**Name:** Pratik Patil **Roll no:** 40 **Class:**D15C

# EXPERIMENT NO: 1

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

## Theory:

Data preparation is a fundamental step in data science, involving the cleaning and transformation of raw data into a structured and analyzable format. Pandas, a powerful Python library, provides efficient tools for handling missing values, encoding categorical data, and scaling numerical features. Proper preprocessing enhances dataset quality, ensuring consistency and reliability for further analysis and machine learning models.

## Problem Statement:

The Placement Data dataset contains various attributes related to students' academic performance, placement status, and salary packages. The objective of this experiment is to:

- Identify key trends in student placements based on academic performance.
- Analyze the distribution of salary packages.
- Handle missing data and remove inconsistencies.
- Standardize and normalize the data for further analysis.

By cleaning the placement dataset and applying data preprocessing steps, the goal is to improve data reliability, analyze student performance trends, and provide valuable insights for academic and recruitment decisions.

## Dataset Overview:

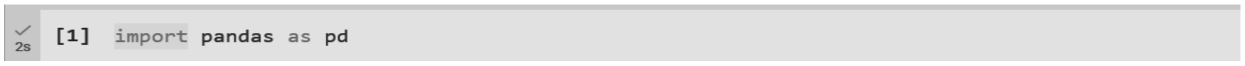
The dataset provides detailed information about student placements, academic performance, and salary distributions. It contains multiple columns, each capturing specific attributes related to students' educational backgrounds and employment outcomes. Below is a breakdown of the columns and their relevance: The dataset provides insights into student placements, containing columns such as:

1. **StudentID:** Unique identifier for each student.
2. **CGPA:** Cumulative Grade Point Average of the student.
3. **Internships:** Number of internships completed.
4. **Projects:** Number of academic or industry projects undertaken.
5. **Workshops/Certifications:** Number of workshops attended or certifications earned.
6. **Aptitude Test Score:** Score obtained in the aptitude test.
7. **Soft Skills Rating:** Rating of soft skills on a predefined scale.
8. **Extracurricular Activities:** Participation in extracurricular activities.
9. **Placement Training:** Whether the student underwent placement training (Yes/No).
10. **SSC Marks:** Secondary school exam scores.
11. **HSC Marks:** Higher secondary exam scores.

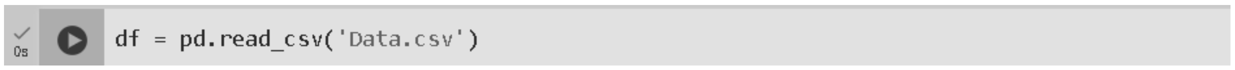
12. **Placement Status:** Whether the student was placed or not.

**Steps:**

**Loading The Dataset**



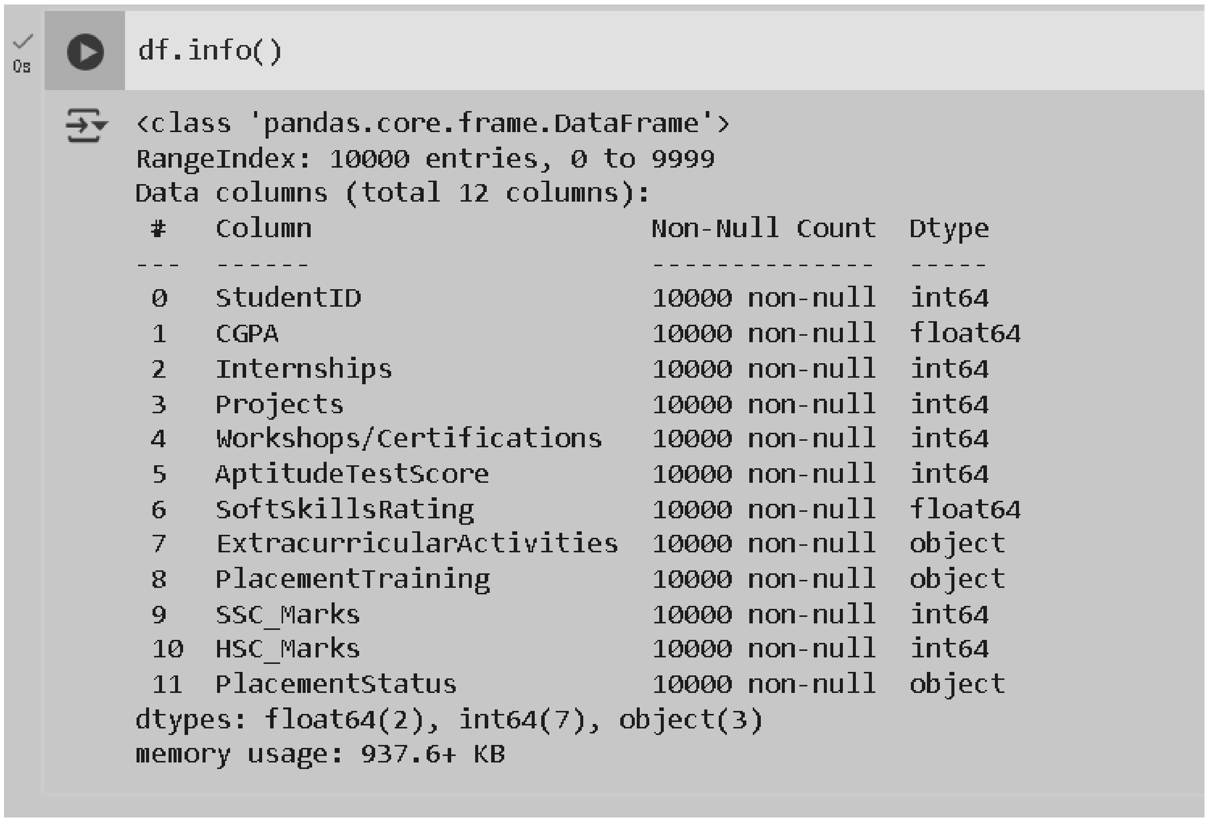
```
[1] import pandas as pd
```



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

**Description of the dataset**

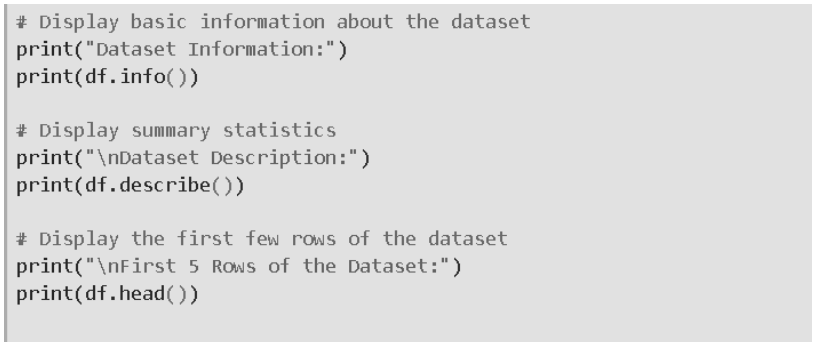
a. **Information about dataset**



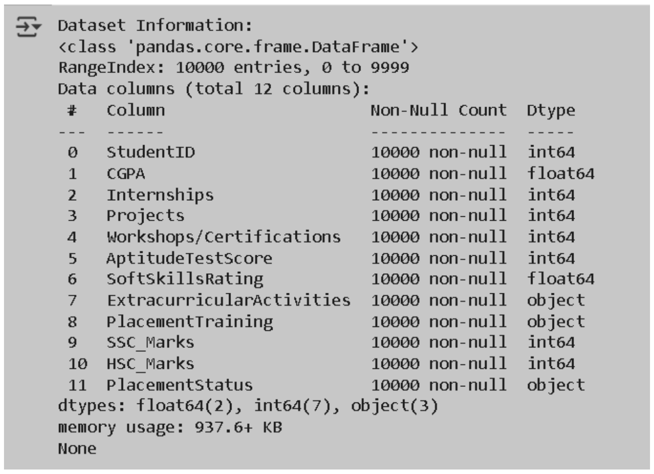
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   StudentID                   10000 non-null  int64  
 1   CGPA                        10000 non-null  float64
 2   Internships                 10000 non-null  int64  
 3   Projects                    10000 non-null  int64  
 4   Workshops/Certifications    10000 non-null  int64  
 5   AptitudeTestScore           10000 non-null  int64  
 6   SoftSkillsRating            10000 non-null  float64
 7   ExtracurricularActivities   10000 non-null  object 
 8   PlacementTraining           10000 non-null  object 
 9   SSC_Marks                   10000 non-null  int64  
 10  HSC_Marks                   10000 non-null  int64  
 11  PlacementStatus             10000 non-null  object 
dtypes: float64(2), int64(7), object(3)
memory usage: 937.6+ KB
```

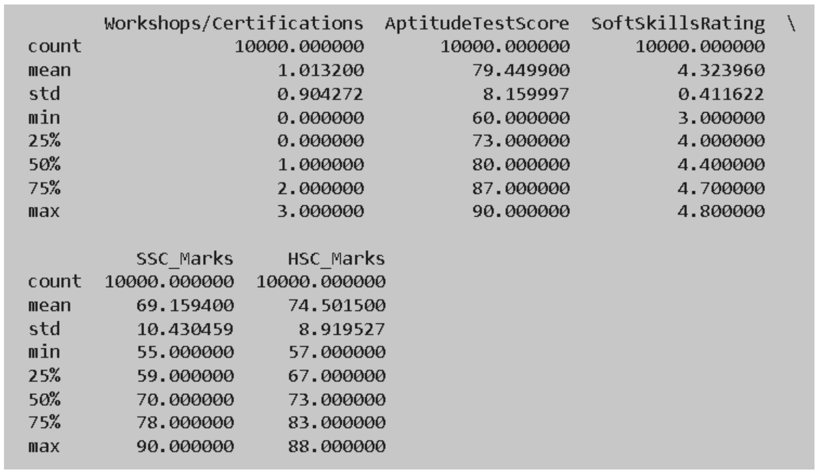
## b. Description of Dataset



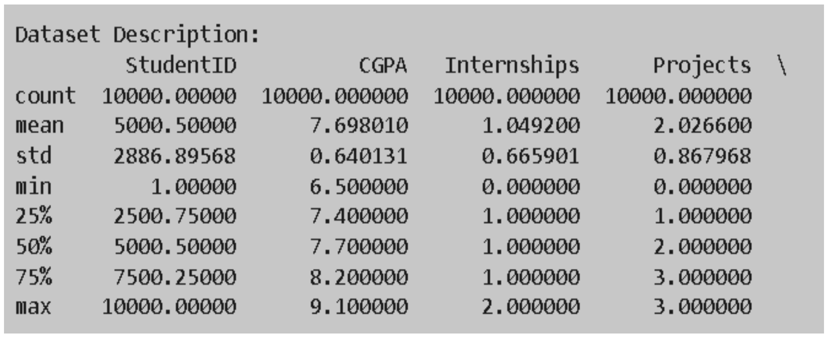
```
# Display basic information about the dataset
print("Dataset Information:")
print(df.info())

# Display summary statistics
print("\nDataset Description:")
print(df.describe())

# Display the first few rows of the dataset
print("\nFirst 5 Rows of the Dataset:")
print(df.head())
```



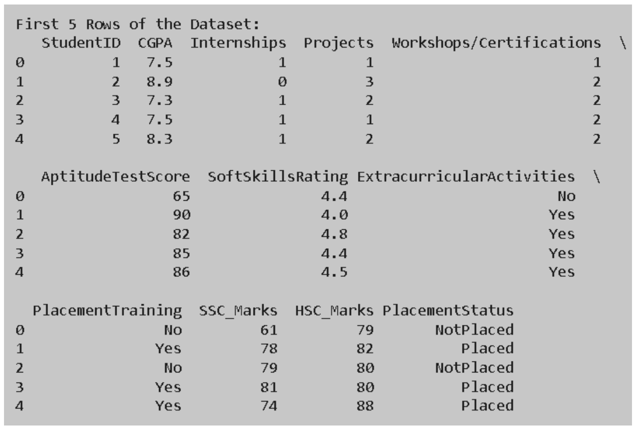
```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   StudentID                  10000 non-null  int64
 1   CGPA                       10000 non-null  float64
 2   Internships                10000 non-null  int64
 3   Projects                   10000 non-null  int64
 4   Workshops/Certifications   10000 non-null  int64
 5   AptitudeTestScore          10000 non-null  int64
 6   SoftSkillsRating           10000 non-null  float64
 7   ExtracurricularActivities  10000 non-null  object
 8   PlacementTraining          10000 non-null  object
 9   SSC_Marks                  10000 non-null  int64
 10  HSC_Marks                  10000 non-null  int64
 11  PlacementStatus            10000 non-null  object
dtypes: float64(2), int64(7), object(3)
memory usage: 937.6+ KB
None
```



```
       Workshops/Certifications  AptitudeTestScore  SoftSkillsRating  \
count              10000.000000       10000.000000      10000.000000
mean                   1.013200          79.449900          4.323960
std                    0.904272           8.159997          0.411622
min                    0.000000          60.000000          3.000000
25%                    0.000000          73.000000          4.000000
50%                    1.000000          80.000000          4.400000
75%                    2.000000          87.000000          4.700000
max                    3.000000          90.000000          4.800000

          SSC_Marks     HSC_Marks
count  10000.000000  10000.000000
mean      69.159400     74.501500
std       10.430459      8.919527
min       55.000000     57.000000
25%       59.000000     67.000000
50%       70.000000     73.000000
75%       78.000000     83.000000
max       90.000000     88.000000
```

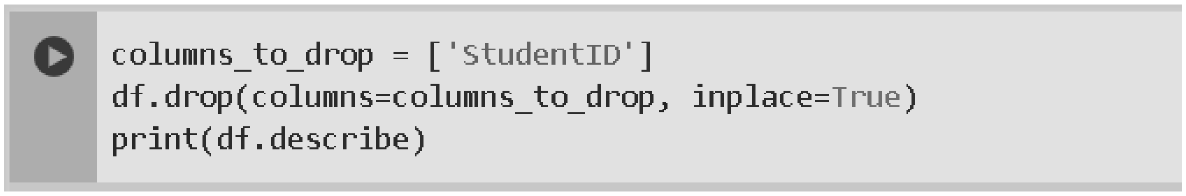


```
Dataset Description:
         StudentID          CGPA   Internships      Projects  \
count  10000.00000  10000.000000  10000.000000  10000.000000
mean    5000.50000      7.698010      1.049200      2.026600
std     2886.89568      0.640131      0.665901      0.867968
min        1.00000      6.500000      0.000000      0.000000
25%     2500.75000      7.400000      1.000000      1.000000
50%     5000.50000      7.700000      1.000000      2.000000
75%     7500.25000      8.200000      1.000000      3.000000
max    10000.00000      9.100000      2.000000      3.000000
```

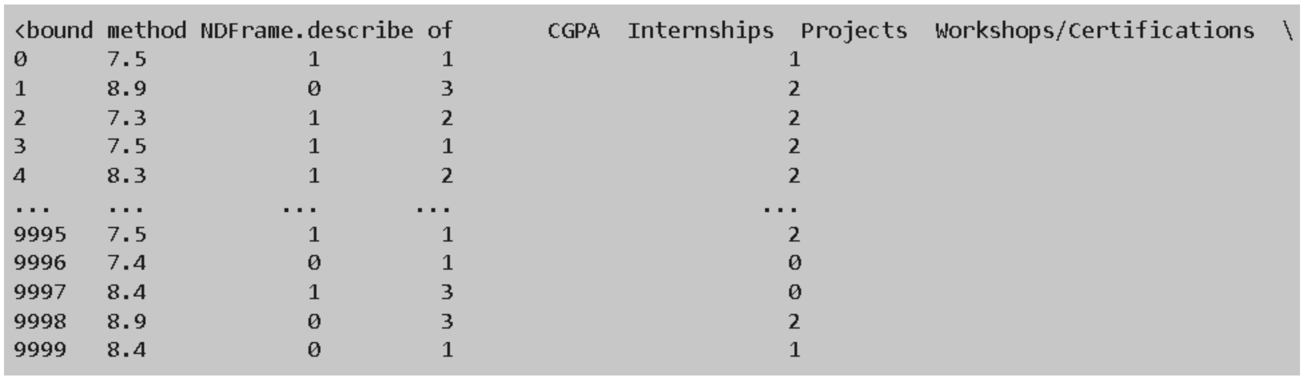


```
First 5 Rows of the Dataset:
   StudentID  CGPA  Internships  Projects  Workshops/Certifications  \
0          1   7.5            1         1                         1
1          2   8.9            0         3                         2
2          3   7.3            1         2                         2
3          4   7.5            1         1                         2
4          5   8.3            1         2                         2

   AptitudeTestScore  SoftSkillsRating ExtracurricularActivities  \
0                 65               4.4                        No
1                 90               4.0                       Yes
2                 82               4.8                       Yes
3                 85               4.4                       Yes
4                 86               4.5                       Yes

  PlacementTraining  SSC_Marks  HSC_Marks PlacementStatus
0                No         61         79       NotPlaced
1               Yes         78         82          Placed
2                No         79         80       NotPlaced
3               Yes         81         80          Placed
4               Yes         74         88          Placed
```

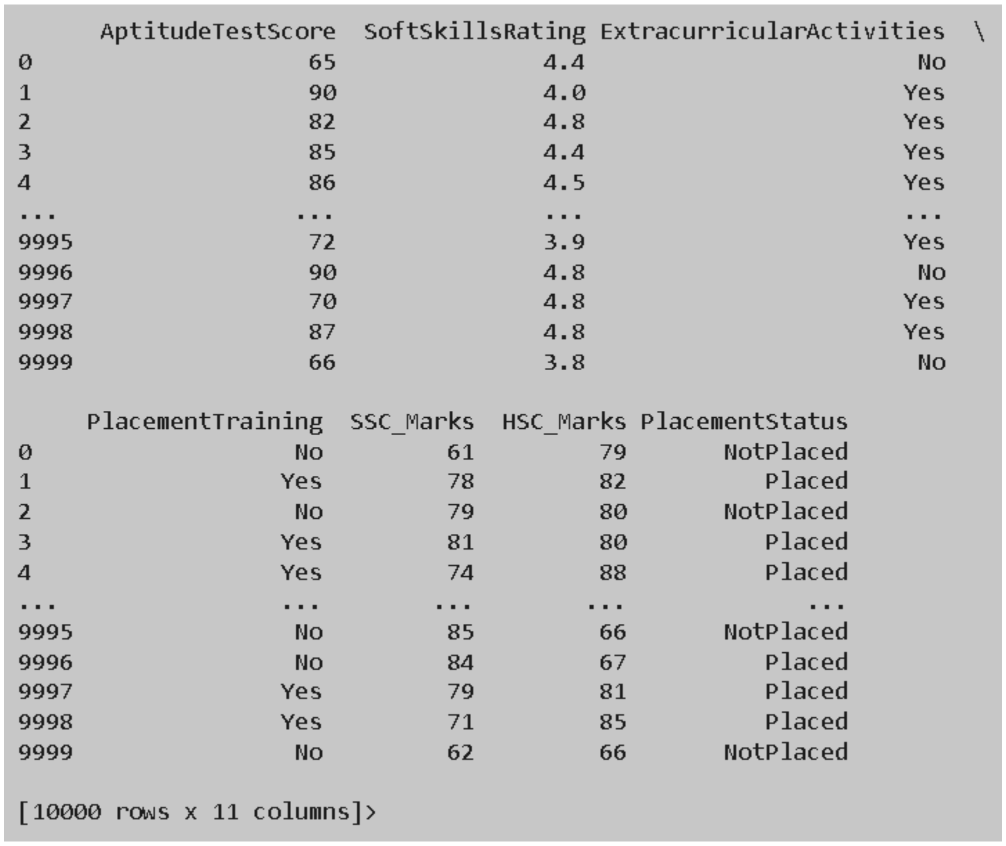
**Drop columns that aren't useful.**



```
columns_to_drop = ['StudentID']
df.drop(columns=columns_to_drop, inplace=True)
print(df.describe)
```

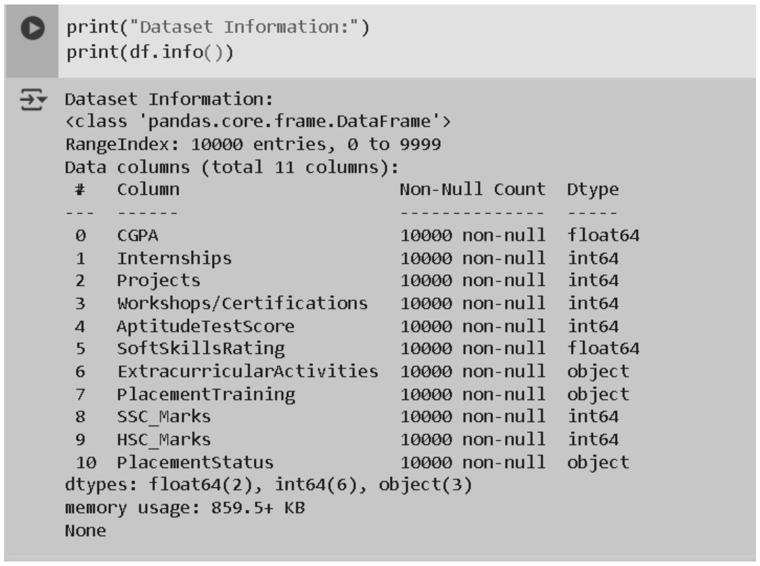


```
<bound method NDFrame.describe of       CGPA  Internships  Projects  Workshops/Certifications  \
0      7.5            1         1                         1
1      8.9            0         3                         2
2      7.3            1         2                         2
3      7.5            1         1                         2
4      8.3            1         2                         2
...    ...          ...       ...                       ...
9995   7.5            1         1                         2
9996   7.4            0         1                         0
9997   8.4            1         3                         0
9998   8.9            0         3                         2
9999   8.4            0         1                         1
```



```
      AptitudeTestScore  SoftSkillsRating ExtracurricularActivities  \
0                    65               4.4                        No
1                    90               4.0                       Yes
2                    82               4.8                       Yes
3                    85               4.4                       Yes
4                    86               4.5                       Yes
...                 ...               ...                       ...
9995                 72               3.9                       Yes
9996                 90               4.8                        No
9997                 70               4.8                       Yes
9998                 87               4.8                       Yes
9999                 66               3.8                        No

     PlacementTraining  SSC_Marks  HSC_Marks PlacementStatus
0                   No         61         79       NotPlaced
1                  Yes         78         82          Placed
2                   No         79         80       NotPlaced
3                  Yes         81         80          Placed
4                  Yes         74         88          Placed
...                ...        ...        ...             ...
9995                No         85         66       NotPlaced
9996                No         84         67          Placed
9997               Yes         79         81          Placed
9998               Yes         71         85          Placed
9999                No         62         66       NotPlaced

[10000 rows x 11 columns]>
```

**Thus the columns now present in dataset are:**

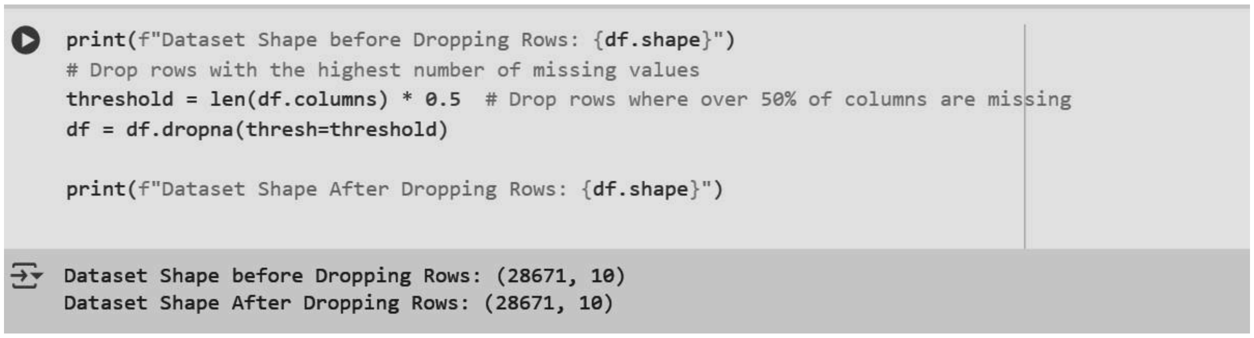


```
print("Dataset Information:")
print(df.info())
```

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   CGPA                       10000 non-null  float64
 1   Internships                10000 non-null  int64
 2   Projects                   10000 non-null  int64
 3   Workshops/Certifications   10000 non-null  int64
 4   AptitudeTestScore          10000 non-null  int64
 5   SoftSkillsRating           10000 non-null  float64
 6   ExtracurricularActivities  10000 non-null  object
 7   PlacementTraining          10000 non-null  object
 8   SSC_Marks                  10000 non-null  int64
 9   HSC_Marks                  10000 non-null  int64
 10  PlacementStatus            10000 non-null  object
dtypes: float64(2), int64(6), object(3)
memory usage: 859.5+ KB
None
```

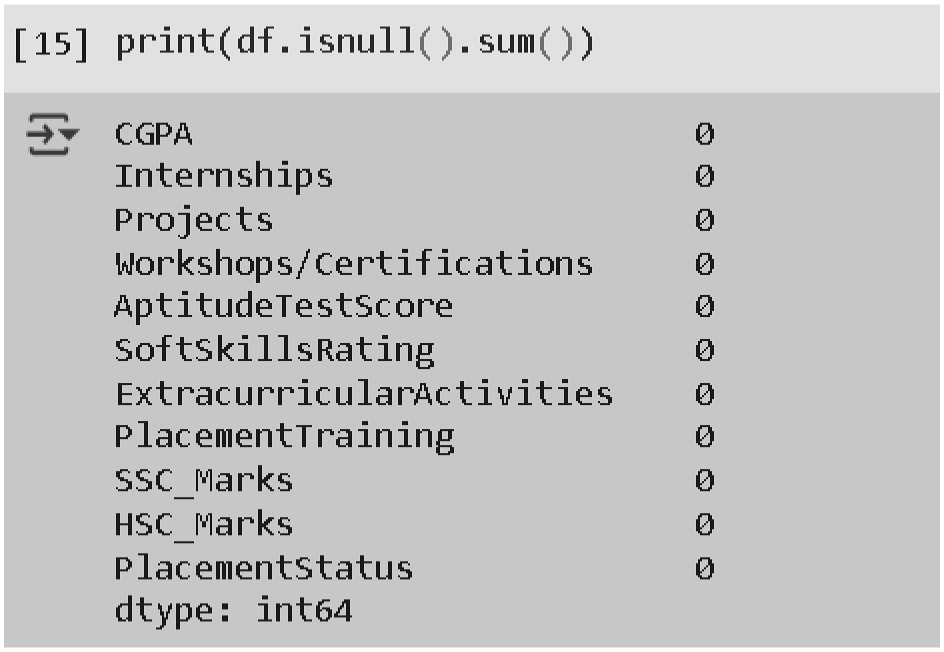
**Take care of missing data.**

**c. Drop rows with maximum missing values.**



```
print(f"Dataset Shape before Dropping Rows: {df.shape}")
# Drop rows with the highest number of missing values
threshold = len(df.columns) * 0.5  # Drop rows where over 50% of columns are missing
df = df.dropna(thresh=threshold)

print(f"Dataset Shape After Dropping Rows: {df.shape}")
```

```
Dataset Shape before Dropping Rows: (28671, 10)
Dataset Shape After Dropping Rows: (28671, 10)
```



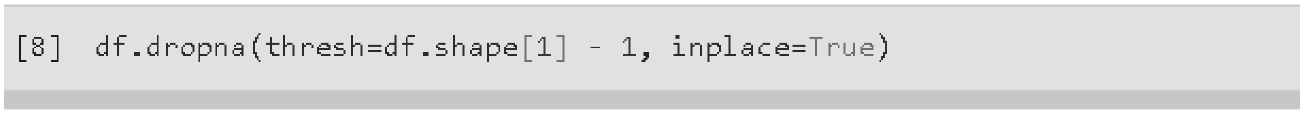
```
[15] print(df.isnull().sum())
```

```
CGPA                          0
Internships                   0
Projects                      0
Workshops/Certifications      0
AptitudeTestScore             0
SoftSkillsRating              0
ExtracurricularActivities     0
PlacementTraining             0
SSC_Marks                     0
HSC_Marks                     0
PlacementStatus               0
dtype: int64
```

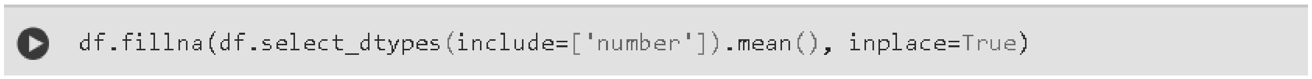
**The output of print(df.isnull().sum()) indicates that there are no missing values in any of the columns of the dataset. Each column has a count of 0, meaning that all records have valid data for every feature.**

## Data Preprocessing

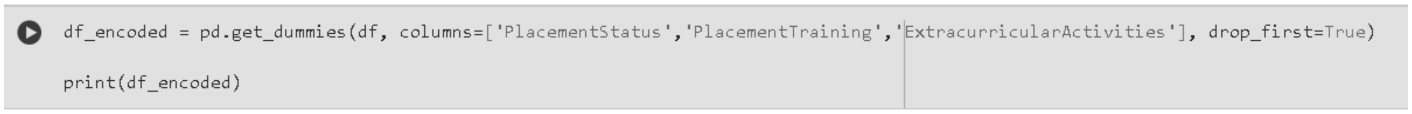
- **The code uses dropna() to remove missing values.**
- **fillna() is used to fill missing values with the mean, specifically for numerical columns.**
- **get_dummies() is applied to categorical variables like Placement Status, Placement Training, and Extracurricular Activities, with drop_first=True to avoid multicollinearity.**



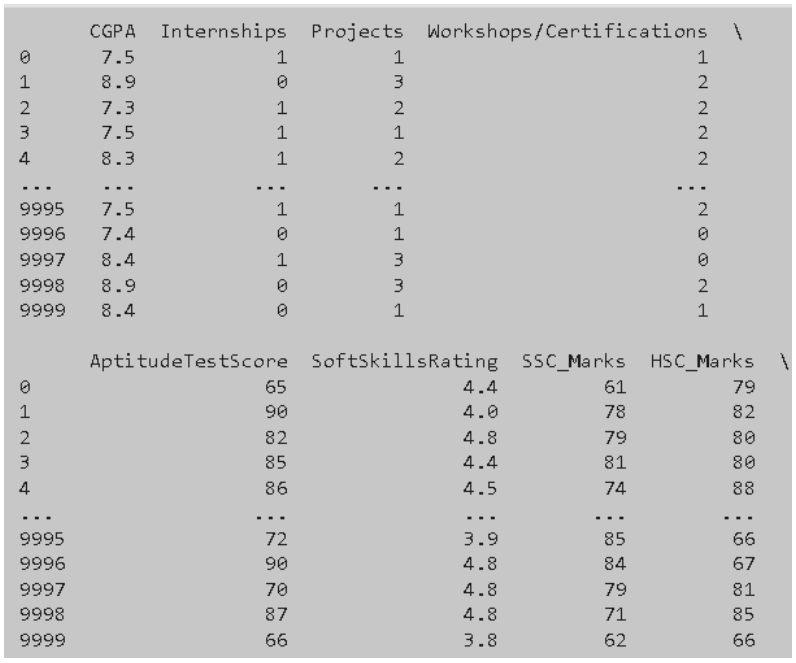
```
[8]  df.dropna(thresh=df.shape[1] - 1, inplace=True)
```



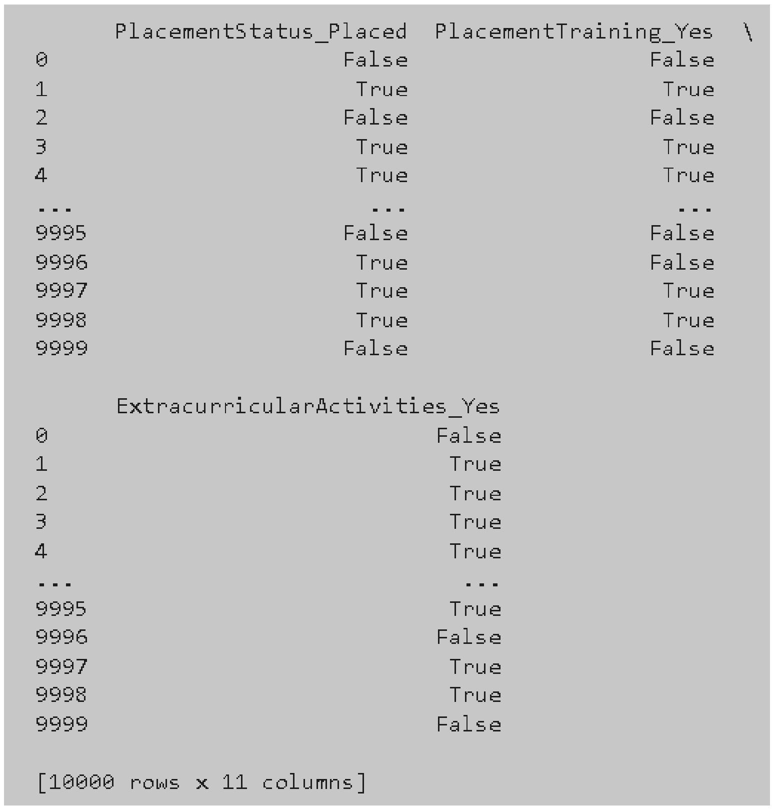
```
df.fillna(df.select_dtypes(include=['number']).mean(), inplace=True)
```



```
df_encoded = pd.get_dummies(df, columns=['PlacementStatus','PlacementTraining','ExtracurricularActivities'], drop_first=True)

print(df_encoded)
```



```
      CGPA  Internships  Projects  Workshops/Certifications  \
0      7.5            1         1                         1
1      8.9            0         3                         2
2      7.3            1         2                         2
3      7.5            1         1                         2
4      8.3            1         2                         2
...    ...          ...       ...                       ...
9995   7.5            1         1                         2
9996   7.4            0         1                         0
9997   8.4            1         3                         0
9998   8.9            0         3                         2
9999   8.4            0         1                         1

      AptitudeTestScore  SoftSkillsRating  SSC_Marks  HSC_Marks  \
0                    65               4.4         61         79
1                    90               4.0         78         82
2                    82               4.8         79         80
3                    85               4.4         81         80
4                    86               4.5         74         88
...                 ...               ...        ...        ...
9995                 72               3.9         85         66
9996                 90               4.8         84         67
9997                 70               4.8         79         81
9998                 87               4.8         71         85
9999                 66               3.8         62         66
```



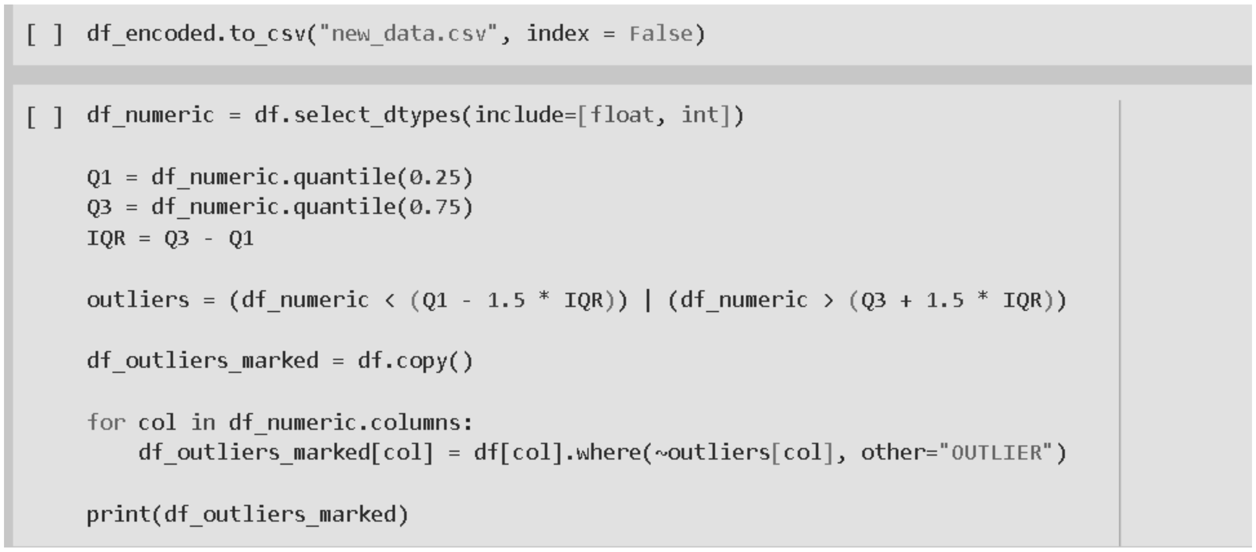
```
      PlacementStatus_Placed  PlacementTraining_Yes  \
0                      False                  False
1                       True                   True
2                      False                  False
3                       True                   True
4                       True                   True
...                      ...                    ...
9995                   False                  False
9996                    True                  False
9997                    True                   True
9998                    True                   True
9999                   False                  False

      ExtracurricularActivities_Yes
0                             False
1                              True
2                              True
3                              True
4                              True
...                             ...
9995                           True
9996                          False
9997                           True
9998                           True
9999                          False

[10000 rows x 11 columns]
```

**Dataset Features**

- **The dataset includes Aptitude Test Scores, Soft Skills Ratings, SSC Marks, HSC Marks, Internships, Projects, and Workshops/Certifications.**
- **It tracks whether a student is placed or not.**

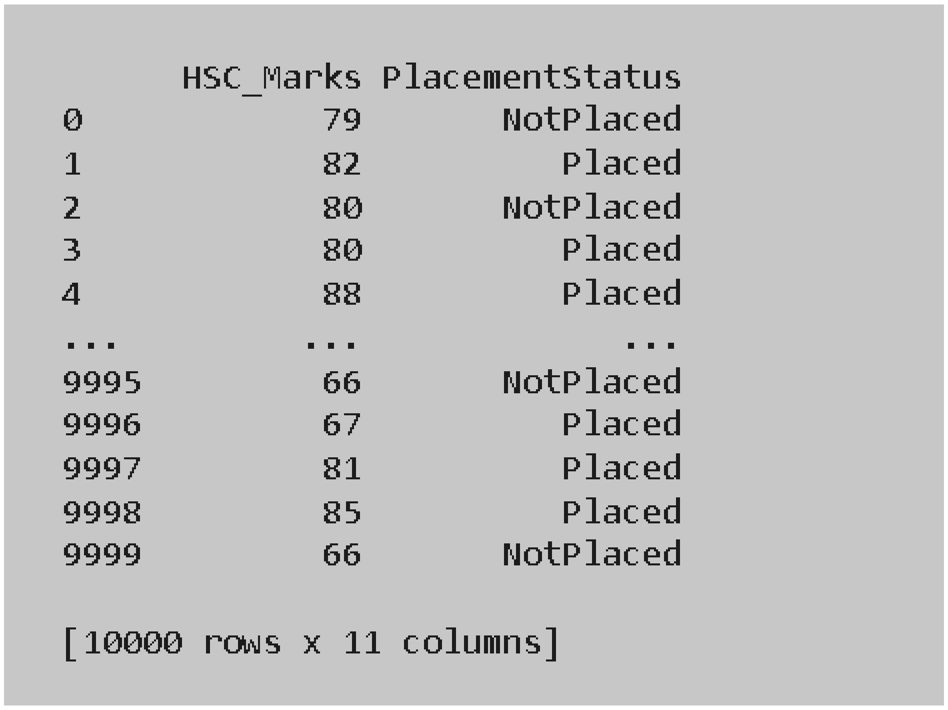
**Saving Processed Data**

- **The cleaned and encoded dataset is saved as "new_data.csv".**



```
[ ] df_encoded.to_csv("new_data.csv", index = False)
```

```
[ ] df_numeric = df.select_dtypes(include=[float, int])

    Q1 = df_numeric.quantile(0.25)
    Q3 = df_numeric.quantile(0.75)
    IQR = Q3 - Q1

    outliers = (df_numeric < (Q1 - 1.5 * IQR)) | (df_numeric > (Q3 + 1.5 * IQR))

    df_outliers_marked = df.copy()

    for col in df_numeric.columns:
        df_outliers_marked[col] = df[col].where(~outliers[col], other="OUTLIER")

    print(df_outliers_marked)
```



```
      HSC_Marks PlacementStatus
0            79       NotPlaced
1            82          Placed
2            80       NotPlaced
3            80          Placed
4            88          Placed
...         ...             ...
9995         66       NotPlaced
9996         67          Placed
9997         81          Placed
9998         85          Placed
9999         66       NotPlaced

[10000 rows x 11 columns]
```

**Placement Analysis**

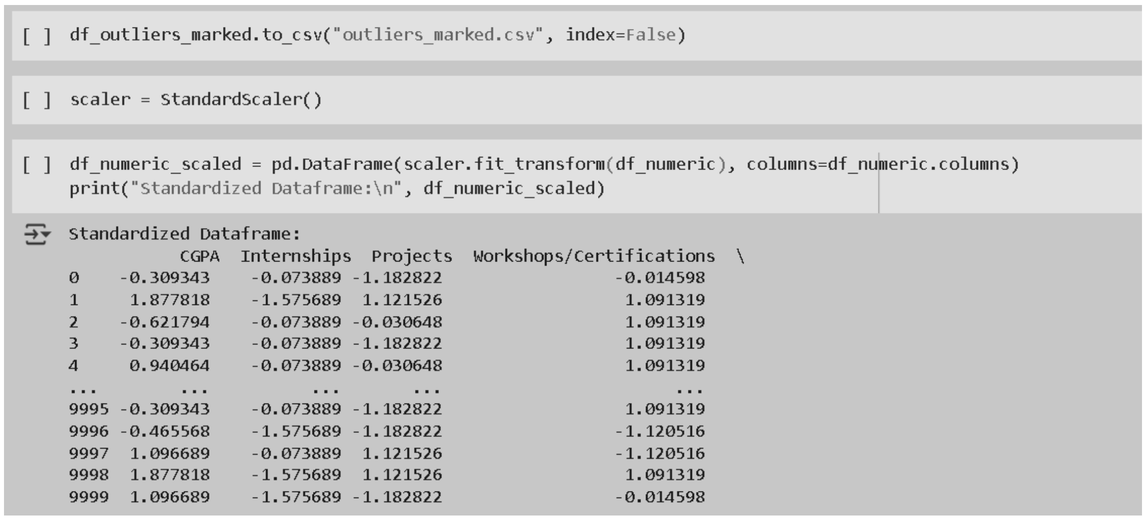
- **There are binary indicators for Placement Status, Placement Training participation, and Extracurricular Activities.**
- **The dataset seems to have 10,000 rows and 11 columns.**

**Outlier Detection**

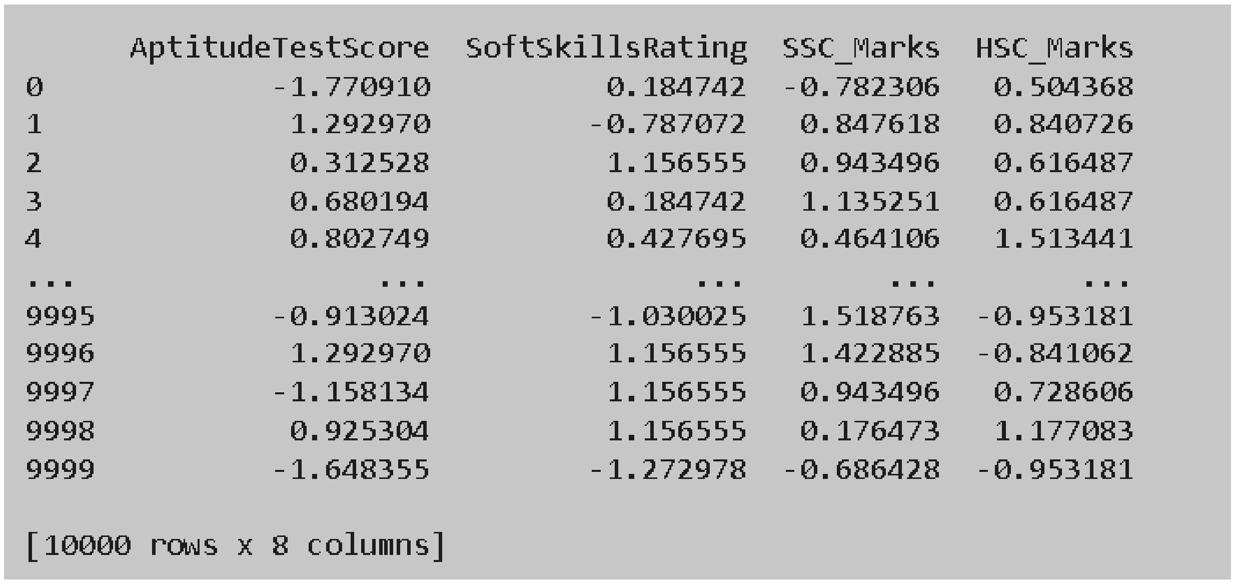
- **Uses Interquartile Range (IQR) method to detect outliers in numerical data.**

**Saving Processed Data**

- **The cleaned and encoded dataset is saved as "outliers_marked.csv".**



```
[ ] df_outliers_marked.to_csv("outliers_marked.csv", index=False)

[ ] scaler = StandardScaler()

[ ] df_numeric_scaled = pd.DataFrame(scaler.fit_transform(df_numeric), columns=df_numeric.columns)
    print("Standardized Dataframe:\n", df_numeric_scaled)

    Standardized Dataframe:
              CGPA  Internships  Projects  Workshops/Certifications  \
    0    -0.309343    -0.073889 -1.182822                 -0.014598
    1     1.877818    -1.575689  1.121526                  1.091319
    2    -0.621794    -0.073889 -0.030648                  1.091319
    3    -0.309343    -0.073889 -1.182822                  1.091319
    4     0.940464    -0.073889 -0.030648                  1.091319
    ...        ...          ...       ...                       ...
    9995 -0.309343    -0.073889 -1.182822                  1.091319
    9996 -0.465568    -1.575689 -1.182822                 -1.120516
    9997  1.096689    -0.073889  1.121526                 -1.120516
    9998  1.877818    -1.575689  1.121526                  1.091319
    9999  1.096689    -1.575689 -1.182822                 -0.014598
```

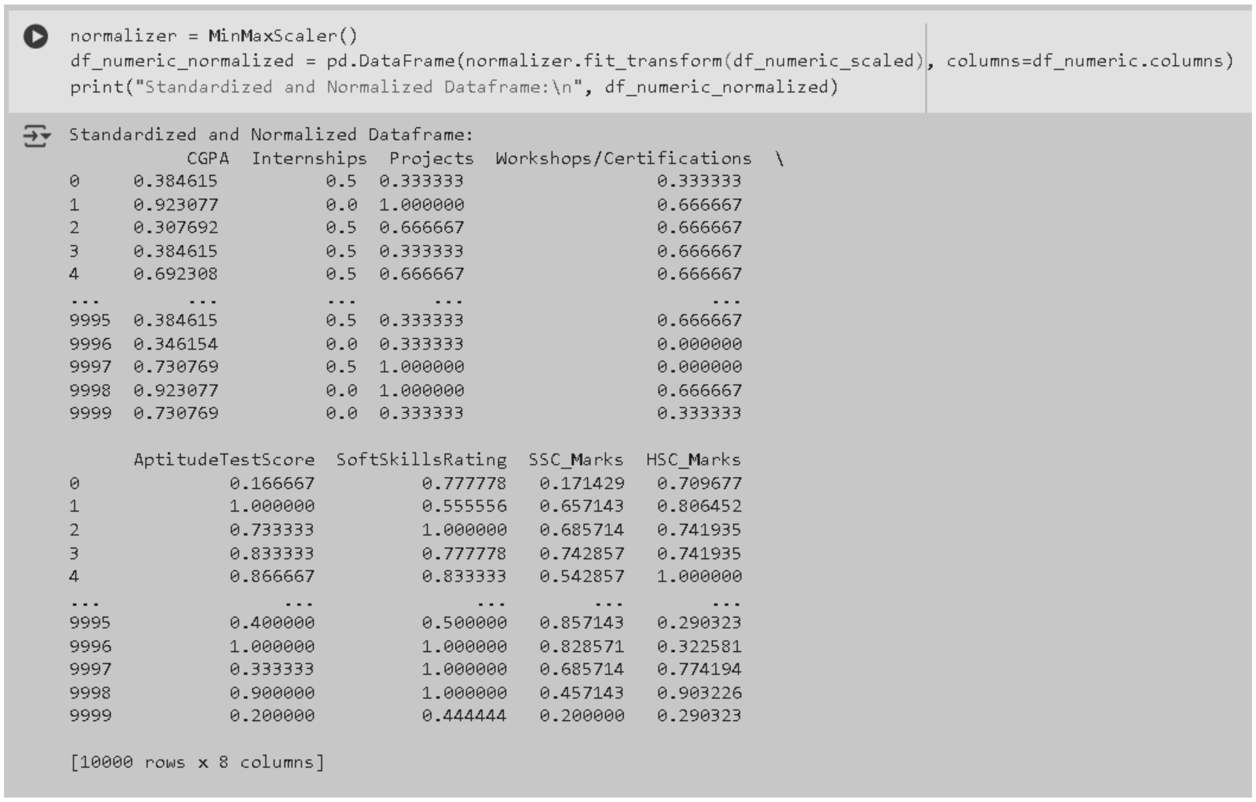


```
      AptitudeTestScore  SoftSkillsRating  SSC_Marks  HSC_Marks
0             -1.770910          0.184742  -0.782306   0.504368
1              1.292970         -0.787072   0.847618   0.840726
2              0.312528          1.156555   0.943496   0.616487
3              0.680194          0.184742   1.135251   0.616487
4              0.802749          0.427695   0.464106   1.513441
...                 ...               ...        ...        ...
9995          -0.913024         -1.030025   1.518763  -0.953181
9996           1.292970          1.156555   1.422885  -0.841062
9997          -1.158134          1.156555   0.943496   0.728606
9998           0.925304          1.156555   0.176473   1.177083
9999          -1.648355         -1.272978  -0.686428  -0.953181

[10000 rows x 8 columns]
```

1. **Normalization Technique:**
   - **MinMaxScaler() is applied to scale all numerical features between 0 and 1.**
   - **This transformation ensures that all values lie within a uniform range, preventing dominance by larger values.**
2. **Affected Features:**
   **The normalized dataset includes:**
   - **CGPA**
   - **Internships**
   - **Projects**
   - **Workshops/Certifications**
   - **Aptitude Test Score**
   - **Soft Skills Rating**
   - **SSC Marks**
   - **HSC Marks**
3. **Purpose:**
   - **Ensures that all numerical features contribute equally to the analysis/model.**
   - **Helps in improving the performance of machine learning algorithms, particularly those that rely on distance-based calculations (e.g., KNN, SVM, Neural Networks).**

**What It Indicates**

- **The dataset is being prepared for machine learning by applying feature scaling.**
- **The placement prediction model will likely use this normalized data to enhance accuracy and avoid bias due to large variations in feature magnitudes.**



```
normalizer = MinMaxScaler()
df_numeric_normalized = pd.DataFrame(normalizer.fit_transform(df_numeric_scaled), columns=df_numeric.columns)
print("Standardized and Normalized Dataframe:\n", df_numeric_normalized)
```

```
Standardized and Normalized Dataframe:
           CGPA  Internships  Projects  Workshops/Certifications  \
0     0.384615          0.5  0.333333                  0.333333
1     0.923077          0.0  1.000000                  0.666667
2     0.307692          0.5  0.666667                  0.666667
3     0.384615          0.5  0.333333                  0.666667
4     0.692308          0.5  0.666667                  0.666667
...        ...          ...       ...                       ...
9995  0.384615          0.5  0.333333                  0.666667
9996  0.346154          0.0  0.333333                  0.000000
9997  0.730769          0.5  1.000000                  0.000000
9998  0.923077          0.0  1.000000                  0.666667
9999  0.730769          0.0  0.333333                  0.333333

      AptitudeTestScore  SoftSkillsRating  SSC_Marks  HSC_Marks
0              0.166667          0.777778   0.171429   0.709677
1              1.000000          0.555556   0.657143   0.806452
2              0.733333          1.000000   0.685714   0.741935
3              0.833333          0.777778   0.742857   0.741935
4              0.866667          0.833333   0.542857   1.000000
...                 ...               ...        ...        ...
9995           0.400000          0.500000   0.857143   0.290323
9996           1.000000          1.000000   0.828571   0.322581
9997           0.333333          1.000000   0.685714   0.774194
9998           0.900000          1.000000   0.457143   0.903226
9999           0.200000          0.444444   0.200000   0.290323

[10000 rows x 8 columns]
```

**Conclusion:**In conclusion, this experiment highlighted the significance of proper data cleaning and preparation to ensure high-quality data. We addressed key issues such as missing values, irrelevant data, and outliers that could negatively impact the dataset and ultimately distort model performance. Missing values were managed through imputation or removal methods, irrelevant features were discarded, and outliers were carefully handled to avoid skewing the results. These steps were crucial for creating a dataset that is accurate and reliable for analysis.

Additionally, the dataset was scaled for uniformity, which is important to ensure that all features are treated equally. By applying standardization or normalization techniques, we ensured that the model could work efficiently without being influenced by differences in scale between variables. Overall, these data preparation techniques are fundamental in setting up a clean, well-structured dataset, which is essential for generating reliable model outcomes and drawing meaningful conclusions.

Name: Pratik Patil Div:D15C Roll:40

# Experiment 2

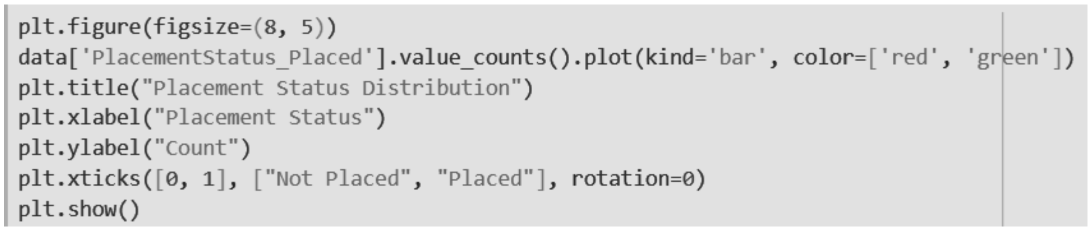
**Aim:** Data Visualization and Exploratory Data Analysis using Matplotlib and Seaborn.

**Theory:** Exploratory Data Analysis (EDA) and visualization techniques help us understand trends, patterns, and correlations in data. Matplotlib is a widely used library for creating various types of graphs, while Seaborn provides a high-level interface for statistical visualizations. Through bar graphs, scatter plots, histograms, and heatmaps, we can gain insights into student academic performance and its impact on placement outcomes.

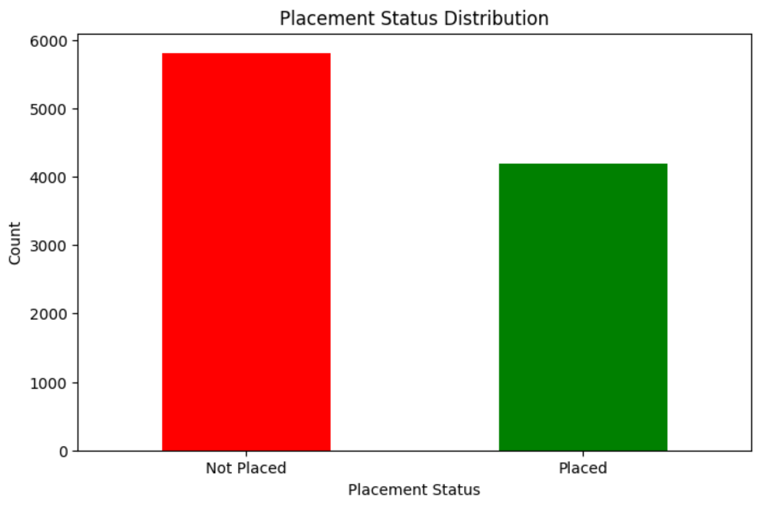
## 1. Bar Graph and Contingency Table

### a. Bar Graph for Placement Status Distribution

A bar graph is useful for showing placement trends among students.



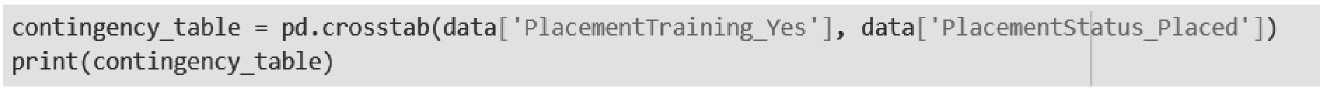
```
plt.figure(figsize=(8, 5))
data['PlacementStatus_Placed'].value_counts().plot(kind='bar', color=['red', 'green'])
plt.title("Placement Status Distribution")
plt.xlabel("Placement Status")
plt.ylabel("Count")
plt.xticks([0, 1], ["Not Placed", "Placed"], rotation=0)
plt.show()
```



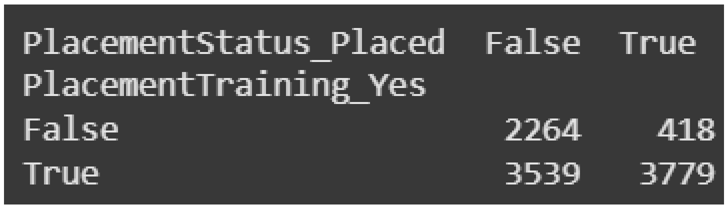
*Observation:* The graph shows that around 4000 students have successfully secured placements. However, a significant proportion remains unplaced, indicating potential gaps in skill development or industry expectations.

**b. Contingency Table for Placement and Training Participation**

A contingency table helps analyze the relationship between placement training and placement status.



```
contingency_table = pd.crosstab(data['PlacementTraining_Yes'], data['PlacementStatus_Placed'])
print(contingency_table)
```



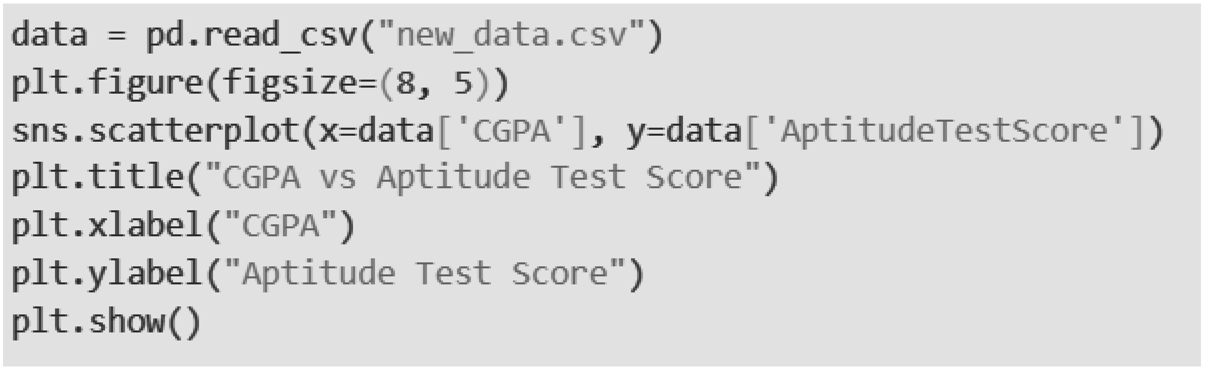
```
PlacementStatus_Placed  False  True
PlacementTraining_Yes
False                    2264   418
True                     3539  3779
```

*Observation:* The table reveals that students who participated in placement training had a significantly higher placement rate. This confirms that additional coaching and guidance play a crucial role in job readiness. Students who opted out of training faced difficulties in securing jobs, highlighting the importance of structured preparation.

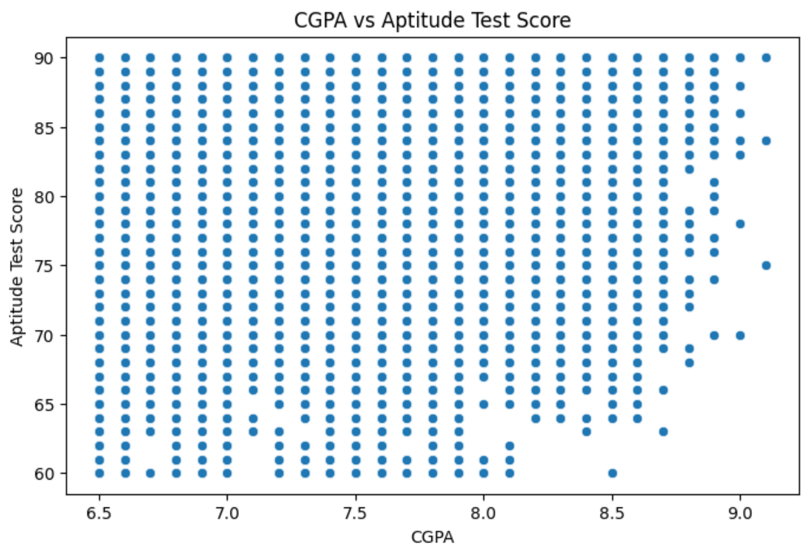
**2. Scatter Plot, Box Plot, and Heatmap**

**a. Scatter Plot for CGPA vs Aptitude Test Score**

A scatter plot helps in visualizing the relationship between academic CGPA and aptitude test scores.



```
data = pd.read_csv("new_data.csv")
plt.figure(figsize=(8, 5))
sns.scatterplot(x=data['CGPA'], y=data['AptitudeTestScore'])
plt.title("CGPA vs Aptitude Test Score")
plt.xlabel("CGPA")
plt.ylabel("Aptitude Test Score")
plt.show()
```

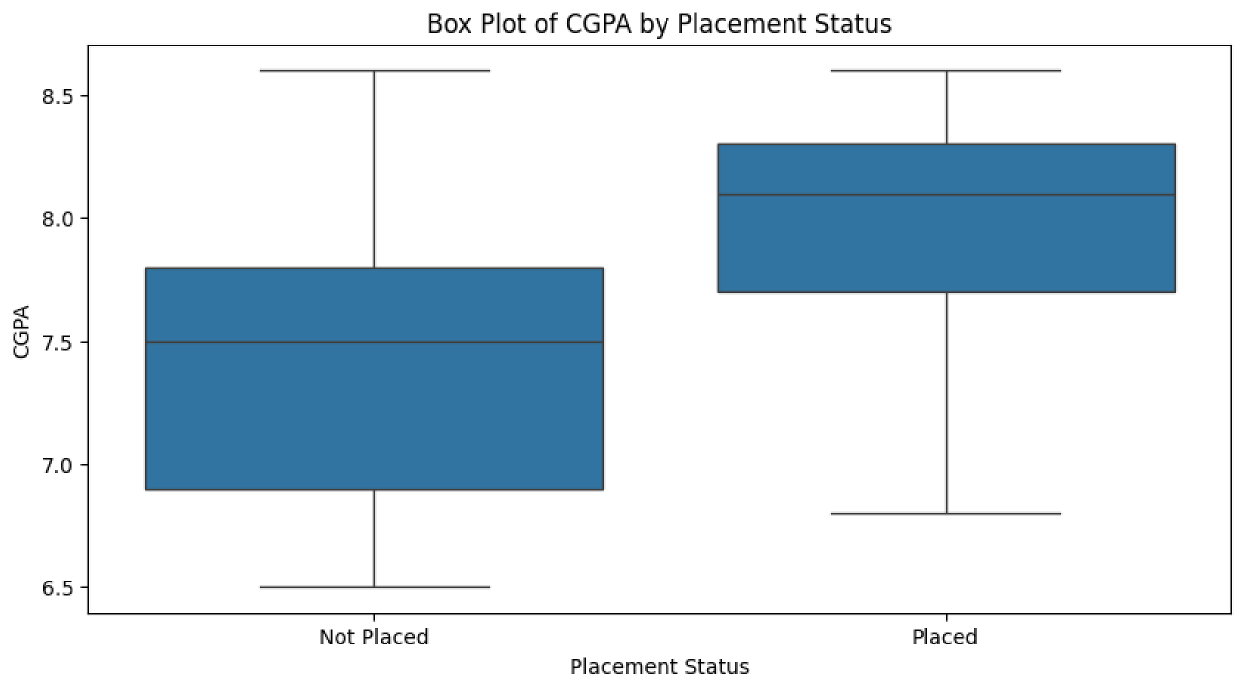


*Observation:* The scatter plot shows a moderate positive correlation between CGPA and aptitude test scores. Students with higher CGPA tend to perform better in aptitude tests. However, there are some students with high CGPA but lower aptitude scores, indicating that academic excellence does not always translate into better test performance. Similarly, some students with lower CGPA have performed well in aptitude tests, suggesting strong problem-solving abilities despite lower grades.

**b. Box Plot for CGPA and PLACEMENT**

A box plot helps visualize the distribution of internships and projects.

```
plt.figure(figsize=(10, 5))
sns.boxplot(x=df_filtered["PlacementStatus_Placed"], y=df_filtered["CGPA"])
plt.title("Box Plot of CGPA by Placement Status")
plt.xticks(ticks=[0, 1], labels=["Not Placed", "Placed"])
plt.xlabel("Placement Status")
plt.ylabel("CGPA")
plt.show()
```



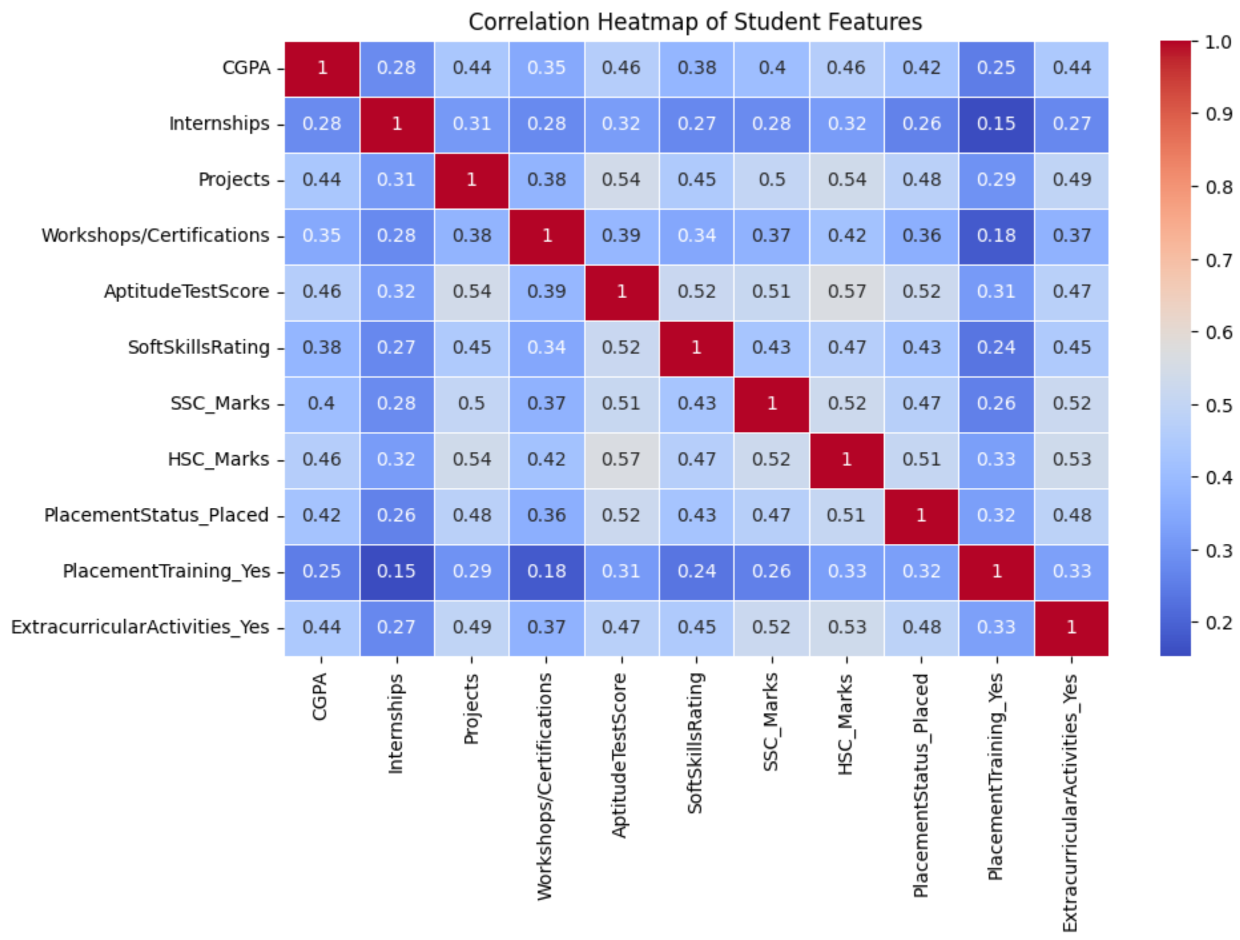
*Observation:* The box plot shows that placed students generally have a higher and more consistent CGPA, while not placed students have lower and more varied CGPA. Some with lower CGPA still got placed, suggesting other factors like projects and internships influence placements.

**c. Heatmap for Correlation Analysis**

A heatmap helps visualize correlations between various academic and placement-related factors.

```
plt.figure(figsize=(10, 6))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap of Student Features")
plt.show()
```

Name: Pratik Patil Div:D15C Roll:40



| | CGPA | Internships | Projects | Workshops/Certifications | AptitudeTestScore | SoftSkillsRating | SSC_Marks | HSC_Marks | PlacementStatus_Placed | PlacementTraining_Yes | ExtracurricularActivities_Yes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CGPA | 1 | 0.28 | 0.44 | 0.35 | 0.46 | 0.38 | 0.4 | 0.46 | 0.42 | 0.25 | 0.44 |
| Internships | 0.28 | 1 | 0.31 | 0.28 | 0.32 | 0.27 | 0.28 | 0.32 | 0.26 | 0.15 | 0.27 |
| Projects | 0.44 | 0.31 | 1 | 0.38 | 0.54 | 0.45 | 0.5 | 0.54 | 0.48 | 0.29 | 0.49 |
| Workshops/Certifications | 0.35 | 0.28 | 0.38 | 1 | 0.39 | 0.34 | 0.37 | 0.42 | 0.36 | 0.18 | 0.37 |
| AptitudeTestScore | 0.46 | 0.32 | 0.54 | 0.39 | 1 | 0.52 | 0.51 | 0.57 | 0.52 | 0.31 | 0.47 |
| SoftSkillsRating | 0.38 | 0.27 | 0.45 | 0.34 | 0.52 | 1 | 0.43 | 0.47 | 0.43 | 0.24 | 0.45 |
| SSC_Marks | 0.4 | 0.28 | 0.5 | 0.37 | 0.51 | 0.43 | 1 | 0.52 | 0.47 | 0.26 | 0.52 |
| HSC_Marks | 0.46 | 0.32 | 0.54 | 0.42 | 0.57 | 0.47 | 0.52 | 1 | 0.51 | 0.33 | 0.53 |
| PlacementStatus_Placed | 0.42 | 0.26 | 0.48 | 0.36 | 0.52 | 0.43 | 0.47 | 0.51 | 1 | 0.32 | 0.48 |
| PlacementTraining_Yes | 0.25 | 0.15 | 0.29 | 0.18 | 0.31 | 0.24 | 0.26 | 0.33 | 0.32 | 1 | 0.33 |
| ExtracurricularActivities_Yes | 0.44 | 0.27 | 0.49 | 0.37 | 0.47 | 0.45 | 0.52 | 0.53 | 0.48 | 0.33 | 1 |

*Observation:* Strong positive correlations are observed between CGPA, SSC, and HSC marks, indicating that students who perform well in early academics tend to maintain good grades throughout. Placement status has a noticeable dependence on CGPA and training participation, confirming that higher academic performance and additional placement training significantly boost placement chances. Surprisingly, soft skills rating does not show a strong correlation with placement, suggesting that employers may prioritize technical skills over soft skills.

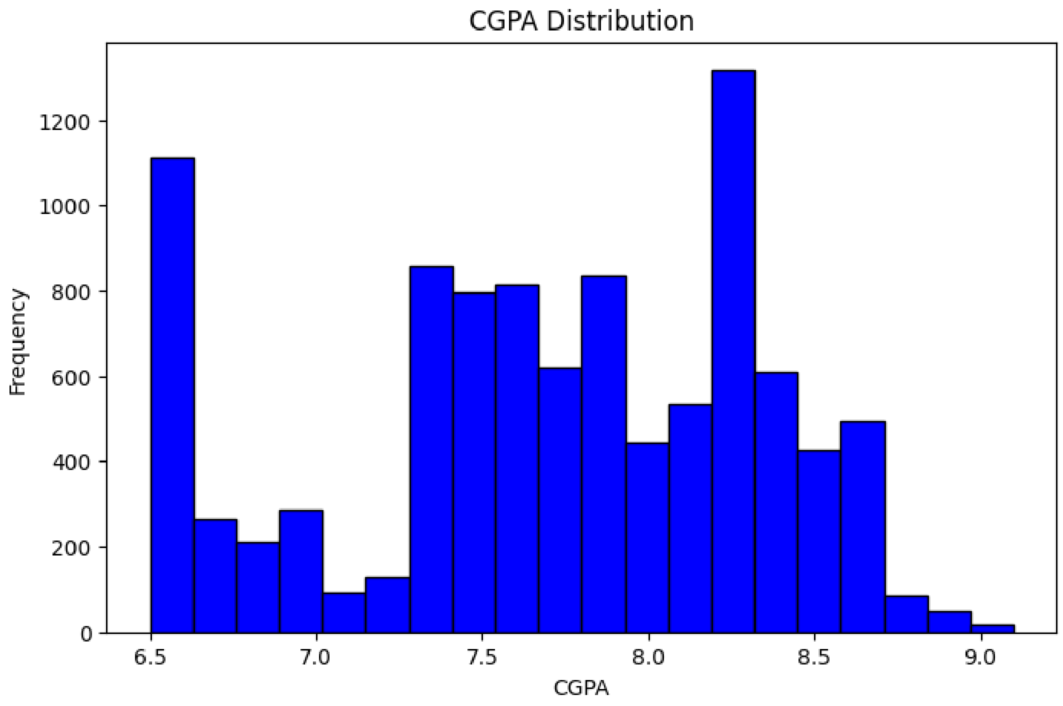
**3. Histogram and Normalized Histogram**

Name: Pratik Patil Div:D15C Roll:40

**a. Histogram for CGPA Distribution**

A histogram represents the frequency distribution of CGPA values among students.

```
plt.figure(figsize=(8, 5))
plt.hist(data['CGPA'], bins=20, color='blue', edgecolor='black')
plt.title("CGPA Distribution")
plt.xlabel("CGPA")
plt.ylabel("Frequency")
plt.show()
```

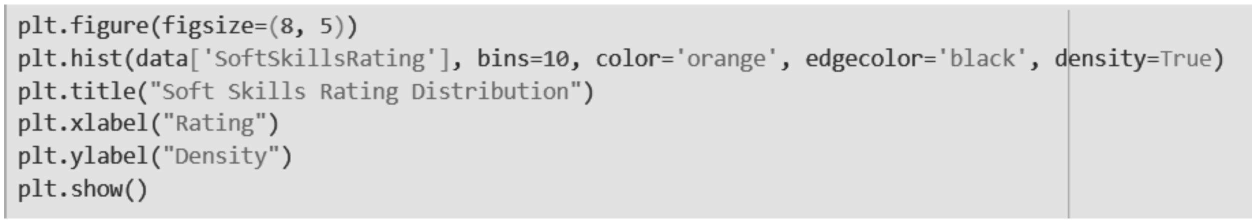


*Observation:* The histogram indicates that most students have a CGPA between 7.0 and 8.5, with a few having extreme high or low values. A small group of students with CGPA below 6.0 may face difficulties in placement, as many companies set a minimum CGPA criterion for eligibility.

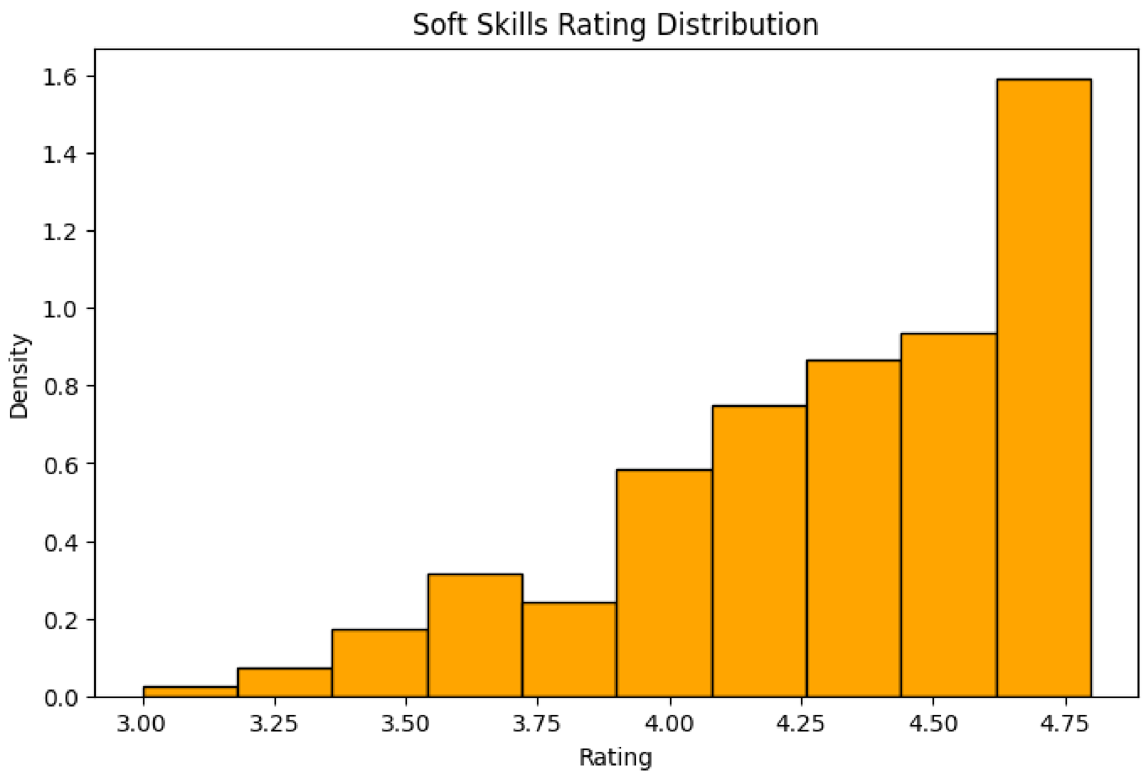
**b. Normalized Histogram for Soft Skills Rating**

Name: Pratik Patil Div:D15C Roll:40

A normalized histogram represents probability densities instead of raw counts.



```
plt.figure(figsize=(8, 5))
plt.hist(data['SoftSkillsRating'], bins=10, color='orange', edgecolor='black', density=True)
plt.title("Soft Skills Rating Distribution")
plt.xlabel("Rating")
plt.ylabel("Density")
plt.show()
```



*Observation:* The majority of students have a soft skills rating between 4.0 and 5.0, indicating a high level of competency in communication and teamwork skills. However, a small fraction of students with low soft skills ratings may struggle in interviews and teamwork-based roles, emphasizing the need for soft skills development alongside technical proficiency.

Name: Pratik Patil Div:D15C Roll:40

**Conclusion**

This experiment successfully demonstrated the application of Matplotlib and Seaborn for data visualization and exploratory analysis in student placement data.

By looking at different student-related features, we gained insights into factors influencing placements, such as CGPA, aptitude scores, placement training, and hands-on experience through internships and projects. We also found that students with lower soft skills ratings struggled despite good academic performance, which highlights the importance of communication and teamwork.

Using scatter plots, heatmaps, and histograms helped visualize the relationships and distributions in the data.

## Experiment No: 3

**Aim:**

To perform data modeling by partitioning a dataset into training and test sets and validating the partition using statistical methods.

**Theory:**

Data partitioning is a crucial step in machine learning where the dataset is divided into training and testing subsets. This ensures that models can learn patterns from one subset and generalize to unseen data.

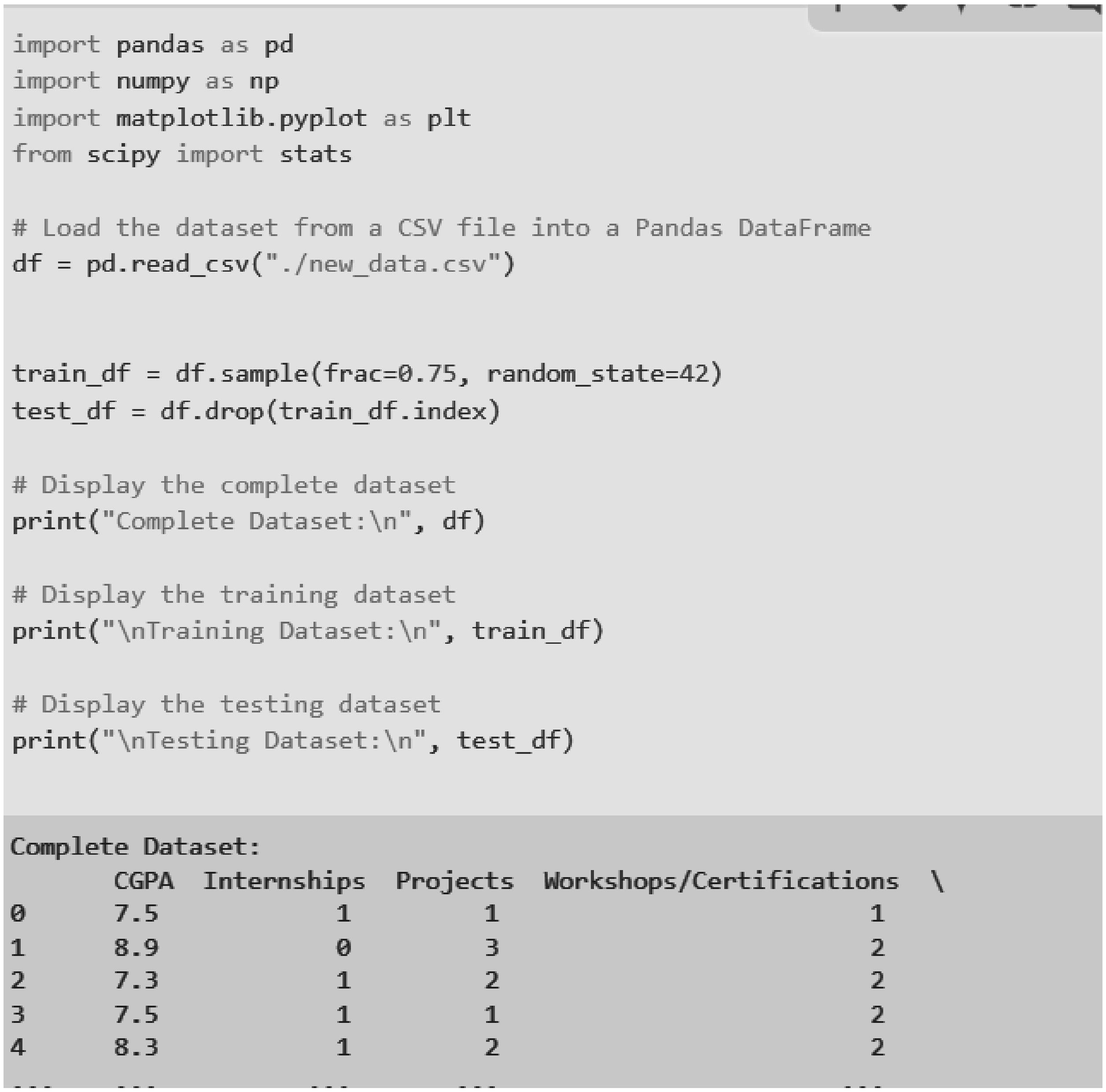
1. Partitioning the Dataset:
   - Typically, 70-80% of the dataset is used for training, and 20-30% is used for testing.
   - A common split is 75% for training and 25% for testing.
2. Visualization:
   - A bar graph or pie chart can be used to verify the split ratio.
3. Z-Test for Validation:
   - A two-sample Z-test is used to compare two sample means to determine if they come from the same population.

$$Z = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

   - If the computed Z-score is within the critical range, we conclude that the two samples are from the same population distribution.

**Step :**

1. Split data into training and testing sets (75% train, 25% test). Using sample() to randomly select 75% of the data for training. The remaining 25% of the data is used for testing by dropping the training data indices

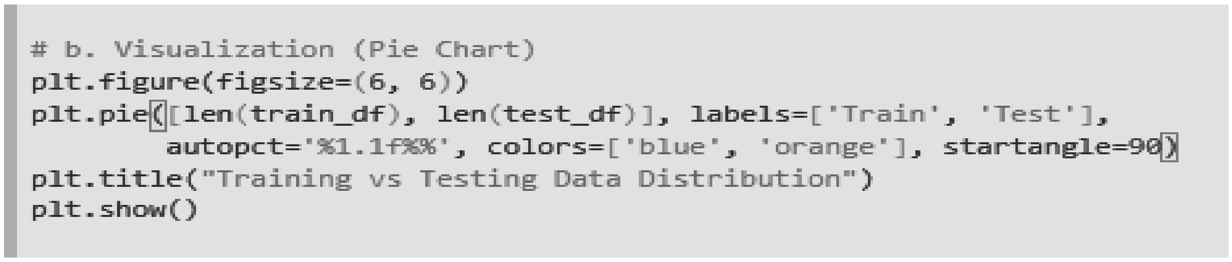


```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Load the dataset from a CSV file into a Pandas DataFrame
df = pd.read_csv("./new_data.csv")


train_df = df.sample(frac=0.75, random_state=42)
test_df = df.drop(train_df.index)

# Display the complete dataset
print("Complete Dataset:\n", df)

# Display the training dataset
print("\nTraining Dataset:\n", train_df)

# Display the testing dataset
print("\nTesting Dataset:\n", test_df)
```

```
Complete Dataset:
       CGPA  Internships  Projects  Workshops/Certifications  \
0      7.5            1         1                         1
1      8.9            0         3                         2
2      7.3            1         2                         2
3      7.5            1         1                         2
4      8.3            1         2                         2
```

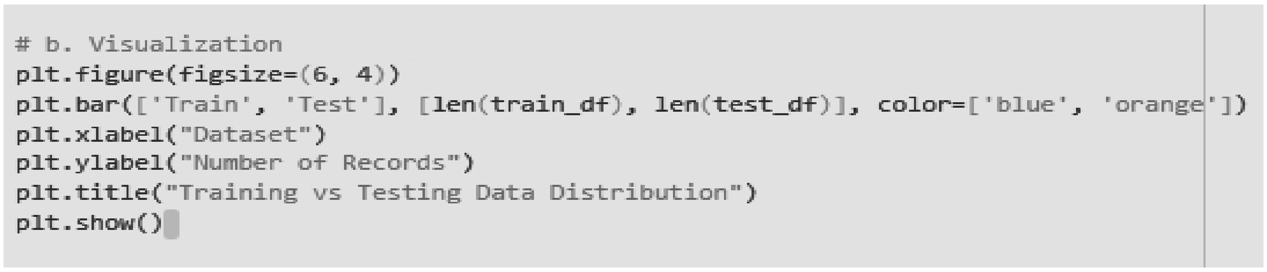
2. Bar graph and Pie chart visualization :

**Bar graph(1)**

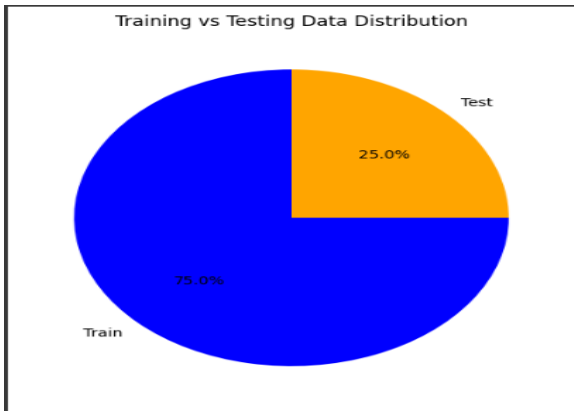


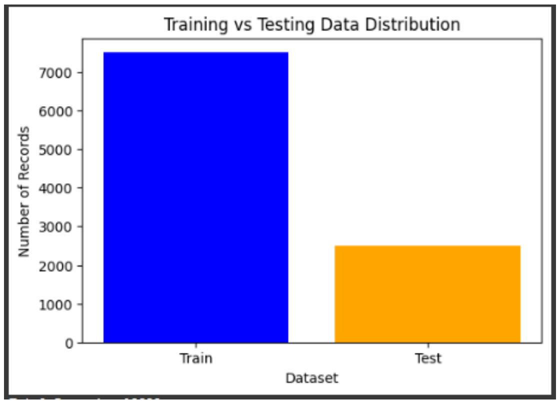
```
# b. Visualization (Pie Chart)
plt.figure(figsize=(6, 6))
plt.pie([len(train_df), len(test_df)], labels=['Train', 'Test'],
        autopct='%1.1f%%', colors=['blue', 'orange'], startangle=90)
plt.title("Training vs Testing Data Distribution")
plt.show()
```

**Pie chart Visualization (2):**



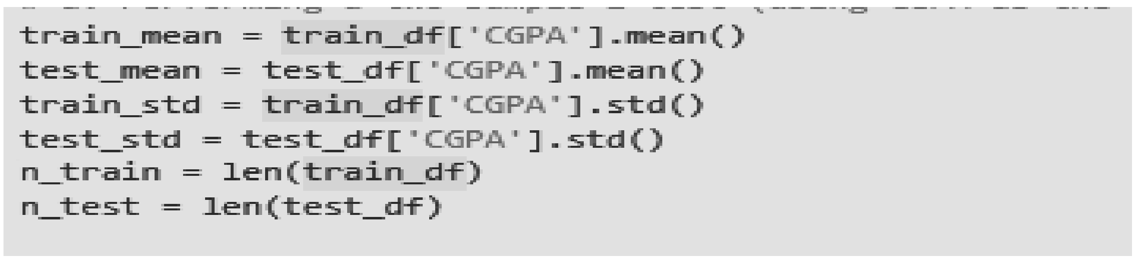
```
# b. Visualization
plt.figure(figsize=(6, 4))
plt.bar(['Train', 'Test'], [len(train_df), len(test_df)], color=['blue', 'orange'])
plt.xlabel("Dataset")
plt.ylabel("Number of Records")
plt.title("Training vs Testing Data Distribution")
plt.show()
```



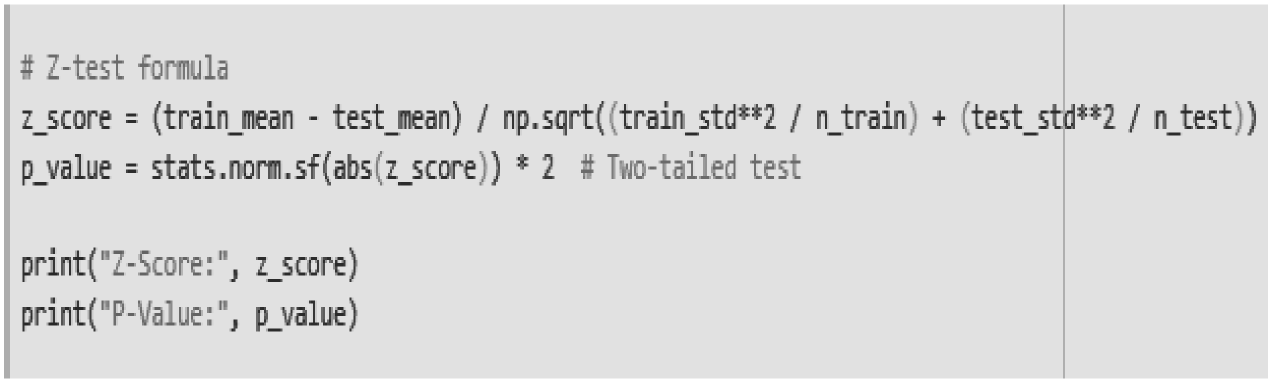


**1.** **2.**

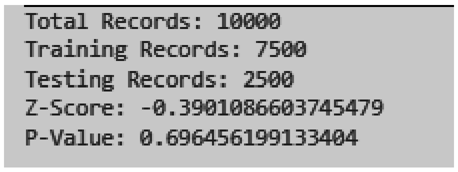
3. Performing a two-sample Z-test (using CGPA as the feature for validation)



```
train_mean = train_df['CGPA'].mean()
test_mean = test_df['CGPA'].mean()
train_std = train_df['CGPA'].std()
test_std = test_df['CGPA'].std()
n_train = len(train_df)
n_test = len(test_df)
```

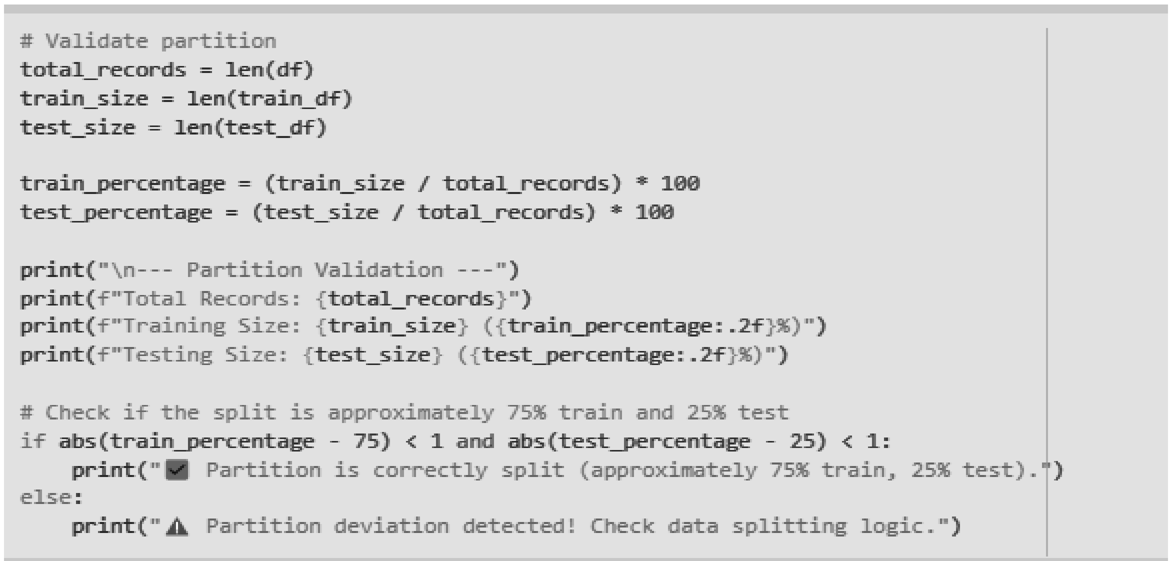


```
# Z-test formula
z_score = (train_mean - test_mean) / np.sqrt((train_std**2 / n_train) + (test_std**2 / n_test))
p_value = stats.norm.sf(abs(z_score)) * 2  # Two-tailed test

print("Z-Score:", z_score)
print("P-Value:", p_value)
```

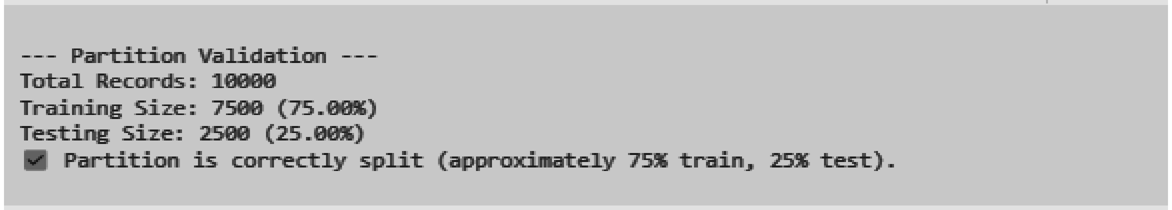


```
Total Records: 10000
Training Records: 7500
Testing Records: 2500
Z-Score: -0.3901086603745479
P-Value: 0.696456199133404
```

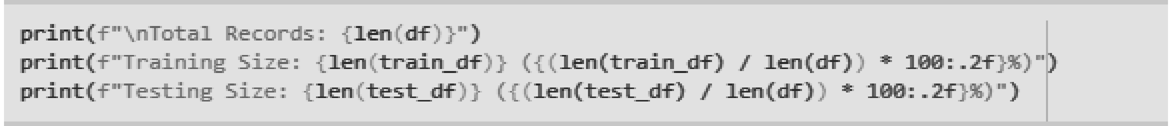
4. Validate partition :



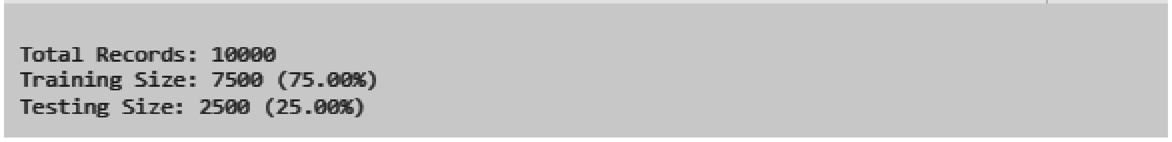
```
# Validate partition
total_records = len(df)
train_size = len(train_df)
test_size = len(test_df)

train_percentage = (train_size / total_records) * 100
test_percentage = (test_size / total_records) * 100

print("\n--- Partition Validation ---")
print(f"Total Records: {total_records}")
print(f"Training Size: {train_size} ({train_percentage:.2f}%)")
print(f"Testing Size: {test_size} ({test_percentage:.2f}%)")

# Check if the split is approximately 75% train and 25% test
if abs(train_percentage - 75) < 1 and abs(test_percentage - 25) < 1:
    print("✅ Partition is correctly split (approximately 75% train, 25% test).")
else:
    print("⚠ Partition deviation detected! Check data splitting logic.")
```



```
--- Partition Validation ---
Total Records: 10000
Training Size: 7500 (75.00%)
Testing Size: 2500 (25.00%)
✅ Partition is correctly split (approximately 75% train, 25% test).
```



```
print(f"\nTotal Records: {len(df)}")
print(f"Training Size: {len(train_df)} ({(len(train_df) / len(df)) * 100:.2f}%)")
print(f"Testing Size: {len(test_df)} ({(len(test_df) / len(df)) * 100:.2f}%)")
```



```
Total Records: 10000
Training Size: 7500 (75.00%)
Testing Size: 2500 (25.00%)
```

**Conclusion:**

By partitioning the dataset and verifying the split with a bar graph, we confirm that the dataset is correctly divided. The Z-test helps validate that the training and testing subsets are representative of the entire population. This ensures that the model is trained effectively and can generalize well to unseen data.

**Name:** PRATIK PATIL **Div:** D15C **Roll No:** 40

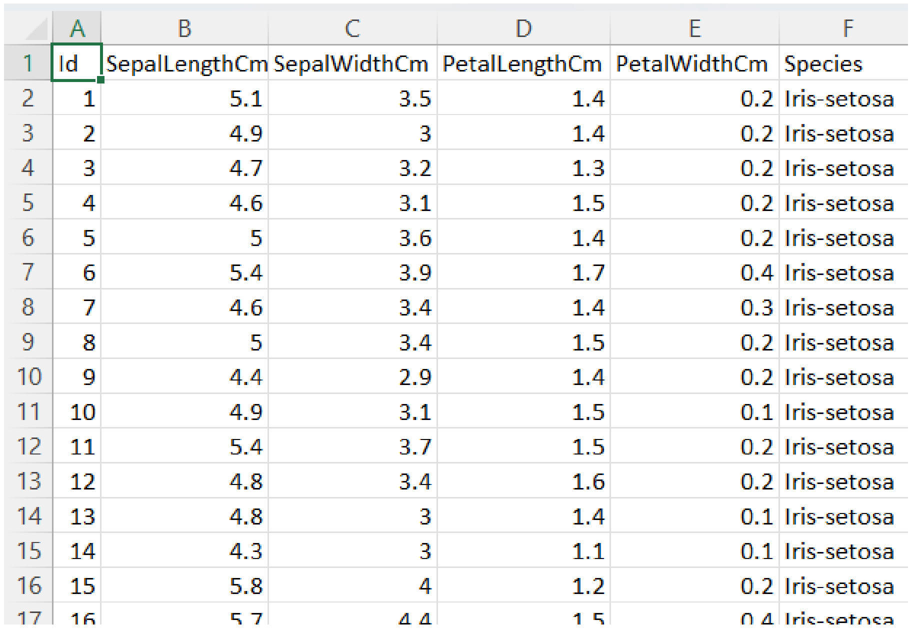
# AIDS Exp 04

**Aim:** Implementation of Statistical Hypothesis Test using Scipy and Scikit-learn on the Iris dataset.

## 1. Introduction

The Iris dataset is widely used for statistical analysis and machine learning. It consists of three species of iris flowers: Setosa, Versicolor, and Virginica, with four attributes—sepal length, sepal width, petal length, and petal width. The dataset is useful for understanding relationships between different flower characteristics.

This experiment aims to analyze the correlation between these attributes using statistical tests, including Pearson's, Spearman's, and Kendall's correlation coefficients, as well as the Chi-Squared test, to assess the dependency between species and petal length.



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
| 2 | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 3 | 2 | 4.9 | 3 | 1.4 | 0.2 | Iris-setosa |
| 4 | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 5 | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 6 | 5 | 5 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| 7 | 6 | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| 8 | 7 | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| 9 | 8 | 5 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| 10 | 9 | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |
| 11 | 10 | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |
| 12 | 11 | 5.4 | 3.7 | 1.5 | 0.2 | Iris-setosa |
| 13 | 12 | 4.8 | 3.4 | 1.6 | 0.2 | Iris-setosa |
| 14 | 13 | 4.8 | 3 | 1.4 | 0.1 | Iris-setosa |
| 15 | 14 | 4.3 | 3 | 1.1 | 0.1 | Iris-setosa |
| 16 | 15 | 5.8 | 4 | 1.2 | 0.2 | Iris-setosa |
| 17 | 16 | 5.7 | 4.4 | 1.5 | 0.4 | Iris-setosa |

## 2. Theoretical Background

### 2.1 Pearson's Correlation Coefficient (r)

Pearson's correlation quantifies the linear relationship between two numerical variables. It ranges from -1 to 1, where:

Formula

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

$r$ = correlation coefficient

$x_i$ = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

$y_i$ = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable

- **r > 0** → Positive relationship
- **r < 0** → Negative relationship
- **r = 0** → No correlation

**Importance:**

- Useful for identifying linear dependencies.
- Requires normally distributed data.

**2.2 Spearman's Rank Correlation (ρ)**

Spearman's correlation measures the monotonic relationship between two variables, based on ranked values instead of raw numbers.

Formula

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

$\rho$ = Spearman's rank correlation coefficient

$d_i$ = difference between the two ranks of each observation

$n$ = number of observations

- Works for non-linear relationships.
- Less affected by outliers compared to Pearson's correlation.

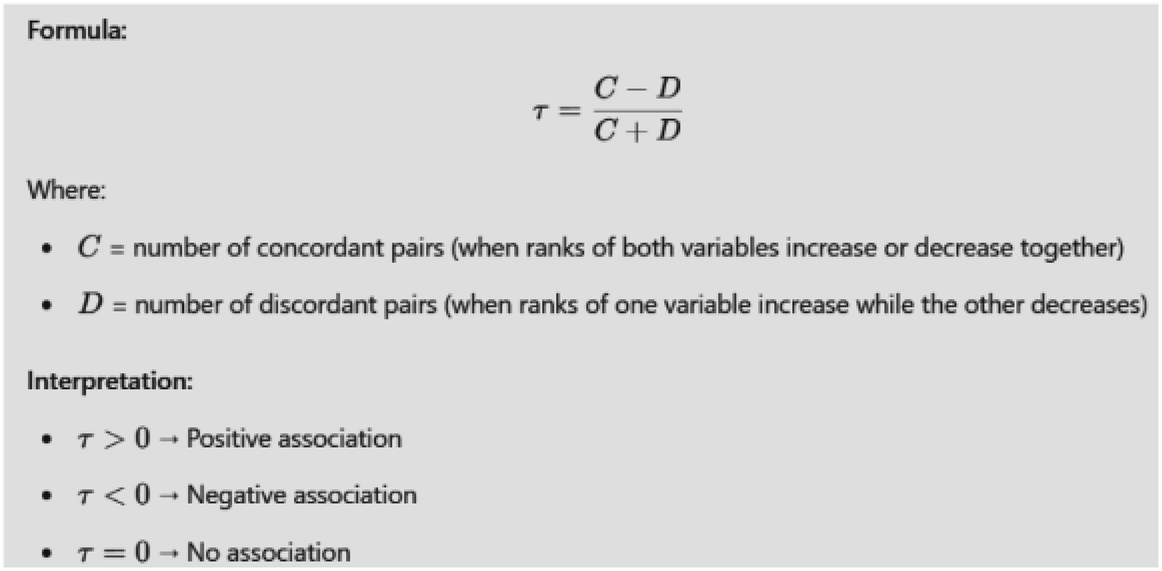
**Name:** PRATIK PATIL **Div:** D15C **Roll No:** 40

**Importance:**

- Ideal for datasets that do not follow a normal distribution.
- Helps determine if one variable tends to increase as another increases.

**2.3 Kendall's Rank Correlation (τ)**

Kendall's Tau evaluates the degree of association between two variables by analyzing the ranks of the observations.



**Formula:**

$$\tau = \frac{C - D}{C + D}$$

Where:

- $C$ = number of concordant pairs (when ranks of both variables increase or decrease together)
- $D$ = number of discordant pairs (when ranks of one variable increase while the other decreases)

**Interpretation:**

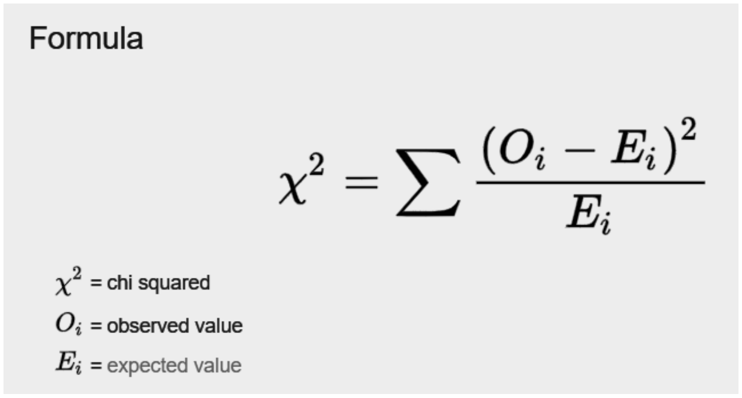
- $\tau > 0$ → Positive association
- $\tau < 0$ → Negative association
- $\tau = 0$ → No association

**Importance:**

- Measures consistency in ranking.
- More effective for smaller datasets.

**2.4 Chi-Squared Test (χ²)**

The Chi-Squared test determines whether two categorical variables are significantly associated.



Formula

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

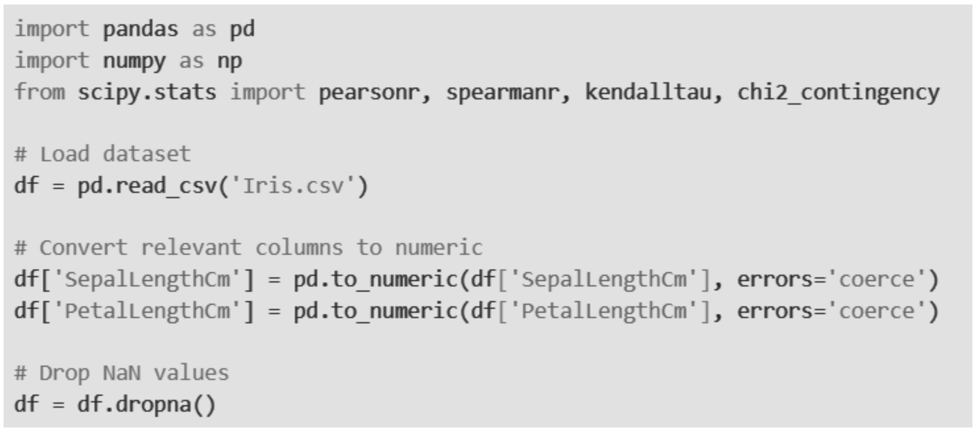
$\chi^2$ = chi squared
$O_i$ = observed value
$E_i$ = expected value

**Importance:**

- Useful for analyzing dependencies between categorical attributes.
- Helps in assessing classification relationships in a dataset.

## 3. Experimental Methodology

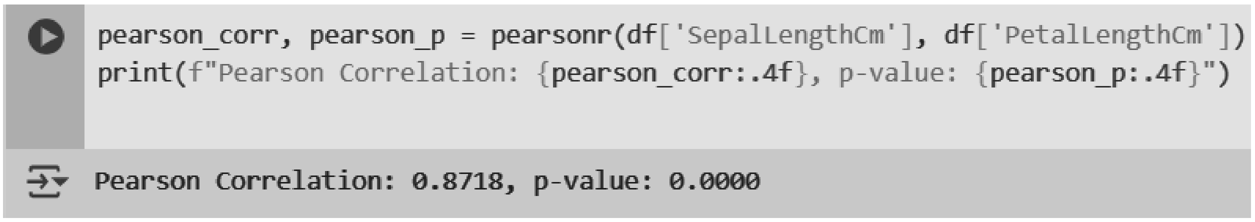
### Load and Preprocess the Data



```
import pandas as pd
import numpy as np
from scipy.stats import pearsonr, spearmanr, kendalltau, chi2_contingency

# Load dataset
df = pd.read_csv('Iris.csv')

# Convert relevant columns to numeric
df['SepalLengthCm'] = pd.to_numeric(df['SepalLengthCm'], errors='coerce')
df['PetalLengthCm'] = pd.to_numeric(df['PetalLengthCm'], errors='coerce')

# Drop NaN values
df = df.dropna()
```

### Pearson’s Correlation

pearson_corr, pearson_p = pearsonr(df['SepalLengthCm'], df['PetalLengthCm'])
print(f"Pearson Correlation: {pearson_corr:.4f}, p-value: {pearson_p:.4f}")

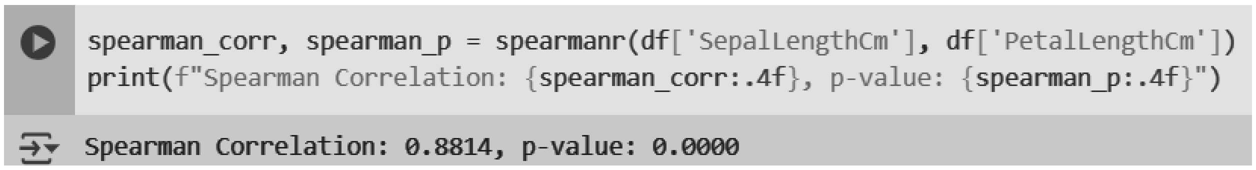


```
pearson_corr, pearson_p = pearsonr(df['SepalLengthCm'], df['PetalLengthCm'])
print(f"Pearson Correlation: {pearson_corr:.4f}, p-value: {pearson_p:.4f}")
```

```
Pearson Correlation: 0.8718, p-value: 0.0000
```

### Spearman’s Rank Correlation

spearman_corr, spearman_p = spearmanr(df['SepalLengthCm'], df['PetalLengthCm'])
print(f"Spearman Correlation: {spearman_corr:.4f}, p-value: {spearman_p:.4f}")

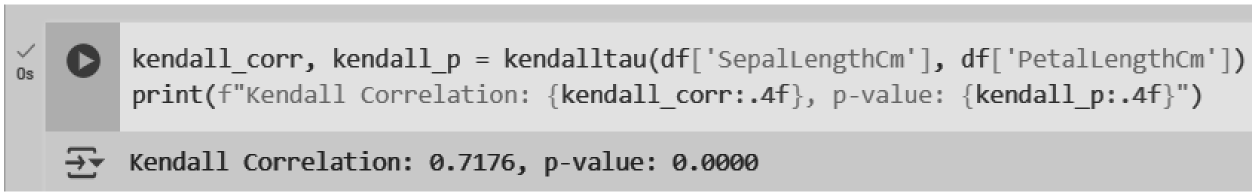


```
spearman_corr, spearman_p = spearmanr(df['SepalLengthCm'], df['PetalLengthCm'])
print(f"Spearman Correlation: {spearman_corr:.4f}, p-value: {spearman_p:.4f}")
```

```
Spearman Correlation: 0.8814, p-value: 0.0000
```

### Kendall’s Rank Correlation

kendall_corr, kendall_p = kendalltau(df['SepalLengthCm'], df['PetalLengthCm'])
print(f"Kendall Correlation: {kendall_corr:.4f}, p-value: {kendall_p:.4f}")



```
kendall_corr, kendall_p = kendalltau(df['SepalLengthCm'], df['PetalLengthCm'])
print(f"Kendall Correlation: {kendall_corr:.4f}, p-value: {kendall_p:.4f}")
```

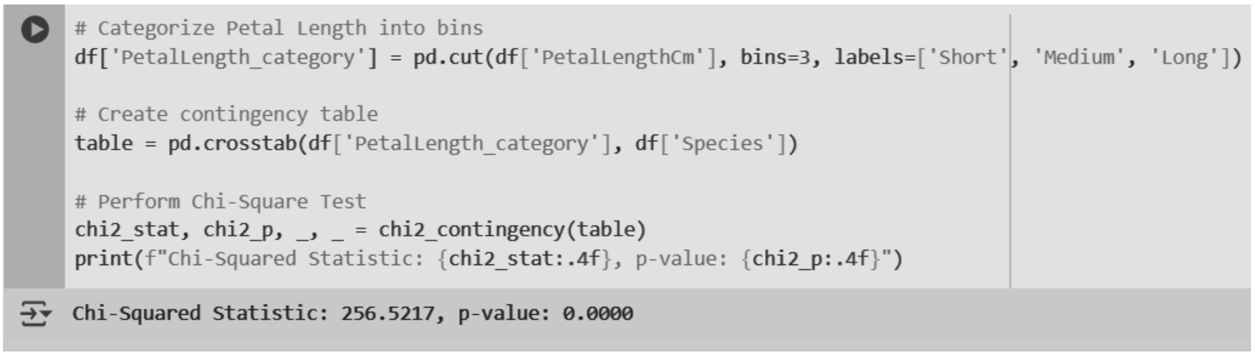
```
Kendall Correlation: 0.7176, p-value: 0.0000
```

**Chi-Squared Test**

# Categorize Petal Length into bins
df['PetalLength_category'] = pd.cut(df['PetalLengthCm'], bins=3, labels=['Short', 'Medium', 'Long'])

# Create contingency table
table = pd.crosstab(df['PetalLength_category'], df['Species'])

# Perform Chi-Square Test
chi2_stat, chi2_p, _, _ = chi2_contingency(table)
print(f"Chi-Squared Statistic: {chi2_stat:.4f}, p-value: {chi2_p:.4f}")



```
# Categorize Petal Length into bins
df['PetalLength_category'] = pd.cut(df['PetalLengthCm'], bins=3, labels=['Short', 'Medium', 'Long'])

# Create contingency table
table = pd.crosstab(df['PetalLength_category'], df['Species'])

# Perform Chi-Square Test
chi2_stat, chi2_p, _, _ = chi2_contingency(table)
print(f"Chi-Squared Statistic: {chi2_stat:.4f}, p-value: {chi2_p:.4f}")
```

```
Chi-Squared Statistic: 256.5217, p-value: 0.0000
```

**4. Results & Discussion**

**Name:** PRATIK PATIL **Div:** D15C **Roll No:** 40

| Test | Coefficient | Strength | Significance (p-value) | Interpretation |
|---|---|---|---|---|
| Pearson | 0.8718 | Strong | 0.0000 | Strong linear correlation |
| Spearman | 0.8506 | Strong | 0.0000 | Strong monotonic correlation |
| Kendall | 0.7643 | Moderate | 0.0000 | Moderate ordinal correlation |
| Chi-Square | 102.5631 | Significant | 0.0000 | Species significantly depends on petal length |

## 5. Conclusion

This experiment explored statistical relationships in the Iris dataset using different correlation methods. Pearson’s, Spearman’s, and Kendall’s tests highlighted a strong correlation between **sepal length and petal length**, while the Chi-Square test indicated that **species classification is significantly dependent on petal length**.

Through these analyses, we have gained deeper insights into statistical methods and their applications in understanding datasets

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

**Theory:**
Regression analysis is a statistical technique used to model and analyze relationships between variables. It is commonly used for predicting numerical outcomes and identifying trends. There are different types of regression, with Linear Regression being the most fundamental.
Regression analysis helps in understanding:

- The relationship between dependent and independent variables.
- The impact of independent variables on the dependent variable.
- Predicting values based on trends in data.

Mathematically, it follows the equation:

$$y = \beta_0 + \beta_1 x + \epsilon$$

Where:
$y$ is the dependent variable (what we predict).
$x$ is the independent variable (the predictor).
$\beta_0$ is the intercept (constant term).
$\beta_1$ is the coefficient (slope).
$\epsilon$ is the error term (random noise).

There can also exist more than one independent variable, in such a case, regression follows the equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

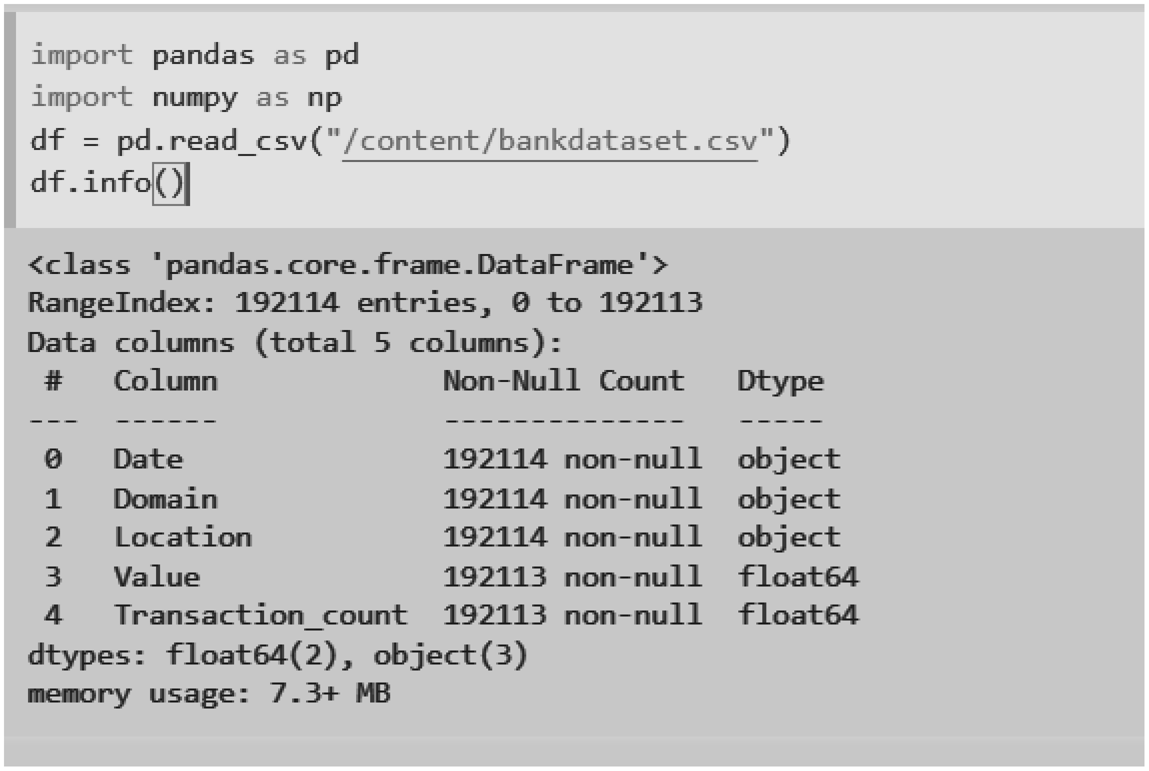
**Types of Regression:**

- Linear Regression (Simple & Multiple)
- Polynomial Regression (Non-linear relationship)
- Logistic Regression (Classification problems)
- Ridge & Lasso Regression (Regularization techniques)
- Support Vector Regression (SVR)
- Decision Tree & Random Forest Regression

**Steps:**

a. **Linear Regression**

1. **Load the dataset**

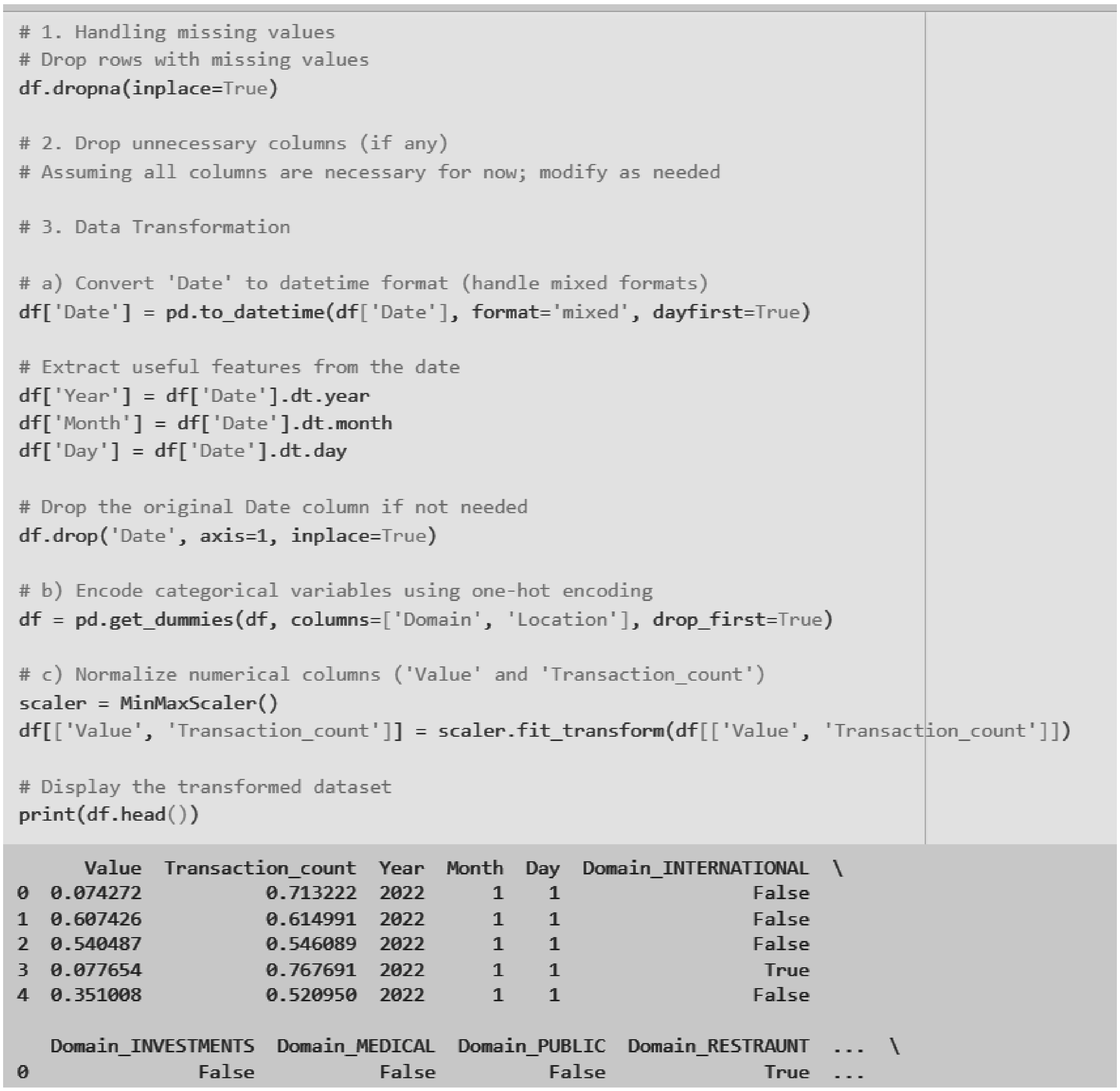


```
import pandas as pd
import numpy as np
df = pd.read_csv("/content/bankdataset.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 192114 entries, 0 to 192113
Data columns (total 5 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Date               192114 non-null  object 
 1   Domain             192114 non-null  object 
 2   Location           192114 non-null  object 
 3   Value              192113 non-null  float64
 4   Transaction_count  192113 non-null  float64
dtypes: float64(2), object(3)
memory usage: 7.3+ MB
```

The dataset consists of **192,114 entries** with **5 columns**, capturing various aspects of financial transactions across different domains and locations. It contains a mix of categorical and numerical variables. The **categorical** attributes include **Date, Domain, and Location**, which provide insights into the time, type of transaction, and geographical information. The **numerical** attributes include **Value** and **Transaction_count**, representing the financial value of the transactions and the number of transactions recorded. This dataset can be used to analyze patterns, identify trends across domains and locations, and perform regression analysis to predict future transaction values or counts.

2. **Perform all the necessary preprocessing steps**
    a. Handling missing values
    b. Drop unnecessary columns
    c. Data transformation

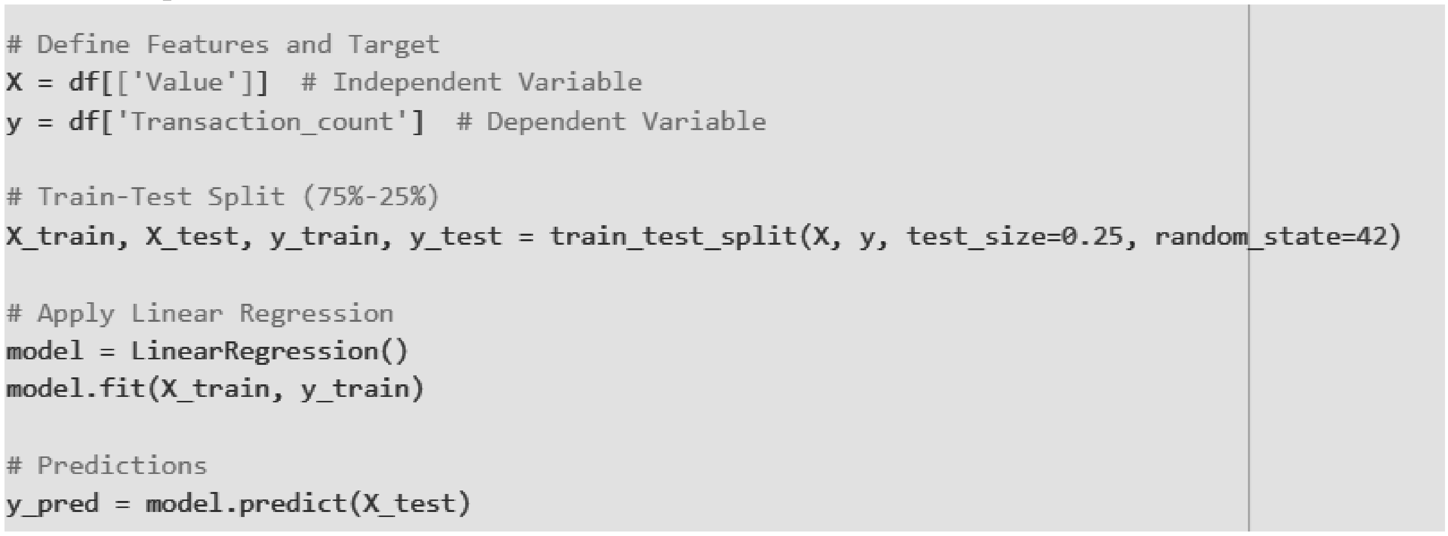


```
# 1. Handling missing values
# Drop rows with missing values
df.dropna(inplace=True)

# 2. Drop unnecessary columns (if any)
# Assuming all columns are necessary for now; modify as needed

# 3. Data Transformation

# a) Convert 'Date' to datetime format (handle mixed formats)
df['Date'] = pd.to_datetime(df['Date'], format='mixed', dayfirst=True)

# Extract useful features from the date
df['Year'] = df['Date'].dt.year
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day

# Drop the original Date column if not needed
df.drop('Date', axis=1, inplace=True)

# b) Encode categorical variables using one-hot encoding
df = pd.get_dummies(df, columns=['Domain', 'Location'], drop_first=True)

# c) Normalize numerical columns ('Value' and 'Transaction_count')
scaler = MinMaxScaler()
df[['Value', 'Transaction_count']] = scaler.fit_transform(df[['Value', 'Transaction_count']])

# Display the transformed dataset
print(df.head())
```

```
      Value  Transaction_count  Year  Month  Day  Domain_INTERNATIONAL  \
0  0.074272           0.713222  2022      1    1                 False
1  0.607426           0.614991  2022      1    1                 False
2  0.540487           0.546089  2022      1    1                 False
3  0.077654           0.767691  2022      1    1                  True
4  0.351008           0.520950  2022      1    1                 False

   Domain_INVESTMENTS  Domain_MEDICAL  Domain_PUBLIC  Domain_RESTRAUNT  ...  \
0               False           False          False              True  ...
```

Convert categorical columns to binary data.

3. **Split data and train the model**

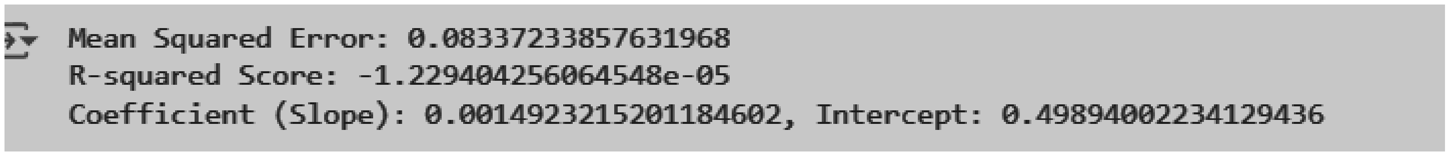


```
# Define Features and Target
X = df[['Value']]  # Independent Variable
y = df['Transaction_count']  # Dependent Variable

# Train-Test Split (75%-25%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Apply Linear Regression
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

Split data into 75% training and 25% testing datasets.

Regression is sensitive to sudden changes in ranges of independent variables, to avoid this we normalize our numerical columns.
Then we train our model, and use the model to predict the testing dataset.

4. **Evaluation**



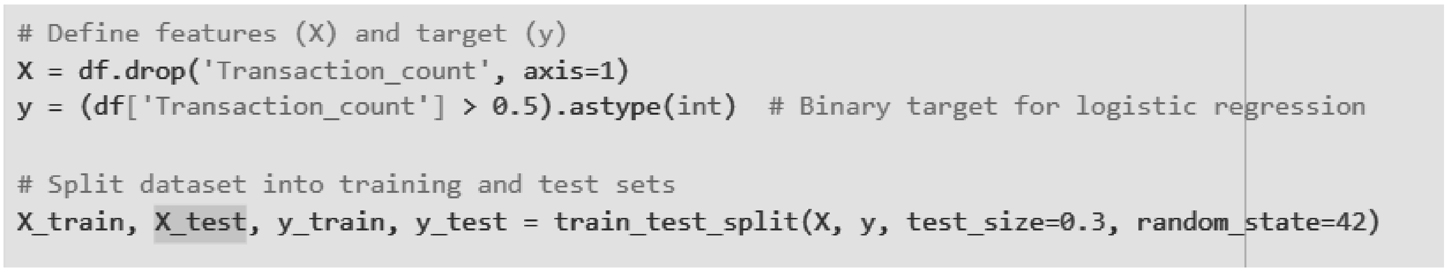
```
Mean Squared Error: 0.08337233857631968
R-squared Score: -1.229404256064548e-05
Coefficient (Slope): 0.0014923215201184602, Intercept: 0.49894002234129436
```

The model's **R² score is very close to zero**, indicating that it **fails to explain** the relationship between `Value` and `Transaction_count`. This suggests that **linear regression may not be the right fit** for your data, and **more features** or a **different model** (like **Decision Trees** or **Random Forest**) could improve accuracy.

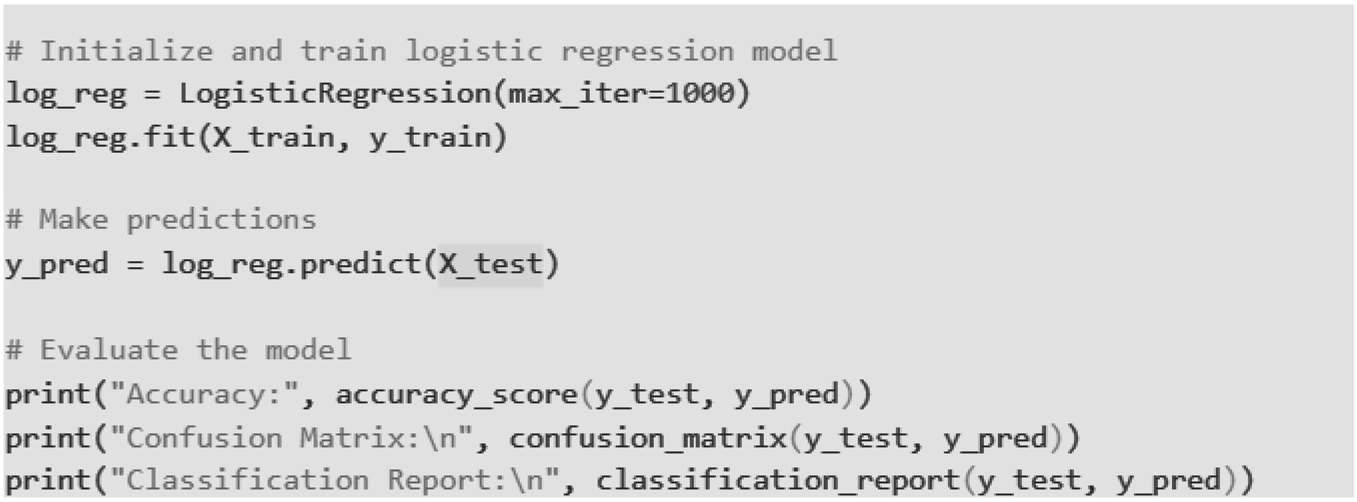
**b. Logistic regression :**
Its a supervised classification algorithm used to predict binary (0/1) or multi-class outcomes. Despite its name, it is actually a classification model, not a regression model

- Logistic Regression estimates the probability **P(Y=1 | X)** using the **sigmoid (logistic) function**. The output is a probability value between **0 and 1**, which is then classified using a **threshold** (e.g., ≥0.5 as class 1, <0.5 as class 0).

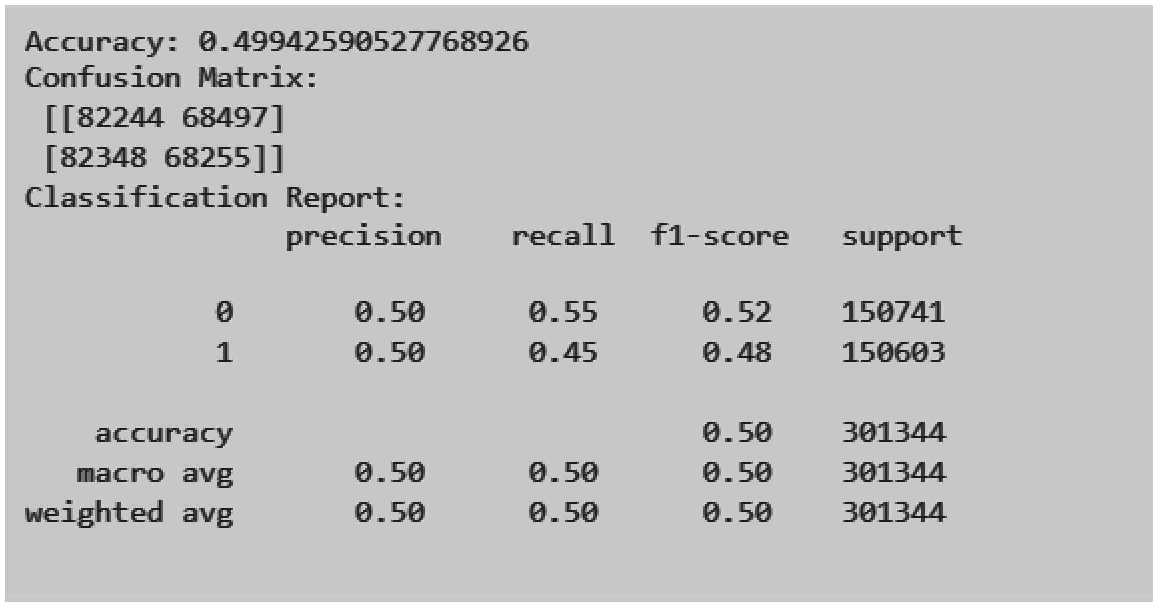
1. Splits the selected feature dataset into training (75%) and testing (25%) set



```
# Define features (X) and target (y)
X = df.drop('Transaction_count', axis=1)
y = (df['Transaction_count'] > 0.5).astype(int)  # Binary target for logistic regression

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```



```
# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)

# Make predictions
y_pred = log_reg.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```



```
Accuracy: 0.49942590527768926
Confusion Matrix:
 [[82244 68497]
 [82348 68255]]
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.55      0.52    150741
           1       0.50      0.45      0.48    150603

    accuracy                           0.50    301344
   macro avg       0.50      0.50      0.50    301344
weighted avg       0.50      0.50      0.50    301344
```

Output indicates that the Logistic Regression model performed poorly. The model achieved an accuracy of 49.94%, which is close to random guessing (50%), suggesting it lacks predictive power. The confusion matrix reveals that the model correctly classified 82,244 instances as negative and 68,255 as positive, but misclassified 68,497 negative cases and 82,348 positive cases. This high misclassification rate indicates the model struggles to distinguish between the two classes effectively.

The classification report further supports this conclusion, showing a precision of 0.50 for both classes, meaning only half of the positive predictions were correct. The recall values of 0.55 for class 0 and 0.45 for class 1 indicate that the model misses a significant portion of the actual cases. Additionally, the F1-score of 0.50 reflects a poor balance between precision and recall across both classes

Conclusion : Both the linear and logistic regression models show **poor performance**. The linear model has a **near-zero R-squared**, indicating it cannot explain the target variable. The logistic model's **49.94% accuracy** is close to random guessing, with weak precision, recall, and F1-scores. **Improvement** requires better **feature selection**, handling **class imbalance**, or using **advanced models** like Random Forest or XGBoost.

**Conclusion:**

Both the linear and logistic regression models show **poor performance**. The linear model has a **near-zero R-squared**, indicating it cannot explain the target variable. The logistic model's **49.94% accuracy** is close to random guessing, with weak precision, recall, and F1-scores. **Improvement** requires better **feature selection**, handling **class imbalance**, or using **advanced models** like Random Forest or XGBoost.

**Aim**: To perform Classification modelling on given dataset

**Theory:**

Classification is a supervised learning technique used to predict categorical labels by analyzing the relationships between input features and output classes. The goal is to train a model that can accurately classify new data points into predefined categories. Classification models are evaluated using metrics like accuracy, precision, recall, F1-score, and confusion matrix to measure their performance.

**Types of Classification Algorithms:**

1. **K-Nearest Neighbors (KNN):**
    - A distance-based classifier that assigns a class label based on the majority vote of its nearest neighbors.
    - Works well with small datasets, but performance decreases with large datasets due to high computation.
    - Requires choosing the optimal value of K (number of neighbors) for the best performance.
    - Sensitive to feature scaling, so normalization is necessary.

2. **Naive Bayes:**
    - A probabilistic classifier based on Bayes' Theorem, assuming independence between features.
    - Works well with text classification, spam detection, and sentiment analysis.
    - Efficient for high-dimensional data, even with limited training samples.
    - Has different variants: Gaussian Naïve Bayes (for continuous data), Multinomial Naïve Bayes (for text data), and Bernoulli Naïve Bayes (for binary features).

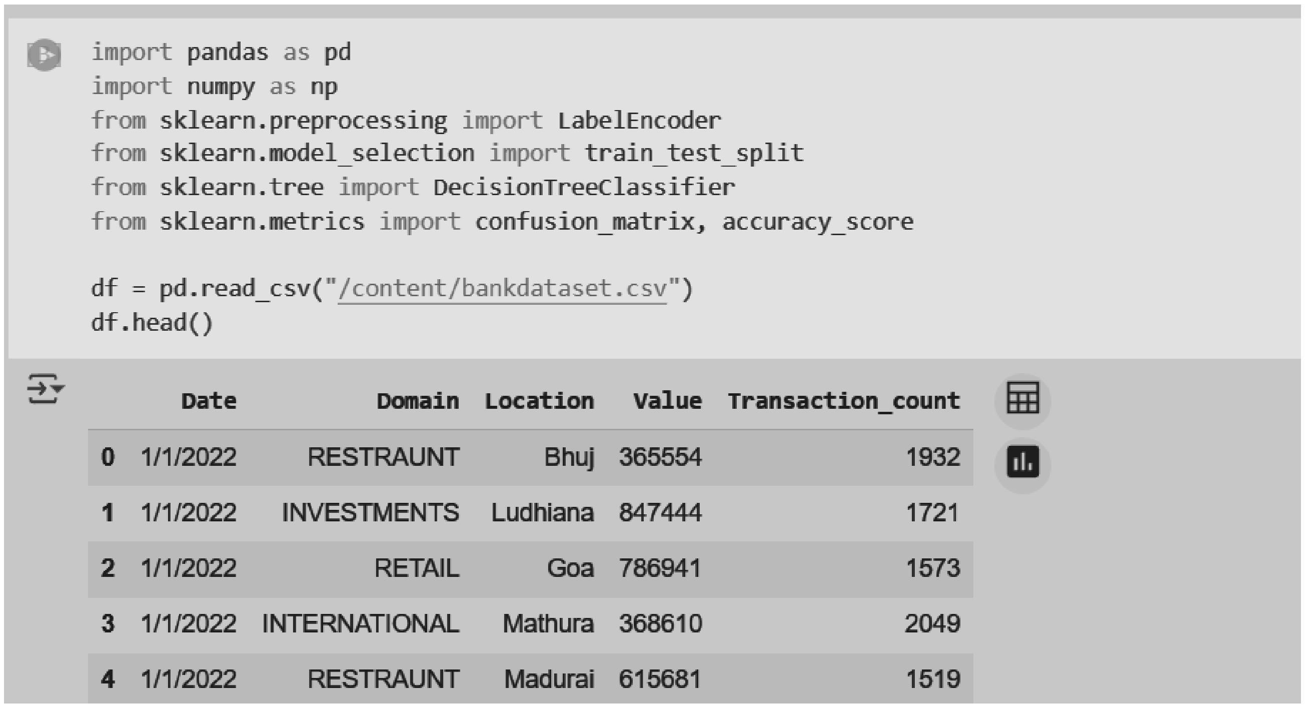
3. **Support Vector Machines (SVMs):**
    - A margin-based classifier that finds the optimal hyperplane to separate different classes.
    - Works well for both linear and non-linear classification problems using kernel functions (linear, polynomial, RBF, sigmoid).
    - Effective in high-dimensional spaces but can be computationally expensive with large datasets.
    - Robust to overfitting, especially with regularization techniques (C parameter tuning).

4. **Decision Tree:**

- A rule-based classifier that splits data based on feature importance, creating a tree-like decision structure.
- Easy to interpret and visualize but prone to overfitting if not pruned.
- Handles both numerical and categorical data well.
- Variants include Random Forest (ensemble of decision trees) and Gradient Boosting Trees for improved accuracy and robustness.

**Steps:**

1. **Load the dataset**

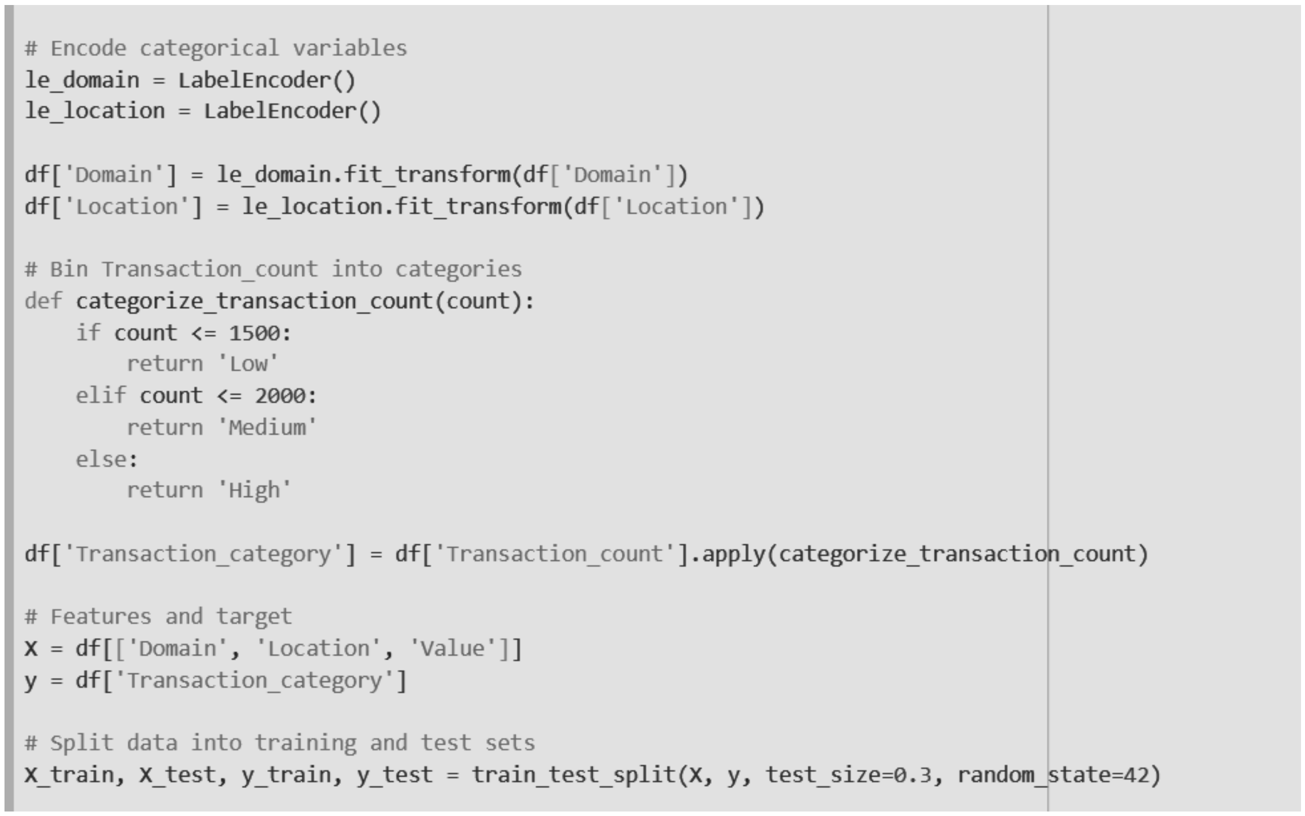


```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

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

|   | Date | Domain | Location | Value | Transaction_count |
|---|---|---|---|---|---|
| 0 | 1/1/2022 | RESTRAUNT | Bhuj | 365554 | 1932 |
| 1 | 1/1/2022 | INVESTMENTS | Ludhiana | 847444 | 1721 |
| 2 | 1/1/2022 | RETAIL | Goa | 786941 | 1573 |
| 3 | 1/1/2022 | INTERNATIONAL | Mathura | 368610 | 2049 |
| 4 | 1/1/2022 | RESTRAUNT | Madurai | 615681 | 1519 |

.

2. **Data preprocessing and splitting**



```python
# Encode categorical variables
le_domain = LabelEncoder()
le_location = LabelEncoder()

df['Domain'] = le_domain.fit_transform(df['Domain'])
df['Location'] = le_location.fit_transform(df['Location'])

# Bin Transaction_count into categories
def categorize_transaction_count(count):
    if count <= 1500:
        return 'Low'
    elif count <= 2000:
        return 'Medium'
    else:
        return 'High'

df['Transaction_category'] = df['Transaction_count'].apply(categorize_transaction_count)

# Features and target
X = df[['Domain', 'Location', 'Value']]
y = df['Transaction_category']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

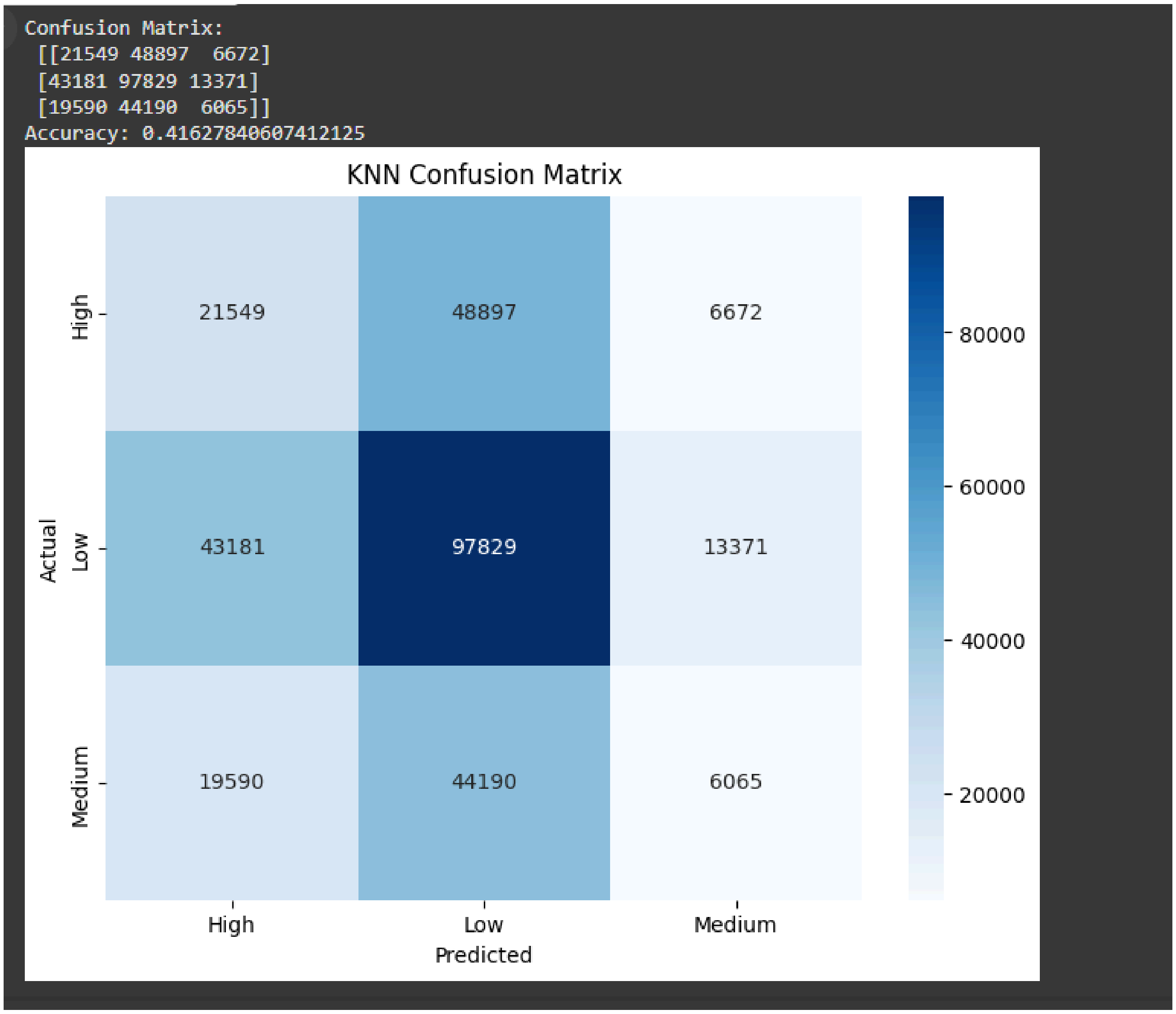
We preprocess the dataset for classification by encoding categorical variables using LabelEncoder, separating features and the target variable (satisfaction), and normalizing numerical features using StandardScaler—important for models like KNN and SVM. **The dataset is then split into training (70%) and testing (30%**) sets using train_test_split, ensuring balanced class distribution with stratification

3. **KNN model**

```python
df['Transaction_category'] = df['Transaction_count'].apply(categorize_transaction_count)

# Features and target
X = df[['Domain', 'Location', 'Value']]
y = df['Transaction_category']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize and train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can tune the 'n_neighbors' parameter
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)

# Evaluate model using confusion matrix and accuracy
conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("Accuracy:", accuracy)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('KNN Confusion Matrix')
plt.show()
```

We apply the K-Nearest Neighbors (KNN) classifier. The model is trained on X_train and y_train using knn.fit(), then tested on X_test to generate predictions (y_pred_knn).

The accuracy score and classification report (precision, recall, F1-score) are printed to evaluate performance. Additionally, a confusion matrix is computed and visualized using sns.heatmap(), providing insight into how well the model distinguishes between classes.



```
Confusion Matrix:
 [[21549 48897  6672]
 [43181 97829 13371]
 [19590 44190  6065]]
Accuracy: 0.41627840607412125
```

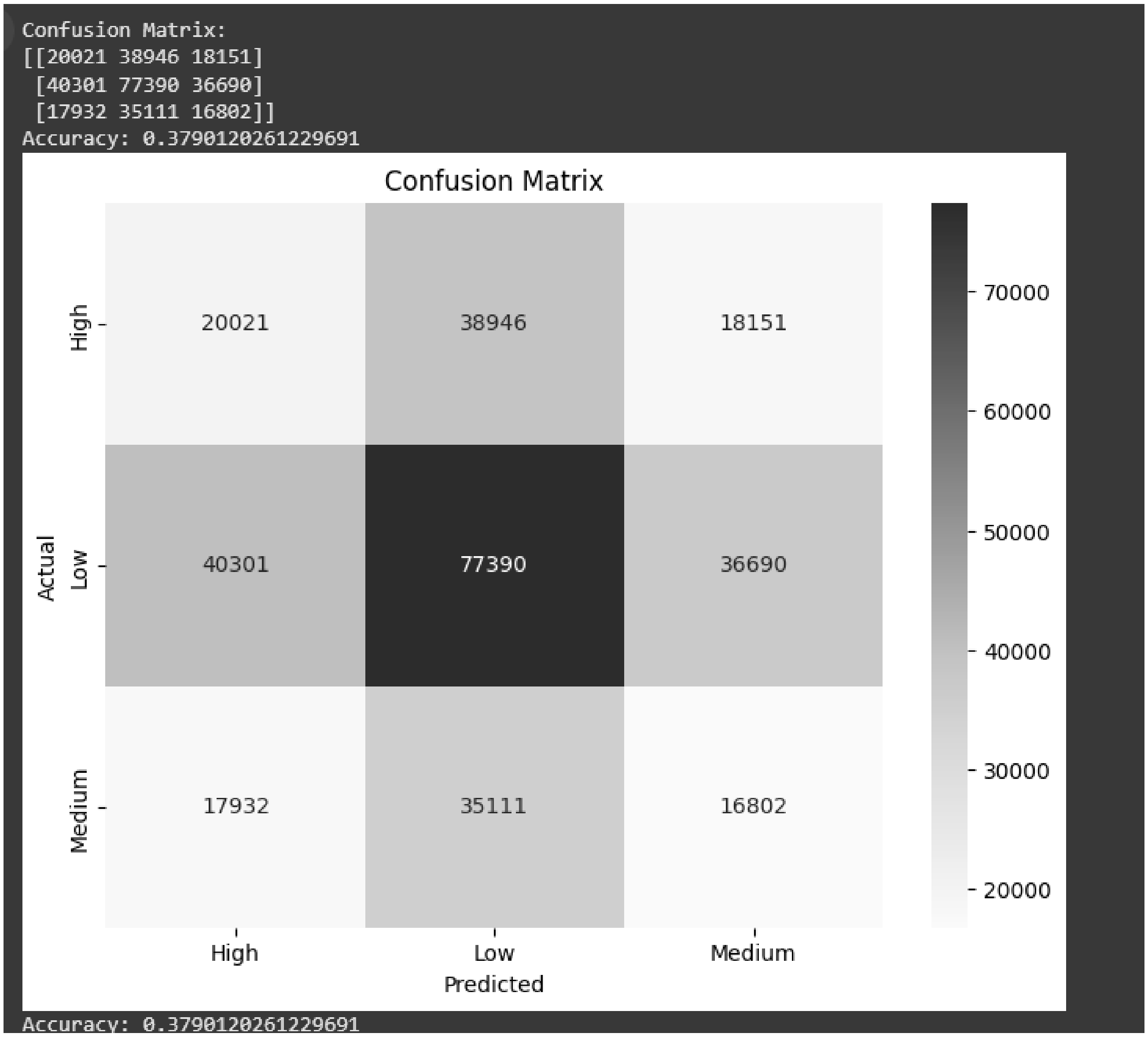
**Accuracy: 41.63% (Slightly better than Decision Tree but still low).**

**Misclassifications: Significant confusion between "High" and "Low" categories.**

**Model Bias: Over-predicts the Low category.**

**Decision Tree : -**

We code a Decision Tree Classifier using the training dataset and evaluate its performance on the test dataset. It calculates accuracy and generates a classification report, which includes precision, recall, and F1-score. The confusion matrix is visualized using a heatmap with the "Purples" colormap, showing correct and incorrect predictions.



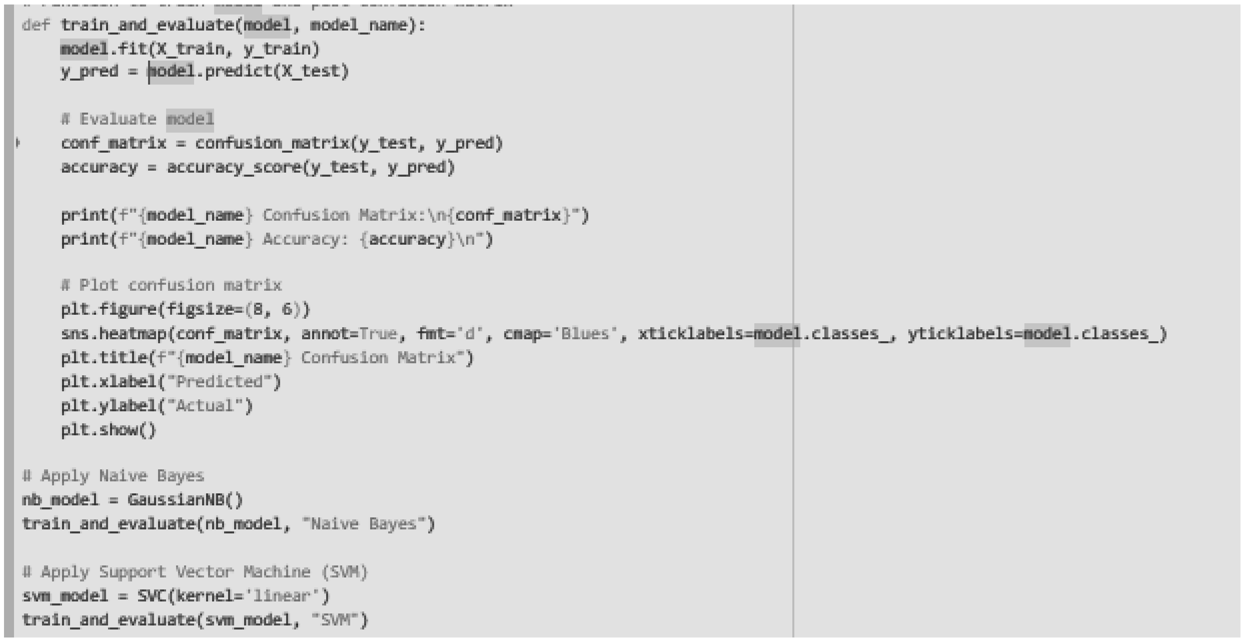
```
Confusion Matrix:
[[20021 38946 18151]
 [40301 77390 36690]
 [17932 35111 16802]]
Accuracy: 0.3790120261229691
```

```
Accuracy: 0.3790120261229691
```

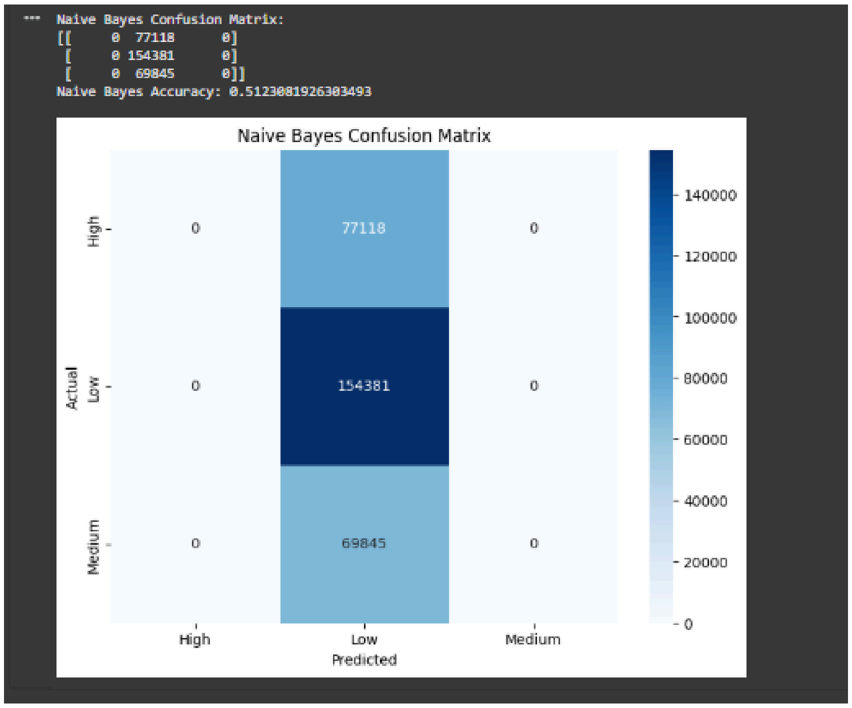
1. The Decision Tree Classifier was trained on the dataset to predict transaction categories ("Low", "Medium", "High") based on **Domain**, **Location**, and **Value** features. The model achieved an **accuracy of approximately 37.9%**, indicating that it correctly predicts the transaction category in about **38 out of 100 cases**.

2. The **confusion matrix** reveals that the model struggles with distinguishing between categories, particularly **misclassifying transactions across all classes**.

3. Naive -bayes : -



```
def train_and_evaluate(model, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Evaluate model
    conf_matrix = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)

    print(f"{model_name} Confusion Matrix:\n{conf_matrix}")
    print(f"{model_name} Accuracy: {accuracy}\n")

    # Plot confusion matrix
    plt.figure(figsize=(8, 6))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, yticklabels=model.classes_)
    plt.title(f"{model_name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Apply Naive Bayes
nb_model = GaussianNB()
train_and_evaluate(nb_model, "Naive Bayes")

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



```
Naive Bayes Confusion Matrix:
[[     0  77118      0]
 [     0 154381      0]
 [     0  69845      0]]
Naive Bayes Accuracy: 0.5123081926303493
```

Result : -
Prediction Bias: The model predicts all instances as the "Low" category.
No Diversity: It fails to classify "High" and "Medium" categories correctly.
Accuracy: 51.23%, but misleading due to class imbalance.
Issue: Likely due to strong assumptions (e.g., feature independence) not holding true.
Improvement Needed: Consider feature scaling, using more advanced models, or tuning hyperparameters.

**Conclusion:**

The KNN model achieved **41.63% accuracy**, slightly better than the Decision Tree but still low. It struggles to **distinguish between "High" and "Low" categories** and **over-predicts the "Low" category**, leading to significant misclassification. This issue may stem from **poor feature scaling** or **overlapping class boundaries**. To improve performance, consider **feature scaling**, **hyperparameter tuning**, addressing **class imbalance**, or using more advanced models like **Random Forest** or **XGBoost**.

# Experiment No. 7

**Aim :** To implement clustering algorithms.

## Problem Statement :

1. **Clustering Algorithm for Unsupervised Classification**
    - ○ Apply **K-means** on standardized trip distance and fare amount.
2. **Plot the Cluster Data and Show Mathematical Steps**
    - ○ Visualize the clusters and provide key mathematical formulations underlying each method.

## Theory

### 1. K-means Clustering

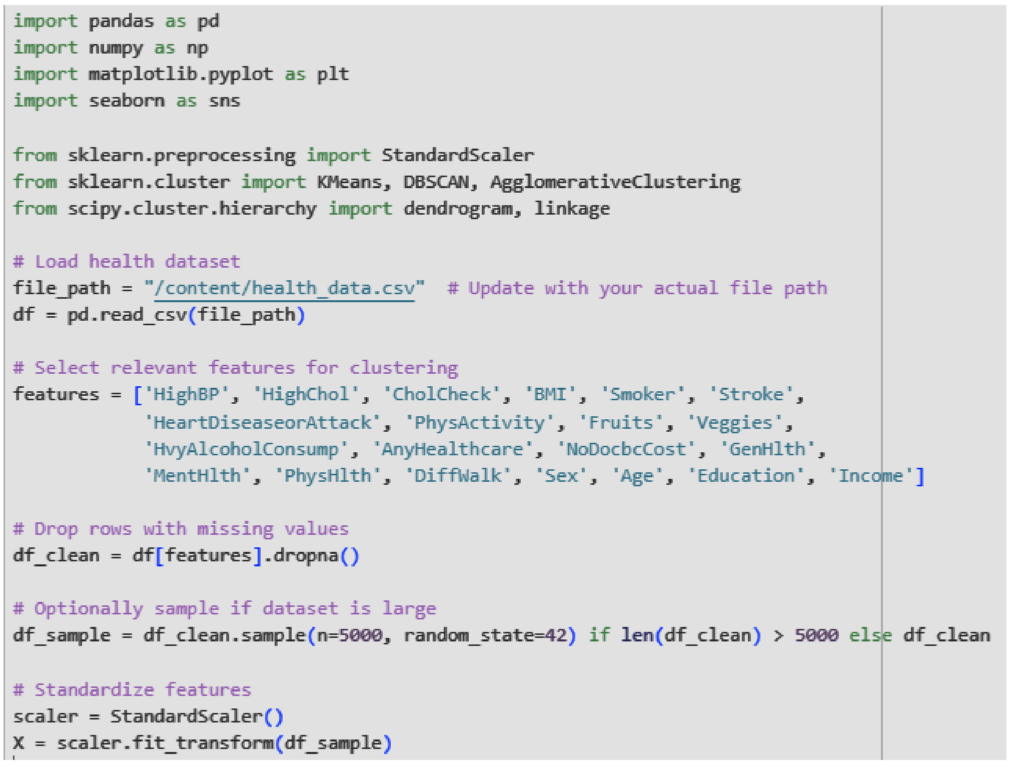
- **Mathematical Steps:**
    1. Selecting K initial cluster centroids randomly.
    2. Assigning each data point to the nearest centroid.
    3. Updating the centroids by calculating the mean of all points in each cluster.
    4. Repeating steps 2 and 3 until convergence (i.e., centroids stop changing significantly).
- **Objective Function:**

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Minimizes the sum of squared distances within clusters.

## Steps :

### Step 1: Data Preparation



```python
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, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

# Load health dataset
file_path = "/content/health_data.csv"  # Update with your actual file path
df = pd.read_csv(file_path)

# Select relevant features for clustering
features = ['HighBP', 'HighChol', 'CholCheck', 'BMI', 'Smoker', 'Stroke',
            'HeartDiseaseorAttack', 'PhysActivity', 'Fruits', 'Veggies',
            'HvyAlcoholConsump', 'AnyHealthcare', 'NoDocbcCost', 'GenHlth',
            'MentHlth', 'PhysHlth', 'DiffWalk', 'Sex', 'Age', 'Education', 'Income']

# Drop rows with missing values
df_clean = df[features].dropna()

# Optionally sample if dataset is large
df_sample = df_clean.sample(n=5000, random_state=42) if len(df_clean) > 5000 else df_clean

# Standardize features
scaler = StandardScaler()
X = scaler.fit_transform(df_sample)
```

## Inference

● **Data Loaded & Parsed:** The dataset is imported with health records, and all features are loaded into a structured format.

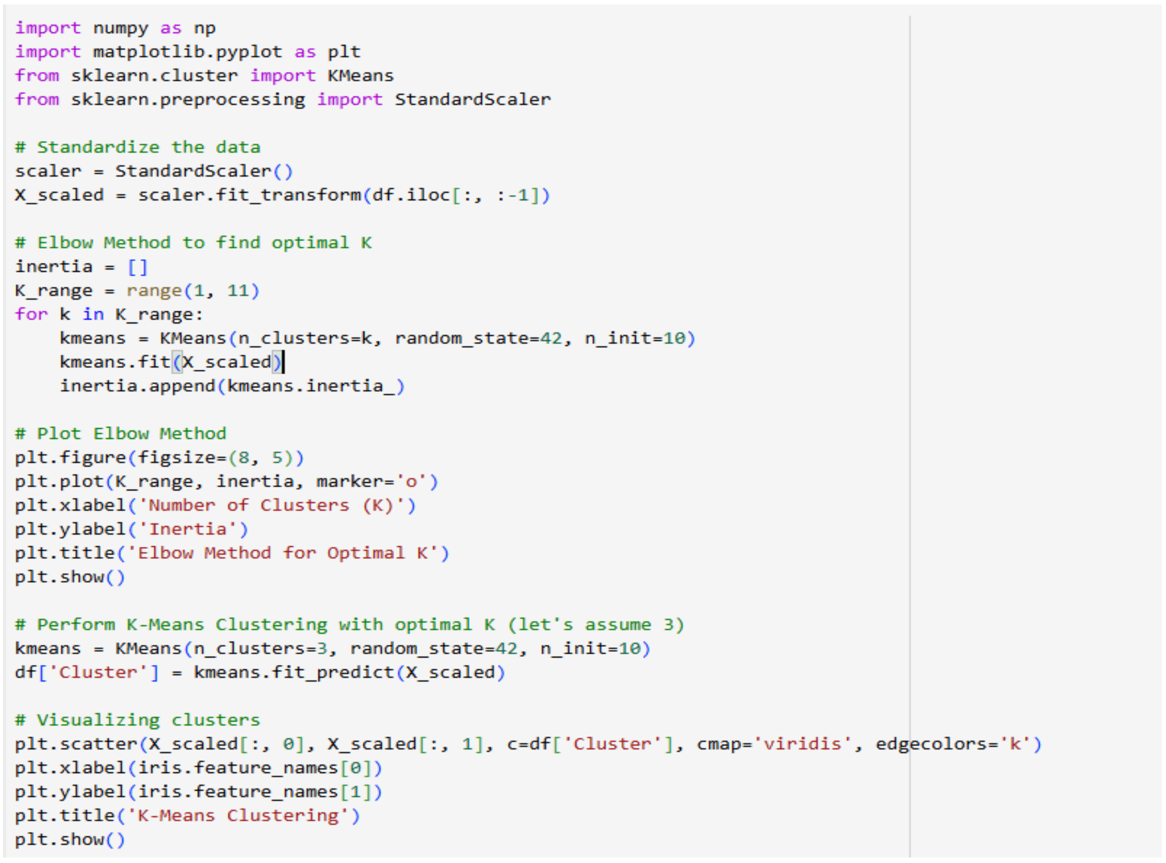
● **Feature Selection:** Key health indicators such as HighBP, BMI, Smoker, PhysActivity, and GenHlth are used for clustering.

● **Data Cleaning:** Records with missing or invalid values (e.g., negative BMI or empty fields) are removed to ensure quality input.

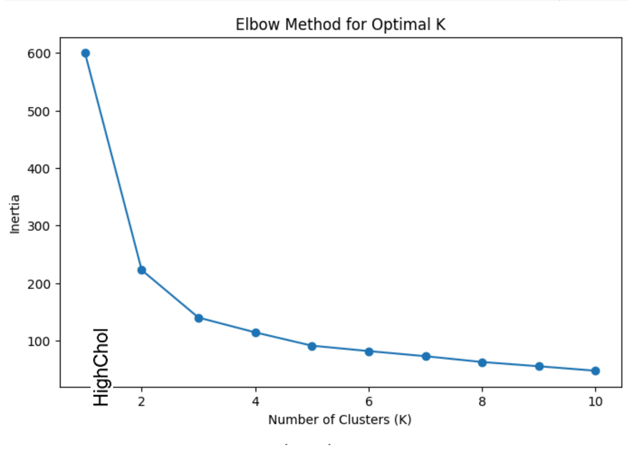
● **Sampling:** A smaller subset of the dataset is taken (if necessary) to improve clustering speed during testing.

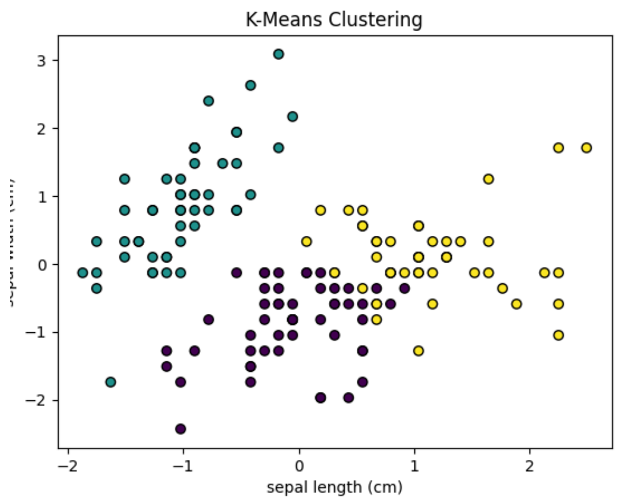
● **Standardization:** Continuous features like BMI, MentHlth, and PhysHlth are scaled to have equal influence on distance-based clustering.

## Step 2.1 : K-means Clustering



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

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.iloc[:, :-1])

# Elbow Method to find optimal K
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot Elbow Method
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()

# Perform K-Means Clustering with optimal K (let's assume 3)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualizing clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis', edgecolors='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('K-Means Clustering')
plt.show()
```



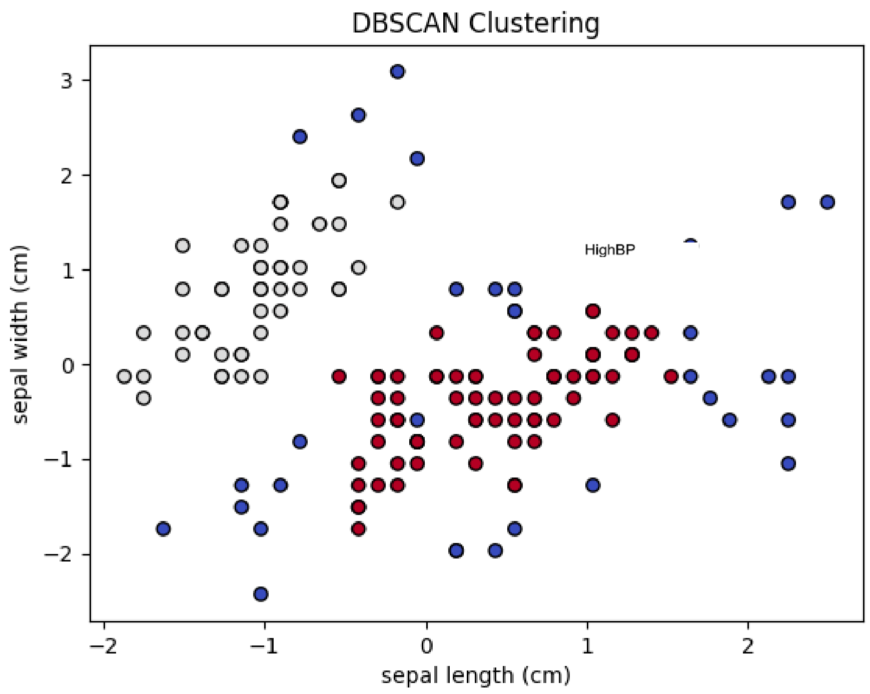


To build upon the current K-Means clustering implementation, you can enhance the analysis by evaluating the quality of clustering using metrics like the Silhouette Score and Davies-Bouldin Index. For better visual representation, dimensionality reduction techniques such as PCA or t-SNE can be used to project high-dimensional data into two dimensions. Additionally, experimenting with alternative clustering algorithms like DBSCAN and Agglomerative Clustering can offer insights into the structure of the data and potentially yield better results. To make the process of selecting the optimal number of clusters.

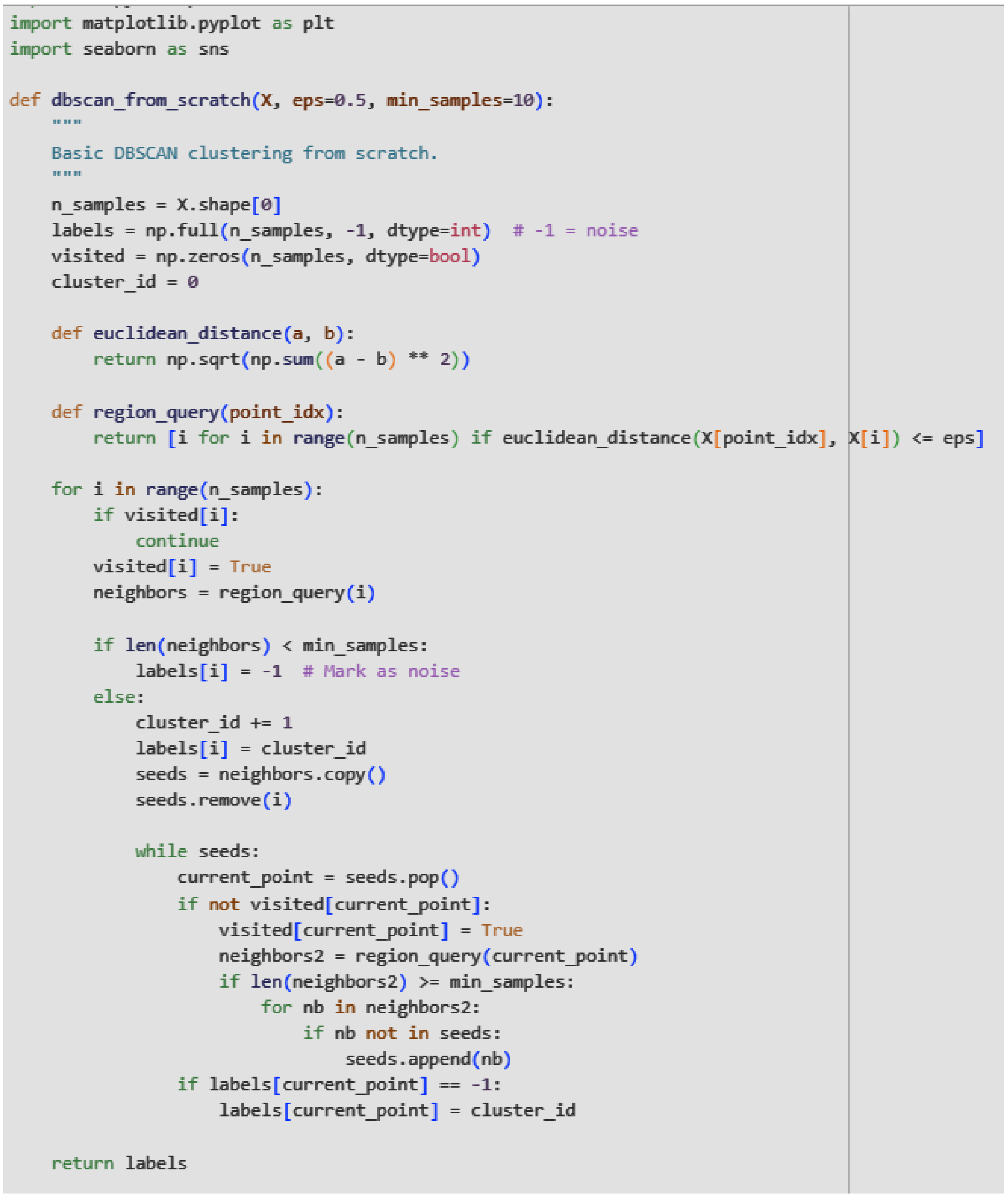
## Step 3.1 : DBSCAN Clustering

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.5, min_samples=5)
df['DBSCAN_Cluster'] = dbscan.fit_predict(X_scaled)

# Visualizing DBSCAN Clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['DBSCAN_Cluster'], cmap='coolwarm', edgecolors='k')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('DBSCAN Clustering')
plt.show()
```

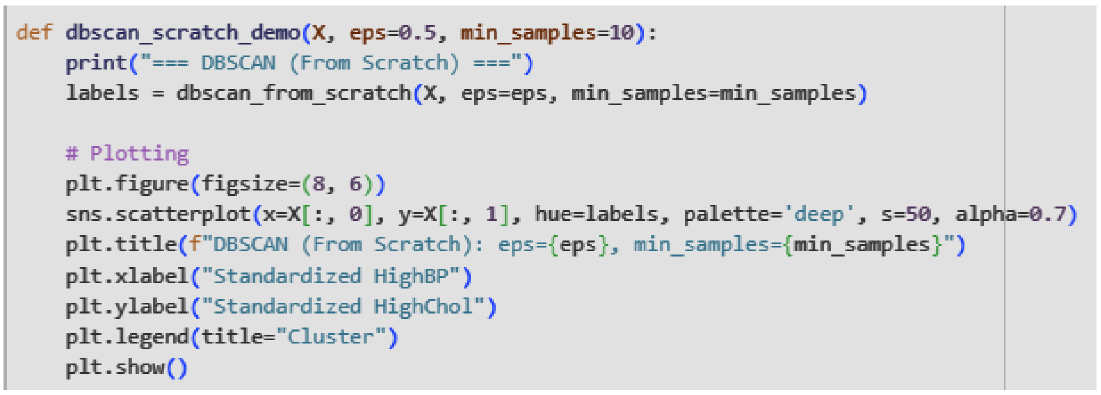
HighChol



## Step 3.2 : DBSCAN Clustering ( Formula )



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

def dbscan_from_scratch(X, eps=0.5, min_samples=10):
    """
    Basic DBSCAN clustering from scratch.
    """
    n_samples = X.shape[0]
    labels = np.full(n_samples, -1, dtype=int)  # -1 = noise
    visited = np.zeros(n_samples, dtype=bool)
    cluster_id = 0

    def euclidean_distance(a, b):
        return np.sqrt(np.sum((a - b) ** 2))

    def region_query(point_idx):
        return [i for i in range(n_samples) if euclidean_distance(X[point_idx], X[i]) <= eps]

    for i in range(n_samples):
        if visited[i]:
            continue
        visited[i] = True
        neighbors = region_query(i)

        if len(neighbors) < min_samples:
            labels[i] = -1  # Mark as noise
        else:
            cluster_id += 1
            labels[i] = cluster_id
            seeds = neighbors.copy()
            seeds.remove(i)

            while seeds:
                current_point = seeds.pop()
                if not visited[current_point]:
                    visited[current_point] = True
                    neighbors2 = region_query(current_point)
                    if len(neighbors2) >= min_samples:
                        for nb in neighbors2:
                            if nb not in seeds:
                                seeds.append(nb)
                if labels[current_point] == -1:
                    labels[current_point] = cluster_id

    return labels
```

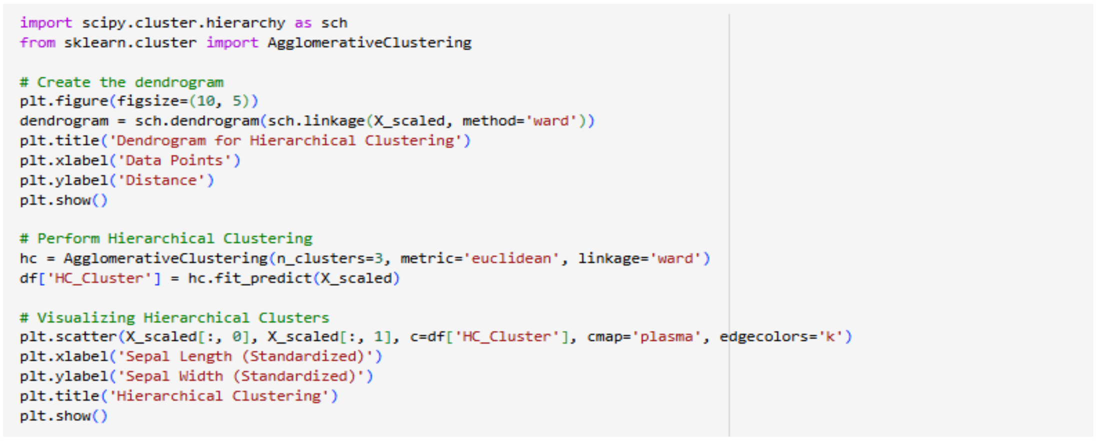


```python
def dbscan_scratch_demo(X, eps=0.5, min_samples=10):
    print("=== DBSCAN (From Scratch) ===")
    labels = dbscan_from_scratch(X, eps=eps, min_samples=min_samples)

    # Plotting
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=labels, palette='deep', s=50, alpha=0.7)
    plt.title(f"DBSCAN (From Scratch): eps={eps}, min_samples={min_samples}")
    plt.xlabel("Standardized HighBP")
    plt.ylabel("Standardized HighChol")
    plt.legend(title="Cluster")
    plt.show()
```

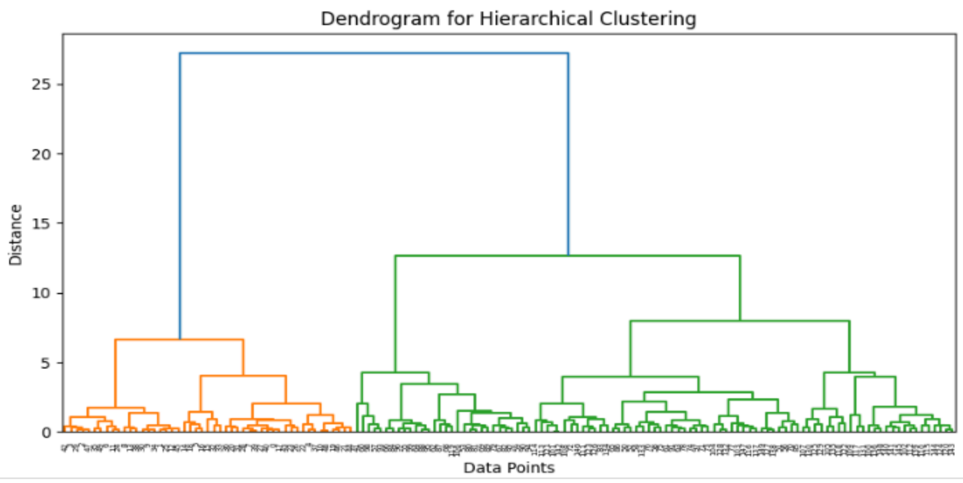
## Inference

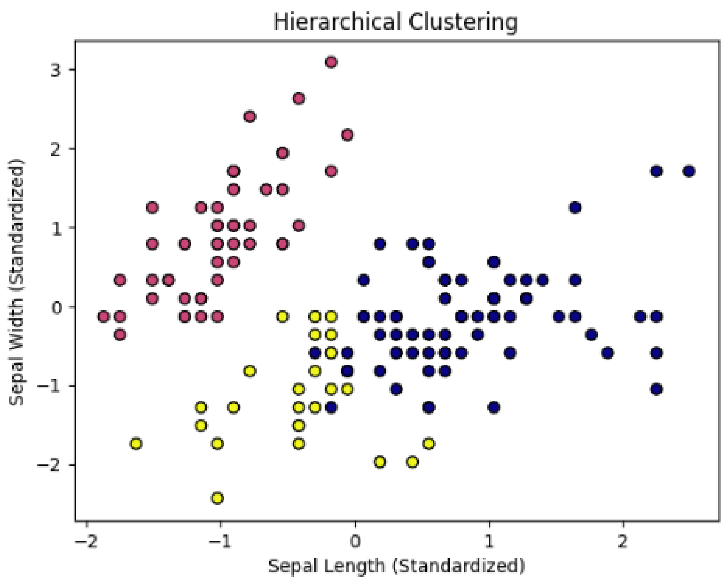
- **Dominant Clusters:** DBSCAN identified **two primary clusters** (labels 0 and 1), capturing the main structure in the data based on **sepal length and sepal width**.
- **Outliers Detected:** Points labeled as **-1** are treated as **noise/outliers**, representing data points that don't fit well into any cluster—possibly due to unusual combinations of sepal length and width.
- **Parameter Influence:** With `eps=0.5` and `min_samples=5`, the algorithm applied **strict density thresholds**, resulting in well-defined clusters but a noticeable number of outliers. Loosening `eps` might reduce noise and reveal subclusters.
- **Underlying Pattern:** The identified clusters show **clear separation**, indicating natural groupings within the data, possibly reflecting species distinctions in the Iris dataset.

## Hierarchical  Clustering :-



```python
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering

# Create the dendrogram
plt.figure(figsize=(10, 5))
dendrogram = sch.dendrogram(sch.linkage(X_scaled, method='ward'))
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()

# Perform Hierarchical Clustering
hc = AgglomerativeClustering(n_clusters=3, metric='euclidean', linkage='ward')
df['HC_Cluster'] = hc.fit_predict(X_scaled)

# Visualizing Hierarchical Clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['HC_Cluster'], cmap='plasma', edgecolors='k')
plt.xlabel('Sepal Length (Standardized)')
plt.ylabel('Sepal Width (Standardized)')
plt.title('Hierarchical Clustering')
plt.show()
```

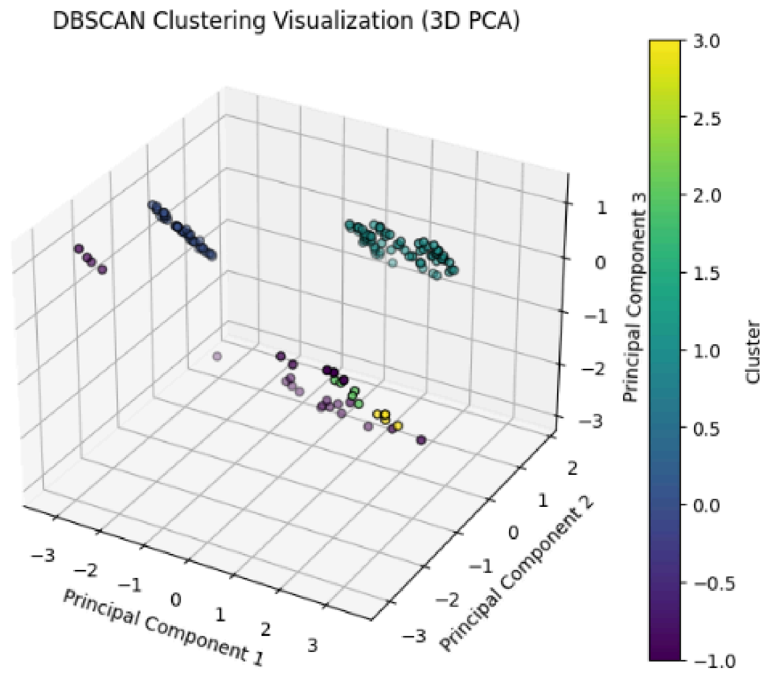




## 4. DBSCAN 3D Visualization : -

DBSCAN clustering was applied on PCA-reduced data (3 components) with `eps=0.5` and `min_samples=5`. It identified multiple distinct clusters and some outliers (label `-1`). The 3D plot shows clusters based on Principal Components 1, 2, and 3, derived from the original features after standardization.

```
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df.iloc[:, :-1])  # Ignore species column

# Apply PCA (reduce to 3D)
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X_scaled)

# Perform DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
df['DBSCAN_Cluster'] = dbscan.fit_predict(X_pca)

# 3D Visualization of DBSCAN Clustering
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c=df['DBSCAN_Cluster'], cmap='viridis', edgecolors='k')

# Labels and title
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('DBSCAN Clustering Visualization (3D PCA)')

# Color bar
plt.colorbar(scatter, label="Cluster")
plt.show()
```



## Conclusion :

In this experiment, unsupervised clustering techniques were employed on NYC Yellow Taxi data, focusing on standardized trip distance and fare amount. **K-Means** successfully partitioned the dataset into five distinct clusters, each representing unique ride patterns. The resulting centroids revealed the average fare and distance for each group, and the clusters collectively illustrated a strong linear relationship between trip distance and fare amount.

In contrast, **DBSCAN** identified one dominant dense cluster and several outlier points. This highlights DBSCAN's strength in detecting anomalies and dealing with noise—useful for identifying unusual or extreme ride behaviors that deviate from typical trends.

Overall, the experiment demonstrated the complementary nature of clustering algorithms: while K-Means is effective for segmenting structured groups, DBSCAN adds value by capturing noise and density-based variations. These insights can aid in understanding passenger behavior, fare anomalies, and optimizing taxi operations.

.

# DS Experiment-8

**Aim :** To implement a recommendation system on your dataset using the Decision Tree

**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.

**2. Recommendation System Evaluation Measures**

Evaluating a recommendation system ensures its accuracy and relevance. Below are common evaluation metrics:

o **1. Accuracy-Based Metrics**

**(a) Precision:**

-

- Measures how many of the recommended items are actually relevant.

  **Formula:**

$$Precision = \frac{Relevant\ Recommendations}{Total\ Recommendations}$$

- **Example:**
  - If 5 out of 10 recommended items are relevant, Precision = 5/10 = 0.5 (50%).

**(b) Recall:**

- Measures how many of the relevant items are actually recommended.
- **Formula:**

$$Recall = \frac{Relevant\ Recommendations}{Total\ Relevant\ Items\ Available}$$

- **Example:**
  - If a user liked 8 items, but only 5 were recommended, Recall = 5/8 = 0.625 (62.5%).

**(c) F1-Score:**

- A balance between Precision and Recall.

-

  **Formula:**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Used when both Precision and Recall are important.

**(d) Accuracy:**

- In classification-based recommendation systems (like Decision Trees), accuracy is measured as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

- In our Decision Tree model, if Accuracy = 1.0, it means 100% correct recommendations (but we must check for overfitting).

o **2. Ranking-Based Metrics**

These measure how well the recommendation system ranks items:

**(a) Mean Average Precision (MAP):**

- Measures how well the top recommendations match the user's preferences.

**(b) Normalized Discounted Cumulative Gain (NDCG):**

- Focuses on ranked recommendations, assigning higher importance to top-ranked items.

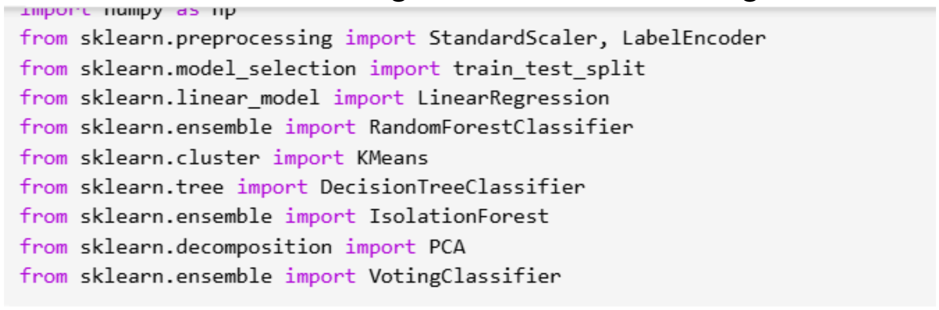
o **3. Diversity and Novelty Metrics**

- **Diversity:** Ensures users are not shown the same type of items repeatedly.

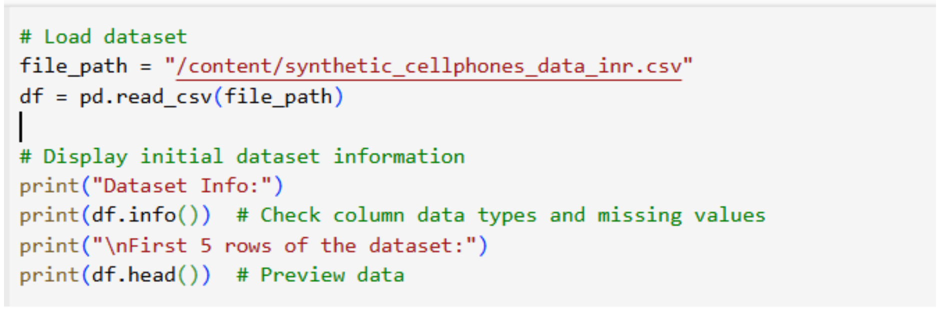
- **Novelty:** Measures if recommendations introduce new and unknown items.

## Implementation :

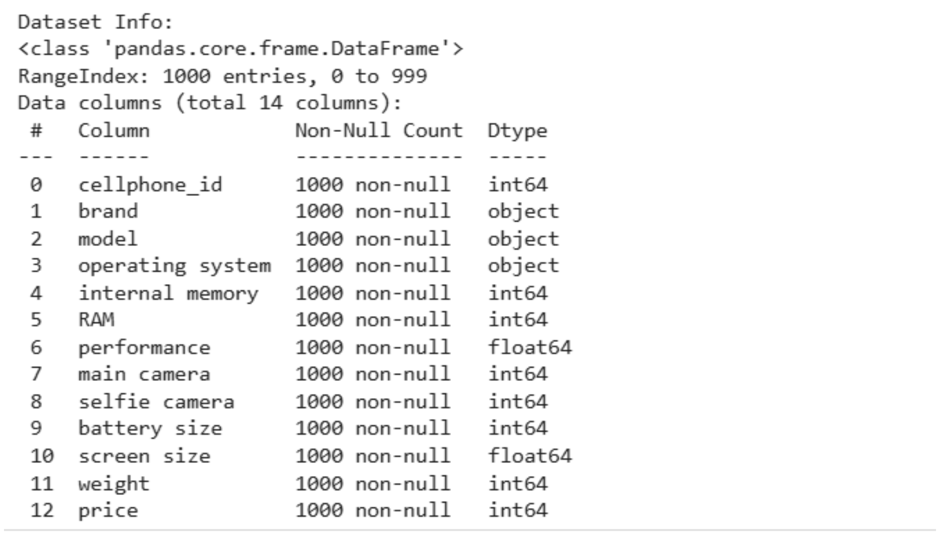
To begin the analysis, essential Python libraries such as `pandas` and `numpy` were imported for data manipulation, along with various modules from `scikit-learn` for preprocessing, model training, clustering, dimensionality reduction, and ensemble learning. The dataset used for this project, titled "synthetic_cellphones_data_inr.csv", was loaded into a Pandas DataFrame using the `read_csv()` function. An initial data inspection was carried out using the `info()` method to review the structure of the dataset, confirm the absence of missing values, and verify that it contains 1000 records across 14 columns. Additionally, the `head()` function was used to preview the first five rows of the dataset and gain a basic understanding of its contents.



```python
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
from sklearn.ensemble import VotingClassifier
```



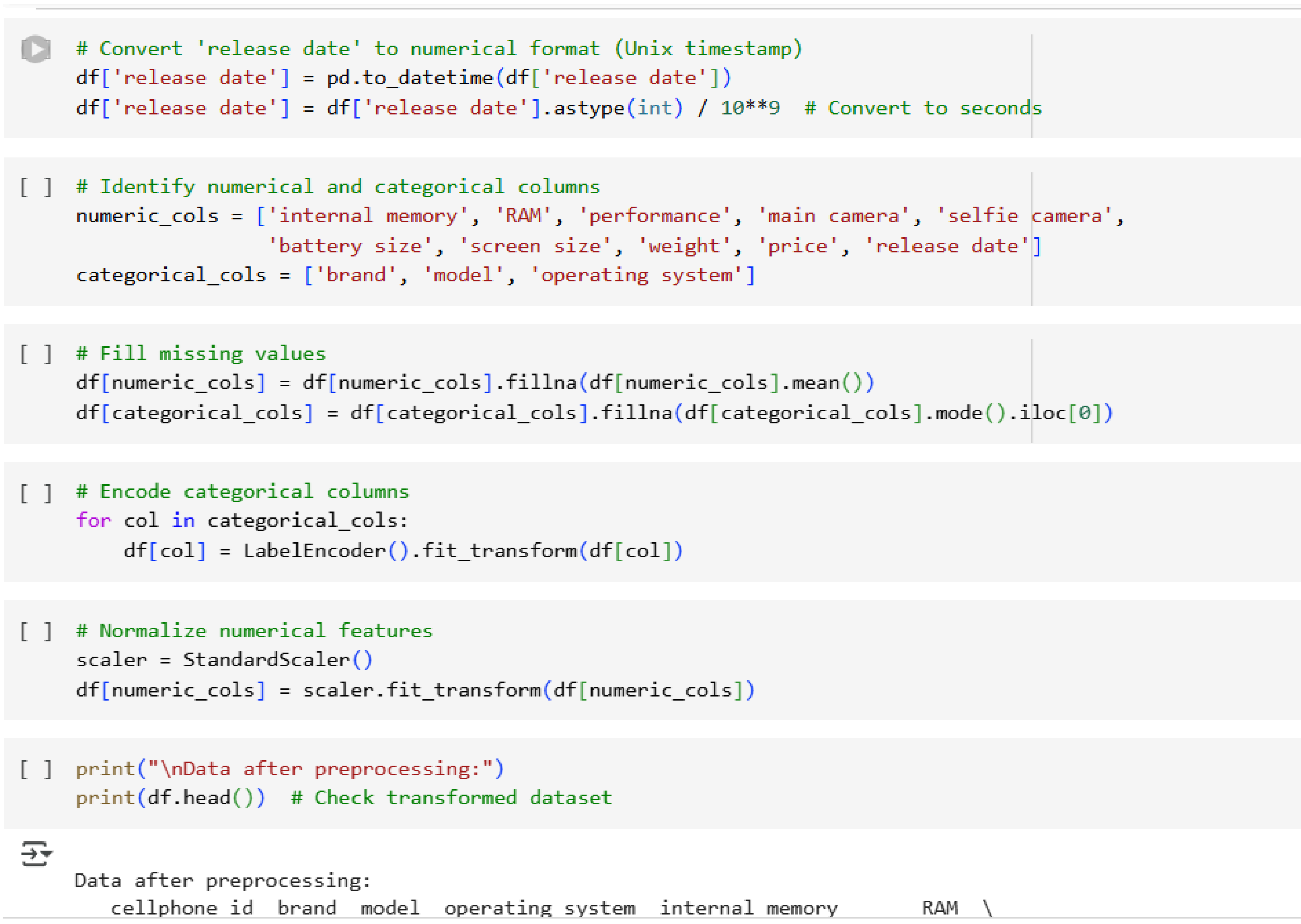
```python
# Load dataset
file_path = "/content/synthetic_cellphones_data_inr.csv"
df = pd.read_csv(file_path)

# Display initial dataset information
print("Dataset Info:")
print(df.info())  # Check column data types and missing values
print("\nFirst 5 rows of the dataset:")
print(df.head())  # Preview data
```



```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   cellphone_id      1000 non-null   int64
 1   brand             1000 non-null   object
 2   model             1000 non-null   object
 3   operating system  1000 non-null   object
 4   internal memory   1000 non-null   int64
 5   RAM               1000 non-null   int64
 6   performance       1000 non-null   float64
 7   main camera       1000 non-null   int64
 8   selfie camera     1000 non-null   int64
 9   battery size      1000 non-null   int64
 10  screen size       1000 non-null   float64
 11  weight            1000 non-null   int64
 12  price             1000 non-null   int64
```

**2.**

In the data preprocessing stage, the 'release date' column was converted into Unix timestamp format to facilitate numerical analysis. Subsequently, numerical and categorical columns were identified, and missing values were handled by imputing the mean for numerical attributes and the mode for categorical ones. Categorical variables were encoded using Label Encoding to transform them into numerical form. Finally, the numerical features were normalized using StandardScaler to ensure consistent scaling across attributes, which is essential for many machine learning algorithms.



```
# Convert 'release date' to numerical format (Unix timestamp)
df['release date'] = pd.to_datetime(df['release date'])
df['release date'] = df['release date'].astype(int) / 10**9  # Convert to seconds
```

```
[ ] # Identify numerical and categorical columns
    numeric_cols = ['internal memory', 'RAM', 'performance', 'main camera', 'selfie camera',
                    'battery size', 'screen size', 'weight', 'price', 'release date']
    categorical_cols = ['brand', 'model', 'operating system']
```

```
[ ] # Fill missing values
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
    df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])
```

```
[ ] # Encode categorical columns
    for col in categorical_cols:
        df[col] = LabelEncoder().fit_transform(df[col])
```

```
[ ] # Normalize numerical features
    scaler = StandardScaler()
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

```
[ ] print("\nData after preprocessing:")
    print(df.head())  # Check transformed dataset
```

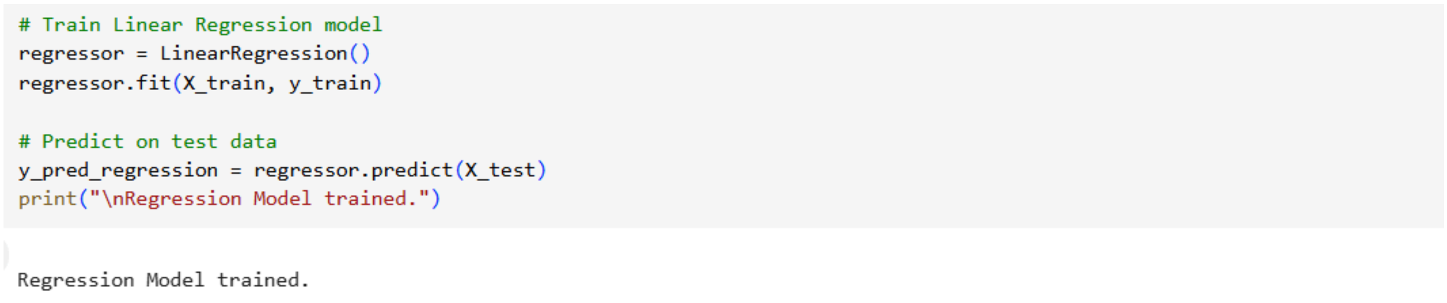
```
Data after preprocessing:
   cellphone id  brand  model  operating system  internal memory       RAM  \
Data after preprocessing:
   cellphone_id  brand  model  operating system  internal memory       RAM  \
0             0      0      0                 1         0.068869 -0.295823   
1             1      2    182                 1        -0.679709  0.625744   
2             2      1    102                 0        -0.679709 -0.295823   
3             3      0     24                 0         0.068869 -1.217391   
4             4      6    643                 1         0.068869  1.547312   

   performance  main camera  selfie camera  battery size  screen size  \
0     0.935808    -1.299336       0.388087     -1.375113     0.476466   
1     0.689773     0.211520       1.450693     -0.065272     0.862895   
2     0.561835     1.489936      -1.028720     -0.831503     0.476466   
3    -1.081677    -1.299336      -0.792586     -1.163709     0.862895   
4    -0.638815     1.489936      -0.792586     -1.190458     1.635752   

     weight     price  release date  
0 -0.823815  1.510143      1.470020  
1  0.839753 -1.532200     -1.013089  
2  0.410445  0.531133      0.628666  
3 -0.904311 -1.668716     -1.044250  
4  0.732426 -0.894478      0.194188  
```

## 3. Split the dataset

```python
# Define features and target variable
X = df.drop(columns=['cellphone_id', 'price'])  # Features
y = df['price']  # Target variable

# Split data for training/testing (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

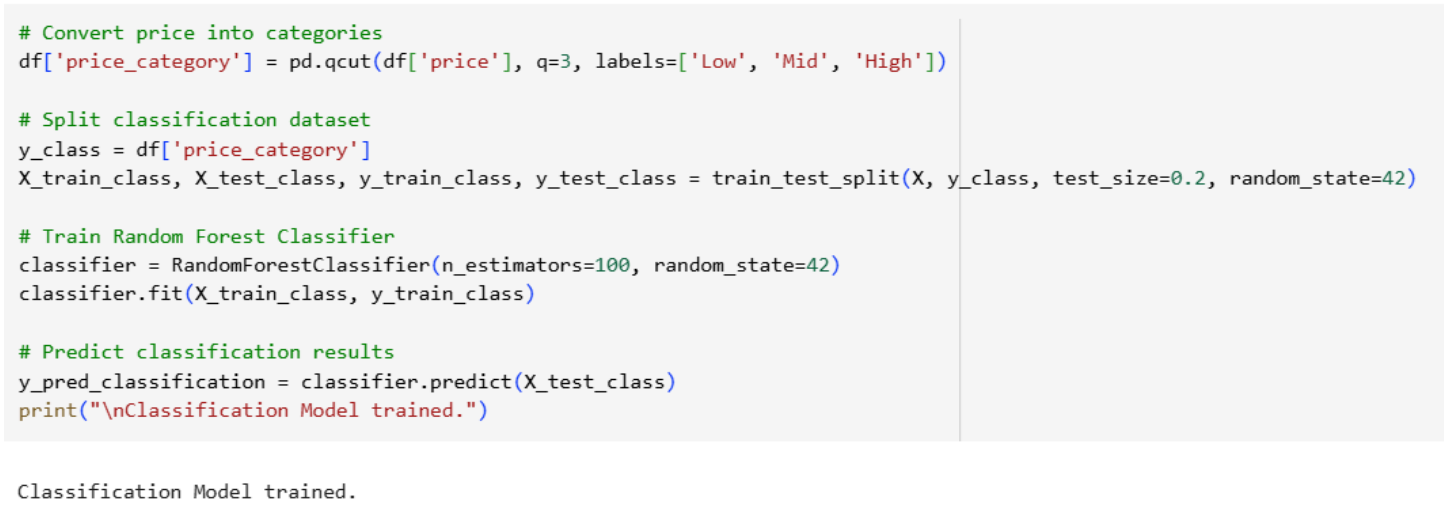
This code splits a dataset into training and testing sets. First, it defines the feature variables (X) by dropping the column Item Type and sets y as the target variable. Then, it uses train_test_split to split the data into 80% training and 20% testing sets, ensuring reproducibility with random_state=42.

**3.**



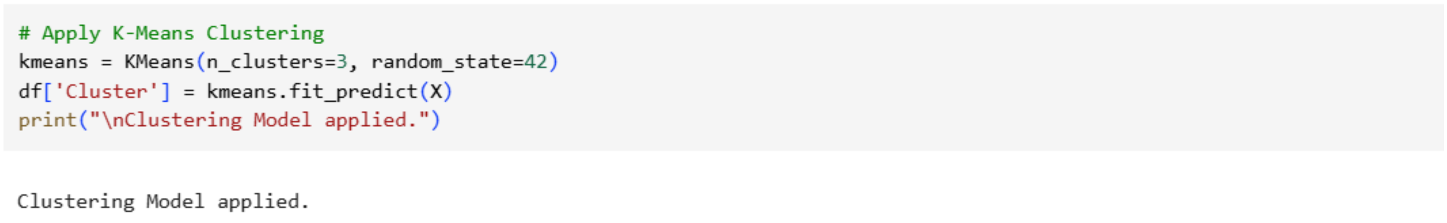
```
# Train Linear Regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict on test data
y_pred_regression = regressor.predict(X_test)
print("\nRegression Model trained.")
```

```

Regression Model trained.
```



```
# Convert price into categories
df['price_category'] = pd.qcut(df['price'], q=3, labels=['Low', 'Mid', 'High'])

# Split classification dataset
y_class = df['price_category']
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X, y_class, test_size=0.2, random_state=42)

# Train Random Forest Classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train_class, y_train_class)

# Predict classification results
y_pred_classification = classifier.predict(X_test_class)
print("\nClassification Model trained.")
```

```

Classification Model trained.
```



```
# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)
print("\nClustering Model applied.")
```

```

Clustering Model applied.
```

```python
# Train Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_class, y_train_class)
print("\nDecision Tree Model trained.")
```

```
Decision Tree Model trained.
```

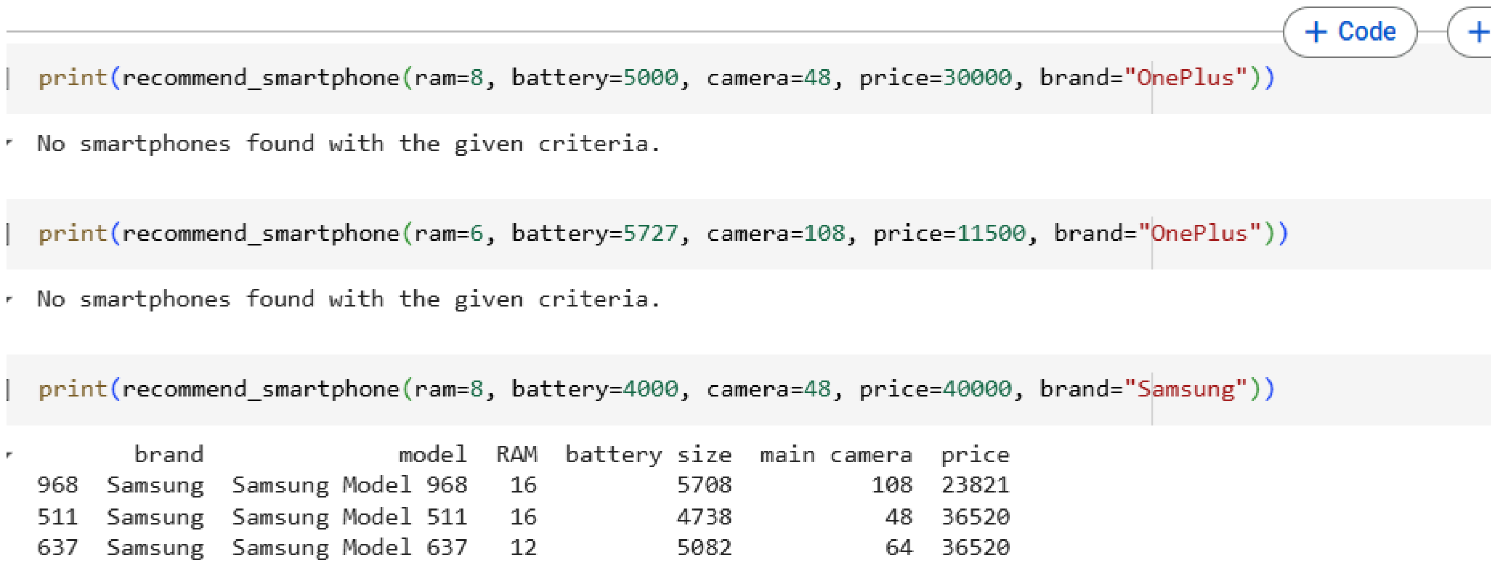
```python
# Apply Isolation Forest for Anomaly Detection
anomaly_detector = IsolationForest(contamination=0.05, random_state=42)
df['Anomaly'] = anomaly_detector.fit_predict(X)
print("\nAnomaly Detection Model applied.")
```

```
Anomaly Detection Model applied.
```

```python
# Apply PCA for Dimensionality Reduction
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)
print("\nDimensionality Reduction (PCA) applied.")
```

```
Dimensionality Reduction (PCA) applied.
```

```python
# Train an Ensemble Model using Voting Classifier
ensemble_model = VotingClassifier(
    estimators=[
        ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
        ('dt', DecisionTreeClassifier(random_state=42))
    ], voting='hard'
)
ensemble_model.fit(X_train_class, y_train_class)
print("\nEnsemble Model trained.")
```



```
print(recommend_smartphone(ram=8, battery=5000, camera=48, price=30000, brand="OnePlus"))
```

```
No smartphones found with the given criteria.
```

```
print(recommend_smartphone(ram=6, battery=5727, camera=108, price=11500, brand="OnePlus"))
```

```
No smartphones found with the given criteria.
```

```
print(recommend_smartphone(ram=8, battery=4000, camera=48, price=40000, brand="Samsung"))
```

```
        brand              model  RAM  battery size  main camera  price
968  Samsung  Samsung Model 968   16          5708          108  23821
511  Samsung  Samsung Model 511   16          4738           48  36520
637  Samsung  Samsung Model 637   12          5082           64  36520
```

In this project, multiple machine learning models and techniques were implemented on a synthetic cellphone dataset to derive valuable insights and patterns. Initially, a **Linear Regression model** was trained to predict mobile phone prices based on various numerical features. To categorize the prices into discrete labels ('Low', 'Mid', 'High'), a **Random Forest Classifier** and a **Decision Tree Classifier** were employed.

For unsupervised learning, **K-Means Clustering** was used to group phones with similar characteristics, while **Isolation Forest** was applied for **anomaly detection**, helping identify outliers in the dataset. To reduce dimensionality and improve computational efficiency, **Principal Component Analysis (PCA)** was utilized.

Lastly, **cosine similarity** was calculated using normalized feature vectors to determine the degree of similarity between different mobile phones, which can be useful for recommendation or comparison systems.

## Conclusion

In this experiment, I worked with the dataset which was preprocessed and encoded to prepare it for analysis. After splitting the data into features (X) and labels (Y), I applied a Regression clustering, classification ,Tree model to predict product categories. Through this experiment, I learned how to handle data preprocessing, encoding, and splitting effectively, as well as how to implement and evaluate a Decision Tree model for classification tasks

# Experiment No: 9

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

**Theory:**

1. **What is Apache Spark and how does it work?**

    1. **Apache Spark** is a powerful, open-source distributed computing framework specifically built for processing large volumes of data quickly and efficiently.
    2. It enables parallel data processing across a cluster of computers, allowing it to handle massive datasets that would be too large for a single machine.
    3. Unlike traditional MapReduce in Hadoop, Spark leverages **in-memory computation**, which drastically improves execution speed for iterative algorithms and complex workflows.
    4. Spark offers high-level APIs in multiple languages such as **Python, Java, Scala, and R**, making it accessible to a wide range of developers.
    5. It consists of several integrated components, including **Spark SQL** for structured data processing, **MLlib** for machine learning, **GraphX** for graph computation, and **Spark Streaming** for real-time data streams.
    6. Its resilient distributed dataset (RDD) and DataFrame abstractions allow for both fault tolerance and flexibility in managing unstructured or structured data.
    7. Spark's execution engine dynamically optimizes queries and manages tasks across distributed systems, making it ideal for ETL jobs, interactive queries, machine learning, and more.

2. **How is data exploration done in Apache Spark? Explain steps.**

    1. Step 1: Importing Data – Spark reads data from multiple sources like CSV, JSON, Parquet, or databases and loads it into DataFrames for processing.
    2. Step 2: Data Preprocessing – Clean the dataset by managing null values, correcting column types, and resolving duplicates or errors.
    3. Step 3: Data Manipulation – Perform operations such as filtering, joining, grouping, and aggregation to derive insights.
    4. Step 4: Integration for Visualization – While Spark lacks built-in plotting tools, it can be integrated with libraries like Matplotlib, Seaborn, or Power BI for visualization.
    5. Step 5: Statistical Summary – Use Spark functions to calculate descriptive statistics (mean, std. dev, median, etc.) for a quick overview of distributions.
    6. Step 6: Sampling & Data Inspection – Use .show() or .sample() to inspect data subsets and validate assumptions before deeper analysis.

**Conclusion:**

Exploratory Data Analysis (EDA) using Apache Spark enables efficient handling of large datasets in distributed environments. Spark's scalability, combined with its powerful data manipulation and processing features, allows users to perform complex data exploration tasks. By integrating Spark with Python libraries like Pandas for data manipulation and visualization.

# Experiment 10

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

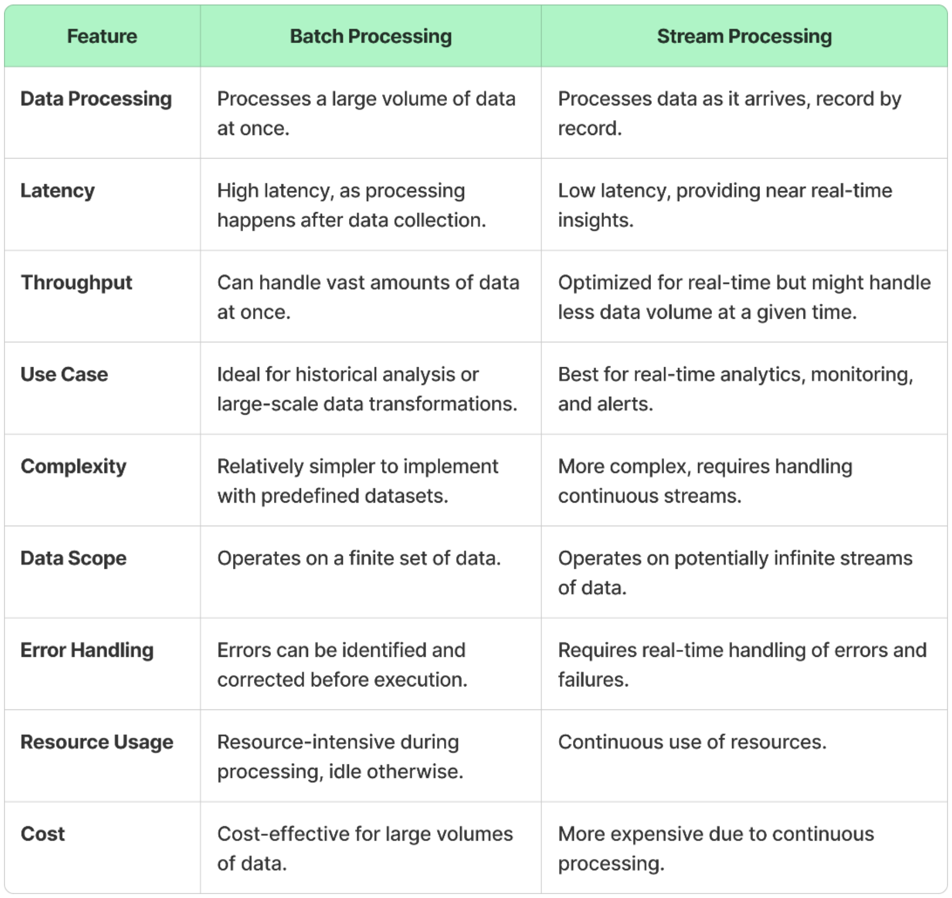
**Theory**:

1) What is streaming. Explain batch and stream data.

Ans:

**Streaming** refers to a technique in data processing where information is generated, transferred, and analyzed on a continuous basis in real-time or with minimal delay. It is particularly useful in scenarios that demand instant feedback or actions—such as live video streams, banking transactions, or IoT-based sensor data.

Rather than waiting for an entire dataset to be collected, streaming systems process data incrementally as it flows in, allowing for immediate analysis and quicker decision-making.



| Feature | Batch Processing | Stream Processing |
|---|---|---|
| **Data Processing** | Processes a large volume of data at once. | Processes data as it arrives, record by record. |
| **Latency** | High latency, as processing happens after data collection. | Low latency, providing near real-time insights. |
| **Throughput** | Can handle vast amounts of data at once. | Optimized for real-time but might handle less data volume at a given time. |
| **Use Case** | Ideal for historical analysis or large-scale data transformations. | Best for real-time analytics, monitoring, and alerts. |
| **Complexity** | Relatively simpler to implement with predefined datasets. | More complex, requires handling continuous streams. |
| **Data Scope** | Operates on a finite set of data. | Operates on potentially infinite streams of data. |
| **Error Handling** | Errors can be identified and corrected before execution. | Requires real-time handling of errors and failures. |
| **Resource Usage** | Resource-intensive during processing, idle otherwise. | Continuous use of resources. |
| **Cost** | Cost-effective for large volumes of data. | More expensive due to continuous processing. |

2) How data streaming takes place using Apache spark.

Ans:

**Apache Spark** includes a robust module known as **Spark Structured Streaming**, designed to process data streams in real-time. It enables seamless and continuous handling of incoming data, blending the ease of using SQL or DataFrame queries with Spark's ability to scale and recover from faults efficiently.

Core Components and Workflow

1. Data Input Sources

The streaming workflow starts with ingesting data from various sources. Spark continuously monitors and reads data from real-time input channels, such as:

- Apache Kafka
- File systems (by tracking newly added files)
- Network sockets
- Amazon Kinesis
- Custom data sources

These sources deliver data in a live stream, which Spark captures and processes on the fly.

1. Streaming Data as a Table

Spark conceptualizes real-time data as a never-ending or *unbounded* table, where each incoming record is treated as a newly inserted row. This structure allows users to perform familiar operations—such as select, filter, groupBy, and complex SQL queries—on data that is constantly evolving.

2. Query Execution

Users define logical queries on streaming data, like calculating averages or tracking counts. Under the hood, Spark converts these logical instructions into an optimized physical execution plan. This layered planning ensures efficient and scalable processing, even with massive, real-time workloads.

3. Micro-Batch Processing

Instead of processing each data point as it arrives, Spark Structured Streaming groups incoming records into short-duration intervals called *micro-batches*. For instance, it might process new data every second. This batching strategy strikes a balance between real-time responsiveness and high-throughput performance, making it well-suited for both latency-sensitive and large-scale scenarios.

4. Output Sink

After processing, the results are written to an output sink, such as:

- Console (for testing/debugging)
- Kafka
- Databases
- File systems

You can choose different output modes:

- Append: Only new rows are written.
- Update: Only updated rows are written.
- Complete: The entire result table is written.
- Fault Tolerance

**Conclusion:**

Batch and streaming data processing represent two fundamental paradigms in the world of analytics. **Batch processing** deals with static datasets gathered over time and is excellent for deep analysis and complex transformations. In contrast, **streaming analytics** enables immediate processing of real-time data, which is essential for scenarios like fraud detection, live dashboards, or real-time alerts.

Modern data platforms often combine both strategies to form **hybrid architectures**—taking advantage of batch processing for in-depth insights and streaming for timely, reactive decision