```
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, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix, roc_curve
from sklearn.preprocessing import MinMaxScaler
from pathlib import Path

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 120)
```

```
df = pd.read_csv("Admission_Predict.csv")
df.head()
```

```
# Standardize columns and create binary target
df = df.copy()

# Strip spaces from column names
df.columns = [c.strip() for c in df.columns]
```

```
# Rename common columns to simpler names
rename_map = {
    'GRE Score': 'GRE',
    'TOEFL Score': 'TOEFL',
    'University Rating': 'Univ_Rating',
    'LOR': 'LOR',
    'LOR ': 'LOR',
    'Chance of Admit': 'Chance',
    'Chance of Admit ': 'Chance',
}
df.rename(columns=rename_map, inplace=True)
```

```
# Determine the probability column for conversion to 0/1
prob_col_candidates = [c for c in df.columns if c.lower().startswith('chance')]
if not prob_col_candidates:
    raise ValueError('Could not find the probability column (Chance of Admit). Please check
your CSV headers.')
prob_col = prob_col_candidates[0]
```

```
# Decide a threshold; modify if your instructor specifies a different one (e.g., 0.7 or 0.8)
threshold = 0.5
df['Admitted'] = (df[prob_col] >= threshold).astype(int)
```

```
expected_cols = ['GRE', 'TOEFL', 'Univ_Rating', 'SOP', 'LOR', 'CGPA', 'Research', 'Admitted']
missing = [c for c in expected_cols if c not in df.columns]
print('Missing (if any):', missing)
df[expected_cols].head()
```

```
# Basic NA handling: drop rows with missing essential values
before = len(df)
df = df.dropna(subset=['GRE', 'TOEFL', 'Univ_Rating', 'SOP', 'LOR', 'CGPA', 'Research',
'Admitted'])
after = len(df)
print(f'Dropped {before - after} rows due to missing values.')
df.describe(include='all')
```

```
# Features and target
X_two = df[['GRE', 'CGPA']]
```

```
X_full = df[['GRE', 'TOEFL', 'Univ_Rating', 'SOP', 'LOR', 'CGPA', 'Research']]
y = df['Admitted']

X2_train, X2_test, y2_train, y2_test = train_test_split(X_two, y, test_size=0.2,
random_state=42, stratify=y)
Xf_train, Xf_test, yf_train, yf_test = train_test_split(X_full, y, test_size=0.2,
random_state=42, stratify=y)
```

```
X2_train.shape, X2_test.shape, Xf_train.shape, Xf_test.shape
```

```
# You can tune hyperparameters (max_depth, min_samples_split, etc.) if desired
tree_two = DecisionTreeClassifier(random_state=42, class_weight='balanced')
tree_two.fit(X2_train, y2_train)

tree_full = DecisionTreeClassifier(random_state=42, class_weight='balanced')
tree_full.fit(Xf_train, yf_train)
```

```
print('Models trained.')
```

```
def evaluate(model, X_tr, X_te, y_tr, y_te, name='Model'):
    ytr_pred = model.predict(X_tr)
    yte_pred = model.predict(X_te)
    # For ROC-AUC, need probabilities; guard if unavailable
    if hasattr(model, 'predict_proba'):
        yte_proba = model.predict_proba(X_te)[:, 1]
        auc = roc_auc_score(y_te, yte_proba)
    else:
        yte_proba = None
        auc = np.nan
    cm = confusion_matrix(y_te, yte_pred)
    metrics = {
        'Accuracy (train)': accuracy_score(y_tr, ytr_pred),
        'Accuracy (test)': accuracy_score(y_te, yte_pred),
        'Precision': precision_score(y_te, yte_pred, zero_division=0),
        'Recall': recall_score(y_te, yte_pred, zero_division=0),
        'F1': f1_score(y_te, yte_pred, zero_division=0),
        'ROC-AUC': auc,
        'Confusion Matrix': cm,
    }
    print(f"\n{name} Results:")
    for k, v in metrics.items():
        if k != 'Confusion Matrix':
            print(f'{k:>18}: {v:.4f}' if isinstance(v, (int, float, np.floating)) else
f'{k:>18}: {v}')
    print('Confusion Matrix (test):\n', cm)

    # Plot ROC Curve if probabilities available
    if yte_proba is not None:
        fpr, tpr, _ = roc_curve(y_te, yte_proba)
        plt.figure()
        plt.plot(fpr, tpr, label=name)
        plt.plot([0, 1], [0, 1], linestyle='--')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'ROC Curve — {name}')
        plt.legend()
        plt.show()
```

```
evaluate(tree_two, X2_train, X2_test, y2_train, y2_test, name='Decision Tree (GRE + CGPA)')
evaluate(tree_full, Xf_train, Xf_test, yf_train, yf_test, name='Decision Tree (All Features)')
# Decision boundary (2D) for GRE vs CGPA model
```

```python
from matplotlib.colors import ListedColormap

def plot_decision_boundary(clf, X, y, title='Decision Boundary (GRE vs CGPA)'):
    x_min, x_max = X.iloc[:,0].min() - 5, X.iloc[:,0].max() + 5
    y_min, y_max = X.iloc[:,1].min() - 0.2, X.iloc[:,1].max() + 0.2
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 300), np.linspace(y_min, y_max, 300))
    grid = np.c_[xx.ravel(), yy.ravel()]
    Z = clf.predict(grid).reshape(xx.shape)
    plt.figure()
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X.iloc[:,0], X.iloc[:,1], c=y, edgecolor='k', alpha=0.7)
    plt.xlabel('GRE')
    plt.ylabel('CGPA')
    plt.title(title)
    plt.show()
```

```python
plot_decision_boundary(tree_two, X_two, y, title='Decision Boundary — Decision Tree (GRE +
CGPA)')
```

```python
# Plot the tree for the 2-feature model (may be large). Adjust max_depth for readability.
plt.figure(figsize=(12, 8))
plot_tree(tree_two, feature_names=['GRE', 'CGPA'], class_names=['No', 'Yes'], filled=True)
plt.title('Decision Tree (GRE + CGPA)')
plt.show()

# Plot the tree for the full model
plt.figure(figsize=(16, 10))
plot_tree(tree_full, feature_names=X_full.columns.tolist(), class_names=['No', 'Yes'],
filled=True)
plt.title('Decision Tree (All Features)')
plt.show()
```