**CO2**

**10.Naive\_Bayes\_Classsification**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

from matplotlib.colors import ListedColormap

from sklearn import datasets

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

def plot\_decision\_boundary(X\_set, y\_set, title):

X1, X2 = np.meshgrid(np.arange(start=X\_set[:, 0].min() - 1, stop=X\_set[:, 0].max() + 1, step=0.01),

np.arange(start=X\_set[:, 1].min() - 1, stop=X\_set[:, 1].max() + 1, step=0.01))

boundary\_input = np.c\_[X1.ravel(), X2.ravel(), np.tile(X\_set[:, 2].mean(), X1.ravel().shape[0]), np.tile(X\_set[:, 3].mean(), X1.ravel().shape[0])]

plt.contourf(X1, X2, classifier.predict(boundary\_input).reshape(X1.shape),

alpha=0.75, cmap=ListedColormap(('red', 'green', 'blue')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c=ListedColormap(('red', 'green', 'blue'))(i), label=iris.target\_names[j])

plt.title(title)

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

plt.ylabel('Sepal Width (cm)')

plt.legend()

plt.show()

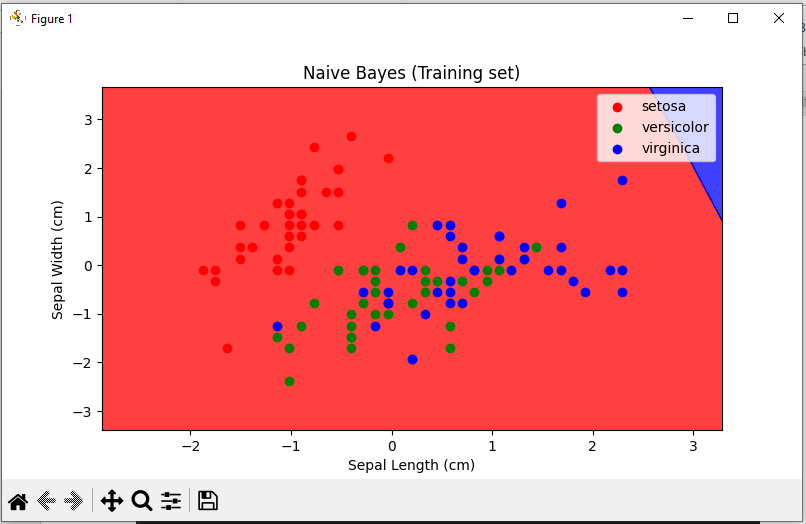
plot\_decision\_boundary(X\_train, y\_train, title='Naive Bayes (Training set)')

plot\_decision\_boundary(X\_test, y\_test, title='Naive Bayes (Test set)')

iris\_df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

print("Iris Dataset:")

print(iris\_df)

**Output** 

**11. KNN-CLASSIFICATION**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

# Generate some sample data (replace this with your dataset)

X = np.random.rand(100, 2) # Features

y = np.random.randint(0, 2, 100) # Target labels (binary classification)

# 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 a KNN classifier with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Fit the classifier to the training data

knn.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = knn.predict(X\_test)

# Calculate and print the confusion matrix

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(conf\_matrix)

# Calculate and print the accuracy score

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

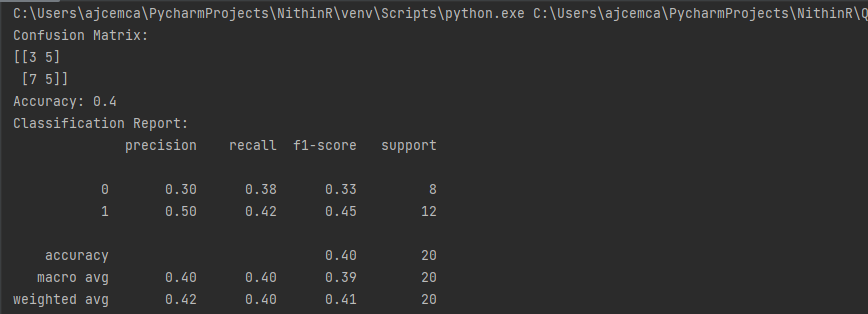
# Print the classification report (including precision, recall, f1-score, and support)

class\_report = classification\_report(y\_test, y\_pred)

print("Classification Report:")

print(class\_report)

**Output**



**12. Linear Regression(without csv)**

import numpy as np

import matplotlib.pyplot as plt

study\_hours = np.array([2, 3, 4, 5, 6])

exam\_scores = np.array([60, 75, 85, 90, 95])

mean\_study\_hours = np.mean(study\_hours)

mean\_exam\_scores = np.mean(exam\_scores)

m = np.sum((study\_hours - mean\_study\_hours) \* (exam\_scores - mean\_exam\_scores)) / np.sum((study\_hours - mean\_study\_hours) \*\* 2)

b = mean\_exam\_scores - m \* mean\_study\_hours

def predict\_exam\_score(hours):

return m \* hours + b

predicted\_score = predict\_exam\_score(4)

print(f"Predicted Exam Score for 4 hours of study: {predicted\_score}")

regression\_line = m \* study\_hours + b

plt.scatter(study\_hours, exam\_scores, color='blue', label='Actual Data')

plt.plot(study\_hours, regression\_line, color='red', label='Regression Line')

plt.scatter(4, predicted\_score, color='green', label='Predicted Point (4 hours)')

plt.xlabel('Study Hours')

plt.ylabel('Exam Score')

plt.title('Simple Linear Regression')

plt.legend()

plt.show()

**Output**

import numpy as np

import matplotlib.pyplot as plt

np.random.seed(42) # for reproducibility

num\_samples = 100

education\_level = np.random.uniform(0, 20, num\_samples) # Simulated education levels

income = 1000 \* education\_level + np.random.normal(0, 5000, num\_samples) # Simulated income

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.hist(education\_level, bins=15, edgecolor='black', alpha=0.7, color='skyblue')

plt.xlabel('Education Level')

plt.ylabel('Frequency')

plt.title('Distribution of Education Level')

# Distribution chart for income

plt.subplot(1, 2, 2)

plt.hist(income, bins=15, edgecolor='black', alpha=0.7, color='orange')

plt.xlabel('Income')

plt.ylabel('Frequency')

plt.title('Distribution of Income')

plt.tight\_layout()

plt.figure(figsize=(8, 6))

plt.scatter(education\_level, income, color='blue', alpha=0.7)

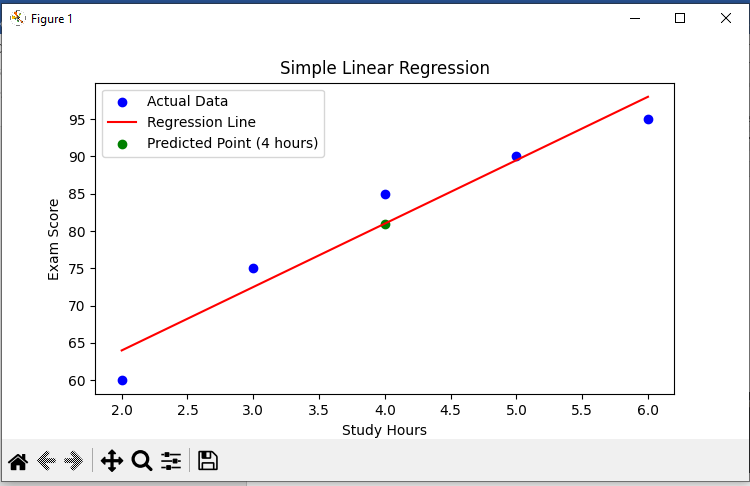
plt.xlabel('Education Level')

plt.ylabel('Income')

plt.title('Income vs. Education Level')

plt.grid(True)

plt.show()



**13. Decision tree classifier(using iris dataset)**from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

from sklearn import tree

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# 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)

# Create a Decision Tree classifier

clf = DecisionTreeClassifier()

clf.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = clf.predict(X\_test)

# Calculate the accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy:.2f}')

# Display the classification report and confusion matrix

print(classification\_report(y\_test, y\_pred))

print(f'Confusion Matrix:\n{confusion\_matrix(y\_test, y\_pred)}')

plt.figure(figsize=(12, 8))

tree.plot\_tree(clf, feature\_names=iris.feature\_names, class\_names=iris.target\_names, filled=True)

plt.show()

**Output**

