import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import make\_column\_selector as selector

from sklearn.metrics import confusion\_matrix

# Read the dataset

data = pd.read\_csv("/content/netflix\_titles.csv", encoding='latin1')

# Dataset Description

print("Dataset Size:")

print("Number of rows:", data.shape[0])

print("Number of columns:", data.shape[1])

print("\nDataset Features:")

print(data.columns)

print("\nPreprocessing Steps:")

print("- Missing values in the 'rating' column were handled by dropping rows with missing target values.")

print("- Numerical features were imputed using the median strategy and scaled using StandardScaler.")

print("- Categorical features were imputed using the most frequent strategy and one-hot encoded using OneHotEncoder.")

# Drop rows with missing target values

data = data.dropna(subset=['rating'])

# Select features and target variable

X = data[['director', 'cast']]

y = data['rating']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Preprocessing for numerical features

numerical\_features = selector(dtype\_include=np.number)(X\_train)

numerical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='median')),

('scaler', StandardScaler())])

# Preprocessing for categorical features

categorical\_features = selector(dtype\_exclude=np.number)(X\_train)

categorical\_transformer = Pipeline(steps=[

('imputer', SimpleImputer(strategy='most\_frequent')),

('onehot', OneHotEncoder(handle\_unknown='ignore'))])

# Bundle preprocessing for numerical and categorical features

preprocessor = ColumnTransformer(

transformers=[

('num', numerical\_transformer, numerical\_features),

('cat', categorical\_transformer, categorical\_features)])

# Define the classifiers

decision\_tree\_classifier = Pipeline(steps=[('preprocessor', preprocessor),

('classifier', DecisionTreeClassifier())])

random\_forest\_classifier = Pipeline(steps=[('preprocessor', preprocessor),

('classifier', RandomForestClassifier())])

# Fit the models

decision\_tree\_classifier.fit(X\_train, y\_train)

random\_forest\_classifier.fit(X\_train, y\_train)

# Predict on the test sets

y\_pred\_dt = decision\_tree\_classifier.predict(X\_test)

y\_pred\_rf = random\_forest\_classifier.predict(X\_test)

# Evaluate the models

accuracy\_dt = accuracy\_score(y\_test, y\_pred\_dt)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_rf)

print("Decision Tree Classifier Accuracy:", accuracy\_dt)

print("Random Forest Classifier Accuracy:", accuracy\_rf)

# Calculate confusion matrix for Decision Tree

cm\_dt = confusion\_matrix(y\_test, y\_pred\_dt)

# Plot confusion matrix for Decision Tree

plt.imshow(cm\_dt, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Decision Tree Confusion Matrix')

plt.colorbar()

tick\_marks\_dt = np.arange(len(decision\_tree\_classifier.classes\_))

plt.xticks(tick\_marks\_dt, decision\_tree\_classifier.classes\_, rotation=45)

plt.yticks(tick\_marks\_dt, decision\_tree\_classifier.classes\_)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# Calculate confusion matrix for Random Forest

cm\_rf = confusion\_matrix(y\_test, y\_pred\_rf)

# Plot confusion matrix for Random Forest

plt.imshow(cm\_rf, interpolation='nearest', cmap=plt.cm.Blues)

plt.title('Random Forest Confusion Matrix')

plt.colorbar()

tick\_marks\_rf = np.arange(len(random\_forest\_classifier.classes\_))

plt.xticks(tick\_marks\_rf, random\_forest\_classifier.classes\_, rotation=45)

plt.yticks(tick\_marks\_rf, random\_forest\_classifier.classes\_)

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

# 1. Bar Plot of Correct vs. Incorrect Predictions

correct\_predictions = (y\_test == y\_pred).sum()

incorrect\_predictions = len(y\_test) - correct\_predictions

plt.figure(figsize=(8, 6))

plt.bar(['Correct Predictions', 'Incorrect Predictions'], [correct\_predictions, incorrect\_predictions], color=['green', 'red'])

plt.xlabel('Prediction Outcome')

plt.ylabel('Count')

plt.title('Bar Plot of Correct vs. Incorrect Predictions')

plt.show()

# 2. Bar Plot of Rating Distribution

plt.figure(figsize=(10, 6))

y\_test.value\_counts().plot(kind='bar')

plt.xlabel('Rating')

plt.ylabel('Count')

plt.title('Rating Distribution')

plt.show()

# 3. Histogram of Predicted Probabilities for Each Class

plt.figure(figsize=(10, 6))

for i, rating in enumerate(classifier.classes\_):

plt.hist(classifier.predict\_proba(X\_test)[:, i], bins=10, alpha=0.5, label=rating)

plt.xlabel('Predicted Probability')

plt.ylabel('Frequency')

plt.title('Histogram of Predicted Probabilities for Each Class')

plt.legend()

plt.show()

# 4. Precision-Recall Curve for Each Class

plt.figure(figsize=(8, 6))

for i, rating in enumerate(classifier.classes\_):

precision, recall, \_ = precision\_recall\_curve(y\_test, classifier.predict\_proba(X\_test)[:, i], pos\_label=rating)

plt.plot(recall, precision, marker='.', label=rating)

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision-Recall Curve for Each Class')

plt.legend()

plt.show()

# 5. Receiver Operating Characteristic (ROC) Curve for Each Class

plt.figure(figsize=(8, 6))

for i, rating in enumerate(classifier.classes\_):

fpr, tpr, \_ = roc\_curve(y\_test, classifier.predict\_proba(X\_test)[:, i], pos\_label=rating)

plt.plot(fpr, tpr, marker='.', label=rating)

plt.plot([0, 1], [0, 1], linestyle='--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve for Each Class')

plt.legend()

plt.show()