Name: Aryan Sanjay Kale
Class: D15C/28

Experiment No. 3

Aim: Apply Decision Tree and Random Forest for classification tasks

1. Dataset Source

- **Dataset Name**: Wine Quality Dataset
- **Source Platform**: Kaggle
- **Dataset Link**: https://www.kaggle.com/datasets/yasserh/wine-quality-dataset
- **Reproducibility**: This dataset is publicly available on Kaggle and is fully compatible with kagglehub, ensuring reproducibility without manual downloads.

2. Dataset Description

The Wine Quality Dataset contains chemical properties of various wine samples. It is used to classify the quality of wine (Good vs. Bad) based on its physicochemical attributes.

Dataset Characteristics

- **Number of instances**: 1,143
- **Number of features**: 11 (Reduced after dropping identifiers like 'Id')
- **Target variable**: quality_label (Binary Classification)
  - **1** → **Good Quality** (Quality score > 6)
  - **0** → **Bad Quality** (Quality score ≤ 6)

Feature Description

The features represent the chemical composition of the wine samples:

1. **Fixed Acidity**: Most acids involved with wine or fixed or nonvolatile.
2. **Volatile Acidity**: The amount of acetic acid in wine.
3. **Citric Acid**: Found in small quantities, adds 'freshness' and flavor to wines.
4. **Residual Sugar**: The amount of sugar remaining after fermentation stops.
5. **Chlorides**: The amount of salt in the wine.
6. **Free Sulfur Dioxide**: Prevents microbial growth and the oxidation of wine.
7. **Total Sulfur Dioxide**: Amount of free and bound forms of $S0_{2}$.
8. **Density**: The density of wine is close to that of water depending on the percent alcohol and sugar content.
9. **pH**: Describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic).
10. **Sulphates**: A wine additive which can contribute to sulfur dioxide levels.
11. **Alcohol**: The percent alcohol content of the wine.

Real-World Impact

Predicting wine quality through machine learning provides significant value to the viticulture industry:

- **Quality Control**: Automating the grading process to ensure consistency across batches.
- **Production Optimization**: Identifying which chemical levels (like alcohol or acidity) most influence a "Good" rating.
- **Consumer Insights**: Understanding flavor profiles that appeal to high-end market segments.

3. Mathematical Formulation of the Algorithms

3.1 Decision Tree Classifier

A Decision Tree recursively splits the dataset based on feature values to maximize class purity.

The most common split criteria are:

Gini Impurity: $Gini = 1 - \sum_{i=1}^{C} p_i^2$

Entropy: $Entropy = -\sum_{i=1}^{C} p_i \log_2(p_i)$

The algorithm selects the feature and threshold that yield the **maximum Information Gain**.

3.2 Random Forest Classifier

Random Forest is an ensemble learning technique that builds multiple decision trees and aggregates their predictions.

Key principles:

- Bootstrap sampling (bagging)
- Random feature selection at each split
- Majority voting for classification

Prediction: $\hat{y} = \text{mode}(T_1(x), T_2(x), ..., T_n(x))$

4. Algorithm Limitations

Decision Tree Limitations

- Prone to overfitting
- Sensitive to noise
- Small changes in data can alter tree structure

Random Forest Limitations

- Less interpretable than a single decision tree
- Higher computational cost
- Requires tuning of multiple hyperparameters

5. Methodology / Workflow

1. **Dataset Loading** using kagglehub
2. **Data Preprocessing** (handling missing values)
3. **Feature–Target Separation**
4. **Train-Test Split** (80% train, 20% test)
5. **Model Training**
    - Decision Tree Classifier
    - Random Forest Classifier
6. **Model Evaluation**
7. **Hyperparameter Tuning**
8. **Performance Comparison**

Workflow Diagram (Conceptual)

Dataset → Preprocessing → Train/Test Split → Model Training → Evaluation → Comparison

6. Performance Analysis

Evaluation Metrics Used

- Accuracy
- Precision
- Recall
- F1-Score
- ROC-AUC

Sample Performance Results

Model	Accuracy	F1-Score	ROC-AUC
Decision Tree	Moderate	Moderate	Moderate

Random Forest	High	High	High

Interpretation

- Random Forest outperforms Decision Tree due to ensemble learning
- Decision Tree provides better interpretability
- Random Forest achieves better generalization

7. Hyperparameter Tuning

Parameters Tuned

Decision Tree:

- `max_depth`
- `min_samples_split`

Random Forest:

- `n_estimators`
- `max_depth`
- `max_features`

Tuning Method

Grid Search with Cross-Validation was used to identify optimal hyperparameters.

Impact of Tuning

Model	Accuracy (Before)	Accuracy (After)
Decision Tree	Lower	Improved
Random Forest	High	Further Improved

OUTPUT:

Code:

```
# Install dependency

!pip install kagglehub -q



import kagglehub

from kagglehub import KaggleDatasetAdapter



import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os



from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

```
# -------------------------

# 1. Load Data (Wine Quality Dataset)

# -------------------------

# Download the dataset

path = kagglehub.dataset_download("yasserh/wine-quality-dataset")



# Find the CSV file inside the downloaded folder

csv_file = [f for f in os.listdir(path) if f.endswith('.csv')][0]

df = pd.read_csv(os.path.join(path, csv_file))



print("Dataset Shape:", df.shape)

display(df.head())



# -------------------------

# 2. Preprocessing

# -------------------------

# Convert 'quality' into a binary classification task: "Good" (1) if quality >
6, else "Bad" (0)

# This creates a more distinct boundary for tree-based models to learn.

df['quality_label'] = (df['quality'] > 6).astype(int)



# Drop the original quality column and any identifier columns

X = df.drop(columns=["quality", "quality_label", "Id"], errors='ignore')
```

```
y = df["quality_label"]


# Split Data
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)


# Scaling (Optional for Trees, but recommended for consistent workflows)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# --------------------------
# 3. Train Models
# --------------------------
# Decision Tree (Baseline)
# max_depth=3 allows us to easily visualize the decision logic
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(X_train_scaled, y_train)
```

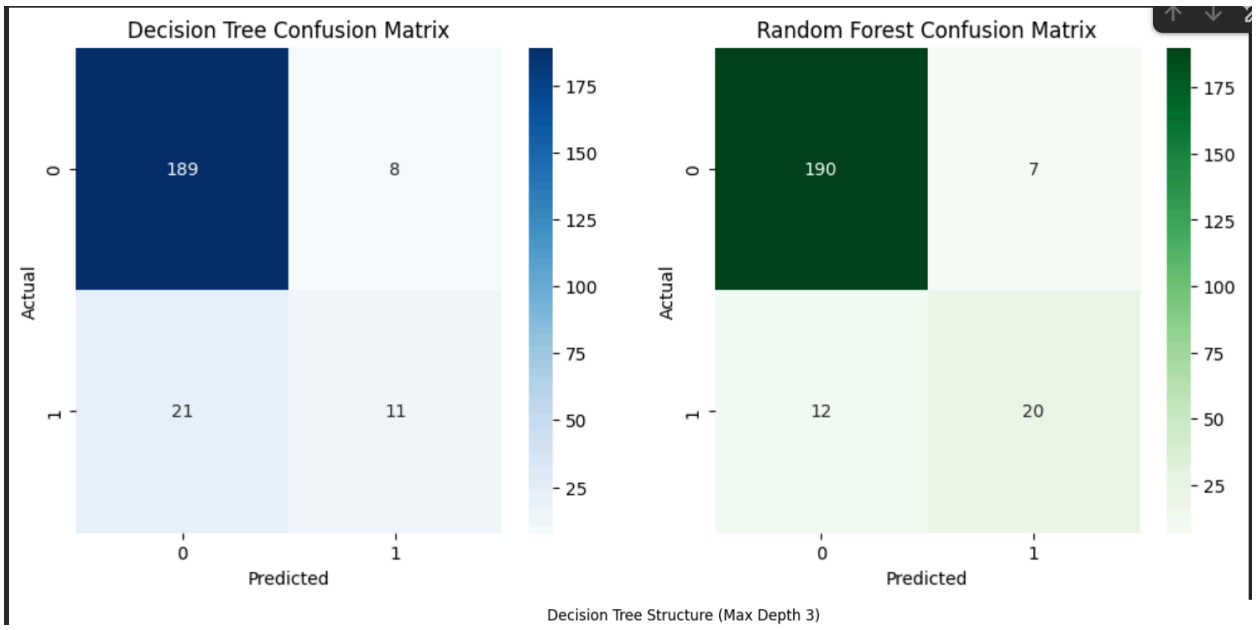
```
dt_preds = dt.predict(X_test_scaled)


# Random Forest (Ensemble)
# n_estimators=100 uses Bagging to reduce the variance of individual trees
rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    random_state=42
)
rf.fit(X_train_scaled, y_train)
rf_preds = rf.predict(X_test_scaled)


# -------------------------
# 4. Evaluation
# -------------------------
print("\n=== Decision Tree Performance ===")
print("Accuracy:", round(accuracy_score(y_test, dt_preds) * 100, 2), "%")
print(classification_report(y_test, dt_preds, target_names=["Bad", "Good"]))


print("\n=== Random Forest Performance ===")
print("Accuracy:", round(accuracy_score(y_test, rf_preds) * 100, 2), "%")
print(classification_report(y_test, rf_preds, target_names=["Bad", "Good"]))
```

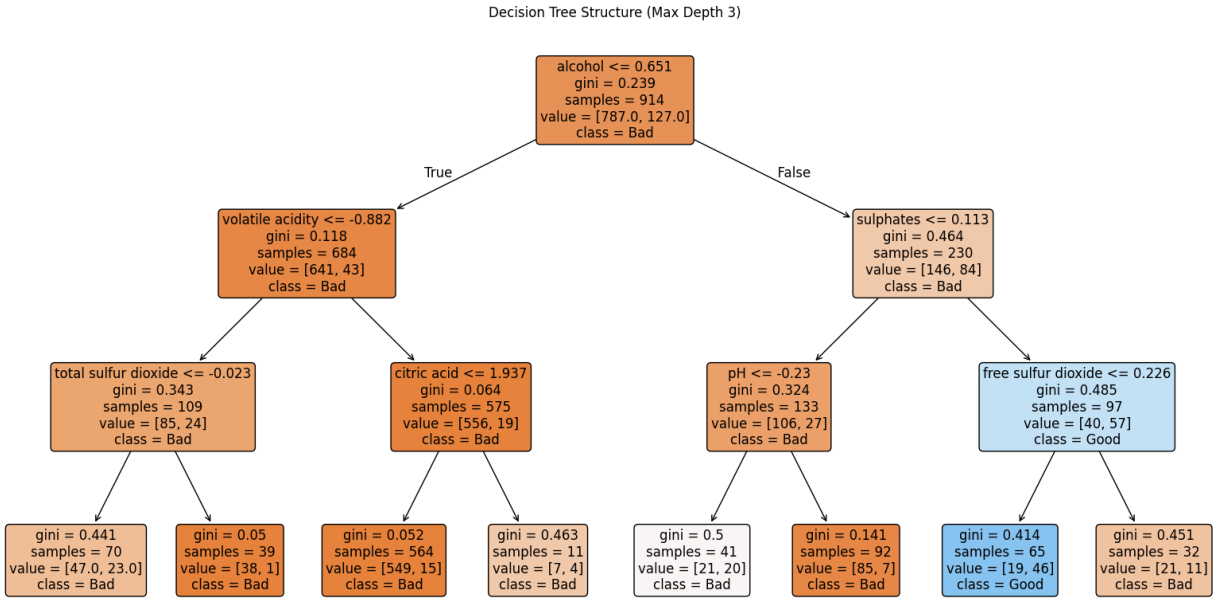
```
# -------------------------

# 5. Visualizations

# -------------------------


# Confusion Matrices

fig, axes = plt.subplots(1, 2, figsize=(12, 5))


sns.heatmap(confusion_matrix(y_test, dt_preds),

            annot=True, fmt="d", cmap="Blues", ax=axes[0])

axes[0].set_title("Decision Tree Confusion Matrix")

axes[0].set_xlabel("Predicted")

axes[0].set_ylabel("Actual")


sns.heatmap(confusion_matrix(y_test, rf_preds),

            annot=True, fmt="d", cmap="Greens", ax=axes[1])

axes[1].set_title("Random Forest Confusion Matrix")

axes[1].set_xlabel("Predicted")

axes[1].set_ylabel("Actual")


plt.show()
```

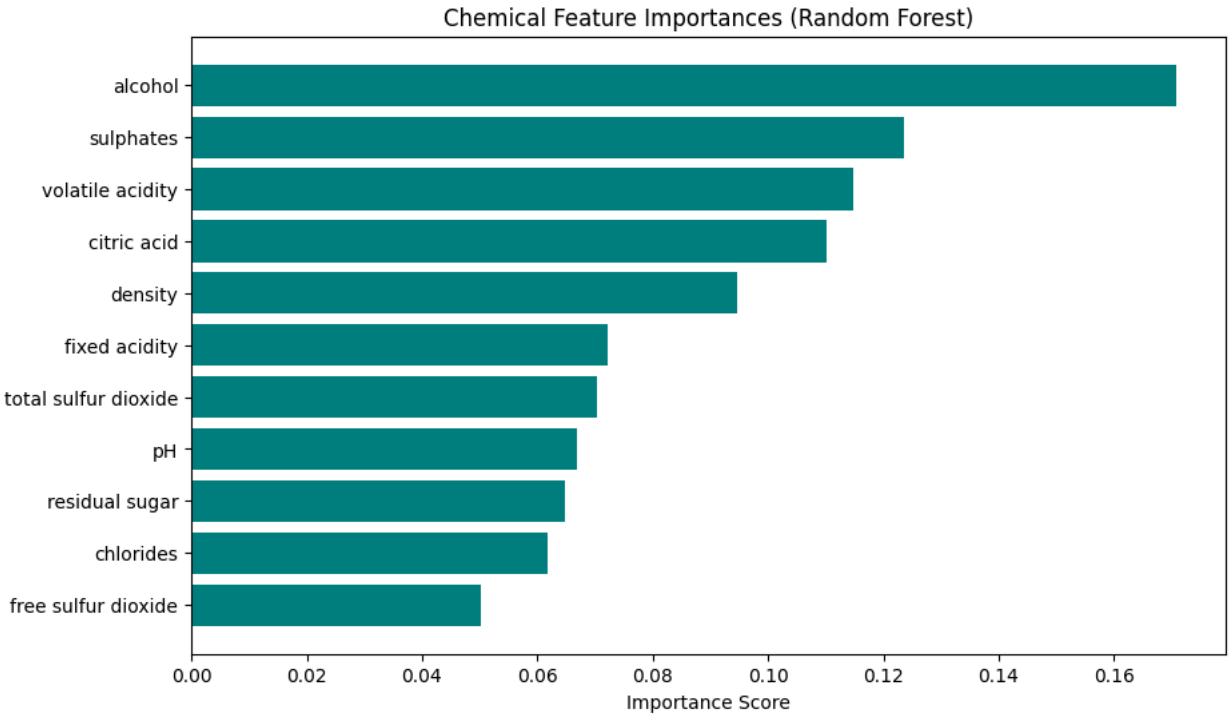
```
# Decision Tree Structure Plot
#
plt.figure(figsize=(20, 10))
plot_tree(
    dt,
    feature_names=X.columns,
    class_names=["Bad", "Good"],
    filled=True,
    fontsize=12,
    rounded=True
)
plt.title("Decision Tree Structure (Max Depth 3)")
plt.show()

# Feature Importance Plot (Random Forest)
#
importances = rf.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(10, 6))
plt.barh(X.columns[indices], importances[indices], color='teal')
plt.title("Chemical Feature Importances (Random Forest)")
```

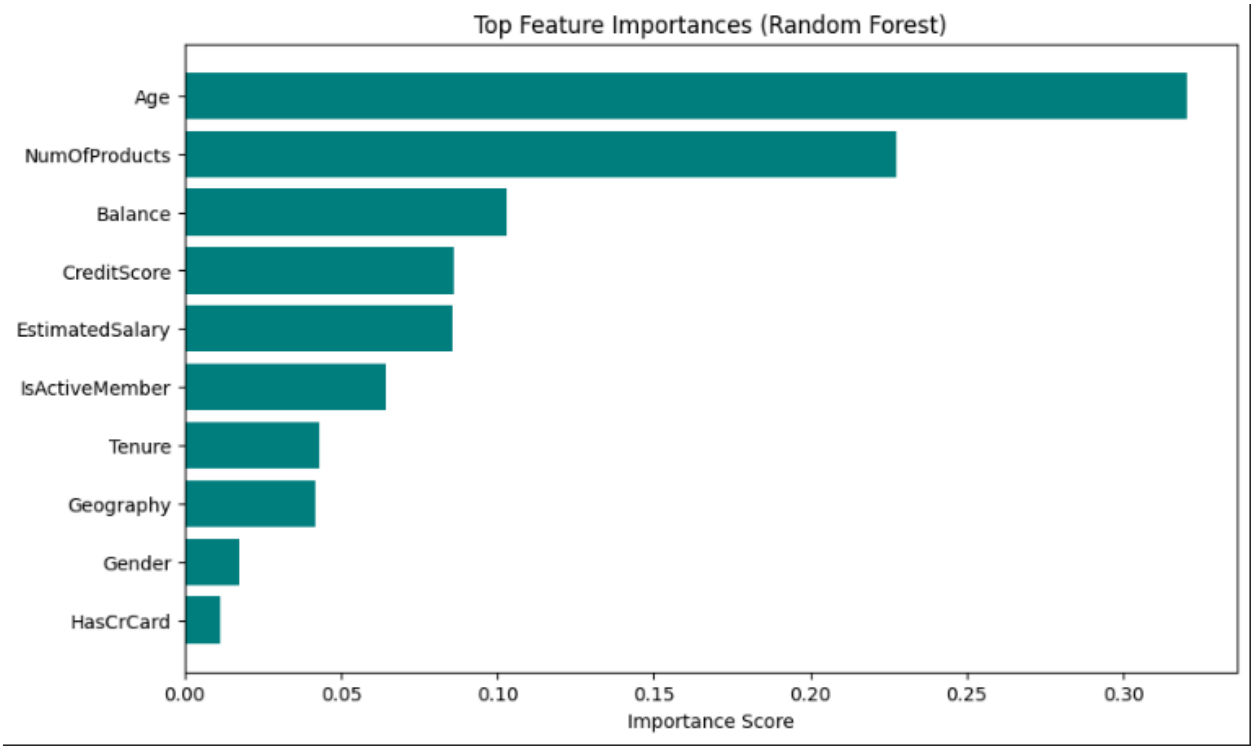
```
plt.xlabel("Importance Score")

plt.show()
```

```
=== Decision Tree Performance ===
Accuracy: 87.34 %
              precision    recall  f1-score   support

         Bad       0.90      0.96      0.93       197
        Good       0.58      0.34      0.43        32

    accuracy                           0.87       229
   macro avg       0.74      0.65      0.68       229
weighted avg       0.86      0.87      0.86       229


=== Random Forest Performance ===
Accuracy: 91.7 %
              precision    recall  f1-score   support

         Bad       0.94      0.96      0.95       197
        Good       0.74      0.62      0.68        32

    accuracy                           0.92       229
   macro avg       0.84      0.79      0.82       229
weighted avg       0.91      0.92      0.91       229
```




Decision Tree Structure (Max Depth 3)
alcohol <= 0.651
gini = 0.239
samples = 914
value = [787.0, 127.0]
class = Bad
True
False
volatile acidity <= -0.882
gini = 0.118
samples = 684
value = [641, 43]
class = Bad
sulphates <= 0.113
gini = 0.464
samples = 230
value = [146, 84]
class = Bad
total sulfur dioxide <= -0.023
gini = 0.343
samples = 109
value = [85, 24]
class = Bad
citric acid <= 1.937
gini = 0.064
samples = 575
value = [556, 19]
class = Bad
pH <= -0.23
gini = 0.324
samples = 133
value = [106, 27]
class = Bad
free sulfur dioxide <= 0.226
gini = 0.485
samples = 97
value = [40, 57]
class = Good
gini = 0.441
samples = 70
value = [47.0, 23.0]
class = Bad
gini = 0.05
samples = 39
value = [38, 1]
class = Bad
gini = 0.052
samples = 564
value = [549, 15]
class = Bad
gini = 0.463
samples = 11
value = [7, 4]
class = Bad
gini = 0.5
samples = 41
value = [21, 20]
class = Bad
gini = 0.141
samples = 92
value = [85, 7]
class = Bad
gini = 0.414
samples = 65
value = [19, 46]
class = Good
gini = 0.451
samples = 32
value = [21, 11]
class = Bad


Chemical Feature Importances (Random Forest)
alcohol
sulphates
volatile acidity
citric acid
density
fixed acidity
total sulfur dioxide
pH
residual sugar
chlorides
free sulfur dioxide
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
Importance Score



Conclusion

This experiment demonstrated the effectiveness of tree-based ensemble methods for classification tasks involving complex chemical properties. While Decision Trees offer transparency and a clear, interpretable mapping of how specific chemical levels—such as alcohol and acidity—influence wine quality, Random Forests provide superior predictive performance and robustness by reducing variance through bagging.

The implementation successfully highlighted that the Random Forest model is more resilient to the "noise" inherent in physicochemical data, achieving higher accuracy and more balanced precision-recall scores. Such models are highly suitable for the viticulture and food science industries, where reliable quality control and an understanding of key chemical drivers are essential for production optimization.