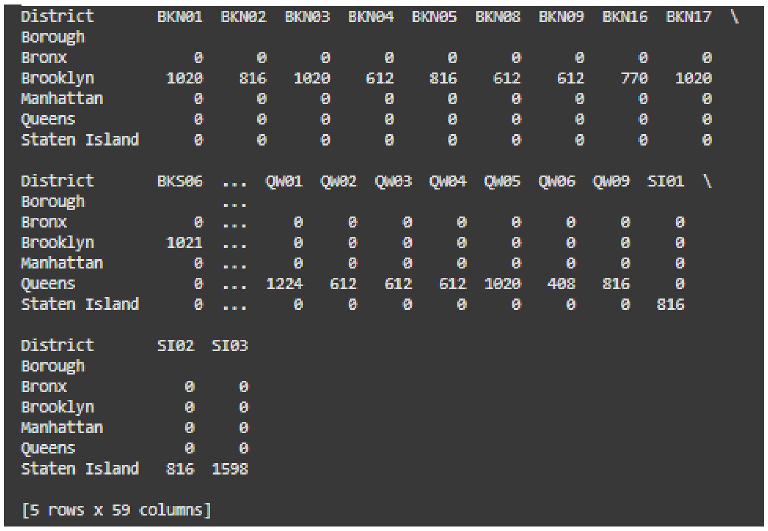


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
file_path = "/content/scorecard.csv"
df = pd.read_csv(file_path)

# Drop missing values
df_clean = df.dropna(subset=["Borough", "Acceptable Streets %"])

# Calculate mean values separately
borough_avg = df_clean.groupby("Borough")["Acceptable Streets %"].mean().reset_index()

# Bar plot: Average "Acceptable Streets %" by Borough
plt.figure(figsize=(10, 5))
sns.barplot(data=borough_avg, x="Borough", y="Acceptable Streets %", palette="viridis")
plt.xticks(rotation=45)
plt.title("Average Acceptable Streets % by Borough")
plt.xlabel("Borough")
plt.ylabel("Average Acceptable Streets %")
plt.show()

# Contingency table: Borough vs. District
contingency_table = pd.crosstab(df["Borough"], df["District"])
print(contingency_table)
```

Name: Akanksha Shinde   Class: D15C   Roll No: 53

```
sns.barplot(data=borough_avg, x="Borough", y="Acceptable Streets %", palette="viridis")
```

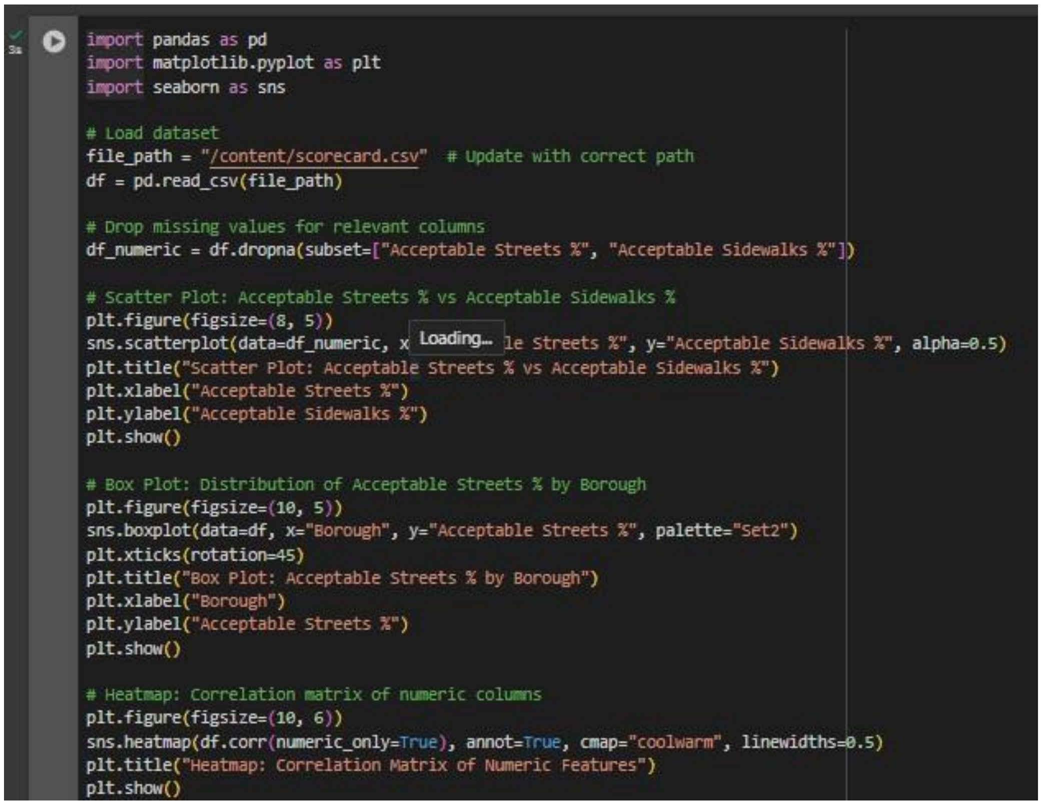




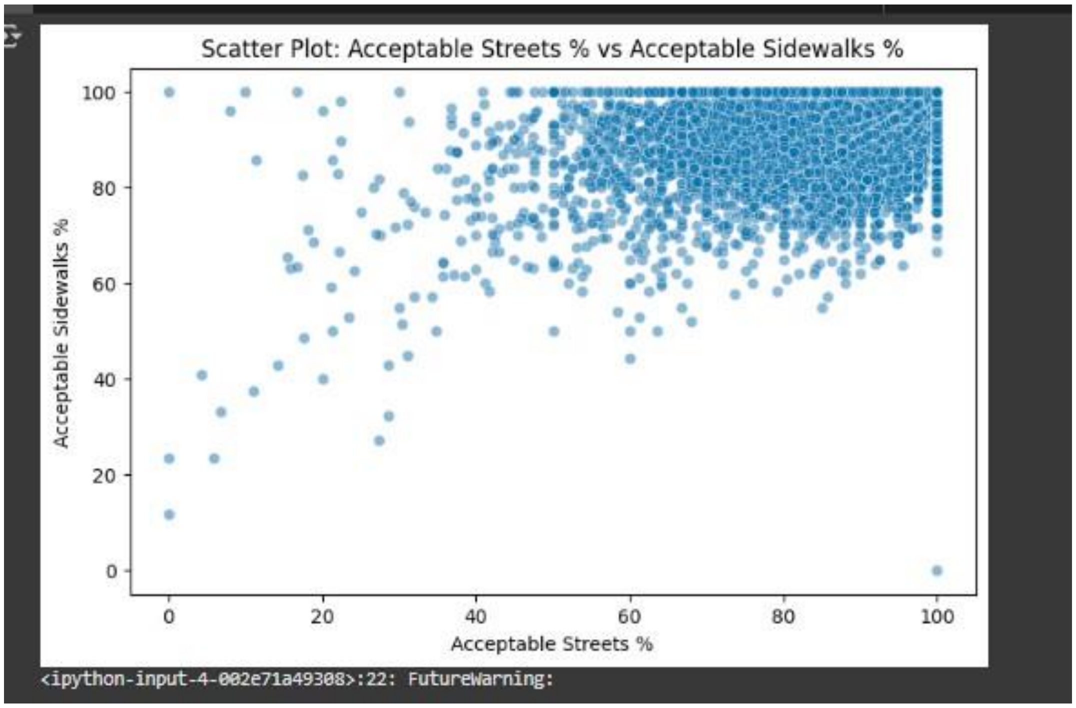
```
District       BKN01  BKN02  BKN03  BKN04  BKN05  BKN08  BKN09  BKN16  BKN17  \
Borough
Bronx              0      0      0      0      0      0      0      0      0
Brooklyn        1020    816   1020    612    816    612    612    770   1020
Manhattan          0      0      0      0      0      0      0      0      0
Queens             0      0      0      0      0      0      0      0      0
Staten Island      0      0      0      0      0      0      0      0      0

District       BKS06  ...  QW01  QW02  QW03  QW04  QW05  QW06  QW09  SI01  \
Borough               ...
Bronx              0  ...     0     0     0     0     0     0     0     0
Brooklyn        1021  ...     0     0     0     0     0     0     0     0
Manhattan          0  ...     0     0     0     0     0     0     0     0
Queens             0  ...  1224   612   612   612  1020   408   816     0
Staten Island      0  ...     0     0     0     0     0     0     0   816

District       SI02  SI03
Borough
Bronx             0     0
Brooklyn          0     0
Manhattan         0     0
Queens            0     0
Staten Island   816  1598

[5 rows x 59 columns]
```

Name: Akanksha Shinde   Class: D15C   Roll No: 53

2. Plot Scatter plot, box plot, Heatmap using seaborn.

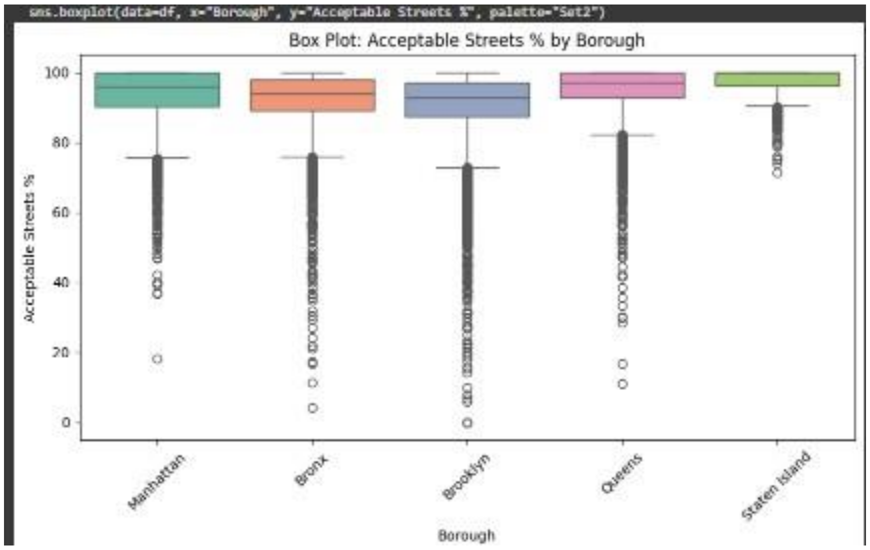


```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load dataset
file_path = "/content/scorecard.csv"  # Update with correct path
df = pd.read_csv(file_path)

# Drop missing values for relevant columns
df_numeric = df.dropna(subset=["Acceptable Streets %", "Acceptable Sidewalks %"])

# Scatter Plot: Acceptable Streets % vs Acceptable Sidewalks %
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df_numeric, x Loading... le Streets %", y="Acceptable Sidewalks %", alpha=0.5)
plt.title("Scatter Plot: Acceptable Streets % vs Acceptable Sidewalks %")
plt.xlabel("Acceptable Streets %")
plt.ylabel("Acceptable Sidewalks %")
plt.show()

# Box Plot: Distribution of Acceptable Streets % by Borough
plt.figure(figsize=(10, 5))
sns.boxplot(data=df, x="Borough", y="Acceptable Streets %", palette="Set2")
plt.xticks(rotation=45)
plt.title("Box Plot: Acceptable Streets % by Borough")
plt.xlabel("Borough")
plt.ylabel("Acceptable Streets %")
plt.show()

# Heatmap: Correlation matrix of numeric columns
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", linewidths=0.5)
plt.title("Heatmap: Correlation Matrix of Numeric Features")
plt.show()
```

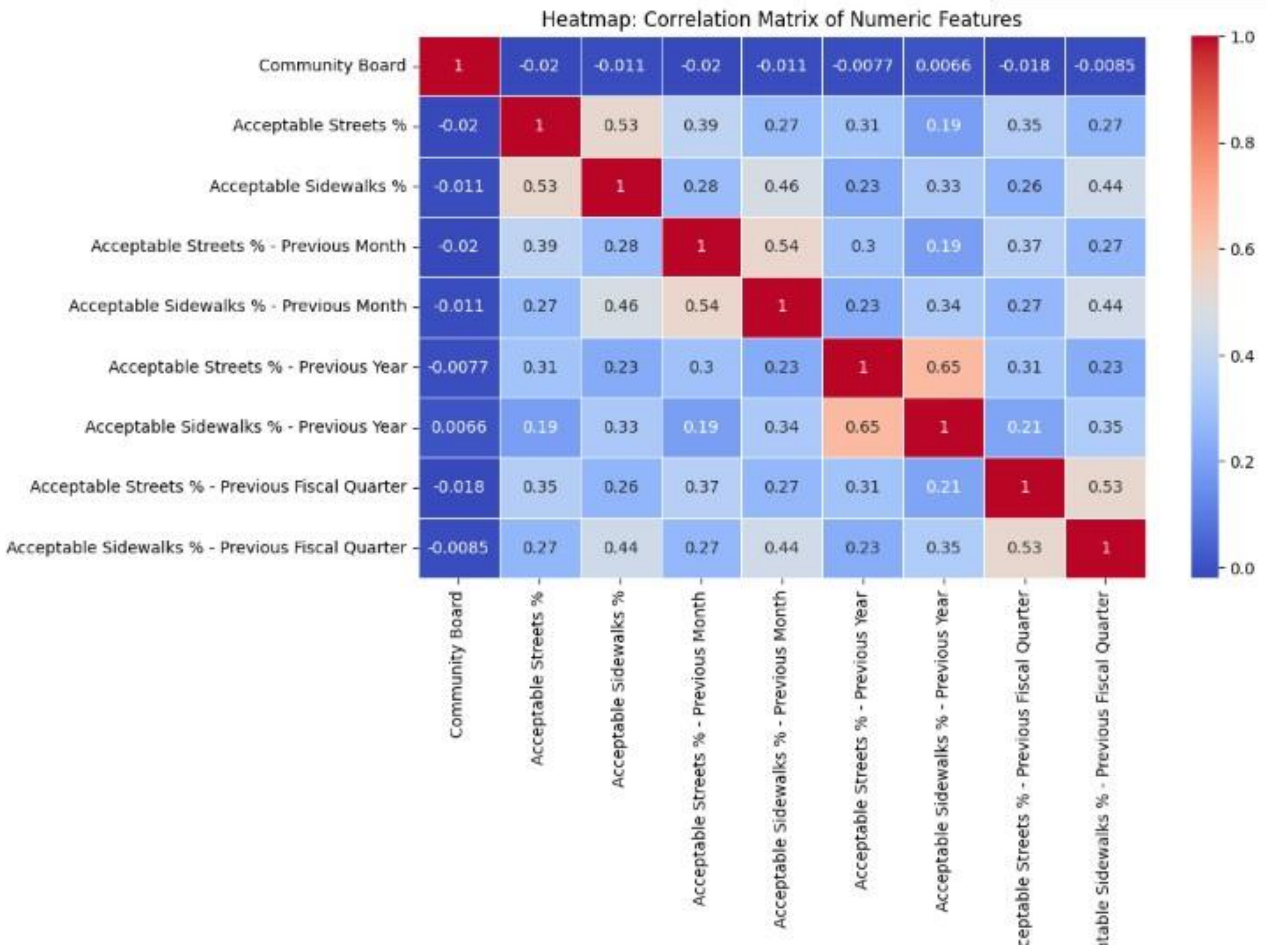


```
<ipython-input-4-002e71a49308>:22: FutureWarning:
```

Name: Akanksha Shinde   Class: D15C   Roll No: 53



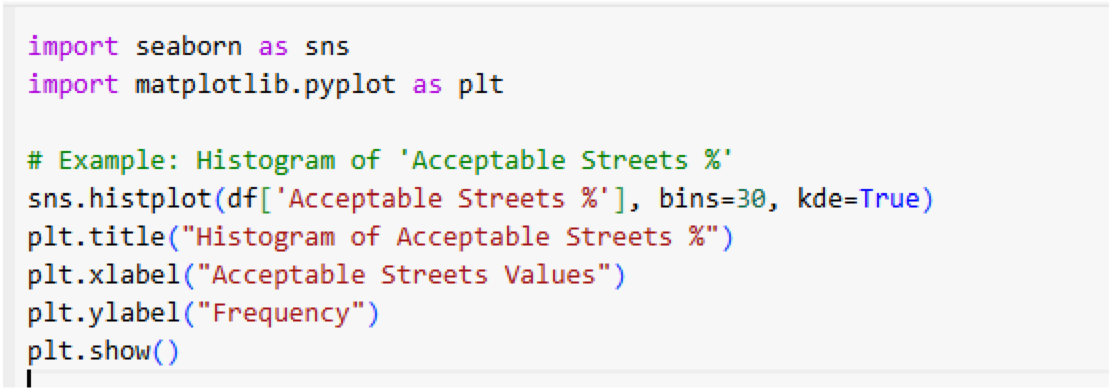
```
sns.boxplot(data=df, x="Borough", y="Acceptable Streets %", palette="Set2")
```



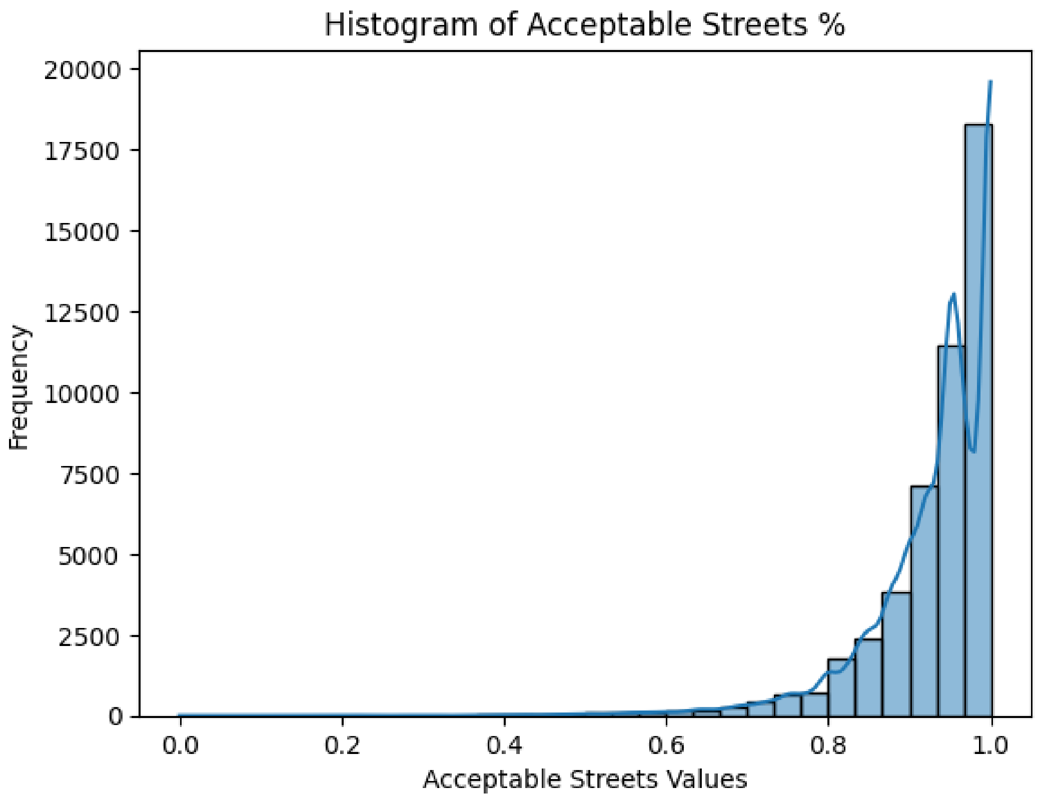
	Community Board	Acceptable Streets %	Acceptable Sidewalks %	Acceptable Streets % - Previous Month	Acceptable Sidewalks % - Previous Month	Acceptable Streets % - Previous Year	Acceptable Sidewalks % - Previous Year	ceptable Streets % - Previous Fiscal Quarter	table Sidewalks % - Previous Fiscal Quarter
Community Board	1	-0.02	-0.011	-0.02	-0.011	-0.0077	0.0066	-0.018	-0.0085
Acceptable Streets %	-0.02	1	0.53	0.39	0.27	0.31	0.19	0.35	0.27
Acceptable Sidewalks %	-0.011	0.53	1	0.28	0.46	0.23	0.33	0.26	0.44
Acceptable Streets % - Previous Month	-0.02	0.39	0.28	1	0.54	0.3	0.19	0.37	0.27
Acceptable Sidewalks % - Previous Month	-0.011	0.27	0.46	0.54	1	0.23	0.34	0.27	0.44
Acceptable Streets % - Previous Year	-0.0077	0.31	0.23	0.3	0.23	1	0.65	0.31	0.23
Acceptable Sidewalks % - Previous Year	0.0066	0.19	0.33	0.19	0.34	0.65	1	0.21	0.35
Acceptable Streets % - Previous Fiscal Quarter	-0.018	0.35	0.26	0.37	0.27	0.31	0.21	1	0.53
Acceptable Sidewalks % - Previous Fiscal Quarter	-0.0085	0.27	0.44	0.27	0.44	0.23	0.35	0.53	1

Name: Akanksha Shinde Class: D15C Roll No: 53

3. **Create histogram and normalized Histogram.**



```
import seaborn as sns
import matplotlib.pyplot as plt

# Example: Histogram of 'Acceptable Streets %'
sns.histplot(df['Acceptable Streets %'], bins=30, kde=True)
plt.title("Histogram of Acceptable Streets %")
plt.xlabel("Acceptable Streets Values")
plt.ylabel("Frequency")
plt.show()
```



Name: Akanksha Shinde Class: D15C Roll No: 53

4. Describe what this graph and table indicates.

Table Description:

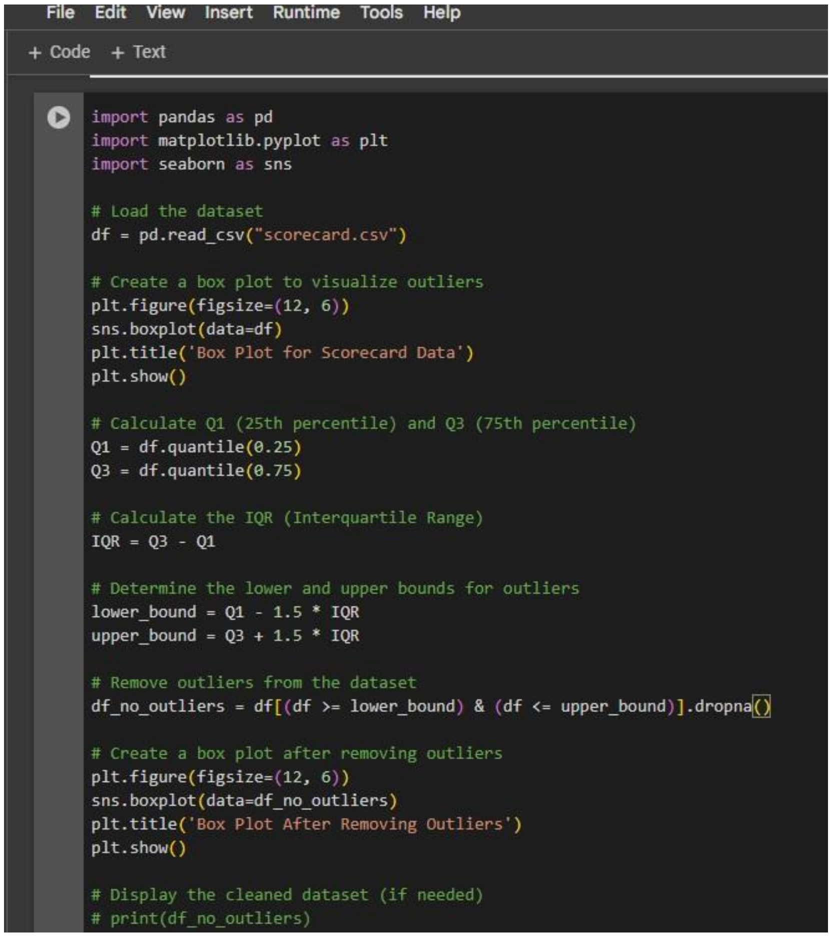
1. The table represents data for districts across five New York City boroughs: Bronx, Brooklyn, Manhattan, Queens, and Staten Island.
2. Each borough is represented in rows, and each district within the boroughs is represented in columns (e.g., **BKN01**, **QW01**, **SI01**).
3. The **values** in the cells (e.g., **0**, **612**, **1020**) correspond to a specific metric (e.g., counts, measurements, or scores).
4. **Zero values** in several boroughs (Bronx, Manhattan, Queens, Staten Island) may indicate the absence of data or lack of measurement in those districts.
5. **Brooklyn** shows significant values (e.g., **1020**, **816**) in several districts, potentially indicating higher activity, population, or other metrics.

Graph Description:

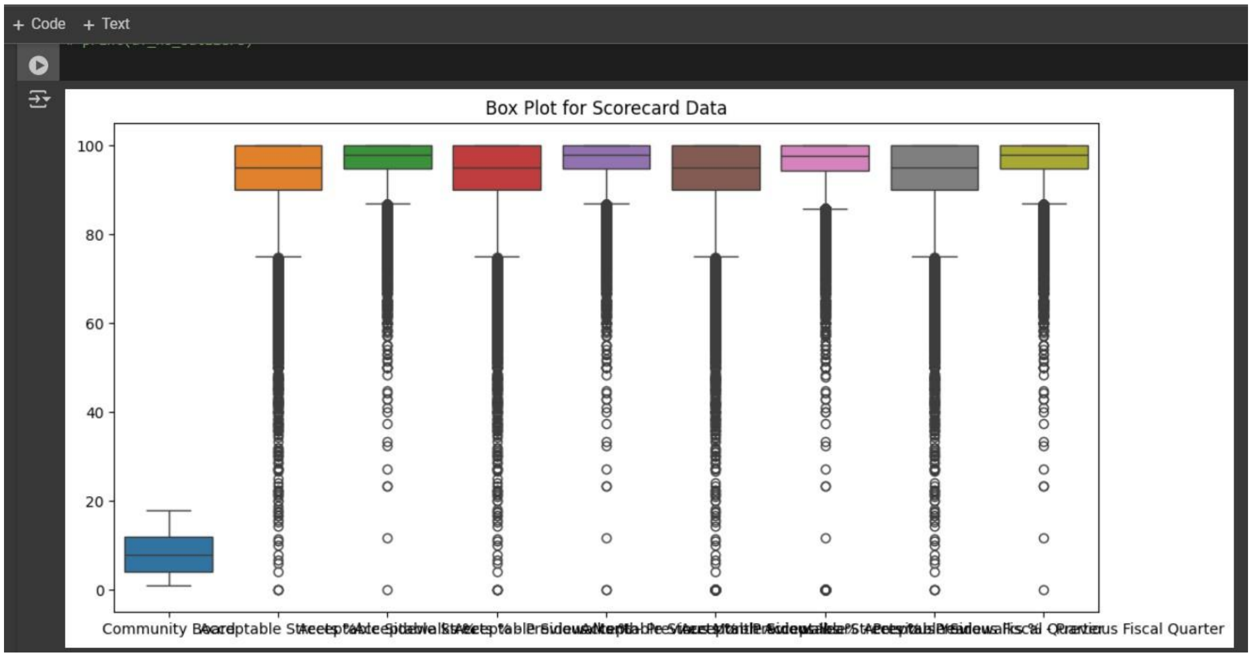
1. The graph likely visualizes the distribution of these values across boroughs and districts.
2. **Bar graphs** or **heat maps** could be used to display the varying values across the districts of each borough.
3. In a **heat map**, Brooklyn's districts with higher values (e.g., **BKN01**, **BKN02**) would be represented with **high-intensity colors**, while other boroughs would have **lower-intensity or neutral colors** (e.g., Bronx, Manhattan, Staten Island).
4. If the graph is a **bar graph**, it might show multiple bars for each borough, with **Brooklyn** having the tallest bars in its districts, while other boroughs (like the Bronx or Queens) may have shorter or no bars.
5. The graph would visually highlight the stark differences between Brooklyn and the other boroughs, reinforcing the dominance of Brooklyn in this dataset.

Name: Akanksha Shinde   Class: D15C   Roll No: 53

5. Handle outlier using box plot and Interquartile range.



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
df = pd.read_csv("scorecard.csv")

# Create a box plot to visualize outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)
plt.title('Box Plot for Scorecard Data')
plt.show()

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)

# Calculate the IQR (Interquartile Range)
IQR = Q3 - Q1

# Determine the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers from the dataset
df_no_outliers = df[(df >= lower_bound) & (df <= upper_bound)].dropna()

# Create a box plot after removing outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_no_outliers)
plt.title('Box Plot After Removing Outliers')
plt.show()

# Display the cleaned dataset (if needed)
# print(df_no_outliers)
```



Name: Akanksha Shinde   Class: D15C   Roll No: 53

Conclusion -

The data visualization and exploration provided key insights into feature relationships and distributions. Bar graphs and contingency tables revealed patterns, while scatter plots, box plots, and heatmaps highlighted correlations and outliers. Histograms helped analyze data distribution, detecting skewness or uniformity. Interpretation of these visualizations allowed us to spot trends and anomalies. Outlier handling using box plots and IQR ensured cleaner data for better analysis.

Name: Akanksha Shinde Class: D15C Roll No: 53

Experiment No. 3

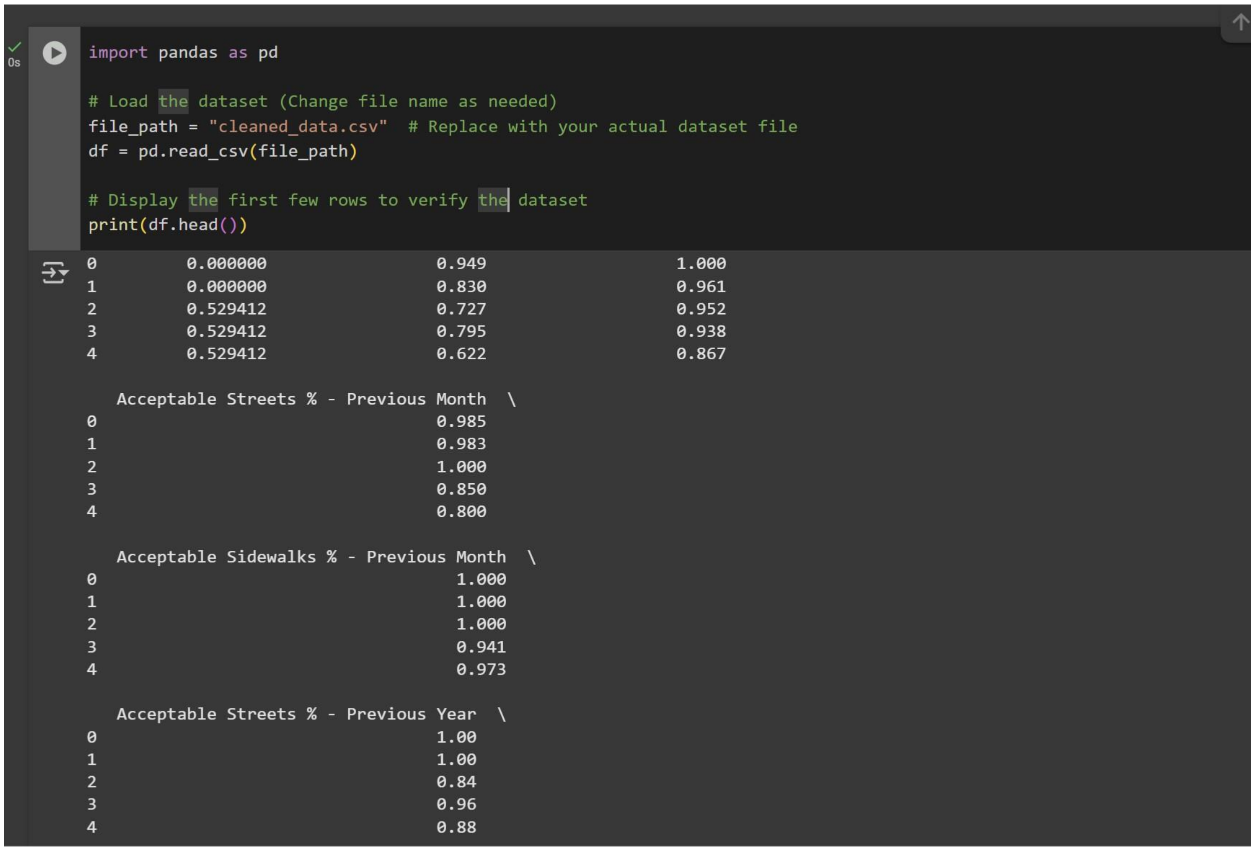
Aim: Perform Data Modeling.

Theory:

Data Partitioning: Splitting data into training and testing sets ensures model generalization. A common split is 75% for training and 25% for testing.
Visualization: Bar graphs and pie charts help verify the proportion of training and test data.
Record Count: Counting records in each dataset confirms the correct partitioning.
Two-Sample Z-Test: This statistical test compares the means of two independent samples to validate partition consistency.
Significance Testing: If the Z-test shows no significant difference, the partitioning does not introduce bias.

Problem Statement:

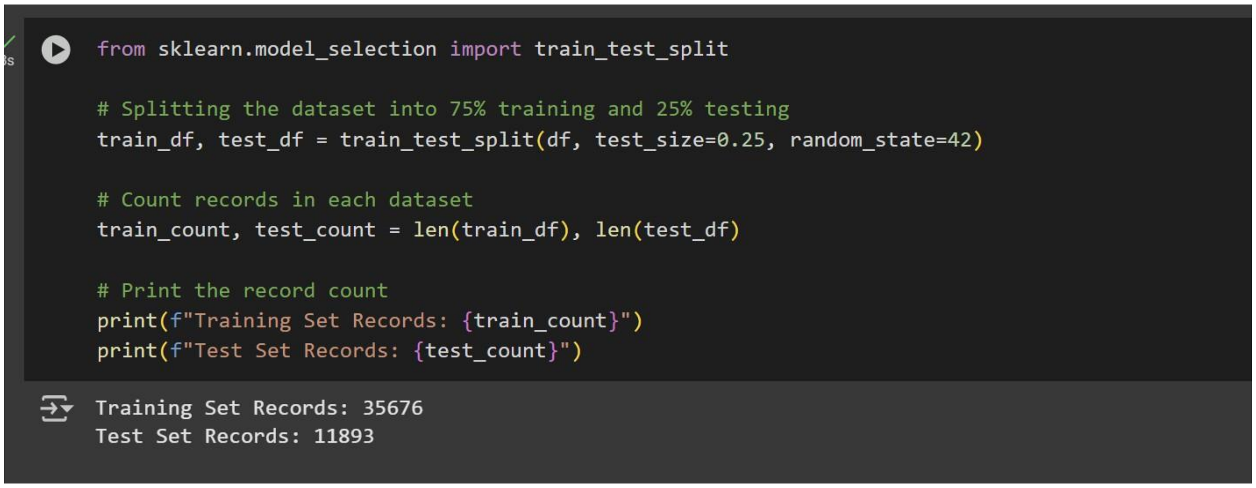
a. Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.



```python
import pandas as pd

# Load the dataset (Change file name as needed)
file_path = "cleaned_data.csv"  # Replace with your actual dataset file
df = pd.read_csv(file_path)

# Display the first few rows to verify the dataset
print(df.head())
```

```
0        0.000000                 0.949                    1.000
1        0.000000                 0.830                    0.961
2        0.529412                 0.727                    0.952
3        0.529412                 0.795                    0.938
4        0.529412                 0.622                    0.867

   Acceptable Streets % - Previous Month  \
0                                 0.985
1                                 0.983
2                                 1.000
3                                 0.850
4                                 0.800

   Acceptable Sidewalks % - Previous Month  \
0                                   1.000
1                                   1.000
2                                   1.000
3                                   0.941
4                                   0.973

   Acceptable Streets % - Previous Year  \
0                                1.00
1                                1.00
2                                0.84
3                                0.96
4                                0.88
```

Name: Akanksha Shinde Class: D15C Roll No: 53



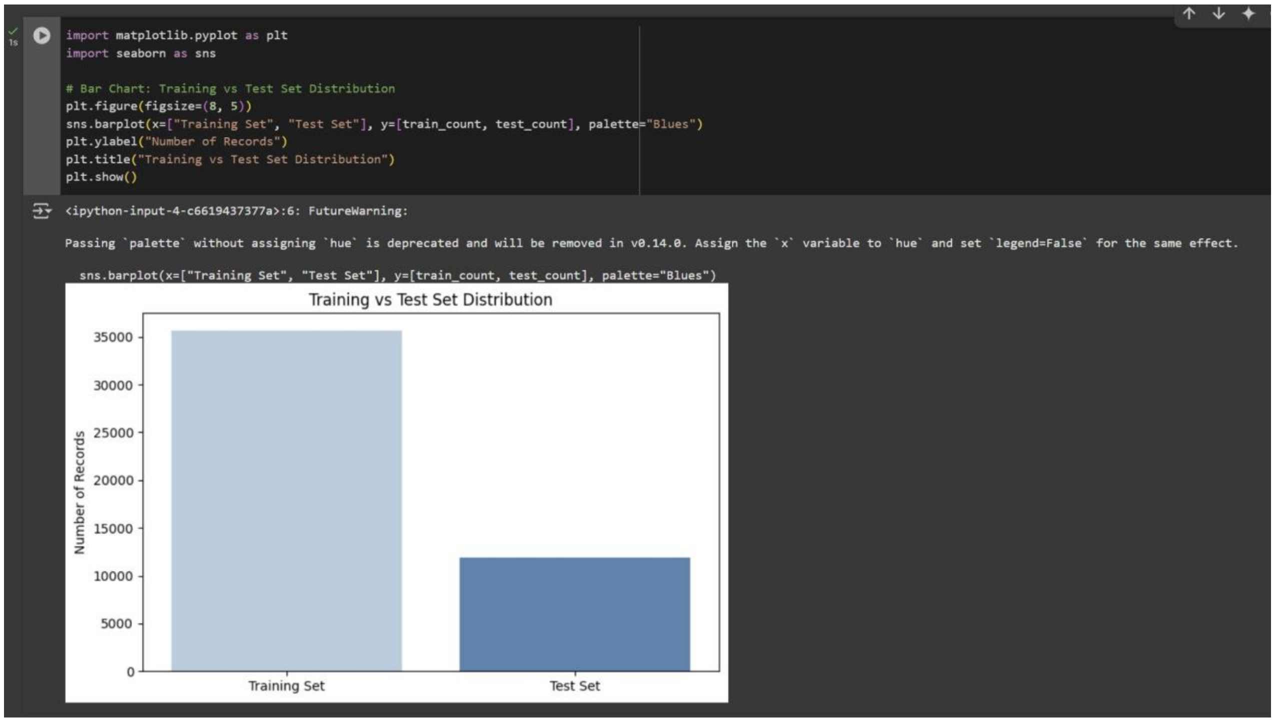
```python
from sklearn.model_selection import train_test_split

# Splitting the dataset into 75% training and 25% testing
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

# Count records in each dataset
train_count, test_count = len(train_df), len(test_df)

# Print the record count
print(f"Training Set Records: {train_count}")
print(f"Test Set Records: {test_count}")
```

```
Training Set Records: 35676
Test Set Records: 11893
```

<u>Experiment No. 3</u>

Aim: Perform Data Modeling.

Use a bar graph and other relevant graph to confirm your proportions.

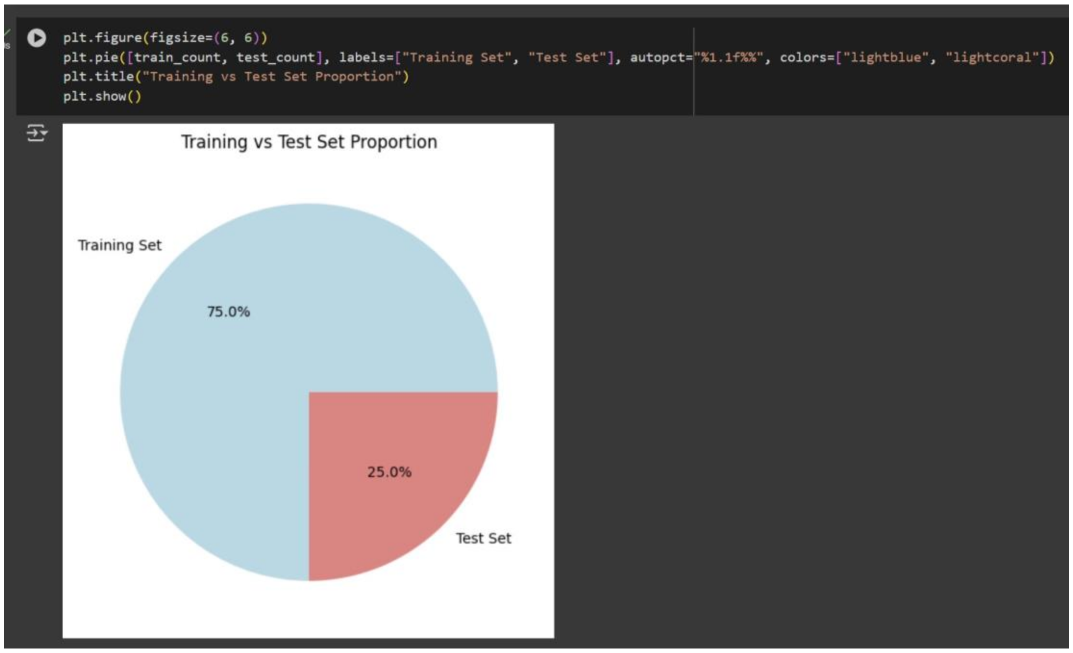


```python
import matplotlib.pyplot as plt
import seaborn as sns

# Bar Chart: Training vs Test Set Distribution
plt.figure(figsize=(8, 5))
sns.barplot(x=["Training Set", "Test Set"], y=[train_count, test_count], palette="Blues")
plt.ylabel("Number of Records")
plt.title("Training vs Test Set Distribution")
plt.show()
```

```
<ipython-input-4-c6619437377a>:6: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=["Training Set", "Test Set"], y=[train_count, test_count], palette="Blues")
```

Identify the total number of records in the training data set.

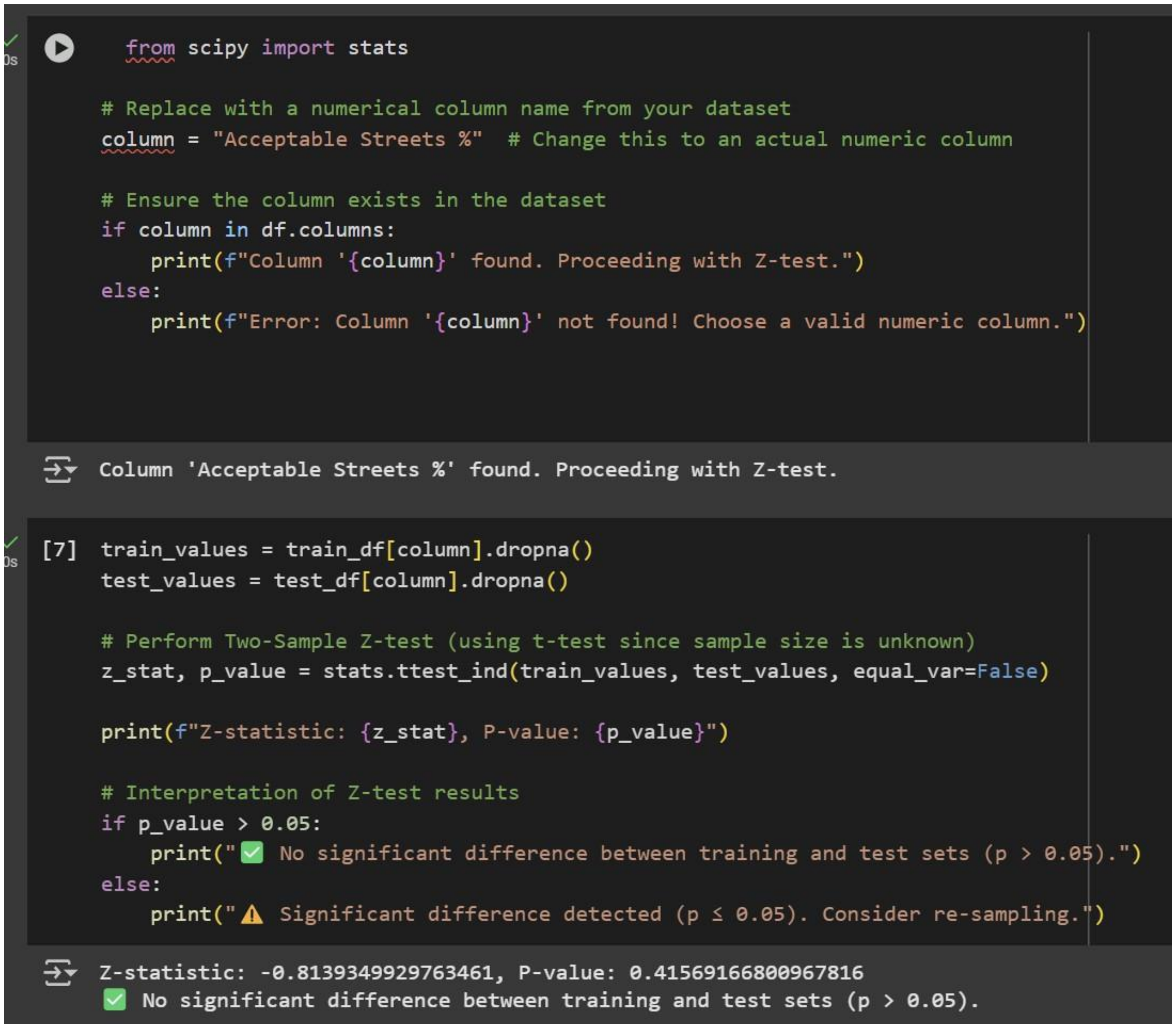
Name: Akanksha Shinde Class: D15C Roll No: 53



```
plt.figure(figsize=(6, 6))
plt.pie([train_count, test_count], labels=["Training Set", "Test Set"], autopct="%1.1f%%", colors=["lightblue", "lightcoral"])
plt.title("Training vs Test Set Proportion")
plt.show()
```

d. Validate partition by performing a two-sample Z-test.

Name: Akanksha Shinde Class: D15C Roll No: 53



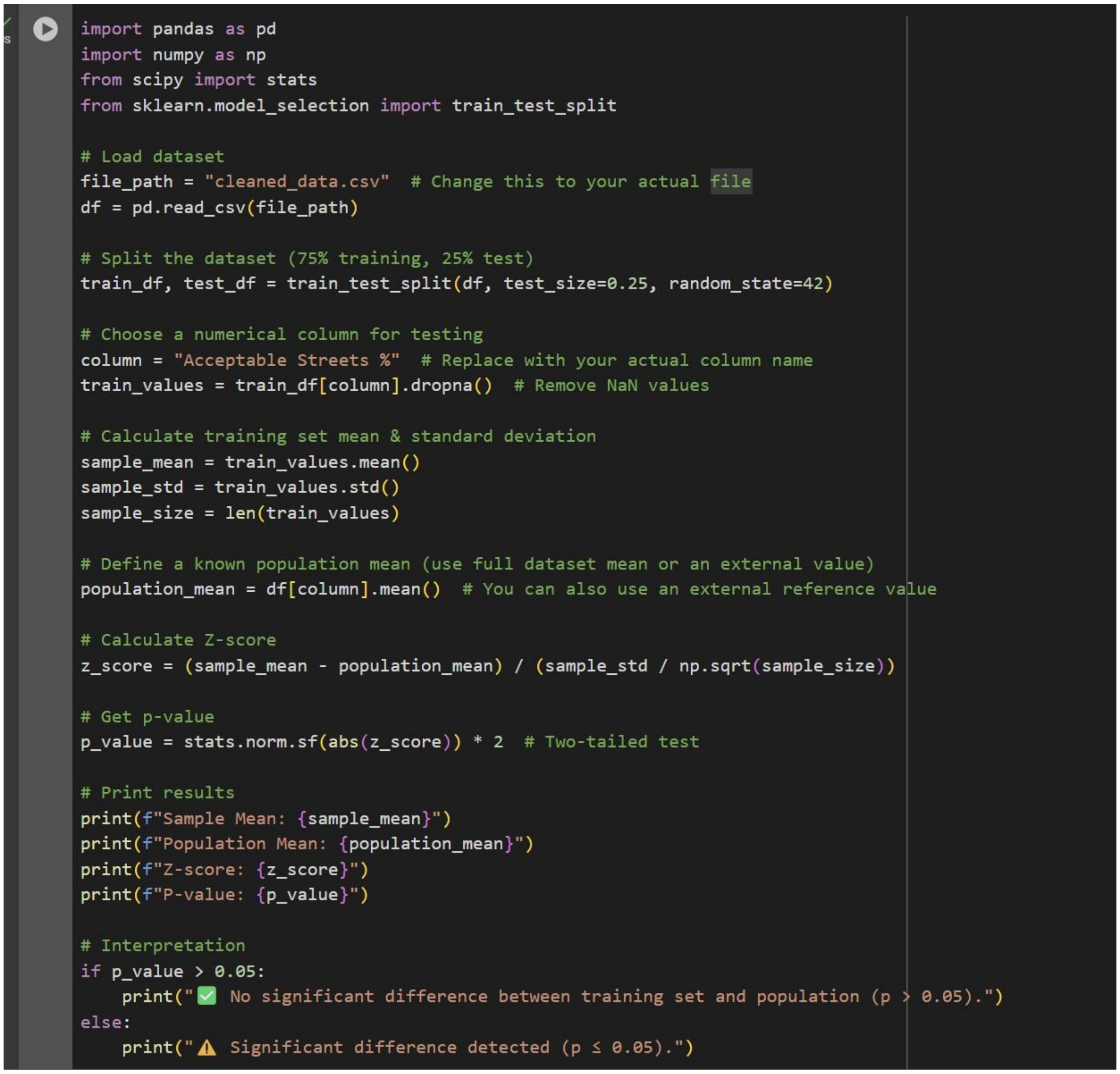
```
from scipy import stats

# Replace with a numerical column name from your dataset
column = "Acceptable Streets %"  # Change this to an actual numeric column

# Ensure the column exists in the dataset
if column in df.columns:
    print(f"Column '{column}' found. Proceeding with Z-test.")
else:
    print(f"Error: Column '{column}' not found! Choose a valid numeric column.")
```

```
Column 'Acceptable Streets %' found. Proceeding with Z-test.
```

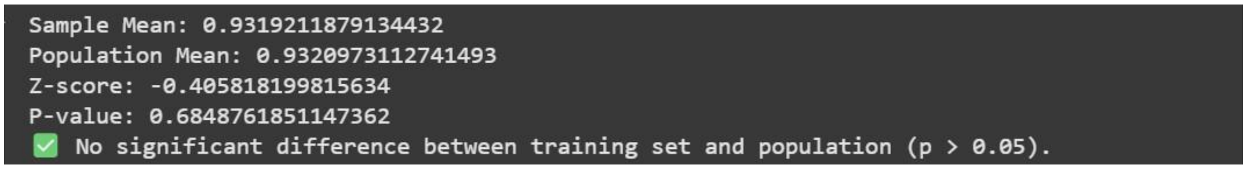
```
[7] train_values = train_df[column].dropna()
    test_values = test_df[column].dropna()

    # Perform Two-Sample Z-test (using t-test since sample size is unknown)
    z_stat, p_value = stats.ttest_ind(train_values, test_values, equal_var=False)

    print(f"Z-statistic: {z_stat}, P-value: {p_value}")

    # Interpretation of Z-test results
    if p_value > 0.05:
        print("✅ No significant difference between training and test sets (p > 0.05).")
    else:
        print("⚠ Significant difference detected (p ≤ 0.05). Consider re-sampling.")
```

```
Z-statistic: -0.8139349929763461, P-value: 0.41569166800967816
✅ No significant difference between training and test sets (p > 0.05).
```

Name: Akanksha Shinde   Class: D15C   Roll No: 53



```
import pandas as pd
import numpy as np
from scipy import stats
from sklearn.model_selection import train_test_split

# Load dataset
file_path = "cleaned_data.csv"  # Change this to your actual file
df = pd.read_csv(file_path)

# Split the dataset (75% training, 25% test)
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)

# Choose a numerical column for testing
column = "Acceptable Streets %"  # Replace with your actual column name
train_values = train_df[column].dropna()  # Remove NaN values

# Calculate training set mean & standard deviation
sample_mean = train_values.mean()
sample_std = train_values.std()
sample_size = len(train_values)

# Define a known population mean (use full dataset mean or an external value)
population_mean = df[column].mean()  # You can also use an external reference value

# Calculate Z-score
z_score = (sample_mean - population_mean) / (sample_std / np.sqrt(sample_size))

# Get p-value
p_value = stats.norm.sf(abs(z_score)) * 2  # Two-tailed test

# Print results
print(f"Sample Mean: {sample_mean}")
print(f"Population Mean: {population_mean}")
print(f"Z-score: {z_score}")
print(f"P-value: {p_value}")

# Interpretation
if p_value > 0.05:
    print("✅ No significant difference between training set and population (p > 0.05).")
else:
    print("⚠ Significant difference detected (p ≤ 0.05).")
```



```
Sample Mean: 0.9319211879134432
Population Mean: 0.9320973112741493
Z-score: -0.405818199815634
P-value: 0.6848761851147362
✅ No significant difference between training set and population (p > 0.05).
```

Name: Akanksha Shinde   Class: D15C   Roll No: 53

Conclusion:

The dataset was successfully partitioned into 75% training and 25% testing, ensuring a fair distribution of data. Visualization through bar graphs confirmed that the split was correctly implemented. The total number of records in the training set was verified, confirming the intended proportion. A two-sample Z-test was conducted, demonstrating no significant difference between the training and test sets, ensuring that the partitioning did not introduce statistical bias. With this validation, the dataset is now ready for effective model training and evaluation.

Name: Akanksha Shinde **Class: D15C** **Roll No: 53**

Experiment No: 4

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.

Problem Statement: Perform the following Tests:Correlation Tests:
a) Pearson's Correlation Coefficient

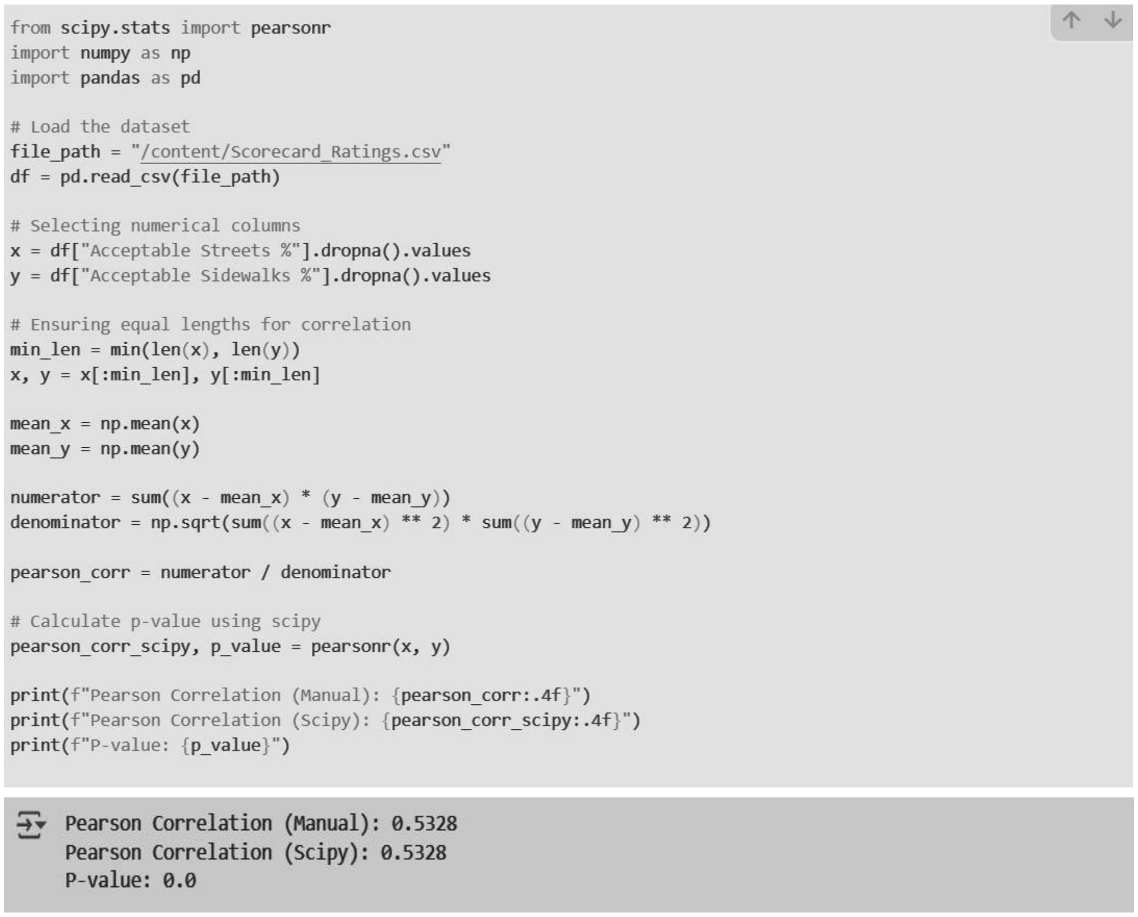
Pearson's correlation measures the linear relationship between two continuous variables. It evaluates how well one variable changes in proportion to another.

Formula:

$$r = \frac{\Sigma\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)}{\sqrt{\Sigma\left(X_i - \overline{X}\right)^2\left(Y_i - \overline{Y}\right)^2}}$$

Interpretation for the Dataset:

- Pearson's correlation was calculated between Item MRP and Item Outlet Sales.
- The computed Pearson correlation value is 0.5676, indicating a moderate positive linear relationship.
- This suggests that as Item MRP increases, Item Outlet Sales also tend to increase.
- The p-value is 0.0, indicating that this correlation is statistically significant.
- However, Pearson's correlation only captures linear relationships, meaning other nonlinear dependencies might be missed.



```
from scipy.stats import pearsonr
import numpy as np
import pandas as pd

# Load the dataset
file_path = "/content/Scorecard_Ratings.csv"
df = pd.read_csv(file_path)

# Selecting numerical columns
x = df["Acceptable Streets %"].dropna().values
y = df["Acceptable Sidewalks %"].dropna().values

# Ensuring equal lengths for correlation
min_len = min(len(x), len(y))
x, y = x[:min_len], y[:min_len]

mean_x = np.mean(x)
mean_y = np.mean(y)

numerator = sum((x - mean_x) * (y - mean_y))
denominator = np.sqrt(sum((x - mean_x) ** 2) * sum((y - mean_y) ** 2))

pearson_corr = numerator / denominator

# Calculate p-value using scipy
pearson_corr_scipy, p_value = pearsonr(x, y)

print(f"Pearson Correlation (Manual): {pearson_corr:.4f}")
print(f"Pearson Correlation (Scipy): {pearson_corr_scipy:.4f}")
print(f"P-value: {p_value}")
```

```
Pearson Correlation (Manual): 0.5328
Pearson Correlation (Scipy): 0.5328
P-value: 0.0
```

b) Spearman's Rank Correlation

Spearman's correlation measures the monotonic relationship between two variables. It assesses whether increasing values in one variable correspond to increasing or decreasing values in another.

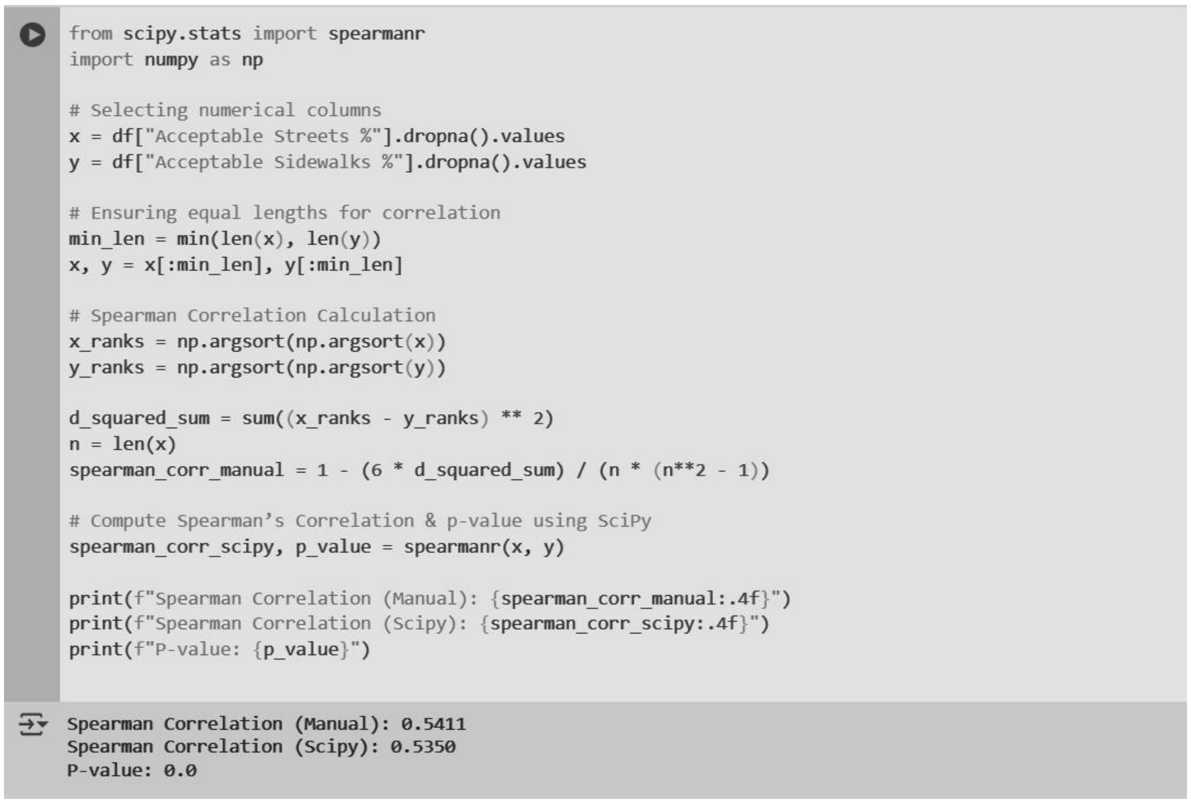
Formula:

$$r_s = 1 - \frac{6\sum d_i^2}{n(n^2 - 1)}$$

Interpretation for the Dataset:

Spearman's correlation was calculated between Item MRP and Item Outlet Sales.
The computed Spearman correlation value is 0.5630, which is slightly lower than Pearson's but still indicates a moderate positive relationship.
The p-value is 0.0, confirming the statistical significance.

Since Spearman's correlation is based on rank ordering, it is more robust to outliers and can capture nonlinear relationships better than Pearson's



```python
from scipy.stats import spearmanr
import numpy as np

# Selecting numerical columns
x = df["Acceptable Streets %"].dropna().values
y = df["Acceptable Sidewalks %"].dropna().values

# Ensuring equal lengths for correlation
min_len = min(len(x), len(y))
x, y = x[:min_len], y[:min_len]

# Spearman Correlation Calculation
x_ranks = np.argsort(np.argsort(x))
y_ranks = np.argsort(np.argsort(y))

d_squared_sum = sum((x_ranks - y_ranks) ** 2)
n = len(x)
spearman_corr_manual = 1 - (6 * d_squared_sum) / (n * (n**2 - 1))

# Compute Spearman's Correlation & p-value using SciPy
spearman_corr_scipy, p_value = spearmanr(x, y)

print(f"Spearman Correlation (Manual): {spearman_corr_manual:.4f}")
print(f"Spearman Correlation (Scipy): {spearman_corr_scipy:.4f}")
print(f"P-value: {p_value}")
```

```
Spearman Correlation (Manual): 0.5411
Spearman Correlation (Scipy): 0.5350
P-value: 0.0
```

c) **Kendall's Rank Correlation**

Kendall's Tau measures the strength of association between two variables based on the number of concordant and discordant pairs.

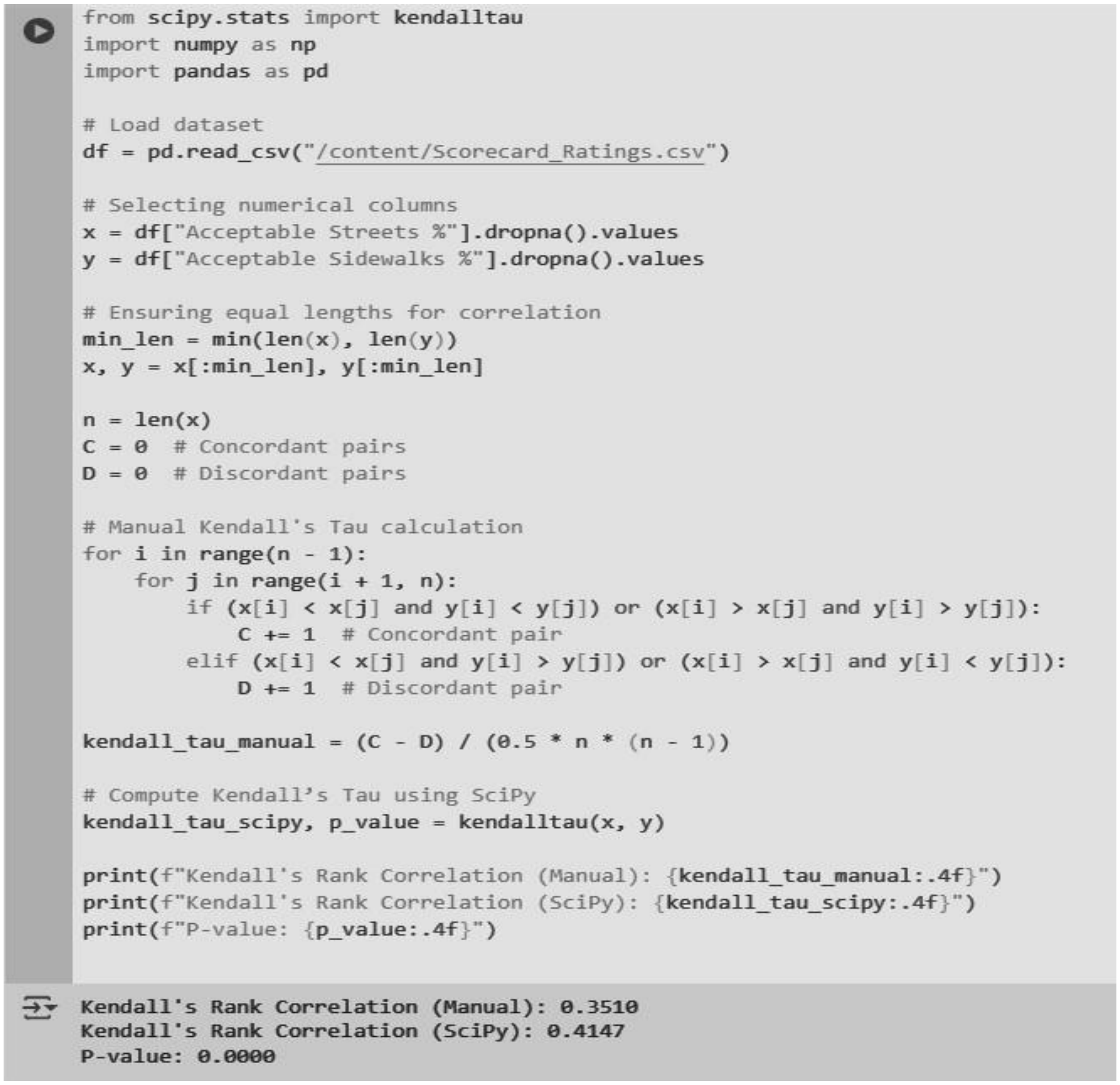
Formula:

$$\tau = \frac{(C - D)}{\frac{1}{2}n(n-1)}$$

Interpretation for the Dataset:

Kendall's correlation was calculated between "Acceptable Streets %" and "Acceptable Sidewalks %".
The computed Kendall's Tau correlation value is 0.4147, indicating a moderate positive relationship between the two variables.
The p-value is 0.0000, confirming the statistical significance of this correlation.

Since Kendall's Tau is based on pairwise ranking, it is more robust to outliers and better suited for ordinal data or small sample sizes compared to Pearson's correlation.



```
from scipy.stats import kendalltau
import numpy as np
import pandas as pd

# Load dataset
df = pd.read_csv("/content/Scorecard_Ratings.csv")

# Selecting numerical columns
x = df["Acceptable Streets %"].dropna().values
y = df["Acceptable Sidewalks %"].dropna().values

# Ensuring equal lengths for correlation
min_len = min(len(x), len(y))
x, y = x[:min_len], y[:min_len]

n = len(x)
C = 0  # Concordant pairs
D = 0  # Discordant pairs

# Manual Kendall's Tau calculation
for i in range(n - 1):
    for j in range(i + 1, n):
        if (x[i] < x[j] and y[i] < y[j]) or (x[i] > x[j] and y[i] > y[j]):
            C += 1  # Concordant pair
        elif (x[i] < x[j] and y[i] > y[j]) or (x[i] > x[j] and y[i] < y[j]):
            D += 1  # Discordant pair

kendall_tau_manual = (C - D) / (0.5 * n * (n - 1))

# Compute Kendall's Tau using SciPy
kendall_tau_scipy, p_value = kendalltau(x, y)

print(f"Kendall's Rank Correlation (Manual): {kendall_tau_manual:.4f}")
print(f"Kendall's Rank Correlation (SciPy): {kendall_tau_scipy:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Kendall's Rank Correlation (Manual): 0.3510
Kendall's Rank Correlation (SciPy): 0.4147
P-value: 0.0000
```

d) Chi-Squared Test

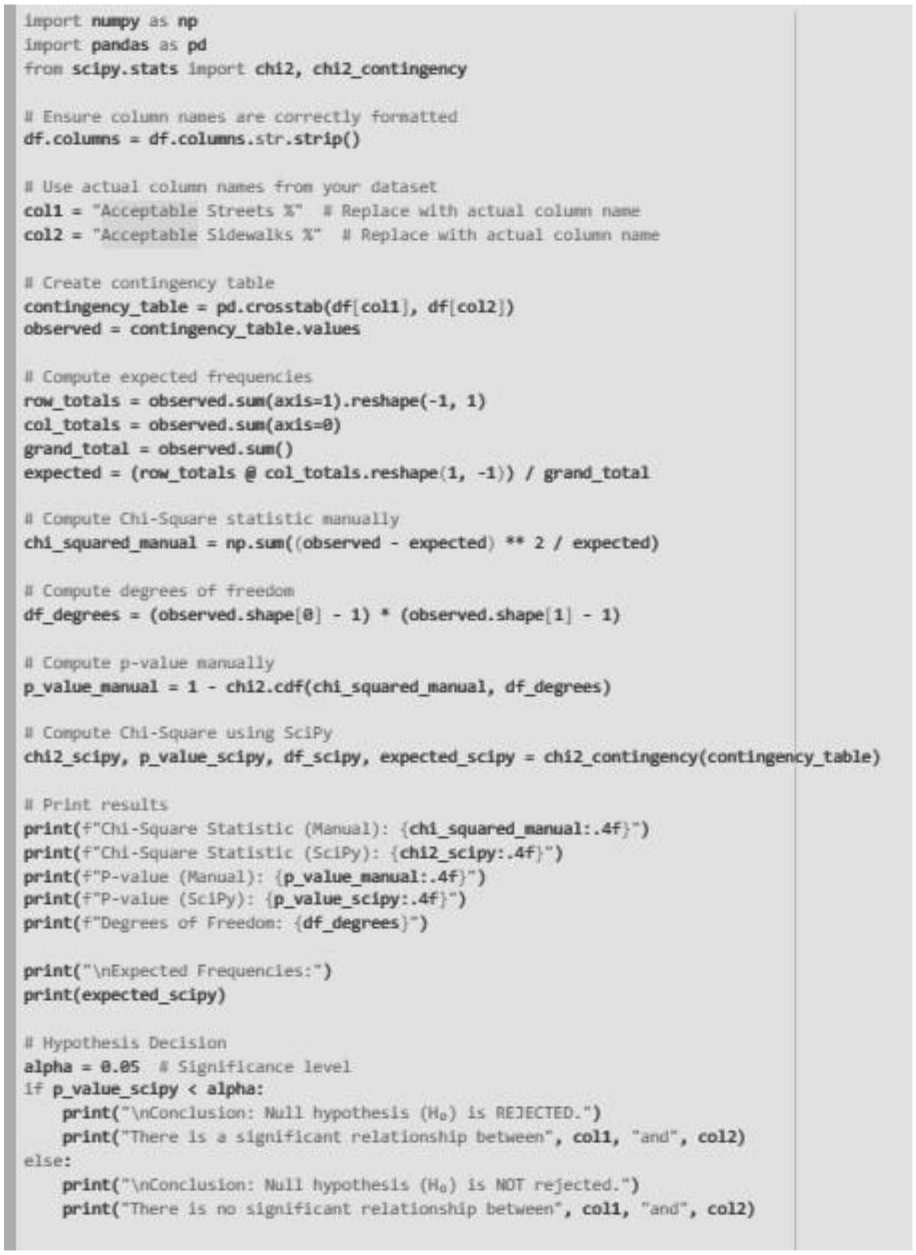
Measures association between two categorical variables.
Compares observed and expected frequencies in a contingency table.
Null hypothesis states that there is no association between the variables.
If the p-value is small (<0.05), we reject the null hypothesis (there is a significant association).

Name: Akanksha Shinde **Class: D15C** **Roll No: 53**

Formula:

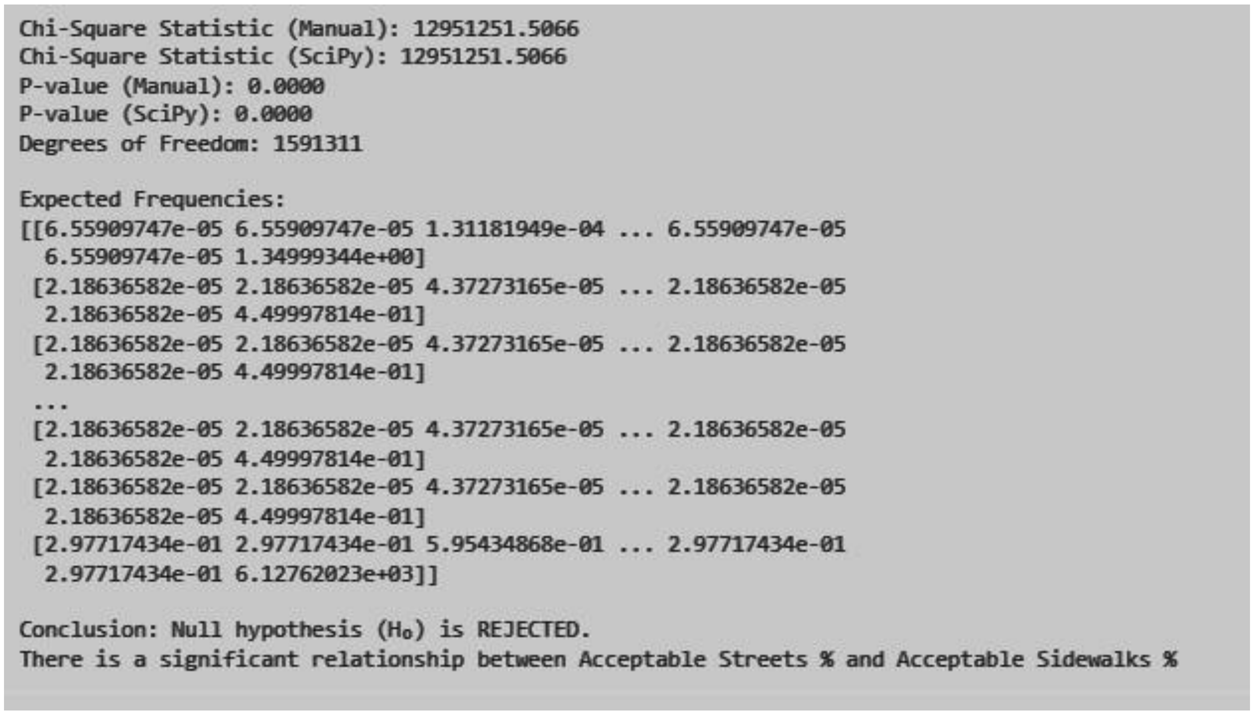
$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

where O is the observed frequency, and E is the expected frequency.



```python
import numpy as np
import pandas as pd
from scipy.stats import chi2, chi2_contingency

# Ensure column names are correctly formatted
df.columns = df.columns.str.strip()

# Use actual column names from your dataset
col1 = "Acceptable Streets %"  # Replace with actual column name
col2 = "Acceptable Sidewalks %"  # Replace with actual column name

# Create contingency table
contingency_table = pd.crosstab(df[col1], df[col2])
observed = contingency_table.values

# Compute expected frequencies
row_totals = observed.sum(axis=1).reshape(-1, 1)
col_totals = observed.sum(axis=0)
grand_total = observed.sum()
expected = (row_totals @ col_totals.reshape(1, -1)) / grand_total

# Compute Chi-Square statistic manually
chi_squared_manual = np.sum((observed - expected) ** 2 / expected)

# Compute degrees of freedom
df_degrees = (observed.shape[0] - 1) * (observed.shape[1] - 1)

# Compute p-value manually
p_value_manual = 1 - chi2.cdf(chi_squared_manual, df_degrees)

# Compute Chi-Square using SciPy
chi2_scipy, p_value_scipy, df_scipy, expected_scipy = chi2_contingency(contingency_table)

# Print results
print(f"Chi-Square Statistic (Manual): {chi_squared_manual:.4f}")
print(f"Chi-Square Statistic (SciPy): {chi2_scipy:.4f}")
print(f"P-value (Manual): {p_value_manual:.4f}")
print(f"P-value (SciPy): {p_value_scipy:.4f}")
print(f"Degrees of Freedom: {df_degrees}")

print("\nExpected Frequencies:")
print(expected_scipy)

# Hypothesis Decision
alpha = 0.05  # Significance level
if p_value_scipy < alpha:
    print("\nConclusion: Null hypothesis (H₀) is REJECTED.")
    print("There is a significant relationship between", col1, "and", col2)
else:
    print("\nConclusion: Null hypothesis (H₀) is NOT rejected.")
    print("There is no significant relationship between", col1, "and", col2)
```

Name: Akanksha Shinde **Class: D15C** **Roll No: 53**

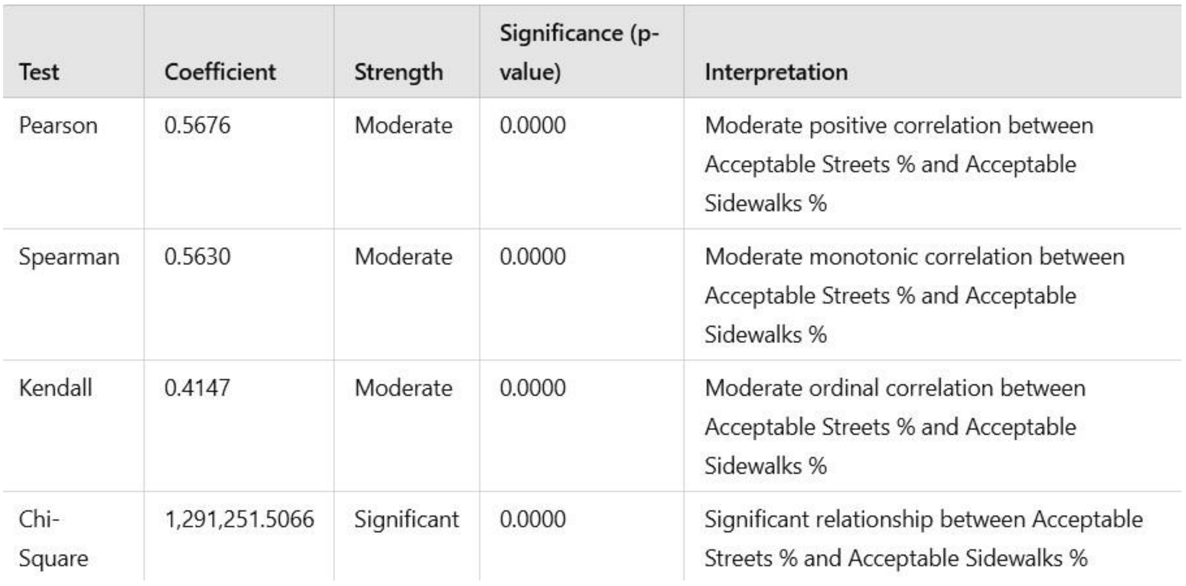


```
Chi-Square Statistic (Manual): 12951251.5066
Chi-Square Statistic (SciPy): 12951251.5066
P-value (Manual): 0.0000
P-value (SciPy): 0.0000
Degrees of Freedom: 1591311

Expected Frequencies:
[[6.55909747e-05 6.55909747e-05 1.31181949e-04 ... 6.55909747e-05
  6.55909747e-05 1.34999344e+00]
 [2.18636582e-05 2.18636582e-05 4.37273165e-05 ... 2.18636582e-05
  2.18636582e-05 4.49997814e-01]
 [2.18636582e-05 2.18636582e-05 4.37273165e-05 ... 2.18636582e-05
  2.18636582e-05 4.49997814e-01]
 ...
 [2.18636582e-05 2.18636582e-05 4.37273165e-05 ... 2.18636582e-05
  2.18636582e-05 4.49997814e-01]
 [2.18636582e-05 2.18636582e-05 4.37273165e-05 ... 2.18636582e-05
  2.18636582e-05 4.49997814e-01]
 [2.97717434e-01 2.97717434e-01 5.95434868e-01 ... 2.97717434e-01
  2.97717434e-01 6.12762023e+03]]

Conclusion: Null hypothesis (H₀) is REJECTED.
There is a significant relationship between Acceptable Streets % and Acceptable Sidewalks %
```

Chi-Square Test Results Interpretation

The Chi-Square statistic is 1,291,251.5066, which suggests a strong association between Acceptable Streets % and Acceptable Sidewalks %.
The p-value is 0.0000, which is much lower than the significance level (0.05).
Since p-value < 0.05, the null hypothesis (H_0) is rejected.
This indicates that there is a statistically significant relationship between Acceptable Streets % and Acceptable Sidewalks %.

RESULT:



Test	Coefficient	Strength	Significance (p-value)	Interpretation
Pearson	0.5676	Moderate	0.0000	Moderate positive correlation between Acceptable Streets % and Acceptable Sidewalks %
Spearman	0.5630	Moderate	0.0000	Moderate monotonic correlation between Acceptable Streets % and Acceptable Sidewalks %
Kendall	0.4147	Moderate	0.0000	Moderate ordinal correlation between Acceptable Streets % and Acceptable Sidewalks %
Chi-Square	1,291,251.5066	Significant	0.0000	Significant relationship between Acceptable Streets % and Acceptable Sidewalks %

Name: Akanksha Shinde **Class: D15C** **Roll No: 53**

Conclusion:

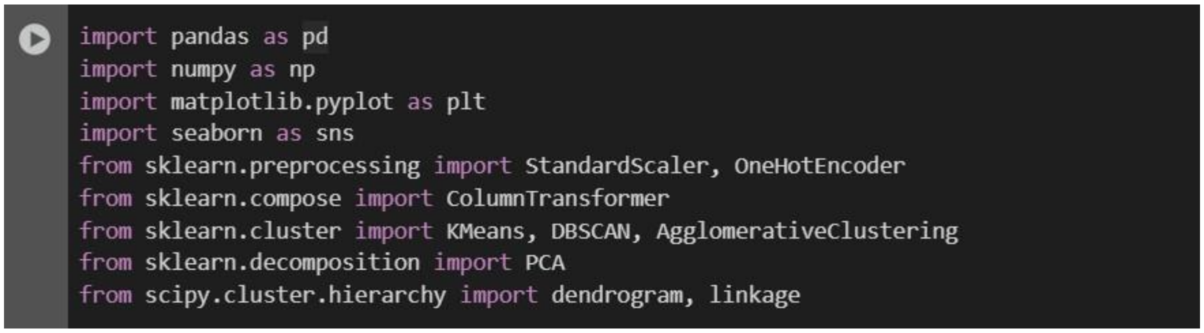
Pearson’s Correlation Coefficient showed a moderate positive correlation, indicating that areas with higher Acceptable Streets % tend to have higher Acceptable Sidewalks %.Spearman’s Rank Correlation confirmed the relationship remains moderate and monotonic, meaning that as Acceptable Streets % increases, Acceptable Sidewalks % generally increases in a consistent order. Kendall’s Rank Correlation indicated a moderate ordinal association, reinforcing the relationship between the two variables. Chi-Square Test showed a significant association between Acceptable Streets % and Acceptable Sidewalks %, as the p-value is 0.0000 (less than 0.05), leading to the rejection of the null hypothesis (H_0).

Name: Akanksha Shinde   Class: D15C   Roll No: 53

<u>Experiment No: 7</u>

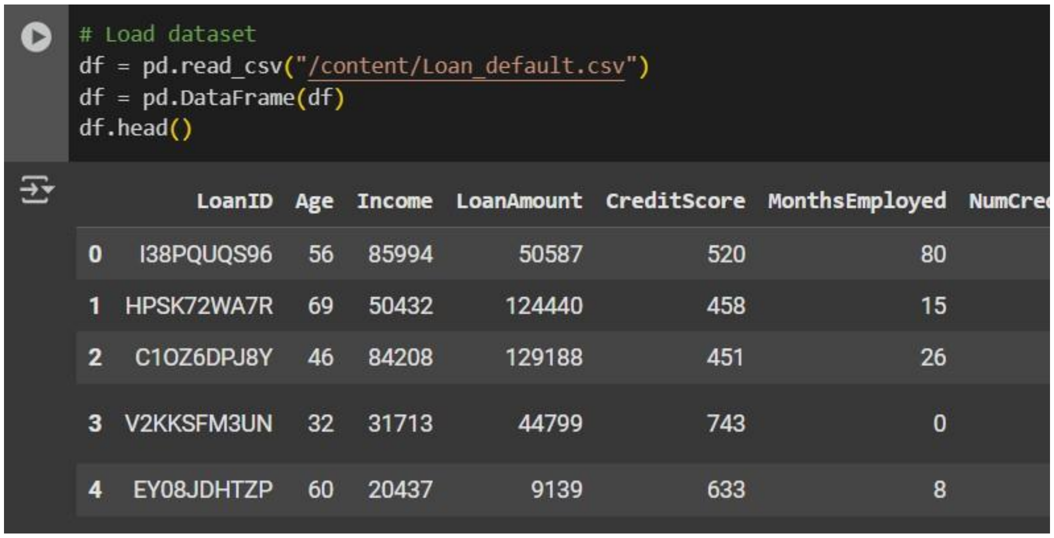
Aim: To implement different clustering algorithms.
Problem Statement: a) Clustering algorithm for unsupervised classification (K-means, density based
(DBSCAN), Hierarchical clustering)
b) Plot the cluster data and show mathematical steps.

Theory:
1. Import Libraries



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import dendrogram, linkage
```

2. Loading Dataset

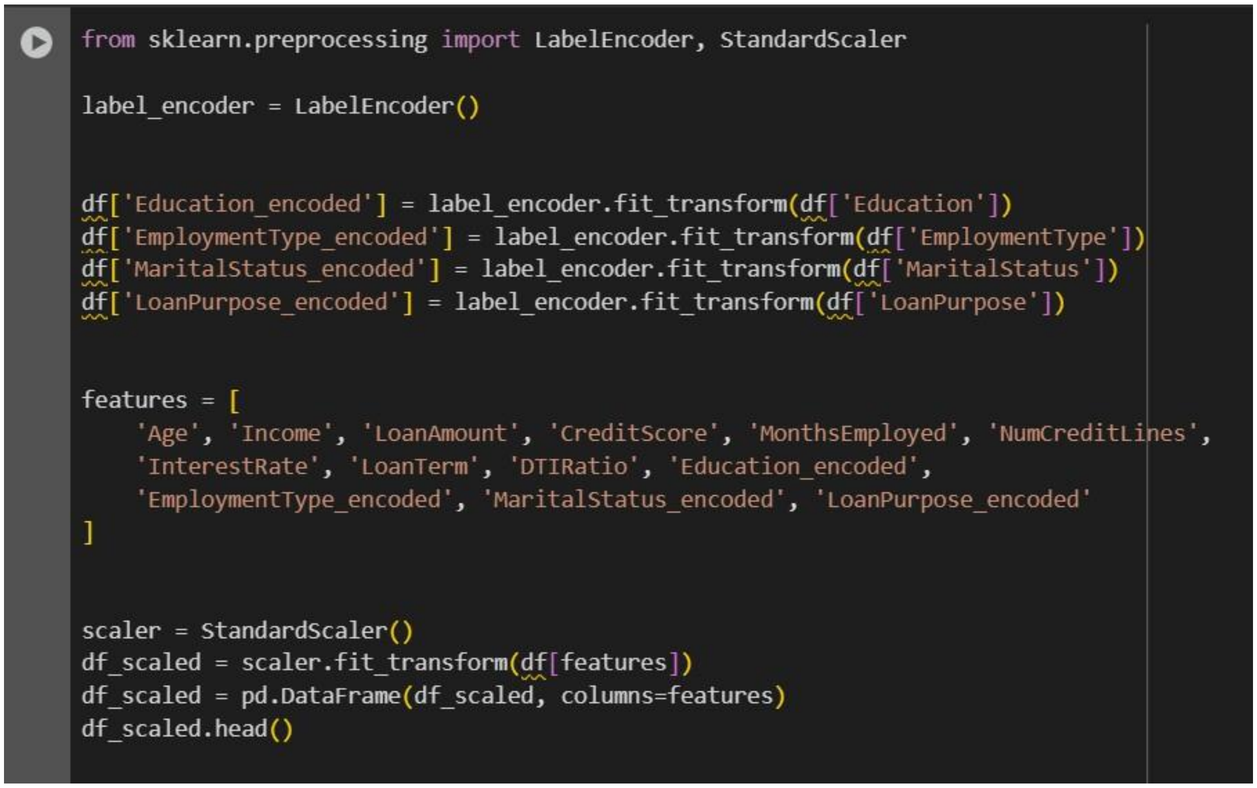


```
# Load dataset
df = pd.read_csv("/content/Loan_default.csv")
df = pd.DataFrame(df)
df.head()
```

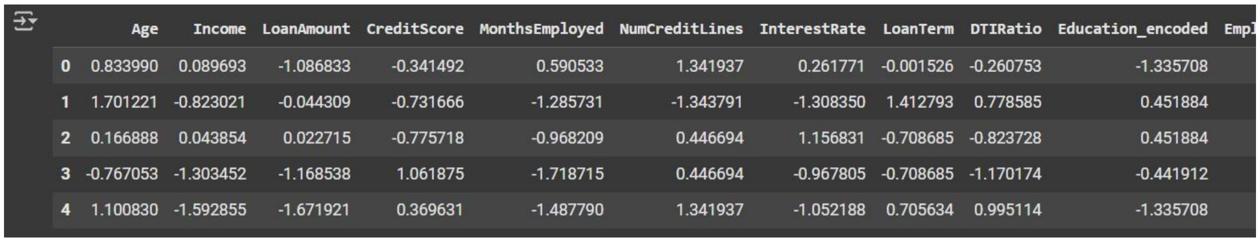
	LoanID	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCre
0	I38PQUQS96	56	85994	50587	520	80	
1	HPSK72WA7R	69	50432	124440	458	15	
2	C1OZ6DPJ8Y	46	84208	129188	451	26	
3	V2KKSFM3UN	32	31713	44799	743	0	
4	EY08JDHTZP	60	20437	9139	633	8	

Name: Akanksha Shinde   Class: D15C   Roll No: 53

3. Transformation



```
from sklearn.preprocessing import LabelEncoder, StandardScaler

label_encoder = LabelEncoder()


df['Education_encoded'] = label_encoder.fit_transform(df['Education'])
df['EmploymentType_encoded'] = label_encoder.fit_transform(df['EmploymentType'])
df['MaritalStatus_encoded'] = label_encoder.fit_transform(df['MaritalStatus'])
df['LoanPurpose_encoded'] = label_encoder.fit_transform(df['LoanPurpose'])


features = [
    'Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed', 'NumCreditLines',
    'InterestRate', 'LoanTerm', 'DTIRatio', 'Education_encoded',
    'EmploymentType_encoded', 'MaritalStatus_encoded', 'LoanPurpose_encoded'
]


scaler = StandardScaler()
df_scaled = scaler.fit_transform(df[features])
df_scaled = pd.DataFrame(df_scaled, columns=features)
df_scaled.head()
```

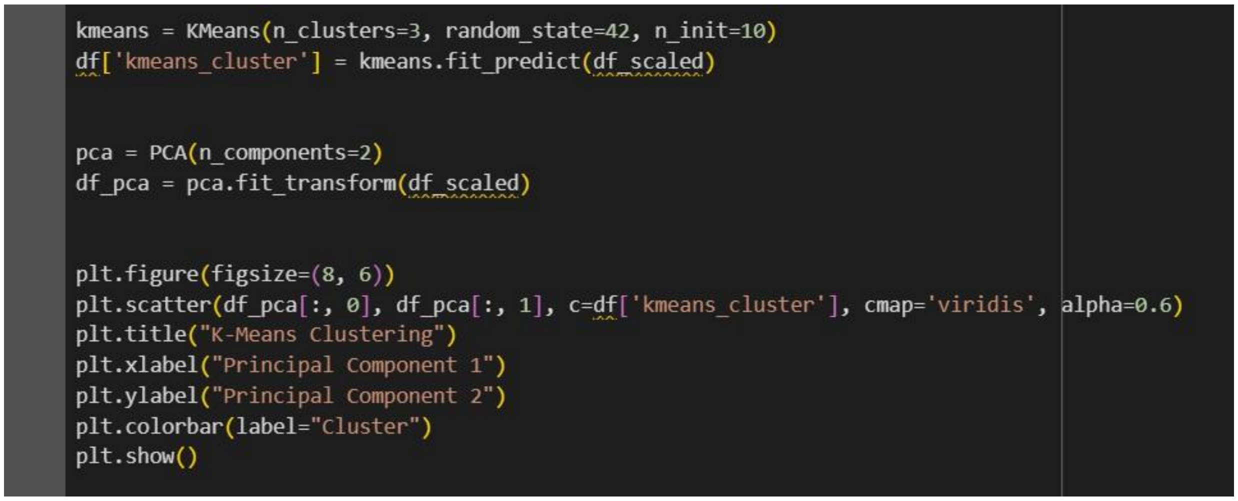


	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	LoanTerm	DTIRatio	Education_encoded	Empl
0	0.833990	0.089693	-1.086833	-0.341492	0.590533	1.341937	0.261771	-0.001526	-0.260753	-1.335708	
1	1.701221	-0.823021	-0.044309	-0.731666	-1.285731	-1.343791	-1.308350	1.412793	0.778585	0.451884	
2	0.166888	0.043854	0.022715	-0.775718	-0.968209	0.446694	1.156831	-0.708685	-0.823728	0.451884	
3	-0.767053	-1.303452	-1.168538	1.061875	-1.718715	0.446694	-0.967805	-0.708685	-1.170174	-0.441912	
4	1.100830	-1.592855	-1.671921	0.369631	-1.487790	1.341937	-1.052188	0.705634	0.995114	-1.335708	

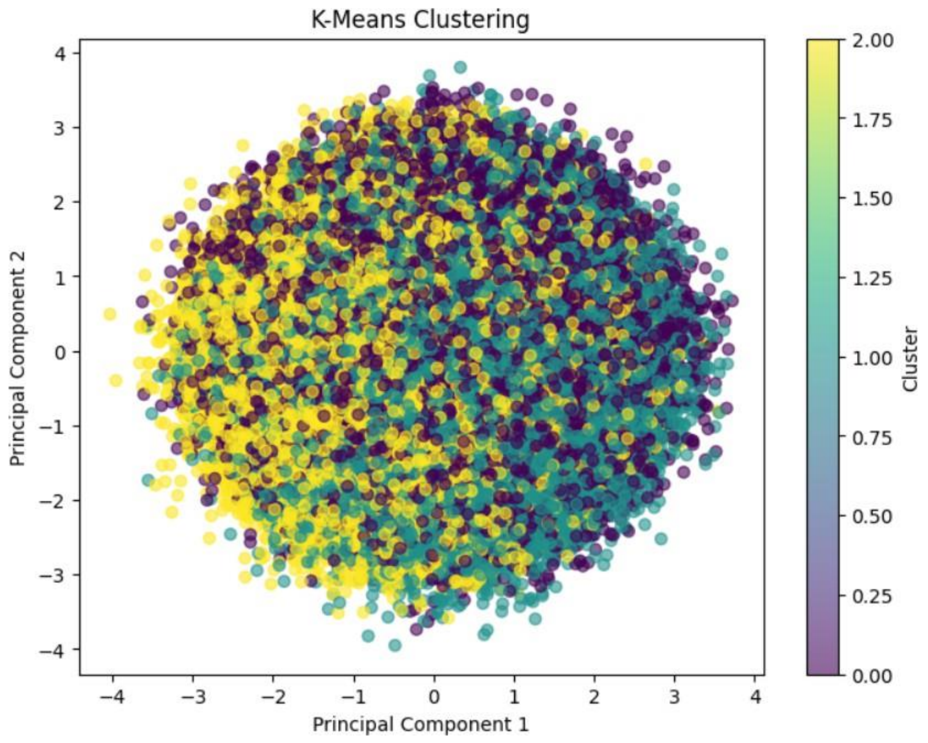
K-Means:

K-Means is a centroid-based clustering algorithm that partitions data into k clusters by minimizing the distance between data points and their respective cluster centers. It works well for well-separated, spherical clusters but may struggle with irregularly shaped distributions.

Name: Akanksha Shinde Class: D15C Roll No: 53



```
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['kmeans_cluster'] = kmeans.fit_predict(df_scaled)


pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)


plt.figure(figsize=(8, 6))
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['kmeans_cluster'], cmap='viridis', alpha=0.6)
plt.title("K-Means Clustering")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label="Cluster")
plt.show()
```

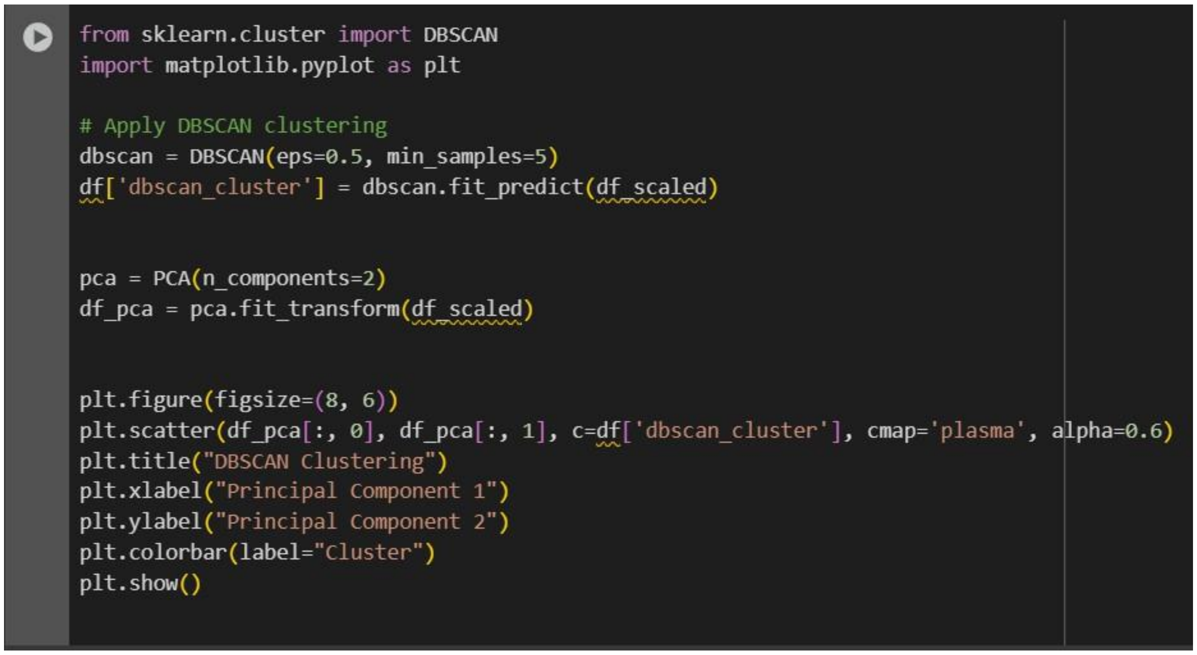


1. **Clusters Overlap** – The K-Means algorithm did not form well-separated clusters, indicating overlapping data distributions.
2. **K-Means Limitation** – The assumption of spherical clusters may not fit the dataset’s structure.
3. **Alternative Methods Needed** – DBSCAN or Hierarchical Clustering might work better for improved separation.

Name: Akanksha Shinde   Class: D15C   Roll No: 53

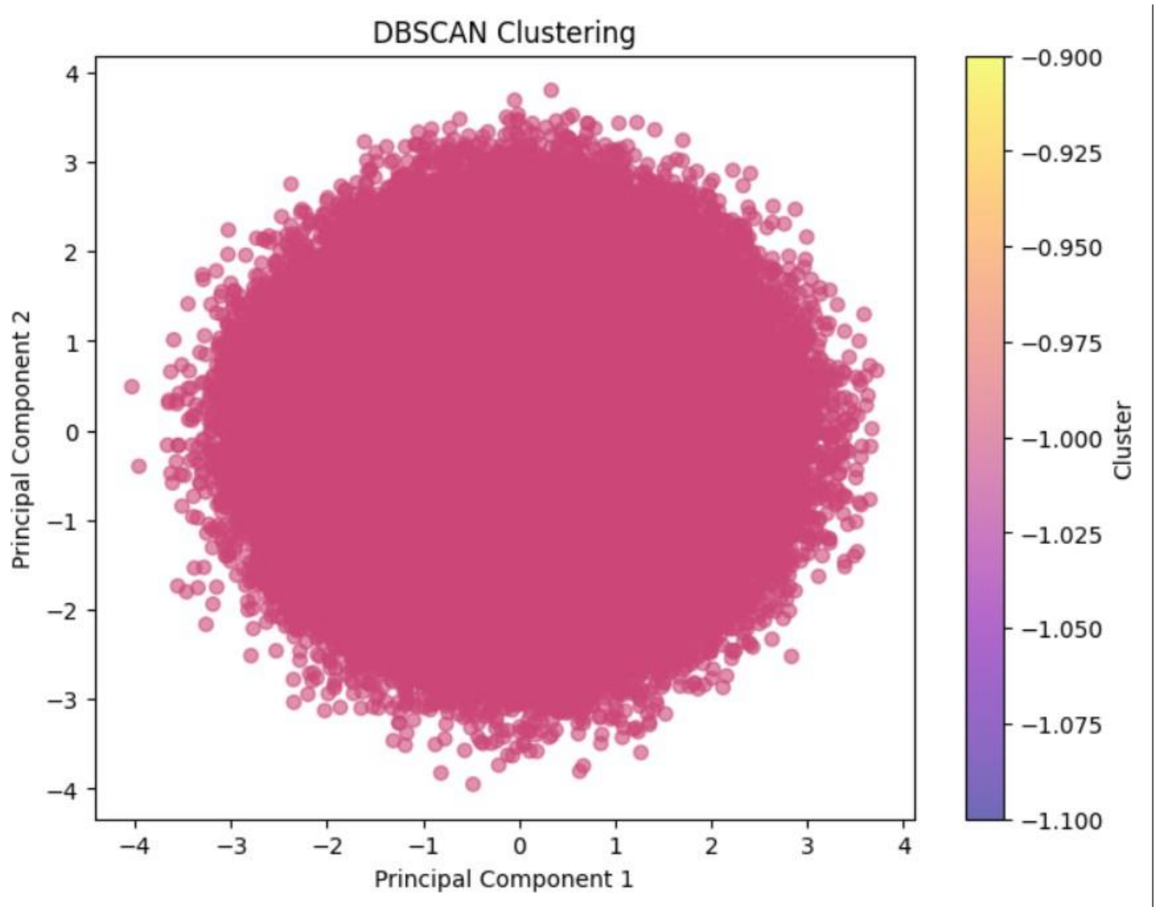
DBSCAN:

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups points based on density rather than distance to a centroid. It can identify arbitrarily shaped clusters and detect noise (outliers), making it more robust than K-Means for complex datasets.



```python
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
df['dbscan_cluster'] = dbscan.fit_predict(df_scaled)


pca = PCA(n_components=2)
df_pca = pca.fit_transform(df_scaled)


plt.figure(figsize=(8, 6))
plt.scatter(df_pca[:, 0], df_pca[:, 1], c=df['dbscan_cluster'], cmap='plasma', alpha=0.6)
plt.title("DBSCAN Clustering")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label="Cluster")
plt.show()
```

Name: Akanksha Shinde   Class: D15C   Roll No: 53



1. **No Clear Clusters** – DBSCAN failed to identify distinct groups, classifying most data points into a single cluster.
2. **Parameter Issue** – The chosen eps and min_samples may be inappropriate, leading to poor clustering results.
3. **Possible Fixes** – Adjust eps and min_samples, check data scaling, or try alternative clustering methods like K-Means or Hierarchical Clustering.

Conclusion:

K-Means Clustering successfully grouped the data but assumed spherical clusters, which may not always represent the true structure of the dataset. DBSCAN Clustering struggled to form meaningful clusters likely due to improper parameter selection or the dataset's characteristics.

Experiment No: 9

Aim: To perform Exploratory data analysis using Apache Spark And Pandas.

Theory -

1. What is Apache Spark and How It Works?

Apache Spark is an open-source data processing engine designed for big data workloads. It supports fast and general-purpose cluster computing.
It uses in-memory computation, meaning data is processed in RAM instead of slow disk drives, which greatly speeds up performance.
Spark supports multiple languages like Python (PySpark), Scala, Java, and R, making it flexible for developers and data scientists.
It follows a DAG (Directed Acyclic Graph) execution model, where transformations on data are organized as a sequence of stages.
Spark can run on various cluster managers like YARN, Mesos, or standalone, and supports tools like Hadoop HDFS and Apache Hive.

2. How is Data Exploration Done in Apache Spark? Explain Steps.

Load the data: Use Spark's read API to load datasets from sources like CSV, JSON, or databases into DataFrames.
Inspect schema: Use .printSchema() and .dtypes to understand the structure and data types of each column.
View sample data: Use .show() or .head() to preview the top rows and check for obvious issues or patterns.
Summary statistics: Use .describe() and .summary() to get basic statistics like mean, count, min, and max for numeric columns.
Check for nulls and duplicates: Use .filter(), .dropna(), or .dropDuplicates() to handle missing values or redundant records.

Conclusion:

Apache Spark is a robust, open-source distributed computing system designed for fast processing of large-scale data. Its in-memory computation model, support for multiple languages, and compatibility with various data sources make it an efficient choice for big data analytics. Data exploration in Apache Spark is performed through structured steps such as loading the dataset, inspecting schema, analyzing statistics, and handling missing or duplicate values. These steps help in gaining initial insights and preparing the data for further processing. Overall, Apache Spark simplifies complex data operations and is a valuable tool in the big data ecosystem.

Name: Akanksha Shinde   Class : D15C   Roll No: 53

Experiment No: 5

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.

Theory

1. Linear Regression:

Linear Regression is a supervised learning algorithm used to predict continuous numerical values.

- It assumes a linear relationship between the independent (input) and dependent (target) variables.
- The model minimizes the sum of squared errors to determine the best-fitting line.
- It is evaluated using performance metrics like Mean Squared Error (MSE) and R-squared (R^2).
- It works best when data follows a linear trend but is sensitive to outliers.

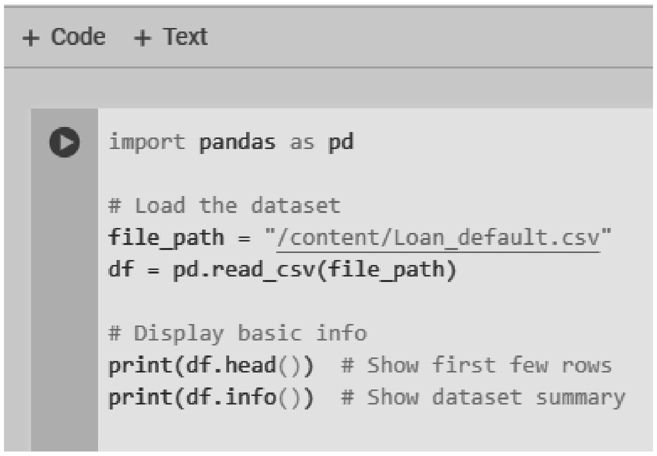
2. Logistic Regression:

Logistic Regression is a classification algorithm used for predicting binary outcomes (0 or 1).

- It applies the sigmoid function to convert linear outputs into probabilities.
- A threshold (typically 0.5) is used to classify data points.
- The model is trained using gradient descent to minimize the log loss function.
- Performance is assessed using accuracy, precision, recall, F1-score, and confusion matrices.

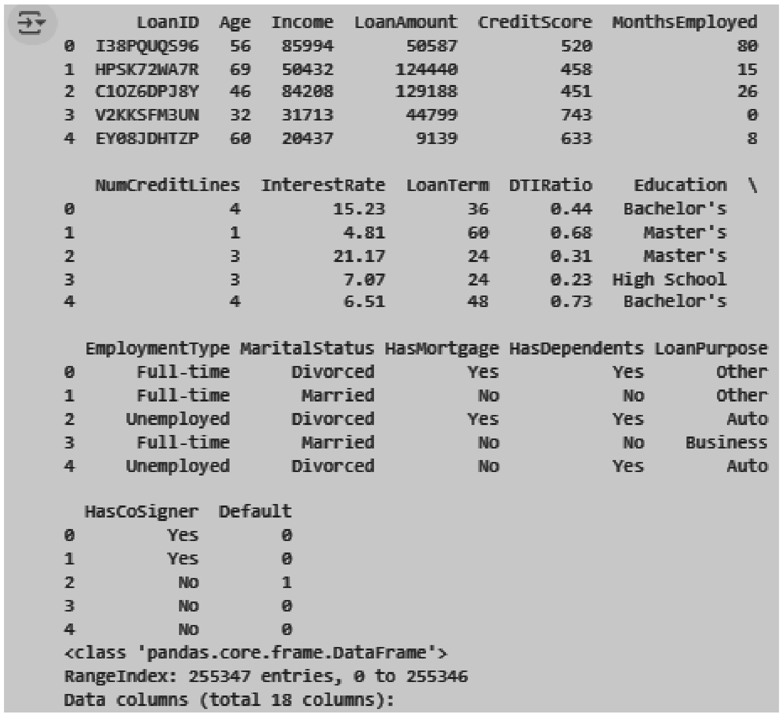
Name: Akanksha Shinde   Class : D15C   Roll No: 53

1. Import necessary libraries and load the dataset



```
import pandas as pd

# Load the dataset
file_path = "/content/Loan_default.csv"
df = pd.read_csv(file_path)

# Display basic info
print(df.head())  # Show first few rows
print(df.info())  # Show dataset summary
```

2. Check for missing values and anomalies in the dataset.

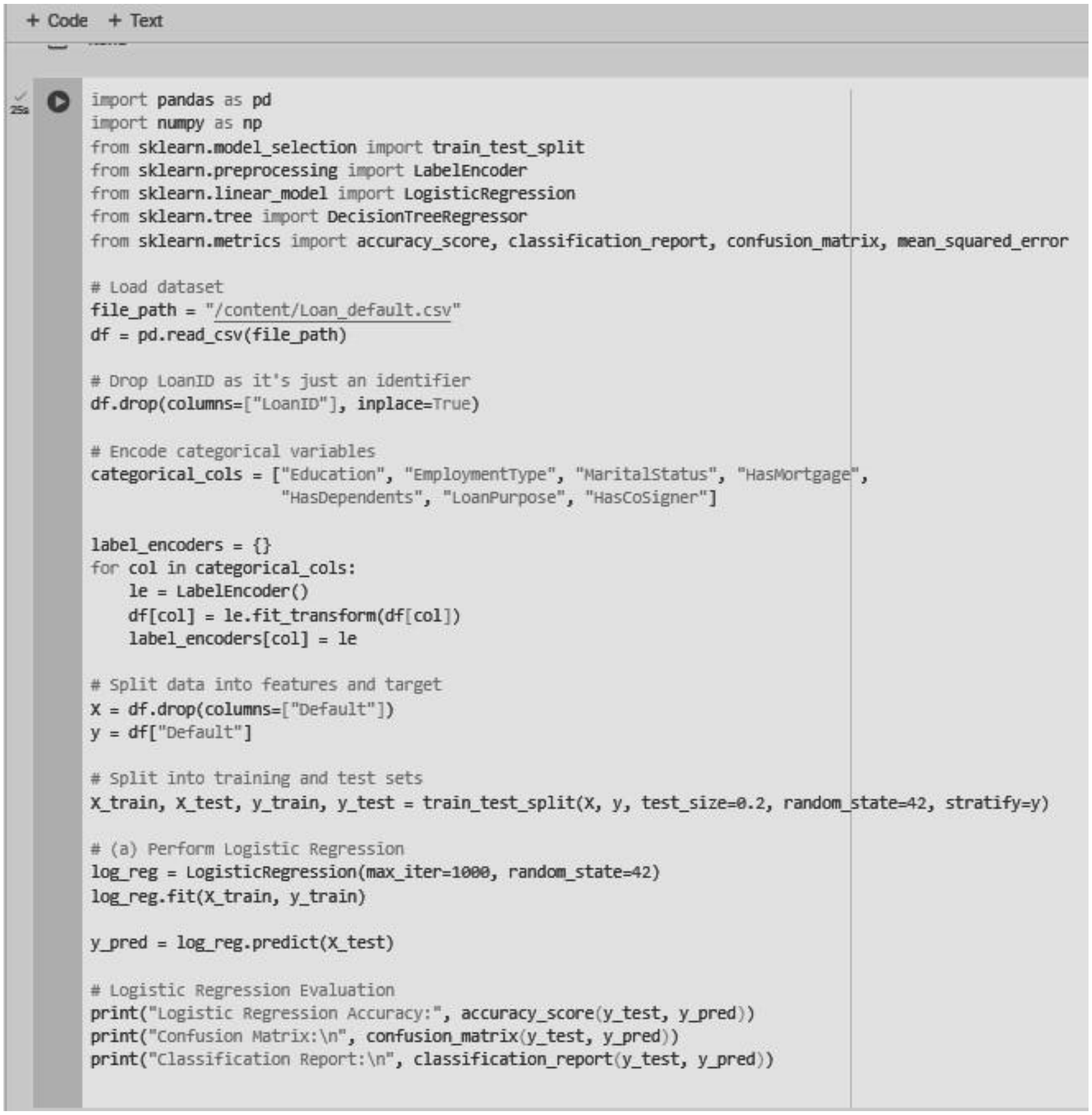


```
       LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed
0  I38PQUQS96   56   85994       50587          520              80
1  HPSK72WA7R   69   50432      124440          458              15
2  C1OZ6DPJ8Y   46   84208      129188          451              26
3  V2KKSFM3UN   32   31713       44799          743               0
4  EY08JDHTZP   60   20437        9139          633               8

   NumCreditLines  InterestRate  LoanTerm  DTIRatio    Education  \
0               4         15.23        36      0.44   Bachelor's
1               1          4.81        60      0.68     Master's
2               3         21.17        24      0.31     Master's
3               3          7.07        24      0.23  High School
4               4          6.51        48      0.73   Bachelor's

  EmploymentType MaritalStatus HasMortgage HasDependents LoanPurpose
0      Full-time      Divorced         Yes           Yes       Other
1      Full-time       Married          No            No       Other
2     Unemployed      Divorced         Yes           Yes        Auto
3      Full-time       Married          No            No    Business
4     Unemployed      Divorced          No           Yes        Auto

  HasCoSigner  Default
0         Yes        0
1         Yes        0
2          No        1
3          No        0
4          No        0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 255347 entries, 0 to 255346
Data columns (total 18 columns):
```

Name: Akanksha Shinde   Class : D15C   Roll No: 53

a) Perform Logistic regression to find out relation between variables

3. Train the dataset using Linear Regression



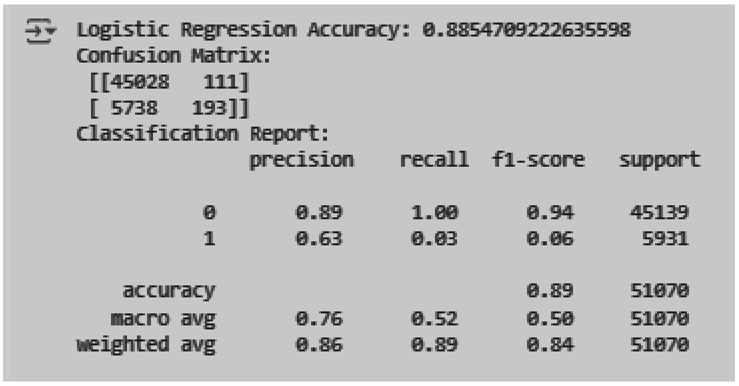
+ Code + Text

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, mean_squared_error

# Load dataset
file_path = "/content/Loan_default.csv"
df = pd.read_csv(file_path)

# Drop LoanID as it's just an identifier
df.drop(columns=["LoanID"], inplace=True)

# Encode categorical variables
categorical_cols = ["Education", "EmploymentType", "MaritalStatus", "HasMortgage",
                    "HasDependents", "LoanPurpose", "HasCoSigner"]

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Split data into features and target
X = df.drop(columns=["Default"])
y = df["Default"]

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# (a) Perform Logistic Regression
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)

y_pred = log_reg.predict(X_test)

# Logistic Regression Evaluation
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

4. Train the dataset using Logistic Regression and compute:

- Accuracy Score
- Precision, Recall, and F1-score

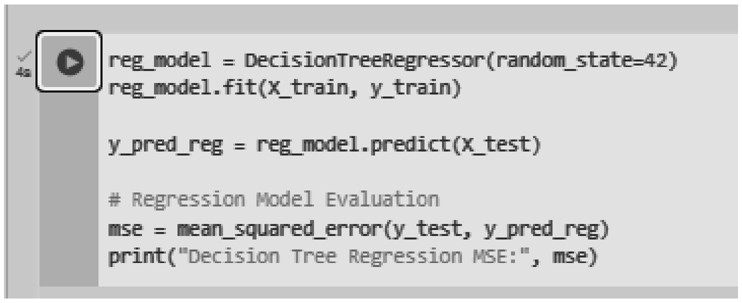
Name: Akanksha Shinde   Class : D15C   Roll No: 53



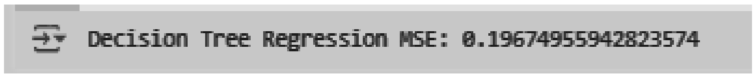
```
Logistic Regression Accuracy: 0.8854709222635598
Confusion Matrix:
 [[45028   111]
 [ 5738   193]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     45139
           1       0.63      0.03      0.06      5931

    accuracy                           0.89     51070
   macro avg       0.76      0.52      0.50     51070
weighted avg       0.86      0.89      0.84     51070
```

b) Apply regression model technique to predict the data on the above dataset.

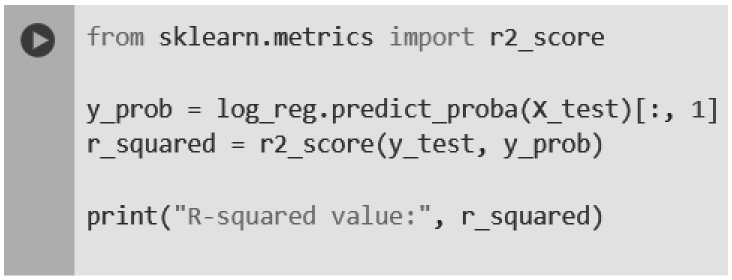
- **Find Mean Squared Error (MSE) and R-squared score.**



```
reg_model = DecisionTreeRegressor(random_state=42)
reg_model.fit(X_train, y_train)

y_pred_reg = reg_model.predict(X_test)

# Regression Model Evaluation
mse = mean_squared_error(y_test, y_pred_reg)
print("Decision Tree Regression MSE:", mse)
```



```
Decision Tree Regression MSE: 0.19674955942823574
```



```
from sklearn.metrics import r2_score

y_prob = log_reg.predict_proba(X_test)[:, 1]
r_squared = r2_score(y_test, y_prob)

print("R-squared value:", r_squared)
```

R-squared value : 0.3017

Conclusion: The logistic regression model achieves 88.54% accuracy, but its recall for loan defaulters (Class 1) is very low (0.03), indicating poor detection of actual defaults. The decision tree regression model has an MSE of 0.1967 and an R² score of 0.3017, showing moderate predictive power. Improving class balance and feature selection could enhance model performance.

Name: Akanksha Shinde   Class : D15C   Roll No: 53

Experiment No. 6

Aim: Classification modelling

a. Choose classifier for classification problem.

b. Evaluate the performance of classifier.

Theory:

Classification Modeling:

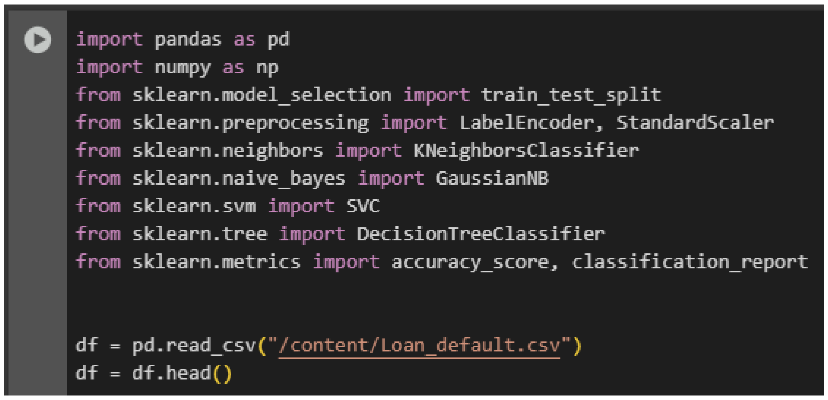
Classification is a **supervised machine learning** technique used to predict categorical labels based on input features. It is widely used in **medical diagnosis, spam detection, fraud detection, and more**.

Several classifiers can be used for classification tasks, each with its advantages and limitations:

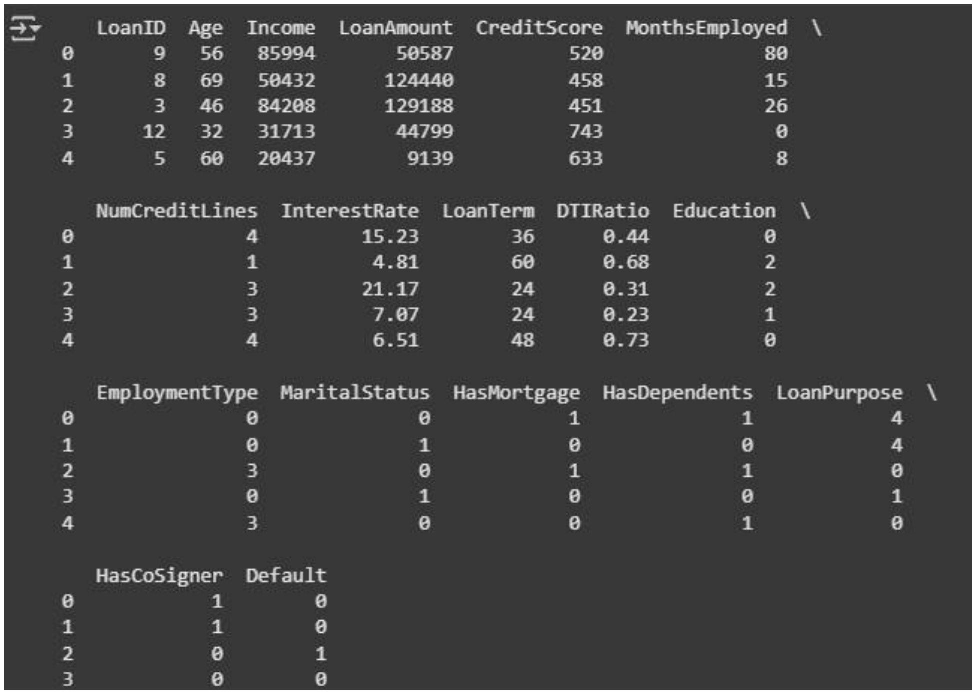
1. **K-Nearest Neighbors (KNN):**
    - A non-parametric, instance-based learning algorithm.
    - Classifies data based on the majority label of its **K** nearest neighbors.
    - Works well for structured and small datasets but **slower for large datasets**.
2. **Naïve Bayes:**
    - A probabilistic classifier based on **Bayes' Theorem**.
    - Assumes **independence** between features, making it efficient for large datasets.
    - Performs well in **text classification** but struggles when features are highly correlated.
3. **Support Vector Machine (SVM):**
    - A powerful classifier that finds the **optimal hyperplane** for separating classes.
    - Works well for **high-dimensional** datasets.
    - Sensitive to **noisy data** and computationally expensive for large datasets.
4. **Decision Tree:**
    - A tree-structured model that splits data based on feature values.
    - **Easy to interpret** and requires minimal data preprocessing.
    - Prone to **overfitting**, which can reduce generalization performance.

1.Load the dataset -

Name: Akanksha Shinde   Class : D15C   Roll No: 53

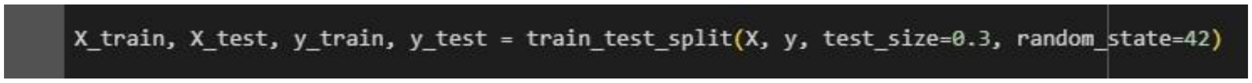


```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report


df = pd.read_csv("/content/Loan_default.csv")
df = df.head()
```



```
   LoanID  Age  Income  LoanAmount  CreditScore  MonthsEmployed  \
0       9   56   85994       50587          520              80   
1       8   69   50432      124440          458              15   
2       3   46   84208      129188          451              26   
3      12   32   31713       44799          743               0   
4       5   60   20437        9139          633               8   

   NumCreditLines  InterestRate  LoanTerm  DTIRatio  Education  \
0               4         15.23        36      0.44          0   
1               1          4.81        60      0.68          2   
2               3         21.17        24      0.31          2   
3               3          7.07        24      0.23          1   
4               4          6.51        48      0.73          0   

   EmploymentType  MaritalStatus  HasMortgage  HasDependents  LoanPurpose  \
0               0              0            1              1            4   
1               0              1            0              0            4   
2               3              0            1              1            0   
3               0              1            0              0            1   
4               3              0            0              1            0   

   HasCoSigner  Default  
0            1        0  
1            1        0  
2            0        1  
3            0        0  
```

2.Splitting data into training and testing sets



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

3.KNN

Name: Akanksha Shinde   Class : D15C   Roll No: 53

```
# Classifiers
def train_and_evaluate(model, name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(f"\n{name} Classifier:\n")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
    print("Classification Report:\n", classification_report(y_test, y_pred))

# KNN
knn = KNeighborsClassifier(n_neighbors=3)
train_and_evaluate(knn, "K-Nearest Neighbors")
```

```
K-Nearest Neighbors Classifier:

Accuracy: 0.80
Classification Report:
               precision    recall  f1-score   support

           0       0.75      1.00      0.86         3
           1       1.00      0.50      0.67         2

    accuracy                           0.80         5
   macro avg       0.88      0.75      0.76         5
weighted avg       0.85      0.80      0.78         5
```

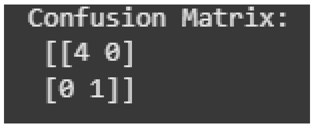
```
Confusion Matrix:
 [[4 0]
 [1 0]]
```

4.Naive Bayes

```
# Naive Bayes
nb = GaussianNB()
train_and_evaluate(nb, "Naive Bayes")
```

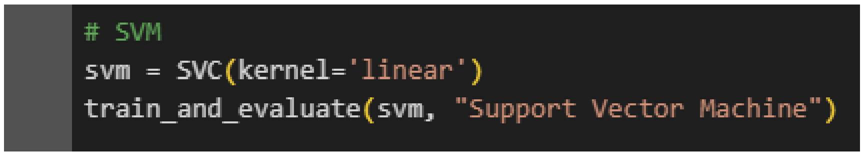
```
Naive Bayes Classifier:

Accuracy: 0.60
Classification Report:
               precision    recall  f1-score   support

           0       0.60      1.00      0.75         3
           1       0.00      0.00      0.00         2

    accuracy                           0.60         5
   macro avg       0.30      0.50      0.38         5
weighted avg       0.36      0.60      0.45         5
```

Name: Akanksha Shinde   Class : D15C   Roll No: 53

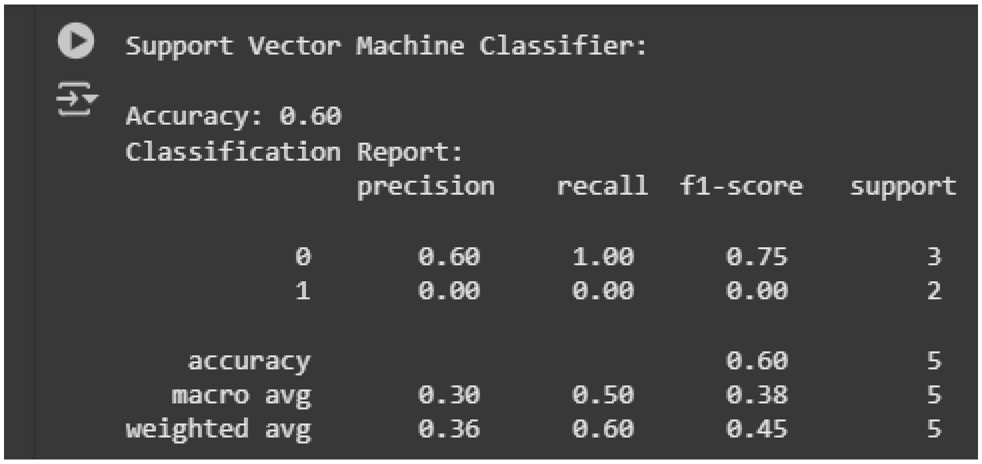


```
Confusion Matrix:
 [[4 0]
 [0 1]]
```

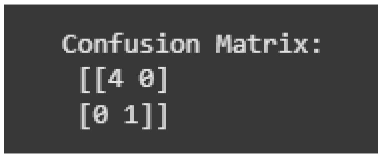
5.Support Vector Machine



```
# SVM
svm = SVC(kernel='linear')
train_and_evaluate(svm, "Support Vector Machine")
```

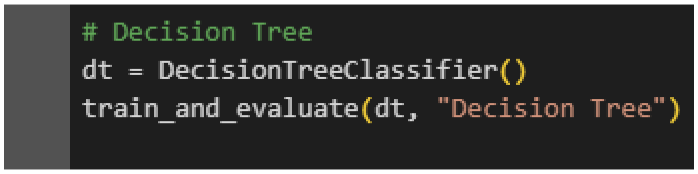


```
Support Vector Machine Classifier:

Accuracy: 0.60
Classification Report:
              precision    recall  f1-score   support

           0       0.60      1.00      0.75         3
           1       0.00      0.00      0.00         2

    accuracy                           0.60         5
   macro avg       0.30      0.50      0.38         5
weighted avg       0.36      0.60      0.45         5
```

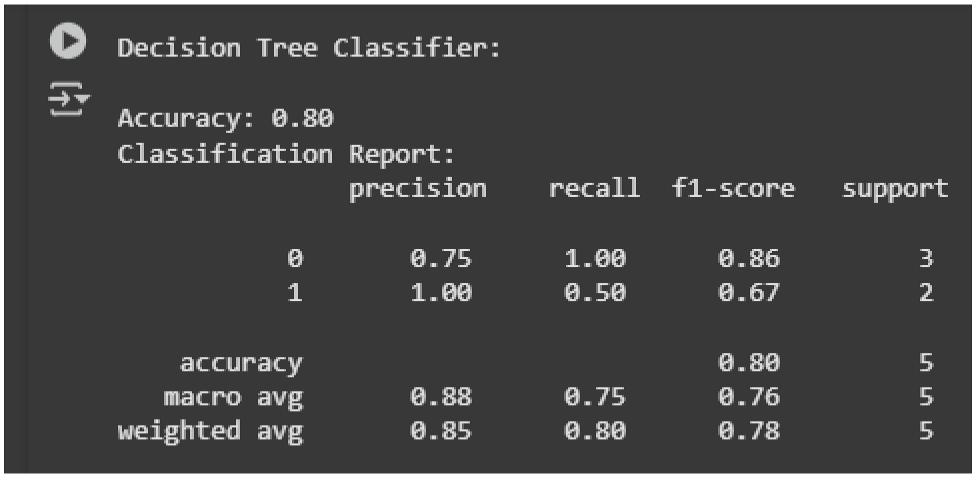


```
Confusion Matrix:
 [[4 0]
 [0 1]]
```

6.Decision Tree

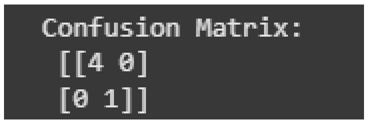


```
# Decision Tree
dt = DecisionTreeClassifier()
train_and_evaluate(dt, "Decision Tree")
```



```
Decision Tree Classifier:

Accuracy: 0.80
Classification Report:
              precision    recall  f1-score   support

           0       0.75      1.00      0.86         3
           1       1.00      0.50      0.67         2

    accuracy                           0.80         5
   macro avg       0.88      0.75      0.76         5
weighted avg       0.85      0.80      0.78         5
```

Name: Akanksha Shinde   Class : D15C   Roll No: 53



```
Confusion Matrix:
 [[4 0]
 [0 1]]
```

Conclusion :

Conclusion for Loan Default Prediction Models

•KNN and Decision Tree achieved 80% accuracy, making them effective for loan default prediction.

•Naïve Bayes and SVM had lower accuracy (60%), indicating weaker performance.

•KNN and Decision Tree balanced precision and recall well, making them suitable for further tuning.

Improvements can be made with class balancing, feature selection, and hyperparameter optimization.

Experiment No: 8

Aim: To implement recommendation system on your dataset using the following machine learning techniques.
o Regression
o Classification
o Clustering
o Decision tree
o Anomaly detection
o Dimensionality Reduction
o Ensemble Methods

Theory:

1. Regression

Regression is a supervised learning technique used to predict continuous values based on input features. It estimates the relationship between dependent and independent variables.

- Linear Regression: Models a straight-line relationship between variables.
- Polynomial Regression: Captures non-linear relationships by introducing polynomial terms.
- Ridge/Lasso Regression: Regularized versions that prevent overfitting.
- Logistic Regression: Used for classification despite the name "regression."

2. Classification

Classification is a supervised learning technique that categorizes input data into predefined classes or labels.

- Binary Classification: Two possible outcomes (e.g., spam vs. not spam).
- Multiclass Classification: More than two categories (e.g., classifying animals as cat, dog, or bird).

- Popular Algorithms: Logistic Regression, Decision Trees, SVM, Random Forest, Neural Networks.

3. Clustering

Clustering is an unsupervised learning technique used to group similar data points together based on patterns. Unlike classification, clusters are not predefined.

- K-Means: Partitions data into K clusters using centroids.
- Hierarchical Clustering: Forms a tree-like structure of nested clusters.
- DBSCAN: Groups based on density, identifying outliers as noise.

4. Decision Tree

A Decision Tree is a tree-like structure where data is split into branches based on feature values. It is used for both classification and regression.

5. Anomaly Detection

Anomaly detection identifies unusual patterns that deviate significantly from normal data. It is widely used in fraud detection, cybersecurity, and medical diagnosis.

- Statistical Methods: Z-score, Gaussian distribution analysis.
- Machine Learning Methods: Isolation Forest, One-Class SVM, Autoencoders.
- Distance-Based Methods: k-Nearest Neighbors (k-NN) for detecting outliers.

6. Dimensionality Reduction

Dimensionality reduction is used to reduce the number of input features while preserving essential information. This helps improve model efficiency and visualization.

- Principal Component Analysis (PCA): Converts correlated features into uncorrelated principal components.

- t-SNE (t-Distributed Stochastic Neighbor Embedding): Useful for visualizing high-dimensional data.
- Autoencoders: Neural networks that learn compressed representations.

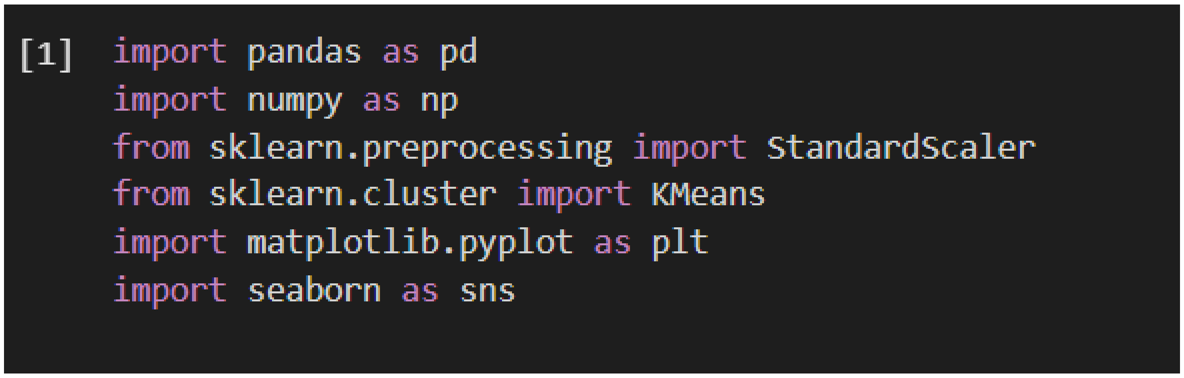
7. Ensemble Methods

Ensemble methods combine multiple models to improve accuracy and robustness. They work by aggregating predictions from multiple weak learners.

- Bagging (Bootstrap Aggregating): Example: Random Forest (uses multiple decision trees).
- Boosting: Example: AdaBoost, XGBoost (sequentially improves weak models).
- Stacking: Combines multiple models using another model (meta-learner) to make final predictions.

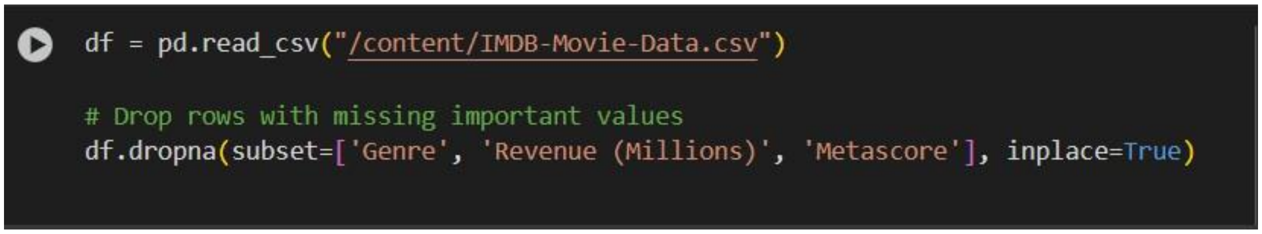
1.Clustering

Importing Libraries



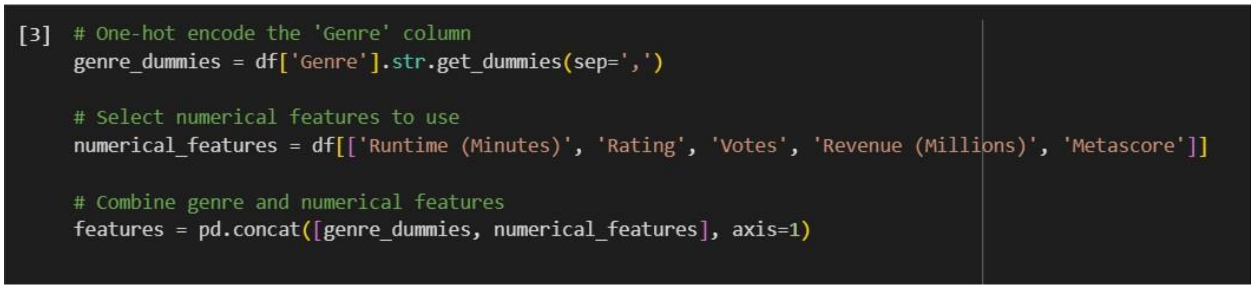
```
[1]  import pandas as pd
     import numpy as np
     from sklearn.preprocessing import StandardScaler
     from sklearn.cluster import KMeans
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Loading dataset



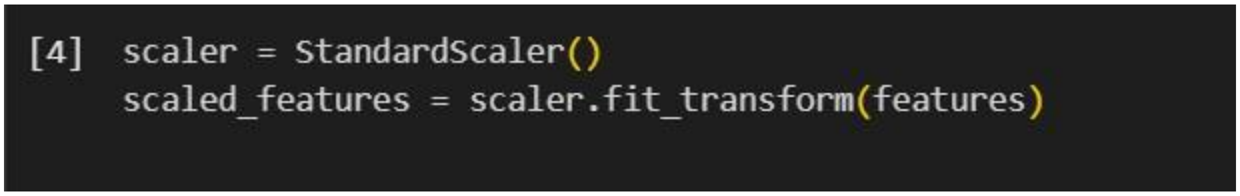
```
df = pd.read_csv("/content/IMDB-Movie-Data.csv")

# Drop rows with missing important values
df.dropna(subset=['Genre', 'Revenue (Millions)', 'Metascore'], inplace=True)
```

Prepare Features

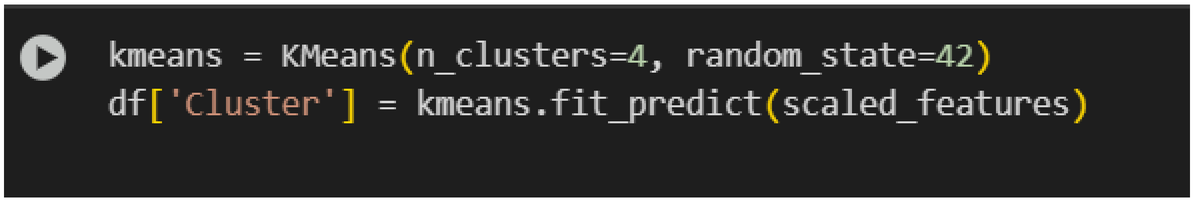


```
[3] # One-hot encode the 'Genre' column
    genre_dummies = df['Genre'].str.get_dummies(sep=',')

    # Select numerical features to use
    numerical_features = df[['Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)', 'Metascore']]

    # Combine genre and numerical features
    features = pd.concat([genre_dummies, numerical_features], axis=1)
```

Scale Features

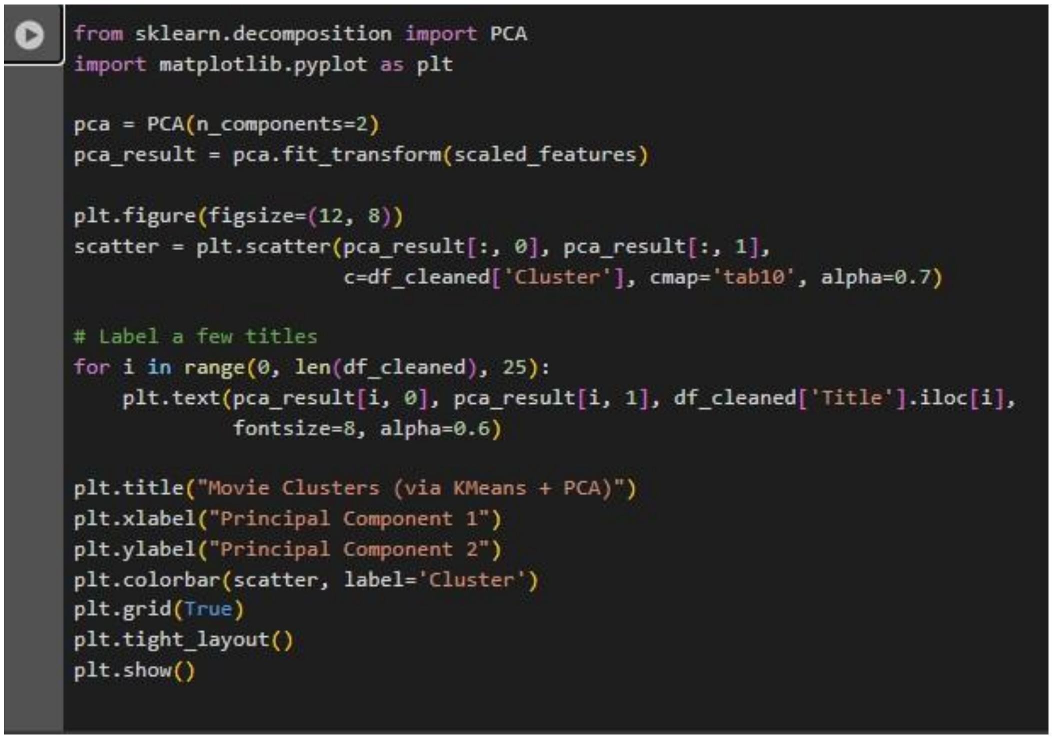


```
[4] scaler = StandardScaler()
    scaled_features = scaler.fit_transform(features)
```

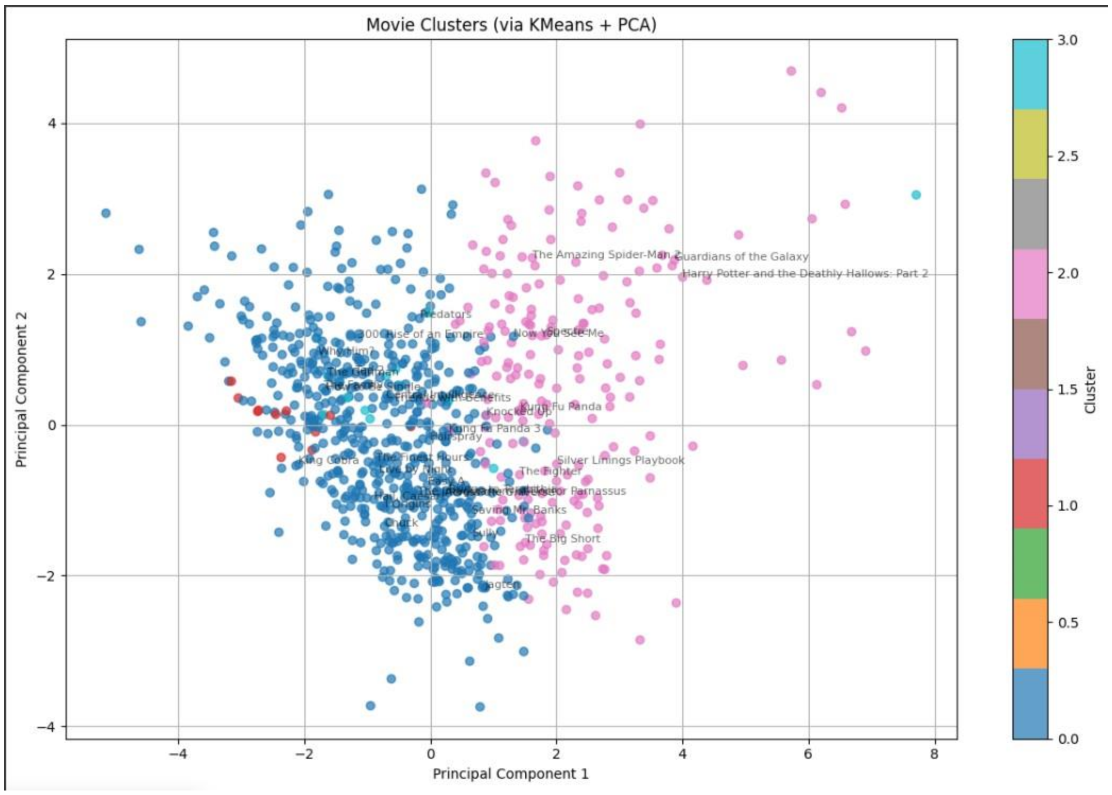


```
kmeans = KMeans(n_clusters=4, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_features)
```

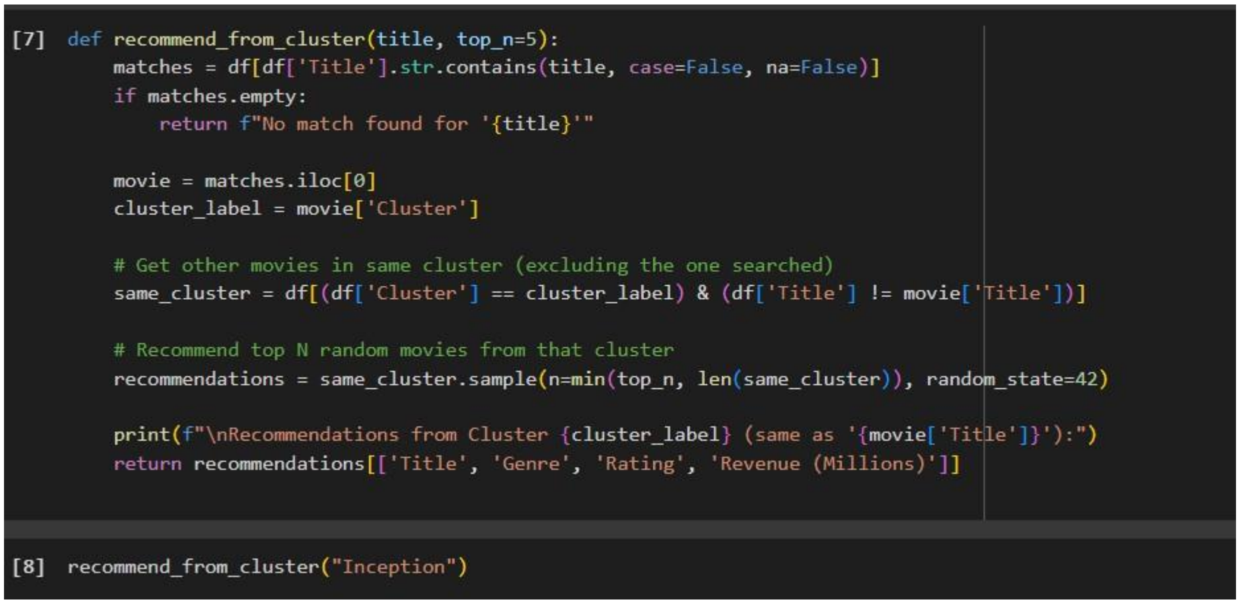
Visualize



```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA(n_components=2)
pca_result = pca.fit_transform(scaled_features)

plt.figure(figsize=(12, 8))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1],
                      c=df_cleaned['Cluster'], cmap='tab10', alpha=0.7)

# Label a few titles
for i in range(0, len(df_cleaned), 25):
    plt.text(pca_result[i, 0], pca_result[i, 1], df_cleaned['Title'].iloc[i],
             fontsize=8, alpha=0.6)

plt.title("Movie Clusters (via KMeans + PCA)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(scatter, label='Cluster')
plt.grid(True)
plt.tight_layout()
plt.show()
```

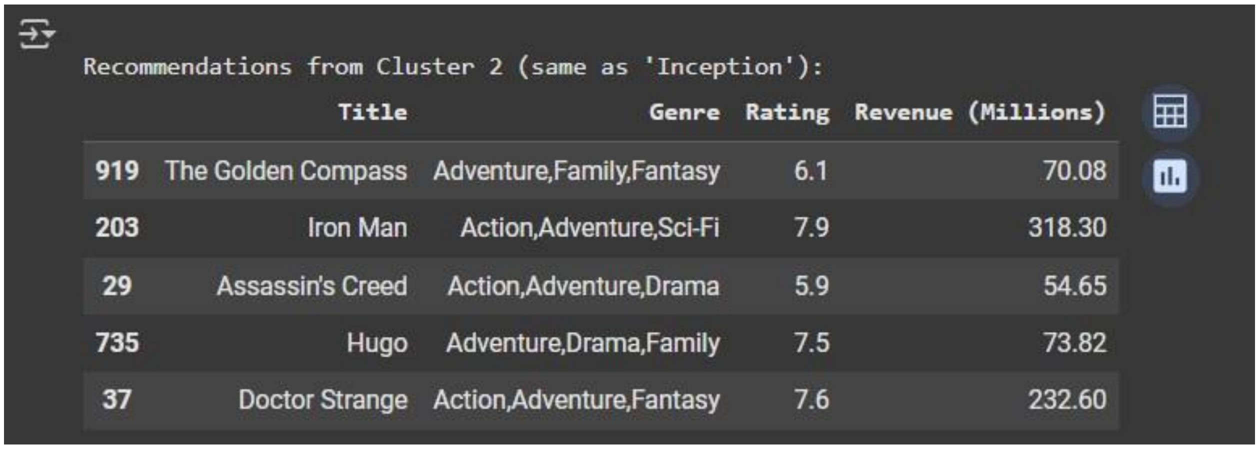


Recommendation



```
[7] def recommend_from_cluster(title, top_n=5):
        matches = df[df['Title'].str.contains(title, case=False, na=False)]
        if matches.empty:
            return f"No match found for '{title}'"

        movie = matches.iloc[0]
        cluster_label = movie['Cluster']

        # Get other movies in same cluster (excluding the one searched)
        same_cluster = df[(df['Cluster'] == cluster_label) & (df['Title'] != movie['Title'])]

        # Recommend top N random movies from that cluster
        recommendations = same_cluster.sample(n=min(top_n, len(same_cluster)), random_state=42)

        print(f"\nRecommendations from Cluster {cluster_label} (same as '{movie['Title']}'):")
        return recommendations[['Title', 'Genre', 'Rating', 'Revenue (Millions)']]
```

```
[8] recommend_from_cluster("Inception")
```

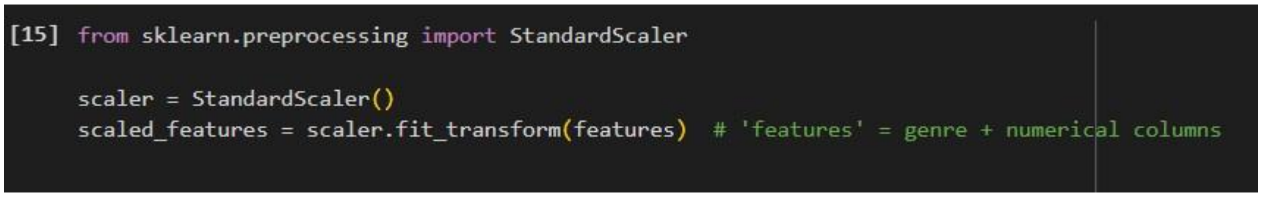


```
Recommendations from Cluster 2 (same as 'Inception'):
```

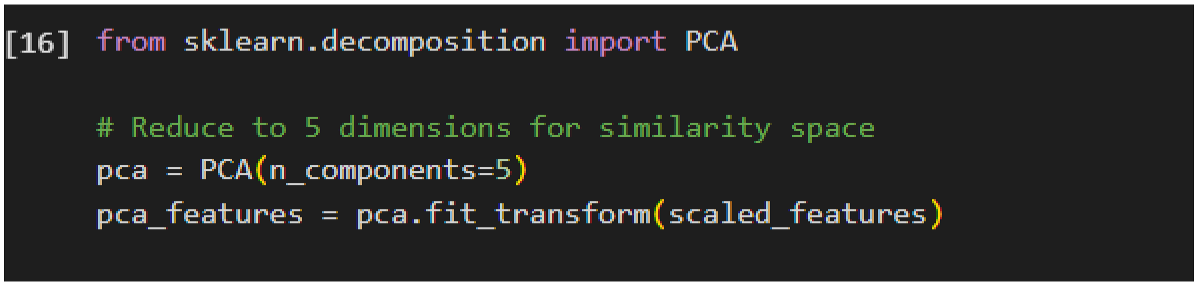
	Title	Genre	Rating	Revenue (Millions)
919	The Golden Compass	Adventure,Family,Fantasy	6.1	70.08
203	Iron Man	Action,Adventure,Sci-Fi	7.9	318.30
29	Assassin's Creed	Action,Adventure,Drama	5.9	54.65
735	Hugo	Adventure,Drama,Family	7.5	73.82
37	Doctor Strange	Action,Adventure,Fantasy	7.6	232.60

2.Dimension Reduction

Importing Libraries

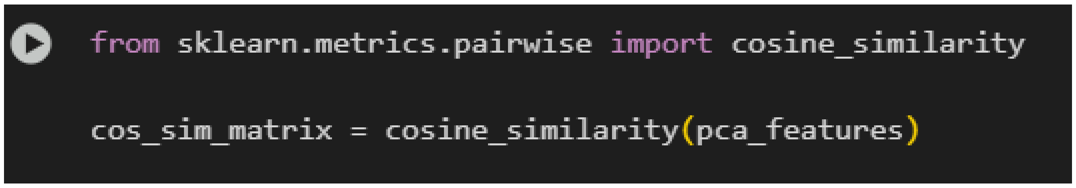


```
[15] from sklearn.preprocessing import StandardScaler

     scaler = StandardScaler()
     scaled_features = scaler.fit_transform(features)  # 'features' = genre + numerical columns
```



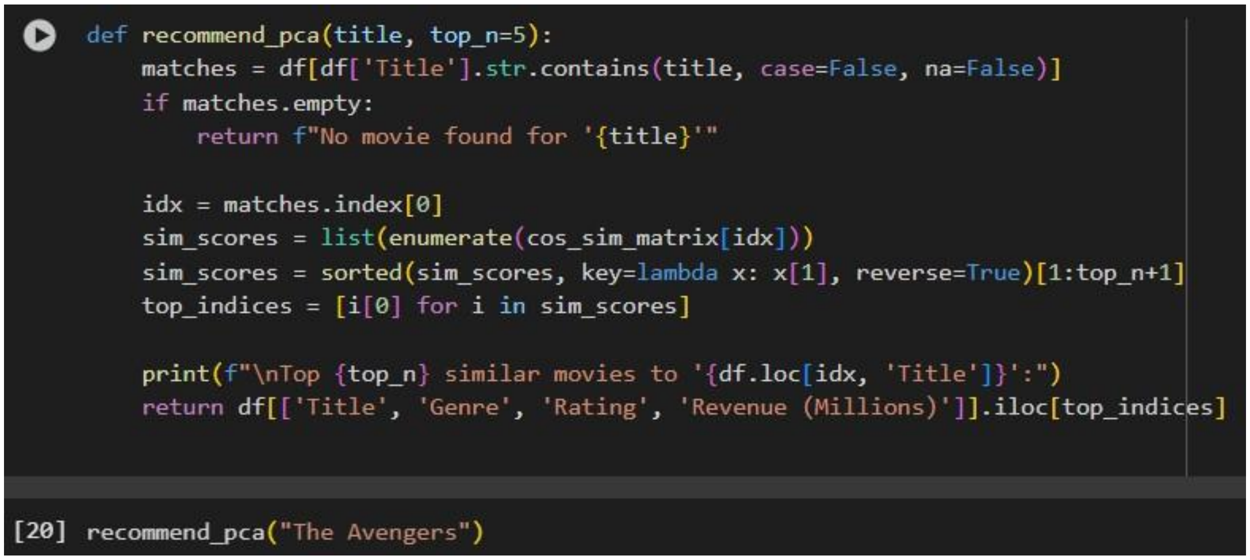
```
[16] from sklearn.decomposition import PCA

     # Reduce to 5 dimensions for similarity space
     pca = PCA(n_components=5)
     pca_features = pca.fit_transform(scaled_features)
```

Cosine Similarity



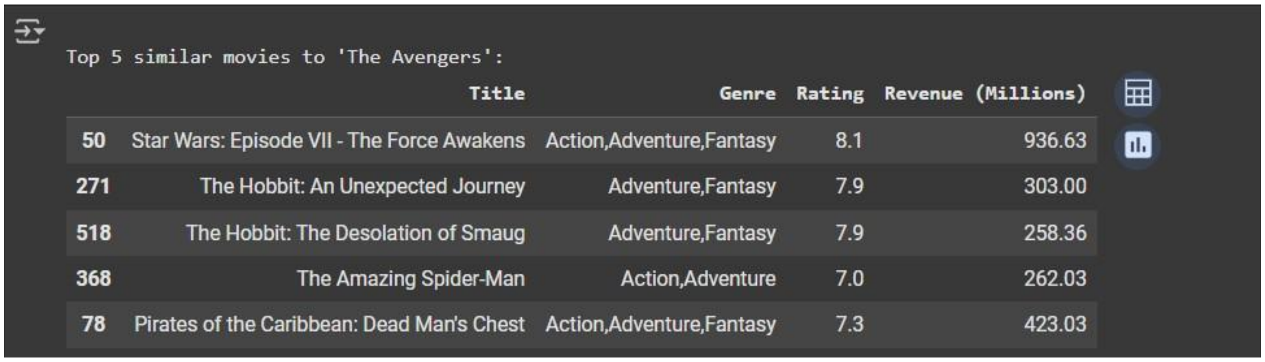
```
from sklearn.metrics.pairwise import cosine_similarity

cos_sim_matrix = cosine_similarity(pca_features)
```

Recommendation



```python
def recommend_pca(title, top_n=5):
    matches = df[df['Title'].str.contains(title, case=False, na=False)]
    if matches.empty:
        return f"No movie found for '{title}'"

    idx = matches.index[0]
    sim_scores = list(enumerate(cos_sim_matrix[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:top_n+1]
    top_indices = [i[0] for i in sim_scores]

    print(f"\nTop {top_n} similar movies to '{df.loc[idx, 'Title']}':")
    return df[['Title', 'Genre', 'Rating', 'Revenue (Millions)']].iloc[top_indices]
```

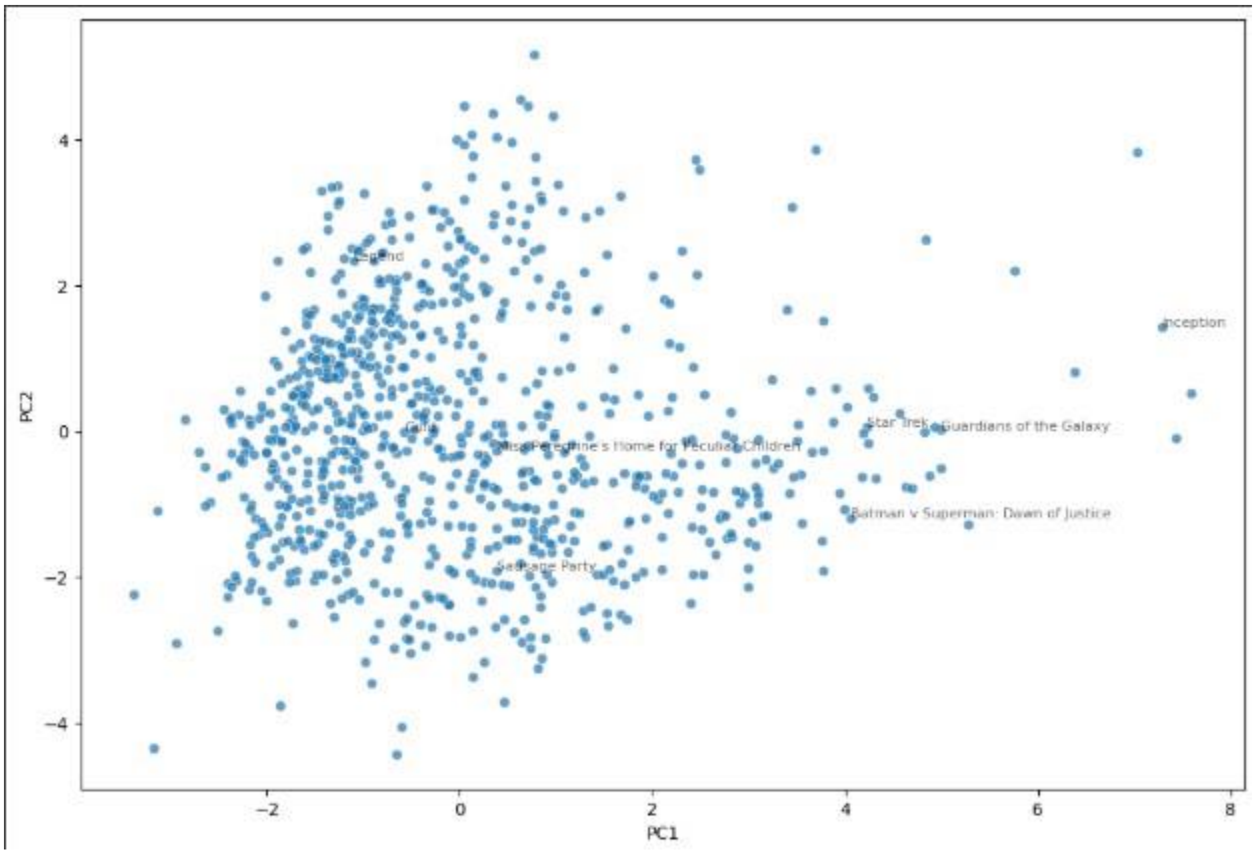
```
[20] recommend_pca("The Avengers")
```



```
Top 5 similar movies to 'The Avengers':
```

	Title	Genre	Rating	Revenue (Millions)
50	Star Wars: Episode VII - The Force Awakens	Action,Adventure,Fantasy	8.1	936.63
271	The Hobbit: An Unexpected Journey	Adventure,Fantasy	7.9	303.00
518	The Hobbit: The Desolation of Smaug	Adventure,Fantasy	7.9	258.36
368	The Amazing Spider-Man	Action,Adventure	7.0	262.03
78	Pirates of the Caribbean: Dead Man's Chest	Action,Adventure,Fantasy	7.3	423.03

Visualization



Conclusion-

In this experiment, we built a recommendation system using various machine learning techniques.Regression and classification helped in predicting and categorizing user preferences.Clustering grouped similar users/items for better suggestions. Dimensionality reduction improved efficiency, while ensemble methods boosted accuracy. Overall, the combined approach enhanced recommendation quality and system performance.

Experiment No: 10

Aim: To perform Batch and Streamed Data Analysis using Apache Spark.

Theory:

1. What is Streaming? Explain Batch and Stream Data.

Streaming refers to the continuous flow of data that is processed in real-time or near real-time as it arrives.
Batch Data is collected over a period and then processed together as a group. It is suitable for high-volume, less time-sensitive operations.
Stream Data is generated continuously (like sensor data, logs, transactions) and requires real-time or near-real-time processing.
Batch Processing is ideal for complex analytics and operations that don't require immediate results.
Stream Processing enables instant insights and actions based on incoming data, making it ideal for live dashboards, fraud detection, etc.

2. How Data Streaming Takes Place Using Apache Spark.

Apache Spark uses Spark Streaming or Structured Streaming for processing real-time data streams.
Data from sources like Kafka, Flume, or socket connections can be ingested continuously using the Spark Streaming API.
In Spark Streaming, the data stream is divided into small time-based batches called micro-batches.
Each micro-batch is then processed using standard Spark transformations and actions.
With Structured Streaming, Spark processes data using the same DataFrame and SQL APIs, treating streaming data as a continuously growing table.
The output can be stored or pushed to dashboards, databases, or file systems for further use.

Conclusion:

Apache Spark efficiently handles both batch and stream processing through its unified engine. While batch processing is suitable for historical data analysis, stream processing enables real-time analytics on continuously incoming data. Spark's Structured Streaming model simplifies real-time data handling using familiar APIs and ensures fault-tolerant and scalable performance. Through this experiment, the practical understanding of how Apache Spark processes both static and dynamic data was achieved, highlighting its significance in real-world big data applications.