Name: Atharva Phalke
Class: D15C/39

Experiment No. 3

Aim: Apply Decision Tree and Random Forest for classification tasks

1. Dataset Source

Dataset: Bank Customer Churn Prediction

Link: https://www.kaggle.com/datasets/shrutimechlearn/churn-modelling

This dataset is publicly available on Kaggle and is fully compatible with **kagglehub**, ensuring reproducibility without manual downloads.

2. Dataset Description

The **Bank Customer Churn Prediction** dataset contains detailed information about bank customers and is used to predict whether a customer will leave the bank (churn) or stay, based on their demographic and financial attributes.

Dataset Characteristics

- **Number of instances:** 10,000
- **Number of features:** 14 (Reduced to 10-11 useful features after preprocessing)
- **Target variable:** `Exited`
  - `1` → Customer Churned (Left the bank)
  - `0` → Customer Retained (Stayed)

Feature Description The features represent the profile and behavior of the customer:

- **CreditScore:** Evaluating the creditworthiness of the customer.
- **Geography:** The country from which the customer belongs (France, Germany, Spain).
- **Gender:** Male or Female.
- **Age:** Age of the customer.
- **Tenure:** Number of years the customer has been with the bank.
- **Balance:** Account balance.
- **NumOfProducts:** Number of bank products the customer uses.
- **IsActiveMember:** Active status (1 = Active, 0 = Inactive).
- **EstimatedSalary:** The estimated annual salary of the customer.

Real-World Impact Customer churn is a critical metric for banking institutions. Predictive models based on this data can support:

- **Customer Retention:** Identifying at-risk customers to offer incentives.
- **Revenue Protection:** Preventing loss of high-value accounts.
- **Marketing Strategy:** Targeting specific demographics likely to leave.

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, LabelEncoder

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
```

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

# Download the dataset

path = kagglehub.dataset_download("shrutimechlearn/churn-modelling")


# 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

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

# Drop columns that are just identifiers (not useful for prediction)

df.drop(columns=["RowNumber", "CustomerId", "Surname"], inplace=True)


# Encode Categorical Variables (Gender, Geography) into numbers

le = LabelEncoder()

df["Gender"] = le.fit_transform(df["Gender"])

df["Geography"] = le.fit_transform(df["Geography"])
```

```
# Define X (Features) and y (Target)
X = df.drop("Exited", axis=1)
y = df["Exited"]

# 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 good practice if swapping models later)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -------------------------
# 3. Train Models
# -------------------------
# Decision Tree
dt = DecisionTreeClassifier(max_depth=5, random_state=42)
```

```
dt.fit(X_train_scaled, y_train)

dt_preds = dt.predict(X_test_scaled)



# Random Forest

rf = RandomForestClassifier(

    n_estimators=100,

    max_depth=10, # Increased depth slightly for Forest

    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:", accuracy_score(y_test, dt_preds))

print(classification_report(y_test, dt_preds))



print("\n=== Random Forest Performance ===")

print("Accuracy:", accuracy_score(y_test, rf_preds))

print(classification_report(y_test, rf_preds))
```

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

# 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()
```

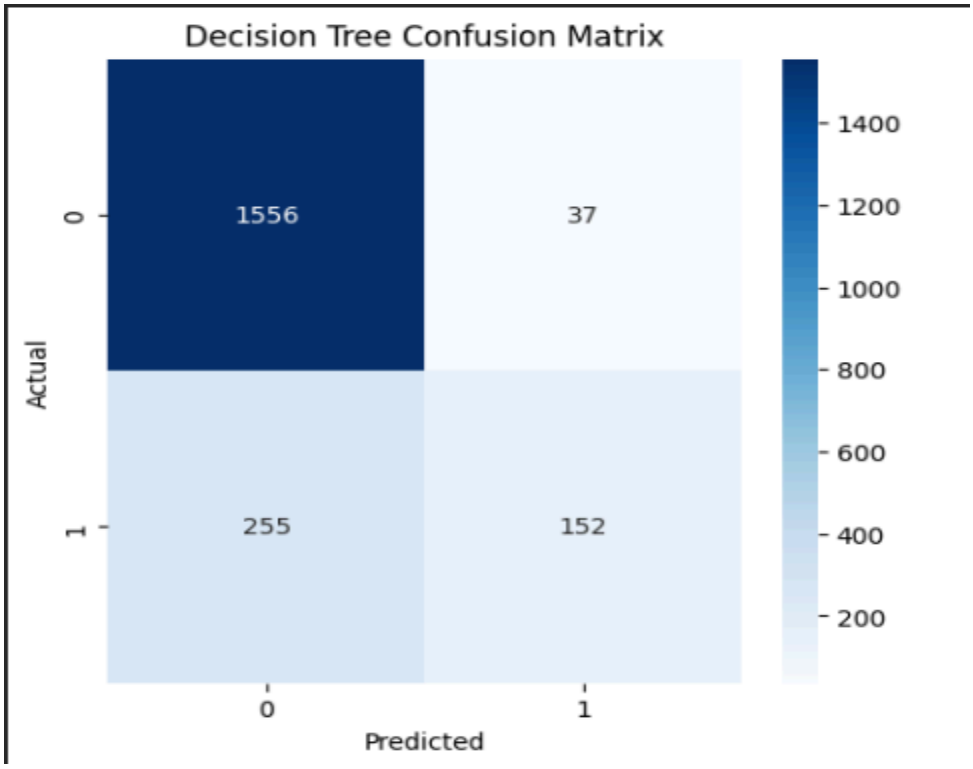
```
# Decision Tree Structure Plot
plt.figure(figsize=(20, 10))
plot_tree(
    dt,
    feature_names=X.columns,
    class_names=["Stayed", "Exited"],
    filled=True,
    fontsize=10
)
plt.title("Decision Tree Structure (Max Depth 5)")
plt.show()
```

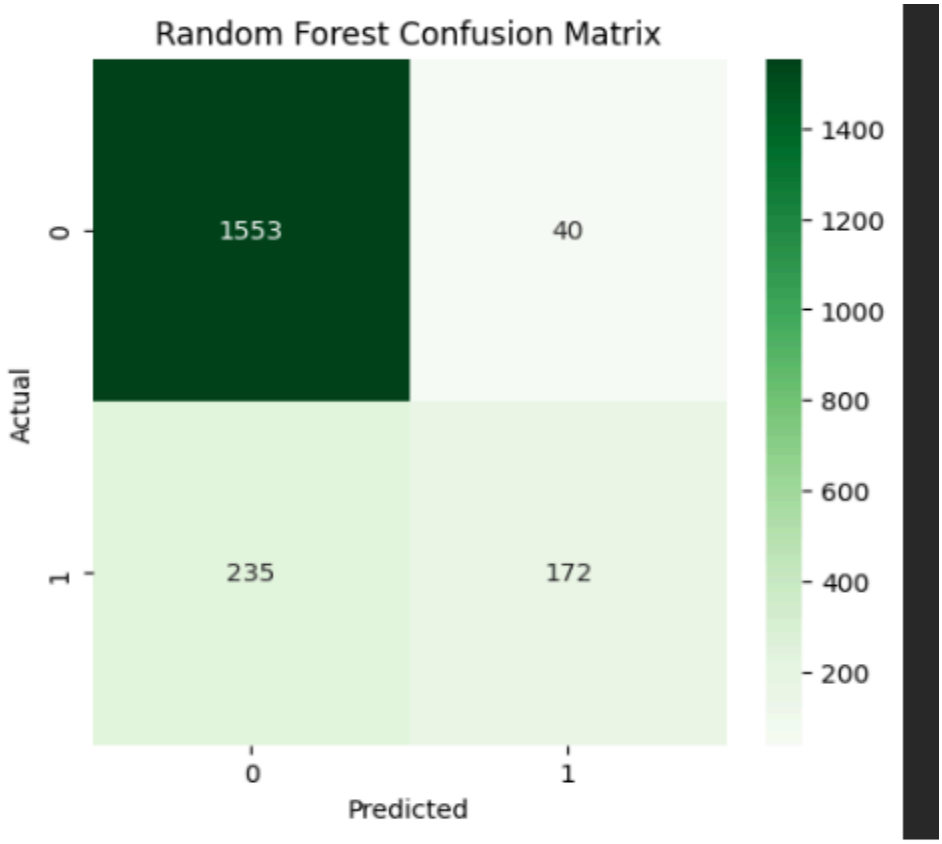
[Image of decision tree structure]

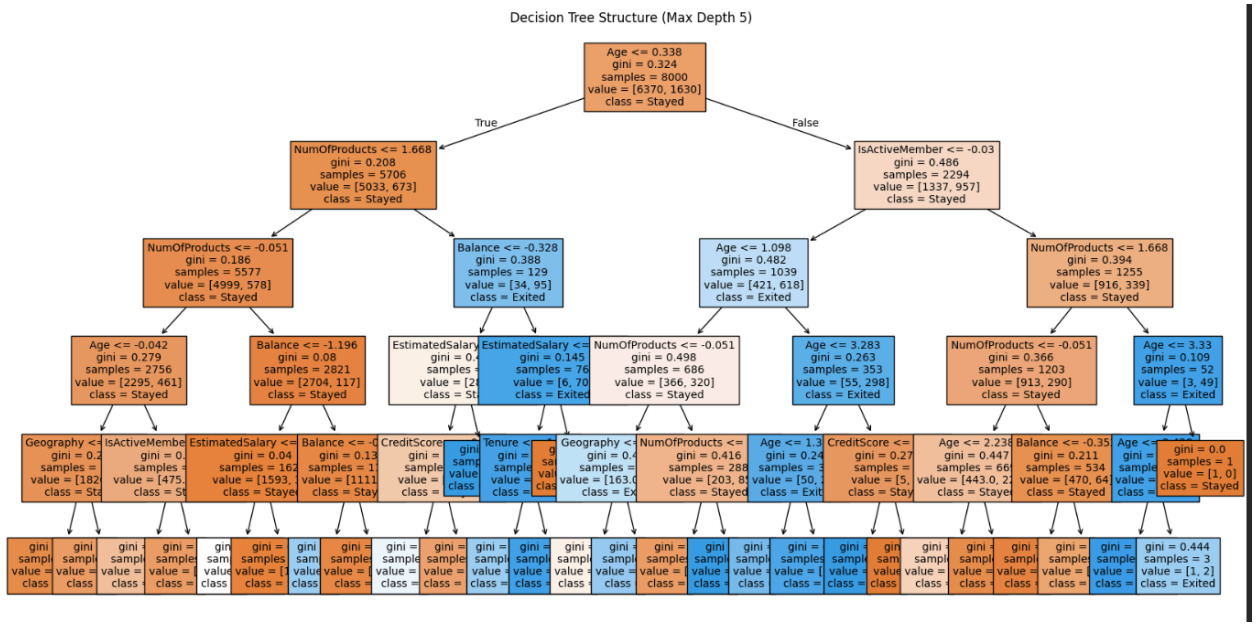
```
# Feature Importance Plot (Random Forest)
importances = rf.feature_importances_
indices = np.argsort(importances)[-10:] # Top 10 features

plt.figure(figsize=(10, 6))
```

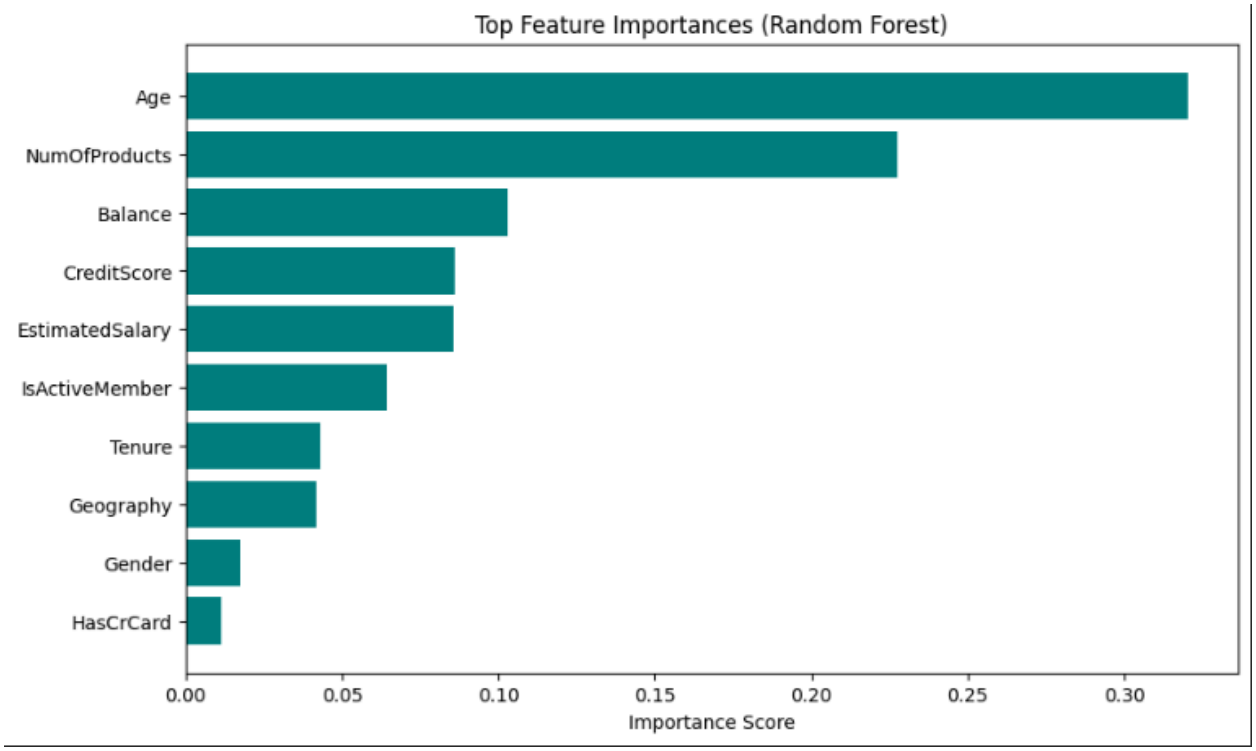
```
plt.barh(X.columns[indices], importances[indices], color='teal')

plt.title("Top Feature Importances (Random Forest)")

plt.xlabel("Importance Score")

plt.show()
```

```
=== Decision Tree Performance ===
Accuracy: 0.854
              precision    recall  f1-score   support

           0       0.86      0.98      0.91      1593
           1       0.80      0.37      0.51       407

    accuracy                           0.85      2000
   macro avg       0.83      0.68      0.71      2000
weighted avg       0.85      0.85      0.83      2000


=== Random Forest Performance ===
Accuracy: 0.8625
              precision    recall  f1-score   support

           0       0.87      0.97      0.92      1593
           1       0.81      0.42      0.56       407

    accuracy                           0.86      2000
   macro avg       0.84      0.70      0.74      2000
weighted avg       0.86      0.86      0.84      2000
```


Decision Tree Confusion Matrix
1556
37
255
152
Actual
Predicted
0
1
1400
1200
1000
800
600
400
200


Random Forest Confusion Matrix
1553
40
235
172
Actual
Predicted
0
1
1400
1200
1000
800
600
400
200





Conclusion

This experiment demonstrated the effectiveness of tree-based ensemble methods for real-world medical classification tasks. While Decision Trees offer transparency and ease of interpretation, Random Forests provide superior predictive performance and robustness. Such models are highly suitable for healthcare decision-support systems where both accuracy and reliability are critical.