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

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

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

Normalizing data: Scales values between 0 and 1.

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

```
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

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

Theory:

We first mounted Google Drive to the Colab environment to access files stored in the 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')
```

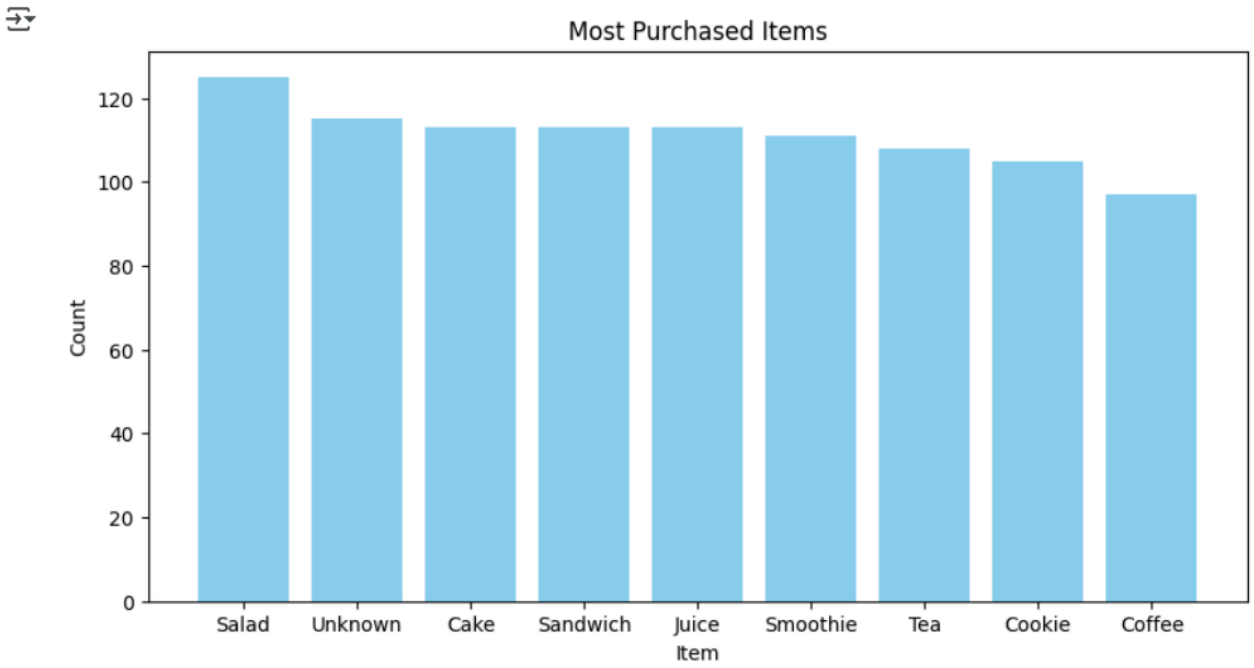
```
Mounted at /content/drive
```

```
[ ]  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.

```python
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

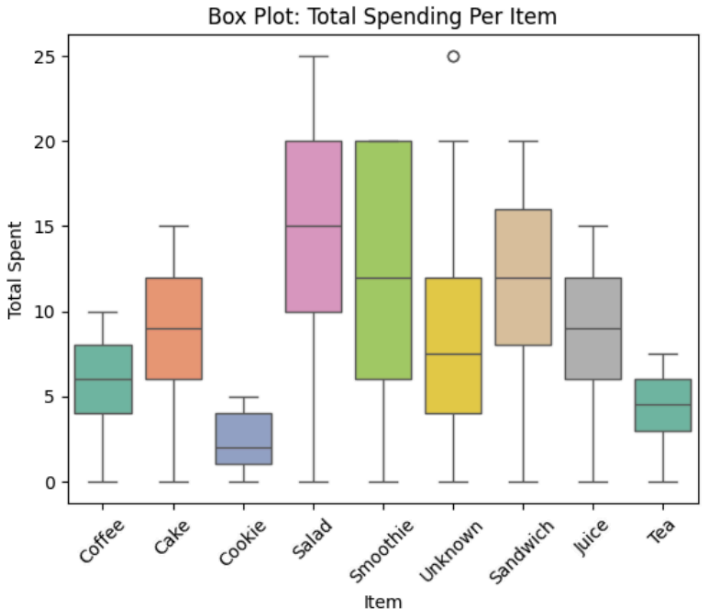
A contingency table in Python can be created using **pandas**. The code typically starts by importing pandas and loading the dataset. The pd.crosstab() function is then used to generate the table, where one categorical variable (e.g., "Item") is placed as rows and another (e.g., "Payment Method") as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

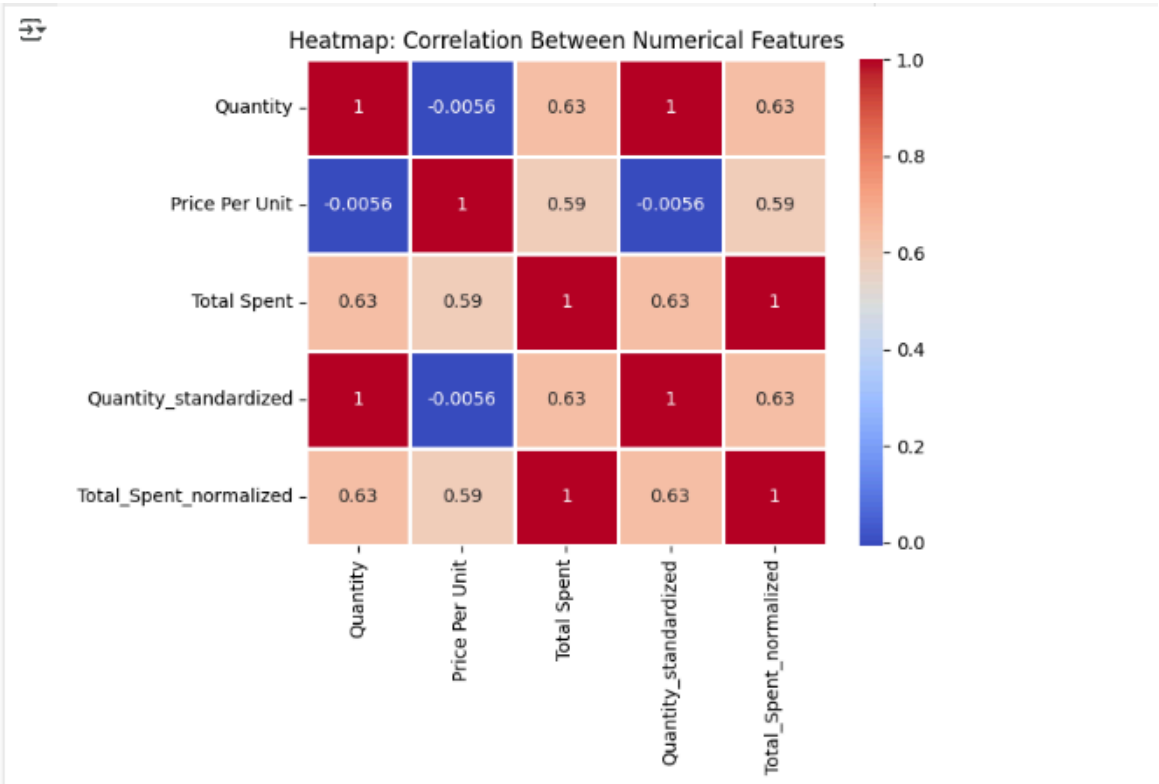
```
sns.boxplot(x=df_subset['Item'], y=df_subset['Total Spent'], palette="Set2")
```



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.

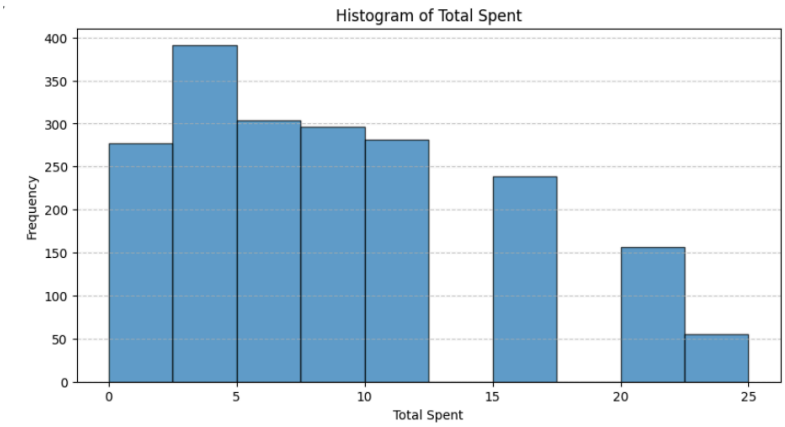
```
sns.heatmap(df_subset.corr(numeric_only=True), annot=True, cmap="coolwarm", linewidths=1)
plt.title("Heatmap: Correlation Between Numerical Features")
plt.show()
```



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).

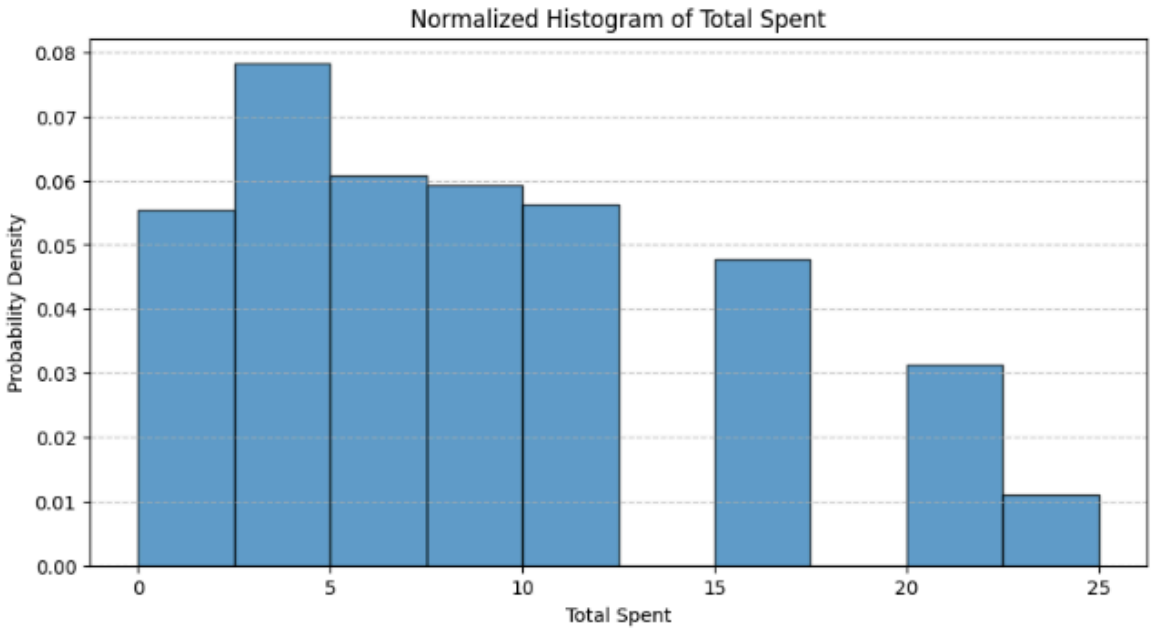
```
column_name = "Total Spent"

# 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.

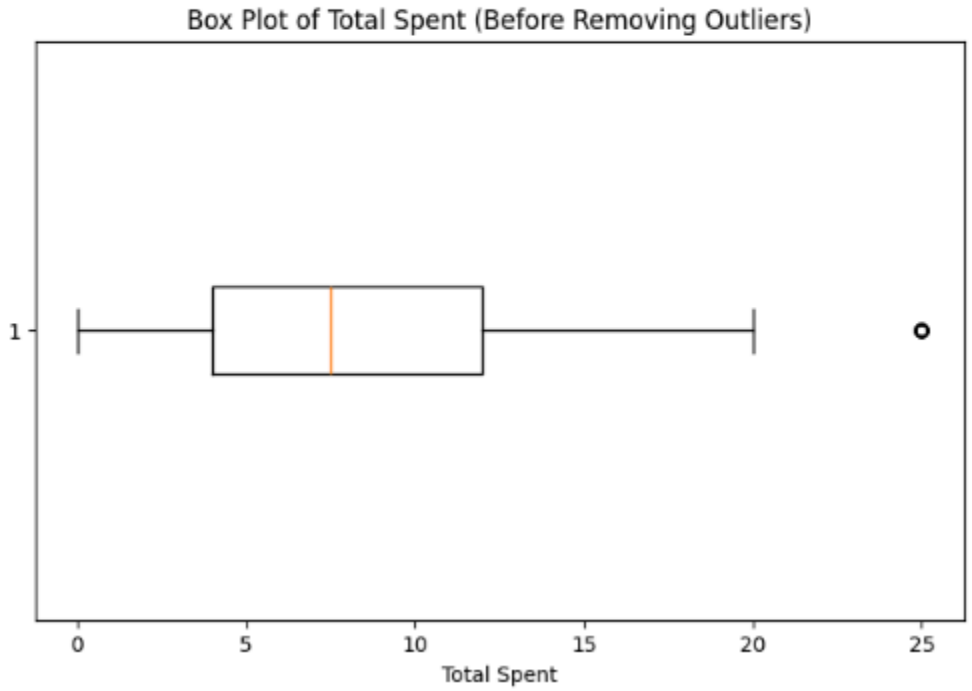
```
plt.figure(figsize=(10, 5))
plt.hist(df[column_name], bins=10, edgecolor='black', alpha=0.7, density=True)
plt.xlabel(column_name)
plt.ylabel("Probability Density")
plt.title(f"Normalized Histogram of {column_name}")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



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()
```

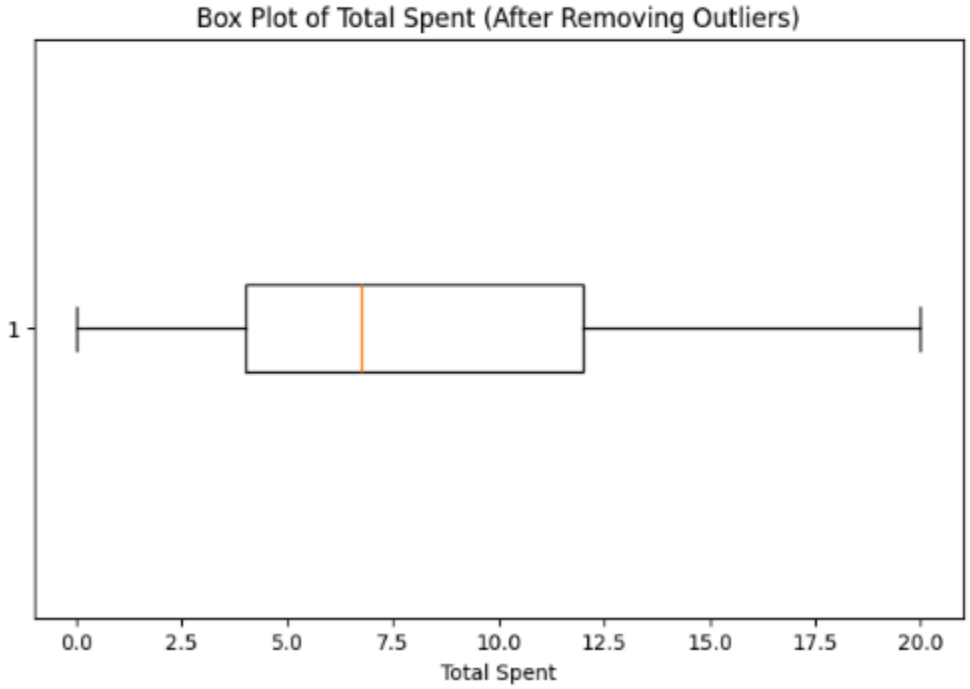


8. Box plot (after removing outlier)

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.

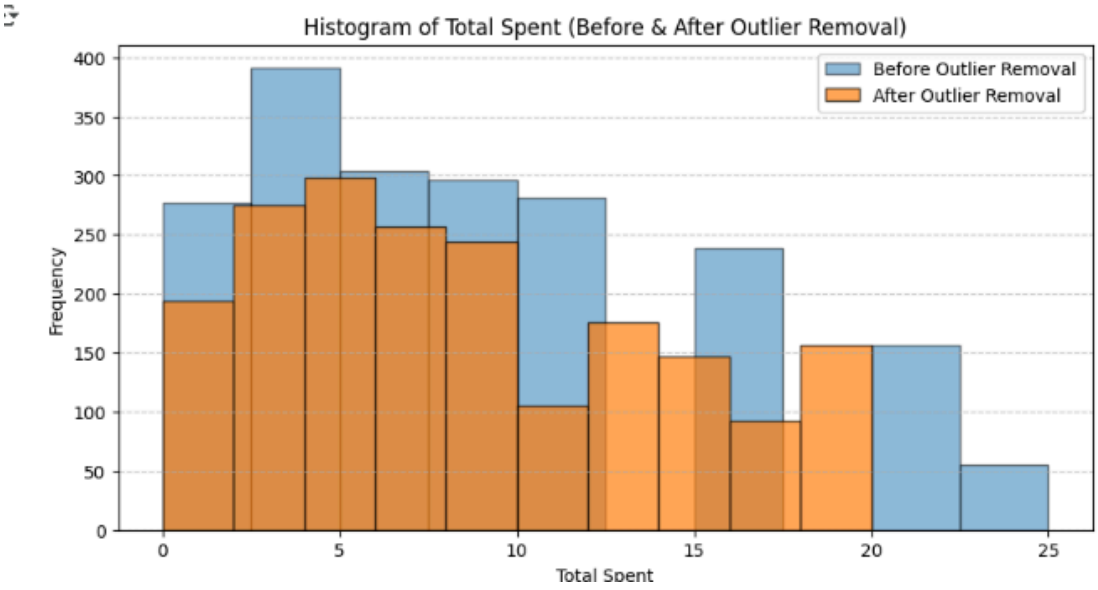
```python
# Calculate IQR (Interquartile Range)
Q1 = df[column_name].quantile(0.25)  # First quartile (25th percentile)
Q3 = df[column_name].quantile(0.75)  # Third quartile (75th percentile)
IQR = Q3 - Q1  # Interquartile Range

# 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)]
```

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



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

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



Conclusion:

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

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.

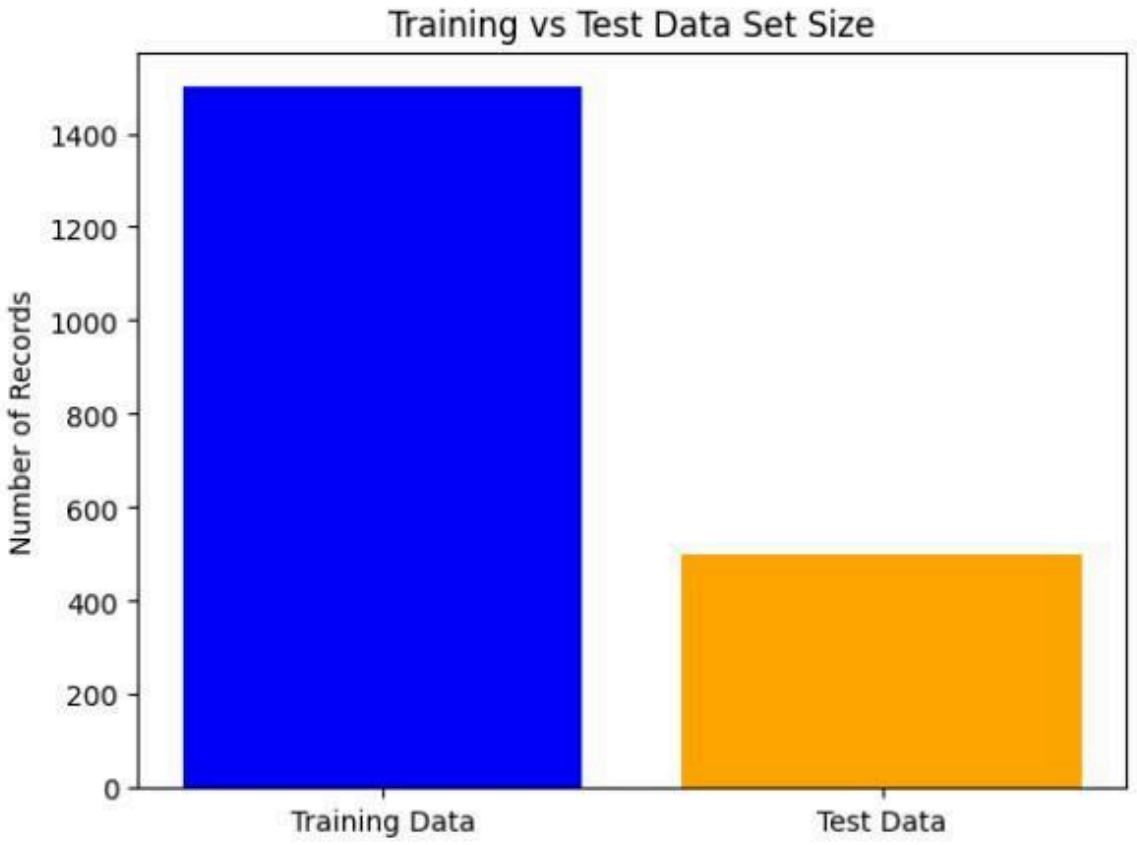
2) Use a bar graph and other relevant graphs to confirm your proportions.

   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

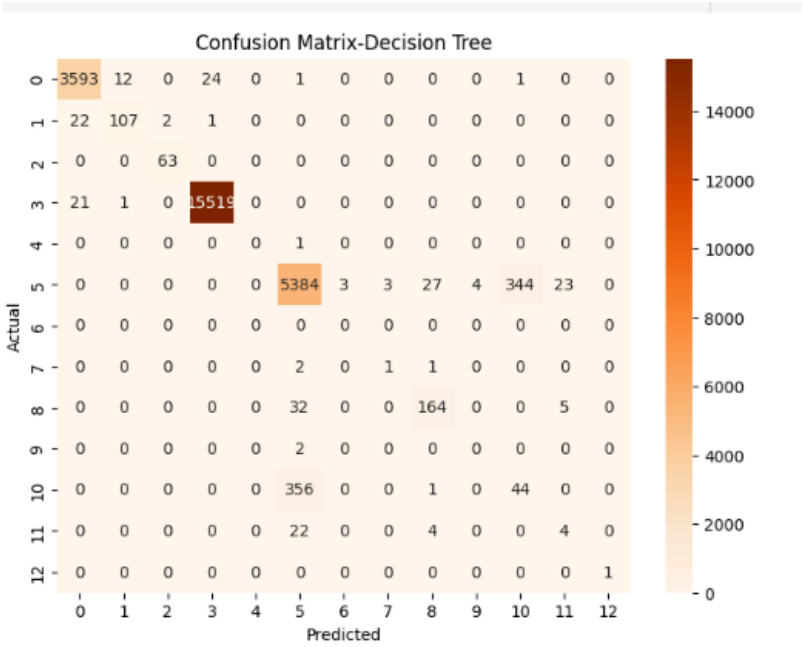
```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train,)

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

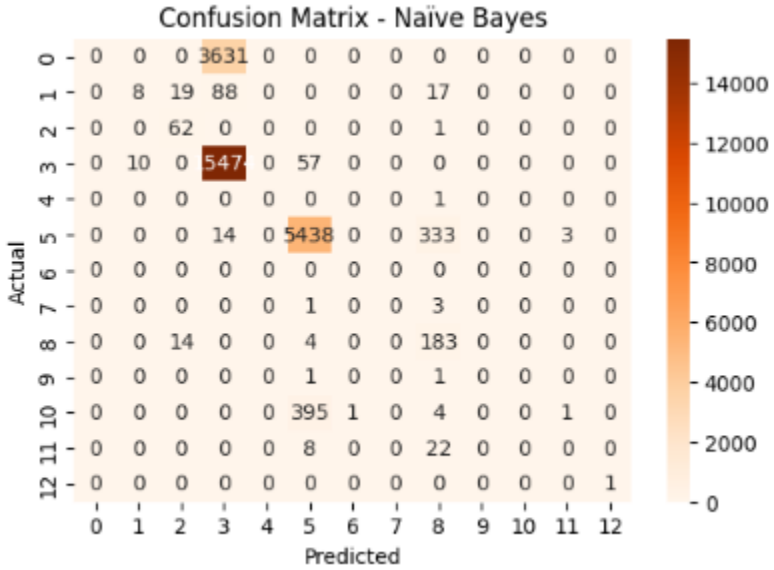
```
print("Naïve Bayes Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Classification Report:\n", classification_report(y_test, y_pred_nb))
```

```
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.

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:

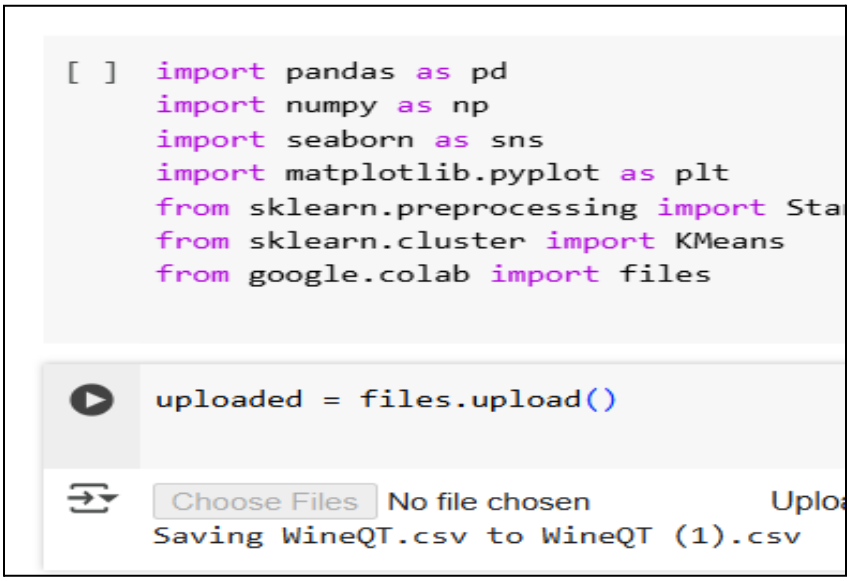
For this experiment we have used the Wine Quality dataset which contains physicochemical properties of red wine samples along with a quality score rated by experts.

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)

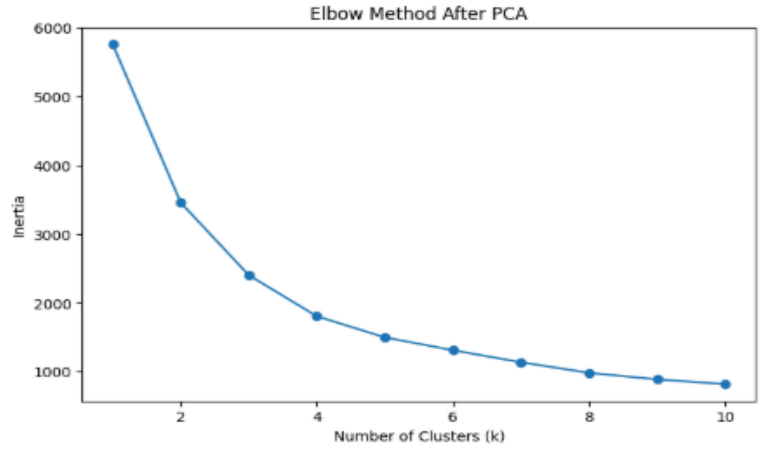
```
from sklearn.decomposition import PCA

# 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)

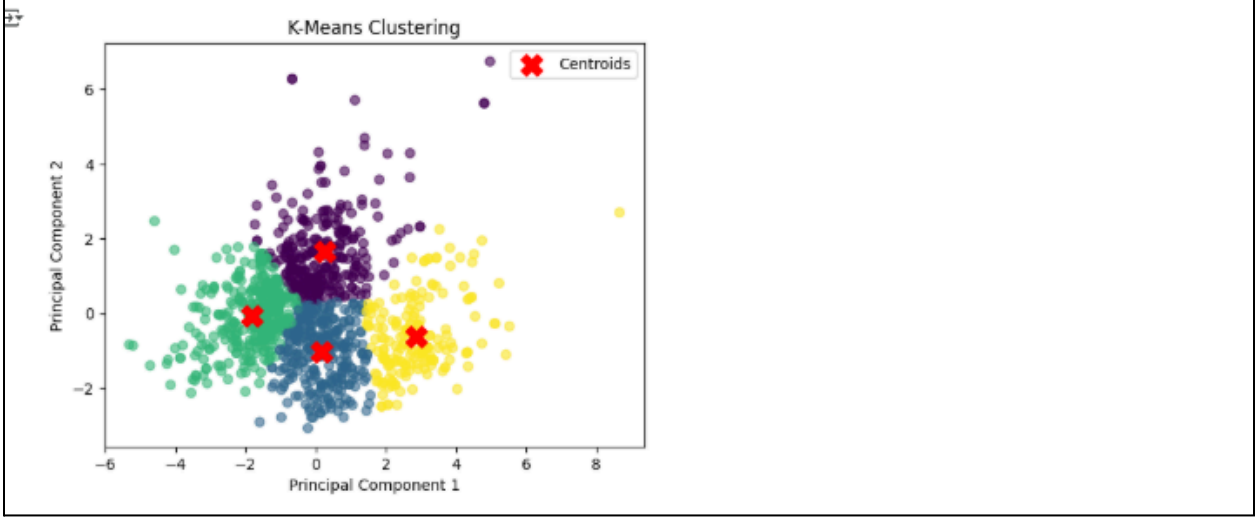
We select the "elbow point" – where the decrease in WCSS slows down – as the best k.

```
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:

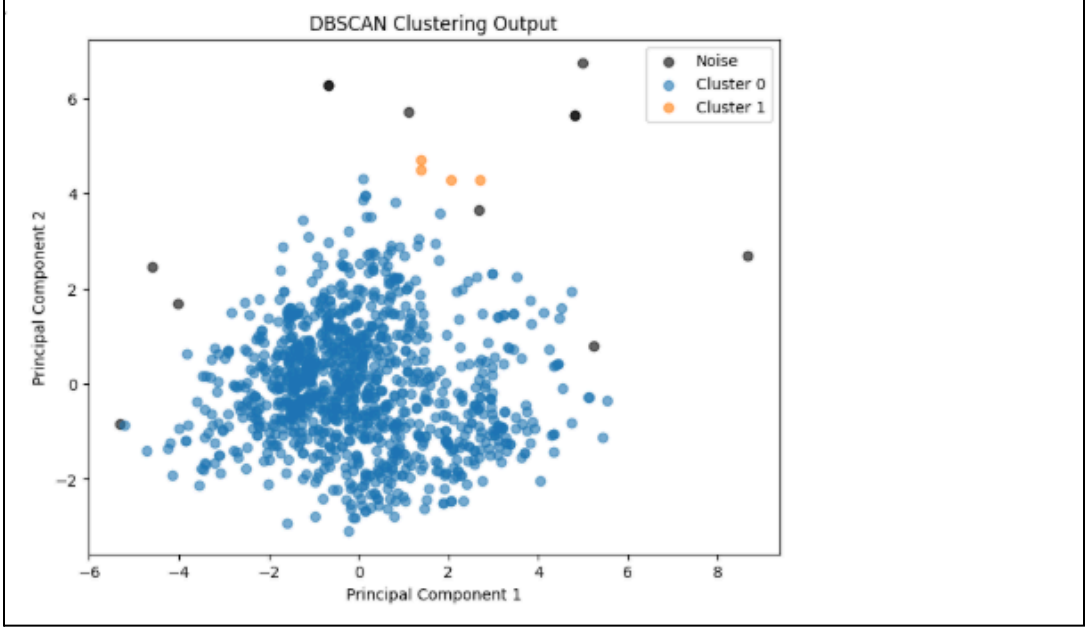
```
from sklearn.metrics import davies_bouldin_score

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()
```



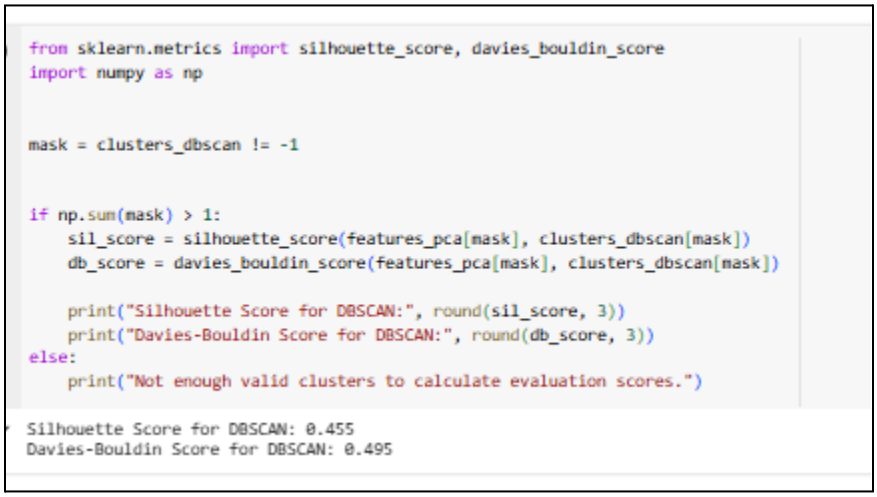
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was applied on the PCA-reduced features to discover clusters based on density. The parameters used were eps=0.8 and min_samples=5. The scatter plot shows several clusters formed with some points labeled as noise (black), indicating 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:

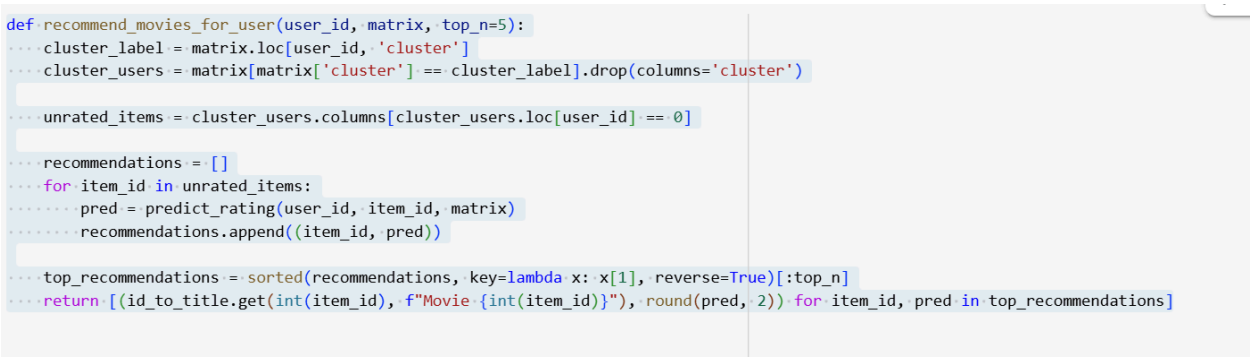
- **User clustering**
- **Cosine similarity**
- **Weighted average of ratings from similar users in the same cluster**

```
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 No: 9

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

Theory:

1. What is Apache Spark and how it works?

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

Key Features:

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

How It Works:

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

2. How is data exploration done in Apache Spark?

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

Steps for EDA in Apache Spark:

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

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

Conclusion:

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

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

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

Experiment No: 10

Aim:

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

Theory:

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

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

Batch Data:

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

Stream Data:

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

2. How Data Streaming Takes Place Using Apache Spark?

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

How It Works:

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

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

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

Key Components:

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

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

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

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

Use Cases:

- Real-time fraud detection.

- Social media sentiment analysis.

- Log processing.

- Monitoring systems and alerts.

Conclusion:

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

Nikita Chhabria
Roll no. 07
D15C

AIDS Assignment 1

04/05

Q1 What is AI?

→ Artifical Intelligence refers to the simulation of human intelligence in machines AI systems can perform tasks such as learning, reasoning problem solving, perception and language understanding.

AI is categorised into various types - including narrow & general AI

Narrow AI : specialised for specific tasks
example :- Siri, Google Assistant

general AI : Hypothetical AI capable of human like reasoning across domains

Q2 What are AI agents terminology, explain with example.

→ An AI agent is an entity that provides perceives its environment through sensors & acts upon its perception action cycle to achieve a goal.

AI agents can be classified as:

1. Simple Reflex Agents : Reacts to current percepts
2. Model Based Agents : Maintain internal model

Q2

(3) Goal Based Agents : Maintains internal models

(4) Utility based :- Maximizes utility functions

The example of each of the different agents is that :

Simple Reflex Agents (Rule-Based)

Thermostat :- Turns on/off heat based on temperature

Model-Based Reflex Agents

Self driving cars : Use sensor data to track obstacles & road conditions

Utility-Based Agents

Stock marketing bots : Optimize for maximum profit while managing risk

Learning Agent :

Recommendation System :- Learn from preference

Q1 AI plays a crucial role in various aspects of healthcare & daily life, such as:

- Early detection & diagnosis : AI-based models detected and analyzed patterns
- Medical Research & Drug Discovery : It discovers about potential treatment
- Healthcare automation : Supply chain automation & remote learning are few other ways AI helped during the pandemic

Q2 How AI agent used to solve 8 puzzle problem?

→ The 8-puzzle problem is a sliding puzzle consisting of 3×3 grid with eight tiles and one empty space. AI technique used to solve it include :

- Breadth First search : Explores all possible moves level by level
- Depth First search : Explores all deeper path first
- A* Algorithm : Uses a heuristic function to find optimal path

Nikita Chhabria
Roll no. 07
DISC

04/08

output, synthetic
preference, genre data

Autolander

landing precision, passenger safety
Runway wind, attitude
Flaps, engines, landing gear
Altimeter, gyroscope, GPS

Sorting Iyer

read detection, accuracy
safety area, disorder
movement
cameras, motion sensors

shopping bot for an offline Bookstore

ability : Partially Observable
ion : Stochastic
: Sequential
Dynamic : Dynamic
ontinous : Discrete
Multi-Agent : Multi Agent

- Greedy Best First Search :- Priorities nodes based on heuristics without considering past cost

Q 4 The PEAS framework stands for Performance measure, Environment, Actuators & Sensors, it is used to define the components of an AI agent by specifying how it interacts with its environment & evaluates its success

- Taxi Driver

Performance :- Safety, speed, passenger comfort
Environment :- Roads, passengers, traffic
Actuator :- steering, acceleration, brakes
Sensors :- GPS, cameras, speedometer

- Medical Diagnosis System

P :- Diagnosis accuracy, treatment success
E :- Patients, Symptoms, Diseases
A :- Recommendations, Prescriptions
S :- Patient history, lab tests

- Music Composer AI

P :- Harmony, Originality
E :- Musical notes, instruments

A :- MIPI output, synthesis

S :- User preference, genre data

- Aircraft Autolander

P :- landing precision, passenger safety

E :- Runway wind, attitude

A :- Flaps, engines, landing gear

S :- Altimeter, gyroscope, GPS

- Robotic Sentry Gun

P :- Threat detection, accuracy

E :- security area, disorder

A :- gun movement

S :- Cameras, motion sensors

Q 5 A shopping bot for an offline Bookstore

Observability : Partially Observable

Determinism : Stochastic

Episodic : Sequential

Static / Dynamic : Dynamic

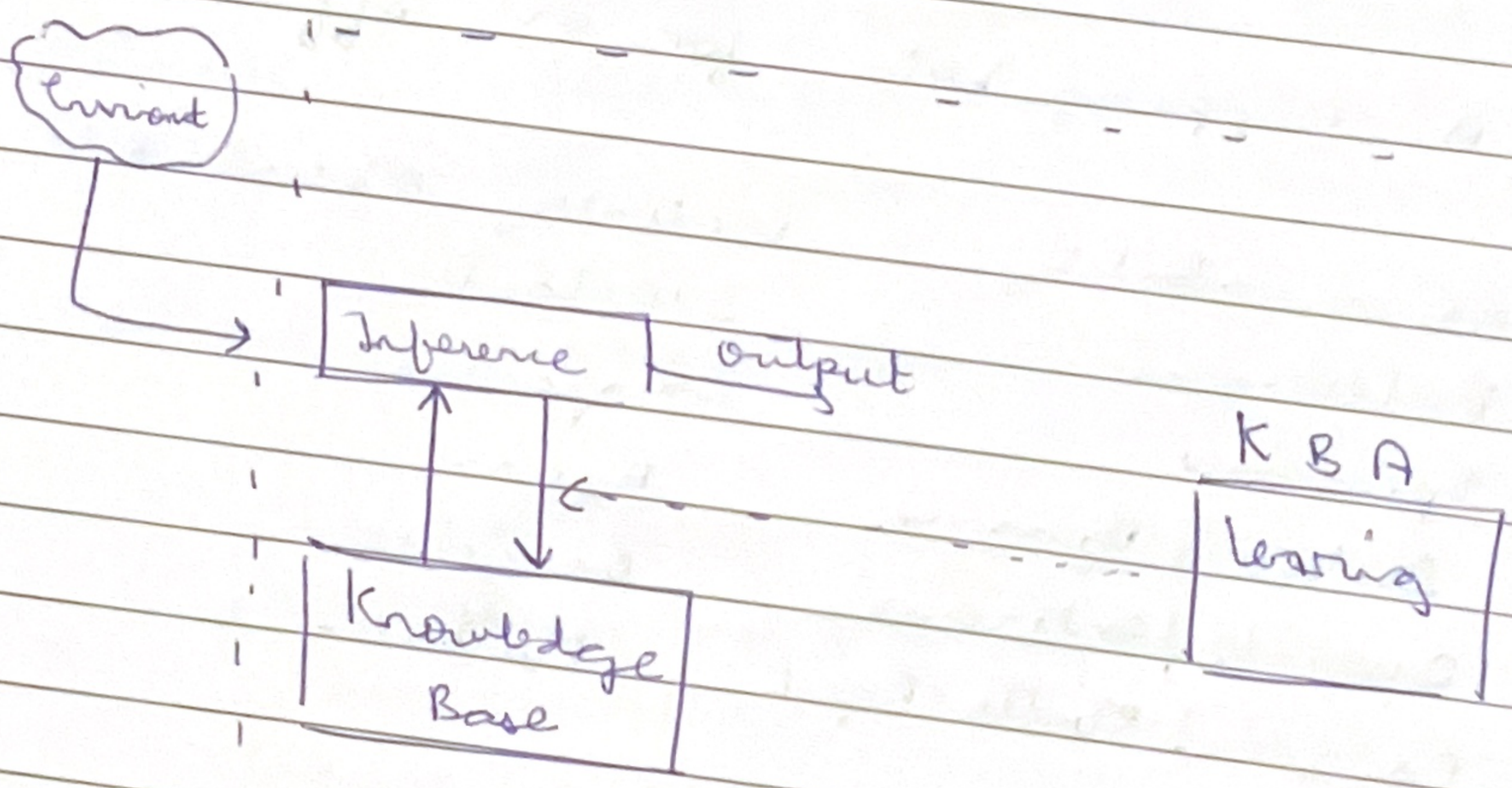
Discrete / Continuous : Discrete

Single / Multi-Agent : Multi Agent

Q1

Feature	Model Based Agent	Utility Based
Internal Model	Maintains an internal model of the world	Evaluates actions based on utility
Decision Making	Uses past state to make decision	Selects action to maximize utility
Example	Self driving cars Car Navigating through traffic	AI recommending personalized products

Q2 Architecture of knowledge based & learning Agents



A knowledge Based Agent uses stored knowledge to make decisions and reason about the environment It's architecture consists of

(1) Knowledge Base :- Stores knowledge to make decisions and reason about the environment Its architecture consists of :-

(1) Knowledge Base (KB) : Stores facts, rules and reason about in a structured format

(2) Inference Engine : Applies logical reasoning to derive conclusion from the Knowledge base

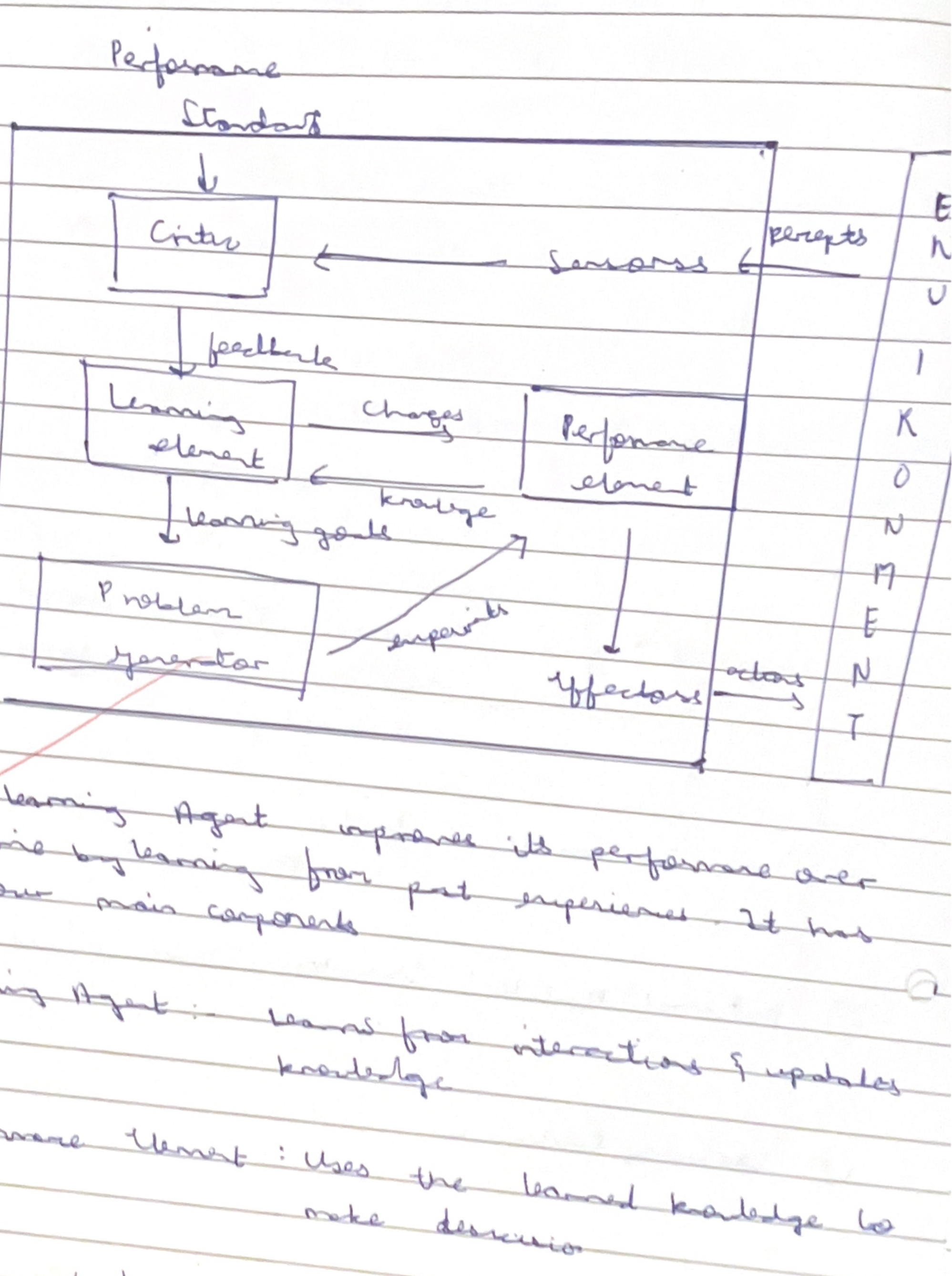
(3) Perception Module : Gather inputs from sensors or external sources

(4) Action Module : Executes action based on derived conclusions

(5) Learning Mechanism : Updates the knowledge base with new information

Example :- A medical diagnosis system that stores diseases, symptoms & applies logical rules

Q 7 Architecture of learning Agent



A learning Agent improves its performance over time by learning from past experiences. It has four main components

- learning Agent :- learns from interactions & updates knowledge
- Performance Element : Uses the learned knowledge to make decisions
- Critic : evaluates the agent's performance by comparing actions with desired outcomes

Predicate logic conversion

a) Anita travels by car if available, otherwise, travels by bus

∀ Available (car) => Travels (Anita, car)
¬ Available (car) -> Travels (Anita, Bus)

(b) Bus goes via Andheri & Goregaon

Routes (Bus, Andheri)
Route (Bus, Goregaon)

(c) Car has a puncture, so it is not available

Puncture (car) => ¬ Available (car)

Forward Reasoning

~~From (1) ¬ Available (car)~~

① Has Puncture (car) → ¬ car Available (car)

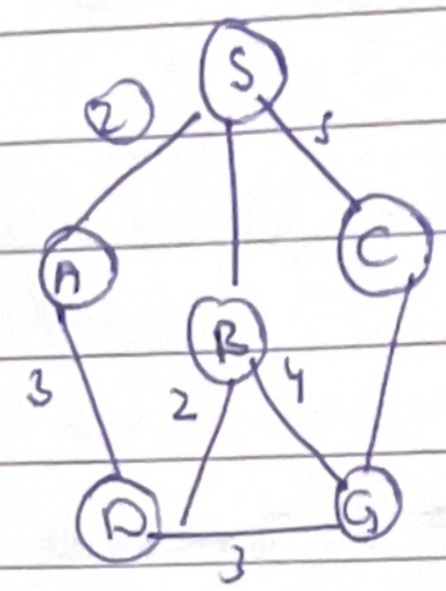
② ¬ car Available (car) → Travels (Anita, Bus)

③ Travels (Anita, Bus) [from 1 & 2]

④ Bus goes via Goregaon [from b]

⑤ Travels (Anita, Goregaon)
Yes, Anita will travel via Goregaon

q 9



S → A cost (2) A → G (cost 3)
S → B Cost 5 C → G (cost 4)
S → C cost 5
A → D cost 3

BFS Traversal from S to G

① Start at S → Queue [S]
② Expand S → Queue [A, B, C]
③ Expand B (no new nodes) → Queue [C, G]
④ Expand C → Queue [D, G, G]
⑤ Expand D → Queue [G, G]
⑥ Expand G → Goal Found

Shortest Path S to G is:

S → A → G

Total Cost : 2 + 3 = 5

q 10 Interative Deepening Search (IDS)

It combines the space efficiency of DFS & completeness of BFS by repeatedly running DLS with increasing depth limit

Advantage: guarantees finding the shortest path while using less memory than BFS

Q12 Hill Climbing Algorithm is a local search algorithm that continously moves towards the best neighboring state with a higher heuristic value, aiming to reach a global optimum

Steps

(1) Start from an intial state
(2) Evaluate neighboring states and moves to the one with the highest heuristic value
(3) Repeat until no better neighbor

Example :-

Imagine a mountain climbing scenario where a hiker moves uphill based on the steepest slope. If they reach a peak that is not the highest (global maximum), they might get stuck

Q14 Drawbacks of Hill Climbing

- Local Maximum
  The Algorithm may stop at peak that not the global optimum

- Plateau Problem :
  A flat region where all neighbouring stat have the same heuristic value, causing search to halt

- No Backtracking
  Once it moves forward, it cannot recover from a bad decision

= Limitation of Steepest - Ascent Hill Climbing

Steepest - Ascent Hill Climbing selects the best posible move in each step but has extra limitations

① Slow progress in Narrow Ridges
② More likely to get Stuck in Local Maxima
③ Ineffient in large search space

Solutions :

Random Restart
Simulated Annealing

Simulated Annealing is a probabilistic search technique that avoids getting trapped in local optima by allowing occasional bad moves

Steps :

1) Start with random solution
2) Evaluate its cost
3) Move to a neighboring solution
4) Accept it if it better, otherwise, accept it with probability decreasing over time
5) Reduce "temperature" gradually
6) Repeat with convergence

Algorithm :

Initialize temperature T, cooling rate α.
Choose an initial solution S
Repeat until $T \to 0$:
a. Select a neighbouring solution S
b. Compute cost diff $DE = cost(S') - cost(S)$
c. If $DE < 0$, move to S
d. Else, move to S' with prob $e^{(-DE/T)}$
e. Reduce T: $T = \alpha * T$

Return best solution found

Ex. Travelling Salesman Problem where a delivery truck must find shortest route

Q.14 A* is an informed search Algorithm that uses

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

where $g(n)$ is the cost from start to node n & $h(n)$ is heuristic

Ex: Finding shortest path in a graph like Google Maps

It is widely used in pathfinding, AI & game development

Uses both $g(n)$ & $h(n)$

A* is optimal

A* is complete

Q.15 The Minimax Algorithm is used in adversarial search (eg. two-player games) to determine the optimal move by assuming both players play optimally

Maximizer (eg. X) tries to get the highest score

Minimizer (eg. O) tries to reduce the score

Algorithm

(1) Generate the game tree upto depth limit

(2) Assign heuristic values to leaf node

(3) Backpropagate values:

• Maximizer picks the maximum value

- Maximiser picks the maximum value e

The root node selects the best move based on propagated

Game tree: A minimax tree for Tic-Tac-Toe would show all possible board states, making the best move at each step

Alpha-Beta Pruning Optimises minimax by skipping unnecessary branches, reducing computation. It uses two values

Alpha (α): Best score maximiser can achieve
Beta (β): Best score Minimiser can achieve

Algorithm:

1) Perform Minimax search
2) Prune branches where:
   if Maximiser best (α) ≥ Minimiser best (β), further evaluation is skipped
   if Minimiser's best (β) ≤ Maximiser's best (α)
3) This reduces time complexity without affecting the final decision

Example: In a game tree, if a branch has already yielded a worse outcome than the best known

Q16 move, it is ignored to save time

Q17

The WUMPUS world is a grid-based AI environment where an agent explores a cave while avoiding hazards like pits & the wumpus monster

PEAS descriptor

Performance Measure: Reaching the gold safely, minimizing steps

Environment: A 4 x 4 grid with pits, gold & wumpus

Actuators: Move Forward, turn, grab, shoot, climb

Sensors: Perceive stench, breeze, glitter

Percept sequence Generation:

1) Agent start at (1, 1), sensing its surroundings

2) If stench is detected, the wumpus is nearby

3) If breeze is detected, a pit is nearby

4) The agent infers safe paths & navigates towards the gold while avoiding hazards

```
  7  5  N  D
  1  0  R  5
-------------
1  0  N  5  2
```

As E = 5 the sum of N K should be a carry

∴ N = 6 , R = 8

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

Assigning the remaining value to D

∴ D = 7

```
  9  5  6  7
  1  0  8  5
-------------
1  0  6  5  2
```

Solution:-

S = 9 , M = 1

E = 5 , O = 0

N = 6 , N = 6

D = 7 , E = 5

Y = 2

Q 19 Modus Ponens is a fundamental rule of inference in propositional logic

If $P \rightarrow Q$ & P is true, then Q is also true

Example :- 1) If it rains, the ground will be wet ($P \rightarrow Q$)

1) It is raining (P is true)

3) Therefore, the ground is wet (Q is true)

Mathematically,

$P \rightarrow Q$

P

$\therefore Q$

Q 20 ① Forward Chaining is a data driven

② It starts from known facts & applies inference rules to derive new facts until the goal is reached

③ It is used in expert systems

Example :-

Fact - Sore Throat

Rule :- If sore throat → infection

New Fact :- Infection

Rule :- If infection → need antibiotics

Conclusion :- "Needs to Antibiotic

Q20 Backward Chaining

(1) It is more goal driven reference.
(2) It starts from goal & works backward to determine facts needed to prove it
(3) It is used in prolog programming

Example:-

goal : Does the patient need antibiotics?

(1) Check : Does the patient have an infection?

(2) Check : Does the patient have a sore throat?

(3) If both hold, conclude " Need antibiotic"

This reduces unnecessary computations by only exploring relevant facts

AIDS-I Assignment No: 2

Q.1: Use the following data set for question 1

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

1. Find the Mean (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Solution:

1. Finding the Mean

The mean is calculated by adding all values and dividing by the total number of values.

Sum of all values: 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1611

Number of values: 20

Mean = 1611 ÷ 20 = 80.55

2. Finding the Median

To find the median, I'll first arrange the data in ascending order:

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

Since there are 20 values (an even number), the median is the average of the 10th and 11th values: 10th value: 81 11th value: 82

Median = (81 + 82) ÷ 2 = 81.5

3. Finding the Mode

The mode is the value that appears most frequently in the data set.

Counting the occurrences: The value 76 appears 3 times, which is more than any other value. Therefore, the mode is 76.

4. Finding the Interquartile Range (IQR)

The IQR is the difference between the third quartile (Q3) and the first quartile (Q1).

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

First, I'll find Q1 (the median of the lower half of the data): Lower half: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81 Median of lower half (Q1) = (76 + 76) ÷ 2 = 76

Next, I'll find Q3 (the median of the upper half of the data): Upper half: 82, 82, 84, 85, 88, 90, 90, 91, 95, 99 Median of upper half (Q3) = (88 + 90) ÷ 2 = 89

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

    - Predictive analytic
    - Descriptive analytic

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

- Supervised learning
- Unsupervised learning
- Reinforcement learning

Solution:

1. Machine Learning for Kids

Target Audience:

- Primary and secondary school students (K-12)
- Teachers without extensive programming or ML background
- Educational institutions focusing on introducing AI concepts to young learners

Use by Target Audience:

- Students use it to create simple ML models through a block-based programming interface (Scratch)
- Teachers implement it in classroom activities to introduce AI and ML concepts
- Used for creating interactive projects like text classifiers, image recognition games, and simple AI-powered applications
- Serves as an introduction to computational thinking and AI literacy

Benefits:

- User-friendly interface designed specifically for children
- Requires minimal technical knowledge to get started
- Integrates with Scratch, which many students are already familiar with
- Provides real hands-on experience with machine learning concepts
- Scaffolded learning approach that simplifies complex ML concepts
- Free for basic use with options for schools

Drawbacks:

- Limited in complexity and scale compared to professional ML tools
- Simplified models may not represent the full capabilities of ML
- Some features require paid subscriptions for classroom use
- Limited customization of models compared to professional tools
- May oversimplify some technical aspects of machine learning

Classification: Predictive Analytics

Machine Learning for Kids focuses on creating models that make predictions based on input data, whether classifying text, images, or numbers. The tool is designed to help children build models that can predict outcomes or categorize new inputs based on training examples, making it clearly aligned with predictive analytics.

Classification: Supervised Learning

This tool primarily uses supervised learning approaches where:

- Students explicitly label and categorize training examples
- The system learns from these labeled examples to classify new inputs
- Models are trained on specific examples provided by users
- The focus is on teaching computers to recognize patterns from labeled data

2. Teachable Machine

Target Audience:

- Beginners and non-technical users interested in ML
- Educators teaching AI concepts without requiring programming
- Creative professionals wanting to incorporate simple ML into projects
- Students learning about AI fundamentals
- Small businesses exploring ML capabilities

Use by Target Audience:

- Creating custom image, sound, or pose recognition models through a browser interface
- Integrating simple ML capabilities into websites, art installations, or educational demonstrations
- Rapid prototyping of ML applications without coding
- Classroom demonstrations of how machine learning works with visual feedback

Benefits:

- No coding required - entirely visual interface
- Immediate visual feedback during training
- Models can be exported to use in various platforms (TensorFlow.js, Arduino)
- Free and accessible through web browsers
- Quick to set up and train basic models
- Transparent visualization of how the model makes decisions

Drawbacks:

- Limited to specific types of inputs (images, sounds, poses)
- Less powerful than professional ML frameworks
- Limited customization of model architecture
- Models may not perform well on complex problems or varied inputs
- No advanced features like transfer learning customization or model tuning
- Privacy concerns when using cloud-based services for training

Classification: Predictive Analytics

Teachable Machine is focused on building models that predict classifications based on new inputs. Whether classifying images, sounds, or poses, the tool creates systems that predict labels for new inputs based on learned patterns, making it firmly in the predictive analytics category.

Classification: Supervised Learning

Teachable Machine employs supervised learning because:

- Users explicitly provide examples for each category
- The system learns from labeled examples provided by the user
- Models are trained on direct examples with known outcomes
- The learning process requires human guidance through providing labeled examples
- The goal is to correctly classify new inputs based on patterns learned from labeled training data

Both tools represent supervised learning approaches because they rely on labeled data provided by users to train models that can classify or predict outcomes for new inputs.

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

- Read the article Kakande, Arthur. February 12. *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.

Solution:

Both of the provided articles discuss how poorly designed data visualizations can mislead the public, particularly during the Covid-19 pandemic.

Foley's article, "How bad Covid-19 data visualizations mislead the public," provides specific examples of misleading visualizations used by US state health departments during the pandemic. The article highlights several "data visualization follies". For example, **Alabama's health department used snapshot data and cluttered numbers, failing to show trends over time**, which are more meaningful. They also inappropriately used **pie charts**, which are generally hard to read and less effective than bar charts or tables for data comparison.

Arkansas' data visualization, while providing consistent data over time, **lacked context**. By showing low percentages of pre-existing conditions on a scale up to 100%, it downplayed the significant number of individuals with these conditions among Covid-19 cases and their increased risk of severe illness.

Arizona's Covid-19 dashboard displayed **charts without a y-axis and a non-uniform color gradient**, making it appear that the magnitude of cases was similar across the entire state and in a single county, despite significant differences in actual numbers. The article also points out that some data visualizations were well done. Washington's health department provided **clarity in labeling** by reporting both the percentage of positive tests and the total number of tests, along with a description of testing changes over time. New York effectively used **tables** to show the disproportionately high fatality rates among Black and Hispanic residents by presenting both the percentage of Covid-19 fatalities and their percentage of the population. Dan Kopf, Quartz Data editor, suggests that if creating a graphic becomes too convoluted, simply publishing the data in a table might be the best approach.

Kakande's article, "What’s in a chart? A Step-by-Step guide to Identifying Misinformation in a Data Visualization," offers a broader perspective on how data visualizations can be misleading. The article distinguishes between **misinformation**, which is incorrect or misleading information causing people to be

misinformed, and **disinformation**, which deliberately spreads false information to influence public opinion.

Kakande outlines several common data misrepresentations that lead to misinformation:

- **Truncated charts or graphs:** Manipulating the y-axis by omitting the baseline or exaggerating the scale can distort the perceived trends and differences in the data.
- **Correlation vs. Causation:** Mistakenly implying that a relationship between two events means one causes the other is a frequent error.
- **The color scale:** Deviating from common color norms (e.g., green for positive, red for negative) without clear explanation can lead to misinterpretation.
- **Trend manipulation:** Selectively showing only a portion of a trend can create a misleading impression if the overall picture is different.
- **Pie charts that don't add up:** Pie charts where the proportions do not equal 100% or 360 degrees indicate manipulated figures.

The article emphasizes that data visualization designers should ensure the purpose is well-defined, the content supports the purpose, the structure accurately represents reality, and the visualization highlights what is important while removing unnecessary elements. Ultimately, while data itself may be accurate, the way it is visualized can distort the message.

How the data visualization method failed:

In this hypothetical scenario, the **truncated y-axis** is the primary flaw. By not starting the y-axis at zero, the small absolute increase from 52 to 58 incidents is visually exaggerated, creating a false impression of a "spike" in crime. The viewer, focusing on the relative height difference of the bars, is likely to perceive a much more significant change than actually occurred. This misrepresentation can lead to **misinformation** as people become unduly alarmed about a sudden crime wave based on a distorted visual representation of the data.

A recent example that echoes these concerns is discussed in the *Financial Times* article *"Measurement matters"* (2024). The article explores how varying methodologies and visual representations of inflation data across countries have led to confusion among the public and investors. Specifically, it highlights how presenting inflation trends without proper context—such as differences in how food and energy prices are weighted—can result in visual comparisons that appear misleading. Charts that fail to account for these differences or that exaggerate minor statistical variations risk misinforming viewers, especially when widely shared on social media or cited in political discourse.

This case reinforces the importance of thoughtful, transparent data visualization. Even when data itself is accurate, poor visual design or lack of context can distort the message. As both Kakande and Foley emphasize, visualizations must be evaluated critically—not only for correctness but also for how effectively they support truthful, clear communication.

Cited Source:

Giles, Chris. "Measurement matters." *Financial Times*, June 21, 2024.

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

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

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved: used smote to resample and balance classes

```
from imblearn.over_sampling import SMOTE
from collections import Counter
from sklearn.model_selection import train_test_split

# Separate features and target
X = df.drop(columns=['Outcome'])
y = df['Outcome']

# Show original class distribution
print("Original class distribution:", Counter(y))

# Apply SMOTE to balance classes
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Show new class distribution
print("Resampled class distribution:", Counter(y_resampled))
```

```
Original class distribution: Counter({0: 500, 1: 268})
Resampled class distribution: Counter({1: 500, 0: 500})
```

- Data Pre-processing must be used: We handled the zero value columns and standardized the data

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Fit only on training data and transform both train and test
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
zero_cols = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
medians = df[zero_cols].median()
df = df.replace({col: {0: np.nan} for col in zero_cols})
df = df.fillna(medians)
# Verify changes
print("\nAfter zero value handling:")
print(df[zero_cols].describe())
```

- Hyper parameter tuning must be used: It is done using random forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid_rf = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Initialize Random Forest
rf = RandomForestClassifier(random_state=42)

# Grid search with 5-fold cross-validation
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, n_jobs=-1, scoring='accuracy')
grid_search_rf.fit(X_train, y_train)

# Best model
best_rf = grid_search_rf.best_estimator_
print("Best Parameters:", grid_search_rf.best_params_)
```

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
```

- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done

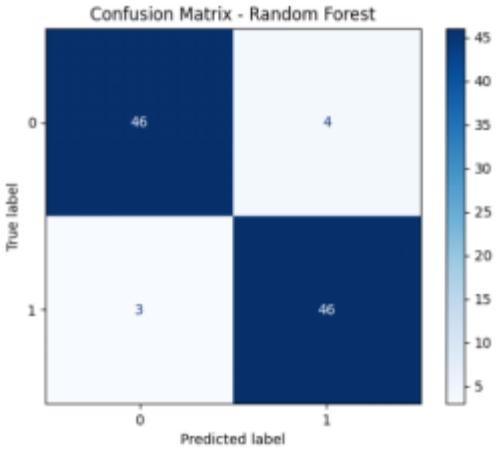
```
from sklearn.model_selection import train_test_split

# Step 1: Split into train (70%) and temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(
    X_resampled, y_resampled, test_size=0.30, random_state=42, stratify=y_resampled)

# Step 2: Split temp into validation (20%) and test (10%) → 2:1 ratio of 30%
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.33, random_state=42, stratify=y_temp)  # 0.33 of 30% ≈ 10%

# Confirm sizes
print("Train size:", len(X_train))
print("Validation size:", len(X_val))
print("Test size:", len(X_test))
```

```
Train size: 700
Validation size: 201
Test size: 99
```

- Classification Accuracy should be maximized

```
test_pred = best_model.predict(X_test_scaled)
test_proba = best_model.predict_proba(X_test_scaled)[:, 1]

print("Accuracy:", accuracy_score(y_test, test_pred))
print("\nClassification Report:")
print(classification_report(y_test, test_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, test_pred))
```

```
Test Accuracy: 0.9293

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        50
           1       0.92      0.94      0.93        49

    accuracy                           0.93        99
   macro avg       0.93      0.93      0.93        99
weighted avg       0.93      0.93      0.93        99
```



The accuracy of the model is about 93 percent.

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

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

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

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

Solution:

Read the file and set the dependent and independent variables

```python
# Step 1: Load and Prepare the Data

import pandas as pd
from sklearn.model_selection import train_test_split

# Load the dataset
data = pd.read_csv('BostonHousing.csv')

# Independent variable: 1st column
X = data.iloc[:, [0]]  # 'crim'

# Dependent variables: columns 2 to 5
y = data.iloc[:, 1:5]  # 'zn', 'indus', 'chas', 'nox'
```

Split data to Train/Test – 70% and 30%

```
# Split data into 70% train, 30% test (random split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Display shapes to confirm
print("X_train:", X_train.shape)
print("y_train:", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
```

```
X_train: (354, 1)
y_train: (354, 4)
X_test: (152, 1)
y_test: (152, 4)
```

RandomForest Regressor

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error

# Define the model
rf = RandomForestRegressor(random_state=42)

# Define parameter grid for tuning
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Setup GridSearchCV
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           cv=5, n_jobs=-1, scoring='r2', verbose=1)

# Fit on training data
grid_search.fit(X_train, y_train)

# Best model
best_rf = grid_search.best_estimator_

# Predict on test data
y_pred = best_rf.predict(X_test)

# Evaluate
r2 = r2_score(y_test, y_pred)
n = X_test.shape[0]
p = X_test.shape[1]
adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
mse = mean_squared_error(y_test, y_pred)

# Print results
print(f"Best Parameters: {grid_search.best_params_}")
print(f"R² Score: {r2:.4f}")
print(f"Adjusted R² Score: {adjusted_r2:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 100}
R² Score: 0.9918
Adjusted R² Score: 0.9916
Mean Squared Error: 0.4213
```

Optimal Parameters: The model achieves the best performance with a moderately deep tree (max_depth=3), along with specific values for splits and the number of trees.

R^2 Value: 0.99182 — This indicates that the model accounts for 99.18% of the variance in the dependent variables, reflecting excellent accuracy.

Adjusted R^2 Value: 0.99160 — Even after adjusting for the number of predictors, the model still explains 99.16% of the variance, demonstrating strong generalization.

Mean Squared Error: 0.4213 — The small average squared difference between predicted and actual values suggests that the model has minimal prediction error.

Q.6 What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

Solution:

The Wine Quality dataset is a widely used dataset in machine learning and statistical analysis, containing physicochemical properties and sensory ratings of red and white wines. The primary objective of this dataset is to predict wine quality based on measurable attributes. This assignment explores the key features of the dataset, their significance in determining wine quality, and the techniques used to handle missing data during feature engineering.

Key Features and Their Importance in Predicting Wine Quality

The dataset includes several physicochemical and sensory attributes that influence wine quality. Below is an analysis of the most important features:

1. **Fixed Acidity** – This refers to non-volatile acids (such as tartaric and malic acid) that contribute to the wine's tartness. A balanced level is essential; too much acidity makes the wine taste harsh, while too little results in a flat taste.
2. **Volatile Acidity** – High levels of volatile acidity (primarily acetic acid) can lead to an unpleasant vinegar-like taste, negatively impacting quality.
3. **Citric Acid** – A small amount enhances freshness and flavor complexity, but excessive citric acid can make the wine overly sour.
4. **Residual Sugar** – Determines the sweetness of the wine. While some sugar improves taste, excessive amounts can make the wine cloying.
5. **Chlorides** – Represent the salt content. High chloride levels can make wine taste salty, reducing its appeal.
6. **Free and Total Sulfur Dioxide** – Sulfur dioxide acts as a preservative, preventing oxidation and microbial spoilage. However, excessive amounts can introduce an undesirable sulfurous odor.
7. **Density** – Influenced by alcohol and sugar content, density can indicate the wine's body and mouthfeel.
8. **pH** – Measures acidity on a logarithmic scale. A balanced pH ensures stability and a pleasant taste.
9. **Sulphates** – Additives like potassium sulphate help preserve wine and influence aging. Higher sulphates may correlate with better quality in some cases.
10. **Alcohol (%)** – A key factor in wine quality, alcohol contributes to body, sweetness, and balance. Wines with moderate alcohol levels are generally preferred.

The **target variable** is *quality*, typically rated on a scale from 0 (poor) to 10 (excellent). Understanding how these features interact helps in building accurate predictive models.

Handling Missing Data in Feature Engineering

Missing data is a common issue in real-world datasets and must be addressed carefully to avoid bias. Below are common techniques for handling missing values in the Wine Quality dataset:

1. Deletion Methods

- **Listwise Deletion (Complete Case Analysis)** – Removes any row with missing values.
    - *Advantage*: Simple and avoids introducing imputation bias.
    - *Disadvantage*: Reduces dataset size, potentially losing valuable information.
- **Pairwise Deletion** – Uses available data points for each analysis.
    - *Advantage*: Retains more data than listwise deletion.
    - *Disadvantage*: Can lead to inconsistencies if missingness is not random.

2. Imputation Methods

- **Mean/Median/Mode Imputation** – Replaces missing values with the mean (for numerical data) or mode (for categorical data).
    - *Advantage*: Easy to implement and works well for small missingness.
    - *Disadvantage*: Reduces variance and may distort statistical relationships.
- **Regression Imputation** – Predicts missing values using regression models based on other features.
    - *Advantage*: More accurate than mean imputation if strong correlations exist.
    - *Disadvantage*: Can overfit if relationships are weak.
- **K-Nearest Neighbors (KNN) Imputation** – Uses similarity between data points to estimate missing values.
    - *Advantage*: Works well for datasets with meaningful distance metrics.
    - *Disadvantage*: Computationally expensive for large datasets.
- **Multiple Imputation (MICE – Multivariate Imputation by Chained Equations)** – Generates multiple imputed datasets and combines results.
    - *Advantage*: Accounts for uncertainty and is robust for Missing at Random (MAR) data.
    - *Disadvantage*: More complex to implement and interpret.

Best Practices for the Wine Quality Dataset

- If missing data is minimal (<5%), mean or median imputation may suffice.
- For larger missingness, advanced techniques like MICE or KNN imputation are preferable.
- Before choosing a method, it is crucial to analyze whether the missingness is **Missing Completely at Random (MCAR), Missing at Random (MAR), or Missing Not at Random (MNAR).**

The Wine Quality dataset provides valuable insights into how physicochemical properties influence wine ratings. Understanding feature importance helps in building better predictive models, while proper missing data handling ensures reliable analysis. By using appropriate imputation techniques, we can maintain data integrity and improve model performance. Future work could explore feature interactions and advanced machine learning models for enhanced predictions.