Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

1. Load data in Pandas.
2. Description of the dataset.
3. Drop columns that aren't useful.
4. Drop rows with maximum missing values.
5. Take care of missing data.
6. Create dummy variables.
7. Find out outliers (manually)
8. standardization and normalization of columns

Steps:

Step 1: Load the File

Pandas library is used to analyze data which provides powerful tools for handling data. It allows us to read **CSV (Comma Separated Values) files** efficiently and perform various data manipulation and analysis tasks.

Commands: import pandas as pd (To Import 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
```

```
[2]  df = pd.read_csv("/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Alzheimer_s_Disease_and_Healthy_Aging_Data.csv")
```

Run command **df.info()** to check whether the file is loaded properly or not .This will print first 5 Rows with all Columns.

```
df.head()
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	ClassID	TopicID
0	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	2022	2022	PA	Pennsylvania	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Native Am/Alaskan Native	POINT (-77.86070029 40.79373015)	C05	TMC01
1	BRFSS~2022~2022~46~Q03~TMC01~AGE~RACE	2022	2022	SD	South Dakota	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Asian/Pacific Islander	POINT (-100.3735306 44.35313005)	C05	TMC01
2	BRFSS~2022~2022~16~Q03~TMC01~AGE~RACE	2022	2022	ID	Idaho	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-114.36373 43.68263001)	C05	TMC01
3	BRFSS~2022~2022~24~Q03~TMC01~AGE~RACE	2022	2022	MD	Maryland	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-76.60926011 39.29058096)	C05	TMC01
4	BRFSS~2022~2022~55~Q03~TMC01~AGE~GENDER	2022	2022	WI	Wisconsin	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Male	POINT (-89.81637074 44.39319117)	C05	TMC01

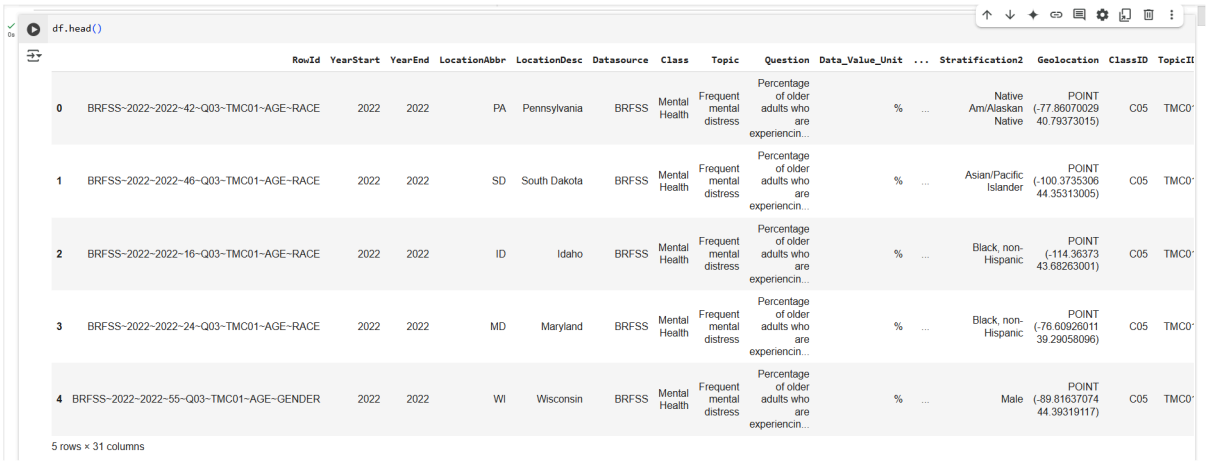
5 rows × 31 columns

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 2: Description of the dataset

1)df.head() : This command is use to print first 5 Rows and all the columns.



```
df.head()
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	ClassID	TopicID
0	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	2022	2022	PA	Pennsylvania	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Native Am/Alaskan Native	POINT (-77.86070029 40.79373015)	C05	TMC01
1	BRFSS~2022~2022~46~Q03~TMC01~AGE~RACE	2022	2022	SD	South Dakota	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Asian/Pacific Islander	POINT (-100.3735306 44.35313005)	C05	TMC01
2	BRFSS~2022~2022~16~Q03~TMC01~AGE~RACE	2022	2022	ID	Idaho	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-114.36373 43.68263001)	C05	TMC01
3	BRFSS~2022~2022~24~Q03~TMC01~AGE~RACE	2022	2022	MD	Maryland	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Black, non-Hispanic	POINT (-76.60926011 39.29058096)	C05	TMC01
4	BRFSS~2022~2022~55~Q03~TMC01~AGE~GENDER	2022	2022	WI	Wisconsin	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	Male	POINT (-89.81637074 44.39319117)	C05	TMC01

5 rows × 31 columns

2)df.info() : This command is use to print all column names with count of non-null values are their data-type.

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

```
<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
```

3)df.describe(): This command is use to print count,mean,std,min,25%,50%,75% and max of all columns which have data type int or float

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
[5] 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	33.027824	42.595333	800.322677
std	2.302815	2.360105	25.213484	25.213484	24.290016	26.156408	2511.564977
min	2015.000000	2015.000000	0.000000	0.000000	-0.700000	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.

```
[6] df.describe(include="all")
```

	RowId	YearStart	YearEnd	LocationAbbr	LocationDesc	Datasource	Class	Topic	Question	Data_Value_Unit	...	Stratification2	Geolocation	Cl
count	284142	284142.000000	284142.000000	284142	284142	284142	284142	284142	284142	284142	...	247269	253653	2
unique	36046	NaN	NaN	59	59	1	7	39	39	2	...	7	54	
top	BRFSS~2022~2022~42~Q03~TMC01~AGE~RACE	NaN	NaN	US	United States, DC & Territories	BRFSS	Overall Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	...	White, non-Hispanic	POINT (-120.1550313 44.56744942)	
freq	15	NaN	NaN	6132	6132	284142	96753	11092	11092	262048	...	36450	5916	
mean	NaN	2018.596065	2018.657735	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
std	NaN	2.302815	2.360105	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
min	NaN	2015.000000	2015.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
25%	NaN	2017.000000	2017.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
50%	NaN	2019.000000	2019.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
75%	NaN	2021.000000	2021.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	
max	NaN	2022.000000	2022.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	

11 rows × 31 columns

Step3 : Drop columns that aren’t useful.

To make the dataset cleaner to work with it we drop the columns that aren’t useful.

df.columns command is used to list down all the columns.

Using this command we will list down the column names and we will pass it to the next command to drop the columns.

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.

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

```
[7] 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')
```

```
columns_to_drop = ["RowId", "LocationDesc", "Data_Value_Footnote_Symbol", "Data_Value_Footnote", "Geolocation"]
df.drop(columns_to_drop, axis = 1, inplace = True)
```

```
[9] 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.

Step 4 : Drop rows with maximum missing values.

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.

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	192808.000000	192808.000000	192597.000000	192597.000000	284142.000000
mean	2018.596065	2018.657735	37.676757	37.676757	33.027824	42.595333	800.322677
std	2.302815	2.360105	25.213484	25.213484	24.290016	26.156408	2511.564977
min	2015.000000	2015.000000	0.000000	0.000000	-0.700000	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

To remove the max missing data rows we follow the below steps:

1) Create a column called missing_count where the sum of all the cells having null values is stored.

Command: df["missing_count"] = df.isnull().sum(axis=1)

```
[10] df["missing_count"] = df.isnull().sum(axis=1)
```

2) The maximum value from this missing_count column is considered for how many rows we have to delete by checking how much it will affect the data and how much it will help in cleaning the data.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Command: max_missing = df["missing_count"].max()

```
[13] max_missing = df["missing_count"].max()

[14] print(max_missing)
```

3) Finally, we update the dataset by keeping the rows which have missing values less than a particular value.
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.

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

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

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%)

```
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	33.027824	42.595333	1145.457766	0.383573
std	2.301531	2.357772	25.213484	25.213484	24.290016	26.156408	2959.026567	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	32.800000	32.800000	27.000000	38.900000	35.000000	0.000000
75%	2021.000000	2021.000000	56.900000	56.900000	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

Step 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:
i) 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**.
ii) 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.

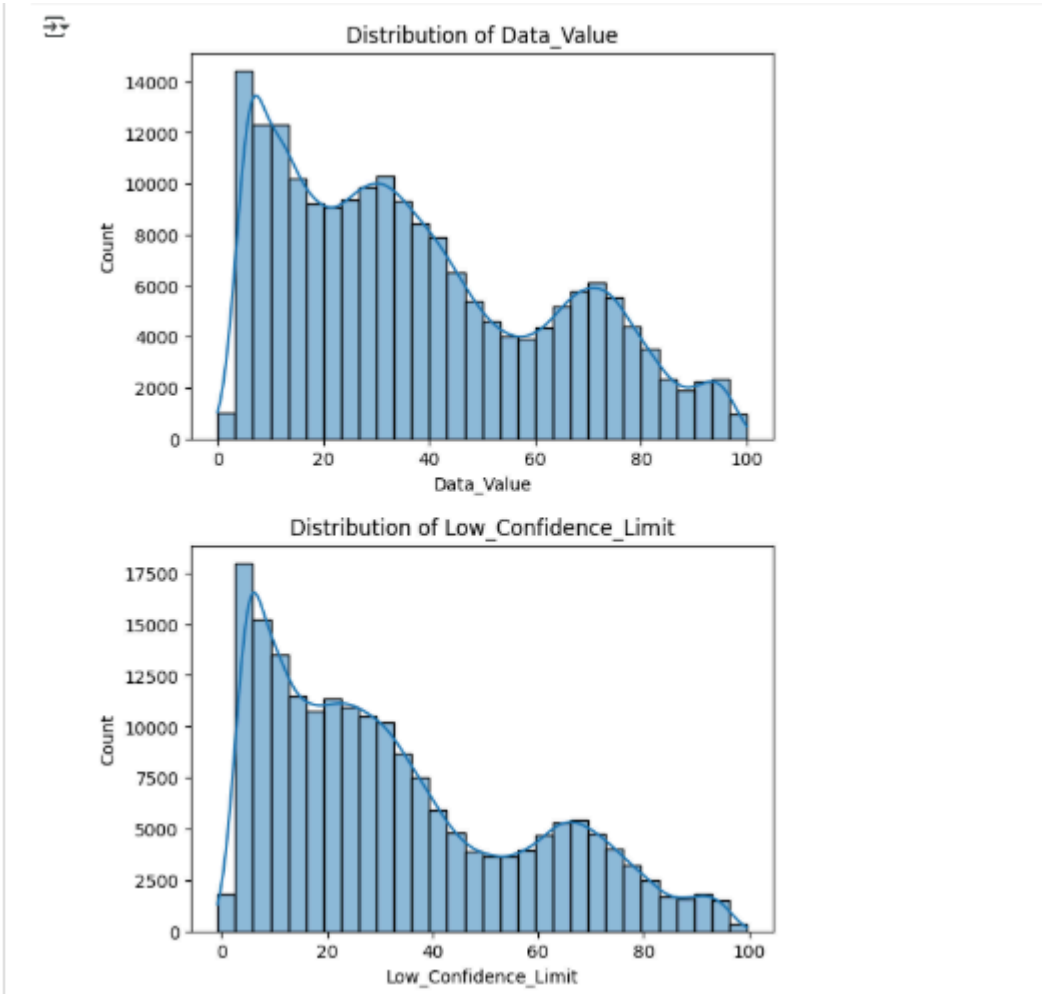
Follow the below steps to get how to fill the missing data:
i) **Check for skewness**

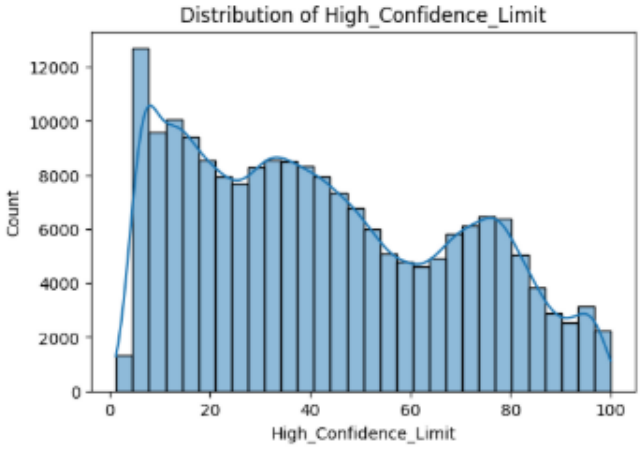
```
import seaborn as sns
import matplotlib.pyplot as plt
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
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()
```





Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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)

```
df["Data_Value"].fillna(df["Data_Value"].median(), inplace=True)
```

```
<ipython-input-21-49989ed8df82>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation ir


  df["Data_Value"].fillna(df["Data_Value"].median(), inplace=True)
```

```
[22] df["Low_Confidence_Limit"].fillna(df["Low_Confidence_Limit"].median(), inplace=True)
```

```
<ipython-input-22-a155792c99bd>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation ir


  df["Low_Confidence_Limit"].fillna(df["Low_Confidence_Limit"].median(), inplace=True)
```

```
[23] df["High_Confidence_Limit"].fillna(df["High_Confidence_Limit"].median(), inplace=True)
```

```
<ipython-input-23-e8d37b371347>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation ir


  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)

```
df["StratificationCategory2"].fillna("Data Unavailable", inplace=True)
df["Stratification2"].fillna("Data Unavailable", inplace=True)
```

```
<ipython-input-32-1f25356b9b45>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assi
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting \

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]


  df["StratificationCategory2"].fillna("Data Unavailable", inplace=True)
<ipython-input-32-1f25356b9b45>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assi
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting \

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]


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

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

```
df.head(80)
```

	YearStart	YearEnd	LocationAbbr	Datasource	Class	Topic	Question	Data_Value_Unit	DataValueTypeID	Data_Value_Type	...	Stratification2	ClassID	TopicID
3	2022	2022	MD	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	PRCTG	Percentage	...	Black, non-Hispanic	C05	TMC01
4	2022	2022	WI	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	PRCTG	Percentage	...	Male	C05	TMC01
6	2022	2022	OK	BRFSS	Mental Health	Frequent mental distress	Percentage of older adults who are experiencin...	%	PRCTG	Percentage	...	Native Am/Alaskan Native	C05	TMC01

Step 6 : Create dummy variables.

It is essential to create dummy variables for 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 output see the new columns added in the list of column.

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

```
df_dummies.columns
```

```
Index(['YearStart', 'YearEnd', 'Datasource', '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=112)
```

Step 7 : Find the 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.

There are 2 ways to find the outliers:

Method 1: Box-Plot

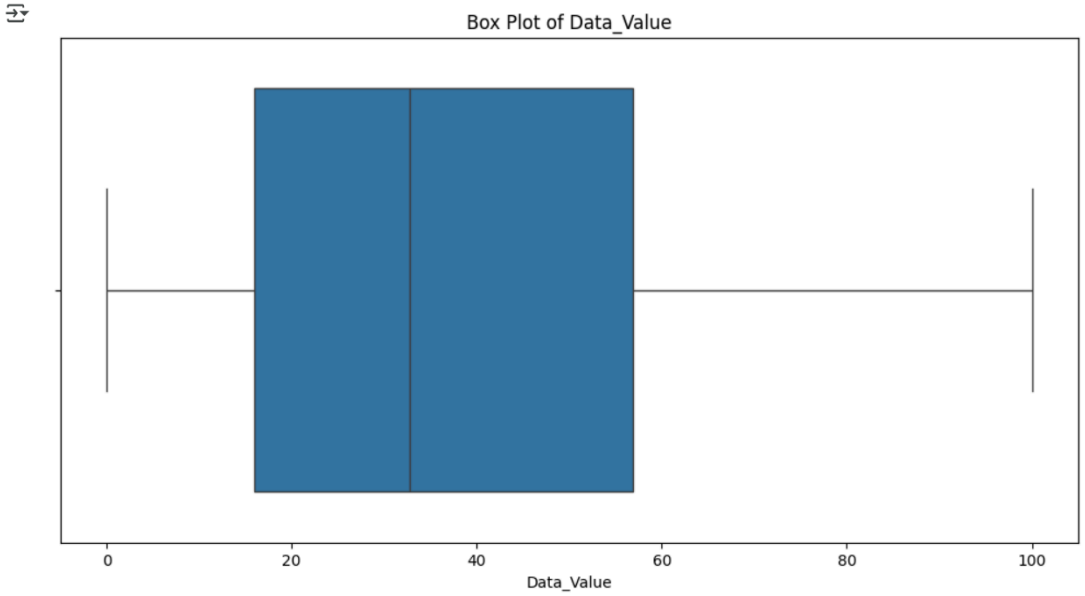
In this method, we use the column values to plot a box-plot graph. The values are usually in a box having lower and higher limits. If any outliers present, they come out of the box of the graph.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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()
```

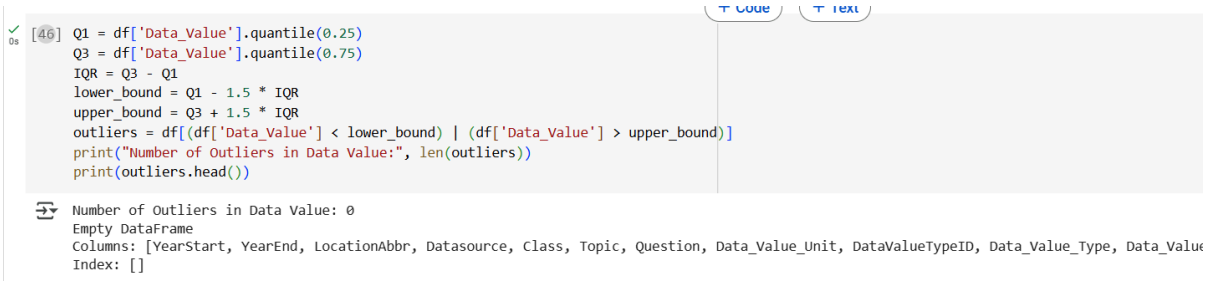


Method 2: Using IQR Value.

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())
```



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

Step 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']])
```

```
[48] df["Standardized_Data_Value"] = (df["Data_Value"] - mean_value) / std_value
```

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

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

	Data_Value	Standardized_Data_Value	Standardized Data Value Scalar
3	9.0	-1.137358	-1.137361
4	5.6	-1.272206	-1.272210
6	21.5	-0.641591	-0.641593
7	10.0	-1.097697	-1.097700
8	39.9	0.088177	0.088177

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']])
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

```
[51] 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)
```

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

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

	Data_Value	Data_Value_Normalized	Normalized Data Value Scalar
3	9.0	0.090	0.090
4	5.6	0.056	0.056
6	21.5	0.215	0.215
7	10.0	0.100	0.100
8	39.9	0.399	0.399

Now to save the Changes in new file .

```
[54] df.to_csv("/content/drive/MyDrive/Semester 6/AIDS/AIDS Lab/Alzheimer_s_Disease_and_Healthy_Aging_Data_Updated.csv", index=False)
```

Conclusion:

We pre-processed the Alzheimer's disease and Healthy Aging dataset. To load the data, we used the pandas `read_csv()` function and checked the first five entries with the `head()` function.
For an overview of the data, we used methods like `head()`, `info()`, and `describe()` to get details about data types, mean, max, min, count, and other statistics.
Next, we dropped unnecessary columns from the dataset using the `drop()` function. These included columns like RowId, LocationDesc, Data_Value_Footnote_Symbol, Data_Value_Footnote, and Geolocation, as they wouldn't contribute much to the analysis.
To handle missing data, we identified and removed rows with the most missing values. This process reduced the dataset from 284,142 entries to 192,808 (~32.14%).
For the remaining missing values, we analyzed the data and used appropriate methods (mean, median, or mode) to fill them in.
Columns like Questions and LocationAbbr were causing issues during analysis, so we converted them into dummy variables (with 0s and 1s) to prevent errors.
To identify outliers, we created a boxplot, which helped us spot values outside the typical range. We then used the IQR method to further analyze and remove the outliers.
Finally, to avoid large values skewing the analysis, we normalized and standardized the data using min-max and standard deviation methods, bringing the values into a reasonable range for smoother analysis.`

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Aim: Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.Perform following data visualization and exploration on your selected dataset.

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

Steps:

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

Bar Graph: A bar graph will be plotted to visualize the relationship between a categorical feature (Class) and a numerical feature (Data_Value). The mean of Data_Value for each category will be represented.

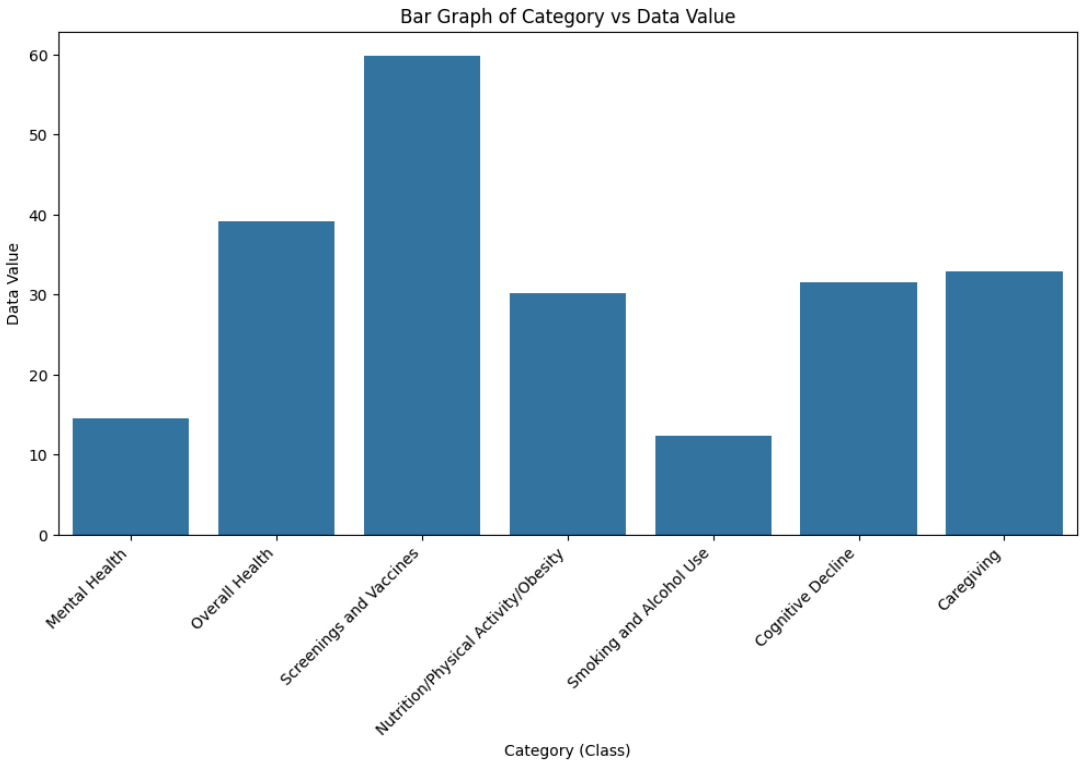
Contingency Table: A contingency table (cross-tabulation) will be generated using two categorical features (Region and Class), followed by a heatmap representation.

1) **Bar graph:**

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 Value")
plt.show()
```

Result:



Inference: The bar graph compares different classes based on their Data_Value. Taller bars mean higher values for that class, while shorter bars mean lower values. This makes it easy to see which classes have higher or lower data points.

2) Contingency table:

Code:

```
contingency_table = df.groupby(["Region", "Class"]).size().unstack()
print(contingency_table)
```

Result:

```
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
```

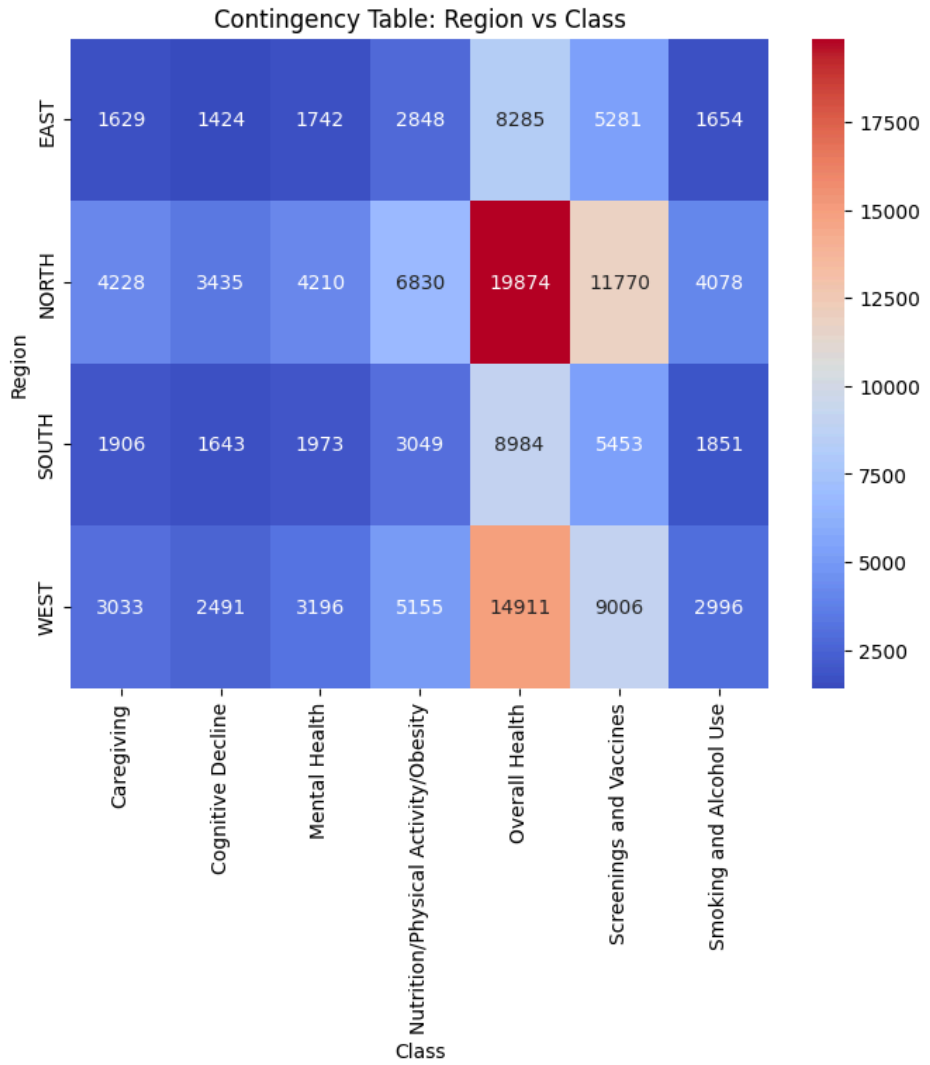
Code:

```
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()
```

Result:

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



Inference: The contingency table helps compare two categories, Region and Class, by showing how often they appear together. It helps find patterns, like whether certain regions have more cases of a particular class. This makes it useful for understanding relationships between different groups.

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

Scatter Plot: A scatter plot will be used to show the relationship between two numerical features.

Box Plot: The distribution of Data_Value will be analyzed using a box plot, which also helps in identifying outliers.

Heatmap: A heatmap will be created to visualize correlations between numerical features in the dataset.

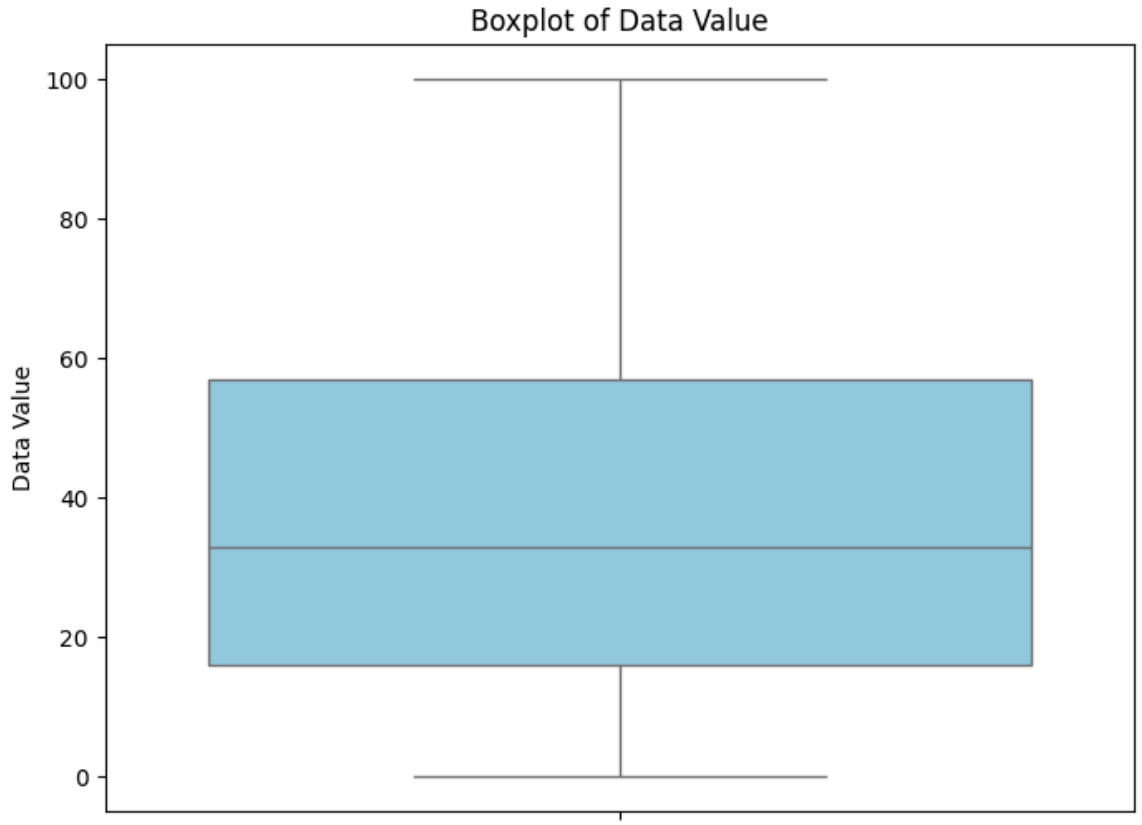
Box Plot:

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()
```

Result:



Inference: The boxplot gives a summary of how Data_Value is spread and helps identify extreme values. The box represents the middle 50% of the data, while the whiskers extend to the minimum and maximum within a range. Any points outside this range are considered outliers, showing if the data has any unusual values.

Step 3: Create histogram and normalized Histogram.

Histogram: A histogram will be plotted to analyze the frequency distribution of Data_Value.

Normalized Histogram: A probability density histogram will be created to normalize the distribution and verify its total area.

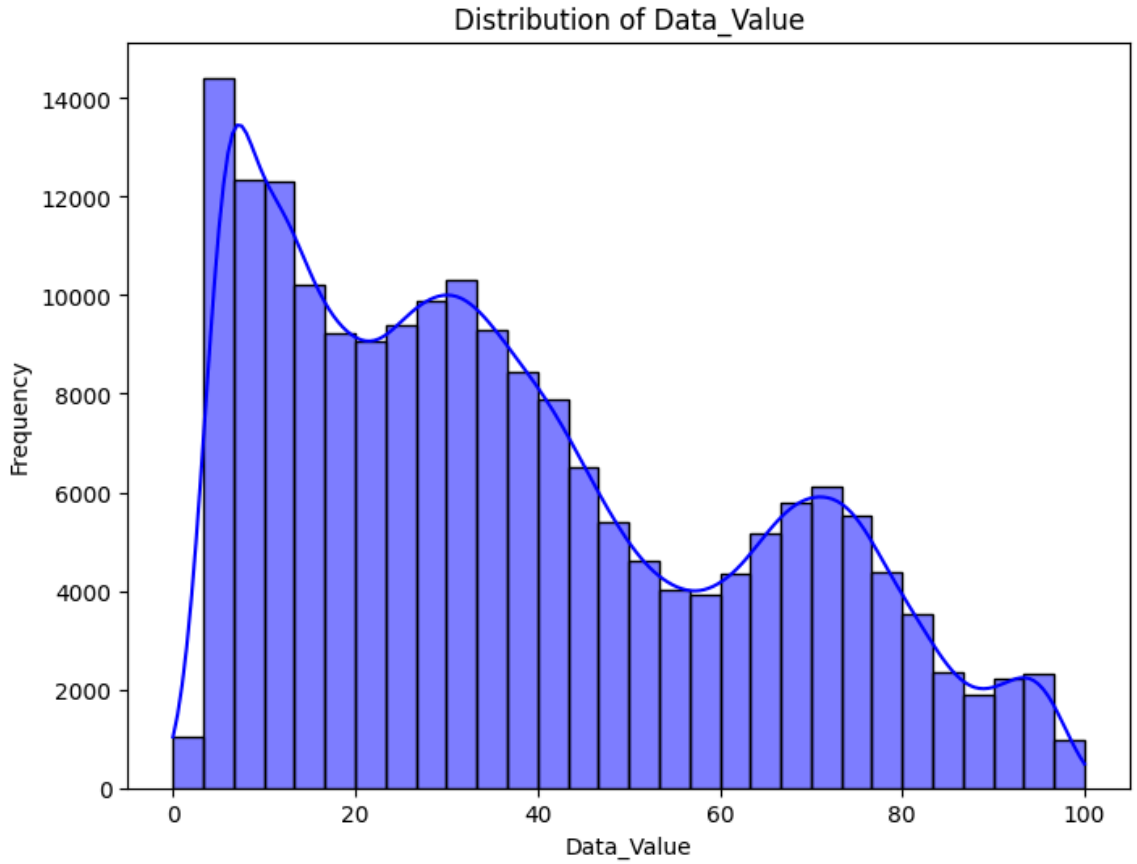
1)Histogram:

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()
```

Result:



Inference: The frequency distribution graph shows how often different Data_Value ranges appear in the dataset. If certain ranges have higher bars, it means more data points fall within them. This helps in spotting patterns, such as clusters of values or unusual gaps.

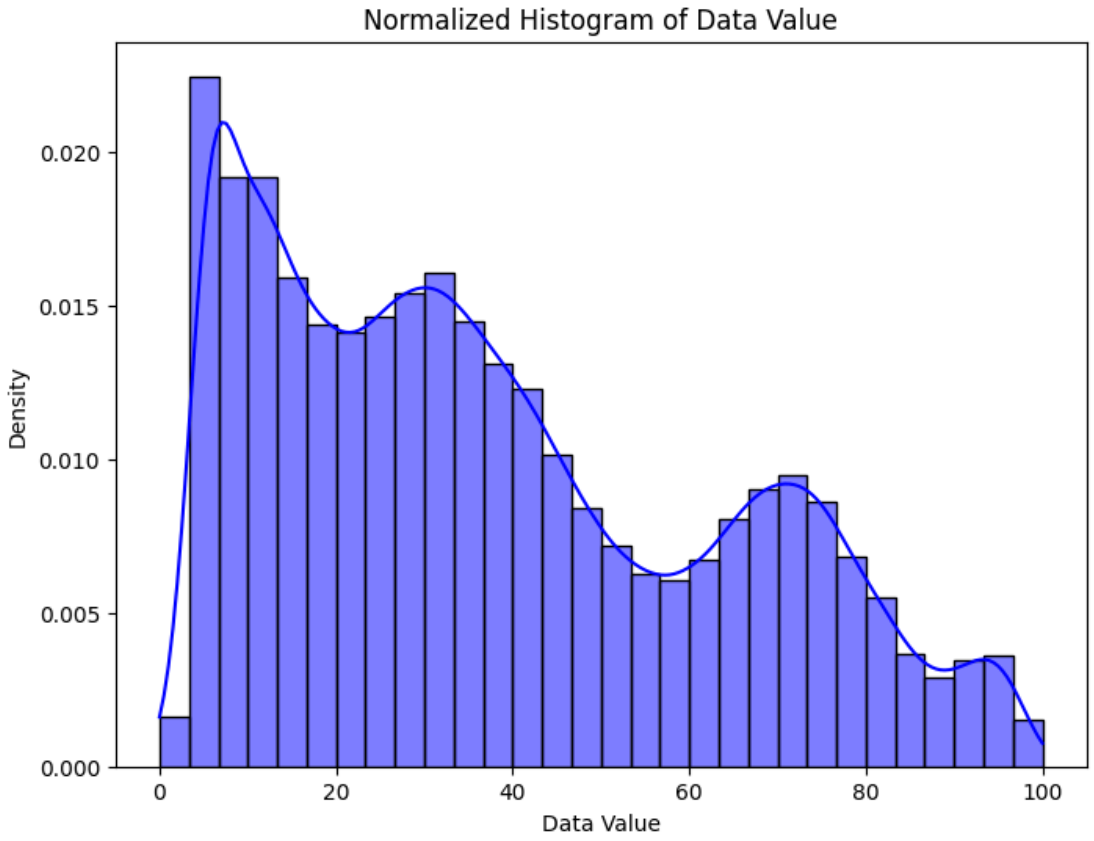
2) Normalized Histogram:

Code:

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

# Plot histogram and store bin heights and edges
plt.figure(figsize=(8,6))
hist_data = sns.histplot(df["Data_Value"], bins=30, kde=True, stat='density', color="blue")
```

```
# Extract heights of bars
heights = [patch.get_height() for patch in hist_data.patches]

# Extract bin widths
bin_width = hist_data.patches[1].get_x() - hist_data.patches[0].get_x()
# Compute total area
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()
```

Result:



Inference: The normalized histogram shows how Data_Value is distributed across different ranges. The x-axis represents the values, while the y-axis shows their probability, making sure the total area equals one. This helps in understanding whether the data follows a normal pattern or is unevenly spread.

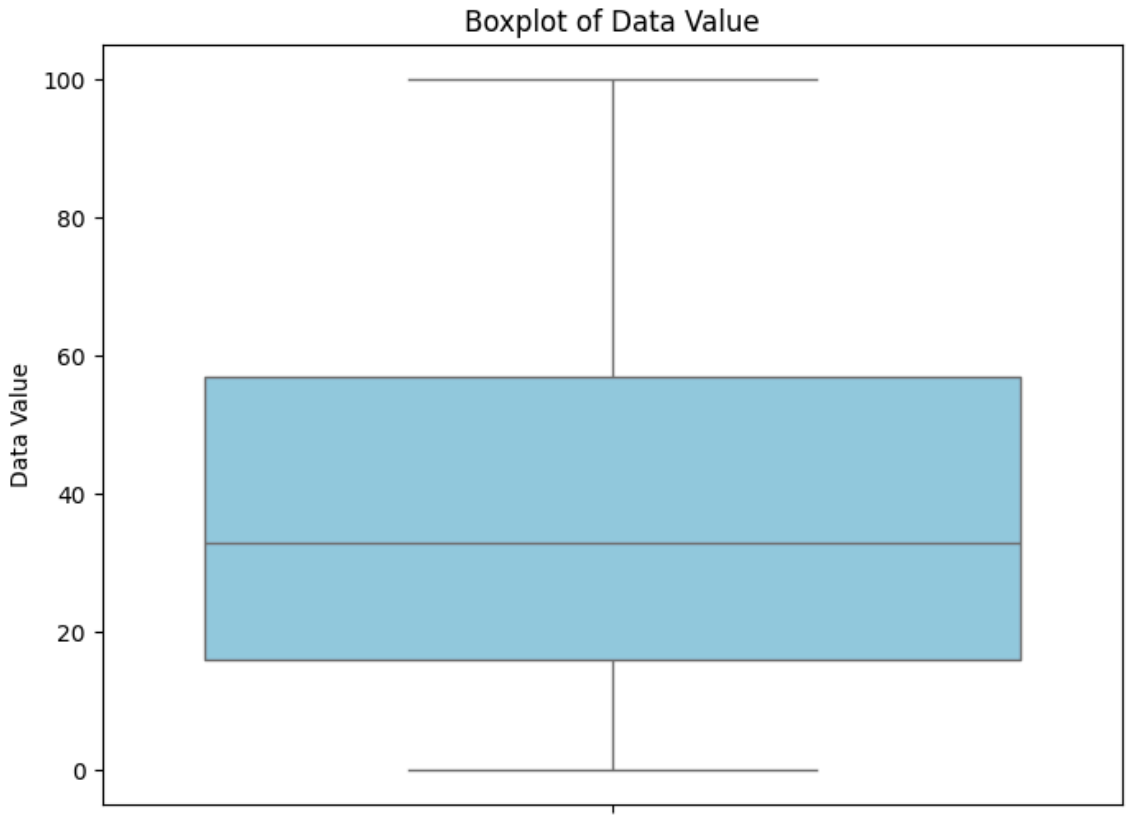
Step 4: Handle outlier using box plot and Inter quartile range.

Box Plot Analysis: Outliers will be detected using box plots.
Interquartile Range (IQR) Method: The IQR method will be applied to determine the lower and upper bounds for identifying and handling outliers.

1) **Box Plot:**
   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()
```

   Result:



Inference: The boxplot gives a summary of how Data_Value is spread and helps identify extreme values. The box represents the middle 50% of the data, while the whiskers extend to the minimum and maximum within a range. Any points outside this range are considered outliers, showing if the data has any unusual values.

2) **Interquartile Range:**

Code:

```
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))
```

Result:

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

Inference: The inference from the Interquartile Range (IQR) analysis is that the dataset does not contain any significant outliers. This suggests that the data values are well-distributed within the expected range and do not have extreme deviations. Since outliers can impact statistical analysis and model performance, the absence of outliers indicates that the dataset is stable and reliable for further analysis without requiring additional preprocessing to handle extreme values.

Conclusion: This analysis helps in understanding the distribution, relationships, and anomalies in the dataset. The findings will provide meaningful insights for further decision-making and model building.
The exploratory data analysis (EDA) revealed significant insights into the dataset's distribution, relationships, and anomalies. The bar graph and contingency table highlighted categorical trends, while scatter plots and heatmaps uncovered correlations between numerical features. Box plots and histograms provided a deeper understanding of data distribution, identifying potential outliers. Using the Interquartile Range (IQR) method, outliers were detected and handled to improve data quality. Overall, this analysis helped in feature selection, pattern recognition, and data preprocessing, ensuring a more refined dataset for future modeling and decision-making.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Aim: Perform Data Modeling.

- 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.
- Use a bar graph and other relevant graphs to confirm your proportions.
- Identify the total number of records in the training data set.
- Validate partition by performing a two-sample Z-test.

Steps:

Step 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.

We split the dataset into two parts: 75% for training (to teach the model) and 25% for testing (to check its accuracy). This ensures the model learns from most data while being tested on unseen data.

Code:

```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)
print(f"Total records: {len(df)}")
print(f"Training records: {len(train_df)}")
print(f"Testing records: {len(test_df)}")
```

Result:

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

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

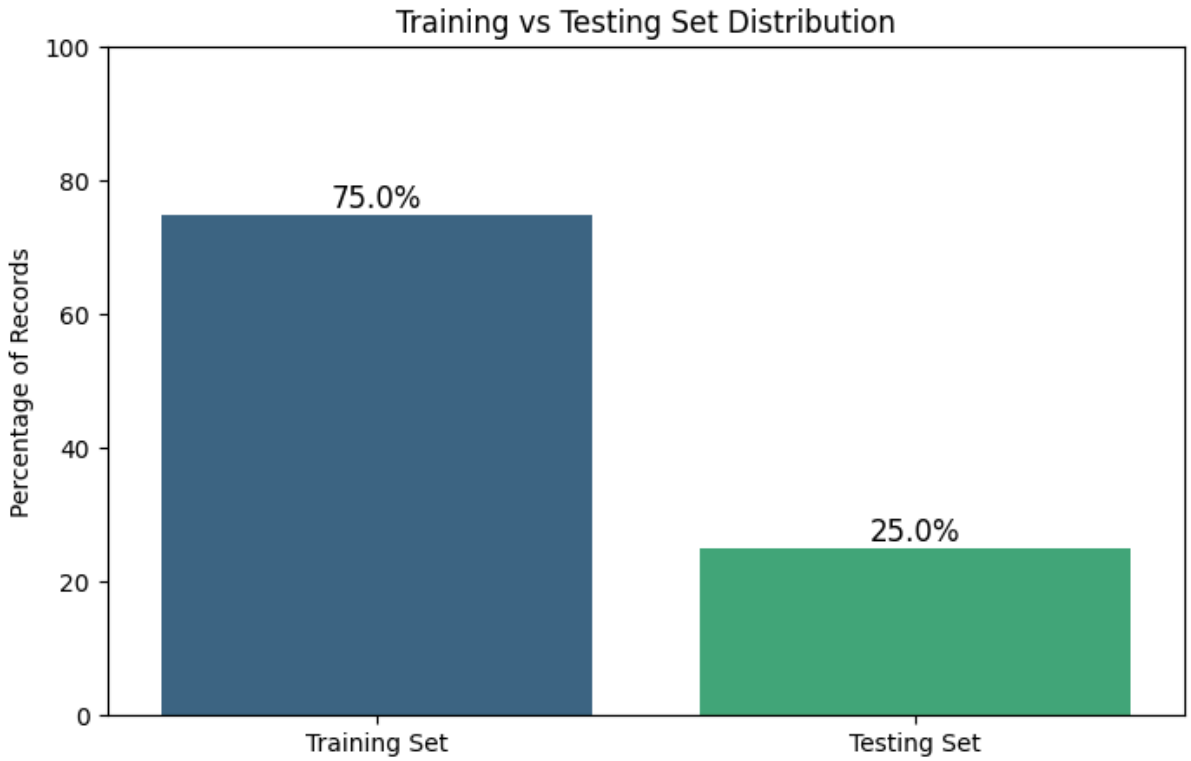
Graphs help us confirm that the data is split correctly. We use bar and pie charts to show the proportion of training and testing data clearly.

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()
```

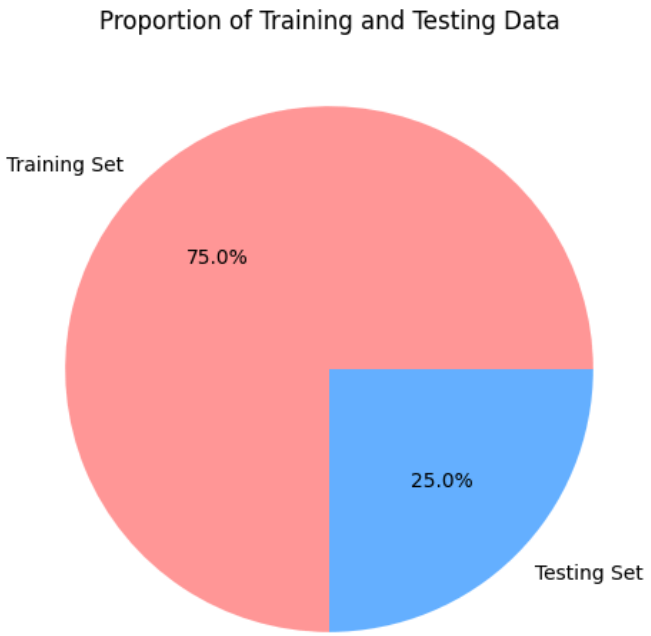
Result:



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()
```

Result:



Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 3: Identify the total number of records in the training 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)
print(f"Total records: {len(df)}")
print(f"Training records: {len(train_df)}")
print(f"Testing records: {len(test_df)}")
```

Result:

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

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

A two-sample Z-test checks if the training and testing datasets have similar characteristics. It compares the mean values to ensure the split is balanced and doesn't 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.")
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Result:

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

Conclusion :

In this experiment, we successfully partitioned the dataset into training and testing sets, ensuring a balanced distribution. We used graphs to visualize the split and performed a Z-test to confirm there is no significant difference in data characteristics. The results showed that the partitioning is valid, meaning the dataset is well-prepared for modeling.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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:

Step 1: Load the dataset using Pandas Library and Import some other Libraries

Code:

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

file_path = "/content/drive/MyDrive/Semester 6/AIDS/AIDS
Lab/cleaned_combined.csv"
df = pd.read_csv(file_path)
```

Step 2: Extract numeric columns from the dataset

Code:

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

Step 3: Pearson's Correlation Coefficient

Pearson's correlation is used to measure the strength and direction of a linear relationship between two numerical variables. Here, we test whether there is a relationship between Flight Distance and Arrival Delay in Minutes.

Hypothesis:

H0 (Null Hypothesis): There is no linear relationship between Flight Distance and Arrival Delay.
H1 (Alternative Hypothesis): There is a linear relationship between Flight Distance and Arrival Delay.

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")
```

Result:

```
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
```

The Pearson correlation coefficient is -0.0020, which is very close to zero, indicating almost no linear relationship between flight distance and arrival delay. The p-value is 0.47708, which is greater than 0.05, meaning the correlation is not statistically significant.

This result confirms that longer flights do not necessarily result in longer delays, and flight distance alone cannot be used to predict arrival delay.

Step 4: Spearman’s Rank Correlation

Spearman’s correlation is useful when analyzing relationships that may not be linear but still follow an increasing or decreasing trend. It measures how well the ranks of two variables correspond to each other.

Hypothesis:

H0 (Null Hypothesis): There is no monotonic relationship between Flight Distance and Arrival Delay.
H1 (Alternative Hypothesis): There is a monotonic relationship between Flight Distance and Arrival Delay.

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")
```

Result:

```
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
```

The Spearman correlation coefficient is -0.0018, which is very close to zero, meaning there is no meaningful increasing or decreasing trend between flight distance and arrival delay. The p-value is 0.505755, which is greater than 0.05, proving that the correlation is not statistically significant.

This result confirms that changes in flight distance do not consistently influence arrival delays.

Step 5: Kendall’s Rank Correlation

Kendall’s correlation measures the strength of the ordinal association between two variables. It is useful for analyzing ranked data, especially when there are many tied values.

Hypothesis:

H0 (Null Hypothesis): There is no ordinal relationship between Flight Distance and Arrival Delay.
H1 (Alternative Hypothesis): There is an ordinal relationship between Flight Distance and Arrival Delay.

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")
```

Result:

```
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
```

The Kendall correlation coefficient is -0.0014, which is very close to zero, meaning there is no strong ordinal relationship between flight distance and arrival delay. The p-value is 0.504888, indicating that the correlation is not statistically significant.

This result suggests that ranking flights by their distance does not help in predicting the ranking of their delays.

Step 6: Chi-Squared Test

The Chi-Square test is used to check whether two categorical variables are dependent on each other. Here, we test if Customer Type (New vs. Loyal) affects Satisfaction Level (Satisfied vs. Not Satisfied).

Hypothesis:

H0 (Null Hypothesis): Customer Type and Satisfaction are independent.
H1 (Alternative Hypothesis): Customer Type and Satisfaction are dependent.

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")
```

Result:

```
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 Chi-Square value is 4493.1888, which is very large, indicating a strong relationship between customer type and satisfaction. The p-value is 0.0000, meaning the result is highly significant.

This result proves that customer type has a significant impact on satisfaction, with loyal customers showing higher satisfaction levels than new customers.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Conclusion:

The Pearson, Spearman, and Kendall correlation tests all showed that there is no significant relationship between flight distance and arrival delay. This confirms that longer flights do not necessarily lead to longer delays.

However, the Chi-Square test found a strong relationship between customer type and satisfaction, showing that loyal customers tend to be more satisfied than new ones. These insights can help airlines improve their customer service strategies based on passenger satisfaction trends.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 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 the above dataset.

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

Theory: Regression analysis is a statistical method used to examine the relationship between a dependent variable and one or more independent variables. Using libraries like **Scipy** and **Scikit-learn**, we can implement both linear and logistic regression models to predict outcomes or classify data. **Linear regression** predicts continuous values, while **logistic regression** handles binary classification problems. These models are trained on historical data and evaluated using metrics such as R^2, MSE, and accuracy.

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) Training 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 initialized and trained on the dataset using .fit(X, y). The coef_ attribute provides the slope, indicating the effect of days left on price, while intercept_ gives the y-intercept. These values are combined to form and display the regression equation summarizing the model.

3) Training and Testing and 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
```

The LinearRegression() class from the sklearn.linear_model module is used to implement linear regression, which estimates the relationship between a dependent variable and one or more independent variables. In this case, it predicts airline ticket prices (dependent variable) using the number of days left before departure (independent variable).

To ensure fair evaluation, the dataset is split into training and testing sets in an 80:20 ratio using train_test_split(). The model is trained using the .fit() method, and predictions are made on the test set. Performance is evaluated using metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2). The coefficient indicates how price changes with each additional day left, and the intercept represents the predicted price when no days are left.

If the R^2 score is low or the error values are high, it suggests that days_left alone is not a strong predictor of ticket prices. The model's linear equation is derived from the slope and intercept values.

4)Regression Line and Plot Graph

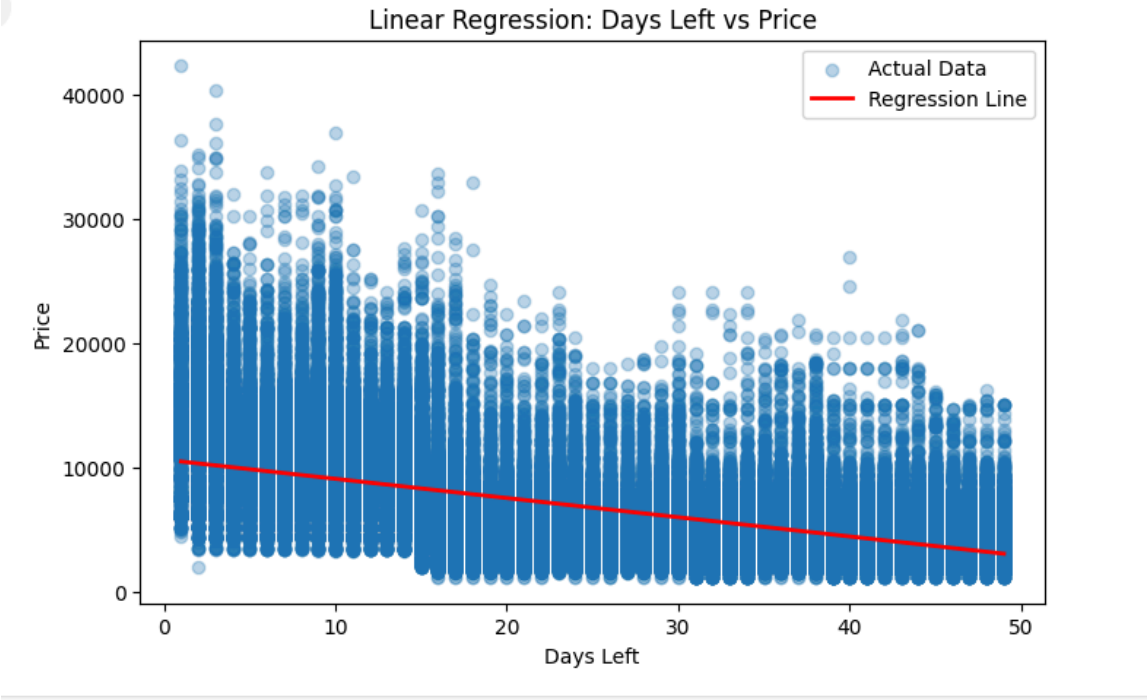
Code:

```
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()
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Output:



In this step, we visualize the output of the linear regression model through a scatter plot and a regression line. A range of values between the minimum and maximum of days_left is generated using np.linspace(), and the trained model predicts the corresponding prices, forming a smooth regression line.

With matplotlib.pyplot, the actual data is displayed as a scatter plot, and the predicted values are shown as a red line. The alpha=0.3 setting makes the data points semi-transparent to enhance visibility. Proper labeling illustrates the relationship between ticket price and days left, helping to visually determine if a linear pattern exists.

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)
```

The LogisticRegression() class from the sklearn.linear_model module is used to handle binary classification tasks. Here, X represents the input features and y represents the target labels. The data is divided into training and testing sets using train_test_split(), with 80% allocated for training and 20% for testing.
Because logistic regression is sensitive to the scale of input features, StandardScaler() is used to standardize the data, ensuring a mean of 0 and a standard deviation of 1. The model is then trained using the .fit() method, which determines the optimal weights to differentiate between the classes. These learned weights are used to estimate the probability of each data point belonging to a specific 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
```

Once the logistic regression model is trained, its performance is assessed using various metrics from the sklearn.metrics module. Predictions on the test data are made using the .predict() method. Accuracy is then calculated to determine the percentage of correct predictions out of all predictions.

The confusion matrix breaks down the results into true positives, true negatives, false positives, and false negatives, helping to evaluate prediction balance. The classification report provides detailed insights into each class through precision, recall, and F1-score. With an accuracy of 66%, the model shows moderate performance, but the results suggest it could be improved through tuning or by incorporating more relevant features.

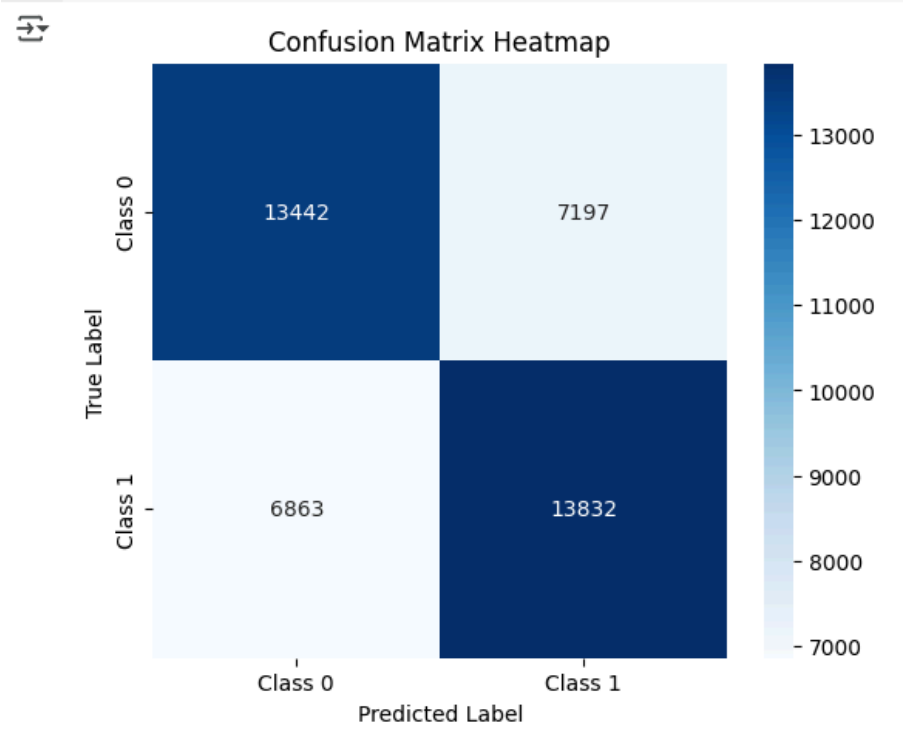
4) Visualize the Logistic Regression

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 how well the logistic regression model performs, Seaborn's heatmap() function is used to display the confusion matrix. This heatmap clearly shows the comparison between actual and predicted values, with darker shades representing higher counts.

Annotations are added using annot=True and formatted as integers with fmt='d', while the 'Blues' color map enhances visual clarity. The axes are labeled to distinguish predicted versus actual classes, and custom tick labels (Class 0, Class 1) make the chart easier to interpret. This visual helps quickly spot areas where the model performs well or needs improvement.

Below the heatmap, the classification report is displayed again, summarizing key metrics like precision, recall, and F1-score for each class.

Conclusion:

In the Logistic Regression model, the price_category variable was used to predict outcomes in the test portion of the dataset. The model achieved an accuracy of 66%, indicating that some predictions were incorrect. This is further confirmed by the confusion matrix heatmap, which reveals the presence of false positives and false negatives.

For Linear Regression, the numerical feature days_left was used to predict the price. After splitting the dataset, the model was trained and applied to the test set. Evaluation metrics like MSE (9425177) and R-squared (0.331) show a significant gap between predicted and actual values, which is also evident from the scatter plot visualization.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Steps:

Step 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 6/AIDS/AIDS Lab/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
```

```
[6] 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

Step 2: Data Preprocessing

1) Drop Unnecessary Columns

For the following experiment, the columns such as Unnamed and price are not required for classification. So to start preprocessing, we drop such columns using **df.drop(columns=[])** command.

```
[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
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

2) 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)
```

3) 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

4) 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
```

Step 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
```

Step 4: Train & Evaluate Classifiers

1)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	0.863818	0.854738	0.859254	33202.00000
1	0.822540	0.833240	0.827856	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

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

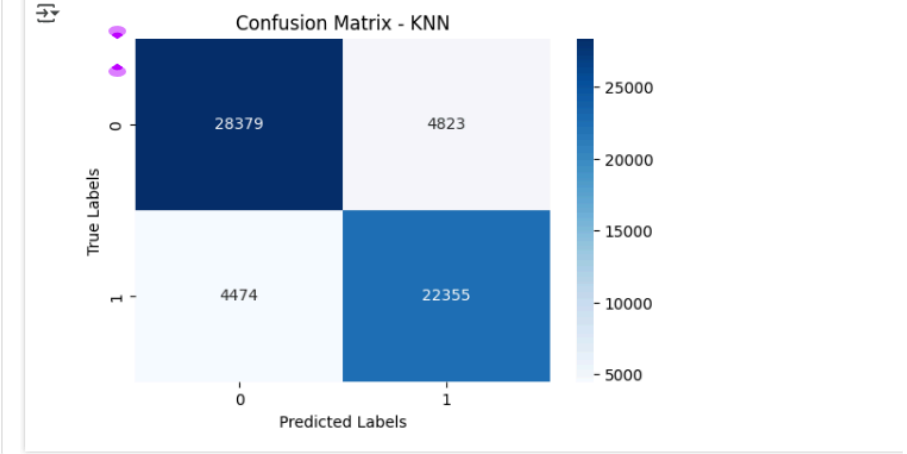
Division: D15C **Roll No: 28**

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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

2) 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
```

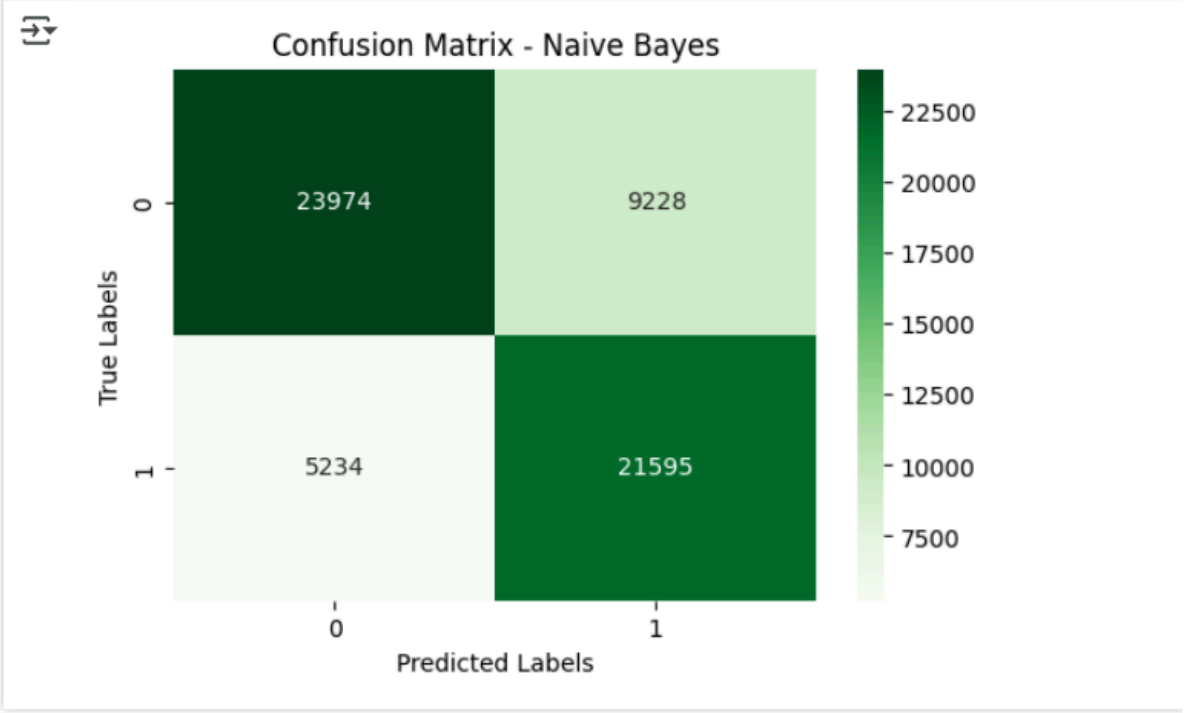
```
0.7590911362462728
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

3)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

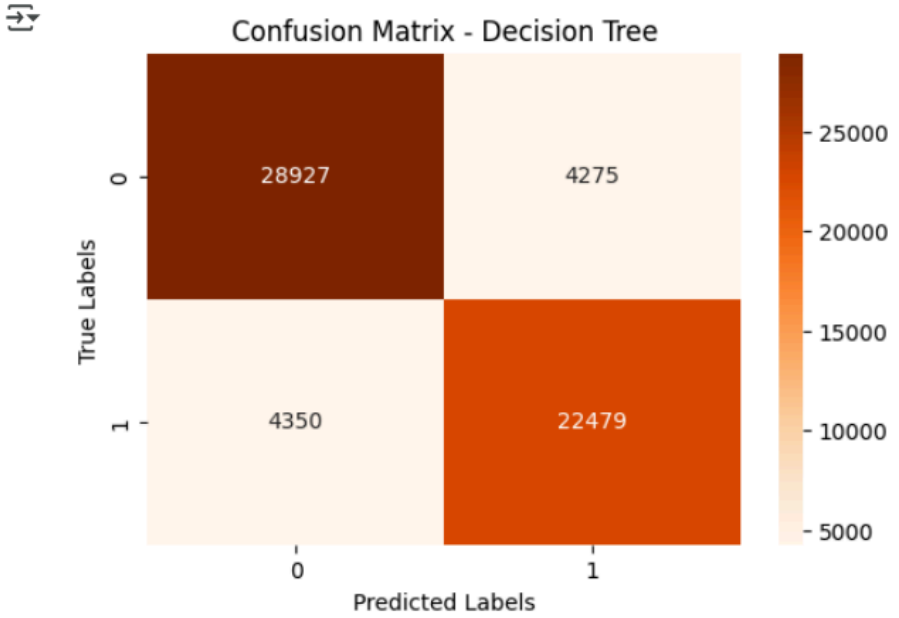
II) Accuracy

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

```
0.8563242324798854
```

III) Confusion Matrix

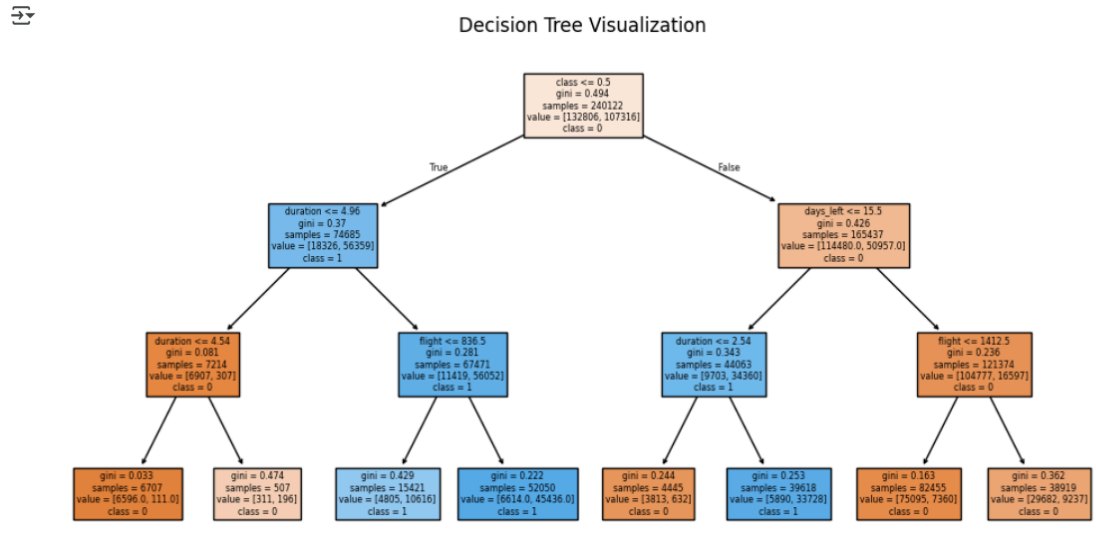
```
# Confusion Matrix
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)

# 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.

4)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
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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	0.666802	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
```

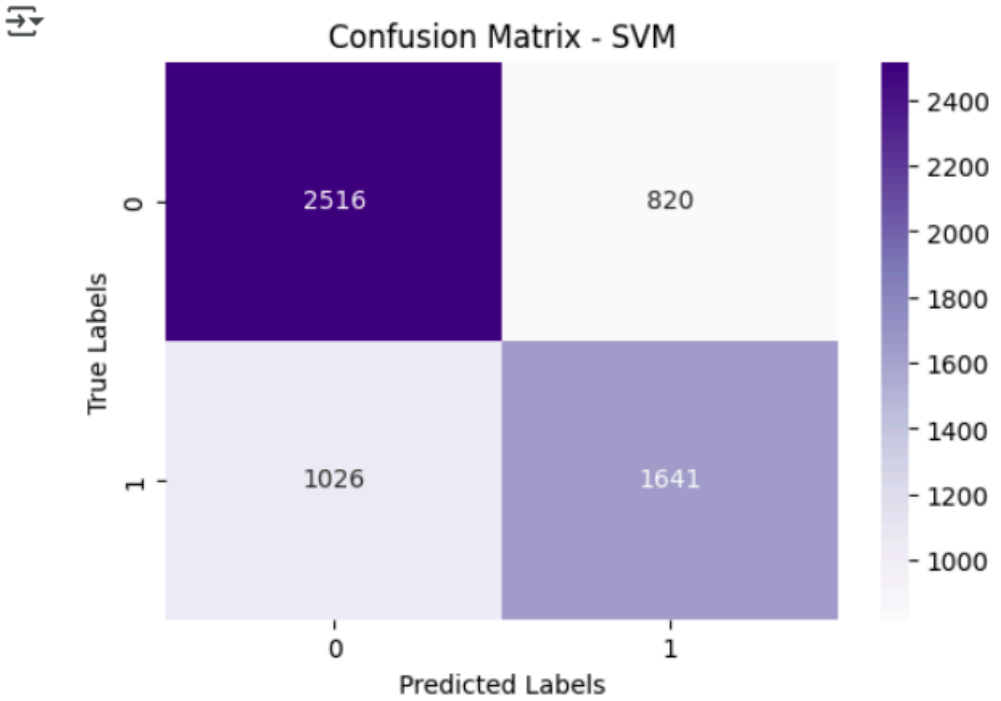
```
0.6924870897884391
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 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

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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 $d(P,Q) \leq \varepsilon$.
- 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:**

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

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

- **Average linkage:**

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

Steps :

Step 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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

Descriptive statistics revealed an average flight duration of ~12.2 hours and an average price of ₹20,889, with a wide price range from ₹1,105 to ₹1,23,071. This exploration set the foundation for further preprocessing and modeling.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Step 2: 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
```

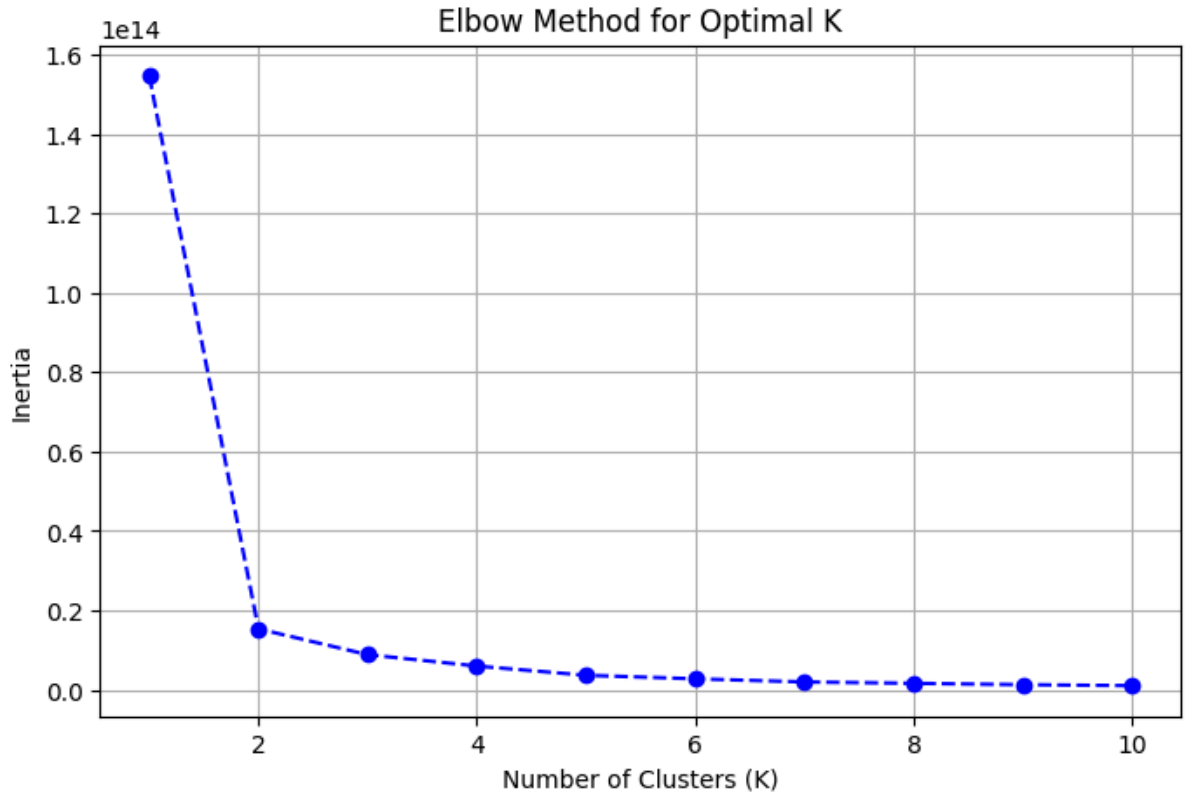
Unnecessary columns like 'Unnamed: 0' and 'flight' were dropped to simplify the dataset. Then, 8 categorical columns—including airline, source/destination cities, and travel class—were encoded using Label Encoding to convert them into numerical format suitable for machine learning models. This step ensured that the dataset was clean, compact, and fully numerical, preparing it for clustering and further analysis.

Step 3: 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()
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

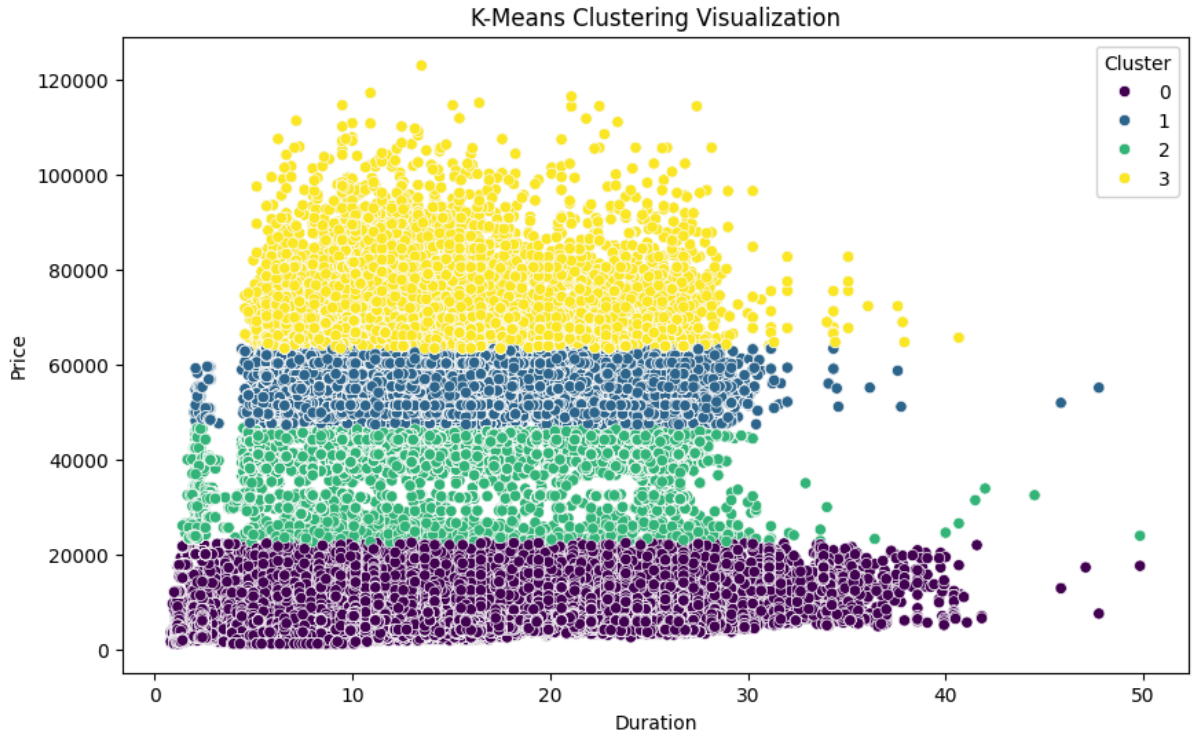
Division: D15C **Roll No: 28**



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()
```



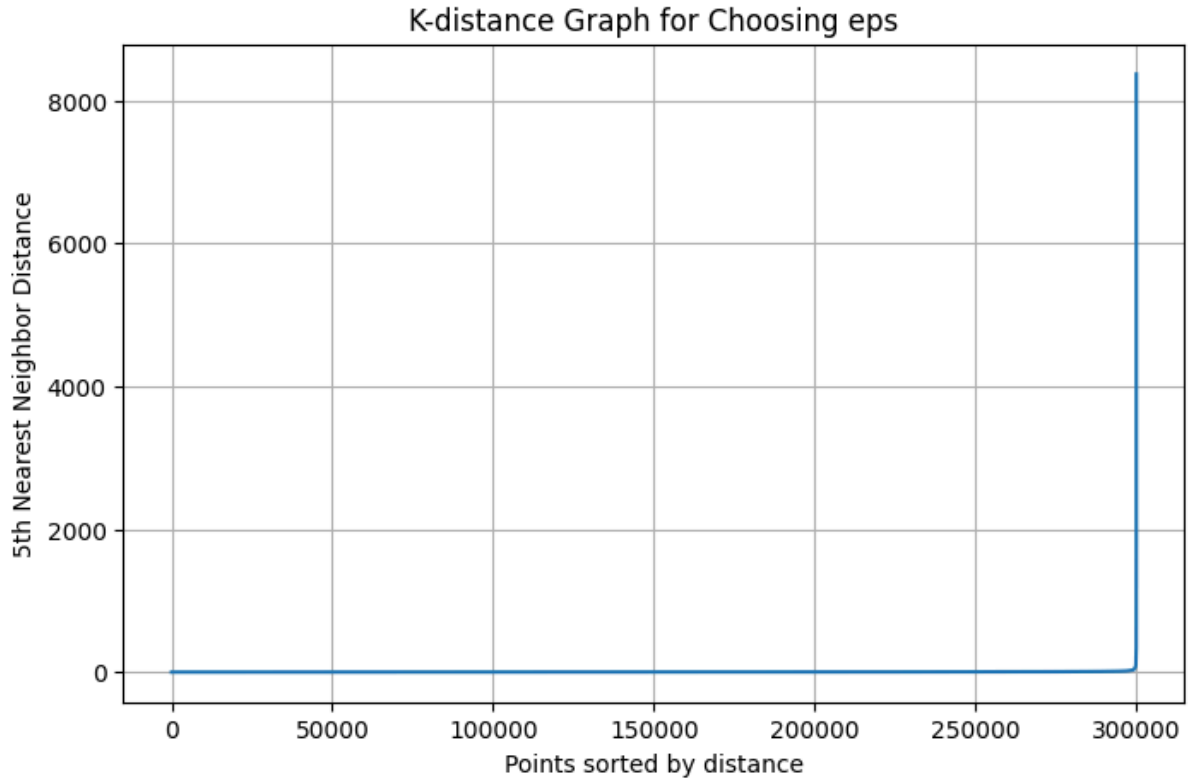
K-Means was applied with K=4 (from the elbow method) to segment flights based on duration, price, and days left. Each flight was assigned a cluster label, and the results were visualized using a scatter plot. The plot reveals four distinct price-based groupings, showing clear patterns in how flight duration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

Step 4: 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()
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

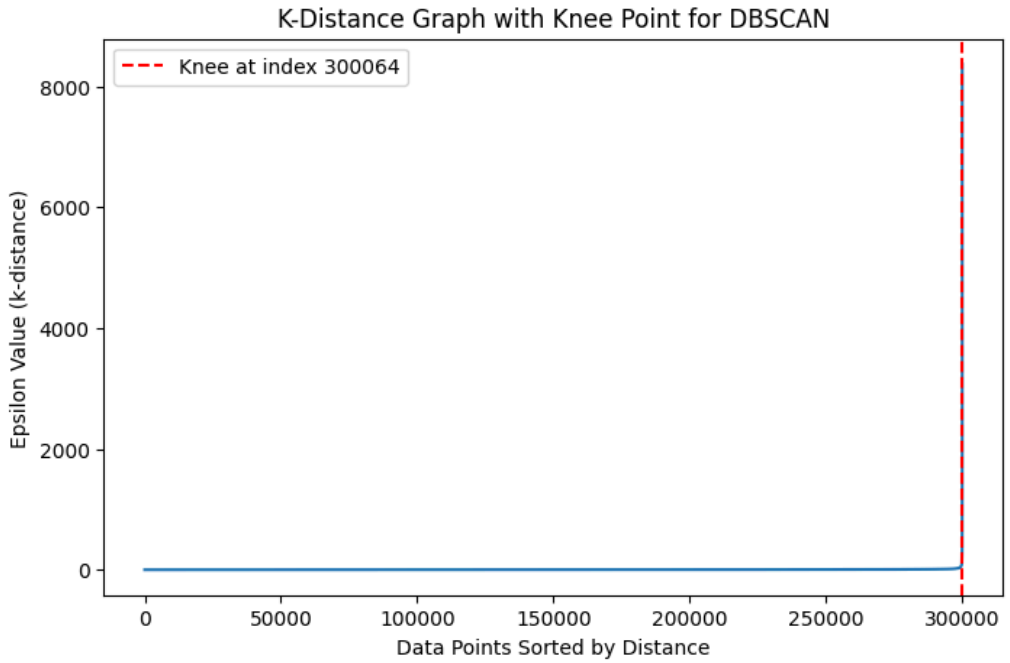
Division: D15C **Roll No: 28**




```
!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]}")
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

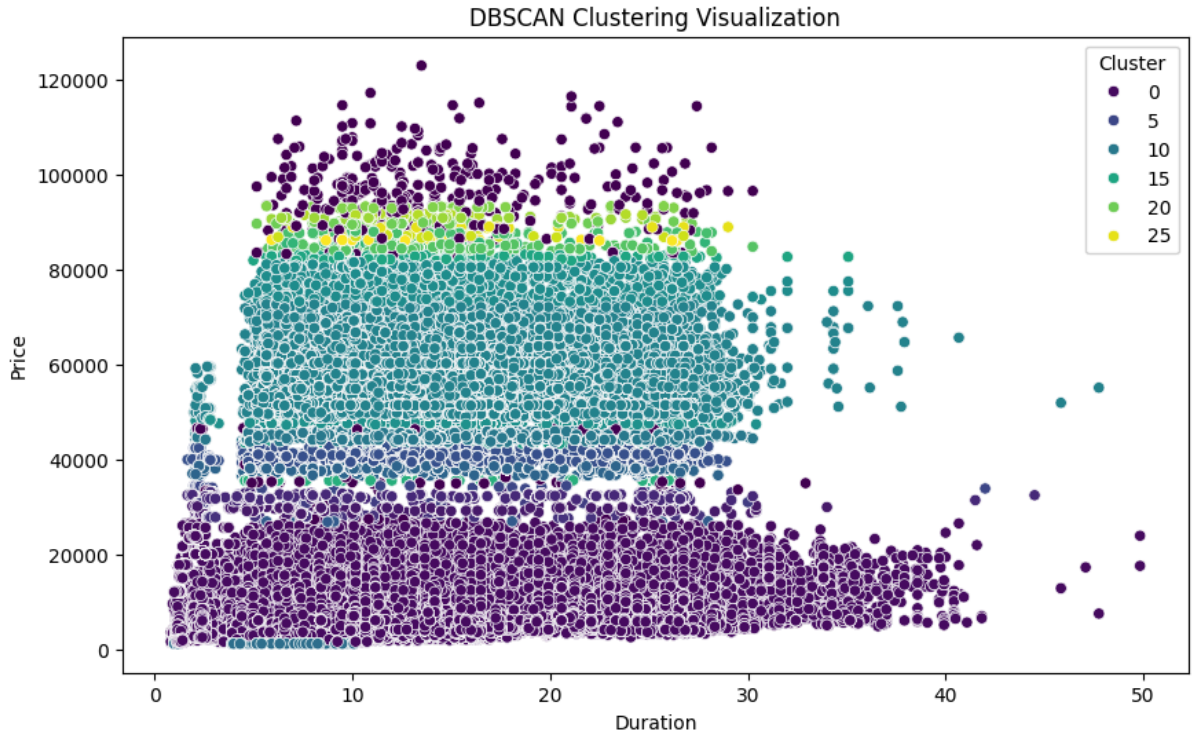
Division: D15C **Roll No: 28**



Suggested Epsilon (ε): 168.02811937291924

```
eps_value = 168.02811937291924
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=(10,6))
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 Visualization')
plt.legend(title="Cluster")
plt.show()
```

```
DBSCAN_Cluster
 0     210014
 11     59594
 10      7710
 6       6765
 8       3279
 12      3162
 2       2290
 13      1873
 1       1179
 3       1011
 5        502
-1        492
 16       428
 19       379
 9        307
 14       288
 15       275
 20       125
 18        76
 4         74
 22        73
 24        64
 17        51
 21        40
 23        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]
```



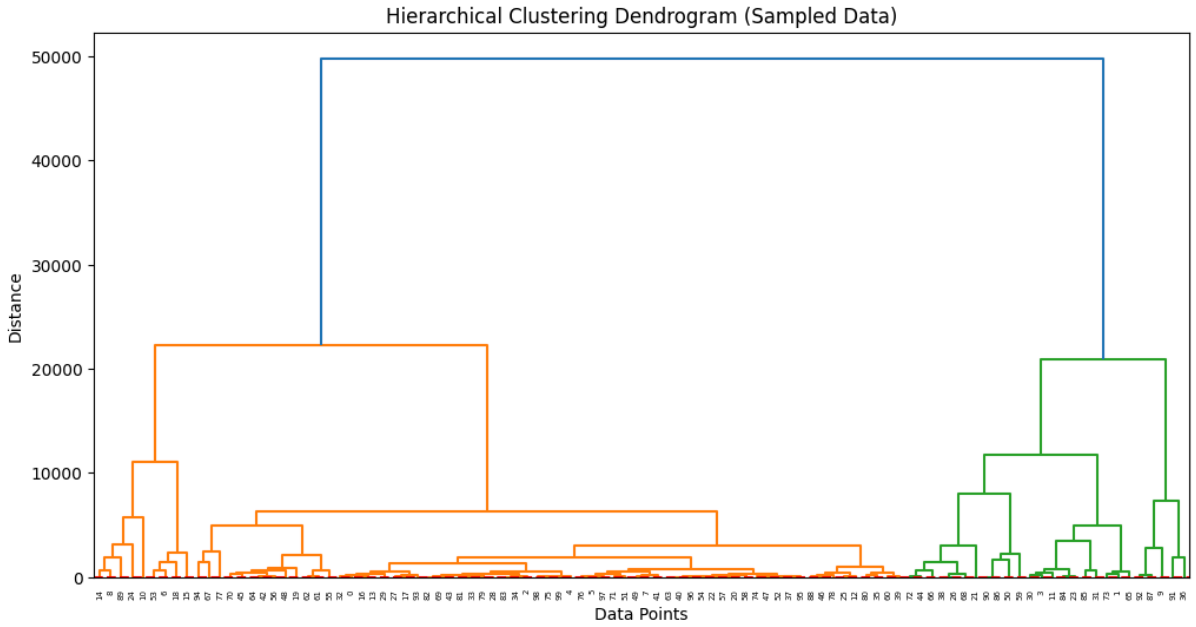
DBSCAN clustering was applied using an optimal epsilon value of **168.03**, determined from the k-distance graph. With min_samples set to 26, the algorithm detected **27 clusters**, including a few outliers labeled as **-1**. Most data points were grouped into dense clusters, clearly visible in the visualization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

Step 5: 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()
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



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 successfully implemented and analyzed **three clustering algorithms—K-Means**, **DBSCAN**, and **Hierarchical Clustering**—to perform **unsupervised classification**. Each algorithm was applied to a cleaned dataset and visualized using scatter plots and dendrograms.

- **K-Means** grouped data based on centroid distance, requiring the number of clusters as input.
- **Hierarchical Clustering** revealed nested data structures and was visualized with a dendrogram.
- **DBSCAN** automatically identified clusters based on data density and detected outliers effectively.

These methods demonstrated different strengths: K-Means is simple and fast, Hierarchical provides insight into data hierarchy, and DBSCAN handles noise and varying densities well. Overall, clustering proved to be a powerful tool for discovering hidden patterns in unlabeled data.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

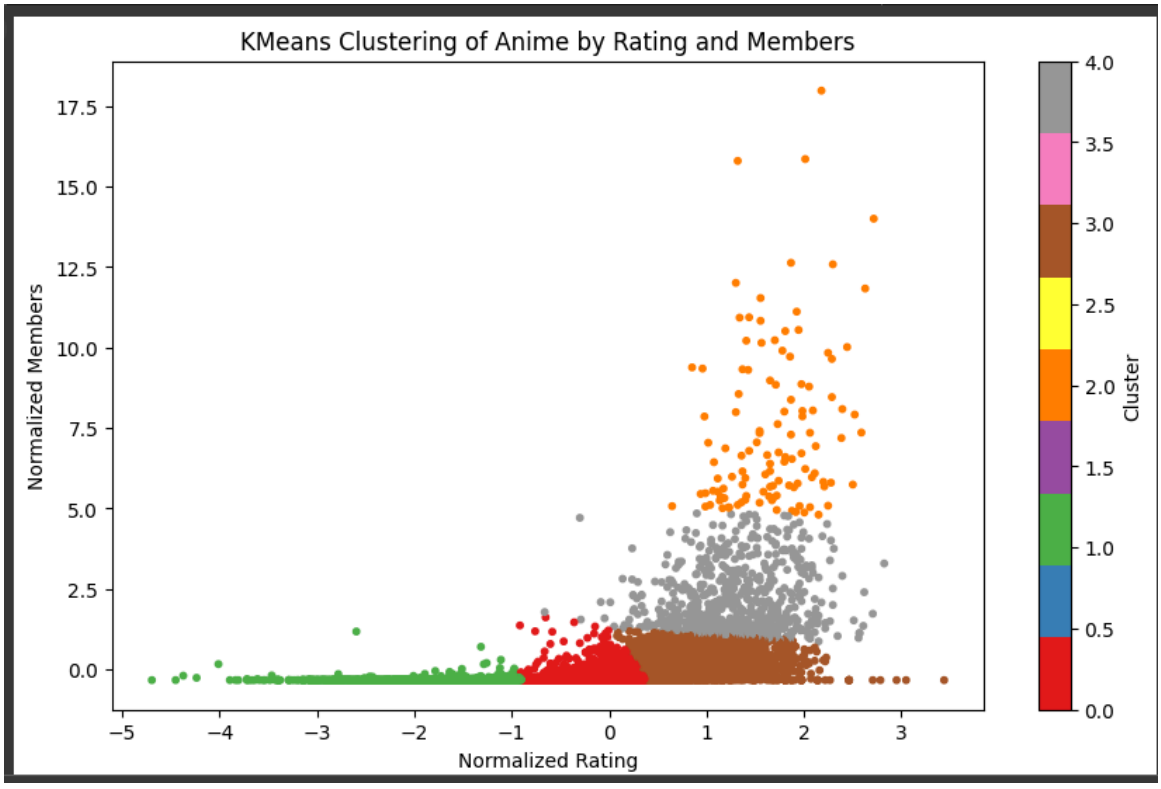
Code:

```
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:

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**



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)

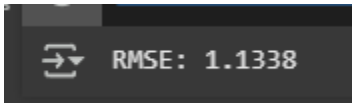
In this step, a recommendation system is constructed using the Singular Value Decomposition (SVD) algorithm from the Surprise library. A Reader object is first defined to specify the rating scale, which ranges from 1 to 10. The Dataset.load_from_df() function is then used to convert the cleaned ratings 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:



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 5tory of The 5ecret 5tar ...
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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 9

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

Theory:

1. What is Apache Spark and How Does It Work?
Answer:
Introduction to Apache Spark
Apache Spark is an open-source, unified analytics engine specifically built for processing large volumes of data at high speed. Originally developed at UC Berkeley, Spark quickly gained popularity due to its ability to handle complex data tasks with ease and efficiency.

It is a big data tool used in situations where traditional tools (like Excel, Pandas, or SQL) are not enough — for example, when working with gigabytes or terabytes of data, or when data is stored across multiple systems like cloud servers, Hadoop clusters, or databases.

Apache Spark:

1. **Category:** Big Data Processing Engine
2. **Use Case:** Handles large datasets that exceed single-machine memory
3. **Working:** Executes tasks in parallel across a cluster
4. **Strengths:** Scalable, fast (in-memory), fault-tolerant

How Apache Spark Works
Imagine you're organizing a massive event with thousands of tasks to do. If one person tries to handle everything, it's slow and inefficient. But if you divide tasks among a team, each person can handle a portion, and things get done much faster. That's exactly what Apache Spark does with your data.

- **Cluster-Based Architecture**
  Spark works on a cluster, which is a group of machines. One machine acts as the driver (manager) and the rest are workers (task doers). The driver coordinates the tasks, and the workers process chunks of data in parallel.

- **RDD (Resilient Distributed Dataset)**
  Spark's basic unit of data is called an RDD. It's like a big list that's split and stored across multiple computers. Each piece of the RDD can be worked on separately without affecting the others, which is why it's so fast and fault-tolerant.

- **In-Memory Computation**
  Unlike other tools that constantly write and read from hard drives, Spark processes data in memory (RAM). This makes it incredibly fast — often 10 to 100 times faster

than traditional big data tools like Hadoop MapReduce.

- **Lazy Evaluation**
  Spark doesn't immediately perform operations as they're written. It waits until the final result is needed and then runs the most optimized path to get there. This avoids unnecessary computations.

- **Rich API Support**
  Spark supports multiple programming languages — Python (PySpark), Scala, Java, and R — making it widely accessible for data analysts and engineers.

Use Cases

1. **Social Media Analysis:** This involves collecting and analyzing massive amounts of posts, comments, and interactions to identify trending topics, popular hashtags, and user sentiment in real-time.
2. **Retail Sales Insights:** EDA can help uncover patterns in customer purchasing behavior, such as peak buying times, popular products, and customer segmentation, using transaction-level data.
3. **IoT Sensor Data Monitoring:** Spark processes real-time data coming from distributed sensors to detect anomalies, monitor performance, and alert in case of unusual patterns.
4. **Financial Transactions:** Huge volumes of transaction data can be analyzed to detect fraudulent activities, such as unauthorized transfers or abnormal spending behaviors.
5. **Healthcare Analytics:** Patient data, including medical histories and treatment responses, can be explored to improve diagnoses, predict outcomes, and personalize care.

2. How is Data Exploration Done in Apache Spark?
Answer:
Exploratory Data Analysis (EDA) is the first step in any data science or machine learning project. It's like getting to know your data — understanding what's inside, spotting errors, identifying trends, and deciding how to clean or transform it.

With Spark, EDA can be done on huge datasets that don't fit into memory using a distributed approach. Here's a deeper breakdown of how EDA is carried out using Apache Spark:

Steps To perform EDA On Apache Spark

Step 1: Initializing Spark Session
Initializing Spark Session then A SparkSession is started, which serves as the entry point for working with DataFrames and datasets in Spark.

Step 2: Reading and Loading Data
Data is loaded from various formats (CSV, JSON, Parquet, etc.) into Spark DataFrames. These DataFrames are distributed collections of data, similar to tables.

Step 3: Data Inspection and Schema Exploration
We examine the structure of the data using commands to display column names, data types, row counts, and sample records.

Step 4: Data Cleaning
This involves handling missing values, fixing data types, removing duplicates, and filtering out invalid records. This step ensures that the data is accurate and usable.

Step 5: Descriptive Statistics
We calculate summary statistics like mean, median, standard deviation, min, and max values to understand the distribution of data.

Step 6: Grouping and Aggregation
Data is grouped by categories and aggregated to analyze trends and patterns across different segments.

Step 7: Filtering and Sorting
Specific subsets of the data are extracted by applying filter conditions and sorting values for deeper insights.

Step 8: Data Sampling and Conversion
For visualization or detailed analysis, a small sample of the Spark DataFrame is converted into a Pandas DataFrame.

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.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Experiment No 10

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

Theory:

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

Answer:

In Data processing, we generally deal with two major types: 1)Batch processing and 2)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 great when you don't need real-time insights and can wait for results.

- Stream Data Processing (also called real-time processing) deals with data that flows in continuously like sensor data, social media updates, or payment transactions. Instead of waiting for a large chunk, stream processing handles data piece by piece, as it comes in.

Streaming is the process of analyzing data in motion. It's especially useful when timely insights matter like fraud detection, server health monitoring, or stock market analytics.

2. How Does Data Streaming Take Place Using Apache Spark?

Answer:

Apache Spark provides a module called Spark Streaming (and its newer version called Structured Streaming) to process 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 the incoming data into small, manageable batches (micro-batches). Even though it's called streaming, it's actually processing in very small intervals like every second or every few seconds.

- Transformations and Actions: Just like in batch mode, you can apply filtering, grouping, or aggregation logic to the data in each micro-batch.

- Output Sink: The results can be saved or pushed to dashboards, databases, or alerts systems giving near-instant updates based on the incoming data.

Spark special is that the same programming model can be used for both batch and stream processing making it flexible and powerful for developers and analysts alike.

Steps for Batch Data Analysis using Apache Spark

1. **Start Apache Spark Environment**
   Launch Apache Spark through a local installation, cloud platform, or a notebook interface like Jupyter or Databricks. This initializes the engine to process your data.

2. **Read a Batch Dataset**
   Load a static file (like a CSV or JSON) that contains stored historical data. This is typically a complete dataset with a defined start and end.

3. **Explore the Dataset**
   Get an overview of the data by checking column names, data types, and sample records. This helps identify what kind of analysis or cleaning might be required.

4. **Clean and Prepare the Data**
   Handle missing values, rename columns for clarity, fix incorrect data types, and remove duplicates. Clean data is essential for meaningful analysis.

5. **Perform Transformations and Aggregations**
   Apply operations like filtering specific rows, grouping by a column (e.g., region, product), and calculating metrics like average, count, or total sales.

6. **Store or Display Output**
   After analysis, the results can be saved to a new file (like CSV or Parquet), visualized using tools, or displayed in the console.

Steps for Streamed Data Analysis using Apache Spark

1. **Initialize Spark with Structured Streaming**
   Start a Spark session and configure it to accept live data using Structured Streaming, which is Spark's modern approach to handling real-time data.

2. **Connect to a Streaming Data Source**
   Link Spark to a live data source like Apache Kafka, Socket, or a directory where new files keep arriving. This tells Spark where to expect new data continuously.

3. **Define the Schema for Streaming Data**
   Since streaming data doesn't always come with a header, you define what each field means e.g., timestamp, value, sensor_id so Spark knows how to process it.

4. **Apply Streaming Operations**
   As new data flows in, apply operations such as filtering rows (e.g., temperature >

100), grouping over time windows (like every 10 seconds), or aggregating (e.g., average values).

5. **Write Stream Output to a Sink**
   The results are written continuously to a destination like the console, a file system, a database, or even a live dashboard.

6. **Monitor the Streaming Pipeline**
   Monitor the job to ensure the stream is running smoothly. You can check data arrival, processing speed, and memory usage. The stream continues running until manually stopped.

Conclusion

This experiment helps us understand two powerful ways of working with data using Apache Spark: batch and streamed processing. Batch analysis is well-suited for historical or static data, allowing deep dives and summary reports. On the other hand, streamed analysis enables real-time decision-making by processing live data as it arrives. Apache Spark handles both seamlessly using its unified framework and scalable infrastructure. By learning both approaches, we equip ourselves with the flexibility to tackle a wide variety of real-world data problems whether they require immediate insight or deep historical trends.

Introduction

Employee retention is crucial to organizational success, especially in today's competitive job market. Understanding why employees leave and detecting irregularities in salary distribution can help companies make informed decisions. This project leverages machine learning to predict employee attrition and identify salary anomalies using classification and outlier detection models. By analyzing key features such as salary, department, years of experience, and job satisfaction, the system provides insights into employee behavior. The project not only enhances HR analytics but also aids in proactive decision-making. The solution is data-driven, interpretable, and aimed at increasing organizational efficiency through predictive modeling.

Objectives

The project aims to enhance decision-making in Human Resource Management by applying machine learning techniques for employee attrition prediction and salary anomaly detection. It addresses challenges in retaining employees and ensuring fair compensation through data-driven analysis. By identifying key attrition factors and salary outliers, the project enables HR teams to take proactive and informed actions.

- **Perform EDA –** Explore and visualize data to uncover patterns, correlations, and trends.
- **Analyze employee data –** Understand employee demographics, job roles, and satisfaction metrics influencing attrition.
- **Build and compare ML models –** Implement various classification algorithms to predict attrition and evaluate their performance.
- **Detecting salary anomalies –** Use outlier detection techniques to identify irregularities in employee salaries.

Motivation

Employee turnover and salary mismanagement have long-term repercussions on a company's growth and culture. Traditional HR analytics are often reactive and lack the precision of predictive models. The motivation behind this project is to empower organizations with proactive tools that anticipate attrition and flag inconsistencies in salary data. By leveraging machine learning, companies can adopt a scientific approach to workforce management, improving satisfaction and transparency. The increasing availability of employee-related data provides a strong foundation to drive impactful insights, reduce churn, and foster a more stable work environment.

Scope of the Work

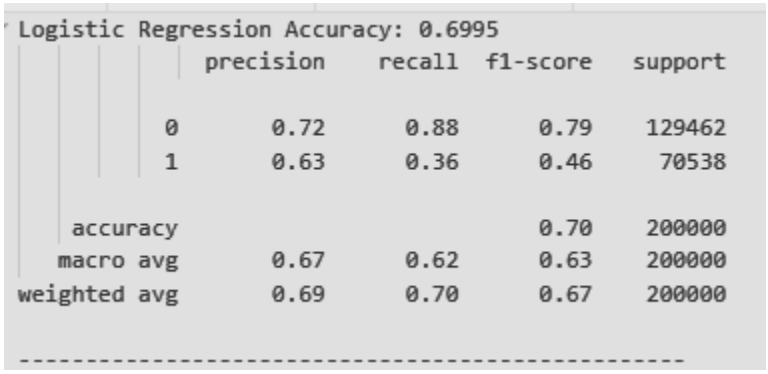
The scope of this project outlines the core components and deliverables required to develop a comprehensive, intelligent system for predicting employee attrition and detecting salary anomalies using machine learning techniques. The focus is on robust model development,

insightful data visualization, and actionable HR analytics.

- **Data Collection and Preprocessing:** Gather employee data from the provided dataset and perform preprocessing steps such as handling missing values, encoding categorical variables, scaling features, and preparing the dataset for modeling.
- **Exploratory Data Analysis (EDA): Perform in-depth EDA to uncover trends and relationships within the data, such as salary distributions, department-wise attrition, and feature correlations. Visualizations will guide further modeling efforts.**
- **Attrition Prediction Modeling:** Implement multiple classification models including Logistic Regression, Decision Tree, Random Forest, SVM, KNN, and XGBoost. Each model will be trained, tested, and optimized using appropriate metrics.
- **Anomaly Detection for Salary Outliers:** Use the Isolation Forest algorithm to detect unusual salary patterns, aiding HR departments in identifying potential inconsistencies or outliers in compensation.
- **Performance Evaluation and Visualization:** Evaluate model performance using metrics such as accuracy, precision, recall, and F1-score. Visualize the confusion matrix for each model to compare effectiveness in classifying attrition.
- **Result Analysis and Interpretation:** Analyze outcomes of both classification and anomaly detection tasks. Provide detailed inferences supported by quantitative results and graphical insights.

Accuracies of Models

1. **Logistic Regression**

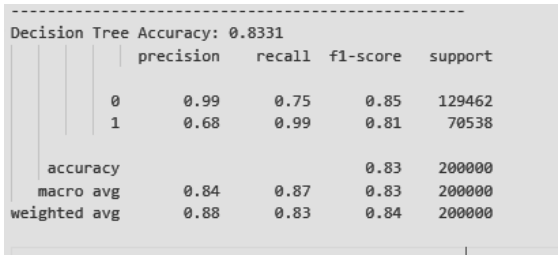


Logistic Regression Accuracy: 0.6995

	precision	recall	f1-score	support
0	0.72	0.88	0.79	129462
1	0.63	0.36	0.46	70538
accuracy			0.70	200000
macro avg	0.67	0.62	0.63	200000
weighted avg	0.69	0.70	0.67	200000

The logistic regression model achieved an accuracy of 0.6995, indicating a moderate ability to predict employee attrition (0 for staying, 1 for leaving) across a dataset of 200,000 employees. It performs better for the majority class (0), with a precision of 0.72, recall of 0.88, and an F1-score of 0.79, reflecting strong identification of non-leaving employees (129,462 instances). However, for the minority class (1), the model struggles, with a precision of 0.63, recall of 0.36, and an F1-score of 0.46, suggesting it misses many leaving employees (70,538 instances). The macro average (0.67 precision, 0.62 recall, 0.63 F1) and weighted average (0.69 precision, 0.70 recall, 0.67 F1) further indicate a bias toward the majority class, making it less effective for balanced prediction.

2. Decision Tree

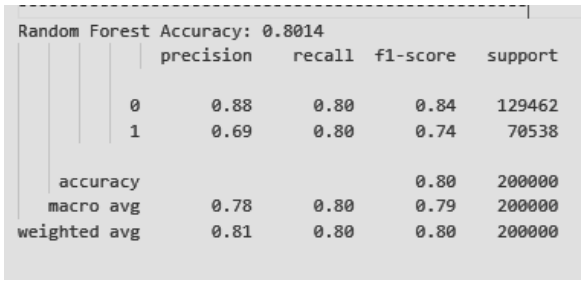


Decision Tree Accuracy: 0.8331

	precision	recall	f1-score	support
0	0.99	0.75	0.85	129462
1	0.68	0.99	0.81	70538
accuracy			0.83	200000
macro avg	0.84	0.87	0.83	200000
weighted avg	0.88	0.83	0.84	200000

The decision tree model shows a higher accuracy of 0.8331, demonstrating a strong overall performance in predicting attrition. It excels at identifying leaving employees (class 1), with a recall of 0.99 and an F1-score of 0.81, despite a precision of 0.68 (70,538 instances), indicating it correctly flags most departures but with some false positives. For non-leaving employees (class 0, 129,462 instances), it has a precision of 0.99 but a lower recall of 0.75, suggesting it misses some who stay. The macro average (0.84 precision, 0.87 recall, 0.83 F1) and weighted average (0.88 precision, 0.83 recall, 0.84 F1) highlight a well-balanced model, making it a robust choice for attrition prediction.

3. Random Forest Analysis



Random Forest Accuracy: 0.8014

	precision	recall	f1-score	support
0	0.88	0.80	0.84	129462
1	0.69	0.80	0.74	70538
accuracy			0.80	200000
macro avg	0.78	0.80	0.79	200000
weighted avg	0.81	0.80	0.80	200000

The random forest model achieves an accuracy of 0.8014, offering a solid predictive performance. It balances the classes well, with a precision of 0.88 and recall of 0.80 for non-leaving employees (129,462 instances), yielding an F1-score of 0.84. For leaving employees (70,538 instances), it maintains a precision of 0.69 and a recall of 0.80, resulting in an F1-score of 0.74. The macro average (0.78 precision, 0.80 recall, 0.79 F1) and weighted average (0.81 precision, 0.80 recall, 0.80 F1) indicate a consistent performance across both classes, suggesting random forest is a reliable model for attrition analysis.

4. **KNN**

```
------------------------------------------------
KNN Accuracy: 0.7973
              precision    recall  f1-score   support

           0       0.87      0.80      0.84    129462
           1       0.69      0.79      0.73     70538

    accuracy                           0.80    200000
   macro avg       0.78      0.79      0.78    200000
weighted avg       0.81      0.80      0.80    200000
```

The KNN model records an accuracy of 0.7973, showing competitive performance in predicting attrition. It performs similarly to random forest, with a precision of 0.87 and recall of 0.80 for non-leaving employees (129,462 instances), leading to an F1-score of 0.84. For leaving employees (70,538 instances), it achieves a precision of 0.69 and recall of 0.79, resulting in an F1-score of 0.73. The macro average (0.78 precision, 0.79 recall, 0.78 F1) and weighted average (0.81 precision, 0.80 recall, 0.80 F1) reflect a balanced approach, making KNN a viable option for attrition prediction, though slightly less effective than the decision tree

5. **XG Boosting Algorithm**

```
XGBoost Accuracy: 0.8334
              precision    recall  f1-score   support

           0       1.00      0.74      0.85    129462
           1       0.68      1.00      0.81     70538

    accuracy                           0.83    200000
   macro avg       0.84      0.87      0.83    200000
weighted avg       0.89      0.83      0.84    200000

------------------------------------------------
```

The XGBoost model achieves an accuracy of 0.8334, demonstrating excellent predictive performance for employee attrition across the 200,000-instance dataset. It excels at identifying leaving employees (class 1, 70,538 instances), with a perfect recall of 1.00 and an F1-score of 0.81, despite a precision of 0.68, indicating it captures all departures but includes some false positives. For non-leaving employees (class 0, 129,462 instances), it boasts a precision of 1.00 but a recall of 0.74, suggesting it accurately identifies those staying but misses some instances. The macro average (0.84 precision, 0.87 recall, 0.83 F1) and weighted average (0.89 precision,

0.83 recall, 0.84 F1) reflect a well-balanced and robust model, making XGBoost a highly effective tool for attrition prediction.

Conclusion

This project effectively combines predictive analytics and anomaly detection to assist HR teams in minimizing attrition and managing salaries transparently. Using employee data, the system leverages models like XGBoost for high-accuracy attrition prediction and Isolation Forest for identifying irregular salary patterns. The results offer actionable insights to improve employee retention and ensure fair compensation practices. With proper visualization and performance evaluation, the system bridges the gap between raw data and strategic HR interventions. Overall, it demonstrates how data science can bring measurable improvements to workforce management.

Future Scope

- **Enhances workplace satisfaction by identifying attrition reasons:** Pinpoints causes like low pay or poor work-life balance via data and NLP, boosting morale and retention as of April 14, 2025.
- **Promotes fair salary distribution and transparency:** Ensures equitable pay with transparent reporting, building trust.
- **Reduces recruitment costs and time with proactive insights:** Identifies at-risk employees early, minimizing hiring needs and preserving knowledge.
- **Helps HR build inclusive teams:** Uses fairness metrics and engagement data to create diverse, balanced teams.
- **Enables data-driven policies for employee welfare:** Leverages deep learning and dashboards for policies like better benefits, enhancing well-being and loyalty.

Societal Impact

- **Enhances workplace satisfaction by identifying attrition reasons:** Pinpoints causes like low pay or poor work-life balance via data and NLP, boosting morale and retention.
- **Promotes fair salary distribution and transparency:** Ensures equitable pay with transparent reporting, building trust.
- **Reduces recruitment costs and time with proactive insights:** Identifies at-risk employees early, minimizing hiring needs and preserving knowledge.
- **Helps HR build inclusive teams:** Uses fairness metrics and engagement data to create a diverse, balanced team.

05/08

Page No.
Date

Name:- Bhushan Mukund Kor
Class:- D15C Roll NO:- 28
Subject:- AIDS

Assignment No-1

Q.1) What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive and renovated our way of life with different applications?

→ AI (Artificial Intelligence) refers to computer systems that can perform tasks requiring human intelligence, such as learning, reasoning, problem-solving, and decision-making.

AI's Role During COVID-19:-

1) Healthcare & Diagnosis- AI helped detect COVID-19 from X-rays/CT scans and predicted outbreaks.

2) Vaccine Development:- Accelerated drug discovery and vaccine trials.

3) Contact Tracing - AI-powered apps tracked virus spread and alerted users.

4) Remote Work & Education:- AI-driven tools enabled virtual meetings, online learning, and automation.

5) Chatbots & Customer Support:- Automated helplines reduced strain on healthcare and business.

6) Robotics & Automation - AI-powered robots assiste in sanitization, delivery and hospital care.

7) Fake News Detection - AI identified misinformatio and ensured reliable information sharing.

AI transformed work, healthcare, and communicatio making life more efficient during the pandemic.

Q.2) What are AI Agents terminology, explain with examples.

→ AI Agents Terminology:

An AI Agent is a system that perceives its environm and takes actions to achieve a goal.

Key Terminologies:-

1) Agent :- A software or hardware entity that interacts with the environment.
eg. A self-driving car

2) Environment :- The external system with which the agent interacts.
eg. Traffic, pedestrains for a self driving car.

3) perception :- The agent's ability to collect data from sensors.
eg. Cameras & LiDAR in autonomous vehicles.

4) Actuators:- The components that execute actions.
eg. Motors & steering in a robotic vacuum cleaner.

5) Sensors:- Devices that gather information from the environment.
eg. Temperature sensors in smart thermostats.

6) Rationality:- The ability of an agent to take the best possible action.
eg. A chess AI choosing the best move.

7) Autonomy:- The degree to which an agent operates without human intervention.
eg. A robotic assistant that learns user preferences over time.

Q. 3) How AI technique is used to solve 8 puzzle problem?

→ The 8-puzzle problem is solved using AI search techniques like Brute force search and Heuristic search.

AI Techniques for Solving the 8-puzzle Problem:-

1) Uninformed search:- (Brute Force):

- Breadth-first search (BFS) - Explores all possible moves level by level. (Guaranteed solution but slow)
- Depth-first search (DFS) - Explores deep paths first. (Might get stuck in infinite loops).

2) Informed Search (Heuristic search):-

Date

- Best-First search (Greedy Algorithm) - Chooses the best move based on heuristics. (Faster but may not be optimal).

- A search Algorithm :- Uses a heuristic function.
  - Heuristic ($h(n)$) = Number of misplaced tiles.
  - Cost ($g(n)$) = steps taken so far
  - $f(n) = g(n) + h(n)$ (Ensures shortest path)

eg.

A* is Best because it guarantees optimal solution & balances exploration and efficiency.

AI helps solve the 8-puzzle by efficiently searching for the shortest path to reach the goal state.

Q.4) What is PEAS descriptor? Give PEAS descriptor for following:-

1) Taxi Driver
2) Medical diagnosis system
3) A music composer
4) An aircraft autolander
5) An essay evaluator
6) A robotic sentry gun for the Keck Lab.

→ PEAS (Performance measure, Environment, Actuators, sensors) is a framework to define an AI agent's components.

PEAS Descriptors for Given Systems :-

	Agents	Performance	Environment	Actuators	Sensors
1)	Taxi Driver	Safety, speed, fuel efficiency, customer satisfaction.	Roads, traffic, weather, passengers.	Steering, accelerator, brakes, indicators.	GPS, cameras, speed sensor, fuel gauge.
2)	Medical Diagnosis system	Accuracy, diagnosis speed, patient satisfaction.	Patient symptoms, medical reports, lab reports.	Display diagnosis, suggest treatment, notify doctors.	Patient input, medical history, test results.
3)	Music Composer AI	Harmony, creativity, genre accuracy, listener ratings.	Music theory, existing songs, user preferences.	Generates sheet music, audio files, MIDI output.	User feedback, musical databases, instruments.
4)	Aircraft Autolander	Safe landing, smoothness, accuracy, efficiency.	Runway, weather, altitude, wind speed.	Landing gear, brakes, flaps, engine throttle.	Radar, GPS, altimeter, speed sensors.
5)	Essay Evaluator AI	Accuracy, grammar check, coherence, objectivity.	Essays, language rules, grading rubrics.	Score assignment, feedback generation, Plagarism check.	Input text, grammar databases, vocabulary rules.
6)	Robotic Sentry Gun for Kerk Lab	Accuracy, threat detection, response time, safety.	Surrounding area, intruders, authorized personnel.	Rotating turret, firing mechanism, alarm system.	Motion detecting cameras, thermal sensors.

Page No.
Date

Q.5) Categorize a shopping bot for an offline bookstore according to each of the six dimensions (fully/partially observable, deterministic/stochastic, episodic/sequential, static/dynamic, discrete/continuous, single/multi agent)

→ Categorize of a Shopping Bot for an Offline Bookstore

1) Observability : Partially Observable
- The bot may not have full information about book availability, customer preferences, or store layout.

2) Deterministic vs. stochastic : Stochastic
- Customer behavior and stock availability are uncertain, making outcomes unpredictable.

3) Episodic vs. Sequential : Sequential
- The bot's actions (eg. suggesting books, handling purchase) affect future interactions.

4) Static vs. Dynamic : Dynamic
- The bookstore environment changes with customer movements, stock updates, and price adjustments.

5) Discrete Vs Continuous : Discrete
- The bot processes distinct inputs (book selection, payment) but may have continuous interactions like voice recognition.

6) Single vs Multi-Agent :- Multi-Agent

• The bot interacts with multiple customers and bookstore staff, making it a multi-agent system.

This classification helps in designing an efficient AI based shopping assistant.

Q.6) Differentiate Model based & Utility based Agent.

→

Model based Agent	Utility based Agent
1) Uses an internal model of the environment to decide actions.	1) Uses a Utility function to choose the best action based on maximum benefit.
2) Maintains knowledge of how the world works and updates it over time.	2) Evaluates different possible actions and picks the one that maximizes utility.
3) Decision Based on past and current perceptions.	3) Decision Based on a computed utility value.
4) Works towards achieving a goal state.	4) Works towards maximizing long-term performance
5) Eg. A self-driving car using a map to navigate.	5) Eg: A stock trading AI choosing the most profitable investment.

7) Explain the architecture of a knowledge based ag and learning Agent:

→ Knowledge - Based Agent:

Architecture

A Knowledge-Based Agent (KBA) uses knowledge stored in a Knowledge Base (KB) to make decisions

Components:

1) Knowledge Base (KB) - stores facts and rules.
2) Inference Engine - Applies reasoning to derive conclusions.
3) Perception (sensors) - Collects environmental data.
4) Actuators - Executes actions.
5) Learning Module - Updates KB overtime.

Example: An expert medical diagnosis system that suggests treatments based on stored medical knowledge.

Learning Agent Architecture:

A Learning Agent improves its performance over time by learning from past experiences.

components:

1) Learning Element - Learns patterns from past actions.
2) Performance Element - Makes decisions based on learned data.
3) Critic - Evaluates agent performance and give feedback
4) Problem Generator - Suggests new experiences for learning.

Example :- A self-driving car that refines its driving skills through real-world experience.

Q. 8) Convert the following to predicates:
a) Anita travels by car if available otherwise travels by bus.
b) Bus goes via Andheri and Goregaon.
c) car has puncture so is not available.
Will Anita travel via Goregaon? Use forward reasoning.

→ Step 1:- Convert sentences into predicates.

1) Anita's travel options:-
- If a car is available, Anita travels by car.
- If a car is not available, Anita travels by bus.

Predicate form:-
- ~~Travels (Anita, car)~~
- Available (car) → Travels (Anita, car)
- ¬Available (car) → Travels (Anita, Bus)

2) Bus Route Information:
- The bus goes via Andheri and Goregaon.

Predicate form:-
- GoesVia (Bus, Andheri)
- GoesVia (Bus, Goregaon)

3) Car Availability:
- The car has a puncture, so it is not available.

Predicate form:

- Puncture (Car)
- ¬ Available (Car)

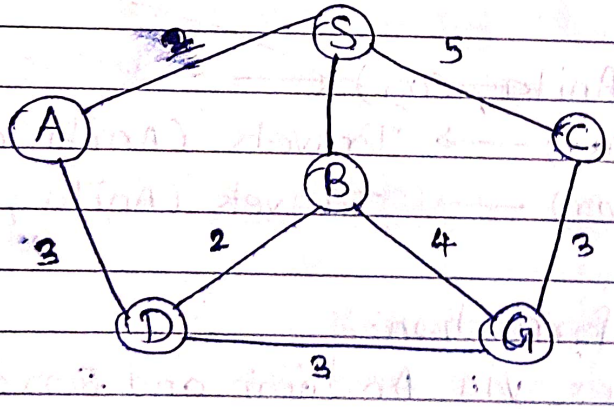
Step 2:- Apply Forward Reasoning.

1) Given ¬ Available (Car), the first rule implies:
   - Travels (Anita, Bus)

2) Since Travels (Anita, Bus) and we know Goes Via (Bus, Goregoan) and Goes Via (Bus, Andheri).

∴ Anita will travel via both Andheri and Goregaon.
And Answer is yes Anita will travel via Goregaon.

Q. 9) Find the route from S to G using BFS.



→ Step 1:- Apply BFS

BFS explores nodes level by level (FIFO order).

Step 1:- Start from 'S' node

Queue = [S]

Explore = {S}

Expand S → Add [A, B, C] to queue

Updated Queue = [A, B, C]

Step 2:- Explored = {S, A}

Expand A → Add [D] to queue

updated Queue : [B, C, D]

Step 3:- Explored = {S, A, B}

Expand B → Add [G] to queue.

(Goal found)

updated Queue = [C, D, G]

Since we have got our goal no need for further exploration.

Path we can get by backtrack.

G came from B.

B came from S.

Path :- S → B → G

Page No.
Date

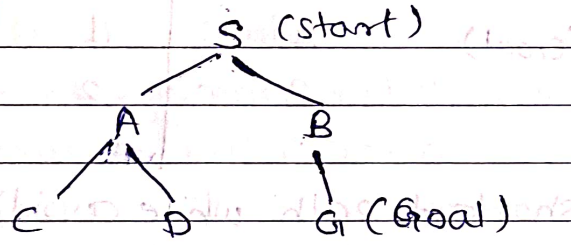
Q.10) What do you mean by depth limited search? Explain Iterative Deepening search with example.

→ **Depth-Limited Search (DLS)**

Definition:

- Depth-Limited Search (DLS) is a modified Depth-First search (DFS) with a predefined depth limits.
- It avoids infinite loops in graphs with cycles and saves memory.
- If a goal is not found within the limit, it fails.

eg.



If depth = 1
Depth 0 : S
Depth 1 : A, B
we cannot find goal
but If depth = 2
Depth 2 : C, D, G
with depth limit = 2, we can find G.

Time complexity = $O(b^l)$ where b = branching factor & l = depth limit.

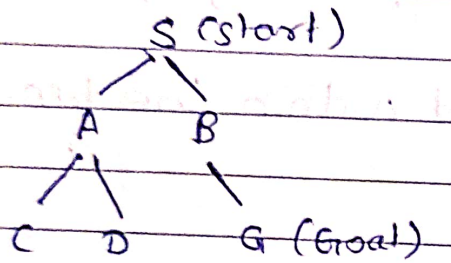
Space complexity = $O(l)$ (Better than BFS)

Iterative Deepening Search (IDS)

Definition:

- IDS is a combination of Depth-First search (DFS) and Breadth-First search (BFS).
- It runs DFS repeatedly, increasing the depth limit one step at a time.
- It finds the shortest path like BFS but uses less memory.

eg.



Iteration	Depth Limit	Nodes Explored
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 the goal

Space complexity = $O(b^d)$

11) Explain Hill climbing and its drawbacks in detail with example. Also state limitations of steepest-ascent hill climbing.

→ **Hill Climbing Algorithm:-**

Hill Climbing is a heuristic search algorithm that continuously moves toward higher valued states (better solutions) until it reaches a peak. It is commonly used in optimization problems.

Types of Hill Climbing:

1) Simple Hill Climbing
2) Steepest-Ascent Hill Climbing
3) Stochastic Hill Climbing

Drawbacks of Hill Climbing:-

1) Local Maxima
- The algorithm may get stuck at a peak that is not the global maximum.
- Eg. In a landscape with multiple peaks, the algorithm might stop at a lower peak.

2) Plateaus
- A flat area where all neighboring states have the same value.
- The algorithm cannot decide where to move next.

eg. In an 8-Queens problem, the heuristic value might not change even after moving a queen.

3) Ridges
- A path of increasing values that cannot be climbed because moves are limited.
- Eg. If an algorithm can move only in 4 directions but the optimal path requires diagonal moves, it may get stuck.

4) Lack of Backtracking
- Once it moves forward, it does not revisit past states.
- Solution: Random restarts or Simulated Annealing.

Limitations of steepest-Ascent Hill Climbing :-

- Computationally expensive: It checks all possible moves before choosing the best one.
- Still suffers from local maxima and plateaus.
- May oscillate between states if two equally good options exist.

Q. 12) Explain simulated annealing and write its algorithm.

→ Simulated Annealing (SA) :-

Simulated Annealing is a metaheuristic optimization algorithm that helps in escaping local maxima by sometimes accepting worse solutions to explore a better global solution.

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

Algorithm:-

1) Initialize the solution and set the initial temperature
2) Repeat until temperature is low:
   - Generate a new neighbor state.
   - Calculate ΔE (change in cost function).
   - If $\Delta E > 0$, accept the new state.
   - Else, accept with probability $P = e^{(\frac{\Delta E}{T})}$
   - Reduce temperature using a cooling function.
3) Return the best solution found.

Q.13) Explain A* Algorithm with an example.

→ A* is an informed search algorithm that finds the shortest path from a start node to a goal node using a combination of cost(g) and heuristics (h).

Formula:- $f(n) = g(n) + h(n)$

g(n):- Cost from the start node to node n.
h(n):- Estimated cost from node n to the goal (heuristic)
f(n):- Total estimated cost (priority for selection)

Algorithm:-

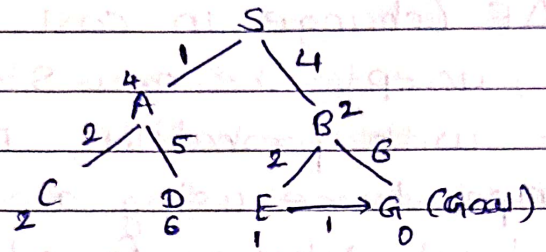
1) Initialize: Add the start node to the open list.
2) Loop until Goal is found:
   - Pick the node with the lowest f(n) from the open list.
   - Move it to the closed list (visited).
   - Generate neighboring nodes.

Date

- Calculate their $f(n) = g(n) + h(n)$.
- If a better path is found, update $g(n)$.

3) Repeat until the goal is reached or open list is empty.

Example :-



Step-by-step Execution

1) Start at S, $f(S) = g(S) + h(S) = 0 +$ heuristic
2) Expand S, select A ($f = 1+4 = 5$) and B ($f = 4+2 = 6$)
3) Expand A, check C ($f = 3+2 = 5$) and D ($f = 6+6 = 12$).
4) Expand B, check E ($f = 6+1 = 7$) and G ($f = 10+0 = 10$).
5) Expand E, finds G with $f = 7+0 = 7$

Shortest path found: $S \rightarrow B \rightarrow E \rightarrow G$
cost = 7.

14) Explain Min max. Explain Minimax Algorithm and draw game tree for Tic TAC Toe game.

→ Minimax is a decision-making algorithm used in two-player games like Tic-Tac-Toe, Chess, and checkers. It assumes that:

- One player (maximizer) tries to maximize the score.
- The other player (minimizer) tries to minimize the score.

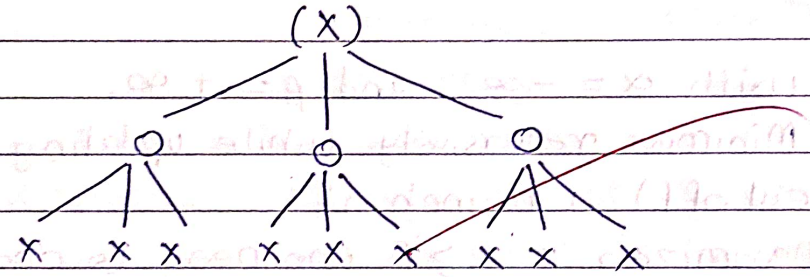
Page No.
Date

Algorithm steps:-

1) Generate the game tree up to a certain depth.
2) Evaluate terminal nodes using a heuristic function.
3) Backpropagate scores:
   - Maximizer selects the maximum score.
   - Minimizer selects the minimum score.
4) Repeat until the best move is found.

Minimax in Tic-Tac-Toe:-

Game Tree (X starts first)



- Maximizer (X) tries to win (score = +1).
- Minimizer (O) tries to block or win (score = -1).
- Draw = 0.

Q. 15) Explain Alpha beta pruning algorithms for adversarial search with example.

→ ## Alpha-Beta Pruning Algorithm:-

Alpha-Beta Pruning is an optimization technique for the Minimax algorithm in adversarial search. It skips unnecessary branches in the game tree, improving

efficiency without affecting the result.

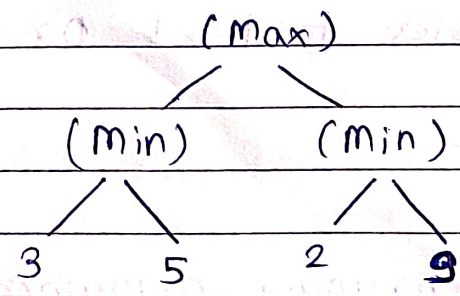
Key concepts:-

- Alpha (α) : Best maximum score for the maximizer (initialized as $-\infty$).
- Beta (β) : Best minimum score for the minimizer (initialized as $+\infty$).
- Pruning : Stops evaluating a branch if it's worse than the current best option.

Algorithm

1) Start with $\alpha = -\infty$ and $\beta = +\infty$.
2) Apply Minimax recursively while updating α and β.
3) Prune (cut off) a branch if:
   - For Maximizer : $\alpha \geq \beta$ (no need to check further).
   - For Minimizer : $\beta \leq \alpha$ (no need to check further).
4) Continue until the best move is found.

E.g.



Maximizer (X) wants max value.
Minimizer (O) wants min value.

1) Left branch : Min(3,5) = 3, so $\beta = 3$
2) Right branch : First value 2 → $\beta = 2$ (less than α)
3) Prune node (9) as it's unnecessary.

Final choice: Max (3,2) = 3
Skipped evaluation of (9), making it faster.

Page No.
Date

Q. 16) Explain WUMPUS world environment giving its PEAS description. Explain how percept sequence is generated?

→ Wumpus World is a grid-based environment used to demonstrate AI agents working in an unknown, partially observable world. The agent navigates a cave with hazards while searching for gold.

PEAS for WUMPUS:-

Performance Measure :- +1000 for finding gold, -1000 for falling into a pit or encountering wumpus, -100 per move to encourage efficiency.

Environment :- Grid based cave with pits, wumpus, gold, walls and agent's starting position.

Actuators :- Move forward, turn left/right, grab gold, shoot arrow (to kill wumpus), climb out.

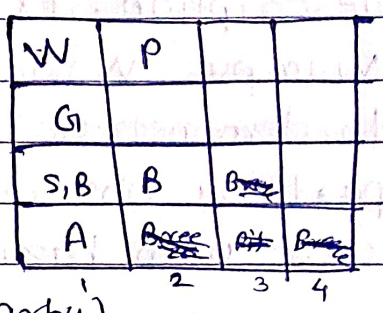
Sensors :- Stench (near wumpus), Breeze (near pits), Glitter (gold present), Bump (wall hit), Scream (wumpus killed).

Percept Sequence:-

- Percepts are generated based on the agent's current location.
- Example: If the agent is next to a Wumpus, it perceives a stench.

• The percept sequence is a list of observations over time.

eg



1 (1, 1) No percept
2 (1, 2) Breeze (pit nearby)
3 (2, 2) Stench, Breeze (Wumpus & pit nearby)
4 (2, 3) Glitter (Gold is here)

Q. 17) Solve the following Crypto-arithmetic problem 1.
SEND + MORE = MONEY.

→ Let's solve the crypto-arithmetic problem.

```
  SEND
+ MORE
-------
 MONEY
```

Step 1: Assign Unique Digits to Letters
Each letter represents a unique digit (0-9). The sum must satisfy the equation.

Step 2:- Identify Constraints

M = 1 (since Money has 5 digits) & M is 1st)
O = 0 (as it's the second digit.

Find S

S + M ≥ 10 (to cause a carry), so

S = 9

Page No.
Date

Find E and N

No carry from column 3 → E = 5, N = E + 1 = 6.

Find R

Carry exists from column 1 → R = 8.

Find D and Y

D + E = 10 + Y ⟶ D = 7, Y = 2

Final answer:-

S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2

Verify:-

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

Ans :⟶ 10652

Q. 18) Consider the following axioms: All people who are graduating are happy. All happy people are smiling. someone is graduating.
Explain the following:-
1) Represent these axioms in first order predicate logic.
2) Convert each formula to clause form.
3) Prove that "Is someone smiling?" using resolution technique. Draw the resolution tree.

→ Step 1:- Represent Axioms in First-Order Predicate Logic.

Let:-

- G(x) → x is graduating
- H(x) → x is happy.
- S(x) → x is smiling.

Given axioms:

1) All people who are graduating are happy.
   $\forall x (G(x) \rightarrow H(x))$

2) All happy people are smiling.
   $\forall x (H(x) \rightarrow S(x))$

3) Someone is graduating
   $\exists x\, G(x)$

Step 2:- Convert to Clause form

Standardize variable (Remove Universal Quantifiers)

1) $G(x) \rightarrow H(x)$ becomes $\neg G(x) \vee H(x)$
2) $H(x) \rightarrow S(x)$ becomes $\neg H(x) \vee S(x)$
3) $\exists x\, G(x)$ becomes $G(A)$ (Introduce a skolem constant A)

Clause form:

- $\neg G(x) \vee H(x)$
- $\neg H(x) \vee S(x)$
- $G(A)$

Date

Step 3:- Prove "Is someone smiling?"

We need to prove $\exists x\, S(x)$ or show that $S(A)$ is true using resolution.

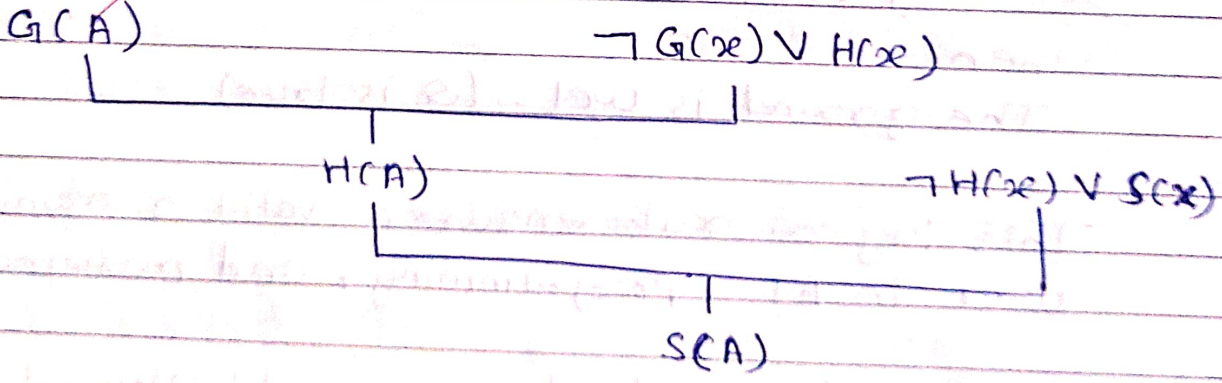
Resolution steps

1) From $G(A)$ and $\neg G(x) \vee H(x)$
   - $G(A)$ unifies with $G(x) \rightarrow H(A)$ is derived.

2) From $H(A)$ and $\neg H(x) \vee S(x)$
   - $H(A)$ unifies with $H(x) \rightarrow S(A)$ is derived.

Since we derived $S(A)$, we conclude that someone is smiling.

Step 4: Resolution Tree



Thus, Someone is smiling is proved using resolution.

Page No.
Date

Q.19) Explain Modus ponen with suitable example.

→ Modus Ponens (Law of Detachment)

Modus ponens is a fundamental rule of inference in logic that states:

If:

1) $P \rightarrow Q$ (If P is true, then Q is true)

2) P (P is true)

Then:

Q must be true.

Example

Premises:-

1) If it rains, the ground will be wet.

$P \rightarrow Q$ (If Rain, then wet ground)

2) It is raining.

P (Rain is true)

then

The ground is wet. (Q is true)

This logical rule ensures valid reasoning and widely used in AI, programming, and mathematical proofs.

Q.20) Explain forward chaining and backward chaining algorithm with the help of example.

→ 1) Forward Chaining (Data Driven)

• Starts with known facts and applies rules to infer

new facts until the goal is reached.
- Used in expert systems, AI planning, and production systems.

eg.

Rules:

1) If it rains, the ground will be wet.
   Rain ⟶ Wet ground
2) If the ground is wet, people carry umbrellas.
   wet Ground ⟶ Umbrella.

Given Fact:

It is raining.

Forward Chaining Steps:

1) It is raining triggers Rule 1 ⟶ The ground is wet.
2) The ground is wet triggers Rule 2 ⟶ People carry umbrellas.

Conclusion: People carry umbrellas.

2) Backward Chaining (Goal-Driven)

- Starts with a goal and works backwards to check if known facts support it.
- Used in AI problem-solving and diagnostic systems.

Date

Example:-

Goal: Are people carrying Umbrellas?

Rules:
1) If the ground is wet, people carry umbrellas
   Wet Ground → Umbrella.
2) If it rains, the ground is wet:
   Rain → Wet Ground.

Backward Chaining steps:-

1) Are people carrying umbrellas? → Check if the ground is wet.
2) Is the ground wet? → Check if it is raining.
3) Is it raining ? → Yes (Given Fact).

Conclusion: Yes, people carry umbrellas.

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Assignment No 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)

Answer:
Dataset: 82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

1. Mean

$$\text{Mean} = \frac{\sum x_i}{n}$$

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

$$= \frac{1611}{20}$$

Mean = 80.55

2. Median
Sort values: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Total values 20 .i.e Even

$$\text{Median} = \frac{x_{(n/2)} + x_{(n/2)+1}}{2}$$

Even number of values → average of 10th and 11th:

$$\text{Median} = \frac{81+82}{2} = 81.5$$

3. Mode
The mode is the value that appears most frequently in a dataset.
Most frequent value = 76 (appears 3 times)
Mode = 76

4. Interquartile Range (IQR)

$$\text{IQR} = Q3 - Q1$$

Q1 = 25th percentile = average of 5th and 6th

Q1 = $\frac{76+76}{2} = 76$

Q3 = 75th percentile = average of 15th and 16th

Q3 = $\frac{88+90}{2} = 89$

IQR = Q3−Q1

= 89−76

IQR =13

Result:

Mean = 80.55 , Median = 81.5 , Mode = 76 , IQR = 13

Q.2 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**

Answer:

We will see the following Machine Learning Tools Comparison:

1. Machine Learning for Kids
2. Teachable Machine

1) Machine Learning for Kids

Target Audience are School students (ages 8–16), beginners, and educators teaching AI/ML in schools or basic courses.

Users can train ML models using Text, Images, Numbers

It connects with platforms like Scratch and Python, allowing users to build interactive projects like:

- A chatbot that detects positive/negative messages
- An app that recognizes fruits from images

Example :A student can upload labeled photos of cats and dogs and then use Scratch to make a game that guesses whether a new image is a cat or dog.

Benefits:

- User-friendly interface, great for young learners.
- Integrates ML with visual programming (Scratch).
- Encourages creative projects and experimentation.
- No coding required (but optional Python use is available).
- Cloud-based, accessible from browsers.

Drawbacks:

- Limited to basic models; lacks depth for real-world ML applications.
- Accuracy is low compared to professional tools.
- Minimal control over algorithm types, hyperparameters, or data preprocessing.
- Not suitable for large datasets or complex models.

2) Teachable Machine

Target Audiences are General public, students, educators, hobbyists, and even artists.
It is use for:

- Creating models by training with webcam/audio/images.
- Exporting the model to use in websites or apps.
- Because No coding required.

Benefits:

- Extremely simple to use with a few clicks.
- Fast real-time training with immediate feedback.
- Supports exporting trained models to real applications.
- Great for interactive demos, art installations, and education.

Drawbacks:

- No deep customization or control over the model architecture.
- No preprocessing options (e.g., normalization).
- Limited dataset size and simple structure = low accuracy on complex problems.
- Cannot handle text or numerical data.

2. Choosing Predictive or Descriptive Analytic.

Tool	Type	Reason
Machine Learning for Kids	Predictive Analytic	It uses trained models to predict categories or outputs from inputs.
Teachable Machine	Predictive Analytic	It predicts labels for new input data (like image or sound classification).

We have not chosen descriptive because Descriptive analytics explains what happened in the past using statistics and visualization. These tools instead predict outcomes using new input data.

3. Choosing Type of Learning.

Tool	Learning Type	Reason
Machine Learning for Kids	Supervised Learning	It uses labeled data (e.g., text labeled as positive/negative) to train.
Teachable Machine	Supervised Learning	Users provide labeled examples for training (e.g., face = “happy”).

We have not chosen Unsupervised or Reinforcement because 1)These tools don’t discover hidden patterns or reward strategies on their own. 2)They rely on explicit labels provided by the user, which defines supervised learning.

Q.3 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.

Answer:

Case Study: Misleading COVID-19 Vaccine Death Visualizations

In early 2024, a wave of misleading posts on social media suggested that COVID-19 vaccines were causing more harm than good. These claims were based on visual data that appeared to show more deaths among vaccinated people than unvaccinated ones in England between July 2021 and May 2023. On the surface, the graphs seemed alarming, but they were missing key context.

(Source: Reuters, March 21, 2024 – "Misleading data used to claim COVID vaccines do more harm than good")

Where the Visualization Went Wrong:

1. No Consideration of Proportions:
   The charts only showed total numbers of deaths, not the size of each group. Since most people in England were vaccinated during that time, it was expected that more deaths would happen in that group. That doesn't mean vaccines were harmful—just that there were more people in that group.

2. Missing Background Info:
   The graphs lacked context about how effective vaccines actually are and didn't explain the difference in the number of people vaccinated versus unvaccinated. Without this, many people misunderstood what the charts were really showing.

3. Leaving Out Mortality Rates:
   By focusing only on total deaths instead of showing the number of deaths per 100,000 people (which gives a fair comparison), the visualizations hid the fact that unvaccinated people actually had a higher risk of dying.

Clarifying the Misrepresentation:
Once experts adjusted the data to account for population size, it became clear that vaccinated individuals had a lower death rate. Data from the UK's Office for National Statistics (ONS) showed that vaccines were effective in reducing COVID-related mortality. The issue wasn't the data itself, but how it was presented. Without proportionality and proper explanation, the graphs ended up spreading misinformation.

Conclusion:
This example shows how easy it is for data visuals to be misread when they're not properly designed. Whether intentional or not, misleading charts can damage public trust, especially when dealing with health information. As Arthur Kakande and Katherine Ellen Foley explain in their articles, it's important for both creators and viewers of data to think critically and ask whether the full picture is being shown. Accurate, clear, and well-contextualized visuals are essential for helping people make informed decisions.

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

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

Answer:
Dataset Description:

The dataset used is based on the **Pima Indian Diabetes dataset**, originally collected by the National Institute of Diabetes and Digestive and Kidney Diseases. It contains diagnostic medical data for female patients of Pima Indian heritage aged 21 or older.

- **Features Include:**
  Number of pregnancies, glucose level, blood pressure, insulin, BMI, diabetes pedigree function, age, etc.
- **Target Variable (Class Label):**
  Outcome – Binary classification:
  - 0 = No diabetes
  - 1 = Diabetes

Pregnancies: Number of times the patient has been pregnant.
Glucose: Plasma glucose concentration after a 2-hour oral glucose tolerance test.
BloodPressure: Diastolic blood pressure (mm Hg).
SkinThickness: Triceps skin fold thickness (mm).
Insulin: 2-hour serum insulin (mu U/ml).
BMI: Body Mass Index (weight in kg / height in m²).
DiabetesPedigreeFunction: A function that scores the likelihood of diabetes based on family history.
Age: Patient age in years.

Model: SVM

Result:

```
After SMOTE: [500 500]
```

Class imbalance was resolved using SMOTE, resulting in a balanced dataset with 500 samples in each class. This helps improve model fairness and performance.

```
Train: 700, Val: 201, Test: 99
```

The dataset was randomly split into training (70%), validation (20%), and test (10%) sets with 700, 201, and 99 samples respectively, ensuring reliable model evaluation.

```
Best Parameters: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

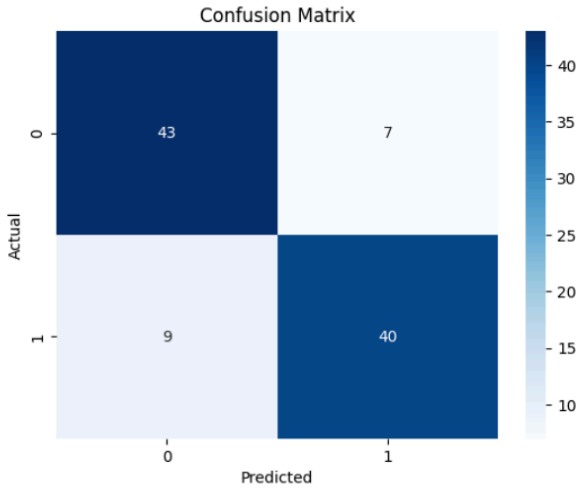
Hyperparameter tuning using GridSearchCV selected the best SVM parameters: C=10, gamma='auto', and kernel='rbf', optimizing model performance on validation data.

```
Validation Accuracy: 0.8059701492537313
              precision    recall  f1-score   support

           0       0.83      0.77      0.80       100
           1       0.79      0.84      0.81       101

    accuracy                           0.81       201
   macro avg       0.81      0.81      0.81       201
weighted avg       0.81      0.81      0.81       201
```

The model achieved 81% validation accuracy, with balanced precision, recall, and F1-scores (~0.81) for both classes, indicating good generalization and class handling.

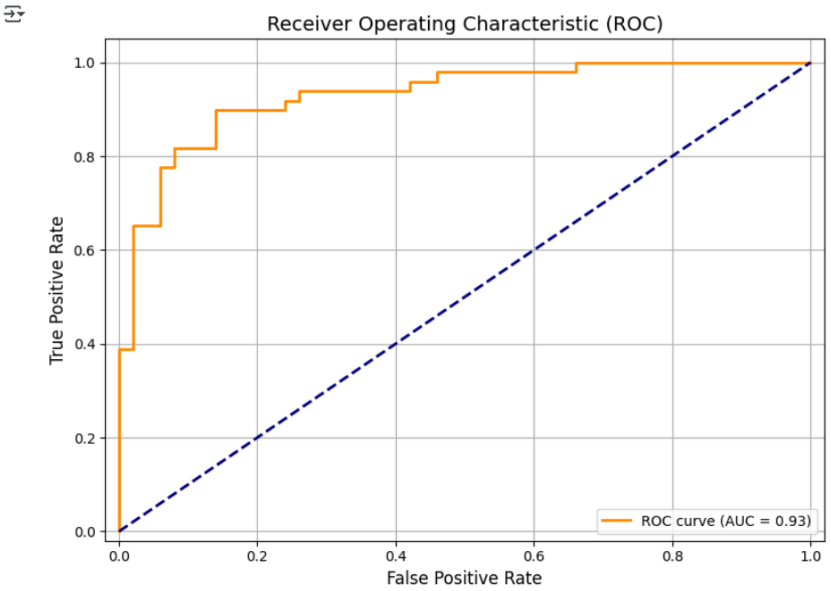
```
Test Accuracy: 0.8383838383838383
              precision    recall  f1-score   support

           0       0.83      0.86      0.84        50
           1       0.85      0.82      0.83        49

    accuracy                           0.84        99
   macro avg       0.84      0.84      0.84        99
weighted avg       0.84      0.84      0.84        99
```

On the test set, the model reached 84% accuracy. It predicted 43 true negatives and 40 true positives, with only 7 false positives and 9 false negatives.



On the test set, the model reached 84% accuracy. It predicted 43 true negatives and 40 true positives, with only 7 false positives and 9 false negatives.

The confusion matrix shows strong classification capability across both classes, with slightly better performance for class 0. Precision, recall, and F1-scores for both classes were around 0.83–0.85, indicating a well-performing and balanced classifier.



The ROC curve shows the trade-off between the true positive rate and false positive rate.
The model achieved an impressive AUC score of 0.93, indicating strong discriminative power.
AUC close to 1 suggests the classifier can effectively distinguish between classes.
The curve stays well above the diagonal, confirming excellent performance.
This further validates the SVM model's reliability on unseen data.

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 the 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**

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

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

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

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

(Refer any one)

Answer:

Dataset Description:

The Dry Bean Dataset contains detailed morphological and shape-based features extracted from images of dry beans. Each row represents a single bean.

Features:

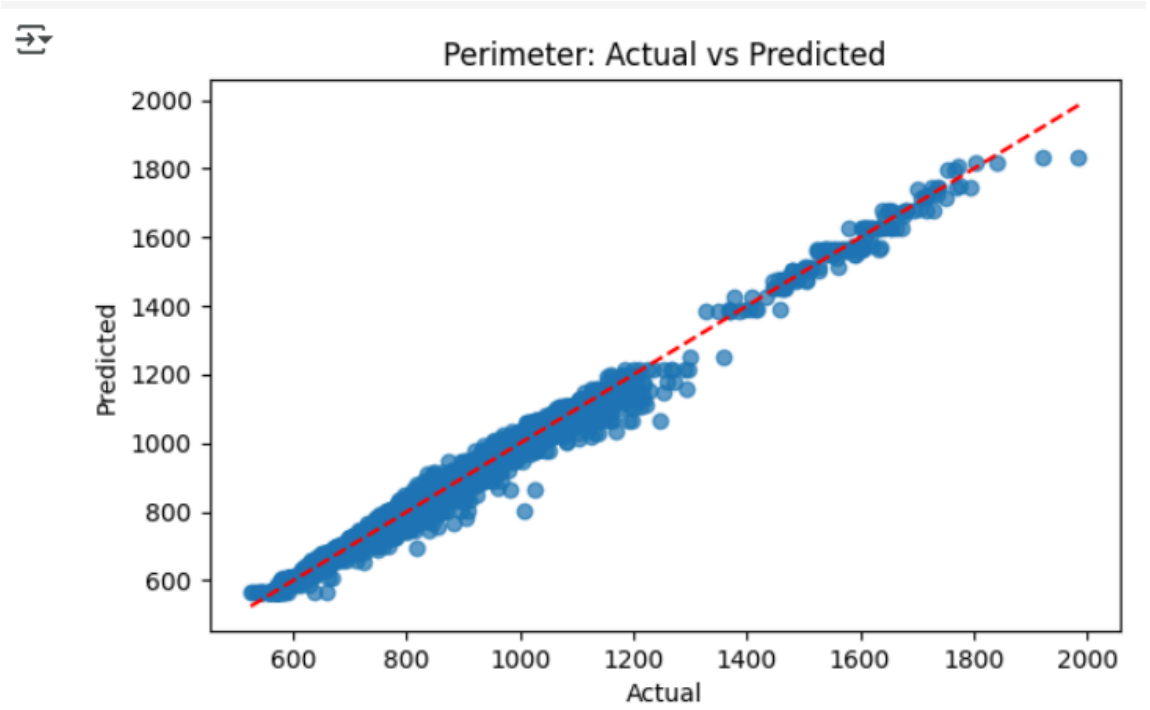
- Area: Total number of pixels inside the bean region (used as independent variable).
- Perimeter: Total distance around the bean boundary.
- MajorAxisLength: Length of the longest axis of the bean shape.
- MinorAxisLength: Length of the shortest axis of the bean shape.
- AspectRatio: Ratio between the major and minor axes.
- Eccentricity: A measure of how elongated the bean is.
- ConvexArea: Number of pixels in the convex hull of the bean region.
- EquivDiameter: Diameter of a circle with the same area as the bean.
- Extent: Ratio of bean area to the area of its bounding box.
- Solidity: Ratio of the area to the convex area.
- Roundness: Circularity of the bean (based on area and perimeter).
- Compactness, ShapeFactor1–4: Mathematical descriptors of the bean's geometrical shape.

Model: RandomForestRegressor

Result:

	R2 Score	Adjusted R2	MSE	MAE
Perimeter	0.988051	0.988048	527.650452	15.994392
MajorAxisLength	0.945817	0.945804	384.484421	15.063804
MinorAxisLength	0.923503	0.923485	145.448719	9.403722
AspectRation	0.379300	0.379148	0.037823	0.148942

The regression model performed exceptionally well in predicting the Perimeter of dry beans using only the Area as the input feature. It achieved an R^2 score of 0.988 and an Adjusted R^2 of 0.988, indicating that the model explains nearly all the variance in the perimeter values. The Mean Squared Error (MSE) was 527.65, and the Mean Absolute Error (MAE) was 15.99, both of which are low and indicate high accuracy. The scatter plot of actual vs predicted values confirms this strong performance, showing a tightly clustered line along the ideal diagonal, highlighting minimal prediction error.



In this regression task, we trained a machine learning model to predict multiple dependent variables (columns 2 to 5) using the first column as the independent variable from the provided dataset. We used an Object-Oriented approach to build a multi-output regression pipeline with Random Forest and performed hyperparameter tuning using GridSearchCV. A 70/30 random train-test split was applied. The model achieved high accuracy with Adjusted $R^2 > 0.99$ for all targets after tuning, confirming excellent prediction performance. Visualizations like Actual vs Predicted plots were generated to validate model effectiveness.

Q.6 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).

Answer:

Step 1: Understanding the Dataset

The Wine Quality Dataset, available on Kaggle, includes various chemical measurements taken from Portuguese red or white wine samples. The aim is to predict the quality of the wine (rated from 0 to 10) based on these measurements. You can find it by searching on Kaggle: "Wine Quality Dataset – UCI" or using this link:
https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

Step 2: Key Features in the Dataset

Here are the main features (columns) in the dataset and what they mean:

1. fixed acidity: These are non-volatile acids that do not evaporate easily.

2. volatile acidity: This represents acetic acid, which gives a sour or vinegar-like taste to the wine.
3. citric acid: A natural preservative that can add a fresh, citrusy note to the wine.
4. residual sugar: Sugar that remains after fermentation. It's more relevant in sweeter wines.
5. chlorides: Refers to the salt content in the wine.
6. free sulfur dioxide: This is unbound SO_2 that helps protect wine from harmful microbes.
7. total sulfur dioxide: The total amount of SO_2, including both free and bound forms.
8. density: The density of wine, which is influenced by sugar and alcohol levels.
9. pH: A measure of the wine's acidity (lower pH = more acidic).
10. sulphates: Added to improve preservation and stability; also influences taste.
11. alcohol: Indicates the alcohol content of the wine, expressed as a percentage.
12. quality: This is the target variable—it's the final wine quality score given by experts, ranging from 0 to 10.

Step 3: Importance of Features in Predicting Wine Quality

Not every feature contributes equally to predicting wine quality. Some have a major impact, while others are less influential.

1. Alcohol is very important. Generally, higher alcohol content is linked to better wine quality.
2. Volatile acidity is also very important but in a negative way. High levels make the wine taste sour, which lowers its quality.
3. Sulphates play a moderately important role. They help preserve wine and improve its stability, which can enhance the taste.
4. Citric acid has a moderate effect. It adds a fresh taste and improves flavor in the right amounts.
5. Residual sugar has a low to moderate impact. It's more relevant in sweet wines and doesn't influence the taste much in dry wines.
6. Chlorides have low importance. High salt content usually reduces the appeal of wine.
7. Free sulfur dioxide has low influence. It prevents spoilage but can make the wine taste harsh if overused.
8. Total sulfur dioxide has low to moderate importance. Too much can affect the aroma and taste negatively.
9. Density has low importance. It's related to other factors like sugar and alcohol, so it doesn't provide much extra information.
10. pH has low impact. While it reflects acidity, its effect on taste is indirect.
11. Fixed acidity has low to moderate importance. It contributes to sourness but varies depending on the wine type.

The most useful features when predicting wine quality are alcohol, volatile acidity, sulphates, and citric acid.

Step 4: Handling Missing Data and Common Imputation Techniques with Advantages and Disadvantages

Name:Bhushan Mukund Kor **Academic Year:2024-2025**

Division: D15C **Roll No: 28**

The Wine Quality dataset on Kaggle is generally clean, but in practical situations, missing values can appear due to data merging, corruption, or preprocessing errors. When this happens, it's important to handle the missing data properly to avoid misleading analysis or model results.

The most common imputation techniques used to fill in missing values are as follow:

1. Mean/Median Imputation

Fills missing values with the mean or median of the column.

Advantages:

- Easy and fast to implement.
- Preserves the general structure and scale of the data.

Disadvantages:

- Can reduce the natural variability of the data.
- Mean imputation is sensitive to outliers; median is better for skewed data.
- Does not consider relationships between features.

2. Mode Imputation

Replaces missing values with the most frequent value in the column (mainly for categorical data).

Advantages:

- Simple and effective for categorical features.
- Preserves the most common category.

Disadvantages:

- Not suitable for numerical or continuous data.
- May add bias if one value dominates.

3. K-Nearest Neighbors (KNN) Imputation

Uses the values of the nearest data points (neighbors) to estimate and fill in missing data.

Advantages:

- Consider patterns and relationships between features.
- Can provide more accurate estimations in structured datasets.

Disadvantages:

- Slower on large datasets.
- Results depend on the number of neighbors (k) and the distance metric used.
- Needs all features to be scaled properly.

4. Multiple Imputation by Chained Equations (MICE)

Builds models to predict missing values based on other features, cycling through each missing feature in rounds.

Advantages:

- Maintains relationships among variables.
- Produces multiple complete datasets, giving a sense of uncertainty and variation.

Disadvantages:

- More complex and slower than basic methods.

- Assumes that data is missing at random, which may not always be true.

5. Dropping Rows or Columns
Simply removes any row or column with missing data.
Advantages:

- Very easy to implement.
- No need for complex calculations.

Disadvantages:

- Can lead to data loss.
- Risk of bias if the missing values are not random.

Choosing the right imputation method depends on:

- The amount and type of missing data
- The size of the dataset
- Whether the data is numerical or categorical
- The importance of preserving feature relationships

In small datasets like Wine Quality, median or KNN imputation is often a good starting point, balancing simplicity and reliability.