NETFLIX MACHINE LEARNING NOTEBOOK:

In [58]:

```
# ========================================
#  Netflix Dataset Cleaning (For Recommender System)
# ========================================

import pandas as pd

# Step 1: Load the dataset
df = pd.read_csv("netflix_titles.csv")

# Step 2: Drop rows with missing key info (title, description, type)
df.dropna(subset=['title', 'description', 'type'], inplace=True)

# Step 3: Fill other columns with empty strings (for concatenation safety)
for col in ['director', 'cast', 'country', 'listed_in', 'rating', 'duration']:
    df[col] = df[col].fillna('')

# Step 4: Clean up and standardize text fields
def clean_text(x):
    if isinstance(x, list):
        return ' '.join(x)
    elif isinstance(x, str):
        return x.replace(',', ' ').replace('&', 'and').replace('-', ' ').strip()
    else:
        return ''

df['director'] = df['director'].apply(clean_text)
df['cast'] = df['cast'].apply(clean_text)
df['country'] = df['country'].apply(clean_text)
df['listed_in'] = df['listed_in'].apply(clean_text)
df['description'] = df['description'].apply(clean_text)
df['rating'] = df['rating'].apply(clean_text)
df['duration'] = df['duration'].apply(clean_text)
df['type'] = df['type'].apply(clean_text)

# Step 5: Ensure unique titles (since duplicates break recommender)
df = df.drop_duplicates(subset='title').reset_index(drop=True)

# Step 6: Combine features into one text column for TF-IDF
df['combined_features'] = (
    df['title'].astype(str) + " " +
    df['director'].astype(str) + " " +
    df['cast'].astype(str) + " " +
    df['country'].astype(str) + " " +
    df['listed_in'].astype(str) + " " +
    df['description'].astype(str) + " " +
    df['type'].astype(str)
)

# Step 7: Final cleanup — remove extra spaces
df['combined_features'] = df['combined_features'].str.replace(r'\s+', ' ', regex=True).s
tr.strip()

# Step 8: Confirm results
print(" Cleaning complete!")
print("Total titles:", len(df))
print("Missing values per column:\n", df.isna().sum())
df.head(10)
```

```
 Cleaning complete!
Total titles: 6172
Missing values per column:
 show_id               0
type                  0
title                 0
```

```
title                 0
director              0
cast                  0
country               0
date_added           10
release_year          0
rating                0
duration              0
listed_in             0
description           0
combined_features     0
dtype: int64
```

Out[58]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in
0	81145628	Movie	Norm of the North: King Sized Adventure	Richard Finn Tim Maltby	Alan Marriott Andrew Toth Brian Dobson Cole...	United States India South Korea China	September 9, 2019	2019	TV PG	90 min	Children and Family Movies Comedies
1	80117401	Movie	Jandino: Whatever it Takes		Jandino Asporaat	United Kingdom	September 9, 2016	2016	TV MA	94 min	Stand Up Comedy
2	70234439	TV Show	Transformers Prime		Peter Cullen Sumalee Montano Frank Welker J...	United States	September 8, 2018	2013	TV Y7 FV	1 Season	Kids' TV
3	80058654	TV Show	Transformers: Robots in Disguise		Will Friedle Darren Criss Constance Zimmer ...	United States	September 8, 2018	2016	TV Y7	1 Season	Kids' TV
4	80125979	Movie	#realityhigh	Fernando Lebrija	Nesta Cooper Kate Walsh John Michael Higgins...	United States	September 8, 2017	2017	TV 14	99 min	Comedies
5	80163890	TV Show	Apaches		Alberto Ammann Eloy Azorín Verónica Echegui ...	Spain	September 8, 2017	2016	TV MA	1 Season	Crime TV Shows International TV Shows Spanis...
6	70304989	Movie	Automata	Gabe Ibáñez	Antonio Banderas Dylan McDermott Melanie Gri...	Bulgaria United States Spain Canada	September 8, 2017	2014	R	110 min	International Movies Sci Fi and Fantasy Thri...
7	80164077	Movie	Fabrizio Copano: Solo pienso en mi	Rodrigo Toro Francisco Schultz	Fabrizio Copano	Chile	September 8, 2017	2017	TV MA	60 min	Stand Up Comedy
8	80117902	TV Show	Fire Chasers			United States	September 8, 2017	2017	TV MA	1 Season	Docuseries Science and Nature TV
					James						

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in
9	70304990	Movie	Good People	Henrik Ruben Genz	Kate Hudson, Tom Wilkinson, Omar...	United States, United Kingdom, Denmark, Sweden	September 8, 2017	2014	R	90 min	Action and Adventure, Thrillers

In [59]:

```
# Dictionary to store duplicate counts for each column
duplicate_counts = {}

# Iterate through each column
for column in df.columns:
    # Use duplicated() to find duplicate values (excluding the first occurrence)
    # and sum them to get the count
    num_duplicates = df[column].duplicated().sum()
    duplicate_counts[column] = num_duplicates

# Print the results
print("Number of duplicate values in each column:")
for col, count in duplicate_counts.items():
    print(f"Column '{col}': {count} duplicates")
```

```
Number of duplicate values in each column:
Column 'show_id': 0 duplicates
Column 'type': 6170 duplicates
Column 'title': 0 duplicates
Column 'director': 2894 duplicates
Column 'cast': 759 duplicates
Column 'country': 5624 duplicates
Column 'date_added': 4655 duplicates
Column 'release_year': 6100 duplicates
Column 'rating': 6157 duplicates
Column 'duration': 5971 duplicates
Column 'listed_in': 5713 duplicates
Column 'description': 7 duplicates
Column 'combined_features': 0 duplicates
```

In [60]:

```
# ========================================
#  Netflix Movie/TV Show Recommender GUI
# ========================================

import tkinter as tk
from tkinter import messagebox, ttk
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pygame

# ------------------------------
# STEP 1: Initialize pygame for sound
# ------------------------------
pygame.init()
pygame.mixer.init()

def play_music(file):
    """Play an MP3 file safely."""
    try:
        pygame.mixer.music.load(file)
        pygame.mixer.music.play()
    except Exception as e:
        print(" Sound error:", e)

# ------------------------------
# STEP 2: Prepare data for recommendation
# ------------------------------
# Ensure df is loaded from your cleaned dataset
df_rec = df[['title', 'combined_features']].dropna()
```

```
# Create TF-IDF vectorizer
tfidf = TfidfVectorizer(stop_words='english', max_features=8000)
tfidf_matrix = tfidf.fit_transform(df_rec['combined_features'])

# Build case-insensitive lookup dictionary
title_to_index = {title.lower(): idx for idx, title in enumerate(df_rec['title'])}

# ------------------------------
# STEP 3: Define recommendation function
# ------------------------------
def recommend(title, num_recommendations=5):
    title = title.strip().lower()

    # Check if the title exists
    if title not in title_to_index:
        play_music("Netflix-error.mp3")
        messagebox.showerror("Error", f"□ '{title.title()}' not found in dataset.")
        return []

    # Play intro sound
    play_music("Netflix-Intro-Sound-Effect.mp3")

    # Get index of the movie
    idx = title_to_index[title]
    movie_vector = tfidf_matrix[idx]

    # Compute similarity with all titles
    sim_scores = cosine_similarity(movie_vector, tfidf_matrix).flatten()

    # Get top similar movies
    similar_indices = sim_scores.argsort()[::-1][1:num_recommendations+1]
    recommendations = df_rec['title'].iloc[similar_indices].tolist()

    return recommendations

# ------------------------------
# STEP 4: Create GUI
# ------------------------------
root = tk.Tk()
root.title("□ Netflix Movie Recommender System")
root.geometry("650x450")
root.config(bg="#121212")

# --- Title Label ---
title_label = tk.Label(root, text="Netflix Recommender",
                       font=("Arial", 22, "bold"), fg="red", bg="#121212")
title_label.pack(pady=10)

# --- Entry Label ---
entry_label = tk.Label(root, text="Enter Movie / TV Show Title:",
                       font=("Arial", 12), fg="white", bg="#121212")
entry_label.pack(pady=5)

# --- Input Box ---
entry_box = tk.Entry(root, font=("Arial", 13), width=45)
entry_box.pack(pady=8)

# --- Result Treeview ---
tree = ttk.Treeview(root, columns=("Recommendations"), show='headings', height=7)
tree.heading("Recommendations", text="Recommended Titles")
tree.pack(pady=10)

# --- Function to display recommendations ---
def show_recommendations():
    tree.delete(*tree.get_children())
    movie_name = entry_box.get().strip()

    if not movie_name:
        messagebox.showwarning("Input Error", "Please enter a movie or show title.")
        return
```

```
    recs = recommend(movie_name)

    if recs:
        for r in recs:
            tree.insert("", "end", values=(r,))
    else:
        tree.insert("", "end", values=("No recommendations found.",))

# --- Recommendation Button ---
recommend_button = tk.Button(root, text="Get Recommendations □",
                             font=("Arial", 12, "bold"),
                             bg="red", fg="white",
                             padx=12, pady=6,
                             command=show_recommendations)
recommend_button.pack(pady=10)

# --- Footer ---
footer = tk.Label(root, text="Developed by Om",
                  font=("Arial", 9), fg="gray", bg="#121212")
footer.pack(side="bottom", pady=8)

# -------------------------------
# STEP 5: Run the GUI
# -------------------------------
root.mainloop()
```

In [ ]:

```
```

In [ ]:

```
```

In [ ]:

```
```

In [61]:

```
# ========================================
# □ Netflix Movie or TV Show Predictor (Fixed Version)
# ========================================

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report
from sklearn.utils import resample

# --- Step □: Prepare data ---
df_pred = df[['type', 'country', 'listed_in', 'rating', 'description']].dropna().copy()

# --- Step □: Encode categorical columns ---
label_encoders = {}
for col in ['country', 'listed_in', 'rating']:
    le = LabelEncoder()
    df_pred[col] = le.fit_transform(df_pred[col].astype(str))
    label_encoders[col] = le

# Encode target (Movie / TV Show)
target_encoder = LabelEncoder()
df_pred['type'] = target_encoder.fit_transform(df_pred['type'])  # Movie=0, TV Show=1

# --- Step □: Text Vectorization (TF-IDF on description) ---
tfidf = TfidfVectorizer(stop_words='english', max_features=500)
desc_features = tfidf.fit_transform(df_pred['description']).toarray()

# --- Step □: Combine categorical + text features ---
```

```
X_cat = df_pred[['country', 'listed_in', 'rating']].values
X = np.hstack([X_cat, desc_features])
y = df_pred['type']

# --- Step 5: Check imbalance ---
print("Before balancing:\n", pd.Series(y).value_counts())

# --- Step 6: Balance the dataset ---
df_temp = df_pred[['country', 'listed_in', 'rating', 'description', 'type']].copy()
df_temp['desc_vector'] = list(desc_features)

# Separate classes
df_movies = df_temp[df_temp['type'] == target_encoder.transform(['Movie'])[0]]
df_tv = df_temp[df_temp['type'] == target_encoder.transform(['TV Show'])[0]]

# Downsample majority (Movies)
df_movies_down = resample(df_movies, replace=False, n_samples=len(df_tv), random_state=4
2)

# Combine balanced dataset
df_balanced = pd.concat([df_movies_down, df_tv]).sample(frac=1, random_state=42).reset_i
ndex(drop=True)
print("\nAfter balancing:\n", df_balanced['type'].value_counts())

# Extract X, y again
X_cat_bal = df_balanced[['country', 'listed_in', 'rating']].values
desc_vectors_bal = np.vstack(df_balanced['desc_vector'])
X_bal = np.hstack([X_cat_bal, desc_vectors_bal])
y_bal = df_balanced['type']

# --- Step 7: Split dataset ---
X_train, X_test, y_train, y_test = train_test_split(
    X_bal, y_bal, test_size=0.2, random_state=42, stratify=y_bal
)

# --- Step 8: Train Random Forest with class_weight ---
model = RandomForestClassifier(n_estimators=150, random_state=42, class_weight='balanced
')
model.fit(X_train, y_train)

# --- Step 9: Evaluate model ---
y_pred = model.predict(X_test)
print("\n Model Accuracy:", round(accuracy_score(y_test, y_pred) * 100, 2), "%")
print("\n Classification Report:\n", classification_report(
    y_test, y_pred, target_names=target_encoder.classes_
))

# --- Step : Predict for a new example ---
new_entry = {
    'country': 'China',
    'listed_in': 'International TV Shows',
    'rating': 'TV-PG',
    'description': 'A gripping drama series full of suspense and mystery.'
}

new_df = pd.DataFrame([new_entry])

# --- Step 10: Encode new input safely ---
for col, le in label_encoders.items():
    val = new_df.at[0, col]
    if val in le.classes_:
        new_df[col] = le.transform([val])
    else:
        # Handle unseen category safely by extending encoder
        print(f"⚠ Warning: '{val}' unseen in column '{col}'. Using fallback handling.")
        le.classes_ = np.append(le.classes_, val)
        new_df[col] = le.transform([val])

# --- Step 11: TF-IDF transform for description ---
new_desc = tfidf.transform(new_df['description']).toarray()

# --- Step 12: Combine all features ---
```

```
X_new = np.hstack([
    new_df[['country', 'listed_in', 'rating']].values,
    new_desc
])

# --- Step 10: Predict and decode result ---
prediction = model.predict(X_new)[0]
predicted_label = target_encoder.inverse_transform([prediction])[0]

print(f"\n Predicted Type for given entry: {predicted_label}")
```

```
Before balancing:
 type
0    4233
1    1939
Name: count, dtype: int64

After balancing:
 type
1    1939
0    1939
Name: count, dtype: int64

 Model Accuracy: 92.65 %

 Classification Report:
              precision    recall  f1-score   support

       Movie       0.92      0.93      0.93       388
     TV Show       0.93      0.92      0.93       388

    accuracy                           0.93       776
   macro avg       0.93      0.93      0.93       776
weighted avg       0.93      0.93      0.93       776

⚠ Warning: 'TV-PG' unseen in column 'rating'. Using fallback handling.

 Predicted Type for given entry: TV Show
```

In [ ]: