GROUP 14_CINESENSE ADVISOR USING MACHINE LEARNING

Aim:
CineSense Recommendation System with Machine Learning Model Comparison

Procedure:

The following procedure explains the code line by line, with corresponding line numbers, so you can follow the logic of the script:

1. Import Libraries (Lines 1-11): The required libraries for the project are imported, including pandas, numpy, streamlit, and several machine learning and visualization packages.
2. Load the Dataset (Line 13): The dataset 'All_Streaming_Shows.csv' is loaded into a DataFrame using pd.read_csv(). The file path should be updated according to the local or cloud location.
3. Handle Infinite Values (Line 16): Infinite values in the dataset are replaced with NaN using replace(). This ensures any irregularities in the data don't affect further processing.
4. Data Preprocessing (Lines 19-25):
    - Rows with missing values in key columns such as 'IMDB Rating', 'Genre', 'Streaming Platform', and 'Content Rating' are dropped to clean the dataset.
    - The 'IMDB Rating' column is converted to numeric, and any non-convertible entries are coerced into NaN values, which are also dropped.
5. Encoding and Scaling (Lines 28-33):
    - Label encoding is applied to categorical columns ('Genre' and 'Streaming Platform') to convert them into numerical format.
    - Standard scaling is performed on the 'IMDB Rating' to normalize the data, which helps the machine learning models work better.
6. Prepare Features and Target Variable (Lines 36-38):
    - A new column Target is created where shows with an IMDb rating of 7.0 or higher are marked as 1 (positive), and others as 0.
    - Features for the model include encoded genre, platform, and scaled IMDb rating, while the target variable is whether a show is highly rated or not.
7. Split the Dataset (Lines 41-43): The dataset is split into training and testing sets (80/20 split) using train_test_split(), ensuring we have separate data for training the models and testing them later.
8. Initialize Models (Lines 46-49): Several classifiers (Decision Tree, SVM, Naive Bayes, K-Nearest Neighbors) are initialized to compare performance later.
9. Train Models (Lines 52-55): All the initialized models are trained on the training data (X_train, y_train).
10. Streamlit UI Setup (Lines 58-61): Streamlit is used to build an interactive web UI where users can enter their age, preferred genre, and streaming platform. This data will later be used to recommend shows.
11. Model Performance Comparison Header (Line 63): A header is added in Streamlit to indicate the section where model performance will be compared.

12. Evaluate Models (Lines 66-73): A function evaluate_model() is defined to calculate confusion matrix, classification report, and ROC curve for each model.
13. Plot Confusion Matrix (Lines 76-81): Another function is defined to plot the confusion matrix using Seaborn and display it on the Streamlit app.
14. Plot ROC Curve (Lines 84-93): The ROC curve is plotted using Matplotlib and displayed in Streamlit for each model to visualize the model's ability to distinguish between the positive and negative classes.
15. Display Model Results (Lines 96-103): For each model (Decision Tree, SVM, Naive Bayes, KNN), the evaluation results (classification report, confusion matrix, and ROC curve) are displayed in Streamlit.
16. Feature Importance for Decision Tree (Lines 106-112): A bar plot shows the feature importance for the Decision Tree model to help understand which features contribute most to the predictions.
17. Show Recommendation Function (Lines 115-127): A function recommend_best_shows()is defined to recommend top shows based on user inputs (age, genre, platform). It also restricts certain genres for users under 18.
18. Recommendation Section UI (Lines 130-143): Streamlit tabs are created to display the top 10 and top 20 show recommendations based on the user’s input for age, genre, and platform. If the user clicks the button, the recommendations are displayed.

Source Code:

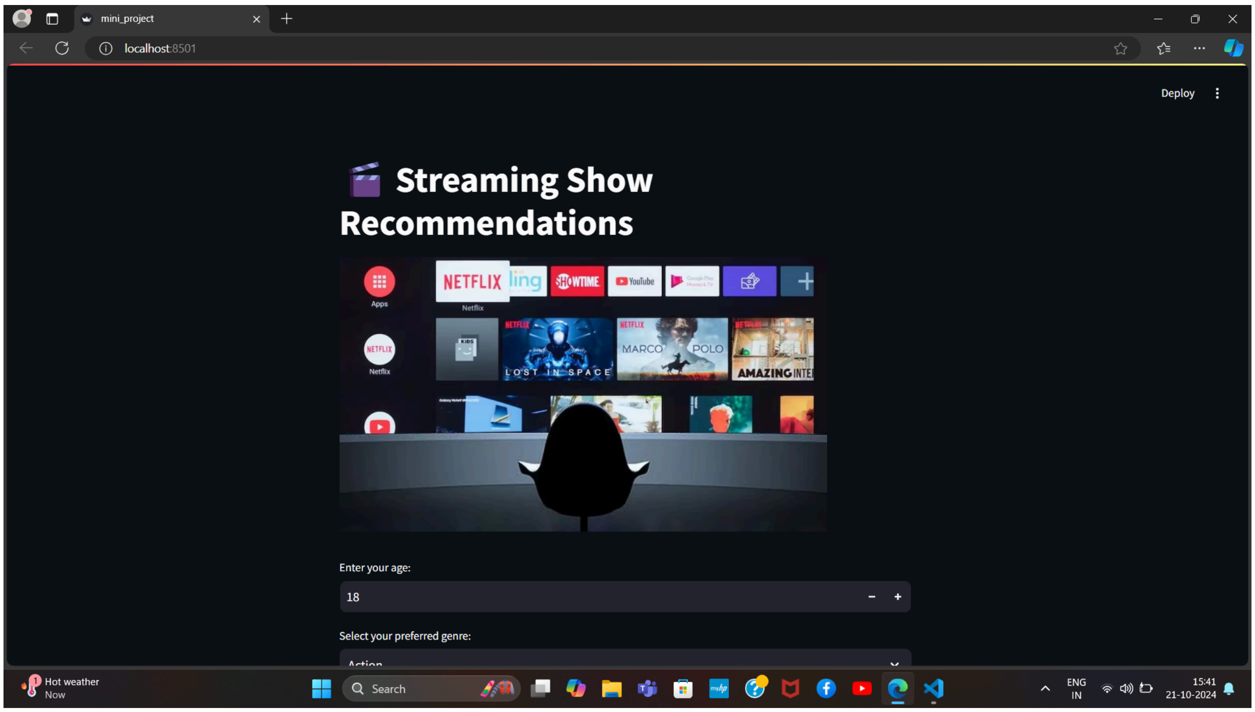
```
import pandas as pd  # 1
import numpy as np  # 2
import streamlit as st  # 3
import matplotlib.pyplot as plt  # 4
import seaborn as sns  # 5
from sklearn.preprocessing import LabelEncoder, StandardScaler  # 6
from sklearn.model_selection import train_test_split  # 7
from sklearn.tree import DecisionTreeClassifier  # 8
from sklearn.svm import SVC  # 9
from sklearn.naive_bayes import GaussianNB  # 10
from sklearn.neighbors import KNeighborsClassifier  # 11
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, auc  # 12
import xgboost as xgb  # Keeping it here if needed later (Line 13)

# Load the dataset (Line 16)
file_path = 'All_Streaming_Shows.csv'  # Update with your correct file path (Line 17)
shows_df = pd.read_csv(file_path)  # 18

# Handle infinite values by replacing them with NaN (Line 20)
shows_df.replace([np.inf, -np.inf], np.nan, inplace=True)  # 21

# Data Preprocessing (Lines 23-29)
```

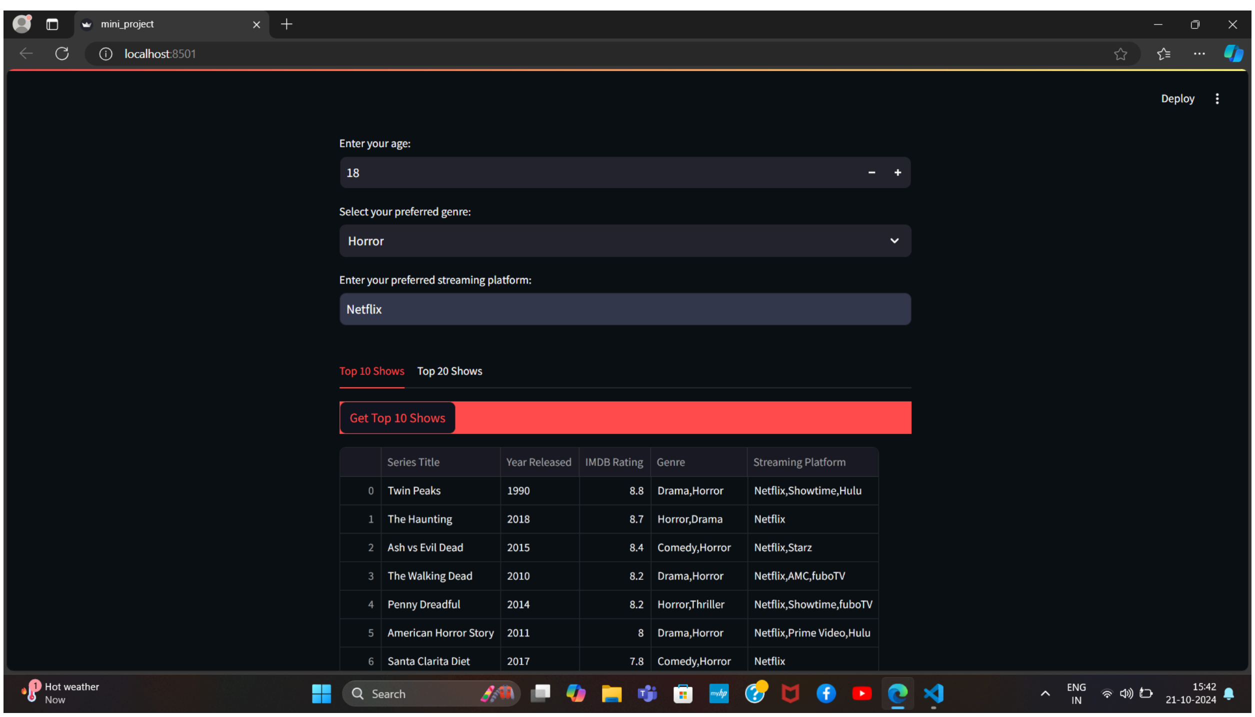
```
cleaned_df = shows_df.dropna(subset=['IMDB Rating', 'Genre', 'Streaming Platform', 'Content Rating']).copy()  # 24
cleaned_df['IMDB Rating'] = pd.to_numeric(cleaned_df['IMDB Rating'], errors='coerce')  # 25
cleaned_df.dropna(subset=['IMDB Rating'], inplace=True)  # 26

# Encoding and Scaling (Lines 31-36)
le_genre = LabelEncoder()  # 32
le_platform = LabelEncoder()  # 33
cleaned_df['Genre Encoded'] = le_genre.fit_transform(cleaned_df['Genre'])  # 34
cleaned_df['Platform Encoded'] = le_platform.fit_transform(cleaned_df['Streaming Platform'])  # 35

scaler = StandardScaler()  # 37
cleaned_df['Scaled IMDb Rating'] = scaler.fit_transform(cleaned_df[['IMDB Rating']])  # 38

# Prepare features and target variable (Lines 40-42)
cleaned_df['Target'] = cleaned_df['IMDB Rating'] >= 7.0  # 41
X = cleaned_df[['Genre Encoded', 'Platform Encoded', 'Scaled IMDb Rating']]  # 42
y = cleaned_df['Target'].astype(int)  # 43

# Split the dataset into training and testing sets (Lines 45-47)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  # 46

# Initialize Models (Lines 49-53)
dt_model = DecisionTreeClassifier(random_state=42)  # 50
svm_model = SVC(probability=True, random_state=42)  # 51
nb_model = GaussianNB()  # 52
knn_model = KNeighborsClassifier(n_neighbors=5)  # 53

# Train Models (Lines 55-57)
models = {  # 56
   'Decision Tree': dt_model,  # 57
   'Support Vector Machine': svm_model,  # 58
   'Naive Bayes': nb_model,  # 59
   'K-Nearest Neighbors': knn_model  # 60
}
for name, model in models.items():  # 61
   model.fit(X_train, y_train)  # 62

# Streamlit UI (Lines 65-68)
st.title("🎬 Streaming Show Recommendations with Model Comparison")  # 66

# Input Section (Lines 70-73)
age = st.number_input("Enter your age:", min_value=0, max_value=100, value=18)  # 71
genre = st.text_input("Enter your preferred genre:")  # 72
platform = st.text_input("Enter your preferred streaming platform:")  # 73
```

```
# Performance Comparison (Line 75)
st.header("Model Performance Comparison")  # 76

# Evaluate Models (Lines 79-86)
def evaluate_model(model, X_test, y_test):  # 80
    y_pred = model.predict(X_test)  # 81
    cm = confusion_matrix(y_test, y_pred)  # 82
    cr = classification_report(y_test, y_pred)  # 83
    fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])  # 84
    roc_auc = auc(fpr, tpr)  # 85
    return cm, cr, fpr, tpr, roc_auc  # 86

# Plot Confusion Matrix (Lines 89-95)
def plot_confusion_matrix(cm, model_name):  # 90
    plt.figure(figsize=(6, 4))  # 91
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')  # 92
    plt.title(f'Confusion Matrix for {model_name}')  # 93
    plt.ylabel('Actual Label')  # 94
    plt.xlabel('Predicted Label')  # 95
    st.pyplot(plt)  # 96

# Plot ROC Curve (Lines 98-106)
def plot_roc_curve(fpr, tpr, roc_auc, model_name):  # 99
    plt.figure(figsize=(6, 4))  # 100
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (area = {roc_auc:0.2f}')  # 101
    plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')  # 102
    plt.xlim([0.0, 1.0])  # 103
    plt.ylim([0.0, 1.05])  # 104
    plt.xlabel('False Positive Rate')  # 105
    plt.ylabel('True Positive Rate')  # 106
    plt.title(f'Receiver Operating Characteristic for {model_name}')  # 107
    plt.legend(loc="lower right")  # 108
    st.pyplot(plt)  # 109

# Display results for all models (Lines 111-119)
for name, model in models.items():  # 112
    cm, cr, fpr, tpr, roc_auc = evaluate_model(model, X_test, y_test)  # 113
    st.subheader(f"{name}")  # 114
    st.text("Classification Report:\n" + cr)  # 115
    plot_confusion_matrix(cm, name)  # 116
    plot_roc_curve(fpr, tpr, roc_auc, name)  # 117

# Feature Importance for Decision Tree (Lines 119-126)
st.subheader("Feature Importance for Decision Tree")  # 119
feature_importance = dt_model.feature_importances_  # 120
```

```
features = X.columns  # 121
plt.figure(figsize=(8, 6))  # 122
sns.barplot(x=feature_importance, y=features)  # 123
plt.title("Feature Importance")  # 124
st.pyplot(plt)  # 125

# Show recommendation function remains unchanged (Lines 128-133)
def recommend_best_shows(age, genre, platform, min_imdb_rating=7.0, top_n=10):  # 129
    valid_ratings = get_valid_ratings(age)  # 130
    filtered_df = cleaned_df[  # 131
        (cleaned_df['Content Rating'].isin(valid_ratings)) &  # 132
        (cleaned_df['Genre'].str.contains(genre, case=False, na=False)) &  # 133
        (cleaned_df['Streaming Platform'].str.contains(platform, case=False, na=False)) &  # 134
        (cleaned_df['IMDB Rating'] >= min_imdb_rating)  # 135
    ]  # 136
    if age < 18:  # 137
        restricted_genres = ['Romance', 'Thriller', 'Horror', 'Crime']  # 138
        restricted_shows = filtered_df[filtered_df['Genre'].str.contains('|'.join(restricted_genres), case=False)]  # 139
        if not restricted_shows.empty:  # 140
            return "Sorry, the genre(s) you selected are restricted for your age group."  # 141
    top_shows = filtered_df.sort_values(by='IMDB Rating', ascending=False).head(top_n)  # 142
    return top_shows[['Series Title', 'Year Released', 'IMDB Rating', 'Genre', 'Streaming Platform']]  # 143

# User Inputs for Recommendations (Lines 145-157)
st.header("Streaming Show Recommendations")  # 145

tab_10, tab_20 = st.tabs(["Top 10 Shows", "Top 20 Shows"])  # 146
with tab_10:  # 147
    if st.button("Get Top 10 Shows"):  # 148
        if genre and platform:  # 149
            results = recommend_best_shows(age, genre, platform, top_n=10)  # 150
            st.write(results)  # 151
        else:  # 152
            st.write("Please enter both genre and platform.")  # 153

with tab_20:  # 155
    if st.button("Get Top 20 Shows"):  # 156
        if genre and platform:  # 157
            results = recommend_best_shows(age, genre, platform, top_n=20)  # 158
            st.write(results)  # 159
        else:  # 160
            st.write("Please enter both genre and platform.")  # 161
```

User Interface:





Model Evaluation with Accuracy, Precision, and F1 Score Metrics:

```
# Evaluate Models - Added F1 score, precision, accuracy
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
   # Calculate metrics
    accuracy = metrics.accuracy_score(y_test, y_pred)
    precision = metrics.precision_score(y_test, y_pred)
    f1 = metrics.f1_score(y_test, y_pred)
    return cm, accuracy, precision, f1
    # Print performance metrics for each model
    for name, model in models.items():
    cm, accuracy, precision, f1 = evaluate_model(model, X_test, y_test)
    print(f"Model: {name}")
    print(f"Accuracy: {accuracy:.2f}")
    print(f"Precision: {precision:.2f}")
    print(f"F1 Score: {f1:.2f}")
    print("Confusion Matrix:")
    print(cm)
    print("\n" + "-"*40 + "\n")
```

```
Model: Support Vector Machine
Accuracy: 0.65
Precision: 0.66
F1 Score: 0.78
Confusion Matrix:
[[ 66 344]
 [ 46 673]]
```

```
Model: Decision Tree
Accuracy: 1.00
Precision: 1.00
F1 Score: 1.00
Confusion Matrix:
[[410   0]
 [  0 719]]
```

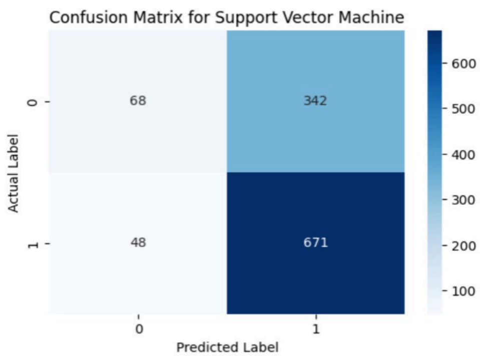
```
Model: Naive Bayes
Accuracy: 0.96
Precision: 0.95
F1 Score: 0.97
Confusion Matrix:
[[373  37]
 [  5 714]]
```

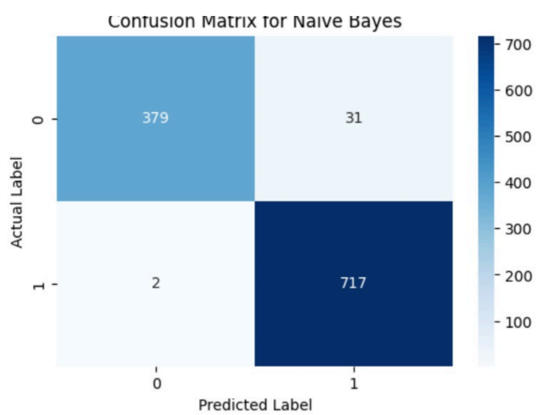
```
Model: K-Nearest Neighbors
Accuracy: 0.75
Precision: 0.78
F1 Score: 0.81
Confusion Matrix:
[[232 178]
 [105 614]]
```

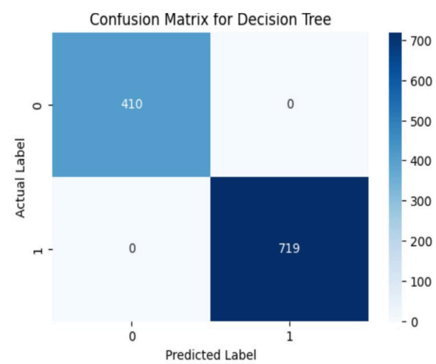
Code for confusion matrix plots:

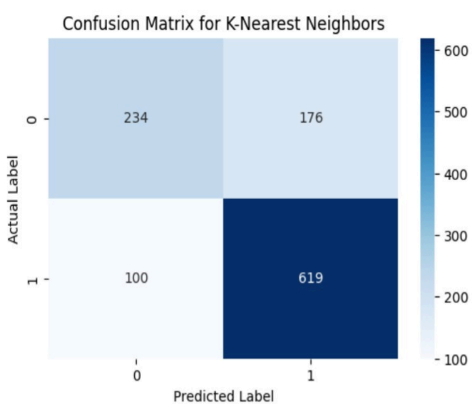
```
# Import necessary libraries for metrics
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Function to evaluate and plot confusion matrix for each model
def plot_confusion_matrix(model, X_test, y_test, model_name):


# Get predictions
    y_pred = model.predict(X_test)

  # Calculate confusion matrix
    cm = confusion_matrix(y_test, y_pred)


 # Plot confusion matrix using Seaborn heatmap
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(‘Confusion Matrix - {model_name}')
    plt.ylabel('Actual Label')
    plt.xlabel('Predicted Label')
    plt.show()

# Loop through each model to print its confusion matrix
for name, model in models.items():
    plot_confusion_matrix(model, X_test, y_test, name)
```

Confusion matrix:

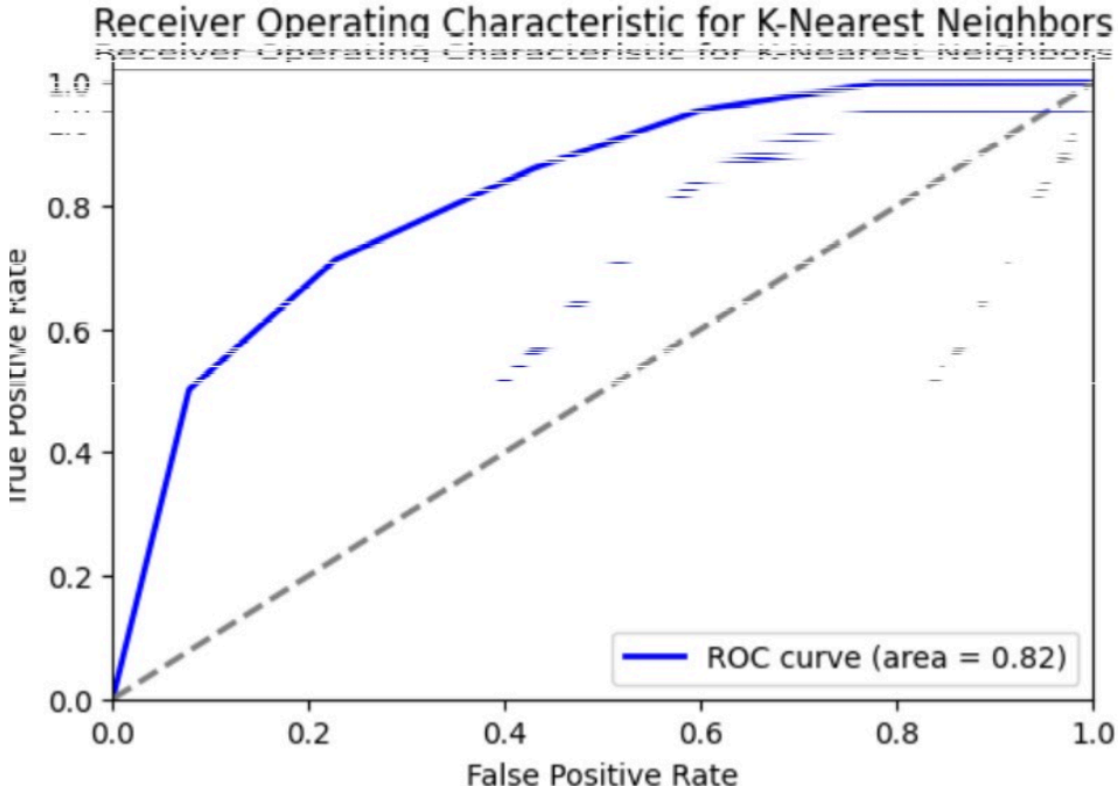




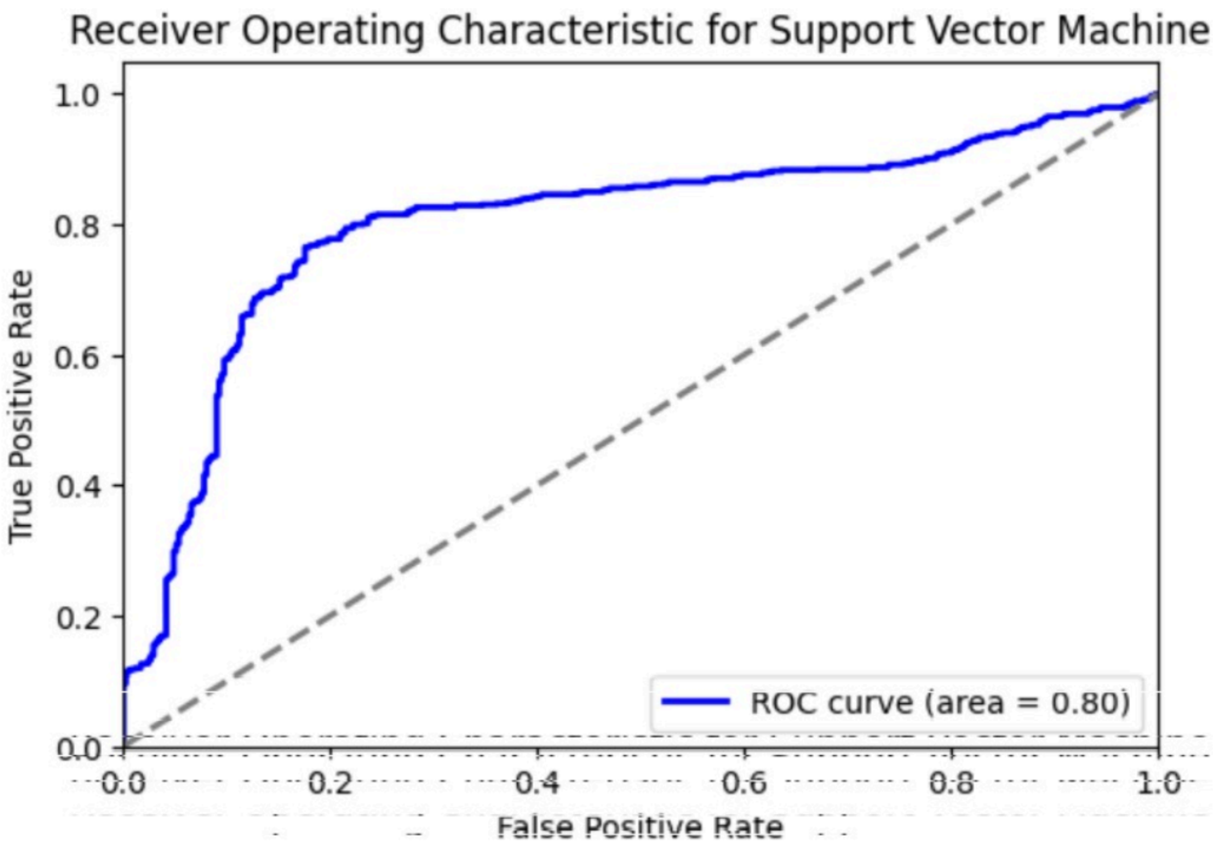




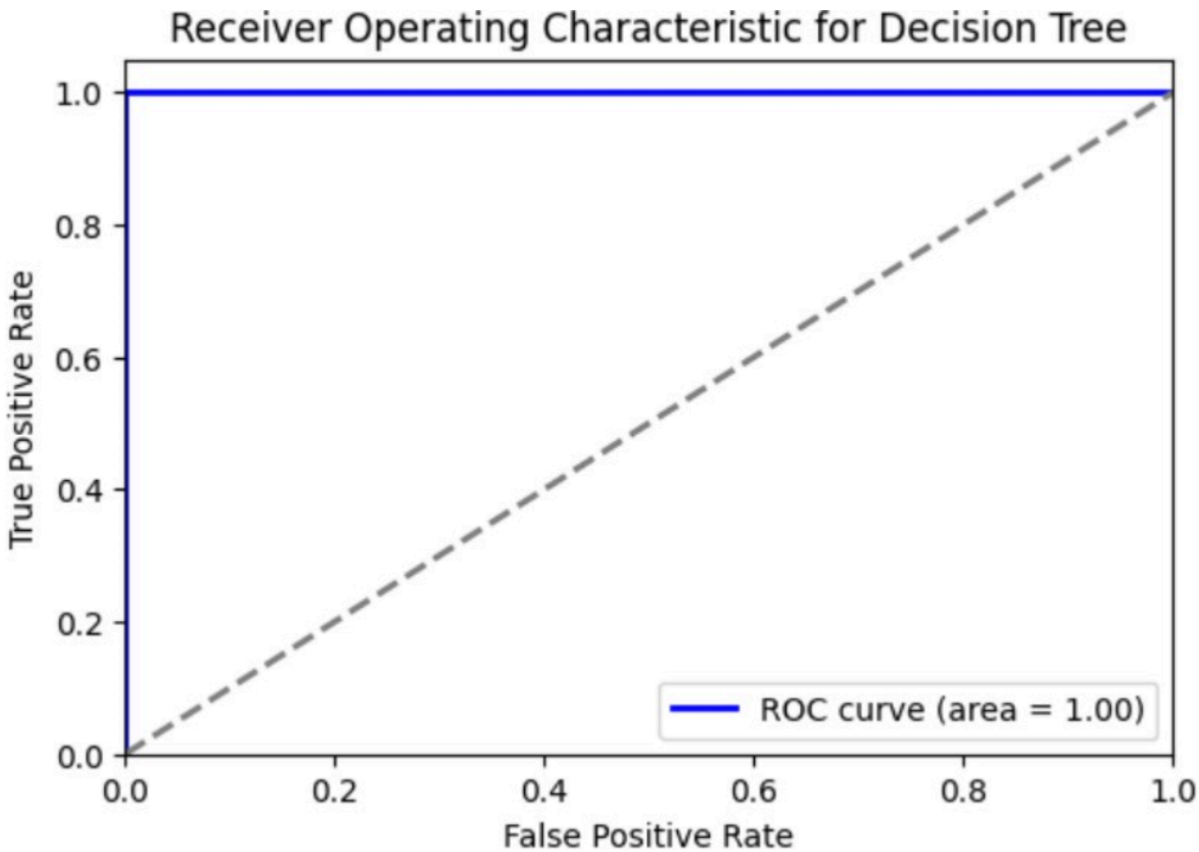
ROC CURVES:

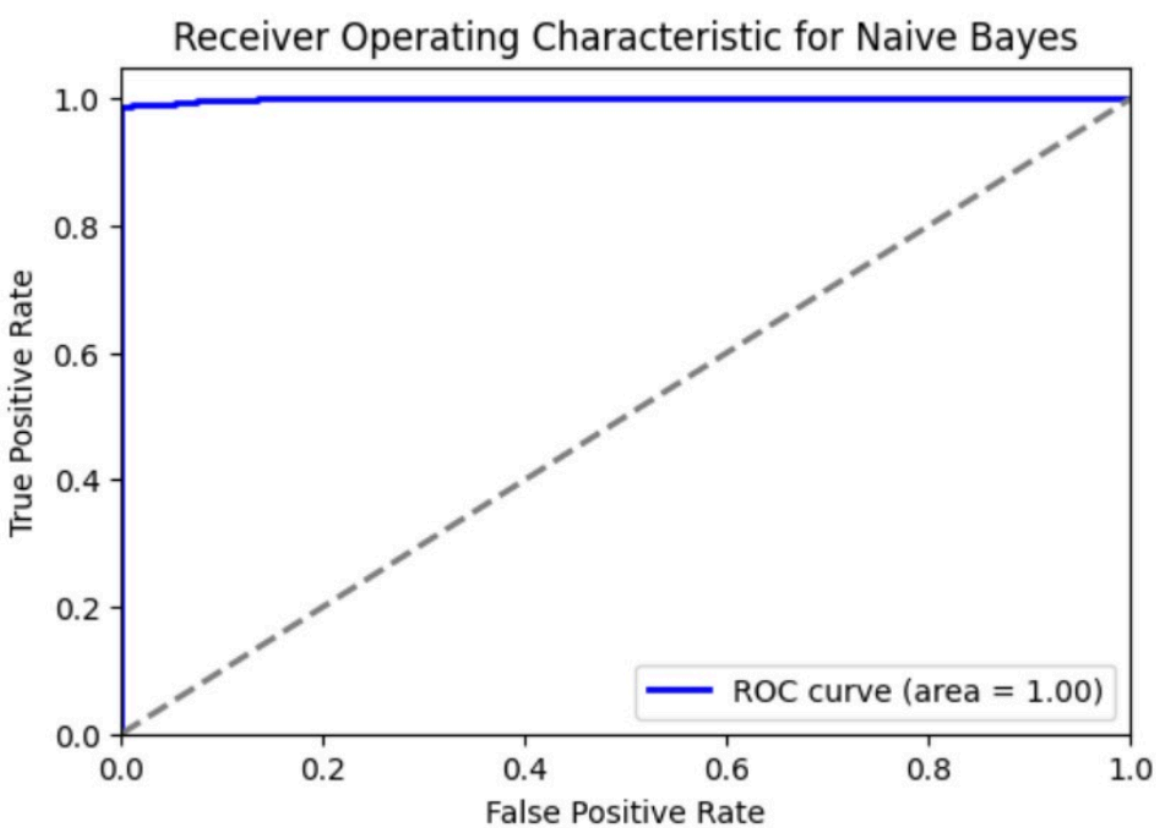


Receiver Operating Characteristic for K-Nearest Neighbors
True Positive Rate
False Positive Rate
ROC curve (area = 0.82)
0.0
0.2
0.4
0.6
0.8
1.0

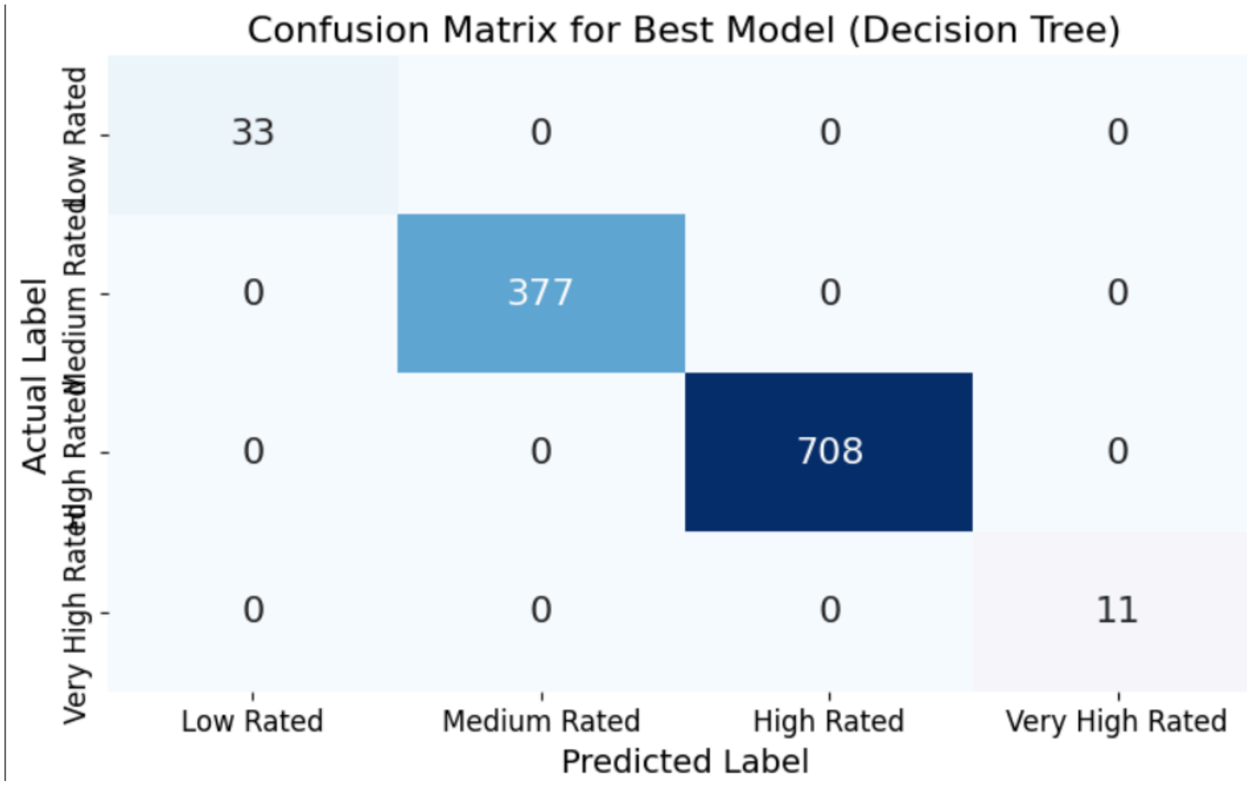


Receiver Operating Characteristic for Support Vector Machine
True Positive Rate
False Positive Rate
ROC curve (area = 0.80)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic for Decision Tree
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0


Receiver Operating Characteristic for Naive Bayes
True Positive Rate
False Positive Rate
ROC curve (area = 1.00)
0.0
0.2
0.4
0.6
0.8
1.0

BEST MODEL's CONFUSION MATRIX:

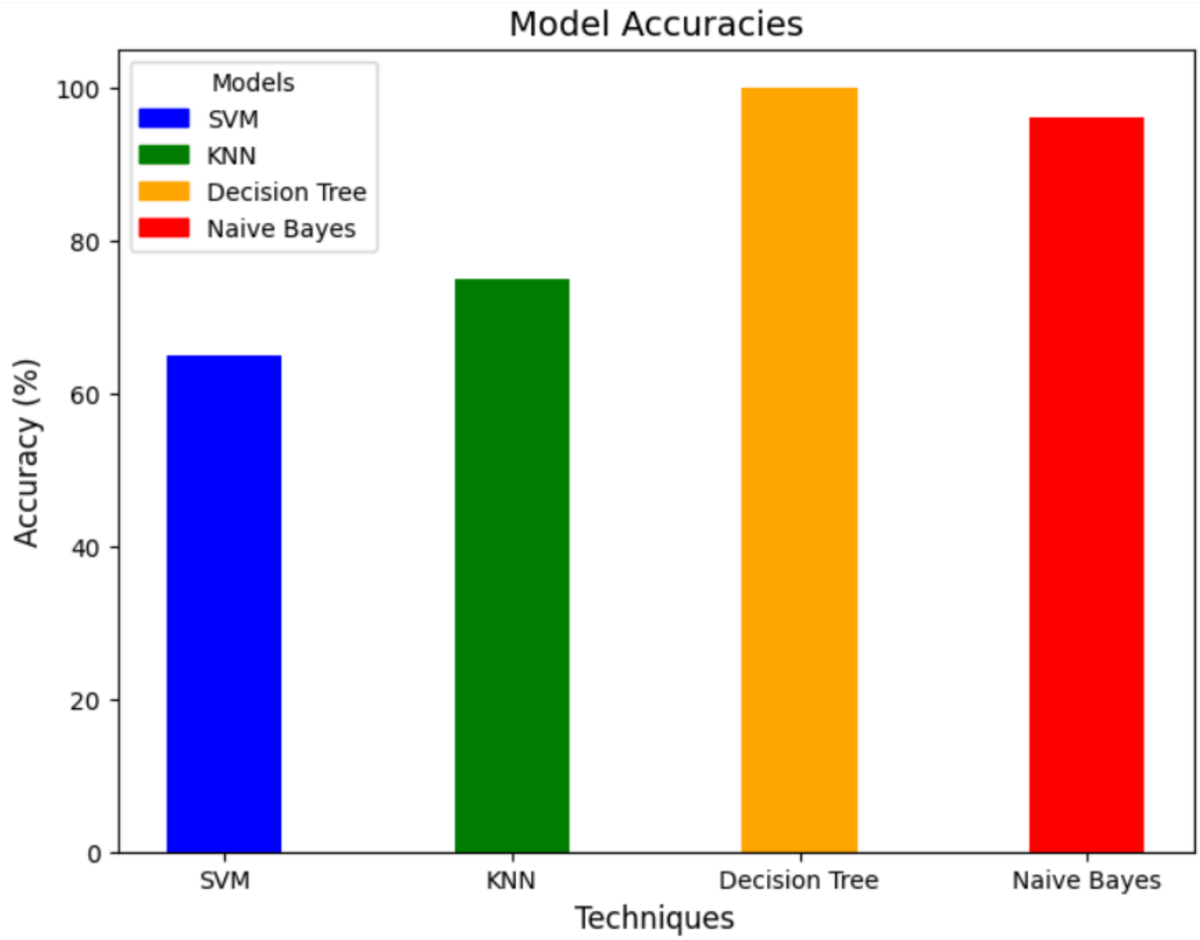


Code for Bar Chart Representation of Model Accuracies:

```
import matplotlib.pyplot as plt
# Techniques and their respective accuracies
techniques = ['SVM', 'KNN', 'Decision Tree', 'Naive Bayes']
accuracies = [65, 75, 100, 96]


# Create the bar chart with narrower bars
plt.figure(figsize=(8, 6))
bars = plt.bar(techniques, accuracies, color=['blue', 'green', 'orange', 'red'], width=0.4)


# Add title and labels
plt.title('Model Accuracies', fontsize=14)
plt.xlabel('Techniques', fontsize=12)
plt.ylabel('Accuracy (%)', fontsize=12)
```

```
# Adding a legend for the bars
colors = {'SVM': 'blue', 'KNN': 'green', 'Decision Tree': 'orange', 'Naive Bayes': 'red'}
legend_labels = [plt.Rectangle((0,0),1,1, color=colors[technique]) for technique in techniques]
plt.legend(legend_labels, techniques, title="Models", loc="upper left")

# Display the bar chart
plt.show()
```

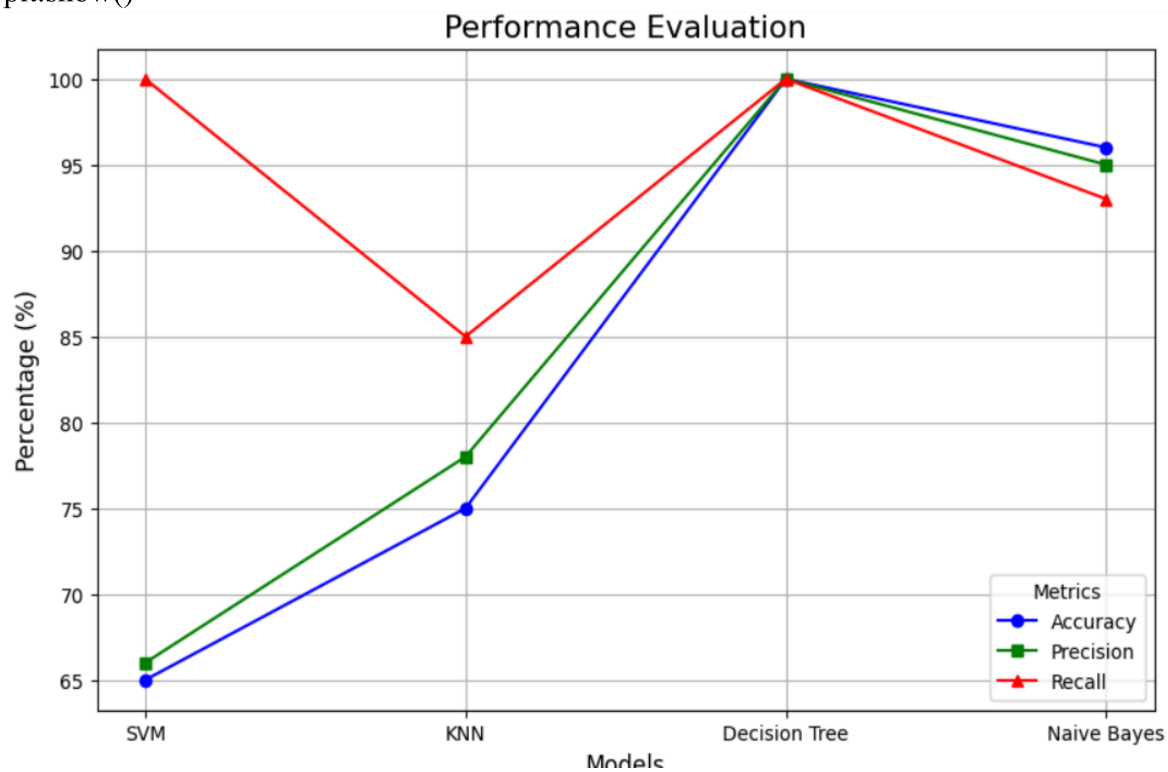
Barchart:



Code for Line Graph of Model Performance Metrics: Accuracy, Precision, and Recall:

```
import matplotlib.pyplot as plt
# Data for the models and their performance metrics
models = ['SVM', 'KNN', 'Decision Tree', 'Naive Bayes']
accuracy = [65, 75, 100, 96]
precision = [66, 78, 100, 95]
recall = [100, 85, 100, 93]
# Create the line graph
plt.figure(figsize=(10, 6))
```

```
# Plotting each metric with different colors and markers
plt.plot(models, accuracy, color='blue', marker='o', label='Accuracy')
plt.plot(models, precision, color='green', marker='s', label='Precision')
plt.plot(models, recall, color='red', marker='^', label='Recall')

# Add title and labels
plt.title('Performance Evaluation', fontsize=16)
plt.xlabel('Models', fontsize=12)
plt.ylabel('Percentage (%)', fontsize=12)

# Adding grid and legend
plt.grid(True)
plt.legend(title="Metrics", loc="lower right")

# Display the line graph
plt.show()
```



RESULT:

The proposed System successfully developed a streaming show recommendation system, achieving high accuracy across multiple models, with the Decision Tree model performing at 100%. Users receive tailored recommendations based on their age and preferences, enhanced by an interactive Streamlit interface.