

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import warnings
6 warnings.filterwarnings("ignore")
```

```
In [2]: 1 netflix_dataset = pd.read_csv("C:/Users/...../Downloads/netflix_dataset.csv")
2 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	Demian Bichir, Héctor Bonilla, Oscar Serrano, ...	Mexico	December 23, 2016	2016	TV-MA	93 min	Dramas, International Movies	di 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...	postaworl ,
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 beci

```
In [3]: 1 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]: 1 # Identifying unique values for each column
2 unique_counts = netflix_dataset.nunique()
3 unique_counts_df = pd.DataFrame(unique_counts, columns=["Unique counts"]) # Just conv. to
4 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]: 1 # Identify the missing values
2 null = netflix_dataset.isnull().sum()
3 # print(null)
4 null_df = null.reset_index()
5 null_df.columns = ['Columns', 'Null_value_count']
6 print(null_df)
```

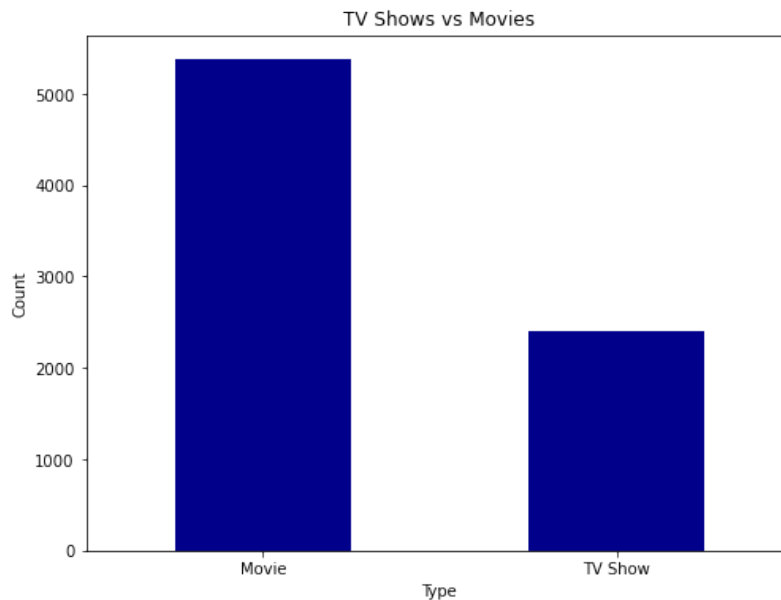
	Columns	Null_value_count
0	show_id	0
1	type	0
2	title	0
3	director	2389
4	cast	718
5	country	506
6	date_added	10
7	release_year	0
8	rating	7
9	duration	0
10	listed_in	0
11	description	0

Analysis of TV Shows and Movies

```
In [6]: 1 netflix_shows=netflix_dataset[netflix_dataset['type']=='TV Show']
2 netflix_movies=netflix_dataset[netflix_dataset['type']=='Movie']
```

```
In [7]: 1 # Count of Movies and TV Shows
2 counts = netflix_dataset['type'].value_counts()
3 print(counts)
4 plt.figure(figsize=(8, 6))
5 counts.plot(kind='bar', color='darkblue')
6 plt.title('TV Shows vs Movies')
7 plt.xlabel('Type')
8 plt.ylabel('Count')
9 plt.xticks(rotation=0) # Optional: Keeps x-axis labels horizontal
10 plt.show()
```

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



TV Shows Analysis

```
In [8]: 1 # Define the desired order of months
2 month_order = ['January', 'February', 'March', 'April', 'May', 'June',
3               'July', 'August', 'September', 'October', 'November', 'December']
4
5 netflix_date = netflix_shows[['date_added']].dropna() # Return non-null values
6 netflix_date['year'] = netflix_date['date_added'].str.split(',').str[-1].str.strip()
7 netflix_date['month'] = netflix_date['date_added'].str.split(',').str[0]
8 netflix_date['month'] = pd.Categorical(netflix_date['month'], categories=month_order, order
9 netflix_date
```

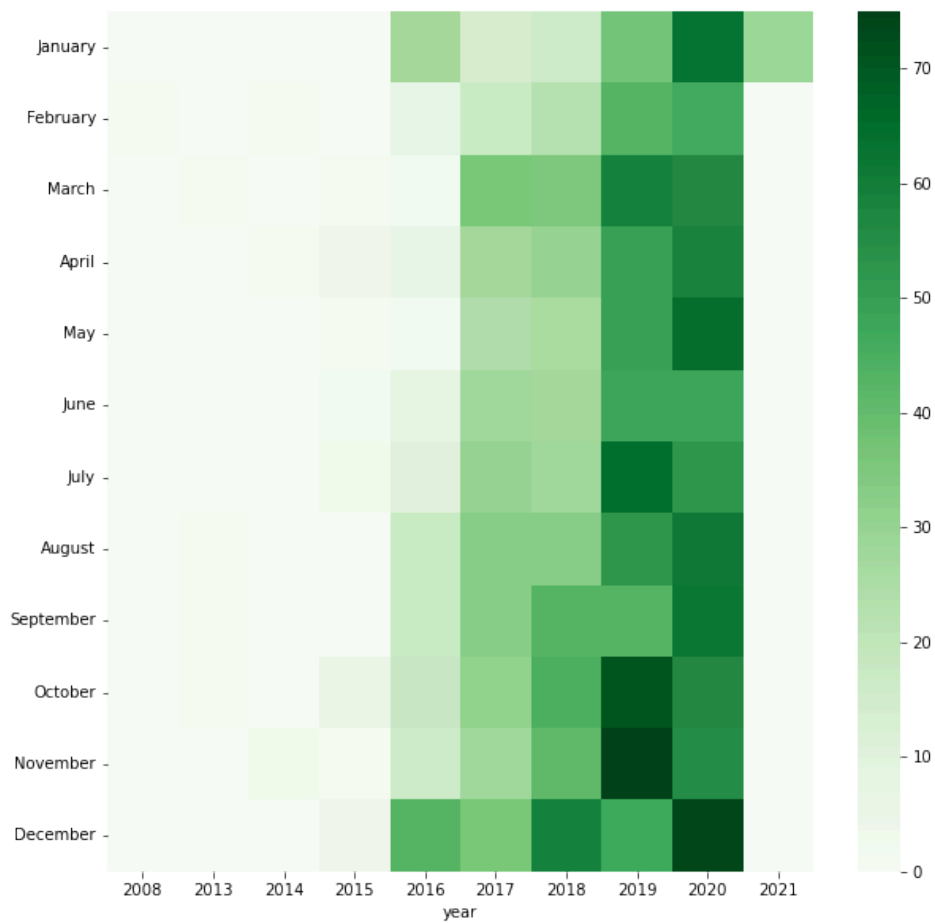
Out[8]:

	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

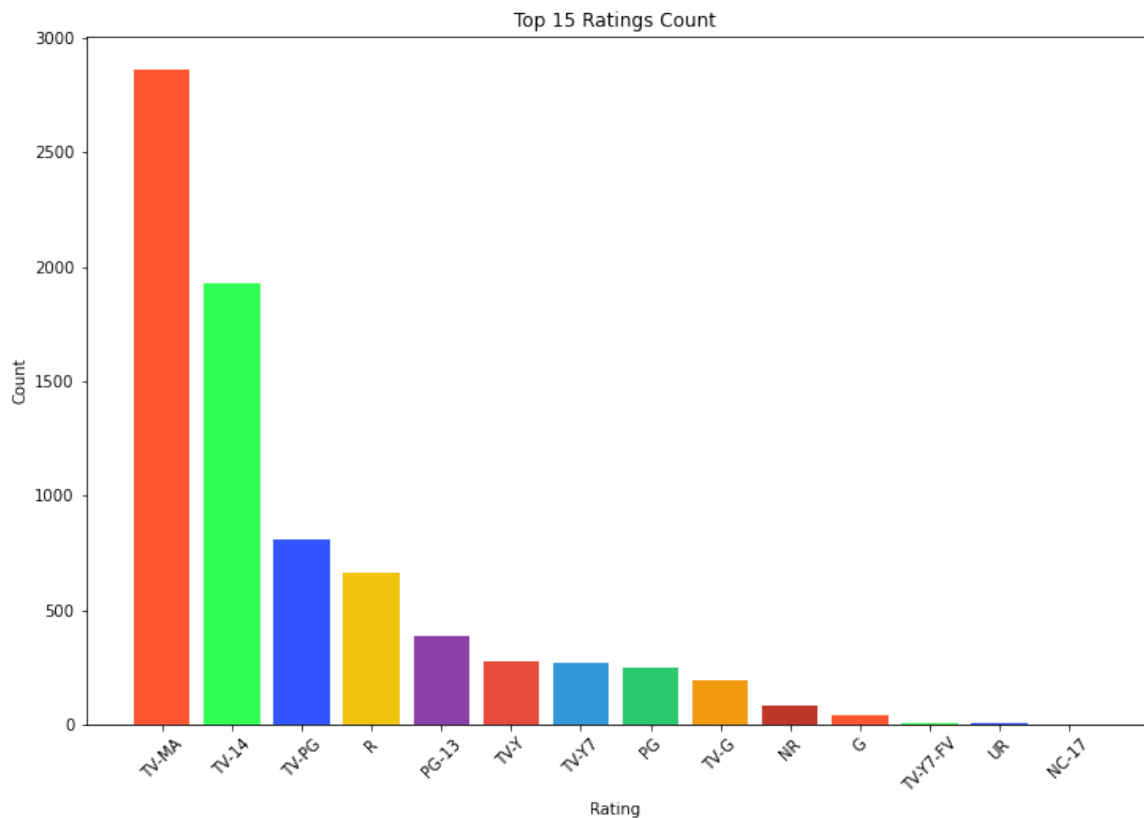
```
In [9]: 1 # Heatmap: Distribution of movies(by date added) by each month in each year
2 df = netflix_date.groupby('year')['month'].value_counts().unstack().fillna(0)[month_order]
3 plt.subplots(figsize=(10,10))
4 sns.heatmap(df,cmap='Greens') #heatmap
```

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



Movie Ratings Analysis

```
In [10]: 1 # Count occurrences of each rating
2 from matplotlib import cm
3 rating_counts = netflix_dataset['rating'].value_counts()
4 # Define a list of colors for the bars
5 colors = ['#FF5733', '#33FF57', '#3357FF', '#F1C40F', '#8E44AD', '#E74C3C', '#3498DB', '#2E86C1', '#9B59B6', '#F1C40F', '#8E44AD', '#E74C3C', '#3498DB', '#2E86C1', '#9B59B6']
6
7 plt.figure(figsize=(12, 8))
8 plt.bar(rating_counts.index, rating_counts.values, color=colors)
9 plt.xticks(rotation=45)
10 plt.xlabel('Rating')
11 plt.ylabel('Count')
12 plt.title('Top 15 Ratings Count')
13 plt.show()
```



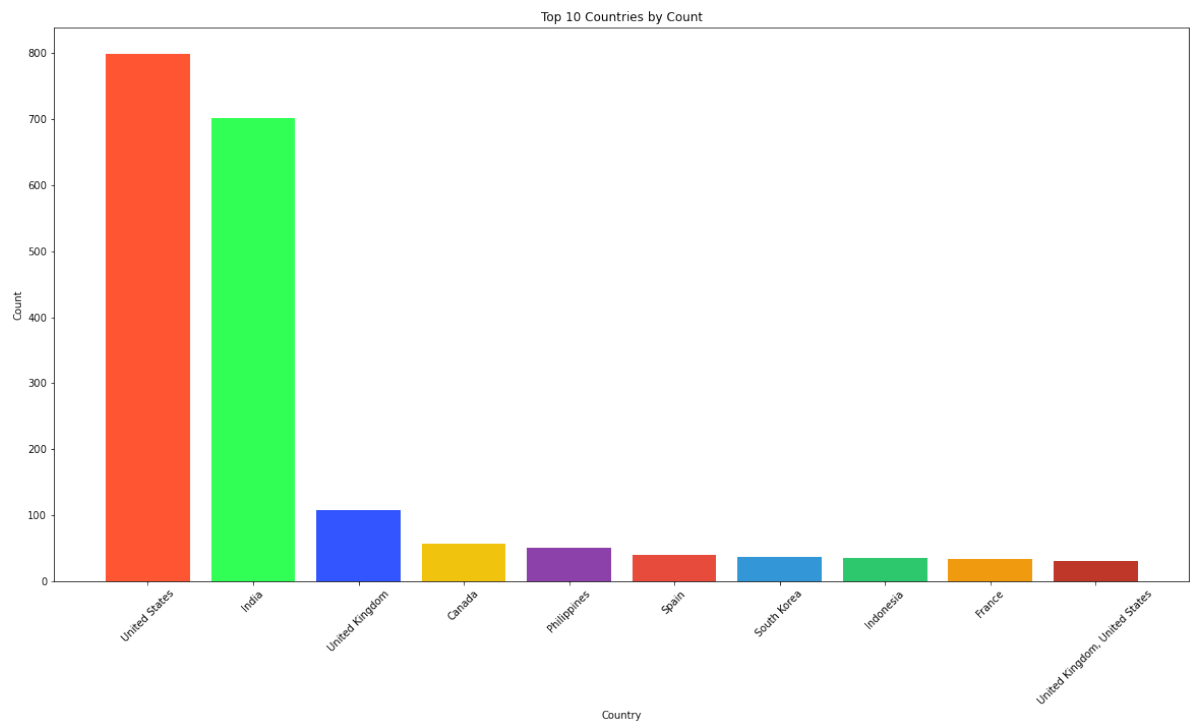
```
In [11]: 1 #Analysing IMDB ratings to get top rated movies on Netflix
2 imdb_ratings=pd.read_csv("C:/Users/...../Downloads/IMDb ratings.csv", usecols=['weighted_av
3 imdb_titles=pd.read_csv("C:/Users/...../Downloads/IMDb movies.csv", usecols=['title','year'
4
5 ratings = pd.DataFrame({'Title':imdb_titles.title,
6                         'Release Year':imdb_titles.year,
7                         'Rating': imdb_ratings.weighted_average_vote,
8                         'Genre':imdb_titles.genre})
9 ratings.drop_duplicates(subset=['Title', 'Release Year', 'Rating'], inplace=True)
```

```
In [12]: 1 #joining netflix dataset and imdb dataset
2 ratings.dropna()
3 joint_data=ratings.merge(netflix_dataset,left_on='Title',right_on='title',how='inner')
4 joint_data=joint_data.sort_values(by='Rating', ascending=False)
5 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

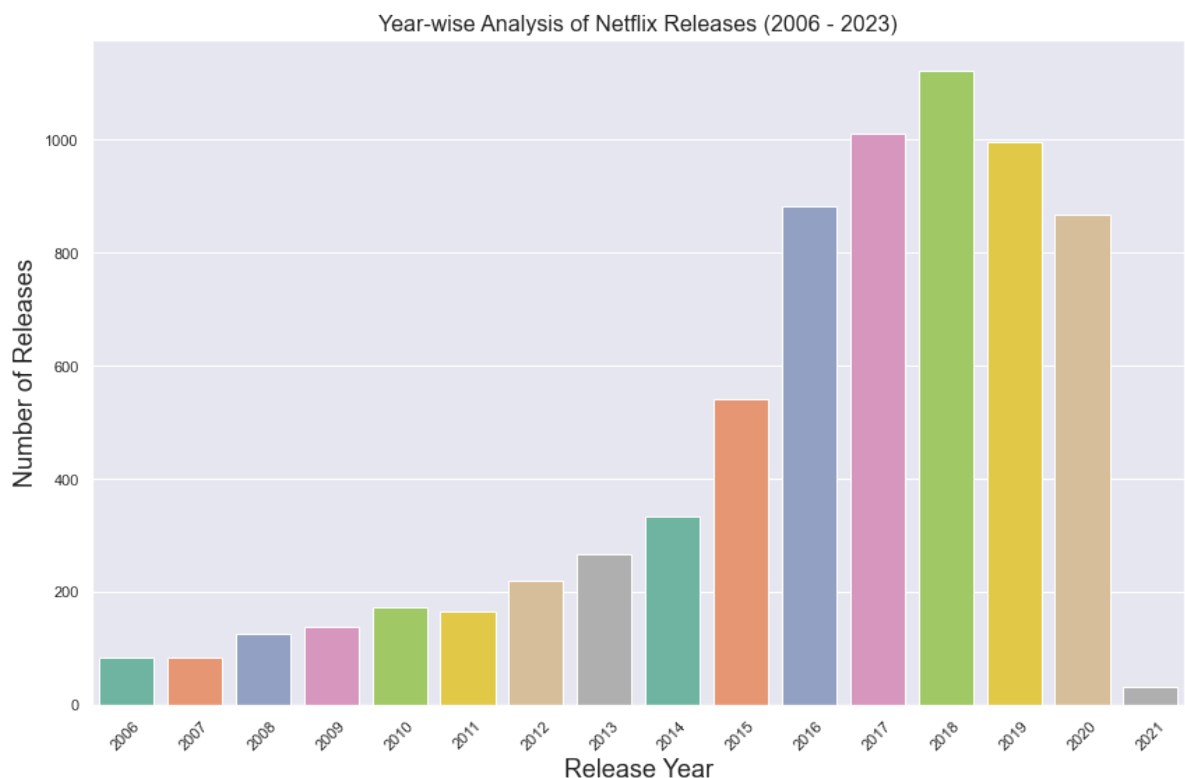
```
In [13]: 1 import matplotlib.pyplot as plt
2
3 country_count = joint_data['country'].value_counts().sort_values(ascending=False)
4 country_count = pd.DataFrame({'Country': country_count.index, 'Count': country_count.values})
5 top_countries = country_count.head(10)
6
7 # Define a list of colors for the bars
8 colors = ['#FF5733', '#33FF57', '#3357FF', '#F1C40F', '#8E44AD', '#E74C3C', '#3498DB', '#2E86C1',
9          '#9B59B6', '#F1C40F']
10
11 plt.figure(figsize=(20, 10))
12 plt.bar(top_countries['Country'], top_countries['Count'], color=colors)
13 plt.xlabel('Country')
14 plt.ylabel('Count')
15 plt.title('Top 10 Countries by Count')
16 plt.xticks(rotation=45)
17 plt.show()
```



```

In [14]: 1 # Year-wise Analysis for the last 20 years
2 recent_years = netflix_dataset[netflix_dataset['release_year'] > 2005]
3
4 # Grouping data by release year and counting occurrences
5 year_counts = recent_years['release_year'].value_counts().sort_index()
6
7 # Plotting the year-wise analysis
8 plt.figure(figsize=(12, 8))
9 sns.set(style="darkgrid")
10
11 # Creating a bar plot
12 sns.barplot(x=year_counts.index, y=year_counts.values, palette="Set2")
13 # Adding titles and labels
14 plt.title('Year-wise Analysis of Netflix Releases (2006 - 2023)', fontsize=16)
15 plt.xlabel('Release Year', fontsize=18)
16 plt.ylabel('Number of Releases', fontsize=18)
17
18 # Displaying the plot
19 plt.xticks(rotation=45)
20 plt.tight_layout()
21 plt.show()

```



Analysis of Duration of Movies

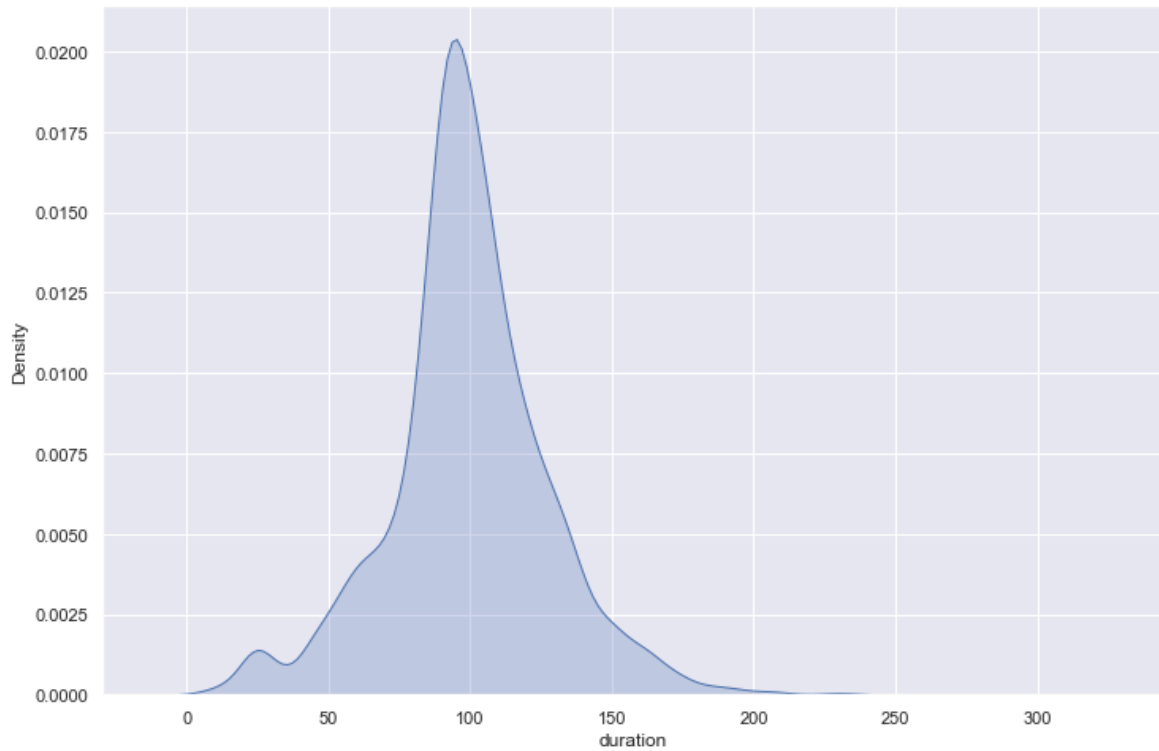
```

In [15]: 1 netflix_movies['duration'] = netflix_movies['duration'].astype(str).str.replace(' min', '')
2 netflix_movies['duration'] = netflix_movies['duration'].astype(int)

```



```
In [16]: 1 sns.set(style="darkgrid")
2 plt.figure(figsize=(12,8))
3 ax=sns.kdeplot(data=netflix_movies['duration'], shade=True)
```



Analysis of Duration of Series

```
In [17]: 1 features=['title','duration']
2 durations= netflix_shows[features]
3 durations['no_of_seasons']=durations['duration'].str.replace(' Season','')
4 durations['no_of_seasons']=durations['no_of_seasons'].str.replace('s','')
5 durations['no_of_seasons']=durations['no_of_seasons'].astype(str).astype(int)
```

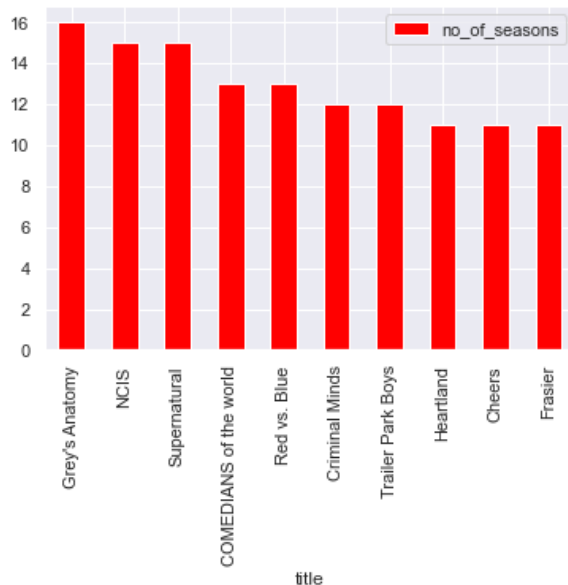
```

In [18]: 1 #TV shows with Largest number of seasons
2 t=['title','no_of_seasons']
3 top=duration[t]
4
5 top=top.sort_values(by='no_of_seasons', ascending=False)
6 top10=top[0:10]
7 plt.figure(figsize=(10,8))
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]: 1 #Plot description based Recommender
2
3 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 $\log(\text{number of documents/documents with term})$. 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]: 1 #Recommendation System(Content Based)
2
3 from sklearn.feature_extraction.text import TfidfVectorizer
4 #Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a', etc.
5 tfidf = TfidfVectorizer(stop_words='english')
6 netflix_dataset['description'] = netflix_dataset['description'].fillna('')
7 tfidf_matrix = tfidf.fit_transform(netflix_dataset['description'])
8 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: $\text{similarity} = \cos(x, y) = (x \cdot y) / (||x|| * ||y||)$, defines how to compute the cosine similarity where:

x and y are the feature vectors of the two movies, $x \cdot y$ 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]: 1 from sklearn.metrics.pairwise import linear_kernel
2 # Cal cosine_sim:
3 cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
4 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]: 1 # Construct a reverse map if indices and movie titles
2
3 indices = pd.Series(netflix_dataset.index, index = netflix_dataset['title']).drop_duplicates
```

```
In [23]: 1 # Function that takes in movie title as input and outputs most similar movies
2
3 def get_recommendations(title, cosine_sim=cosine_sim):
4     # Get the index of the movie that matches the title
5     idx = indices[title]
6
7     # Get the pairwise similarity scores of all movies with that movie
8     sim_scores = list(enumerate(cosine_sim[idx]))
9
10    # Sort the movies based on the similarity scores
11    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
12
13    # Get the scores of the 10 most similar movies
14    sim_scores = sim_scores[1:11]
15
16    # Get the movie indices
17    movie_indices = [i[0] for i in sim_scores]
18
19    # Return the top 10 most similar movies
20    return netflix_dataset['title'].iloc[movie_indices]
```

```
In [24]: 1 # Test
2
3 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]: 1 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]: 1 #Filling null values with empty string.
2 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]: 1 def clean_data(x):
2         return str.lower(x.replace(" ", ""))
3         #Identifying features on which the model is to be filtered.
4         features=['title', 'director', 'cast', 'listed_in', 'description']
5         filledna=filledna[features]
```

```
In [28]: 1 for feature in features:
2         filledna[feature] = filledna[feature].apply(clean_data)
3         filledna.head()
```

Out[28]:

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

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]: 1 def create_soup(x):
2         return x['title'] + ' ' + x['director'] + ' ' + x['cast'] + ' ' + x['listed_in'] + ' ' + x['description']
3
4         filledna['soup'] = filledna.apply(create_soup, axis=1)
5         filledna.tail()
```

Out[29]:

	title	director	cast	listed_in
7782	zozo	joseffares	imadcreidi,antoinetteturk,eliasgergi,carmenleb...	dramas,
7783	zubaan	mozezsingh	vickykaushal,sarah-janedias,raaghavchanana,man...	dramas,internationalmovi
7784	zulumanin	japan	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]: 1 # Import CountVectorizer and create the count matrix
2 from sklearn.feature_extraction.text import CountVectorizer
3
4 count = CountVectorizer(stop_words='english')
5 count_matrix = count.fit_transform(filledna['soup'])
6
7 from sklearn.metrics.pairwise import cosine_similarity
8 cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
9
10 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]: 1 # Reset index of our main DataFrame and construct reverse mapping as before
2 filledna=filledna.reset_index()
3 indices = pd.Series(filledna.index, index=filledna['title'])
```

```
In [32]: 1 def get_recommendations_new(title, cosine_sim=cosine_sim):
2     title=title.replace(' ', '').lower()
3     idx = indices[title]
4
5     # Get the pairwise similarity scores of all movies with that movie
6     sim_scores = list(enumerate(cosine_sim[idx]))
7
8     # Sort the movies based on the similarity scores
9     sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
10
11     # Get the scores of the 10 most similar movies
12     sim_scores = sim_scores[1:11]
13
14     # Get the movie indices
15     movie_indices = [i[0] for i in sim_scores]
16
17     # Return the top 10 most similar movies
18     return netflix_dataset['title'].iloc[movie_indices]
```

```
In [33]: 1 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]: 1 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]: 1 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 [ ]: 1
```