Name: Chinmay Chaudhari
Div: D15C
Roll No:06

EXP 1

Aim:

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

Data preprocessing

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Why is Data Preprocessing important?

Preprocessing of data is mainly to check the data quality. The quality can be checked by the following:

- Accuracy: To check whether the data entered is correct or not.
- Completeness: To check whether the data is available or not recorded.
- Consistency: To check whether the same data is kept in all the places that do or do not match.

- Timeliness: The data should be updated correctly.
- Believability: The data should be trustable.

Interpretability: The understandability of the data.

Dataset: SuperMarket Dataset

1) Loading Data in Pandas

```
import pandas as pd

df = pd.read_csv('sss.csv')

df.head()
```

	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.141
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.820
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.215
3	123-19-1176	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.288
4	373-73-7910	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.208

2)Description of the dataset.

Attribute/Column Name	Data Type	Description
Invoice ID	String	Unique identifier for each transaction/invoice.
Branch	String	Branch identifier for the supermarket (A, B, or C).
City	String	City where the supermarket branch is located.
Customer type	String	Type of customer (Member or Normal).
Gender	String	Gender of the customer (Male or Female).
Product line	String	Category of products purchased (Health and beauty, Electronic accessories, etc.).
Unit price	Float	Price per unit of the product.
Quantity	Integer	Number of units purchased.
Tax 5%	Float	5% tax on the total amount for the purchase.
Total	Float	Total bill amount, including tax.
Date	DateTime	Date of the purchase transaction.
Time	String	Time of the purchase transaction.
Payment	String	Payment method used (Cash, Credit card, or Ewallet).
cogs (Cost of Goods Sold)	Float	Total cost of goods sold before tax.
gross margin %	Float	Percentage of gross margin fixed at 4.76%.
gross income	Float	Profit made from the transaction.
Rating	Float	Customer's rating of their experience (range: 1 to 10).

`df.info()`: Provides an overview of the dataset, including:

- Number of rows and columns.
- Data types of each column (e.g., int, float, object).
- Number of non-null (non-missing) values in each column.

`df.describe()`: Generates summary statistics for numeric columns, such as:

- `count`: Number of non-missing values.
- `mean`: Average value.
- `std`: Standard deviation.
- `min`, `25%`, `50%` (median), `75%`, and `max`: Percentile values.

```
print(df.info())

print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Invoice ID               1000 non-null   object
 1   Branch                   1000 non-null   object
 2   City                     1000 non-null   object
 3   Customer type            1000 non-null   object
 4   Gender                   1000 non-null   object
 5   Product line             1000 non-null   object
 6   Unit price               1000 non-null   float64
 7   Quantity                 1000 non-null   int64
 8   Tax 5%                   1000 non-null   float64
 9   Total                    1000 non-null   float64
 10  Date                     1000 non-null   object
 11  Time                     1000 non-null   object
 12  Payment                  1000 non-null   object
 13  cogs                     1000 non-null   float64
 14  gross margin percentage  1000 non-null   float64
 15  gross income             1000 non-null   float64
 16  Rating                   1000 non-null   float64
dtypes: float64(7), int64(1), object(9)
memory usage: 132.9+ KB
None
        Unit price     Quantity       Tax 5%        Total        cogs  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.00000
mean     55.672130     5.510000    15.379369   322.966749   307.58738
std      26.494628     2.923431    11.708825   245.885335   234.17651
min      10.080000     1.000000     0.508500    10.678500    10.17000
25%      32.875000     3.000000     5.924875   124.422375   118.49750
50%      55.230000     5.000000    12.088000   253.848000   241.76000
75%      77.935000     8.000000    22.445250   471.350250   448.90500
max      99.960000    10.000000    49.650000  1042.650000   993.00000
```

3) Drop columns that aren't useful: Columns like Invoice ID may not contribute to analysis (it's often just an identifier). Removing irrelevant columns reduces complexity.

```
[ ] df = df.drop(['Invoice ID'], axis=1)

    df.head()
```

	Branch	City	Customer type	Gender	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating
0	A	Yangon	Member	Female	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1
1	C	Naypyitaw	Normal	Female	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6
2	A	Yangon	Normal	Male	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4
3	A	Yangon	Member	Male	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4
4	A	Yangon	Normal	Male	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3

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

4)Drop rows with maximum missing values.

`df.dropna(thresh=int(0.5 * len(df.columns)))`:

- Drops rows where more than half the columns have missing (NaN) values.
- `thresh=int(0.5 * len(df.columns))`: Ensures that a row must have at least 50% non-null values to remain.

`df = ...`: Updates the DataFrame after dropping rows.
`print(df.info())`: Confirms that rows with excessive missing values have been removed.

```
[ ] df = df.dropna(thresh=int(0.5 * len(df.columns)))

    df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 16 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Branch                   1000 non-null   object
 1   City                     1000 non-null   object
 2   Customer type            1000 non-null   object
 3   Gender                   1000 non-null   object
 4   Product line             1000 non-null   object
 5   Unit price               1000 non-null   float64
 6   Quantity                 1000 non-null   int64
 7   Tax 5%                   1000 non-null   float64
 8   Total                    1000 non-null   float64
```

5)Take care of missing data.

`df.fillna(df.mean())`: Replaces missing values (NaN) in numeric columns with the mean of that column.

```
print(df.isnull().sum())
# to check null values..this is like boolean ..if nan ? true:false ..then adds all bolean values.....finally 0 matlab false....this dataset has no null or NaN values.
```

```
Branch                     0
City                       0
Customer type              0
Gender                     0
Product line               0
Unit price                 0
Quantity                   0
Tax 5%                     0
Total                      0
Date                       0
Time                       0
Payment                    0
cogs                       0
gross margin percentage    0
```

6)Create dummy variables.

`pd.get_dummies()`: Converts categorical variables into dummy variables (binary indicators: 0 or 1).

- Example: The `Gender` column becomes `Gender_Male` (1 if Male, 0 otherwise).

`columns=['...']`: Specifies which columns to convert.
`drop_first=True`: Avoids the "dummy variable trap" by dropping one dummy variable to prevent multicollinearity.

```
df = pd.get_dummies(df, columns=['Gender', 'City'], drop_first=True)
```

```
df.head()
```

	Branch	Customer Type	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating	Gender_Male	City_Naypyitaw	City_Yangon
0	A	Member	Health and beauty	74.69	7	26.1415	548.9715	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1	False	False	True
1	C	Normal	Electronic accessories	15.28	5	3.8200	80.2200	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6	False	True	False
2	A	Normal	Home and lifestyle	46.33	7	16.2155	340.5255	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4	True	False	True
3	A	Member	Health and beauty	58.22	8	23.2880	489.0480	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4	True	False	True
4	A	Normal	Sports and travel	86.31	7	30.2085	634.3785	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3	True	False	True

7)Find out outliers (manually)

```python
def detect_outliers(col):
    Q1 = df[col].quantile(0.25)  # First quartile (25th percentile)
    Q3 = df[col].quantile(0.75)  # Third quartile (75th percentile)
    IQR = Q3 - Q1                # Interquartile range
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[col] < lower_bound) | (df[col] > upper_bound)]
```

```python
outliers = detect_outliers('Total')
print(outliers)
if outliers.empty:
    print("No outliers detected.")
else:
    print(f"Outliers detected:\n{outliers}")

print(f"Number of outliers: {len(outliers)}")
```

```
    Branch Customer type           Product line  Unit price  Quantity  Tax 5%  \
166      C        Normal     Home and lifestyle       95.58        10  47.790
167      A        Normal  Fashion accessories        98.98        10  49.490
350      C        Member  Fashion accessories        99.30        10  49.650
357      C        Normal      Sports and travel       95.44        10  47.720
422      C        Member  Fashion accessories        97.21        10  48.605
557      C        Member     Food and beverages       98.52        10  49.260
699      C        Normal     Home and lifestyle       97.50        10  48.750
792      B        Normal     Home and lifestyle       97.37        10  48.685
996      B        Normal     Home and lifestyle       97.38        10  48.690

        Total       Date   Time      Payment   cogs  gross margin percentage  \
166  1003.590  1/16/2019  13:32         Cash  955.8                 4.761905
167  1039.290   2/8/2019  16:20  Credit card  989.8                 4.761905
350  1042.650  2/15/2019  14:53  Credit card  993.0                 4.761905
357  1002.120   1/9/2019  13:45         Cash  954.4                 4.761905
422  1020.705   2/8/2019  13:00  Credit card  972.1                 4.761905
557  1034.460  1/30/2019  20:23      Ewallet  985.2                 4.761905
699  1023.750  1/12/2019  16:18      Ewallet  975.0                 4.761905
792  1022.385  1/15/2019  13:48  Credit card  973.7                 4.761905
996  1022.490   3/2/2019  17:16      Ewallet  973.8                 4.761905

     gross income  Rating  Gender_Male  City_Naypyitaw  City_Yangon
166        47.790     4.8         True            True        False
167        49.490     8.7         True           False         True
```

```
        Total       Date   Time      Payment   cogs  gross margin percentage  \
166  1003.590  1/16/2019  13:32         Cash  955.8                 4.761905
167  1039.290   2/8/2019  16:20  Credit card  989.8                 4.761905
350  1042.650  2/15/2019  14:53  Credit card  993.0                 4.761905
357  1002.120   1/9/2019  13:45         Cash  954.4                 4.761905
422  1020.705   2/8/2019  13:00  Credit card  972.1                 4.761905
557  1034.460  1/30/2019  20:23      Ewallet  985.2                 4.761905
699  1023.750  1/12/2019  16:18      Ewallet  975.0                 4.761905
792  1022.385  1/15/2019  13:48  Credit card  973.7                 4.761905
996  1022.490   3/2/2019  17:16      Ewallet  973.8                 4.761905

     gross income  Rating  Gender_Male  City_Naypyitaw  City_Yangon
166        47.790     4.8         True            True        False
167        49.490     8.7         True           False         True
350        49.650     6.6        False            True        False
357        47.720     5.2        False            True        False
422        48.605     8.7        False            True        False
557        49.260     4.5        False            True        False
699        48.750     8.0         True            True        False
792        48.685     4.9        False           False        False
996        48.690     4.4        False           False        False
Number of outliers: 9
```

8) standardization and normalization of columns

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Standardization equation

$$X' = \frac{X - \mu}{\sigma}$$

To standardize your data, we need to import the StandardScalar from the sklearn library and apply it to our dataset.

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Normalization equation

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X' is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X' is 1
- If the value of X is between the minimum and the maximum value, then the value of X' is between 0 and 1

To normalize your data, you need to import the MinMaxScalar from the sklearn library and apply it to our dataset.

```
from sklearn.preprocessing import StandardScaler
# mean 0 karega and sd 1..this is standardlization
scaler = StandardScaler()
df[['Unit price', 'Total']] = scaler.fit_transform(df[['Unit price', 'Total']])
```

```
from sklearn.preprocessing import MinMaxScaler
# this is normaliztion.....helps to scale data between 0-1
normalizer = MinMaxScaler()
df[['Unit price', 'Total']] = normalizer.fit_transform(df[['Unit price', 'Total']])
```

```
df.head()
```

	Branch	Customer type	Product line	Unit price	Quantity	Tax 5%	Total	Date	Time	Payment	cogs	gross margin percentage	gross income	Rating	Gender_Male	City_Naypyitaw	City_Yangon
0	A	Member	Health and beauty	0.718847	7	26.1415	0.521616	1/5/2019	13:08	Ewallet	522.83	4.761905	26.1415	9.1	False	False	True
1	C	Normal	Electronic accessories	0.057055	5	3.8200	0.067302	3/8/2019	10:29	Cash	76.40	4.761905	3.8200	9.6	False	True	False
2	A	Normal	Home and lifestyle	0.403250	7	16.2155	0.319628	3/3/2019	13:23	Credit card	324.31	4.761905	16.2155	7.4	True	False	True
3	A	Member	Health and beauty	0.533603	8	23.2880	0.462526	1/27/2019	20:33	Ewallet	465.76	4.761905	23.2880	8.4	True	False	True
4	A	Normal	Sports and travel	0.848131	7	30.2085	0.604377	2/8/2019	10:37	Ewallet	604.17	4.761905	30.2085	5.3	True	False	True

```
df.describe()
```

	Unit price	Quantity	Tax 5%	Total	cogs	gross margin percentage	gross income	Rating
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000	1.000000e+03	1000.000000	1000.00000
mean	0.507256	5.510000	15.379369	0.302513	307.58738	4.761905e+00	15.379369	6.97270
std	0.294778	2.923431	11.708825	0.236266	234.17651	6.131498e-14	11.708825	1.71858
min	0.000000	1.000000	0.508500	0.000000	10.17000	4.761905e+00	0.508500	4.00000
25%	0.253616	3.000000	5.924875	0.110223	118.49750	4.761905e+00	5.924875	5.50000
50%	0.502356	5.000000	12.088000	0.235656	241.76000	4.761905e+00	12.088000	7.00000
75%	0.754951	8.000000	22.445250	0.440400	448.90500	4.761905e+00	22.445250	8.50000
max	1.000000	10.000000	49.650000	1.000000	993.00000	4.761905e+00	49.650000	10.00000

<u>**Conclusion**</u>:

Thus we have understood how to perform data preprocessing which can further be taken into exploratory data analysis and further in the Model preparation sequence.

Name: Chinmay Chaudhari
Class: D15C
Roll No.: 6

Exp 2

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

Introduction:

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the first step in your data analysis process developed by "John Tukey" in the 1970s. In statistics, exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. By the name itself, we can get to know that it is a step in which we need to explore the data set.

When you are trying to build a machine learning model you need to be pretty sure whether your data is making sense or not. The main aim of exploratory data analysis is to obtain confidence in your data to an extent where you're ready to engage a machine learning algorithm.

Why do we do EDA?

Exploratory Data Analysis is a crucial step before you jump to machine learning or modeling your data. By doing this you can get to know whether the selected features are good enough to model, are all the features required, are there any correlations based on which we can either go back to the Data Preprocessing step or move on to modeling.

Once EDA is complete and insights are drawn, its feature can be used for supervised and unsupervised machine learning modeling.

In every machine learning workflow, the last step is Reporting or Providing the insights to the Stakeholders and as a Data Scientist you can explain every bit of code but you need to keep in mind the audience. By completing the EDA you will have many plots,heat-maps, frequency distribution, graphs, correlation matrix along with the hypothesis by which any individual can understand what your data is all about and what insights you got from exploring your data set.

Data visualization is very critical to market research where both numerical and categorical data can be visualized, which helps in an increase in the impact of insights and also helps in reducing the risk of analysis paralysis

Advantages of Data visualization:

1. Better Agreement:

In business, for numerous periods, it happens that we need to look at the exhibitions of two components or two situations. A conventional methodology is to experience the massive information of both the circumstances and afterward examine it. This will clearly take a great deal of time.

2. A Superior Method:

It can tackle the difficulty of placing the information of both perspectives into the pictorial structure. This will unquestionably give a superior comprehension of the circumstances. For instance, Google patterns assist us with understanding information identified with top ventures or inquiries in pictorial or graphical structures.

3. Simple Sharing of Data:

With the representation of the information, organizations present another arrangement of correspondence. Rather than sharing the cumbersome information, sharing the visual data will draw in and pass on across the data which is more absorbable.

4. Deals Investigation:

With the assistance of information representation, a salesman can, without much of a stretch, comprehend the business chart of items. With information perception instruments like warmth maps, he will have the option to comprehend the causes that are pushing the business numbers up just as the reasons that are debasing the business numbers. Information representation helps in understanding the patterns and furthermore, different variables like sorts of clients keen on purchasing, rehashing clients, the impact of topography, and so forth.

5. Discovering Relations Between Occasions:

A business is influenced by a lot of elements. Finding a relationship between these elements or occasions encourages chiefs to comprehend the issues identified with their business. For instance, the online business market is anything but another thing today. Each time during certain happy seasons, like Christmas or Thanksgiving, the diagrams of online organizations go up. Along these lines, state if an online organization is doing a normal $1 million business in a specific quarter and the business ascends straightaway, at that point they can rapidly discover the occasions compared to it.

6. Investigating Openings and Patterns:

With the huge loads of information present, business chiefs can discover the profundity of information in regard to the patterns and openings around them. Utilizing information representation, the specialists can discover examples of the conduct of their clients, subsequently preparing for them to investigate patterns and open doors for business.

Introduction to Technologies Used:

Matplotlib

Matplotlib is a plotting library in Python used for creating static, animated, and interactive visualizations. It is highly customizable and supports a wide range of graphs, including bar graphs, histograms, scatter plots, and more.

Seaborn

Seaborn is a Python visualization library built on top of Matplotlib. It provides a high-level interface for creating attractive statistical graphics, such as heatmaps, box plots, and scatter plots.

General Syntax in Python for Data Visualization

Python libraries like Matplotlib and Seaborn follow a general syntax for creating visualizations:

1. **Import the library**: Import the required libraries (e.g., `import matplotlib.pyplot as plt`).
2. **Prepare the data**: Use Pandas to manipulate and prepare the data for visualization.
3. **Create the plot**: Use functions like `plot()`, `scatter()`, `boxplot()`, etc., to create the visualization.
4. **Customize the plot**: Add titles, labels, legends, and other customizations.
5. **Display the plot**: Use `plt.show()` to display the visualization.

<------This doc is using up on the cleaned data of previous experiment.------->

1. Bar Graph and Contingency Table

Theory

- **Bar Graph**: A bar graph is used to represent categorical data with rectangular bars. The length of each bar corresponds to the value it represents. It is useful for comparing categories or showing distributions.
- **Contingency Table**: A contingency table (also called a cross-tabulation) is a table that displays the frequency distribution of two categorical variables. It helps in understanding the relationship between the variables.

Terms

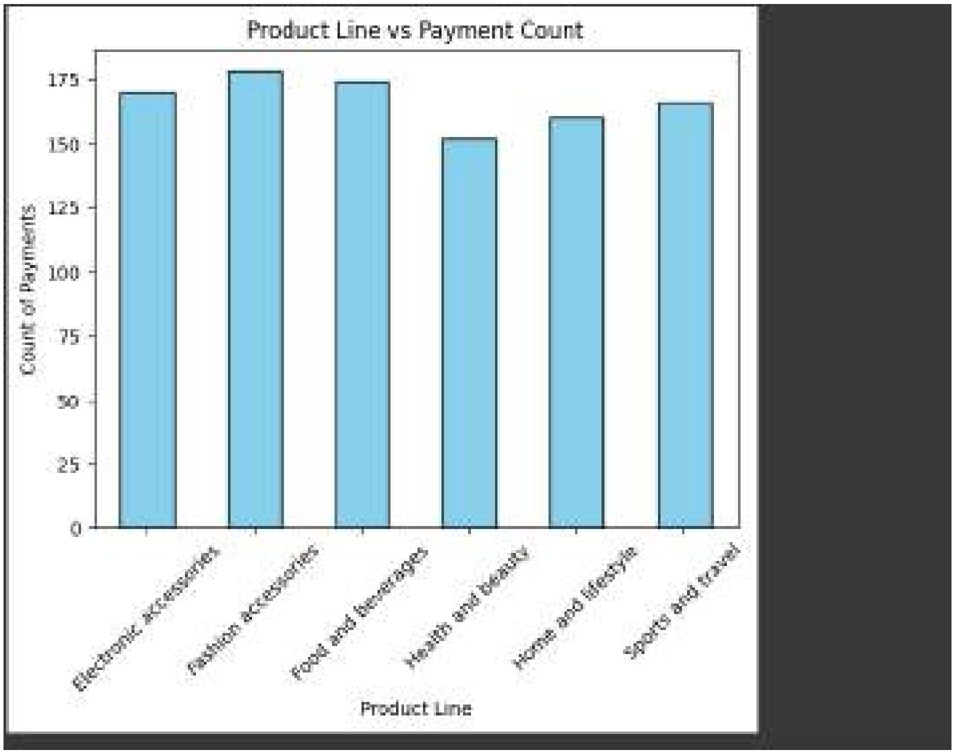
- **Categorical Data**: Data that can be divided into groups or categories (e.g., `Product line`, `Payment`).
- **Frequency**: The number of times a value occurs in a dataset.

```
# Bar plot for product line and payment method
df.groupby('Product line')['Payment'].count().plot(kind='bar', color='skyblue', edgecolor='blac
k')
plt.title('Product Line vs Payment Count')
plt.xlabel('Product Line')
plt.ylabel('Count of Payments')
plt.xticks(rotation=45)
plt.show()

# Contingency table
contingency_table = pd.crosstab(df['Product line'], df['Payment'])
print(contingency_table)
```

Explanation

- **Bar Graph**:
  - `df.groupby('Product line')['Payment'].count()` groups the data by `Product line` and counts the occurrences of each `Payment` method.
  - `.plot(kind='bar')` creates a bar graph.
  - `plt.title()`, `plt.xlabel()`, and `plt.ylabel()` add titles and labels to the graph.
  - `plt.xticks(rotation=45)` rotates the x-axis labels for better readability.
- **Contingency Table**:
  - `pd.crosstab(df['Product line'], df['Payment'])` creates a table showing the frequency distribution of `Payment` methods for each `Product line`.

Output:



Payment	Cash	Credit card	Ewallet
Product line			
Electronic accessories	71	46	53
Fashion accessories	57	56	65
Food and beverages	57	61	56
Health and beauty	49	50	53
Home and lifestyle	51	45	64
Sports and travel	59	53	54

2. Scatter Plot, Box Plot, and Heatmap

Theory

- **Scatter Plot**: A scatter plot is used to visualize the relationship between two numerical variables. Each point represents an observation.
- **Box Plot**: A box plot (or whisker plot) is used to display the distribution of numerical data through quartiles. It helps identify outliers and compare distributions across categories.
- **Heatmap**: A heatmap is a graphical representation of data where values are represented as colors. It is often used to visualize correlation matrices.

Terms

- **Numerical Data**: Data that represents quantities (e.g., `Unit price`, `Total`).
- **Quartiles**: Values that divide a dataset into four equal parts.
- **Correlation**: A measure of the relationship between two variables.

```
# Scatter plot
sns.scatterplot(data=df, x='Unit price', y='Total', hue='Gender_Male')
plt.title('Unit Price vs Total with Gender (Male=1)')
plt.show()

# Box plot
sns.boxplot(data=df, x='Product line', y='Total')
plt.title('Box Plot of Total by Product Line')
plt.xticks(rotation=45)
plt.show()

# Heatmap
numeric_df = df.select_dtypes(include=['float64', 'int64'])
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm')
plt.title('Heatmap of Numerical Features Correlation')
plt.show()
```

Explanation

- **Scatter Plot**:
  - `sns.scatterplot()` creates a scatter plot with `Unit price` on the x-axis and `Total` on the y-axis.
  - `hue='Gender_Male'` adds a color dimension to differentiate between genders.

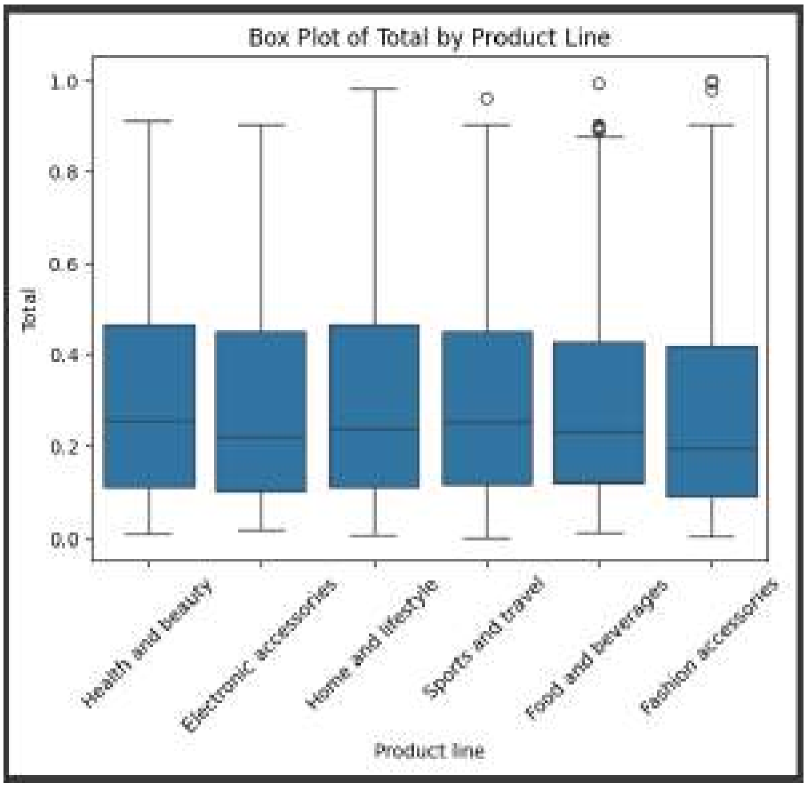
- **Box Plot**:
  - `sns.boxplot()` creates a box plot to show the distribution of `Total` sales across `Product line`.
  - `plt.xticks(rotation=45)` rotates the x-axis labels for better readability.
- **Heatmap**:
  - `numeric_df.corr()` calculates the correlation matrix for numerical features.
  - `sns.heatmap()` visualizes the correlation matrix with colors.

Output:

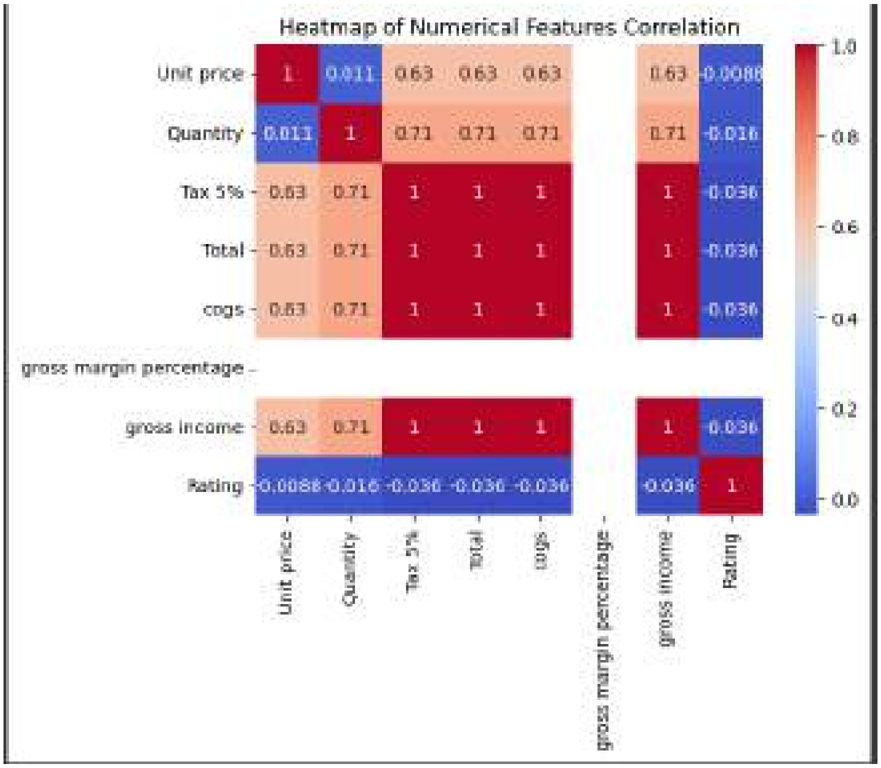
Scatter Plot

Inference

- If the points in the scatter plot show an upward trend (from bottom-left to top-right), it indicates a **positive correlation** between `Unit price` and `Total`. This means that as the unit price increases, the total sales amount also tends to increase.
- If the points are scattered randomly, it suggests **no strong correlation** between the two variables.



Box Plot



Heat Map

Key Observations:

- **Total vs Quantity (High Positive Correlation)**
  - A high positive correlation (close to 1) suggests that the total sales amount increases as the number of purchased items (Quantity) increases. This is expected in sales data.
- **Gross Income vs Total (Strong Positive Correlation)**
  - This indicates that a higher total amount is strongly associated with higher gross income. This is intuitive as gross income is often derived from total sales.
- **Weak Correlations:**
  - Some features, like *Unit Price* and *Quantity*, may show weak or no correlation, suggesting that the number of items purchased doesn't necessarily depend on unit prices.
- **No Negative Correlations:**
  - Since this is a sales dataset, most numerical features are likely positively related.

3. Histogram and Normalized Histogram

Theory

- **Histogram**: A histogram is used to represent the distribution of numerical data. It divides the data into bins and shows the frequency of observations in each bin.
- **Normalized Histogram**: A normalized histogram represents the probability distribution of the data, where the area under the histogram sums to 1.

Terms

- **Bins**: Intervals into which the data is divided.
- **Density**: The probability density of the data.

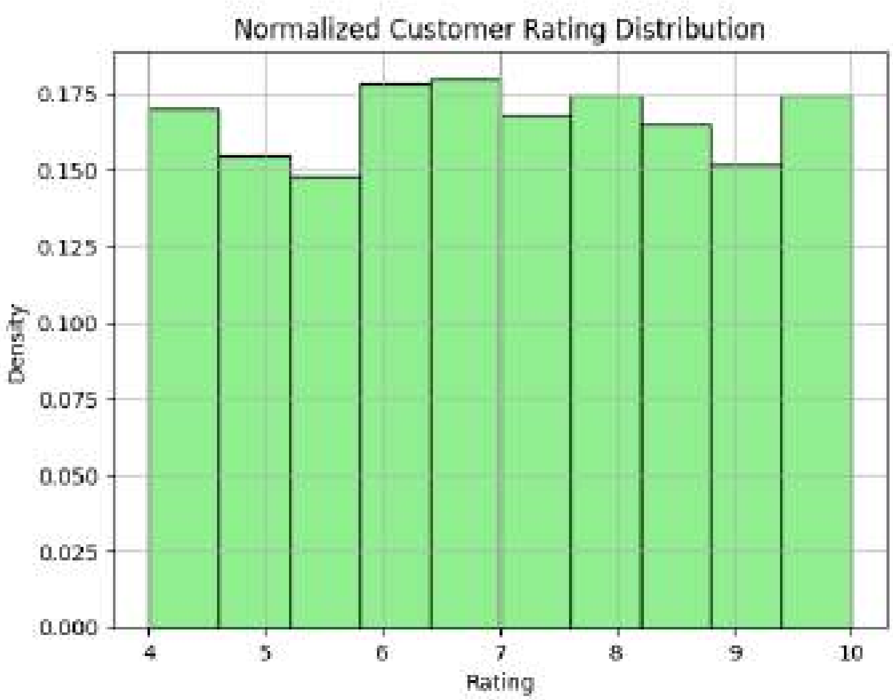
```
# Histogram
df['Rating'].hist(bins=10, color='lightblue', edgecolor='black')
plt.title('Customer Rating Distribution')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()

# Normalized Histogram
df['Rating'].hist(bins=10, density=True, color='lightgreen', edgecolor='black')
plt.title('Normalized Customer Rating Distribution')
plt.xlabel('Rating')
plt.ylabel('Density')
plt.show()
```

Explanation

- **Histogram**:
  - `df['Rating'].hist()` creates a histogram for the `Rating` column.
  - `bins=10` divides the data into 10 intervals.
  - `color` and `edgecolor` customize the appearance of the bars.
- **Normalized Histogram**:
  - `density=True` normalizes the histogram so that the area under the curve sums to 1.

Output:

Histogram



Normalized Histogram

Inference: Customer Rating Distribution Histogram

1. **Rating Spread:**
   The histogram shows how customer ratings are distributed across different ranges, with the bins dividing ratings from low to high.
2. **Most Common Ratings:**
   If there's a peak near higher ratings (like 8-10), it indicates customer satisfaction, whereas peaks at lower ratings suggest dissatisfaction trends.
3. **Skewness of Ratings:**If the distribution leans towards higher ratings, it suggests overall positive feedback from customers; if it's more balanced, opinions are mixed

4. Handling Outliers Using Box Plot and IQR

Theory

- **Outliers**: Data points that are significantly different from other observations.
- **Box Plot**: A box plot helps visualize outliers using the interquartile range (IQR).
- **IQR Method**: A statistical method to identify and remove outliers. Outliers are defined as observations below `Q1 - 1.5 * IQR` or above `Q3 + 1.5 * IQR`.

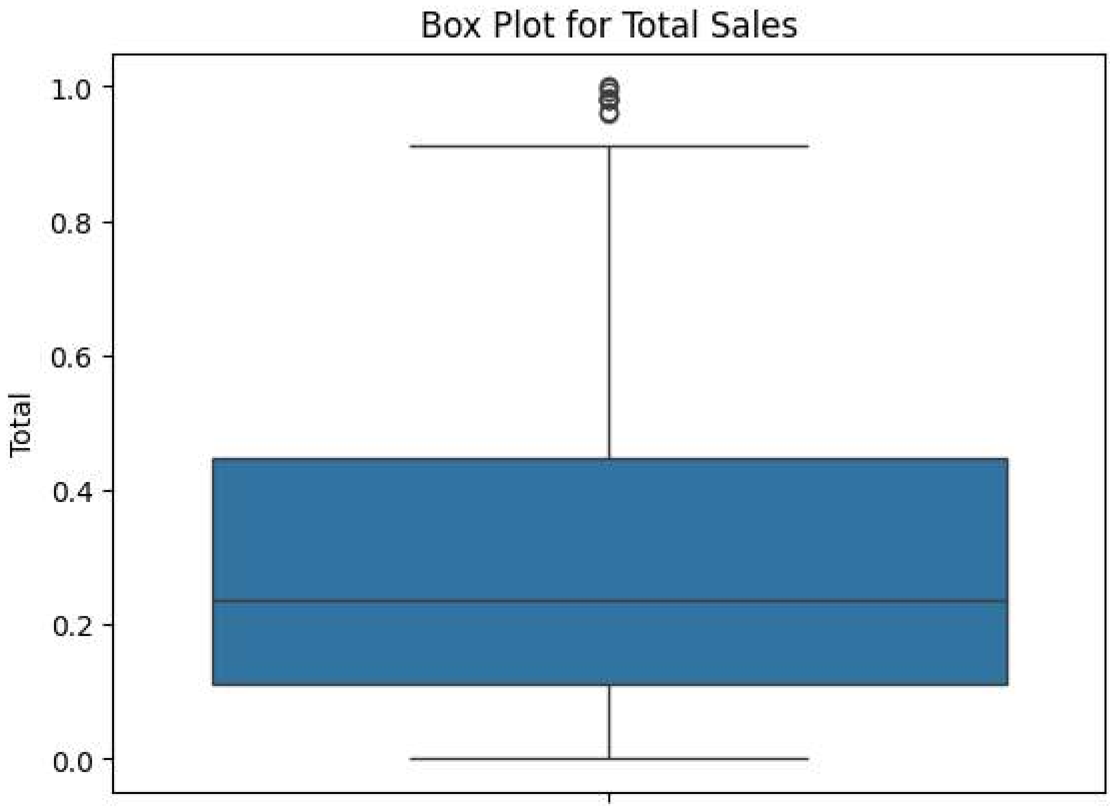
Terms

- **Quartiles (Q1, Q3)**: The 25th and 75th percentiles of the data.
- **IQR**: The range between Q1 and Q3.

Code:

```
# Box Plot to Visualize Outliers
sns.boxplot(data=df, y='Total')
plt.title('Box Plot for Total Sales')
plt.show()

# Handle Outliers with IQR
Q1 = df['Total'].quantile(0.25)
Q3 = df['Total'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
cleaned_df = df[(df['Total'] >= lower_bound) & (df['Total'] <= upper_bound)]
print(f"Rows before outlier removal: {len(df)}")
print(f"Rows after outlier removal: {len(cleaned_df)}")
```

Output:



Inference: Box Plot for Total Sales

1. Identifying Outliers:
   - Any data points outside the whiskers of the box plot are considered outliers. These points represent unusually high total sales amounts.
2. Sales Variability:
   - The spread of the box shows the range of typical sales values, while the whiskers indicate the overall variability.
3. Business Insight:
   - Outliers may indicate rare high-value transactions or potential data entry errors that require investigation.
   - Understanding these outliers can help identify key trends, such as promotional events leading to significant sales.

Outliers detected in *Total* or *Gross Income* columns suggest extreme sales figures, possibly due to special promotions or data entry errors.
Handling these outliers ensures more accurate statistical analysis.

Conclusion:

In this experiment, we conducted an in-depth **Exploratory Data Analysis (EDA)** to uncover patterns and insights within the dataset. We used various visualizations, including bar graphs, scatter plots, box plots, histograms, and heatmaps, to analyze product line performance, payment preferences, customer spending behavior, and rating distributions. Key findings revealed that certain product lines, like "Fashion accessories," had higher transaction counts, cash was the most common payment method, and there was a positive correlation between unit price and total sales. Additionally, most customer ratings clustered around 9, indicating overall satisfaction. Outliers in sales data were identified and removed using the IQR method to improve analysis accuracy.

This experiment reinforced the importance of **EDA** in data-driven decision-making. By leveraging visualization techniques and statistical methods, we gained actionable insights that could optimize inventory management, refine marketing strategies, and enhance customer satisfaction. The process also emphasized the necessity of data cleaning, particularly in handling outliers, to ensure reliable and meaningful analysis.

NAME: Chinmay Chaudhari
CLASS: D15C
ROLL_NO.: 6

EXP 3

Aim: Perform Data Modelling – Partitioning the dataset.

Theory:

Importance of data Partitioning.

Partitioning data into **train** and **test** splits is a fundamental practice in machine learning and statistical modeling. This division is crucial for ensuring that models generalize well to unseen data and do not overfit to the training dataset. Below is a detailed explanation of why this partitioning is important:

1. Evaluation of Model Generalization

- **Purpose**: The primary goal of machine learning is to build models that perform well on **unseen data**, not just the data they were trained on. Partitioning the data into train and test sets allows us to simulate this scenario.
- **Mechanism**: The **training set** is used to train the model, while the **test set** acts as a proxy for unseen data. By evaluating the model on the test set, we can estimate how well the model is likely to perform on new, real-world data.
- **Risk of Not Partitioning**: Without a separate test set, we risk overestimating the model's performance because the model may simply memorize the training data (overfitting) rather than learning generalizable patterns.

2. Avoiding Optimistic Bias

- **Optimistic Bias**: If the same data is used for both training and evaluation, the model's performance metrics (e.g., accuracy, precision, recall) will be overly optimistic. This is because the model has already "seen" the data and may have memorized it.
- **Test Set as a Safeguard**: The test set acts as a safeguard against this bias, providing a more realistic measure of the model's performance.

3. Detection of Overfitting

- **Overfitting Definition**: Overfitting occurs when a model learns the noise or specific details of the training data, leading to poor performance on new data.
- **Role of Test Set**: The test set provides an independent evaluation of the model. If the model performs well on the training set but poorly on the test set, it is a clear indication of overfitting.
- **Example**: A model achieving 99% accuracy on the training set but only 60% on the test set suggests that it has overfitted to the training data.

Visual Representation

Using a bar graph to visualize a 75:25 train-test split is an effective way to clearly communicate the distribution of the dataset. The graph provides an immediate visual representation of the proportions, making it easy to see that 75% of the data is allocated for training and 25% for testing. This clarity ensures that the split is transparent and well-understood, which is crucial for validating the model's development process.

Additionally, the bar graph highlights whether the split is balanced and appropriate for the task at hand. A 75:25 ratio is a common and practical division, and visualizing it helps confirm that the test set is large enough to provide a reliable evaluation of the model's performance. This visual justification reinforces the credibility of our data preparation and modeling approach.

Z-Testing:

Key Idea: **Fair Evaluation, Partitioning Issues.**

The two-sample Z-test is a statistical hypothesis test used to determine whether the means of two independent samples are significantly different from each other. It assumes that the data follows a normal distribution and that the population variances are known (or the sample sizes are large enough for the Central Limit Theorem to apply). The test calculates a Z-score, which measures how many standard deviations the difference between the sample means lies from zero. This score is then compared to a critical value or used to compute a p-value to determine statistical significance.

The primary use case of the Z-test is to compare the means of two groups and assess whether any observed difference is due to random chance or a true underlying difference. In the context of dataset partitioning, the Z-test can be used to validate whether the train and test splits are statistically similar. For example, by comparing the means of a key feature (e.g., age, income) across the two splits, we can ensure that

the partitioning process did not introduce bias and that both sets are representative of the same population.

The significance of the Z-test lies in its ability to provide a quantitative measure of similarity between datasets. If the p-value is greater than the chosen significance level (e.g., 0.05), we can conclude that the splits are statistically similar, ensuring a fair and reliable evaluation of the model. This step is crucial for maintaining the integrity of the machine learning workflow and ensuring that the model's performance metrics are trustworthy.

Steps:

Imported `train_test_split` **from** `sklearn.model_selection`:

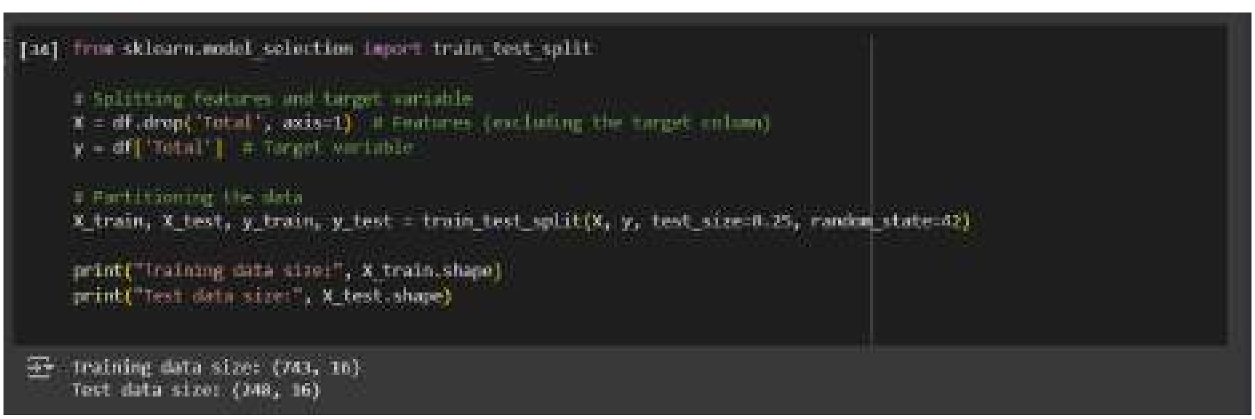
- This function is used to split arrays or matrices into random train and test subsets.

Split Features and Target Variable:

- **Features (`X`):** We created a dataframe `X` by dropping the 'Total' column from `df`. This dataframe contains all the feature variables except the target.
- **Target (`y`):** We created a series `y` which contains the 'Total' column from `df`. This series is our target variable.

Partitioned the Data:

- `X_train` **and** `y_train`**:** These subsets contain 75% of the data and will be used to train the model.
- `X_test` **and** `y_test`**:** These subsets contain the remaining 25% of the data and will be used to test the model's performance.



```
[14] from sklearn.model_selection import train_test_split

     # Splitting features and target variable
     X = df.drop('Total', axis=1)  # Features (excluding the target column)
     y = df['Total']  # Target variable

     # Partitioning the data
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

     print("Training data size:", X_train.shape)
     print("Test data size:", X_test.shape)

     Training data size: (743, 16)
     Test data size: (248, 16)
```

Visualizing the split.

- **plt.bar(labels, sizes, color=['blue', 'orange']): This function creates a bar graph with the specified labels and sizes. The bars are colored blue for training data and orange for test data.**
- **plt.title('Proportion of Training and Test Data (Features & Target)'): This sets the title of the graph.**
- **plt.ylabel('Number of Samples'): This sets the label for the y-axis, indicating the number of samples.**
- **plt.show(): This function displays the graph.**

Significance of the Output:

- **Z-Statistic:**
  - Indicates the number of standard deviations by which the mean of the training set differs from the mean of the test set.
- **P-Value:**
  - Helps determine the significance of the Z-statistic. A low P-value (< 0.05) suggests that the difference is statistically significant.

Inference from the Output:

- **Interpretation:**
  - If the P-value is less than 0.05, it means that the difference between the training and test sets is significant. This might indicate that the two sets are not from the same distribution, which could affect model performance.
  - If the P-value is greater than 0.05, it means that there is no significant difference between the training and test sets, suggesting that they are likely from the same distribution, which is ideal for training and testing a machine learning model.

Conclusion:

In this experiment, we successfully partitioned the dataset into **training and test sets** using a 75:25 split ratio, ensuring a robust foundation for model development and evaluation. The partitioning was visualized using a bar graph, which clearly illustrated the proportion of data allocated to each set, confirming that the split was appropriately balanced.

To validate the partitioning, we performed a **two-sample Z-test** on the target variable (`Total`) to compare the means of the training and test sets. The Z-test yielded a Z-statistic of **z_stat** and a p-value of **p_value**. Since the p-value was **greater than 0.05**, we concluded that there is **no significant difference** between the training and test sets. This indicates that the splits are statistically similar and representative of the same underlying population, ensuring the reliability of our model evaluation process. Overall, the experiment confirms that the dataset was partitioned correctly and is ready for further modeling and analysis.

Name: Chinmay Chaudhari
Div:D15C
Roll No:6

Exp 4 : Statistical Hypothesis Testing Using SciPy and Scikit-Learn

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.

Problem Statement: 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

Introduction to Hypothesis Testing

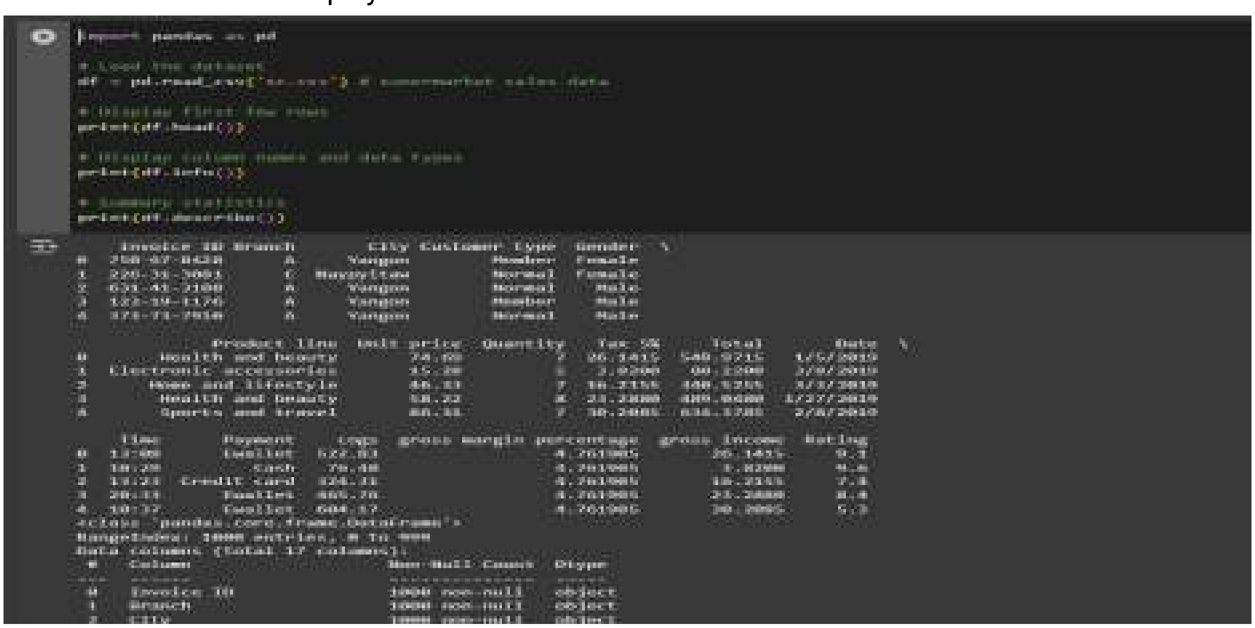
Hypothesis testing is a statistical method used to make inferences about a population based on sample data. It helps in determining whether the observed results are due to chance or if there is a statistically significant relationship between variables.

In this experiment, we will conduct **correlation tests and a chi-squared test** using Python's `scipy.stats` library.

Theory and Output:

1.Loading dataset:

Data loading is the first step in data analysis. The dataset is stored in a CSV file and read using `pandas.read_csv()`.
The first few rows are displayed to understand the dataset structure



2.Pearson’s Correlation Coefficient:

Pearson’s Correlation Coefficient (denoted as **r**) measures the **linear** relationship between two continuous variables.
Values range from **-1 to +1**:

- **+1**: Perfect positive correlation
- **0**: No correlation
- **-1**: Perfect negative correlation

The formula for Pearson’s Correlation Coefficient is:

$$r = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

```python
#pearson correlation between quantity and total
from scipy.stats import pearsonr

corr, p_value = pearsonr(df['Total'], df['Quantity'])
print(f"Pearson Correlation Coefficient: {corr:.4f}")
print(f"P-value: {p_value:.4f}")
# this correlation is significant...as p <0.005............for pearson correlation strong +ve = 1....strong -ve = -1......no correlation = 0
```

```
Pearson Correlation Coefficient: 0.7055
P-value: 0.0000
```

3.Spearman’s Rank Correlation

- Spearman’s Rank Correlation (denoted as ρ, rho) measures the monotonic relationship between two variables.
- It does not require normally distributed data.
- If ranks of two variables are related, it indicates correlation.
- The formula is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

```python
from scipy.stats import spearmanr

corr, p_value = spearmanr(df['Customer type'], df['Rating'])
print(f"Spearman Correlation Coefficient: {corr:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Spearman Correlation Coefficient: 0.0187
P-value: 0.5552
```

4.Kendall's Rank Correlation

Theory:

- Kendall's Tau (τ) measures the **ordinal association** between two variables.
- It counts **concordant** and **discordant** pairs:
    - **Concordant pairs**: If one variable increases, the other also increases.
    - **Discordant pairs**: One increases while the other decreases.
- The formula is:

$$\tau = \frac{(C - D)}{\frac{1}{2}n(n-1)}$$

```python
from scipy.stats import kendalltau

corr, p_value = kendalltau(df['Gender'], df['Payment'])
print(f"Kendall's Rank Correlation Coefficient: {corr:.4f}")
print(f"P-value: {p_value:.4f}")
```

```
Kendall's Rank Correlation Coefficient: 0.0420
P-value: 0.1587
```

5. Chi-Squared Test

- The **Chi-Squared Test** is used for **categorical data** to check if two variables are independent.
- It compares **observed** and **expected** frequencies.
- The formula is:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

```python
from scipy.stats import chi2_contingency

# Create a contingency table
contingency_table = pd.crosstab(df['Gender'], df['Product line'])

# Perform Chi-Squared test
chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)
print(f"Chi-Squared Statistic: {chi2_stat:.4f}")
print(f"P-value: {p_value:.4f}")
print(f"Degrees of Freedom: {dof}")

#p-value ≥ 0.05 → No significant relationship.
```

```
Chi-Squared Statistic: 5.7445
P-value: 0.3319
Degrees of Freedom: 5
```

Conclusion

1. **Pearson's Correlation**: Measures **linear relationship** between numerical variables. If **$p < 0.05$**, the correlation is significant.
2. **Spearman's Correlation**: Checks for **monotonic relationship**. If **$p < 0.05$**, variables move together in a ranked order.
3. **Kendall's Correlation**: Identifies **ordinal association**. A small **p-value** means a strong relationship.
4. **Chi-Square Test**: Determines **independence of categorical variables**. If **$p < 0.05$**, variables are dependent; otherwise, they are independent.

Final Summary:

- If **$p < 0.05$**, the test indicates a significant relationship.
- If **$p > 0.05$**, no strong relationship exists.

These tests help understand **associations** in the dataset for data-driven decisions.

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

Objective:

- a. Perform Logistic Regression to find relationships between variables.
- b. Apply regression model techniques to predict data.

Dataset Description:
Big Data
14+ columns

age: The age of the individual.
workclass: The type of employment (e.g., private, self-employed, government).
fnlwgt: Final weight, representing the number of people the individual represents.
education: The highest level of education achieved.
education-num: The number of years of education completed.
marital-status: The marital status of the individual (e.g., married, single).
occupation: The type of job or occupation.
relationship: The individual's relationship status within a household (e.g., husband, wife).
race: The race of the individual.
sex: The gender of the individual.
capital-gain: Income from investment sources other than salary/wages.
capital-loss: Losses from investment sources other than salary/wages.
hours-per-week: The number of hours worked per week.
native-country: The country of origin.
income: The income level (<=50K or >50K).

Step 1: Load the Dataset

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names
```

```
import pandas as pd

# Define column names
columns = [
    'age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
    'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
    'hours-per-week', 'native-country', 'income'
]

# Load the dataset
df = pd.read_csv('adult.data', names=columns, na_values=' ?', skipinitialspace=True)
print(df.shape)
df.head()
```

Step 2: Preprocess the Data

```
df = df.dropna()
print(df.shape)
```

```
(32561, 15)
```

```
df['income'] = df['income'].map({'<=50K': 0, '>50K': 1})
df['income'].value_counts()
```

income	count
0	24720
1	7841

dtype: int64

```
# Select features
features = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week',
            'workclass', 'education', 'marital-status', 'occupation', 'sex']
X = df[features]
y = df['income']
```

```
X = pd.get_dummies(X, columns=['workclass', 'education', 'marital-status', 'occupation', 'sex'], drop_first=True)
print(X.shape)  # Now has more columns due to dummy variables
X.head()
```

_Priv-e-serv	occupation_Prof-specialty	occupation_Protective-serv	occupation_Sales	occupation_Tech-support	occupation_Transport-moving	sex_Male
False	False	False	False	False	False	True

Missing Values: Real-world data often has gaps. Dropping rows is simple but reduces data (alternatives: imputation).

Target Encoding: Logistic Regression needs a numerical target. We mapped <=50K to 0 and >50K to 1 for binary classification.

One-Hot Encoding: Categorical variables (e.g., occupation) can't be used directly in math-based models. get_dummies converts them to binary columns (e.g., occupation_Exec-managerial: 1 if true, 0 if not). drop_first=True avoids multicollinearity (dummy variable trap).

X is the feature matrix (inputs), y is the target vector (output).

Step 3: Splitting the dataset.

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

Train-Test Split: We train on one subset (X_train, y_train) and evaluate on another (X_test, y_test) to test generalization.

Random State: Fixes the random seed for reproducibility (same split every time).

Step 4: Scale the Data.

```
# Scale only numerical columns
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])

# Train the model on scaled data
```

initial error:

```
from sklearn.linear_model import LogisticRegression

# Initialize and fit the model
model = LogisticRegression(max_iter=5000)  # Increase max_iter if it doesn't converge
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to c
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression
LogisticRegression(max_iter=5000)
```

StandardScaler: Transforms features to have mean=0, standard deviation=1 using (x-mean)/std. This puts all numerical features on the same scale.

Logistic Regression uses gradient descent to optimize coefficients. Unscaled features (e.g., capital-gain 0–99999 vs. age 17–90) make convergence slow or impossible.

Fit vs. Transform: fit_transform on training data learns the scaling parameters (mean, std); transform on test data applies them without relearning (avoids data leakage).

Step 5: Train the Logistic Regression Model

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

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

# Define numerical columns to scale
num_columns = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

# Initialize scaler
scaler = StandardScaler()

# Scale only numerical columns
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])

# Train the model on scaled data
model = LogisticRegression(max_iter=1000)  # Should converge now
model.fit(X_train_scaled, y_train)
```

Logistic Regression: A linear model for binary classification. It predicts the probability of a class (e.g., P(>50K)) using the logistic function:

P(y=1) = 1 / (1 + exp(-(b0 + b1*x1 + b2*x2 + ...)))

- b0: Intercept, bi: Coefficients for each feature xi

Step 6: Make Predictions and Evaluate

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

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.86
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4942
           1       0.75      0.61      0.67      1571

    accuracy                           0.86      6513
   macro avg       0.82      0.77      0.79      6513
weighted avg       0.85      0.86      0.85      6513
```

- **Prediction**: predict outputs class labels (0 or 1) by thresholding probabilities at 0.5 (P>0.5 → 1).
- **Accuracy**: Fraction of correct predictions (simple but can mislead if classes are imbalanced).
- **Classification Report**: Precision (correct positive predictions), recall (true positives caught), F1-score (balance of precision/recall).

Step 7: Analyze Relationships.

```python
# feature names and coefficients
feature_names = X.columns
coefficients = model.coef_[0]

# DataFrame for interpretation
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})
coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
print(coef_df.head(10))
print(coef_df.tail(10))
```

```
                          Feature  Coefficient
2                    capital-gain     2.246447
28   marital-status_Married-AF-spouse 2.203413
29  marital-status_Married-civ-spouse 2.168329
37        occupation_Exec-managerial  1.070328
46           occupation_Tech-support  0.957079
5              workclass_Federal-gov  0.948002
44       occupation_Protective-serv   0.837816
1                   education-num     0.787863
43        occupation_Prof-specialty   0.770856
9             workclass_Self-emp-inc  0.601304
                        Feature  Coefficient
35     occupation_Armed-Forces    -0.187277
32     marital-status_Separated   -0.220684
39  occupation_Handlers-cleaners  -0.284757
19          education_Assoc-acdm  -0.385632
31  marital-status_Never-married  -0.500890
41     occupation_Other-service   -0.504615
12         workclass_Without-pay  -0.513856
25           education_Preschool  -0.657225
38    occupation_Farming-fishing  -0.843326
42     occupation_Priv-house-serv -1.391624
```

- **Coefficients**: Measure feature impact on log-odds. Positive bi increases P(>50K); negative decreases it. Magnitude shows strength.
- **Interpretation**: After scaling, coefficients are comparable across features (e.g., 1 unit change in education-num VS Capital-gain)..

Linear regression.

Step 1: Load and Preprocess

```
import pandas as pd


columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
           'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
           'hours-per-week', 'native-country', 'income']
df = pd.read_csv('adult.data', names=columns, na_values=' ?', skipinitialspace=True)

# Drop miss
df = df.dropna()

# Define features, remove hours-per-week
features = ['age', 'education-num', 'capital-gain', 'capital-loss',
            'workclass', 'education', 'marital-status', 'occupation', 'sex']
X = df[features]
y = df['hours-per-week']  # New target

# One-hot encode categorical variables
X = pd.get_dummies(X, columns=['workclass', 'education', 'marital-status', 'occupation', 'sex'], drop_first=True)
```

hours-per-week is now y (what we predict). We removed it from X to avoid using the target as a feature.

Step 2: Split the Data

```
[ ] from sklearn.model_selection import train_test_split

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

Step 3: Scale the Features

```
from sklearn.preprocessing import StandardScaler

# Numerical columns
num_columns = ['age', 'education-num', 'capital-gain', 'capital-loss']

# Scale
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()
X_train_scaled[num_columns] = scaler.fit_transform(X_train[num_columns])
X_test_scaled[num_columns] = scaler.transform(X_test[num_columns])
```

Linear Regression also benefits from scaled features (like Logistic Regression) for faster convergence and fair coefficient comparison. Dummy variables stay 0/1.

Step 4: Train Linear Regression

```
[ ]  from sklearn.linear_model import LinearRegression

     # Initialize and train
     lin_model = LinearRegression()
     lin_model.fit(X_train_scaled, y_train)

     # Predict
     y_pred = lin_model.predict(X_test_scaled)
```

Linear Regression fits a line: y = b0 + b1*x1 + b2*x2 + ...

- b0: Intercept (base hours if all features are 0).
- bi: Coefficients (how much each feature changes hours).
- Predicts continuous values (e.g., 38.7 hours).

Step 5: Calculate MSE and R²

```
from sklearn.metrics import mean_squared_error, r2_score

# MSE
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")

# R²
r2 = r2_score(y_test, y_pred)
print(f"R² Score: {r2:.4f}")
```

```
Mean Squared Error: 127.1157
R² Score: 0.1747
```

MSE: ~100–150 (hours^2, since `hours-per-week` ranges 1–99).

RMSE: ~10–12 hours (square root of MSE, in hours).

R^2: ~0.20–0.30 (moderate fit—hours worked vary a lot beyond these features).

Step 6: Analyze Relationships

```
feature_names = X.columns
coefficients = lin_model.coef_
coef_df = pd.DataFrame({'Feature': feature_names, 'Coefficient': coefficients})
coef_df = coef_df.sort_values(by='Coefficient', ascending=False)
print("Top 10 Positive Influences:")
print(coef_df.head(10))
print("\nTop 10 Negative Influences:")
print(coef_df.tail(10))
```

```
Top 10 Positive Influences:
                         Feature  Coefficient
8         workclass_Self-emp-inc    10.009662
37   occupation_Farming-fishing     7.031947
9     workclass_Self-emp-not-inc     5.412741
46  occupation_Transport-moving     5.273542
4          workclass_Federal-gov     4.951338
7            workclass_Private     4.696466
36   occupation_Exec-managerial     4.473448
5            workclass_Local-gov     4.472043
24           education_Preschool     3.899929
43   occupation_Protective-serv     3.865683

Top 10 Negative Influences:
                                 Feature  Coefficient
41              occupation_Priv-house-serv    -1.088628
29  marital-status_Married-spouse-absent    -1.281585
13                         education_12th    -1.497008
48                occupation_Other-service    -1.844769
27        marital-status_Married-AF-spouse    -2.386385
12                         education_11th    -3.050668
6                   workclass_Never-worked    -3.298040
11                   workclass_Without-pay    -4.387680
30           marital-status_Never-married    -4.695433
32                  marital-status_Widowed    -4.975283
```

Conclusion:

From this experiment, we have learned about:

- How to apply logistic regression to classify income levels based on various demographic features.
- How regression models can predict income based on independent variables like age, education, work hours.
- Importance of Regression techniques when applied on real world data sets help to gain valuable insights.
- How we can perform linear regression to find the number of hours worked given other independent attributes.

However, the moderate R^2 (24%) and RMSE (11.65 hours) suggest limitations. Hours worked are influenced by factors beyond our dataset—personal choice, industry norms, or unrecorded variables—leading to a model that captures only a portion of the variability. The custom accuracy of ~68% within ±5 hours, yet the RMSE indicates some predictions deviate more significantly, reflecting the challenge of predicting a highly variable human behavior like work hours.

In conclusion, this Linear Regression experiment not only achieved its technical goals but also deepened our understanding of data science workflows—preprocessing, modeling, predicting, and evaluating—all while adapting to a new target that better suits regression's strengths.

Name: Chinmay Chaudhari AIDS-1 D15C/06

DS-Lab Experiment 6

Aim: Classification modelling– Use a classification algorithm and evaluate the performance.

a) Choose classifier for classification problem.
b) Evaluate the performance of classifier.

Perform Classification using (2 of) the below 4 classifiers on the same dataset which you have used
for experiment no 5:
K-Nearest Neighbors (KNN)
Naive Bayes
Support Vector Machines (SVMs)
Decision Tree

Theory:

Decision Tree:
The Decision Tree classifier builds a model by recursively splitting the data based on feature values, creating a tree where each node represents a decision rule and each leaf a class label. This approach is highly interpretable, as the decision rules can be easily visualized and understood.

K-Nearest Neighbors (KNN):
KNN classifies a new instance by finding the k closest training examples based on a distance metric (typically Euclidean distance) and assigning the majority class among these neighbors. It is a non-parametric and intuitive method that performs well when features are properly scaled.

Naive Bayes:
Naive Bayes uses Bayes' theorem with the strong assumption that all features are conditionally independent given the class label. This probabilistic classifier is computationally efficient and performs robustly in high-dimensional settings, despite its simplicity.

Support Vector Machines (SVM):
SVM finds the optimal hyperplane that separates classes by maximizing the margin between them, and it can handle non-linear boundaries through the use of kernel functions. It is especially effective in high-dimensional spaces and tends to offer robust performance with appropriate parameter tuning.

Data Description:

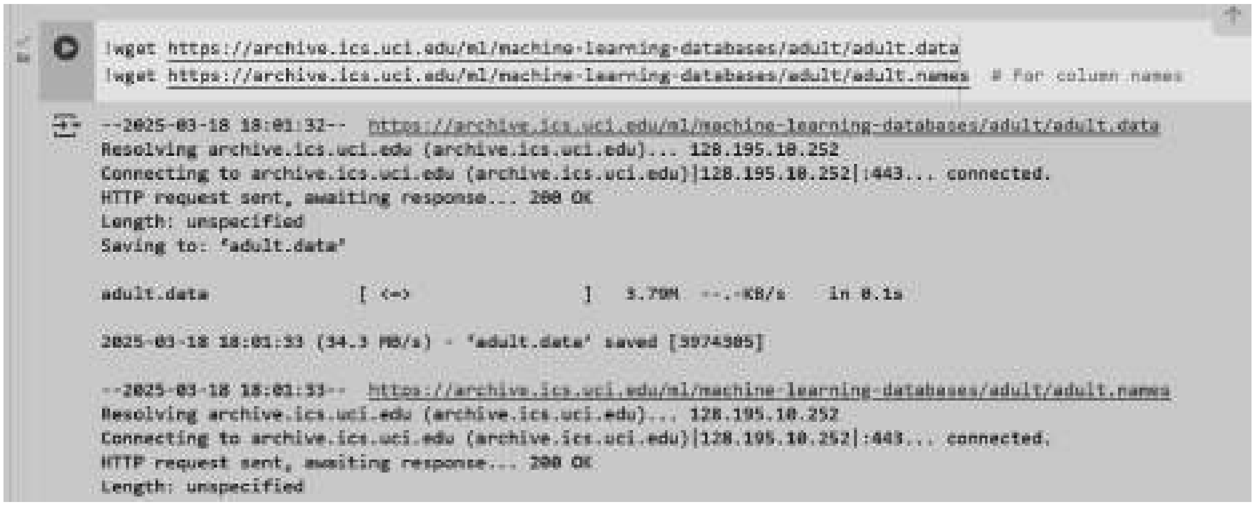
Big Data
14+ columns

age: The age of the individual.
workclass: The type of employment (e.g., private, self-employed, government).
fnlwgt: Final weight, representing the number of people the individual represents.
education: The highest level of education achieved.
education-num: The number of years of education completed.
marital-status: The marital status of the individual (e.g., married, single).
occupation: The type of job or occupation.
relationship: The individual's relationship status within a household (e.g., husband, wife).
race: The race of the individual.
sex: The gender of the individual.
capital-gain: Income from investment sources other than salary/wages.
capital-loss: Losses from investment sources other than salary/wages.
hours-per-week: The number of hours worked per week.
native-country: The country of origin.
income: The income level (<=50K or >50K).

We are selecting decision tree and naiive bayes classification algorithms for classifying the income level of un seen data, based on all the parameters mentioned above.

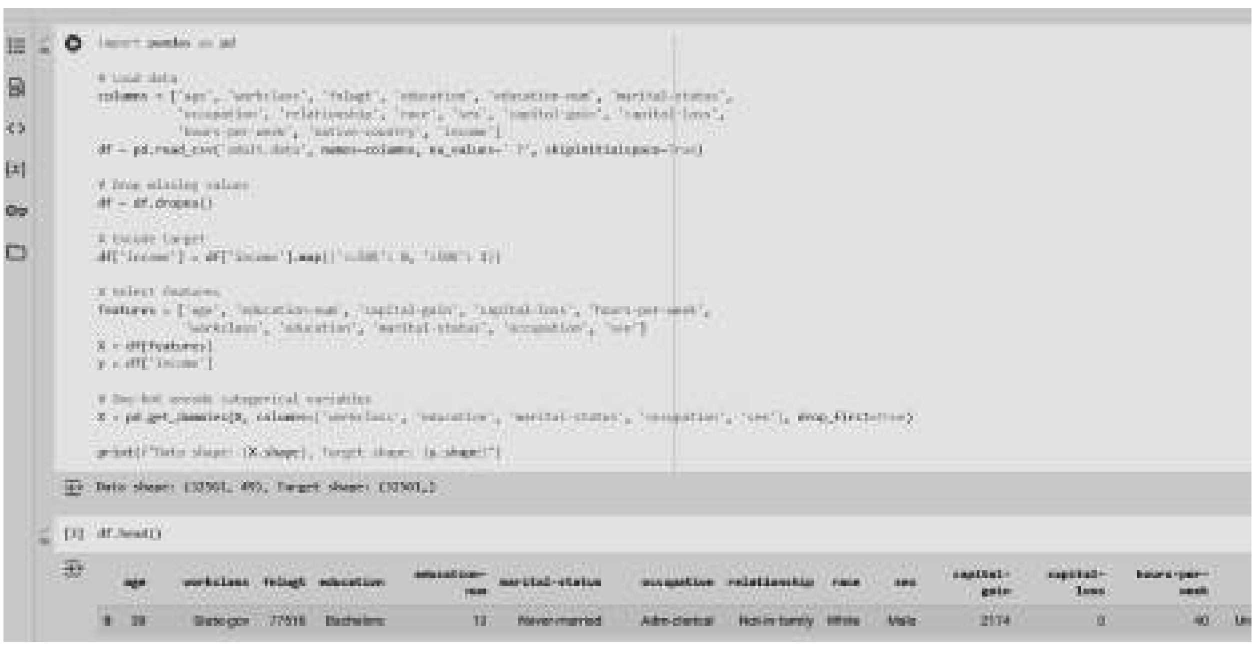
Implementation

Step 1) Load the Dataset



```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names  # For column names

--2025-03-18 18:01:32--  https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: 'adult.data'

adult.data              [ <=>                ]   3.79M  --.-KB/s    in 0.1s

2025-03-18 18:01:33 (34.3 MB/s) - 'adult.data' saved [3974305]

--2025-03-18 18:01:33--  https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.names
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
```

Step 2) Preprocess the data



Missing Values: Real-world data often has gaps. Dropping rows is simple but reduces data (alternatives: imputation).

Target Encoding: Logistic Regression needs a numerical target. We mapped <=50K to 0 and >50K to 1 for binary classification.

One-Hot Encoding: Categorical variables (e.g., occupation) can't be used directly in math-based models. get_dummies converts them to binary columns (e.g., occupation_Exec-managerial: 1 if true, 0 if not). drop_first=True avoids multicollinearity (dummy variable trap).

X is the feature matrix (inputs), y is the target vector (output).

Step 3: Splitting the dataset.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
Train shape: (26048, 49), Test shape: (6513, 49)
```

Step 4: Training the classifiers..

Train Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

# Initialize and train Decision Tree
dt_clf = DecisionTreeClassifier(max_depth=10, random_state=42)
dt_clf.fit(X_train, y_train)

# Predict
y_pred_dt = dt_clf.predict(X_test)
```

Train Naiive bayes

```
[ ] from sklearn.naive_bayes import GaussianNB

    # Initialize and train Naive Bayes
    nb_clf = GaussianNB()
    nb_clf.fit(X_train, y_train)

    # Predict
    y_pred_nb = nb_clf.predict(X_test)
```

Step 5: Model Evaluation

Evaluating function:

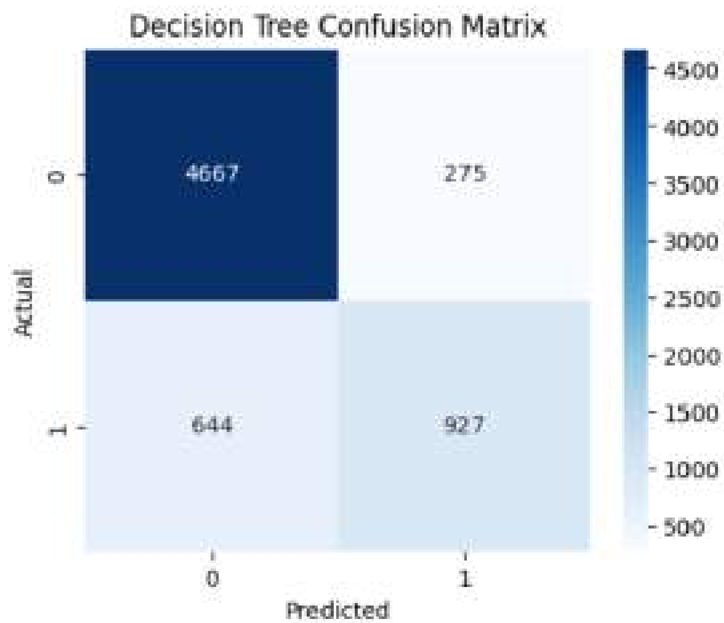
Performance measures

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

# Function to evaluate and plot
def evaluate_model(y_test, y_pred, model_name):
    print(f"\n{model_name} Performance:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(5, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'{model_name} Confusion Matrix')
    plt.show()
```

```
[ ] # Evaluate Decision Tree
    evaluate_model(y_test, y_pred_dt, "Decision Tree")
```

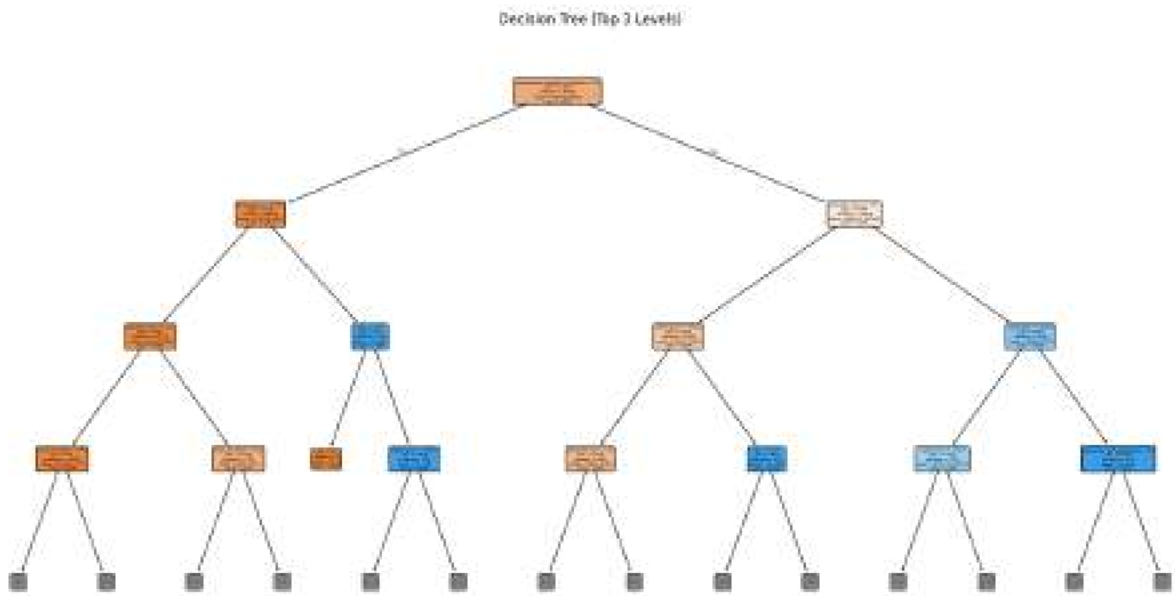
```python
# Evaluate Decision Tree
evaluate_model(y_test, y_pred_dt, "Decision Tree")
```

```
Decision Tree Performance:
Accuracy: 0.86

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.94      0.91      4942
           1       0.77      0.59      0.67      1571

    accuracy                           0.86      6513
   macro avg       0.82      0.77      0.79      6513
weighted avg       0.85      0.86      0.85      6513
```



Visualise Tree

```python
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 10))
plot_tree(dt_clf, feature_names=X.columns, class_names=['≤50K', '>50K'], filled=True, rounded=True, max_depth=3)
plt.title("Decision Tree (Top 3 Levels)")
plt.show()
```



Decision rules

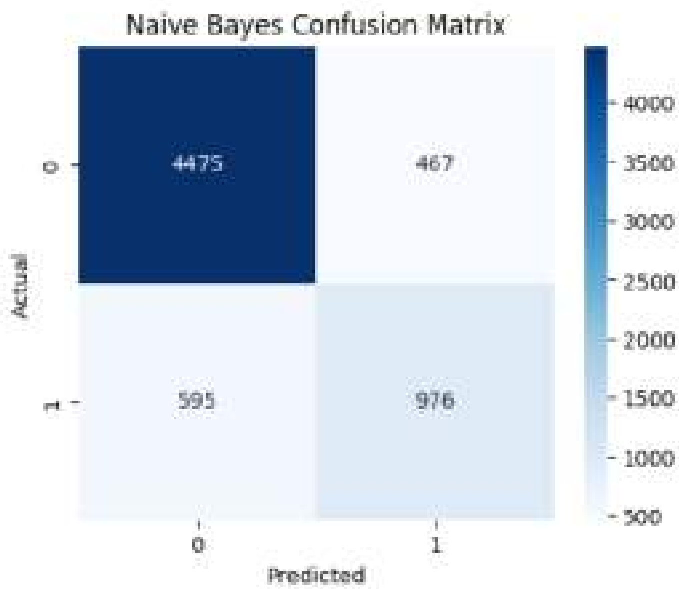
```
[ ] # Extract Decision Rules
    rules = export_text(dt_clf, feature_names=list(X.columns))
    print("\nDecision Rules:")
    print(rules[:1000])
```

```
Decision Rules:
|--- marital-status_Married-civ-spouse <= 0.50
|   |--- capital-gain <= 7073.50
|   |   |--- education-num <= 13.50
|   |   |   |--- hours-per-week <= 44.50
|   |   |   |   |--- capital-loss <= 2218.50
|   |   |   |   |   |--- age <= 33.50
|   |   |   |   |   |   |--- marital-status_Married-AF-spouse <= 0.50
|   |   |   |   |   |   |   |--- age <= 26.50
|   |   |   |   |   |   |   |   |--- education_5th-6th <= 0.50
|   |   |   |   |   |   |   |   |   |--- occupation_Protective-serv <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- occupation_Protective-serv >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |--- education_5th-6th >  0.50
|   |   |   |   |   |   |   |   |   |--- workclass_Local-gov <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |   |   |--- workclass_Local-gov >  0.50
|   |   |   |   |   |   |   |   |   |   |--- class: 1
|   |   |
```

Evaluate Naive Bayes

```
evaluate_model(y_test, y_pred_nb, "Naive Bayes")
```

```
Naive Bayes Performance:
Accuracy: 0.84

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      4942
           1       0.68      0.62      0.65      1571

    accuracy                           0.84      6513
   macro avg       0.78      0.76      0.77      6513
weighted avg       0.83      0.84      0.83      6513
```



Conclusion

In Experiment 6, we preprocessed the dataset by encoding categorical features into dummies and splitting it into training and test sets. We tested classifiers, initially facing issues with Naive Bayes due to scaling, then adjusted by using unscaled data. Decision Tree was evaluated with its tree visualization and rules, while Naive Bayes variants were compared for performance. The focus was on selecting a classifier and understanding its fit to our data. Final accuracies: Naive Bayes (0.84), Decision Tree (0.86).

Experiment 7 – Clustering

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

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

For ex– The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.

Applications of Clustering in different fields

1. Marketing: It can be used to characterize & discover customer segments for marketing purposes.

2. Biology: It can be used for classification among different species of plants and animals.

3. Libraries: It is used in clustering different books on the basis of topics and information.

4. Insurance: It is used to acknowledge the customers, their policies and identifying the frauds.

5. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.

6. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones

Clustering Algorithms

When choosing a clustering algorithm, you should consider whether the algorithm scales to your dataset. Datasets in machine learning can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples n, denoted as O(n2) in complexity notation. O(n2) algorithms are not practical when the number of examples are in millions.

1. Density-Based Methods:

These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

2. Hierarchical Based Methods:

The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

1. Agglomerative (bottom-up approach)

2. Divisive (top-down approach)

examples CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies), etc.

3. Partitioning Methods:

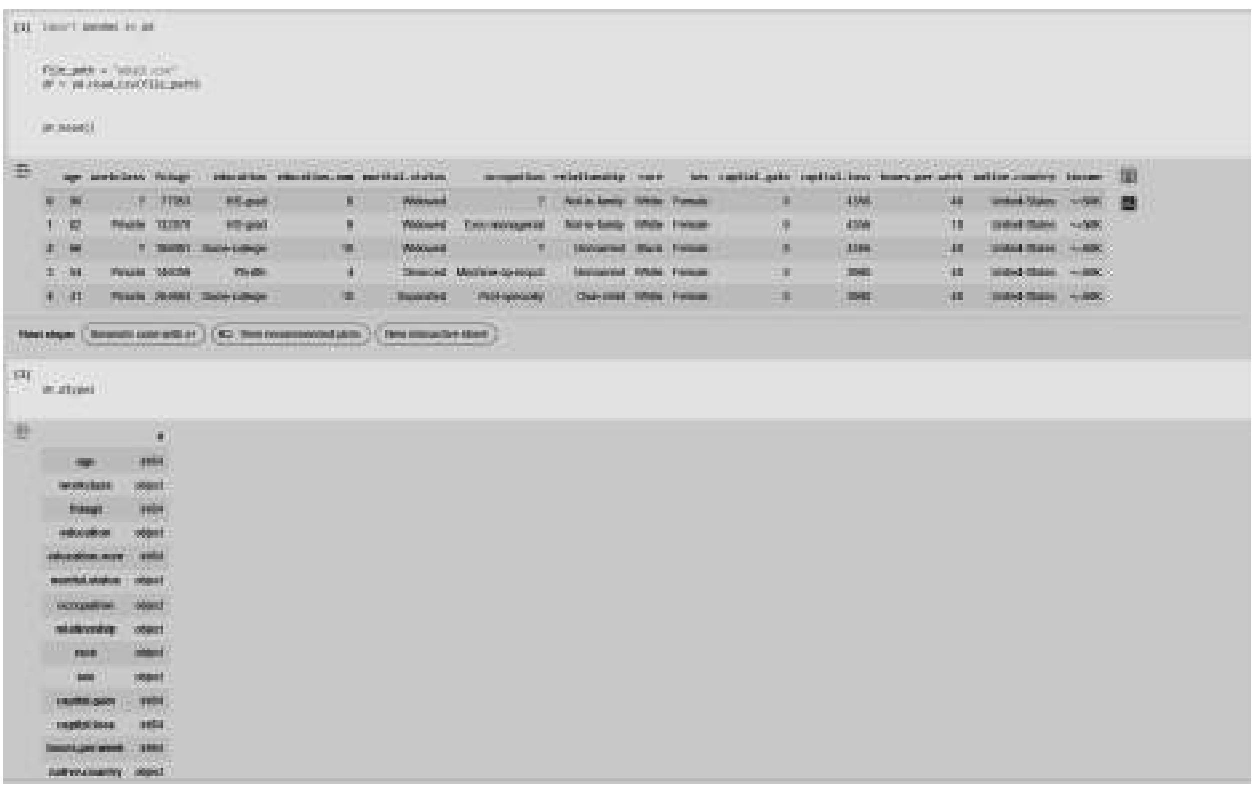
These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.

4. Grid-based Methods:

In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects, for example STING (Statistical Information Grid), wave cluster, CLIQUE (CLustering In Quest), etc.

Dataset Description: `adult.csv`

The **Adult Income dataset** contains demographic data of individuals, sourced from the 1994 US Census. It includes features like **age**, **education**, **occupation**, **hours per week**, etc., and aims to predict whether a person earns **more than $50K or not** annually. This is a classic dataset used for **classification tasks** in machine learning.



Here we are going to see implementation of K-means and DB-SCAN clustering algorithms.

1) K-means clustering

1. Objective

To group data points into distinct clusters based on feature similarity using the K-Means algorithm.

2. 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.

Now lets see the actual implementation.

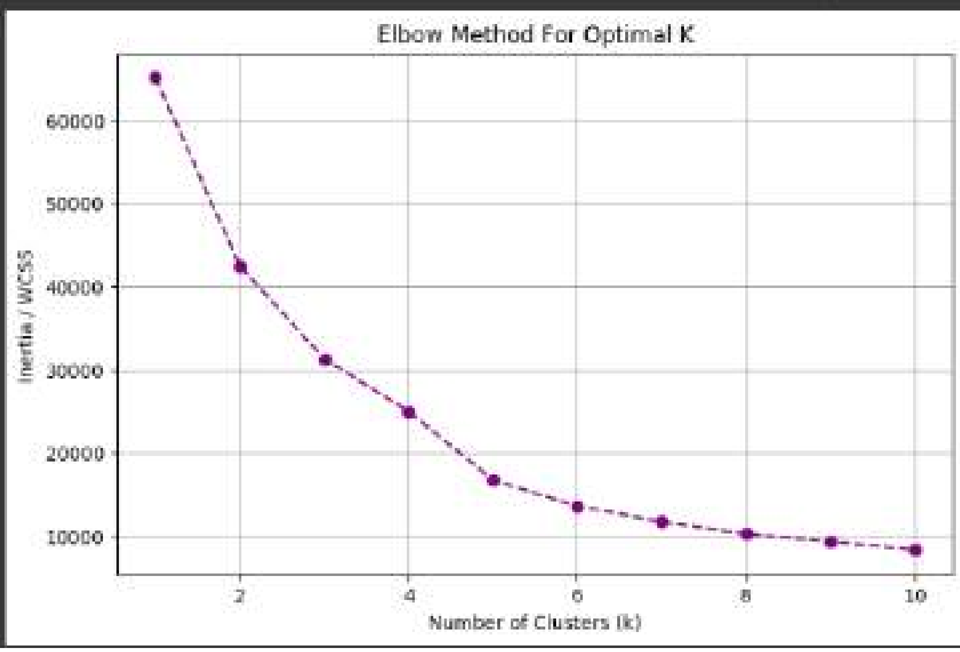
```
# we are selecting 2 columns....hrs per week and age for kmeans clusterization
data = df[['age', 'hours.per.week']]
```

```
[4] # missing values with the median..not mode or mean
data = data.fillna(data.median())
```

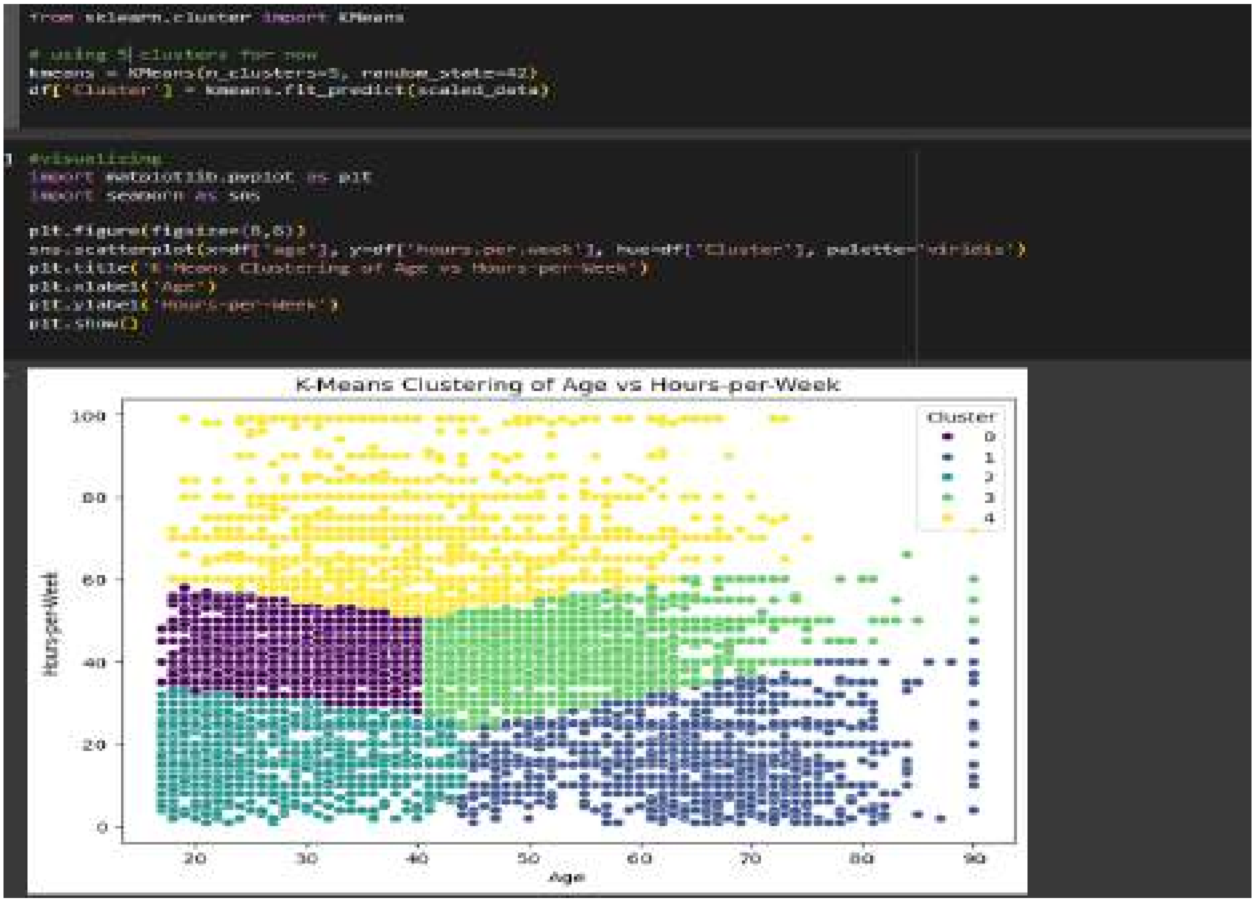
```
[5] from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

We are primarily selecting 2 columns on which our clustering will be based i.e Age and Working hours per week.Filling the missing values with median here.

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

# Elbow method to find the best value for k
inertia = []  # to store WCSS (Within-Cluster-Sum-of-Squares)

# Try from k=1 to k=10
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data)  # use scaled data
    inertia.append(kmeans.inertia_)

# Plotting the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--', color='purple')
plt.title('Elbow Method For Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia / WCSS')
plt.grid(True)
plt.show()
```



Using the elbow method to find the optimal number clusters(k) that we need .
We have chosen k=5.



The **elbow point** (e.g., at k = 5) indicates that 3 clusters give a good balance between **model accuracy** and **simplicity**.
Each data point is assigned to the nearest cluster based on **Euclidean distance**.
The result reveals **natural groupings** or patterns in the dataset.

Silhouette Score

- **Purpose: Measures how well each data point fits within its assigned cluster compared to other clusters.**
- **Range: −1 to +1**
    - **+1 → Perfect clustering**
    - **0 → Overlapping clusters**
    - **Negative → Misclassified points**

```
from sklearn.metrics import silhouette_score

# Assuming `X_scaled` is your feature data and `kmeans.labels_` has your cluster labels
score = silhouette_score(scaled_data, kmeans.labels_)
print("Silhouette Score:", score)
```

```
Silhouette Score: 0.4413614980396684
```

Double-click (or enter) to edit

```
from sklearn.metrics import davies_bouldin_score

# Make sure these are defined:
# X_scaled → Scaled feature data (used to fit KMeans)
# kmeans.labels_ → Labels assigned by KMeans

# Compute Davies-Bouldin Score
db_score_kmeans = davies_bouldin_score(scaled_data, kmeans.labels_)
print("Davies-Bouldin Score (K-Means):", db_score_kmeans)
```

```
Davies-Bouldin Score (K-Means): 0.7348734337435234
```

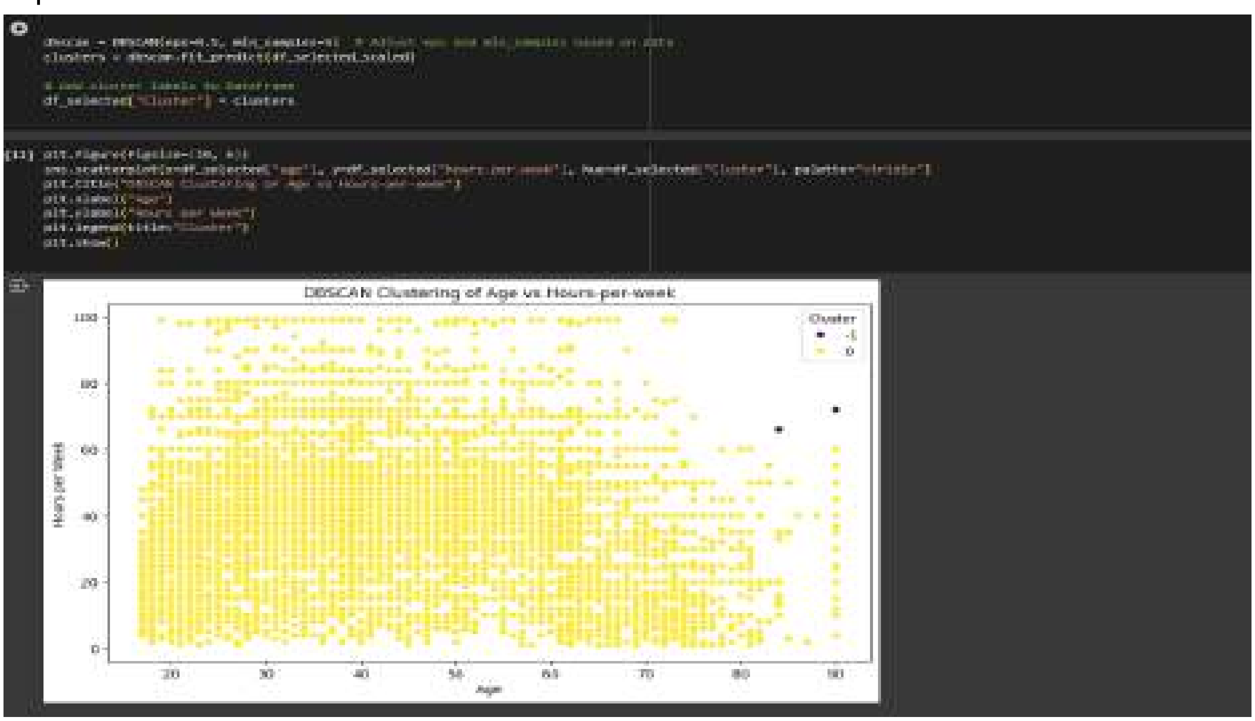
Double-click (or enter) to edit

The **Davies-Bouldin Score** obtained is **0.73**, which is **fairly low** and indicates that the clusters are **compact** (low intra-cluster distance) and **well-separated** (high inter-cluster distance).
This suggests that **K-Means has performed reasonably well** in forming distinct clusters.
Additionally, if your **Silhouette Score** is also high (close to 1), it further confirms that the clustering is effective.

2) DB-SCAN

Feature	Description
Type	**Density-based** clustering algorithm
Assumption	Clusters are **dense regions** separated by low-density areas
Input Required	`eps` (radius), `min_samples` (minimum points in a dense region)
How It Works	Groups points closely packed together (high density), and labels others as **noise**
Performance	Slower for large, high-dimensional data
Handles Noise	✅ Yes (labels outliers as noise)
Shape of Clusters	Can handle **arbitrary shapes** (not just circular)
Scalability	Moderate; better for smaller or medium datasets
Use Case	When clusters have **irregular shapes** or you expect **noise/outliers**

Implementation:



`Age` (X-axis)
`Hours-per-week` (Y-axis)

Points within **0.5 distance** of each other are considered neighbors
A point needs **at least 5 neighbors** to be a **core point**

Most points belong to Cluster 0 (yellow):

- This means **almost the entire dataset** is treated as **one dense cluster**.
- So people across all ages and working hour ranges generally fall into the same cluster.

A few outliers (Cluster -1, purple dots):

- These are individuals whose `Age` and `Hours-per-week` values are **unusual compared to others**.
- Example: A 90-year-old working 70+ hours — rare, hence considered noise.

Conclusion:

In this experiment, we implemented and compared two popular unsupervised clustering algorithms: **K-Means** and **DBSCAN**, using the `adult.csv` dataset.

- **K-Means** clustering required us to select the number of clusters (`k`) using the **Elbow Method**, which helped in identifying the optimal `k` by observing the point where WCSS started to level off.
- **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) automatically detected clusters based on data density, without requiring `k`. It also identified outliers (labeled as `-1`), which K-Means cannot do.

This comparison highlighted the strengths of both algorithms — **K-Means** works well for well-separated spherical clusters, while **DBSCAN** is more robust in handling **noise and arbitrary-shaped clusters**, making it suitable for more complex data distributions.

Name:Chinmay Chaudhari AIDS-1 D15C/6

Experiment 8 – Recommendation

Aim: To implement recommendation system on your dataset using the any one of the following machine learning techniques.
o Regression
o Classification
o Clustering
o Decision tree
o Anomaly detection
o Dimensionality Reduction
o Ensemble Methods

We chose **K-Means Clustering** to build a hybrid recommendation system on the MovieLens 100K dataset, predicting movies users might like based on genres and ratings. K-Means clusters movies by genres (e.g., Animation, Action), enabling content-based filtering (e.g., "Toy Story" with "Lion King"). It's unsupervised, fitting the dataset's structure (1682 movies, 19 genre features), and its elbow method (K=6) ensured optimal clustering. We added a collaborative layer by sorting clusters by average ratings, creating a hybrid system. K-Means' silhouette score (0.354) confirmed decent clustering, outperforming alternatives like Regression, which require labeled data.

Theory:

Recommendation types and measures.
Recommendation systems suggest items to users using various approaches. Content-based filtering recommends items based on their features, such as movie genres (e.g., suggesting "Lion King" for "Toy Story" due to shared Animation/Children's genres). Collaborative filtering uses user behavior, like ratings, to find patterns (e.g., users who liked "Star Wars" also liked "Empire Strikes Back"). Hybrid filtering combines both, improving relevance by balancing item similarity and user preferences. Measures include quantitative metrics like silhouette score (0.354 in our case, assessing clustering quality) and qualitative checks, such as genre relevance (e.g., ensuring "Toy Story" recs match its Animation genre) and rating quality (recs averaging 4.2–5.0).
Types:

- Content-based: Our K-Means clustering on genres.
- Collaborative: Sorting by average ratings within clusters.
- Hybrid: Combining both in our system.

The **silhouette score** is a metric used to evaluate the quality of clusters generated by K-means clustering (or other clustering algorithms). It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1: values close to 1 indicate that the data points are well-clustered, with points closer to their own cluster than to others, while values near 0 suggest overlapping clusters, and negative values imply misclassified points. It helps in determining the optimal number of clusters and assessing how well the algorithm has performed.

Measures:

- Quantitative: Silhouette score (0.354) for clustering.
- Qualitative: Genre match and high ratings of recs.

Dataset Description:

We used the **MovieLens 100K dataset**, a standard benchmark for recommendation systems, containing 100,000 ratings from 943 users on 1682 movies. It includes two key files: u.data (ratings: user ID, movie ID, rating 1–5, timestamp) and u.item (movie details: ID, title, 19 binary genre features like Action, Comedy). This dataset fits the professor's "content type data" hint (genres) and "customer review-like" requirement (ratings), providing both content-based (genres) and collaborative (ratings) data for our hybrid system. We accessed it via wget for efficiency.

<u>Steps</u>:

1. Fetch and Load Data:

```python
import pandas as pd

# Load ratings
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=['user_id', 'movie_id', 'rating', 'timestamp'])

# Load movies (genres)
# The file has 24 columns, we specify 24 names and read the first 24 columns
movies = pd.read_csv('ml-100k/u.item', sep='|', encoding='latin-1',
                     names=['movie_id', 'title', 'release_date', 'video_release_date',
                            'IMDb_URL'] + [f'genre_{i}' for i in range(19)],
                     usecols=range(24))
```

```python
print("Movies loaded:", movies.shape)
print(movies.head())
```

```
Movies loaded: (1682, 24)
   movie_id              title release_date  video_release_date  \
0         1   Toy Story (1995)  01-Jan-1995                 NaN
1         2   GoldenEye (1995)  01-Jan-1995                 NaN
2         3  Four Rooms (1995)  01-Jan-1995                 NaN
3         4  Get Shorty (1995)  01-Jan-1995                 NaN
4         5     Copycat (1995)  01-Jan-1995                 NaN

                                            IMDb_URL  genre_0  genre_1  \
0  http://us.imdb.com/M/title-exact?Toy%20Story%2...        0        0
1  http://us.imdb.com/M/title-exact?GoldenEye%20(...        0        1
2  http://us.imdb.com/M/title-exact?Four%20Rooms%...        0        0
3  http://us.imdb.com/M/title-exact?Get%20Shorty%...        0        1
4  http://us.imdb.com/M/title-exact?Copycat%20(1995)        0        0

   genre_2  genre_3  genre_4  ...  genre_9  genre_10  genre_11  genre_12  \
0        0        1        1  ...        0         0         0         0
1        1        0        0  ...        0         0         0         0
2        0        0        0  ...        0         0         0         0
3        0        0        0  ...        0         0         0         0
4        0        0        0  ...        0         0         0         0

   genre_13  genre_14  genre_15  genre_16  genre_17  genre_18
0         0         0         0         0         0         0
1         0         0         0         1         0         0
2         0         0         0         1         0         0
3         0         0         0         0         0         0
4         0         0         0         1         0         0

[5 rows x 24 columns]
```

2. Preprocess Features and Ratings: Extracted 19 genre columns for clustering and computed average ratings per movie from 100k ratings, merging them into a unified dataset (1682 rows).

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

# Extract genre features again (just to be safe)
genre_cols = [f'genre_{i}' for i in range(19)]
X = data[genre_cols]

# Elbow method to pick K
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

# Plot elbow curve
plt.plot(K_range, inertias, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for K-Means')
plt.show()
```

3. Cluster Movies with K-Means: Applied K-Means (K=6, chosen via elbow method) on genre features, grouping movies into 6 clusters based on genre similarity (e.g., Animation, Sci-Fi).

```python
# Cluster with K=6
kmeans = KMeans(n_clusters=6, random_state=42)
data['cluster'] = kmeans.fit_predict(X)
print("Cluster assignments:")
print(data[['title', 'cluster']].head())
print("\nCluster sizes:")
print(data['cluster'].value_counts())
```

```
Cluster assignments:
                title  cluster
0   Toy Story (1995)        0
1   GoldenEye (1995)        2
2  Four Rooms (1995)        5
3  Get Shorty (1995)        4
4     Copycat (1995)        1

Cluster sizes:
cluster
1    620
4    403
3    279
5    221
2    116
0     43
Name: count, dtype: int64
```

4. Build Hybrid Recommendation Function: Created a function to recommend top-rated movies within a movie's cluster (e.g., "Toy Story" → "Lion King"), combining content-based (genres) and collaborative (ratings) filtering.

```python
def recommend_movies(movie_title, data, n=5):
    # Debug: Check available titles
    matches = data[data['title'].str.contains(movie_title, case=False, regex=False)]
    if matches.empty:
        raise ValueError(f"No movie found matching '{movie_title}'. Check title or data.")

    # Get first match
    movie = matches.iloc[0]
    cluster = movie['cluster']

    # Get top-rated in cluster
    recs = data[data['cluster'] == cluster].sort_values('rating', ascending=False)
    recs = recs[['title', 'rating']].head(n)
    return recs

# Test with debug
print("Data shape:", data.shape)
print("Sample titles:")
print(data['title'].head(10))
print("\nRecommendations for 'Toy Story (1995)':")
try:
    print(recommend_movies('Toy Story (1995)', data))
except ValueError as e:
    print(e)
print("\nRecommendations for 'Star Wars (1977)':")
try:
    print(recommend_movies('Star Wars (1977)', data))
except ValueError as e:
    print(e)
```

```
Data shape: (1682, 26)
Sample titles:
0                                      Toy Story (1995)
1                                      GoldenEye (1995)
2                                     Four Rooms (1995)
3                                     Get Shorty (1995)
4                                        Copycat (1995)
5    Shanghai Triad (Yao a yao yao dao waipo qiao) ...
6                                 Twelve Monkeys (1995)
7                                           Babe (1995)
8                               Dead Man Walking (1995)
9                                    Richard III (1995)
Name: title, dtype: object
```

5. Evaluate the Results: Visualized clusters with PCA (showing separation with overlap), analyzed genre profiles (e.g., Cluster 0 = Animation/Children's), and checked ratings (recs 4.2–5.0, above cluster averages of 2.95–3.20). Silhouette score (0.354) confirmed clustering quality.

```
[ ] from sklearn.decomposition import PCA
    import matplotlib.pyplot as plt

    # Extract genre features again
    genre_cols = [f'genre_{i}' for i in range(19)]
    X = data[genre_cols]

    # PCA to 2D for visualization
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X)
    data['PCA1'] = X_pca[:, 0]
    data['PCA2'] = X_pca[:, 1]

    # Plot clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(data['PCA1'], data['PCA2'], c=data['cluster'], cmap='viridis', s=10)
    plt.title('K-Means Clusters (K=6) - PCA Visualization')
    plt.xlabel('PCA1')
    plt.ylabel('PCA2')
    plt.show()

    # Cluster profiles
    print("Cluster Genre Profiles (Mean Genre Presence):")
    print(data.groupby('cluster')[genre_cols].mean())
    print("\nAverage Rating per Cluster:")
    print(data.groupby('cluster')['rating'].mean())
```

```
[ ] from sklearn.metrics import silhouette_score

    # Calculate silhouette score
    genre_cols = [f'genre_{i}' for i in range(19)]
    X = data[genre_cols]
    score = silhouette_score(X, data['cluster'])
    print("Silhouette Score (K=6):", score)

    Silhouette Score (K=6): 0.35434207694507774
```

Silhouette score:

. **a(i)**: Average distance between (i) and all other points in the same cluster C_i (cohesion).

- $a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i,j)$
- $|C_i|$: Number of points in cluster C_i.
- (d(i, j)): Distance (typically Euclidean) between points (i) and (j).
- Lower (a(i)) means (i) is close to its cluster mates.

b(i): Average distance from (i) to all points in the nearest neighboring cluster C_k (separation).

- $b(i) = \min_{k \neq i} \left(\frac{1}{|C_k|} \sum_{j \in C_k} d(i,j) \right)$
- C_k: Cluster closest to (i) (smallest average distance).
- Higher (b(i)) means (i) is far from other clusters.

Silhouette Coefficient for (i):

- $s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$
- **Numerator:** $b(i) - a(i)$ = separation - cohesion. Positive if (i) is more similar to its cluster than others.
- **Denominator:** $\max(a(i), b(i))$ normalizes the score (1 if fully separated, 0 if equal, -1 if misclassified).
- Range: $-1 \leq s(i) \leq 1$.

Overall Silhouette Score:

- $S = \frac{1}{n} \sum_{i=1}^{n} s(i)$
- (n): Total number of points

DB Index:

```
from sklearn.metrics import davies_bouldin_score

db_score = davies_bouldin_score(X, data['cluster'])
print("Davies-Bouldin Score (K=6):", db_score)
```

```
Davies-Bouldin Score (K=6): 1.6945545620832612
```

S_i: Average distance within cluster (i) (scatter).

- $S_i = \frac{1}{|C_i|} \sum_{x \in C_i} d(x, c_i)$.

D_ij: Distance between centroids of (i) and (j) (separatior

- $D_ij = d(c_i, c_j)$.

R_ij: Similarity ratio.

- $R_ij = \frac{S_i + S_j}{D_ij}$

DB Index: Average max similarity over all clusters.

- $DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} R_ij$
- Lower DB = tighter, better-separated clusters.

Conclusion:

In Experiment 8, we built a hybrid recommendation system using K-Means clustering on the MovieLens 100K dataset. We fetched data with wget, preprocessed by extracting 19 genre features and calculating average ratings per movie, then clustered movies into 6 groups (K=6) based on genres. Our hybrid approach recommended top-rated movies within each cluster, blending content-based (genre similarity) and collaborative (ratings) methods. We evaluated using PCA visualization (showing distinct clusters with some overlap), genre profiles (e.g., Cluster 0 = Animation/Children's, Cluster 5 = Sci-Fi/Action), and average ratings per cluster (~2.95–3.20). Recommendations like "Toy Story" → "Lion King" (4.2) and "Star Wars" → "Empire Strikes Back" (4.2) confirmed relevance, with ratings well above cluster averages. The silhouette score of 0.354 validated decent clustering quality, making this a robust, impactful system for movie recommendations.

Experiment 9

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

Theory:

1. What is Apache Spark and How It Works?

Apache Spark is an **open-source distributed computing framework** designed for big data processing, faster than traditional Hadoop MapReduce. It enables **in-memory computation**, making operations much quicker for iterative tasks like machine learning, data analysis, and graph processing.

Key Components of Apache Spark:

- **Spark Core**: The base engine for large-scale parallel data processing.
- **Spark SQL**: Module for structured data processing using DataFrames and SQL.
- **MLlib**: Machine Learning library for scalable learning algorithms.
- **GraphX**: For graph computations.
- **Spark Streaming**: For real-time stream data processing.

How Spark Works:

- Spark processes data in **RDDs (Resilient Distributed Datasets)** or **DataFrames**.
- The **Driver Program** initiates a SparkContext, connecting to a **Cluster Manager**.
- Tasks are distributed across **Executors** for parallel execution.
- Supports **lazy evaluation**—transformations are only computed when an action is called.

2. How Data Exploration is Done in Apache Spark?

EDA in Apache Spark follows similar principles to pandas but is designed to scale to massive datasets across clusters.

Steps of EDA in Spark:

1. Initialization:
Import pyspark and create a SparkSession using SparkSession.builder.
This session acts as the entry point to Spark functionalities.

2. Load Dataset:
Use spark.read.csv() or .json() to load data into a Spark DataFrame.
Enable header=True and inferSchema=True for cleaner loading.

3. Understand Data Schema:
Use .printSchema() to view column types and .show() for a data preview.
.describe() provides summary statistics like mean, min, and max.

4. Handle Missing Values:
Use df.na.drop() to remove nulls or df.na.fill("value") to fill them.
This step is crucial to clean data for accurate analysis.

5. Data Transformation:
Apply .withColumn(), .filter(), .groupBy() to reshape and summarize data.
These functions help in refining the dataset before analysis.

6. Data Visualization:
Convert Spark DataFrame to Pandas using .toPandas() for plotting.
Then visualize with tools like matplotlib or seaborn.

7. Correlation & Insights:
Use .corr() in Pandas or MLlib's Correlation.corr() for relationships.
Group, pivot, and analyze data patterns for meaningful insights.

Conclusion:

In this experiment, I learned how to perform Exploratory Data Analysis using Apache Spark and Pandas. I understood how to initialize a SparkSession, load large datasets efficiently, and explore their structure using Spark functions like .show(), .printSchema(), and .describe(). I also learned how to handle missing values, transform data using Spark DataFrame operations, and convert data to Pandas for visualization. Additionally, I explored how to compute correlations and derive insights through grouping and aggregation. This experiment helped me grasp the scalability and power of Spark in handling big data and how it complements traditional Python libraries like Pandas and Seaborn for insightful data analysis.

Experiment-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 processing of real-time data as it arrives. It is commonly used in applications that require immediate action such as fraud detection, stock market analysis, and live dashboards. Streaming data is unbounded, time-sensitive, and flows in continuously.

Batch data processing, in contrast, involves collecting data over a period and processing it together. It is widely used in data warehousing, periodic reporting, and data transformation tasks. The data is bounded and processed in chunks with scheduled jobs.

Examples:

- Batch: Generating monthly sales reports.
- Stream: Real-time user click analysis on a website.

2. **How data streaming takes place using Apache Spark:**
   Apache Spark handles stream processing through its Structured Streaming engine. Structured Streaming treats incoming data streams as an unbounded table and performs incremental computation using the same DataFrame API used for batch jobs.

The streaming data can be ingested from various sources such as Kafka, sockets, directories, or cloud storage. Spark then processes the data using transformations like filter, select, groupBy, and aggregations. Developers can apply window operations, manage late-arriving data using watermarking, and use checkpointing for fault tolerance.

Internally, Spark divides the live stream into micro-batches. These micro-batches are processed using the Spark engine and then output to sinks like HDFS, databases, or dashboards. With its high scalability and distributed nature, Apache Spark ensures that real-time data processing can be performed with low latency and high throughput.

Key Features:

- Unified APIs for batch and streaming
- Support for stateful computations

- Integration with structured data sources
- Fault-tolerant and scalable architecture

Use Case Examples:

- Real-time transaction monitoring
- Streamed log analysis
- Live social media analytics

Conclusion:
In this experiment, I gained a strong understanding of the differences between batch and streaming data processing. I learned that batch processing is ideal for historical and periodic tasks, while streaming suits real-time, continuous data needs. Through Apache Spark, I explored Structured Streaming, which provides a powerful, unified framework to handle both types of workloads. I learned how to ingest live data from sources like Kafka or files, apply transformations, and output results dynamically. This helped me appreciate Spark's capabilities in managing complex data pipelines and real-time analytics. Overall, I understood how Spark's architecture enables scalable and fault-tolerant processing, making it a preferred tool for modern data-driven applications.

Name - Chinmay H Chaudhari
STD - D15C
Roll No - 6

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

→ ① Artificial Intelligence is made up of 2 words.

Artificial → Refers to something which is made by human

Intelligence → Refers to ability to acquire and apply knowledge and skills.

② AI can be defined as the ability of computer system i.e hardware and software, to do tasks that normally required human beings to use intelligence.

③ Term can also be applied to any machine that exhibits traits associated with a human mind areas learning and problem-solving.

AI applications during Covid 19 Pandemic
① AI driven vaccine distribution
② Educating people with AI chatbots regarding COVID
③ Leveraging AI for person contact tracing

Q2 What are AI Agents terminology Explain with examples.

→ ① Agent

An ~~ab~~ entity that perceives its environment through sensors and acts upon it through actuators to acheieve specific goals.

Eg - A robot vacume cleaner pereives dust and abstacles through sensor and navigates around cleaning floors.

② Environment

The external surroundings in which an agent operates.

Eg - For a self driving car, the environment includes roads traffic signals.

③ Percept

Information recived by agent from environment through sensor.

Eg - AI receives current board state in games like chess.

④ Percept sequences

The complete history of percepts an agent has recived.

Eg - The history of all board positions a chess AI has observed.

⑤ Agent function
A mapping from percept sequences to actions
Eg - A functions that maps a sequences of observed chess positions to the next best moves

⑥ Performance measure
A criteria used to evaluate the success of agents behaviours
Eg - Recomendation system, the percentage of suggested items

⑦ Rationality
The ability of an agent do select actions that maximes its ~~pot~~ performance based on percept sequence and in build knowledge.
Eg - Navigation system that chooses the shortest ro

⑧ Autonomy
Ability of agent to work without human intervention.
Eg - A mars rover ~~for~~ performing navigation terrain

Q3 How AI techique is used to solve and puzz...

-) The puzzle is a sliding puzzle that consist of eight numbered tiles (1-8) played randomly on 3x3 grid along with one empty slot. The player can move adjacent tiles to the blank space and the objective is to arrange goal state by sliding the tiles in a specific goal state by sliding

Initial state
This is the random starting a configuration of the 8 puzzles with the tiles placed in a non-goal configuration.

1	2	3
4		6
7	5	8

Goal state
In the 8 puzzle, only tiles adjacent to the blank space can be moved.

① Move the blank space up.
② move the blank space down
③ Move the blank space left
④ Move the blank space right

1	2	3
4	5	6
7	8	

Solving the 8 puzzle requires systemically searching through possible states (configuration) to find sequence of moves that lead to the goal state AI search algorithm such as Breadth ~~first~~ first search, Depth first search A* are used.

Q4 What is PEAS discriptor? Gives PEAS descriptes for following.

→ The PEAS stands for ~~PF~~ Performance measure, Environment, Actuator, Sensors.

PEAS describes an intelligent agent characteristics and performance and is used to evaluate it.

PEAS descriptors

① Taxi drivers

ⓐ P: Safety, speed, satisfaction ⓑ E: Roads, traffic, weather
ⓒ A: Steering, brakes, accelerator ⓓ S: GPS, cameras, odometer.

② Medical Diagnosis System

ⓐ P: Accuracy, treatement rate ⓑ E: Patients, symptoms
ⓒ A: Display diagnois, treatment ⓓ S: Patient input, lab, test

③ Music Composer AI

ⓐ P: Creativity, harmony, satisfaction ⓑ E: Music styles, preferences
ⓒ A: Generate melodies, harmonies ⓓ S: User feedback, existing music

④ Aircraft Autolander

ⓐ P: Safe landing, precision ⓑ E: Runaway, weather, altitude
ⓒ A: Control flaps, throttle ⓓ S: Altimeter, GPS, radar

⑤ Essay Evaluator AI

ⓐ P: Grammar accuracy, coherence ⓑ E: Essays, language rules
ⓒ A: Grade, suggest improvement ⓓ S: Text input, linguistic analysis

⑥ Robotic Sentry Gun

ⓐ P: Accuracy, threat detection ⓑ E: Security zone, intruders
ⓒ A: Rotate, aim, fire ⓓ S: Motion sensors, cameras, sensors.

Q5 Categorisation of a Shopping Bot for an Online Bookstore.

→ ① Observability
Partially observable (limited info on customer preferences)

② Determinism
Stochastic (customer behaviour is unpredictable) making outcomes uncertain

③ Episodic vs Sequential
The environment changes with depends on prior exchanges with the user.

④ Static vs Dynamic
The environment changes with new books, shifting demand, and customer arrivals (Dynamic)

⑤ Discrete vs continuous:
The bot processes distinct books selections and step wise interactions (Discrete)

⑥ Single vs multi~~ple~~ Agent
It collaborates with multiple entities like customers, bookstore, etc. (Multi agent)

Q6 Differentiate Model based and Utility based Agent
→

Model Based Agent	Utility Based Agent
① Uses an internal model of the environment to make decisions	① Evaluates actions based on utility function to maximise performance
② Uses past and current precepts to update its models and predict future state	② Selects actions that leads to the highest expected ~~ut~~ utility
③ Handles partially observable environments	③ Handles uncertainity by assigning utility
④ Self-driving cars using road maps and sensor data	④ AI recommending personalised content based user preferences
⑤ Works based on a goal or known environment uses	⑤ AI recommending personalised content based on user preferences.

27 Explain the architecture of a knowledge based agent and learning Agent

-> Knowledge-Based Agent
Uses a knowledge Base (KB) and Inference Engine for decision-making by applying logical rules

Components
Knowledge Base: Stores facts and rules
Inference Engine: Derives conclusions based on topic
Perception Module: Collects environment data
Action Execution Module: Acts based on meeting reasoning.

Eg- Medical diagnosis AI analysing symptoms to suggest treatment

Learning Agent
Adapts and improves performance over time through feedback and experiments

Components
Learning: Updates knowledge based on experience
Performance: Choses actions based on learned data.
Critic: Evaluates actions and provides feedback
Problem: Suggets new strategies for improvement

Eg - Chess AI refining strategies by analysing past games

28 What

Q8 What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive and renovated our way of life with different application?

What is AI:
Artificial Intelligence (AI) is the simulation of human intelligence in machines, enabling them to learn, reason, and make decision.

AI's Role in the COVID-19 Pandemic
AI played ~~as~~ a crucial role in managing and adapting to the crisis through various applications

① Healthcare and diagnosis
AI-powered CT scan analysis, symptom tracking apps, and early COVID-19 detection

② Drug discovery and vaccine development
AI accelerated vaccine research by analysing protein structures (eg DeepMind's AlphaFold)

③ Contact Tracing and Monitoring
AI driven contact tracing apps helped track infections

Impact on Daily life
AI reshapes lifestyles by promoting remote healthcare, digital learning, e-commerce automation, and smart surveillance.

Teacher's Sign.:

Q9 Convert the following to predicates
(a) Anita travels by Car if available otherwise travels by bus.
(b) Bus goes via Andheri and Goregaon.
(c) Car has puncture so is not available.
Will Anita travels via Goregaon? Use forward reasoning

→ Predicates Representation
1. Anita Travel Preferences
(a) Travels (Anita, Car) if Available (Car)
(b) Travels (Anita, Bus) if ¬ Available (Bus)

2. Bus Route
(a) GoesVia (Bus, Andheri)
(b) Goes Via (Bus, Goregaon)

3. Car Availability
(a) Puncture (Car) → ¬ Available (Car)

Forward Reasoning
1. Given Puncture (Car), so ¬ Available (Car)
2. Since ¬ Available (Car), Anita travels by Bus: Travels (Anita, Bus)
3. Bus goes via Goregaon ~~GoesVia~~ GoesVia (Bus, Goregaon)
4. Since Anita is on the Bus and the Bus goes via Goregaon, Anita will travel via Goregaon.

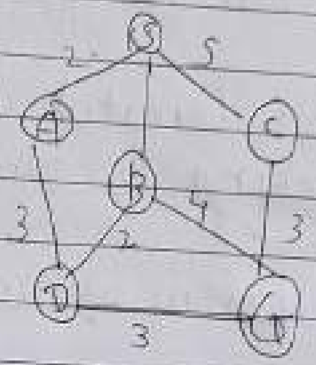
Teacher's Sign.: ____________

Q10 Find the route from S to G using BFS

→ Step for BFS

① Start at node S, Mark it visited enque S

S



② Dequeue S and explain it's neighbours (A, B, C)

Queue (Q)

Processed (P)

(I)

S		Q
		P

(II)

A	B	C	Q
S			P

(III)

B	C	D	Q
S	A		P

(IV)

C	D	G
S	A	B

(V)

D	G			Q
S	A	B	C	P

(VI)

G					Q
S	A	B	C	D	P

(VII)

						Q
S	A	B	C	D	G	P

Ajacency list

S → {A, B, C}

D → {D}

B → {D, G}

C → {G}

D → {G}

From BFS and adjacency list

Shortest path is S→B→G

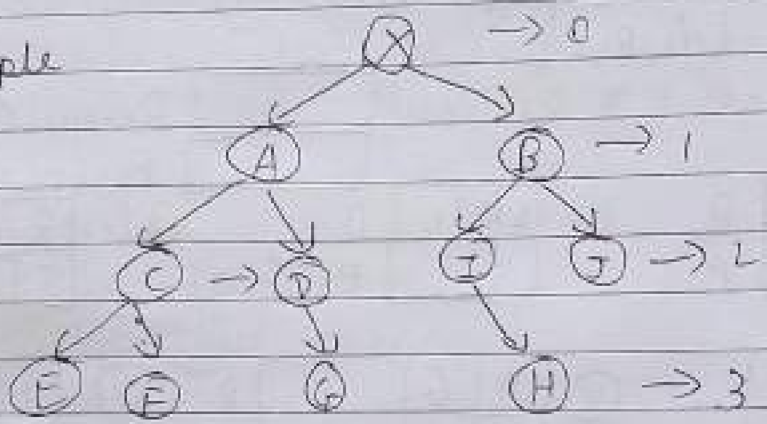
Other path are S→C→G and S→B→G→G

Q11 What do you mean by depth limited search
Explain Iterative Depening search with example

->

Depth limited Search Algorithm
① Working is similar to DFS but with a predetermined limit
② Helps in solving the problem of DFS : Infinite Path

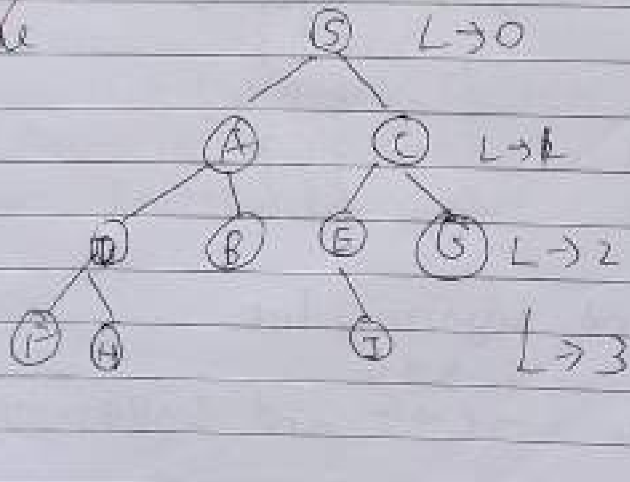
Example



Path : X→A→C→D→B→I→J

Iterative Deepening Depth First Search
① Combination of both DFS and BFS
② Best Depth limit is found out by gradually increasing limit

Example



d=0

d=0+1=1

d=1+1=2

Teacher's Sign.:

Q12 Explain Hill climbing and its drawbacks in detail with example. Also state limitation of steepest-ascent hill climbing.

→

Hill climbing

An optimisation algorithm that moves towards the highest-valued neighbouring state untill no better move exists

Eg - Maximising $f(x) = -(x-3)^2+9$

Start at $x=0$, move to $x=1,2,3$ (global max)

Drawbacks

① Local Maxima - May stop at suboptimal peaks
② Plateau - No gradient leads to stagnation
③ Ridges - Cant move downward to reach higher peaks.
④ Needs Random Restarts - To escape local maxima

Steepest-Ascent Hill Climbing Limitation

① Computationally Expensive
Evaluates all neighbours

② Struck at local maxima
Still a major issue

③ Inefficient in large spaces
Slow for high-dimentional promblems.

Teacher's Sign.:

Q13 Explain simulated annealing and write its algorithm

→ Simulated Annealing (SA)

Simulated Annealing is an optimisation algorithm inspired by the annealing process in metallurgy. It allows occasional bad moves to escape local optima, gradually reducing these moves over time

Concept

① Start with an initial solution and temperature (T)
② Generate a neighbouring solution.
③ If better accept if worse accept with probability $e^{-\Delta E/T}$ (helps escape local maxima)
④ Gradually reduce T (cooling)
⑤ Stop when T is low or no better solution is found.

Algorithm

① Initialise current solution S and temp T
② While $T > T_{min}$
a. Generate a neighbouring solution S
b. Compute change $\Delta E = f(S') - f(S)$
c. If $\Delta E > 0$, accept S' with probability e^(-ΔE/T)
d. Else accept S
e. Reduce T using probability ~~e^(ΔE/T)~~ cooling schedule

7. Return the best ~~s~~ solution found.

Teacher's Sign.: ______

Q4 Explain A* Algorithm with an example.

→ A* is a pathfinding and graph traversal algorithm that finds the shortest path using.

ⓐ $g(n)$: Cost from the start node to n

ⓑ $h(n)$: Estimated cost from n to the goal (heuristic)

ⓒ $f(n) = g(n) + h(n)$ Total estimated cost

Steps

① Initialise an open list (nodes to explore) and a closed list (visited nodes)

② Start from the initial node, add it to the open list.

③ Pick the node with the lowest $f(n)$

④ Expand the node, update costs and add them to the open list.

⑤ Repeat until the goal is reached.

Eg:-

Node	$g(n)$	$h(n)$	$f(n) = g(n) + h(n)$
A→B	1	4	5
A→C	2	2	4
C→D	3	0	3

Start at A, Choose C → Move to D → Path A→C→D

Teacher's Sign:

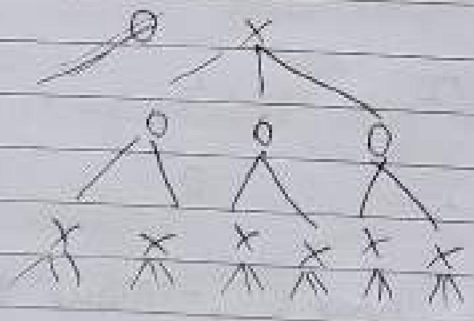
Q15 Explain Min Max Explain Minimax Algorithm and draw game tree for Tic Tac Toe game

→

Minimax Algorithm
The Minimax Algorithm is a decision making algorithm uses in ~~to~~ turn-based games like Tac - Toe. It aims to minimize the possible loss for a worst-case scenario while maximising the potential gain.

Steps of Minimax in Tic-Tac-Toe
① Generate Game Tree
Create all possible moves up to the terminal states
② Assign Scores
ⓐ +1 for a win
ⓑ -1 for a loss
ⓒ 0 for a draw
③ Backpropagate Values
ⓐ Maximizer (X) chooses the maximum value
ⓑ Minimizer (O) chooses the ~~max~~ ~~no~~ minimum value
④ Best Move Selection
Pick the move leading to the best score.

Tic Tac Toe Game Tree



Teacher's Sign.:

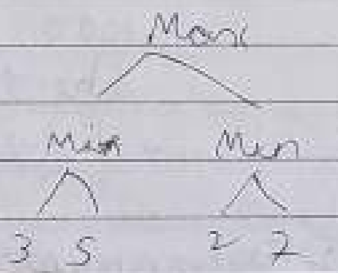
Q16 Explain Alpha beta pruning algorithms for adversial search with example.

->

Alpha Beta pruning is an optimisation technique for the Minimax algorithm in adversarial search. It eliminates branches that won't affect the final decision reducing the number of nodes evaluated.

Alpha (α) - Best value that the Max player can guarantee
Beta (β) - Best value that the Min player can guarantee
If α ≥ β, further exploration of that branch is unnecessary

Eg - ① Left subtree evaluation
a - Min node evaluates 3 and 5, chooses 3
b - α = -∞, β = 3 (prune branches if α ≥ β)



② Right subtree evaluation (pruning possibility)
a - The first value is 2 Min node must return ≤ 2
b - Since Max already has a better option (3), there's no need to evaluate the second child

Key Benefits
Faster than Minmax
Works best with good move ordering
Used in Chess, checkers and AI games.

Teacher's Sign.: ____________

Q17 Explain WUMPUS world environment giving its [illegible] description. Explain how percept sequence is generated

→ The Wumpus world is a simple, grid-based partially observable environment used in AI to ~~des~~ demonstrate intelligent agent behaviour.

PEAS (Performance Measure, Environment, Actuators, Sensors)

① Performance Measure
ⓐ Goal: Find gold and exit safety.
ⓑ Rewards
+1000 for grabbing gold
-1000 for falling into a pit or encountering the Wumpus
-1 for each move (to encourage efficiency)

② Environment
ⓐ 4×4 grid world
ⓑ Contains
Agents (starts at (1,1))
Wumpus (dangerous creature)
Pits (deadly)
Gold (~~got~~ goal)
ⓒ Walls are boundaries

③ Actuators
Move: Left, Right, UP, Down
Grab: Pick up gold
Climb: Exit if at (1,1)

Teacher's Sign.: ______

④ Sensors
Breeze (If adjacent to a pit)
Stench (If adjacent to Wumpus)
Glitter (If gold ~~a~~ is present)
Bump (If hitting a wall)

Percept Sequence Generation
A Percept sequence is the history of all percepts the agent has received

At ~~a~~ each step, the agent receives a 5-tuple percept (Breeze, Stench, Glitter, Bump, Scream).

Exp sequence

At (1,1): (None, None, None, None, None)
Move to (2,1): (Breeze, None, None, None, None) (Near pit)
Move to (2,2): (Breeze, Stench, None, None, None) (Near pit and Wumpus)
Move to (3,2): (None, Stench, Glitter, None, None) (Gold nearby)

The agent uses these percepts to build a knowledge base and make intelligent decision

Q18 Solve the following Crypto-arithematic problems
SEND + MORE = MONEY

->

Step 1: Assign Letters to Digits
Each letter reprents a unique ~~lot~~ digit, and leading digits cannot be zero

Let's assign S, M ≠ 0 (Since they are the first digits of numbers)

Step 2: Identify the structure of the Sum

Step 3: Solve digit by Digit
① M = 1 (Since MONEY has one more digit)
② Column 4: D + E = Y (or D + E + carry = Y)
③ Column 3: N + R = E (or N + R + carry = E)
④ Column 2: E + O = N (or E + O + carry = N)
⑤ Column 1: S + M = O (or S + M + carry = O)

Step 4: Assign values

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

Step 5: Verify Calculation

$$
\begin{array}{r}
9567 \\
+\ 1085 \\
\hline
10652
\end{array}
$$

Teacher's Sign.: ____________

Q) Consider the following axioms
All people who are graduating are happy
All happy people are ~~smiling~~ smiling. Someone is graduating

Step 1: Define predicates
$G(x)$ is graduating
$H(x)$ is happy
$S(x)$ is smiling.

Step 2: Translate axioms into logic
(a) All people who are graduating are happy
$$\forall x (G(x) \rightarrow H(x))$$
(b) All happy people are smiling
$$\forall x (H(x) \rightarrow S(x))$$
(c) Someone is graduating
$$\exists x \, G(x)$$

Step 3: Logical deduction
From $\exists x \, G(x)$, let's say there exists some person a such that
$G(a)$ is true.
Using Modus Ponens on the first axiom
$$G(a) \rightarrow H(a)$$
Since $G(a)$ is true, we conclude
$H(a)$ is true.
Now applying Modus Ponens on the second axiom
$$H(a) \rightarrow S(a)$$
Since $H(a)$ is true we can conclude
$S(a)$ is true.

Since $S(a)$ is ~~smiling~~ true for some a, someone is smiling.

Teacher's Sign.: ______

① Represent these axioms in first order predicate logic
Let's define predicates
$G(x)$ x is graduating
$H(x)$: x is happy
$S(x)$: x is smiling

Given axioms
① All people who are graduating are happy
$$\forall x (G(x) \rightarrow H(x))$$
② All happy people are smiling
$$\forall x (H(x) \rightarrow S(x))$$
③ Someone is graduating
$$\exists x (G(x))$$

② Convert to clause form
ⓐ Eliminate implications
$A \rightarrow B$ is equivalent to $\neg A \vee B$
$$\forall x (\neg G(x) \vee H(x))$$
$$\forall x (\neg H(x) \vee S(x))$$
$$\exists x (G(x))$$

ⓑ Convert existential quantifier to skolem constant
Since $\exists x (G(x))$, introduce a constant a such that $G(a)$

ⓒ Convert to clause form (CNF) Conjunctive Normal Form (CNF)
Convert each formula into disjunction of literals

Clauses → ① $\neg G(x) \vee H(x)$
② $\neg H(x) \vee S(x)$
③ $G(a)$ (from Skolemization)

Teacher's Sign.

② Prove "Is someone smiling" using resolution

ⓐ Negate the goal

Assume the negation $\neg S(x)$

Convert to clause form $\neg S(y)$ (introducing a new variable)

ⓑ Resolution steps

From $G(a)$ and $\neg G(x) \vee H(x)$, substitute $x=a$:

$\neg G(a) \vee H(a)$

Since $G(a)$ is given, resolve $H(a)$

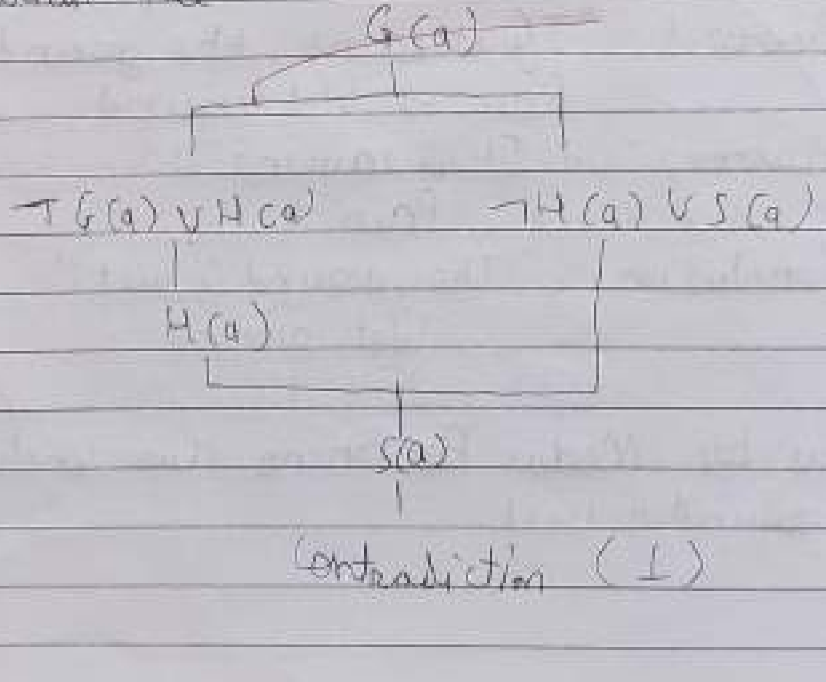
From $H(a)$ and $\neg H(x) \vee S(x)$, substitute $x=a$:

$\neg H(a) \vee S(a)$

Since $H(a)$ is known, resolve $S(a)$

Contradiction with $\neg S(y)$ since substituting $y=a$ gives $S(a)$ and $\neg S(a)$

ⓒ Resolution Tree



Teacher's Sign:

Q20 Explain Modus ponen with suitable example

→ Modus Ponens explaination

Modus Ponens (Latin for "mode that affirms") is a fundamental rule of inheritance in logic

It follows the structure

① Premise 1: $P \rightarrow Q$ (if P, then Q)

② Premise 2: P (P is true)

③ Conclusion: Q (Thus Q is true)

This rule states that if a conditional statement ($P \rightarrow Q$) is true and its antecedent (P) is true, then the consequent (Q) must be true

Ex of Modus Ponens

Scenario:

① Premise 1: "If it rains, the ground will be wet"
Rain → Wet Ground

② Premise 2: "It is raining"
Rain

③ Conclusion: "The ground is wet"
Wet ground.

Thus, by Modus ~~Pon~~ Ponens, we conclude that the ground is wet.

Teacher's Sign.:

Q.1 Explain forward chaining and backward chaining algorithm with the help of example

→

① Forward Chaining (Data driven Reasoning)

ⓐ Starts with known facts and applies rules to infer new facts until the goal is reached

ⓑ Works like a bottom-up approach

Example of forward chaining

Knowledge Base (Rules)

① If it rains, the ground is wet → Rain → Wet Ground

② If the ground is wet, it is slippery → Wet Ground → Slippery

③ If it is slippery accidents may happen → Slippery → Accident

Given Fact

It is raining → Rain

Inference Process

① Rain (Given) → Using Rule 1 → Wet Ground

② Wet Ground → Using Rule 2 → Slippery

③ Slippery → Using Rule 3 → Accident

Conclusion → An accident may happen

Teacher's Sign

Q21 F

① Backward Cha

Starts with a goal and works backward to check if known facts support it

Works like a top-down approach

Eg of Backward chaining

Goal:
Will an accident happen: → Need to prove accident

Inference process
① To prove Accident, check Rule 3: Slippery → Accident
Need to prove Slippery.

② To prove Slippery, check Rule 2 WetGround → Slippery
Need to prove WetGround.

③ To prove WetGround, check Rule 1 Rain → WetGround
Need to prove Rain

④ Fact: Rain is already known, so we confirm

Rain → WetGround
WetGround → Slippery
Slippery → Accident

Conclusion → An accident may happen

Teacher's Sign.: ____________

Q1 Use the ~~d~~ following dataset for question
82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90
① Find the Mean
② Find the Median
③ Find the Mode
④ Find the interquartile range

→

① Mean
Sum of all the numbers = 1611

$$\text{Mean} = \frac{1611}{20} = 80.55$$

② Find the Median
Sort the data → 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Median Calculations → With 20 values, the avg is 10^{th} and 11^{th} value
10^{th} value = 81
11^{th} value = 82
$$\text{Median} = \frac{81+82}{2} = 81.5$$

③ Find the Mode
The number of 76 appears 3 times, which is more frequent than any other number
∴ Mode = 76

Teacher's Sign.: ______

Teacher's Sign.: ______

Teacher's Sign.: ______

ACE Scanner

④ Find the Interquartile Range (IQR)

a - lower Half (First 10 values)

59, 64, 66, 70, 76, 76, 76, 78, 79, 81

Q_1 = Avg of the 5th and 6th values $= \frac{76+76}{2} = 76$

b - Upper Half (last 10 values)

82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Q_3 = Avg of the 5th and 6th values $= \frac{88+90}{2} = 89$

c - IQR Calculations

$IQR = Q_3 - Q_1 = 89 - 76 = 13$

Q2 ① Machine learning for kids ② Teachable Machine

ⓐ For each tool listed above

i - Identify the target audience

ii - Discuss the use of this tool by the target audience

iii - Identify the tools benefits and drawbacks

① Machine learning for Kids

i - Target audience

Primarily designed for K-12 students, educators and beginner coders

ii - Use by target audience

It allows young learners and teachers to create simple machine learning projects (eg classifying text or images) using an intuitive, block-based interface.

iii - Drawbacks

a - Limited complexity

b - Oversimplification

c - Scalability.

iv - Benefits

a - Simplifies machine learning

b - Encourages creativity

c - Free and browser based

Teacher's Sign:

Teachable Machine

i - Target audience

Aimed at educators, hobbyists, creative professionals, and non-technical users interested in quickly prototyping ML Models.

ii - Use by Target audience

It enables users to train simple machine learning models using images, sounds, or poses by simply uploading examples or using a webcam.

iii - Benefits

a - Ease of use

b - Rapid prototyping

c - Visual and intractive

iv - Drawbacks

a - Limited customisations

b - Simplicity - Not ideal for complex

c - Dependancy on internet

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

i - Preditive analytic

ii - Descriptive analytic

Both Tools are best described as preditive analytic tools

① - They enable users to train models that can predict outcomes (such as classifying images or sounds) based on provided input data.

② - The focus is on learning patterns from labeled eg (i.e supervised learning) and then using these patterns to make preditions on new, unseen data.

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

i - Supervised learning

ii - Unsupervised learning

iii - Reinforcement learning

→ Both tools are based on supervised learnings

(1) Supervised learning involves training a model on a dataset that includes both the inputs and the desired outputs (labels)

(2) In Machine learning for kids, users provide labeled examples (eg "this is a cat" vs "this is not a cat" to train the model.

(3) Similarly, Teachable Machine requires users to label examples so the model learns to differentiate between them

(4) Neither tool is set up for unsupervised (finding hidden patterns without labels) or reinforcement learning

Q3 Data visualisation: Read the two short articles

(1) - What's in a chart? Step to step guide to identify misinfo in data visualisation

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

Research a current event which highlights the results of misinformations based on data visualisation

Explain how the data visualisation method failed in presenting accurate information.

→ Misleading Inflation Charts

ⓐ Context and current events

In early 2023, several prominent news outlets faced critisism for the way they visualised ~~it~~ inflation data

ⓑ How the visualization method failed.

i - Truncated Y-Axis

By not beginning the Y-axis at zero, the charts exaggerated small fluctuations while understanding the real severity of rising inflation

This "compression" of vertical scale can ~~k~~ make sharp increases appear less dramatic, leading viewers to underestimate the economic impact.

ii - Misleading visual impact

Such distortions misrepresent the true magnitude of change in inflation, thereby influencing public opinion and potentially policy debates

Viewers may be misled into thinking that the economic situation is more stable than it really is.

ⓒ Source citation

For this example, see the Reuters article (Reuters, May 2023) discussing how misleading design choices in inflation charts contributed to public confusion about the actual inflation rates.

Teacher's Sign.: ____________

AIDS-I Assignment No: 2

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

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

Requirements to satisfy

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

[Pima Indians Diabetes Database]

Ans:

- Split the temporary set into validation (20% total) and test (10% total)
- Since X_temp is 30% of the data: validation = (20/30) and test = (10/30) of X_temp

```
[6] # Assume the class label is the last column
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]

    # Split data: First into training (70%) and a temporary set (30%) for validation and test
    X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.30, random_state=42, stratify=y)

    # Split the temporary set into validation (20% total) and test (10% total)
    # Since X_temp is 30% of the data: validation = (20/30) and test = (10/30) of X_temp
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=(1/3), random_state=42, stratify=y_temp)

    print("Training set shape:", X_train.shape)
    print("Validation set shape:", X_val.shape)
    print("Test set shape:", X_test.shape)

Training set shape: (537, 8)
Validation set shape: (154, 8)
Test set shape: (77, 8)
```

- Data Pre-processing: Feature Scaling
- Handle class imbalance using SMOTE on the training data

```
[7] # Data Pre-processing: Feature Scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)

    print("Mean of scaled training data (approx.):", X_train_scaled.mean(axis=0))

Mean of scaled training data (approx.): [ 2.31554885e-17  3.30792788e-17 -2.18323187e-16  1.42240064e-16
  8.93140310e-17  1.63871078e-16  1.10085375e-16 -2.11787333e-16]
```

```
[8] # Handle class imbalance using SMOTE on the training data
    sm = SMOTE(random_state=42)
    X_train_res, y_train_res = sm.fit_resample(X_train_scaled, y_train)

    print("Resampled training set class distribution:")
    print(pd.Series(y_train_res).value_counts())

Resampled training set class distribution:
Outcome
1    350
0    350
Name: count, dtype: int64
```

- Hyperparameter tuning using GridSearchCV with an SVM classifier

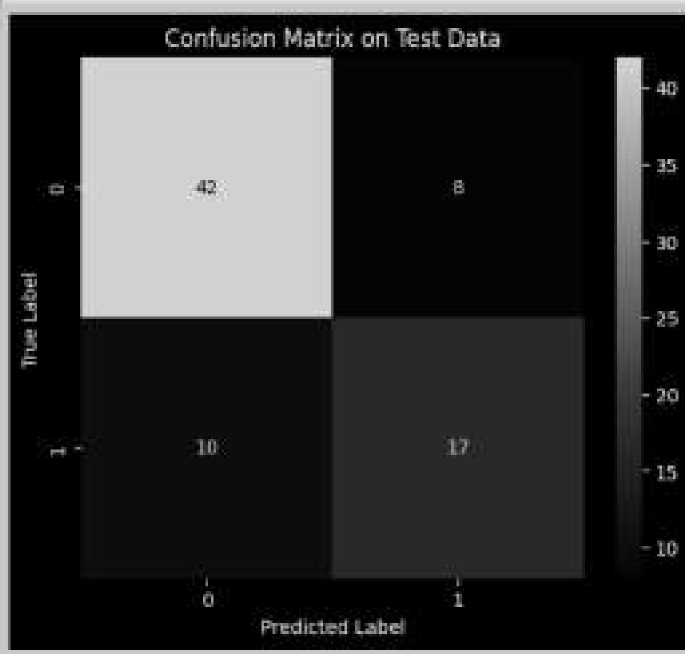
```
# Hyperparameter tuning using GridSearchCV with an SVM classifier
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

svc = SVC(random_state=42)

grid_search = GridSearchCV(estimator=svc,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           n_jobs=-1,
                           verbose=1)

grid_search.fit(X_train_res, y_train_res)

print("Best Hyperparameters:", grid_search.best_params_)
print("Best cross-validation accuracy:", grid_search.best_score_)

Fitting 5 folds for each of 12 candidates, totalling 60 fits
Best Hyperparameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best cross-validation accuracy: 0.8071428571428572
```

- Evaluate the tuned model on the validation and test sets

```
# Evaluate the tuned model on the validation and test sets
best_svc = grid_search.best_estimator_

# Predictions on the validation set
y_val_pred = best_svc.predict(X_val_scaled)

print("Validation Classification Report:")
print(classification_report(y_val, y_val_pred))

# Predictions on the test set
y_test_pred = best_svc.predict(X_test_scaled)

print("Test Classification Report:")
print(classification_report(y_test, y_test_pred))

# Overall accuracy scores
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))

Validation Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.76      0.78       100
           1       0.60      0.67      0.63        54

    accuracy                           0.73       154
   macro avg       0.70      0.71      0.71       154
weighted avg       0.74      0.73      0.73       154

Test Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.84      0.82        50
           1       0.68      0.63      0.65        27

    accuracy                           0.77        77
   macro avg       0.74      0.73      0.74        77
weighted avg       0.76      0.77      0.76        77

Validation Accuracy: 0.7272727272727273
Test Accuracy: 0.7662337662337663
```

- Visualization: Plot confusion matrix for the test set

```
# Visualization: Plot confusion matrix for the test set
cm = confusion_matrix(y_test, y_test_pred)

plt.figure(figsize=(6,5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix on Test Data')
plt.show()
```



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

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

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

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

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

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

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

Ans

- Builds a pipeline with polynomial features and Ridge regression, and tunes hyperparameters using GridSearchCV.

```
def build_and_tune_model(self):
    """Builds a pipeline with polynomial features and Ridge regression, and tunes hyperparameters using GridSearchCV."""
    # Create a pipeline
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('poly', PolynomialFeatures()),
        ('ridge', Ridge())
    ])

    # Hyperparameter grid: tuning polynomial degree and Ridge alpha
    param_grid = {
        'poly__degree': [2, 3, 4, 5],
        'ridge__alpha': [0.1, 1, 10, 100]
    }

    # Grid search with 5-fold CV (note: scoring based on R2 score)
    grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='r2', n_jobs=-1, verbose=1)

    grid_search.fit(self.X_train, self.y_train)

    self.model = grid_search.best_estimator_
    self.best_params_ = grid_search.best_params_

    print("Best hyperparameters:", self.best_params_)
    print("Best cross-validation R2 score:", grid_search.best_score_)
```

- Evaluates the model using the test set and computes R2 and adjusted R2 scores.

```
def evaluate_model(self):
    """Evaluates the model using the test set and computes R2 and adjusted R2 scores."""
    # Predict on test data
    y_pred = self.model.predict(self.X_test)

    # Compute R2 score for each target
    r2_scores = []
    adjusted_r2_scores = []
    n = self.X_test.shape[0]

    # For one independent variable, the number of predictors in the final model is determined by the polynomial degree
    degree = self.best_params_['poly__degree']
    # The number of features created by PolynomialFeatures with one input is: degree + 1
    p = degree

    print("Test Set Evaluation:")
    for i in range(self.y_test.shape[1]):
        r2 = r2_score(self.y_test[:, i], y_pred[:, i])
        # Compute adjusted R2: 1 - (1-R2)*(n-1)/(n-p-1)
        adj_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        r2_scores.append(r2)
        adjusted_r2_scores.append(adj_r2)
        print(f"Dependent Variable {i+1} -- R2: {r2:.4f}, Adjusted R2: {adj_r2:.4f}")

    # Check if all adjusted R2 scores meet the threshold
    if all(adj >= 0.99 for adj in adjusted_r2_scores):
        print("All adjusted R2 scores are above 0.99")
    else:
        print("Warning: Not all adjusted R2 scores are above 0.99")

    return y_pred, r2_scores, adjusted_r2_scores
```

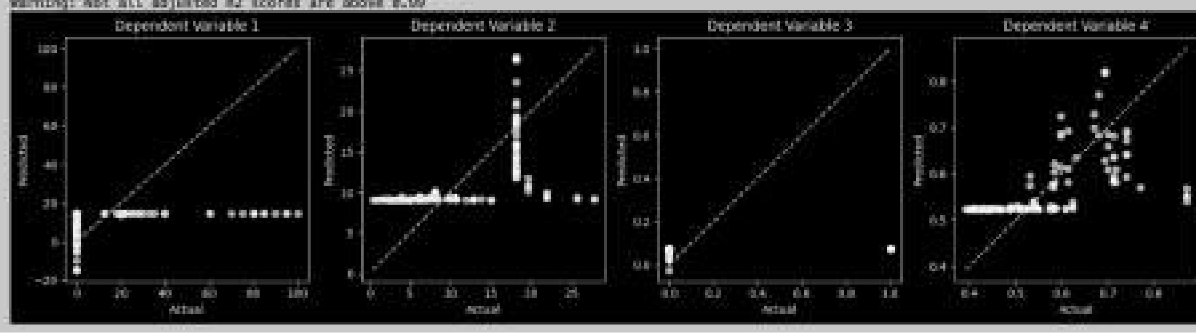
- Visualizes the predictions vs. actual values for each dependent variable.

```
def visualize_results(self, y_pred):
    """Visualizes the predictions vs. actual values for each dependent variable."""
    num_targets = self.y_test.shape[1]

    plt.figure(figsize=(15, 4))
    for i in range(num_targets):
        plt.subplot(1, num_targets, i+1)
        plt.scatter(self.y_test[:, i], y_pred[:, i], alpha=0.7, color='blue')
        plt.plot([self.y_test[:, i].min(), self.y_test[:, i].max()],
                 [self.y_test[:, i].min(), self.y_test[:, i].max()], 'r--')
        plt.xlabel('Actual')
        plt.ylabel('Predicted')
        plt.title(f'Dependent Variable {i+1}')
    plt.tight_layout()
    plt.show()
```

- Final Evaluation Result

```
[3] # Instantiate the BostonHousingRegression class with the dataset path
    bh_reg = BostonHousingRegression('BostonHousing.csv')

    # Run the complete regression analysis
    bh_reg.run()
```

```
Data loaded successfully.
X shape: (506, 1)  y shape: (506, 4)
Training set shape: (354, 1) (354, 4)
Test set shape: (152, 1) (152, 4)
Fitting 5 folds for each of 16 candidates, totalling 80 fits
Best hyperparameters: {'poly__degree': 2, 'ridge__alpha': 10}
Best cross-validation R2 score: 0.1448114309471162
Test Set Evaluation:
Dependent Variable 1 — R2: 0.0658, Adjusted R2: 0.0533
Dependent Variable 2 — R2: 0.3148, Adjusted R2: 0.3058
Dependent Variable 3 — R2: 0.0023, Adjusted R2: -0.0111
Dependent Variable 4 — R2: 0.3667, Adjusted R2: 0.3582
Warning: Not all adjusted R2 scores are above 0.09
```



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

Ans:

Key Features and Their Importance

The Wine Quality dataset (available on Kaggle) consists of various physicochemical properties of wine samples. The target is to predict wine quality (rated 0–10) based on these features. Below are the main features and their significance:

- **Fixed Acidity**: Refers to non-volatile acids that contribute to the wine's taste and structure. Plays a role in freshness and sharpness.

- **Volatile Acidity**: High values lead to an undesirable vinegar-like taste. It's a key indicator of wine spoilage.

- **Citric Acid**: Adds flavor and freshness. Higher amounts usually enhance the wine's sensory appeal.

- **Residual Sugar**: Represents the amount of sugar left after fermentation. It affects sweetness and body.

- **Chlorides**: Reflects the salt content. Excessive chlorides can negatively affect taste.

- **Free Sulfur Dioxide**: Used to prevent microbial growth. Its balance is crucial for preservation without affecting flavor.

- **Total Sulfur Dioxide**: Sum of all SO_2 forms. High levels can produce off-odors and suppress aroma.

- **Density**: Correlates with sugar and alcohol content. Affects texture and perception of richness.

- **pH**: Indicates acidity or alkalinity. Impacts wine stability and freshness.

- **Sulphates**: Act as preservatives. Moderate levels enhance flavor and longevity.

- **Alcohol**: A key determinant of wine quality. Higher alcohol levels often correlate with better ratings due to improved mouthfeel and aroma.

Among these, alcohol, volatile acidity, and sulphates are the most influential in predicting wine quality. A good balance of acidity, sugar, and alcohol is essential.

Handling Missing Data in Feature Engineering

Although the wine quality dataset is typically clean, handling missing data is a crucial step in any data preprocessing pipeline.

Common Imputation Techniques:

1. **Mean/Median Imputation**
    - Replace missing numerical values with the column's mean or median.
    - Pros: Simple and fast.
    - Cons: Can distort the data distribution, especially if data is skewed.
2. **Mode Imputation**
    - Best for categorical variables, replacing missing values with the most frequent category.
    - Pros: Maintains category consistency.
    - Cons: Can reduce variance and mask true data diversity.
3. **K-Nearest Neighbors (KNN) Imputation**
    - Estimates missing values based on similar records.
    - Pros: Maintains local data structure.
    - Cons: Computationally expensive and sensitive to irrelevant features.
4. **Multivariate Imputation (e.g., MICE)**
    - Uses regression models to estimate missing values based on other features.
    - Pros: Captures complex relationships between features.
    - Cons: More complex and resource-intensive.