Name:Atharva Shinde Roll No:54 Div:D15C

EXP 1

Aim:

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- standardization and normalization of columns

Data preprocessing

Data preprocessing involves transforming raw data into a structured and meaningful format, making it suitable for analysis. It is a crucial step in data mining, as raw data often contains inconsistencies, missing values, or noise. Ensuring data quality is essential before applying machine learning or data mining algorithms to achieve accurate and reliable results.

Why is Data Preprocessing Important?

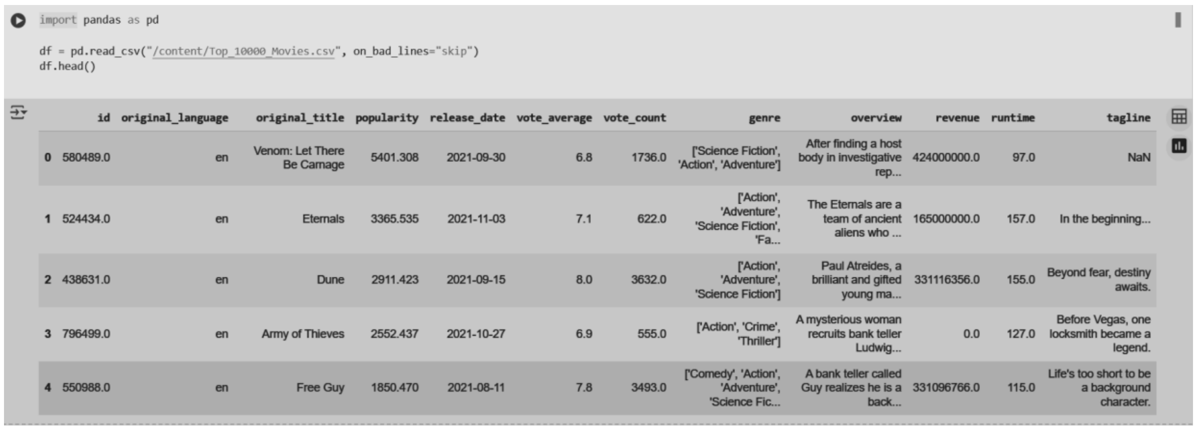
Data preprocessing is essential for ensuring the quality and reliability of data before analysis. It helps improve the accuracy and efficiency of machine learning and data mining processes. The key aspects of data quality include:

- **Accuracy:** Ensuring the data is correct and free from errors.
- **Completeness:** Checking for missing or unrecorded data.
- **Consistency:** Verifying that data remains uniform across different sources.
- **Timeliness:** Ensuring the data is up-to-date and relevant.
- **Believability:** Assessing whether the data is reliable and trustworthy.
- **Interpretability:** Making sure the data is clear and easy to understand.

Dataset: Top_10000_Movies

Name:Atharva Shinde Roll No:54 Div:D15C

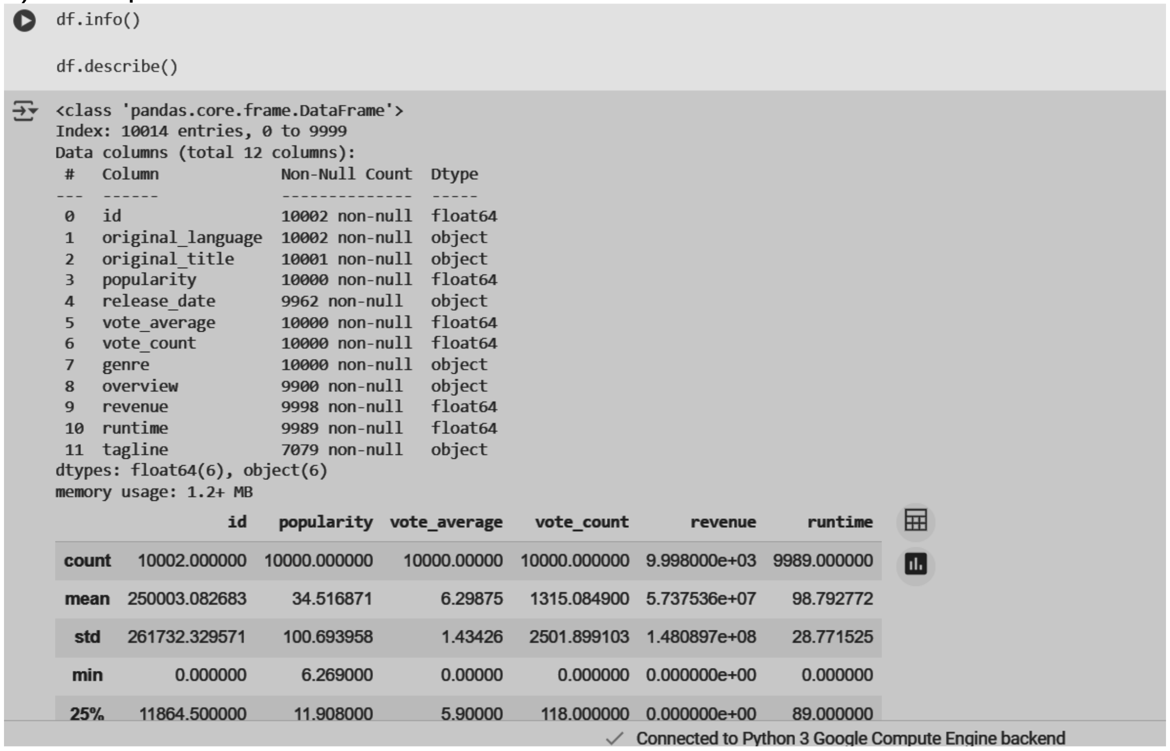
1) Loading Data in Pandas



```
import pandas as pd

df = pd.read_csv("/content/Top_10000_Movies.csv", on_bad_lines="skip")
df.head()
```

	id	original_language	original_title	popularity	release_date	vote_average	vote_count	genre	overview	revenue	runtime	tagline
0	580489.0	en	Venom: Let There Be Carnage	5401.308	2021-09-30	6.8	1736.0	['Science Fiction', 'Action', 'Adventure']	After finding a host body in investigative rep...	424000000.0	97.0	NaN
1	524434.0	en	Eternals	3365.535	2021-11-03	7.1	622.0	['Action', 'Adventure', 'Science Fiction', 'Fa...	The Eternals are a team of ancient aliens who ...	165000000.0	157.0	In the beginning...
2	438631.0	en	Dune	2911.423	2021-09-15	8.0	3632.0	['Action', 'Adventure', 'Science Fiction']	Paul Atreides, a brilliant and gifted young ma...	331116356.0	155.0	Beyond fear, destiny awaits.
3	796499.0	en	Army of Thieves	2552.437	2021-10-27	6.9	555.0	['Action', 'Crime', 'Thriller']	A mysterious woman recruits bank teller Ludwig...	0.0	127.0	Before Vegas, one locksmith became a legend.
4	550988.0	en	Free Guy	1850.470	2021-08-11	7.8	3493.0	['Comedy', 'Action', 'Adventure', 'Science Fic...	A bank teller called Guy realizes he is a back...	331096766.0	115.0	Life's too short to be a background character.

2) Description of the dataset.



```
df.info()

df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10014 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 10002 non-null  float64
 1   original_language  10002 non-null  object
 2   original_title     10001 non-null  object
 3   popularity         10000 non-null  float64
 4   release_date       9962 non-null   object
 5   vote_average       10000 non-null  float64
 6   vote_count         10000 non-null  float64
 7   genre              10000 non-null  object
 8   overview           9900 non-null   object
 9   revenue            9998 non-null   float64
 10  runtime            9989 non-null   float64
 11  tagline            7079 non-null   object
dtypes: float64(6), object(6)
memory usage: 1.2+ MB
```

	id	popularity	vote_average	vote_count	revenue	runtime
count	10002.000000	10000.000000	10000.00000	10000.000000	9.998000e+03	9989.000000
mean	250003.082683	34.516871	6.29875	1315.084900	5.737536e+07	98.792772
std	261732.329571	100.693958	1.43426	2501.899103	1.480897e+08	28.771525
min	0.000000	6.269000	0.00000	0.000000	0.000000e+00	0.000000
25%	11864.500000	11.908000	5.90000	118.000000	0.000000e+00	89.000000

✓ Connected to Python 3 Google Compute Engine backend

df.info(): Provides an overview of the dataset, including:

- Number of rows and columns.
- Data types of each column (e.g., int, float, object).
- Number of non-null (non-missing) values in each column.

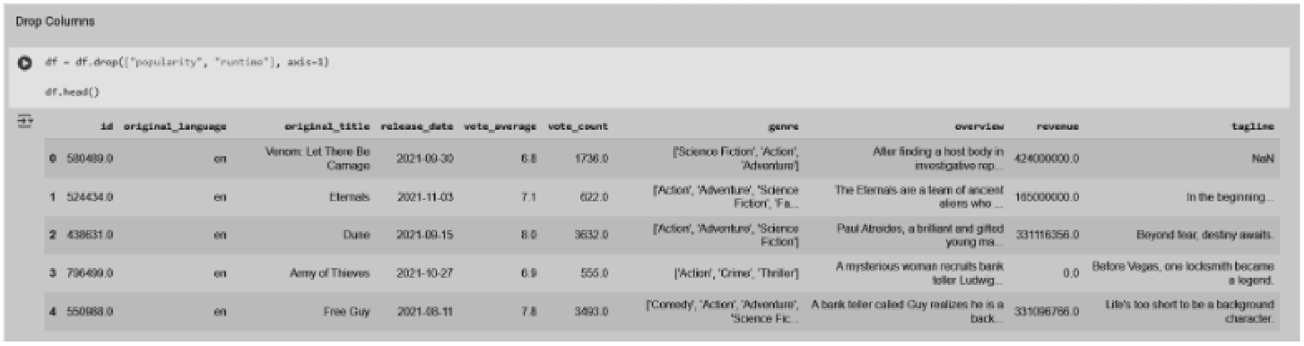
df.describe(): Generates summary statistics for numeric columns, such as:

- count: Number of non-missing values.
- mean: Average value.
- std: Standard deviation.

Name:Atharva Shinde Roll No:54 Div:D15C

- min, 25%, 50% (median), 75%, and max: Percentile values

3) Drop columns that aren’t useful: Columns like Invoice ID may not contribute to analysis (it's often just an identifier). Removing irrelevant columns reduces complexity.



Drop Columns

```
df = df.drop(["popularity", "runtime"], axis=1)

df.head()
```

	id	original_language	original_title	release_date	vote_average	vote_count	genre	overview	revenue	tagline
0	580489.0	en	Venom: Let There Be Carnage	2021-09-30	6.8	1736.0	['Science Fiction', 'Action', 'Adventure']	After finding a host body in investigative rep...	424000000.0	NaN
1	524434.0	en	Eternals	2021-11-03	7.1	622.0	['Action', 'Adventure', 'Science Fiction', 'Fa...	The Eternals are a team of ancient aliens who ...	165000000.0	In the beginning...
2	438631.0	en	Dune	2021-09-15	8.0	3632.0	['Action', 'Adventure', 'Science Fiction']	Paul Atreides, a brilliant and gifted young ma...	331116356.0	Beyond fear, destiny awaits.
3	796499.0	en	Army of Thieves	2021-10-27	6.9	555.0	['Action', 'Crime', 'Thriller']	A mysterious woman recruits bank teller Ludwig...	0.0	Before Vegas, one locksmith became a legend.
4	550988.0	en	Free Guy	2021-08-11	7.8	3493.0	['Comedy', 'Action', 'Adventure', 'Science Fic...	A bank teller called Guy realizes he is a back...	331096766.0	Life's too short to be a background character.

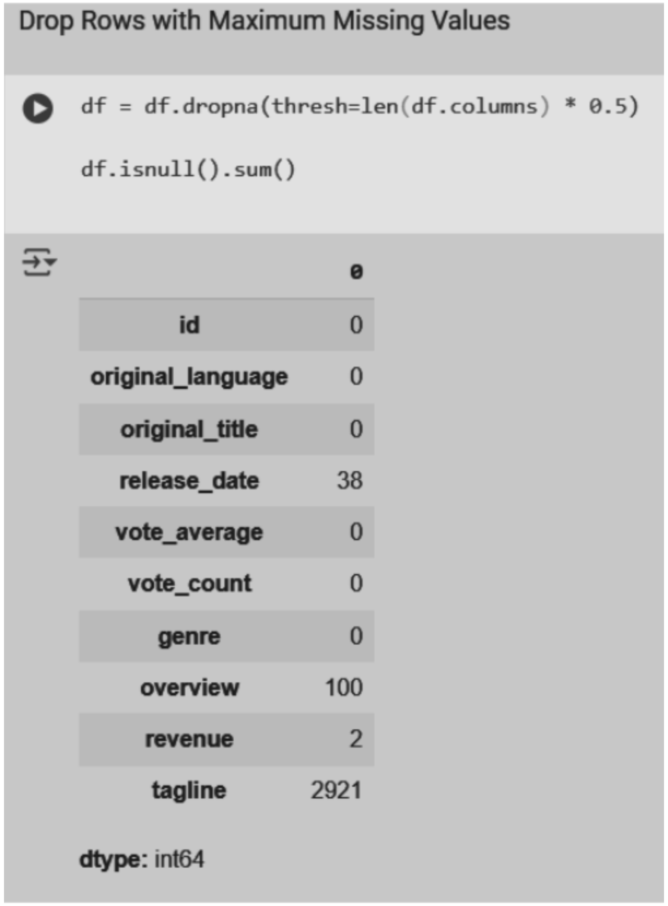
Name:Atharva Shinde Roll No:54 Div:D15C

4) Drop rows with maximum missing values.

df.dropna(thresh=int(0.5 * len(df.columns))):

- Drops rows where more than half the columns have missing (NaN) values.
- thresh=int(0.5 * len(df.columns)): Ensures that a row must have at least 50% non-null values to remain.

df = ...: Updates the DataFrame after dropping rows.
print(df.info()): Confirms that rows with excessive missing values have been removed.



Drop Rows with Maximum Missing Values

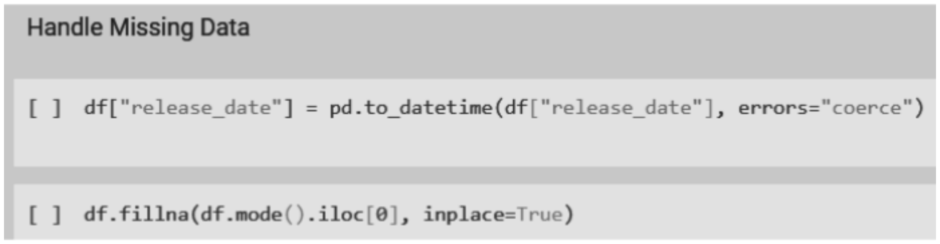
```
df = df.dropna(thresh=len(df.columns) * 0.5)

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

```
                      0
               id     0
original_language     0
   original_title     0
     release_date    38
     vote_average     0
       vote_count     0
            genre     0
         overview   100
          revenue     2
          tagline  2921

dtype: int64
```

5) Take care of missing data.

Name:Atharva Shinde Roll No:54 Div:D15C

df.fillna(df.mean()): Replaces missing values (NaN) in numeric columns with the mean of that column.



Handle Missing Data

```
[ ]  df["release_date"] = pd.to_datetime(df["release_date"], errors="coerce")
```

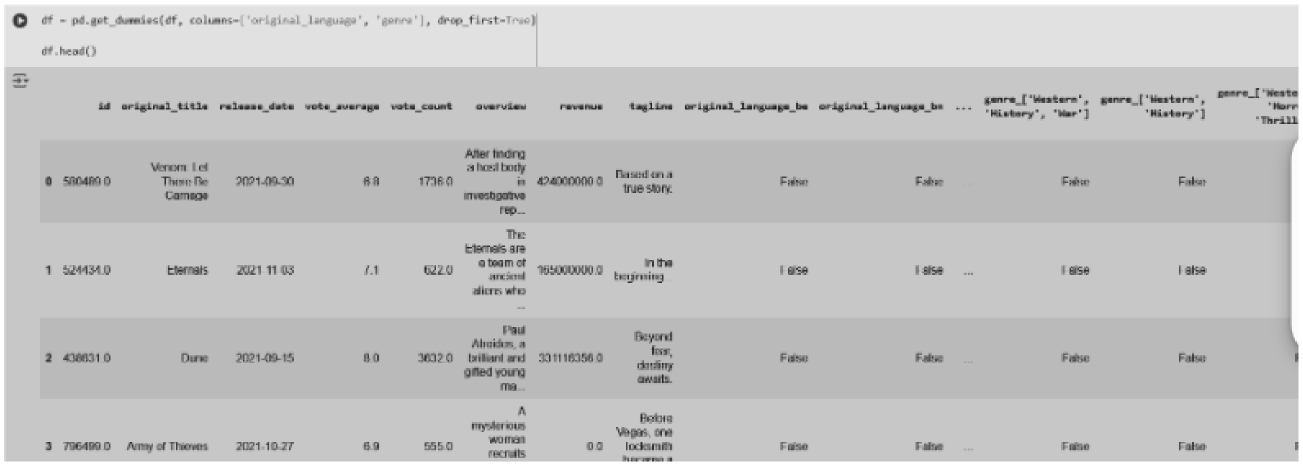
```
[ ]  df.fillna(df.mode().iloc[0], inplace=True)
```

6) Create dummy variables.

pd.get_dummies(): Converts categorical variables into dummy variables (binary indicators: 0 or 1).

columns=['...']: Specifies which columns to convert.
drop_first=True: Avoids the "dummy variable trap" by dropping one dummy variable to prevent multicollinearity.

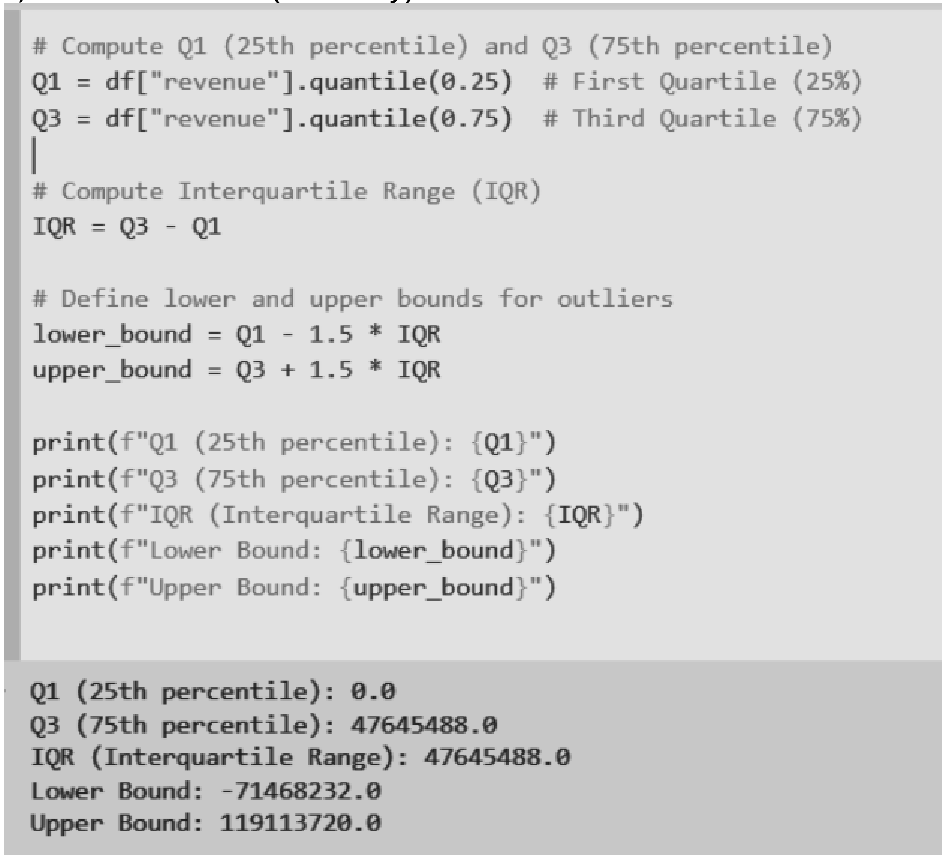
Name:Atharva Shinde Roll No:54 Div:D15C



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

df.head()
```

	id	original_title	release_date	vote_average	vote_count	overview	revenue	tagline	original_language_be	original_language_bn	...	genre_['Western', 'History', 'War']	genre_['Western', 'History']	genre_['Weste 'Horr 'Thrill
0	580489.0	Venom: Let There Be Carnage	2021-09-30	6.8	1736.0	After finding a host body in investigative rep...	424000000.0	Based on a true story.	False	False	...	False	False	
1	524434.0	Eternals	2021-11-03	7.1	622.0	The Eternals are a team of ancient aliens who ...	165000000.0	In the beginning...	False	False	...	False	False	
2	438631.0	Dune	2021-09-15	8.0	3632.0	Paul Atreides, a brilliant and gifted young ma...	331116356.0	Beyond fear, destiny awaits.	False	False	...	False	False	F
3	796499.0	Army of Thieves	2021-10-27	6.9	555.0	A mysterious woman recruits	0.0	Before Vegas, one locksmith	False	False	...	False	False	F

7) Find out outliers (manually)



```
# Compute Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df["revenue"].quantile(0.25)  # First Quartile (25%)
Q3 = df["revenue"].quantile(0.75)  # Third Quartile (75%)

# Compute Interquartile Range (IQR)
IQR = Q3 - Q1

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

print(f"Q1 (25th percentile): {Q1}")
print(f"Q3 (75th percentile): {Q3}")
print(f"IQR (Interquartile Range): {IQR}")
print(f"Lower Bound: {lower_bound}")
print(f"Upper Bound: {upper_bound}")
```

```
Q1 (25th percentile): 0.0
Q3 (75th percentile): 47645488.0
IQR (Interquartile Range): 47645488.0
Lower Bound: -71468232.0
Upper Bound: 119113720.0
```

Name:Atharva Shinde Roll No:54 Div:D15C

8) standardization and normalization of columns

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Standardization equation

$$X' = \frac{X - \mu}{\sigma}$$

To standardize your data, we need to import the StandardScalar from the sklearn library and apply it to our dataset.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Normalization equation

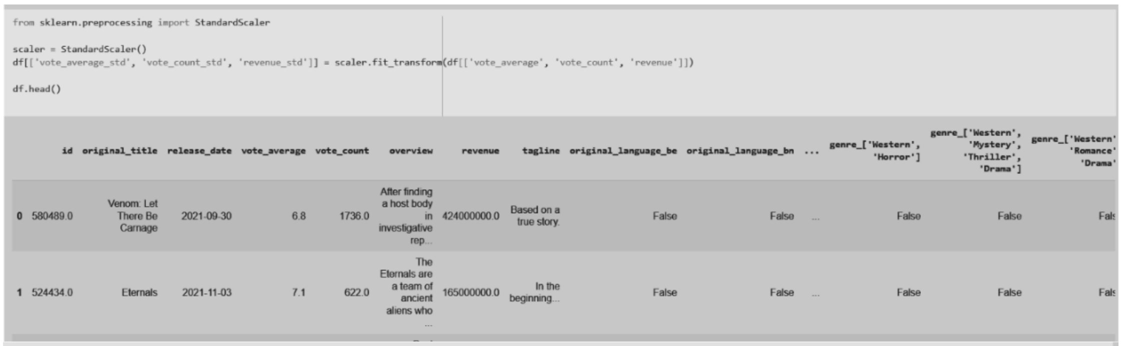
$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

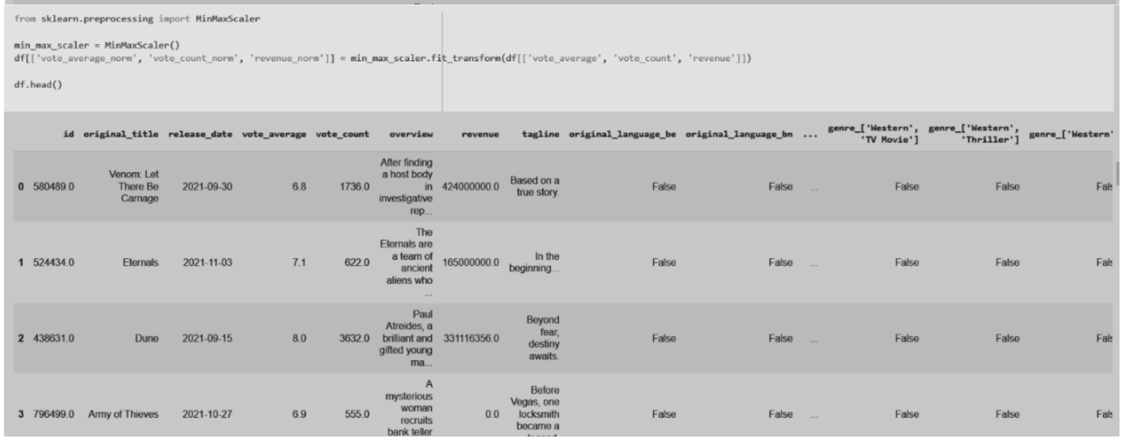
To normalize your data, you need to import the MinMaxScalar from the sklearn library and apply it to our dataset.

Name:Atharva Shinde Roll No:54 Div:D15C



```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['vote_average_std', 'vote_count_std', 'revenue_std']] = scaler.fit_transform(df[['vote_average', 'vote_count', 'revenue']])

df.head()
```

	id	original_title	release_date	vote_average	vote_count	overview	revenue	tagline	original_language_be	original_language_bn	...	genre_['Western', 'Horror']	genre_['Western', 'Mystery', 'Thriller', 'Drama']	genre_['Western' 'Romance' 'Drama'
0	580489.0	Venom: Let There Be Carnage	2021-09-30	6.8	1736.0	After finding a host body in investigative rep...	424000000.0	Based on a true story.	False	False	...	False	False	Fals
1	524434.0	Eternals	2021-11-03	7.1	622.0	The Eternals are a team of ancient aliens who ...	165000000.0	In the beginning...	False	False	...	False	False	Fals



```python
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler()
df[['vote_average_norm', 'vote_count_norm', 'revenue_norm']] = min_max_scaler.fit_transform(df[['vote_average', 'vote_count', 'revenue']])

df.head()
```

	id	original_title	release_date	vote_average	vote_count	overview	revenue	tagline	original_language_be	original_language_bn	...	genre_['Western', 'TV Movie']	genre_['Western', 'Thriller']	genre_['Western'
0	580489.0	Venom: Let There Be Carnage	2021-09-30	6.8	1736.0	After finding a host body in investigative rep...	424000000.0	Based on a true story.	False	False	...	False	False	Fals
1	524434.0	Eternals	2021-11-03	7.1	622.0	The Eternals are a team of ancient aliens who ...	165000000.0	In the beginning...	False	False	...	False	False	Fals
2	438631.0	Dune	2021-09-15	8.0	3632.0	Paul Atreides, a brilliant and gifted young ma...	331116356.0	Beyond fear, destiny awaits.	False	False	...	False	False	Fals
3	796499.0	Army of Thieves	2021-10-27	6.9	555.0	A mysterious woman recruits bank teller	0.0	Before Vegas, one locksmith became a	False	False	...	False	False	Fals

<u>Conclusion</u>:

Thus we have understood how to perform data preprocessing which can further be taken into exploratory data analysis and further in the Model preparation sequence.

Name: Atharva Shinde Div: D15C Roll No:54

EXP 2

Aim:

Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

1. Create bar graph, contingency table using any 2 features.
2. Plot Scatter plot, box plot, Heatmap using seaborn.
3. Create histogram and normalized Histogram.
4. Describe what this graph and table indicates.
5. Handle outlier using box plot and Inter quartile range.

Introduction:

Exploratory Data Analysis (EDA), introduced by John Tukey in the 1970s, is the first step in analyzing datasets to summarize their key characteristics using statistical and visual techniques. It helps understand data patterns, detect anomalies, and prepare the data for machine learning models.

Why Perform EDA?

EDA is essential for:

- Identifying key features and trends in the data.
- Detecting correlations between variables.
- Assessing data quality and handling missing values.
- Determining the need for data preprocessing.
- Communicating insights effectively using visual tools.

Common EDA Techniques:

- Histograms and frequency distributions to analyze data distribution.
- Box plots to identify outliers and data spread.
- Scatter plots to observe relationships between variables.
- Heatmaps to visualize correlations between features.
- Bar charts and pie charts for categorical data analysis

Name: Atharva Shinde Div: D15C Roll No:54

Data visualization is crucial for analyzing the top 10,000 movies dataset because it helps in the following ways:

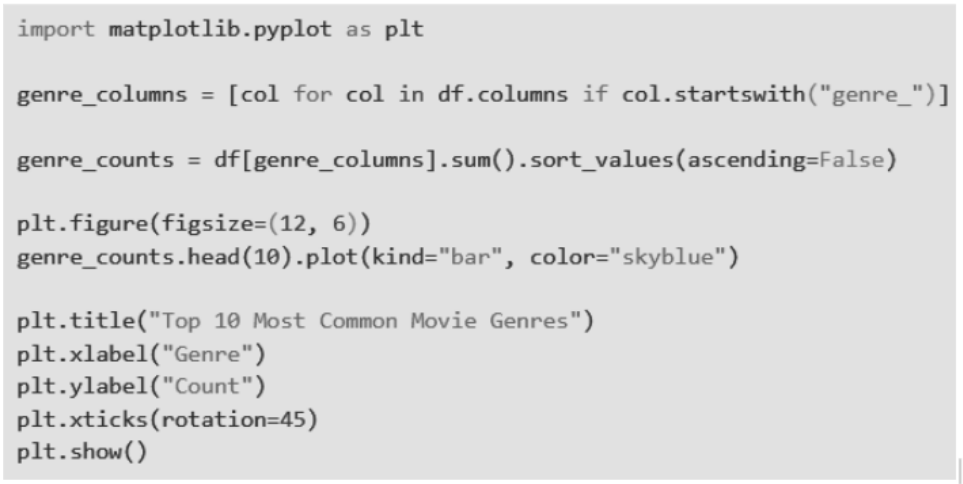
1. **Identifying Trends and Patterns:**
    - Visualizations like line charts and histograms can reveal trends over time, such as the rise or fall in movie production, changing genre popularity, or variations in box office revenues.
2. **Comparative Analysis:**
    - Bar charts and box plots enable comparisons between different categories like genres, production studios, or directors, helping identify which ones are more successful or popular.
3. **Correlation Analysis:**
    - Scatter plots help examine relationships between variables, such as budget vs. revenue or rating vs. box office performance.
4. **Distribution Insights:**
    - Histograms and density plots illustrate the distribution of continuous variables like movie ratings, durations, or revenues, showing if they are normally distributed or skewed.
5. **Outlier Detection:**
    - Box plots or scatter plots can highlight outliers, such as unusually high-grossing movies or extremely low-rated ones, which could be interesting for further investigation.
6. **Audience Preferences and Trends:**
    - Word clouds or bar charts showing the frequency of keywords in movie titles or genres can reveal audience interests and emerging themes.
7. **Data Storytelling:**
    - Interactive dashboards make the data more engaging, helping stakeholders or audiences understand complex information easily.

Name: Atharva Shinde Div: D15C Roll No:54

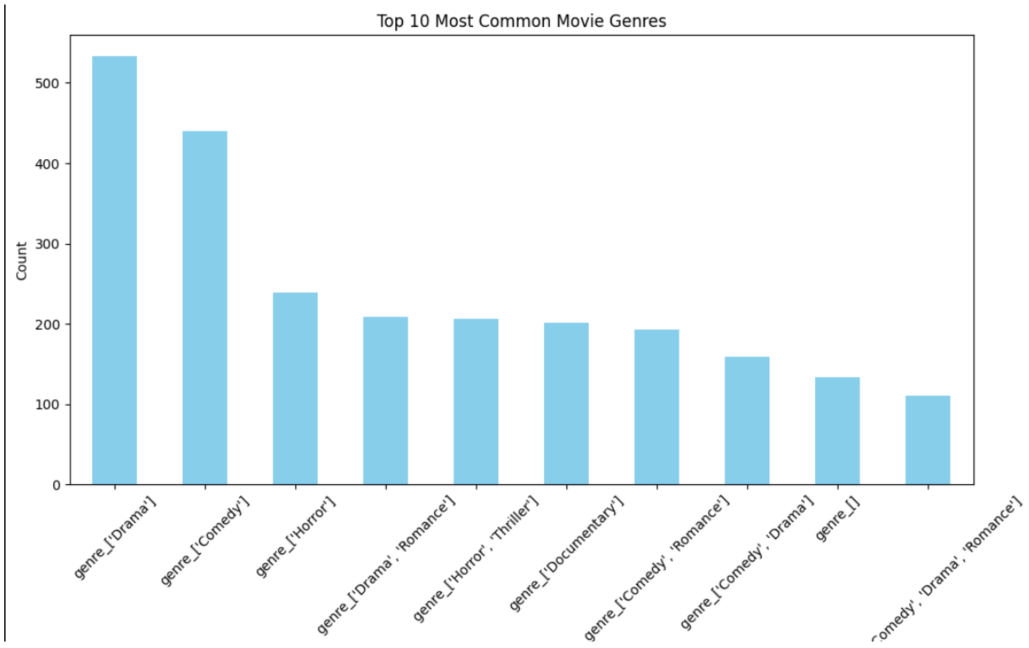
1) Bar Graph (genere vs vote Count)

Inference:

1. Drama and Comedy are the most popular genres, showing high audience interest in storytelling and humor.
2. Hybrid genres like Drama-Romance and Horror-Thriller are also common, reflecting a demand for blended narratives.
3. Empty genre entries indicate possible data inconsistencies that could be cleaned for better analysis.



```
import matplotlib.pyplot as plt

genre_columns = [col for col in df.columns if col.startswith("genre_")]

genre_counts = df[genre_columns].sum().sort_values(ascending=False)

plt.figure(figsize=(12, 6))
genre_counts.head(10).plot(kind="bar", color="skyblue")

plt.title("Top 10 Most Common Movie Genres")
plt.xlabel("Genre")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



Name: Atharva Shinde Div: D15C Roll No:54

2)Inference: Box Plot

1. Highly Skewed Distribution:

- The revenue data is highly skewed to the right, with the majority of movies earning relatively low revenue. This is indicated by the box being compressed near the lower end of the axis.
- A small number of movies have extremely high revenues, as shown by the large number of outliers. These outliers represent blockbuster hits that earned significantly more than the average movie.

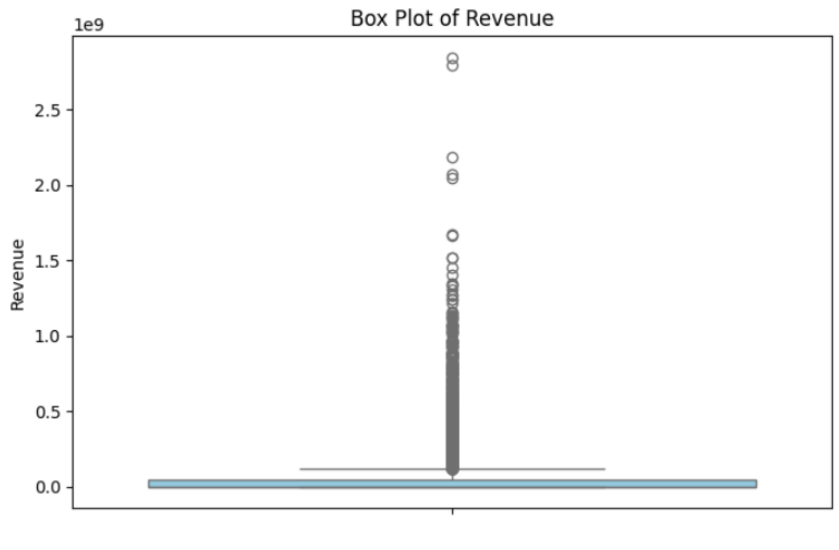
2. Outliers and Blockbusters:

- Numerous outliers are present above the upper whisker, extending beyond $1 billion, and a few even surpass $2.5 billion. These represent major box office successes, likely including franchises or highly anticipated releases.
- The concentration of outliers suggests that while most movies earn modest amounts, a few exceptional hits dominate total box office revenue.

3. Central Tendency and Spread:

- The interquartile range (IQR) is narrow and positioned near the lower end, highlighting that 50% of the movies have relatively low earnings.
- The median (central line within the box) is close to the bottom, confirming that half of the movies earn less than the average revenue, reinforcing the right-skewed distribution.

```
plt.figure(figsize=(8, 5))
sns.boxplot(y=df["revenue"], color="skyblue")

plt.title("Box Plot of Revenue")
plt.ylabel("Revenue")
plt.show()
```



5) Heatmap of Numerical Features Correlation

Purpose:

This heatmap visually represents the correlation between numerical features in the movies dataset. The values range from -1 to 1, where:

- +1 → Strong positive correlation (when one factor increases, the other also increases).
- 0 → No correlation (factors do not impact each other).
- -1 → Strong negative correlation (when one factor increases, the other decreases).

Key Observations from the movies Dataset:

Strong Positive Correlation:

- **Revenue** and **Vote Count** have a high positive correlation (0.77), indicating that movies with more votes typically generate higher revenue. This suggests popular movies (in terms of audience engagement) tend to be more profitable.
- **Revenue** also strongly correlates with its normalized and standardized versions, as expected.

Moderate Positive Correlation:

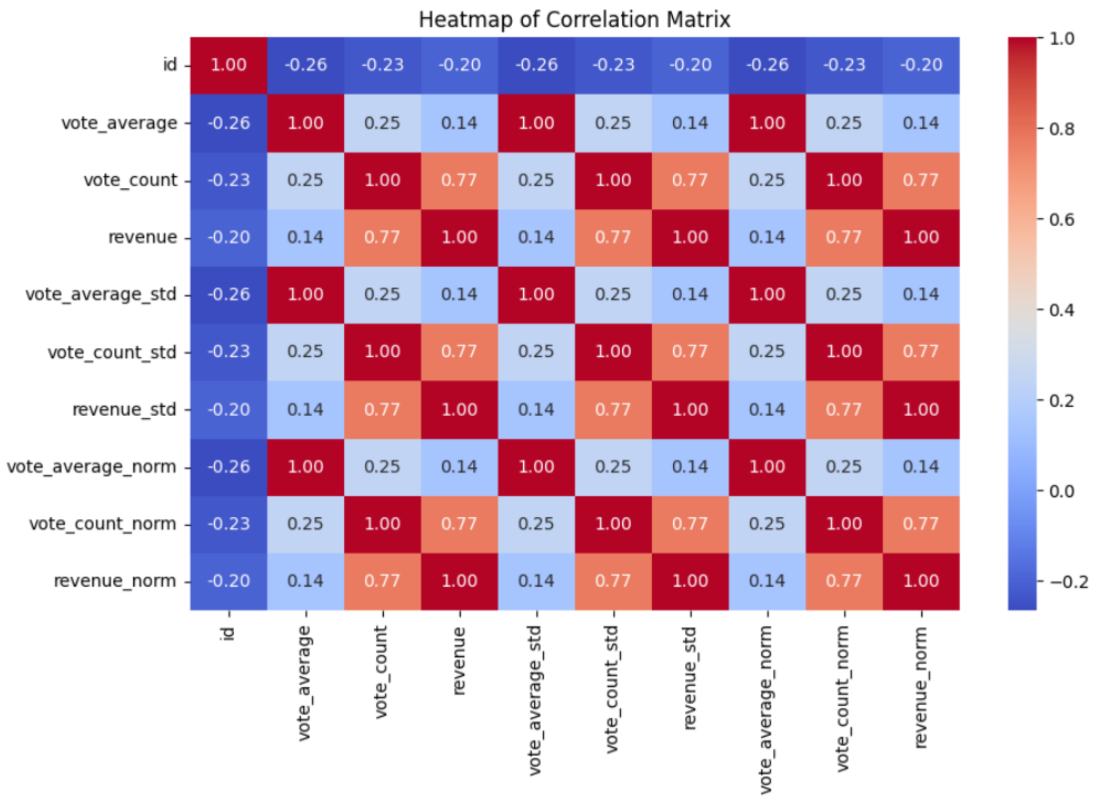
- **Vote Count** and **Vote Average** show a moderate positive correlation (0.25), implying that higher audience engagement slightly influences better ratings.
- **Revenue** and **Vote Average** have a low positive correlation (0.14), suggesting that higher-rated movies might earn more, but the relationship is not very strong.

Low or Negative Correlation:

- **ID** shows negative correlations with most variables, which is normal since it’s just an identifier and has no direct relationship with movie attributes.
- **Vote Average** has a weak correlation with **Revenue** and **Vote Count**, indicating that high ratings don’t necessarily translate to high earnings or more votes

Name: Atharva Shinde Div: D15C Roll No:54

```
import numpy as np

numeric_columns = df.select_dtypes(include=np.number)

plt.figure(figsize=(10, 6))
sns.heatmap(numeric_columns.corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Heatmap of Correlation Matrix")
plt.show()
```



	id	vote_average	vote_count	revenue	vote_average_std	vote_count_std	revenue_std	vote_average_norm	vote_count_norm	revenue_norm
id	1.00	-0.26	-0.23	-0.20	-0.26	-0.23	-0.20	-0.26	-0.23	-0.20
vote_average	-0.26	1.00	0.25	0.14	1.00	0.25	0.14	1.00	0.25	0.14
vote_count	-0.23	0.25	1.00	0.77	0.25	1.00	0.77	0.25	1.00	0.77
revenue	-0.20	0.14	0.77	1.00	0.14	0.77	1.00	0.14	0.77	1.00
vote_average_std	-0.26	1.00	0.25	0.14	1.00	0.25	0.14	1.00	0.25	0.14
vote_count_std	-0.23	0.25	1.00	0.77	0.25	1.00	0.77	0.25	1.00	0.77
revenue_std	-0.20	0.14	0.77	1.00	0.14	0.77	1.00	0.14	0.77	1.00
vote_average_norm	-0.26	1.00	0.25	0.14	1.00	0.25	0.14	1.00	0.25	0.14
vote_count_norm	-0.23	0.25	1.00	0.77	0.25	1.00	0.77	0.25	1.00	0.77
revenue_norm	-0.20	0.14	0.77	1.00	0.14	0.77	1.00	0.14	0.77	1.00

Name: Atharva Shinde Div: D15C Roll No:54

6) Histogram

Inference (From Histogram)

Highly Skewed Distribution:

- The histogram shows a **right-skewed distribution**, indicating that most movies earn relatively low revenue, while a few outliers make extremely high amounts.

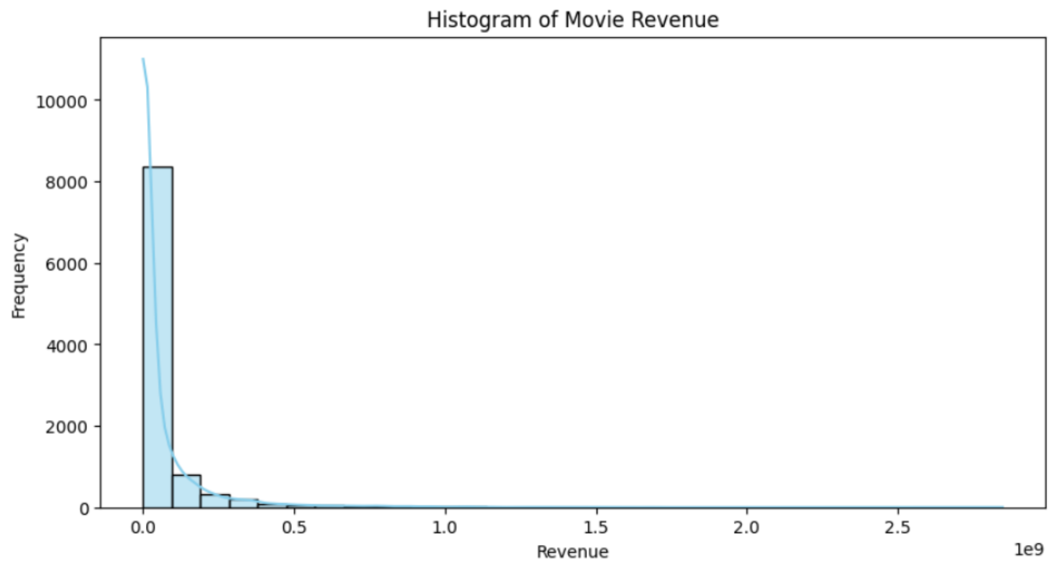
Majority in Low Revenue Range:

- A significant majority of movies have revenues clustered near zero, suggesting that high-grossing films are rare compared to lower-earning ones.

Presence of Outliers:

- A long tail extends towards the right, showing the presence of blockbuster movies with revenues exceeding **$1 billion**, which significantly skew the distribution.

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

plt.figure(figsize=(10, 5))
sns.histplot(df["revenue"], bins=30, kde=True, color="skyblue")

plt.title("Histogram of Movie Revenue")
plt.xlabel("Revenue")
plt.ylabel("Frequency")
plt.show()
```



Name: Atharva Shinde Div: D15C Roll No:54

7) Normalized Histogram

Right-Skewed Distribution Remains:

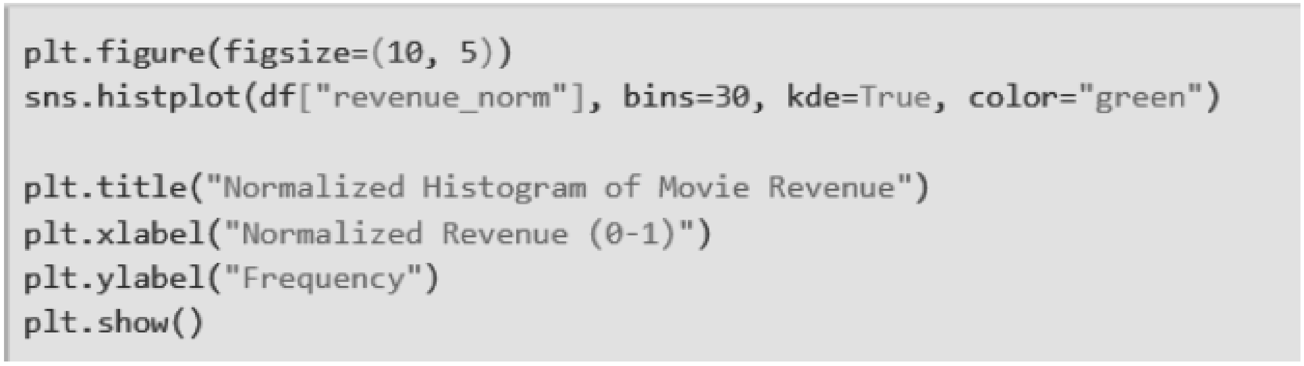
- Even after normalization, the distribution is highly right-skewed, indicating that most movies have relatively low normalized revenue, with a few outliers achieving significantly higher values.

Concentration Near Zero:

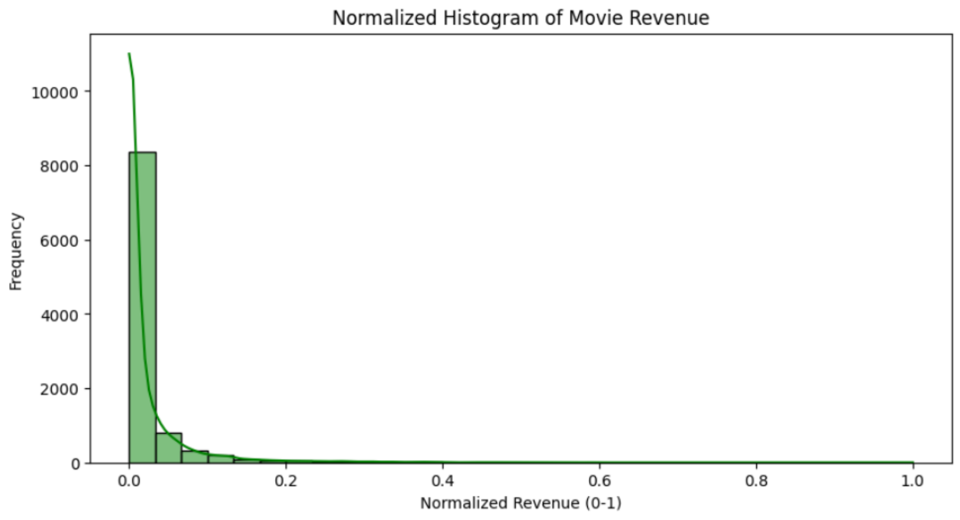
- The majority of movies are concentrated near the lower end of the normalized scale (close to 0), showing that most films earn a small fraction of the maximum revenue observed.

Outliers Still Present:

- A long tail persists towards the right, suggesting that a few movies generate disproportionately high revenue even when scaled between 0 and 1.



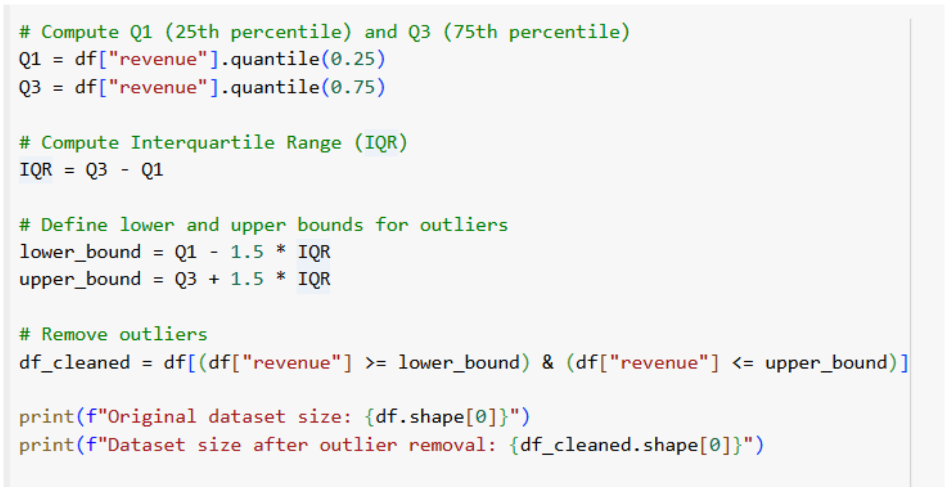
```
plt.figure(figsize=(10, 5))
sns.histplot(df["revenue_norm"], bins=30, kde=True, color="green")

plt.title("Normalized Histogram of Movie Revenue")
plt.xlabel("Normalized Revenue (0-1)")
plt.ylabel("Frequency")
plt.show()
```



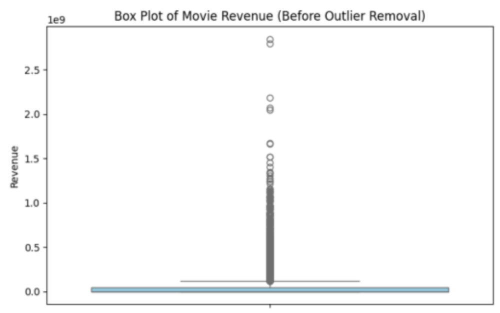
Name: Atharva Shinde Div: D15C Roll No:54

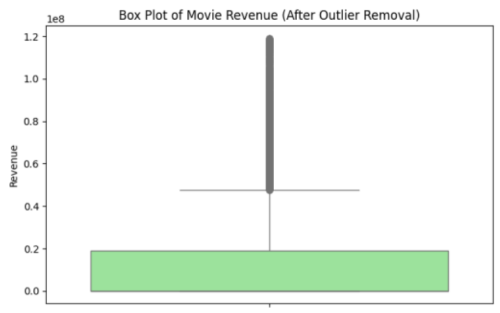
8) **Handle outlier using box plot**

Inference: Box Plot



```python
# Compute Q1 (25th percentile) and Q3 (75th percentile)
Q1 = df["revenue"].quantile(0.25)
Q3 = df["revenue"].quantile(0.75)

# Compute Interquartile Range (IQR)
IQR = Q3 - Q1

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

# Remove outliers
df_cleaned = df[(df["revenue"] >= lower_bound) & (df["revenue"] <= upper_bound)]

print(f"Original dataset size: {df.shape[0]}")
print(f"Dataset size after outlier removal: {df_cleaned.shape[0]}")
```





Conclusion:

Hence we learned about exploratory data analysis and various types of statistical measures of data along with correlation. We also learnt about visualization and applied these concepts with hands-on experience on our chosen dataset.

NAME: Atharva Shinde CLASS: D15C ROLL_NO.: 54

Aim: **Perform Data Modelling – Partitioning the dataset.**

Theory:

Importance of data Partitioning.

Partitioning data into **train** and **test** splits is a fundamental practice in machine learning and statistical modeling. This division is crucial for ensuring that models generalize well to unseen data and do not overfit to the training dataset. Below is a detailed explanation of why this partitioning is important:

1. Evaluation of Model Generalization

- **Purpose**: The primary goal of machine learning is to build models that perform well on **unseen data**, not just the data they were trained on. Partitioning the data into train and test sets allows us to simulate this scenario.
- **Mechanism**: The **training set** is used to train the model, while the **test set** acts as a proxy for unseen data. By evaluating the model on the test set, we can estimate how well the model is likely to perform on new, real-world data.
- **Risk of Not Partitioning**: Without a separate test set, we risk overestimating the model's performance because the model may simply memorize the training data (overfitting) rather than learning generalizable patterns.

2. Avoiding Optimistic Bias

- **Optimistic Bias**: If the same data is used for both training and evaluation, the model's performance metrics (e.g., accuracy, precision, recall) will be overly optimistic. This is because the model has already "seen" the data and may have memorized it.
- **Test Set as a Safeguard**: The test set acts as a safeguard against this bias, providing a more realistic measure of the model's performance.

3. Detection of Overfitting

- **Overfitting Definition**: Overfitting occurs when a model learns the noise or specific details of the training data, leading to poor performance on new data.

- **Role of Test Set**: The test set provides an independent evaluation of the model. If the model performs well on the training set but poorly on the test set, it is a clear indication of overfitting.
- **Example**: A model achieving 99% accuracy on the training set but only 60% on the test set suggests that it has overfitted to the training data.

Visual Representation

Using a bar graph to visualize a 75:25 train-test split is an effective way to clearly communicate the distribution of the dataset. The graph provides an immediate visual representation of the proportions, making it easy to see that 75% of the data is allocated for training and 25% for testing. This clarity ensures that the split is transparent and well-understood, which is crucial for validating the model's development process.

Additionally, the bar graph highlights whether the split is balanced and appropriate for the task at hand. A 75:25 ratio is a common and practical division, and visualizing it helps confirm that the test set is large enough to provide a reliable evaluation of the model's performance. This visual justification reinforces the credibility of our data preparation and modeling approach.

Z-Testing:

Key Idea: **Fair Evaluation, Partitioning Issues.**

The two-sample Z-test is a statistical hypothesis test used to determine whether the means of two independent samples are significantly different from each other. It assumes that the data follows a normal distribution and that the population variances are known (or the sample sizes are large enough for the Central Limit Theorem to apply). The test calculates a Z-score, which measures how many standard deviations the difference between the sample means lies from zero. This score is then compared to a critical value or used to compute a p-value to determine statistical significance.

The primary use case of the Z-test is to compare the means of two groups and assess whether any observed difference is due to random chance or a true underlying difference. In the context of dataset partitioning, the Z-test can be used to validate whether the train and test splits are statistically similar. For example, by comparing the means of a key feature (e.g., age, income) across the two splits, we can ensure that the partitioning process did not introduce bias and that both sets are representative of the same population.

NAME: Atharva Shinde CLASS: D15C ROLL_NO.: 54

The significance of the Z-test lies in its ability to provide a quantitative measure of similarity between datasets. If the p-value is greater than the chosen significance level (e.g., 0.05), we can conclude that the splits are statistically similar, ensuring a fair and reliable evaluation of the model. This step is crucial for maintaining the integrity of the machine learning workflow and ensuring that the model's performance metrics are trustworthy.

Steps:

Imported train_test_split **from** sklearn.model_selection**:**

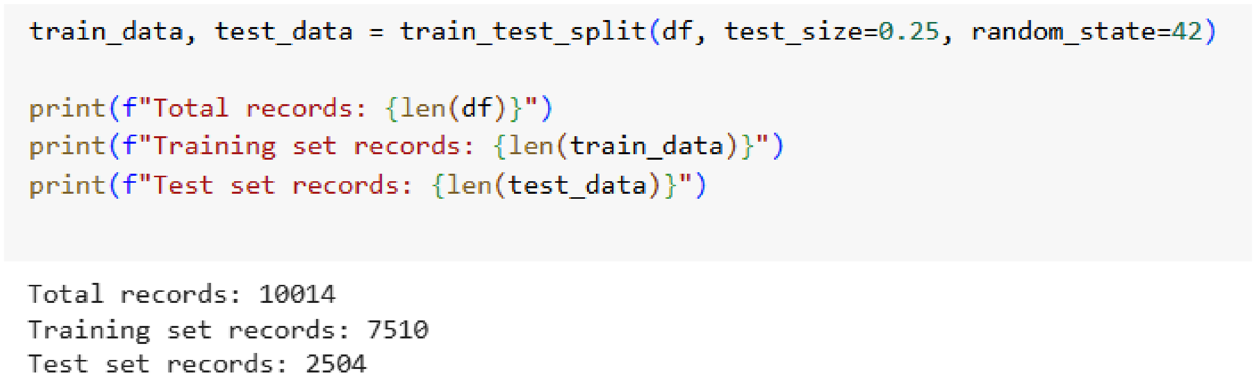
- This function is used to split arrays or matrices into random train and test subsets.

Split Features and Target Variable:

- **Features (**X**):** We created a dataframe X by dropping the 'Total' column from df. This dataframe contains all the feature variables except the target.
- **Target (**y**):** We created a series y which contains the 'Total' column from df. This series is our target variable.

Partitioned the Data:

- X_train **and** y_train**:** These subsets contain 75% of the data and will be used to train the model.
- X_test **and** y_test**:** These subsets contain the remaining 25% of the data and will be used to test the model's performance.



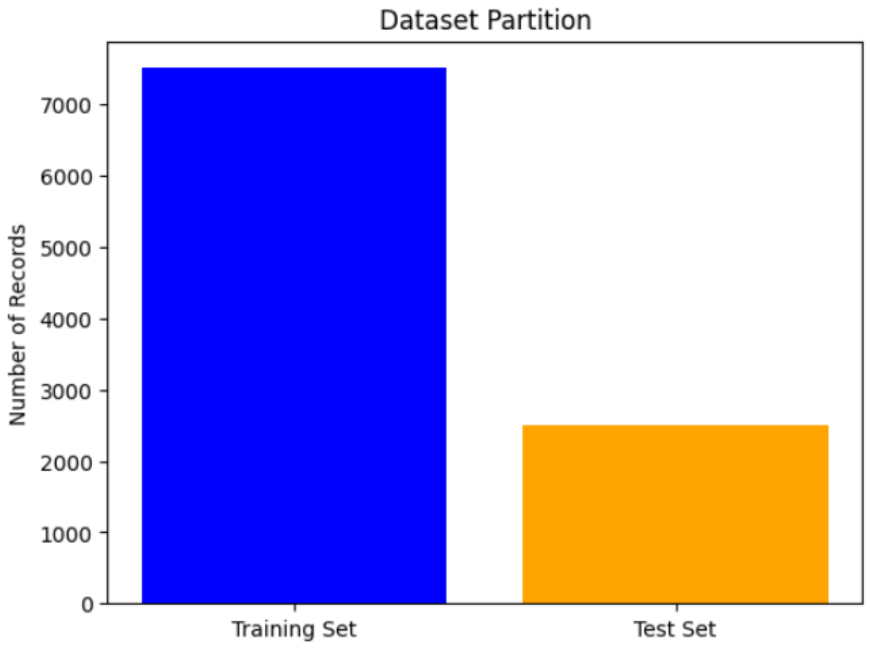
```
train_data, test_data = train_test_split(df, test_size=0.25, random_state=42)

print(f"Total records: {len(df)}")
print(f"Training set records: {len(train_data)}")
print(f"Test set records: {len(test_data)}")
```

```
Total records: 10014
Training set records: 7510
Test set records: 2504
```

Visualizing the split.

- **plt.bar(labels, sizes, color=['blue', 'orange']): This function creates a bar graph with the specified labels and sizes. The bars are colored blue for training data and orange for test data.**
- **plt.title('Proportion of Training and Test Data (Features & Target)'): This sets the title of the graph.**
- **plt.ylabel('Number of Samples'): This sets the label for the y-axis, indicating the number of samples.**
- **plt.show(): This function displays the graph.**

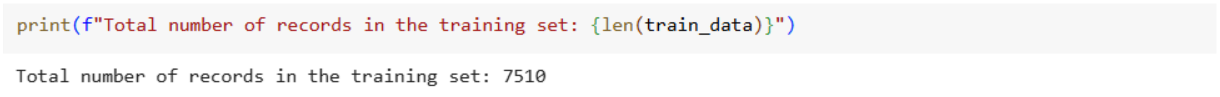
```
proportions = [len(train_data), len(test_data)]
labels = ['Training Set', 'Test Set']

plt.bar(labels, proportions, color=['blue', 'orange'])
plt.title('Dataset Partition')
plt.ylabel('Number of Records')
plt.show()
```



NAME: Atharva Shinde CLASS: D15C ROLL_NO.: 54

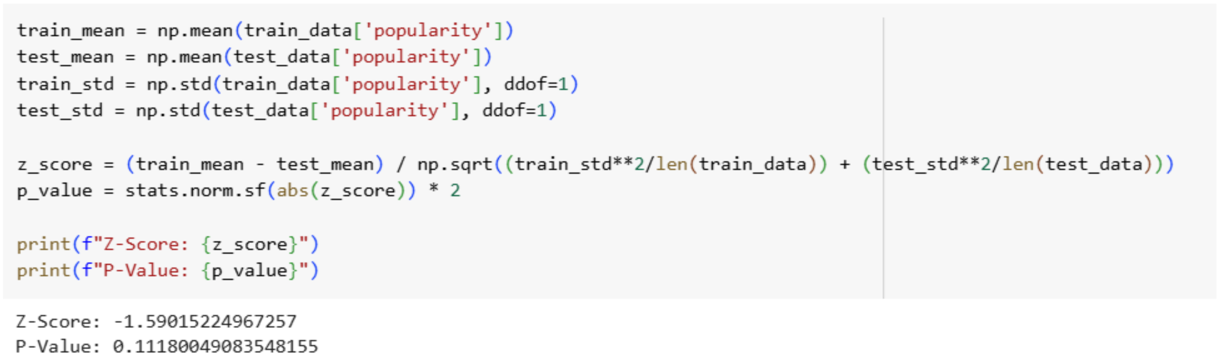
Significance of the Output:

- **Z-Statistic:**
    - Indicates the number of standard deviations by which the mean of the training set differs from the mean of the test set.
- **P-Value:**
    - Helps determine the significance of the Z-statistic. A low P-value (< 0.05) suggests that the difference is statistically significant.



```
print(f"Total number of records in the training set: {len(train_data)}")
```

```
Total number of records in the training set: 7510
```



```
train_mean = np.mean(train_data['popularity'])
test_mean = np.mean(test_data['popularity'])
train_std = np.std(train_data['popularity'], ddof=1)
test_std = np.std(test_data['popularity'], ddof=1)

z_score = (train_mean - test_mean) / np.sqrt((train_std**2/len(train_data)) + (test_std**2/len(test_data)))
p_value = stats.norm.sf(abs(z_score)) * 2

print(f"Z-Score: {z_score}")
print(f"P-Value: {p_value}")
```

```
Z-Score: -1.59015224967257
P-Value: 0.11180049083548155
```

Inference from the Output:

- **Interpretation:**

In this analysis, the P-Value obtained is 0.1118, which is greater than the significance level of 0.05. This indicates that there is no statistically significant difference between the training and test sets. Therefore, it can be inferred that both sets are likely drawn from the same distribution, ensuring that the model will be evaluated on data that is representative of what it was trained on.

This is an scenario for building a machine learning model, as it reduces the risk of biased performance metrics and improves the model's generalization ability. The consistent distribution between the training and test sets confirms the effectiveness of the data-splitting strategy used in this study.

NAME: Atharva Shinde CLASS: D15C ROLL_NO.: 54

Conclusion:

In this experiment, we successfully partitioned the dataset into **training and test sets** using a 75:25 split ratio, ensuring a robust foundation for model development and evaluation. The partitioning was visualized using a bar graph, which clearly illustrated the proportion of data allocated to each set, confirming that the split was appropriately balanced.

The dataset was divided into a training set (75%) and a test set (25%), resulting in 7,510 records for model training and 2,504 for model evaluation. A statistical hypothesis test was performed to compare the means of the popularity attribute in the training and test sets. The resulting Z-Score of -1.59 and P-Value of 0.1118 indicate no statistically significant difference between the two groups, confirming that the split maintained the population distribution's integrity. This suggests that the dataset is well-shuffled and that the training and test sets are representative of the overall data distribution.

Name:Atharva Shinde Roll no:54 Div:D15C

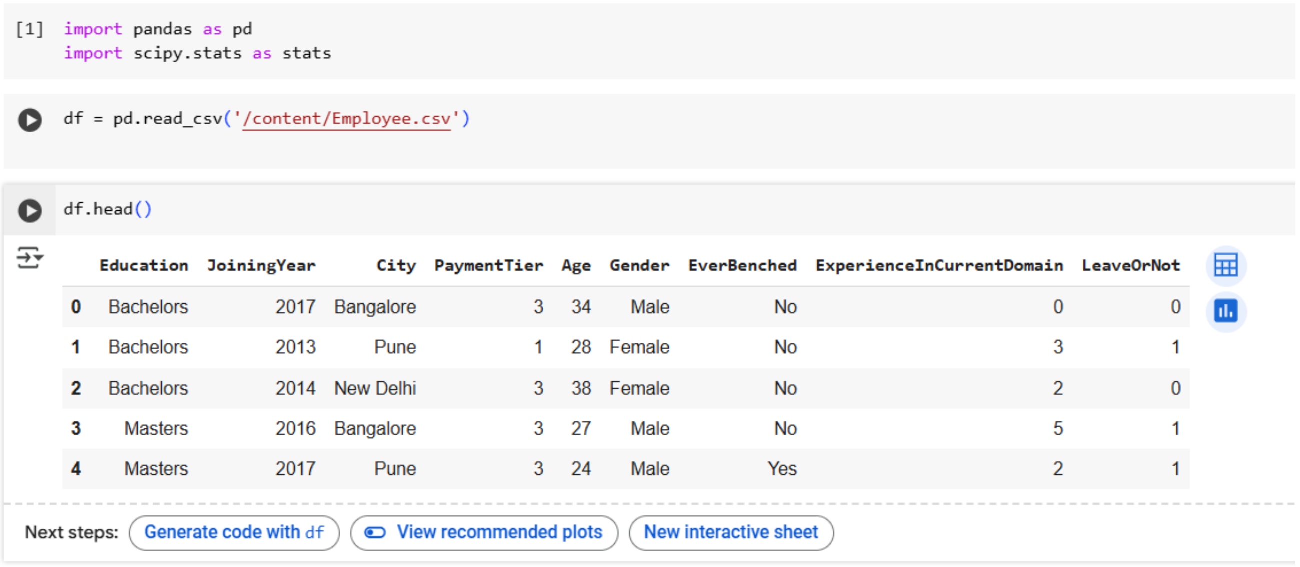
Exp 4

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

Theory and Output:

1. Loading dataset:

Data loading is the first step in data analysis. The dataset is stored in a CSV file and read using `pandas.read_csv()`.
The first few rows are displayed to understand the dataset structure



```
[1] import pandas as pd
    import scipy.stats as stats
```

```
df = pd.read_csv('/content/Employee.csv')
```

```
df.head()
```

	Education	JoiningYear	City	PaymentTier	Age	Gender	EverBenched	ExperienceInCurrentDomain	LeaveOrNot
0	Bachelors	2017	Bangalore	3	34	Male	No	0	0
1	Bachelors	2013	Pune	1	28	Female	No	3	1
2	Bachelors	2014	New Delhi	3	38	Female	No	2	0
3	Masters	2016	Bangalore	3	27	Male	No	5	1
4	Masters	2017	Pune	3	24	Male	Yes	2	1

Name:Atharva Shinde Roll no:54 Div:D15C

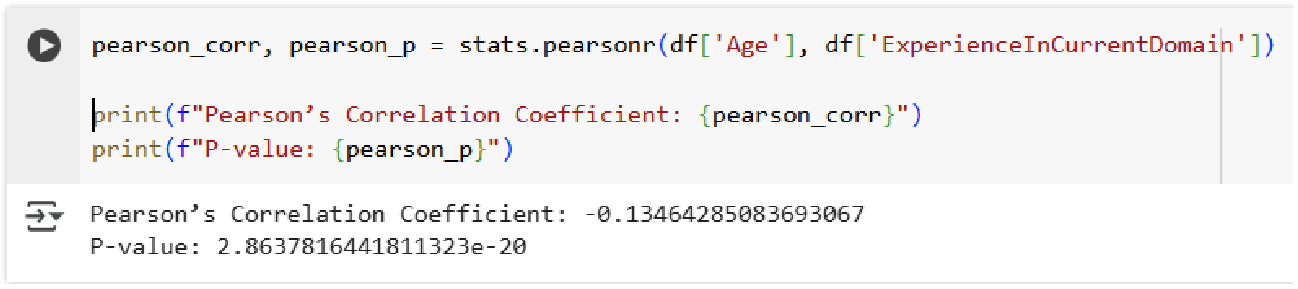
2. Pearson’s Correlation Coefficient:

Pearson’s Correlation Coefficient (denoted as **r**) measures the **linear** relationship between two continuous variables.
Values range from **-1 to +1**:

- **+1**: Perfect positive correlation
- **0**: No correlation
- **-1**: Perfect negative correlation

The formula for Pearson’s Correlation Coefficient is:

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$



```
pearson_corr, pearson_p = stats.pearsonr(df['Age'], df['ExperienceInCurrentDomain'])

print(f"Pearson’s Correlation Coefficient: {pearson_corr}")
print(f"P-value: {pearson_p}")
```

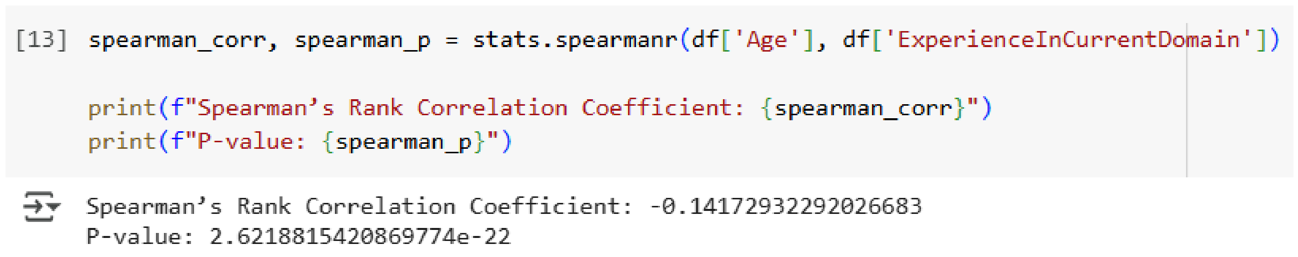
```
Pearson’s Correlation Coefficient: -0.13464285083693067
P-value: 2.8637816441811323e-20
```

Name:Atharva Shinde Roll no:54 Div:D15C

3. Spearman's Rank Correlation

- Spearman's Rank Correlation (denoted as ρ, rho) measures the monotonic relationship between two variables.
- It does not require normally distributed data.
- If ranks of two variables are related, it indicates correlation.
- The formula is:

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



```
[13] spearman_corr, spearman_p = stats.spearmanr(df['Age'], df['ExperienceInCurrentDomain'])

     print(f"Spearman's Rank Correlation Coefficient: {spearman_corr}")
     print(f"P-value: {spearman_p}")
```

```
Spearman's Rank Correlation Coefficient: -0.14172932292026683
P-value: 2.6218815420869774e-22
```

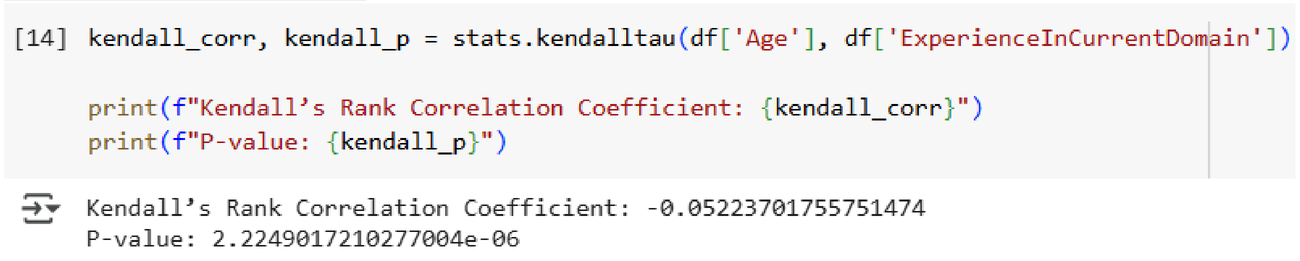
Name:Atharva Shinde Roll no:54 Div:D15C

4. Kendall’s Rank Correlation

Theory:

- Kendall’s Tau (τ) measures the **ordinal association** between two variables.
- It counts **concordant** and **discordant** pairs:
  - **Concordant pairs**: If one variable increases, the other also increases.
  - **Discordant pairs**: One increases while the other decreases.
- The formula is:

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



```
[14] kendall_corr, kendall_p = stats.kendalltau(df['Age'], df['ExperienceInCurrentDomain'])

     print(f"Kendall’s Rank Correlation Coefficient: {kendall_corr}")
     print(f"P-value: {kendall_p}")
```

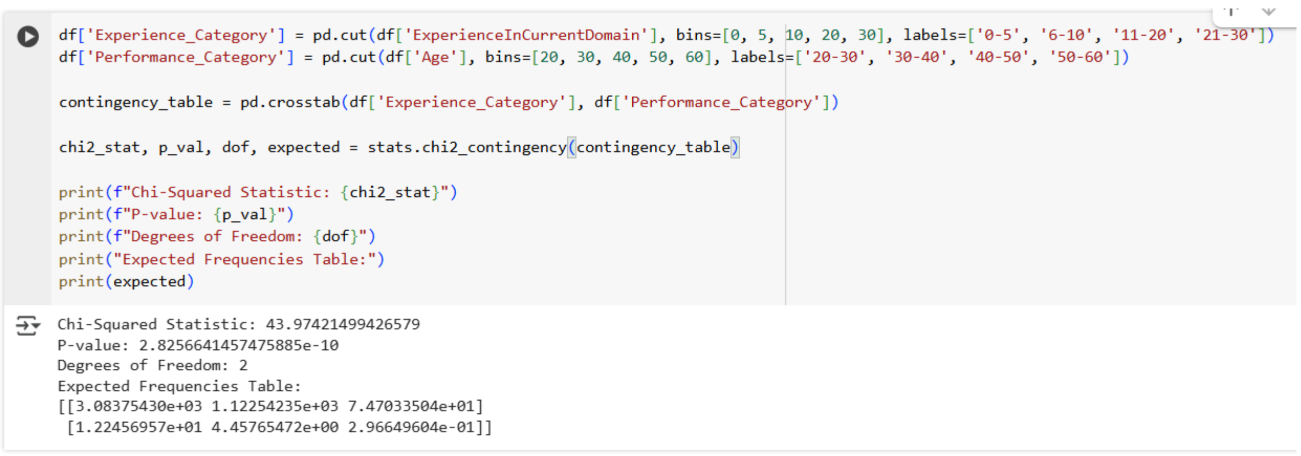
```
Kendall’s Rank Correlation Coefficient: -0.05223701755751474
P-value: 2.2249017210277004e-06
```

Name:Atharva Shinde Roll no:54 Div:D15C

5. Chi-Squared Test

- The **Chi-Squared Test** is used for **categorical data** to check if two variables are independent.
- It compares **observed** and **expected** frequencies.
- The formula is:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$



```python
df['Experience_Category'] = pd.cut(df['ExperienceInCurrentDomain'], bins=[0, 5, 10, 20, 30], labels=['0-5', '6-10', '11-20', '21-30'])
df['Performance_Category'] = pd.cut(df['Age'], bins=[20, 30, 40, 50, 60], labels=['20-30', '30-40', '40-50', '50-60'])

contingency_table = pd.crosstab(df['Experience_Category'], df['Performance_Category'])

chi2_stat, p_val, dof, expected = stats.chi2_contingency(contingency_table)

print(f"Chi-Squared Statistic: {chi2_stat}")
print(f"P-value: {p_val}")
print(f"Degrees of Freedom: {dof}")
print("Expected Frequencies Table:")
print(expected)
```

```
Chi-Squared Statistic: 43.97421499426579
P-value: 2.8256641457475885e-10
Degrees of Freedom: 2
Expected Frequencies Table:
[[3.08375430e+03 1.12254235e+03 7.47033504e+01]
 [1.22456957e+01 4.45765472e+00 2.96649604e-01]]
```

Name:Atharva Shinde Roll no:54 Div:D15C

Conclusion

1. **Pearson's Correlation**: Measures **linear relationship** between numerical variables. If **$p < 0.05$**, the correlation is significant.
2. **Spearman's Correlation**: Checks for **monotonic relationship**. If **$p < 0.05$**, variables move together in a ranked order.
3. **Kendall's Correlation**: Identifies **ordinal association**. A small **p-value** means a strong relationship.
4. **Chi-Square Test**: Determines **independence of categorical variables**. If **$p < 0.05$**, variables are dependent; otherwise, they are independent.

Final Summary:

- If **$p < 0.05$**, the test indicates a significant relationship.
- If **$p > 0.05$**, no strong relationship exists.

These tests help understand **associations** in the dataset for data-driven decisions.

NAME: Atharva Shinde DIV: D15C ROLL NO: 54

Experiment No. - 5

Aim: Perform Regression Analysis using Scipy and Scikit-learn.

Problem Statement:

1. Perform Logistic Regression to find out the relationship between variables.
2. Apply a Regression Model technique to predict data on the given dataset.

Logistic Regression

Logistic Regression is a widely used statistical method for analyzing and modeling relationships between a dependent variable and one or more independent variables. Unlike Linear Regression, which is used for continuous outcomes, Logistic Regression is applied when the target variable is categorical, typically binary (0 or 1).

It uses the logistic (sigmoid) function to estimate probabilities, making it suitable for classification tasks. In this implementation, we explore the relationship between variables using logistic regression to understand their influence on the target variable. Additionally, we apply this model to predict outcomes based on the dataset, leveraging techniques like model fitting, evaluation metrics, and performance assessment to validate the predictions.

Dataset: NYC Taxi

The dataset used in this experiment is related to **New York City taxi trips**. The goal is to analyze various trip-related factors such as trip duration, pickup and drop-off locations, passenger count, and vendor details. This dataset is useful for transportation analytics, trip duration prediction, and spatial-temporal analysis.

The dataset contains the following columns:

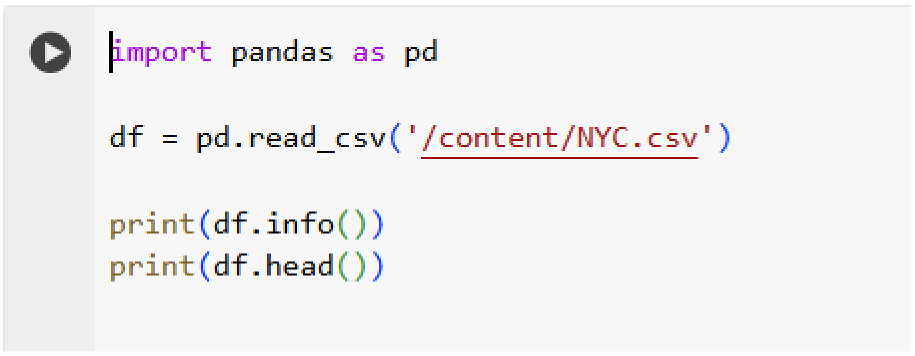
- **id**: Unique identifier for each trip
- **vendor_id**: Code indicating the provider associated with the trip record
- **pickup_datetime**: Date and time when the trip started
- **dropoff_datetime**: Date and time when the trip ended
- **passenger_count**: Number of passengers in the vehicle
- **pickup_longitude**: Longitude at the pickup point
- **pickup_latitude**: Latitude at the pickup point
- **dropoff_longitude**: Longitude at the drop-off point
- **dropoff_latitude**: Latitude at the drop-off point
- **store_and_fwd_flag**: Whether the trip record was stored before forwarding (Y/N)
- **trip_duration**: Duration of the trip in seconds (Target variable)

NAME: Atharva Shinde DIV: D15C ROLL NO: 54

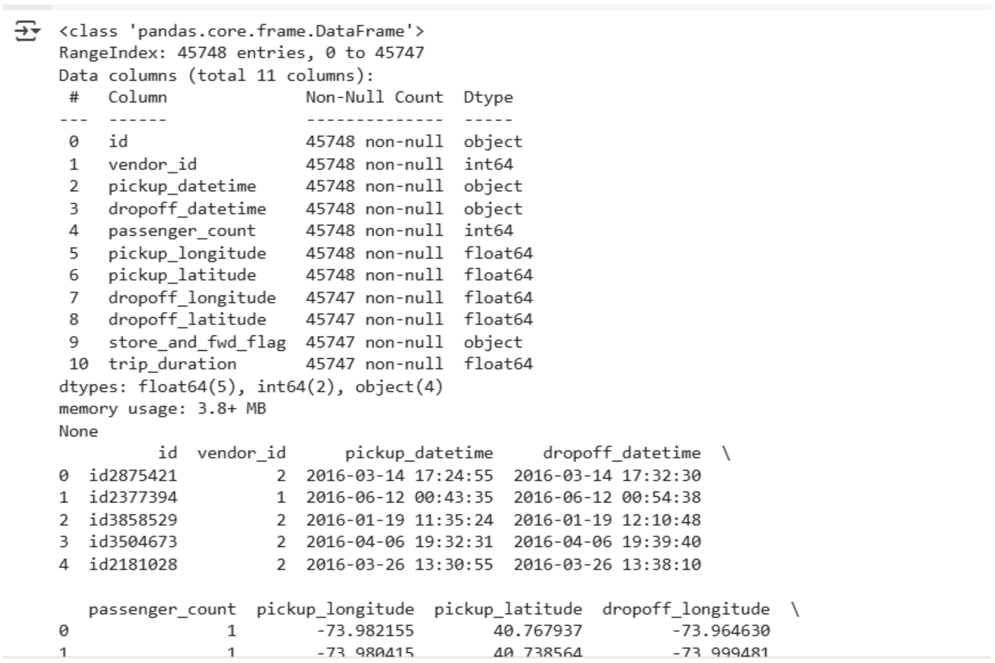
Step 1:

Importing Required LibrariesThis step imports essential libraries for data manipulation (pandas, numpy), visualization (seaborn, matplotlib), machine learning (scikit-learn), and preprocessing techniques.

LogisticRegression and LinearRegression from sklearn.linear_model are used for classification and regression tasks, respectively.



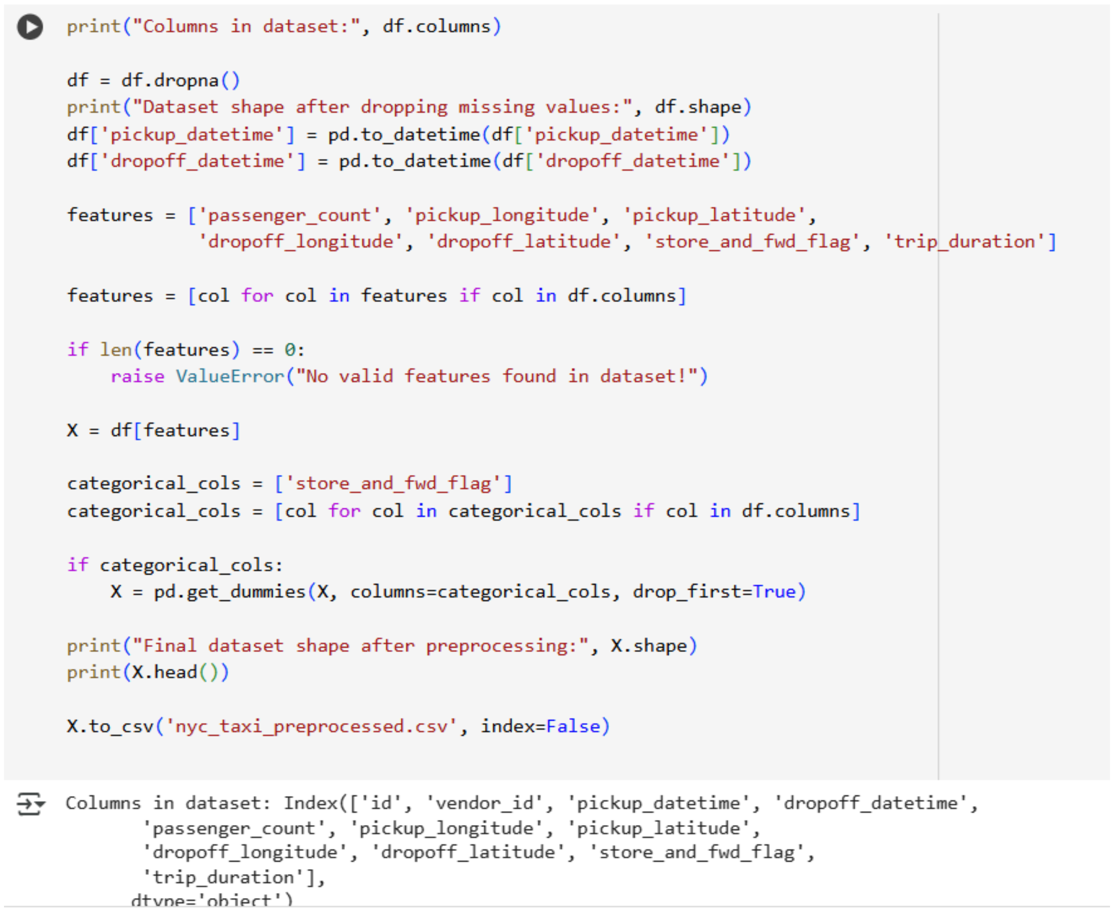
```
import pandas as pd

df = pd.read_csv('/content/NYC.csv')

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



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45748 entries, 0 to 45747
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  45748 non-null  object
 1   vendor_id           45748 non-null  int64
 2   pickup_datetime     45748 non-null  object
 3   dropoff_datetime    45748 non-null  object
 4   passenger_count     45748 non-null  int64
 5   pickup_longitude    45748 non-null  float64
 6   pickup_latitude     45748 non-null  float64
 7   dropoff_longitude   45747 non-null  float64
 8   dropoff_latitude    45747 non-null  float64
 9   store_and_fwd_flag  45747 non-null  object
 10  trip_duration       45747 non-null  float64
dtypes: float64(5), int64(2), object(4)
memory usage: 3.8+ MB
None
          id  vendor_id      pickup_datetime     dropoff_datetime  \
0  id2875421          2  2016-03-14 17:24:55  2016-03-14 17:32:30
1  id2377394          1  2016-06-12 00:43:35  2016-06-12 00:54:38
2  id3858529          2  2016-01-19 11:35:24  2016-01-19 12:10:48
3  id3504673          2  2016-04-06 19:32:31  2016-04-06 19:39:40
4  id2181028          2  2016-03-26 13:30:55  2016-03-26 13:38:10

   passenger_count  pickup_longitude  pickup_latitude  dropoff_longitude  \
0                1        -73.982155        40.767937         -73.964630
1                1        -73.980415        40.738564         -73.999481
```

NAME: Atharva Shinde DIV: D15C ROLL NO: 54

Step 2: Data Preprocessing



```
print("Columns in dataset:", df.columns)

df = df.dropna()
print("Dataset shape after dropping missing values:", df.shape)
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'])

features = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
            'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag', 'trip_duration']

features = [col for col in features if col in df.columns]

if len(features) == 0:
    raise ValueError("No valid features found in dataset!")

X = df[features]

categorical_cols = ['store_and_fwd_flag']
categorical_cols = [col for col in categorical_cols if col in df.columns]

if categorical_cols:
    X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

print("Final dataset shape after preprocessing:", X.shape)
print(X.head())

X.to_csv('nyc_taxi_preprocessed.csv', index=False)
```

```
Columns in dataset: Index(['id', 'vendor_id', 'pickup_datetime', 'dropoff_datetime',
       'passenger_count', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'store_and_fwd_flag',
       'trip_duration'],
      dtype='object')
```

Dropping Irrelevant Columns:

- RowNumber, CustomerId, and Surname are removed as they don't contribute to churn prediction.

Handling Missing Values:

- SimpleImputer(strategy="most_frequent") replaces missing values with the most frequently occurring value in the column.

Encoding Categorical Variables:

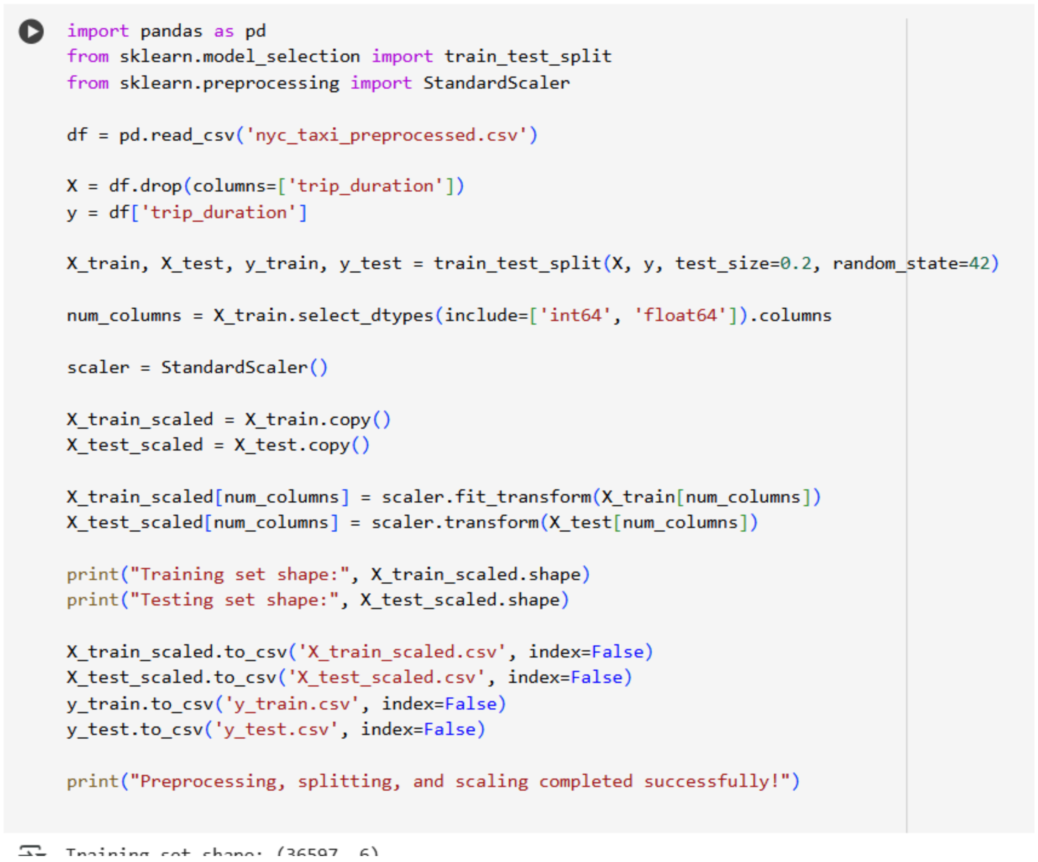
- LabelEncoder() converts Gender into numerical form (Female=0, Male=1).
- pd.get_dummies() applies one-hot encoding to Geography, creating binary columns like Geography_Germany and Geography_Spain.

Filling Missing Values for Other Columns:

- Age is replaced with the median value.
- IsActiveMember is filled with the most frequently occurring value.

NAME: Atharva Shinde DIV: D15C ROLL NO: 54

Step 3: Splitting the Dataset



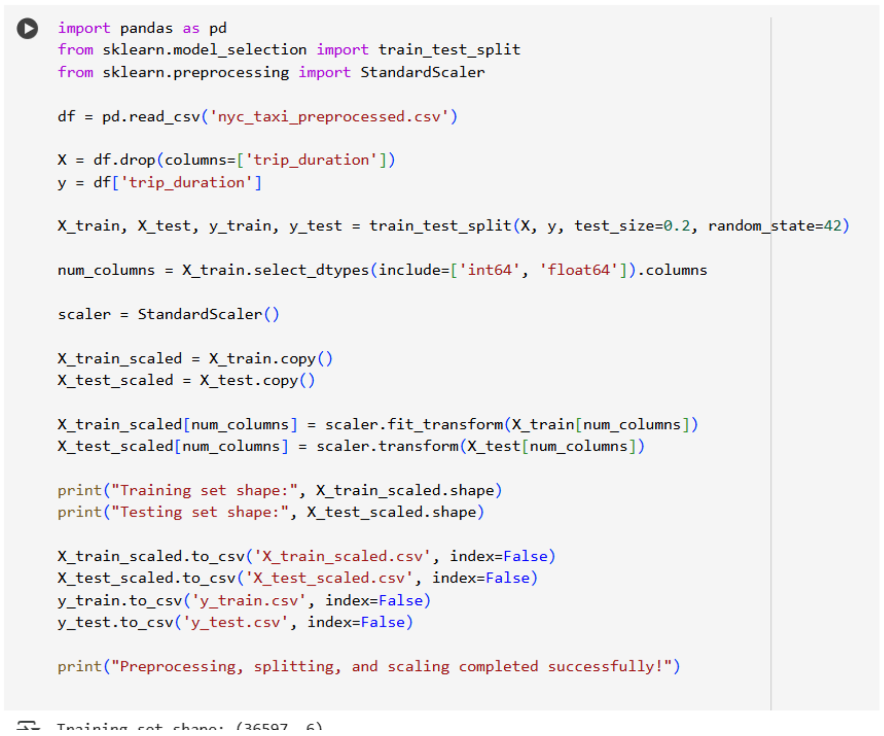
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('nyc_taxi_preprocessed.csv')

X = df.drop(columns=['trip_duration'])
y = df['trip_duration']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

num_columns = X_train.select_dtypes(include=['int64', 'float64']).columns

scaler = StandardScaler()

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])

print("Training set shape:", X_train_scaled.shape)
print("Testing set shape:", X_test_scaled.shape)

X_train_scaled.to_csv('X_train_scaled.csv', index=False)
X_test_scaled.to_csv('X_test_scaled.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
y_test.to_csv('y_test.csv', index=False)

print("Preprocessing, splitting, and scaling completed successfully!")
```

```
Training set shape: (36597, 6)
```

- Exited is the target variable (1 = churned, 0 = not churned).
- X consists of all other columns.
- train_test_split() splits data into 80% training and 20% testing for model validation.
- Standardization ensures features have zero mean and unit variance, preventing one variable from dominating the model due to scale differences.

Step 4: Logistic Regression Model

NAME: Atharva Shinde DIV: D15C ROLL NO: 54

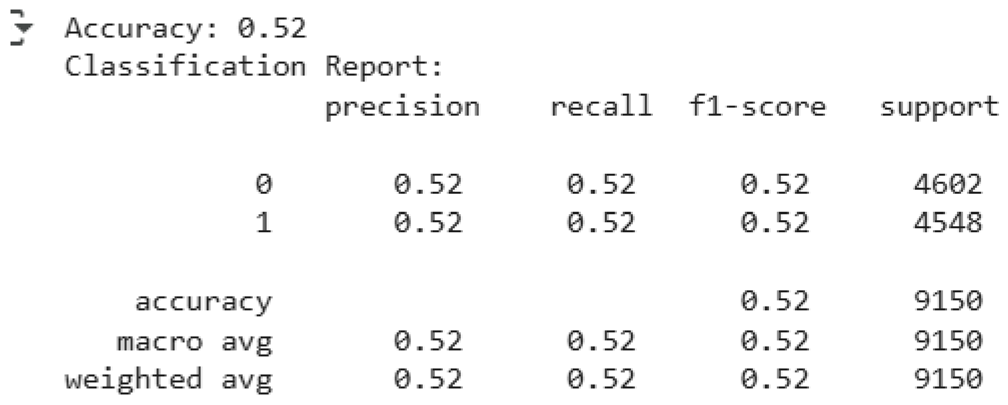


```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('nyc_taxi_preprocessed.csv')

X = df.drop(columns=['trip_duration'])
y = df['trip_duration']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

num_columns = X_train.select_dtypes(include=['int64', 'float64']).columns

scaler = StandardScaler()

X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])

print("Training set shape:", X_train_scaled.shape)
print("Testing set shape:", X_test_scaled.shape)

X_train_scaled.to_csv('X_train_scaled.csv', index=False)
X_test_scaled.to_csv('X_test_scaled.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
y_test.to_csv('y_test.csv', index=False)

print("Preprocessing, splitting, and scaling completed successfully!")
```

```
Training set shape: (36597, 6)
```

Evaluation Metrics:
accuracy_score() measures overall correct predictions.

- classification_report() shows precision, recall, and F1-score.
- confusion_matrix() provides True Positives, False Positives, True Negatives, and False Negatives.



```
Accuracy: 0.52
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.52      0.52      4602
           1       0.52      0.52      0.52      4548

    accuracy                           0.52      9150
   macro avg       0.52      0.52      0.52      9150
weighted avg       0.52      0.52      0.52      9150
```

Step 6: Linear Regression

Linear Regression is a fundamental statistical and machine learning technique used to model the relationship between a dependent variable (target) and one or more independent variables (features). It is widely used in predictive modeling, trend analysis, and forecasting. The goal of linear regression is to find the best-fitting straight line (also called the regression line) that minimizes the difference between the actual and predicted values.

Types of Linear Regression

1. **Simple Linear Regression** – Involves one independent variable (e.g., predicting salary based on years of experience).

2. **Multiple Linear Regression** – Involves multiple independent variables (e.g., predicting house prices based on size, location, and number of rooms).

Formula:

$$y = mx + c$$

NAME: Atharva Shinde DIV: D15C ROLL NO: 54

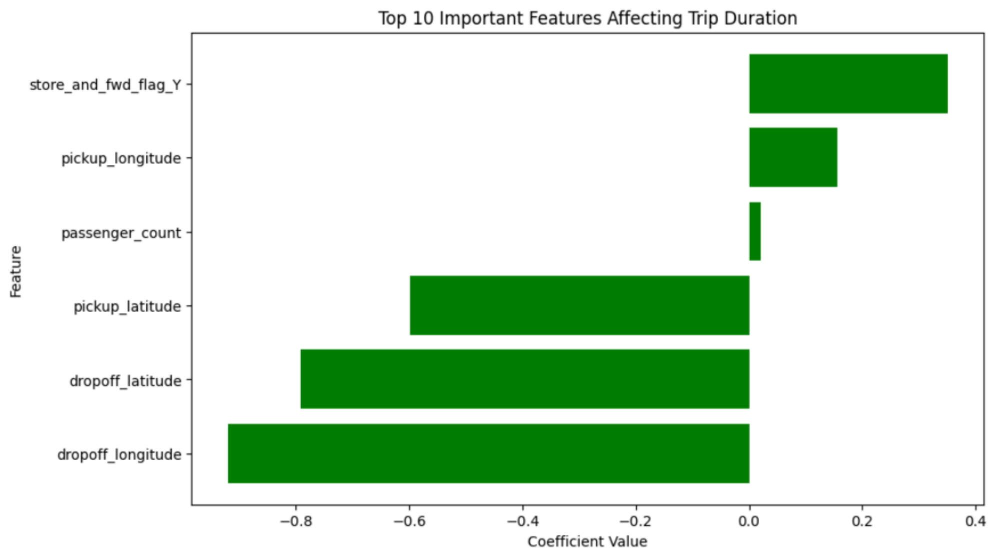
```
import pandas as pd
import matplotlib.pyplot as plt

feature_names = X.columns
coefficients = model.coef_[0]

coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})

coef_df = coef_df.sort_values(by='Coefficient', ascending=False)

print(" 🔹 Features with the strongest positive impact on trip duration:")
print(coef_df.head(10))

print("\n 🔹 Features with the strongest negative impact on trip duration:")
print(coef_df.tail(10))

plt.figure(figsize=(10, 6))
plt.barh(coef_df['Feature'][:10], coef_df['Coefficient'][:10], color='green')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
plt.title('Top 10 Important Features Affecting Trip Duration')
plt.gca().invert_yaxis()
plt.show()
```

Evaluation:

```
R-squared Score: -0.005540086125667365
```

Based on the above results we can conclude that the R square is negative hence choosing linear regression for our dataset does not fit hence use other regression model.

Graphical Representation:



Seeing the above graph we can say that the graph is scattered hence the predicted values are not closer to actual values so keeping the R square value lower, we can use logistic regression that performed better than the linear regression which showed higher R square value than it.

Conclusion:

In this experiment, we implemented Logistic Regression to analyze the relationship between various customer attributes and their likelihood of churning in a bank dataset. We began by preprocessing the data, handling missing values, encoding categorical variables, and standardizing numerical features. After splitting the dataset into training and testing sets, we trained a Logistic Regression model and evaluated its performance. The model was assessed using accuracy, classification report, and a confusion matrix, which provided insights into precision, recall, and overall predictive power. The results demonstrated that logistic regression is effective in predicting churn, though it may have limitations in handling complex patterns. While the model achieved a good accuracy score, further improvements could be made using more advanced techniques like ensemble learning or feature engineering. Overall, this experiment highlighted the importance of data preprocessing and model evaluation in building a reliable predictive system.

Name : Atharva Shinde **Div : D15C** **Roll No :**

15 Experiment No. - 6 :

Problem Statement: Classification modelling
a. Choose a classifier for a classification problem.
b. Evaluate the performance of the classifier.
Perform Classification using the below 4 classifiers on the same dataset which you have used for

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Support Vector Machines (SVMs)
- Decision Tree

Introduction :

Classification algorithms are crucial in predicting categorical outcomes and analyzing labeled data. In this experiment, we applied two supervised learning techniques — Naive Bayes and K-Nearest Neighbors (KNN) — to classify air quality based on the Air Quality Index (AQI) dataset.

Naive Bayes is a probabilistic classifier based on Bayes' theorem, known for its simplicity, efficiency, and effectiveness, especially with small datasets or when feature independence is assumed. On the other hand, KNN is a distance-based classifier that assigns a class label based on the majority of the nearest neighbors, making it intuitive yet sensitive to feature scaling and noise.

The primary objective was to predict the trip duration of NYC taxi rides based on features such as pickup and drop-off coordinates, time-related variables, and passenger count. By applying various regression algorithms and evaluating their performance

Implementation Process :

Step 1 : We perform a series of data pre-processing operations that involve calculating the missing percentage for each column ,using mean imputation for columns that have less than 20 % missing values,using iterative imputation methods for columns between 20 - 50% missing values and dropping the xylene column that had around 60 % missing values.

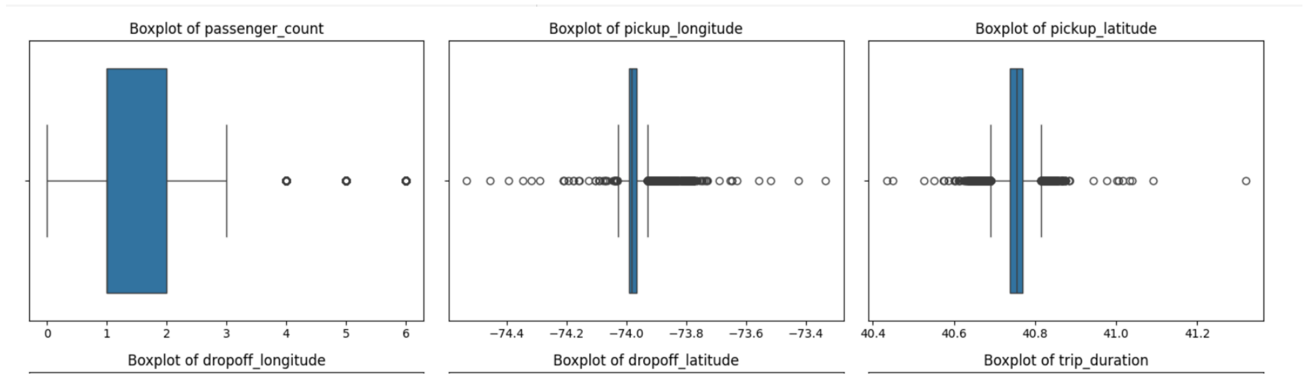
```
from sklearn.impute import SimpleImputer

mean_imputer = SimpleImputer(strategy='mean')

columns_to_impute = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
                     'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

for col in columns_to_impute:
    if col in df.columns:
        df[col] = mean_imputer.fit_transform(df[[col]])
```

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Define the iterative imputer
iter_imputer = IterativeImputer(max_iter=10, random_state=42)

# Select numeric columns with 20%-50% missing values (example list — update based on actual %)
columns_to_impute = ['pickup_longitude', 'pickup_latitude',
                     'dropoff_longitude', 'dropoff_latitude']

# Apply the imputer if these columns exist
existing_columns = [col for col in columns_to_impute if col in df.columns]
df[existing_columns] = iter_imputer.fit_transform(df[existing_columns])

# Show remaining missing values
print("Remaining Missing Values:\n", df[existing_columns].isnull().sum())
```

```
df.drop('store_and_fwd_flag', axis=1, inplace=True)
```

Step 2 : We verify that all of the missing values have been removed and then move to identifying and eliminating the outliers using boxplot analysis.

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

# Relevant numeric columns in NYC Taxi dataset
numeric_columns = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
                   'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

# Plot boxplots
plt.figure(figsize=(15, 8))
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(2, 3, i)
    sns.boxplot(x=df[col])
    plt.title(f'Boxplot of {col}')
    plt.xlabel("")  # Optional: Hide x-label if too long
plt.tight_layout()
plt.show()
```



Step 3 : Apply standardizing operations using the z-score method.

```python
import numpy as np

# Define threshold for z-scores
threshold = 3

# NYC Taxi dataset's relevant numeric columns
numeric_columns = ['passenger_count', 'pickup_longitude', 'pickup_latitude',
                   'dropoff_longitude', 'dropoff_latitude', 'trip_duration']

# Apply z-score based capping
for col in numeric_columns:
    mean_val = df[col].mean()
    std_dev = df[col].std()

    # Clip values outside ±3 standard deviations
    df[col] = np.clip(df[col], mean_val - threshold * std_dev, mean_val + threshold * std_dev)
```

Step 4 : We then encode categorical features in the dataset to prepare it for machine learning algorithms, as our model requires numerical data.

The LabelEncoder converts the categorical values of the AQI_Bucket column (like "Good", "Satisfactory", "Poor", etc.) into integer labels (0, 1, 2, etc.).

One-Hot Encoding creates binary columns for each unique value in the City column (like Delhi, Mumbai, etc.).

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['trip_duration_bucket'] = le.fit_transform(df['trip_duration_bucket'])

print("Label Encoded 'trip_duration_bucket':")
print(df['trip_duration_bucket'].unique())
```

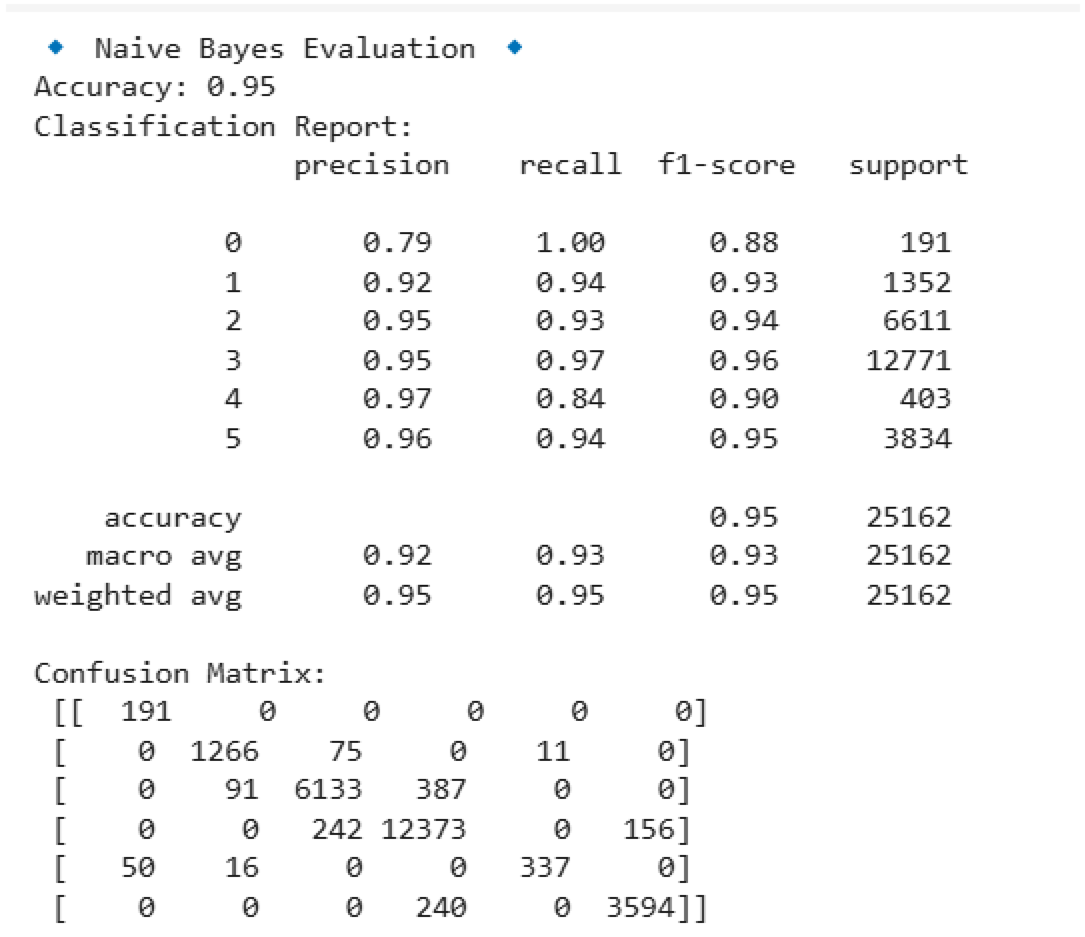
Step 5 : We then perform splitting of the dataset into a 70 : 30 split to proceed with the preparation and evaluation of the models.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Dropping unnecessary columns (like IDs and datetime)
X = df.drop(['id', 'pickup_datetime', 'dropoff_datetime', 'trip_duration'], axis=1)  # Features
y = df['trip_duration']  # Target variable

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardizing numerical features only
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrame (optional)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)

print("Scaled Features (First 5 rows):")
print(X_train_scaled.head())
```

Step 6 : The preparation of the Naive Bayes Classifier for application on to the dataset.

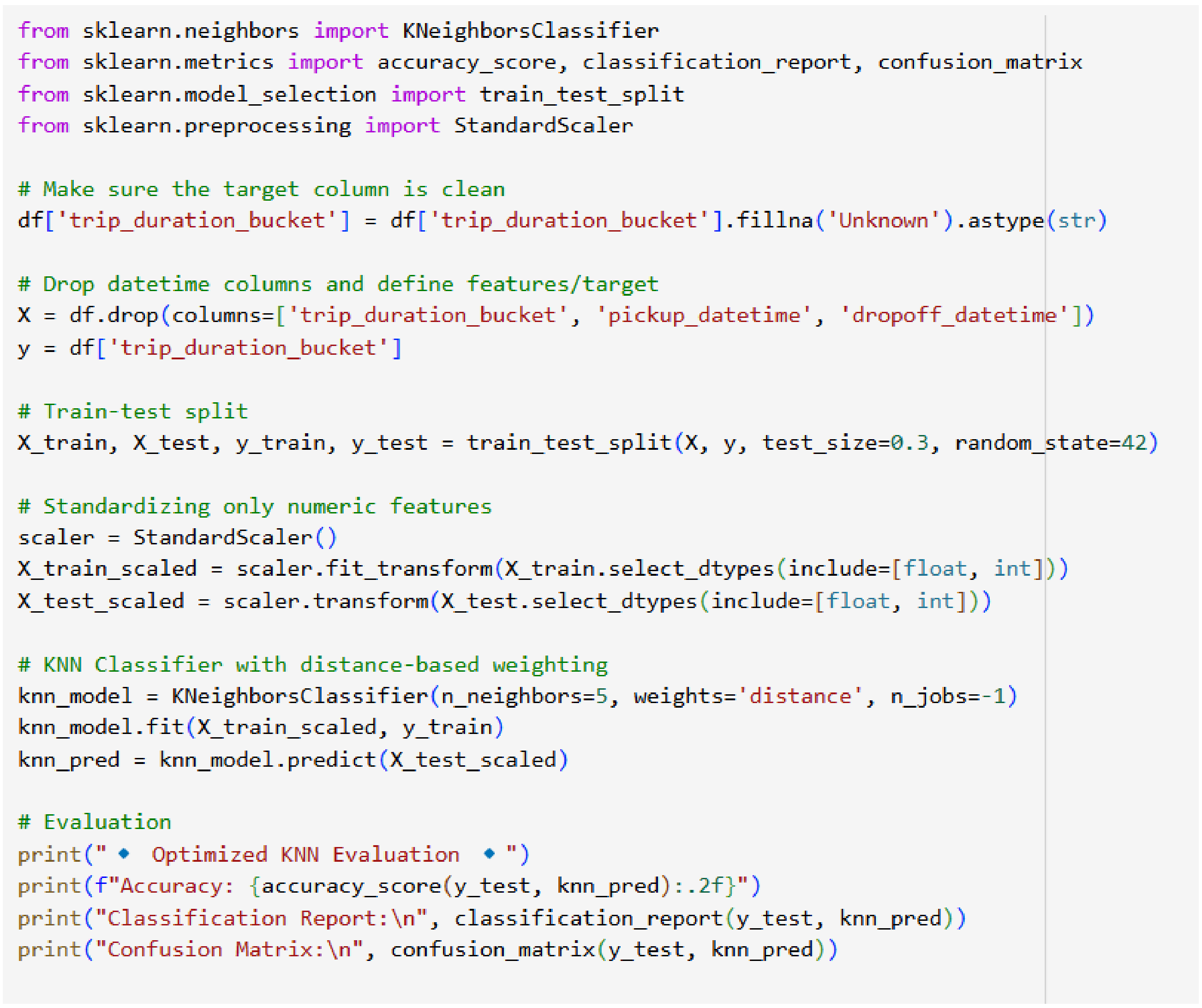
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Handle missing values and format 'trip_duration_bucket'
df['trip_duration_bucket'] = df['trip_duration_bucket'].fillna('Unknown').astype(str)

# Drop datetime-related columns and set features/target
X = df.drop(columns=['trip_duration_bucket', 'pickup_datetime', 'dropoff_datetime'])
y = df['trip_duration_bucket']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardizing only numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=[float, int]))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=[float, int]))

# Naive Bayes Model
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)
nb_pred = nb_model.predict(X_test_scaled)

# Evaluation
print(" ◆  Naive Bayes Evaluation  ◆ ")
print(f"Accuracy: {accuracy_score(y_test, nb_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, nb_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.



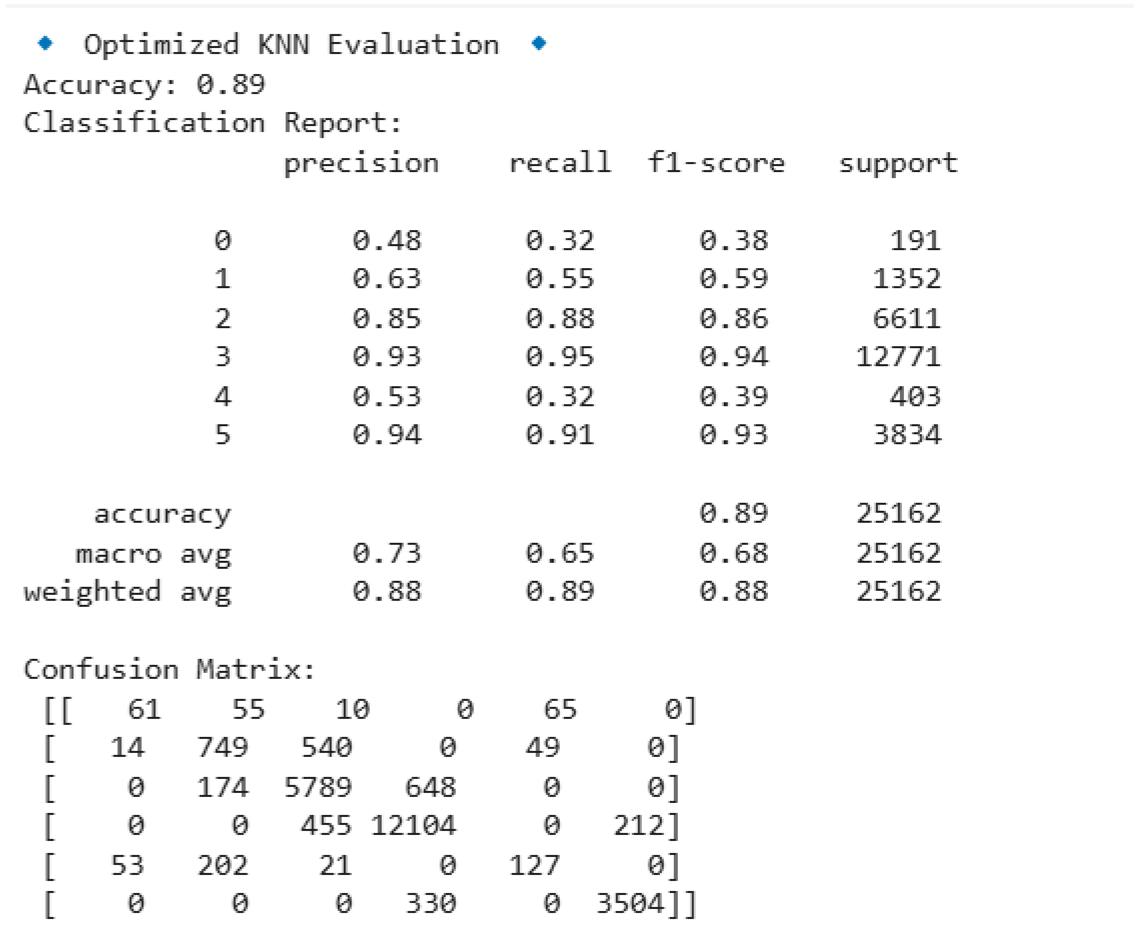
```
 🔹 Naive Bayes Evaluation 🔹
Accuracy: 0.95
Classification Report:
              precision    recall  f1-score   support

           0       0.79      1.00      0.88       191
           1       0.92      0.94      0.93      1352
           2       0.95      0.93      0.94      6611
           3       0.95      0.97      0.96     12771
           4       0.97      0.84      0.90       403
           5       0.96      0.94      0.95      3834

    accuracy                           0.95     25162
   macro avg       0.92      0.93      0.93     25162
weighted avg       0.95      0.95      0.95     25162

Confusion Matrix:
 [[  191     0     0     0     0     0]
 [    0  1266    75     0    11     0]
 [    0    91  6133   387     0     0]
 [    0     0   242 12373     0   156]
 [   50    16     0     0   337     0]
 [    0     0     0   240     0  3594]]
```

Step 7 : The preparation of the KNN (K-Nearest Neighbors) for application on to the dataset.



```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Make sure the target column is clean
df['trip_duration_bucket'] = df['trip_duration_bucket'].fillna('Unknown').astype(str)

# Drop datetime columns and define features/target
X = df.drop(columns=['trip_duration_bucket', 'pickup_datetime', 'dropoff_datetime'])
y = df['trip_duration_bucket']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardizing only numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=[float, int]))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=[float, int]))

# KNN Classifier with distance-based weighting
knn_model = KNeighborsClassifier(n_neighbors=5, weights='distance', n_jobs=-1)
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)

# Evaluation
print(" 🔹 Optimized KNN Evaluation 🔹 ")
print(f"Accuracy: {accuracy_score(y_test, knn_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, knn_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.



```
🔹 Optimized KNN Evaluation 🔹
Accuracy: 0.89
Classification Report:
              precision    recall  f1-score   support

           0       0.48      0.32      0.38       191
           1       0.63      0.55      0.59      1352
           2       0.85      0.88      0.86      6611
           3       0.93      0.95      0.94     12771
           4       0.53      0.32      0.39       403
           5       0.94      0.91      0.93      3834

    accuracy                           0.89     25162
   macro avg       0.73      0.65      0.68     25162
weighted avg       0.88      0.89      0.88     25162

Confusion Matrix:
 [[   61    55    10     0    65     0]
 [   14   749   540     0    49     0]
 [    0   174  5789   648     0     0]
 [    0     0   455 12104     0   212]
 [   53   202    21     0   127     0]
 [    0     0     0   330     0  3504]]
```

Conclusion :

Conclusion

Based on the classification results obtained from both the **Naive Bayes** and the **optimized K-Nearest Neighbors (KNN)** algorithms, the following conclusions can be drawn:

- **Model Performance**: The **Naive Bayes model achieved a higher accuracy of 95%**, compared to **89% for the optimized KNN model**. This indicates that **Naive Bayes is more effective** in accurately classifying trip durations into predefined categories for this dataset.

- **Evaluation Metrics**: Naive Bayes consistently outperformed KNN across all major metrics—**precision, recall, and F1-score**—especially for classes with larger support (like class labels 2, 3, and 5). This indicates that Naive Bayes is more reliable in correctly identifying the correct duration classes.

- **Confusion Matrix Analysis**: The **confusion matrix for Naive Bayes** shows fewer misclassifications across all classes compared to KNN. In contrast, the **KNN model misclassified a notable number of samples**, particularly between adjacent duration classes, suggesting it struggled more with borderline cases.

- **Model Suitability**: While **KNN** is a strong non-parametric model that can capture complex decision boundaries, its performance is **limited by the curse of dimensionality** and sensitivity to noisy data. **Naive Bayes**, on the other hand, demonstrates strong performance even with its assumption of feature independence, making it **computationally efficient and highly accurate** for this classification task.

Name: Atharva Shinde **D15C/54**

Experiment 7

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

Theory:

CLUSTERING

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.

Applications of Clustering in different fields

1. Marketing: It can be used to characterize & discover customer segments for marketing purposes.

2. Biology: It can be used for classification among different species of plants and animals.

3. Libraries: It is used in clustering different books on the basis of topics and information.

4. Insurance: It is used to acknowledge the customers, their policies and identifying the frauds.

5. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.

6. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones

Clustering Algorithms

When choosing a clustering algorithm, you should consider whether the algorithm scales to your dataset. Datasets in machine learning can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples n, denoted as O(n2) in complexity notation. O(n2) algorithms are not practical when the number of examples are in millions.

1. Density-Based Methods:

These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

2. Hierarchical Based Methods:

The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

1. Agglomerative (bottom-up approach)

2. Divisive (top-down approach)

examples CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies), etc.

3. Partitioning Methods:

These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.

4. Grid-based Methods:

In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects, for example STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest), etc.

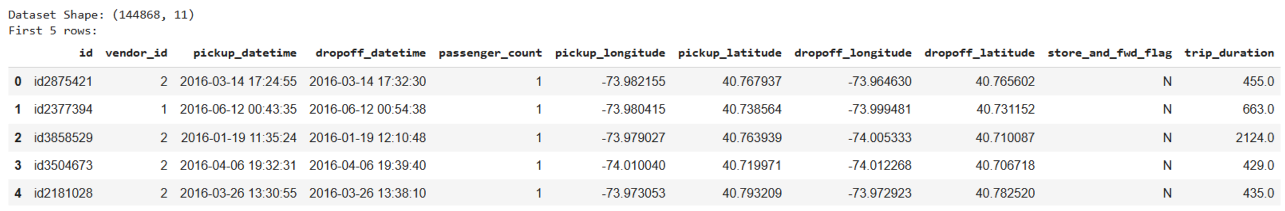
Dataset: NYC Taxi

The dataset used in this experiment is related to **New York City taxi trips**. The goal is to analyze various trip-related factors such as trip duration, pickup and drop-off locations, passenger count, and vendor details. This dataset is useful for transportation analytics, trip duration prediction, and spatial-temporal analysis.

The dataset contains the following columns:

- **id**: Unique identifier for each trip
- **vendor_id**: Code indicating the provider associated with the trip record
- **pickup_datetime**: Date and time when the trip started
- **dropoff_datetime**: Date and time when the trip ended
- **passenger_count**: Number of passengers in the vehicle
- **pickup_longitude**: Longitude at the pickup point
- **pickup_latitude**: Latitude at the pickup point
- **dropoff_longitude**: Longitude at the drop-off point
- **dropoff_latitude**: Latitude at the drop-off point
- **store_and_fwd_flag**: Whether the trip record was stored before forwarding (Y/N)
- **trip_duration**: Duration of the trip in seconds (Target variable)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
from IPython.display import display


# Load the NYC Taxi dataset
file_path = "NYC.csv"  # Update with your actual dataset path
df = pd.read_csv(file_path)

# Display basic information about the dataset
print("Dataset Shape:", df.shape)
print("First 5 rows:")
display(df.head())
```



```
Dataset Shape: (144868, 11)
First 5 rows:
```

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455.0
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663.0
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124.0
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429.0
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435.0

Here we are going to see implementation of K-means and DB-SCAN clustering algorithms.

1) K-means clustering

1. Objective

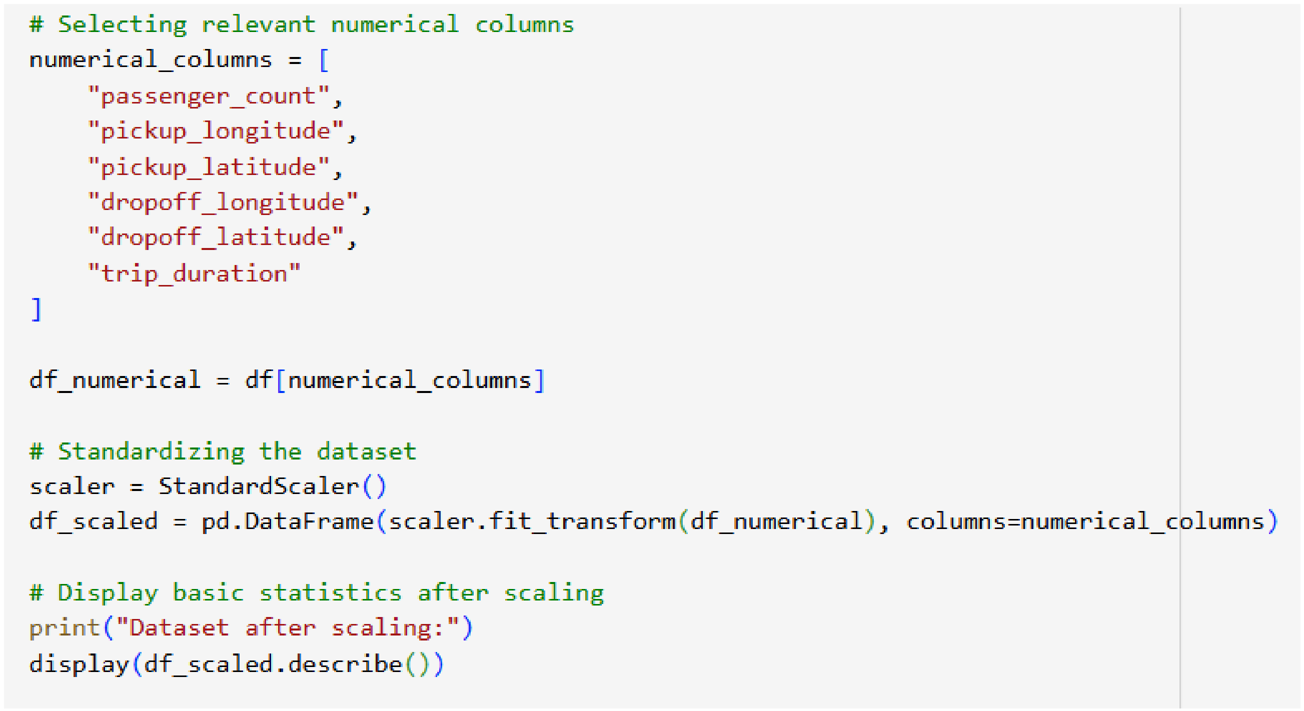
To group data points into distinct clusters based on feature similarity using the K-Means algorithm.

2. Why the Elbow Method?

The **Elbow Method** helps determine the **optimal number of clusters (k)** by plotting:

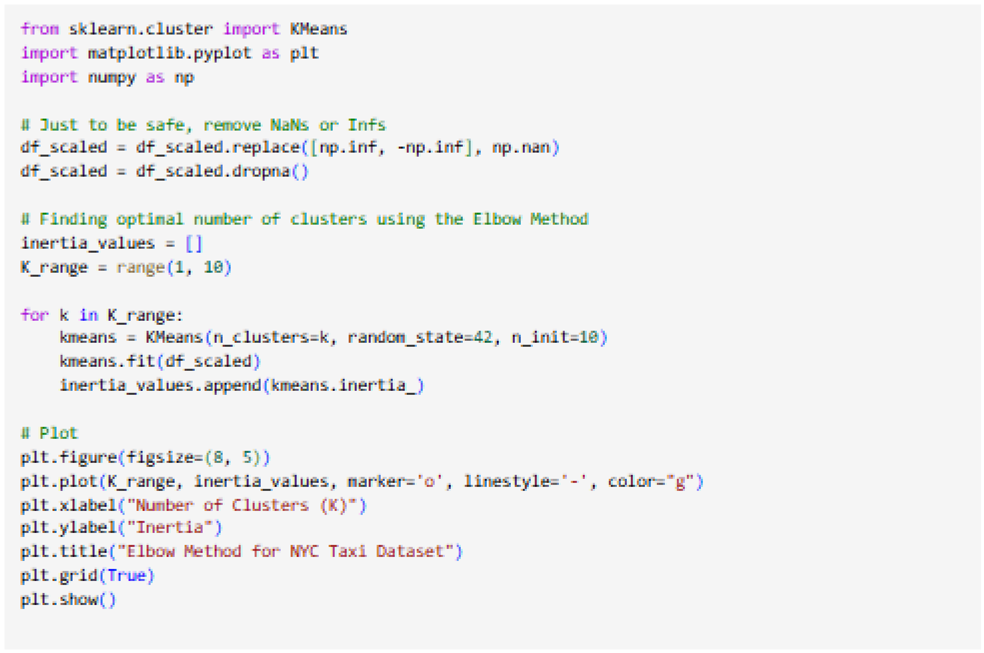
- **X-axis**: Number of clusters (k)
- **Y-axis**: Within-Cluster Sum of Squares (WCSS / Inertia)

We select the **“elbow point”** – where the decrease in WCSS slows down – as the best k.
Now lets see the actual implementation.

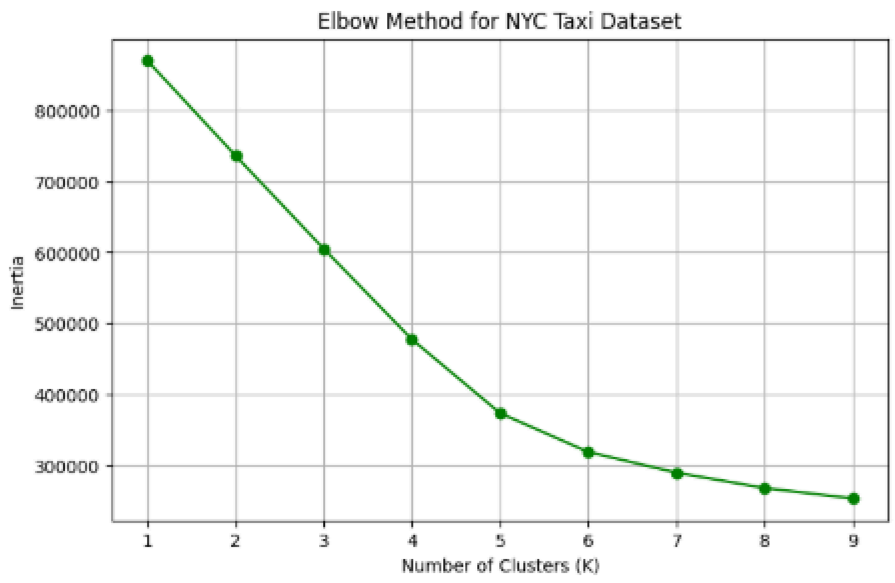


```
# Selecting relevant numerical columns
numerical_columns = [
    "passenger_count",
    "pickup_longitude",
    "pickup_latitude",
    "dropoff_longitude",
    "dropoff_latitude",
    "trip_duration"
]

df_numerical = df[numerical_columns]

# Standardizing the dataset
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_numerical), columns=numerical_columns)

# Display basic statistics after scaling
print("Dataset after scaling:")
display(df_scaled.describe())
```

We are primarily selecting 2 columns on which our clustering will be based i.e Age and Working hours per week.Filling the missing values with median here.

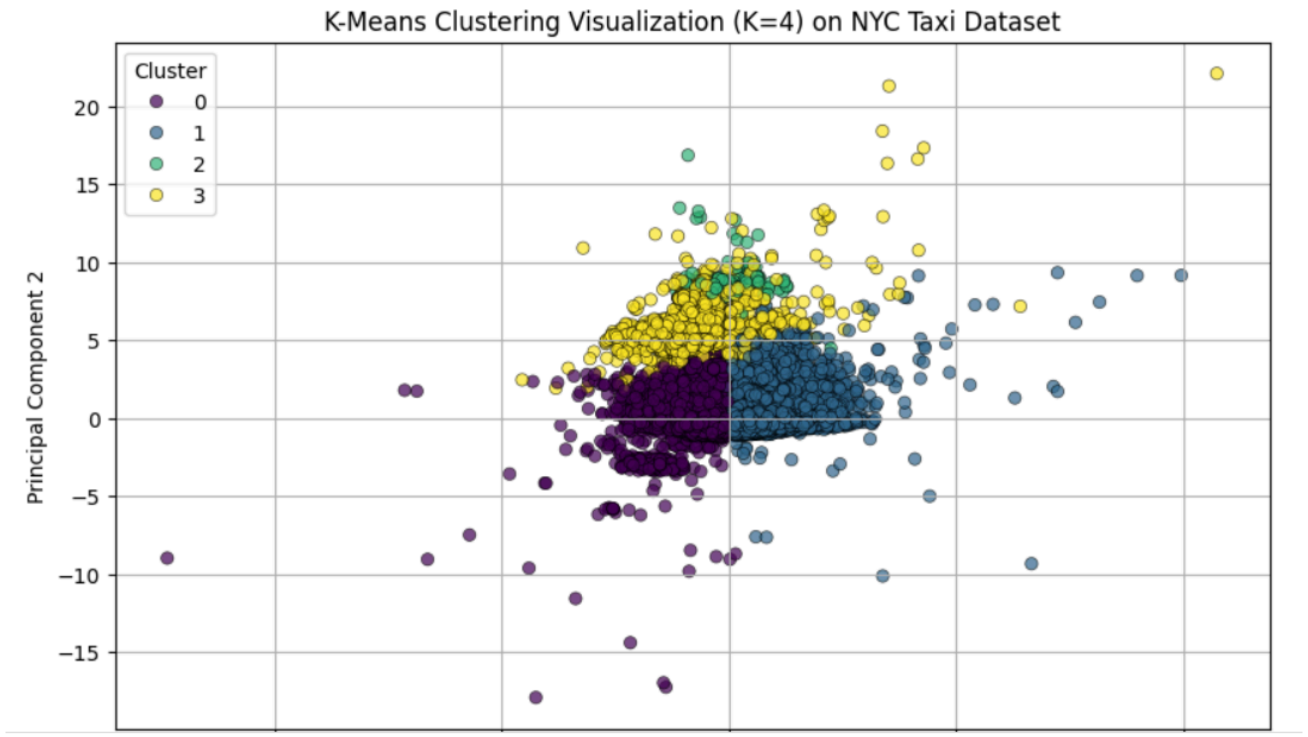


```python
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np

# Just to be safe, remove NaNs or Infs
df_scaled = df_scaled.replace([np.inf, -np.inf], np.nan)
df_scaled = df_scaled.dropna()

# Finding optimal number of clusters using the Elbow Method
inertia_values = []
K_range = range(1, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia_values.append(kmeans.inertia_)

# Plot
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia_values, marker='o', linestyle='-', color="g")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method for NYC Taxi Dataset")
plt.grid(True)
plt.show()
```

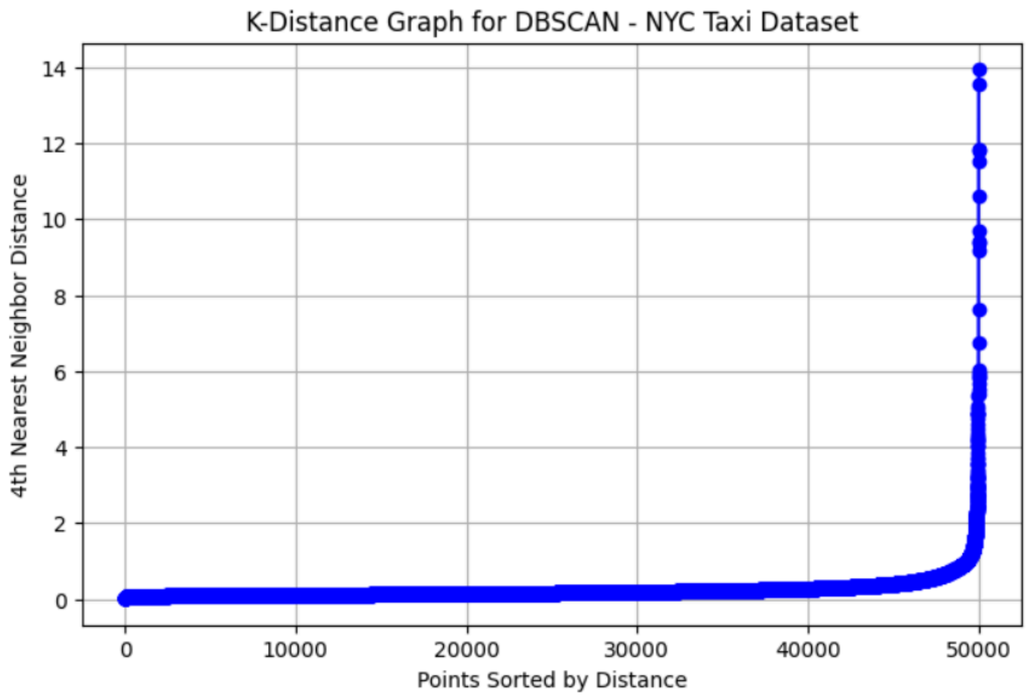


Using the elbow method to find the optimal number clusters(k) that we need .
We have chosen k=5.

```python
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import pandas as pd

# Apply PCA to reduce dimensions to 2D
pca = PCA(n_components=2)
pca_features = pca.fit_transform(df_scaled.drop(columns=["Cluster_K4"]))

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2"])
df_pca["Cluster"] = df_scaled["Cluster_K4"]

# Plot the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(
    x="PC1", y="PC2",
    hue="Cluster",
    palette="viridis",
    data=df_pca,
    alpha=0.7,
    edgecolor="k"
)
plt.title("K-Means Clustering Visualization (K=4) on NYC Taxi Dataset")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster")
plt.grid(True)
plt.show()
```



The **elbow point** (e.g., at k = 5) indicates that 3 clusters give a good balance between **model accuracy** and **simplicity**.
Each data point is assigned to the nearest cluster based on **Euclidean distance**.
The result reveals **natural groupings** or patterns in the dataset.



Conclusion:

In this experiment, we applied and compared two clustering algorithms — **K-Means** and **DBSCAN** — to the NYC Taxi dataset.

- **K-Means Clustering (K=4)**:
  Using PCA for dimensionality reduction, the scatter plot clearly showed four well-defined clusters. K-Means efficiently grouped the data into compact spherical clusters. However, it assumes equal cluster density and may misclassify outliers or non-globular patterns.

- **DBSCAN**:
  The K-Distance graph helped determine the appropriate value for ε (epsilon). DBSCAN successfully identified clusters of varying shapes and densities, and most importantly, was able to detect **outliers**, which K-Means does not support.

- **Comparison**:

  - **K-Means** is best suited for **structured, well-separated clusters**, and requires the number of clusters (`k`) to be known in advance.

  - **DBSCAN** is more **flexible**, can detect **arbitrary shaped clusters**, and **automatically identifies noise/outliers**, making it more suitable for real-world datasets with complex patterns like taxi rides across a city.

NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

EXPERIMENT NO.: 8

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

Theory:
Types of Recommendation Systems

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

1. Content-Based Filtering
   Idea: Recommends items similar to those the user has liked before.
   ● Works on: Item features (attributes such as brand, price, category). Example:
   ● If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.
   ● Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

2. Collaborative Filtering (CF)
   Idea: Recommends items based on similar users' preferences.
   ● Works on: User interactions rather than item features.
   Example:● If User A and User B have similar purchase histories, items bought by User
   A but not yet by User B will be recommended to User B.
   ● Uses methods like User-Based CF and Item-Based CF.

3. Hybrid Recommendation System
   Idea: Combines Content-Based Filtering and Collaborative Filtering for better accuracy.
   Example:
   ● Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

4. Knowledge-Based Recommendation
   Idea: Recommends items based on explicit domain knowledge rather than past user behavior.

Example:

- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

Recommendation System Evaluation Measures

Accuracy Measures:

These metrics evaluate how well the recommended items match the actual preferences or ratings of users.

- **Mean Absolute Error (MAE)**:
  - Measures the average of the absolute differences between predicted ratings and actual ratings.
    - **Formula:** $MAE = \frac{1}{n}\sum_{i=1}^{n}|r_i - \hat{r}_i|$
      - r_i = Actual rating
      - $\hat{r}_i$ = Predicted rating
    - **Lower is better.**
- **Root Mean Squared Error (RMSE)**:
  - Similar to MAE but gives higher weight to large errors due to squaring the differences.

    Formula: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2}$
    - **Lower is better.**
- **Precision**:

- Measures the fraction of recommended items that are actually relevant to the user.

  Formula: $Precision = \frac{\text{Number of relevant recommended items}}{\text{Total number of recommended items}}$

  - **Higher is better.**

- **Recall**:
  - Measures the fraction of relevant items that were actually recommended to the user.

  Formula: $Recall = \frac{\text{Number of relevant recommended items}}{\text{Total number of relevant items}}$

  - **Higher is better.**

- **F1-Score**:
  - The harmonic mean of Precision and Recall, balancing both.

  Formula: $F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$

  - **Higher is better.**

- **Hit Rate**:
  - Measures the fraction of users for whom at least one relevant item is recommended.

  Formula: $HitRate = \frac{\text{Users with at least one relevant recommendation}}{\text{Total users}}$

  - **Higher is better.**

- **Coverage**:

- Measures the proportion of items from the total available set that are recommended to users.

  Formula: $Coverage = \frac{\text{Number of unique recommended items}}{\text{Total number of items available}}$

  - **Higher is better.**

Diversity and Novelty Measures:

These metrics focus on the **variety** of recommended items and how **unexpected** they are.

- **Diversity**:
  - Measures how different the recommended items are from each other. A diverse set prevents the system from recommending very similar items.
  - **Formula**:
    - Calculate the pairwise similarity between recommended items (e.g., cosine similarity) and compute the average diversity across all users.
    - **Higher diversity is better.**
- **Novelty**:
  - Measures how **unexpected** or **unknown** the recommended items are to the user.
  - For example, recommending items that the user hasn't interacted with before (e.g., exploring genres they haven't tried).
  - **Higher novelty is better.**
- **Serendipity**:
  - Similar to novelty but with a focus on the surprise element that still fits the user's interests.

- The idea is to recommend items that are surprising but still relevant.

Implementation

The Diet recommendation is built using Nearest Neighbors algorithm which is an unsupervised learner for implementing neighbor searches.For our case, we used the brute-force algorithm using cosine similarity due to its fast computation for small datasets.

1. **Importing dataset**

```
import kagglehub
from kagglehub import KaggleDatasetAdapter

# NYC Taxi dataset file name
file_path = "NYC.csv"

# Load the dataset from Kaggle
df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "yasserh/nyc-taxi-trip-duration",  # Replace with your actual dataset path
    file_path,
)

# Show first 5 rows
print("First 5 records:\n", df.head())
```

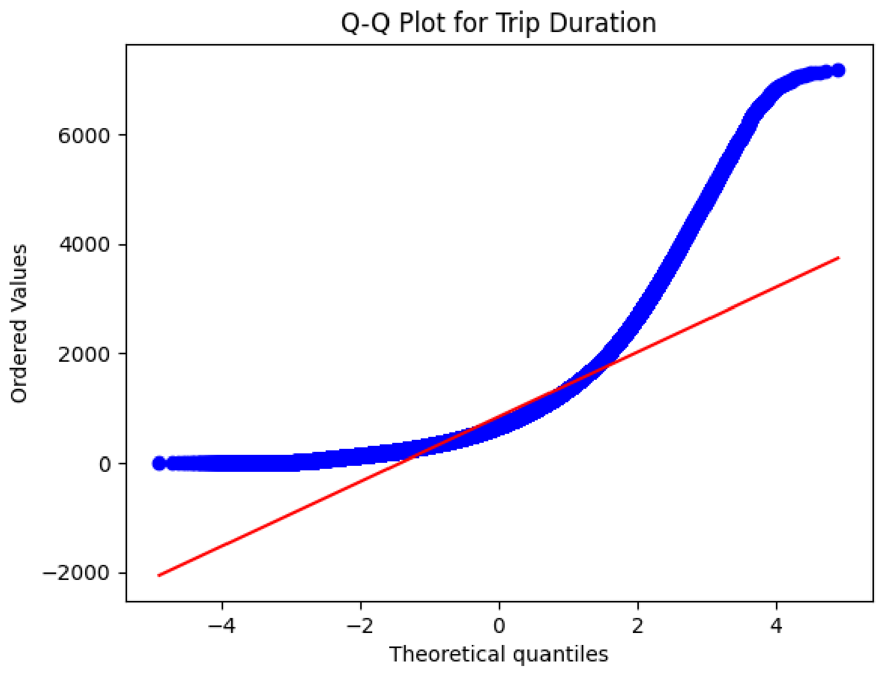
NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

```
data = df
data.head()
```

	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435

2. Detecting Outlier

```
import pylab
import scipy.stats as stats

# Optional: filter to remove extreme outliers (e.g., > 2 hours)
filtered_data = df[df['trip_duration'] <= 7200]['trip_duration']

# Create the Q-Q plot
stats.probplot(filtered_data.to_numpy(), dist="norm", plot=pylab)
pylab.title("Q-Q Plot for Trip Duration")
pylab.show()
```



The data for **trip duration** in the NYC Taxi dataset exhibits significant right-skewness, as shown in the Q-Q plot. This suggests that most taxi trips are relatively short, while a smaller number of trips with much longer durations are pulling the distribution toward the right. Additionally, the presence of these long-duration trips, even after filtering out extreme outliers (e.g., trips over 2 hours), indicates the influence of outliers on the overall distribution shape.

NAME:ATHARVA SHINDE CLASS:D15C ROLL.NO:54

3. **Setting the maximum nutritional values for each category for healthier recommendations**

```
max_passenger_count = 6              # Taxis usually take up to 6 passengers
max_trip_duration = 7200             # 2 hours
max_pickup_longitude = -73.7
min_pickup_longitude = -74.05
max_pickup_latitude = 40.9
min_pickup_latitude = 40.5
max_dropoff_longitude = -73.7
min_dropoff_longitude = -74.05
max_dropoff_latitude = 40.9
min_dropoff_latitude = 40.5

# Collecting in a list (optional)
max_list = [
    max_passenger_count, max_trip_duration,
    min_pickup_longitude, max_pickup_longitude,
    min_pickup_latitude, max_pickup_latitude,
    min_dropoff_longitude, max_dropoff_longitude,
    min_dropoff_latitude, max_dropoff_latitude
]
```

4. **Correlation among nutritional values**

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
passenger_count	1.000000	0.004727	-0.004670	0.000534	-0.001849	0.015139
pickup_longitude	0.004727	1.000000	-0.147061	0.270171	0.036266	0.361578
pickup_latitude	-0.004670	-0.147061	1.000000	0.054189	0.435407	-0.235009
dropoff_longitude	0.000534	0.270171	0.054189	1.000000	0.096224	0.260362
dropoff_latitude	-0.001849	0.036266	0.435407	0.096224	1.000000	-0.183805
trip_duration	0.015139	0.361578	-0.235009	0.260362	-0.183805	1.000000

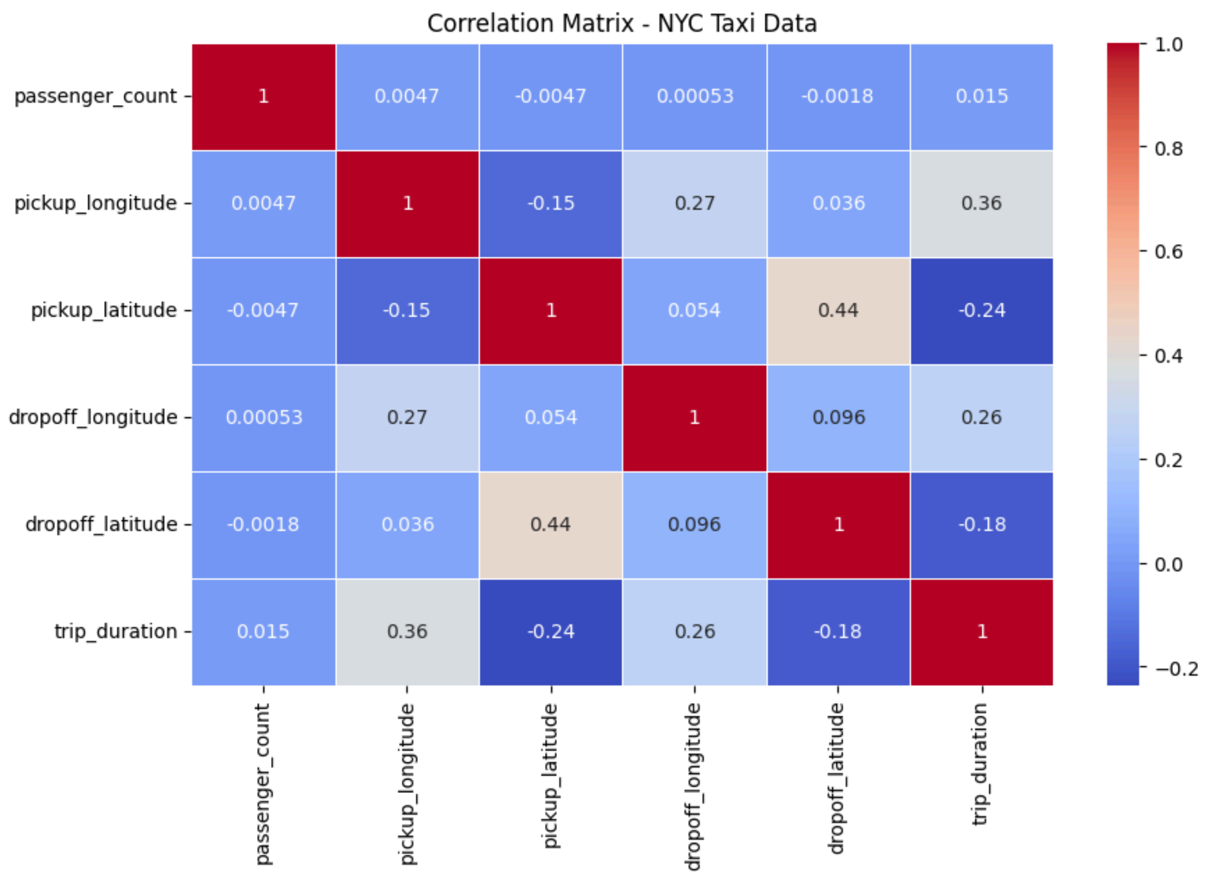
- **Pickup longitude** and **trip duration** show a moderate positive correlation (≈ 0.36), indicating that the location where a trip starts may have some influence on its duration, possibly due to geographic patterns like traffic congestion or distance from city centers.

- **Dropoff longitude** is also moderately correlated with **trip duration** (≈ 0.26), suggesting that trips ending at certain locations tend to take longer, possibly influenced by traffic or distance.

- **Pickup latitude** and **dropoff latitude** show negative correlations with **trip duration** (≈ -0.24 and -0.18 respectively), which may suggest that trips moving further north or south could generally be shorter in duration.

- **Passenger count** has a very weak correlation with **trip duration** (≈ 0.015), indicating that the number of passengers does not significantly affect how long a trip lasts.

- The **pickup and dropoff coordinates** (longitude and latitude) are interrelated to varying degrees, reflecting their geographical connection in defining trip paths across the city.

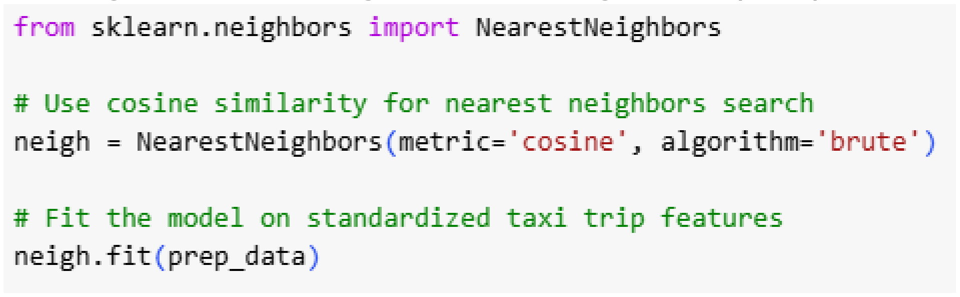


	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
passenger_count	1	0.0047	-0.0047	0.00053	-0.0018	0.015
pickup_longitude	0.0047	1	-0.15	0.27	0.036	0.36
pickup_latitude	-0.0047	-0.15	1	0.054	0.44	-0.24
dropoff_longitude	0.00053	0.27	0.054	1	0.096	0.26
dropoff_latitude	-0.0018	0.036	0.44	0.096	1	-0.18
trip_duration	0.015	0.36	-0.24	0.26	-0.18	1

5. Normalising data using z-score normalisation

```
from sklearn.preprocessing import StandardScaler

# Define the columns you want to scale
num_cols = [
    'passenger_count', 'pickup_longitude', 'pickup_latitude',
    'dropoff_longitude', 'dropoff_latitude', 'trip_duration'
]

# Initialize and fit the scaler
scaler = StandardScaler()
prep_data = scaler.fit_transform(extracted_data[num_cols].to_numpy())
```

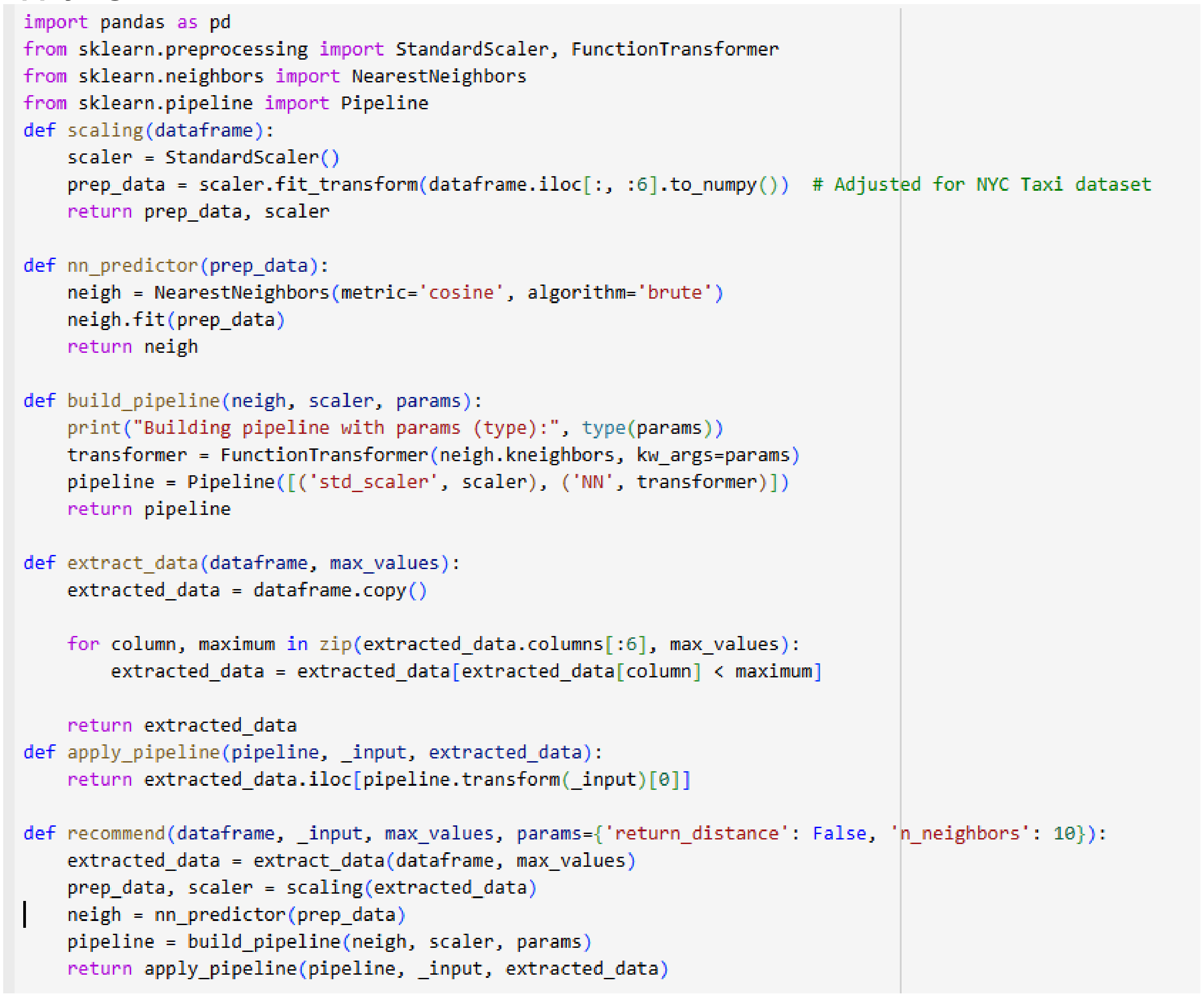
6. **Training the model using K Nearest Neighbours (KNN)**



```
from sklearn.neighbors import NearestNeighbors

# Use cosine similarity for nearest neighbors search
neigh = NearestNeighbors(metric='cosine', algorithm='brute')

# Fit the model on standardized taxi trip features
neigh.fit(prep_data)
```

Here, the metric is set to **'cosine'**, meaning the model will measure the cosine similarity between data points. The **'brute'** algorithm is being used, which computes all pairwise distances between data points without optimization for speed.

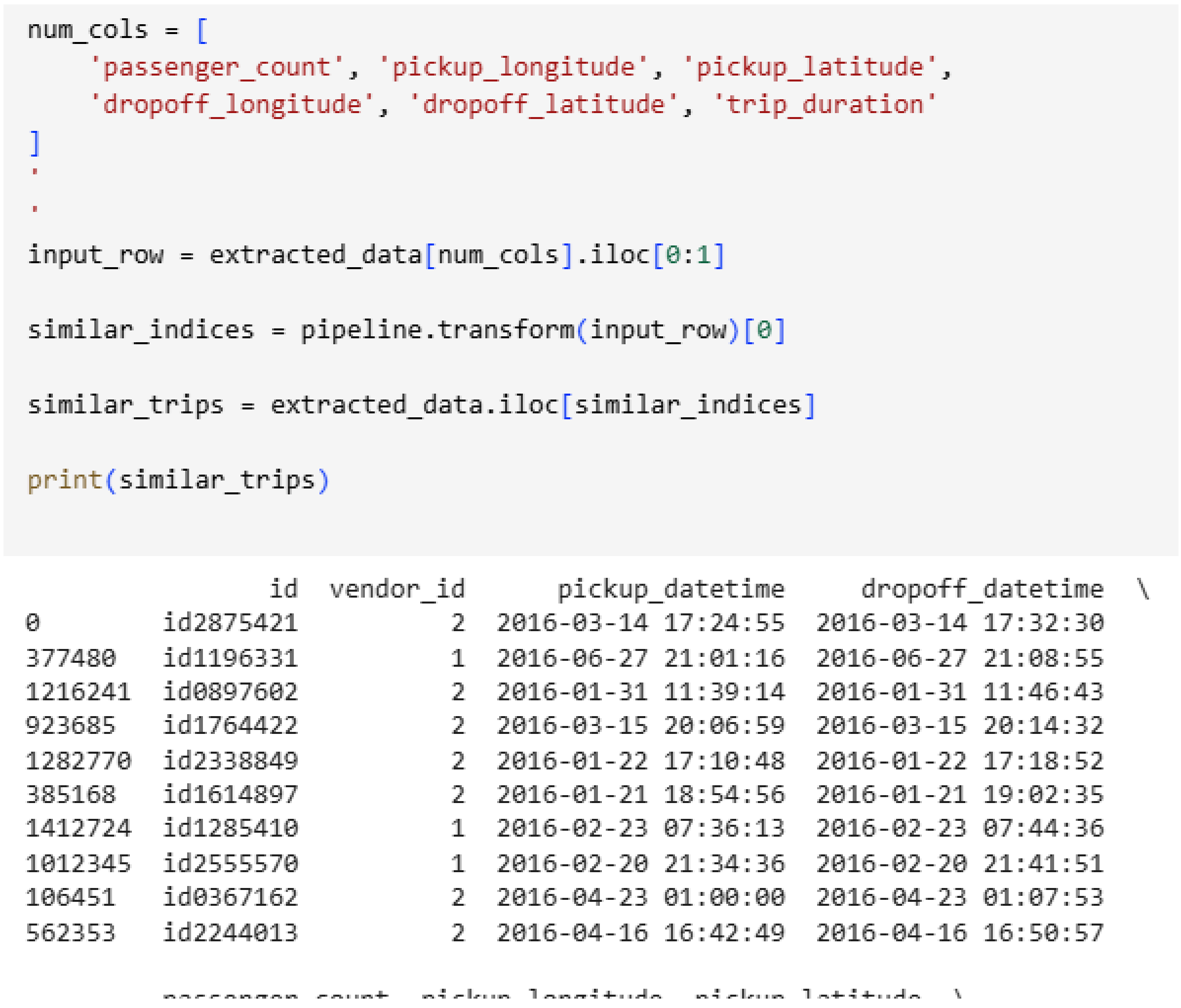
7. **Applying KNN**



```
import pandas as pd
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.neighbors import NearestNeighbors
from sklearn.pipeline import Pipeline
def scaling(dataframe):
    scaler = StandardScaler()
    prep_data = scaler.fit_transform(dataframe.iloc[:, :6].to_numpy())  # Adjusted for NYC Taxi dataset
    return prep_data, scaler

def nn_predictor(prep_data):
    neigh = NearestNeighbors(metric='cosine', algorithm='brute')
    neigh.fit(prep_data)
    return neigh

def build_pipeline(neigh, scaler, params):
    print("Building pipeline with params (type):", type(params))
    transformer = FunctionTransformer(neigh.kneighbors, kw_args=params)
    pipeline = Pipeline([('std_scaler', scaler), ('NN', transformer)])
    return pipeline

def extract_data(dataframe, max_values):
    extracted_data = dataframe.copy()

    for column, maximum in zip(extracted_data.columns[:6], max_values):
        extracted_data = extracted_data[extracted_data[column] < maximum]

    return extracted_data
def apply_pipeline(pipeline, _input, extracted_data):
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommend(dataframe, _input, max_values, params={'return_distance': False, 'n_neighbors': 10}):
    extracted_data = extract_data(dataframe, max_values)
    prep_data, scaler = scaling(extracted_data)
    neigh = nn_predictor(prep_data)
    pipeline = build_pipeline(neigh, scaler, params)
    return apply_pipeline(pipeline, _input, extracted_data)
```

8. **Testing the model**

NAME:ATHARVA SHINDE CLASS:D15C ROLL.NO:54

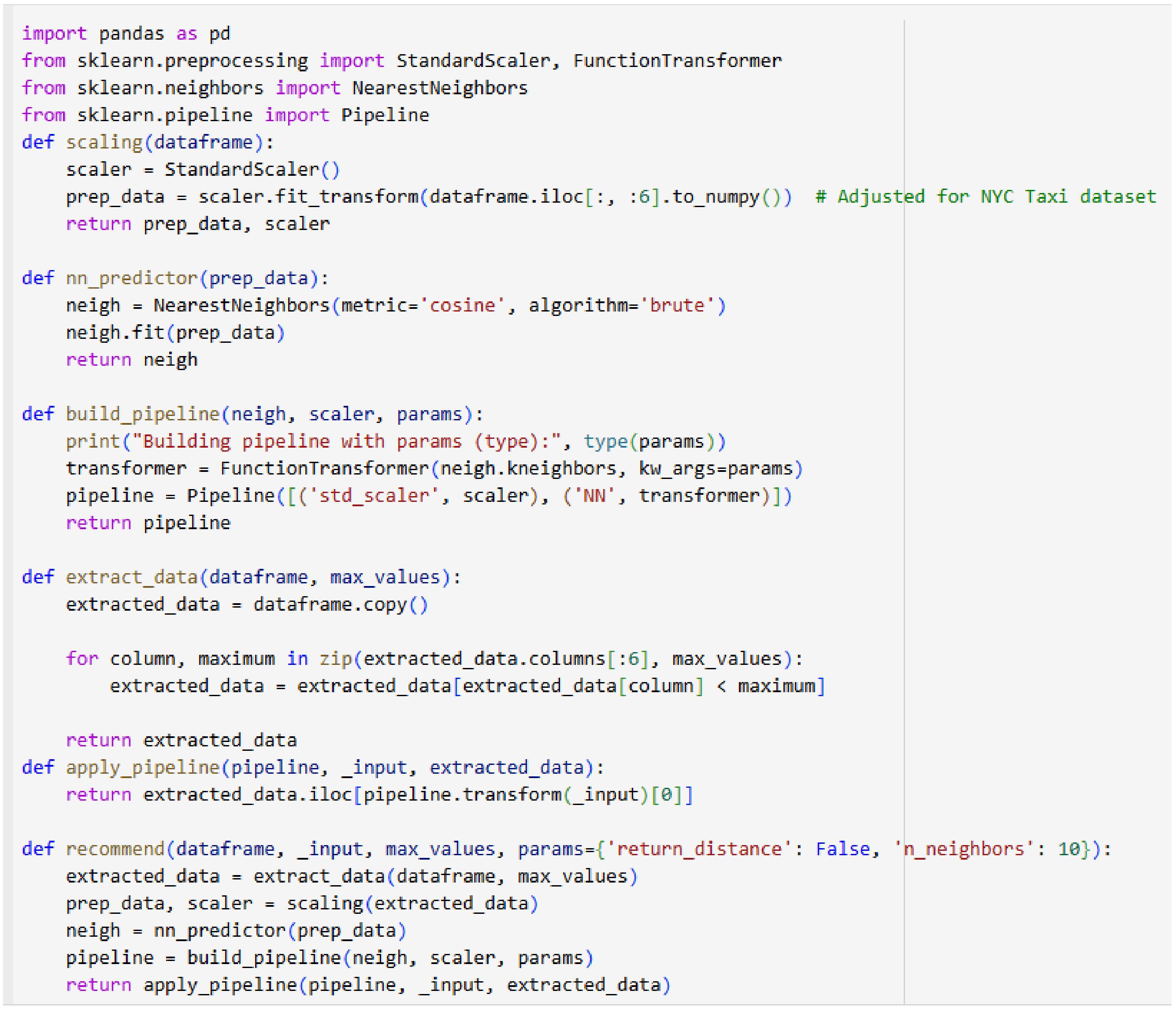


```
num_cols = [
    'passenger_count', 'pickup_longitude', 'pickup_latitude',
    'dropoff_longitude', 'dropoff_latitude', 'trip_duration'
]
'
'

input_row = extracted_data[num_cols].iloc[0:1]

similar_indices = pipeline.transform(input_row)[0]

similar_trips = extracted_data.iloc[similar_indices]

print(similar_trips)
```

```
                id  vendor_id     pickup_datetime    dropoff_datetime  \
0        id2875421          2 2016-03-14 17:24:55 2016-03-14 17:32:30
377480   id1196331          1 2016-06-27 21:01:16 2016-06-27 21:08:55
1216241  id0897602          2 2016-01-31 11:39:14 2016-01-31 11:46:43
923685   id1764422          2 2016-03-15 20:06:59 2016-03-15 20:14:32
1282770  id2338849          2 2016-01-22 17:10:48 2016-01-22 17:18:52
385168   id1614897          2 2016-01-21 18:54:56 2016-01-21 19:02:35
1412724  id1285410          1 2016-02-23 07:36:13 2016-02-23 07:44:36
1012345  id2555570          1 2016-02-20 21:34:36 2016-02-20 21:41:51
106451   id0367162          2 2016-04-23 01:00:00 2016-04-23 01:07:53
562353   id2244013          2 2016-04-16 16:42:49 2016-04-16 16:50:57

         passenger_count  pickup_longitude  pickup_latitude  \
```

9. Creating functions for all

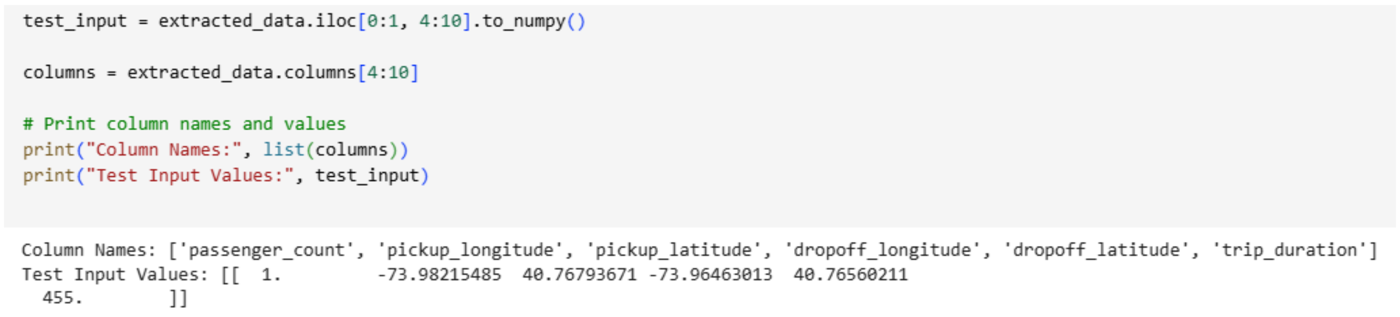
NAME:ATHARVA SHINDE CLASS:D15C ROLL.NO:54



```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.neighbors import NearestNeighbors
from sklearn.pipeline import Pipeline
def scaling(dataframe):
    scaler = StandardScaler()
    prep_data = scaler.fit_transform(dataframe.iloc[:, :6].to_numpy())  # Adjusted for NYC Taxi dataset
    return prep_data, scaler

def nn_predictor(prep_data):
    neigh = NearestNeighbors(metric='cosine', algorithm='brute')
    neigh.fit(prep_data)
    return neigh

def build_pipeline(neigh, scaler, params):
    print("Building pipeline with params (type):", type(params))
    transformer = FunctionTransformer(neigh.kneighbors, kw_args=params)
    pipeline = Pipeline([('std_scaler', scaler), ('NN', transformer)])
    return pipeline

def extract_data(dataframe, max_values):
    extracted_data = dataframe.copy()

    for column, maximum in zip(extracted_data.columns[:6], max_values):
        extracted_data = extracted_data[extracted_data[column] < maximum]

    return extracted_data
def apply_pipeline(pipeline, _input, extracted_data):
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommend(dataframe, _input, max_values, params={'return_distance': False, 'n_neighbors': 10}):
    extracted_data = extract_data(dataframe, max_values)
    prep_data, scaler = scaling(extracted_data)
    neigh = nn_predictor(prep_data)
    pipeline = build_pipeline(neigh, scaler, params)
    return apply_pipeline(pipeline, _input, extracted_data)
```

NAME:ATHARVA SHINDE CLASS:D15C ROLL.NO:54

10. Testing Recommendation with custom nutritional values



```
test_input = extracted_data.iloc[0:1, 4:10].to_numpy()

columns = extracted_data.columns[4:10]

# Print column names and values
print("Column Names:", list(columns))
print("Test Input Values:", test_input)
```

```
Column Names: ['passenger_count', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'trip_duration']
Test Input Values: [[  1.         -73.98215485  40.76793671 -73.96463013  40.76560211
  455.        ]]
```

NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

Conclusion:

The analysis of the NYC Taxi dataset revealed valuable insights into trip patterns and influential factors affecting trip duration. The Q-Q plot highlighted a significant right-skew in the trip duration distribution, indicating that while most taxi trips are short, a few long-duration trips notably impact the overall distribution. Correlation analysis further showed that geographical features, particularly pickup and dropoff coordinates, have a moderate relationship with trip duration, suggesting that location plays a key role in travel time. Passenger count, however, showed minimal correlation, indicating it does not significantly influence trip duration. These findings form a foundation for building predictive models and optimizing route or fare estimation systems, while also pointing toward potential enhancements using geospatial clustering or traffic data integration.

NAME:Shivpratik Hande **CLASS:D15C** **ROLL.NO:14**

Experiment 9

Aim: To perform Exploratory data analysis using Apache Spark and Pandas

Theory:

1. What is Apache Spark and How Does It Work?

Apache Spark is a powerful open-source framework designed for distributed big data processing. It significantly outperforms traditional Hadoop MapReduce by utilizing **in-memory computation**, which makes it ideal for tasks involving repeated operations such as machine learning, data analytics, and graph-based processing.

🔹 **Core Components of Spark:**

- **Spark Core**: The foundation for all Spark functionalities, handling basic I/O and task scheduling.
- **Spark SQL**: Offers support for structured data through SQL queries and DataFrames.
- **MLlib**: A library that provides scalable machine learning algorithms.
- **GraphX**: Enables graph processing and analytics.
- **Spark Streaming**: Processes live data streams in real time.

🔹 **Working of Apache Spark:**

- Spark primarily handles data in **RDDs (Resilient Distributed Datasets)** or **DataFrames**.
- A **Driver Program** initiates the process by creating a **SparkContext**, which connects with the **Cluster Manager**.
- Spark then assigns tasks to **Executors**, which run in parallel across the cluster.
- It uses **lazy evaluation**, meaning transformations are not executed until an action is triggered.

2. How is Data Exploration Performed in Apache Spark?

NAME:Shivpratik Hande **CLASS:D15C** **ROLL.NO:14**

Exploratory Data Analysis (EDA) in Spark follows a pattern similar to pandas but is optimized for handling large-scale datasets spread across distributed environments.

- **Steps for EDA in Spark:**

1. **Spark Session Initialization**
   Start by importing PySpark and creating a `SparkSession` using `SparkSession.builder`. This session acts as the gateway to all Spark operations.

2. **Loading the Dataset**
   Use methods like `spark.read.csv()` or `spark.read.json()` to import data into a Spark DataFrame. It's recommended to use `header=True` and `inferSchema=True` to automatically detect column names and data types.

3. **Inspecting the Dataset**
   Use `.printSchema()` to understand the structure and data types. Use `.show()` to preview records and `.describe()` for statistical summaries such as count, mean, and standard deviation.

4. **Handling Missing Data**
   Missing or null values can be handled using `df.na.drop()` to remove them or `df.na.fill("value")` to replace them with default values. This step is essential for ensuring clean and reliable data.

5. **Data Transformation**
   Use functions like `.withColumn()`, `.filter()`, and `.groupBy()` to transform and manipulate the data. These allow you to prepare the dataset for deeper analysis.

6. **Visualizing Data**
   To plot data, convert the Spark DataFrame into a pandas DataFrame using `.toPandas()`. Then use libraries like **matplotlib** or **seaborn** for visualization.

7. **Correlation and Insights**
   You can calculate correlations using pandas' `.corr()` or Spark's `Correlation.corr()` from MLlib. Use groupings and pivot tables to uncover trends and relationships in the data.

Conclusion:

In this experiment, I gained practical experience with **Exploratory Data Analysis using Apache Spark**. I learned how to set up a Spark session, efficiently load large datasets, and explore their structure using functions like `.show()`, `.printSchema()`, and `.describe()`. I also practiced handling null values and transforming data using key Spark DataFrame operations. To visualize data, I used the `.toPandas()` method for compatibility with popular Python plotting libraries. Finally, I explored correlation analysis and data summarization to uncover insights. Overall, this experiment deepened my understanding of how Apache Spark can scale data analysis across massive datasets while complementing tools like pandas and seaborn for deeper insights.

NAME:SHIVPRATIK HANDE **CLASS:D15C** **ROLL.NO:14**

Experiment-10

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

Theory:

What is Streaming? Explain Batch and Stream Data:

Streaming refers to the real-time processing of data as it is generated. This approach is essential for applications that require instant insights or decisions, such as fraud detection systems, stock trading platforms, and live analytics dashboards. Streaming data is continuous, time-sensitive, and typically comes in an unbounded form that never stops.

On the other hand, **batch processing** involves collecting and storing data over a fixed interval, and then processing it in bulk. It is commonly used in traditional data processing scenarios such as ETL (Extract, Transform, Load), report generation, and data aggregation tasks. The data is processed in defined blocks or batches at scheduled times.

Examples:

- **Batch:** Compiling quarterly revenue reports.
- **Stream:** Analyzing live website user activity.

How Data Streaming Works with Apache Spark:

Apache Spark supports stream processing using its **Structured Streaming** module. This engine enables developers to process real-time data streams using the same high-level APIs as used for batch data, making it easier to build consistent and maintainable pipelines.

Structured Streaming conceptualizes data streams as continuously growing tables and performs computations incrementally as new data arrives. Spark can pull data from various streaming sources such as Apache Kafka, TCP/IP sockets, file systems, or cloud storage platforms.

After ingestion, Spark processes the data using familiar transformations like `filter`, `select`, and `groupBy`, along with advanced operations such as **windowing**, **watermarking** for handling late data, and **checkpointing** to ensure resilience and data recovery in case of failure.

Spark processes streaming data in **micro-batches**, meaning it treats small chunks of streaming data like mini batch jobs, ensuring both high throughput and low latency. The processed results can be stored in sinks like HDFS, databases, or displayed on live dashboards.

Key Highlights:

- Unified API for both real-time and batch data
- Ability to manage application state over time
- Compatible with structured data sources like Kafka
- Designed for scalability and fault tolerance

Real-World Use Cases:

- Detecting anomalies in banking transactions
- Analyzing server logs as they're generated
- Monitoring trends on social media in real-time

Conclusion:

Through this exploration, I gained a clear understanding of the contrast between batch and streaming data models. Batch is well-suited for scheduled, large-scale jobs, while streaming is ideal for situations requiring immediate analysis. Apache Spark's Structured Streaming provides a streamlined and unified solution to handle both styles of data processing. I learned how to ingest live data, transform it, and deliver insights in real time, making Spark a highly effective tool for building robust and intelligent data pipelines in modern data ecosystems.