PROGRAM

Exp1.ipynb

```
import pandas as pd

df=pd.read_csv("100_students_dataset.csv")

df

df.head(2)

df.tail(5)
```

OUTPUT

[2]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Allison Hill	18	Male	EEE	54.7	4.84	Chennai	Yes	No
1	2	Noah Rhodes	17	Male	CSE	53.1	7.03	Chennai	Yes	No
2	3	Angie Henderson	20	Female	CIVIL	56.7	9.22	Pune	Yes	No
3	4	Daniel Wagner	22	Female	ECE	52.9	8.58	Chennai	Yes	No
4	5	Cristian Santos	18	Female	EEE	76.2	8.84	Hyderabad	No	Yes
...	...	...	...	...	...	...	...	...	...	...
95	96	Anna Henderson	24	Male	CSE	64.0	4.44	Hyderabad	Yes	Yes
96	97	Aaron Wise	19	Other	EEE	45.1	5.49	Kolkata	No	Yes
97	98	Deborah Figueroa	25	Female	MECH	94.5	5.78	Kolkata	No	No
98	99	Jessica Smith	17	Other	CSE	96.9	5.25	Bangalore	No	Yes
99	100	Stephen Mckee	19	Male	ECE	73.1	4.94	Delhi	No	No

100 rows × 10 columns

[3]:

```
df.head(2)
```

[3]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Allison Hill	18	Male	EEE	54.7	4.84	Chennai	Yes	No
1	2	Noah Rhodes	17	Male	CSE	53.1	7.03	Chennai	Yes	No

[4]:

```
df.tail(5)
```

[4]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
95	96	Anna Henderson	24	Male	CSE	64.0	4.44	Hyderabad	Yes	Yes
96	97	Aaron Wise	19	Other	EEE	45.1	5.49	Kolkata	No	Yes
97	98	Deborah Figueroa	25	Female	MECH	94.5	5.78	Kolkata	No	No
98	99	Jessica Smith	17	Other	CSE	96.9	5.25	Bangalore	No	Yes
99	100	Stephen Mckee	19	Male	ECE	73.1	4.94	Delhi	No	No

PROGRAM

Exp2.ipynb

```
import pandas as pd
df=pd.read_csv("student.csv")
df

df.describe()
df.info()
df.isnull().sum()
df.dtypes
df.shape

df1=df.fillna("n")
df1

df2=df.fillna(5)
df2

#Dictionary
df1=df.fillna({'chol':1,'fbs':2})
df1.isnull().sum()
df1

#Carry forward
df1=df.ffill()
df1

#Backward fill
df1=df.bfill()
df1
```

```
#Fill avg value
df1=df.interpolate()
df1

#Drop NA row/column
df1=df.dropna()
df1
df1
```

OUTPUT

[24]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Aarav Sharma	22.0	Male	CSE	85.2	8.6	Chennai	Yes	No
1	2	Ishita Verma	21.0	Female	ECE	72.5	7.2	Mumbai	No	Yes
2	3	Rahul Mehta	23.0	Male	MECH	65.1	6.8	Delhi	Yes	No
3	4	Priya Iyer	NaN	Female	CIVIL	92.0	9.1	Bangalore	No	Yes
4	5	Siddharth Jain	24.0	NaN	EEE	48.5	5.3	Kolkata	Yes	No
5	6	Kavya Reddy	20.0	Female	CSE	90.3	8.9	NaN	Yes	Yes
6	7	Aman Kumar	22.0	Male	ECE	55.0	NaN	Chennai	No	No
7	8	Meena S	19.0	Female	MECH	81.2	7.5	Mumbai	NaN	Yes
8	9	Varun Raj	25.0	Male	NaN	88.8	9.0	Hyderabad	Yes	NaN
9	10	Nisha Das	18.0	Female	CIVIL	NaN	6.7	Pune	No	No

[25]:
```
df.describe()
```

[25]:

	StudentID	Age	Attendance	CGPA
count	10.00000	9.000000	9.000000	9.000000
mean	5.50000	21.555556	75.400000	7.677778
std	3.02765	2.297341	16.051012	1.311276
min	1.00000	18.000000	48.500000	5.300000
25%	3.25000	20.000000	65.100000	6.800000
50%	5.50000	22.000000	81.200000	7.500000
75%	7.75000	23.000000	88.800000	8.900000
max	10.00000	25.000000	92.000000	9.100000

[26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   StudentID       10 non-null     int64
 1   Name            10 non-null     object
 2   Age             9 non-null      float64
 3   Gender          9 non-null      object
 4   Department      9 non-null      object
 5   Attendance      9 non-null      float64
 6   CGPA            9 non-null      float64
 7   City            9 non-null      object
 8   HostelResident  9 non-null      object
 9   Scholarship     9 non-null      object
dtypes: float64(3), int64(1), object(6)
memory usage: 932.0+ bytes
```

[27]:
```
df.isnull().sum()
```

[27]:
```
StudentID        0
Name             0
Age              1
Gender           1
Department       1
Attendance       1
CGPA             1
City             1
HostelResident   1
Scholarship      1
dtype: int64
```

[28]:
```
df.dtypes
```

[28]:
```
StudentID          int64
Name              object
Age              float64
Gender            object
Department        object
Attendance       float64
CGPA             float64
City              object
HostelResident    object
Scholarship       object
dtype: object
```

[29]:
```
df.shape
```

[29]:
```
(10, 10)
```

[33]:
```
#Dictionary
df1=df.fillna({'chol':1,'fbs':2})
df1.isnull().sum()
df1
```

[33]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Aarav Sharma	22.0	Male	CSE	85.2	8.6	Chennai	Yes	No
1	2	Ishita Verma	21.0	Female	ECE	72.5	7.2	Mumbai	No	Yes
2	3	Rahul Mehta	23.0	Male	MECH	65.1	6.8	Delhi	Yes	No
3	4	Priya Iyer	NaN	Female	CIVIL	92.0	9.1	Bangalore	No	Yes
4	5	Siddharth Jain	24.0	NaN	EEE	48.5	5.3	Kolkata	Yes	No
5	6	Kavya Reddy	20.0	Female	CSE	90.3	8.9	NaN	Yes	Yes
6	7	Aman Kumar	22.0	Male	ECE	55.0	NaN	Chennai	No	No
7	8	Meena S	19.0	Female	MECH	81.2	7.5	Mumbai	NaN	Yes
8	9	Varun Raj	25.0	Male	NaN	88.8	9.0	Hyderabad	Yes	NaN
9	10	Nisha Das	18.0	Female	CIVIL	NaN	6.7	Pune	No	No

[34]:
```
#Carry forward
df1=df.ffill()
df1
```

[34]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Aarav Sharma	22.0	Male	CSE	85.2	8.6	Chennai	Yes	No
1	2	Ishita Verma	21.0	Female	ECE	72.5	7.2	Mumbai	No	Yes
2	3	Rahul Mehta	23.0	Male	MECH	65.1	6.8	Delhi	Yes	No
3	4	Priya Iyer	23.0	Female	CIVIL	92.0	9.1	Bangalore	No	Yes
4	5	Siddharth Jain	24.0	Female	EEE	48.5	5.3	Kolkata	Yes	No
5	6	Kavya Reddy	20.0	Female	CSE	90.3	8.9	Kolkata	Yes	Yes
6	7	Aman Kumar	22.0	Male	ECE	55.0	8.9	Chennai	No	No
7	8	Meena S	19.0	Female	MECH	81.2	7.5	Mumbai	No	Yes
8	9	Varun Raj	25.0	Male	MECH	88.8	9.0	Hyderabad	Yes	Yes
9	10	Nisha Das	18.0	Female	CIVIL	88.8	6.7	Pune	No	No

[35]:
```
#Backward fill
df1=df.bfill()
df1
```

[35]:

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Aarav Sharma	22.0	Male	CSE	85.2	8.6	Chennai	Yes	No
1	2	Ishita Verma	21.0	Female	ECE	72.5	7.2	Mumbai	No	Yes
2	3	Rahul Mehta	23.0	Male	MECH	65.1	6.8	Delhi	Yes	No
3	4	Priya Iyer	24.0	Female	CIVIL	92.0	9.1	Bangalore	No	Yes

```
#Drop NA row/column
df1=df.dropna()
df1
```

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Aarav Sharma	22.0	Male	CSE	85.2	8.6	Chennai	Yes	No
1	2	Ishita Verma	21.0	Female	ECE	72.5	7.2	Mumbai	No	Yes
2	3	Rahul Mehta	23.0	Male	MECH	65.1	6.8	Delhi	Yes	No

```
df1
```

	StudentID	Name	Age	Gender	Department	Attendance	CGPA	City	HostelResident	Scholarship
0	1	Aarav Sharma	22.0	Male	CSE	85.2	8.6	Chennai	Yes	No
1	2	Ishita Verma	21.0	Female	ECE	72.5	7.2	Mumbai	No	Yes
2	3	Rahul Mehta	23.0	Male	MECH	65.1	6.8	Delhi	Yes	No

PROGRAM

Exp3.ipynb

```
import pandas as pd
data = pd.DataFrame({
    "A": [1,2,3,4,5,6],
    "B": [7,8,9,10,11,12],
    "C": [0,0,0,0,0,0],
    "D": [21,54,32,85,35,21]
})
data
from sklearn.feature_selection import VarianceThreshold
var_thres = VarianceThreshold(threshold = 0)
var_thres.fit(data)
var_thres.get_support()
data.columns[var_thres.get_support()]

# Step 1: Get the selected (non-zero variance) column names
selected_columns = data.columns[var_thres.get_support()]

# Step 2: Create an empty list to store the dropped columns (constant columns)
constant_columns = []

# Step 3: Go through all original columns
for column in data.columns:

# Step 4: If column is not in selected columns, it means it was dropped
    if column not in selected_columns:
        constant_columns.append(column)
```

```
# Print how many columns were dropped
print(len(constant_columns))

# Print the name(s) of dropped columns
for feature in constant_columns:
    print(feature)
data.drop(constant_columns, axis=1)
```

OUTPUT

[1]:

	A	B	C	D
0	1	7	0	21
1	2	8	0	54
2	3	9	0	32
3	4	10	0	85
4	5	11	0	35
5	6	12	0	21

[4]:
```
var_thres.get_support()
```

[4]:
```
array([ True,  True, False,  True])
```

[5]:
```
data.columns[var_thres.get_support()]
```

[5]:
```
Index(['A', 'B', 'D'], dtype='object')
```

[8]:
```
# Print how many columns were dropped
print(len(constant_columns))

# Print the name(s) of dropped columns
for feature in constant_columns:
    print(feature)
```

```
1
C
```

[9]:
```
data.drop(constant_columns, axis=1)
```

[9]:

	A	B	D
0	1	7	21
1	2	8	54
2	3	9	32
3	4	10	85
4	5	11	35
5	6	12	21

PROGRAM

Exp4.ipynb

```
# Step 1: Load dataset
import pandas as pd
df = pd.read_csv("diabetes.csv")

# Step 2: Feature matrix and label
X = df.drop(columns='Outcome', axis=1)
Y = df['Outcome']
X
Y

# Step 3: Train-test split
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y,
random_state=42)
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("Y_train shape:", Y_train.shape)
print("Y_test shape:", Y_test.shape)

# Step 4: Model training
model = LogisticRegression(max_iter=1000)
model.fit(X_train, Y_train)

# Step 5: Training accuracy
from sklearn.metrics import accuracy_score
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

```
print("Accuracy on Training data :", round(training_data_accuracy * 100, 2), "%")

# Step 6: Test accuracy
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
print("Accuracy on Test data :", round(test_data_accuracy * 100, 2), "%")

# Step 7: Confusion Matrix visualization
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y_test, X_test_prediction)
sns.heatmap(cm, annot=True, fmt='d', cmap='PuRd')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Step 8: Precision - Training
from sklearn.metrics import precision_score
precision_train = precision_score(Y_train, X_train_prediction)
print("Training data Precision =", precision_train)

# Step 9: Precision - Test
precision_test = precision_score(Y_test, X_test_prediction)
print("Test data Precision =", precision_test)

# Step 10: Recall - Training
from sklearn.metrics import recall_score
```

```
recall_train = recall_score(Y_train, X_train_prediction)
print("Training data Recall =", recall_train)



# Step 11: Recall - Test
recall_test = recall_score(Y_test, X_test_prediction)
print("Test data Recall =", recall_test)

# Step 12: F1 Score - Training
from sklearn.metrics import f1_score
f1_score_train = f1_score(Y_train, X_train_prediction)
print("Training data F1 Score =", f1_score_train)

# Step 13: F1 Score - Test
f1_score_test = f1_score(Y_test, X_test_prediction)
print("Test data F1 Score =", f1_score_test)

# Step 14: Generalized Evaluation Function
def precision_recall_f1_score(true_labels, pred_labels):
    precision_value = precision_score(true_labels, pred_labels)
    recall_value = recall_score(true_labels, pred_labels)
    f1_score_value = f1_score(true_labels, pred_labels)
    print('Precision =', precision_value)
    print('Recall =', recall_value)
    print('F1 Score =', f1_score_value)

# Step 15: Evaluate Training Data
precision_recall_f1_score(Y_train, X_train_prediction)
```

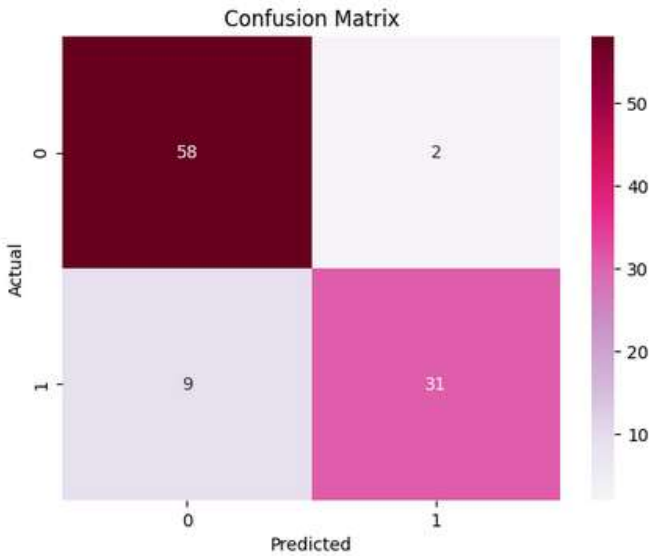
```
# Step 16: Evaluate Test Data
precision_recall_f1_score(Y_test, X_test_prediction)
```

OUTPUT

[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	-0.188093	-2.598075	-1.546738	5.012214	-2.931152	-3.599275	1.636806	-0.530395
1	-1.410538	-3.754932	-1.413862	3.383986	-0.907609	-2.568217	-0.814938	0.360186
2	-1.543991	-2.163572	0.392428	1.502724	1.642945	1.569722	-2.988584	-2.536647
3	-2.826774	-2.433848	-0.985261	0.323552	-1.035475	-1.422035	-0.328273	1.630794
4	-1.455335	-4.198515	1.154567	3.986387	1.074079	1.012788	-3.806394	-3.084259
...	...	...	...	...	...	...	...	...
495	-3.304711	1.653106	0.802366	-4.467550	2.071321	-0.410598	-1.657271	-0.521276
496	-3.050721	2.061946	-3.528852	-3.414110	-0.748969	3.720574	2.378373	0.658707
497	-2.326950	1.244817	-2.697114	0.511028	-0.999666	1.127345	1.822784	-3.003061
498	-1.704565	-1.133554	4.642454	0.742939	0.540381	-3.420329	-2.901381	-2.453981
499	0.073046	-3.039904	-2.735186	5.044834	0.568121	-1.032021	-1.244723	-2.811563

500 rows × 8 columns



[43]:
```
# Step 15: Evaluate Training Data
precision_recall_f1_score(Y_train, X_train_prediction)
```

```
Precision = 0.9115646258503401
Recall = 0.8322981366459627
F1 Score = 0.8701298701298701
```

[44]:
```
# Step 16: Evaluate Test Data
precision_recall_f1_score(Y_test, X_test_prediction)
```

```
Precision = 0.9393939393939394
Recall = 0.775
F1 Score = 0.8493150684931506
```

[ ]:

PROGRAM

Exp5a.ipynb

```
# Step 1: Import Dependencies
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB

# Step 2: Load the dataset
df = pd.read_csv("titanic.csv")
print(df.head())

# Step 3: Split the features and target
inputs = df.drop(['PassengerId','Name','SibSp','Parch','Ticket','Cabin','Embarked','Survived'],
axis='columns')
target = df['Survived']
df

# Step 4: Handle categorical data (Sex → one-hot encoding)
dummies = pd.get_dummies(inputs.Sex)
inputs = pd.concat([inputs, dummies], axis='columns')
inputs.drop(['Sex','male'], axis='columns', inplace=True)
dummies
inputs

# Step 5: Handle missing values (fill Age with mean)
inputs.Age = inputs.Age.fillna(inputs.Age.mean())
inputs

# Step 6: Split into Training and Testing sets
X_train, X_test, y_train, y_test = train_test_split(inputs, target, test_size=0.3)
```

```python
# Step 7: Train Gaussian Naïve Bayes model
model = GaussianNB()
model.fit(X_train, y_train)

# Step 8: Find accuracy on test data
print("Test Accuracy:", model.score(X_test, y_test))

# Step 9: Show some predictions vs actual
print("Actual values (first 20):")
print(y_test[:20].values)
print("Predicted values (first 20):")
print(model.predict(X_test[:20]))

# Step 10: Show prediction probabilities
print("Prediction Probabilities (first 10):")
print(model.predict_proba(X_test[:10]))

# Step 11: Perform Cross Validation
scores = cross_val_score(model, inputs, target, cv=5)
print("Cross Validation Scores:", scores)
print("Average CV Score:", scores.mean())
```

OUTPUT

```
[1]: df
```

	PassengerId	Name	Pclass	Sex	Age	SibSp	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	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	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
5	6	Moran Mr. James	3	male	NaN	0	0	330877	8.4583	NaN	Q	0
6	7	McCarthy Mr. Timothy J	1	male	54.0	0	0	17463	51.8625	E46	S	0
7	8	Palsson Master. Gosta Leonard	3	male	2.0	3	1	349909	21.0750	NaN	S	0
8	9	Johnson Mrs. Oscar W	3	female	27.0	0	2	347742	11.1333	NaN	S	1
9	10	Nasser Mrs. Nicholas	2	female	14.0	1	0	237736	30.0708	NaN	C	1

```
[11]: # Step 5: Handle missing values (fill Age with mean)
      inputs.Age = inputs.Age.fillna(inputs.Age.mean())
```

```
[19]: inputs
```

	Pclass	Age	Fare	female
0	3	22.000000	7.2500	False
1	1	38.000000	71.2833	True
2	3	26.000000	7.9250	True
3	1	35.000000	53.1000	True
4	3	35.000000	8.0500	False
5	3	28.111111	8.4583	False
6	1	54.000000	51.8625	False
7	3	2.000000	21.0750	False
8	3	27.000000	11.1333	True
9	2	14.000000	30.0708	True

```
Actual values (first 20):
[1 0 1]
Predicted values (first 20):
[1 0 1]
```

```
[17]: # Step 10: Show prediction probabilities
      print("Prediction Probabilities (first 10):")
      print(model.predict_proba(X_test[:10]))
```

```
Prediction Probabilities (first 10):
[[0. 1.]
 [1. 0.]
 [0. 1.]]
```

```
[18]: # Step 11: Perform Cross Validation
      scores = cross_val_score(model, inputs, target, cv=5)
      print("Cross Validation Scores:", scores)
      print("Average CV Score:", scores.mean())
```

```
Cross Validation Scores: [1.  1.  1.  0.5 1. ]
Average CV Score: 0.9
```

PROGRAM

Exp5b.ipynb

```
# Step 1: Import Dependencies
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.pipeline import Pipeline

# Step 2: Load the dataset
df = pd.read_csv("spam.csv")
print(df.head())

# Step 3: Categorize the target (convert ham/spam to 0/1)
df['spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)
print(df.head())

# Step 4: Split into Training and Test sets
X_train, X_test, y_train, y_test = train_test_split(df.Message, df.spam, test_size=0.25)

# Step 5: Import CountVectorizer to convert text → numeric features
v = CountVectorizer()
X_train_count = v.fit_transform(X_train.values)

# Step 6: Train Multinomial Naïve Bayes model
model = MultinomialNB()
model.fit(X_train_count, y_train)
```

```
# Step 7: Transform test data and check accuracy
X_test_count = v.transform(X_test)
print("Test Accuracy:", model.score(X_test_count, y_test))

# Step 8: Use Pipeline (CountVectorizer + MultinomialNB together)
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])

# Step 9: Train Pipeline model
clf.fit(X_train, y_train)

# Step 10: Evaluate accuracy using Pipeline
print("Pipeline Test Accuracy:", clf.score(X_test, y_test))

# Step 11: Predict on new emails
emails = [
    "Hello, please call our customer care immediately for a prize",
    "Hi John, are we still meeting for lunch tomorrow?"
]
print("Predictions for new emails:", clf.predict(emails))
```

OUTPUT

```
[13]: # Step 2: Load the dataset
df = pd.read_csv("spam.csv")
print(df.head())
```

```
  Category                                           Message
0      ham             Let's catch up over coffee this weekend.
1      ham                         Are you free for lunch today?
2     spam  Congratulations, you are selected for a cash r...
3      ham                      Hey, are we still meeting later?
4     spam                     Claim your free gift voucher today!
```

```
[14]: # Step 3: Categorize the target (convert ham/spam to 0/1)
df['spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)
print(df.head())
```

```
  Category                                           Message  spam
0      ham             Let's catch up over coffee this weekend.     0
1      ham                         Are you free for lunch today?     0
2     spam  Congratulations, you are selected for a cash r...     1
3      ham                      Hey, are we still meeting later?     0
4     spam                     Claim your free gift voucher today!     1
```

```
[21]: # Step 10: Evaluate accuracy using Pipeline
print("Pipeline Test Accuracy:", clf.score(X_test, y_test))
```

```
Pipeline Test Accuracy: 0.92
```

```
[22]: # Step 11: Predict on new emails
emails = [
    "Hello, please call our customer care immediately for a prize",
    "Hi John, are we still meeting for lunch tomorrow?"
]
print("Predictions for new emails:", clf.predict(emails))
```

```
Predictions for new emails: [1 0]
```

PROGRAM

Exp6.ipynb

```
# Step 1: Import the dependencies
import numpy as np
import pandas as pd
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import DiscreteBayesianNetwork
from pgmpy.inference import VariableElimination
from sklearn.preprocessing import KBinsDiscretizer

# Step 2: Load the dataset into a Pandas DataFrame
hd = pd.read_csv("heart.csv")
hd = hd.replace('?', np.nan)   # handle missing values
print("Original dataset sample:")
print(hd.head())

# Step 3: Discretize continuous features into bins
continuous_cols = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
disc = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
# Fit and transform continuous columns
hd[continuous_cols] = disc.fit_transform(hd[continuous_cols])
# Convert discretized values to int
hd[continuous_cols] = hd[continuous_cols].astype(int)

# Step 4: Define Bayesian Model structure
model = DiscreteBayesianNetwork([
    ('age','target'), ('sex','target'), ('cp','target'),
    ('trestbps','target'), ('chol','target'), ('fbs','target'),
    ('restecg','target'), ('thalach','target'), ('exang','target'),
    ('oldpeak','target'), ('slope','target'), ('ca','target'),
```

```
    ('thal','target')
])
# Step 5: Convert all columns to string (categorical labels)
for col in hd.columns:
    hd[col] = hd[col].astype(str)
# Train the model with Maximum Likelihood Estimator
model.fit(hd, estimator=MaximumLikelihoodEstimator)
for cpd in model.get_cpds():
    print(cpd)

# Step 6: Use Variable Elimination for inference
hd_infer = VariableElimination(model)

# Step 7: Query the model with evidence
print("\n1. Probability of heart disease given evidence = restecg:1")
q1 = hd_infer.query(variables=['target'], evidence={'restecg': '1'})
print(q1)

print("\n2. Probability of heart disease given evidence = cp:2")
q2 = hd_infer.query(variables=['target'], evidence={'cp': '2'})
print(q2)
```

OUTPUT

```
print(hd.head())
```

```
Original dataset sample:
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
0   67    1   1       136   145    1        0      149      1      4.2      0
1   57    1   2       114   554    0        0      161      0      5.0      0
2   43    1   3       155   292    0        0      166      0      4.8      1
3   71    0   0        99   419    1        0      128      0      0.9      1
4   36    1   0       145   252    0        1      128      0      2.6      0

   ca  thal  target
0   4     0       0
1   3     0       1
2   1     0       0
3   3     0       0
4   3     2       0
```

```
print(cpd)
```

```
+--------+----------+
| age(0) | 0.186667 |
+--------+----------+
| age(1) | 0.213333 |
+--------+----------+
| age(2) | 0.19     |
+--------+----------+
| age(3) | 0.203333 |
+--------+----------+
| age(4) | 0.206667 |
+--------+----------+
```

```
1. Probability of heart disease given evidence = restecg:1
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.5000 |
+-----------+---------------+
| target(1) |        0.5000 |
+-----------+---------------+

2. Probability of heart disease given evidence = cp:2
+-----------+---------------+
| target    |   phi(target) |
+===========+===============+
| target(0) |        0.5000 |
+-----------+---------------+
| target(1) |        0.5000 |
+-----------+---------------+
```

PROGRAM

Exp7.ipynb

```
# Step 1 — Imports
import os, warnings
os.environ["LOKY_MAX_CPU_COUNT"] = "4"
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
from sklearn.mixture import GaussianMixture
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns

# (Optional) display settings
%matplotlib inline
sns.set(style="whitegrid")

# Step 2 — Load the CSV (use the iris.csv I created)
df = pd.read_csv("iris.csv")
print("Shape:", df.shape)
df.head()

# Step 3 — Prepare features and (optional) true labels
X = df.iloc[:, 0:4].values      # feature columns
y_true = df['target'].values    # ground-truth species (0,1,2) for evaluation

# Standardize features (recommended for GMM)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
# Step 4 — Fit Gaussian Mixture Model (EM)
# n_components: number of clusters (3 for Iris)
gmm = GaussianMixture(n_components=3, covariance_type='full', random_state=42,
init_params='kmeans', max_iter=200)
gmm.fit(X_scaled)

# Predicted cluster labels
y_pred = gmm.predict(X_scaled)

print("Converged:", gmm.converged_)
print("n_iter:", gmm.n_iter_)

# Step 5 — Inspect cluster sizes and means
unique, counts = np.unique(y_pred, return_counts=True)
print("Cluster counts:", dict(zip(unique, counts)))

# Cluster means in the scaled space (for interpretation convert back if needed)
print("Cluster means (scaled):\n", gmm.means_)

# Step 6 — Map cluster labels to true labels for evaluation (best matching)
# Since clustering labels are arbitrary, find mapping by majority vote per cluster
from scipy.stats import mode
import numpy as np

label_map = {}
for cluster in np.unique(y_pred):
    mask = (y_pred == cluster)
    if mask.sum() == 0:
        label_map[cluster] = -1
    else:
```

```
        m = mode(y_true[mask], keepdims=True)
        label_map[cluster] = int(m.mode[0] if hasattr(m.mode, "__getitem__") else m.mode)

mapped_preds = np.array([label_map[c] for c in y_pred])

print("Label mapping (cluster -> majority true label):", label_map)

# Step 7 — Evaluation (confusion matrix & accuracy)
cm = confusion_matrix(y_true, mapped_preds)
acc = accuracy_score(y_true, mapped_preds)

print("\nConfusion Matrix:\n", cm)
print("\nAccuracy (after mapping): {:.4f}".format(acc))

# Step 8 — Visualize clusters (2D scatter using first two principal components)
from sklearn.decomposition import PCA

pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.title("Clusters by GMM (predicted)")
plt.scatter(X_pca[:,0], X_pca[:,1], c=y_pred, cmap='viridis', s=40)
plt.xlabel("PC1"); plt.ylabel("PC2")

plt.subplot(1,2,2)
plt.title("True labels")
plt.scatter(X_pca[:,0], X_pca[:,1], c=y_true, cmap='viridis', s=40)
```

```
plt.xlabel("PC1"); plt.ylabel("PC2")

plt.tight_layout()
plt.show()
```

OUTPUT

```
df.head()
```

Shape: (150, 5)

[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
Converged: True
n_iter: 11
```

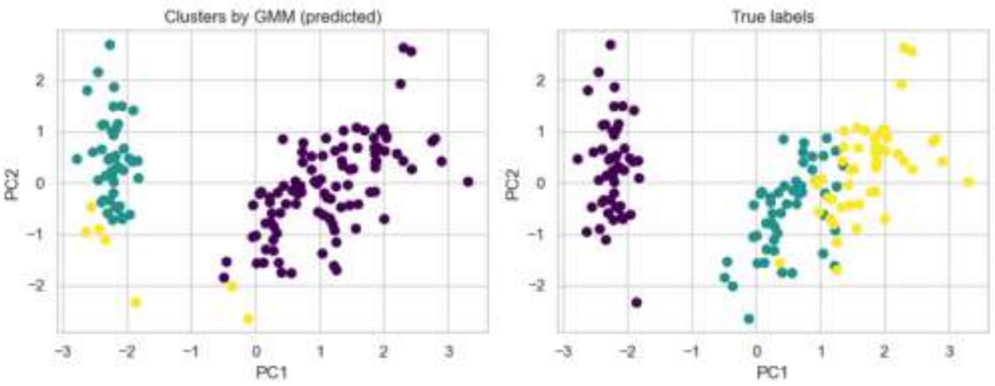
[7]:

```
# Step 5 — Inspect cluster sizes and means
unique, counts = np.unique(y_pred, return_counts=True)
print("Cluster counts:", dict(zip(unique, counts)))

# Cluster means in the scaled space (for interpretation convert back if needed)
print("Cluster means (scaled):\n", gmm.means_)
```

```
Cluster counts: {0: 98, 1: 45, 2: 7}
Cluster means (scaled):
 [[ 0.53745909 -0.39369142  0.6693573   0.64500292]
 [-0.93852253  0.98617415 -1.29410958 -1.24871335]
 [-1.53616188 -0.9148767  -1.05760659 -1.00758605]]
```

```
plt.tight_layout()
plt.show()
```



PROGRAM

Exp8.ipynb

```
#Step 1: Import Necessary Libraries
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline

#Step 2: Load the Iris Dataset
iris = load_iris()

#Step 3: Explore Feature and Target Names
print(iris.feature_names)
print(iris.target_names)

#Step 4: Create a DataFrame from the Data
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df.head()  # Display the first 5 rows

#Step 5: Add Flower Names for Readability
df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
df.head()  # Display updated DataFrame

#Step 6: Explore Subsets of Data
df[df.target == 0].head()
```

```
df[df.target == 1].head()
df[df.target == 2].head()

#Step 7: Prepare Features (X) and Target (y)
X = df.drop(['target', 'flower_name'], axis='columns')
y = df.target

#Step 8: Split Data into Training and Testing Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
print(len(X_train))
print(len(X_test))

#Step 9: Create and Train the k-NN Model
knn = KNeighborsClassifier(n_neighbors=10)
knn.fit(X_train, y_train)

#Step 10: Evaluate Model Accuracy
print(knn.score(X_test, y_test))

#Step 12: Generate Confusion Matrix
y_pred = knn.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
print(cm)
plt.figure(figsize=(7, 5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
plt.show()  # Displays the heatmap plot
#Step 13: Generate Classification Report
print(classification_report(y_test, y_pred))
```

OUTPUT

[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

[5]:

```
#Step 5: Add Flower Names for Readability
df['flower_name'] = df.target.apply(lambda x: iris.target_names[x])
df.head()  # Display updated DataFrame
```

[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

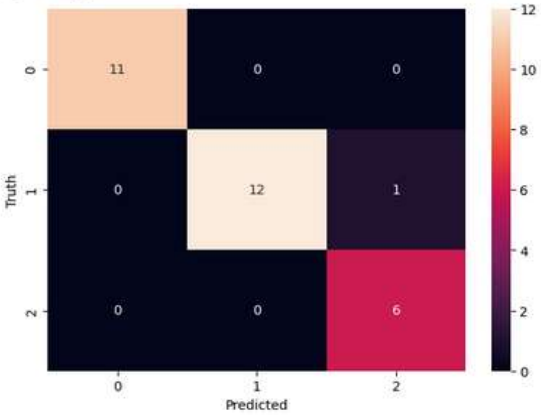
[6]:

```
#Step 6: Explore Subsets of Data
df[df.target == 0].head()  # Setosa samples
df[df.target == 1].head()  # Versicolor samples
df[df.target == 2].head()  # Virginica samples
```

[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	flower_name
100	6.3	3.3	6.0	2.5	2	virginica
101	5.8	2.7	5.1	1.9	2	virginica
102	7.1	3.0	5.9	2.1	2	virginica
103	6.3	2.9	5.6	1.8	2	virginica
104	6.5	3.0	5.8	2.2	2	virginica

```
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]
```



[15]:
```
#Step 14: Generate Classification Report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.92      0.96        13
           2       0.86      1.00      0.92         6

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

PROGRAM

Exp9.ipynb

```
#Step 1: Import Necessary Libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
%matplotlib inline

#Step 2: Load the Dataset
df = pd.read_csv('salaries.csv')
df

#Step 3: Encode Categorical Variables
le_company = LabelEncoder()
le_job = LabelEncoder()
le_degree = LabelEncoder()

df['company_n'] = le_company.fit_transform(df['company'])
df['job_n'] = le_job.fit_transform(df['job'])
df['degree_n'] = le_degree.fit_transform(df['degree'])

df  # Display updated DataFrame with encoded columns

#Step 4: Prepare Features (X) and Target (y)
X = df.drop(['company', 'job', 'degree', 'salary_more_then_100k'], axis='columns')
y = df['salary_more_then_100k']

#Step 5: Build and Fit the Unpruned Decision Tree
model_unpruned = DecisionTreeClassifier(criterion='gini', random_state=0)
```

```
model_unpruned.fit(X, y)
print(model_unpruned.score(X, y))

#Step 6: Visualize the Unpruned Tree
plt.figure(figsize=(12, 8))
plot_tree(model_unpruned, feature_names=['company_n', 'job_n', 'degree_n'],
class_names=['<=100k', '>100k'], filled=True, rounded=True)
plt.title('Unpruned Decision Tree')
plt.show()
# The tree will have a depth of around 3 and 7 nodes (full fit to data)

#Step 7: Build and Fit the Pruned Decision Tree
model_pruned = DecisionTreeClassifier(criterion='gini', max_depth=2, random_state=0)
model_pruned.fit(X, y)
print(model_pruned.score(X, y))

#Step 8: Analyze and Compare the Structures
def tree_summary(model):
    print(f"Depth: {model.tree_.max_depth}")
    print(f"Number of leaves: {model.tree_.n_leaves}")
    print(f"Number of nodes: {model.tree_.node_count}")

print("Unpruned Tree Summary:")
tree_summary(model_unpruned)
# Sample Output: Depth: 3, Number of leaves: 4, Number of nodes: 7

print("\nPruned Tree Summary:")
tree_summary(model_pruned)
# Sample Output: Depth: 2, Number of leaves: 3, Number of nodes: 5
```

print("\nAnalysis: The unpruned tree is deeper and has more nodes/leaves, fully fitting the data (potentially overfitting noise). The pruned tree is shallower and simpler, reducing complexity while maintaining high accuracy, making it better for generalization on noisy data.")

#Step 9: Make Predictions (Example)

print(model_unpruned.predict([[2, 1, 0]]))

print(model_pruned.predict([[2, 1, 0]]))

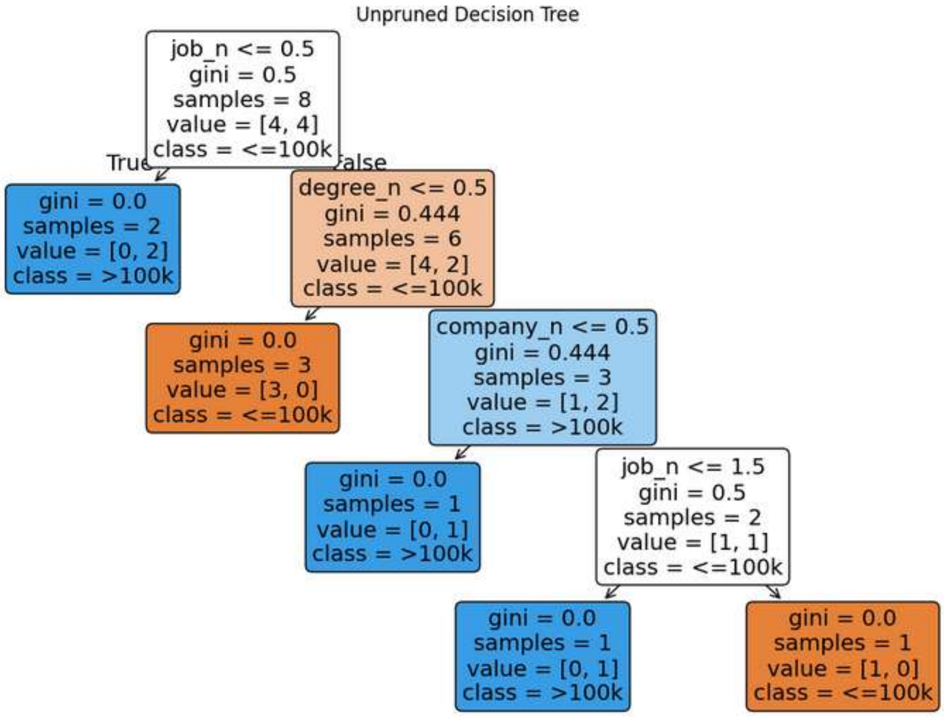
print(model_unpruned.predict([[2, 1, 1]]))

OUTPUT

```
df
```

[3]:

	company	job	degree	salary_more_then_100k
0	google	sales executive	bachelors	0
1	google	sales executive	masters	0
2	google	business manager	bachelors	1
3	google	business manager	masters	1
4	google	computer programmer	bachelors	0
5	google	computer programmer	masters	1
6	abc pharma	sales executive	masters	1
7	abc pharma	computer programmer	bachelors	0



```
Unpruned Tree Summary:
Depth: 4
Number of leaves: 5
Number of nodes: 9

Pruned Tree Summary:
Depth: 2
Number of leaves: 3
Number of nodes: 5

Analysis: The unpruned tree is deeper and has more nodes/leaves, fully fitting the data (potentially overfitting noise).
impler, reducing complexity while maintaining high accuracy, making it better for generalization on noisy data.
```

[11]:

```python
#Step 12: Make Predictions (Example)
print(model_unpruned.predict([[2, 1, 0]]))

print(model_pruned.predict([[2, 1, 0]]))

print(model_unpruned.predict([[2, 1, 1]]))
```

```
[0]
[0]
[1]
```

PROGRAM

Exp10.ipynb

```
#Step 1: Import Necessary Libraries
import numpy as np

#Step 2: Define the Neural Network Class
class NN:
    def __init__(self, input_size, hidden_size, output_size):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        # Initialize weights randomly
        self.w1 = np.random.randn(self.input_size, self.hidden_size)
        self.w2 = np.random.randn(self.hidden_size, self.output_size)

    def forward(self, x):
        # Forward pass
        self.z = np.dot(x, self.w1)
        self.z2 = 1 / (1 + np.exp(-self.z))  # Sigmoid activation for hidden layer
        self.z3 = np.dot(self.z2, self.w2)
        o = 1 / (1 + np.exp(-self.z3))  # Sigmoid activation for output layer
        return o

    def backward(self, x, y, o):
        # Backpropagation
        self.o_error = y - o  # Error in output
        self.o_delta = self.o_error * o * (1 - o)  # Derivative of sigmoid for output
        self.z2_error = np.dot(self.o_delta, self.w2.T)  # Error in hidden layer
        self.z2_delta = self.z2_error * self.z2 * (1 - self.z2)  # Derivative of sigmoid for hidden
        # Update weights (learning rate implicit as 1.0)
```

```
        self.w1 += np.dot(x.T, self.z2_delta)
        self.w2 += np.dot(self.z2.T, self.o_delta)

#Step 3: Prepare the Dataset (XOR)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

#Step 4: Initialize and Train the Model
nn = NN(2, 2, 1)  # Input size 2, hidden 2, output 1

for i in range(10000):
    o = nn.forward(X)
    nn.backward(X, y, o)
    loss = np.mean(np.square(y - o))  # Mean squared error
    if i % 1000 == 0:
        print("Loss:", loss)

#Step 5: Make Predictions and Test
predicted = nn.forward(X)
print("Input:\n", X)
print("Actual Output:\n", y)
print("Predicted Output:\n", predicted)

#Step 6: Test on a New Sample
sample = np.array([[0, 0]])
print("Predicted for [0,0]:", nn.forward(sample))
```

OUTPUT

```
Loss: 0.2581671448741397
Loss: 0.13226318945341617
Loss: 0.12716325362540953
Loss: 0.12624223624388126
Loss: 0.1258652253012918
Loss: 0.12566159952429598
Loss: 0.12553455971534855
Loss: 0.12544791625206964
Loss: 0.12538512835219917
Loss: 0.1253375802505379
```

```
[5]: #Step 5: Make Predictions and Test
predicted = nn.forward(X)
print("Input:\n", X)
print("Actual Output:\n", y)
print("Predicted Output:\n", predicted)
```

```
Input:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]]
Actual Output:
 [[0]
 [1]
 [1]
 [0]]
Predicted Output:
 [[0.01941308]
 [0.49979919]
 [0.98218634]
 [0.50030626]]
```

```
[7]: #Step 6: Test on a New Sample
sample = np.array([[0, 0]])
print("Predicted for [0,0]:", nn.forward(sample))
```

```
Predicted for [0,0]: [[0.01941308]]
```

PROGRAM

Exp11.ipynb

```
import pandas as pd

import numpy as np

from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC


cancer_data = load_breast_cancer()

cancer_data.target_names

df = pd.DataFrame(cancer_data.data, columns=cancer_data.feature_names)

df['diagnosis'] = cancer_data.target

df.head()

X = df.drop(columns='diagnosis', axis=1)

y = df['diagnosis']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, stratify=y,
random_state=42)


print(f"X_train shape:{X_train.shape}")

print(f"X_test shape:{X_test.shape}")

print(f"y_train shape: {y_train.shape}")

print(f"y_test shape: {y_test.shape}")


svm = SVC(kernel='linear')

svm.fit(X_train, y_train)


print(f"Accuracy on training subset is: {svm.score(X_train, y_train):.3f}")

print(f"Accuracy on test subset is: {svm.score(X_test, y_test):.3f}")
```

```
scaler = StandardScaler()

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

svm = SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)

# Display the accuracy on the scaled data (In[10] in image)
train_acc_scaled = svm.score(X_train_scaled, y_train)
test_acc_scaled = svm.score(X_test_scaled, y_test)

print(f"Accuracy on training subset is: {train_acc_scaled:.3f}")
print(f"Accuracy on test subset is:    {test_acc_scaled:.3f}")
```

OUTPUT

```
cancer_data.target_names
```

```
array(['malignant', 'benign'], dtype='<U9')
```

```
df.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...

5 rows × 31 columns

```
print(f"X_train shape:{X_train.shape}")
print(f"X_test shape:{X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape:(426, 30)
X_test shape:(143, 30)
y_train shape: (426,)
y_test shape: (143,)
```

```
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
```

```
SVC
▶ Parameters
```

```python
print(f"Accuracy on training subset is: {svm.score(X_train, y_train):.3f}")
print(f"Accuracy on test subset is: {svm.score(X_test, y_test):.3f}")
```

```
Accuracy on training subset is: 0.962
Accuracy on test subset is: 0.951
```

```python
svm = SVC(kernel='linear')
svm.fit(X_train_scaled, y_train)

# Step 10: Display the accuracy on the scaled data (In[10] in image)
train_acc_scaled = svm.score(X_train_scaled, y_train)
test_acc_scaled = svm.score(X_test_scaled, y_test)

print(f"Accuracy on training subset is: {train_acc_scaled:.3f}")
print(f"Accuracy on test subset is:    {test_acc_scaled:.3f}")
```

```
Accuracy on training subset is: 0.991
Accuracy on test subset is:    0.986
```

PROGRAM

Exp12a.ipynb

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn.preprocessing import LabelEncoder # Required for text preprocessing


data = pd.read_csv('spam.csv')

data.head()

data['Message'].value_counts()

# Label Encoding for Target Variable 'Category'

encoder = LabelEncoder()

data['Category'] = encoder.fit_transform(data['Category'])

# 'ham' becomes 0, 'spam' becomes 1

# Since Logistic Regression works on numbers, we must convert the text message into a
numerical representation.

from sklearn.feature_extraction.text import TfidfVectorizer


data['Message'] = data['Message'].astype(str)

feature_extractor = TfidfVectorizer(min_df=1, stop_words='english', lowercase=True)

X_features = feature_extractor.fit_transform(data['Message'])

y = data['Category']

X = X_features

y

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=2)

print(X.shape,X_train.shape,X_test.shape)


model = LogisticRegression(max_iter=1000)
```

```
# Train the model using the training data
model.fit(X_train, y_train)
X_train_prediction = model.predict(X_train)
trained_data_accuracy = accuracy_score(y_train, X_train_prediction)

print('Accuracy on training data: {} %'.format(round(trained_data_accuracy * 100, 2)))
# Accuracy on Test Data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(y_test, X_test_prediction)

print('Accuracy on test data: {} %'.format(round(test_data_accuracy * 100, 2)))
```

OUTPUT

	Category	Message
0	ham	Go until jurong point
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	Nah I don't think he goes to usf
4	spam	Had your mobile 11 months or more? U R entitle...

```
data['Message'].value_counts()
```

```
Message
Congratulations! You've been selected to receive a FREE $100 Gift Card to use at any high street store.
67
Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for
86030    66
You have won $1000 cash prize! Call 09061701323 now to claim!
66
URGENT! Your mobile number has won a $1000 cash prize! Call 09061701323 now to claim!
66
Hi Babe
66

..
I'm going to be more grateful.
1
I'm going to be more mindful.
1
I'm going to be more positive.
1
I'm going to be more optimistic.
1
nan
1
Name: count, Length: 130, dtype: int64
```

```
X = X_features
```

```
y
```

```
0       0
1       0
2       1
3       0
4       1
       ..
1398    0
1399    0
1400    1
1401    0
1402    0
Name: Category, Length: 1403, dtype: int64
```

```python
print(X.shape,X_train.shape,X_test.shape)
```

```
(1403, 241) (1122, 241) (281, 241)
```

```python
model = LogisticRegression(max_iter=1000)

# Train the model using the training data
model.fit(X_train, y_train)
```

```
LogisticRegression
▶ Parameters
```

```python
X_train_prediction = model.predict(X_train)
trained_data_accuracy = accuracy_score(y_train, X_train_prediction)

print('Accuracy on training data: {} %'.format(round(trained_data_accuracy * 100, 2)))
```

```
Accuracy on training data: 100.0 %
```

```python
# Accuracy on Test Data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(y_test, X_test_prediction)

print('Accuracy on test data: {} %'.format(round(test_data_accuracy * 100, 2)))
```

```
Accuracy on test data: 99.29 %
```

PROGRAM

Exp12b.ipynb

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


data = pd.read_csv('diabetes.csv')

data.head()

data['Outcome'].value_counts()

X = data.drop(columns='Outcome', axis=1)

y = data['Outcome']

X

y

# Split data into 80% Training and 20% Test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=2)

print(X.shape,X_train.shape,X_test.shape)

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

X_train_prediction = model.predict(X_train)

# Calculate accuracy score

trained_data_accuracy = accuracy_score(y_train, X_train_prediction)

print(trained_data_accuracy)

print('Accuracy on training data: {} %'.format(round(trained_data_accuracy * 100, 2)))

X_test_prediction = model.predict(X_test)

# Calculate accuracy score

test_data_accuracy = accuracy_score(y_test, X_test_prediction)

print(test_data_accuracy)

print('Accuracy on test data: {} %'.format(round(test_data_accuracy * 100, 2)))
```

OUTPUT

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
x
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33
...	...	...	...	...	...	...	...	...

```
y
```

```
0      1
1      0
2      1
3      0
4      1
      ..
443    1
444    0
445    0
446    1
447    0
Name: Outcome, Length: 448, dtype: int64
```

```python
print(X.shape,X_train.shape,X_test.shape)
```

```
(448, 8) (358, 8) (90, 8)
```

```python
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
X_train_prediction = model.predict(X_train)

# Calculate accuracy score
trained_data_accuracy = accuracy_score(y_train, X_train_prediction)
print(trained_data_accuracy)
```

```
0.770949720670391
```

```python
X_test_prediction = model.predict(X_test)

# Calculate accuracy score
test_data_accuracy = accuracy_score(y_test, X_test_prediction)
print(test_data_accuracy)
```

```
0.7888888888888889
Accuracy on test data: 78.89 %
```

```python
print('Accuracy on test data: {} %'.format(round(test_data_accuracy * 100, 2)))
```

```
Accuracy on test data: 78.89 %
```

PROGRAM

Exp13.ipynb

```
# Customer Segmentation using K-Means Clustering

# Step 1: Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
sns.set(style='whitegrid')
customer_data = pd.read_csv('Mall_Customers.csv')

# Step 2: First 5 rows in the dataframe
print(customer_data.head())

# Step 3: Finding the number of rows and columns (as seen in your image)
print(customer_data.shape)

# Step 4: Getting some information about the dataset
customer_data.info()

# Step 5: Checking for missing values (as seen in your image)
print(customer_data.isnull().sum())
X = customer_data.iloc[:, [3, 4]].values
print(X[:10])
```

```
# Step 6 & 7: Finding WCSS and Plotting the Elbow Graph
# ---------------------------------------------------------------------------

# Finding wcss value for different number of clusters
wcss = []
# Test for k=1 up to k=10 clusters
for i in range(1, 11):
    # 'k-means++' ensures smart initialization of centroids to speed up convergence
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42, n_init='auto')
    kmeans.fit(X)
    wcss.append(kmeans.inertia_) # 'inertia_' is the WCSS value
# Plot an elbow graph
plt.figure(figsize=(4, 4))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('The Elbow Point Graph')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS')
plt.show()
# Step 8: Optimum Number of Clusters = 5 (Determined from the elbow point in the graph)

#Training the k-Means Clustering Model with K=5
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42, n_init='auto')

# Step 10: Return a label for each data point based on their cluster
Y = kmeans.fit_predict(X)
print(Y)

# Get the coordinates of the final centroids
centroids = kmeans.cluster_centers_
```

```
#5 clusters - 0,1,2,3,4
#Visualizing all the clusters
#plotting all the clusters and their Centroids

plt.figure(figsize=(6, 4))
plt.title('Customer Groups (K=5 Clustering)')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')

plt.scatter(X[Y == 0, 0], X[Y == 0, 1], s=50, c='green', label='Cluster 1')
plt.scatter(X[Y == 1, 0], X[Y == 1, 1], s=50, c='red', label='Cluster 2')
plt.scatter(X[Y == 2, 0], X[Y == 2, 1], s=50, c='yellow', label='Cluster 3')
plt.scatter(X[Y == 3, 0], X[Y == 3, 1], s=50, c='violet', label='Cluster 4')
plt.scatter(X[Y == 4, 0], X[Y == 4, 1], s=50, c='blue', label='Cluster 5')

#plot the centroids
plt.scatter(centroids[:, 0], centroids[:, 1], s=100, c='cyan', label='Centroids', marker='o')

plt.legend()
plt.show()
```

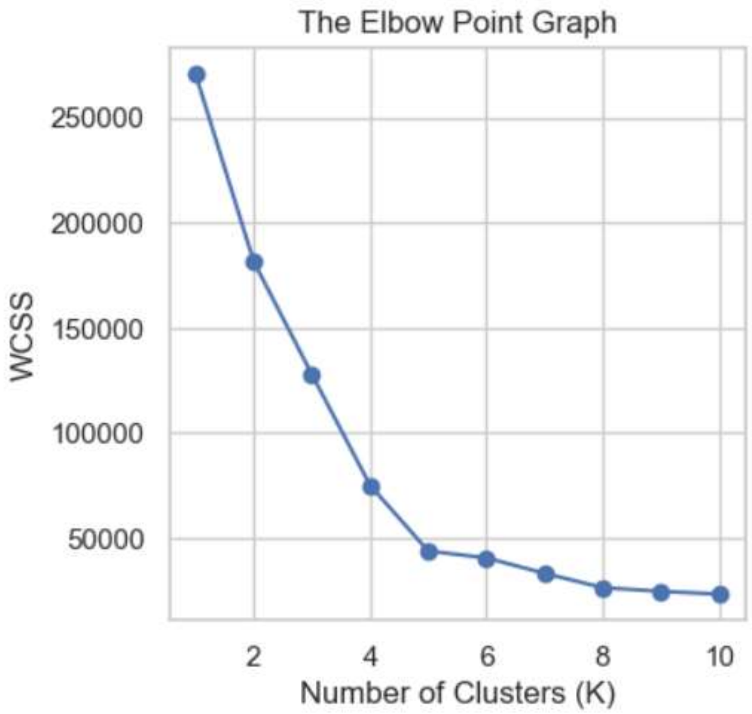
OUTPUT

```
# Step 2: First 5 rows in the dataframe
print(customer_data.head())
```

```
   CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

```
X = customer_data.iloc[:, [3, 4]].values
print(X[:10])
```

```
[[15 39]
 [15 81]
 [16  6]
 [16 77]
 [17 40]
 [17 76]
 [18  6]
 [18 94]
 [19  3]
 [19 72]]
```



```python
#Training the k-Means Clustering Model with K=5
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42, n_init='auto')

# Step 10: Return a label for each data point based on their cluster
Y = kmeans.fit_predict(X)
print(Y)

# Get the coordinates of the final centroids
centroids = kmeans.cluster_centers_
```

```
[4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4
 2 4 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 3 1 0 1 3 1 3 1 0 1 3 1 3 1 3 1 3 1 0 1 3 1 3 1
 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3 1 3
 1 3 1 3 1 1 1 3 1 3 1 3 1 3 1]
```