Experiment 1

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

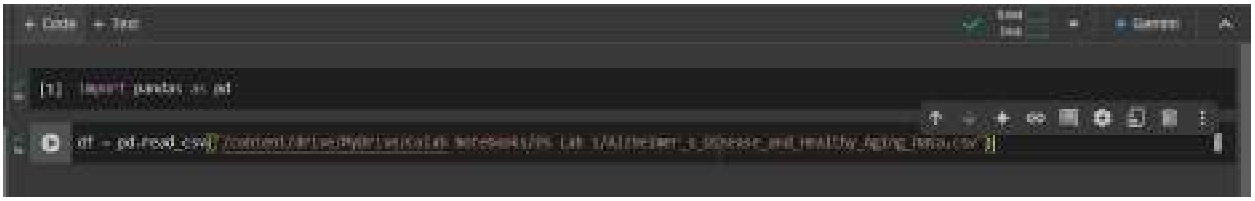
- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren’t useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- standardization and normalization of columns

Steps:

1) Load the file.

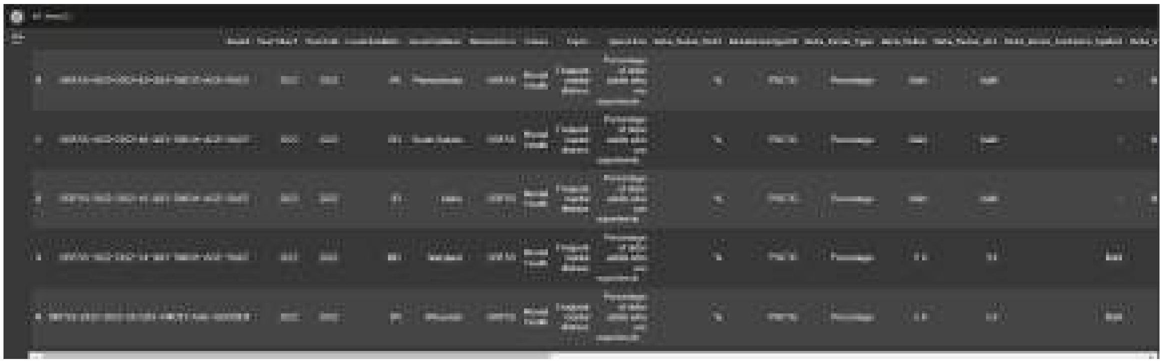
To load a file onto python for analysis, we need to make use of the pandas library. It gives us functionalities to read a CSV (Comma Separated Values) file and perform various functions on it.

Commands: **import pandas as pd** (Importing the pandas library onto Google Colab Notebook)
df = pd.read_csv(<Path_of_csv_file>) (Mounts and reads the file in Python and assigns it to variable df for ease of use further)
(Note: Replace <Path_of_csv_file> with the actual path of the file in “”)



```
[1] import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/DS Lab 1/Alzheimer_s_Disease_and_Healthy_Aging_Data.csv')
```

To check whether the file is loaded or not, we will run the command **df.head().** This command gives the first 5 rows of the dataset as the output.



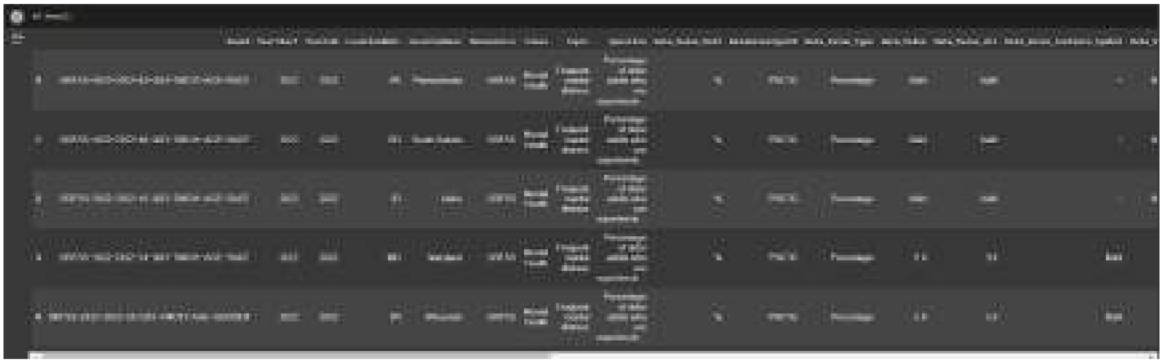
```
df.head()
```

2) Description of the dataset

The description of the dataset gives the user an idea on what are the features, what is the count of rows and columns, etc. To achieve this, we can use the following commands.

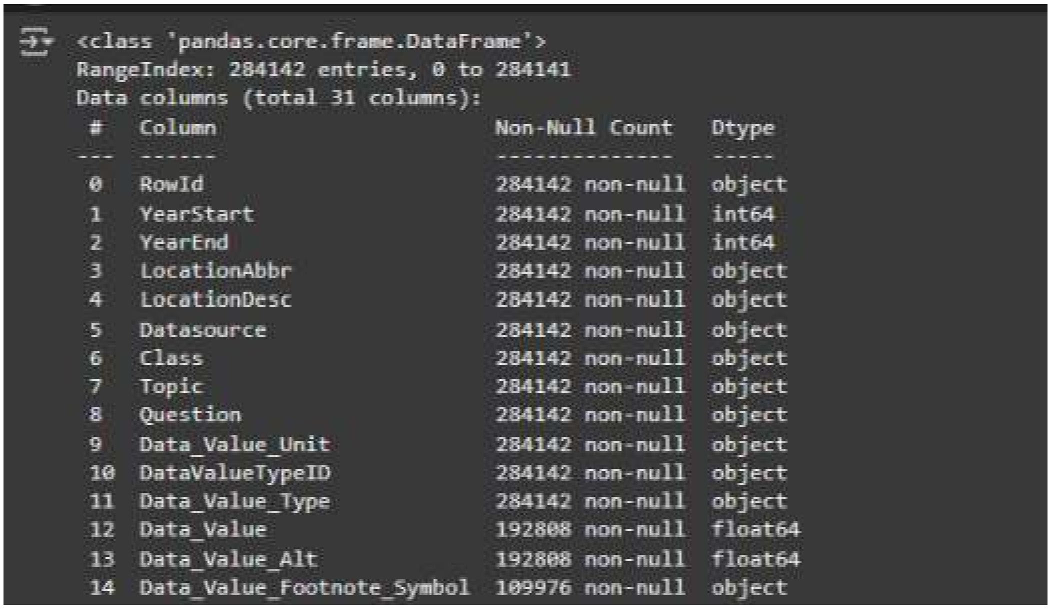
Command 1: df.head()

As mentioned before, **head** function give us the first 5 rows of the dataset. This allows for the user to get an overview on what values are being listed in the dataset.

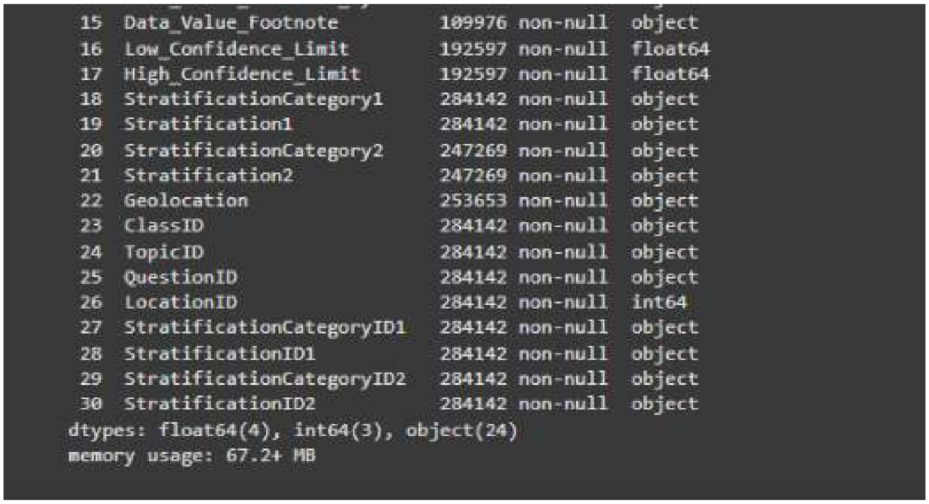


Command 2: df.info()

This command gives all the information about the features (columns) of the dataset and the data type of each of these columns. It also gives a summary of all the values in the dataset.



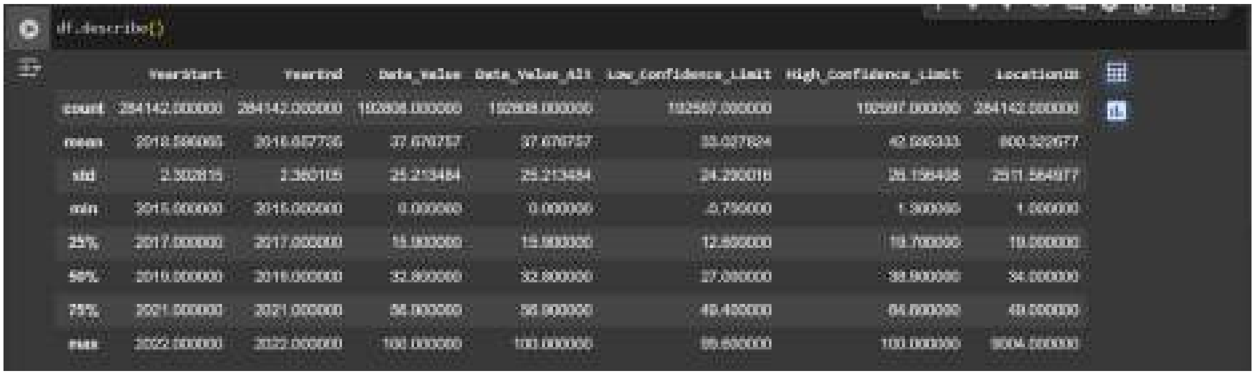
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284142 entries, 0 to 284141
Data columns (total 31 columns):
 #   Column                     Non-Null Count   Dtype
---  ------                     --------------   -----
 0   RowId                      284142 non-null  object
 1   YearStart                  284142 non-null  int64
 2   YearEnd                    284142 non-null  int64
 3   LocationAbbr               284142 non-null  object
 4   LocationDesc               284142 non-null  object
 5   Datasource                 284142 non-null  object
 6   Class                      284142 non-null  object
 7   Topic                      284142 non-null  object
 8   Question                   284142 non-null  object
 9   Data_Value_Unit            284142 non-null  object
 10  DataValueTypeID            284142 non-null  object
 11  Data_Value_Type            284142 non-null  object
 12  Data_Value                 192808 non-null  float64
 13  Data_Value_Alt             192808 non-null  float64
 14  Data_Value_Footnote_Symbol 109976 non-null  object
```



```
 15  Data_Value_Footnote        109976 non-null  object
 16  Low_Confidence_Limit       192597 non-null  float64
 17  High_Confidence_Limit      192597 non-null  float64
 18  StratificationCategory1    284142 non-null  object
 19  Stratification1            284142 non-null  object
 20  StratificationCategory2    247269 non-null  object
 21  Stratification2            247269 non-null  object
 22  Geolocation                253653 non-null  object
 23  ClassID                    284142 non-null  object
 24  TopicID                    284142 non-null  object
 25  QuestionID                 284142 non-null  object
 26  LocationID                 284142 non-null  int64
 27  StratificationCategoryID1  284142 non-null  object
 28  StratificationID1          284142 non-null  object
 29  StratificationCategoryID2  284142 non-null  object
 30  StratificationID2          284142 non-null  object
dtypes: float64(4), int64(3), object(24)
memory usage: 67.2+ MB
```

Command 3: df.describe()

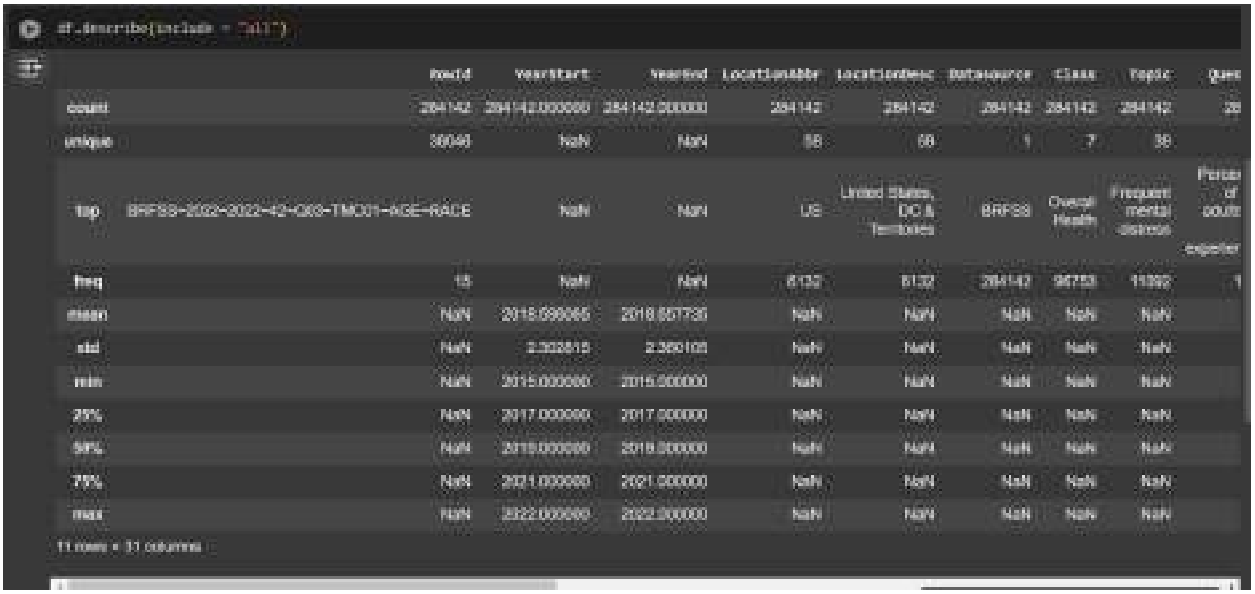
This command gives the details of all the values under all the features of the dataset. The command having no parameters gives information about count, max, min, standard deviation, top 25%ile, 50%ile, 75%ile and max value of the dataset.



```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID
count	284142.000000	284142.000000	192808.000000	192808.000000	192597.000000	192597.000000	284142.000000
mean	2018.596065	2018.657735	37.676757	37.676757	33.027624	42.696333	800.322677
std	2.302815	2.360105	25.213484	25.213484	24.290016	26.196436	2511.564877
min	2015.000000	2015.000000	0.000000	0.000000	-0.790000	1.300000	1.000000
25%	2017.000000	2017.000000	15.900000	15.900000	12.600000	19.700000	19.000000
50%	2019.000000	2019.000000	32.800000	32.800000	27.000000	38.900000	34.000000
75%	2021.000000	2021.000000	56.900000	56.900000	49.400000	64.600000	49.000000
max	2022.000000	2022.000000	100.000000	100.000000	99.600000	100.000000	9004.000000

If the parameter of include=”all” is included { df.describe(include=“all”)}, this includes even the non numeric values and gives some more information on fields such as count of unique values, top value, etc.



```
df.describe(include = "all")
```

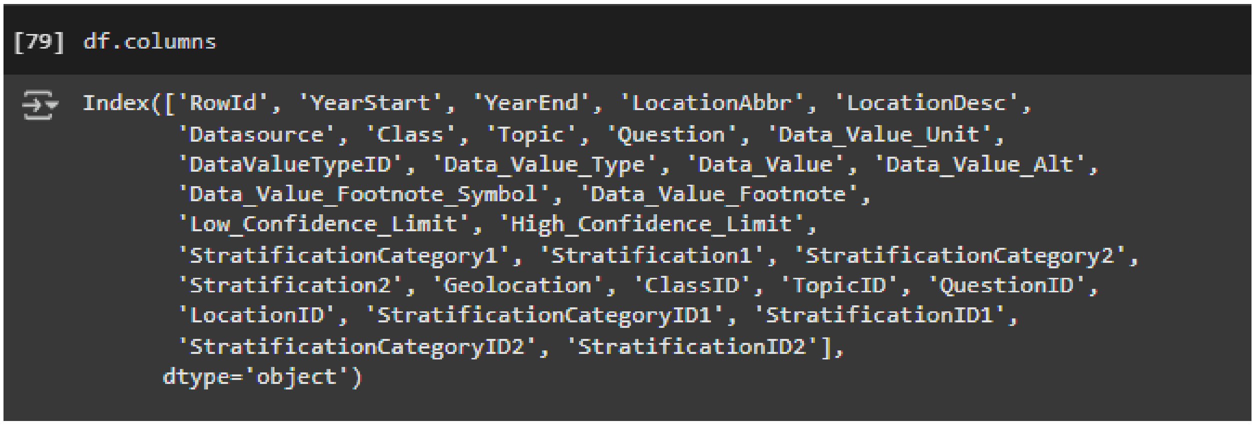
	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Ques
count	284142	284142.000000	284142.000000	284142	284142	284142	284142	284142	28
unique	38046	NaN	NaN	58	68	1	7	39	
top	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	NaN	NaN	US	United States, DC & Territories	BRFSS	Overall Health	Frequent mental distress	Perce of adult experien
freq	15	NaN	NaN	6132	6132	284142	96753	11092	1
mean	NaN	2018.598065	2018.657730	NaN	NaN	NaN	NaN	NaN	
std	NaN	2.302815	2.360105	NaN	NaN	NaN	NaN	NaN	
min	NaN	2015.000000	2015.000000	NaN	NaN	NaN	NaN	NaN	
25%	NaN	2017.000000	2017.000000	NaN	NaN	NaN	NaN	NaN	
50%	NaN	2019.000000	2019.000000	NaN	NaN	NaN	NaN	NaN	
75%	NaN	2021.000000	2021.000000	NaN	NaN	NaN	NaN	NaN	
max	NaN	2022.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	

11 rows × 31 columns

3) Drop columns that are not useful

In data science, it is important to drop the columns that would not help the user while working on the dataset as it would make it cleaner to work with.

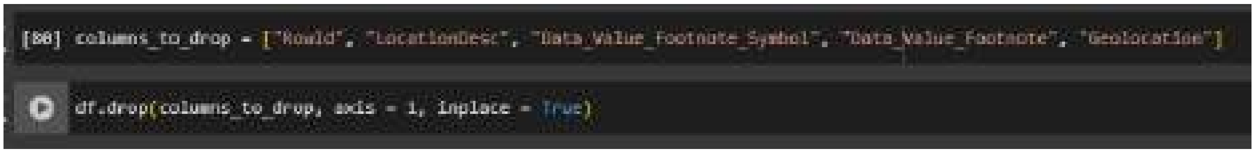
Here, we will first check the columns that are present using **df.columns** command



```
[79] df.columns

Index(['RowId', 'YearStart', 'YearEnd', 'LocationAbbr', 'LocationDesc',
       'Datasource', 'Class', 'Topic', 'Question', 'Data_Value_Unit',
       'DataValueTypeID', 'Data_Value_Type', 'Data_Value', 'Data_Value_Alt',
       'Data_Value_Footnote_Symbol', 'Data_Value_Footnote',
       'Low_Confidence_Limit', 'High_Confidence_Limit',
       'StratificationCategory1', 'Stratification1', 'StratificationCategory2',
       'Stratification2', 'Geolocation', 'ClassID', 'TopicID', 'QuestionID',
       'LocationID', 'StratificationCategoryID1', 'StratificationID1',
       'StratificationCategoryID2', 'StratificationID2'],
      dtype='object')
```

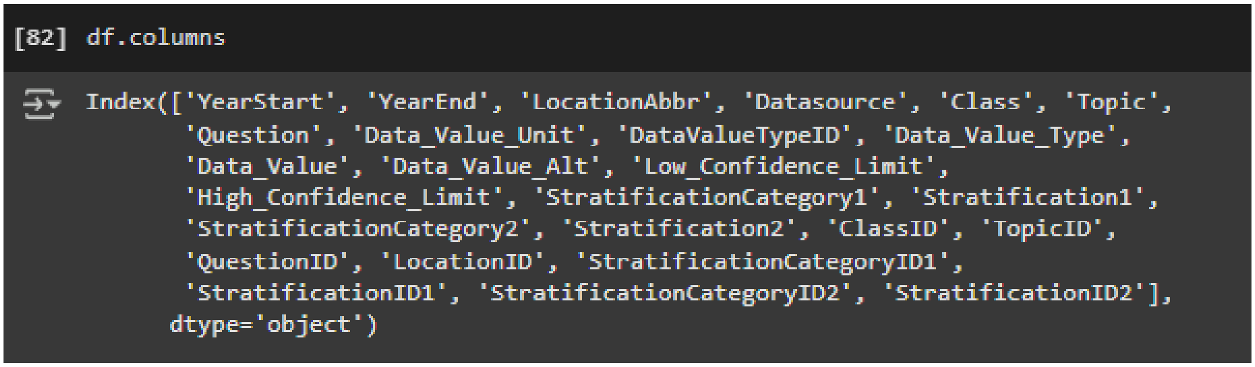
Now, we will list down the columns that are to be dropped and then pass it on to the command **df.drop(<column_names>, axis=1, inplace=True)**

Replace column names with either the list created previously, or with the column names itself. The inplace attribute takes care that the dataset will stay updated for the rest of the analysis.



```
[80] columns_to_drop = ["RowId", "LocationDesc", "Data_Value_Footnote_Symbol", "Data_Value_Footnote", "Geolocation"]

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

After running these commands, we run the **df.columns** command once again to check with the list of columns.



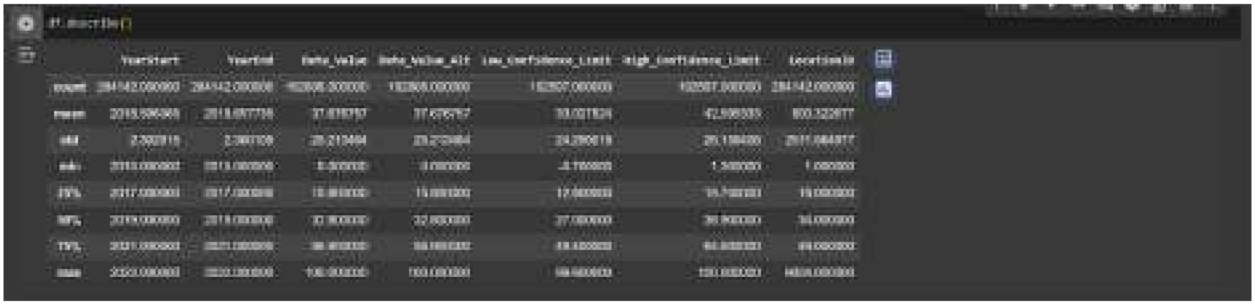
```
[82] df.columns

Index(['YearStart', 'YearEnd', 'LocationAbbr', 'Datasource', 'Class', 'Topic',
       'Question', 'Data_Value_Unit', 'DataValueTypeID', 'Data_Value_Type',
       'Data_Value', 'Data_Value_Alt', 'Low_Confidence_Limit',
       'High_Confidence_Limit', 'StratificationCategory1', 'Stratification1',
       'StratificationCategory2', 'Stratification2', 'ClassID', 'TopicID',
       'QuestionID', 'LocationID', 'StratificationCategoryID1',
       'StratificationID1', 'StratificationCategoryID2', 'StratificationID2'],
      dtype='object')
```

As observed here, the columns of RowId, LocationDesc, Data_Value_Footnote_Symbol, Data_Value_Footnote and Geolocation have been dropped.

4) Drop rows with maximum missing columns

It is important to drop the rows with maximum missing values as they would hinder the performance of the analysis and can lead to inaccuracies in the dataset. To perform this, follow these steps:

df.describe(): This command will give the count of rows present in the dataset.



```
df.describe()
```

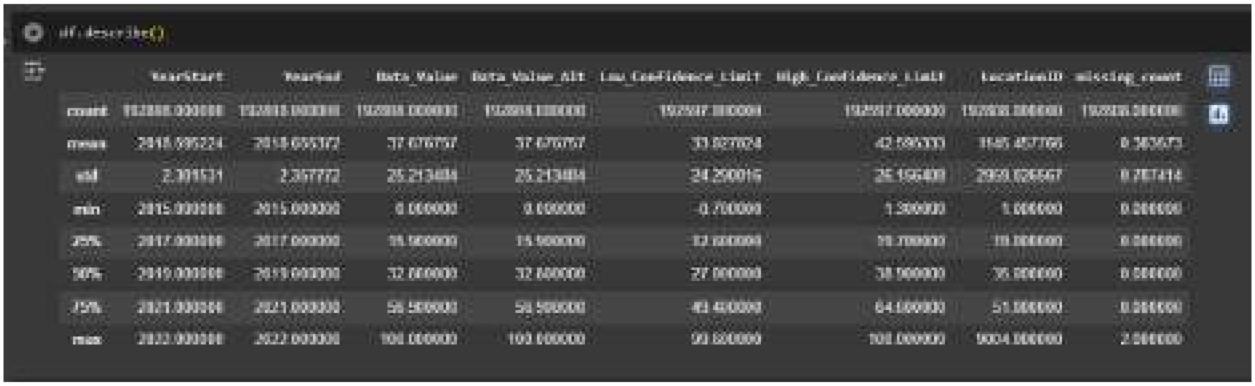
	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID
count	284142.000000	284142.000000	102608.000000	102608.000000	102507.000000	102507.000000	284142.000000
mean	2018.[illegible]	2018.[illegible]	37.[illegible]	37.[illegible]	33.[illegible]	42.[illegible]	800.[illegible]
std	2.[illegible]	2.[illegible]	20.[illegible]	20.[illegible]	24.[illegible]	20.[illegible]	2511.[illegible]
min	2013.000000	2013.000000	0.000000	0.000000	[illegible]	1.300000	1.000000
25%	2017.000000	2017.000000	[illegible]	[illegible]	[illegible]	[illegible]	19.000000
50%	2019.000000	2019.000000	32.800000	32.800000	27.000000	[illegible]	[illegible]
75%	2021.000000	2021.000000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
max	2022.000000	2022.000000	100.000000	100.000000	[illegible]	100.000000	[illegible]

df["missing_count"] = df.isnull().sum(axis=1)
max_missing = df["missing_count"].max()

Here the maximum missing count is 6. So to clean up some of the data, we will remove the rows with 4 or more missing values.

df = df[df["missing_count"] < 4]

The above set of commands do the following function:
i) Create a column called missing_count where the sum of all the cells having null values is stored.
ii) The maximum value from this missing_count column is considered for deletion
iii) Finally, we update the dataset by keeping the rows which have missing values less than a particular value.



```
df.describe()
```

	YearStart	YearEnd	Data_Value	Data_Value_Alt	Low_Confidence_Limit	High_Confidence_Limit	LocationID	missing_count
count	192808.000000	192808.000000	192808.000000	192808.000000	192597.000000	192597.000000	192808.000000	192808.000000
mean	2018.595224	2018.655372	37.676757	37.676757	33.827824	42.596333	1645.457766	0.383673
std	2.301531	2.357772	25.213484	25.213484	24.290016	25.156400	2969.026667	0.787414
min	2015.000000	2015.000000	0.000000	0.000000	-0.700000	1.300000	1.000000	0.000000
25%	2017.000000	2017.000000	15.900000	15.900000	12.600000	19.700000	19.000000	0.000000
50%	2019.000000	2019.000000	32.000000	32.000000	27.000000	38.900000	35.000000	0.000000
75%	2021.000000	2021.000000	56.500000	56.500000	49.400000	64.600000	51.000000	0.000000
max	2022.000000	2022.000000	100.000000	100.000000	99.600000	100.000000	9004.000000	2.000000

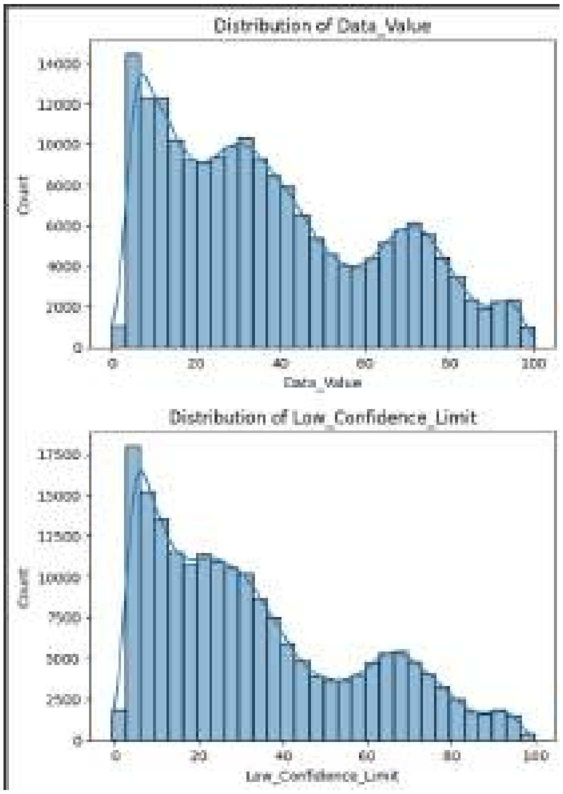
After running these sets of commands, we run the command **df.describe()** once again. Using this, we can see that the number of rows dropped from 284142 to 192808. (~32.14%)

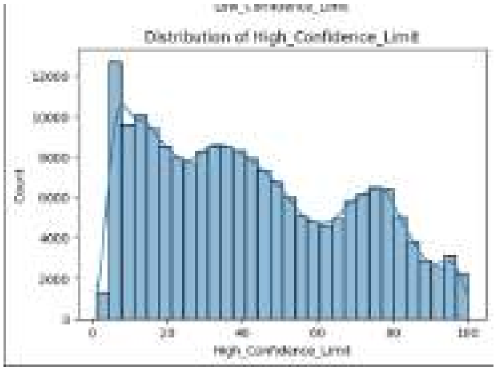
5) Take care of missing data

To take care of the missing data that has not been removed, one of the 2 methods can be used:
→ If the feature is of a numeric data type, we can use either mean, median or mode of the feature. If the data is normally distributed, use mean, if it is skewed, use median, and if many values are repeated, use mode.
→ If the feature contains different categories, there are 2 ways. Either fill it with the mode of the column, or add a custom value such as “Data Unavailable”.
Here, we would be filling the missing data for columns of Data_Value, Low_Confidence_Limit and High_Confidence_Limit
To check how to fill the missing data, follow the steps below.

i) **Check for skewness**

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

num_cols = ["Data_Value", "Low_Confidence_Limit", "High_Confidence_Limit"]

for col in num_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f"Distribution of {col}")
    plt.show()
```





As we can see here, there is a skewness to the left of the graph for each parameter, which means the data is not evenly distributed. Hence we use median.

Commands:

```
df[“Data_Value”].fillna(df[“Data_Value”].median(), inplace=True)
df[“Low_Confidence_Limit”].fillna(df[“Low_Confidence_Limit”].median(), inplace=True)
df[“High_Confidence_Limit”].fillna(df[“High_Confidence_Limit”].median(), inplace=True)
```

For columns StratificationCategory2 and Stratification2, as sufficient data is not available, we would fill the missing values with a placeholder “Data Unavailable”

Commands:

```
df["StratificationCategory2"].fillna("Data Unavailable", inplace=True)
```

```
df["Stratification2"].fillna("Data Unavailable", inplace=True)
```

Now, we check the values by using the **df.head()** command.

6) Create Dummy Variables

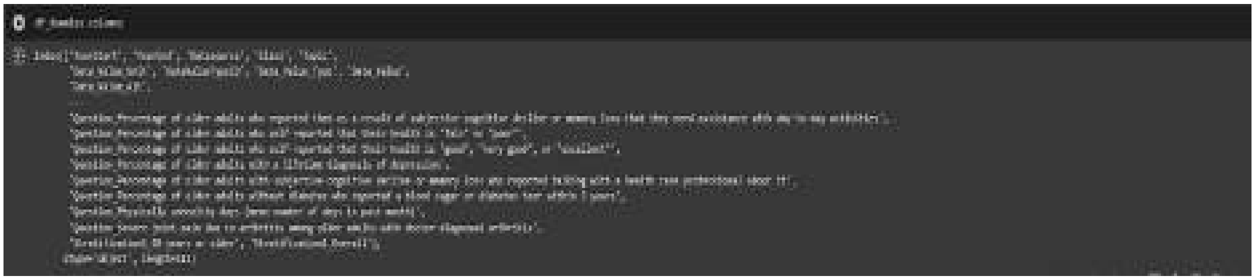
It is essential to create dummy variables to the columns that contain categorical data as most of the algorithms cannot understand the data directly. So they are classified as True and False or 0 and 1 which makes it easier.

To create the dummy variables, we will list the columns that fall under categorical columns and then create another variable as pd_dummies to get the output of this. Pandas library provides a inbuilt function called as get_dummies which takes the data from the columns and create all the required dummy variables

Command:

```
categorial_columns = ["LocationAbbr", "Question", "StratificationCategory1", "Stratification1"]
df_dummies = pd.get_dummies(df, columns=categorial_columns, drop_first=True)
```

To check these new columns, we use the command **df_dummies.columns.** As we see in the output, each Question is being treated as a new column.



```
df.columns

Index(['YearStart', 'YearEnd', 'LocationAbbr', 'Class', 'Topic',
       'Data_Value_Unit', 'DataValueTypeID', 'Data_Value_Type', 'Data_Value',
       'Data_Value_Alt',
       ...
       'Question_Percentage of older adults who reported that as a result of subjective cognitive decline or memory loss that they need assistance with day-to-day activities',
       'Question_Percentage of older adults who self-reported that their health is "fair" or "poor"',
       'Question_Percentage of older adults who self-reported that their health is "good", "very good", or "excellent"',
       'Question_Percentage of older adults with a lifetime diagnosis of depression',
       'Question_Percentage of older adults with subjective cognitive decline or memory loss who reported talking with a health care professional about it',
       'Question_Percentage of older adults without diabetes who reported a blood sugar or diabetes test within 3 years',
       'Question_Physically unhealthy days (mean number of days in past month)',
       'Question_Severe joint pain due to arthritis among older adults with doctor-diagnosed arthritis',
       'Stratification1_65 years or older', 'Stratification1_Overall'],
      dtype='object', length=[illegible])
```

7) Find out Outliers

Outliers are those data values that vary vastly from the other dataset values. It is important to detect these values as they affect the analysis result.

To find the outliers, there are 2 ways:

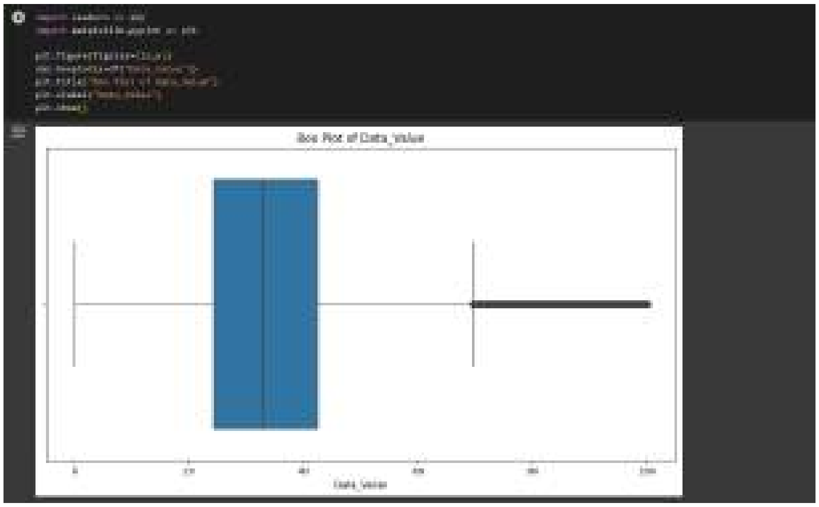
Method 1: Box-Plot

In this method, we use the column values to plot a box-plot graph. The values are usually in a box having lower and higher limit. If any outliers present, the come out of the box of the graph.

Command:

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
sns.boxplot(x=df["Data_Value"])
plt.title("Box Plot of Data_Value")
plt.xlabel("Data_Value")
plt.show()



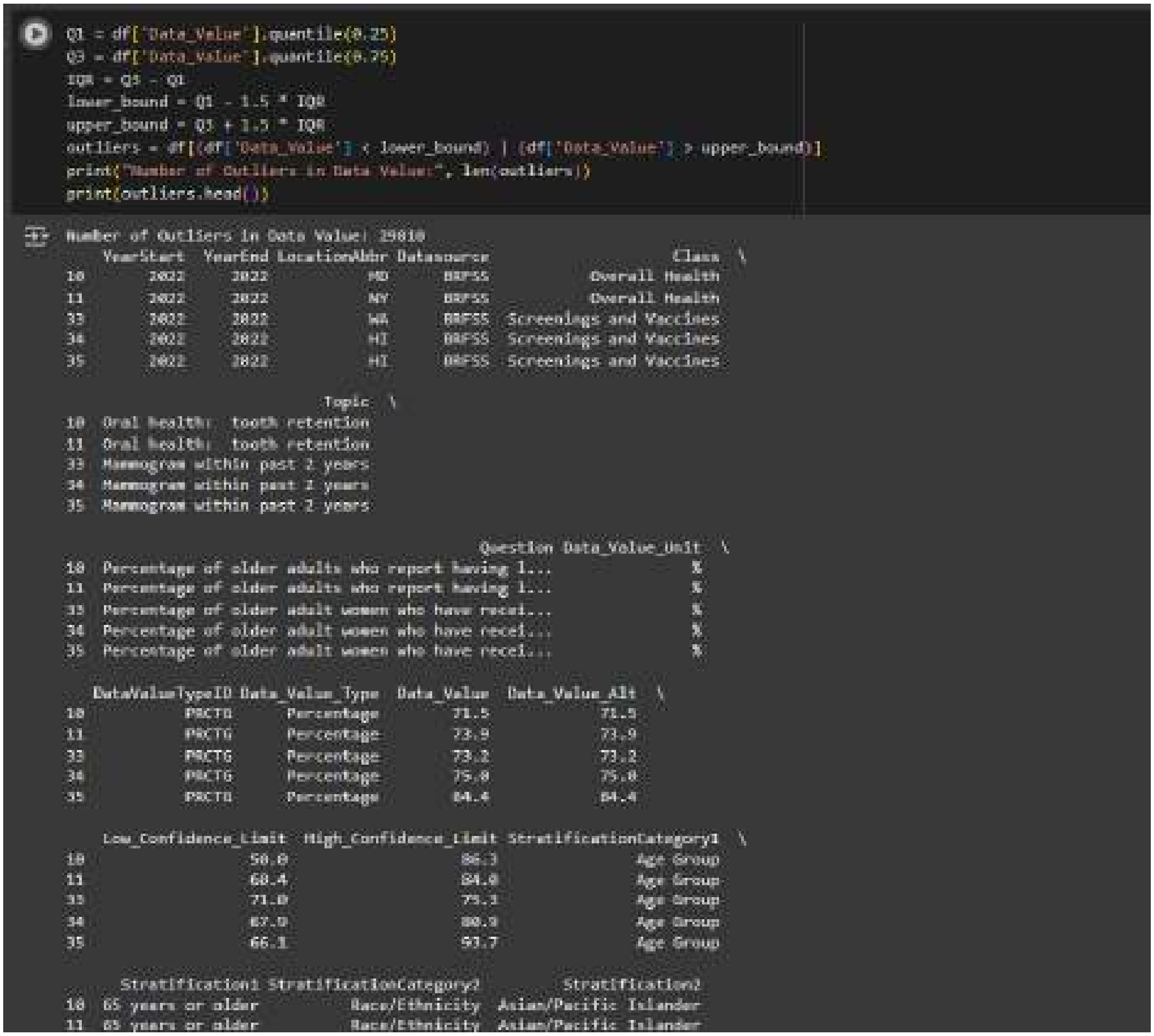
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
sns.boxplot(x=df["Data_Value"])
plt.title("Box Plot of Data_Value")
plt.xlabel("Data_Value")
plt.show()
```

Method 2: Using IQR Value.

In this method, we find the IQR value fo the column; which is the difference between Q1 - 1.5 * IQR and Q3 + 1.5 * IQR. This is a standard that is followed, the factor 1.5 can be modified between 1 to 3 based on the requirement.

Command:

```
Q1 = df['Data_Value'].quantile(0.25)
Q3 = df['Data_Value'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
print("Number of Outliers in Data Value:", len(outliers))
print(outliers.head())
```



```
Q1 = df['Data_Value'].quantile(0.25)
Q3 = df['Data_Value'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
print("Number of Outliers in Data Value:", len(outliers))
print(outliers.head())
```

```
Number of Outliers in Data Value: 29810
    YearStart  YearEnd LocationAbbr Datasource                   Class  \
10       2022     2022           MD      BRFSS          Overall Health
11       2022     2022           NY      BRFSS          Overall Health
33       2022     2022           WA      BRFSS  Screenings and Vaccines
34       2022     2022           HI      BRFSS  Screenings and Vaccines
35       2022     2022           HI      BRFSS  Screenings and Vaccines

                           Topic  \
10  Oral health:  tooth retention
11  Oral health:  tooth retention
33  Mammogram within past 2 years
34  Mammogram within past 2 years
35  Mammogram within past 2 years

                                             Question Data_Value_Unit  \
10  Percentage of older adults who report having 1...               %
11  Percentage of older adults who report having 1...               %
33  Percentage of older adult women who have recei...               %
34  Percentage of older adult women who have recei...               %
35  Percentage of older adult women who have recei...               %

   DataValueTypeID Data_Value_Type  Data_Value  Data_Value_Alt  \
10           PRCTG      Percentage        71.5            71.5
11           PRCTG      Percentage        73.9            73.9
33           PRCTG      Percentage        73.2            73.2
34           PRCTG      Percentage        75.0            75.0
35           PRCTG      Percentage        84.4            84.4

    Low_Confidence_Limit  High_Confidence_Limit StratificationCategory1  \
10                  50.0                   86.3               Age Group
11                  60.4                   84.0               Age Group
33                  71.0                   75.1               Age Group
34                  67.0                   80.3               Age Group
35                  66.1                   93.7               Age Group

      Stratification1 StratificationCategory2         Stratification2
10  65 years or older          Race/Ethnicity  Asian/Pacific Islander
11  65 years or older          Race/Ethnicity  Asian/Pacific Islander
```

From both the outputs, we get to know that there are some outliers present in the dataset. We can analyse the dataset manually to get the outliers, or use IQR score which gives us how many outliers are present based on our conditions.

8) Standardization and Normalization of columns

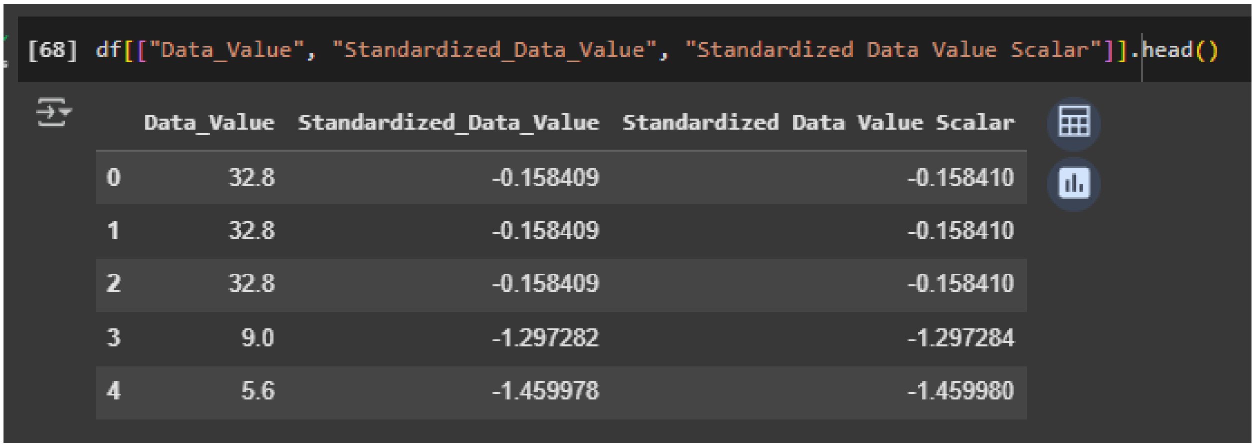
We can standardize and normalize columns using 1 of 2 methods. Either by their formulae, or by the SKLearn Library.

Standardize Column:
Using formula:

mean_value = df["Data_Value"].mean()
std_value = df["Data_Value"].std()
df["Standardized_Data_Value"] = (df["Data_Value"] - mean_value) / std_value

Using Library:

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Standardized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])



```
[68] df[["Data_Value", "Standardized_Data_Value", "Standardized Data Value Scalar"]].head()
```

	Data_Value	Standardized_Data_Value	Standardized Data Value Scalar
0	32.8	-0.158409	-0.158410
1	32.8	-0.158409	-0.158410
2	32.8	-0.158409	-0.158410
3	9.0	-1.297282	-1.297284
4	5.6	-1.459978	-1.459980

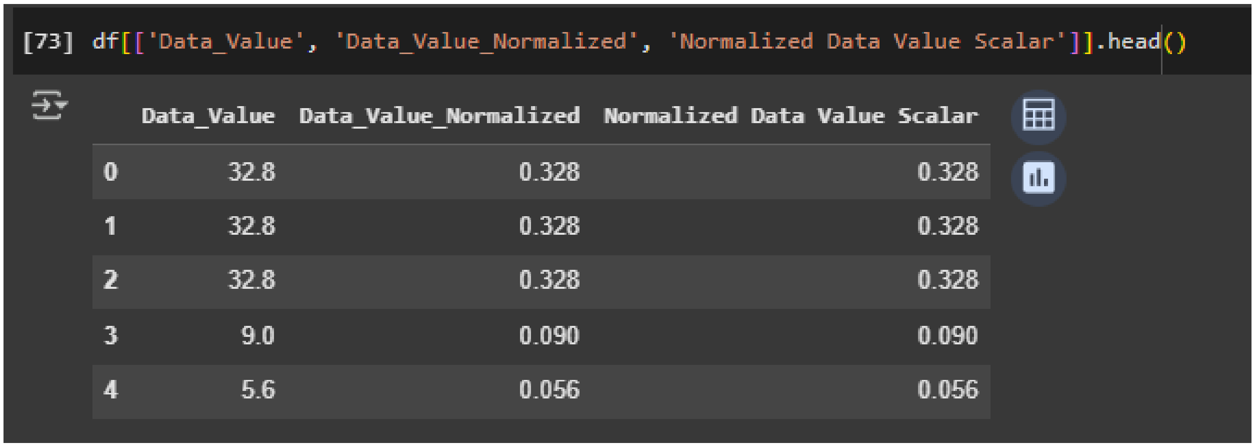
Normalize column:
Method 1: Formula

min_val = df['Data_Value'].min()
max_val = df['Data_Value'].max()

df['Data_Value_Normalized'] = (df['Data_Value'] - min_val) / (max_val - min_val)

Method 2: Scaler library

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Normalized Data Value Scalar'] = scaler.fit_transform(df[['Data_Value']])



```
[73] df[['Data_Value', 'Data_Value_Normalized', 'Normalized Data Value Scalar']].head()
```

```
   Data_Value  Data_Value_Normalized  Normalized Data Value Scalar
0        32.8                  0.328                         0.328
1        32.8                  0.328                         0.328
2        32.8                  0.328                         0.328
3         9.0                  0.090                         0.090
4         5.6                  0.056                         0.056
```

Conclusion:

Thus we have performed pre-processing on the dataset of Alzhiemers diseases and Healthy Aging data.

To load the data into pandas, we used the read_csv() function of the pandas library to load it. To verify this, we used the head() function to show the 1st 5 entries of the dataset.

For a description, we used various methods such as head(), info(), describe() which gave information such as data types, mean, max, min, count, etc.

Using drop() command, we dropped the columns off the dataset that would not have had much impact on the analysis. In this dataset, we dropped columns such as RowId, LocationDesc, Data_Value_Footnote_Symbol, Data_Value_Footnote and Geolocation.

For dropping rows with maximum missing values, we implemented a series of commands on our dataset that checked each entry for missing values, selected the max from amongst them and then deleted those rows with maximum missing values. This is done so that it does not bring up the skewness of the dataset. In our dataset, the data is reduced from 284142 to 192808. (~32.14%)

Now, to take care of the missing data, we analysed the data which is available by taking graphs of it, and used the apt method (mean, median, mode) to fill up the missing values of the database.

Columns such as Questions, LoactionAbbr would have caused an error while performing analysis on the dataset. To reduce this, such columns are being converted to dummy variables where their entries are 0 or 1.

To find the outliers, we first plotted the boxplot. From this graph, we got to know about the outliers present outside the box graph. After manual analysis, we decided to use the IQR index technique to check out the outliers of the dataset.

Now, while analysis, data with higher values can tend to affect the analysis. To reduce this anomaly, we normalize and standardize the graph based on minmax/standard deviation methods to get the values to a reasonable range for the analysis to take place smoothly.

Sairam Konar D15C 27 2024-2025

Experiment 2

Aim: Perform following data visualization and exploration on your selected dataset.

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

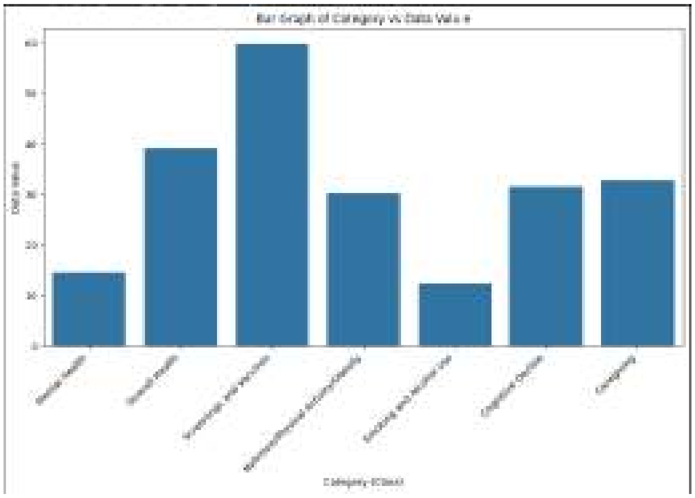
Steps:

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

Seaborn and Matplotlib are used together to create an effective bar graph. Matplotlib sets the figure size, labels, and titles to ensure clarity in the visualization. Seaborn's barplot function is used to plot categorical data, where the x-axis represents different categories (Class), and the y-axis represents the corresponding data values. The estimator='mean' argument allows automatic aggregation, providing the average value for each category. This helps in identifying trends and comparisons across different classes efficiently. The rotation of x-axis labels enhances readability.

Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12,6))
sns.barplot(x=df["Class"], y=df["Data_Value"], estimator='mean', ci=None)
plt.xticks(rotation=45, ha='right')
plt.xlabel("Category (Class)")
plt.ylabel("Data Value")
plt.title("Bar Graph of Category vs Data Valu e")
plt.show()
```

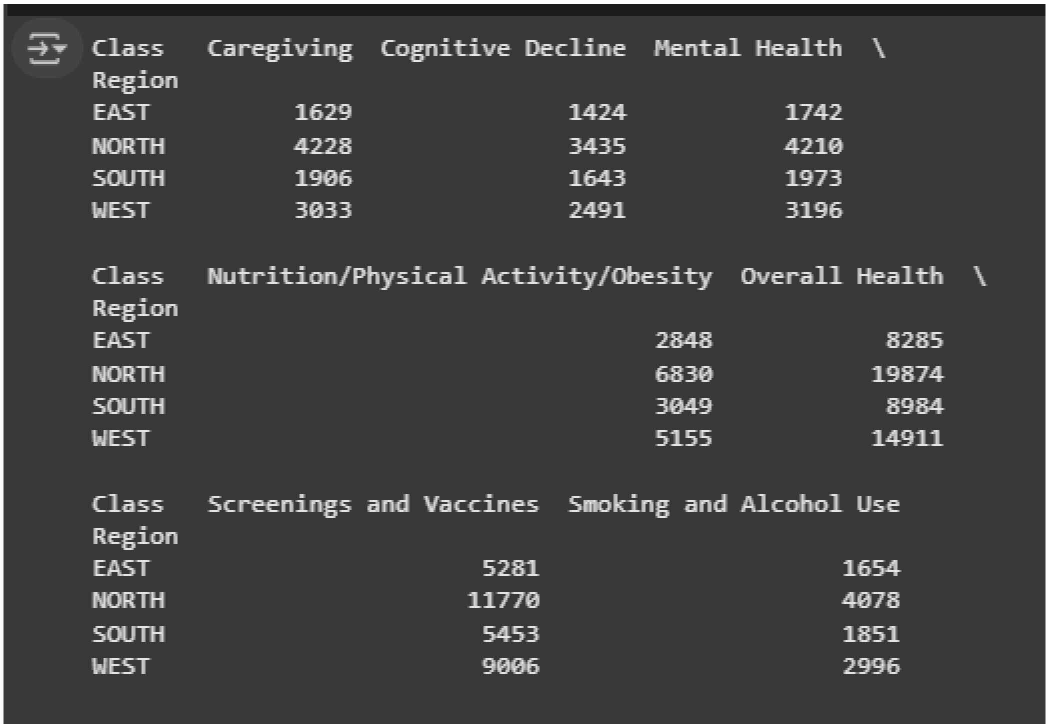


The bar graph compares the average Data Value across categories, highlighting dominant classes. Taller bars, such as Overall Health and Screenings & Vaccines, indicate higher averages, while categories like Mental Health & Smoking and Alcohol Use show more variability. It effectively reveals categorical trends and relative impact in the dataset.

The contingency table is visualized using Seaborn’s heatmap function, which highlights patterns in data distribution with color gradients. The table represents the relationship between two categorical variables, Region and Class. The color intensity helps in quickly identifying regions with higher or lower values for different categories. Matplotlib is used to define the figure settings and ensure the visualization is clear. This approach allows for an intuitive understanding of data distribution, making it easier to identify trends and variations between different regions and classes.

Code:

```
import pandas as pd
contingency_table = pd.crosstab(df['Region'], df['Class'])
print(contingency_table)
```



```
Class   Caregiving  Cognitive Decline  Mental Health  \
Region
EAST          1629               1424           1742
NORTH         4228               3435           4210
SOUTH         1906               1643           1973
WEST          3033               2491           3196

Class   Nutrition/Physical Activity/Obesity  Overall Health  \
Region
EAST                                   2848            8285
NORTH                                  6830           19874
SOUTH                                  3049            8984
WEST                                   5155           14911

Class   Screenings and Vaccines  Smoking and Alcohol Use
Region
EAST                       5281                     1654
NORTH                     11770                     4078
SOUTH                      5453                     1851
WEST                       9006                     2996
```

The contingency table displays category frequencies across regions, helping identify regional variations. The heatmap enhances interpretation using color gradients, where darker shades indicate higher counts. This visualization simplifies pattern detection, revealing regional dominance in certain categories and making data comparison easier by highlighting imbalances and trends across different regions.

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

In this part, we have plotted Heatmap and seaborn as they are relevant to our dataset.

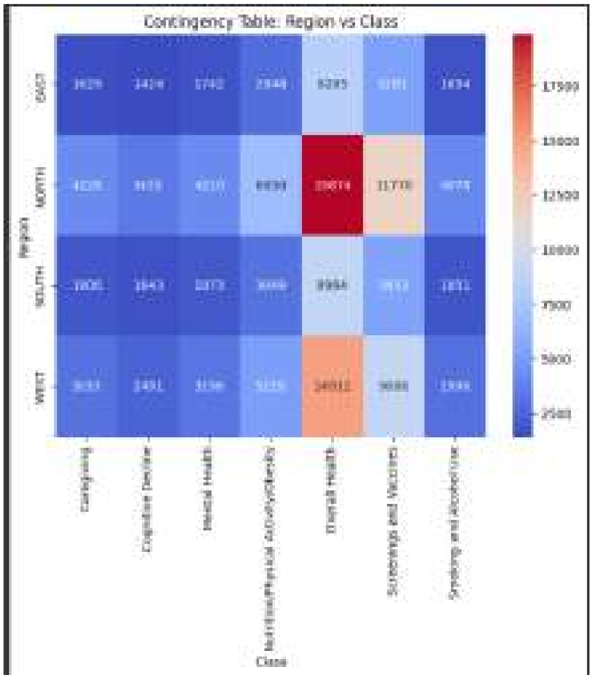
A heatmap is used to visualize the contingency table, which represents the relationship between two categorical variables, Region and Class. Seaborn's heatmap function is applied to display data distribution using a color gradient (cmap="coolwarm"), where darker shades indicate higher values. The annot=True argument ensures that numerical values are displayed in each cell for clarity. Matplotlib is used to set figure size and labels, enhancing readability. This visualization helps in quickly identifying trends and correlations between categories.

Code:

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

plt.figure(figsize=(8,6))
sns.heatmap(contingency_table, annot=True, cmap="coolwarm", fmt='d')
```

```
plt.title("Contingency Table: Region vs Class")
plt.xlabel("Class")
plt.ylabel("Region")
plt.show()
```

Output:

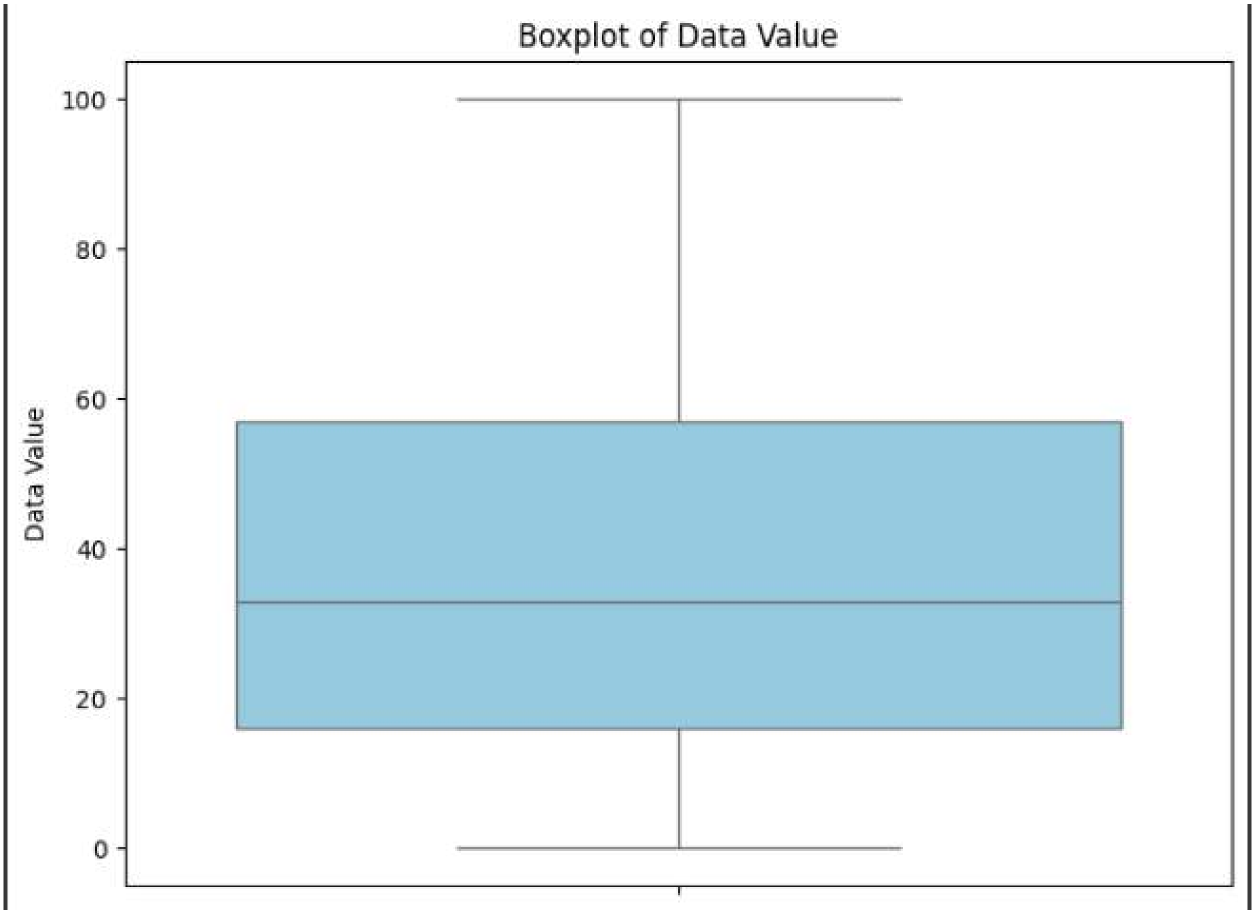


A box plot is used to summarize the distribution of the Data_Value column, helping to identify outliers, the median, and the interquartile range. Seaborn's boxplot function plots the Data_Value on the y-axis with a skyblue color for better visibility. The box represents the middle 50% of the data, while the whiskers indicate the overall spread. Matplotlib ensures proper figure formatting with clear labels and titles. This visualization is useful for detecting anomalies and understanding data spread efficiently.

Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,6))
sns.boxplot(y=df["Data_Value"], color="skyblue")
plt.ylabel("Data Value")
plt.title("Boxplot of Data Value")
plt.show()
```

Output:



The boxplot visually summarizes the distribution of Data Value, showing its spread and central tendency. The box represents the interquartile range (middle 50% of data), with the median slightly below center, indicating right skewness. The wider IQR and longer upper whisker suggest high variability and possible extreme values affecting the distribution.

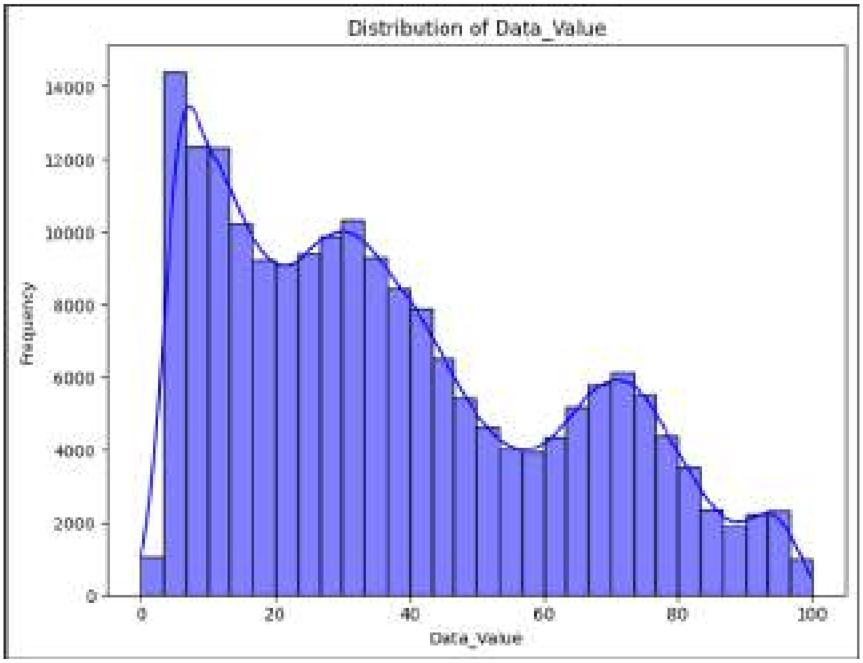
3) Create histogram and normalized Histogram

A histogram is used to visualize the distribution of Data_Value, showing how frequently different values occur. The sns.histplot function from Seaborn is used with bins=30 to divide data into 30 intervals, providing a detailed view of the distribution. The kde=True parameter adds a Kernel Density Estimate (KDE) curve for a smooth representation. The color is set to blue for better visibility. This helps in understanding data skewness, spread, and common value ranges.

Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))
sns.histplot(df["Data_Value"], bins=30, kde=True, color="blue")  # KDE for smooth curve
plt.title("Distribution of Data_Value")
plt.xlabel("Data_Value")
plt.ylabel("Frequency")
plt.show()
```

Output:

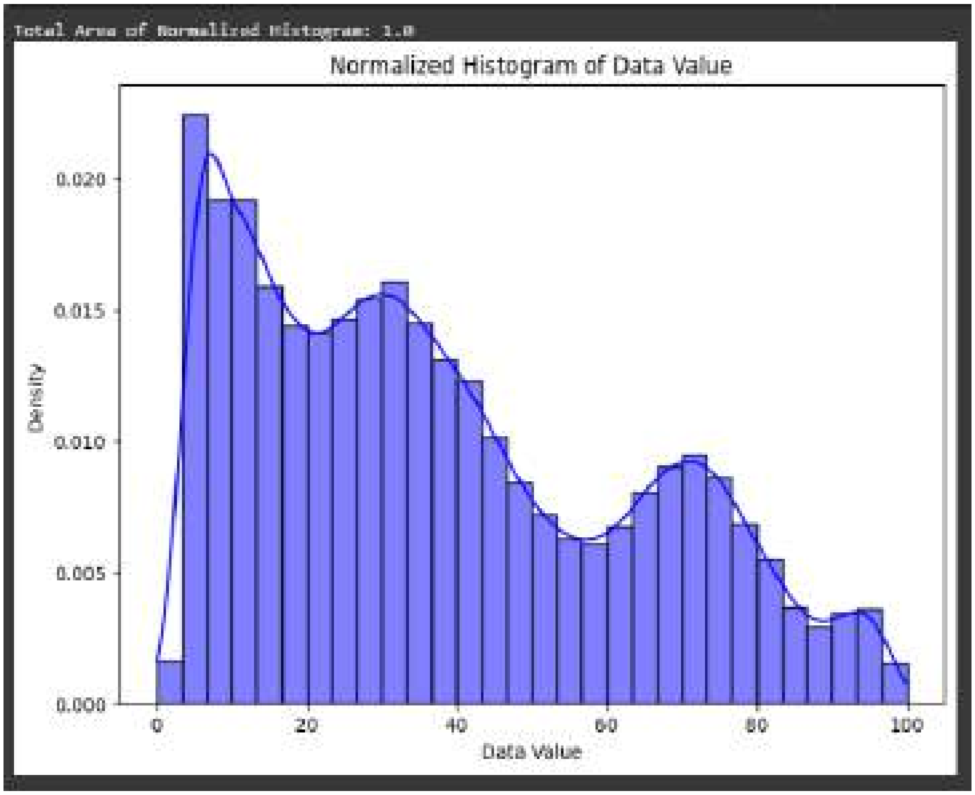


A normalized histogram represents data in terms of density instead of raw frequency, ensuring that the total area under the bars equals 1. The stat='density' parameter in sns.histplot normalizes the counts. The bin width is calculated, and the total area of the histogram is verified to confirm normalization. This type of histogram is useful when comparing distributions across different datasets or when absolute frequencies are not relevant.

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,6))
hist_data = sns.histplot(df["Data_Value"], bins=30, kde=True, stat='density', color="blue")
heights = [patch.get_height() for patch in hist_data.patches]
bin_width = hist_data.patches[1].get_x() - hist_data.patches[0].get_x()
area = sum(heights) * bin_width
print("Total Area of Normalized Histogram:", area)
plt.xlabel("Data Value")
plt.ylabel("Density")
plt.title("Normalized Histogram of Data Value")
plt.show()
```

Output:



```
Total Area of Normalized Histogram: 1.0
```

The histogram shows the frequency of Data_Value occurrences, while the normalized histogram represents probability density. Multiple peaks suggest a multimodal distribution, indicating distinct groupings. The KDE curve highlights trends, and skewness hints at extreme values. The normalized histogram aids comparison by removing sample size effects, enhancing distribution analysis.

4) **Describe what this graph and table indicates.**

- **Bar graph:**

The bar graph compares the average Data Value across categories, highlighting dominant classes. Taller bars, such as Overall Health and Screenings & Vaccines, indicate higher averages, while categories like Mental Health & Smoking and Alcohol Use show more variability. It effectively reveals categorical trends and relative impact in the dataset.

- **Contingency table and Heatmap**

The contingency table displays category frequencies across regions, helping identify regional variations. The heatmap enhances interpretation using color gradients, where darker shades indicate higher counts. This visualization simplifies pattern detection, revealing regional dominance in certain categories and making data comparison easier by highlighting imbalances and trends across different regions.

- **Box Plot**

The boxplot visually summarizes the distribution of Data Value, showing its spread and central tendency. The box represents the interquartile range (middle 50% of data), with the median slightly below center, indicating right skewness. The wider IQR and longer upper whisker suggest high variability and possible extreme values affecting the distribution.

- **Histogram and Normalised Histogram**

The histogram shows the frequency of Data_Value occurrences, while the normalized histogram represents probability density. Multiple peaks suggest a multimodal distribution, indicating distinct groupings. The KDE curve highlights trends, and skewness hints at extreme values. The normalized histogram aids comparison by removing sample size effects, enhancing distribution analysis.

5) **Handle outlier using box plot and Inter quartile range.**

Outliers in a dataset can significantly impact statistical analysis, making it essential to identify and handle them effectively. A boxplot is a useful visualization tool for detecting outliers, as it highlights the spread of data, median, and interquartile range (IQR). The IQR method is a robust technique for outlier detection, where values falling outside **Q1 - 1.5 * IQR** and **Q3 + 1.5 * IQR** are considered potential outliers. By computing Q1 (25th percentile) and Q3 (75th percentile), we determine the IQR and set boundaries to identify extreme values. Outliers can distort measures of central tendency and variability, potentially leading to misleading insights. Once detected, these values can be analyzed to determine if they result from data entry errors, anomalies, or natural variability. Proper handling, such as transformation, capping, or removal, depends on the dataset's context. The combination of a boxplot and IQR analysis ensures a balanced approach to managing outliers.

Code:

Box-plot:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,6))
sns.boxplot(y=df["Data_Value"], color="skyblue")
plt.ylabel("Data Value")
plt.title("Boxplot of Data Value")
plt.show()
```

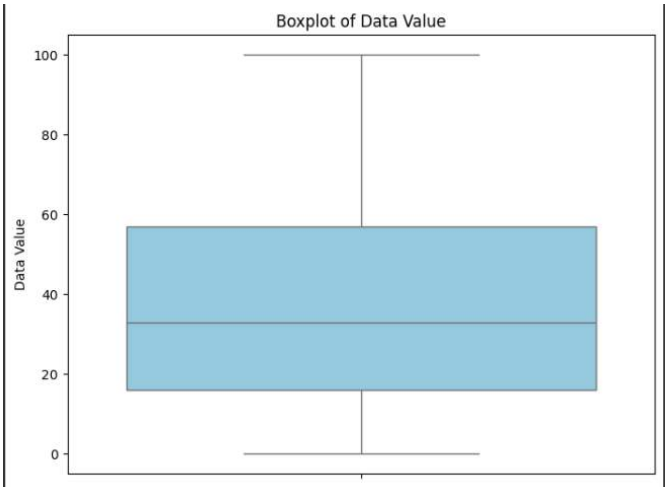
IQR:

```
Q1 = df['Data_Value'].quantile(0.25)
Q3 = df['Data_Value'].quantile(0.75)
```

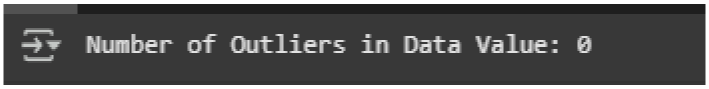
```
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Data_Value'] < lower_bound) | (df['Data_Value'] > upper_bound)]
print("Number of Outliers in Data Value:", len(outliers))
```

Outputs:

Box Plot:



IQR:



```
Number of Outliers in Data Value: 0
```

Conclusion: This experiment provided valuable insights into the dataset through various visualizations. The bar graph highlighted variations in Data_Value across different categories, helping identify dominant and underrepresented classes. The contingency table and heatmap effectively showcased regional differences, revealing areas with higher concentrations in specific categories.The histogram and KDE curve indicated a multimodal distribution, suggesting the presence of distinct subgroups within the data. The box plot and IQR method identified several outliers, with extreme values that could potentially skew statistical analysis. Detecting and managing these outliers is crucial to maintaining data accuracy and ensuring reliable conclusions. Overall, this study reinforced the importance of exploratory data analysis (EDA) in understanding data distribution, detecting patterns, and handling anomalies. These visualizations provide a clearer perspective on data trends, aiding in better decision-making and more precise interpretations of the dataset.

Sairam Konar D15C 27 2024-2025

Experiment 3

Aim: Perform Data Modeling.
Problem Statement:
a. Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.
b. Use a bar graph and other relevant graph to confirm your proportions.
c. Identify the total number of records in the training data set.
d. Validate partition by performing a two-sample Z-test.

Steps:

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

Code:

```
from sklearn.model_selection import train_test_split
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)
```

This function imports the train_test_split function from sklearn.model_selection library. This makes 2 dataframes, a train_df and test_df. Here, based on the test_size parameter, it would divide the dataset into that percent of values and insert it in the test_df dataframe. The remaining values are put in the train_df dataframe. Defining the random_state parameter helps the splitting to be consistent. The value of the parameter does not matter, only the condition being it should be consistent.

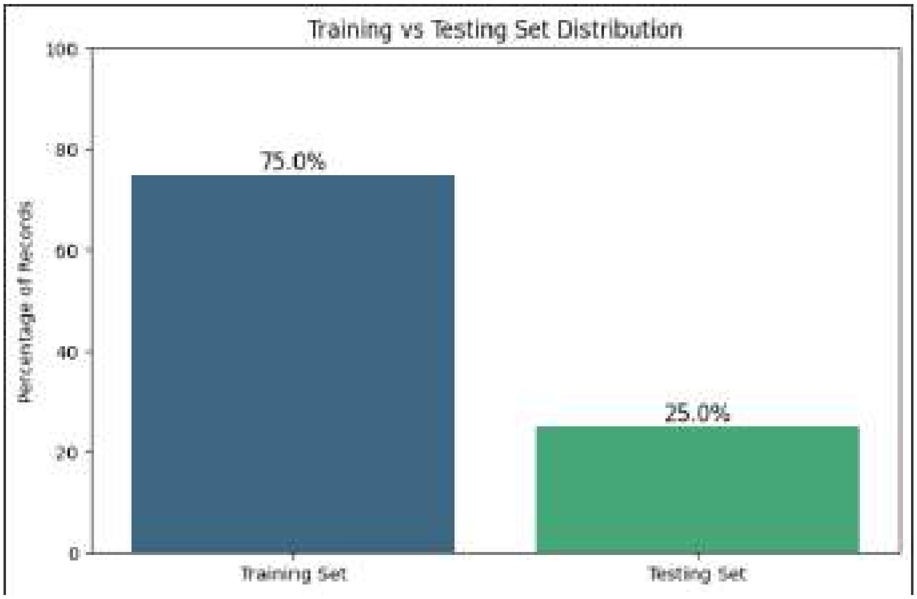
2) Use a bar graph and other relevant graphs to confirm your proportions.

Graphs help validate the correct division of data. Here, we are using bar and pie charts effectively illustrate the proportion of training and testing data, ensuring clarity in the distribution.

<u>Bar Graph:</u>
Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
total = len(df)
sizes = [len(train_df) / total * 100, len(test_df) / total * 100]
labels = ['Training Set', 'Testing Set']
plt.figure(figsize=(8, 5))
sns.barplot(x=labels, y=sizes, palette="viridis")
for i, v in enumerate(sizes):
    plt.text(i, v + 1, f"{v:.1f}%", ha='center', fontsize=12)
plt.ylabel("Percentage of Records")
plt.title("Training vs Testing Set Distribution")
plt.ylim(0, 100)  # Ensure y-axis goes from 0 to 100%
plt.show()
```

Output:

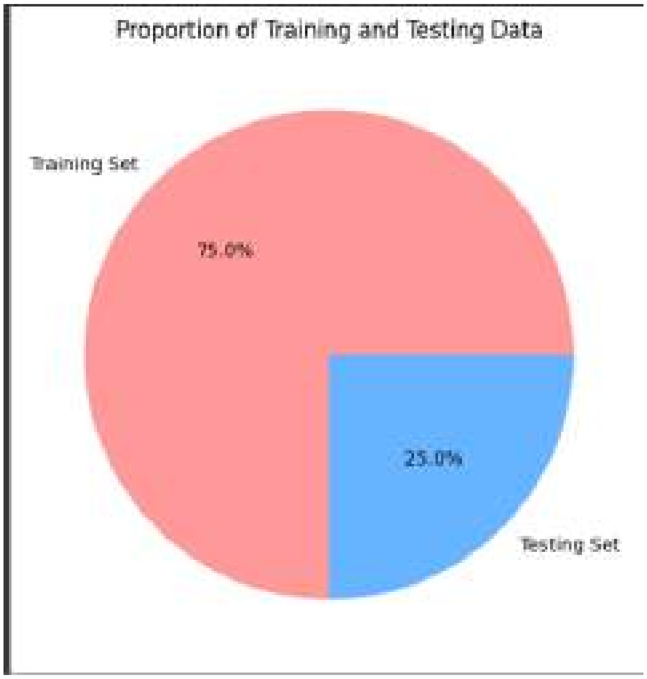


<u>Pie chart:</u>

Code:

```
plt.figure(figsize=(6,6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['#ff9999','#66b3ff'])
plt.title("Proportion of Training and Testing Data")
plt.show()
```

Output:

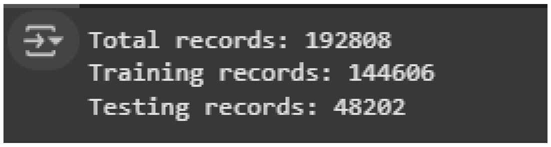


3) Identify the total number of records in the training data set.

Code:

```
print(f"Total records: {len(df)}")
print(f"Training records: {len(train_df)}")
print(f"Testing records: {len(test_df)}")
```

Output:



```
Total records: 192808
Training records: 144606
Testing records: 48202
```

4) Validate partition by performing a two-sample Z-test.

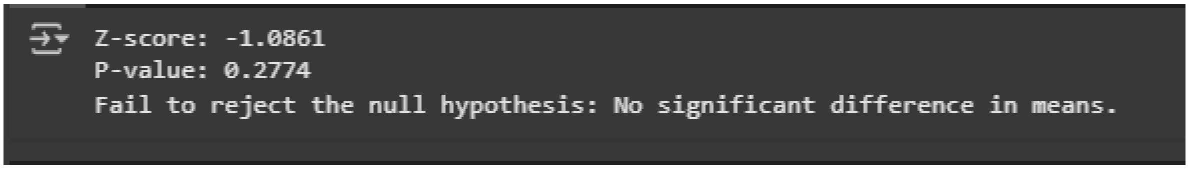
A two-sample Z-test evaluates whether the training and testing datasets share similar characteristics. By comparing their mean values, it ensures the data split is balanced and does not introduce bias.

Code:

```
import numpy as np
from scipy.stats import norm

train_values = train_df["Data_Value"]
test_values = test_df["Data_Value"]

mean_train = np.mean(train_values)
mean_test = np.mean(test_values)
std_train = np.std(train_values, ddof=1)
std_test = np.std(test_values, ddof=1)

n_train = len(train_values)
n_test = len(test_values)

z_score = (mean_train - mean_test) / np.sqrt((std_train**2 / n_train) + (std_test**2 / n_test))

p_value = 2 * (1 - norm.cdf(abs(z_score)))

print(f"Z-score: {z_score:.4f}")
print(f"P-value: {p_value:.4f}")

alpha = 0.05
```

```
if p_value < alpha:
    print("Reject the null hypothesis: The means are significantly different.")
else:
    print("Fail to reject the null hypothesis: No significant difference in means.")
```

Output:



```
Z-score: -1.0861
P-value: 0.2774
Fail to reject the null hypothesis: No significant difference in means.
```

As the p_calue came out to be greater than 0.05, we would accept that there is no difference in the means and hence, the splitting performed is indeed valid.

Conclusion: The data partitioning process successfully divided the dataset into 75% training data and 25% testing data. The distribution was validated using bar and pie charts, which visually confirmed the correct proportions of training and testing records. The bar chart effectively displayed the percentage of records in each subset, while the pie chart provided a clear representation of their proportions. The total number of records in the training set came out to be 1,44,606, which is 75% of the total records in the dataset. Further validation was performed using a two-sample Z-test, which compared the mean values of the training and testing datasets to ensure a balanced split without introducing bias. The calculated Z-score (-1.0861) and p-value (0.2774) indicated whether there was a significant difference between the two subsets. If the p-value was greater than the significance level (0.05), it confirmed that the means of the training and testing sets were statistically similar, ensuring a fair representation of the data. Overall, this experiment ensured that the dataset was properly divided, maintaining consistency in distribution while preserving statistical integrity for effective model training and evaluation.

Sairam Konar D15C 27 2024-2025

Experiment 4

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.
Perform the following Tests:Correlation Tests:
a) Pearson’s Correlation Coefficient
b) Spearman’s Rank Correlation
c) Kendall’s Rank Correlation
d) Chi-Squared Test

Dataset used: https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction

Steps:

1) Load the dataset using Pandas

```
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
file_path = "/content/sample_data/cleaned_combined.csv"
df = pd.read_csv(file_path)
```

2) Extract numeric columns from the dataset

```
df_numeric = df.copy()
for col in df_numeric.select_dtypes(include=['object']).columns:
    df_numeric[col] = df_numeric[col].astype('category').cat.codes
```

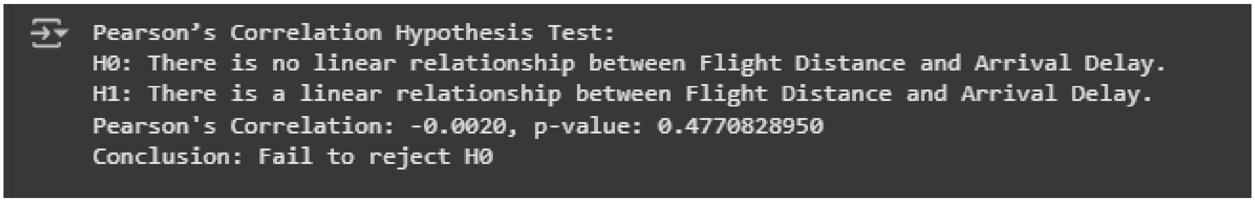
3) Perform Pearson Correlation.

The Pearson correlation coefficient (r) measures the linear relationship between two variables, ranging from -1 to 1. A value close to 1 indicates a strong positive correlation, -1 a strong negative correlation, and 0 no correlation. It is widely used in statistics for predictive analysis.

Code:

```
pearson_corr, pearson_p = stats.pearsonr(df_numeric['Flight Distance'], df_numeric['Arrival Delay in Minutes'])
print("Pearson’s Correlation Hypothesis Test:")
print("H0: There is no linear relationship between Flight Distance and Arrival Delay.")
print("H1: There is a linear relationship between Flight Distance and Arrival Delay.")
print(f"Pearson's Correlation: {pearson_corr:.4f}, p-value: {pearson_p:.10f}")
print("Conclusion:", "Fail to reject H0" if pearson_p > 0.05 else "Reject H0", "\n")
```

Output:



```
Pearson’s Correlation Hypothesis Test:
H0: There is no linear relationship between Flight Distance and Arrival Delay.
H1: There is a linear relationship between Flight Distance and Arrival Delay.
Pearson's Correlation: -0.0020, p-value: 0.4770828950
Conclusion: Fail to reject H0
```

In the scipy library, we have a library function called as scipy.stats which we will be using to find the correlation coefficients and p-score of the columns mentioned. Here, we are finding the Pearson’s Correlation Coefficient between the column Flight Distance and Arrival Delay in Minutes.

Flight Distance and Arrival Delay show almost no linear relationship, with a near-zero correlation (-0.0020) and an insignificant p-value (0.4771), indicating that changes in flight distance do not predict arrival delay. As p_value > 0.05, we are not able to reject the null hypothesis and hence, there is no linear relationship between Flight Distance and Arrival Delay

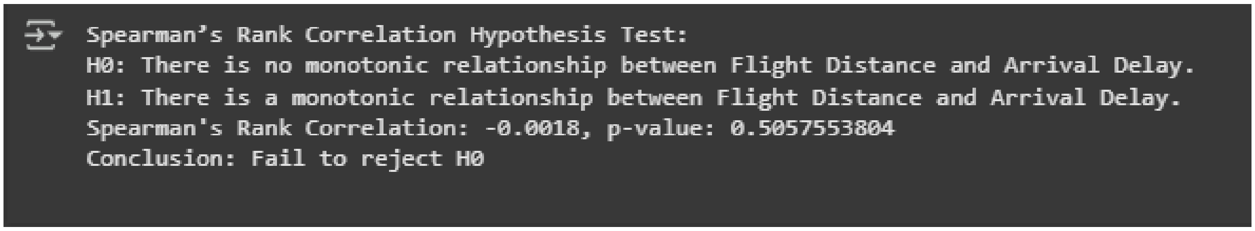
4) Perform Spearman’s Rank Correlation.

Spearman's rank correlation coefficient (ρ) measures the monotonic relationship between two variables, assessing how well their ranks correspond. It ranges from -1 to 1, where 1 indicates a perfect increasing relationship, -1 a perfect decreasing relationship, and 0 no correlation. It is useful for nonlinear and ordinal data.

Code:

```
spearman_corr, spearman_p = stats.spearmanr(df_numeric['Flight Distance'], df_numeric['Arrival Delay in Minutes'])
print("Spearman’s Rank Correlation Hypothesis Test:")
print("H0: There is no monotonic relationship between Flight Distance and Arrival Delay.")
print("H1: There is a monotonic relationship between Flight Distance and Arrival Delay.")
print(f"Spearman's Rank Correlation: {spearman_corr:.4f}, p-value: {spearman_p:.10f}")
print("Conclusion:", "Fail to reject H0" if spearman_p > 0.05 else "Reject H0", "\n")
```

Output:



```
Spearman’s Rank Correlation Hypothesis Test:
H0: There is no monotonic relationship between Flight Distance and Arrival Delay.
H1: There is a monotonic relationship between Flight Distance and Arrival Delay.
Spearman's Rank Correlation: -0.0018, p-value: 0.5057553804
Conclusion: Fail to reject H0
```

In the SciPy library, we use scipy.stats to compute the Spearman’s Rank Correlation between Flight Distance and Arrival Delay in Minutes. The correlation coefficient (-0.0018) and p-value (0.5058) suggest no meaningful monotonic relationship, meaning changes in flight distance do not consistently influence arrival delay. As p_value > 0.05, we fail to reject the null hypothesis and hence, no monotonic relationship is observed between Flight Distance and Arrival TIme.

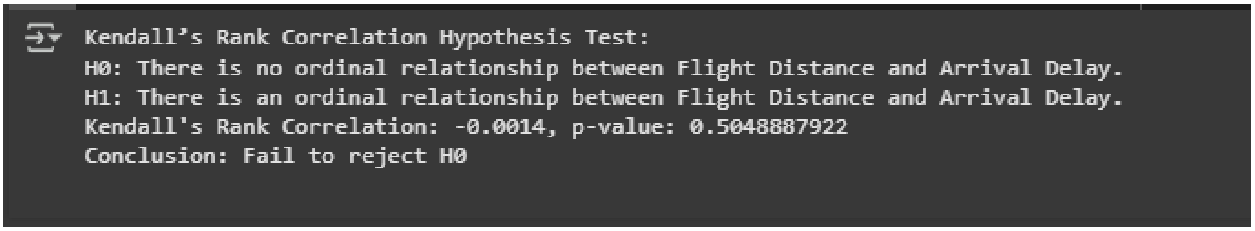
5) Perform Kendall’s Rank Correlation

Kendall’s rank correlation coefficient (τ) measures the ordinal association between two variables. It evaluates the consistency of rank ordering between them. Ranging from -1 to 1, τ = 1 indicates perfect agreement, -1 perfect disagreement, and 0 no correlation. It is robust for small datasets and tied ranks.

Code:

```
kendall_corr, kendall_p = stats.kendalltau(df_numeric['Flight Distance'], df_numeric['Arrival Delay in Minutes'])
print("Kendall’s Rank Correlation Hypothesis Test:")
print("H0: There is no ordinal relationship between Flight Distance and Arrival Delay.")
print("H1: There is an ordinal relationship between Flight Distance and Arrival Delay.")
print(f"Kendall's Rank Correlation: {kendall_corr:.4f}, p-value: {kendall_p:.10f}")
print("Conclusion:", "Fail to reject H0" if kendall_p > 0.05 else "Reject H0", "\n")
```

Output:



```
Kendall’s Rank Correlation Hypothesis Test:
H0: There is no ordinal relationship between Flight Distance and Arrival Delay.
H1: There is an ordinal relationship between Flight Distance and Arrival Delay.
Kendall's Rank Correlation: -0.0014, p-value: 0.5048887922
Conclusion: Fail to reject H0
```

Using the scipy.stats module, we determine the Kendall’s Rank Correlation between Flight Distance and Arrival Delay in Minutes. The correlation (-0.0014) and p-value (0.5049) indicate no significant ordinal relationship, meaning ranking flight distances does not predict ranking arrival delays. Here, as p_value > 0.05, we fail to reject the null hypothesis and hence, observe no monotonic relationship between Flight Distance and Arrival Delay.

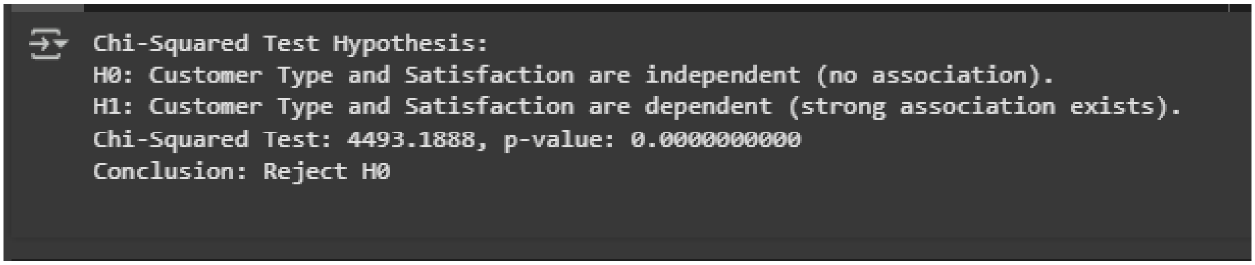
6) Perform Chi-Square Test

The Chi-Square test is a statistical test used to determine if there is a significant association between two categorical variables. It compares observed and expected frequencies in a contingency table. A higher Chi-Square value suggests a stronger relationship. It is widely used in independence testing and goodness-of-fit analysis.

Code:

```
customer_satisfaction_ct = pd.crosstab(df['Customer Type'], df['satisfaction'])
chi2, chi_p, _, _ = stats.chi2_contingency(customer_satisfaction_ct)
print("Chi-Squared Test Hypothesis:")
print("H0: Customer Type and Satisfaction are independent (no association).")
print("H1: Customer Type and Satisfaction are dependent (strong association exists).")
print(f"Chi-Squared Test: {chi2:.4f}, p-value: {chi_p:.10f}")
print("Conclusion:", "Fail to reject H0" if chi_p > 0.05 else "Reject H0", "\n")
```

Output:



```
Chi-Squared Test Hypothesis:
H0: Customer Type and Satisfaction are independent (no association).
H1: Customer Type and Satisfaction are dependent (strong association exists).
Chi-Squared Test: 4493.1888, p-value: 0.0000000000
Conclusion: Reject H0
```

The scipy.stats module is used to perform a Chi-Squared Test on Customer Type and Satisfaction. A high chi-square statistic (4493.1888) and near-zero p-value indicate a strong dependence between these variables, meaning customer type significantly influences satisfaction levels. As the p_value < 0.05, we have to reject the null hypothesis. Hence, there is a dependence between Customer Type and Satisfaction.

Conclusion:

The statistical hypothesis tests performed on the dataset provide insights into relationships between various airline passenger attributes. The Pearson's Correlation Coefficient between Flight Distance and Arrival Delay in Minutes was found to be -0.0020 with a p-value of 0.4771, indicating no significant linear relationship between these variables. Similarly, the Spearman's Rank Correlation was -0.0018 with a p-value of 0.5058, confirming that no monotonic relationship exists. Furthermore, the Kendall's Rank Correlation resulted in -0.0014 with a p-value of 0.5049, reinforcing the finding that changes in flight distance do not systematically influence arrival delays. In all three correlation tests, since the p-value was greater than 0.05, the null hypothesis was not rejected, meaning there is no significant association between flight distance and arrival delay. However, the Chi-Square Test between Customer Type and Satisfaction yielded a chi-square statistic of 4493.1888 and a p-value close to zero, indicating a strong dependence between these categorical variables. This suggests that passenger satisfaction is significantly influenced by customer type. Overall, while flight distance does not appear to predict arrival delays, customer type plays a crucial role in determining passenger satisfaction.

Sairam Konar D15C 27 2024-2025

Experiment 5

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

Problem Statement:
a) Perform Logistic regression to find out relation between variables
b) Apply regression model technique to predict the data on above dataset.

Dataset used: https://yulimezab.github.io/Data-Mining-Project/

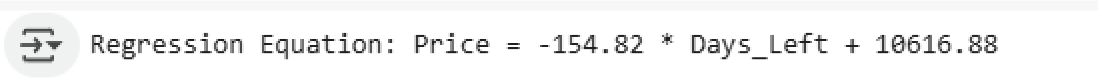
Steps:
Linear Regression:
1) Import Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

2) Train the Linear Regression Model

```
model = LinearRegression()
model.fit(X, y)

slope = model.coef_[0]
intercept = model.intercept_
equation = f"Price = {slope:.2f} * Days_Left + {intercept:.2f}"

print("Regression Equation:", equation)
```

Output:



```
Regression Equation: Price = -154.82 * Days_Left + 10616.88
```

A LinearRegression object is created and fitted to the data using .fit(X, y). The coef_ gives the slope (impact of days left), and intercept_ gives the y-intercept. The regression equation is printed as a summary of the model.

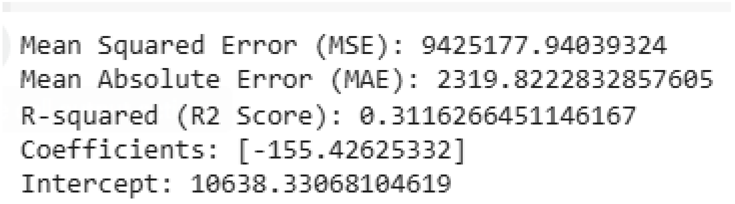
3) Train-Test Split & Evaluation Metrics

Code:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
X_lin = df_economy[['days_left']].values  # Independent variables
y_lin = df_economy['price'].values  # Dependent variable

X_train_lin, X_test_lin, y_train_lin, y_test_lin = train_test_split(X_lin, y_lin, test_size=0.2, random_state=42)

lin_reg = LinearRegression()
lin_reg.fit(X_train_lin, y_train_lin)

y_pred_lin = lin_reg.predict(X_test_lin)

mse = mean_squared_error(y_test_lin, y_pred_lin)
mae = mean_absolute_error(y_test_lin, y_pred_lin)
r2 = r2_score(y_test_lin, y_pred_lin)

coefficients = lin_reg.coef_
intercept = lin_reg.intercept_

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared (R2 Score): {r2}")
print(f"Coefficients: {coefficients}")
print(f"Intercept: {intercept}")
```

Output:



```
Mean Squared Error (MSE): 9425177.94039324
Mean Absolute Error (MAE): 2319.8222832857605
R-squared (R2 Score): 0.3116266451146167
Coefficients: [-155.42625332]
Intercept: 10638.33068104619
```

In the sklearn.linear_model module, we use the LinearRegression() class to perform linear regression, a statistical method that models the relationship between a dependent variable and one or more independent variables. In this case, we are predicting the airline ticket price (dependent variable) based on the number of days_left before departure (independent variable).

The dataset is split into training and testing sets using train_test_split() in an 80:20 ratio to ensure unbiased model evaluation. The model is then trained using the .fit() method, and predictions are generated for the test data. Evaluation metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R²) are computed to assess the performance of the model. The coefficient represents how much the price changes with a unit change in days_left, and the intercept is the expected price when days_left is zero.

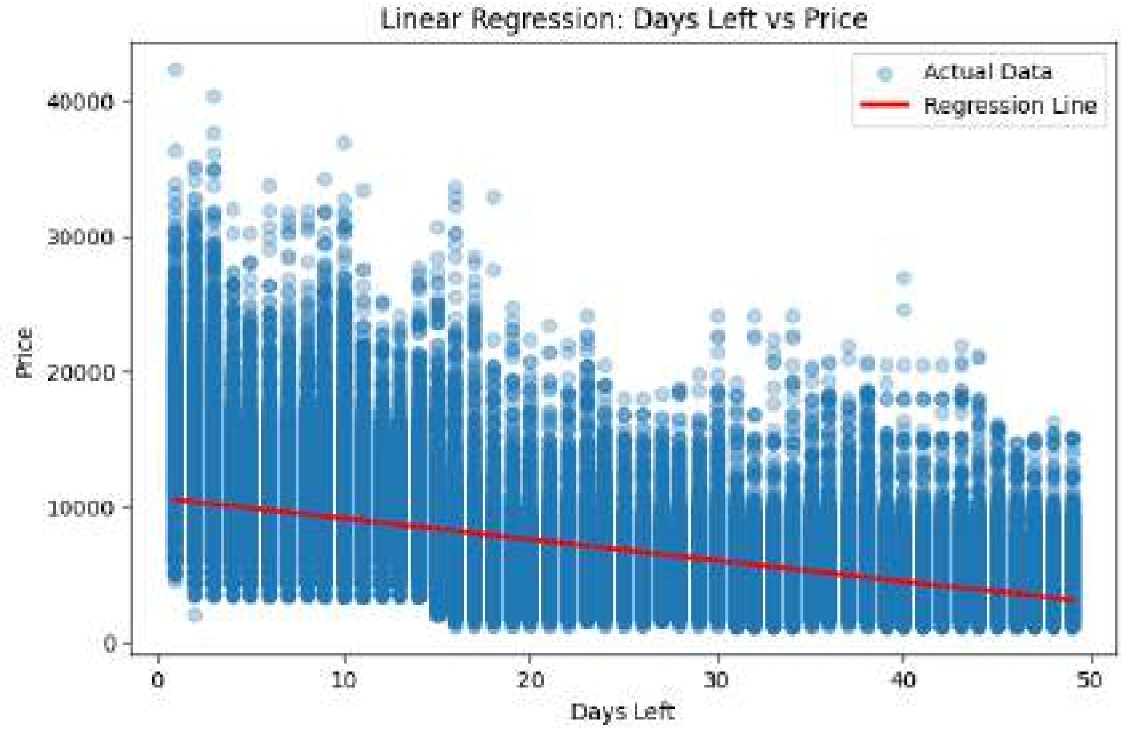
A low R² score or high error values would suggest that days_left alone does not strongly predict ticket prices. The values of coefficients and intercept together form the linear equation used for predictions.

4) Plot graph for Regression plot and line.

Code:

```
days_range = np.linspace(df_economy["days_left"].min(), df_economy["days_left"].max(), 100).reshape(-1, 1)
price_pred = model.predict(days_range)

plt.figure(figsize=(8, 5))
plt.scatter(df_economy["days_left"], df_economy["price"], alpha=0.3, label="Actual Data")
plt.plot(days_range, price_pred, color='red', linewidth=2, label="Regression Line")

plt.xlabel("Days Left")
plt.ylabel("Price")
plt.title("Linear Regression: Days Left vs Price")
plt.legend()
plt.show()
```

Output:



In this step, we visualize the results of the linear regression model using a scatter plot and regression line. We first generate a sequence of values between the minimum and maximum of days_left using np.linspace(), and use the trained model to predict corresponding ticket prices. This gives us a smooth line that represents the model's understanding of the relationship.

Using matplotlib.pyplot, we plot the actual data points as a scatter plot and overlay the predicted regression line in red. The alpha=0.3 parameter makes the data points semi-transparent for better visual clarity. The graph is labeled appropriately to show how ticket prices vary with the number of days left before departure. This helps assess visually whether a linear trend exists between the two variables.

Logistic Regression:

1) Import Required Libraries

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

2) Data Preprocessing

Code:

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

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)
```

In the sklearn.linear_model module, we use the LogisticRegression() class to model binary classification problems. In this case, X and y represent the features and target labels, respectively. The dataset is split into training and testing sets using train_test_split() with 80% used for training and 20% for evaluation.

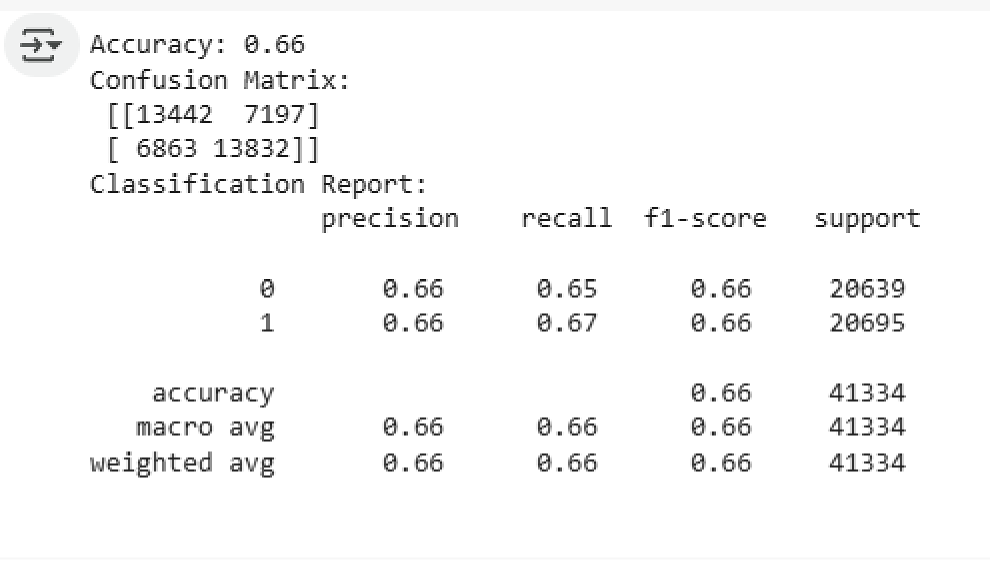
Since logistic regression is sensitive to feature scaling, we use StandardScaler() to normalize the feature values so that they have a mean of 0 and standard deviation of 1. The model is then trained using the .fit() method, which learns the optimal weights that separate the classes based on the input features. These weights are used to compute the probability of a data point belonging to a particular class.

3) Prediction & Evaluation

Code:

```
y_pred = log_reg.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

Output:



```
Accuracy: 0.66
Confusion Matrix:
 [[13442  7197]
 [ 6863 13832]]
Classification Report:
               precision    recall  f1-score   support

           0       0.66      0.65      0.66     20639
           1       0.66      0.67      0.66     20695

    accuracy                           0.66     41334
   macro avg       0.66      0.66      0.66     41334
weighted avg       0.66      0.66      0.66     41334
```

After training the logistic regression model, we evaluate its performance using several metrics from the sklearn.metrics module. The .predict() function generates predictions on the test set. We then calculate the accuracy, which measures the percentage of correct predictions out of the total.

The confusion matrix provides a breakdown of true positives, true negatives, false positives, and false negatives — useful for understanding the balance of predictions. The classification report shows precision, recall, and F1-score for each class, offering a deeper look at the model's performance across both categories (0 and 1).

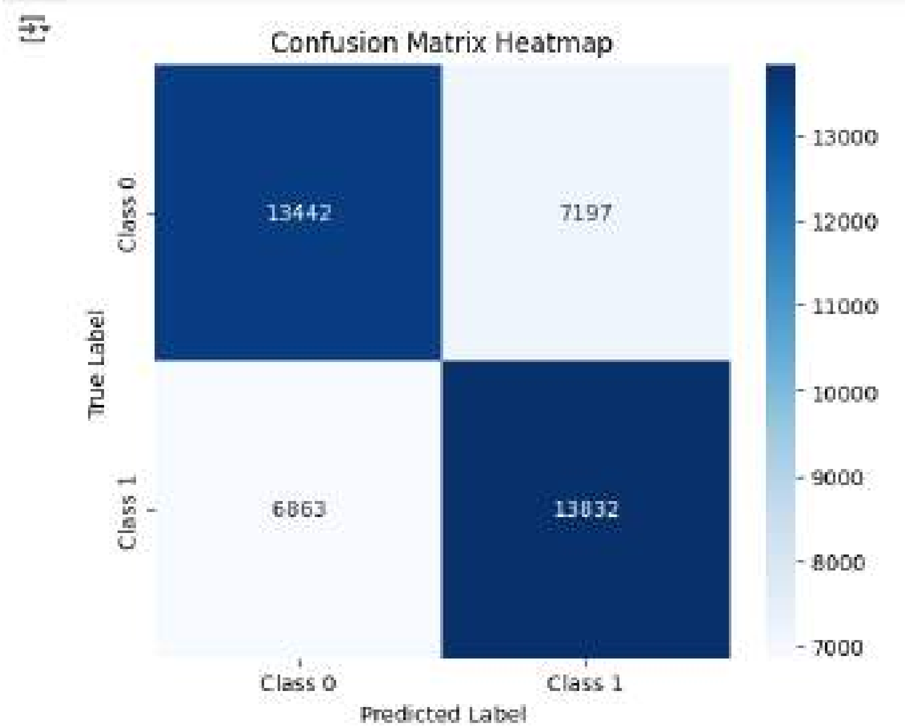
The results show an accuracy of 66%, indicating moderate classification performance. Precision and recall are balanced across both classes. However, the model may benefit from tuning or using additional features, as a 66% accuracy suggests room for improvement.

4) Visualize the Logistic Regression

Code:

```
import seaborn as sns
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Class 0', 'Class 1'],
yticklabels=['Class 0', 'Class 1'])

plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix Heatmap")
plt.show()

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:



To visualize the performance of the logistic regression model, we use Seaborn's heatmap() function to plot the confusion matrix. The heatmap provides a clear visual representation of how the model's predictions compare to the actual values, with darker colors indicating higher counts.

We annotate the heatmap with actual values using annot=True and format them as integers (fmt='d'). The color map Blues adds clarity to the visual. Axes are labeled to distinguish between predicted and true classes, and custom tick labels (Class 0, Class 1) improve interpretability. The plot helps quickly identify where the model is performing well or struggling.

Below the heatmap, the classification report is printed again for reference, showing metrics like precision, recall, and F1-score for each class.

Conclusion:

In Logistic Regression, we used the price_category to predict the values in the test split of the dataset. After running the regression model using python library, the accuracy of the model comes out to be 66% which means that few values were not predicted correctly. To support this, the heatmap of the confusion matrix shows that there are false positives and false negatives in the trained model.

In case of linear regression, the ‘days_left’ numeric column is used to predict the ‘price’ in the testing set. Once the dataset is splitted, the training model is used and regression model is applied on it. The performance parameters such as MSE (9425177) and R-squared score (0.331) indicate that there is a great difference between the predicted and actual values which are also visible in the scatterplot.

Sairam Konar D15C 27 2024-2025

Experiment 6

Aim: Classification modelling

a. Choose a classifier for a classification problem.

b. Evaluate the performance of the classifier.

Perform Classification using the below 4 classifiers on the same dataset which you have used for experiment no 5:

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

Theory:

Classification Modeling: Theory & Techniques

Classification modeling is a type of supervised learning in machine learning where the goal is to predict the category or class of a given data point based on input features. The model is trained using labeled data (i.e., data where the output class is known).

Classification problems can be:

- **Binary Classification:** Two classes (e.g., spam vs. not spam).
- **Multiclass Classification:** More than two classes (e.g., classifying types of flowers).
- **Multi-label Classification:** Each sample can belong to multiple classes.

1. K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric classification algorithm based on proximity to labeled examples.

Working Principle:

1. Choose a value for K (number of neighbors).
2. Compute the distance between the new data point and all training samples.
3. Select the K nearest neighbors.
4. Assign the majority class among the K neighbors as the predicted class.

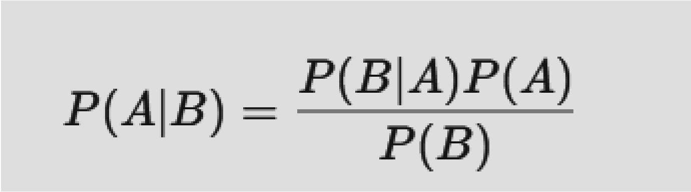
Common Distance Metrics:

- **Euclidean Distance:** $d=(\sum(xi-yi)^{2})^{½}$ (Most commonly used)
- **Manhattan Distance:** $d=\sum|xi-yi|$
- **Minkowski Distance:** A generalization of Euclidean and Manhattan distances.

2. Naïve Bayes (NB)

Naïve Bayes is a probabilistic classifier based on **Bayes’ Theorem**, assuming independence between predictors.

Bayes' Theorem:



$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where:

- P(A|B) = Probability of class A given data B
- P(B|A) = Probability of data B given class A
- P(A) = Prior probability of class A
- P(B) = Prior probability of data B

Types of Naïve Bayes Classifiers:

1. **Gaussian Naïve Bayes:** Assumes normal distribution of features.
2. **Multinomial Naïve Bayes:** Used for text classification (e.g., spam detection).
3. **Bernoulli Naïve Bayes:** Used when features are binary (e.g., word presence in spam detection).

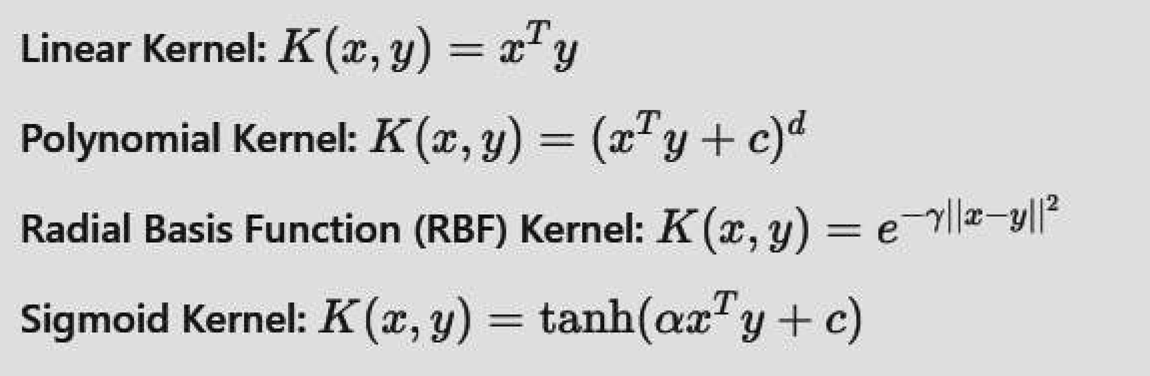
3. Support Vector Machines (SVMs)

SVM is a powerful classification algorithm that finds the optimal hyperplane to separate data points into different classes.

Working Principle:

1. **Hyperplane:** A decision boundary that maximizes the margin between two classes.
2. **Support Vectors:** Data points that lie closest to the hyperplane and influence its position.
3. **Kernel Trick:** SVM can handle non-linearly separable data using kernel functions to transform the input space.

Common Kernel Functions:



Linear Kernel: $K(x, y) = x^T y$

Polynomial Kernel: $K(x, y) = (x^T y + c)^d$

Radial Basis Function (RBF) Kernel: $K(x, y) = e^{-\gamma||x-y||^2}$

Sigmoid Kernel: $K(x, y) = \tanh(\alpha x^T y + c)$

4. Decision Tree

A Decision Tree is a flowchart-like structure where internal nodes represent features, branches represent decisions, and leaves represent class labels.

Working Principle:

1. **Splitting Criteria:** Choose the best feature to split the data.
    - **Gini Index:** Measures impurity (Gini=$1-\sum pi^2$).
    - **Entropy (Information Gain):** Measures information gained from a split.
2. **Recursive Splitting:** Continue splitting nodes until a stopping criterion is met.
3. **Pruning:** Reduces overfitting by trimming branches.

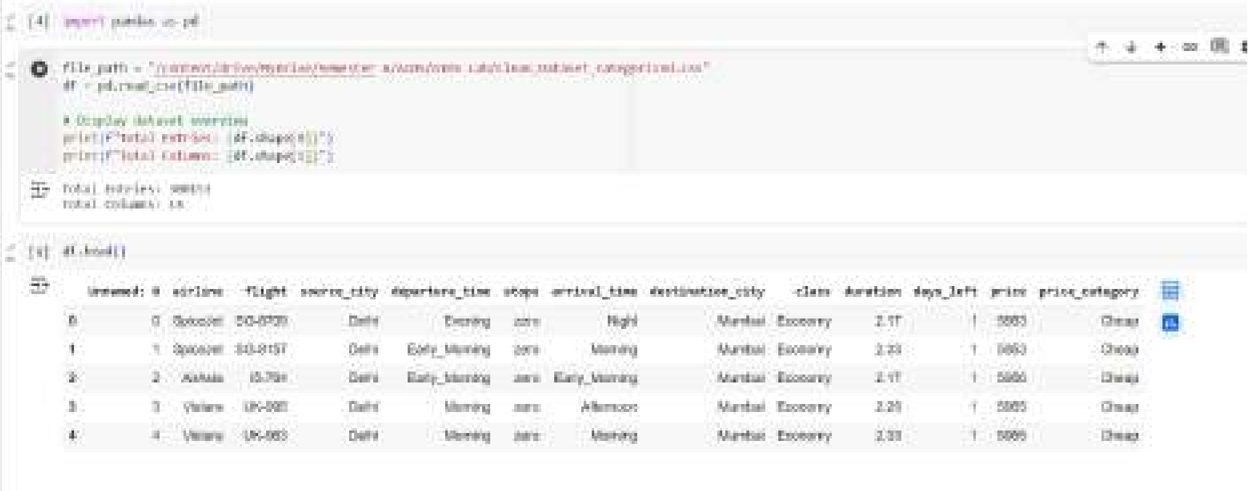
Types of Decision Trees:

- **ID3 (Iterative Dichotomiser 3)** – Uses entropy for splitting.
- **C4.5 & C5.0** – Improvement over ID3 (handles continuous data).
- **CART (Classification and Regression Trees)** – Uses Gini Index.

Steps:

1) Load the Dataset

The dataset is loaded from a CSV file using pandas and First 5 entries in the Dataset is shown using df.head() and Total rows and columns are printed using df.shape[n].



```
[4] import pandas as pd
```

```
file_path = "/content/drive/MyDrive/semester [illegible]/clean_dataset_categorized.csv"
df = pd.read_csv(file_path)

# Display dataset overview
print(f"Total entries: {df.shape[0]}")
print(f"Total columns: {df.shape[1]}")
```

```
Total entries: 300153
Total columns: 13
```

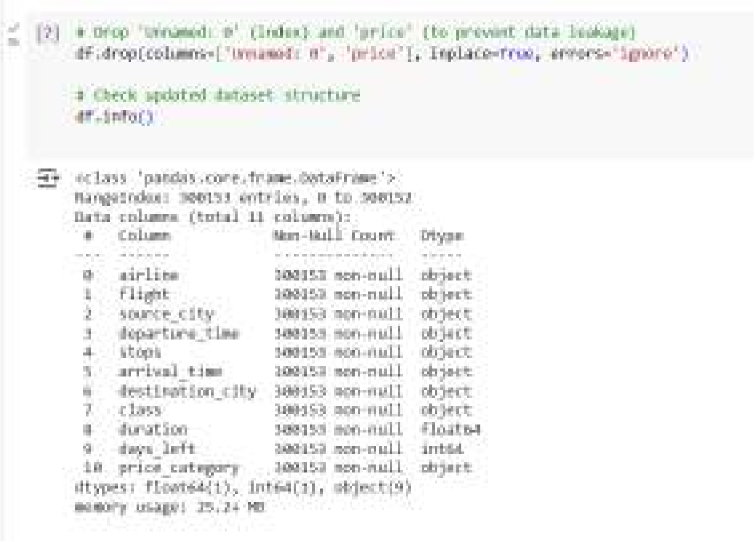
```
[5] df.head()
```

```
   Unnamed: 0  airline   flight source_city departure_time stops  arrival_time destination_city    class duration days_left price price_category
0           0 SpiceJet  SG-8709       Delhi        Evening  zero         Night           Mumbai  Economy     2.17         1  5953          Cheap
1           1 SpiceJet  SG-8157       Delhi  Early_Morning  zero       Morning           Mumbai  Economy     2.33         1  5953          Cheap
2           2  AirAsia   I5-764       Delhi  Early_Morning  zero Early_Morning           Mumbai  Economy     2.17         1  5956          Cheap
3           3  Vistara   UK-995       Delhi        Morning  zero     Afternoon           Mumbai  Economy     2.25         1  5955          Cheap
4           4  Vistara   UK-963       Delhi        Morning  zero       Morning           Mumbai  Economy     2.33         1  5955          Cheap
```

2) Data Preprocessing

(a) Drop Unnecessary Columns

Here, we drop the unnecessary columns such as Unnamed: 0 and price, which are currently not required for the following experiment.

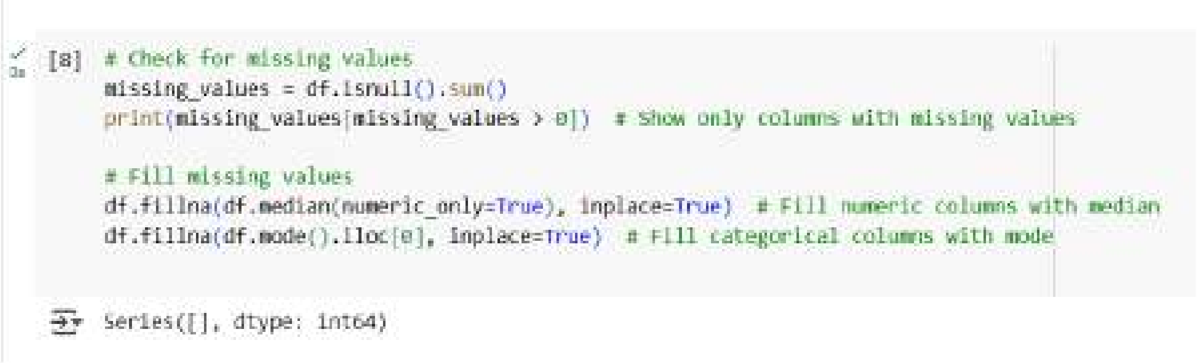


```
[7] # Drop 'Unnamed: 0' (index) and 'price' (to prevent data leakage)
    df.drop(columns=['Unnamed: 0', 'price'], inplace=True, errors='ignore')

    # Check updated dataset structure
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 11 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   airline           300153 non-null  object 
 1   flight            300153 non-null  object 
 2   source_city       300153 non-null  object 
 3   departure_time    300153 non-null  object 
 4   stops             300153 non-null  object 
 5   arrival_time      300153 non-null  object 
 6   destination_city  300153 non-null  object 
 7   class             300153 non-null  object 
 8   duration          300153 non-null  float64
 9   days_left         300153 non-null  int64  
 10  price_category    300153 non-null  object 
dtypes: float64(1), int64(1), object(9)
memory usage: 25.2+ MB
```

(b) Handling Missing Values

We fill up the missing values such that the numeric columns are filled with the median values, and the categorical columns are filled with mode of that column.

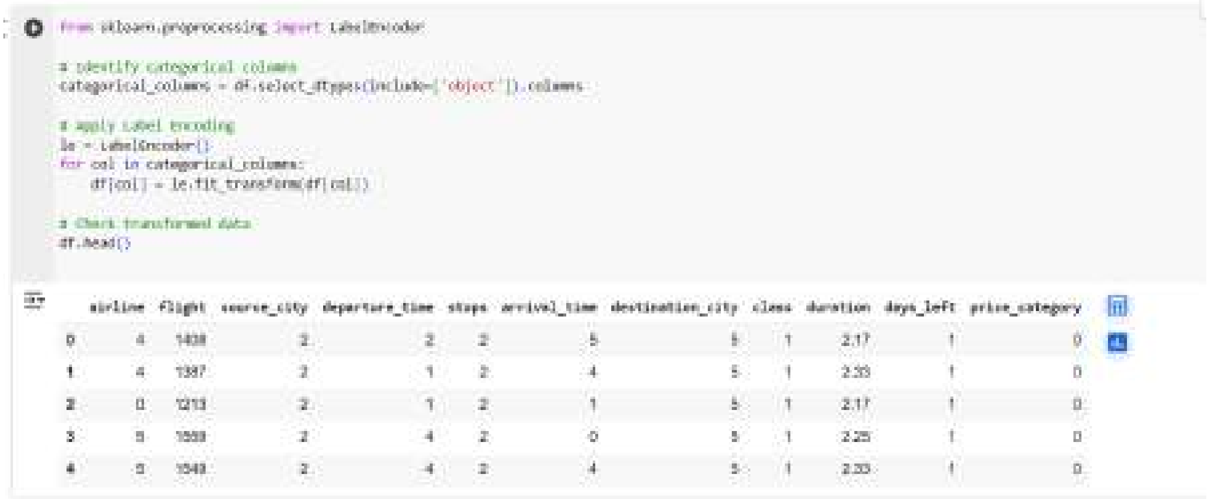


```
[8] # Check for missing values
    missing_values = df.isnull().sum()
    print(missing_values[missing_values > 0])  # Show only columns with missing values

    # Fill missing values
    df.fillna(df.median(numeric_only=True), inplace=True)  # Fill numeric columns with median
    df.fillna(df.mode().iloc[0], inplace=True)  # Fill categorical columns with mode
```

```
Series([], dtype: int64)
```

(c) Encode Categorical Variables

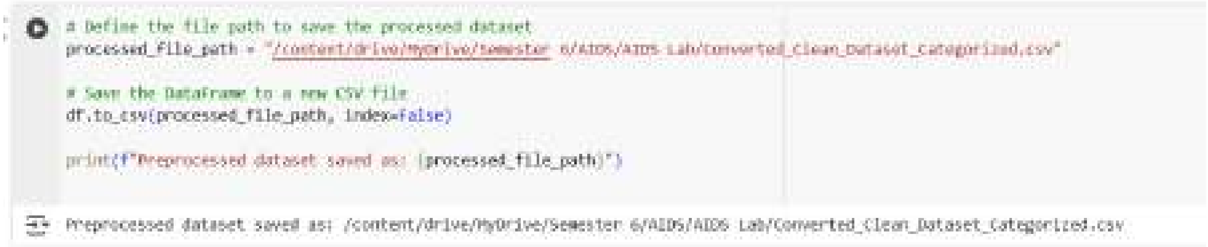
The categorical columns are encoded so that it becomes suitable for the algorithm to make it easier for the algorithm to make the classification.



```
from sklearn.preprocessing import LabelEncoder

# identify categorical columns
categorical_columns = df.select_dtypes(include=['object']).columns

# apply Label Encoding
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])

# Check transformed data
df.head()
```

```
   airline  flight  source_city  departure_time  stops  arrival_time  destination_city  class  duration  days_left  price_category
0        4    1408            2               2      2             5                 5      1      2.17          1               0
1        4    1387            2               1      2             4                 5      1      2.33          1               0
2        0    1213            2               1      2             1                 5      1      2.17          1               0
3        5    1559            2               4      2             0                 5      1      2.25          1               0
4        5    1549            2               4      2             4                 5      1      2.33          1               0
```

(d) Save Data In New File

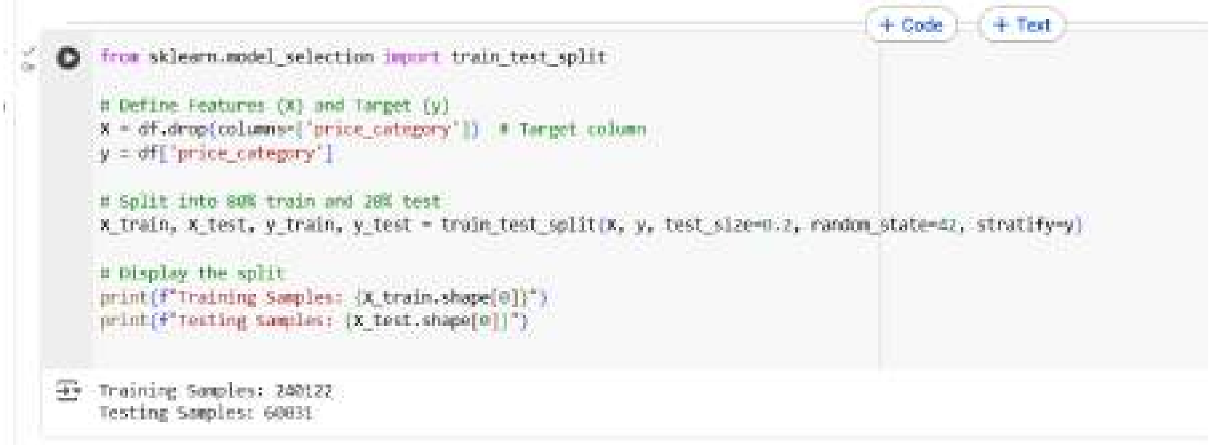


```
# Define the file path to save the processed dataset
processed_file_path = "/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Converted_Clean_Dataset_Categorized.csv"

# Save the DataFrame to a new CSV file
df.to_csv(processed_file_path, index=False)

print(f"Preprocessed dataset saved as: {processed_file_path}")
```

```
Preprocessed dataset saved as: /content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Converted_Clean_Dataset_Categorized.csv
```

3) Split Into Train and Test

The dataset it then split into training and testing such that the models are trained on 80% of the dataset and 20% is used to test the models for their accuracy.



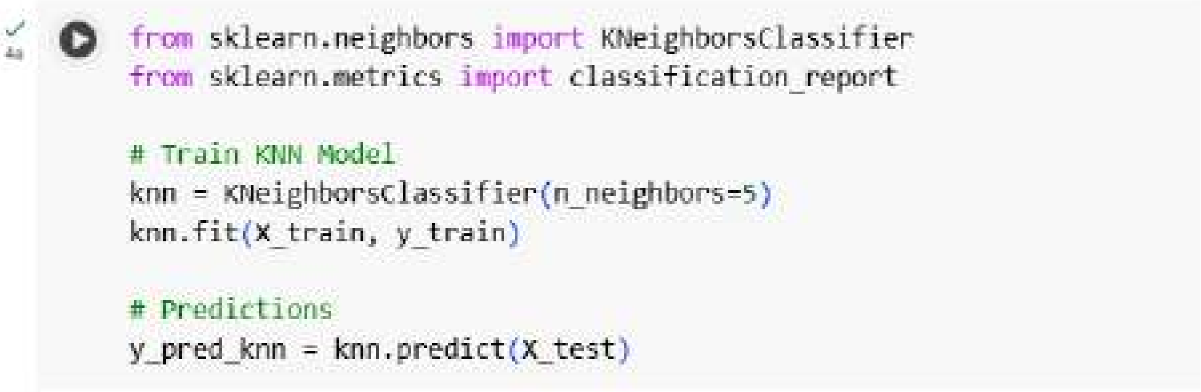
```
from sklearn.model_selection import train_test_split

# Define Features (X) and Target (y)
X = df.drop(columns=['price_category'])  # Target column
y = df['price_category']

# Split into 80% train and 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Display the split
print(f"Training Samples: {X_train.shape[0]}")
print(f"Testing Samples: {X_test.shape[0]}")
```

```
Training Samples: 240122
Testing Samples: 60031
```

4) Train & Evaluate Classifiers

- **K-Nearest Neighbors (KNN)**

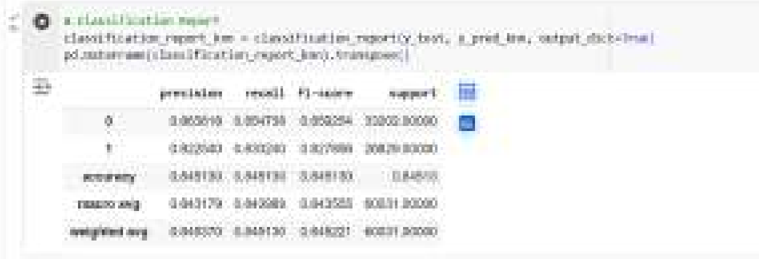
From sklearn.neighbors library, we import the KNeighboursClassifier. We call this function and pass a parameter for the number of neighbours to be used. Here, we are passing 5 neighbours. After that, we fit the model with our training datasets (X and y) and create a variable to store the predicted values.



```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Train KNN Model
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

# Predictions
y_pred_knn = knn.predict(X_test)
```

I) Classification Report

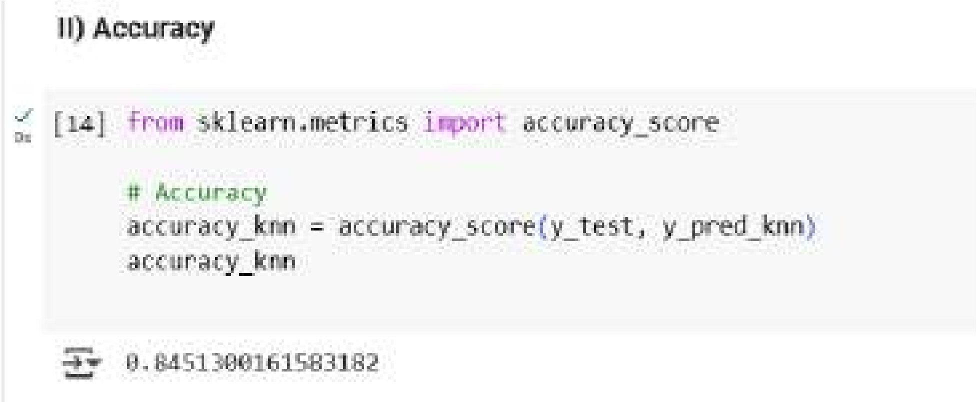
Using the classification_report function, we generate the classification report which would give the performance metrics such as accuracy, precision, recall, f1-score, support, etc.



```
# Classification Report
classification_report_knn = classification_report(y_test, y_pred_knn, output_dict=True)
pd.DataFrame(classification_report_knn).transpose()
```

```
              precision    recall  f1-score       support
0              0.863619  0.854739  0.859254  33202.00000
1              0.822543  0.833240  0.827686  26829.00000
accuracy       0.845130  0.845130  0.845130      0.84513
macro avg      0.843179  0.843989  0.843555  60031.00000
weighted avg   0.845370  0.845130  0.845221  60031.00000
```

II) Accuracy

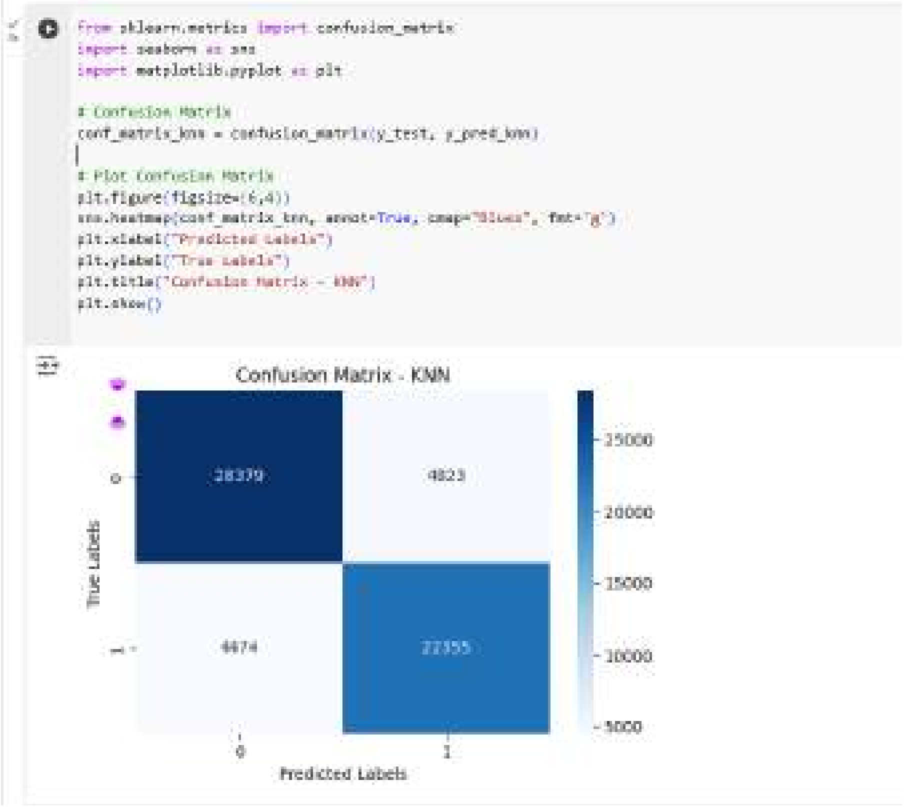


II) Accuracy

```
[14] from sklearn.metrics import accuracy_score

     # Accuracy
     accuracy_knn = accuracy_score(y_test, y_pred_knn)
     accuracy_knn
```

```
0.8451300161583182
```

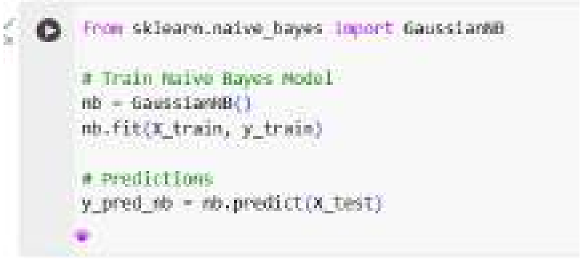
III) Confusion Matrix



```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion Matrix
conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_knn, annot=True, cmap="Blues", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - KNN")
plt.show()
```

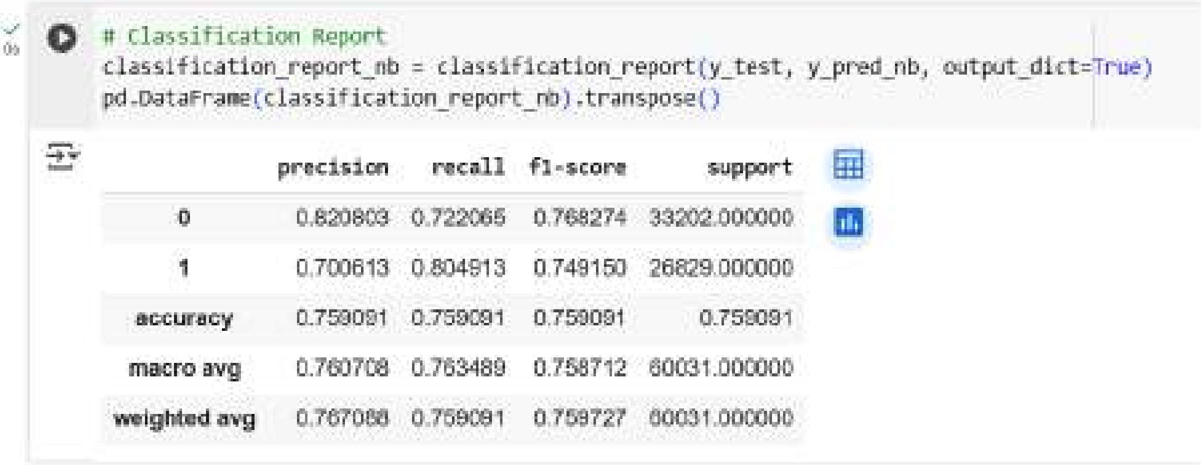
The KNN model achieves 84.51% accuracy, performing well across both classes. Class 0 has slightly higher precision (0.8633) and recall (0.8547) than Class 1 (0.8225, 0.8332), indicating a minor bias. The confusion matrix shows 28,379 True Negatives (TN) and 22,355 True Positives (TP), with 4,823 False Positives (FP) and 4,474 False Negatives (FN). Misclassification is fairly balanced, though Class 1 has slightly higher errors. Tuning K-values, distance metrics, and handling class imbalance can improve performance.

- **Naive Bayes**



```
from sklearn.naive_bayes import GaussianNB

# Train Naive Bayes Model
nb = GaussianNB()
nb.fit(X_train, y_train)

# Predictions
y_pred_nb = nb.predict(X_test)
```

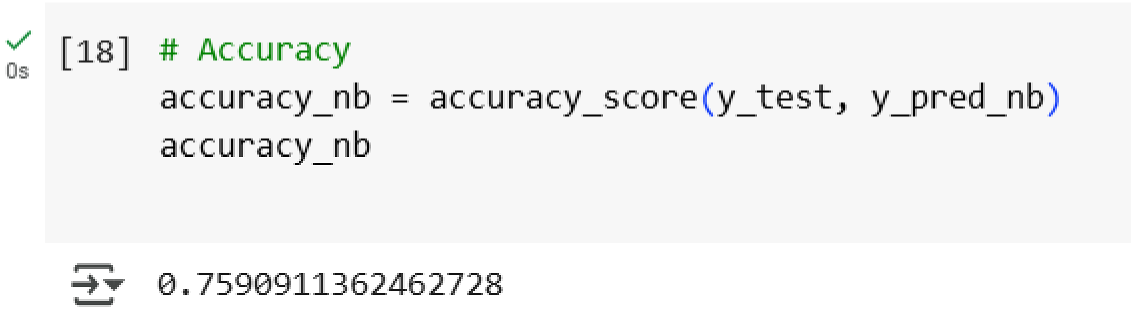
I) Classification Report



```
# Classification Report
classification_report_nb = classification_report(y_test, y_pred_nb, output_dict=True)
pd.DataFrame(classification_report_nb).transpose()
```

	precision	recall	f1-score	support
0	0.820803	0.722065	0.768274	33202.000000
1	0.700613	0.804913	0.749150	26829.000000
accuracy	0.759091	0.759091	0.759091	0.759091
macro avg	0.760708	0.763489	0.758712	60031.000000
weighted avg	0.767088	0.759091	0.759727	60031.000000

II) Accuracy

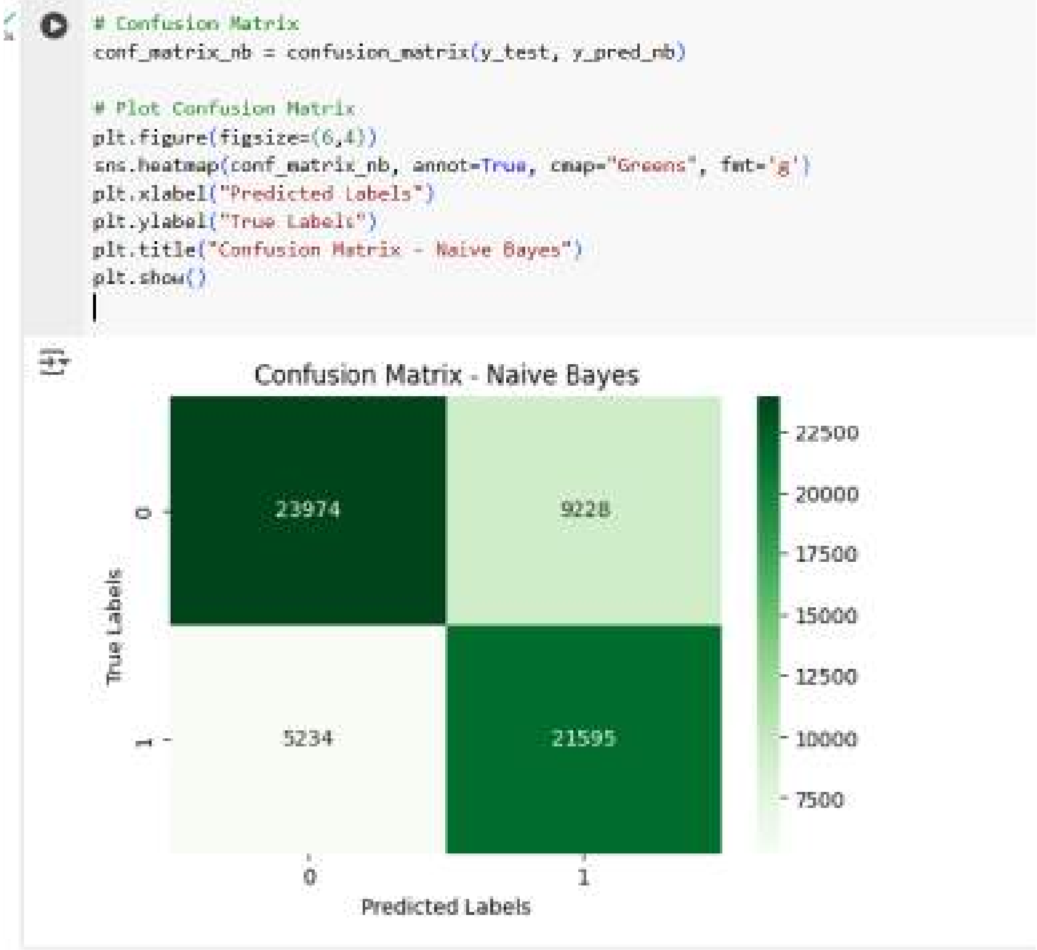


```
[18] # Accuracy
     accuracy_nb = accuracy_score(y_test, y_pred_nb)
     accuracy_nb
```

```
0.7590911362462728
```

\

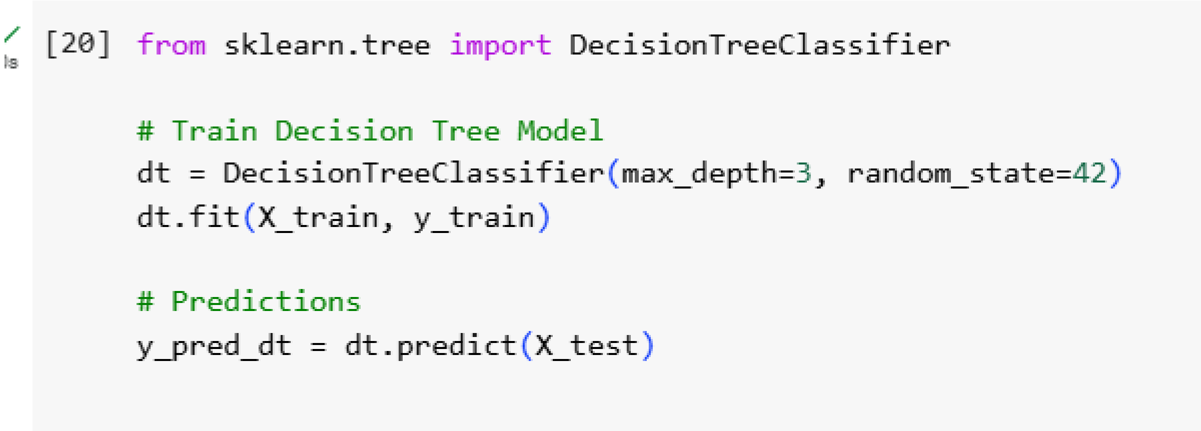
III) Confusion Matrix



```
# Confusion Matrix
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_nb, annot=True, cmap="Greens", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - Naive Bayes")
plt.show()
```

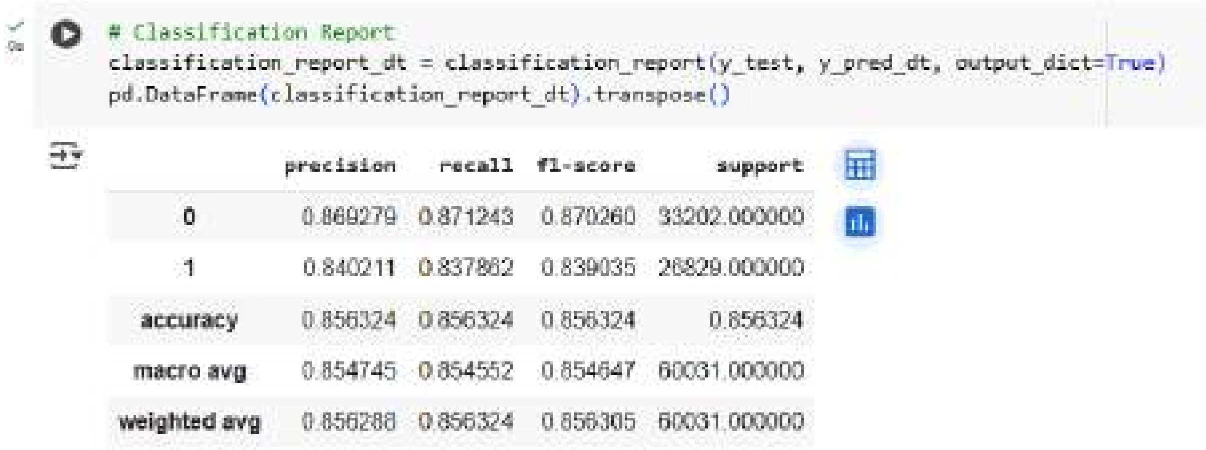
The Naive Bayes model (GaussianNB) achieved an accuracy of 75.91%. It performed slightly better for class 1 in terms of recall (80.49%), but precision was higher for class 0 (82.08%), showing a trade-off between the two. The F1-scores were moderately balanced, around 0.75–0.76 for both classes. The confusion matrix revealed more false positives (9,228) and false negatives (5,234) than KNN, indicating it was more prone to misclassification. Overall, Naive Bayes performed decently but not as accurately as KNN.

- **Decision Tree (With Visualization)**



```
[20] from sklearn.tree import DecisionTreeClassifier

     # Train Decision Tree Model
     dt = DecisionTreeClassifier(max_depth=3, random_state=42)
     dt.fit(X_train, y_train)

     # Predictions
     y_pred_dt = dt.predict(X_test)
```

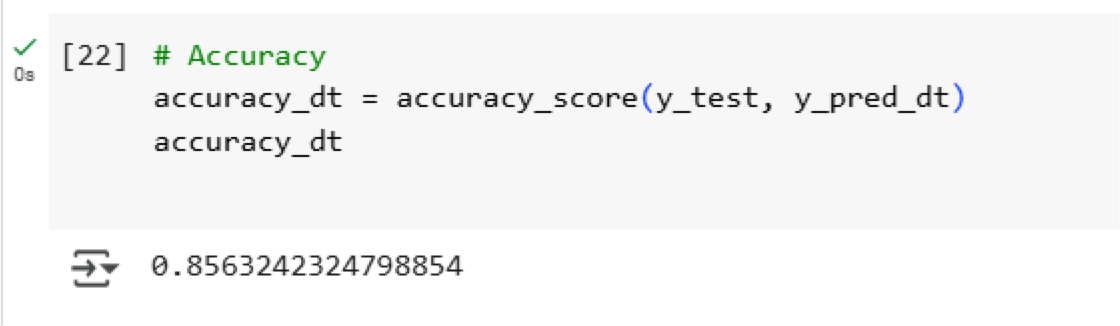
I) Classification Report



```
# Classification Report
classification_report_dt = classification_report(y_test, y_pred_dt, output_dict=True)
pd.DataFrame(classification_report_dt).transpose()
```

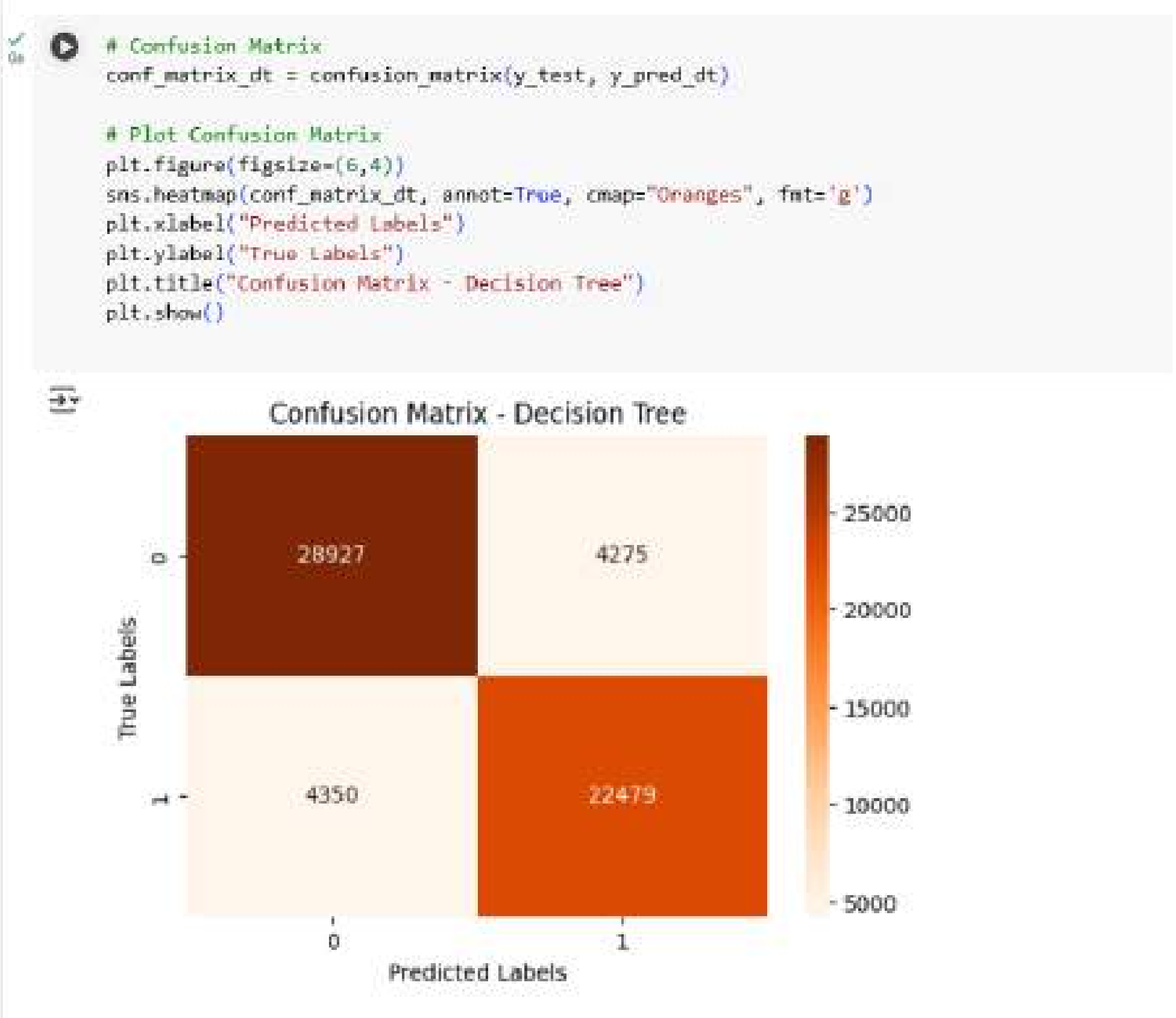
	precision	recall	f1-score	support
0	0.869279	0.871243	0.870260	33202.000000
1	0.840211	0.837862	0.839035	26829.000000
accuracy	0.856324	0.856324	0.856324	0.856324
macro avg	0.854745	0.854552	0.854647	60031.000000
weighted avg	0.856288	0.856324	0.856305	60031.000000

II) Accuracy



```
[22] # Accuracy
     accuracy_dt = accuracy_score(y_test, y_pred_dt)
     accuracy_dt

0.8563242324798854
```

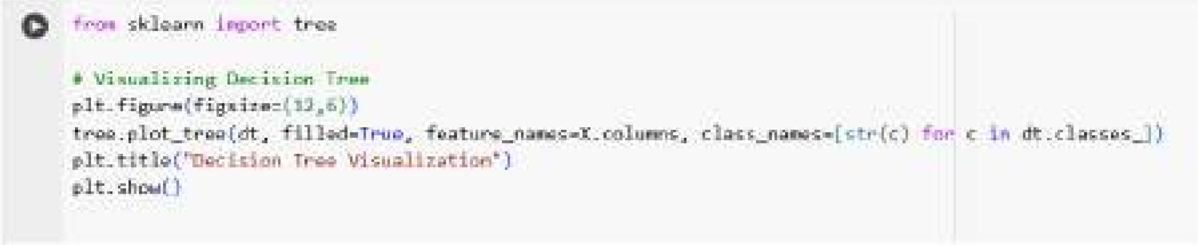
III) Confusion Matrix



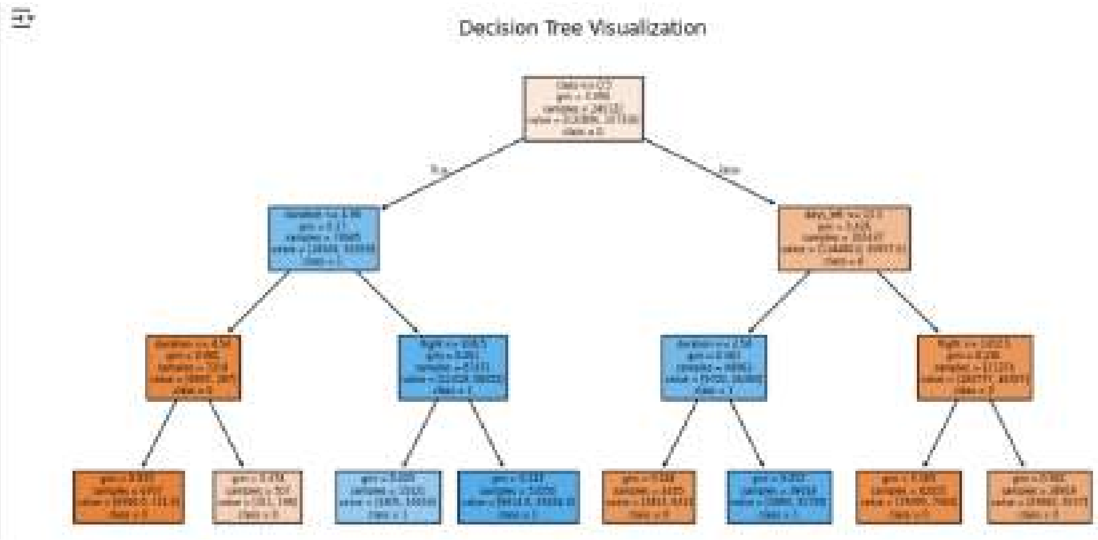
```
# Confusion Matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_dt, annot=True, cmap="Oranges", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - Decision Tree")
plt.show()
```

The Decision Tree classifier achieved an overall accuracy of approximately 85.63%, demonstrating a balanced performance across both classes. From the classification report, we observe that class 0 had slightly higher precision and recall compared to class 1, indicating the model is slightly better at predicting class 0 instances. The F1-scores for both classes are close—0.870 for class 0 and 0.839 for class 1—showing a good trade-off between precision and recall. The confusion matrix further confirms this, with a high number of correctly predicted instances for both classes and a relatively low number of misclassifications.

IV) Visualization



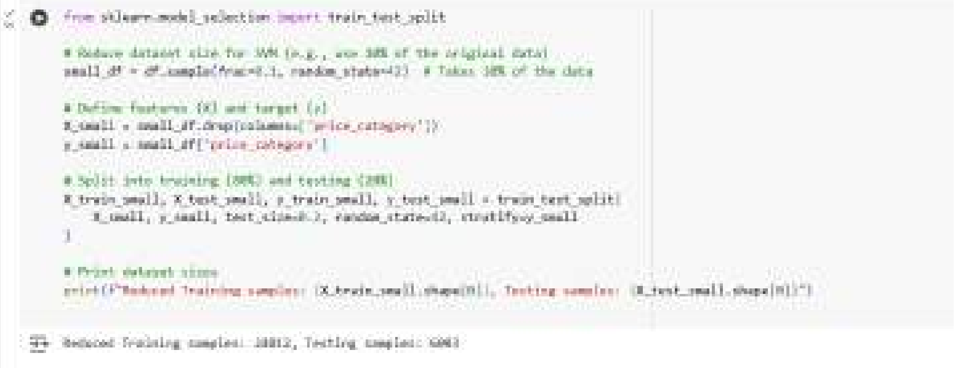
```
from sklearn import tree

# Visualizing Decision Tree
plt.figure(figsize=(12,6))
tree.plot_tree(dt, filled=True, feature_names=X.columns, class_names=[str(c) for c in dt.classes_])
plt.title('Decision Tree Visualization')
plt.show()
```



The Decision Tree visualization, limited to a maximum depth of 3, reveals that features such as duration, flight, and days_left played a key role in the classification. This controlled depth helps maintain interpretability while preventing overfitting. Overall, the model shows strong predictive capability and clear decision logic, making it a solid baseline for classification tasks.

- **Support Vector Machines (SVM)**

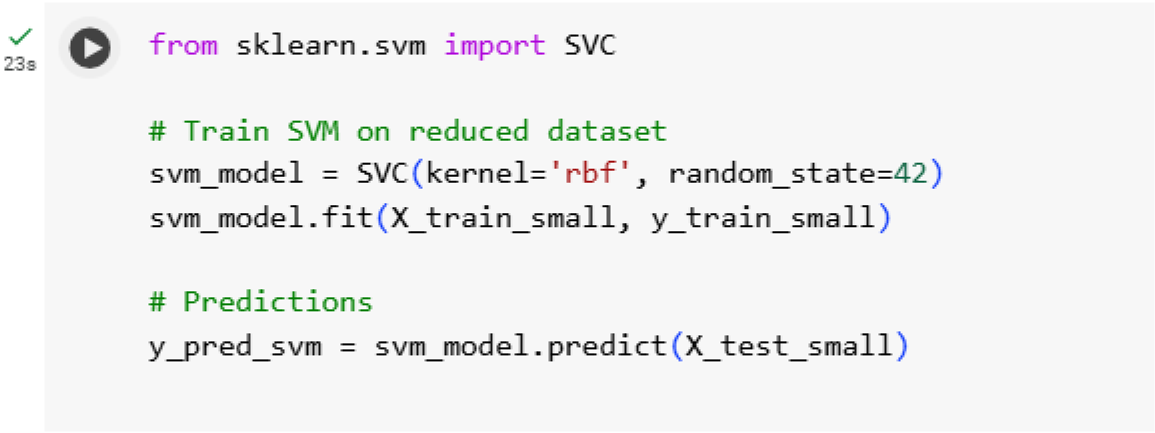
1) Performed On Small Set



```
from sklearn.model_selection import train_test_split

# Reduce dataset size for SVM (e.g., use 10% of the original data)
small_df = df.sample(frac=0.1, random_state=42)  # Takes 10% of the data

# Define features (X) and target (y)
X_small = small_df.drop(columns=['price_category'])
y_small = small_df['price_category']

# Split into training (80%) and testing (20%)
X_train_small, X_test_small, y_train_small, y_test_small = train_test_split(
    X_small, y_small, test_size=0.2, random_state=42, stratify=y_small
)

# Print dataset sizes
print(f"Reduced Training samples: {X_train_small.shape[0]}, Testing samples: {X_test_small.shape[0]}")
```

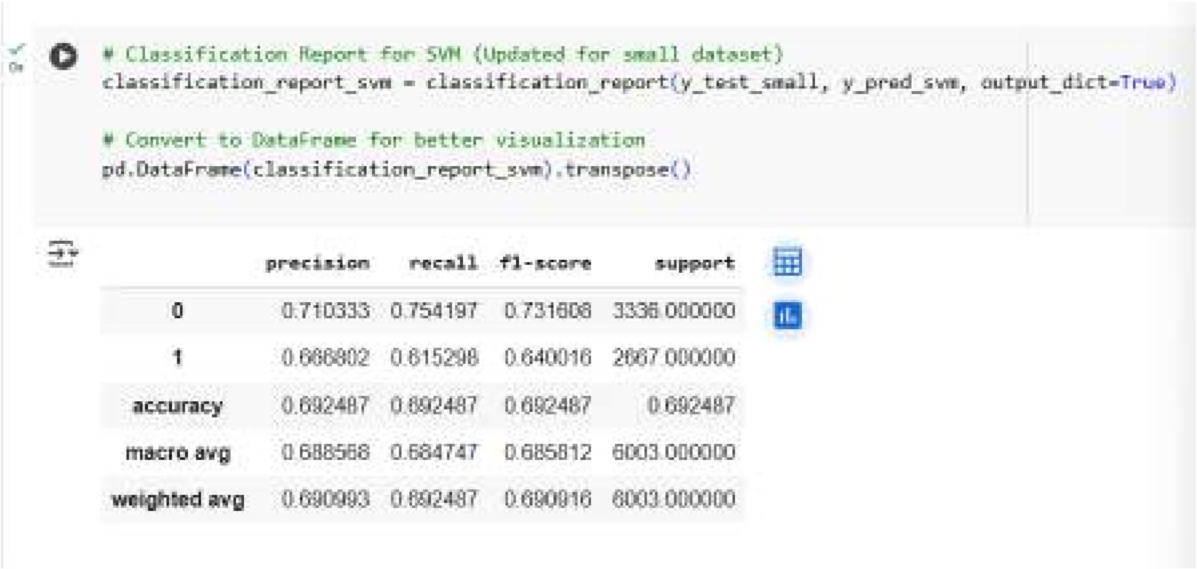
```
Reduced Training samples: 24012, Testing samples: 6003
```

2) Model Train



```
from sklearn.svm import SVC

# Train SVM on reduced dataset
svm_model = SVC(kernel='rbf', random_state=42)
svm_model.fit(X_train_small, y_train_small)

# Predictions
y_pred_svm = svm_model.predict(X_test_small)
```

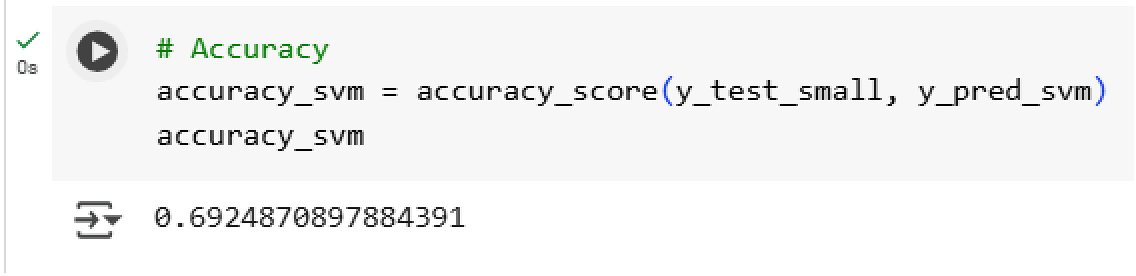
I) Classification Report



```
# Classification Report for SVM (Updated for small dataset)
classification_report_svm = classification_report(y_test_small, y_pred_svm, output_dict=True)

# Convert to DataFrame for better visualization
pd.DataFrame(classification_report_svm).transpose()
```

	precision	recall	f1-score	support
0	0.710333	0.754197	0.731608	3336.000000
1	0.668802	0.615298	0.640016	2667.000000
accuracy	0.692487	0.692487	0.692487	0.692487
macro avg	0.688568	0.684747	0.685812	6003.000000
weighted avg	0.690993	0.692487	0.690916	6003.000000

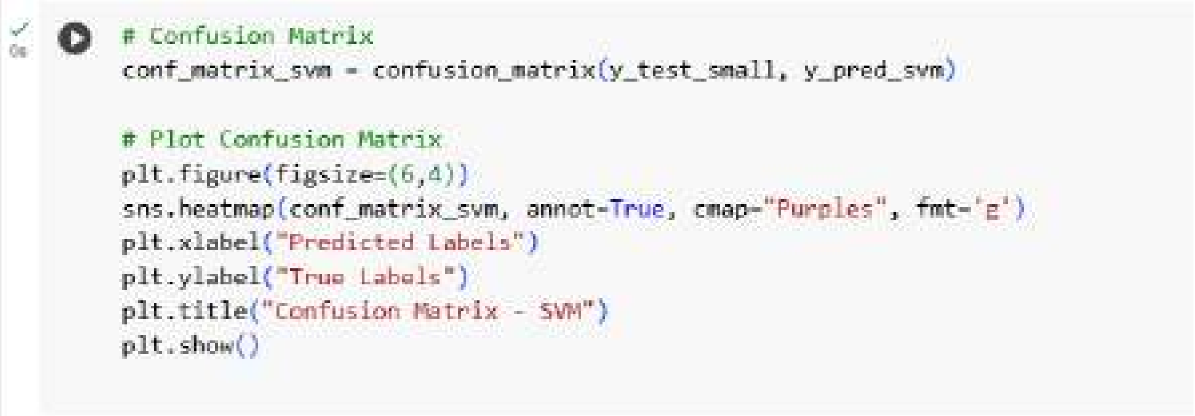
II) Accuracy



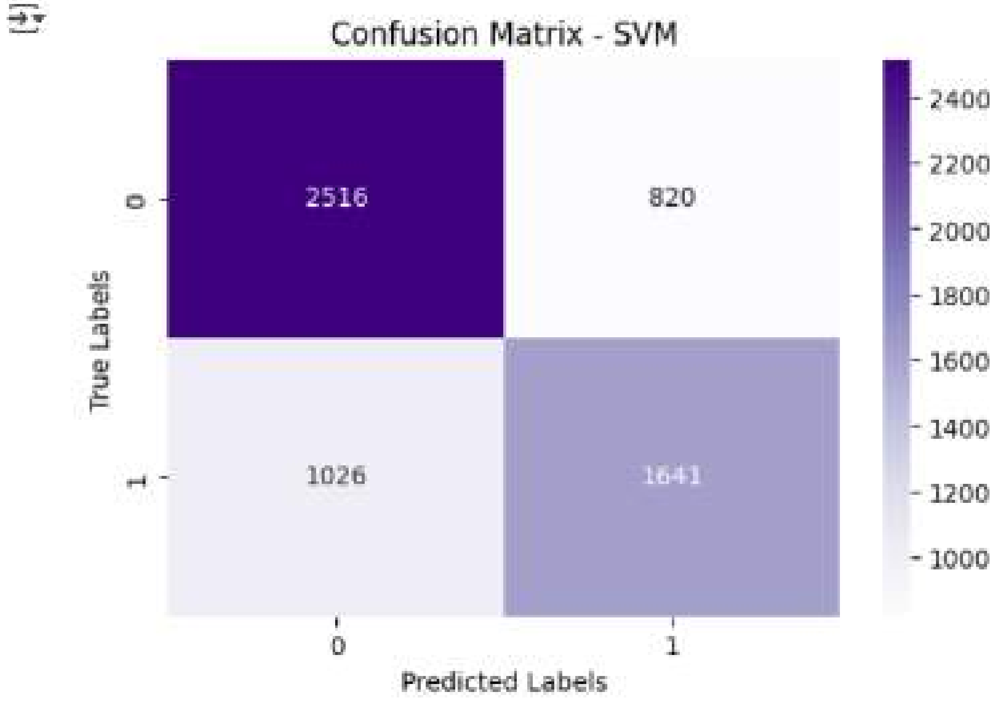
```
# Accuracy
accuracy_svm = accuracy_score(y_test_small, y_pred_svm)
accuracy_svm
```

```
0.6924870897884391
```

III) Confusion Matrix

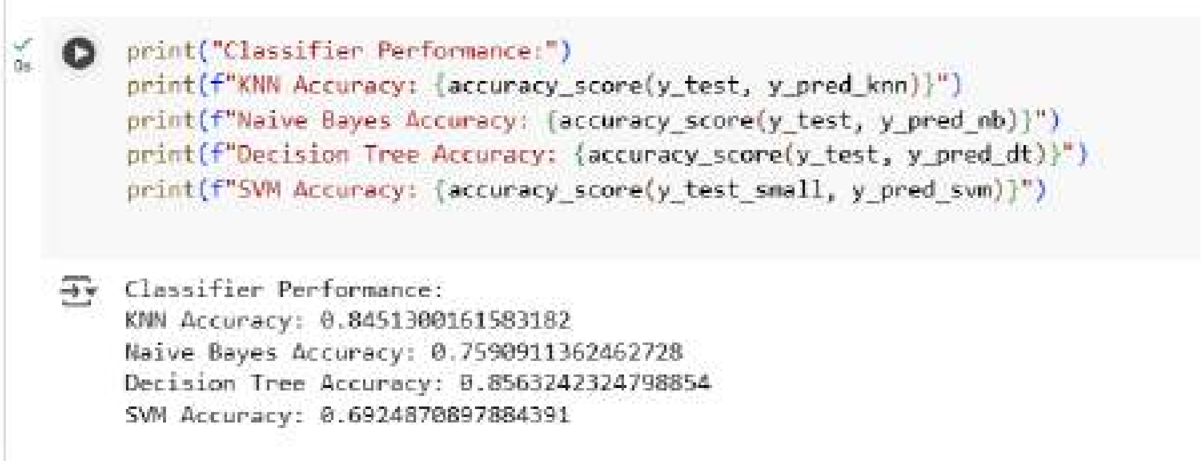


```
# Confusion Matrix
conf_matrix_svm = confusion_matrix(y_test_small, y_pred_svm)

# Plot Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix_svm, annot=True, cmap="Purples", fmt='g')
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix - SVM")
plt.show()
```



To evaluate the Support Vector Machine (SVM) model, a reduced dataset consisting of 10% of the original data was used to optimize computational efficiency. The SVM, trained with an RBF kernel, achieved an accuracy of approximately 69.25%. From the classification report, class 0 had a higher precision (0.71) and recall (0.75), while class 1 showed slightly lower performance with a precision of 0.67 and recall of 0.62. This indicates that the model performs better in identifying class 0 instances. The confusion matrix shows 2,516 correct predictions for class 0 and 1,641 for class 1, while misclassifications include 820 and 1,026 instances respectively. Although the overall performance is lower compared to the Decision Tree model, the SVM still demonstrates decent generalization on a smaller dataset and could benefit from further hyperparameter tuning or feature scaling to improve classification of minority cases.

5) Compare Classifiers



```
print("Classifier Performance:")
print(f"KNN Accuracy: {accuracy_score(y_test, y_pred_knn)}")
print(f"Naive Bayes Accuracy: {accuracy_score(y_test, y_pred_nb)}")
print(f"Decision Tree Accuracy: {accuracy_score(y_test, y_pred_dt)}")
print(f"SVM Accuracy: {accuracy_score(y_test_small, y_pred_svm)}")
```

```
Classifier Performance:
KNN Accuracy: 0.8451300161583182
Naive Bayes Accuracy: 0.7590911362462728
Decision Tree Accuracy: 0.8563242324798854
SVM Accuracy: 0.6924870897884391
```

Conclusion:

In the comparison of classifiers, the **Decision Tree** model achieved the **highest accuracy at 85.63%**, making it the best-performing model in this evaluation. The **K-Nearest Neighbors (KNN)** model followed closely with an accuracy of **84.51%**, demonstrating strong predictive performance as well. The **Naive Bayes** classifier achieved a moderate accuracy of **75.91%**, suggesting it may not capture the complexity of the data as effectively. The **Support Vector Machine (SVM)**, trained on a reduced dataset due to computational limitations, yielded the lowest accuracy at **69.25%**. Although SVM generally performs well with well-separated data, its performance here may be hindered by dataset reduction or lack of feature scaling. Overall, the Decision Tree appears to be the most suitable model for this task, balancing interpretability and performance effectively.

Experiment 7

Aim: To implement different clustering algorithms.

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

Theory:

Clustering Algorithms for Unsupervised Classification

Clustering is an unsupervised machine learning technique used to group similar data points based on certain features. Below are three widely used clustering algorithms:

1. K-Means Clustering

K-Means is a centroid-based clustering algorithm that partitions data into k clusters.

Steps of K-Means Algorithm:

1. Choose the number of clusters k.
2. Initialize k cluster centroids randomly.
3. Assign each data point to the nearest centroid based on Euclidean distance.
4. Compute the new centroids as the mean of all points in each cluster.
5. Repeat steps 3 and 4 until centroids no longer change or a stopping criterion is met.

Mathematical Steps:

- Compute the distance between a point x_i and centroid C_j:

$$d(x_i, C_j) = \sqrt{\sum_{d=1}^{n} (x_{id} - C_{jd})^2}$$

- Update centroid:

$$C_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

where S_j is the set of points assigned to cluster

2. DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN is a density-based clustering algorithm that groups points that are closely packed together while marking outliers as noise.

Steps of DBSCAN Algorithm:

1. Select a random point P and check if it has at least MinPts neighbors within radius ε.
2. If yes, create a new cluster and expand it by adding density-reachable points.
3. If no, mark P as noise.
4. Repeat until all points are processed.

Mathematical Concepts:

- A point P is a **core point** if it has at least MinPts neighbors within ε.
- A point Q is **density-reachable** from P if d(P,Q)≤ε.
- A point is **noise** if it does not belong to any cluster.

3. Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters using either **Agglomerative (bottom-up)** or **Divisive (top-down)** approaches.

Steps of Agglomerative Clustering (Bottom-Up Approach):

1. Treat each data point as its own cluster.
2. Compute the distance between all pairs of clusters.
3. Merge the two closest clusters.
4. Repeat steps 2-3 until one cluster remains.

Mathematical Concepts:

- **Single linkage:**

$$d(A,B) = \min_{a \in A, b \in B} d(a,b)$$

- **Complete linkage:**

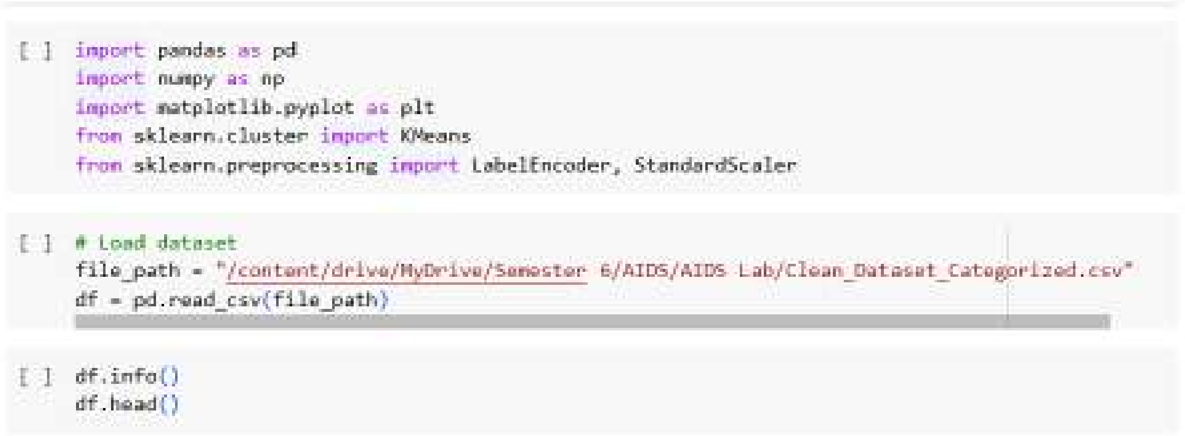
$$d(A,B) = \max_{a \in A, b \in B} d(a,b)$$

- **Average linkage:**

$$d(A,B) = \frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} d(a,b)$$

Steps :

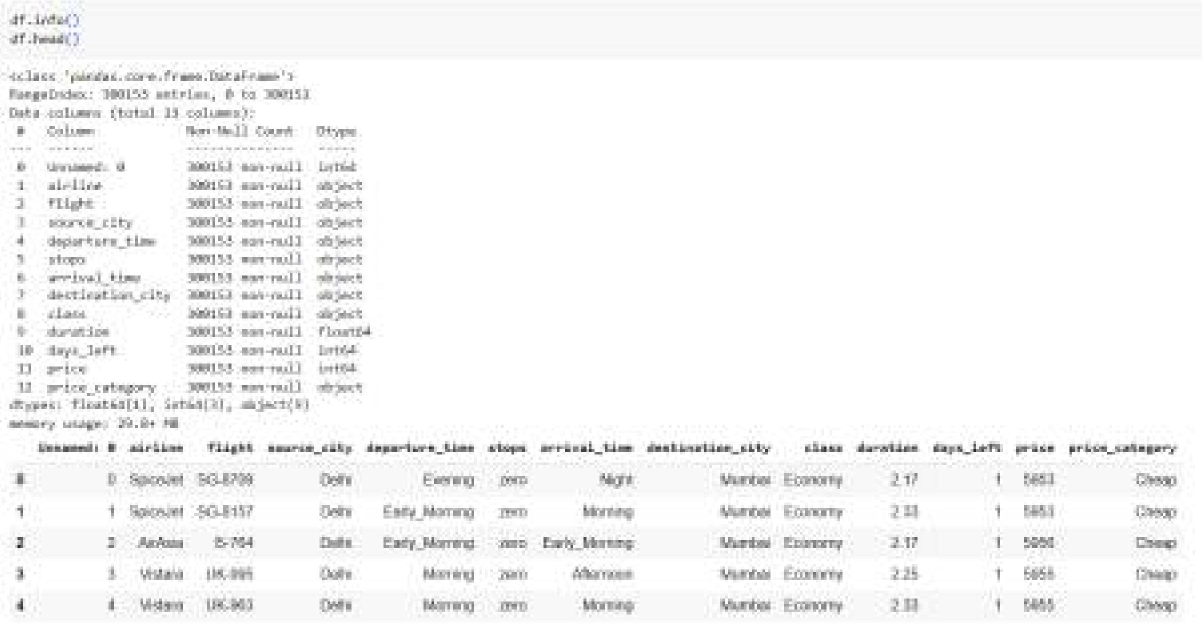
1) **Load and Explore Data**



```
[ ] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
[ ] # Load dataset
    file_path = "/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Clean_Dataset_Categorized.csv"
    df = pd.read_csv(file_path)
```

```
[ ] df.info()
    df.head()
```



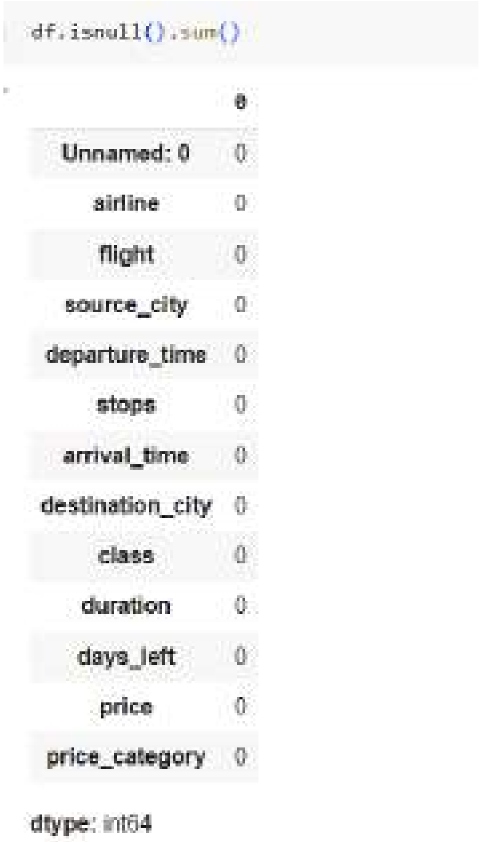
```
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300153 entries, 0 to 300152
Data columns (total 13 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Unnamed: 0        300153 non-null  int64
 1   airline           300153 non-null  object
 2   flight            300153 non-null  object
 3   source_city       300153 non-null  object
 4   departure_time    300153 non-null  object
 5   stops             300153 non-null  object
 6   arrival_time      300153 non-null  object
 7   destination_city  300153 non-null  object
 8   class             300153 non-null  object
 9   duration          300153 non-null  float64
 10  days_left         300153 non-null  int64
 11  price             300153 non-null  int64
 12  price_category    300153 non-null  object
dtypes: float64(1), int64(3), object(9)
memory usage: 29.8+ MB
```

	Unnamed: 0	airline	flight	source_city	departure_time	stops	arrival_time	destination_city	class	duration	days_left	price	price_category
0	0	SpiceJet	SG-8709	Delhi	Evening	zero	Night	Mumbai	Economy	2.17	1	5953	Cheap
1	1	SpiceJet	SG-8157	Delhi	Early_Morning	zero	Morning	Mumbai	Economy	2.33	1	5953	Cheap
2	2	AirAsia	I5-764	Delhi	Early_Morning	zero	Early_Morning	Mumbai	Economy	2.17	1	5956	Cheap
3	3	Vistara	UK-995	Delhi	Morning	zero	Afternoon	Mumbai	Economy	2.25	1	5955	Cheap
4	4	Vistara	UK-963	Delhi	Morning	zero	Morning	Mumbai	Economy	2.33	1	5955	Cheap

In this step, the cleaned flight fare dataset containing 300,153 entries was loaded and explored. Each entry includes details like airline, source and destination cities, departure/arrival times, flight duration, and price.

2) Printing Missing Values

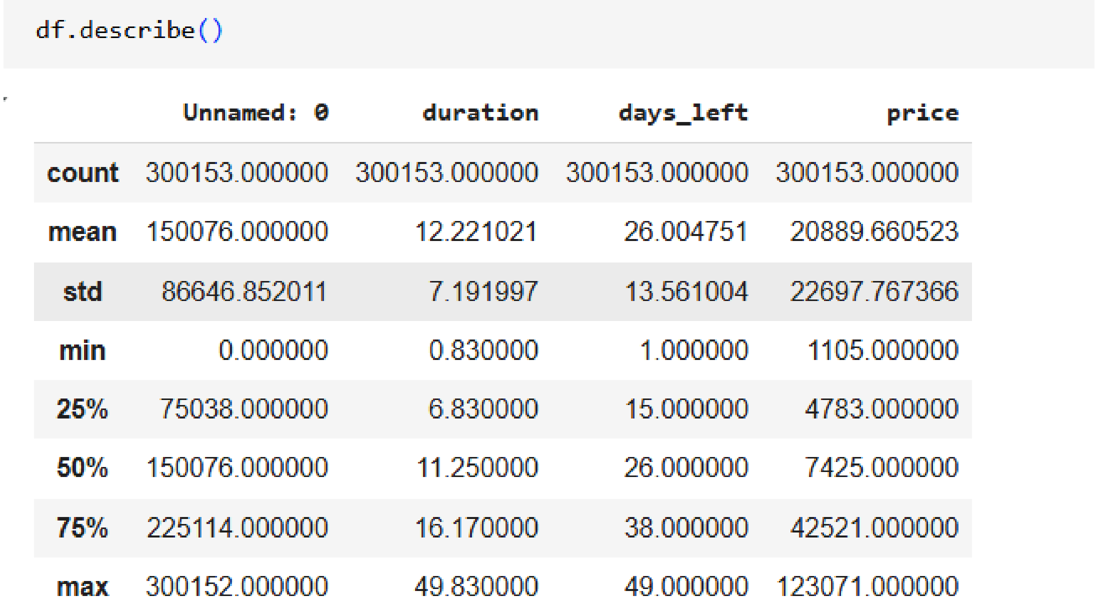


```
df.isnull().sum()
```

	0
Unnamed: 0	0
airline	0
flight	0
source_city	0
departure_time	0
stops	0
arrival_time	0
destination_city	0
class	0
duration	0
days_left	0
price	0
price_category	0

dtype: int64

No missing values were found across the 13 columns.

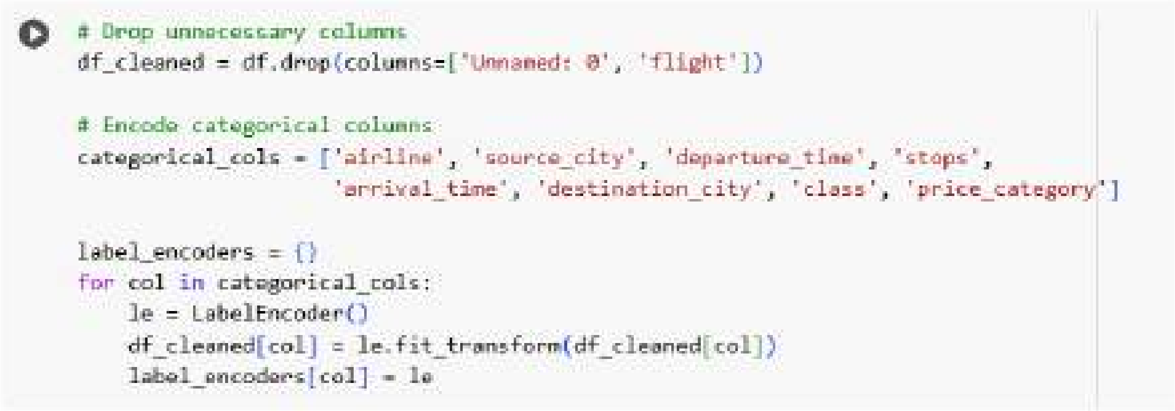


```
df.describe()
```

	Unnamed: 0	duration	days_left	price
count	300153.000000	300153.000000	300153.000000	300153.000000
mean	150076.000000	12.221021	26.004751	20889.660523
std	86646.852011	7.191997	13.561004	22697.767366
min	0.000000	0.830000	1.000000	1105.000000
25%	75038.000000	6.830000	15.000000	4783.000000
50%	150076.000000	11.250000	26.000000	7425.000000
75%	225114.000000	16.170000	38.000000	42521.000000
max	300152.000000	49.830000	49.000000	123071.000000

3) Data Cleaning and Preprocessing

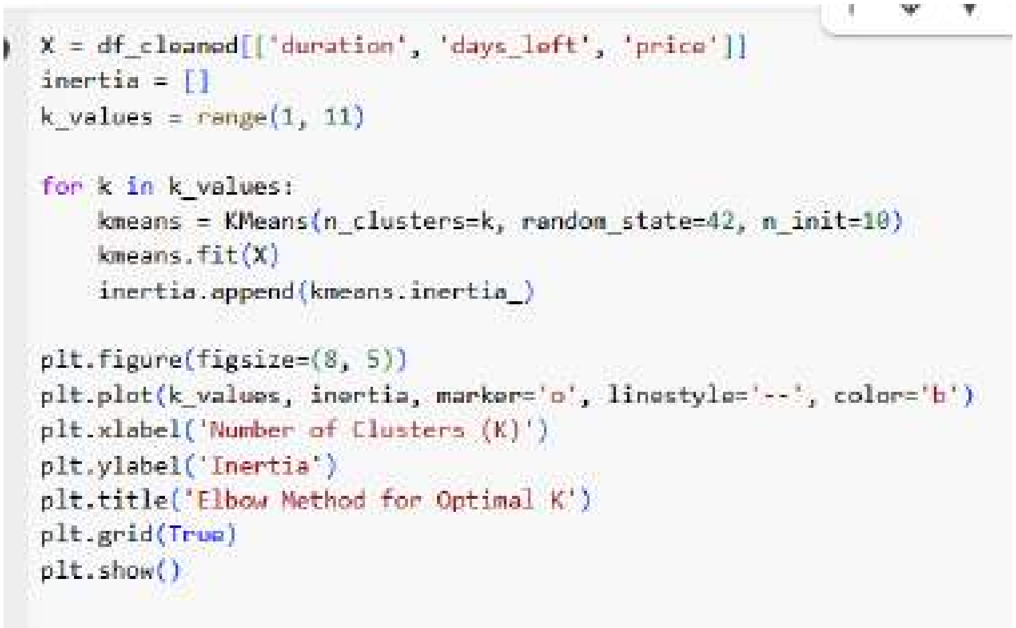
Drop Unnecessary and Convert categorical data



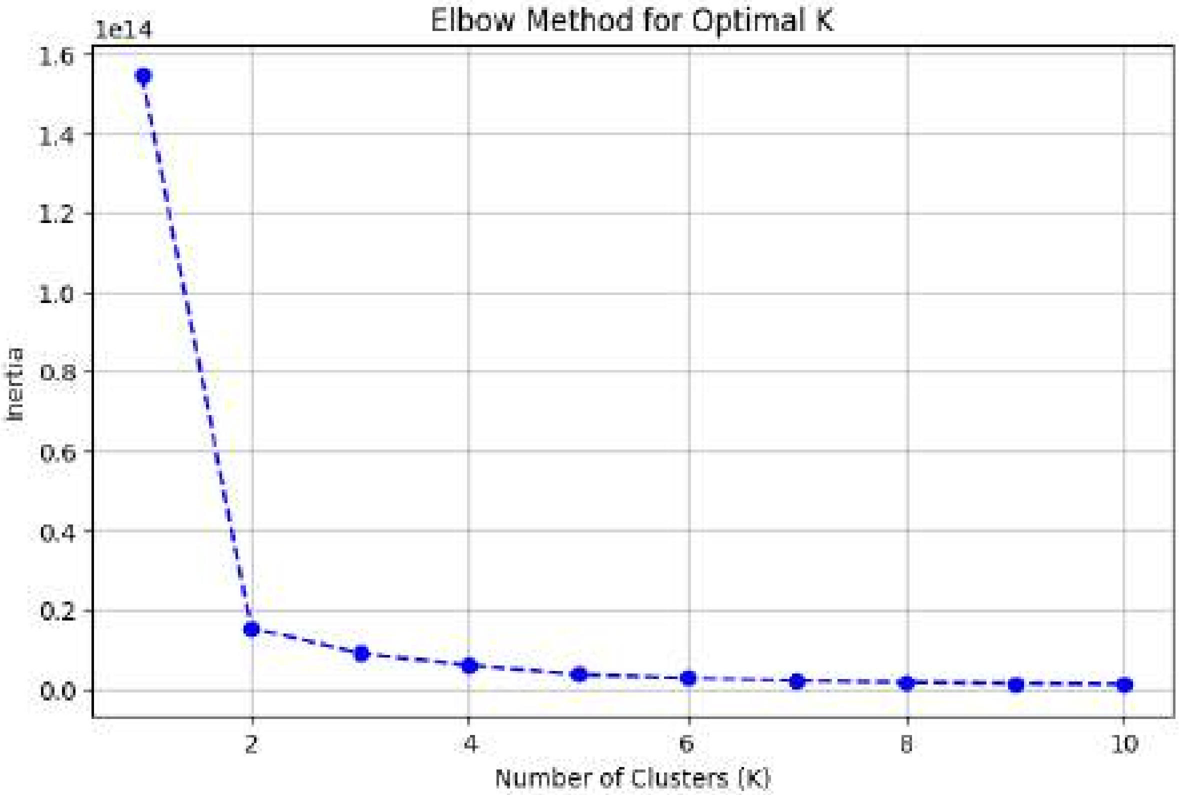
```
# Drop unnecessary columns
df_cleaned = df.drop(columns=['Unnamed: 0', 'flight'])

# Encode categorical columns
categorical_cols = ['airline', 'source_city', 'departure_time', 'stops',
                    'arrival_time', 'destination_city', 'class', 'price_category']

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

Unnecessary columns like 'Unnamed: 0' and 'flight' were dropped to simplify the dataset. Then, the categorical columns are encoded using Label Encoding to convert them into numerical format suitable for machine learning models. This step ensured that the dataset was clean, compact, and fully numerical, preparing it for clustering and further analysis.

4) K-Means Clustering

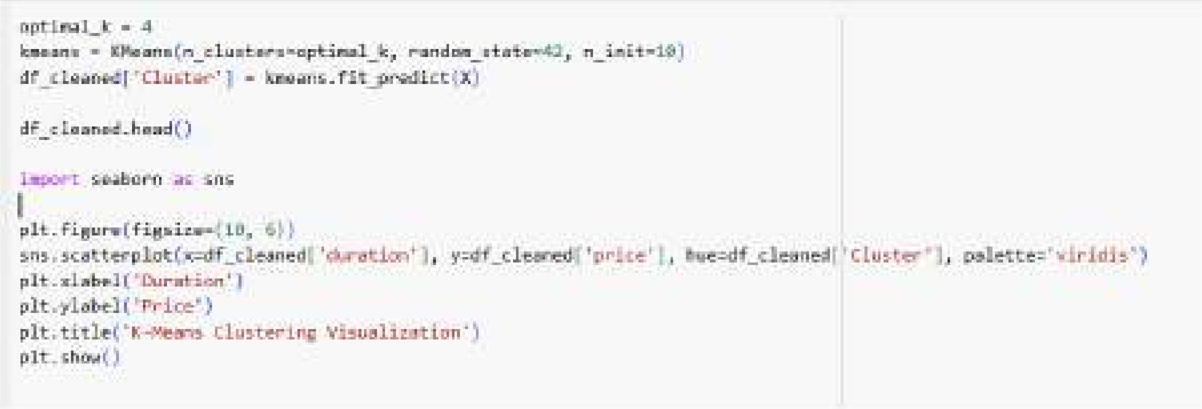


```
X = df_cleaned[['duration', 'days_left', 'price']]
inertia = []
k_values = range(1, 11)

for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(k_values, inertia, marker='o', linestyle='--', color='b')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()
```

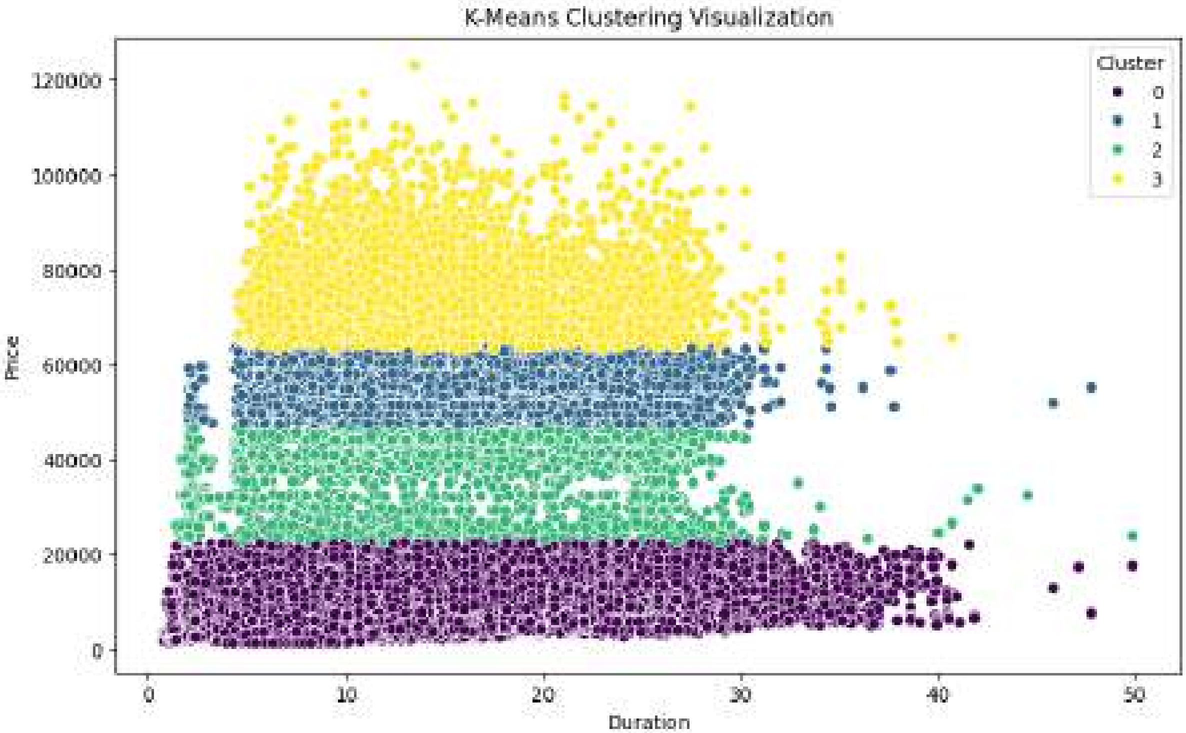


K-Means clustering was applied using the numerical features 'duration', 'days_left', and 'price'. To determine the optimal number of clusters (K), the Elbow Method was used by plotting inertia values for K ranging from 1 to 10. The elbow point in the curve appears around **K=4**, indicating that 4 clusters provide a good balance between model complexity and performance. This step is crucial for identifying distinct flight pricing patterns.

Code to Apply K-Means

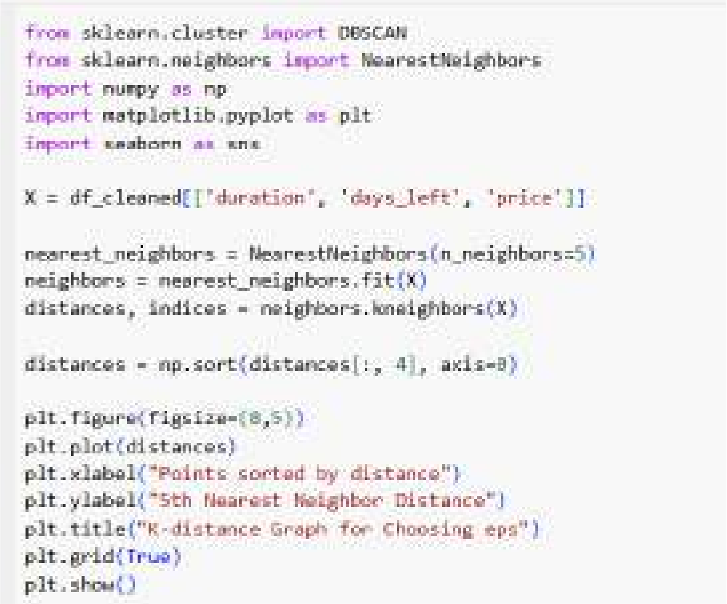


```
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
df_cleaned['Cluster'] = kmeans.fit_predict(X)

df_cleaned.head()

import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_cleaned['duration'], y=df_cleaned['price'], hue=df_cleaned['Cluster'], palette='viridis')
plt.xlabel('Duration')
plt.ylabel('Price')
plt.title('K-Means Clustering Visualization')
plt.show()
```

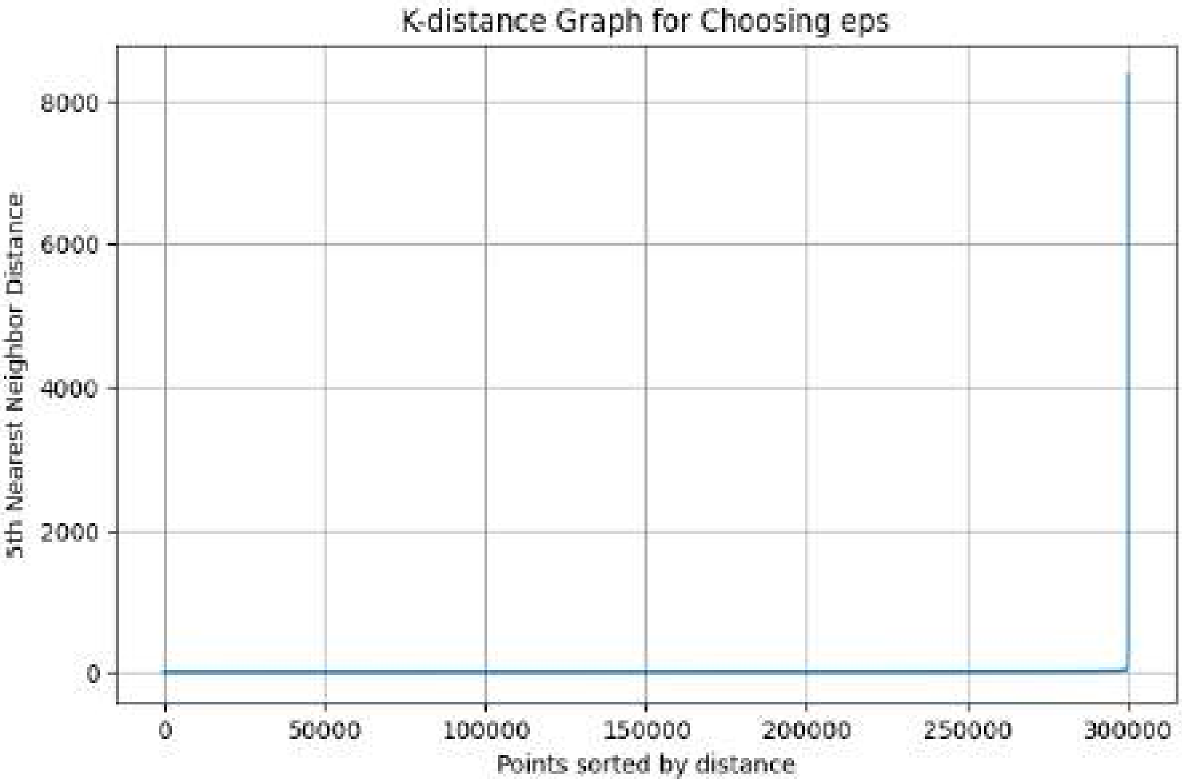


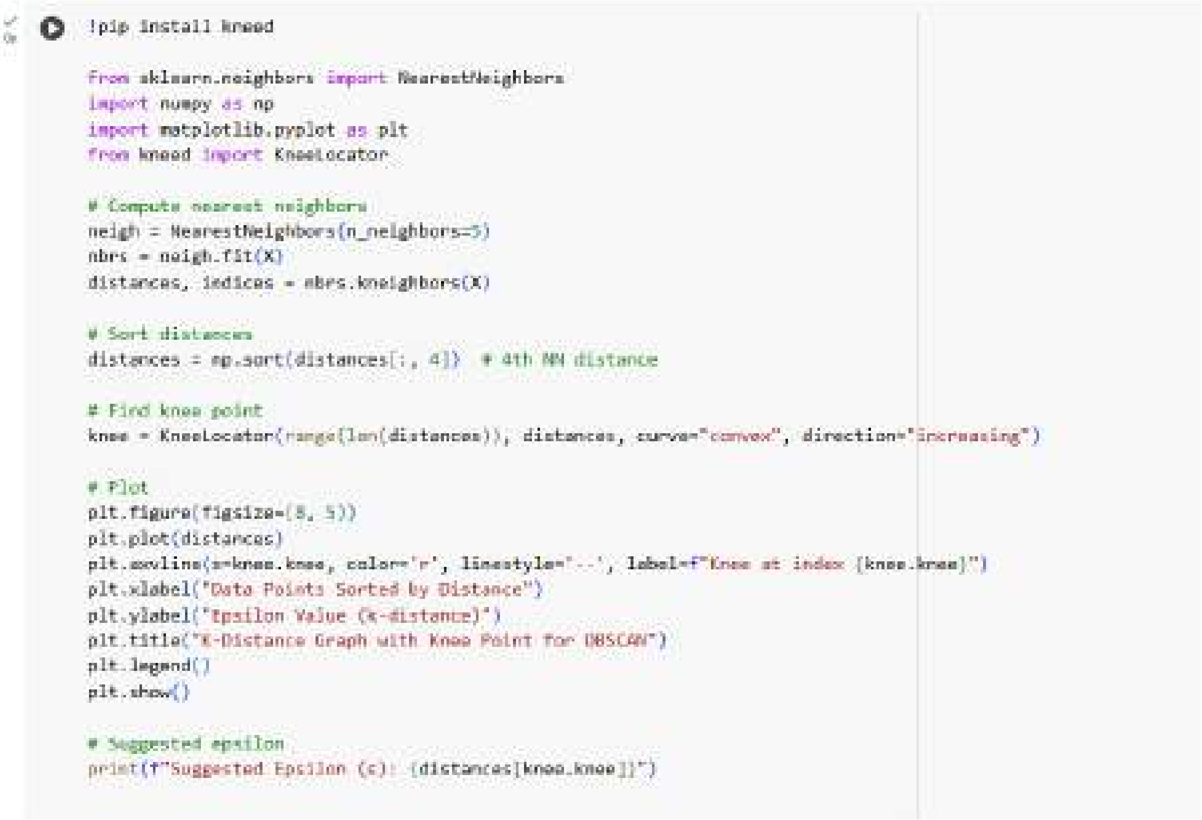
K-Means was applied with K=4 (from the elbow method) to segment flights based on duration, price, and days left. Each flight was assigned a cluster label, and the results were visualized using a scatter plot. The plot reveals four distinct price-based groupings, showing clear patterns in how flight duration and price correlate. This clustering can help identify trends in fare segmentation and assist in price prediction or customer targeting.

5) DBSCAN Clustering

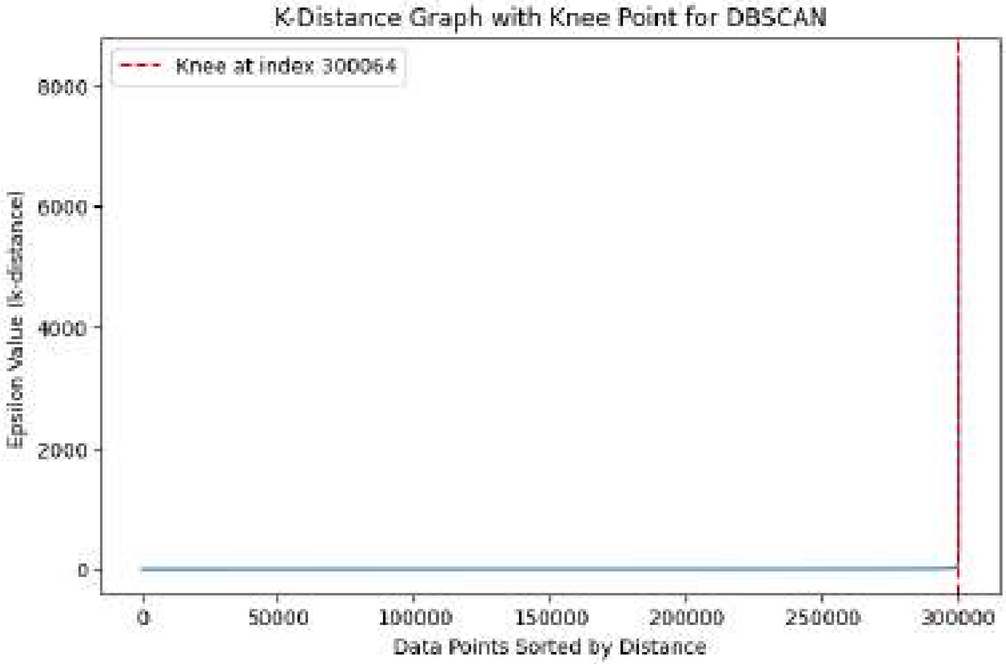


```
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

X = df_cleaned[['duration', 'days_left', 'price']]

nearest_neighbors = NearestNeighbors(n_neighbors=5)
neighbors = nearest_neighbors.fit(X)
distances, indices = neighbors.kneighbors(X)

distances = np.sort(distances[:, 4], axis=0)

plt.figure(figsize=(8,5))
plt.plot(distances)
plt.xlabel("Points sorted by distance")
plt.ylabel("5th Nearest Neighbor Distance")
plt.title("K-distance Graph for Choosing eps")
plt.grid(True)
plt.show()
```

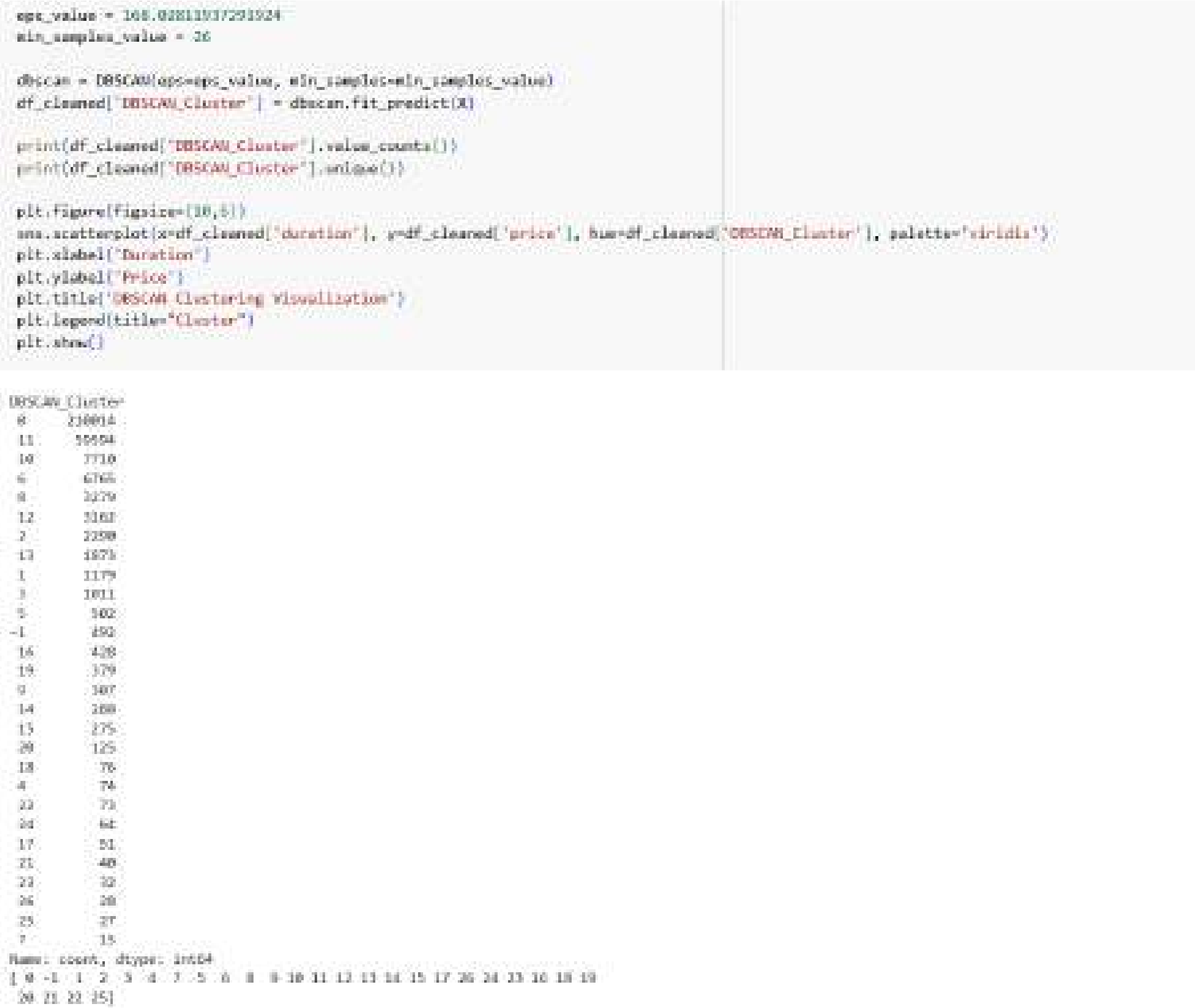




```
!pip install kneed

from sklearn.neighbors import NearestNeighbors
import numpy as np
import matplotlib.pyplot as plt
from kneed import KneeLocator

# Compute nearest neighbors
neigh = NearestNeighbors(n_neighbors=5)
nbrs = neigh.fit(X)
distances, indices = nbrs.kneighbors(X)

# Sort distances
distances = np.sort(distances[:, 4])  # 4th NN distance

# Find knee point
knee = KneeLocator(range(len(distances)), distances, curve="convex", direction="increasing")

# Plot
plt.figure(figsize=(8, 5))
plt.plot(distances)
plt.axvline(x=knee.knee, color='r', linestyle='--', label=f"Knee at index {knee.knee}")
plt.xlabel("Data Points Sorted by Distance")
plt.ylabel("Epsilon Value (k-distance)")
plt.title("K-Distance Graph with Knee Point for DBSCAN")
plt.legend()
plt.show()

# Suggested epsilon
print(f"Suggested Epsilon (ε): {distances[knee.knee]}")
```

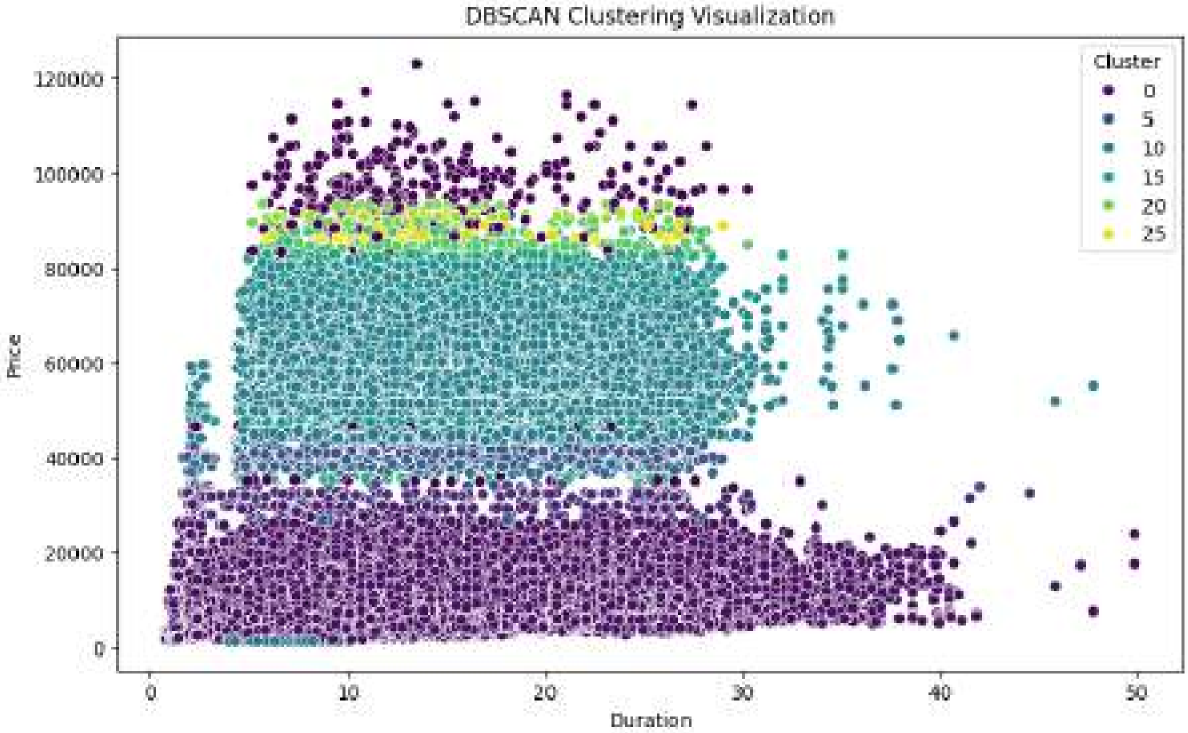


Suggested Epsilon (ε): 168.02811937291924



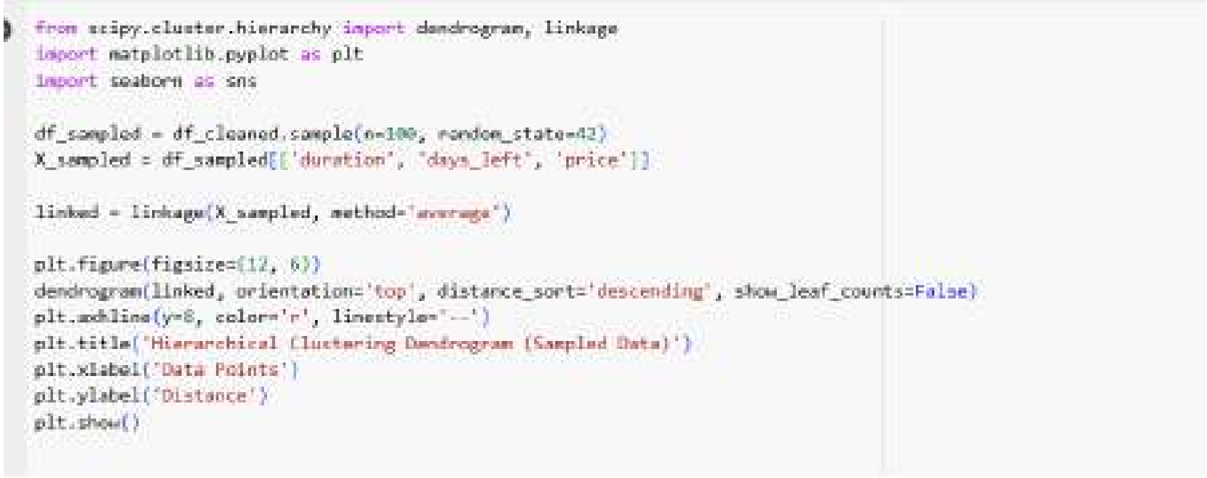
```
eps_value = 168.02813537291924
min_samples_value = 26

dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value)
df_cleaned['DBSCAN_Cluster'] = dbscan.fit_predict(X)

print(df_cleaned['DBSCAN_Cluster'].value_counts())
print(df_cleaned['DBSCAN_Cluster'].unique())

plt.figure(figsize=(10,5))
sns.scatterplot(x=df_cleaned['duration'], y=df_cleaned['price'], hue=df_cleaned['DBSCAN_Cluster'], palette='viridis')
plt.xlabel('Duration')
plt.ylabel('Price')
plt.title('DBSCAN Clustering Visuallization')
plt.legend(title="Cluster")
plt.show()
```

```
DBSCAN_Cluster
 0     210014
 11     59554
 10      7710
 6       6765
 8       3279
 12      3162
 2       2250
 13      1875
 1       1179
 3       1011
 5        502
-1        492
 16       428
 19       379
 9        307
 14       288
 15       275
 20       125
 18        76
 4         74
 23        73
 24        64
 17        51
 21        48
 22        32
 26        28
 25        27
 7         15
Name: count, dtype: int64
[ 0 -1  1  2  3  4  7  5  6  8  9 10 11 12 13 14 15 17 26 24 23 16 18 19
 20 21 22 25]
```

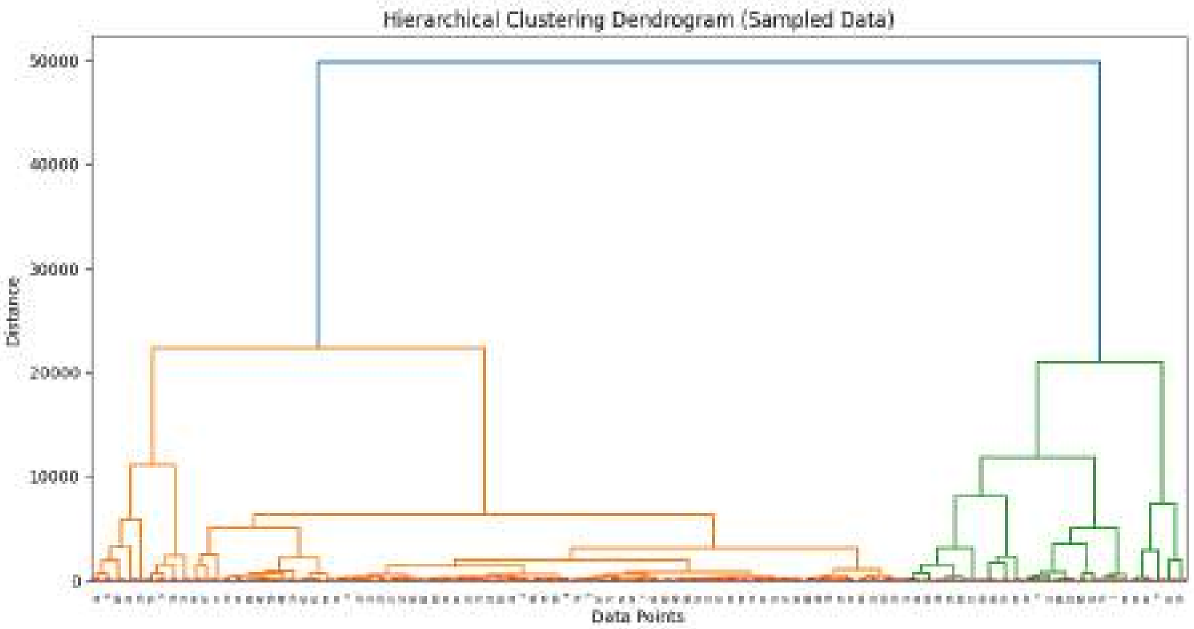


DBSCAN clustering was applied using an optimal epsilon value of **168.03**, determined from the k-distance graph. With min_samples set to 26, the algorithm detected **27 clusters**, including a few outliers labeled as **-1**. Most data points were grouped into dense clusters, clearly visible in the visualization using 'duration' and 'price'. The method effectively identified natural groupings in the data without needing the number of clusters in advance.

6) Hierarchical Clustering



```
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
import seaborn as sns

df_sampled = df_cleaned.sample(n=100, random_state=42)
X_sampled = df_sampled[['duration', 'days_left', 'price']]

linked = linkage(X_sampled, method='average')

plt.figure(figsize=(12, 6))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=False)
plt.axhline(y=8, color='r', linestyle='--')
plt.title('Hierarchical Clustering Dendrogram (Sampled Data)')
plt.xlabel('Data Points')
plt.ylabel('Distance')
plt.show()
```



Hierarchical clustering was applied to a random sample of 100 flight records using average linkage. The dendrogram shows how data points are progressively merged based on their similarity in duration, price, and days left. A horizontal cut at distance = 8 (red line) suggests the presence of **3–4 optimal clusters**, supporting the K-Means result. This technique provides a visual understanding of cluster hierarchy and the similarity between data points.

Conclusion:

In this project, we implemented and analyzed three clustering algorithms—K-Means, DBSCAN, and Hierarchical Clustering—for unsupervised classification, each applied to a cleaned dataset and visualized through scatter plots and dendrograms. K-Means efficiently grouped data based on centroid proximity but required predefined cluster numbers, while Hierarchical Clustering revealed nested structures through dendrograms, offering deeper insights into data hierarchy. DBSCAN excelled in identifying clusters of varying densities and detecting outliers without prior assumptions about cluster count. The comparison highlighted K-Means' simplicity and speed, Hierarchical Clustering's interpretability, and DBSCAN's robustness to noise, collectively demonstrating clustering's effectiveness in uncovering hidden patterns within unlabeled data.

Sairam Konar D15C 27 2024-2025

Experiment 8

Aim: To implement a recommendation system on your dataset using the following machine learning techniques.

Theory:

Types of Recommendation Systems:

Recommendation systems are generally categorized into three main types: content-based filtering, collaborative filtering, and hybrid approaches.

Content-based filtering makes recommendations by analyzing the characteristics of items a user has previously interacted with or liked. For instance, if a user frequently watches romantic comedies, the system might suggest movies with similar genres, actors, or directors.

Collaborative filtering, on the other hand, leverages the preferences of multiple users to generate recommendations. In user-based collaborative filtering, users with similar preferences are identified, and items liked by one user are recommended to others with similar tastes. In item-based collaborative filtering, items that are often liked together by many users are recommended based on their co-occurrence patterns.

Hybrid recommendation systems combine both content-based and collaborative filtering techniques. These systems aim to capitalize on the advantages of each method while addressing common challenges like the cold start problem, which arises when there is insufficient data about new users or items, reducing recommendation accuracy.

Evaluation Measures for Recommendation Systems:

To evaluate the performance of a recommendation system, several metrics are commonly used:

Root Mean Square Error (RMSE) measures the difference between predicted ratings and actual user ratings. A lower RMSE indicates better predictive accuracy.

Precision@K evaluates the proportion of relevant items among the top K recommendations, focusing on the system's ability to recommend high-quality items.

Recall@K measures the proportion of all relevant items that appear in the top K results, reflecting the system's completeness in capturing user preferences.

F1-score is the harmonic mean of precision and recall, offering a balanced assessment when both metrics are critical.These evaluation metrics play a crucial role in comparing different recommendation models and in optimizing them for enhanced performance.

Steps :

1) **Import Required Libraries**

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from surprise import SVD, Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy
```

This code snippet imports all the essential libraries needed to build a recommendation system that combines clustering and collaborative filtering techniques. Pandas and NumPy are used for efficient data manipulation and numerical computations, while Matplotlib.pyplot supports the visualization of data and results. From the scikit-learn library, KMeans is employed for clustering users or items based on similarities, and StandardScaler is used to normalize the data for better clustering performance. The Surprise library, which is specifically designed for recommendation systems, is utilized to implement the SVD (Singular Value Decomposition) algorithm. It also provides helpful tools such as Dataset, Reader, train_test_split, and accuracy to load and preprocess the dataset, split it into training and testing sets, and evaluate the model's performance. Overall, this collection of imports forms the backbone for developing, analyzing, and visualizing a recommendation system using both clustering and collaborative filtering methods.

2) **Load the Dataset**

Code:

```
anime = pd.read_csv('/content/drive/MyDrive/anime.csv')
ratings = pd.read_csv('/content/drive/MyDrive/rating.csv')
```

This code loads two CSV files from your Google Drive into pandas DataFrames:

- anime.csv → Contains information about anime shows (like title, genre, type, rating, etc.).
- rating.csv → Contains user ratings for those anime (user ID, anime ID, and the score given).

These two datasets will be used together to build the recommendation system — one for the content metadata and the other for user interaction data.

3) Data Cleaning

Code:

```
anime.dropna(inplace=True)
anime = anime[anime['genre'] != 'Unknown']
ratings = ratings[ratings['rating'] != -1]
```

This code snippet performs essential data cleaning to ensure that only relevant and meaningful data is used to build the recommendation system. It begins by removing rows with missing values from the anime dataset using dropna(), which helps prevent errors during model training. It then filters out entries with the genre labeled as 'Unknown', since such entries lack informative value for generating recommendations. Additionally, it cleans the ratings dataset by eliminating records where the rating is -1, as these typically represent unrated entries that do not contribute to understanding user preferences. Together, these cleaning steps refine the dataset and enhance the overall performance of the recommendation model.

4) Merging Anime Metadata with Ratings

Code:

```
merged_df = ratings.merge(anime, on='anime_id')
```

This step performs an inner join between the ratings and anime DataFrames using anime_id as the common key. The resulting DataFrame, merged_df, combines user rating information with relevant anime metadata such as titles and genres. Merging these datasets is a crucial step in building a recommendation system, as it links user preferences to specific anime attributes. This integration enables the model to generate more personalized and accurate recommendations based on both user behavior and item characteristics.

5) Clustering Anime Based on Popularity and Rating

Code:

```
anime_cluster = anime.copy()
anime_cluster = anime_cluster[anime_cluster['members'] > 0]
```

```
anime_cluster['rating'] = pd.to_numeric(anime_cluster['rating'], errors='coerce')
anime_cluster.dropna(subset=['rating'], inplace=True)

features = anime_cluster[['rating', 'members']]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

kmeans = KMeans(n_clusters=5, random_state=42, n_init='auto')
anime_cluster['cluster'] = kmeans.fit_predict(scaled_features)
```

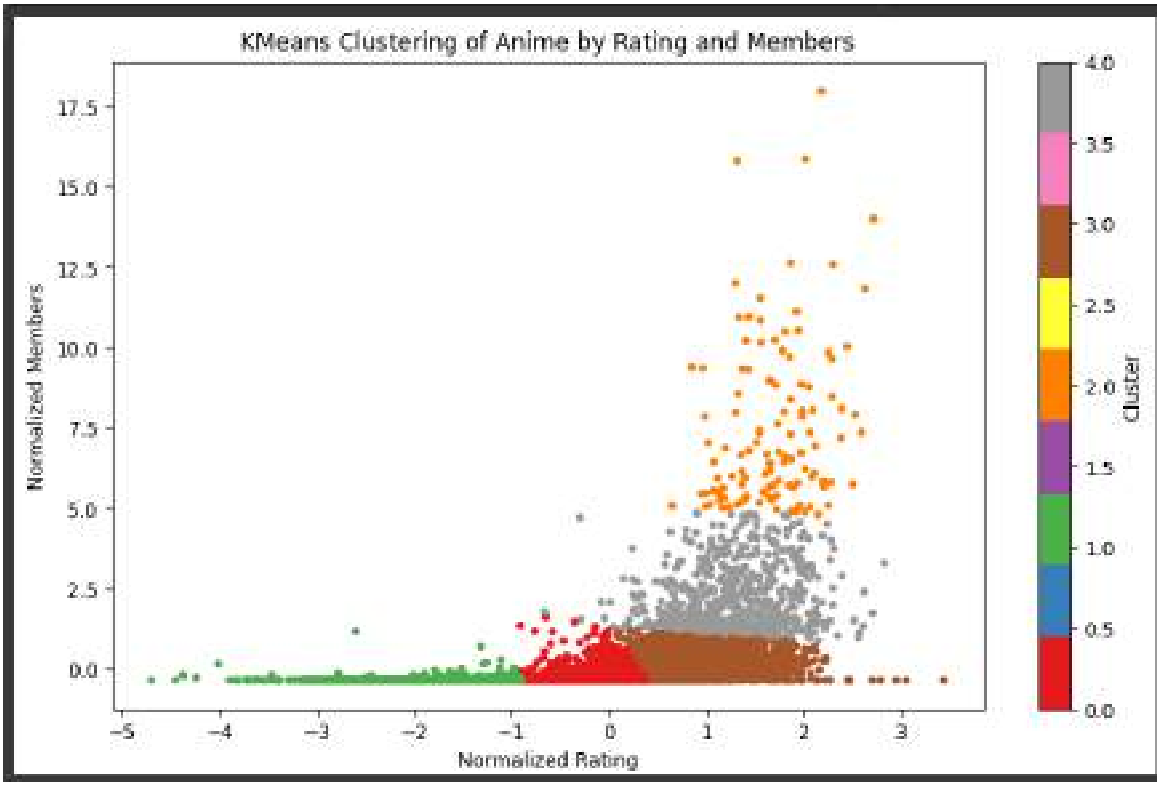
In this step, a copy of the original anime dataset is created to prepare it for clustering. The data is first filtered to include only anime entries with more than zero members, ensuring that only titles with some user engagement are considered. The rating column is then converted to a numeric format, with any non-convertible values handled gracefully. Rows with missing ratings are subsequently removed to maintain clean and reliable input for clustering. The clustering process uses two key features: the average user rating and the number of members. These features are standardized using StandardScaler to ensure they are on the same scale—an essential step for accurate clustering. Finally, the KMeans algorithm is applied to group the anime into five distinct clusters. Each anime is assigned a cluster label based on its similarity in terms of rating and popularity, enabling deeper analysis or visualization of anime with similar characteristics.

6) Visualizing Clusters of Anime

Code:

```
plt.figure(figsize=(10,6))
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=anime_cluster['cluster'],
cmap='Set1', s=10)
plt.title('KMeans Clustering of Anime by Rating and Members')
plt.xlabel('Normalized Rating')
plt.ylabel('Normalized Members')
plt.colorbar(label='Cluster')
plt.show()
```

Output:



This step creates a scatter plot to visually interpret the clusters formed by the KMeans algorithm. In the plot, the x-axis represents the normalized average ratings, while the y-axis shows the normalized number of members for each anime. Each point corresponds to an individual anime title, and its color indicates the cluster to which it has been assigned. The colormap cmap='Set1' is used to assign distinct and easily distinguishable colors to each cluster, enhancing visual clarity. A colorbar is included to map each color to its corresponding cluster label. This visualization helps reveal how anime titles are grouped based on similarities in user ratings and popularity, offering valuable insights into audience preferences and content trends.

7) Building and Training the SVD-Based Recommendation Model

Code:

```
reader = Reader(rating_scale=(1, 10))
data = Dataset.load_from_df(ratings[['user_id', 'anime_id', 'rating']], reader)
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

model = SVD()
model.fit(trainset)
```

```
predictions = model.test(testset)
```

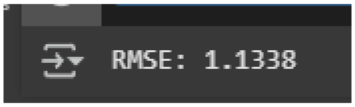
In this step, a recommendation system is constructed using the Singular Value Decomposition (SVD) algorithm from the Surprise library. A Reader object is first defined to specify the rating scale, which ranges from 1 to 10. The Dataset.load_from_df() function is then used to convert the cleaned ratings data into a format compatible with the Surprise framework. The dataset is split into training and test sets using an 80-20 ratio to facilitate model evaluation. The SVD model is trained on the training data to learn latent factors that capture underlying patterns in user preferences and anime characteristics. Once trained, the model is used to predict ratings on the test set. This process lays the groundwork for generating personalized anime recommendations through collaborative filtering.

8) Model Evaluation using RMSE

Code:

```
rmse = accuracy.rmse(predictions)
```

Output:



```
RMSE: 1.1338
```

In this step, the performance of the recommendation model is evaluated using the Root Mean Squared Error (RMSE). The obtained RMSE value of 1.1338 reflects the average deviation between the actual user ratings and the predictions made by the SVD model. Given that the rating scale ranges from 1 to 10, an RMSE around 1.1 is considered reasonably acceptable. This indicates that the model is fairly accurate in its predictions and can be relied upon to provide meaningful and personalized anime recommendations to users.

9) Function to Generate Top-N Anime Recommendations for a User

Code:

```
def get_top_n_recommendations(user_id, anime_df, ratings_df, model, n=10):
    all_anime_ids = anime_df['anime_id'].unique()
    rated_anime_ids = ratings_df[ratings_df['user_id'] == user_id]['anime_id'].unique()
    unseen_anime_ids = [aid for aid in all_anime_ids if aid not in rated_anime_ids]
    predictions = [model.predict(user_id, aid) for aid in unseen_anime_ids]
    top_predictions = sorted(predictions, key=lambda x: x.est, reverse=True)[:n]
    top_anime_ids = [pred.iid for pred in top_predictions]
        recommendations = anime_df[anime_df['anime_id'].isin(top_anime_ids)][['name',
'genre', 'type', 'rating']]
```

```
    return recommendations
```

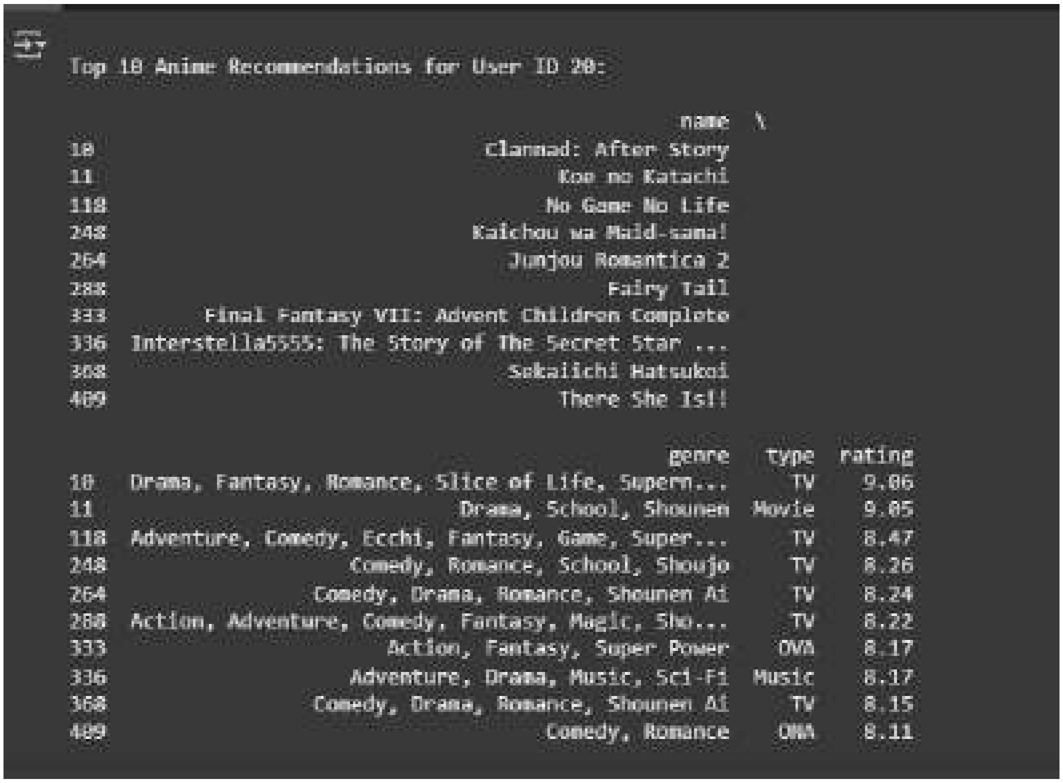
This function is designed to provide personalized anime recommendations for a specific user based on the trained recommendation model. It first retrieves all available anime IDs and filters out the ones the user has already rated. It then uses the model to predict ratings for the unseen anime and selects the top N (default 10) highest-rated predictions. The function finally returns detailed information (name, genre, type, and rating) about these top recommendations. This allows the system to generate relevant suggestions tailored to the user's interests.

10) Displaying Top 10 Anime Recommendations for a Specific User

Code:

```
user_id_sample =20
print(f"\nTop 10 Anime Recommendations for User ID {user_id_sample}:\n")
print(get_top_n_recommendations(user_id_sample, anime, ratings, model))
```

Output:



```
Top 10 Anime Recommendations for User ID 20:

                                                  name  \
10                                Clannad: After Story   
11                                      Koe no Katachi   
118                                    No Game No Life   
248                              Kaichou wa Maid-sama!   
264                                 Junjou Romantica 2   
288                                         Fairy Tail   
333        Final Fantasy VII: Advent Children Complete   
336  Interstella5555: The Story of The Secret Star ...   
368                                 Sekaiichi Hatsukoi   
409                                     There She Is!!   

                                                 genre   type  rating  
10   Drama, Fantasy, Romance, Slice of Life, Supern...     TV    9.06  
11                              Drama, School, Shounen  Movie    9.05  
118  Adventure, Comedy, Ecchi, Fantasy, Game, Super...     TV    8.47  
248                     Comedy, Romance, School, Shoujo     TV    8.26  
264                 Comedy, Drama, Romance, Shounen Ai     TV    8.24  
288  Action, Adventure, Comedy, Fantasy, Magic, Sho...     TV    8.22  
333                       Action, Fantasy, Super Power    OVA    8.17  
336                    Adventure, Drama, Music, Sci-Fi  Music    8.17  
368                 Comedy, Drama, Romance, Shounen Ai     TV    8.15  
409                                    Comedy, Romance    ONA    8.11  
```

In this step, the system fetches and prints the top 10 anime recommendations for a user with user_id 20. By calling the get_top_n_recommendations() function and passing in the required data and model, it displays a curated list of anime titles that the user has not yet rated but is likely to enjoy based on the trained SVD model. This output demonstrates how the recommendation system can be personalized for individual users and showcases the system’s practical usage in making intelligent suggestions.

Conclusion:

In this experiment, we developed a hybrid recommendation system by integrating clustering and collaborative filtering techniques. K-Means was employed to group similar anime based on features such as average rating and popularity, enabling content-based insights. Simultaneously, the SVD algorithm was used to predict user ratings for unseen anime, capturing collaborative filtering patterns. The model achieved satisfactory accuracy and was able to generate personalized Top-N recommendations. This highlights the effectiveness of combining content-based and collaborative approaches to build intelligent and user-centric recommendation systems.

Sairam Konar D15C 27 2024-2025

Experiment 9

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

Theory:

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

Answer:

- **Introduction to Apache Spark**
  Apache Spark is an open-source, distributed computing framework designed for fast and scalable big data processing. Developed at UC Berkeley's AMPLab, Spark provides an optimized engine for large-scale data analytics, machine learning, and real-time processing. Unlike traditional single-node tools (like Pandas or Excel), Spark efficiently handles massive datasets by distributing computations across clusters.

- **Key Features of Apache Spark**
  - **Unified Analytics Engine:** Supports batch processing, real-time streaming, machine learning (MLlib), and graph processing (GraphX).
  - **In-Memory Processing:** Retains intermediate data in RAM, significantly speeding up iterative algorithms.
  - **Fault Tolerance:** Recovers lost data partitions automatically using lineage information.
  - **Multi-Language Support**: APIs available in Python (PySpark), Scala, Java, and R.
  - **Lazy Evaluation:** Optimizes execution plans by delaying computation until necessary.

- **How Apache Spark Works**
  Spark operates in a master-slave architecture, where:

  - **Driver Node (Master)**: Coordinates tasks, schedules jobs, and manages execution.
  - **Worker Nodes (Slaves)**: Perform distributed computations in parallel.

  Core Components:

  - **Resilient Distributed Dataset (RDD)**: Fundamental data structure in Spark.
    Immutable, partitioned collections processed in parallel.
    Supports transformations (e.g., map, filter) and actions (e.g., count, collect).
  - **DataFrames & Datasets:** Higher-level abstractions built on RDDs, optimized for structured/semi-structured data.
    Support SQL-like operations (e.g., groupBy, join).
  - **Spark SQL**: Enables querying structured data using SQL syntax.
  - **Spark Streaming:** Processes real-time data streams in micro-batches.

- **MLlib & GraphX:** Libraries for machine learning and graph processing.

- **Use Cases of Apache Spark:**
  - **Real-Time Analytics**: Fraud detection, IoT sensor monitoring.
  - **Large-Scale ETL:** Processing terabytes of log files.
  - **Machine Learning:** Training models on distributed datasets.
  - **Social Media Analysis:** Sentiment analysis, trend detection.
  - **Financial Data Processing**: Risk modeling, transaction analysis.

2) How is Data Exploration Done in Apache Spark?
Answer:

- **Initializing Spark Session**
  Before performing any operations in Spark, we need to create a SparkSession, which acts as the entry point for interacting with Spark's functionalities. This session configures the application name, cluster settings, and other parameters required for distributed computing.

- **Loading Data into Spark**
  Spark supports reading data from various sources (CSV, JSON, Parquet, databases, etc.). When loading data:
  - **Header:** Specifies if the first row contains column names.
  - **Infer Schema:** Automatically detects data types (e.g., integers, strings) or enforces a predefined schema.
  - **Partitioning:** Large datasets are split into chunks (partitions) for parallel processing.

- **Inspecting Data Structure**
  After loading, we examine:
  - **Schema:** Lists column names and their data types (e.g., StringType, IntegerType).
  - **Sample Records:** Displays the first few rows to understand the data layout.
  - **Summary Statistics:** Computes basic metrics (count, mean, min/max, standard deviation) for numerical columns.

- **Handling Missing Data**
  Missing values (null/NaN) can distort analysis. Common strategies:
  - **Dropping Rows**: Remove records with missing values (if they're insignificant).
  - **Imputation:** Replace nulls with mean/median (for numerical data) or mode (for categorical data).
  - **Flagging:** Mark missing values for further investigation.
- **Aggregations and Grouping**
  Spark allows:
  - **GroupBy:** Segregate data by categories (e.g., sales by region).

  - **Aggregate Functions:** Compute summaries like sum(), avg(), countDistinct().
  - **Pivoting:** Reshape data for cross-tabulation (e.g., sales per product per month).

- **Filtering and Sorting**
  - **Filtering:** Subset data based on conditions (e.g., transactions > $100).
  - **Sorting:** Order records by specific columns (ascending/descending).

- **Data Visualization (via Pandas Conversion)**
  Spark DataFrames lack built-in plotting, so we:

  - **Limit Data Size:** Convert a subset (e.g., 1000 rows) to a Pandas DataFrame.
  - **Use Matplotlib/Seaborn**: Generate histograms, scatter plots, or box plots for trends/outliers.

Conclusion:
This experiment aimed to understand how Exploratory Data Analysis (EDA) can be performed using both Apache Spark and Pandas. Spark, with its distributed and in-memory processing, is ideal for analyzing large-scale datasets, while Pandas is better suited for smaller, quick analysis tasks. We explored the step-by-step EDA process—loading, inspecting, cleaning, summarizing, and analyzing data. Each tool serves a specific purpose, and together they offer flexibility and efficiency across data sizes. Understanding how these tools work prepares data professionals to handle diverse analytical challenges with the right approach.

Sairam Konar D15C 27 2024-2025

Experiment 10

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

Theory:

1. What is Streaming? Explain Batch and Stream Data.
In data processing, we generally deal with two major types: Batch processing and Stream processing.
Batch Data Processing refers to collecting data over a period of time and then processing it all at once. Think of it like baking cookies: you prepare a whole batch and then put it in the oven. It's ideal when real-time insights aren't necessary.
Stream Data Processing (also known as real-time processing) handles continuously flowing data, such as sensor feeds, social media updates, or payment transactions. Instead of waiting for a complete dataset, stream processing processes each piece of data as it arrives.
Streaming is the process of analyzing data in motion. It's especially useful when immediate insights are crucial, such as in fraud detection, server monitoring, or stock market analysis.

2. How Does Data Streaming Take Place Using Apache Spark?
Apache Spark provides a module called Spark Streaming and its newer, more advanced version, Structured Streaming, for processing real-time data streams.

Working:

- **Data Source:** Data is continuously received from sources like Kafka, socket connections, or files being updated in real time.
- **Spark Streaming Engine**: Spark divides incoming data into small batches (micro-batches). Despite being a streaming method, it operates by processing these small batches at regular intervals.
- **Transformations and Actions:** Similar to batch mode, filtering, grouping, or aggregation logic can be applied to each micro-batch.
- **Output Sink**: Results are pushed to dashboards, databases, or alert systems for near-instant insights.

  The power of Spark lies in its unified programming model for both batch and stream processing, making it both flexible and efficient.

  Steps for Batch Data Analysis using Apache Spark
- **Start Apache Spark Environment:** Launch Apache Spark locally, on the cloud, or using a notebook interface like Jupyter or Databricks.

- **Read a Batch Dataset:** Load a static dataset (e.g., CSV or JSON) with a fixed structure.

- **Explore the Dataset:** Inspect columns, data types, and sample records.
- **Clean and Prepare the Data:** Handle missing values, rename columns, fix data types, and remove duplicates.
- **Perform Transformations and Aggregations:** Apply filtering, grouping, and compute metrics like averages or totals.
- **Store or Display Output:** Save results to files, visualize them, or print to console.

Steps for Streamed Data Analysis using Apache Spark

- **Initialize Spark with Structured Streaming:** Start a Spark session with structured streaming configuration.
- **Connect to a Streaming Data Source:** Connect to sources like Kafka, socket, or folders receiving new files.
- **Define the Schema for Streaming Data:** Manually specify schema fields like timestamp, value, etc.
- **Apply Streaming Operations:** Perform real-time filtering, windowed grouping, and aggregation.
- **Write Stream Output to a Sink:** Continuously write results to console, files, databases, or dashboards.
- **Monitor the Streaming Pipeline:** Track job status, data throughput, and resource usage during streaming.

Conclusion

This experiment demonstrates the power of Apache Spark in handling both batch and streamed data processing. Batch analysis is ideal for static historical data, while streamed analysis provides real-time insights for live data scenarios. Apache Spark’s unified architecture makes it an efficient and scalable solution for diverse data processing needs.

SAIRAM. V. KONAR
D15C
27

05/05

AIDS-I Assignment 1

Q.1] What is AI? Considering the covid-19 pandemic situation, how AI has helped to survive and renovate our own way of life with different applications?

⇒ Artificial Intelligence (AI) is the field of computer science that enables machines to structure, simulate human ~~responses~~ intelligence including learning, reasoning and decision-making. AI encompasses technologies such as ML, NLP, CV automate tasks

Role of AI in COVID-19

1] Early Detection and Diagnostic:- AI models detected outbreak early, and AI assisted CT scans helped in quick diagnosis.

2> Drug and Vaccine Development:- AI accelerated drug discovery and vaccine research.

3> Contact Tracing and safety: AI powered apps tracked virus spreading, thermal cameras detected fever.

4> Healthcare and Robotics: AI chatbots assisted with self-diagnostics, robots disinfected hospitals and delivered medicines

5> Remote Work and Education: AI improved video conferencing, virtual assistants and adaptive learning platforms.

Q.2] What is AI agent terminology, explain with example

⇒ An AI agent perceives its environment through sensors and acts using actuators to achieve goals. The key terminologies include.

Agent - An entity that percepts and acts.
Percept - Input received by the agent
Percept sequence - The history of all past percepts
Actuators - Components that allow agent to act.

Sensors - Devices collecting data

Environment - The surroundings where the agent operate

Eg:- Self Driving Car

Sensors	Percepts	Actuators	Environ
Camera, LiDAR, GPS	Detects speed limits traffic and obstacle	Accelerate, brake, steer	Road, vehi

Q) How AI technique is used to solve 8-puzzle problem

⇒ The 8-puzzle problem consists of a 3×3 grid with numbered tiles and 1 empty tile. The goal is to re a target arrangement by sliding tiles.

Various informed and uninformed technique are used such as BFS, DFS, IDDFS, A* etc.

For informed search, heuristics such as displaced tile or manhattan distance can be used.

Consider the following:-

Initial state :- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 6 & 8 \\ 7 & 5 & \_ \end{bmatrix}$ Goal state:- $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & - \end{bmatrix}$

The legal moves AI can make are shift blank to left, right, up, down.

Suppose the heuristic decided is 1 cost for move then the algorithm would look for least cost to goal state

State changes:-

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 6 & \_ \\ 7 & 5 & 8 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & - & 6 \\ 7 & 5 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & - & 8 \end{bmatrix} \rightarrow \underset{\substack{\uparrow \\ \text{Goal state} \\ \text{reached}}}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & - \end{bmatrix}}.$$

Q.4] What is PEAS description? Give it for the following
(i) Taxi Driver (ii) Medical Diagnosis System (iii) An aircraft autolander (iv) A music composer (v) An essay evaluator (vi) A realistic sentry for brick lab.

⇒ A PEAS description is a framework used to define the components of an intelligent agent. It helps describe how an AI system interact with its environment and performance tasks

	Performance	Environment	Actuators	Sensors
axi Driver	Safety, fuel efficiency, speed, comfort, traffic	Roads, traffic, pedestrians, passengers weather	steering, accelometer, brakes, indicators	GPS, cameras speed sensors, proximity sensor
dical agnosis system	Accuracy, Effective ness.	Patient records, medical symptoms, hospital database	Display diagnosis, suggesting treatment	Patient inputs, lab, test results.
rcraft utoloader	Safe lander, smooth tochdown, weather adoptation	Runway, weather conditions, wind speed altitude	Controlling thrust, flaps, landing gear, breaking system	Radar, altimeter, GPS, wind sensor, speedometer
music mposer	Creativity harmony, originality satisfaction	Musical notes, user preferences, genre	Generating melodies modifying pitch/tempo	Music database, user feedback, real time input based style
ssay valuator	Accuracy of grading fairness, grammer, spelling correctness	Essay, grammatic rules, plagarism database, rubrics.	Assigning grades, providing feedback, suggesting feedback	Text input, word count syntax and grammar checker.
	Intruder detection accuracy, response time minimal false alarms	laboratory area, authorized personnel intruders	Rotating turrent, firing warning shots, sound alarms.	Motion sensor. thermal cameras.

Teacher's Sign: ____________

Q.5] Categorize as shopping bot for an offline bookstore to each of the dimensions

⇒ (i) Observability :- Partially observable

(ii) Deterministic / stochastic :- Stoichastic

(iii) Episodic / sequential :- sequential

(iv) Static / Dynamic :- Dynamic

(v) Discrete / Continuous :- Discrete

(vi) single / multi agent :- Multi-agent

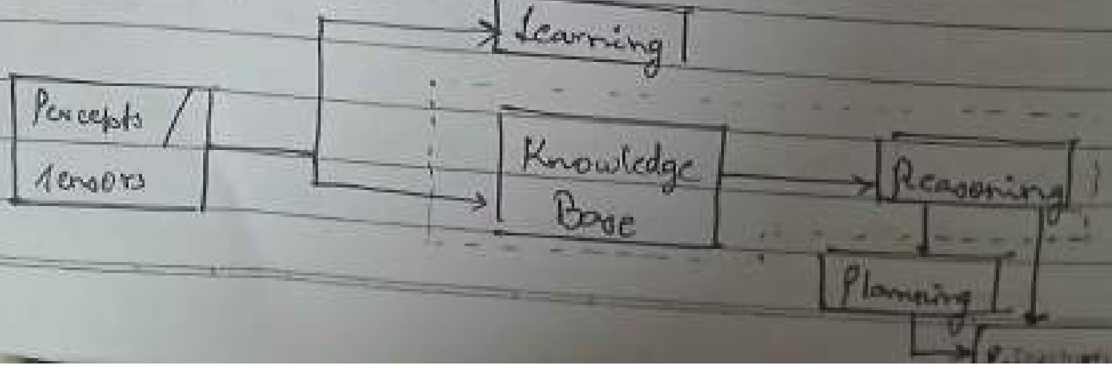
Q.6] Differentiate Model based and utility based Agent.

⇒

Model based agent	Utility based Agent
(i) Uses an internal model of the world	(i) Chooses actions based on a utility function that maxi
~~(ii) Uses stored knowledge (model) to simulate the future state.~~	performance. (ii) Compares different actions an select the one with the utility value.
(iii) Works towards achieving a predefined goal.	(iii) Not just goal-driven, but optimizes for the best possi outcome.
(iv) Limited handling of uncertainty	(iv) Handles uncertainty by assigning utilities.

Q.7]. Explain the architecture of a knowledge based agent an learning agent.

⇒



A Knowledge based agent (KBA) makes decisions using a knowledge base and reasoning mechanisms.

Components :-

1> Knowledge Base stores facts, rules and logical statements

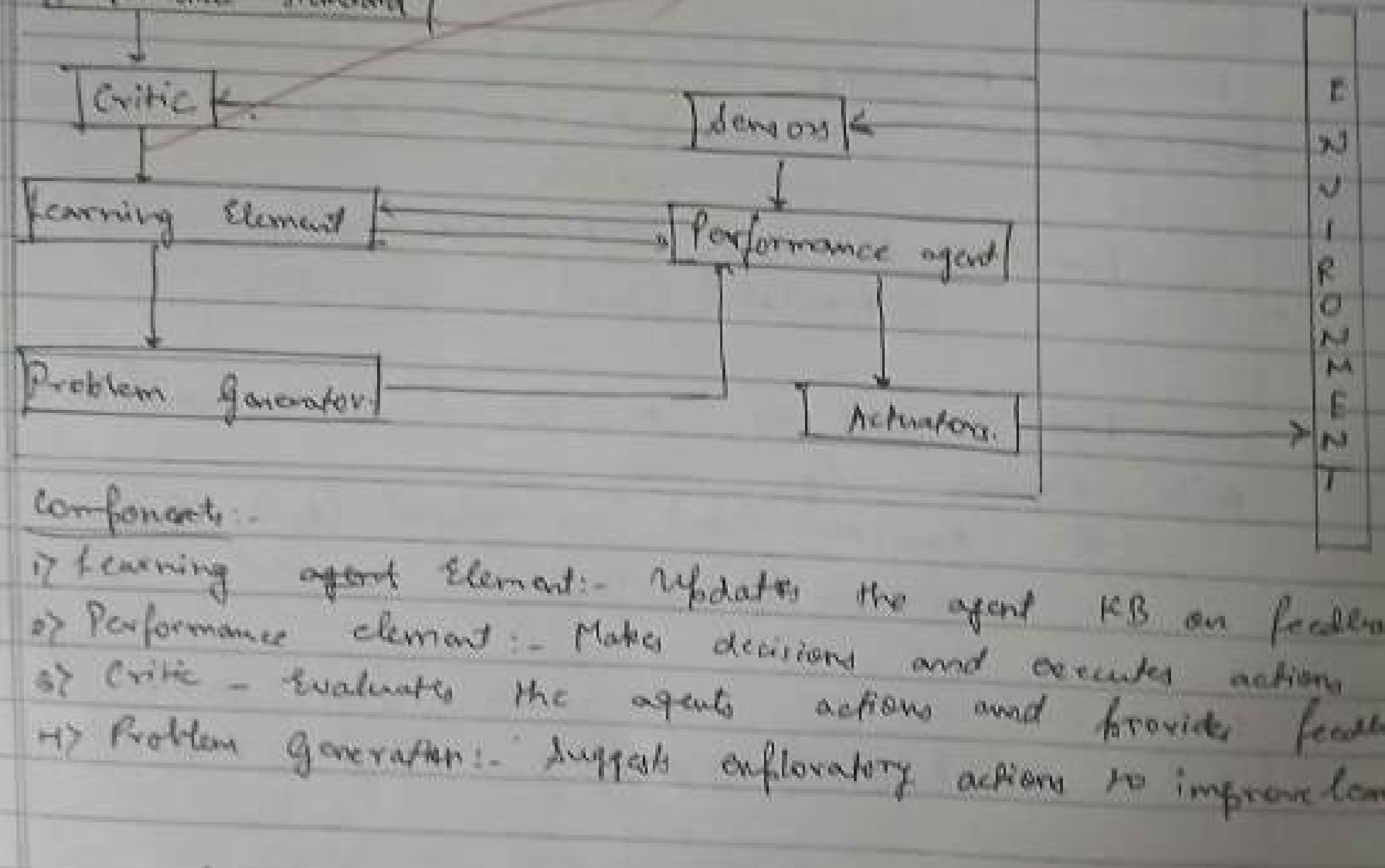
2> Inference Engine :- Applies logical reasoning to derive conclusions from the KB

3> Perception :- Collects input factors from the environment

4> Action Execution:- Performance actions based on decisions

5> Updating Mechanism:- Modifies the KB as new information is learning

Learning based agent architecture:-



Components :-

1> Learning ~~agent~~ Element :- Updates the agent KB on feedback

2> Performance element :- Makes decisions and executes actions

3> Critic - Evaluates the agents actions and provide feedback

4> Problem Generator :- Suggests exploratory actions to improve learning

Q.9] Convert the following to fredicat

→ Anita travels by car if available otherwise travels bus.

→ Bus goes to Andheri via Goregaon

→ Car has puncture so it is not available.

Will Anita travel via Goregaon? Use Forward Reasoning.

⇒ (a) Available (car) → travels (Anita, car) —— (i)

¬Available (car) → travels (Anita, Bus) —— (ii)

(b) visits (Bus, Goregaon) ∧ visits (Bus, Andheri) — (iii)

(c) Puncture (Car) → (¬Available (Car)) —— (iv).

Forward Reasoning

From Fact (iv), we get the Car has Puncture so it is not available.

Puncture (Car) → (¬Available (Car)).

∴ ¬Available (Car) —— (v).

From Fact (i) and (v), as car is not available, Anita travels by bus.

¬Available (car) → travels (Anita, Bus).

∴ travels (Anita, Bus) —— (vi)

From Fact (ii), bus travels from Andheri via Goreg

visits (Bus, Goregaon) ∧ visits (Bus, Andheri).

From Fact (vi), Anita travels via bus

∴ From Fact (ii) and (vi).

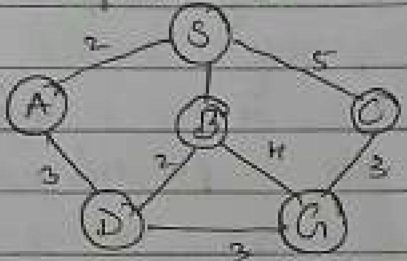
∴ travelsvia(Anita, Goregaon)

∴ Anita travels via Goregaon.

Teacher's Sign :

Q.10] Find the route from S to G using BFS.

→



⇒ For breadth-first search, we consider a queue.

Step 1:- add source S to queue and mark as visited.

Visited = {S} | S | |

Step 2:- Dequeue S, enqueue its neighbours and mark S visited.

Visited :- {S,A,B,C} | A | B | C |

Step 3:- Dequeue A, enqueue its neighbours and mark visited.

Visited :- {S, A.} | B | C | D |

Step 4:- Dequeue B, enqueue its neighbours and mark visited.

Visited :- {S, A, B} | C | D | G. |

Goal state G is found.

$\boxed{\text{Route} = S \rightarrow B \rightarrow G.}$

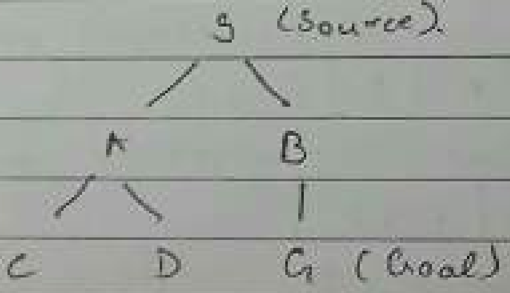
Q.11] What do you mean by Depth Limited Search? Explain Iterative Deepening search with example

⇒ (i) Depth Limited Search is a modified Depth-First Search (DFS) with a predefined limit.

(ii) It avoids infinite loops in graphs with cycles & saves memory

(iii) If a goal is not found within the limit fails

eg:-



If depth = 1.

Depth level 0 → S.

Depth level 1 → A & B

Cannot find goal.

If depth = 2

Depth level 0 → S

Depth level 1 → A, B.

Depth level 2 → C, D, G.

Goal state G can be found

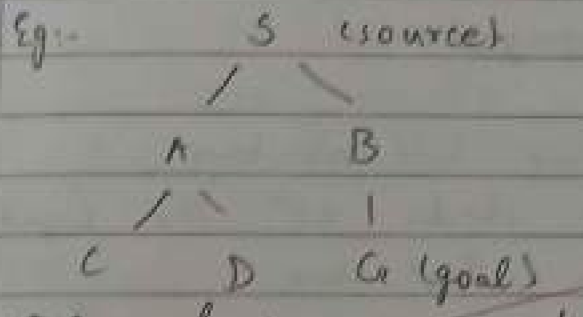
Time complexity = $O(b^d)$

where b = branching factor, d = depth limit

Space complexity = $O(d)$.

(i) IDS is a combination of DFS and BFS
(ii) It runs DFS repeatedly, increasing the depth limit one step at a time.
(iii) It finds the shortest path using BFS and uses less memory

Eg:-



Iteration	Depth limit	Node
1	0	S
2	1	S, A, B
3	2	S, A, B, C, D, G

IDS finds the shortest path while avoiding deep infinite loops

Time complexity = $O(b^d)$.
where b = branching factor ; d = depth of goal.
Space complexity = $O(b^d)$.

Q.2 Explain Hill Climbing and its drawbacks in detail with example. Also state limitations of steepest-ascent hill climbing

⇒ Algorithm:-
(i) Start with an initial solution (state)
(ii) Evaluate neighbouring state.
(iii) Move to the neighbour with the highest value
(iv) Repeat until no better neighbour exists

→ Hill climbing is a local search algorithm used to find the optimal solution by iteratively making small changes to the current state and choosing the best improvement

Drawbacks of Hill Climbing

(i) Local Minima: May get stuck at a peak that is not the global maximum

(ii) Plateau: A flat region with no improvement the algorithm may fail to follow

(iii) No Backtracking: Once a move is made, from state are not recognized

Limitations of steepest - ascent hill climbing

(i) Computationally expensive: It checks all the possible moves before choosing the best one

(ii) Still suffers from local maxima and plateau

(iii) May oscillate between states if 2 equally good options exist

Q.13) Explain simulated annealing and write its algorithm

⇒ (i) Simulated Annealing is a metaheuristic optimization algorithm that helps in escaping local maxima by sometime accepting worse solutions to explore a better global solution.

(ii) It is inspired by the annealing process in metallurgy where metals are heated and cooled slowly to reach a stable structure

Algorithm:-

(i) Initialize the solution and set the initial temp

(ii) Repeat until temperature is low:

- Generate a new neighbour state
- Calculate ΔE
- If ΔE > 0 accept new state.
- Else, accept with probability $p = e^{\left(\frac{\Delta E}{T}\right)}$
- Reduce Temperature using a cooling function.

(iii) Return the best solution

Teacher's Sign :

Q 14] Explain A* algorithm with an example.

⇒ A* is like best-first search algorithm, used in pathfinding and graph traversal. It uses the following formulas.

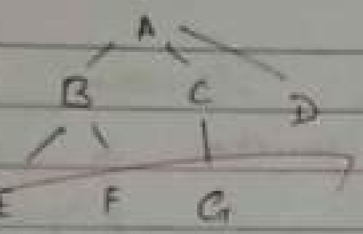
$f(n) = g(n) + h(n)$

$g(n)$ → cost to reach 'n' from start.

$h(n)$ → heuristic estimate of cost to reach from goal to 'n'.

$f(n)$ → total estimated cost

Eg.-



~~Som~~ Initial state = A   Goal state = G.

Node	g(A, n)	h(n).
A	0	6
B	1	4
C	2	2
D	4	7
E	3	5
F	5	3
G	6	0

Steps:-

1) Start at initial state A.

$f(A) = g(A) + h(A) = 6.$

2) Expand Neighbours.

$f(B) = g(B) + h(B) = 1+4=5$

$f(C) = 2+2=4$

$f(D) = 4+7=11.$

3> Select lowest value that is f(c).
4> Expand neighbours of C.
∴ f(G) = 6+0 = 6
5> Goal reached at G with cost 6.

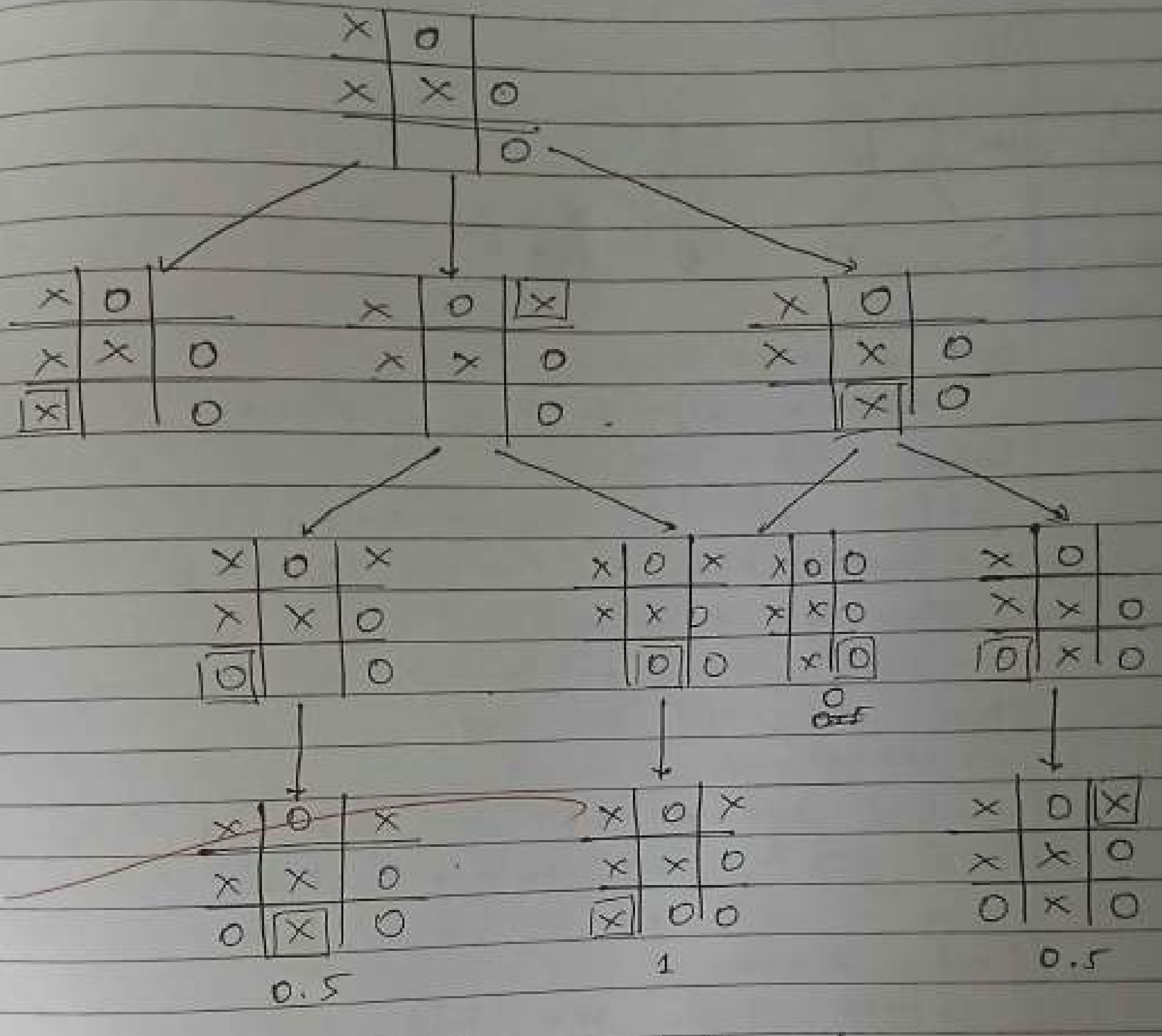
Advantages.

It is efficient for finding shortest path in a wei graph balances exploration by considering bo g(n) and h(n).

Q.15] Explain min-max algorithm and draw game tree

→ The min-max algorithm is a decision making algorithm used in 2 player games.
It assumes one player (MAX) tries to maximise score and other player (MIN) tries to minimize score.

Algorithm:-
1> Generate game tree
2> Assign scores
3> MAX picks highest value from children.
MIN picks lowest value.
4> Repeat until root node is evaluated starting bottom up approach.



Q.14] Explain alpha beta pruning algorithm for an adversial search with an example.

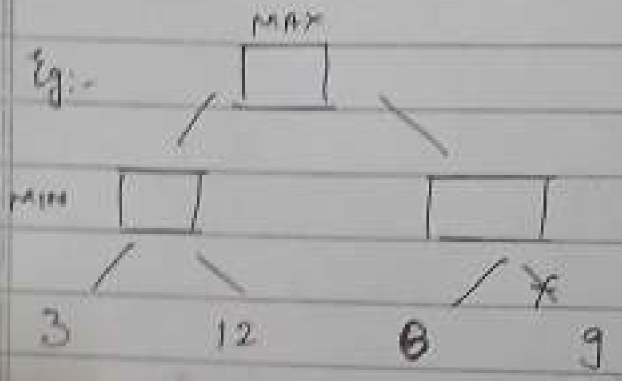
⇒ Alpha Beta pruning is an optimization technique used in min max algorithm to reduce number of nodes evaluated in adversial search problems like game playing AI.

Alpha (α) :- The best maximum score that the maximizing player can guarantee so far.

Beta (β) :- The best minimum score that the minimizing player can guarantee so far.

The algorithm prunes branches that will not influence the final decision.

Eg:-



Start at MAX Node.

- Initialize Alpha (α) = -∞ , β = +∞.

Evaluate Left MIN node

- First child = 3, update β = 3
- Second child = 12, MIN = min (3,12) = 3.
- Now α = -∞ , β = 3.
- Left subtree returns 3 to MAX node.

Move to right MIN node

- MAX node :- α = 3.
- First child of right MIN node = 8 ∴ MIN (8) = 8

So α ≥ β, MIN does not check other child (9) as it will get a worse value.

Final decision

- MAX chooses max (3,8) = 8.

Q.17] Explain WUMPUS world Problem, giving its PEAS descript Explain how percept sequence is generated.

⇒ The wumpus world environment is a simple grid le environment, used in AI to study intelligent agent behaviour in uncertain environments. It is a turn based environment where an agent must navigate a cave to find gold while avoiding hazards like pits and monster called Wumpus.

PEAS:-

P:- The agent is rewarded for gambling gold and exiting safely. Penalty is imposed for falling into pits and getting eaten by wumpus.

E:- 4x4 grid would containing the agent, wumpus, pit, gold.

A:- The agent can move forward, left, right, shoot, climb

S:- Agent perceives stench (near wumpus), breeze (near a pit), glitter (near gold), bump and scream.

Percept sequence generation:-

It is the history of all perceptions received by the agent At each time step, the agent ~~at~~ perceives information based on its current location and surroundings.

Example:-

1> Agent starts at (1,1)

- No breeze, no stench, no glitter (safe space)

2> Agent moves to (2,1)

- Breeze detected, A pit is nearby, not in current square.

3> Agent moves to (2,2)

Glitter detected → gold is here

4> Agent moves to (2,1)

5> Agent moves to (1,1)

Climbs out, wins the game.

Q.18) Solve the Cryptarithmetic SEND+MORE = MONEY

⇒ Step 1:-

$$\begin{array}{r} SEND \\ +\ MORE \\ \hline M\ ONEY \\ \hline \end{array}$$

M = 1 as the addition of 2 4 digit number cannot value ≥ 20000.

Also, S+M will generate a carry of 1

$$\begin{array}{r} SEND \\ +\ 1ORE \\ \hline 1\ ONEY \\ \hline \end{array}$$

Step 2:- Let E+O = 10+N.

Thus, E+O will generate a carry

∴ S must be 8 or 9 as it should generate a carry

But S ≠ 9 as 9+1+1 = 10+① → O O ≠ ~~0~~1 as M=1.

∴ Let S = 8 and O = 0 (zero)

Step 3:- But now, E + O = N ∴ E=N which is not possible and this cannot generate carry

∴ Let S = 9.

$$\begin{array}{r} 9END \\ +\ 1ORE \\ \hline 1\ ONEY \\ \hline \end{array}$$ ∴ O = 0 (zero).

Step 4:- E + O = N ∴ E=N which is not true

∴ E + O + 1 = N

i.e. E+1 = N. ∴ N+R must generate carry.

Also consider N+R+ C1 = 10+E.

∴ R + C1 = 10 − 1 = 9.

If C1 = 0, ~~But~~ R ≠ 9 as S = 9.

C1 = 1 (Carry). ∴ D+E = Y+10 ∴ R = 8

```
    9 E N D
+ 1 0 8 E
---------
1 0 N E Y
```

Now, N+8+1= E + 10

∴ N - E = 1

lets assume E = 5 .

∴ N = 6.

```
    9 5 6 D
+ 1 0 8 5
---------
1 0 6 5 Y
```

Now, only equation left is D + 5 = Y+10.

∴ Remaining values = {2, 3, 4, 7}.

As 2, 3, 4 cannot generate carry with 5,

∴ D = 7

∴ 7 + 5 = Y + 10

∴ Y = 2

```
∴   9 5 6 7          ∴ S = 9
  + 1 0 8 5            E = 5
  ---------            N = 6
  1 0 6 5 2            D = 7
                       M = 1
                       O = 0
                       R = 8.
                       Y = 2.
```

Q.19] Consider the following axioms:
- → All people who are graduating are happy
- → All happy people are smiling.
- → Someone is graduating

(i) Represent in FO predicate logic
(ii) Convert each formula to clause form
(iii) Prove that "Is someone smiling?" using resolution technique. Draw the resolution tree.

⇒ Consider:-

$G(x)$: x is graduating.
$H(x)$: x is happy
$S(x)$: x is smiling

$\therefore \forall(x) : G(x) \rightarrow H(x)$ — (i)
$\therefore \forall(x) : H(x) \rightarrow S(x)$ — (ii).
$\therefore \exists(x) : G(x)$ — (iii)

Converting to clause form.

Fact 1:- $\therefore$ ~~$\forall(x)$~~ : $\neg G(x) \vee H(x)$ — (iv)
Fact 2:- $\therefore$ ~~$\forall(x)$~~ : $\neg H(x) \vee S(x)$ — (v)
Fact 3:- $\therefore$ $G(x)$. — (vi)

Apply resolution:-

Consider contradiction that ~~no one~~ x is not smiling
$\therefore \neg S(x) \rightarrow$ Goal. — (vii)

Resolve (iv) and (v).

$\therefore \neg G(x) \vee H(x) \qquad \neg H(x) \vee S(x)$

$\neg G(x) \vee S(x)$ — (viii)

Resolve (vii) and (vi).

∴ $\neg G(x) \vee S(x)$ $\qquad$ $G(x)$.

∴ $S(x)$. — $(\overset{ix}{\cancel{viii}})$

Resolve $(\overset{ix}{\cancel{viii}})$ and (vi).

∴ $S(x)$ $\qquad$ $\neg S(x)$

$\{ \phi \}$.

As it returns empty set.

∴ Our assumption of the contradictions is wrong.

∴ Someone is smiling.

Q 20 Explain modus ponen with example

⇒ Modus Ponen is a fundamental rule of inference in propositional logic that allows us to deduce a conclusion from a conditional statement and its antecedent.

It follows the form.

ie $P \rightarrow Q$ (P implies Q)

i.e. If P is true.

Then Q must be true

Eg:- If it rains, the ground is wet

Let P = rain, Q = ground to is wet.

∴ $P \rightarrow Q$.

Now, according to Modus Ponen.

If it rains (P is true).

the ground must be wet (Q must be true).

Q.21] Explain forward chaining and backward chaining with an example.

⇒ Forward Chaining:- It starts with given facts and applies inference rules to derive new facts until the goal is reached. It is a data driven approach because it begins with known data and works forward to reach a conclusion

& Backward Chaining:- It starts with a goal and works backwards by checking what facts are needed to support it. It is a goal driven approach

Example:- Diagnosing a disease.

Rules:-

1. If a person has a fever and cough they have a flu.

Facts:-

1. Patient has ~~cold~~ cough
2. Patient has fever

Goal:- Prove patient has flu.

Forward chaining

Consider ~~rules~~ facts 1 and 2:-

Patient has cough and fever

From rule 1:-

$\forall x : fever(x) \wedge cough(x) \rightarrow flu(x)$

The If x:- patient

∴ Patient has the flu

Backward chaining

We have to prove that patient has flu.

For that, from rule 1

$\forall x: fever(x) \wedge cough(x) \rightarrow flu(x)$

If patient has flu,

→ patient must have fever and cough.

From facts 1 and 2,

fever (patient) cough(patient)

∴ Patient has the flu.

Sairam Konar **D15C** **27** **2024-2025**

AI & DS - I (Assignment - 2)

Q.1: Use the following data set for question 1
82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90
1. Find the Mean (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Ans:

$$\text{Mean} = \frac{\sum(data\ elements)}{Number\ of\ data\ elements} = \frac{(82+66+70+59+90+78+76+95+99+84+88+76+82+81+91+64+79+76+85+90)}{20}$$

= 1611 / 20 = 80.55

Mean = 80.55

For finding the median, we have to arrange the give data in ascending order.

Data = 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Number of elements = 20.

Thus, Median = {(n/2)th element + [(n/2)+1]th element} / 2

n/2 = 20 / 2 = 10

Median = (10th element + 11th element)/2
= (81+82) / 2 = 163 / 2
Median = 81.5

The mode of a dataset with discrete elements is the element with the most amount of occurrences.

In this data, 76 is occurring the most number of times (3 times).

Thus, **Mode = 76**

Interquartile range = Q3 - Q1
Where Q3 and Q1 are the Upper and Lower quartiles respectively.

To find Q3 and Q1, we split the dataset amongst the median.
Data (Lower) = 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
Data (Higher) = 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Now, Q1 = median of Lower data = (5th element + 6th element)/2 = (76+76)/2 = 76
Q3 = median of Higher data = (5th element + 6th element)/2 = (88+90)/2 = 89

Thus IQR = Q3 - Q1 = 89 - 76
IQR = 13

Answer:
Mean = 80.55
Median = 81.5
Mode = 76
IQR = 13

Q2] 1) Machine Learning for Kids 2) Teachable Machine
1. For each tool listed above

- identify the target audience
- discuss the use of this tool by the target audience
- identify the tool's benefits and drawbacks

2. From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?

- Predictive analytic
- Descriptive analytic

3. From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Ans:
For **Machine Learning for Kids:**

1) **Target Audience:** Primarily designed for school students (ages 8–16) and educators. It is also suitable for beginners with no prior coding experience who want to explore machine learning

2) **Usage:**
   - Provides a child-friendly interface for creating ML models using text, images, numbers, or sounds.
   - Often integrated with tools like Scratch or Python to develop interactive projects (e.g., games or chatbots).
   - Teachers use it to introduce fundamental ML concepts and encourage creative problem-solving.

3) **Benefits:**
   - Free, web-based, and easily accessible.
   - Promotes experiential learning through hands-on projects.
   - Combines coding and ML concepts effectively.
   - Supports multiple data types (text, images, numbers).

4) **Drawbacks:**
   - Limited scope for advanced ML projects.
   - Data collection and labeling can be time-consuming.
   - Not suitable for real-world ML deployment.

5) This tool would be described as a **predictive analytic** tool as the models learn from labeled data to predict or classify new inputs (e.g., recognizing if text is positive or negative).

6) This tool would use **supervised learning** as it would have a set of predefined outputs set up for a set of questions and would use the inputs from the kids to compare with the predefined outputs to provide a score.

For **Teachable Machine:**

1) **Target Audience:** Suitable for general users, including students, educators, artists, developers, and non-coders. Ideal for quick prototyping and demonstrations.

2) **Usage:**
   - Allows users to train ML models using images, sounds, or poses with minimal effort.
   - No coding required—models can be exported for web, apps, or TensorFlow.
   - Useful for quick demos, accessibility tools, and creative tech projects.

3) **Benefits:**
   - Extremely user-friendly with drag-and-drop functionality.
   - Supports multiple input types (image, audio, pose).
   - Provides instant feedback and model export options.
   - Great for visual learners and rapid experimentation.

4) **Drawbacks:**
    - Models trained on small datasets may not generalize well.
    - Limited customization and no coding flexibility.
    - Not designed for production-level accuracy or fine-tuning.

5) This tool would be described as a **predictive analytic** tool as the models learn from labeled data/answers to predict or classify new inputs (e.g., recognizing if text is positive or negative).

6) This tool would use **supervised learning** as it would have a set of predefined outputs set up for a set of questions and would use the inputs from the users to compare with the predefined outputs to provide a score.

Q3] Data Visualization: Read the following two short articles:

- Read the article Kakande, Arthur. February 12. “What’s in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization.” Medium
- Read the short web page Foley, Katherine Ellen. June 25, 2020. “How bad Covid-19 data visualizations mislead the public.” Quartz
- Research a current event which highlights the results of misinformation based on data visualization.

Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

Ans:

1) **What’s in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization**

1. The Growing Need for Responsible Data Visualization

With over 2.5 quintillion bytes of data generated daily, effective visualization has become crucial for making sense of complex information. However, the same tools meant to clarify data can also mislead through intentional manipulation or unintentional design flaws. As organizations like Mozilla have found, platforms and creators share responsibility for combating misinformation - making it essential for both designers and consumers to develop critical data literacy skills.

2. Common Visualization Pitfalls That Spread Misinformation

Several techniques frequently distort data representation: truncated y-axes exaggerate minor differences; misapplied color schemes reverse conventional meanings (like using red for growth); cherry-picked timeframes present misleading trends; and pie charts displaying impossible percentages (like 110%). Perhaps most dangerously, visualizations often falsely imply causation from correlation - a problem highlighted during COVID-19 when people misinterpreted coinciding data points as proof of relationships.

3. Building Defenses Against Deceptive Visuals

Combating visualization misinformation requires vigilance from both creators and consumers. Designers must maintain ethical standards by using proper scales, consistent color coding, and full-context data. Consumers should scrutinize axes, check sources, and verify whether visual claims match underlying numbers. As noted by data integrity advocates like Pollicy, developing these analytical skills represents our best defense against misleading data representations in an increasingly visual information ecosystem.

2) How bad Covid-19 data visualizations mislead the public.

1. Lack of Context and Misleading Scales

Many state dashboards presented COVID-19 data without proper context, leading to misinterpretation. For example, Arkansas visualized preexisting health conditions using percentage arcs scaled to 100%, making serious comorbidities like hypertension (9.6%) appear insignificant. In reality, this represented over 1,700 high-risk patients—a critical detail lost in the visualization. Similarly, Arizona's case charts omitted y-axis labels, making statewide cases (thousands) appear comparable to county-level data (under 100). Without clear reference points, these visualizations distorted public perception of risk (Quartz, 2020).

2. Overuse of Problematic Chart Types

Several states relied on ineffective formats like pie charts and cluttered snapshots. Alabama's daily reports used pie charts to display case demographics, despite research showing pie charts hinder proportional comparisons (Few, 2007). The state also published dense numerical snapshots without trendlines, obscuring the pandemic's progression. As Quartz's data editor noted, bar charts or tables would have better served public understanding by simplifying cognitive load (Quartz, 2020).

3. Inconsistent or Missing Reference Data

Visualizations often failed to include benchmarks like testing rates or demographic baselines. Washington succeeded by pairing positive test percentages with total tests administered, clarifying testing capacity growth. In contrast, New York's early dashboards showed case counts without adjusting for population density, exaggerating rural outbreaks. Such omissions fueled misinformation, as seen when unlabeled scales in Arizona's heatmaps obscured disproportionate Native American infection rates (NYT, 2020). These cases underscore how missing reference data can alter policy decisions and public behavior.

3) Climate change sceptics use misleading Arctic ice data to make case

In April 2024, a misleading headline from the *Daily Sceptic* claiming that "Arctic Sea Ice Soars to Highest Level for 21 Years" went viral on social media. The article compared sea ice extent on January 8, 2024, with the same date in 2004 and suggested that fears about global warming were exaggerated. This claim, despite being technically accurate for that specific date, was shared alongside comments denying climate change and accusing scientists of promoting a false narrative. Posts like these spread misinformation by presenting isolated data points out of context and ignoring broader scientific trends.

Experts quickly pointed out that this type of comparison is a classic example of “cherry-picking.” Scientists from the National Snow and Ice Data Center (NSIDC) and the British Antarctic Survey clarified that using just one day’s data to assess long-term changes is scientifically invalid. Arctic sea ice levels fluctuate seasonally, typically peaking in March and reaching their lowest levels in September. Comparing single days across years doesn’t reflect these natural patterns. In fact, although January 8, 2024, showed slightly more sea ice than the same day in 2004, large parts of February and March 2024 had *lower* sea ice levels compared to 2004. This shows that relying on just one favorable data point gives a misleading impression.

The long-term trend, as documented by satellite records since 1979, shows a clear and consistent decline in Arctic sea ice. According to NSIDC, the average minimum ice extent dropped from 6.95 million sq km in 1979–1990 to 4.42 million sq km in 2011–2020. Climate experts emphasize that this continuous decline is far more meaningful than any short-term variation. Misleading visualizations like the one used by the *Daily Sceptic* distort public understanding and can undermine support for urgent climate action. It highlights the importance of presenting data within proper context and educating the public on how to interpret visual information critically.

Source: Climate change sceptics use misleading Arctic ice data to make case

Q. 4 Train Classification Model and visualize the prediction performance of trained model required information

- Data File: Classification data.csv
- Class Label: Last Column
- Use any Machine Learning model (SVM, Naïve Base Classifier)

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

Pima Indians Diabetes Database

Ans:

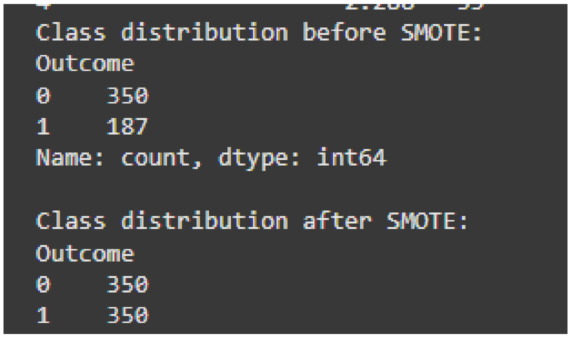
Dataset Description: The Pima Indians Diabetes Dataset contains 768 records of female patients aged 21+, with 8 medical features and a target column Outcome indicating diabetes (1) or not (0). It's widely used for binary classification and poses challenges like missing values, class imbalance, and non-linear relationships.

Features provided:

- **Pregnancies**:
  Number of times the patient has been pregnant. It is a numeric count and can influence diabetes risk due to hormonal changes.
- **Glucose**:
  Plasma glucose concentration (mg/dL) measured 2 hours after a glucose tolerance test. Higher values often indicate diabetes.
- **BloodPressure**:
  Diastolic blood pressure (mm Hg). Elevated blood pressure is often correlated with diabetes and other metabolic issues.
- **SkinThickness**:
  Triceps skinfold thickness (mm). It provides an estimate of body fat percentage.
- **Insulin**:
  2-hour serum insulin level (mu U/mL). Helps assess insulin resistance; zero values may indicate missing data.
- **BMI**:
  Body Mass Index (weight in kg/(height in m)^2). A high BMI can be a risk factor for diabetes.
- **DiabetesPedigreeFunction**:
  A function that scores likelihood of diabetes based on family history (genetic predisposition).
- **Age**:
  Age of the patient in years. Risk of diabetes generally increases with age.
- **Outcome**:
  Target variable — 1 if the patient has diabetes, 0 otherwise. Used for classification.

Models:

1) **Class Imbalance:**
   The dataset's class imbalance issue is addressed using SMOTE (Synthetic Minority Oversampling Technique) from the ***imblearn*** library. After splitting and scaling the training data, SMOTE is applied to generate synthetic samples of the minority class, ensuring a balanced dataset for model training. This helps the model avoid bias toward the majority class and improves learning of decision boundaries for underrepresented classes. The balanced dataset is then used to train both SVM and Naïve Bayes classifiers, leading to more robust and fair performance during validation and testing phases.



```
Class distribution before SMOTE:
Outcome
0    350
1    187
Name: count, dtype: int64

Class distribution after SMOTE:
Outcome
0    350
1    350
```

Applying SMOTE to balance the classes significantly improved the models' ability to recognize minority class instances. Without this step, the model would likely have favored the majority class, leading to skewed predictions and poor recall for the minority class.

2) **Data Preprocessing:**

Data preprocessing is essential to make raw features suitable for machine learning models. In this case, the features were scaled using StandardScaler, which standardizes each feature to have zero mean and unit variance. This normalization step ensures that all attributes like Glucose, BMI, and Insulin contribute equally to model learning, rather than allowing features with larger magnitudes to dominate.

3) **Train Test Validation splitting/Random splitting**

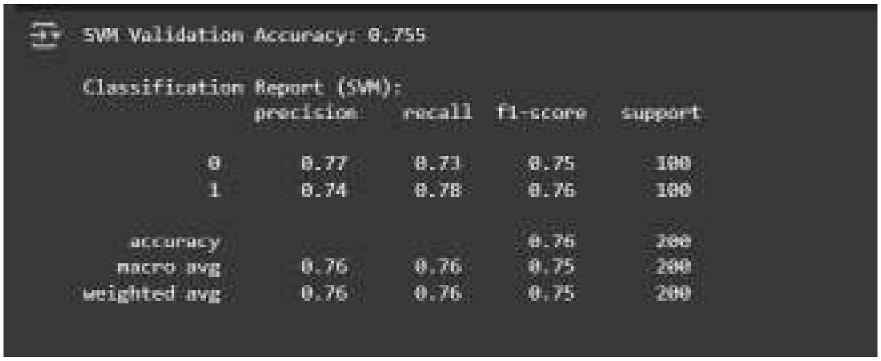
The train, validation, and test split (70/20/10) is achieved by first splitting the dataset into training + validation (70%) and test (10%) using train_test_split. Then, the training data is further divided into training (about 62.22%) and validation (about 17.77%) sets, resulting in approximately 70% for training, 20% for validation, and 10% for testing. The splitting is done randomly using train_test_split, ensuring a random distribution of data for each subset. The parameter random_state=42 ensures reproducibility of the splits, making sure the data is consistently divided across different runs.

4) **Models used:**
   - **SVM** is a powerful classification model that finds the optimal hyperplane to separate classes. It was tuned using GridSearchCV for optimal parameters like C, kernel, and gamma. SVM works well for high-dimensional data and can handle both linear and non-linear boundaries.
   - **Naïve Bayes** assumes feature independence and works well with imbalanced datasets. The Gaussian Naïve Bayes model assumes features follow a normal distribution. Despite its simplicity, it performed better than SVM, making it the final model for testing. Both models were evaluated on their classification performance.

Validation Performance:

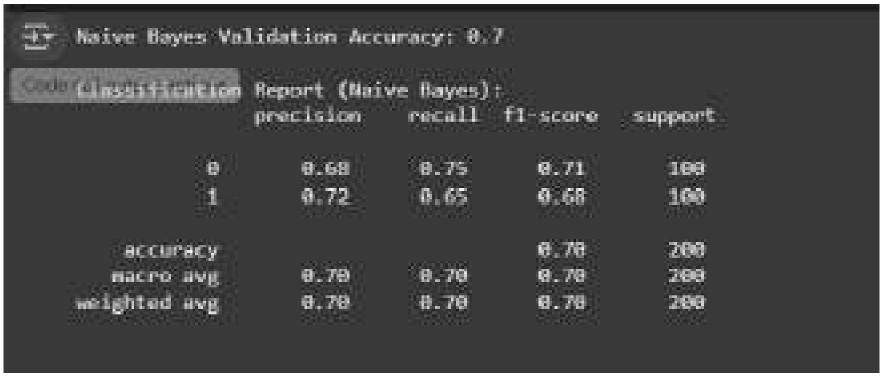
The validation performance of both models varied significantly. The Support Vector Machine (SVM) achieved a validation accuracy of 70.77%, which, while decent, was lower than expected given its ability to handle high-dimensional data. SVM's performance was limited by class imbalance and the need for careful hyperparameter tuning. On the other hand, Naïve Bayes outperformed SVM with a validation accuracy of 77.27%, demonstrating its robustness in handling imbalanced datasets despite the simplifying assumption of feature independence. This highlights Naïve Bayes' strength in such contexts, where its simplicity and efficiency provide a better fit for the data.

SVM Validation Accuracy:



```
SVM Validation Accuracy: 0.755

Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.77      0.73      0.75       100
           1       0.74      0.78      0.76       100

    accuracy                           0.76       200
   macro avg       0.76      0.76      0.75       200
weighted avg       0.76      0.76      0.75       200
```

Naive-Bayes Accuracy:



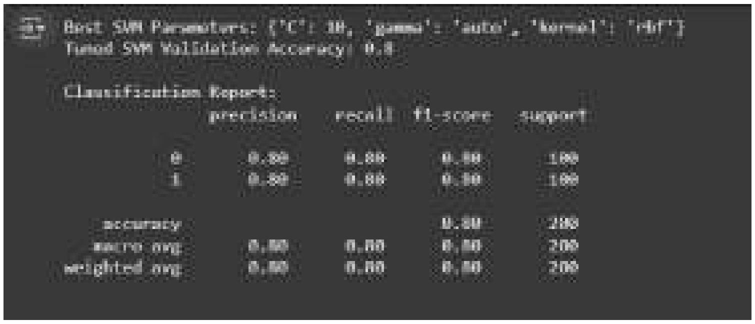
```
Naive Bayes Validation Accuracy: 0.7

Classification Report (Naive Bayes):
              precision    recall  f1-score   support

           0       0.68      0.75      0.71       100
           1       0.72      0.65      0.68       100

    accuracy                           0.70       200
   macro avg       0.70      0.70      0.70       200
weighted avg       0.70      0.70      0.70       200
```

As it is required to select the model giving a better accuracy of predictions, we would be choosing the SVM Model for further performance.

5) **Hyper parameter tuning**

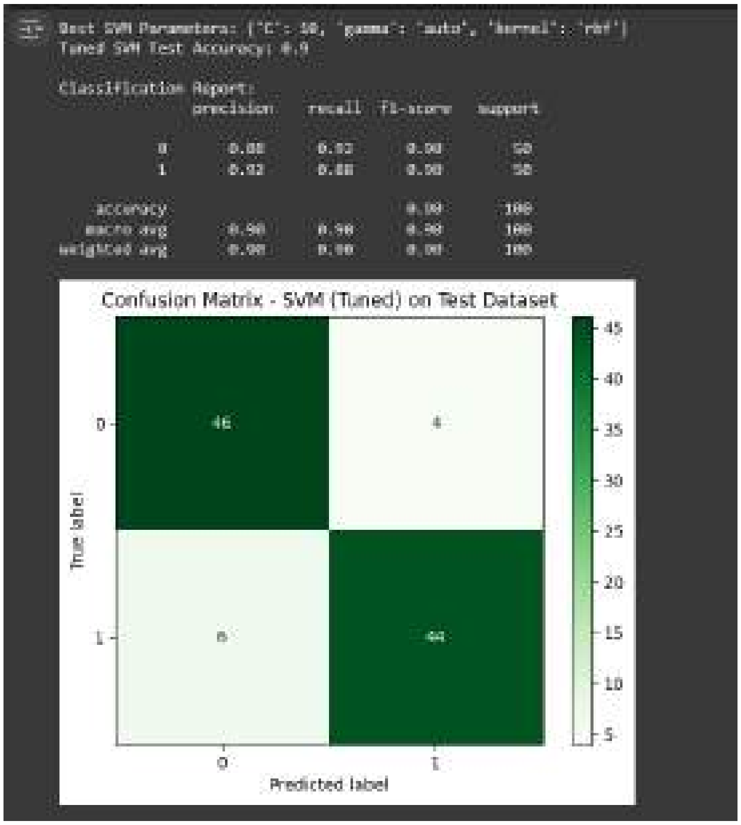
Hyperparameter tuning involves optimizing the parameters of a machine learning model to achieve the best performance. In this case, GridSearchCV was used to tune the hyperparameters of the Support Vector Machine (SVM) model, specifically the regularization parameter C, kernel type, gamma, and polynomial degree. The goal was to find the optimal configuration that maximized the model's validation accuracy.

SVM Accuracy (After Hyperparameter Tuning):



```
Best SVM Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Tuned SVM Validation Accuracy: 0.8

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.80      0.80       100
           1       0.80      0.80      0.80       100

    accuracy                           0.80       200
   macro avg       0.80      0.80      0.80       200
weighted avg       0.80      0.80      0.80       200
```

Also, after running this same code on the test dataset, we have achieved a high accuracy of 90%.



```
Best SVM Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
Tuned SVM Test Accuracy: 0.9

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.92      0.90        50
           1       0.92      0.88      0.90        50

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
weighted avg       0.90      0.90      0.90       100
```

Using this method (SVM with Hyperparameter tuning) we have achieved a high accuracy of Validation accuracy at 80% and Test Accuracy of 90%. These accuracies are also supported by the other performance parameters such as precision, recall and f1-score.

Q.5 Train Regression Model and visualize the prediction performance of trained model

- Data File: Regression data.csv
- Independent Variable: 1st Column
- Dependent variables: Column 2 to 5

Use any Regression model to predict the values of all Dependent variables using values of 1st column.

Requirements to satisfy:

- Programming Language: Python
- OOP approach must be followed
- Hyper parameter tuning must be used
- Train and Test Split should be 70/30
- Train and Test split must be randomly done
- Adjusted R2 score should more than 0.99
- Use any Python library to present the accuracy measures of trained model

https://github.com/Sutanoy/Public-Regression-Datasets

https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv

https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20data%20set.xlsx

Ans:

Dataset Description:

The Dry Bean Dataset consists of 13,611 samples of seven types of dry beans. Each sample is represented by various numerical features extracted from images through shape analysis techniques. These features provide geometric, statistical, and morphological insights into each bean's physical characteristics.

The dataset is often used for classification and regression tasks related to agricultural or computer vision applications.

Feature Description:

- Area: Total number of pixels inside the bean boundary, representing its size.
- Perimeter: Total length of the bean's outer boundary (contour).
- MajorAxisLength: Length of the major axis of the ellipse that best fits the bean.
- MinorAxisLength: Length of the minor axis of the ellipse that best fits the bean.
- Eccentricity: Measure of how elongated the shape is, ranging from 0 (circle) to 1 (line).
- ConvexArea: Number of pixels in the convex hull that encloses the bean.
- EquivDiameter: Diameter of a circle with the same area as the bean.
- Extent: Ratio of the bean area to the area of its bounding box.
- Solidity: Ratio of the bean's area to its convex area, measuring convexity.
- Compactness: Shape compactness measured using the perimeter and area.
- roundness: Indicates how close the bean shape is to a perfect circle.

- AspectRation: Ratio of major axis length to minor axis length, indicating elongation.
- ShapeFactor1: Derived shape descriptor using area and perimeter.
- ShapeFactor2: Derived shape descriptor using perimeter and major axis.
- ShapeFactor3: Derived shape descriptor using area and minor axis.
- ShapeFactor4: Derived shape descriptor using area and major axis.
- Class: Categorical label representing the bean type (e.g., SIRA, HOROZ, etc.).

Models used:

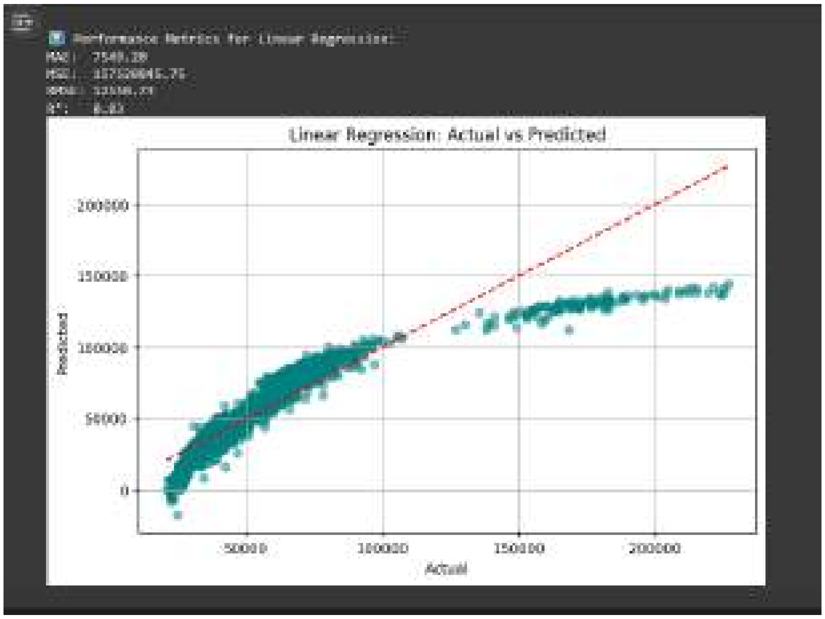
In this analysis, two regression models were employed: Linear Regression and Random Forest Regressor. Linear Regression is a simple, interpretable model that assumes a linear relationship between input features and the target variable. It serves as a good baseline but may underperform with complex, non-linear data. The Random Forest Regressor, on the other hand, is an ensemble-based model that builds multiple decision trees and averages their predictions, improving accuracy and reducing overfitting. It captures non-linear patterns effectively. Hyperparameter tuning further enhances its performance, making it more robust and reliable for real-world prediction tasks involving complex feature interactions.

Inference:

The predictive performance of both the Linear Regression and the tuned Random Forest Regressor was evaluated using key regression metrics — Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2). The Linear Regression model, though relatively simple, achieved an R^2 value of 0.83, indicating that it could explain about 83% of the variability in the bean area using the selected features. However, it also exhibited a high MAE of 7549.20 and an RMSE of 12550.73, suggesting substantial prediction errors and lower reliability for precise estimations.

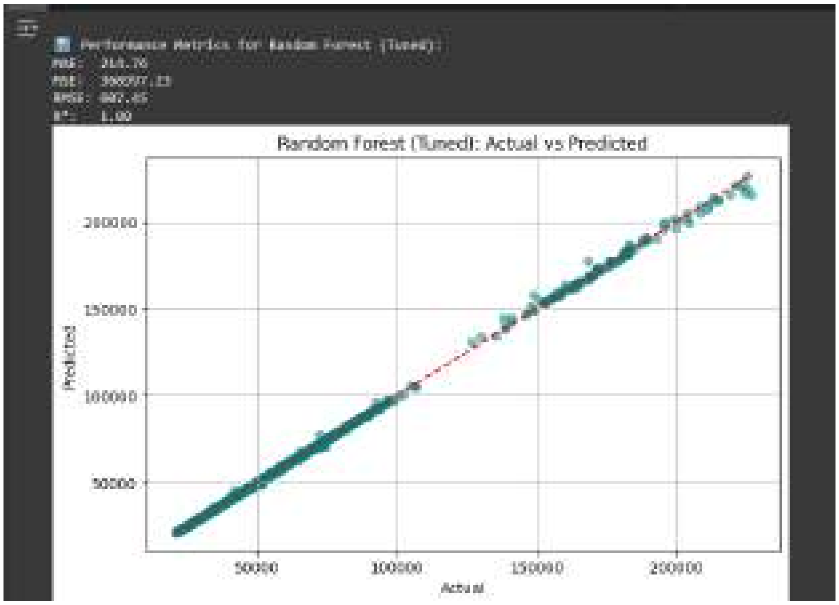
On the other hand, the Random Forest Regressor, after hyperparameter tuning, significantly outperformed Linear Regression across all metrics. It achieved a near-perfect R^2 score of 1.00, indicating that it almost completely captured the variance in the target variable. The MAE dropped to just 214.76, and RMSE reduced dramatically to 607.45, reflecting far more accurate and stable predictions. This performance gap highlights that Random Forest's ensemble-based, non-linear approach is better suited for this dataset, likely because it can model complex relationships between features that a linear model cannot. Therefore, for tasks involving the prediction of bean area based on physical shape features, the Random Forest Regressor is clearly the more effective and robust choice.

Performance of linear Regression:



```
📊 Performance Metrics for Linear Regression:
MAE:  7549.28
MSE:  157320845.75
RMSE: 12538.73
R²:   0.83
```

Performance of Random Forest Regressor:



```
📊 Performance Metrics for Random Forest (Tuned):
MAE:  218.76
MSE:  398597.23
RMSE: 607.65
R²:   1.00
```

Q6] What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

Ans:

- **Feature Importance**
  1. **Alcohol (0.476)** – Higher alcohol content is strongly associated with better wine quality.
  2. **Sulphates (0.251)** – Moderate levels of sulphates contribute positively to perceived wine quality.
  3. **Citric Acid (0.226)** – Citric acid enhances freshness and has a mild positive impact on quality.
  4. **Fixed Acidity (0.124)** – Slightly helps in preserving wine quality, but not a major predictor.
  5. **Residual Sugar (0.014**) – Has minimal influence on quality; too little or too much may be undesirable.
  6. **Free Sulfur Dioxide (-0.051)** – Shows negligible negative correlation; excessive use may degrade quality.
  7. **pH (-0.058)** – Very weak negative relation; lower pH (more acidic wine) could slightly improve quality.
  8. **Chlorides (-0.129)** – Higher salt content tends to reduce the quality of wine.
  9. **Density (-0.175)** – Denser wines tend to be of lower quality, possibly due to higher sugar content.
  10. **Total Sulfur Dioxide (-0.185)** – Excessive sulfur dioxide can negatively affect wine quality and taste.
  11. **Volatile Acidity (-0.391)** – Strong negative impact; higher levels often indicate spoilage or poor quality.

- **Handle Missing Data**

One common method for handling missing data is mean or median imputation, where missing values in a feature are replaced with the mean or median of that column. This technique is simple and works well when the data is normally distributed or has low variability, though it may distort variance and relationships between variables. Another method is mode imputation, used particularly for categorical data, although it can be less applicable in a numerical dataset like wine quality. K-Nearest Neighbors (KNN) imputation considers the similarity between observations and imputes values based on the average of the nearest samples. While more accurate, it is computationally expensive. Multivariate imputation techniques, such as Multiple Imputation by Chained Equations (MICE), use the relationships between multiple variables to estimate missing values, making them more robust for datasets with complex interdependencies.

- **Advantages/Disadvantages of these methods**

1) **Mean/Median Imputation:**

Advantages:

Mean or median imputation is one of the simplest and quickest methods to handle missing data. It helps retain the entire dataset without losing any rows and is easy to implement. Median imputation, in particular, is useful when the data contains outliers, as it is not influenced by extreme values.

Disadvantages:

This method can distort the natural variability in the data and weaken the relationships between variables, especially when a large number of values are missing. Additionally, it assumes that the data is missing completely at random, which may not always hold true in real-world scenarios.

2) **Mode Imputation:**

Advantages:

Mode imputation is best suited for categorical features. It is easy to use and ensures that the dataset size remains the same. This method is helpful when a particular category is dominant in the data.

Disadvantages:

It can lead to a bias toward the most frequent category, especially if the mode is overrepresented due to imputation. This may reduce the predictive power of the model, especially when diversity in categories is important.

3) **K-nearest Neighbours:**

Advantages:

KNN imputation considers the similarity between instances and uses neighboring data points to estimate the missing values, which often results in more accurate and context-aware imputations. It preserves the underlying structure and relationships in the data.

Disadvantages:

It can be computationally expensive, especially for large datasets, and may not perform well when many values are missing or when the data has noise. It also requires proper tuning of the number of neighbors and careful feature scaling.

4) **Dropping Rows/Columns**

Advantages:

Dropping rows or columns is the easiest method and ensures that the remaining data is complete. It is useful when only a very small proportion of the dataset is affected by missing values.

Disadvantages:

This method can result in significant loss of valuable information, especially when the missing data is not random or when many rows or columns are removed. It can also introduce bias if the removed data had specific patterns or relationships.