```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# Define the path to the dataset
# Assuming 'bank-full.csv' is uploaded to your Colab environment or
current directory
PATH = './bank.csv'

# Load the data. The separator is often a semicolon ';' in this
specific UCI dataset.
try:
    df = pd.read_csv(PATH, sep=';')
    if 'y' not in df.columns and len(df.columns) == 1:
        print("Semicolon separator failed. Assuming comma separator.")
        df = pd.read_csv(PATH, sep=',')
    print("Dataset loaded successfully.")
except FileNotFoundError:
    print(f"Error: File not found at {PATH}. Please upload 'bank-
full.csv' and verify the path.")
    raise

# Inspect the initial data structure
print("\nInitial Data Structure:")
print(df.head())
print(df.info())
```

```
Semicolon separator failed. Assuming comma separator.
Dataset loaded successfully.

Initial Data Structure:
   age         job  marital  education default  balance housing loan
contact  \
0   59      admin.  married  secondary      no     2343     yes   no
unknown
1   56      admin.  married  secondary      no       45      no   no
unknown
2   41  technician  married  secondary      no     1270     yes   no
unknown
3   55    services  married  secondary      no     2476     yes   no
unknown
```

```
4   54       admin.  married   tertiary      no       184       no   no
unknown

   day month  duration  campaign  pdays  previous poutcome deposit
0    5   may      1042         1     -1         0  unknown     yes
1    5   may      1467         1     -1         0  unknown     yes
2    5   may      1389         1     -1         0  unknown     yes
3    5   may       579         1     -1         0  unknown     yes
4    5   may       673         2     -1         0  unknown     yes
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11162 entries, 0 to 11161
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        11162 non-null  int64 
 1   job        11162 non-null  object
 2   marital    11162 non-null  object
 3   education  11162 non-null  object
 4   default    11162 non-null  object
 5   balance    11162 non-null  int64 
 6   housing    11162 non-null  object
 7   loan       11162 non-null  object
 8   contact    11162 non-null  object
 9   day        11162 non-null  int64 
 10  month      11162 non-null  object
 11  duration   11162 non-null  int64 
 12  campaign   11162 non-null  int64 
 13  pdays      11162 non-null  int64 
 14  previous   11162 non-null  int64 
 15  poutcome   11162 non-null  object
 16  deposit    11162 non-null  object
dtypes: int64(7), object(10)
memory usage: 1.4+ MB
None
```

Data PreProcessing

```
X = df.drop('deposit', axis=1)
y = df['deposit'].map({'yes': 1, 'no': 0})

numerical_cols=X.select_dtypes(include=np.number).columns.tolist()
categorical_cols=X.select_dtypes(include='object').columns.tolist()
print("\nCategorical columns to be encoded:", categorical_cols)

numerical_transformer=StandardScaler()
categorical_tansformer=OneHotEncoder(handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_tansformer, categorical_cols)
    ],
    remainder='passthrough' # Keep any other columns if they exist
(not applicable here)
)
```

```
Categorical columns to be encoded: ['job', 'marital', 'education',
'default', 'housing', 'loan', 'contact', 'month', 'poutcome']
```

Data Splitting

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42, stratify=y)

print(f"\nTraining set size: {X_train.shape[0]} samples")
print(f"Testing set size: {X_test.shape[0]} samples")
```

```
Training set size: 8371 samples
Testing set size: 2791 samples
```

Model Training

```
models = {
    'Baseline Model (Default)':
DecisionTreeClassifier(random_state=42),
    'Pruned Model (Max Depth 4)': DecisionTreeClassifier(max_depth=4,
random_state=42),
    'Robust Model (Min Leaf 20)':
DecisionTreeClassifier(min_samples_leaf=20, random_state=42)
}

results = {}

print("\n" + "="*50)
print("STARTING MODEL TRAINING & EVALUATION")
print("="*50)

for name, model in models.items():

    # --- Create Full Pipeline ---
    # Combine preprocessing steps and the model into a single pipeline
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('classifier', model)])

    # --- Training ---
    print(f"\nTraining {name}...")
    pipeline.fit(X_train, y_train)
```

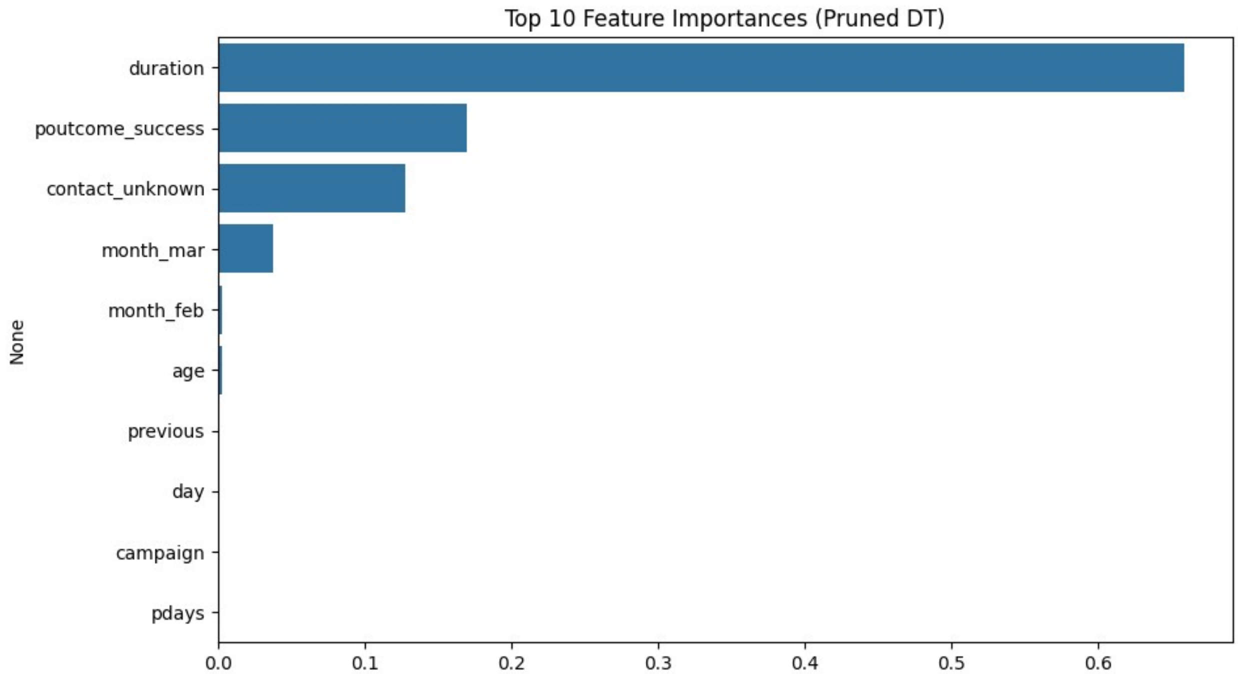
```python
    # --- Prediction ---
    y_pred = pipeline.predict(X_test)

    # --- Evaluation ---
    accuracy = accuracy_score(y_test, y_pred)

    # Use cross-validation for a robust measure of performance
    cv_scores = cross_val_score(pipeline, X, y, cv=5,
scoring='accuracy')

    results[name] = {
        'accuracy': accuracy,
        'cv_mean': cv_scores.mean(),
        'cv_std': cv_scores.std(),
        'model': pipeline,
        'y_pred': y_pred
    }

    # --- Print Results ---
    print(f"-> {name} Test Accuracy: {accuracy:.4f}")
    print(f"-> {name} 5-Fold CV Mean Accuracy: {cv_scores.mean():.4f}
(+/- {cv_scores.std():.4f})")
    print("-" * 50)
```

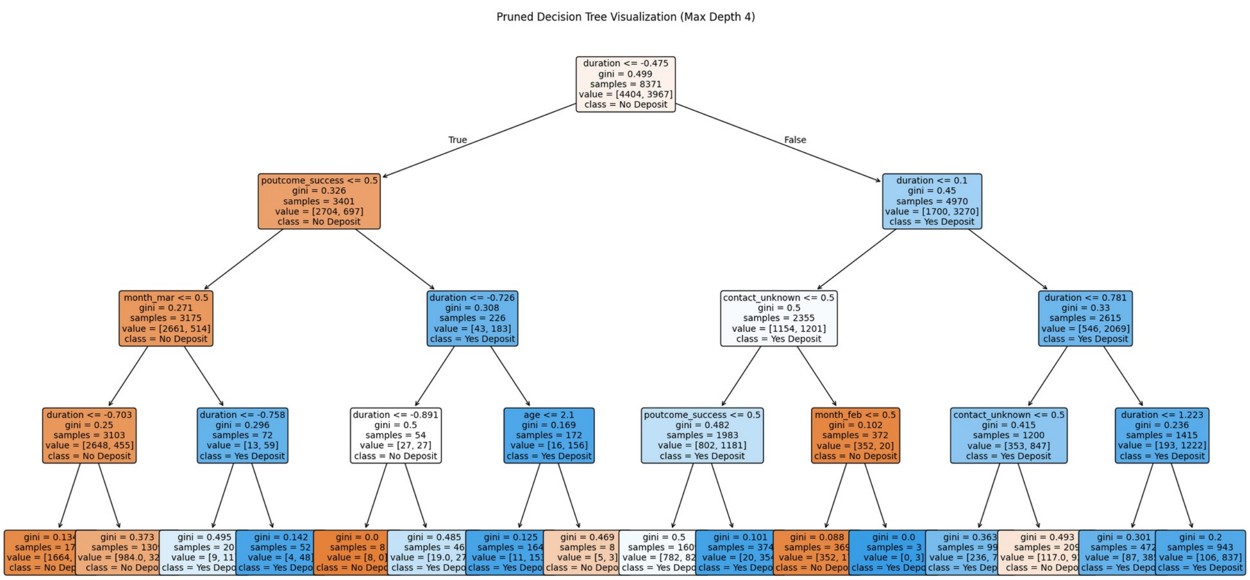
```
==================================================
STARTING MODEL TRAINING & EVALUATION
==================================================

Training Baseline Model (Default)...
-> Baseline Model (Default) Test Accuracy: 0.7861
-> Baseline Model (Default) 5-Fold CV Mean Accuracy: 0.7479 (+/-
0.0293)
--------------------------------------------------

Training Pruned Model (Max Depth 4)...
-> Pruned Model (Max Depth 4) Test Accuracy: 0.7825
-> Pruned Model (Max Depth 4) 5-Fold CV Mean Accuracy: 0.7463 (+/-
0.0280)
--------------------------------------------------

Training Robust Model (Min Leaf 20)...
-> Robust Model (Min Leaf 20) Test Accuracy: 0.8158
-> Robust Model (Min Leaf 20) 5-Fold CV Mean Accuracy: 0.7665 (+/-
0.0479)
--------------------------------------------------
```

result

```python
print("\n" + "="*50)
print("INTERPRETATION OF THE PRUNED MODEL (max_depth=4)")
print("="*50)

# 1. Extract the trained Decision Tree Classifier from the pipeline
pruned_pipeline = results['Pruned Model (Max Depth 4)']['model']
dt_classifier = pruned_pipeline.named_steps['classifier']

# 2. Extract feature names after One-Hot Encoding and Scaling
# Get the feature names from the preprocessor's categorical
transformer
ohe_feature_names =
list(pruned_pipeline.named_steps['preprocessor'].named_transformers_['
cat'].get_feature_names_out(categorical_cols))
all_feature_names = numerical_cols + ohe_feature_names

# 3. Plot Feature Importance
feature_importances = pd.Series(dt_classifier.feature_importances_,
index=all_feature_names).sort_values(ascending=False).head(10)

plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importances.values, y=feature_importances.index)
plt.title('Top 10 Feature Importances (Pruned DT)')
plt.show()


# 4. Visualize the Decision Tree (Highly Interpretive)
plt.figure(figsize=(25, 12))
plot_tree(dt_classifier,
          feature_names=all_feature_names,
          class_names=['No Deposit', 'Yes Deposit'],
          filled=True,
          rounded=True,
          fontsize=10)
plt.title("Pruned Decision Tree Visualization (Max Depth 4)")
plt.show()
# 5. Final Classification Report for the Best Model (e.g., Pruned)
print("\nClassification Report for Pruned Model:")
print(classification_report(y_test, results['Pruned Model (Max Depth
4)']['y_pred']))
```

```
==================================================
INTERPRETATION OF THE PRUNED MODEL (max_depth=4)
==================================================
```






```
Classification Report for Pruned Model:
              precision    recall  f1-score   support

           0       0.85      0.72      0.78      1469
           1       0.73      0.86      0.79      1322

    accuracy                           0.78      2791
   macro avg       0.79      0.79      0.78      2791
```

```
weighted avg       0.79      0.78      0.78      2791
```