# CS170_Balita_Rono

February 13, 2021

[6]:
```
# CS2170 Final Project
# lenz Baron Balita and Cara L. Roño

!pip install gensim
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: gensim in /home/cara/.local/lib/python3.9/site-
packages (3.8.3)
Requirement already satisfied: numpy>=1.11.3 in
/home/cara/.local/lib/python3.9/site-packages (from gensim) (1.19.4)
Requirement already satisfied: smart-open>=1.8.1 in
/home/cara/.local/lib/python3.9/site-packages (from gensim) (4.1.2)
Requirement already satisfied: scipy>=0.18.1 in
/home/cara/.local/lib/python3.9/site-packages (from gensim) (1.5.4)
Requirement already satisfied: six>=1.5.0 in /usr/lib/python3.9/site-packages
(from gensim) (1.15.0)
```

[7]:
```
!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')
```

```python
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))
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pyLDAvis==2.1.2 in
/home/cara/.local/lib/python3.9/site-packages (2.1.2)
Requirement already satisfied: jinja2>=2.7.2 in
/home/cara/.local/lib/python3.9/site-packages (from pyLDAvis==2.1.2) (2.11.2)
Requirement already satisfied: numpy>=1.9.2 in
/home/cara/.local/lib/python3.9/site-packages (from pyLDAvis==2.1.2) (1.19.4)
Requirement already satisfied: wheel>=0.23.0 in
/home/cara/.local/lib/python3.9/site-packages (from pyLDAvis==2.1.2) (0.36.2)
Requirement already satisfied: pandas>=0.17.0 in
/home/cara/.local/lib/python3.9/site-packages (from pyLDAvis==2.1.2) (1.1.4)
Requirement already satisfied: future in /home/cara/.local/lib/python3.9/site-
packages (from pyLDAvis==2.1.2) (0.18.2)
Requirement already satisfied: joblib>=0.8.4 in
/home/cara/.local/lib/python3.9/site-packages (from pyLDAvis==2.1.2) (1.0.0)
Requirement already satisfied: pytest in /home/cara/.local/lib/python3.9/site-
packages (from pyLDAvis==2.1.2) (6.2.2)
Requirement already satisfied: numexpr in /home/cara/.local/lib/python3.9/site-
packages (from pyLDAvis==2.1.2) (2.7.2)
Requirement already satisfied: scipy>=0.18.0 in
/home/cara/.local/lib/python3.9/site-packages (from pyLDAvis==2.1.2) (1.5.4)
Requirement already satisfied: funcy in /home/cara/.local/lib/python3.9/site-
packages (from pyLDAvis==2.1.2) (1.15)
Requirement already satisfied: MarkupSafe>=0.23 in
/home/cara/.local/lib/python3.9/site-packages (from
jinja2>=2.7.2->pyLDAvis==2.1.2) (1.1.1)
Requirement already satisfied: pytz>=2017.2 in
/home/cara/.local/lib/python3.9/site-packages (from
pandas>=0.17.0->pyLDAvis==2.1.2) (2020.4)
Requirement already satisfied: python-dateutil>=2.7.3 in
/home/cara/.local/lib/python3.9/site-packages (from
pandas>=0.17.0->pyLDAvis==2.1.2) (2.8.1)
```

```
Requirement already satisfied: six>=1.5 in /usr/lib/python3.9/site-packages
(from python-dateutil>=2.7.3->pandas>=0.17.0->pyLDAvis==2.1.2) (1.15.0)
Requirement already satisfied: packaging in
/home/cara/.local/lib/python3.9/site-packages (from pytest->pyLDAvis==2.1.2)
(20.4)
Requirement already satisfied: attrs>=19.2.0 in
/home/cara/.local/lib/python3.9/site-packages (from pytest->pyLDAvis==2.1.2)
(20.3.0)
Requirement already satisfied: pluggy<1.0.0a1,>=0.12 in
/home/cara/.local/lib/python3.9/site-packages (from pytest->pyLDAvis==2.1.2)
(0.13.1)
Requirement already satisfied: iniconfig in
/home/cara/.local/lib/python3.9/site-packages (from pytest->pyLDAvis==2.1.2)
(1.1.1)
Requirement already satisfied: py>=1.8.2 in
/home/cara/.local/lib/python3.9/site-packages (from pytest->pyLDAvis==2.1.2)
(1.10.0)
Requirement already satisfied: toml in /home/cara/.local/lib/python3.9/site-
packages (from pytest->pyLDAvis==2.1.2) (0.10.2)
Requirement already satisfied: pyparsing>=2.0.2 in
/home/cara/.local/lib/python3.9/site-packages (from
packaging->pytest->pyLDAvis==2.1.2) (2.4.7)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nltk in /home/cara/.local/lib/python3.9/site-
packages (3.5)
Requirement already satisfied: click in /home/cara/.local/lib/python3.9/site-
packages (from nltk) (7.1.2)
Requirement already satisfied: joblib in /home/cara/.local/lib/python3.9/site-
packages (from nltk) (1.0.0)
Requirement already satisfied: regex in /usr/lib/python3.9/site-packages (from
nltk) (2020.10.15)
Requirement already satisfied: tqdm in /home/cara/.local/lib/python3.9/site-
packages (from nltk) (4.56.2)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: sklearn in /home/cara/.local/lib/python3.9/site-
packages (0.0)
Requirement already satisfied: scikit-learn in
/home/cara/.local/lib/python3.9/site-packages (from sklearn) (0.24.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/home/cara/.local/lib/python3.9/site-packages (from scikit-learn->sklearn)
(2.1.0)
Requirement already satisfied: scipy>=0.19.1 in
/home/cara/.local/lib/python3.9/site-packages (from scikit-learn->sklearn)
(1.5.4)
Requirement already satisfied: numpy>=1.13.3 in
/home/cara/.local/lib/python3.9/site-packages (from scikit-learn->sklearn)
(1.19.4)
Requirement already satisfied: joblib>=0.11 in
```

```
/home/cara/.local/lib/python3.9/site-packages (from scikit-learn->sklearn)
(1.0.0)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: gensim in /home/cara/.local/lib/python3.9/site-
packages (3.8.3)
Requirement already satisfied: six>=1.5.0 in /usr/lib/python3.9/site-packages
(from gensim) (1.15.0)
Requirement already satisfied: numpy>=1.11.3 in
/home/cara/.local/lib/python3.9/site-packages (from gensim) (1.19.4)
Requirement already satisfied: scipy>=0.18.1 in
/home/cara/.local/lib/python3.9/site-packages (from gensim) (1.5.4)
Requirement already satisfied: smart-open>=1.8.1 in
/home/cara/.local/lib/python3.9/site-packages (from gensim) (4.1.2)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: seaborn in /home/cara/.local/lib/python3.9/site-
packages (0.11.0)
Requirement already satisfied: numpy>=1.15 in
/home/cara/.local/lib/python3.9/site-packages (from seaborn) (1.19.4)
Requirement already satisfied: pandas>=0.23 in
/home/cara/.local/lib/python3.9/site-packages (from seaborn) (1.1.4)
Requirement already satisfied: scipy>=1.0 in
/home/cara/.local/lib/python3.9/site-packages (from seaborn) (1.5.4)
Requirement already satisfied: matplotlib>=2.2 in
/home/cara/.local/lib/python3.9/site-packages (from seaborn) (3.3.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/cara/.local/lib/python3.9/site-packages (from matplotlib>=2.2->seaborn)
(1.3.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
/home/cara/.local/lib/python3.9/site-packages (from matplotlib>=2.2->seaborn)
(2.4.7)
Requirement already satisfied: pillow>=6.2.0 in /usr/lib/python3.9/site-packages
(from matplotlib>=2.2->seaborn) (8.1.0)
Requirement already satisfied: python-dateutil>=2.1 in
/home/cara/.local/lib/python3.9/site-packages (from matplotlib>=2.2->seaborn)
(2.8.1)
Requirement already satisfied: cycler>=0.10 in
/home/cara/.local/lib/python3.9/site-packages (from matplotlib>=2.2->seaborn)
(0.10.0)
Requirement already satisfied: six in /usr/lib/python3.9/site-packages (from
cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)
Requirement already satisfied: pytz>=2017.2 in
/home/cara/.local/lib/python3.9/site-packages (from pandas>=0.23->seaborn)
(2020.4)
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: plotly in /home/cara/.local/lib/python3.9/site-
packages (4.14.3)
Requirement already satisfied: retrying>=1.3.3 in
/home/cara/.local/lib/python3.9/site-packages (from plotly) (1.3.3)
```

```
Requirement already satisfied: six in /usr/lib/python3.9/site-packages (from
plotly) (1.15.0)
```

```
[nltk_data] Downloading package wordnet to /home/cara/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package words to /home/cara/nltk_data...
[nltk_data]   Package words is already up-to-date!
[nltk_data] Downloading package punkt to /home/cara/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/cara/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

## 0.1 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 7000 rows of different movies and TV shows and listing each of their description, genres, title, directors, and much more.

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

[8]:
```
#If will be opened in jupyter remove this
#from google.colab import drive
#drive.mount('/content/drive')
```

[9]:
```
#path for Jupyter
df=pd.read_csv((r"/home/cara/Documents/CS170-Project/netflix_titles.csv"))

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

[10]:
```
# The data set contains the following columns
df.columns
```

[10]:
```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description'],
      dtype='object')
```

[11]:
```
#With 7787 rows and 12 columns
df.shape
```

[11]:
```
(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 we will delete some columns that will not be used in the 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. There will also be checking of duplicate rows and null 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.

[12]:
```
# checking data types of fields
df.dtypes
```

[12]:
```
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
```

[13]:
```
# Dropping irrelevant columns that will not be used.
df = df.drop(['show_id', 'cast', ], axis=1)
df.head(2)
```

[13]:
```
      type title          director country         date_added  release_year  \
0  TV Show    3%               NaN  Brazil    August 14, 2020          2020
1    Movie  7:19  Jorge Michel Grau  Mexico  December 23, 2016          2016

  rating   duration                                          listed_in  \
0  TV-MA  4 Seasons  International TV Shows, TV Dramas, TV Sci-Fi &...
1  TV-MA     93 min                       Dramas, International Movies

                                         description
0  In a future where the elite inhabit an island ...
1  After a devastating earthquake hits Mexico Cit...
```

[14]:
```
# renaming some column names.
df = df.rename(columns={"listed_in":"genre"})
df.columns
```

[14]:
```
Index(['type', 'title', 'director', 'country', 'date_added', 'release_year',
       'rating', 'duration', 'genre', 'description'],
```

```
      dtype='object')
```

[15]:
```python
# splitting the dataset between Movies and TV shows
netflix_movies = df[df['type'] == 'Movie']
netflix_shows = df[df['type'] == 'TV Show']
```

[16]:
```python
#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)
```

[17]:
```python
#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
```

[18]:
```python
#replacing null values with 'not specified' or none

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

[18]:
```
         type                                     title           director  \
0     TV Show                                        3%               none
1       Movie                                      7:19  Jorge Michel Grau
2       Movie                                     23:59       Gilbert Chan
3       Movie                                         9        Shane Acker
4       Movie                                        21     Robert Luketic
...       ...                                       ...                ...
7782    Movie                                      Zozo        Josef Fares
7783    Movie                                    Zubaan        Mozez Singh
7784    Movie                         Zulu Man in Japan               none
7785  TV Show                      Zumbo's Just Desserts               none
7786    Movie  ZZ TOP: THAT LITTLE OL' BAND FROM TEXAS           Sam Dunn

                                               country          date_added  \
0                                               Brazil     August 14, 2020
```

```
1                                                Mexico   December 23, 2016
2                                             Singapore   December 20, 2018
3                                         United States   November 16, 2017
4                                         United States     January 1, 2020
...                                                 ...                 ...
7782  Sweden, Czech Republic, United Kingdom, Denmar...    October 19, 2020
7783                                              India       March 2, 2019
7784                                               none  September 25, 2020
7785                                          Australia    October 31, 2020
7786              United Kingdom, Canada, United States       March 1, 2020

      release_year rating   duration  \
0             2020  TV-MA  4 Seasons
1             2016  TV-MA     93 min
2             2011      R     78 min
3             2009  PG-13     80 min
4             2008  PG-13    123 min
...            ...    ...        ...
7782          2005  TV-MA     99 min
7783          2015  TV-14    111 min
7784          2019  TV-MA     44 min
7785          2019  TV-PG   1 Season
7786          2019  TV-MA     90 min

                                                  genre  \
0     International TV Shows, TV Dramas, TV Sci-Fi &...
1                          Dramas, International Movies
2                   Horror Movies, International Movies
3     Action & Adventure, Independent Movies, Sci-Fi...
4                                                Dramas
...                                                 ...
7782                       Dramas, International Movies
7783     Dramas, International Movies, Music & Musicals
7784  Documentaries, International Movies, Music & M...
7785                 International TV Shows, Reality TV
7786                    Documentaries, Music & Musicals

                                            description
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...
...                                                 ...
7782  When Lebanon's Civil War deprives Zozo of his ...
7783  A scrappy but poor boy worms his way into a ty...
7784  In this documentary, South African rapper Nast...
```

```
7785   Dessert wizard Adriano Zumbo looks for the nex...
7786   This documentary delves into the mystique behi...

[7787 rows x 10 columns]
```

[19]:
```
#standardizing numeric field 'release_year'
from scipy import stats
df['release_year_z']= stats.zscore(df['release_year'])
df['release_year_z']
```

[19]:
```
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
```

[20]:
```
# 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
```

[20]:
```
Empty DataFrame
Columns: [type, title, director, country, date_added, release_year, rating,
duration, genre, description, release_year_z]
Index: []
```

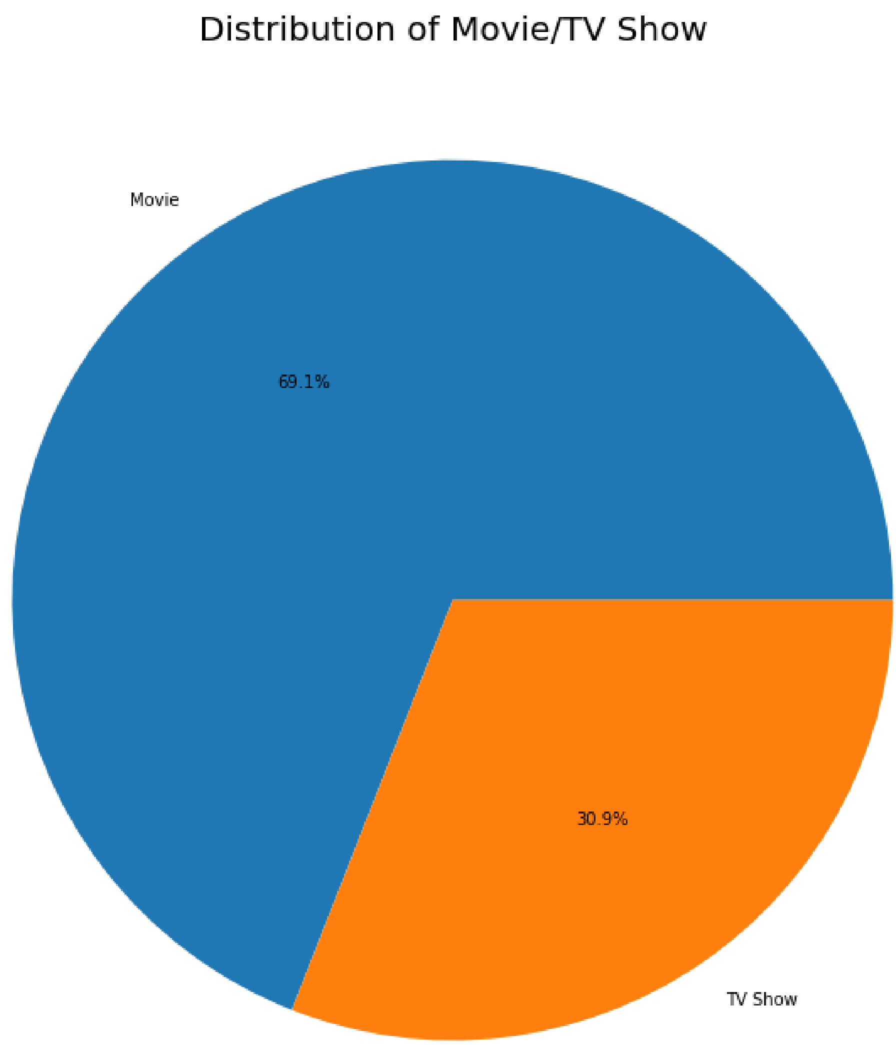
# 1 III. Exploratory Data Analysis

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?
```

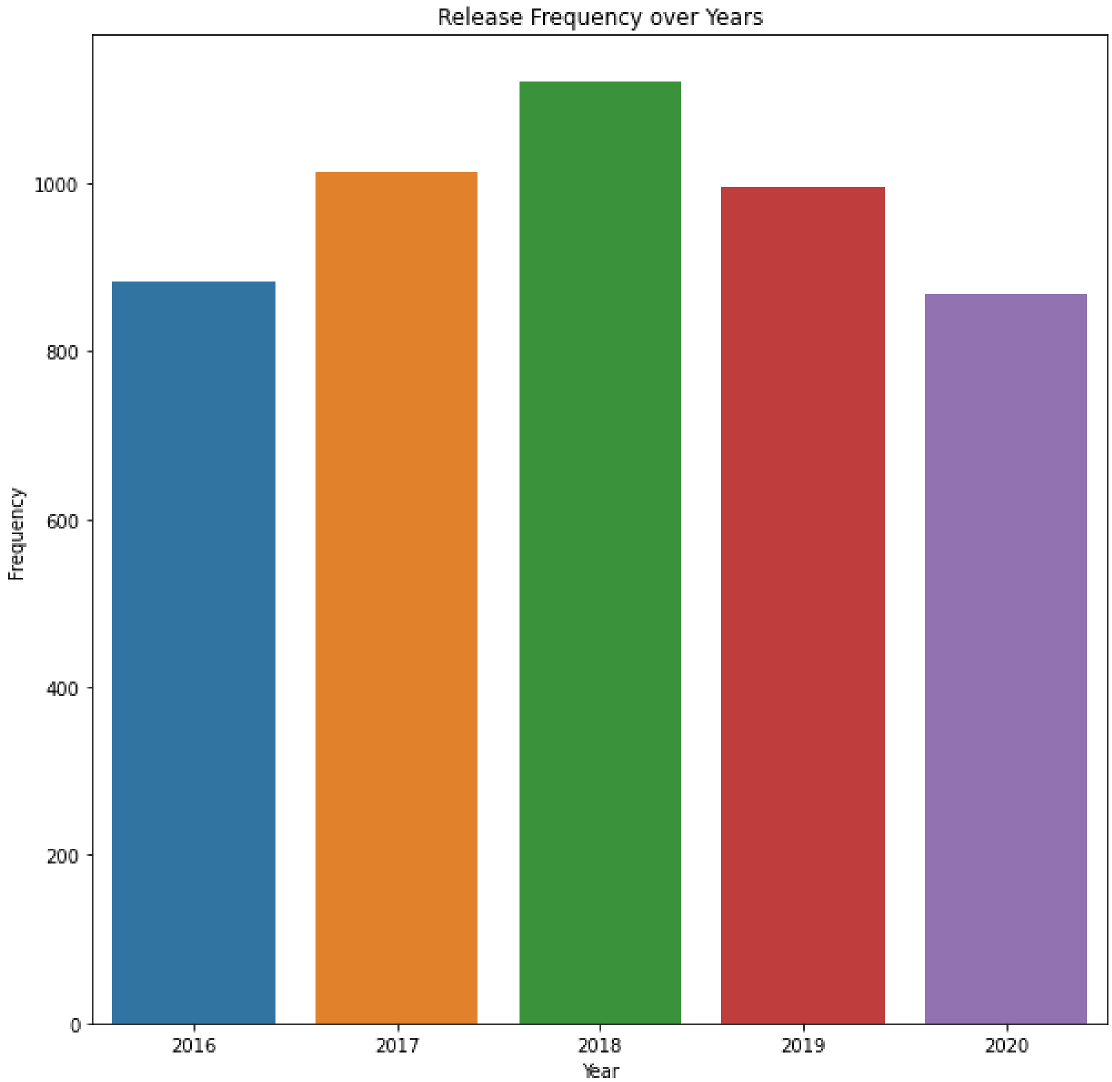
[21]:
```
# Distribution of Movies/Shows
bar, ax = plt.subplots(figsize = (12,12))
plt.pie(df['type'].value_counts(), labels = df['type'].value_counts().index,␣
 ↪autopct="%.1f%%")
plt.title('Distribution of Movie/TV Show', size=20)
```

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




[22]:
```
#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')
```

[22]: Text(0.5, 1.0, 'Release Frequency over Years')

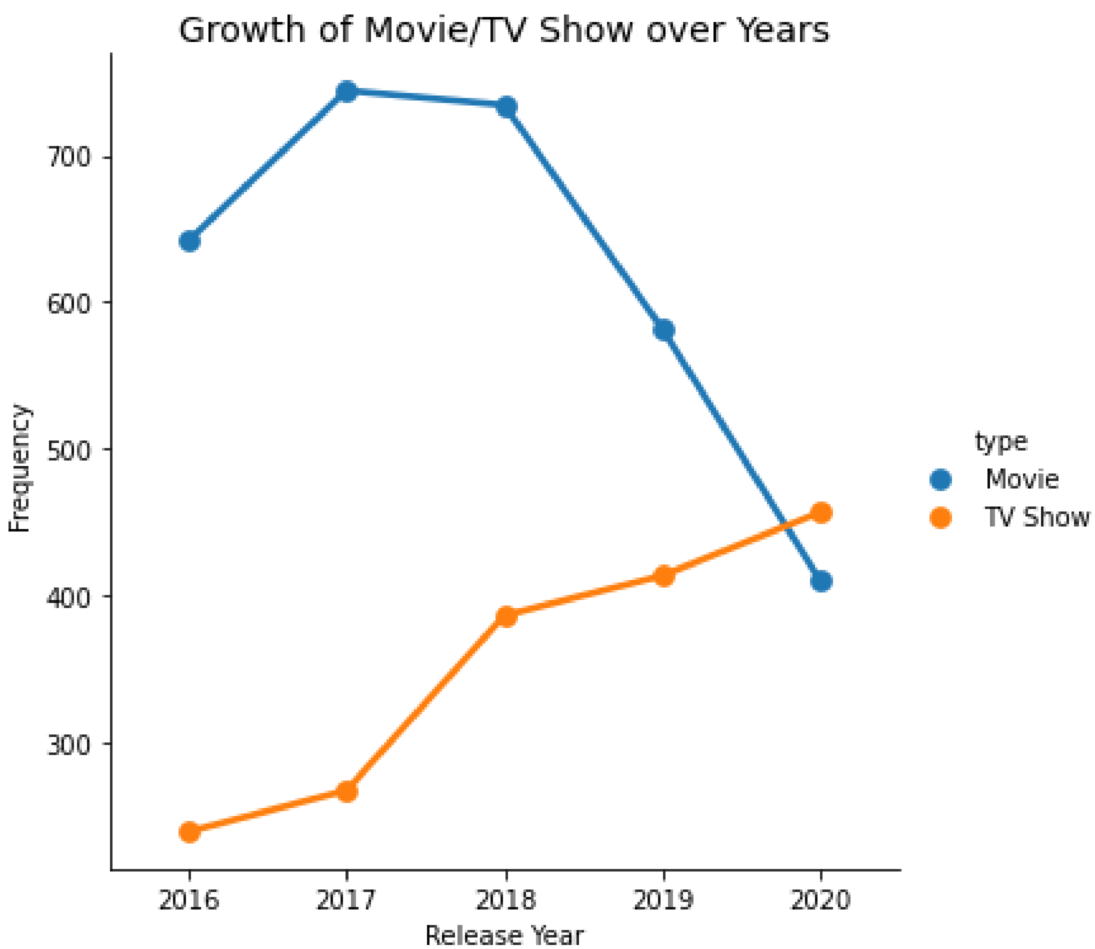



The bar plot above indicates that the year of the mose production of movies and TV shows in netflix was in 2018

[23]:
```
#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]])
```

[24]:
```
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)
```

[24]:
```
Text(0.5, 1.0, 'Growth of Movie/TV Show over Years')
```

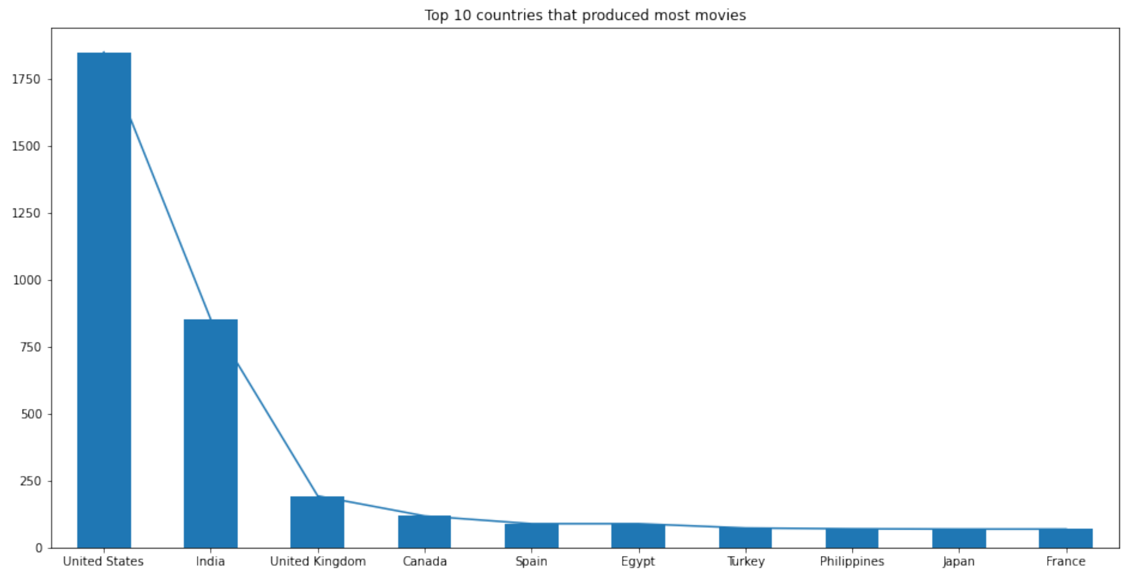



In the plot above it can be seen how there is a decrease of production of Movies from 2018 onwards and increase in the production of TV shows.

[25]:
```
#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))
```

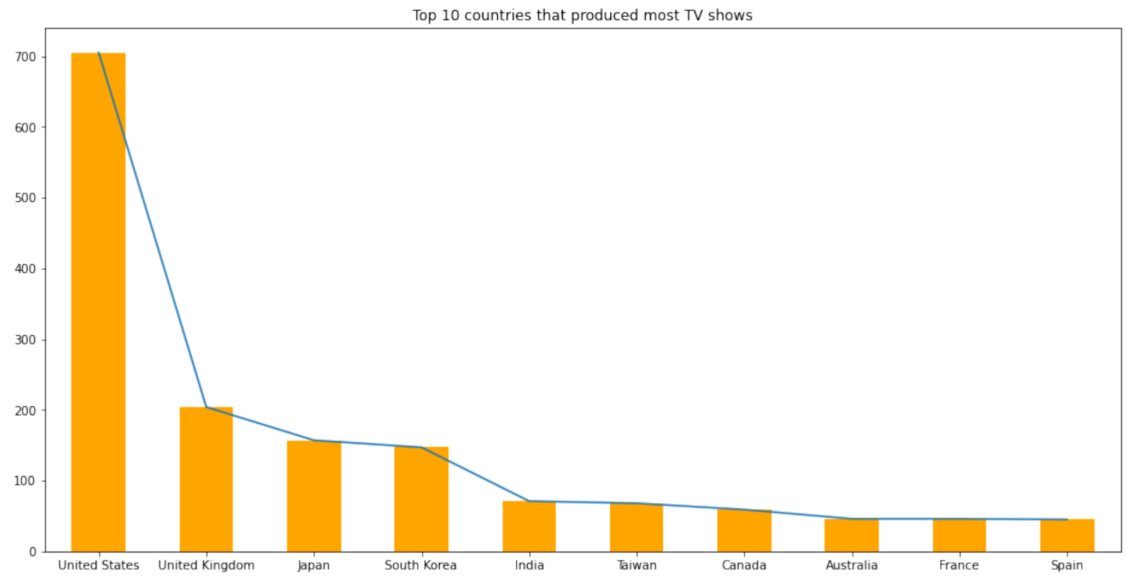
[25]: <AxesSubplot:title={'center':'Top 10 countries that produced most movies'}>




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 which is cool.

[26]:
```
#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))
```

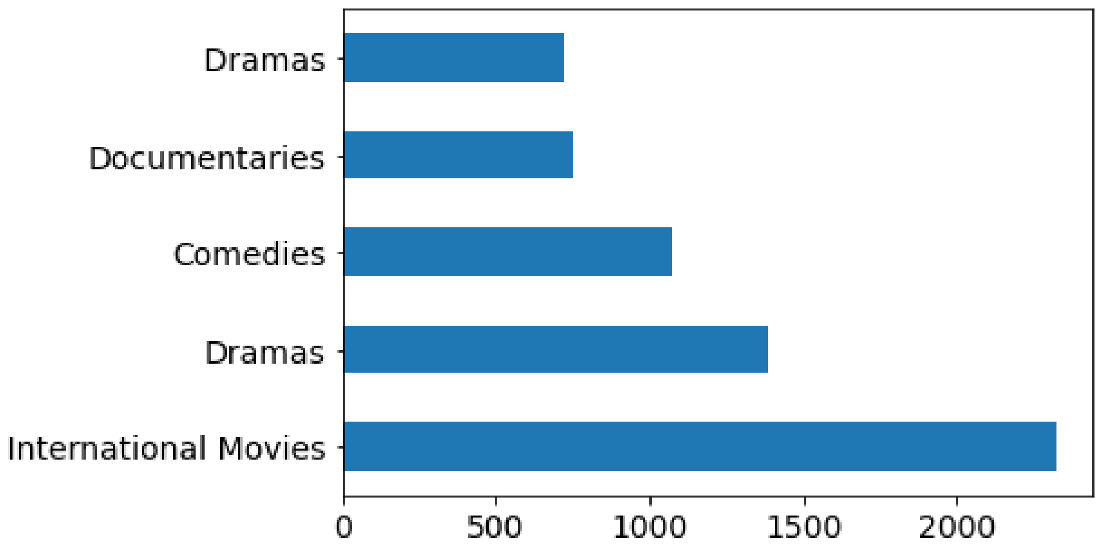
[26]: <AxesSubplot:title={'center':'Top 10 countries that produced most TV shows'}>




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

[27]:
```python
# 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)
```

[27]: <AxesSubplot:>




## 1.1 Recommendations

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

[28]:
```python
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, norm=u'l2', smooth_idf = True):
        self.data_frame = data_frame
        self.model = TfidfVectorizer(max_df=threshold,stop_words=stop_words, lowercase=lowercase, use_idf=use_idf,norm=norm,smooth_idf=smooth_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] ))
```

[29]:
```
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
```

[30]:
```
#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
```

## 2 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

[31]:
```
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")
```

```python
from gensim.models import CoherenceModel

#Choosing best parameter and topics
topics_arr = [20, 40]
learning_decay = [0.5, 0.7, 0.9]
minimum_probability= [0.01, 0.05, 0.08]

#Chosen from the above function (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_probability=min_probability,␣
↪decay=learning_decay)

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-33 => 0.064*"work" + 0.048*"break" + 0.039*"beloved" + 0.033*"attempt" +
0.028*"reporter" + 0.027*"community" + 0.023*"danger" + 0.022*"living" +
0.021*"family" + 0.021*"father"
Topic-29 => 0.054*"killer" + 0.054*"steal" + 0.041*"trouble" + 0.034*"hilarious"
+ 0.033*"may_be" + 0.023*"chaos" + 0.022*"professor" + 0.021*"wilderness" +
0.020*"evidence" + 0.019*"wedding"
Topic-8 => 0.097*"career" + 0.074*"come" + 0.074*"personal" + 0.035*"teacher" +
0.027*"wrong" + 0.025*"thriller" + 0.025*"documentary" + 0.025*"outside" +
0.023*"intimate" + 0.019*"rise"
Topic-14 => 0.089*"power" + 0.035*"soccer" + 0.034*"get" + 0.034*"black" +
0.027*"professional" + 0.026*"struggle" + 0.026*"summer" + 0.024*"newly" +
0.021*"personal" + 0.019*"ancient"
Topic-13 => 0.044*"inspire" + 0.040*"party" + 0.038*"sport" + 0.033*"politics" +
0.032*"tale" + 0.031*"culture" + 0.030*"series" + 0.030*"remote" + 0.024*"twist"
+ 0.021*"house"
Topic-23 => 0.093*"history" + 0.058*"try" + 0.031*"big" + 0.030*"world" +
0.027*"science" + 0.025*"documentary" + 0.025*"make" + 0.023*"extreme" +
0.022*"hire" + 0.021*"private"
Topic-5 => 0.093*"comedy" + 0.070*"relationship" + 0.053*"couple" +
0.050*"each_other" + 0.041*"dance" + 0.034*"start" + 0.032*"romantic" +
0.029*"talented" + 0.027*"catch" + 0.018*"special"
Topic-37 => 0.062*"music" + 0.049*"local" + 0.046*"explore" + 0.035*"modern" +
```

```
0.030*"country" + 0.027*"young" + 0.026*"artist" + 0.026*"fall" +
0.026*"ambitious" + 0.023*"small_town"
Topic-0 => 0.077*"comedian" + 0.077*"leave" + 0.075*"mission" + 0.045*"village"
+ 0.029*"family" + 0.028*"seek" + 0.028*"trap" + 0.026*"rival" + 0.025*"small" +
0.020*"special"
Topic-28 => 0.096*"police" + 0.056*"doctor" + 0.052*"unexpected" +
0.043*"corruption" + 0.039*"businessman" + 0.032*"hospital" + 0.031*"medical" +
0.024*"birth" + 0.022*"model" + 0.021*"lead"
```

[32]:

```
#Description-topic distributions for our training set  - It lists top 4 keywords␣
 ↪and Dominant topic for each sentence
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()
```

[32]:

```
                                      Description  \
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...

                          Top Keywords  Dominant Topic  Probability
0              search,human,quest,prove              24         0.30
1                 history,try,big,world              23         0.43
2               order,world,teen,military               3         0.46
3  friend,wealthy,detective,investigate              35         0.48
4               fight,marry,social,action              26         0.86
```

[33]:

```
#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_3_prob", 'Tokens']).round(2)
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()
```

[33]:

```
                                       Description  \
0  When frustrated politicians name a historical ...
1  This enlightening series from Vox digs into a ...
2  The level-headed owner of a struggling coffee ...
3  When a recently single psychologist moves to a...
4  Determined to get his mitts on $9 billion in a...

                                              Tokens topic_1 topic_1_prob  \
0  [when, name, a, historical, figure, as, the, n...       7     0.128128
1  [this, enlightening, series, from, digs, into,...       2     0.128977
2  [the, owner, of, a, struggling, coffee, shop, ...       5     0.141794
3  [when, a, recently, single, psychologist, to, ...       5     0.202474
4  [determined, to, get, his, on, billion, in, a,...      10     0.142548

  topic_2 topic_2_prob topic_3 topic_3_prob
0      16     0.128129      30     0.253134
1       6     0.154082      11     0.475998
2      12     0.249032      16     0.173079
3      11     0.102516      34     0.102516
4      16     0.113924      26     0.307586
```

clustering data set with the model results

[34]:

```
#Clustering each movie/show per genre

#adding genre column in lds_distribution table
genres = pd.DataFrame(df['genre'])
```

```python
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
```

[34]:
```
                         Genre  \
0       International TV Shows
1                       Dramas
2                Horror Movies
3           Action & Adventure
4                       Dramas
...                        ...
6998                  Kids' TV
6999                  Kids' TV
7000   Children & Family Movies
7001          British TV Shows
7002   Children & Family Movies

                                            Description  \
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...
...                                                 ...
6998  A drug dealer starts having doubts about his t...
6999  Dragged from civilian life, a former superhero...
7000  When Lebanon's Civil War deprives Zozo of his ...
7001  A scrappy but poor boy worms his way into a ty...
7002  In this documentary, South African rapper Nast...

                          Top_Keywords  Dominant_Topic  Probability  \
0              search,human,quest,prove              24         0.30
1                 history,try,big,world              23         0.43
2               order,world,teen,military               3         0.46
```

```
3     friend,wealthy,detective,investigate              35          0.48
4                fight,marry,social,action              26          0.86
...                                    ...             ...           ...
6998        aspire,actor,competition,movie              32          0.29
6999             fight,marry,social,action              26          0.25
7000   adventure,friendship,travel,magical              11          0.27
7001               future,take,hotel,owner              12          0.22
7002        people,documentary,world,right              27          0.37

         type
0     TV Show
1       Movie
2       Movie
3       Movie
4       Movie
...       ...
6998  TV Show
6999  TV Show
7000    Movie
7001  TV Show
7002    Movie

[7003 rows x 6 columns]
```

[35]:

```
#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')]
```

[36]:

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

[37]:

```
#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)
```

[38]:

```
#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)
```

[39]:

```
#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)
```

[40]:

```
#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
```

[40]:

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

[41]:

```
#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
```

[41]:
```
      Genre   Top_Topics
0  [Drama]  [9, 18, 19]
```

[42]:
```
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
```

[42]:
```
      Genre   Top_Topics
0   [Drama]  [9, 18, 19]
1  [Comedy]  [19, 3, 18]
```

[43]:
```
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
```

[43]:
```
             Genre   Top_Topics
0          [Drama]  [9, 18, 19]
1         [Comedy]  [19, 3, 18]
2  [Documentaries]  [14, 5, 18]
```

```
[44]: 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
```

```
[44]:                     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

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

```
Perplexity:  -13.325055223165805
```

```
[46]: #Coherence - measure the degree of semantic similarity between high scoring␣
↪words in each topic (and then average across topics)
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:  0.5737443699030829
```

# 3 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: What are the common topics in netflix based on their 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 helpful when assigning genres, recommending movies, etc. First we will look into the common topics per genre. We will only look into the top genres in Netflix: Dramas, Documentaries, Comedy, and International movies.

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" description can be seen with sum of "n" number of topics and 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 unsupervised method. Coherence score is good, but with the unseen data predictions the probability of the distributions of topics are low.

[47]:
```
# In here we will see the top 3 topics per genre, but they are still in number␣
↪and will be translated into the words using the pyLDAvis chart below .
TopTopicsDF
```

[47]:
```
                      Genre   Top_Topics
0                   [Drama]  [9, 18, 19]
1                  [Comedy]  [19, 3, 18]
2           [Documentaries]  [14, 5, 18]
3  [International Movies]    [27, 5, 9]
```

[48]:
```
# top topics per Type(Movies/TVshows)
TopTopicsTypes
```

[48]:
```
          Type   Top_Topics
0     [Movies]  [9, 14, 18]
1   [TV Shows]  [3, 18, 14]
```

[49]:
```
#To easily read the top topics per genre, you just have to input the number of␣
↪the topic in the texinput of the chart
#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 prevalent each topic is and how they relate to each other.
#the bars represent the terms that are most useful in interpreting the topic␣
↪currently selected

#once the topic number is inputted it will show the relevant terms in that␣
↪specific topic.
```

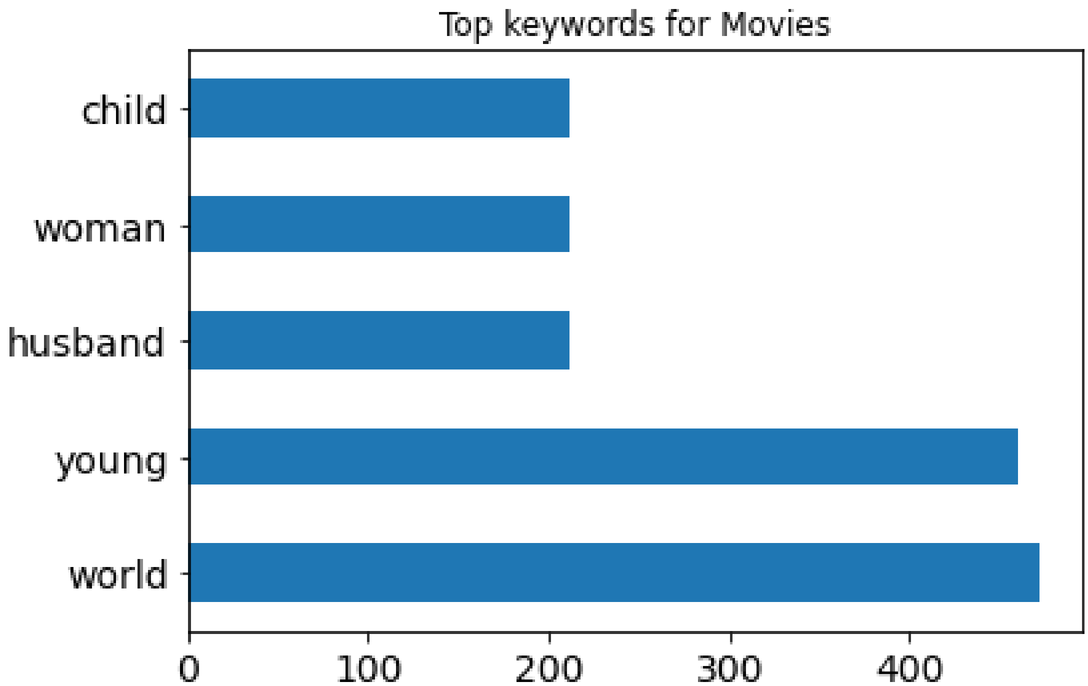
```
[49]: PreparedData(topic_coordinates=              x         y  topics  cluster
Freq
topic
0      0.031566 -0.104618       1        1  2.808576
1      0.001444  0.007141       2        1  2.269457
2      0.089662 -0.047609       3        1  2.666737
3     -0.092610  0.115941       4        1  2.028351
4     -0.098057 -0.084574       5        1  2.243303
5     -0.009724 -0.061580       6        1  2.815199
6      0.076841  0.075756       7        1  2.520752
7     -0.053073  0.023681       8        1  1.976706
8     -0.124483 -0.093513       9        1  2.420316
9     -0.087035 -0.093481      10        1  2.183711
10     0.036945 -0.037846      11        1  2.760313
11    -0.053035  0.089096      12        1  2.925085
12     0.050931 -0.007591      13        1  2.283241
13    -0.042620  0.109384      14        1  2.646668
14    -0.073208  0.000414      15        1  2.244349
15    -0.035807  0.010987      16        1  2.507680
16     0.027442 -0.072457      17        1  2.371223
17    -0.111184  0.056972      18        1  2.206662
18    -0.077309  0.039162      19        1  2.027325
19     0.142639  0.098181      20        1  2.450942
20    -0.041382 -0.100080      21        1  2.077542
21    -0.003622  0.082061      22        1  3.067269
22     0.060718 -0.042692      23        1  3.149176
23    -0.074327  0.122498      24        1  2.517311
24     0.061241  0.014891      25        1  2.463551
25     0.265756  0.025103      26        1  3.682007
26    -0.029535 -0.007088      27        1  2.739529
27    -0.080629  0.123478      28        1  2.282976
28    -0.093610 -0.093275      29        1  2.151034
29    -0.023716  0.015414      30        1  1.983090
30    -0.078679  0.086710      31        1  2.457412
31     0.004464 -0.127740      32        1  3.141392
32    -0.021047 -0.038307      33        1  2.236048
33     0.096794 -0.089461      34        1  2.374653
34     0.131319 -0.034069      35        1  2.420394
35     0.134784  0.055424      36        1  3.190554
36     0.012185 -0.010533      37        1  2.480913
37     0.026524  0.070576      38        1  3.090204
38     0.075056  0.046669      39        1  2.010604
39    -0.021619 -0.123025      40        1  2.127745, topic_info=          Term
Freq        Total Category  logprob  loglift
215      woman  392.000000  392.000000  Default  30.0000  30.0000
48       death  147.000000  147.000000  Default  29.0000  29.0000
211     mother  157.000000  157.000000  Default  28.0000  28.0000
```

```
290      fight  141.000000  141.000000  Default  27.0000  27.0000
57      school  147.000000  147.000000  Default  26.0000  26.0000
...        ...         ...         ...      ...      ...      ...
1541    thanks   11.680279   16.416565  Topic40  -4.4381   3.5097
2778   willing    8.614086   10.625916  Topic40  -4.7426   3.6402
169     family   20.126260  477.950991  Topic40  -3.8940   0.6826
433     social    9.988147   69.102287  Topic40  -4.5946   1.9159
342   director    8.957705   47.244052  Topic40  -4.7035   2.1873

[1588 rows x 6 columns], token_table=      Topic      Freq       Term
term
387      30  0.951327    abandon
2442      7  0.868261  abduction
2243     10  0.901487    ability
2540     27  0.950826     aboard
1877     21  0.932123      abuse
...     ...       ...        ...
114      36  0.056380      young
114      38  0.064671      young
114      39  0.028190      young
883      29  0.936612      youth
2407     14  0.928380     zombie

[1872 rows x 3 columns], R=30, lambda_step=0.01, plot_opts={'xlab': 'PC1',
'ylab': 'PC2'}, topic_order=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
36, 37, 38, 39, 40])
```

[55]:
```
#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, shows, and all of types in Netflix.

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

```
/home/cara/.local/lib/python3.9/site-packages/ipykernel/ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

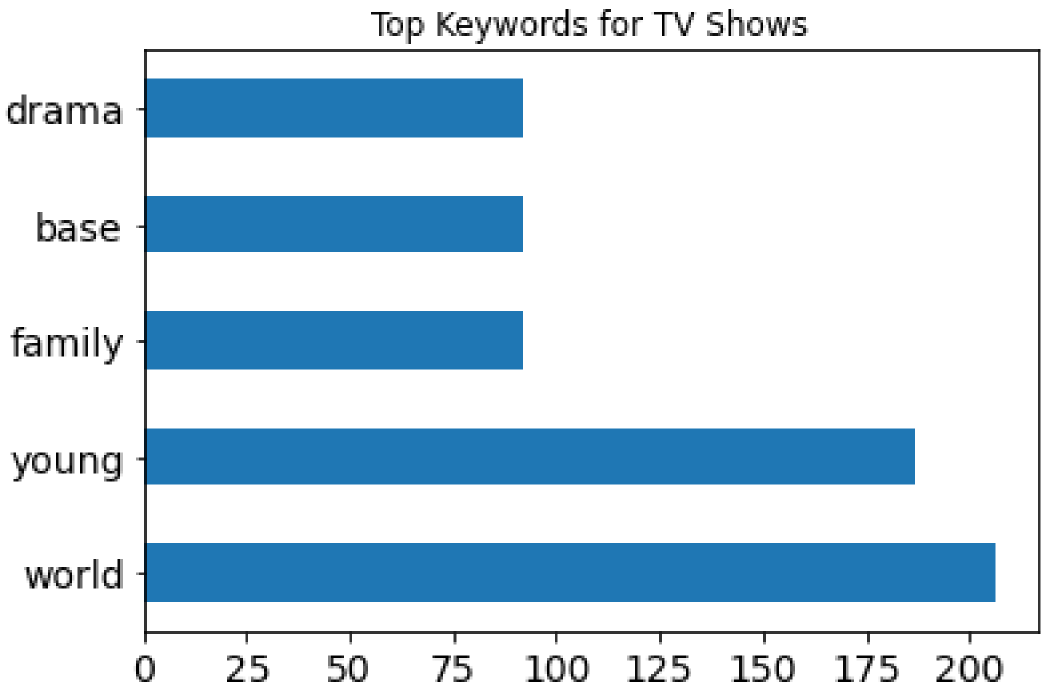
[55]:
```
<AxesSubplot:title={'center':'Top keywords for Movies'}>
```




[52]:
```
KWShows.plot(kind="barh", fontsize=14, title="Top Keywords for TV Shows")
```

```
/home/cara/.local/lib/python3.9/site-packages/ipykernel/ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

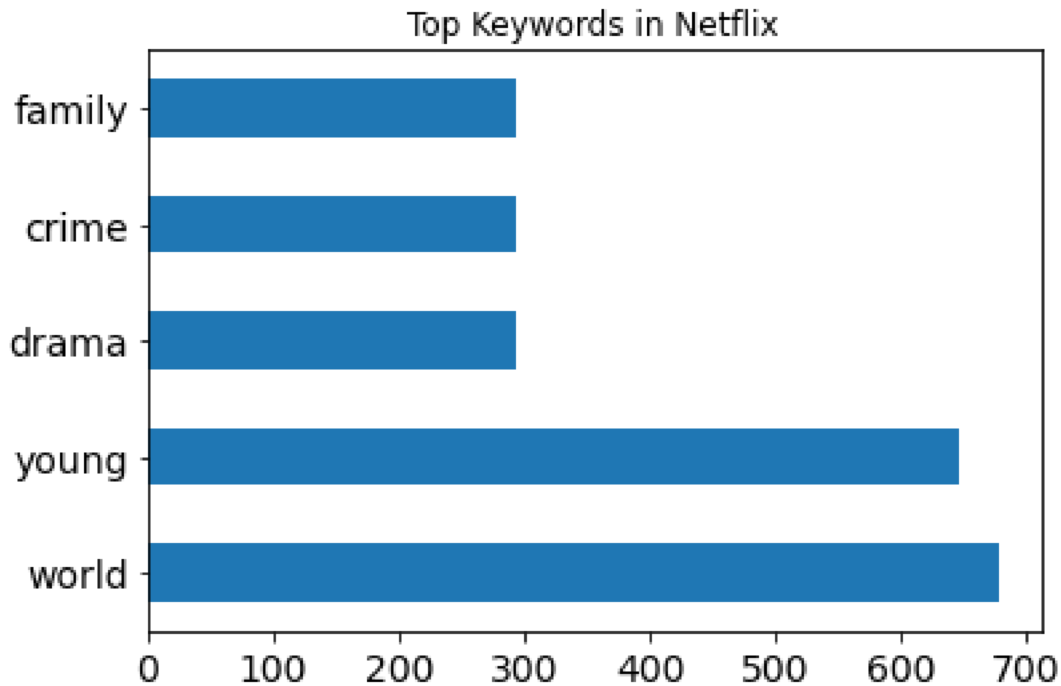
[52]:
```
<AxesSubplot:title={'center':'Top Keywords for TV Shows'}>
```




[53]:
```
KW.plot(kind="barh", fontsize=14, title="Top Keywords in Netflix")
```

```
/home/cara/.local/lib/python3.9/site-packages/ipykernel/ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[53]:
```
<AxesSubplot:title={'center':'Top Keywords in Netflix'}>
```




[54]:
```
#If people were to guess the same description, would they have the same answers?
↪- As seen here in this data set, every topic in each description ended up with␣
↪different results.
unseen_df.head()
```

```
/home/cara/.local/lib/python3.9/site-packages/ipykernel/ipkernel.py:287:
DeprecationWarning: `should_run_async` will not call `transform_cell`
automatically in the future. Please pass the result to `transformed_cell`
argument and any exception that happen during thetransform in
`preprocessing_exc_tuple` in IPython 7.17 and above.
  and should_run_async(code)
```

[54]:
```
                                         Description  \
0  When frustrated politicians name a historical ...
1  This enlightening series from Vox digs into a ...
2  The level-headed owner of a struggling coffee ...
3  When a recently single psychologist moves to a...
4  Determined to get his mitts on $9 billion in a...

                                              Tokens topic_1 topic_1_prob  \
0  [when, name, a, historical, figure, as, the, n...       7     0.128128
1  [this, enlightening, series, from, digs, into,...       2     0.128977
2  [the, owner, of, a, struggling, coffee, shop, ...       5     0.141794
3  [when, a, recently, single, psychologist, to, ...       5     0.202474
```

```
4  [determined, to, get, his, on, billion, in, a,...      10     0.142548

  topic_2 topic_2_prob topic_3 topic_3_prob
0      16     0.128129      30     0.253134
1       6     0.154082      11     0.475998
2      12     0.249032      16     0.173079
3      11     0.102516      34     0.102516
4      16     0.113924      26     0.307586
```

# 4 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 the dataset. Luckily, we found a dataset that already 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 possible and we extracted meaningful results that answers our 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 seen in the pyLDavis Chart). Also, based on the unseen data 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/show.

#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

**LDA Topic Modelling sources:** - https://towardsdatascience.com/lda-topic-modeling-an-explanation-e184c90aadcd - https://towardsdatascience.com/latent-dirichlet-allocation-lda-9d1cd064ffa2

**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


[ ]: