Predictive Analysis with Decision Trees

Submitted by

SANMITRAA V R

727723EUCS207

CSE – ‘C’

III YEAR

COMPUTER SCIENCE AND ENGINEERING

1. Introduction

1.1 Predictive Analysis

Predictive analysis is the process of analyzing historical data to identify patterns and predict future outcomes. It uses machine learning techniques to support decision-making in areas such as finance, healthcare, and business.

1.2 Decision Trees

Decision Trees are supervised machine learning algorithms used for classification and regression. They represent decisions in a tree-like structure where nodes represent conditions based on features and leaf nodes represent predictions. Decision Trees are easy to understand and interpret, making them suitable for predictive analysis.

1.3 Overfitting and Pruning

One of the major challenges of Decision Trees is overfitting, where the model learns noise from training data and performs poorly on new data. Pruning techniques are used to control the growth of the tree by removing unnecessary branches, thereby improving the model's generalization performance.

2. Problem Statement

Decision Trees are widely used for classification problems due to their simplicity and interpretability. However, they often suffer from overfitting, which reduces their ability to generalize to unseen data. The objective of this project is to apply Decision Tree classification to **predict loan approval status** based on applicant details and to reduce overfitting by using appropriate pruning techniques for improved prediction accuracy.

3. Objectives

- To build a classification model using Decision Tree algorithms.
- To predict loan approval status based on applicant-related features.

- To understand and apply decision tree splitting criteria such as Gini index and entropy.
- To reduce overfitting by applying pruning techniques.
- To evaluate the performance of the model using appropriate metrics.

4. Dataset Description

The dataset used in this project is a loan approval dataset. It contains applicant-related information used to predict whether a loan will be approved or rejected. The dataset includes numerical and categorical features such as income, loan amount, credit history, and education. The target variable **Loan_Status** represents the loan approval decision.

5. Decision Tree Theory

A Decision Tree is a supervised learning algorithm used for classification and regression. It represents decisions in a tree structure where internal nodes denote decision conditions, branches represent outcomes, and leaf nodes represent final class labels. The tree is constructed by selecting features that best split the data at each node.

To choose the optimal split, impurity measures such as **Entropy** and **Gini Index** are used.

5.1 Entropy

Entropy measures the amount of randomness or impurity in a dataset. A dataset with mixed classes has higher entropy, while a pure dataset has lower entropy.

Formula:

$$Entropy(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Where:

- $p_i$is the probability of class i
- $n$is the number of classes

Lower entropy indicates better class separation.

5.2 Information Gain

Information Gain measures the reduction in entropy after a dataset is split on a feature. The feature with the highest information gain is selected for splitting.

Formula:

$$IG(S,A) = Entropy(S) - \sum_{v \in A} \frac{|\,S_v\,|}{|\,S\,|} Entropy(S_v)$$

Where:

- $S$is the original dataset
- $A$is the feature used for splitting
- $S_v$is the subset of data after the split

5.3 Gini Index

The Gini Index is another impurity measure commonly used in Decision Trees. It calculates the probability of incorrect classification of a randomly chosen element.

Formula:

$$Gini = 1 - \sum_{i=1}^{n} p_i^2$$

Lower Gini Index values indicate purer nodes and better splits.

6. Overfitting and Pruning

Overfitting occurs when a Decision Tree becomes too complex and learns noise from the training data, leading to poor performance on new data. To overcome this issue, pruning techniques are applied to reduce tree complexity.

Pre-pruning limits tree growth during training by restricting parameters such as tree depth. Post-pruning removes unnecessary branches after training. Both methods help improve model generalization and prediction accuracy.

7. Methodology

1. The loan approval dataset is collected from Kaggle.
2. Data preprocessing is performed to handle missing values and categorical attributes.
3. The dataset is divided into input features and the target variable.
4. A Decision Tree classification model is trained using the processed data.
5. Pruning techniques are applied to reduce overfitting.
6. The model performance is evaluated using accuracy metrics.
7. The Decision Tree and feature importance are visualized for interpretation.

8. Tools and Technologies

- **Programming Language:** Python
- **Libraries:**
  - Pandas – data handling and preprocessing
  - NumPy – numerical computations
  - Scikit-learn – decision tree model implementation
  - Matplotlib – data and model visualization
- **Development Environment:** Jupyter Notebook / VS Code

9. Results and Analysis

The Decision Tree classification model was evaluated using accuracy as the performance metric. The model showed improved performance after applying pruning techniques, indicating a reduction in overfitting. The comparison between the pruned and unpruned models demonstrates better generalization on unseen data. Feature importance analysis highlights the key attributes influencing loan approval decisions.

10. Conclusion

This project demonstrated the application of Decision Tree classification for predictive analysis in loan approval prediction. The model effectively classified loan applications based on applicant features. Pruning techniques helped reduce overfitting and improved the model’s generalization performance. Overall, the project highlights the simplicity, interpretability, and effectiveness of Decision Trees in solving real-world classification problems.

11. Future Enhancements

- The model performance can be improved by using ensemble methods such as Random Forests or Gradient Boosting.
- Hyperparameter tuning can be applied to further optimize the Decision Tree.
- The system can be tested with larger and more diverse datasets.
- Additional features can be included to improve prediction accuracy.

SOURCE CODE

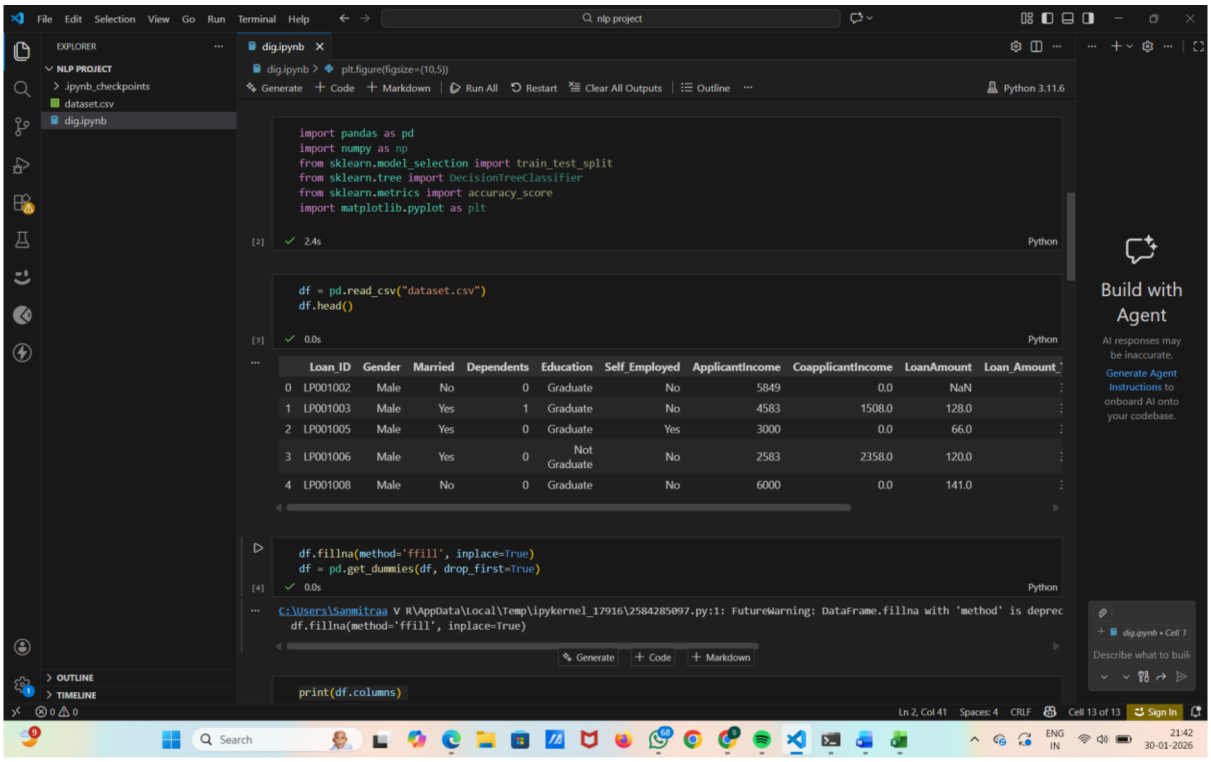
```
# Import required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier, plot_tree

from sklearn.metrics import accuracy_score


# Load the dataset

df = pd.read_csv("train.csv")


# Remove extra spaces in column names

df.columns = df.columns.str.strip()


# Convert categorical variables to numerical form

df = pd.get_dummies(df, drop_first=True)


# Separate features and target variable

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

y = df['Loan_Status_Y']


# Remove Loan ID related columns

X = X.loc[:, ~X.columns.str.startswith('Loan_ID')]


# Split the dataset into training and testing sets

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

# Train Decision Tree model without pruning
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

# Evaluate model before pruning
y_pred = dt.predict(X_test)
accuracy_before = accuracy_score(y_test, y_pred)
print("Accuracy without pruning:", accuracy_before)

# Train Decision Tree model with pre-pruning
dt_pruned = DecisionTreeClassifier(
    max_depth=4,
    min_samples_split=10,
    random_state=42
)
dt_pruned.fit(X_train, y_train)

# Evaluate model after pruning
y_pred_pruned = dt_pruned.predict(X_test)
accuracy_after = accuracy_score(y_test, y_pred_pruned)
```

```
print("Accuracy with pruning:", accuracy_after)


# Visualize the pruned Decision Tree
plt.figure(figsize=(20, 10))
plot_tree(
    dt_pruned,
    feature_names=X.columns,
    class_names=['Rejected', 'Approved'],
    filled=True
)
plt.title("Decision Tree with Pruning")
plt.show()


# Feature importance analysis
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': dt_pruned.feature_importances_
}).sort_values(by='Importance', ascending=False)


print("\nTop Important Features:")
print(feature_importance.head())


# Feature importance visualization
```

```
plt.figure(figsize=(10, 5))

plt.bar(feature_importance['Feature'], feature_importance['Importance'])

plt.xticks(rotation=90)

plt.title("Feature Importance")

plt.tight_layout()

plt.show()

print("\nProject Executed Successfully")
```

SCREENSHOTS


File Edit Selection View Go Run Terminal Help
nlp project
EXPLORER
NLP PROJECT
.ipynb_checkpoints
dataset.csv
dig.ipynb
dig.ipynb
dig.ipynb > plt.figure(figsize=(10,5))
Generate + Code + Markdown Run All Restart Clear All Outputs Outline
Python 3.11.6
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
OUTLINE
TIMELINE
Ln 2, Col 41 Spaces: 4 CRLF Cell 13 of 13 Sign In
Search
ENG IN
21:42 30-01-2026

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```
[2] ✓ 2.4s Python

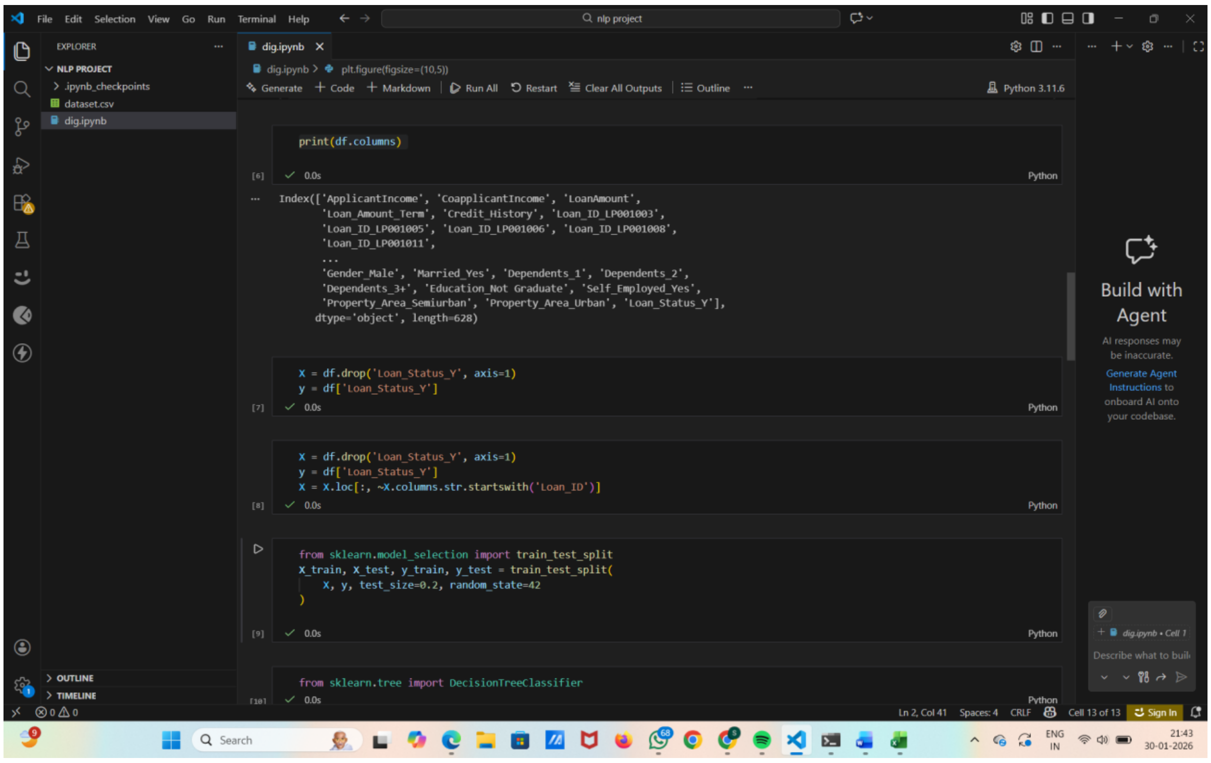
```
df = pd.read_csv("dataset.csv")
df.head()
```
[3] ✓ 0.0s Python

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

```
df.fillna(method='ffill', inplace=True)
df = pd.get_dummies(df, drop_first=True)
```
[4] ✓ 0.0s Python

```
C:\Users\Sanmitraa V R\AppData\Local\Temp\ipykernel_17916\2584285097.py:1: FutureWarning: DataFrame.fillna with 'method' is deprec
  df.fillna(method='ffill', inplace=True)
```

```
print(df.columns)
```



```
print(df.columns)
```

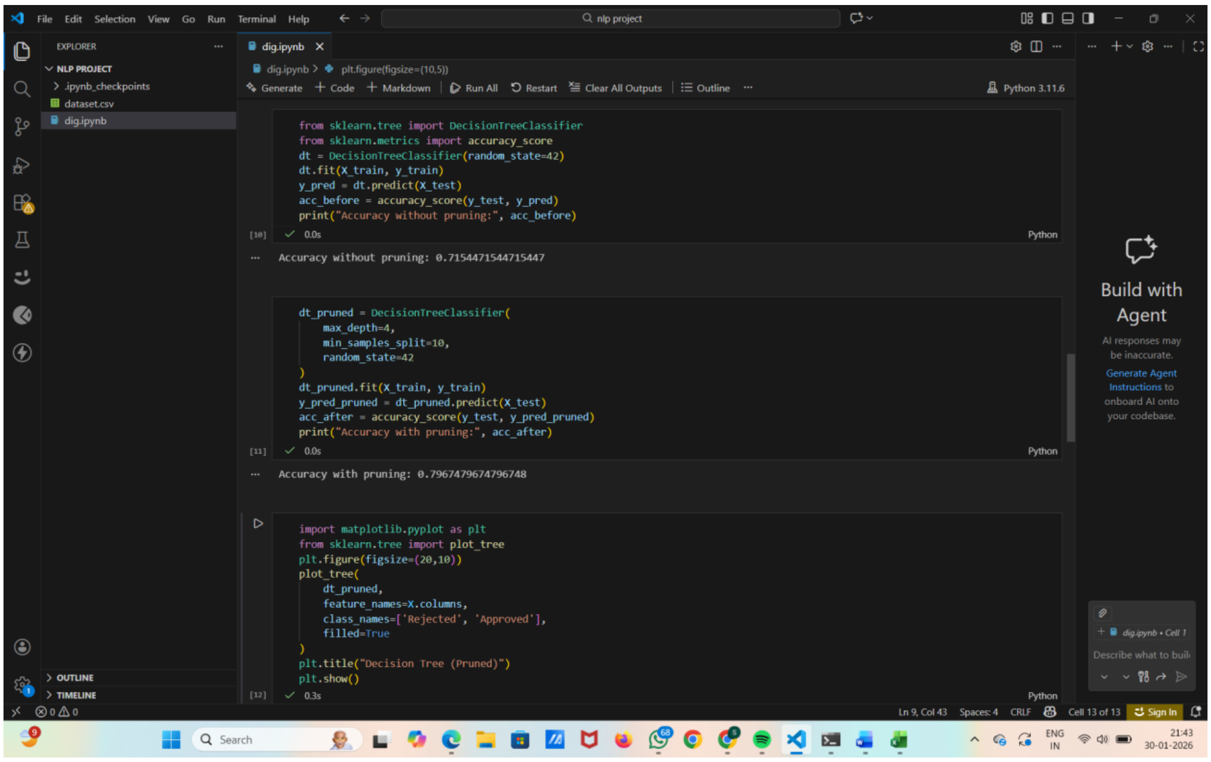
```
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Loan_ID_LP001003',
       'Loan_ID_LP001005', 'Loan_ID_LP001006', 'Loan_ID_LP001008',
       'Loan_ID_LP001011',
       ...
       'Gender_Male', 'Married_Yes', 'Dependents_1', 'Dependents_2',
       'Dependents_3+', 'Education_Not Graduate', 'Self_Employed_Yes',
       'Property_Area_Semiurban', 'Property_Area_Urban', 'Loan_Status_Y'],
      dtype='object', length=628)
```

```
X = df.drop('Loan_Status_Y', axis=1)
y = df['Loan_Status_Y']
```

```
X = df.drop('Loan_Status_Y', axis=1)
y = df['Loan_Status_Y']
X = X.loc[:, ~X.columns.str.startswith('Loan_ID')]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
from sklearn.tree import DecisionTreeClassifier
```



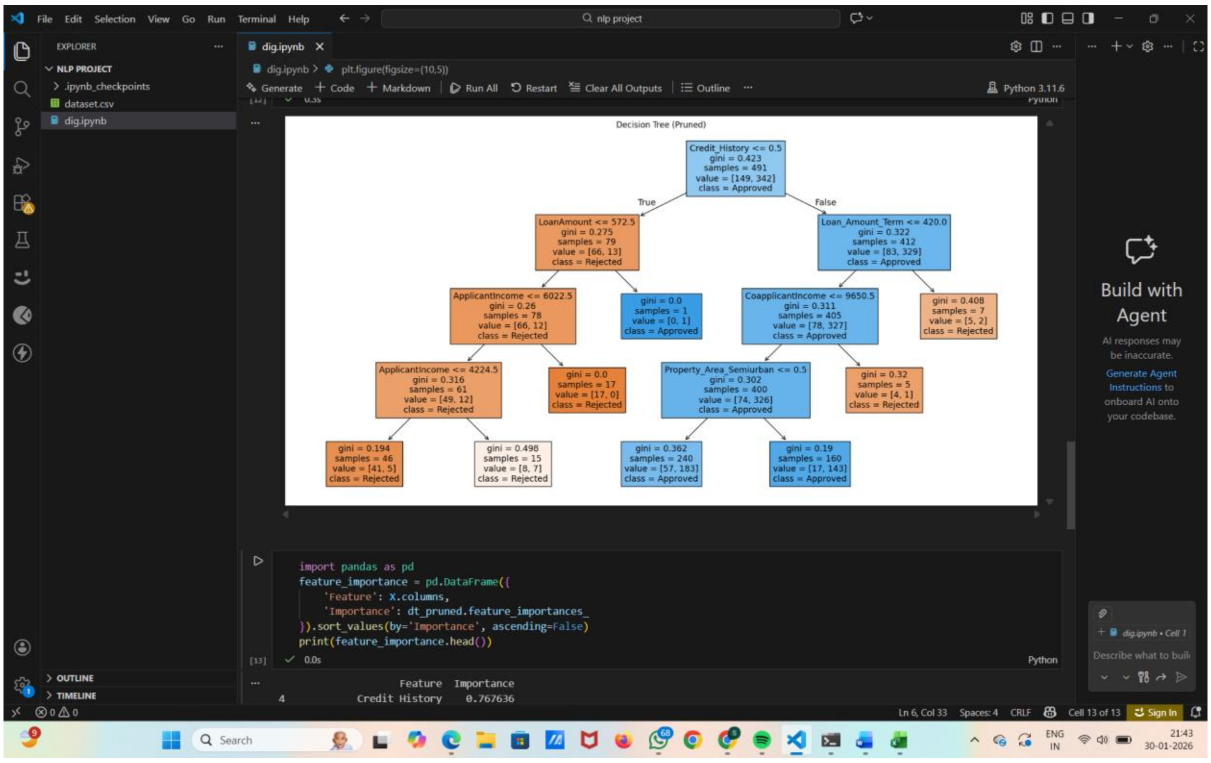
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred = dt.predict(X_test)
acc_before = accuracy_score(y_test, y_pred)
print("Accuracy without pruning:", acc_before)
```

```
Accuracy without pruning: 0.7154471544715447
```

```
dt_pruned = DecisionTreeClassifier(
    max_depth=4,
    min_samples_split=10,
    random_state=42
)
dt_pruned.fit(X_train, y_train)
y_pred_pruned = dt_pruned.predict(X_test)
acc_after = accuracy_score(y_test, y_pred_pruned)
print("Accuracy with pruning:", acc_after)
```

```
Accuracy with pruning: 0.7967479674796748
```

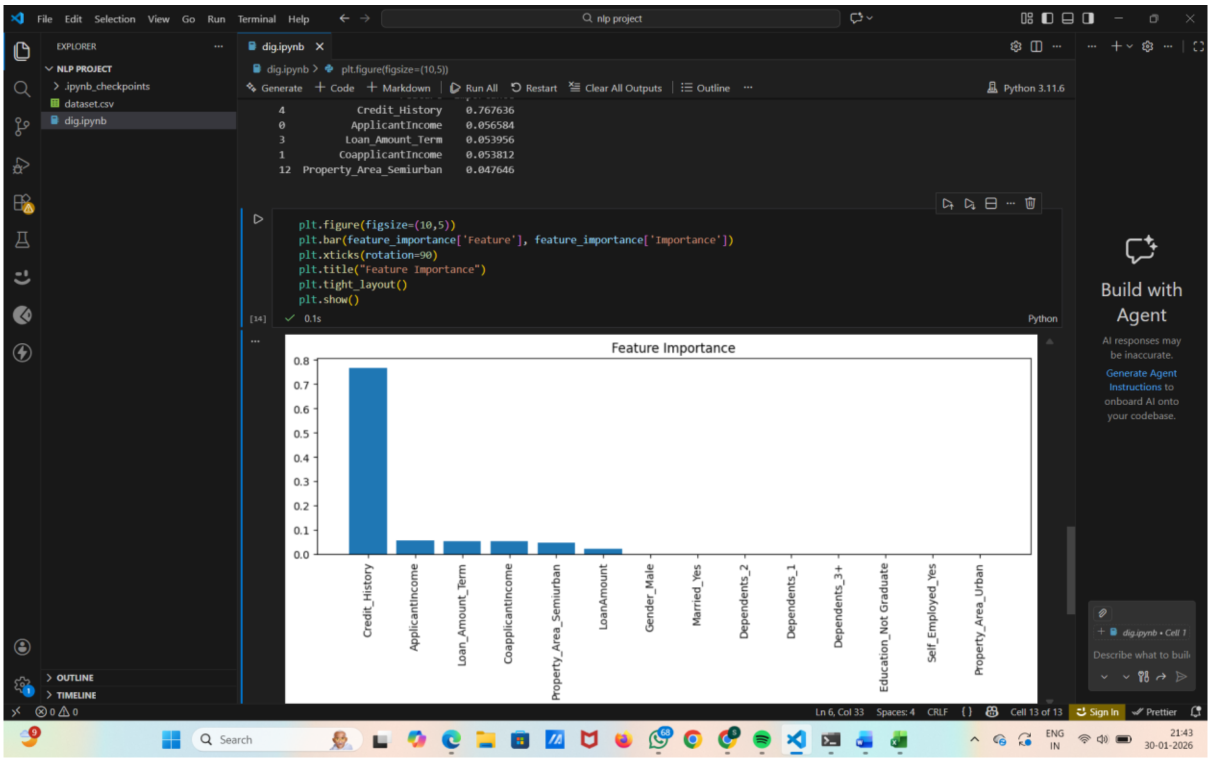
```
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(20,10))
plot_tree(
    dt_pruned,
    feature_names=X.columns,
    class_names=['Rejected', 'Approved'],
    filled=True
)
plt.title("Decision Tree (Pruned)")
plt.show()
```


Decision Tree (Pruned)
Credit_History <= 0.5 gini = 0.423 samples = 491 value = [149, 342] class = Approved
True
False
LoanAmount <= 572.5 gini = 0.275 samples = 79 value = [66, 13] class = Rejected
Loan_Amount_Term <= 420.0 gini = 0.322 samples = 412 value = [83, 329] class = Approved
ApplicantIncome <= 6022.5 gini = 0.26 samples = 78 value = [66, 12] class = Rejected
gini = 0.0 samples = 1 value = [0, 1] class = Approved
CoapplicantIncome <= 9650.5 gini = 0.311 samples = 405 value = [78, 327] class = Approved
gini = 0.408 samples = 7 value = [5, 2] class = Rejected
ApplicantIncome <= 4224.5 gini = 0.316 samples = 61 value = [49, 12] class = Rejected
gini = 0.0 samples = 17 value = [17, 0] class = Rejected
Property_Area_Semiurban <= 0.5 gini = 0.302 samples = 400 value = [74, 326] class = Approved
gini = 0.32 samples = 5 value = [4, 1] class = Rejected
gini = 0.194 samples = 46 value = [41, 5] class = Rejected
gini = 0.498 samples = 15 value = [8, 7] class = Rejected
gini = 0.362 samples = 240 value = [57, 183] class = Approved
gini = 0.19 samples = 160 value = [17, 143] class = Approved

```python
import pandas as pd
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': dt_pruned.feature_importances_
}).sort_values(by='Importance', ascending=False)
print(feature_importance.head())
```

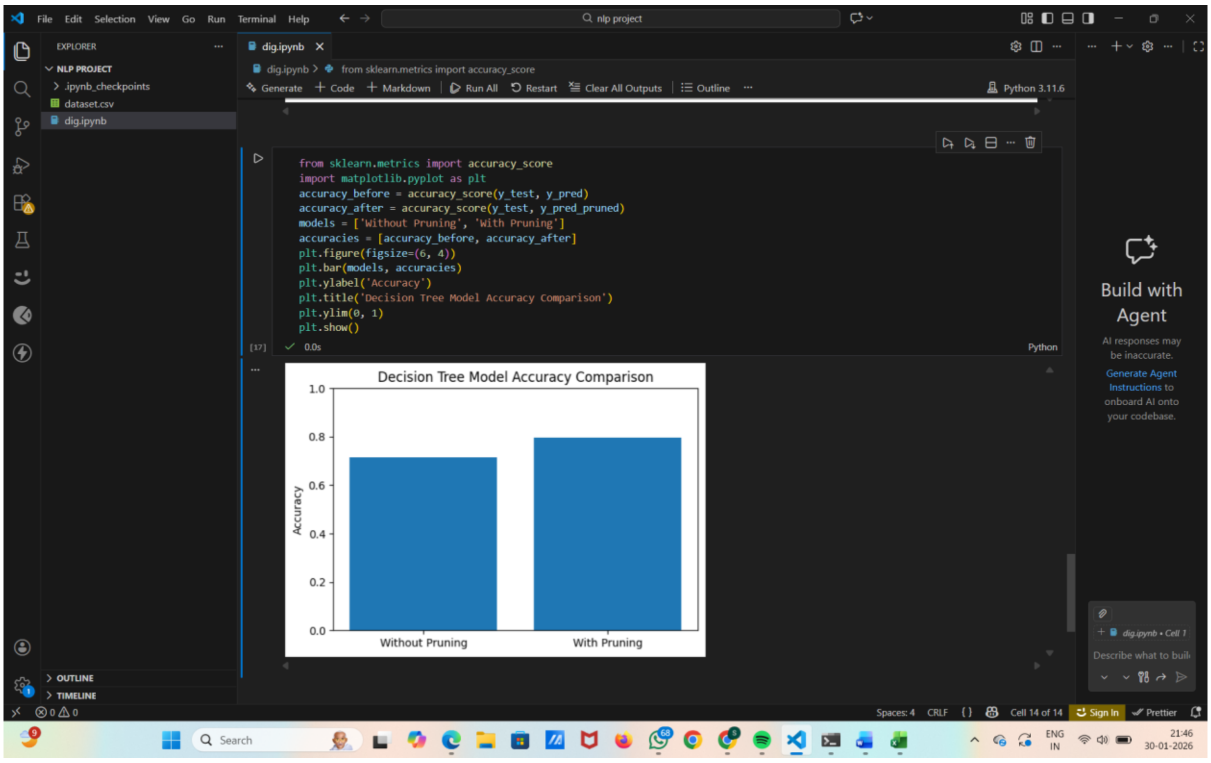
[13] ✓ 0.0s

```
                  Feature  Importance
4          Credit History    0.767636
```


```
4          Credit_History    0.767636
0         ApplicantIncome    0.056584
3        Loan_Amount_Term    0.053956
1       CoapplicantIncome    0.053812
12  Property_Area_Semiurban    0.047646
```

```python
plt.figure(figsize=(10,5))
plt.bar(feature_importance['Feature'], feature_importance['Importance'])
plt.xticks(rotation=90)
plt.title("Feature Importance")
plt.tight_layout()
plt.show()
```

[14] ✓ 0.1s



```
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
accuracy_before = accuracy_score(y_test, y_pred)
accuracy_after = accuracy_score(y_test, y_pred_pruned)
models = ['Without Pruning', 'With Pruning']
accuracies = [accuracy_before, accuracy_after]
plt.figure(figsize=(6, 4))
plt.bar(models, accuracies)
plt.ylabel('Accuracy')
plt.title('Decision Tree Model Accuracy Comparison')
plt.ylim(0, 1)
plt.show()
```

[17] ✓ 0.0s