SCSA2412	MACHINE LEARNING LAB	L	T	P	Credits	Total Marks
		0	0	4	2	100

COURSE OBJECTIVES

- Make use of Data sets in implementing the machine learning algorithms
- Implement the machine learning concepts and algorithms in any suitable language of choice.
- Apply machine learning algorithms to real world problems

SUGGESTED LIST OF EXPERIMENTS:

1. Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.
2. Implement Simple Linear Regression
3. Implement Multivariate Linear Regression
4. Implement Logistic Regression
5. Implement Multivariate Logistic Regression
6. Data preprocessing for classification
7. Confusion matrix for a binary classifier.
8. Implement Support Vector Machines.
9. Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
10. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.
11. 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.
12. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
13. Implement k-Mean's algorithm to cluster a set of data stored in a .CSV file.
14. Implement Random Forest models for automatic classification
15. Implement Ensemble Model to perform classification.

COURSEOUTCOMES

At the end of the course, the student will be able to,

CO1: Understand the implementation procedures for the machine learning algorithms.
CO2: Design python programs for various learning algorithms.
CO3: Apply appropriate data sets to the machine learning algorithms.
CO4: Identify and apply machine learning algorithms to solve real world problems.
CO5: Apply various clustering algorithms for various applications
CO6: Analyze various machine learning algorithms and their applications

1) *Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .CSV file.*

CODE :

```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")

#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)

#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)

#training function to implement find-s algorithm
def train(c,t):
    for i, val in enumerate(t):
        if val == "Yes":
            specific_hypothesis = c[i].copy()
            break

    for i, val in enumerate(c):
        if t[i] == "Yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
                else:
```

```
            pass

    return specific_hypothesis

#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

Output + Dataset :

```
=============== RESTART: C:/Users/sanke/Downloads/ML 1415/1/1P.py ==============
     Outlook Temperature Humidity  Windy PlayTennis
0      Sunny         Hot     High  False         No
1      Sunny         Hot     High   True         No
2   Overcast         Hot     High  False        Yes
3      Rainy        Mild     High  False        Yes
4      Rainy        Cool   Normal  False        Yes
5      Rainy        Cool   Normal   True         No
6   Overcast        Cool   Normal   True        Yes
7      Sunny        Mild     High  False         No
8      Sunny        Cool   Normal  False        Yes
9      Rainy        Mild   Normal  False        Yes
10     Sunny        Mild   Normal   True        Yes
11  Overcast        Mild     High   True        Yes
12  Overcast         Hot   Normal  False        Yes
13     Rainy        Mild     High   True         No n
n The attributes are:  [['Sunny' 'Hot' 'High' False]
 ['Sunny' 'Hot' 'High' True]
 ['Overcast' 'Hot' 'High' False]
 ['Rainy' 'Mild' 'High' False]
 ['Rainy' 'Cool' 'Normal' False]
 ['Rainy' 'Cool' 'Normal' True]
 ['Overcast' 'Cool' 'Normal' True]
 ['Sunny' 'Mild' 'High' False]
 ['Sunny' 'Cool' 'Normal' False]
 ['Rainy' 'Mild' 'Normal' False]
 ['Sunny' 'Mild' 'Normal' True]
 ['Overcast' 'Mild' 'High' True]
 ['Overcast' 'Hot' 'Normal' False]
 ['Rainy' 'Mild' 'High' True]]
n The target is:  ['No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes' 'Yes' 'Yes'
 'Yes' 'Yes'
 'No']
n The final hypothesis is: ['?' '?' '?' '?']
```

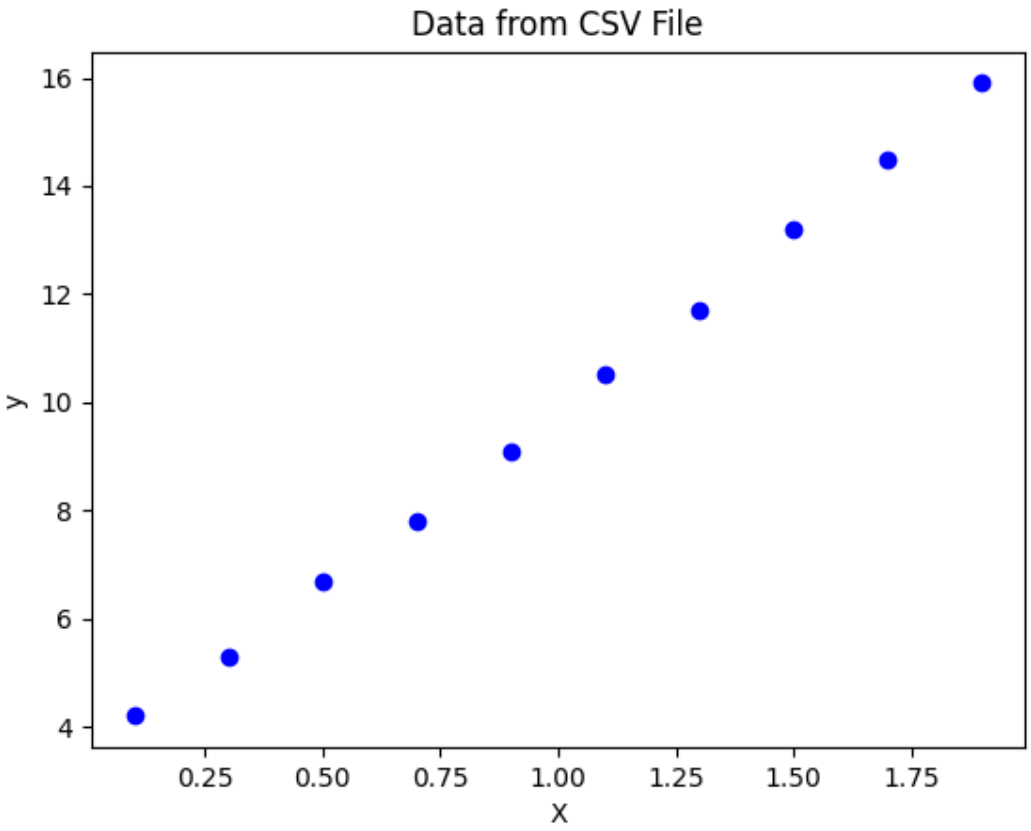
2) Implement Simple Linear Regression

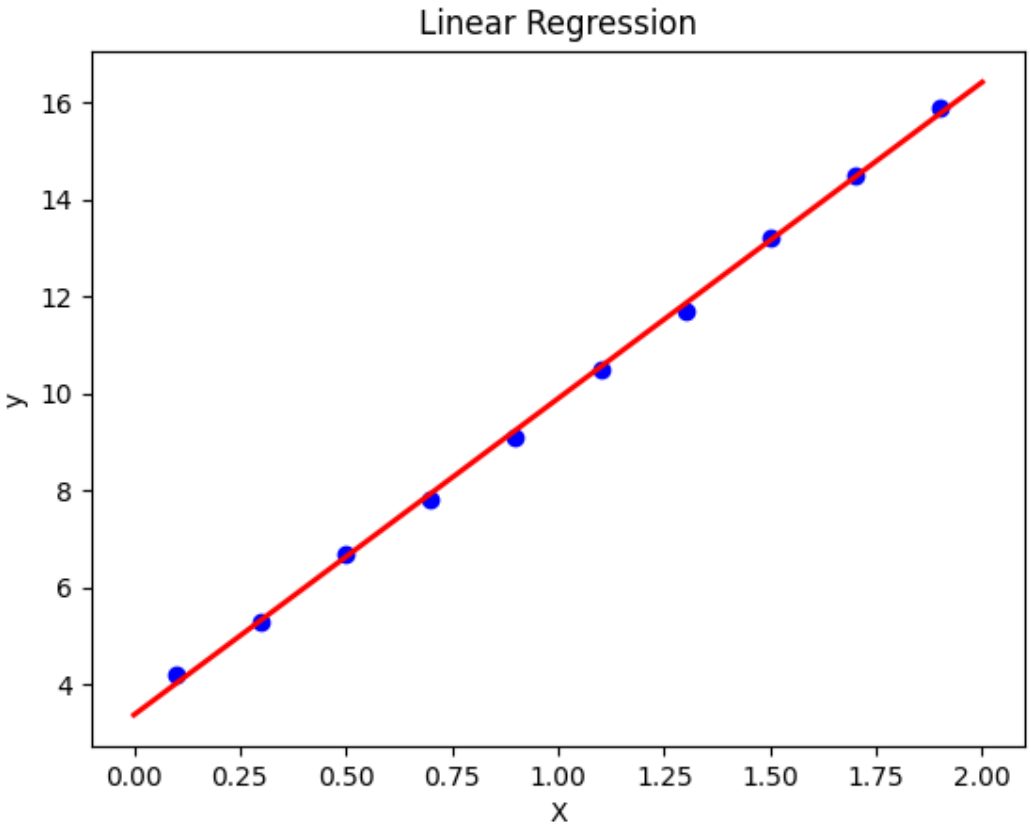
Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Load data from CSV file
data = pd.read_csv('simple1.csv')

# Extract X and y from the dataset
X = data['X_column'].values.reshape(-1, 1)  # Assuming 'X_column' is the column containing the feature
y = data['y_column'].values.reshape(-1, 1)  # Assuming 'y_column' is the column containing the target variable

# Create a scatter plot of the data
plt.scatter(X, y, color='blue')
plt.title('Data from CSV File')
plt.xlabel('X')
plt.ylabel('y')
plt.show()

# Train a linear regression model
model = LinearRegression()
model.fit(X, y)

# Make predictions
X_new = np.array([[0], [2]])  # New data points for prediction
y_pred = model.predict(X_new)

# Plot the linear regression line
plt.scatter(X, y, color='blue')
plt.plot(X_new, y_pred, color='red', linewidth=2)
```

```
plt.title('Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.show()

# Print the coefficients
print("Intercept:", model.intercept_[0])
print("Coefficient:", model.coef_[0][0])
```

Output :





Dataset:

X_column	y_column
0.1	4.2
0.3	5.3
0.5	6.7
0.7	7.8
0.9	9.1
1.1	10.5
1.3	11.7
1.5	13.2
1.7	14.5
1.9	15.9

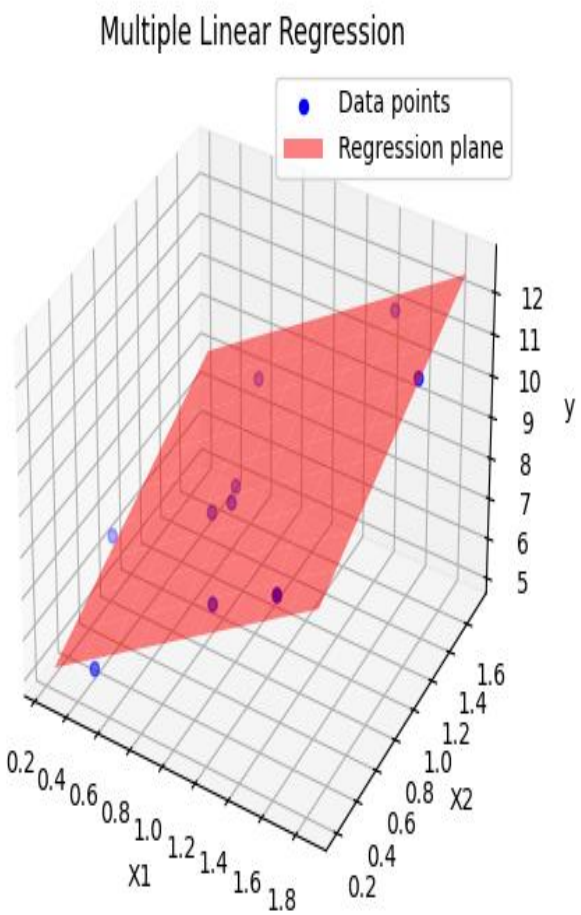
3) Implement Multivariate Linear Regression

Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from mpl_toolkits.mplot3d import Axes3D

# Load the data from a CSV file
file_path = 'book1.csv'  # Replace 'your_dataset.csv' with the actual file path
df = pd.read_csv(file_path)

# Extract features (X) and target variable (y)
X = df[['X1', 'X2']].values
y = df['y'].values

# Train a multiple linear regression model
model = LinearRegression()
model.fit(X, y)

# Print the coefficients
print("Intercept:", model.intercept_)
print("Coefficients:", model.coef_)

# Plot the data points and the plane predicted by the model
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111, projection='3d')

# Plot the data points
ax.scatter(X[:, 0], X[:, 1], y, color='blue', label='Data points')

# Plot the plane predicted by the model
```

```
x1, x2 = np.meshgrid(np.linspace(X[:, 0].min(), X[:, 0].max(), 10), np.linspace(X[:, 1].min(), X[:, 1].max(), 10))
y_plane = model.intercept_ + model.coef_[0] * x1 + model.coef_[1] * x2
ax.plot_surface(x1, x2, y_plane, alpha=0.5, color='red', label='Regression plane')

ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('y')
ax.set_title('Multiple Linear Regression')
ax.legend()

plt.show()
```

Output :



Dataset :

X1	X2	y
1.0976	0.4304	7.5301
1.8598	1.2068	11.6733
0.8504	0.8137	7.9842
1.4404	1.7152	11.0346
0.9646	0.8265	8.3872
1.6098	0.2212	9.4422
0.2515	0.7572	6.5067
0.8767	1.0103	8.0184
0.5107	0.2033	5.6365
0.8344	1.3094	9.6074

4) Implement Logistic Regression

Code :

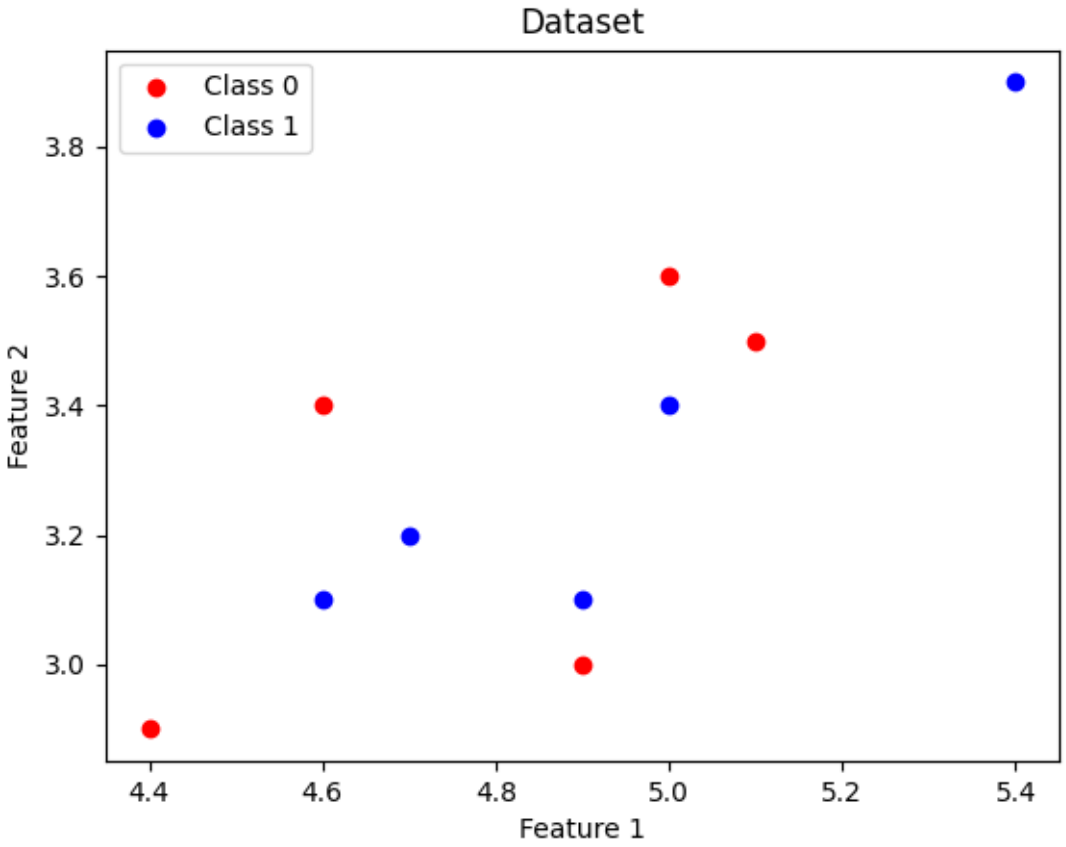
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
data = pd.read_csv('lgr2.csv')

# Display the first few rows of the dataset
print("Dataset:")
print(data.head())

# Separate features (X) and target variable (y)
X = data.drop(columns=['target_column'])
y = data['target_column']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the logistic regression model
model = LogisticRegression()

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

# Make predictions on the testing data
y_pred = model.predict(X_test)
```

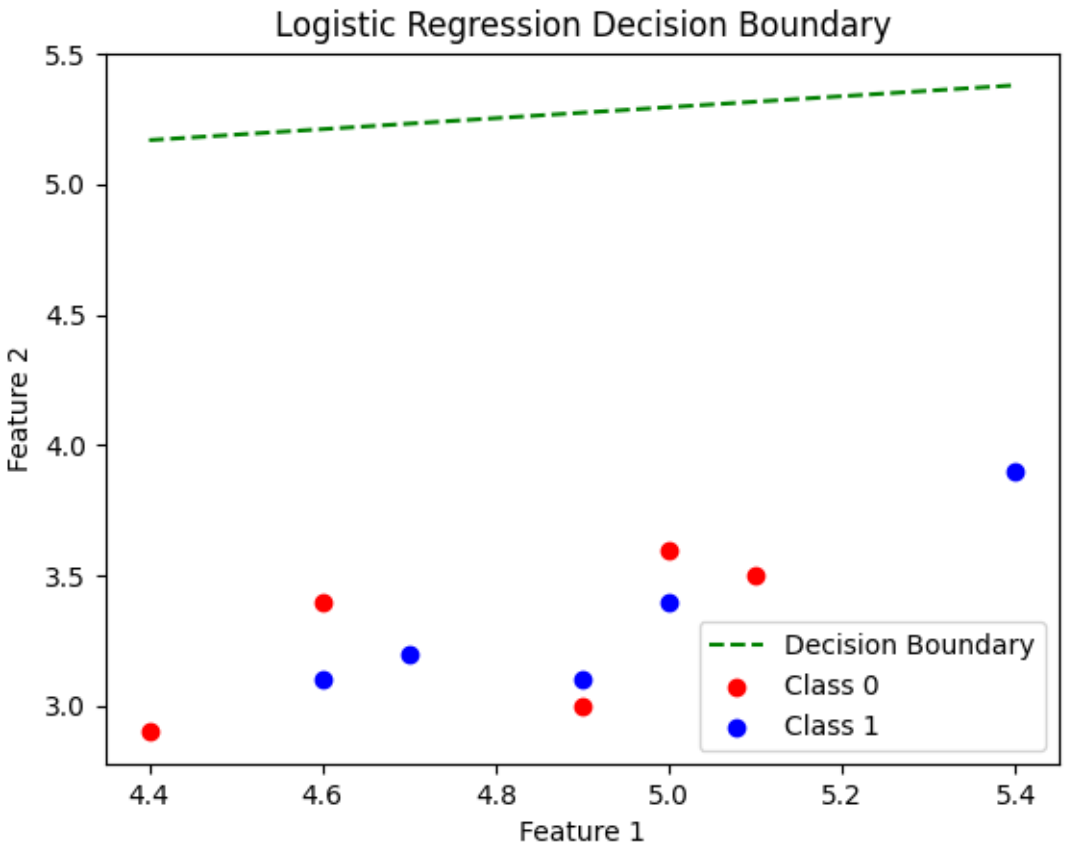
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Plot the dataset
plt.scatter(X[y == 0]['feature1'], X[y == 0]['feature2'], color='red', label='Class 0')
plt.scatter(X[y == 1]['feature1'], X[y == 1]['feature2'], color='blue', label='Class 1')
plt.title('Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()

# Plot the decision boundary
x_values = np.linspace(X['feature1'].min(), X['feature1'].max(), 100)
y_values = -(model.coef_[0][0] * x_values + model.intercept_[0]) / model.coef_[0][1]
plt.plot(x_values, y_values, color='green', linestyle='--', label='Decision Boundary')
plt.scatter(X[y == 0]['feature1'], X[y == 0]['feature2'], color='red', label='Class 0')
plt.scatter(X[y == 1]['feature1'], X[y == 1]['feature2'], color='blue', label='Class 1')
plt.title('Logistic Regression Decision Boundary')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output :

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2.0
1	0.00	0.00	0.00	0.0
accuracy			0.00	2.0
macro avg	0.00	0.00	0.00	2.0
weighted avg	0.00	0.00	0.00	2.0

```
Confusion Matrix:
[[0 2]
 [0 0]]
```




Logistic Regression Decision Boundary
5.5
5.0
4.5
4.0
3.5
3.0
Feature 2
4.4
4.6
4.8
5.0
5.2
5.4
Feature 1
Decision Boundary
Class 0
Class 1

5) Implement Multivariate Logistic Regression

Code :

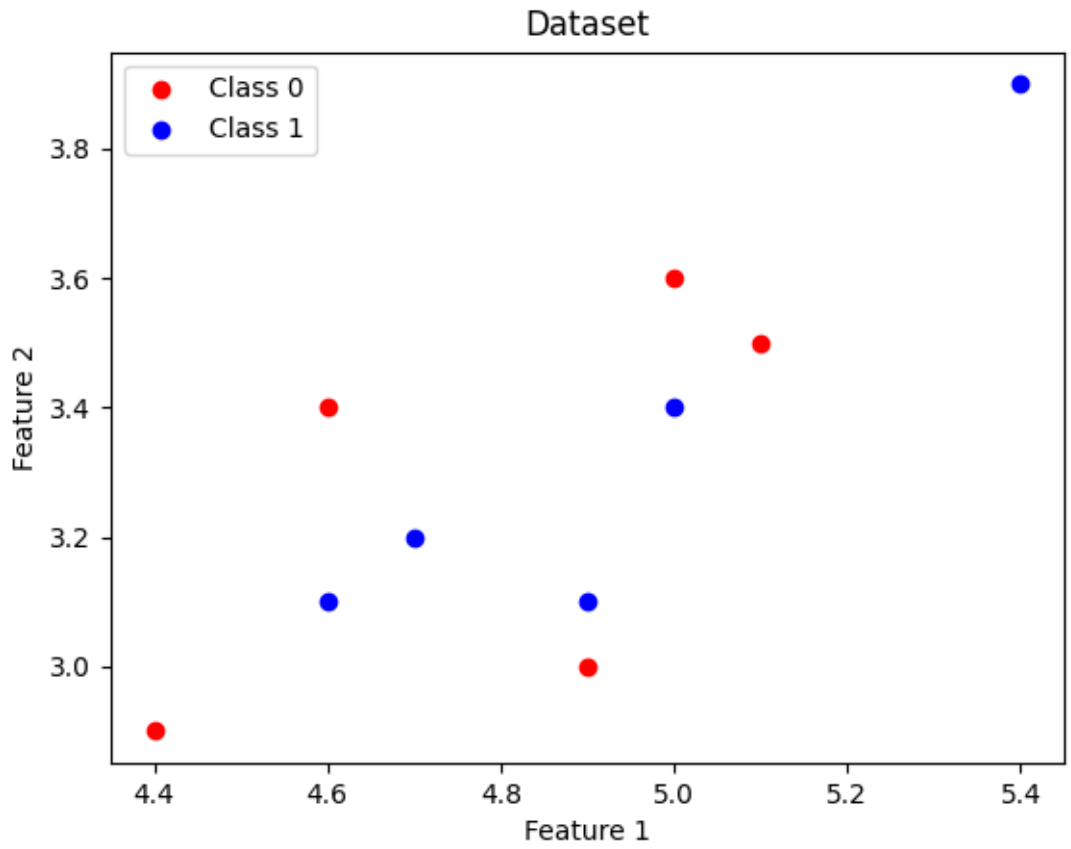
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset
data = pd.read_csv('lgr2.csv')

# Display the first few rows of the dataset
print("Dataset:")
print(data.head())

# Separate features (X) and target variable (y)
X = data.drop(columns=['target_column'])
y = data['target_column']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the logistic regression model
model = LogisticRegression()

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

# Make predictions on the testing data
y_pred = model.predict(X_test)
```

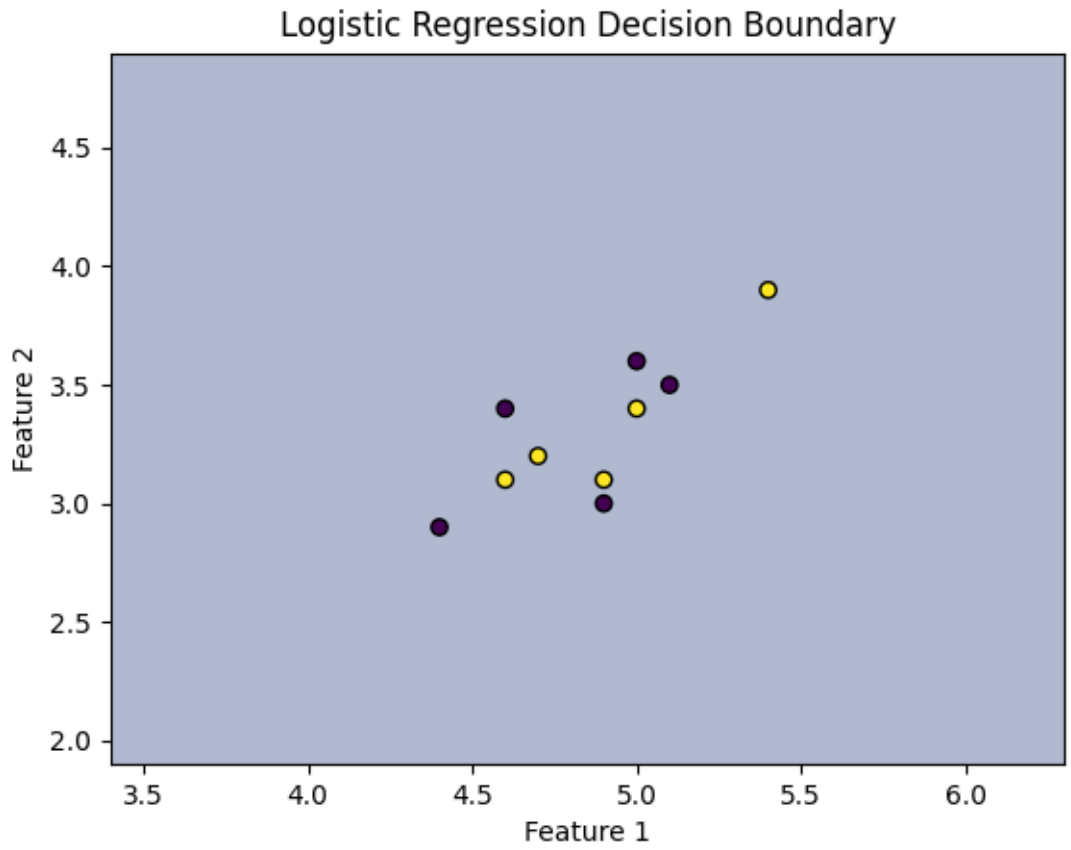
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", accuracy)

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Plot decision boundary for two features only
if X.shape[1] == 2:
    # Plot the dataset
    plt.scatter(X[y == 0].iloc[:, 0], X[y == 0].iloc[:, 1], color='red', label='Class 0')
    plt.scatter(X[y == 1].iloc[:, 0], X[y == 1].iloc[:, 1], color='blue', label='Class 1')
    plt.title('Dataset')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.legend()
    plt.show()

    # Plot the decision boundary
    x_min, x_max = X.iloc[:, 0].min() - 1, X.iloc[:, 0].max() + 1
    y_min, y_max = X.iloc[:, 1].min() - 1, X.iloc[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.4)
    plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c=y, marker='o', edgecolor='k')
    plt.title('Logistic Regression Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
```

plt.show()

Output :

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       2.0
           1       0.00      0.00      0.00       0.0

    accuracy                           0.00       2.0
   macro avg       0.00      0.00      0.00       2.0
weighted avg       0.00      0.00      0.00       2.0


Confusion Matrix:
[[0 2]
 [0 0]]
```





Dataset :

feature1	feature2	target_column
5.1	3.5	0
4.9	3	0
4.7	3.2	1
4.6	3.1	1
5	3.6	0
5.4	3.9	1
4.6	3.4	0
5	3.4	1
4.4	2.9	0
4.9	3.1	1

Note : ***Same for logistic and multivariant logistic***

6) Data preprocessing for classification

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

# Define the dataset
data = {
    'sepal_length': [5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9],
    'sepal_width': [3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1],
    'petal_length': [1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5],
    'petal_width': [0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1],
    'target': [0, 0, 0, 0, 1, 0, 0, 2, 0, 0]
}

# Convert the dictionary to a DataFrame
df = pd.DataFrame(data)

# Display the entire dataset
print("Original dataset:")
print(df.head())

# Extract features (X) and target (y)
X = df.drop('target', axis=1).values
y = df['target'].values

# Step 1: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 2: Data Transformation
```

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Step 3: Display scaled training data

print("\nScaled training data:")

print(pd.DataFrame(data=X_train_scaled, columns=df.columns[:-1]).head())


# Step 4: Display scaled test data

print("\nScaled test data:")

print(pd.DataFrame(data=X_test_scaled, columns=df.columns[:-1]).head())
```

Output :

```
Original dataset:
   sepal_length  sepal_width  petal_length  petal_width  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       1

Scaled training data:
   sepal_length  sepal_width  petal_length  petal_width
0      1.341641     0.937043     -0.408248     0.000000
1      0.819892     0.390434      1.020621     0.000000
2     -0.745356    -0.702782     -1.837117     0.000000
3      0.298142    -1.249390      1.020621    -1.870829
4      0.819892     1.483651     -0.408248     0.000000

Scaled test data:
   sepal_length  sepal_width  petal_length  petal_width
0     -2.310604    -2.342606     -0.408248     0.000000
1      0.298142    -1.795998     -0.408248     0.000000
2      2.906888     3.123475      3.878359     3.741657
```

7) Confusion matrix for a binary classifier.

Code :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Read the CSV file into a DataFrame
df = pd.read_csv('sample_data.csv')

# Split data into features (X) and labels (y)
X = df[['Feature']]  # Corrected column name to 'Feature'
y = df['Label']     # Corrected column name to 'Label'

# 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=42)

# Initialize and train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Calculate performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the confusion matrix and performance metrics
print("Confusion Matrix:")
print(cm)
print("\nAccuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

Output :

```
Confusion Matrix:
[[1 0]
 [0 1]]

Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

Dataset :

Feature	Label
1	0
2	0
3	0
4	0
5	1
6	1
7	1
8	1
9	1
10	1

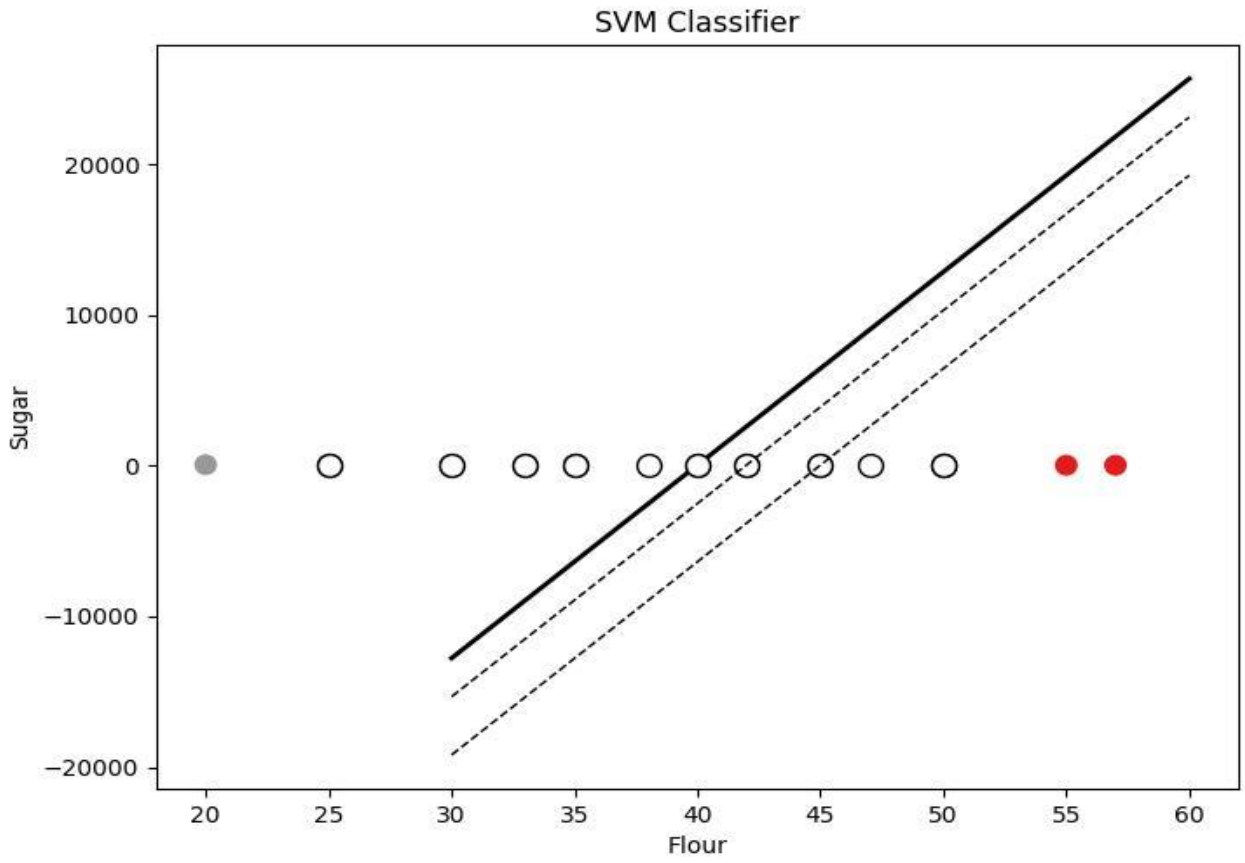
8) Implement Multivariate Logistic Regression

Code :

```
import pandas as pd
import numpy as np
from sklearn import svm
import matplotlib.pyplot as plt

# Read data from CSV file
recipes = pd.read_csv('svm1.csv')

# Extract features and labels
features = recipes[['Flour', 'Sugar']].to_numpy()
label = np.where(recipes['Type'] == 'Muffin', 0, 1)

# Train SVM model
model = svm.SVC(kernel='linear')
model.fit(features, label)

# Get the separating hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(30, 60)
yy = a * xx - (model.intercept_[0]) / w[1]

# Plot the parallels to the separating hyperplane that pass through the support vectors
b = model.support_vectors_
yy_down = a * xx + (b[0, 1] - a * b[0, 0])
yy_up = a * xx + (b[-1, 1] - a * b[-1, 0])

# Plot the hyperplane
plt.figure(figsize=(8, 6))
plt.scatter(features[:, 0], features[:, 1], c=label, cmap='Set1', s=70)
plt.plot(xx, yy, linewidth=2, color='black')
```

```
plt.plot(b[:, 0], b[:, 1], 'o', markerfacecolor='white', markeredgecolor='black', markersize=10)
plt.plot(xx, yy_down, 'k--', linewidth=1)
plt.plot(xx, yy_up, 'k--', linewidth=1)
plt.xlabel('Flour')
plt.ylabel('Sugar')
plt.title('SVM Classifier')
plt.show()
```

Output :



DATASET :

Flour	Sugar	Type
35	40	Cupcake
45	30	Muffin
30	50	Cupcake
50	20	Muffin
40	45	Cupcake
25	55	Muffin
38	42	Cupcake
47	33	Muffin
42	48	Cupcake
33	57	Muffin
55	25	Muffin
20	60	Cupcake
50	30	Muffin
30	40	Cupcake
45	35	Muffin
57	23	Muffin
35	45	Cupcake
40	30	Muffin
50	20	Cupcake
25	55	Muffin
33	58	Muffin
42	47	Cupcake

9) Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

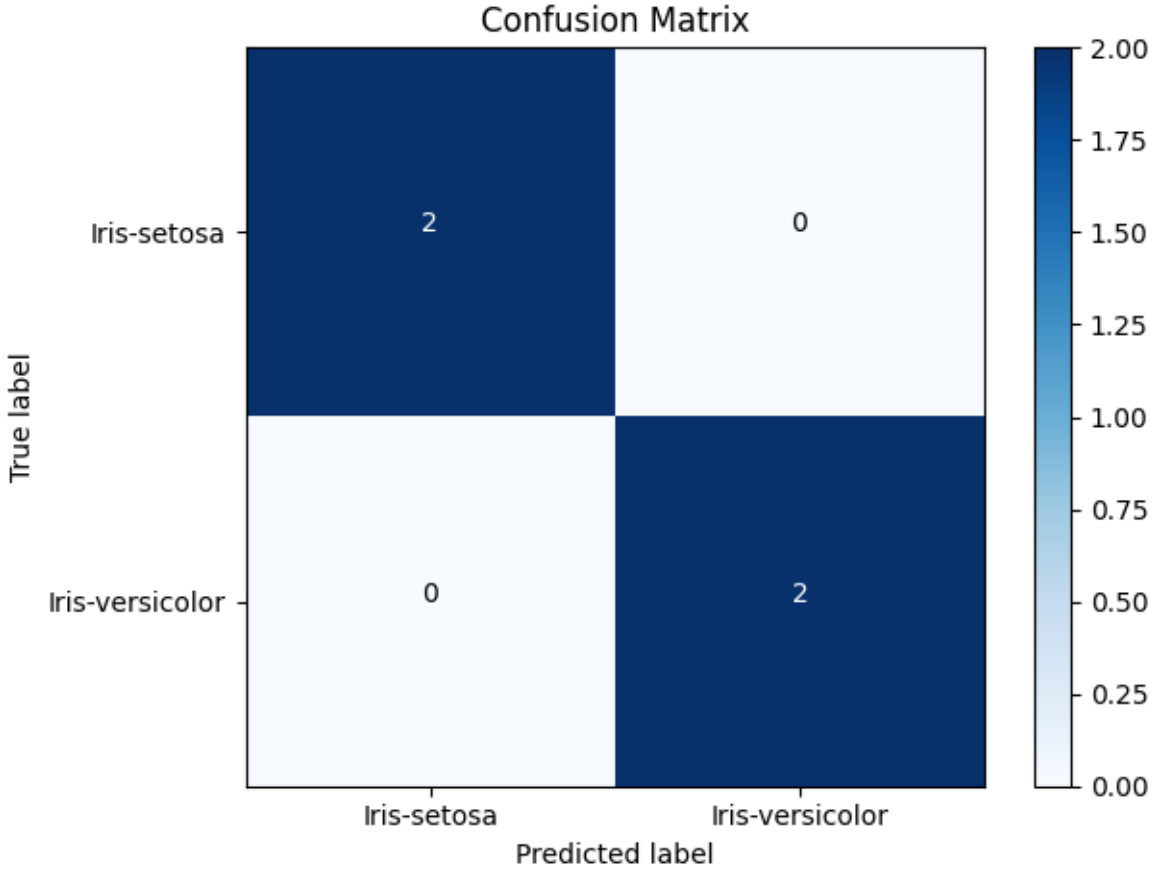
Code :

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix, classification_report

# Load the Iris dataset from CSV file
iris_df = pd.read_csv("iris_dataset.csv")

# Splitting the dataset into features (X) and target variable (y)
X = iris_df.drop(columns=['Species']).values
y = iris_df['Species'].values

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Training the Naive Bayes classifier
clf = GaussianNB()
clf.fit(X_train, y_train)

# Making predictions
y_pred = clf.predict(X_test)

# Generating confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
```

```
plt.colorbar()

classes = iris_df['Species'].unique()
tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)

thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, format(cm[i, j], 'd'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()

# Print classification report
print(classification_report(y_test, y_pred, target_names=classes))

plt.show()
```

Output :

```
========== RESTART: C:/Users/sanke/Downloads/ml lab 2/navie baies/9.py ========
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         2
Iris-versicolor       1.00      1.00      1.00         2

       accuracy                           1.00         4
      macro avg       1.00      1.00      1.00         4
   weighted avg       1.00      1.00      1.00         4
```



Dataset :

SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
4.6	3.1	1.5	0.2	Iris-setosa
5	3.6	1.4	0.2	Iris-setosa
5.4	3.9	1.7	0.4	Iris-setosa
4.6	3.4	1.4	0.3	Iris-setosa
5	3.4	1.5	0.2	Iris-setosa
4.4	2.9	1.4	0.2	Iris-setosa
4.9	3.1	1.5	0.1	Iris-setosa
5.4	3.7	1.5	0.2	Iris-setosa
4.8	3.4	1.6	0.2	Iris-setosa
4.8	3	1.4	0.1	Iris-setosa
4.3	3	1.1	0.1	Iris-setosa
5.8	4	1.2	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
5.5	2.3	4	1.3	Iris-versicolor
6.5	2.8	4.6	1.5	Iris-versicolor

10) Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

Code :

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import
train_test_splitfrom sklearn.preprocessing import
StandardScaler from sklearn.neighbors import
KNeighborsClassifier import pandas as pd

# Load data from CSV file
df = pd.read_csv('iris_data.csv')

# Convert data to numpy arrays
X = df[['Petal length', 'Petal
width']].valuesy = df['Target'].values

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train =
scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define KNN classifier
classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
classifier.fit(X_train, y_train)

# Predict on the test set
y_pred = classifier.predict(X_test)
```

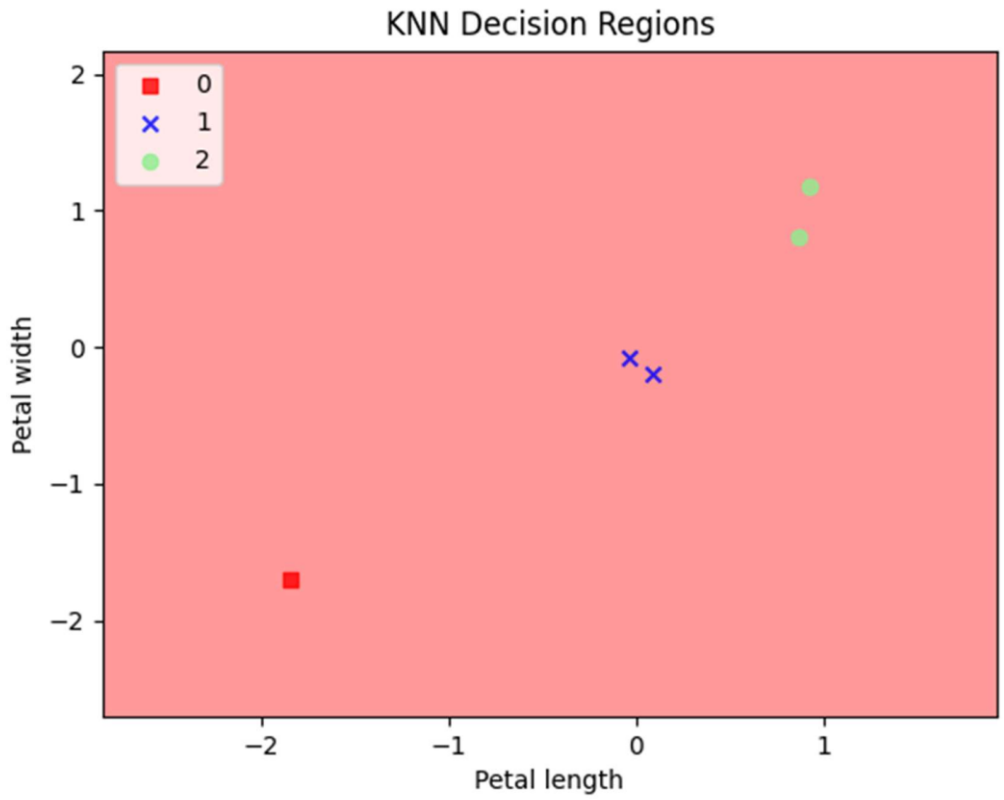
```
# Calculate accuracy
accuracy = np.mean(y_pred ==
y_test)print(f"Accuracy:
{accuracy:.2f}")

# Plot decision regions
def plot_decision_regions(X, y, classifier,
    resolution=0.02):markers = ('s', 'x', 'o')
    colors = ('red', 'blue', 'lightgreen')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(),
    xx2.ravel()]).T)Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4,
    cmap=cmap)plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in
        enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y ==
        cl, 1],
                alpha=0.8, c=[cmap(idx)], marker=markers[idx], label=cl)

# Plot decision regions
plot_decision_regions(X_train, y_train,
classifier)plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.legend(loc='upper left')
plt.title('KNN Decision
Regions')plt.show()
```

Dataset:

Petal length	Petal width	Target
1.4	0.2	0
1.3	0.2	0
1.5	0.2	0
4.7	1.4	1
4.5	1.5	1
4.9	1.5	1
6.1	2.5	2
6	2.2	2

Output:

Accuracy: 0.33



11) 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.

Code :

```
import pandas as pd
from pgmpy.models import BayesianNetwork
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.inference import VariableElimination

# Load Heart Disease Data Set
data = pd.read_csv('heart_disease.csv')

# Display the data
print(data)

# Define the Bayesian Network structure
model = BayesianNetwork([('age', 'heartdisease'), ('sex', 'heartdisease'), ('cp', 'heartdisease'),
                         ('restecg', 'heartdisease'), ('exang', 'heartdisease'), ('chol', 'heartdisease')])

# Train the model with Maximum Likelihood Estimator
model.fit(data, estimator=MaximumLikelihoodEstimator)

# Perform inference using Variable Elimination
infer = VariableElimination(model)

# Predicting heart disease
print('\n\nPredicting for Heart Disease:')
query = infer.query(variables=['heartdisease'], evidence={'age': 73, 'sex': 0, 'cp': 1, 'restecg': 2, 'exang': 1})
print(query)

# Predicting cholesterol level given heart disease
print('\n\nPredicting for Cholesterol level:')
query = infer.query(variables=['chol'], evidence={'heartdisease': 1})
print(query)
```

Output :

```
============ RESTART: C:\Users\sanke\Downloads\ML 1415\heart\hv1.py ============
    age  sex  cp  restecg  exang  chol  heartdisease
0    73    0   1        2      1   551             0
1    76    1   4        0      1   183             0
2    29    0   2        1      0   417             1
3    32    1   3        1      0   484             0
4    32    1   4        1      0   245             0
5    68    0   4        0      1   393             0
6    38    0   1        2      1   556             1
7    48    1   3        0      0   208             1
8    50    1   4        2      1   217             0
9    65    1   1        2      0   510             1
10   52    1   2        0      0   524             0
11   35    0   4        2      1   225             0
12   53    1   2        0      0   179             0
13   53    0   4        0      1   522             0
14   41    1   4        0      1   247             0


Predicting for Heart Disease:
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.5333 |
+-----------------+---------------------+
| heartdisease(1) |              0.4667 |
+-----------------+---------------------+


Predicting for Cholesterol level:
+-----------+-------------+
| chol      |   phi(chol) |
+===========+=============+
| chol(179) |      0.0666 |
+-----------+-------------+
| chol(183) |      0.0665 |
+-----------+-------------+
| chol(208) |      0.0669 |
+-----------+-------------+
| chol(217) |      0.0666 |
+-----------+-------------+
| chol(225) |      0.0667 |
+-----------+-------------+
| chol(245) |      0.0666 |
+-----------+-------------+
| chol(247) |      0.0665 |
+-----------+-------------+
| chol(393) |      0.0666 |
+-----------+-------------+
| chol(417) |      0.0668 |
+-----------+-------------+
| chol(484) |      0.0667 |
+-----------+-------------+
| chol(510) |      0.0669 |
+-----------+-------------+
| chol(522) |      0.0664 |
+-----------+-------------+
| chol(524) |      0.0667 |
+-----------+-------------+
| chol(551) |      0.0667 |
+-----------+-------------+
| chol(556) |      0.0669 |
+-----------+-------------+
```

Dataset :

age	sex	cp	restecg	exang	chol	heartdisease
73	0	1	2	1	551	0
76	1	4	0	1	183	0
29	0	2	1	0	417	1
32	1	3	1	0	484	0
32	1	4	1	0	245	0
68	0	4	0	1	393	0
38	0	1	2	1	556	1
48	1	3	0	0	208	1
50	1	4	2	1	217	0
65	1	1	2	0	510	1
52	1	2	0	0	524	0
35	0	4	2	1	225	0
53	1	2	0	0	179	0
53	0	4	0	1	522	0
41	1	4	0	1	247	0
28	1	3	0	1	511	1

12) Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Code :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

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

# Convert categorical variables into numerical using one-hot encoding
data = pd.get_dummies(data)

# Split data into features and target
X = data.drop(columns=['target_yes', 'target_no'])  # Drop one of the target columns
y = data['target_yes']  # Use one of the target columns as target variable

# 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=42)

# Create decision tree classifier
clf = DecisionTreeClassifier()

# Train decision tree classifier
clf.fit(X_train, y_train)

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

# Calculate accuracy
```

```
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)


# Now, let's classify a new sample

# You need to encode the new sample using the same one-hot encoding technique

# For simplicity, let's assume the new sample is: age_group=<25, gender=M

new_sample = pd.DataFrame({'age_group_<25': [1], 'age_group_25-40': [0], 'age_group_>40': [0],

                           'gender_F': [0], 'gender_M': [1]})

prediction = clf.predict(new_sample)

print("Predicted class for new sample:", prediction[0])
```

Dataset :

age_group	gender	target
<25	M	yes
<25	F	no
25-40	M	yes
>40	F	yes
25-40	F	no
>40	M	no

Output :

```
============== RESTART: C:\Users\sanke\Downloads\ML 1415\12\12.py ==============
Accuracy: 0.5
Predicted class for new sample: False
'
```

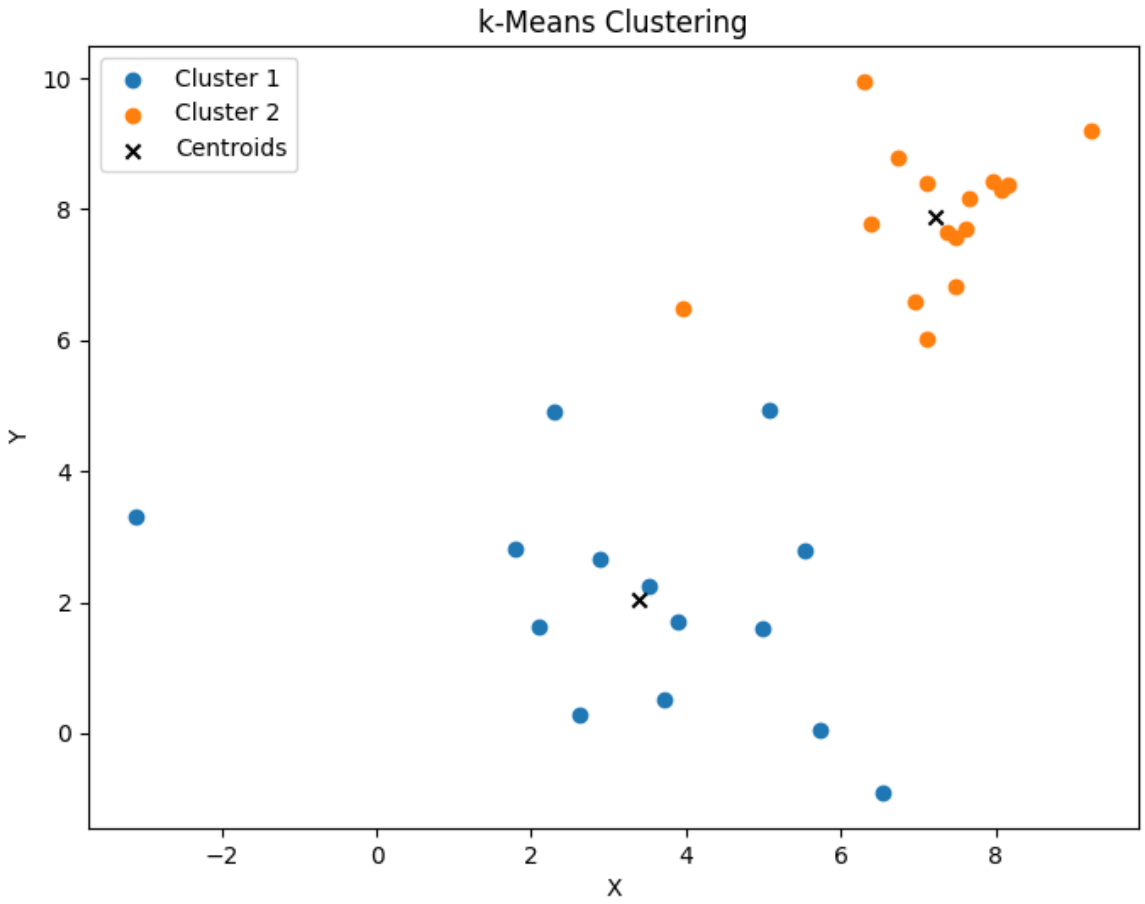
13) Implement k-Mean's algorithm to cluster a set of data stored in a .CSV file.

Code :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def kmeans(data, k, max_iter=100):
    # Initialize centroids randomly
    centroids = data[np.random.choice(data.shape[0], k, replace=False)]

    for _ in range(max_iter):
        # Assign each data point to the nearest centroid
        labels = np.argmin(((data - centroids[:, np.newaxis])**2).sum(axis=2), axis=0)

        # Update centroids
        new_centroids = np.array([data[labels == i].mean(axis=0) for i in range(k)])

        # Check for convergence
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return labels, centroids

def plot_clusters(data, labels, centroids):
    plt.figure(figsize=(8, 6))
    for i in range(k):
        plt.scatter(data[labels == i][:, 0], data[labels == i][:, 1], label=f'Cluster {i+1}')
    plt.scatter(centroids[:, 0], centroids[:, 1], color='black', marker='x', label='Centroids')
    plt.title('k-Means Clustering')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
```

```
    plt.show()

# Load data from CSV
df = pd.read_csv('sample_data.csv')
data = df.values

# Specify the number of clusters (k) and maximum iterations
k = 2
max_iter = 100

# Perform k-Means clustering
labels, centroids = kmeans(data, k, max_iter)

# Plot the clustered data
plot_clusters(data, labels, centroids)

# Output the clusters and final centroids
print("Clusters:", labels)
print("Final Centroids:")
for i, centroid in enumerate(centroids):
    print(f"Centroid {i+1}: {centroid}")
```

Output :



Dataset :

x	y
5.528105	2.800314
3.957476	6.481786
5.735116	0.045444
3.900177	1.697286
1.793562	2.821197
2.288087	4.908547
3.522075	2.24335
2.887726	2.667349
4.988158	1.589683
2.626135	0.291809
-3.10598	3.307237
3.728872	0.51567
6.539509	-0.90873
2.091517	1.625632
5.065558	4.938718
8.154947	8.378163
7.112214	6.019204
7.652088	8.156349

9.230291	9.20238
7.612673	7.697697
6.951447	6.579982
6.29373	9.950775
7.490348	7.561926
6.747205	8.77749
6.386102	7.78726
7.104533	8.386902
7.489195	6.819368
7.971818	8.428332
8.066517	8.302472
7.365678	7.637259

14) Implement Random Forest models for automatic classification

Code :

```
import pandas as pd

# Load dataset from CSV file
dataset_path = 'your_dataset.csv'  # Provide the path to your CSV file
df = pd.read_csv(dataset_path)

# Separate features (X) and labels (y)
X = df.drop(columns=['label']).values
y = df['label'].values

# Splitting the data into training and testing sets (if needed)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initializing the Random Forest classifier
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Training the classifier on the training data
rf_classifier.fit(X_train, y_train)

# Making predictions on the testing data
y_pred = rf_classifier.predict(X_test)

# Evaluating the performance of the classifier
from sklearn.metrics import accuracy_score, classification_report
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Output :

```
============== RESTART: C:\Users\sanke\Downloads\ML 1415\14\14.py ==============
Accuracy: 0.3333333333333333
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       0.50      0.50      0.50         2

    accuracy                           0.33         3
   macro avg       0.25      0.25      0.25         3
weighted avg       0.33      0.33      0.33         3
```

Dataset :

feature1	feature2	feature3	feature4	label
0.37454	0.950714	0.731994	0.598658	1
0.156019	0.155995	0.058084	0.866176	0
0.601115	0.708073	0.020584	0.96991	1
0.832443	0.212339	0.181825	0.183405	0
0.304242	0.524756	0.431945	0.291229	0
0.611853	0.139494	0.292145	0.366362	1
0.45607	0.785176	0.199674	0.514234	1
0.592415	0.04645	0.607545	0.170524	0
0.065052	0.948886	0.965632	0.808397	1
0.304614	0.097672	0.684233	0.440152	1
0.122038	0.495177	0.034389	0.90932	1
0.25878	0.662522	0.311711	0.520068	0
0.54671	0.184854	0.969585	0.775133	0
0.939499	0.894827	0.5979	0.921874	0
0.088493	0.195983	0.045227	0.32533	0

15) Implement Ensemble Model to perform classification.

Code :

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the dataset from CSV
df = pd.read_csv('your_dataset.csv')

# Split the data into features and target variable
X = df.drop(columns=['target'])
y = df['target']

# Split the 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=42)

# Create base classifier
base_classifier = DecisionTreeClassifier()

# Number of base classifiers in the ensemble
num_classifiers = 10

# Create an ensemble of decision tree classifiers using bagging
ensemble_model = BaggingClassifier(base_classifier, n_estimators=num_classifiers)

# Train the ensemble model
```

```
ensemble_model.fit(X_train, y_train)

# Make predictions
predictions = ensemble_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print("Accuracy:", accuracy)
```

Output :

Accuracy: 1.0

Dataset :

feature_0	feature_1	feature_2	feature_3	feature_4	target
0.469244	-0.34377	-0.57537	-0.4535	1.057122	1
0.663319	0.24926	-0.19301	-0.49712	0.93128	1
-0.59486	0.877553	1.102127	0.661401	-1.76304	1
-1.09222	-1.91264	-0.94968	0.524415	-0.3011	0
0.788035	0.586233	0.015481	-0.53378	1.031	1
0.792025	-0.82457	-1.17731	-0.8133	-0.46064	1
-1.19643	-0.99824	-0.11479	0.789223	-0.71984	0
-1.05812	-0.71246	0.042236	0.731349	-0.11565	0
0.439152	0.1795	-0.11557	-0.32628	-0.83922	1
-0.96971	-0.90929	-0.17759	0.620051	-1.47852	0
-0.09928	0.84147	0.770357	0.246472	0.611676	0
-0.14628	1.210004	1.109883	0.357311	-0.38508	0
-1.23872	1.410767	1.943516	1.295712	0.324084	0
-0.79887	0.426977	0.846001	0.741084	-0.67692	0
0.576179	-0.30228	-0.60538	-0.53339	0.343618	1

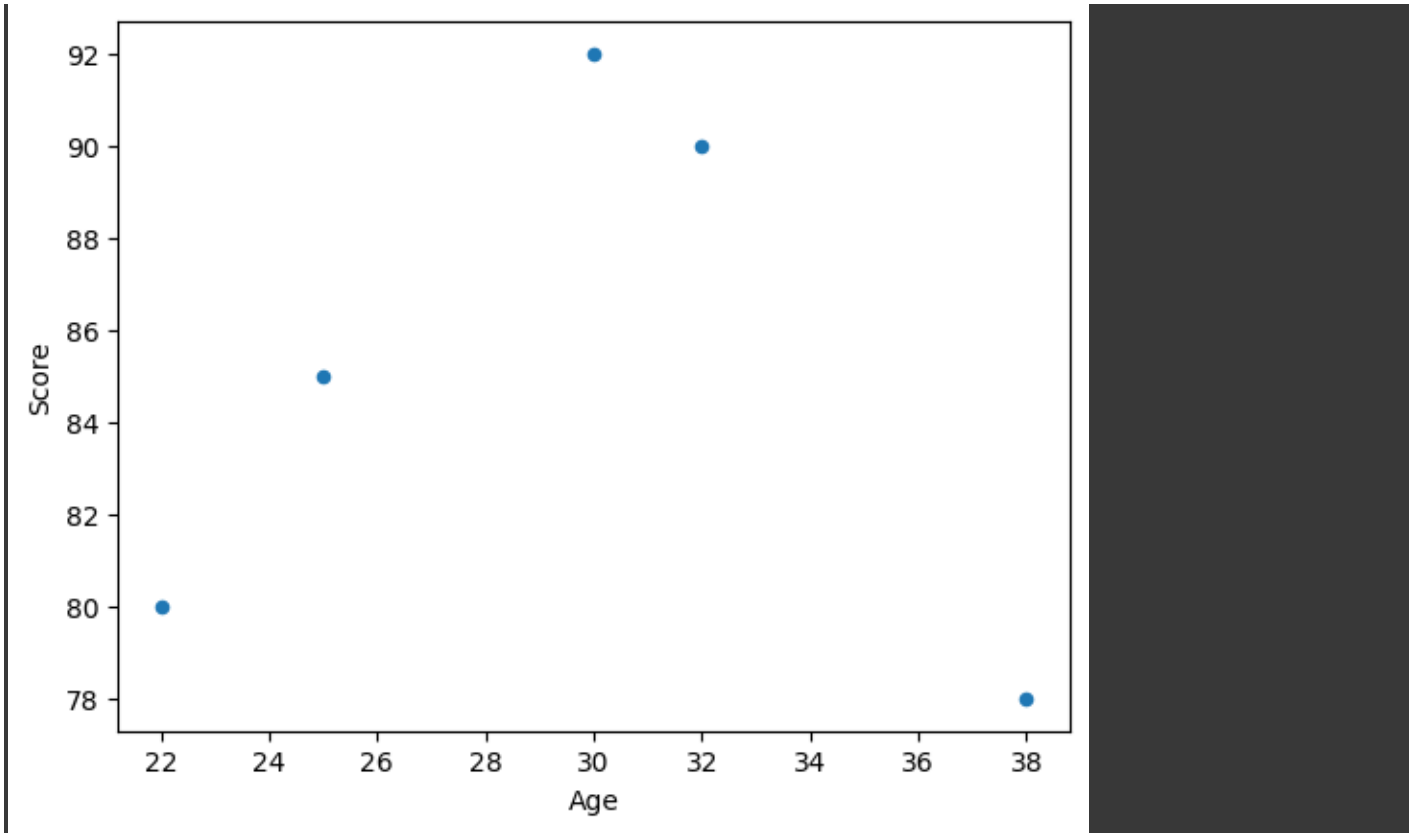
Code :

```
import pandas as pd


from google.colab import drive

import pandas as pd


# Mount your Google Drive

drive.mount('/content/drive')


# Specify the file path within your Drive

file_path = '/content/drive/MyDrive/data1.csv'  # Replace with your actual path


# Read the CSV file into a DataFrame

df = pd.read_csv(file_path)


# (a) Convert DataFrame to array and display

array = df.to_numpy()

print("(a) Array:\n", array)


# (b) Display first 5 rows and last 3 columns

print("(b) First 5 rows, last 3 columns:\n", df.iloc[:5, -3:])


# (c) Describe the DataFrame

print("(c) DataFrame description:\n", df.describe())


# (d) Replace missing values with mean or duplicate values

# Assuming 'Score' column has missing values

df['Score'].fillna(df['Score'].mean(), inplace=True)  # Replace with mean

# Or

# df['Score'].fillna(df['Score'].mode()[0], inplace=True)  # Replace with mode (most frequent)


print("(d) DataFrame after filling missing values:\n", df)


# (e) Plot a graph with two features
```

```
df.plot(x='Age', y='Score', kind='scatter')


# (f) Display the header
print("(f) Header:\n", df.columns)
```

Output

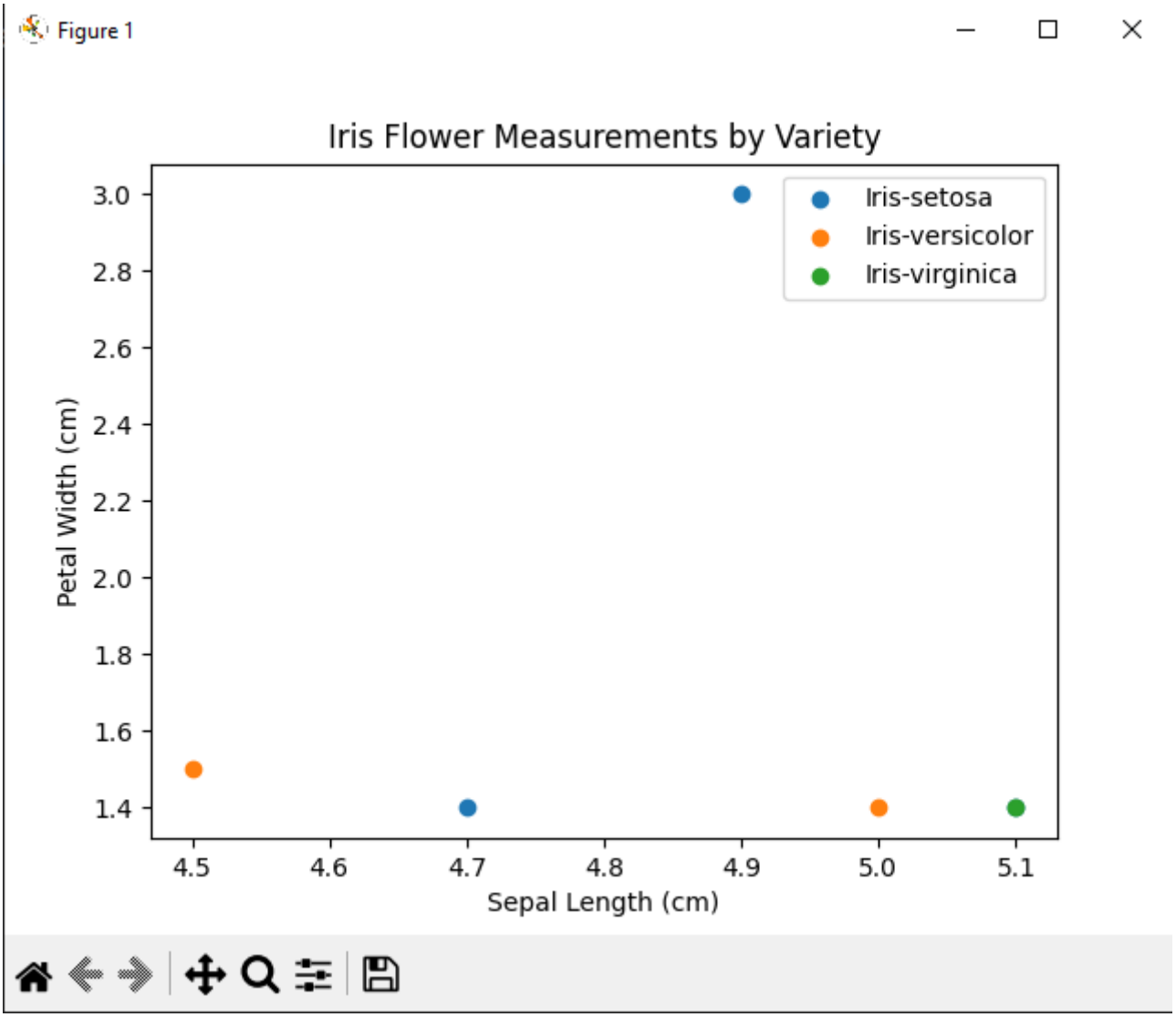
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
(a) Array:
 [['Alice' 25 'New York' 85]
 ['Bob' 30 'Los Angeles' 92]
 ['Charlie' 38 'Chicago' 78]
 ['David' 32 'Houston' 90]
 ['Eve' 22 'San Francisco' 80]]
(b) First 5 rows, last 3 columns:
    Age           City  Score
0   25       New York     85
1   30    Los Angeles     92
2   38        Chicago     78
3   32        Houston     90
4   22  San Francisco     80
(c) DataFrame description:
              Age      Score
count   5.000000   5.000000
mean   29.400000  85.000000
std     6.228965   6.082763
min    22.000000  78.000000
25%    25.000000  80.000000
50%    30.000000  85.000000
75%    32.000000  90.000000
max    38.000000  92.000000
(d) DataFrame after filling missing values:
       Name  Age           City  Score
0    Alice   25       New York     85
1      Bob   30    Los Angeles     92
2  Charlie   38        Chicago     78
3    David   32        Houston     90
4      Eve   22  San Francisco     80
(f) Header:
 Index(['Name', 'Age', 'City', 'Score'], dtype='object')
```


92
90
88
86
84
82
80
78
Score
22
24
26
28
30
32
34
36
38
Age

Program

```
import pandas as pd

import matplotlib.pyplot as plt


# Sample data (replace with your actual CSV file path)

file_path = "iris1.csv"  # Replace with your actual file path


# Read the CSV file into a DataFrame

df = pd.read_csv(file_path)# Read the CSV file into a DataFrame




# Extract unique varieties

species = df["Variety"].unique()


# Create scatter plot for each variety

for i in species:

    subset = df.loc[df["Variety"] == i]

    plt.scatter(subset["sepal.length"], subset["petal.width"], label=i)


# Add labels and title

plt.xlabel("Sepal Length (cm)")

plt.ylabel("Petal Width (cm)")

plt.title("Iris Flower Measurements by Variety")


# Display legend

plt.legend()


# Show the plot

plt.show()
```

Output



dataset

sepal.length	petal.width	Variety
5.1	1.4	Iris-setosa
4.9	3	Iris-setosa
4.7	1.4	Iris-setosa
5	1.4	Iris-versicolor
4.5	1.5	Iris-versicolor
5.1	1.4	Iris-virginica