AI and DS-1

Experiment1

Aim: 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_5255387	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	2023-05-08
1999	TXN_5764993	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	2023-06-11
...	...	...	...	...	...	...	...
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-a9dea45c16d4>: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	2023-04-26

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

```
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: $$\text{Xnormalized} = \frac{\text{X}-\text{Xmin}}{Xmax-Xmin}$$

```
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')
```

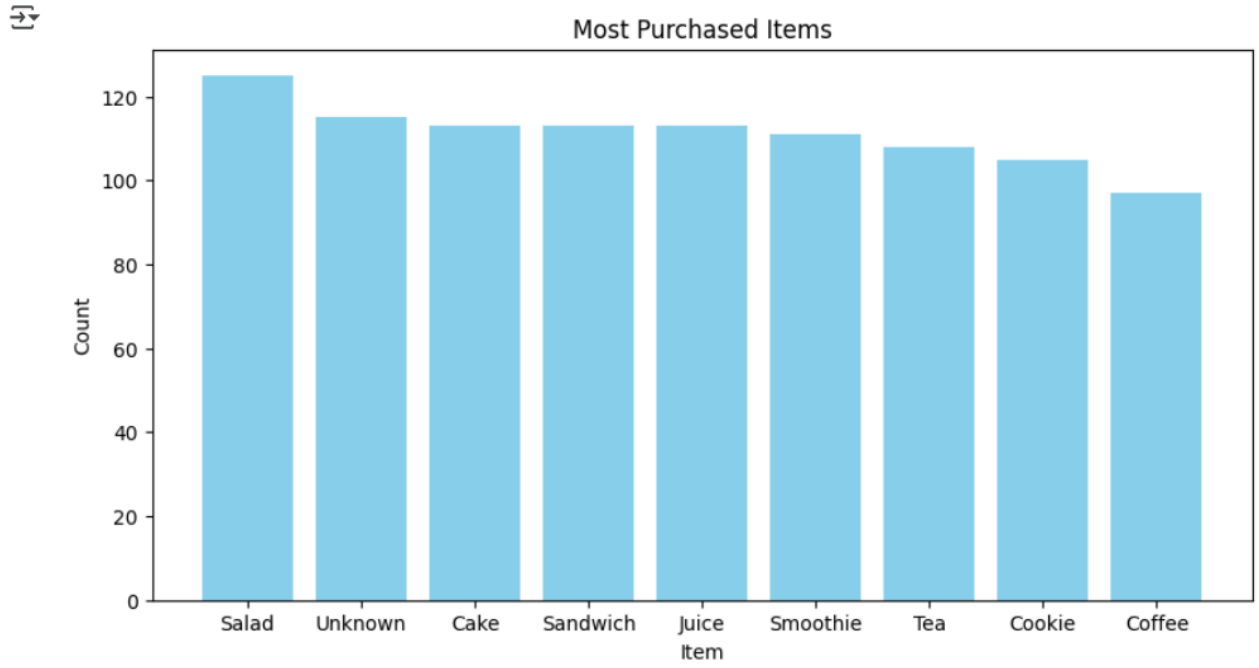
```
Mounted at /content/drive
```

```
[ ]  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

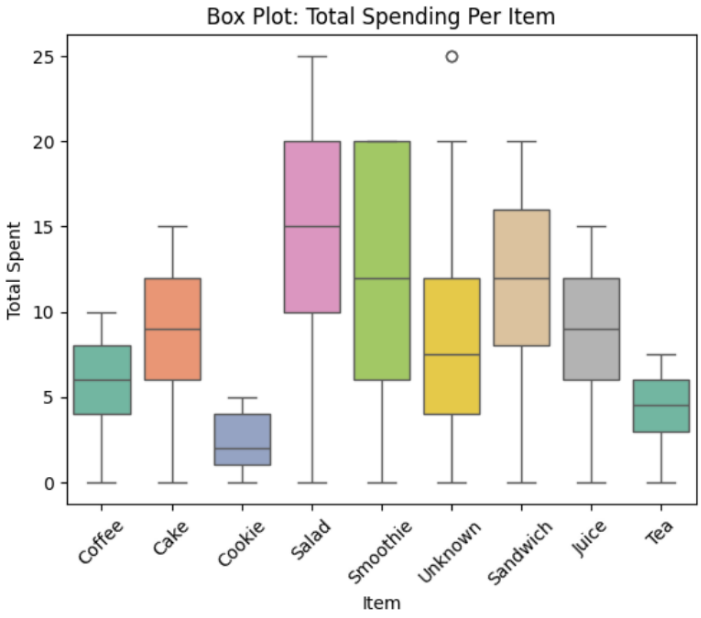
A contingency table in Python can be created using **pandas**. The code typically starts by importing pandas and loading the dataset. The pd.crosstab() function is then used to generate the table, where one categorical variable (e.g., "Item") is placed as rows and another (e.g., "Payment Method") as 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.

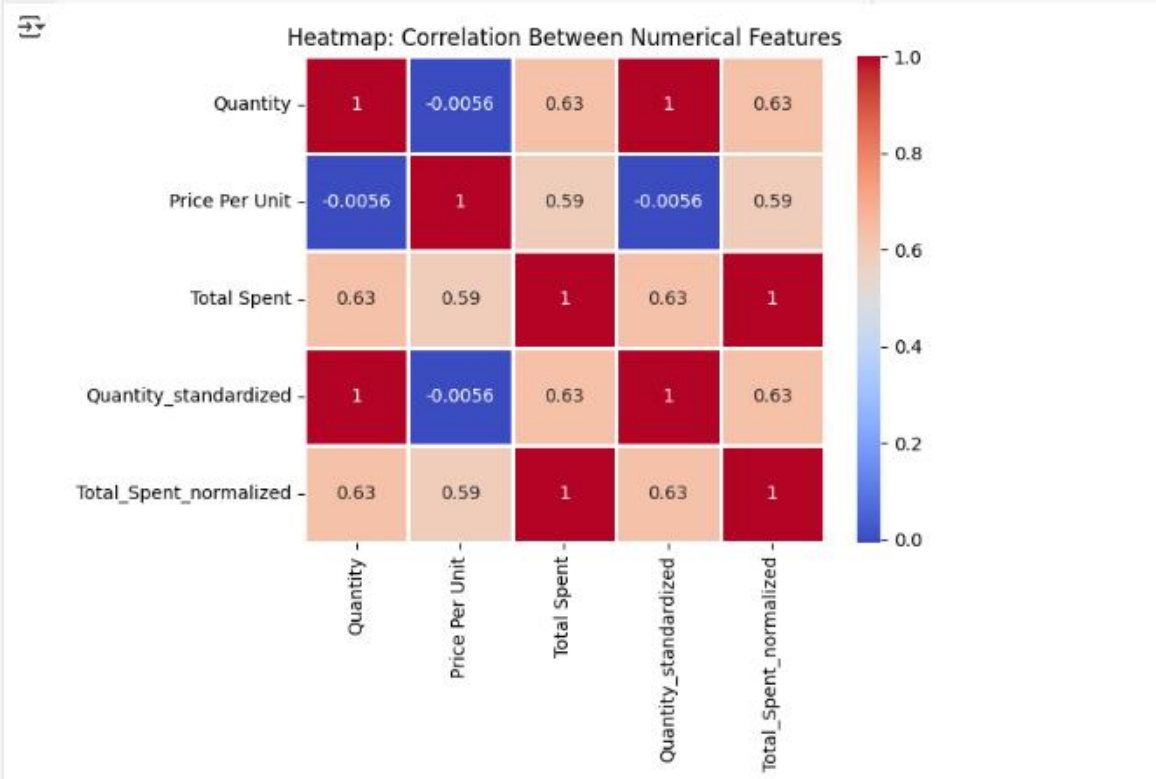
```
sns.boxplot(x=df_subset['Item'], y=df_subset['Total Spent'], palette="Set2")
```



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.

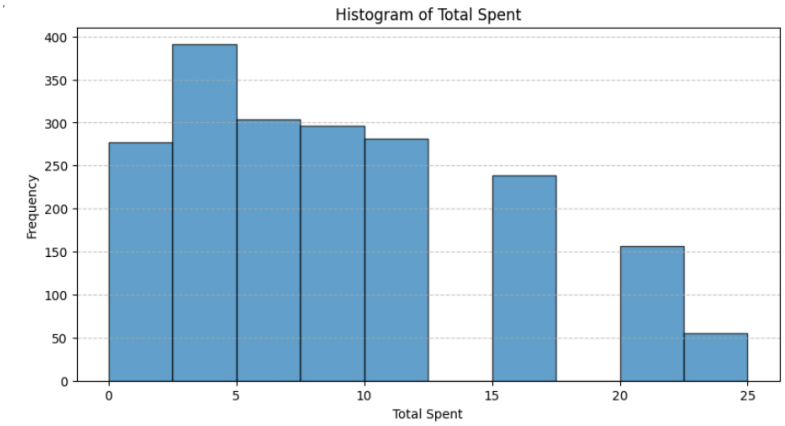
```
sns.heatmap(df_subset.corr(numeric_only=True), annot=True, cmap="coolwarm", linewidths=1)
plt.title("Heatmap: Correlation Between Numerical Features")
plt.show()
```



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).

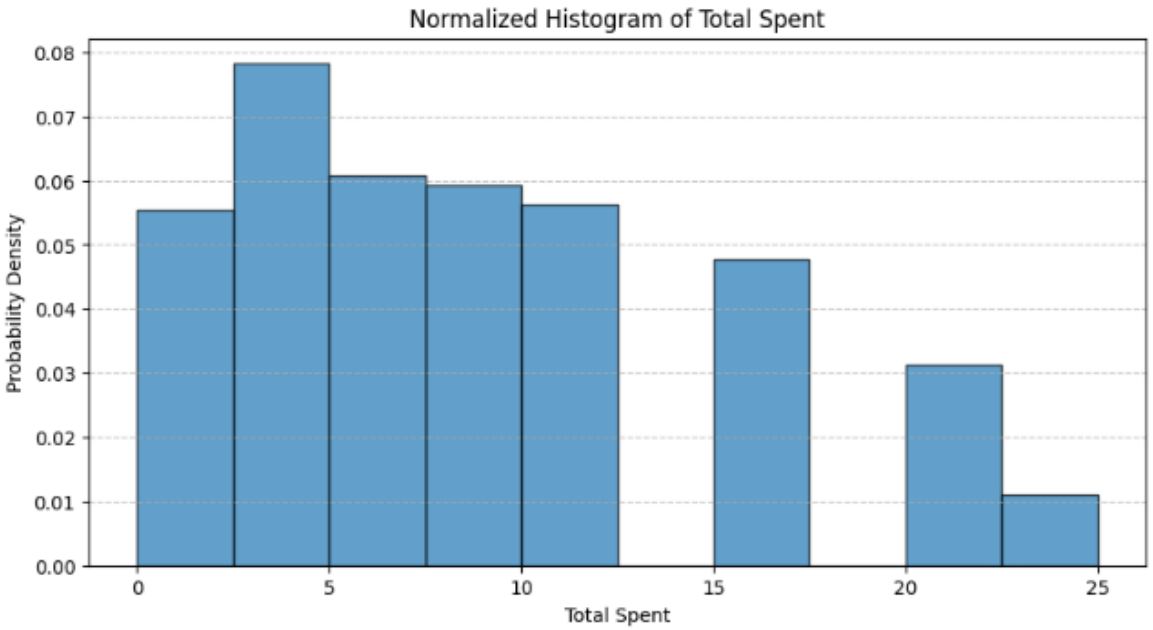
```
column_name = "Total Spent"

# 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.

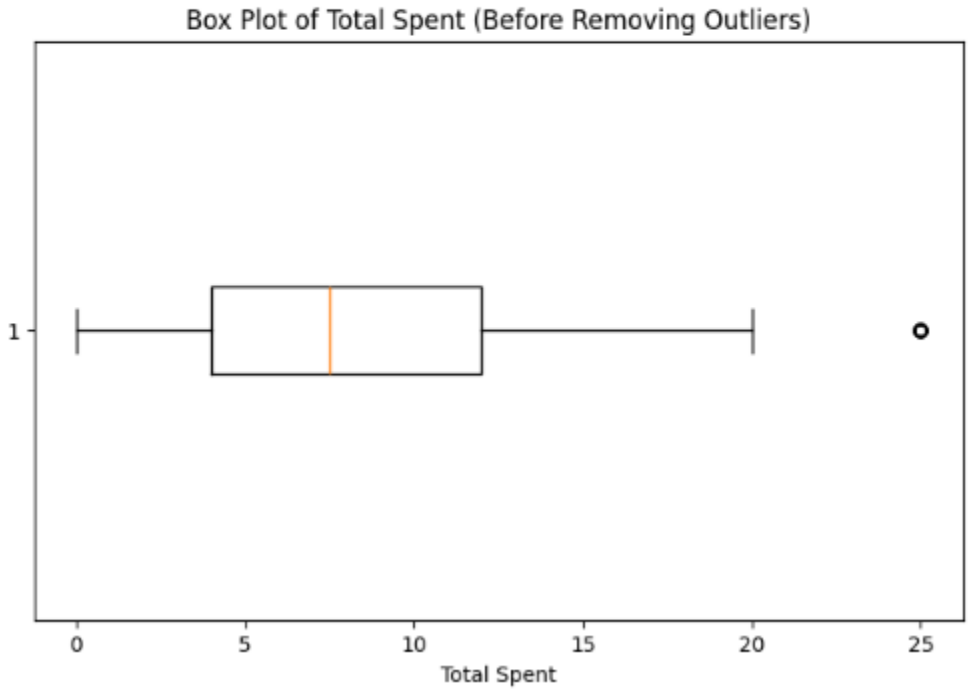
```
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, edgecolor='black', alpha=0.7, density=True)
plt.xlabel(column_name)
plt.ylabel("Probability Density")
plt.title(f"Normalized Histogram of {column_name}")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



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()
```

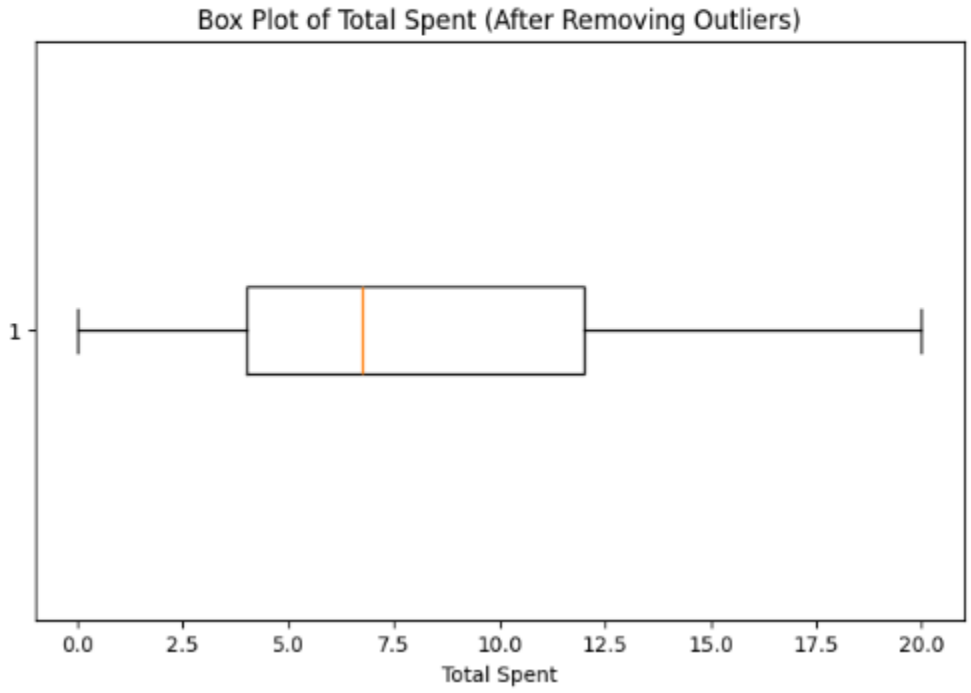


8. Box plot (after removing outlier)

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.

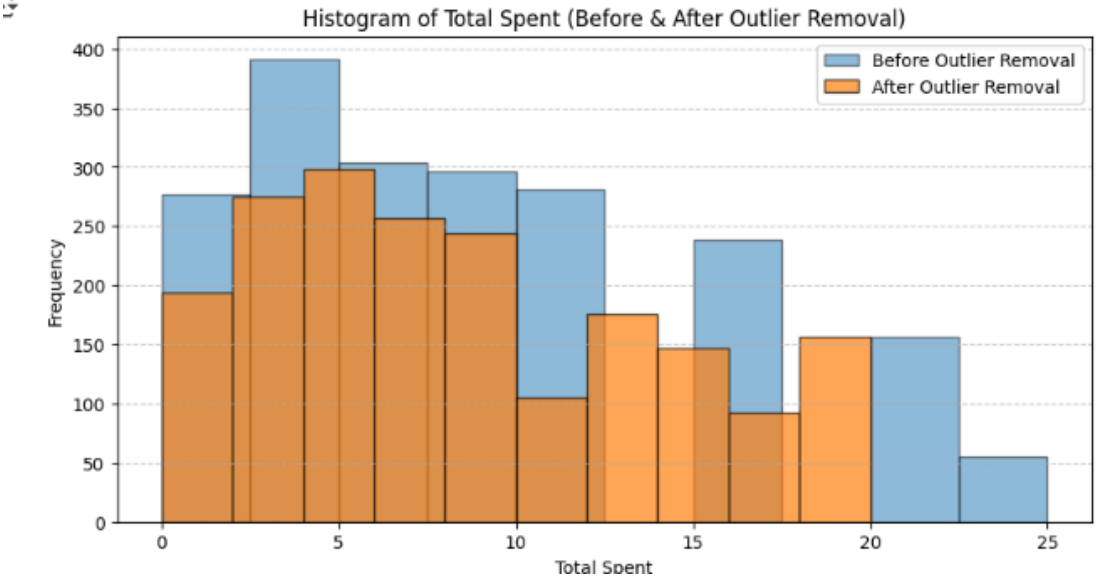
```
# Calculate IQR (Interquartile Range)
Q1 = df[column_name].quantile(0.25)  # First quartile (25th percentile)
Q3 = df[column_name].quantile(0.75)  # Third quartile (75th percentile)
IQR = Q3 - Q1  # Interquartile Range

# 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.

```python
# 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.

Experiment 3

Aim: Perform Data Modeling.

Problem Statement:

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

Steps:

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

Code:

```
from sklearn.model_selection import train_test_split

# 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:

2) 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
```

3) 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**.

Above, we performed the Z-score test manually without using any libraries. To check if we get the same results by using a library, we performed the test with an imported library

```
from statsmodels.stats.weightstats import ztest

z_stat, p_value = ztest(train_data["Total Spent"], test_data["Total Spent"])

print(f"Z-test result for total_spent: Z-stat = {z_stat}, P-value = {p_value}")
```

```
Z-test result for total_spent: Z-stat = -0.20397977377150175, P-value = 0.8383693048042808
```

Conclusion: The Z-test results confirm that the training and testing datasets do not differ significantly, ensuring a balanced partition. To maintain accuracy and reproducibility, it is important to consistently split the dataset into 75% training and 25% testing using a fixed random_state. Additionally, verifying column names, data types, and handling missing values properly helps avoid potential issues in future tests. Checking means, standard deviations, and sample sizes before computing the Z-score ensures the correctness of statistical comparisons

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: The Pearson correlation coefficient between Lead Time and Total Special Requests is -0.0031, indicating an almost nonexistent linear relationship. This means that as lead time increases or decreases, special requests remain nearly unchanged.The p-value is 0.27950, which is greater than 0.05, meaning the result is not statistically significant. Since we fail to reject the null hypothesis (H_0), there is no evidence of a linear relationship between these two variables. This suggests that lead time does not meaningfully impact the number of special requests in a predictive way.

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, indicating a very weak negative monotonic relationship. This means that as lead time increases, special requests tend to decrease slightly, but the effect is negligible.The p-value is extremely small (less than 0.01), which suggests statistical significance—meaning we reject the null hypothesis (H_0) that there is no relationship. However, despite the statistical significance, the correlation is so weak that it has little to no practical significance in predicting special requests based on lead time.

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's Tau correlation coefficient between Lead Time and Total Special Requests is -0.0577, indicating a very weak negative ordinal relationship. This means that as lead time increases, the ranking of special requests slightly decreases, but the effect is extremely small.The p-value is effectively 0 (extremely small, below any reasonable significance threshold), meaning the result is statistically significant and we reject the null hypothesis (H_0). However, despite this statistical significance, the correlation is so weak that it has no meaningful impact in practice. In other words, while a relationship technically exists, lead time is not a useful predictor of special requests.

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 statistic for the relationship between Hotel Type and Meal Type is 11,973.6428, with a p-value effectively 0 (extremely small). Since the p-value is far below 0.05, we reject the null hypothesis (H_0), meaning there is a statistically significant association between hotel type and meal type.This suggests that meal preferences are not independent of hotel type—guests at different hotel types (City Hotel vs. Resort Hotel) tend to choose meals differently. This dependency could be due to differences in hotel dining options, guest demographics, or package inclusions affecting meal choices.

Conclusion: Based on the statistical hypothesis tests performed, we analyzed the relationships between Lead Time and Total Special Requests using Pearson, Spearman, and Kendall correlation tests and examined the association between Hotel Type and Meal Type using the Chi-Square test.

The Pearson correlation coefficient (-0.0031) and p-value (0.27950) indicate no significant linear relationship between Lead Time and Total Special Requests, meaning that lead time does not predict the number of special requests a customer makes. Similarly, the Spearman (-0.0741) and Kendall (-0.0577) correlation coefficients suggest a very weak negative monotonic and ordinal relationship, with extremely small p-values confirming statistical significance. However, despite

this significance, the correlation values are so close to zero that the effect is negligible in practice. This implies that while there may be a detectable relationship, lead time is not a meaningful factor in determining special requests.

On the other hand, the Chi-Square test ($\chi^2 = 11{,}973.6428$, p-value ≈ 0) between Hotel Type and Meal Type confirms a strong dependency between the two variables, leading us to reject the null hypothesis. This means that meal preferences are significantly influenced by hotel type, possibly due to differences in dining services, guest demographics, or package offerings at City Hotels versus Resort Hotels.

AI and DS-1

Experiment 5

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

Objectives:

1. To perform Logistic regression to find out relation between variables
2. To apply regression model technique to predict the data on the selected dataset.

For this experiment we have selected the “Amazon Sales Report” dataset which consists of 24 columns and 130000 rows.

Steps performed:

1. Loading the dataset:

```
import pandas as pd

df = pd.read_csv("Amazon Sale Report.csv")

df.head()
```

	Order ID	Date	Status	Fulfilment	ship-service-level	Category	Size	Courier Status	Qty	Amount	ship-city	ship-state	ship-postal-code	B2B	fulfilled-by
0	405-8078784-5731545	04-30-22	Cancelled	Merchant	Standard	Set	S	NaN	0	647.62	MUMBAI	MAHARASHTRA	400081.0	False	Easy Ship
1	171-9198151-1101146	04-30-22	Shipped - Delivered to Buyer	Merchant	Standard	kurta	3XL	Shipped	1	406.00	BENGALURU	KARNATAKA	560085.0	False	Easy Ship
2	404-0687676-7273146	04-30-22	Shipped	Amazon	Expedited	kurta	XL	Shipped	1	329.00	NAVI MUMBAI	MAHARASHTRA	410210.0	True	NaN
3	403-9615377-8133951	04-30-22	Cancelled	Merchant	Standard	Western Dress	L	NaN	0	753.33	PUDUCHERRY	PUDUCHERRY	605008.0	False	Easy Ship
4	407-1069790-7240320	04-30-22	Shipped	Amazon	Expedited	Top	3XL	Shipped	1	574.00	CHENNAI	TAMIL NADU	600073.0	False	NaN

2. Preprocessing:

   Since the dataset is not clean, performing preprocessing is necessary before applying any models.

   a. Dropping columns that are not required for training the model

```
drop_cols = [
    'index', 'Order ID', 'SKU', 'ASIN', 'Date', 'Status', 'Fulfilment',
    'Sales Channel', 'ship-service-level', 'Courier Status', 'promotion-ids',
    'fulfilled-by', 'ship-city', 'ship-state', 'ship-postal-code', 'ship-country',
    'Unnamed: 22'
]
df = df.drop(columns=drop_cols, errors='ignore')
```

b. Replacing missing values in amount column with median

```
df['Amount'].fillna(df['Amount'].median())
```

	Amount
0	647.62
1	406.00
2	329.00
3	753.33
4	574.00
...	...
128970	517.00
128971	999.00
128972	690.00
128973	1199.00
128974	696.00

128975 rows × 1 columns

dtype: float64

c. Handling categorical values

```
from sklearn.preprocessing import LabelEncoder

df["B2B"] = df["B2B"].astype(int)

category_new = LabelEncoder()
df["Category"] = category_new.fit_transform(df["Category"])

size_new = LabelEncoder()
df["Size"] = size_new.fit_transform(df["Size"])

print(df.head())
```

```
   Category  Size  Qty  Amount  B2B
0         5     7    0  647.62    0
1         8     0    1  406.00    0
2         8     8    1  329.00    1
3         7     5    0  753.33    0
4         6     0    1  574.00    0
```

3. Importing necessary libraries

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import mean_squared_error,r2_score
```

Linear regression model:

1. Splitting and training the model

```
X = df.drop(columns=["Amount"])
y = df["Amount"]
```

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

model = LinearRegression()
model.fit(X_train, y_train)


y_pred = model.predict(X_test)
```

We have used linear regression model to predict amount based on other features in the dataset.
Y is the target variable (amount)
X contains the features that will be used for predicting amount (Category,Size,Qty,B2B)

2. Calculating MSE and R^2

We have computed two key metrics: Mean Squared Error (MSE) and R^2 Score to evaluate the performance of the 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
```

- MSE measures how far the predicted values are from the actual values.
- A lower MSE is better because it means that the model is making smaller errors.
- In our case the MSE came out to be 60078 which is quite high, indicating the model's predictions are not very close to the actual values.

- R^2 (R-squared) is a statistical measure that tells us how well our independent variables (features) explain the variation in the dependent variable (Amount).
- It ranges from 0 to 1:
  0 → The model explains none of the variability (bad model).
  1 → The model explains 100% of the variability (perfect model).
  0.206 (our case) → The model explains only 20.6% of the variability in Amount, which is low.

High MSE and Low R^2 suggest that:

1. Some important features might be missing.
2. There might be non-linear relationships that Linear Regression can't capture.

Using a different set of features to perform linear regression:

Here we have computed the average transaction amount per city and state using the ship-city and ship-state columns.

```
# Compute mean Amount per city & state
city_avg_price = df.groupby('ship-city')['Amount'].mean().to_dict()
state_avg_price = df.groupby('ship-state')['Amount'].mean().to_dict()

# Map the computed values to new columns
df['city_avg_amount'] = df['ship-city'].map(city_avg_price)
df['state_avg_amount'] = df['ship-state'].map(state_avg_price)
```

Here we have calculated the Qty_size by multiplying values from quantity and size columns. This represents the total volume or weight of the order depending on the size.

```
df['Qty_Size'] = df['Qty'] * df['Size']
```

Based on the average transaction amount, we have calculated the Qty_CityImpact

```
df['Qty_CityImpact'] = df['Qty'] * df['city_avg_amount']
df['Qty_StateImpact'] = df['Qty'] * df['state_avg_amount']
```

Splitting and testing the model again:

```
df.drop(columns=['ship-city', 'ship-state'], inplace=True)

X = df[['Qty', 'Size', 'city_avg_amount', 'state_avg_amount', 'Qty_Size', 'Qty_CityImpact', 'Qty_StateImpact']
       + list(df.filter(like='Category_').columns)]  # Keeping one-hot encoded Category

y = df['Amount']
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

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

# Train Model
model = LinearRegression()
model.fit(X_train, y_train)

# Make Predictions
y_pred = model.predict(X_test)

# Evaluate Model
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 67979.03941884013
```

The MSE is still very high which means even after feature engineering the model's predictions did not improve.

We can conclude that,

- The relationship between Amount and other columns is not a simple straight-line relationship.
- Linear Regression assumes y = mX + b, but Amount might depend on complex interactions or non-linear patterns.
- If external factors (e.g., demand, seasonality, discounts) impact Amount, but those are not present in our dataset.

Logistic regression:

We are using logistic regression to classify the customer as high spender or low spender

1. Creating target variable

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd
import numpy as np

df["High_Spend"] = (df["Amount"] > df["Amount"].median()).astype(int)
```

The target variable (High_Spend) is defined as whether the order's Amount is greater than the median Amount.
If Amount is greater than the median, it assigns 1 (High Spend); otherwise, 0 (Low Spend).

2. Handling categorical values

```
categorical_cols = ["Category", "Size"]  # Specifing the categorical columns
X_categorical = df[categorical_cols]

# One-hot encode categorical features
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
X_categorical_encoded = encoder.fit_transform(X_categorical)
X_categorical_encoded = pd.DataFrame(X_categorical_encoded, columns=encoder.get_feature_names_out(categorical_cols))
```

OneHotEncoder converts categorical features (Category and Size) into numerical format for the model.

3. Selecting numerical features

```
# Select only relevant numeric columns (excluding Amount & High_Spend)
X_numeric = df.select_dtypes(include=['number']).drop(columns=["Amount", "High_Spend"])
```

Here the features other than amount and high spend have been selected. We have not included high spend since it is the target variable and amount because it directly determines target variable.

4. Train test split

```
X = pd.concat([X_numeric, X_categorical_encoded], axis=1)

y = df["High_Spend"]

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

Now we have the numeric + encoded categorical values in X and Y contains target variable High Spend.
Next, we have splitted the dataset - 80% training and 20% testing.

5. Standardizing numerical features

```
# Standardize numerical features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Standardizing helps the model perform better

6. Training and predictions

```
# Train Logistic Regression Model
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Make Predictions
y_pred = model.predict(X_test_scaled)
```

We have fit the scaled data to the regression model and the model can now classify the data as high spend or low spend

7. Evaluating model performance

```
# 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
```

Here we have calculated the accuracy of the model which is 84.43%

Classification report summary:

- Class 0 (Low Spend Customers)
    - Precision = 0.87 → When the model predicts "Low Spend," it is correct 87% of the time.
    - Recall = 0.83 → The model correctly identifies 83% of actual "Low Spend" customers.
    - F1-score = 0.85 → A good balance between precision and recall.

- Class 1 (High Spend Customers)
    - Precision = 0.82 → When the model predicts "High Spend," it is correct 82% of the time.
    - Recall = 0.86 → The model correctly identifies 86% of actual "High Spend" customers.
    - F1-score = 0.84 → A strong score, showing that both precision and recall are balanced.

Conclusion:

In this experiment we used linear regression to predict amount based on other features but the MSE turned out to be very high and R^2 value very low which indicates that amount may not have a linear relationship with other attributes. We used logistic regression to classify spenders as high spender and low spender based on the amount attribute. The accuracy of the model came out to be 84.43% which tells us that the model is accurately classifying the data.

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.

```
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

```
# 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())
```

```
# 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

```
# 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

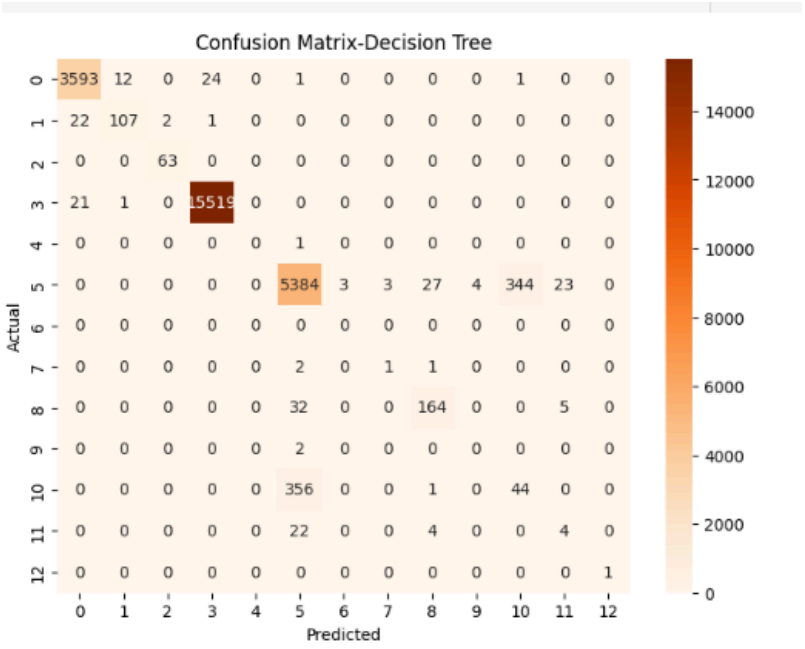
```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train,)

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

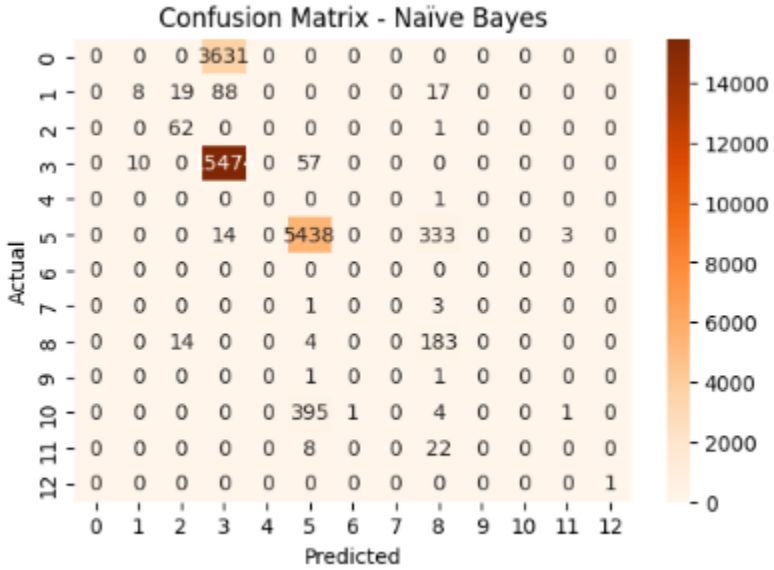
```
print("Naïve Bayes Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
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      0.98      0.78        63
           3       0.81      1.00      0.89     15541
           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       201
           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.

Name: Nayaab Jindani D15C Rollno.20

AI and DS-1

Experiment 7

Aim: To implement different clustering algorithms.

Problem statement:

a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN))

b) Plot the cluster data and show mathematical steps.

Theory:

Clustering is a type of unsupervised learning technique. In unsupervised learning, we work with datasets that do not have predefined labels or outputs. The goal is to uncover hidden patterns, structures, or groupings within the data. Clustering specifically involves grouping a collection of data points into clusters, where points within the same cluster are more similar to each other than to those in other clusters.

The primary objective of clustering is to group data based on similarities, so that elements in the same group share common traits or characteristics.

Real-World Applications of Clustering

1. Marketing: Clustering is used to segment customers based on purchasing behavior or demographics to enable targeted marketing strategies.

2. Biology: It's applied in identifying and grouping various species based on genetic traits or observable features.

3. Library Organization: Books and resources can be grouped based on topics or genres for easier access.

4. Insurance Sector: Helps in grouping policyholders to detect fraudulent claims or tailor insurance plans.

5. Urban Development: Clustering assists in analyzing housing patterns and neighborhood planning based on location, price, and amenities.

6. Seismology: Clustering earthquake-prone zones can aid in identifying risk levels and preparing for natural disasters.

Types of Clustering Algorithms

When selecting a clustering algorithm, scalability and efficiency are crucial—especially with large datasets. Some algorithms compute the similarity between every pair of points, which makes them computationally expensive, with time complexity of $O(n^2)$, making them impractical for millions of data points.

1. Density-Based Clustering

These methods define clusters as dense regions in the data space, separated by areas of lower density. They perform well with irregularly shaped clusters and can handle noise.
Examples: DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS.

2. Hierarchical Clustering

This approach builds a tree of clusters based on hierarchical relationships. It starts either from individual data points (bottom-up) or from a single cluster (top-down).
Types:

- Agglomerative: Merges smaller clusters into larger ones.

- Divisive: Splits large clusters into smaller groups.
  Examples: CURE, BIRCH.

3. Partitioning Clustering

This technique divides the dataset into a fixed number of clusters (k). It seeks to optimize a given objective function, often based on distance or similarity.
Examples: K-Means, CLARANS.

4. Grid-Based Clustering

In this method, the data space is divided into a grid structure, and clustering is performed on these grids instead of individual points. It is highly efficient and independent of the data size.
Examples: STING, CLIQUE, WaveCluster.

Dataset description:

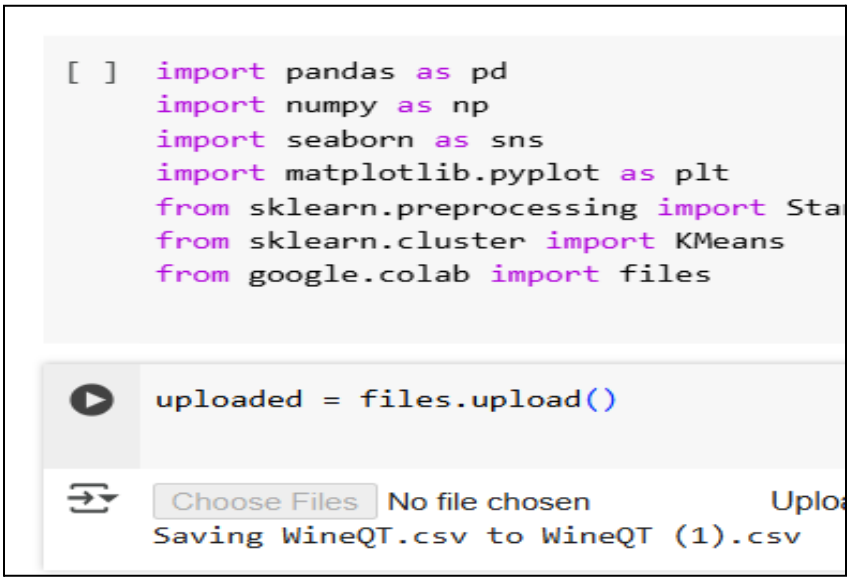
For this experiment we have used the Wine Quality dataset which contains physicochemical properties of red wine samples along with a quality score rated by experts.

Its attributes are:

1. fixed acidity – Non-volatile acids (e.g., tartaric acid) contributing to stability and flavor.
2. volatile acidity – Acetic acid content; high levels may result in an unpleasant taste.
3. citric acid – Adds freshness and flavor to wine.
4. residual sugar – Sugar left after fermentation; affects sweetness.
5. chlorides – Salt content in the wine.
6. free sulfur dioxide – SO_2 available to protect wine from microbes and oxidation.
7. total sulfur dioxide – Total concentration of SO_2 (free + bound).
8. density – Affected by sugar and alcohol content.
9. pH – Indicates the acidity or alkalinity of wine.
10. sulphates – Acts as a preservative and antioxidant.
11. alcohol – Percentage of alcohol by volume.
12. quality – Target variable; wine quality score (typically 3 to 8).
13. Id – Unique identifier for each sample (not used in modeling).

Steps performed:

Importing libraries and loading dataset:



This is the first step wherein we are importing the necessary libraries and loading the dataset.

Performing Kmeans clustering:

We are primarily selecting 2 columns on which our clustering will be based i.e volatile acidity and alcohol.

Step 1. Scaling the features:

```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

Here we are creating a new data frame with only the required 2 features and applying scaling to the features. Scaling is applied to standardize the features, i.e., scale them so that: Mean = 0 and Standard Deviation = 1

Step 2. Using PCA (Principal component analysis)

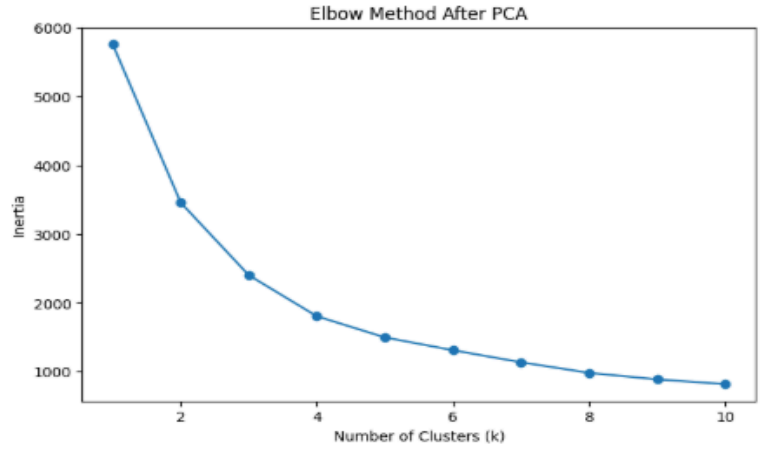
```
from sklearn.decomposition import PCA

# Reduce dimensions to 2 or 3
pca = PCA(n_components=2)
features_pca = pca.fit_transform(features_scaled)

# Check variance retained
print(f"Total variance retained: {sum(pca.explained_variance_ratio_):.2f}")
```

```
Total variance retained: 0.46
```

We are using PCA to reduce the number of features to 2 dimensions so we can visualize clusters and simplify the data while still retaining most of the important information (variance).

Step 3. Elbow method

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

inertia = []
K_range = range(1, 11)  # Test k from 1 to 10

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

# Plot the Elbow curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker="o")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.title("Elbow Method After PCA")
plt.show()
```



Here we have used the Elbow Method to find the optimal number of clusters for KMeans.

Why the Elbow Method?

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

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

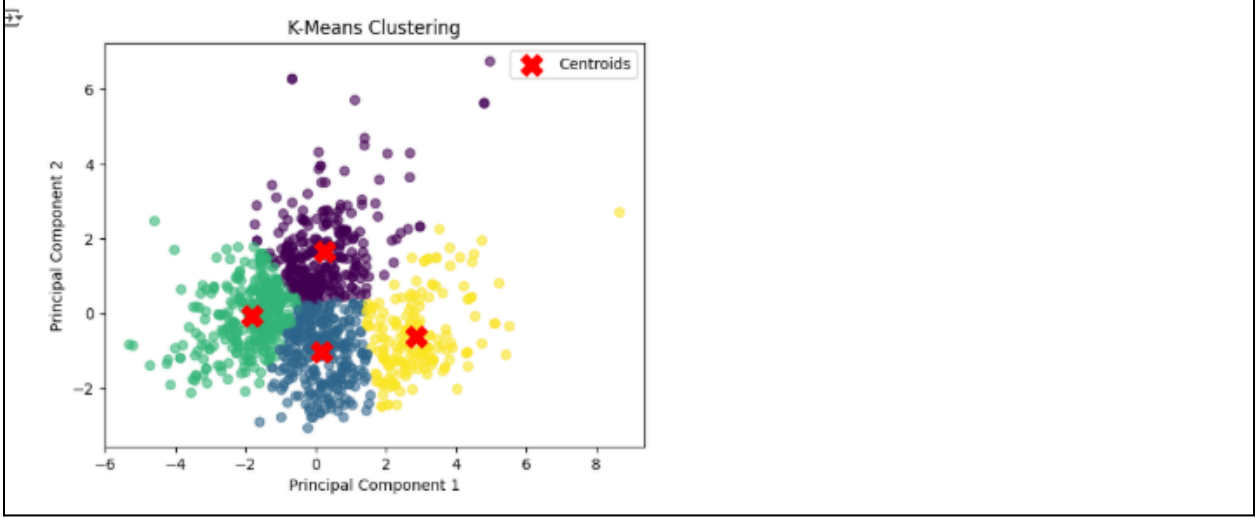
We select the "elbow point" – where the decrease in WCSS slows down – as the best k.

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
clusters = kmeans.fit_predict(features_pca)
```

Based on the elbow method we have selected k=4 and now we will visualize the clusters

Step 4. Visualizing the clusters

```
# Visualize clusters
plt.scatter(features_pca[:, 0], features_pca[:, 1], c=clusters, cmap="viridis", alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], c="red", marker="X", s=200, label="Centroids")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("K-Means Clustering")
plt.legend()
plt.show()
```



The scatter plot visualizes the clusters formed using K-Means after reducing dimensionality with PCA. Five distinct clusters are clearly observed, with red X marks indicating the centroids. The clusters are relatively compact and well-separated, suggesting that K-Means performed effective segmentation of the data. Only minor overlaps and a few outliers are visible.

Step 5. Calculating silhouette score and Davies Bouldin score

Now we will calculate the Silhouette Score, which measures how well the data points fit within their assigned clusters.

- Range: -1 to +1
    - +1 → Perfect clustering
    - 0 → Overlapping clusters
    - Negative → Misclassified points

    Here is the calculated silhouette score:

```
from sklearn.metrics import silhouette_score

score = silhouette_score(features_pca, clusters)
print(f"Silhouette Score: {score:.3f}")
```

```
Silhouette Score: 0.372
```

A silhouette score of 0.372 shows that clustering is moderate. Clusters exist, but they overlap or are not well-defined.

Calculating Davies Bouldin score:

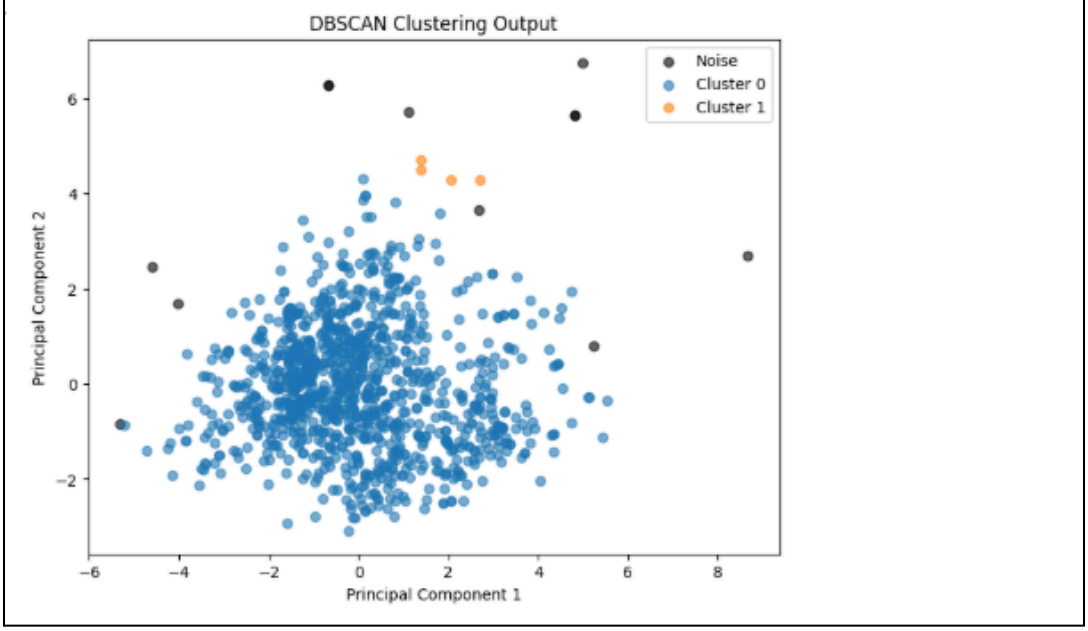
```
from sklearn.metrics import davies_bouldin_score

db_score = davies_bouldin_score(features_pca, clusters)
print(f"Davies-Bouldin Score: {db_score:.3f}")
```

```
Davies-Bouldin Score: 0.852
```

A Davies-Bouldin Score of 0.852 indicates that the clusters are reasonably well-separated and moderately compact.

Performing DBSCAN:

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


dbscan = DBSCAN(eps=0.8, min_samples=5)
clusters_dbscan = dbscan.fit_predict(features_pca)

unique_labels = np.unique(clusters_dbscan)


plt.figure(figsize=(8, 6))
for label in unique_labels:
    if label == -1:
        plt.scatter(features_pca[clusters_dbscan == label, 0],
                    features_pca[clusters_dbscan == label, 1],
                    color='black', label='Noise', alpha=0.6)
    else:
        plt.scatter(features_pca[clusters_dbscan == label, 0],
                    features_pca[clusters_dbscan == label, 1],
                    label=f'Cluster {label}', alpha=0.6)

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("DBSCAN Clustering Output")
plt.legend()
plt.show()
```



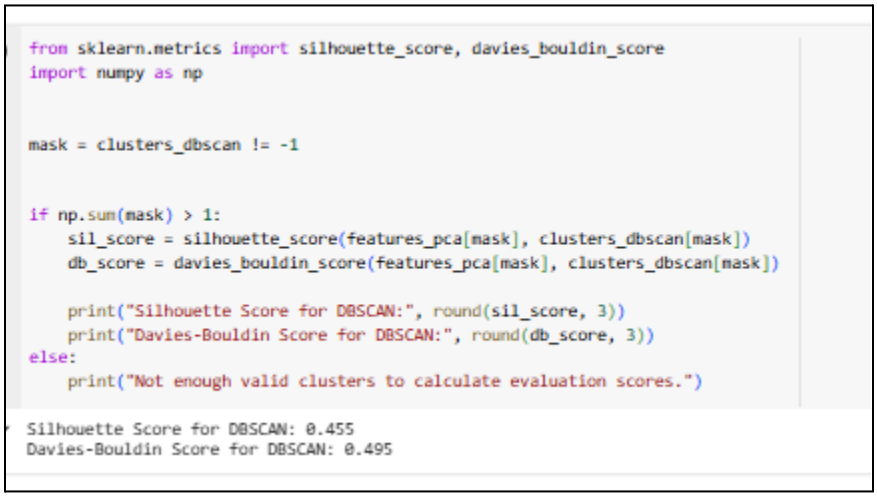
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was applied on the PCA-reduced features to discover clusters based on density. The parameters used were eps=0.8 and min_samples=5. The scatter plot shows several clusters formed with some points labeled as noise (black), indicating that they didn't belong to any dense region.

What are eps and min_samples in DBSCAN

- eps (epsilon): This defines the maximum distance between two points for them to be considered neighbors.
  → Smaller eps → tighter clusters, more noise points.
  → Larger eps → looser clusters, fewer noise points.

- min_samples: This is the minimum number of points required to form a dense region (a cluster).
  → Smaller min_samples → more, smaller clusters (can lead to noise).
  → Larger min_samples → fewer, larger clusters (may merge nearby clusters).

Together, these parameters control how sensitive DBSCAN is to density. We usually need to try different combinations of eps and min_samples to get the best clustering result.



```
from sklearn.metrics import silhouette_score, davies_bouldin_score
import numpy as np


mask = clusters_dbscan != -1


if np.sum(mask) > 1:
    sil_score = silhouette_score(features_pca[mask], clusters_dbscan[mask])
    db_score = davies_bouldin_score(features_pca[mask], clusters_dbscan[mask])

    print("Silhouette Score for DBSCAN:", round(sil_score, 3))
    print("Davies-Bouldin Score for DBSCAN:", round(db_score, 3))
else:
    print("Not enough valid clusters to calculate evaluation scores.")
```

```
Silhouette Score for DBSCAN: 0.455
Davies-Bouldin Score for DBSCAN: 0.495
```

The DBSCAN algorithm achieved a Silhouette Score of 0.455, indicating moderately well-defined clusters. The Davies-Bouldin Score of 0.495 suggests that the clusters are compact and well-separated, confirming that DBSCAN effectively identified meaningful groupings in the data.

Conclusion:

Based on the evaluation metrics, DBSCAN performed better than KMeans for clustering the wine dataset. While KMeans achieved a Silhouette Score of 0.372 and a Davies-Bouldin Score of 0.852, DBSCAN showed improved clustering quality with a Silhouette Score of 0.455 and a lower Davies-Bouldin Score of 0.495. This indicates that DBSCAN formed more compact and well-separated clusters. Additionally, DBSCAN's ability to identify outliers and handle clusters of varying shapes contributed to its superior performance in this context.

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**

```
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 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 realtime. 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, highthroughput, 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.

Name : Nayaab Jindani Class & Div.: D15 C Page No.:

Roll No.: 20 Subject : Topic : Date : 05/05

AIDS Assignment 1.

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

→ AI is a technology that enables computers and machines to simulate human learning, comprehension, problem solving, decision making, creativity and autonomy. Devices equipped with AI can understand and respond to human language, learn from new information and can act independently.

AI's help during COVID-19:
Amid the crisis of COVID-19, AI emerged as a critical tool for addressing the pandemic's multifaceted challenges.
1. During the early stages of the pandemic, china implemented AI technologies to manage and mitigate virus spread.
2. AI also enabled contact tracing and diagnostic tools that played key roles in several provinces.
3. Major cities employed AI based thermal imaging and facial recognition at transport hubs to efficiently identify symptomatic individuals and enforce quarantine measures.
4. AI-powered chatbots and virtual assistants became integral in healthcare offering patients remote consultations and medical advice.

2. What are AI agents terminology, explain with examples.

→ • Performance measure of agent: Determine the success of an agent
• Behaviour / action of agent: It is the action performed

by an agent after any specified sequence of percepts.
- Percept : Agent's perceptual inputs at a specified instance.
- Percept sequence : History of everything that an agent has perceived till date.
- Agent function : Map the percept sequence to action

Example : Vacuum cleaner problem.
- Performance measure :- All rooms are well ~~cleaned~~ cleaned.
- Behaviour :- Left, right, suck and no-op (Doing nothing)
- Percept - Location and status
- Agent function - Mapping (percept sequence, action)

Eg.

Percept sequence	action
[A, clean]	right
[A, Dirty]	suck
[B, clean]	left

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

—> AI techiques are used to solve 8 puzzle problem by applying search algorithms and heuristic functions.

1. Problem representation : The 8 puzzle is represented as a state space where each state is a 3×3 grid configuration.

Initial state :
1 2 3
4 0 6
7 5 8

Goal state :
1 2 3
4 5 6
7 8 0

Steps to solve 8 puzzle problem by A* :
1. Initialize a priority queue.
2. Insert the initial state with $f(n) = g(n) + h(n)$

Name : _____ Class & Div.: _____ Page No.: _____

Roll No.: _____ Subject : _____ Topic : _____ Date : _____

3. While queue not empty :
   Remove state with lowest $f(n)$
   If state = goal, return solution
   Generate valid moves (up, down, left, right)
   Compute $g(n)$ and $h(n)$ for new states.
   Insert new states into queue.
4. Repeat ~~up~~ until goal is reached.

4. What is PEAS descriptor ? Give PEAS descriptor for following :

→ PEAS stands for Performance, Environment, Actuators and Sensors.
P → criteria to evaluate agent's success.
E → surroundings / external area where agent operates
A → components that allow agent to take actions
S → components that help agent perceive its environment.

PEAS for following :

1. Taxi driver
   P - Reaching destination, fuel efficiency
   E - Roads, traffic, pedestrians
   A - Steering wheel, brakes, accelerator
   S - Camera, GPS, speedometer

2. Medical diagonosis system :
   P - Accuracy of diagnosis, speed of ~~dig~~ diagnosis
   E - Medical records, test results, ~~symptoms~~, hospitals
   A - Recommending treatments, sending alerts to patients and doctors, updating medical records.

S - Wearable sensors, medical imaging devices

3. Music composer
P - User satisfaction, quality of composition
E - music production studio, streaming platforms
A - Adjusting pitch and key of compositions, suggesting chord progressions and melodies.
S - MIDI inputs, music databases, lyrics or text for melody generation

4. Aircraft autolander
P - Smooth and safer landing
E - weather, runway
A - gear, throttle, flaps.
S - altitude sensor, GPS, wind direction sensor.

5. Essay evaluator
P - Grading accuracy, feedback quality, accuracy in error checking
E - ~~Submitted Essays~~ Educational institutes, competitive exams
A - Highlighting grammar and spelling errors, checking for plagiarism
S - Text input, NLP, AI based readability assessment tools.

6. Robotic sentry gun for Keck lab
P - correctly identifying threats and targets, speed of response
E - Keck lab facility, research centers
A - Tracking, alerting security personnel.
S - Cameras, motion ~~sensors~~ sensors, AI based threat recognition

Name : _____ Class & Div.: _____ Page No.: _____

Roll No.: _____ Subject : _____ Topic : _____ Date : _____

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

→ • Partially observable - The bot cannot fully observe customer preferences or book placements.
• Stochastic - customer behaviour and book availbility are unpredictable.
• Sequential - Each interaction depends on previous queries and actions.
• Dynamic - The bookstore environment constantly changes.
• Discrete - The bot operates with a finite set of books, actions and interactions.
• ~~Mu~~ Multi - agent - The bot interacts with customers, employees and inventory systems.

6. Differentiate model based and utility based agent.

→

Model based agent	Utility based agent
1. Uses an internal model of environment to make decisions	1. chooses actions based on utility function that measures performance.
2. Decisions are based on past and present percepts	2. Selects action based on maximizing utility.
3. Can be goal based but doesn't necessarily optimize for best outcomes.	3. Is goal based and searches for the most optimal solution

Attitude

4. Example: Robot vacuum using a map to navigate	4. Example: self driving car.

7. Explain the architecture of a knowledge based agent and learning agent.

→ Architecture of knowledge based agent:
It uses stored knowledge to make decisions and consists of the following:
- Knowledge base → stores facts, rules and logic
- Inference engine - Uses reasoning to derive conclusions
- Perception (sensors) - gather new information from environment.
- Action mechanism - Performs appropriate actions based on reasoning

Architecture of learning agent:
This agent improves performance over time and consists of the following:
- Learning element - updates 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

9. Convert the following to predicates:

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

Name : _____ Class & Div.: _____ Page No.: _____

Roll No.: _____ Subject : _____ Topic : _____ Date : _____

→ CarAvailable → Travels By Car (Anita)
¬ CarAvailable → Travels By Bus (Anita)

b. Bus goes via Andheri and goregaon

→ GoesVia (Bus, Andheri) ∧ GoesVia (Bus, Goregaon)

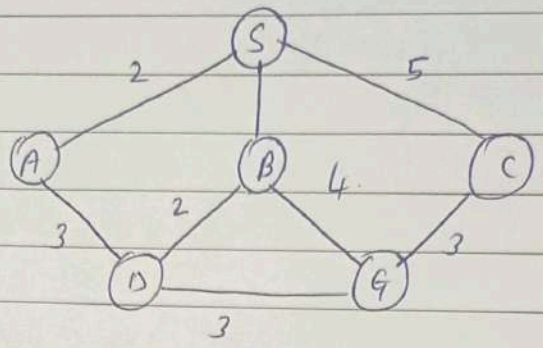
c. Car has puncture so is not available.
Puncture (car)
Puncture (car) → ¬ CarAvailable.

Will Anita travel via Goregaon? Use forward reasoning.

From (c)
Puncture (car) is true
As Puncture (car) → ¬CarAvailable
From (a)
¬CarAvailable, we use ¬CarAvailable → Travels By Bus (Anita)
From (b)
~~¬CarAvailable, we use~~
Goes via (Bus, Goregaon)
Since Anita travels by bus she will follow this route

Thus, Anita will travel via goregaon.

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



→ 1. Start at S
Queue = [S]
2. Dequeue S and explore its ~~children~~ neighbouring nodes
Queue = [A, B, C]
3. Dequeue A and explore neighbours.
Queue = [B, C, D]
4. Dequeue B and explore neighbours.
Queue = [C, D, G]
5. Dequeue C and explore neighbours
Queue = [D, G]
6. Dequeue D
Queue = [G]
7. Dequeue G

∵ G is our destination node, BFS will stop here.
Route from S to G : S -> B -> G

11. What do you mean by depth limited search ? Explain Iterative Deepening search with example.

→ Depth limited search (DLS) is an uninformed search algorithm that modifies DFS ~~by~~ by introducing a depth limit L preventing exploration beyond the defined level. This prevents infinite ~~loo~~ loops in graphs but

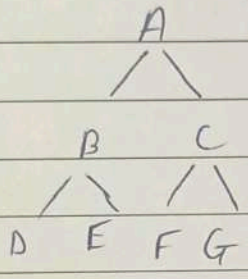
Name : _____ Class & Div.: _____ Page No.: _____

Roll No.: _____ Subject : _____ Topic : _____ Date : _____

risks missing goals beyond L.

Iterative Deepening Search (IDS) combines DLS with BFS by incrementally increasing the depth limit

Example :



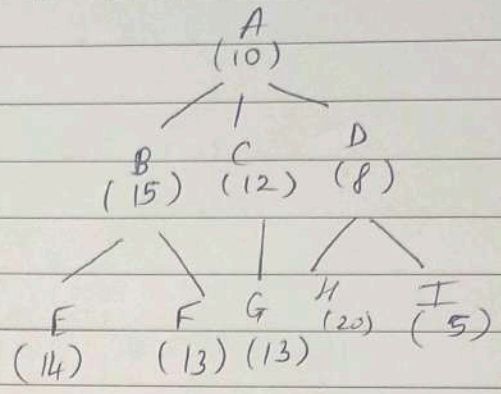
Goal : G

- Initially the depth limit is 0 for iteration 1
  Nodes visited = A
  Goal not found
- Iteration 2, Limit = 1
  Nodes visited = A -> B -> C
  Goal not found
- Iteration 3, Limit = 2
  Nodes visited = A -> B -> D -> E -> C -> F -> G
  Goal G is found.

12. Explain hill climbing and its drawbacks in detail with example. Also state limitations of steepest ascent hill climbing

-> Hill climbing is a local search optimization algorithm which moves forwards towards a better neighbouring solution until it reaches a peak.

Example :



Goal : G

Steps :
1. Start at root node A (10)
2. Compare its children B, C, D
3. Move to child with highest value ie B(15)
4. Repeat for B's children E and F
5. Terminate at E(14)

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

Drawbacks :
1. Local maxima - The algorithm greedily selects the best immediate child and can thus get stuck on local maxima.
2. Plateau - If siblings have equal values, the algorithm can't decide the next step and gets stuck.
3. Ridges - Narrow uphill paths require backtracking which hill climbing algorithm does not support

Limitations of steepest ascent hill climbing :
1. Computationally expensive : Evaluates all neighbours

Name:_____ Class:_____ Div:_____ Roll No:_____

Subject:_____ Topic:_____ Date:_____ Page No:_____

before selecting the best.

2. Can get stuck - It can still get stuck in local maxima, plateaus or ridges.
3. No global optimality - It only focuses on immediate improvements.

13. Explain simulated annealing and write its algorithm.

→ Simulated annealing is a probabilistic optimization algorithm inspired by metallurgical process of annealing where materials are heated and cooled to reduce defects. It escapes the local optima by temporarily accepting worse solution with a probability.

Algorithm:

1. Initialize
   - Set an initial solution and define an initial temperature T
2. Repeat until stopping condition
   - Generate a new neighbour solution
   - Compute changes in cost
   - If new solution is better then accept it
   - If worse, accept it with probability
   - Decrease temperature T
3. Return best solution

Example: Travelling salesman problem.

14. Explain A* algorithm with an example.

→ A* is a best first search algorithm used in pathfinding and graph traversal. It uses the following formulas

AASTMT

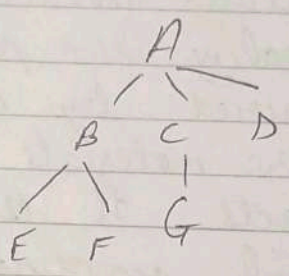
$f(n) = g(n) + h(n)$

$g(n)$ → cost to reach n from start

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

$f(n)$ → total estimated cost.

Goal : G



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

Steps

1. Start at root node A
   $f(A) = g(A) + h(A) = 0 + 6$
   $= 6$
2. Expand neighbors B, C, D
   $f(B) = 1 + 4 = 5$
   $f(C) = 2 + 2 = 4$
   $f(D) = 4 + 7 = 11$
3. Choose lowest value that is $f(C)$
4. Expand neighbours of C

Name: _____ Class: ____ Div: ____ Roll No: ____

Subject: _____ Topic: _____ Date: _____ Page No: _____

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

5. Goal reached at G with total cost 6

Advantages →

- efficient for finding shortest path in weighted graphs
- balances exploration by considering both $g(n)$ and $h(n)$.

15. Explain min max algorithm and draw game tree for Tic Tac Toe game.

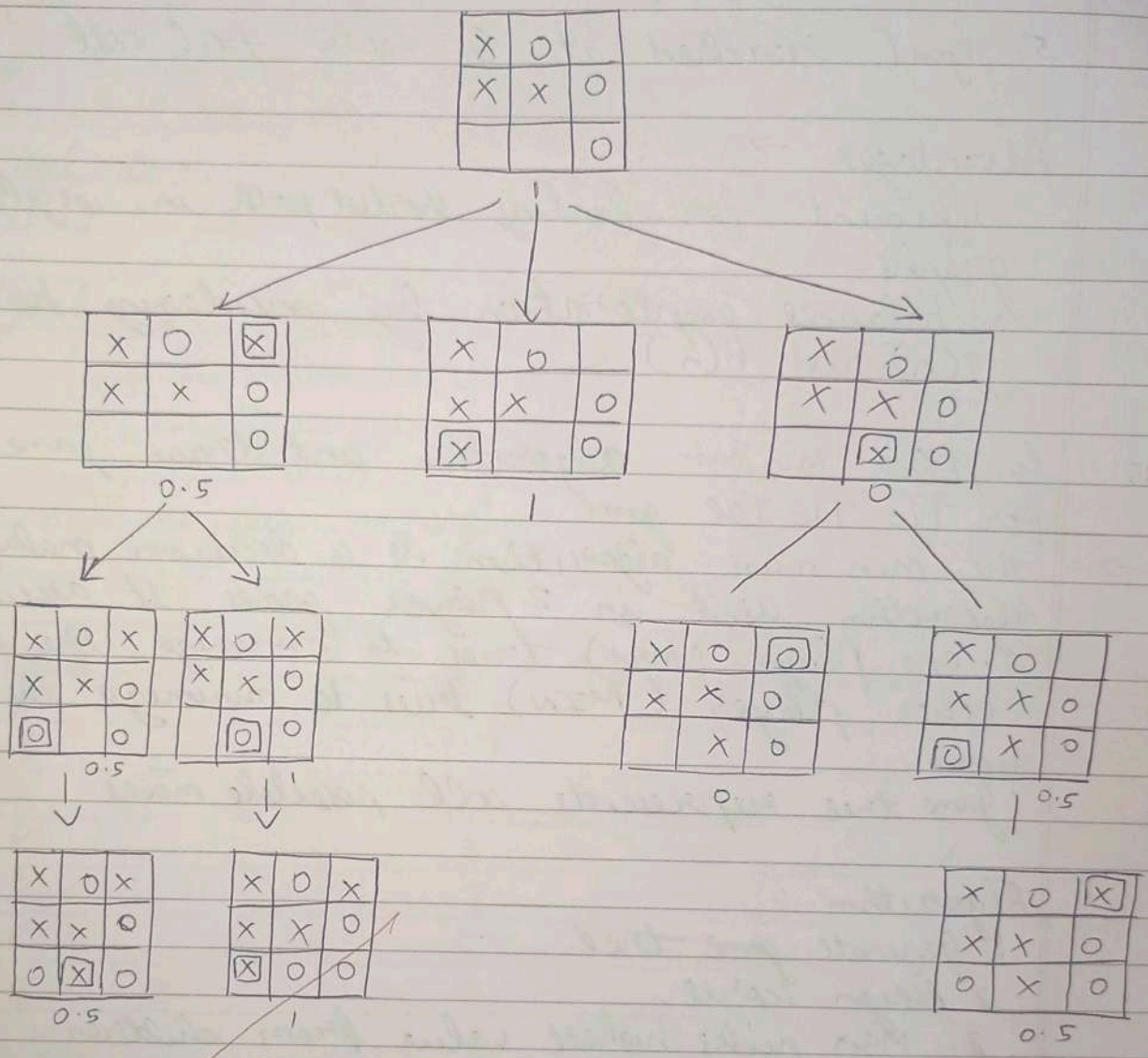
→ The min max algorithm is a decision making algorithm used in 2 player games. It assumes.

- 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
2. Assign scores
3. Max picks highest value from children
   MIN picks lowest value.
4. Repeat until root node is evaluated.

Game tree for tic tac toe game :



16. Explain alpha beta pruning algorithm for adversarial search with example.

-> Alpha beta pruning is an optimization technique used in minimax algorithm to reduce the number of nodes evaluated in adversarial search problems like game-playing AI (eg. chess, tic-tac-toe)

Alpha beta pruning includes:

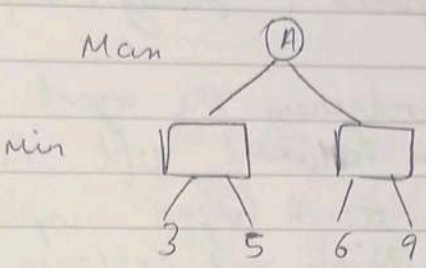
Alpha (α): The best maximum score that the maximizing player can guarantee so far.

Name: _____ Class: ____ Div: ____ Roll No: ____

Subject: _____ Topic: _____ Date: _____ Page No: _____

Beta (β) : The best minimum score that the minimizing player can guarantee so far.
The algorithm prunes branches that will not influence final decision

Example :



1. Start at root node A
   $\alpha = -\infty$, $\beta = \infty$
2. check left min node (child of A)
   - check first child. value = 3 -> update $\beta = 3$
   - check second child : Value = 5 -> β remains 3
   - Min node returns 3 to MAX.
3. Right Min node (child of A)
   - check first child : value = 6 -> $\beta = 6$
   - pruning will occur : $\beta(6)$
   - Here $\alpha = 3$ at MAX node but $\beta(6) > \alpha(3)$ so no pruning
   - Explore 2nd child (9) -> Here pruning will occur.
     ∵ MIN node already has a value ≤ 6, it will never choose 9 and so we prune the node with value 9.
4. Max value = 6.

17. Explain WUMPUS world environment, giving its PEAS description. Explain how percept sequence is generated.

-> The wumpus world environment is a simple grid-based environment used in AI to study intelligent intelligent agent

behaviour in uncertain environments. It is a turn based environment where an agent must navigate a cave to find gold while avoiding hazards like pits and a monster called wumpus.

PEAS:

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

E: 4X4 grid world containing the agent, wumpus, pits, gold.

A: The agent can Move Forward, Left, Right, Shoot, Climb

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

Percept sequence generation:

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

Example percept sequence:

1. Agent starts at (1,1):
   - No breeze, no stench, no glitter → safe square.
2. Agent moves to (2,1):
   - Breeze detected → A pit is nearby but not in current square)
3. Agent moves to (1,2):
   - Stench detected → wumpus is in an adjacent cell.
4. Agent moves to (2,2):
   - Glitter detected → gold is here.
5. Agent moves back to (1,1) and climbs out.

18. Solve the following crypto-arithmetic problem:

1. SEND + MORE = MONEY.

Name:_____ Class:_____ Div:_____ Roll No:_____

Subject:_____ Topic:_____ Date:_____ Page No:_____

→ Step 1:
M must be 1. The sum of 2 four digit numbers cannot be more than 10,000.

```
  SEND
+ 1ORE
------
 1ONEY
```

Step 2:
Now S must be 8 because there is 1 carry over from column EON. O must be 0 (if S=8) and there is a 1 carried or (if S=9 and there is a 1 carried) But 1 is already taken, so O must be 0.

```
  SEND
+ 1ORE
------
 1ONEY
```

Step 3:
There cannot be a carry from column EON because any digit+0 <10, unless there is a carry from the column NRE and E=9. But this cannot be the case because then N would be 0 and 0 is already taken. So E<9 and there is no carry from this column. Therefore S=9 because 9+1=10.

Step 4:
case 1:
No carry : N+R = 10 + (N-1) = N+9
R = 9
But 9 is already taken so will ~~not~~ not work.
case 2:
Carry : N+R+1 = 9
R = 9-1 = 8. This must be the solution of R.

AASMAN

Step 5:
Let's consider E = 5 or 6.
E = 5,
then D = 7 and Y = 3. So this part will work but look at the column N8E. There is a carry from the column D5Y. N+8+1 = 16. But then N = 7 and 7 is taken by D therefore E = 5

```
  95ND
+ 1085
------
 10N5Y
```

Now,
N+8+1 = 15, N = 6

```
  956D
+ 1085
------
 1065Y
```

Step 6:
The digits left are 7, 4, 3 and 2. we know there is carry from column D5Y, so only pair that works is D = 7 and Y = 2

```
  9567
+ 1085
------
 10652
```

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

-> (1) Represent these axioms in first order predicate logic.

Name : _____ Class & Div.: _____ Page No.: _____

Roll No.: _____ Subject : _____ Topic : _____ Date : _____

We define the following predicates :

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

Translating axiom into predicate logic :

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

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

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

(2) Convert each formula to clause form :

1. Convert implications to clausal form
   $\forall x \, (G(x) \rightarrow H(x))$
   - Using implication removal :
     $\forall x \, (\neg G(x) \vee H(x))$
   - In clause form :
     $\{ \neg G(x), H(x) \}$

2. $\forall x \, (H(x) \rightarrow S(x))$
   - Using implication removal :
     $\forall x \, (\neg H(x) \vee S(x))$
   - In clause form :
     $\{ \neg H(x), S(x) \}$

3. $\exists x \, G(x)$

· In clause form : { G (a) }

(3) Prove "is someone smiling ?" using resolution

1. collect all clauses
   (1) { ¬ G (x), H (x) }
   (2) { ¬ H (x), S (x) }
   (3) { G (a) }

2. Apply resolution
   · Resolve (1) { ¬ G (x), H (x) } with (3) { G (a) } :
      · Substituting x = a :
        { ¬ G (a), H (a) }
      · ∵ we have G (a), resolving gives :
        { H (a) }
   · Resolve (2) { ¬ H (x), S (x) } with { H (a) } :
      · Substituting x = a :
        { ¬ H (a), S (a) }
      · Since we have H (a) resolving gives :
        { S (a) }

Since we have derived S(a) we conclude that someone (a) is smiling.

20. → Explain modus ponen with suitable example.
Modus ponen is a fundamental rule of inference in propositional logic that allows us to deduce a conclusion from a conditional statement and its antecedent.

Name : _____ Class & Div.: _____ Page No.: _____

Roll No.: _____ Subject : _____ Topic : _____ Date : _____

It follows the form:

1. $P \rightarrow Q$ (if P then Q)
2. P (P is true)

$\therefore$ Q (Q must be true)

Example:

1. ~~It~~ If it rains, the ground will be wet. $\rightarrow P \rightarrow Q$
2. It is raining $\rightarrow P$

$\therefore$ ~~Ground~~ ground is wet $\rightarrow Q$.

21. Explain forward chaining and backward chaining algorithm with the help of example.

→ Forward chaining: It starts with given facts and applies inference rules to derive new facts until the goal is reached. It is a data driven to approach because it begins with known data and works forwards to ~~reach~~ reach a conclusion.

Example: Diagnosing a disease.

Rules:

1. If a person has a fever and cough they might have flu.
2. If a person has a sore throat and fever, they might have cold.

Facts:

- The patient has a fever
- The patient has cough

Inference:

1. Fever + cough → flu (rule 1 applies)
2. Conclusion: The patient might have flu.

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

Example: Diagnosing a disease

Goal: Determine if patient has flu.

Rules:
1. (Fever ∧ cough) → flu.
2. (Sore Throat ∧ Fever) → Cold.

Process using backward chaining:
1. We want to prove flu.
2. Looking at rule 1: (Fever ∧ cough) → Flu, we need to check if patient has fever and cough.
3. We check our known facts:
   - Patient has fever
   - Patient has cough
4. Since both conditions are met, we confirm that flu is true.

Nayaab Jindani D15C Roll No.20

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
2. Find the Median
3. Find the Mode
4. Find the Interquartile range

Ans.

1. Find the mean:

To find the mean we use the following formula:

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

=1611/20

=80.55

2. Find the median:

To find median, first we will sort the data in ascending order

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

Since n here is even we will find the average of the 2 numbers at middle position (10th and 11th position)

Median=(81 + 82) / 2 = 81.5

3. Find the mode:

mode is the value that appears most frequently in a dataset.

In the given dataset 76 appears 3 times,which is the most frequent and so mode is 76.

4. Find the interquartile range:

Q1 = median of first 10 = (76 + 76) / 2 = 76

Q3 = median of last 10 = (88 + 90) / 2 = 89

IQR = Q3 - Q1 = 89 - 76 = 13

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

1. For each tool listed above

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

Ans.

Target audience:

Machine learning for kids:
School students, teachers and educators introducing machine learning concepts

Teachable machine:
General users, developers, students with little/no coding background.

Use by target audience:

Machine learning for kids:
Students can build machine learning models (text, image, or number-based) through a drag-and-drop interface and also use pretrained models. Teachers can use it in classroom activities to teach AI/ML fundamentals in a simple, visual way.

Teachable machine:
Users can train ML models using webcam, microphone, or file uploads (e.g., image, sound, pose). It's often used in interactive projects, demos, and classroom activities.

Tool's benefits and drawbacks:

Machine learning for kids:
Benefits:

- Easy to use, no prior coding experience needed
- Educational and aligned with school curriculum
- Supports various data types: text, images, numbers
- Encourages creativity through Scratch integration

Drawbacks:

- Limited scalability and complexity
- Not suitable for advanced ML concepts
- Mostly focused on learning rather than production use

Teachable machine:
Benefits:

- No coding required
- Fast, intuitive, web-based interface
- Supports real-time model training with live data
- Easy export to TensorFlow.js and other formats

Drawbacks:

- Not suitable for complex datasets or large-scale projects
- Performance depends on browser and hardware

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

- Predictive analytic
- Descriptive analytic

Ans. Both Machine Learning for Kids and Teachable Machine are examples of predictive analytics tools. They are designed to train models on labeled data and use those models to make predictions on new inputs. In Machine Learning for Kids, users provide examples of data with labels and the tool learns to predict similar outcomes. Similarly, Teachable Machine allows users to train models using images, sounds, or poses, and predicts the class of new inputs in real time. These tools are not used to summarize or analyze past data (which would be descriptive analytics), but rather to make forward-looking predictions based on training data. Hence, both are best categorized under predictive analytic.

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

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Ans. Both Machine Learning for Kids and Teachable Machine are examples of supervised learning. In both tools, users provide labeled training data, which allows the model to learn the relationship between inputs and outputs. For example, a student might label text as "positive" or "negative" in Machine Learning for Kids, or label images as different classes in Teachable Machine. The model then learns to classify new inputs based on this training. Since both tools rely on labeled datasets, they fall under the supervised learning category.

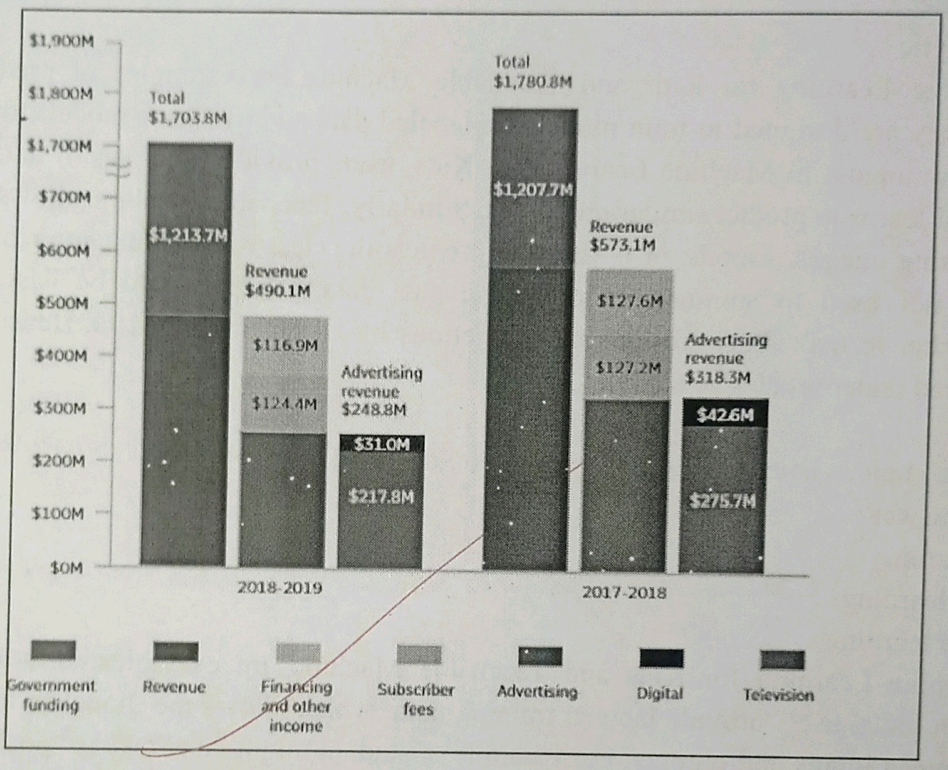
Q.3 Data Visualization: Read the following two short articles:

- 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.

Misleading CBC(Canadian national broadcast company) Funding Bar Chart:

In April 2023, a bar chart from the Canadian Broadcasting Corporation's (CBC) 2018–2019 Annual Report resurfaced online and quickly drew widespread criticism for its misleading design. The chart was intended to show CBC's funding sources, including government appropriations and advertising revenue. However, it sparked controversy after viewers noticed serious issues with how the data was presented.



The axis had a sudden break—jumping from $700M to $1.7B—causing a $490M revenue bar to appear visually larger than the $1.2B government funding bar. This misleading scale made it seem like television revenue equaled or exceeded government funding, which is factually incorrect. Additionally, the chart layout was flawed: revenue and advertising revenue were shown as separate bars rather than subdivisions of the total income, creating further confusion.

Sources:

https://thedeepdive.ca/cbc-commits-a-chart-crime-in-representing-its-funding-and-revenue-sources/

https://www.codeconquest.com/blog/12-bad-data-visualization-examples-explained/htoc-bad-data-visualization-example-2

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

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

```
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

SMOTE (Synthetic Minority Over-sampling Technique) is used to fix class imbalance by creating synthetic samples for the minority class

In the above given line of code,

fit_resample() applies SMOTE to the training set which balances the class distribution.
X_train_res, y_train_res now contain:
Original majority class samples and Synthetic minority class samples

Step 6. Feature scaling

```
scaler = StandardScaler()
X_train_res = scaler.fit_transform(X_train_res)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Feature scaling standardizes the values of your features so they all have: Mean = 0 and Standard Deviation = 1

In the above code,

We fit the scaler on the training data, then transform the training data using those values.Finally we transform validation and test data using the same scaler.

Step 7. Training the Naive Bayes model

```
model = GaussianNB()
model.fit(X_train_res, y_train_res)
```

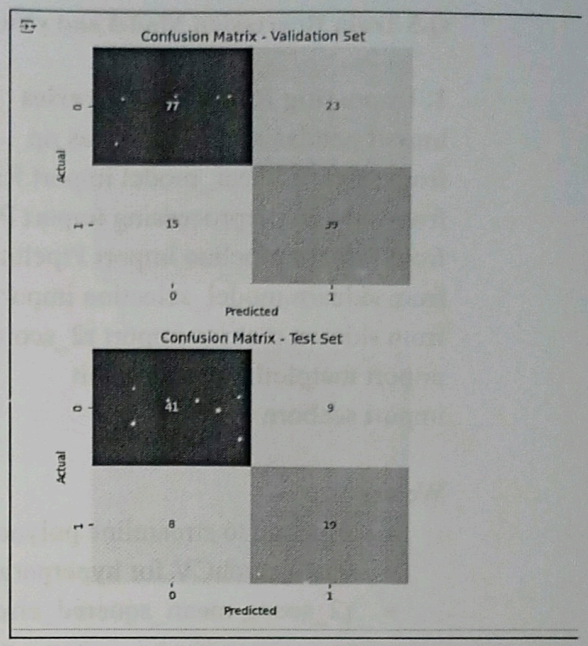
```
GaussianNB
GaussianNB()
```

This creates a Naive Bayes model using the GaussianNB class.
GaussianNB is used when the features are continuous and assumed to follow a normal (Gaussian) distribution — which is common for numeric data like BMI, glucose, etc.

Step 8. Evaluating the model

```
# On Validation Set
y_val_pred = model.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))

# On Test Set
y_test_pred = model.predict(X_test)
print("\nTest Accuracy:", accuracy_score(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
```

```
Validation Accuracy: 0.7532467532467533
              precision    recall  f1-score   support

           0       0.84      0.77      0.80       100
           1       0.63      0.72      0.67        54

    accuracy                           0.75       154
   macro avg       0.73      0.75      0.74       154
weighted avg       0.76      0.75      0.76       154

Confusion Matrix:
 [[77 23]
 [15 39]]

Test Accuracy: 0.7792207792207793
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        50
           1       0.68      0.70      0.69        27

    accuracy                           0.78        77
   macro avg       0.76      0.76      0.76        77
weighted avg       0.78      0.78      0.78        77

Confusion Matrix:
 [[41  9]
 [ 8 19]]
```



Validation Accuracy: 75.3%
Test Accuracy: 77.9%
This shows good generalization and consistent performance.

Recall for Class 1 (Diabetic):
72% (Validation), 70% (Test)
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

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%
20data%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)

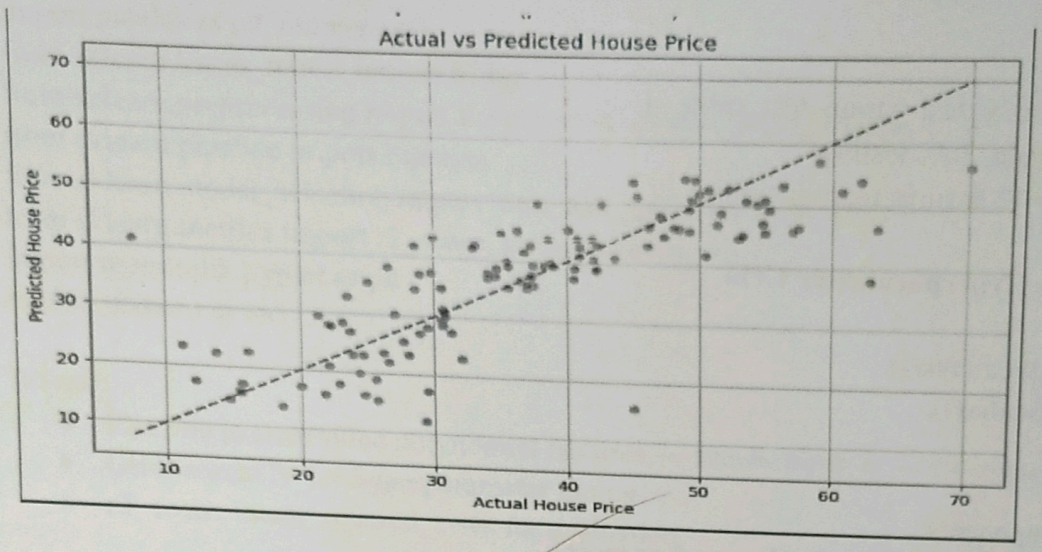
y_test_actual, y_pred = reg_model.evaluate(best_model, X_test, y_test)

- 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

Ans.

The Wine Quality dataset from Kaggle includes several key physicochemical features that influence the quality of wine, such as

fixed acidity,

volatile acidity,
citric acid,
residual sugar,
chlorides,
free and total sulfur dioxide,
density,
pH,
sulphates,
alcohol.

Each of these features plays a role in determining the taste, stability, and preservation of the wine. For instance, high levels of volatile acidity often lower the quality, while alcohol content and sulphates generally show a positive correlation with better wine ratings.

During the feature engineering process, handling missing data is crucial for model performance. Although the original dataset is relatively clean, if missing values are present, the following techniques can be used:

1. Remove Missing Data:
Delete rows or columns with missing values.
Best when only a small portion of data is missing.

2. Fill with Mean:
Replace missing values with the column's average.
Works well for normally distributed numerical data.

3. Fill with Median:
Use the middle value of the column for imputation.
Better than mean when data has outliers or is skewed.

5. Fill with Constant:
Replace missing data with a fixed value like 0 or "unknown."
Helps retain rows while signaling missingness.

Advantages and disadvantages of different imputation techniques

1. Mean Imputation

Advantage: Simple to implement and works well with normally distributed data.
Disadvantage: Sensitive to outliers and may distort the overall variance.

2. Median Imputation

Advantage: More robust than mean; not affected by outliers.
Disadvantage: Does not consider relationships between different features.

3. Mode Imputation

Advantage: Ideal for categorical data; retains the most common category.
Disadvantage: Can lead to overrepresentation of one category.

4. Constant Value Imputation

Advantage: Easy to implement and helps highlight missing data.
Disadvantage: May introduce meaningless or misleading values.