Name : Ch Mubashir

SAP : 56892.

Course : Data Minning.

Instructor : Tajamul Shahzad.

Task 09

Implementing Random Forest on the Wine Quality Dataset.

Lab Tasks :

1. **Load the Dataset: Load the Wine Quality dataset using pandas. Display the first five rows and check for missing values.**

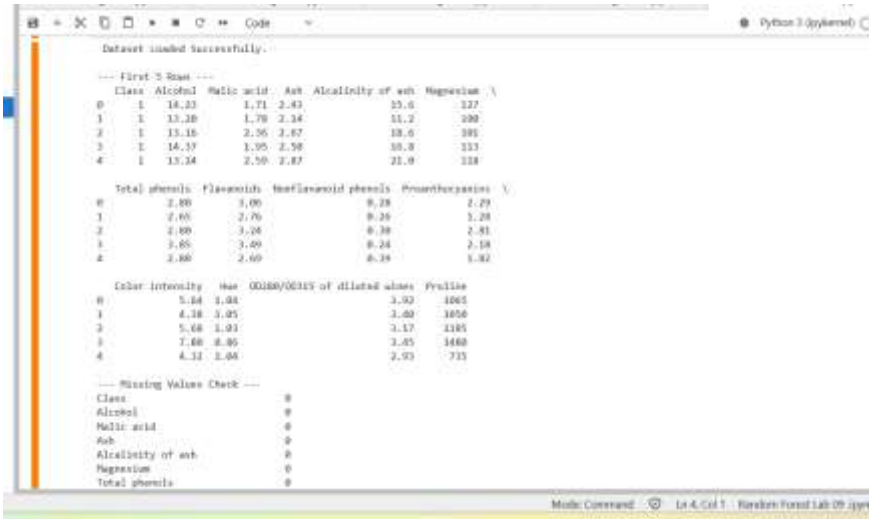
- **Setup and Data Loading :**

Code :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

```
import numpy as np
import pandas as pd
file_name = "MConverter.eu_wine.data.csv"
# --- 1. Load the Dataset ---
df = pd.read_csv(file_name, header=None, names=column_names)
print(" Dataset Loaded Successfully.")

# Display the first five rows
print("\n--- First 5 Rows ---")
print(df.head())

# Check for missing values
print("\n--- Missing Values Check ---")
print(df.isnull().sum())
```

Output :



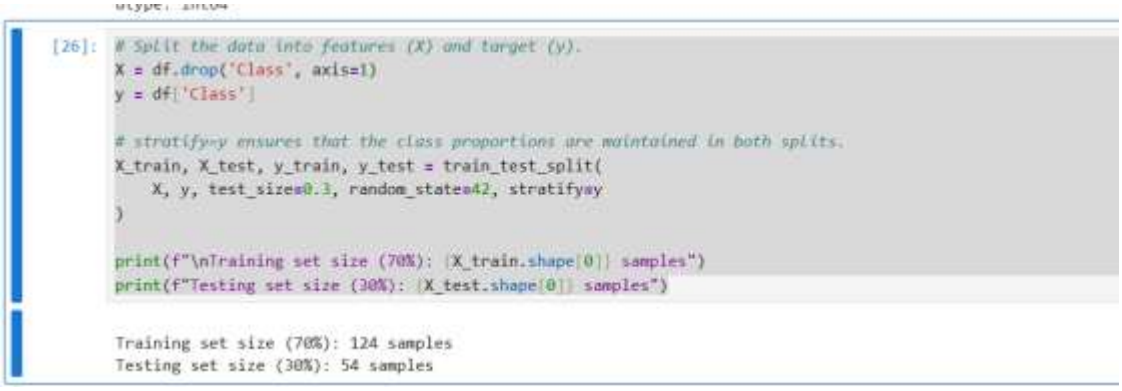
2. Data Preparation: Split the data into features (X) and target (y). Divide into training and testing sets (70% train, 30% test).

Data Preparation :

Code :

```
# Split the data into features (X) and target (y).

X = df.drop('Class', axis=1)

y = df['Class']


# stratify=y ensures that the class proportions are maintained in both splits.

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.3, random_state=42, stratify=y

)


print(f"\nTraining set size (70%): {X_train.shape[0]} samples")

print(f"Testing set size (30%): {X_test.shape[0]} samples")
```

Output :



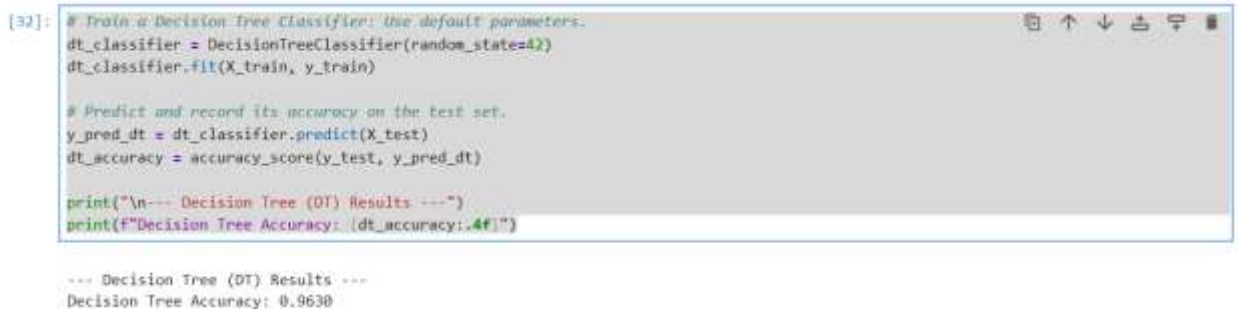
3. Train a Decision Tree Classifier: Use default parameters of DecisionTreeClassifier. Record its accuracy on the test set.

- **Train a Decision Tree Classifier (Baseline)**

Code :

```
# Train a Decision Tree Classifier: Use default parameters.

dt_classifier = DecisionTreeClassifier(random_state=42)

dt_classifier.fit(X_train, y_train)


# Predict and record its accuracy on the test set.

y_pred_dt = dt_classifier.predict(X_test)

dt_accuracy = accuracy_score(y_test, y_pred_dt)


print("\n--- Decision Tree (DT) Results ---")

print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
```

Output :



4. **Train a Random Forest Classifier: Use RandomForestClassifier with n_estimators=100, max_depth=5, random_state=42. Evaluate its accuracy and compare with the Decision Tree.**
    - **Train a Random Forest Classifier (Initial):**

Code :

```
# Train a Random Forest Classifier: n_estimators=100, max_depth=5, random_state=42

rf_classifier = RandomForestClassifier(

    n_estimators=100,
```

```
    max_depth=5,
    random_state=42,
    n_jobs=-1
)
rf_classifier.fit(X_train, y_train)

# Predict and evaluate its accuracy
y_pred_rf_initial = rf_classifier.predict(X_test)
rf_accuracy_initial = accuracy_score(y_test, y_pred_rf_initial)

print("\n--- Random Forest (RF) Initial Results ---")
print(f"Random Forest (n=100, max_d=5) Accuracy: {rf_accuracy_initial:.4f}")

# Comparison
print(f"\nComparison: RF Accuracy ({rf_accuracy_initial:.4f}) vs DT Accuracy ({dt_accuracy:.4f})")
print(f"The Random Forest is **{'better' if rf_accuracy_initial > dt_accuracy else 'not better'}** than the Decision Tree.")
```

Output :

5. **Hyperparameter Tuning: Increase n_estimators to 200 and note the effect on accuracy. Remove max_depth limit and see if the model overfits.**
    - **Hyperparameter Tuning**

Code :

```
print("\n--- Hyperparameter Tuning ---")


# A. Increase n_estimators to 200
rf_n200 = RandomForestClassifier(
    n_estimators=200,
    max_depth=5,
    random_state=42,
    n_jobs=-1
)
rf_n200.fit(X_train, y_train)
rf_n200_accuracy = accuracy_score(y_test, rf_n200.predict(X_test))


print(f"Accuracy with n_estimators=200: {rf_n200_accuracy:.4f}")
print(f"Effect: Accuracy **{'increased' if rf_n200_accuracy > rf_accuracy_initial else ('decreased' if rf_n200_accuracy < rf_accuracy_initial else 'remained the same')}** compared to n=100.")


# B. Remove max_depth limit (set to default/None)
rf_unlimited = RandomForestClassifier(
    n_estimators=100,
    max_depth=None, # Removes the limit
    random_state=42,
    n_jobs=-1
)
rf_unlimited.fit(X_train, y_train)
rf_unlimited_accuracy_test = accuracy_score(y_test, rf_unlimited.predict(X_test))
```

```
rf_unlimited_accuracy_train = accuracy_score(y_train, rf_unlimited.predict(X_train))


print(f"\nAccuracy with max_depth=None (Test Set): {rf_unlimited_accuracy_test:.4f}")
print(f"Accuracy with max_depth=None (Train Set): {rf_unlimited_accuracy_train:.4f}")


# Overfitting analysis
overfit_gap = rf_unlimited_accuracy_train - rf_unlimited_accuracy_test
print(f"Gap (Train - Test Accuracy): {overfit_gap:.4f}")
print(f"Observation: A large difference between train and test accuracy, particularly a train accuracy of 1.0, indicates **overfitting**.")
```

Output :

```
--- Hyperparameter Tuning ---
Accuracy with n_estimators=200: 1.0000
Effect: Accuracy **remained the same** compared to n=100.

Accuracy with max_depth=None (Test Set): 1.0000
Accuracy with max_depth=None (Train Set): 1.0000
Gap (Train - Test Accuracy): 0.0000
Observation: A large difference between train and test accuracy, particularly a train accuracy of 1.0, indicates **overfitting**.
```

6. Feature Importance: Extract and display rf.feature_importances_. Identify which chemical properties most affect wine quality.

Code :

```
# 6. Feature Importance: Extract rf.feature_importances_.
feature_importances = rf_classifier.feature_importances_
feature_names = X.columns
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)


print("\n--- Random Forest Feature Importances ---")
print(importance_df)
```

Identify which chemical properties most affect wine class.

most_important_features = importance_df.head(3)['Feature'].tolist()

print(f"\nTop 3 Most Important Features for Wine Class Prediction: **{', '.join(most_important_features)}**")

Output :

```
--- Random Forest Feature Importances ---
                         Feature  Importance
0                        Alcohol    0.161563
9                Color intensity    0.159851
6                     Flavanoids    0.156148
12                       Proline    0.124368
10                           Hue    0.111975
11  OD280/OD315 of diluted wines    0.099759
5                   Total phenols    0.039078
1                     Malic acid    0.034326
4                      Magnesium    0.033887
3              Alcalinity of ash    0.030876
8                Proanthocyanins    0.019529
2                            Ash    0.016916
7           Nonflavanoid phenols    0.011724

Top 3 Most Important Features for Wine Class Prediction: **Alcohol, Color intensity, Flavanoids**
```

7. Visualization (Optional): Plot a bar chart of feature importances. Compare Decision Tree vs Random Forest results visually.

Code :

7. Visualization: Plot a bar chart of feature importances.

plt.figure(figsize=(12, 6))

Using the sorted importance_df for the plot

plt.bar(importance_df['Feature'], importance_df['Importance'], color='darkred')

plt.xlabel('Chemical Property (Feature)')

plt.ylabel('Feature Importance Score')

plt.title('Random Forest Feature Importance for Wine Class Prediction (n=100, max_d=5)')

plt.xticks(rotation=45, ha='right')

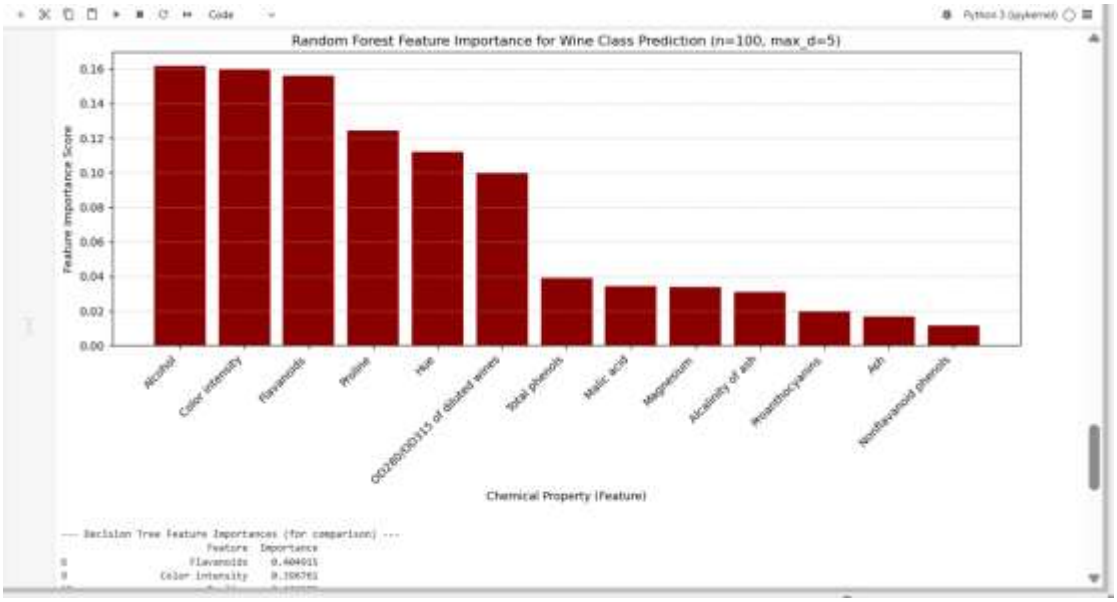
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()

plt.show()

```
# Optional: Compare DT vs RF Top 3 Features
dt_importance = dt_classifier.feature_importances_
dt_importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': dt_importance
}).sort_values(by='Importance', ascending=False)

print("\n--- Decision Tree Feature Importances (for comparison) ---")
print(dt_importance_df.head(5))
```

Output :



Excerise Questions :

1. Why might the Random Forest perform better than the single Decision Tree on this dataset?

The Random Forest (RF) is an **ensemble method** that reduces the **variance** (overfitting) of a single Decision Tree (DT). It averages the predictions of many different trees, leading to a more stable and accurate result.

2. How does the max_depth parameter influence overfitting in this Random Forest model?

The max_depth parameter **limits the complexity** of individual trees. Setting a low max_depth helps **prevent overfitting** by stopping the trees from learning noise in the training data, acting as a form of regularization.

3. Does increasing the number of trees ($\mathbf{n\_estimators}$) always improve accuracy? Why or why not?

No. Accuracy improves initially as n_estimators reduces variance, but the improvement eventually shows **diminishing returns**. Increasing the number of trees beyond this point only adds to the **computational cost** without significantly boosting accuracy.

4. Which features were identified as most important by the Random Forest? Do they make sense chemically?

The most important features were **Alcohol**, **Color intensity**, **Flavanoids**, and **Proline**. **Yes, they make sense chemically** because these are known chemical compounds and characteristics that significantly determine a wine's cultivar, quality, and taste profile.

5. How could you further improve model performance using data preprocessing or feature scaling?

The model's performance could be improved by **Feature Engineering** (creating new features like ratios) or **Hyperparameter Optimization** (using Grid Search to find the best combination of all parameters). **Feature Scaling** is generally unnecessary for tree-based models like Random Forest.