FACULTY OF ENGINEERING AND TECHNOLOGY

BACHELOR OF TECHNOLOGY

MACHINE LEARNING

LABORATORY

(303105354)

6TH – SEMESTER

(ARTIFICIAL INTELLIGENCE)

COMPUTER SCIENCE & ENGINEERING

DEPARTMENT

LAB MANUAL

CERTIFICATE

This Is To Certify That Mr./Ms. **BHANUVARDHAN .MEDAPALLI**

With Enrollment No : **2203031240160** Has Completed His/Her

Laboratory Experiments **(303105354)** From The Department Of

COMPUTER SCIENCE (ARTIFICIAL INTELLGENCE)

During The Academic Year **2024 – 2025**

Date of Submission :……………..…………….. Staff Incharge:……………..…………..

Head Of Department :…………………………..

INDEX

SR.NO	EXPERIMENT	PAGE NO		DATE OF COMPLETION	SIGN	MARKS
		FROM	TO			
1.	Dealing with Data using Numpy, Pandas, Statistics libray					
2.	Data Analysis & Visualization on Diwali Sales Dataset.					
3.	Implement linear regression and logistic regression.					
4.	implement the naïve Bayesian classifier for a sample training data set stored as a (.CSV) file. Compute the accuracy of the classifier, considering a few test data sets.					
5.	Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task.					
6.	Decision tree-based ID3 algorithm.					
7.	Write a program to implement the K-Nearest Neighbour algorithm to classify the iris data set.					
8.	Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm.					
9.	Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.					
10.	Compare the various supervised learning algorithm by using appropriate dataset. (Linear Regression, Support Vector Machine, Decision Tree).					

11.	Compare the various Unsupervised learning algorithm by using the appropriate datasets. (K Means Clustering, K Mode,).					
12.	Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.					

PRACTICAL - 1

❖ **AIM :** DEALING WITH DATA USING NUMPY, PANDAS, STATISTICS LIBRARY.

❖ **PROCEDURE :**

❖ **NUMPY :**

- CREATION OF AN ARRAY : Employ the APIs np.array() or np.zeros(), np.ones(), and np.arange() in arrays creation.

- OPERATIONS OF AN ARRAY : Carry out operations on the array elements, broadcasting and np functions such as np.sum(), np.mean()..numerical operations.

- INDEXING AND SLICING : Use indexing and slicing methods to construct and reconstruct the arrays.

- LINEAR ALGEBRA : Apply np.dot(), np.linalg.inv() and np.linalg.eig() to perform required matrix operations.

- RANDOM NUMBER GENERATION / DISTRIBUTION : Apply the method np.random or generate random numbers, obtain random samples and shuffle data.

- PERFORMANCE : Thus, for numerical computation, NumPy arrays tend to be more memory efficient and faster than using Python lists.

❖ **PANDAS :**

- DATA STRUCTURES : the most primary data structure used are Series (1D) and DataFrame (2D)

- DATA IMPORT/EXPORT : For importing and exporting datasets, use pd.read_csv(), for example, pd.read_excel() and df.to_csv () for preparing and presenting data.

- DATA MANIPULATION : Some operations are performed such as filtering, groupby, merging using merge, concatenating with concat.

- HANDLING MISSING DATA : To deal with the missing values df.isnull(), df.fillna() and df.dropna() are required.

- DATA TRANSFORMATION : Data Frames can be transformed by applying functions through apply, map, and applymap.

- TIME SERIES ANALYSIS : Use the functions exposed by the Pandas package for any kind of time and date data, including resampling and time zone.

❖ **STATISTICS LIBRARY :**

- DESCRIPTIVE STATISTICS : Calculate measures of central tendency and dispersion to summarize data.

- PROBABILITY DISTRIBUTIONS : Understand and use functions for normal, binomial, and other distributions.

- HYPOTHESIS TESTING : Conduct tests like t-tests and chi-squared tests to make inferences about populations.

- CORRELATION AND REGRESSION : Use correlation coefficients and regression analysis to explore relationships between variables.

- DATA VISUALIZATION : While the Statistics library does not provide visualization, integrate with libraries like Matplotlib or Seaborn for graphical representation of statistical results.

❖ PROGRAM :

IMPORTING LIBRARIES :

```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12, 5);
sns.set_style('whitegrid')
import matplotlib.colors as mcolors
! pip install dexplot
import dexplot as dxp
import re
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
```

```
Requirement already satisfied: dexplot in /usr/local/lib/python3.10/dist-packages (0.1.2)
Requirement already satisfied: matplotlib>=3.1 in /usr/local/lib/python3.10/dist-packages (from dexplot) (3.8.0)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from dexplot) (2.2.2)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist-packages (from dexplot) (1.26.4)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-packages (from dexplot) (1.13.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.1->dexplot) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->dexplot) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->dexplot) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=3.1->dexplot) (1.17.0)
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

READ CSV FILE : df=pd.read_csv('/content/zomato.csv')

HEAD : df.head()

```
df.head()
```

	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	dish_liked	cuisines	approx_cost(for two people)	reviews_list	menu_item	listed_in(type)	listed_in(city)
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4.1/5	775	080 42297555\r\n+91 9743772233	Banashankari	Casual Dining	Pasta, Lunch Buffet, Masala Papad, Paneer Laja...	North Indian, Mughlai, Chinese	800	[('Rated 4.0', 'RATED\n A beautiful place to ...	[]	Buffet	Banashankari
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1/5	787	080 41714161	Banashankari	Casual Dining	Momos, Lunch Buffet, Chocolate Nirvana, Thai G...	Chinese, North Indian, Thai	800	[('Rated 4.0', 'RATED\n Had been here for din...	[]	Buffet	Banashankari
2	https://www.zomato.com/SanchurroBangalore?cont...	1112, Next to KIMS Medical College, 17th Cross...	San Churro Cafe	Yes	No	3.8/5	918	+91 9663487993	Banashankari	Cafe, Casual Dining	Churros, Cannelloni, Minestrone Soup, Hot Choc...	Cafe, Mexican, Italian	800	[('Rated 3.0', "RATED\n Ambience is not that ...	[]	Buffet	Banashankari
3	https://www.zomato.com/bangalore/addhuri-udupi...	1st Floor, Annakuteera, 3rd Stage, Banashankar...	Addhuri Udupi Bhojana	No	No	3.7/5	88	+91 9620009302	Banashankari	Quick Bites	Masala Dosa	South Indian, North Indian	300	[('Rated 4.0', "RATED\n Great food and proper...	[]	Buffet	Banashankari
4	https://www.zomato.com/bangalore/grand-village...	10, 3rd Floor, Lakshmi Associates, Gandhi Baza...	Grand Village	No	No	3.8/5	166	+91 8026612447\r\n+91 9901210005	Basavanagudi	Casual Dining	Panipuri, Gol Gappe	North Indian, Rajasthani	600	[('Rated 4.0', 'RATED\n Very good restaurant ...	[]	Buffet	Banashankari

TAIL : df.tail()

```
df.tail()
```

	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	dish_liked	cuisines	approx_cost(for two people)	reviews_list	menu_item	listed_in(type)	listed_in(city)
51712	https://www.zomato.com/bangalore/best-brews-fo...	Four Points by Sheraton Bengaluru, 43/3, White...	Best Brews - Four Points by Sheraton Bengaluru...	No	No	3.6 /5	27	080 40301477	Whitefield	Bar	NaN	Continental	1,500	[('Rated 5.0', "RATED\n Food and service are ...	[]	Pubs and bars	Whitefield
51713	https://www.zomato.com/bangalore/vinod-bar-and...	Number 10, Garudachar Palya, Mahadevapura, Whi...	Vinod Bar And Restaurant	No	No	NaN	0	+91 8197675843	Whitefield	Bar	NaN	Finger Food	600	[]	[]	Pubs and bars	Whitefield
51714	https://www.zomato.com/bangalore/plunge-sherat...	Sheraton Grand Bengaluru Whitefield Hotel & Co...	Plunge - Sheraton Grand Bengaluru Whitefield H...	No	No	NaN	0	NaN	Whitefield	Bar	NaN	Finger Food	2,000	[]	[]	Pubs and bars	Whitefield
51715	https://www.zomato.com/bangalore/chime-sherato...	Sheraton Grand Bengaluru Whitefield Hotel & Co...	Chime - Sheraton Grand Bengaluru Whitefield Ho...	No	Yes	4.3 /5	236	080 49652769	ITPL Main Road, Whitefield	Bar	Cocktails, Pizza, Buttermilk	Finger Food	2,500	[('Rated 4.0', 'RATED\n Nice and friendly pla...	[]	Pubs and bars	Whitefield
51716	https://www.zomato.com/bangalore/the-nest-the-...	ITPL Main Road, KIADB Export Promotion Industr...	The Nest - The Den Bengaluru	No	No	3.4 /5	13	+91 8071117272	ITPL Main Road, Whitefield	Bar, Casual Dining	NaN	Finger Food, North Indian, Continental	1,500	[('Rated 5.0', 'RATED\n Great ambience , look...	[]	Pubs and bars	Whitefield

INFO : df.info()

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   url                          51717 non-null  object
 1   address                      51717 non-null  object
 2   name                         51717 non-null  object
 3   online_order                 51717 non-null  object
 4   book_table                   51717 non-null  object
 5   rate                         43942 non-null  object
 6   votes                        51717 non-null  int64
 7   phone                        50509 non-null  object
 8   location                     51696 non-null  object
 9   rest_type                    51490 non-null  object
 10  dish_liked                   23639 non-null  object
 11  cuisines                     51672 non-null  object
 12  approx_cost(for two people)  51371 non-null  object
 13  reviews_list                 51717 non-null  object
 14  menu_item                    51717 non-null  object
 15  listed_in(type)              51717 non-null  object
 16  listed_in(city)              51717 non-null  object
dtypes: int64(1), object(16)
memory usage: 6.7+ MB
```

BOOLEAN VALUES : df.isnull().sum()

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

	0
url	0
address	0
name	0
online_order	0
book_table	0
rate	7775
votes	0
phone	1208
location	21
rest_type	227
dish_liked	28078
cuisines	45
approx_cost(for two people)	346
reviews_list	0
menu_item	0
listed_in(type)	0
listed_in(city)	0

dtype: int64

DESCRIBE : df.describe()

```
df.describe()
```

	votes
count	12654.000000
mean	268.741189
std	759.931003
min	0.000000
25%	7.000000
50%	41.000000
75%	186.000000
max	16345.000000

SHAPE OF CSV FILE : df.shape

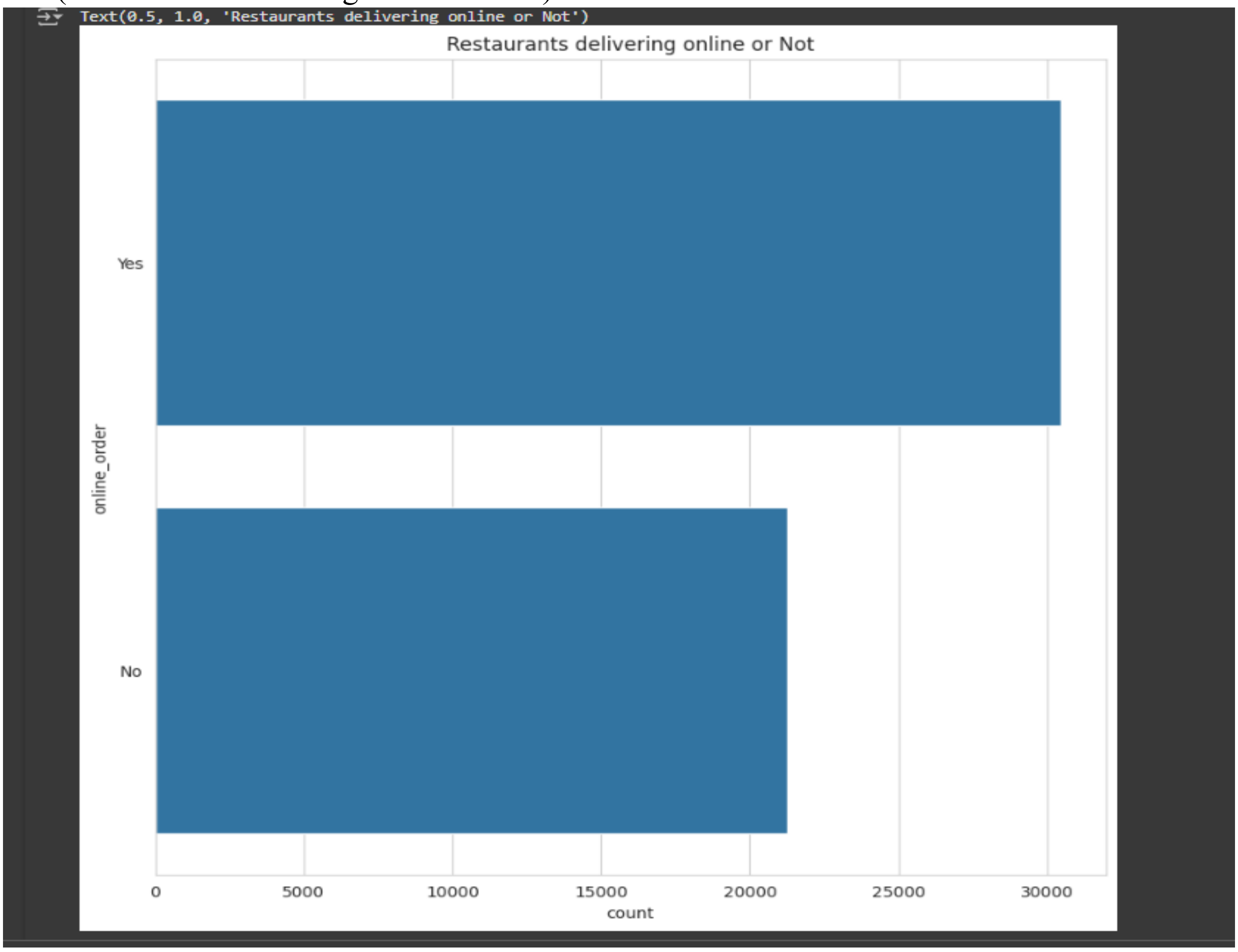
```
df.shape
(12655, 17)
```

DISTINICT VALUES OF COLUMNS : df['menu_item'].unique()

```
df['menu_item'].unique()

array(['[]',
       "['Chocolate Fantasy (Pack Of 5)', 'Pan Cake (Pack Of 6)', 'Gulab Jamun (Pack Of 10)', 'Gulkand Shot (Pack Of 5)', 'Chocolate Decadence (Pack of 2)', 'CheeseCake (Pack Of 2)', 'Red Velvet Slice Cake (Pack of 2)', 'Red Velvet Slice Cake & Cheese Cake (Pack of 2)', 'Red Velvet Slice Cake & Chocolate Decadence Cake (Pack of 2)', 'Hazelnut Brownie (Pack of 2)', 'Moments', 'Red Velvet Cake With Butter Cream Frosting (750 Gm)', 'Red Velvet Slice Cake (Pack of 2)', 'Red Velvet Slice Cake & Cheese Cake (Pack of 2)', 'Red Velvet Slice Cake & Chocolate Decadence Cake (Pack of 2)', 'Red Velvet Slice Cake (Pack of 1)', 'Valentine Red Velvet Jar', 'Valentine Chocolate Jar', 'Valentines Jar Combo', 'Pink Guava 500 ML', 'Oreo Vanilla 500 ML', 'Cookie Crumble 500 ML', 'Chocolate Fantasy', 'Gulkand-E-Bahar', 'Pan Cake', 'Hazelnut Brownie (Pack Of 1)', 'Gulab Jamun (Pack Of 2)', 'Plum Cake', 'Red Velvet Cake With Butter Cream Frosting (750 Gm)', 'Chocolate Mud Cake (700 Gms)', 'CheeseCake (Pack of 1)', 'Chocolate Decadence (Pack of 1)', 'Red Velvet Slice Cake (Pack of 1)']",
       "['Chole Kulcha Meal', 'Upvas Aloo Paratha With Dahi', 'Singhada Aloo Paratha with Hare Tamatar Ki Sabji', 'Smoked Butter Chicken Combo', 'Paneer Methi Chaman Combo', 'Mutton Bhuna Combo', 'Rajma Masala Meal', 'Dal Makhani Veg Starter Combo', 'Dal Makhani Non-Veg Starter Combo', 'Malai Kofta Combo', 'Jumbo Chicken Wrap', 'Jumbo Veg Wrap', 'Jumbo Falafel Salsa Wrap', 'Chicken Overload Jumbo Wrap', 'Veg Pizza Wrap', 'Chicken Pizza Wrap', 'Mexican Potato Salsa Wrap', 'American Smokey Sausage Wrap', 'Makhani Falafel Wrap', 'Mutton Overload Wrap', 'Mac & Cheese Chicken Wrap', 'Mac & Cheese Veg Wrap', 'Barbeque Chicken Wrap', 'Mutton Boti Wrap', 'Masala Paneer Tikka Wrap', 'Fiery Paneer Tikka Wrap', 'Masala Chicken Tikka Wrap', 'Cheesy Corn Salsa Wrap', 'Chicken Mayo Wrap', 'Cheese Melt Chicken Wrap', 'Cheese Melt Paneer Wrap', 'Double Cheese Meatball Wrap', 'Reshmi Chicken Kebab Wrap', 'Egg Cheese Sausage Wrap', 'Double Egg Chatpata Wrap', 'Cheesy Potato Wrap', 'Veg Falafel Wrap', 'Chicken Bhuna Wrap', 'Chatpate Chole Wrap', 'Fiery Paneer Wrap', 'Fiery Chicken Wrap', 'Mac & Cheese Wrap', 'Hare Tamatar & Sabudana Wada Royal Thali', 'Singhada Aloo Paratha & Hare Tamatar Royal Satvik Meal', 'Smoked Butter Chicken With Omelette', 'Rajma Masala Royal Combo', 'Paneer Methi Chaman Royal Combo', 'Mutton Bhuna Royal Combo', 'Smoked Butter Chicken Royal Combo', 'Malai Kofta Royal Combo', 'Fusion Breakfast', 'Pan Cake', 'Aloo Paratha Combo', 'Chai for 4', 'Cheesy Chicken Meatballs', 'Peach Tea (Serves 4)', 'Falafel Nuggets with Mayo Dip', 'Potato Chilli Shots with Mayo Dip', 'Pan Cake', 'Kashmiri Kahwa (Serves 4)', 'Masala Chai (Serves 4)', 'Chai for 4 + Nature valley bar', 'Kulcha', 'Flavorful Rice Tub', 'Dal Makhani Bowl (Half KG)', 'Smoked Butter Chicken Bowl (Half KG)', 'Mutton Bhuna Bowl (Half KG)', 'Singhada Aloo Paratha Tub', 'Curd Bowl (Half KG)', 'Hare Tamatar Ki Subji (Half KG)', 'Rajma Masala Bowl (Half KG)', 'Triangle Paratha Tub', 'Malai Kofta Bowl (Half Kg)', 'Chocolate Fantasy', 'SWIG Jeera Masala', 'SWIG Green apple', 'Kesar Muesli', 'Gulab Jamun (Pack Of 2)', 'Plum Cake', 'Gulab Jamun (Pack of 1)', 'Moments', 'Hazelnut Brownie (Pack Of 1)', 'CheeseCake (Pack of 1)', 'Chocolate Decadence (Pack of 1)', 'Red Velvet Slice Cake (Pack of 1)', 'Mint Chaas']",
```

PLOT GRAPH OF RESTAURANTS DELIVERING ONLINE OR NOT :

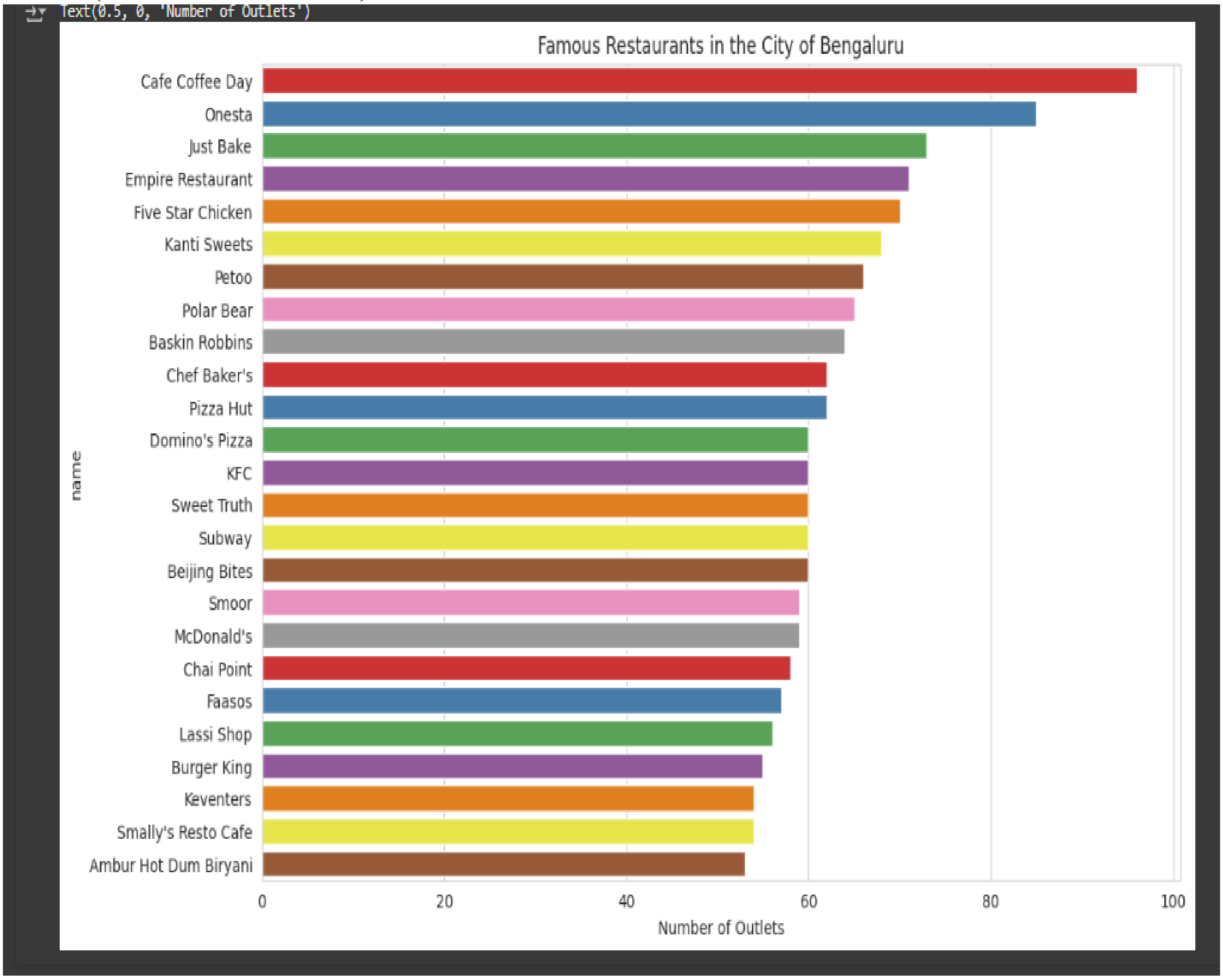
# Plot of the Restaurant, whether they are accepting the online_order or not
sns.countplot(df['online_order'])
fig = plt.gcf()
fig.set_size_inches(10,10)
plt.title('Restaurants delivering online or Not')



PLOT GRAPH OF FAMOUS RESTAURANTS IN THE CITY OF BENGALURU :

Plotting for the Top Restuarant in the Bengaluru

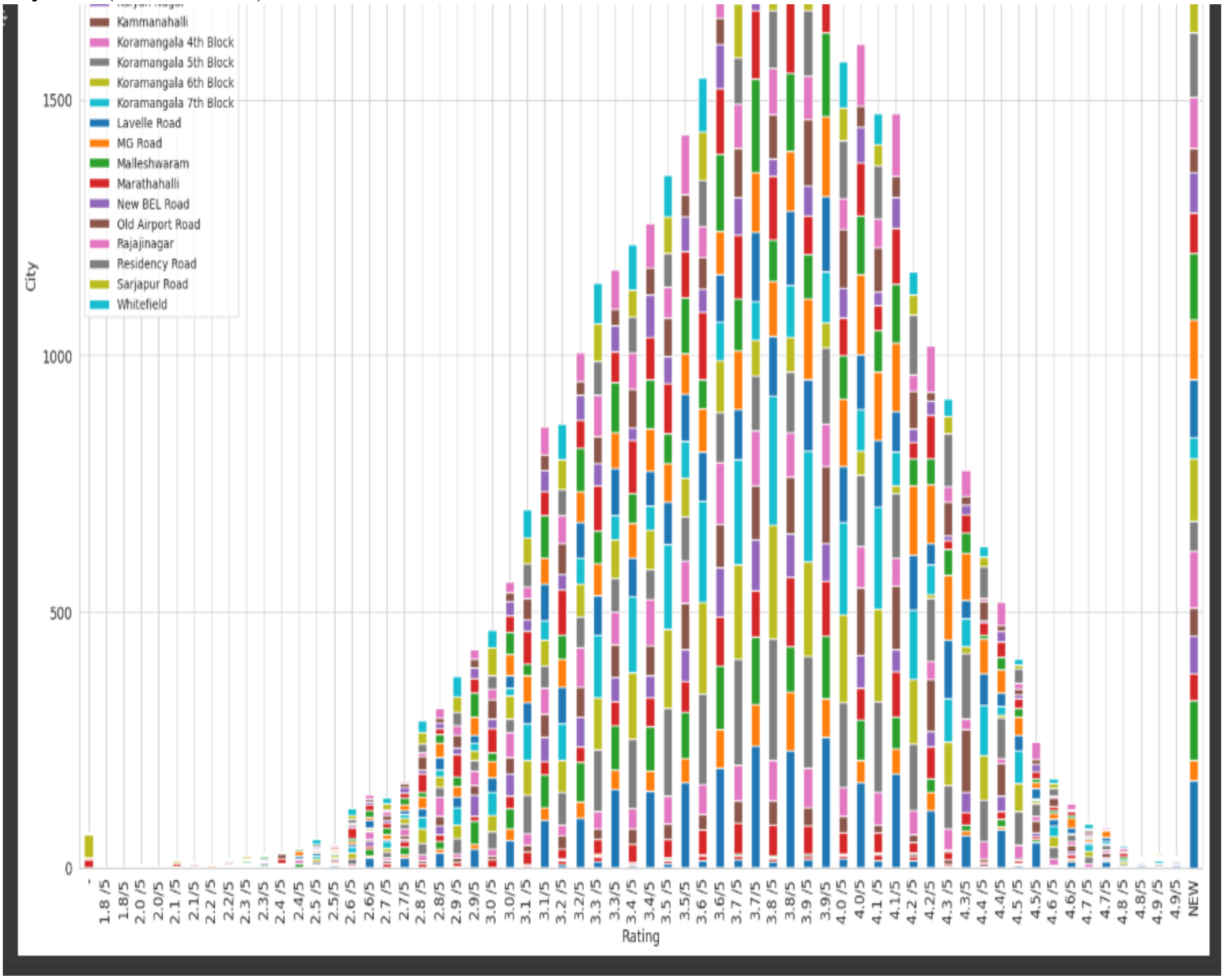
```
plt.figure(figsize=(12,8))
val = df['name'].value_counts()[:25]  # Considering for the top 25 restaurants
sns.barplot(x=val, y=val.index, palette = "Set1")
plt.title("Famous Restaurants in the City of Bengaluru")
plt.xlabel("Number of Outlets")
```



PLOT GRAPH OF RATING :

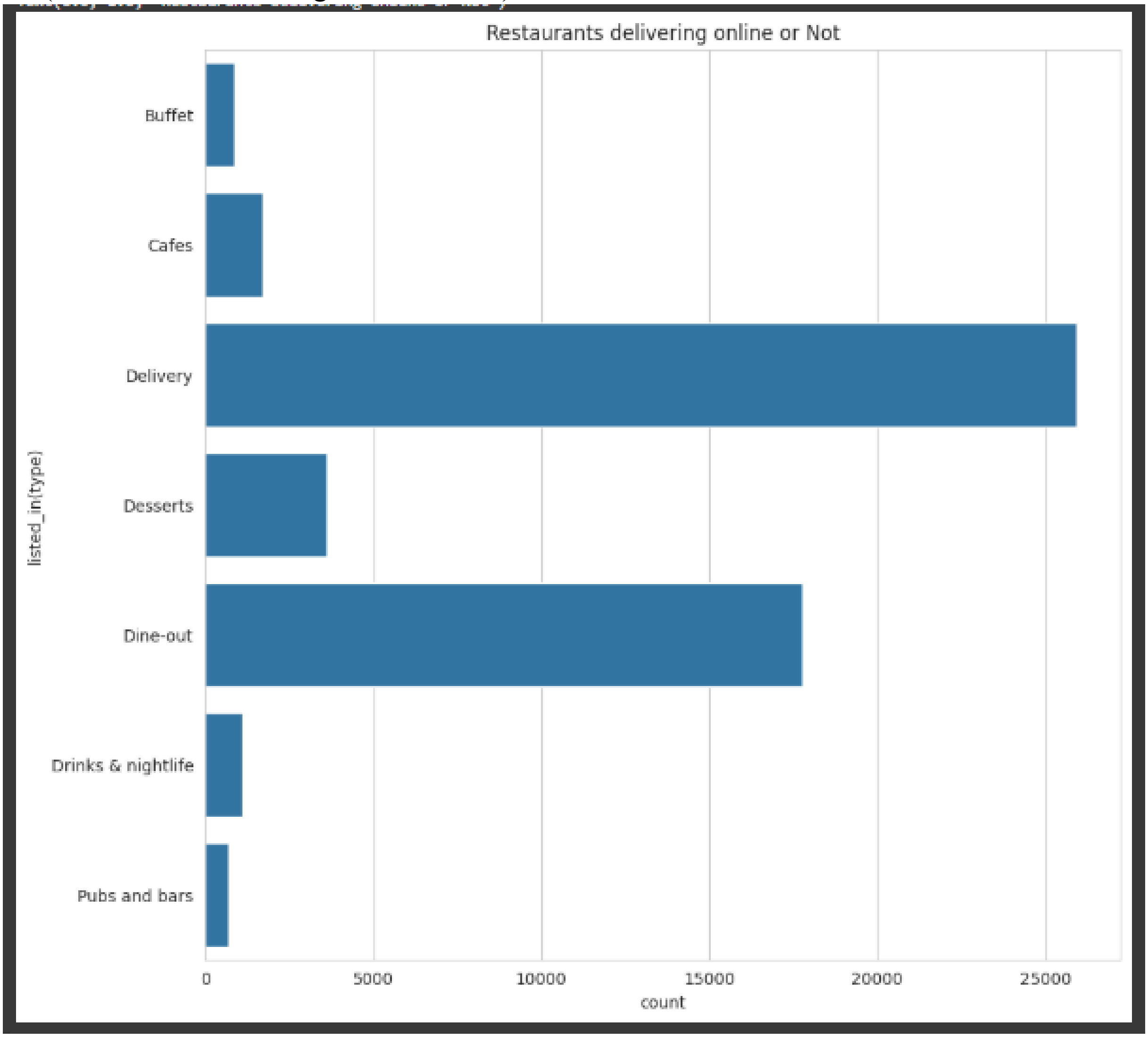
Lets plot to see that whether there is any relation between the rating of the restaurant and the area in which they are located.

rel_type = pd.crosstab(df['rate'], df['listed_in(city)'])
rel_type.plot(kind='bar', stacked=True, figsize=(20,16))
plt.title('City - Rating',fontsize=18)
plt.ylabel('City',fontsize=12)
plt.xlabel('Rating',fontsize=12)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)



PLOT GRAPH OF RESTAURANTS DELIVERING ONLINE OR NOT :

```
sns.countplot(df['listed_in(type)'])
fig = plt.gcf()
fig.set_size_inches(10,10)
plt.title('Restaurants delivering online or Not')
```



PRACTICAL - 2

❖ **AIM :** DATA ANALYSIS AND DATA VISUALIZATION ON DIWALI SALES DATASET.

❖ **PROCEDURE :**

Clearly outline the goals of your analysis. For instance, you might aim to understand sales trends during the Diwali festival, identify popular products, or analyze customer demographics.

1. DATA COLLECTION :
   Obtain the Diwali sales dataset, typically available in CSV format. This dataset should include relevant fields such as :
   - **Date**: Date of sale
   - **Sales**: Sales amount
   - **Product**: Type of product sold
   - **User_ID**: Customer ID
   - **Cust_name**: Customer name
   - **Gender**: Customer gender
   - **Age Group**: Age category of the customer
   - **Marital_Status**: Customer's marital status
   - **State**: State information
   - **Product_Category**: Category of the product sold

2. DATA CLEANING :
   Prepare the data for analysis by :
   - Checking for missing values and handling them appropriately (e.g., imputation or removal).
   - Correcting inconsistencies in data formats.
   - Removing any duplicate entries.

3. EXPLORATORY DATA ANALYSIS (EDA) :
   Conduct EDA to gain insights into the dataset :
   - Use Python libraries like Pandas to load and manipulate the data.
   - Generate summary statistics (mean, median, mode) to understand central tendencies.
   - Identify patterns or correlations among variables using correlation matrices.

4. DATA VISUALIZATION :
   Create visual representations of your findings to make complex data understandable :
   - Utilize libraries such as Matplotlib and Seaborn in Python for visualizations.
   - Common visualizations include :
     - **Bar Charts**: To show sales by product category.
     - **Line Graphs**: To illustrate sales trends over time.
     - **Pie Charts**: To represent the proportion of sales by gender or age group.
     - **Heat Maps**: To visualize sales performance across different states or regions.

5. INTERPRETATION OF RESULTS :
   Analyze the visualizations and summarize key insights :
   - Identify which products had the highest sales.
   - Understand customer demographics that contributed to sales.
   - Recognize any seasonal trends or anomalies in sales data.

6. REPORTING FINDINGS :
   Prepare a report or presentation that includes :
   - Visualizations that support your findings.
   - A narrative that explains the insights derived from the analysis.
   - Recommendations based on your analysis, such as marketing strategies or inventory management.

❖ PROGRAM :

IMPORTING LIBRARIES :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt  # visualizing data%matplotlib inline
import seaborn as sns
```

READING CSV FILE :

```
# import csv file
df = pd.read_csv('Diwali Sales Data.csv', encoding= 'unicode_escape')
```

SHAPE OF CSV FILE : df.shape

```
df.shape
```

```
(11251, 15)
```

HEAD : df.head()

```
df.head()
```

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	Marital_Status	State	Zone	Occupation	Product_Category	Orders	Amount	Status	unnamed1
0	1002903	Sanskriti	P00125942	F	26-35	28	0	Maharashtra	Western	Healthcare	Auto	1	23952.0	NaN	NaN
1	1000732	Kartik	P00110942	F	26-35	35	1	Andhra Pradesh	Southern	Govt	Auto	3	23934.0	NaN	NaN
2	1001990	Bindu	P00118542	F	26-35	35	1	Uttar Pradesh	Central	Automobile	Auto	3	23924.0	NaN	NaN
3	1001425	Sudevi	P00237842	M	0-17	16	0	Karnataka	Southern	Construction	Auto	2	23912.0	NaN	NaN
4	1000588	Joni	P00057942	M	26-35	28	1	Gujarat	Western	Food Processing	Auto	2	23877.0	NaN	NaN

TO CHECK DATATYPES OF ALL COLUMN : df.info()

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11251 entries, 0 to 11250
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   User_ID           11251 non-null  int64
 1   Cust_name         11251 non-null  object
 2   Product_ID        11251 non-null  object
 3   Gender            11251 non-null  object
 4   Age Group         11251 non-null  object
 5   Age               11251 non-null  int64
 6   Marital_Status    11251 non-null  int64
 7   State             11251 non-null  object
 8   Zone              11251 non-null  object
 9   Occupation        11251 non-null  object
 10  Product_Category  11251 non-null  object
 11  Orders            11251 non-null  int64
 12  Amount            11239 non-null  float64
 13  Status            0 non-null      float64
 14  unnamed1          0 non-null      float64
dtypes: float64(3), int64(4), object(8)
memory usage: 1.3+ MB
```

DROP UNRELATED/BLANK COLUMNS :

df.drop(['Status', 'unnamed1'], axis=1, inplace=True)
#check for null values
pd.isnull(df).sum()

```
#check for null values

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

```
User_ID              0
Cust_name            0
Product_ID           0
Gender               0
Age Group            0
Age                  0
Marital_Status       0
State                0
Zone                 0
Occupation           0
Product_Category     0
Orders               0
Amount              12
dtype: int64
```

CHANGE DATA TYPE :
df['Amount'] = df['Amount'].astype('int')
df['Amount'].dtypes

```
# change data type
df['Amount'] = df['Amount'].astype('int')

df['Amount'].dtypes

... dtype('int32')
```

COLUMNS : df.columns

```
df.columns

... Index(['User_ID', 'Cust_name', 'Product_ID', 'Gender', 'Age Group', 'Age',
       'Marital_Status', 'State', 'Zone', 'Occupation', 'Product_Category',
       'Orders', 'Amount'],
      dtype='object')
```

DESCRIBE RETURNS DESCRIPTION OF THE DATA IN THE DATAFRAME : df.describe()

```
df.describe()
```

	User_ID	Age	Marital_Status	Orders	Amount
count	1.123900e+04	11239.000000	11239.000000	11239.000000	11239.000000
mean	1.003004e+06	35.410357	0.420055	2.489634	9453.610553
std	1.716039e+03	12.753866	0.493589	1.114967	5222.355168
min	1.000001e+06	12.000000	0.000000	1.000000	188.000000
25%	1.001492e+06	27.000000	0.000000	2.000000	5443.000000
50%	1.003064e+06	33.000000	0.000000	2.000000	8109.000000
75%	1.004426e+06	43.000000	1.000000	3.000000	12675.000000
max	1.006040e+06	92.000000	1.000000	4.000000	23952.000000

df[['Age', 'Orders', 'Amount']].describe()

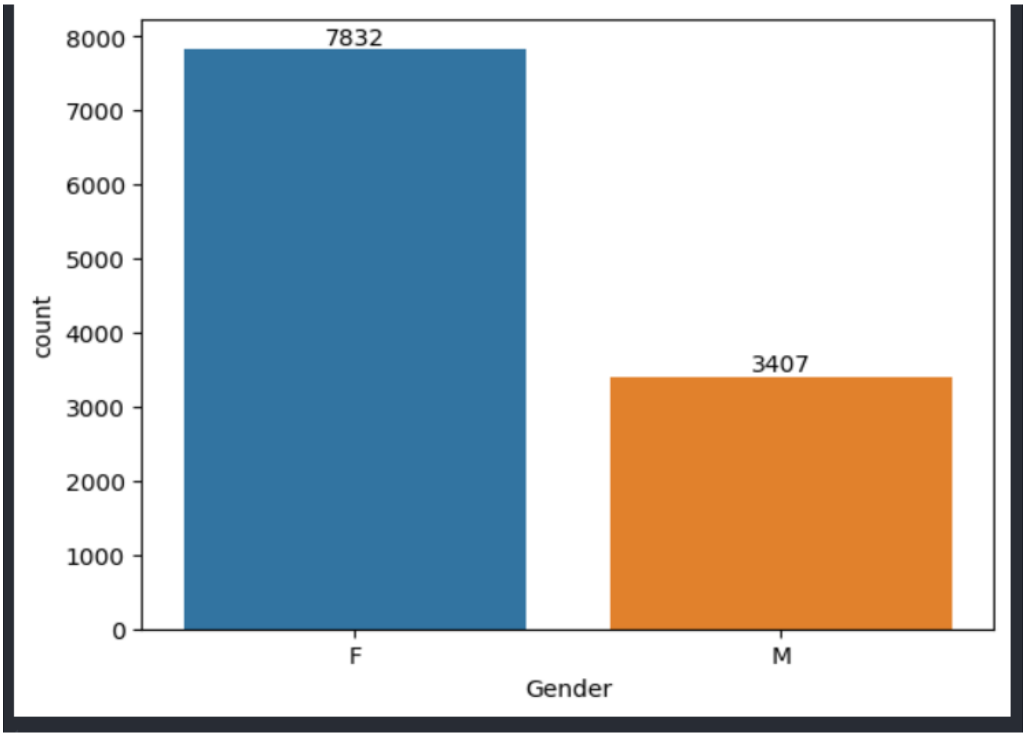
```
# use describe() for specific columns

df[['Age', 'Orders', 'Amount']].describe()
```

	Age	Orders	Amount
count	11239.000000	11239.000000	11239.000000
mean	35.410357	2.489634	9453.610553
std	12.753866	1.114967	5222.355168
min	12.000000	1.000000	188.000000
25%	27.000000	2.000000	5443.000000
50%	33.000000	2.000000	8109.000000
75%	43.000000	3.000000	12675.000000
max	92.000000	4.000000	23952.000000

EXPLORATORY DATA ANALYSIS :

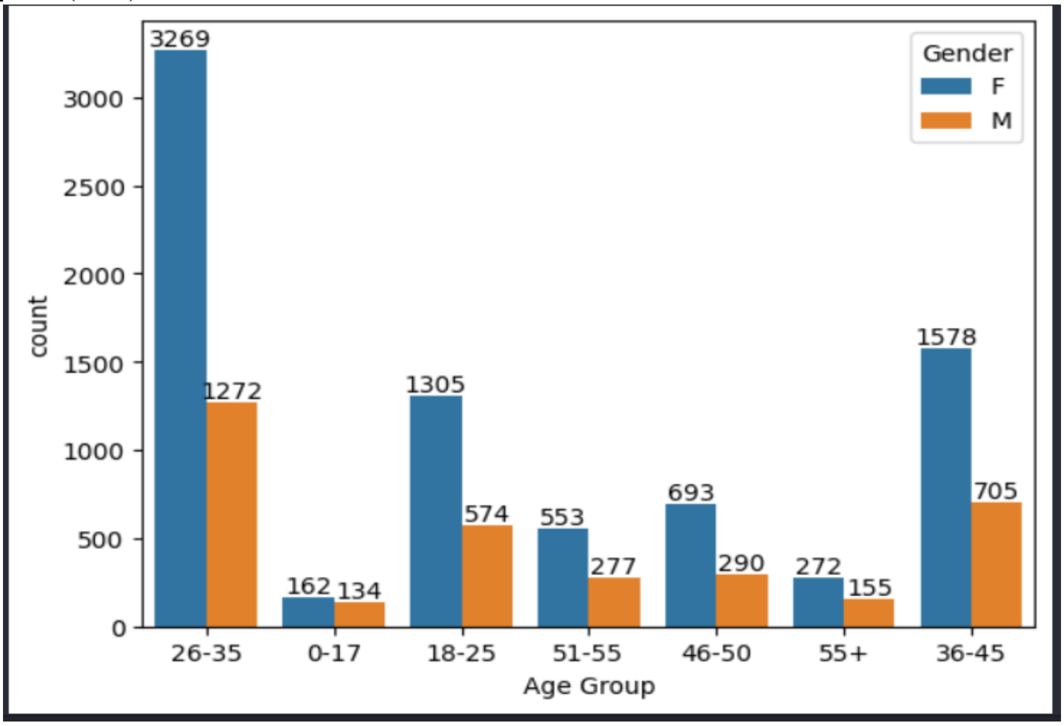
GENDER GRAPH PLOTING :

```
# plotting a bar chart for Gender and it's count
ax = sns.countplot(x = 'Gender',data = df)
for bars in ax.containers:
    ax.bar_label(bars)
```

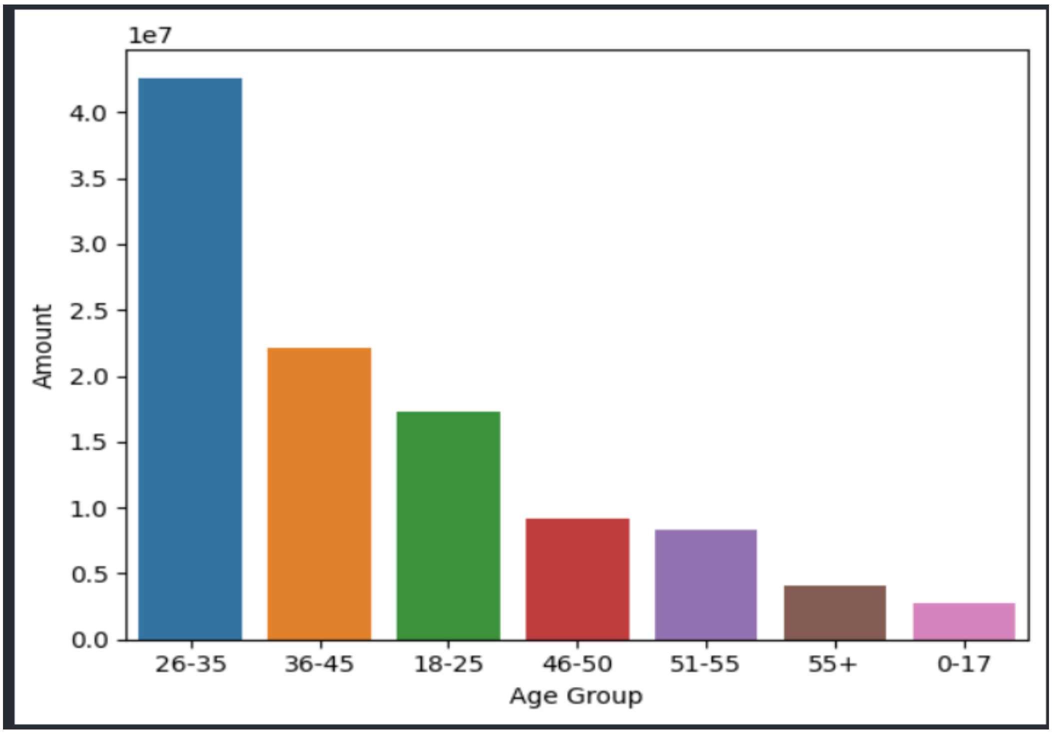


<u>AGE GRAPH PLOTING</u> :

```
ax = sns.countplot(data = df, x = 'Age Group', hue = 'Gender')
for bars in ax.containers:
    ax.bar_label(bars)
```

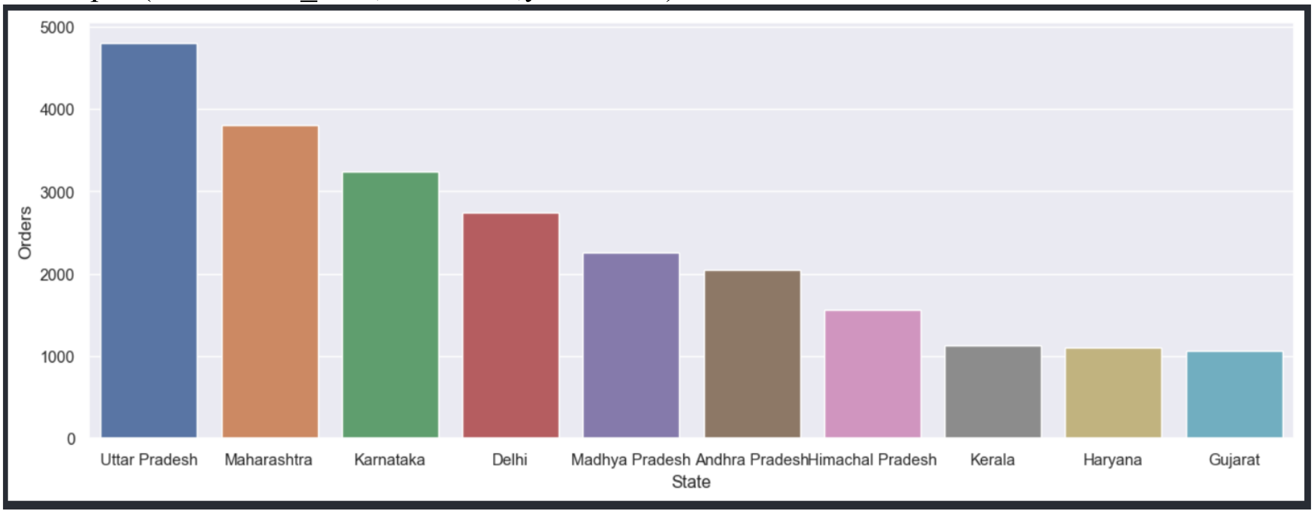



```
# Total Amount vs Age Group
sales_age = df.groupby(['Age Group'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)
sns.barplot(x = 'Age Group',y= 'Amount' ,data = sales_age)
```



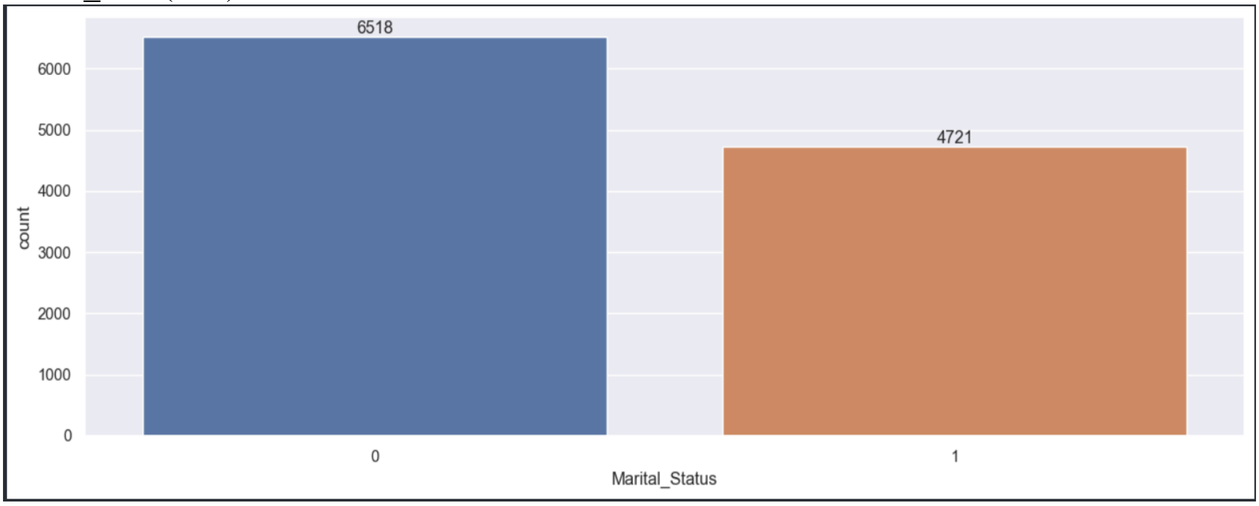
STATE GRAPH PLOTING :

```
# total number of orders from top 10 states
sales_state = df.groupby(['State'], as_index=False)['Orders'].sum().sort_values(by='Orders', ascending=False).head(10)
sns.set(rc={'figure.figsize':(15,5)})
sns.barplot(data = sales_state, x = 'State',y= 'Orders')
```

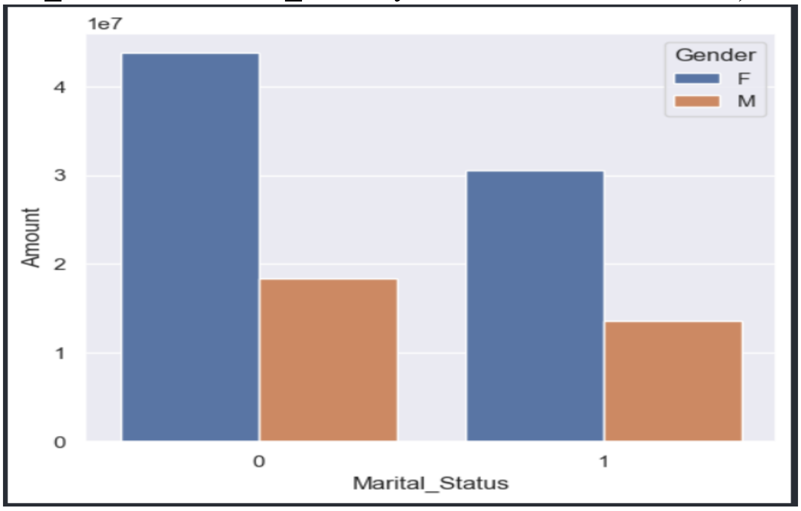


MARITAL STATUS GRAPH PLOTING :

```
ax = sns.countplot(data = df, x = 'Marital_Status')
sns.set(rc={'figure.figsize':(7,5)})
for bars in ax.containers:
    ax.bar_label(bars)
```

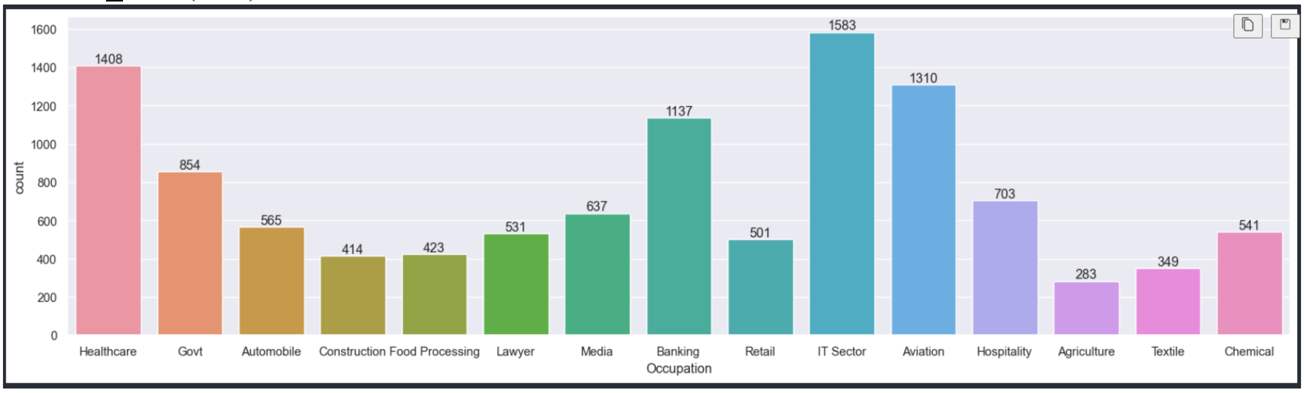


```
sales_state = df.groupby(['Marital_Status', 'Gender'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)
sns.set(rc={'figure.figsize':(6,5)})
sns.barplot(data = sales_state, x = 'Marital_Status',y= 'Amount', hue='Gender')
```

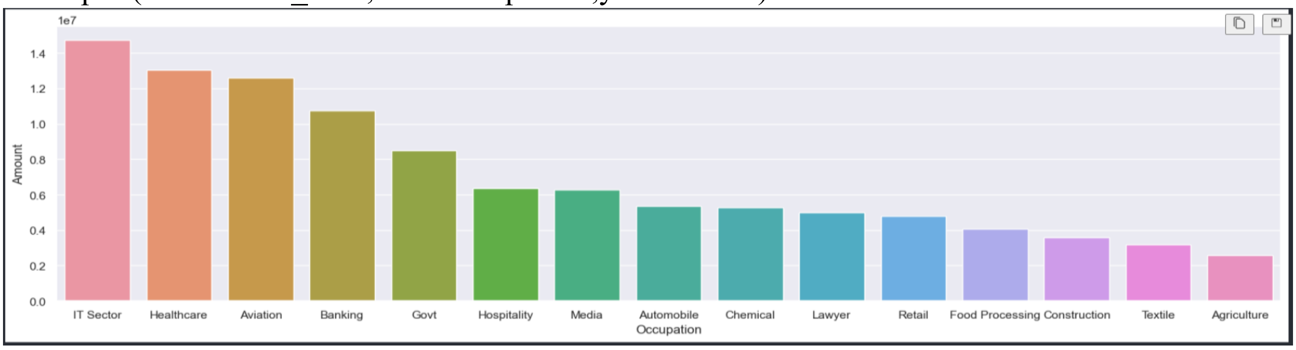


OCCUPATION GRAPH PLOTING :

```
sns.set(rc={'figure.figsize':(20,5)})
ax = sns.countplot(data = df, x = 'Occupation')
for bars in ax.containers:
    ax.bar_label(bars)
```

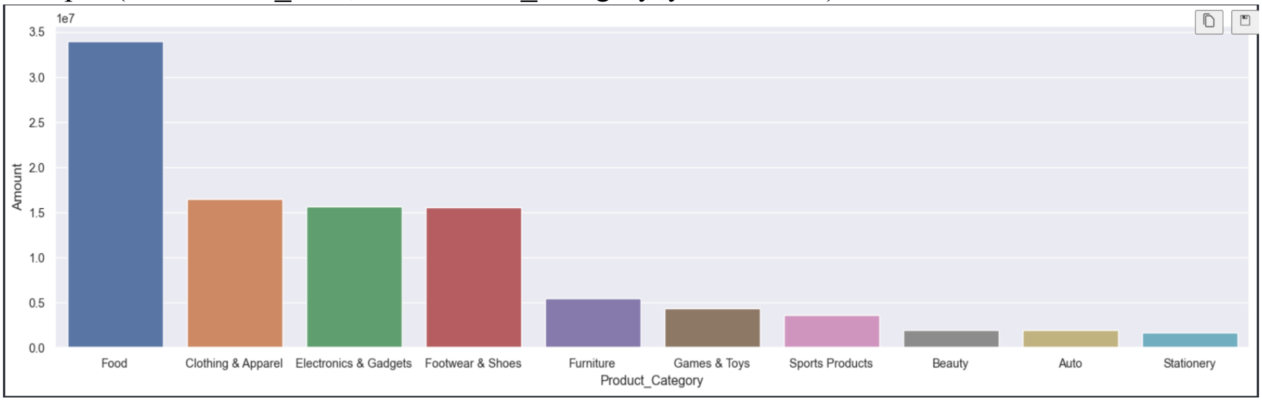



```
Sales_state = df.groupby(['Occupation'], as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)
sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Occupation',y= 'Amount')
```



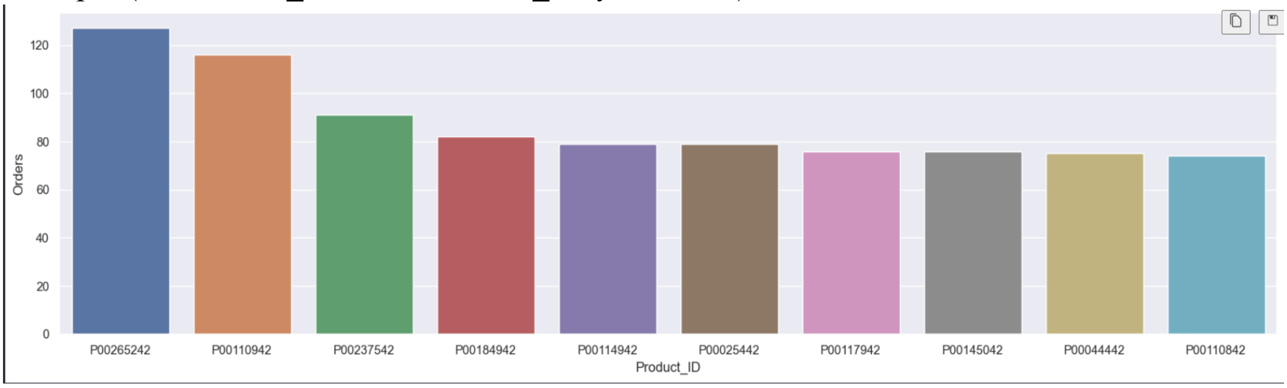
PRODUCT CATEGORY GRAPH PLOTING :

```
sales_state = df.groupby(['Product_Category'],
as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False).head(10)
sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Product_Category',y= 'Amount')
```



PRODUCT ID GRAPH PLOTING :

```
sales_state = df.groupby(['Product_ID'], as_index=False)['Orders'].sum().sort_values(by='Orders',
ascending=False).head(10)
sns.set(rc={'figure.figsize':(20,5)})
sns.barplot(data = sales_state, x = 'Product_ID',y= 'Orders')
```



PRACTICAL - 3

❖ **AIM :** IMPLEMENT LINEAR REGRESSION AN LOGISTIC REGRESSION.

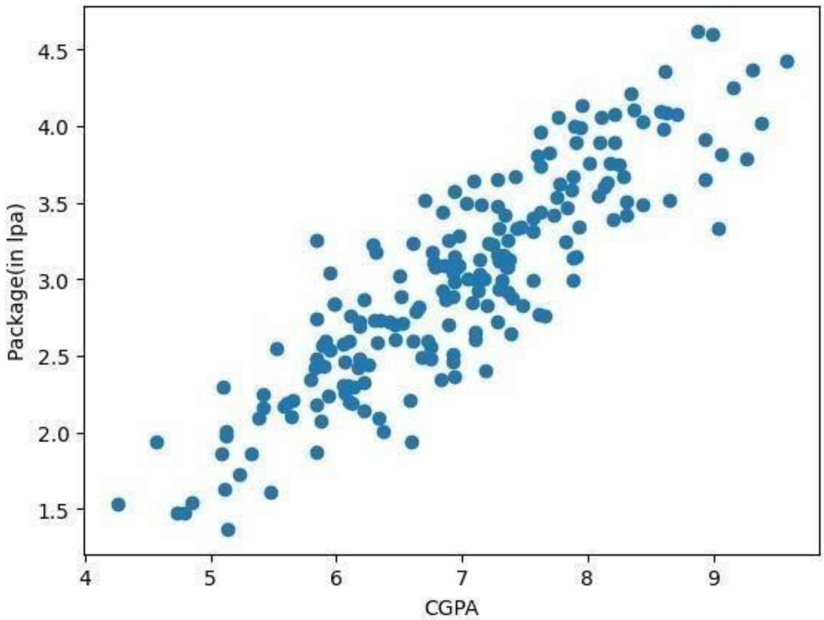
❖ **PROCEDURE :**

- **LINEAR REGRESSION :**
  1. Import Libraries :
     - Import necessary libraries such as NumPy, pandas, and scikit-learn.
  2. Prepare the Data :
     - Load your dataset and split it into features (X) and target variable (y).
     - Optionally, split the data into training and testing sets.
  3. Create the Model :
     - Instantiate the linear regression model.
  4. Fit the Model :
     - Train the model using the training data.
  5. Make Predictions :
     - Use the model to make predictions on the test set.
  6. Evaluate the Model :
     - Assess the model's performance using metrics like Mean Squared Error (MSE) or R^2 score.
  7. Visualize Results (Optional) :
     - Plot the regression line against the data points.

- **LOGISTIC REGRESSION :**
  1. Import Libraries:
     - Import necessary libraries such as NumPy, pandas, and scikit-learn.
  2. Prepare the Data:
     - Load your dataset and split it into features (X) and target variable (y).
     - Convert the target variable to binary if necessary.
     - Optionally, split the data into training and testing sets.
  3. Create the Model:
     - Instantiate the logistic regression model.
  4. Fit the Model:
     - Train the model using the training data.
  5. Make Predictions:
     - Use the model to make predictions on the test set.
  6. Evaluate the Model:
     - Assess the model's performance using metrics like accuracy, confusion matrix, or ROC-AUC score.
  7. Visualize Results (Optional) :
     - Plot the decision boundary or ROC curve.

❖ **PROGRAM** :

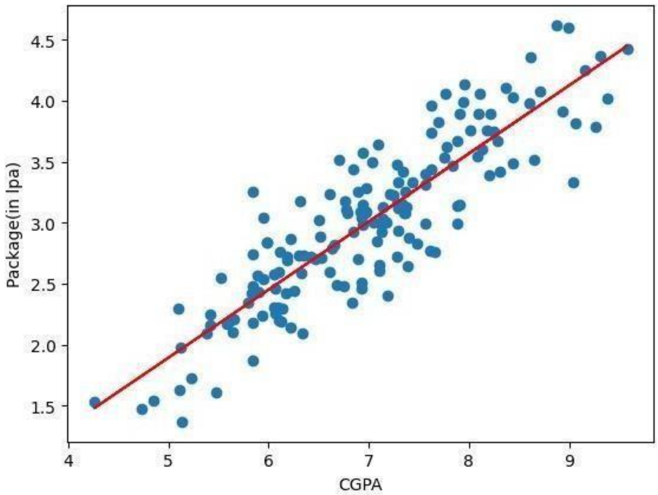
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('placement.csv')
df.head()
plt.scatter(df['cgpa'],df['package'])
plt.xlabel('CGPA')
plt.ylabel('Package(in lpa)')
```



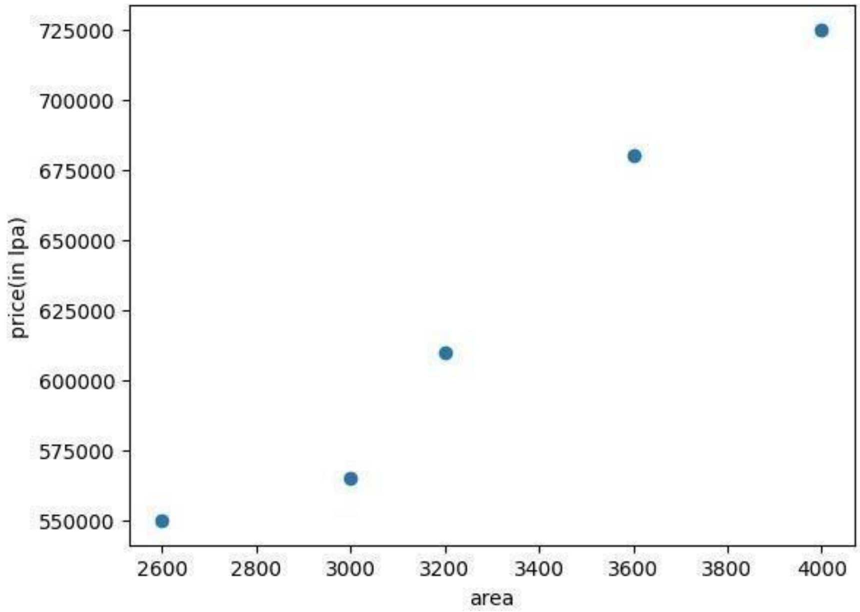

```
x=df.iloc[:,0:1]
y=df.iloc[:,-
1] y
x
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,
random_state=2)
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```


```
X_test
```

```
y_test
lr.predict(x_test.iloc[0].values.reshape(1,1))
plt.scatter(x_train,y_train)
plt.plot(x_train,lr.predict(x_train),color='red') plt.xlabel('CGPA')
plt.ylabel('Package(in lpa)')
```



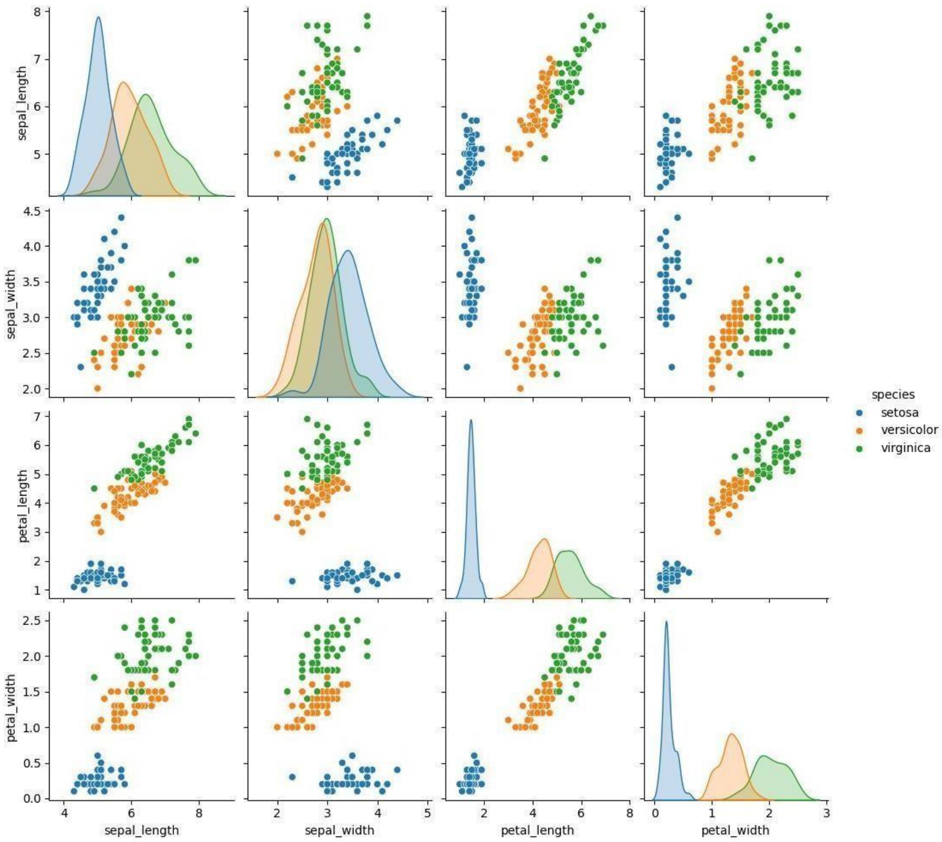

```
m=lr.coef_ a=[1,2,3,4,5,6,7,10,12,13,14,15,16,17,18,19,20,21,22,25,26,27,28,
29,30,31,32,34,35,36,37,38,39,40,42,44,45,46,47,48,49,50,51,52,53,
54,55,56,58,59,60,61,63,64,65,67,68,72,74]
print(len(a))
b=lr.intercept
_ m * 8.58+b
m*9.5+b
m*100+b
from sklearn.metrics import r2_score, mean_squared_error ,
mean_absolute_error
y_pred=lr.predict(x_test)
 score = r2_score(y_test,y_pred)
print(f'Accuracy Score : {score}')
y_pred
score1=lr.score(x_test,y_test)
print(f'Accracy Score:{score1}')
MSE = mean_squared_error(y_test,y_pred) MSE
MAE = mean_absolute_error(y_test,y_pred) MAE
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('homeprices.csv')
df.head()
plt.scatter(df['area'],df['price'])
plt.xlabel('area')  plt.ylabel('price(in
lpa)')
```




```
x=df.iloc[:,0:1]
y=df.iloc[:,-
1] x
y
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=2)

from sklearn.linear_model import LinearRegression
lrr=LinearRegression
```

Logistic Linear Regression:-

```
import seaborn as sns
import pandas as pd
import numpy as np

df=sns.load_dataset('iris')
df.head()
df['species'].unique()
df.isnull().sum()

##EDA
sns.pairplot(df,hue='species')
```



df=df[df['species']!='setosa'] df.head()
df['species']=df['species'].map({'versicolor':0,'virginica':1}) df.head()
x=df.iloc[:,:-1]
y=df.iloc[:,-
1] x
y
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.25, random_state=42)
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
classifier.fit(X_train,y_train)

```
▾ LogisticRegression
LogisticRegression()
```

from sklearn.model_selection import GridSearchCV parameters =
{'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,40,50],
'max_iter':[100,200,300]}
classifier_regressor = GridSearchCV(classifier, param_grid=parameters,
scoring='accuracy', cv=5)
classifier_regressor.fit(X_train,y_train)

```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:528: FitFailedWarning:
330 fits failed out of a total of 495.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
165 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1389, in wrapper
    return fit_method(estimator, *args, **kwargs)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1193, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 63, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or None penalties, got l1 penalty.

--------------------------------------------------------------------------------
165 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/base.py", line 1389, in wrapper
...
        nan 0.96            nan        nan 0.96            nan
        nan 0.96            nan        nan 0.96            nan
        nan 0.96            nan]
  warnings.warn(
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

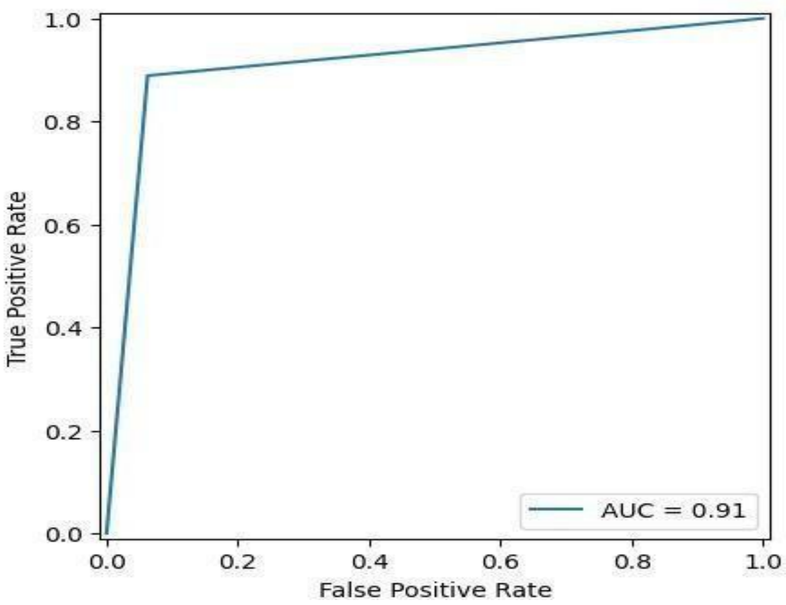
print(classification_report(y_test,y_pred))

```
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        16
           1       0.89      0.89      0.89         9

    accuracy                           0.92        25
   macro avg       0.91      0.91      0.91        25
weighted avg       0.92      0.92      0.92        25
```

from sklearn.metrics import confusion_matrix cm =
confusion_matrix(y_test,y_pred)
print(cm)

```
[[15  1]
 [ 1  8]]
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score, RocCurveDisplay fpr, tpr,
thresholds = roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)

RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=auc_score).plot()

plt.show()
```



PRACTICAL - 4

❖ **AIM :** IMPLEMENT THE NAIVE BAYESIAN CLASSIFIER FOR A SAMPLE TRAINING DATA SET STORED AS A (.CSV) FILE. COMPUTE THE ACCURACY OF THE CLASSIFIER, CONSIDERING A FEW TEST DATA SETS.

❖ PROCEDURE :

To implement a naïve Bayesian classifier using a sample training dataset in CSV format, follow these general steps :

1. Load the CSV data into your programming environment.
2. Preprocess the data (handle missing values, encode categorical variables).
3. Split the data into training and testing sets.
4. Train the Naïve Bayes model on the training data.
5. Make predictions on the test data.
6. Compute the accuracy by comparing predicted labels with actual labels.

▪ PREPROCESS THE DATA :

- Encode categorical variables using **LabelEncoder** or one-hot encoding.
- Convert all attributes to floating-point numbers for numerical processing.

▪ SPLIT THE DATA :

- Divide the dataset into training and testing sets using a specified ratio (e.g., 70% training, 30% testing).

▪ TRAIN THE NAÏVE BAYES MODEL :

- Calculate the mean and standard deviation for each attribute in each class from the training data.

▪ MAKE PREDICTIONS :

- Use the trained model to predict the class labels for the test data.
- Load and preprocess the dataset.
- Split the data into training and testing sets.
- Train the Naïve Bayes model and make predictions.
- Calculate and display the accuracy of the model.

❖ PROGRAM :

```
import numpy as np import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split from sklearn.naive_bayes import GaussianNB

np.set_printoptions(suppress=True, precision=6)

df = pd.read_csv("titanic-data.csv")

df.head()
```

	passenger_id	name	p_class	gender	age	sib_sp	parch	ticket	fare	cabin	embarked	survived
0	1	Braund, Mr. Owen Harris	3	male	22.0	1	0	A/5 21171	7.2500	NaN	S	0
1	2	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	female	38.0	1	0	PC 17599	71.2833	C85	C	1
2	3	Heikkinen, Miss. Laina	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S	1
3	4	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	female	35.0	1	0	113803	53.1000	C123	S	1
4	5	Allen, Mr. William Henry	3	male	35.0	0	0	373450	8.0500	NaN	S	0

```
df.drop(["passenger_id", "name", "sib_sp", "parch", "ticket", "cabin", "embarked"], axis=1,
inplace=True)

df.head()
```

	p_class	gender	age	fare	survived
0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

```
target = df["survived"]

inputs = df.drop("survived", axis=1)

display(target.head())

display(inputs.head())
```

	survived
0	0
1	1
2	1
3	1
4	0

dtype: int64

	p_class	gender	age	fare
0	3	male	22.0	7.2500
1	1	female	38.0	71.2833
2	3	female	26.0	7.9250
3	1	female	35.0	53.1000
4	3	male	35.0	8.0500

```
dummies = pd.get_dummies(inputs["gender"])

display(dummies.head())
print(dummies.dtypes)
```

	female	male
0	False	True
1	True	False
2	True	False
3	True	False
4	False	True

```
female    bool
male      bool
dtype: object
```

```
inputs = pd.concat([inputs, dummies], axis=1)

inputs.head()
```

	p_class	gender	age	fare	survived
0	3	male	22.0	7.2500	0
1	1	female	38.0	71.2833	1
2	3	female	26.0	7.9250	1
3	1	female	35.0	53.1000	1
4	3	male	35.0	8.0500	0

```
inputs.drop(["gender"], axis=1, inplace=True)

inputs.head()
```

	p_class	age	fare	female	male
0	3	22.0	7.2500	False	True
1	1	38.0	71.2833	True	False
2	3	26.0	7.9250	True	False
3	1	35.0	53.1000	True	False
4	3	35.0	8.0500	False	True

inputs.age[:10]

	age
0	22.0
1	38.0
2	26.0
3	35.0
4	35.0
5	NaN
6	54.0
7	2.0
8	27.0
9	14.0

dtype: float64

inputs["age"] = inputs["age"].fillna(inputs["age"].mean())

inputs.age[:10]

	age
0	22.000000
1	38.000000
2	26.000000
3	35.000000
4	35.000000
5	29.699118
6	54.000000
7	2.000000
8	27.000000
9	14.000000

dtype: float64

X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.2) print(len(X_train), len(X_test), len(inputs))

print(len(X_train) / len(inputs))

print(len(X_test) / len(inputs))

```
712 179 891
0.7991021324354658
0.20089786756453423
```

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

```
▾ GaussianNB
GaussianNB()
```

```
model.score(X_test, y_test)
```

```
0.7988826815642458
```

```
pred = np.array(model.predict(X_test))

pred_probability = np.array(model.predict_proba(X_test))

print(pred[:5])

for i in range(1, 6):
    print(pred_probability[i][0],end=", ")
```

```
[0 0 1 0 1]
0.9895202768716742, 0.003675565181087268, 0.9908563111883015, 0.036794007058115134, 0.9787398295999188,
```

PRACTICAL - 5

❖ **AIM :** ASSUMING A SET OF DOCUMENTS THAT NEED TO BE CLASSIFIED, USE THE NAIVE BAYESIAN CLASSIFIER MODEL TO PERFORM THIS TASK.

❖ PROCEDURE :

- **DATA COLLECTION :**
  - Gather Documents : Collect a labeled dataset of documents. Each document should be associated with a class label (e.g., categories like "sports," "politics," "technology," etc.).

- **PREPROCESSING :**
  - Text Cleaning : Remove any irrelevant information, such as HTML tags, special characters, and numbers.
  - Tokenization : Split the text into individual words or tokens.
  - Lowercasing : Convert all text to lowercase to ensure uniformity.
  - Stop Word Removal : Remove common words (e.g., "and," "the," "is") that do not contribute to the meaning.
  - Stemming/Lemmatization : Reduce words to their base or root form (e.g., "running" to "run").

- **FEATURE EXTRACTION :**
  - Create a Vocabulary : Build a list of unique words (features) from the training documents.
  - Vectorization : Convert documents into numerical format using techniques like :
    - Bag of Words (BoW) : Count the frequency of each word in the document.
    - Term Frequency-Inverse Document Frequency (TF-IDF) : Weigh the frequency of words based on their importance across all documents.

- **SPLITTING THE DATASET :**
  - Train-Test Split : Divide the dataset into a training set and a test set (e.g., 80% training, 20% testing).

❖ PROGRAM :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer from
sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report

msg=pd.read_csv('document.csv', names=['message','label'])
print("Total Instances of dataset")
msg['labelnum']=msg['label'].map({'pos':1,'neg':0})
X=msg['message']
y=msg['label'] print(msg)
```

```
Total Instances of dataset
                                    message label  labelnum
0                     I love this sandwich   pos          1
1                 This is an amazing place   pos          1
2       I feel very good about these beers   pos          1
3                     This is my best work   pos          1
4                     What an awesome view   pos          1
5            I do not like this restaurant   neg          0
6                 I am tired of this stuff   neg          0
7                   I can't deal with this   neg          0
8                     He is my sworn enemy   neg          0
9                      My boss is horrible   neg          0
10                This is an awesome place   pos          1
11   I do not like the taste of this juice   neg          0
12                         I love to dance   pos          1
13       I am sick and tired of this place   neg          0
14                    What a great holiday   pos          1
15          That is a bad locality to stay   neg          0
16          We will have good fun tomorrow   pos          1
17        I went to my enemy's house today   neg          0
```

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
vectorizer = CountVectorizer()
 Xtrain_dm = vectorizer.fit_transform(Xtrain)
 Xtest_dm = vectorizer.transform(Xtest)
Xtrain_dm
```

```
<13x39 sparse matrix of type '<class 'numpy.int64'>'
	with 65 stored elements in Compressed Sparse Row format>
```

```
clf=MultinomialNB()
clf.fit(Xtrain_dm, ytrain)
```

```
▾ MultinomialNB
MultinomialNB()
```

from sklearn.metrics import accuracy_score, confusion_matrix pred =
clf.predict(Xtest_dm)
print('Accuracy: ', accuracy_score(ytest, pred)) print('Confusion
Matrix:\n', confusion_matrix(ytest, pred))

```
Accuracy:  0.6
Confusion Matrix:
 [[1 2]
 [0 2]]
```

pred

```
array(['pos', 'pos', 'pos', 'pos', 'neg'], dtype='<U3')
```

user_input = input("Enter a message to predict its sentiment: ") user_input_dm =
vectorizer.transform([user_input])
user_pred = clf.predict(user_input_dm) sentiment =
'pos' if user_pred[0] == 1 else 'neg'
print(f"The sentiment of your message is: {sentiment}")

```
Enter a message to predict its sentiment: men don't celebrate valentine day
The sentiment of your message is: neg
```

PRACTICAL - 6

❖ **AIM :** DECISION TREE BASED ID3 ALGORITHM.

❖ PROCEDURE :

- Start with the Entire Dataset :Begin with the entire dataset as the root node of the tree.
- Check for Stopping Conditions :
    - If all instances in the dataset belong to the same class, create a leaf node with that class label.
    - If there are no remaining attributes to split on, create a leaf node with the majority class label of the instances in the dataset.
- Calculate Information Gain :
  For each attribute in the dataset, calculate the Information Gain (IG) based on the entropy of the dataset before and after the split.
- Entropy is calculated using the formula :
  [\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \log_2(p_i)] where (p_i) is the proportion of instances belonging to class (i) in the dataset (S), and (c) is the number of classes.
- Information Gain is calculated as :
  [IG(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)] where (S_v) is the subset of (S) for which attribute (A) has value (v).
- Select the Best Attribute :
  Choose the attribute with the highest Information Gain to split the dataset. This attribute will become the decision node in the tree.
- Create Subsets :
  Split the dataset into subsets based on the values of the selected attribute.
- Recursively Build the Tree :
    - For each subset, repeat steps 2 to 5 :
    - Use the subset as the new dataset.
    - Remove the selected attribute from consideration for further splits.
    - Continue until stopping conditions are met.
- Construct the Tree :
  As the recursion unwinds, construct the decision tree by linking nodes and leaf nodes based on the splits made.
- Output the Decision Tree :
  Once the recursion is complete, the final decision tree can be outputted, which can then be used for classification of new instances.

❖ PROGRAM :

STEP 1 : IMPORT NECESSARY LIBRARIES

```
import math

import pandas as pd

from graphviz import Digraph

from IPython.display import Image
```

STEP 2 : FUNCTION TO CALCULATE ENTROPY

```
def calculate_entropy(dataset):
    class_counts = dataset.iloc[:, -1].value_counts()
    total_instances = len(dataset)
    entropy = 0
    for count in class_counts:
        probability = count / total_instances
        entropy -= probability * math.log2(probability)
    return entropy
```

STEP 3 : FUNCTION TO CALCULATE INFORMATION GAIN

```
def calculate_information_gain(dataset, attribute):
    total_entropy = calculate_entropy(dataset)
    attribute_values = dataset[attribute].unique()
    weighted_entropy = 0
    for value in attribute_values:
        subset = dataset[dataset[attribute] == value]
        weighted_entropy += (len(subset) / len(dataset)) * calculate_entropy(subset)
    return total_entropy - weighted_entropy
```

STEP 4 : FUNCTION TO BUILD THE DECISION TREE

```
def build_decision_tree(dataset, attributes, parent_node=None, graph=None): # Build the decision tree
    if graph is None:
        graph = Digraph('DecisionTree')
    if len(dataset.iloc[:, -1].unique()) == 1: # Make Attributes of in image
        leaf = dataset.iloc[0, -1]
        graph.node(parent_node, label=leaf, shape='ellipse', style='filled', color='lightblue')
```

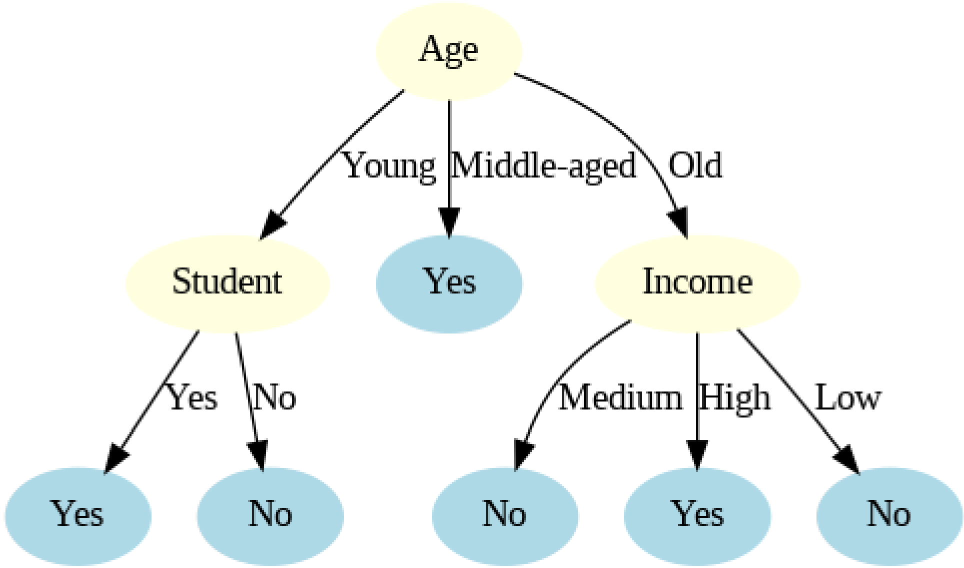
```
        return leaf
    if len(attributes) == 0:
        majority_class = dataset.iloc[:, -1].mode()[0]
        graph.node(parent_node, label=majority_class, shape='ellipse', style='filled', color='lightgreen')
        return majority_class
    best_attribute = max(attributes, key=lambda attr: calculate_information_gain(dataset, attr))
    graph.node(parent_node, label=best_attribute, shape='ellipse', style='filled', color='lightyellow')# This Main Entity
    attribute_values = dataset[best_attribute].unique()
    for value in attribute_values:
        subset = dataset[dataset[best_attribute] == value].drop(columns=[best_attribute])
        child_node = f'{parent_node}_{value}'
        graph.edge(parent_node, child_node, label=value)
        build_decision_tree(subset, [attr for attr in attributes if attr != best_attribute], child_node, graph)
    return graph
```

STEP 5 : EXAMPLE USAGE

```
data = {
    'Age': ['Young', 'Young', 'Middle-aged', 'Old', 'Old', 'Old'],
    'Income': ['High', 'High', 'Low', 'Medium', 'High', 'Low'],
    'Student': ['Yes', 'No', 'No', 'Yes', 'Yes', 'No'],
    'Buy Computer': ['Yes', 'No', 'Yes', 'No', 'Yes', 'No']
}
dataset = pd.DataFrame(data)
attributes = ['Age', 'Income', 'Student']
```

STEP 6 : BUILD AND VISUALIZE THE DECISION TREE

```
graph = build_decision_tree(dataset, attributes, 'Root')
graph.render('decision_tree', format='png')
Image(filename='decision_tree.png')
```


Age
Young
Middle-aged
Old
Student
Yes
Income
Yes
No
Medium
High
Low
Yes
No
No
Yes
No

PRACTICAL - 7

❖ **AIM :** WRITE A PROGRAM TO IMPLEMENT THE K-NEAREST NEIGHBOUR ALGORITHM TO CLASSIFY THE IRIS DATA SET.

❖ **PROCEDURE :**

▪ **IMPLEMENT K-NEAREST NEIGHBORS (KNN) FOR THE IRIS DATASET :**

1. Import Required Libraries
   Import necessary Python libraries such as numpy, pandas, matplotlib, seaborn, and sklearn.
2. Load the Iris Dataset
   Use sklearn.datasets.load_iris() to load the dataset and convert it into a DataFrame.
3. Preprocess the Data
   - Split the dataset into features (X) and target labels (y).
   - Normalize or standardize data if necessary.
   - Split the data into training and testing sets using train_test_split().
4. Implement K-Nearest Neighbors (KNN) Algorithm
   - Use KNeighborsClassifier from sklearn.neighbors.
   - Choose an appropriate value for K (e.g., K=5).
   - Fit the model with training data.
5. Make Predictions & Evaluate Model
   - Predict the class labels for test data.
   - Compute accuracy using accuracy_score().
   - Display confusion matrix and classification report.

❖ PROGRAM :

Step 1: IMPORTING NECESSARY LIBRARIES

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
StandardScaler from sklearn.neighbors import
KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Step 2: LOAD THE IRIS DATASET

```
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Labels (species)
```

Step 3: DATA PREPROCESSING

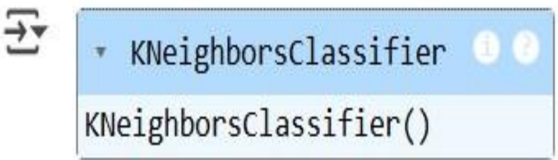
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Step 4: SPLIT THE DATA INTO TRAINING AND TESTING SETS (80% TRAIN, 20% TEST)

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

Step 5: TRAIN THE KNN MODEL

```
knn = KNeighborsClassifier(n_neighbors=5)  # You can change the number of neighbors
knn.fit(X_train, y_train)
```



Step 6: MAKE PREDICTIONS

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

Step 7: EVALUATE THE MODEL

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy * 100:.2f}%') Output: Accuracy : 100.00%

Step 8: CONFUSION MATRIX

print("\nConfusion Matrix:")

print(confusion_matrix(y_test, y_pred))

```
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

Step 9: CLASSIFICATION REPORT

print("\nClassification Report:")

print(classification_report(y_test, y_pred))

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

Step 10: VISUALIZE THE CONFUSION MATRIX

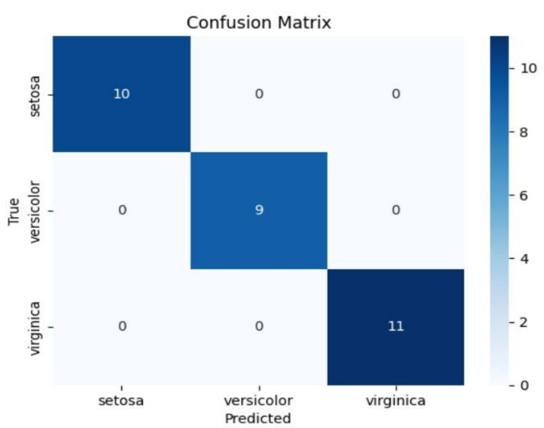
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names, yticklabels=iris.target_names) plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()



PRACTICAL - 8

❖ **AIM :** APPLY EM ALGORITHM TO CLUSTER A SET OF DATA STORED IN A (.CSV) FILE. USE THE SAME DATA SET FOR CLUSTERING USING K-MEANS ALGORITHM.

❖ **PROCEDURE :**
To apply the Expectation-Maximization (EM) Algorithm and K-Means Algorithm to cluster a given dataset stored in a .csv file and compare their results.

❖ **PROGRAM :**

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm.

Step 1: Import Necessary Libraries

```
# Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
```

Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm.

Step 1: Import Necessary Libraries

```
# Importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
```

Step 2: Load the Dataset from CSV

Make sure you upload your .CSV file to Google Colab before running the following code. If your file is named data.csv, you can load it as follows.

Load the dataset from a CSV file

file_path = '/content/data.csv' # Update with your file path

df = pd.read_csv(file_path)

Display the first few rows to understand the structure

df.head()

```
# Load the dataset from a CSV file
file_path = '/content/data.csv'  # Update with your file path
df = pd.read_csv(file_path)

# Display the first few rows to understand the structure
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Step 3: Data Preprocessing.

We will standardize the data (i.e., mean = 0, standard deviation = 1), as both K-Means and GMM benefit from scaling for better performance.

Dropping non-numeric columns if any (adjust according to your data)

df = df.select_dtypes(include=[np.number])

Standardize the data (important for K-Means and GMM)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(df)

Check the scaled data

print(X_scaled[:5]) # First 5 rows of scaled data

```
# Dropping non-numeric columns if any (adjust according to your data)
df = df.select_dtypes(include=[np.number])

# Standardize the data (important for K-Means and GMM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df)

# Check the scaled data
print(X_scaled[:5])  # First 5 rows of scaled data
```

```
[[-1.72054204 -0.90068117  1.03205722 -1.3412724  -1.31297673]
 [-1.69744751 -1.14301691 -0.1249576  -1.3412724  -1.31297673]
 [-1.67435299 -1.38535265  0.33784833 -1.39813811 -1.31297673]
 [-1.65125846 -1.50652052  0.10644536 -1.2844067  -1.31297673]
 [-1.62816394 -1.02184904  1.26346019 -1.3412724  -1.31297673]]
```

Step 4: Apply K-Means Clustering

Now, we'll use K-Means to cluster the data. First, we will try clustering for a given number of clusters, say k=3. You can adjust this based on your dataset.

Apply K-Means clustering

kmeans = KMeans(n_clusters=3, random_state=42)

kmeans.fit(X_scaled)

Assign labels to each point

kmeans_labels = kmeans.predict(X_scaled)

Evaluate the clustering with silhouette score

silhouette_kmeans = silhouette_score(X_scaled, kmeans_labels)

print(f"K-Means Silhouette Score: {silhouette_kmeans:.4f}")

Plot the clusters

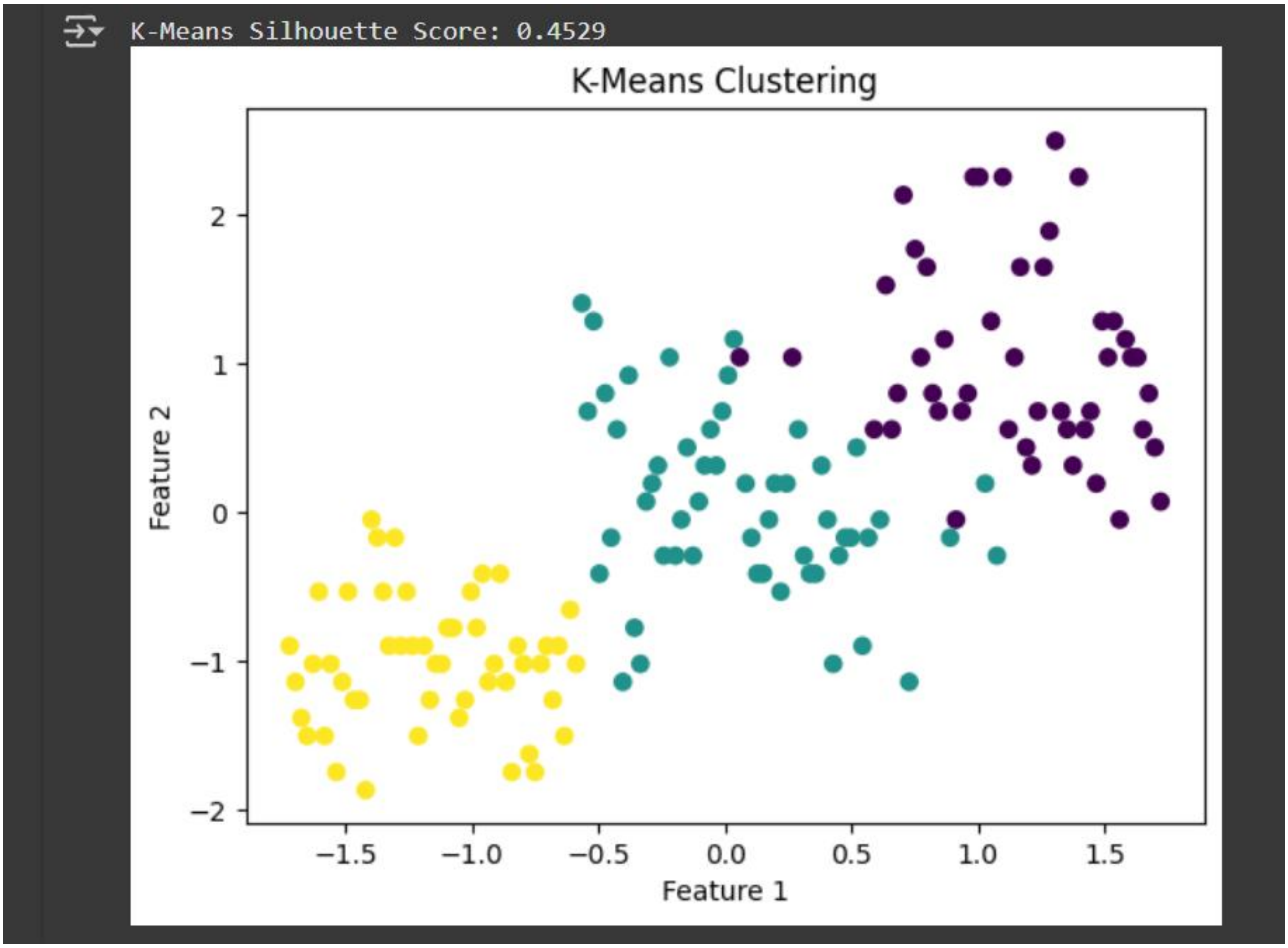
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=kmeans_labels, cmap='viridis')

plt.title('K-Means Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

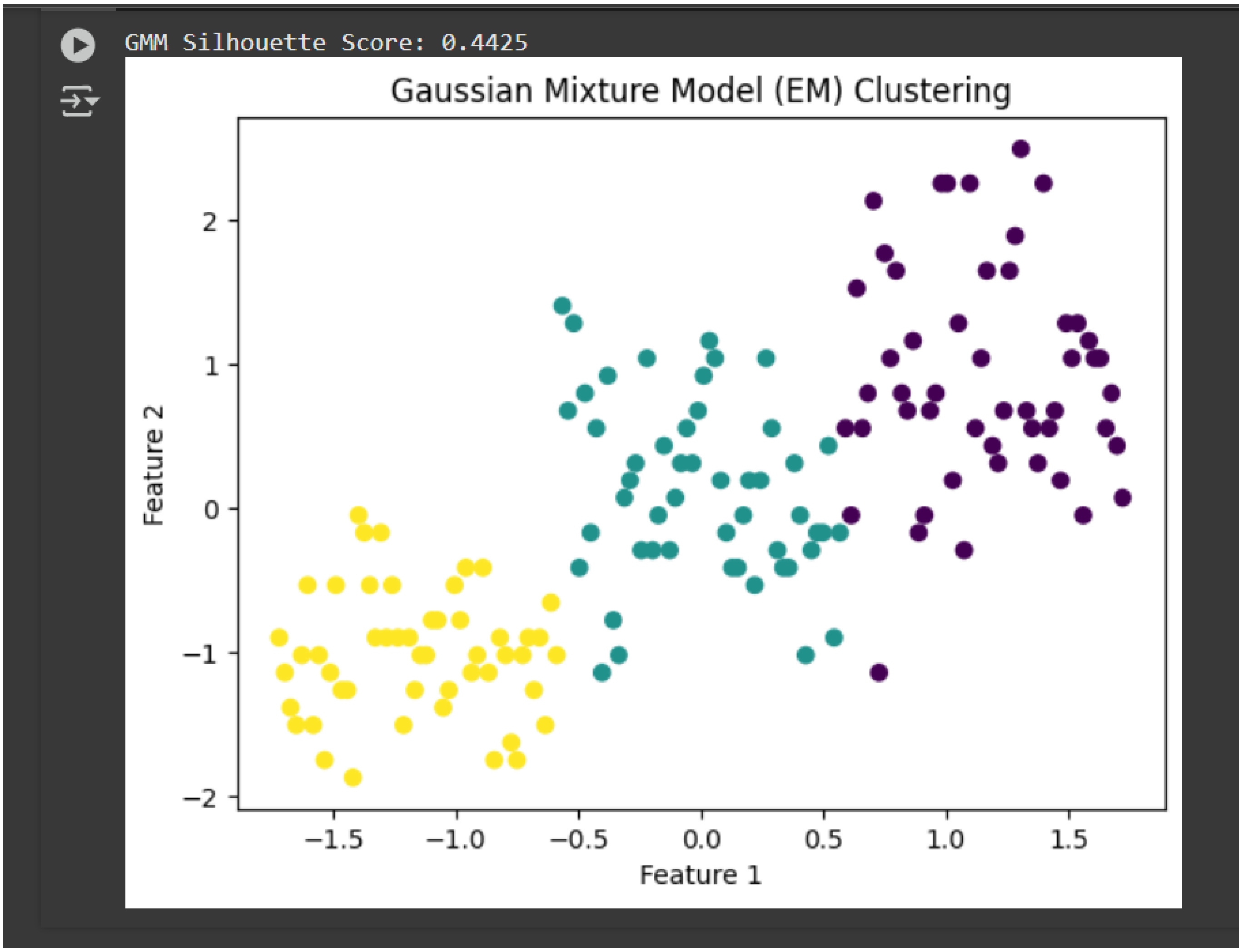


Step 5: Apply Expectation-Maximization (GMM)

The Expectation-Maximization algorithm is typically implemented using Gaussian Mixture Models in sklearn. Here, we will cluster the data using a GMM.

```
# Apply Gaussian Mixture Model (GMM) for clustering
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(X_scaled)
# Assign labels to each point
gmm_labels = gmm.predict(X_scaled)
# Evaluate the clustering with silhouette score
silhouette_gmm = silhouette_score(X_scaled, gmm_labels)
print(f"GMM Silhouette Score: {silhouette_gmm:.4f}")
# Plot the clusters
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=gmm_labels, cmap='viridis')
```

```
plt.title('Gaussian Mixture Model (EM) Clustering')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()
```



Step 6: Compare Results

Now, let's compare the clustering results visually and numerically using the silhouette scores for each method.

```
# Print comparison of silhouette scores

print(f"K-Means Silhouette Score: {silhouette_kmeans:.4f}")

print(f"GMM (EM) Silhouette Score: {silhouette_gmm:.4f}")
```

```
# Print comparison of silhouette scores
print(f"K-Means Silhouette Score: {silhouette_kmeans:.4f}")
print(f"GMM (EM) Silhouette Score: {silhouette_gmm:.4f}")
```

```
K-Means Silhouette Score: 0.4529
GMM (EM) Silhouette Score: 0.4425
```

PRACTICAL - 9

❖ **AIM** : WRITE A PROGRAM TO CONSTRUCT A BAYESIAN NETWORK CONSIDERING MEDICAL DATA. USE THIS MODEL TO DEMONSTRATE THE DIAGNOSIS OF HEART PATIENTS USING STANDARD HEART DISEASE DATA SET.

❖ **PROCEDURE** :

To construct a Bayesian Network using medical data and use this model to diagnose heart disease based on the Standard Heart Disease Dataset.

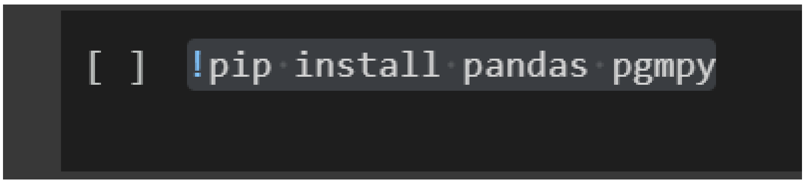
❖ **PROGRAM** :

Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set.

Step 1: Install Required Libraries

In a new Google Colab notebook, the first thing you'll need to do is install the necessary libraries (pandas and pgmpy). Add the following code in a cell:

!pip install pandas pgmpy



Step 2: Import Required Libraries

import pandas as pd

import matplotlib.pyplot as plt

import networkx as nx

from pgmpy.models import BayesianModel

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.inference import VariableElimination

Step 3: Upload the Dataset

To upload your dataset into Colab, run the following code to prompt the file upload:

from google.colab import files

uploaded = files.upload()

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

```
Choose Files No file chosen     Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving dataset.csv to dataset (1).csv
```

Step 4: Load the Data

Load the uploaded dataset into a pandas DataFrame:

data = pd.read_csv("/content/dataset.csv") # Ensure the path matches your file's name

heart_disease = pd.DataFrame(data)

```
data = pd.read_csv("/content/dataset.csv")  # Ensure the path matches your file's name
heart_disease = pd.DataFrame(data)
```

Step 5: Define and Fit the Bayesian Network

Define the structure of the Bayesian Network and fit it with the data using MaximumLikelihoodEstimator:

```
# Define the Bayesian Network structure
model = BayesianModel([
    ('age', 'Lifestyle'),
    ('Gender', 'Lifestyle'),
    ('Family', 'heartdisease'),
    ('diet', 'cholestrol'),
    ('Lifestyle', 'diet'),
    ('cholestrol', 'heartdisease'),
    ('diet', 'cholestrol')
])
# Fit the model using MaximumLikelihoodEstimator
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
```

```
# Define the Bayesian Network structure
model = BayesianModel([
    ('age', 'Lifestyle'),
    ('Gender', 'Lifestyle'),
    ('Family', 'heartdisease'),
    ('diet', 'cholestrol'),
    ('Lifestyle', 'diet'),
    ('cholestrol', 'heartdisease'),
    ('diet', 'cholestrol')
])

# Fit the model using MaximumLikelihoodEstimator
model.fit(heart_disease, estimator=MaximumLikelihoodEstimator)
```

```
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
```

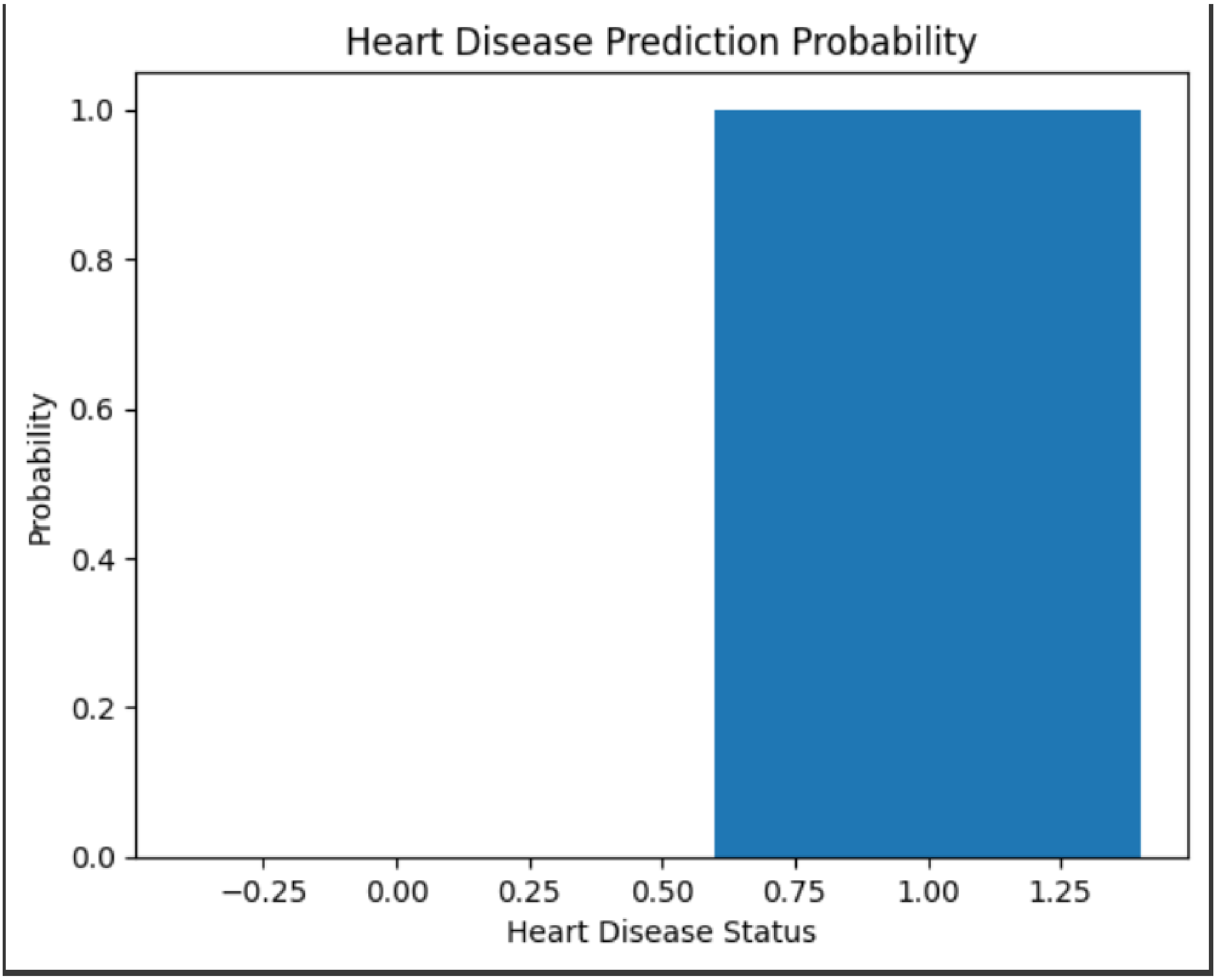
Step 6: Perform Inference and Predict Heart Disease

Take user input for the various factors, perform inference, and display the prediction:

```
HeartDisease_infer = VariableElimination(model)

# Instructions for user input

print('For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }')

print('For Gender Enter { Male:0, Female:1 }')

print('For Family History Enter { yes:1, No:0 }')

print('For diet Enter { High:0, Medium:1 }')

print('For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }')

print('For cholesterol Enter { High:0, BorderLine:1, Normal:2 }')

try:

    age = int(input('Enter age (0: SuperSeniorCitizen, 1: SeniorCitizen, 2: MiddleAged, 3: Youth, 4: Teen): '))

    gender = int(input('Enter Gender (0: Male, 1: Female): '))

    family_history = int(input('Enter Family History (1: Yes, 0: No): '))

    diet = int(input('Enter Diet (0: High, 1: Medium): '))

    lifestyle = int(input('Enter Lifestyle (0: Athlete, 1: Active, 2: Moderate, 3: Sedentary): '))

    cholestrol = int(input('Enter Cholestrol (0: High, 1: BorderLine, 2: Normal): '))

    # Query the model for heart disease prediction

    q = HeartDisease_infer.query(variables=['heartdisease'], evidence={

        'age': age,

        'Gender': gender,

        'Family': family_history,
```

```
        'diet': diet,
        'Lifestyle': lifestyle,
        'cholestrol': cholestrol
    })
    # Display the heart disease prediction
    print("Heart Disease Prediction:")
    for state, prob in zip(q.values, q.state_names['heartdisease']):
        print(f"{state}: {prob:.4f}")
    # Plot the inference results as a bar chart
    plt.bar(q.state_names['heartdisease'], q.values)
    plt.xlabel('Heart Disease Status')
    plt.ylabel('Probability')
    plt.title('Heart Disease Prediction Probability')
    plt.show()
except ValueError:
    print("Invalid input. Please enter valid integers for the options.")
```

```
For age Enter { SuperSeniorCitizen:0, SeniorCitizen:1, MiddleAged:2, Youth:3, Teen:4 }
For Gender Enter { Male:0, Female:1 }
For Family History Enter { yes:1, No:0 }
For diet Enter { High:0, Medium:1 }
For lifeStyle Enter { Athlete:0, Active:1, Moderate:2, Sedentary:3 }
For cholesterol Enter { High:0, BorderLine:1, Normal:2 }
Enter age (0: SuperSeniorCitizen, 1: SeniorCitizen, 2: MiddleAged, 3: Youth, 4: Teen): 4
Enter Gender (0: Male, 1: Female): 0
Enter Family History (1: Yes, 0: No): 1
Enter Diet (0: High, 1: Medium): 0
Enter Lifestyle (0: Athlete, 1: Active, 2: Moderate, 3: Sedentary): 1
Enter Cholestrol (0: High, 1: BorderLine, 2: Normal): 0
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
WARNING:pgmpy:BayesianModel has been renamed to BayesianNetwork. Please use BayesianNetwork class, BayesianModel will be removed in future.
Heart Disease Prediction:
0.0: 0.0000
1.0: 1.0000
```


Heart Disease Prediction Probability
Probability
1.0
0.8
0.6
0.4
0.2
0.0
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
Heart Disease Status

PRACTICAL - 10

❖ **AIM :** COMPARE THE VARIOUS SUPERVISED LEARNING ALGORITHM BY USING APPROPRIATE DATASET.(LINEAR REGRESSION, SUPPORT VECTOR MACHINE, DECISION TREE).

❖ PROCEDURE :

1. IMPORT REQUIRED LIBRARIES :
   - Load essential libraries such as NumPy, Pandas, Matplotlib, Seaborn, and Scikit- learn.
2. SELECT AN APPROPRIATE DATASET :
   - Choose a dataset suitable for regression (e.g., California Housing Dataset) or classification (e.g., Iris Dataset).
3. PREPROCESS THE DATA :
   - Handle missing values, normalize/standardize the data, and split it into training and testing sets (typically 80% train, 20% test).
4. TRAIN THE MODELS :
   - Train Linear Regression for regression tasks.
   - Train Support Vector Machine (SVM) using a linear kernel.
   - Train Decision Tree with an appropriate depth.
5. EVALUATE MODEL PERFORMANCE :
   - Use appropriate metrics like Mean Squared Error (MSE) for regression models and Accuracy Score for classification models.
6. COMPARE THE MODELS :
   - Visualize the results using bar plots to compare the errors or accuracies of each algorithm.
7. DRAW CONCLUSIONS :
   - Identify which algorithm performs best based on the dataset characteristics and chosen evaluation metrics.

❖ **PROGRAM :**

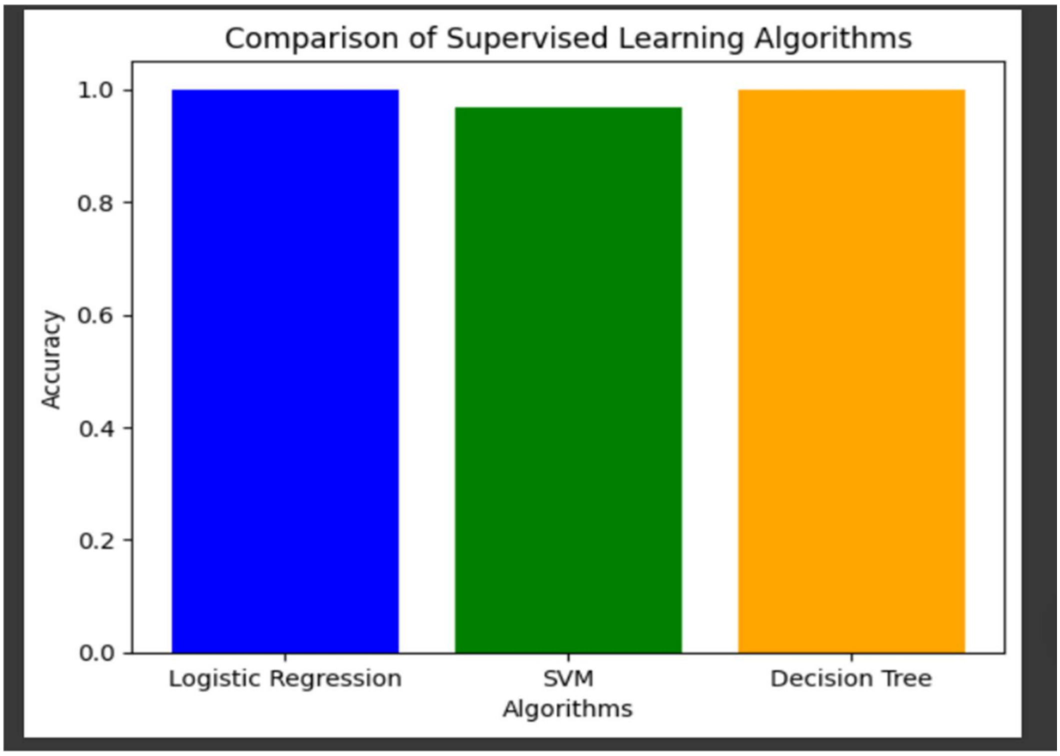
```
# Importing required libraries
import numpy as np
import pandas as pd
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the dataset
iris = datasets.load_iris()
X=pd.DataFrame(iris.data)
X.columns=['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']
y=pd.DataFrame(iris.target)
y.columns=['Targets']

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize the models (Logistic Regression, SVM, Decision Tree) lr =
LogisticRegression(max_iter=200)
svm = SVC(kernel='linear')  # Linear kernel for simplicity
dt = DecisionTreeClassifier(random_state=42)

# Train the models
lr.fit(X_train_scaled, y_train)
svm.fit(X_train_scaled, y_train)
dt.fit(X_train, y_train)  # Decision Tree is not sensitive to feature scaling
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:1408: DataC
  y = column_or_1d(y, warn=True)
        DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

```
#Predictions
y_pred_lr = lr.predict(X_test_scaled)
y_pred_svm = svm.predict(X_test_scaled)
y_pred_dt = dt.predict(X_test)

# Evaluate the models using accuracy score
acc_lr = accuracy_score(y_test, y_pred_lr)
acc_svm = accuracy_score(y_test, y_pred_svm)
acc_dt = accuracy_score(y_test, y_pred_dt)

# Print the accuracy scores for each model
print(f"Logistic Regression Accuracy: {acc_lr:.4f}")
print(f"Support Vector Machine Accuracy: {acc_svm:.4f}")
print(f"Decision Tree Accuracy: {acc_dt:.4f}")
```

```
Logistic Regression Accuracy: 1.0000
Support Vector Machine Accuracy: 0.9667
Decision Tree Accuracy: 1.0000
```

```
# Visualize the accuracy comparison
algorithms = ['Logistic Regression', 'SVM', 'Decision Tree']
accuracies = [acc_lr, acc_svm, acc_dt]

plt.bar(algorithms, accuracies, color=['blue', 'green', 'orange'])
plt.xlabel('Algorithms')
plt.ylabel('Accuracy')
plt.title('Comparison of Supervised Learning Algorithms') plt.show()
```



PRACTICAL - 11

❖ **AIM :** COMPARE THE VARIOUS UNSUPERVISED LEARNING ALGORITHM BY USING THE APPROPRIATE DATASETS.(K MEANS CLUSTERING, K MODE).

❖ **PROCEDURE :**

1.IMPORT REQUIRED LIBRARIES :
- Load essential libraries such as NumPy, Pandas, Matplotlib, Seaborn, and Scikit-learn.

2.SELECT AN APPROPRIATE DATASET :
- Choose a dataset suitable for clustering tasks, such as :
  a) K-Means Clustering: Works well with numerical datasets (e.g., Iris Dataset).
  b) K-Modes Clustering: Used for categorical data (e.g., Mushroom Dataset).

3.PREPROCESS THE DATA :
- Handle missing values, normalize numerical data, and encode categorical variables if necessary.

4.APPLY K-MEANS CLUSTERING :
- Determine the optimal number of clusters using the Elbow Method or Silhouette Score.
- Train the K-Means model and assign cluster labels to the dataset.

5.APPLY K-MODES CLUSTERING :
- Convert categorical data into an appropriate format.
- Train the K-Modes model and assign cluster labels.

6.EVALUATE THE CLUSTERING PERFORMANCE :
- Use metrics such as Inertia, Silhouette Score, or visualization techniques (e.g., scatter plots, cluster heatmaps) to analyze results.

7.COMPARE THE ALGORITHMS :
- Compare the clustering results in terms of accuracy, interpretability, and efficiency.

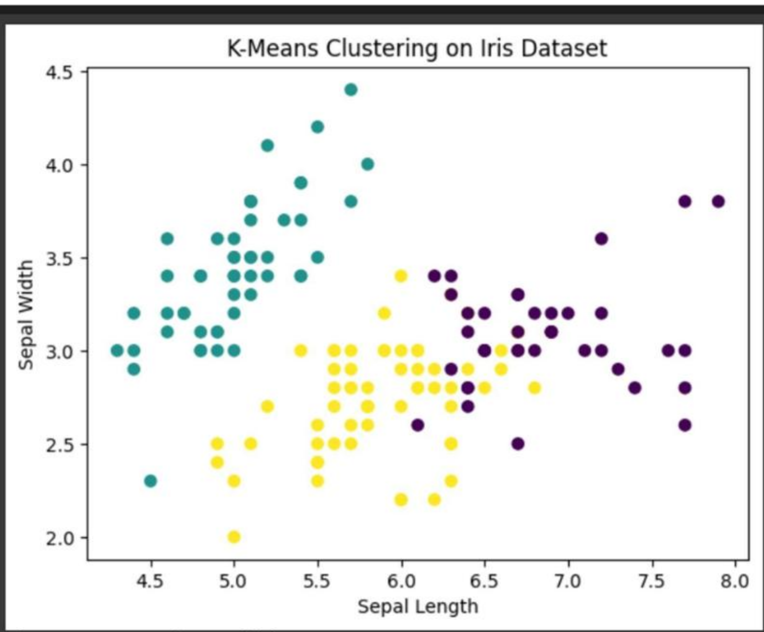
8.DRAW CONCLUSIONS :
- Identify which algorithm performs better based on dataset characteristics and clustering effectiveness.

❖ PROGRAM :

```
# Install the necessary libraries
!pip install scikit-learn matplotlib kmodes
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
from kmodes.kmodes import KModes
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Collecting kmodes
  Downloading kmodes-0.12.2-py2.py3-none-any.whl.metadata (8.1 kB)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplotlib) (1.17.0)
Downloading kmodes-0.12.2-py2.py3-none-any.whl (20 kB)
Installing collected packages: kmodes
Successfully installed kmodes-0.12.2
```

```
# Load the Iris dataset iris =
load_iris()
X = iris.data

# Features: Sepal length, Sepal width, Petal length, Petal width # Apply K-
Means clustering with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Get the cluster labels
labels = kmeans.labels_
# Visualize the K-Means clustering results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title('K-Means Clustering on Iris Dataset')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.show()

#Print the cluster centers (centroids)
print("Cluster centers (centroids):\n", kmeans.cluster_centers_)
```




```
Cluster centers (centroids):
 [[6.85384615 3.07692308 5.71538462 2.05384615]
 [5.006      3.428      1.462      0.246     ]
 [5.88360656 2.74098361 4.38852459 1.43442623]]
```

```
# Load Titanic dataset from a URL
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
data = pd.read_csv(url)

# Display the first few rows of the dataset to understand its structure
data.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
# Select categorical columns for clustering
categorical_data = data[['Sex', 'Embarked', 'Pclass']]

# Encode categorical data as numeric codes
categorical_data = categorical_data.apply(lambda col: col.astype('category').cat.codes)


# Apply K-Modes clustering with 3 clusters
kmodes = KModes(n_clusters=3, init='Huang', n_init=5, verbose=1)
clusters = kmodes.fit_predict(categorical_data)

# Add the cluster labels to the original dataset
```

```
data['Cluster'] = clusters
# Display the data with the assigned clusters
print(data[['Sex', 'Embarked', 'Pclass', 'Cluster']].head())

# Print the cluster centers (modes)
print("Cluster centers (modes):\n", kmodes.cluster_centroids_)
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 1, iteration: 1/100, moves: 96, cost: 509.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 2, iteration: 1/100, moves: 179, cost: 603.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 3, iteration: 1/100, moves: 0, cost: 575.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 4, iteration: 1/100, moves: 84, cost: 644.0
Init: initializing centroids
Init: initializing clusters
Starting iterations...
Run 5, iteration: 1/100, moves: 0, cost: 577.0
Best run was number 1
      Sex Embarked  Pclass  Cluster
0    male        S       3        0
1  female        C       1        1
2  female        S       3        2
3  female        S       1        1
4    male        S       3        0
Cluster centers (modes):
 [[1 2 2]
 [0 0 0]
 [0 2 2]]
```

PRACTICAL - 12

❖ **AIM :** BUILD AN ARTIFICIAL NEURAL NETWORK BY IMPLEMENTING THE BACKPROPAGATION ALGORITHM AND TEST THE SAME USING APPROPRIATE DATA SETS.

❖ **PROCEDURE :**

1. UNDERSTAND THE BACKPROPAGATION ALGORITHM
   - Forward propagation: Compute the output using initial weights.
   - Compute the loss: Measure the difference between predicted and actual output.
   - Backpropagation: Compute gradients using the chain rule.
   - Weight update: Adjust weights using gradient descent.
2. IMPORT REQUIRED LIBRARIES
   - Use **NumPy** for numerical computations.
   - Use **pandas** and **scikit-learn** for dataset handling.
3. DEFINE THE NEURAL NETWORK STRUCTURE
   - Input layer: Number of features in the dataset.
   - Hidden layer: Choose an appropriate number of neurons.
   - Output layer: Number of target classes.
   - Activation function: Use **sigmoid** or **ReLU** for hidden layers and **softmax** for multi-class classification.
4. INITIALIZE WEIGHTS AND BIASES
   - Use random values for weights and set biases to zero.
5. IMPLEMENT FORWARD PROPAGATION
   - Compute weighted sums and apply activation functions layer by layer.
6. COMPUTE THE LOSS FUNCTION
   - Use Mean Squared Error (MSE) for regression and Cross-Entropy loss for classification.
7. IMPLEMENT BACKPROPAGATION
   - Compute gradients of the loss function with respect to weights and biases.
   - Adjust weights using **Gradient Descent** or **Adam Optimizer**.
8. TRAIN THE NETWORK
   - Feed training data into the model.
   - Update weights iteratively over multiple epochs.

- Monitor loss reduction during training.

9. EVALUATE THE MODEL
   - Use a test dataset to check the model's accuracy.
   - Compare predicted outputs with actual values.
10. OPTIMIZE THE MODEL
    - Adjust learning rate, hidden layers, and activation functions for better performance.
11. TEST THE MODEL WITH A REAL DATASET
    - Use datasets like **Iris**, **MNIST**, or any tabular dataset for classification.

❖ PROGRAM :

```
# Step_1 Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


# Step_2 Part_1
# Define the Neural Network class with backpropagation
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01):
        # Initialize network parameters
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
# Step_2 Part_2: Initialize weights and biases with random values
        self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)
        self.bias_hidden = np.random.randn(1, self.hidden_size)
        self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)
        self.bias_output = np.random.randn(1, self.output_size)
```

```
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
    def sigmoid_derivative(self, x):
        return x * (1 - x)
# Step_2 Part_3
    def forward(self, X):
        # Forward pass: calculate hidden layer output and final output
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)
        self.final_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.final_output = self.sigmoid(self.final_input)
        return self.final_output
# Step_2 Part_4
    def backward(self, X, y):
        # Backpropagation: calculate gradients and update weights
        output_error = y - self.final_output
        output_delta = output_error * self.sigmoid_derivative(self.final_output)
        hidden_error = output_delta.dot(self.weights_hidden_output.T)
        hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_output)
# Update weights and biases using the gradients
        self.weights_input_hidden += X.T.dot(hidden_delta) * self.learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * self.learning_rate
        self.weights_hidden_output += self.hidden_output.T.dot(output_delta) * self.learning_rate
        self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * self.learning_rate
# Step_2 Part_5
    def train(self, X, y, epochs=1000):
# Training the network: run forward pass and backward pass for multiple epochs
        for epoch in range(epochs):
            self.forward(X)
            self.backward(X, y)
```

```
        if epoch % 100 == 0:
            loss = np.mean(np.square(y - self.final_output))
            print(f'Epoch {epoch}, Loss: {loss}')
# Step_2 Part_6
    def predict(self, X):
        # Make predictions with the trained network
        return self.forward(X)

# Step_3
# Generate a synthetic dataset for classification
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15, n_classes=2)
# Scale the data for better performance in training
scaler = StandardScaler()
X = scaler.fit_transform(X)
# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y.reshape(-1, 1), test_size=0.2, random_state=42)

# Step_4
# Initialize the neural network with input size, hidden size, and output size
nn = NeuralNetwork(input_size=20, hidden_size=10, output_size=1, learning_rate=0.01)
# Train the network with the training data (1000 epochs)
nn.train(X_train, y_train, epochs=1000)
```

```
Epoch 0, Loss: 0.2502172022855033
Epoch 100, Loss: 0.08991957284799572
Epoch 200, Loss: 0.06760875450455225
Epoch 300, Loss: 0.05107551543490075
Epoch 400, Loss: 0.042791161933662704
Epoch 500, Loss: 0.036625564655806596
Epoch 600, Loss: 0.03232724083004919
Epoch 700, Loss: 0.028755841917128835
Epoch 800, Loss: 0.02591051392948267
Epoch 900, Loss: 0.02342603045676408
```

```
#step5
# Test the trained model on the test set
predictions = nn.predict(X_test)
# Convert predictions to binary (0 or 1)
predictions = (predictions > 0.5).astype(int)
# Calculate accuracy: compare predicted values with actual values
accuracy = np.mean(predictions == y_test)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 87.00%
```

```
#step6
# Neural network class with loss tracking for visualization
class NeuralNetworkWithLoss(NeuralNetwork):
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01):
        super().__init__(input_size, hidden_size, output_size, learning_rate)
        self.loss_history = []
#step6 part2
def train(self, X, y, epochs=1000):
        for epoch in range(epochs):
            self.forward(X)
            self.backward(X, y)
            if epoch % 100 == 0:
                loss = np.mean(np.square(y - self.final_output))
                self.loss_history.append(loss)
                print(f'Epoch {epoch}, Loss: {loss}')
#step6 prac3
# Re-train using the modified class to store the loss
nn_with_loss = NeuralNetworkWithLoss(input_size=20, hidden_size=10, output_size=1,
learning_rate=0.01)
nn_with_loss.train(X_train, y_train, epochs=1000)
```

```
Epoch 0, Loss: 0.46936617477890524
Epoch 100, Loss: 0.12411160196289518
Epoch 200, Loss: 0.09914075921479235
Epoch 300, Loss: 0.07783312799646006
Epoch 400, Loss: 0.0595244840988669
Epoch 500, Loss: 0.046070768146957236
Epoch 600, Loss: 0.036989619528565096
Epoch 700, Loss: 0.03136187735015099
Epoch 800, Loss: 0.027356785929926294
Epoch 900, Loss: 0.024565783779983114
```

```
# Plot loss over epochs
plt.plot(nn_with_loss.loss_history)
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```