Random Forest Implementation

Project Overview

This project demonstrates the implementation of a machine learning classification task using the Random Forest algorithm. The dataset used is "Social_Network_Ads.csv," sourced from Kaggle, which contains information about users' age, estimated salary, and whether they purchased a product advertised on a social network (binary outcome: 0 for not purchased, 1 for purchased). The goal is to predict whether a user will purchase the product based on their age and salary. The project compares the performance of a Decision Tree Classifier with a Random Forest Classifier, evaluates their effectiveness using confusion matrices and classification reports, and includes data exploration and visualization steps.

Dataset Description

- **Source**: Kaggle
- **Dataset**: Social_Network_Ads
- **Columns**:
  - Age: Age of the user (integer).
  - EstimatedSalary: Estimated annual salary of the user (integer).
  - Purchased: Binary target variable (0 = did not purchase, 1 = purchased).
- **Rows**: 400 entries.
- **Data Types**: All columns are integers (int64).
- **Missing Values**: None (all 400 entries are non-null).

The dataset is relatively small and clean, making it suitable for a straightforward classification task.

Code

1. Libraries

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

- **Key Module**: train_test_split is used to split the dataset into training and testing sets.

2. Loading the Dataset

```
df = pd.read_csv("C:/.../Social_Network_Ads.csv")
```

3. EDA

```
df.head()
```

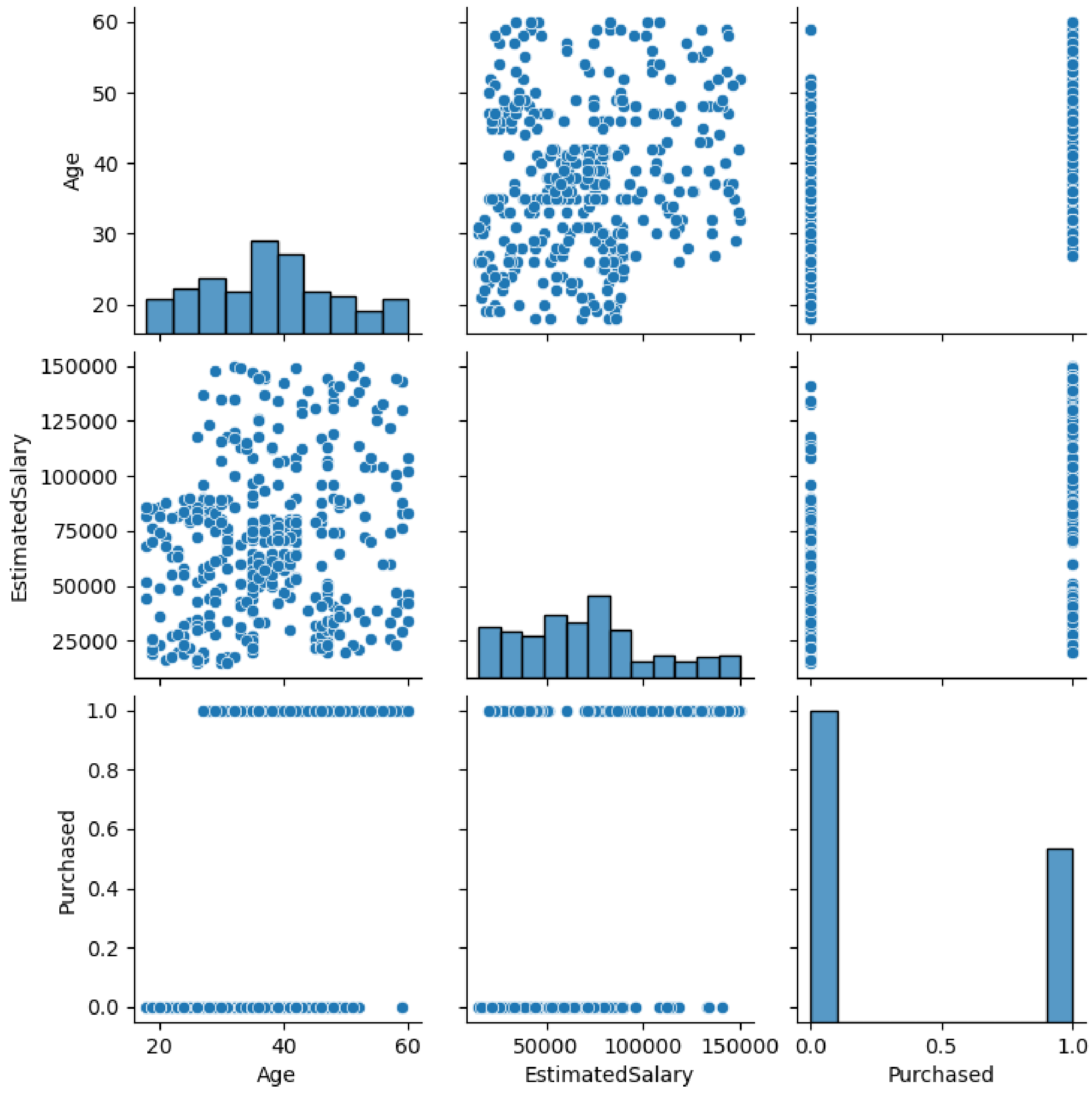
- **Observation**: The dataset included users of varying ages and salaries, with most in the sample not purchasing (Purchased = 0).

```
df.info()
```

- **Observation**: 400 rows, no missing values, and all columns are integers.

4. Data Visualization

```
sns.pairplot(df)
```



- **Output**: A pairplot (scatterplot matrix) showing relationships between Age, EstimatedSalary, and Purchased. The diagonal shows histograms for each variable, while off-diagonal plots are scatterplots colored by the Purchased variable.

- **Observation**:
  - Younger users with lower salaries tend not to purchase (Purchased = 0).
  - Older users with higher salaries are more likely to purchase (Purchased = 1).
  - There is some overlap, indicating the decision boundary may not be perfectly linear.

5. Preparing the Data

```
X = df.drop(\"Purchased\", axis=1)

y = df[\"Purchased\"]

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

- **Purpose**:
  - X: Features (Age, EstimatedSalary) are separated from the target variable.
  - y: Target variable (Purchased).
  - train_test_split: Splits the data into 80% training (320 samples) and 20% testing (80 samples) sets, with a fixed random_state for reproducibility.

6. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier()

decision_tree.fit(X_train, y_train)

tree_pred = decision_tree.predict(X_test)
```

- **Purpose**:
  - Train a Decision Tree Classifier on the training data.
  - Predict outcomes on the test set.

7. Evaluating Decision Tree Performance

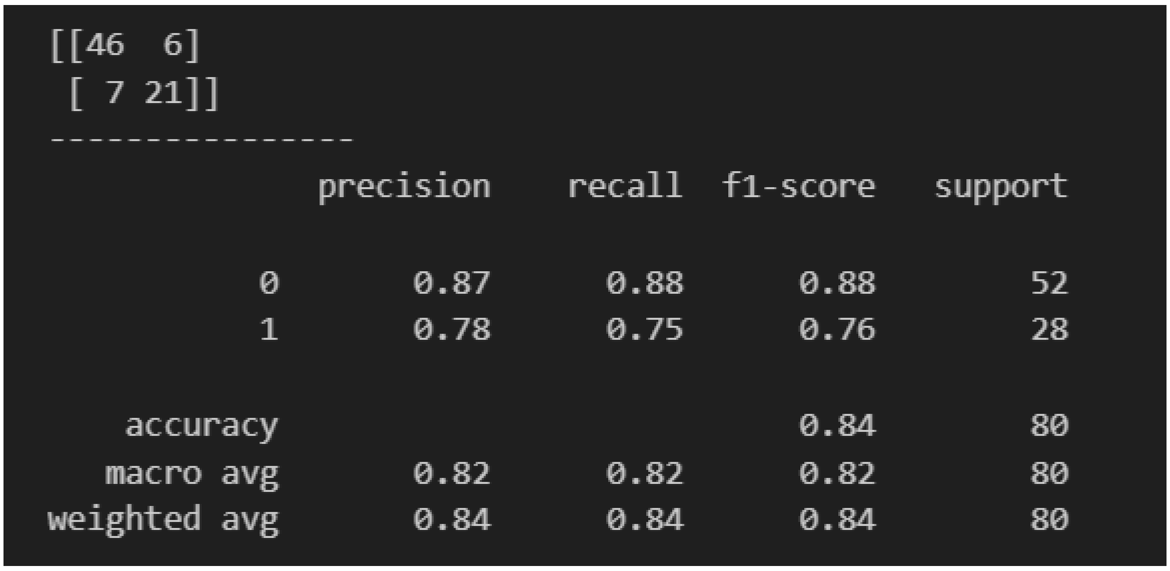
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, tree_pred))

print("----------------")

print(classification_report(y_test, tree_pred))
```

- **Output**:



```
[[46  6]
 [ 7 21]]
----------------
              precision    recall  f1-score   support

           0       0.87      0.88      0.88        52
           1       0.78      0.75      0.76        28

    accuracy                           0.84        80
   macro avg       0.82      0.82      0.82        80
weighted avg       0.84      0.84      0.84        80
```

  - **Confusion Matrix**:
    - True Negatives (TN): 46 (correctly predicted non-purchases).
    - False Positives (FP): 6 (non-purchases predicted as purchases).
    - False Negatives (FN): 7 (purchases predicted as non-purchases).
    - True Positives (TP): 21 (correctly predicted purchases).
  - **Metrics**:
    - Accuracy: 84%
    - Precision (class 0): 87%, (class 1): 78%.
    - Recall (class 0): 88%, (class 1): 75%.
    - F1-Score: Balanced measure of precision and recall, 0.88 for class 0 and 0.76 for class 1.
  - **Observation**: The model performs reasonably well but struggles slightly with class 1 (purchases), likely due to fewer positive samples.

8. Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier()

random_forest_model.fit(X_train, y_train)

random_preds = random_forest_model.predict(X_test)
```

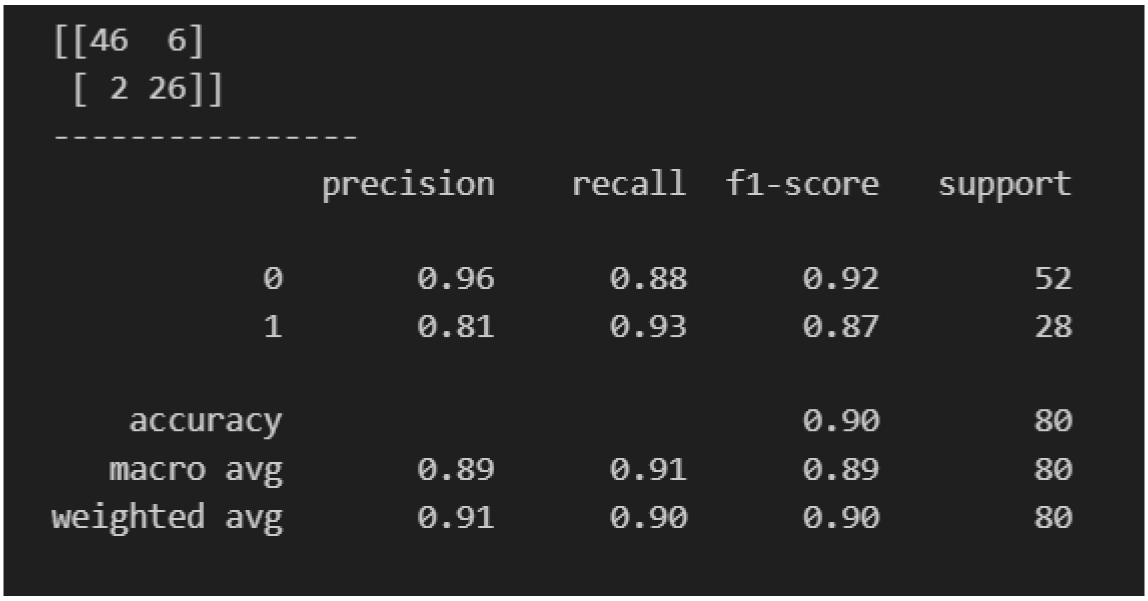
9. Evaluating Random Forest Performance

```
print(confusion_matrix(y_test, random_preds))

print("----------------")
```

print(classification_report(y_test, random_preds))

- **Output**:



```
[[46  6]
 [ 2 26]]
---------------
              precision    recall  f1-score   support

           0       0.96      0.88      0.92        52
           1       0.81      0.93      0.87        28

    accuracy                           0.90        80
   macro avg       0.89      0.91      0.89        80
weighted avg       0.91      0.90      0.90        80
```

  - **Confusion Matrix**:
    - TN: 46, FP: 6, FN: 2, TP: 26.
    - Fewer false negatives (2 vs. 7) compared to the Decision Tree.
  - **Metrics**:
    - Accuracy: 90%
    - Precision (class 0): 96%, (class 1): 81%.
    - Recall (class 0): 88%, (class 1): 93%.
    - F1-Score: 0.92 for class 0, 0.87 for class 1.
  - **Observation**: Random Forest outperforms the Decision Tree, especially in recall for class 1 (93% vs. 75%), indicating better detection of purchases.

Random Forest Algorithm

What is Random Forest?

Random Forest is an ensemble learning method used for classification and regression tasks. It operates by constructing multiple decision trees during the training phase and combining their outputs to improve the overall accuracy and stability of the model. The key idea is to reduce overfitting (a common issue with single decision trees) by averaging the predictions of many trees.

How It Works

1. **Bootstrap Sampling (Bagging)**:

   - Random Forest creates multiple subsets of the training data by sampling with replacement (bootstrap aggregating or "bagging"). Each subset is used to train a separate decision tree.
   - This introduces diversity among the trees, as each tree sees a slightly different version of the data.
2. **Feature Randomness**:
   - At each node of a decision tree, Random Forest randomly selects a subset of features to consider for splitting, rather than evaluating all features. This reduces correlation between trees and enhances generalization.
3. **Tree Construction**:
   - Each decision tree is grown to its maximum depth (or until a stopping criterion is met) without pruning, making them fully grown trees.
4. **Prediction**:
   - For classification: Each tree "votes" for a class, and the class with the majority vote across all trees is the final prediction.
   - For regression: The average of all tree predictions is taken.

Key Parameters (Default in this Project)

- n_estimators: Number of trees (default = 100).
- max_features: Number of features to consider at each split (default = "auto", typically sqrt(n_features)).
- random_state: Ensures reproducibility.

In this project, Random Forest leverages these properties to outperform the Decision Tree Classifier, as evidenced by higher accuracy and better recall for the minority class (Purchased = 1).

Results and Comparison

- **Decision Tree**:
   - Accuracy: 84%
   - Strengths: Simple, interpretable.
   - Weaknesses: Higher false negatives (7), indicating it misses some purchases.
- **Random Forest**:
   - Accuracy: 90%
   - Strengths: Higher accuracy, better recall for class 1 (93%), fewer false negatives (2).
   - Weaknesses: Slightly more complex and less interpretable.

- **Visualization Insight**: The pairplot suggests a non-linear decision boundary, which Random Forest captures better due to its ensemble nature.

Conclusion

The Random Forest model demonstrates superior performance (90% accuracy vs. 84%) due to its ability to handle the dataset's complexity and class imbalance more effectively. Potential improvements could include hyperparameter tuning (e.g., adjusting n_estimators or max_depth) or feature scaling, though the raw features worked well here.