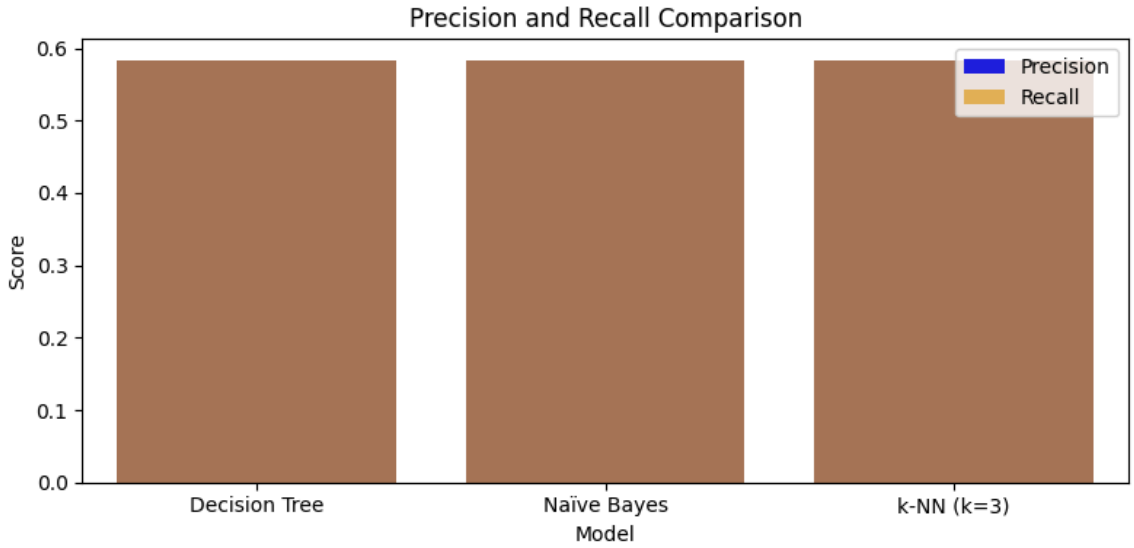
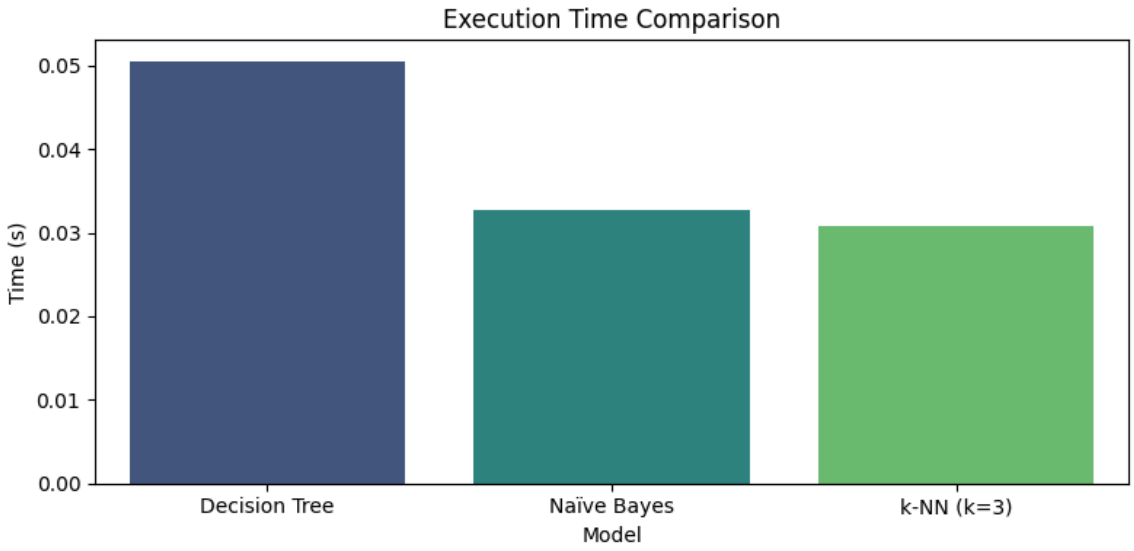
```
# 📌 Import necessary libraries
import pandas as pd
import time
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score, recall_score
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# 📌 Load the dataset
file_path = '/content/PlayTennis.csv'  # Update path if needed
weather = pd.read_csv(file_path)

# 📌 Save original data before encoding
weather_original = weather.copy()

# 📌 Encode categorical variables with separate LabelEncoder instances for each column
encoders = {}
weather_encoded = weather.copy()
for column in weather.columns:
    encoders[column] = LabelEncoder()
    weather_encoded[column] = encoders[column].fit_transform(weather[column])

# 📌 Split data into features and target
X = weather_encoded.drop('Play Tennis', axis=1)
y = weather_encoded['Play Tennis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 📌 Function to evaluate models
def evaluate_model(model, name):
    start_time = time.time()
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    end_time = time.time()
    precision = precision_score(y_test, predictions, average='macro')
    recall = recall_score(y_test, predictions, average='macro')
    exec_time = round(end_time - start_time, 4)
    return {'Model': name, 'Precision': precision, 'Recall': recall, 'Execution Time': exec_time}

# 📌 Evaluate classifiers
results = []
results.append(evaluate_model(DecisionTreeClassifier(), "Decision Tree"))
results.append(evaluate_model(CategoricalNB(), "Naïve Bayes"))
results.append(evaluate_model(KNeighborsClassifier(n_neighbors=3), "k-NN (k=3)"))

# 📌 Convert results to DataFrame
results_df = pd.DataFrame(results)

# 📊 Plot Precision and Recall
plt.figure(figsize=(8, 4))
sns.barplot(x='Model', y='Precision', data=results_df, color='blue', label='Precision')
sns.barplot(x='Model', y='Recall', data=results_df, color='orange', label='Recall', alpha=0.7)
plt.title('Precision and Recall Comparison')
plt.ylabel('Score')
plt.legend()
plt.tight_layout()
plt.show()

# 📊 Plot Execution Time
plt.figure(figsize=(8, 4))
sns.barplot(x='Model', y='Execution Time', data=results_df, hue='Model', palette='viridis', legend=False)
plt.title('Execution Time Comparison')
plt.ylabel('Time (s)')
plt.tight_layout()
plt.show()

# ---------------------------------------------
# 📌 Association Rule Mining (Apriori Algorithm)
# ---------------------------------------------
```

```
# 📌 Prepare transactions using the original unencoded data
transactions = weather_original.apply(lambda row: [f"{col}={row[col]}" for col in weather_original.columns], axis=1).tolist()

# 📌 Transaction Encoding
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df = pd.DataFrame(te_ary, columns=te.columns_)

# 📌 Apply Apriori
frequent_itemsets = apriori(df, min_support=0.3, use_colnames=True)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# 📌 Display Association Rules
print("\n📌 Association Rules (support >= 0.3, confidence >= 0.7):\n")
for idx, rule in rules.iterrows():
    print(f"Rule: {set(rule['antecedents'])} => {set(rule['consequents'])}")
    print(f" - Support: {rule['support']:.2f}")
    print(f" - Confidence: {rule['confidence']:.2f}")
    print(f" - Lift: {rule['lift']:.2f}\n")
```






```
📌 Association Rules (support >= 0.3, confidence >= 0.7):

Rule: {'Humidity=Normal'} => {'Play Tennis=Yes'}
 - Support: 0.43
 - Confidence: 0.86
 - Lift: 1.33

Rule: {'Wind=Weak'} => {'Play Tennis=Yes'}
 - Support: 0.43
 - Confidence: 0.75
 - Lift: 1.17
```