Name : Anshuman Kumar Sharma
Roll No. : 21419CMP007
Subject : Machine Learning
Semester : MSc. Semester - III (2021 - 2023)
Under the Guidance of : Dr. S. Suresh

Assignment 0: Python Tutorials

(a) Count the number of words in a text file.

Steps:

1) Load the required Libraries
2) Read the text file as input
3) Run a loop with counter i for each sentence in the file.
4) Increment the counter variable for each word detected.
5) At Last, print the final value of count once the final line of text file is parsed. (EOF)
6) Output: Total no. of words = count.

(b) Elementwise Matrix-Vector Multiplication and Calculation of Mean, Standard Deviation and Histogram of resultant vector.

Steps:

1) Load the required Libraries
2) Take input for dimensions of matrix Z (say x * y)
3) Create matrix Z and initialize it with random values
4) Create vector V of dimension y x 1 with normal distribution using, $\mu = 2$ and standard diviation, $\sigma = 0.01$.
5) Calculate element wise product for each row of matrix Z wrt vector V and store the value in corresponding row of vector C (Resultant vector).
6) Calculate the mean and standard deviation of vector (using inbuilt functions).
7) $\text{Mean} = \sum_{i=1}^{n} c[i]/n$ where $n = \text{len}(c)$

$$\text{standard deviation} = \sqrt{\frac{\Sigma (ci - \text{mean})^2}{n}}$$

8) Plot the histogram for vector C with bin value = 5.

Exercise 0: Python Tutorials (you may use other programming language/tools of your choice)

a) In this task you have to write a word count program (using IPython). Your program should read a text document. You should save your session and it should include Headings and comments at some important steps to explain the working of code.

In [ ]:
```
#Input: Required File
file="text.txt"

wordCount=0
#Counting Words
with open(file,'r') as file:
    for line in file:
        #Increment the wordCount variable by 1 for each word detected
        wordCount=(wordCount + len(line.split()))

#Output: Word Count
print("There are total ",wordCount," Number of Words in the given file.")
```

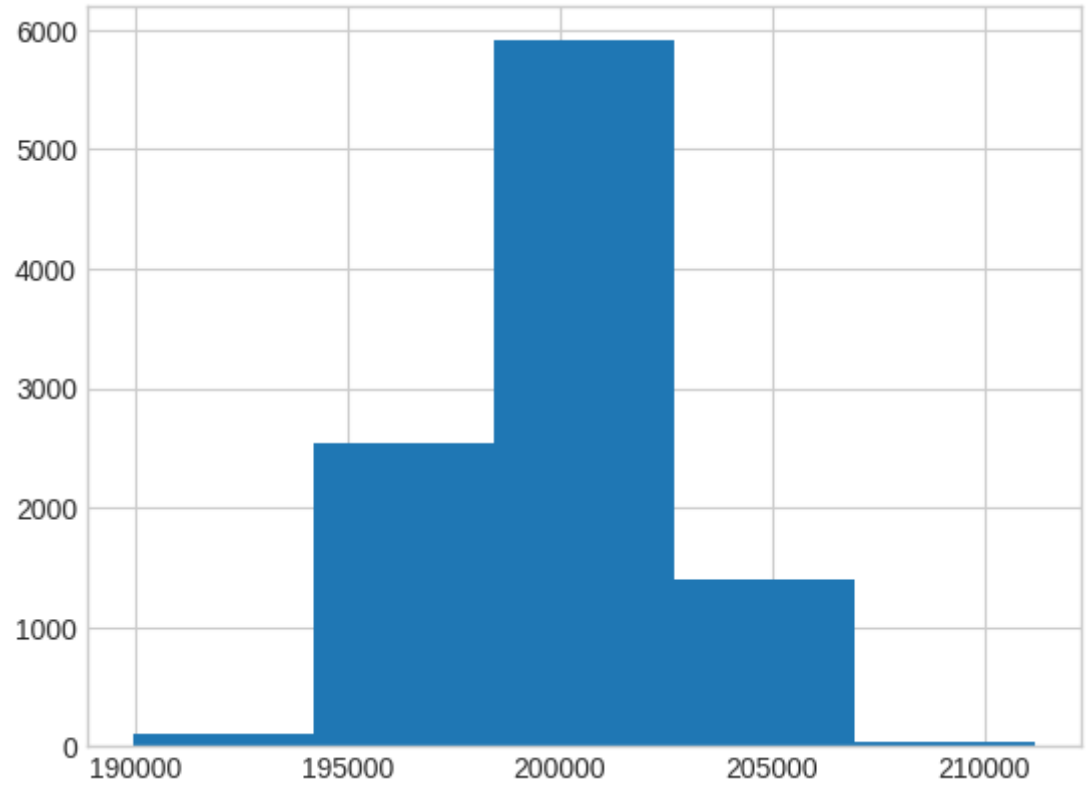
```
There are total  91  Number of Words in the given file.
```

b) Create a matrix A of dimensions n × m, where n = 100 and m = 20. Initialize Matrix A. Create a vector v of dimension m × 1. Initialize the matrix with a random values and vector with normal distribution using μ = 2 and σ = 0.01 (use numpy).

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

#Input: Number of Rows and Columns of Matrix
rows = int(input("Enter the number of rows: "))
cols = int(input("\nEnter the number of columns: "))

A = np.random.randint(low = 1, high = 100, size = (rows,cols))
V = np.random.normal(loc = 2.0, scale = 0.01, size = (cols,1))

#Storing Elementwise Multiplication of Matrix A and Vector V in Vector C
C = A.dot(V)

#Output: The Resultant Vector, its Mean, Standard Deviation and the Histo
print("\nThe Resultant Vector is:\n",C,"\n")
print("The Mean of the Resultant Vector = ",np.mean(C))
print("\nThe Standard Deviation of the Resultant Vector = ",np.std(C))
plt.style.use('seaborn-whitegrid')
print("\nThe Histogram for the Vector C is as follow:\n", plt.hist(C, bin
plt.show()
```

```
The Resultant Vector is:
 [[201981.98028119]
 [198540.91552939]
 [203819.08276216]
 ...
 [200600.87322011]
 [201552.08421756]
 [198354.38651345]]

The Mean of the Resultant Vector =  200024.91272434662

The Standard Deviation of the Resultant Vector =  2559.281525471748

The Histogram for the Vector C is as follow:
 (array([ 111., 2545., 5912., 1397.,   35.]), array([189943.29838214, 19
4198.75336162, 198454.2083411 , 202709.66332058,
       206965.11830006, 211220.57327954]), <BarContainer object of 5 art
ists>)
```

```
/tmp/ipykernel_4850/1600338010.py:18: MatplotlibDeprecationWarning: The
seaborn styles shipped by Matplotlib are deprecated since 3.6, as they n
o longer correspond to the styles shipped by seaborn. However, they will
remain available as 'seaborn-v0_8-<style>'. Alternatively, directly use
the seaborn API instead.
  plt.style.use('seaborn-whitegrid')
```



Assignment 1: Linear Regression

1(a): Linear Regression with Least Square Method.

Steps:

1) Load the required libraries
2) Import the dataset.
3) Preprocess the dataset (Remove null or missing values)
4) Seperate the dataset into dependent and independent variables, Y and Z respectively
5) Split the dataset into train and test data with your desired ratio
6) Manual Method
   a) Calculate the mean value of test and train dataset
   b) Calculate the least square.

```
for each row of train data
    num += (X_train[i] - X_train_mean) *
           (Y_train[i] - Y_tain_mean)
    den += (X_train[i] - X_train_mean)^2
```

   c) Calculate the slope and intersection

```
m = num/den
c = Y_train_mean - (m * X_train_mean)
```

   d) Predict the value of Y for each X of test dataset as

```
Y_pred = m * X_test + C
```

   e) Calculate the RMS and R2 score taking predicted and actual values of Y as parameter
7) Scikit-learn method:
   a) Load, Train and predict using inbuilt functions
   b) Calculate RMS and R2 score
8) Compare the results of Both Methods
9) Plot the line of Regression for both methods saperately.

Exercise 1: Linear Regression

a) Implement the linear regression using least square method in your own code. Also implement the linear regression by using existing library (scikit-learn). Compare the performance of both implementations.

Manual Method

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import preprocessing, svm, datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score

# Input: Data Set
dataSet = pd.read_csv('headbrain.csv')

# Taking only two attributes of the Dataset
data_binary = dataSet[['Head Size(cm^3)','Brain Weight(grams)']]
data_binary.columns = ['Head Size','Brain Weight']

# Eliminating NaN or missing input numbers
data_binary.fillna(method ='ffill', inplace = True)

# Dropping any rows with Nan values
data_binary.dropna(inplace = True)

# Separating the dataSet into independent and dependent variables
# Converting each dataframe into a numpy array
X = np.array(data_binary['Head Size']).reshape(-1, 1)
Y = np.array(data_binary['Brain Weight']).reshape(-1, 1)

# Splitting the dataSet into training and testing dataSet
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2

# Mean X and Y
mean_x_train = np.mean(X_train)
mean_y_train = np.mean(Y_train)
mean_x_test = np.mean(X_test)
mean_y_test = np.mean(Y_test)

# Total number of values
n = len(X_train)

# Building the Model
num = 0
den = 0

# Using the Least Square Method to calculate 'm' and 'c'
for i in range(n):
    num += (X_train[i] - mean_x_train) * (Y_train[i] - mean_y_train)
    den += (X_train[i] - mean_x_train) ** 2
```

```
m1 = num / den
c1 = mean_y_train - (m1 * mean_x_train)

Y_pred1 = c1 + m1 * X_test

# Calculating Root Mean Squares Error & R2 Score
rmse = 0
ss_tot = 0
ss_res = 0
for i in range(len(Y_test)):
    y_pred1 = c1 + m1 * X_test[i]
    rmse += (Y_test[i] - y_pred1) ** 2
    ss_tot += (Y_test[i] - mean_y_test) ** 2
    ss_res += (Y_test[i] - y_pred1) ** 2

rmse = np.sqrt(rmse/len(Y_test))
r2 = 1 - (ss_res/ss_tot)
```

(SciKit-Learn) Method

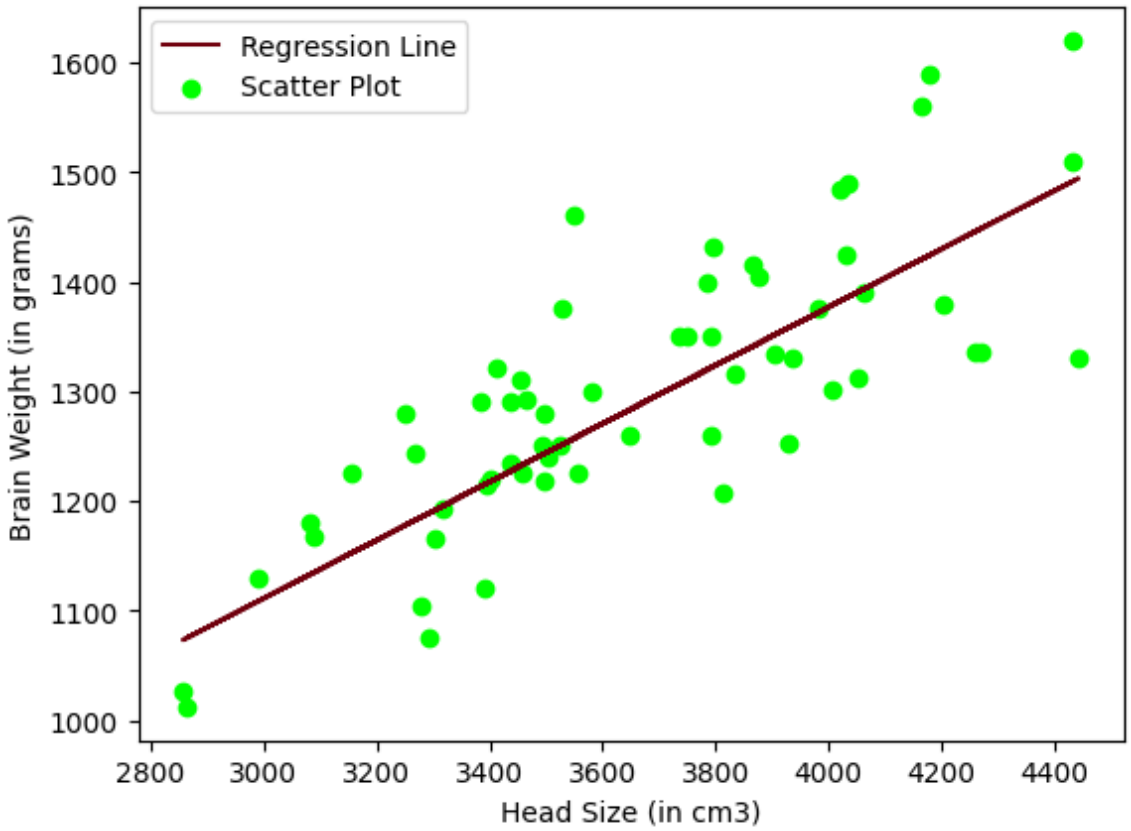
In [ ]:
```
# Create Linear Regression object
regr = LinearRegression()

# Train the model using the training sets
regr.fit(X_train, Y_train)

# Make predictions using the testing set
Y_pred2 = regr.predict(X_test)
```

Output and Comparison of Both Methods.

In [ ]:
```
# For Manual Method
# Plotting Line and Scatter Points
plt.plot(X_test, Y_pred1, color='#70000d', label='Regression Line')
plt.scatter(X_test, Y_test, c='#00ff00', label='Scatter Plot')
plt.xlabel('Head Size (in cm3)')
plt.ylabel('Brain Weight (in grams)')
plt.legend()
# Output: The Plot for Regression Line, Coefficients, RMSE and the R2 Sco
print("FOR LINEAR REGRESSION USING LEAST SQUARE METHOD MANUALLY \n")
plt.show()
print("\nCoefficients: m = ",m1," ; c = ", c1)
print('\nRMSE: %.4f' %rmse)
print('\nR2 Score: %.4f' %r2)

# For SciKit-Learn Method
# Plotting Line and Scatter Points
plt.plot(X_test, Y_pred2, color='#70000d', label='Regression Line')
plt.scatter(X_test, Y_test, c='#00ff00', label='Scatter Plot')
plt.xlabel('Head Size (in cm3)')
plt.ylabel('Brain Weight (in grams)')
plt.legend()
# Output: The Plot for Regression Line, Coefficients, RMSE and the R2 Sco
print("_____
print("\nFOR LINEAR REGRESSION USING LEAST SQUARE METHOD WITH SCIKIT-LEAR
plt.show()
print("\nCoefficients: m = ",regr.coef_," ; c = ", regr.intercept_)
```

```
print("\nRMSE: %.4f" % mean_squared_error(Y_test, Y_pred2, squared = Fals
print('\nR2 Score: %.4f' % r2_score(Y_test, Y_pred2))
```

FOR LINEAR REGRESSION USING LEAST SQUARE METHOD MANUALLY

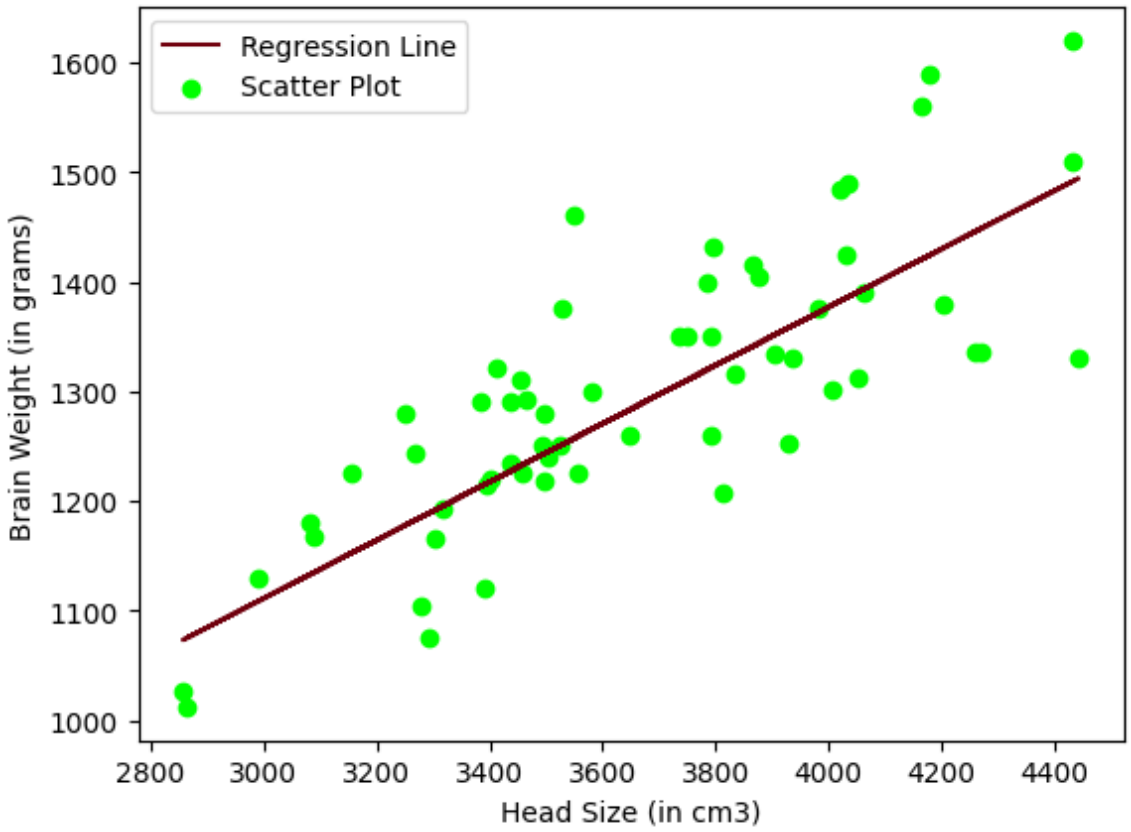


Coefficients: m = [0.26568351] ; c = [314.02213626]

RMSE: 78.1081

R2 Score: 0.6101

FOR LINEAR REGRESSION USING LEAST SQUARE METHOD WITH SCIKIT-LEARN




```
Coefficients: m =  [[0.26568351]]  ; c =  [314.02213626]

RMSE: 78.1081

R2 Score: 0.6101
```

On comparison, we can see that both the methods (viz. Least Square Method Manually and Least Square Method with SciKit-Learn) return same value of the Coefficients as well as the Root Mean Square Error and R2 Score.

1(b): Linear Regression using Gradient Descent Method

Steps:

1) Load the require libraries.
2) Import the dataset
3) Preprocess the dataset
4) Seperate the datset into independent and dependent variable
5) Split the dataset into train and test data with your desired ratio.
6) Manual Method:
   (a) Define the learning rate and number of iterations.
   (b) Perform Gradient Descent Method over training dataset as:
      (i) Initialize the parameters m and c with random values.
      (ii) Iterate for each.

$$y\_pred = m * X\_train + c$$

$$D\_m = [-2/len(X\_train)] * \Sigma[X\_train * (Y\_train - Y\_pred)$$

$$D\_c = [-2/len(X\_train)] * \Sigma(Y\_train - y\_pred).$$

$$m = m - (learningrate * D\_m$$

$$c = c - (learning\ rate * D\_c$$

   (c) Predict
   (d) Calculate RMS and R2

7) Scikit-Learn Method
   a) Load the S.G.D Regression() Method.
   b) Train, Predict, Calculate RMS and R2

8) Compare both the Results
9) Plot the line of Regression for both Methods

b) Implement the linear regression using Gradient Descent method in your own code. Also implement the linear regression by using existing library (scikit-learn). Compare the performance of both implementations.

Manual Method

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import SGDRegressor
from sklearn import preprocessing, svm, datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

# Function for Linear Regression using Gradient Descent
def SGD(X, y, lr=0.05, epoch=10, batch_size=1):
    m, c = 0, 0                                          # Initialize P
    for _ in range(epoch):
        indexes = np.random.randint(0, len(X), batch_size) # Random Sampl
        Xs = np.take(X, indexes)
        ys = np.take(y, indexes)
        N = len(Xs)
        f = ys - (m*Xs + c)
        # Updating parameters m and b
        m -= lr * (-2 * Xs.dot(f).sum() / N)
        c -= lr * (-2 * f.sum() / N)
    return m, c

# Input: Dataset
data = pd.read_csv('sgdregress.csv')

# Taking only two attributes of the Dataset
data_binary = data[['C1','C2']]

# Eliminating NaN or missing input numbers
data_binary.fillna(method ='ffill', inplace = True)

# Dropping any rows with Nan values
data_binary.dropna(inplace = True)

# Separating the data into independent and dependent variables
# Converting each dataframe into a numpy array
X = np.array(data_binary['C1']).reshape(-1, 1)
Y = np.array(data_binary['C2']).reshape(-1, 1)

# Dividing into test and training sets
X_train, X_test, Y_train, Y_test =  train_test_split(X,Y,test_size=0.3)

# Total number of Test values
n = len(Y_test)

# Mean X and Y
mean_x_train = np.mean(X_train)
mean_y_train = np.mean(Y_train)
mean_x_test = np.mean(X_test)
```

```
mean_y_test = np.mean(Y_test)

# Training the Regression Model over Training Set
m, c = SGD(X_train, Y_train, lr=0.0001, epoch=1000, batch_size=2)

# Testing of Regression Model over Testing Set
Y_pred = m*X_test + c

# Calculating Root Mean Squares Error & R2 Score
rmse = 0
ss_tot = 0
ss_res = 0
for i in range(n):
    y_pred = c + m * X_test[i]
    rmse += (Y_test[i] - y_pred) ** 2
    ss_tot += (Y_test[i] - mean_y_test) ** 2
    ss_res += (Y_test[i] - y_pred) ** 2

rmse = np.sqrt(rmse/n)
r2 = 1 - (ss_res/ss_tot)
```

(SciKit-Learn) Method.

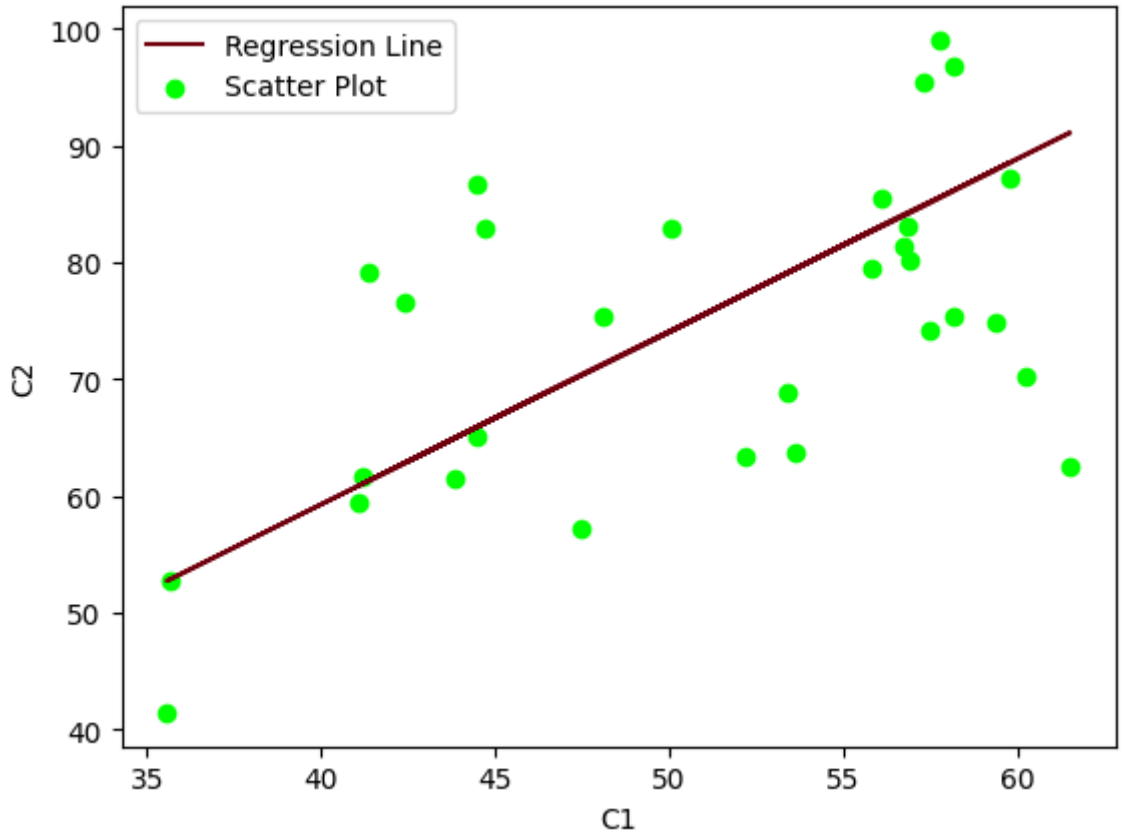
In [ ]:
```
# Create Linear Regression object
clf = SGDRegressor(max_iter=10000, learning_rate='constant',eta0=0.0001)

# Train the model
clf.fit(X_train,Y_train)

# Make predictions
Y_predict= clf.predict(X_test)
```

Output and Comparison of Both Methods.

In [ ]:
```
# For Manual Method
# Plotting Line and Scatter Points
plt.plot(X_test, Y_pred, color='#70000d', label='Regression Line')
plt.scatter(X_test, Y_test, c='#00ff00', label='Scatter Plot')
plt.xlabel('C1')
plt.ylabel('C2')
plt.legend()
# Output: The Plot for Regression Line, Coefficients, RMSE and the R2 Sco
print("FOR LINEAR REGRESSION USING GRADIENT DESCENT METHOD MANUALLY \n")
plt.show()
print("\nCoefficients: m = ",m," ; c = ", c)
print('\nRMSE: %.4f' %rmse)
print('\nR2 Score: %.4f' %r2)

# For SciKit-Learn Method
# Plotting Line and Scatter Points
plt.plot(X_test, Y_predict, color='#70000d', label='Regression Line')
plt.scatter(X_test, Y_test, c='#00ff00', label='Scatter Plot')
plt.xlabel('C1')
plt.ylabel('C2')
plt.legend()
# Output: The Plot for Regression Line, Coefficients, RMSE and the R2 Sco
print("_____")
print("\nFOR LINEAR REGRESSION USING GRADIENT DESCENT METHOD WITH SCIKIT-
plt.show()
```

```
print("\nCoefficients: m = ",clf.coef_," ; c = ", clf.intercept_)
print("\nRMSE: %.4f" % mean_squared_error(Y_test, Y_predict, squared = Fa
print('\nR2 Score: %.4f' % r2_score(Y_test, Y_predict))
```

FOR LINEAR REGRESSION USING GRADIENT DESCENT METHOD MANUALLY

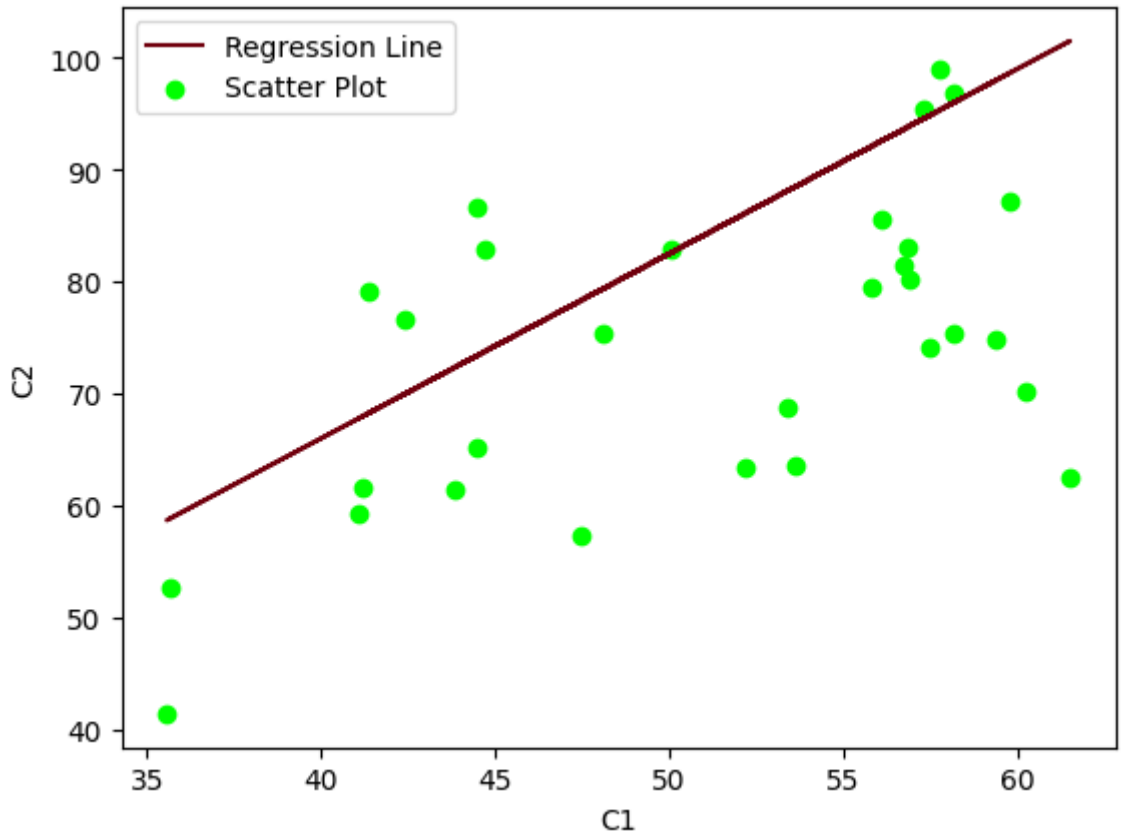


Coefficients: m = 1.4796655176468976 ; c = 0.06449702408542972

RMSE: 11.8472

R2 Score: 0.2007

FOR LINEAR REGRESSION USING GRADIENT DESCENT METHOD WITH SCIKIT-LEARN




```
Coefficients: m =  [1.64865489]  ; c =  [0.04304978]

RMSE: 15.8981

R2 Score: -0.4393
```

On comparison, we can see that both the methods (viz. Gradient Descent Method Manually and Gradient Descent Method with SciKit-Learn) return approximately same value of Root Mean Square Error and R2 Score with a slight difference between the Coefficients calculated.

Also, on comparing the R2 Scores of both methods, it can be seen that the Model created by Manual Method fits more as compared to Model created by SciKit-Learn Method.

Assignment 2: Logistic Regression

Steps

(i) Load the required libraries

(ii) Import the Dataset.

(iii) Preprocess the Dataset

(iv) Split the dataset into testing and training dataset.

(v) Separate dependent and independent variable

(vi) Manual Method:

(a) Initialize weights (theta).

(b) for each iteration

(i) Initialize the parameters for model

(ii) Calculate sigmoid function value for the parameters.

(iii) Calculate Gradient value as

$$\text{Gradient} = \frac{X\_train * (\text{sigmoid}(\text{Parameters}) - Y\_train)}{\text{len}(Y\_train)}$$

(iv) Update theta as

theta -= learning rate * Gradient

(v) Calculate new sigmoid value over updated parameters

(vi) Calculate Loss for the iteration.

(c) Predict value of y for X_test by predicting probability value of based on the weight of the der iterations.

(d) Plot the decision boundary and calculate Accuracy confusion matrix and classification report.

(vii) Scikit-Learn Method

(a) Load Logistic Regressor() model, Train then predict, calculate classification report and plot.

(viii) Compare the results for both methods.

Exercise 2: Logistic Regression

Implement the logistic regression in your own code. Also implement the logistic regression by using existing library (e.g. scikit-learn). Compare the performance of both implementations and show the results. Use the following evaluation metrics: (a) Mean Squared Error (MSE) (b) Root Mean Squared Error (RMSE) (c) Mean Absolute Error (MAE) (d) R Squared (R^2).

Manual Method

In [ ]:

```
import numpy as np
from matplotlib import pyplot as plt
import sklearn.metrics as mt
import pandas as pd
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classificat

# Class for Logistic Regression Model
class LogisticRegress:
    def __init__(self, lr=0.01, num_iter=100000, fit_intercept=True, verb
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose

    # Function to define the Intercept value
    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    # Sigmoid Function to Predict Y
    def __sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    # Loss Function to minimize the Error of our Model
    def __loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    # Function for Model Training
    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # Weights Initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, (h - y)) / y.size
            self.theta -= self.lr * gradient

            z = np.dot(X, self.theta)
```

```
            h = self.__sigmoid(z)
            loss = self.__loss(h, y)

            if(self.verbose ==True and i % 100000 == 0):
                print(f'loss: {loss} \t')

    # Predict Probability Values based on generated W values out of all i
    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)
        return self.__sigmoid(np.dot(X, self.theta))

    # To predict the Actual Values (0 or 1)
    def predict(self, X):
        return self.predict_prob(X).round()

# Input: Dataset
iris = datasets.load_iris()
X = iris.data[:, :2]
y = (iris.target != 0) * 1

# Dividing into test and training sets
X_train, X_test, Y_train, Y_test =  train_test_split(X,y,test_size=0.3)

# Create Logistic Regression object
model = LogisticRegress(lr=0.01, num_iter=300000)

# Train the model using the training sets
model.fit(X_train, Y_train)

# Make predictions using the testing set
preds = model.predict(X_test)
```

Logistic Regression (SciKit-Learn).

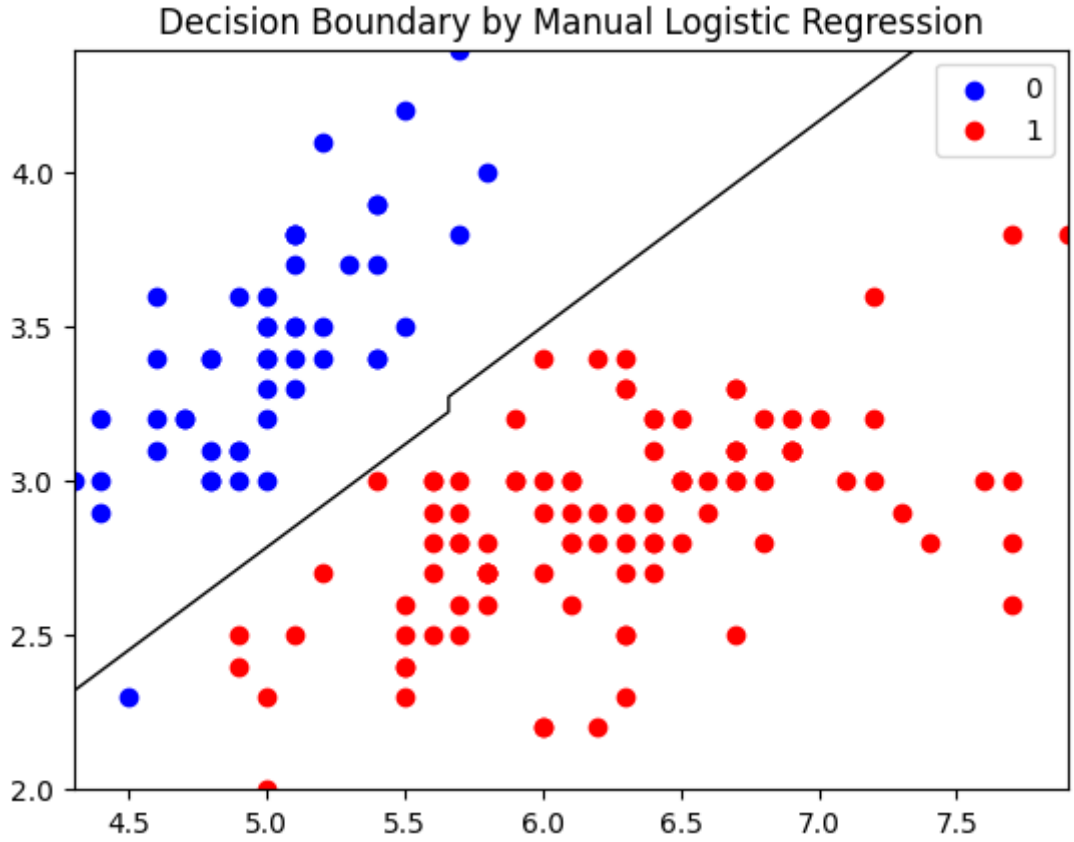
In [ ]:
```
# Create Logistic Regression object
logreg = LogisticRegression(max_iter=300000)

# Train the model using the training sets
logreg.fit(X_train,Y_train)

# Make predictions using the testing set
Y_pred = logreg.predict(X_test)
```

Output and Comparison of Both Methods.

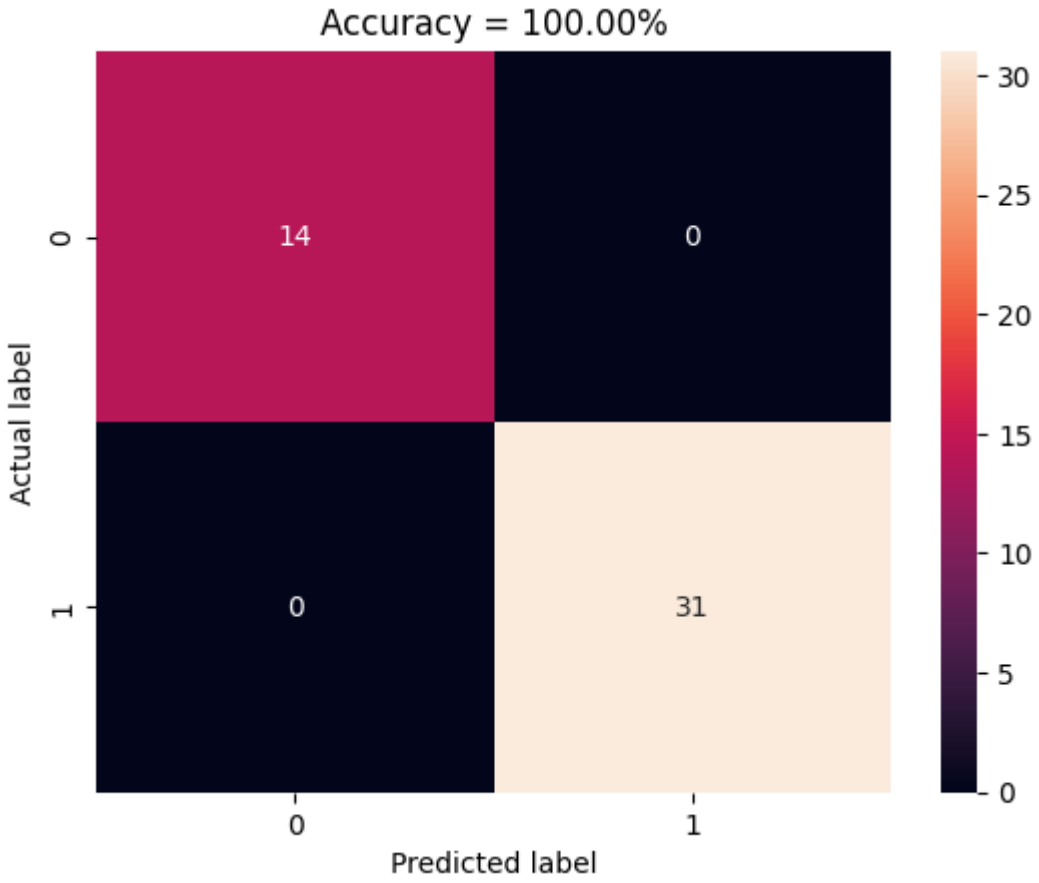
In [ ]:
```
# For Manual Method
# Plotting Line and Scatter Points
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='1')
plt.title("Decision Boundary by Manual Logistic Regression")
plt.legend()
x1_min, x1_max = X[:,0].min(), X[:,0].max(),
x2_min, x2_max = X[:,1].min(), X[:,1].max(),
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x
grid = np.c_[xx1.ravel(), xx2.ravel()]
probs = model.predict(grid).reshape(xx1.shape)
plt.contour(xx1, xx2, probs, [0.5], linewidths=1, colors='black');
```

```
# Output: Regression Line Plot, Confusion Matrix, Accuracy & Classificati
print("FOR LOGISTIC REGRESSION USING MANUAL METHOD \n")
plt.show()
print("\nConfusion Matrix\n")
cm = mt.confusion_matrix(Y_test, preds)
cm_df = confusion_matrix(Y_test, preds)
sns.heatmap(cm_df, annot=True)
plt.title('Accuracy = {0:.2f}%'.format(accuracy_score(Y_test, preds)*100)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\n\nClassification Report\n\n",classification_report(Y_test, preds

# For SciKit-Learn Method
# Plotting Line and Scatter Points
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], color='b', label='0')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='r', label='1')
plt.title("Decision Boundary by SciKit-Learn Logistic Regression")
plt.legend()
x1_min, x1_max = X[:,0].min(), X[:,0].max(),
x2_min, x2_max = X[:,1].min(), X[:,1].max(),
xx1, xx2 = np.meshgrid(np.linspace(x1_min, x1_max), np.linspace(x2_min, x
grid = np.c_[xx1.ravel(), xx2.ravel()]
probs = logreg.predict(grid).reshape(xx1.shape)
plt.contour(xx1, xx2, probs, [0.5], linewidths=1, colors='black');

# Output: Regression Line Plot, Confusion Matrix, Accuracy & Classificati
print("_____
print("\nFOR LOGISTIC REGRESSION USING SCIKIT-LEARN METHOD \n")
plt.show()
print("\nConfusion Matrix\n")
cm = mt.confusion_matrix(Y_test, Y_pred)
cm_df = confusion_matrix(Y_test, Y_pred)
sns.heatmap(cm_df, annot=True)
plt.title('Accuracy = {0:.2f}%'.format(accuracy_score(Y_test, Y_pred)*100
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\nClassification Report\n\n",classification_report(Y_test, Y_pred)
```

```
FOR LOGISTIC REGRESSION USING MANUAL METHOD
```



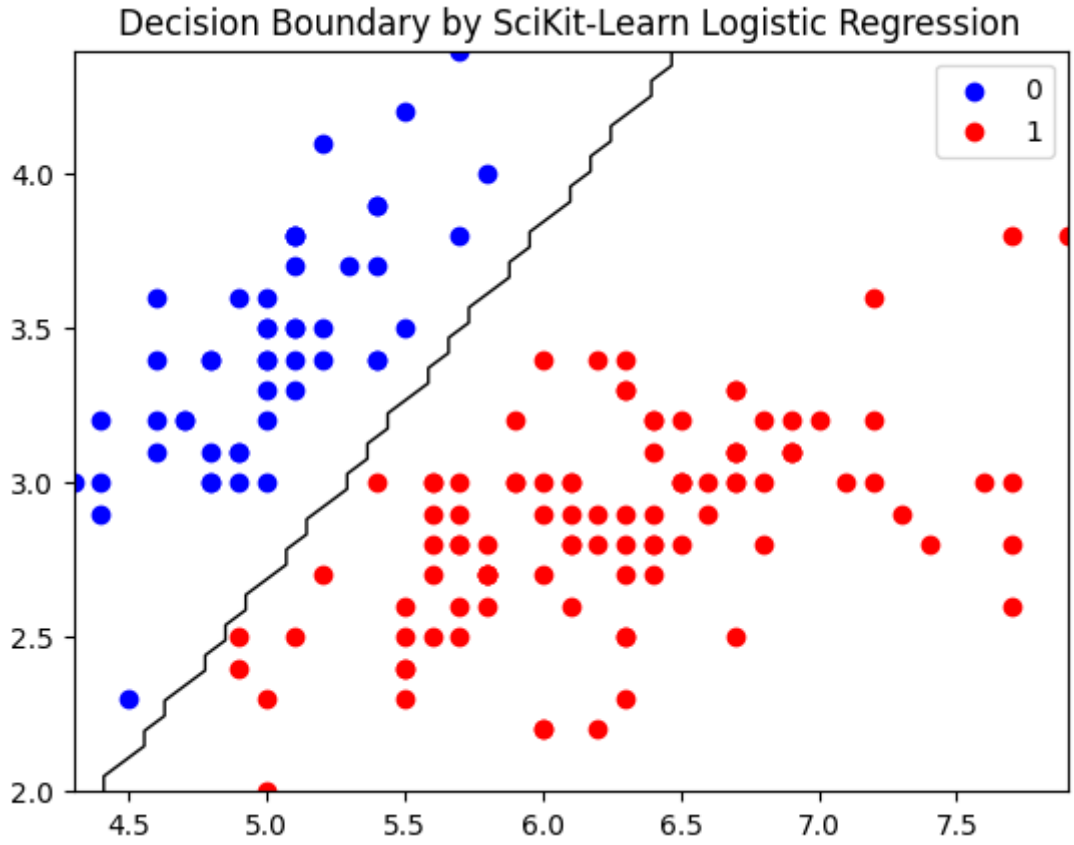
Confusion Matrix



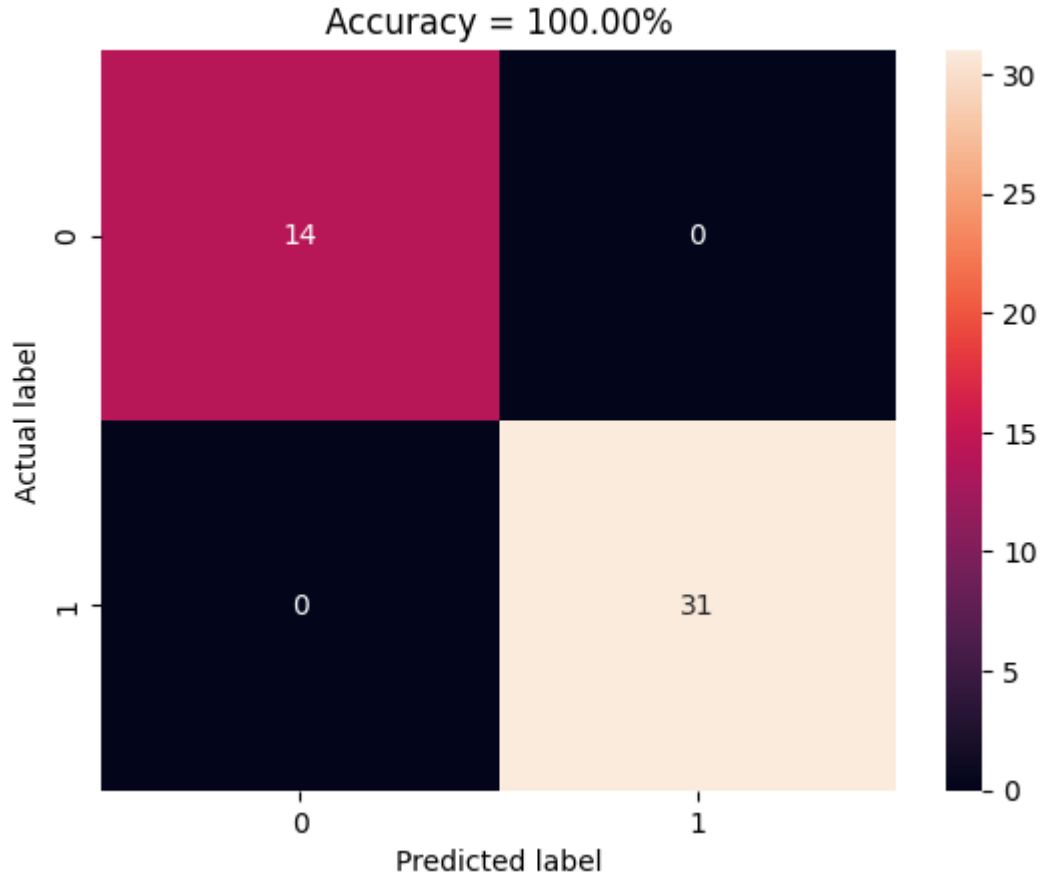
Classification Report

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
1	1.00	1.00	1.00	31
accuracy			1.00	45
macro avg	1.00	1.00	1.00	45
weighted avg	1.00	1.00	1.00	45

FOR LOGISTIC REGRESSION USING SCIKIT-LEARN METHOD



Confusion Matrix




```
Classification Report

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        14
           1       1.00      1.00      1.00        31

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

On comparison, we can see that Logistic Regression Method using SciKit-Learn is more accurate than the Manual Logistic Regression Method. The SciKit-Learn Method has an Accuracy of 100% whereas Manual Method gave an Accuracy of 97.78%.

Assignment 3: Linear Discriminant Analysis

Steps:

(i) Load the required libraries.

(ii) Import the database of dataset

(iii) Encode categorical class labels

(iv) Standardize features by removing the mean and scaling to unit variance

(v) Construct within-class covariant Scatter Matrix

(vi) Construct between-class scatter Matrix

(vii) Calculate sorted Eigen values and Eigen Vectors of Inverse of (within class scatter matrix b/w class scatter matrix).

(viii) Project original features onto the new feature space.

(ix) Save the reduced dataset as Reduced_Manual.

(x) Load the LinearDiscriminant() Model.

(xi) Load the original Dataset (for sci-kit learn)

(xii) Train the dataset over model to obtain the reduced dimensionality dataset.

(xiii) Save it as Reduced_Scikit_learn.

(xiv) To check the accuracy of an algorithm (say KNN classifier) train the KNN classifier model over Original Dataset, Reduced_Manual and Reduced_Manual scikit-learn datasets seperately

(xv) Calculate the Accuracy of each dataset.

Exercise 3: Linear Discriminant Analysis

Implement the Linear Discriminant Analysis algorithm in your own code for class classification. Also implement the Discriminant Analysis algorithm by using existing library (e.g. scikit-learn). Compare the performance of both implementations and show the results.

Manual Method

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.metrics
import warnings
warnings.filterwarnings('ignore')
import seaborn as sns
from sklearn import preprocessing, svm, linear_model, decomposition
from sklearn.metrics import mean_squared_error, r2_score, confusion_matri
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
np.set_printoptions(precision=4)

# Input: Dataset
iris = pd.read_csv('iris.csv', header=None)

# Encode Categorical Class Labels
class_le = LabelEncoder()
Y = class_le.fit_transform(iris[4].values)

# Standardize features
stdsc = StandardScaler()
X = stdsc.fit_transform(iris.iloc[:,range(0,4)].values)

# Construct within-class covariant scatter matrix S_W
S_W = np.zeros((4,4))
for i in range(3):
    S_W += np.cov(X[Y==i].T)

# Construct between-class scatter matrix S_B
N=np.bincount(Y)
vecs=[]
[vecs.append(np.mean(X[Y==i],axis=0)) for i in range(3)] # Class Means
mean_overall = np.mean(X, axis=0)                       # Overall Mean
S_B=np.zeros((4,4))
for i in range(3):
    S_B += N[i]*(((vecs[i]-mean_overall).reshape(4,1))
                 .dot(((vecs[i]-mean_overall).reshape(1,4))))

# Calculate Sorted Eigen Values and Eigen Vectors of  inverse(S_W)dot(S_B
eigen_vals, eigen_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
eigen_pairs = [(np.abs(eigen_vals[i]), eigen_vecs[:,i])
               for i in range(len(eigen_vals))]
eigen_pairs = sorted(eigen_pairs,key=lambda k: k[0], reverse=True)
```

```
# Project original features onto the new feature space
W=np.hstack((eigen_pairs[0][1][:, ].reshape(4,1),eigen_pairs[1][1][:, ].
             reshape(4,1))).real
X_lda = X.dot(W)

data=pd.DataFrame(X_lda)
data['class']=Y
data.columns=["LD1","LD2","class"]

# Save the Reduced Dimension Dataset
irisLDA_Manual = pd.concat([data],axis=1)
irisLDA_Manual.to_csv('irisLDA_Manual.csv')
```

(SciKit-Learn) Method.

In [ ]:
```
# Create Linear Discriminant Analysis object
LDA = LinearDiscriminantAnalysis(n_components=2)

# Train the model
X_lda2 = LDA.fit_transform(X, Y)

data2=pd.DataFrame(X_lda2)
data2['class']=Y
data2.columns=["LD1","LD2","class"]

# Save the Reduced Dimension Dataset
irisLDA_SKL = pd.concat([data2],axis=1)
irisLDA_SKL.to_csv('irisLDA_SKL.csv')
```

Output and Comparison of Both Methods.

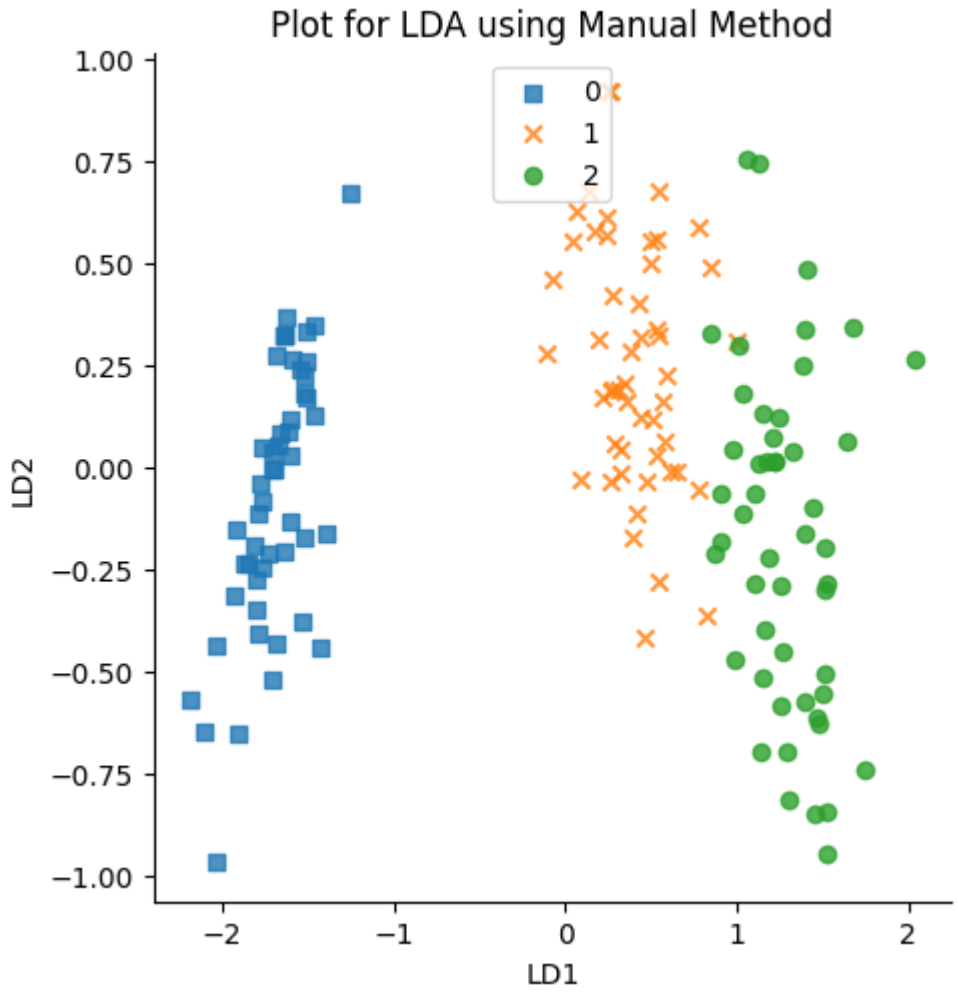
In [ ]:
```
# Metric Calculation for Original Dataset
# Check Accuracy over Original Dataset
print("\nACCURACY METRIC OF THE ORIGINAL DATASET\n")
# Split the Original Dataset into Training and Testing Datasets and
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3
Y_train = Y_train.ravel()
Y_test = Y_test.ravel()

# Output: The Shape of the Original Dataset and its Accuracy over KNN Cla
print("\nShape of Original Dataset : X = ", X.shape," Y = ", Y.shape)
print('\nAccuracy of IRIS Dataset Before LDA \n')
for K in range(25):
    K_value = K+1
    #Using KNN Classifier to Check Accuracy before LDA
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform'
                                 algorithm='auto')
    neigh.fit(X_train, Y_train)
    Y_pred = neigh.predict(X_test)
    if(K_value%2 == 0):
        print ("    Accuracy = ", accuracy_score(Y_test,Y_pred)*100,
               "% for K-Value = ",K_value)


# Metric Calculation for Reduced Dataset obtained by LDA using Manual Met
print("\n_____________________________________________________________
print("\nFOR LINEAR DISCRIMINANT ANALYSIS USING MANUAL METHOD\n")
```

```
# Load the Reduced Dataset into Pandas DataFrame
irisldam = pd.read_csv("irisLDA_Manual.csv")

features = ['LD1', 'LD2']
# Separating out the Features
X1 = irisldam.loc[:, features].values
# Separating out the Target
Y1 = irisldam.loc[:,['class']].values

# Check Accuracy over Reduced Dataset
# Split the Original Dataset into Training and Testing Datasets
X_train1, X_test1, Y_train1, Y_test1 = train_test_split( X1, Y1, test_siz
Y_train1 = Y_train1.ravel()
Y_test1 = Y_test1.ravel()

# Output: The Shape of Reduced Dataset, its Plot & its Accuracy over KNN
print("\nShape of Reduced Dataset  : X = ", X1.shape," Y = ", Y1.shape)
print('\nAccuracy of IRIS Dataset after LDA\n')
for K in range(25):
    K_value = K+1

    # Using KNN Classifier to Check Accuracy after LDA
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform'
                                 algorithm='auto')
    neigh.fit(X_train1, Y_train1)
    Y_pred1 = neigh.predict(X_test1)
    if(K_value%2 == 0):
        print ("    Accuracy = ", accuracy_score(Y_test1,Y_pred1)*100,
               "% for K-Value =",K_value)

markers = ['s', 'x','o']
sns.lmplot(x="LD1", y="LD2", data=data,markers=markers,fit_reg=False,
           hue='class', legend=False)
plt.legend(loc='upper center')
plt.title('Plot for LDA using Manual Method')
plt.show()

# Metric Calculation for Reduced Dataset obtained by LDA using SciKit-Lea
print("\n_____________________________________________________________________
print("\nFOR LINEAR DISCRIMINANT ANALYSIS USING SCIKIT-LEARN METHOD\n")
# Load the Reduced Dataset into Pandas DataFrame
irisldas = pd.read_csv("irisLDA_SKL.csv")

features = ['LD1', 'LD2']
# Separating out the Features
X2 = irisldas.loc[:, features].values
# Separating out the Target
Y2 = irisldas.loc[:,['class']].values

# Check Accuracy over Reduced Dataset
# Split the Original Dataset into Training and Testing Datasets
X_train2, X_test2, Y_train2, Y_test2 = train_test_split( X2, Y2, test_siz
Y_train2 = Y_train2.ravel()
Y_test2 = Y_test2.ravel()

# Output: The Shape of Reduced Dataset, its Plot & its Accuracy over KNN
print("\nShape of Reduced Dataset  : X = ", X2.shape," Y = ", Y2.shape)
print('\nAccuracy of IRIS Dataset after LDA\n')
for K in range(25):
    K_value = K+1
```

```
    # Using KNN Classifier to Check Accuracy after LDA
    neigh = KNeighborsClassifier(n_neighbors = K_value, weights='uniform'
                                 algorithm='auto')
    neigh.fit(X_train2, Y_train2)
    Y_pred2 = neigh.predict(X_test2)
    if(K_value%2 == 0):
        print ("    Accuracy = ", accuracy_score(Y_test2,Y_pred2)*100,
               "% for K-Value =",K_value)

markers = ['s', 'x','o']
sns.lmplot(x="LD1", y="LD2", data=data2, markers=markers, fit_reg=False,
           hue='class', legend=False)
plt.legend(loc='upper center')
plt.title('Plot for LDA using SciKit-Learn Method')
plt.show()
```

```
ACCURACY METRIC OF THE ORIGINAL DATASET


Shape of Original Dataset : X =  (150, 4)  Y =  (150,)

Accuracy of IRIS Dataset Before LDA

    Accuracy =  91.11111111111111 % for K-Value =  2
    Accuracy =  91.11111111111111 % for K-Value =  4
    Accuracy =  91.11111111111111 % for K-Value =  6
    Accuracy =  93.33333333333333 % for K-Value =  8
    Accuracy =  93.33333333333333 % for K-Value =  10
    Accuracy =  93.33333333333333 % for K-Value =  12
    Accuracy =  95.55555555555556 % for K-Value =  14
    Accuracy =  93.33333333333333 % for K-Value =  16
    Accuracy =  93.33333333333333 % for K-Value =  18
    Accuracy =  91.11111111111111 % for K-Value =  20
    Accuracy =  88.88888888888889 % for K-Value =  22
    Accuracy =  88.88888888888889 % for K-Value =  24
```

```
FOR LINEAR DISCRIMINANT ANALYSIS USING MANUAL METHOD


Shape of Reduced Dataset  : X =  (150, 2)  Y =  (150, 1)

Accuracy of IRIS Dataset after LDA

    Accuracy =  97.77777777777777 % for K-Value = 2
    Accuracy =  100.0 % for K-Value = 4
    Accuracy =  100.0 % for K-Value = 6
    Accuracy =  100.0 % for K-Value = 8
    Accuracy =  100.0 % for K-Value = 10
    Accuracy =  100.0 % for K-Value = 12
    Accuracy =  100.0 % for K-Value = 14
    Accuracy =  100.0 % for K-Value = 16
    Accuracy =  100.0 % for K-Value = 18
    Accuracy =  100.0 % for K-Value = 20
    Accuracy =  100.0 % for K-Value = 22
    Accuracy =  100.0 % for K-Value = 24
```

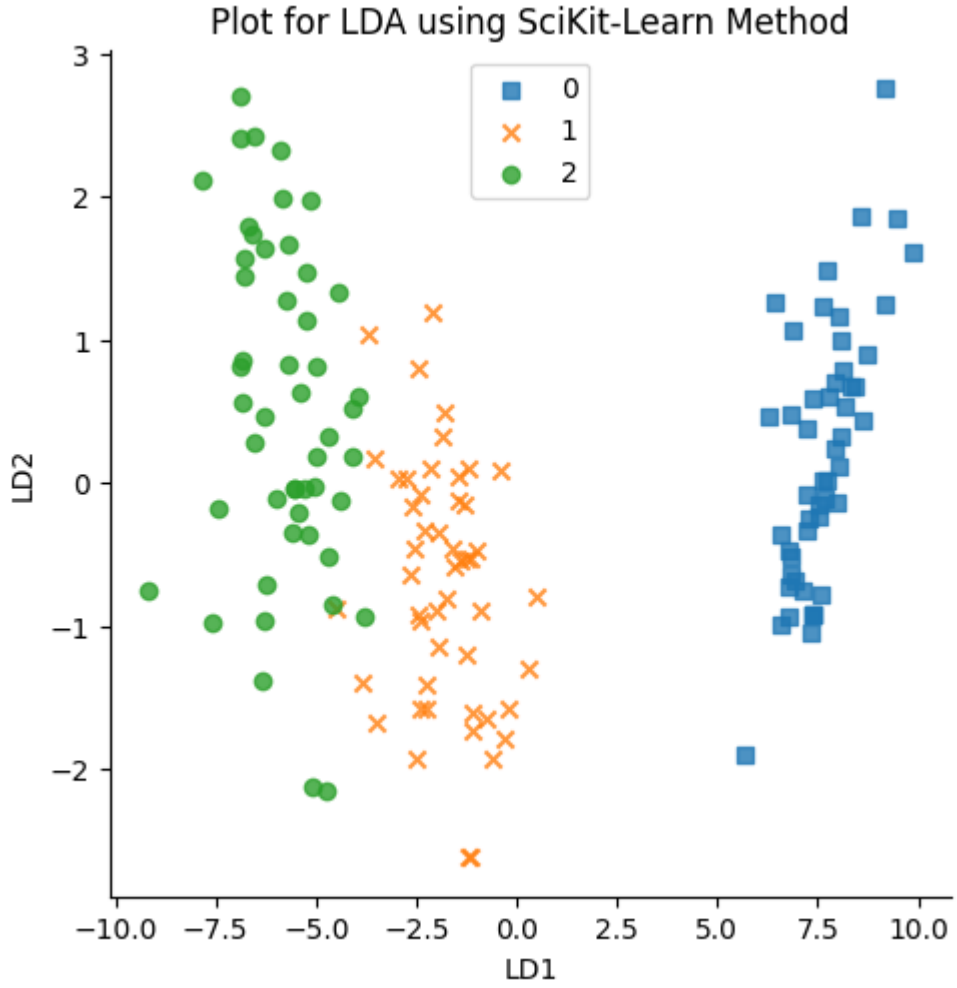


FOR LINEAR DISCRIMINANT ANALYSIS USING SCIKIT-LEARN METHOD

Shape of Reduced Dataset : X = (150, 2) Y = (150, 1)

Accuracy of IRIS Dataset after LDA

```
    Accuracy =  95.55555555555556 % for K-Value = 2
    Accuracy =  97.77777777777777 % for K-Value = 4
    Accuracy =  97.77777777777777 % for K-Value = 6
    Accuracy =  97.77777777777777 % for K-Value = 8
    Accuracy =  97.77777777777777 % for K-Value = 10
    Accuracy =  97.77777777777777 % for K-Value = 12
    Accuracy =  97.77777777777777 % for K-Value = 14
    Accuracy =  97.77777777777777 % for K-Value = 16
    Accuracy =  97.77777777777777 % for K-Value = 18
    Accuracy =  97.77777777777777 % for K-Value = 20
    Accuracy =  97.77777777777777 % for K-Value = 22
    Accuracy =  100.0 % for K-Value = 24
```



On comparison, we can see that both the methods (viz. Linear Discriminant Analysis Manually and Linear Discriminant Analysis with SciKit-Learn) reduce the dataset from 4 dimensions to 2 dimensions.

On comparing the Accuracy of a Classifier (here KNN Classifier) we can see that the Reduced Datasets obtained using LDA increases with respect to the Classifier Accuracy over Original Dataset.

The KNN Classifier gives higher accuracy for Dataset reduced by Manual Method over SciKit-Learn Method for smaller values of k, but attains higher accuracy for Dataset reduced by SciKit-Learn Method over Manual Method for larger k values.

Assignment 4: Decision Tree

Steps:

(i) Load the required libraries

(ii) Import the dataset

(iii) Separate dependent and Independent variables

(iv) Label Encoding of target values

(v) Split the dataset into testing and training dataset

(vi) Define maximum depth.

(vii) Initialize the tree with Original training set as root node.

(viii) for each iteration (until max depth is not achieved).

(a) Iterate through very unused attribute of the dataset and calculate entropy and Information Gain of the attribute.

(b) Select the attribute which has the lowest Entropy or highest Information Gain.

(c) Split the tree by selected attribute to produce Child Node (Subset of Data).

(d) Continue to recur on each subset until max-depth is not achieved.

Entropy $H_S = \sum -p_i \log_2 p_i$ where p_i is the probability of features of state S.

Information Gain = Entropy(T) - Entropy(T, X)

Difference blw Entropy of parent Node from Sum of Entropies of Child Nodes.

(ix) Predict the value of y for X-test by using the decision tree obtained from model training.

(x) Compute the Confusion Matrix, Classification Report and Accuracy.

Exercise 4: Implement Decision Tree

Classification Datasets: You can use one of the two datasets (or optionally, both datasets).

(a) Car Evaluation dataset D1: Target attribute safety :{low, med, high}. https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

(b) Iris dataset D2: Target attribute class :{Iris Setosa, Iris Versicolour, Iris Virginica}. https://archive.ics.uci.edu/ml/datasets/Iris Implement Decision Tree. In this task you will implement a decision tree. As a starting point you can implement your complete decision tree model for classification tasks. You have to split data into two parts train and test (70% and 30% respectively). Using the train data you will build a decision tree. Use Cross Entropy as a Quality-criterion. You have to provide information on the learning process that includes.

1. Define an appropriate stopping criteria i.e. max depth gain is too small or reduction in cost is

small 2. At each decision step (or split) present the probability of each class using histogram (properly labeled figure) 3. At each decision step, plot the Cross Entropy of each attribute. 4. Note down the Information Gain at each new node created, you can store it in node structure or class. Display it at the end. 5. Print your tree using a breath first tree traversal. (you can also print node hierarchical level, information gain and decision rule, etc). 6. On a test set measure the cross entropy loss (i.e. logloss, note that this time problem is not binary classification).

In [ ]:

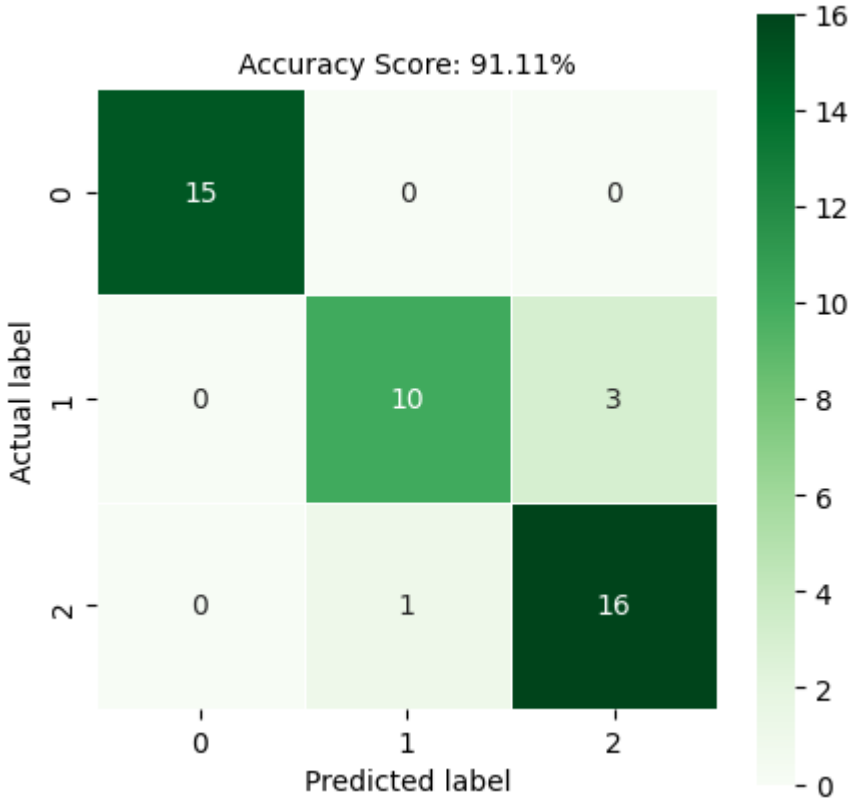
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import plot_tree

# Input: Dataset
iris = sns.load_dataset('iris')

target = iris['species']
iris1 = iris.copy()
iris1 = iris1.drop('species', axis =1)

# Defining the attributes
X = iris1
le = LabelEncoder()                # Label Encoding
target = le.fit_transform(target)
Y = target

# Dividing into test and training sets
```

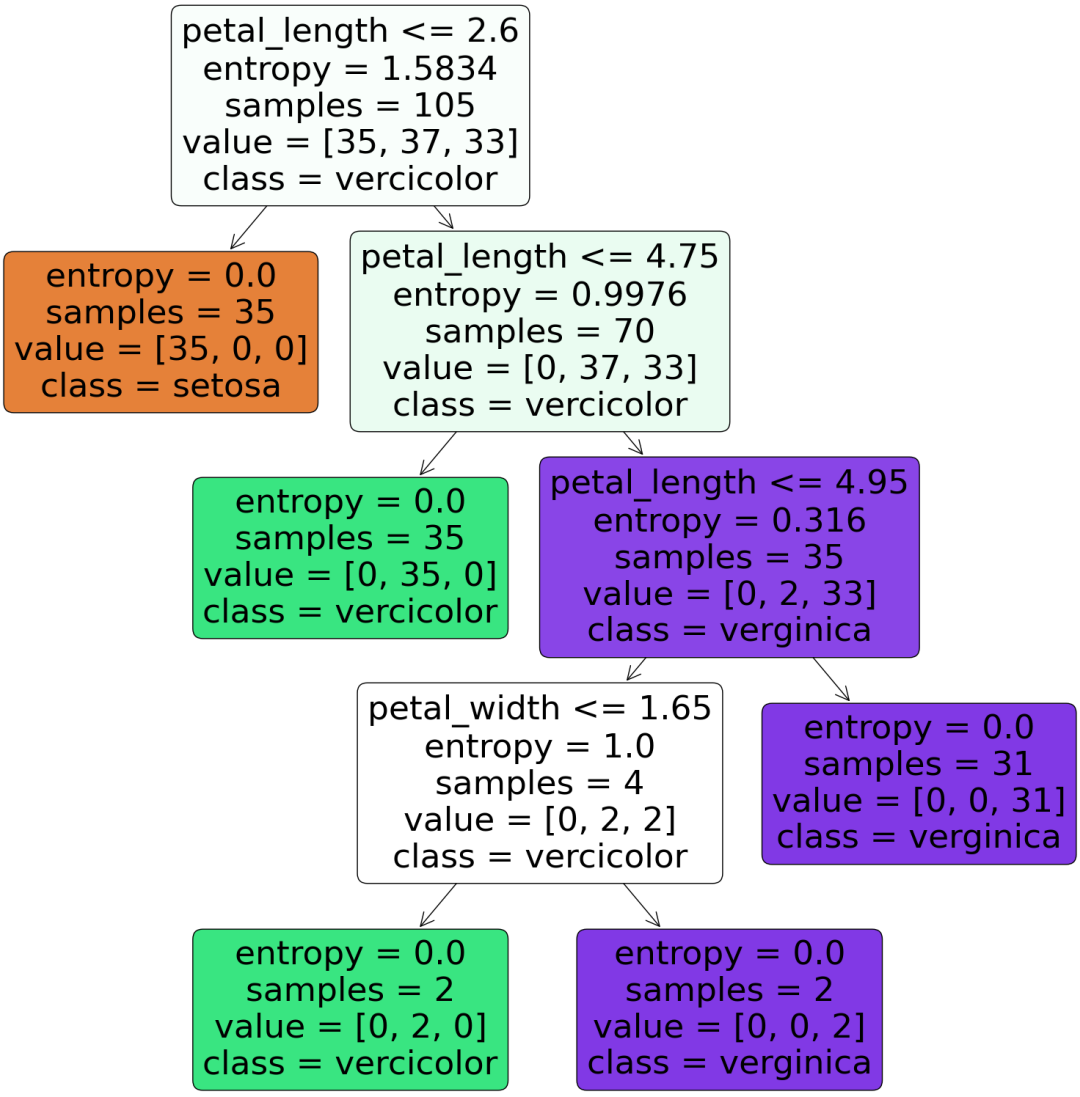
```
X_train, X_test, Y_train, Y_test = train_test_split(X , Y, test_size = 0.

# Create Decision Tree Classifier object
dtree = DecisionTreeClassifier(criterion = "entropy", max_depth = 5)

# Train the model using the training sets
dtree.fit(X_train,Y_train)

# Make predictions using the testing set
Y_pred = dtree.predict(X_test)
```

Output for the Decision Tree.

In [ ]:
```
# Output: The Classification Report, Confusion Matrix and Decision Tree
print("Classification Report - \n", classification_report(Y_test,Y_pred))
cm = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,square = True,  cmap = 'Gre
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score: {0:.2f}%'.format(dtree.score(X_test,
plt.title(all_sample_title, size = 10)
plt.figure(figsize = (20,20))
dec_tree = plot_tree(decision_tree=dtree, feature_names = iris.columns,
                     class_names =["setosa", "vercicolor", "verginica"] ,
                     filled = True , precision = 4, rounded = True)
```

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

           0       1.00      1.00      1.00        15
           1       0.91      0.77      0.83        13
           2       0.84      0.94      0.89        17

    accuracy                           0.91        45
   macro avg       0.92      0.90      0.91        45
weighted avg       0.91      0.91      0.91        45
```


Accuracy Score: 91.11%
Actual label
Predicted label
0
1
2
15
0
0
0
10
3
0
1
16
16
14
12
10
8
6
4
2
0


petal_length <= 2.6
entropy = 1.5834
samples = 105
value = [35, 37, 33]
class = vercicolor
entropy = 0.0
samples = 35
value = [35, 0, 0]
class = setosa
petal_length <= 4.75
entropy = 0.9976
samples = 70
value = [0, 37, 33]
class = vercicolor
entropy = 0.0
samples = 35
value = [0, 35, 0]
class = vercicolor
petal_length <= 4.95
entropy = 0.316
samples = 35
value = [0, 2, 33]
class = verginica
petal_width <= 1.65
entropy = 1.0
samples = 4
value = [0, 2, 2]
class = vercicolor
entropy = 0.0
samples = 31
value = [0, 0, 31]
class = verginica
entropy = 0.0
samples = 2
value = [0, 2, 0]
class = vercicolor
entropy = 0.0
samples = 2
value = [0, 0, 2]
class = verginica

On Implementation of Decision Tree Classifier over Iris Dataset with Maximum Depth of 5, we got an accuracy of approximately 98%. The Decision Tree shows the Entropy, Frequency of Classes and the Predicted Class for each node. Also the Information Gain can be computed for a node by subtracting its Entropy from the sum of Entropy of its Child Nodes.

Assignment 5: Random Forest Regression

Steps:

(i) Load the required Libraries

(ii) Import the dataset

(iii) Separate dependent and independent variables

(iv) Split the dataset into testing and traiing dataset

(v) Sort the dataset values according to X for visualization purpose

(vi) Manual Method:

(a) Define number of trees and maximum depth

(b) for each iteration (total trees)

(i) Construct tree using Decision Tree Regressor with atleast n splits (predefined) and max-depth

(ii) Use Bootstrap Aggregating ie select a random sample with replacement of training set

(iii) Train the tree over this random sample

(iv) Append this tree to trees[].

(c) Predict the value of y for test dataset as

(i) Predict trees for X_test from trees[].

(ii) Assign the value of most common label from trees to y.

(vii) Scikit-learn Method:

(a) Create Random Forest Regression() Object for the model, Train then predict.

(ix) Calculate the Mean Absolute Error and Root Mean Square Error for both methods and plot their regression lines separately.

(x) Compare the results for both methods.

Exercise 5: Random Forest Regression

Implement the random forest regression model. Evaluate your model using k-fold cross validation. Use the following evaluation metrics: (a) Mean Squared Error (MSE) (b) Root Mean Squared Error (RMSE) (c) Mean Absolute Error (MAE) (d) R Squared (R^2).

Manual Code

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn import preprocessing, datasets, linear_model
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.model_selection import train_test_split
from collections import Counter

def bootstrap_sample(X, y):
    n_samples = X.shape[0]
    idxs = np.random.choice(n_samples, n_samples, replace=True)
    return X[idxs], y[idxs]

def most_common_label(y):
    counter = Counter(y)
    most_common = counter.most_common(1)[0][0]
    return most_common

class RandomForest:

    def __init__(self, n_trees=10, min_samples_split=2,
                 max_depth=100, n_feats=None):
        self.n_trees = n_trees
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth
        self.trees = []

    def fit(self, X, y):
        self.trees = []
        for _ in range(self.n_trees):
            tree = DecisionTreeRegressor(min_samples_split=self.min_sampl
                max_depth=self.max_depth)
            X_samp, y_samp = bootstrap_sample(X, y)
            tree.fit(X_samp, y_samp)
            self.trees.append(tree)

    def predict(self, X):
        tree_preds = np.array([tree.predict(X) for tree in self.trees])
        tree_preds = np.swapaxes(tree_preds, 0, 1)
        y_pred = [most_common_label(tree_pred) for tree_pred in tree_pred
        return np.array(y_pred)

# Input: Dataset
data = pd.read_csv('rfgregress.csv')
```

```
# Separating Dependent and Independent Variables
X = data.iloc[:,0:1].values
Y = data.iloc[:, 1].values

# Splitting the data into training and testing data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.4

# Sorting the Test Dataset for Better Visualization of Plot
X_test_sort = (X_test)[1:-1].ravel()
X_test_sort = X_test_sort.tolist()
Y_test_sort = Y_test.tolist()
mat = [[0]*2 for i in range(len(X_test_sort))]
k=0
# Zipping both X and Y values to sort them together
for i, j in zip(X_test_sort, Y_test_sort):
    mat[k][0]=i
    mat[k][1]=j
    k = k+1
mats = sorted(mat)               # Sort the Zipped Matrix
X_tsort=[]
Y_tsort=[]
for i in range(len(mats)):       # Separating the X and Y values for Prea
    X_tsort.append(mats[i][0])
    Y_tsort.append(mats[i][1])
X_tsort=np.array(X_tsort).reshape(-1,1)

# Create Random Forest Regressor object
clf = RandomForest(n_trees = 100, max_depth = 10)

# Train the model using the training sets
clf.fit(X_train, Y_train)

# Make predictions using the testing set
y_pred = clf.predict(X_tsort)
```

Python Program for Random Forest Regression (SciKit-Learn).

In [ ]:
```
# Create Random Forest Regressor object
rfg = RandomForestRegressor(n_estimators = 1000)

# Train the model using the training sets
rfg.fit(X_train, Y_train)

# Make predictions using the testing set
Y_pred = rfg.predict(X_tsort)
```

Output and Comparison of Both Methods.

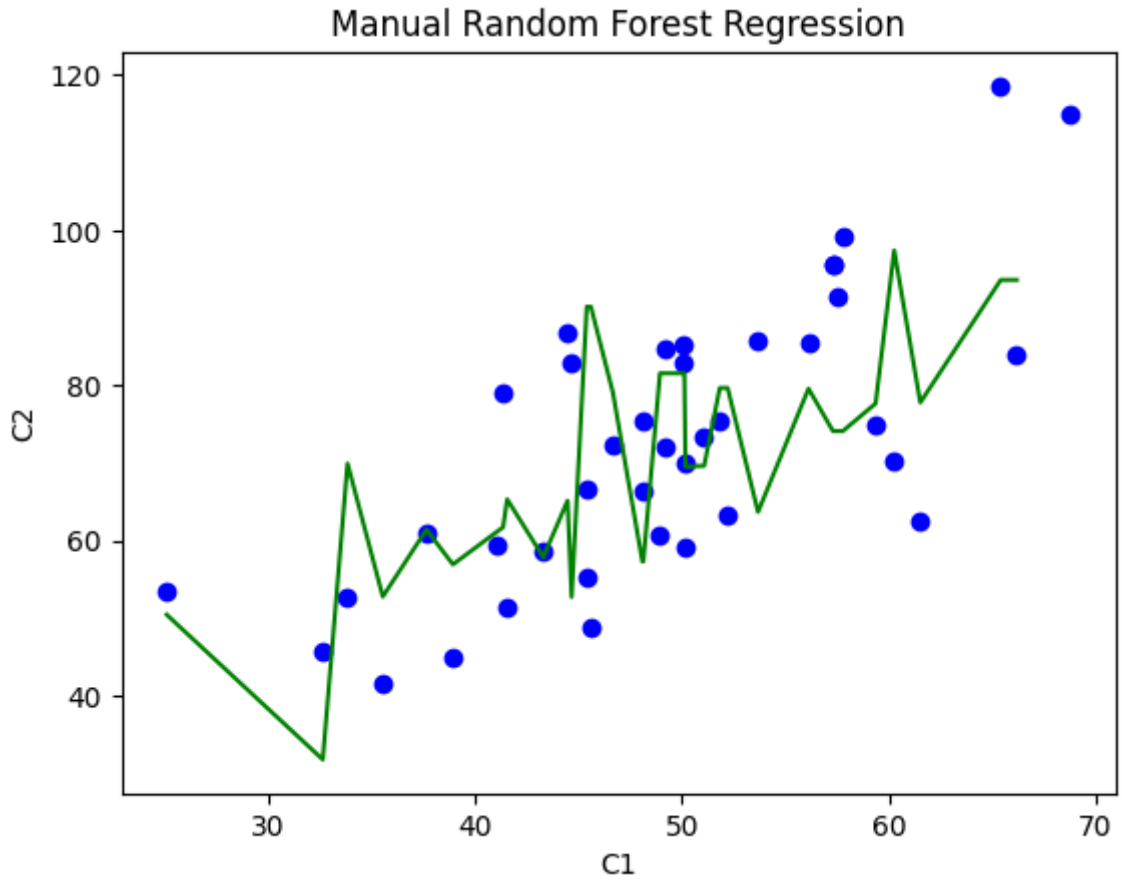
In [ ]:
```
# For Manual Method
# Output: The Regression Line Plot, Mean Absolute Error and RMSE
print("\n\nFOR RANDOM FOREST REGRESSION USING MANUAL METHOD")
print('\nMean Absolute Error: %.2f' % mean_absolute_error(Y_tsort, y_pred
print('\nRoot Mean Squared Error: %.2f' % mean_squared_error(Y_tsort, y_p
                                                    squared = Fa
plt.plot(X_tsort, y_pred, color = 'green')
plt.scatter(X_test, Y_test, color = 'blue')
plt.title('Manual Random Forest Regression')
plt.xlabel('C1')
plt.ylabel('C2')
```

```
plt.show()

# For SciKit-Learn Method
# Output: The Regression Line Plot, Mean Absolute Error and RMSE
print("_____
print("\nFOR RANDOM FOREST REGRESSION USING SCIKIT-LEARN METHOD")
print('\nMean Absolute Error: %.2f' % mean_absolute_error(Y_tsort, Y_pred
print('\nRoot Mean Squared Error: %.2f' % mean_squared_error(Y_tsort, Y_p
                                                          squared = Fa
plt.plot(X_tsort, Y_pred, color = 'green')
plt.scatter(X_test, Y_test, color = 'red')
plt.title('SciKit-Learn Random Forest Regression')
plt.xlabel('C1')
plt.ylabel('C2')
plt.show()
```

FOR RANDOM FOREST REGRESSION USING MANUAL METHOD

Mean Absolute Error: 18.56

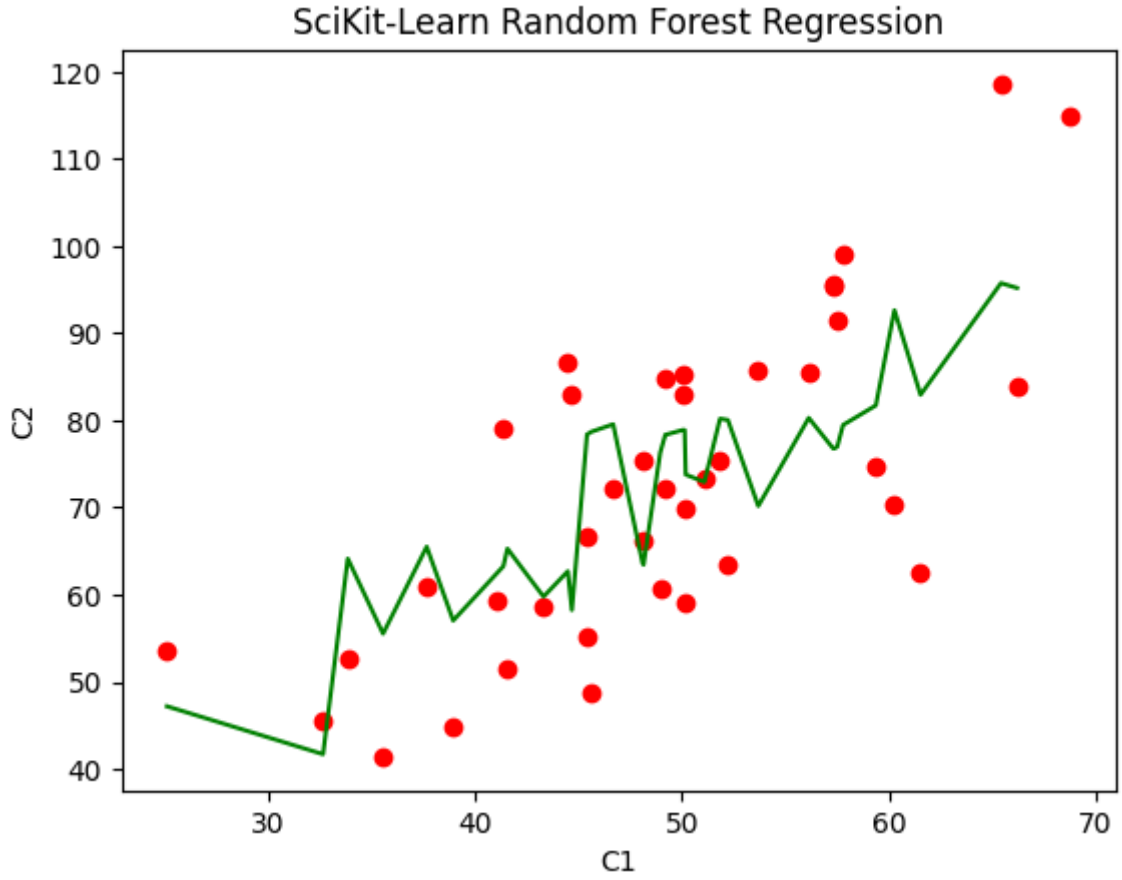
Root Mean Squared Error: 21.70



FOR RANDOM FOREST REGRESSION USING SCIKIT-LEARN METHOD

Mean Absolute Error: 17.88

Root Mean Squared Error: 20.86



On Comparison of both methods, we can see that the Random Forest Regression model using SciKit-Learn Method is more accurate as compared to the Random Forest Regression Model using Manual Method.

Assignment 6: Naive Bayes Classification.

Steps:

(i) Load the required Libraries.

(ii) Import the dataset.

(iii) Seperate data and target as X and Y

(iv) Split the dataset into testing and training set.

(v) Create / Load the Gaussian NB() Model.

(vi) Train the model over training dataset as

(a) Calculate the class Probability of each class in y-train

(b) Calculate the conditional Probability of each attribute or set of Attributes.

(vii) Predict the class of X-test as

(a) Calculate the conditional Probability for the set of attributes of X-test

(b) for each class:

p = conditional probability (X-test) * probability (class).

(c) Select the maximum p value class as the class of X-test.

(viii) Calculate the Accuracy, Confusion Matrix and classification Report for the model.

Exercise 6: Naïve Bayes Classification:

Dataset: You can use iris dataset (i.e. D2 dataset in exercise 2) or any dataset of your choice. Implement Naïve Bayes Classification algorithm and show the results. Split data into a train and a test split (70% and 30% respectively).

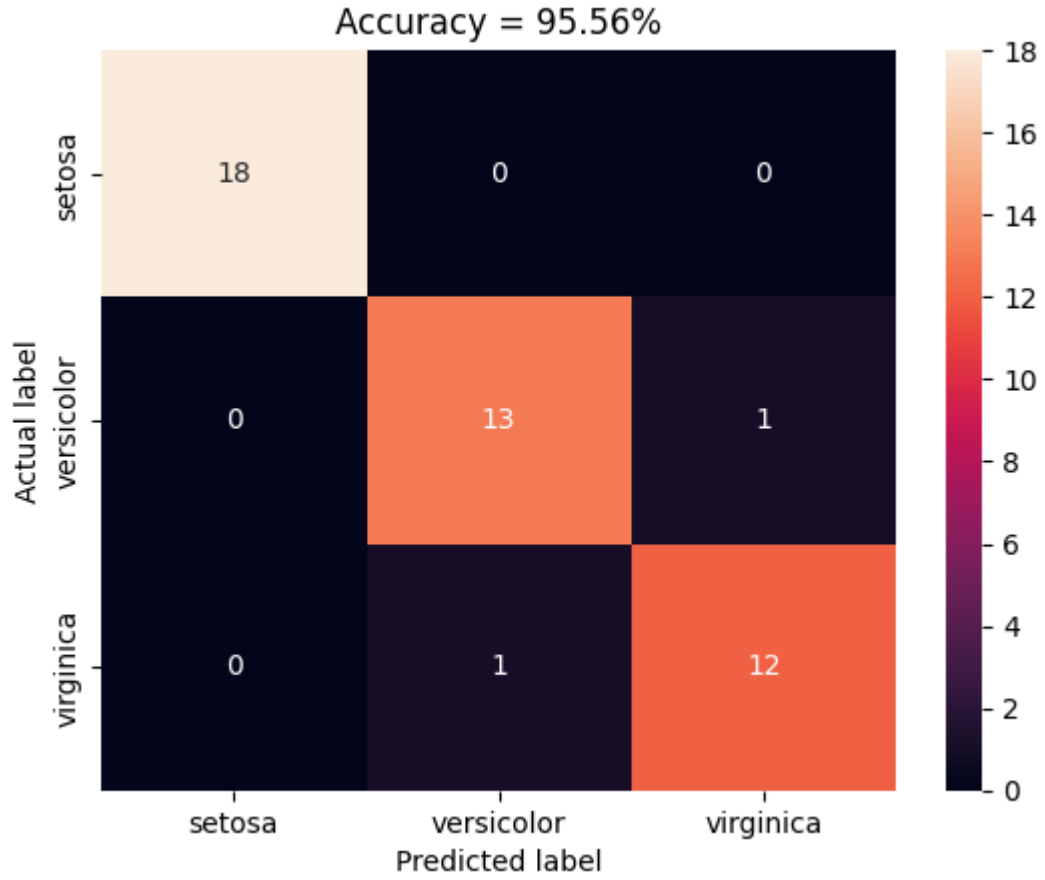
In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
import numpy as np
from matplotlib import pyplot as plt
from matplotlib import colors
from itertools import product
import pandas as pd
import seaborn as sns

# Input: Dataset
iris = load_iris()
X = iris.data
y = iris.target

# Splitting the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Create Gaussian Naive Bayes object
gnb = GaussianNB()

# Train the model using the training sets
gnb.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = gnb.predict(X_test)
```

Output for Naive Bayes Classification.

In [ ]:
```
# Output: The Predicted vs Actual Class, Confusion Matrix and Classificat
print("\n\nACCURACY OF GAUSSIAN NAIVE BAYES CLASSIFIER WITH ALL ATTRIBUTE
print("\nPredicted Class: \n")
print(*y_pred, sep=' ')
print("\nActual Class: \n")
print(*y_test, sep=' ')
print("\nNumber of mislabeled points out of a total %d points : %d"
      % (X_test.shape[0], (y_test != y_pred).sum()))
print("\nThe Confusion Matrix for the Gaussian Naive Bayes Model\n")
cm = metrics.confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm,
                     index = ['setosa','versicolor','virginica'],
                     columns = ['setosa','versicolor','virginica'])
sns.heatmap(cm_df, annot=True)
plt.title('Accuracy = {0:.2f}%'.format(metrics.accuracy_score(y_test, y_p
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\nThe Classification Report for the Gaussian Naive Bayes Model\n\n
      ,metrics.classification_report(y_test, y_pred))
print("\n_____
```

```
# Output: The Decision Boundary Plot for Pair of Attributes
Feature = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width' ]
print("\n\nPLOTS OF GAUSSIAN NAIVE BAYES MODEL WITH ANY TWO ATTRIBUTES")
for i in (0,1,2,3):
    for j in (0,1,2,3):
        if(j!=i):
            if(j<i):
                X = X_test[:, [i, j]]
                y = y_test
                gnb.fit(X,y)
                # Plotting decision regions
                x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
                y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
                xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                                     np.arange(y_min, y_max, 0.1))
                Z = gnb.predict(np.c_[xx.ravel(), yy.ravel()])
                Z = Z.reshape(xx.shape)
                plt.contourf(xx, yy, Z, alpha=0.4)
                plt.scatter(X[:, 0], X[:, 1], c=y, s=20, edgecolor='k')
                plt.xlabel('{}'.format(Feature[i]))
                plt.ylabel('{}'.format(Feature[j]))
                plt.title('Naive Bayes Classification for Feature {} vs F
                          .format(i,j))
                plt.show()
```

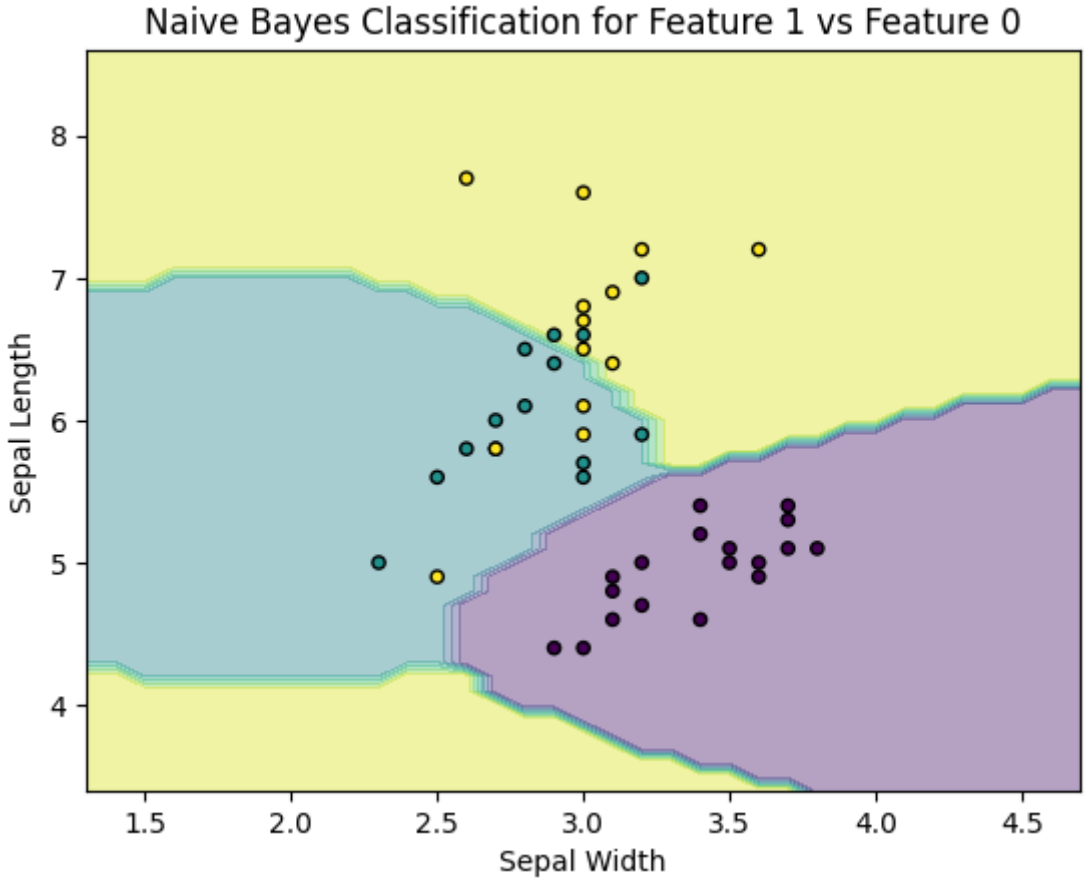
```
ACCURACY OF GAUSSIAN NAIVE BAYES CLASSIFIER WITH ALL ATTRIBUTES

Predicted Class:

0 0 1 1 1 0 0 1 1 1 0 2 1 0 2 0 0 1 0 0 2 2 2 2 0 0 0 1 2 0 0 2 0 1 1 2
2 2 0 2 2 0 1 1 1

Actual Class:

0 0 1 1 1 0 0 1 2 1 0 2 1 0 2 0 0 1 0 0 2 2 2 2 0 0 0 1 2 0 0 1 0 1 1 2
2 2 0 2 2 0 1 1 1

Number of mislabeled points out of a total 45 points : 2

The Confusion Matrix for the Gaussian Naive Bayes Model
```

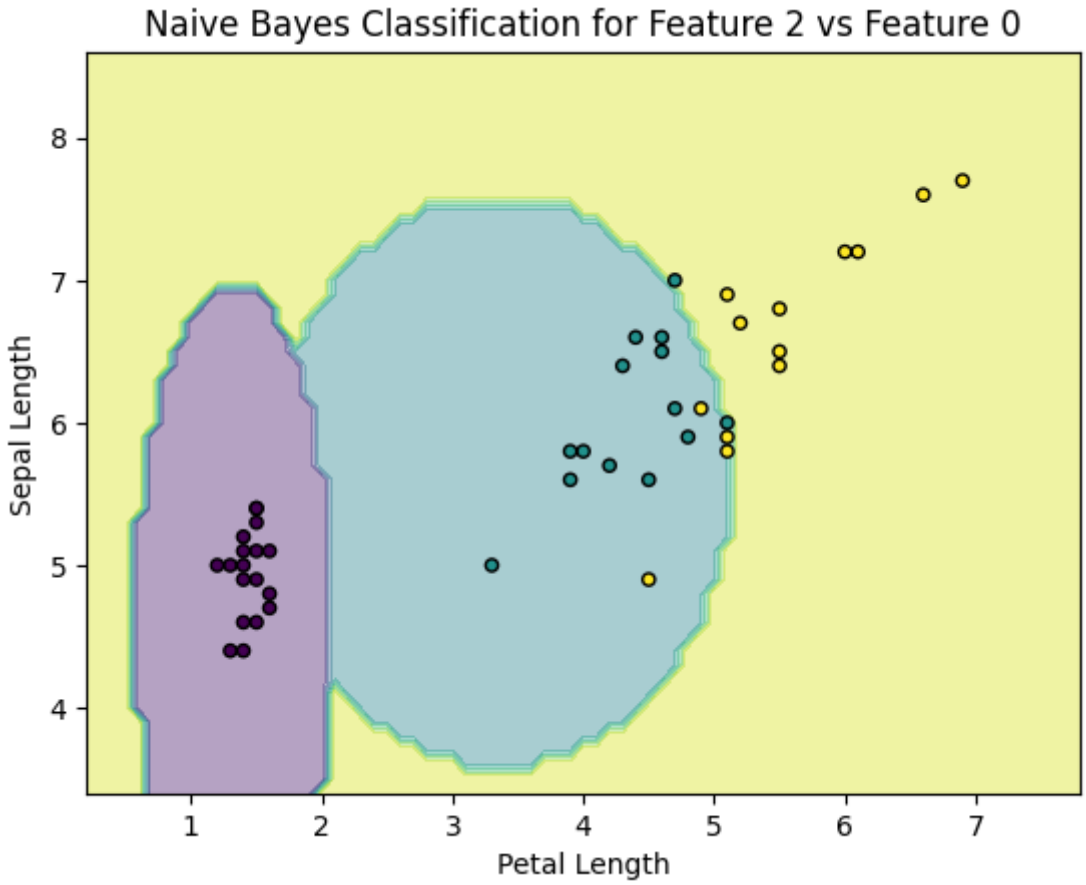


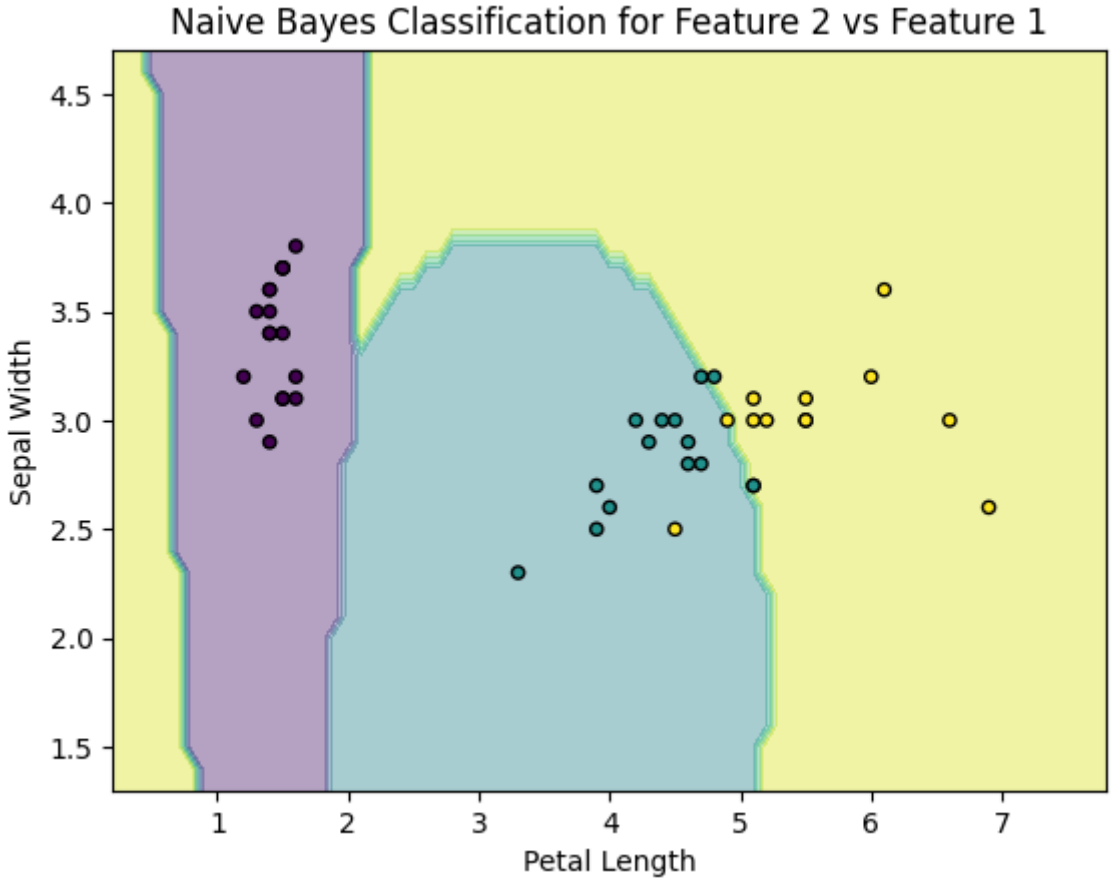
The Classification Report for the Gaussian Naive Bayes Model

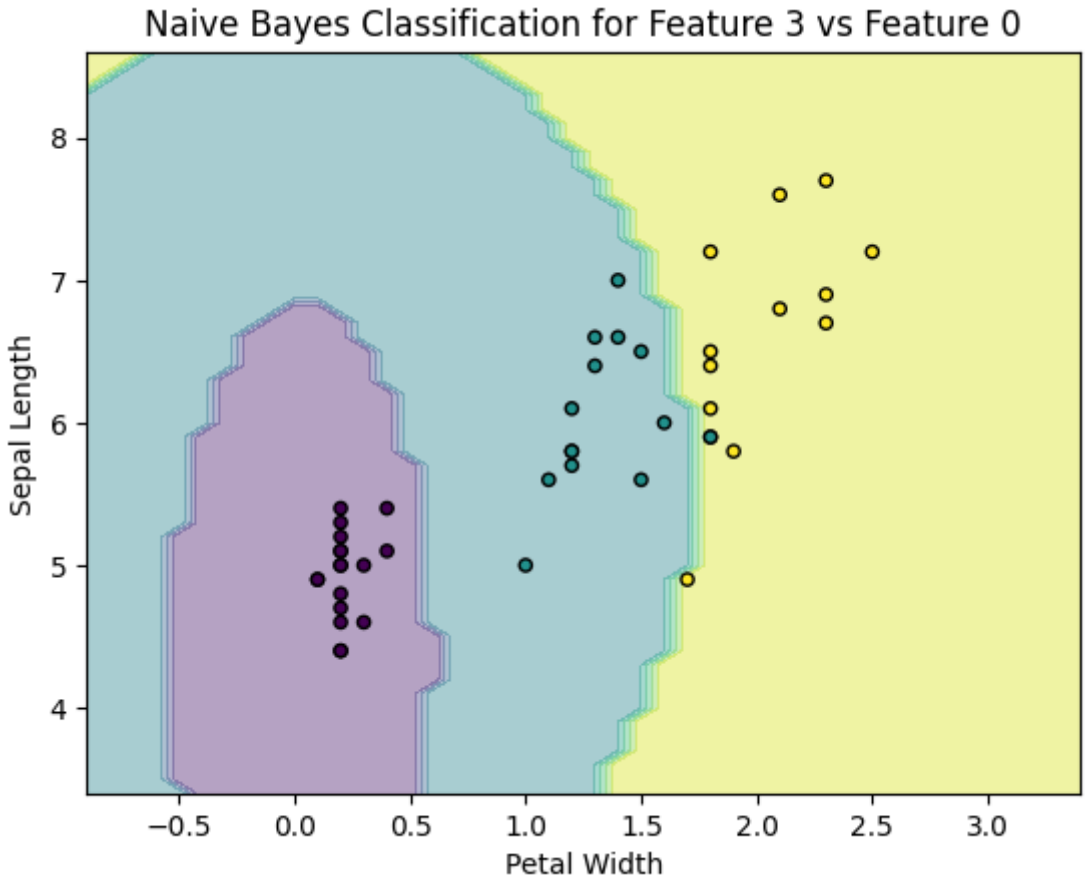
	precision	recall	f1-score	support
0	1.00	1.00	1.00	18
1	0.93	0.93	0.93	14
2	0.92	0.92	0.92	13
accuracy			0.96	45
macro avg	0.95	0.95	0.95	45
weighted avg	0.96	0.96	0.96	45

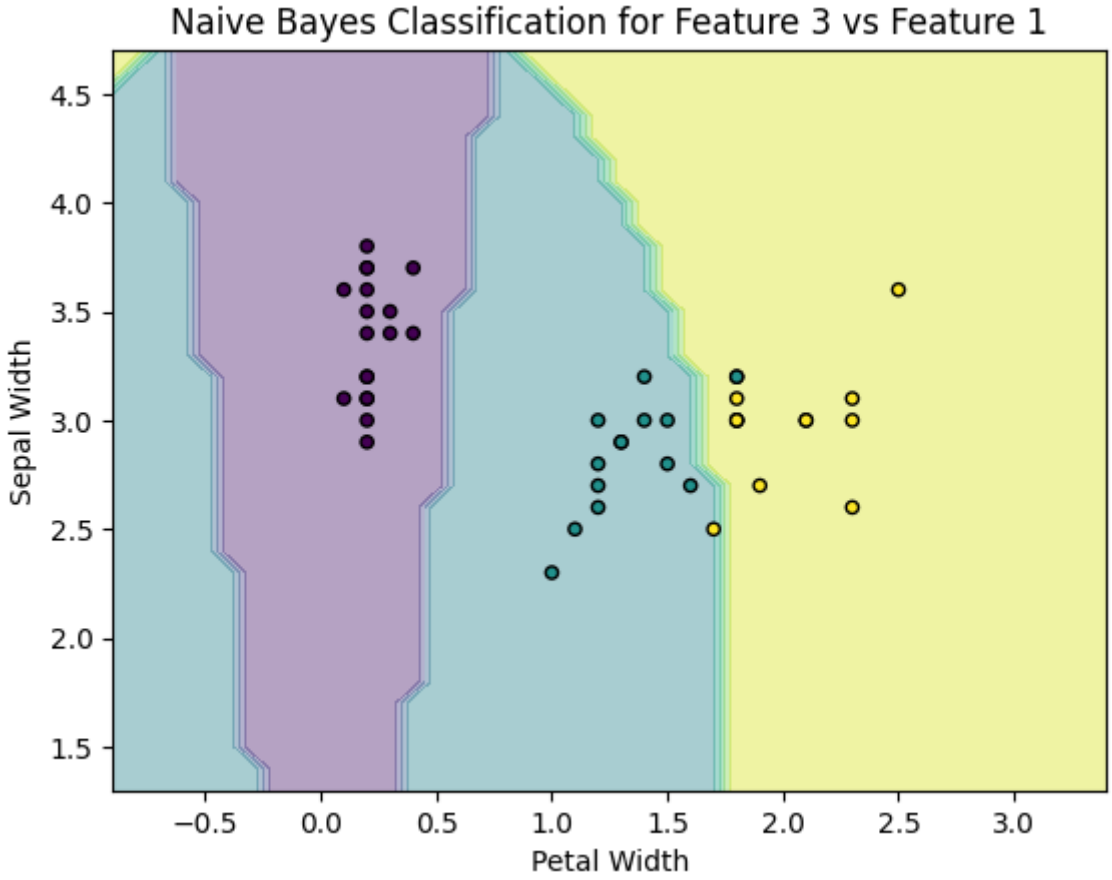
PLOTS OF GAUSSIAN NAIVE BAYES MODEL WITH ANY TWO ATTRIBUTES

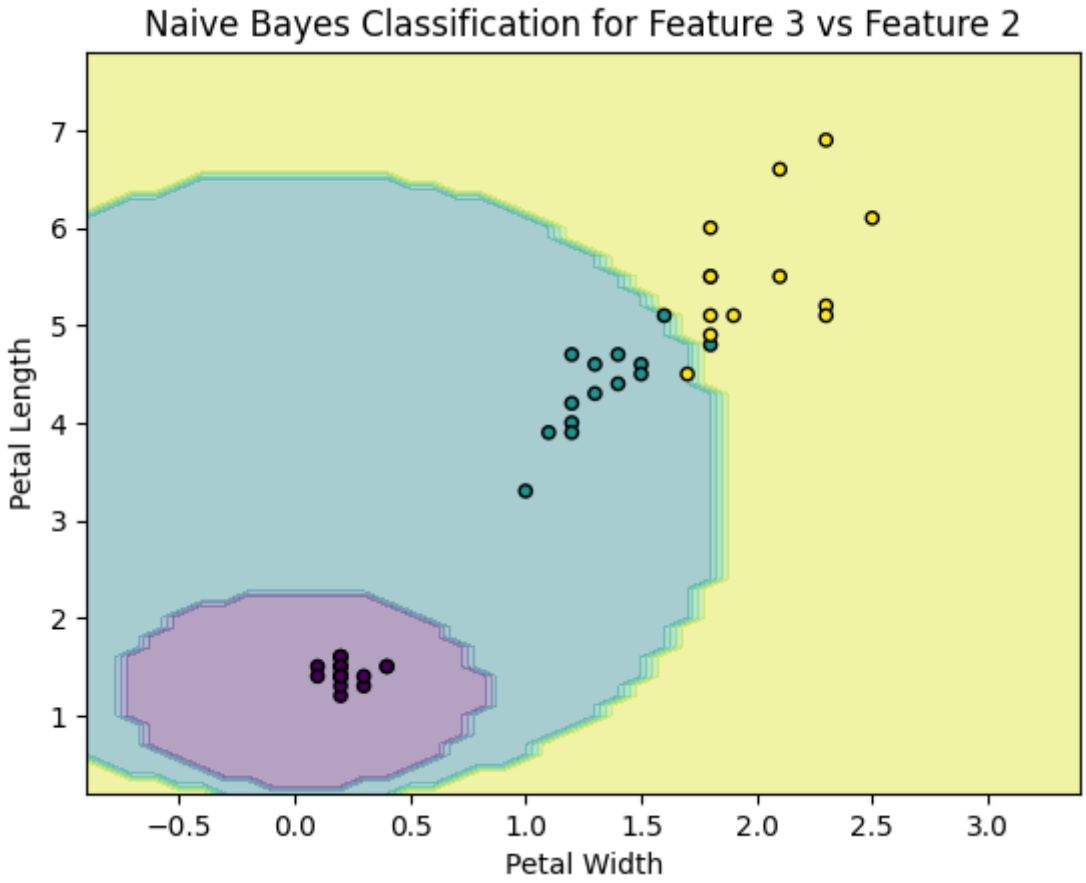

Naive Bayes Classification for Feature 1 vs Feature 0
Sepal Length
Sepal Width
8
7
6
5
4
1.5
2.0
2.5
3.0
3.5
4.0
4.5


Naive Bayes Classification for Feature 2 vs Feature 0
Sepal Length
Petal Length
8
7
6
5
4
1
2
3
4
5
6
7


Naive Bayes Classification for Feature 2 vs Feature 1
Sepal Width
Petal Length
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1
2
3
4
5
6
7


Naive Bayes Classification for Feature 3 vs Feature 0
Sepal Length
Petal Width
8
7
6
5
4
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0





To implement the Naive Bayes Classification Algorithm, Gaussian Naive Bayes was used over Iris Dataset with Train-Test Split of 70:30. The model gives an accuracy of 93.33% and is a good fit to predict the classes for the given Dataset.

The plot for the trained Gaussian Naive Bayes Model was plotted for taking any two attributes in consideration. It can be concluded from the plots that the results predicted

by the model are satisfactory.

Assignment 7: KNN Classification

Steps:

(i) Load the required library.

(ii) Import the data set

(iii) Separating features and target values as X & Y

(iv) Take Input the ratio of test_train split and value of k (Neighbors)

(v) Split the dataset into testing and training dataset using the given ratio.

(vi) For each iteration (length of testing dataset), i

(a) Generate Neighbour for X_test[i] using training dataset as follows:

* for each iteration (length of training dataset), j

(A) Calculate Euclidean distance b/w X_test[i] and X_train[j]

(B) Append (X_train[j], y_train[j], distance) in a list

* Sort the list and select top k datapoints.

(b) Count the frequency of target (category) in the list

(c) Assign the target with maximum frequency as the class of all datapoints in the list or the datapoint in the test dataset

(vii) Compute the Confusion Matrix, Classification Report and Accuracy.

Exercise 7: K-Nearest Neighbor (KNN) Classification:

Datasets:

1.Classification Datasets: You can use one of the two datasets (or optionally, both datasets).

(a) Iris dataset D2 (i.e. D2 dataset in exercise 2): Target attribute class :{Iris Setosa, Iris Versicolour, Iris Virginica}.

(b) Wine Quality dataset D3 : Target attribute quality:{0 to 10}.
https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Implement K-Nearest Neighbor (KNN) Classification:

Your task is to implement KNN Classification algorithm. To implement KNN you have to

• Split data into a train and a test split (70% and 30% respectively).

• Implement a similarity (or a distance) measure. To begin with you can implement the Euclidean Distance.

• Implement a function that returns top K Nearest Neighbors for a given query (data point).

• You should provide the prediction for a given query (use majority voting for classification).

• Measure the quality of your prediction. [Hint: You have to choose a quality criterion].

In [ ]:
```
import csv, random, math
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Function to Calculate Euclidean Distance
def euc(obj1, obj2, size):
    dist = 0
    for i in range(size):
        dist = dist + pow((obj1[i]-obj2[i]), 2)
    return math.sqrt(dist)

# Function to Return Top k Neighbours
def gen_nbors(X_train, y_train, X_test, k):
    dist = []
    nbors = []
    c = (len(X_test))
    for i in range(len(X_train)):
        d = euc(X_test, X_train[i], c)        # Calculating Euclidean Dis
        dist.append((X_train[i], y_train[i], d))
```

```
    dist = sorted(dist, key = lambda i: i[2]) # Sort datapoints
    for i in range(k):
        nbors.append(dist[i][0:2])            # Select Top k datapoints
    return nbors

# KNN Algorithm
def knn(nbors):
    temp = {}
    for i in range(k):
        pred = nbors[i][-1]
        if pred in temp:
            temp[pred] = temp[pred]+1
        else:
            temp[pred] = 1
    sorted_pred = list(temp.items());        # Sort the list in decreasin
    return sorted_pred[0][0]

# Input: Dataset
dataset = load_iris()

# Separating Features and Target Values
X = dataset.data
y = dataset.target

# Input: Size of Test Dataset and Number of Neighbours (k)
print("\n\nTaking Input Parameters\n")
te_size = input("Enter the Testing Data Size (as decimal ratio): ")
te_size = float(te_size)
k = input("Enter the Value of k: ")
k = int(k)

# Splitting the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = te_

predicted = []
print('\nNumber of Training data samples: '+str(len(X_test)))

for i in range(len(X_test)):
    nbors = gen_nbors(X_train, y_train, X_test[i], k);
    predd= knn(nbors)
    predicted.append(predd)
```

```
Taking Input Parameters


Number of Training data samples: 75
```

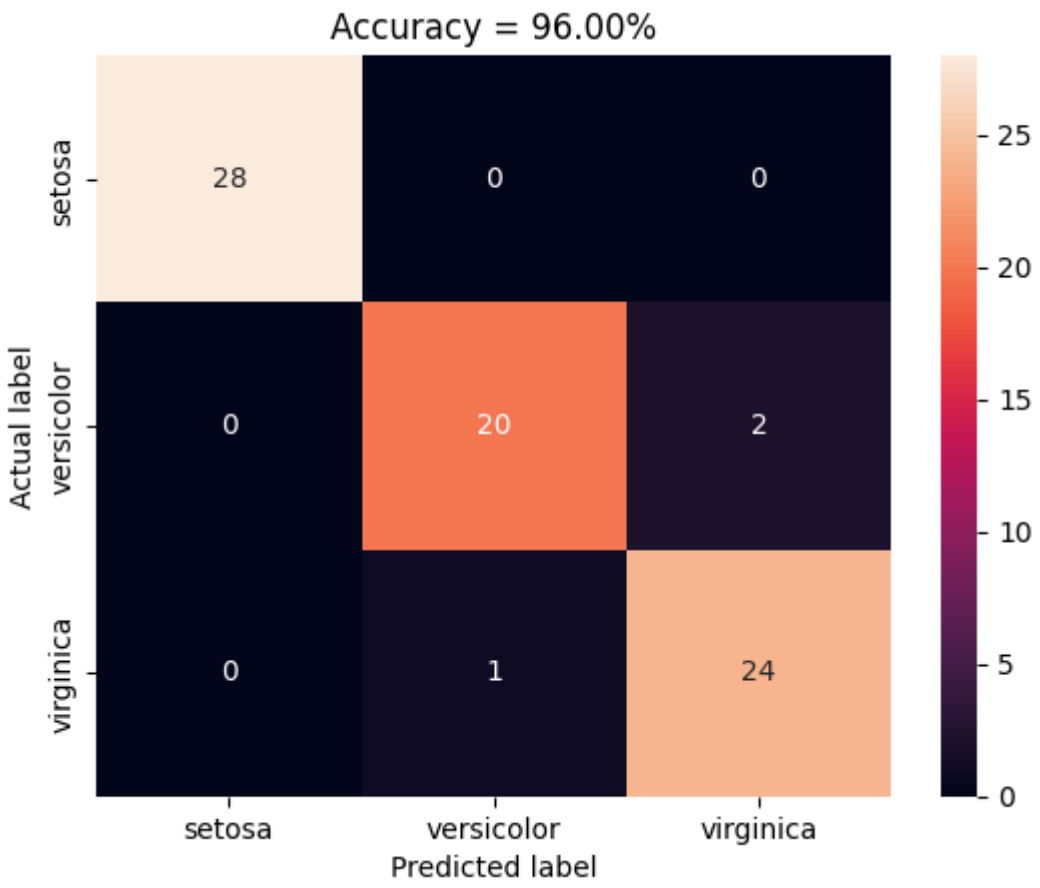
Output for K Nearest Neighbour Classification.

In [ ]:

```
# Output: The Predicted vs Actual Class, Confusion Matrix & Classificatio
print("\n\nACCURACY METRIC OF K NEAREST NEIGHBOUR CLASSIFIER")
print("\nPredicted Class: \n")
print(*predicted, sep=' ')
print("\nActual Class: \n")
print(*y_test, sep=' ')
print("\nNumber of mislabeled points out of a total %d points : %d"
      % (X_test.shape[0], (y_test != predicted).sum()))
print("\nThe Confusion Matrix for the K Nearest Neighbour Model\n\n")
cm = metrics.confusion_matrix(y_test, predicted)
```

```
cm_df = pd.DataFrame(cm,
                     index = ['setosa','versicolor','virginica'],
                     columns = ['setosa','versicolor','virginica'])
sns.heatmap(cm_df, annot=True)
plt.title('Accuracy = {0:.2f}%'.format(metrics
                                       .accuracy_score(y_test, predicted)*
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\nThe Classification Report for the K Nearest Neighbour Model\n\n"
      ,metrics.classification_report(y_test, predicted))
```

```
ACCURACY METRIC OF K NEAREST NEIGHBOUR CLASSIFIER

Predicted Class:

0 0 2 2 0 2 1 0 2 2 1 2 1 1 0 1 1 2 0 1 0 0 2 2 2 2 0 2 0 0 1 0 0 2 2 0
2 1 2 0 0 0 2 0 0 2 0 2 1 2 1 0 1 1 0 1 1 2 2 0 2 0 0 1 1 2 2 0 1 0 1 2
1 0 1

Actual Class:

0 0 2 2 0 2 1 0 2 2 1 2 1 1 0 1 1 2 0 1 0 0 2 2 2 2 0 2 0 0 1 0 0 2 2 0
2 1 1 0 0 0 2 0 0 2 0 2 1 2 1 0 1 1 0 1 2 2 2 0 1 0 0 1 1 2 2 0 1 0 1 2
1 0 1

Number of mislabeled points out of a total 75 points : 3

The Confusion Matrix for the K Nearest Neighbour Model
```



```
The Classification Report for the K Nearest Neighbour Model

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        28
           1       0.95      0.91      0.93        22
           2       0.92      0.96      0.94        25

    accuracy                           0.96        75
   macro avg       0.96      0.96      0.96        75
weighted avg       0.96      0.96      0.96        75
```

Predicting Class for User Query.

In [ ]:
```
# To take query from User
lst = []

# Input: Feature Values for Target Prediction
print("\nEnter Feature Values (Sepal Width,Sepal Length,Petal Width,Petal
for i in range(0, 4):
    ele = float(input())
    lst.append(ele)

# Making Prediction
nbors= gen_nbors(X_train, y_train, lst, k)
predd= knn(nbors)
if(predd<=0.5):
    predd1=0
elif(predd<=1.5):
    predd1=1
else:
    predd1=2

# Output: Predicted Class
print("The Predicted Class for Feature Set ",lst," is ", predd1,".")
```

```
Enter Feature Values (Sepal Width,Sepal Length,Petal Width,Petal Length)
The Predicted Class for Feature Set  [2.0, 2.0, 2.0, 2.0]  is  0 .
```

The KNN Classifier Model gives an accuracy of 93.33% for Iris Dataset (70% Training Data and 30% Testing Data) when trained with value of k=5. The model can be used to predict the class for a given set of features, with the help of query code built.

Assignment 8: K means Clustering

Steps:

(i) Load the required libraries

(ii) Import the dataset

(iii) Take any two attributes of Dataset

(iv) Store the Actual target labels in a separate list

(v) Encode the target labels in a separate list

(vi) Scikit-learn Method:

(a) Load the kMeans() Model with maximum iterations and no. of clusters as desired.

(b) Train the model over training dataset

(c) Get the centroids of each cluster.

(d) To get the optimum value of k, calculate intertia.

(vii) Manual Method:

(a) For each iteration, Initialize by taking k random centroids.

(A) Assign each datapoint to their closest centroid

(B) Calculate the variance and assign a new centroid for each cluster.

(C) Reassign each datapoint to the new closest

(D) If any reassignment occurs, repeat step(B) else final centroids are obtained.

(b) To get the optimum value of k, perform Elbow Method.

(viii) Compare results for both methods.

Exercise 8: Implement K Means clustering algorithm

Document dataset:

(a) IRIS dataset D4: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass/iris.scale

(b) rcv1v2 (topics; subsets D5:https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel.html#rcv1v2(topics subsets)

1: Implement K Means clustering algorithm (using any software library). You should use D4 or D5 datasets. Also, you should also choose a criterion for selecting an optimal value of k (number of clusters).

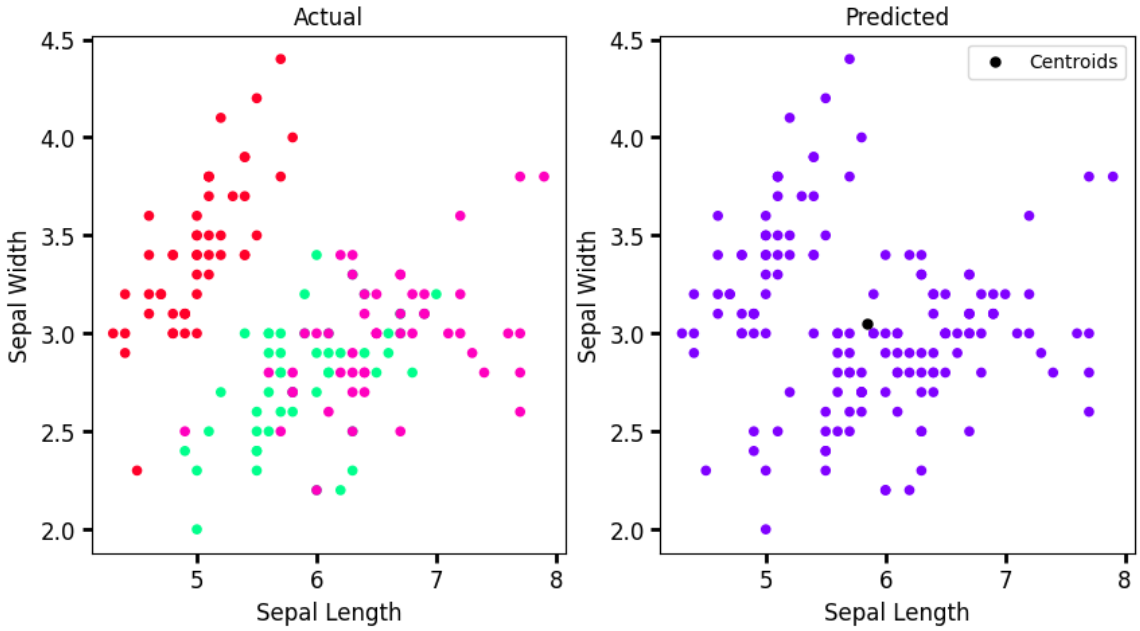
2: Cluster the data set D4 or D5 using your own implementation of K-means algorithm. Show the results.

In [ ]:

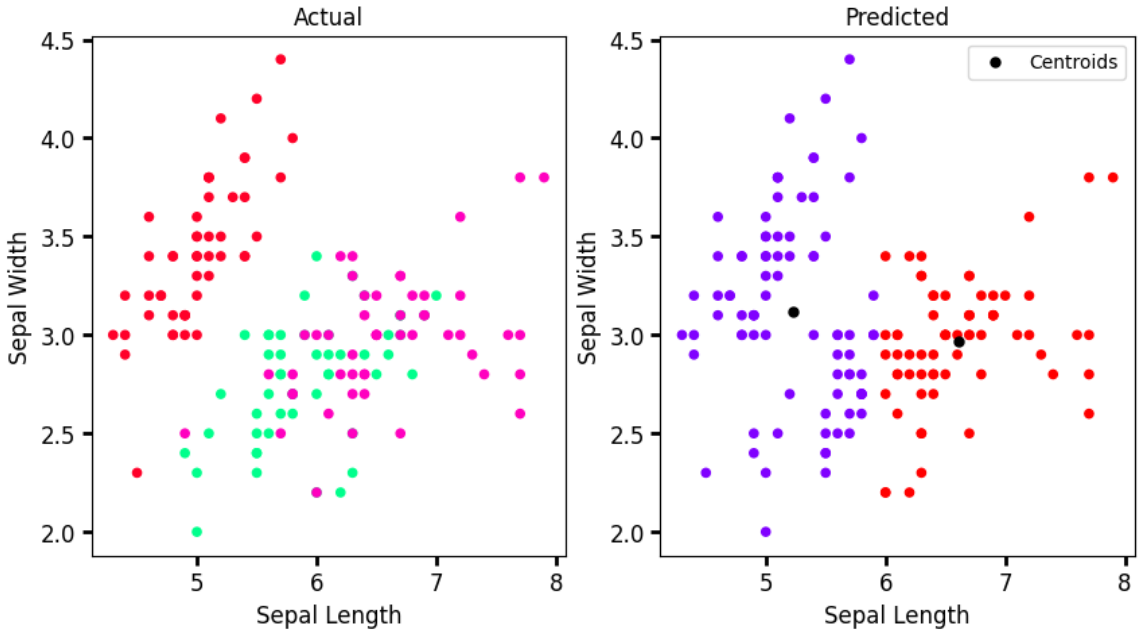
```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import warnings
warnings.filterwarnings('ignore')
import matplotlib.cm as cm

# Input: Dataset
df = pd.read_csv('iris.data')

iris_datasets = np.array(df)

# Taking only two attributes of the Dataset
test_x = iris_datasets[:,[0,1]]

# Storing the Actual Target value in a new Array
actuallabel = iris_datasets[:,-1:]

for iters in range(0,149):
    if actuallabel[iters] == 'Iris-setosa':
        actuallabel[iters] = 0
    elif actuallabel[iters] == 'Iris-versicolor':
        actuallabel[iters] = 1
    elif actuallabel[iters] == 'Iris-virginica':
        actuallabel[iters] = 2

Actual_Value = []
for iters in range(len(actuallabel)):
    Actual_Value.append(actuallabel[iters][0])

# K Means Clustering Algorithm
sse = []
for k in range(1, 11):                # Running the Model for different v
```

```
    # Create K Means Clustering object and Train it over the Dataset
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(test_x)

    # Centers obtained after training of model
    centers = kmeans.cluster_centers_

    # Output: The Cluster Centers obtained
    print("\n\nFor k = ",k,"\n")
    for k in range(k):
        print("  Center ",k+1," = ",centers[k])

    # Calculating the sum of distances of samples to their closest cluste
    sse.append(kmeans.inertia_)

    # Output: The Actual Cluster Plot vs Predicted Cluster Plot for given
    fig, axes = plt.subplots(1, 2, figsize=(10,5))
    axes[0].scatter(test_x[:, 0], test_x[:, 1], c=actuallabel,
                    cmap='gist_rainbow', s=20)
    axes[1].scatter(test_x[:, 0], test_x[:, 1], c=kmeans.labels_,
                    cmap='rainbow', s=20)
    axes[0].set_xlabel('Sepal Length', fontsize=12)
    axes[0].set_ylabel('Sepal Width', fontsize=12)
    axes[1].set_xlabel('Sepal Length', fontsize=12)
    axes[1].set_ylabel('Sepal Width', fontsize=12)

    axes[0].tick_params(direction='out', length=5, width=2,
                        colors='k', labelsize=12)
    axes[1].tick_params(direction='out', length=5, width=2,
                        colors='k', labelsize=12)
    axes[0].set_title('Actual', fontsize=12)
    axes[1].set_title('Predicted', fontsize=12)
    plt.scatter(kmeans.cluster_centers_[:, 0],
                kmeans.cluster_centers_[:,1],
                s = 25, c = 'Black', label = 'Centroids')
    plt.legend()
    plt.show()

# Output: The Elbow Method Curve wrt Inertia
K_array=np.arange(1,11,1)
plt.figure(figsize=(10,5))
plt.plot(K_array,sse)
plt.xlim(0, 10)
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("The Elbow Method Curve to find Optimum k")
plt.show()
```

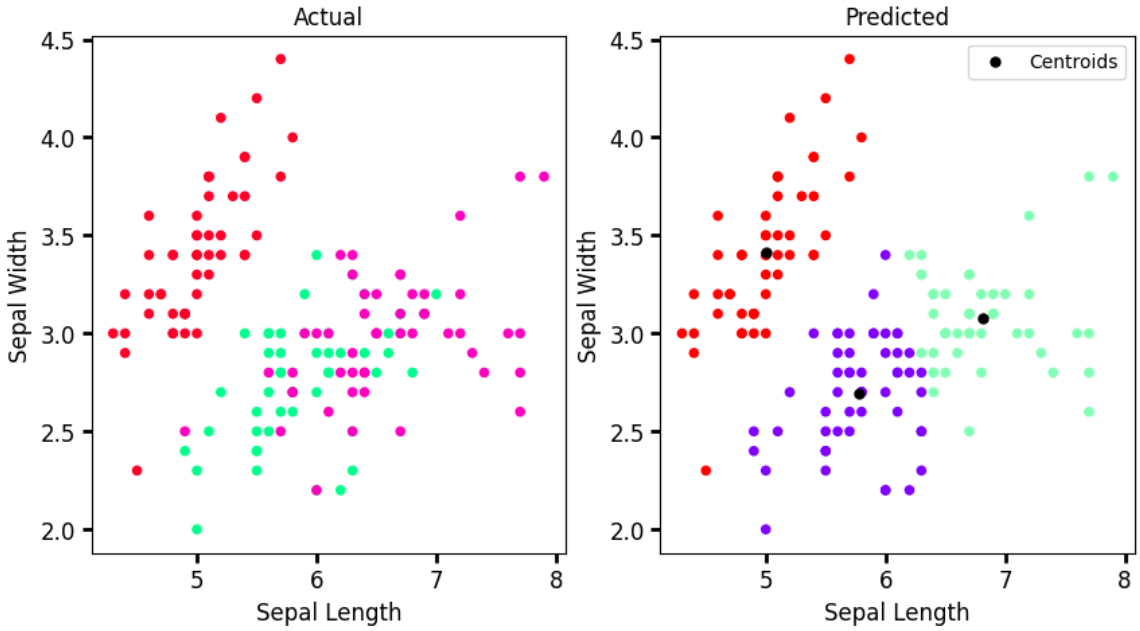
```
For k =  1

  Center  1  =  [5.84832215 3.05100671]
```



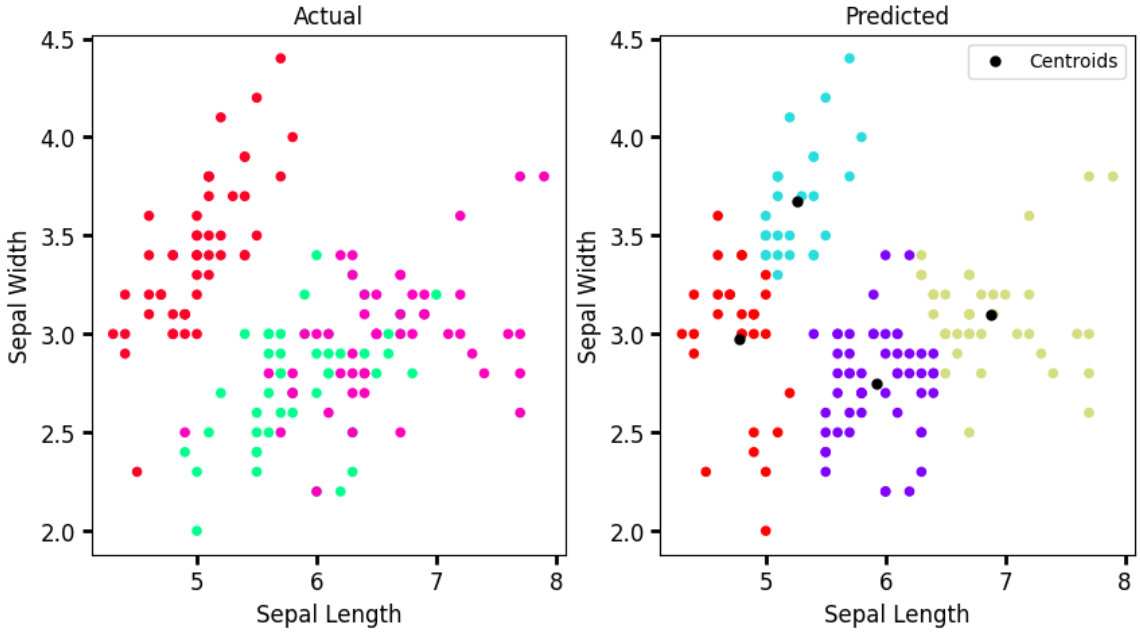

```
For k =  2

  Center  1  =  [5.22560976 3.12073171]
  Center  2  =  [6.61044776 2.96567164]
```



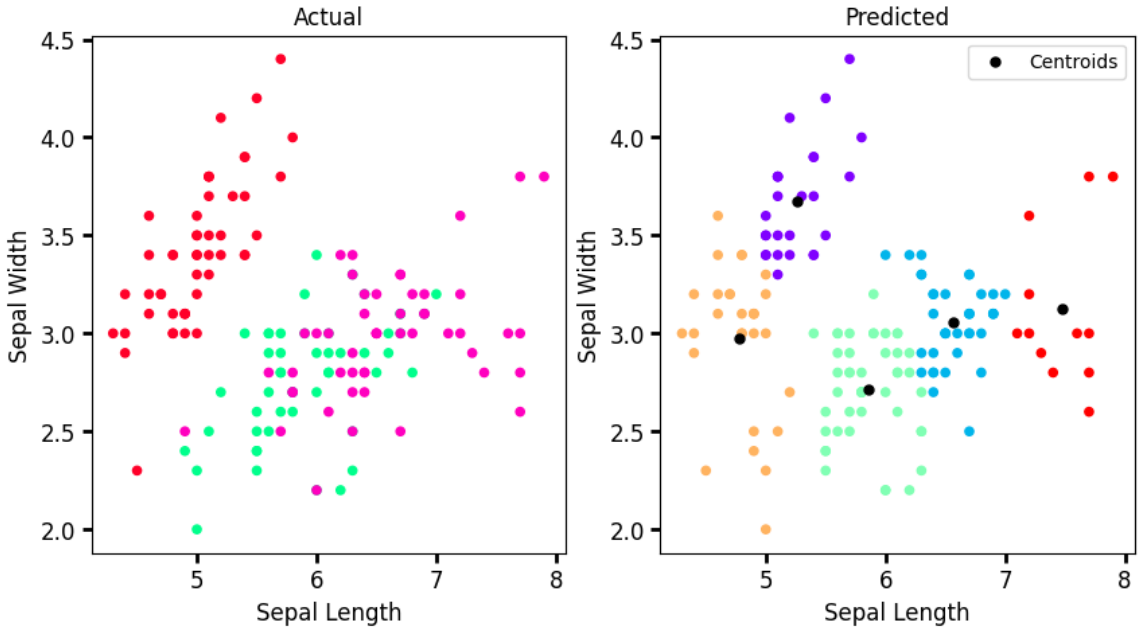

```
For k =  3

  Center  1  =  [5.77358491 2.69245283]
  Center  2  =  [6.81276596 3.07446809]
  Center  3  =  [5.00408163 3.41632653]
```



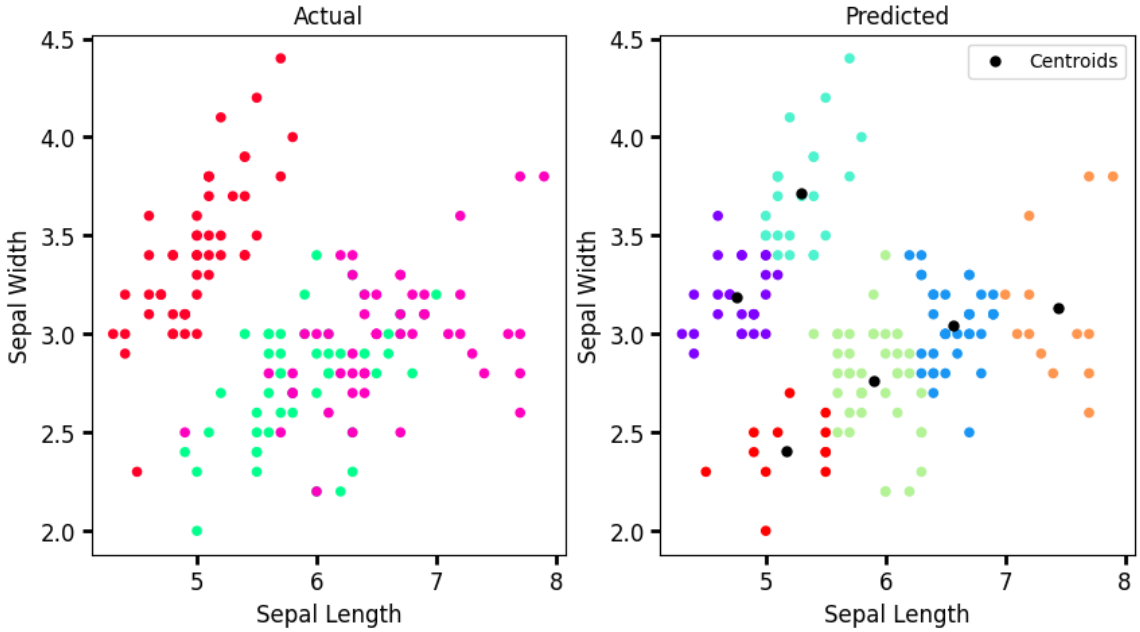

```
For k =  4

  Center  1  =  [5.9245283 2.7509434]
  Center  2  =  [5.26153846 3.67692308]
  Center  3  =  [6.8804878  3.09756098]
  Center  4  =  [4.77586207 2.97241379]
```



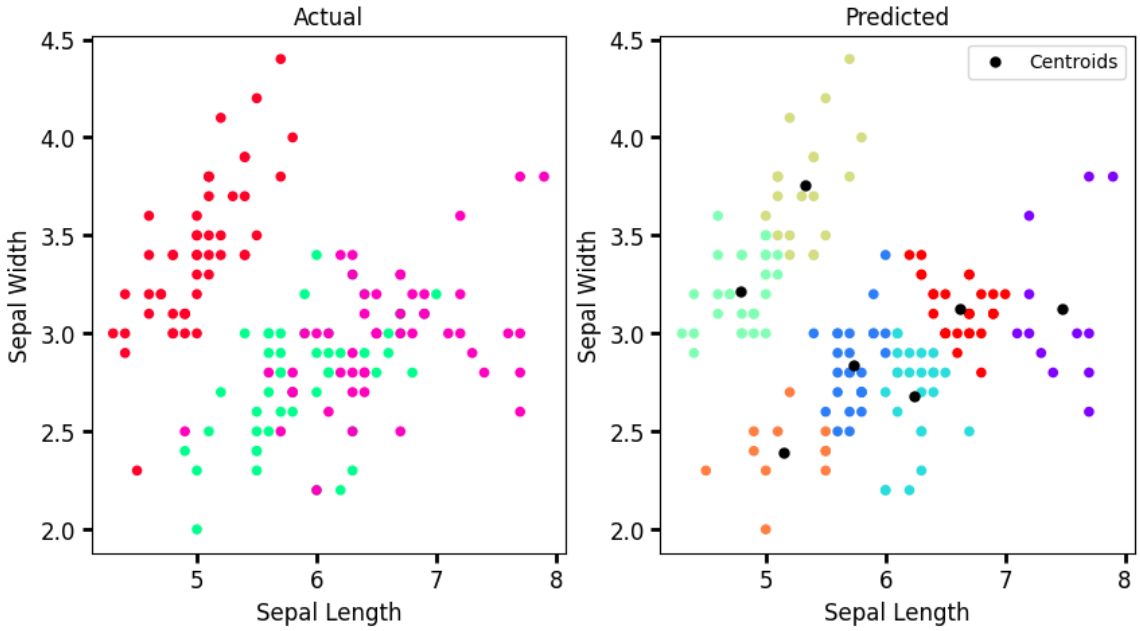

```
For k =  5

  Center  1  =  [5.26153846 3.67692308]
  Center  2  =  [6.56216216 3.05945946]
  Center  3  =  [5.85777778 2.71333333]
  Center  4  =  [4.77586207 2.97241379]
  Center  5  =  [7.475 3.125]
```




```
For k =  6

  Center  1  =  [4.76  3.184]
  Center  2  =  [6.56571429 3.04571429]
  Center  3  =  [5.29130435 3.7173913 ]
  Center  4  =  [5.90487805 2.76341463]
  Center  5  =  [7.43846154 3.13076923]
  Center  6  =  [5.175      2.40833333]
```

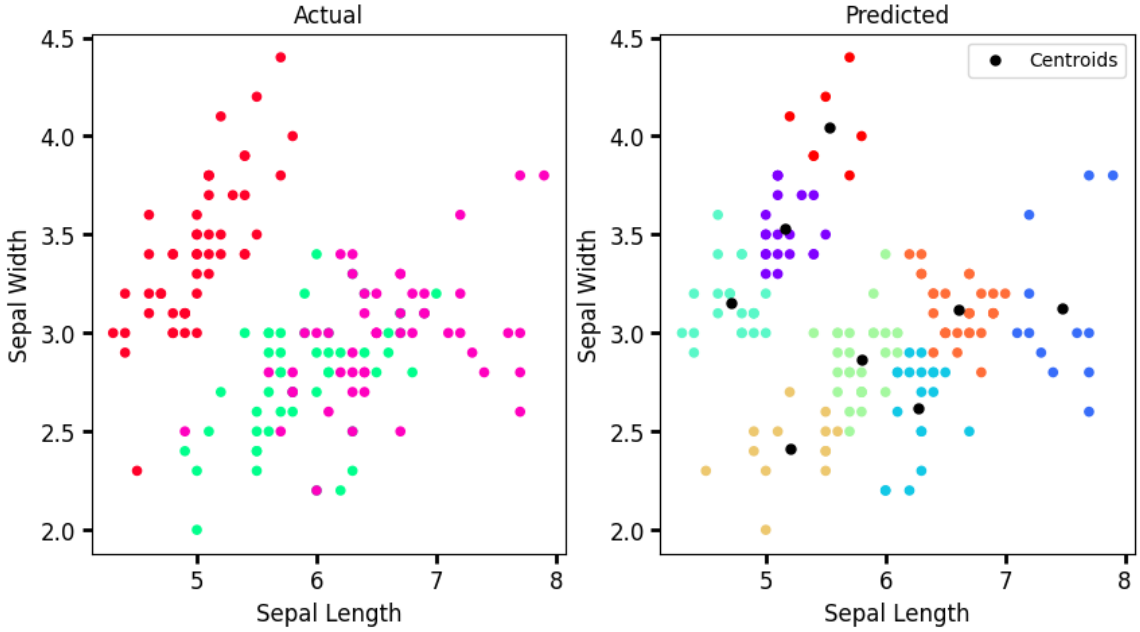



```
For k =  7

  Center  1  =  [7.475 3.125]
  Center  2  =  [5.73846154 2.83846154]
  Center  3  =  [6.24166667 2.67916667]
  Center  4  =  [4.78928571 3.21428571]
  Center  5  =  [5.33  3.755]
  Center  6  =  [5.14545455 2.39090909]
  Center  7  =  [6.62142857 3.12857143]
```

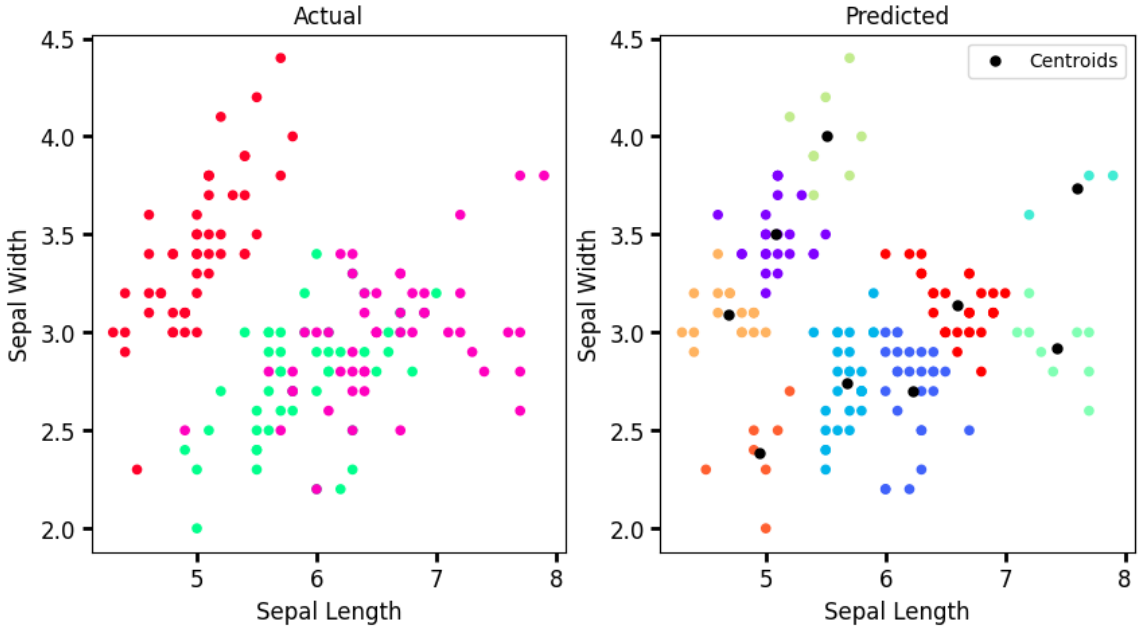


For k = 8

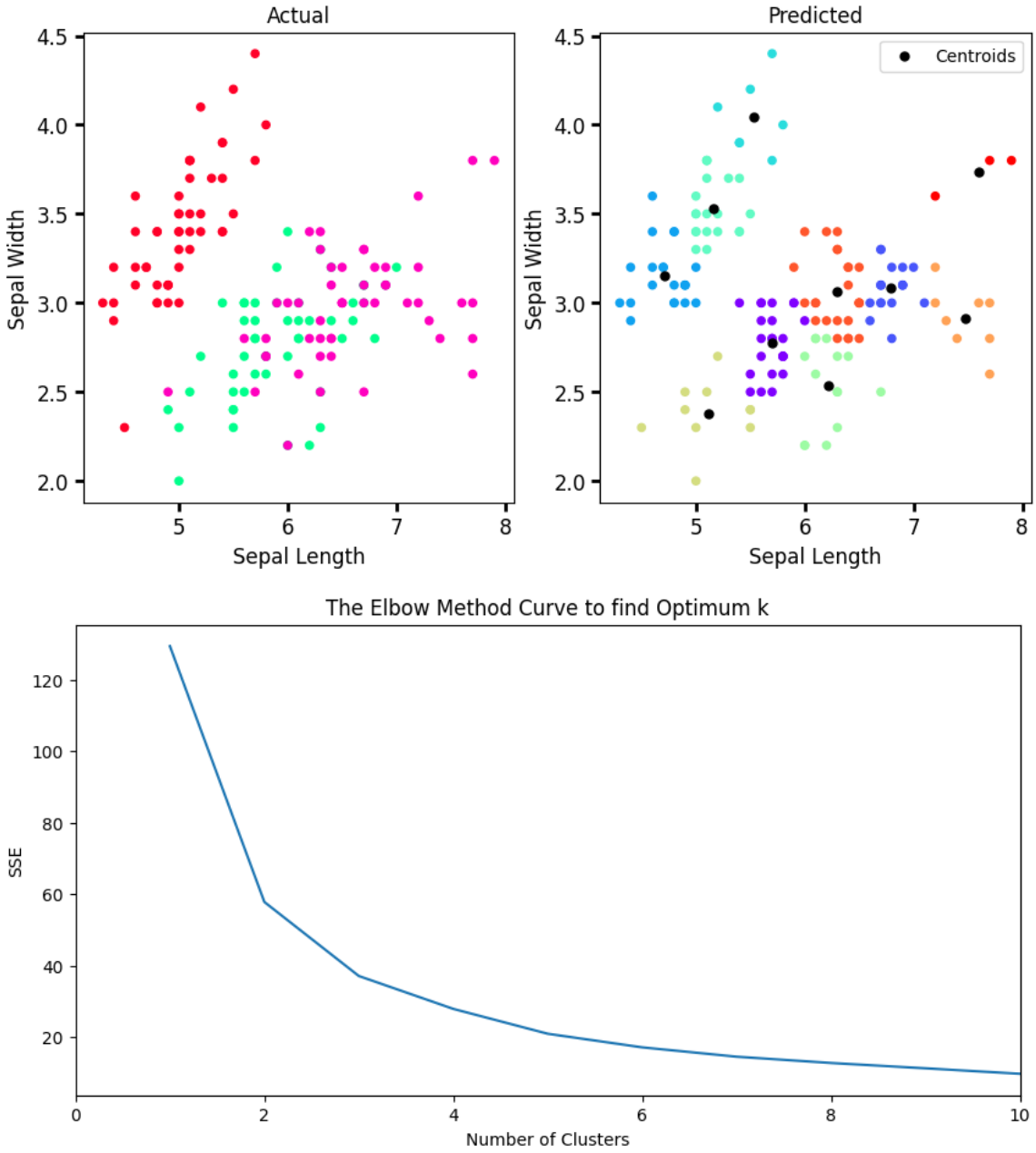
```
Center  1  =  [5.155 3.53 ]
Center  2  =  [7.475 3.125]
Center  3  =  [6.26842105 2.62105263]
Center  4  =  [4.70952381 3.15238095]
Center  5  =  [5.8        2.86785714]
Center  6  =  [5.20769231 2.41538462]
Center  7  =  [6.6137931  3.12068966]
Center  8  =  [5.52857143 4.04285714]
```



```
For k =  9

  Center  1  =  [5.0826087 3.5      ]
  Center  2  =  [6.22307692 2.7       ]
  Center  3  =  [5.67407407 2.74444444]
  Center  4  =  [7.6        3.73333333]
  Center  5  =  [7.43333333 2.92222222]
  Center  6  =  [5.5125 4.    ]
  Center  7  =  [4.68823529 3.09411765]
  Center  8  =  [4.94285714 2.38571429]
  Center  9  =  [6.6        3.13793103]
```




```
For k =  10

  Center  1  =  [5.7        2.77916667]
  Center  2  =  [6.78888889 3.08333333]
  Center  3  =  [4.70952381 3.15238095]
  Center  4  =  [5.52857143 4.04285714]
  Center  5  =  [5.155 3.53 ]
  Center  6  =  [6.21428571 2.53571429]
  Center  7  =  [5.11 2.38]
  Center  8  =  [7.475  2.9125]
  Center  9  =  [6.29166667 3.0625    ]
  Center  10  =  [7.6        3.73333333]
```



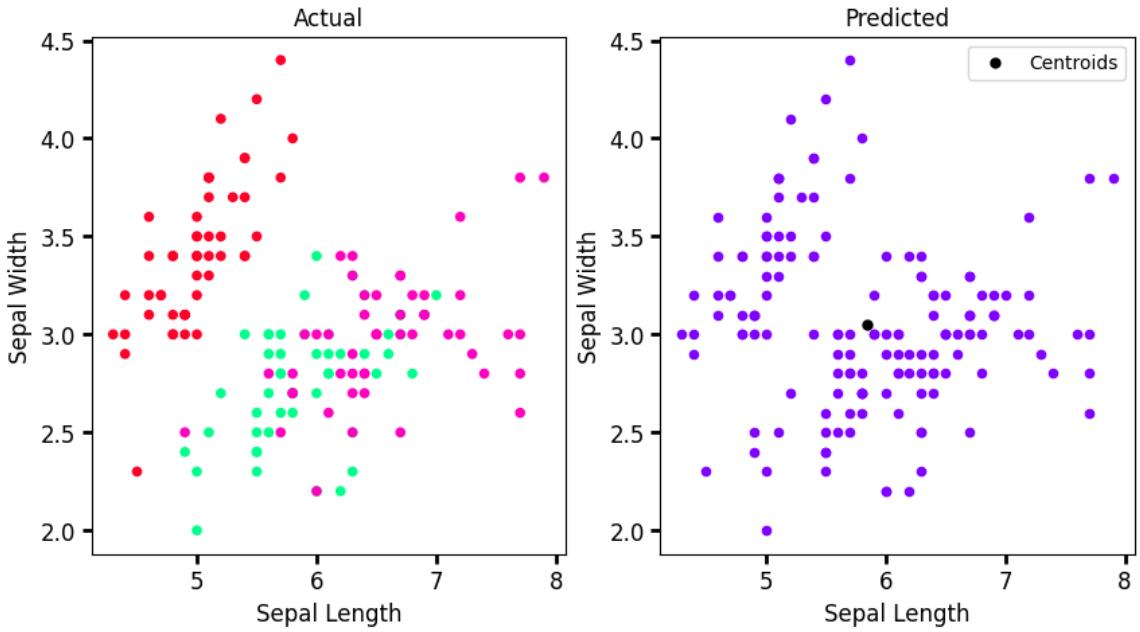
Python Program for K Means Clustering (Manual).

In [ ]:

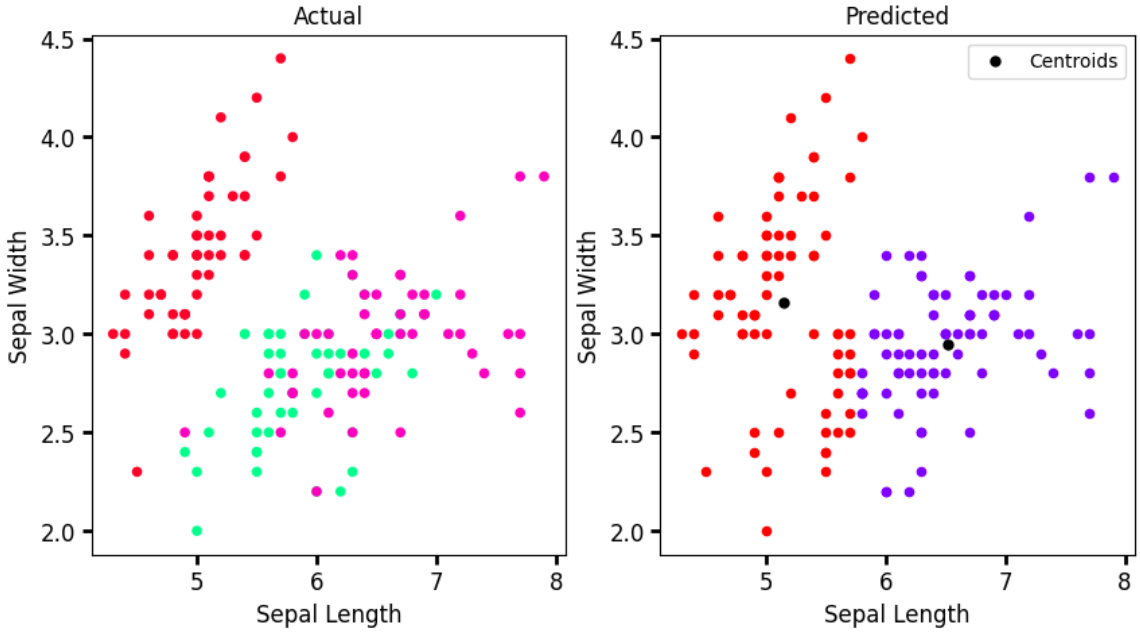
```
# Input: Dataset
dataset = pd.read_csv('iris.data')

# Taking only two attributes of the Dataset
X = dataset.iloc[:, [0, 1]].values

m=X.shape[0] # Number of Training Examples
n=X.shape[1] # Number of Features
n_iter=1000

# K Means Clustering Algorithm
WCSS_array=[]
for K in range(1,11):              # Model training for different k values
    Centroids=np.array([]).reshape(n,0)
    for i in range(K):
        rand=random.randint(0,m-1)
        # Taking Random Samples as Initial Centroids
        Centroids=np.c_[Centroids,X[rand]]
    Output={}
```

```
    for i in range(n_iter):
        EuclidianDistance=np.array([]).reshape(m,0)
        for k in range(K):
            tempDist=np.sum((X-Centroids[:,k])**2,axis=1)
            # Calculating Euclidean Distance
            EuclidianDistance=np.c_[EuclidianDistance,tempDist]
        C=np.argmin(EuclidianDistance,axis=1)+1
        Y={}
        for k in range(K):
            Y[k+1]=np.array([]).reshape(2,0)
        for i in range(m):
            Y[C[i]]=np.c_[Y[C[i]],X[i]]
        for k in range(K):
            Y[k+1]=Y[k+1].T
        for k in range(K):
            Centroids[:,k]=np.mean(Y[k+1],axis=0)
        Output=Y                # Final Centers
    ys = [str(i) for i in range(K)]
    colors = cm.rainbow(np.linspace(0, 1, len(ys)))

    # Output: The Cluster Centers obtained
    print("\n\nFor K = ",K)
    for k in range(K):
        print("  Center ",k+1," = ",' '.join(map(str, Centroids[:,k])))

    # Output: The Actual Clusters vs Predicted Clusters for given value o
    fig, axes = plt.subplots(1, 2, figsize=(10,5))
    axes[0].scatter(test_x[:, 0], test_x[:, 1],
                    c=actuallabel, cmap='gist_rainbow', s=20)
    for k in range(K):
        axes[1].scatter(Output[k+1][:,0],
                        Output[k+1][:,1], color=colors[k], s=20)
    axes[0].set_xlabel('Sepal Length', fontsize=12)
    axes[0].set_ylabel('Sepal Width', fontsize=12)
    axes[1].set_xlabel('Sepal Length', fontsize=12)
    axes[1].set_ylabel('Sepal Width', fontsize=12)
    axes[0].tick_params(direction='out', length=5, width=2, colors='k', l
    axes[1].tick_params(direction='out', length=5, width=2, colors='k', l
    axes[0].set_title('Actual', fontsize=12)
    axes[1].set_title('Predicted', fontsize=12)
    plt.scatter(Centroids[0,:],Centroids[1,:],s=25,c='Black',label='Centr
    plt.legend()
    plt.show()

    # Calculating the sum of distances of samples to their closest cluste
    wcss=0
    for k in range(K):
        wcss+=np.sum((Output[k+1]-Centroids[:,k])**2)
    WCSS_array.append(wcss)

# Output: The Elbow Method Curve wrt Inertia
K_array=np.arange(1,11,1)
plt.figure(figsize=(10,5))
plt.plot(K_array,WCSS_array)
plt.xlim(0, 10)
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("The Elbow Method Curve to find Optimum k")
plt.show()
```

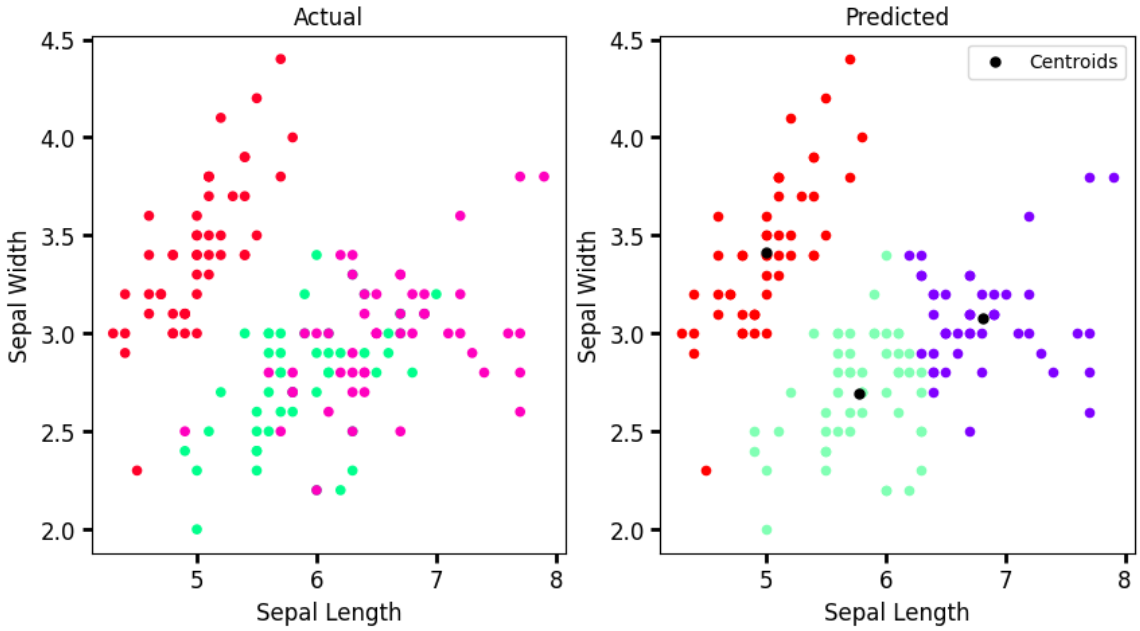
```
For K =  1
  Center  1  =  5.8483221476510066 3.051006711409396
```



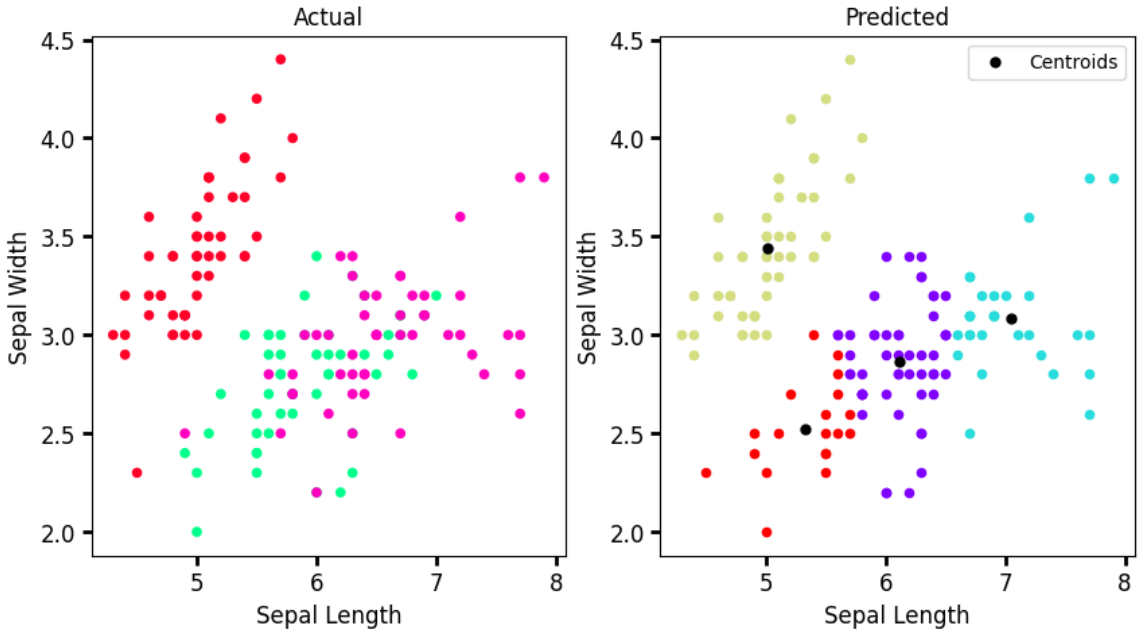

```
For K =  2
  Center  1  =  6.518421052631578 2.948684210526316
  Center  2  =  5.15068493150685 3.1575342465753424
```



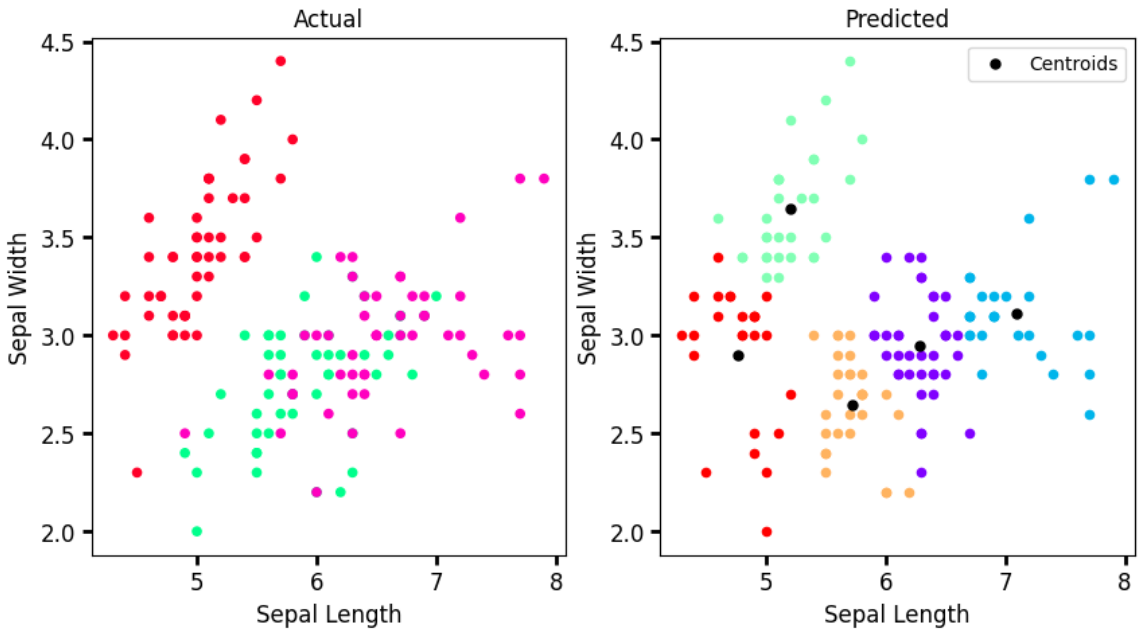

```
For K =  3
  Center  1  =  6.812765957446807 3.074468085106383
  Center  2  =  5.773584905660377 2.692452830188679
  Center  3  =  5.004081632653061 3.4163265306122454
```



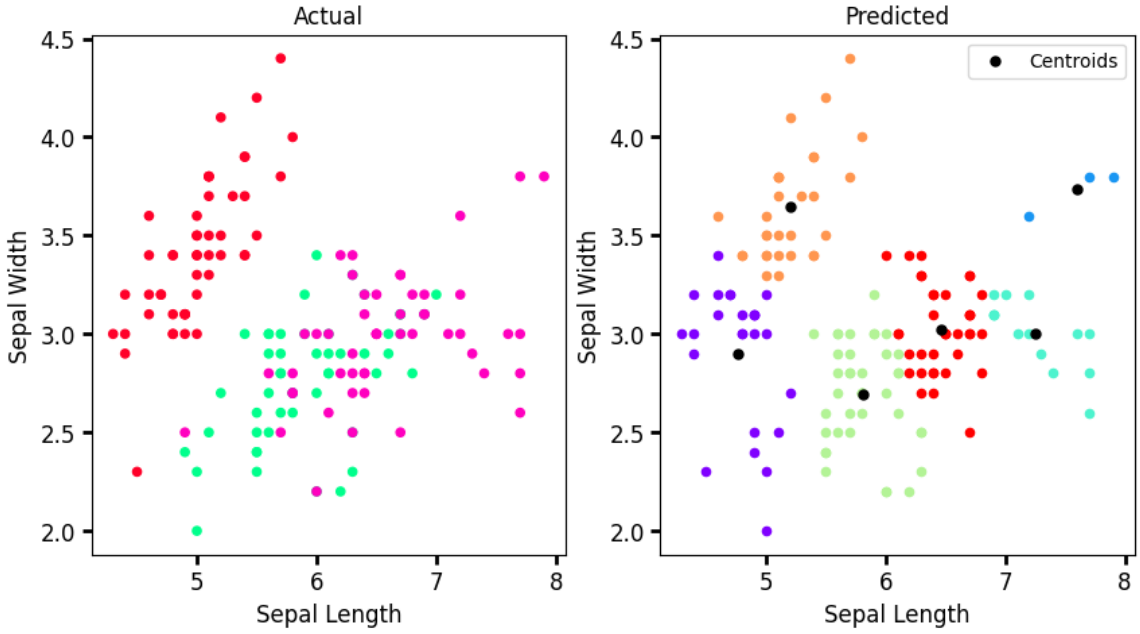

```
For K =  4
  Center  1  =  6.113461538461538 2.8673076923076923
  Center  2  =  7.05 3.0833333333333326
  Center  3  =  5.014583333333333 3.4395833333333337
  Center  4  =  5.3315789473684205 2.5210526315789474
```



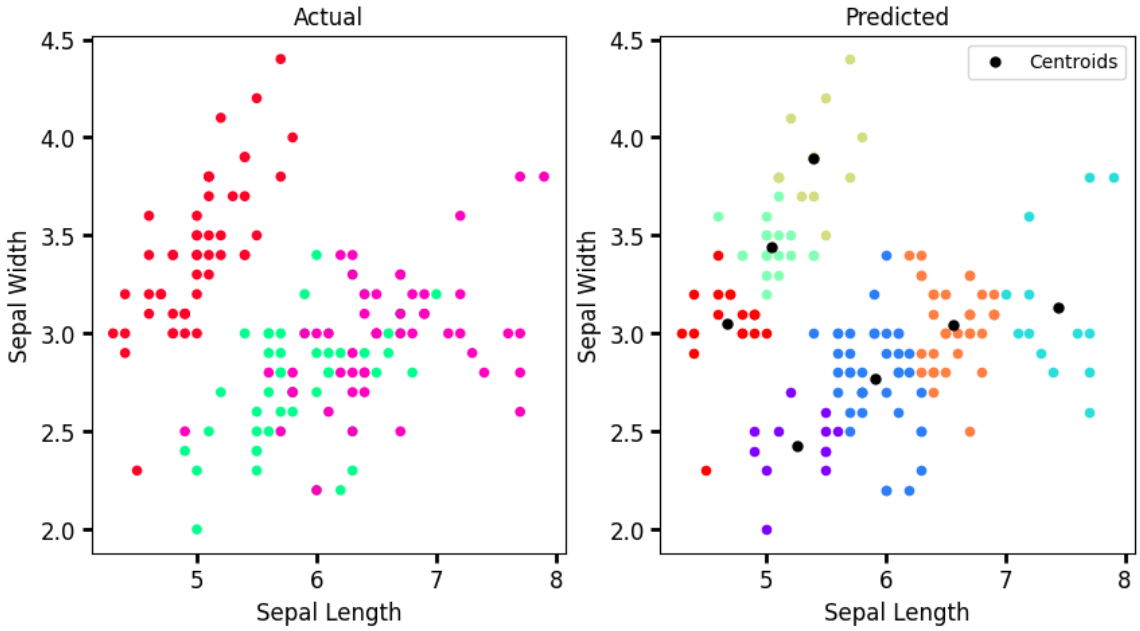

```
For K =  5
  Center  1  =  6.281578947368422 2.944736842105263
  Center  2  =  7.096296296296296 3.1148148148148147
  Center  3  =  5.2 3.6433333333333326
  Center  4  =  5.717241379310345 2.6482758620689655
  Center  5  =  4.772 2.9
```



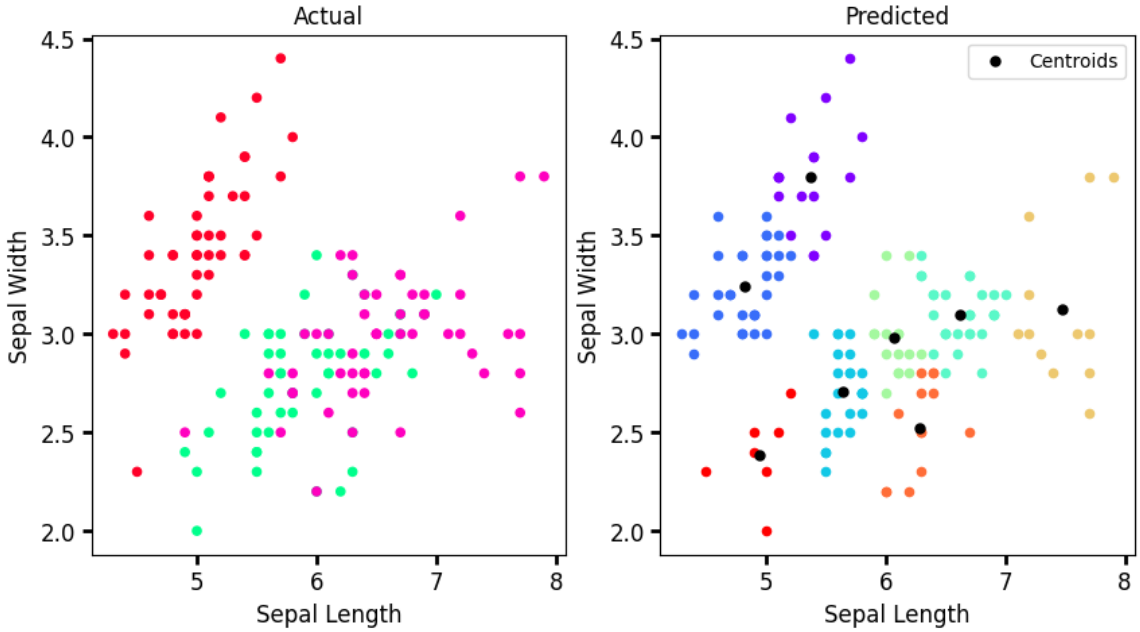

```
For K =  6
  Center  1  =  4.772 2.9
  Center  2  =  7.6000000000000005 3.733333333333333
  Center  3  =  7.250000000000002 3.0
  Center  4  =  5.817500000000001 2.6925
  Center  5  =  5.2 3.6433333333333326
  Center  6  =  6.462162162162161 3.0243243243243243
```



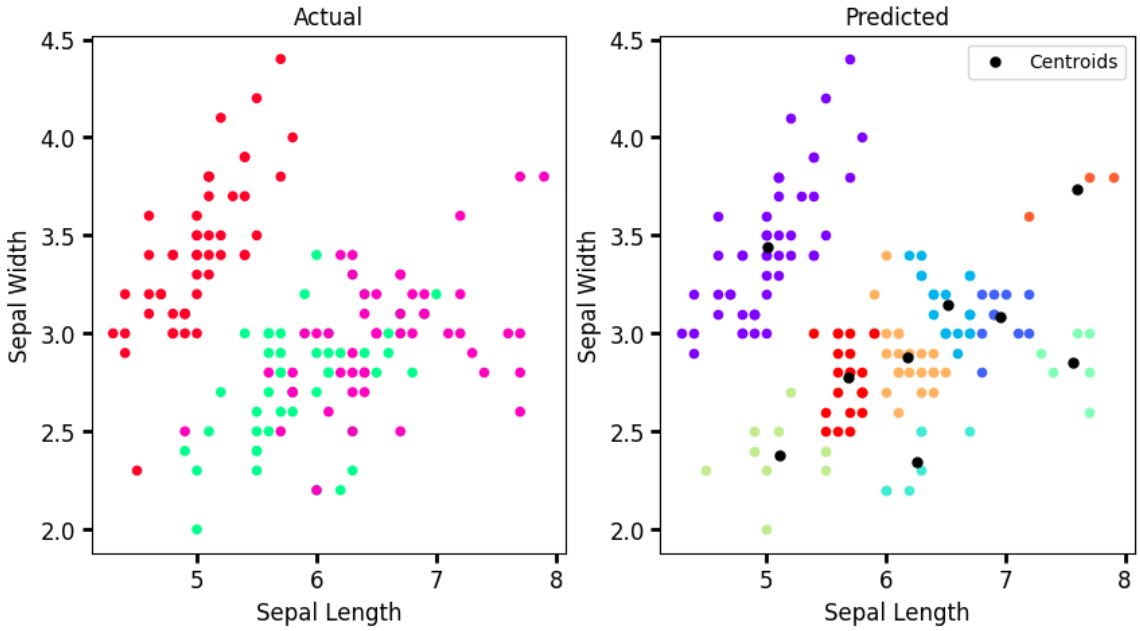

```
For K =  7
  Center  1  =  5.266666666666667 2.4250000000000003
  Center  2  =  5.9125 2.77
  Center  3  =  7.43846153846154 3.1307692307692303
  Center  4  =  5.044444444444444 3.4388888888888887
  Center  5  =  5.4 3.892307692307692
  Center  6  =  6.565714285714285 3.0457142857142854
  Center  7  =  4.677777777777778 3.0500000000000003
```



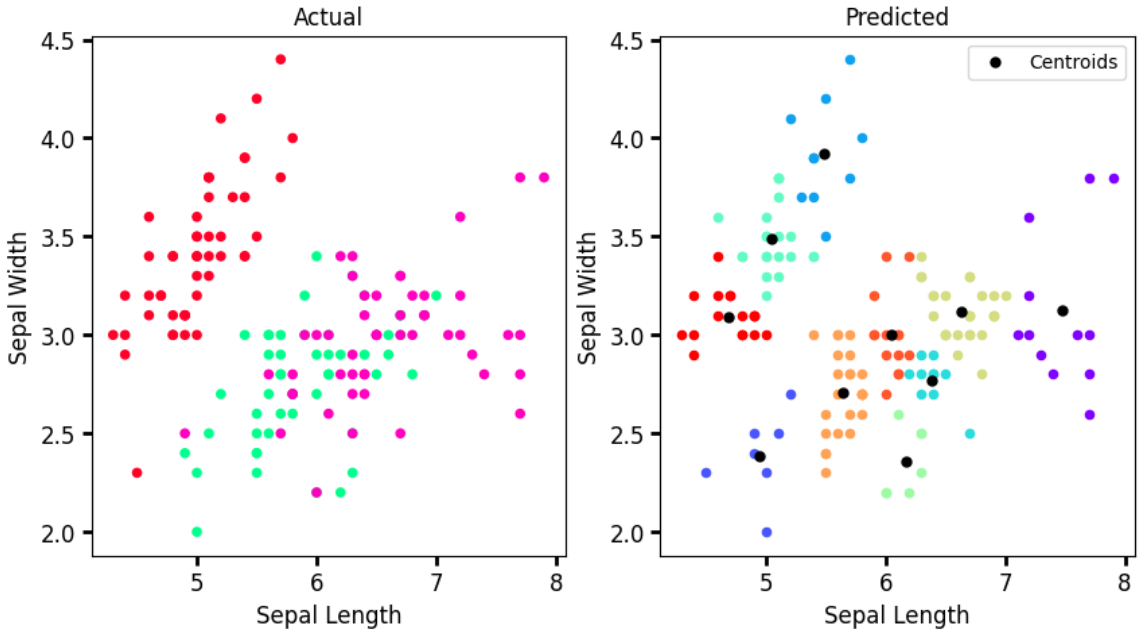

```
For K =  8
  Center  1  =  5.370588235294117 3.8
  Center  2  =  4.8193548387096765 3.2419354838709684
  Center  3  =  5.645833333333333 2.704166666666667
  Center  4  =  6.624137931034483 3.0999999999999996
  Center  5  =  6.06875 2.98125
  Center  6  =  7.475000000000001 3.125
  Center  7  =  6.284615384615384 2.5230769230769234
  Center  8  =  4.942857142857143 2.3857142857142857
```

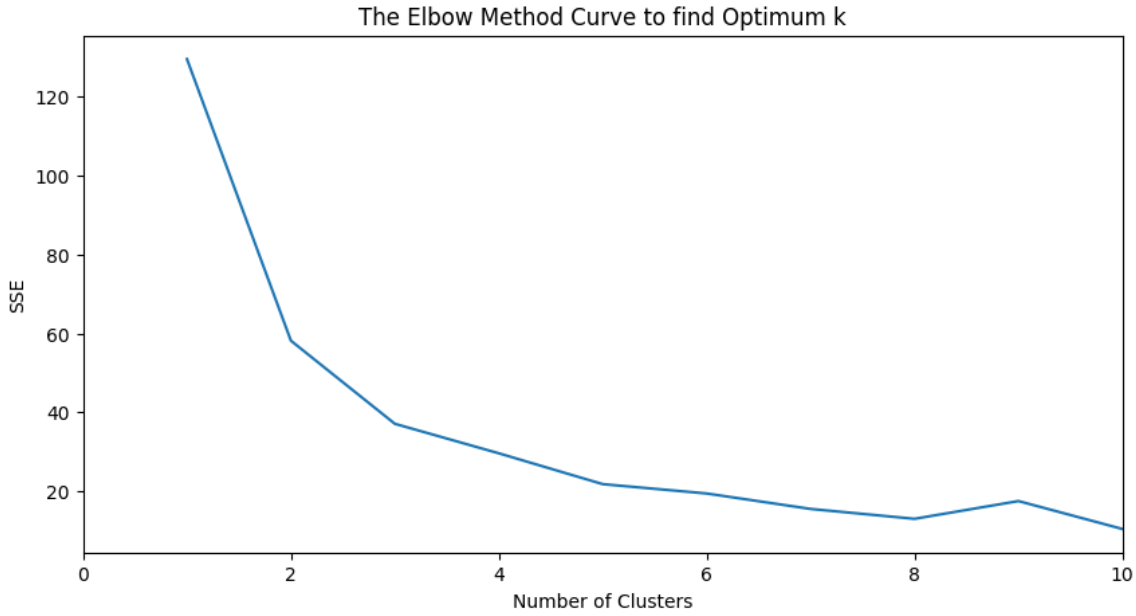



```
For K =  9
  Center  1  =  5.014583333333333 3.4395833333333337
  Center  2  =  6.954545454545454 3.0818181818181825
  Center  3  =  6.5200000000000005 3.145
  Center  4  =  6.257142857142857 2.3428571428571425
  Center  5  =  7.566666666666666 2.85
  Center  6  =  5.11 2.38
  Center  7  =  6.185714285714286 2.8761904761904757
  Center  8  =  7.6000000000000005 3.733333333333333
  Center  9  =  5.68695652173913 2.773913043478261
```




```
For K =  10
  Center  1  =  7.475000000000001 3.125
  Center  2  =  4.942857142857143 2.3857142857142857
  Center  3  =  5.49 3.9200000000000004
  Center  4  =  6.39 2.7700000000000005
  Center  5  =  5.052380952380952 3.4904761904761905
  Center  6  =  6.171428571428571 2.357142857142857
  Center  7  =  6.637037037037037 3.1185185185185187
  Center  8  =  5.645833333333333 2.704166666666667
  Center  9  =  6.042857142857144 2.9999999999999996
  Center  10  =  4.688235294117646 3.0941176470588236
```





Exercise-8

Comparison of Both Methods.

We trained both the models for different values of k with Maximum Number of Iteration allowed as 1000. The Sci-Kit Learn Method efficiently trained the model for all values of k under given parameters, whereas the Manual Method failed to iterate the same for k>=9.

We performed Elbow Method to find the Optimum Value of k, which can be observed from the plotted graphs. Both the methods give Optimum Results for k=3.

Overall, both methods are efficient, but, SciKit-Learn Method can be preferred over Manual Method as it runs faster and gives a result within few number of iterations.

Assignment 10: Spam Filter using Support Vector Machine

Steps:

(i) Load the required libraries

(ii) Import the dataset.

(iii) Seperating the data into Independent and dependent variables.

(iv) Split the dataset into testing and training set

(v) Train the SVM model over training set

(a) Find the Optimal hyperplane as

(A) For each line separating the two classes find the points closest to the line from both classes These points are support vectors.

(B) Calculate the Euclidean Distance between the points ie support vectors.

(c) Find the line with maximum margin value.

(D) This line makes the Optimal hyperplane

(vi) Predict class value for X-test using trained model

(vii) Compute the Accuracy and classification Report & the Accuracy and Classification Report & the confusion matrix.

Exercise 10: A spam filter using SVM:

UCI dataset: Spambase D6: https://archive.ics.uci.edu/ml/datasets/Spambase Build a spam filter using a pre-processed dataset. A spam filter to classify an email to be Ham or Spam, using the content of an email as features. Build a basic spam filter using SVM. You have to use libsvm https://github.com/cjlin1/libsvm/tree/master/python. libsvm accepts data in a libsvm format. Each data row in a libsvm format is given as

< label >< index1 >:< value1 >< index2 >:< value2 >...

Convert dataset D3 into a libsvm format. Follow the readme document given on the libsvm link to see how you can use it to solve your problem. You have to learn a spam classifier on train part of the dataset and evaluate it on test dataset. Also optimize the hyper parameter i.e. value of C. [hint: when choosing the range of hyperparameter its always useful to check a diverse range i.e. C ={1,2,3,4} is not a good range to check for optimal value, you might want to check a broader range going from 0.1 to 100 etc.]. Present your results in form of graphs and tables, listing details. You have to choose a quality criterion according to the given problem i.e. classification

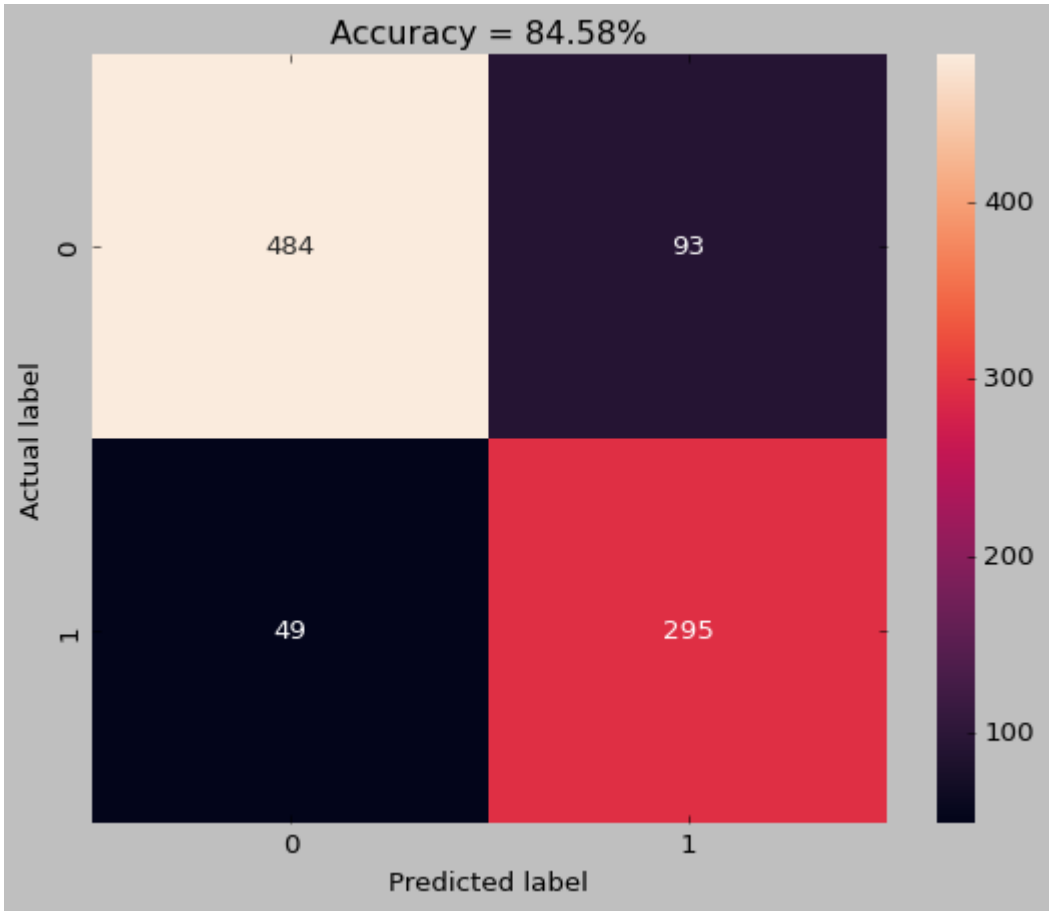
In [ ]:
```
import pandas as pd
from libsvm.svmutil import *
from sklearn.model_selection import train_test_split
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('classic')
import seaborn as sns
from sklearn import metrics
# Input: Dataset
data = pd.read_csv("spambase.data", sep=',', header= None)
# Separating the data into independent and dependent variables
X = data.iloc[:,:-1]
Y= data.iloc[:,-1].values
# Splitting the data into training and testing data
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2)
X_train = X_train.to_numpy()
X_test = X_test.to_numpy()
m = svm_train(Y_train, X_train, '-c 4')
p_label, p_acc, p_val = svm_predict(Y_test, X_test, m)
cm = metrics.confusion_matrix(Y_test, p_label)
```

```
....*..*
optimization finished, #iter = 6348
nu = 0.233709
obj = -2069.534770, rho = -0.147558
nSV = 2159, nBSV = 291
Total nSV = 2159
Accuracy = 84.582% (779/921) (classification)
```

Output: The Confuse Matrix and Classification Report

In [ ]:
```
cm_df = pd.DataFrame(cm)
sns.heatmap(cm_df, annot=True, fmt='g')
```

```
plt.title('Accuracy = {0:.2f}%'.format(p_acc[0]))
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
print("\nThe Classification Report for the Spam Filter\n\n" ,metrics.clas
```




```
The Classification Report for the Spam Filter

              precision    recall  f1-score   support

           0       0.91      0.84      0.87       577
           1       0.76      0.86      0.81       344

    accuracy                           0.85       921
   macro avg       0.83      0.85      0.84       921
weighted avg       0.85      0.85      0.85       921
```