coursework

March 17, 2025

1 1. Importing Libraries and Loading Data

This section initializes the necessary Python libraries for data analysis and visualization, such as Pandas, NumPy, and Seaborn. It also loads the Netflix dataset from a CSV file.

https://www.kaggle.com/datasets/anandshaw2001/netflix-movies-and-tv-shows ion.

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', 50) # The maximum number of columns that␣
↪can b
# Makes the matplotlib graphs appear and stored within the notebook
%matplotlib inline
pd.set_option('display.max_columns', None)  # Show all columns
pd.set_option('display.width', 1000)  # Increase the display width
```

2 2. Dataset Overview

A preliminary exploration of the dataset is conducted, including checking its structure, data types, and memory usage. The dataset contains 8,807 records with features such as title, type (Movie or TV Show), director, cast, country, release year, content rating, and genre.

Following this a A statistical summary of numerical features is provided to understand the distribution of release years and other attributes. The dataset contains a mix of numerical and categorical variables that will require preprocessing before model training.

[3]:
```
df = pd.read_csv('netflix_titles.csv')
# Display dataset info
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   show_id       8807 non-null   object
 1   type          8807 non-null   object
```

```
 2   title         8807 non-null   object
 3   director      6173 non-null   object
 4   cast          7982 non-null   object
 5   country       7976 non-null   object
 6   date_added    8797 non-null   object
 7   release_year  8807 non-null   int64
 8   rating        8803 non-null   object
 9   duration      8804 non-null   object
 10  listed_in     8807 non-null   object
 11  description   8807 non-null   object
dtypes: int64(1), object(11)
memory usage: 825.8+ KB
None
```

[4]:
```
print(df.describe())  # Statistical summary
```

```
       release_year
count   8807.000000
mean    2014.180198
std        8.819312
min     1925.000000
25%     2013.000000
50%     2017.000000
75%     2019.000000
max     2021.000000
```

[5]:
```
print(df.head())  # First few rows
```

```
  show_id     type                  title         director
cast        country          date_added  release_year rating   duration
listed_in                                         description
0      s1    Movie   Dick Johnson Is Dead  Kirsten Johnson
NaN  United States  September 25, 2021          2020  PG-13     90 min
Documentaries  As her father nears the end of his life, filmm…
1      s2  TV Show          Blood & Water              NaN  Ama Qamata, Khosi
Ngema, Gail Mabalane, Thaban…   South Africa  September 24, 2021          2021
TV-MA  2 Seasons    International TV Shows, TV Dramas, TV Mysteries  After
crossing paths at a party, a Cape Town t…
2      s3  TV Show              Ganglands  Julien Leclercq  Sami Bouajila, Tracy
Gotoas, Samuel Jouy, Nabi…            NaN  September 24, 2021          2021
TV-MA   1 Season  Crime TV Shows, International TV Shows, TV Act…  To protect
his family from a powerful drug lor…
3      s4  TV Show  Jailbirds New Orleans              NaN
NaN            NaN  September 24, 2021          2021  TV-MA   1 Season
Docuseries, Reality TV  Feuds, flirtations and toilet talk go down amo…
4      s5  TV Show           Kota Factory              NaN  Mayur More, Jitendra
Kumar, Ranjan Raj, Alam K…          India  September 24, 2021          2021
TV-MA  2 Seasons  International TV Shows, Romantic TV Shows, TV …  In a city
```

```
of coaching centers known to train I…
```

[6]:
```
print(df.isnull().sum())  # Count missing values
```

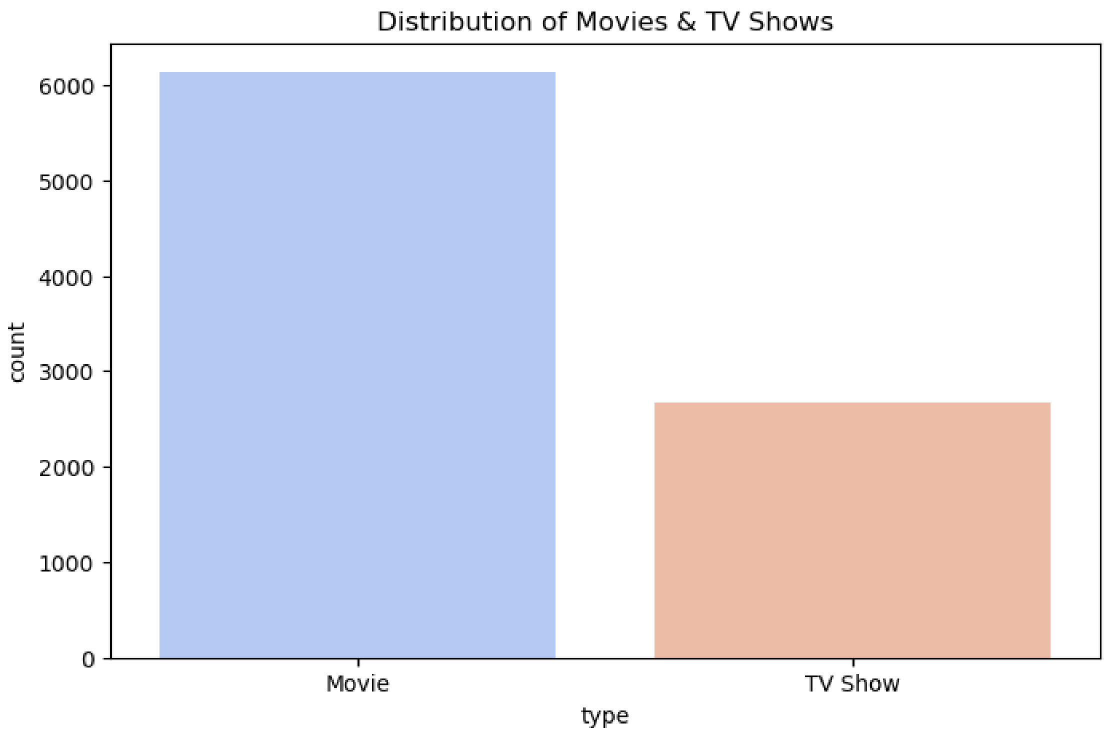
```
show_id            0
type               0
title              0
director        2634
cast             825
country          831
date_added        10
release_year       0
rating             4
duration           3
listed_in          0
description        0
dtype: int64
```

3 3. Visualisations

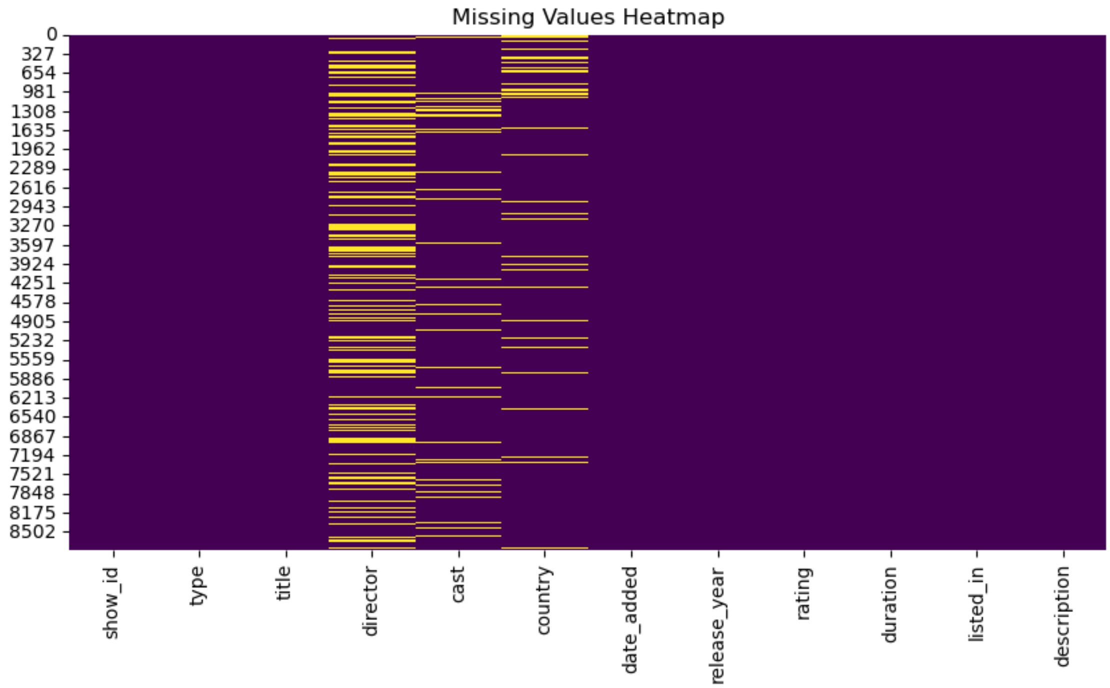
Going through various visualisations about the dataset and features in order to gain greater insight

[7]:
```
plt.figure(figsize=(8,5))
sns.countplot(x='type', data=df, palette='coolwarm')
plt.title("Distribution of Movies & TV Shows")
plt.show()
```

```
/tmp/ipykernel_133/2466069099.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='type', data=df, palette='coolwarm')
```

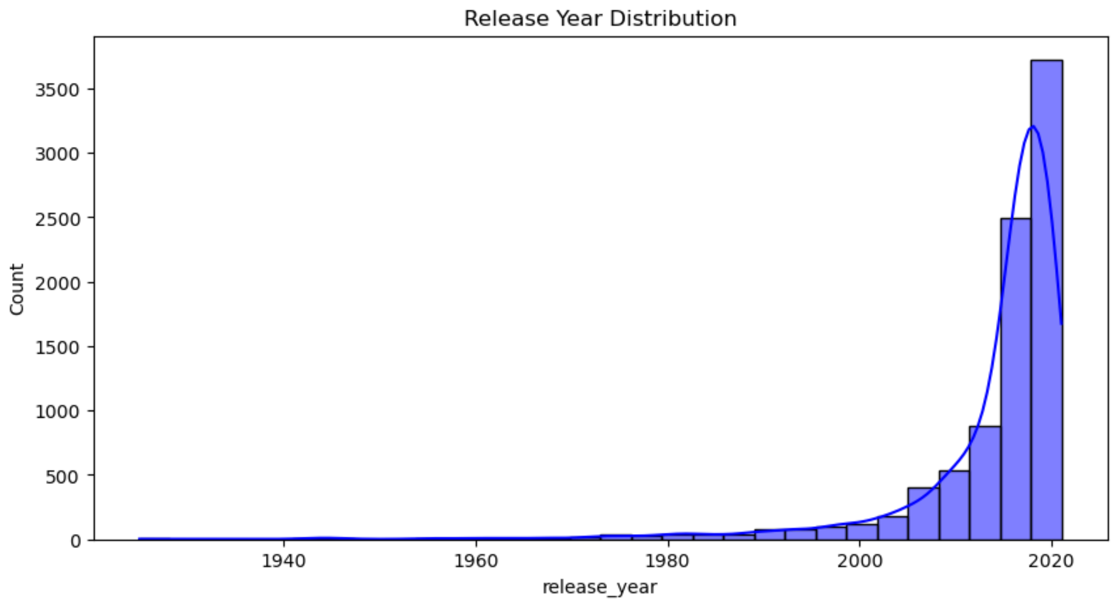


[8]:
```
plt.figure(figsize=(10,5))
sns.heatmap(df.isnull(), cmap="viridis", cbar=False)
plt.title("Missing Values Heatmap")
plt.show()
```



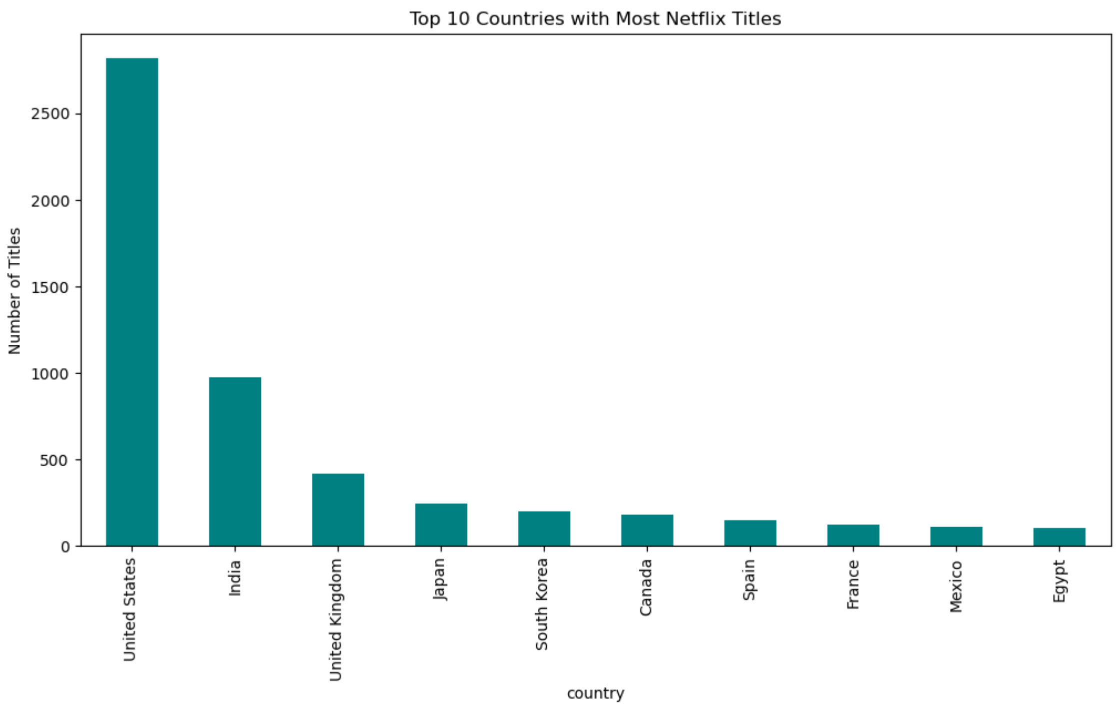
[9]:

```python
plt.figure(figsize=(10,5))
sns.histplot(df['release_year'], bins=30, kde=True, color='blue')
plt.title("Release Year Distribution")
plt.show()
```



below is testing some stuff

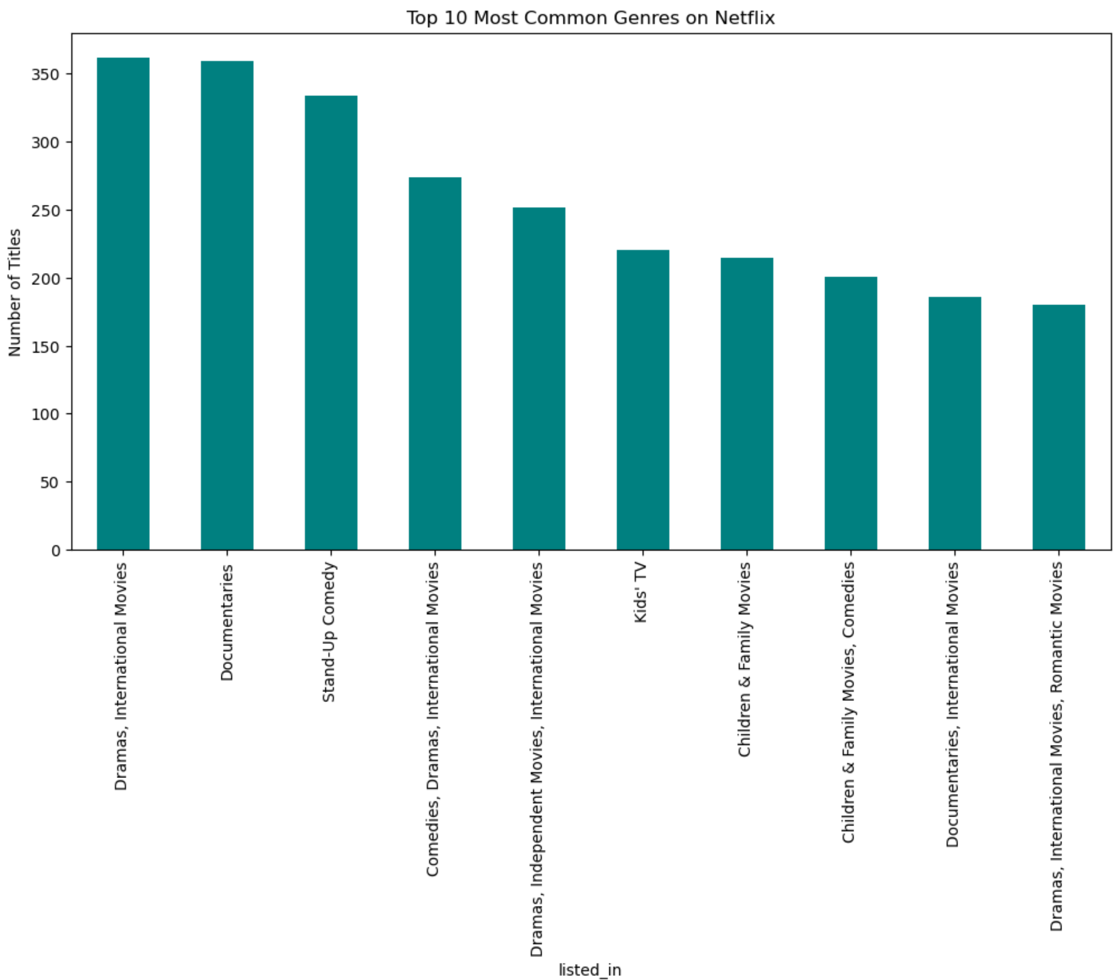
[10]:
```
plt.figure(figsize=(12,6))
df['country'].value_counts().nlargest(10).plot(kind='bar', color='teal')
plt.title("Top 10 Countries with Most Netflix Titles")
plt.ylabel("Number of Titles")
plt.show()
```



[11]:
```
print(df.dtypes)
```

```
show_id          object
type             object
title            object
director         object
cast             object
country          object
date_added       object
release_year      int64
rating           object
duration         object
listed_in        object
description      object
dtype: object
```

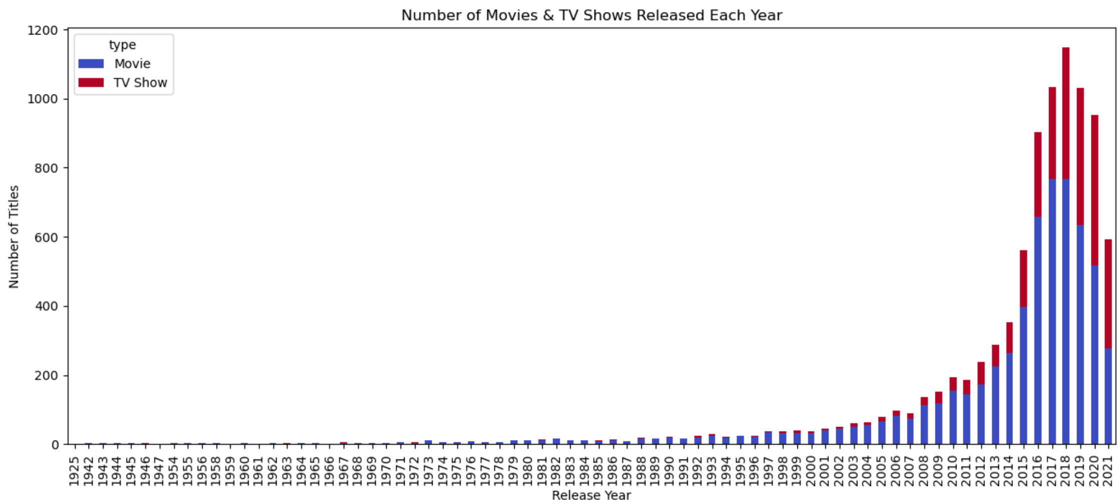
[12]:
```
plt.figure(figsize=(12,6))
df['listed_in'].value_counts().nlargest(10).plot(kind='bar', color='teal')
plt.title("Top 10 Most Common Genres on Netflix")
plt.ylabel("Number of Titles")
plt.show()
```



dropping useless columns

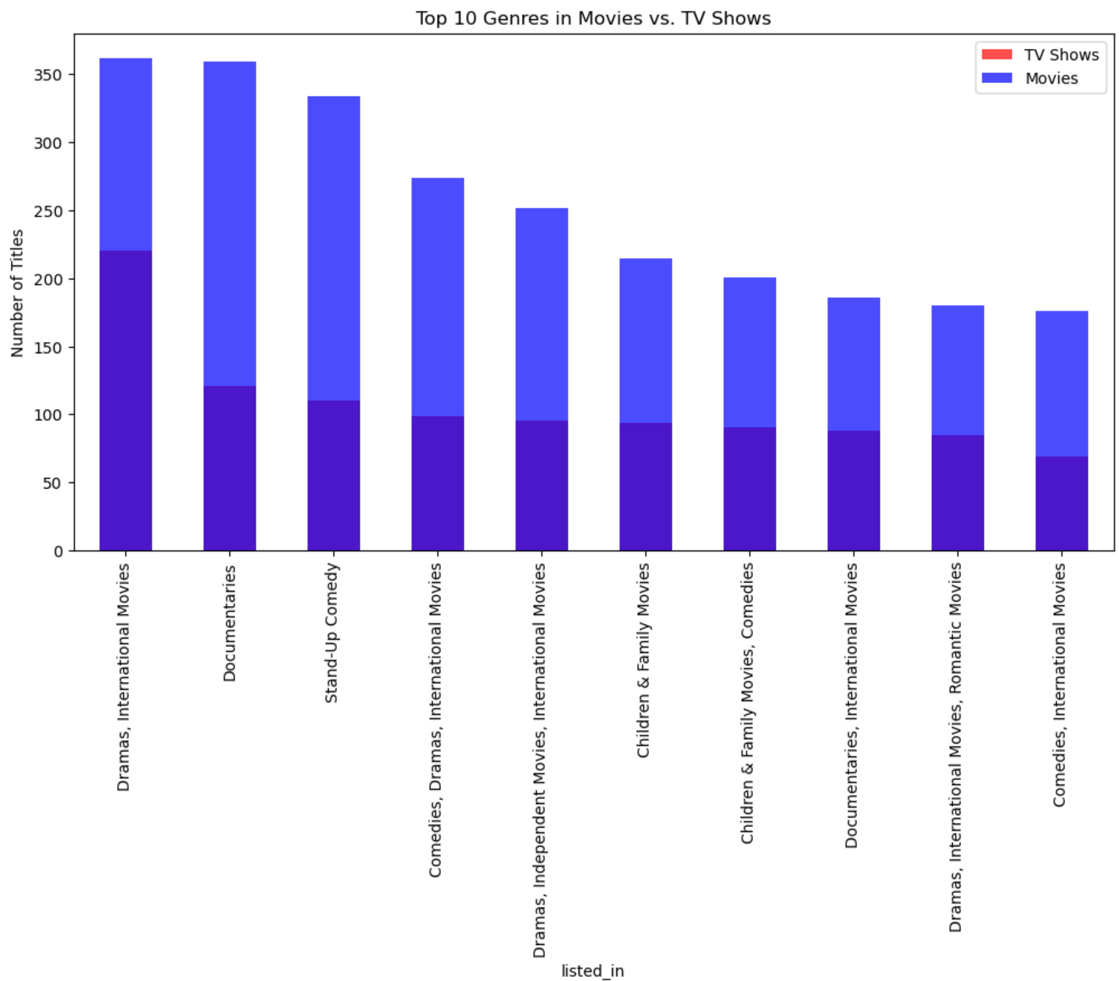
[13]:
```
plt.figure(figsize=(12,6))
df.groupby(['release_year', 'type']).size().unstack().plot(kind='bar',␣
↪stacked=True, figsize=(15,6), colormap="coolwarm")
plt.title("Number of Movies & TV Shows Released Each Year")
plt.xlabel("Release Year")
plt.ylabel("Number of Titles")
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



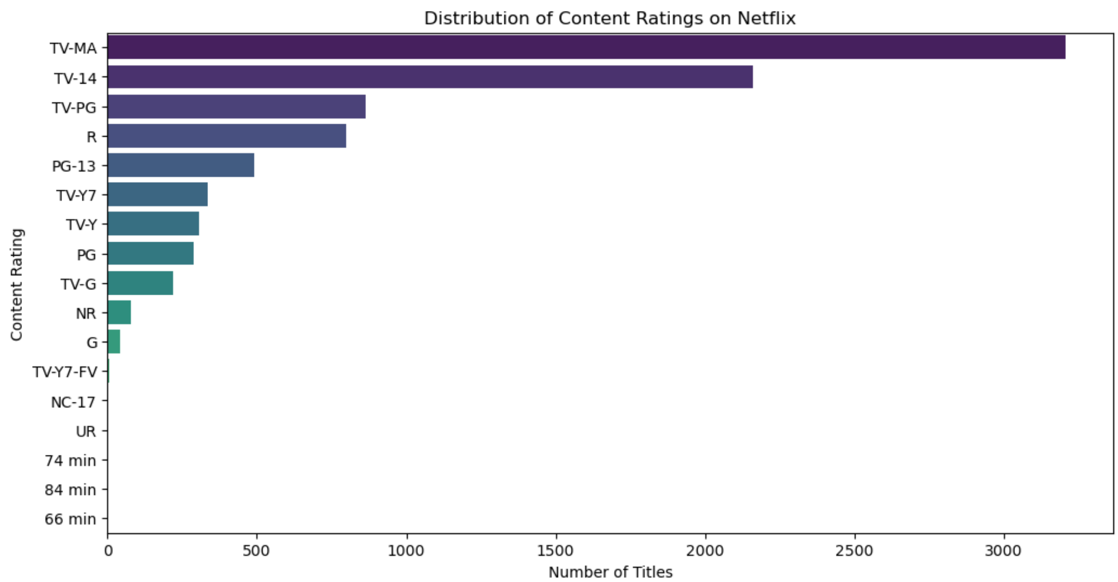
[14]:

```
plt.figure(figsize=(12,6))
tv_show_genres = df[df['type'] == 'TV Show']['listed_in'].value_counts().
↪nlargest(10)
movie_genres = df[df['type'] == 'Movie']['listed_in'].value_counts().
↪nlargest(10)

# Plot
tv_show_genres.plot(kind='bar', color='red', alpha=0.7, label="TV Shows",␣
↪figsize=(12,6))
movie_genres.plot(kind='bar', color='blue', alpha=0.7, label="Movies")
plt.title("Top 10 Genres in Movies vs. TV Shows")
plt.ylabel("Number of Titles")
plt.legend()
plt.show()
```



[15]:
```
plt.figure(figsize=(12,6))
sns.countplot(y=df['rating'], order=df['rating'].value_counts().index,␣
↪palette="viridis")
plt.title("Distribution of Content Ratings on Netflix")
plt.xlabel("Number of Titles")
plt.ylabel("Content Rating")
plt.show()
```

```
/tmp/ipykernel_133/3744753829.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(y=df['rating'], order=df['rating'].value_counts().index,
palette="viridis")
```



4 5 Data Preprocessing and Feature Engineering

This section involves preparing the dataset for machine learning. Steps include:

Dropping irrelevant columns (e.g., title, description, show_id, director, and cast). Handling missing values (e.g., filling missing country values with the most common country). Transforming categorical data using one-hot encoding for country, rating, and listed_in (genres). Converting the duration feature, where TV Shows in seasons are mapped to an estimated duration in minutes. we also do a chi squared test to see feature strength

[16]:
```
df.drop(columns=['director','cast','date_added'],inplace=True)
```

[17]:
```
# Check for missing values
print("\nMissing Values:\n", df.isnull().sum())
```

```
Missing Values:
 show_id          0
type            0
title           0
country       831
release_year    0
rating          4
duration        3
listed_in       0
description     0
dtype: int64
```

[18]:
```python
# Fill missing categorical values with "Unknown"
df['country'].fillna("Unknown", inplace=True)

# Function to convert 'duration' column
def convert_duration(value):
    if isinstance(value, str):  # Check if value is a string
        if 'min' in value:
            return int(value.replace(" min", ""))  # Extract only the numeric part
        elif 'Season' in value:
            return int(value.split(' ')[0]) * 600  # Convert seasons to minutes (1 Season   600 min)
    return np.nan  # If value is NaN, keep it as NaN

# Apply the conversion
df['duration'] = df['duration'].apply(convert_duration)

# Fill missing duration values with the median duration
df['duration'].fillna(df['duration'].median(), inplace=True)

# Convert to integer type
df['duration'] = df['duration'].astype(int)

# Check if the issue is resolved
print(df['duration'].head())
```

```
0      90
1    1200
2     600
3     600
4    1200
Name: duration, dtype: int64
```

[19]:
```python
print(df.isnull().sum())
```

```
show_id         0
type            0
title           0
country         0
release_year    0
rating          4
duration        0
listed_in       0
description     0
dtype: int64
```

[20]:
```
# we bext fill in missing values in 'release_year' with the median
df['release_year'].fillna(df['release_year'].median(), inplace=True)

# now can fill missing values in 'rating' with the most common value
df['rating'].fillna(df['rating'].mode()[0], inplace=True)

# Check if all missing values are fixed
print(df.isnull().sum())
```

```
show_id         0
type            0
title           0
country         0
release_year    0
rating          0
duration        0
listed_in       0
description     0
dtype: int64
```

as we can see now all missing values gone

[21]:
```
# Convert 'type' to binary (0 = Movie, 1 = TV Show)
df['type'] = df['type'].map({'Movie': 0, 'TV Show': 1})
```

[22]:
```
# One-hot encode categorical columns
df = pd.get_dummies(df, columns=['country', 'listed_in', 'rating'],␣
 ↪drop_first=True)
```

[23]:
```
X = df.drop(columns=['type', 'title', 'description', 'show_id'])  # Drop␣
 ↪irrelevant columns
y = df['type']  # Target variable (Movie vs. TV Show)
```

[24]:
```
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder
import numpy as np
import pandas as pd

# Convert categorical target variable to numeric
y_encoded = LabelEncoder().fit_transform(y)

# Apply Chi-Square test only to categorical features
# Select all columns related to rating, country, and listed_in
X_categorical = df.filter(like='rating').join(df.filter(like='country')).
 ↪join(df.filter(like='listed_in'))
X_encoded = pd.get_dummies(X_categorical)  # One-hot encode for Chi-Square test

# Compute Chi-Square scores
```

```
chi_scores, p_values = chi2(X_encoded, y_encoded)

# Convert to DataFrame
chi2_results = pd.DataFrame({'Feature': X_encoded.columns, 'Chi2 Score':␣
 ↪chi_scores, 'p-value': p_values})
chi2_results = chi2_results.sort_values(by='Chi2 Score', ascending=False)

print("\nTop Features by Chi-Square Test:\n", chi2_results.head(10))
```

```
Top Features by Chi-Square Test:
                                                Feature  Chi2 Score
p-value
1185                                 listed_in_Kids' TV  504.043348
1.253794e-111
7                                              rating_R  343.017633
1.405052e-76
1175        listed_in_International TV Shows, TV Dramas  277.223842
3.023986e-62
1007  listed_in_Crime TV Shows, International TV Sho…  252.021674
9.412466e-57
266                                       country_India  227.641640
1.948376e-51
1196                       listed_in_Kids' TV, TV Comedies  226.819507
2.944257e-51
450                                 country_South Korea  225.994373
4.455905e-51
1209                              listed_in_Reality TV  217.655082
2.936853e-49
1156  listed_in_International TV Shows, Romantic TV …  215.363976
9.282536e-49
6                                          rating_PG-13  213.870494
1.965484e-48
```

[25]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Columns: 1279 entries, release_year to rating_UR
dtypes: bool(1277), int64(2)
memory usage: 10.9 MB
```

[26]:
```
print(X.columns)
```

```
Index(['release_year', 'duration', 'country_, South Korea', 'country_Argentina',
'country_Argentina, Brazil, France, Poland, Germany, Denmark',
'country_Argentina, Chile', 'country_Argentina, Chile, Peru',
'country_Argentina, France', 'country_Argentina, France, United States, Germany,
```

```
Qatar', 'country_Argentina, Italy',
       …
       'rating_PG-13', 'rating_R', 'rating_TV-14', 'rating_TV-G', 'rating_TV-
MA', 'rating_TV-PG', 'rating_TV-Y', 'rating_TV-Y7', 'rating_TV-Y7-FV',
'rating_UR'], dtype='object', length=1279)
```

Test train split next

5 6. Splitting the Data and Scaling

The dataset is split into 80% training and 20% testing using stratified sampling to maintain class distribution. Standardization is applied using StandardScaler() to ensure that numeric features have a mean of 0 and a standard deviation of 1. This step is particularly important for Logistic Regression, which is sensitive to feature scaling.

[27]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

print("Training set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
Training set size: (7045, 1279)
Test set size: (1762, 1279)
```

[28]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

6 7. Model Training (Default Parameters)

Two classification models are trained using default hyperparameters:

Random Forest Classifier: A tree-based ensemble method that captures non-linear relationships. Logistic Regression: A linear model suitable for binary classification problems. Performance is evaluated using accuracy, precision, recall, and F1-score.

[29]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
#making and training model process
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

# Evaluate performance
```

```
rf_acc = accuracy_score(y_test, y_pred_rf)
print("Random Forest Accuracy:", rf_acc)
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1227
           1       1.00      1.00      1.00       535

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762
```

[30]:
```
from sklearn.linear_model import LogisticRegression

# #making and training model process
logreg_model = LogisticRegression(max_iter=5000, class_weight='balanced')
logreg_model.fit(X_train_scaled, y_train)
y_pred_logreg = logreg_model.predict(X_test_scaled)

# Evaluate performance
logreg_acc = accuracy_score(y_test, y_pred_logreg)
print("Logistic Regression Accuracy:", logreg_acc)
print("\nClassification Report:\n", classification_report(y_test,␣
 ↪y_pred_logreg))
```

```
Logistic Regression Accuracy: 0.9920544835414302

Classification Report:
               precision    recall  f1-score   support

           0       0.99      1.00      0.99      1227
           1       1.00      0.97      0.99       535

    accuracy                           0.99      1762
   macro avg       0.99      0.99      0.99      1762
weighted avg       0.99      0.99      0.99      1762
```

7 8. Hyperparameter Tuning with GridSearchCV

To optimize model performance, hyperparameter tuning is performed using GridSearchCV:

Random Forest is tuned for: n_estimators, max_depth, min_samples_split, and min_samples_leaf. Logistic Regression is tuned for: C (regularization strength) and solver. The best parameters are selected based on cross-validated accuracy.

8 8.5 Model Evaluation and Performance Metrics

Accuracy, precision, recall, and F1-score are calculated for both default and tuned models. A comparison table is created to summarize model performance. Confusion matrices are visualized to analyze misclassification patterns.

FINE TUNING MAY TAKE A FEW MINUTES SO BARE WITH IT FOR SOME TIME

[31]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define hyperparameters to tune
rf_params = {
    'n_estimators': [50, 100],  # Reduce options
    'max_depth': [10, 20],  # Remove None to avoid overfitting
    'min_samples_split': [5, 10],  # Skip 2 since it's default
    'min_samples_leaf': [2, 4]  # Slightly larger values
}

# Initialize Random Forest
rf = RandomForestClassifier(random_state=42)

# Perform Grid Search
rf_grid = GridSearchCV(rf, rf_params, cv=5, scoring='accuracy', n_jobs=-1)
rf_grid.fit(X_train, y_train)

# Get best parameters and accuracy
print("Best Parameters for Random Forest:", rf_grid.best_params_)
print("Best Random Forest Accuracy:", rf_grid.best_score_)

# Train model with best parameters
best_rf = rf_grid.best_estimator_
y_pred_rf_tuned = best_rf.predict(X_test)

# Evaluate performance
from sklearn.metrics import classification_report
print("\nFinal Tuned Random Forest Model Performance:\n",␣
 ↪classification_report(y_test, y_pred_rf_tuned))
```

```
Best Parameters for Random Forest: {'max_depth': 20, 'min_samples_leaf': 4,
'min_samples_split': 5, 'n_estimators': 100}
Best Random Forest Accuracy: 0.9985805535841022

Final Tuned Random Forest Model Performance:
               precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00      1227
              1       1.00      0.99      1.00       535

       accuracy                           1.00      1762
      macro avg       1.00      1.00      1.00      1762
   weighted avg       1.00      1.00      1.00      1762
```

explain reasoning for dropping certain text base columns like director and cast but encoding countries

[32]:

```python
from sklearn.linear_model import LogisticRegression

# Define hyperparameters to tune
logreg_params = {
    'C': [0.01, 0.1, 1, 10, 100],    # Regularization strength
    'solver': ['liblinear', 'lbfgs'] # Solver for optimization
}

# Perform Grid Search
logreg = LogisticRegression(max_iter=5000, class_weight='balanced')
logreg_grid = GridSearchCV(logreg, logreg_params, cv=5, scoring='accuracy',␣
 ↪n_jobs=-1)
logreg_grid.fit(X_train_scaled, y_train)

# Get best parameters and accuracy
print("Best Parameters for Logistic Regression:", logreg_grid.best_params_)
print("Best Logistic Regression Accuracy:", logreg_grid.best_score_)

# Train model with best parameters
best_logreg = logreg_grid.best_estimator_
y_pred_logreg_tuned = best_logreg.predict(X_test_scaled)

# Evaluate performance
print("\nFinal Tuned Logistic Regression Model Performance:\n",␣
 ↪classification_report(y_test, y_pred_logreg_tuned))
```

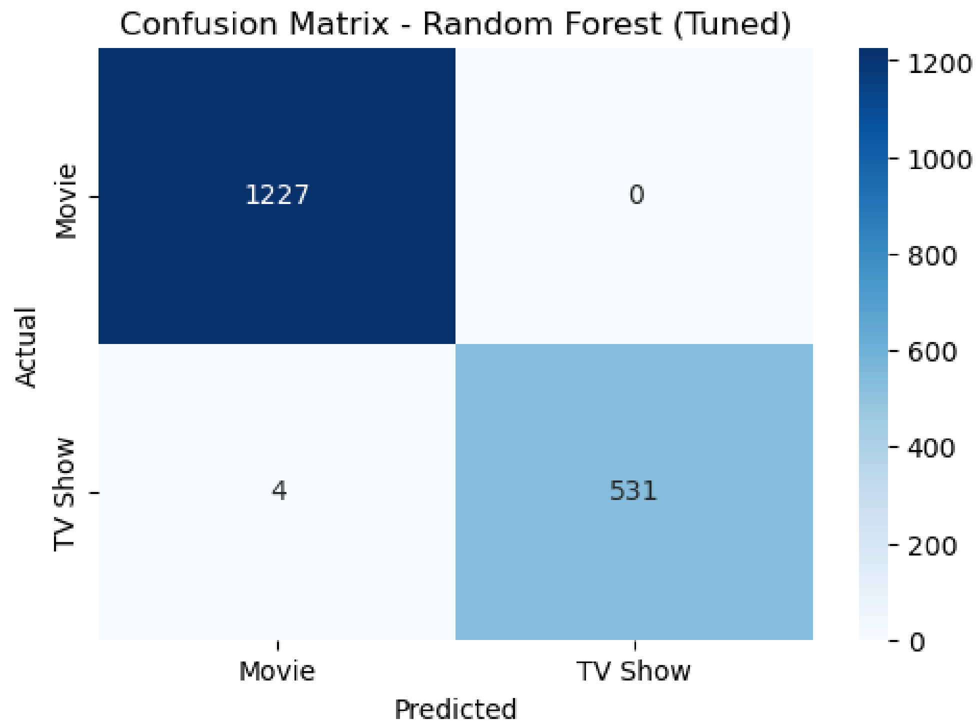
```
Best Parameters for Logistic Regression: {'C': 0.1, 'solver': 'liblinear'}
Best Logistic Regression Accuracy: 0.9990063875088715

Final Tuned Logistic Regression Model Performance:
               precision    recall  f1-score   support

              0       0.99      1.00      1.00      1227
              1       1.00      0.98      0.99       535

       accuracy                           0.99      1762
```

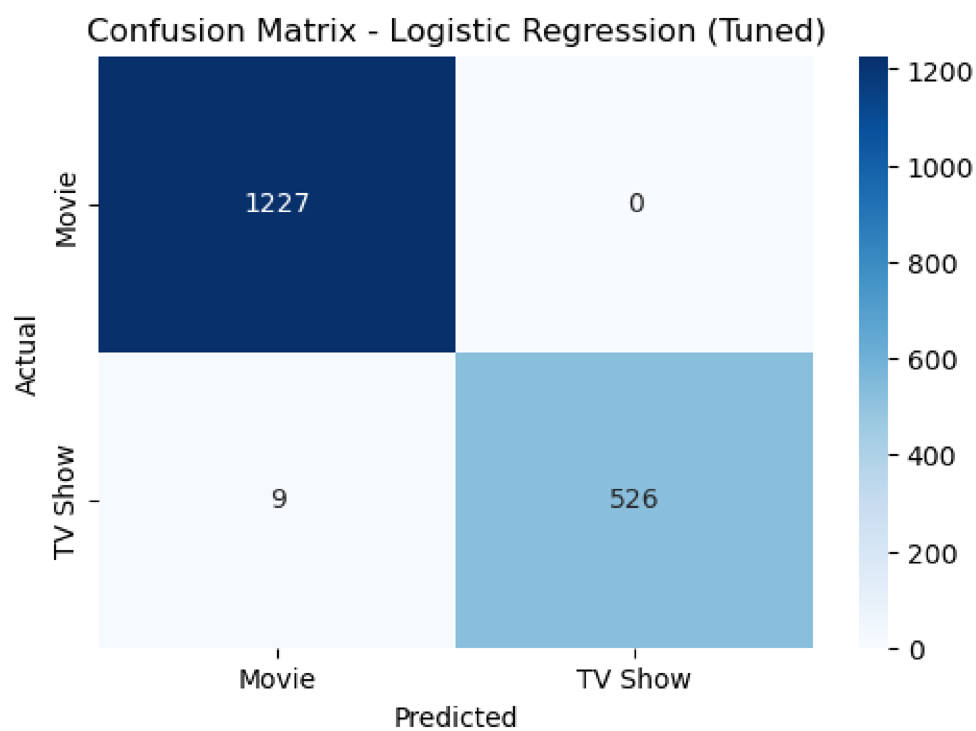
```
   macro avg       1.00      0.99      0.99      1762
weighted avg       0.99      0.99      0.99      1762
```

9 9 Confusion Matrix

here we use a confusion matrix to test our fine tuned models

[33]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Plot Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, model_name):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Movie',␣
 ↪'TV Show'], yticklabels=['Movie', 'TV Show'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.show()

# Plot for both models
plot_confusion_matrix(y_test, y_pred_rf_tuned, "Random Forest (Tuned)")
plot_confusion_matrix(y_test, y_pred_logreg_tuned, "Logistic Regression␣
 ↪(Tuned)")
```


Confusion Matrix - Random Forest (Tuned)
Actual
Movie
TV Show
1227
0
4
531
Predicted
Movie
TV Show
1200
1000
800
600
400
200
0


Confusion Matrix - Logistic Regression (Tuned)
Actual
Movie
TV Show
1227
0
9
526
Predicted
Movie
TV Show
1200
1000
800
600
400
200
0

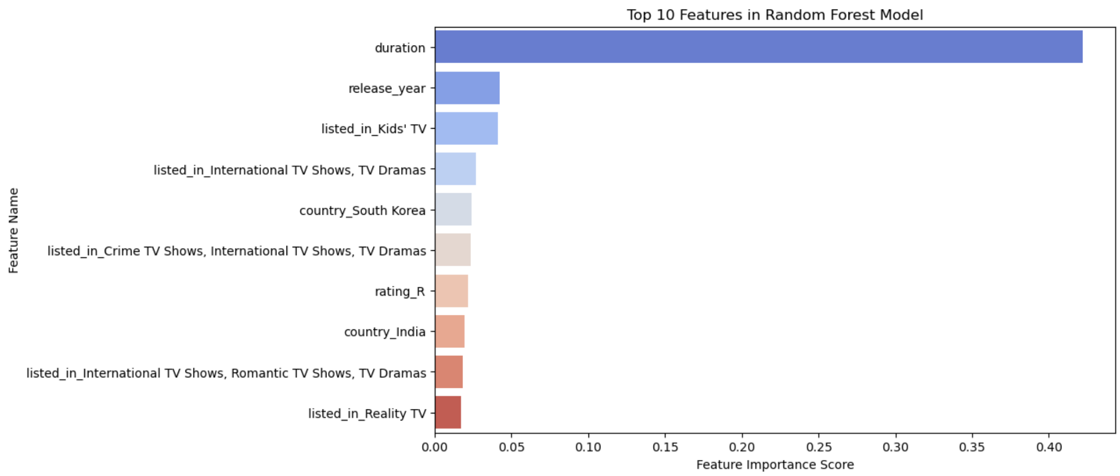
10 BONUS Feature Importance Analysis

Random Forest feature importance is analyzed to determine which attributes contribute the most to classification. A bar chart visualizes the top 10 most influential features.

[34]:
```python
# Get feature importances
importances = best_rf.feature_importances_
feature_names = X_train.columns

# Sort and display top features
feature_imp = sorted(zip(feature_names, importances), key=lambda x: x[1],␣
↪reverse=True)

print("  Top Features Influencing Classification:")
for feature, importance in feature_imp[:10]:  # Display top 10
    print(f"{feature}: {importance:.4f}")

# Plot feature importance
plt.figure(figsize=(10,6))
sns.barplot(x=[x[1] for x in feature_imp[:10]], y=[x[0] for x in feature_imp[:
↪10]], palette="coolwarm")
plt.xlabel("Feature Importance Score")
plt.ylabel("Feature Name")
plt.title("Top 10 Features in Random Forest Model")
plt.show()
```

```
  Top Features Influencing Classification:
duration: 0.4221
release_year: 0.0426
listed_in_Kids' TV: 0.0413
listed_in_International TV Shows, TV Dramas: 0.0268
country_South Korea: 0.0238
listed_in_Crime TV Shows, International TV Shows, TV Dramas: 0.0234
rating_R: 0.0219
country_India: 0.0195
listed_in_International TV Shows, Romantic TV Shows, TV Dramas: 0.0182
listed_in_Reality TV: 0.0174
```

```
/tmp/ipykernel_133/1018452132.py:14: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=[x[1] for x in feature_imp[:10]], y=[x[0] for x in
feature_imp[:10]], palette="coolwarm")
```



11 10. Cross-Validation for Generalization

To ensure that the models generalize well to unseen data:

5-fold cross-validation is performed on both default and tuned models. Cross-validation accuracy scores are compared to determine which model is more robust.

may take arround a minute or so to run

[35]:
```
from sklearn.model_selection import cross_val_score

# Cross-validation for Default Random Forest and TUned
rf_default_cv = cross_val_score(rf_model, X_train, y_train, cv=5)
print("Default Random Forest Mean CV Accuracy:", rf_default_cv.mean())
rf_tuned_cv = cross_val_score(best_rf, X_train, y_train, cv=5)
print("Tuned Random Forest Mean CV Accuracy:", rf_tuned_cv.mean())

# Cross-validation for Default Logistic Regression and Tuned
logreg_default_cv = cross_val_score(logreg_model, X_train_scaled, y_train, cv=5)
print("Default Logistic Regression Mean CV Accuracy:", logreg_default_cv.mean())
logreg_tuned_cv = cross_val_score(best_logreg, X_train_scaled, y_train, cv=5)
print("Tuned Logistic Regression Mean CV Accuracy:", logreg_tuned_cv.mean())
```

```
Default Random Forest Mean CV Accuracy: 1.0
Tuned Random Forest Mean CV Accuracy: 0.9985805535841022
Default Logistic Regression Mean CV Accuracy: 0.9913413768630234
Tuned Logistic Regression Mean CV Accuracy: 0.9990063875088715
```

[36]:
```
print(df.dtypes)  # Check column data types
```

```
show_id              object
type                  int64
```

```
title               object
release_year         int64
duration             int64
                     …
rating_TV-PG          bool
rating_TV-Y           bool
rating_TV-Y7          bool
rating_TV-Y7-FV       bool
rating_UR             bool
Length: 1283, dtype: object
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: