In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
netflix_dataset = pd.read_csv("C:/Users/SINDHU RATHOD/Downloads/netflix_dataset.csv")
netflix_dataset.head()
```

Out[2]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	de
0	s1	TV Show	3%	NaN	João Miguel, Bianca Comparato, Michel Gomes, R...	Brazil	August 14, 2020	2020	TV-MA	4 Seasons	International TV Shows, TV Dramas, TV Sci-Fi &...	wher
1	s2	Movie	07:19	Jorge Michel Grau	Demián Bichir, Héctor Bonilla, Oscar Serrano, ...	Mexico	December 23, 2016	2016	TV-MA	93 min	Dramas, International Movies	de earth M
2	s3	Movie	23:59	Gilbert Chan	Tedd Chan, Stella Chung, Henley Hii, Lawrence ...	Singapore	December 20, 2018	2011	R	78 min	Horror Movies, International Movies	Whe recru
3	s4	Movie	9	Shane Acker	Elijah Wood, John C. Reilly, Jennifer Connelly...	United States	November 16, 2017	2009	PG-13	80 min	Action & Adventure, Independent Movies, Sci-Fi...	posta worl r
4	s5	Movie	21	Robert Luketic	Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar...	United States	January 1, 2020	2008	PG-13	123 min	Dramas	Abrill o bec

In [3]:

```
netflix_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7787 entries, 0 to 7786
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   show_id       7787 non-null   object
 1   type          7787 non-null   object
 2   title         7787 non-null   object
 3   director      5398 non-null   object
 4   cast          7069 non-null   object
 5   country       7281 non-null   object
 6   date_added    7777 non-null   object
 7   release_year  7787 non-null   int64 
 8   rating        7780 non-null   object
 9   duration      7787 non-null   object
 10  listed_in     7787 non-null   object
 11  description   7787 non-null   object
dtypes: int64(1), object(11)
memory usage: 730.2+ KB
```

In [4]:
```
# Identifying unique values for each column
unique_counts = netflix_dataset.nunique()
unique_counts_df = pd.DataFrame(unique_counts, columns=["Unique counts"])  # Just conv. to
print(unique_counts_df)
```

```
              Unique counts
show_id                7787
type                      2
title                  7787
director               4049
cast                   6831
country                 681
date_added             1565
release_year             73
rating                   14
duration                216
listed_in               492
description            7769
```

In [5]:
```
# Identify the missing values
null = netflix_dataset.isnull().sum()
# print(null)
null_df = null.reset_index()
null_df.columns = ['Columns', 'Null_value_count']
print(null_df)
```

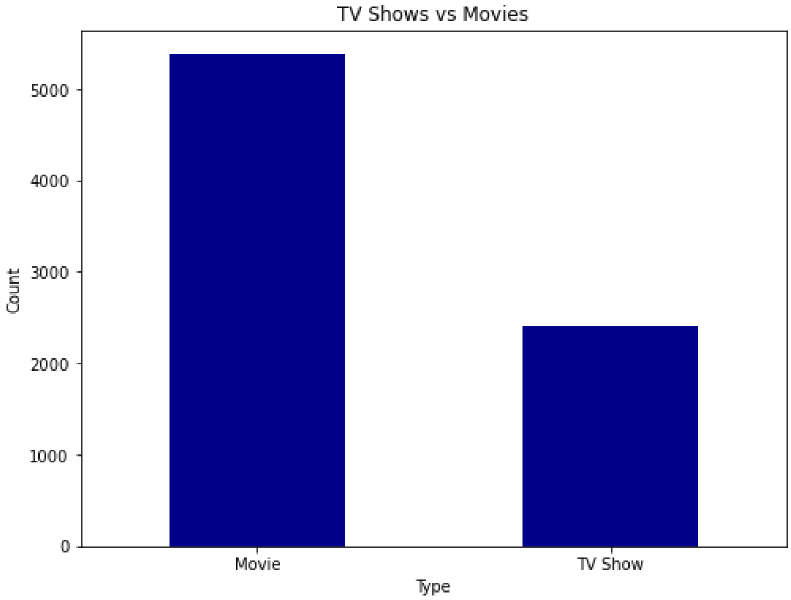
```
         Columns  Null_value_count
0        show_id                 0
1           type                 0
2          title                 0
3       director              2389
4           cast               718
5        country               506
6     date_added                10
7   release_year                 0
8         rating                 7
9       duration                 0
10     listed_in                 0
11   description                 0
```

Analysis of TV Shows and Movies

In [6]:
```
netflix_shows=netflix_dataset[netflix_dataset['type']=='TV Show']
netflix_movies=netflix_dataset[netflix_dataset['type']=='Movie']
```

In [7]:
```
# Count of Movies and TV shows
counts = netflix_dataset['type'].value_counts()
print(counts)
plt.figure(figsize=(8, 6))
counts.plot(kind='bar', color='darkblue')
plt.title('TV Shows vs Movies')
plt.xlabel('Type')
plt.ylabel('Count')
plt.xticks(rotation=0)  # Optional: Keeps x-axis labels horizontal
plt.show()
```

```
Movie      5377
TV Show    2410
Name: type, dtype: int64
```



TV Shows Analysis

In [8]:
```
# Define the desired order of months
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']

netflix_date = netflix_shows[['date_added']].dropna()  # Return non-null values
netflix_date['year'] = netflix_date['date_added'].str.split(',').str[-1].str.strip()
netflix_date['month'] = netflix_date['date_added'].str.split(' ').str[0]
netflix_date['month'] = pd.Categorical(netflix_date['month'], categories=month_order, order
netflix_date
```

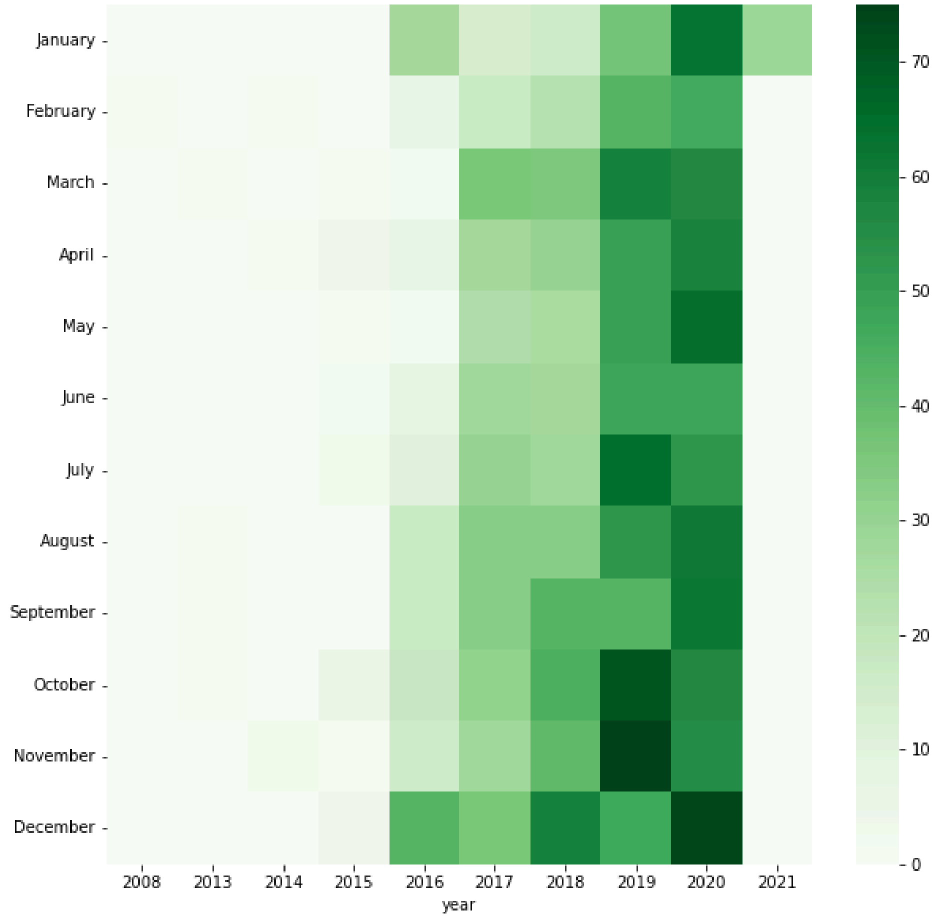
Out[8]:

	date_added	year	month
0	August 14, 2020	2020	August
5	July 1, 2017	2017	July
11	November 30, 2018	2018	November
12	May 17, 2019	2019	May
16	March 20, 2019	2019	March
...	...	...	...
7767	December 15, 2016	2016	December
7775	August 14, 2020	2020	August
7777	July 1, 2019	2019	July
7779	November 26, 2019	2019	November
7785	October 31, 2020	2020	October

2400 rows × 3 columns

In [9]:
```
# Heatmap: Distribution of movies(by date added) by each month in each year
df = netflix_date.groupby('year')['month'].value_counts().unstack().fillna(0)[month_order].
plt.subplots(figsize=(10,10))
sns.heatmap(df,cmap='Greens') #heatmap
```

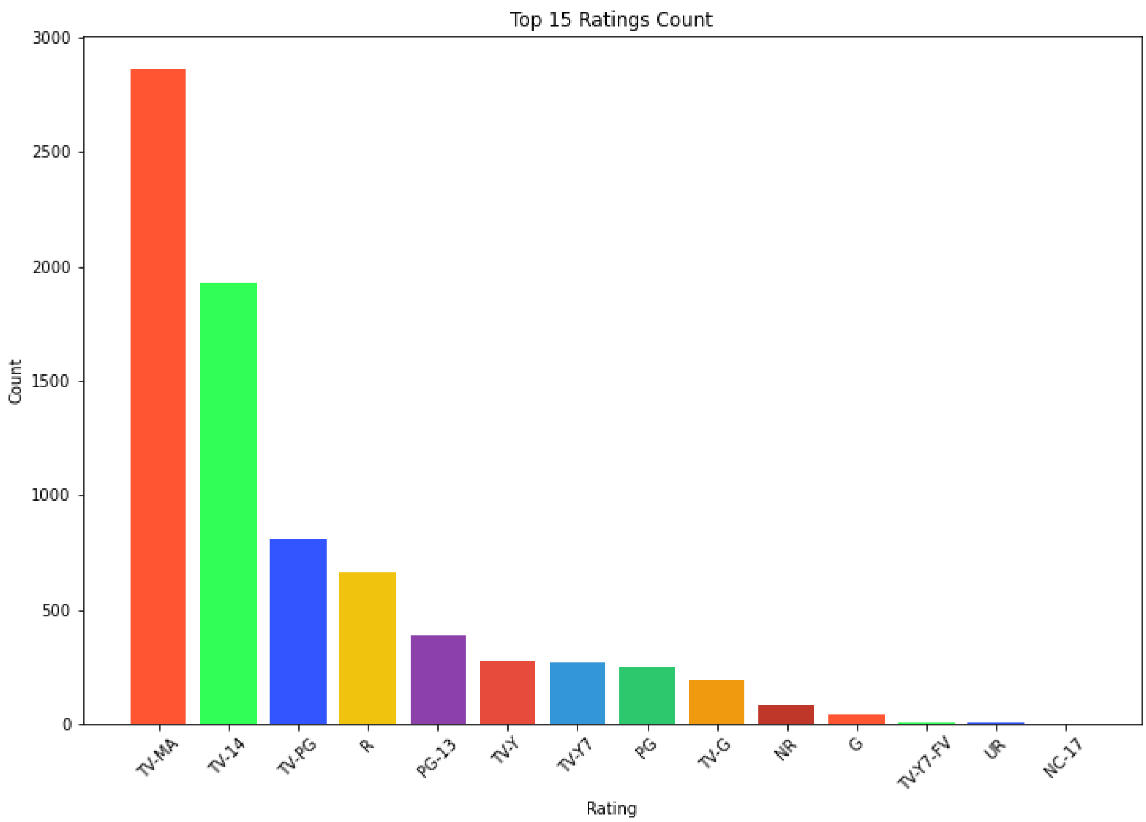
Out[9]: <AxesSubplot:xlabel='year'>



Movie Ratings Analysis

In [10]:
```
# Count occurrences of each rating
from matplotlib import cm
rating_counts = netflix_dataset['rating'].value_counts()
# Define a list of colors for the bars
colors = ['#FF5733', '#33FF57', '#3357FF', '#F1C40F', '#8E44AD', '#E74C3C', '#3498DB', '#2E

plt.figure(figsize=(12, 8))
plt.bar(rating_counts.index, rating_counts.values, color=colors)
plt.xticks(rotation=45)
plt.xlabel('Rating')
plt.ylabel('Count')
plt.title('Top 15 Ratings Count')
plt.show()
```



In [11]:
```
#Analysing IMDB ratings to get top rated movies on Netflix
imdb_ratings=pd.read_csv("C:/Users/SINDHU RATHOD/Downloads/IMDb ratings.csv", usecols=['wei
imdb_titles=pd.read_csv("C:/Users/SINDHU RATHOD/Downloads/IMDb movies.csv", usecols=['title

ratings = pd.DataFrame({'Title':imdb_titles.title,
                    'Release Year':imdb_titles.year,
                    'Rating': imdb_ratings.weighted_average_vote,
                    'Genre':imdb_titles.genre})
ratings.drop_duplicates(subset=['Title','Release Year','Rating'], inplace=True)
```

In [12]:
```
#joining netflix dataset and imdb dataset
ratings.dropna()
joint_data=ratings.merge(netflix_dataset,left_on='Title',right_on='title',how='inner')
joint_data=joint_data.sort_values(by='Rating', ascending=False)
joint_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2741 entries, 991 to 1205
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Title         2741 non-null   object 
 1   Release Year  2741 non-null   object 
 2   Rating        2741 non-null   float64
 3   Genre         2741 non-null   object 
 4   show_id       2741 non-null   object 
 5   type          2741 non-null   object 
 6   title         2741 non-null   object 
 7   director      2413 non-null   object 
 8   cast          2680 non-null   object 
 9   country       2697 non-null   object 
 10  date_added    2739 non-null   object 
 11  release_year  2741 non-null   int64  
 12  rating        2740 non-null   object 
 13  duration      2741 non-null   object 
 14  listed_in     2741 non-null   object 
 15  description   2741 non-null   object 
dtypes: float64(1), int64(1), object(14)
memory usage: 364.0+ KB
```

Top 10 Leading Countries in Film and Television Production

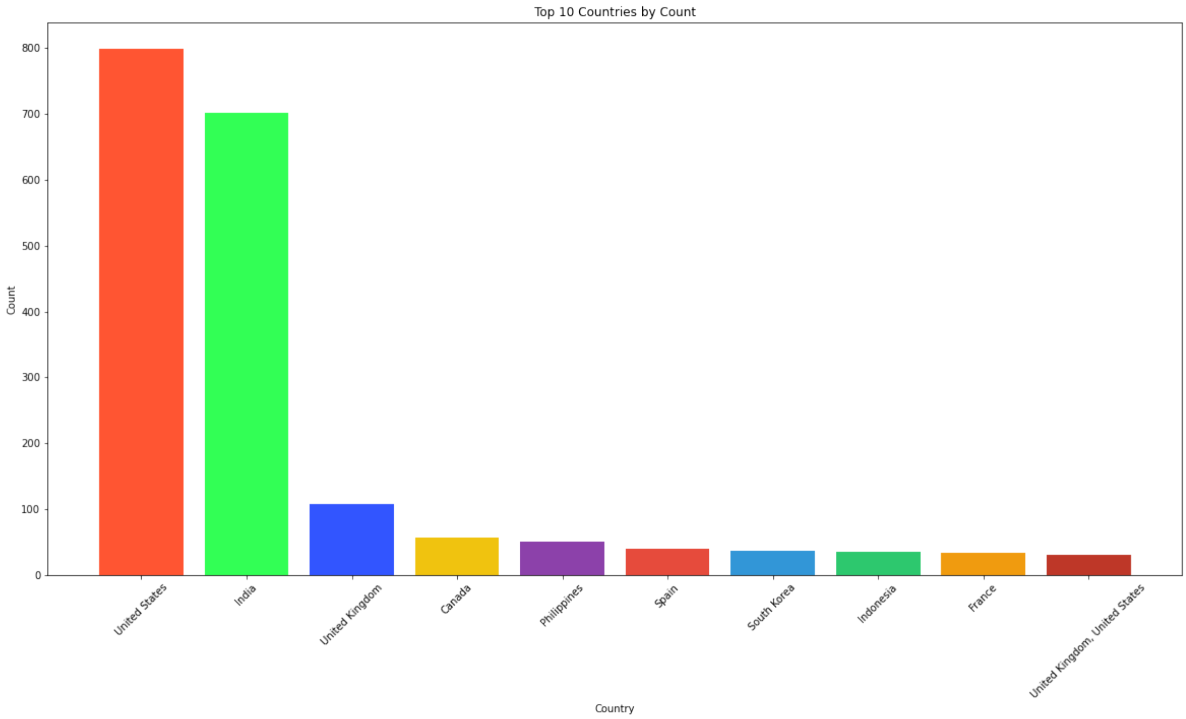
In [13]:

```
import matplotlib.pyplot as plt

country_count = joint_data['country'].value_counts().sort_values(ascending=False)
country_count = pd.DataFrame({'Country': country_count.index, 'Count': country_count.values
top_countries = country_count.head(10)

# Define a list of colors for the bars
colors = ['#FF5733', '#33FF57', '#3357FF', '#F1C40F', '#8E44AD', '#E74C3C', '#3498DB', '#2E

plt.figure(figsize=(20, 10))
plt.bar(top_countries['Country'], top_countries['Count'], color=colors)
plt.xlabel('Country')
plt.ylabel('Count')
plt.title('Top 10 Countries by Count')
plt.xticks(rotation=45)
plt.show()
```



In [14]:

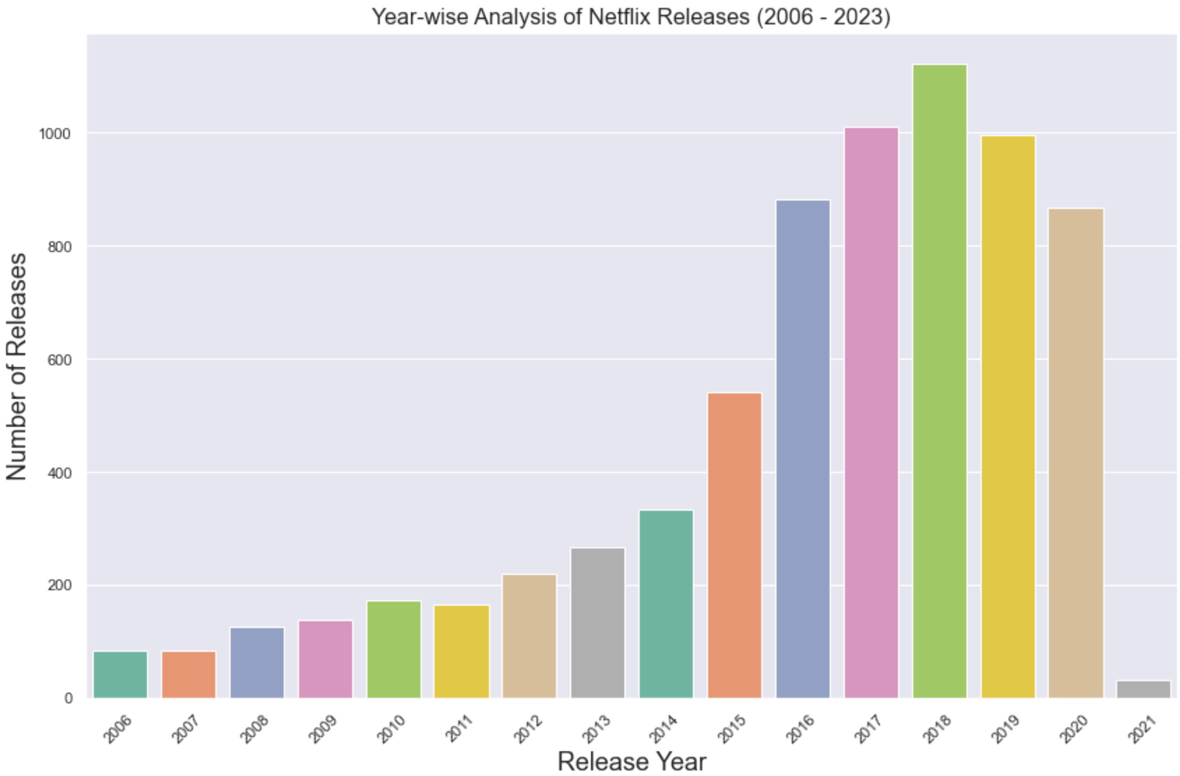
```
# Year-wise Analysis for the last 20 years
recent_years = netflix_dataset[netflix_dataset['release_year'] > 2005]

# Grouping data by release year and counting occurrences
year_counts = recent_years['release_year'].value_counts().sort_index()

# Plotting the year-wise analysis
plt.figure(figsize=(12, 8))
sns.set(style="darkgrid")

# Creating a bar plot
sns.barplot(x=year_counts.index, y=year_counts.values, palette="Set2")
# Adding titles and labels
plt.title('Year-wise Analysis of Netflix Releases (2006 - 2023)', fontsize=16)
plt.xlabel('Release Year', fontsize=18)
plt.ylabel('Number of Releases', fontsize=18)

# Displaying the plot
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

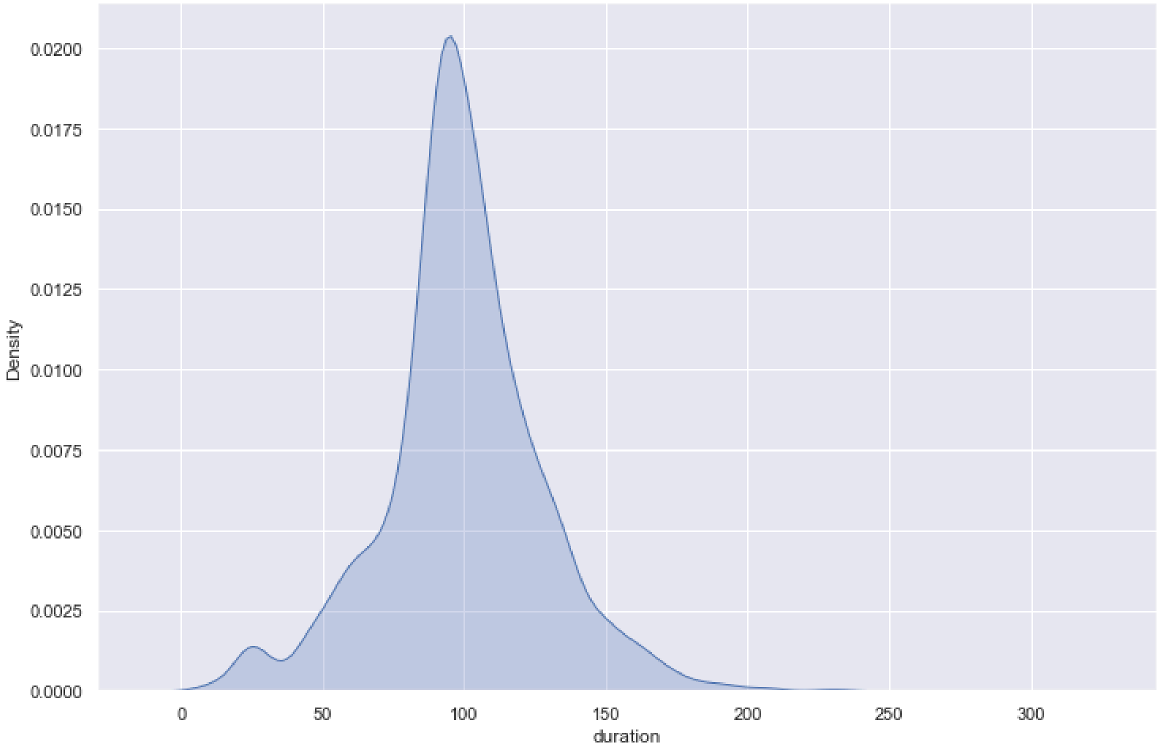


Analysis of Duration of Movies

In [15]:

```
netflix_movies['duration'] = netflix_movies['duration'].astype(str).str.replace(' min', '',
netflix_movies['duration'] = netflix_movies['duration'].astype(int)
```

In [16]:
```
sns.set(style="darkgrid")
plt.figure(figsize=(12,8))
ax=sns.kdeplot(data=netflix_movies['duration'], shade=True)
```



Analysis of Duration of Series

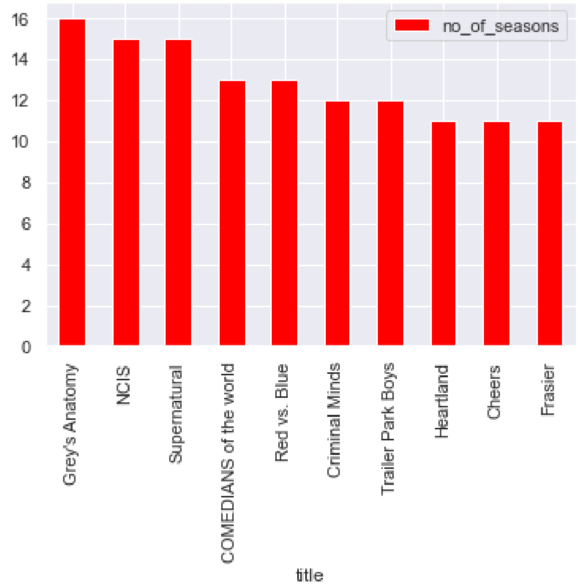
In [17]:
```
features=['title','duration']
durations= netflix_shows[features]
durations['no_of_seasons']=durations['duration'].str.replace(' Season','')
durations['no_of_seasons']=durations['no_of_seasons'].str.replace('s','')
durations['no_of_seasons']=durations['no_of_seasons'].astype(str).astype(int)
```

In [18]:
```
#TV shows with largest number of seasons
t=['title','no_of_seasons']
top=durations[t]

top=top.sort_values(by='no_of_seasons', ascending=False)
top10=top[0:10]
plt.figure(figsize=(10,8))
top10.plot(kind='bar',x='title',y='no_of_seasons', color='red')
```

Out[18]:
```
<AxesSubplot:xlabel='title'>

<Figure size 720x576 with 0 Axes>
```



Content Based Recommendations¶

We will compute pairwise similarity scores for all movies based on their plot descriptions and recommend movies based on that similarity score.The plot description is given in the description feature of our dataset

In [19]:
```
#Plot description based Recommender

netflix_dataset['description'].head()
```

Out[19]:
```
0    In a future where the elite inhabit an island ...
1    After a devastating earthquake hits Mexico Cit...
2    When an army recruit is found dead, his fellow...
3    In a postapocalyptic world, rag-doll robots hi...
4    A brilliant group of students become card-coun...
Name: description, dtype: object
```

The TF-IDF(Term Frequency-Inverse Document Frequency (TF-IDF)) score is the frequency of a word occurring in a document, down-weighted by the number of documents in which it occurs. This is done to reduce the importance of words that occur frequently in plot overviews and therefore, their significance in computing the final similarity score.

Now if you are wondering what is Term Frequency (TF), it is the relative frequency of a word in a document and is given as (term instances/total instances). Inverse Document Frequency (IDF) is the relative count of documents containing the term is given as log(number of documents/documents with term) The overall importance of each word to the documents in which they appear is equal to TF * IDF

This will give us a matrix where each column represents a word in the description vocabulary (all the words that appear in at least one document) and each row represents a movie, as before. This is done to reduce the importance of words that occur frequently in plot descriptions and therefore, their significance in computing the final similarity score.

In [20]:
```
#Recommedation System(Content Based)

from sklearn.feature_extraction.text import TfidfVectorizer
#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a', etc.
tfidf = TfidfVectorizer(stop_words='english')
netflix_dataset['description'] = netflix_dataset['description'].fillna('')
tfidf_matrix = tfidf.fit_transform(netflix_dataset['description'] )
tfidf_matrix.shape
```

Out[20]:
```
(7787, 17905)
```

This means there are 17,905 different words describing the 7787 movies in our dataset.

With this matrix in hand, we can now compute a similarity score. There are several methods for this; such as the euclidean, the Pearson and the cosine similarity scores. There is no right answer to which score is the best. Different scores work well in different scenarios.

cosine similarity used to measure the similarity between two movies based on their features, such as keywords or descriptions. Cosine similarity is preferred because it is independent of the magnitude of the vectors, focusing only on the orientation, which makes it effective for this type of analysis.

Formula: similarity = cos(x, y) = (x . y) / (||x|| * ||y||), defines how to compute the cosine similarity where:

x and y are the feature vectors of the two movies, x . y is the dot product of the two vectors, ||x|| and ||y|| are the magnitudes (norms) of the vectors. Using the TF-IDF vectorizer transforms the text into numeric form, which allows for this calculation to be made directly through the dot product of the vectors.

The use of sklearn's linear_kernel() is recommended over cosine_similarities() because it is computationally faster, making it more suitable for large datasets, such as those encountered in movie recommendations or collaborative filtering applications.

In [21]:
```
from sklearn.metrics.pairwise import linear_kernel
# Cal cosine_sim:
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim
```

Out[21]:
```
array([[1.        , 0.        , 0.05827946, ..., 0.        , 0.        ,
        0.        ],
       [0.        , 1.        , 0.        , ..., 0.09600035, 0.        ,
        0.        ],
       [0.05827946, 0.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.09600035, 0.        , ..., 1.        , 0.        ,
        0.02819239],
       [0.        , 0.        , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.02819239, 0.        ,
        1.        ]])
```

We are going to define a function that takes in a movie title as an input and outputs a list of the 10 most similar movies. Firstly, for this, we need a reverse mapping of movie titles and DataFrame indices. In other words, we need a mechanism to identify the index of a movie in our netflix DataFrame, given its title.

In [22]:
```
# Construct a reverse map if indices and movie titles

indices = pd.Series(netflix_dataset.index, index = netflix_dataset['title']).drop_duplicate
```

In [23]:
```python
# Function that takes in movie title as input and outputs most similar movies

def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return netflix_dataset['title'].iloc[movie_indices]
```

In [24]:
```python
# Test

get_recommendations('Peaky Blinders')
```

Out[24]:
```
4692                     Our Godfather
4358                    My Stupid Boss
1807                               Don
6344                          The Fear
3219    Jonathan Strange & Mr Norrell
4953                 Power Rangers Zeo
6783                        The Prison
6950                        The Tudors
6236                      The Con Is On
6585     The Legend of Michael Mishra
Name: title, dtype: object
```

In [25]:
```python
get_recommendations('3 Idiots')
```

Out[25]:
```
1463                          College Romance
2005                        Engineering Girls
1197                                Candy Jar
4261                                Mr. Young
55       100 Things to do Before High School
4739                                   Pahuna
851                            Best Neighbors
777                                Be with Me
4171                              Moms at War
3790                                 Lovesong
Name: title, dtype: object
```

While our system has done a decent job of finding movies with similar plot descriptions, the quality of recommendations is not that great. "3 idiots" returns movies with similar plots(College Life). But if someone wants the same director or actors, it fails.

Therefore, more metrics should be added to the model to improve performance.

Content-based Filtering

Content based filtering on the following factors:

Title, Cast, Director, Listed in, Plot.

In [26]:
```python
#Filling null values with empty string.
filledna=netflix_dataset.fillna('')
```

To ensure that our vectorizer treats variations in names consistently, we should convert all names and keyword instances to lowercase and remove any spaces. This way, "Tony Stark" and "Tony Anthony" will not be considered equivalent due to differences in casing and spacing. This normalization step enhances the accuracy of our analysis by

In [27]:
```python
def clean_data(x):
    return str.lower(x.replace(" ",""))
#Identifying features on which the model is to be filtered.
features=['title','director','cast','listed_in','description']
filledna=filledna[features]
```

In [28]:
```python
for feature in features:
    filledna[feature] = filledna[feature].apply(clean_data)
filledna.head()
```

Out[28]:

	title	director	cast	listed_in	
0	3%		joãomiguel,biancacomparato,michelgomes,rodolfo...	internationaltvshows,tvdramas,tvsci-fi&fantasy	inafutu
1	07:19	jorgemichelgrau	demiánbichir,héctorbonilla,oscarserrano,azalia...	dramas,internationalmovies	aftera
2	23:59	gilbertchan	teddchan,stellachung,henleyhii,lawrencekoh,tom...	horrormovies,internationalmovies	whenan
3	9	shaneacker	elijahwood,johnc.reilly,jenniferconnelly,chris...	action&adventure,independentmovies,sci-fi&fantasy	inapo
4	21	robertluketic	jimsturgess,kevinspacey,katebosworth,aaronyoo,...	dramas	abrillian

Create the Bag of Words: Once you have a string for each entry, you can compile them into a list(as below) or another appropriate structure for further processing.

Vectorization: Use a text vectorizer (such as CountVectorizer or TfidfVectorizer from libraries like scikit-learn) to transform this bag of words into numerical representations suitable for modeling.

In [29]:
```python
def create_soup(x):
    return x['title']+ ' ' + x['director'] + ' ' + x['cast'] + ' ' +x['listed_in']+' '+ x['

filledna['soup'] = filledna.apply(create_soup, axis=1)
filledna.tail()
```

Out[29]:

	title	director	cast	
7782	zozo	joseffares	imadcreidi,antoinetteturk,eliasgergi,carmenleb...	dramas,
7783	zubaan	mozezsingh	vickykaushal,sarah-janedias,raaghavchanana,man...	dramas,internationalmovi
7784	zulumaninjapan		nastyc	documentaries,internationalmovi
7785	zumbo'sjustdesserts		adrianozumbo,rachelkhoo	internatic
7786	zztop:thatlittleol'bandfromtexas	samdunn		documentari

The next steps involve creating a recommender system similar to the plot description-based one, but with an important modification: instead of using TF-IDF, we will use CountVectorizer.

The reason for this choice is that we want to treat the presence of actors and directors equally, regardless of how many films they have been involved in. Using CountVectorizer allows us to focus on the actual presence of these features without down-weighting them based on their frequency across multiple films, which aligns better with the goal of the recommendation system.

In [30]:
```
# Import CountVectorizer and create the count matrix
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(filledna['soup'])

from sklearn.metrics.pairwise import cosine_similarity
cosine_sim2 = cosine_similarity(count_matrix, count_matrix)

cosine_sim2
```

Out[30]:
```
array([[1.        , 0.        , 0.        , ..., 0.        , 0.06917145,
        0.        ],
       [0.        , 1.        , 0.06726728, ..., 0.0836242 , 0.        ,
        0.        ],
       [0.        , 0.06726728, 1.        , ..., 0.07312724, 0.        ,
        0.        ],
       ...,
       [0.        , 0.0836242 , 0.07312724, ..., 1.        , 0.        ,
        0.30151134],
       [0.06917145, 0.        , 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.30151134, 0.        ,
        1.        ]])
```

In [31]:
```
# Reset index of our main DataFrame and construct reverse mapping as before
filledna=filledna.reset_index()
indices = pd.Series(filledna.index, index=filledna['title'])
```

In [32]:
```
def get_recommendations_new(title, cosine_sim=cosine_sim):
    title=title.replace(' ','').lower()
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return netflix_dataset['title'].iloc[movie_indices]
```

In [33]:
```
get_recommendations_new('3 Idiots', cosine_sim2)
```

Out[33]:
```
4872                            PK
7477              War Chhod Na Yaar
6585    The Legend of Michael Mishra
5097                Rang De Basanti
3982                          Maska
5968                        Talaash
2571                         Haapus
2149             Ferrari Ki Sawaari
5377                          Sanju
5904                     Super Nani
Name: title, dtype: object
```

In [34]:
```
get_recommendations_new('Peaky Blinders', cosine_sim2)
```

Out[34]:
```
2419                                 Giri / Haji
6374                   The Frankenstein Chronicles
6693                         The Murder Detectives
3692                                        Loaded
3412                                 Kiss Me First
2616                                  Happy Valley
2381                                      Get Even
2846    How to Live Mortgage Free with Sarah Beeny
2886                                  I AM A KILLER
3013                       Inside the Criminal Mind
Name: title, dtype: object
```

In [35]:
```
get_recommendations_new('Andhadhun', cosine_sim2)
```

Out[35]:
```
751         Bareilly Ki Barfi
1104    Brij Mohan Amar Rahe
2571                  Haapus
1261          Chal Dhar Pakad
1032               Bombairiya
4757           Papa the Great
580                  Arisan 2
3052              Irada Pakka
3417                Kita Kita
5695                  Soldier
Name: title, dtype: object
```

In [ ]:
```

```