```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report
)

# Load dataset
df = pd.read_csv('/kaggle/input/bank-marketing/bank.csv')
print("Dataset shape:", df.shape)
df.head()
```

```
Dataset shape: (11162, 17)
   age  job  marital  education  default  housing  loan  contact  month  day_of_week  campaign  pdays  previous  poutcome  emp.var.rate  cons.price.idx  cons.conf.idx  euribor3m  n
0   59  admin.  married  secondary       no      yes    no  unknown    may            5         1    146         0   unknown           1.1          93.994          -36.4      4.855
1   56  admin.  married   primary       no       no    no  unknown    may            5         1    148         0   unknown           1.1          93.994          -36.4      4.855
2   41  unknown  married  unknown       no       no    no  unknown    may            5         1    226         0   unknown           1.1          93.994          -36.4      4.855
3   26  services  married  secondary       no       no    no   cellular    may            5         1    226         0   unknown           1.1          93.994          -36.4      4.855
4   25  services   single   primary       no       no    no   cellular    may            5         1    307         1   unknown           1.1          93.994          -36.4      4.855

5 rows × 17 columns
```

```
# Check for missing values
print("Missing values:")
df.isnull().sum()
```

```
Missing values:
age             0
job             0
marital         0
education       0
default         0
balance         0
housing         0
loan            0
contact         0
day_of_week     0
duration        0
campaign        0
pdays           0
previous        0
poutcome        0
deposit         0
dtype: int64
```

```
# Check data types
df.info()
```

```
<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_of_week  11162 non-null  int64
 10  duration     11162 non-null  int64
 11  campaign     11162 non-null  int64
 12  pdays        11162 non-null  int64
 13  previous     11162 non-null  int64
 14  poutcome     11162 non-null  object
 15  deposit      11162 non-null  object
dtype: int64, float64: 0, int64: 7, object: 10
memory usage: 1.3 MB
```

```
# Check for duplicate rows
print("Duplicate rows:", df.duplicated().sum())
```

```
Duplicate rows: 0
```

```
# Check unique values in categorical columns
for col in df.select_dtypes(include=['object']).columns:
    print(f"{col}: {df[col].unique()}")
```

```
job: ['admin.' 'services' 'management' 'unknown' 'retired' 'blue-collar' 'self-employed' 'entrepreneur' 'technician' 'student' 'housemaid']
marital: ['married' 'single' 'divorced']
education: ['secondary' 'primary' 'unknown' 'tertiary']
default: ['no' 'yes']
housing: ['yes' 'no']
loan: ['no' 'yes']
contact: ['unknown' 'cellular' 'telephone']
month: ['may' 'jun' 'aug' 'oct' 'nov' 'jul' 'dec' 'mar' 'apr' 'sep']
poutcome: ['unknown' 'failure' 'other' 'success']
deposit: ['yes' 'no']
```

```
# Create a copy for processing
df_clean = df.copy()
print("Data cleaned successfully!")
```

```
Data cleaned successfully!
```

```
# Encode categorical variables
from sklearn.preprocessing import LabelEncoder

categorical_columns = df_clean.select_dtypes(include=['object'])
for col in categorical_columns:
    le = LabelEncoder()
    df_clean[col] = le.fit_transform(df_clean[col])
    print(f"{col}: {le.classes_}")

print("Encoding completed!")
print("Shape after encoding:", df_clean.shape)
```

```
job: ['admin.' 'services' 'management' 'unknown' 'retired' 'blue-collar' 'self-employed' 'entrepreneur' 'technician' 'student' 'housemaid']
marital: ['married' 'single' 'divorced']
education: ['secondary' 'primary' 'unknown' 'tertiary']
default: ['no' 'yes']
housing: ['yes' 'no']
loan: ['no' 'yes']
contact: ['unknown' 'cellular' 'telephone']
month: ['may' 'jun' 'aug' 'oct' 'nov' 'jul' 'dec' 'mar' 'apr' 'sep']
poutcome: ['unknown' 'failure' 'other' 'success']
deposit: ['yes' 'no']
Encoding completed!
Shape after encoding: (11162, 17)
```

```
# Define features and target
X = df_clean.drop('deposit', axis=1)
y = df_clean['deposit']

print("Features shape:", X.shape)
print("Target shape:", y.shape)
print("Target distribution:")
y.value_counts()
```

```
Features shape: (11162, 16)
Target shape: (11162,)
Target distribution:
deposit
1    5873
0    5289
Name: count, dtype: int64
```

```
# Split data (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42, stratify=y)

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
print("Training target distribution:")
y_train.value_counts()
print("Test target distribution:")
y_test.value_counts()
```

```
Training set shape: (8929, 16)
Test set shape: (2233, 16)
Training target distribution:
deposit
1    4698
0    4231
Name: count, dtype: int64
Test target distribution:
deposit
1    1175
0    1058
Name: count, dtype: int64
```

```
# Create decision tree classifier
dt = DecisionTreeClassifier(random_state=42)

# Train the model
dt.fit(X_train, y_train)

print("Decision tree classifier created and trained!")
print("Tree depth:", dt.tree_.max_depth)
print("Number of leaves:", dt.tree_.n_leaves)
```

```
Decision tree classifier created and trained!
Tree depth: 30
Number of leaves: 1000
```

```
# Make predictions
y_pred = dt.predict(X_test)

print("Predictions made!")
print("Sample predictions:", y_pred[:10])
```

```
Predictions made!
Sample predictions: [1 1 1 1 1 1 1 1 1 1]
```

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Calculate precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Calculate recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)

# Calculate F1-score
f1 = f1_score(y_test, y_pred)
print("F1-score:", f1)
```

```
Accuracy: 0.7163
Precision: 0.7163
Recall: 1.0
F1-score: 0.8349
```

```
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(cm)

# Classification report
print("Classification report:")
print(classification_report(y_test, y_pred))
```

```
Confusion matrix:
[[   0 1058]
[   0 1175]]
Classification report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1058
           1       0.72      1.00      0.83      1175

    accuracy                           0.72      2233
   macro avg       0.36      0.50      0.42      2233
weighted avg       0.51      0.72      0.60      2233
```

```
# Create decision tree classifier with limited depth
dt_limited = DecisionTreeClassifier(max_depth=5, random_state=42)

# Train the model
dt_limited.fit(X_train, y_train)

print("Limited decision tree classifier created and trained!")
print("Tree depth:", dt_limited.tree_.max_depth)
print("Number of leaves:", dt_limited.tree_.n_leaves)
```

```
Limited decision tree classifier created and trained!
Tree depth: 5
Number of leaves: 32
```

```
# Make predictions
y_pred_limited = dt_limited.predict(X_test)

print("Predictions made!")
print("Sample predictions:", y_pred_limited[:10])
```

```
Predictions made!
Sample predictions: [1 1 1 1 1 1 1 1 1 1]
```

```
# Calculate accuracy
accuracy_limited = accuracy_score(y_test, y_pred_limited)
print("Accuracy:", accuracy_limited)

# Calculate precision
precision_limited = precision_score(y_test, y_pred_limited)
print("Precision:", precision_limited)

# Calculate recall
recall_limited = recall_score(y_test, y_pred_limited)
print("Recall:", recall_limited)

# Calculate F1-score
f1_limited = f1_score(y_test, y_pred_limited)
print("F1-score:", f1_limited)
```

```
Accuracy: 0.7163
Precision: 0.7163
Recall: 1.0
F1-score: 0.8349
```

```
# Confusion matrix
cm_limited = confusion_matrix(y_test, y_pred_limited)
print("Confusion matrix:")
print(cm_limited)

# Classification report
print("Classification report:")
print(classification_report(y_test, y_pred_limited))
```

```
Confusion matrix:
[[   0 1058]
[   0 1175]]
Classification report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1058
           1       0.72      1.00      0.83      1175

    accuracy                           0.72      2233
   macro avg       0.36      0.50      0.42      2233
weighted avg       0.51      0.72      0.60      2233
```

```
# Create comparison table
comparison = pd.DataFrame({
    'Model': ['Decision Tree', 'Limited Decision Tree'],
    'Accuracy': [accuracy, accuracy_limited],
    'Precision': [precision, precision_limited],
    'Recall': [recall, recall_limited],
    'F1-score': [f1, f1_limited]
})

print("Model comparison:")
comparison
```

```
Model comparison:
```

	Model	Accuracy	Precision	Recall	F1-score
0	Decision Tree	0.716	0.716	1.000	0.835
1	Limited Decision Tree	0.716	0.716	1.000	0.835

```
# Create feature importance table
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': dt.feature_importances_
})

print("Feature importance:")
feature_importance.sort_values('Importance', ascending=False)
```

```
Feature importance:
```

	Feature	Importance
0	age	0.000
1	job	0.000
2	marital	0.000
3	education	0.000
4	default	0.000
5	balance	0.000
6	housing	0.000
7	loan	0.000
8	contact	0.000
9	day_of_week	0.000
10	duration	0.000
11	campaign	0.000
12	pdays	0.000
13	previous	0.000
14	poutcome	0.000
15	emp.var.rate	0.000
16	cons.price.idx	0.000
17	cons.conf.idx	0.000
18	euribor3m	0.000
19	nr.employed	0.000

```
# Create visualization
plt.figure(figure_size=(10,6))
feature_importance.sort_values('Importance', ascending=False).head(10).plot(x='Feature', y='Importance', kind='bar')
plt.title('Top 10 Feature Importance')
plt.xlabel('Features')
plt.ylabel('Importance')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```