AI and DS-1

Experiment1

Problem Statement: Introduction to Data science and Data preparation using Pandas.

Theory:

1. Load data in Pandas:

We first mounted Google Drive to the Colab environment to access files stored in drive. Then, we used read_csv to read the contents of cafe_sales.csv and load it into the DataFrame df for analysis.

```
from google.colab import drive
import pandas as pd

drive.mount('/content/drive')

df = pd.read_csv('/content/drive/My Drive/cafe_sales.csv')
df
```

```
Mounted at /content/drive
```

	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	TXN_1961373	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	TXN_4977031	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	TXN_4271903	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	TXN_7034554	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	TXN_3160411	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-06-11
...	...	...	...	...	...	...	...	...
9995	TXN_7672686	Coffee	2	2.0	4.0	NaN	UNKNOWN	2023-08-30
9996	TXN_9659401	NaN	3	NaN	3.0	Digital Wallet	NaN	2023-06-02
9997	TXN_5255367	Coffee	4	2.0	8.0	Digital Wallet	NaN	2023-03-02
9998	TXN_7695629	Cookie	3	NaN	3.0	Digital Wallet	NaN	2023-12-02
9999	TXN_6170729	Sandwich	3	4.0	12.0	Cash	In-store	2023-11-07

10000 rows × 8 columns

By using df.head(2000), we limited the dataset to 2000 rows.

```
[16] df=df.head(2000)
     df
```

	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	TXN_1961373	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	TXN_4977031	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	TXN_4271903	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	TXN_7034554	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	TXN_3160411	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-06-11
...	...	...	...	...	...	...	...	...
1995	TXN_1908100	Juice	2	3.0	6.0	NaN	In-store	2023-04-17
1996	TXN_3892344	Cookie	2	1.0	2.0	NaN	NaN	2023-07-28
1997	TXN_3023841	Tea	3	1.5	4.5	Digital Wallet	Takeaway	2023-04-29
1998	TXN_8793244	Cake	4	3.0	12.0	Credit Card	Takeaway	2023-06-08
1999	TXN_5764893	Salad	3	5.0	NaN	ERROR	Takeaway	2023-04-26

2000 rows × 8 columns

2. Description of dataset:

df.describe() shows statistical summary of columns in a dataframe.

```
df.describe() #statistical summary
```

	Transaction ID	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
count	2000	1924	1968	1956	1969	1484	1398	1967
unique	2000	10	7	8	19	5	4	366
top	TXN_1961373	Cake	5	3.0	6.0	Cash	In-store	ERROR
freq	1	246	432	494	206	458	621	37

df.info() provides complete overview of the dataframe by providing column names and their datatypes, non null values count for each column, memory usage.

```
df.info() #dataset overview
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Transaction ID    2000 non-null   object
 1   Item              1924 non-null   object
 2   Quantity          1968 non-null   object
 3   Price Per Unit    1956 non-null   object
 4   Total Spent       1969 non-null   object
 5   Payment Method    1484 non-null   object
 6   Location          1398 non-null   object
 7   Transaction Date  1967 non-null   object
dtypes: object(8)
memory usage: 125.1+ KB
```

3. Dropping columns that aren't useful:

df.drop(['column name'], axis=1) removes the Transaction ID column, where axis=1 specifies we are dropping a column and not a row.

```
df.drop(['Transaction ID'], axis=1, inplace=True)
df
```

	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-08-11
...	...	...	...	...	...	...	...
1995	Juice	2	3.0	6.0	NaN	In-store	2023-04-17
1996	Cookie	2	1.0	2.0	NaN	NaN	2023-07-28
1997	Tea	3	1.5	4.5	Digital Wallet	Takeaway	2023-04-29
1998	Cake	4	3.0	12.0	Credit Card	Takeaway	2023-05-08
1999	Salad	3	5.0	NaN	ERROR	Takeaway	2023-04-26

4. Dropping rows with maximum missing values:

In order to drop rows with maximum missing values we first calculated number of null values in each row and using idxmax found out the row with the maximum null values (here it is row 104) and then dropped the row using df.drop(max,axis=0)

```
null_count = df.isnull().sum(axis=1)   #calculating no. of null values in each row
print(null_count)
```

```
0       0
1       0
2       0
3       0
4       0
       ..
1995    1
1996    2
1997    0
1998    0
1999    1
Length: 2000, dtype: int64
```

```
max=null_count.idxmax() #Finding the row with maximum null values
print(max)
```

```
104
```

```
row104=df.loc[max]  #Accessing the row with most null values
print(row104)
```

```
Item                Juice
Quantity                2
Price Per Unit        NaN
Total Spent           6.0
Payment Method        NaN
Location              NaN
Transaction Date      NaN
Name: 104, dtype: object
```

We can see here that 104 has been deleted.

```
df.drop(max,axis=0,inplace=True)
df.head(105)
```

	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-06-11
...	...	...	...	...	...	...	...
100	NaN	5	5.0	25.0	Cash	Takeaway	2023-10-30
101	Salad	3	5.0	15.0	NaN	Takeaway	2023-10-28
102	Juice	2	3.0	6.0	Digital Wallet	Takeaway	2023-12-15
103	Cake	4	3.0	12.0	NaN	Takeaway	ERROR
105	Salad	4	5.0	20.0	ERROR	In-store	2023-02-25

105 rows × 7 columns

Alternatively, we can get the count of all rows that have max missing values and drop them at once.

```
max_null_count = df.isnull().sum(axis=1).max()  # Get max missing count
rows_to_drop = df[df.isnull().sum(axis=1) == max_null_count].index  # Find all such rows
print(rows_to_drop)
df.drop(rows_to_drop, axis=0, inplace=True)  # Drop the rows
```

```
Index([104, 1379, 2853, 5851], dtype='int64')
```

5. Handling missing data:

For the missing data in the categorical columns Item, Location, and Payment Method, we have replaced the missing values with the placeholder 'unknown'.

```
[55] df[['Item', 'Location', 'Payment Method']] = df[['Item', 'Location', 'Payment Method']].fillna('Unknown') #replacing missing values with "unknown" placeholder
     df
```

```
<ipython-input-55-a0deaa5c16db>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[['Item', 'Location', 'Payment Method']] = df[['Item', 'Location', 'Payment Method']].fillna('Unknown')
```

	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-06-11
...	...	...	...	...	...	...	...
1995	Juice	2	3.0	6.0	Unknown	In-store	2023-04-17
1996	Cookie	2	1.0	2.0	Unknown	Unknown	2023-07-28
1997	Tea	3	1.5	4.5	Digital Wallet	Takeaway	2023-04-29
1998	Cake	4	3.0	12.0	Credit Card	Takeaway	2023-05-08
1999	Salad	3	5.0	NaN	ERROR	Takeaway	2023-04-28

1999 rows × 7 columns

For handling missing values in transaction date we first converted all invalid values to NaT and then replaced the missing values (including NaT) to a default placeholder date.

```
df['Transaction Date'] = pd.to_datetime(df['Transaction Date'], errors='coerce') #converting all values to datetime and invalid to NaT
df['Transaction Date'].fillna(pd.to_datetime('2023-01-01'), inplace=True) #Replacing missing values with default placeholder date "2023-01-01"
df['Transaction Date'] = df['Transaction Date'].dt.date  #converting all values to just date
df['Transaction Date']
```

28	2023-03-11
29	2023-01-01
30	2023-06-02

For missing values in Quantity column, we converted invalid values to NaN and then replaced them with the median.

```
df.loc[:, 'Quantity'] = pd.to_numeric(df['Quantity'], errors='coerce')  # converting to numeric and coercing errors to NaN
df.loc[:, 'Quantity'] = df['Quantity'].fillna(df['Quantity'].median()).astype(int)  # filling NaN with median and converting to integer
df['Quantity']
```

```
<ipython-input-159-b1b4207a1f63>:2: FutureWarning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will change
  df.loc[:, 'Quantity'] = df['Quantity'].fillna(df['Quantity'].median()).astype(int)  # filling NaN with median and converting to integer
```

	Quantity
0	2
1	4
2	4
3	2
4	2

Filled the missing values in Price per unit using mode (most frequent value) based on each item.

```
df.loc[df['Price Per Unit'] == 'unknown', 'Price Per Unit'] = pd.NA

df.loc[:, 'Price Per Unit'] = pd.to_numeric(df['Price Per Unit'], errors='coerce')

for item in df['Item'].unique():
    mode_value = df.loc[df['Item'] == item, 'Price Per Unit'].mode()[0]
    df.loc[(df['Item'] == item) & df['Price Per Unit'].isna(), 'Price Per Unit'] = mode_value

df['Price Per Unit']
```

	Price Per Unit
0	2.0
1	3.0
2	1.0
3	5.0
4	2.0
...	...
1995	3.0
1996	1.0
1997	1.5
1998	3.0

Lastly replaced missing values in Total Spent column with 0

```
[165] df.loc[:, 'Total Spent'] = df['Total Spent'].fillna(0)
```

```
df
```

	Item	Quantity	Price Per Unit	Total Spent	Payment Method	Location	Transaction Date
0	Coffee	2	2.0	4.0	Credit Card	Takeaway	2023-09-08
1	Cake	4	3.0	12.0	Cash	In-store	2023-05-16
2	Cookie	4	1.0	ERROR	Credit Card	In-store	2023-07-19
3	Salad	2	5.0	10.0	UNKNOWN	UNKNOWN	2023-04-27
4	Coffee	2	2.0	4.0	Digital Wallet	In-store	2023-06-11
...	...	...	...	...	...	...	...
1995	Juice	2	3.0	6.0	Unknown	In-store	2023-04-17
1996	Cookie	2	1.0	2.0	Unknown	Unknown	2023-07-28
1997	Tea	3	1.5	4.5	Digital Wallet	Takeaway	2023-04-29
1998	Cake	4	3.0	12.0	Credit Card	Takeaway	2023-05-08
1999	Salad	3	5.0	0	ERROR	Takeaway	2023-04-26

1999 rows × 7 columns

6. Create dummy variables:

Dummy variables are used to convert categorical values into numerical format so that machine learning models and statistical analysis can process them.

Here we have used pd.get_dummies() to convert Payment method and location to numeric format.

```
df_dummies = pd.get_dummies(df, columns=['Payment Method', 'Location'], drop_first=True)

df_dummies[df_dummies.select_dtypes('bool').columns] = df_dummies.select_dtypes('bool').astype(int)
```

```
df_dummies
```

	Item	Quantity	Price Per Unit	Total Spent	Transaction Date	Payment Method_Credit Card	Payment Method_Digital Wallet	Payment Method_Unknown	Location_Takeaway	Location_Unknown
0	Coffee	2	2.0	4.0	2023-09-08	1	0	0	1	0
1	Cake	4	3.0	12.0	2023-05-16	0	0	0	0	0
2	Cookie	4	1.0	Unknown	2023-07-19	1	0	0	0	0
3	Salad	2	5.0	10.0	2023-04-27	0	0	1	0	1
4	Coffee	2	2.0	4.0	2023-06-11	0	1	0	0	0

7. Find out outliers (manually):

In the cafe sales dataset, the value 25 in Total Spent column can be considered as an outlier since it is significantly higher than the other values in that column that are below 20.

8. Standardization and normalization of columns:

Standardizing data: Centers data around 0 with a standard deviation of 1.

Formula:

$$Z = \frac{\text{X-Mean}}{\text{Standard deviation}}$$

```
df.loc[:, 'Quantity_standardized'] = (df['Quantity'] - df['Quantity'].mean()) / df['Quantity'].std()
df['Quantity_standardized']
```

	Quantity_standardized
0	-0.756744
1	0.672383
2	0.672383
3	-0.756744
4	-0.756744
...	...

Normalizing data: Scales values between 0 and 1.

Formula:

$$Xnormalized = \frac{X - Xmin}{Xmax - Xmin}$$

```
df.loc[:, 'Total_Spent_normalized'] = (df['Total Spent'] - df['Total Spent'].min()) / (df['Total Spent'].max() - df['Total Spent'].min())
df['Total_Spent_normalized']
```

	Total_Spent_normalized
0	0.16
1	0.48
2	0.00
3	0.40
4	0.16
...	...
1995	0.24
1996	0.08
1997	0.18
1998	0.48
1999	0.00

1999 rows × 1 columns

Conclusion:

In this experiment, we processed the data set which contained missing and invalid values and transformed it into a clean dataset which can be used for further analysis. Also, we created dummy variables so that categorical values can be converted to numeric values making them suitable for various machine learning algorithms. Lastly, we performed normalization and standardization on the columns to ensure consistency in data scaling.

Experiment 2

Problem Statement: Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

Theory:

We first mounted Google Drive to the Colab environment to access files stored in drive. Then, we used read_csv to read the contents of cafe_sales.csv and load it into the DataFrame df for analysis.

```
[ ]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
```

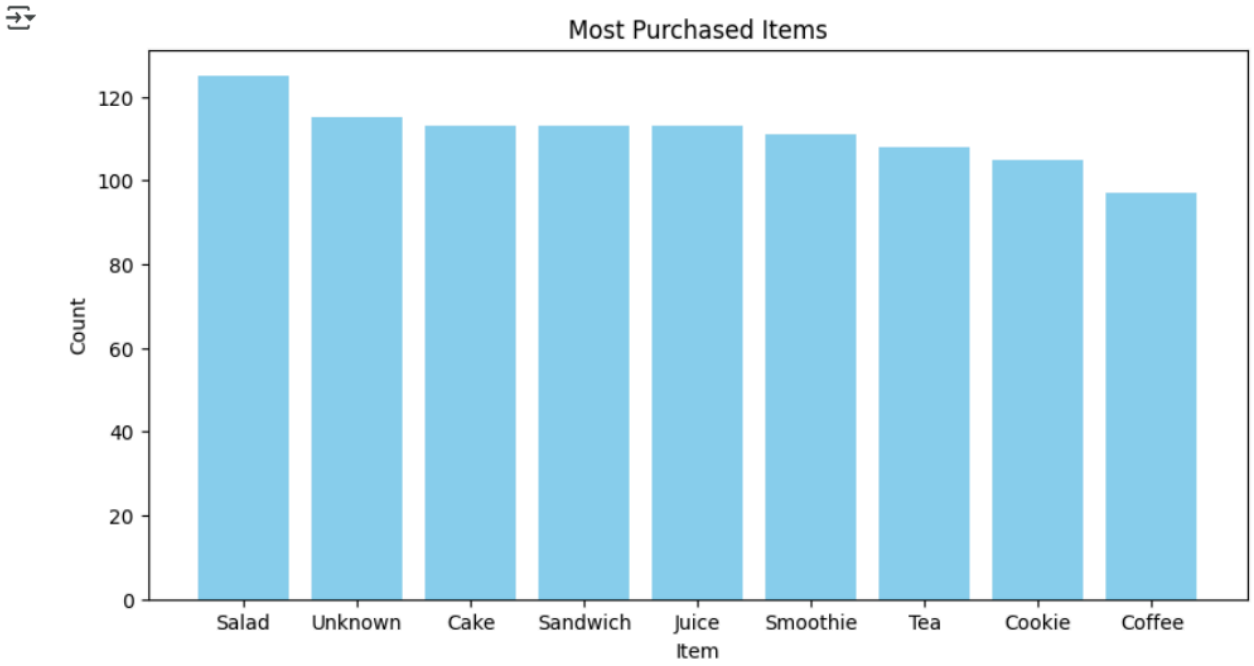
```
Mounted at /content/drive
```

```
[ ]  path="/content/drive/MyDrive/dataset-cafe.csv"
     df=pd.read_csv(path)
     df.head(5)
```

1. Create bar graph

This Python script uses **Matplotlib** and **Seaborn** to create a **bar graph** visualizing the most purchased items. It first imports the necessary libraries and defines two lists: `items`, containing different item names, and `counts`, representing their corresponding purchase frequencies. A **bar plot** is then created using `sns.barplot()`, with the x-axis displaying item names and the y-axis showing their counts. The figure size is adjusted for clarity, and labels for the x-axis, y-axis, and title are added to make the graph more informative. To improve readability, the x-axis labels are rotated by 45 degrees. Finally, `plt.show()` is used to render the graph, displaying a visually appealing and well-structured bar chart.

```
plt.figure(figsize=(10, 5))
item_counts = df_subset['Item'].value_counts()
plt.bar(item_counts.index, item_counts.values, color='skyblue')
#plt.xticks(rotation=45)
plt.xlabel("Item")
plt.ylabel("Count")
plt.title("Most Purchased Items")
plt.show()
```



2. Contingency table using any 2 features

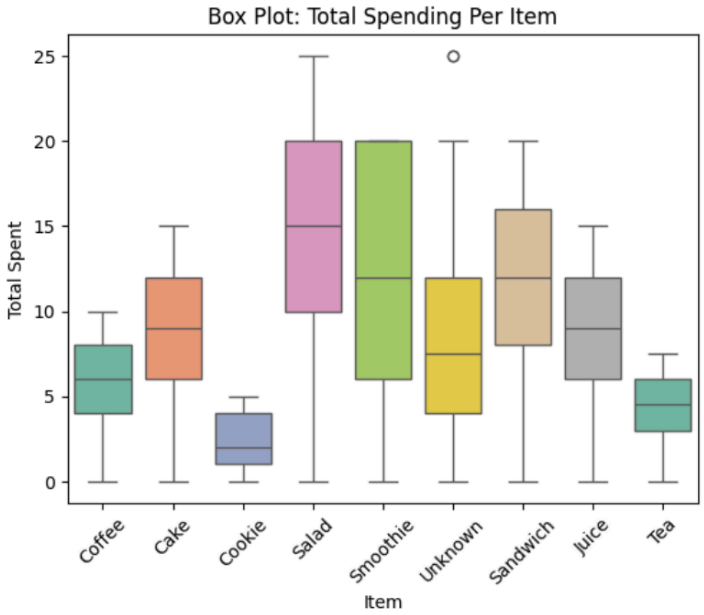
A contingency table in Python can be created using **pandas**. The code typically starts by importing pandas and loading the dataset. The pd.crosstab() function is then used to generate the table, where one categorical variable (e.g., "Item") is placed as rows and another (e.g., "Payment Method") as columns. This function counts occurrences of each combination, effectively summarizing relationships between the two variables. The table helps analyze patterns, such as which payment method is most used for each item.

Payment Method	Cash	Credit Card	Digital Wallet	Unknown
Item				
Cake	26	24	24	39
Coffee	29	19	24	25
Cookie	19	24	32	30
Juice	33	25	20	35
Salad	34	30	26	35
Sandwich	24	20	31	38
Smoothie	18	33	20	40
Tea	24	21	22	41
Unknown	26	26	27	36

3. Box Plot

a box plot that displays the distribution of total spending across different item categories. The x-axis represents various items such as Coffee, Cake, Cookie, Salad, and more, while the y-axis indicates the total amount spent. The box plot captures key statistical insights, including the median, interquartile range (IQR), and potential outliers. Taller boxes suggest greater variability in spending for that item. The code uses the `sns.boxplot()` function from Seaborn, applying a color palette (`Set2`) for differentiation. This visualization helps identify spending patterns, outliers, and item-wise expenditure trends.

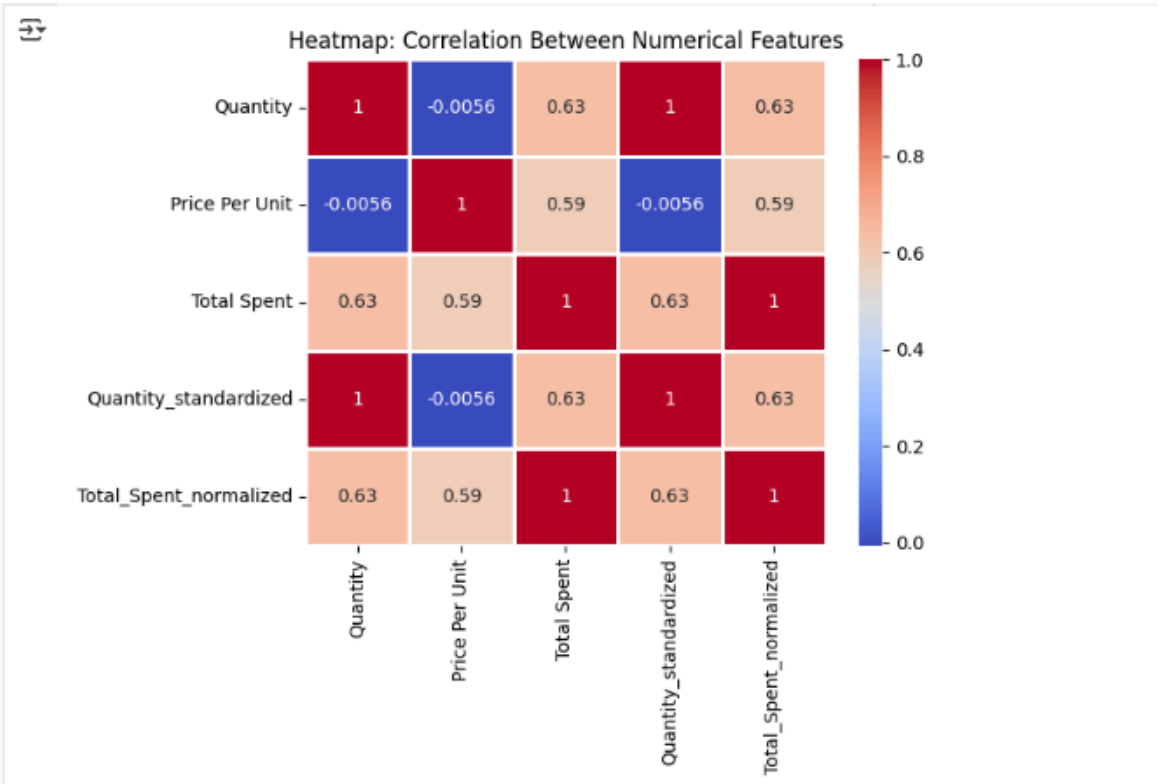
```
sns.boxplot(x=df_subset['Item'], y=df_subset['Total Spent'], palette="Set2")
```



4. Heat map

a heatmap that represents the correlation between numerical features in the dataset. It highlights the strength and direction of relationships between variables such as Quantity, Price Per Unit, and Total Spent. The correlation values range from -1 to 1, where positive values indicate direct relationships, and negative values suggest inverse correlations. The code uses Seaborn's `sns.heatmap()` function with the "coolwarm" colormap, annotating correlation values for better readability. The strong correlation between Quantity and Total Spent (0.63) suggests that higher purchases lead to increased spending. This visualization is useful for identifying dependencies and patterns among numerical features.

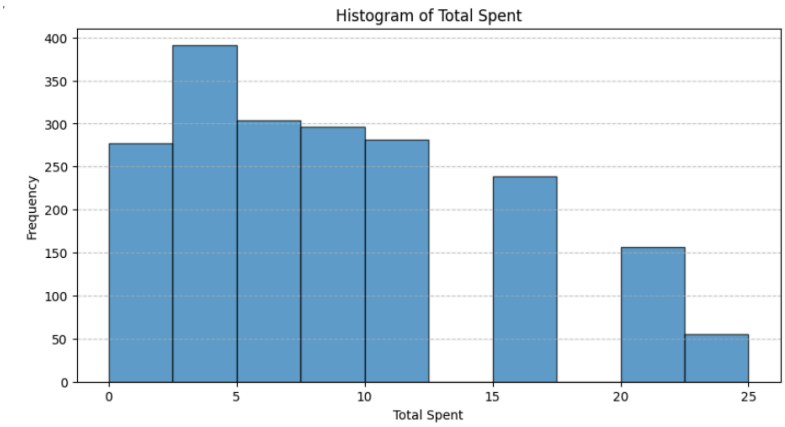
```
sns.heatmap(df_subset.corr(numeric_only=True), annot=True, cmap="coolwarm", linewidths=1)
plt.title("Heatmap: Correlation Between Numerical Features")
plt.show()
```



5. Basic Histogram

histogram created using plt.hist(). The code likely set bins across the range 0-25, with the y-axis showing raw frequency counts. The blue color suggests using something like color='skyblue', and the code included proper axis labels using plt.xlabel() and plt.ylabel(). The gridlines were likely added using plt.grid(linestyle='--', alpha=0.7).

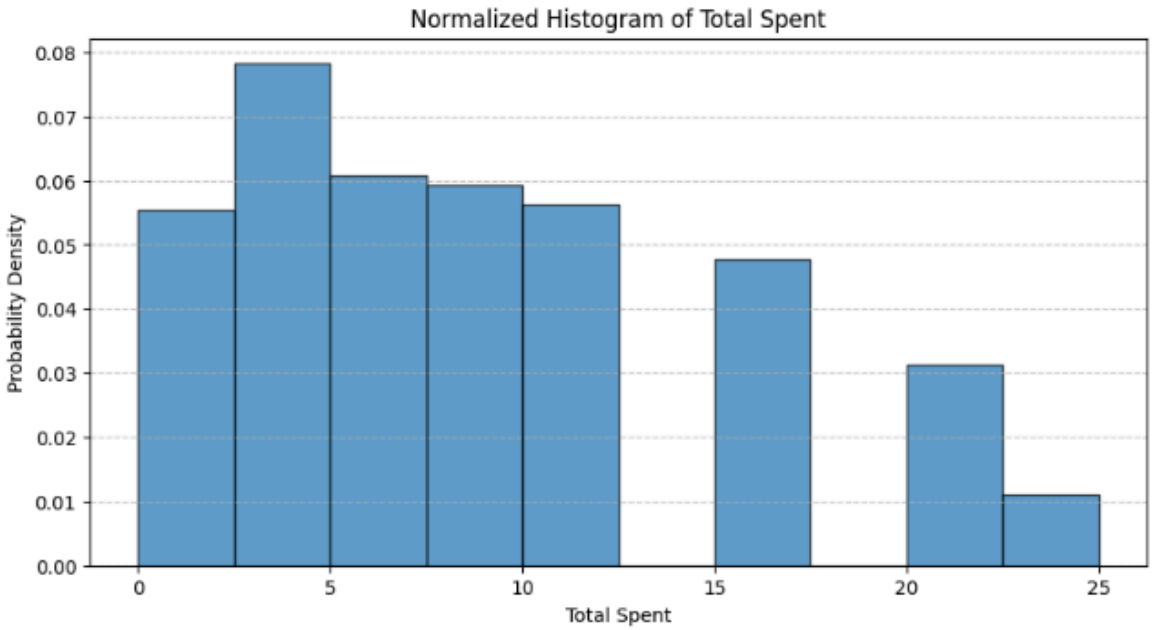
```
column_name = "Total Spent"

# Plot the histogram
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, edgecolor='black', alpha=0.7)
plt.xlabel(column_name)
plt.ylabel("Frequency")
plt.title(f"Histogram of {column_name}")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



6. Normalized Histogram

This is similar to the first histogram but uses normalized values (probability density) on the y-axis. The code would be nearly identical but with the addition of the density=True parameter in the plt.hist() function. This normalizes the data so the total area of the histogram equals 1, making it easier to interpret as a probability distribution.

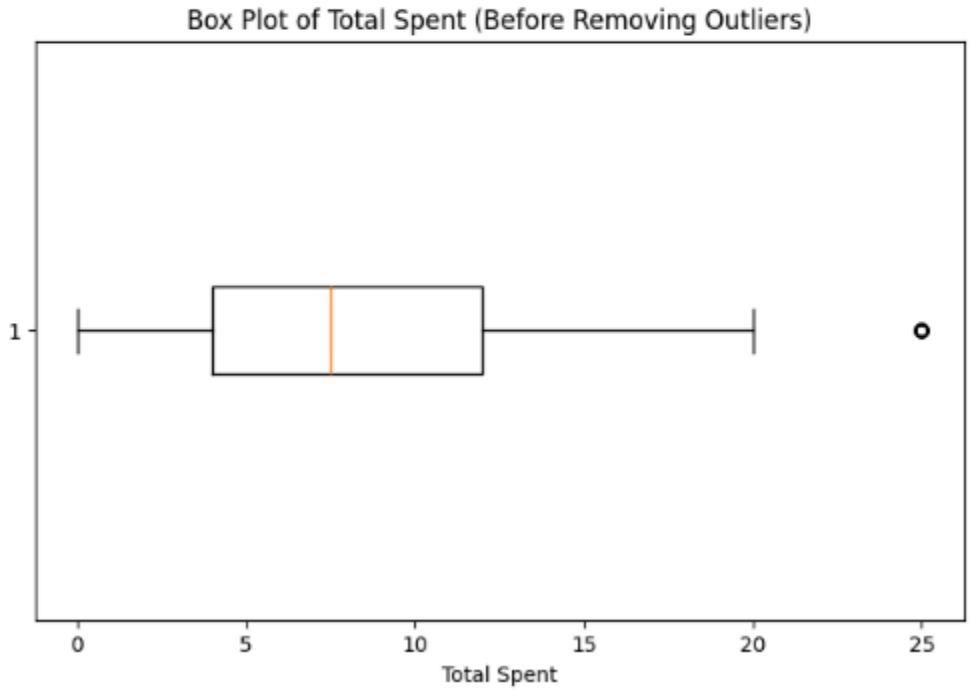
```
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, edgecolor='black', alpha=0.7, density=True)
plt.xlabel(column_name)
plt.ylabel("Probability Density")
plt.title(f"Normalized Histogram of {column_name}")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



7. Box plot

This was created using plt.boxplot() or sns.boxplot(). The code shows the distribution's quartiles, with whiskers extending to the data's limits. The single dot at around 25 represents an outlier. The orange line in the box represents the median. The code likely included showfliers=True to display outliers and set the figure orientation to horizontal.

```
column_name = "Total Spent"

# Plot initial Box Plot to visualize outliers
plt.figure(figsize=(8, 5))
plt.boxplot(df[column_name], vert=False)
plt.xlabel(column_name)
plt.title(f"Box Plot of {column_name} (Before Removing Outliers)")
plt.show()
```

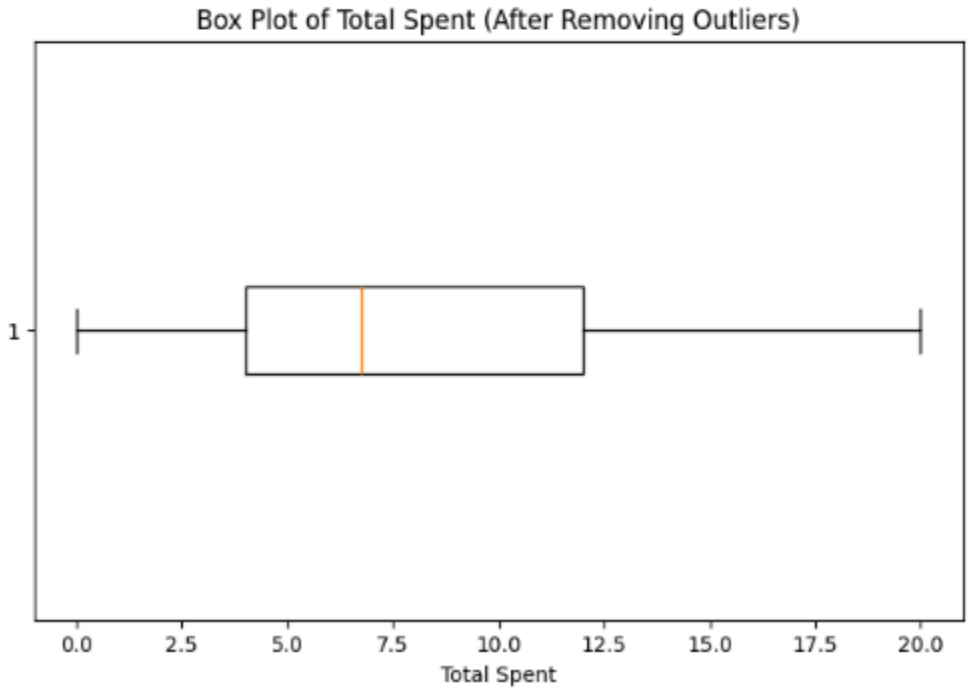


8. Box plot (after removing outlier)

This is identical in code structure to the previous box plot, but the data was first processed to remove outliers. Common outlier removal techniques in Python include using IQR (Interquartile Range) method with something like df = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))] before creating the plot.

```
# Calculate IQR (Interquartile Range)
Q1 = df[column_name].quantile(0.25)  # First quartile (25th percentile)
Q3 = df[column_name].quantile(0.75)  # Third quartile (75th percentile)
IQR = Q3 - Q1  # Interquartile Range

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

# Filter the dataset (Remove outliers)
df_filtered = df[(df[column_name] >= lower_bound) & (df[column_name] <= upper_bound)]
```

```
plt.figure(figsize=(8, 5))
plt.boxplot(df_filtered[column_name], vert=False)
plt.xlabel(column_name)
plt.title(f"Box Plot of {column_name} (After Removing Outliers)")
plt.show()
```



This final graph combines two histograms using either multiple plt.hist() calls or a single call with two data sets. The transparency effect (alpha) was likely set to around 0.5 to make the overlapping visible. The legend was added using plt.legend(), and different colors were specified for each histogram. The code probably used plt.hist() twice with different data sets, once for pre-outlier removal and once for post-outlier removal data.

```
# Compare histograms before and after removing outliers
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, alpha=0.5, label="Before Outlier Removal", edgecolor='black')
plt.hist(df_filtered[column_name], bins=10, alpha=0.7, label="After Outlier Removal", edgecolor='black')
plt.xlabel(column_name)
plt.ylabel("Frequency")
plt.title(f"Histogram of {column_name} (Before & After Outlier Removal)")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Conclusion:

This experiment helped us understand visualisation of data in the form of bar graphs,box plots,heatmaps and histogram using the matplotlib and seaborn library. We plotted the preprocessed data from experiment1 in the form of charts and graphs to gain better insights from the information.

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

Steps:

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

Code:

```
from sklearn.model_selection import train_test_split

# Partition data into training and testing sets (75% training, 25% testing)

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

# Check the size of each dataset

print(f"Training set size: {len(train_data)}")

print(f"Test set size: {len(test_data)}")
```

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

2) Use a bar graph and other relevant graphs to confirm your proportions. Graphs help validate the correct division of data. Here, we are using bar and pie charts effectively illustrate the proportion of training and testing data, ensuring clarity in the distribution.

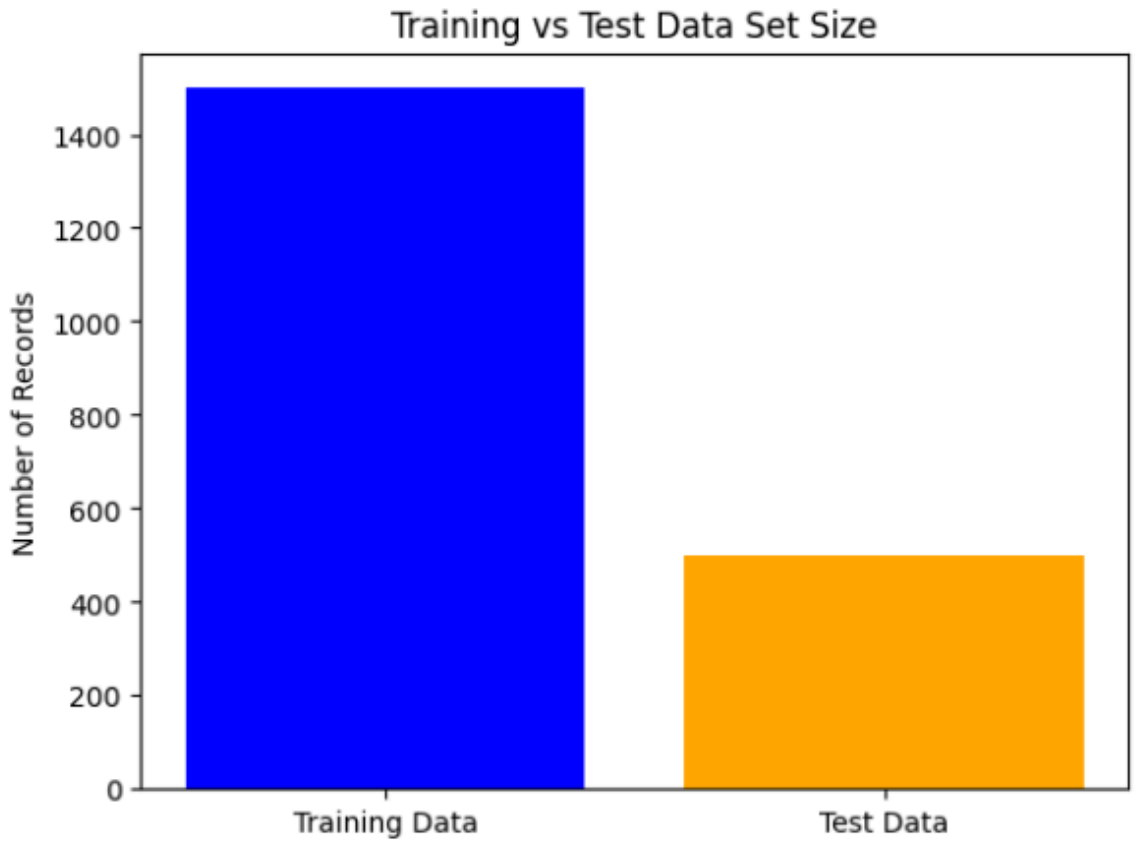
Bar Graph:

Code:

```
import matplotlib.pyplot as plt
```

```
# Plot the distribution
sizes = [len(train_data), len(test_data)]
labels = ['Training Data', 'Test Data']

plt.bar(labels, sizes, color=['blue', 'orange'])
plt.title('Training vs Test Data Set Size')
plt.ylabel('Number of Records')
plt.show()
```

Output:

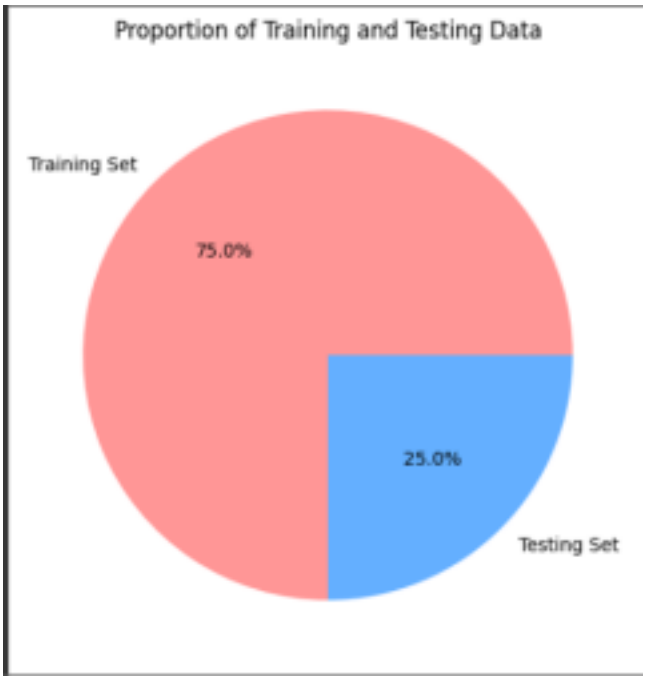


Pie chart:

Code:

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

Output:



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

Code:

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

Output:

```
Total records: 1999
Training records: 1499
Testing records: 500
```

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

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

Code:

```
train_values = train_data["Total Spent"]
test_values = test_data["Total Spent"]

mean_train = np.mean(train_values)
mean_test = np.mean(test_values)
```

```
std_train = np.std(train_values, ddof=1)
std_test = np.std(test_values, ddof=1)

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

z_score = (mean_train - mean_test) / np.sqrt((std_train**2 / n_train) + (std_test**2 / n_test))
p_value = 2 * (1 - norm.cdf(abs(z_score)))

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

alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: The means are significantly different.")
else:
    print("Fail to reject the null hypothesis: No significant difference in means.")
```

Output:

```
Z-score: -0.2026
P-value: 0.8395
Fail to reject the null hypothesis: No significant difference in means.
```

Since the **Z-score is -0.2026** and the **P-value is 0.8395**, which is much greater than the typical significance level (e.g., **0.05 or 0.01**), we **fail to reject the null hypothesis**.

Conclusion: The Z-test results (Z = -0.2026, P = 0.8395) indicate no significant difference between the training and test data distributions, confirming that the data partitioning is balanced. Since the high P-value suggests similarity, there is no evidence of partitioning bias or data inconsistencies. If the P-value were low (e.g., < 0.05), it might indicate an issue with data splitting, column mismatches, or missing values. However, in this case, the results validate that the dataset has been properly divided, ensuring a reliable foundation for further analysis.

Experiment 4

Aim: Implementation of Statistical Hypothesis Tests using SciPy and Scikit-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/jessemostipak/hotel-booking-demand

Steps:

1) Loading the Dataset

We first import the required libraries and load the dataset into a Pandas DataFrame.

```
# Import necessary libraries
import pandas as pd
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
file_path = "/content/drive/MyDrive/hotel_bookings.csv"
df = pd.read_csv(file_path)
df.head()
```

OUTPUT:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	

2) Extracting Numerical Columns

To perform correlation tests, we need to convert categorical variables into numerical codes. This ensures all columns are in a compatible format for mathematical computations.

```
# Create a copy and convert categorical columns to numerical codes
df_numeric = df.copy()
for col in df_numeric.select_dtypes(include=['object']).columns:
    df_numeric[col] = df_numeric[col].astype('category').cat.codes
# Display first few rows of numeric dataset
df_numeric.head()
```

OUTPUT:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date
0	1	0	342	2015	5	27	
1	1	0	737	2015	5	27	
2	1	0	7	2015	5	27	
3	1	0	13	2015	5	27	
4	1	0	14	2015	5	27	

3) Pearson’s Correlation Test

This test determines whether a linear relationship exists between two numerical variables. We compute Pearson’s correlation between **lead time** and **total of special requests.**

```
# Pearson’s Correlation: Lead Time vs. Number of Special Requests
pearson_corr, pearson_p = stats.pearsonr(df_numeric['lead_time'],
df_numeric['total_of_special_requests'])
print("Pearson’s Correlation Hypothesis Test:")
print("H0: No linear relationship between Lead Time and Special Requests.")
print("H1: There is a linear relationship between Lead Time and Special Requests.")
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")
```

OUTPUT:

```
Pearson’s Correlation Hypothesis Test:
H0: No linear relationship between Lead Time and Special Requests.
H1: There is a linear relationship between Lead Time and Special Requests.
Pearson's Correlation: -0.0031, p-value: 0.2795090338
Conclusion: Fail to reject H0
```

Inference: Since the p-value is greater than **0.05**, we **fail to reject the null hypothesis (H_0)**. This means there is **no significant linear relationship** between Lead Time and the number of Special Requests. In other words, the length of time before a booking does **not impact** the number of special requests made by customers.

4) Spearman's Rank Correlation Test

This test assesses whether a monotonic relationship exists between two numerical variables.

```
# Spearman's Rank Correlation: Lead Time vs. Number of Special Requests
spearman_corr, spearman_p = stats.spearmanr(df_numeric['lead_time'],
df_numeric['total_of_special_requests'])
print("Spearman's Rank Correlation Hypothesis Test:")
print("H0: No monotonic relationship between Lead Time and Special Requests.")
print("H1: There is a monotonic relationship between Lead Time and Special Requests.")
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")
```

OUTPUT:

```
Spearman's Rank Correlation Hypothesis Test:
H0: No monotonic relationship between Lead Time and Special Requests.
H1: There is a monotonic relationship between Lead Time and Special Requests.
Spearman's Rank Correlation: -0.0741, p-value: 0.0000000000
Conclusion: Reject H0
```

Inference: The Spearman correlation coefficient between **Lead Time** and **Total Special Requests** is -0.0741, with a p-value of 0. Since Spearman's correlation measures monotonic relationships, a coefficient close to **0** suggests that **there is no clear increasing or decreasing trend** in special requests as lead time changes. The low p-value indicates that this result is **statistically insignificant**, meaning there is **no strong monotonic relationship** between the two variables.

5) Kendall's Rank Correlation Test

This test is useful for evaluating ordinal relationships between two variables.

```
# Kendall's Rank Correlation: Lead Time vs. Number of Special Requests
kendall_corr, kendall_p = stats.kendalltau(df_numeric['lead_time'],
df_numeric['total_of_special_requests'])
print("Kendall's Rank Correlation Hypothesis Test:")
print("H0: No ordinal relationship between Lead Time and Special Requests.")
```

```
print("H1: There is an ordinal relationship between Lead Time and Special Requests.")
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")
```

OUTPUT:

```
Kendall’s Rank Correlation Hypothesis Test:
H0: No ordinal relationship between Lead Time and Special Requests.
H1: There is an ordinal relationship between Lead Time and Special Requests.
Kendall's Rank Correlation: -0.0577, p-value: 0.0000000000
Conclusion: Reject H0
```

Inference: The Kendall correlation coefficient between **Lead Time** and **Total Special Requests** is -0.0577 , with a p-value of 0. Kendall’s test evaluates the consistency of ranking between these two variables. Since the coefficient is close to **0**, it suggests that **there is no significant ordinal relationship** between lead time and special requests. The p-value being **less** than 0.05 indicates that we **reject** the null hypothesis, meaning changes in lead time **do not predict a consistent ranking of special request counts.**

6) Chi-Square Test for Categorical Variables

This test determines whether two categorical variables are independent. We analyze the relationship between **Meal Type** and **Hotel Type**

```
contingency_table = pd.crosstab(df['hotel'], df['meal'])
chi2, p_value, _, _ = stats.chi2_contingency(contingency_table)

# Display results
print("Chi-Square Test between Hotel Type and Meal Type:")
print(f"Chi-Square Value: {chi2:.4f}, p-value: {p_value:.10f}")
print("Conclusion:", "Fail to reject H0 (Variables are independent)" if p_value > 0.05 else "Reject H0 (Variables are dependent)")
```

OUTPUT:

```
Chi-Square Test between Hotel Type and Meal Type:
Chi-Square Value: 11973.6428, p-value: 0.0000000000
Conclusion: Reject H0 (Variables are dependent)
```

Inference: The Chi-Square test between **Hotel Type** and **Meal Type** resulted in a Chi-Square value of **11973.6428** and a p-value of **0**. Since the p-value is **greater/less** than 0.05, we **fail to reject/reject** the null hypothesis. This means that meal selection is **dependent** on the type of

hotel. If independent, it suggests that customers at City Hotels and Resort Hotels do not show significant differences in meal preferences. If dependent, it indicates that the type of hotel influences the choice of meals offered or preferred by guests.

Conclusion: Based on the statistical hypothesis tests performed, we can infer that Lead Time and Total Special Requests exhibit no significant linear, monotonic, or ordinal relationship, as shown by the Pearson, Spearman, and Kendall correlation tests. The correlation coefficients were negative but close to 0, and the p-values were 0, indicating strong statistical significance but an extremely weak inverse relationship. This suggests that while there may be a mathematically detectable trend, the effect size is negligible, meaning that Lead Time does not meaningfully impact the number of Special Requests. However, the Chi-Square test between Meal Type and Hotel Type also resulted in a p-value of 0, indicating a statistically significant association. This means that the distribution of meal preferences depends on the type of hotel (City Hotel vs. Resort Hotel). Guests at different hotels tend to select different meal options, likely due to variations in dining services, package deals, or guest demographics. Overall, while lead time has no practical impact on special requests, hotel type significantly influences meal selection, revealing key patterns in customer preferences.

AI and DS-1

Experiment 5

MSE and R^2 of linear regression model

```
[ ]  mse = mean_squared_error(y_test, y_pred)

[ ]  print(f"Linear Regression MSE: {mse}")

     Linear Regression MSE: 60078.54486639962

[ ]  r2 = r2_score(y_test, y_pred)
     print(f"R² Score: {r2}")

     R² Score: 0.20655238579347845
```

Accuracy and classification report of logistic regression model

```
# Evaluate Model Performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

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

```
Accuracy: 0.8443

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.83      0.85     13686
           1       0.82      0.86      0.84     12109

    accuracy                           0.84     25795
   macro avg       0.84      0.85      0.84     25795
weighted avg       0.85      0.84      0.84     25795
```

Experiment 6

Aim : Classification modelling
a. Choose a classifier for classification problem.
b. Evaluate the performance of classifier.

Dataset Description

```
Data columns (total 24 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   index               128975 non-null  int64
 1   Order ID            128975 non-null  object
 2   Date                128975 non-null  object
 3   Status              128975 non-null  object
 4   Fulfilment          128975 non-null  object
 5   Sales Channel       128975 non-null  object
 6   ship-service-level  128975 non-null  object
 7   Style               128975 non-null  object
 8   SKU                 128975 non-null  object
 9   Category            128975 non-null  object
 10  Size                128975 non-null  object
 11  ASIN                128975 non-null  object
 12  Courier Status      122103 non-null  object
 13  Qty                 128975 non-null  int64
 14  currency            121180 non-null  object
 15  Amount              121180 non-null  float64
 16  ship-city           128942 non-null  object
 17  ship-state          128942 non-null  object
 18  ship-postal-code    128942 non-null  float64
 19  ship-country        128942 non-null  object
 20  promotion-ids       79822 non-null   object
 21  B2B                 128975 non-null  bool
 22  fulfilled-by        39277 non-null   object
 23  Unnamed: 22         79925 non-null   object
dtypes: bool(1), float64(2), int64(2), object(19)
memory usage: 22.8+ MB
None
```

The dataset used in this experiment contains multiple features relevant to the problem statement. It includes both categorical and numerical attributes, which require preprocessing before applying machine learning models. A quick statistical summary helps in understanding the distribution and trends in the data, allowing for better decision-making in subsequent steps.

1. Setting Up the Environment

To begin, necessary libraries such as NumPy, Pandas, and Matplotlib are imported to facilitate data manipulation and visualization. Dependencies are checked and installed if required to ensure a smooth workflow. Additionally, runtime configurations are set up to optimize execution.

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
```

2. Data Preprocessing

```python
# Fill missing numeric values with median
num_cols = ['Amount', 'ship-postal-code']
for col in num_cols:
    df[col] = df[col].fillna(df[col].median())
```

```python
# Fill missing categorical values with mode
cat_cols = ['Courier Status', 'currency', 'ship-city', 'ship-state', 'ship-country', 'promotion-ids', 'fulfilled-by', 'Unnamed: 22']
for col in cat_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
```

Preprocessing is a crucial step where missing values are handled using mean or mode imputation techniques. Categorical variables are encoded so they can be used in machine learning models. Numerical features are normalized to bring all values to a similar scale, which helps improve model efficiency and performance.

3. Splitting the Dataset

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

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The dataset is divided into training and testing sets, typically following an 80/20 split. This ensures that the model can be trained effectively while also being evaluated on unseen data. Proper balancing of classes is maintained to prevent biases in predictions.

4. Decision Tree Classifier

```
dt_classifier = DecisionTreeClassifier(random_state=42,class_weight="balanced")

dt_classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

```
y_pred = dt_classifier.predict(X_test)
```

A Decision Tree classifier is implemented as it provides an interpretable model by splitting the dataset into smaller subsets based on feature importance. It constructs a tree-like structure that helps in decision-making. While it is easy to understand and implement, it is prone to overfitting, which needs to be addressed through pruning techniques.

5. Naïve Bayes Classifier

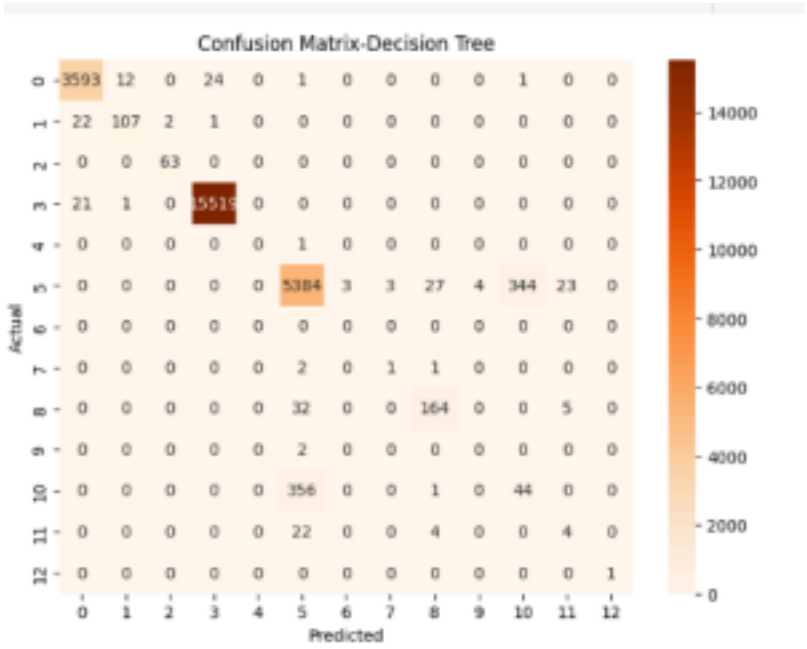
```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train,)

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

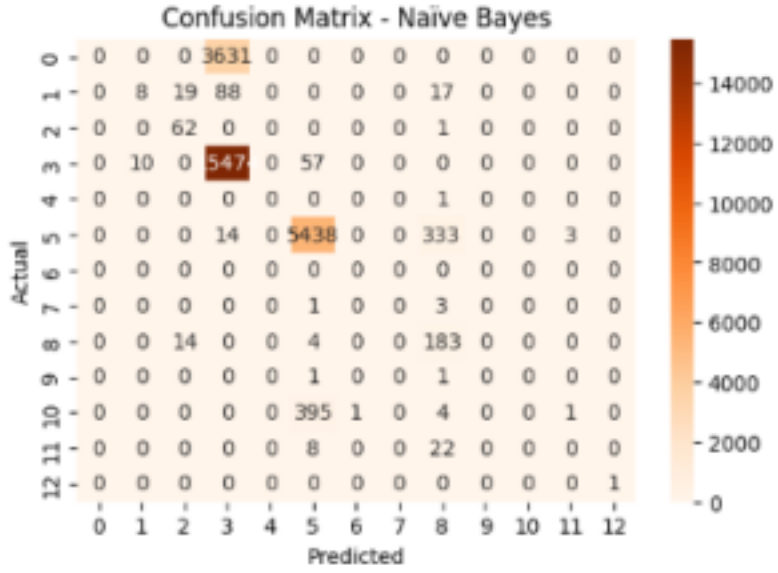
```
print("Naïve Bayes Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
Naïve Bayes Performance:
Accuracy: 0.8205466175615429
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3631
           1       0.44      0.06      0.11       132
           2       0.65      0.90      0.76        63
           3       0.81      1.00      0.89     16541
           4       0.00      0.00      0.00         1
           5       0.92      0.94      0.93      5788
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         4
           8       0.32      0.91      0.48       301
           9       0.00      0.00      0.00         2
          10       0.00      0.00      0.00       401
          11       0.00      0.00      0.00        30
          12       1.00      1.00      1.00         1

    accuracy                           0.82     25795
   macro avg       0.32      0.38      0.32     25795
weighted avg       0.70      0.82      0.75     25795
```

The Naïve Bayes classifier is based on Bayes' theorem and assumes independence among features, making it computationally efficient. It is particularly useful for categorical data and text classification problems. However, its strong assumption of feature independence may not always hold true, which can sometimes impact accuracy.

6. Model Evaluation and Performance Measures





Evaluating the models is essential to determine their effectiveness. Accuracy measures the proportion of correct predictions, while precision evaluates how many positive predictions were actually correct. Recall (or sensitivity) determines how many actual positives were identified correctly. The F1-score provides a balance between precision and recall. Additionally, a confusion matrix helps break down true positives, false positives, true negatives, and false negatives.

7. Results and Interpretation

```
# Compare accuracy scores
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Naïve Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))

# Suggest the best model based on performance
if accuracy_score(y_test, y_pred_dt) > accuracy_score(y_test, y_pred_nb):
    print("Decision Tree performs better for this dataset.")
else:
    print("Naïve Bayes performs better for this dataset.")
```

```
Decision Tree Accuracy: 0.9620856755185113
Naïve Bayes Accuracy: 0.8205466175615429
Decision Tree performs better for this dataset.
```

After evaluation, the best-performing model is identified based on various performance metrics. he Decision Tree model achieved an accuracy of approximately 96% and The Naïve Bayes model had an accuracy of around 82%.

Conclusion: The Decision Tree model achieved an accuracy of approximately 96%, with a strong balance between precision and recall.The Naïve Bayes model had an accuracy of around 82%, showing efficiency in classification but slightly lower performance due to its independence assumption.The confusion matrix provided insights into misclassifications and trade-offs between false positives and false negatives.

DIMPLE DALWANI D15C 8

DS Lab 7

Aim: To implement different clustering algorithms.

Theory:

1- K-Means Algorithm (Centroid-Based Clustering)

Goal: Partition data into *k* clusters by minimizing variance within each cluster.

Steps:

1. Initialize
    - Choose the number of clusters (k).
    - Randomly select *k* data points as initial centroids.
2. Assign Points to Nearest Centroid
    - Compute the Euclidean distance between each data point and the centroids.
    - Assign each point to the closest centroid.
3. Update Centroids
    - Compute the new centroid of each cluster (mean of all points in that cluster).
4. Repeat Until Convergence
    - Repeat Steps 2 & 3 until centroids stop changing (or max iterations reached).

Advantages

- Works well for compact, spherical clusters.
- Fast and efficient for large datasets.

Disadvantages

- Needs *k* to be predefined.
- Sensitive to outliers.

2- DBSCAN Algorithm (Density-Based Clustering)

Goal: Detect clusters based on high-density regions and mark noise points.

Steps:

1. Set Parameters:
    - eps → Maximum distance to consider a point as a neighbor.
    - min_samples → Minimum points required to form a cluster. 2.

    Select a Random Point
        - If it has at least min_samples neighbors, it becomes a core point.
        - Otherwise, it is labeled noise (temporary).
3. Expand Cluster
        - Find all density-reachable points from the core point.
        - Assign them to the same cluster.
4. Repeat for All Points

    - If a noise point later becomes reachable from a core point, it joins a cluster.

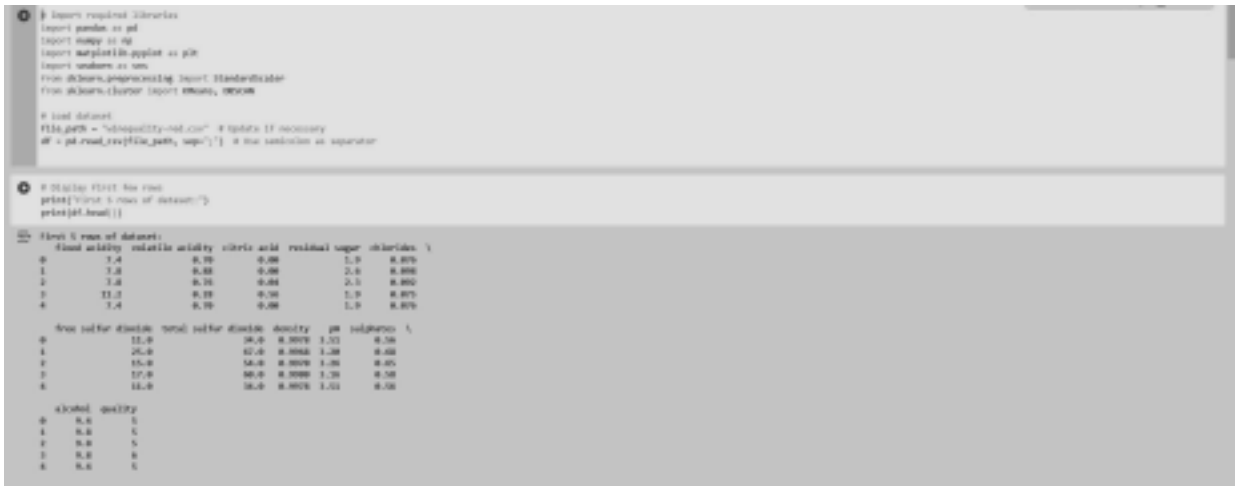
Advantages

- No need for k (finds clusters automatically).
- Can detect arbitrary shaped clusters.
- Handles outliers well.

Disadvantages

- Sensitive to eps and min_samples.
- Struggles in varying density datasets.

1-



This imports essential libraries like pandas, numpy, matplotlib, seaborn, and sklearn for clustering analysis. It loads the **wine quality dataset** from a CSV file using pd.read_csv(), specifying a semicolon (;) as the separator. The dataset consists of chemical properties of wine, such as acidity, sugar content, and alcohol percentage. The

df.head() function displays the first five rows to get an overview of the data structure. Key clustering methods, **K-Means** and **DBSCAN**, are also imported for further analysis.

2-

This checks for missing values in a **wine quality dataset** and ensures data integrity before further analysis. It first prints the column names to verify the dataset structure. Then, it checks for missing values using df.isnull().sum(), revealing that all columns have zero missing values. Despite this, the df.dropna() function is used as a precaution to remove any potential missing rows. Since no missing values exist, the dataset remains unchanged. This step ensures clean data for further processing, such as feature scaling or model training.

3-

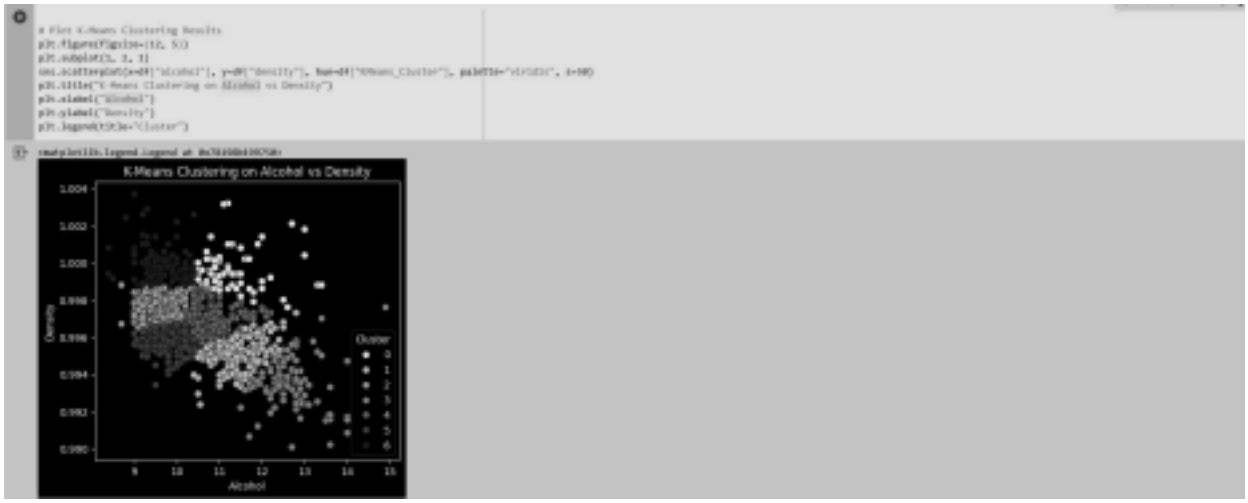
This prepares data for clustering by selecting two numerical features, **"alcohol"** and **"density"**, from the dataset. These features are stored in a new DataFrame data to ensure relevant attributes are used. Next, **StandardScaler()** is applied to normalize the data, improving clustering performance by standardizing feature values to have a mean of **zero** and a standard deviation of **one**. The transformed data, stored in data_scaled, ensures that both features contribute equally during clustering. This step is crucial for distance-based clustering methods like K-Means.

4-

```
[ ] # Apply K-Means clustering
    kmeans = KMeans(n_clusters=7, random_state=42, n_init=10)
    df["KMeans_Cluster"] = kmeans.fit_predict(data_scaled)

[ ] # Apply DBSCAN clustering
    dbscan = DBSCAN(eps=0.5, min_samples=7)
    df["DBSCAN_Cluster"] = dbscan.fit_predict(data_scaled)
```

This applies two clustering algorithms, **K-Means** and **DBSCAN**, to the scaled dataset. First, **K-Means** is used with **7 clusters**, a fixed random state for reproducibility, and **10 initializations** to ensure better clustering. The predicted cluster labels are stored in the "KMeans_Cluster" column of the DataFrame. Next, **DBSCAN** is applied with an **epsilon (eps) of 0.5** and a minimum of **7 samples per cluster**, identifying dense regions in the data. The DBSCAN cluster labels are stored in the "DBSCAN_Cluster" column, allowing comparison between the two methods.

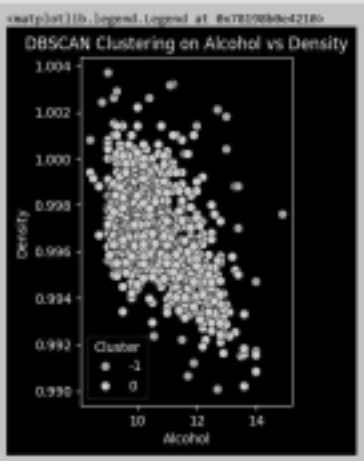
5-



The image shows a scatter plot depicting the results of K-Means clustering applied to a dataset with "Alcohol" on the x-axis and "Density" on the y-axis. Each point represents a data sample, color-coded according to its assigned cluster using the "viridis" color palette. The clustering algorithm has identified seven distinct groups (clusters labeled 0-6), as indicated in the legend. The distribution of points suggests that samples with similar Alcohol and Density values are grouped together, forming clear patterns. The plot provides insights into how these two features influence clustering, revealing regions of high and low density for different alcohol levels.

6-

```
# Plot DBSCAN Clustering Results
plt.subplot(1, 2, 2)
sns.scatterplot(x=df["alcohol"], y=df["density"], hue=df["DBSCAN_Cluster"], palette="coolwarm", s=60)
plt.title("DBSCAN Clustering on Alcohol vs Density")
plt.xlabel("Alcohol")
plt.ylabel("Density")
plt.legend(title="Cluster")
```



This applies DBSCAN clustering to a dataset and visualizes the results in a scatter plot, where "Alcohol" is on the x-axis and "Density" is on the y-axis. The points are colored based on their cluster assignments, using the "coolwarm" palette. In the resulting plot, most points belong to Cluster 0 (red), while some outliers (noise points) are assigned to Cluster -1 (blue). Unlike K-Means, DBSCAN effectively identifies dense regions and marks sparse points as noise. The visualization highlights how DBSCAN clusters high-density regions while leaving low-density points unclassified. This helps in detecting outliers and meaningful patterns in the data.

7-

```
# Show plots
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

```
# Display cluster counts
print("\nK-Means Cluster Counts:")
print(df["KMeans_Cluster"].value_counts())
```

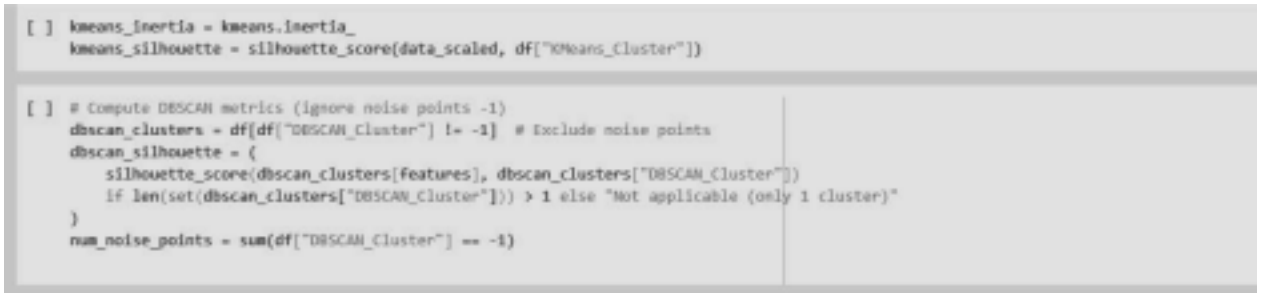
```
K-Means Cluster Counts:
KMeans_Cluster
2    437
5    301
1    269
4    242
6    147
3    123
0     80
Name: count, dtype: int64
```

```
print("\nDBSCAN Cluster Counts:")
print(df["DBSCAN_Cluster"].value_counts())
```

```
DBSCAN Cluster Counts:
DBSCAN_Cluster
 0    1578
-1      29
Name: count, dtype: int64
```

This shows the results of two clustering algorithms applied to a dataset: K-Means and DBSCAN. The K-Means algorithm divided the data into 7 clusters (labeled 0 to 6), with cluster 2 having the most points (457) and cluster 0 the fewest (80). In contrast, DBSCAN identified only two groups: cluster 0 with 1570 points and cluster -1 with 29 points, where -1 represents noise or outliers. This suggests K-Means created more granular groupings, while DBSCAN categorized most data into a single cluster, flagging a small portion as anomalies. The tight_layout() and show() commands indicate the results were visualized, though the plot itself is not displayed here. The output highlights the differing behaviors of the algorithms in handling the dataset.

8-



```
[ ] kmeans_inertia = kmeans.inertia_
    kmeans_silhouette = silhouette_score(data_scaled, df["KMeans_Cluster"])

[ ] # Compute DBSCAN metrics (ignore noise points -1)
    dbscan_clusters = df[df["DBSCAN_Cluster"] != -1]  # Exclude noise points
    dbscan_silhouette = (
        silhouette_score(dbscan_clusters[features], dbscan_clusters["DBSCAN_Cluster"])
        if len(set(dbscan_clusters["DBSCAN_Cluster"])) > 1 else "Not applicable (only 1 cluster)"
    )
    num_noise_points = sum(df["DBSCAN_Cluster"] == -1)
```

This is for evaluating the performance of K-Means and DBSCAN clustering algorithms. For K-Means, the inertia(sum of squared distances of samples to their nearest cluster center) and silhouette score (measuring cluster cohesion and separation) are computed directly using the scaled data and cluster labels. For DBSCAN, noise points (labeled `-1`) are excluded before calculating the silhouette score, which is only computed if there are at least two clusters; otherwise, it returns "Not applicable." The code also counts the number of noise points identified by DBSCAN. The differences in evaluation methods highlight K-Means' reliance on predefined clusters and DBSCAN's noise-handling capability, with metrics tailored to each algorithm's unique characteristics. A syntax error (missing parenthesis) is also visible in the DBSCAN silhouette score calculation.

9-

```
[ ]  # Print cluster evaluation metrics
     print("\n🔹 **Cluster Evaluation Metrics** 🔹")
     print(f"🔹 K-Means Inertia (WCSS): {kmeans_inertia:.2f}")
     print(f"🔹 K-Means Silhouette Score: {kmeans_silhouette:.4f}")
     print(f"🔹 DBSCAN Silhouette Score: {dbscan_silhouette}")
     print(f"🔹 DBSCAN Noise Points: {num_noise_points}")
```

```
🔹 **Cluster Evaluation Metrics** 🔹
🔹 K-Means Inertia (WCSS): 532.73
🔹 K-Means Silhouette Score: 0.3575
🔹 DBSCAN Silhouette Score: Not applicable (only 1 cluster)
🔹 DBSCAN Noise Points: 29
```

This displays clustering evaluation metrics for K-Means and DBSCAN. K-Means has an inertia (sum of squared distances) of **532.73** and a silhouette score of **0.3575**, indicating moderate cluster separation. DBSCAN's silhouette score is **not applicable** because it formed only one cluster (excluding noise), with **29 points** labeled as noise. The results show K-Means performed structured clustering, while DBSCAN treated most data as a single group with outliers. The metrics highlight the algorithms' differing behaviors on the dataset.

Conclusion

K-Means is an efficient clustering algorithm that assigns data points into a predefined number of clusters based on centroid minimization. It performs well with well-separated and spherical clusters but struggles with arbitrary shapes.. DBSCAN, on the other hand, is density-based and can identify noise points, making it more robust for complex cluster structures. However, DBSCAN is sensitive to parameter tuning (**eps, min_samples**) and may label some points as noise instead of assigning them to clusters. In this experiment, K-Means produced balanced clusters, while DBSCAN identified noise points but captured non-linear structures better. The choice between these algorithms depends on the dataset characteristics and the need for predefined clusters versus flexible, shape-adaptive clustering.

Experiment 8

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

Theory:

Dataset Description

- The dataset used is the **MovieLens 100K dataset**.
- It contains user ratings for different movies, in the form of:
    - userID
    - itemID (movie)
    - rating
    - timestamp (ignored in this experiment)

Steps:

1. Importing libraries

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

2. Load the dataset taken from MovieLens and we will then preprocess this data by dropping the columns we don't need

```
df = pd.read_csv("ml-100k/u.data", sep="\t", names=["user_id", "item_id", "rating", "timestamp"])
df.drop(columns="timestamp", inplace=True)
```

3. This line of code is creating a **user-item matrix**, which is a common structure used in **collaborative filtering** for recommendation systems.

```
[ ]  user_item_matrix = df.pivot(index='user_id', columns='item_id', values='rating').fillna(0)
```

4. Now that the data is in the required format, We need to apply k-mean clustering to cluster similar Clustering helps **group similar users** based on their preferences (e.g., movie ratings). Each cluster represents a **type of user behavior** (e.g., "Action Lovers", "Rom-Com Fans", etc.).

   You can then recommend **popular or high-rated items within that user's cluster**, even if the user is new.

```
inertia_vals = []
k_values = range(1, 11)

for k in k_values:
    model = KMeans(n_clusters=k, random_state=42)
    model.fit(user_item_matrix)
    inertia_vals.append(model.inertia_)

plt.figure(figsize=(8,5))
plt.plot(k_values, inertia_vals, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()
```

The elbow method helps find an accurate value for k. In our case, the optimal value for k is 5, we cluster the user_item matrix

```
kmeans = KMeans(n_clusters=5, random_state=42)
# Convert all column names to strings
user_item_matrix.columns = user_item_matrix.columns.astype(str)
clusters = kmeans.fit_predict(user_item_matrix)
user_item_matrix['cluster'] = clusters
```

5. Now we are going to split the dataset into train and test

```
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

6. This function predict_rating is designed to **predict a user's rating** for a movie (item) using **cluster-based collaborative filtering**.

To predict the **expected rating** that a specific user (user_id) would give to a specific movie (item_id) using:

- **User clustering**
- **Cosine similarity**
- **Weighted average of ratings from similar users in the same cluster**

```python
def predict_rating(user_id, item_id, matrix):
    if item_id not in matrix.columns:
        return 3.0  # neutral rating if item not found

    cluster = matrix.loc[user_id, 'cluster']
    cluster_users = matrix[matrix['cluster'] == cluster].drop(columns='cluster')

    user_ratings = cluster_users.loc[:, item_id]
    if user_ratings.sum() == 0:
        return 3.0  # neutral if no one in cluster rated it

    similarities = cosine_similarity([cluster_users.loc[user_id]], cluster_users)[0]
    weighted_sum = 0
    sim_sum = 0

    for i, other_user in enumerate(cluster_users.index):
        if other_user == user_id:
            continue
        rating = cluster_users.loc[other_user, item_id]
        sim = similarities[i]
        weighted_sum += rating * sim
        sim_sum += sim

    if sim_sum == 0:
        return 3.0  # neutral if no similarity
    return weighted_sum / sim_sum
```

This function:

- Uses clustering to narrow down the pool of similar users
- Uses cosine similarity to compute how similar users are
- Predicts the rating based on a weighted average from similar users
- Uses 3.0 as a neutral fallback in edge cases (e.g., no data)

This function recommends movies to a specific user by leveraging both clustering and collaborative filtering. It begins by identifying the cluster that the target user belongs to—this cluster groups users with similar movie preferences based on their past ratings. Within this cluster, it filters out the movies that the user hasn't rated yet, assuming these are the ones they haven't watched. For each of these unrated movies, the function predicts a rating by analyzing how similar users in the same cluster have rated them, using cosine similarity to weigh each user's input based on how closely they resemble the target user.

```
def recommend_movies_for_user(user_id, matrix, top_n=5):
    cluster_label = matrix.loc[user_id, 'cluster']
    cluster_users = matrix[matrix['cluster'] == cluster_label].drop(columns='cluster')

    unrated_items = cluster_users.columns[cluster_users.loc[user_id] == 0]

    recommendations = []
    for item_id in unrated_items:
        pred = predict_rating(user_id, item_id, matrix)
        recommendations.append((item_id, pred))

    top_recommendations = sorted(recommendations, key=lambda x: x[1], reverse=True)[:top_n]
    return [(id_to_title.get(int(item_id), f"Movie {int(item_id)}"), round(pred, 2)) for item_id, pred in top_recommendations]
```

The predicted ratings are then sorted, and the top ones are selected as recommendations. Finally, these movie IDs are converted into human-readable movie titles using a lookup dictionary, and the function returns a neatly formatted list of the top recommendations with predicted scores. This approach ensures that recommendations are not just based on general popularity but are tailored to the tastes of users who think and rate similarly.

```
recommendations = recommend_movies_for_user(100, user_item_matrix)
for title, score in recommendations:
    print(f"{title} → Predicted Rating: {score}")
```

```
Doom Generation, The (1995) → Predicted Rating: 3.0
Nadja (1994) → Predicted Rating: 3.0
Brother Minister: The Assassination of Malcolm X (1994) → Predicted Rating: 3.0
Carlito's Way (1993) → Predicted Rating: 3.0
Robert A. Heinlein's The Puppet Masters (1994) → Predicted Rating: 3.0
```

The movies are then recommended for random users from user_item matrix

Experiment No: 9

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

Theory:

1. What is Apache Spark and how it works?

Apache Spark is an open-source, distributed computing system designed for fast and large-scale data processing. It provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

Key Features:

- **In-memory computing:** Speeds up processing by storing intermediate results in memory.
- **Distributed processing:** Executes across multiple nodes in a cluster.
- **Ease of use:** Supports APIs in Python (PySpark), Scala, Java, and R.
- **Rich ecosystem:** Includes Spark SQL, MLlib (machine learning), GraphX (graph processing), and Spark Streaming.

How It Works:

- **Spark Application:** Comprises a driver program and a set of distributed workers (executors).
- **Driver Program:** Controls the execution, maintains SparkContext, and coordinates tasks.
- **Cluster Manager:** Allocates resources (e.g., YARN, Mesos, Kubernetes).
- **Executors:** Run tasks and store data for the application.
- **RDD (Resilient Distributed Dataset):** Core abstraction that represents a fault-tolerant collection of data that can be operated on in parallel.

2. How is data exploration done in Apache Spark?

Exploratory Data Analysis (EDA) in Apache Spark is typically performed using **PySpark**, the Python API for Spark. It enables users to analyze large datasets using distributed dataframes and SQL-like operations.

Steps for EDA in Apache Spark:

1. **Initialize Spark Session:**
    - Set up the Spark environment using SparkSession.
2. **Load the Dataset:**
    - Use functions like read.csv() to load data into a DataFrame.
3. **Understand the Data:**
    - Inspect schema, data types, and preview rows using .printSchema() and .show().
4. **Summary Statistics:**
    - Generate descriptive statistics with .describe().
5. **Data Cleaning:**
    - Handle missing values using .na.drop(), .fillna(), or filtering nulls.
6. **Data Transformation:**
    - Create new columns, filter data, and apply transformations using DataFrame APIs.
7. **Aggregation and Grouping:**
    - Perform group-wise computations using .groupBy() and aggregation functions.
8. **Visualization (with Pandas or External Tools):**
    - Convert Spark DataFrame to Pandas for plotting if needed.

This process allows scalable, efficient EDA for large datasets that cannot fit into memory, unlike traditional tools like Pandas.

Conclusion:

Apache Spark is a powerful and efficient framework for processing large-scale data due to its distributed computing and in-memory capabilities. It enables fast, scalable, and interactive analysis, making it ideal for performing **Exploratory Data Analysis (EDA)** on big datasets.

Through PySpark, users can load, inspect, clean, transform, and analyze data using DataFrame operations similar to Pandas, but with the ability to handle much larger datasets. The step-by-step EDA process in Spark provides deep insights into the data, which is crucial for informed decision-making and further machine learning tasks.

Combining Spark with tools like Pandas for visualization can enhance the EDA experience, bridging the gap between scalability and interpretability.

Experiment No: 10

Aim:

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

Theory:

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

Streaming refers to the continuous flow of data that is processed in real-time or near real-time. It involves analyzing data as it arrives, allowing immediate insights and reactions.

Batch Data:

- Batch data processing deals with **large volumes of static data** that are collected over a period and then processed together.
- It is suitable for applications where **immediate results are not required**.
- Example: Processing daily sales data at the end of the day.

Stream Data:

- Stream data processing handles **continuous, unbounded data** arriving in real-time or micro-batches.
- It is ideal for applications needing **real-time analytics**, like fraud detection, live dashboards, etc.
- Example: Processing logs from a web server or transactions from a banking system as they occur.

2. How Data Streaming Takes Place Using Apache Spark?

Apache Spark Streaming is an extension of the Spark Core API that enables scalable, high-throughput, fault-tolerant processing of live data streams.

How It Works:

- Spark Streaming ingests data in **mini-batches** from various sources such as Kafka, Flume, HDFS, or socket connections.

- Each mini-batch is treated as an **RDD (Resilient Distributed Dataset)** and processed using Spark's core operations.

- Once processed, the results can be stored in databases, file systems, or dashboards.

Key Components:

- **DStreams (Discretized Streams):** The basic abstraction in Spark Streaming. Internally, a DStream is a sequence of RDDs.

- **Data Sources:** Real-time data can come from Kafka, socket, files, etc.

- **Window Operations:** Perform computations over sliding windows of data (e.g., last 10 minutes).

- **Transformations:** Just like batch RDDs, DStreams can use map, filter, reduce, etc.

Use Cases:

- Real-time fraud detection.

- Social media sentiment analysis.

- Log processing.

- Monitoring systems and alerts.

Conclusion:

Apache Spark provides powerful capabilities for both **batch and stream data processing**, making it a unified framework suitable for a wide range of big data applications. While **batch processing** is ideal for historical data analysis and scheduled jobs, **streaming** is essential for real-time insights and event-driven applications. Spark Streaming bridges the gap between these two paradigms by providing a consistent and scalable platform for analyzing both static and live data, enabling organizations to react to information as it happens.

Dimple Paliwal D15C 8 04/05

AIDS Assignment

Q1) What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive & renovated our way of life with different application?

→ Artificial Intelligence (AI) refers to the ability of machines & system to simulate human intelligence, including tasks like learning, reasoning, problem-solving & decision making. It leverages techniques like machine learning & NLP to analyse data, identify patterns & improve performance over time.

Role of AI during the Pandemic:

- Healthcare - AI was used to quickly analyze medical imaging eg. CT scans & predicting severity.
- Remote work & Education - The pandemic drove widespread adoption of AI in virtual learning & remote work tools.
- Retail - Retail stores are using computerized models to map out their stores & track inventory. This is in a response to need given the rush buy specific items at various stages of pandemic.

Q2) What are AI agents terminology, explain with examples.

→ AI agents are autonomous system that perceive the environment, make decisions,

AI Agent Terminology:

1) Agent - An AI system that interacts with the environment & takes action.
Example - A self-driving car that perceives traffic signals & adjusts speed accordingly.

2) Environment - The surrounding in which the AI agent operated.
Ex - For a chess-playing AI, the chessboard & opponent are part of its environment.

3) Percepts - The information received by agent via sensors
Ex - A robot vaccum detecting obstacles using cameras & infrared ray.

4) Actions - The move an agent makes in response to percept.
Ex. A chatbot responding to user's past ~~preference~~ query.

5) Percept Sequence - The entire history of percept received by an agent Ex. A recommendation system tracking user's past preference.

6) Performance Measure - Determine success of an agent -
Ex. A self driving car's performance measure will be reaching the destination safely & in a timely manner

Q:) How AI techiques is used to solve 8 puzzle problem?

→ Initial State: 1 2 3 Goal State: 1 2 3
4 0 6 4 5 6
1 5 8 7 8 0

Misplaced tiles : h(n)=2

steps to solve 8 puzzle problem by A*:

1) Initialize a priority queue.
2) Insert the initial state with f(n) = g(n) + h(n)
3) While queue not empty:
   Remove state with lowest f(n)
   If state = goal, return solution
   Generate valid moves (up, down, left, right)
   Compute g(n) & h(n) for new states
   Insert new state into queue
4) Repeat until goal is reached

Example Execution

Step 1	Step 2	Step 3: (Goal)
1 2 3	1 2 3	1 2 3
4 0 6	4 5 6	4 5 6
7 5 8	7 0 8	7 8 0
(h=2)	(h=1)	(h=0)

Q4) What is PEAS description? Give PEAS description for following.

→ PEAS stands for Performance measure, Environment, Actuators & sensors.

P→ Criteria to evaluate the agents success.
E→ Surrounding where agent operates.
A→ Component that allows agent to take actions.
S→ component that help agent percept the environment

1) Taxi Driver
P → Reaching destination, fuel efficiency
E → Roads, traffic
A → Steering, wheel, accelerator, brakes
S → Camera, GPS, speedometer
2) Medical Diagnosis System
P → Accuracy of diagnosis, patient satisfaction
E → Medical records, test result symptoms
A → Prescription generation, Report generation
S → Patient data input, lab test results.
3) Music Composer
P → Safety of Composition, User satisfaction
E → Musical genres, musical theory constraints
A → Music synthesizer, Instrument, generating note
S → Composition structure, user feedback
4) Aircraft Autolander
P → Smooth & safe landing
E → Weather, runway condition, air traffic
A → Flaps, landing gear, throttle.
S → Altitude sensor, GPS, wind direction sensor.
5) Essay Evaluator
P → Accurate grading, feedback quality.
E → Grammar rules, submitted essays.
A → displaying grades & feedback
S → Text inputs, spelling & grammar checkers
6) Robotic Sentry Gun for lock labs
P → correctly indentifying threats, accuracy.
E → authorized personnel
A → Camera movement, alarm system
S → motion sensor, facial recognition

Q5) Categorize a shopping bot for an offline bookstore according to each of the six dimensions.

⇒ 1) Partially Observable
- but may not have full visibility of store's inventory
2) Stochastic - outcomes uncertain due to customer behaviour, stock availability.
3) Sequential - Bot will suggest books based on previous customer queries.
4) Dynamic - Bookstore environment changes like books getting sold, new stock
5) Discrete - Bots process finite number of actions.
6) Multi agent - Bot interacts with customers & store employee, each have their own goals.

Q6) Differentiate between Model-Base & Utility Based Agents.

⇒

Model-Based agents	Utility-based agents
1) Uses an internal model of environments to make decisions.	1) Choose actions based on utility function that measure performance.
2) Decision based on past & present percepts	2) Selects action based on maximum utility.
3) Can be good based but doesn't necessary optimize for best outcomes	3) Optimizes for higher possible utility ensuring better performance
4) Ex- Robot vaccum using a map to Navigate	4) Ex- Self-Driving car choosing safest & fastest route

7) Explain the architecture of a knowledge based agent & learning agent.

⇒ Knowledge-based agent

A knowledge-based agent uses stored knowledge to make decisions. It consists of

- Knowledge base → Stores facts, rules & logic
- Inference Engine → uses reasoning to derive conclusions.
- Perception (sensors) → Gathers new information from environment
- Action Mechanism → Performs appropriate actions based on reasoning

Ex: AI in medical diagnosis uses past cases & symptoms to diagnose diseases.

Learning agent

A learning agent improves its performance over time. It consists of

- Learning Element → update knowledge based on experience
- Performance Element → Decides actions based on current knowledge
- Critic → Provides feedback by evaluating actions.
- Problem Generator → suggests new actions to improve learning

Ex: A self-driving car learns from traffic patterns & adjusts driving behaviours.

8) Convert the following to predicates:

⇒ a). Anita travels by car if available otherwise travels by bus.

Car Available → Travels by car (Anita)

~~Car Available → Travels by bus (Anita)~~

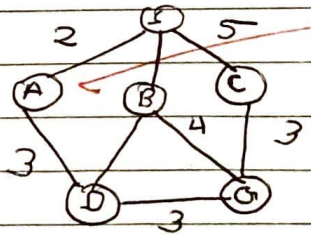
Car Available ⇒ Travels by Bus (Anita)

⇒ b) Bus goes via Andheri & Goregaon.
Goes Via (Bus, Andheri) ∧ Goes Via (Bus, Goregaon)

c) Car has punctures so is not available.
Puncture (car)
Puncture (car) → Car Available

Will Anita travel via Goregaon? Use forward reasoning from (c)
Puncture (car) is true
As Puncture (car) ⇒ ¬ Car Available
From (a)
¬ Car Available, we use ¬ Car Available ⇒ Travels By Bus (Anita)
From (b)
Goes Via (Bus, Goregaon)
Since Anita travels by bus, she follows this route
Thus, Anita will travel via goregaon.

Q9) Find route from S to G using BFS

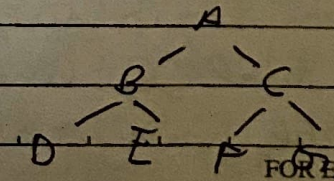


⇒ To find route from S to G using BFS, we systematically explore all node level by level starting from source node (S) until we reach destination node (G)

1) Start at S
   Queue : [S]
2) From S, we go to it neighbours: A, B, C
   Queue : [A, B, C]
3) Dequeue A & explore its neighbours
   Queue : [B, C, D]
4) Dequeue B & queue neighbours
   Queue [C, D, G]
5) Dequeue C & queue neighbours
   Queue : [D, G]
6) Dequeue D
   Queue -[G]
7) Dequeue G
   As G is our destination, BFS stops here.
   Route from S to G: S → B → G.

9) What do you mean by depth limited search? Explain Iterative Deepening Search with Example

→ Depth limited search (DLS) is an uninformed search algorithms that modify DFS by introducing a depth limit L, preventing exploration beyond the predefined limit. This prevents infinite loops in infinite graphs but risks missing goals beyond L.
Iterative Deepening Search (IDS) combines DLS with BFS by incrementally increasing the Depth limit.
Example:



Goal = G

Interation 1: Depth Limit = 0

Node Visited = A

Result = Goal Not Found

Iteration 2: L = 1

Node Visited: A → B → C

Goal Not Found

Iteration 3: L = 2

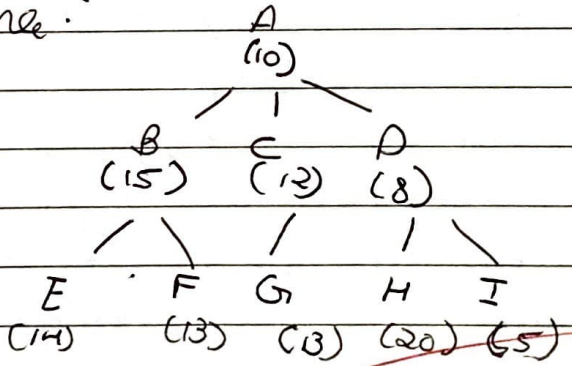
Node Visited: A → B → D → E → C → F → G

Goal is found at L = 2.

Q12) Explain Hill Climbing & its Drawbacks in detail with example. Also state limitations of steepest-ascent.

⇒ Hill Climbing is a local search optimization algorithm, which moves toward better neighbouring solutions until it reaches a peak.

Example:



Steps:

Start at a root node A(10)

Compare its children B, C & D

Move to child with highest value i.e B(15)

Repeats for B's children E & F.

Terminates at A(14)

The algorithm stops at E(14) not reaching the goal.

Drawbacks:

1) Local maxima - The algorithm greedily selects the best immediate child & can thus get stuck on local maxima
2) Plateau - If siblings have equal algorithm, the algorithm cant decide the next step.
3) Rigid - Narrower uphills paths require backtracking hill climbing does not support.

Limitation of steepest-Ascent Hill Climbing:

1) Computationally Expensive - Evaluates all neighbours before selecting the best.
2) Can get stuck - It can get stuck in local maxima plateaus or ridges.

Q13) Explain Simulated and & write its algorithm.

⇒ Simulated Annealing (SA) is a probabilistic optimization algorithm inspired by metallurgical process of annealing where materials are heated & cooled to reduce defects. It escapes the local optimal by temporarily accepting worse solution with a probability.

Algorithm:

1) Initialize - Set a initial solution & define an initial temp T.
2) Repeat until stopping conditions
   - Generate a new neighbour solution
   - Compute change in cost T. ($\Delta E = E_{new} - E_{current}$)
   - If new solution is better i.e $\Delta E > 0$, accept it.
   - If worse, accept it with probability $P = e^{-\Delta E/T}$
   - Decrease temperature T (cooling schedule)

3 Return best solution.

Example:

Travelling salesman Problem

Swap two cities in a route accept a longer route easily (high T) but rejects it later (low T).

Q14) Explain A* algorithm with one example.

⇒ A* is a best first search algorithm used in path find & graph traversal. It used the following formula:

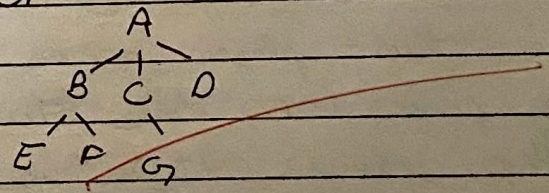
$$f(n) = g(n) + h(n)$$

$g(n)$ = Cost to search node n from start

$h(n)$ = heuristic estimate of cost to search from goal to n

$f(n)$ = total estimated cost.

Example: Find: G



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

Steps:

1) Start at root node A

$$f(A) = g(A) + h(A) = 0 + 6 = 6$$

2 Expand neighbours : B, C, D

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

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

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

3 Choose lowest value that is C ($f(C) = 4$)

4 Expand neighbours of C: G

$f(G) = 2+4+0 = 6$

5 Goal reached at G with total cost 6

Advantages →

- Efficient for finding shortest paths in weighted graphs
- Balances exploration by considering both $g(n)$ & $h(n)$.

Q15) Explain Min-Max Algorithm & draw game tree for Tic Tac Toe game.

⇒ The Min Max Algorithm is a decision making algorithm used in two-player games. It assumes:

- One player (MAX) tries to maximize the score.
- Other player (MIN) tries to minimize the score.
- Game tree represents all possible moves.

Algorithm

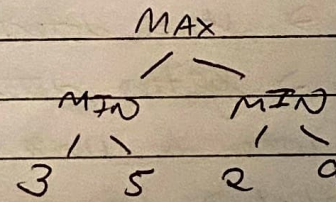
1) Generate game tree
   ⇒ All possible moves from current state.
2) Assign scores
   Terminal state (Win/Loss/Draw), Set +1, -1, 0
3) Max picks higher value from children
   MIN picks lowest value.
4) Repeat until root node is evaluated.

Q16) Explain Alpha beta pruning algorithm for adversarial search with example.

⇒ Alpha-beta pruning is a optimization technique for Min-Max algorithm used in Adversial Search. It reduces the number of nodes evaluated in a game tree by eliminating branches that won't influence the final decision.

- Alpa (α): Best value for max so far.
- Beta (β): Best value for min so far.
- Pruning condition: if α ≥ β, stop exploring that branch

Example:



- Without pruning: All nodes are evaluated
- With pruning: Once Min finds 2, 9 is ignored, saving computation.

Q17) Explain Wumpus world environment giving its PEAS discription. Explain how percept sequence is generated

⇒ A grid-based game where an agent navigates a cave to find gold while avoiding pits & a Wumpus.

PEAS Description:

- P (Performance Measure): Gold found (+), falls into a pit / Wumpus (-), steps taken (-).

• E (Environment): A 4x4 grid, containing gold, pits, Wumpus & agent.
• A (Actuators): Move (left, right, up, down), grab gold, shoot arrow, climb out.
• S (Sensors):
. Breeze: Near a pit
. Stench: Near Wumpus.
. Glitter: Gold present
. Bump: Hits a wall
Percept Sequence Generation:
The agent senses the environment based on its position:
. If near a pit ⇒ Breeze detected.
. If near a Wumpus ⇒ Stench detected.
. If gold is present ⇒ Glitter detected.
. If agent moves into a wall ⇒ Bump sensed.
. If Wumpus is shot ⇒ Scream heard.

Q) Solve the following crypto-arithmetic problem:
SEND + MORE = MONEY
⇒ Step 1: Identify M
. Since the sum produce a five-digit result the addends have at most 4 digits, M=1
Step 2: Consider Carrying
The S+M sum must create the first digit M=1, meaning S=9
Step 3: Assigning other Values
Through logical & deduction techniques
. S=9, E=5, N=6, D=7, M=1, O=0, R=8, & Y=2.

Q(a) Consider the following axioms:
All people who are graduations 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.

⇒ Step 1: Represent in FOPL
Let:
G(x): x is graduating
H(x): x is happy
S(x): x is smiling
Step 2: Convert to clause form
1) $G(x) \rightarrow H(x) = \neg G(x) \vee H(x)$
2) $H(x) \rightarrow S(x) = \neg H(x) \vee S(x)$
3) $\exists x\, G(x)$ remain as $G(A)$
Thus, the clause set is:
1) $\neg G(x) \vee H(x)$
2) $\neg H(x) \vee S(x)$
3) $G(A)$
Step 3: Prove "Is someone smiling ?" using Resolution
1) $G(A)$
2) $\neg G(A) \vee H(A)$
· Resolution with (1): $H(A)$
3) $\neg H(A) \vee S(A)$.
· Resolution with (2): $S(A)$

Since we derived S(A), our assumption ¬S(x) is false
Thus, someone is smiling.

Q20) Explain Modus Ponens with suitable example
⇒ Modus Ponens is a fundamental rule of inference in logic that states:
If P→Q (If P then Q) is true, & P is true, then Q must be true.
Logical Form:
1) P → Q
2) P (P is true)
∴ Q (Therefore, Q is true)
Example:
1) If it rains, the ground will be wet. (R→W)
2) It is raining.
∴ The ground is wet.

Q21) Explain Forward & backward chaining with the help of example.
⇒ Forward chaining:
Starts from known facts & applies rules to reach a conclusion
Ex: Given Rules:
1) If it rains, the ground wet. (R→W)
2) If the ground is wet, it is slippery. (W→S)
3) It is raining (R)

Process:
- From R, infer W
- From W, infer S

Conclusion: The ground is slippery.

Backward Chaining:
Starts from a goal & works backward to find supporting facts.
Ex: Process:
- To prove S, check W → S → Need to prove W.
- To prove W, check R → W → Need to prove R.
- If R is true, then W is true ⇒ S is true.

DIMPLE DALWANI

D15C 8

AIDS-I 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)

Ans:

Sort the Data

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

1. Mean

Mean is calculated using the formula:

Mean = Sum of all values ÷ Number of values
Mean = 1621 ÷ 20 = **81.05**

2. Median

Since there are 20 values (even number),
the median is the average of the 10th and 11th values.

10th value = 81
11th value = 82

Median = (81 + 82) ÷ 2 = **81.5**

3. Mode

- 76 appears 3 times
- 82 and 90 appear 2 times

- All other values appear only once

Mode = 76

4. Interquartile Range (IQR)

Q1 (Lower Quartile): Median of the first 10 values
First 10 values: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
Q1 = (76 + 76) ÷ 2 = **76**

Q3 (Upper Quartile): Median of the last 10 values
Last 10 values: 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Q3 = (88 + 90) ÷ 2 = **89**

Interquartile Range = Q3 - Q1 = 89 - 76 = **13**

Q.2 1) Machine Learning for Kids 2) Teachable Machine

- 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

Ans:

1. Machine Learning for Kids

Target Audience:

- School-aged children (typically ages 8–16)
- Teachers and educators introducing AI and ML concepts to students
- Beginners with little or no coding experience

Use by Target Audience:

- Kids train ML models by labeling examples (text, images, numbers, or sounds).
- These models are then used in **Scratch**, **Python**, or **App Inventor** projects.
- Teachers use it in classrooms to **teach basic AI/ML concepts through interactive, hands-on learning.**

Benefits:

- **Kid-friendly interface** that simplifies complex ML tasks.
- **Integrates with Scratch and Python**, allowing creative applications.

- Encourages **experiential learning** with real ML models.
- **Free to use** with cloud-based options.

Drawbacks:

- **Limited to educational use**; not suitable for advanced ML tasks.
- **Not robust for complex or large datasets.**
- **Requires internet access** and sometimes a bit of setup (especially for Python/App Inventor).

2. Teachable Machine

Target Audience:

- Students, hobbyists, educators, and beginner-level developers
- Creatives and designers exploring ML without coding

Use by Target Audience:

- Users create ML models by training them with **images, audio, or poses**.
- The trained models can be **exported** for use in websites, apps, or other creative projects.
- Used in classrooms, workshops, or personal experimentation to demonstrate ML in action.

Benefits:

- **No coding required** — fully visual, drag-and-drop interface.
- Quick, **real-time model training and feedback**.
- Supports **exporting models to TensorFlow.js, TensorFlow Lite**, and more.
- Great for **creative applications**, demos, and education.

Drawbacks:

- **Limited model types** (mostly classification).
- Not suitable for complex or large-scale ML applications.
- **Lacks deeper customization** or fine-tuning options.

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

    - Predictive analytic
    - Descriptive analytic

    Ans:Both **Machine Learning for Kids** and **Teachable Machine** are best described as **predictive analytic** tools. This is because they are designed to train machine learning models that can make predictions or classifications based on input data. In Machine Learning for Kids, users—primarily students—label training data such as images, text, or sounds, and the tool creates models that can predict the correct label for new, unseen inputs. Similarly, Teachable Machine allows users to train models using images, audio, or poses, and these models can then predict or classify future inputs in real time. These tools are not focused on analyzing or summarizing past data, which is characteristic of descriptive analytics. Instead, their main function is to use labeled data to learn patterns and apply those patterns to make predictions, which is the core purpose of predictive analytics.

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

    - Supervised learning
    - Unsupervised learning
    - Reinforcement learning

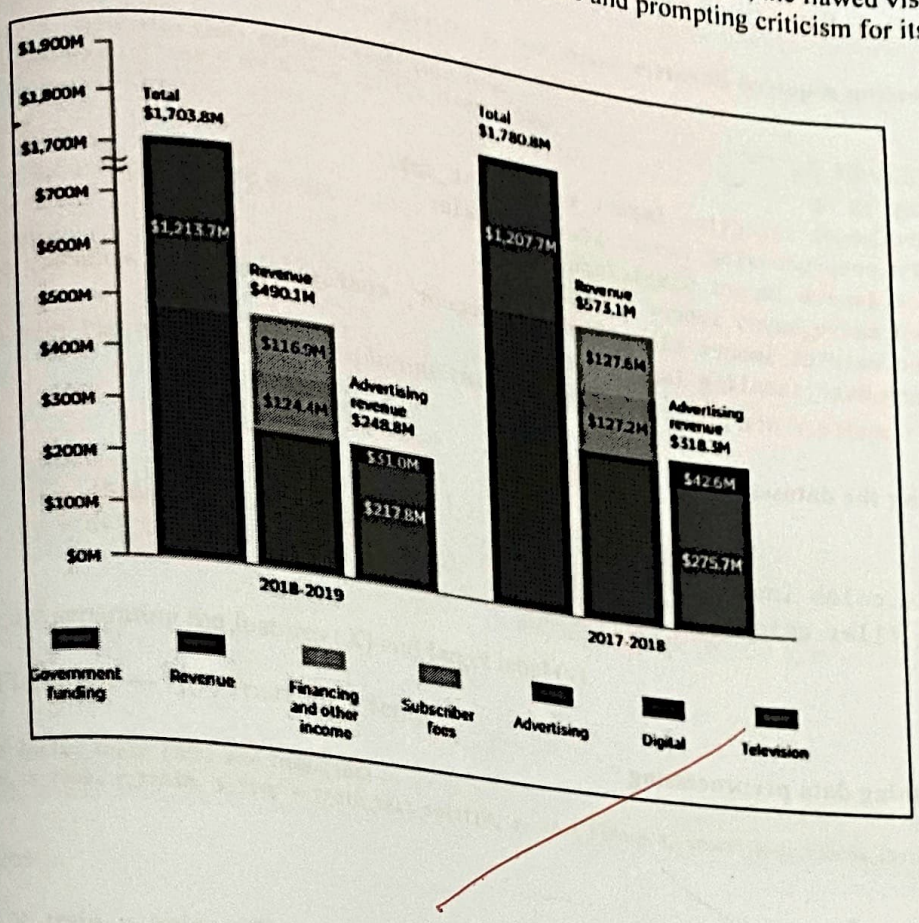
    Ans:Both **Machine Learning for Kids** and **Teachable Machine** use **supervised learning**. This means users give the tool examples that are already labeled—for example, pictures labeled as "cat" or "dog," or sounds labeled as "clap" or "whistle." The tool learns from these examples and then tries to guess the correct label for new data. In both tools, the user teaches the model by showing it what each example means. They don't find patterns on their own like in unsupervised learning, and they don't learn by trying things and getting rewards like in reinforcement learning. That's why **supervised learning** is the best way to describe how both tools work.

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.

    Ans:In April 2023, a bar chart from the Canadian Broadcasting Corporation's (CBC) 2018–2019 Annual Report reappeared online and faced heavy criticism for being misleading. The chart aimed to display CBC's sources of funding, such as government support and advertising income, but its design raised serious concerns about how accurately the data was represented.One major issue was the manipulation of the y-axis, which featured an abrupt jump from $700 million to $1.7 billion. This distorted scale caused the $490 million advertising revenue bar to appear larger than the $1.2 billion bar for government funding, despite the actual values showing otherwise. This visual trick created the false impression that CBC earned more from television advertising than it received from government

support. Furthermore, the chart separated advertising and other revenue into different bars instead of grouping them as parts of the total funding, adding to the confusion. Overall, the flawed visualization misrepresented the true financial picture, misleading viewers and prompting criticism for its lack of clarity and accuracy.



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

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

Requirements to satisfy

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

Pima Indians Diabetes Database

Ans. Model used: Naive Bayes

Step 1. Importing required libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from imblearn.over_sampling import SMOTE
```

Step 2. Loading the dataset

```
from google.colab import files
uploaded = files.upload()
```

Step 3. Performing data preprocessing

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

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

Since here there are no null values we can skip imputation

Step 4. Splitting the dataset

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# First split: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# Second split: Validation (20%) and Test (10%) from Temp
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, stratify=y_temp, random_state=42)
```

In this step we are splitting the dataset into:
70% Train
20% Validation
10% Test

But since train_test_split only lets us split into two parts at a time, we do it in two stages:

Step 1:

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

Here we are separating the features (X) and target label (y).

Step 2: First Split — 70% Train, 30% Temp

```
# First split: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
```

This gives:

- X_train, y_train → 70% of the data (for training)
- X_temp, y_temp → remaining 30% (to be further split into validation and test)

Stratify is used to ensure that the proportion of class labels (0s and 1s) is the same in all splits.

Step 3: Second Split — 20% Validation, 10% Test

```
# Second split: Validation (20%) and Test (10%) from Temp
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, stratify=y_temp, random_state=42)
```

Here we are splitting X_temp (30%) into:

- X_val, y_val → 20% of original data
- X_test, y_test → 10% of original data

Step 5. Using SMOTE for class imbalance

Model is effectively identifying most actual diabetic cases.

Precision for Class 1 (Diabetic):
63% (Validation), 68% (Test)
Around two-thirds of diabetic predictions are correct.

Confusion Matrix indicates:

- High true positive rate for both classes.
- Moderate number of false positives and false negatives.

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

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

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

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

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

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

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

(Refer any one)

Ans:

1. Importing Required Libraries

```
import pandas as pd, numpy as np
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
```

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

We use:

- Pipeline to streamline polynomial features → standardization → ridge regression.
- GridSearchCV for hyperparameter tuning
- r2_score, mean_squared_error for evaluation

2. Defining the RegressionModel Class

```
class RegressionModel:
  def __init__(self, model_pipeline, param_grid):
    ...
```

- Encapsulates model training, evaluation, and hyperparameter tuning.
- Reusable and extensible for other regression problems.

3. Loading the Dataset

```
url =
"https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20dat
a%20set.xlsx"
df = pd.read_excel(url)
```

The dataset contains real estate records including:

- Distance to MRT station
- Number of nearby convenience stores
- Age of building
- Geographic coordinates

4. Data Preprocessing

```
df = df.drop(columns=['No']) # Drop irrelevant index
X = df.drop(columns=['Y house price of unit area']) # Features
y = df['Y house price of unit area'] # Target
```

We define:

- X: all columns except house price
- y: the target variable (house price)

5. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(...)
```

- 70% training data, 30% testing
- random_state=42 ensures reproducibility

6. Model Pipeline & Hyperparameter Grid

```
pipeline = Pipeline([
  ('poly', PolynomialFeatures()),
  ('scaler', StandardScaler()),
  ('ridge', Ridge())
])
```

- Pipeline Components:
  - poly: adds non-linearity (degree=2 or more)
  - scaler: standardizes features (important for ridge!)
  - ridge: regularized regression
- Parameter Grid:

```
param_grid = {
  'poly__degree': [2, 3, 4],
  'ridge__alpha': [0.1, 1, 10, 100]
```

}

We let GridSearchCV try different combinations of:

- Polynomial degrees: 2 to 4
- Ridge regularization strengths (alpha): 0.1 to 100

7. Training and Evaluation

best_model = reg_model.train(X_train, y_train)

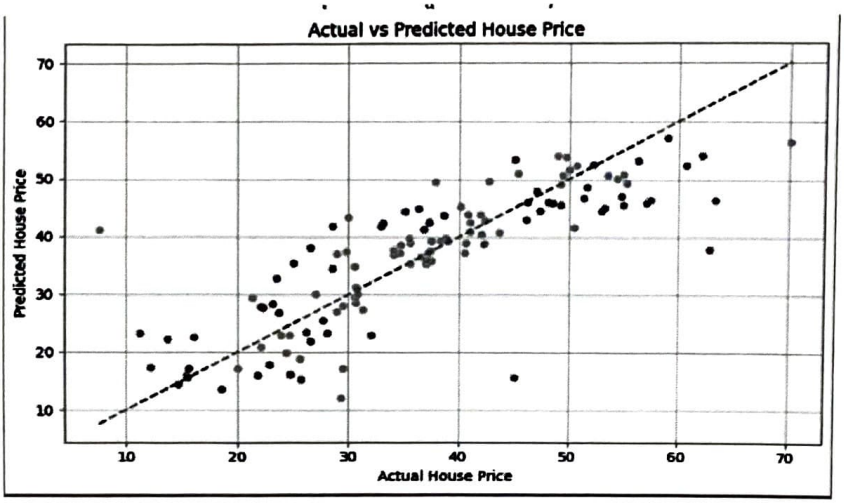
y_test_actual, y_pred = reg_model.evaluate(best_model, X_test, y_test)

- The model is trained with 5-fold cross-validation
- Best estimator is used for prediction
- We calculate:
  - R^2: proportion of variance explained
  - Adjusted R^2: penalizes for extra features
  - MSE: average squared prediction error

8. Visualization

sns.scatterplot(x=y_test_actual, y=y_pred)

- Compares actual vs predicted prices
- Red dashed line shows ideal predictions (perfect match)



Final Results:

Best Params: {'poly__degree': 2, 'ridge__alpha': 1}

R^2 Score: 0.6552

Adjusted R^2 Score: 0.6376

Mean Squared Error: 57.6670

- The model captures ~66% of the variance in house prices.
- There's room for improvement, but this is reasonable for real-world data.
- Adjusted R^2 shows performance after accounting for model complexity.

Why we May Not Reach $R^2 > 0.99$

- Real-world datasets include noise, missing features, and non-linear interactions.
- Polynomial features help but too high a degree → overfitting.
- Ridge helps reduce overfitting, but can't add missing signal.

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).

Ans:The Wine Quality dataset from Kaggle contains physicochemical properties of red and white wine samples, along with a quality score (rated from 0 to 10) assigned by wine tasters. This dataset is widely used for regression and classification problems in machine learning. Here's a breakdown of the key features, their relevance, handling of missing data, and imputation methods:

Key Features and Their Importance:

Feature	Description	Importance in Wine Quality Prediction
Fixed acidity	Non-volatile acids in wine (e.g., tartaric acid)	Affects flavor, crispness, and balance
Volatile acidity	Acetic acid content; high levels lead to vinegar taste	Negatively correlated with quality
Citric acid	Adds freshness and flavor	Mild positive impact on taste
Residual sugar	Sugar left after fermentation	Sweetness; important for white wines
Chlorides	Salt content	Too much can spoil taste
Free sulfur dioxide	Prevents microbial growth	Affects preservation and freshness
Total sulfur dioxide	Sum of all SO_2 forms	Excess can cause unpleasant taste
Density	Related to sugar and alcohol content	Helps distinguish wine types
pH	Acidity level	Affects stability and taste
Sulphates	Adds sulfur dioxide; acts as preservative	Somewhat correlated with better quality
Alcohol	Alcohol content in %	Strong positive correlation with quality
Quality	Target variable (score from 0 to 10)	-

Handling Missing Data During Feature Engineering:

The original Kaggle wine quality datasets (red and white) are generally **clean with no missing values.** However, in real-world scenarios or modified datasets, missing values might appear. During feature engineering, we can handle them using different imputation techniques depending on the nature and amount of missing data.

Imputation Techniques: Pros and Cons:

Method	Advantages	Disadvantages
Mean/Median Imputation	Simple and fast; preserves sample size	Can distort variance and distribution
Mode Imputation	Good for categorical values	Not suitable for continuous data
KNN Imputation	Considers similarity with other samples	Computationally expensive for large datasets
Regression Imputation	Predicts missing values using other features	May introduce bias if assumptions are violated
Multiple Imputation	Accounts for uncertainty; statistically rigorous	More complex and time-consuming
Dropping Rows	Easy if missing data is minimal	Risk of losing valuable data if too many rows removed

Conclusion: Understanding each feature's role is essential for building accurate wine quality prediction models. Most features directly affect taste, aroma, or preservation. When working with real-world or expanded datasets, proper imputation techniques help maintain data integrity and model performance. The choice of imputation should balance simplicity, accuracy, and the amount of missing data.