INDEX

Date:	**21/10/2024**
Program no:01	**Exploratory Data Analysis**

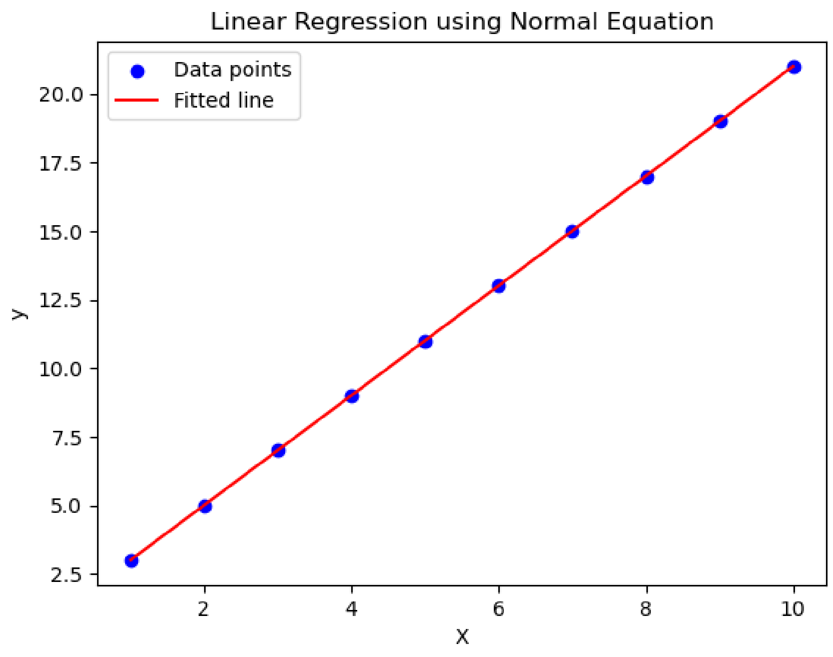
Date:	**4/11/2024**
Program no:02	**Linear Regression**

Source Code:

```
import numpy as np

import matplotlib.pyplot as plt

# Create a simple dataset (X and y)

# For example: y = 2x + 1 with some random noise

X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])  # Feature (independent variable)

y = np.array([3, 5, 7, 9, 11, 13, 15, 17, 19, 21])  # Target (dependent variable)

# Step 1: Calculate necessary sums

n = len(X)

sum_x = np.sum(X)

sum_y = np.sum(y)

sum_x2 = np.sum(X**2)

sum_xy = np.sum(X * y)

# Step 2: Calculate coefficients a (intercept) and b (slope) using the formulas

b = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x**2)

a = (sum_y * sum_x2 - sum_x * sum_xy) / (n * sum_x2 - sum_x**2)

# Step 3: Print the results

print(f"Intercept (a): {a}")

print(f"Slope (b): {b}")

# Step 4: Make predictions

y_pred = a + b * X

# Step 5: Visualize the data and the regression line

plt.scatter(X, y, color='blue', label='Data points')  # Plot original data

plt.plot(X, y_pred, color='red', label='Fitted line')  # Plot regression line

plt.xlabel('X')

plt.ylabel('y')

plt.title('Linear Regression using Normal Equation')

plt.legend()

plt.show()
```

<u>OUTPUT:</u>

```
Intercept (a): 1.0
Slope (b): 2.0
```

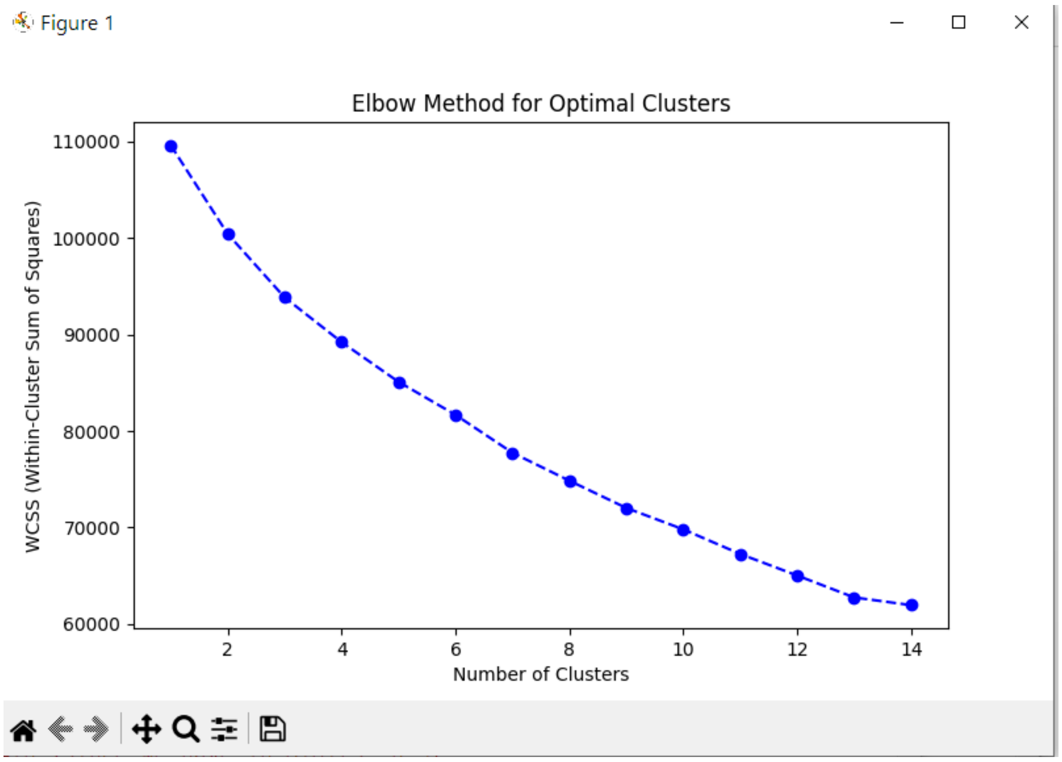


Date:	**18/11/2024**
Program No:03	K-Nearest Neighbours

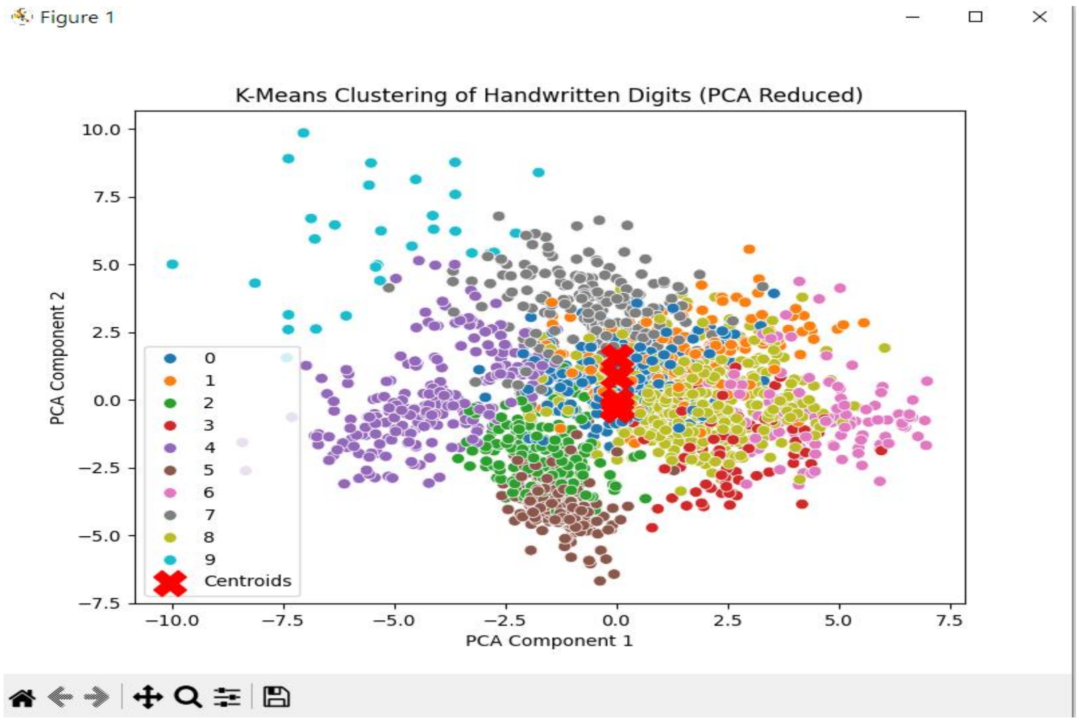
Source Code:

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.datasets import load_digits

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

# Load the inbuilt Digits dataset

digits = load_digits()

X = digits.data  # Features (64 pixel values for each image)

y = digits.target  # Actual digit labels (0-9)

# Scale the data for better clustering

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

# Find the optimal number of clusters using the Elbow Method

wcss = []

for i in range(1, 15):  # Checking clusters from 1 to 15

    kmeans = KMeans(n_clusters=i, random_state=42, n_init=10)

    kmeans.fit(X_scaled)

    wcss.append(kmeans.inertia_)

# Plot Elbow Method

plt.figure(figsize=(8, 5))

plt.plot(range(1, 15), wcss, marker='o', linestyle='--', color='b')
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS (Within-Cluster Sum of Squares)")
plt.title("Elbow Method for Optimal Clusters")
plt.show()

# Choosing k=10 (since we have 10 digits: 0-9)
optimal_clusters = 10
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)

# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Scatter plot of clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette='tab10', s=50, legend='full')  # Use 'tab10' palette for distinct colors
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, color='red', marker='X', label='Centroids')
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("K-Means Clustering of Handwritten Digits (PCA Reduced)")
plt.legend()
plt.show()
```

OUTPUT:



OUTPUT:



Date:	25/11/2024
Program no:04	KN-Means

Source Code:

```
from sklearn.datasets import load_iris

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt


# Load the Iris dataset

iris = load_iris()

X = iris.data  # The features (sepal length, sepal width, petal length, petal width)


# Apply KMeans clustering (3 clusters for the Iris dataset)

kmeans = KMeans(n_clusters=3, random_state=42)

kmeans.fit(X)


# Get the cluster centers and labels

centroids = kmeans.cluster_centers_

labels = kmeans.labels_


# Visualize the clusters using the first two features (sepal length and sepal width)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')


# Plot the centroids

plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label='Centroids')

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.title('KMeans Clustering on Iris Dataset')

plt.legend()

plt.show()
```

OUTPUT:

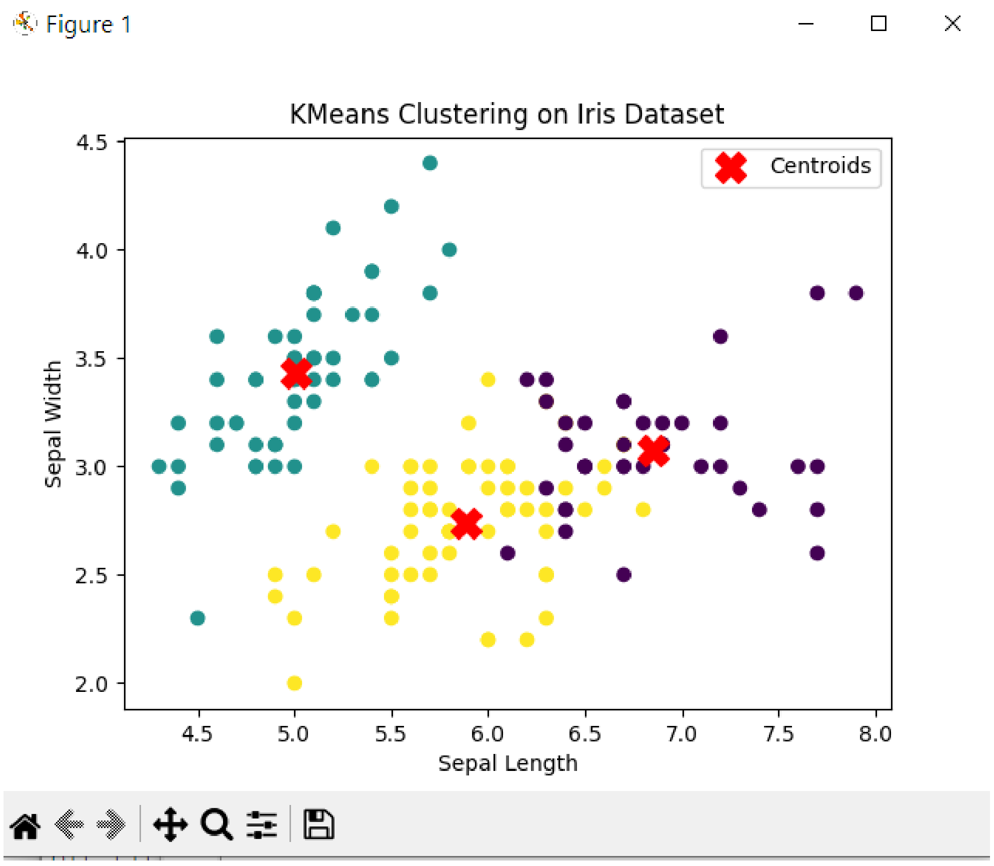


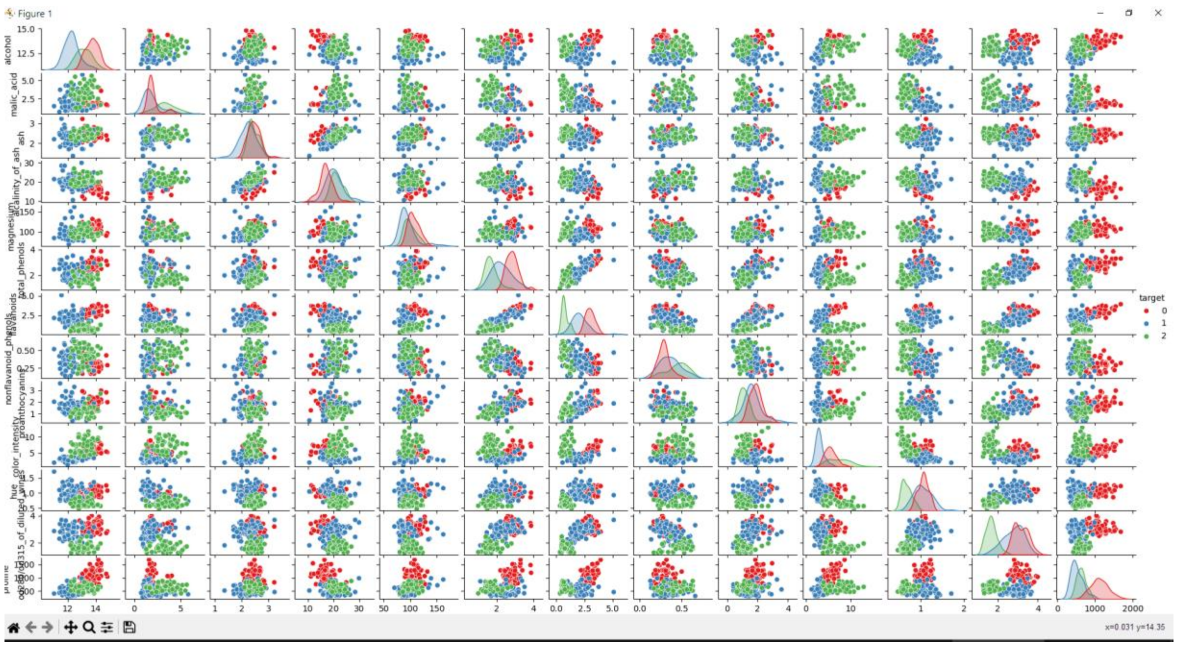
Figure 1
KMeans Clustering on Iris Dataset
Centroids
Sepal Width
Sepal Length
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

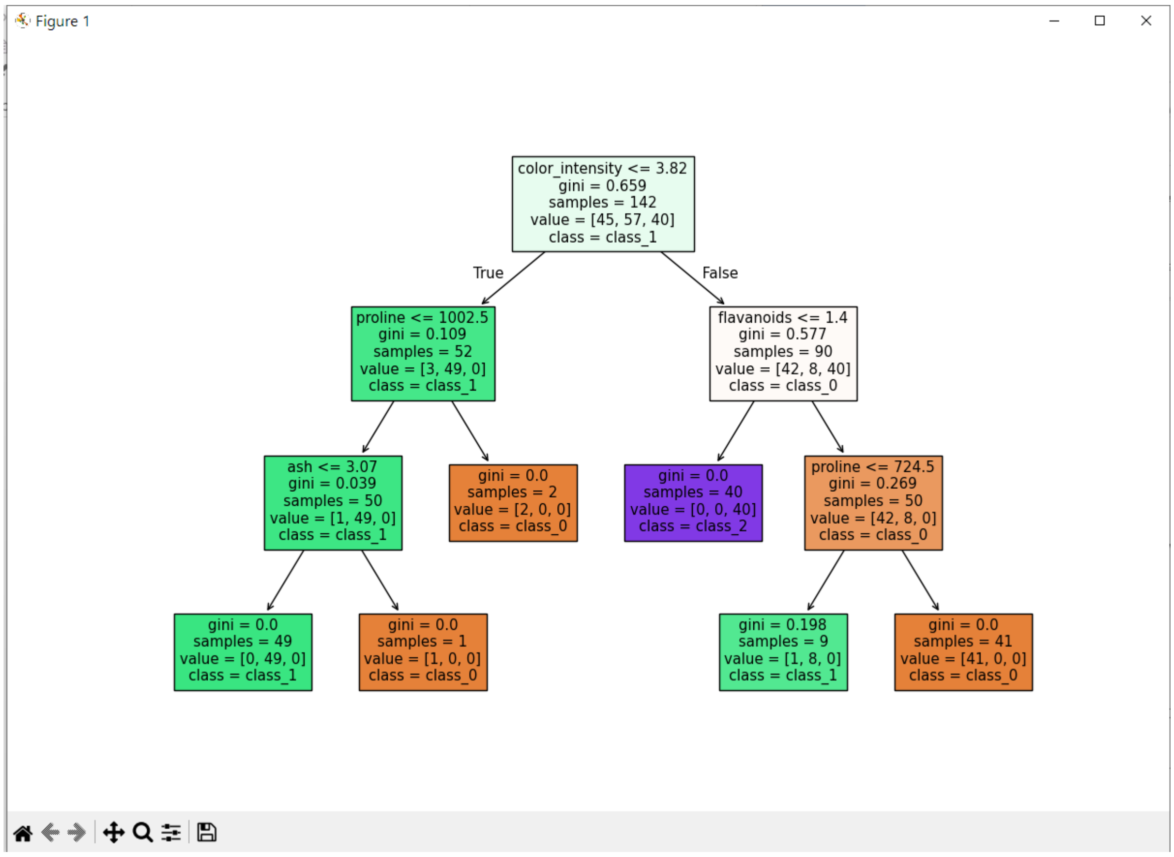
Date:	**9/12/2025**
Program no:05	Decision tree

Source Code:

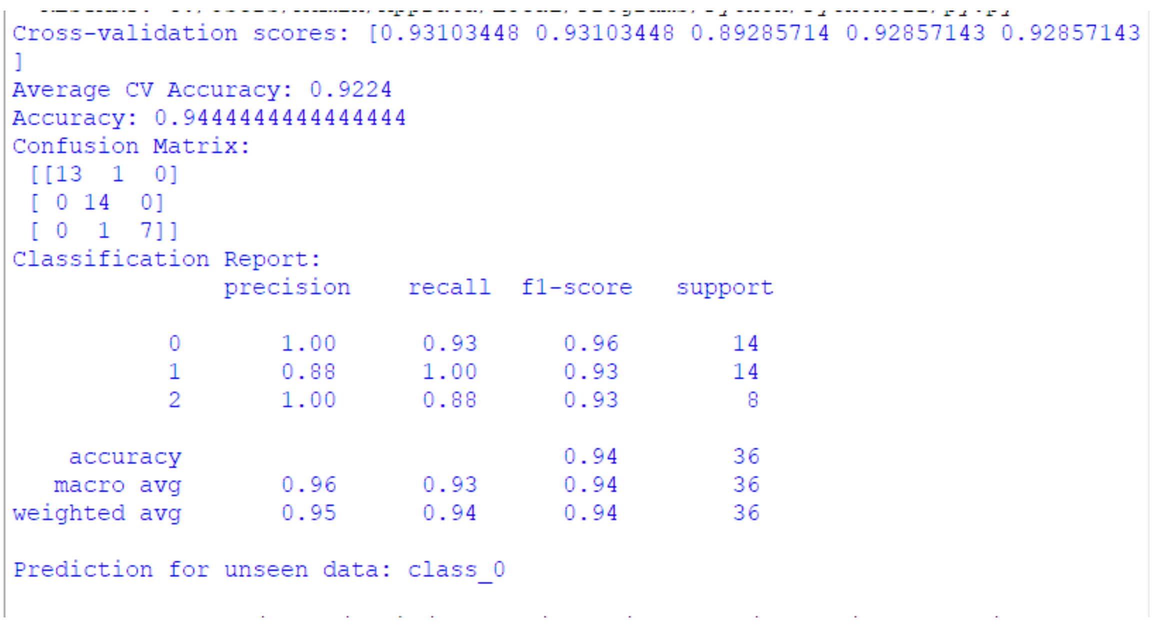
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

from sklearn.datasets import load_wine


# Load dataset

data = load_wine()

df = pd.DataFrame(data.data, columns=data.feature_names)

df['target'] = data.target


# Visualize dataset

sns.pairplot(df, hue='target', palette='Set1')

plt.show()


# Feature Engineering: Add a new feature (interaction term)

df['alcohol_flavanoids'] = df['alcohol'] * df['flavanoids']

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

y = df['target']


# Split dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Train Decision Tree model

model = DecisionTreeClassifier(max_depth=3, random_state=42)

model.fit(X_train, y_train)
```

```
# Visualize Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(model, feature_names=X.columns, class_names=data.target_names, filled=True)
plt.show()
# Cross-validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5)
print(f'Cross-validation scores: {cv_scores}')
print(f'Average CV Accuracy: {np.mean(cv_scores):.4f}')
# Evaluate on test data
y_pred = model.predict(X_test)
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Confusion Matrix:\n', confusion_matrix(y_test, y_pred))
print('Classification Report:\n', classification_report(y_test, y_pred))
# Test on unseen data (Fixed Warning Issue)
unseen_data = np.array([[13.5, 2.3, 2.5, 22.0, 85.0, 2.2, 2.8, 0.3, 1.6, 4.0, 1.05, 3.0, 750, 13.5*2.8]])
# Convert unseen data into a DataFrame with the correct column names
unseen_data_df = pd.DataFrame(unseen_data, columns=X_train.columns)
# Predict
unseen_pred = model.predict(unseen_data_df)
print(f'Prediction for unseen data: {data.target_names[unseen_pred[0]]}')
```





```
color_intensity <= 3.82
gini = 0.659
samples = 142
value = [45, 57, 40]
class = class_1

proline <= 1002.5
gini = 0.109
samples = 52
value = [3, 49, 0]
class = class_1

flavanoids <= 1.4
gini = 0.577
samples = 90
value = [42, 8, 40]
class = class_0

ash <= 3.07
gini = 0.039
samples = 50
value = [1, 49, 0]
class = class_1

gini = 0.0
samples = 2
value = [2, 0, 0]
class = class_0

gini = 0.0
samples = 40
value = [0, 0, 40]
class = class_2

proline <= 724.5
gini = 0.269
samples = 50
value = [42, 8, 0]
class = class_0

gini = 0.0
samples = 49
value = [0, 49, 0]
class = class_1

gini = 0.0
samples = 1
value = [1, 0, 0]
class = class_0

gini = 0.198
samples = 9
value = [1, 8, 0]
class = class_1

gini = 0.0
samples = 41
value = [41, 0, 0]
class = class_0
```



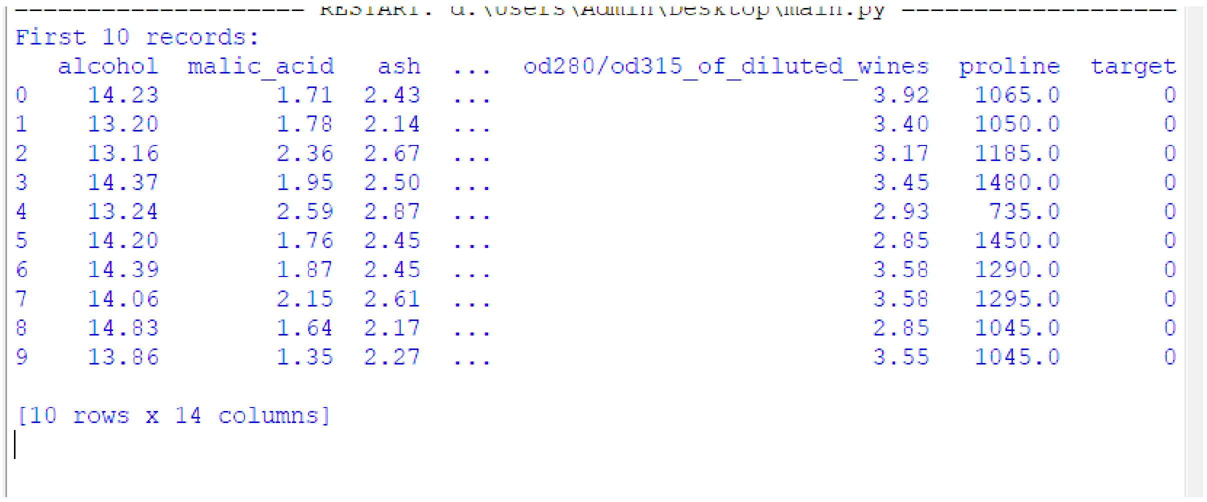
```
Cross-validation scores: [0.93103448 0.93103448 0.89285714 0.92857143 0.92857143
]
Average CV Accuracy: 0.9224
Accuracy: 0.9444444444444444
Confusion Matrix:
 [[13  1  0]
 [ 0 14  0]
 [ 0  1  7]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.93      0.96        14
           1       0.88      1.00      0.93        14
           2       1.00      0.88      0.93         8

    accuracy                           0.94        36
   macro avg       0.96      0.93      0.94        36
weighted avg       0.95      0.94      0.94        36

Prediction for unseen data: class_0
```

Date:	**9/01/2025**
Program no:06	**Support Vector Machine**

Source Code:

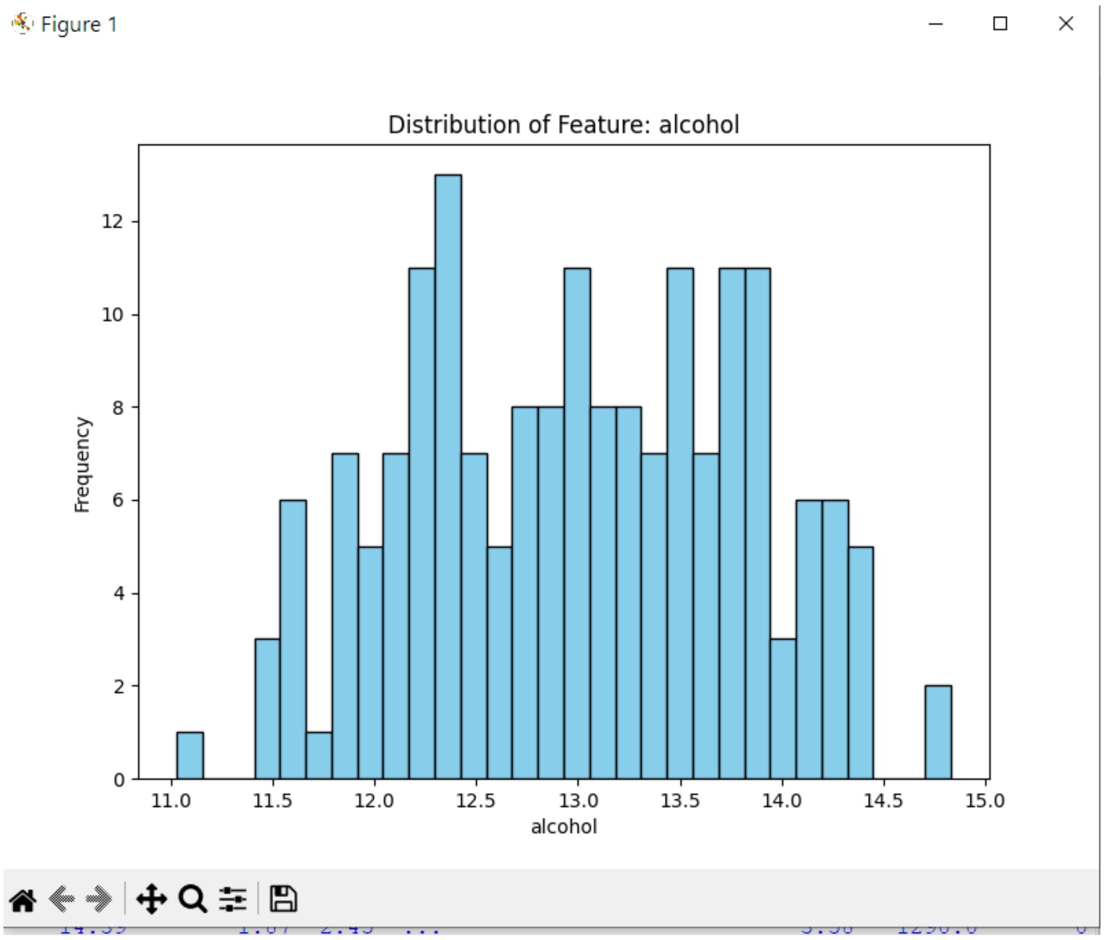
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm, metrics
# Load the Wine dataset
wine = datasets.load_wine()
# Convert to DataFrame for easier manipulation
data = pd.DataFrame(data=wine.data, columns=wine.feature_names)
data['target'] = wine.target
# 1. Print the first 10 records
print("First 10 records:")
print(data.head(10))
# Data visualization
# Plot a histogram of the first feature
plt.figure(figsize=(8, 6))
plt.hist(data[wine.feature_names[0]], bins=30, color='skyblue', edgecolor='black')
plt.title("Distribution of Feature: " + wine.feature_names[0])
plt.xlabel(wine.feature_names[0])
plt.ylabel("Frequency")
plt.show()
# Scatter plot of two features
plt.figure(figsize=(8, 6))
plt.scatter(data[wine.feature_names[0]], data[wine.feature_names[1]], c=data['target'],
cmap='viridis', alpha=0.7)
plt.title(f"Scatter Plot of {wine.feature_names[0]} vs {wine.feature_names[1]}")
plt.xlabel(wine.feature_names[0])
plt.ylabel(wine.feature_names[1])
plt.colorbar(label='Target (0, 1, 2)')
plt.show()
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3,
random_state=109)
# Create and train SVM model
clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
# Predict for the test dataset
y_pred = clf.predict(X_test)
# Model evaluation
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision (weighted):", metrics.precision_score(y_test, y_pred, average='weighted'))
print("Recall (weighted):", metrics.recall_score(y_test, y_pred, average='weighted'))
# 3. Predict for manually provided unseen data
# Manually create 5 unseen data points
manual_unseen_data = np.array([
 [13.2, 2.7, 2.36, 20.0, 95.0, 1.1, 1.7, 0.36, 1.2, 3.5, 1.02, 3.4, 820],
 [12.8, 1.9, 2.14, 15.2, 100.0, 2.5, 2.1, 0.26, 1.8, 3.2, 0.98, 2.8, 750],
 [13.7, 2.6, 2.5, 18.5, 101.0, 1.6, 1.6, 0.43, 1.3, 3.6, 1.12, 3.0, 880],
 [12.5, 2.4, 2.25, 17.0, 85.0, 1.3, 1.9, 0.31, 1.5, 3.0, 1.01, 2.5, 720],
 [14.0, 3.0, 2.55, 25.0, 105.0, 2.8, 2.9, 0.33, 2.2, 4.0, 1.25, 3.8, 1050]
])
# Predict outcomes for manual unseen data
predicted_outcomes = clf.predict(manual_unseen_data)
# Display manually provided unseen data predictions
print("\nManual Unseen Data Predictions:")
for i in range(len(manual_unseen_data)):
 print(f"Data point {i+1}: Predicted = Class {predicted_outcomes[i]}")
```

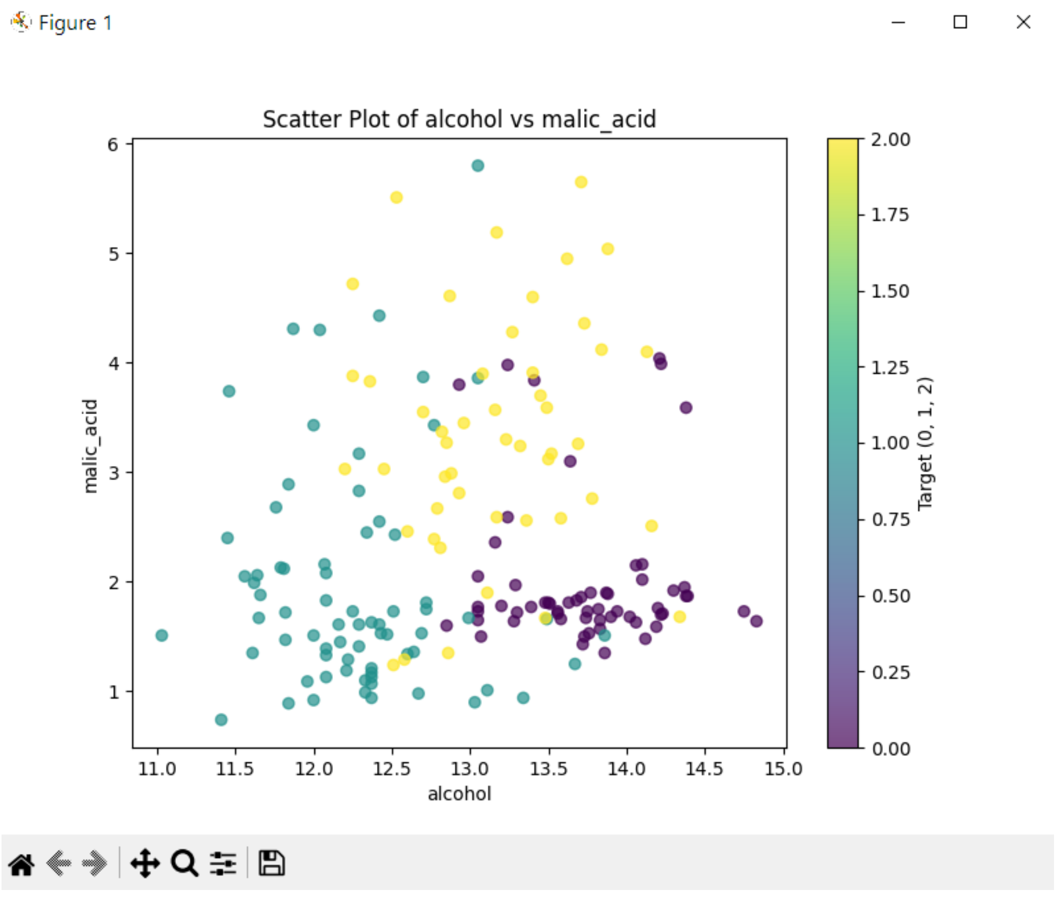
OUTPUT:

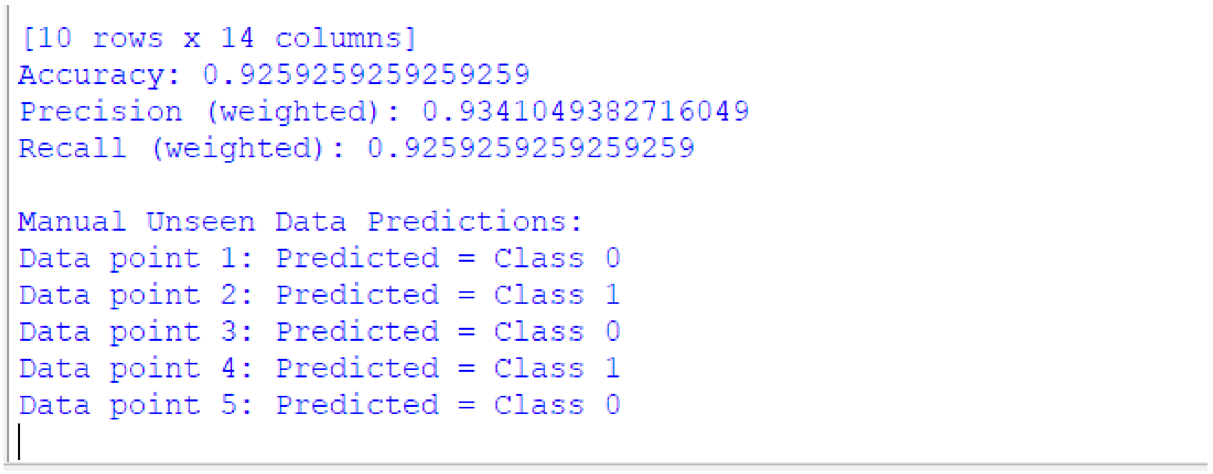


```
==================== RESTART: d:\Users\Admin\Desktop\main.py ===================
First 10 records:
   alcohol  malic_acid   ash  ...  od280/od315_of_diluted_wines  proline  target
0    14.23        1.71  2.43  ...                          3.92   1065.0       0
1    13.20        1.78  2.14  ...                          3.40   1050.0       0
2    13.16        2.36  2.67  ...                          3.17   1185.0       0
3    14.37        1.95  2.50  ...                          3.45   1480.0       0
4    13.24        2.59  2.87  ...                          2.93    735.0       0
5    14.20        1.76  2.45  ...                          2.85   1450.0       0
6    14.39        1.87  2.45  ...                          3.58   1290.0       0
7    14.06        2.15  2.61  ...                          3.58   1295.0       0
8    14.83        1.64  2.17  ...                          2.85   1045.0       0
9    13.86        1.35  2.27  ...                          3.55   1045.0       0

[10 rows x 14 columns]
```

OUTPUT:






```
[10 rows x 14 columns]
Accuracy: 0.9259259259259259
Precision (weighted): 0.9341049382716049
Recall (weighted): 0.9259259259259259

Manual Unseen Data Predictions:
Data point 1: Predicted = Class 0
Data point 2: Predicted = Class 1
Data point 3: Predicted = Class 0
Data point 4: Predicted = Class 1
Data point 5: Predicted = Class 0
```

Date:	**10/02/2025**
Parogram no:06	**Naïve bayes**

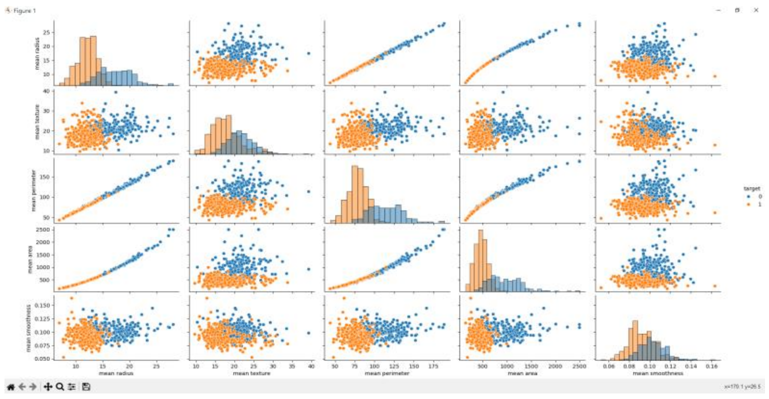
Source Code:

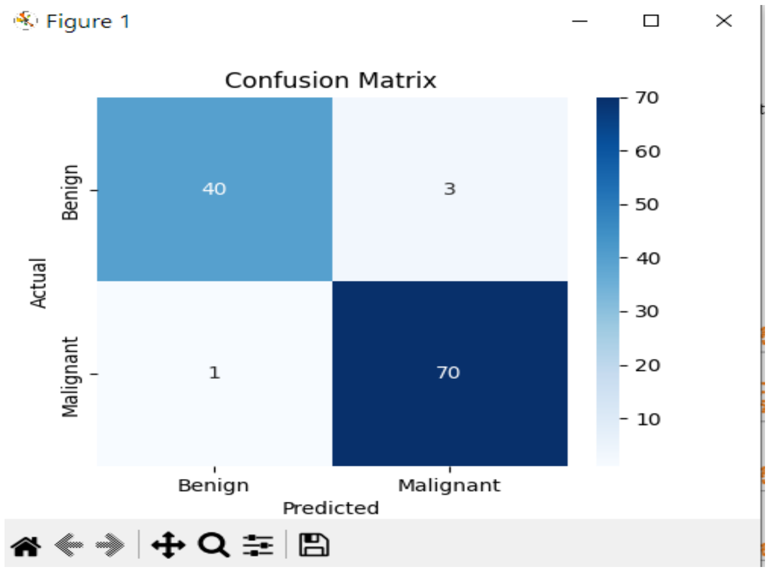
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load Breast Cancer Dataset
cancer = datasets.load_breast_cancer()
df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
df['target'] = cancer.target

# Convert target to string for better visualization
df['target'] = df['target'].astype(str)

# Data Visualization (First 5 features)
sns.pairplot(df.iloc[:, :5].join(df[['target']]), hue='target', diag_kind='hist')
plt.show()

# Feature Scaling
scaler = StandardScaler()
X = scaler.fit_transform(df.iloc[:, :-1])  # Scaling features
y = cancer.target  # Keeping target as an array

# Splitting the Data (Train: 80%, Test: 20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

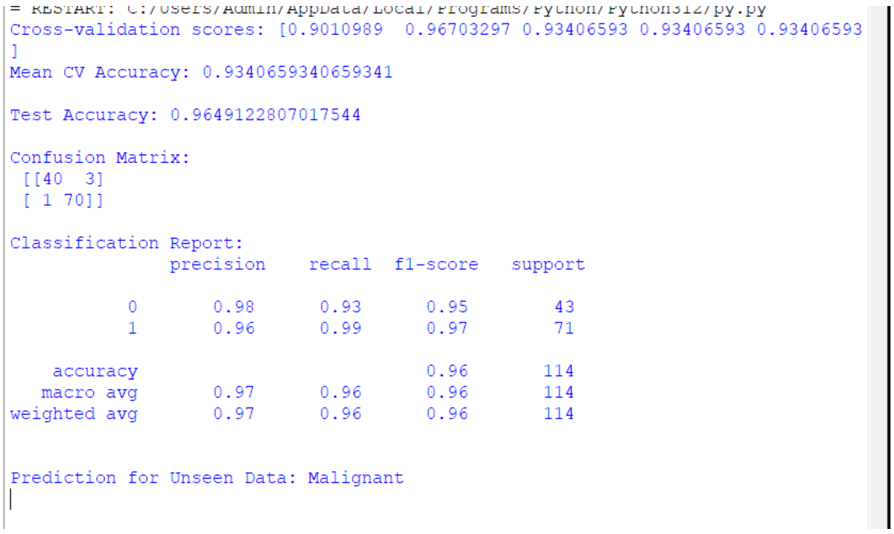
```
# Model Implementation (Naïve Bayes)
model = GaussianNB()
model.fit(X_train, y_train)

# Cross-validation (5-fold)
cross_val_scores = cross_val_score(model, X_train, y_train, cv=5)
print("Cross-validation scores:", cross_val_scores)
print("Mean CV Accuracy:", np.mean(cross_val_scores))

# Model Evaluation on Test Data
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("\nTest Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)

# Visualizing Confusion Matrix
plt.figure(figsize=(5,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Benign', 'Malignant'],
yticklabels=['Benign', 'Malignant'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Testing on Unseen Data (New Sample)
unseen_sample = [[15.0, 20.0, 100.0, 700.0, 0.1, 0.2, 0.3, 0.4, 0.1, 0.05,
              0.5, 1.0, 3.0, 50.0, 0.005, 0.02, 0.03, 0.004, 0.02, 0.005,
              20.0, 30.0, 140.0, 1000.0, 0.15, 0.3, 0.4, 0.2, 0.2, 0.07]]
```

```
# Convert to DataFrame for proper feature scaling
unseen_sample_df = pd.DataFrame(unseen_sample, columns=cancer.feature_names)

# Scale using previously fitted StandardScaler
unseen_sample_scaled = scaler.transform(unseen_sample_df)

# Predict class
prediction = model.predict(unseen_sample_scaled)

# Corrected output: 1 is Benign, 0 is Malignant
print("\nPrediction for Unseen Data:", "Benign" if prediction[0] == 1 else "Malignant")
```

<u>OUTPUT:</u>







```
= RESTART: C:/Users/Admin/AppData/Local/Programs/Python/Python312/py.py
Cross-validation scores: [0.9010989  0.96703297 0.93406593 0.93406593 0.93406593
]
Mean CV Accuracy: 0.9340659340659341

Test Accuracy: 0.9649122807017544

Confusion Matrix:
 [[40  3]
 [ 1 70]]

Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114


Prediction for Unseen Data: Malignant
```

Date	**3/03/2025**
Program no:07	**Artificial Neural Network**

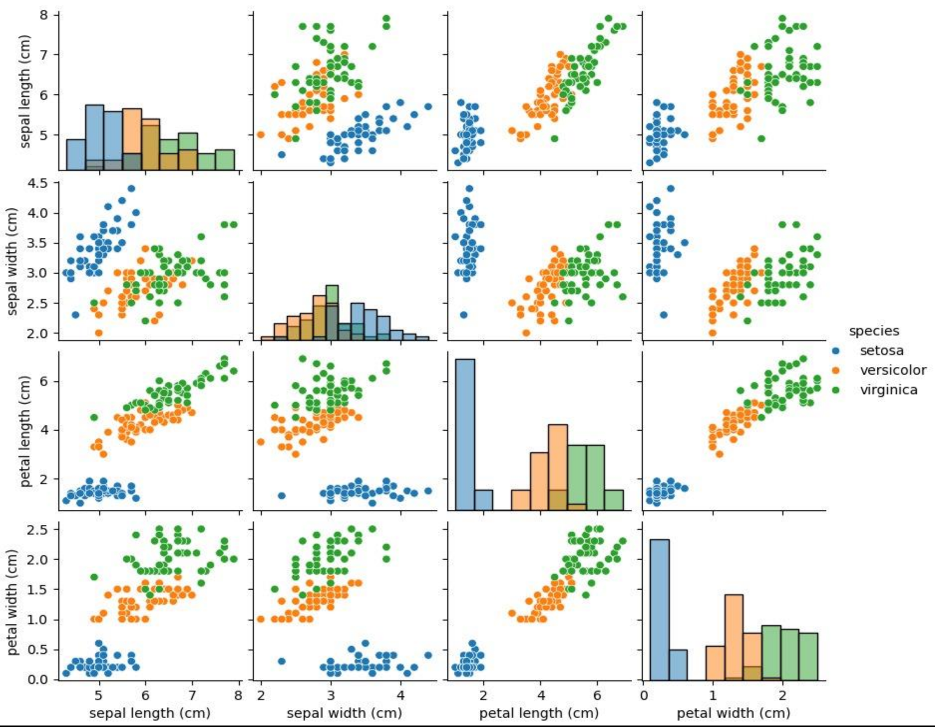
Source Code:

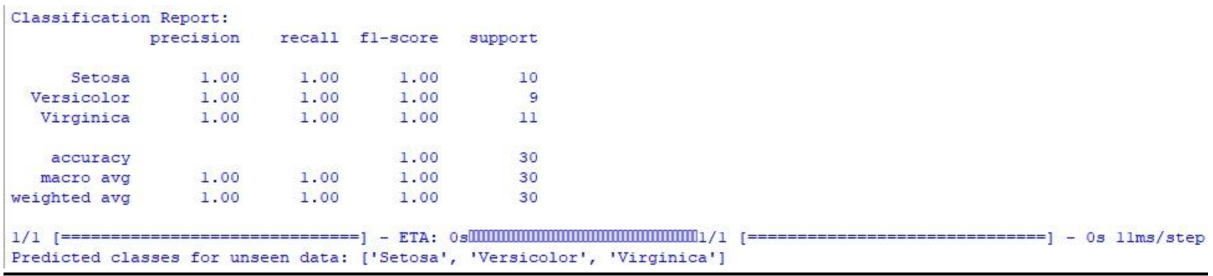
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import tensorflow as tf

# Step 1: Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Step 2: Preprocess the Data
scaler = StandardScaler()
X = scaler.fit_transform(X)  # Standardize features
y = to_categorical(y, num_classes=3)  # One-hot encoding of target labels

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Build the ANN Model
model = Sequential()
```

```
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(3, activation='softmax'))  # 3 output classes (Setosa, Versicolor, Virginica)

# Step 4: Compile the Model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Step 5: Train the Model
history = model.fit(X_train, y_train, epochs=50, batch_size=10, validation_data=(X_test, y_test))

# Step 6: Evaluate the Model
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy*100:.2f}%')

# Step 7: Make Predictions on the Test Data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Convert predictions to class labels
y_true = np.argmax(y_test, axis=1)  # Convert true labels to class labels

# Step 8: Calculate Precision, Recall, F1 Score
precision = precision_score(y_true, y_pred_classes, average='weighted')
recall = recall_score(y_true, y_pred_classes, average='weighted')
f1 = f1_score(y_true, y_pred_classes, average='weighted')

print(f'Precision (weighted): {precision:.2f}')
print(f'Recall (weighted): {recall:.2f}')
print(f'F1 Score (weighted): {f1:.2f}')

# Print classification report
print("\nClassification Report:")
print(classification_report(y_true, y_pred_classes, target_names=['Setosa', 'Versicolor', 'Virginica']))
```

```
# Step 9: Make Predictions on Unseen Data
# Example of unseen data: random new iris data (sepal length, sepal width, petal length, petal width)
unseen_data = np.array([[5.1, 3.5, 1.4, 0.2],  # Setosa-like
                        [7.0, 3.2, 4.7, 1.4],  # Versicolor-like
                        [6.3, 3.3, 6.0, 2.5]]) # Virginica-like

# Standardize unseen data based on the training set statistics
unseen_data = scaler.transform(unseen_data)

# Predict the classes for the unseen data
predictions = model.predict(unseen_data)
predicted_classes = np.argmax(predictions, axis=1)

# Mapping predicted classes to iris species names
iris_species = ['Setosa', 'Versicolor', 'Virginica']
predicted_species = [iris_species[i] for i in predicted_classes]

print("Predicted classes for unseen data:", predicted_species)

# Step 10: Visualize Data
# 1. Visualizing the Iris Dataset using pairplot
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['species'] = [iris_species[i] for i in iris.target]

# Pairplot to see the distribution of each feature
sns.pairplot(iris_df, hue="species")
plt.suptitle("Iris Dataset Pairplot", y=1.02)
plt.show()

# 2. Plotting Training & Validation Accuracy and Loss during Training
# Accuracy plot
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training vs Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

OUTPUT:





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

      Setosa       1.00      1.00      1.00        10
  Versicolor       1.00      1.00      1.00         9
   Virginica       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

1/1 [==============================] - ETA: 0s1/1 [==============================] - 0s 11ms/step
Predicted classes for unseen data: ['Setosa', 'Versicolor', 'Virginica']
```

PART-B

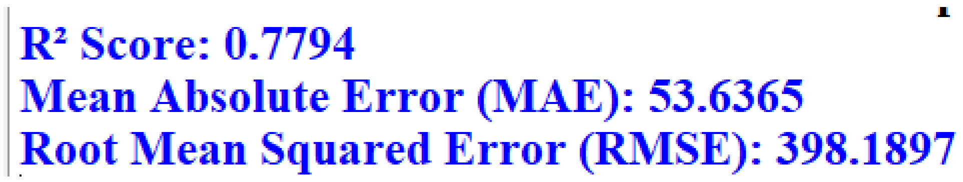
1.GUI.py

```
import tkinter as tk
from tkinter import ttk, messagebox, Canvas
import pandas as pd
import numpy as np
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# Load dataset
df = pd.read_csv('reduced_dataset.csv')
df.fillna(0, inplace=True)

# Encoding season values
season_map = {'Kharif': 1, 'Rabi': 2, 'Summer': 3, 'Winter': 4, 'Whole Year': 5}
df['season_encoded'] = df['season'].map(season_map).fillna(0)

# Define features and target
features = ['area', 'production', 'season_encoded', 'state_name']
target = 'crop_type'

X = df[features]
y = df[target]

# One-hot encoding for categorical data (state_name)
ohe = OneHotEncoder(sparse_output=False, drop='first', handle_unknown='ignore')
X_encoded = ohe.fit_transform(X[['state_name']])
encoded_df = pd.DataFrame(X_encoded, columns=ohe.get_feature_names_out())
```

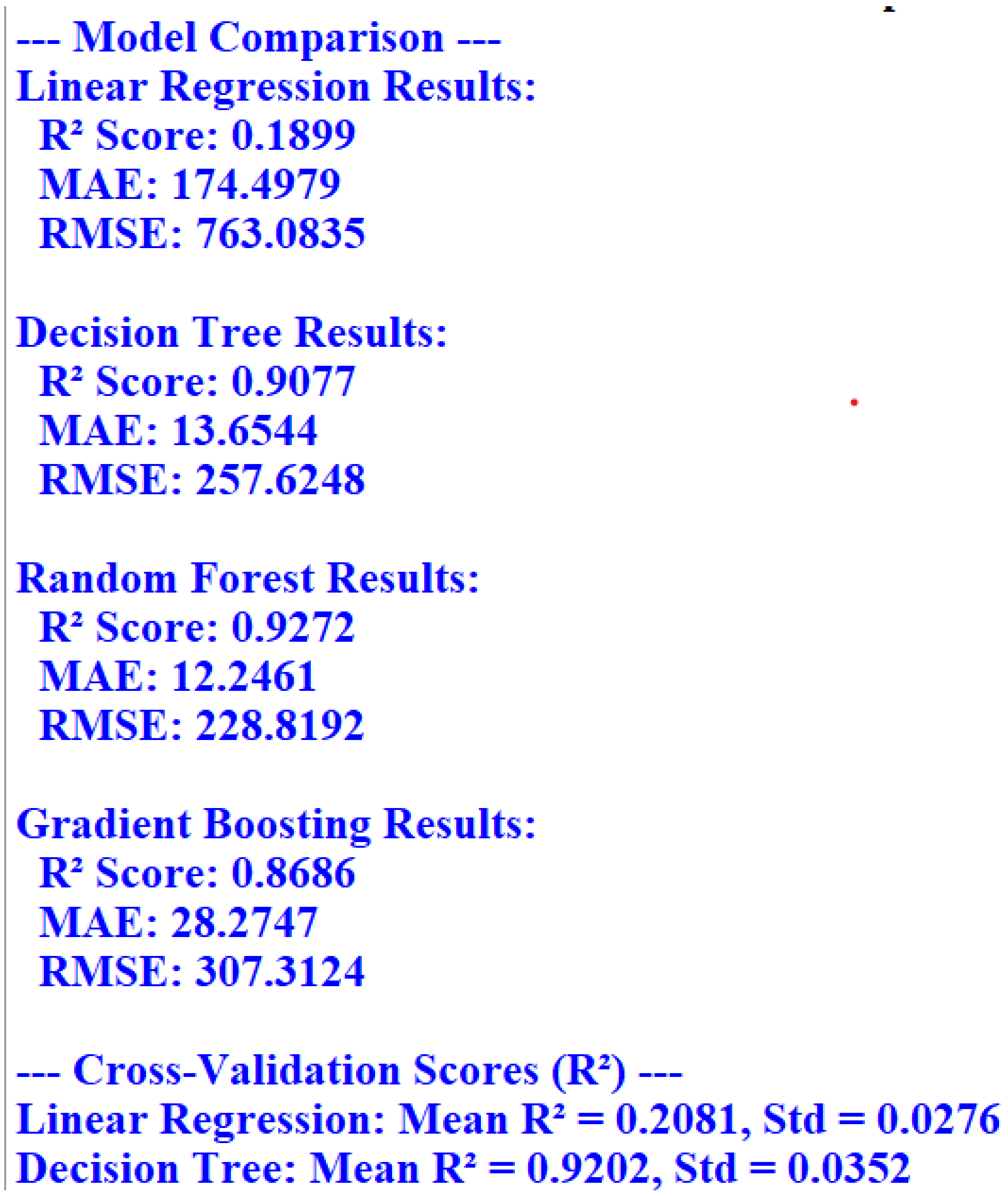
```
# Final feature set
X_final = pd.concat([
    X[['area', 'production', 'season_encoded']].reset_index(drop=True),
    encoded_df.reset_index(drop=True)
], axis=1)

# Scaling numerical values
scaler = StandardScaler()
X_final[['area', 'production', 'season_encoded']] = scaler.fit_transform(
    X_final[['area', 'production', 'season_encoded']]
)

# Train model
model = DecisionTreeClassifier(random_state=42)
model.fit(X_final, y)

# Tkinter GUI setup
root = tk.Tk()
root.title("Crop Prediction System with Advanced Visualizations")
root.geometry("1100x850")
root.configure(bg='#e3f2fd')

# Scrollable frame
canvas = Canvas(root)
scroll_y = ttk.Scrollbar(root, orient="vertical", command=canvas.yview)
frame = ttk.Frame(canvas)

frame.bind("<Configure>", lambda e: canvas.configure(scrollregion=canvas.bbox("all")))

canvas.create_window((0, 0), window=frame, anchor="nw")
canvas.configure(yscrollcommand=scroll_y.set)
canvas.pack(side="left", fill="both", expand=True)
```

```
scroll_y.pack(side="right", fill="y")

# Input Section
input_frame = ttk.LabelFrame(frame, text="🌱 Enter Crop Details", padding=25)
input_frame.pack(pady=20, padx=30, fill="x")

def create_input(label_text):
    label = ttk.Label(input_frame, text=label_text)
    label.pack(pady=8)
    entry = ttk.Entry(input_frame, width=35)
    entry.pack()
    return entry

area_entry = create_input("Area (in hectares):")
production_entry = create_input("Production (in tonnes):")

season_label = ttk.Label(input_frame, text="Season:")
season_label.pack(pady=8)
season_combo = ttk.Combobox(input_frame, values=list(season_map.keys()), width=33)
season_combo.pack()

state_label = ttk.Label(input_frame, text="State:")
state_label.pack(pady=8)
state_combo = ttk.Combobox(input_frame, values=df['state_name'].unique().tolist(), width=33)
state_combo.pack()

# Table for crop types
table_frame = ttk.Frame(frame)
table_frame.pack(pady=10)

tree = ttk.Treeview(table_frame, columns=("Crop Type", "Crops"), show='headings', height=7)
tree.heading("Crop Type", text="Crop Type")
tree.heading("Crops", text="Crop Name")
tree.column("Crop Type", width=200)
tree.column("Crops", width=600)
tree.pack(side="left", fill="y")

scrollbar = ttk.Scrollbar(table_frame, orient="vertical", command=tree.yview)
tree.configure(yscrollcommand=scrollbar.set)
scrollbar.pack(side="right", fill="y")

# Graph visualization frame
plot_frame = ttk.Frame(frame)
plot_frame.pack(pady=10)

result_label = ttk.Label(frame, text="", font=('Verdana', 16, 'bold'))
result_label.pack(pady=10)

# Function to visualize crop comparison
def visualize_comparison(prediction):
    try:
        related_crops = df[df['crop_type'] == prediction]['crop_name'].value_counts()

        if related_crops.empty:
            messagebox.showinfo("Info", "No related crops found for visualization.")
            return

        fig, ax = plt.subplots(figsize=(12, 6))  # Increased figure size
        related_crops.plot(kind='bar', color='#4caf50', ax=ax)

        ax.set_ylabel('Occurrences', fontsize=12)
        ax.set_xlabel('Crop Names', fontsize=12)
        ax.set_title(f'Comparison of Crops in {prediction} Category', fontsize=14, fontweight='bold')
        plt.xticks(rotation=45, ha="right", fontsize=10)  # Rotate labels
        plt.subplots_adjust(bottom=0.35, left=0.1, right=0.95, top=0.9)  # Adjust layout

        for widget in plot_frame.winfo_children():
            widget.destroy()

        canvas = FigureCanvasTkAgg(fig, master=plot_frame)
        canvas.draw()
        canvas.get_tk_widget().pack()

    except Exception as e:
        messagebox.showerror("Error", f"Error in visualization: {str(e)}")

# Function to update table
def update_table(prediction):
    tree.delete(*tree.get_children())  # Clear old data

    related_crops = df[df['crop_type'] == prediction][['crop_type', 'crop_name']].drop_duplicates()

    if related_crops.empty:
        tree.insert("", "end", values=("No Data", "No related crops found"))
        return

    for _, row in related_crops.iterrows():
        tree.insert("", "end", values=(row['crop_type'], row['crop_name']))

# Crop prediction function
def predict_crop():
    try:
        area = float(area_entry.get())
        production = float(production_entry.get())
        season = season_map.get(season_combo.get(), 0)
        state = state_combo.get()

        if not state:
            messagebox.showwarning("Warning", "Please select a state.")
            return

        input_df = pd.DataFrame({'area': [area], 'production': [production], 'season_encoded': [season]})
        encoded_input = ohe.transform(np.array([[state]]))
        encoded_input_df = pd.DataFrame(encoded_input, columns=ohe.get_feature_names_out())

        final_input = pd.concat([input_df.reset_index(drop=True), encoded_input_df], axis=1)
        final_input[['area', 'production', 'season_encoded']] = scaler.transform(
            final_input[['area', 'production', 'season_encoded']]
        )

        prediction = model.predict(final_input)[0]
        result_label.config(text=f"🌾 Predicted Crop: {prediction}", foreground='#1b5e20')

        update_table(prediction)  # Update table with predicted crops
        visualize_comparison(prediction)  # Show graph

    except ValueError:
        messagebox.showerror("Error", "Please enter valid numeric values for area and production.")
    except Exception as e:
        messagebox.showerror("Error", f"An error occurred: {str(e)}")

# Clear input fields
def clear_inputs():
    area_entry.delete(0, tk.END)
    production_entry.delete(0, tk.END)
    season_combo.set('')
    state_combo.set('')
    result_label.config(text='')
    tree.delete(*tree.get_children())  # Clear table
    for widget in plot_frame.winfo_children():
        widget.destroy()

# Buttons
predict_btn = ttk.Button(frame, text="Predict Crop", command=predict_crop)
predict_btn.pack(pady=12)

clear_btn = ttk.Button(frame, text="Clear Inputs", command=clear_inputs)
clear_btn.pack(pady=5)

# Run Tkinter loop
root.mainloop()
```

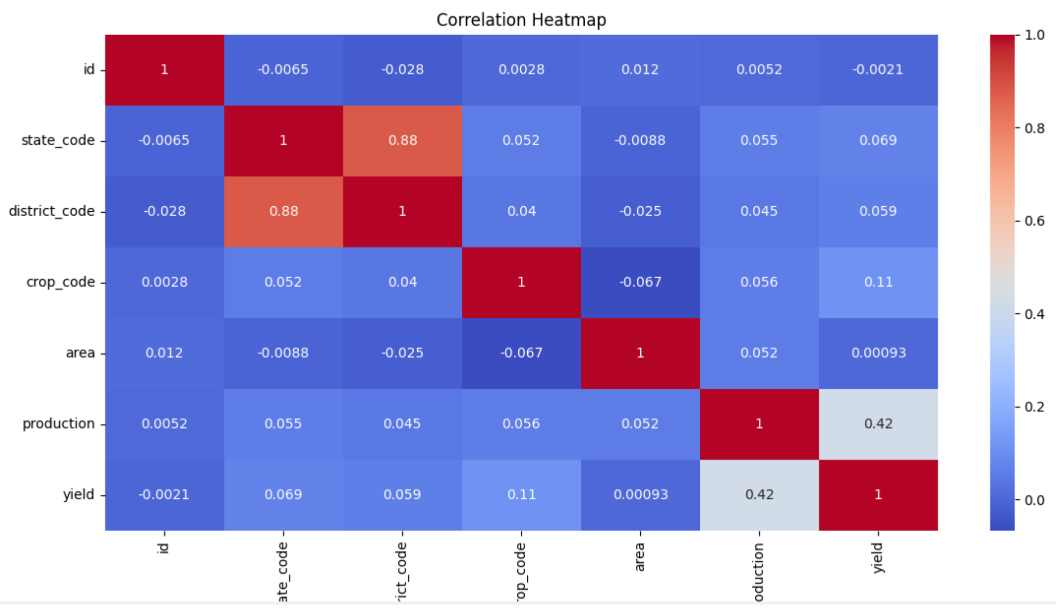
OUTPUT:



```
R² Score: 0.7794
Mean Absolute Error (MAE): 53.6365
Root Mean Squared Error (RMSE): 398.1897
```

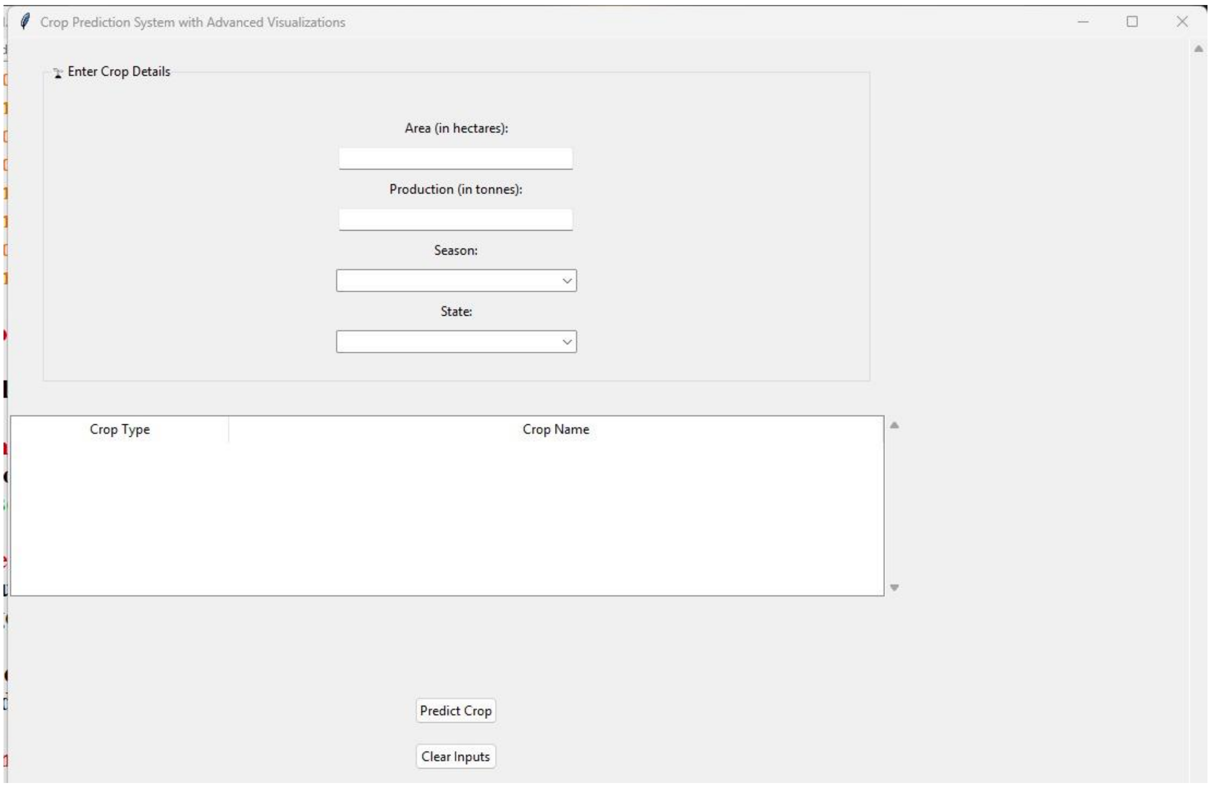


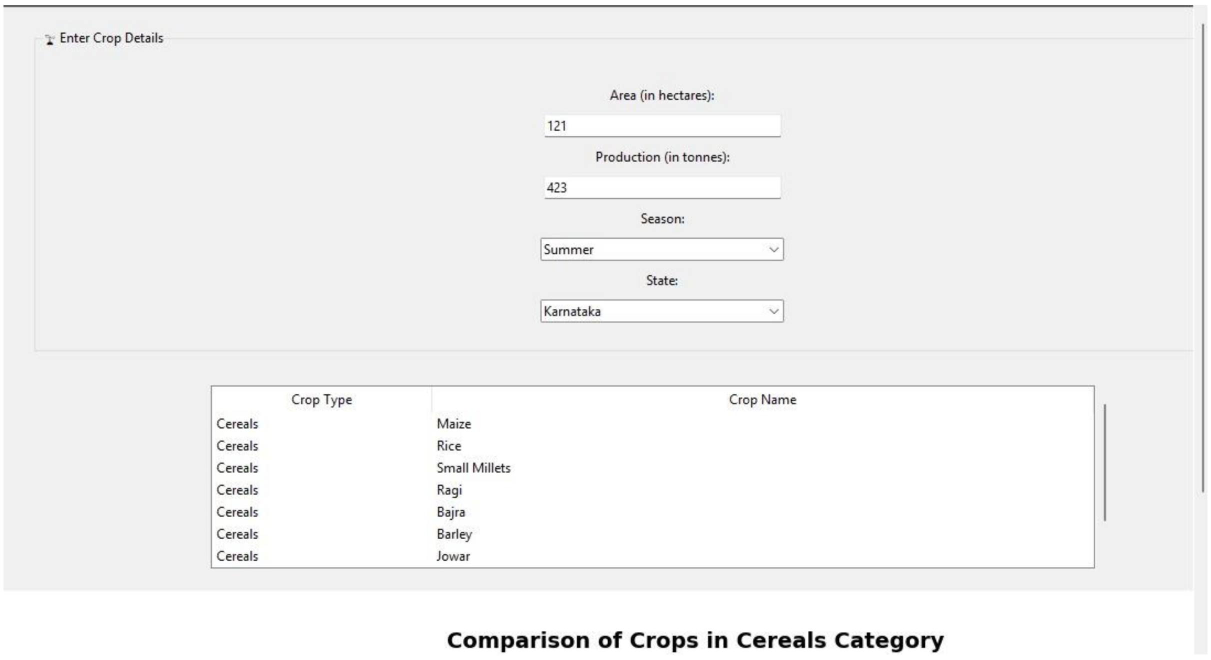
```
--- Model Comparison ---
Linear Regression Results:
  R² Score: 0.1899
  MAE: 174.4979
  RMSE: 763.0835

Decision Tree Results:
  R² Score: 0.9077
  MAE: 13.6544
  RMSE: 257.6248

Random Forest Results:
  R² Score: 0.9272
  MAE: 12.2461
  RMSE: 228.8192

Gradient Boosting Results:
  R² Score: 0.8686
  MAE: 28.2747
  RMSE: 307.3124

--- Cross-Validation Scores (R²) ---
Linear Regression: Mean R² = 0.2081, Std = 0.0276
Decision Tree: Mean R² = 0.9202, Std = 0.0352
```



	id	state_code	district_code	crop_code	area	production	yield
id	1	-0.0065	-0.028	0.0028	0.012	0.0052	-0.0021
state_code	-0.0065	1	0.88	0.052	-0.0088	0.055	0.069
district_code	-0.028	0.88	1	0.04	-0.025	0.045	0.059
crop_code	0.0028	0.052	0.04	1	-0.067	0.056	0.11
area	0.012	-0.0088	-0.025	-0.067	1	0.052	0.00093
production	0.0052	0.055	0.045	0.056	0.052	1	0.42
yield	-0.0021	0.069	0.059	0.11	0.00093	0.42	1

```
= RESTART: C:/Users/HP/OneDrive/Desktop/MachineLearning/Pro_feature.py
Training set shape: (64000, 102)
Testing set shape: (16000, 102)
```

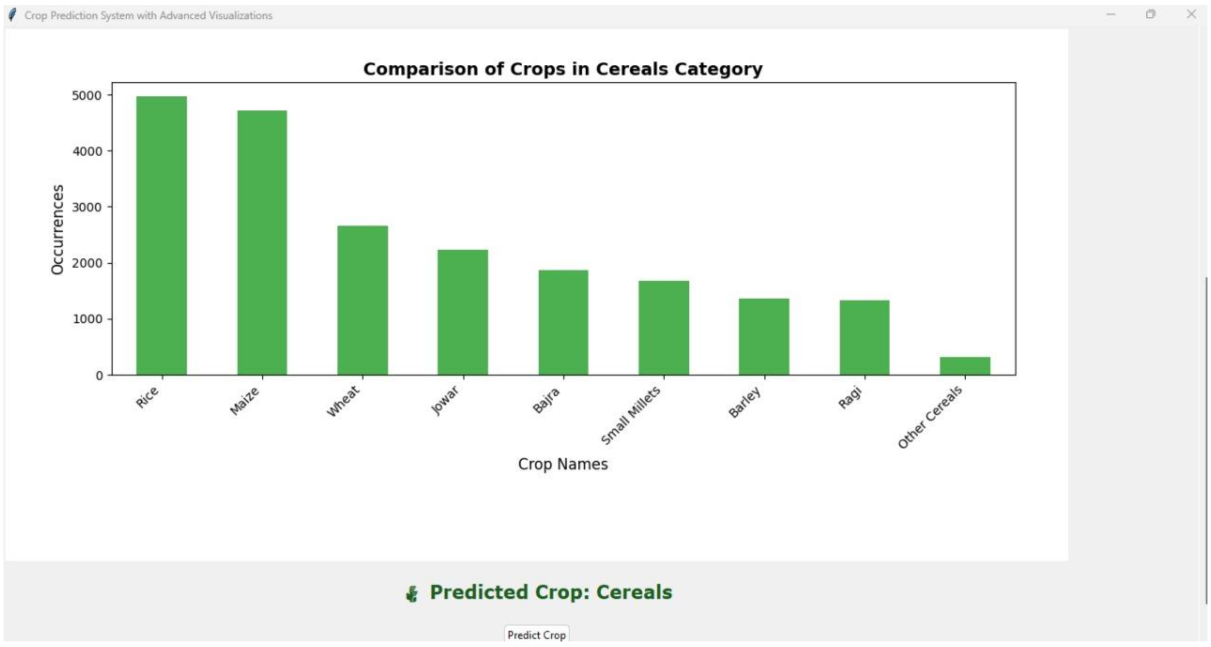




Area (in hectares): 121
Production (in tonnes): 423
Season: Summer
State: Karnataka

Crop Type	Crop Name
Cereals	Maize
Cereals	Rice
Cereals	Small Millets
Cereals	Ragi
Cereals	Bajra
Cereals	Barley
Cereals	Jowar

Comparison of Crops in Cereals Category



Predicted Crop: Cereals