CSCI - 6409 - The Process of Data Science - Fall 2022

</center>

Assignment 4

</center>

Meagan Sinclair
B00737317

Samarth Jariwala
B00899380

1. Data Understanding

In [1]:
```
from google.colab import drive
import pandas as pd
import warnings
import nltk
from nltk.tokenize import word_tokenize
```

In [2]:
```
drive.mount('/content/drive')
df = pd.read_csv("/content/drive/MyDrive/wiki_movie_plots_deduped.csv")
```

```
Mounted at /content/drive
```

In [3]:
```
df = df.convert_dtypes()
```

In [4]:
```
# wiki page URL not useful info, drop
df = df.drop('Wiki Page', axis='columns')
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34886 entries, 0 to 34885
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   Release Year      34886 non-null  Int64 
 1   Title             34886 non-null  string
 2   Origin/Ethnicity  34886 non-null  string
 3   Director          34886 non-null  string
 4   Cast              33464 non-null  string
 5   Genre             34886 non-null  string
 6   Plot              34886 non-null  string
dtypes: Int64(1), string(6)
memory usage: 1.9 MB
```

1.1 Data Quality Report

In [6]:

```python
# code source: Tutorial

def build_categorical_features_report(data_df):

    """Build tabular report for categorical features"""

    def _mode(df):
        return df.apply(lambda ft: ft.mode().to_list())

    def _mode_freq(df):
        return df.apply(lambda ft: ft.value_counts()[ft.mode()].sum())

    def _second_mode(df):
        return df.apply(lambda ft: ft[~ft.isin(ft.mode())].mode().to_list())

    def _second_mode_freq(df):
        return df.apply(
            lambda ft: ft[~ft.isin(ft.mode())]
            .value_counts()[ft[~ft.isin(ft.mode())].mode()]
            .sum()
        )

    stats = {
        "Count": len,
        "Miss %": lambda df: df.isna().sum() / len(df) * 100,
        "Card.": lambda df: df.nunique(),
        "Mode": _mode,
        "Mode Freq": _mode_freq,
        "Mode %": lambda df: _mode_freq(df) / len(df) * 100,
        "2nd Mode": _second_mode,
        "2nd Mode Freq": _second_mode_freq,
        "2nd Mode %": lambda df: _second_mode_freq(df) / len(df) * 100,
    }

    cat_feat_names = data_df.select_dtypes(exclude="number").columns
    continuous_data_df = data_df[cat_feat_names]

    report_df = pd.DataFrame(index=cat_feat_names, columns=stats.keys())

    for stat_name, fn in stats.items():
        # NOTE: ignore warnings for empty features
        with warnings.catch_warnings():
            warnings.simplefilter("ignore", category=RuntimeWarning)
            report_df[stat_name] = fn(continuous_data_df)

    return report_df
```

In [7]:

```python
build_categorical_features_report(df)
```

Out[7]:

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %
Title	34886	0.000000	32432	[Cinderella, The Three Musketeers]	16	0.045864	[Treasure Island]	7	0.020065
Origin/Ethnicity	34886	0.000000	24	[American]	17377	49.810812	[British]	3670	10.519979
Director	34886	0.000000	12593	[Unknown]	1124	3.221923	[Michael Curtiz]	79	0.226452
Cast	34886	4.076134	32182	[Tom and Jerry]	80	0.229318	[Three Stooges]	56	0.160523

	Count	Miss %	Card.	Mode	Mode Freq	Mode %	2nd Mode	2nd Mode Freq	2nd Mode %
Genre	34886	0.000000	2265	[unknown]	6083	17.436794	[drama]	5964	17.095683
Plot	34886	0.000000	33869	[(マッスル人参争奪！超人大戦争, Massuru Ninjin Soudatsu! Cho...	6	0.017199	[The films take place three years after the ev...	10	0.028665

In [8]:

```
df['Title_length'] = df.Title.str.split().str.len()
df['Plot_length']=df.Plot.str.split().str.len()
df
```

Out[8]:

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length
0	1901	Kansas Saloon Smashers	American	Unknown	<NA>	unknown	A bartender is working at a saloon, serving dr...	3	83
1	1901	Love by the Light of the Moon	American	Unknown	<NA>	unknown	The moon, painted with a smiling face hangs ov...	7	86
2	1901	The Martyred Presidents	American	Unknown	<NA>	unknown	The film, just over a minute long, is composed...	3	76
3	1901	Terrible Teddy, the Grizzly King	American	Unknown	<NA>	unknown	Lasting just 61 seconds and consisting of two ...	5	153
4	1902	Jack and the Beanstalk	American	George S. Fleming, Edwin S. Porter	<NA>	unknown	The earliest known adaptation of the classic f...	4	140
...	...	...	...	...	...	...	...	...	...
34881	2014	The Water Diviner	Turkish	Director: Russell Crowe	Director: Russell Crowe Cast: Russell Crowe, ...	unknown	The film begins in 1919, just after World War ...	3	591
34882	2017	Çalgı Çengi İkimiz	Turkish	Selçuk Aydemir	Ahmet Kural, Murat Cemcir	comedy	Two musicians, Salih and Gürkan, described the...	3	11

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length
34883	2017	Olanlar Oldu	Turkish	Hakan Algül	Ata Demirer, Tuvana Türkay, Ülkü Duru	comedy	Zafer, a sailor living with his mother Döndü i...	2	67
34884	2017	Non-Transferable	Turkish	Brendan Bradley	YouTubers Shanna Malcolm, Shira Lazar, Sara Fl...	romantic comedy	The film centres around a young woman named Am...	1	193
34885	2017	İstanbul Kırmızısı	Turkish	Ferzan Özpetek	Halit Ergenç, Tuba Büyüküstün, Mehmet Günsür, ...	romantic	The writer Orhan Şahin returns to İstanbul aft...	2	48

34886 rows × 9 columns

The non-alphanumerical data is mostly found in cast and plot columns Reference: https://www.w3resource.com/python-exercises/pandas/string/python-pandas-string-exercise-30.php

In [9]:

```
import re as re
def find_nonalpha(text):
    result = re.findall("[^A-Za-z0-9 ]",text)
    return result
df['nonalpha_plot']=df['Plot'].apply(lambda x: find_nonalpha(x))
df['nonalpha_title']=df['Title'].apply(lambda x: find_nonalpha(x))

df
```

Out[9]:

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length
0	1901	Kansas Saloon Smashers	American	Unknown	<NA>	unknown	A bartender is working at a saloon, serving dr...	3	83
1	1901	Love by the Light of the Moon	American	Unknown	<NA>	unknown	The moon, painted with a smiling face hangs ov...	7	86
2	1901	The Martyred Presidents	American	Unknown	<NA>	unknown	The film, just over a minute long, is composed...	3	76
3	1901	Terrible Teddy, the Grizzly King	American	Unknown	<NA>	unknown	Lasting just 61 seconds and consisting of two ...	5	153

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length
4	1902	Jack and the Beanstalk	American	George S. Fleming, Edwin S. Porter	<NA>	unknown	The earliest known adaptation of the classic f...	4	140
...	...	...	...	...	...	...	...	...	...
34881	2014	The Water Diviner	Turkish	Director: Russell Crowe	Director: Russell Crowe Cast: Russell Crowe, ...	unknown	The film begins in 1919, just after World War ...	3	591
34882	2017	Çalgı Çengi İkimiz	Turkish	Selçuk Aydemir	Ahmet Kural, Murat Cemcir	comedy	Two musicians, Salih and Gürkan, described the...	3	11
34883	2017	Olanlar Oldu	Turkish	Hakan Algül	Ata Demirer, Tuvana Türkay, Ülkü Duru	comedy	Zafer, a sailor living with his mother Döndü i...	2	67
34884	2017	Non-Transferable	Turkish	Brendan Bradley	YouTubers Shanna Malcolm, Shira Lazar, Sara Fl...	romantic comedy	The film centres around a young woman named Am...	1	193
34885	2017	İstanbul Kırmızısı	Turkish	Ferzan Özpetek	Halit Ergenç, Tuba Büyüküstün, Mehmet Günsür, ...	romantic	The writer Orhan Şahin returns to İstanbul aft...	2	48

34886 rows × 11 columns

Word Counter reference: https://predictivehacks.com/?all-tips=word-counts-in-pandas-data-frames

In [10]:

```
Title_Word_Count = df['Title'].str.lower().str.replace('[^\w\s]','')
new_df = Title_Word_Count.str.split(expand=True).stack().value_counts().reset_index()
new_df.columns = ['Word', 'Frequency']
new_df
```

```
<ipython-input-10-1f93ab7d655a>:1: FutureWarning: The default value of regex will change f
rom True to False in a future version.
  Title_Word_Count = df['Title'].str.lower().str.replace('[^\w\s]','')
```

Out[10]:

	Word	Frequency
0	the	8672
1	of	2699
2	a	1219

	Word	Frequency
3	in	1015
4	and	974
...	...	...
22931	merchant	1
22932	sandakan	1
22933	ponmudi	1
22934	ansatsu	1
22935	kırmızısı	1

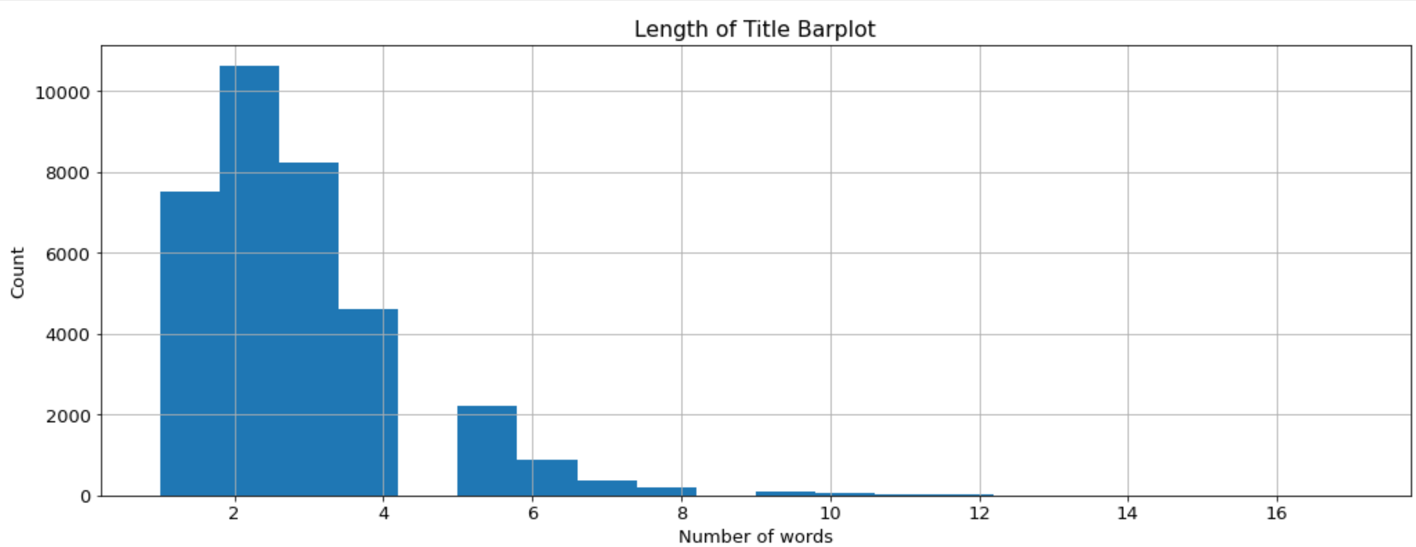
22936 rows × 2 columns

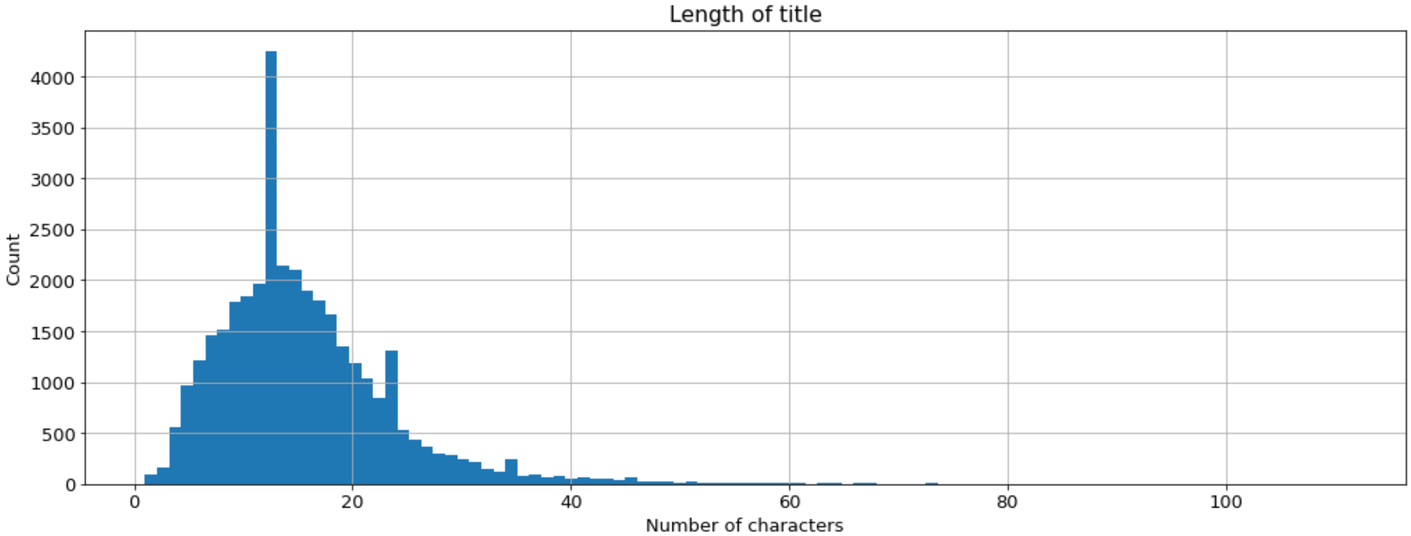
In [11]:

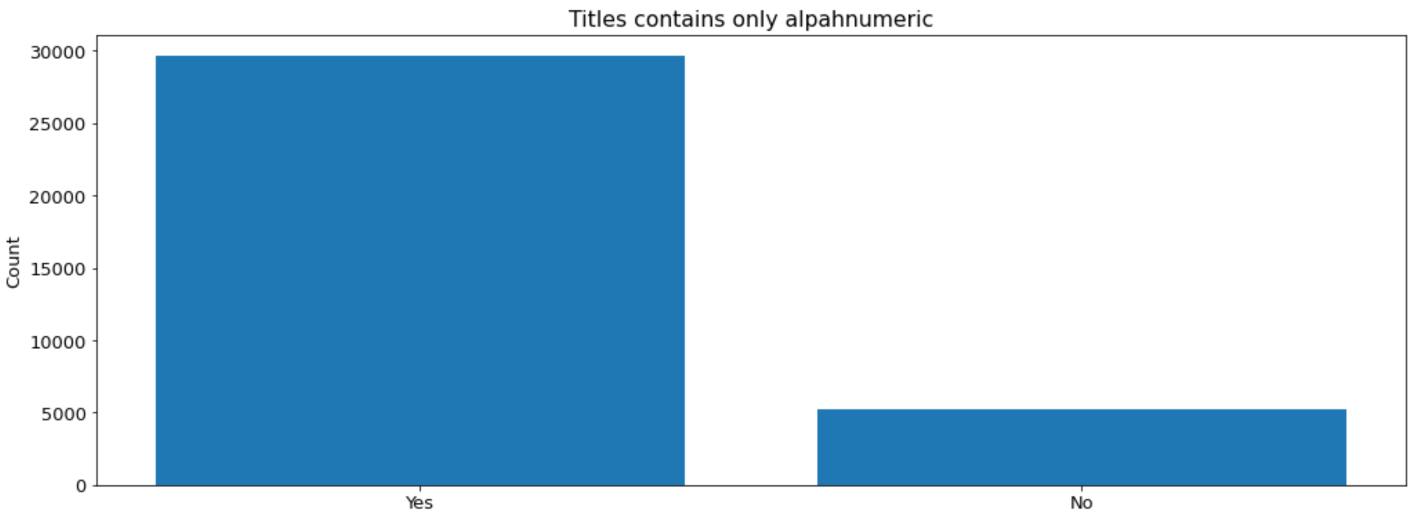
```
from matplotlib import pyplot as plt
plt.rcParams["figure.figsize"] = [17, 6]
plt.rcParams["font.size"] = 13
```

In [12]:

```
df['Title_length'].hist(bins=20)
plt.xlabel("Number of words")
plt.ylabel("Count")
plt.title("Length of Title Barplot")
plt.show()

df['Title'].str.len().hist(bins=100)
plt.xlabel("Number of characters")
plt.ylabel("Count")
plt.title("Length of title")
plt.show()

y = 0
for val in df["nonalpha_title"].values:
  if val == []:
    y += 1

plt.bar([0, 1], [y, len(df['Title'])-y], tick_label=["Yes", "No"])
plt.title("Titles contains only alpahnumeric")
plt.ylabel("Count")
plt.show()
```

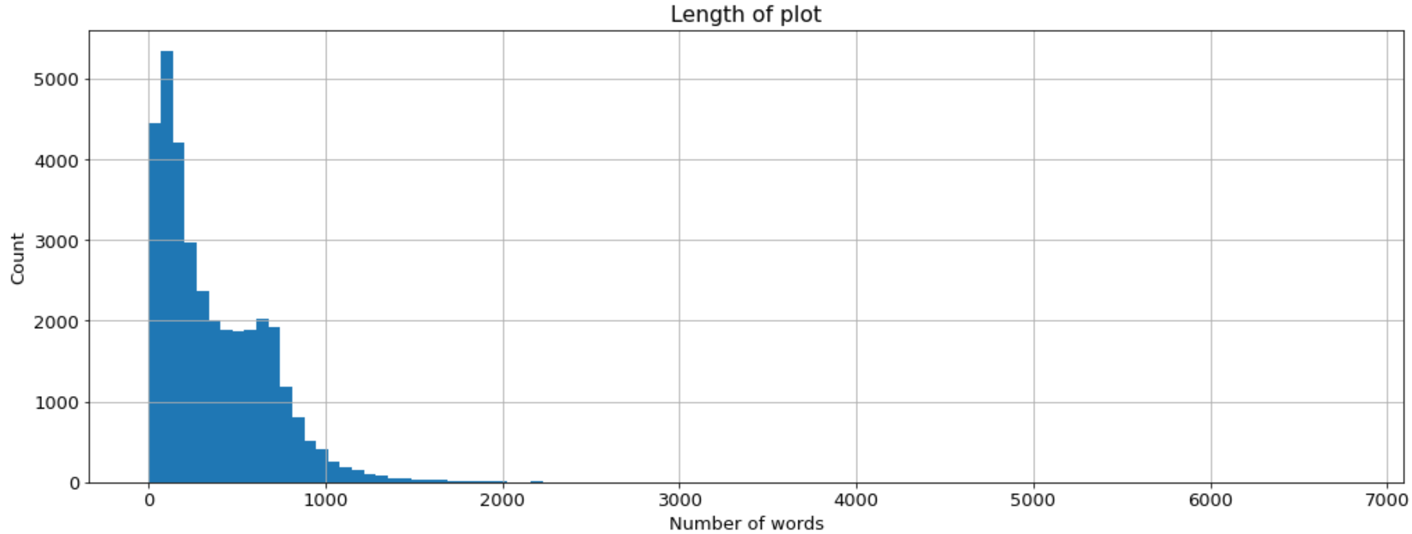


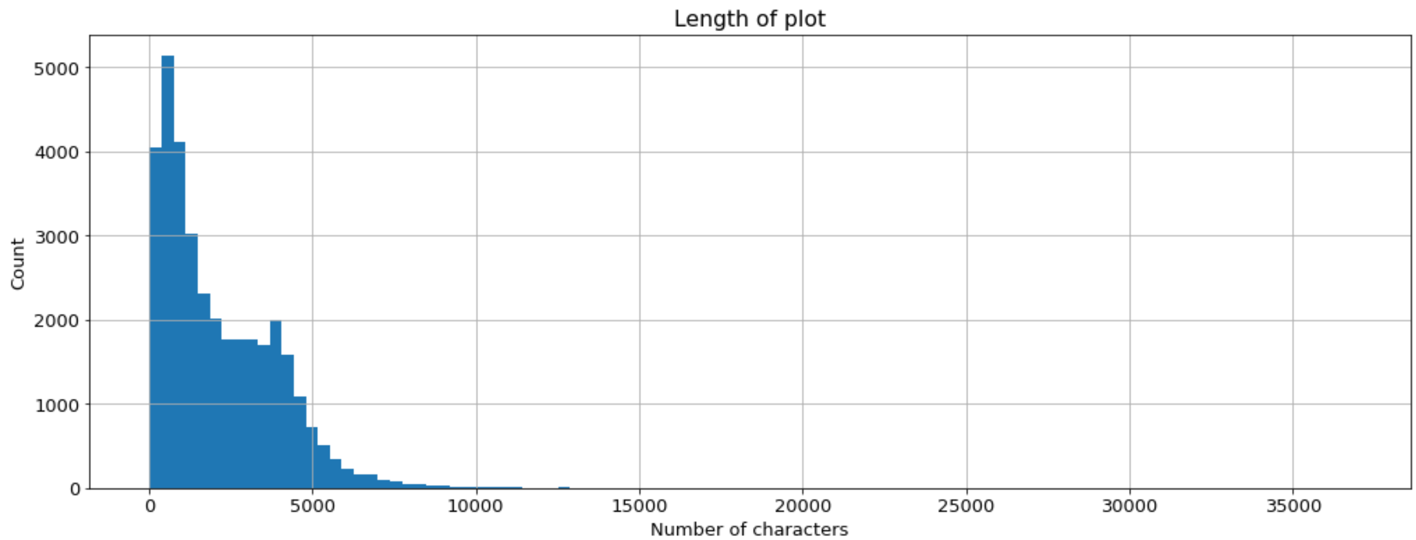


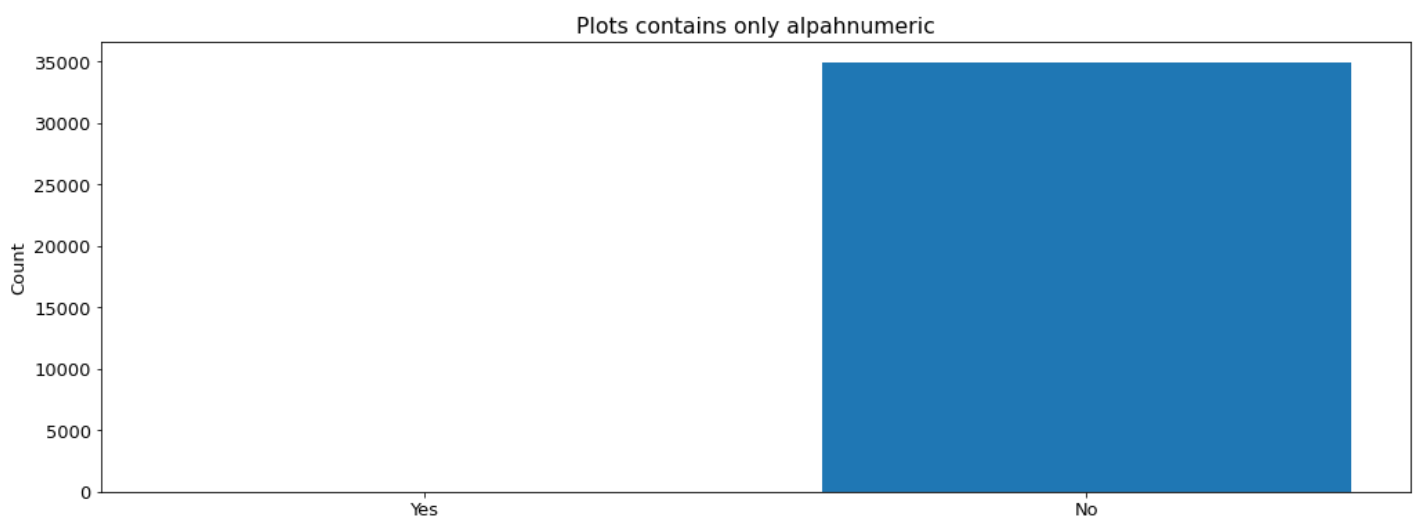


In [13]:

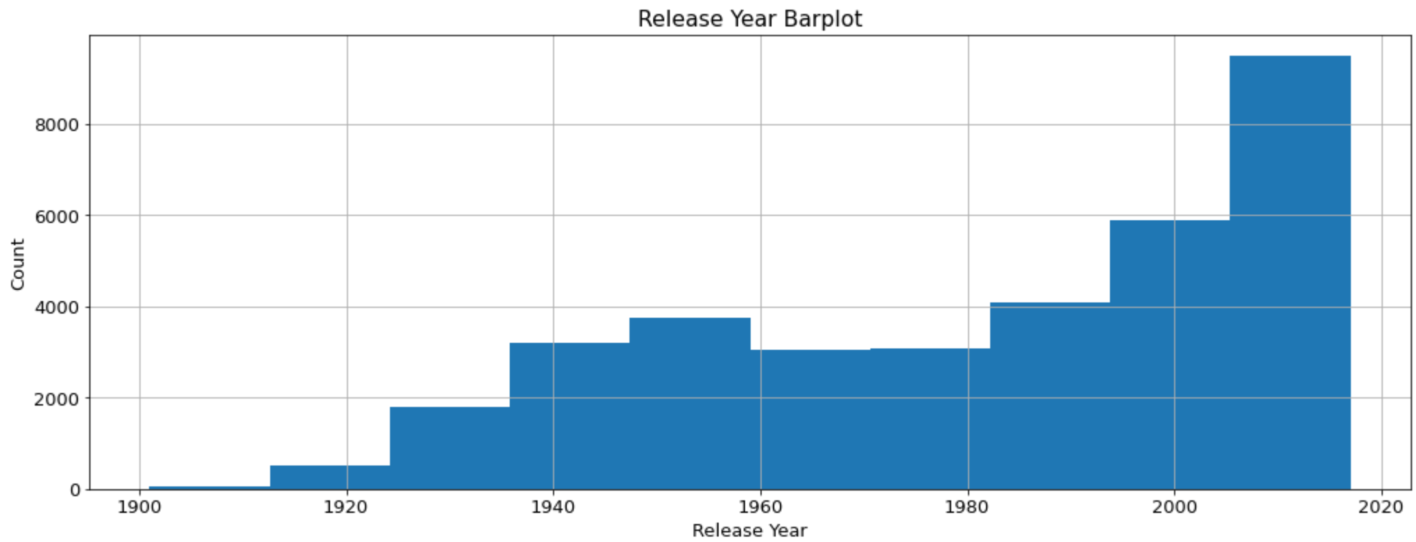
```
df['Plot_length'].hist(bins=100)
plt.xlabel("Number of words")
plt.ylabel("Count")
plt.title("Length of plot")
plt.show()

df['Plot'].str.len().hist(bins=100)
plt.xlabel("Number of characters")
plt.ylabel("Count")
plt.title("Length of plot")
plt.show()

y = 0
for val in df["nonalpha_plot"].values:
  if val == []:
    y += 1

plt.bar([0, 1], [y, len(df['Plot'])-y], tick_label=["Yes", "No"])
plt.title("Plots contains only alpahnumeric")
plt.ylabel("Count")
plt.show()
```





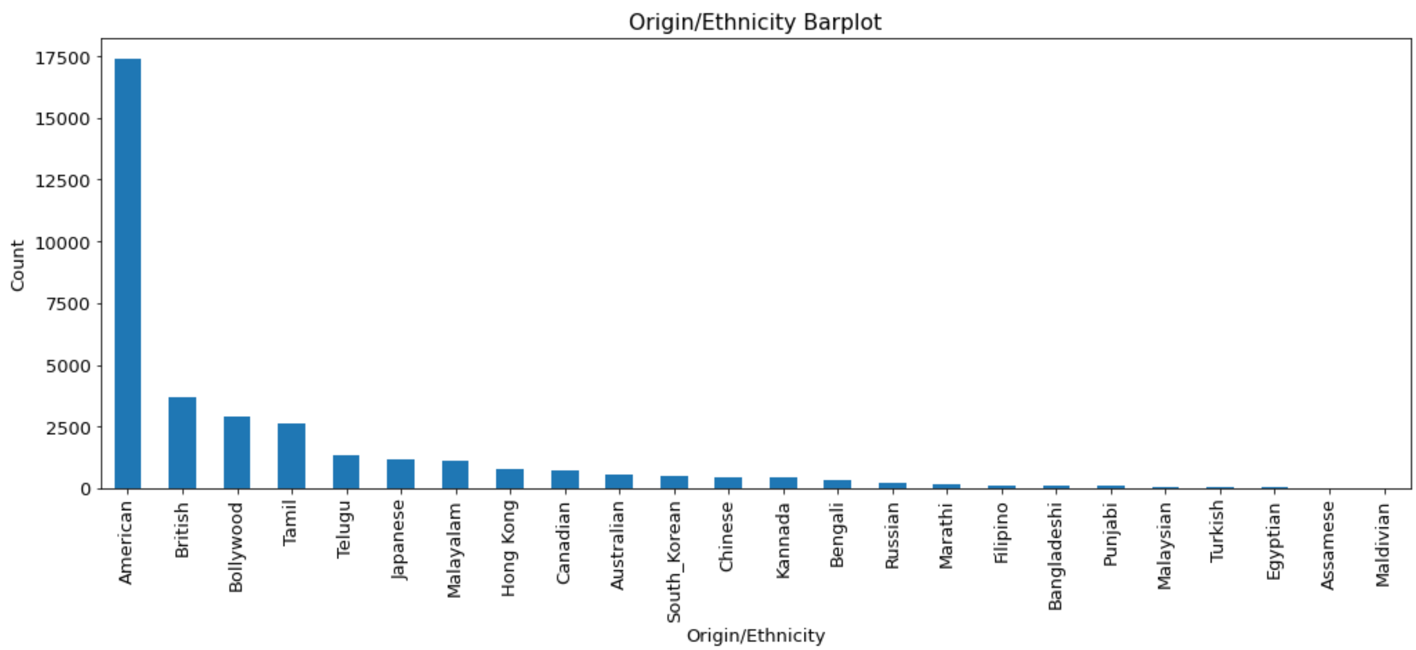


In [14]:
```
df['Release Year'].hist()
plt.xlabel("Release Year")
plt.ylabel("Count")
plt.title("Release Year Barplot")
plt.show()
```



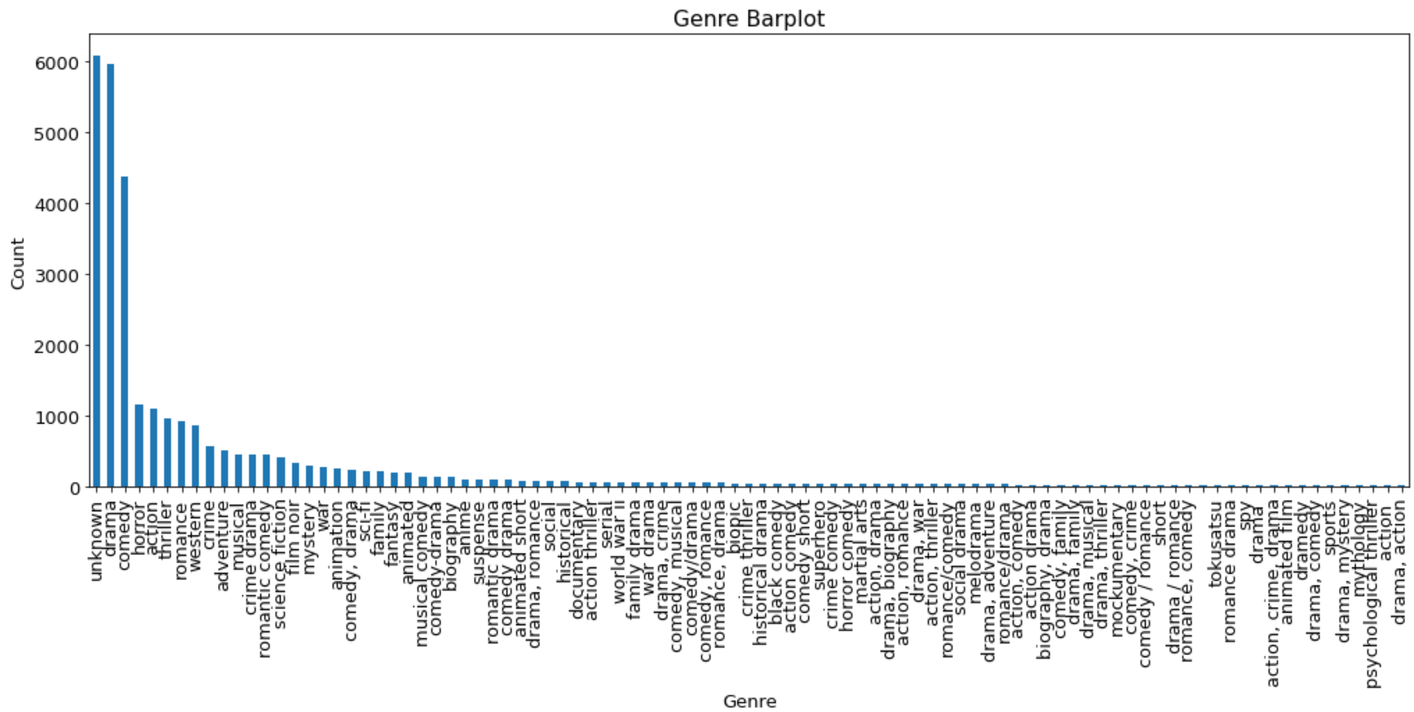
In [15]:

```
df['Origin/Ethnicity'].value_counts().plot.bar()
plt.xlabel("Origin/Ethnicity")
plt.ylabel("Count")
plt.title("Origin/Ethnicity Barplot")
plt.show()
```



In [16]:

```
vcounts = df['Genre'].value_counts()
vcounts[vcounts > 20].plot.bar()
plt.xlabel("Genre")
plt.ylabel("Count")
plt.title("Genre Barplot")
plt.show()
```



1.2 Data Quality Issues and Plan

Missing values - In many of the columns, the missing values is replaced with unknown. But, it is seen the cast column has 4% of null values which is less but still we will replace it with unknown.

Irregular Cardinality - The cardinality is neither 1 nor too high for any columns therefore there is no issue for irregular cardinality.

Outliers - Outliers in the categorical data can be detected by count of features. For example the count of origin, as the Assamism and Maldivian rarely exsists in the data. There are very few movies from 1902 to 1913 in the data. There are very few title with length 9 to 11 and almost does not exsists above length 12.

In [17]:
```
df.head(10)
```

Out[17]:

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length	nonal
0	1901	Kansas Saloon Smashers	American	Unknown	<NA>	unknown	A bartender is working at a saloon, serving dr...	3	83	[,, ., ', ,
1	1901	Love by the Light of the Moon	American	Unknown	<NA>	unknown	The moon, painted with a smiling face hangs ov...	7	86	[,, ., .,
2	1901	The Martyred Presidents	American	Unknown	<NA>	unknown	The film, just over a minute long, is composed...	3	76	[,, ,, ., —, ,, .

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length	nonal
3	1901	Terrible Teddy, the Grizzly King	American	Unknown	<NA>	unknown	Lasting just 61 seconds and consisting of two ...	5	153	[,, ., - ", ", "
4	1902	Jack and the Beanstalk	American	George S. Fleming, Edwin S. Porter	<NA>	unknown	The earliest known adaptation of the classic f...	4	140	[,, " " ', .
5	1903	Alice in Wonderland	American	Cecil Hepworth	May Clark	unknown	Alice follows a large white rabbit down a "Rab...	3	229	[", -, " " " .,
6	1903	The Great Train Robbery	American	Edwin S. Porter	<NA>	western	The film opens with two bandits breaking into ...	4	239	[,, ', ' ., " ;
7	1904	The Suburbanite	American	Wallace McCutcheon	<NA>	comedy	The film is about a family who move to the sub...	2	34	
8	1905	The Little Train Robbery	American	Edwin Stanton Porter	<NA>	unknown	The opening scene shows the interior of the ro...	4	646	[', ., ., ", ,
9	1905	The Night Before Christmas	American	Edwin Stanton Porter	<NA>	unknown	Scenes are introduced using lines of the poem....	4	83	[., [,], "

1.3 Data Preprocessing

We don't need to model the outliers with the other data. Every point in the data can be really essential at times, in which case we must identify or create a model that can also account for outliers and can handle even a very tiny subset of the data.

In [18]:

```
print(df.isnull().sum())
```

```
Release Year          0
Title                 0
Origin/Ethnicity      0
Director              0
Cast               1422
Genre                 0
Plot                  0
Title_length          0
Plot_length           0
```

```
nonalpha_plot          0
nonalpha_title         0
dtype: int64
```

In [19]:

```
print(df['Cast'])
print(df['Cast'].isnull().sum())
```

```
0                                                     <NA>
1                                                     <NA>
2                                                     <NA>
3                                                     <NA>
4                                                     <NA>
                               ...
34881    Director: Russell Crowe
Cast: Russell Crowe, ...
34882                            Ahmet Kural, Murat Cemcir
34883                Ata Demirer, Tuvana Türkay, Ülkü Duru
34884    YouTubers Shanna Malcolm, Shira Lazar, Sara Fl...
34885    Halit Ergenç, Tuba Büyüküstün, Mehmet Günsür, ...
Name: Cast, Length: 34886, dtype: string
1422
```

Filling missing values in cast with unknown

In [20]:

```
df['Cast'].fillna('Unknown', inplace=True)
```

In [21]:

```
print(df['Cast'])
print(df['Cast'].isnull().sum())
```

```
0                                                  Unknown
1                                                  Unknown
2                                                  Unknown
3                                                  Unknown
4                                                  Unknown
                               ...
34881    Director: Russell Crowe
Cast: Russell Crowe, ...
34882                            Ahmet Kural, Murat Cemcir
34883                Ata Demirer, Tuvana Türkay, Ülkü Duru
34884    YouTubers Shanna Malcolm, Shira Lazar, Sara Fl...
34885    Halit Ergenç, Tuba Büyüküstün, Mehmet Günsür, ...
Name: Cast, Length: 34886, dtype: string
0
```

1.4

What is the distribution of the top 50 most frequent words (excluding the stop words) for each of the textual features?

In [22]:

```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

stop_words = stopwords.words('english')

Title_SW = df['Title'].str.lower().apply(lambda x: ' '.join([word for word in x.split() if

Title_words_freq = Title_SW.str.split(expand=True).stack().value_counts().reset_index()

Title_words_freq.columns = ['Word', 'Frequency']
```

```
print("Top 50 most frequent words in the titles:")
Title_words_freq.head(50)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
Top 50 most frequent words in the titles:
```

Out[22]:

	Word	Frequency
0	man	456
1	love	425
2	night	281
3	2	227
4	girl	224
5	life	184
6	&	181
7	last	181
8	big	176
9	story	172
10	lady	169
11	woman	169
12	black	168
13	little	167
14	one	163
15	movie	161
16	mr.	152
17	time	150
18	house	147
19	men	142
20	day	135
21	city	134
22	three	130
23	dead	127
24	new	119
25	king	118
26	world	114
27	great	114
28	red	114
29	two	113
30	dark	111
31	death	109

	Word	Frequency
32	white	107
33	street	105
34	high	105
35	secret	103
36	war	100
37	thethe	100
38	young	99
39	wild	96
40	murder	95
41	blue	92
42	back	92
43	blood	92
44	ki	91
45	boy	91
46	ii	90
47	lost	90
48	star	89
49	heart	89

In [23]:

```
Plot_SW = df['Plot'].str.lower().apply(lambda x: ' '.join([word for word in x.split() if w

Plot_words_freq = Plot_SW.str.split(expand=True).stack().value_counts().reset_index()

Plot_words_freq.columns = ['Word', 'Frequency']
print("Top 50 most frequent words in the plots: ")
Plot_words_freq.head(50)
```

```
Top 50 most frequent words in the plots:
```

Out[23]:

	Word	Frequency
0	one	27307
1	tells	19607
2	back	19274
3	two	19244
4	him.	19095
5	also	16950
6	get	16740
7	new	16340
8	love	15767
9	finds	15759

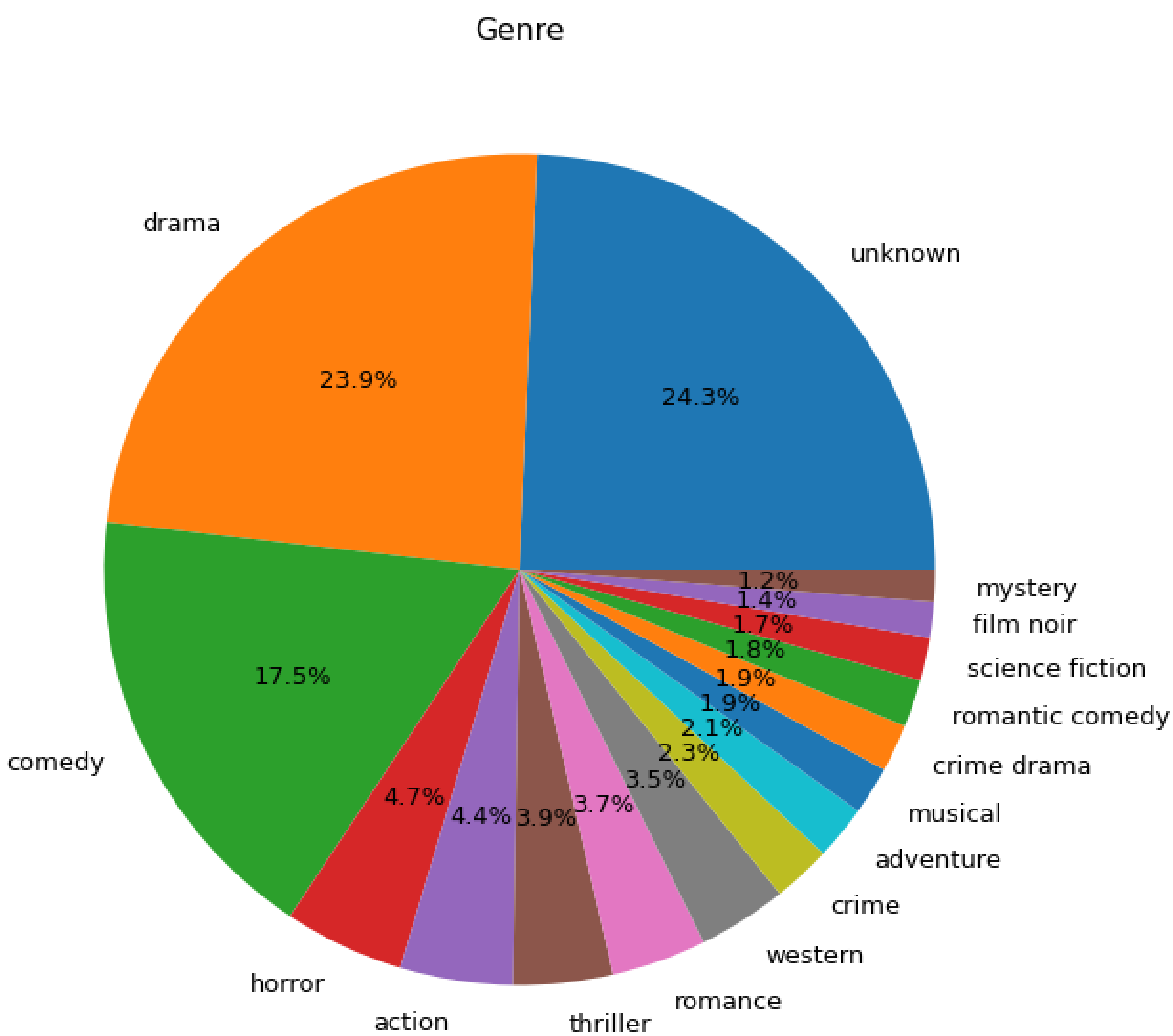
	Word	Frequency
10	find	14637
11	goes	14201
12	father	13916
13	takes	13865
14	police	13607
15	however,	13014
16	gets	12813
17	take	12758
18	man	12665
19	her.	12564
20	tries	12032
21	go	11929
22	family	11366
23	time	11003
24	film	10648
25	help	10619
26	comes	10277
27	becomes	10256
28	young	10255
29	him,	10179
30	decides	10140
31	kill	10042
32	home	10026
33	life	9930
34	next	9896
35	house	9618
36	mother	9535
37	first	9474
38	asks	9435
39	make	9283
40	later	9211
41	another	9118
42	meets	8952
43	wife	8910
44	killed	8798
45	away	8449

	Word	Frequency
46	returns	8383
47	begins	8296
48	son	8284
49	meanwhile,	8272

What is the proportion of each genre in the dataset?

In [24]:

```
plt.rcParams["figure.figsize"] = [10, 10]
vcounts = df['Genre'].value_counts()
vcounts = vcounts[vcounts > 300]
plt.pie(vcounts, labels = vcounts.index, autopct='%1.1f%%')

plt.title("Genre")
plt.show()
plt.rcParams["figure.figsize"] = [17, 6]
```



What is the most/least common origin of the movies?\ From the origin bar plot, the most common origin is America and the least common is Maldives.

What trends can you find in your data?\ Trends noticed in the data include increasing numbers of movies released each year. Additionally, the number of movies which are from America are significantly more than any other country. Drama and Comedy are the most popular genres by a large margin as well, excluding the unknown category.

2. Genres Selection and Understanding

Films which contain only 1 of the selected genres will belong to that class regardless if they have other genre tags. Films which contain multiple of the target genres will be sorted based on which genre is listed first, as this is likely the most relevant. These genres will be the targets for the model and other data points will be dropped.

In [25]:

```
import nltk
nltk.download('popular')
```

```
[nltk_data] Downloading collection 'popular'
[nltk_data]    |
[nltk_data]    | Downloading package cmudict to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/cmudict.zip.
[nltk_data]    | Downloading package gazetteers to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gazetteers.zip.
[nltk_data]    | Downloading package genesis to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/genesis.zip.
[nltk_data]    | Downloading package gutenberg to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/gutenberg.zip.
[nltk_data]    | Downloading package inaugural to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/inaugural.zip.
[nltk_data]    | Downloading package movie_reviews to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/movie_reviews.zip.
[nltk_data]    | Downloading package names to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/names.zip.
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/shakespeare.zip.
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package treebank to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/treebank.zip.
[nltk_data]    | Downloading package twitter_samples to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping corpora/twitter_samples.zip.
[nltk_data]    | Downloading package omw to /root/nltk_data...
[nltk_data]    | Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet to /root/nltk_data...
[nltk_data]    | Downloading package wordnet2021 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet31 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/wordnet_ic.zip.
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/words.zip.
[nltk_data]    | Downloading package maxent_ne_chunker to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping chunkers/maxent_ne_chunker.zip.
[nltk_data]    | Downloading package punkt to /root/nltk_data...
[nltk_data]    |   Unzipping tokenizers/punkt.zip.
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    | Downloading package averaged_perceptron_tagger to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data]    |
[nltk_data]  Done downloading collection popular
```

Out[25]:

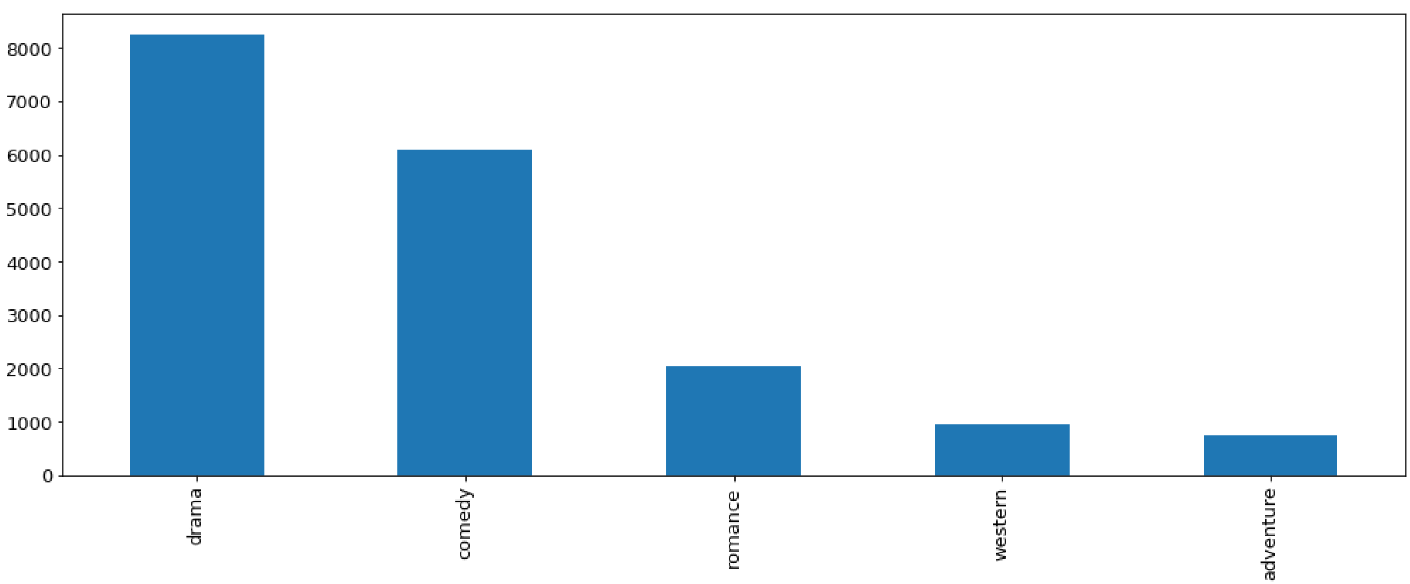
```
True
```

In [26]:
```
target_genres = ['drama', 'comedy', 'adventure', 'romance', 'western']

drop_inds = []

# split the genres into words
for index, row in df.iterrows():
  gen_list = nltk.tokenize.word_tokenize(row['Genre'])

  found = False
  # iter through listed genres
  for gen in gen_list:
    # check with each target
    if found: break

    # check for this synonym
    if gen == 'romantic':
        df["Genre"][index] = 'romance'
        #inds[target_genres.index('romance')].append(index)
        found = True
        break


    for test in target_genres:
      if gen == test:
        df["Genre"][index] = test
        #inds[target_genres.index(test)].append(index)
        found = True
        break

  if not found:
    # drop other genres
    drop_inds.append(index)
```

In [27]:
```
df_target = df.drop(drop_inds)
```

In [28]:
```
df_target['Genre'].value_counts().plot.bar()
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb439f61610>



In [29]:
```
df_target = df_target.reset_index()
```

In [30]:

```
top_1000s = []

# for each of the target genres
for gen in target_genres:
  # get the films in that genre
  genres_df =  df_target.loc[df_target['Genre'] == gen]
  print("Genre: ", gen)

  tokenized_titles = []
  tokenized_plots = []

  for index, row in genres_df.iterrows():
    word_tokens = nltk.tokenize.word_tokenize(row['Title'])
    tokenized_titles.append(word_tokens)

    word_tokens = nltk.tokenize.word_tokenize(row['Plot'])
    tokenized_plots.append(word_tokens)


  # list the titles and find most common word
  flat_titles = [item for sublist in tokenized_titles for item in sublist]
  f = nltk.probability.FreqDist(flat_titles)
  print("most frequent words in Titles:")
  print(f.most_common(10))

  # distribution of words in plot descrpition
  flat_plots = [item for sublist in tokenized_plots for item in sublist]
  f = nltk.probability.FreqDist(flat_plots)
  print("Word distrubutions in plots:")
  print(f.most_common(10))


  # vocab alignment
  temp = f.most_common(1000)
  most_common_words = [i[0] for i in temp]
  top_1000s.append(most_common_words)
```

```
Genre:  drama
most frequent words in Titles:
[('The', 1551), ('of', 682), ('the', 602), ("'s", 237), ('and', 226), ('in', 224), ('A', 1
92), ('(', 171), (')', 171), (',', 143)]
Word distrubutions in plots:
[(',', 173834), ('.', 144770), ('the', 140829), ('to', 108065), ('and', 97690), ('a', 7750
3), ('of', 53278), ('is', 50864), ('his', 46562), ('in', 44789)]
Genre:  comedy
most frequent words in Titles:
[('The', 966), ('the', 523), ('of', 369), ("'s", 303), ('and', 245), ('in', 231), ('a', 19
5), (',', 194), ('to', 166), (':', 134)]
Word distrubutions in plots:
[(',', 137457), ('the', 114002), ('.', 105702), ('to', 85388), ('and', 77056), ('a', 6191
1), ('of', 38063), ('is', 36223), ('his', 32939), ('in', 31749)]
Genre:  adventure
most frequent words in Titles:
[('The', 229), ('of', 164), ('the', 156), (':', 60), ('and', 32), ("'s", 29), ('Tarzan', 2
7), ('in', 21), ('Sea', 18), ('Jungle', 18)]
Word distrubutions in plots:
[(',', 19395), ('the', 19035), ('.', 14802), ('to', 11748), ('and', 10795), ('a', 7538),
('of', 5998), ('is', 4754), ('his', 4372), ('in', 4096)]
Genre:  romance
most frequent words in Titles:
[('The', 157), ('Love', 103), ('of', 73), ('the', 67), ('(', 50), (')', 50), ('in', 49),
("'s", 45), ('You', 41), ('and', 38)]
Word distrubutions in plots:
[(',', 39421), ('.', 37833), ('to', 28754), ('the', 28692), ('and', 26091), ('a', 18398),
('her', 14516), ('is', 13362), ('in', 11342), ('of', 11266)]
```

```
Genre:  western
most frequent words in Titles:
[('The', 257), ('the', 121), ('of', 104), ('Man', 31), ('Gun', 31), ('and', 31), ('to', 2
6), ("'s", 25), ('Kid', 22), ('West', 21)]
Word distrubutions in plots:
[(',', 17957), ('the', 15871), ('.', 14411), ('to', 10614), ('and', 9795), ('a', 7179),
('of', 5128), ('is', 4800), ('his', 4399), ('in', 4009)]
```

In [31]:

```
import numpy as np
result = np.ones((5, 5))
for i in range(5):
  test = top_1000s[i]
  for j in range(5):

    test2 =  top_1000s[j]

    res = len(list(set(test).intersection(test2)))
    result[i, j] = res/10

print("Matrix showing vocabulary alignment between genres")
print(target_genres)
print(result)
```

```
Matrix showing vocabulary alignment between genres
['drama', 'comedy', 'adventure', 'romance', 'western']
[[100.   85.6  71.4  81.5  67.2]
 [ 85.6 100.   69.5  79.2  65.3]
 [ 71.4  69.5 100.   63.5  66.5]
 [ 81.5  79.2  63.5 100.   60. ]
 [ 67.2  65.3  66.5  60.  100. ]]
```

3. Text Normalization and Feature Engineering

We are doing text normalization and feature engineering to plot column as it is the only column in dataset which has sentences and contains the most relevant information for our task. We have decided to focus on this section of the corpus only as the plot description directly relates to the target, while other features do not.

Lemmatization was chosen because speed is not a critical concern, the actual word matters in this case and the context matters.

We have already removed stopwords for plot in previous step.

In [32]:

```
Plot_SW
```

Out[32]:

```
0        bartender working saloon, serving drinks custo...
1        moon, painted smiling face hangs park night. y...
2        film, minute long, composed two shots. first, ...
3        lasting 61 seconds consisting two shots, first...
4        earliest known adaptation classic fairytale, f...
                               ...
34881    film begins 1919, world war ended, centres aro...
34882    two musicians, salih gürkan, described adventu...
34883    zafer, sailor living mother döndü coastal vill...
34884    film centres around young woman named amy tyle...
34885    writer orhan şahin returns iştanbul many year...
Name: Plot, Length: 34886, dtype: object
```

We are going to remove all the characters except alphabeticals. Therefore if we replace all the non alphabeticals with a space we are going to get our correct output instead of apostrophe's. Because if we replace apostrophe's with a space, we are getting highest word frequency of letter s. Therefore first we will replace apostrophe's without having any space and then rest with space.

In [33]:

```
Update_Plot = Plot_SW.str.replace("'", "")
df['New_Plot'] = Update_Plot.str.replace('[^a-zA-Z]', ' ')
df['New_Plot']
```

```
<ipython-input-33-7f87c6bd46a1>:2: FutureWarning: The default value of regex will change f
rom True to False in a future version.
  df['New_Plot'] = Update_Plot.str.replace('[^a-zA-Z]', ' ')
```

Out[33]:

```
0        bartender working saloon  serving drinks custo...
1        moon  painted smiling face hangs park night  y...
2        film  minute long  composed two shots  first  ...
3        lasting    seconds consisting two shots  first...
4        earliest known adaptation classic fairytale  f...
                               ...
34881    film begins       world war ended  centres aro...
34882    two musicians  salih g rkan  described adventu...
34883    zafer  sailor living mother d nd  coastal vill...
34884    film centres around young woman named amy tyle...
34885    writer orhan  ahin returns i stanbul many year...
Name: New_Plot, Length: 34886, dtype: object
```

In [34]:

```
New_Plot_Freq = df['New_Plot'].str.split(expand=True).stack().value_counts().reset_index()
New_Plot_Freq.columns = ['Word', 'Frequency']
New_Plot_Freq.head(50)
```

Out[34]:

	Word	Frequency
0	him	30261
1	one	29889
2	back	22486
3	father	20707
4	her	20434
5	two	20297
6	love	19616
7	tells	19615
8	home	17608
9	also	17539
10	man	17425
11	time	17333
12	later	17235
13	house	16967
14	get	16907
15	new	16700
16	police	16626
17	life	16616
18	family	16584
19	finds	15800
20	day	14849

	Word	Frequency
21	find	14758
22	however	14463
23	goes	14339
24	mother	14022
25	takes	13922
26	s	13619
27	wife	13453
28	go	13215
29	film	13210
30	take	12947
31	son	12892
32	help	12889
33	gets	12849
34	away	12764
35	tries	12079
36	money	11973
37	night	11658
38	killed	11331
39	first	11180
40	death	11168
41	young	11134
42	daughter	11107
43	friend	10987
44	men	10430
45	comes	10415
46	next	10318
47	friends	10279
48	becomes	10276
49	kill	10259

In [35]:

```
import nltk
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer

# Create WordNetLemmatizer object
wnl = WordNetLemmatizer()

df['Lem_Plot'] = df['New_Plot'].apply(lambda x: ' '.join([wnl.lemmatize(word,'v') for word
```

```
print(df['Lem_Plot'])
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
0        bartender work saloon serve drink customers fi...
1        moon paint smile face hang park night young co...
2        film minute long compose two shots first girl ...
3        last second consist two shots first shoot set ...
4        earliest know adaptation classic fairytale fil...
                              ...
34881    film begin world war end centre around joshua ...
34882    two musicians salih g rkan describe adventure ...
34883    zafer sailor live mother d nd coastal village ...
34884    film centre around young woman name amy tyler ...
34885    writer orhan ahin return i stanbul many years ...
Name: Lem_Plot, Length: 34886, dtype: object
```

In [36]:

```
Lem_Plot_Freq = df['Lem_Plot'].str.split(expand=True).stack().value_counts().reset_index()
Lem_Plot_Freq.columns = ['Word', 'Frequency']
Lem_Plot_Freq.head(50)
```

Out[36]:

	Word	Frequency
0	find	37697
1	take	37079
2	get	35651
3	go	35633
4	kill	33850
5	leave	30945
6	him	30261
7	one	29889
8	tell	29721
9	love	24393
10	make	24207
11	father	23969
12	back	22875
13	try	21746
14	see	21625
15	meet	21184
16	return	20743
17	come	20585
18	her	20434
19	become	20396
20	two	20297
21	time	19102

	Word	Frequency
22	give	18876
23	man	18474
24	home	17923
25	also	17539
26	house	17517
27	help	17314
28	later	17235
29	new	16700
30	police	16681
31	life	16616
32	family	16584
33	marry	16530
34	work	15592
35	mother	15576
36	reveal	15182
37	decide	15139
38	live	15064
39	end	14885
40	film	14865
41	day	14849
42	use	14598
43	however	14463
44	fall	14433
45	know	14376
46	call	13915
47	ask	13902
48	begin	13813
49	name	13681

References:

1. https://medium.com/@ashwinnaidu1991/creating-a-tf-idf-model-from-scratch-in-python-71047f16494e
2. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/?utm_source=blog&utm_medium=predicting-movie-genres-nlp-multi-label-classification
3. https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/?utm_source=blog&utm_medium=predicting-movie-genres-nlp-multi-label-classification

In [37]:

```
df.head(10)
```

Out[37]:

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length	no
0	1901	Kansas Saloon Smashers	American	Unknown	Unknown	unknown	A bartender is working at a saloon, serving dr...	3	83	[,,
1	1901	Love by the Light of the Moon	American	Unknown	Unknown	unknown	The moon, painted with a smiling face hangs ov...	7	86	[,,
2	1901	The Martyred Presidents	American	Unknown	Unknown	unknown	The film, just over a minute long, is composed...	3	76	[,, —
3	1901	Terrible Teddy, the Grizzly King	American	Unknown	Unknown	unknown	Lasting just 61 seconds and consisting of two ...	5	153	[, ",
4	1902	Jack and the Beanstalk	American	George S. Fleming, Edwin S. Porter	Unknown	unknown	The earliest known adaptation of the classic f...	4	140	[,,
5	1903	Alice in Wonderland	American	Cecil Hepworth	May Clark	unknown	Alice follows a large white rabbit down a "Rab...	3	229	[", ,,
6	1903	The Great Train Robbery	American	Edwin S. Porter	Unknown	western	The film opens with two bandits breaking into ...	4	239	[, .
7	1904	The Suburbanite	American	Wallace McCutcheon	Unknown	comedy	The film is about a family who move to the sub...	2	34	
8	1905	The Little Train Robbery	American	Edwin Stanton Porter	Unknown	unknown	The opening scene shows the interior of the ro...	4	646	[', .

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length	no
9	1905	The Night Before Christmas	American	Edwin Stanton Porter	Unknown	unknown	Scenes are introduced using lines of the poem....	4	83	[., [

4. Model Building - Genre Prediction Model

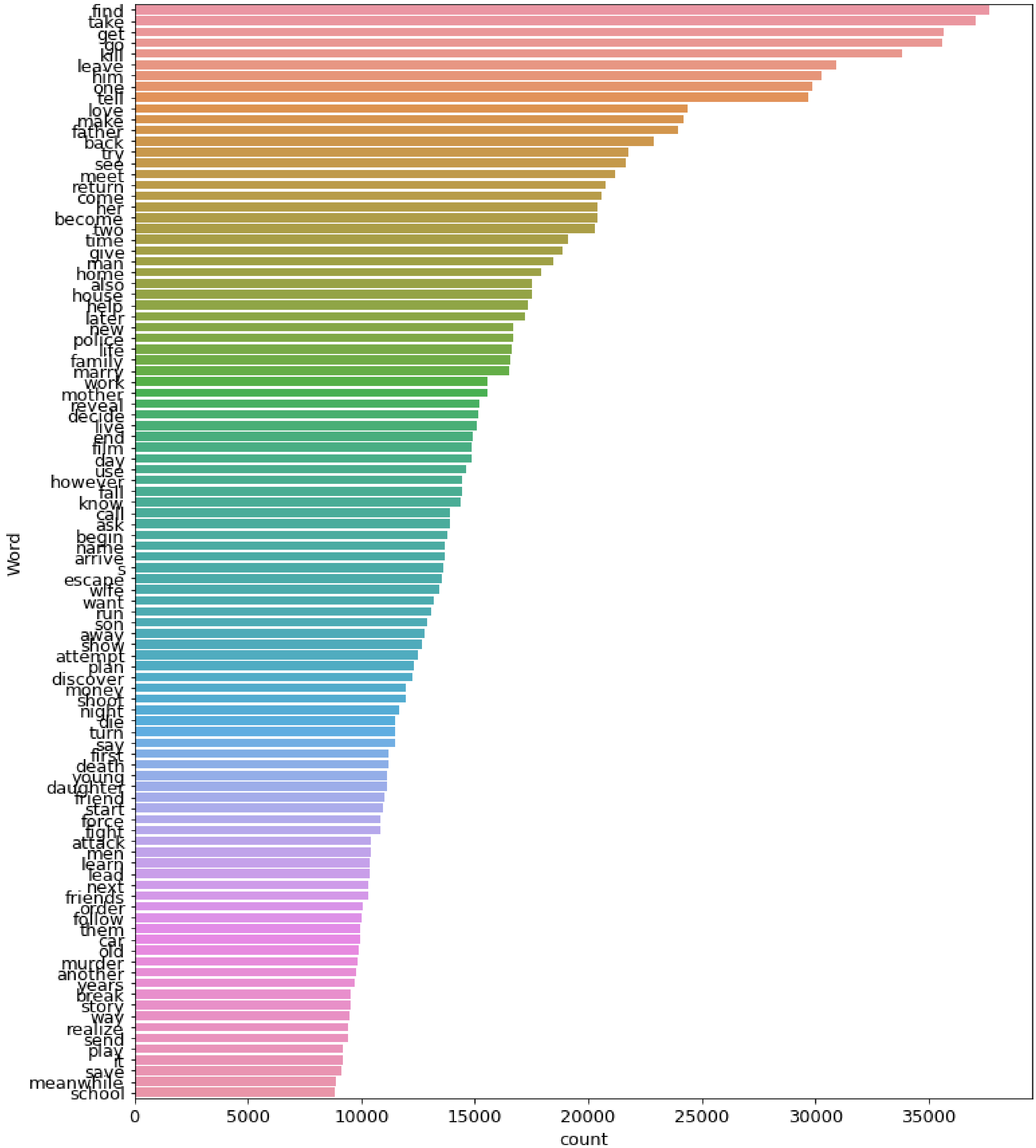
4.1 Problem Definition

Our task is to create a tool which will output a specific label of genre from the plot information so this a supervised classification problem.

Reference: https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/

In [41]:

```
import seaborn as sns
def freq_words(x, terms = 30):
  all_words = ' '.join([text for text in x])
  all_words = all_words.split()
  fdist = nltk.FreqDist(all_words)
  words_df = pd.DataFrame({'word':list(fdist.keys()), 'count':list(fdist.values())})

  # selecting top 20 most frequent words
  d = words_df.nlargest(columns="count", n = terms)

  # visualize words and frequencies
  plt.figure(figsize=(12,15))
  ax = sns.barplot(data=d, x= "count", y = "word")
  ax.set(ylabel = 'Word')
  plt.show()

# print 100 most frequent words
freq_words(df['Lem_Plot'], 100)
```



In [42]:

```
x = df.Genre.value_counts().head(10).reset_index().rename({'index':'genre'})
x
```

Out[42]:

	index	Genre
0	drama	8239
1	comedy	6099
2	unknown	6083
3	romance	2037
4	horror	1167

	index	Genre
5	action	1098
6	thriller	966
7	western	944
8	adventure	738
9	crime	568

In [43]:

```
list1 = list(x['index'].unique())
list1
```

Out[43]:

```
['drama',
 'comedy',
 'unknown',
 'romance',
 'horror',
 'action',
 'thriller',
 'western',
 'adventure',
 'crime']
```

In [44]:

```
new_df = df[df['Genre'].isin(list1)]
new_df
```

Out[44]:

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length
0	1901	Kansas Saloon Smashers	American	Unknown	Unknown	unknown	A bartender is working at a saloon, serving dr...	3	83
1	1901	Love by the Light of the Moon	American	Unknown	Unknown	unknown	The moon, painted with a smiling face hangs ov...	7	86
2	1901	The Martyred Presidents	American	Unknown	Unknown	unknown	The film, just over a minute long, is composed...	3	76
3	1901	Terrible Teddy, the Grizzly King	American	Unknown	Unknown	unknown	Lasting just 61 seconds and consisting of two ...	5	153

	Release Year	Title	Origin/Ethnicity	Director	Cast	Genre	Plot	Title_length	Plot_length
4	1902	Jack and the Beanstalk	American	George S. Fleming, Edwin S. Porter	Unknown	unknown	The earliest known adaptation of the classic f...	4	140
...	...	...	...	...	...	...	...	...	...
34881	2014	The Water Diviner	Turkish	Director: Russell Crowe	Director: Russell Crowe Cast: Russell Crowe, ...	unknown	The film begins in 1919, just after World War ...	3	591
34882	2017	Çalgı Çengi İkimiz	Turkish	Selçuk Aydemir	Ahmet Kural, Murat Cemcir	comedy	Two musicians, Salih and Gürkan, described the...	3	11
34883	2017	Olanlar Oldu	Turkish	Hakan Algül	Ata Demirer, Tuvana Türkay, Ülkü Duru	comedy	Zafer, a sailor living with his mother Döndü i...	2	67
34884	2017	Non-Transferable	Turkish	Brendan Bradley	YouTubers Shanna Malcolm, Shira Lazar, Sara Fl...	romance	The film centres around a young woman named Am...	1	193
34885	2017	İstanbul Kırmızısı	Turkish	Ferzan Özpetek	Halit Ergenç, Tuba Büyüküstün, Mehmet Günsür, ...	romance	The writer Orhan Şahin returns to İstanbul aft...	2	48

27939 rows × 13 columns

4.2 Feature Selection

In [38]:
```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
```

In [39]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=20000)
```

In [45]:
```
l = tfidf_vectorizer.fit_transform(new_df['Lem_Plot'])
```

In [46]:
```
k = pd.DataFrame.sparse.from_spmatrix(l)
```

In [ ]:

```
# source : tutorial

# select the best features using mutual information

K_features = 10000

ft_scorer = SelectKBest(mutual_info_classif, k=K_features)
X = ft_scorer.fit_transform(l, new_df["Genre"])

#print("The input and the target matrix shapes:", X.shape, y.shape)
```

In [48]:

```
selected_feats = pd.Series(ft_scorer.scores_*1000, index=k.columns).sort_values(ascending=
```

4.3 Evaluation Metric

The evaluation metric which was chosen was overall accuracy of prediction as accuracy for each genre is equally valuable for our business problem.

4.4 Hyperparameter Tuning

Not applicable for logistic regression

4.5 Training and Evaluation

In [49]:

```
new_df.Genre.nunique()
```

Out[49]:

```
10
```

In [50]:

```
y = pd.get_dummies(new_df['Genre'])
y
```

Out[50]:

	action	adventure	comedy	crime	drama	horror	romance	thriller	unknown	western
0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...
34881	0	0	0	0	0	0	0	0	1	0
34882	0	0	1	0	0	0	0	0	0	0
34883	0	0	1	0	0	0	0	0	0	0
34884	0	0	0	0	0	0	1	0	0	0
34885	0	0	0	0	0	0	1	0	0	0

27939 rows × 10 columns

In [51]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=10000)
```

In [52]:
```
from sklearn.model_selection import train_test_split
xtrain, xval, ytrain, yval = train_test_split(new_df['Lem_Plot'], y, test_size=0.2, randor
```

In [53]:
```
xtrain_tfidf = tfidf_vectorizer.fit_transform(xtrain)
xval_tfidf = tfidf_vectorizer.transform(xval)
```

In [54]:
```
# NOTE: we acheived better accuracy without using the selected features based on mutual in
# left out in the final run

#xtrain_tfidf = xtrain_tfidf[:, selected_feats]
#xval_tfidf = xval_tfidf[:, selected_feats]
```

In [55]:
```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")

# Binary Relevance
from sklearn.multiclass import OneVsRestClassifier

# Performance metric
from sklearn.metrics import f1_score

lr = LogisticRegression(max_iter=10000)
clf = OneVsRestClassifier(lr)

# fit model on train data
clf.fit(xtrain_tfidf, ytrain)

# make predictions for validation set
y_pred = clf.predict(xval_tfidf)

y_pred
```

Out[55]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [56]:
```
list1
```

Out[56]:
```
['drama',
 'comedy',
 'unknown',
 'romance',
 'horror',
 'action',
 'thriller',
 'western',
 'adventure',
 'crime']
```

In [57]:
```
list2 = ['action',       'adventure',     'comedy',        'crime',         'drama',         'h
def replace(row):
```

```
    return list2[row]

y_in = pd.DataFrame(y_pred).idxmax(axis=1).reset_index(drop = True)
y_act = y_in.apply(replace)
y_act
```

Out[57]:

```
0       action
1       action
2       action
3        drama
4       action
         ...
5583    action
5584    action
5585    action
5586    action
5587    action
Length: 5588, dtype: object
```

In [58]:

```
yval
```

Out[58]:

	action	adventure	comedy	crime	drama	horror	romance	thriller	unknown	western
9793	0	0	0	0	1	0	0	0	0	0
21287	0	0	0	0	0	0	0	0	1	0
8145	0	0	1	0	0	0	0	0	0	0
20921	0	0	0	0	1	0	0	0	0	0
8116	0	0	0	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
700	0	0	0	0	0	0	1	0	0	0
6452	0	0	0	0	1	0	0	0	0	0
32539	0	0	0	0	0	0	1	0	0	0
16166	0	0	0	0	1	0	0	0	0	0
32495	1	0	0	0	0	0	0	0	0	0

5588 rows × 10 columns

In [59]:

```
y_true = yval.idxmax(axis=1)
y_true
```

Out[59]:

```
9793       drama
21287    unknown
8145      comedy
20921      drama
8116      horror
          ...
700      romance
6452       drama
32539    romance
16166      drama
32495     action
Length: 5588, dtype: object
```

In [60]:

```
from sklearn.metrics import classification_report
```

```
print(classification_report(y_true,y_act))
```

```
              precision    recall  f1-score   support

      action       0.06      0.86      0.11       236
   adventure       0.70      0.04      0.08       164
      comedy       0.78      0.35      0.48      1203
       crime       1.00      0.01      0.02        97
       drama       0.63      0.28      0.39      1637
      horror       0.87      0.16      0.27       245
     romance       0.62      0.05      0.10       377
    thriller       0.00      0.00      0.00       211
     unknown       0.75      0.36      0.48      1226
     western       0.87      0.32      0.47       192

    accuracy                           0.29      5588
   macro avg       0.63      0.24      0.24      5588
weighted avg       0.67      0.29      0.37      5588
```

In [61]:
```
for i in range(10):
  print("Movie: ", new_df['Title'][i], "\nPredicted genre: ", y_act[i]), print("Actual ger
```

```
Movie:  Kansas Saloon Smashers
Predicted genre:  action
Actual genre:  unknown

Movie:  Love by the Light of the Moon
Predicted genre:  action
Actual genre:  unknown

Movie:  The Martyred Presidents
Predicted genre:  action
Actual genre:  unknown

Movie:  Terrible Teddy, the Grizzly King
Predicted genre:  drama
Actual genre:  unknown

Movie:  Jack and the Beanstalk
Predicted genre:  action
Actual genre:  unknown

Movie:  Alice in Wonderland
Predicted genre:  action
Actual genre:  unknown

Movie:  The Great Train Robbery
Predicted genre:  action
Actual genre:  western

Movie:  The Suburbanite
Predicted genre:  action
Actual genre:  comedy

Movie:  The Little Train Robbery
Predicted genre:  action
Actual genre:  unknown

Movie:  The Night Before Christmas
Predicted genre:  action
Actual genre:  unknown
```

In [62]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_true, y_act)
```

Out[62]:

```
array([[ 203,    0,    3,    0,    7,    0,    1,    0,   22,    0],
       [ 140,    7,    0,    0,   13,    0,    0,    0,    2,    2],
       [ 681,    0,  420,    0,   67,    1,    5,    0,   23,    6],
       [  83,    0,    1,    1,   11,    0,    0,    0,    1,    0],
       [1051,    2,   57,    0,  460,    0,    2,    0,   65,    0],
       [ 192,    0,    3,    0,    6,   39,    0,    0,    5,    0],
       [ 273,    0,   26,    0,   36,    0,   20,    0,   22,    0],
       [ 180,    0,    3,    0,   20,    2,    0,    0,    6,    0],
       [ 653,    1,   28,    0,  100,    3,    4,    0,  436,    1],
       [ 126,    0,    0,    0,    5,    0,    0,    0,    0,   61]])
```

4.6 Preventing Overfitting

Overfitting can be prevented in this model by utilizing cross validation during training, as shown below in the learning curves. The utilization of different data subsets will help stop overfitting. The validation set accuracy should not begin to drop during training or overfitting likely has occured.

4.7 Plot Learning Curve

In [63]:

```
# learning curve
# code soure tutorial
import numpy as np
from sklearn.model_selection import learning_curve

def plot_learning_curve(
    estimator,
    title,
    X,
    y,
    axes=None,
    ylim=None,
    cv=None,
    n_jobs=None,
    train_sizes=np.linspace(0.1, 1.0, 5),
):

    _, axes = plt.subplots(1, 3, figsize=(20, 5))

    axes[0].set_title(title)
    if ylim is not None:
        axes[0].set_ylim(*ylim)
    axes[0].set_xlabel("Training examples")
    axes[0].set_ylabel("Score")

    train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
        estimator,
        X,
        y,
        cv=cv,
        n_jobs=n_jobs,
        train_sizes=train_sizes,
        return_times=True,
        scoring="accuracy",
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    fit_times_mean = np.mean(fit_times, axis=1)
    fit_times_std = np.std(fit_times, axis=1)
```

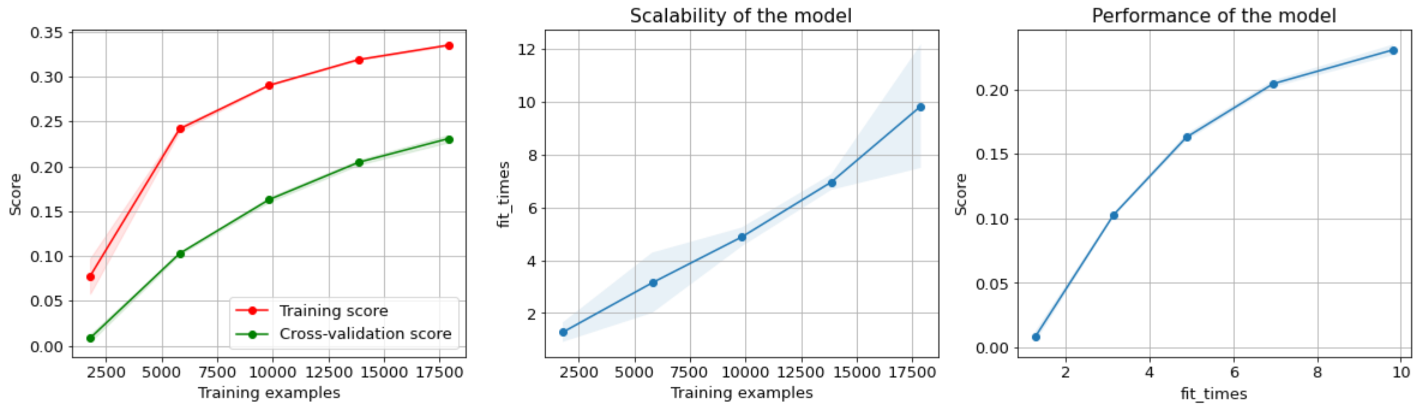
```python
    # Plot learning curve
    axes[0].grid()
    axes[0].fill_between(
        train_sizes,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.1,
        color="r",
    )
    axes[0].fill_between(
        train_sizes,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.1,
        color="g",
    )
    axes[0].plot(
        train_sizes, train_scores_mean, "o-", color="r", label="Training score"
    )
    axes[0].plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Cross-validation score"
    )
    axes[0].legend(loc="best")

    # Plot n_samples vs fit_times
    axes[1].grid()
    axes[1].plot(train_sizes, fit_times_mean, "o-")
    axes[1].fill_between(
        train_sizes,
        fit_times_mean - fit_times_std,
        fit_times_mean + fit_times_std,
        alpha=0.1,
    )
    axes[1].set_xlabel("Training examples")
    axes[1].set_ylabel("fit_times")
    axes[1].set_title("Scalability of the model")

    # Plot fit_time vs score
    fit_time_argsort = fit_times_mean.argsort()
    fit_time_sorted = fit_times_mean[fit_time_argsort]
    test_scores_mean_sorted = test_scores_mean[fit_time_argsort]
    test_scores_std_sorted = test_scores_std[fit_time_argsort]
    axes[2].grid()
    axes[2].plot(fit_time_sorted, test_scores_mean_sorted, "o-")
    axes[2].fill_between(
        fit_time_sorted,
        test_scores_mean_sorted - test_scores_std_sorted,
        test_scores_mean_sorted + test_scores_std_sorted,
        alpha=0.1,
    )
    axes[2].set_xlabel("fit_times")
    axes[2].set_ylabel("Score")
    axes[2].set_title("Performance of the model")
    return plt
```

In [64]:

```python
plot_learning_curve(clf, "", xtrain_tfidf, ytrain,  n_jobs=-1)
```

Out[64]:

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.8/dist-packages/matplotlib/pyplo
t.py'>
```



4.8 Analyze Results

The overall accuracy of the model is 29%. Genres such as thriller and crime have almost 0% accuracy. The model does not seem to be overfit as validation accuracy did not drop. The confusion matrix shows the most often predicted genre is Action.

5. Applying Part-of-speech Tagging

In [65]:

```
# source: https://stackoverflow.com/questions/33587667/extracting-all-nouns-from-a-text-fi

# function to test if something is a noun
is_noun = lambda pos: pos[:2] == 'NN'

xtrain_nouns = []
for string in xtrain:
  tokens=nltk.word_tokenize(string)
  nouns = [word for (word, pos) in nltk.pos_tag(tokens) if is_noun(pos)]
  xtrain_nouns.append(' '.join(nouns))
```

In [66]:

```
xval_nouns = []
for string in xval:
  tokens=nltk.word_tokenize(string)
  nouns = [word for (word, pos) in nltk.pos_tag(tokens) if is_noun(pos)]
  xval_nouns.append(' '.join(nouns))
```

In [67]:

```
xtrain_nouns_tfidf = tfidf_vectorizer.fit_transform(xtrain_nouns)
xval_nouns_tfidf = tfidf_vectorizer.transform(xval_nouns)
```

In [68]:

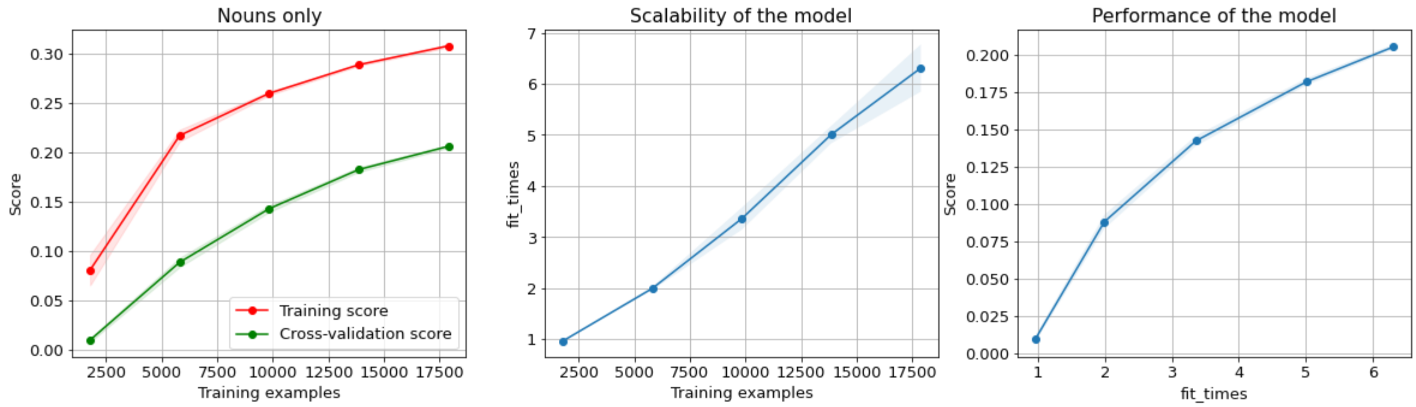
```
lr = LogisticRegression(max_iter=10000)
clf2 = OneVsRestClassifier(lr)

# fit model on train data
clf2.fit(xtrain_nouns_tfidf, ytrain)

# make predictions for validation set
y_pred = clf2.predict(xval_nouns_tfidf)
```

In [69]:

```
plot_learning_curve(clf, "Nouns only", xtrain_nouns_tfidf, ytrain,  n_jobs=-1)
```

Out[69]:

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.8/dist-packages/matplotlib/pyplo
t.py'>
```



In [70]:
```
y_in = pd.DataFrame(y_pred).idxmax(axis=1).reset_index(drop = True)
y_act = y_in.apply(replace)
```

In [71]:
```
y_true = yval.idxmax(axis=1)
```

In [72]:
```
print(classification_report(y_true,y_act))
```

```
              precision    recall  f1-score   support

      action       0.05      0.87      0.10       236
   adventure       0.75      0.04      0.07       164
      comedy       0.74      0.28      0.41      1203
       crime       1.00      0.01      0.02        97
       drama       0.63      0.26      0.37      1637
      horror       0.89      0.16      0.28       245
     romance       0.58      0.05      0.09       377
    thriller       0.00      0.00      0.00       211
     unknown       0.73      0.32      0.45      1226
     western       0.91      0.33      0.48       192

    accuracy                           0.27      5588
   macro avg       0.63      0.23      0.23      5588
weighted avg       0.66      0.27      0.34      5588
```

In [73]:
```
print(confusion_matrix(y_true,y_act))
```

```
[[ 205    0    2    0    8    0    2    0   19    0]
 [ 145    6    1    0   10    0    0    0    1    1]
 [ 776    0  339    0   62    1    1    0   21    3]
 [  85    0    1    1    9    0    0    0    1    0]
 [1071    1   59    0  432    0    6    0   67    1]
 [ 192    0    3    0    7   40    1    0    2    0]
 [ 263    0   28    0   37    0   18    0   31    0]
 [ 178    0    3    0   26    1    0    0    3    0]
 [ 707    1   22    0   92    3    3    0  397    1]
 [ 125    0    0    0    4    0    0    0    0   63]]
```

5.2

The model utilizing nouns only achieved an overall accuracy of 27%. This is 2% lower than the previous model.

6. Applying K-means Clustering

In [84]:
```
from sklearn.cluster import KMeans
```

```
k = 10
```

K-means++ init

In [85]:
```
kmeans = KMeans(n_clusters=k, init='k-means++',  max_iter=1000).fit(xtrain_tfidf)
```

In [86]:
```
ypred = kmeans.predict(xval_tfidf)

y_in = pd.DataFrame(y_pred).idxmax(axis=1).reset_index(drop = True)
y_act = y_in.apply(replace)

y_true = yval.idxmax(axis=1)
```

In [87]:
```
print(classification_report(yval,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.01      0.03       236
           1       0.75      0.04      0.07       164
           2       0.74      0.28      0.41      1203
           3       1.00      0.01      0.02        97
           4       0.62      0.26      0.37      1637
           5       0.89      0.16      0.28       245
           6       0.58      0.05      0.09       377
           7       0.00      0.00      0.00       211
           8       0.73      0.33      0.45      1226
           9       0.91      0.33      0.48       192

   micro avg       0.70      0.23      0.35      5588
   macro avg       0.72      0.15      0.22      5588
weighted avg       0.69      0.23      0.33      5588
 samples avg       0.23      0.23      0.23      5588
```

In [88]:
```
print(confusion_matrix(y_true,y_act))
```

```
[[ 205    0    2    0    8    0    2    0   19    0]
 [ 145    6    1    0   10    0    0    0    1    1]
 [ 776    0  339    0   62    1    1    0   21    3]
 [  85    0    1    1    9    0    0    0    1    0]
 [1071    1   59    0  432    0    6    0   67    1]
 [ 192    0    3    0    7   40    1    0    2    0]
 [ 263    0   28    0   37    0   18    0   31    0]
 [ 178    0    3    0   26    1    0    0    3    0]
 [ 707    1   22    0   92    3    3    0  397    1]
 [ 125    0    0    0    4    0    0    0    0   63]]
```

Random init

In [89]:
```
kmeans = KMeans(n_clusters=k, random_state=0, init='random', max_iter=1000).fit(xtrain_tfi
```

In [90]:
```
ypred = kmeans.predict(xval_tfidf)

y_in = pd.DataFrame(y_pred).idxmax(axis=1).reset_index(drop = True)
y_act = y_in.apply(replace)

y_true = yval.idxmax(axis=1)
```

In [91]:
```
print(classification_report(yval,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.01      0.03       236
           1       0.75      0.04      0.07       164
           2       0.74      0.28      0.41      1203
           3       1.00      0.01      0.02        97
           4       0.62      0.26      0.37      1637
           5       0.89      0.16      0.28       245
           6       0.58      0.05      0.09       377
           7       0.00      0.00      0.00       211
           8       0.73      0.33      0.45      1226
           9       0.91      0.33      0.48       192

   micro avg       0.70      0.23      0.35      5588
   macro avg       0.72      0.15      0.22      5588
weighted avg       0.69      0.23      0.33      5588
 samples avg       0.23      0.23      0.23      5588
```

In [92]:
```
print(confusion_matrix(y_true,y_act))
```

```
[[ 205    0    2    0    8    0    2    0   19    0]
 [ 145    6    1    0   10    0    0    0    1    1]
 [ 776    0  339    0   62    1    1    0   21    3]
 [  85    0    1    1    9    0    0    0    1    0]
 [1071    1   59    0  432    0    6    0   67    1]
 [ 192    0    3    0    7   40    1    0    2    0]
 [ 263    0   28    0   37    0   18    0   31    0]
 [ 178    0    3    0   26    1    0    0    3    0]
 [ 707    1   22    0   92    3    3    0  397    1]
 [ 125    0    0    0    4    0    0    0    0   63]]
```

random init with more initialization runs

In [93]:
```
kmeans = KMeans(n_clusters=k, init='random',n_init=25, max_iter=1000).fit(xtrain_tfidf)
```

In [94]:
```
ypred = kmeans.predict(xval_tfidf)

y_in = pd.DataFrame(y_pred).idxmax(axis=1).reset_index(drop = True)
y_act = y_in.apply(replace)

y_true = yval.idxmax(axis=1)
```

In [95]:
```
print(classification_report(yval,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.01      0.03       236
           1       0.75      0.04      0.07       164
           2       0.74      0.28      0.41      1203
           3       1.00      0.01      0.02        97
           4       0.62      0.26      0.37      1637
           5       0.89      0.16      0.28       245
           6       0.58      0.05      0.09       377
           7       0.00      0.00      0.00       211
           8       0.73      0.33      0.45      1226
           9       0.91      0.33      0.48       192

   micro avg       0.70      0.23      0.35      5588
```

```
   macro avg       0.72      0.15      0.22      5588
weighted avg       0.69      0.23      0.33      5588
 samples avg       0.23      0.23      0.23      5588
```

In [96]:

```
print(confusion_matrix(y_true,y_act))
```

```
[[ 205    0    2    0    8    0    2    0   19    0]
 [ 145    6    1    0   10    0    0    0    1    1]
 [ 776    0  339    0   62    1    1    0   21    3]
 [  85    0    1    1    9    0    0    0    1    0]
 [1071    1   59    0  432    0    6    0   67    1]
 [ 192    0    3    0    7   40    1    0    2    0]
 [ 263    0   28    0   37    0   18    0   31    0]
 [ 178    0    3    0   26    1    0    0    3    0]
 [ 707    1   22    0   92    3    3    0  397    1]
 [ 125    0    0    0    4    0    0    0    0   63]]
```

k-means++ init with more init runs

In [97]:

```
kmeans = KMeans(n_clusters=k, init='k-means++',n_init=25,  max_iter=1000).fit(xtrain_tfidf
```

In [98]:

```
ypred = kmeans.predict(xval_tfidf)

y_in = pd.DataFrame(y_pred).idxmax(axis=1).reset_index(drop = True)
y_act = y_in.apply(replace)

y_true = yval.idxmax(axis=1)
```

In [99]:

```
print(classification_report(yval,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.01      0.03       236
           1       0.75      0.04      0.07       164
           2       0.74      0.28      0.41      1203
           3       1.00      0.01      0.02        97
           4       0.62      0.26      0.37      1637
           5       0.89      0.16      0.28       245
           6       0.58      0.05      0.09       377
           7       0.00      0.00      0.00       211
           8       0.73      0.33      0.45      1226
           9       0.91      0.33      0.48       192

   micro avg       0.70      0.23      0.35      5588
   macro avg       0.72      0.15      0.22      5588
weighted avg       0.69      0.23      0.33      5588
 samples avg       0.23      0.23      0.23      5588
```

In [100…

```
print(confusion_matrix(y_true,y_act))
```

```
[[ 205    0    2    0    8    0    2    0   19    0]
 [ 145    6    1    0   10    0    0    0    1    1]
 [ 776    0  339    0   62    1    1    0   21    3]
 [  85    0    1    1    9    0    0    0    1    0]
 [1071    1   59    0  432    0    6    0   67    1]
 [ 192    0    3    0    7   40    1    0    2    0]
 [ 263    0   28    0   37    0   18    0   31    0]
 [ 178    0    3    0   26    1    0    0    3    0]
```

```
 [ 707    1   22    0   92    3    3    0  397    1]
 [ 125    0    0    0    4    0    0    0    0   63]]
```

6.2

Clustering could reproduce the classes in the dataset with 35% overall accuracy. This accuracy was not impacted by different intialization techniques. The accuracy was improved upon from the previous models so clustering is a good option for our business problem.