Business Case: Netflix EDA

```
!pip install pandas
```

```
Requirement already satisfied: pandas in e:\rasa\lib\site-packages
(2.2.0)
Requirement already satisfied: numpy<2,>=1.23.2 in e:\rasa\lib\site-
packages (from pandas) (1.24.3)
Requirement already satisfied: python-dateutil>=2.8.2 in e:\rasa\lib\
site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in e:\rasa\lib\site-
packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in e:\rasa\lib\site-
packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in e:\rasa\lib\site-packages
(from python-dateutil>=2.8.2->pandas) (1.16.0)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#importing dataset
data = pd.read_csv('G:/dsml-scaler/python/business
case/netflix_data.csv', sep=',')
```

Observations of data

```
data.head()
```

```
  show_id     type                  title         director  \
0      s1    Movie   Dick Johnson Is Dead  Kirsten Johnson
1      s2  TV Show          Blood & Water              NaN
2      s3  TV Show              Ganglands  Julien Leclercq
3      s4  TV Show  Jailbirds New Orleans              NaN
4      s5  TV Show           Kota Factory              NaN

                                                cast        country  \
0                                                NaN  United States
1  Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...   South Africa
2  Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...            NaN
3                                                NaN            NaN
4  Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...          India

           date_added  release_year rating   duration  \
0  September 25, 2021          2020  PG-13     90 min
1  September 24, 2021          2021  TV-MA  2 Seasons
2  September 24, 2021          2021  TV-MA   1 Season
3  September 24, 2021          2021  TV-MA   1 Season
4  September 24, 2021          2021  TV-MA  2 Seasons
```

```
                                              listed_in  \
0                                         Documentaries
1    International TV Shows, TV Dramas, TV Mysteries
2  Crime TV Shows, International TV Shows, TV Act...
3                               Docuseries, Reality TV
4  International TV Shows, Romantic TV Shows, TV ...

                                           description
0  As her father nears the end of his life, filmm...
1  After crossing paths at a party, a Cape Town t...
2  To protect his family from a powerful drug lor...
3  Feuds, flirtations and toilet talk go down amo...
4  In a city of coaching centers known to train I...
```

```
data.shape
```

```
(8807, 12)
```

```
data.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
```

```
#unnesting of columns

# 1. Unnest 'director' column
data['director'] = data['director'].str.split(', ')
data= data.explode('director')

# 2. Unnest 'cast' column
data['cast'] = data['cast'].str.split(', ')
data = data.explode('cast')
```

```
# 3. Unnest 'country' column
data['country'] = data['country'].str.split(', ')
data = data.explode('country')

# 4. Unnest 'director' column
data['listed_in'] = data['listed_in'].str.split(', ')
data = data.explode('listed_in')

data.shape
```

```
(201991, 12)
```

```
# Conversion of 'date_added' to datetime
data['date_added'] = pd.to_datetime(data['date_added'],
errors='coerce')

data.dtypes
```

```
show_id                 object
type                    object
title                   object
director                object
cast                    object
country                 object
date_added      datetime64[ns]
release_year             int64
rating                  object
duration                object
listed_in               object
description             object
dtype: object
```

```
# Missing Value Detection
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
show_id             0
type                0
title               0
director        50643
cast             2146
country         11897
date_added       1746
release_year        0
rating             67
duration            3
listed_in           0
```

```
description          0
dtype: int64
```

```
# handling missing values of director
data['director'].fillna('Unknown', inplace=True)
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7024\1408846104.py:2:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data['director'].fillna('Unknown', inplace=True)
```

```
data['cast'] = data.groupby(['type', 'country', 'release_year'])
['cast'].transform(lambda x: x.fillna(x.mode().iloc[0] if not
x.mode().empty else np.nan))
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7024\1101921029.py:1:
FutureWarning: Downcasting object dtype arrays
on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to
the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  data['cast'] = data.groupby(['type', 'country', 'release_year'])
['cast'].transform(lambda x: x.fillna(x.mode().iloc[0] if not
x.mode().empty else np.nan))
```

```
# handling missing values of date_added
data.dropna(subset=['date_added'], inplace=True)
```

```
# handling missing value of rating
data['rating'] = data.groupby(['type', 'listed_in'])
['rating'].transform(lambda x: x.fillna(x.mode().iloc[0] if not
x.mode().empty else np.nan))
```

```
# handling missing value of duration
data = data.dropna(subset=['duration'])
```

```
#handling missing value of country
data['country'] = data.groupby(['listed_in', 'release_year'])
['country'].transform(lambda x: x.fillna(x.mode().iloc[0]if not
x.mode().empty else np.nan))
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7024\3415337459.py:2:
FutureWarning: Downcasting object dtype arrays
on .fillna, .ffill, .bfill is deprecated and will change in a future
version. Call result.infer_objects(copy=False) instead. To opt-in to
the future behavior, set
`pd.set_option('future.no_silent_downcasting', True)`
  data['country'] = data.groupby(['listed_in', 'release_year'])
['country'].transform(lambda x: x.fillna(x.mode().iloc[0]if not
x.mode().empty else np.nan))
```

```
missing_values = data.isnull().sum()
print("Missing Values:")
print(missing_values)
```

```
Missing Values:
show_id             0
type                0
title               0
director            0
cast            12019
country           176
date_added          0
release_year        0
rating              0
duration            0
listed_in           0
description         0
dtype: int64
```

```
data = data.drop_duplicates(subset=['show_id', 'type'])
```

```
data['cast']=data.groupby(['listed_in', 'type'])
['cast'].transform(lambda x: x.fillna(x.mode().iloc[0] if not
x.mode().empty else np.nan))
```

```
data = data.dropna(subset=['country'])
```

```
missing_values_after_dropping_duplicates = data.isnull().sum()
print("Missing Values after dropping duplicates:")
print(missing_values_after_dropping_duplicates)
```

```
Missing Values after dropping duplicates:
show_id         0
type            0
title           0
director        0
cast            0
country         0
date_added      0
release_year    0
rating          0
```

```
duration        0
listed_in       0
description     0
dtype: int64

total_duplicates = data.duplicated().sum()
total_duplicates

0

data.head()

  show_id     type                   title         director
cast  \
0      s1    Movie   Dick Johnson Is Dead  Kirsten Johnson  Andre
Robinson
1      s2  TV Show          Blood & Water          Unknown      Ama
Qamata
2      s3  TV Show              Ganglands  Julien Leclercq      Adil
Dehbi
3      s4  TV Show  Jailbirds New Orleans          Unknown      Adil
Dehbi
4      s5  TV Show           Kota Factory          Unknown      Mayur
More

         country date_added  release_year rating   duration  \
0  United States 2021-09-25          2020  PG-13     90 min
1   South Africa 2021-09-24          2021  TV-MA  2 Seasons
2  United States 2021-09-24          2021  TV-MA   1 Season
3  United States 2021-09-24          2021  TV-MA   1 Season
4          India 2021-09-24          2021  TV-MA  2 Seasons

                listed_in
description
0           Documentaries  As her father nears the end of his life,
filmm...
1  International TV Shows  After crossing paths at a party, a Cape
Town t...
2         Crime TV Shows  To protect his family from a powerful drug
lor...
3              Docuseries  Feuds, flirtations and toilet talk go down
amo...
4  International TV Shows  In a city of coaching centers known to
train I...

# Generate a statistical summary for numerical columns
statistical_summary = data.describe()

# Display the statistical summary
print("Statistical Summary:")
print(statistical_summary)
```

```
Statistical Summary:
                           date_added  release_year
count                            8695   8695.000000
mean   2019-05-23 03:51:01.759632128   2014.212766
min              2008-01-01 00:00:00   1942.000000
25%              2018-04-20 00:00:00   2013.000000
50%              2019-07-12 00:00:00   2017.000000
75%              2020-08-25 12:00:00   2019.000000
max              2021-09-25 00:00:00   2021.000000
std                              NaN      8.767059
```

```
# Generate a more comprehensive statistical summary
comprehensive_summary = data.describe(include='all')

# Display the comprehensive statistical summary
print("Comprehensive Statistical Summary:")
print(comprehensive_summary)
```

```
Comprehensive Statistical Summary:
       show_id   type                  title director        cast  \
count     8695   8695                   8695     8695        8695
unique    8695      2                   8695     4401        5075
top         s1  Movie  Dick Johnson Is Dead  Unknown  Adil Dehbi
freq         1   6122                      1     2533         164
mean       NaN    NaN                    NaN      NaN         NaN
min        NaN    NaN                    NaN      NaN         NaN
25%        NaN    NaN                    NaN      NaN         NaN
50%        NaN    NaN                    NaN      NaN         NaN
75%        NaN    NaN                    NaN      NaN         NaN
max        NaN    NaN                    NaN      NaN         NaN
std        NaN    NaN                    NaN      NaN         NaN

              country                     date_added  release_year
rating  \
count            8695                           8695   8695.000000
8695
unique             89                            NaN           NaN
14
top     United States                            NaN           NaN
TV-MA
freq             3711                            NaN           NaN
3180
mean              NaN  2019-05-23 03:51:01.759632128   2014.212766
NaN
min               NaN            2008-01-01 00:00:00   1942.000000
NaN
25%               NaN            2018-04-20 00:00:00   2013.000000
NaN
50%               NaN            2019-07-12 00:00:00   2017.000000
NaN
```

```
75%                NaN            2020-08-25 12:00:00   2019.000000
NaN
max                NaN            2021-09-25 00:00:00   2021.000000
NaN
std                NaN                            NaN      8.767059
NaN

        duration listed_in
description
count       8695      8695
8695
unique       219        36
8664
top     1 Season    Dramas  Paranormal activity at a lush, abandoned
prope...
freq         1786      1600
4
mean          NaN       NaN
NaN
min           NaN       NaN
NaN
25%           NaN       NaN
NaN
50%           NaN       NaN
NaN
75%           NaN       NaN
NaN
max           NaN       NaN
NaN
std           NaN       NaN
NaN
```

```
categorical_columns = ['type',
'director','cast','country','rating','duration']

for column in categorical_columns:
    # Value counts
    value_counts_result = data[column].value_counts()
    print(f"\nValue Counts for {column}:")
    print(value_counts_result)

    # Unique values
    unique_values_result = data[column].unique()
    print(f"\nUnique Values for {column}:")
    print(unique_values_result)
```

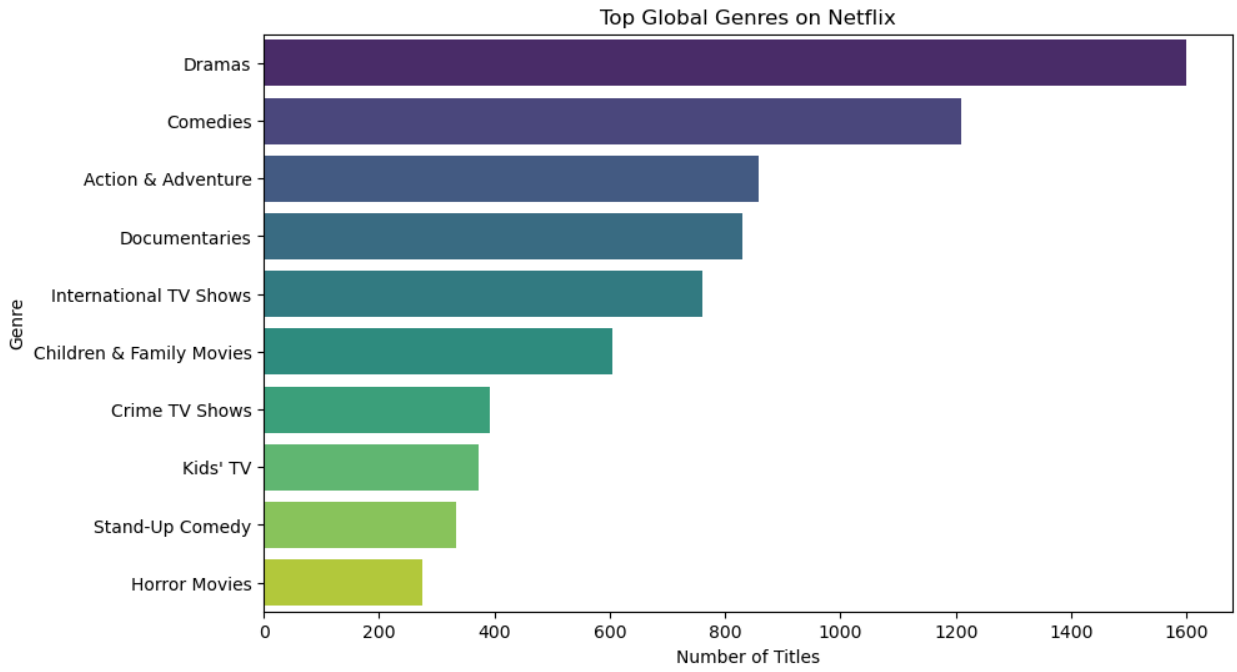
```
Value Counts for type:
type
Movie      6122
```

```
TV Show    2573
Name: count, dtype: int64

Unique Values for type:
['Movie' 'TV Show']

Value Counts for director:
director
Unknown          2533
Rajiv Chilaka      22
Raúl Campos        18
Suhas Kadav        16
Marcus Raboy       16
                 ...
Azazel Jacobs       1
Jung Ji-woo         1
Matt D'Avella       1
Parthiban           1
Mozez Singh         1
Name: count, Length: 4401, dtype: int64

Unique Values for director:
['Kirsten Johnson' 'Unknown' 'Julien Leclercq' ... 'Majid Al Ansari'
 'Peter Hewitt' 'Mozez Singh']

Value Counts for cast:
cast
Adil Dehbi                  164
Aaron Yan                   164
James Franco                148
Michela Luci                118
Aamir Khan                   86
                           ...
Ian Somerhalder               1
Peng Yuchang                  1
Katherine Escobar Farfan      1
Nirmal Rishi                  1
Sanam Saeed                   1
Name: count, Length: 5075, dtype: int64

Unique Values for cast:
['Andre Robinson' 'Ama Qamata' 'Adil Dehbi' ... 'Gökhan Atalay'
 'Michael Johnston' 'Sanam Saeed']

Value Counts for country:
country
United States     3711
India             1039
United Kingdom     642
Japan              384
```

```
Canada             261
                  ...
Namibia              1
Senegal              1
Cameroon             1
Syria                1
Somalia              1
Name: count, Length: 89, dtype: int64

Unique Values for country:
['United States' 'South Africa' 'India' 'United Kingdom' 'Germany'
 'Mexico' 'Thailand' 'South Korea' 'Turkey' 'Australia' 'Finland'
'China'
 'Nigeria' 'Japan' 'Spain' 'France' 'Belgium' 'Argentina' 'Saudi
Arabia'
 'Russia' 'Canada' 'Hong Kong' 'Italy' '' 'Taiwan' 'Ireland' 'New
Zealand'
 'Jordan' 'Colombia' 'Switzerland' 'Israel' 'Brazil' 'Bulgaria'
'Poland'
 'Indonesia' 'Egypt' 'Kuwait' 'Malaysia' 'Vietnam' 'Sweden' 'Lebanon'
 'Romania' 'Philippines' 'Iceland' 'Denmark' 'United Arab Emirates'
 'Netherlands' 'Norway' 'Syria' 'Mauritius' 'Austria' 'Czech Republic'
 'Cameroon' 'Uruguay' 'United Kingdom,' 'Kenya' 'Chile' 'Luxembourg'
 'Bangladesh' 'Portugal' 'Hungary' 'Senegal' 'Singapore' 'Serbia'
 'Namibia' 'Peru' 'Mozambique' 'Belarus' 'Ghana' 'Zimbabwe' 'Puerto
Rico'
 'Pakistan' 'Cyprus' 'Paraguay' 'Croatia' 'United States,' 'Cambodia'
 'Georgia' 'Soviet Union' 'Greece' 'West Germany' 'Iran' 'Venezuela'
 'Poland,' 'Slovenia' 'Guatemala' 'Ukraine' 'Jamaica' 'Somalia']

Value Counts for rating:
rating
TV-MA       3180
TV-14       2130
TV-PG        838
R            798
PG-13        490
TV-Y7        332
TV-Y         300
PG           287
TV-G         210
NR            78
G             41
TV-Y7-FV       5
NC-17          3
UR             3
Name: count, dtype: int64

Unique Values for rating:
['PG-13' 'TV-MA' 'PG' 'TV-14' 'TV-PG' 'TV-Y' 'TV-Y7' 'R' 'TV-G' 'G'
```

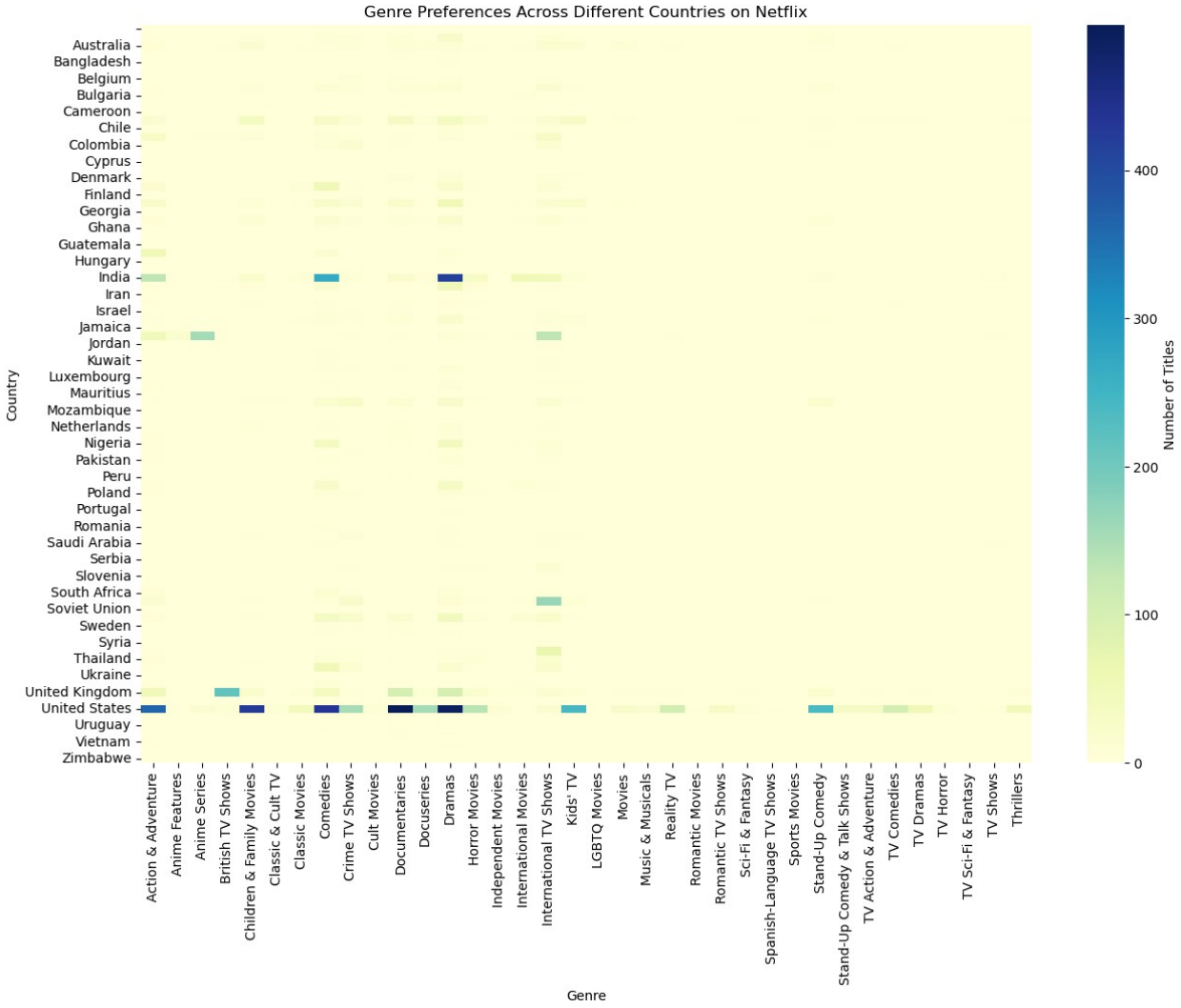
```
 'NC-17' 'NR' 'TV-Y7-FV' 'UR']

Value Counts for duration:
duration
1 Season     1786
2 Seasons     384
3 Seasons     178
90 min        152
93 min        146
             ...
253 min         1
208 min         1
5 min           1
16 min          1
191 min         1
Name: count, Length: 219, dtype: int64

Unique Values for duration:
['90 min' '2 Seasons' '1 Season' '91 min' '125 min' '9 Seasons' '104
min'
 '127 min' '4 Seasons' '67 min' '94 min' '5 Seasons' '161 min' '61
min'
 '166 min' '147 min' '103 min' '97 min' '106 min' '111 min' '3
Seasons'
 '110 min' '105 min' '96 min' '124 min' '116 min' '98 min' '23 min'
 '115 min' '122 min' '99 min' '88 min' '100 min' '6 Seasons' '102 min'
 '93 min' '95 min' '85 min' '83 min' '113 min' '13 min' '182 min' '48
min'
 '145 min' '87 min' '92 min' '80 min' '117 min' '128 min' '119 min'
 '143 min' '114 min' '118 min' '108 min' '63 min' '121 min' '142 min'
 '154 min' '120 min' '82 min' '109 min' '101 min' '86 min' '229 min'
 '76 min' '89 min' '156 min' '112 min' '107 min' '129 min' '135 min'
 '136 min' '165 min' '150 min' '133 min' '70 min' '84 min' '140 min'
 '78 min' '7 Seasons' '64 min' '59 min' '139 min' '69 min' '148 min'
 '189 min' '141 min' '130 min' '138 min' '81 min' '132 min' '10
Seasons'
 '123 min' '65 min' '68 min' '66 min' '62 min' '74 min' '131 min' '46
min'
 '38 min' '8 Seasons' '17 Seasons' '126 min' '39 min' '155 min' '159
min'
 '137 min' '12 min' '273 min' '36 min' '77 min' '60 min' '49 min' '58
min'
 '72 min' '204 min' '212 min' '25 min' '73 min' '29 min' '47 min' '32
min'
 '35 min' '71 min' '149 min' '33 min' '15 min' '54 min' '224 min'
 '162 min' '37 min' '75 min' '79 min' '55 min' '158 min' '164 min'
 '173 min' '181 min' '185 min' '21 min' '24 min' '51 min' '151 min'
 '42 min' '22 min' '134 min' '177 min' '13 Seasons' '52 min' '14 min'
 '53 min' '8 min' '57 min' '28 min' '50 min' '9 min' '26 min' '45 min'
```

```
 '171 min' '27 min' '44 min' '146 min' '20 min' '157 min' '17 min'
 '203 min' '41 min' '30 min' '194 min' '15 Seasons' '233 min' '237
min'
 '230 min' '195 min' '253 min' '152 min' '190 min' '160 min' '208 min'
 '180 min' '144 min' '5 min' '174 min' '170 min' '192 min' '209 min'
 '187 min' '172 min' '16 min' '186 min' '11 min' '193 min' '176 min'
 '56 min' '169 min' '40 min' '34 min' '10 min' '3 min' '168 min' '312
min'
 '153 min' '214 min' '31 min' '163 min' '19 min' '12 Seasons' '179
min'
 '43 min' '200 min' '196 min' '167 min' '178 min' '228 min' '18 min'
 '205 min' '201 min' '191 min']

# Q1.a: What are the top global genres on Netflix?
global_genre_counts = data['listed_in'].value_counts().head(10)

# Plotting the top global genres
plt.figure(figsize=(10, 6))
sns.barplot(x=global_genre_counts.values, y=global_genre_counts.index,
palette='viridis')
plt.title('Top Global Genres on Netflix')
plt.xlabel('Number of Titles')
plt.ylabel('Genre')
plt.show()
```



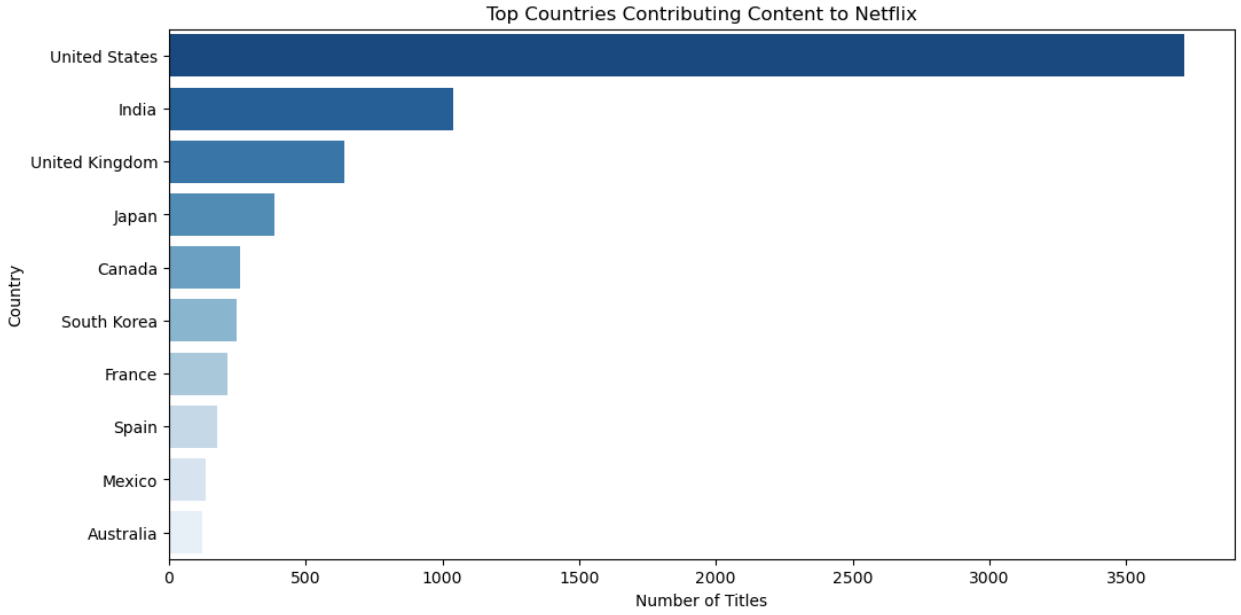

```
# Q1. b: How do genre preferences vary across different countries or
regions?
```

```
# Create a heatmap to show genre preferences across different
countries
plt.figure(figsize=(15, 10))
genre_by_country = data.groupby(['country',
'listed_in']).size().unstack().fillna(0)
sns.heatmap(genre_by_country, cmap='YlGnBu', cbar_kws={'label':
'Number of Titles'})
plt.title('Genre Preferences Across Different Countries on Netflix')
plt.xlabel('Genre')
plt.ylabel('Country')
plt.show()
```

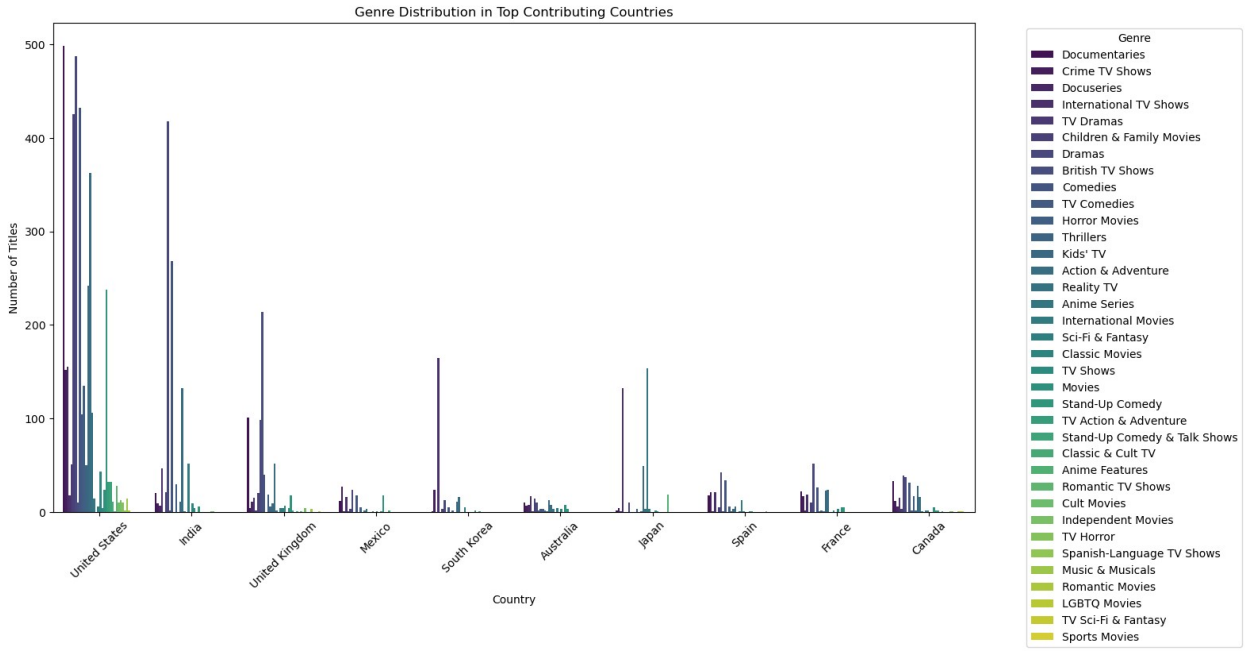



```
# Q2.a: Which countries contribute the most content to Netflix?
top_countries = data['country'].value_counts().head(10)

# Plotting the top contributing countries
plt.figure(figsize=(12, 6))
```

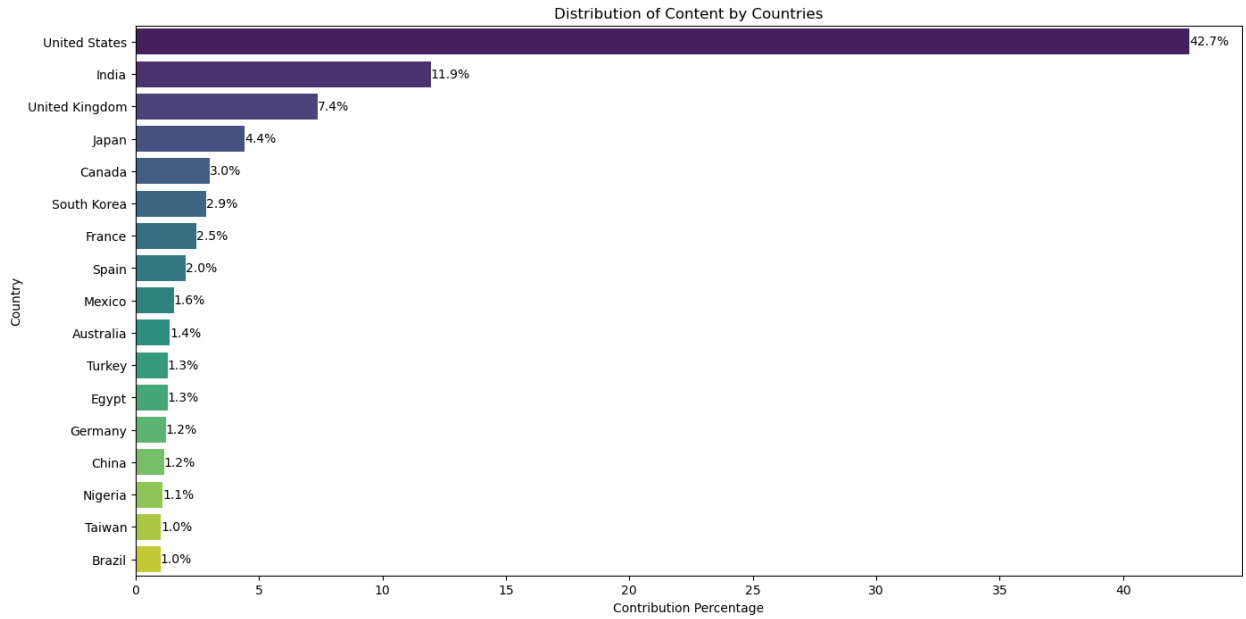
```
sns.barplot(x=top_countries.values, y=top_countries.index,
palette='Blues_r')
plt.title('Top Countries Contributing Content to Netflix')
plt.xlabel('Number of Titles')
plt.ylabel('Country')
plt.show()
```




```
#Q.2b: identify content trends specific to certain regions?

plt.figure(figsize=(15, 8))
sns.countplot(x='country', hue='listed_in',
data=data[data['country'].isin(top_countries.index)],
palette='viridis')
plt.title('Genre Distribution in Top Contributing Countries')
plt.xlabel('Country')
plt.ylabel('Number of Titles')
plt.xticks(rotation=45)
plt.legend(title='Genre', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

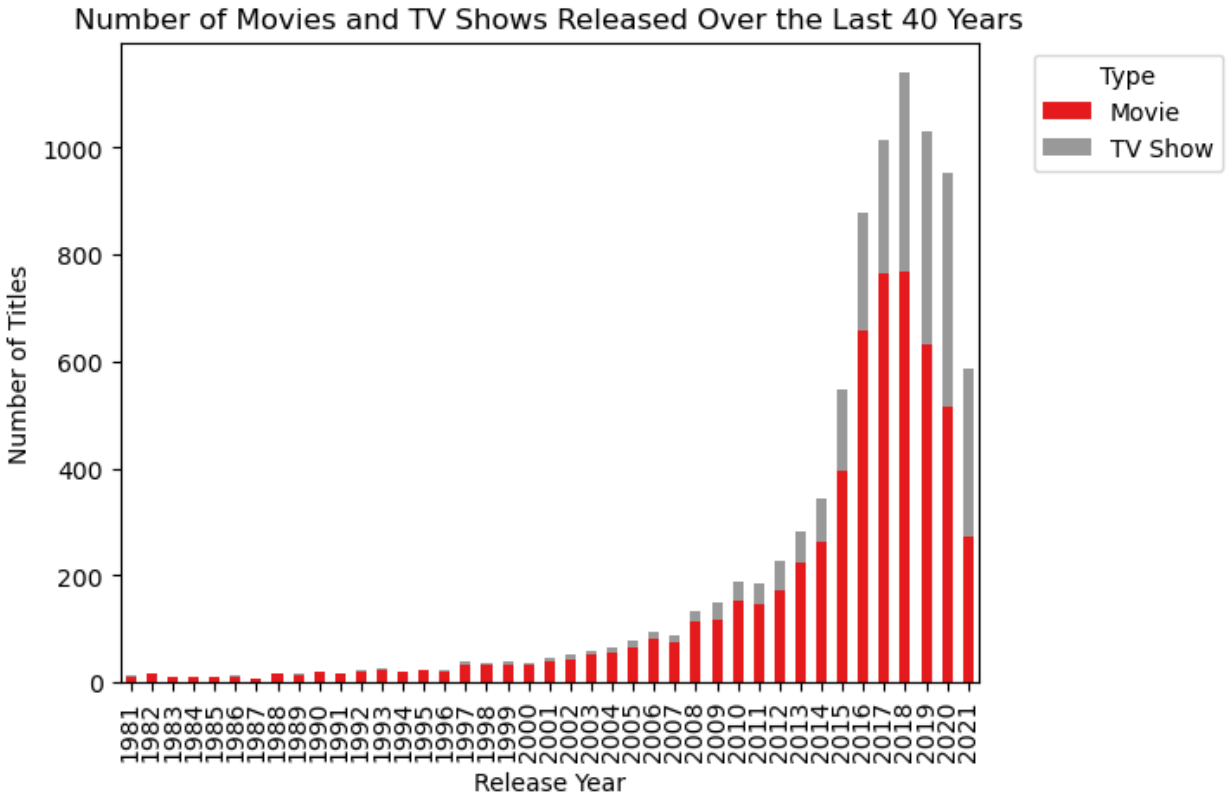



```
#Q2.c  Calculate the contribution percentages by country
country_contribution = data['country'].value_counts(normalize=True) *
100

# Filter out countries with a small contribution percentage for better
visualization
threshold = 1  # You can adjust this threshold as needed
significant_countries = country_contribution[country_contribution >=
threshold]

# Plot the distribution of content by countries
plt.figure(figsize=(16, 8))
sns.barplot(x=significant_countries.values,
y=significant_countries.index, palette='viridis')
plt.title('Distribution of Content by Countries')
plt.xlabel('Contribution Percentage')
plt.ylabel('Country')

# Add labels showing the percentage contribution on each bar
for index, value in enumerate(significant_countries):
    plt.text(value, index, f'{value:.1f}%', ha='left', va='center',
color='black')
plt.show()
```

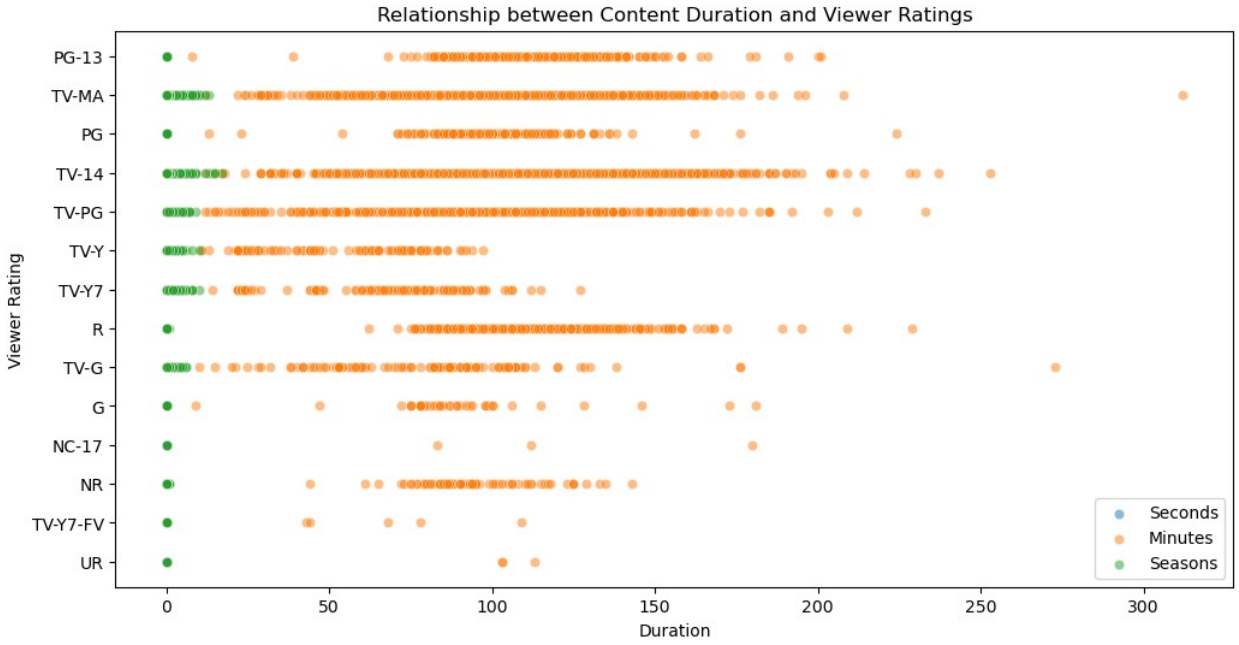



```
#Q3.a: How has the number of movies and TV shows released per year
changed over the last few decades?
# Filter data for the last 40 years
recent_data = data[data['release_year'] >= data['release_year'].max()
- 40]

# Group by 'release_year' and 'type' for the recent data
trends_by_year = recent_data.groupby(['release_year',
'type']).size().unstack().fillna(0)

# Plotting the trends over time
plt.figure(figsize=(20, 10))  # Adjust figure size
trends_by_year.plot(kind='bar', stacked=True, colormap='Set1')

plt.title('Number of Movies and TV Shows Released Over the Last 40
Years')
plt.xlabel('Release Year')
plt.xticks(rotation='vertical', fontsize=10)  # Adjust font size
plt.ylabel('Number of Titles')
plt.legend(title='Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

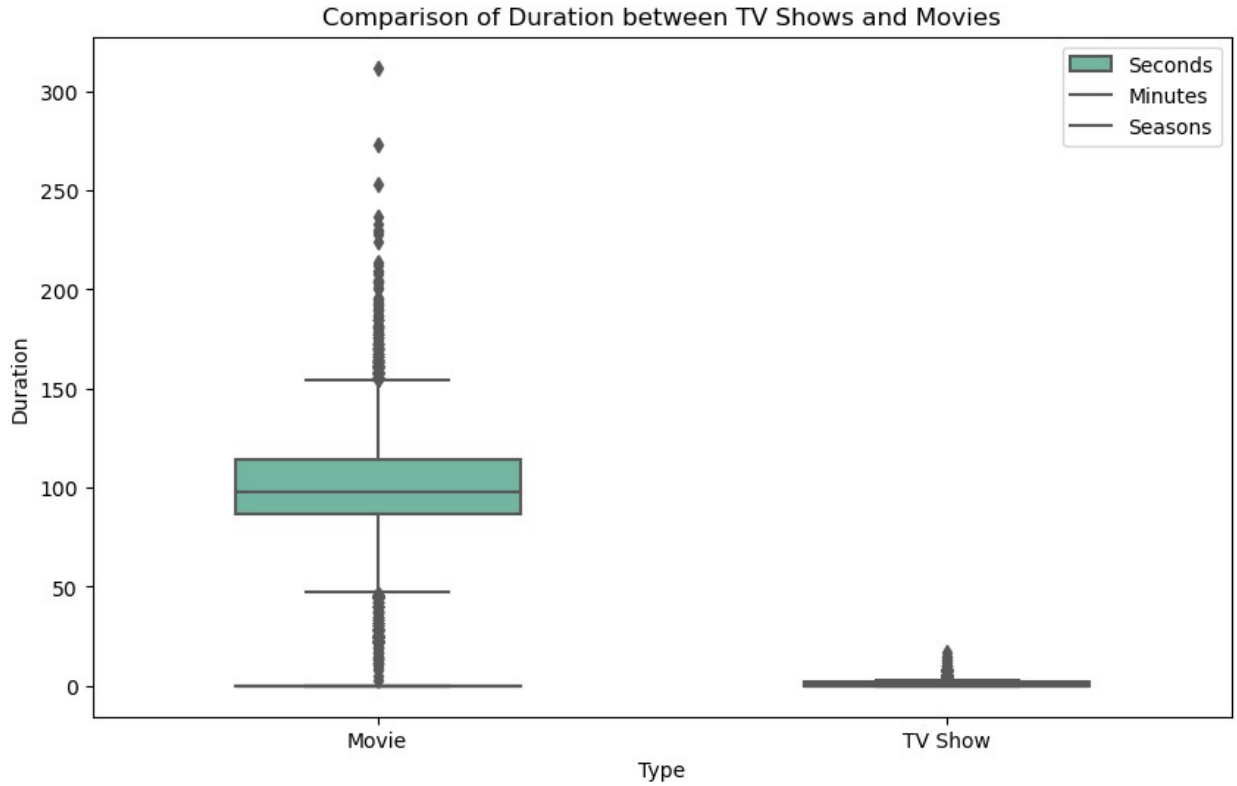
```
<Figure size 2000x1000 with 0 Axes>
```




```
#Q4.a: Investigate the relationship between content duration and
viewership, considering its impact on user ratings.
# Convert 'duration' to numerical values (seconds, minutes, seasons)
and create new columns
data['duration_sec'] = data['duration'].apply(lambda x: int(x.split('
')[0]) if 'sec' in x else 0)
data['duration_min'] = data['duration'].apply(lambda x: int(x.split('
')[0]) if 'min' in x else 0)
data['duration_season'] = data['duration'].apply(lambda x:
int(x.split(' ')[0]) if 'Season' in x else 0)

# Plotting the relationship between duration and viewer ratings
plt.figure(figsize=(12, 6))
sns.scatterplot(x='duration_sec', y='rating', data=data,
label='Seconds', alpha=0.5)
sns.scatterplot(x='duration_min', y='rating', data=data,
label='Minutes', alpha=0.5)
sns.scatterplot(x='duration_season', y='rating', data=data,
label='Seasons', alpha=0.5)
plt.title('Relationship between Content Duration and Viewer Ratings')
plt.xlabel('Duration')
plt.ylabel('Viewer Rating')
plt.legend()
plt.show()
```

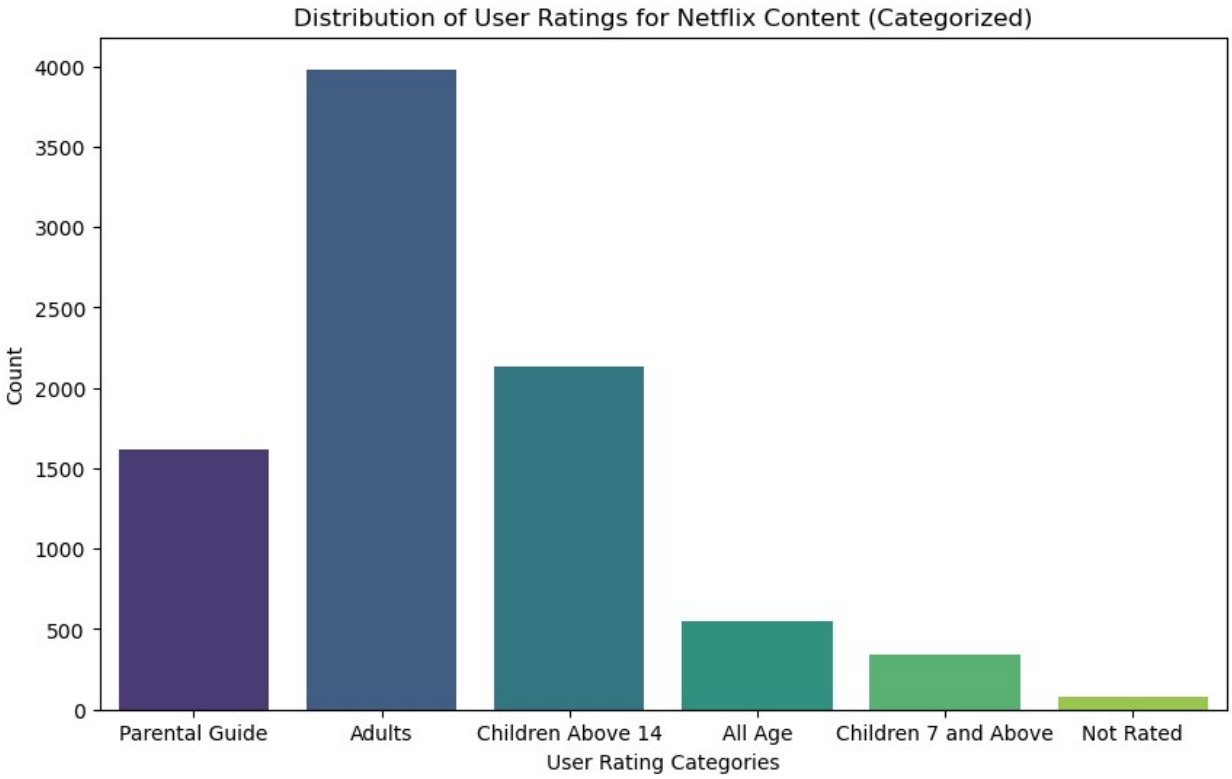



```
#Q4. b: Comparing the duration of TV shows to that of movies
plt.figure(figsize=(10, 6))
sns.boxplot(x='type', y='duration_sec', data=data, palette='Set2',
width=0.5)
sns.boxplot(x='type', y='duration_min', data=data, palette='Set2',
width=0.5)
sns.boxplot(x='type', y='duration_season', data=data, palette='Set2',
width=0.5)

# Set legend
plt.legend(['Seconds', 'Minutes', 'Seasons'])

plt.title('Comparison of Duration between TV Shows and Movies')
plt.xlabel('Type')
plt.ylabel('Duration')
plt.show()
```



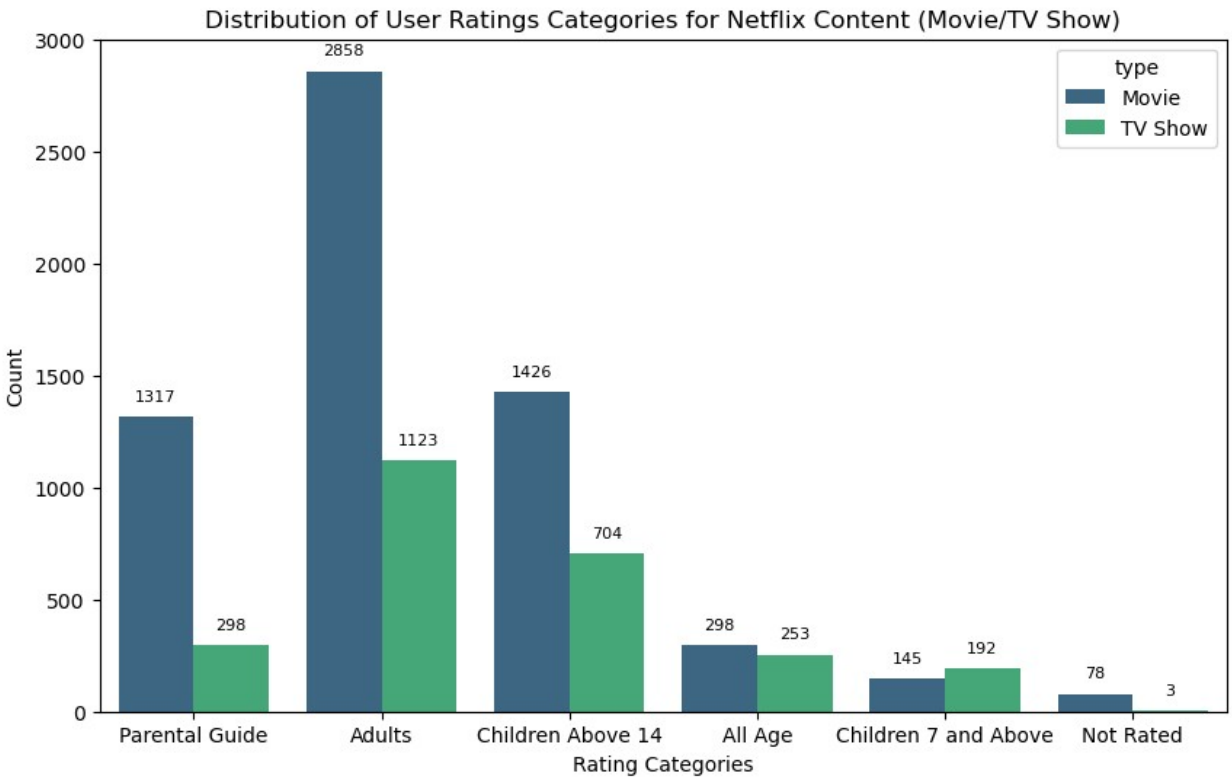

```
# Create a function to categorize ratings
def categorize_rating(rating):
    if rating in ['TV-MA', 'R', 'NC-17']:
        return 'Adults'
    elif rating in ['G', 'TV-G', 'TV-Y']:
        return 'All Age'
    elif rating in ['PG', 'PG-13', 'TV-PG']:
        return 'Parental Guide'
    elif rating in ['TV-Y7-FV', 'TV-Y7']:
        return 'Children 7 and Above'
    elif rating == 'TV-14':
        return 'Children Above 14'
    elif rating in ['UR', 'NR']:
        return 'Not Rated'
    else:
        return 'Unknown'  # You can modify this based on your specific
case

# Apply the function to create a new column 'rating_category'
data['rating_category'] = data['rating'].apply(categorize_rating)

#Q5.a: Distribution of user ratings for Netflix content based on new
rating categories
plt.figure(figsize=(10, 6))
sns.countplot(x='rating_category', data=data, palette='viridis')
```

```
plt.title('Distribution of User Ratings for Netflix Content
(Categorized)')
plt.xlabel('User Rating Categories')
plt.ylabel('Count')

plt.show()
```




```
data['cast'] =
data['cast'].str.encode('unicode_escape').str.decode('utf-8')

#Q5.b: distribution of new rating categories by type (Movie/TV Show)
plt.figure(figsize=(10, 6))
ax=sns.countplot(x='rating_category', hue='type', data=data,
palette='viridis')
# Add labels on the bars
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() /
2., p.get_height()),
                ha='center', va='center', xytext=(0, 10),
textcoords='offset points', fontsize=8)

plt.title('Distribution of User Ratings Categories for Netflix Content
(Movie/TV Show)')
plt.xlabel('Rating Categories')
```

```
plt.ylabel('Count')
plt.show()
```



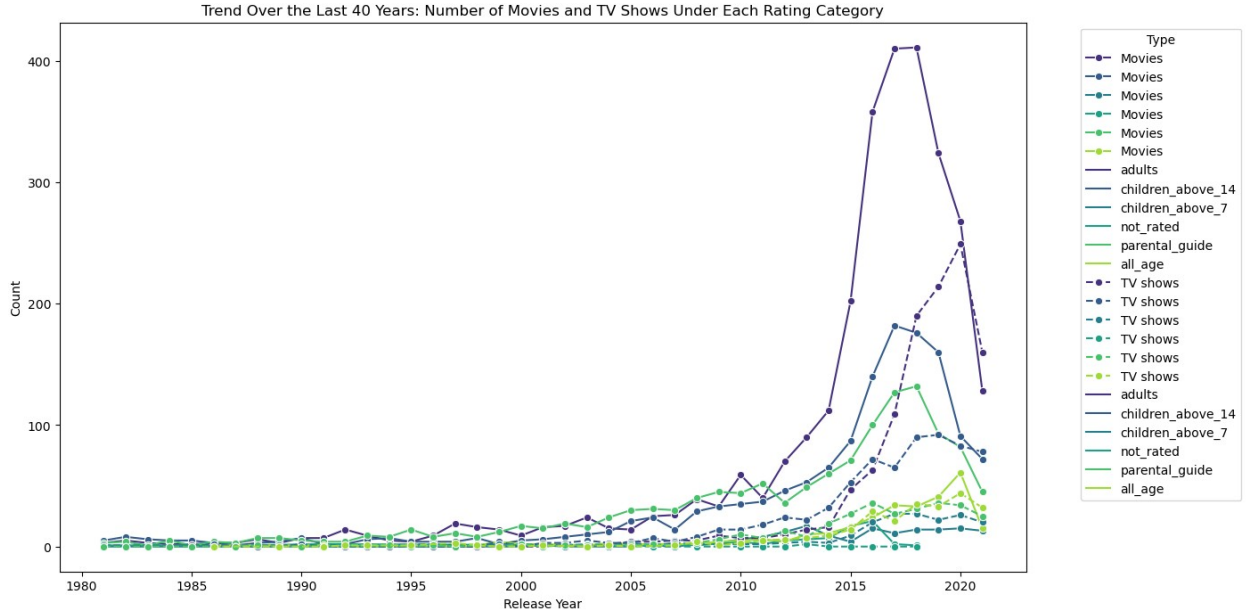

```
# Create a mapping dictionary for new rating categories
rating_category_mapping = {'adults': ['TV-MA', 'R', 'NC-17'],
                           'all_age': ['G', 'TV-G', 'TV-Y'],
                           'parental_guide': ['PG', 'PG-13', 'TV-
PG'],
                           'children_above_7': ['TV-Y7-FV', 'TV-Y7'],
                           'children_above_14': ['TV-14'],
                           'not_rated': ['UR', 'NR']}

# Create a new column 'rating_category' based on the mapping
data['rating_category'] = data['rating'].apply(lambda x: next((key for
key, value in rating_category_mapping.items() if x in value), None))

# Convert 'rating_category' to a numeric format for better
visualization
rating_category_numeric_mapping = {'adults': 1, 'all_age': 2,
'parental_guide': 3, 'children_above_7': 4, 'children_above_14': 5,
'not_rated': 6}
data['rating_category_numeric'] =
data['rating_category'].map(rating_category_numeric_mapping)
```

```
#Q5.c. Trend of movies and shows based on rating categories ober last
40 years

# Filter data for the last 40 years
recent_data = data[data['release_year'] >= (data['release_year'].max()
- 40)]

# Group data by 'release_year', 'rating_category', and 'type' and
calculate counts
rating_trend = recent_data.groupby(['release_year', 'rating_category',
'type']).size().reset_index(name='count')

# Pivot the table for better plotting
rating_trend_pivot = rating_trend.pivot_table(index=['release_year',
'rating_category'], columns='type', values='count',
fill_value=0).reset_index()

# Plotting the trend over the last 40 years for the number of movies
and TV shows under each rating category
plt.figure(figsize=(14, 8))
sns.lineplot(x='release_year', y='Movie', hue='rating_category',
data=rating_trend_pivot, marker='o', palette='viridis',label='Movies')
sns.lineplot(x='release_year', y='TV Show', hue='rating_category',
data=rating_trend_pivot, marker='o', palette='viridis',
linestyle='dashed',label='TV shows')

plt.title('Trend Over the Last 40 Years: Number of Movies and TV Shows
Under Each Rating Category')
plt.xlabel('Release Year')
plt.ylabel('Count')
plt.legend(title='Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

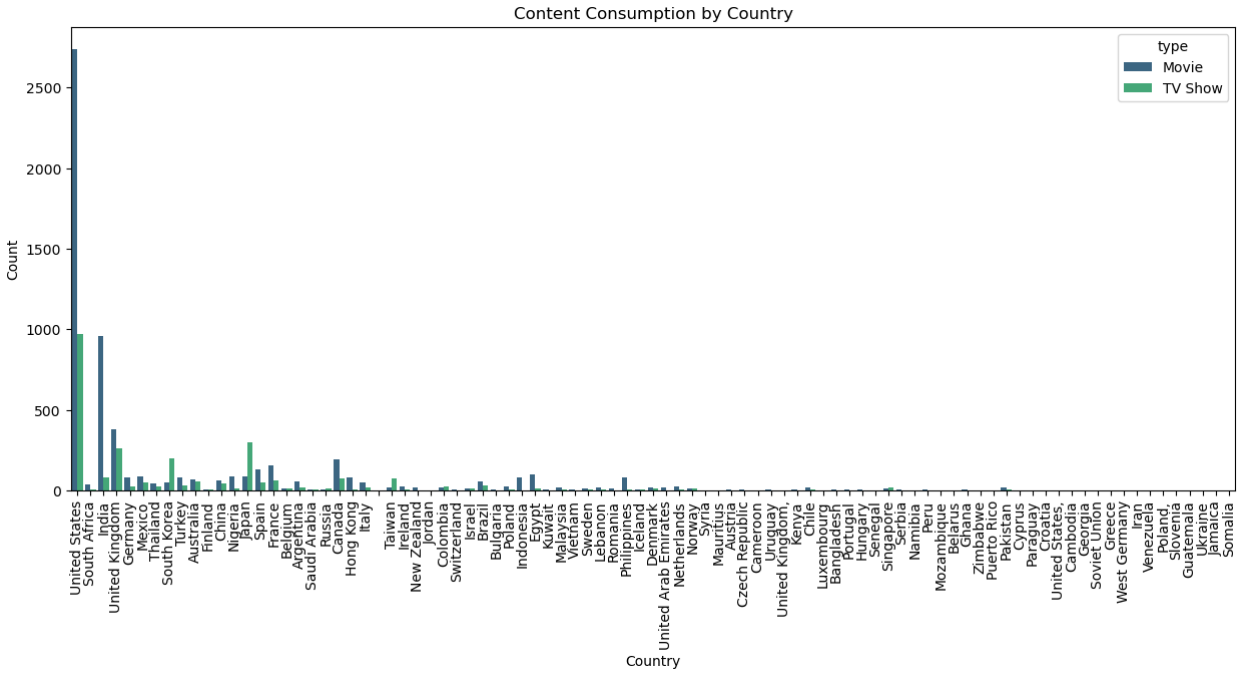
```
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
```

```
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future
version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

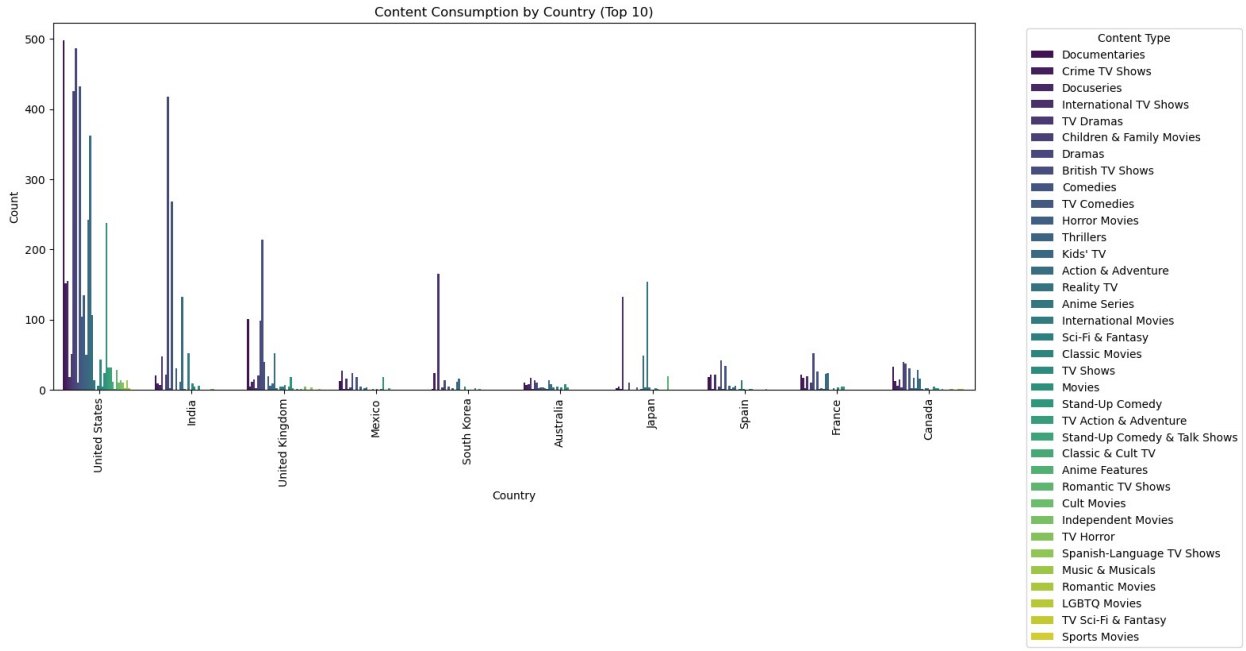
```
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
E:\rasa\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning:
When grouping with a length-1 list-like, you will need to pass a
length-1 tuple to get_group in a future version of pandas. Pass
`(name,)` instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```



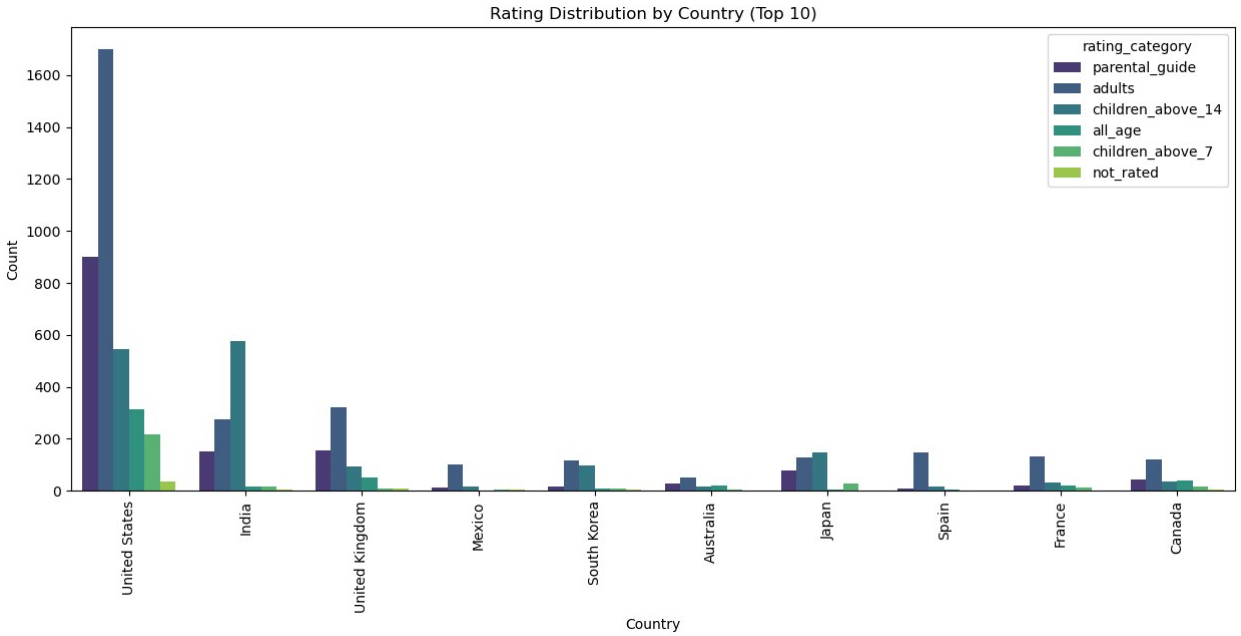

```
#Q.6.a: Content consumption coutry wise
plt.figure(figsize=(15, 6))
sns.countplot(x='country', hue='type', data=data, palette='viridis')
plt.title('Content Consumption by Country')
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```




```
#Q.6.b: Identify the top 10 countries based on their contribution to
content
top_countries = data['country'].value_counts().nlargest(10).index

# Filter the data for the top 10 countries
df_top_countries = data[data['country'].isin(top_countries)]

# Plot content consumption for the top 10 countries
plt.figure(figsize=(15, 6))
ax=sns.countplot(x='country', hue='listed_in', data=df_top_countries,
palette='viridis')
# Move the legend to the left
ax.legend(title='Content Type', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.title('Content Consumption by Country (Top 10)')
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```




```
#Q6.c Identify the top 10 countries based on their contribution to content
top_countries = data['country'].value_counts().nlargest(10).index

# Filter the data for the top 10 countries
df_top_countries = data[data['country'].isin(top_countries)]

# Plot count of rating categories by country
plt.figure(figsize=(15, 6))
ax = sns.countplot(x='country', hue='rating_category',
data=df_top_countries, palette='viridis')

plt.title('Rating Distribution by Country (Top 10)')
plt.xlabel('Country')
plt.ylabel('Count')
plt.xticks(rotation=90)

plt.show()
```




```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import CountVectorizer

import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
True
```

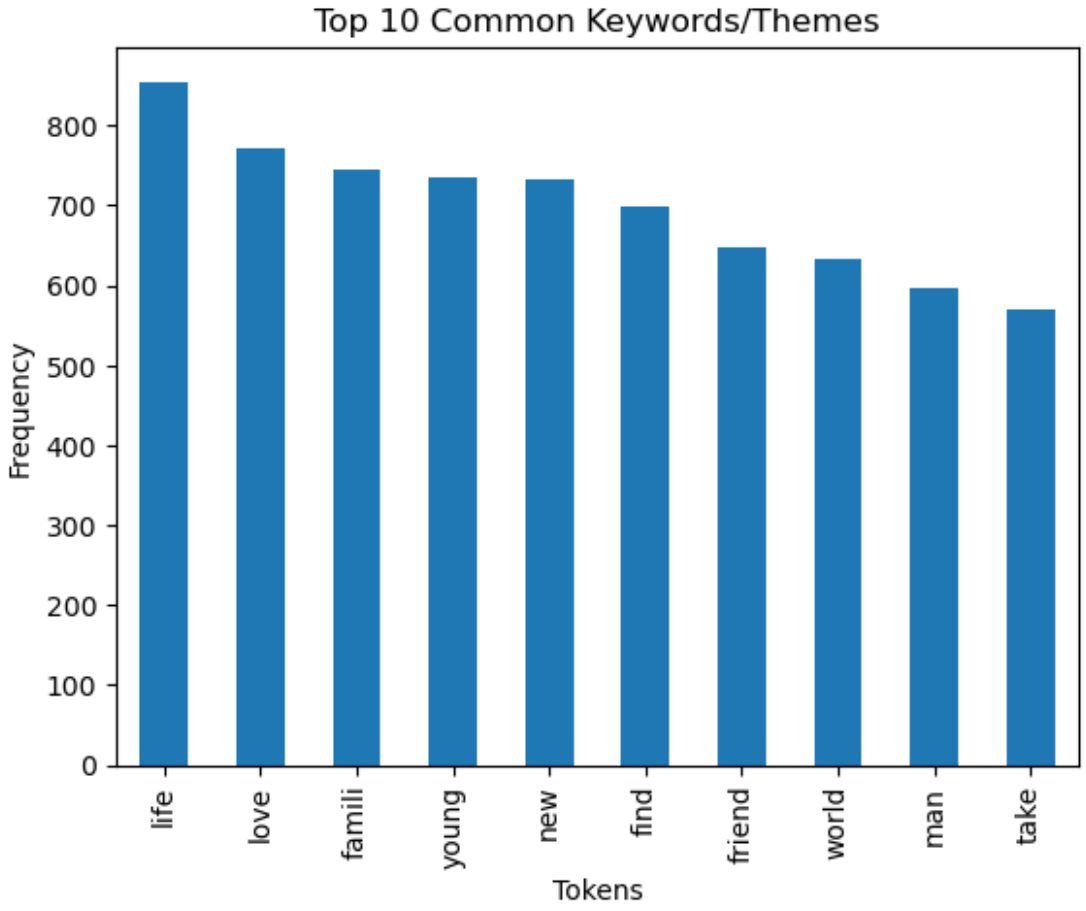
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

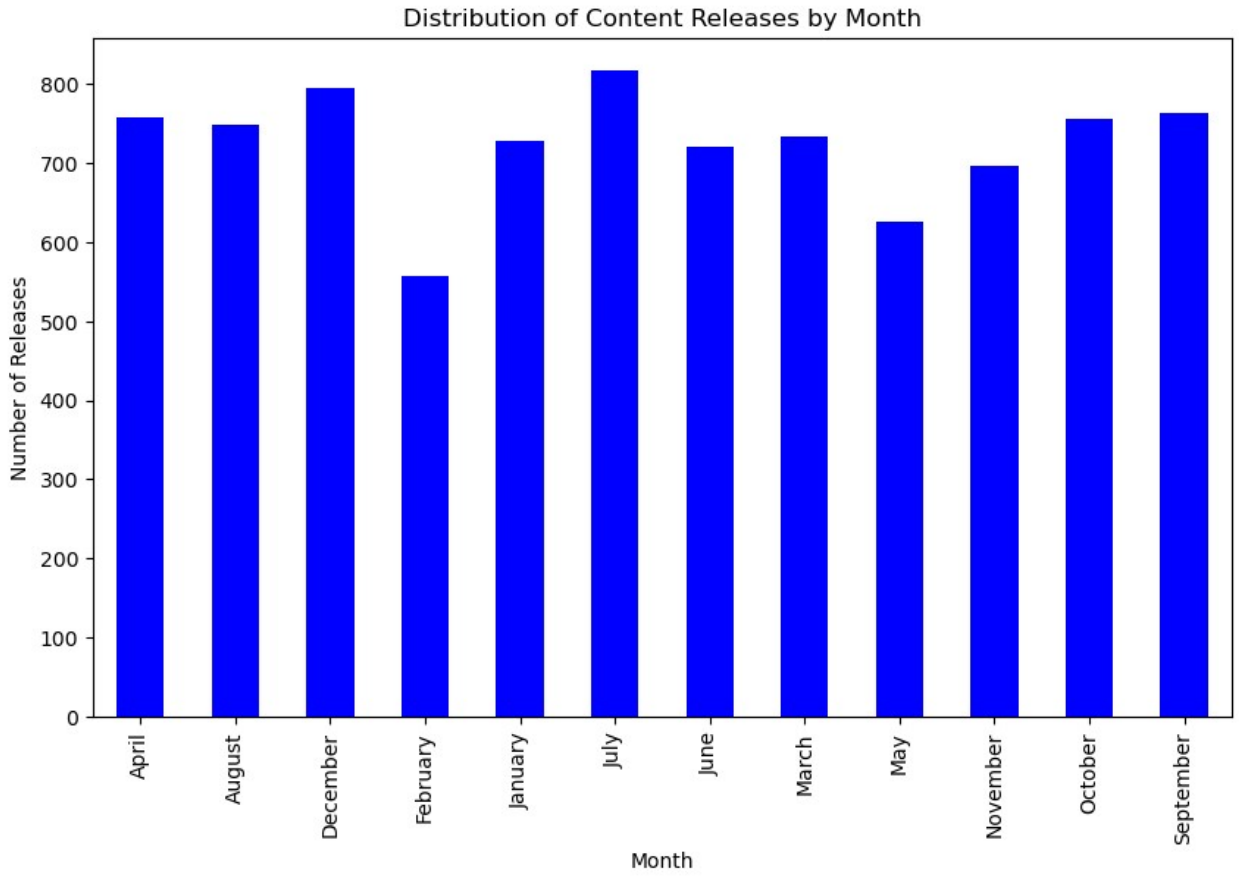
```
True
```

```
#Q7.  What are the most common keywords or themes in show titles and
descriptions
titles_descriptions = data['title'] + ' ' + data['description']

# Tokenization
tokens = titles_descriptions.apply(lambda x: word_tokenize(x.lower()))

# Remove stopwords
stop_words = set(stopwords.words('english'))
```

```
filtered_tokens = tokens.apply(lambda x: [word for word in x if
word.isalnum() and word not in stop_words])

# Stemming
stemmer = PorterStemmer()
stemmed_tokens = filtered_tokens.apply(lambda x: [stemmer.stem(word)
for word in x])

# Flatten the list of tokens
flat_tokens = [item for sublist in stemmed_tokens for item in sublist]

# Create a DataFrame with token counts
token_counts = pd.Series(flat_tokens).value_counts()

# Plot the top N tokens
top_n = 10
token_counts.head(top_n).plot(kind='bar', xlabel='Tokens',
ylabel='Frequency', title='Top 10 Common Keywords/Themes')
plt.show()
```

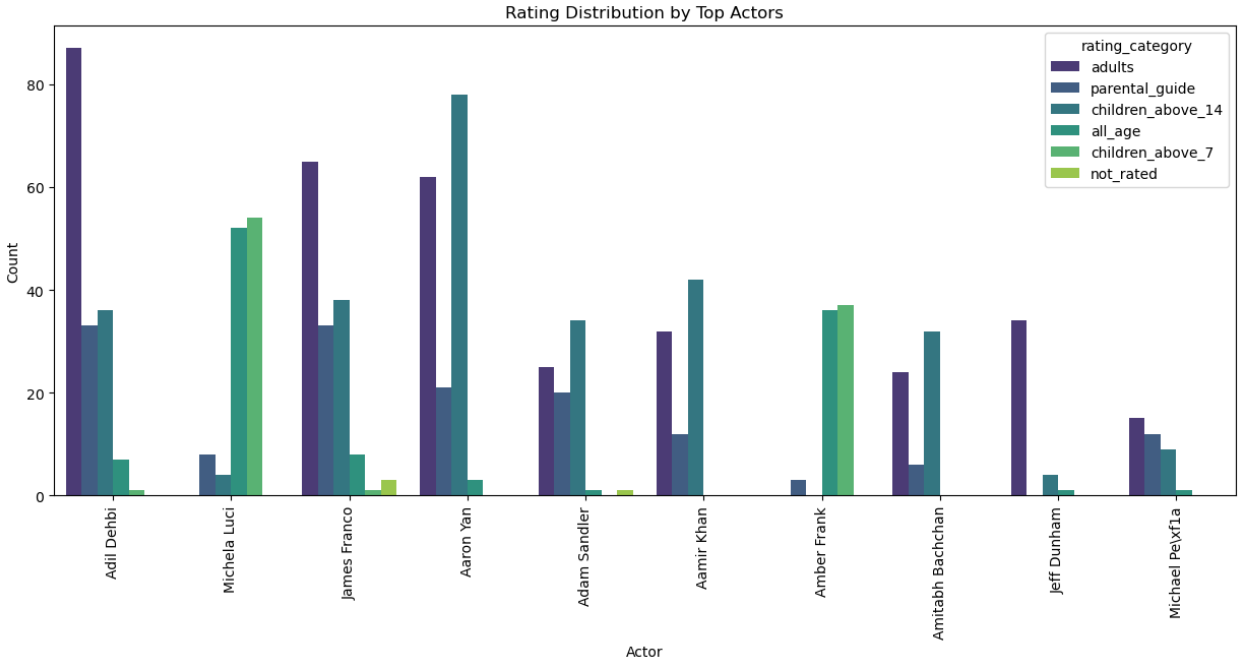


```
#Q8. Is there a seasonality effect on the popularity of content based
on release month
data['release_month'] = data['date_added'].dt.month_name()

# Count the number of content releases per month
monthly_counts = data['release_month'].value_counts().sort_index()

# Plot the distribution of content releases across different months
plt.figure(figsize=(10, 6))
monthly_counts.plot(kind='bar',  color='blue')
plt.title('Distribution of Content Releases by Month')
plt.xlabel('Month')
plt.ylabel('Number of Releases')
plt.show()
```




```
#Q9.a # Identify the top actors based on their contribution to content
top_actors = data['cast'].value_counts().nlargest(10).index

# Filter the data for the top actors
df_top_actors = data[data['cast'].isin(top_actors)]

# Plot count of rating categories by actor
```

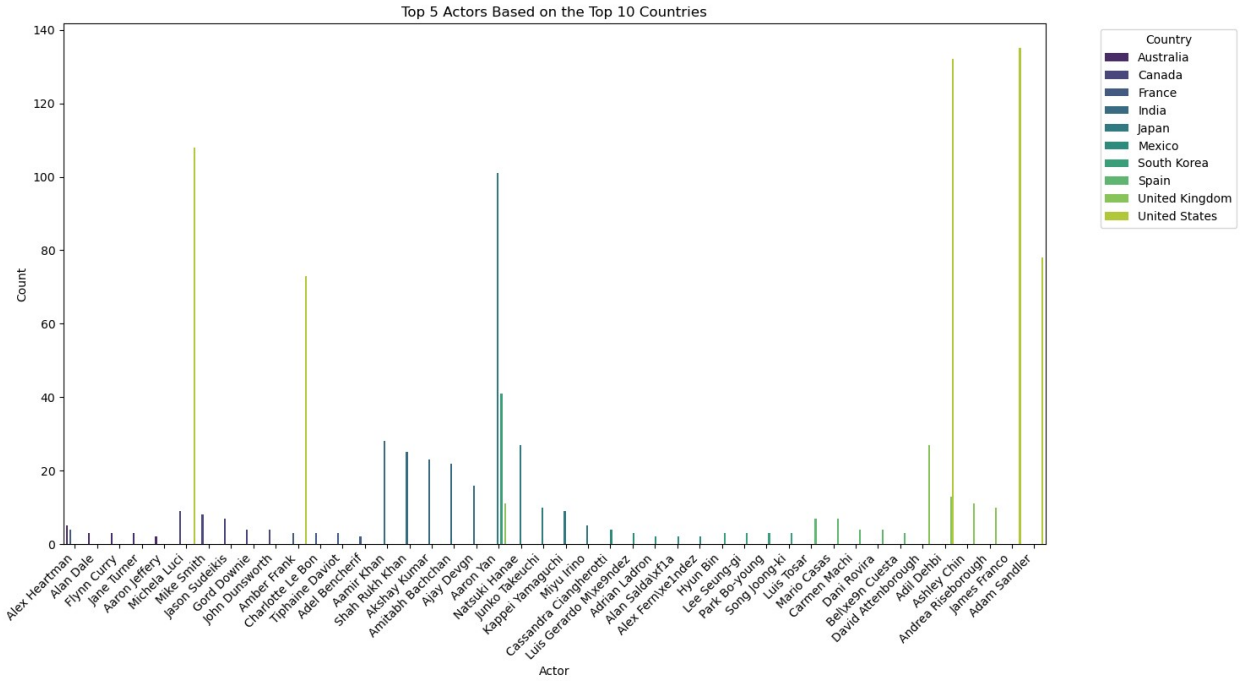
```
plt.figure(figsize=(15, 6))
ax = sns.countplot(x='cast', hue='rating_category',
data=df_top_actors, palette='viridis')

plt.title('Rating Distribution by Top Actors')
plt.xlabel('Actor')
plt.ylabel('Count')
plt.xticks(rotation=90)

plt.show()
```




```
#Q9.b Filter for the top 10 countries
top_countries = data['country'].value_counts().nlargest(10).index
df_top_countries = data[data['country'].isin(top_countries)]

# Group by country and actor, calculate the count
top_actors_by_country = df_top_countries.groupby(['country',
'cast']).size().reset_index(name='count')

# Find the top 5 actors for each country
top_5_actors_by_country =
top_actors_by_country.groupby('country').apply(lambda x: x.nlargest(5,
'count')).reset_index(drop=True)

# Plotting the top 5 actors for each of the top 10 countries
plt.figure(figsize=(15, 8))
sns.barplot(x='cast', y='count', hue='country',
data=top_5_actors_by_country, palette='viridis')
plt.title('Top 5 Actors Based on the Top 10 Countries')
```

```
plt.xlabel('Actor')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Country', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.show()
```

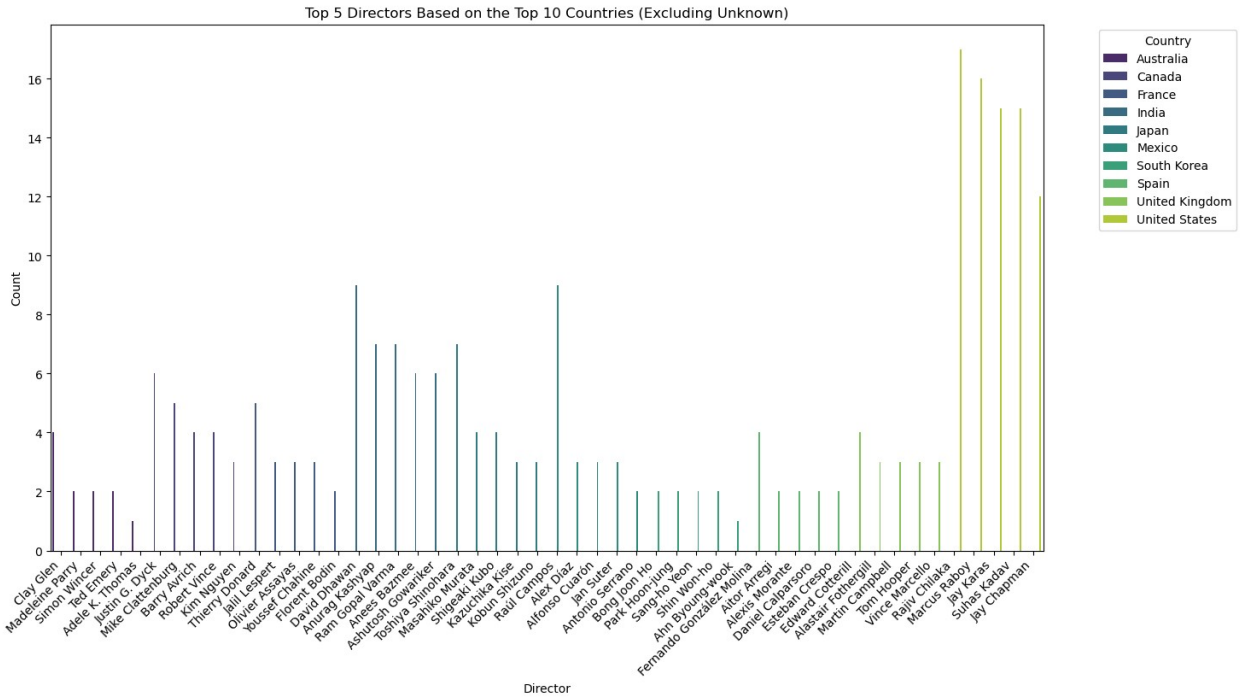
```
C:\Users\user\AppData\Local\Temp\ipykernel_7024\1492064982.py:9:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this
warning.
  top_5_actors_by_country =
top_actors_by_country.groupby('country').apply(lambda x: x.nlargest(5,
'count')).reset_index(drop=True)
```




```
#Q9.vc. Top directors based on top 10 countires
# Filter for the top 10 countries
top_countries = data['country'].value_counts().nlargest(10).index
df_top_countries = data[data['country'].isin(top_countries)]

# Remove rows with unknown directors
df_top_countries = df_top_countries[df_top_countries['director'] !=
'Unknown']

# Group by country and director, calculate the count
```

```
top_directors_by_country = df_top_countries.groupby(['country',
'director']).size().reset_index(name='count')

# Find the top 5 directors for each country
top_5_directors_by_country =
top_directors_by_country.groupby('country').apply(lambda x:
x.nlargest(5, 'count')).reset_index(drop=True)

# Plotting the top 5 directors for each of the top 10 countries
plt.figure(figsize=(15, 8))
sns.barplot(x='director', y='count', hue='country',
data=top_5_directors_by_country, palette='viridis')
plt.title('Top 5 Directors Based on the Top 10 Countries (Excluding
Unknown)')
plt.xlabel('Director')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Country', bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.show()
```

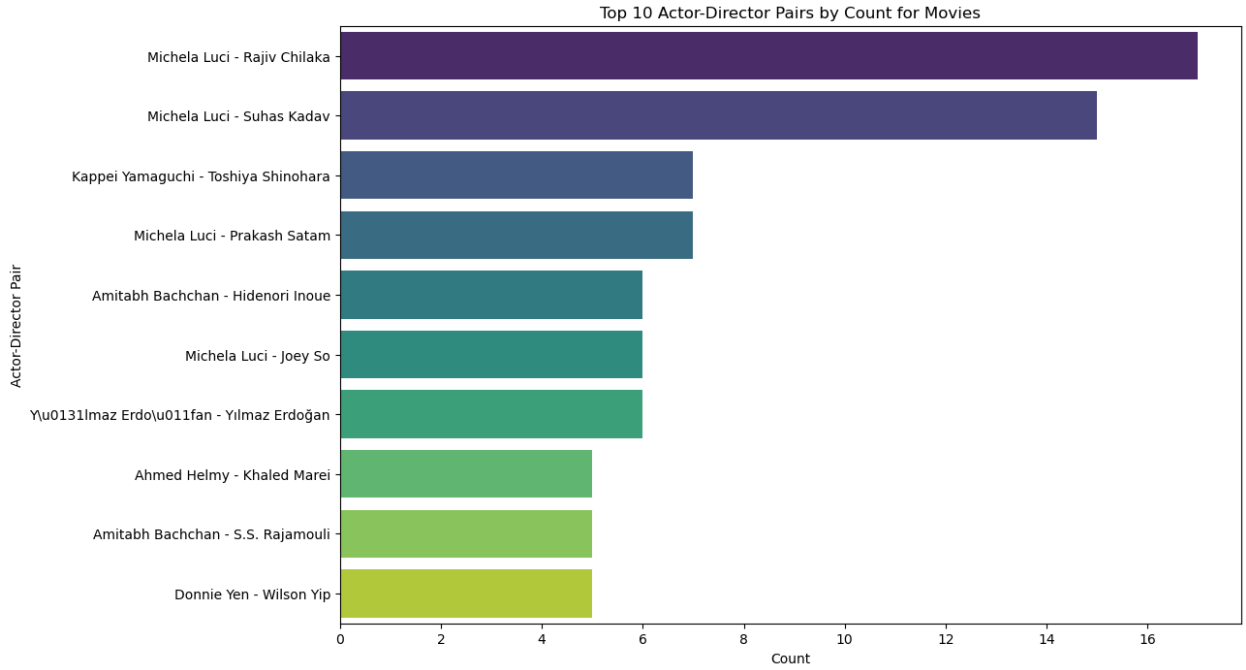
```
C:\Users\user\AppData\Local\Temp\ipykernel_7024\389538803.py:13:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping
columns. This behavior is deprecated, and in a future version of
pandas the grouping columns will be excluded from the operation.
Either pass `include_groups=False` to exclude the groupings or
explicitly select the grouping columns after groupby to silence this
warning.
  top_5_directors_by_country =
top_directors_by_country.groupby('country').apply(lambda x:
x.nlargest(5, 'count')).reset_index(drop=True)
```

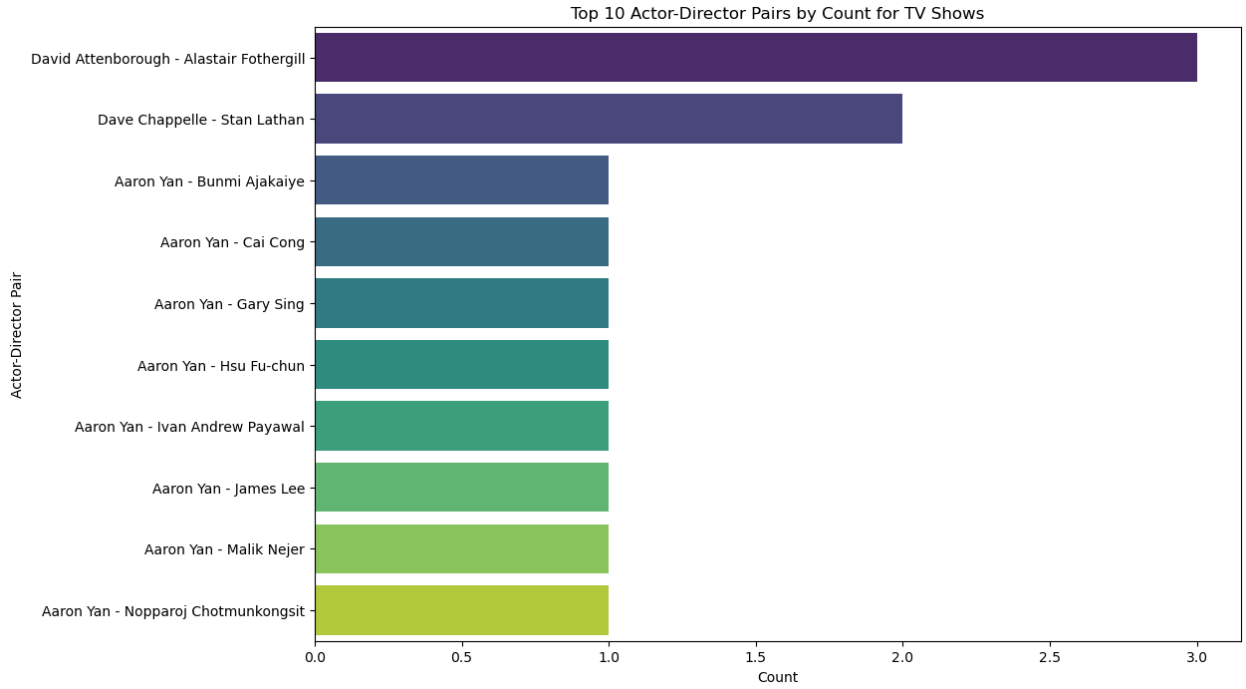



```
#Q9.d: Top actor-director pair
# Filter out rows where 'director' is unknown
df_filtered = data[data['director'] != 'Unknown']

# Concatenate 'cast' and 'director' to create a new column
'actor_director'
df_filtered['actor_director'] = df_filtered['cast'] + ' - ' +
df_filtered['director']

# Group by 'actor_director' and 'type', calculate the count
actor_director_type_count = df_filtered.groupby(['actor_director',
'type']).size().reset_index(name='count')

# Top N pairs by count for each type
top_pairs_movie =
actor_director_type_count[actor_director_type_count['type'] ==
'Movie'].nlargest(10, 'count')
top_pairs_tvshow =
actor_director_type_count[actor_director_type_count['type'] == 'TV
Show'].nlargest(10, 'count')

# Plotting for movies
plt.figure(figsize=(12, 8))
sns.barplot(x='count', y='actor_director', data=top_pairs_movie,
palette='viridis')
plt.title('Top 10 Actor-Director Pairs by Count for Movies')
plt.xlabel('Count')
plt.ylabel('Actor-Director Pair')
```

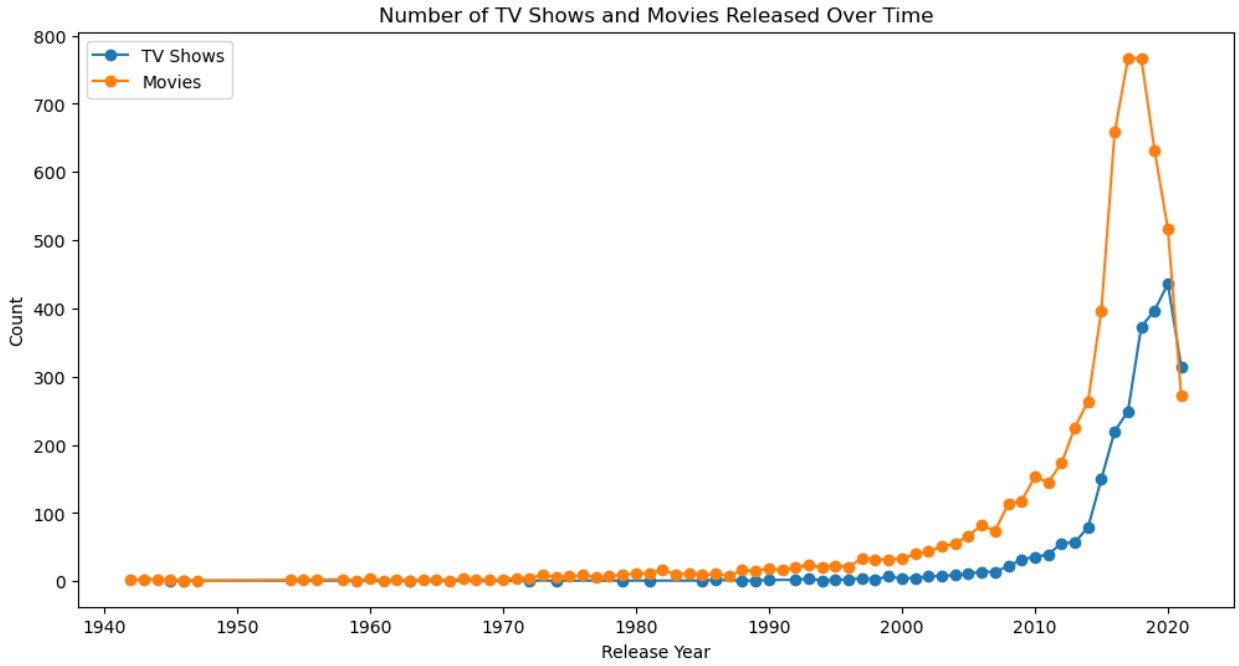
```
plt.show()

# Plotting for TV shows
plt.figure(figsize=(12, 8))
sns.barplot(x='count', y='actor_director', data=top_pairs_tvshow,
palette='viridis')
plt.title('Top 10 Actor-Director Pairs by Count for TV Shows')
plt.xlabel('Count')
plt.ylabel('Actor-Director Pair')
plt.show()
```

```
C:\Users\user\AppData\Local\Temp\ipykernel_7024\730161647.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  df_filtered['actor_director'] = df_filtered['cast'] + ' - ' +
df_filtered['director']
```



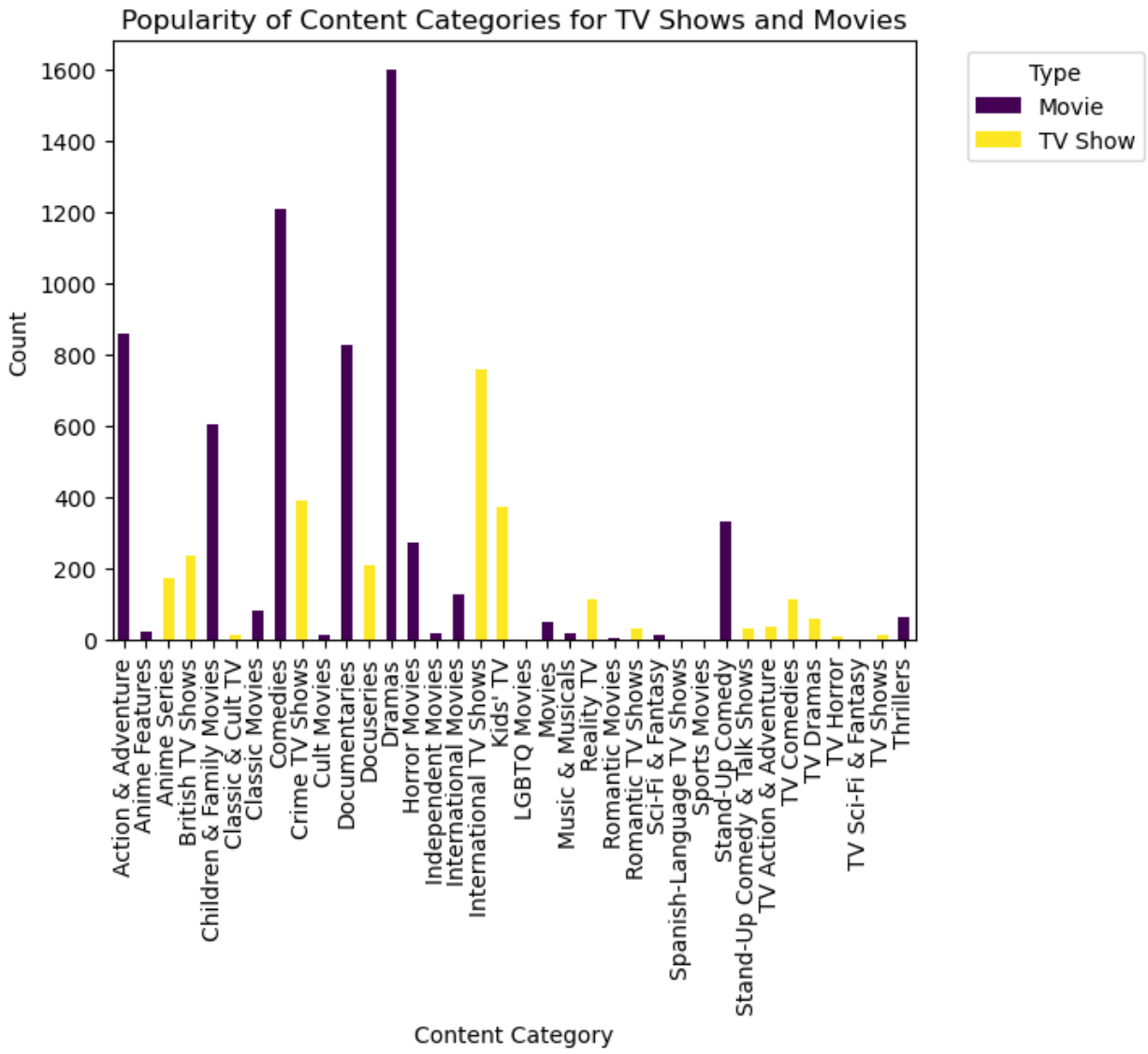



```
#Q10.a. Movies and tv show trend
# Separate TV shows and movies
tv_shows = data[data['type'] == 'TV Show']
movies = data[data['type'] == 'Movie']

# Group by release year
tv_shows_by_year = tv_shows.groupby('release_year').size()
movies_by_year = movies.groupby('release_year').size()

# Plotting trends
plt.figure(figsize=(12, 6))
plt.plot(tv_shows_by_year.index, tv_shows_by_year.values, label='TV
Shows', marker='o')
plt.plot(movies_by_year.index, movies_by_year.values, label='Movies',
marker='o')
plt.title('Number of TV Shows and Movies Released Over Time')
plt.xlabel('Release Year')
plt.ylabel('Count')
plt.legend()
plt.show()
```




```
# Explore content categories and their popularity
category_popularity = data.groupby(['listed_in',
'type']).size().unstack()

# Plotting
plt.figure(figsize=(14, 8))
category_popularity.plot(kind='bar', stacked=True, colormap='viridis')
plt.title('Popularity of Content Categories for TV Shows and Movies')
plt.xlabel('Content Category')
plt.ylabel('Count')
plt.legend(title='Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```


Popularity of Content Categories for TV Shows and Movies
Type
Movie
TV Show
Count
0
200
400
600
800
1000
1200
1400
1600
Action & Adventure
Anime Features
Anime Series
British TV Shows
Children & Family Movies
Classic & Cult TV
Classic Movies
Comedies
Crime TV Shows
Cult Movies
Documentaries
Docuseries
Dramas
Horror Movies
Independent Movies
International Movies
International TV Shows
Kids' TV
LGBTQ Movies
Movies
Music & Musicals
Reality TV
Romantic Movies
Romantic TV Shows
Sci-Fi & Fantasy
Spanish-Language TV Shows
Sports Movies
Stand-Up Comedy
Stand-Up Comedy & Talk Shows
TV Action & Adventure
TV Comedies
TV Dramas
TV Horror
TV Sci-Fi & Fantasy
TV Shows
Thrillers
Content Category