netflix

May 10, 2025

Business Problem:

Analyze the data and generate insights that could help Netflix in deciding which type of shows/movies to produce, which geners are more popular, what is the best time to launch a Movie or TV Show. Based on which Netflix can plan how they can grow the business in different countries.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

1 1. Basic Metrics:

[2]:
```
netflix = pd.read_csv('netflix.csv')
netflix.head()
```

[2]:
```
  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…
```

[3]:
```
netflix.shape
```

[3]:
```
(8807, 12)
```

[4]:
```
netflix.columns
```

[4]:
```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description'],
      dtype='object')
```

[5]:
```
netflix.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
```

[6]:
```
netflix.describe(include='all')
```

[6]:

	show_id	type	title	director	cast
count	8807	8807	8807	6173	7982
unique	8807	2	8807	4528	7692
top	s8807	Movie	Zubaan	Rajiv Chilaka	David Attenborough
freq	1	6131	1	19	19
mean	NaN	NaN	NaN	NaN	NaN
std	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

	country	date_added	release_year	rating	duration
count	7976	8797	8807.000000	8803	8804
unique	748	1767	NaN	17	220
top	United States	January 1, 2020	NaN	TV-MA	1 Season
freq	2818	109	NaN	3207	1793
mean	NaN	NaN	2014.180198	NaN	NaN
std	NaN	NaN	8.819312	NaN	NaN
min	NaN	NaN	1925.000000	NaN	NaN
25%	NaN	NaN	2013.000000	NaN	NaN
50%	NaN	NaN	2017.000000	NaN	NaN
75%	NaN	NaN	2019.000000	NaN	NaN
max	NaN	NaN	2021.000000	NaN	NaN

	listed_in
count	8807
unique	514
top	Dramas, International Movies
freq	362
mean	NaN
std	NaN
min	NaN
25%	NaN
50%	NaN
75%	NaN
max	NaN

	description
count	8807
unique	8775
top	Paranormal activity at a lush, abandoned prope…
freq	4
mean	NaN
std	NaN
min	NaN

```
25%                                                    NaN
50%                                                    NaN
75%                                                    NaN
max                                                    NaN
```

2 2. Missing Value & Conversion of Attributes:

2.0.1 2.1 Handling Missing Value:

[7]:
```
''' Here We have observed some discripancies where values of 'duration' are␣
 ↪filled in rating
 and duration is left blank hence making the correction '''
netflix.loc[netflix['title'].str.contains('Louis C.K.', na=False), 'rating'] =␣
 ↪np.nan
netflix.loc[netflix['title'].str.contains('Louis C.K. 2017', na=False),␣
 ↪'duration'] = '74 min'
netflix.loc[netflix['title'].str.contains('Louis C.K.: Hilarious', na=False),␣
 ↪'duration'] = '84 min'
netflix.loc[netflix['title'].str.contains('Louis C.K.: Live at the Comedy␣
 ↪Store', na=False), 'duration'] = '66 min'
```

[8]:
```
# Removing leading and trailing spaces and Converting 'date_added' column␣
 ↪datatype to datetime
date_added = netflix['date_added'].str.lstrip()
netflix['date_added'] = date_added
netflix['date_added'] = pd.to_datetime(netflix['date_added'], errors='coerce')
```

[9]:
```
# Checking The Missing Values:
missing_values = netflix.isnull().sum()
missing_values[missing_values > 0]
```

[9]:
```
director      2634
cast           825
country        831
date_added      10
rating           7
dtype: int64
```

[10]:
```
# Replacing the missing values by respective defaults:
netflix['date_added'].fillna(pd.to_datetime(netflix['release_year']),␣
 ↪inplace=True)
netflix['rating'].fillna( netflix['rating'].mode()[0], inplace=True)
netflix['director'].fillna('Unknown', inplace=True)
netflix['cast'].fillna('Unknown', inplace=True)
netflix['country'].fillna('Unknown', inplace=True)
```

```
<ipython-input-10-6e6718a39b9a>: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.


  netflix['date_added'].fillna(pd.to_datetime(netflix['release_year']),
inplace=True)
<ipython-input-10-6e6718a39b9a>:3: 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.


  netflix['rating'].fillna( netflix['rating'].mode()[0], inplace=True)
<ipython-input-10-6e6718a39b9a>:4: 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.


  netflix['director'].fillna('Unknown', inplace=True)
<ipython-input-10-6e6718a39b9a>:5: 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.


```

```
  netflix['cast'].fillna('Unknown', inplace=True)
<ipython-input-10-6e6718a39b9a>:6: 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.


```

```
  netflix['country'].fillna('Unknown', inplace=True)
```

[11]:
```
missing_values = netflix.isnull().sum()
missing_values[missing_values > 0]
```

[11]:
```
Series([], dtype: int64)
```

2.0.2 2.2 Conversion Of Attributes:

[12]:
```
# Optional: Split 'listed_in' genres into lists (for multi-genre analysis)
netflix['genres'] = netflix['listed_in'].str.split(', ').tolist()
```

[13]:
```
# Convert to category types
netflix['type'] = netflix['type'].astype('category')
netflix['rating'] = netflix['rating'].astype('category')
netflix['country'] = netflix['country'].astype('category')
```

[14]:
```
# Extract duration in minutes or number of seasons
# Create separate columns for movie duration
netflix['movie_duration'] = netflix.apply(
lambda x: int(x['duration'].split()[0]) if pd.notnull(x['duration']) and␣
 ↪x['type'] == 'Movie' else None,axis=1)

# For TV shows - extract number of seasons
netflix['Number_of_Seasons'] = netflix.apply(
lambda x: int(x['duration'].split()[0]) if pd.notnull(x['duration']) and␣
 ↪x['type'] == 'TV Show' else None,axis=1)
```

[15]:
```
# Strip whitespaces in 'director', 'cast', 'country'
for col in ['director', 'cast', 'country','genres']:
```

```
    netflix[col] = netflix[col].astype(str).str.strip()
```

2.0.3 2.3 Rechecking Missing Values After Attribute Conversion and New Attribute Creation:

[16]:
```
missing_values = netflix.isnull().sum()
missing_values[missing_values > 0]
```

[16]:
```
movie_duration       2676
Number_of_Seasons    6131
dtype: int64
```

[17]:
```
# Handling the missing Values from the newly created column 'movie_duration'␣
 ↪and 'Number_of_Seasons'
netflix['movie_duration'].fillna(0, inplace=True)
netflix['Number_of_Seasons'].fillna(0, inplace=True)
```

```
<ipython-input-17-eeb51233c2e5>: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.


  netflix['movie_duration'].fillna(0, inplace=True)
<ipython-input-17-eeb51233c2e5>:3: 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.


  netflix['Number_of_Seasons'].fillna(0, inplace=True)
```

[18]:
```
missing_values = netflix.isnull().sum()
missing_values[missing_values > 0]
```

[18]:
```
Series([], dtype: int64)
```

[19]:
```
netflix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8807 entries, 0 to 8806
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   show_id            8807 non-null   object
 1   type               8807 non-null   category
 2   title              8807 non-null   object
 3   director           8807 non-null   object
 4   cast               8807 non-null   object
 5   country            8807 non-null   object
 6   date_added         8807 non-null   datetime64[ns]
 7   release_year       8807 non-null   int64
 8   rating             8807 non-null   category
 9   duration           8807 non-null   object
 10  listed_in          8807 non-null   object
 11  description        8807 non-null   object
 12  genres             8807 non-null   object
 13  movie_duration     8807 non-null   float64
 14  Number_of_Seasons  8807 non-null   float64
dtypes: category(2), datetime64[ns](1), float64(2), int64(1), object(9)
memory usage: 912.6+ KB
```

3 3. Outliers:

3.0.1 3.1 Checking For Outliers:

[20]:
```
df_movies = netflix[netflix['type'] == 'Movie'].copy()
df_movies['movie_duration'] = pd.to_numeric(df_movies['duration'].str.
 ↪extract(r'(\d+)')[0])
# Interquartile Range (IQR) method for movie durations
Q1 = df_movies['movie_duration'].quantile(0.25)
Q3 = df_movies['movie_duration'].quantile(0.75)
IQR = Q3 - Q1
# Define bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df_movies[(df_movies['movie_duration'] < lower_bound) |␣
 ↪(df_movies['movie_duration'] > upper_bound)]
outliers[['title', 'movie_duration']].sort_values(by='movie_duration',␣
 ↪ascending=False).head(5)
```

[20]:
```
                          title  movie_duration
4253   Black Mirror: Bandersnatch             312
717   Headspace: Unwind Your Mind             273
2491       The School of Mischief             253
2487                No Longer kids             237
2484           Lock Your Girls In             233
```

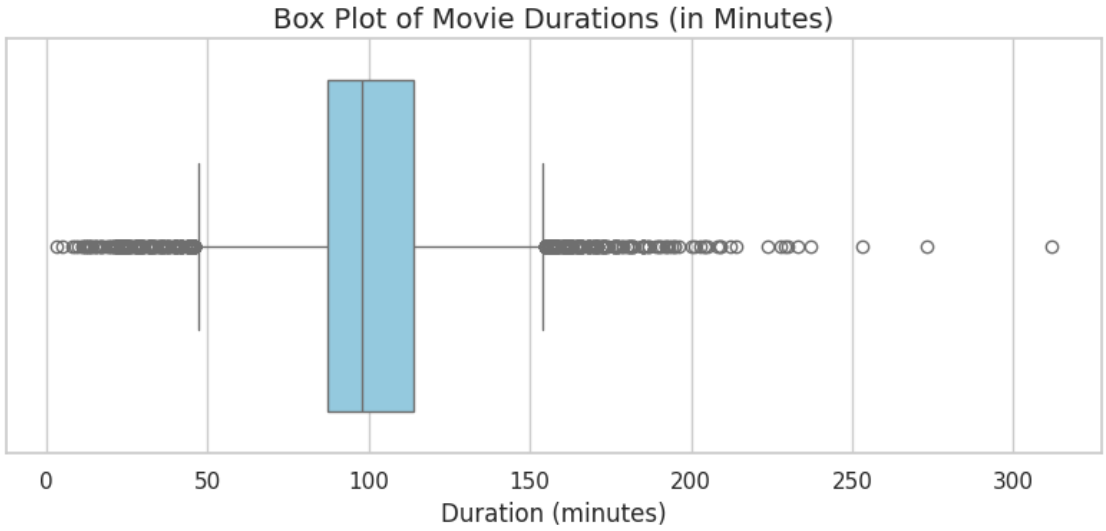
3.0.2 3.2 Upper and Lower Limit Outliers:

[21]:
```
# Finding out top 5 longest and top 5 shortest movies
longest_outliers = df_movies[df_movies['movie_duration'] > upper_bound]
longest_outliers = longest_outliers[['title', 'duration', 'movie_duration']].
 ↪sort_values(by='movie_duration', ascending=False).head(5)

shortest_outliers = df_movies[df_movies['movie_duration'] < lower_bound]
shortest_outliers = shortest_outliers[['title', 'duration', 'movie_duration']].
 ↪sort_values(by='movie_duration').head(5)

outliers = pd.concat([longest_outliers, shortest_outliers])
outliers
```

[21]:
```
                                          title duration  \
4253                      Black Mirror: Bandersnatch  312 min
717                     Headspace: Unwind Your Mind  273 min
2491                         The School of Mischief  253 min
2487                                 No Longer kids  237 min
2484                             Lock Your Girls In  233 min
3777                                         Silent    3 min
2713                                     Sol Levante    5 min
1484                               Cops and Robbers    8 min
1557                                         Canvas    9 min
3535  American Factory: A Conversation with the Obamas   10 min

      movie_duration
4253             312
717              273
2491             253
2487             237
2484             233
3777               3
2713               5
1484               8
1557               9
3535              10
```

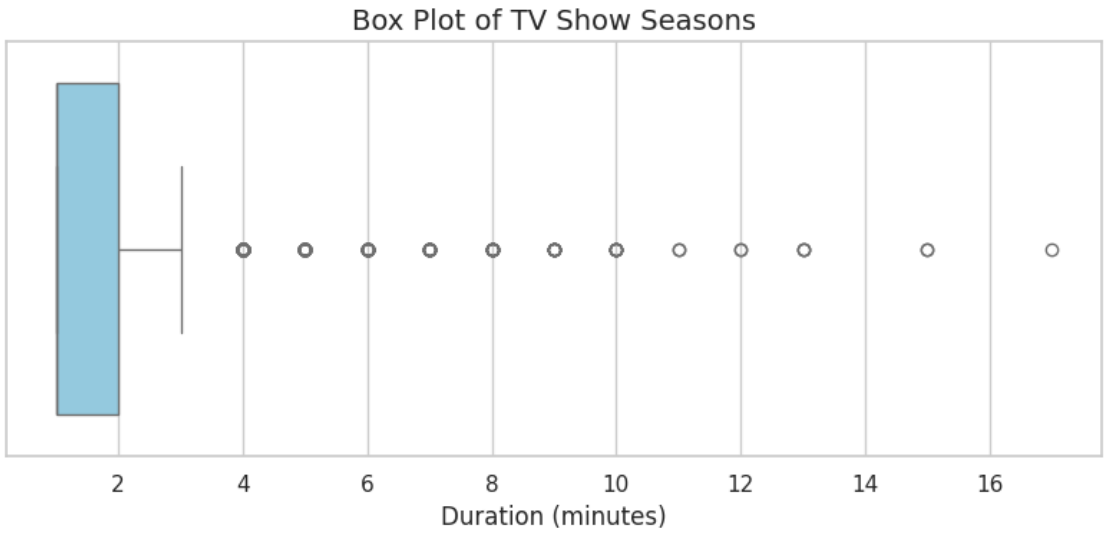
3.0.3 3.3 Visual Analysis [Boxplot] to Indentify Outliers in Movie Durations:

[22]:
```
# Check for outliers in Movie durations
df_movies = netflix[netflix['type'] == 'Movie'].copy()
df_movies['movie_duration'] = pd.to_numeric(df_movies['duration'].str.
 ↪extract(r'(\d+)')[0])

sns.set(style="whitegrid")
plt.figure(figsize=(8, 4))
sns.boxplot(x=df_movies['movie_duration'], color='skyblue')
plt.title('Box Plot of Movie Durations (in Minutes)', fontsize=14)
plt.xlabel('Duration (minutes)', fontsize=12)
plt.tight_layout()
plt.show()
```



3.0.4 3.4 Visual Analysis [Boxplot] to Indentify Outliers in TV Shows Durations:

[23]:
```
df_tv = netflix[netflix['type'] == 'TV Show'].copy()
df_tv['Number_of_Seasons'] = pd.to_numeric(df_tv['duration'].str.
 ↪extract(r'(\d+)')[0])

sns.set(style="whitegrid")
plt.figure(figsize=(8, 4))
sns.boxplot(x=df_tv['Number_of_Seasons'], color='skyblue')
plt.title('Box Plot of TV Show Seasons', fontsize=14)
plt.xlabel('Duration (minutes)', fontsize=12)
plt.tight_layout()
plt.show()
```



3.0.5 3.4 Insights based on Outliers Analysis:

1. The central 50% of movies Interquartile Range falls roughly between 85 and 105 minutes.
2. This indicates that Netflix movies generally follow standard film length conventions (~90 mins).
3. The median movie duration is around 95 minutes, which aligns with industry norms for feature films.

4 4. Non-Graphical Analysis:

4.0.1 4.1 Value Counts:

[24]:
```
netflix['type'].value_counts()
```

[24]:
```
type
Movie      6131
TV Show    2676
Name: count, dtype: int64
```

[25]:
```
netflix['rating'].value_counts()
```

[25]:
```
rating
TV-MA       3214
TV-14       2160
TV-PG        863
R            799
PG-13        490
TV-Y7        334
```

```
TV-Y          307
PG            287
TV-G          220
NR             80
G              41
TV-Y7-FV        6
NC-17           3
UR              3
Name: count, dtype: int64
```

[26]:
```
netflix['release_year'].value_counts().head()
```

[26]:
```
release_year
2018    1147
2017    1032
2019    1030
2020     953
2016     902
Name: count, dtype: int64
```

[27]:
```
# Column 'country' contains nested values. Hence Unnesting and then checking␣
 ↪the value_counts
unnested_country = netflix.copy()
unnested_country['country'] = unnested_country['country'].str.split(",")
unnested_country = unnested_country.explode('country')
unnested_country['country'] = unnested_country['country'].str.strip()
unnested_country['country'].value_counts().head()
```

[27]:
```
country
United States     3690
India             1046
Unknown            831
United Kingdom     806
Canada             445
Name: count, dtype: int64
```

[28]:
```
# Column 'listed_in' contains nested values. Hence Unnesting and then checking␣
 ↪the value_counts
unnested_listing = netflix.copy()
unnested_listing['listed_in'] = unnested_listing['listed_in'].str.split(",")
unnested_listing = unnested_listing.explode('listed_in')
unnested_listing['listed_in'] = unnested_listing['listed_in'].str.strip()
unnested_listing['listed_in'].value_counts().head()
```

[28]:
```
listed_in
International Movies      2752
Dramas                    2427
```

```
Comedies                   1674
International TV Shows     1351
Documentaries               869
Name: count, dtype: int64
```

[29]:
```
# Column 'director' contains nested values. Hence Unnesting and then checking the value_counts
unnested_director = netflix.copy()
unnested_director['director'] = unnested_director['director'].str.split(",")
unnested_director = unnested_director.explode('director')
unnested_director['director'] = unnested_director['director'].str.strip()
unnested_director['director'].value_counts().head()
```

[29]:
```
director
Unknown          2634
Rajiv Chilaka      22
Jan Suter          21
Raúl Campos        19
Suhas Kadav        16
Name: count, dtype: int64
```

[30]:
```
# Column 'cast' contains nested values. Hence Unnesting and then checking the value_counts
unnested_cast = netflix.copy()
unnested_cast['cast'] = unnested_cast['cast'].str.split(", ")
unnested_cast = unnested_cast.explode('cast')
unnested_cast['cast'] = unnested_cast['cast'].str.strip()
unnested_cast['cast'].value_counts().head()
```

[30]:
```
cast
Unknown             825
Anupam Kher          43
Shah Rukh Khan       35
Julie Tejwani        33
Naseeruddin Shah     32
Name: count, dtype: int64
```

[31]:
```
# Removing Default 0.0 as those records represent 'Movie'
Seasons = netflix[netflix['Number_of_Seasons'] != 0]
Seasons['Number_of_Seasons'].value_counts()
```

[31]:
```
Number_of_Seasons
1.0     1793
2.0      425
3.0      199
4.0       95
5.0       65
```

```
6.0       33
7.0       23
8.0       17
9.0        9
10.0       7
13.0       3
12.0       2
15.0       2
11.0       2
17.0       1
Name: count, dtype: int64
```

4.0.2 4.2 Unique Attributes

[32]:
```
unique_counts = netflix.nunique().sort_values(ascending=False)
unique_counts
```

[32]:
```
show_id              8807
title                8807
description          8775
cast                 7693
director             4529
date_added           1722
country               749
genres                514
listed_in             514
duration              220
movie_duration        206
release_year           74
Number_of_Seasons      16
rating                 14
type                    2
dtype: int64
```

[NOTE: Columns 'director', 'country', 'cast', 'listed_in' have multiple values in one cell hence spliting/unnesting and then checking for unique counts]

[33]:
```
director_name = netflix['director'].apply(lambda x:x.split(", ")).tolist()
director_name = pd.DataFrame(director_name, index = netflix['title']).stack().
 ↪reset_index().drop(columns = 'level_1').rename(columns = {0:'director'})

country_name = netflix['country'].apply(lambda x:x.split(", ")).tolist()
country_name = pd.DataFrame(country_name, index = netflix['title']).stack().
 ↪reset_index().drop(columns = 'level_1').rename(columns = {0:'country'})

cast_name = netflix['cast'].apply(lambda x:x.split(", ")).tolist()
```

```
cast_name = pd.DataFrame(cast_name, index = netflix['title']).stack().
 ↪reset_index().drop(columns = 'level_1').rename(columns = {0:'cast'})

listing_name = netflix['listed_in'].apply(lambda x:x.split(", ")).tolist()
listing_name = pd.DataFrame(listing_name, index = netflix['title']).stack().
 ↪reset_index().drop(columns = 'level_1').rename(columns = {0:'listed_in'})

uni_cnt = pd.concat([director_name, country_name, cast_name, listing_name])
uni_cnt.nunique()
```

[33]:
```
title         8807
director      4994
country        128
cast         36440
listed_in       42
dtype: int64
```

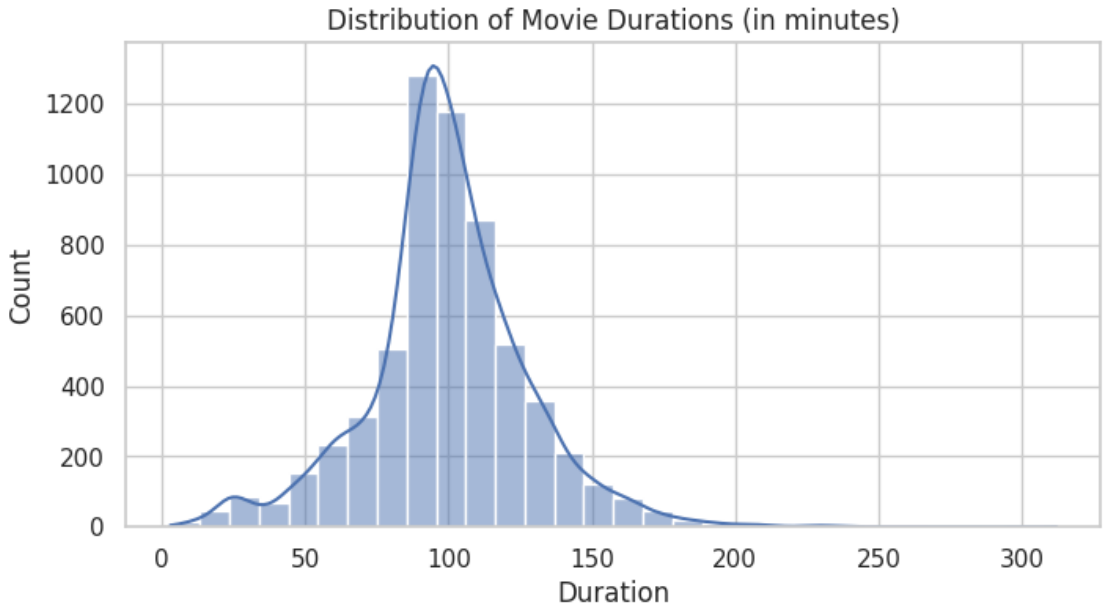
5 5. Visual Analysis:

5.0.1 5.1 Univariate Visual Analysis:

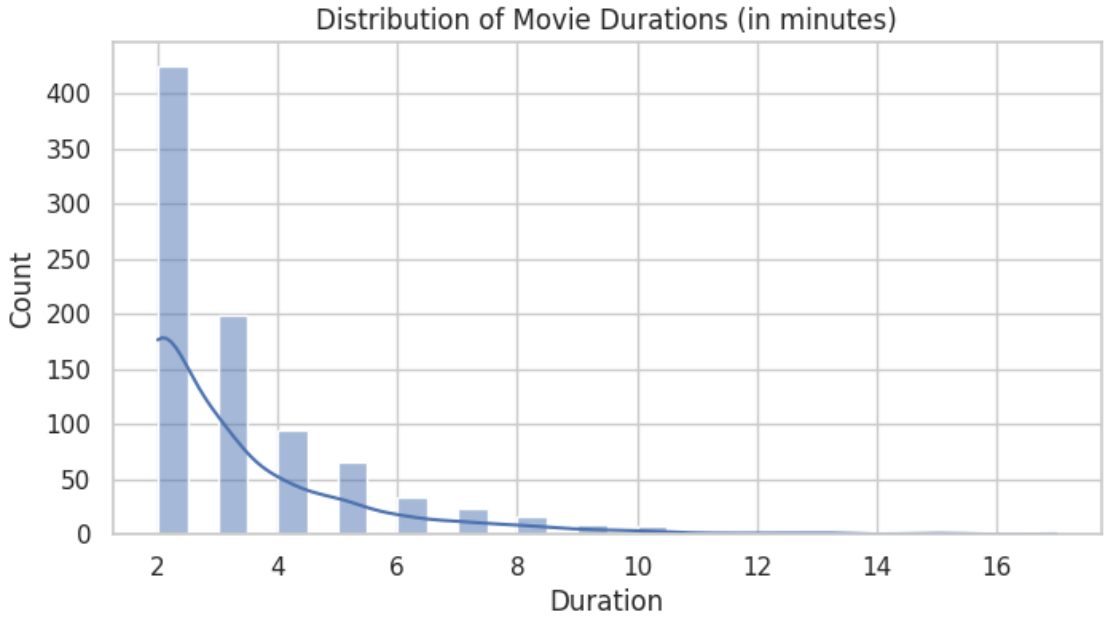
5.1.1 Visualization for Movie/TV Show Duration:

[34]:
```
sns.set(style="whitegrid")
plt.rcParams["figure.figsize"] = (12, 6)
```

[35]:
```
# 1. Histogram for 'Movie' Duration
movie_df = netflix[netflix['duration'].str.contains('min', na=False)]

plt.figure(figsize=(8, 4))
sns.histplot(movie_df['movie_duration'].dropna(), bins=30, kde=True)
plt.title("Distribution of Movie Durations (in minutes)")
plt.xlabel("Duration")
plt.ylabel("Count")
plt.show()
```

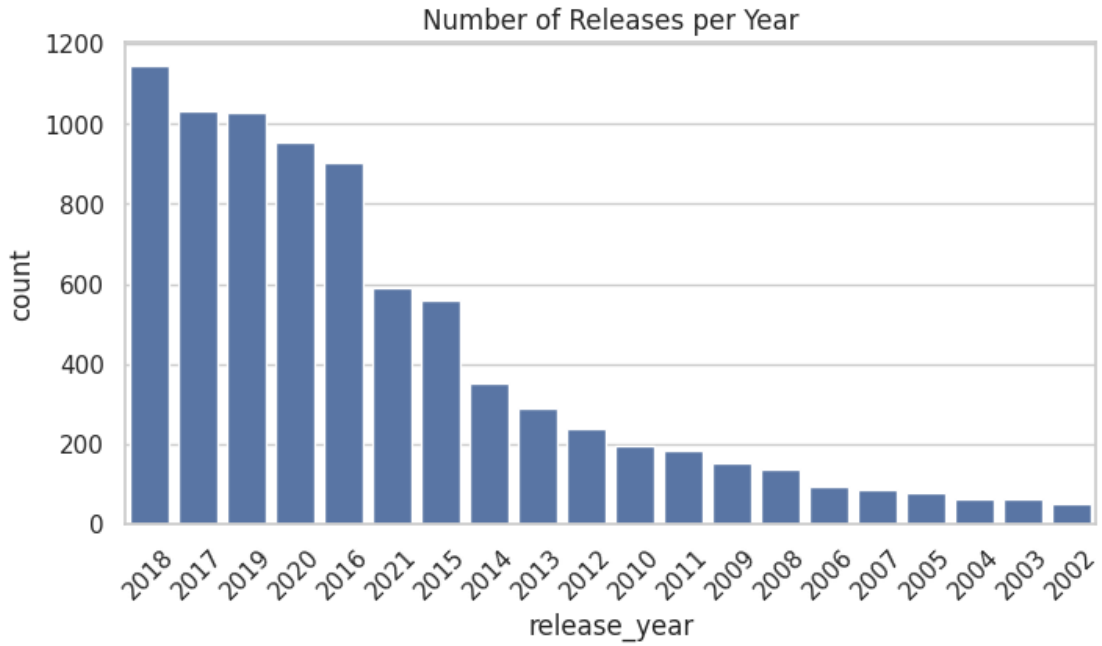


[36]:
```
# 2. Histogram for 'TV Show' Duration
TV_df = netflix[netflix['duration'].str.contains('Seasons', na=False)]

plt.figure(figsize=(8, 4))
sns.histplot(TV_df['Number_of_Seasons'].dropna(), bins=30, kde=True)
plt.title("Distribution of Movie Durations (in minutes)")
plt.xlabel("Duration")
plt.ylabel("Count")
plt.show()
```

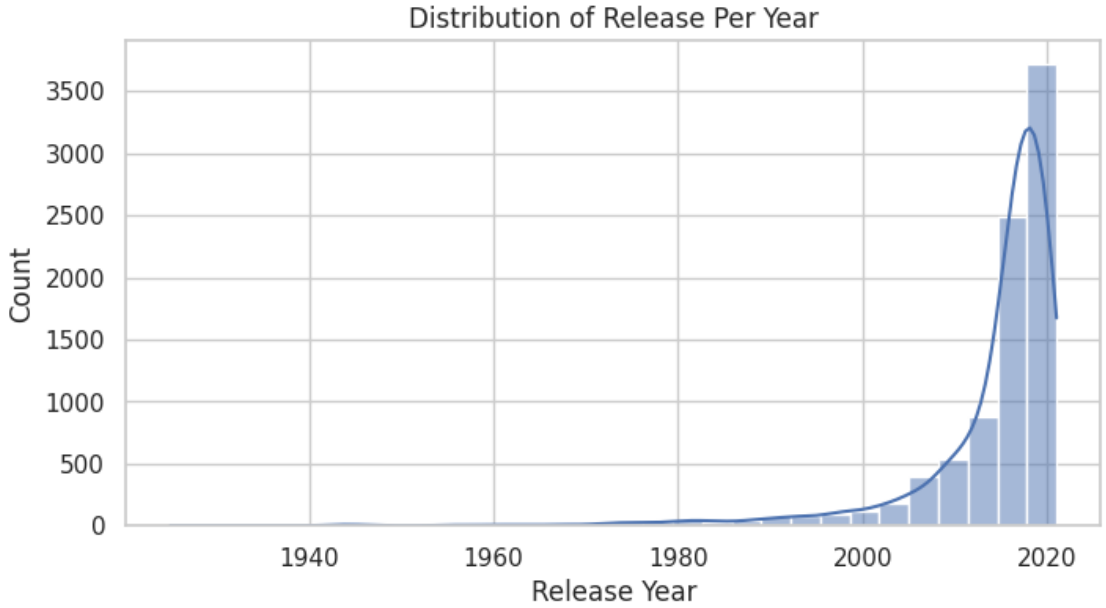


5.1.2 Visualizing No.Of release per year:

[37]:
```
# 1. Using Countplot:
plt.figure(figsize=(8, 4))
sns.countplot(data=netflix, x='release_year', order=netflix['release_year'].
 ↪value_counts().index[:20])
plt.title('Number of Releases per Year')
plt.xticks(rotation=45)
plt.show()
```



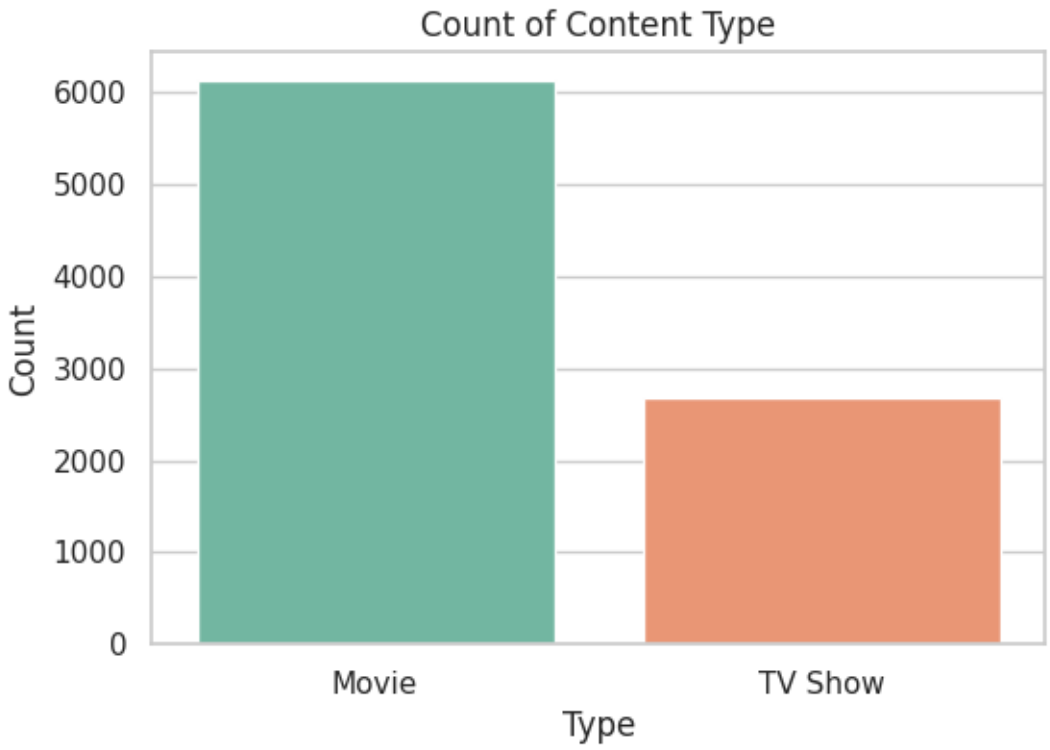
[38]:
```
# 2. Using Histogram:
plt.figure(figsize=(8, 4))
sns.histplot(netflix['release_year'], bins=30, kde=True)
plt.title("Distribution of Release Per Year")
plt.xlabel("Release Year")
plt.ylabel("Count")
plt.show()
```



5.1.3 Countplot of Type (Movie vs TV Show):

[39]:
```
plt.figure(figsize=(6, 4))
sns.countplot(data=netflix, x='type', palette='Set2')
plt.title("Count of Content Type")
plt.xlabel("Type")
plt.ylabel("Count")
plt.show()
```

```
<ipython-input-39-a42abd59e2ed>: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(data=netflix, x='type', palette='Set2')
```

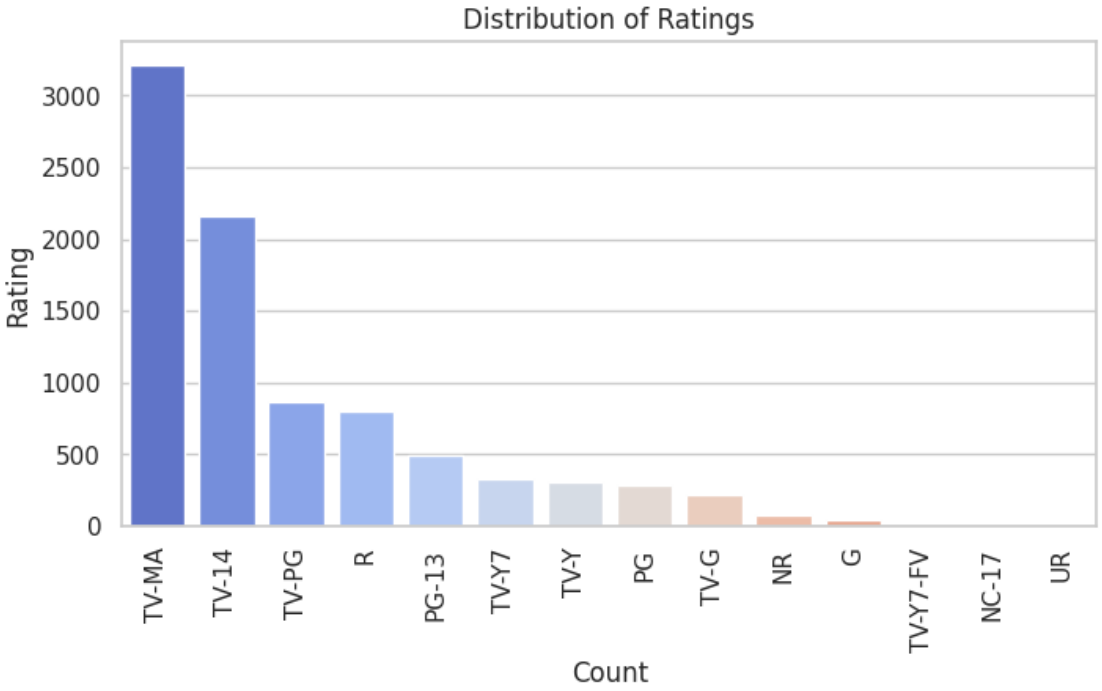


5.1.4 Countplot of Ratings:

[40]:
```
plt.figure(figsize=(8, 4))
```

```
sns.countplot(data=netflix, x='rating', order=netflix['rating'].value_counts().
  ↪index, palette='coolwarm')
plt.title("Distribution of Ratings")
plt.xlabel("Count")
plt.ylabel("Rating")
plt.xticks(rotation=90)
plt.show()
```

```
<ipython-input-40-feac8dbea0ce>: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(data=netflix, x='rating',
order=netflix['rating'].value_counts().index, palette='coolwarm')
```



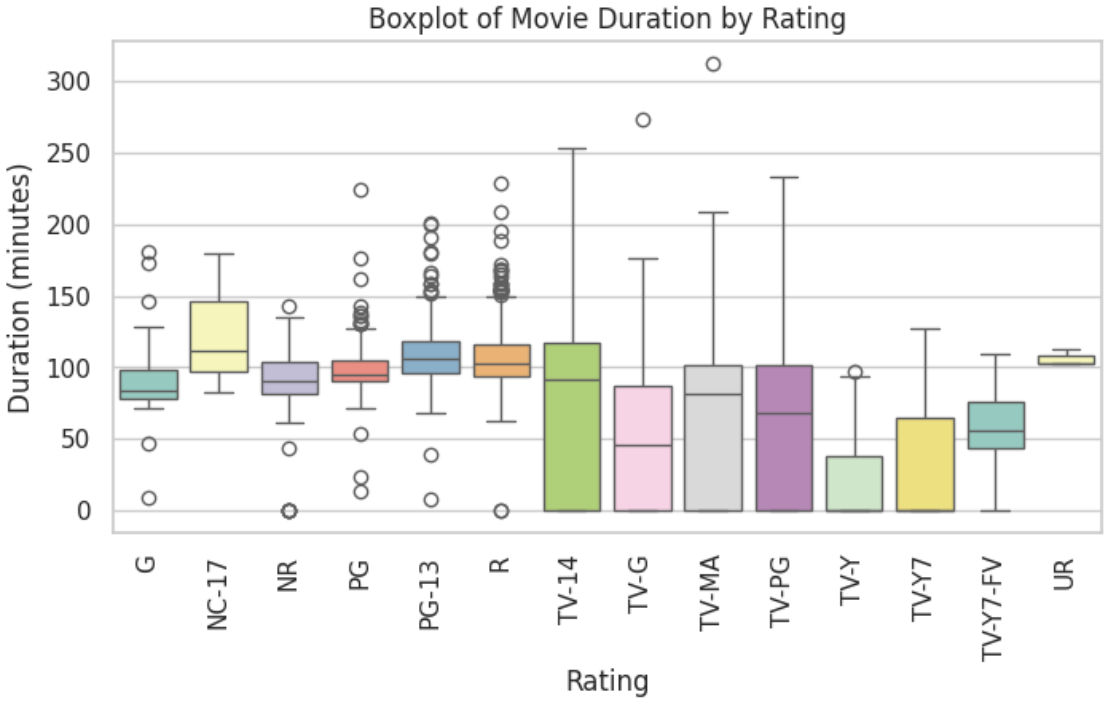
5.0.2 5.2 Bivariate Analysis:

5.2.1 Boxplot of Movie Duration by Rating:

[41]:
```
# 1. Boxplot of Movie Duration by Rating (only for movies with duration in␣
  ↪minutes)
plt.figure(figsize=(8, 4))
sns.boxplot(data=netflix, x='rating', y='movie_duration', palette='Set3')
```

```
plt.title("Boxplot of Movie Duration by Rating")
plt.xlabel("Rating")
plt.ylabel("Duration (minutes)")
plt.xticks(rotation=90)
plt.show()
```

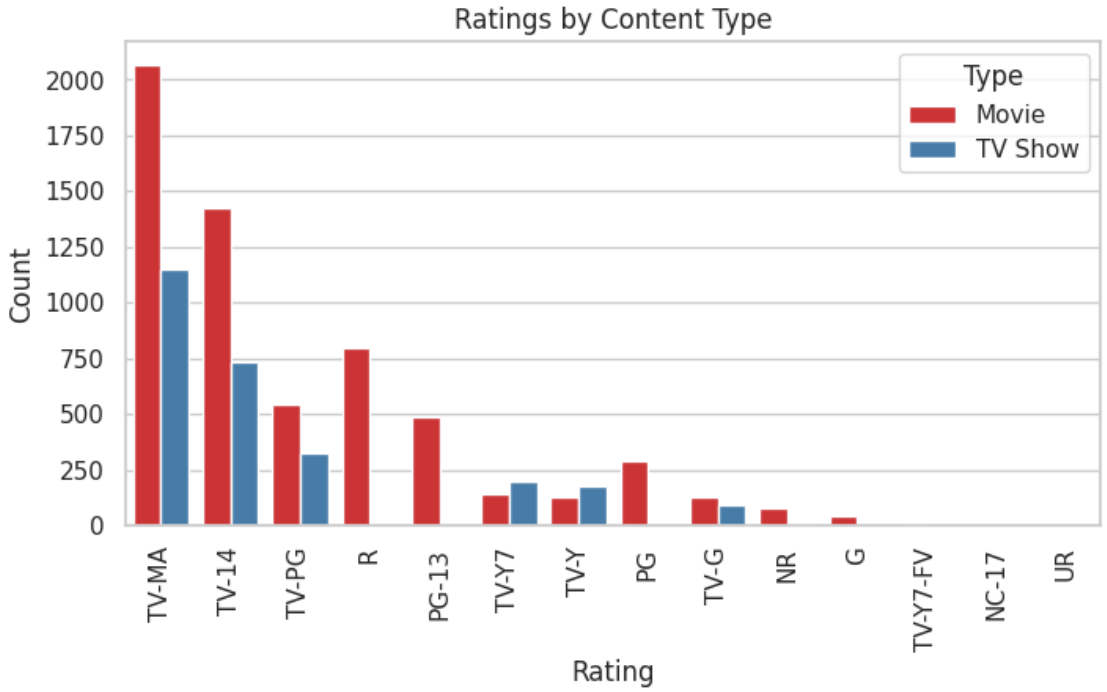
```
<ipython-input-41-7bf211de5524>:3: FutureWarning:

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.boxplot(data=netflix, x='rating', y='movie_duration', palette='Set3')
```



5.2.2 Countplot of Ratings by Type (Movie vs TV Show):

[42]:
```
plt.figure(figsize=(8, 4))
sns.countplot(data=netflix, x='rating', hue='type', palette='Set1',␣
  ↪order=netflix['rating'].value_counts().index)
plt.title("Ratings by Content Type")
plt.xlabel("Rating")
plt.ylabel("Count")
plt.xticks(rotation=90)
plt.legend(title='Type')
```

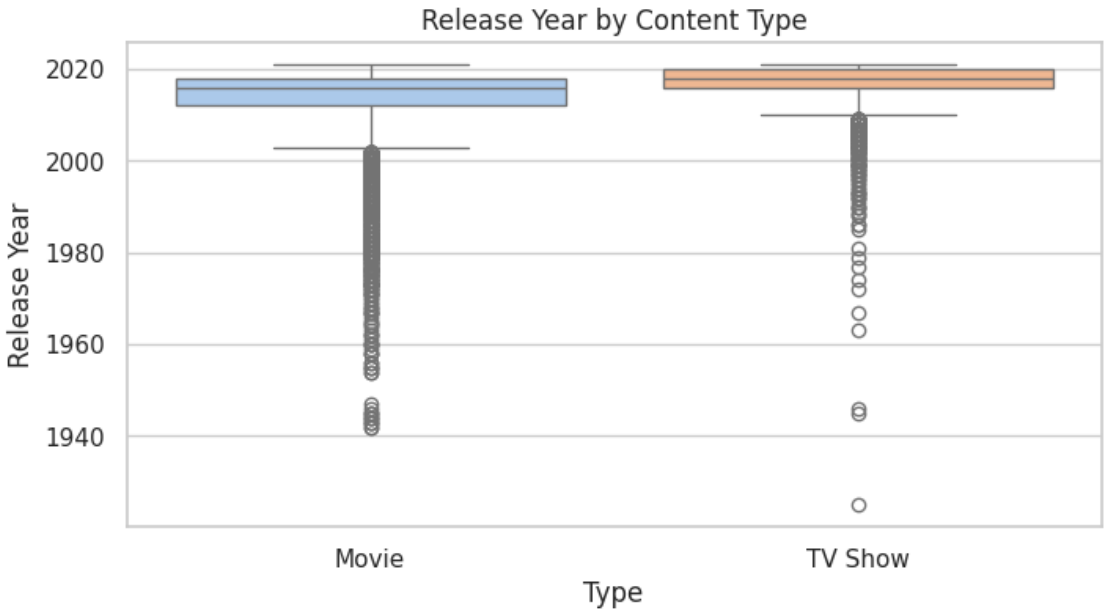
```
plt.show()
```



5.2.3 Boxplot of Release Year by Type:

[43]:
```
plt.figure(figsize=(8, 4))
sns.boxplot(data=netflix, x='type', y='release_year', palette='pastel')
plt.title("Release Year by Content Type")
plt.xlabel("Type")
plt.ylabel("Release Year")
plt.show()
```

```
<ipython-input-43-0bd4772aed7d>: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.boxplot(data=netflix, x='type', y='release_year', palette='pastel')
```



5.2.4 Insights From Bivariate Visual Analysis:

1. Movie Duration by Rating:

- Movies rated PG, PG-13 & TV-PG tend to have slightly higher median durations.
- Mature content (e.g., TV-MA) shows a wider variation in duration.

2. Ratings by Content Type:

- TV-MA is the most frequent rating across both movies and TV shows.
- TV shows tend to dominate categories like TV-Y, TV-Y7 and TV-G, reflecting more family or children-oriented programming.

3. Release Year by Type:

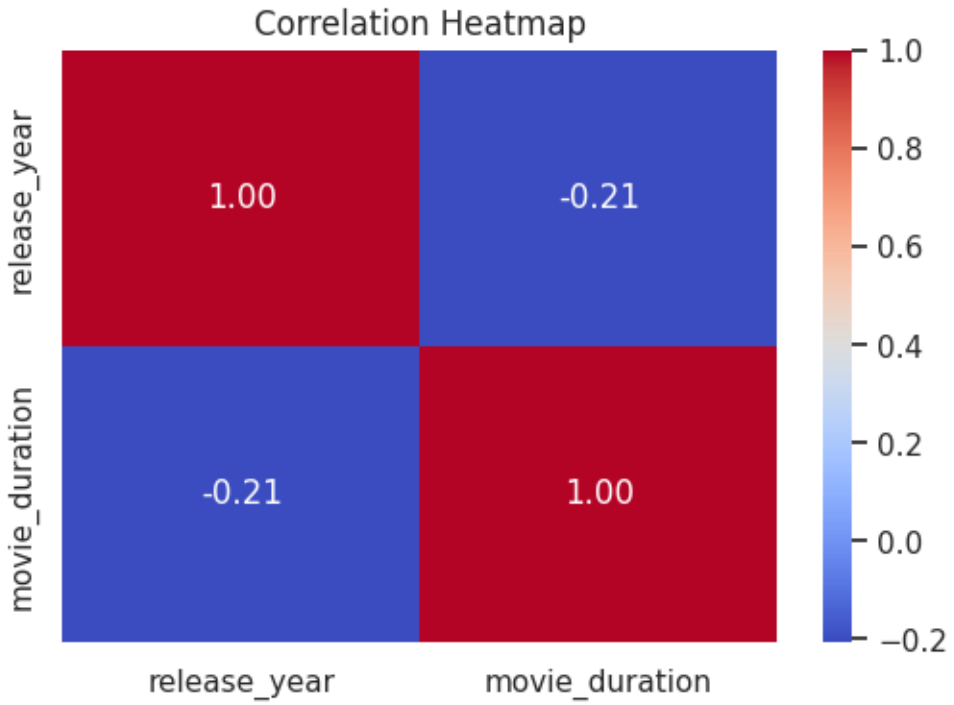
- TV shows generally have more recent release years, likely due to Netflix's recent investments in serialized content.
- Movies have a broader distribution, including more classic or older entries.

5.0.3 5.3 Correlation Analysis:

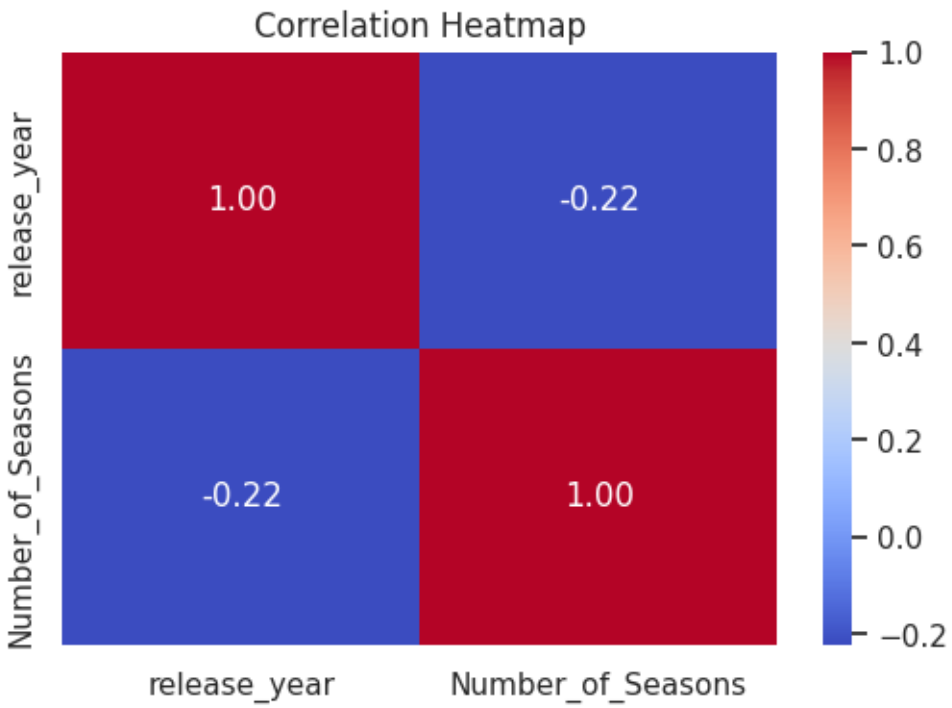
5.3.1 Heatmap of correlations:

[44]:
```
# Checking for correlations between 'release_year' and 'movie_duration' using␣
↪Heatmap
movie_df = netflix[netflix['duration'].str.contains('min', na=False)]
corr = movie_df[['release_year', 'movie_duration']].corr()
plt.figure(figsize=(6, 4))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
```

```
plt.show()
```



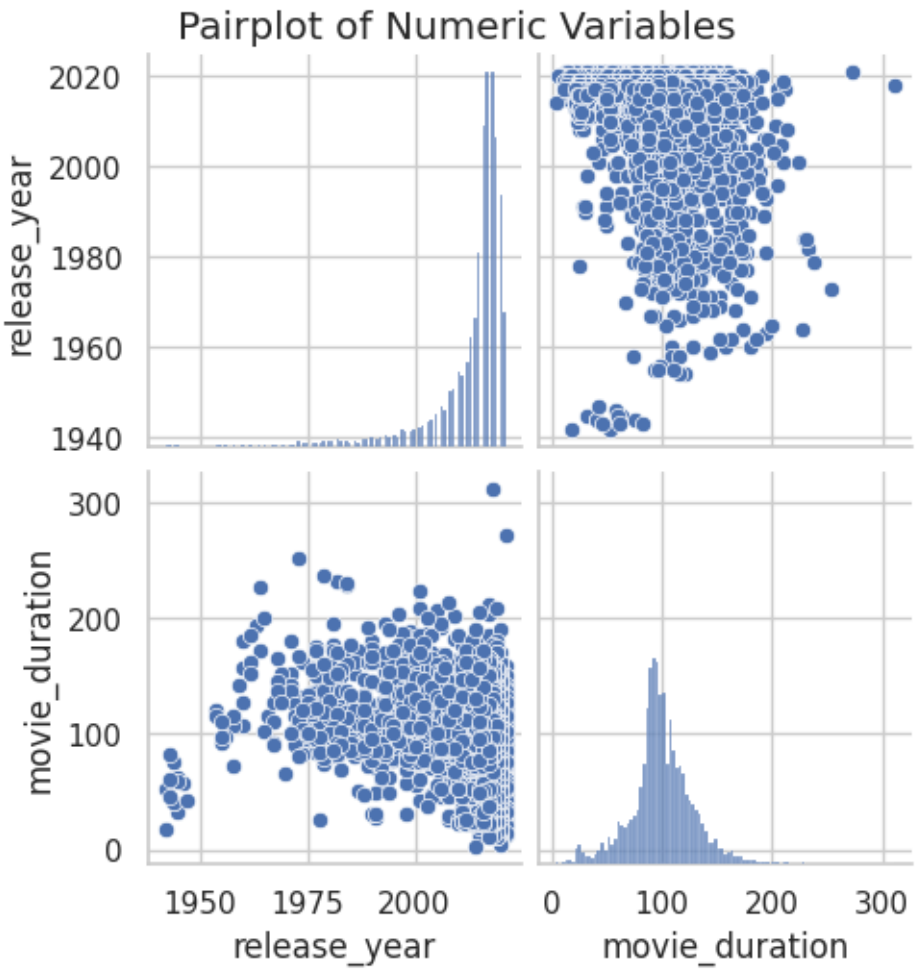
[45]:
```
# Checking for correlations between 'release_year' and 'Number_of_Seasons'␣
 ↪using Heatmap
tv_df = netflix[netflix['duration'].str.contains('Seasons', na=False)]
corr = tv_df[['release_year', 'Number_of_Seasons']].corr()
plt.figure(figsize=(6, 4))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```



5.3.2. Pairplot:

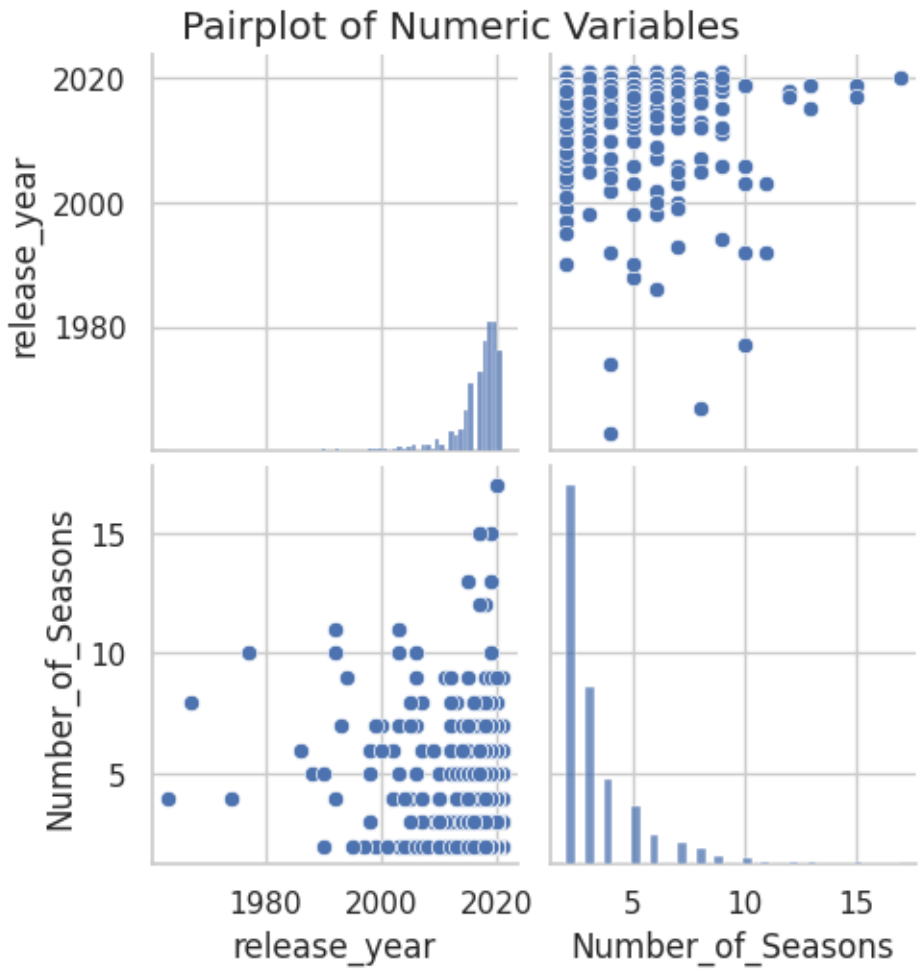
[46]:
```
# Checking for correlations between 'release_year' and 'movie_duration' using␣
↪Pairplot
movie_df = netflix[netflix['duration'].str.contains('min', na=False)]

plt.figure(figsize=(4, 4))
sns.pairplot(movie_df[['release_year', 'movie_duration']].dropna())
plt.suptitle("Pairplot of Numeric Variables", y=1.02)
plt.show()
```

```
<Figure size 400x400 with 0 Axes>
```



[47]:
```
# Checking for correlations between 'release_year' and 'Number_of_Seasons'␣
↪using Pairplot
TV_df = netflix[netflix['duration'].str.contains('Seasons', na=False)]

plt.figure(figsize=(4, 4))
sns.pairplot(TV_df[['release_year', 'Number_of_Seasons']].dropna())
plt.suptitle("Pairplot of Numeric Variables", y=1.02)
plt.show()
```

```
<Figure size 400x400 with 0 Axes>
```



5.3.3 Insights From Visual Correlation Analysis using Heatmap & Pairplot:

- The correlation between release_year and movie_duration is very low (~0.01), indicating no strong linear relationship between how recent a movie/show is and its duration.
- The scatterplot confirms that durations are spread across all release years without a consistent trend.
- No strong correlation between release year and duration, implying that newer content isn't necessarily longer or shorter.

5.0.4 5.4 Insights and Recommendations Based on Visual Analysis:

Strategic Insights: 1. Content Type and Duration Strategy: * Movies dominate in volume, especially with standard durations (80–120 minutes). * TV shows are increasingly recent, indicating Netflix is aggressively expanding serialized content.

2. Target Audience via Ratings:

- Majority of content is rated TV-MA, TV-14, or TV-PG, catering to mature and young adult audiences.

3. Regional Content Growth:

- The country column (unnested earlier) will allow deeper analysis of production by geography.

4. Genre Focus:

- The listed_in genres can be leveraged to detect high-performing themes.

Recommendations for Netflix:

1. Continue focusing on movies in this sweet spot, but experiment with shorter and mid-length films for mobile viewers.
2. Invest in original series production, especially in regions with growing streaming demand.
3. While maintaining mature content, increase family and teen-friendly options (e.g., TV-G, PG) to appeal to broader demographics, especially in international markets.
4. Analyze trends by country to identify underrepresented regions. Expand investment in local-language originals, especially in emerging markets like India, South Korea, and parts of Africa.
5. Evaluate genre frequency and performance to tailor investments — e.g., high demand for "International TV Shows" suggests further diversification.

6 6. Insights based on Non-Graphical and Visual Analysis

6.0.1 6.1 Comments on the Range of Attributes:

1. Type: Two categories — Movie and TV Show.
2. Title: 8,807 unique titles.
3. Director / Cast: Highly variable; some entries have no data, others include multiple individuals.
4. Country: Multi-valued; primarily U.S., India, U.K., but many entries missing or involving multiple countries.
5. Date Added: Mostly post-2015, indicating a growth phase.
6. Release Year: Ranges from early 1900s to 2021; most content is from 2000s onward.
7. Rating: Contains over a dozen TV/movie ratings; most common include TV-MA, TV-14, TV-PG.
8. Duration: Dual nature (minutes for movies, seasons for TV shows); numeric values range from 1 minute to over 200.
9. Listed_in: Contains genre-like labels (e.g., "International TV Shows", "Dramas"); multiple values per entry.
10. Description: Textual summary; useful for NLP-based recommendations but not analyzed here.

6.0.2 6.2 Comments on Distribution and Relationships:

1. Distributions:

- Release Year is right-skewed, heavily concentrated after 2010.
- Duration for movies follows a bell-shaped distribution centered around 90–100 minutes.
- Rating is categorical with skew toward mature content.

2. Relationships:

- Low correlation between release_year and duration_int (confirmed by heatmap and pairplot).
- Content type influences release trends — TV shows are more recent.
- Ratings show dependency on content type — e.g., TV-Y mostly applies to TV shows.

6.0.3 6.3 Comments for Each Plot:

1. Univariate Plots:

1. Histogram – Release Year: * Sharp increase in content released after 2010 reflects Netflix's growth and original content investments.
2. Countplot – Type: * Shows that Netflix offers more Movies than TV Shows, though the latter is growing.
3. Histogram – Duration (Movies): * Most movies are around 90–100 minutes, which aligns with standard feature film norms.
4. Countplot – Rating: * Dominance of TV-MA and TV-14 suggests focus on mature/teen content.

2. Bivariate Plots:

1. Boxplot – Duration by Rating (Movies): * Ratings like PG, TV-PG have slightly higher medians; TV-MA and R show broader variation, indicating flexibility in mature content length.
2. Countplot – Ratings by Type: * Family/kids ratings (e.g., TV-Y, TV-G) are almost exclusive to TV Shows. * Movies dominate adult categories (R, PG-13), confirming strategic rating-based targeting.
3. Boxplot – Release Year by Type: * TV Shows have a more recent median release year, reflecting newer production trends, possibly due to Netflix Originals.
4. Heatmap + Pairplot: * No strong correlation between release year and duration, implying that newer content isn't necessarily longer or shorter.

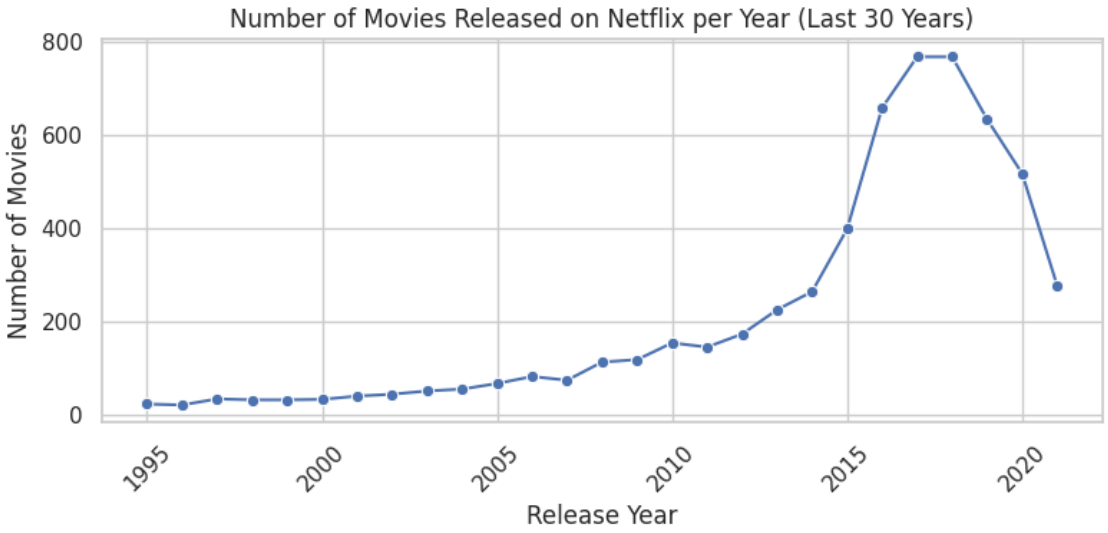
7 7.Movies Vs. TV Shows:

7.0.1 7.1 Change in release per year over the last 20-30 years:

7.1.1 For Movies:

[48]:
```
# Filter only movies
movie_df = netflix[netflix['type'] == 'Movie']
```

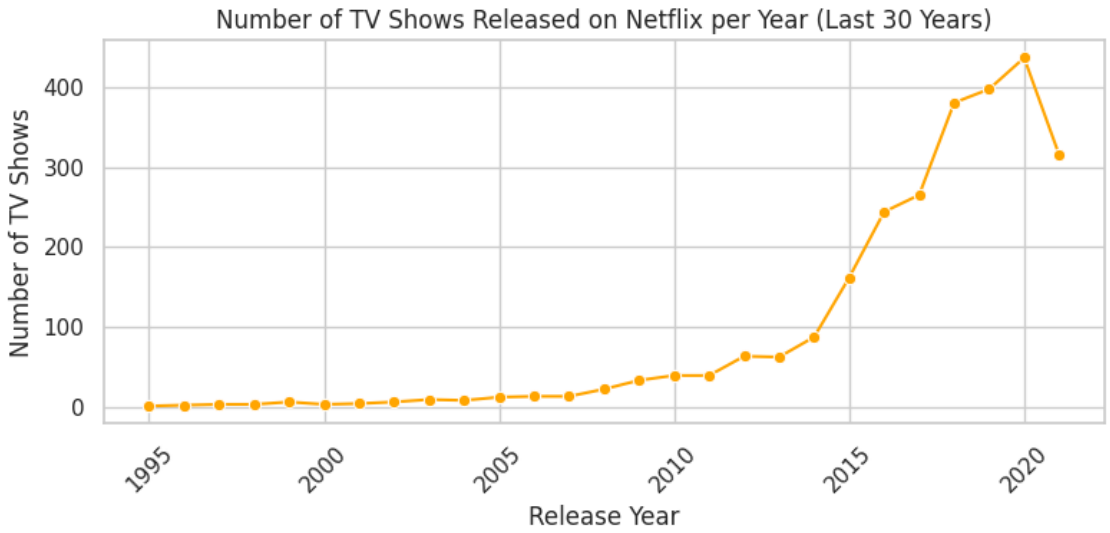
```
# Count  Movies released each year
movie_counts_by_year = movie_df['release_year'].value_counts().sort_index()

# Filter for the last 30 years
recent_movie_counts = movie_counts_by_year[movie_counts_by_year.index >= (2025␣
 ↪- 30)]

# Plot the data
plt.figure(figsize=(8, 4))
sns.lineplot(x=recent_movie_counts.index, y=recent_movie_counts.values,␣
 ↪marker='o')
plt.title("Number of Movies Released on Netflix per Year (Last 30 Years)")
plt.xlabel("Release Year")
plt.ylabel("Number of Movies")
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



7.1.2 For TV shows:

[49]:
```
# Filter only TV Shows
tv_df = netflix[netflix['type'] == 'TV Show']

# Count TV Shows released each year
tv_counts_by_year = tv_df['release_year'].value_counts().sort_index()

# Filter for the last 30 years
recent_tv_counts = tv_counts_by_year[tv_counts_by_year.index >= (2025 - 30)]
```

```
# Plot the data
plt.figure(figsize=(8, 4))
sns.lineplot(x=recent_tv_counts.index, y=recent_tv_counts.values, marker='o',␣
 ↪color='orange')
plt.title("Number of TV Shows Released on Netflix per Year (Last 30 Years)")
plt.xlabel("Release Year")
plt.ylabel("Number of TV Shows")
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



7.1.3 Content Release Analysis: **For Movie:** 1. The chart shows a notable increase in the number of movies released over the past 30 years, especially after 2010. Key observations:

2. From 1990s to early 2000s Netflix Relatively few movie releases.Whereas Post-2010 there is sharp growth, peaking around 2017–2019.
3. A decline around 2020–2021, likely due to global production delays during the COVID-19 pandemic.
4. This trend reflects Netflix's strategy shift toward increasing its catalog with newer titles and global content, particularly in the last decade.

For TV Shows: 1. The chart shows a sharp rise in TV show releases on Netflix over the last 10–15 years:

2. Before 2010 very few TV shows were released.
3. During 2015–2020 a major spike reflecting Netflix's strategic investment in original series and

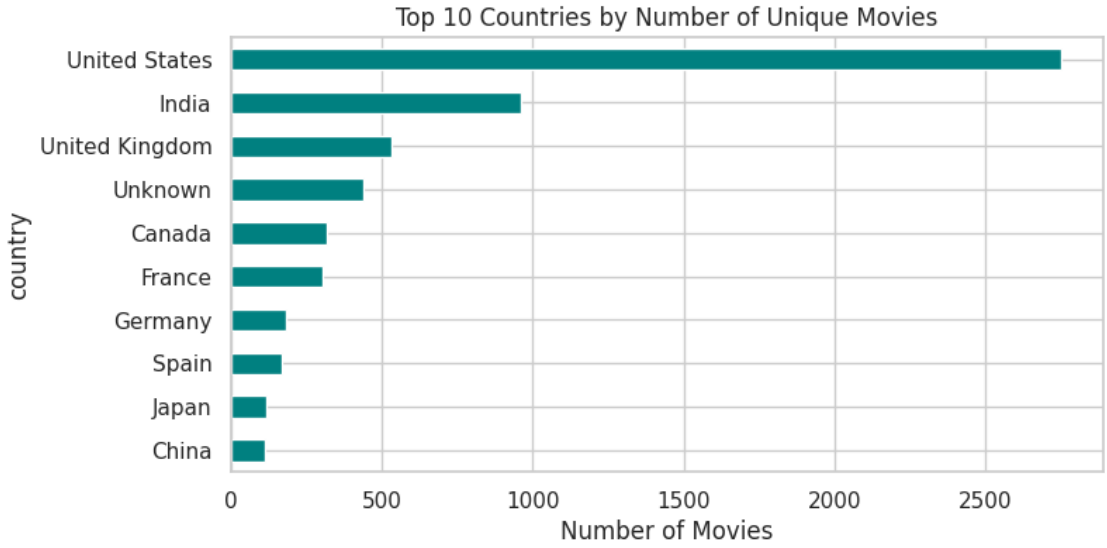
international TV content.

4. A slight drop is observed post-2020, likely influenced by pandemic-related production delays.
5. This pattern confirms Netflix's strong shift toward serialized content in recent years.

7.0.2 7.2 Content Against Countries:

7.2.1 For Movies:

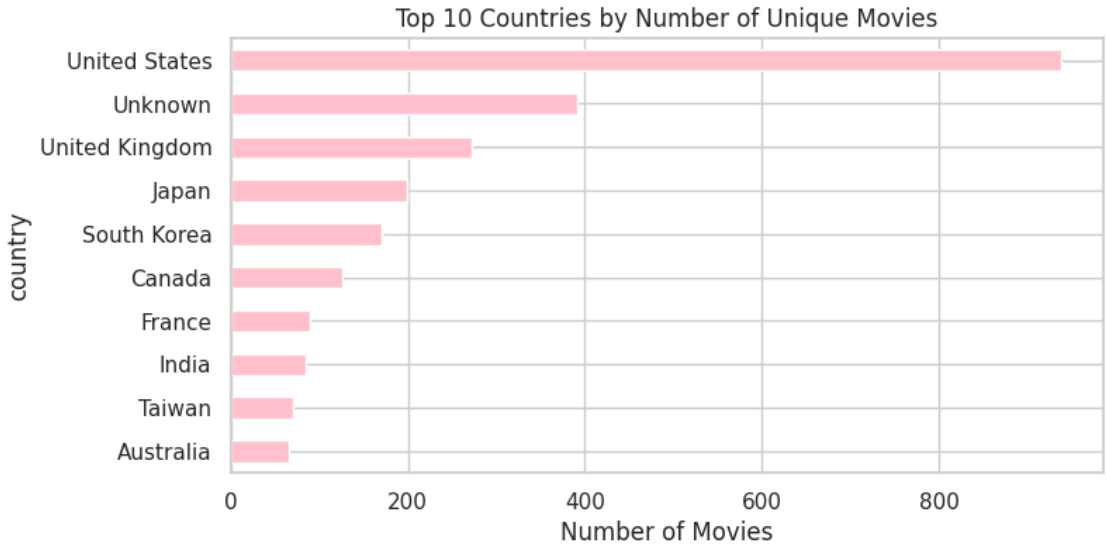
[50]:
```
# Filter only movies
movies_df = netflix[netflix['type'] == 'Movie']

# Handle multiple countries per row by splitting
movie_countries = movies_df[['title', 'country']].dropna()
movie_countries = movie_countries.assign(country=movie_countries['country'].str.
 ↪split(', ')).explode('country')

# Group by country and count unique movie titles
movies_by_country = movie_countries.groupby('country')['title'].nunique().
 ↪sort_values(ascending=False).head(10)

# Plot top 10
movies_by_country.plot(kind='barh', figsize=(8, 4), color='teal', title='Top 10␣
 ↪Countries by Number of Unique Movies')
plt.xlabel("Number of Movies")
plt.gca().invert_yaxis()
plt.show()

movies_by_country
```



[50]:
```
country
United States     2751
India              962
United Kingdom     532
Unknown            440
Canada             319
France             303
Germany            182
Spain              171
Japan              119
China              114
Name: title, dtype: int64
```

7.2.2 For TV Shows:

[51]:
```
# Filter only TV Shows
tv_df = netflix[netflix['type'] == 'TV Show']

# Handle multiple countries per row by splitting
tv_countries = tv_df[['title', 'country']].dropna()
tv_countries = tv_countries.assign(country=tv_countries['country'].str.split(', ')).explode('country')

# Group by country and count unique movie titles
tv_by_country = tv_countries.groupby('country')['title'].nunique().sort_values(ascending=False).head(10)

# Plot top 10
tv_by_country.plot(kind='barh', figsize=(8, 4), color='pink', title='Top 10 Countries by Number of Unique Movies')
plt.xlabel("Number of Movies")
plt.gca().invert_yaxis()
plt.show()

tv_by_country
```



[51]:
```
country
United States     938
Unknown           391
United Kingdom    272
Japan             199
South Korea       170
Canada            126
France             90
India              84
Taiwan             70
Australia          66
Name: title, dtype: int64
```

7.2.3 Insights Based on Production in Each Country:

1. Movies:

- U.S. dominates movie production, showcasing Netflix's heavy reliance on Hollywood content.
- India and the UK are also major contributors, reflecting the popularity of Bollywood and British cinema globally.
- Netflix has broad geographic diversity, with representation from Europe (France, Germany, Spain) and Asia (Japan, South Korea) — pointing to strong international content partnerships.
- Countries like Mexico and South Korea also stand out, showing rising interest in regional content (e.g., Korean dramas, Latin films).

2. TV Shows:

- TV content is more internationally distributed than movies — showing Netflix's strong investment in regional and localized shows.

- South Korea and Japan are among the top producers — reinforcing Netflix's strategic push into Asian TV content (especially with the global success of K-Dramas).
- The UK and Canada are key English-speaking markets for shows, serving both domestic and global audiences.

3. Recommendations:

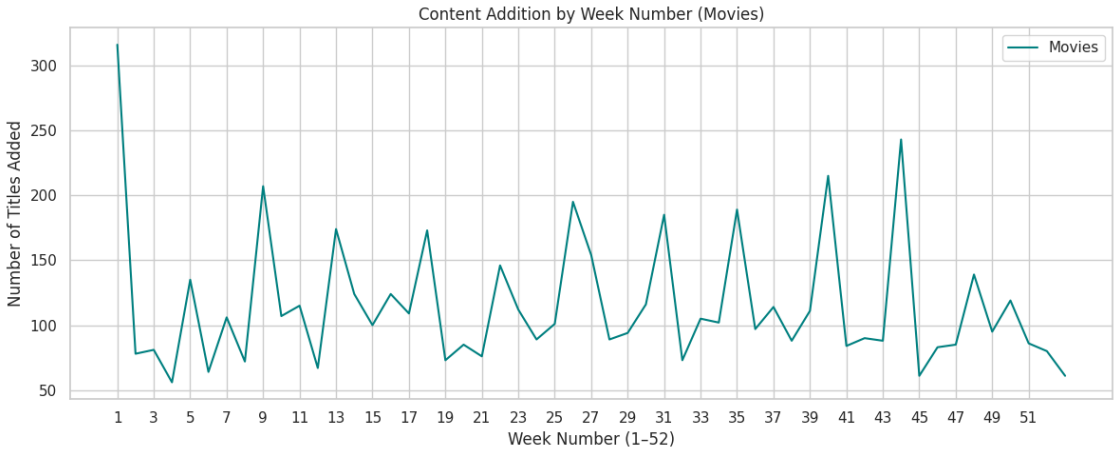
- Double down on regional TV content: With high global demand for Korean, Japanese, and Spanish shows, Netflix should continue investing in regional production houses.
- Expand movie production in emerging markets: Countries like Mexico, Spain, and Germany show potential — localized movies can strengthen Netflix's market position.
- Boost partnerships in underrepresented countries: Africa, Southeast Asia, and Eastern Europe have lower content volume — strategic investments there can boost market penetration.

8 8. Best Time To Launch:

8.0.1 8.1 Best Time to Launch a Movie:

[52]:
```
netflix['week_added'] = netflix['date_added'].dt.isocalendar().week

# Separate data for movies
movies_df = netflix[netflix['type'] == 'Movie']

# Group by week number and count entries
weekly_movies = movies_df.groupby('week_added')['show_id'].count()

# Combine into one DataFrame for plotting
weekly_combined = pd.DataFrame({
    'Movies': weekly_movies
}).fillna(0)

# Plotting
plt.figure(figsize=(8, 4))
weekly_combined.plot(figsize=(12, 5), title='Content Addition by Week Number␣
 ↪(Movies)', color=['teal'])
plt.xlabel("Week Number (1-52)")
plt.ylabel("Number of Titles Added")
plt.xticks(range(1, 53, 2))
plt.grid(True)
plt.tight_layout()
plt.show()

# Identify peak weeks
peak_movie_week = weekly_movies.idxmax()

peak_movie_week, weekly_movies.max()
```

```
<Figure size 800x400 with 0 Axes>
```

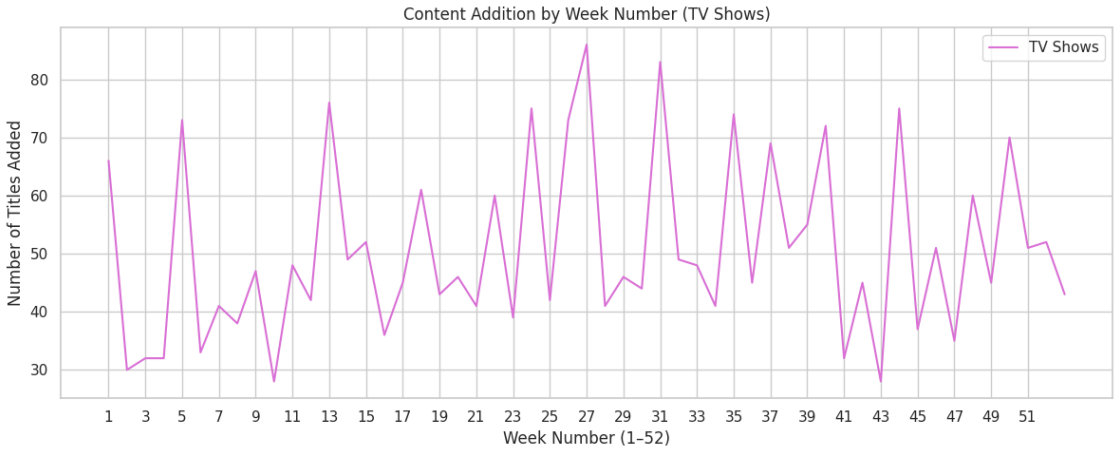


[52]:
```
(np.uint32(1), 316)
```

8.0.2 8.2 Best Time to Launch a TV Show:

[53]:
```
# Separate data for TV shows
tv_shows_df = netflix[netflix['type'] == 'TV Show']

# Group by week number and count entries
weekly_tv = tv_shows_df.groupby('week_added')['show_id'].count()

# Combine into one DataFrame for plotting
weekly_combined = pd.DataFrame({
    'TV Shows': weekly_tv
}).fillna(0)

# Plotting
plt.figure(figsize=(8, 4))
weekly_combined.plot(figsize=(12, 5), title='Content Addition by Week Number␣
 ↪(TV Shows)', color=['orchid'])
plt.xlabel("Week Number (1-52)")
plt.ylabel("Number of Titles Added")
plt.xticks(range(1, 53, 2))
plt.grid(True)
plt.tight_layout()
plt.show()

# Identify peak weeks
peak_tv_week = weekly_tv.idxmax()

peak_tv_week, weekly_tv.max()
```

```
<Figure size 800x400 with 0 Axes>
```



[53]:
```
(np.uint32(27), 86)
```

8.0.3 8.3 Insights For Best Time To Launch Content:

1. TV Shows:

- Peak weeks for adding new TV shows usually fall between weeks 36–42, which corresponds to September to mid-October.
- This aligns with traditional fall TV season launches in many countries and capitalizes on back-to-school routines and cooler weather when users are more likely to stay in and stream.
- Strategic Insight: Launching new TV shows in late Q3 or early Q4 can increase viewership due to seasonal engagement spikes.

2. Movies:

- Netflix tends to release more movies steadily year-round, but notable spikes occur in:

  a) Weeks 25–27 (June/early July), capitalizing on summer breaks.

  b)Weeks 50–52 (mid–late December), Holiday season & New Year, when families and individuals stream more.

- Strategic Insight: Releasing major movie titles in early summer and holiday season can maximize exposure and engagement.

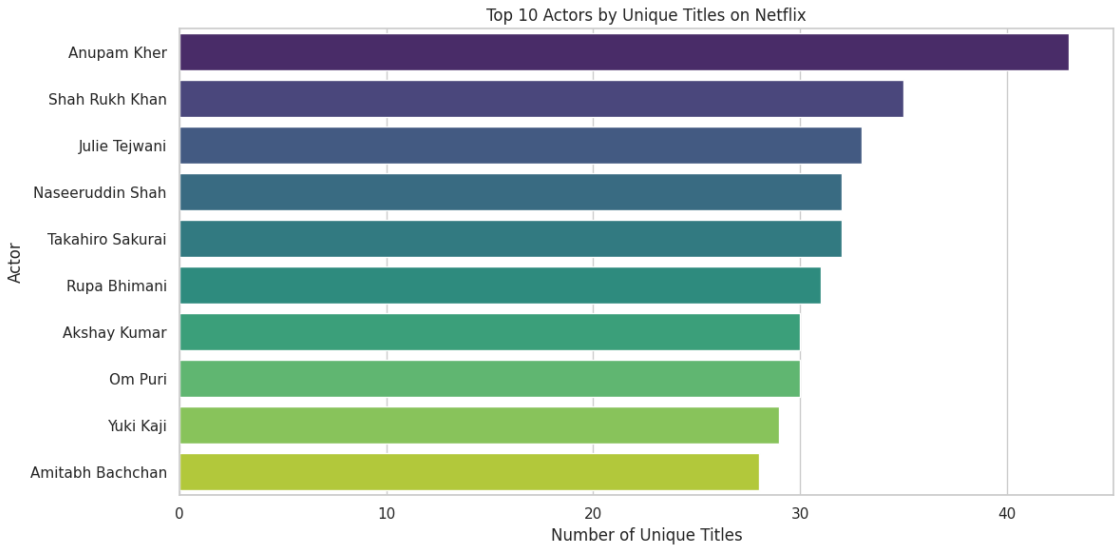
9 9. Analysis of Actors/Directors:

9.0.1 9.1 Top 10 Actors:

[54]:
```
# Split the cast column into individual actors
actor_exploded = netflix.assign(actor=netflix["cast"].str.split(", ")).
↪explode("actor")
```

```
# Removing default values as there is large number of missing values which are␣
 ↪replaced by 'Unknown'
actor_exploded = actor_exploded[actor_exploded["cast"] != 'Unknown']

# Count unique titles per actor
top_actors = (actor_exploded.groupby("actor")["title"].nunique().
 ↪sort_values(ascending=False).head(10)
    .reset_index(name="unique_title_count"))

# Plotting top 10 actors
plt.figure(figsize=(12, 6))
sns.barplot(x="unique_title_count", y="actor", data=top_actors,␣
 ↪palette="viridis")
plt.title("Top 10 Actors by Unique Titles on Netflix")
plt.xlabel("Number of Unique Titles")
plt.ylabel("Actor")
plt.tight_layout()
plt.show()

top_actors
```

```
<ipython-input-54-e4980a0e8a11>:13: 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="unique_title_count", y="actor", data=top_actors,
palette="viridis")
```



[54]:

	actor	unique_title_count
0	Anupam Kher	43
1	Shah Rukh Khan	35
2	Julie Tejwani	33
3	Naseeruddin Shah	32
4	Takahiro Sakurai	32
5	Rupa Bhimani	31
6	Akshay Kumar	30
7	Om Puri	30
8	Yuki Kaji	29
9	Amitabh Bachchan	28

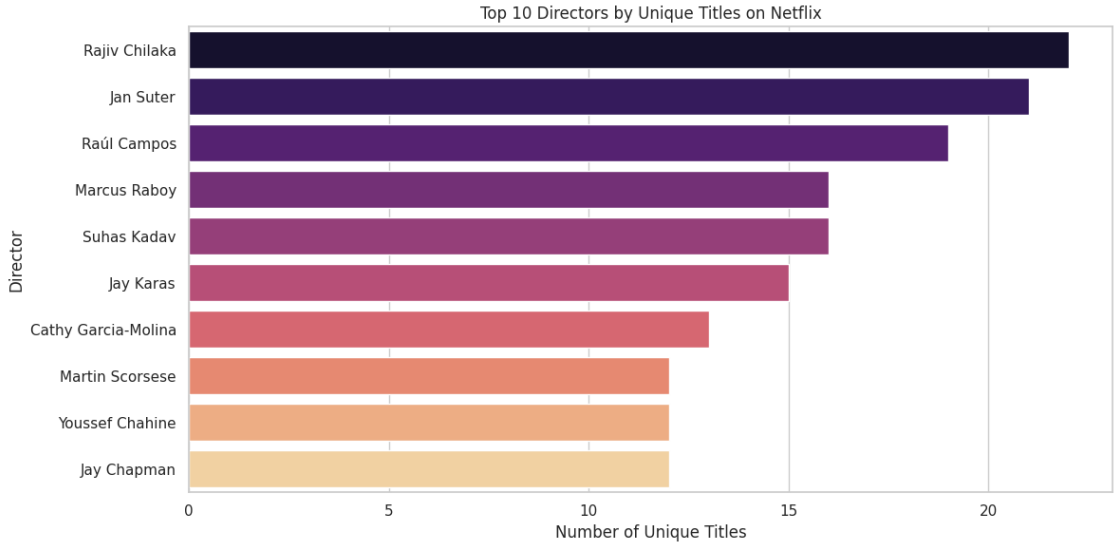
9.0.2 9.2 Top 10 Directors:

[55]:
```
# Split the director column if multiple directors exist
director_exploded = netflix.assign(director=netflix["director"].str.split(",␣
 ↪")).explode("director")

# Removing default values as there is large number of missing values which are␣
 ↪replaced by 'Unknown'
director_exploded = director_exploded[director_exploded["director"] !=␣
 ↪'Unknown']

# Count unique titles per director
top_directors = (director_exploded.groupby("director")["title"].nunique().
 ↪sort_values(ascending=False).head(10)
    .reset_index(name="unique_title_count"))

# Plotting top 10 directors
plt.figure(figsize=(12, 6))
sns.barplot(x="unique_title_count", y="director", data=top_directors,␣
 ↪palette="magma")
plt.title("Top 10 Directors by Unique Titles on Netflix")
plt.xlabel("Number of Unique Titles")
plt.ylabel("Director")
plt.tight_layout()
plt.show()

top_directors
```

```
<ipython-input-55-5116995a31d3>:13: 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="unique_title_count", y="director", data=top_directors,
palette="magma")
```



[55]:

	director	unique_title_count
0	Rajiv Chilaka	22
1	Jan Suter	21
2	Raúl Campos	19
3	Marcus Raboy	16
4	Suhas Kadav	16
5	Jay Karas	15
6	Cathy Garcia-Molina	13
7	Martin Scorsese	12
8	Youssef Chahine	12
9	Jay Chapman	12

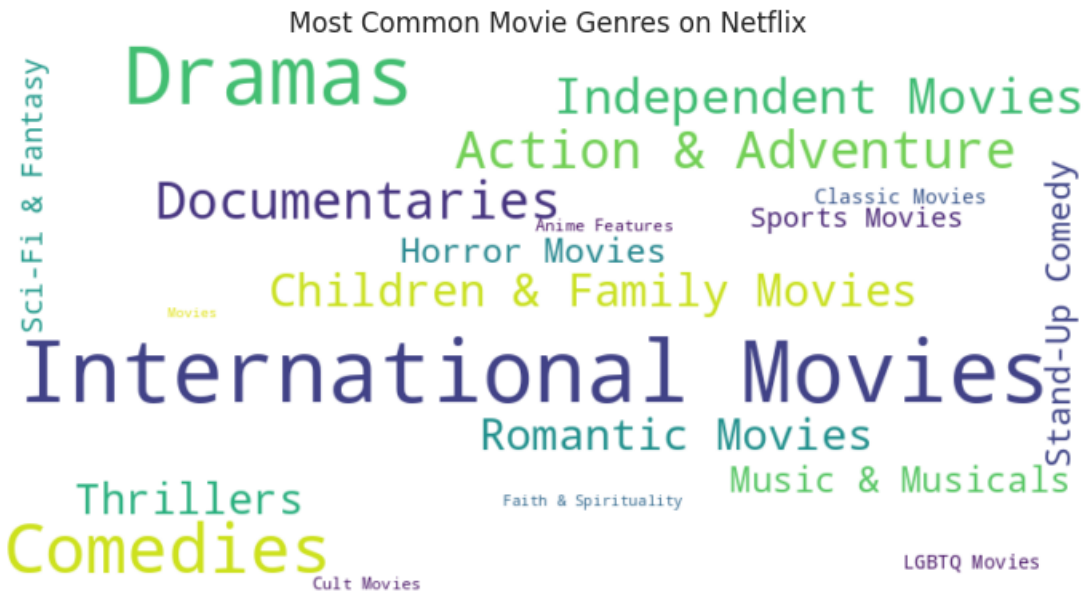
9.0.3 9.3 Insights:

- Top Actor by Unique Titles on Netflix is Anupam Kher leading with 43 titles.
- Directors by Unique Titles on Netflix is topped by Rajiv Chilaka with 22 titles

10 10 Most Popular Gener:

10.0.1 10.1 Most Popular Gener for Movie:

[56]:
```
from collections import Counter
from wordcloud import WordCloud

# Filter only movies
movies = netflix[netflix['type'] == 'Movie']

# Drop missing genres and split genre strings into individual genres
movie_genres_series = movies['listed_in'].dropna().str.split(', ')
movie_genres_flat = [genre for sublist in movie_genres_series for genre in␣
 ↪sublist]

# Count frequency of each genre
genre_counts = Counter(movie_genres_flat)

# Generate and display the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white',␣
 ↪colormap='viridis')
wordcloud.generate_from_frequencies(genre_counts)

plt.figure(figsize=(8, 4))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Most Common Movie Genres on Netflix")
plt.tight_layout()
plt.show()

# Print top 10 genres
print("Top 10 Movie Genres:")
for genre, count in genre_counts.most_common(10):
    print(f"{genre}: {count}")
```




```
Top 10 Movie Genres:
International Movies: 2752
Dramas: 2427
Comedies: 1674
Documentaries: 869
Action & Adventure: 859
Independent Movies: 756
Children & Family Movies: 641
Romantic Movies: 616
Thrillers: 577
Music & Musicals: 375
```

10.0.2 10.2 Most Popular Gener for TV Shows:

[57]:
```
# Filter only TV Shows
tv_shows = netflix[netflix['type'] == 'TV Show']

# Drop missing genres and split genre strings into individual genres
tv_genres_series = tv_shows['listed_in'].dropna().str.split(', ')
tv_genres_flat = [genre for sublist in tv_genres_series for genre in sublist]

# Count frequency of each genre
genre_counts = Counter(tv_genres_flat)

# Generate and display the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white',␣
 ↪colormap='plasma')
```

```
wordcloud.generate_from_frequencies(genre_counts)

plt.figure(figsize=(8, 4))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Most Common TV Show Genres on Netflix")
plt.tight_layout()
plt.show()

# Print top 10 genres
print("Top 10 TV Show Genres:")
for genre, count in genre_counts.most_common(10):
    print(f"{genre}: {count}")
```


```
Top 10 TV Show Genres:
International TV Shows: 1351
TV Dramas: 763
TV Comedies: 581
Crime TV Shows: 470
Kids' TV: 451
Docuseries: 395
Romantic TV Shows: 370
Reality TV: 255
British TV Shows: 253
Anime Series: 176
```

10.0.3 10.3 Wordcloud Insights:

1. Movie Genres:

- Dramas appear most frequently, reflecting Netflix's investment in emotion-driven narratives that appeal globally.
- Comedies and Action & Adventure also rank high, showing audience demand for lighter entertainment and high-energy content.
- Categories like International Movies and Foreign Movies are very common, indicating Netflix's strategy to cater to global audiences by producing and licensing regional films.
- Documentaries are prevalent, highlighting the platform's role in educational and socially relevant storytelling.
- Thrillers and Romantic Movies suggest a strong viewer base interested in suspense and emotional arcs.

2. TV Show Genres:

- International TV Shows are one of the most dominant categories—especially K-Dramas, Indian dramas, and British series—showing Netflix's localization strategy.
- TV Dramas are very prominent, paralleling the trend in movies but with serialized, long-form storytelling.
- TV Comedies and Reality TV have a strong presence, especially with the rise in binge-watching and casual content.
- Docuseries stand out as Netflix creates or licenses popular real-life crime and investigative series.
- Children & Family TV is a frequently seen category, which reflects Netflix's commitment to building a content library for all age groups.

11 11. Difference between Release Date and Added to Netflix:

[58]:
```
netflix['date_added'] = pd.to_datetime(netflix['date_added'], errors='coerce')

# Create a new column 'release_date' as January 1st of the release year
netflix['release_date'] = pd.to_datetime(netflix['release_year'].astype(str) +␣
 ↪'-01-01', errors='coerce')

# Calculate the difference in days between date_added and release_date
netflix['days_to_add'] = (netflix['date_added'] - netflix['release_date']).dt.
 ↪days

# Drop rows with NaN values in the new column
valid_differences = netflix['days_to_add'].dropna()

# Get the mode (most common value)
mode_days_to_add = valid_differences.mode()

mode_days_to_add
```

[58]:
```
0    334
Name: days_to_add, dtype: int64
```

[59]:
```
valid_differences.describe(include='all')
```

[59]:
```
count     8807.000000
mean      1876.087658
std       3260.279871
min     -17532.000000
25%         267.000000
50%         578.000000
75%        2065.000000
max       34331.000000
Name: days_to_add, dtype: float64
```

11.0.1 Insights:

- Most movies are added to Netflix roughly 11 months (334 days) after their release, though the median and mean suggest a much wider variation.
- This suggests that, based on recent data, movies are typically added to Netflix about 11 months after their initial release.

12 12. Business Insights:

1. **Content Volume Skewed Toward Movies:** Movies make up a larger portion of Netflix's catalog compared to TV Shows, suggesting a historical emphasis on feature films.
2. **Recent Surge in Content Production:** Most content was added after 2015, especially TV Shows, indicating a strong push toward newer content in recent years.
3. **Focus on Mature and Teen Audiences:** The most common ratings are TV-MA and TV-14, showing that Netflix targets mature viewers and young adults more than families or kids.
4. **Standardized Movie Length:** Most movies range from 80 to 120 minutes, aligning with industry norms and suggesting Netflix sticks to viewer comfort zones.
5. **TV Shows Are More Recent:** TV Shows tend to have later release years than movies, showing Netflix's recent strategic push into episodic and original series.
6. **Genres and Countries Are Diverse but Uneven:** Certain genres like "International TV Shows" and countries like the U.S. dominate the catalog, while many other regions are underrepresented.
7. **Limited Correlation Between Duration and Release Year:** There's no consistent relationship between how new a movie/show is and how long it is — both short and long content exist in all years.
8. **Family-Friendly Content Is Niche:** Content with ratings like TV-Y and TV-G is limited, suggesting fewer offerings for children and family audiences.

9. **Directors and Actors are Repeated in Clusters:** Certain directors and actors appear frequently, pointing to preferred collaborations or popular personalities.
10. **Multiple Genres per Title:** Each show or movie is listed under several genres, indicating a strategy to attract wider audiences through cross-categorization.

13 13. Recommendations:

1. **Invest More in TV Shows:** TV shows are gaining momentum — expand Netflix Originals in this area to keep users engaged over time.
2. **Balance the Ratings Portfolio:** Add more family-friendly and kids' content to broaden the user base and cater to shared viewing households.
3. **Target Emerging Markets:** Increase content from growing markets like India, South Korea, and Africa, where Netflix viewership is accelerating.
4. **Diversify Genre Offering:** Study underrepresented genres and produce content in those areas to attract niche audiences and reduce redundancy.
5. **Produce Shorter Content for Mobile Users:** Offer shorter movies and mini-series (under 60 minutes) for mobile and on-the-go viewers.
6. **Leverage Popular Talent:** Identify frequently appearing directors and actors — use them strategically in new projects for guaranteed viewer interest.
7. **Update Older Catalog Items:** Repackage or highlight older, quality content to add variety without new production costs.
8. **Analyze and Promote Cross-Genre Hits:** Use genre tags to find what combinations (e.g., comedy + drama) perform best, and produce more of those hybrids.
9. **Tailor Regional Content with Local Stars:** In each country, use local languages and familiar actors to increase content relatability and adoption.
10. **Highlight New Releases More Aggressively:** Since viewers lean toward newer content, improve the discoverability of fresh titles on the homepage and recommendations.