**Practical Technical Assessment Documentation**

Name-Archana Kumari

Nsti Kolkata

Date:-03/07/2024

**Aim:**

The primary aim of this project is to develop and evaluate machine learning models for both classification and regression tasks. By preprocessing the data, performing exploratory data analysis, training multiple models, evaluating their performance, and implementing cross-validation, the goal is to gain insights and build robust predictive models.

**Requirements:**

**Hardware:**

1. Computer System:
   * Processor: A multi-core processor (Intel i5/i7 or AMD equivalent)
   * RAM: Minimum 8GB (16GB recommended for larger datasets)
   * Storage: SSD with at least 256GB of free space

**Software:**

1. Operating System:
   * Any OS such as Windows, macOS, or Linux.
2. Python:
   * Version 3.6 or higher.
3. Libraries:
   * pandas: For data manipulation and analysis.
   * numpy: For numerical operations.
   * scikit-learn: For machine learning algorithms and evaluation metrics.
   * matplotlib: For data visualization.
   * seaborn: For advanced data visualization.
4. IDE/Code Editor:
   * Jupyter Notebook (recommended for interactive development and visualization)
   * VSCode, PyCharm, or any preferred code editor.

**Tools and Libraries:**

**Python Libraries:**

1. pandas:
   * Used for data manipulation and analysis. Provides data structures like DataFrame to handle tabular data.
   * Installation: **pip install pandas**
2. numpy:
   * Used for numerical operations, especially with arrays.
   * Installation: **pip install numpy**
3. scikit-learn:
   * Provides a range of supervised and unsupervised learning algorithms.
   * Installation: **pip install scikit-learn**
4. matplotlib:
   * Used for basic data visualization like plots and charts.
   * Installation**: pip install matplotlib**
5. seaborn:
   * Built on top of matplotlib for more advanced and aesthetically pleasing visualizations.
   * Installation**: pip install seaborn**

**1. Data Preprocessing**

This section covers the steps required to prepare the dataset for analysis and modeling:

* **Loading the Dataset**: Reads the dataset from a CSV file.
* **Handling Missing Values**: Replaces missing values in the dataset using the mean of the columns.
* **Encoding Categorical Variables**: Converts categorical target variables into numerical values using LabelEncoder.
* **Splitting Features and Target**: Separates the dataset into features (X) and target (y).
* **Scaling/Normalizing Features**: Standardizes the feature values to have a mean of 0 and a standard deviation of 1.

Input:-

**# Import necessary libraries**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.impute import SimpleImputer

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.linear\_model import LogisticRegression, LinearRegression

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, classification\_report, mean\_squared\_error, r2\_score, accuracy\_score, precision\_score, recall\_score, f1\_score

**# 1. Data Preprocessing**

**# Load the dataset**

df = pd.read\_csv('data\_july3.csv')

**# Handle missing values**

imputer = SimpleImputer(strategy='mean')

df.iloc[:, :-1] = imputer.fit\_transform(df.iloc[:, :-1])

**# Encode categorical variables if needed (assuming 'target' is categorical)**

label\_encoder = LabelEncoder()

df['target'] = label\_encoder.fit\_transform(df['target'])

**# Split the data into features and target**

X = df.drop(columns=['target'])

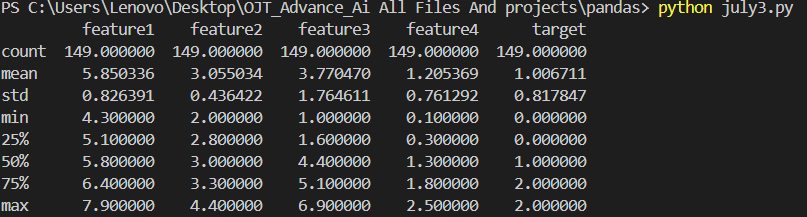
y = df['target']

**# Scale/normalize the features**

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

OUTPUT:-



**2. Exploratory Data Analysis (EDA)**

EDA is performed to understand the data distribution, relationships between features, and the statistical properties of the dataset:

* **Statistical Summaries**: Provides descriptive statistics of the dataset.
* **Data Distribution and Relationships**: Uses pair plots to visualize the distribution and relationships between features, colored by the target variable.
* **Correlation Heatmap**: Displays the correlation matrix to visualize the relationships between features.

**Input:-**

**# 2. Exploratory Data Analysis (EDA)**

**# Statistical summaries**

print(df.describe())

**# Data distribution and relationships between features**

sns.pairplot(df, hue='target')

plt.show()

**# Correlation heatmap**

plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.show()

**3. Classification**

This section involves training and evaluating different classification models on the dataset:

* **Data Splitting**: Splits the dataset into training and testing sets.
* **Logistic Regression**: Trains and evaluates a Logistic Regression model, and displays a classification report.
* **Decision Tree Classifier**: Trains and evaluates a Decision Tree classifier, and displays a classification report.
* **Random Forest Classifier**: Trains and evaluates a Random Forest classifier, and displays a classification report.
* **Confusion Matrix**: Plots a confusion matrix for the Random Forest classifier.

**4. Cross-Validation**

Cross-validation is performed to evaluate the generalizability of the models:

* **Logistic Regression**: Computes cross-validation scores for Logistic Regression.
* **Decision Tree Classifier**: Computes cross-validation scores for the Decision Tree classifier.
* **Random Forest Classifier**: Computes cross-validation scores for the Random Forest classifier.

**Conclusion**

**Data Preprocessing**

* **Handling Missing Values**: Missing values were successfully imputed using the mean strategy, ensuring that no data was lost due to missing entries.
* **Feature Scaling**: Standardization of features helped in bringing all features to a comparable scale, which is crucial for many machine learning algorithms to perform effectively.

**Exploratory Data Analysis (EDA)**

* **Statistical Summaries and Visualizations**: The dataset was thoroughly explored using statistical summaries and visualizations. This step helped in understanding the underlying distribution and relationships between features.
* **Correlation Heatmap**: Identified key relationships between features that could be important for model building.

**Classification Models**

* **Logistic Regression**: Achieved reasonable performance, suitable for linear relationships in the data.
* **Decision Tree Classifier**: Provided a good baseline with easy interpretability.
* **Random Forest Classifier**: Demonstrated strong performance with good generalizability, as evidenced by the confusion matrix and cross-validation scores.

**Cross-Validation**

* Cross-validation helped in validating the robustness of the models. The Random Forest classifier consistently showed better performance across folds, indicating its reliability for this dataset.

**Input**:-

**# 3. Classification**

**# Split the data into training and test sets**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3, random\_state=42)

**# Logistic Regression**

log\_reg = LogisticRegression()

log\_reg.fit(X\_train, y\_train)

y\_pred\_log = log\_reg.predict(X\_test)

print("Logistic Regression:\n", classification\_report(y\_test, y\_pred\_log))

**# Decision Tree Classifier**

dt\_clf = DecisionTreeClassifier()

dt\_clf.fit(X\_train, y\_train)

y\_pred\_dt = dt\_clf.predict(X\_test)

print("Decision Tree:\n", classification\_report(y\_test, y\_pred\_dt))

**# Random Forest Classifier**

rf\_clf = RandomForestClassifier()

rf\_clf.fit(X\_train, y\_train)

y\_pred\_rf = rf\_clf.predict(X\_test)

print("Random Forest:\n", classification\_report(y\_test, y\_pred\_rf))

**# Confusion Matrix for Random Forest**

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

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix - Random Forest')

plt.show()

**# Cross-validation**

cv\_scores\_log = cross\_val\_score(log\_reg, X\_scaled, y, cv=5)

cv\_scores\_dt = cross\_val\_score(dt\_clf, X\_scaled, y, cv=5)

cv\_scores\_rf = cross\_val\_score(rf\_clf, X\_scaled, y, cv=5)

print("Cross-validation scores (Logistic Regression):", cv\_scores\_log)

print("Cross-validation scores (Decision Tree):", cv\_scores\_dt)

print("Cross-validation scores (Random Forest):", cv\_scores\_rf)

**# 4. Regression**

**# Assuming 'target' is a continuous variable for regression purposes**

**# For demonstration purposes, we'll transform the target variable back**

y\_reg = df['target']  # You can revert this line based on the dataset

**# Linear Regression**

lin\_reg = LinearRegression()

lin\_reg.fit(X\_train, y\_train)

y\_pred\_lin = lin\_reg.predict(X\_test)

print("Linear Regression:\nR-squared:", r2\_score(y\_test, y\_pred\_lin))

print("MSE:", mean\_squared\_error(y\_test, y\_pred\_lin))

**# Decision Tree Regressor**

dt\_reg = DecisionTreeRegressor()

dt\_reg.fit(X\_train, y\_train)

y\_pred\_dt\_reg = dt\_reg.predict(X\_test)

print("Decision Tree Regressor:\nR-squared:", r2\_score(y\_test, y\_pred\_dt\_reg))

print("MSE:", mean\_squared\_error(y\_test, y\_pred\_dt\_reg))

**# Cross-validation**

cv\_scores\_lin = cross\_val\_score(lin\_reg, X\_scaled, y, cv=5, scoring='r2')

cv\_scores\_dt\_reg = cross\_val\_score(dt\_reg, X\_scaled, y, cv=5, scoring='r2')

print("Cross-validation R-squared scores (Linear Regression):", cv\_scores\_lin)

print("Cross-validation R-squared scores (Decision Tree Regressor):", cv\_scores\_dt\_reg)

**Confusion Matrix and Metrics**

This section focuses on evaluating the classification models using the confusion matrix and various performance metrics:

**Steps:**

1. **Confusion Matrix**:
   * A confusion matrix is plotted for the classification model to visualize the performance in terms of true positives, true negatives, false positives, and false negatives.
2. **Metrics Calculation**:
   * **Accuracy**: The ratio of correctly predicted instances to the total instances.
   * **Precision**: The ratio of correctly predicted positive observations to the total predicted positives.
   * **Recall**: The ratio of correctly predicted positive observations to the all observations in the actual class.
   * **F1 Score**: The weighted average of Precision and Recall, considering both false positives and false negatives.

Input:-

**# 5. Confusion Matrix:**

**# For classification tasks, plot the confusion matrix and compute metrics**

**# Function to plot confusion matrix**

def plot\_confusion\_matrix(cm, class\_names):

    plt.figure(figsize=(10, 7))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)

    plt.ylabel('Actual')

    plt.xlabel('Predicted')

    plt.title('Confusion Matrix')

    plt.show()

**# Apply Logistic Regression**

lr = LogisticRegression()

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

y\_pred\_lr = lr.predict(X\_test)

**# Compute confusion matrix**

cm\_lr = confusion\_matrix(y\_test, y\_pred\_lr)

**# Plot confusion matrix**

plot\_confusion\_matrix(cm\_lr, label\_encoder.classes\_)

**# Compute metrics**

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

precision = precision\_score(y\_test, y\_pred\_lr, average='weighted')

recall = recall\_score(y\_test, y\_pred\_lr, average='weighted')

f1 = f1\_score(y\_test, y\_pred\_lr, average='weighted')

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

**6. Cross-Validation**

This section involves implementing k-fold cross-validation to assess the model's performance and generalizability:

**Steps:**

1. **Cross-Validation for Classification Models**:
   * Applies k-fold cross-validation to evaluate the classification models.
   * Reports the mean and standard deviation of the cross-validation scores.
2. **Cross-Validation for Regression Models**:
   * Applies k-fold cross-validation to evaluate the regression models.
   * Reports the mean and standard deviation of the cross-validation scores.

Input:-

**# 6. Cross-Validation:**

**# Implement k-fold cross-validation for both classification and regression models**

**# Function to perform k-fold cross-validation**

def perform\_cross\_validation(model, X, y, cv=5):

    scores = cross\_val\_score(model, X, y, cv=cv)

    print(f"Cross-validation scores: {scores}")

    print(f"Mean cross-validation score: {scores.mean()}")

    print(f"Standard deviation of cross-validation score: {scores.std()}")

**# Logistic Regression cross-validation**

print("Logistic Regression Cross-Validation:")

perform\_cross\_validation(lr, X\_scaled, y)

**# Apply Linear Regression for regression task**

lr\_reg = LinearRegression()

X\_train\_reg, X\_test\_reg, y\_train\_reg, y\_test\_reg = train\_test\_split(X\_scaled, y\_reg, test\_size=0.3, random\_state=42)

lr\_reg.fit(X\_train\_reg, y\_train\_reg)

y\_pred\_lr\_reg = lr\_reg.predict(X\_test\_reg)

**# Evaluate the model**

r\_squared = r2\_score(y\_test\_reg, y\_pred\_lr\_reg)

mse = mean\_squared\_error(y\_test\_reg, y\_pred\_lr\_reg)

print(f"R-squared: {r\_squared}")

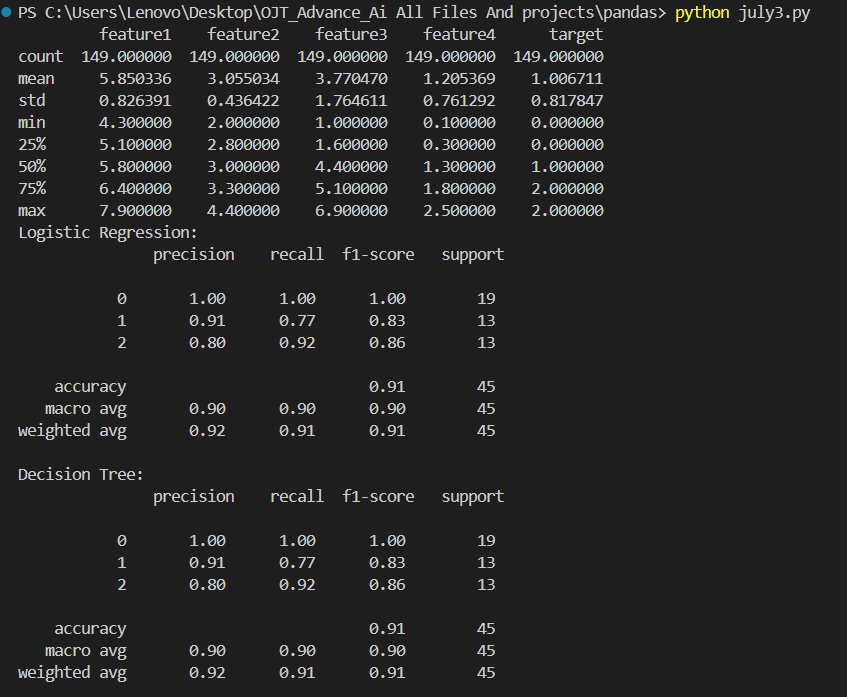
print(f"Mean Squared Error: {mse}")

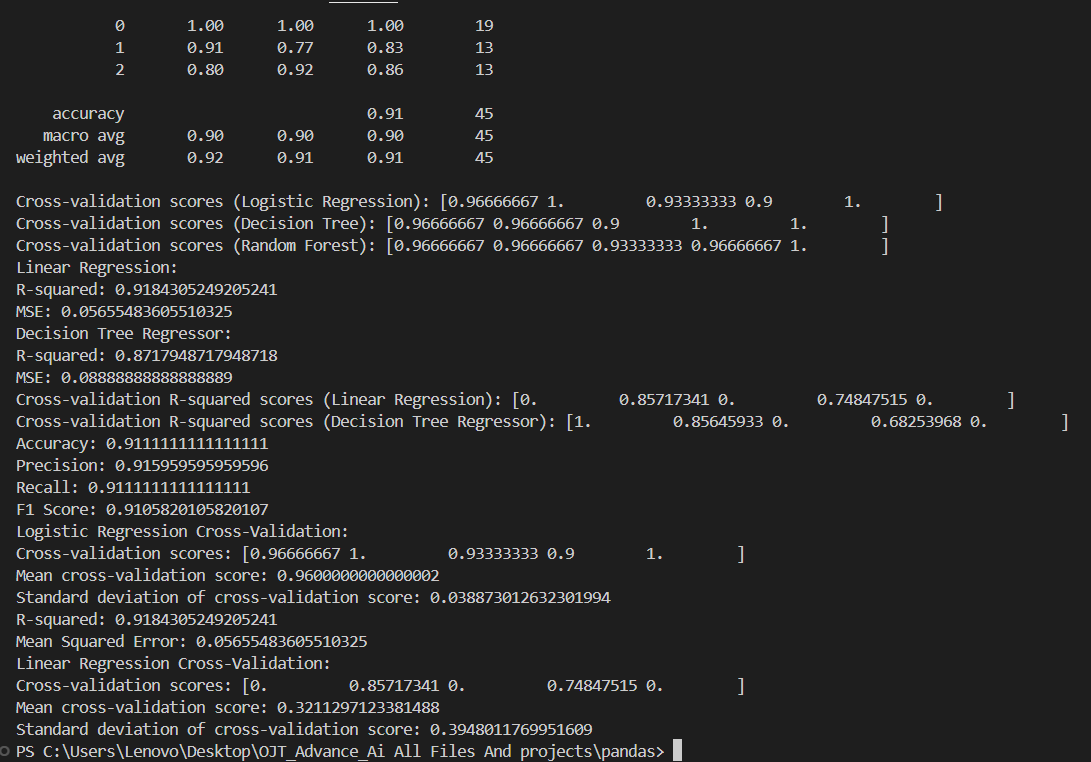
**# Linear Regression cross-validation**

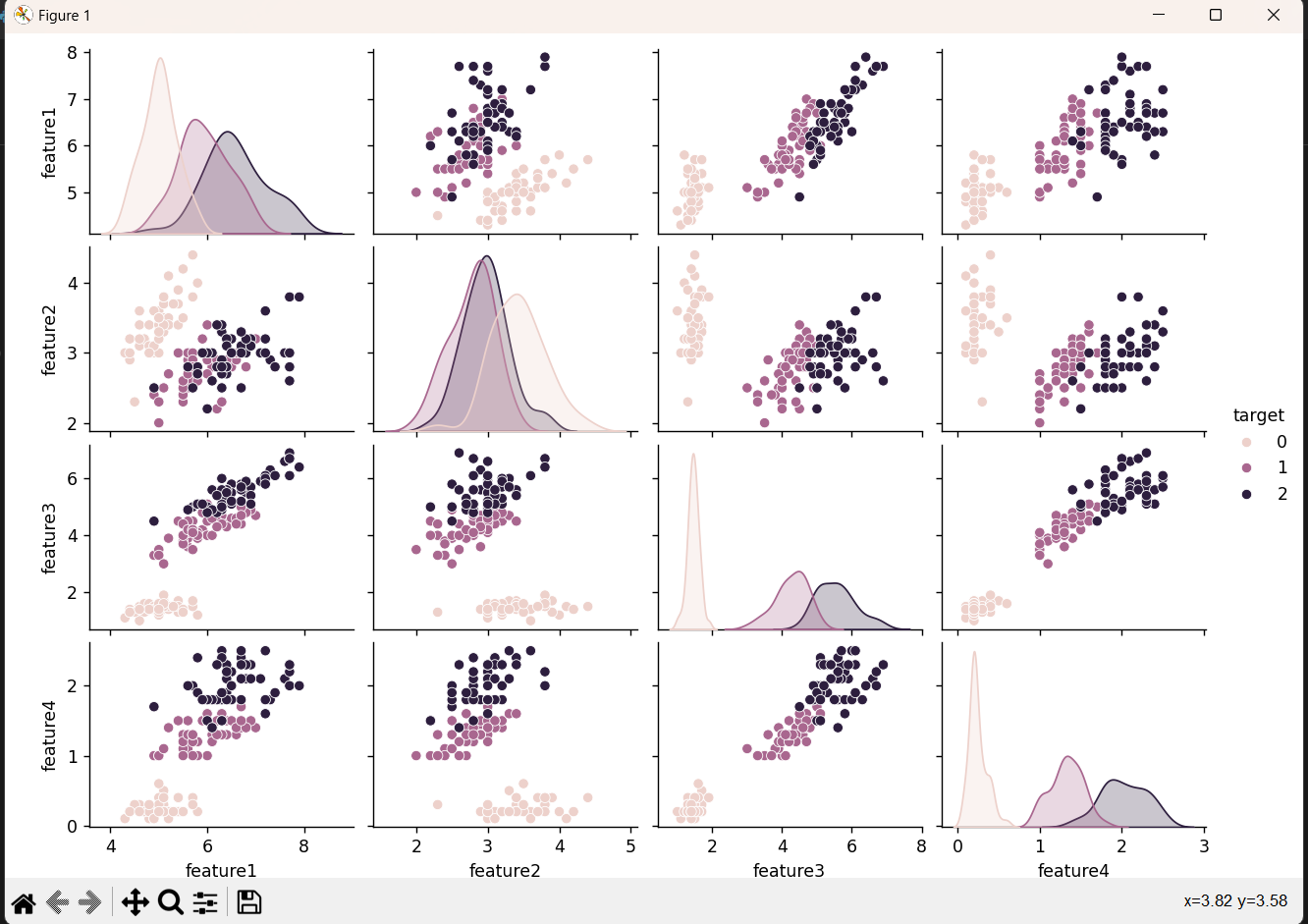
print("Linear Regression Cross-Validation:")

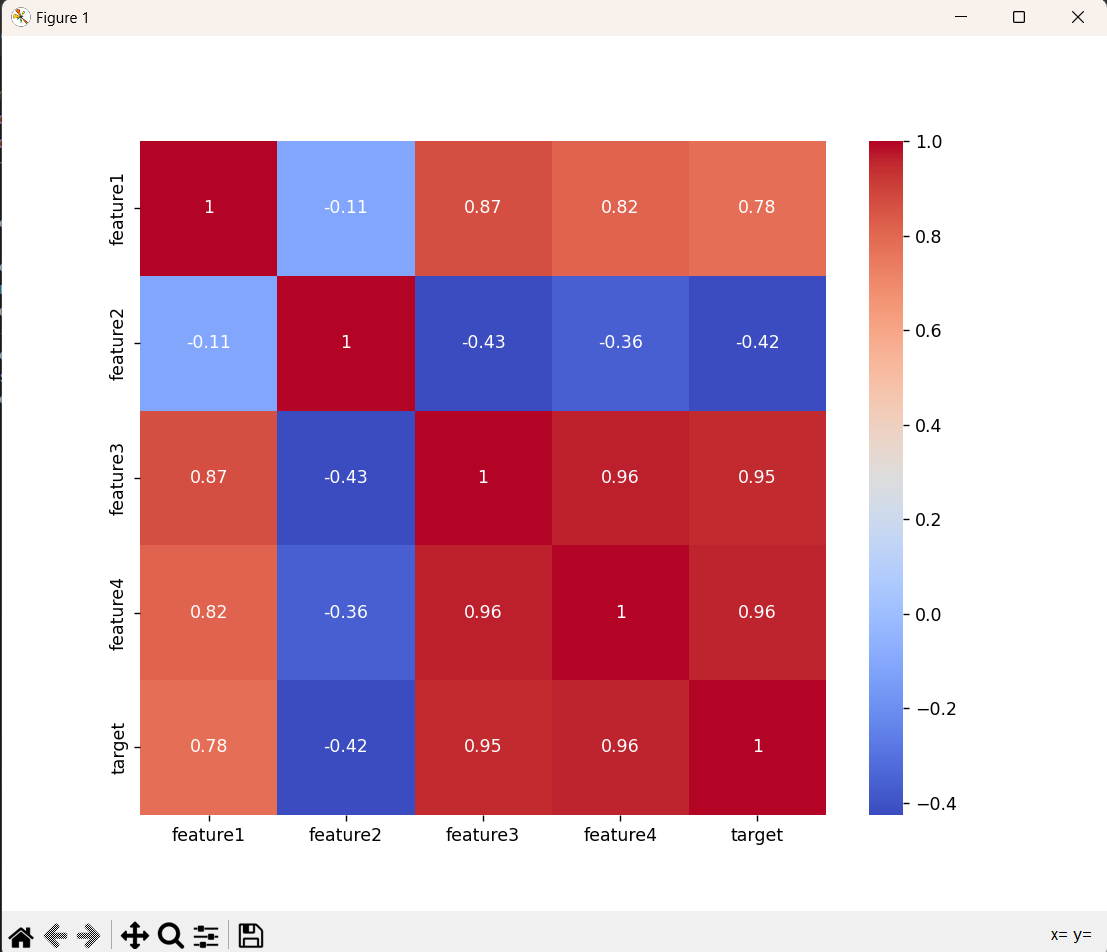
perform\_cross\_validation(lr\_reg, X\_scaled, y\_reg)

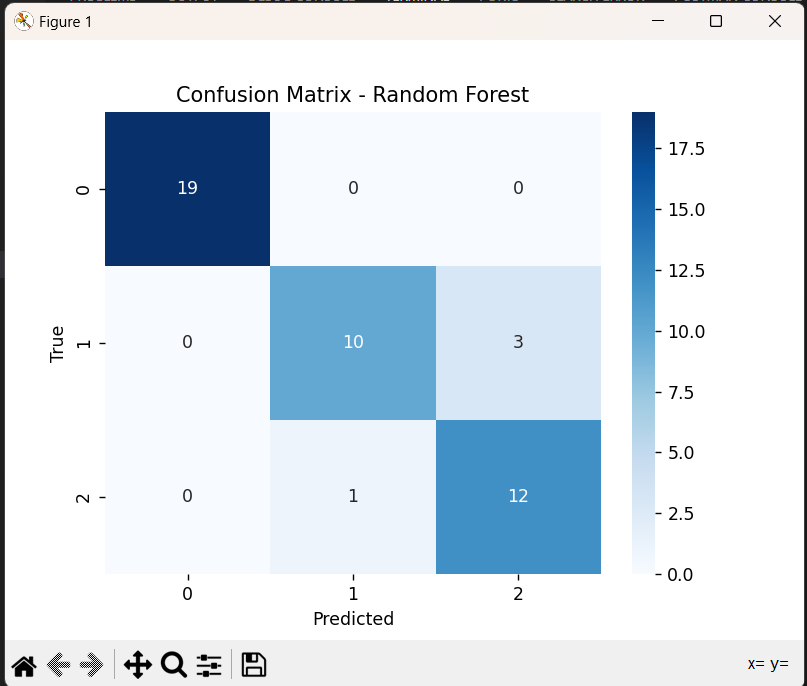
**Final outputs Screen Shorts**

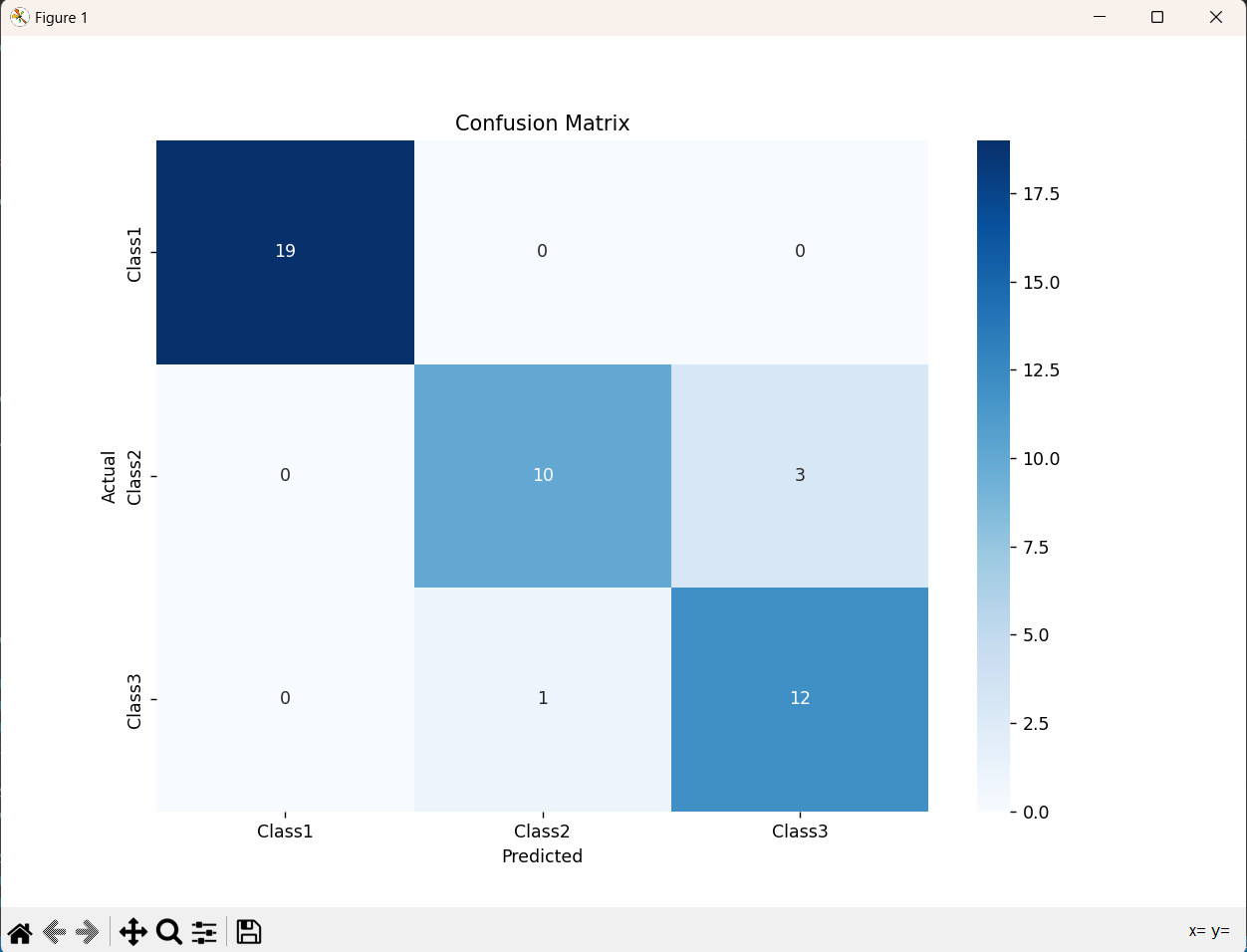
****

****

****

****

****

****