In [1]:
```
# CS2170 Final Project
# lenz Baron Balita and Cara L. Roño
```

In [2]:
```
!pip install pyLDAvis==2.1.2
!pip install nltk
!pip install sklearn
!pip install gensim
!pip install seaborn
!pip install plotly
import pandas as pd
import numpy as np
import statsmodels.tools.tools as stattools
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
import random


#topic modeling
import nltk
nltk.download('wordnet')
nltk.download('words')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.util import ngrams
import nltk, re, string, gensim
from nltk.corpus import stopwords, wordnet as wn
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from collections import defaultdict
from gensim import corpora
from nltk.stem import SnowballStemmer

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
Requirement already satisfied: pyLDAvis==2.1.2 in c:\python39\lib\site-packages (2.1.2)
Requirement already satisfied: scipy>=0.18.0 in c:\users\slash\appdata\roaming\python\python39\site-packages (frc
Requirement already satisfied: numpy>=1.9.2 in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (1.19.3)
Requirement already satisfied: future in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (0.18.2)
```

```
Requirement already satisfied: pytest in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (6.2.2)
Requirement already satisfied: numexpr in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (2.7.2)
Requirement already satisfied: wheel>=0.23.0 in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (0.36.2)
Requirement already satisfied: joblib>=0.8.4 in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (1.0.0)
Requirement already satisfied: jinja2>=2.7.2 in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (2.11.2)
Requirement already satisfied: pandas>=0.17.0 in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (1.2.1)
Requirement already satisfied: funcy in c:\python39\lib\site-packages (from pyLDAvis==2.1.2) (1.15)
Requirement already satisfied: MarkupSafe>=0.23 in c:\python39\lib\site-packages (from jinja2>=2.7.2->pyLDAvis==2
Requirement already satisfied: python-dateutil>=2.7.3 in c:\python39\lib\site-packages (from pandas>=0.17.0->pyLD
Requirement already satisfied: pytz>=2017.3 in c:\python39\lib\site-packages (from pandas>=0.17.0->pyLDAvis==2.1.
Requirement already satisfied: six>=1.5 in c:\python39\lib\site-packages (from python-dateutil>=2.7.3->pandas>=0.
Requirement already satisfied: toml in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2) (0.10.2)
Requirement already satisfied: py>=1.8.2 in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2) (1.10.0)
Requirement already satisfied: atomicwrites>=1.0 in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2)
Requirement already satisfied: iniconfig in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2) (1.1.1)
Requirement already satisfied: attrs>=19.2.0 in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2) (20.
Requirement already satisfied: pluggy<1.0.0a1,>=0.12 in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1
Requirement already satisfied: packaging in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2) (20.4)
Requirement already satisfied: colorama in c:\python39\lib\site-packages (from pytest->pyLDAvis==2.1.2) (0.4.4)
Requirement already satisfied: pyparsing>=2.0.2 in c:\python39\lib\site-packages (from packaging->pytest->pyLDAvi

WARNING: You are using pip version 20.3.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.
WARNING: You are using pip version 20.3.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.

Requirement already satisfied: nltk in c:\python39\lib\site-packages (3.5)
Requirement already satisfied: joblib in c:\python39\lib\site-packages (from nltk) (1.0.0)
Requirement already satisfied: tqdm in c:\python39\lib\site-packages (from nltk) (4.56.2)
Requirement already satisfied: regex in c:\python39\lib\site-packages (from nltk) (2020.11.13)
Requirement already satisfied: click in c:\python39\lib\site-packages (from nltk) (7.1.2)
Requirement already satisfied: sklearn in c:\python39\lib\site-packages (0.0)
Requirement already satisfied: scikit-learn in c:\python39\lib\site-packages (from sklearn) (0.24.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\python39\lib\site-packages (from scikit-learn->sklearn)
Requirement already satisfied: numpy>=1.13.3 in c:\python39\lib\site-packages (from scikit-learn->sklearn) (1.19.
Requirement already satisfied: joblib>=0.11 in c:\python39\lib\site-packages (from scikit-learn->sklearn) (1.0.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\slash\appdata\roaming\python\python39\site-packages (frc

WARNING: You are using pip version 20.3.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.

Requirement already satisfied: gensim in c:\python39\lib\site-packages (3.8.3)
Requirement already satisfied: numpy>=1.11.3 in c:\python39\lib\site-packages (from gensim) (1.19.3)
Requirement already satisfied: smart-open>=1.8.1 in c:\python39\lib\site-packages (from gensim) (4.1.2)
```

```
Requirement already satisfied: scipy>=0.18.1 in c:\users\slash\appdata\roaming\python\python39\site-packages (frc
Requirement already satisfied: six>=1.5.0 in c:\python39\lib\site-packages (from gensim) (1.15.0)

WARNING: You are using pip version 20.3.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.

Requirement already satisfied: seaborn in c:\python39\lib\site-packages (0.11.0)
Requirement already satisfied: matplotlib>=2.2 in c:\python39\lib\site-packages (from seaborn) (3.3.3)
Requirement already satisfied: numpy>=1.15 in c:\python39\lib\site-packages (from seaborn) (1.19.3)
Requirement already satisfied: pandas>=0.23 in c:\python39\lib\site-packages (from seaborn) (1.2.1)
Requirement already satisfied: scipy>=1.0 in c:\users\slash\appdata\roaming\python\python39\site-packages (from s
Requirement already satisfied: cycler>=0.10 in c:\python39\lib\site-packages (from matplotlib>=2.2->seaborn) (0.1
Requirement already satisfied: pillow>=6.2.0 in c:\python39\lib\site-packages (from matplotlib>=2.2->seaborn) (8.
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\python39\lib\site-packages (from ma
Requirement already satisfied: kiwisolver>=1.0.1 in c:\python39\lib\site-packages (from matplotlib>=2.2->seaborn)
Requirement already satisfied: python-dateutil>=2.1 in c:\python39\lib\site-packages (from matplotlib>=2.2->seabc
Requirement already satisfied: six in c:\python39\lib\site-packages (from cycler>=0.10->matplotlib>=2.2->seaborn)
Requirement already satisfied: pytz>=2017.3 in c:\python39\lib\site-packages (from pandas>=0.23->seaborn) (2020.4

WARNING: You are using pip version 20.3.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.

Requirement already satisfied: plotly in c:\python39\lib\site-packages (4.14.3)
Requirement already satisfied: six in c:\python39\lib\site-packages (from plotly) (1.15.0)
Requirement already satisfied: retrying>=1.3.3 in c:\python39\lib\site-packages (from plotly) (1.3.3)

WARNING: You are using pip version 20.3.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\slash\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\slash\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\slash\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\slash\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

I. Data Gathering

the dataset that will be used in this study is sourced from Kaggle. The dataset is about Netflix movies and TV shows. It contains more than 7 each of their description, genres, title, directors, and much more.

In importing the dataset there are two path specified either by colab or jupyter.

In [3]:
```
#If will be opened in jupyter remove this
#from google.colab import drive
#drive.mount('/content/drive')
```

In [4]:
```
#path for Jupyter
df=pd.read_csv(r"..\CS170-BALITA-RONO-FINALPROJECT\netflix_titles.csv")

#path for colab
#df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/CS170 Project/netflix_titles.csv')
df.head(5)
```

Out[4]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duratio
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 [5]:
```
# The data set contains the following columns
df.columns
```

Out[5]:
```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description'],
      dtype='object')
```

In [6]:
```
#With 7787 rows and 12 columns
df.shape
```

Out[6]:
```
(7787, 12)
```

#II. Data Cleaning

In here we will first look into the type of fields of each column and to know what we're working on. The data set has quite a few columns thus study. We will also change some names of some columns to make them suitable for their contents.

The data set will be split into Movies and Shows because it will used seperately to see differences of results between movies and shows. Th values and if there are any will be deleted or filled. Some numeric field will also be standardized to check some outliers in the data.

In [7]:
```
# checking data types of fields
df.dtypes
```

Out[7]:
```
show_id         object
type            object
title           object
director        object
cast            object
country         object
date_added      object
release_year     int64
rating          object
duration        object
listed_in       object
description     object
dtype: object
```

In [8]:
```
# Dropping irrelevant columns that will not be used.
df = df.drop(['show_id', 'cast', ], axis=1)
df.head(2)
```

Out[8]:

	type	title	director	country	date_added	release_year	rating	duration	listed_
0	TV Show	3%	NaN	Brazil	August 14, 2020	2020	TV-MA	4 Seasons	International TV Shows, TV Dramas, TV Sci-Fi &
1	Movie	7:19	Jorge Michel Grau	Mexico	December 23, 2016	2016	TV-MA	93 min	Dramas, International Movi

In [9]:
```
# renaming some column names.
df = df.rename(columns={"listed_in":"genre"})
df.columns
```

Out[9]:
```
Index(['type', 'title', 'director', 'country', 'date_added', 'release_year',
       'rating', 'duration', 'genre', 'description'],
      dtype='object')
```

In [10]:
```
# splitting the dataset between Movies and TV shows
netflix_movies = df[df['type'] == 'Movie']
netflix_shows = df[df['type'] == 'TV Show']
```

In [11]:
```
#checking duplicate rows. There are no duplicate rows
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

```
number of duplicate rows:  (0, 10)
```

In [12]:
```
#checking null columns
print(df.isnull().sum())
```

```
type              0
title             0
director       2389
country         507
date_added       10
release_year      0
rating            7
duration          0
genre             0
description       0
dtype: int64
```

In [13]:
```
#replacing null values with 'not specified' or none

df.fillna('Not specified')
df.replace(np.nan, ' none')
```

Out[13]:

	type	title	director	country	date_added	release_year	rating	d
0	TV Show	3%	none	Brazil	August 14, 2020	2020	TV-MA	
1	Movie	7:19	Jorge Michel Grau	Mexico	December 23, 2016	2016	TV-MA	
2	Movie	23:59	Gilbert Chan	Singapore	December 20, 2018	2011	R	
3	Movie	9	Shane Acker	United States	November 16, 2017	2009	PG-13	
4	Movie	21	Robert Luketic	United States	January 1, 2020	2008	PG-13	
...	...	...	...	...	...	...	...	
7782	Movie	Zozo	Josef Fares	Sweden, Czech Republic, United Kingdom, Denmar...	October 19, 2020	2005	TV-MA	
7783	Movie	Zubaan	Mozez Singh	India	March 2, 2019	2015	TV-14	
7784	Movie	Zulu Man in Japan	none	none	September 25, 2020	2019	TV-MA	
7785	TV Show	Zumbo's Just Desserts	none	Australia	October 31, 2020	2019	TV-PG	
7786	Movie	ZZ TOP: THAT LITTLE OL' BAND FROM TEXAS	Sam Dunn	United Kingdom, Canada, United States	March 1, 2020	2019	TV-MA	

7787 rows × 10 columns

In [14]:
```
#standardizing numeric field 'release_year'
from scipy import stats
df['release_year_z']= stats.zscore(df['release_year'])
df['release_year_z']
```

Out[14]:
```
0       0.692878
1       0.236092
2      -0.334890
3      -0.563284
4      -0.677480
          ...
7782   -1.020070
7783    0.121896
7784    0.578682
7785    0.578682
7786    0.578682
Name: release_year_z, Length: 7787, dtype: float64
```

In [15]:
```
# detecting outliers z-values wither greater than 3 or less than -3. There are no outliers.
outliers = df.query('release_year_z>3 | release_year<-3')
outliers
```

Out[15]:

type	title	director	country	date_added	release_year	rating	duration	genre	description	release_year_z

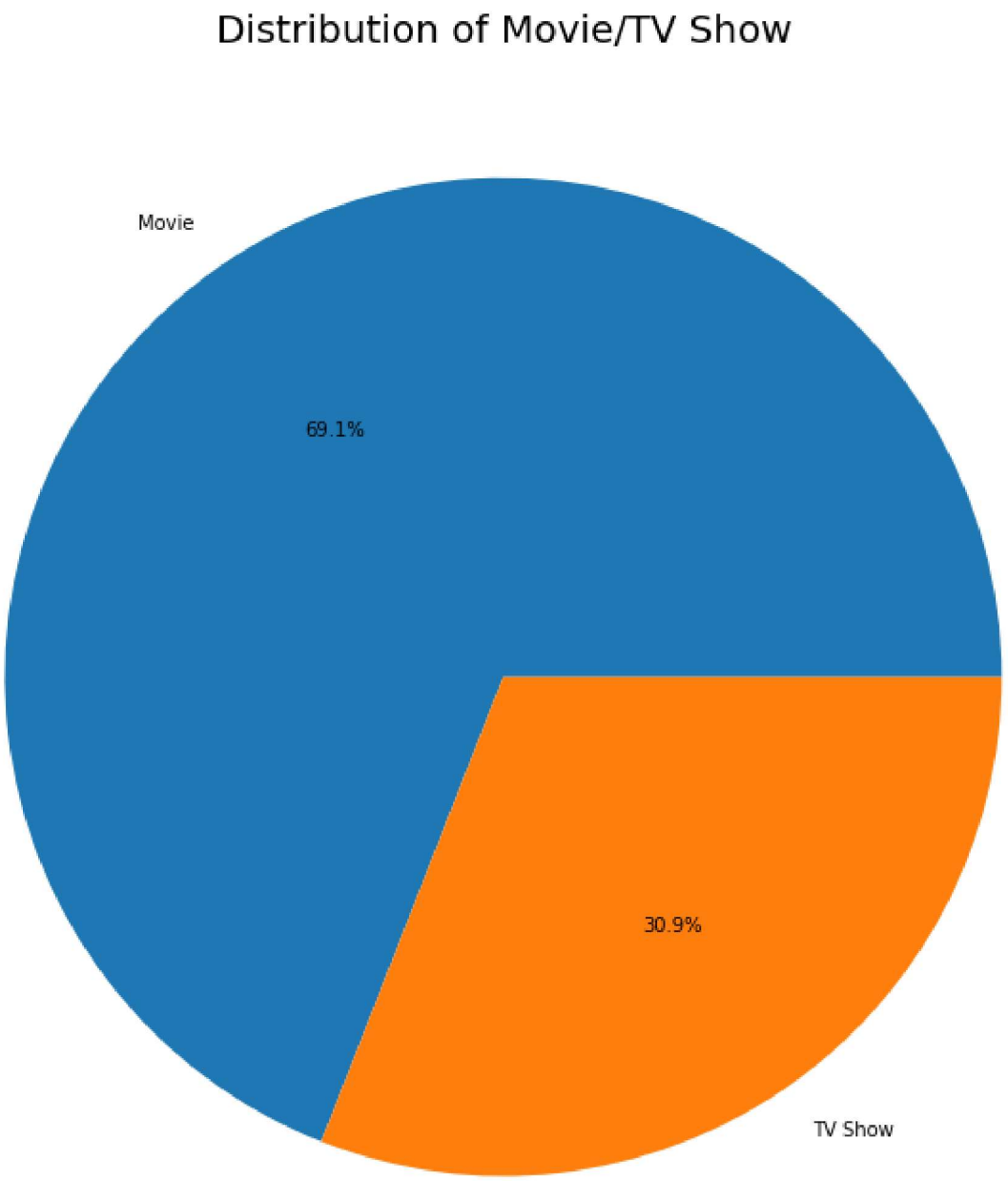
III. Exploratory Data Analysis

General Question: **What are the common topics in Netflix based on their descriptions**

- What are the common topics per genre and type(Movies/TV shows)?
- what are the top keywords used in Movies and TV shows ?
- Are there similarities with the Movie/Show's genre and its description?

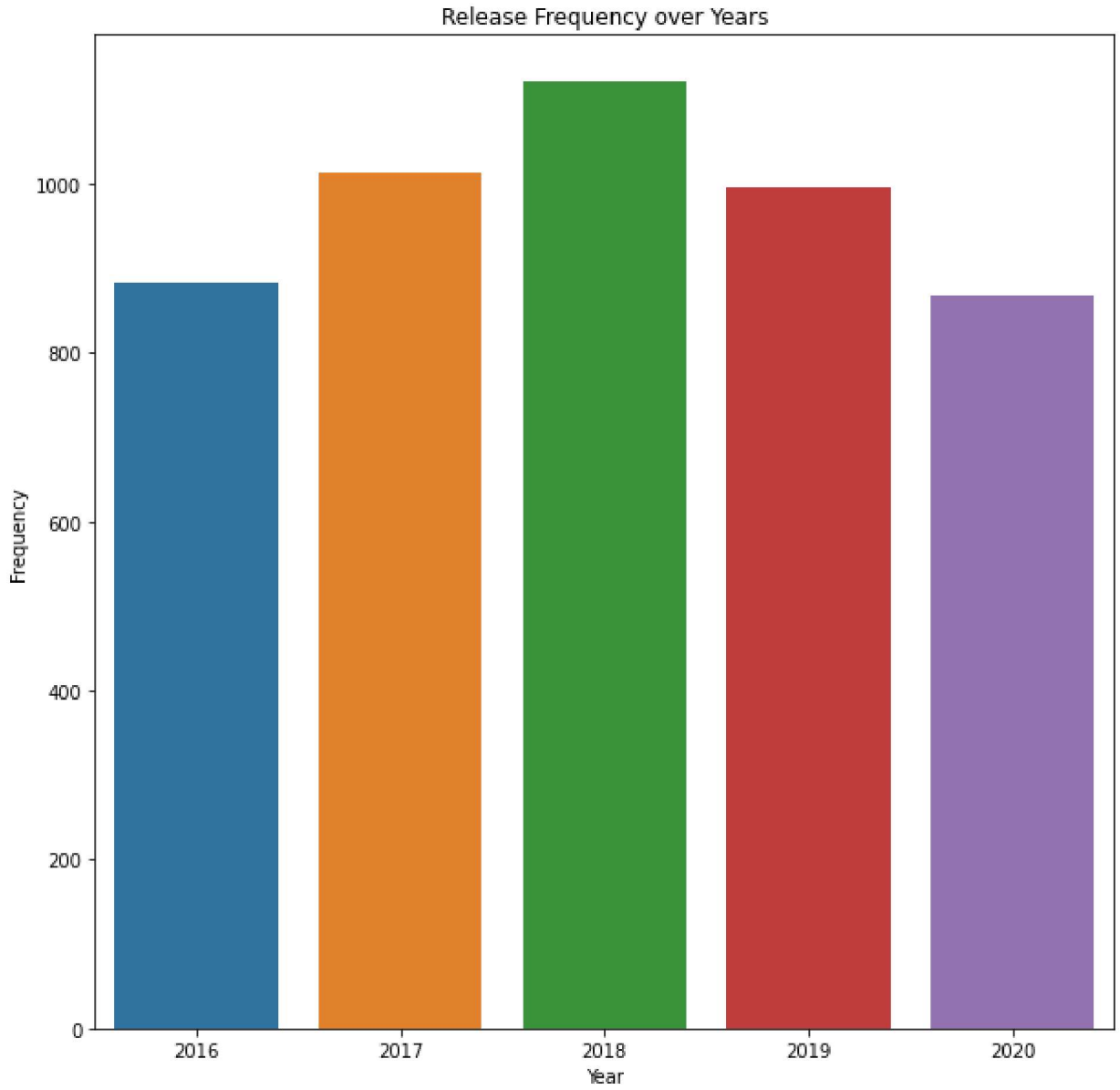
In [16]:
```
# Distribution of Movies/Shows
bar, ax = plt.subplots(figsize = (12,12))
plt.pie(df['type'].value_counts(), labels = df['type'].value_counts().index, autopct="%.1f%%")
plt.title('Distribution of Movie/TV Show', size=20)
```

Out[16]:
```
Text(0.5, 1.0, 'Distribution of Movie/TV Show')
```


Distribution of Movie/TV Show
Movie
69.1%
30.9%
TV Show

In [17]:
```
#changes size of figure
bar, ax = plt.subplots(figsize = (10,10))
#plotting a barplot using seaborn. with the x value ot release year and y value of total produced
sns.barplot(x = df['release_year'].value_counts().index[:5], y = df['release_year'].value_counts()[:5])
plt.xlabel('Year')
plt.ylabel('Frequency')
plt.title('Release Frequency over Years')
```

Out[17]:
```
Text(0.5, 1.0, 'Release Frequency over Years')
```

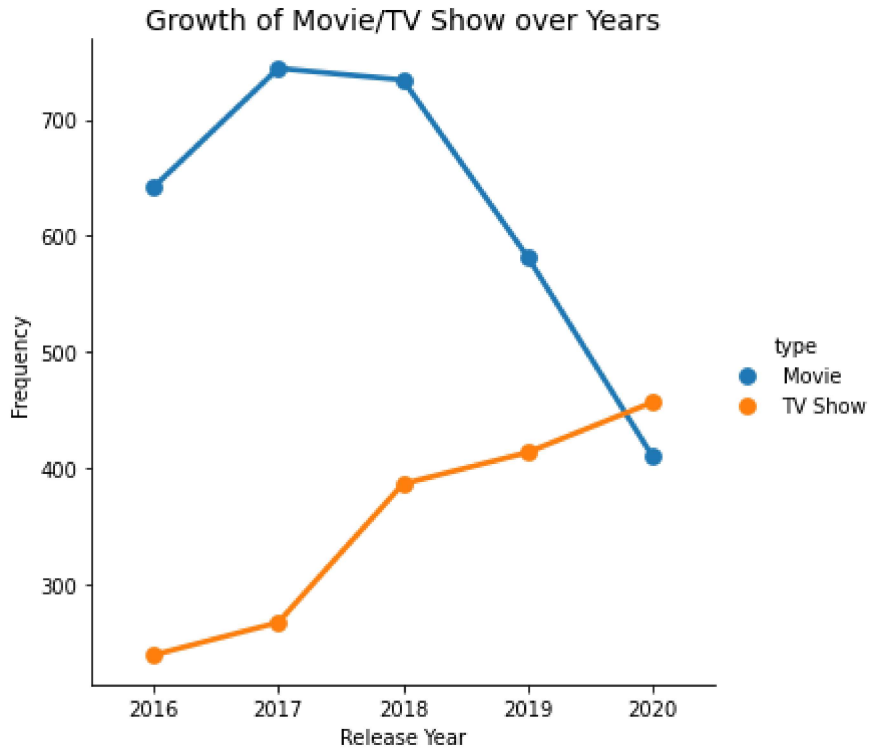


The bar plot above indicates that the year of the mose production of movies and TV shows in netflix was in 2018

In [18]:
```
#Growth of Movie/Show over the years
movie_data = df[df['type'] == 'Movie']
tv_show_data = df[df['type'] == 'TV Show']
temp = df[['type', 'release_year']]
temp = temp.value_counts().to_frame()
temp.reset_index(level=[0,1], inplace=True)
temp = temp.rename(columns = {0:'count'})
temp = pd.concat([temp[temp['type'] == 'Movie'][:5], temp[temp['type']== 'TV Show'][:5]])
```

In [19]:
```
sns.catplot(x = 'release_year', y = 'count', hue = 'type', data = temp, kind = 'point')
plt.xlabel('Release Year')
plt.ylabel('Frequency')
plt.title('Growth of Movie/TV Show over Years', size=14)
```

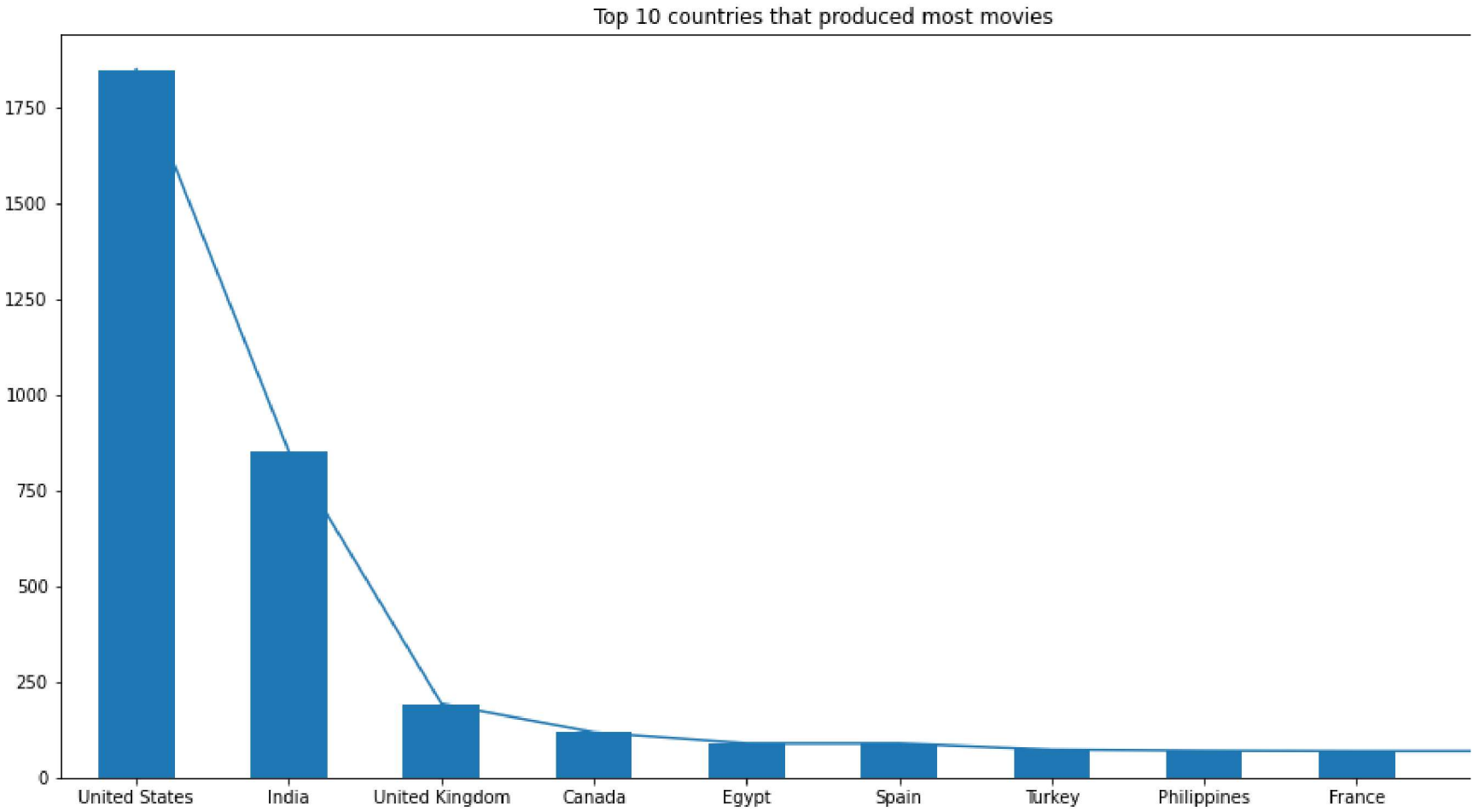
Out[19]:
```
Text(0.5, 1.0, 'Growth of Movie/TV Show over Years')
```



In the plot above it can be seen how there is a decrease of production of Movies from 2018 onwards and increase in the production of TV sh

In [20]:
```
#Top 10 countries that produced most movies in Netflix

histo1= netflix_movies.country.value_counts().head(10)
#barplot
histo1.plot(kind='bar')
plt.title('Top 10 countries that produced most movies')
histo1.plot(figsize=(16,8))
```

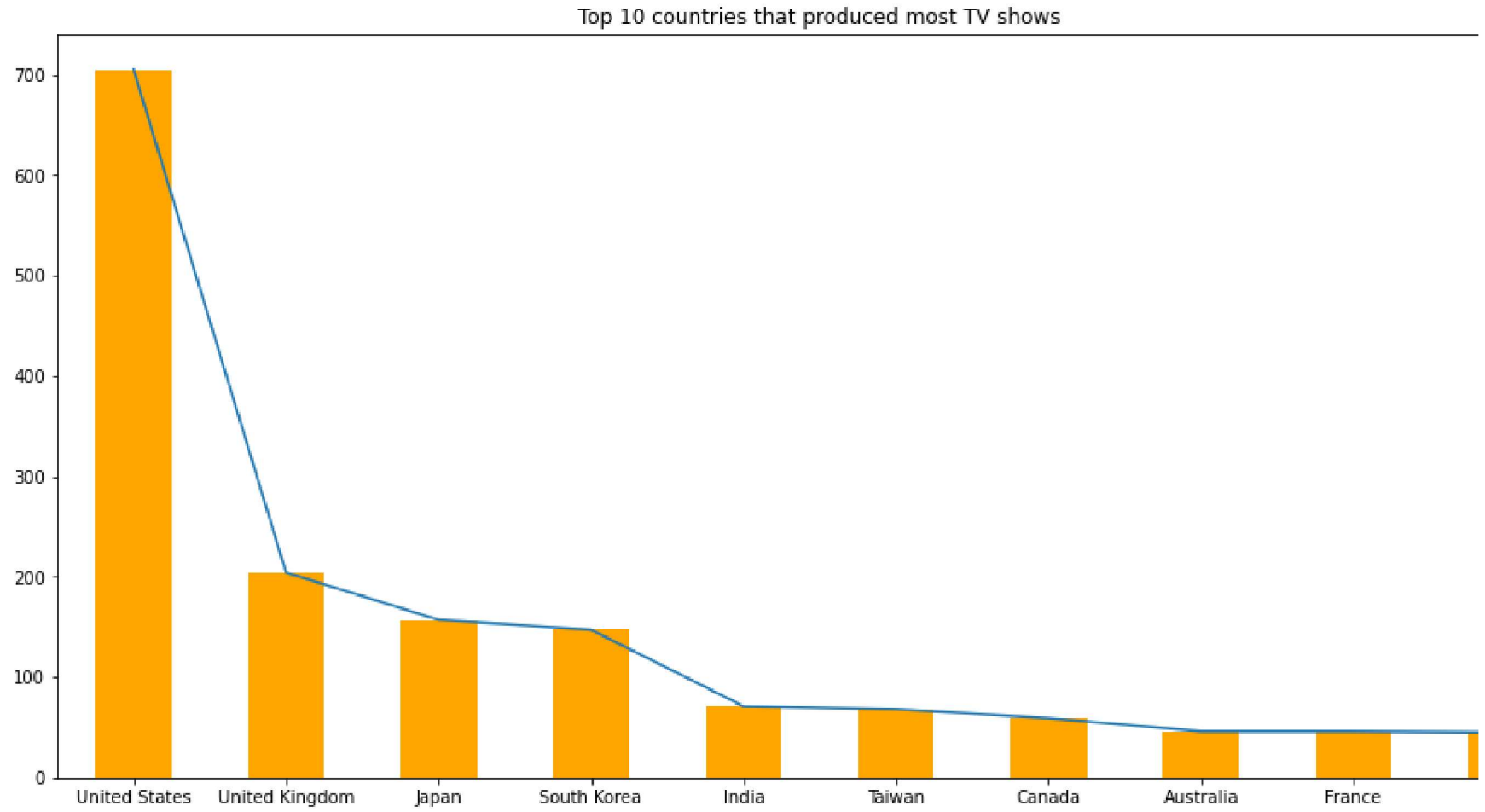
Out[20]: <AxesSubplot:title={'center':'Top 10 countries that produced most movies'}>



In the barplot above it can be seen that the US followed by Indi tops the most movies produced in Netflix. Philippines is in the 8th place whic

In [21]:
```
#Top 10 countries that produced most TV shows in Netflix

histo2=netflix_shows.country.value_counts().head(10)
histo2.plot(kind='bar', color='orange')
plt.title('Top 10 countries that produced most TV shows')
histo2.plot(figsize=(16,8))
```

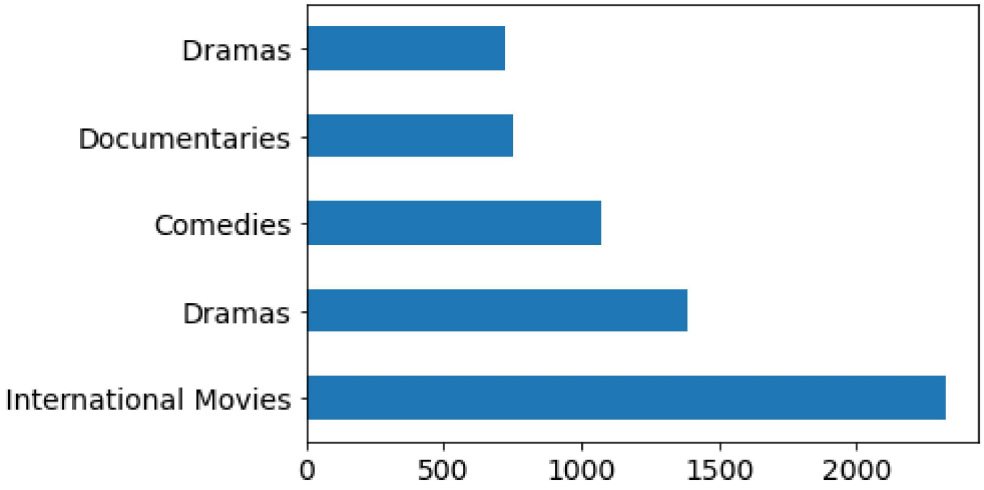
Out[21]: <AxesSubplot:title={'center':'Top 10 countries that produced most TV shows'}>



as seen in this chart United states tops the production of TV shows followed by UK and Japan

In [22]:
```
# top genres in Netflix

#filling null values
df['genre']=df['genre'].fillna('Not Specified')
#Splitting multiple genres.
FGenre=pd.DataFrame()
FGenre=df['genre'].str.split(',', expand=True).stack()
FGenre=FGenre.to_frame()
FGenre.columns=['Genre']
#grouping different genres and getting total content
genres=FGenre.groupby(['Genre']).size().reset_index(name='Total Content')
genres=genres[genres.Genre !='Not Specified']
genres=genres.sort_values(by=['Total Content'],ascending=False)
genres=genres.head()
#plotting bar chart
genres=FGenre.Genre.value_counts().head(5)
genres.plot(kind="barh", fontsize=14)
```

Out[22]: <AxesSubplot:>



Recommendations

Recommends movie based on description and grabs the similarity as its results

In [23]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

class ContentAnalysis():
    def __init__(self, data_frame, threshold = 0.1, stop_words = 'english', lowercase = True, use_idf = True, nor
        self.data_frame = data_frame
        self.model = TfidfVectorizer(max_df=threshold,stop_words=stop_words, lowercase=lowercase, use_idf=use_idf
        self.vector = False

    def generate_vector(self, data):
        self.vector = self.model.fit_transform(data)


    def find_movies(self, request, top = 10):
        if self.vector is not False:
            content_transformation = self.model.transform([request])
            movie_relatively = np.array(np.dot(content_transformation,np.transpose(self.vector)).toarray()[0])
            index = np.argsort(movie_relatively)[-top:][::-1]
            rate = [movie_relatively[i] for i in index]
            result = zip(index, rate)
            self.render_result(request, result)


    def recommend_movie(self, request_index , top = 15):
        if self.vector is not False:
            cosine_similarity = linear_kernel(self.vector[request_index:request_index+1], self.vector).flatten()
            index = cosine_similarity.argsort()[-top-1:-1][::-1]
            rate = [cosine_similarity[i] for i in index]
            result = zip(index, rate)
            self.render_result(str(self.data_frame[request_index:request_index+1]), result)

    def render_result(self, request_content,indices):
        print('Your request : ' + request_content)
        print('----------------------------------')
        print('Best Results :')
        data = self.data_frame
        for index, rate in indices:
            print('Confidence: {:.2f}%, {}'.format(rate*100, data['title'].loc[index] ))
```

In [24]:
```
vector = ContentAnalysis(df)
vector.generate_vector(df["title"])
vector.find_movies('Happy Birthday')
```

```
Your request : Happy Birthday
--------------------------------
Best Results :
Confidence: 62.52%, My Birthday Song
Confidence: 59.06%, Happy And
Confidence: 59.06%, Almost Happy
Confidence: 59.06%, Happy!
Confidence: 42.85%, My Happy Family
Confidence: 37.23%, Merry Happy Whatever
Confidence: 37.23%, Happy Go Lucky
Confidence: 37.23%, Happy Hunting
Confidence: 37.23%, Happy Times
Confidence: 36.64%, Happy Valley
```

In [25]:
```
#Recommends movie based on description and grabs the similarity as its results
vector = ContentAnalysis(df)
vector.generate_vector(df["description"])
vector.recommend_movie(100)
```

```
Your request :       type     title         director country      date_added  release_year  \
100  Movie  3 Idiots  Rajkumar Hirani   India  August 1, 2019          2009

    rating duration                                    genre  \
100  PG-13  164 min  Comedies, Dramas, International Movies

                                          description  release_year_z
100  While attending one of India's premier college...       -0.563284
---------------------------------
Best Results :
Confidence: 23.19%, College Romance
Confidence: 17.27%, Engineering Girls
Confidence: 15.25%, Candy Jar
Confidence: 15.13%, Mr. Young
Confidence: 14.78%, 100 Things to do Before High School
Confidence: 14.70%, Pahuna
Confidence: 14.66%, Best Neighbors
Confidence: 13.64%, Be with Me
Confidence: 13.47%, Moms at War
Confidence: 12.88%, Lovesong
Confidence: 12.67%, Limitless
Confidence: 12.65%, The Prince & Me
Confidence: 11.99%, Singles Villa
Confidence: 11.32%, Barrio Universitario
Confidence: 10.70%, LEGO Friends: The Power of Friendship
```

IV. Modeling

TOPIC MODELING - This creates topics/groups based on descriptions.

We used the Latent Dirichlet Allocation(LDA) that will look into the descriptions of each movies and hows in the dataset.

training LDA model that is based on the title's descriptions and used to determine the topic of unseen description

Libraries used: gensim, nltk

In [26]:
```
ps = SnowballStemmer('english')
lemmatizer = WordNetLemmatizer()

tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV


#Words from corpus - dictionary
words = set(nltk.corpus.words.words())


#Text cleaning - tokenization, remove special characters, punctions, meaningless words etc.
def txt_clean(txt):
    tokens = nltk.word_tokenize(txt.lower())
    tokens_clean = [w for w in tokens if w.isalpha() and w in words]
    return tokens_clean

#create unigram/bigram/trigram words, remove stopwords, lemmatize
def lemmatize_stem(tokens, ngram_type=None):

    bigram = gensim.models.Phrases(tokens, min_count=2, threshold=100)
    bigram_tokens = [bigram[tokens[w]] for w in range(len(tokens))]
    trigram = gensim.models.Phrases(bigram[tokens],threshold=100)
    trigram_tokens = [trigram[tokens[w]] for w in range(len(tokens))]

    tokens_clean = []
    if ngram_type == "bigram":
        tokens_c = bigram_tokens
    elif ngram_type == "trigram":
        tokens_c = trigram_tokens
    else:
        tokens_c = tokens

    for i in range(len(tokens)-1):
        txt = tokens_c[i]
        txt_above5 = [k for k in txt if len(k)>=5 and k not in gensim.parsing.preprocessing.STOPWORDS]
        lemma_txt = [lemmatizer.lemmatize(w,pos=tag_map[tg[0]]) for w,tg in nltk.pos_tag(txt_above5)]
        stem_txt = [w for w in lemma_txt]
        tokens_clean.append(stem_txt)
```

```
    dictionary = corpora.Dictionary(tokens_clean)
    corpus = [dictionary.doc2bow(text) for text in tokens_clean]

    return dictionary, corpus, tokens_clean
#splitting data sets for unseen data
unseen_len = int(round(0.10 * len(df),0))
unseen_data = df["description"].sample(unseen_len) #random sample
txt_data = df["description"].drop(unseen_data.index)

#Tokenize and clean
tokens = list(txt_data.apply(lambda x: txt_clean(x)))

#Chose bigram - it had the best performace (tried both unigram and trigram)
dictionary, corpus, tokens_clean = lemmatize_stem(tokens, "bigram")

from gensim.models import CoherenceModel

#(The number of topics and decay shouldn't be too high or too low)
topic_num = 40
min_probability = 0.05
learning_decay = 0.5


ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics = topic_num, id2word=dictionary, passes=15, minimum

print("\nSample of Topics:")
for i,j in ldamodel.show_topics(formatted=True,num_words= 10):
    print("Topic-{} => {}".format(i,j))
topics = ldamodel.print_topics(num_words=10)
```

```
Sample of Topics:
Topic-19 => 0.110*"death" + 0.083*"force" + 0.047*"government" + 0.030*"corruption" + 0.024*"capture" + 0.023*"yo
r" + 0.016*"uncle"
Topic-37 => 0.081*"dangerous" + 0.061*"history" + 0.046*"director" + 0.041*"notorious" + 0.036*"fortune" + 0.035*
0.020*"forever"
Topic-6 => 0.099*"couple" + 0.079*"relationship" + 0.052*"teenage" + 0.046*"romantic" + 0.035*"young" + 0.033*"fa
rience" + 0.025*"supernatural"
Topic-36 => 0.054*"discover" + 0.048*"single" + 0.048*"wealthy" + 0.043*"family" + 0.039*"break" + 0.034*"company
*"lose"
```

```
Topic-24 => 0.070*"new_york" + 0.056*"street" + 0.051*"social" + 0.041*"raise" + 0.033*"widow" + 0.031*"learn" +
*"woman"
Topic-20 => 0.078*"base" + 0.065*"story" + 0.065*"drama" + 0.046*"sister" + 0.043*"friendship" + 0.036*"true_stor
*"grow"
Topic-13 => 0.104*"leave" + 0.084*"village" + 0.042*"teen" + 0.037*"strange" + 0.030*"small" + 0.021*"rise" + 0.0
ution"
Topic-15 => 0.094*"crime" + 0.061*"business" + 0.060*"bring" + 0.051*"inspire" + 0.034*"beloved" + 0.032*"series"
22*"ancient"
Topic-38 => 0.072*"people" + 0.055*"miss" + 0.044*"investigate" + 0.034*"survive" + 0.027*"sinister" + 0.024*"you
ion" + 0.018*"settle"
Topic-27 => 0.078*"police" + 0.059*"take" + 0.055*"rescue" + 0.050*"deadly" + 0.035*"turn" + 0.023*"embark" + 0.0
```

In [27]:
```
#Description-topic distributions for our training set  - It lists top 4 keywords and Dominant topic for each sent
arr = []
for i, j in enumerate(ldamodel[corpus]):
    if len(j) > 0:
        max_val = sorted([w[1] for w in j],reverse=True)[0]
        max_topic = [w[0] for w in j if w[1]==max_val][0]
        keywords = ldamodel.show_topic(max_topic,topn=4)
        keywords = [k[0] for k in keywords]
        description = txt_data.iloc[i]
        arr.append([description,  ",".join(keywords), max_topic, round(max_val,2),])

lda_distribution = pd.DataFrame(arr, columns=['Description', 'Top Keywords', 'Dominant Topic', 'Probability'])
lda_distribution.head()
```

Out[27]:

	Description	Top Keywords	Dominant Topic	Probability
0	After a devastating earthquake hits Mexico Cit...	future,magical,try,scientist	33	0.29
1	In a postapocalyptic world, rag-doll robots hi...	group,fight,search,power	12	0.23
2	A brilliant group of students become card-coun...	high_school,legendary,navigate,documentary	10	0.57
3	A genetics professor experiments with a treatm...	short,classic,professor,creative	16	0.60
4	After an awful accident, a couple admitted to ...	accident,prove,action,medical	8	0.40

In [28]:
```
#predicting topics for unseen data
unseen_clean = unseen_data.apply(lambda x: txt_clean(x))

arr = []
for i in unseen_clean:
    lemma_txt = [lemmatizer.lemmatize(w,pos=tag_map[tg[0]]) for w,tg in nltk.pos_tag(i)]
    lemma_txt2 = [w for w in lemma_txt if w not in gensim.parsing.preprocessing.STOPWORDS]
    arr2 = ldamodel[dictionary.doc2bow(lemma_txt2)]
    max_arr2 = sorted([x[1] for x in arr2],reverse=True)[:3]
    sel_arr2 = [list(x) for x in arr2 if x[1] in max_arr2][:3]
    sel_arr2 = sum(sel_arr2,[])
    if len(sel_arr2) != 6:
        sel_arr2.extend(["None"]*(6-len(sel_arr2)))

    sel_arr2.extend([i])
    arr.append(sel_arr2)

unseen_df = pd.DataFrame(arr, columns = ["topic_1", "topic_1_prob", "topic_2", "topic_2_prob", "topic_3", "topic_
unseen_df["Description"] = list(unseen_data)
cols = list(unseen_df.columns)
cols = cols[-1:] + [cols[-2]] + cols[:-2]
unseen_df = unseen_df[cols]
unseen_df.head()
```

Out[28]:

	Description	Tokens	topic_1	topic_1_prob	topic_2	topic_2_prob	top
0	A penniless actor new to Mumbai and a beautifu...	[a, penniless, actor, new, to, and, a, beautif...	0	0.12	1	0.182802	
1	On his wedding day, an arrogant, greedy accoun...	[on, his, wedding, day, an, arrogant, greedy, ...	4	0.17	14	0.323722	
2	Convicted of rape and murder at age 18, Daniel...	[of, rape, and, murder, at, age, holden, nearl...	5	0.13	19	0.128298	
3	This period drama set in impoverished East Lon...	[this, period, drama, set, in, east, in, the, ...	18	0.15	20	0.146461	
4	After each of them loses a child to murder, tw...	[after, each, of, them, a, child, to, murder, ...	12	0.22	23	0.33612	

clustering data set with the model results

In [29]:
```
#Clustering each movie/show per genre

#adding genre column in lds_distribution table
genres = pd.DataFrame(df['genre'])
Ntype =pd.DataFrame(df['type'])
LDA = pd.DataFrame(lda_distribution)
LDA = LDA.join(genres)
LDA = LDA.join(Ntype)
FGenre = pd.DataFrame(LDA['genre'])
FGenre=LDA['genre'].str.split(',', expand=True)
FGenre=FGenre[0]
FGenre=FGenre.to_frame()
FGenre.columns=['genre']

LDA.insert(0,"Genre", FGenre)
LDA = LDA.drop(['genre'], axis=1)

LDA = LDA.rename(columns={"Dominant Topic" : "Dominant_Topic"})
LDA = LDA.rename(columns={"Top Keywords" : "Top_Keywords"})
LDA
```

Out[29]:

	Genre	Description	Top_Keywords	Dominant_Topic	Proba
0	International TV Shows	After a devastating earthquake hits Mexico Cit...	future,magical,try,scientist	33	
1	Dramas	In a postapocalyptic world, rag-doll robots hi...	group,fight,search,power	12	
2	Horror Movies	A brilliant group of students become card-coun...	high_school,legendary,navigate,documentary	10	
3	Action & Adventure	A genetics professor experiments with a treatm...	short,classic,professor,creative	16	
4	Dramas	After an awful accident, a couple admitted to ...	accident,prove,action,medical	8	
...	...	...	...	...	
6997	Documentaries	Dragged from civilian life, a former superhero...	accident,prove,action,medical	8	
6998	Kids' TV	When Lebanon's Civil War deprives Zozo of his ...	leave,village,teen,strange	13	
6999	Kids' TV	A scrappy but poor boy worms his way into a ty...	mother,family,daughter,young	29	
7000	Children & Family Movies	In this documentary, South African rapper Nast...	night,battle,murder,documentary	5	
7001	British TV Shows	Dessert wizard Adriano Zumbo looks for the nex...	each_other,come,remote,money	26	

7002 rows × 6 columns

In [30]:
```
#Now we will cluster the results from the model by genre and type
#sort by the genres: Dramas, Documentaries, Comedy, International movies

Drama=LDA[LDA['Genre'].str.contains('Dramas')]
Docu=LDA[LDA['Genre'].str.contains('Documentaries')]
Comedy=LDA[LDA['Genre'].str.contains('Comedy')]
IntMovies = LDA[LDA['Genre'].str.contains('International Movies')]

# sort by type (movie/TV show)
movies=LDA[LDA['type'].str.contains('Movie')]
shows=LDA[LDA['type'].str.contains('TV Show')]
```

In [31]:
```
TopTopicsDF = pd.DataFrame(columns=['Genre', 'Top_Topics'])
TopTopicsTypes = pd.DataFrame(columns=['Type', 'Top_Topics'])
```

In [32]:
```
#Sorting top key words for movies

#Splitting multiple keywords
TopKeyWordsM=pd.DataFrame()
TopKeyWordsM=movies['Top_Keywords'].str.split(',', expand=True).stack()
TopKeyWordsM=TopKeyWordsM.to_frame()
TopKeyWordsM.columns=['Top_Keywords']
#grouping different genres and getting total content
KWMovies=TopKeyWordsM.groupby(['Top_Keywords']).size().reset_index(name='Total Content')
KWMovies=KWMovies.sort_values(by=['Top_Keywords'],ascending=False)
KWMovies=KWMovies.head()
#plotting bar chart of top keywords for movies
KWMovies=TopKeyWordsM.Top_Keywords.value_counts().head(5)
```

In [33]:
```
#Sorting top key words for Netflix
#Splitting multiple keywords
TopKeyWordsM=pd.DataFrame()
TopKeyWordsM=LDA['Top_Keywords'].str.split(',', expand=True).stack()
TopKeyWordsM=TopKeyWordsM.to_frame()
TopKeyWordsM.columns=['Top_Keywords']
#grouping different genres and getting total content
KW=TopKeyWordsM.groupby(['Top_Keywords']).size().reset_index(name='Total Content')
KW=KW.sort_values(by=['Top_Keywords'],ascending=False)
KW=KW.head()
#plotting bar chart of top keywords for movies
KW=TopKeyWordsM.Top_Keywords.value_counts().head(5)
```

In [34]:
```
#Sorting top key words for Shows
#Splitting multiple keywords
TopKeyWordsM=pd.DataFrame()
TopKeyWordsM=shows['Top_Keywords'].str.split(',', expand=True).stack()
TopKeyWordsM=TopKeyWordsM.to_frame()
TopKeyWordsM.columns=['Top_Keywords']
#grouping different genres and getting total content
KWShows=TopKeyWordsM.groupby(['Top_Keywords']).size().reset_index(name='Total Content')
KWShows=KWShows.sort_values(by=['Top_Keywords'],ascending=False)
KWShows=KWShows.head()
#plotting bar chart of top keywords for movies
KWShows=TopKeyWordsM.Top_Keywords.value_counts().head(5)
```

In [35]:
```
#Gets the top dominant topics per type(Movies/Shows)
DTopicsType=pd.DataFrame()
DTopics=movies['Dominant_Topic']
DTopics=DTopics.to_frame()
DTopics.columns=['Dominant_Topic']

#gets the total content of Dominant Topics
TTopics=DTopics.groupby(['Dominant_Topic']).size().reset_index(name='Total Content')
TTopics=TTopics.sort_values(by=['Dominant_Topic'],ascending=False)
TTopics=TTopics.head()


#displays the top 3 dominant topics for movies
TTopics=DTopics.Dominant_Topic.value_counts().head(3)

#manually inputted the topics into a dictionary
Data={'Type':['Movies'],
       'Top_Topics':['9', '14', '18']}
#appended to the dataframe
TopTopicsTypes = TopTopicsTypes.append(Data, ignore_index=True)

#same process but for TV shows
DTopicsType=pd.DataFrame()
DTopics=shows['Dominant_Topic']
DTopics=DTopics.to_frame()
DTopics.columns=['Dominant_Topic']
TTopics=DTopics.groupby(['Dominant_Topic']).size().reset_index(name='Total Content')
TTopics=TTopics.sort_values(by=['Dominant_Topic'],ascending=False)
TTopics=TTopics.head()
TTopics=DTopics.Dominant_Topic.value_counts().head(3)
Data={'Type':['TV Shows'],
       'Top_Topics':['3', '18', '14']}
TopTopicsTypes = TopTopicsTypes.append(Data, ignore_index=True)
TopTopicsTypes
```

Out[35]:

	Type	Top_Topics
0	[Movies]	[9, 14, 18]
1	[TV Shows]	[3, 18, 14]

In [36]:
```
#same process as the top dominant topic for type but this time for genres
DTopics=pd.DataFrame()
DTopics=Drama['Dominant_Topic']
DTopics=DTopics.to_frame()
DTopics.columns=['Dominant_Topic']

TTopics=DTopics.groupby(['Dominant_Topic']).size().reset_index(name='Total Content')
TTopics=TTopics.sort_values(by=['Dominant_Topic'],ascending=False)
TTopics=TTopics.head()
TTopics=DTopics.Dominant_Topic.value_counts().head(3)
TTopics

DramaData={'Genre':['Drama'],
           'Top_Topics':['9', '18', '19']}

TopTopicsDF = TopTopicsDF.append(DramaData, ignore_index=True)
TopTopicsDF
```

Out[36]:

	Genre	Top_Topics
0	[Drama]	[9, 18, 19]

In [37]:
```
DTopics=pd.DataFrame()
DTopics=Comedy['Dominant_Topic']
DTopics=DTopics.to_frame()
DTopics.columns=['Dominant_Topic']
#joining two data frames
TTopics=DTopics.groupby(['Dominant_Topic']).size().reset_index(name='Total Content')
TTopics=TTopics.sort_values(by=['Dominant_Topic'],ascending=False)
TTopics=TTopics.head()
#plotting bar chart
TTopics=DTopics.Dominant_Topic.value_counts().head(3)
TTopics

DramaData={'Genre':['Comedy'],
           'Top_Topics':['19', '3', '18']}
TopTopicsDF = TopTopicsDF.append(DramaData, ignore_index=True)
TopTopicsDF
```

Out[37]:

	Genre	Top_Topics
0	[Drama]	[9, 18, 19]
1	[Comedy]	[19, 3, 18]

In [38]:
```
DTopics=pd.DataFrame()
DTopics=Docu['Dominant_Topic']
DTopics=DTopics.to_frame()
DTopics.columns=['Dominant_Topic']
#joining two data frames
TTopics=DTopics.groupby(['Dominant_Topic']).size().reset_index(name='Total Content')
TTopics=TTopics.sort_values(by=['Dominant_Topic'],ascending=False)
TTopics=TTopics.head()
#plotting bar chart
TTopics=DTopics.Dominant_Topic.value_counts().head(3)
TTopics
DramaData={'Genre':['Documentaries'],
           'Top_Topics':['14', '5', '18']}
TopTopicsDF = TopTopicsDF.append(DramaData, ignore_index=True)
TopTopicsDF
```

Out[38]:

	Genre	Top_Topics
0	[Drama]	[9, 18, 19]
1	[Comedy]	[19, 3, 18]
2	[Documentaries]	[14, 5, 18]

In [39]:
```
DTopics=pd.DataFrame()
DTopics=IntMovies['Dominant_Topic']
DTopics=DTopics.to_frame()
DTopics.columns=['Dominant_Topic']
#joining two data frames
TTopics=DTopics.groupby(['Dominant_Topic']).size().reset_index(name='Total Content')
TTopics=TTopics.sort_values(by=['Dominant_Topic'],ascending=False)
TTopics=TTopics.head()
#plotting bar chart
TTopics=DTopics.Dominant_Topic.value_counts().head(3)
TTopics

DramaData={'Genre':['International Movies'],
           'Top_Topics':['27', '5', '9']}
TopTopicsDF = TopTopicsDF.append(DramaData, ignore_index=True)
TopTopicsDF
```

Out[39]:

	Genre	Top_Topics
0	[Drama]	[9, 18, 19]
1	[Comedy]	[19, 3, 18]
2	[Documentaries]	[14, 5, 18]
3	[International Movies]	[27, 5, 9]

##Model Evaluation

In [40]:
```
#Perplexity - how probable where some of the new unseen data that were given to the model that was learned earlie
print('Perplexity: ', ldamodel.log_perplexity(corpus))
```

```
Perplexity:  -13.34869066240852
```

In [41]:
```
#Coherence - measure the degree of semantic similarity between high scoring words in each topic (and then average
coherence_model_lda = CoherenceModel(model=ldamodel, texts = tokens_clean, dictionary=dictionary, coherence='c_v'
coherence_lda = coherence_model_lda.get_coherence()
print('Coherence Score: ', coherence_lda)
```

```
Coherence Score:  0.5730231737638842
```

V. Evaluation (Interpreting Results)

Now that we have used the Latent Dirichlet allocation model to look into the Netflix data sets we will use it results to answer the question: Wh descriptions? We will look into the common topics per genre, type.

The model's purpose is to create topics/groups based on descriptions. Given a new movie, can we predict the topic probabilities? This will be etc. First we will look into the common topics per genre. We will only look into the top genres in Netflix: Dramas, Documentaries, Comedy, ar

As seen from each description formed different kinds of topics with different probabilities, which shows that each topic had their own views.

Topic number we got is around 40 with 5k rows (train set) represented with 40 topics along with different probabilities. The "unseen" descript each of these topics are not unique so we can't define the topics.

Since precision and recall necessarily depend on the notion of true classes for the data, it can't be applied because LDA model is an unsupe unseen data predictions the probability of the distributions of topics are low.

In [42]:
```
# In here we will see the top 3 topics per genre, but they are still in number and will be translated into the wo
TopTopicsDF
```

Out[42]:

	Genre	Top_Topics
0	[Drama]	[9, 18, 19]
1	[Comedy]	[19, 3, 18]
2	[Documentaries]	[14, 5, 18]
3	[International Movies]	[27, 5, 9]

In [43]:
```
# top topics per Type(Movies/TVshows)
TopTopicsTypes
```

Out[43]:

	Type	Top_Topics
0	[Movies]	[9, 14, 18]
1	[TV Shows]	[3, 18, 14]

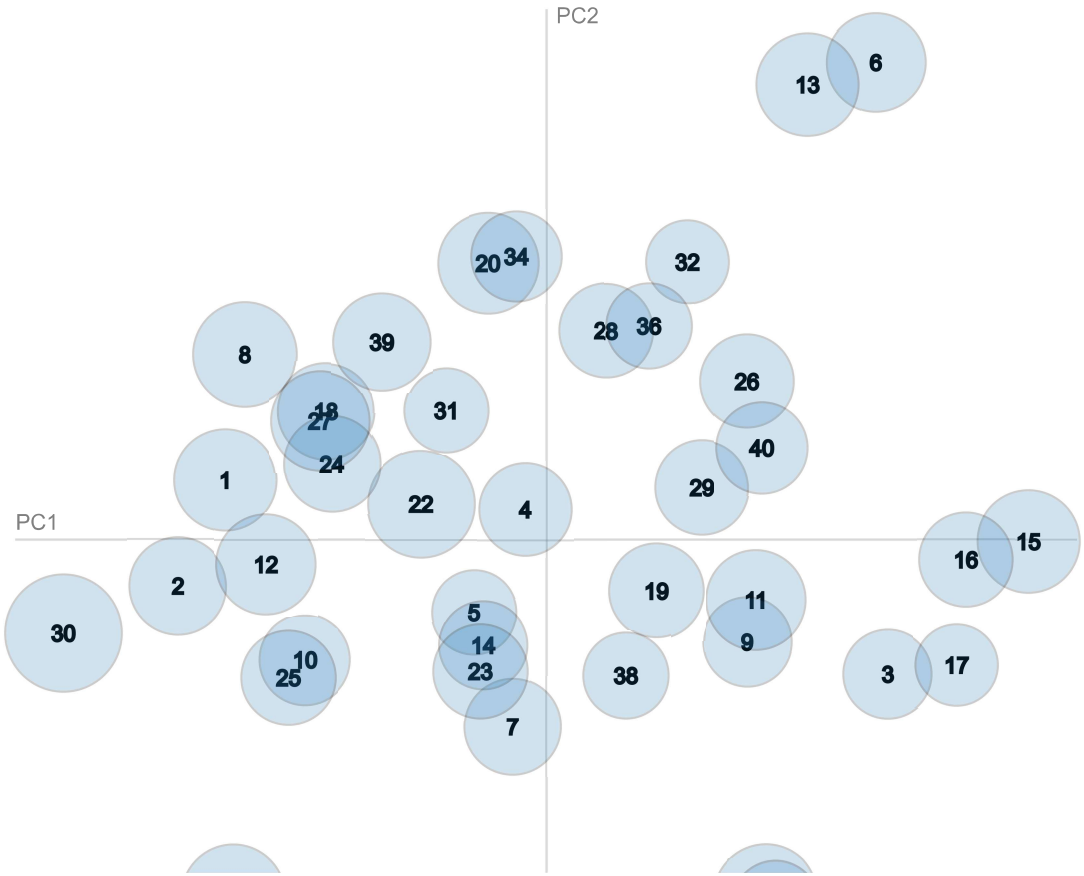
In [44]:
```
#To easily read the top topics per genre, you just have to input the number of the topic in the texinput of the c
#and it will display the top words used in each topic for easier reading.
import pyLDAvis.gensim
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary, sort_topics=False)
vis

#Here we are using the PyLDAvis visual tool. In the left side it depicts the global view of the model, on how pre
#the bars represent the terms that are most useful in interpreting the topic currently selected
```

Out[44]:

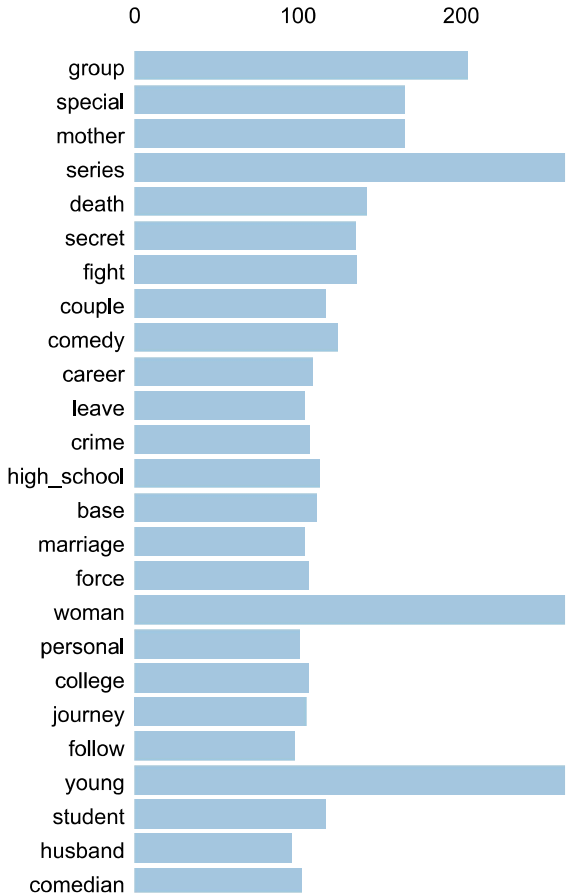
Selected Topic: 0 | Previous Topic | Next Topic | Clear Topic

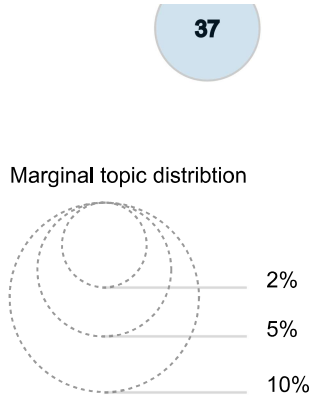
Slide to adjust relevance metric:(2
λ = 1

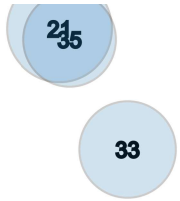
Intertopic Distance Map (via multidimensional scaling)



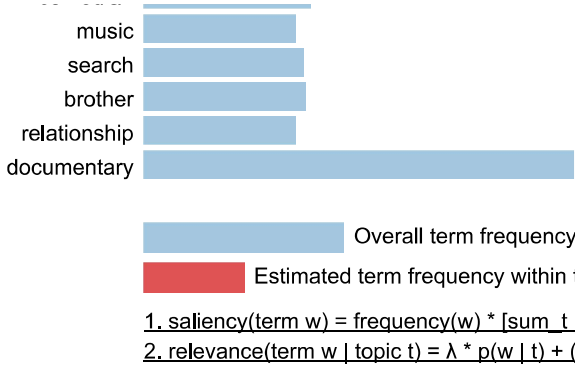
Top-30




37
Marginal topic distribtion
2%
5%
10%


21
35

33

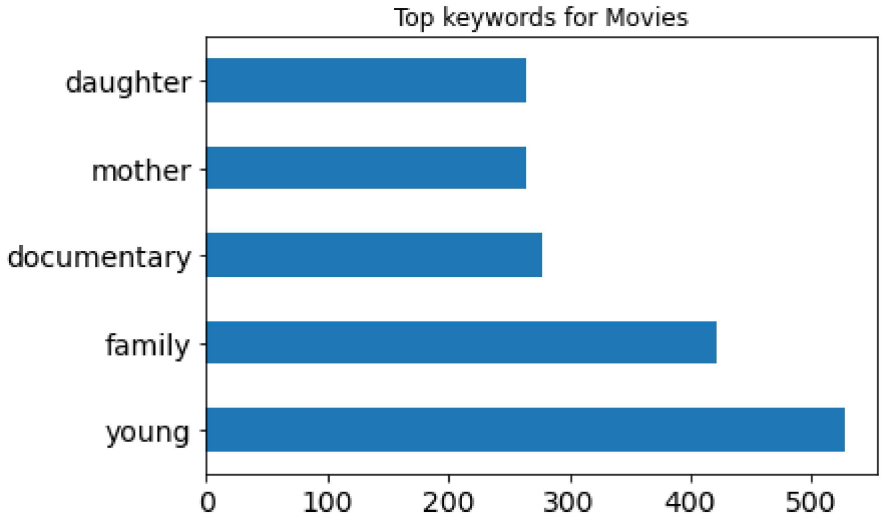

music
search
brother
relationship
documentary
Overall term frequency
Estimated term frequency within
1. saliency(term w) = frequency(w) * [sum_t
2. relevance(term w | topic t) = λ * p(w | t) + (

In [ ]:

In [45]:
```
#Top Keywords per type (Movies and TV Shows)
#This will show the top keywords from their description and will tell what are the common themes for movies, show

KWMovies.plot(kind="barh", fontsize=14, title="Top keywords for Movies")
```

```
c:\python39\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `tr
esult to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tupl
  and should_run_async(code)
```

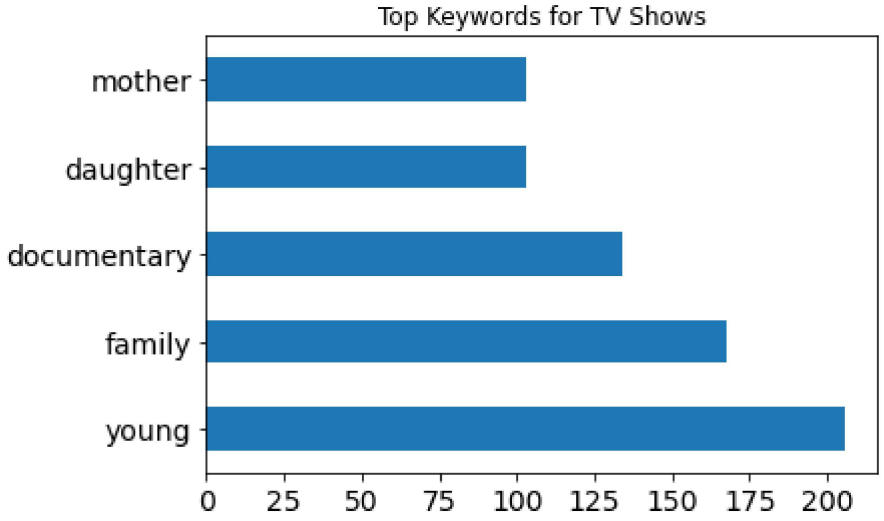
Out[45]:
```
<AxesSubplot:title={'center':'Top keywords for Movies'}>
```



In [46]:
```
KWShows.plot(kind="barh", fontsize=14, title="Top Keywords for TV Shows")
```

```
c:\python39\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `tr
esult to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tupl
  and should_run_async(code)
```

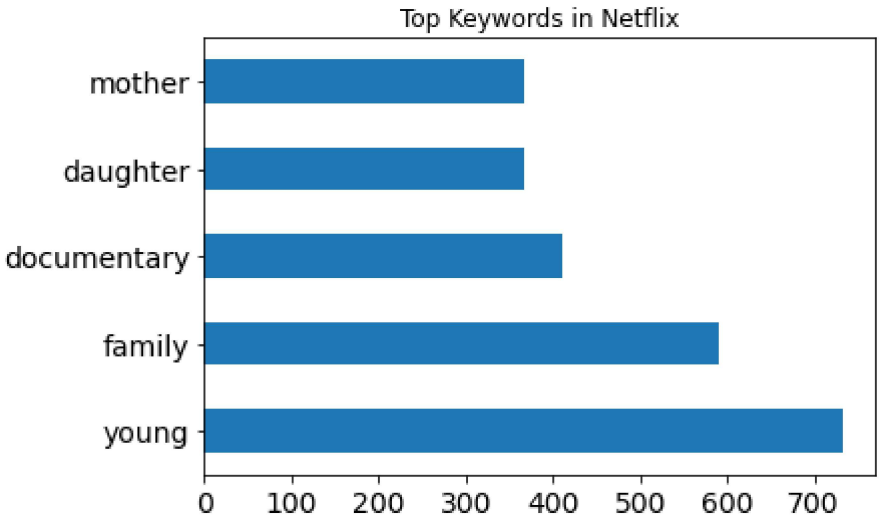
Out[46]:
```
<AxesSubplot:title={'center':'Top Keywords for TV Shows'}>
```



In [47]:
```
KW.plot(kind="barh", fontsize=14, title="Top Keywords in Netflix")
```

```
c:\python39\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `tr
esult to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tupl
  and should_run_async(code)
```

Out[47]:
```
<AxesSubplot:title={'center':'Top Keywords in Netflix'}>
```



In [48]:
```
#If people were to guess the same description, would they have the same answers?- As seen here in this data set,
unseen_df.head()
```

```
c:\python39\lib\site-packages\ipykernel\ipkernel.py:287: DeprecationWarning: `should_run_async` will not call `tr
esult to `transformed_cell` argument and any exception that happen during thetransform in `preprocessing_exc_tupl
  and should_run_async(code)
```

Out[48]:

	Description	Tokens	topic_1	topic_1_prob	topic_2	topic_2_prob	top
0	A penniless actor new to Mumbai and a beautifu...	[a, penniless, actor, new, to, and, a, beautif...	0	0.12	1	0.182802	
1	On his wedding day, an arrogant, greedy accoun...	[on, his, wedding, day, an, arrogant, greedy, ...	4	0.17	14	0.323722	
2	Convicted of rape and murder at age 18, Daniel...	[of, rape, and, murder, at, age, holden, nearl...	5	0.13	19	0.128298	
3	This period drama set in impoverished East Lon...	[this, period, drama, set, in, east, in, the, ...	18	0.15	20	0.146461	
4	After each of them loses a child to murder, tw...	[after, each, of, them, a, child, to, murder, ...	12	0.22	23	0.33612	

Final Learnings:

Doing this project was a challenge for us especially when looking for the most suitable model to use based on what we want to know about t consists of useful information and requires little data cleaning.

We had the most difficulty in using the LDA model and visualizing it because it was not part of the discussion in class, but it was made possil questions.

Based on the result we saw how there are common topics for most genres and types like the topics 19 and 18 (where tha words can be seer extracted from the model it can be seen how other words related to the words in their description gives a different definition for the movie/sho

#References:

Data set:

Retrieved from: Kaggle.com

Title: Netflix Movies and TV Shows (Movies and TV Shows listings on Netflix)

link: https://www.kaggle.com/shivamb/netflix-shows/notebooks (https://www.kaggle.com/shivamb/netflix-shows/notebooks)

LDA Topic Modelling sources:

- https://towardsdatascience.com/lda-topic-modeling-an-explanation-e184c90aadcd (https://towardsdatascience.com/lda-topic-modeling-a
- https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2 (https://towardsdatascience.com/latent-dirichlet-allocation-ld

Other Sources

- .Larose and D. Larose, (2019) Data Science Using Python and R
- https://stats.stackexchange.com/questions/185983/calculating-precision-and-recall-for-lda (https://stats.stackexchange.com/questions/1

In [ ]: