MACHINE LEARNING LAB - 3

Import necessary libraies

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc
```

Load the dataset

```
##Read the data set
from google.colab import drive
drive.mount("/content/drive")
path = "/content/drive/MyDrive/ML-2/data3.csv"
df = pd.read_csv(path)
value=['Outlook','Temperature','Humidity','Wind']
print(df)
```

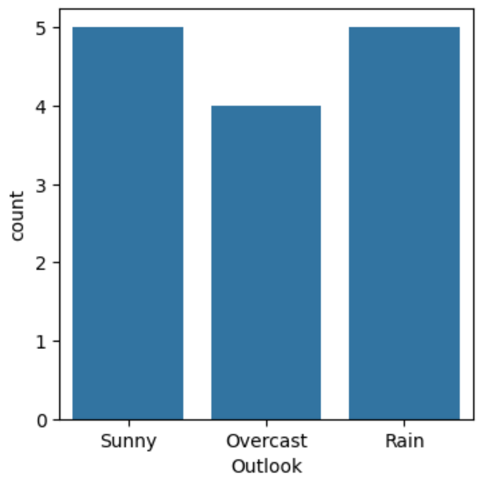
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
     Outlook Temperature Humidity    Wind PlayTennis
0      Sunny         Hot     High    Weak         No
1      Sunny         Hot     High  Strong         No
2   Overcast         Hot     High    Weak        Yes
3       Rain        Mild     High    Weak        Yes
4       Rain        Cool   Normal    Weak        Yes
5       Rain        Cool   Normal  Strong         No
6   Overcast        Cool   Normal  Strong        Yes
7      Sunny        Mild     High    Weak         No
8      Sunny        Cool   Normal    Weak        Yes
9       Rain        Mild   Normal    Weak        Yes
10     Sunny        Mild   Normal  Strong        Yes
11  Overcast        Mild     High  Strong        Yes
12  Overcast         Hot   Normal    Weak        Yes
13      Rain        Mild     High  Strong         No
```

```
df.head()
df.shape
```

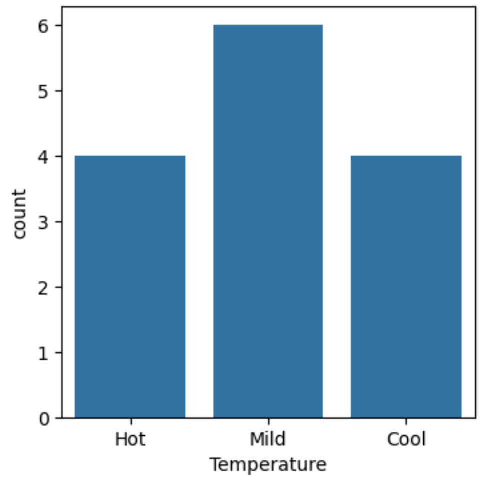
```
(14, 5)
```

EDA

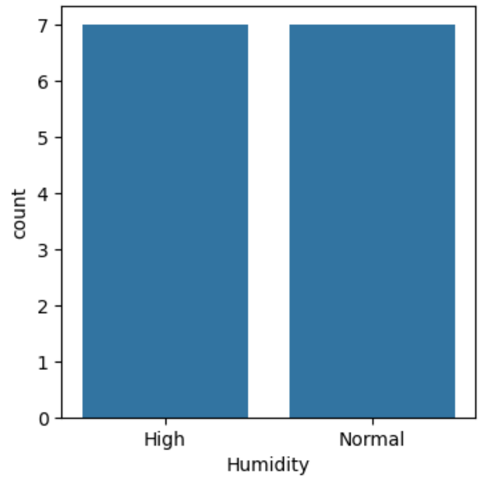
```
ax = plt.subplots(figsize=(4,4))
ax = sns.countplot(x=df['Outlook'])
plt.show()
```



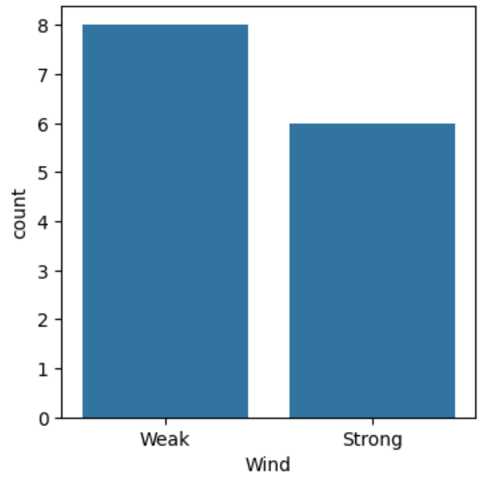
```
ax = plt.subplots(figsize=(4,4))
ax = sns.countplot(x=df['Temperature'])
plt.show()
```



```
ax = plt.subplots(figsize=(4,4))
ax = sns.countplot(x=df['Humidity'])
plt.show()
```



```
ax = plt.subplots(figsize=(4,4))
ax = sns.countplot(x=df['Wind'])
plt.show()
```



Feature Extraction

```
# Separate features (X) and target (y)
```

```
X = df.iloc[:, :-1]  # all columns except last
y = df.iloc[:, -1]   # last column
```

Use OneHotEncoder for categorical variables in features

```
categorical_features = ['Outlook','Temperature','Humidity','Wind']

preprocessor = ColumnTransformer(
    transformers=[
        ('encoder', OneHotEncoder(), categorical_features)
    ],
    remainder='passthrough'
)

X_encoded = preprocessor.fit_transform(X)
```

Use LabelEncoder for the target variable

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

Split the dataset into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0.2, random_state=42)
```

Create a Decision Tree classifier

```
dt_classifier = DecisionTreeClassifier(random_state=42)
```

Fit the model to the training data

```
dt_classifier.fit(X_train, y_train)
```

```
▾     DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

Make predictions on the test set

```
y_pred = dt_classifier.predict(X_test)
```

Evaluate the classifier

```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
```

Print results

```
print(f"DT Accuracy: {accuracy:.2f}")
print(f"DT Confusion Matrix:\n{conf_matrix}")
print(f"DT Classification Report:\n{class_report}")
```

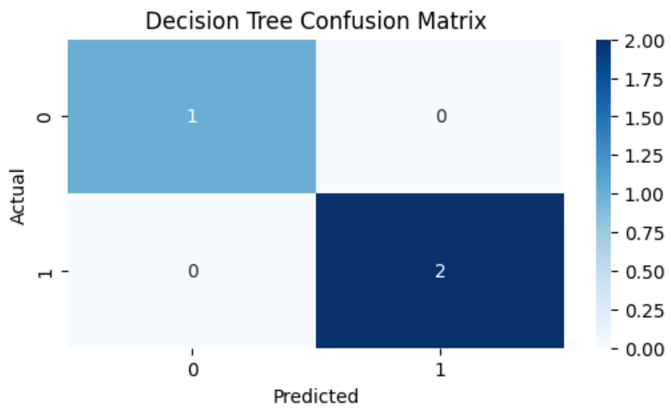
```
DT Accuracy: 1.00
DT Confusion Matrix:
[[1 0]
 [0 2]]
DT Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3
```

Visualize Confusion Matrix

```
plt.figure(figsize=(6,3))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Decision Tree Confusion Matrix")
plt.show()
```



Cross-validation

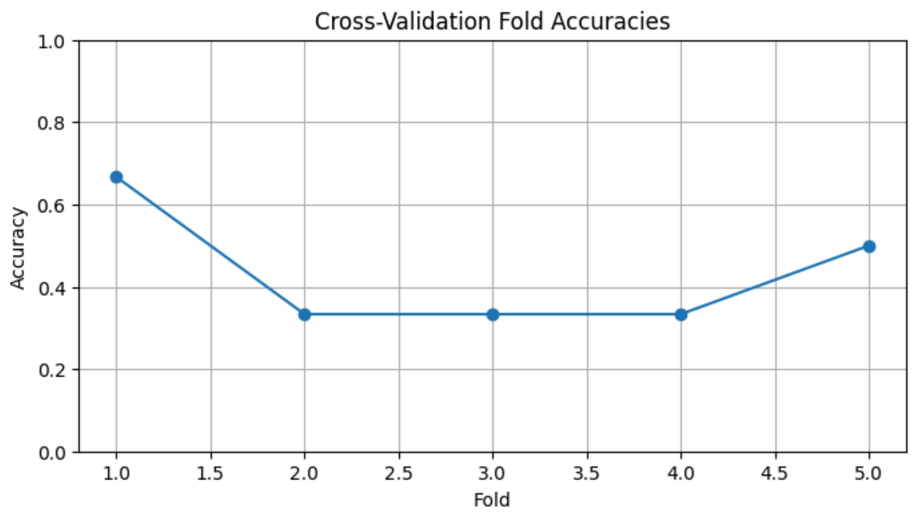
```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scoring = 'accuracy'
cv_results = cross_val_score(dt_classifier, X_encoded, y_encoded, cv=cv, scoring=scoring)

print("Cross-Validation Results:")
print("Individual Accuracies:", cv_results)
print("Average Accuracy:", np.mean(cv_results))
```

```
Cross-Validation Results:
Individual Accuracies: [0.66666667 0.33333333 0.33333333 0.33333333 0.5        ]
Average Accuracy: 0.43333333333333335
```

Plot individual fold accuracies

```
plt.figure(figsize=(8,4))
plt.plot(range(1, len(cv_results)+1), cv_results, marker='o')
plt.xlabel("Fold")
plt.ylabel("Accuracy")
plt.title("Cross-Validation Fold Accuracies")
plt.ylim([0,1])
plt.grid(True)
plt.show()
```



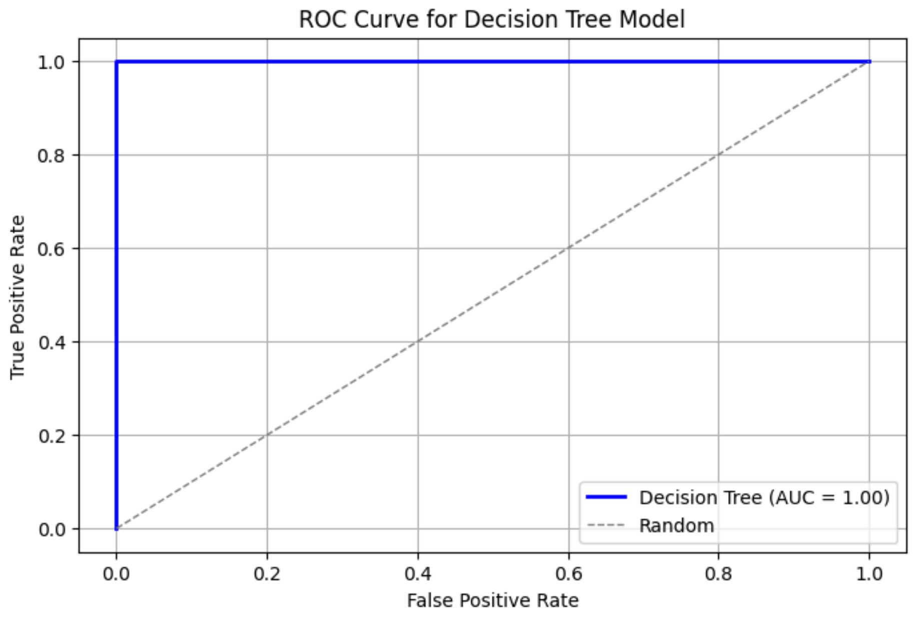
ROC Curve and AUC

Note: For binary classification (Yes/No), this works fine

```
y_test_binary = label_encoder.transform(label_encoder.inverse_transform(y_test))
y_pred_proba = dt_classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test_binary, y_pred_proba)
roc_auc = auc(fpr, tpr)
```

Plot ROC Curve

```
plt.figure(figsize=(8,5))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'Decision Tree (AUC = {roc_auc:.2f})')
plt.plot([0,1], [0,1], color='gray', lw=1, linestyle='--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Decision Tree Model')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()
```


ROC Curve for Decision Tree Model
True Positive Rate
False Positive Rate
Decision Tree (AUC = 1.00)
Random