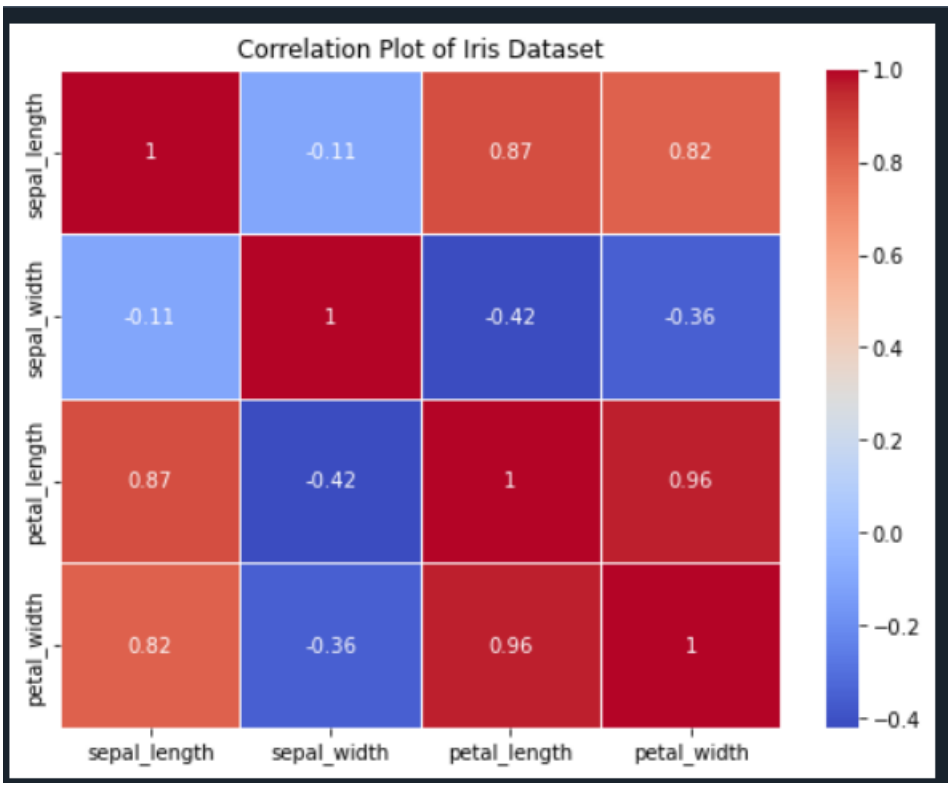
NAME :Ashutosh Rode ROLL NO : 72 DIV : B

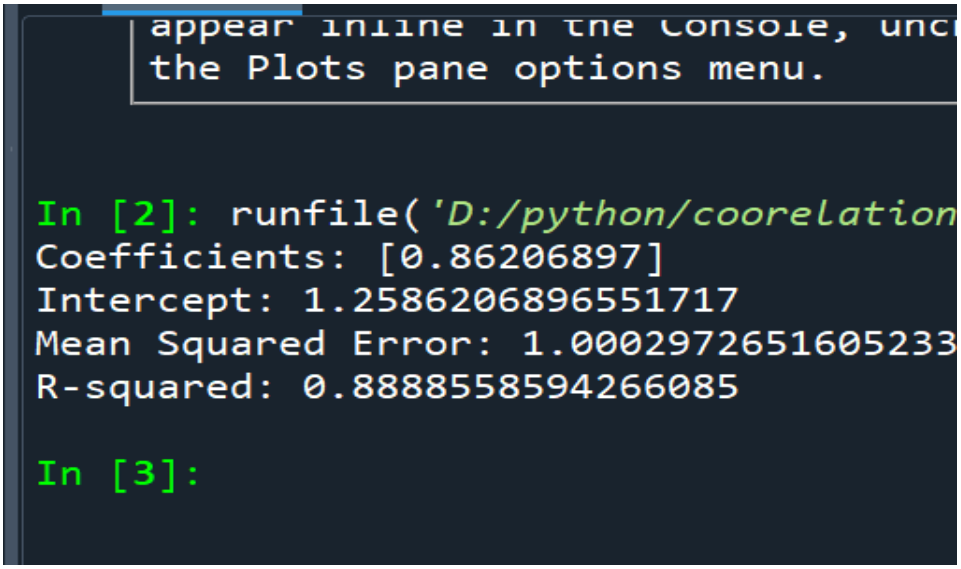
Q1] Write a program to Plot the correlation plot on dataset and visualize giving an overview of relationships among data on iris data.

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load the Iris dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

column_names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"]

iris_data = pd.read_csv(url, names=column_names)

# Calculate the correlation matrix

correlation_matrix = iris_data.corr()

# Create a heatmap to visualize the correlations

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

sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", linewidths=0.5)

plt.title("Correlation Plot of Iris Dataset")

plt.show()
```



Q2] Write a program to implement linear regression algorithm to create and evaluate a model on a given dataset.

```
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

# Sample dataset (replace this with your own dataset)

# X represents the independent variable, and y represents the dependent variable.

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]).reshape(-1, 1)

y = np.array([2, 4, 5, 4, 5, 6, 8, 8, 10, 10])

# 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 linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X_train, y_train)

# Make predictions on the test data

y_pred = model.predict(X_test)

# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)

# Print the model's coefficients and evaluation metrics

print("Coefficients:", model.coef_)

print("Intercept:", model.intercept_)

print("Mean Squared Error:", mse)

print("R-squared:", r2)
```

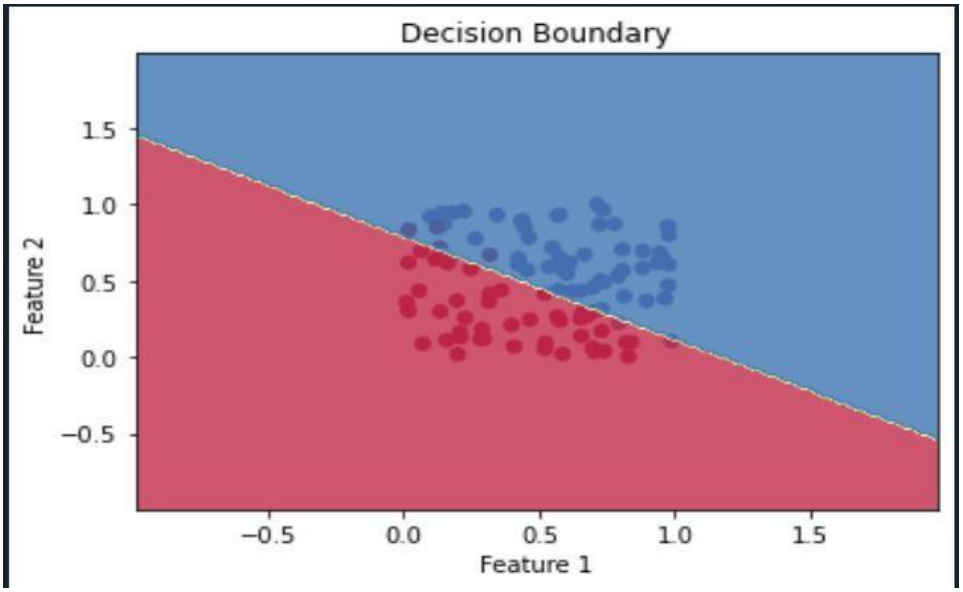


Q3] Write a program to classify the given dataset using logistic regression and evaluate the model.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Generating a synthetic dataset for binary classification
np.random.seed(0)
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] > 1).astype(int)
# 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 logistic regression model
model = LogisticRegression()
# Train the model on the training data
model.fit(X_train, y_train)


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

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
# Visualize the decision boundary (works for 2D data only)
if X.shape[1] == 2:
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Spectral)
    h = .02  # step size in the mesh
    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, h), np.arange(y_min, y_max, h))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral, alpha=0.8)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title("Decision Boundary")
    plt.show()
```

```
Console 1/A

In [5]: runfile('D:/python/coorelation.py', wdir='D:/python')
Accuracy: 0.9
Confusion Matrix:
 [[ 8  2]
 [ 0 10]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.80      0.89        10
           1       0.83      1.00      0.91        10

    accuracy                           0.90        20
   macro avg       0.92      0.90      0.90        20
weighted avg       0.92      0.90      0.90        20
```



Q4] Write a program to implement support vector machine algorithm

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Load a sample dataset (you can replace this with your own data)
# We'll use the built-in Iris dataset for this example.
iris = datasets.load_iris()
X = iris.data
```

```
y = iris.target
# For binary classification, let's consider only two classes (0 and 1).
X = X[y != 2]
y = y[y != 2]
# 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 an SVM classifier
model = SVC(kernel='linear')
# Train the SVM model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
In [6]: runfile('D:/python/coorelation.py', wdir='D:/python')
Accuracy: 1.0
Confusion Matrix:
 [[12  0]
 [ 0  8]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00         8

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

Q5] Write a program to implement Decision Tree model on the given dataset

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Load a sample dataset (you can replace this with your own data)
# We'll use the built-in Iris dataset for this example.
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a Decision Tree classifier
model = DecisionTreeClassifier()
# Train the Decision Tree model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
# Print the evaluation metrics
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

```
In [7]: runfile('D:/python/coorelation.py', wdir='D:/python')
Accuracy: 1.0
Confusion Matrix:
 [[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
Classification Report:
              precision    recall  f1-score   support

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

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

Q6] Write a program to implement Bayesian classification on given dataset.

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
# Load a sample dataset (you can replace this with your own data)
# We'll use the built-in Iris dataset for this example.
iris = datasets.load_iris()
X = iris.data
y = iris.target
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create a Naive Bayes classifier (Gaussian Naive Bayes)
model = GaussianNB()
# Train the Naive Bayes model on the training data
model.fit(X_train, y_train)
# Make predictions on the test data
y_pred = model.predict(X_test)
```

```
# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

class_report = classification_report(y_test, y_pred)

# Print the evaluation metrics

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("Classification Report:\n", class_report)
```