# milestone

September 25, 2021

## 1 0. Imports and constants

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import warnings

from ast import literal_eval
from random import sample
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from statistics import mode
from wordcloud import WordCloud
```

[2]:
```
seed = 42
warnings.filterwarnings('ignore')
```

## 2 1. Preprocessing

#### 2.0.1 1. Read data frames

[3]:
```
df_movies = pd.read_csv('movies.csv')
df_keywords = pd.read_csv('keywords.csv')
```

#### 2.0.2 2. Remove missing or malformed data

[4]:
```
columns = ['id','genres']
df_movies = df_movies[columns]
```

[5]:
```
df_movies.dropna(inplace=True)
df_movies['as_int'] = df_movies.id.apply(lambda x: x.isnumeric())
df_movies = df_movies[(df_movies.genres != '[]') & (df_movies.as_int)]
df_movies['id'] = df_movies.id.astype(int)
df_movies.drop(columns='as_int', inplace=True)
```

[6]:
```
df_keywords.dropna(inplace=True)
df_keywords = df_keywords[df_keywords.keywords != '[]']
```

### 2.0.3 3. Extract genres and keywords

[7]:
```
def extract_name(string):
    expression = literal_eval(string)
    return [literal['name'] for literal in expression]

df_movies['genres'] = df_movies.genres.apply(extract_name)
df_keywords['keywords'] = df_keywords.keywords.apply(extract_name)
```

### 2.0.4 4. Merge two data frames

[8]:
```
df = df_movies.merge(df_keywords, on='id')
```

[9]:
```
df.shape
```

[9]:
```
(31283, 3)
```

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

[10]:
```
      id                        genres  \
0    862  [Animation, Comedy, Family]
1   8844 [Adventure, Fantasy, Family]
2  15602            [Romance, Comedy]
3  31357     [Comedy, Drama, Romance]
4  11862                     [Comedy]

                                            keywords
0  [jealousy, toy, boy, friendship, friends, riva…
1  [board game, disappearance, based on children'…
2  [fishing, best friend, duringcreditsstinger, o…
3  [based on novel, interracial relationship, sin…
4  [baby, midlife crisis, confidence, aging, daug…
```

# 3 2. List all unique genres

[11]:
```
genres_list = df.genres.to_list()
genres_list = [set(genres) for genres in genres_list]
genre_set = set.union(*genres_list)
num_genres = len(genre_set)
```

[12]:
```
num_genres
```

[12]: 20

# 4 3. Vectorize keywords

[13]:
```
keywords_list = df.keywords.apply(lambda x: ' '.join(x)).to_list()
```

[14]:
```
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(keywords_list)
```

[15]:
```
X.shape
```

[15]: (31283, 12584)

[16]:
```
sample(vectorizer.get_feature_names(), 10)
```

[16]:
```
['promiscuous',
 'mutations',
 'tug',
 'dust',
 'prizefighting',
 'clearer',
 'alamo',
 'distraint',
 'ceremony',
 'guantánamo']
```

# 5 4. K-means clustering

### 5.0.1 1. Find the optimal k

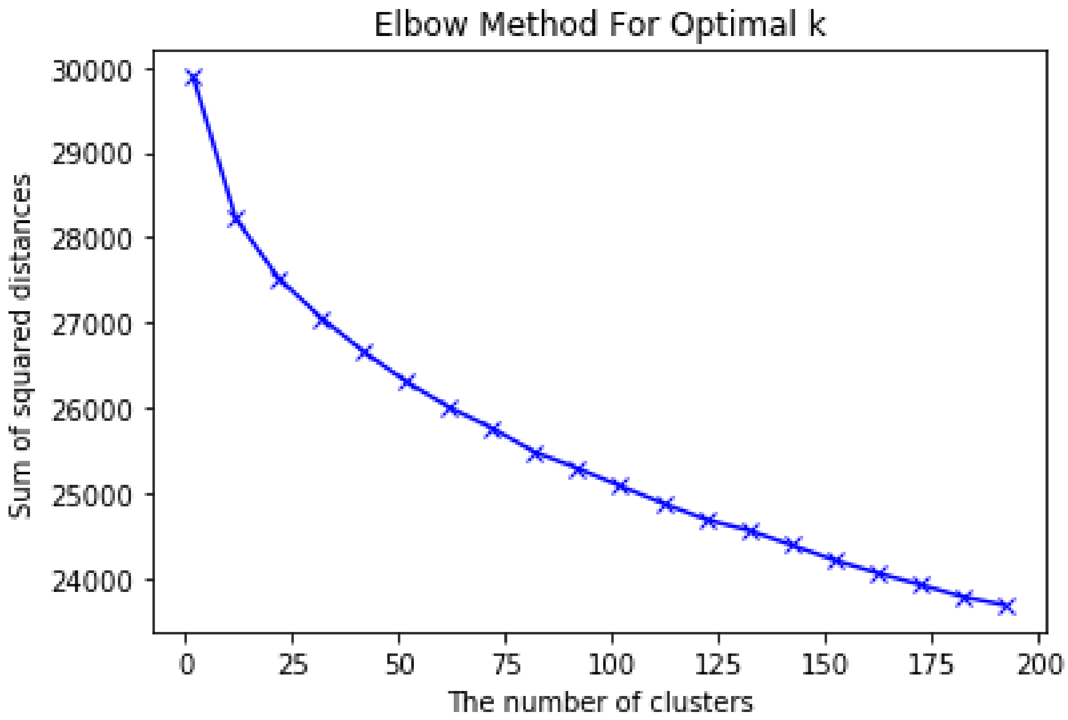
[17]:
```
distances = []
ks = range(2, 200, 10)
for k in ks:
    print(f'k = {k} start')
    kmeans = KMeans(n_clusters=k, random_state=seed)
    kmeans = kmeans.fit(X)
    print(f'k = {k} end: distance = {kmeans.inertia_}')
    distances.append(kmeans.inertia_)
```

```
k = 2 start
k = 2 end: distance = 29888.000684192273
k = 12 start
k = 12 end: distance = 28218.406994448087
k = 22 start
k = 22 end: distance = 27523.02128333069
k = 32 start
```

```
k = 32 end: distance = 27055.269220307724
k = 42 start
k = 42 end: distance = 26654.876260755533
k = 52 start
k = 52 end: distance = 26303.804759622788
k = 62 start
k = 62 end: distance = 26003.280829297593
k = 72 start
k = 72 end: distance = 25761.15802920406
k = 82 start
k = 82 end: distance = 25478.309631933604
k = 92 start
k = 92 end: distance = 25289.66914578733
k = 102 start
k = 102 end: distance = 25083.02332577031
k = 112 start
k = 112 end: distance = 24878.546495098482
k = 122 start
k = 122 end: distance = 24689.826445496867
k = 132 start
k = 132 end: distance = 24562.497530184573
k = 142 start
k = 142 end: distance = 24390.46982231676
k = 152 start
k = 152 end: distance = 24211.8180369881
k = 162 start
k = 162 end: distance = 24061.79976640658
k = 172 start
k = 172 end: distance = 23927.169681795946
k = 182 start
k = 182 end: distance = 23783.7355484722
k = 192 start
k = 192 end: distance = 23684.17443218657
```

[18]:

```
plt.plot(ks, distances, 'bx-')
plt.xlabel('The number of clusters')
plt.ylabel('Sum of squared distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```




#### 5.0.2 2. Try k = num_genres

[19]:
```
kmeans = KMeans(n_clusters=num_genres, random_state=seed)
kmeans = kmeans.fit(X)
```

[20]:
```
kmeans.labels_.shape
```

[20]: (31283,)

[21]:
```
kmeans.labels_
```

[21]: array([ 0, 17,  7, …,  0, 14,  0], dtype=int32)

#### 5.0.3 3. Find most common genre in each cluster

[22]:
```
df = pd.concat([df, pd.DataFrame(kmeans.labels_)], axis=1)
columns = df.columns.to_list()
columns[-1] = 'cluster'
df.columns = columns
```

[23]:
```
df.head()
```

[23]:
```
      id                         genres  \
0    862   [Animation, Comedy, Family]
1   8844  [Adventure, Fantasy, Family]
2  15602             [Romance, Comedy]
3  31357      [Comedy, Drama, Romance]
4  11862                      [Comedy]

                                            keywords  cluster
0  [jealousy, toy, boy, friendship, friends, riva…        0
1  [board game, disappearance, based on children'…       17
2  [fishing, best friend, duringcreditsstinger, o…        7
3  [based on novel, interracial relationship, sin…        5
4  [baby, midlife crisis, confidence, aging, daug…       13
```

[24]:
```python
most_common_genres = []
for k in range(0, num_genres):
    genres_list = df[df.cluster == k].genres.to_list()
    genres = [g for genres in genres_list for g in genres]
    most_common_genre = mode(genres)
    most_common_genres.append(most_common_genre)
```

[25]:
```python
most_common_genres
```

[25]:
```
['Drama',
 'Music',
 'Drama',
 'Drama',
 'Crime',
 'Drama',
 'Drama',
 'Comedy',
 'Drama',
 'Thriller',
 'Drama',
 'Drama',
 'Drama',
 'Drama',
 'Drama',
 'Drama',
 'Drama',
 'Drama',
 'Comedy',
 'Comedy']
```

# 6 5. Visualization

### 6.0.1 1. Create a scatter plot in a 2D space consisting of the first two principle components

[26]:
```
svd = TruncatedSVD(n_components=2, random_state=seed)
svd.fit(X)
```

[26]:
```
TruncatedSVD(algorithm='randomized', n_components=2, n_iter=5, random_state=42,
             tol=0.0)
```

[27]:
```
svd.explained_variance_ratio_
```

[27]:
```
array([0.04055117, 0.02195666])
```

[28]:
```
X_pca = svd.transform(X)
```

[29]:
```
X_pca
```

[29]:
```
array([[0.00258812, 0.0044618 ],
       [0.00358528, 0.00613359],
       [0.00294513, 0.00422809],
       …,
       [0.00109853, 0.00473279],
       [0.00353458, 0.00584774],
       [0.0013492 , 0.00231497]])
```

[32]:
```
X_pca.shape
```

[32]:
```
(31283, 2)
```

[33]:
```
fig = px.scatter(
    X_pca, x=0, y=1,
    color=kmeans.labels_,
    labels={
        '0': 'Principle Component 1',
        '1': 'Principle Component 2',
    },
)
fig.show()
```

### 6.0.2 2. Generate word clouds

[34]:
```
wordcloud = WordCloud(background_color='white')
for k in range(0, num_genres):
    keywords = df[df.cluster == k].keywords.to_list()
    keywords = [' '.join(w) for w in keywords]
    keywords = ' '.join(keywords)
```

```python
cloud = wordcloud.generate(keywords)
plt.imshow(cloud, interpolation='bilinear')
plt.title(f'Cluster {k}\nMost common genre = {most_common_genres[k]}')
plt.axis("off")
plt.show()
```

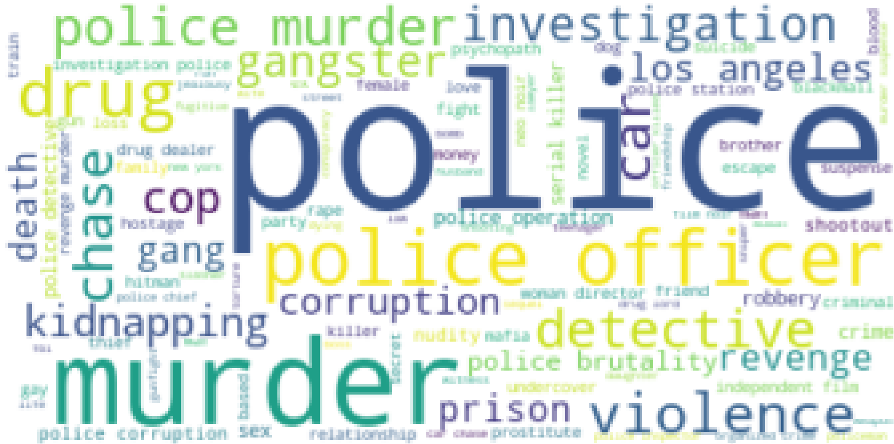
Cluster 0
Most common genre = Drama


Cluster 1
Most common genre = Music

Cluster 2
Most common genre = Drama

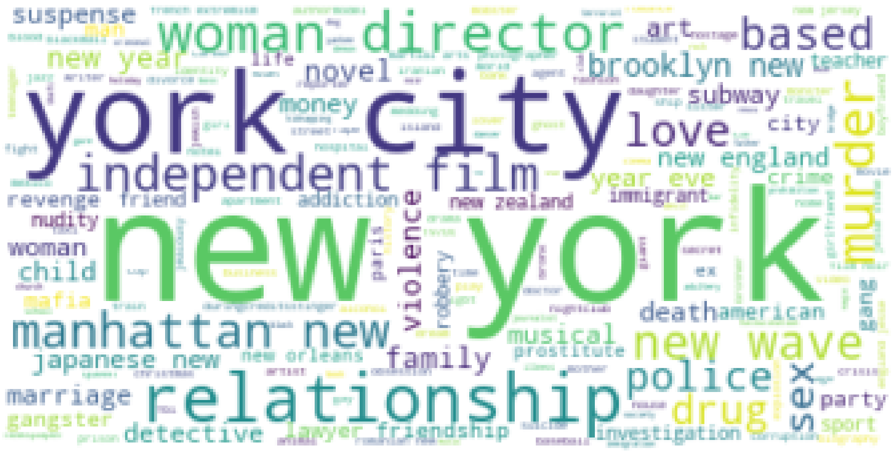

Cluster 3
Most common genre = Drama

Cluster 4
Most common genre = Crime



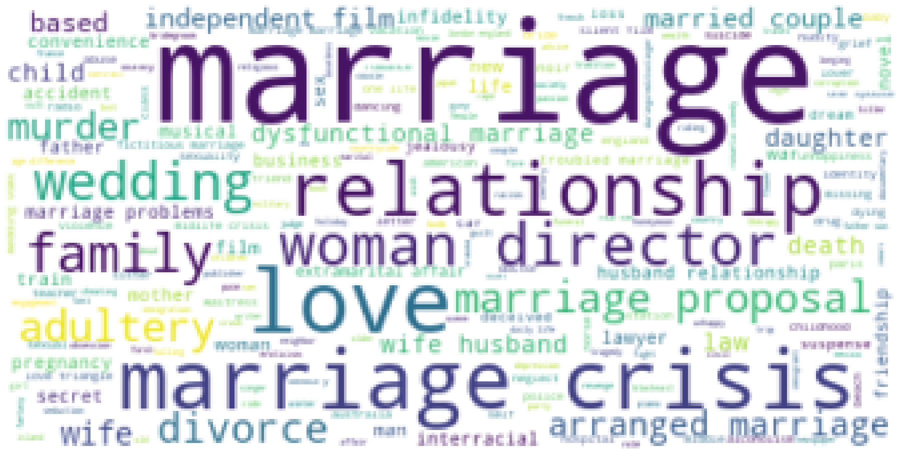

Cluster 5
Most common genre = Drama

Cluster 6
Most common genre = Drama

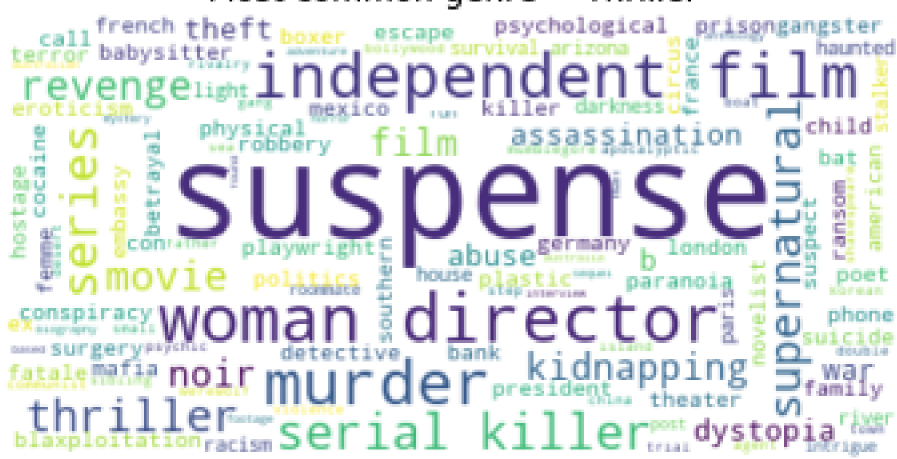



Cluster 7
Most common genre = Comedy

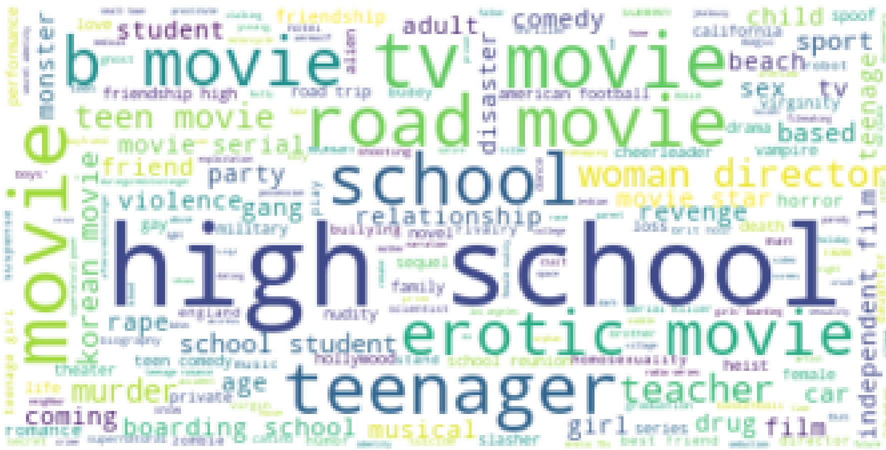
Cluster 8
Most common genre = Drama




Cluster 9
Most common genre = Thriller

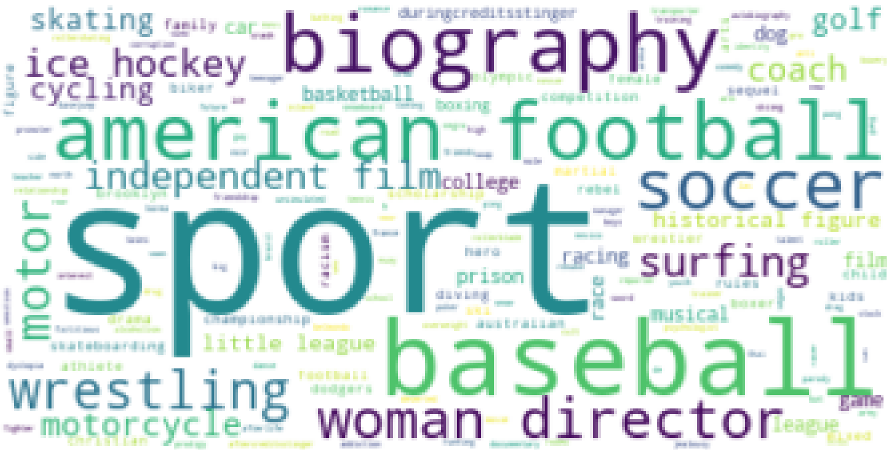


Cluster 10
Most common genre = Drama



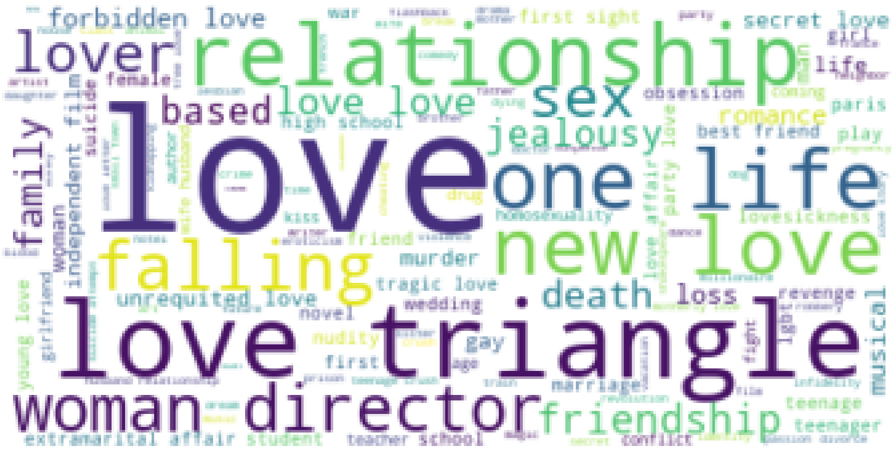

Cluster 11
Most common genre = Drama

Cluster 12
Most common genre = Drama



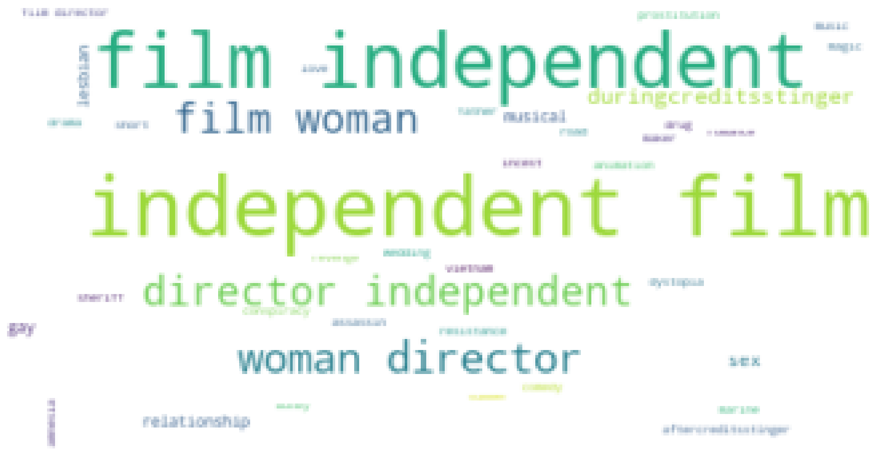

Cluster 13
Most common genre = Drama

Cluster 14
Most common genre = Drama

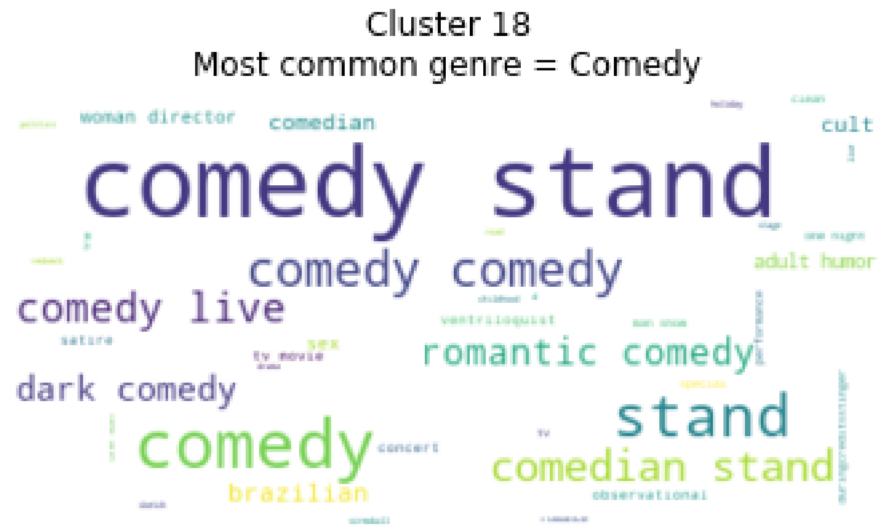


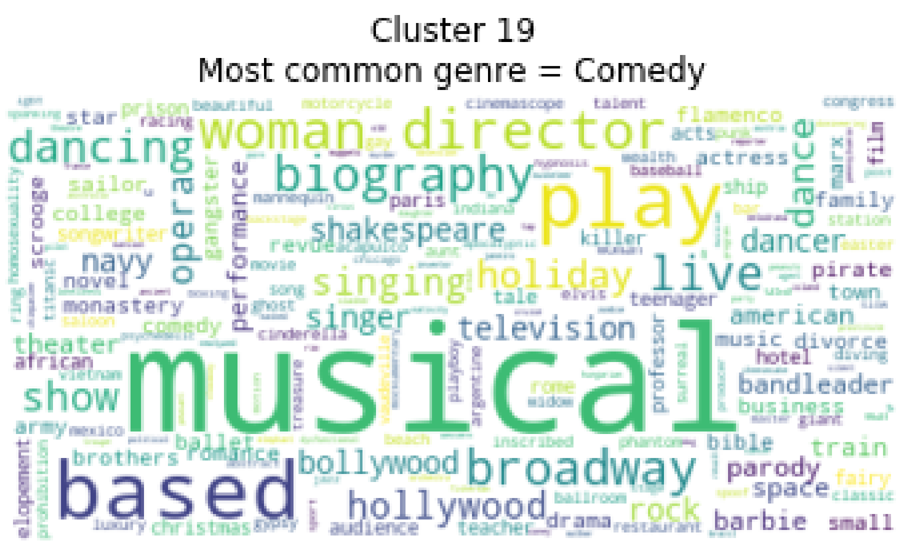

Cluster 15
Most common genre = Drama

Cluster 16
Most common genre = Drama




Cluster 17
Most common genre = Drama




## 7 Hidden

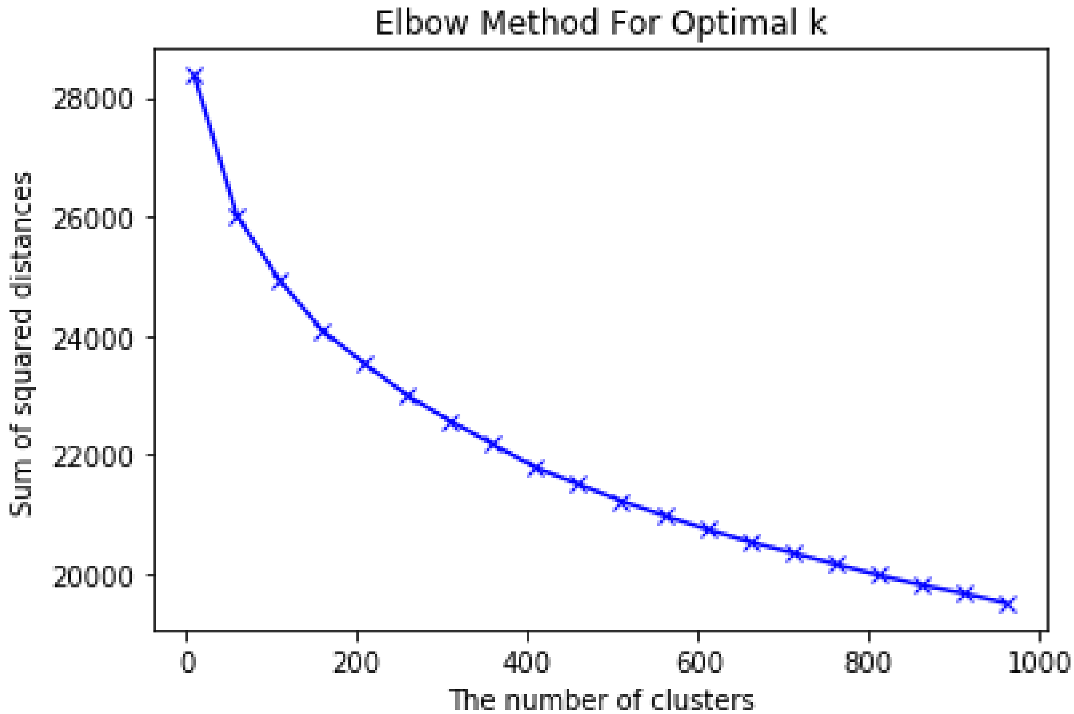
[35]:
```
distances = []
ks = range(10, 1000, 50)
for k in ks:
    print(f'k = {k} start')
    kmeans = KMeans(n_clusters=k, random_state=seed)
    kmeans = kmeans.fit(X)
    print(f'k = {k} end: distance = {kmeans.inertia_}')
```

```
    distances.append(kmeans.inertia_)
```

```
k = 10 start
k = 10 end: distance = 28373.555785436896
k = 60 start
k = 60 end: distance = 26007.698852031575
k = 110 start
k = 110 end: distance = 24926.726066920113
k = 160 start
k = 160 end: distance = 24084.972293957755
k = 210 start
k = 210 end: distance = 23519.39625023975
k = 260 start
k = 260 end: distance = 22985.427463817367
k = 310 start
k = 310 end: distance = 22560.678447825976
k = 360 start
k = 360 end: distance = 22169.284817642958
k = 410 start
k = 410 end: distance = 21776.061426616747
k = 460 start
k = 460 end: distance = 21499.279476578005
k = 510 start
k = 510 end: distance = 21217.987608508025
k = 560 start
k = 560 end: distance = 20971.750451639346
k = 610 start
k = 610 end: distance = 20741.44157609023
k = 660 start
k = 660 end: distance = 20531.932542709023
k = 710 start
k = 710 end: distance = 20345.06705547501
k = 760 start
k = 760 end: distance = 20156.229176562207
k = 810 start
k = 810 end: distance = 19973.196527848242
k = 860 start
k = 860 end: distance = 19807.131195846334
k = 910 start
k = 910 end: distance = 19664.452400597267
k = 960 start
k = 960 end: distance = 19500.658491228092
```

[36]:
```
plt.plot(ks, distances, 'bx-')
plt.xlabel('The number of clusters')
plt.ylabel('Sum of squared distances')
plt.title('Elbow Method For Optimal k')
```

```
plt.show()
```




## 8 References


- https://towardsdatascience.com/clustering-documents-with-python-97314ad6a78d
- https://pythonprogramminglanguage.com/kmeans-text-clustering/
- https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html#sklearn.deco
- https://plotly.com/python/pca-visualization/#2d-pca-scatter-plot