Intro

TV Shows and Movies listed on Netflix This dataset consists of tv shows and movies available on Netflix as of 2019. The dataset is collected from Flixable which is a third-party Netflix search engine.

In 2018, they released an interesting report which shows that the number of TV shows on Netflix has nearly tripled since 2010. The streaming service's number of movies has decreased by more than 2,000 titles since 2010, while its number of TV shows has nearly tripled. It will be interesting to explore what all other insights can be obtained from the same dataset.

The dataset was retrieved on 8/7/2021 from kaggle

Dataset

- Show ID - unique ID of that particular show
- Type - type of the video - movie, TV Series etc.
- Title - title of the video
- Director - director name
- Cast - cast members
- Country - country where content was produced.
- Data Added - date when it became live on NETFLIX
- Release Year - year of release
- Rating - motion picture/Serie content rating system
- Duration - duration of the movie, TV Series etc.
- Listed in - Genre information
- Description - concise plot of the series

Goal

- Make a recomander based on simularity

Imports

In [1]:
```
import pandas as pd
import missingno as msno
import numpy as np
from collections import Counter

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

Data exploration

In [2]:
```
df = pd.read_csv("netflix_titles.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duratic
0	s1	TV Show	3%	NaN	João Miguel, Bianca Comparato, Michel Gomes, R...	Brazil	August 14, 2020	2020	TV-MA	Seasor
1	s2	Movie	7:19	Jorge Michel Grau	Demián Bichir, Héctor Bonilla, Oscar Serrano, ...	Mexico	December 23, 2016	2016	TV-MA	93 m
2	s3	Movie	23:59	Gilbert Chan	Tedd Chan, Stella Chung, Henley Hii, Lawrence ...	Singapore	December 20, 2018	2011	R	78 m
3	s4	Movie	9	Shane Acker	Elijah Wood, John C. Reilly, Jennifer Connelly...	United States	November 16, 2017	2009	PG-13	80 m
4	s5	Movie	21	Robert Luketic	Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar...	United States	January 1, 2020	2008	PG-13	123 m

In [4]:
```
df.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       7280 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 [5]:
```
df.shape
```

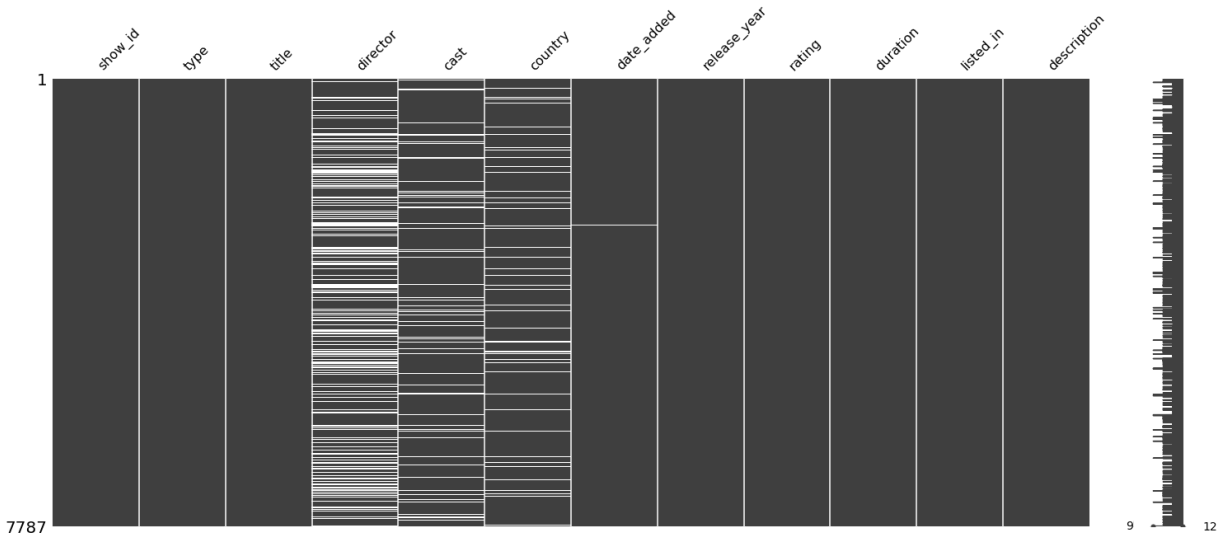
Out[5]:
```
(7787, 12)
```

In [6]:
```
#check missing data
df.isnull().sum()
```

Out[6]:
```
show_id            0
type               0
title              0
director        2389
cast             718
country          507
date_added        10
release_year       0
rating             7
duration           0
listed_in          0
description        0
dtype: int64
```

In [7]:
```
# Visualize missing values as a matrix
msno.matrix(df)
```

Out[7]: <AxesSubplot:>

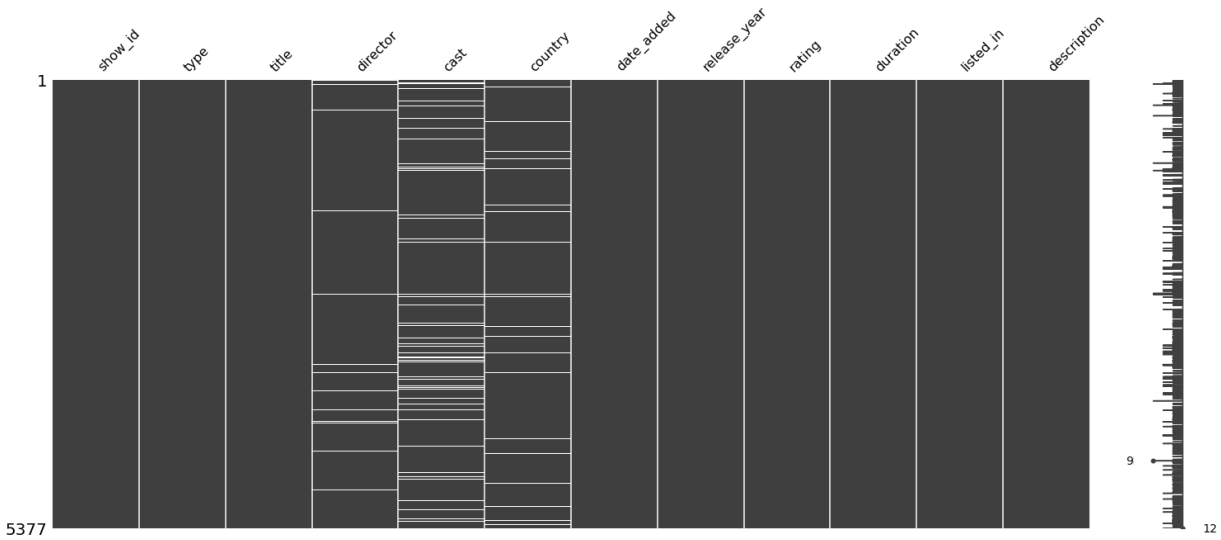


As you can see there is lots of missing values for the director's since there are series and movies it might be that series have no director

In [8]:
```
# Make dataset with only movies
dfMovies = df[df["type"] == "Movie"]
```

In [9]:
```
# Check missing values movies
msno.matrix(dfMovies)
```

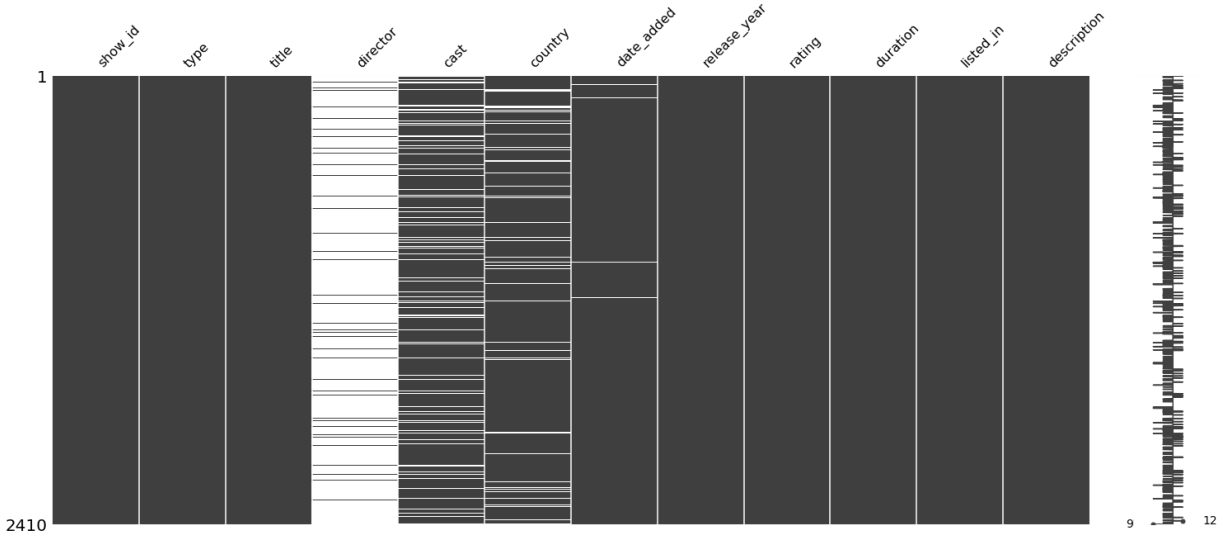
Out[9]: <AxesSubplot:>



In [10]:
```
# Make dataset with only series
dfSeries = df[df["type"] == "TV Show"]
```

In [11]:
```
# Check missing values series
msno.matrix(dfSeries)
```

Out[11]: <AxesSubplot:>

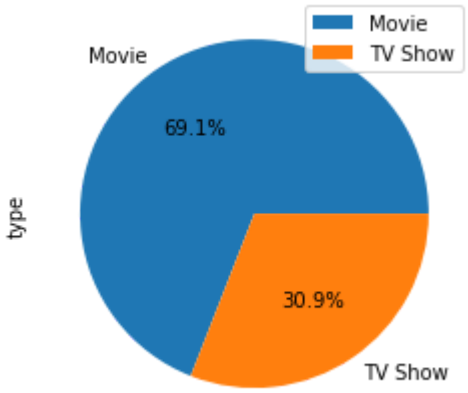


In [12]:
```
# Series don't have one director so it is logical to not have them
```

Data Visualisation

In [13]:
```
# What procentage of the data are movies?
df["type"].value_counts().plot(kind="pie",autopct="%1.1f%%")
plt.legend()
```

Out[13]: <matplotlib.legend.Legend at 0x1ea6e51caf0>



In [14]:
```
# So around 69% of what was published were movies
```

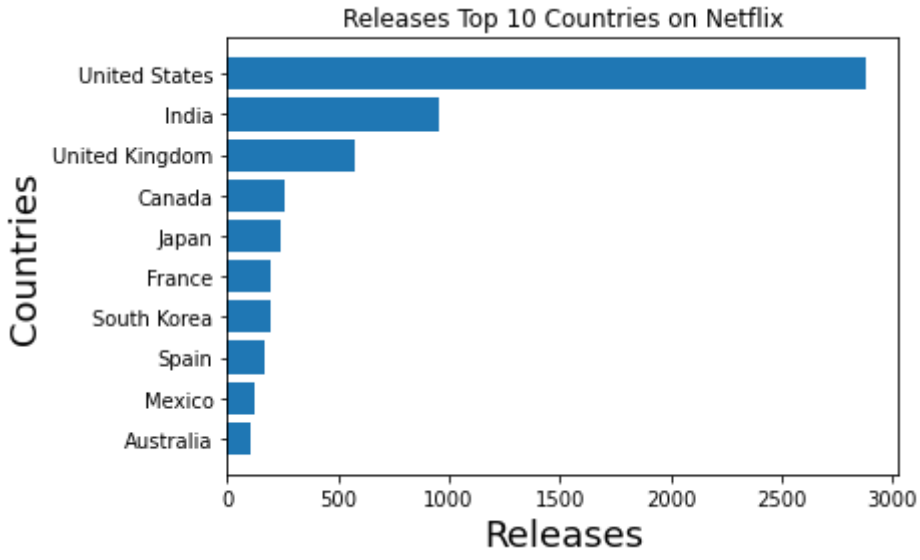
In [15]:
```
# How many countries are in the dataset?
dfCountries = df["country"]
dfCountries = dfCountries.dropna()
dfCountries = dfCountries.tolist()
dfCountries = [i.split(',')[0] for i in dfCountries]
country_dict = dict([(i, dfCountries.count(i)) for i in set(dfCountries) ] )
country_dict = dict(sorted(country_dict.items(), key=lambda x: x[1], reverse=True))
print('There are: ' + str(len(country_dict)) + " countries in this dataset")
```

```
There are: 81 countries in this dataset
```

In [16]:
```
country_dict = dict(sorted(country_dict.items(), key=lambda x: x[1], reverse=True)[:
print(country_dict)

plt.barh(range(len(country_dict)), country_dict.values(), align='center')
plt.yticks(range(len(country_dict)), country_dict.keys())
plt.gca().invert_yaxis()
plt.xlabel('Releases', fontsize=18)
plt.ylabel('Countries', fontsize=18)
plt.title('Releases Top 10 Countries on Netflix')
plt.show()
```

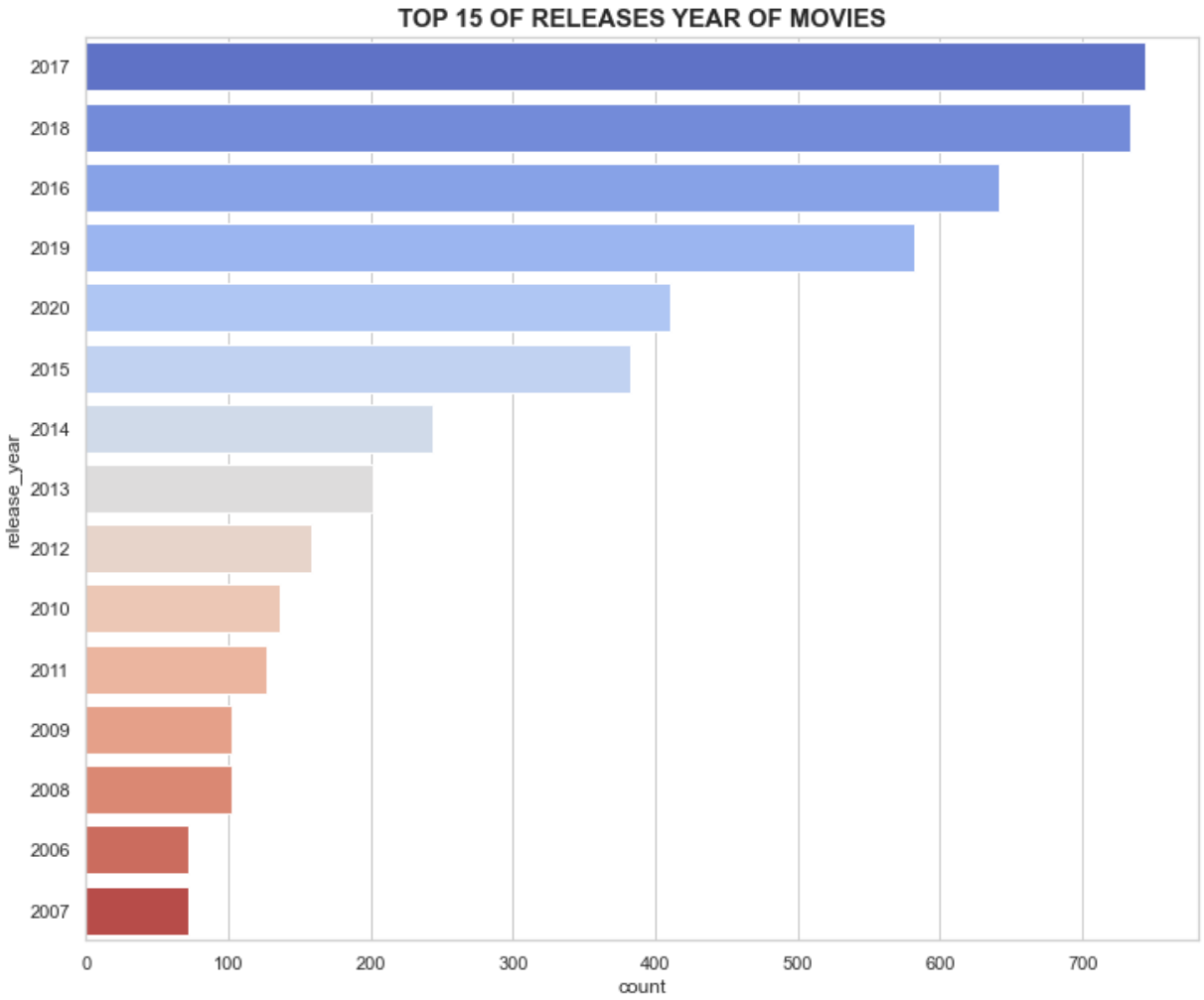
```
{'United States': 2883, 'India': 956, 'United Kingdom': 577, 'Canada': 259, 'Japan':
237, 'France': 196, 'South Korea': 194, 'Spain': 168, 'Mexico': 123, 'Australia': 10
8}
```



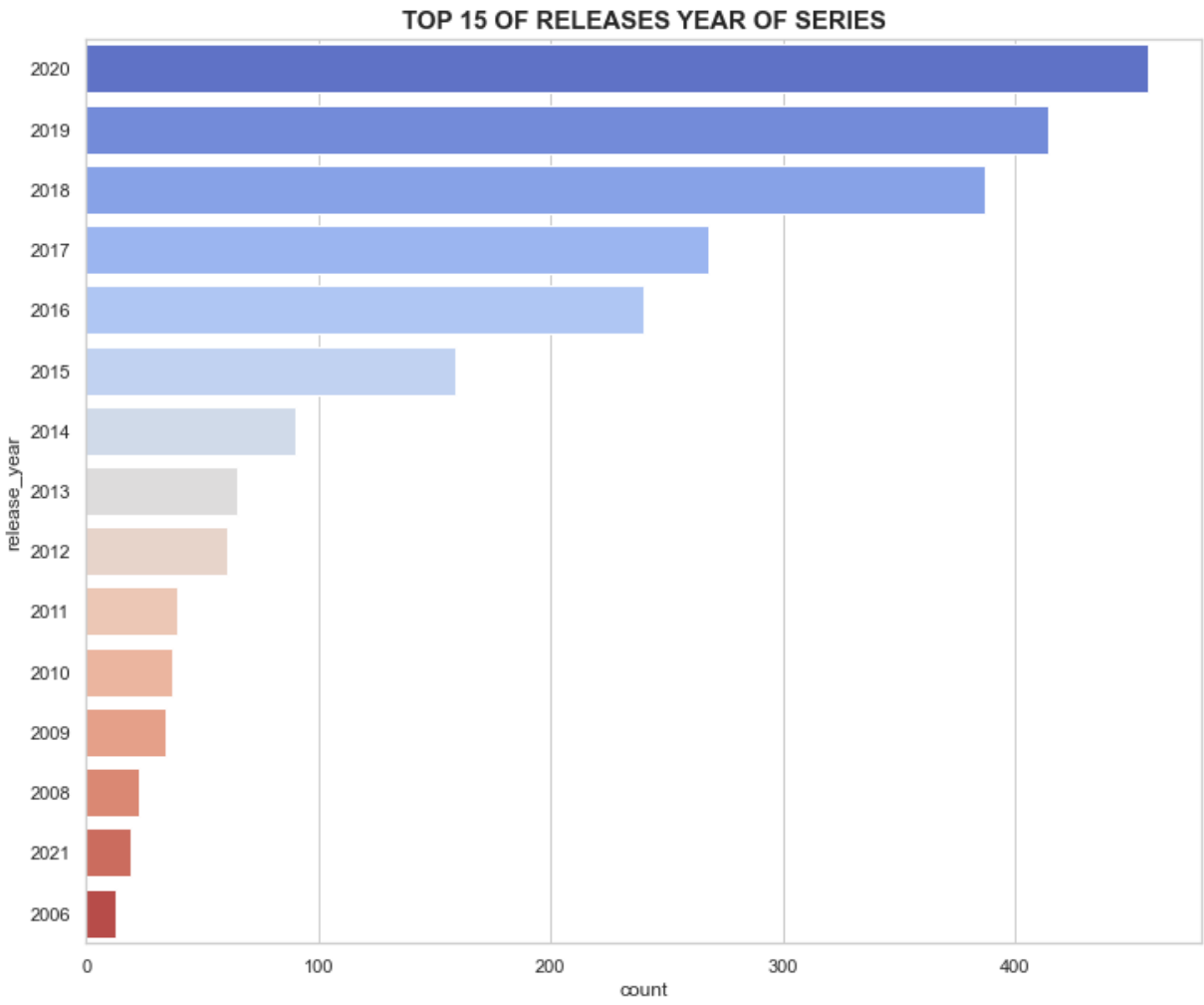
In [17]:
```
# The United States released the most series/movies on netflix
```

In [18]:
```
#Which years released the most movies
```

```
plt.figure(figsize=(12,10))
sns.set(style="whitegrid")
ax = sns.countplot(y="release_year", data=dfMovies, palette="coolwarm", order=dfMovi
plt.title('TOP 15 OF RELEASES YEAR OF MOVIES', fontsize=15, fontweight='bold')
plt.show()
```



In [19]:
```
#Which years released the most series
plt.figure(figsize=(12,10))
sns.set(style="whitegrid")
ax = sns.countplot(y="release_year", data=dfSeries, palette="coolwarm", order=dfSeri
plt.title('TOP 15 OF RELEASES YEAR OF SERIES', fontsize=15, fontweight='bold')
plt.show()
```



Data Cleaning

In [20]:
```
# I want to use the director, cast , title, discription and listing_in and for serie
# title, discription and listing_in

def clean_data(x):
    return str.lower(x.replace(' ', ''))
```

In [21]:
```
features = ['title', 'director', 'cast', 'listed_in', 'description']
df = df.fillna(' ')
df_features = df[features]


for feature in features:
    df_features[feature] = df_features[feature].apply(clean_data)

df_features.head(2)
df_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7787 entries, 0 to 7786
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   title        7787 non-null   object
 1   director     7787 non-null   object
 2   cast         7787 non-null   object
 3   listed_in    7787 non-null   object
 4   description  7787 non-null   object
dtypes: object(5)
memory usage: 304.3+ KB
```

```
<ipython-input-21-84cd026df8d1>:7: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_features[feature] = df_features[feature].apply(clean_data)
```

In [22]:
```
def create_soup(x):
    return x['title'] + ' ' + x['director'] + ' ' + x['cast'] + ' ' + x['listed_in']
```

In [23]:
```
df_features['soup'] = df_features.apply(create_soup, axis = 1)
```

```
<ipython-input-23-9d1e0d897a1e>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  df_features['soup'] = df_features.apply(create_soup, axis = 1)
```

In [24]:
```
df_features['soup']
```

Out[24]:
```
0       3%  joãomiguel,biancacomparato,michelgomes,rod...
1       7:19 jorgemichelgrau demiánbichir,héctorbonill...
2       23:59 gilbertchan teddchan,stellachung,henleyh...
3       9 shaneacker elijahwood,johnc.reilly,jenniferc...
4       21 robertluketic jimsturgess,kevinspacey,kateb...
                              ...
7782    zozo joseffares imadcreidi,antoinetteturk,elia...
7783    zubaan mozezsingh vickykaushal,sarah-janedias,...
7784    zulumaninjapan  nastyc documentaries,internati...
7785    zumbo'sjustdesserts  adrianozumbo,rachelkhoo i...
7786    zztop:thatlittleol'bandfromtexas samdunn  docu...
Name: soup, Length: 7787, dtype: object
```

Building the recomanders

recommender using director, cast, listed_in description and title

In [25]:
```
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df_features['soup'])

cosine_sim_score = cosine_similarity(count_matrix, count_matrix)
```

In [42]:
```
df_features = df_features.reset_index()
indices = pd.Series(df_features.index, index = df_features['title'])
```

Out[42]:
```
title
3%                                    0
7:19                                  1
23:59                                 2
9                                     3
21                                    4
                                   ...
zozo                               7782
zubaan                             7783
zulumaninjapan                     7784
zumbo'sjustdesserts                7785
zztop:thatlittleol'bandfromtexas   7786
Length: 7787, dtype: int64
```

In [27]:
```
def recommendations(title, cosine_sim = cosine_sim_score):

    title = title.replace(' ', '').lower()
    idx = indices[title]
```

```
    # Get the pairwise similarity scores of all movies with that movies
    sim_scores = list(enumerate(cosine_sim[idx]))
#     print(sim_scores)
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse = True)
    print(sim_scores[:11])
    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

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

    # Return the top 10 Similar movies
    return df['title'].iloc[movie_indices]
```

In [28]:
```
recommendations('American History x')
```

```
[(466, 1.0000000000000002), (7463, 0.16666666666666669), (476, 0.14433756729740646),
(3518, 0.1307440900921227), (993, 0.12598815766974242), (6851, 0.12171612389003691),
(5288, 0.11111111111111113), (1609, 0.10540925533894599), (2939, 0.1054092553389459
9), (153, 0.10050378152592121), (4969, 0.09829463743659811)]
```

Out[28]:
```
7463                        Wakefield
476               American Warfighter
3518                        Lady-Like
993                              Blow
6851                        The Score
5288                         Rounders
1609                 Dare to Be Wild
2939    If Anything Happens I Love You
153                  A Bridge Too Far
4969                       Primal Fear
Name: title, dtype: object
```

In [29]:
```
recommendations('friends')
```

```
[(2288, 0.9999999999999998), (2015, 0.3076923076923077), (2263, 0.2401922307076307
4), (6082, 0.24019223070763074), (5818, 0.2223747949983304), (1577, 0.21483446221182
984), (4811, 0.21483446221182984), (7120, 0.21483446221182984), (1300, 0.19611613513
818404), (6953, 0.17541160386140586), (3899, 0.16012815380508716)]
```

Out[29]:
```
2015                                Episodes
2263                                 Frasier
6082                The Andy Griffith Show
5818                             Still Game
1577                             Dad's Army
4811                   Pee-wee's Playhouse
7120                       Toast of London
1300                                 Cheers
6953    The Twilight Zone (Original Series)
3899                        Man with a Plan
Name: title, dtype: object
```

In [30]:
```
recommendations('Naruto')
```

```
[(4404, 0.9999999999999999), (4405, 0.27863910628767646), (4410, 0.2519763153394848
4), (4407, 0.24525573579398632), (4906, 0.23904572186687872), (973, 0.23145502494313
79), (4406, 0.2279211529192759), (4408, 0.22291128503014115), (3959, 0.2070196678027
0625), (5345, 0.20701966780270625), (3379, 0.19446111706564934)]
```

Out[30]:
```
4405                   Naruto Shippûden the Movie: Bonds
4410    Naruto the Movie 2: Legend of the Stone of Gelel
4407                     Naruto Shippuden : Blood Prison
4906                                      Pop Team Epic
973                                              Bleach
4406        Naruto Shippûden the Movie: The Will of Fire
4408                         Naruto Shippuden: The Movie
3959                             Marvel Anime: Wolverine
5345                        Saint Seiya: The Lost Canvas
```

```
3379                                        Kill la Kill
Name: title, dtype: object
```

In [31]:
```
recommendations('Making a murderer')
```

```
[(3877, 0.9999999999999999), (1773, 0.3162277660168379), (2886, 0.3162277660168379),
(3013, 0.3162277660168379), (3143, 0.3162277660168379), (3388, 0.3162277660168379),
(6033, 0.3162277660168379), (6238, 0.3162277660168379), (6295, 0.3162277660168379),
(6493, 0.3162277660168379), (7219, 0.3162277660168379)]
```

Out[31]:
```
1773                    Dirty Money
2886                  I AM A KILLER
3013        Inside the Criminal Mind
3143    Jeffrey Epstein: Filthy Rich
3388                  Killer Ratings
6033           Terrorism Close Calls
6238            The Confession Tapes
6295             The Devil Next Door
6493             The Innocence Files
7219                  Trial By Media
Name: title, dtype: object
```

recommender using description

In [64]:
```
# now lets check with only the description
#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')
#Construct a reverse map of indices and movie titles
indices2 = pd.Series(df.index, index = df['title'].str.lower())
```

In [59]:
```
#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df['description'])
```

In [60]:
```
#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

Out[60]:
```
(7787, 17905)
```

In [61]:
```
# Compute the cosine similarity matrix
cosine_sim2 = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [66]:
```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations_version2(title, cosine_sim=cosine_sim2):
    # Get the index of the movie that matches the title
    title = title.lower()
    idx = indices2[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]
    print(sim_scores[:11])
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

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

In [67]:
```
get_recommendations_version2('Making a murderer')
```

```
[(6293, 0.22057712226702508), (6356, 0.19584512500224993), (7476, 0.1922293413024547
```

```
3), (7218, 0.1681788095539813), (6365, 0.1509613877459247), (7269, 0.149040276314683
97), (606, 0.1482888053952589), (5566, 0.1416322895502919), (965, 0.1407907344687655
8), (2349, 0.1381837589439507)]
```

Out[67]:
```
6293              The Devil Inside
6356                      The Five
7476             War Against Women
7218                       Trial 4
6365                    The Force
7269                        Tunnel
606     Asura: The City of Madness
5566                  Shimmer Lake
965                     Black Spot
2349                     Gangaajal
Name: title, dtype: object
```

In [68]:
```
get_recommendations_version2('naruto')
```

```
[(4408, 0.2603523447005411), (4407, 0.22644549478051224), (5330, 0.197495447691230
5), (872, 0.1896595388996567), (4405, 0.1810819500470362), (2501, 0.1791835012046291
4), (4411, 0.16089374092128322), (5345, 0.1570825820111596), (4936, 0.14189031272005
484), (3578, 0.1405978045469308)]
```

Out[68]:
```
4408                          Naruto Shippuden: The Movie
4407                       Naruto Shippuden : Blood Prison
5330                                              Sabrina
872                                 Beyblade: Metal Fusion
4405                     Naruto Shippûden the Movie: Bonds
2501                                              Gormiti
4411    Naruto the Movie 3: Guardians of the Crescent ...
5345                          Saint Seiya: The Lost Canvas
4936                             Power Rangers Ninja Steel
3578                     LEGO Bionicle: The Journey to One
Name: title, dtype: object
```

conclusion

The one with only the discription as vector gives different results than the recommander where discription listing, cast and director is give. Furhtermore it gives less simularty values. Thus the first methods is better. Also with the first method whereby the discription listing, cast and director was used also takes into account for those preffences instead of only the discription

In [ ]:
```
```