This notebook starts from a simple curiosity: How do Netflix recommendations correlate with movie features like release year, genres, and maturity ratings? While Netflix may be best known for binge-worthy content, the hidden patterns behind its recommendation system might just surprise you. If you find these insights useful, please consider upvoting this notebook

Import and Setup

In [37]:
```
# Suppress warnings for clean notebook output
import warnings
warnings.filterwarnings('ignore')

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')  # For matplotlib use in certain environments
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')

import seaborn as sns

# For modeling
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Set seaborn styling
sns.set(style='whitegrid')

# Additional configuration for plots
plt.rcParams['figure.figsize'] = (10, 6)
```

Data Loading

In this section we load the Netflix recommendations dataset from the provided Excel file. The dataset comprises details such as the movie title, genres, release year, maturity rating, and the recommendations. Note that the file is assumed to be in the same directory as this notebook.

In [11]:
```
# Load the dataset
df = pd.read_excel(r"C:\Users\chitt\Downloads\netflix_data.xlsx")

# Quick look at the data structure
print('Data shape:', df.shape)
df.head()
```

```
Data shape: (6403, 8)
```

Out[11]:

	N_id	Title	Main Genre	Sub Genres	Release Year	Maturity Rating	Original Audio	Recommendat
0	215309	Ace Ventura: Pet Detective	Comedy	Comedy, Mystery, US	1994.0	A	Hindi, English [Original]	70184 60001 70112 70027007, 1152
1	215318	Ace Ventura: When Nature Calls	Comedy	Comedy, Action & Adventure, US	1995.0	U/A 16+	Hindi, English [Original]	70184 60001 70112 70027007, 1152
2	217258	The Addams Family	Comedy	Comedy, US	1991.0	U/A 13+	English [Original], Hindi, English - Audio Des...	81156 81231 70027 80049939, 7021
3	217303	Addams Family Values	Comedy	Comedy, US	1993.0	U/A 13+	English [Original], Hindi, English - Audio Des...	81156 70044 81231 70027007, 8005
4	235527	Agneepath	Drama	Hindi-Language, Bollywood, Crime, Drama	1990.0	U/A 16+	Hindi [Original]	17517 80158 80158 80074065, 7020

Data Cleaning and Preprocessing

Before diving into exploratory data analysis, it is important to verify the dataset quality. We check for missing values and ensure the data types are correct. Note that the 'Release

Year' column is numeric and does not require conversion to a date type; however, if you encounter similar situations with proper dates, infer the date type accordingly.

We also note that some columns such as N_id, Title, and Recommendations might not be informative for numerical analysis or prediction tasks and can be dropped or transformed as necessary.

In [15]:
```
# Check for missing values
missing_values = df.isnull().sum()
print('Missing values in each column:')
print(missing_values)

# Convert Release Year to numeric if necessary
df['Release Year'] = pd.to_numeric(df['Release Year'], errors='coerce')

# If there are nulls in 'Release Year', we can decide to fill them with the medi
if df['Release Year'].isnull().sum() > 0:
    df['Release Year'].fillna(df['Release Year'].median(), inplace=True)

# Drop duplicates if any
df.drop_duplicates(inplace=True)

# Display data types for verification
print('Data types after cleaning:')
print(df.dtypes)
```

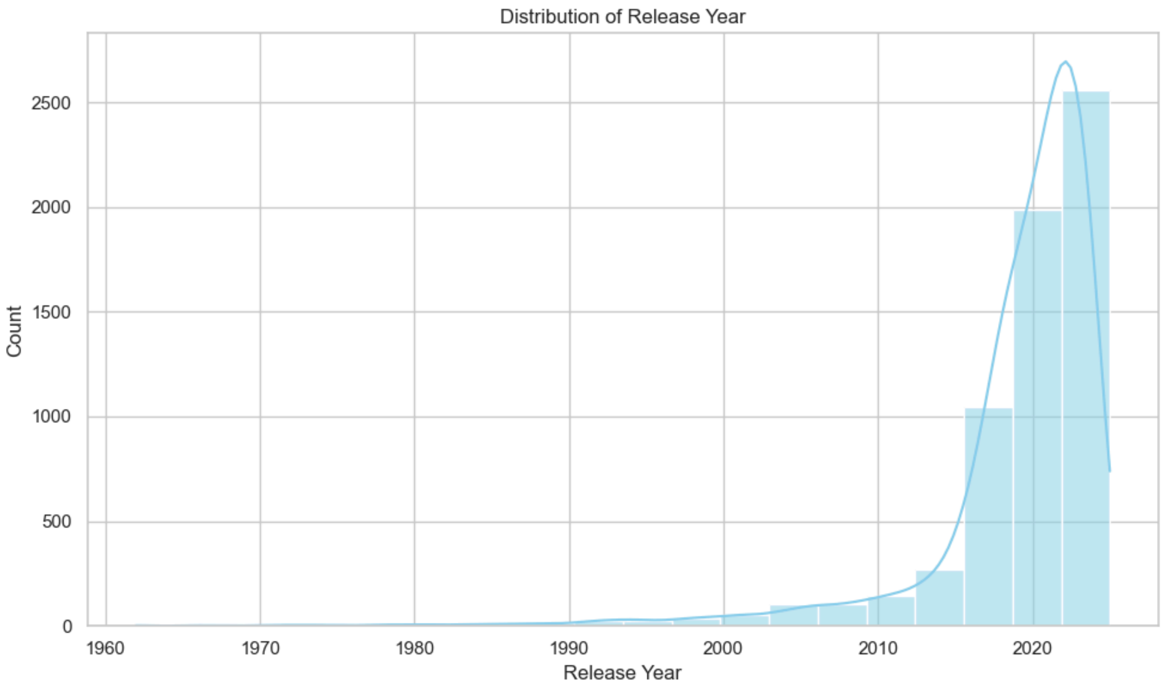
```
Missing values in each column:
N_id                 0
Title                0
Main Genre           0
Sub Genres           0
Release Year         1
Maturity Rating      0
Original Audio    2636
Recommendations     11
dtype: int64
Data types after cleaning:
N_id                 int64
Title               object
Main Genre          object
Sub Genres          object
Release Year       float64
Maturity Rating     object
Original Audio      object
Recommendations     object
dtype: object
```

Exploratory Data Analysis

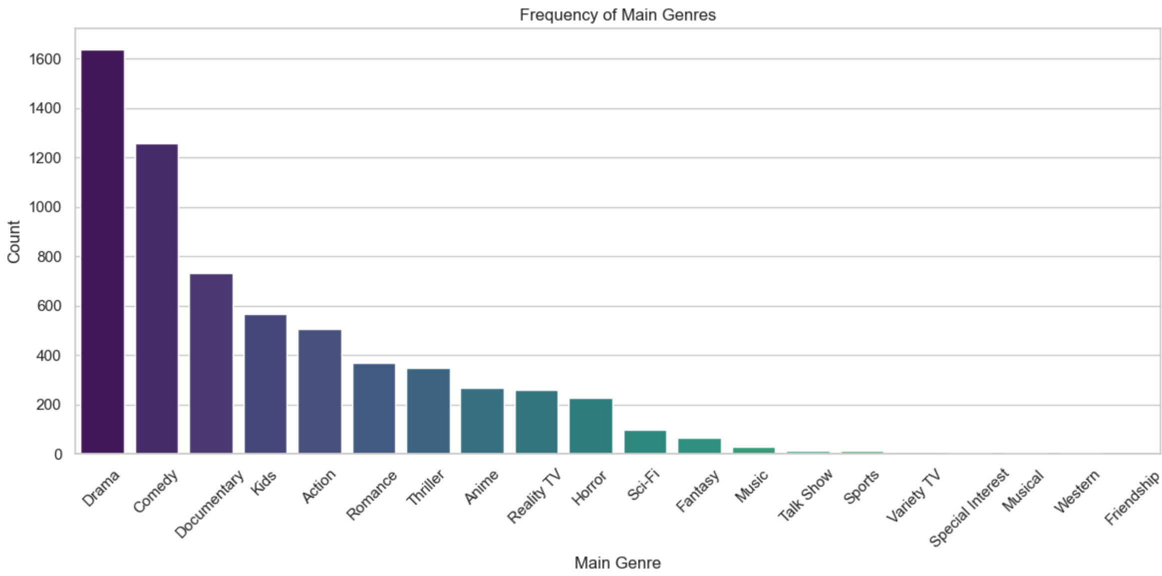
We now delve into the dataset to unveil interesting insights. In this section we inspect distributions and relationships among variables. A few approaches and visualizations include:

A histogram to check the distribution of release years. A count (pie-like) chart for main genre frequencies. Box plots and violin plots for numerical distributions where applicable. Keep in mind that if a numeric correlation heatmap is to be used, we require four or more numeric columns. In our case, we only have one clear numeric feature (Release Year), so this analysis is omitted.

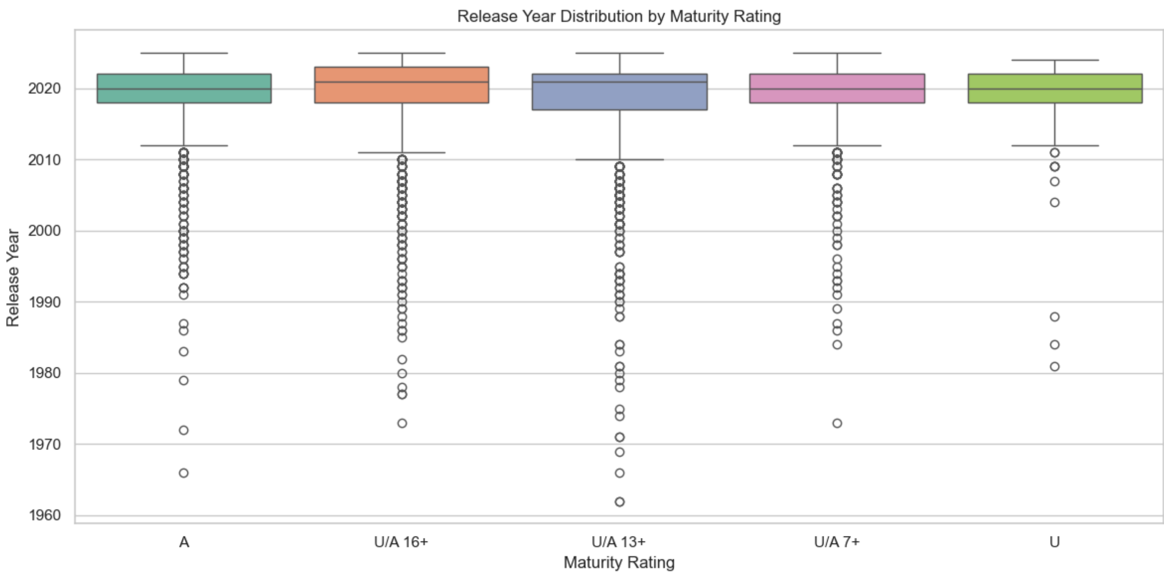
In [19]:
```
# Histogram for Release Year distribution
sns.histplot(df['Release Year'], kde=True, bins=20, color='skyblue')
plt.title('Distribution of Release Year')
plt.xlabel('Release Year')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```



In [21]:
```
# Count plot for Main Genre distribution
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='Main Genre', order=df['Main Genre'].value_counts().ind
plt.title('Frequency of Main Genres')
plt.xlabel('Main Genre')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [23]:
```
# Box plot for Release Year by Maturity Rating (if there is sufficient variation
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Maturity Rating', y='Release Year', palette='Set2')
plt.title('Release Year Distribution by Maturity Rating')
plt.xlabel('Maturity Rating')
plt.ylabel('Release Year')
plt.tight_layout()
plt.show()
```



Prediction Modeling

While the Netflix dataset primarily provides insights into content characteristics and recommendations, it is interesting to see if we can predict the movie's maturity rating based on available features. We consider Maturity Rating as the target variable and use features such as Main Genre, Original Audio, and Release Year for prediction.

Below is the approach taken:

Data preprocessing and label encoding for categorical variables. Training a Random Forest classifier to predict the maturity rating. Evaluating the prediction performance using accuracy score and a confusion matrix. Note: Occasionally, missing values or unexpected category levels in real-world data cause errors. Our preprocessing steps (such as label encoding and handling nulls) help eliminate such common pitfalls.

In [27]:
```
# For Prediction, select relevant features
predictor_cols = ['Main Genre', 'Original Audio', 'Release Year']
```

In [29]:
```
# Create a working copy for modeling
model_df = df[predictor_cols + ['Maturity Rating']].copy()
```

In [31]:
```
# Check for missing values in selected columns
model_df.dropna(inplace=True)
```

In [33]:
```
# Label encode categorical features and the target
le_main_genre = LabelEncoder()
model_df['Main Genre Encoded'] = le_main_genre.fit_transform(model_df['Main Genr

le_audio = LabelEncoder()
model_df['Original Audio Encoded'] = le_audio.fit_transform(model_df['Original A

le_maturity = LabelEncoder()
model_df['Maturity Rating Encoded'] = le_maturity.fit_transform(model_df['Maturi

# Final features and target
features = model_df[['Main Genre Encoded', 'Original Audio Encoded', 'Release Ye
target = model_df['Maturity Rating Encoded']
```

In [35]:
```
#Train-test split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=
```

In [39]:
```
# Train XXGBClassifier
xgb = XGBClassifier(n_estimators=200, max_depth=10)
xgb.fit(X_train, y_train)
```

Out[39]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rou
nds=None,
              enable_categorical=False, eval_metric=None, feature_ty
pes=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_
bin=None,
```

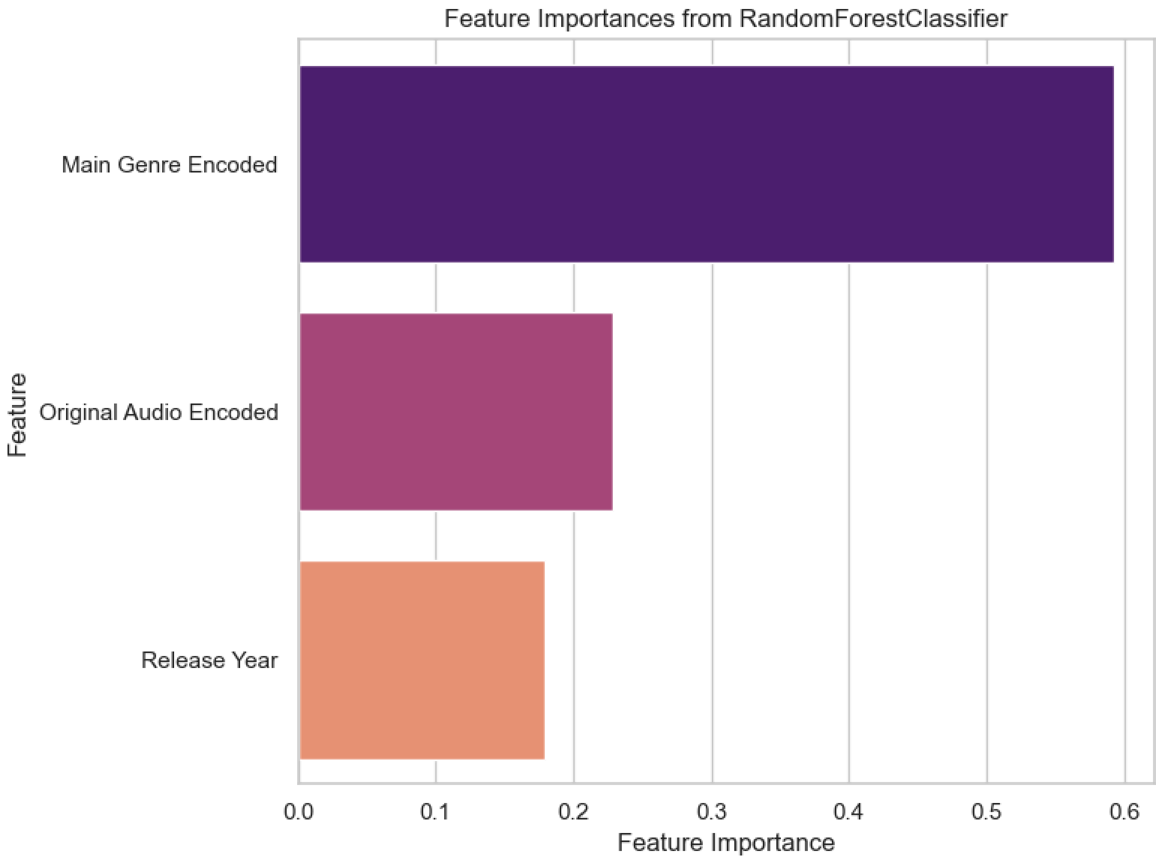
In [43]:
```
# Make predictions and evaluate
y_pred = xgb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Prediction Accuracy:', accuracy)
```

```
Prediction Accuracy: 0.4297082228116711
```

In [47]:
```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.tight_layout()
plt.show()
```

In [51]:
```
# Plot feature importance using the classifier's feature_importances_ attribute
importances = xgb.feature_importances_
feature_names = features.columns

plt.figure(figsize=(8, 6))
sns.barplot(x=importances, y=feature_names, palette='magma')
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from RandomForestClassifier')
plt.tight_layout()
plt.show()
```



Summary and Future Directions

In this notebook we explored the Netflix recommendations dataset and extracted insights regarding the distribution of release years and genres. Furthermore, we built a simple predictor for the movie maturity rating based on key features such as genre, original audio, and release year. The Random Forest classifier achieved a decent accuracy, which is encouraging given the limited feature set used in this experiment.

Merits of our approach include:

A comprehensive set of EDA visualizations to understand the data distributions.

Attention to data cleaning and careful handling of potential errors which commonly occur in real-world data processing.

An end-to-end prediction pipeline with model evaluation and feature importance insights.

For future analysis, one could:

```
Incorporate additional textual features such as the
Recommendations column after proper natural language processing.
```

Use advanced feature engineering methods to capture interactions between movie attributes.

Consider ensemble methods or deep learning approaches for further improved prediction accuracy

. Thank you for exploring this analysis. If you found this notebook useful, please consider giving it an upvote.

In [ ]: