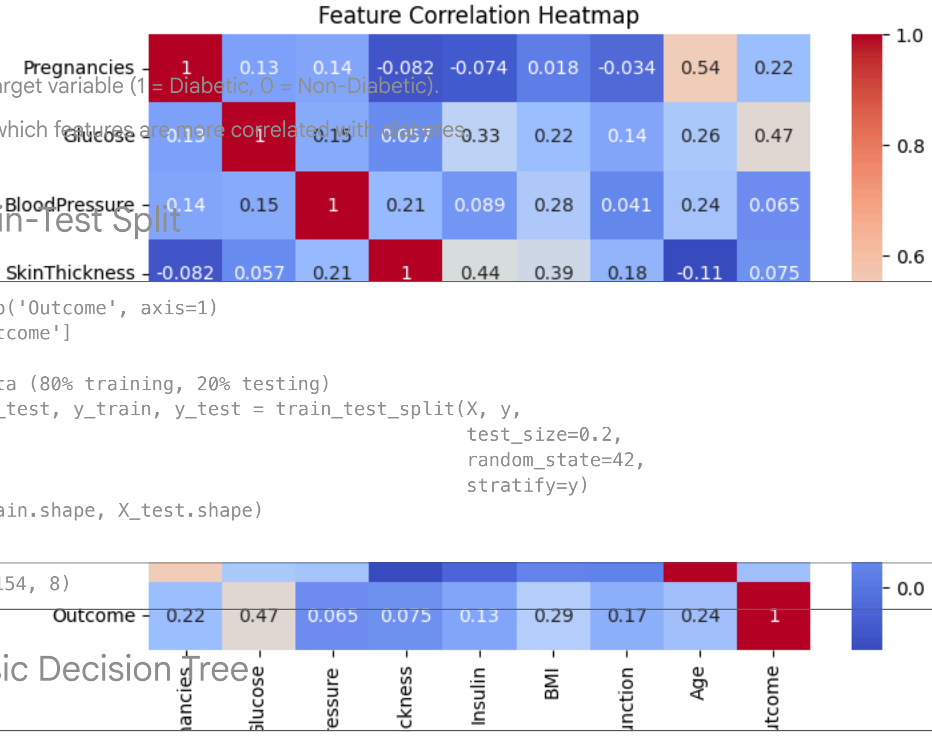
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_graphviz
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import graphviz
from sklearn import tree

# Load dataset (PIMA Indians Diabetes dataset)
# If dataset is not already available, download from Kaggle/UCI ML Repo
url = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv"
df = pd.read_csv(url)

# Show first 5 rows
print(df.head())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

📊 Step 1: Exploratory Data Analysis (EDA)

```
# Basic info
print(df.info())
print(df.describe())

# Check class distribution
print(df['Outcome'].value_counts())

# Correlation heatmap
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
Outcome
0    500
1    268
Name: count, dtype: int64
```



Explanation:

Outcome is the target variable (1 = Diabetic, 0 = Non-Diabetic).

Heatmap shows which features are more correlated with diabetes.

Step 2: Train-Test Split

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

# Split data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y)

print(X_train.shape, X_test.shape)
```

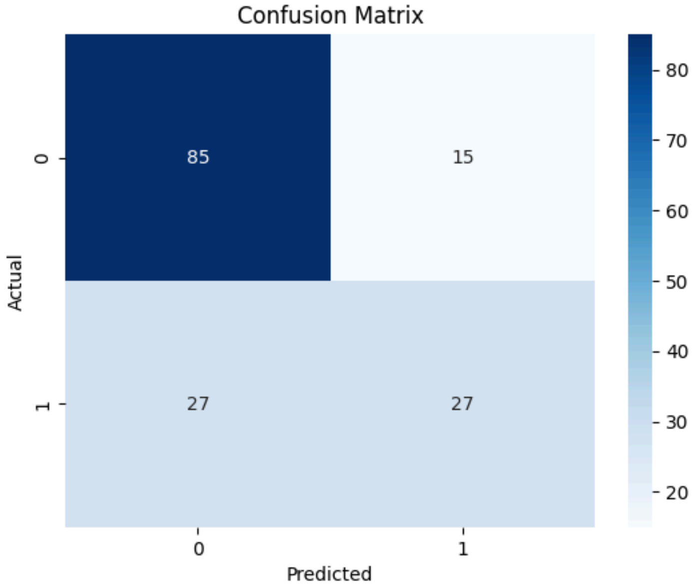
```
(614, 8) (154, 8)
```

Train a Basic Decision Tree

```
# Initialize model
dt = DecisionTreeClassifier(random_state=42)

# Train
dt.fit(X_train, y_train)

# Predictions
y_pred = dt.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues", xticklabels=[0,1], yticklabels=[0,1])
```

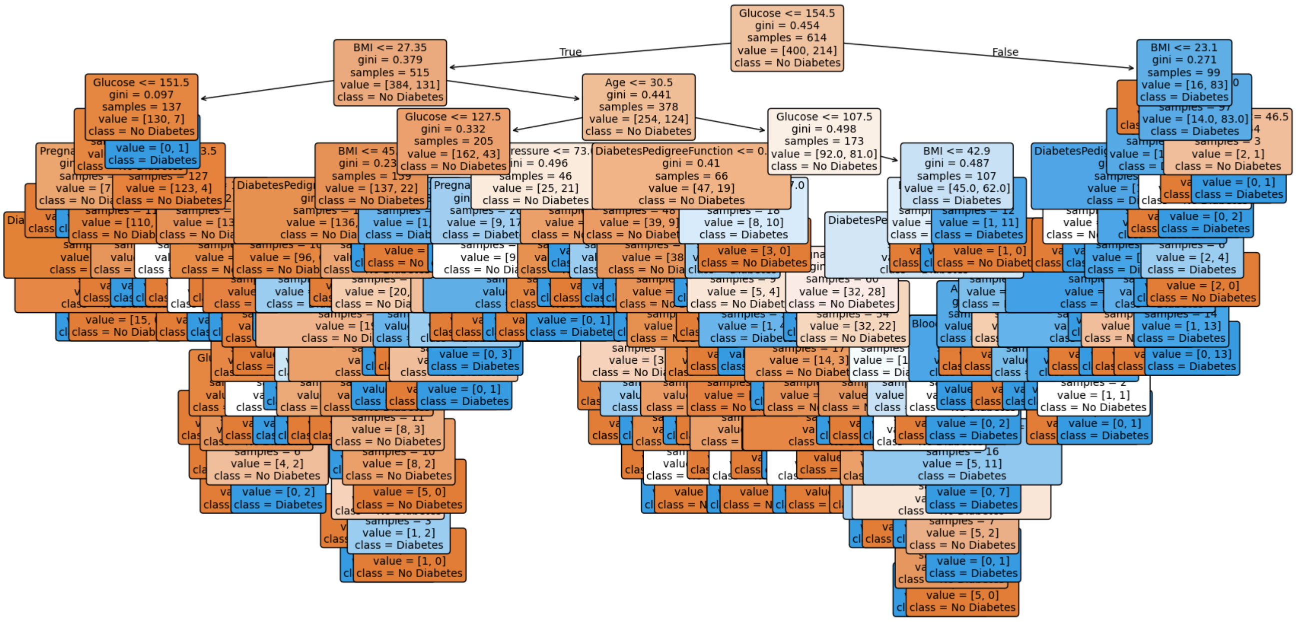
```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
Accuracy: 0.7272727272727273

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.85      0.80       100
           1       0.64      0.50      0.56        54

    accuracy                           0.73       154
   macro avg       0.70      0.68      0.68       154
weighted avg       0.72      0.73      0.72       154
```

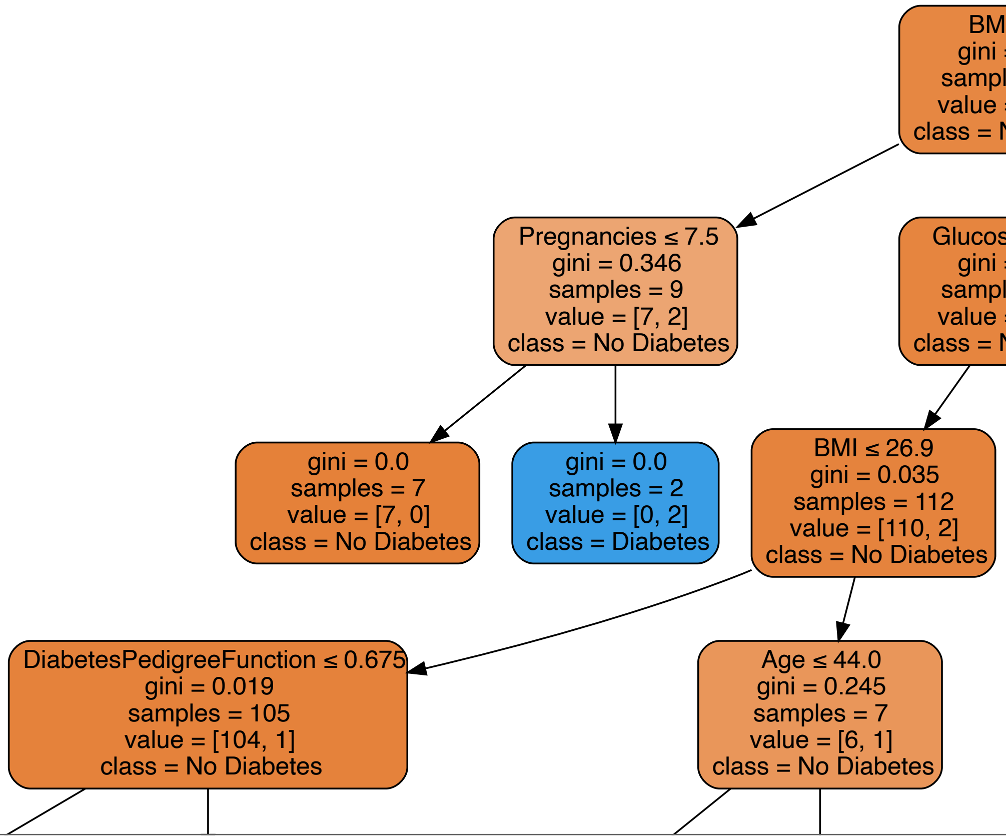


Visualize the Decision Tree

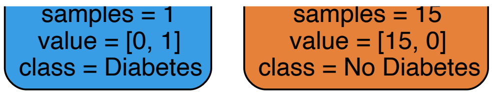
```
plt.figure(figsize=(20,10))
plot_tree(dt, feature_names=X.columns, class_names=["No Diabetes", "Diabetes"],
          filled=True, rounded=True, fontsize=10)
plt.show()
```

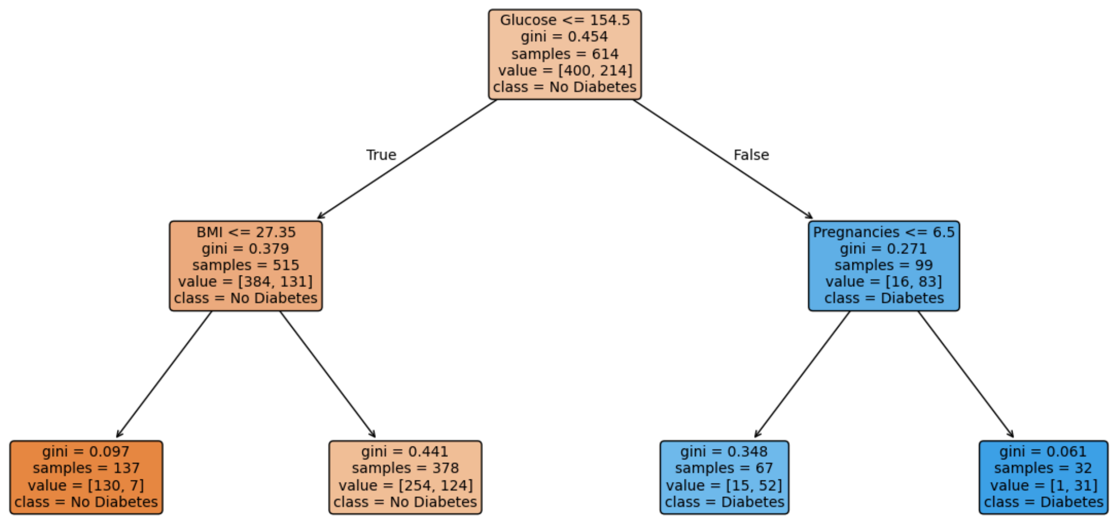


```
dot_data = tree.export_graphviz(dt, out_file=None,feature_names=X.columns,class_names=["No Diabetes", "Diabetes"],
graph = graphviz.Source(dot_data)
graph.render("decision_tree_diabetes", format="png", cleanup=True)
graph
```



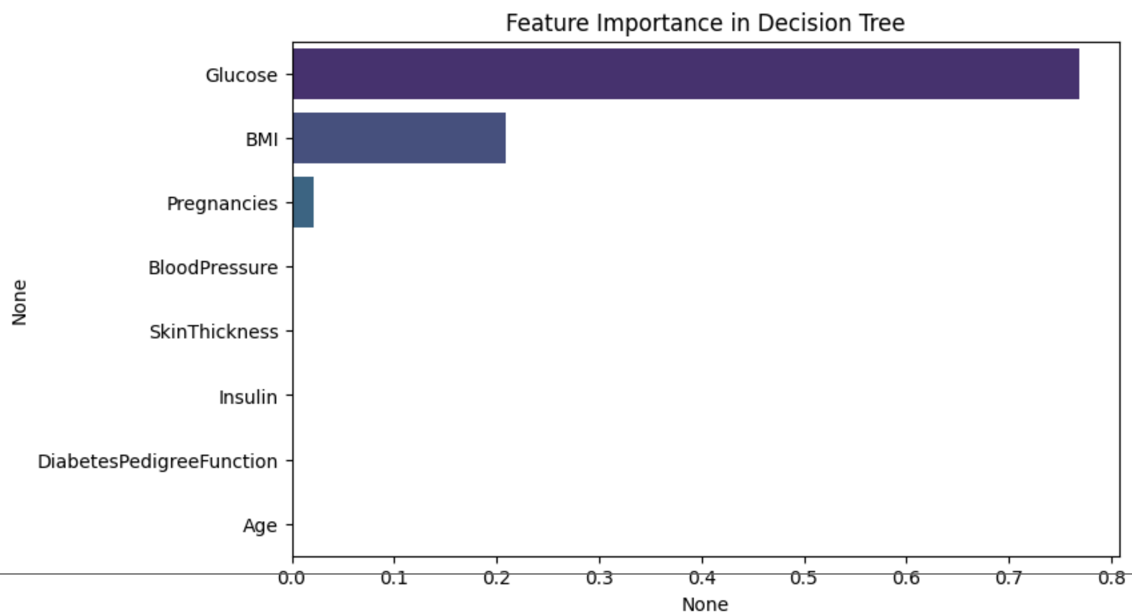

```
plt.figure(figsize=(16,8))
plot_tree(best_dt,
          feature_names=X.columns,
          class_names=["No Diabetes", "Diabetes"],
          filled=True,
          rounded=True,
          fontsize=10,
          max_depth=3)   # show only top 3 levels
plt.show()
```






```
importances = best_dt.feature_importances_
feat_imp = pd.Series(importances, index=X.columns).sort_values(ascending=False)

plt.figure(figsize=(8,5))
sns.barplot(x=feat_imp, y=feat_imp.index, palette="viridis")
plt.title("Feature Importance in Decision Tree")
plt.show()

#Instead of plotting a huge tree, sometimes feature importance is more interpretable:
```

```
/tmp/ipython-input-1553920348.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to

  sns.barplot(x=feat_imp, y=feat_imp.index, palette="viridis")
```



Cross-Validation for Model Validation

```
cv_scores = cross_val_score(dt, X, y, cv=5)
print("Cross-Validation Scores:", cv_scores)
```

```
Cross-Validation Scores: [0.71428571 0.66233766 0.64935065 0.81045752 0.74509804]
Average CV Score: 0.7163059163059163
```

Cross-validation ensures the model generalizes well and avoids overfitting to one train-test split.

Hyperparameter Tuning (Pruning)

```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 4, 6, 8, 10, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10]
}

grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42),
                           param_grid,
                           cv=5,
                           scoring='accuracy',
                           n_jobs=-1,
                           verbose=1)

grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Cross-Validation Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
Best Parameters: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 10, 'min_samples_split': 2}
Best Cross-Validation Score: 0.7557643609222977
```

Evaluate Tuned Model

```
best_dt = grid_search.best_estimator_

# Predict with tuned model
y_pred_best = best_dt.predict(X_test)

print("Tuned Decision Tree Accuracy:", accuracy_score(y_test, y_pred_best))
print("\nClassification Report:\n", classification_report(y_test, y_pred_best))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_best)
sns.heatmap(cm, annot=True, fmt='d', cmap="Greens", xticklabels=[0,1], yticklabels=[0,1])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Tuned Model")
plt.show()

# Visualize tuned tree
plt.figure(figsize=(20,10))
plot_tree(best_dt, feature_names=X.columns, class_names=["No Diabetes", "Diabetes"],
          filled=True, rounded=True, fontsize=10)
plt.show()
```

```
Tuned Decision Tree Accuracy: 0.6948051948051948

Classification Report:
               precision    recall  f1-score   support

           0       0.70      0.92      0.80       100
           1       0.65      0.28      0.39        54

    accuracy                           0.69       154
   macro avg       0.68      0.60      0.59       154
weighted avg       0.68      0.69      0.65       154
```