CLUSTERING, CLASSIFICATION, AND TOPIC MODELING USING MOVIE REVIEWS

Steve Desilets

MSDS 453: Natural Language Processing

Section 56 – Summer 2023

July 23, 2023

1. Introduction and Problem Statement

Natural language processing (NLP) techniques, such as clustering, classification, and topic modeling, represent incredible tools that can empower data scientists to identify important topics or to automatically group similar documents into classes or clusters. In theory, these fascinating NLP techniques could significantly benefit movie lovers interested in being able to automatically glean insights from movie reviews via cutting-edge algorithm-driven applications. With this goal of optimizing movie review classification, clustering, and topic modeling NLP pipelines in mind, the underlying movie review data leveraged in this study consists of 190 rows and 9 columns, as described in Table 1 below.

Table 1: Description of the Structure of the Movie Review Dataset

Variable Name	Python Data Type	Description
DSI_Title	Object	Unique identifier for the movie review (including student initials, Doc ID, and movie title)
Submission File Name	Object	Unique identifier for the movie review (including student initials, Doc ID, and movie title)
Student Name	Object	Initials of student who added review to the corpus
Genre of Movie	Object	Movie genre (action, comedy, horror, or sci-fi)
Review Type (pos or neg)	Object	Type of review (positive or negative)
Movie Title	Object	Movie title
Text	Object	First 500 words of the original movie review's text
Descriptor	Object	One-word summary of movie review (including genre, movie name, review type, and Doc ID)
Doc_ID	Integer	Unique ID for movie review document

In this paper, we explore the application of clustering, classification, and topic modeling NLP algorithms to this corpus of movie reviews. We study which combinations of data wrangling, document vectorization, and clustering / classification techniques are most effective at grouping movie reviews by sentiment, movie genre, and movie title. We also examine which data wrangling, vectorization, and topic modeling techniques prove to be most useful for identifying the (sentiment, movie genre, and movie title) topics underpinning our 190 movie reviews. Through these analyses, we hope to learn how to most effectively construct NLP pipelines focused on empowering movie lovers via clustering, classification, and topic modeling applications.

2. Research Design and Modeling Methods

Our exploration of NLP pipelines throughout this study was divided into four sections: 1) Clustering Experiments, 2) Sentiment Analysis Experiments, 3) Genre Classification Experiments, and 4) Topic Modeling Experiments. While conducting all four categories of experiments, we paid particular attention to the algorithms' effectiveness for reviews of our primary movie of interest in the corpus, "Sisters" (Moore 2015).

In the first phase of our study, the Clustering Experimentation phase, we began by importing the corpus of movie reviews into a Jupyter Notebook in Python, conducting exploratory data analysis, and implementing three methodologies for data wrangling and vectorization of this corpus. This step was important because applying varying data preprocessing techniques will allow us to more astutely examine the interaction between each of our NLP algorithms and the preprocessing techniques used to prepare the corpus for these applications. The three data wrangling and vectorization techniques leveraged are summarized in Table 2 below.

Table 2: Descriptions of the three Data Wrangling and Vectorization Methodologies

Data Wrangling and Vectorization Method 1	Data Wrangling and Vectorization Method 2	Data Wrangling and Vectorization Method 3
• Tokenization • Punctuation removal • Removal of non-alphabetic tokens • Removal of Tokens with three or fewer characters • Transformation of characters to lowercase • Removal of NLTK's English stop words • TF-IDF vectorization	• Tokenization • Punctuation removal • Removal of non-alphabetic tokens • Removal of tokens with three or fewer characters • Transformation of all characters to lowercase • Removal of NLTK's English stop words • Lemmatization • Removal of 21 of the top 100 terms (based on TF-IDF score) that we believed might add little value to a NLP pipeline focused on movie reviews (i.e. "film", "movie", etc.) • Doc2Vec vectorization	• Tokenization • Punctuation removal • Removal of non-alphabetic tokens • Removal of tokens with three or fewer characters • Transformation of all characters to lowercase • Removal of NLTK's English stop words • Lemmatization • Removal of 21 of the top 100 terms (based on TF-IDF score) that we believed might add little value to a NLP pipeline focused on movie reviews (i.e. "film", "movie", etc.) • TF-IDF vectorization

After applying these three data wrangling and vectorization techniques to the corpus of movie reviews, we then applied K-Means Clustering algorithms to each resulting term-document matrix (TDM). For each of our three TDMs, we created 24 K-Means Clustering models (72 models in total) with k ranging from 2 to 25 for each set of models. To assist us in comparing how effectively these models clustered the documents, we created 72 silhouette plots – one for each of the K-Means Clustering models generated as well as three line graphs (one for each data wrangling and vectorization method) that displayed the average silhouette score for each model (with number of clusters on the x-axis and average silhouette score on the y-axis). These plots enabled us to gauge whether the algorithm might be performing best when k equals 2, 4, or 19 – the number of sentiment categories, movie genres and distinct movie titles in our dataset. To help us further analyze the effectiveness of these clustering algorithms when setting k equal to 19 (to reflect the number of distinct movies), we also implemented Multi-Dimensional Scaling that allowed us to visually display the 19 resulting clusters in a two-dimensional

space. The visualizations resulting from these clustering experiments are all displayed in Appendix A and the full Python code associated with these experiments is presented in section 1 of Appendix E.

After completing the clustering experiments, we next turned our attention to the sentiment analysis experiments to determine whether Random Forest, Naïve Bayes, or Support Vector Machine Classification Models could effectively classify movie reviews based on their (positive or negative) sentiment. We applied each of these three algorithms to the TDMs resulting from each of our three data wrangling and vectorization techniques (though we were not able to apply Naïve Bayes Classification to the Doc2Vec vectors). After constructing each of these 8 models, we generated confusion matrices and Receiving Operating Characteristic (ROC) Curves based on application of the models to a test dataset, as well as summary statistics, such as testing accuracy, F1 score, precision, recall, and Area Under the Curve (AUC). For the Random Forest Classification Models, we also created bar charts to visualize the importance of the top 25 most important terms / features. The graphs associated with these sentiment analysis experiments are provided in Appendix B and the full Python code associated with these experiments are available in section 2 of Appendix E.

Having completed our sentiment analysis classification experiments, we pivoted toward conducting highly similar classification experiments that focused on movie genre classification. Again, we leveraged Random Forest, Naïve Bayes, and Support Vector Machine Classification models, but this time the models classified our movies into one of four genres: comedy, horror, science fiction, or action. We applied these three algorithms to the data resulting from each of our three data wrangling and vectorization techniques (though again, we could not apply Naïve Bayes Classification to the Doc2Vec vectors). After applying each of our 8 models to a test dataset, we presented the resulting testing confusion matrix and model performance metrics, such as accuracy and F1 score. For the Random Forest Classification Models, we also constructed bar charts to visualize the importance of the top 25 most important model features / terms. The charts resulting from these movie genre classification experiments are presented in Appendix C and the full Python code leveraged to run these experiments is available in section 3 of Appendix E.

For our last section of experiments, we focused on the effectiveness of two topic modeling algorithms – Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) – in uncovering topics underpinning our corpus of movie reviews. We applied our two topic modeling algorithms (LSA and LDA) to each of our three corpora (resulting from data wrangling and vectorization methods 1, 2, and 3) and repeated the experiments three times to allow the number of topics to equal 2, 4, and 19 (to reflect the number of sentiment categories, movie genres, and distinct movie titles). This experiment structure meant that we created 18 topic modeling NLP pipelines in total. For each topic modeling algorithm, we generated a cosine similarity heatmaps using the topic importance vectors generated for each movie review document, and we printed the top 10 terms associated with each topic. These analyses enabled us to gauge how well the sentiments, movie genres, or movie titles might be being captured by the resulting topics. We also created tables to summarize the coherences of each of these 18 models succinctly. The heatmaps, topic – top term dataframes, and coherence data tables are presented in Appendix D and the full Python code leveraged for these topic modeling experiments is available in section 4 of Appendix E.

3. Results

The results from the clustering, sentiment analysis classification, movie genre classification, and topic modeling analyses for this study are displayed in Appendices A – E. Appendix A displays a broad array of outputs from our K-Means clustering experiments (including silhouette plots, line graphs plotting average silhouette scores for various numbers of clusters, and multi-dimensional scaling-enabled cluster visualizations) that allow us to assess the performance of these models in clustering movie reviews with similar sentiments, genres, or movie titles together. Appendices B and C present a suite of outputs from our sentiment analysis and movie genre classification experiments – including confusion matrices, ROC curves, and model performance metrics – that empower us to determine how well our Random Forest, Naïve Bayes, and Support Vector Machine Classification models perform at classifying documents by sentiment or movie genre. The outputs of our 18 topic modeling experiments - including cosine similarity heatmaps, lists of the top 10 terms associated with each topic, and coherence measures – are presented in Appendix D to empower us to determine the effectiveness of our LSA and LDA models in uncovering

topics related to sentiments, movie genres, and movie titles. To promote transparency and reproducibility of our results, Appendix E contains the Python code leveraged to conduct our full analysis as well.

4. Analysis and Interpretation

Studying the clustering experiment resulting presented in Appendix A illuminates that K-Means clustering algorithms can perform quite well at clustering movie reviews based on their underlying characteristics. Upon an initial examination of the silhouette plots and average silhouette score line graphs resulting from our nine clustering algorithms, a researcher might initially believe that these clustering algorithms perform most strongly when applied to Doc2Vec vectors (such as those generated via data wrangling and vectorization method one). However, a closer look at the clusters generated after conducting multi-dimensional scaling reveals that K-Means clustering actually performed not particularly well at clustering movie reviews by movie title when leveraging Doc2Vec-backed data. However, K-Means clustering performed exceptionally well at clustering movies by movie title when applied to the TF-IDF-backed data generated from data wrangling methods 1 and 3. In fact, only two of the 190 movie reviews appear to have not been classified perfectly into 19 movie categories after conducting K-Means clustering. These results display that K-Means clustering (when applied to TF-IDF vectorized data) is a highly effective tool for clustering movie reviews by movie title.

Examination of the results from the sentiment analysis experiments reveals that none of the nine tested methodologies performed well at predicting movie review sentiments (as being positive or negative). The three Support Vector Machine Classification models performed particularly weakly – predicting that all movie reviews corresponded to the same sentiment, and in turn, each achieving an accuracy of just 47% and a F1 score of 0.474. Interestingly, even though all three SVM Classification models predicted the same default value for each review, the AUCs for the three SVM models (corresponding to the three data wrangling and vectorization methods) varied somewhat widely. At 0.52, the AUC associated with the application of the SVM model to the Doc2Vec-backed TDM matrix was the lowest, while the SVM model that was applied to data resulting from Data Wrangling and Vectorization Method 1 achieved an AUC of 0.73. Like the SVM classification models, the Naïve Bayes classification

models also performed poorly at classifying reviews by sentiment. The accuracy of the two Naïve Bayes models were just 36.8% and 42.1% when applied to the testing datasets, and both ROC curves appeared to be more concave up than concave down (which was further confirmed by both AUCs being less than 0.5). Though the Random Forest Classification models performed the best, these models did not perform particularly well either. The Random Forest Classification model performed best when applied to the Doc2Vec-backed TDM (from Data Wrangling and Vectorization Method 2), but even then, the testing accuracy, F1 score, and AUC were only 52.6%, 0.526, and 0.494, respectively. The poor performance of these Random Forest models is underscored by the bar charts displaying the most important terms from the Random Forest models, which seem to include seemingly unimportant terms like "another," "could," "look," and "together."

While the classification models performed poorly for classifying reviews based on sentiment, these models all performed stunningly well at classifying movies by genre. When applied to the TF-IDF-backed TDMs resulting from data wrangling methods 1 and 3, the Random Forest, Naïve Bayes, and Support Vector Machine Models all achieved testing accuracy and F1 scores of 100% and 1, respectively! This finding that 6 of the models performed perfectly when applied to the testing dataset underscores that these classification models may work best when applied to TF-IDF-backed TDMs rather than the Doc2Vec-backed TDMs generated from data wrangling and vectorization method 2. This point is emphasized by the fact that the Doc2Vec-supported Random Forest and Support Vector Machine Classification models achieved testing accuracies of just 63.2% and 15.8%, respectively. Interestingly, the Random Forest Classification models seemed to have identified some great and broadly applicable terms for classifying our movie reviews by genre too, including "scifi," "comedy,", "batman," "action," "joke," and "science," which suggests that the Random Forest models might even perform well on more broad populations of movie reviews that include other movies.

The topic modeling experiments revealed that while Latent Semantic Analysis holds some potential for uncovering important topics from movie reviews, Latent Dirichlet Allocation models may perform relatively less effectively. Specifically, application of LSA to the data resulting from data

wrangling and vectorization methods 2 and 3 seems to have produced cosine similarity heatmaps that highlight the 19 different movie titles quite vividly. Unfortunately, LSA seems to perform worse at identifying sentiment and movie genre topics when we set the number of topics to two or four. In fact, when we set the number of topics to two, the LSA model appears to just focus on whether a movie review is focused on the movie "Covenant" or literally any other movie – as underscored by the resulting pattern in the heatmap. Furthermore, when the number of topics is set to four, instead of clearly classifying movies by genre, three of the top 10 terms associated with the first topic in two of the models are "action," "horror," and "comedy," which suggests that many movies in these genres may be somewhat grouped together under the same topic by this algorithm. While LSA performed quite well at classifying reviews by movie title, the LDA models all appear to have performed quite poorly at identifying sentiment, movie genre, or movie title topics within our corpus. None of the nine heatmaps display clear patterns that would suggest the LDA models had uncovered clear trends reflecting sentiments, genres, or titles, and all the LDA model coherences are 0.314 or lower. Furthermore, the tables displaying the top terms when the number of terms equals 2 or 4 don't suggest that the LDA models successfully identified terms that would group reviews by genre or sentiment.

5. Conclusions

The experiments conducted throughout this analysis have helped us better understand the utility of clustering, classification, and topic modeling algorithms for identifying patterns in movie reviews. The clustering experiments highlighted that K-Means Clustering, when applied to TF-IDF-vectorized TDMs, proves to be a stunningly effective method of clustering movie reviews by movie title. The classification experiments revealed that while Random Forest, Naïve Bayes, and Support Vector Machine Classification models perform poorly at classifying movie reviews by sentiment, these models perform exceptionally well at classifying reviews by movie genre – especially when applied to term document matrices that leverage TF-IDF scores. Last, the topic modeling experiments suggested that while Latent Dirichlet Allocation may not perform well at identifying topics related to sentiment, genre, or movie title, Latent Semantic Analysis performs quite well at identifying topics that could by leveraged to classify movies by

movie title. These results suggest that some NLP clustering, classification, and topic modeling algorithms could be highly useful for movie lovers and researchers interested in automatically obtaining insights from movie reviews.

References

Moore, Jason. 2015. *Sisters*. United States: Universal Pictures.

Appendix A – Clustering Experiment Results

The images below display some of the most important outputs from the K-Means Clustering experiments conducted upon the corpus of movie reviews. Larger versions of each of these images are available in section 1 of Appendix E.

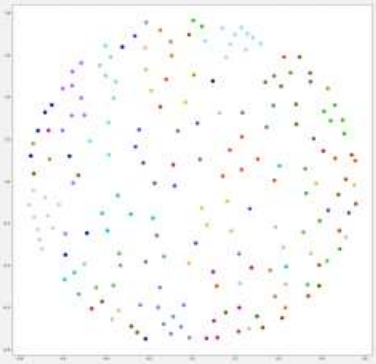
The first image below displays the silhouette plots resulting from conducting K-Means Clustering Experiments with 2, 4, or 19 clusters after implementing each of our three data wrangling and vectorization methods on the corpus.

The line graphs displayed below present the average silhouette scores resulting from K-Means Clustering experiments after applying each of our three data wrangling and vectorization methods to the corpus of movie reviews. The x-axis of each graph displays the number of clusters in an experiment and the y-axis displays the average silhouette score resulting from that experiment.

The three images below display visualizations of the clusters resulting from the K-Means Clustering experiments after setting the number of clusters (k) to 19 and leveraging multi-dimensional scaling so that we can view the results in two dimensions. The three images correspond to the experiment results after application of each of our three data wrangling and vectorization methods to the corpus of movie reviews, and the text beneath each image explains which clusters contain which movie review documents.

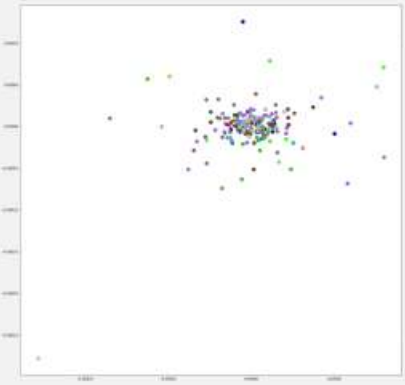
Visualization of K-Means Clustering Clusters Using Multi-Dimensional Scaling

Data Wrangling & Vectorization Method 1 – 19 Clusters

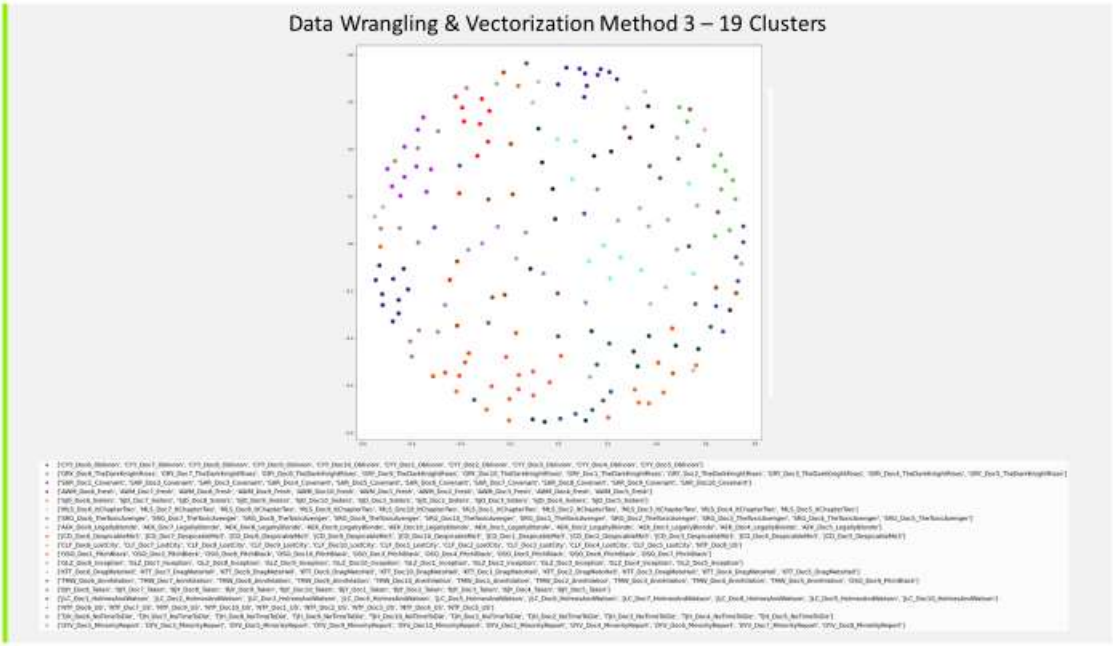


Visualization of K-Means Clustering Clusters Using Multi-Dimensional Scaling

Data Wrangling & Vectorization Method 2 – 19 Clusters

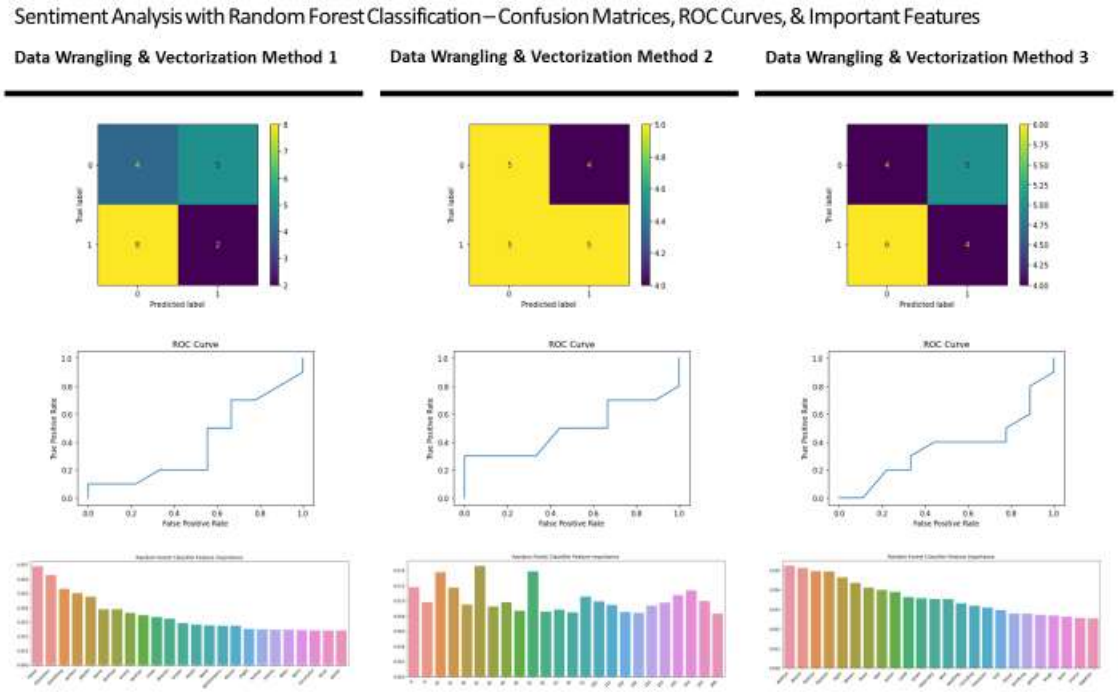


Visualization of K-Means Clustering Clusters Using Multi-Dimensional Scaling

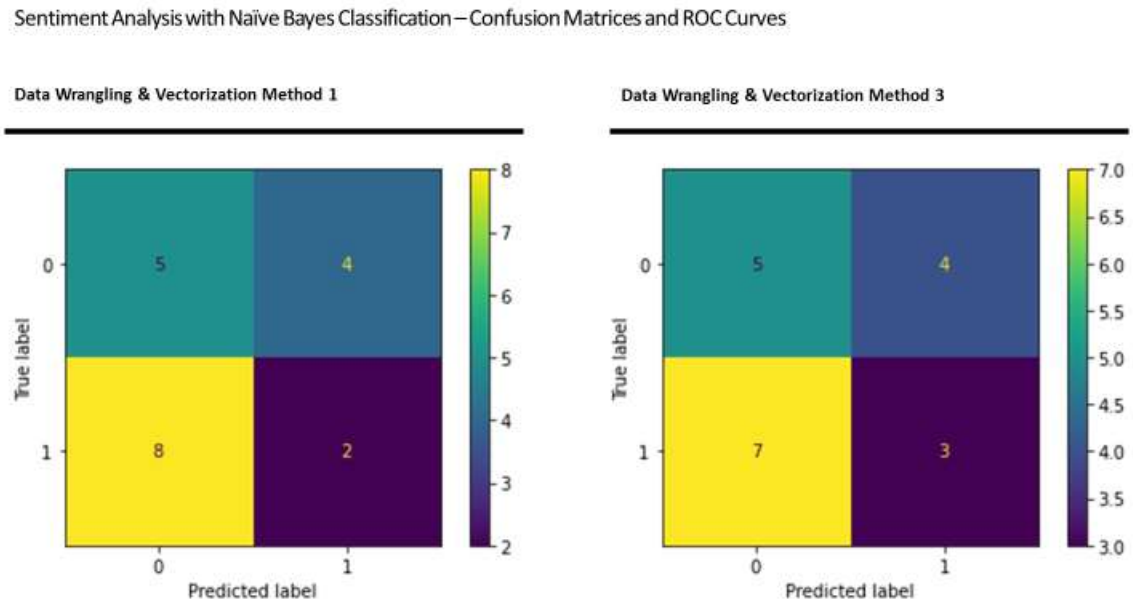


Appendix B – Sentiment Analysis Experiment Results

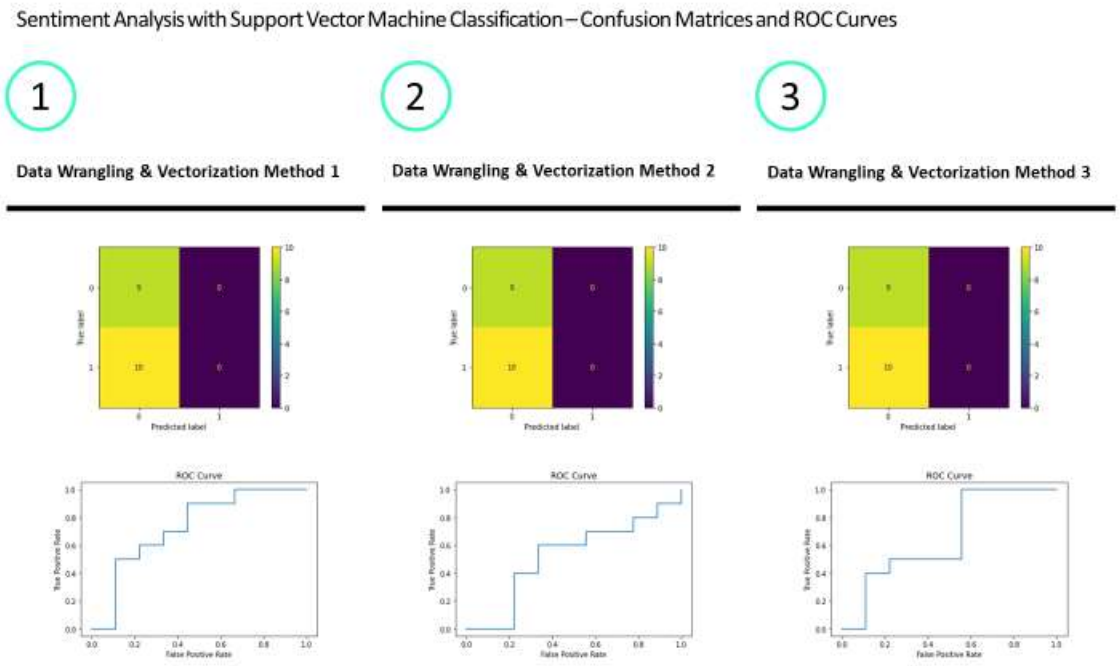
The image below displays the results of our Sentiment Analysis Classification Experiments that leveraged Random Forest Classification to predict which movie reviews were positive or negative. The resulting outputs include the confusion matrices and Receiving Operating Characteristic Curves associated with application of the Random Forest Models to a test dataset, as well as the 25 most important terms / features for the Random Forest Classification models.



The image below displays the confusion matrices resulting from applying a Naïve Bayes Classification Model to our corpus of movie reviews to predict which reviews are positive or negative. These Naïve Bayes Classification models were constructed using the data resulting from data wrangling and vectorization methods 1 and 3, and the confusion matrices display the results when these models were applied to test datasets.



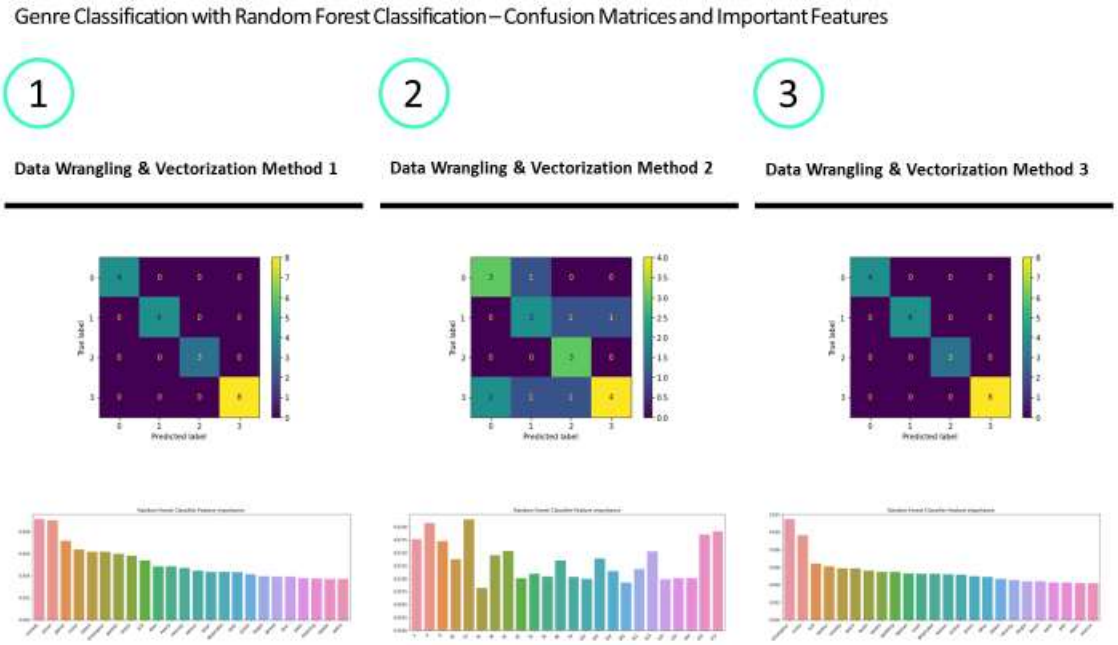
The image below displays the confusion matrices and ROC Curves resulting from sentiment analysis experiments that applied Support Vector Machine Classification models to the corpus of movie reviews. These models were constructed after applying data wrangling and vectorization methods 1, 2, and 3 to the corpus, and the results below reflect the application of these models to a test dataset.



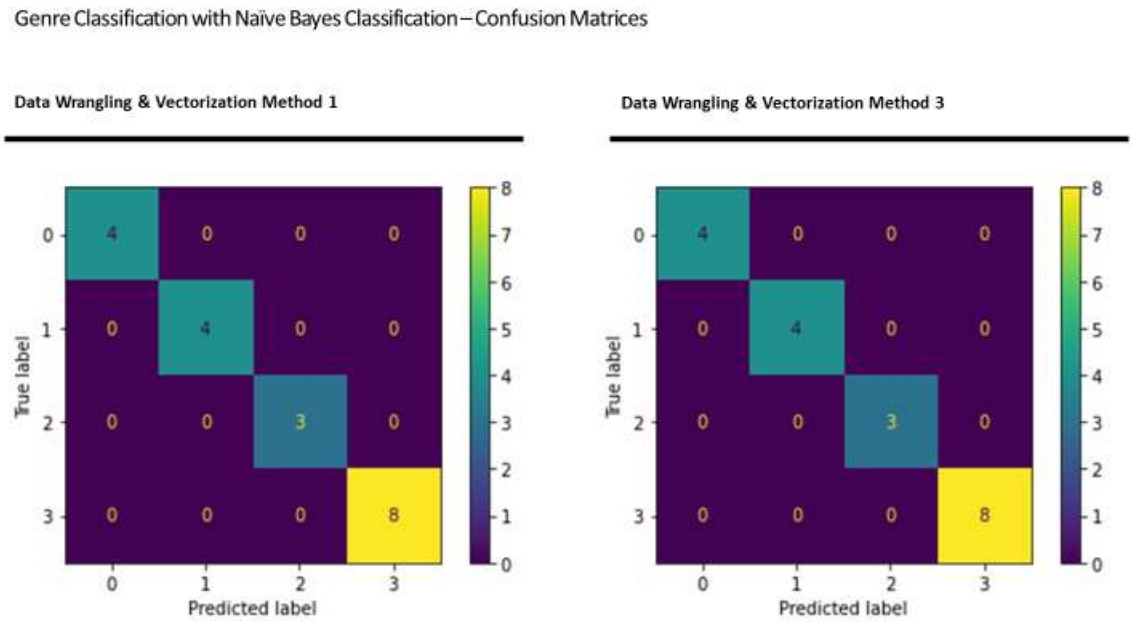
Larger versions of each of the graphics presented in Appendix B are available in section 2 of Appendix E.

Appendix C – Genre Classification Experiment Results

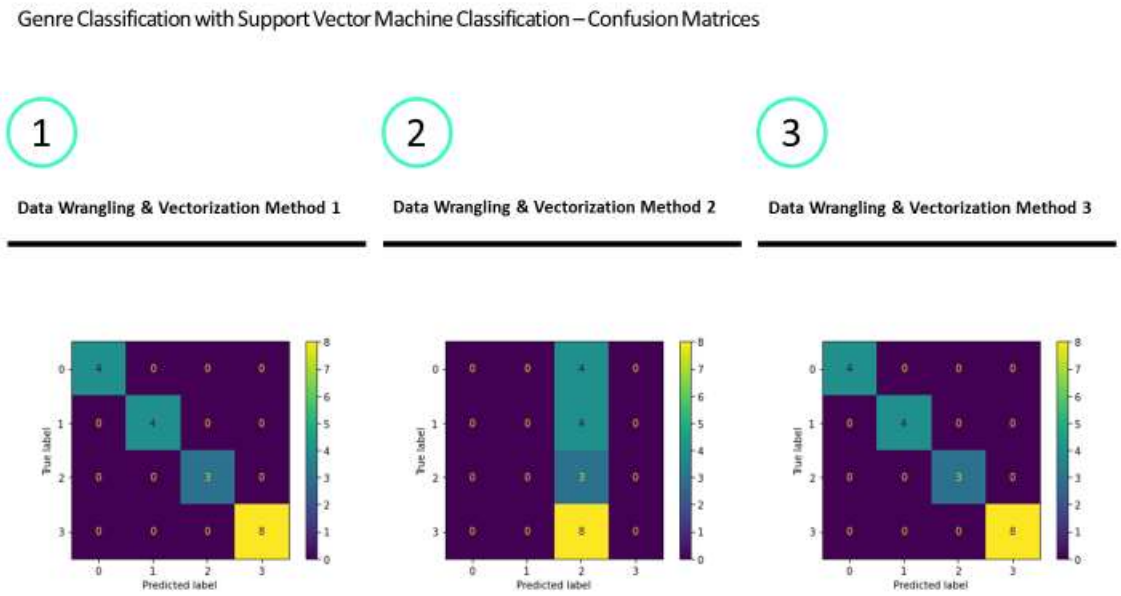
The image below displays the results of our Genre Classification Experiments that leveraged Random Forest Classification to predict which movie reviews corresponded to which movie genres. The resulting outputs include the confusion matrices and Receiving Operating Characteristic Curves associated with application of the Random Forest Models to a test dataset, as well as the 25 most important terms / features for the Random Forest Classification models.



The image below displays the confusion matrices resulting from applying a Naïve Bayes Classification Model to our corpus of movie reviews to predict which reviews correspond to which movie genres. These Naïve Bayes Classification models were constructed using the data resulting from data wrangling and vectorization methods 1 and 3, and the confusion matrices display the results when these models were applied to test datasets.



The image below displays the confusion matrices and ROC Curves resulting from genre classification experiments that applied Support Vector Machine Classification models to the corpus of movie reviews. These models were constructed after applying data wrangling and vectorization methods 1, 2, and 3 to the corpus, and the results below reflect the application of these models to a test dataset.

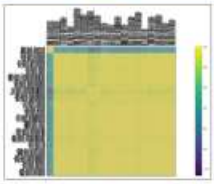
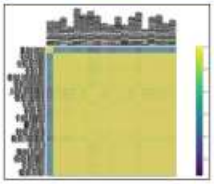
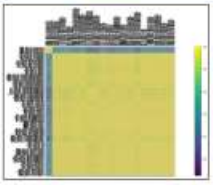
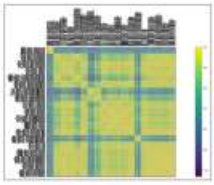
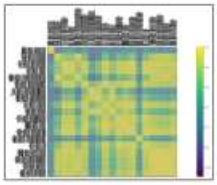
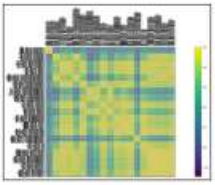
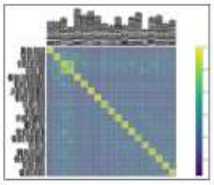
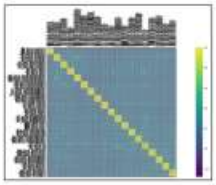
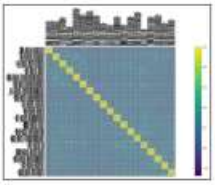


Larger versions of each of the graphics presented in Appendix C are available in section 3 of Appendix E.

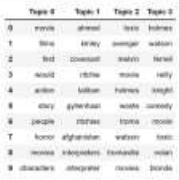
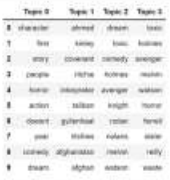
Appendix D – Topic Modeling Experiment Results

The first three images below display the outputs of Topic Modeling Experiments that applied Latent Semantic Analysis to our corpus of movie revies. The nine LSA experiments specified that there should be 2, 4, and 19 topics created when applying LSA to the data resulting from the application of data wrangling and vectorization methods 1, 2, and 3 to the corpus. The first image displays cosine similarity heatmaps resulting from the nine LSA experiments. The second image displays the top 10 terms associated with each of the topics for each experiment, and the third image presents the coherence associated with each of the nine LSA experiments. Larger versions of each of the graphics resulting from our LSA models are available in section 4.1 of Appendix E.

Latent Semantic Analysis – Cosine Similarity Heatmaps

Number of Categories	Data Wrangling and Vectorization Method		
	Method 1	Method 2	Method 3
2 Categories			
4 Categories			
19 Categories			

Latent Semantic Analysis – Most Important Terms Per Topic

Number of Categories	Data Wrangling and Vectorization Method		
	Method 1	Method 2	Method 3
2 Categories			
4 Categories			
19 Categories			

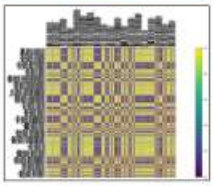
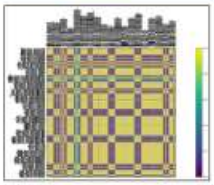
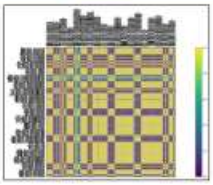
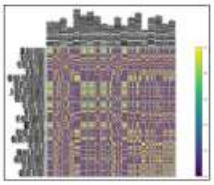
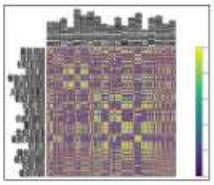
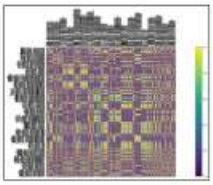
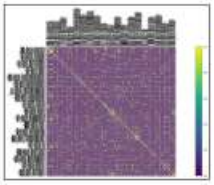
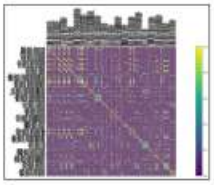
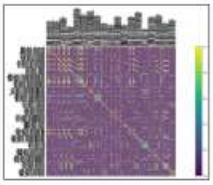
Latent Semantic Analysis – Coherence by Data Wrangling / Vectorization Method and Number of Topics

Number of Topics	Data Wrangling and Vectorization Method		
	Method 1	Method 2	Method 3
2 Topics	0568	0.716	0.716
4 Topics	0.629	0.421	0.578
19 Topics	0.475	0.426	0.484

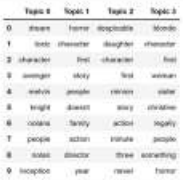
The first three images below display the outputs of Topic Modeling Experiments that applied Latent Dirichlet Allocation to our corpus of movie revies. The nine LDA experiments specified that there should be 2, 4, and 19 topics created when applying LDA to the data resulting from the application of data

wrangling and vectorization methods 1, 2, and 3 to the corpus. The first image displays cosine similarity heatmaps resulting from the nine LDA experiments. The second image displays the top 10 terms associated with each of the topics for each experiment, and the third image presents the coherence associated with each of the nine LDA experiments. Larger versions of each of the graphics resulting from our LDA models are available in section 4.2 of Appendix E.

Latent Dirichlet Allocation – Cosine Similarity Heatmaps

Number of Categories	Data Wrangling and Vectorization Method		
	Method 1	Method 2	Method 3
2 Categories			
4 Categories			
19 Categories			

Latent Dirichlet Allocation – Most Important Terms Per Topic

Number of Categories	Data Wrangling and Vectorization Method		
	Method 1	Method 2	Method 3
2 Categories			
4 Categories			
19 Categories			

Latent Dirichlet Allocation – Coherence by Data Wrangling / Vectorization Method and Number of Topics

Number of Topics	Data Wrangling and Vectorization Method		
	Method 1	Method 2	Method 3
2 Topics	0.243	0.289	0.289
4 Topics	0.244	0.285	0.285
19 Topics	0.314	0.314	0.314

Appendix E - Supporting Python Code

1) Clustering

1.1) Loading The Corpus

Let's begin by importing our corpus - the dataset of moview reviews.

In [1]:
```
# Import relevant packages
import pandas as pd
import os
import numpy as np
import re
import string
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
import random
from dataclasses import dataclass

from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
from nltk.stem import PorterStemmer

import gensim
from gensim import corpora, similarities
from gensim.models import Word2Vec, LdaMulticore, TfidfModel, CoherenceModel
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
from gensim.models import LsiModel,LdaModel

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.manifold import TSNE, MDS
from sklearn.cluster import KMeans
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, KFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, C
from sklearn.metrics import precision_score, recall_score, roc_auc_score, roc_curve, P
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

import scipy.cluster.hierarchy

from IPython.display import display, HTML

from typing import List, Callable, Dict
```

```
C:\Users\steve\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumP
y version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected versio
n 1.25.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Complete NLTK Downloads
# Only run this once, they will be downloaded.
nltk.download('stopwords',quiet=True)
nltk.download('wordnet',quiet=True)
nltk.download('punkt',quiet=True)
nltk.download('omw-1.4',quiet=True)
```

Out[2]:
```
True
```

In [3]:
```
print("Genism Version: ", gensim.__version__)
```

```
Genism Version:  4.1.2
```

In [4]:
```
# Suppress Warning Messages
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [5]:
```
# Create corpus dataframe

# Create Document class
@dataclass
class Document:
    doc_id: str
    text: str

def add_movie_descriptor(data: pd.DataFrame, corpus_df: pd.DataFrame):
    """
    Adds "Movie Description" to the supplied dataframe, in the form {Genre}_{P|N}_{Mov
    """
    review = np.where(corpus_df['Review Type (pos or neg)'] == 'Positive', 'P', 'N')
    data['Descriptor'] = corpus_df['Genre of Movie'] + '_' + corpus_df['Movie Title']

def get_corpus_df(path):
    data = pd.read_csv(path, encoding="utf-8")
    add_movie_descriptor(data, data)
    sorted_data = data.sort_values(['Descriptor'])
    indexed_data = sorted_data.set_index(['Doc_ID'])
    indexed_data['Doc_ID'] = indexed_data.index
    return indexed_data

# Define documents in this new class
corpus_df = get_corpus_df('MSDS453_ClassCorpus_Final_Sec56_v5_20230720.csv')
documents = [Document(x, y) for x, y in zip(corpus_df.Doc_ID, corpus_df.Text)]
```

1.2) Exploratory Data Analysis

Having imported the corpus of movie reviews, let's conduct exploratory data analysis on the raw dataset.

In [6]:
```
corpus_df.shape
```

Out[6]:
```
(190, 9)
```

In [7]:
```
corpus_df.head(4).T
```

Out[7]:

Doc_ID	101	102	103	10
DSI_Title	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_Doc4_Covena
Submission File Name	SAR_Doc1_Covenant	SAR_Doc2_Covenant	SAR_Doc3_Covenant	SAR_Doc4_Covena
Student Name	SAR	SAR	SAR	SA
Genre of Movie	Action	Action	Action	Acti
Review Type (pos or neg)	Negative	Negative	Negative	Negati
Movie Title	Covenant	Covenant	Covenant	Covena
Text	Nearly two years after the American military w...	Have Guy Ritchie and Jake Gyllenhaal switched ...	Guy Ritchie's The Covenant notably marks the f...	In a weird throwback those Chuck Nor "M
Descriptor	Action_Covenant_N_101	Action_Covenant_N_102	Action_Covenant_N_103	Action_Covenant_N_1(
Doc_ID	101	102	103	1(

In [8]:
```
print(corpus_df.info());
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 190 entries, 101 to 217
Data columns (total 9 columns):
 #   Column                   Non-Null Count  Dtype 
---  ------                   --------------  ----- 
 0   DSI_Title                190 non-null    object
 1   Submission File Name     190 non-null    object
 2   Student Name             190 non-null    object
 3   Genre of Movie           190 non-null    object
 4   Review Type (pos or neg) 190 non-null    object
 5   Movie Title              190 non-null    object
 6   Text                     190 non-null    object
 7   Descriptor               190 non-null    object
 8   Doc_ID                   190 non-null    int64 
dtypes: int64(1), object(8)
memory usage: 14.8+ KB
None
```

In [9]:
```
print(corpus_df['Movie Title'].unique())
```

```
['Covenant' 'Inception' 'No time to die' 'Taken' 'The Dark Knight Rises'
 'Despicable Me 3' 'Holmes and Watson' 'Legally Blonde' 'Lost City'
 'Sisters' 'Drag Me to Hell' 'Fresh' 'It Chapter Two' 'The Toxic Avenger'
 'US' 'Annihilation' 'Minority Report' 'Oblivion' 'Pitch Black']
```

In [10]:
```
# Gather the number of reviews by genre
counts_df = corpus_df[['Genre of Movie']].copy()
counts_df['Count'] = 1
counts_df.groupby(['Genre of Movie']).count().reset_index()
```

Out[10]:

	Genre of Movie	Count
0	Action	50
1	Comedy	50
2	Horror	50
3	Sci-Fi	40

In [11]:
```
corpus_df.columns
```

Out[11]:
```
Index(['DSI_Title', 'Submission File Name', 'Student Name', 'Genre of Movie',
       'Review Type (pos or neg)', 'Movie Title', 'Text', 'Descriptor',
       'Doc_ID'],
      dtype='object')
```

1.3) Data Wrangling And Vectorization

1.3.1) Data Wrangling and Vectorization Method 1

Let's implement our first data wrangling and vectorization methodology.

As specified in the clean_doc_method_one and tfidf functions below, this methodology will consist of:

- Tokenization
- Punctuation Removal
- Removal of Non-Alphabetic Tokens
- Removal of Tokens with 3 or Fewer Characters
- Changing all characters to lowercase
- Removal of Standard English Stop Words
- TF-IDF Vectorization of Documents

The outputs will be stored in the "processed_text_method_one" column of the corpus_df dataframe and in the "processed_text_method_one" list.

In [12]:
```
def normalize_document(document: Document) -> Document:
    text = document.text
    text = remove_punctuation(text)
    text = lower_case(text)
    text = remove_tags(text)
    text = remove_special_chars_and_digits(text)

    return Document(document.doc_id, text)

def normalize_documents(documents: List[Document]) -> List[Document]:
    """
```

```
    Normalizes text for all given documents.
    Removes punctuation, converts to lower case, removes tags and special characters.
    """
    return [normalize_document(x) for x in documents]

@dataclass
class TokenizedDocument:
    doc_id: str
    tokens: List[str]

def tokenize_document(document: Document) -> TokenizedDocument:
    tokens = nltk.word_tokenize(document.text)
    return TokenizedDocument(document.doc_id, tokens)

def tokenize_documents(documents: List[Document]) -> List[TokenizedDocument]:
    return [tokenize_document(x) for x in documents]

def lemmatize(documents: List[TokenizedDocument]) -> List[TokenizedDocument]:
    result = []
    lemmatizer = WordNetLemmatizer()
    for document in documents:
        output_tokens = [lemmatizer.lemmatize(w) for w in document.tokens]
        result.append(TokenizedDocument(document.doc_id, output_tokens))

    return result

def stem(documents: List[TokenizedDocument]) -> List[TokenizedDocument]:
    result = []
    stemmer = PorterStemmer()
    for document in documents:
        output_tokens = [stemmer.stem(w) for w in document.tokens]
        result.append(TokenizedDocument(document.doc_id, output_tokens))

    return result

def remove_stop_words(documents: List[TokenizedDocument]) -> List[TokenizedDocument]:
    result = []

    stop_words = set(nltk.corpus.stopwords.words('english'))
    for document in documents:
        filtered_tokens = [w for w in document.tokens if not w in stop_words]
        result.append(TokenizedDocument(document.doc_id, filtered_tokens))

    return result

def add_flags(data: pd.DataFrame, sisters_doc_ids: List[int], comedy_doc_ids: List[int
    data['is_sisters'] = data.index.isin(sisters_doc_ids)
    data['is_comedy'] = data.index.isin(comedy_doc_ids)

def get_all_tokens(documents: List[TokenizedDocument]) -> List[str]:
    tokens = {y for x in documents for y in x.tokens}
    return sorted(list(tokens))

def clean_doc_method_one(doc):
    #split document into individual words
    tokens=doc.split()
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # remove remaining tokens that are not alphabetic
```

```
    tokens = [word for word in tokens if word.isalpha()]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 4]
    #lowercase all words
    tokens = [word.lower() for word in tokens]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # word stemming
    # ps=PorterStemmer()
    # tokens=[ps.stem(word) for word in tokens]
    return tokens

def final_processed_text_disabled(doc):
    #this is a function to join the processed text back
    ' '.join(doc)
    return doc

def tfidf(corpus, titles, ngram_range = (1,1)):
    #this is a function to created the tfidf matrix
    Tfidf=TfidfVectorizer(ngram_range=(1,1))

    #fit the vectorizer using final processed documents.  The vectorizer requires the
    #stiched back together document.

    TFIDF_matrix=Tfidf.fit_transform(corpus)

    #creating dataframe from TFIDF Matrix

    # https://stackoverflow.com/questions/70215049/attributeerror-tfidfvectorizer-obje
    #words = Tfidf.get_feature_names() # For sklearn <= 0.24.x
    #matrix=pd.DataFrame(TFIDF_matrix.toarray(), columns=Tfidf.get_feature_names(), ir

    words = Tfidf.get_feature_names_out() # For sklearn >= 1.0.x
    matrix=pd.DataFrame(TFIDF_matrix.toarray(), columns=Tfidf.get_feature_names_out(),
    return matrix #,words
```

In [13]:
```
#adding two columns to the dataframe to store the processed text and tokenized text
corpus_df['processed_text_method_one'] = corpus_df['Text'].apply(lambda x: clean_doc_m

#creating final processed text variables for matrix creation
final_processed_text_method_one = [' '.join(x) for x in corpus_df['processed_text_meth
titles = corpus_df['DSI_Title'].tolist()
processed_text_method_one = corpus_df['processed_text_method_one'].tolist()
```

In [14]:
```
tfidf_matrix_method_one = tfidf(final_processed_text_method_one, titles, ngram_range =
```

In [15]:
```
tfidf_matrix_method_one
```

Out[15]:

	aardman	aaron	abandon	abandoned	abandonment	abandons	abdicate	abc
SAR_Doc1_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc2_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc3_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc4_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc5_Covenant	0.0	0.0	0.0	0.099853	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
OSO_Doc3_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
OSO_Doc4_PitchBlack	0.0	0.0	0.0	0.063045	0.0	0.0	0.0	
OSO_Doc5_PitchBlack	0.0	0.0	0.0	0.057051	0.0	0.0	0.0	
OSO_Doc6_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
OSO_Doc7_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	

190 rows × 11273 columns

1.3.2) Data Wrangling and Vectorization Method 2

We will now implment Data Wrangling and Vectorization Method 2.

This methodology is very similar to method 1, except we also lemmatize the corpus, remove custom stop words, and use Doc2Vec vectorization of the documents (instead of TF-IDF vectorization).

In summary, these are the steps of this data wrangling and vectorizaiton methodology:

- Tokenization
- Punctuation Removal
- Removal of Non-Alphabetic Tokens
- Removal of Tokens with 3 or Fewer Characters
- Changing all characters to lowercase
- Removal of Standard English Stop Words
- Lemmatization
- Removal of Custom Stop Words
- Doc2Vec Vectorization of Documents

The outputs will be stored in the "processed_text_method_three" column of the corpus_df dataframe and in the "final_processed_text_method_three" list.

In [145…

```
# Define New Function for Data Wrangling
def clean_doc_method_three(doc):
    #split document into individual words
    tokens=doc.split()
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
```

```
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 4]
    #lowercase all words
    tokens = [word.lower() for word in tokens]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    # filter out custom stop words
    custom_stop_words = set(['film', 'movie', 'ha', 'wa', 'get', 'make', 'also', 'much
    tokens = [w for w in tokens if not w in custom_stop_words]
    return tokens

#adding two columns to the dataframe to store the processed text and tokenized text
corpus_df['processed_text_method_three'] = corpus_df['Text'].apply(lambda x: clean_doc

corpus_df['processed_text_method_three_string'] = corpus_df['processed_text_method_thr

documents_method_three = [Document(x, y) for x, y in zip(corpus_df.Doc_ID, corpus_df.p

tokenized_documents_method_three = tokenize_documents(documents_method_three)

descriptors_by_doc_ids = {x: y for x, y in zip(corpus_df['Doc_ID'], corpus_df['Descrip

descriptors_list = [x for x in corpus_df['Descriptor']]

def run_doc2vec(documents: List[TokenizedDocument], embedding_size: int, descriptors_b
    tagged_documents = [TaggedDocument(document.tokens, [i]) for i, document in enumer
    doc2vec_model = Doc2Vec(tagged_documents, vector_size=embedding_size, window=3, mi

    global doc2vec_df

    doc2vec_df = pd.DataFrame()
    for document in documents:
        vector = pd.DataFrame(doc2vec_model.infer_vector(document.tokens)).transpose()
        doc2vec_df = pd.concat([doc2vec_df, vector], axis=0)

    doc2vec_df['Descriptor'] = [x for x in descriptors_list]
    doc2vec_df.set_index(['Descriptor'], inplace=True)
    return doc2vec_df

run_doc2vec(tokenized_documents_method_three, 300, 'Doc2Vec_Vectorization')

#creating final processed text variables for matrix creation
final_processed_text_method_three = [' '.join(x) for x in corpus_df['processed_text_me
```

1.3.3) Data Wrangling and Vectorization Method 3

We will now implment Data Wrangling and Vectorization Method 3.

This methodology is very similar to method 1, except we also lemmatize the corpus and remove custom stop words.

In summary, these are the steps of this data wrangling and vectorizaiton methodology:

- Tokenization
- Punctuation Removal
- Removal of Non-Alphabetic Tokens
- Removal of Tokens with 3 or Fewer Characters
- Changing all characters to lowercase
- Removal of Standard English Stop Words
- Lemmatization
- Removal of Custom Stop Words
- TF-IDF Vectorization of Documents

The outputs will be stored in the "processed_text_method_four" column of the corpus_df dataframe and in the "processed_text_method_four" list.

In [21]:
```
# Define New Function for Data Wrangling
def clean_doc_method_four(doc):
    #split document into individual words
    tokens=doc.split()
    re_punc = re.compile('[%s]' % re.escape(string.punctuation))
    # remove punctuation from each word
    tokens = [re_punc.sub('', w) for w in tokens]
    # remove remaining tokens that are not alphabetic
    tokens = [word for word in tokens if word.isalpha()]
    # filter out short tokens
    tokens = [word for word in tokens if len(word) > 4]
    #lowercase all words
    tokens = [word.lower() for word in tokens]
    # filter out stop words
    stop_words = set(stopwords.words('english'))
    tokens = [w for w in tokens if not w in stop_words]
    # lemmatization
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    # filter out custom stop words
    custom_stop_words = set(['film', 'movie', 'ha', 'wa', 'get', 'make', 'also', 'much
    tokens = [w for w in tokens if not w in custom_stop_words]
    return tokens
```

In [22]:
```
#adding two columns to the dataframe to store the processed text and tokenized text
corpus_df['processed_text_method_four'] = corpus_df['Text'].apply(lambda x: clean_doc_

#creating final processed text variables for matrix creation
final_processed_text_method_four = [' '.join(x) for x in corpus_df['processed_text_met



titles = corpus_df['DSI_Title'].tolist()
processed_text_method_four = corpus_df['processed_text_method_four'].tolist()

# Create TF-IDF Matrix
tfidf_matrix_method_four = tfidf(final_processed_text_method_four, titles, ngram_range

tfidf_matrix_method_four
```

Out[22]:

	aardman	aaron	abandon	abandoned	abandonment	abdicate	abdicating	ab
SAR_Doc1_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc2_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc3_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc4_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc5_Covenant	0.0	0.0	0.0	0.100975	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
OSO_Doc3_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
OSO_Doc4_PitchBlack	0.0	0.0	0.0	0.063795	0.0	0.0	0.0	
OSO_Doc5_PitchBlack	0.0	0.0	0.0	0.058085	0.0	0.0	0.0	
OSO_Doc6_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
OSO_Doc7_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	

190 rows × 10397 columns

1.4) Clustering Experiments

1.4.1) K-Means Clustering Experiments

Clustering Experiment Experiment 1: K-Means Clustering With Data Wrangling and Vectorization Method 1

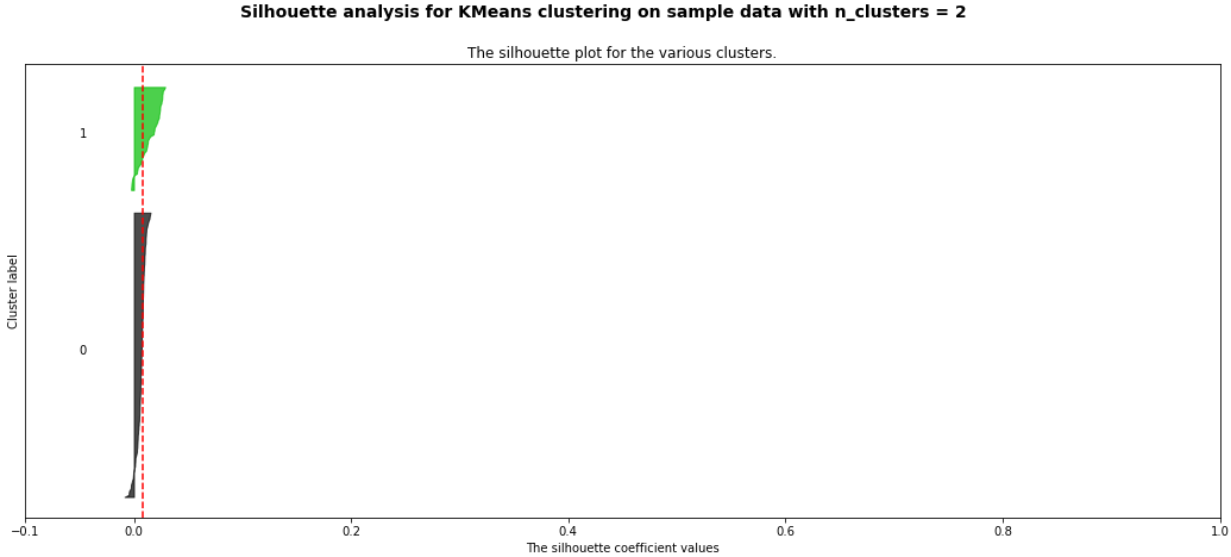
Let's conduct our first clustering experiments via K-Means Clustering to determine whether we can successfully cluster movie reviews based on review type, genre, or movie title.

In [23]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.cm as cm

# Generating the sample data from make_blobs

range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20

average_silhouette_scores = []

def k_means_cluster_count_tuner(tdm_matrix):

    for n_clusters in range_n_clusters:
        # Create a subplot with 1 row and 2 columns
        fig, ax1 = plt.subplots(1, 1)
        fig.set_size_inches(18, 7)

        # The 1st subplot is the silhouette plot
        # The silhouette coefficient can range from -1, 1 but in this example all
        # lie within [-0.1, 1]
        ax1.set_xlim([-0.1, 1])
```

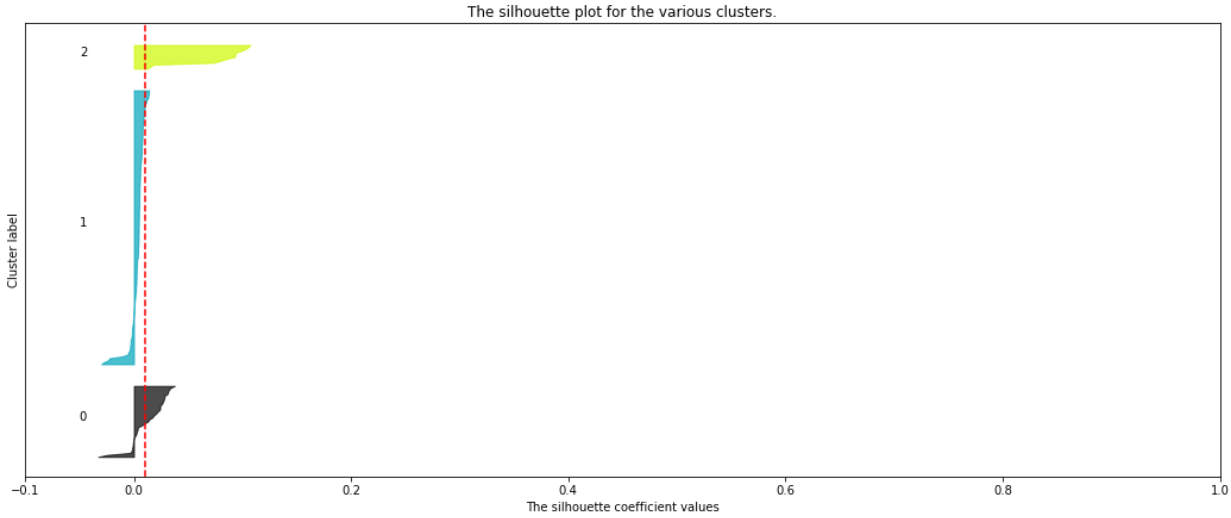
```
        # The (n_clusters+1)*10 is for inserting blank space between silhouette
        # plots of individual clusters, to demarcate them clearly.
        ax1.set_ylim([0, len(tdm_matrix) + (n_clusters + 1) * 10])

        # Initialize the clusterer with n_clusters value and a random generator
        # seed of 10 for reproducibility.
        clusterer = KMeans(n_clusters=n_clusters, n_init=10, random_state=10)
        cluster_labels = clusterer.fit_predict(tdm_matrix)

        # The silhouette_score gives the average value for all the samples.
        # This gives a perspective into the density and separation of the formed
        # clusters
        silhouette_avg = silhouette_score(tdm_matrix, cluster_labels)
        print(
            "For n_clusters =",
            n_clusters,
            "The average silhouette_score is :",
            silhouette_avg,
        )
        average_silhouette_scores.append(silhouette_avg)

        # Compute the silhouette scores for each sample
        sample_silhouette_values = silhouette_samples(tdm_matrix, cluster_labels)

        y_lower = 10
        for i in range(n_clusters):
            # Aggregate the silhouette scores for samples belonging to
            # cluster i, and sort them
            ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels ==

            ith_cluster_silhouette_values.sort()

            size_cluster_i = ith_cluster_silhouette_values.shape[0]
            y_upper = y_lower + size_cluster_i

            color = cm.nipy_spectral(float(i) / n_clusters)
            ax1.fill_betweenx(
                np.arange(y_lower, y_upper),
                0,
                ith_cluster_silhouette_values,
                facecolor=color,
                edgecolor=color,
                alpha=0.7,
            )

            # Label the silhouette plots with their cluster numbers at the middle
            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

            # Compute the new y_lower for next plot
            y_lower = y_upper + 10  # 10 for the 0 samples

        ax1.set_title("The silhouette plot for the various clusters.")
        ax1.set_xlabel("The silhouette coefficient values")
        ax1.set_ylabel("Cluster label")

        # The vertical line for average silhouette score of all the values
        ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

        ax1.set_yticks([])  # Clear the yaxis labels / ticks
        ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
```

```
        plt.suptitle(
            "Silhouette analysis for KMeans clustering on sample data with n_clusters
            % n_clusters,
            fontsize=14,
            fontweight="bold",
        )

    plt.show()


k_means_cluster_count_tuner(tfidf_matrix_method_one)
```

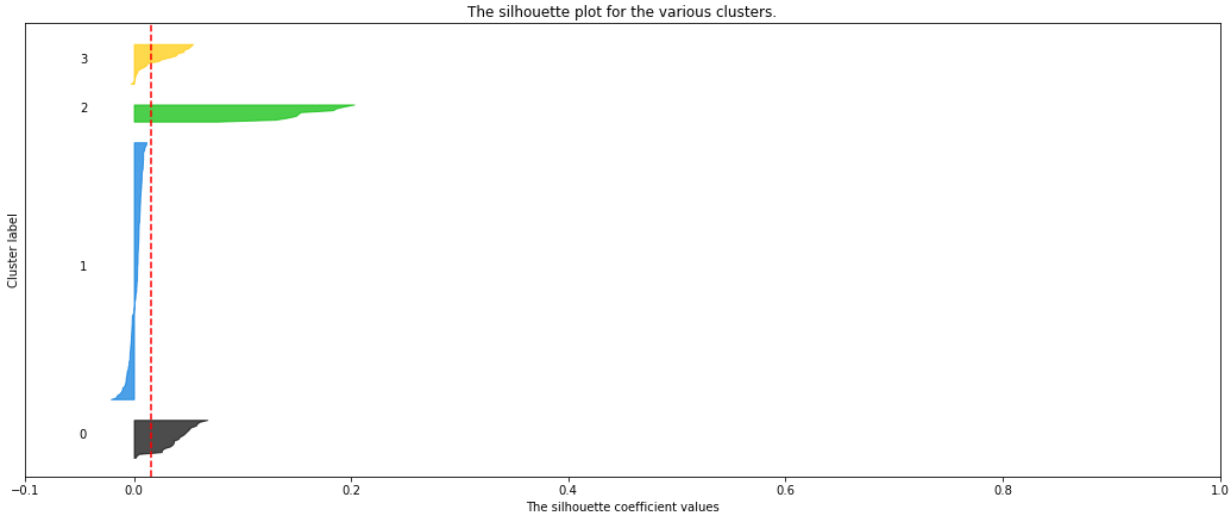
```
For n_clusters = 2 The average silhouette_score is : 0.008285619416885338
For n_clusters = 3 The average silhouette_score is : 0.00988616929557615
For n_clusters = 4 The average silhouette_score is : 0.015725629786719462
For n_clusters = 5 The average silhouette_score is : 0.017073801784710924
For n_clusters = 6 The average silhouette_score is : 0.021126492433763112
For n_clusters = 7 The average silhouette_score is : 0.02960674075362221
For n_clusters = 8 The average silhouette_score is : 0.0320510702712986
For n_clusters = 9 The average silhouette_score is : 0.03960679713007109
For n_clusters = 10 The average silhouette_score is : 0.04474203576566233
For n_clusters = 11 The average silhouette_score is : 0.04571236252636947
For n_clusters = 12 The average silhouette_score is : 0.04554585032533766
For n_clusters = 13 The average silhouette_score is : 0.05122041721839052
For n_clusters = 14 The average silhouette_score is : 0.05631300884489371
For n_clusters = 15 The average silhouette_score is : 0.06373076762616592
For n_clusters = 16 The average silhouette_score is : 0.0668775824186552
For n_clusters = 17 The average silhouette_score is : 0.06571610012705084
For n_clusters = 18 The average silhouette_score is : 0.06751060460450785
For n_clusters = 19 The average silhouette_score is : 0.07484495173737643
For n_clusters = 20 The average silhouette_score is : 0.06596922345391097
For n_clusters = 21 The average silhouette_score is : 0.07182063186313954
For n_clusters = 22 The average silhouette_score is : 0.0692696587832053
For n_clusters = 23 The average silhouette_score is : 0.06966303826380454
For n_clusters = 24 The average silhouette_score is : 0.06566611527590435
For n_clusters = 25 The average silhouette_score is : 0.06795071150410006
```



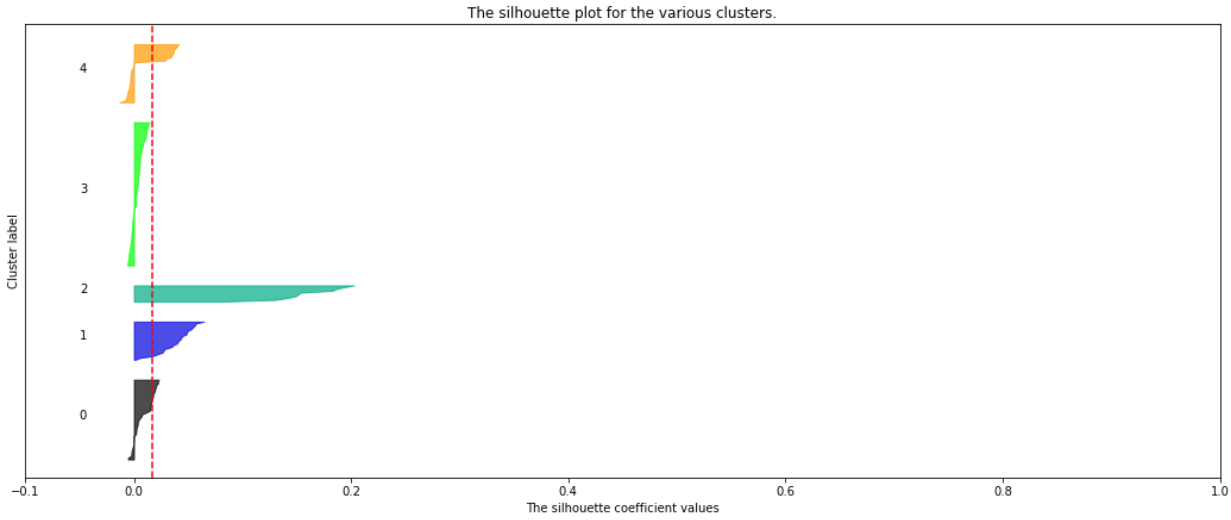
Silhouette analysis for KMeans clustering on sample data with n_clusters = 3

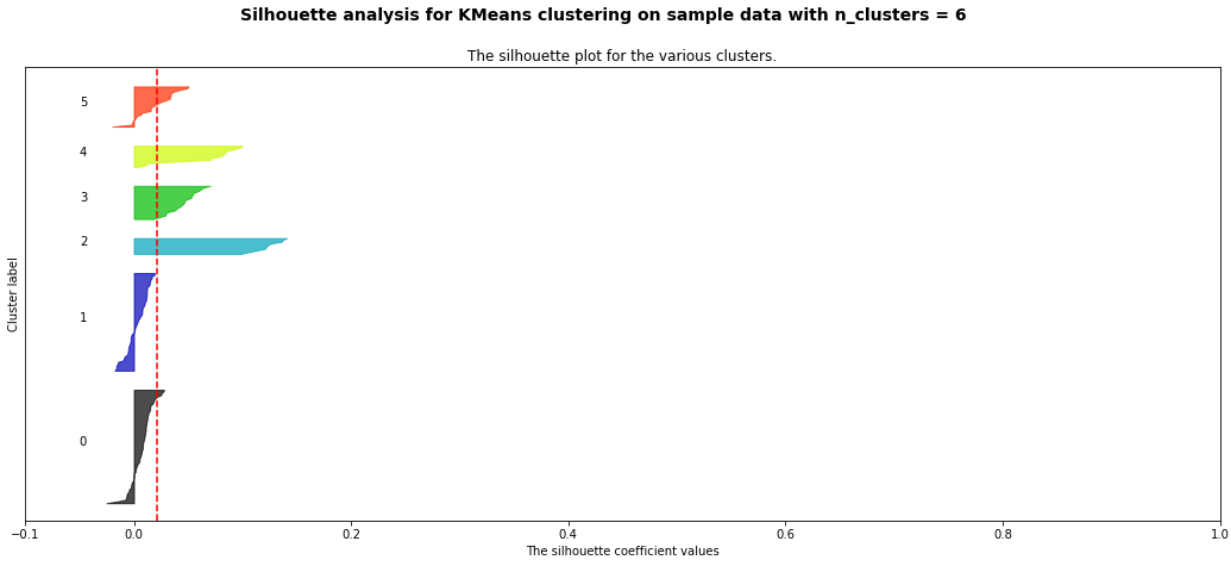


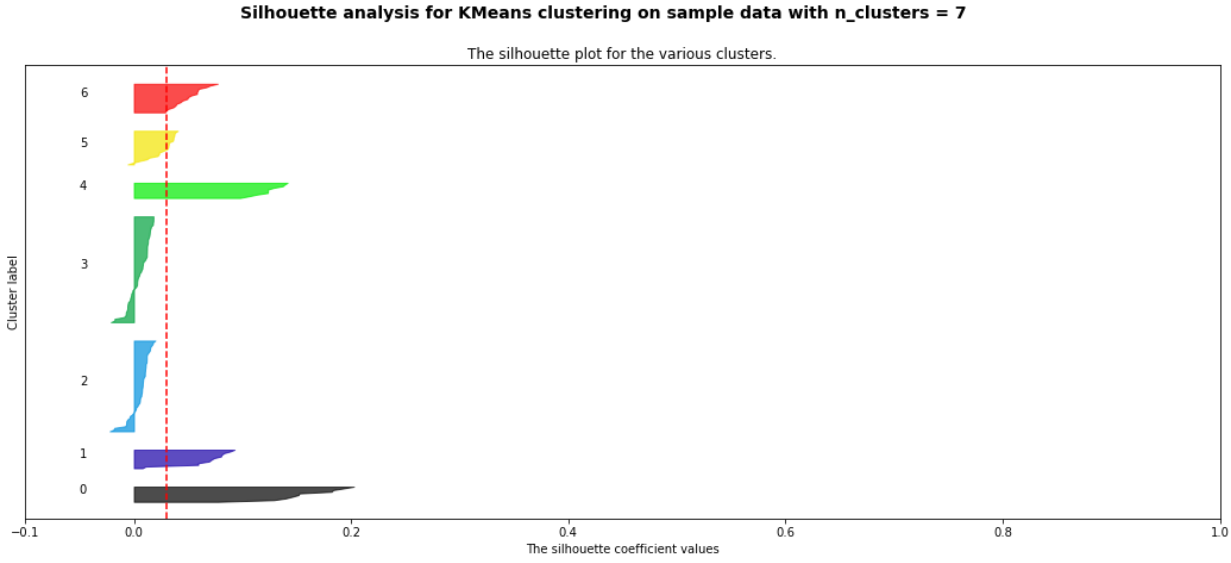
Silhouette analysis for KMeans clustering on sample data with n_clusters = 4

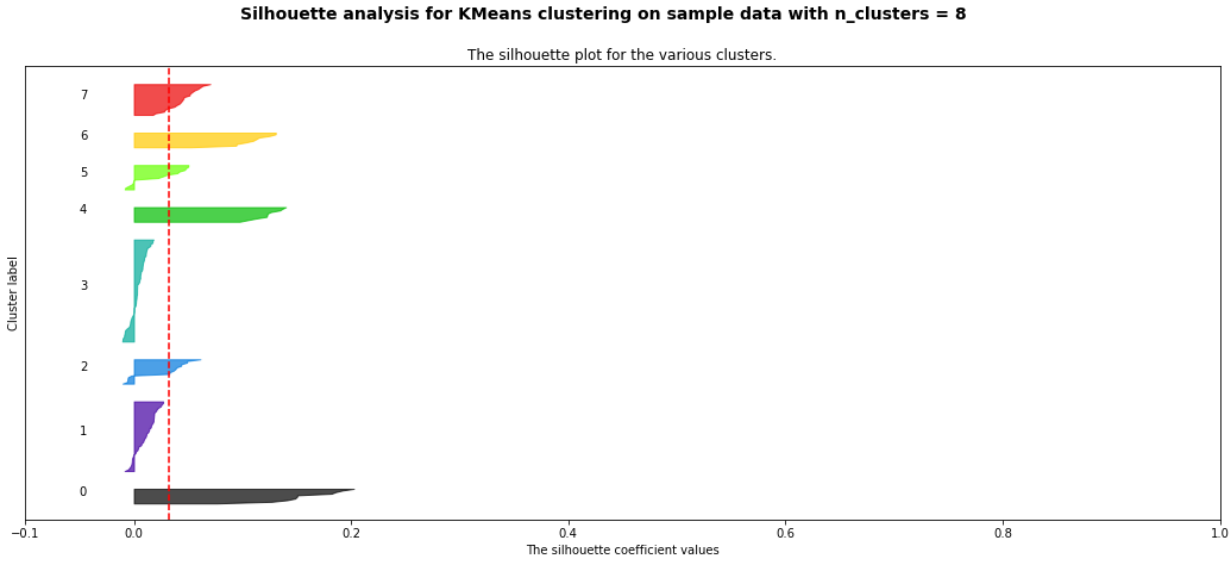


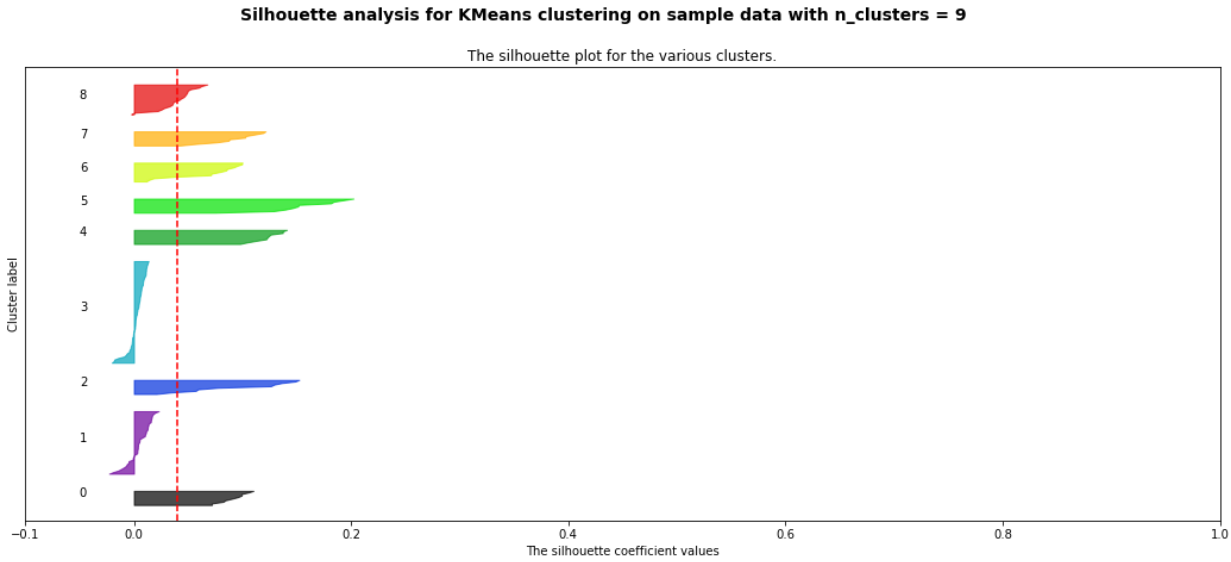
Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

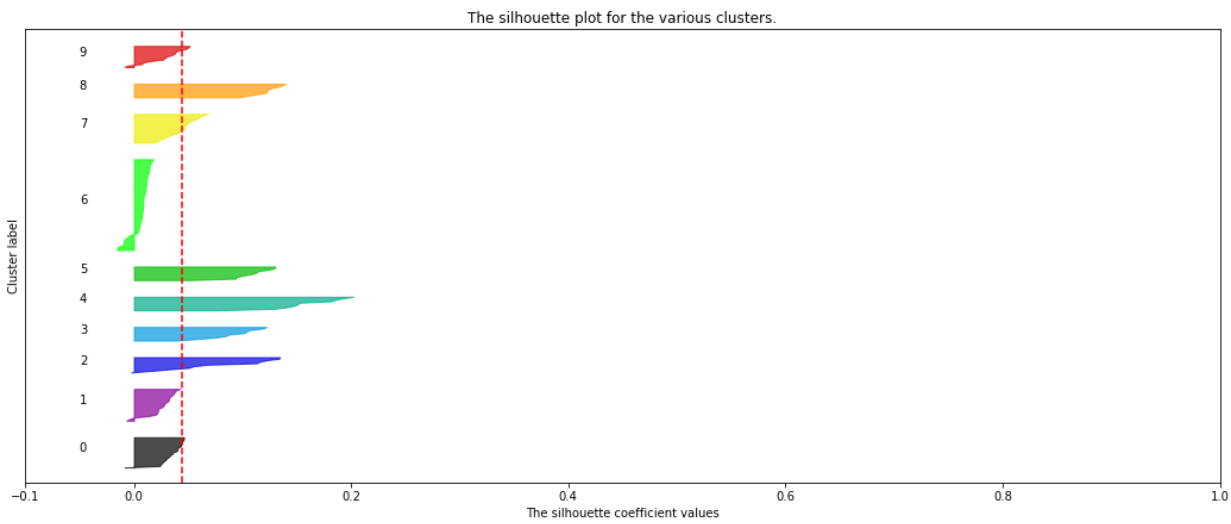


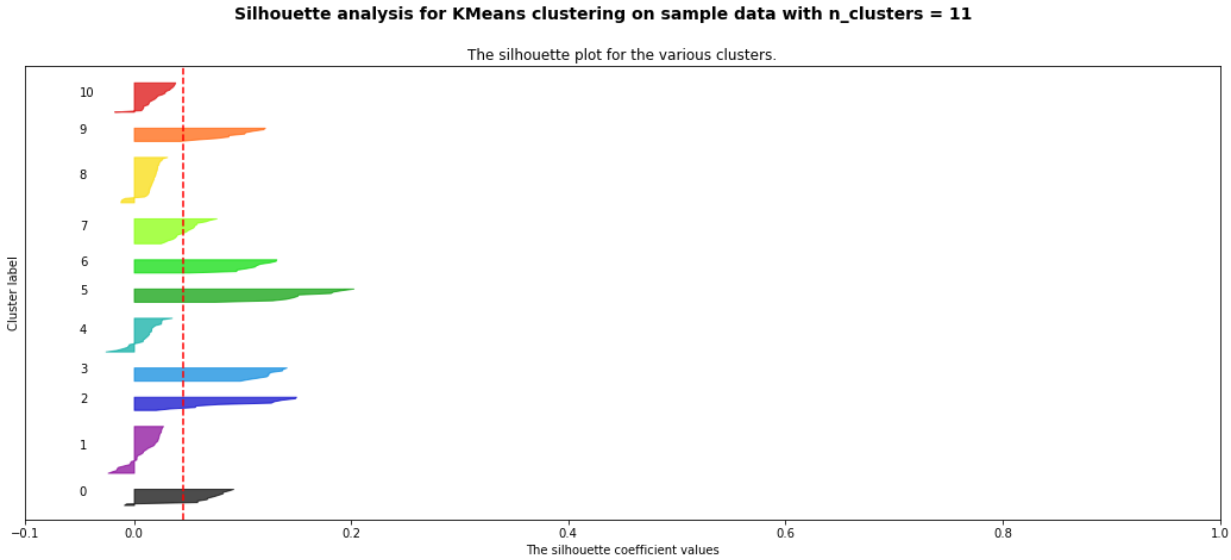

Silhouette analysis for KMeans clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
Cluster label
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

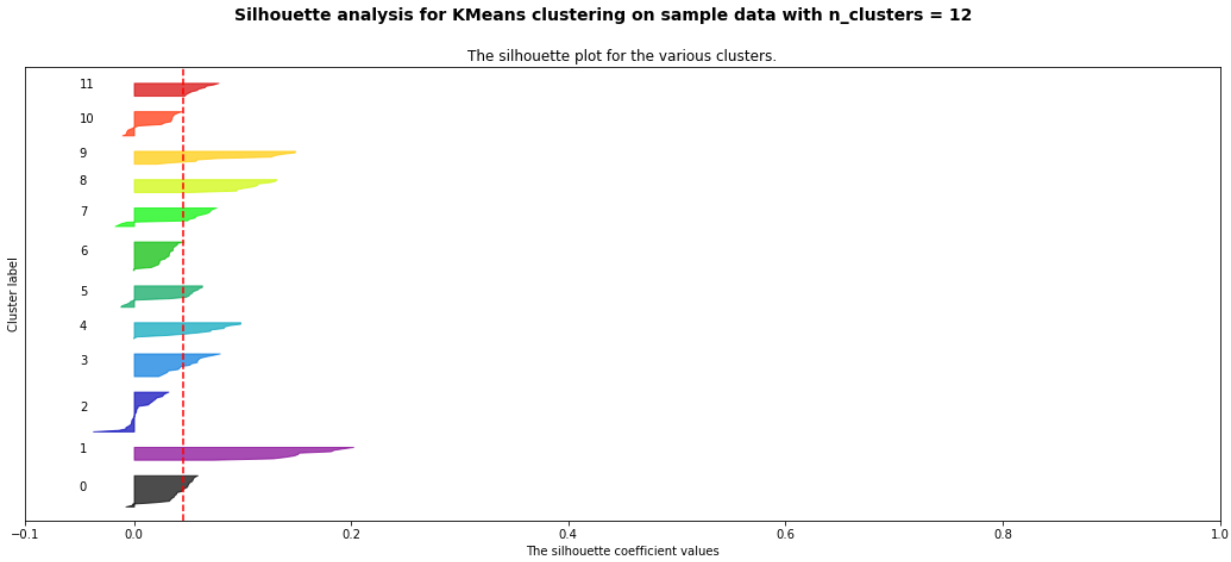

Silhouette analysis for KMeans clustering on sample data with n_clusters = 7
The silhouette plot for the various clusters.
Cluster label
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

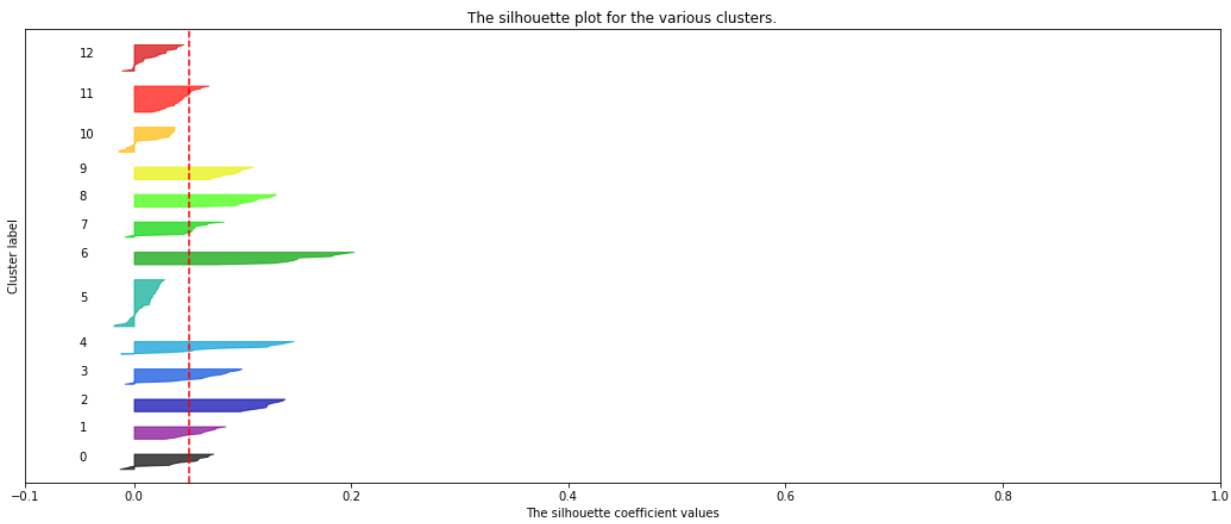

Silhouette analysis for KMeans clustering on sample data with n_clusters = 8
The silhouette plot for the various clusters.
Cluster label
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

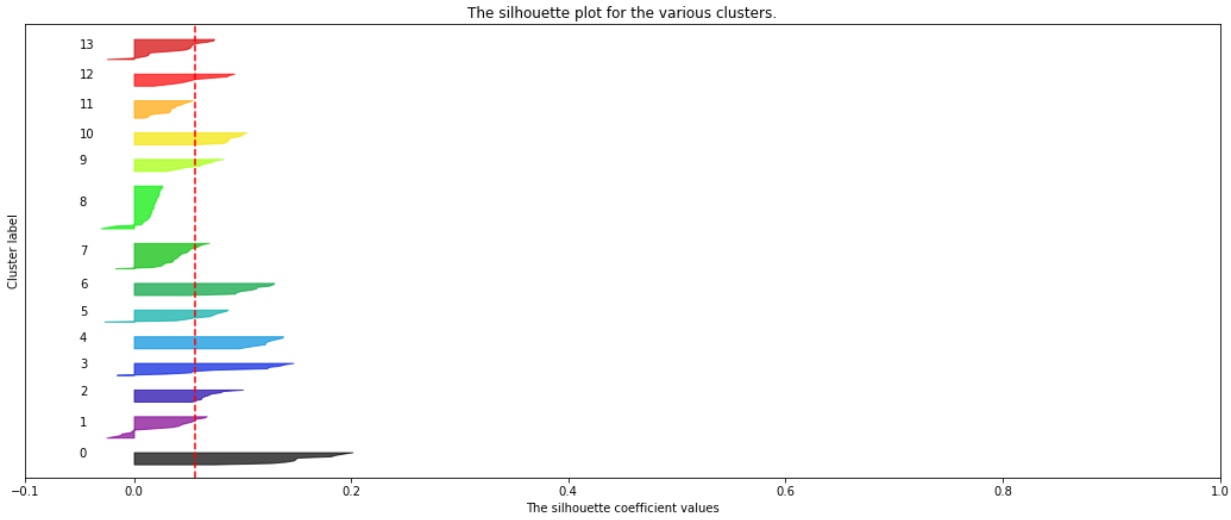

Silhouette analysis for KMeans clustering on sample data with n_clusters = 9
The silhouette plot for the various clusters.
Cluster label
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 10
The silhouette plot for the various clusters.
Cluster label
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

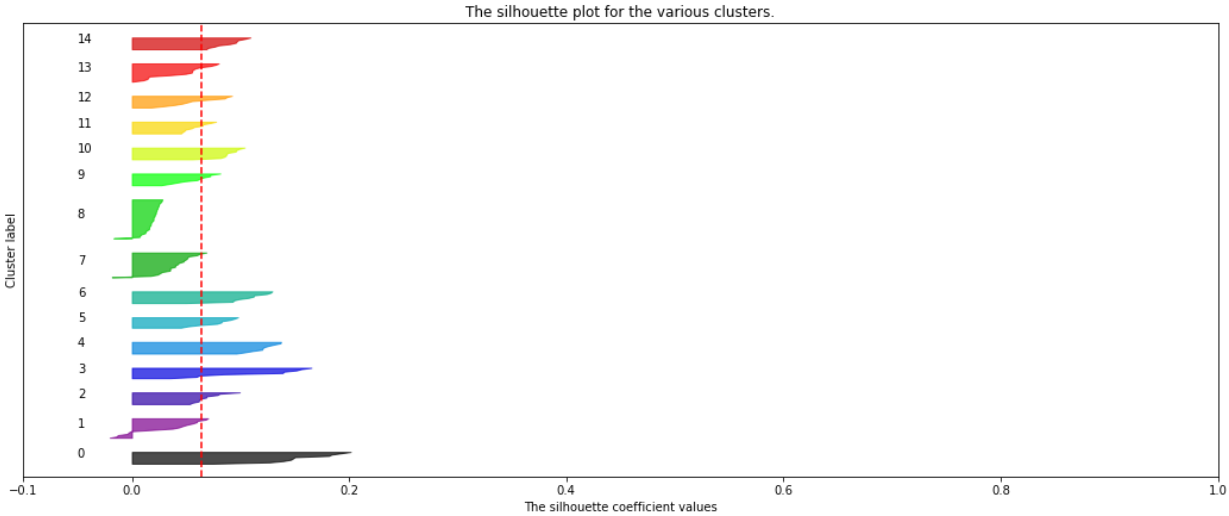

Silhouette analysis for KMeans clustering on sample data with n_clusters = 11
The silhouette plot for the various clusters.
Cluster label
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 12
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

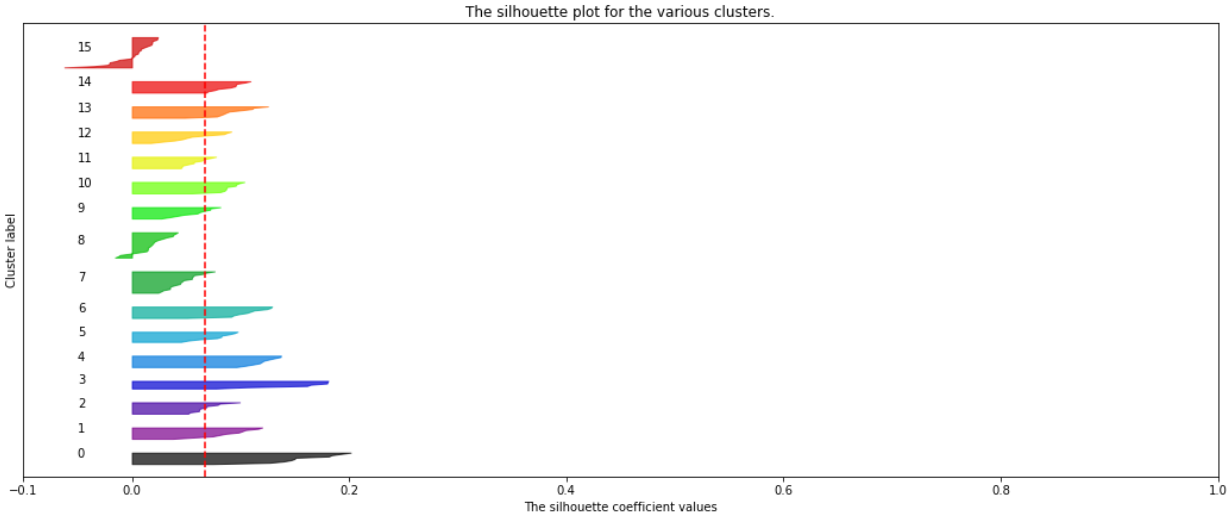

Silhouette analysis for KMeans clustering on sample data with n_clusters = 13
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


Silhouette analysis for KMeans clustering on sample data with n_clusters = 14
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
13
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

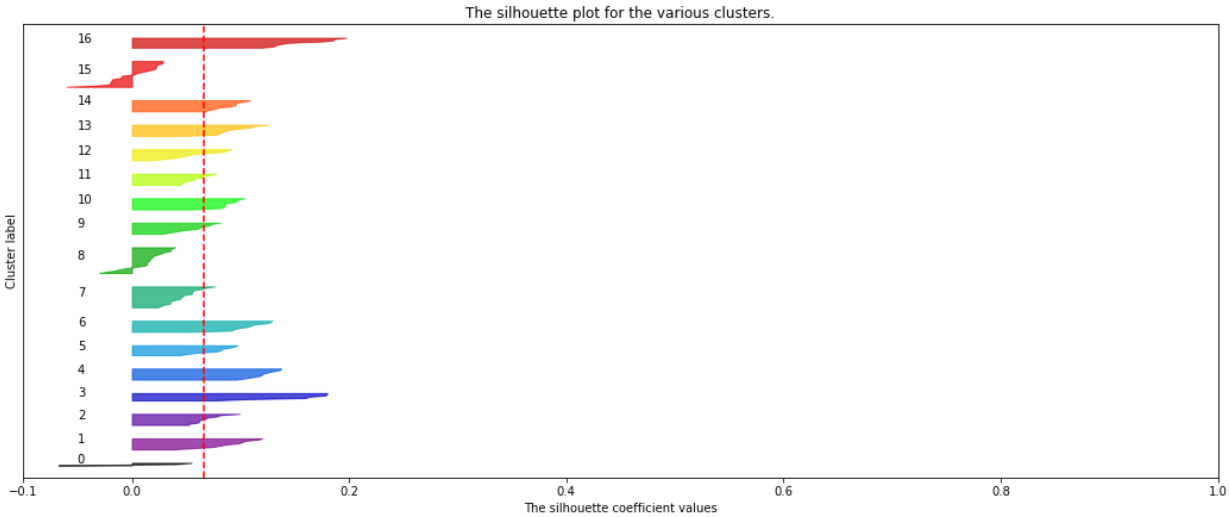
Silhouette analysis for KMeans clustering on sample data with n_clusters = 15



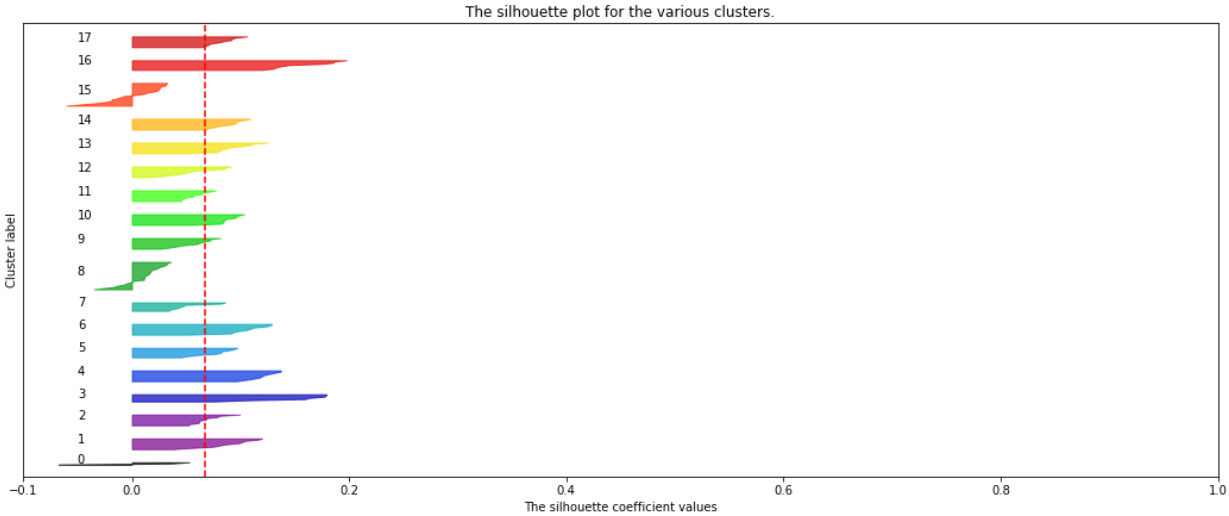
Silhouette analysis for KMeans clustering on sample data with n_clusters = 16



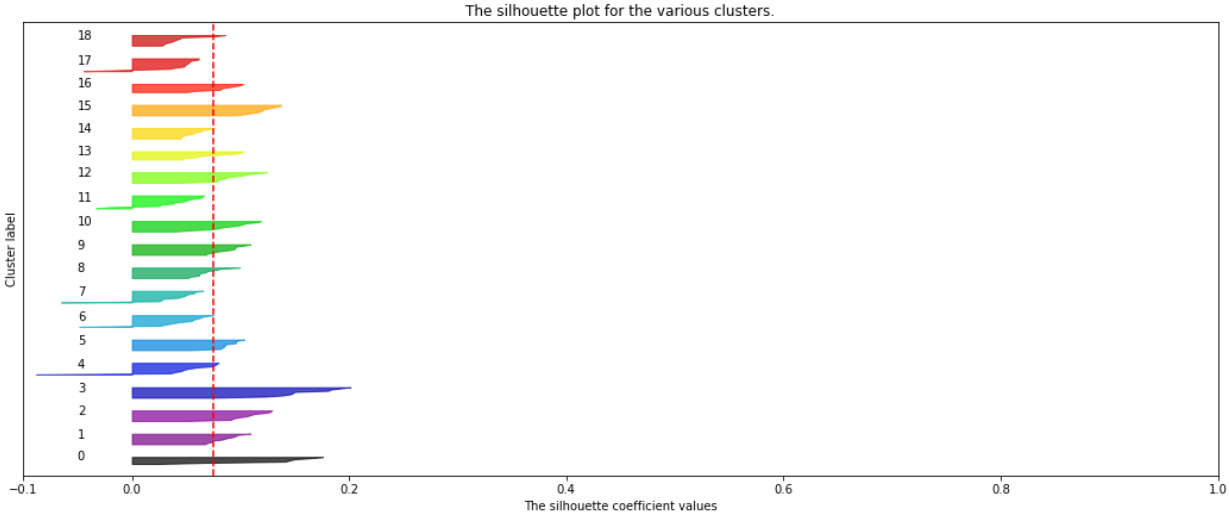
Silhouette analysis for KMeans clustering on sample data with n_clusters = 17



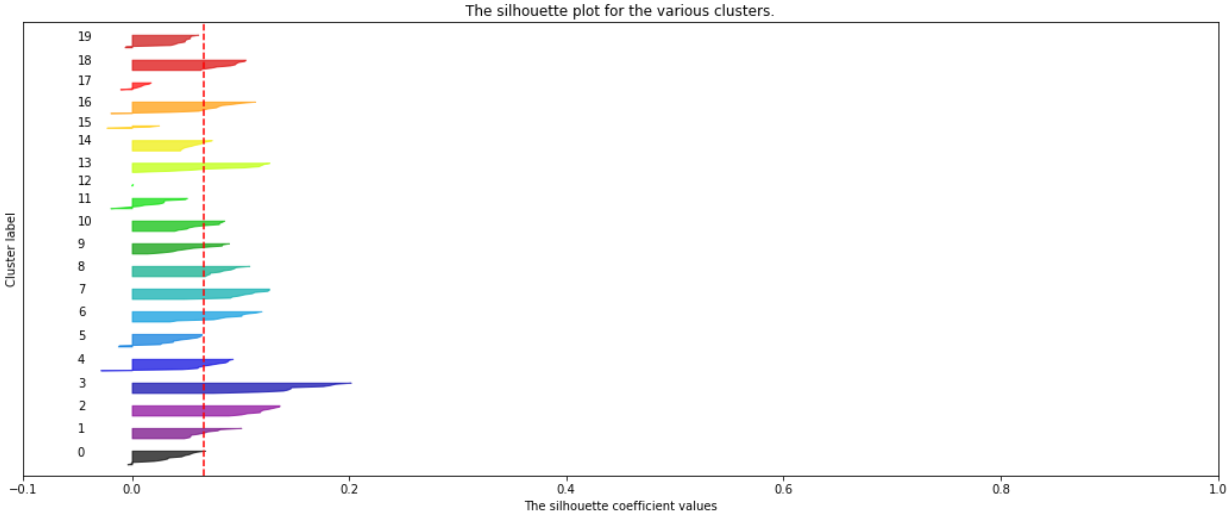
Silhouette analysis for KMeans clustering on sample data with n_clusters = 18

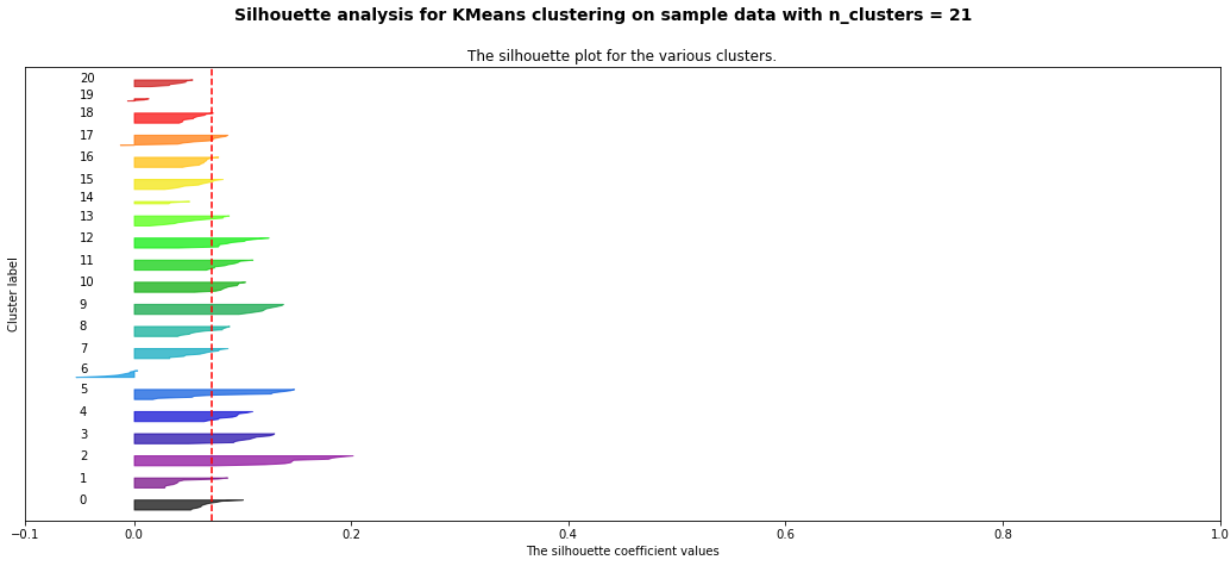


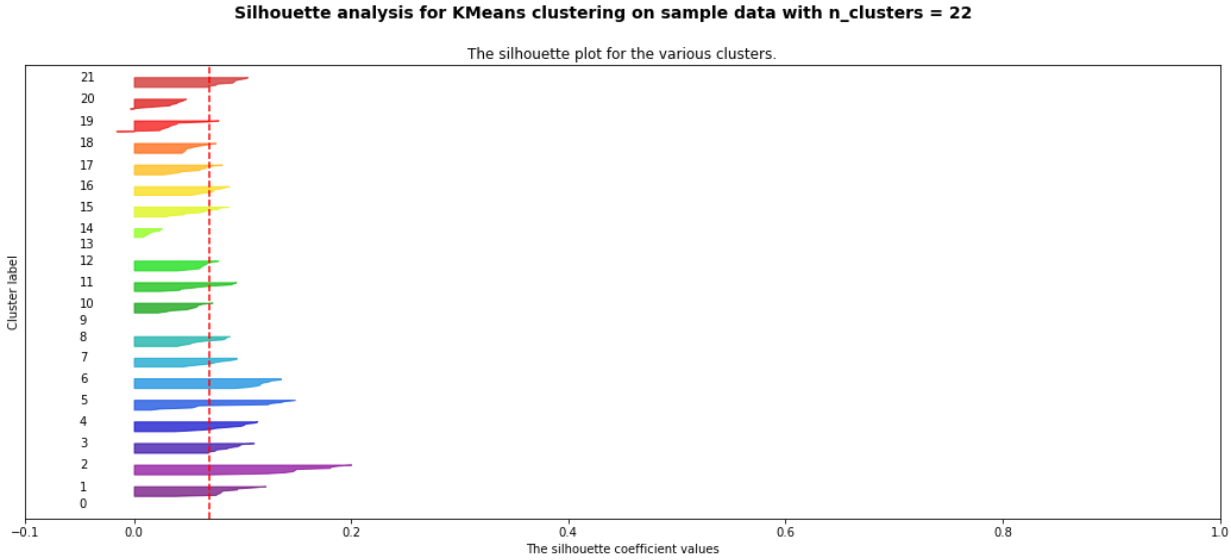
Silhouette analysis for KMeans clustering on sample data with n_clusters = 19

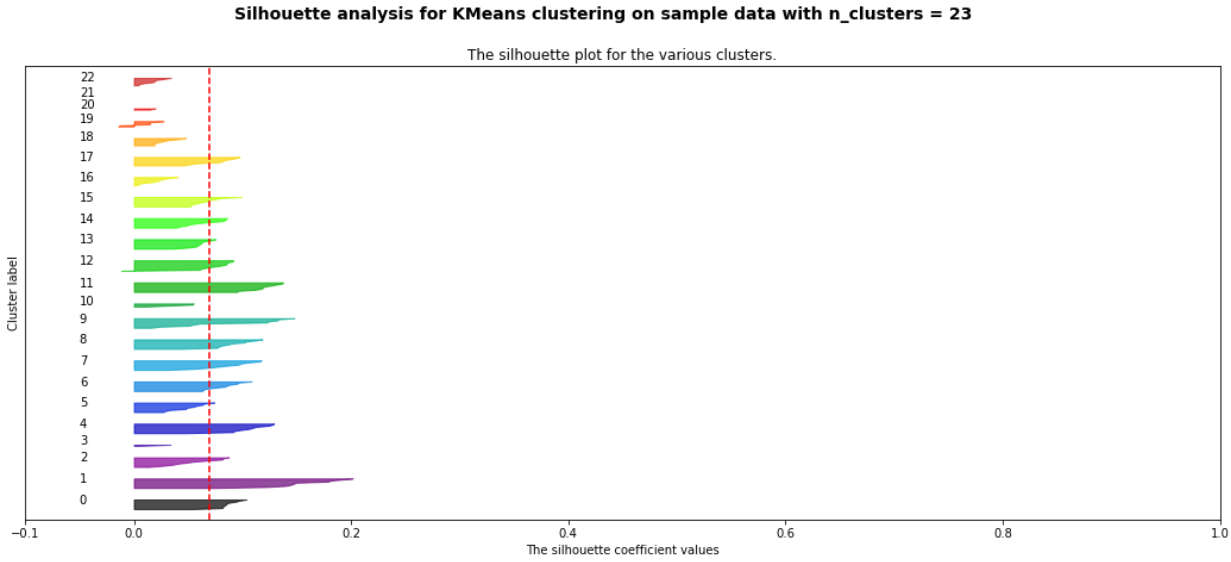


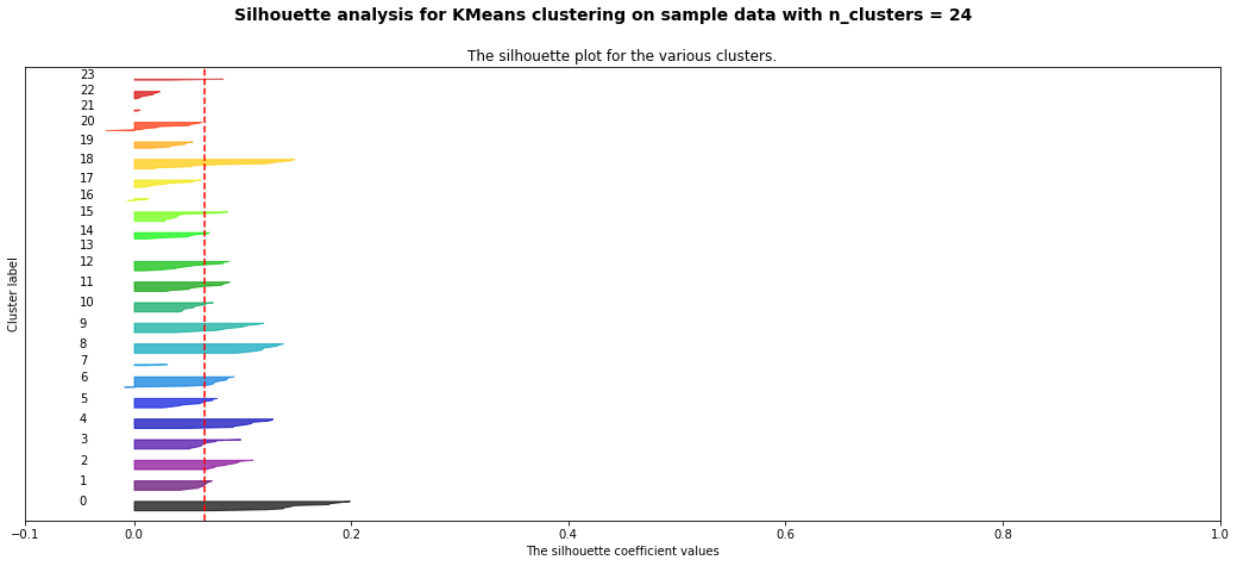
Silhouette analysis for KMeans clustering on sample data with n_clusters = 20

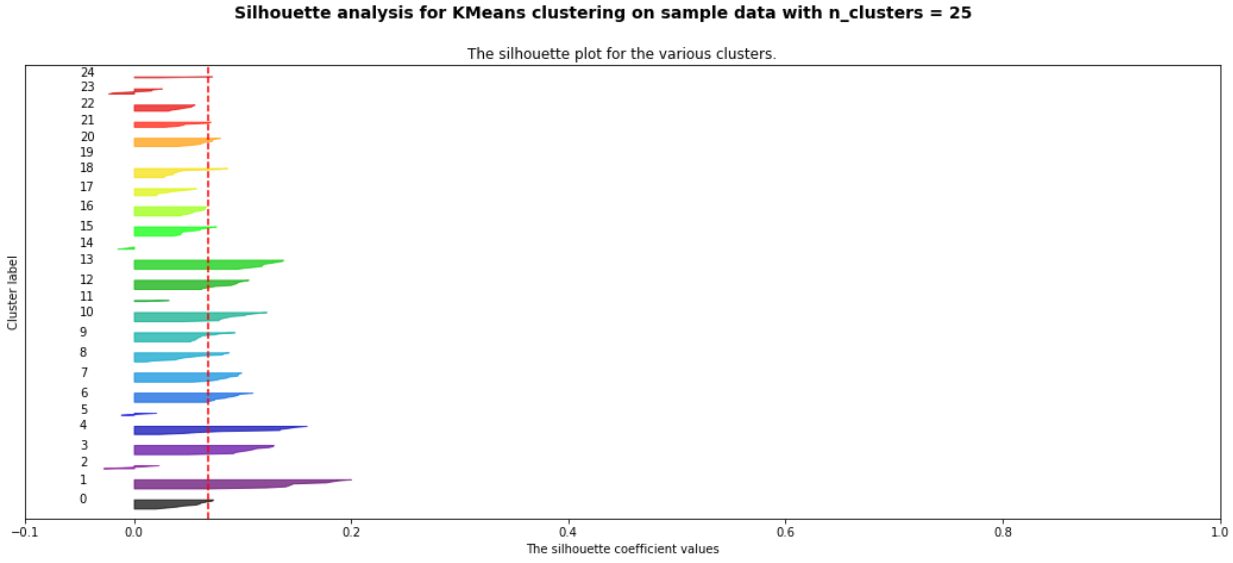



Silhouette analysis for KMeans clustering on sample data with n_clusters = 21
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

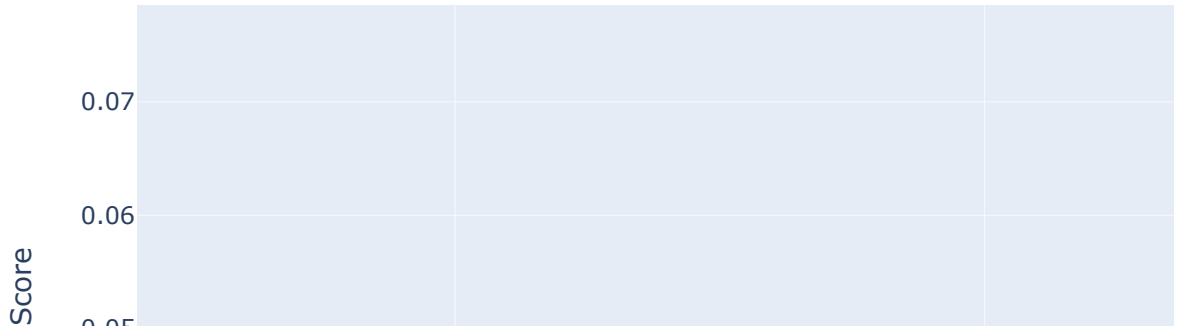

Silhouette analysis for KMeans clustering on sample data with n_clusters = 22
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 23
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values





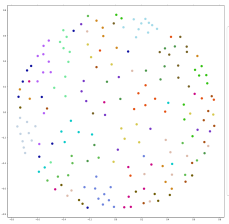
In [24]:
```
import plotly.express as px

avg_silhouette_score_dic = {'Number of Clusters':range_n_clusters, 'Average Silhouette
avg_silhouette_score_df = pd.DataFrame(avg_silhouette_score_dic)


fig = px.line(avg_silhouette_score_df, x = "Number of Clusters", y = "Average Silhouet
              title = "Average Silhouette Score By Number of Clusters")

fig.update_layout(height = 600, xaxis_title = 'Number of Clusters')
```



In [25]:
```
def k_means(titles, tdm_matrix, k, processed_text):

    #this is a function to generate the k-means output using the tfidf matrix.  Inputs
    #to the function include: titles of text, processed text, and desired k value.
    #Returns dataframe indicating cluster number per document

    km = KMeans(n_clusters=k, random_state =89)
    km.fit(tdm_matrix)
    clusters = km.labels_.tolist()

    Dictionary={'Doc Name':titles, 'Cluster':clusters,  'Text': processed_text}
    frame=pd.DataFrame(Dictionary, columns=['Cluster', 'Doc Name','Text'])
    #dictionary to store clusters and respective titles
    cluster_title={}

    #note doc2vec clusters will not have individual words due to the vector representa
    #is based on the entire document not indvidual words. As a result, there won't be
    #word outputs from each cluster.
    for i in range(k):
        temp=frame[frame['Cluster']==i]
        temp_title_list=[]
```

```
        for title in temp['Doc Name']:
            temp_title_list.append(title)
        cluster_title[i]=temp_title_list

    return cluster_title,clusters,frame


def plot_tfidf_matrix(cluster_title, clusters, TFIDF_matrix):
    # convert two components as we're plotting points in a two-dimensional plane
    # "precomputed" because we provide a distance matrix
    # we will also specify `random_state` so the plot is reproducible.

    mds = MDS(n_components=2, dissimilarity="precomputed", random_state=1)
    dist = 1 - cosine_similarity(TFIDF_matrix)
    pos = mds.fit_transform(dist)  # shape (n_components, n_samples)
    xs, ys = pos[:, 0], pos[:, 1]


    #set up colors per clusters using a dict.  number of colors must correspond to K
    cluster_colors = {0: 'black', 1: 'grey', 2: 'blue', 3: 'rosybrown', 4: 'firebrick'
                      5:'red', 6:'darksalmon', 7:'sienna', 8:'darkorange3', 9:'darkorc
                     10:'seagreen2', 11:'deeppink2', 12:'gold2', 13:'indigo',
                     14:'olive', 15:'sepia', 16:'slateblue2', 17:'maroon',
                     18:'aqua'}

    #set up cluster names using a dict.
    cluster_dict=cluster_title

    #create data frame that has the result of the MDS plus the cluster numbers and tit
    df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=range(0,len(clusters))))

    #group by cluster
    groups = df.groupby('label')

    fig, ax = plt.subplots(figsize=(20,20)) # set size
    ax.margins(0.05) # Optional, just adds 5% padding to the autoscaling

    #iterate through groups to layer the plot
    #note that I use the cluster_name and cluster_color dicts with the 'name' lookup t
    for name, group in groups:

        r = random.random()
        b = random.random()
        g = random.random()
        color = (r, g, b)

        ax.plot(group.x, group.y, marker='o', linestyle='', ms=12,
                label=cluster_dict[name], color=color,
                mec='none')
        ax.set_aspect('auto')
        ax.tick_params(\
            axis= 'x',          # changes apply to the x-axis
            which='both',      # both major and minor ticks are affected
            bottom='off',      # ticks along the bottom edge are off
            top='off',         # ticks along the top edge are off
            labelbottom='on')
        ax.tick_params(\
            axis= 'y',         # changes apply to the y-axis
            which='both',      # both major and minor ticks are affected
            left='off',      # ticks along the bottom edge are off
```

```
            top='off',         # ticks along the top edge are off
            labelleft='on')

    ax.legend(loc='center left', bbox_to_anchor=(1, 0.5), prop={'size': 30})      #sho



#tfidf_matrix = tfidf(final_processed_text, titles, ngram_range = (1,1))

cluster_title, clusters, k_means_df = k_means(titles,
                                              tdm_matrix = tfidf_matrix_method_one,
                                              k=19,
                                              processed_text = final_processed_text_me

cluster_title[9]

plot_tfidf_matrix(cluster_title, clusters, tfidf_matrix_method_one)
```



```
['DIV_Doc1_MinorityReport', 'CYY_Doc6_Oblivion', 'CYY_Doc7_Oblivion', 'CYY_Doc8_Oblivion', 'CYY_Doc9_Oblivion', 'CYY_Doc10_Oblivion', 'CYY_Doc1_Oblivion', 'CYY_Doc2_Oblivion', 'CYY_Doc3_Oblivion', 'CYY_Doc4_Oblivion', 'CYY_Doc5_Oblivion']
['GRY_Doc6_TheDarkKnightRises', 'GRY_Doc7_TheDarkKnightRises', 'GRY_Doc8_TheDarkKnightRises', 'GRY_Doc9_TheDarkKnightRises', 'GRY_Doc10_TheDarkKnightRises', 'GRY_Doc1_TheDarkKnightRises', 'GRY_Doc2_TheDarkKnightRises', 'GRY_Doc3_TheDarkKnightRises', 'GRY_Doc4_TheDarkKnightRises', 'GRY_Doc5_TheDarkKnightRises']
['SAR_Doc1_Covenant', 'SAR_Doc2_Covenant', 'SAR_Doc3_Covenant', 'SAR_Doc4_Covenant', 'SAR_Doc5_Covenant', 'SAR_Doc6_Covenant', 'SAR_Doc7_Covenant', 'SAR_Doc8_Covenant', 'SAR_Doc9_Covenant', 'SAR_Doc10_Covenant']
['AWM_Doc6_Fresh', 'AWM_Doc7_Fresh', 'AWM_Doc8_Fresh', 'AWM_Doc9_Fresh', 'AWM_Doc10_Fresh', 'AWM_Doc1_Fresh', 'AWM_Doc2_Fresh', 'AWM_Doc3_Fresh', 'AWM_Doc4_Fresh', 'AWM_Doc5_Fresh']
['SJD_Doc6_Sisters', 'SJD_Doc7_Sisters', 'SJD_Doc8_Sisters', 'SJD_Doc9_Sisters', 'SJD_Doc10_Sisters', 'SJD_Doc1_Sisters', 'SJD_Doc2_Sisters', 'SJD_Doc3_Sisters', 'SJD_Doc4_Sisters', 'SJD_Doc5_Sisters']
['MLS_Doc6_ItChapterTwo', 'MLS_Doc7_ItChapterTwo', 'MLS_Doc8_ItChapterTwo', 'MLS_Doc9_ItChapterTwo', 'MLS_Doc10_ItChapterTwo', 'MLS_Doc1_ItChapterTwo', 'MLS_Doc2_ItChapterTwo', 'MLS_Doc3_ItChapterTwo', 'MLS_Doc4_ItChapterTwo', 'MLS_Doc5_ItChapterTwo']
['SRG_Doc6_TheToxicAvenger', 'SRG_Doc7_TheToxicAvenger', 'SRG_Doc8_TheToxicAvenger', 'SRG_Doc9_TheToxicAvenger', 'SRG_Doc10_TheToxicAvenger', 'SRG_Doc1_TheToxicAvenger', 'SRG_Doc2_TheToxicAvenger', 'SRG_Doc3_TheToxicAvenger', 'SRG_Doc4_TheToxicAvenger', 'SRG_Doc5_TheToxicAvenger']
['AEK_Doc6_LegallyBlonde', 'AEK_Doc7_LegallyBlonde', 'AEK_Doc8_LegallyBlonde', 'AEK_Doc9_LegallyBlonde', 'AEK_Doc10_LegallyBlonde', 'AEK_Doc1_LegallyBlonde', 'AEK_Doc2_LegallyBlonde', 'AEK_Doc3_LegallyBlonde', 'AEK_Doc4_LegallyBlonde', 'AEK_Doc5_LegallyBlonde']
['JCD_Doc6_DespicableMe3', 'JCD_Doc7_DespicableMe3', 'JCD_Doc8_DespicableMe3', 'JCD_Doc9_DespicableMe3', 'JCD_Doc10_DespicableMe3', 'JCD_Doc1_DespicableMe3', 'JCD_Doc2_DespicableMe3', 'JCD_Doc3_DespicableMe3', 'JCD_Doc4_DespicableMe3', 'JCD_Doc5_DespicableMe3']
['CLF_Doc6_LostCity', 'CLF_Doc7_LostCity', 'CLF_Doc8_LostCity', 'CLF_Doc9_LostCity', 'CLF_Doc10_LostCity', 'CLF_Doc1_LostCity', 'CLF_Doc2_LostCity', 'CLF_Doc3_LostCity', 'CLF_Doc4_LostCity', 'CLF_Doc5_LostCity', 'NTF_Doc8_US']
['OSO_Doc1_PitchBlack', 'OSO_Doc2_PitchBlack', 'OSO_Doc6_PitchBlack', 'OSO_Doc9_PitchBlack', 'OSO_Doc10_PitchBlack', 'OSO_Doc3_PitchBlack', 'OSO_Doc4_PitchBlack', 'OSO_Doc5_PitchBlack', 'OSO_Doc6_PitchBlack', 'OSO_Doc7_PitchBlack']
['GLZ_Doc6_Inception', 'GLZ_Doc7_Inception', 'GLZ_Doc8_Inception', 'GLZ_Doc9_Inception', 'GLZ_Doc10_Inception', 'GLZ_Doc1_Inception', 'GLZ_Doc2_Inception', 'GLZ_Doc3_Inception', 'GLZ_Doc4_Inception', 'GLZ_Doc5_Inception']
['KTT_Doc6_DragMetoHell', 'KTT_Doc7_DragMetoHell', 'KTT_Doc8_DragMetoHell', 'KTT_Doc9_DragMetoHell', 'KTT_Doc10_DragMetoHell', 'KTT_Doc1_DragMetoHell', 'KTT_Doc2_DragMetoHell', 'KTT_Doc3_DragMetoHell', 'KTT_Doc4_DragMetoHell', 'KTT_Doc5_DragMetoHell']
['TMW_Doc6_Annihilation', 'TMW_Doc7_Annihilation', 'TMW_Doc8_Annihilation', 'TMW_Doc9_Annihilation', 'TMW_Doc10_Annihilation', 'TMW_Doc1_Annihilation', 'TMW_Doc2_Annihilation', 'TMW_Doc3_Annihilation', 'TMW_Doc4_Annihilation', 'TMW_Doc5_Annihilation']
['BJY_Doc6_Taken', 'BJY_Doc7_Taken', 'BJY_Doc8_Taken', 'BJY_Doc9_Taken', 'BJY_Doc10_Taken', 'BJY_Doc1_Taken', 'BJY_Doc2_Taken', 'BJY_Doc3_Taken', 'BJY_Doc4_Taken', 'BJY_Doc5_Taken']
['JLC_Doc1_HolmesAndWatson', 'JLC_Doc2_HolmesAndWatson', 'JLC_Doc3_HolmesAndWatson', 'JLC_Doc4_HolmesAndWatson', 'JLC_Doc5_HolmesAndWatson', 'JLC_Doc6_HolmesAndWatson', 'JLC_Doc7_HolmesAndWatson', 'JLC_Doc8_HolmesAndWatson', 'JLC_Doc9_HolmesAndWatson', 'JLC_Doc10_HolmesAndWatson']
['NTF_Doc6_US', 'NTF_Doc7_US', 'NTF_Doc9_US', 'NTF_Doc10_US', 'NTF_Doc1_US', 'NTF_Doc2_US', 'NTF_Doc3_US', 'NTF_Doc4_US', 'NTF_Doc5_US']
['TJH_Doc6_NoTimeToDie', 'TJH_Doc7_NoTimeToDie', 'TJH_Doc8_NoTimeToDie', 'TJH_Doc9_NoTimeToDie', 'TJH_Doc10_NoTimeToDie', 'TJH_Doc1_NoTimeToDie', 'TJH_Doc2_NoTimeToDie', 'TJH_Doc3_NoTimeToDie', 'TJH_Doc4_NoTimeToDie', 'TJH_Doc5_NoTimeToDie']
['DIV_Doc3_MinorityReport', 'DIV_Doc5_MinorityReport', 'DIV_Doc9_MinorityReport', 'DIV_Doc10_MinorityReport', 'DIV_Doc2_MinorityReport', 'DIV_Doc4_MinorityReport', 'DIV_Doc6_MinorityReport', 'DIV_Doc7_MinorityReport', 'DIV_Doc8_MinorityReport']
```

In [26]:

```
review_type_labels = corpus_df['Review Type (pos or neg)'].apply(lambda x: 0 if x.lowe
print(review_type_labels)
```

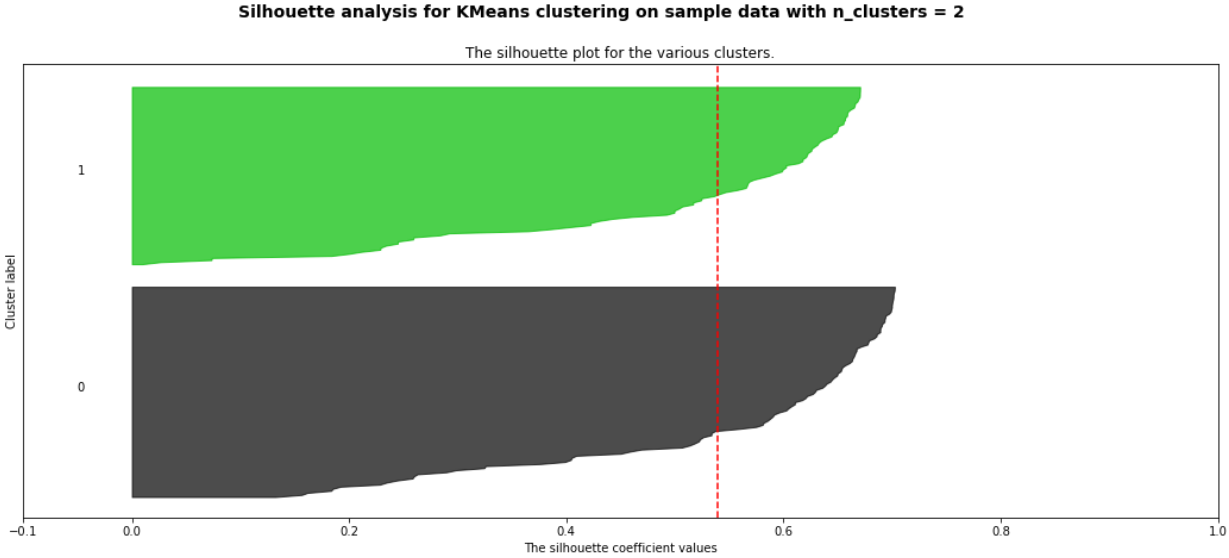
```
Doc_ID
101    0
102    0
103    0
104    0
105    0
      ..
213    1
214    1
215    1
216    1
217    1
Name: Review Type (pos or neg), Length: 190, dtype: int64
```

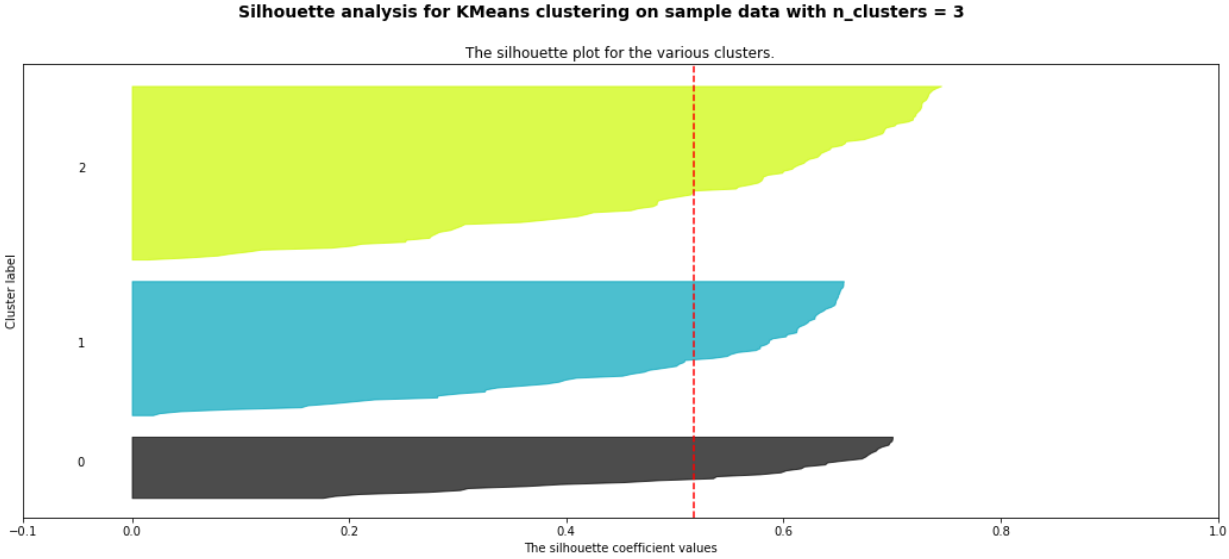
Clustering Experiment 2: K-Means Clustering with Data Wrangling and Vectorization Method 2

In [146…

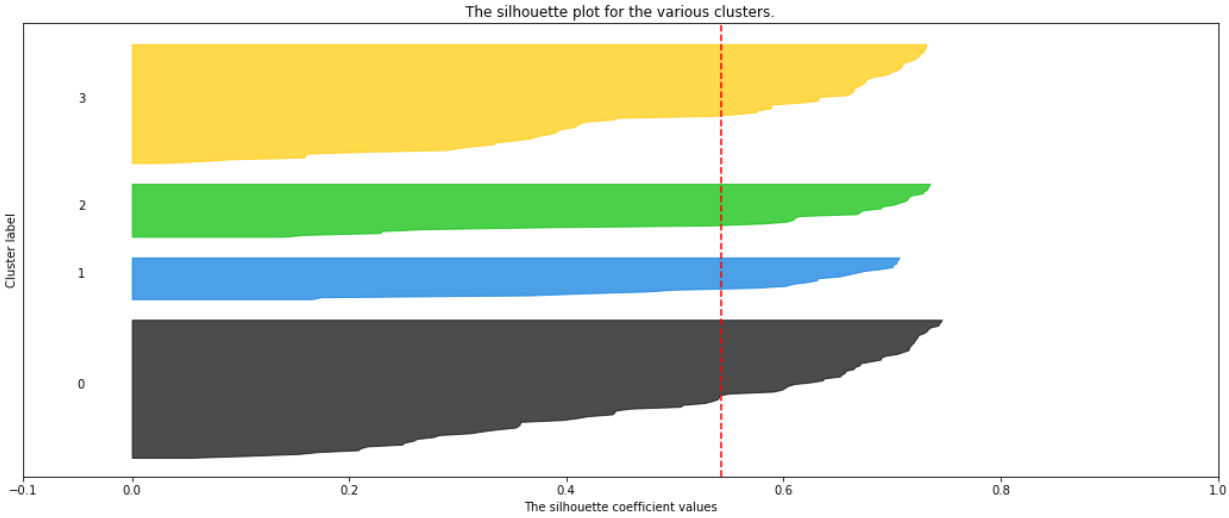
```
average_silhouette_scores = []

k_means_cluster_count_tuner(doc2vec_df)
```

```
For n_clusters = 2 The average silhouette_score is : 0.5394829
For n_clusters = 3 The average silhouette_score is : 0.5176463
For n_clusters = 4 The average silhouette_score is : 0.5427742
For n_clusters = 5 The average silhouette_score is : 0.53554285
For n_clusters = 6 The average silhouette_score is : 0.521654
For n_clusters = 7 The average silhouette_score is : 0.50656337
For n_clusters = 8 The average silhouette_score is : 0.4840957
For n_clusters = 9 The average silhouette_score is : 0.50139666
For n_clusters = 10 The average silhouette_score is : 0.49916288
For n_clusters = 11 The average silhouette_score is : 0.48444888
For n_clusters = 12 The average silhouette_score is : 0.4919362
For n_clusters = 13 The average silhouette_score is : 0.49080762
For n_clusters = 14 The average silhouette_score is : 0.4803433
For n_clusters = 15 The average silhouette_score is : 0.47484925
For n_clusters = 16 The average silhouette_score is : 0.4554981
For n_clusters = 17 The average silhouette_score is : 0.44362697
For n_clusters = 18 The average silhouette_score is : 0.4497786
For n_clusters = 19 The average silhouette_score is : 0.44582617
For n_clusters = 20 The average silhouette_score is : 0.44209328
For n_clusters = 21 The average silhouette_score is : 0.42866376
For n_clusters = 22 The average silhouette_score is : 0.4308696
For n_clusters = 23 The average silhouette_score is : 0.4273105
For n_clusters = 24 The average silhouette_score is : 0.40478086
For n_clusters = 25 The average silhouette_score is : 0.3953512
```

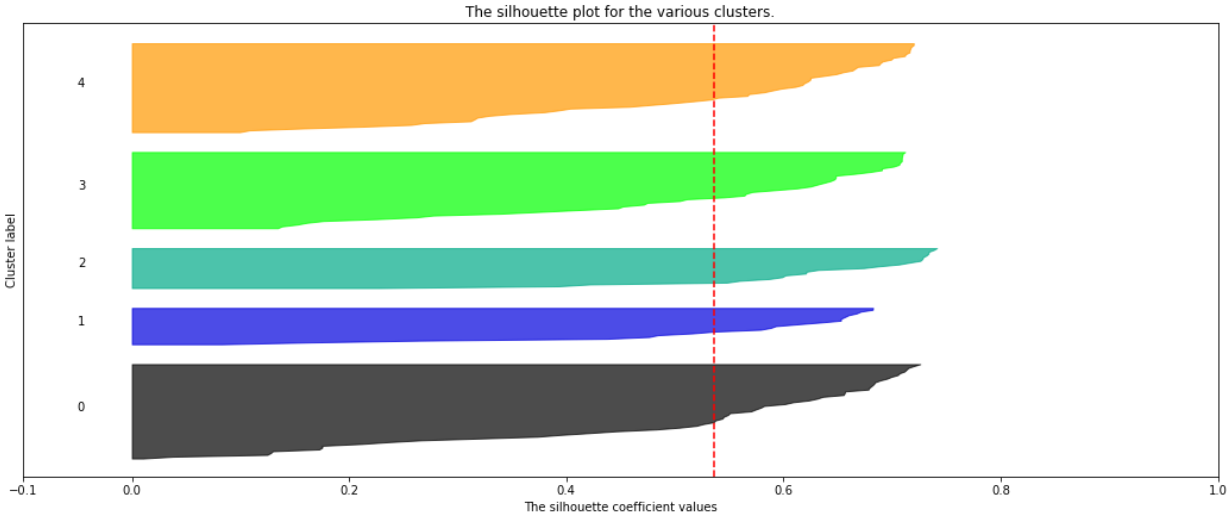




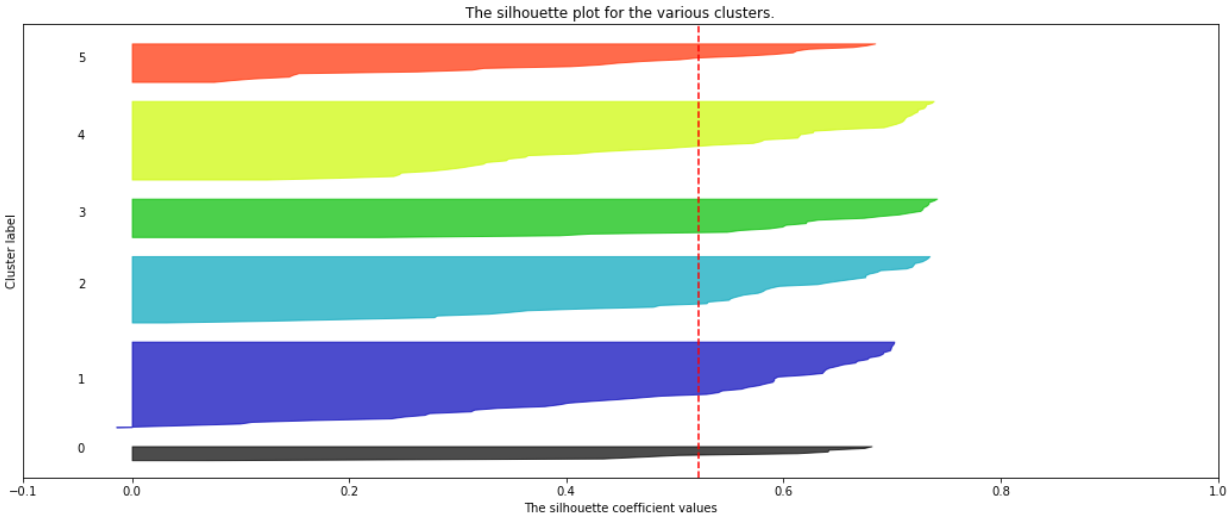
Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



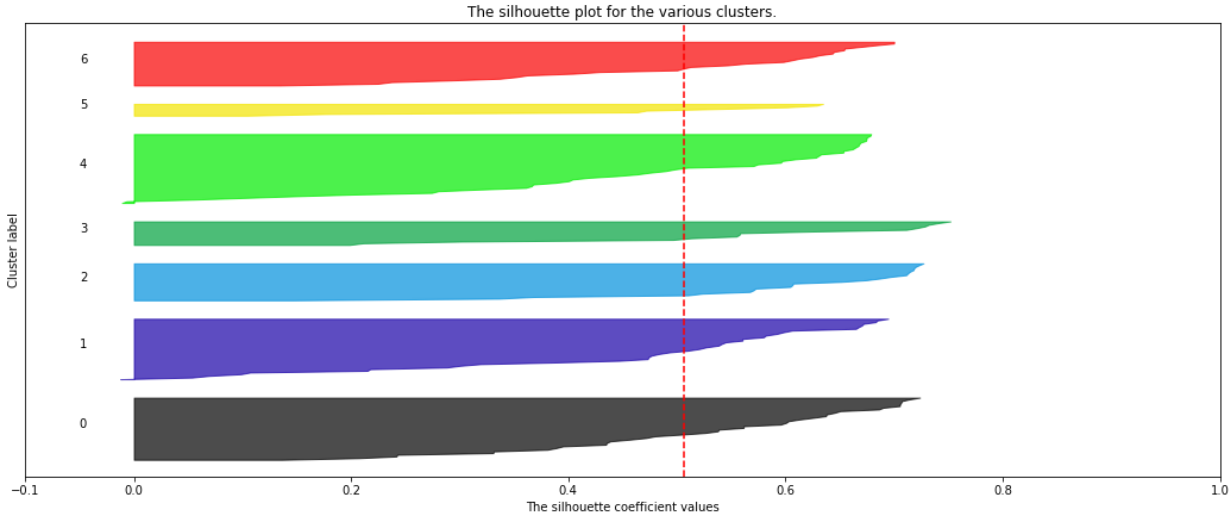
Silhouette analysis for KMeans clustering on sample data with n_clusters = 5



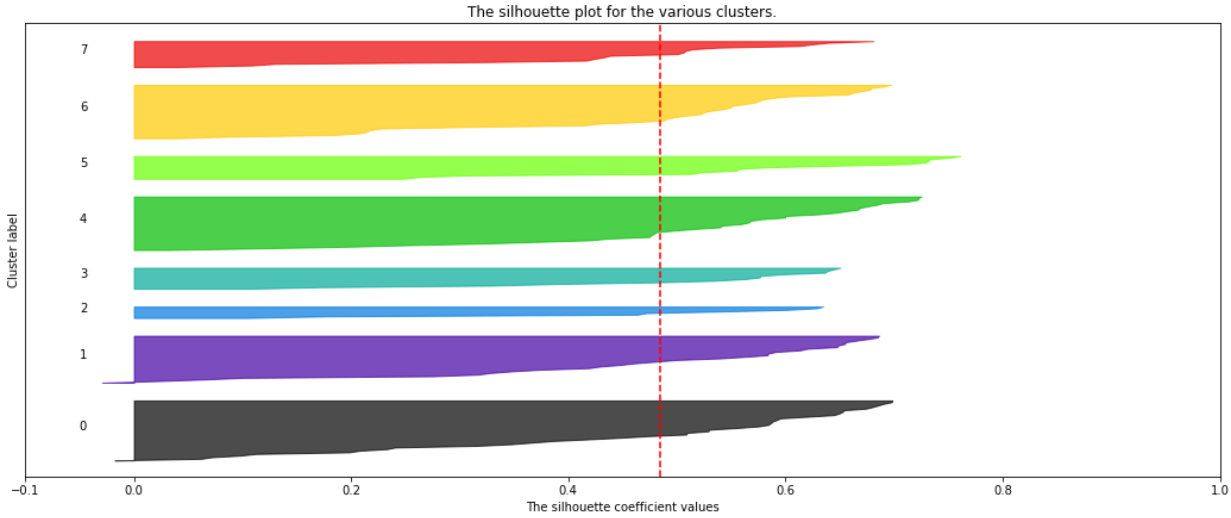
Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



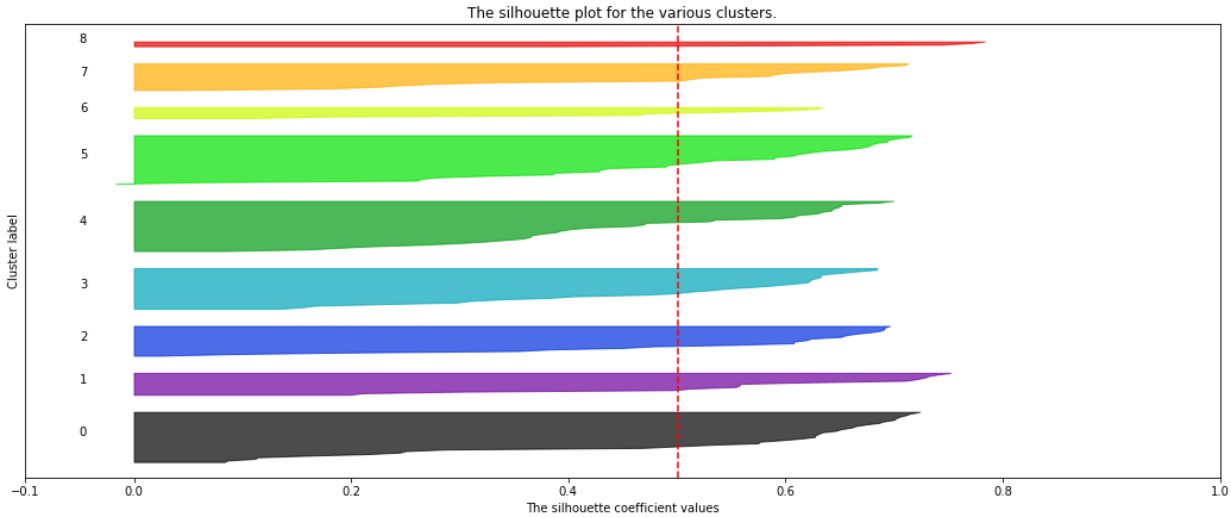
Silhouette analysis for KMeans clustering on sample data with n_clusters = 7

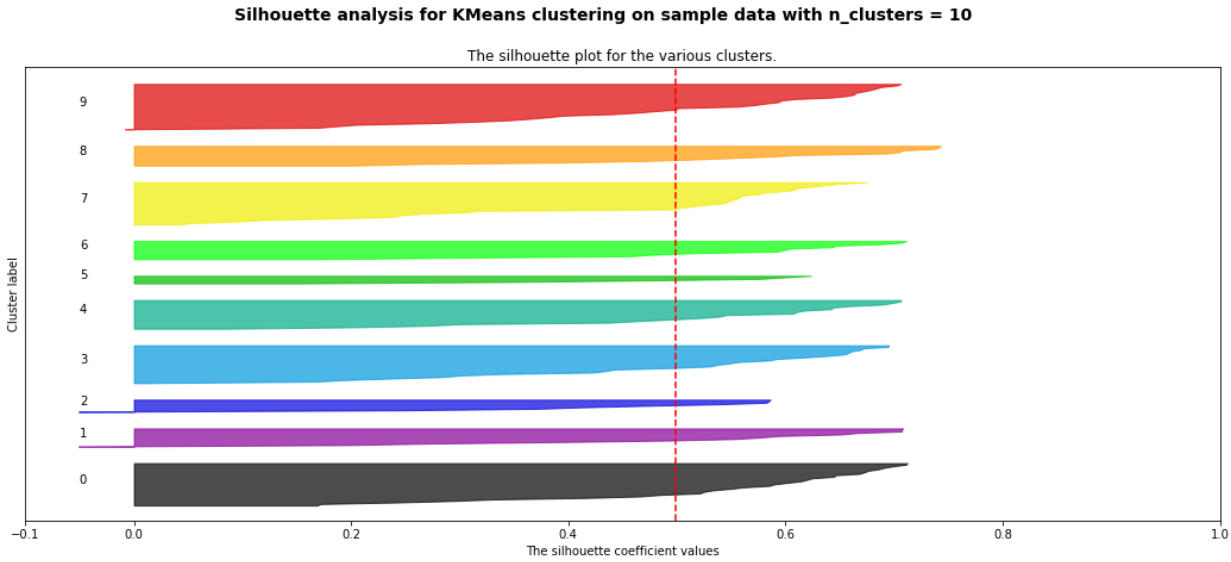


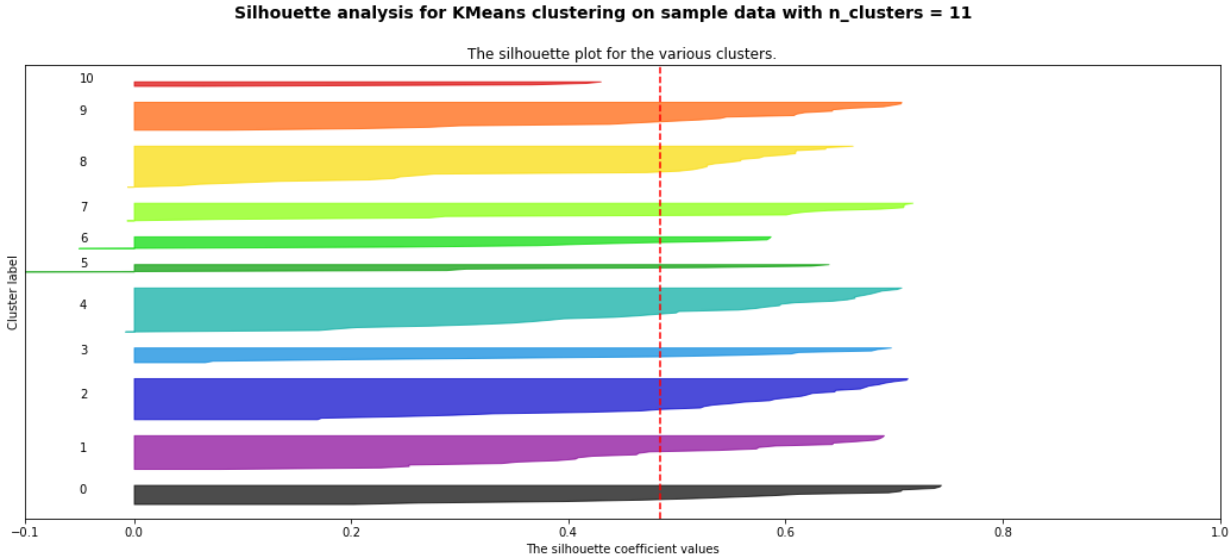
Silhouette analysis for KMeans clustering on sample data with n_clusters = 8

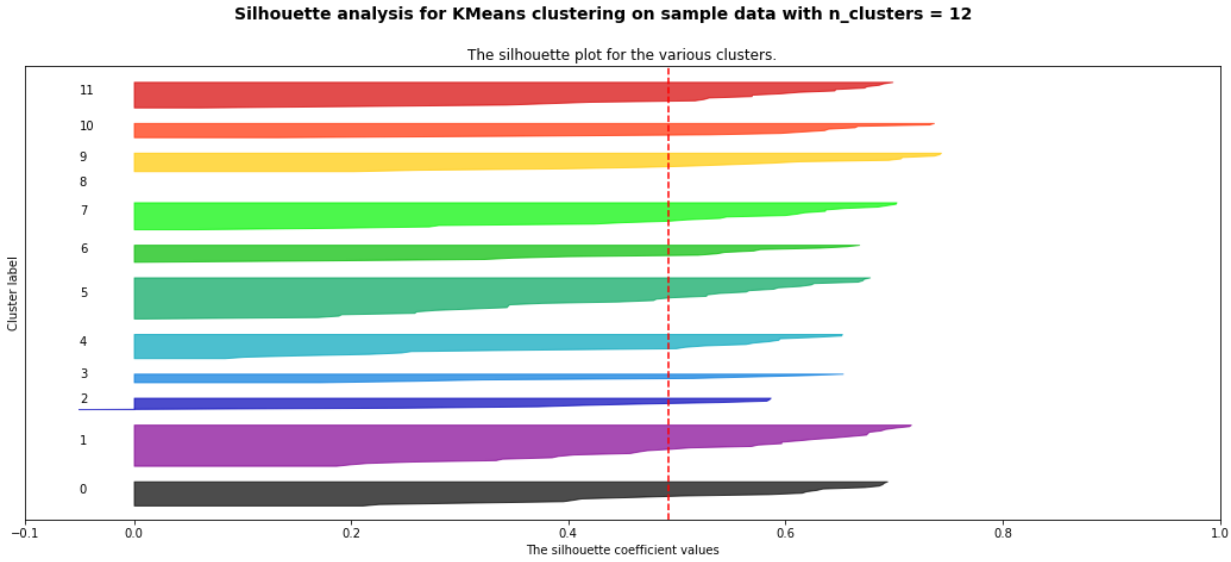


Silhouette analysis for KMeans clustering on sample data with n_clusters = 9

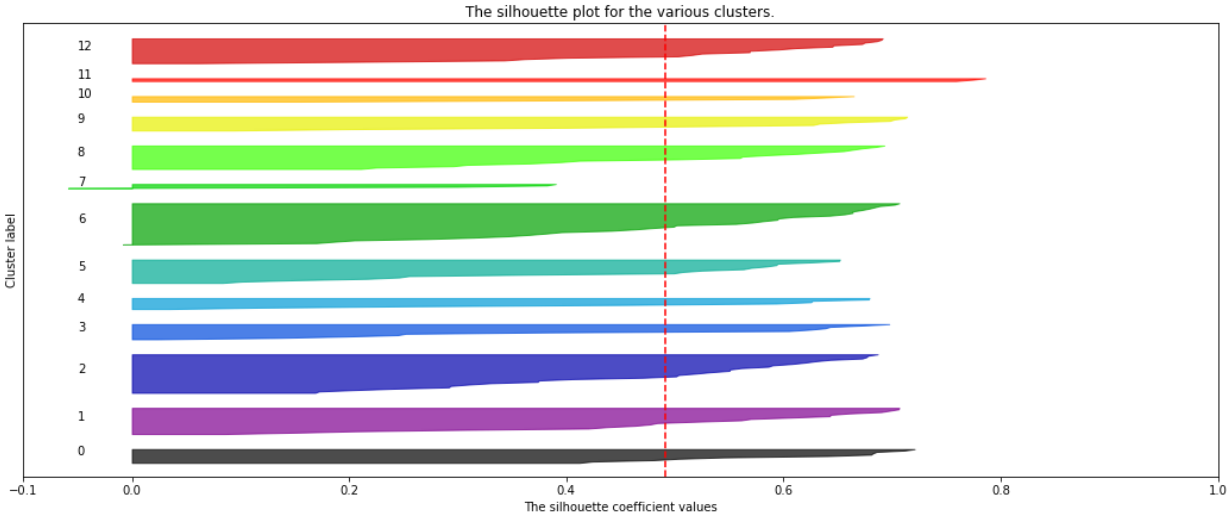



Silhouette analysis for KMeans clustering on sample data with n_clusters = 10
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

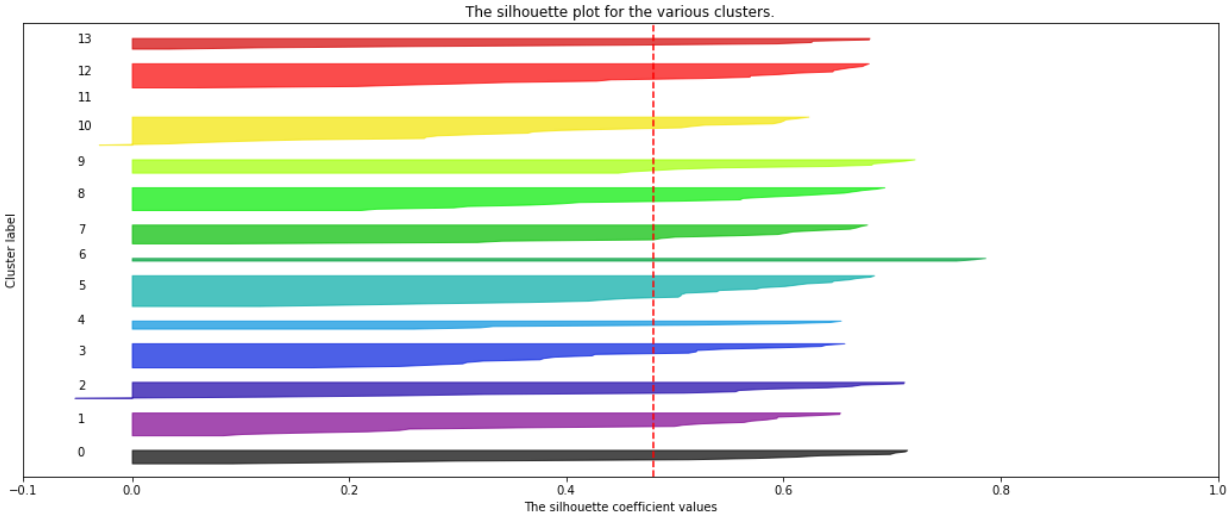

Silhouette analysis for KMeans clustering on sample data with n_clusters = 11
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


Silhouette analysis for KMeans clustering on sample data with n_clusters = 12
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

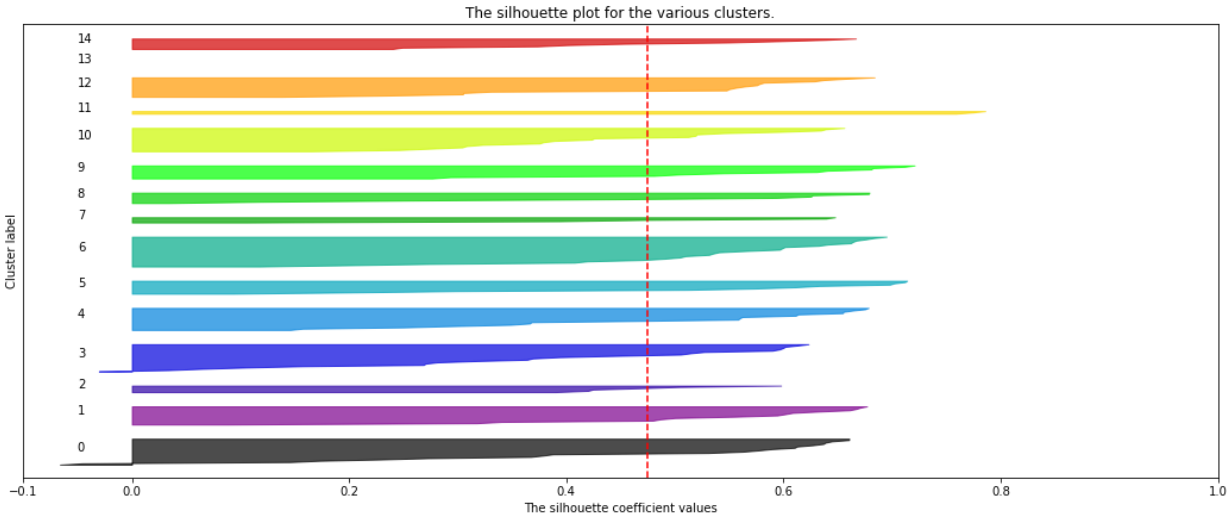
Silhouette analysis for KMeans clustering on sample data with n_clusters = 13

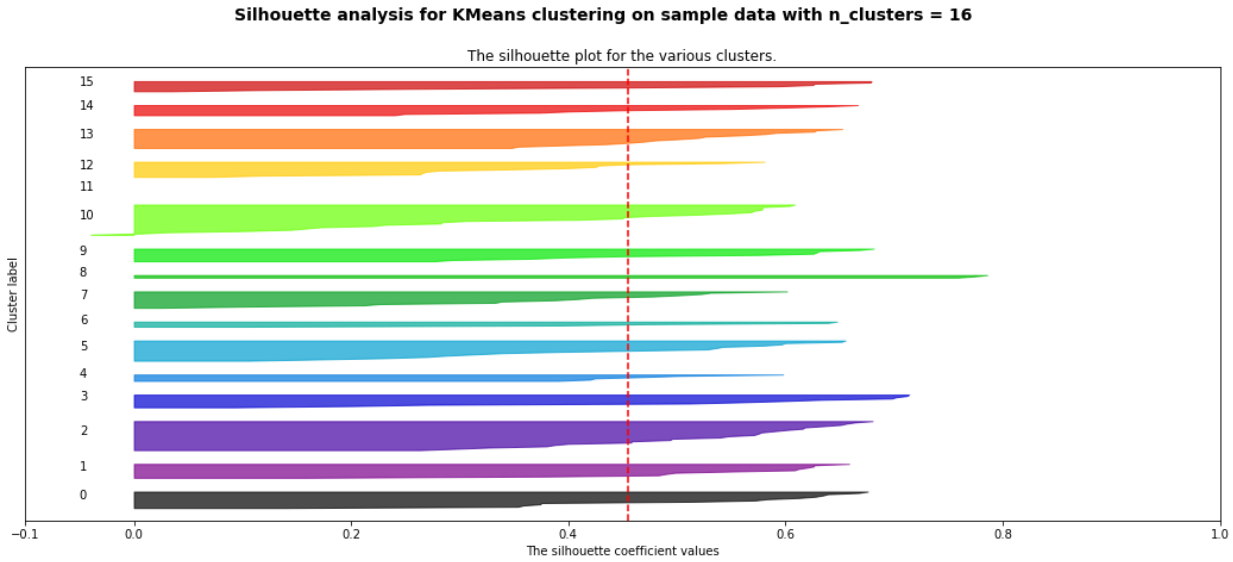


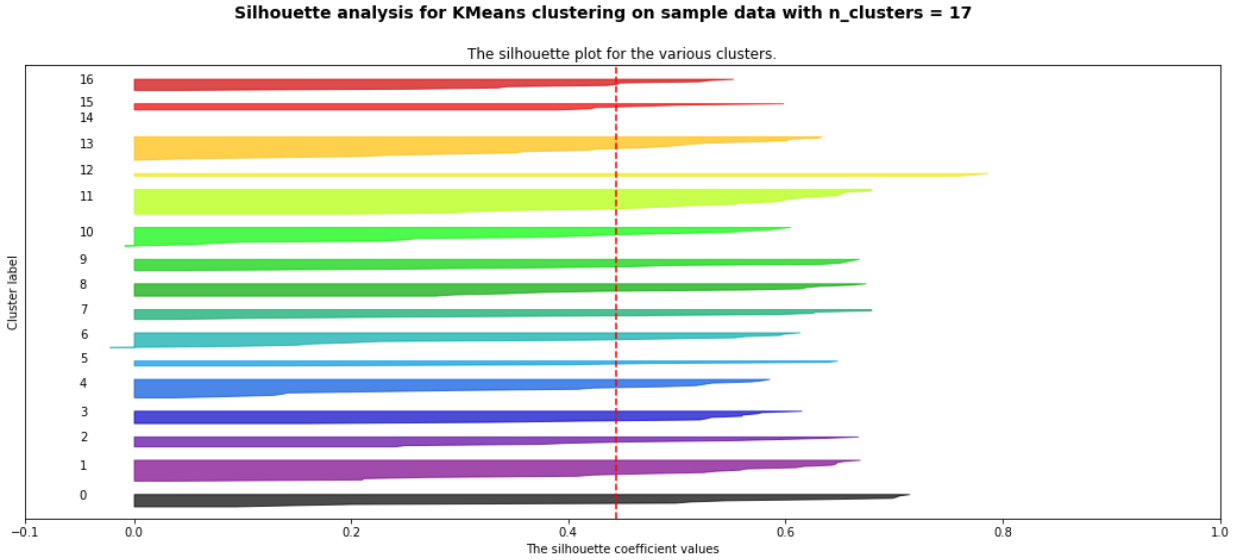
Silhouette analysis for KMeans clustering on sample data with n_clusters = 14

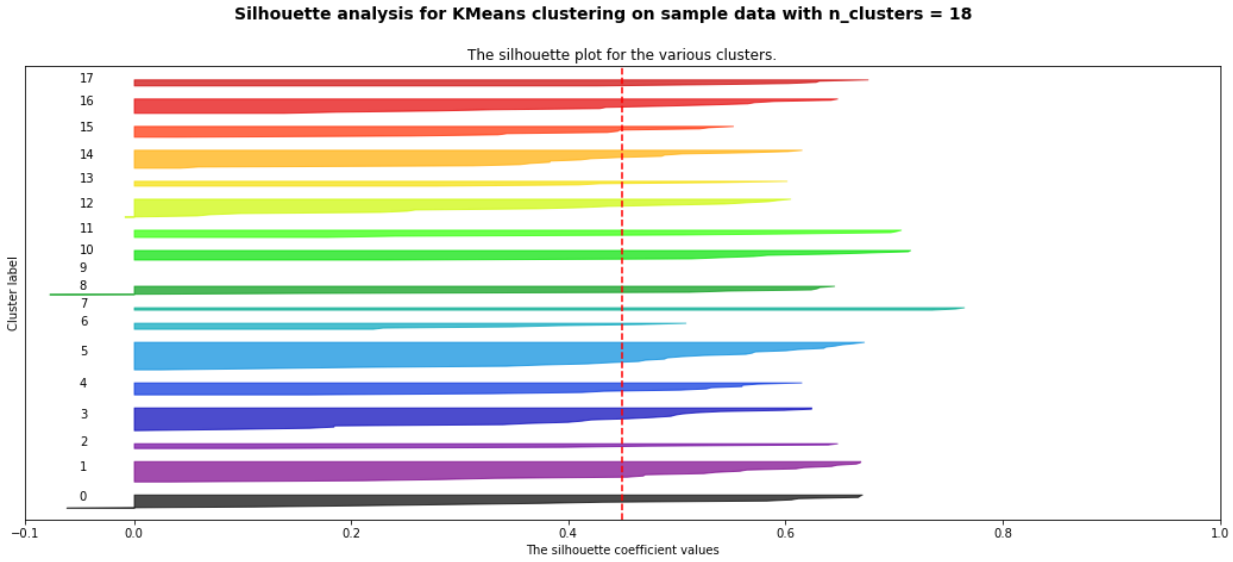


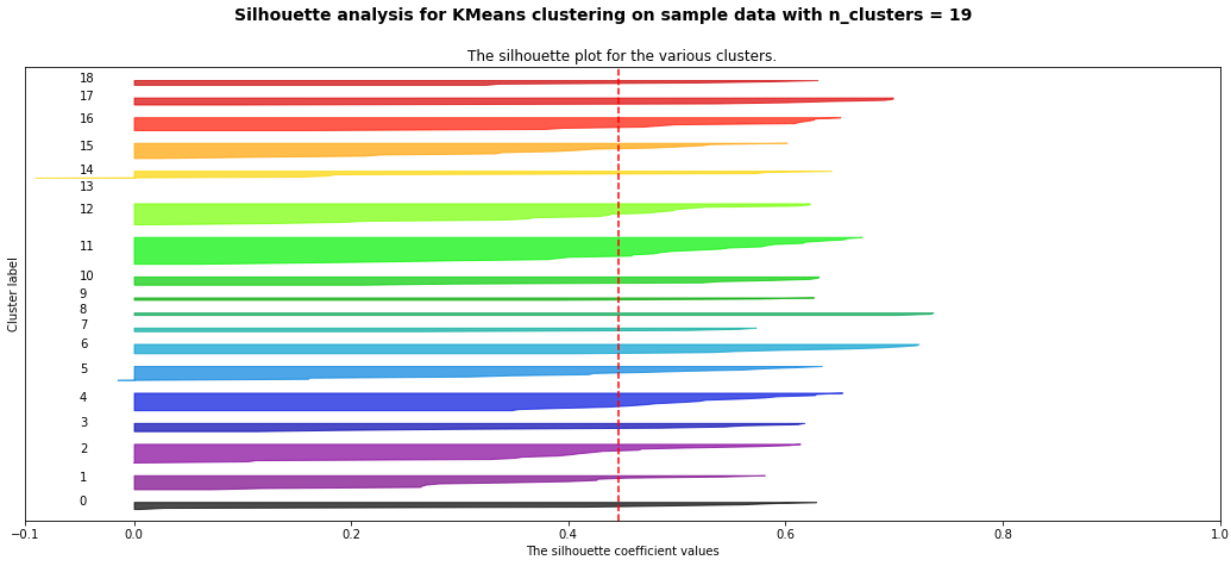
Silhouette analysis for KMeans clustering on sample data with n_clusters = 15

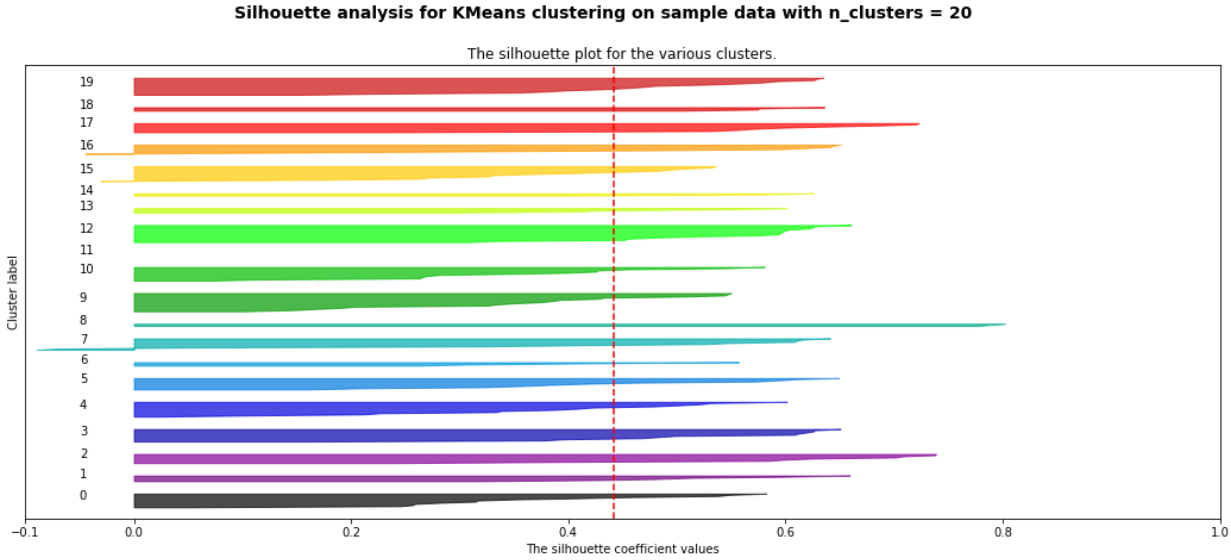


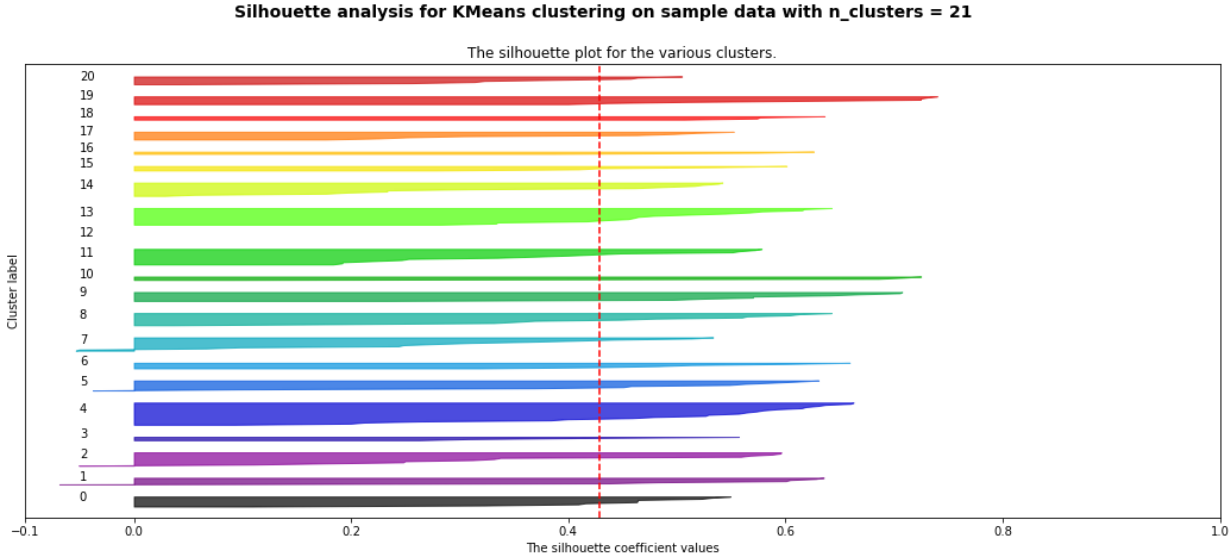

Silhouette analysis for KMeans clustering on sample data with n_clusters = 16
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

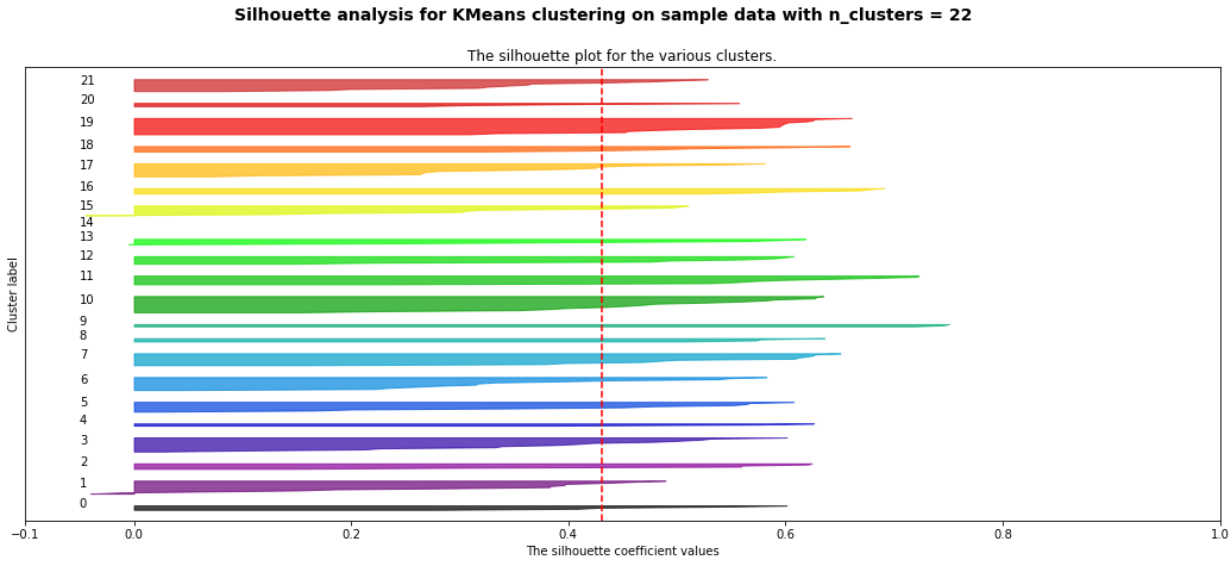

Silhouette analysis for KMeans clustering on sample data with n_clusters = 17
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

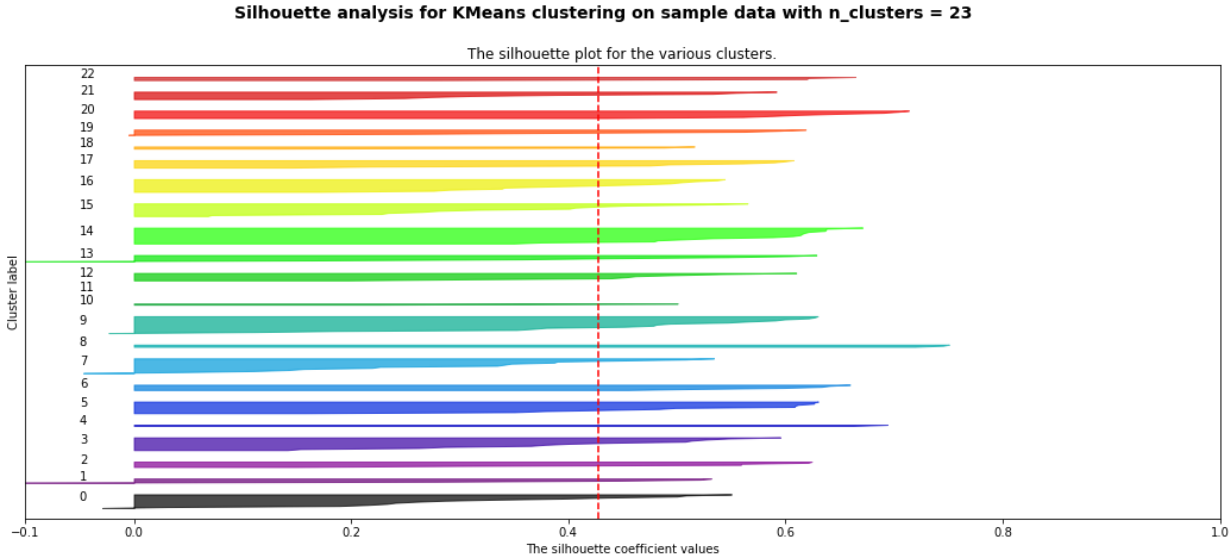

Silhouette analysis for KMeans clustering on sample data with n_clusters = 18
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

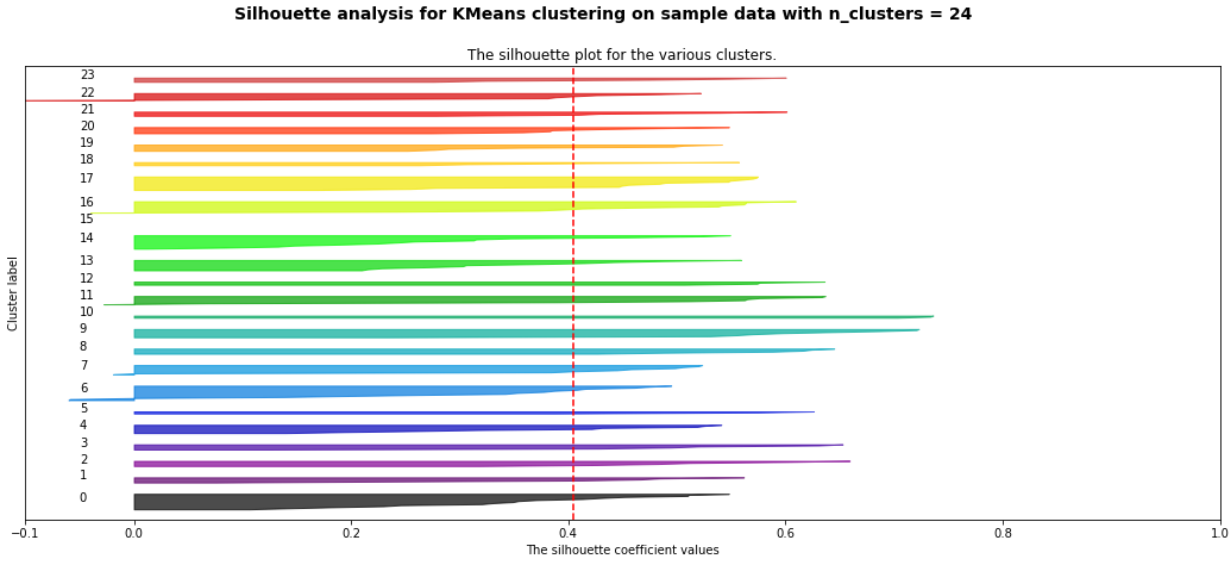

Silhouette analysis for KMeans clustering on sample data with n_clusters = 19
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

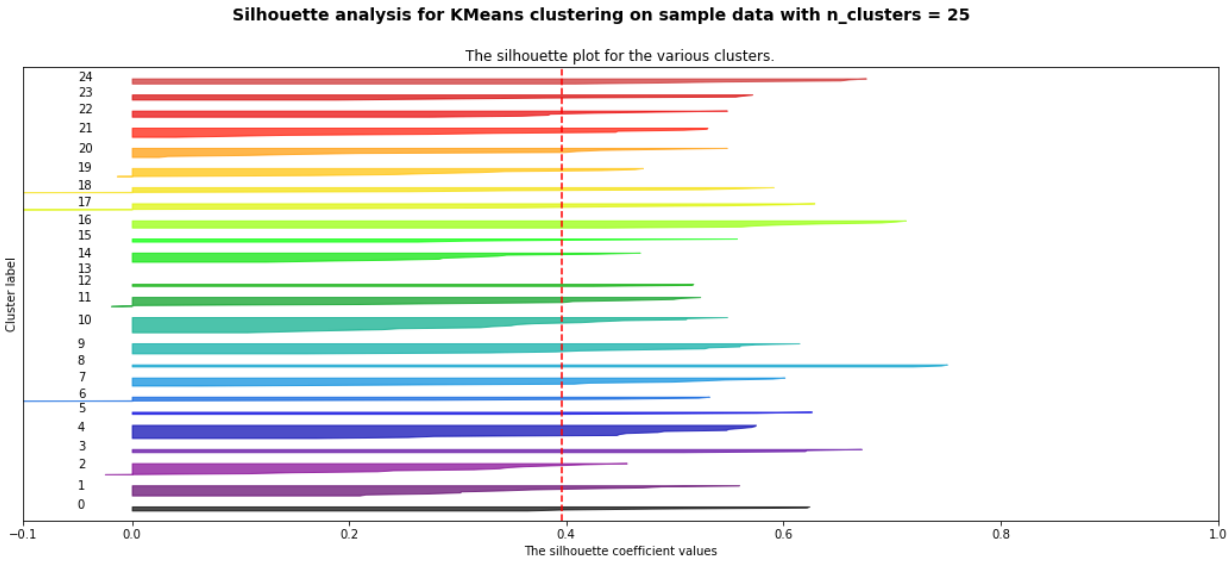

Silhouette analysis for KMeans clustering on sample data with n_clusters = 20
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 21
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 22
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

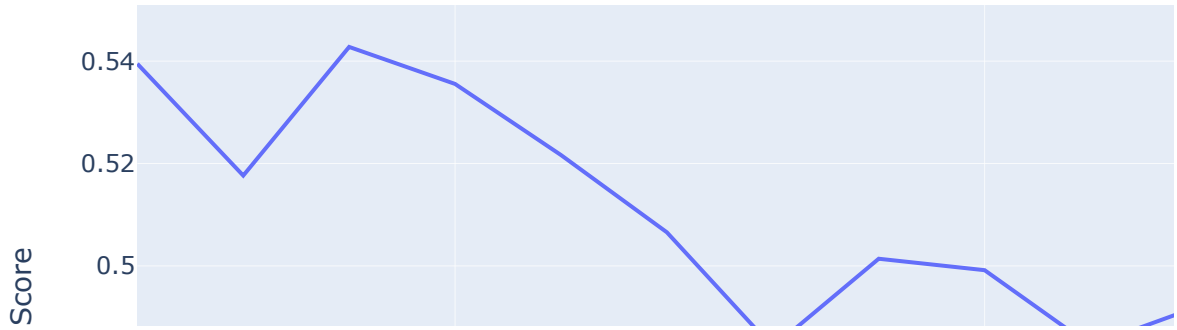

Silhouette analysis for KMeans clustering on sample data with n_clusters = 23
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 24
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values



In [147…

```
avg_silhouette_score_dic = {'Number of Clusters':range_n_clusters, 'Average Silhouette
avg_silhouette_score_df = pd.DataFrame(avg_silhouette_score_dic)


fig = px.line(avg_silhouette_score_df, x = "Number of Clusters", y = "Average Silhouet
              title = "Average Silhouette Score By Number of Clusters")

fig.update_layout(height = 600, xaxis_title = 'Number of Clusters')
```



In [148…

```
cluster_title, clusters, k_means_df = k_means(titles,
                                              tdm_matrix = doc2vec_df,
                                              k=19,
                                              processed_text = final_processed_text_me


cluster_title[9]

plot_tfidf_matrix(cluster_title, clusters, doc2vec_df)
```

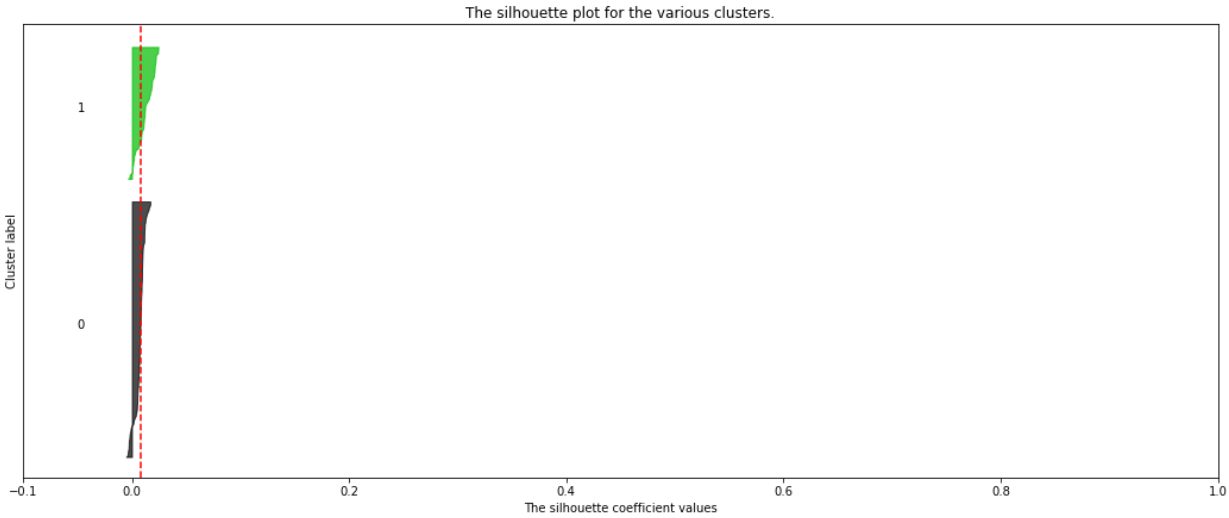
Clustering Experiment 3: K-Means Clustering with Data Wrangling and Vectorization Method 3

In [27]:

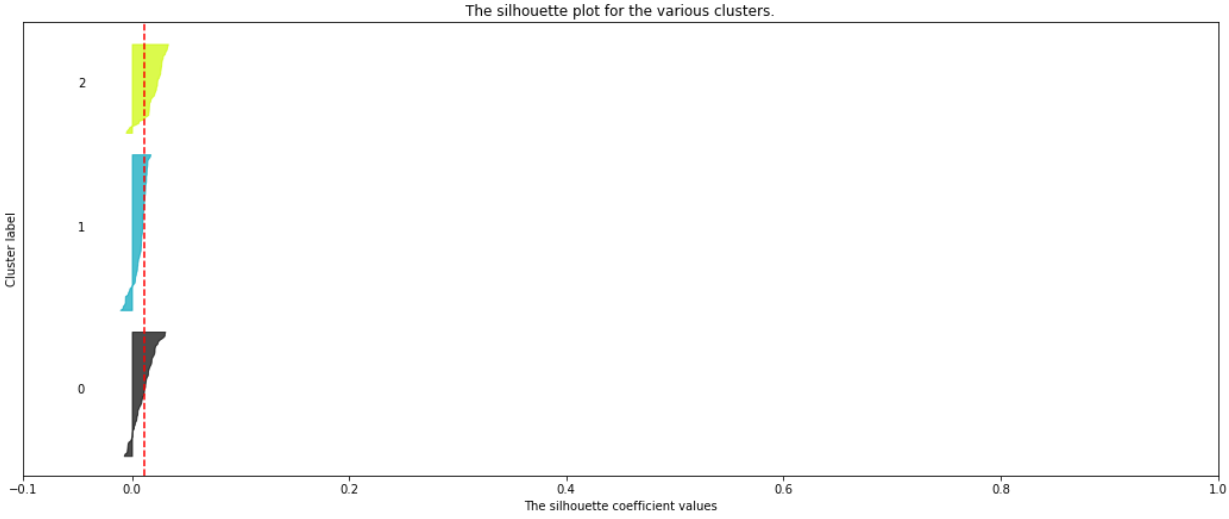
```
average_silhouette_scores = []

k_means_cluster_count_tuner(tfidf_matrix_method_four)
```

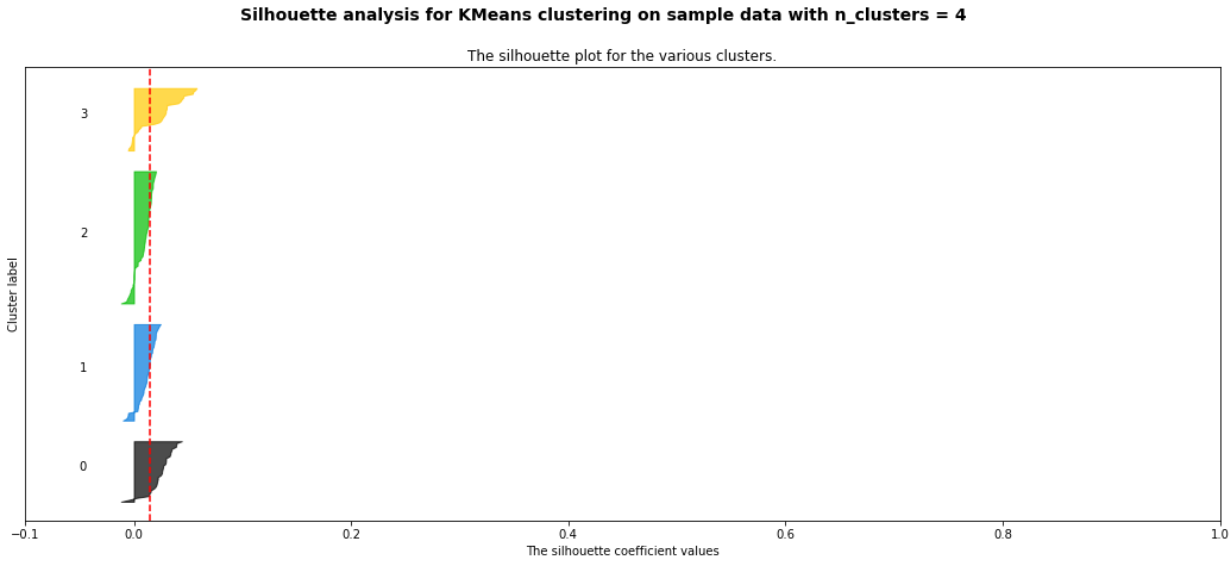
```
For n_clusters = 2 The average silhouette_score is : 0.008663709066639933
For n_clusters = 3 The average silhouette_score is : 0.01107107408302278
For n_clusters = 4 The average silhouette_score is : 0.014439666688113582
For n_clusters = 5 The average silhouette_score is : 0.01759062813683313
For n_clusters = 6 The average silhouette_score is : 0.023869546822147078
For n_clusters = 7 The average silhouette_score is : 0.03668387668401417
For n_clusters = 8 The average silhouette_score is : 0.03511409353055053
For n_clusters = 9 The average silhouette_score is : 0.04149921108580751
For n_clusters = 10 The average silhouette_score is : 0.04502605575832928
For n_clusters = 11 The average silhouette_score is : 0.04776189430411533
For n_clusters = 12 The average silhouette_score is : 0.051901391086749606
For n_clusters = 13 The average silhouette_score is : 0.06140140202551258
For n_clusters = 14 The average silhouette_score is : 0.059039888936819766
For n_clusters = 15 The average silhouette_score is : 0.06281274533434772
For n_clusters = 16 The average silhouette_score is : 0.062155201671209914
For n_clusters = 17 The average silhouette_score is : 0.06808554965353222
For n_clusters = 18 The average silhouette_score is : 0.07607498324340509
For n_clusters = 19 The average silhouette_score is : 0.07467544245467485
For n_clusters = 20 The average silhouette_score is : 0.08150436292389848
For n_clusters = 21 The average silhouette_score is : 0.07667748467116818
For n_clusters = 22 The average silhouette_score is : 0.07498712487222027
For n_clusters = 23 The average silhouette_score is : 0.07519650877367222
For n_clusters = 24 The average silhouette_score is : 0.07267072834782883
For n_clusters = 25 The average silhouette_score is : 0.06491531734384066
```

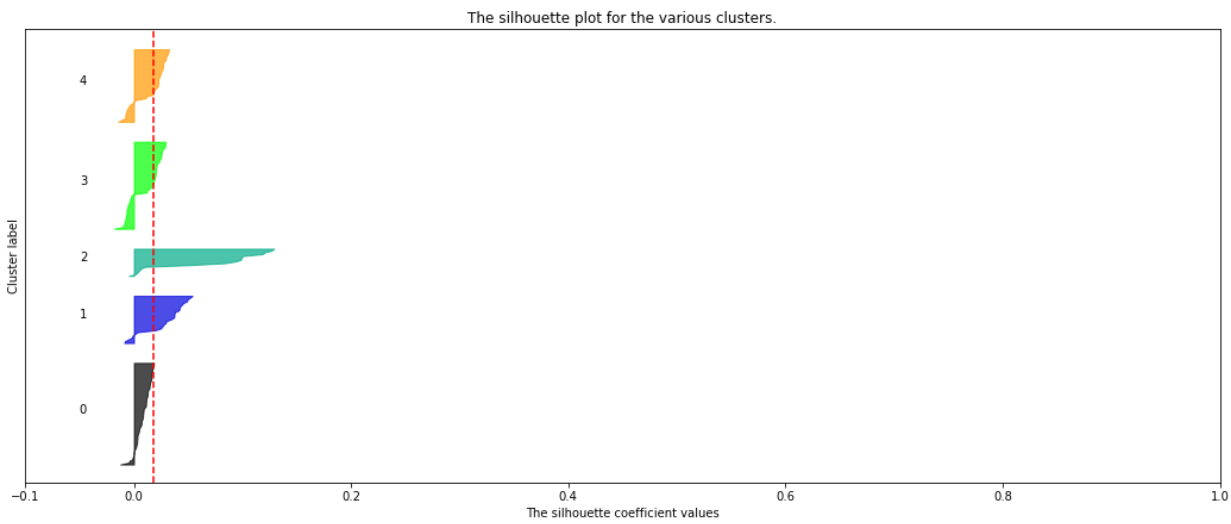
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2

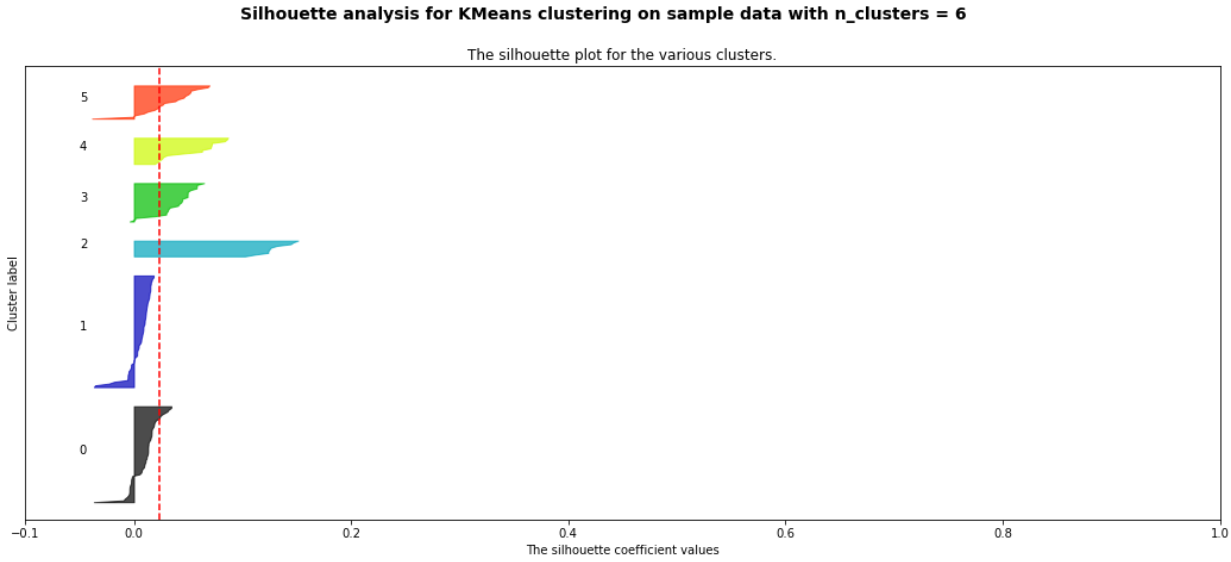


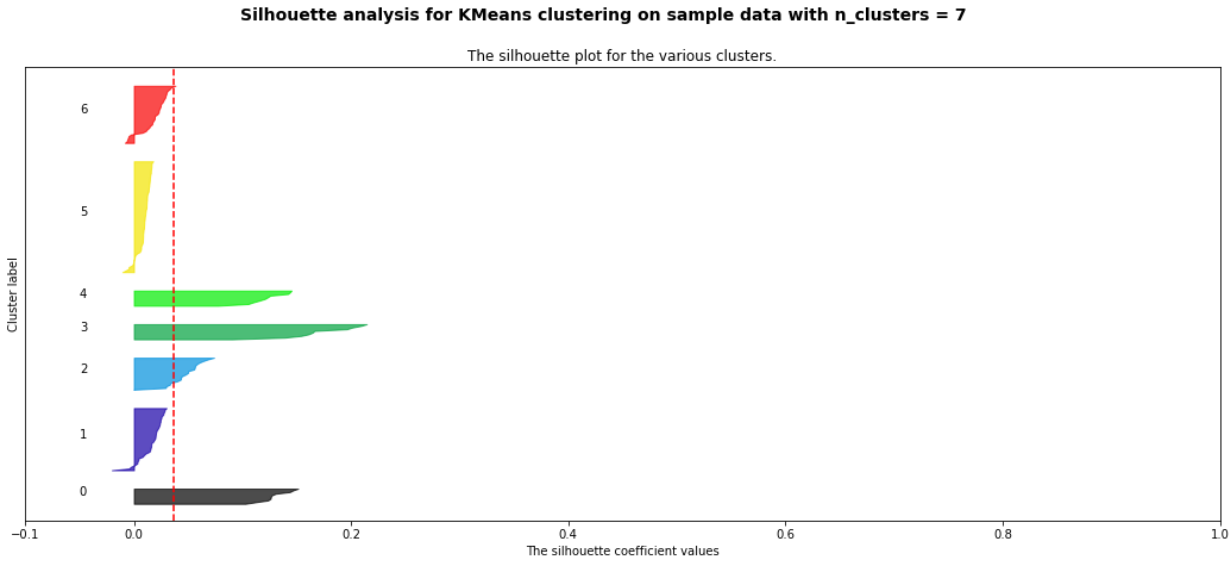
Silhouette analysis for KMeans clustering on sample data with n_clusters = 3

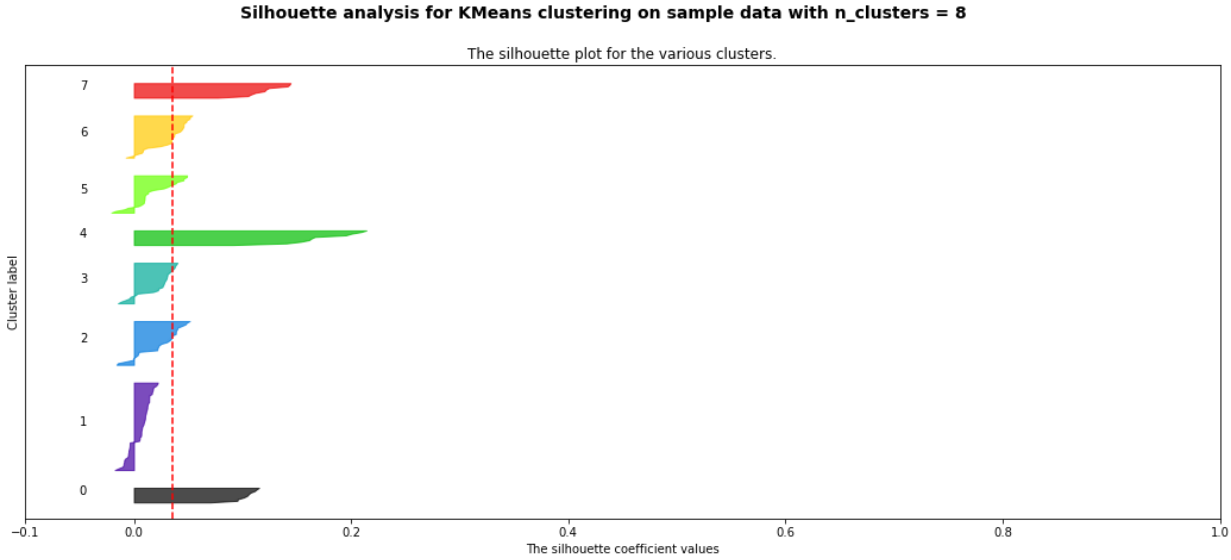


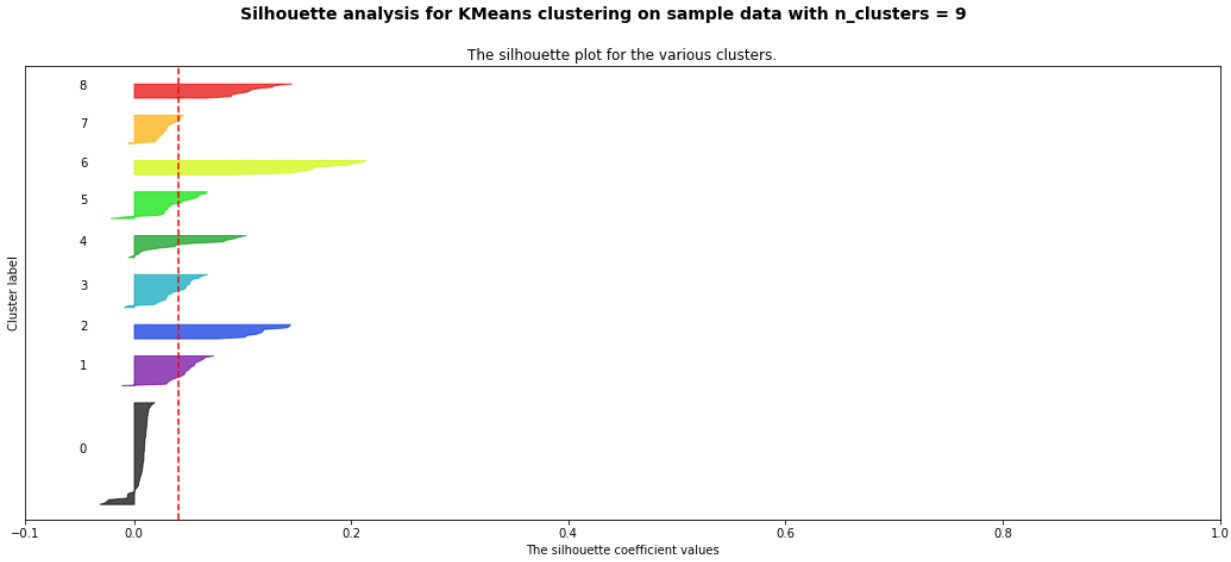

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values

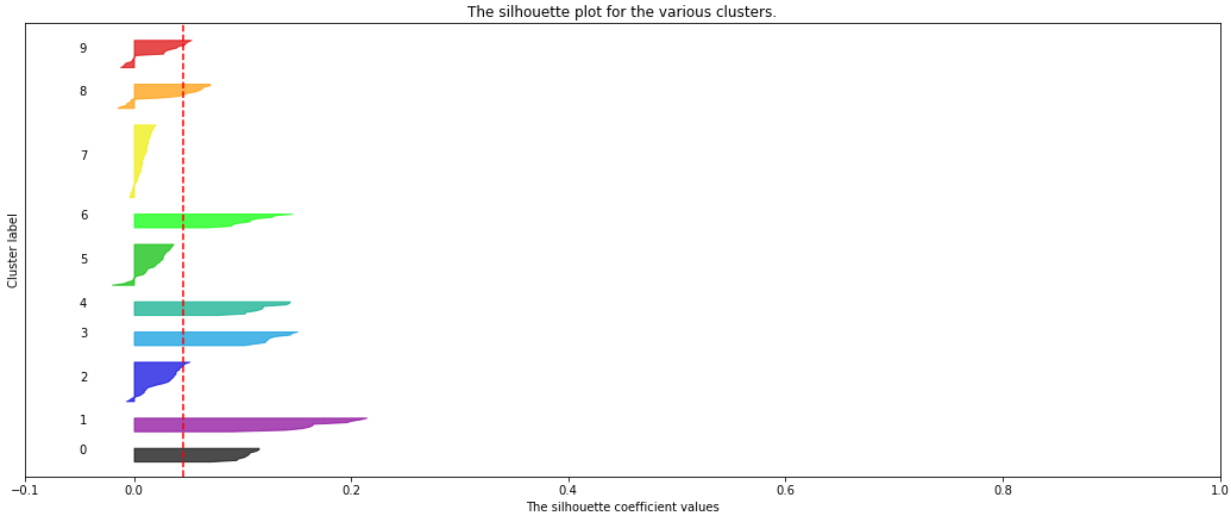

Silhouette analysis for KMeans clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
Cluster label
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 7
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

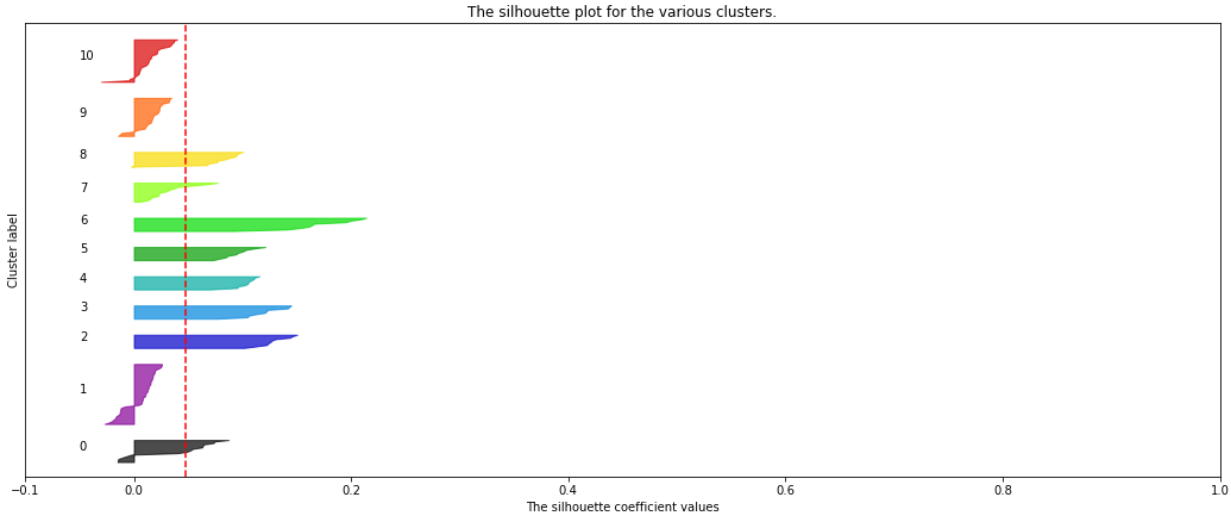

Silhouette analysis for KMeans clustering on sample data with n_clusters = 8
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


Silhouette analysis for KMeans clustering on sample data with n_clusters = 9
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

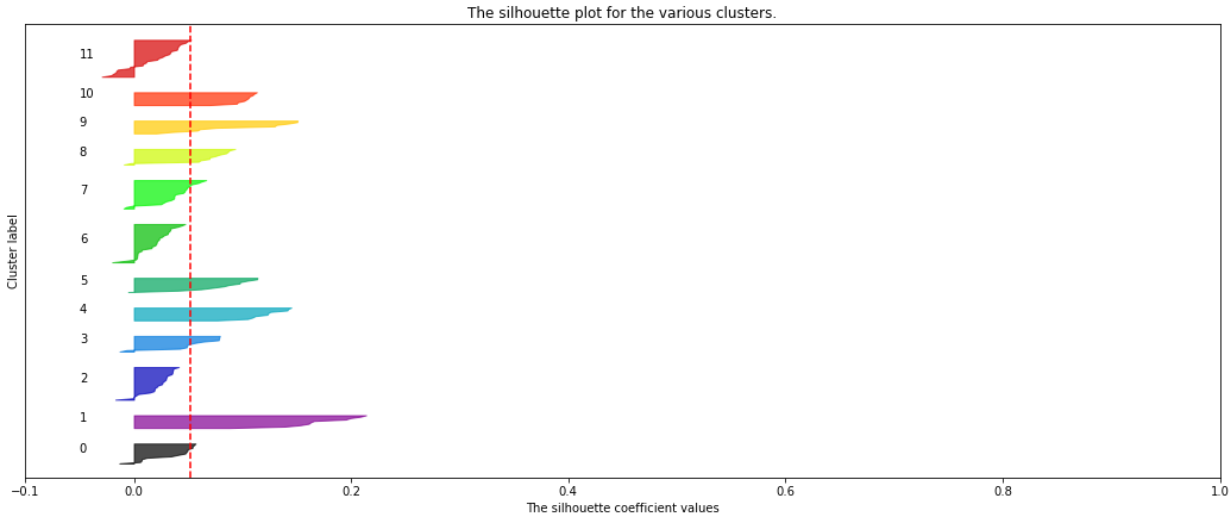
Silhouette analysis for KMeans clustering on sample data with n_clusters = 10

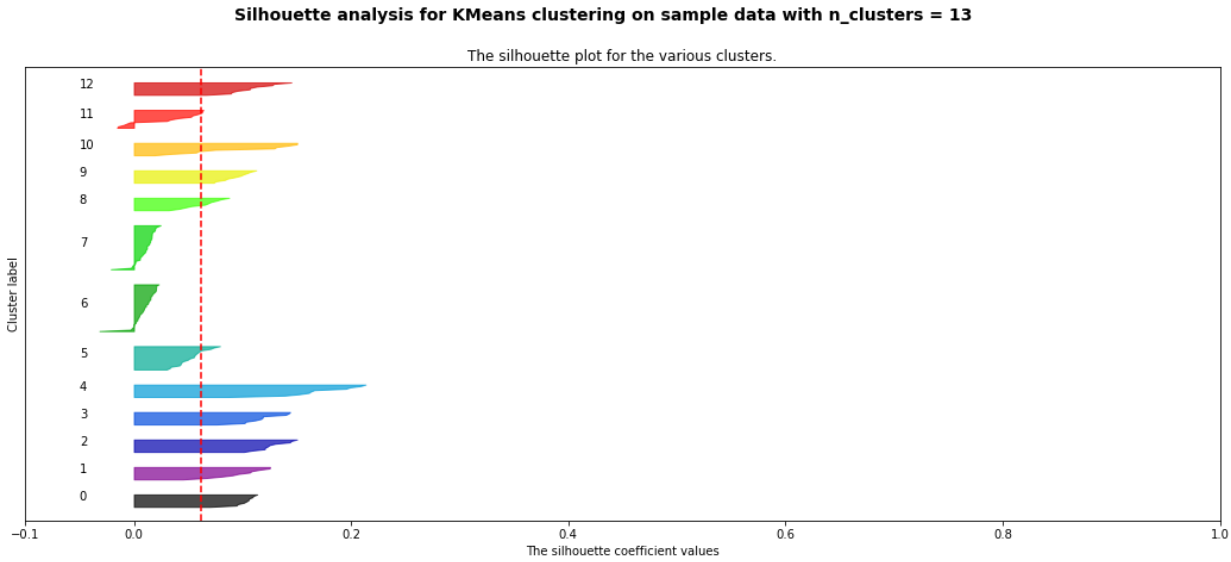


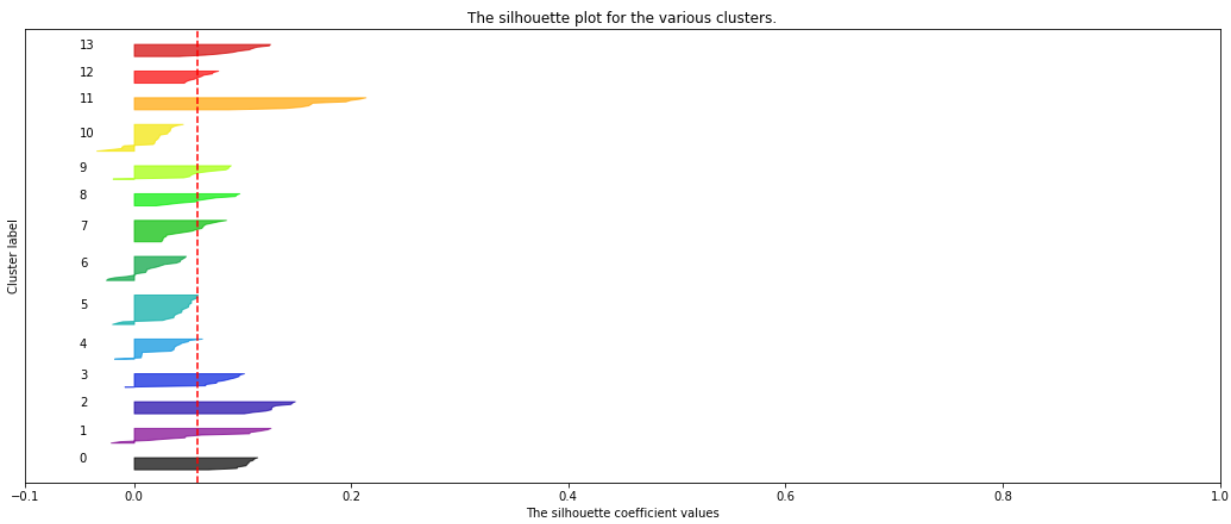
Silhouette analysis for KMeans clustering on sample data with n_clusters = 11

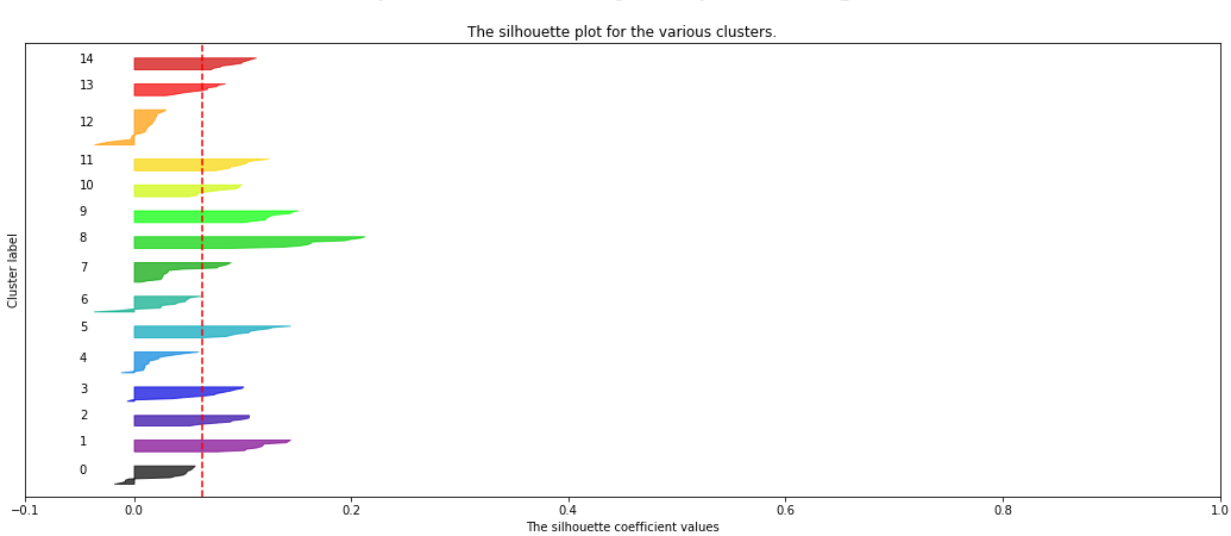


Silhouette analysis for KMeans clustering on sample data with n_clusters = 12

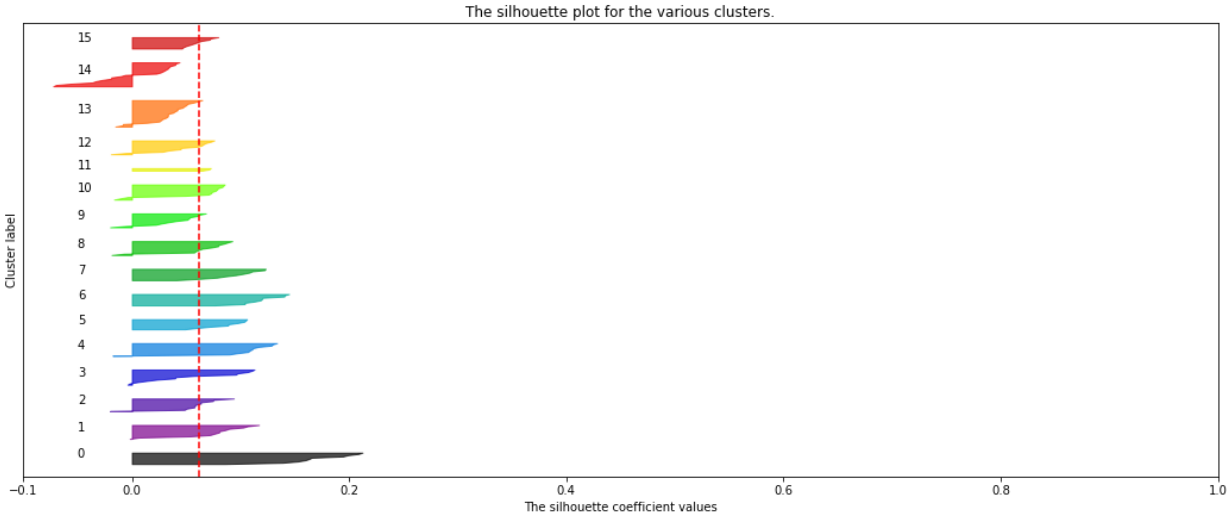



Silhouette analysis for KMeans clustering on sample data with n_clusters = 13
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

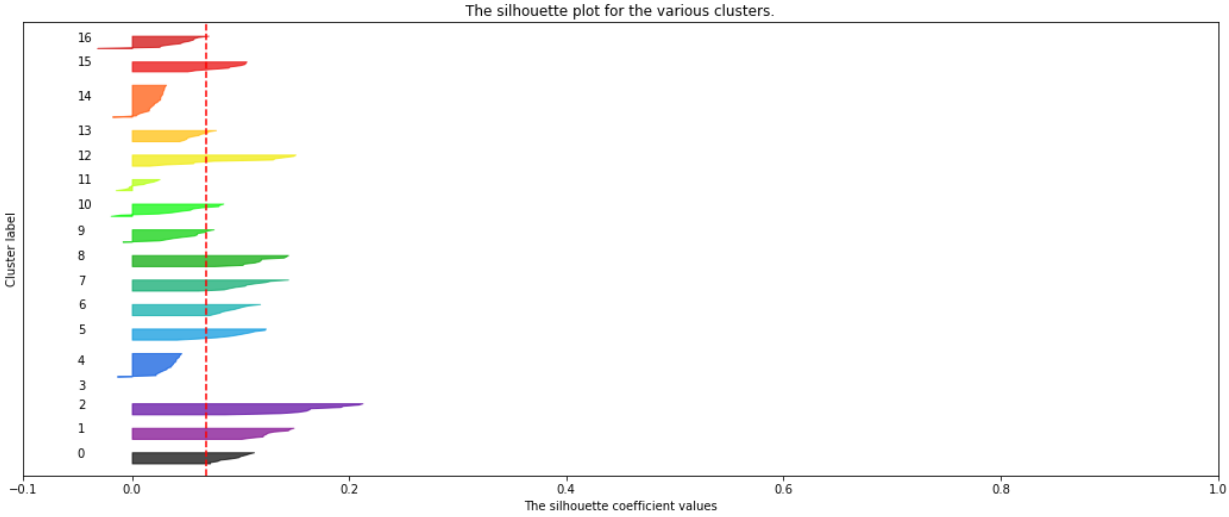

Silhouette analysis for KMeans clustering on sample data with n_clusters = 14
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
13
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


Silhouette analysis for KMeans clustering on sample data with n_clusters = 15
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

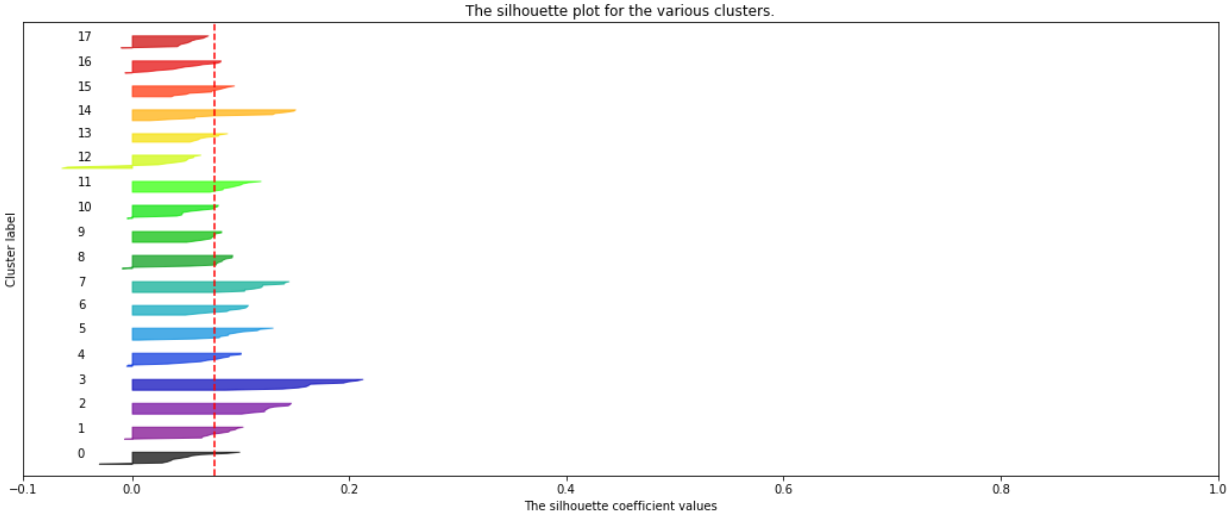
Silhouette analysis for KMeans clustering on sample data with n_clusters = 16

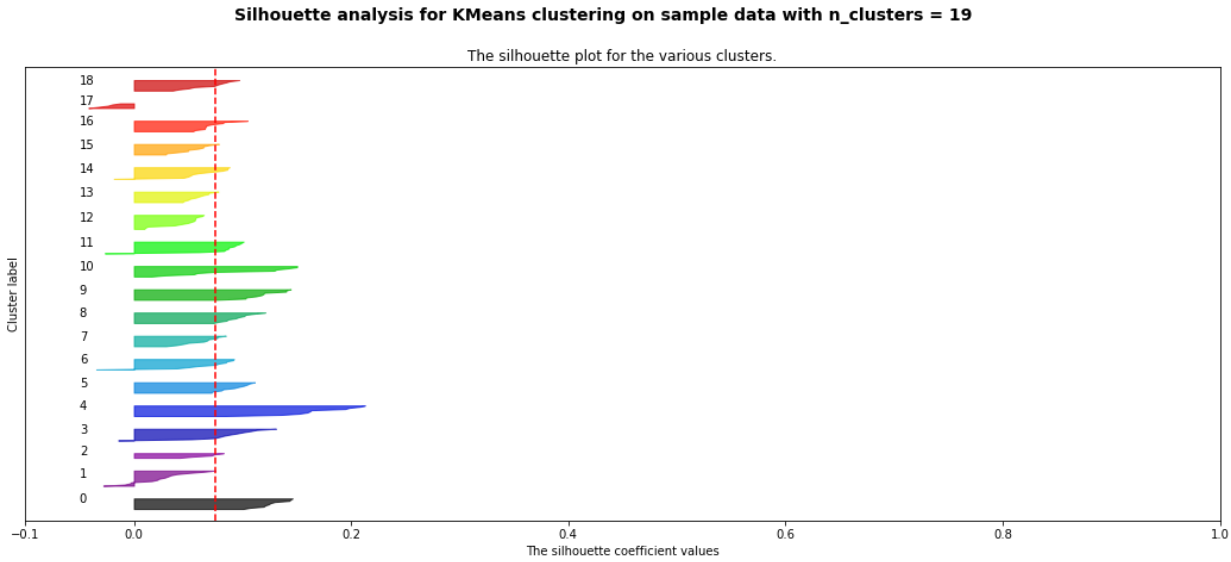


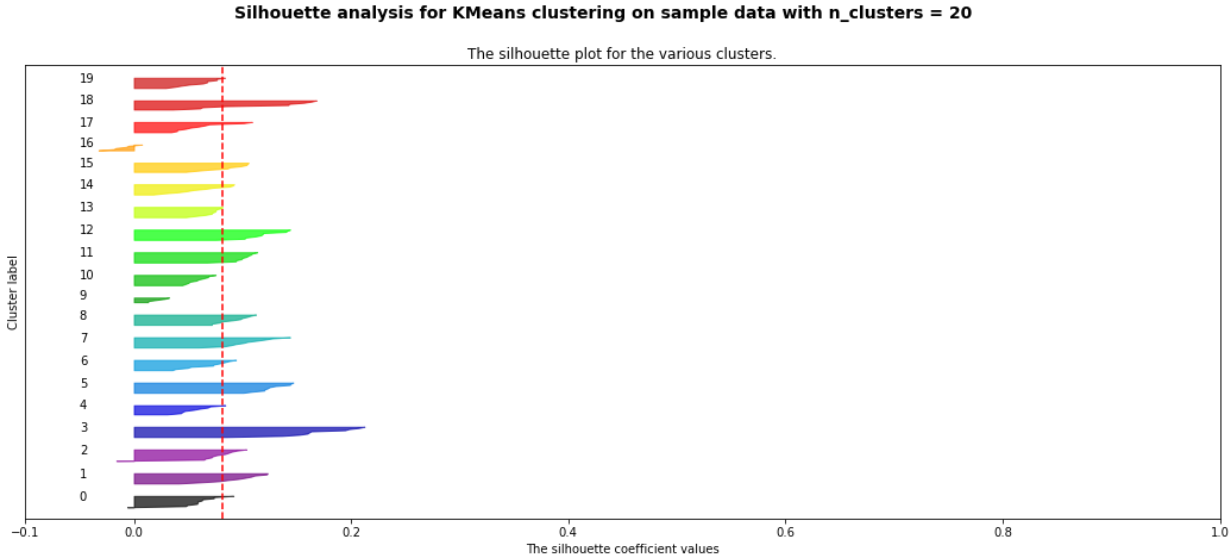
Silhouette analysis for KMeans clustering on sample data with n_clusters = 17

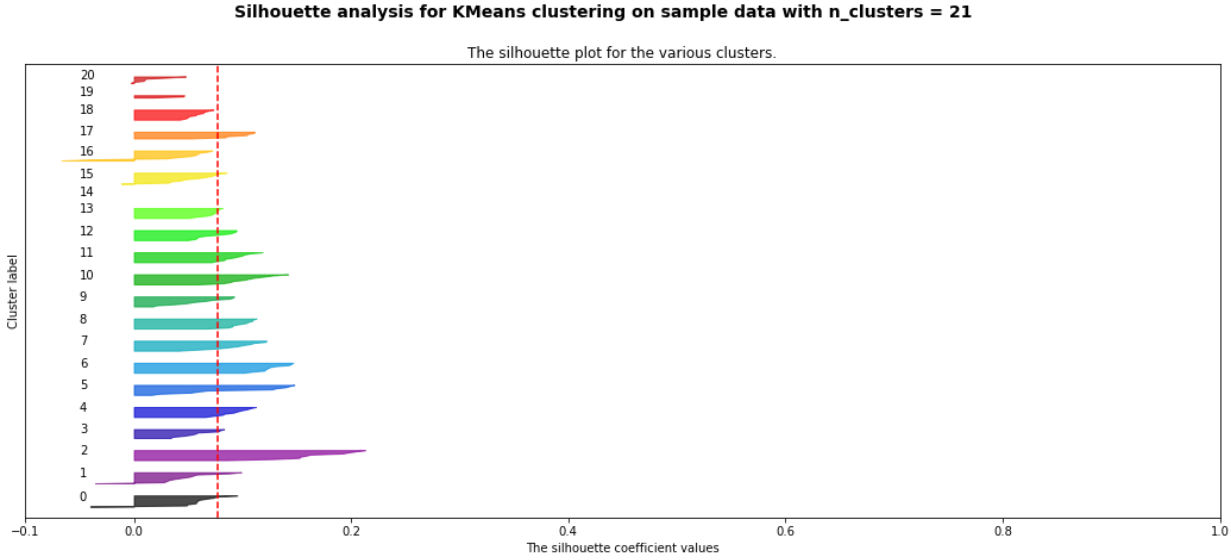


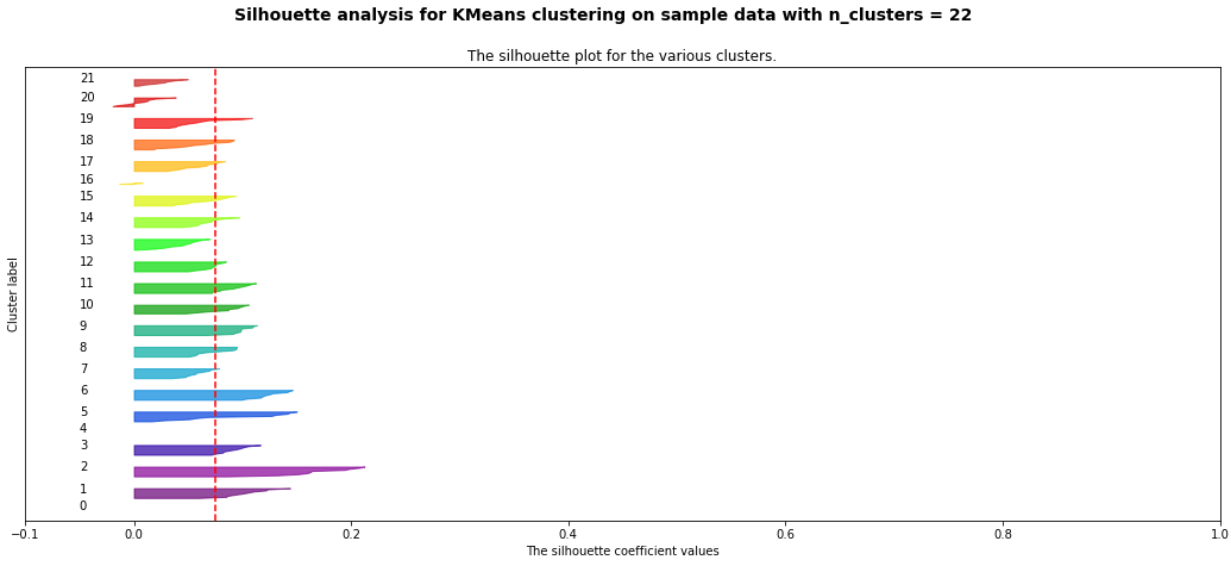
Silhouette analysis for KMeans clustering on sample data with n_clusters = 18

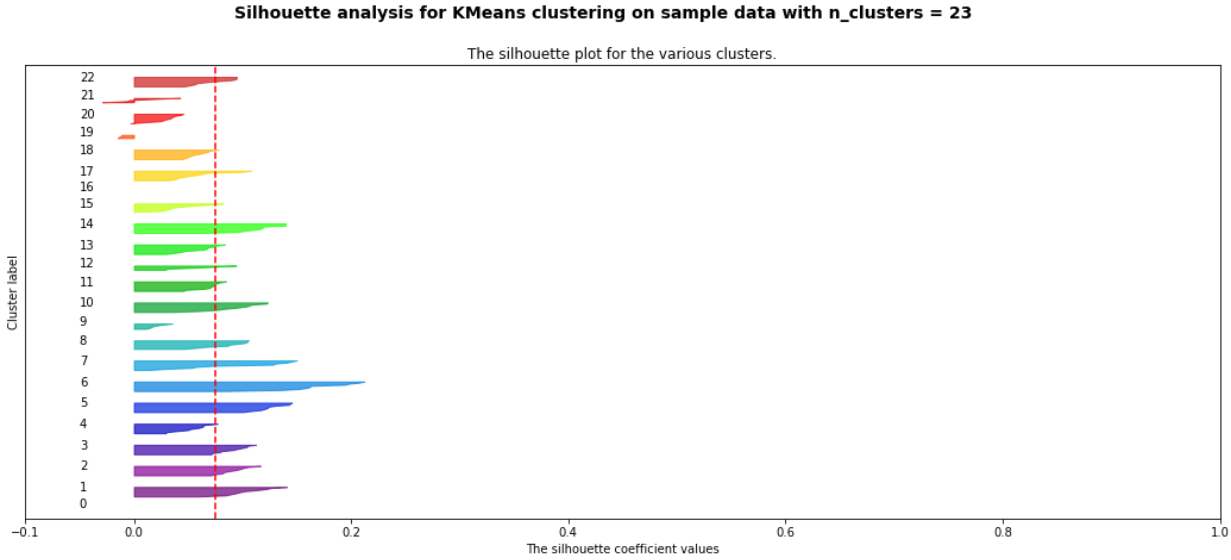


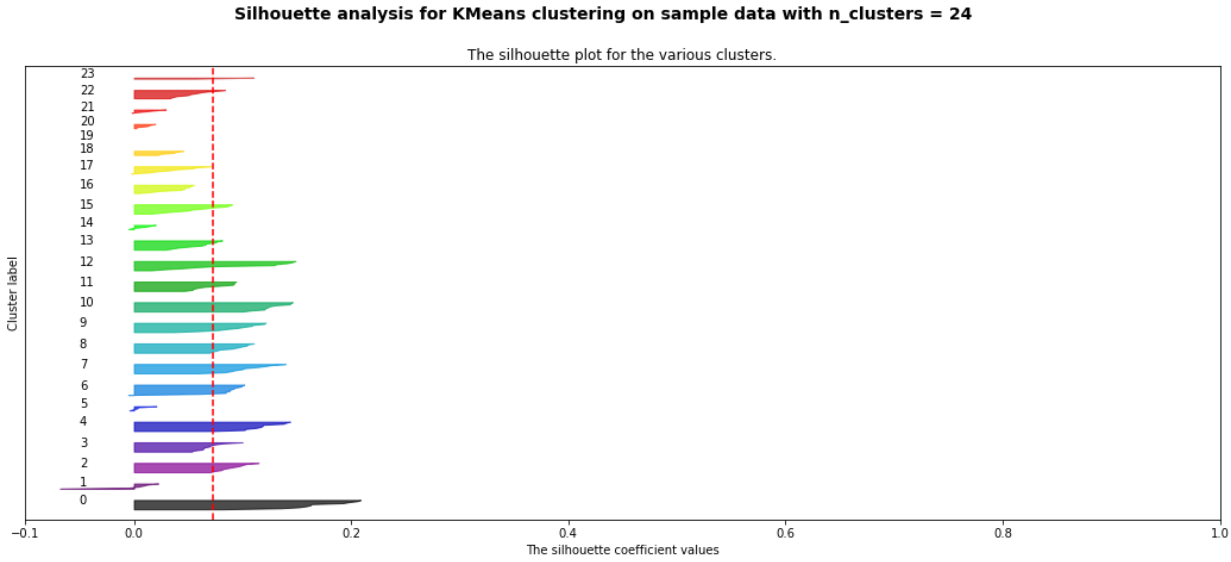

Silhouette analysis for KMeans clustering on sample data with n_clusters = 19
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

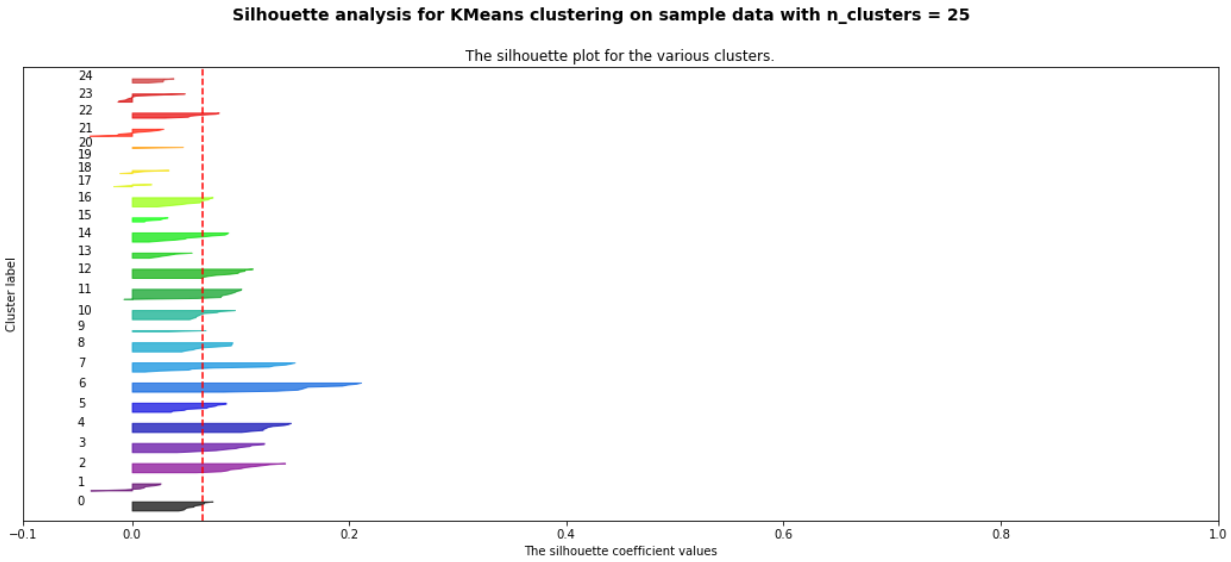

Silhouette analysis for KMeans clustering on sample data with n_clusters = 20
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0


Silhouette analysis for KMeans clustering on sample data with n_clusters = 21
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0

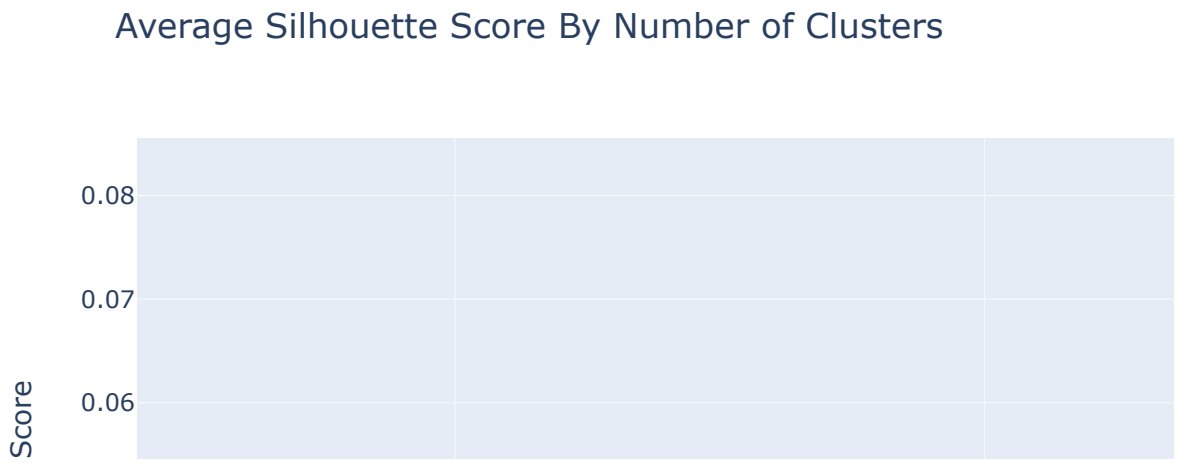

Silhouette analysis for KMeans clustering on sample data with n_clusters = 22
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values


Silhouette analysis for KMeans clustering on sample data with n_clusters = 23
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values

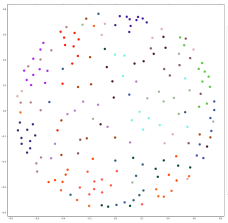

Silhouette analysis for KMeans clustering on sample data with n_clusters = 24
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values



In [28]:
```
avg_silhouette_score_dic = {'Number of Clusters':range_n_clusters, 'Average Silhouette
avg_silhouette_score_df = pd.DataFrame(avg_silhouette_score_dic)


fig = px.line(avg_silhouette_score_df, x = "Number of Clusters", y = "Average Silhouet
              title = "Average Silhouette Score By Number of Clusters")

fig.update_layout(height = 600, xaxis_title = 'Number of Clusters')
```



In [29]:
```
cluster_title, clusters, k_means_df = k_means(titles,
                                              tdm_matrix = tfidf_matrix_method_four,
                                              k=19,
                                              processed_text = final_processed_text_me


cluster_title[9]

plot_tfidf_matrix(cluster_title, clusters, tfidf_matrix_method_four)
```



2) Sentiment Analysis

Let's examine the effectiveness of various classification algorithms for conducting sentiment analysis on our corpus. We can leverage Random Forest Classification, Naive Bayes Classification, and Support Vector Machine Classification to predict which reviews are positive or negative.

2.1) Random Forest Classifier Experiments

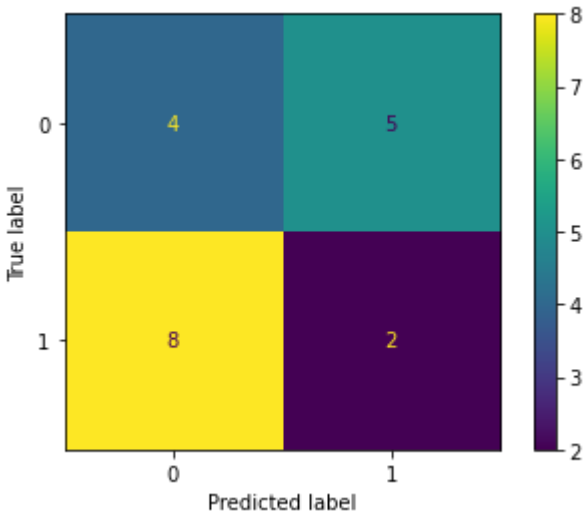
Let's apply random forest classification to our corpus to see how effectively we can predict which reviews are positive or negative.

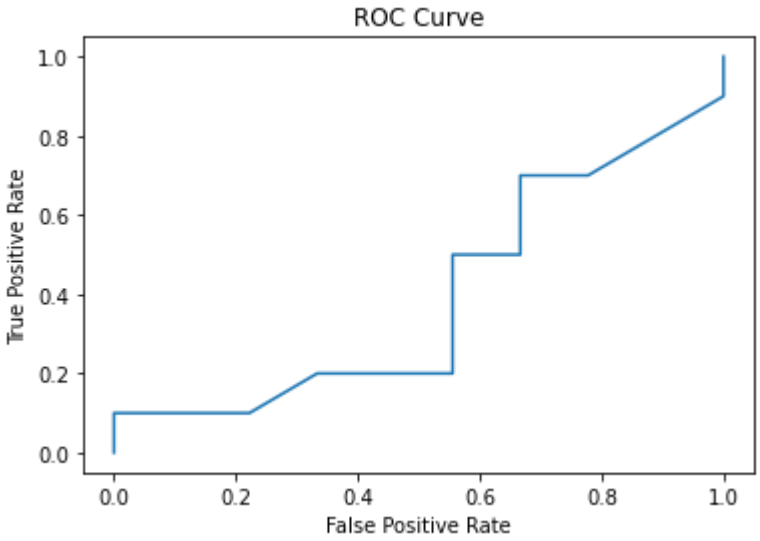
Sentiment Analysis Experiment 1: Random Forest Calssification and Data Wrangling and Vectorization Method 1

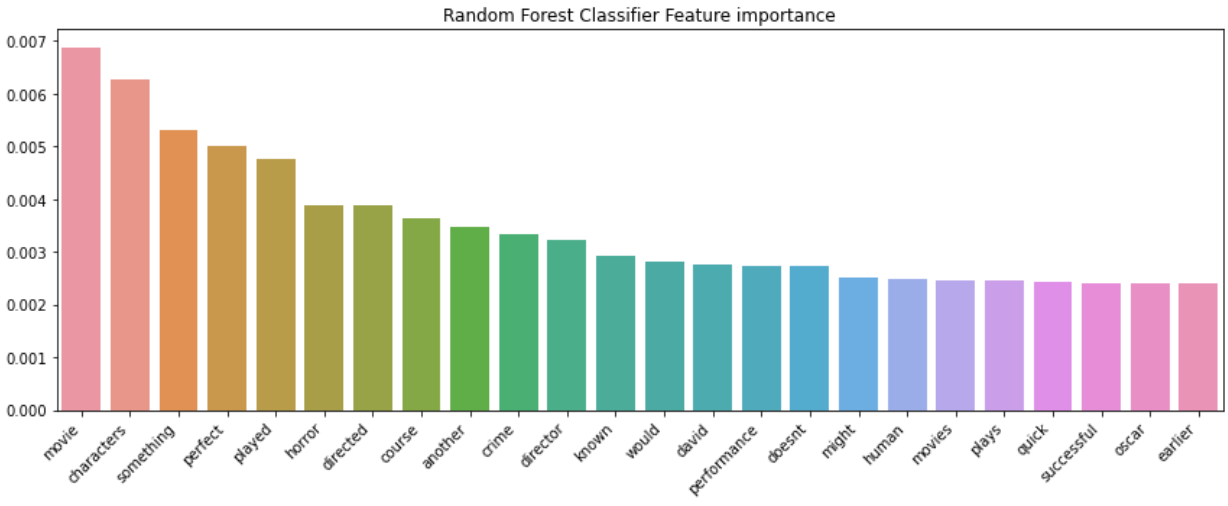
In [30]:
```
def classifiers(x, y, model_type, cv = 3):

    #this function is to fit 3 different model scenarios.  Support vector machines, lo
    #svm = Support vector machin
    #logistic = Logistic regression
    #naive_bayes = Naive Bayes Multinomial

    #can define cv value for cross validation.

    #function returns the train test split scores of each model.

    number_of_categories = y.nunique()

    if model_type == 'svm':
        print("Support Vector Machine Classifier")
        model = SVC(probability = True)

    elif model_type == 'logistic':
        print("logistic Regression Classifier")
        model = LogisticRegression()

    elif model_type == 'naive_bayes':
        print("Naive Bayes Classifier")
        model = MultinomialNB()

    elif model_type == 'randomforest':
        print("Random Forest Classifier")
        model = RandomForestClassifier(n_estimators = 100, max_features = 'sqrt', boot

    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.10, random_s
    model.fit(X_train, y_train)


    predictions = model.predict(X_test)
    probabilities = model.predict_proba(X_test)[:,1]

    accy = accuracy_score(y_test, predictions)
    F1_Score = f1_score(y_test, predictions, average = 'micro')
    print("Accuracy = ", round(100*accy,1), "%")
    print("F1 Score = ", round(F1_Score,3))

    if number_of_categories == 2:
```

```
        precision = precision_score(y_test, predictions)
        recall = recall_score(y_test, predictions)
        auc = roc_auc_score(y_test, probabilities)
        print("Precision = ", round(precision,3))
        print("Recall = ", round(recall,3))
        print("AUC = ", round(auc, 3))

    print(" ")
    print("Confusion Matrix")

    # Create the confusion matrix of the predictions
    cm = confusion_matrix(y_test, predictions)
    ConfusionMatrixDisplay(confusion_matrix=cm).plot();


    # Create the ROC Curve
    if number_of_categories == 2:
        fpr, tpr, _ = roc_curve(y_test, probabilities)
        roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
        plt.title('ROC Curve')

    if model_type == 'randomforest':
        # Calculate feature importances
        importances = model.feature_importances_

        # Visualize Feature Importance
        # Sort feature importances in descending order
        indices = np.argsort(importances)[::-1]

        # Rearrange feature names so they match the sorted feature importances
        names = [X_train.columns[i] for i in indices]
        abridged_names = names[1:25]

        plt.figure(figsize = (12, 5))
#       sns.set_style("whitegrid")
        chart = sns.barplot(x = abridged_names, y=importances[indices][1:25])
        plt.xticks(rotation=45, horizontalalignment='right', fontweight='light')
        plt.title('Random Forest Classifier Feature importance')
        plt.tight_layout()



classifiers(tfidf_matrix_method_one, review_type_labels, 'randomforest')
```

```
Random Forest Classifier
Accuracy =  31.6 %
F1 Score =  0.316
Precision =  0.286
Recall =  0.2
AUC =  0.394

Confusion Matrix
```





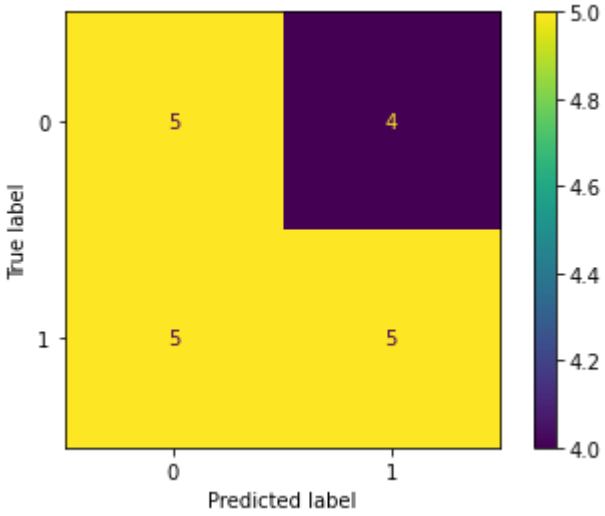


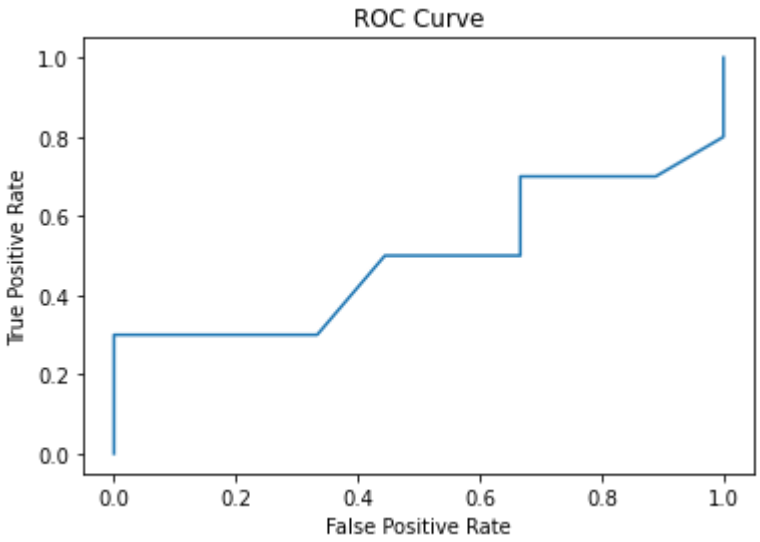
Sentiment Analysis Experiment 2: Random Forest Classification and Data Wrangling and Vectorization Method 2

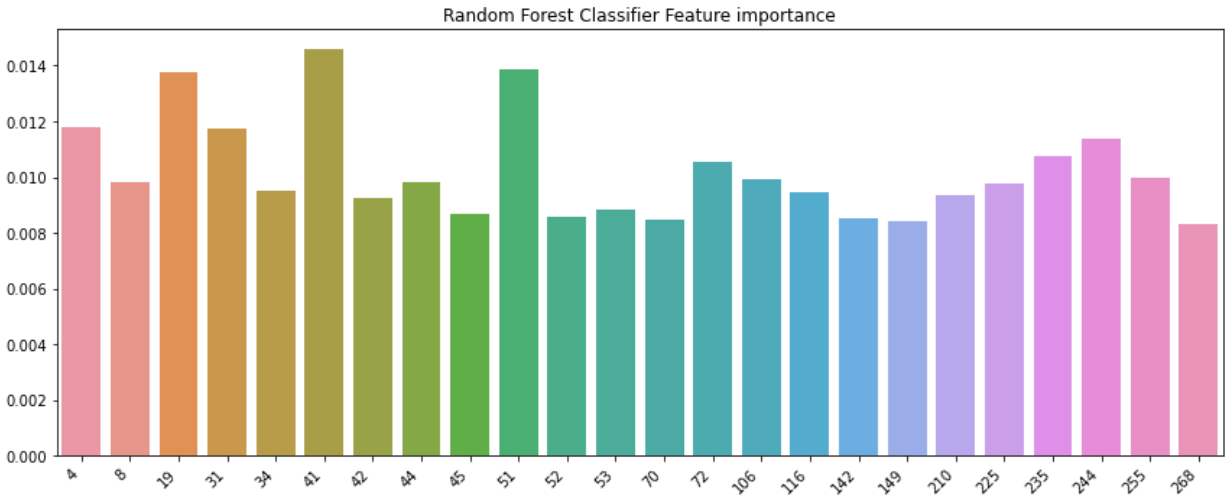
In [149…

```
classifiers(doc2vec_df, review_type_labels, 'randomforest')
```

```
Random Forest Classifier
Accuracy =  52.6 %
F1 Score =  0.526
Precision =  0.556
Recall =  0.5
AUC =  0.494

Confusion Matrix
```



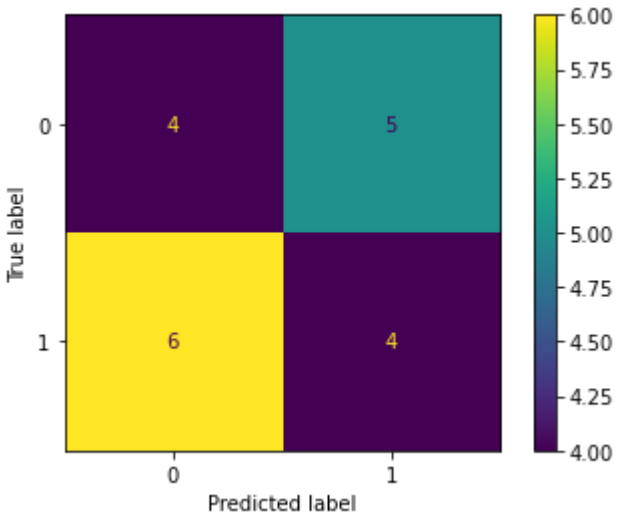


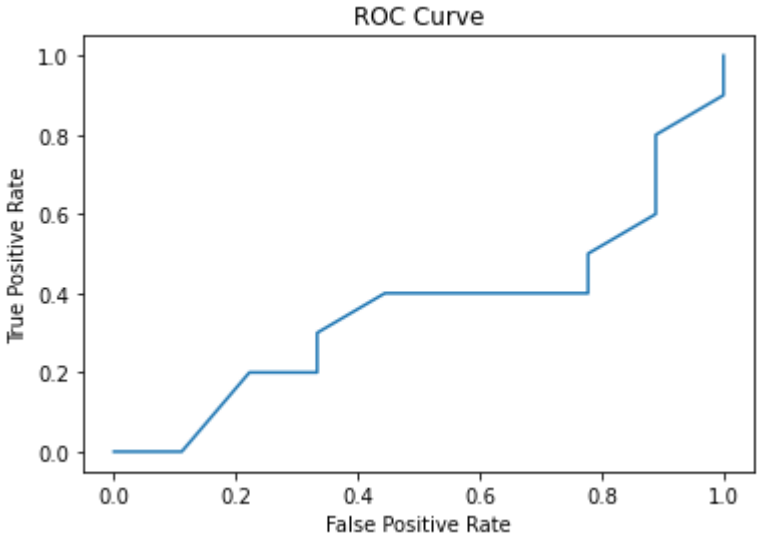


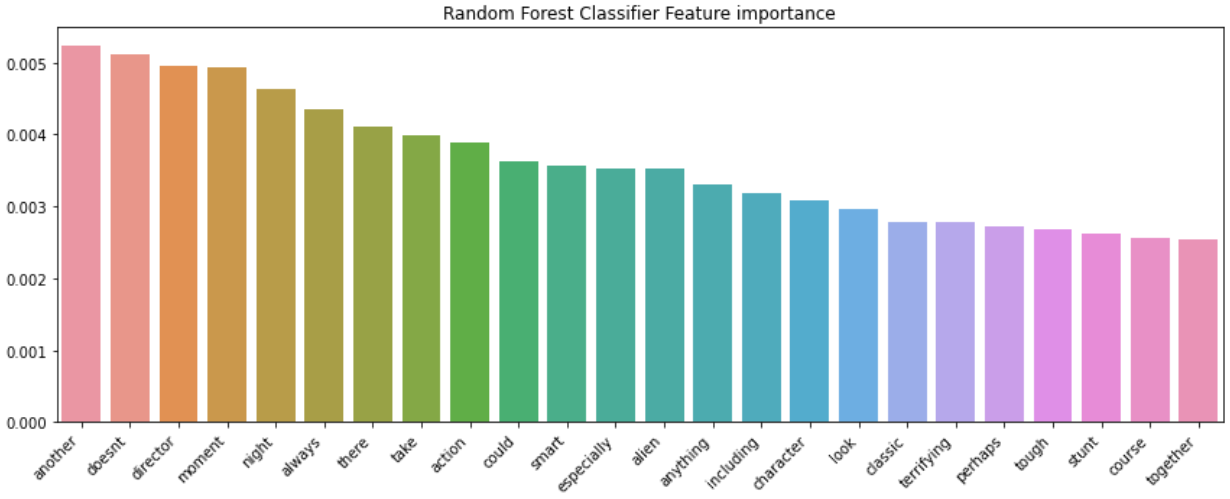
Sentiment Analysis Experiment 3: Random Forest Classification and Data Wrangling and Vectorization Method 3

In [31]:
```
classifiers(tfidf_matrix_method_four, review_type_labels, 'randomforest')
```

```
Random Forest Classifier
Accuracy =  42.1 %
F1 Score =  0.421
Precision =  0.444
Recall =  0.4
AUC =  0.361

Confusion Matrix
```







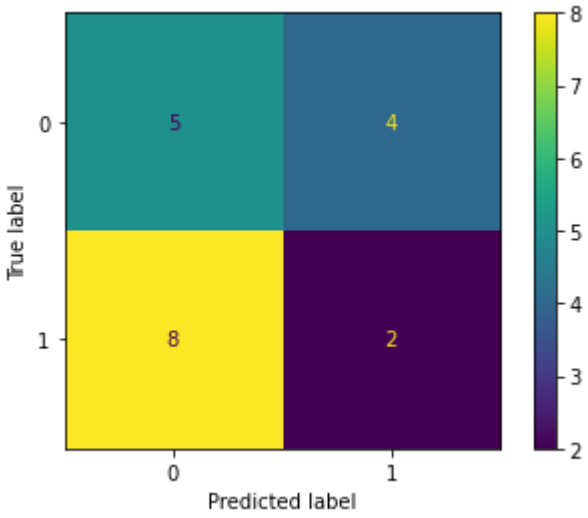
2.2) Naive Bayes Classifier Experiments

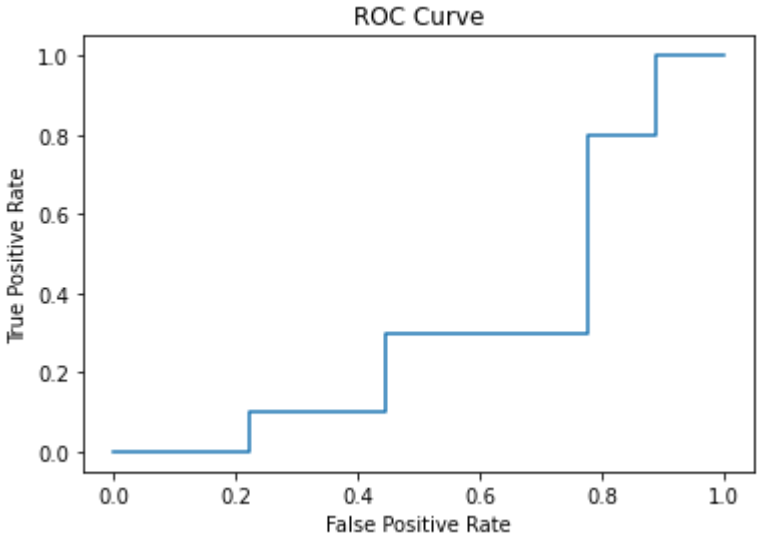
Let's apply Naive Bayes Classificaiton to our corpus to determine how effectively we can predict which reviews are positive or negative.

Sentiment Analysis Experiment 4: Naive Bayes Classification and Data Wrangling and Vectorization Method 1

In [32]:
```
classifiers(tfidf_matrix_method_one, review_type_labels, 'naive_bayes')
```

```
Naive Bayes Classifier
Accuracy =  36.8 %
F1 Score =  0.368
Precision =  0.333
Recall =  0.2
AUC =  0.322

Confusion Matrix
```

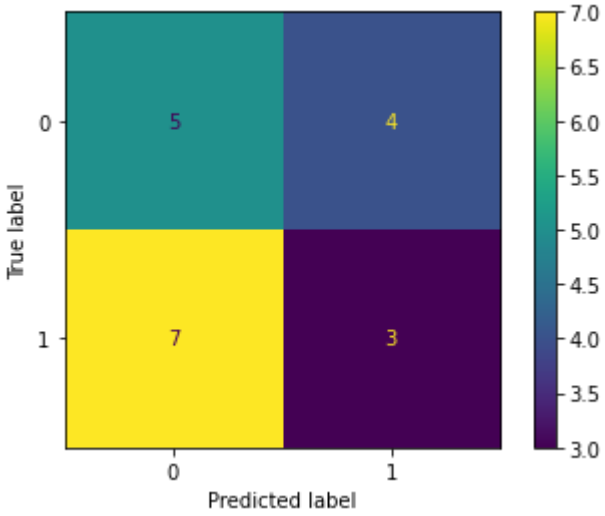


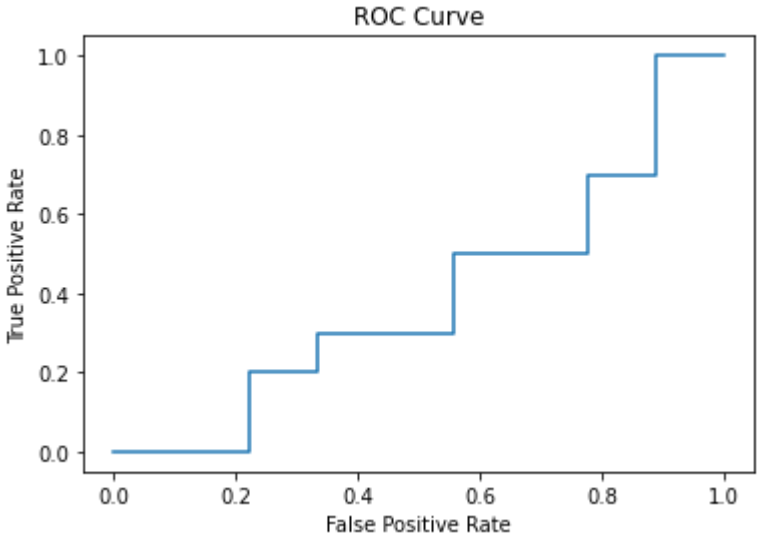


Sentiment Analysis Experiment 5: Naive Bayes Classification and Data Wrangling and Vectorization Method 3

In [33]:
```
classifiers(tfidf_matrix_method_four, review_type_labels, 'naive_bayes')
```

```
Naive Bayes Classifier
Accuracy =  42.1 %
F1 Score =  0.421
Precision =  0.429
Recall =  0.3
AUC =  0.389

Confusion Matrix
```





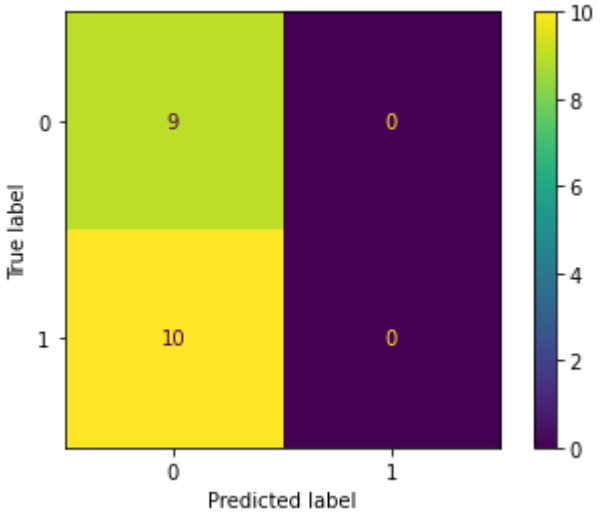
2.3) Support Vector Machine Classification Experiments

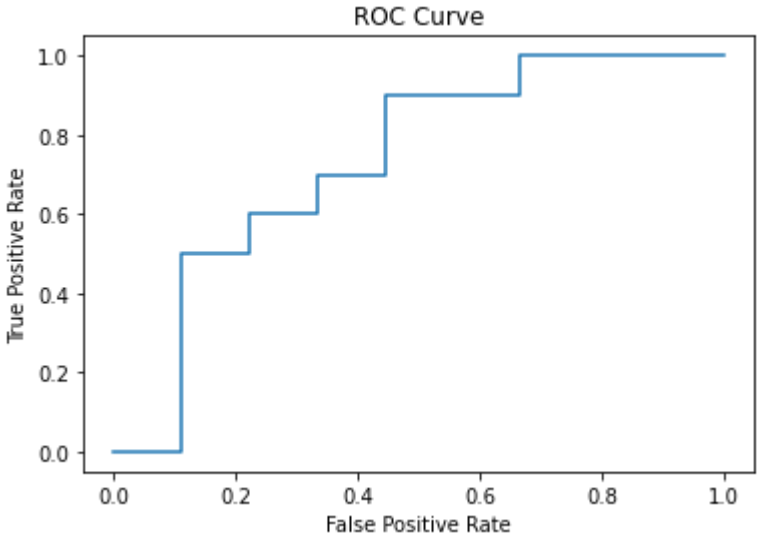
Let's apply Support Vector Machine Classification to our corpus to determine how effectively we can predict which reviews are positive or negative.

Sentiment Analysis Experiment 6: SVM Classification with Data Wrangling and Vectorization Method 1

In [34]:
```
classifiers(tfidf_matrix_method_one, review_type_labels, 'svm')
```

```
Support Vector Machine Classifier
Accuracy =  47.4 %
F1 Score =  0.474
Precision =  0.0
Recall =  0.0
AUC =  0.733

Confusion Matrix
```



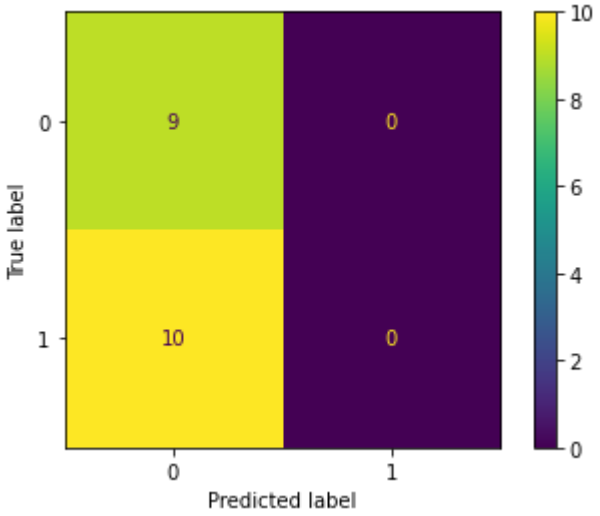


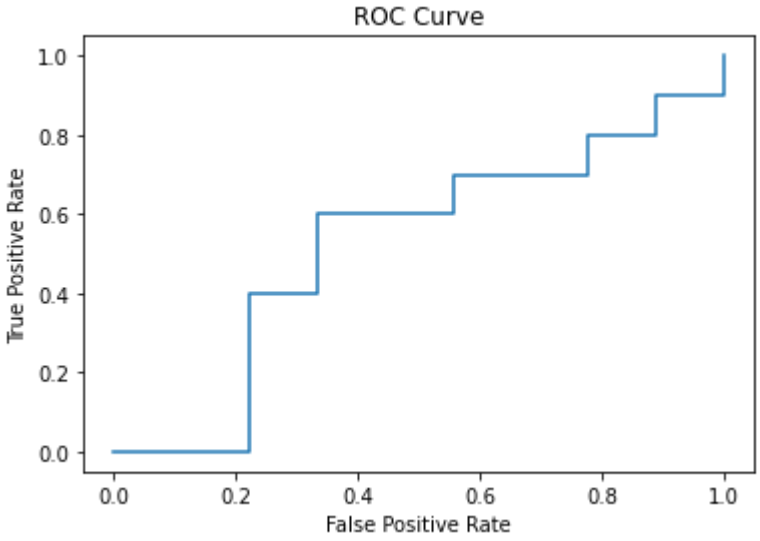
Sentiment Analysis Experiment 7: SVM Classification with Data Wrangling and Vectorization Mehtod 2

In [151…

```
classifiers(doc2vec_df, review_type_labels, 'svm')
```

```
Support Vector Machine Classifier
Accuracy =  47.4 %
F1 Score =  0.474
Precision =  0.0
Recall =  0.0
AUC =  0.522

Confusion Matrix
```

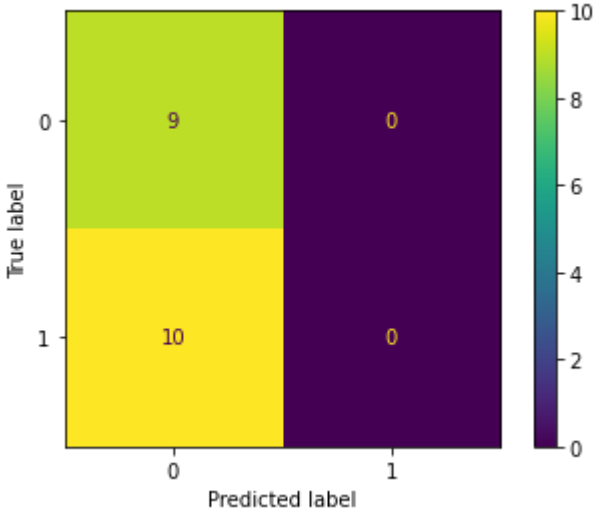


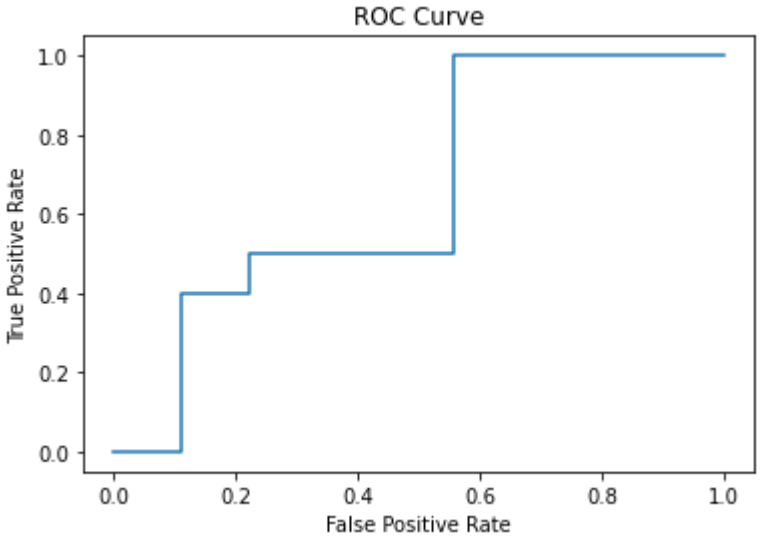


Sentiment Analysis Experiment 8: SVM Classification with Data Wrangling and Vectorization Method 3

In [35]:
```
classifiers(tfidf_matrix_method_four, review_type_labels, 'svm')
```

```
Support Vector Machine Classifier
Accuracy =  47.4 %
F1 Score =  0.474
Precision =  0.0
Recall =  0.0
AUC =  0.656

Confusion Matrix
```





3) Multi-Class Classification

Let's apply various classification algorithms to our corpus to determine whether Random Forest Classification, Naive Bayes Classification, or Support Vector Machine Classification perform well at predicting movie genre using movie review data.

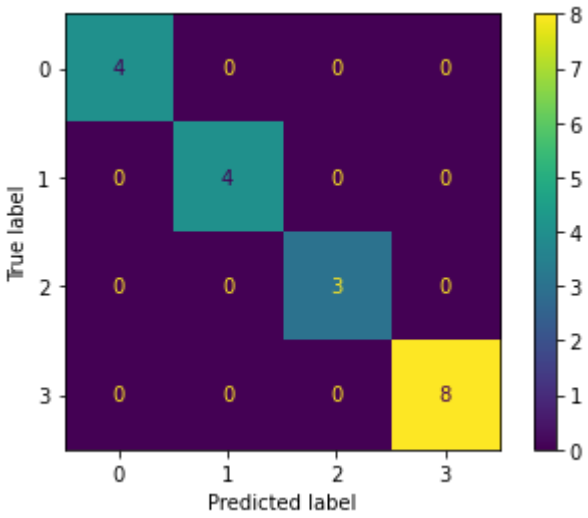
3.1) Random Forest Classifier Experiments

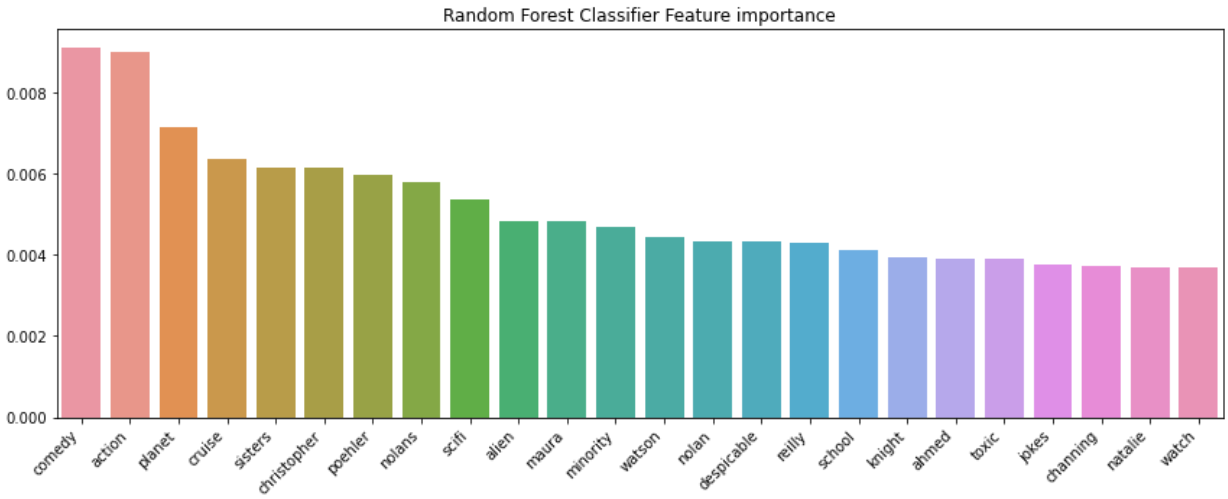
Let's apply Random Forest Classification to our corpus to predict genre based on the movie review data.

Genre Classification Experiment 1: Random Forest Classification with Data Wrangling and Vectorization Method 1

In [36]:
```
genre_labels = corpus_df['Genre of Movie']

classifiers(tfidf_matrix_method_one, genre_labels, 'randomforest')
```

```
Random Forest Classifier
Accuracy =  100.0 %
F1 Score =  1.0

Confusion Matrix
```



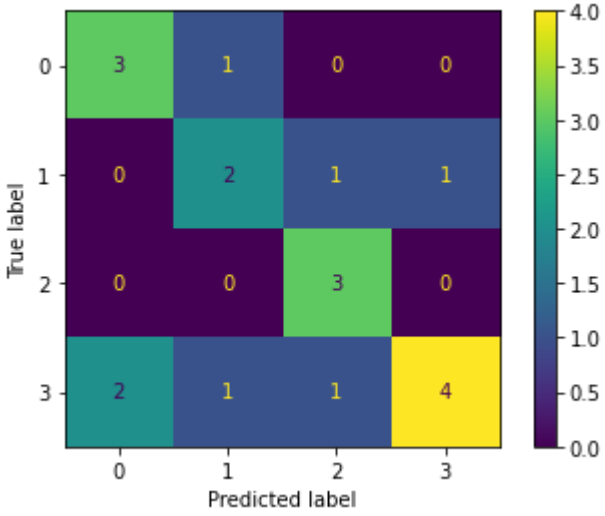


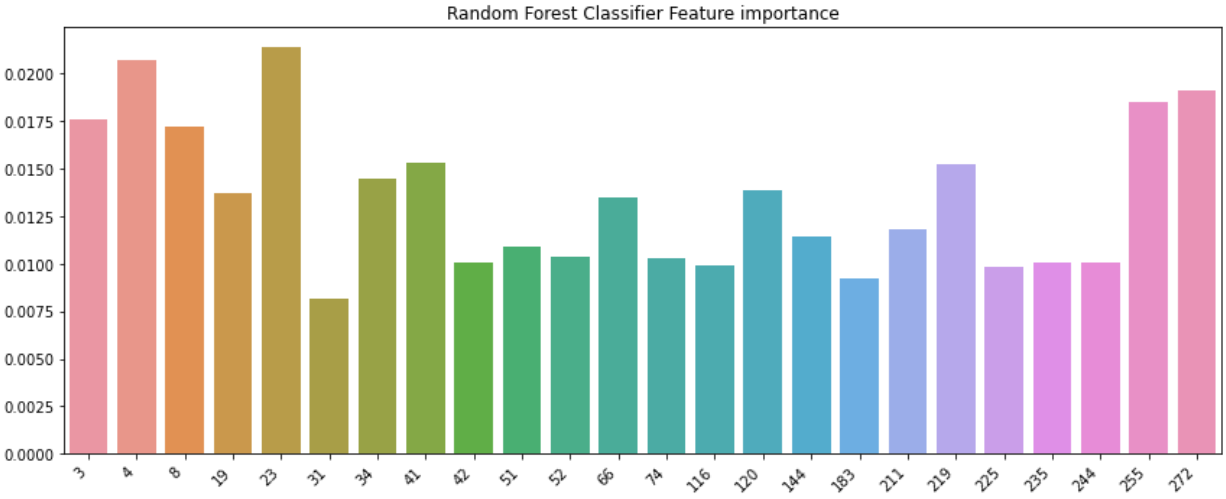
Genre Classification Experiment 2: Random Forest Classification with Data Wrangling and Vectorization Method 2

In [152…

```
classifiers(doc2vec_df, genre_labels, 'randomforest')
```

```
Random Forest Classifier
Accuracy =  63.2 %
F1 Score =  0.632

Confusion Matrix
```

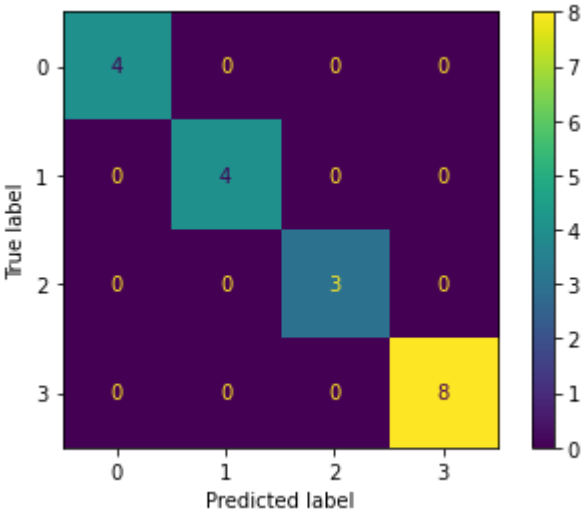


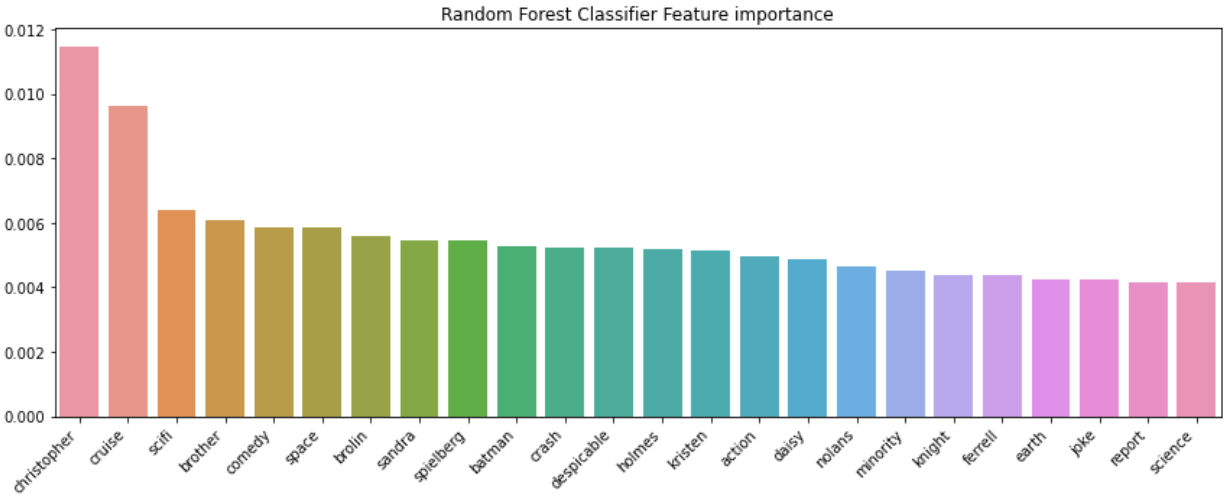


Genre Classification Experiment 3: Random Forest Classification with Data Wrangling and Vectorization Method 3

In [37]:
```
classifiers(tfidf_matrix_method_four, genre_labels, 'randomforest')
```

```
Random Forest Classifier
Accuracy =  100.0 %
F1 Score =  1.0

Confusion Matrix
```





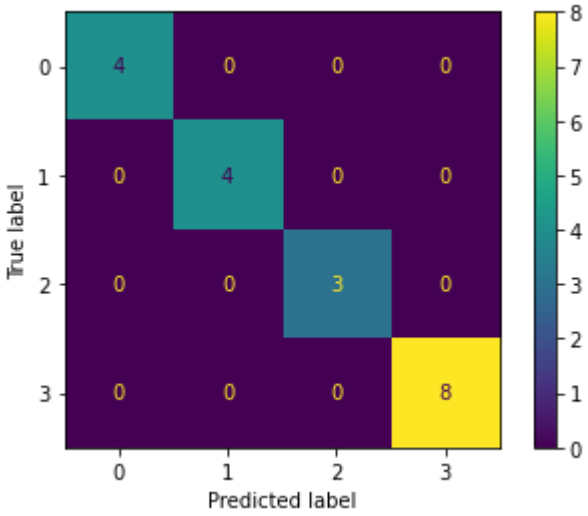
3.2) Naive Bayes Classifier Experiments

Let's apply Naive Bayes Classification to our corpus to examine how well we can predict movie genre based on the movie review.

Genre Classification Experiment 4: Naive Bayes Classification with Data Wrangling and Vectorization Method 1

In [38]:
```
classifiers(tfidf_matrix_method_one, genre_labels, 'naive_bayes')
```

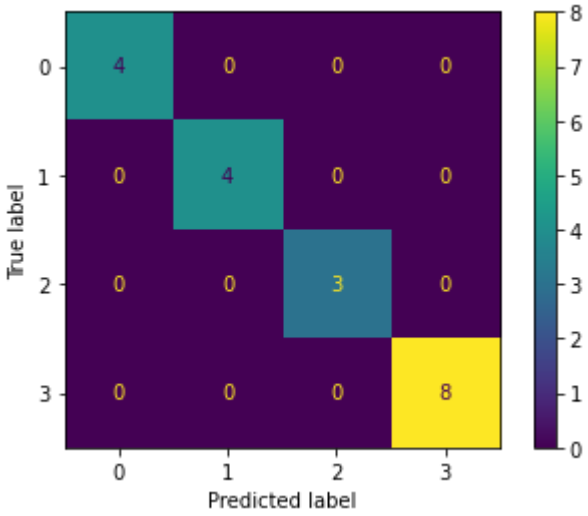
```
Naive Bayes Classifier
Accuracy =  100.0 %
F1 Score =  1.0

Confusion Matrix
```



Genre Classification Experiment 5: Naive Bayes Classification with Data Wrangling and Vectorization Method 3

In [39]:
```
classifiers(tfidf_matrix_method_four, genre_labels, 'naive_bayes')
```

```
Naive Bayes Classifier
Accuracy =  100.0 %
F1 Score =  1.0

Confusion Matrix
```



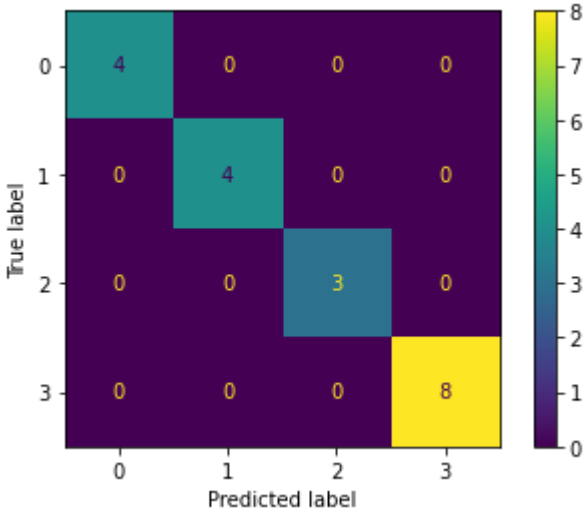
3.3) Support Vector Machine Experiments

Let's apply Support Vector Machine Classification to our corpus data to examine how well we can predict movie genre using the movie review data.

Genre Classification Experiment 6: SVM Classification with Data Wrangling and Vectorization Method 1

In [40]:
```
classifiers(tfidf_matrix_method_one, genre_labels, 'svm')
```

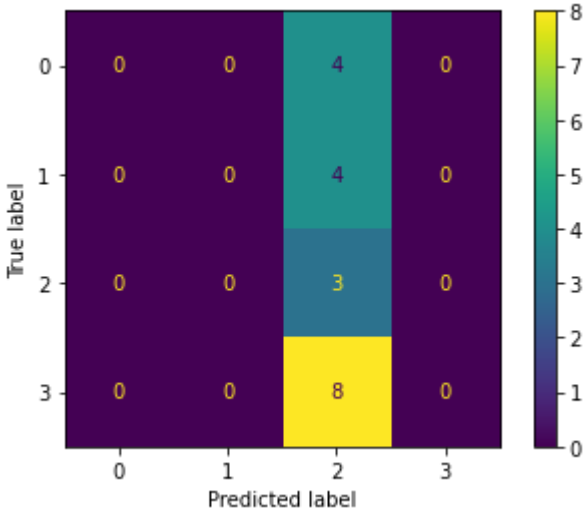
```
Support Vector Machine Classifier
Accuracy =  100.0 %
F1 Score =  1.0

Confusion Matrix
```



Genre Classification Experiment 7: SVM Classification with Data Wrangling and Vectorization Method 2

In [154…
```
classifiers(doc2vec_df, genre_labels, 'svm')
```

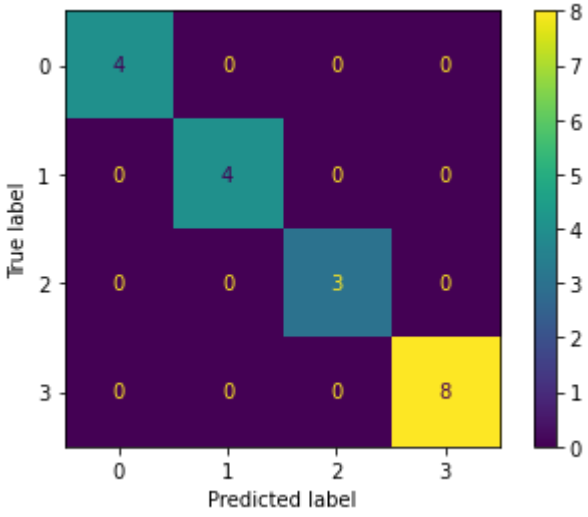
```
Support Vector Machine Classifier
Accuracy =  15.8 %
F1 Score =  0.158

Confusion Matrix
```



Genre Classification Experiment 8: SVM Classification with Data Wrangling and Vectorization Method 3

In [41]:
```
classifiers(tfidf_matrix_method_four, genre_labels, 'svm')
```

```
Support Vector Machine Classifier
Accuracy =  100.0 %
F1 Score =  1.0

Confusion Matrix
```



4) Topic Modeling

4.1) Latent Semantic Analysis Experiments

Let's implement latent semantic analysis to determine its effectiveness at topic modeling for our corpus of movie reviews.

In [42]:
```
tfidf_matrix_method_one
```

Out[42]:

	aardman	aaron	abandon	abandoned	abandonment	abandons	abdicate	abc
SAR_Doc1_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc2_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc3_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc4_Covenant	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
SAR_Doc5_Covenant	0.0	0.0	0.0	0.099853	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	
OSO_Doc3_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
OSO_Doc4_PitchBlack	0.0	0.0	0.0	0.063045	0.0	0.0	0.0	
OSO_Doc5_PitchBlack	0.0	0.0	0.0	0.057051	0.0	0.0	0.0	
OSO_Doc6_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	
OSO_Doc7_PitchBlack	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	

190 rows × 11273 columns

In [43]:
```
def create_gensim_lsa_model(doc_clean, number_of_topics, words):

    # Creating the term dictionary of our courpus, where every unique term is assigned
    dictionary = corpora.Dictionary(doc_clean)
    # Converting list of documents (corpus) into Document Term Matrix using dictionary
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
    # generate LSA model
    # train model
    lsamodel = LsiModel(doc_term_matrix
                        ,num_topics=number_of_topics
                        ,id2word = dictionary
                        ,power_iters=100)
    print(lsamodel.print_topics(num_topics=number_of_topics, num_words=words))

    topic_word_tuples = lsamodel.print_topics(num_topics=number_of_topics, num_words=w
    for i in topic_word_tuples:
        topic_word_list.append(i)

    index = similarities.MatrixSimilarity(lsamodel[doc_term_matrix])

    return lsamodel, dictionary, index



def lsa(tfidf_matrix, terms, n_components = 10):
    #this is a function to execute lsa.  inputs to the function include the tfidf matr
    #the desired number of components.

    LSA = TruncatedSVD(n_components=10)
    LSA.fit(tfidf_matrix)
```

```
    for i, comp in enumerate(LSA.components_):
        terms_comp = zip(terms, comp)
        sorted_terms = sorted(terms_comp, key= lambda x:x[1], reverse=True)[:7]
        print("Topic "+str(i)+": ")
        for t in sorted_terms:
            print(t[0])

#    if number_of_topics == 2:
#        pd.options.display.float_format = '{:,.16f}'.format
#        topic_encoded_df = pd.DataFrame(LSA, columns = ["topic_1", "topic_2"])
#        topic_encoded_df["DSI_Title"] = corpus_df['DSI_Title']
#        display(topic_encoded_df[["DSI_Title", "topic_1", "topic_2"]])

#        display(topic_encoded_df[["DSI_Title", "topic_1", "topic_2"]])


def word2vec(processed_text, size = 100):
    #This is a function to generate the word2vec matrix. Input parameters include the
    #tokenized text and matrix size

    #word to vec
    model_w2v = Word2Vec(processed_text, size=100, window=5, min_count=1, workers=4)

    #join all processed DSI words into single list
    processed_text_w2v=[]
    for i in processed_text:
        for k in i:
            processed_text_w2v.append(k)

    #obtian all the unique words from DSI
    w2v_words=list(set(processed_text_w2v))

    #can also use the get_feature_names() from TFIDF to get the list of words
    #w2v_words=Tfidf.get_feature_names()

    #empty dictionary to store words with vectors
    w2v_vectors={}

    #for loop to obtain weights for each word
    for i in w2v_words:
        temp_vec=model_w2v.wv[i]
        w2v_vectors[i]=temp_vec

    #create a final dataframe to view word vectors
    w2v_df=pd.DataFrame(w2v_vectors).transpose()
    print(w2v_df)
    return w2v_df


def plot_lsa(number_of_topics, words, processed_text):

    model,dictionary,index=create_gensim_lsa_model(processed_text,number_of_topics,wor

    for doc in processed_text:
        vec_bow = dictionary.doc2bow(doc)
        vec_lsi = model[vec_bow]  # convert the query to LSI space
        sims = index[vec_lsi] # perform a similarity query against the corpus

    print(model)
```

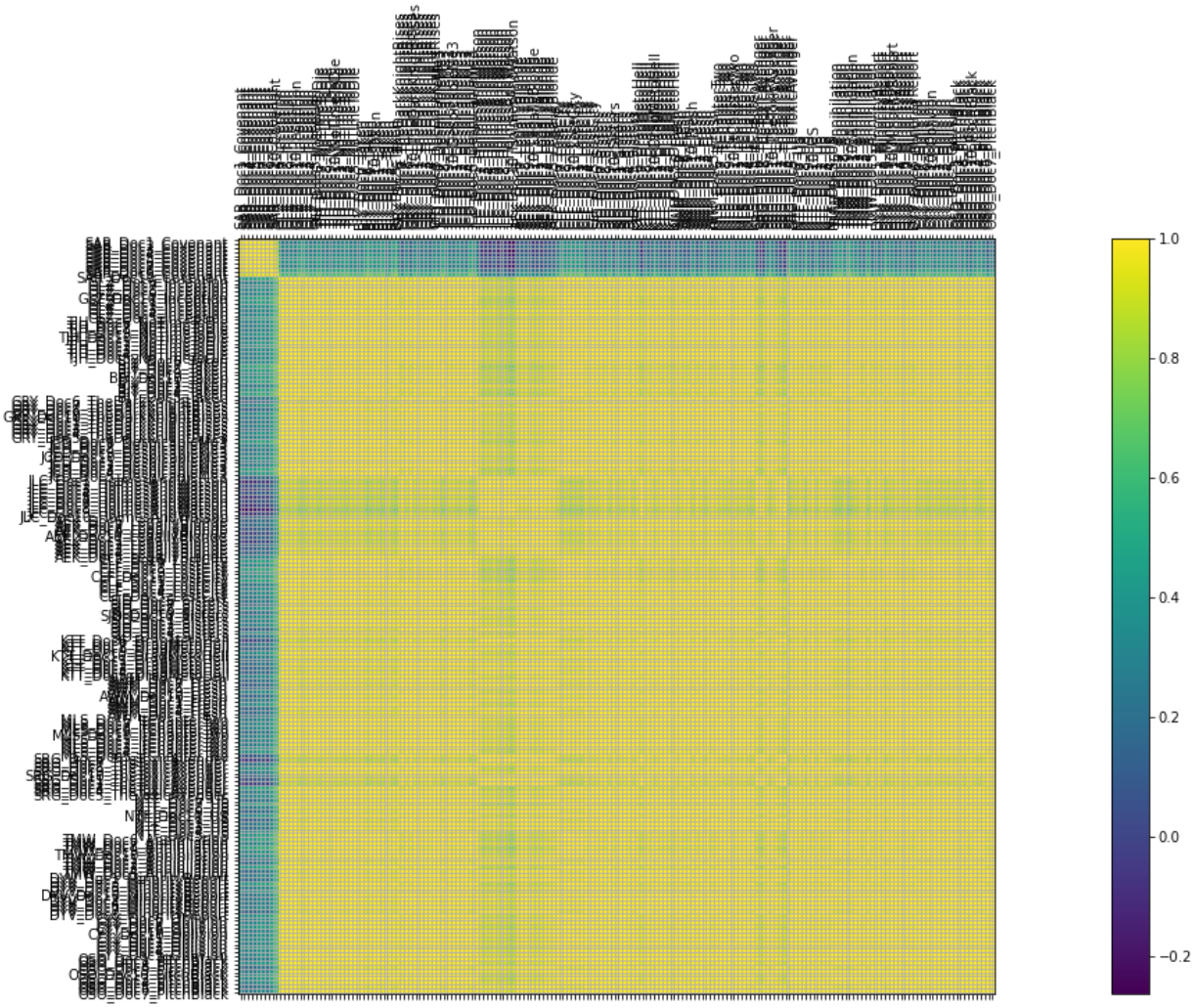
```
    print(type(model))

    fig, ax = plt.subplots(figsize=(30, 10))
    cax = ax.matshow(index, interpolation='nearest')
    ax.grid(True)
    plt.xticks(range(len(processed_text)), titles, rotation=90);
    plt.yticks(range(len(processed_text)), titles);
    fig.colorbar(cax)
    plt.show()
    return model


def create_topics_words_df(number_of_topics, number_of_words):

    regex = re.compile('[^a-zA-Z]')

    for i in topic_word_list:
        topic_number_list.append(i[0])
        words_list.append(i[1])

    for j in words_list:
        split_words_list.append(j.split("+"))

    for k in split_words_list:
        for l in k:
            processed_words_list.append(regex.sub('', l))


    if number_of_topics == 2:
        topic_zero_words = processed_words_list[0: number_of_words]
        topic_one_words = processed_words_list[number_of_words : 2*number_of_words ]

        topic_words_df = pd.DataFrame(list(zip(topic_zero_words,
                                               topic_one_words)),
               columns = ['Topic 0',
                          'Topic 1'])

        display(topic_words_df)


    elif number_of_topics == 4:
        topic_zero_words = processed_words_list[0: number_of_words]
        topic_one_words = processed_words_list[number_of_words : 2*number_of_words ]
        topic_two_words = processed_words_list[2*number_of_words: 3*number_of_words]
        topic_three_words = processed_words_list[3*number_of_words : 4*number_of_words

        topic_words_df = pd.DataFrame(list(zip(topic_zero_words,
                                               topic_one_words,
                                               topic_two_words,
                                               topic_three_words)),
               columns = ['Topic 0',
                          'Topic 1',
                          'Topic 2',
                          'Topic 3'])

        display(topic_words_df)

    elif number_of_topics == 19:
        topic_zero_words = processed_words_list[0: number_of_words]
        topic_one_words = processed_words_list[number_of_words : 2*number_of_words ]
```

```
        topic_two_words = processed_words_list[2*number_of_words: 3*number_of_words]
        topic_three_words = processed_words_list[3*number_of_words : 4*number_of_words
        topic_four_words = processed_words_list[4* number_of_words : 5*number_of_words
        topic_five_words = processed_words_list[5*number_of_words : 6*number_of_words
        topic_six_words = processed_words_list[6*number_of_words: 7*number_of_words]
        topic_seven_words = processed_words_list[7*number_of_words : 8*number_of_words
        topic_eight_words = processed_words_list[8*number_of_words: 9*number_of_words]
        topic_nine_words = processed_words_list[9*number_of_words : 10*number_of_words
        topic_ten_words = processed_words_list[10*number_of_words: 11*number_of_words]
        topic_eleven_words = processed_words_list[11*number_of_words : 12*number_of_wo
        topic_twelve_words = processed_words_list[12*number_of_words: 13*number_of_wor
        topic_thirteen_words = processed_words_list[13*number_of_words : 14*number_of_
        topic_fourteen_words = processed_words_list[14*number_of_words: 15*number_of_w
        topic_fifteen_words = processed_words_list[15*number_of_words : 16*number_of_w
        topic_sixteen_words = processed_words_list[16*number_of_words : 17*number_of_w
        topic_seventeen_words = processed_words_list[17*number_of_words: 18*number_of_
        topic_eighteen_words = processed_words_list[18*number_of_words : 19*number_of_

        topic_words_df = pd.DataFrame(list(zip(topic_zero_words,
                                               topic_one_words,
                                               topic_two_words,
                                               topic_three_words,
                                               topic_four_words,
                                               topic_five_words,
                                               topic_six_words,
                                               topic_seven_words,
                                               topic_eight_words,
                                               topic_nine_words,
                                               topic_ten_words,
                                               topic_eleven_words,
                                               topic_twelve_words,
                                               topic_thirteen_words,
                                               topic_fourteen_words,
                                               topic_fifteen_words,
                                               topic_sixteen_words,
                                               topic_seventeen_words,
                                               topic_eighteen_words)),
                               columns = ['Topic 0',
                                          'Topic 1',
                                          'Topic 2',
                                          'Topic 3',
                                          'Topic 4',
                                          'Topic 5',
                                          'Topic 6',
                                          'Topic 7',
                                          'Topic 8',
                                          'Topic 9',
                                          'Topic 10',
                                          'Topic 11',
                                          'Topic 12',
                                          'Topic 13',
                                          'Topic 14',
                                          'Topic 15',
                                          'Topic 16',
                                          'Topic 17',
                                          'Topic 18'])

        display(topic_words_df)
```

Topic Modeling Experiment 1: 2-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 1

In [44]:
```
topic_word_list = []

model_2concepts_10words=plot_lsa(2, 10, processed_text = processed_text_method_one)
```

```
[(0, '-0.420*"movie" + -0.164*"films" + -0.164*"first" + -0.126*"would" + -0.125*"act
ion" + -0.124*"story" + -0.122*"people" + -0.119*"horror" + -0.118*"movies" + -0.116
*"characters"'), (1, '0.462*"ahmed" + 0.397*"kinley" + 0.282*"covenant" + 0.242*"ritc
hie" + 0.219*"taliban" + 0.178*"gyllenhaal" + 0.143*"ritchies" + 0.133*"afghanistan"
+ 0.126*"interpreters" + 0.116*"interpreter"')]
LsiModel(num_terms=11273, num_topics=2, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



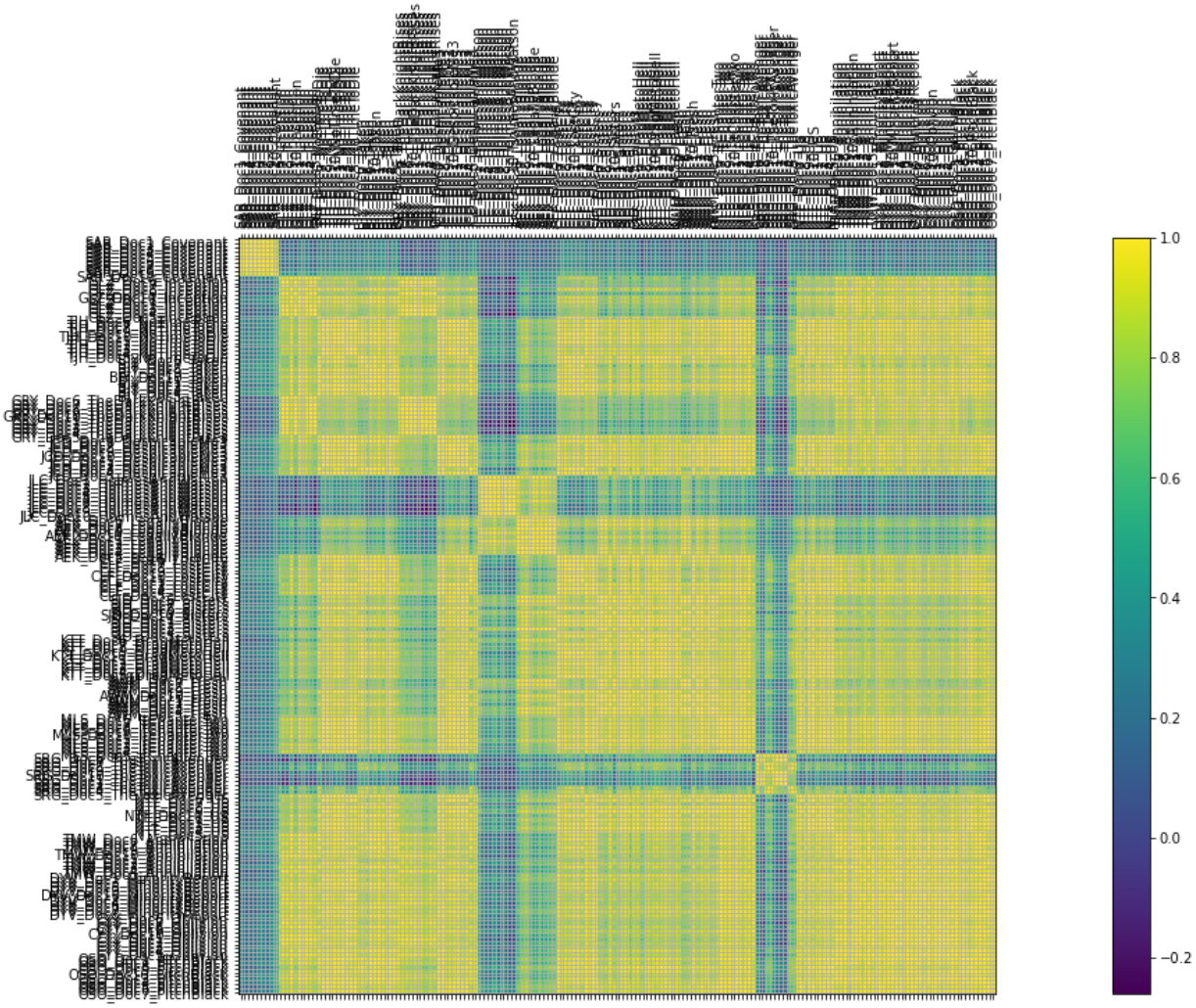
In [47]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 2,
                       number_of_words = 10)
```

	Topic 0	Topic 1
0	movie	ahmed
1	films	kinley
2	first	covenant
3	would	ritchie
4	action	taliban
5	story	gyllenhaal
6	people	ritchies
7	horror	afghanistan
8	movies	interpreters
9	characters	interpreter

Topic Modeling Experiment 2: 4-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 1

In [48]:
```
topic_word_list = []

model_4concepts_10words=plot_lsa(4, 10, processed_text = processed_text_method_one)
```

```
[(0, '-0.420*"movie" + -0.164*"films" + -0.164*"first" + -0.126*"would" + -0.125*"act
ion" + -0.124*"story" + -0.122*"people" + -0.119*"horror" + -0.118*"movies" + -0.116
*"characters"'), (1, '0.462*"ahmed" + 0.397*"kinley" + 0.282*"covenant" + 0.242*"ritc
hie" + 0.219*"taliban" + 0.178*"gyllenhaal" + 0.143*"ritchies" + 0.133*"afghanistan"
+ 0.126*"interpreters" + 0.116*"interpreter"'), (2, '-0.551*"toxic" + -0.344*"avenge
r" + -0.260*"melvin" + -0.202*"movie" + 0.189*"holmes" + -0.149*"waste" + -0.145*"tro
ma" + 0.130*"watson" + -0.109*"tromaville" + -0.099*"movies"'), (3, '0.420*"holmes" +
0.287*"watson" + 0.195*"ferrell" + 0.184*"reilly" + -0.181*"knight" + 0.179*"comedy"
+ -0.151*"movie" + 0.146*"toxic" + -0.124*"nolan" + 0.111*"blonde"')]
LsiModel(num_terms=11273, num_topics=4, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



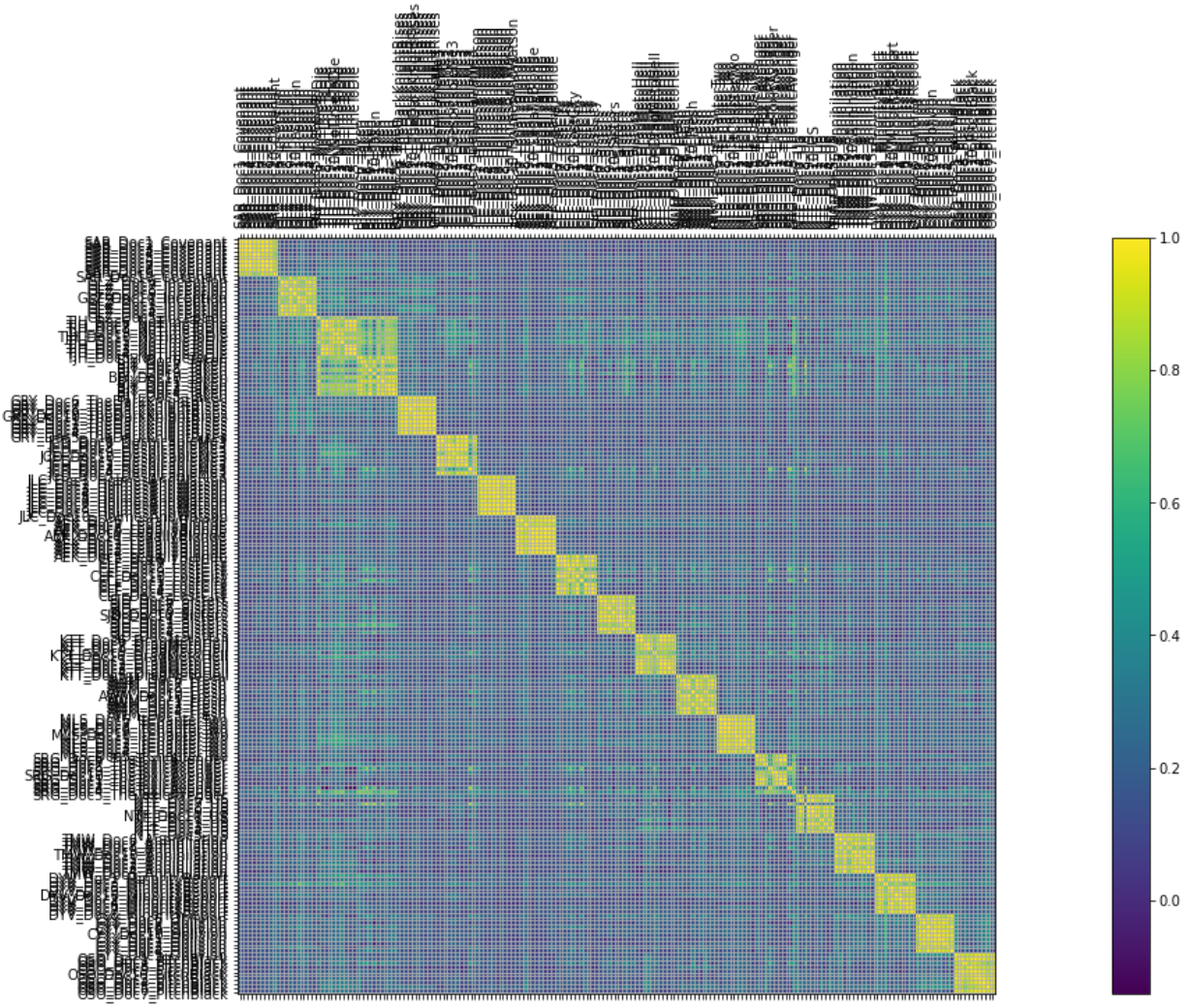
In [49]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 4,
                       number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3
0	movie	ahmed	toxic	holmes
1	films	kinley	avenger	watson
2	first	covenant	melvin	ferrell
3	would	ritchie	movie	reilly
4	action	taliban	holmes	knight
5	story	gyllenhaal	waste	comedy
6	people	ritchies	troma	movie
7	horror	afghanistan	watson	toxic
8	movies	interpreters	tromaville	nolan
9	characters	interpreter	movies	blonde

Topic Modeling Experiment 3: 19-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 1

In [50]:
```
topic_word_list = []

model_19concepts_10words=plot_lsa(19, 10, processed_text = processed_text_method_one)
```

```
[(0, '-0.420*"movie" + -0.164*"films" + -0.164*"first" + -0.126*"would" + -0.125*"act
ion" + -0.124*"story" + -0.122*"people" + -0.119*"horror" + -0.118*"movies" + -0.116
*"characters"'), (1, '0.462*"ahmed" + 0.397*"kinley" + 0.282*"covenant" + 0.242*"ritc
hie" + 0.219*"taliban" + 0.178*"gyllenhaal" + 0.143*"ritchies" + 0.133*"afghanistan"
+ 0.126*"interpreters" + 0.116*"interpreter"'), (2, '-0.551*"toxic" + -0.344*"avenge
r" + -0.260*"melvin" + -0.202*"movie" + 0.189*"holmes" + -0.149*"waste" + -0.145*"tro
ma" + 0.130*"watson" + -0.109*"tromaville" + -0.099*"movies"'), (3, '-0.420*"holmes"
+ -0.287*"watson" + -0.195*"ferrell" + -0.184*"reilly" + 0.181*"knight" + -0.179*"com
edy" + 0.151*"movie" + -0.146*"toxic" + 0.124*"nolan" + -0.111*"blonde"'), (4, '-0.28
0*"holmes" + 0.251*"blonde" + -0.234*"knight" + 0.196*"legally" + -0.193*"watson" + -
0.140*"nolan" + 0.139*"school" + 0.138*"sisters" + -0.137*"batman" + 0.130*"movie"'),
(5, '-0.427*"movie" + 0.341*"horror" + 0.182*"films" + -0.134*"sisters" + -0.128*"hol
mes" + 0.111*"chapter" + -0.111*"poehler" + 0.110*"family" + 0.105*"first" + 0.103*"p
eele"'), (6, '0.351*"blonde" + 0.272*"legally" + 0.223*"knight" + -0.197*"movie" + 0.
187*"school" + 0.171*"harvard" + 0.157*"witherspoon" + 0.148*"warner" + 0.129*"batma
n" + 0.127*"nolan"'), (7, '-0.187*"despicable" + 0.187*"oblivion" + -0.183*"knight" +
0.155*"annihilation" + -0.154*"sisters" + -0.140*"minions" + -0.131*"family" + -0.130
*"poehler" + 0.129*"cruise" + -0.115*"horror"'), (8, '-0.340*"despicable" + -0.259*"m
inions" + 0.250*"sisters" + 0.209*"poehler" + -0.185*"first" + 0.183*"horror" + 0.138
*"party" + -0.137*"bratt" + 0.136*"black" + 0.129*"maura"'), (9, '0.372*"chapter" +
0.224*"derry" + 0.215*"pennywise" + -0.192*"despicable" + 0.183*"losers" + 0.161*"jam
es" + -0.155*"steve" + -0.146*"minions" + -0.142*"fresh" + -0.140*"family"'), (10, '-
0.252*"sisters" + 0.252*"horror" + -0.211*"poehler" + 0.208*"movie" + -0.152*"anderto
n" + -0.138*"party" + -0.133*"future" + -0.129*"maura" + -0.123*"despicable" + 0.123
*"loretta"'), (11, '-0.318*"black" + -0.305*"pitch" + -0.178*"planet" + -0.167*"riddi
ck" + 0.158*"anderton" + -0.146*"despicable" + 0.133*"future" + -0.129*"diesel" + 0.1
25*"fresh" + -0.124*"creatures"'), (12, '0.275*"annihilation" + -0.217*"loretta" + 0.
180*"shimmer" + -0.140*"black" + 0.137*"garland" + -0.132*"pitch" + 0.130*"dreams" +
0.128*"jason" + -0.121*"tatum" + -0.116*"bullock"'), (13, '-0.242*"fresh" + 0.225*"ac
tion" + -0.212*"dreams" + -0.190*"steve" + 0.176*"daughter" + -0.166*"dream" + 0.144
*"neeson" + 0.128*"taken" + 0.120*"adelaide" + -0.116*"inception"'), (14, '-0.227*"an
nihilation" + 0.216*"dreams" + 0.190*"movie" + 0.180*"dream" + -0.177*"films" + 0.160
*"anderton" + -0.155*"shimmer" + 0.135*"inception" + 0.130*"minority" + -0.127*"garla
nd"'), (15, '-0.316*"loretta" + 0.206*"movie" + 0.205*"fresh" + 0.177*"steve" + -0.16
8*"tatum" + -0.159*"bullock" + -0.131*"dreams" + 0.114*"dating" + -0.100*"daniel" + -
0.097*"channing"'), (16, '0.220*"dreams" + 0.211*"action" + 0.204*"daughter" + 0.175
*"dream" + 0.163*"neeson" + 0.153*"taken" + -0.149*"knight" + -0.149*"loretta" + 0.13
0*"craig" + 0.127*"films"'), (17, '0.415*"oblivion" + 0.194*"earth" + -0.162*"annihil
ation" + 0.131*"scifi" + 0.126*"victoria" + 0.122*"cruise" + -0.117*"anderton" + -0.1
17*"black" + 0.115*"drones" + 0.112*"kosinski"'), (18, '0.365*"christine" + 0.275*"ra
imi" + -0.188*"adelaide" + -0.164*"peele" + -0.142*"family" + 0.137*"raimis" + 0.119
*"curse" + 0.117*"lohman" + 0.112*"ganush" + 0.110*"woman"')]
LsiModel(num_terms=11273, num_topics=19, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



In [51]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 19,
                       number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	
0	movie	ahmed	toxic	holmes	holmes	movie	blonde	despicable	despicable	
1	films	kinley	avenger	watson	blonde	horror	legally	oblivion	minions	
2	first	covenant	melvin	ferrell	knight	films	knight	knight	sisters	pe
3	would	ritchie	movie	reilly	legally	sisters	movie	annihilation	poehler	de
4	action	taliban	holmes	knight	watson	holmes	school	sisters	first	
5	story	gyllenhaal	waste	comedy	nolan	chapter	harvard	minions	horror	
6	people	ritchies	troma	movie	school	poehler	witherspoon	family	party	
7	horror	afghanistan	watson	toxic	sisters	family	warner	poehler	bratt	
8	movies	interpreters	tromaville	nolan	batman	first	batman	cruise	black	
9	characters	interpreter	movies	blonde	movie	peele	nolan	horror	maura	

In [53]:
```
topics = [2, 4, 19]
coherence_values = []
index_number = 0
words = 10

def lsa_coherence_function(processed_text, topics, words):
    lsamodel, dictionary, index = create_gensim_lsa_model(processed_text, t, words)


    coherence_model_lsa = CoherenceModel(model=lsamodel,
                                         dictionary=dictionary,
                                         texts=processed_text,
                                         coherence='c_v')
    coherence_lsa = coherence_model_lsa.get_coherence()
    coherence_values.append(coherence_lsa)

for t in topics:
    lsa_coherence_function(processed_text = processed_text_method_one,
                           topics = t,
                           words = 10)

    coherence ={f'{topics[index_number]} concepts and {words} words':coherence_values[

    print(coherence)

    index_number += 1
```

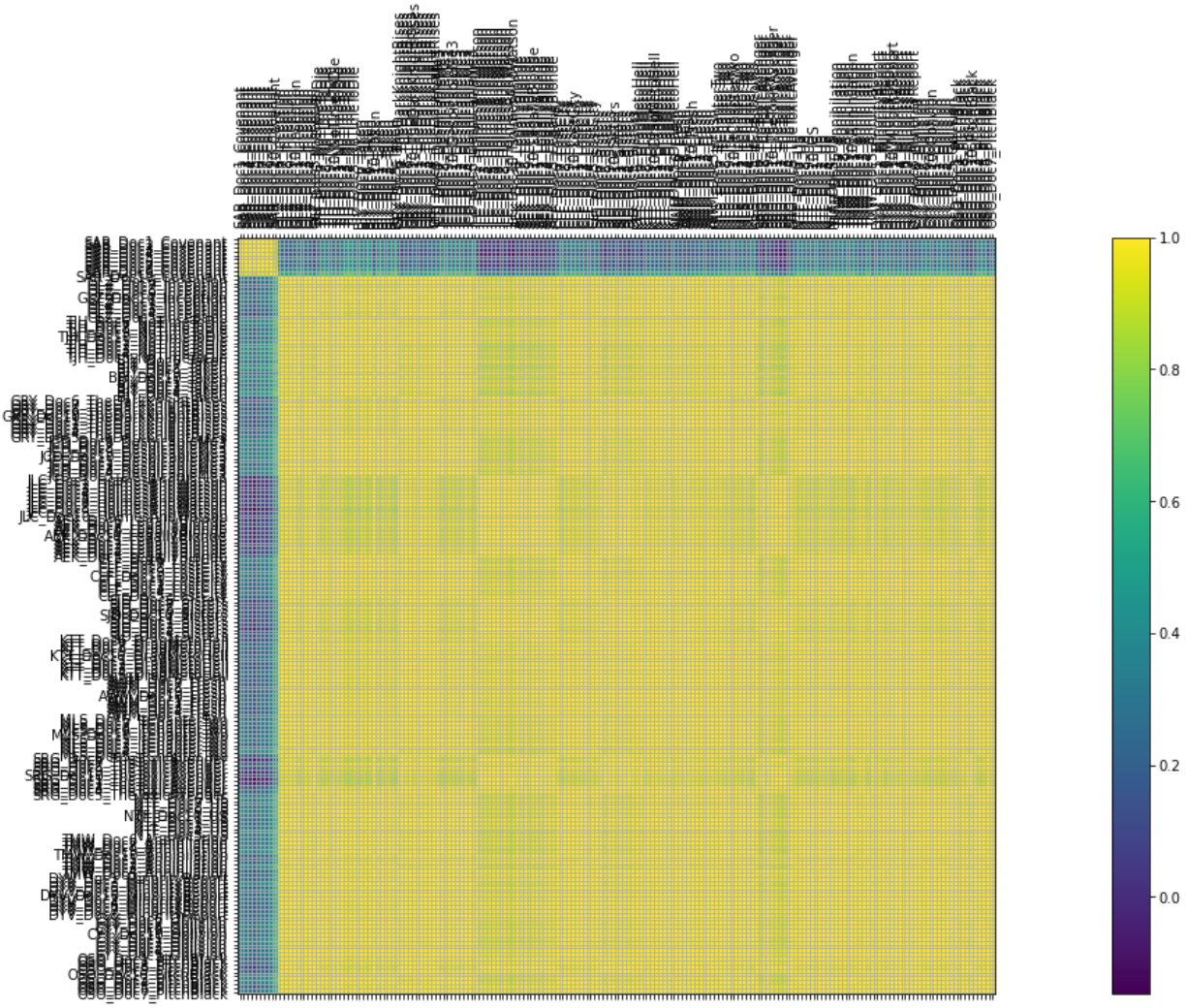
```
[(0, '-0.420*"movie" + -0.164*"films" + -0.164*"first" + -0.126*"would" + -0.125*"act
ion" + -0.124*"story" + -0.122*"people" + -0.119*"horror" + -0.118*"movies" + -0.116
*"characters"'), (1, '-0.462*"ahmed" + -0.397*"kinley" + -0.282*"covenant" + -0.242
*"ritchie" + -0.219*"taliban" + -0.178*"gyllenhaal" + -0.143*"ritchies" + -0.133*"afg
hanistan" + -0.126*"interpreters" + -0.116*"interpreter"')]
{'2 concepts and 10 words': 0.5677282941826053}
[(0, '-0.420*"movie" + -0.164*"films" + -0.164*"first" + -0.126*"would" + -0.125*"act
ion" + -0.124*"story" + -0.122*"people" + -0.119*"horror" + -0.118*"movies" + -0.116
*"characters"'), (1, '0.462*"ahmed" + 0.397*"kinley" + 0.282*"covenant" + 0.242*"ritc
hie" + 0.219*"taliban" + 0.178*"gyllenhaal" + 0.143*"ritchies" + 0.133*"afghanistan"
+ 0.126*"interpreters" + 0.116*"interpreter"'), (2, '-0.551*"toxic" + -0.344*"avenge
r" + -0.260*"melvin" + -0.202*"movie" + 0.189*"holmes" + -0.149*"waste" + -0.145*"tro
ma" + 0.130*"watson" + -0.109*"tromaville" + -0.099*"movies"'), (3, '-0.420*"holmes"
+ -0.287*"watson" + -0.195*"ferrell" + -0.184*"reilly" + 0.181*"knight" + -0.179*"com
edy" + 0.151*"movie" + -0.146*"toxic" + 0.124*"nolan" + -0.111*"blonde"')]
{'4 concepts and 10 words': 0.629412312843186}
[(0, '-0.420*"movie" + -0.164*"films" + -0.164*"first" + -0.126*"would" + -0.125*"act
ion" + -0.124*"story" + -0.122*"people" + -0.119*"horror" + -0.118*"movies" + -0.116
*"characters"'), (1, '0.462*"ahmed" + 0.397*"kinley" + 0.282*"covenant" + 0.242*"ritc
hie" + 0.219*"taliban" + 0.178*"gyllenhaal" + 0.143*"ritchies" + 0.133*"afghanistan"
+ 0.126*"interpreters" + 0.116*"interpreter"'), (2, '-0.551*"toxic" + -0.344*"avenge
r" + -0.260*"melvin" + -0.202*"movie" + 0.189*"holmes" + -0.149*"waste" + -0.145*"tro
ma" + 0.130*"watson" + -0.109*"tromaville" + -0.099*"movies"'), (3, '-0.420*"holmes"
+ -0.287*"watson" + -0.195*"ferrell" + -0.184*"reilly" + 0.181*"knight" + -0.179*"com
edy" + 0.151*"movie" + -0.146*"toxic" + 0.124*"nolan" + -0.111*"blonde"'), (4, '0.280
*"holmes" + -0.251*"blonde" + 0.234*"knight" + -0.196*"legally" + 0.193*"watson" + 0.
140*"nolan" + -0.139*"school" + -0.138*"sisters" + 0.137*"batman" + -0.130*"movie"'),
(5, '-0.427*"movie" + 0.341*"horror" + 0.182*"films" + -0.134*"sisters" + -0.128*"hol
mes" + 0.111*"chapter" + -0.111*"poehler" + 0.110*"family" + 0.105*"first" + 0.103*"p
eele"'), (6, '-0.351*"blonde" + -0.272*"legally" + -0.223*"knight" + 0.197*"movie" +
-0.187*"school" + -0.171*"harvard" + -0.157*"witherspoon" + -0.148*"warner" + -0.129
*"batman" + -0.127*"nolan"'), (7, '-0.187*"despicable" + 0.187*"oblivion" + -0.183*"k
night" + 0.155*"annihilation" + -0.154*"sisters" + -0.140*"minions" + -0.131*"family"
+ -0.130*"poehler" + 0.129*"cruise" + -0.115*"horror"'), (8, '-0.340*"despicable" + -
0.259*"minions" + 0.250*"sisters" + 0.209*"poehler" + -0.185*"first" + 0.183*"horror"
+ 0.138*"party" + -0.137*"bratt" + 0.136*"black" + 0.129*"maura"'), (9, '0.372*"chapt
er" + 0.224*"derry" + 0.215*"pennywise" + -0.192*"despicable" + 0.183*"losers" + 0.16
1*"james" + -0.155*"steve" + -0.146*"minions" + -0.142*"fresh" + -0.140*"family"'),
(10, '-0.252*"sisters" + 0.252*"horror" + -0.211*"poehler" + 0.208*"movie" + -0.152
*"anderton" + -0.138*"party" + -0.133*"future" + -0.129*"maura" + -0.123*"despicable"
+ 0.123*"loretta"'), (11, '-0.318*"black" + -0.305*"pitch" + -0.178*"planet" + -0.167
*"riddick" + 0.158*"anderton" + -0.146*"despicable" + 0.133*"future" + -0.129*"diese
l" + 0.125*"fresh" + -0.124*"creatures"'), (12, '-0.275*"annihilation" + 0.217*"loret
ta" + -0.180*"shimmer" + 0.140*"black" + -0.137*"garland" + 0.132*"pitch" + -0.130*"d
reams" + -0.128*"jason" + 0.121*"tatum" + 0.116*"bullock"'), (13, '-0.242*"fresh" +
0.225*"action" + -0.212*"dreams" + -0.190*"steve" + 0.176*"daughter" + -0.166*"dream"
+ 0.144*"neeson" + 0.128*"taken" + 0.120*"adelaide" + -0.116*"inception"'), (14, '0.2
27*"annihilation" + -0.216*"dreams" + -0.190*"movie" + -0.180*"dream" + 0.177*"films"
+ -0.160*"anderton" + 0.155*"shimmer" + -0.135*"inception" + -0.130*"minority" + 0.12
7*"garland"'), (15, '-0.316*"loretta" + 0.206*"movie" + 0.205*"fresh" + 0.177*"steve"
+ -0.168*"tatum" + -0.159*"bullock" + -0.131*"dreams" + 0.114*"dating" + -0.100*"dani
el" + -0.097*"channing"'), (16, '0.220*"dreams" + 0.211*"action" + 0.204*"daughter" +
0.175*"dream" + 0.163*"neeson" + 0.153*"taken" + -0.149*"knight" + -0.149*"loretta" +
0.130*"craig" + 0.127*"films"'), (17, '0.415*"oblivion" + 0.194*"earth" + -0.162*"ann
ihilation" + 0.131*"scifi" + 0.126*"victoria" + 0.122*"cruise" + -0.117*"anderton" +
-0.117*"black" + 0.115*"drones" + 0.112*"kosinski"'), (18, '-0.365*"christine" + -0.2
75*"raimi" + 0.188*"adelaide" + 0.164*"peele" + 0.142*"family" + -0.137*"raimis" + -
0.119*"curse" + -0.117*"lohman" + -0.112*"ganush" + -0.110*"woman"')]
{'19 concepts and 10 words': 0.47459116670353885}
```

Topic Modeling Experiment 4: 2-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 2

In [155…

```
topic_word_list = []

model_2concepts_10words=plot_lsa(2, 10, processed_text = processed_text_method_three)
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '-0.454*"ahmed" + -0.391*"kinley" + -0.276*"covenant" + -0.241*"ri
tchie" + -0.240*"interpreter" + -0.226*"taliban" + -0.177*"gyllenhaal" + -0.140*"ritc
hies" + -0.131*"afghanistan" + -0.115*"afghan"')]
LsiModel(num_terms=10397, num_topics=2, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



In [156…

```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 2,
                       number_of_words = 10)
```

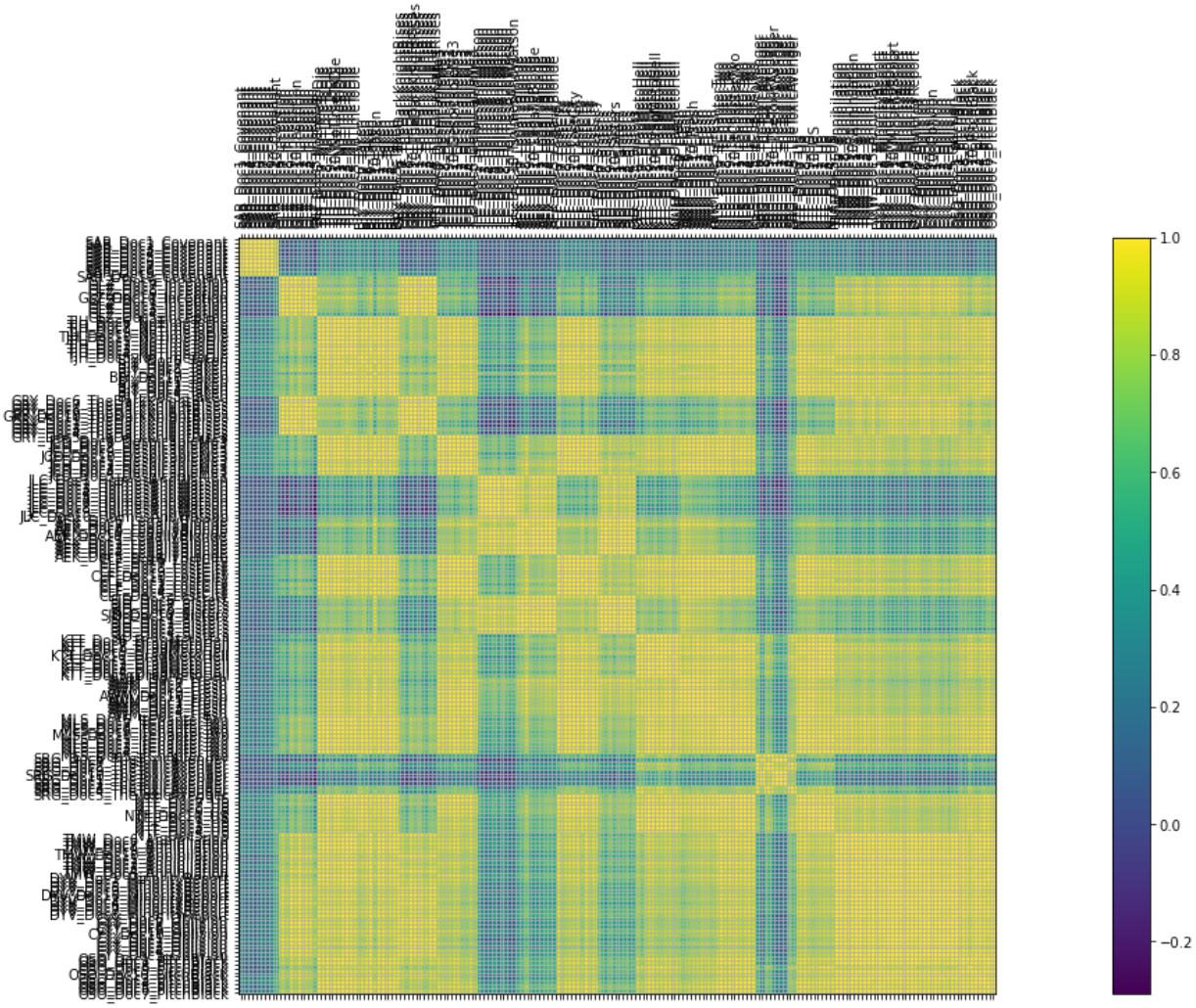
	Topic 0	Topic 1
0	character	ahmed
1	first	kinley
2	story	covenant
3	people	ritchie
4	horror	interpreter
5	action	taliban
6	doesnt	gyllenhaal
7	year	ritchies
8	comedy	afghanistan
9	dream	afghan

Topic Modeling Experiment 5: 4-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 2

In [157…

```
topic_word_list = []

model_2concepts_10words=plot_lsa(4, 10, processed_text = processed_text_method_three)
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '-0.454*"ahmed" + -0.391*"kinley" + -0.276*"covenant" + -0.241*"ri
tchie" + -0.240*"interpreter" + -0.226*"taliban" + -0.177*"gyllenhaal" + -0.140*"ritc
hies" + -0.131*"afghanistan" + -0.115*"afghan"'), (2, '0.347*"dream" + -0.314*"toxic"
+ -0.216*"comedy" + -0.206*"holmes" + -0.197*"avenger" + 0.186*"knight" + 0.152*"nola
n" + 0.143*"nolans" + -0.143*"melvin" + -0.143*"watson"'), (3, '-0.474*"toxic" + 0.31
2*"holmes" + -0.303*"avenger" + -0.227*"melvin" + 0.219*"watson" + -0.166*"horror" +
0.141*"ferrell" + 0.138*"sister" + 0.133*"reilly" + -0.132*"waste"')]
LsiModel(num_terms=10397, num_topics=4, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



In [158…

```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 4,
                       number_of_words = 10)
```

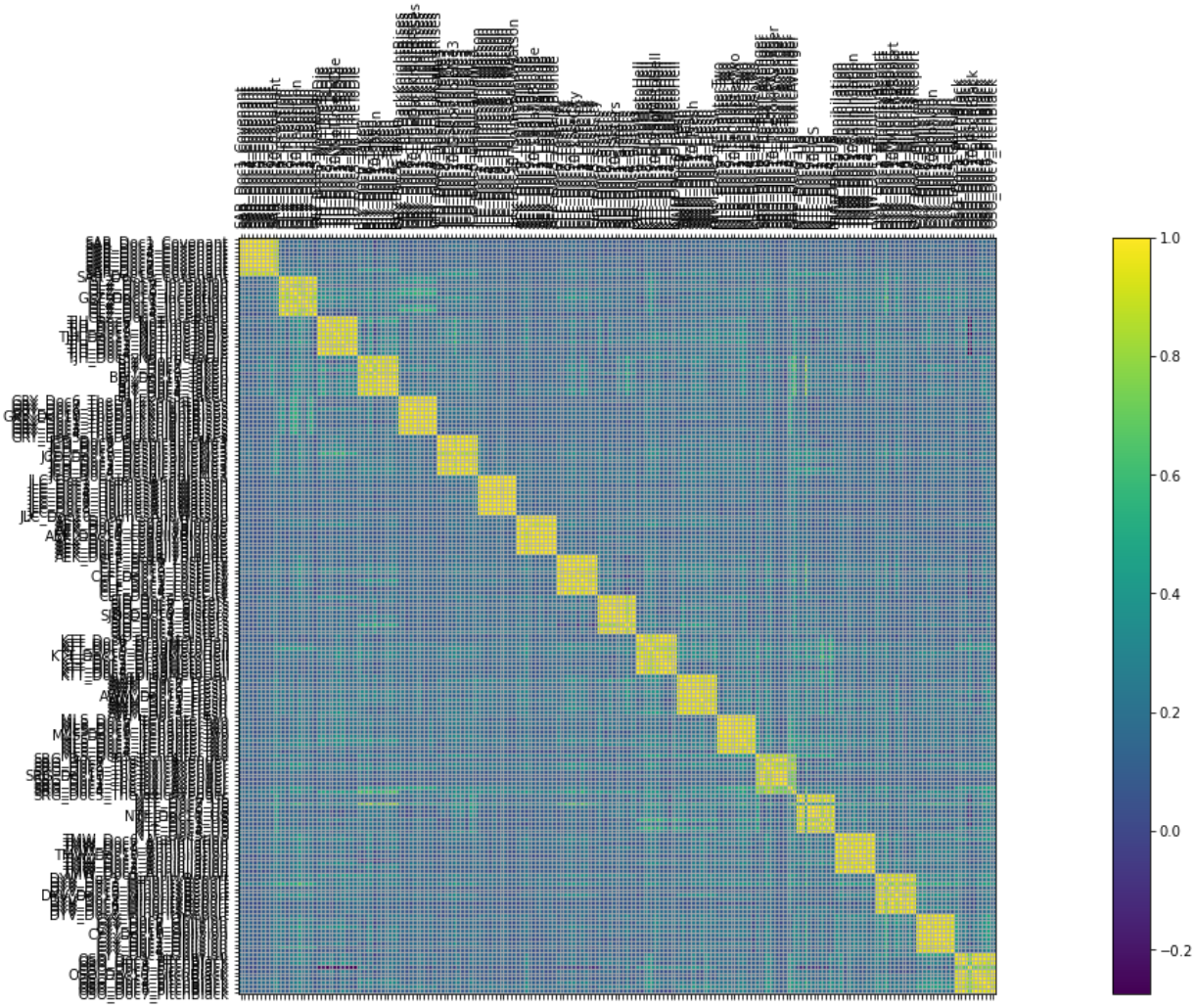
	Topic 0	Topic 1	Topic 2	Topic 3
0	character	ahmed	dream	toxic
1	first	kinley	toxic	holmes
2	story	covenant	comedy	avenger
3	people	ritchie	holmes	melvin
4	horror	interpreter	avenger	watson
5	action	taliban	knight	horror
6	doesnt	gyllenhaal	nolan	ferrell
7	year	ritchies	nolans	sister
8	comedy	afghanistan	melvin	reilly
9	dream	afghan	watson	waste

Topic Modeling Experiment 6: 19-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 2

In [159…

```
topic_word_list = []

model_2concepts_10words=plot_lsa(19, 10, processed_text = processed_text_method_three)
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '-0.454*"ahmed" + -0.391*"kinley" + -0.276*"covenant" + -0.241*"ri
tchie" + -0.240*"interpreter" + -0.226*"taliban" + -0.177*"gyllenhaal" + -0.140*"ritc
hies" + -0.131*"afghanistan" + -0.115*"afghan"'), (2, '0.347*"dream" + -0.314*"toxic"
+ -0.216*"comedy" + -0.206*"holmes" + -0.197*"avenger" + 0.186*"knight" + 0.152*"nola
n" + 0.143*"nolans" + -0.143*"melvin" + -0.143*"watson"'), (3, '0.474*"toxic" + -0.31
2*"holmes" + 0.303*"avenger" + 0.227*"melvin" + -0.219*"watson" + 0.166*"horror" + -
0.141*"ferrell" + -0.138*"sister" + -0.133*"reilly" + 0.132*"waste"'), (4, '0.406*"ho
lmes" + 0.282*"watson" + 0.209*"dream" + 0.180*"ferrell" + 0.173*"reilly" + 0.172*"to
xic" + 0.158*"knight" + -0.147*"blonde" + -0.128*"sister" + 0.117*"nolan"'), (5, '-0.
351*"blonde" + -0.260*"legally" + -0.245*"dream" + -0.190*"school" + -0.182*"withersp
oon" + -0.171*"warner" + -0.164*"harvard" + 0.161*"horror" + 0.124*"holmes" + -0.118
*"knight"'), (6, '-0.411*"sister" + -0.285*"poehler" + -0.239*"party" + 0.209*"blond
e" + -0.176*"maura" + 0.151*"legally" + 0.131*"first" + -0.120*"parent" + 0.107*"with
erspoon" + 0.106*"warner"'), (7, '0.231*"despicable" + 0.178*"minion" + 0.175*"knigh
t" + -0.165*"sister" + -0.152*"oblivion" + -0.148*"pitch" + 0.142*"family" + -0.141
*"cruise" + 0.137*"first" + 0.137*"horror"'), (8, '0.359*"horror" + -0.296*"despicabl
e" + -0.231*"minion" + 0.136*"black" + 0.123*"peele" + 0.122*"adelaide" + -0.115*"bra
tt" + -0.114*"brother" + -0.106*"loretta" + 0.106*"christine"'), (9, '0.512*"dream" +
-0.312*"knight" + -0.184*"batman" + -0.179*"rise" + -0.172*"wayne" + -0.170*"gotham"
+ 0.141*"inception" + 0.125*"within" + -0.123*"bruce" + -0.082*"selina"'), (10, '-0.3
23*"chapter" + -0.203*"derry" + -0.191*"pennywise" + -0.184*"character" + 0.180*"fami
ly" + -0.168*"loser" + -0.147*"james" + 0.134*"fresh" + 0.128*"steve" + -0.125*"clow
n"'), (11, '-0.271*"black" + -0.254*"pitch" + -0.187*"character" + -0.178*"creature"
+ 0.168*"anderton" + -0.163*"alien" + -0.156*"planet" + -0.154*"despicable" + 0.146
*"chapter" + 0.142*"future"'), (12, '-0.331*"loretta" + -0.212*"bullock" + -0.201*"ta
tum" + -0.159*"annihilation" + 0.133*"despicable" + -0.127*"husband" + 0.119*"pitch"
+ -0.117*"fresh" + 0.110*"black" + -0.109*"shimmer"'), (13, '-0.331*"annihilation" +
-0.236*"garland" + -0.222*"shimmer" + 0.178*"loretta" + 0.137*"fresh" + 0.127*"andert
on" + 0.117*"bullock" + -0.114*"jason" + 0.109*"tatum" + 0.097*"minority"'), (14, '-
0.296*"fresh" + -0.246*"steve" + 0.189*"family" + -0.184*"woman" + 0.178*"adelaide" +
0.164*"loretta" + -0.151*"dating" + 0.142*"peele" + 0.134*"action" + -0.116*"edgarjon
es"'), (15, '0.327*"daughter" + 0.309*"action" + 0.228*"neeson" + 0.199*"taken" + 0.1
85*"paris" + -0.165*"loretta" + -0.151*"oblivion" + 0.115*"woman" + -0.109*"bullock"
+ -0.102*"sister"'), (16, '0.374*"oblivion" + -0.200*"annihilation" + 0.176*"earth" +
0.153*"victoria" + -0.143*"garland" + 0.139*"cruise" + -0.138*"anderton" + -0.133*"sh
immer" + 0.125*"daughter" + 0.118*"action"'), (17, '0.383*"christine" + 0.289*"raimi"
+ 0.145*"raimis" + -0.138*"fresh" + 0.136*"curse" + -0.134*"adelaide" + 0.123*"lohma
n" + 0.120*"woman" + 0.116*"ganush" + -0.115*"family"'), (18, '0.303*"craig" + 0.197
*"madeleine" + 0.187*"character" + 0.168*"bond" + 0.167*"craigs" + 0.166*"spectre" +
-0.165*"daughter" + 0.163*"series" + -0.151*"creature" + 0.146*"casino"')]
LsiModel(num_terms=10397, num_topics=19, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



In [160…

```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 19,
                       number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	
0	character	ahmed	dream	toxic	holmes	blonde	sister	despicable	horror	
1	first	kinley	toxic	holmes	watson	legally	poehler	minion	despicable	
2	story	covenant	comedy	avenger	dream	dream	party	knight	minion	
3	people	ritchie	holmes	melvin	ferrell	school	blonde	sister	black	
4	horror	interpreter	avenger	watson	reilly	witherspoon	maura	oblivion	peele	
5	action	taliban	knight	horror	toxic	warner	legally	pitch	adelaide	
6	doesnt	gyllenhaal	nolan	ferrell	knight	harvard	first	family	bratt	ir
7	year	ritchies	nolans	sister	blonde	horror	parent	cruise	brother	
8	comedy	afghanistan	melvin	reilly	sister	holmes	witherspoon	first	loretta	
9	dream	afghan	watson	waste	nolan	knight	warner	horror	christine	

In [161…

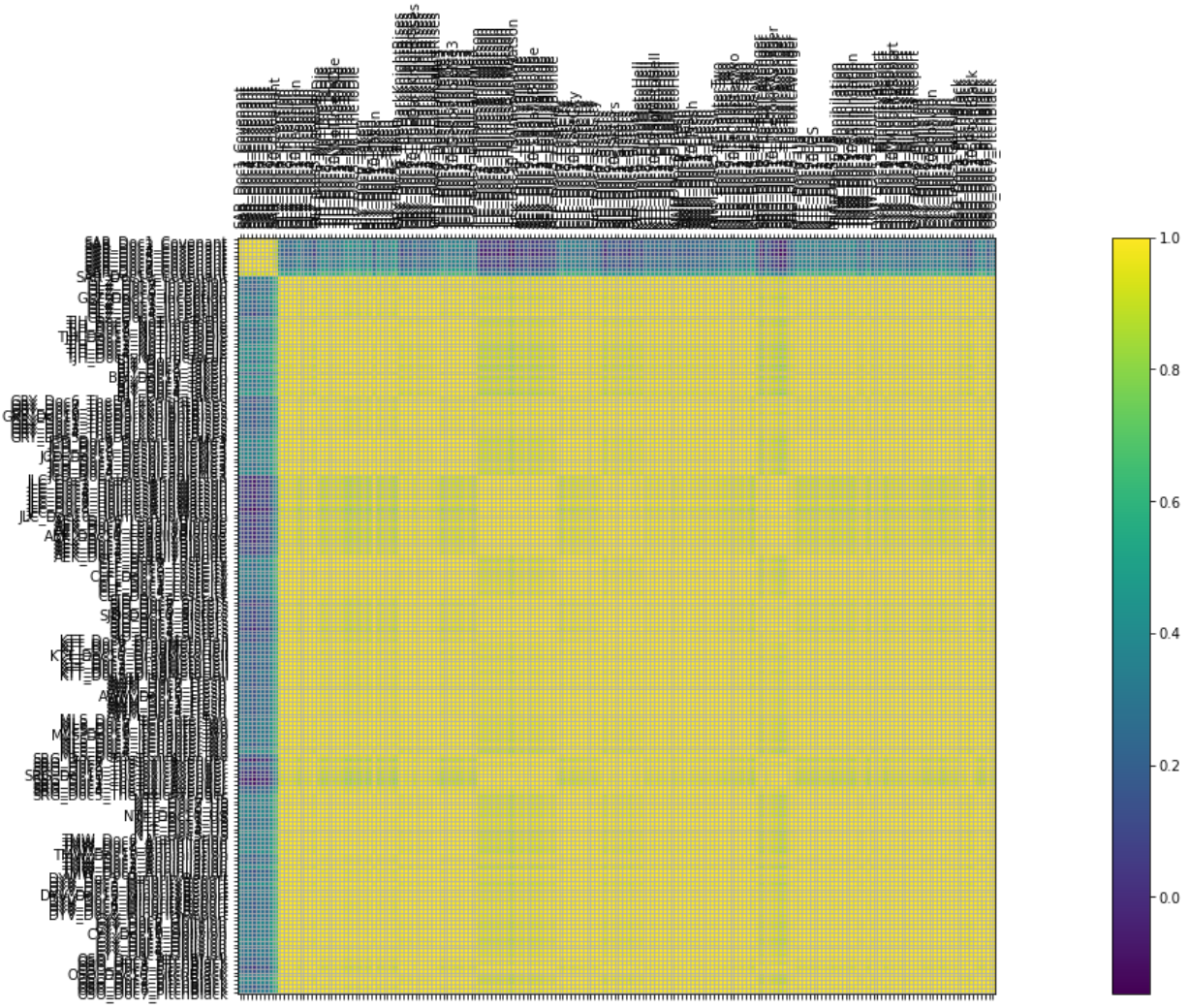
```
topics = [2, 4, 19]
coherence_values = []
index_number = 0

for t in topics:
    lsa_coherence_function(processed_text = processed_text_method_three,
                           topics = t,
                           words = 10)

    coherence ={f'{topics[index_number]} concepts and {words} words':coherence_values[

    print(coherence)

    index_number += 1
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"')]
{'2 concepts and 10 words': 0.716046318264559}
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '-0.454*"ahmed" + -0.391*"kinley" + -0.276*"covenant" + -0.241*"ri
tchie" + -0.240*"interpreter" + -0.226*"taliban" + -0.177*"gyllenhaal" + -0.140*"ritc
hies" + -0.131*"afghanistan" + -0.115*"afghan"'), (2, '-0.347*"dream" + 0.314*"toxic"
+ 0.216*"comedy" + 0.206*"holmes" + 0.197*"avenger" + -0.186*"knight" + -0.152*"nola
n" + -0.143*"nolans" + 0.143*"melvin" + 0.143*"watson"'), (3, '0.474*"toxic" + -0.312
*"holmes" + 0.303*"avenger" + 0.227*"melvin" + -0.219*"watson" + 0.166*"horror" + -0.
141*"ferrell" + -0.138*"sister" + -0.133*"reilly" + 0.132*"waste"')]
{'4 concepts and 10 words': 0.42062880346860215}
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '-0.454*"ahmed" + -0.391*"kinley" + -0.276*"covenant" + -0.241*"ri
tchie" + -0.240*"interpreter" + -0.226*"taliban" + -0.177*"gyllenhaal" + -0.140*"ritc
hies" + -0.131*"afghanistan" + -0.115*"afghan"'), (2, '0.347*"dream" + -0.314*"toxic"
+ -0.216*"comedy" + -0.206*"holmes" + -0.197*"avenger" + 0.186*"knight" + 0.152*"nola
n" + 0.143*"nolans" + -0.143*"melvin" + -0.143*"watson"'), (3, '-0.474*"toxic" + 0.31
2*"holmes" + -0.303*"avenger" + -0.227*"melvin" + 0.219*"watson" + -0.166*"horror" +
0.141*"ferrell" + 0.138*"sister" + 0.133*"reilly" + -0.132*"waste"'), (4, '-0.406*"ho
lmes" + -0.282*"watson" + -0.209*"dream" + -0.180*"ferrell" + -0.173*"reilly" + -0.17
2*"toxic" + -0.158*"knight" + 0.147*"blonde" + 0.128*"sister" + -0.117*"nolan"'), (5,
'-0.351*"blonde" + -0.260*"legally" + -0.245*"dream" + -0.190*"school" + -0.182*"with
erspoon" + -0.171*"warner" + -0.164*"harvard" + 0.161*"horror" + 0.124*"holmes" + -0.
118*"knight"'), (6, '0.411*"sister" + 0.285*"poehler" + 0.239*"party" + -0.209*"blond
e" + 0.176*"maura" + -0.151*"legally" + -0.131*"first" + 0.120*"parent" + -0.107*"wit
herspoon" + -0.106*"warner"'), (7, '0.231*"despicable" + 0.178*"minion" + 0.175*"knig
ht" + -0.165*"sister" + -0.152*"oblivion" + -0.148*"pitch" + 0.142*"family" + -0.141
*"cruise" + 0.137*"first" + 0.137*"horror"'), (8, '-0.359*"horror" + 0.296*"despicabl
e" + 0.231*"minion" + -0.136*"black" + -0.123*"peele" + -0.122*"adelaide" + 0.115*"br
att" + 0.114*"brother" + 0.106*"loretta" + -0.106*"christine"'), (9, '0.512*"dream" +
-0.312*"knight" + -0.184*"batman" + -0.179*"rise" + -0.172*"wayne" + -0.170*"gotham"
+ 0.141*"inception" + 0.125*"within" + -0.123*"bruce" + -0.082*"selina"'), (10, '-0.3
23*"chapter" + -0.203*"derry" + -0.191*"pennywise" + -0.184*"character" + 0.180*"fami
ly" + -0.168*"loser" + -0.147*"james" + 0.134*"fresh" + 0.128*"steve" + -0.125*"clow
n"'), (11, '0.271*"black" + 0.254*"pitch" + 0.187*"character" + 0.178*"creature" + -
0.168*"anderton" + 0.163*"alien" + 0.156*"planet" + 0.154*"despicable" + -0.146*"chap
ter" + -0.142*"future"'), (12, '0.331*"loretta" + 0.212*"bullock" + 0.201*"tatum" +
0.159*"annihilation" + -0.133*"despicable" + 0.127*"husband" + -0.119*"pitch" + 0.117
*"fresh" + -0.110*"black" + 0.109*"shimmer"'), (13, '-0.331*"annihilation" + -0.236
*"garland" + -0.222*"shimmer" + 0.178*"loretta" + 0.137*"fresh" + 0.127*"anderton" +
0.117*"bullock" + -0.114*"jason" + 0.109*"tatum" + 0.097*"minority"'), (14, '0.296*"f
resh" + 0.246*"steve" + -0.189*"family" + 0.184*"woman" + -0.178*"adelaide" + -0.164
*"loretta" + 0.151*"dating" + -0.142*"peele" + -0.134*"action" + 0.116*"edgarjone
s"'), (15, '-0.327*"daughter" + -0.309*"action" + -0.228*"neeson" + -0.199*"taken" +
-0.185*"paris" + 0.165*"loretta" + 0.151*"oblivion" + -0.115*"woman" + 0.109*"bulloc
k" + 0.102*"sister"'), (16, '0.374*"oblivion" + -0.200*"annihilation" + 0.176*"earth"
+ 0.153*"victoria" + -0.143*"garland" + 0.139*"cruise" + -0.138*"anderton" + -0.133
*"shimmer" + 0.125*"daughter" + 0.118*"action"'), (17, '0.383*"christine" + 0.289*"ra
imi" + 0.145*"raimis" + -0.138*"fresh" + 0.136*"curse" + -0.134*"adelaide" + 0.123*"l
ohman" + 0.120*"woman" + 0.116*"ganush" + -0.115*"family"'), (18, '-0.303*"craig" + -
0.197*"madeleine" + -0.187*"character" + -0.168*"bond" + -0.167*"craigs" + -0.166*"sp
ectre" + 0.165*"daughter" + -0.163*"series" + 0.151*"creature" + -0.146*"casino"')]
{'19 concepts and 10 words': 0.4261907680736007}
```

Topic Modeling Experiment 7: 2-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 3

In [54]:
```
topic_word_list = []

model_2concepts_10words=plot_lsa(2, 10, processed_text = processed_text_method_four)
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"')]
LsiModel(num_terms=10397, num_topics=2, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



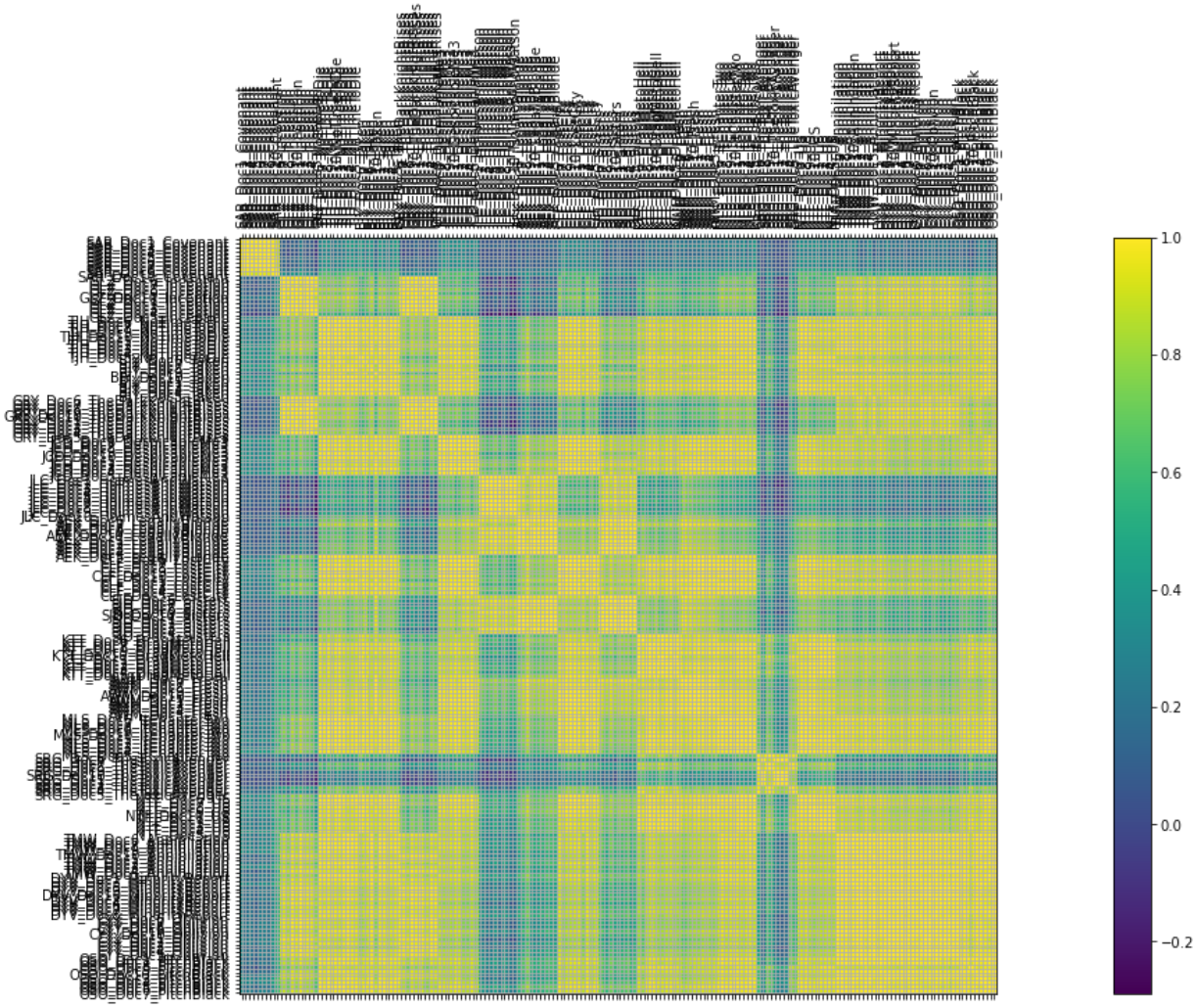
In [55]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 2,
                       number_of_words = 10)
```

	Topic 0	Topic 1
0	character	ahmed
1	first	kinley
2	story	covenant
3	people	ritchie
4	horror	interpreter
5	action	taliban
6	doesnt	gyllenhaal
7	year	ritchies
8	comedy	afghanistan
9	dream	afghan

Topic Modeling Experiment 8: 4-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 3

In [56]:
```
topic_word_list = []

model_4concepts_10words=plot_lsa(4, 10, processed_text = processed_text_method_four)
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"'), (2, '0.347*"dream" + -0.314*"toxic" + -0.216
*"comedy" + -0.206*"holmes" + -0.197*"avenger" + 0.186*"knight" + 0.152*"nolan" + 0.1
43*"nolans" + -0.143*"melvin" + -0.143*"watson"'), (3, '-0.474*"toxic" + 0.312*"holme
s" + -0.303*"avenger" + -0.227*"melvin" + 0.219*"watson" + -0.166*"horror" + 0.141*"f
errell" + 0.138*"sister" + 0.133*"reilly" + -0.132*"waste"')]
LsiModel(num_terms=10397, num_topics=4, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



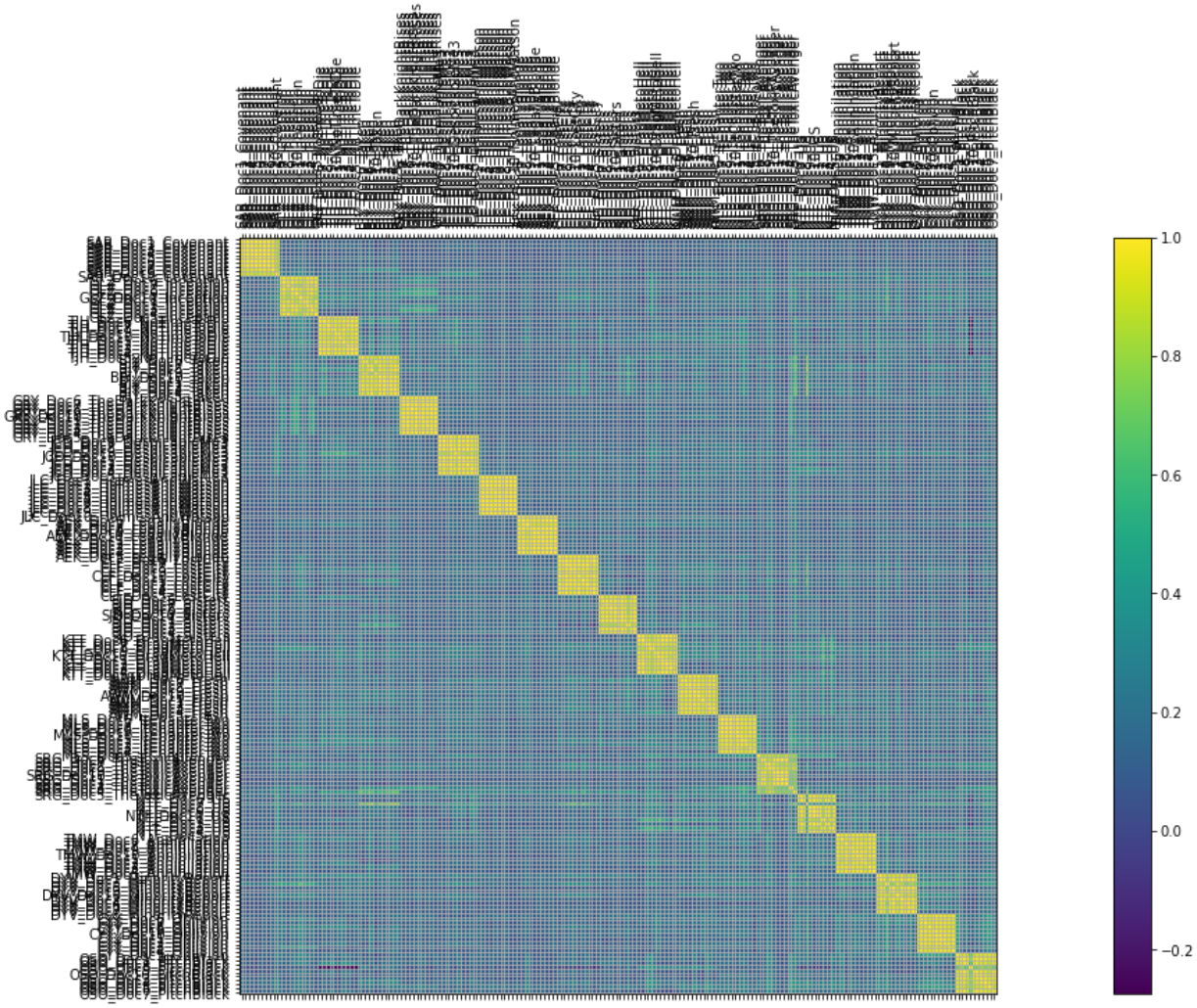
In [57]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 4,
                       number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3
0	character	ahmed	dream	toxic
1	first	kinley	toxic	holmes
2	story	covenant	comedy	avenger
3	people	ritchie	holmes	melvin
4	horror	interpreter	avenger	watson
5	action	taliban	knight	horror
6	doesnt	gyllenhaal	nolan	ferrell
7	year	ritchies	nolans	sister
8	comedy	afghanistan	melvin	reilly
9	dream	afghan	watson	waste

Topic Modeling Experiment 9: 19-Topic Latent Sentiment Analysis with Data Wrangling and Vectorization Method 3

In [58]:
```
topic_word_list = []

model_19concepts_10words=plot_lsa(19, 10, processed_text = processed_text_method_four)
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"'), (2, '0.347*"dream" + -0.314*"toxic" + -0.216
*"comedy" + -0.206*"holmes" + -0.197*"avenger" + 0.186*"knight" + 0.152*"nolan" + 0.1
43*"nolans" + -0.143*"melvin" + -0.143*"watson"'), (3, '-0.474*"toxic" + 0.312*"holme
s" + -0.303*"avenger" + -0.227*"melvin" + 0.219*"watson" + -0.166*"horror" + 0.141*"f
errell" + 0.138*"sister" + 0.133*"reilly" + -0.132*"waste"'), (4, '0.406*"holmes" +
0.282*"watson" + 0.209*"dream" + 0.180*"ferrell" + 0.173*"reilly" + 0.172*"toxic" +
0.158*"knight" + -0.147*"blonde" + -0.128*"sister" + 0.117*"nolan"'), (5, '0.351*"blo
nde" + 0.260*"legally" + 0.245*"dream" + 0.190*"school" + 0.182*"witherspoon" + 0.171
*"warner" + 0.164*"harvard" + -0.161*"horror" + -0.124*"holmes" + 0.118*"knight"'),
(6, '0.411*"sister" + 0.285*"poehler" + 0.239*"party" + -0.209*"blonde" + 0.176*"maur
a" + -0.151*"legally" + -0.131*"first" + 0.120*"parent" + -0.107*"witherspoon" + -0.1
06*"warner"'), (7, '-0.231*"despicable" + -0.178*"minion" + -0.175*"knight" + 0.165
*"sister" + 0.152*"oblivion" + 0.148*"pitch" + -0.142*"family" + 0.141*"cruise" + -0.
137*"first" + -0.137*"horror"'), (8, '0.359*"horror" + -0.296*"despicable" + -0.231
*"minion" + 0.136*"black" + 0.123*"peele" + 0.122*"adelaide" + -0.115*"bratt" + -0.11
4*"brother" + -0.106*"loretta" + 0.106*"christine"'), (9, '0.512*"dream" + -0.312*"kn
ight" + -0.184*"batman" + -0.179*"rise" + -0.172*"wayne" + -0.170*"gotham" + 0.141*"i
nception" + 0.125*"within" + -0.123*"bruce" + -0.082*"selina"'), (10, '0.323*"chapte
r" + 0.203*"derry" + 0.191*"pennywise" + 0.184*"character" + -0.180*"family" + 0.168
*"loser" + 0.147*"james" + -0.134*"fresh" + -0.128*"steve" + 0.125*"clown"'), (11,
'0.271*"black" + 0.254*"pitch" + 0.187*"character" + 0.178*"creature" + -0.168*"ander
ton" + 0.163*"alien" + 0.156*"planet" + 0.154*"despicable" + -0.146*"chapter" + -0.14
2*"future"'), (12, '-0.331*"loretta" + -0.212*"bullock" + -0.201*"tatum" + -0.159*"an
nihilation" + 0.133*"despicable" + -0.127*"husband" + 0.119*"pitch" + -0.117*"fresh"
+ 0.110*"black" + -0.109*"shimmer"'), (13, '0.331*"annihilation" + 0.236*"garland" +
0.222*"shimmer" + -0.178*"loretta" + -0.137*"fresh" + -0.127*"anderton" + -0.117*"bul
lock" + 0.114*"jason" + -0.109*"tatum" + -0.097*"minority"'), (14, '-0.296*"fresh" +
-0.246*"steve" + 0.189*"family" + -0.184*"woman" + 0.178*"adelaide" + 0.164*"loretta"
+ -0.151*"dating" + 0.142*"peele" + 0.134*"action" + -0.116*"edgarjones"'), (15, '0.3
27*"daughter" + 0.309*"action" + 0.228*"neeson" + 0.199*"taken" + 0.185*"paris" + -0.
165*"loretta" + -0.151*"oblivion" + 0.115*"woman" + -0.109*"bullock" + -0.102*"siste
r"'), (16, '-0.374*"oblivion" + 0.200*"annihilation" + -0.176*"earth" + -0.153*"victo
ria" + 0.143*"garland" + -0.139*"cruise" + 0.138*"anderton" + 0.133*"shimmer" + -0.12
5*"daughter" + -0.118*"action"'), (17, '0.383*"christine" + 0.289*"raimi" + 0.145*"ra
imis" + -0.138*"fresh" + 0.136*"curse" + -0.134*"adelaide" + 0.123*"lohman" + 0.120
*"woman" + 0.116*"ganush" + -0.115*"family"'), (18, '-0.303*"craig" + -0.197*"madelei
ne" + -0.187*"character" + -0.168*"bond" + -0.167*"craigs" + -0.166*"spectre" + 0.165
*"daughter" + -0.163*"series" + 0.151*"creature" + -0.146*"casino"')]
LsiModel(num_terms=10397, num_topics=19, decay=1.0, chunksize=20000)
<class 'gensim.models.lsimodel.LsiModel'>
```



In [59]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 19,
                       number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	
0	character	ahmed	dream	toxic	holmes	blonde	sister	despicable	horror	
1	first	kinley	toxic	holmes	watson	legally	poehler	minion	despicable	
2	story	covenant	comedy	avenger	dream	dream	party	knight	minion	
3	people	ritchie	holmes	melvin	ferrell	school	blonde	sister	black	
4	horror	interpreter	avenger	watson	reilly	witherspoon	maura	oblivion	peele	
5	action	taliban	knight	horror	toxic	warner	legally	pitch	adelaide	
6	doesnt	gyllenhaal	nolan	ferrell	knight	harvard	first	family	bratt	ir
7	year	ritchies	nolans	sister	blonde	horror	parent	cruise	brother	
8	comedy	afghanistan	melvin	reilly	sister	holmes	witherspoon	first	loretta	
9	dream	afghan	watson	waste	nolan	knight	warner	horror	christine	

In [60]:
```
topics = [2, 4, 19]
coherence_values = []
index_number = 0

for t in topics:
    lsa_coherence_function(processed_text = processed_text_method_four,
                           topics = t,
                           words = 10)

    coherence ={f'{topics[index_number]} concepts and {words} words':coherence_values[

    print(coherence)

    index_number += 1
```

```
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"')]
{'2 concepts and 10 words': 0.716046318264559}
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"'), (2, '0.347*"dream" + -0.314*"toxic" + -0.216
*"comedy" + -0.206*"holmes" + -0.197*"avenger" + 0.186*"knight" + 0.152*"nolan" + 0.1
43*"nolans" + -0.143*"melvin" + -0.143*"watson"'), (3, '0.474*"toxic" + -0.312*"holme
s" + 0.303*"avenger" + 0.227*"melvin" + -0.219*"watson" + 0.166*"horror" + -0.141*"fe
rrell" + -0.138*"sister" + -0.133*"reilly" + 0.132*"waste"')]
{'4 concepts and 10 words': 0.5780408293494885}
[(0, '-0.237*"character" + -0.181*"first" + -0.154*"story" + -0.150*"people" + -0.149
*"horror" + -0.126*"action" + -0.108*"doesnt" + -0.106*"year" + -0.106*"comedy" + -0.
103*"dream"'), (1, '0.454*"ahmed" + 0.391*"kinley" + 0.276*"covenant" + 0.241*"ritchi
e" + 0.240*"interpreter" + 0.226*"taliban" + 0.177*"gyllenhaal" + 0.140*"ritchies" +
0.131*"afghanistan" + 0.115*"afghan"'), (2, '-0.347*"dream" + 0.314*"toxic" + 0.216
*"comedy" + 0.206*"holmes" + 0.197*"avenger" + -0.186*"knight" + -0.152*"nolan" + -0.
143*"nolans" + 0.143*"melvin" + 0.143*"watson"'), (3, '0.474*"toxic" + -0.312*"holme
s" + 0.303*"avenger" + 0.227*"melvin" + -0.219*"watson" + 0.166*"horror" + -0.141*"fe
rrell" + -0.138*"sister" + -0.133*"reilly" + 0.132*"waste"'), (4, '-0.406*"holmes" +
-0.282*"watson" + -0.209*"dream" + -0.180*"ferrell" + -0.173*"reilly" + -0.172*"toxi
c" + -0.158*"knight" + 0.147*"blonde" + 0.128*"sister" + -0.117*"nolan"'), (5, '0.351
*"blonde" + 0.260*"legally" + 0.245*"dream" + 0.190*"school" + 0.182*"witherspoon" +
0.171*"warner" + 0.164*"harvard" + -0.161*"horror" + -0.124*"holmes" + 0.118*"knigh
t"'), (6, '0.411*"sister" + 0.285*"poehler" + 0.239*"party" + -0.209*"blonde" + 0.176
*"maura" + -0.151*"legally" + -0.131*"first" + 0.120*"parent" + -0.107*"witherspoon"
+ -0.106*"warner"'), (7, '0.231*"despicable" + 0.178*"minion" + 0.175*"knight" + -0.1
65*"sister" + -0.152*"oblivion" + -0.148*"pitch" + 0.142*"family" + -0.141*"cruise" +
0.137*"first" + 0.137*"horror"'), (8, '-0.359*"horror" + 0.296*"despicable" + 0.231
*"minion" + -0.136*"black" + -0.123*"peele" + -0.122*"adelaide" + 0.115*"bratt" + 0.1
14*"brother" + 0.106*"loretta" + -0.106*"christine"'), (9, '0.512*"dream" + -0.312*"k
night" + -0.184*"batman" + -0.179*"rise" + -0.172*"wayne" + -0.170*"gotham" + 0.141
*"inception" + 0.125*"within" + -0.123*"bruce" + -0.082*"selina"'), (10, '0.323*"chap
ter" + 0.203*"derry" + 0.191*"pennywise" + 0.184*"character" + -0.180*"family" + 0.16
8*"loser" + 0.147*"james" + -0.134*"fresh" + -0.128*"steve" + 0.125*"clown"'), (11,
'-0.271*"black" + -0.254*"pitch" + -0.187*"character" + -0.178*"creature" + 0.168*"an
derton" + -0.163*"alien" + -0.156*"planet" + -0.154*"despicable" + 0.146*"chapter" +
0.142*"future"'), (12, '-0.331*"loretta" + -0.212*"bullock" + -0.201*"tatum" + -0.159
*"annihilation" + 0.133*"despicable" + -0.127*"husband" + 0.119*"pitch" + -0.117*"fre
sh" + 0.110*"black" + -0.109*"shimmer"'), (13, '0.331*"annihilation" + 0.236*"garlan
d" + 0.222*"shimmer" + -0.178*"loretta" + -0.137*"fresh" + -0.127*"anderton" + -0.117
*"bullock" + 0.114*"jason" + -0.109*"tatum" + -0.097*"minority"'), (14, '0.296*"fres
h" + 0.246*"steve" + -0.189*"family" + 0.184*"woman" + -0.178*"adelaide" + -0.164*"lo
retta" + 0.151*"dating" + -0.142*"peele" + -0.134*"action" + 0.116*"edgarjones"'), (1
5, '0.327*"daughter" + 0.309*"action" + 0.228*"neeson" + 0.199*"taken" + 0.185*"pari
s" + -0.165*"loretta" + -0.151*"oblivion" + 0.115*"woman" + -0.109*"bullock" + -0.102
*"sister"'), (16, '-0.374*"oblivion" + 0.200*"annihilation" + -0.176*"earth" + -0.153
*"victoria" + 0.143*"garland" + -0.139*"cruise" + 0.138*"anderton" + 0.133*"shimmer"
+ -0.125*"daughter" + -0.118*"action"'), (17, '0.383*"christine" + 0.289*"raimi" + 0.
145*"raimis" + -0.138*"fresh" + 0.136*"curse" + -0.134*"adelaide" + 0.123*"lohman" +
0.120*"woman" + 0.116*"ganush" + -0.115*"family"'), (18, '-0.303*"craig" + -0.197*"ma
deleine" + -0.187*"character" + -0.168*"bond" + -0.167*"craigs" + -0.166*"spectre" +
0.165*"daughter" + -0.163*"series" + 0.151*"creature" + -0.146*"casino"')]
{'19 concepts and 10 words': 0.4846473496577823}
```

4.2) Latent Dirichlet Allocation Experiments

Let's implement latent dirichlet allocation models to determine their effectiveness at performing topic modeling on our corpus of movie reviews.

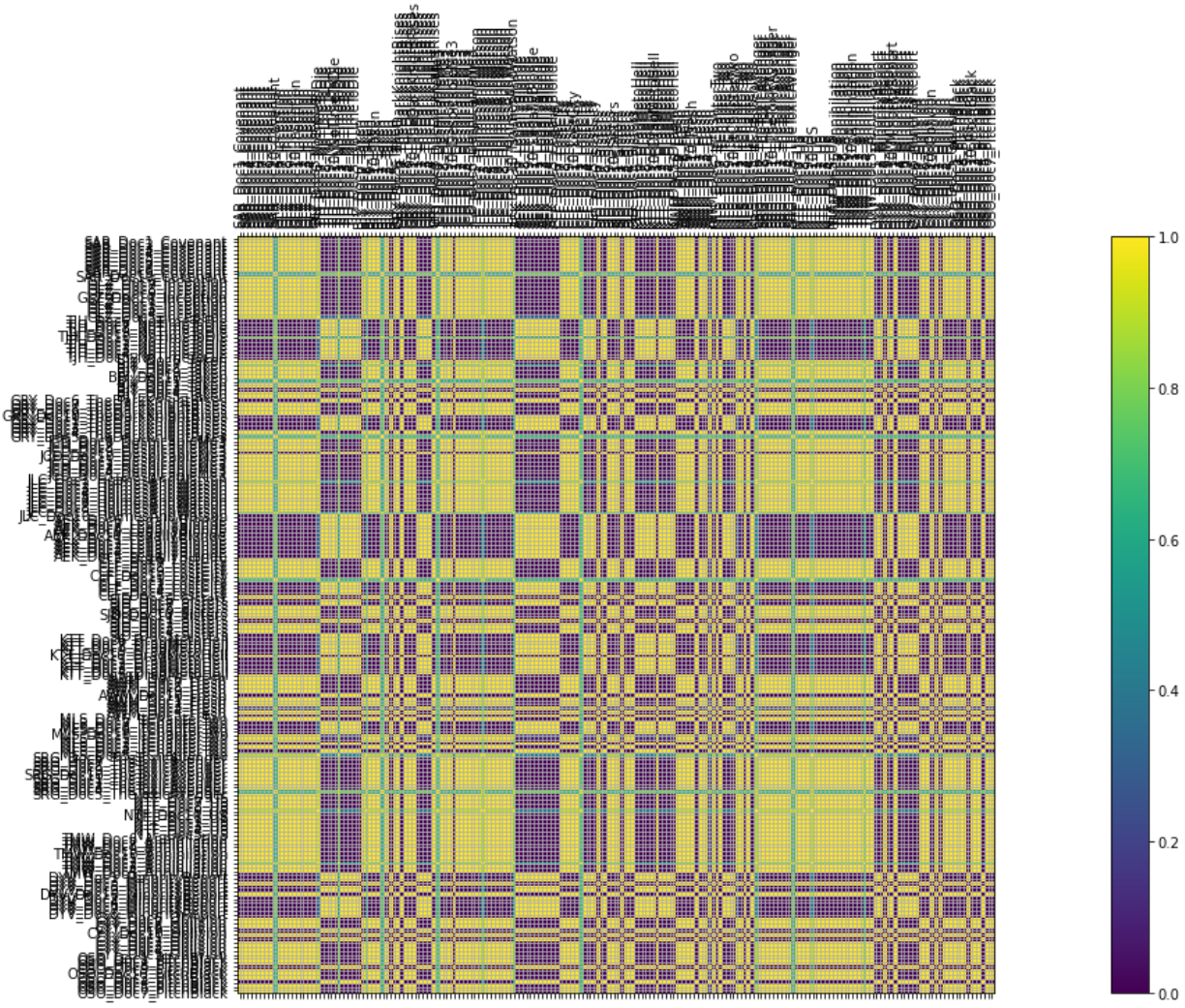
Topic Modeling Experiment 10: 2-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 1

In [61]:
```
topic_word_list = []

def create_gensim_lda_model(doc_clean, number_of_topics, words):

    # Creating the term dictionary of our courpus, where every unique term is assigned
    dictionary = corpora.Dictionary(doc_clean)
    # Converting list of documents (corpus) into Document Term Matrix using dictionary
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
    # generate LDA model
    ldamodel = LdaModel(doc_term_matrix
                        ,num_topics=number_of_topics
                        ,id2word = dictionary
                        ,alpha='auto'
                        ,eta='auto'
                        ,iterations=100
                        ,random_state=23
                        ,passes=20)
    # train model
    print(ldamodel.print_topics(num_topics=number_of_topics, num_words=words))
    topic_word_tuples = ldamodel.print_topics(num_topics=number_of_topics, num_words=w
    for i in topic_word_tuples:
        topic_word_list.append(i)
    index = similarities.MatrixSimilarity(ldamodel[doc_term_matrix])
    return ldamodel, dictionary, index, doc_term_matrix

#def lda(tfidf_matrix, terms, topics = 3, num_words = 10):
    #this is a function to perform lda on the tfidf matrix.  function varibales includ
    #tfidf matrix, desired number of topic, and number of words per topic.

#    topics = 3
#    num_words = 10
#    lda = LatentDirichletAllocation(n_components=topics).fit(tfidf_matrix)

#    topic_dict = {}
#    for topic_num, topic in enumerate(lda.components_):
#        topic_dict[topic_num] = " ".join([terms[i]for i in topic.argsort()[:-num_word

#    print(topic_dict)
```

In [62]:
```
def run_lda(processed_text, number_of_topics, words):
    model2, dictionary2, index2, doctermmatrix2 = create_gensim_lda_model(processed_te

    for doc in processed_text:
        vec_bow2 = dictionary2.doc2bow(doc)
        vec2 = model2[vec_bow2]  # convert the query to embedded space
        sims2 = index2[vec2]  # perform a similarity query against the corpus
        #print(list(enumerate(sims2)))

    fig, ax = plt.subplots(figsize=(30, 10))
```

```
    cax = ax.matshow(index2, interpolation='nearest')
    ax.grid(True)
    plt.xticks(range(len(processed_text)), titles, rotation=90);
    plt.yticks(range(len(processed_text)), titles);
    fig.colorbar(cax)
    plt.show()

run_lda(processed_text = processed_text_method_one,
       number_of_topics = 2,
       words = 10)
```

```
[(0, '0.003*"movie" + 0.003*"films" + 0.002*"first" + 0.002*"blonde" + 0.002*"people"
+ 0.002*"years" + 0.002*"would" + 0.002*"horror" + 0.002*"character" + 0.002*"legall
y"'), (1, '0.008*"movie" + 0.003*"first" + 0.003*"films" + 0.003*"story" + 0.003*"mov
ies" + 0.002*"would" + 0.002*"horror" + 0.002*"action" + 0.002*"characters" + 0.002
*"people"')]
```



In [63]:

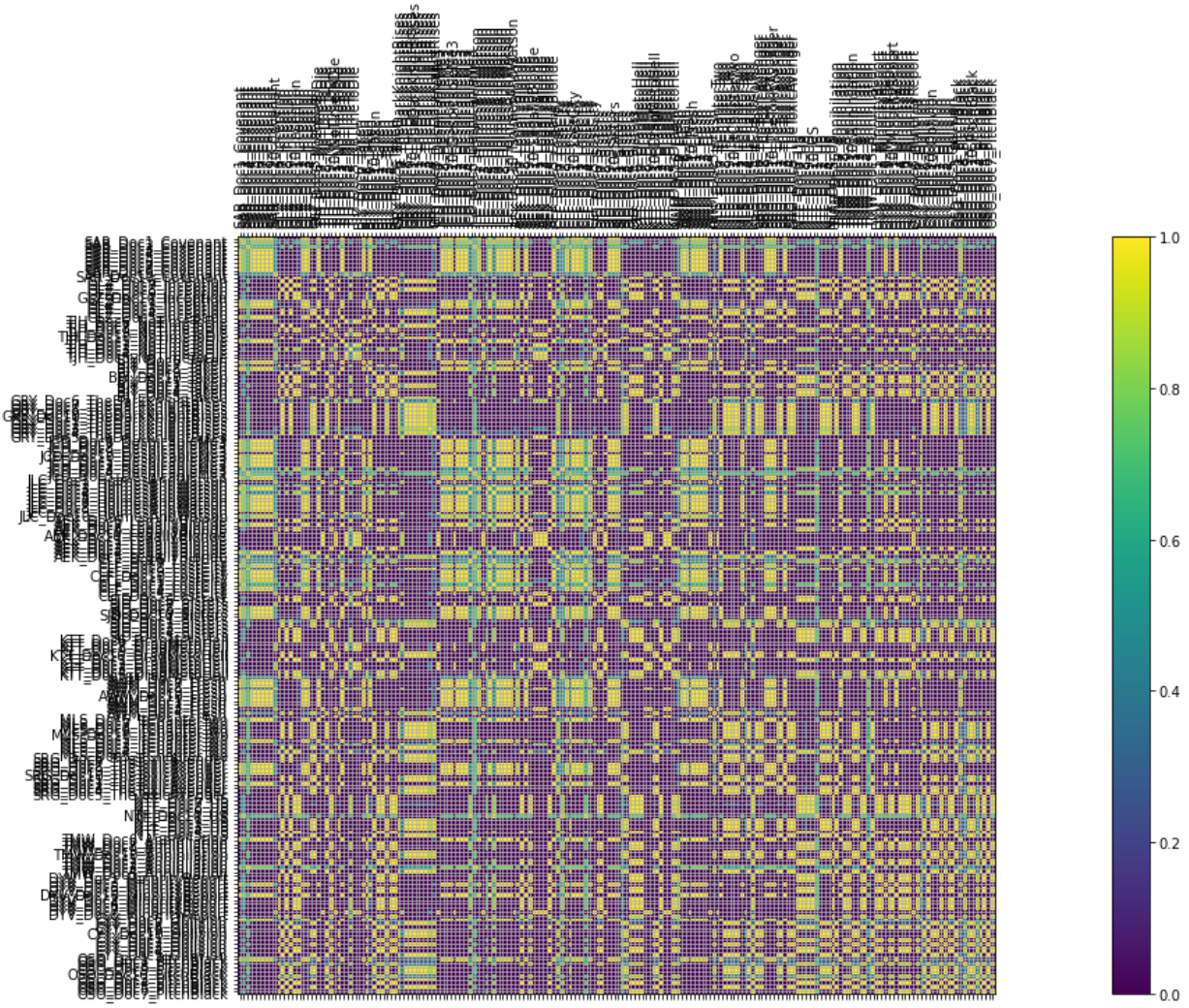
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 2, number_of_words = 10)
```

	Topic 0	Topic 1
0	movie	movie
1	films	first
2	first	films
3	blonde	story
4	people	movies
5	years	would
6	would	horror
7	horror	action
8	character	characters
9	legally	people

Topic Modeling Experiment 11: 4-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 1

In [64]:
```
run_lda(processed_text = processed_text_method_one,
       number_of_topics = 4,
       words = 10)
```

```
[(0, '0.003*"films" + 0.003*"blonde" + 0.002*"first" + 0.002*"people" + 0.002*"legall
y" + 0.002*"funny" + 0.002*"little" + 0.002*"still" + 0.002*"would" + 0.002*"horro
r"'), (1, '0.008*"movie" + 0.004*"first" + 0.004*"ahmed" + 0.004*"story" + 0.003*"hol
mes" + 0.003*"kinley" + 0.003*"would" + 0.003*"movies" + 0.003*"doesnt" + 0.003*"char
acters"'), (2, '0.007*"movie" + 0.003*"action" + 0.003*"films" + 0.003*"something" +
0.002*"horror" + 0.002*"character" + 0.002*"years" + 0.002*"first" + 0.002*"would" +
0.002*"story"'), (3, '0.007*"movie" + 0.004*"films" + 0.004*"knight" + 0.004*"horror"
+ 0.003*"first" + 0.003*"toxic" + 0.002*"batman" + 0.002*"black" + 0.002*"chapter" +
0.002*"years"')]
```



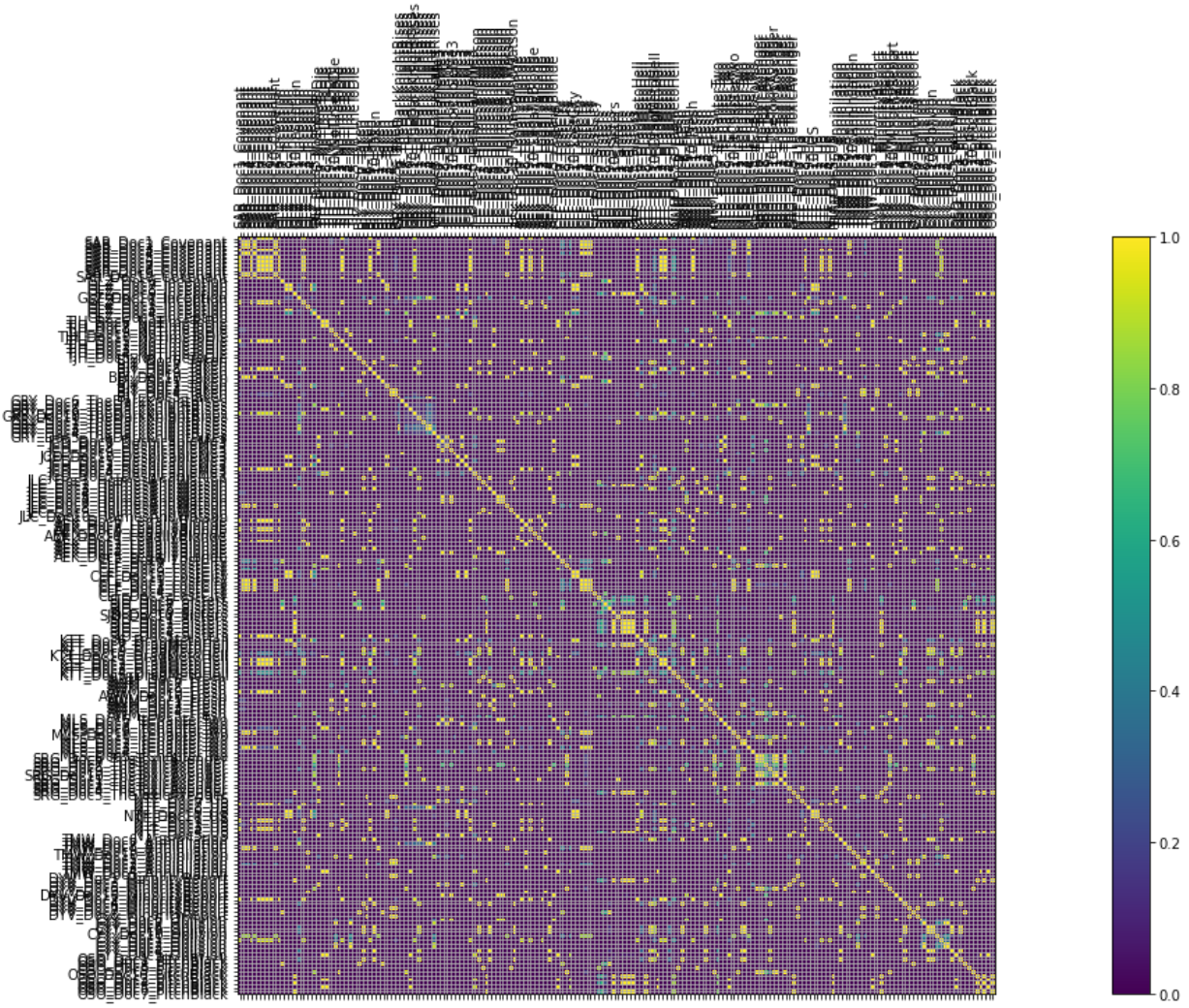
In [65]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 4, number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3
0	movie	movie	films	movie
1	films	first	blonde	first
2	first	films	first	ahmed
3	blonde	story	people	story
4	people	movies	legally	holmes
5	years	would	funny	kinley
6	would	horror	little	would
7	horror	action	still	movies
8	character	characters	would	doesnt
9	legally	people	horror	characters

Topic Modeling Experiment 12: 19-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 1

In [66]:
```
run_lda(processed_text = processed_text_method_one,
       number_of_topics = 19,
       words = 10)
```

```
[(0, '0.005*"loretta" + 0.004*"blonde" + 0.004*"house" + 0.003*"funny" + 0.003*"seque
nce" + 0.003*"christine" + 0.003*"first" + 0.003*"audience" + 0.003*"school" + 0.003
*"tough"'), (1, '0.006*"first" + 0.004*"watson" + 0.004*"despicable" + 0.004*"holmes"
+ 0.004*"future" + 0.004*"might" + 0.003*"story" + 0.003*"movie" + 0.003*"characters"
+ 0.003*"films"'), (2, '0.008*"movie" + 0.007*"action" + 0.005*"films" + 0.004*"neeso
n" + 0.004*"human" + 0.004*"something" + 0.003*"daughter" + 0.003*"thats" + 0.003*"ga
rland" + 0.003*"great"'), (3, '0.006*"films" + 0.004*"movie" + 0.004*"planet" + 0.003
*"first" + 0.003*"would" + 0.003*"character" + 0.003*"loretta" + 0.003*"never" + 0.00
2*"though" + 0.002*"series"'), (4, '0.009*"oblivion" + 0.008*"earth" + 0.006*"scifi"
+ 0.005*"cruise" + 0.005*"movie" + 0.005*"victoria" + 0.004*"films" + 0.004*"first" +
0.004*"planet" + 0.004*"kosinski"'), (5, '0.015*"toxic" + 0.009*"avenger" + 0.008*"mo
vie" + 0.008*"melvin" + 0.006*"horror" + 0.005*"films" + 0.005*"movies" + 0.005*"wast
e" + 0.004*"troma" + 0.004*"people"'), (6, '0.008*"holmes" + 0.006*"watson" + 0.005
*"reilly" + 0.005*"creatures" + 0.005*"people" + 0.004*"steve" + 0.004*"someone" + 0.
004*"would" + 0.003*"ferrell" + 0.003*"fresh"'), (7, '0.005*"fresh" + 0.005*"first" +
0.004*"craig" + 0.004*"steve" + 0.003*"familiar" + 0.003*"something" + 0.003*"thats"
+ 0.003*"films" + 0.003*"seems" + 0.003*"takes"'), (8, '0.011*"movie" + 0.006*"ahmed"
+ 0.005*"horror" + 0.004*"kinley" + 0.004*"action" + 0.004*"films" + 0.004*"ritchie"
+ 0.004*"first" + 0.004*"story" + 0.003*"blonde"'), (9, '0.008*"movie" + 0.006*"part
y" + 0.005*"character" + 0.005*"sisters" + 0.004*"poehler" + 0.004*"would" + 0.004*"d
aughter" + 0.003*"spielberg" + 0.003*"maura" + 0.003*"people"'), (10, '0.008*"movie"
+ 0.005*"people" + 0.005*"would" + 0.004*"holmes" + 0.004*"characters" + 0.003*"obliv
ion" + 0.003*"scene" + 0.003*"movies" + 0.003*"comedy" + 0.003*"doesnt"'), (11, '0.00
8*"sisters" + 0.006*"poehler" + 0.006*"movie" + 0.004*"years" + 0.004*"maura" + 0.003
*"theres" + 0.003*"family" + 0.003*"pitch" + 0.003*"riddick" + 0.003*"first"'), (12,
'0.008*"dreams" + 0.005*"movie" + 0.005*"inception" + 0.004*"dream" + 0.004*"adelaid
e" + 0.004*"family" + 0.004*"really" + 0.004*"nolans" + 0.003*"despicable" + 0.003*"h
orror"'), (13, '0.019*"knight" + 0.010*"rises" + 0.010*"batman" + 0.009*"gotham" + 0.
009*"wayne" + 0.009*"nolan" + 0.007*"nolans" + 0.005*"bruce" + 0.005*"selina" + 0.004
*"christopher"'), (14, '0.005*"films" + 0.004*"toxic" + 0.004*"doesnt" + 0.004*"schoo
l" + 0.004*"movie" + 0.003*"comedy" + 0.003*"warner" + 0.003*"fresh" + 0.003*"steve"
+ 0.003*"around"'), (15, '0.012*"movie" + 0.007*"kinley" + 0.006*"ahmed" + 0.005*"cha
pter" + 0.005*"covenant" + 0.004*"loretta" + 0.004*"first" + 0.004*"taliban" + 0.004
*"action" + 0.004*"tatum"'), (16, '0.008*"movie" + 0.008*"despicable" + 0.008*"minion
s" + 0.004*"first" + 0.004*"story" + 0.003*"monster" + 0.003*"movies" + 0.003*"james"
+ 0.003*"films" + 0.003*"brother"'), (17, '0.008*"derry" + 0.006*"chapter" + 0.005*"l
osers" + 0.004*"pennywise" + 0.004*"films" + 0.004*"series" + 0.004*"would" + 0.003
*"years" + 0.003*"batman" + 0.003*"knight"'), (18, '0.005*"blonde" + 0.004*"annihilat
ion" + 0.003*"films" + 0.003*"story" + 0.003*"characters" + 0.003*"character" + 0.003
*"school" + 0.003*"dream" + 0.003*"fiction" + 0.003*"future"')]
```



In [67]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 19, number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topi
0	movie	movie	films	movie	movie	movie	loretta	first	movie	fil
1	films	first	blonde	first	action	films	blonde	watson	action	mo
2	first	films	first	ahmed	films	knight	house	despicable	films	pla
3	blonde	story	people	story	something	horror	funny	holmes	neeson	f
4	people	movies	legally	holmes	horror	first	sequence	future	human	wo
5	years	would	funny	kinley	character	toxic	christine	might	something	charac
6	would	horror	little	would	years	batman	first	story	daughter	lore
7	horror	action	still	movies	first	black	audience	movie	thats	ne
8	character	characters	would	doesnt	would	chapter	school	characters	garland	thou
9	legally	people	horror	characters	story	years	tough	films	great	se

In [68]:
```
topics = [2, 4, 19]
coherence_values = []

def lda_coherence_function(processed_text, words):
    for t in topics:
        ldamodel, dictionary, index, matrix = create_gensim_lda_model(processed_text,


        coherence_model_lda = CoherenceModel(model=ldamodel,
                                             dictionary=dictionary,
                                             texts=processed_text,
                                             coherence='c_v')
        coherence_lda = coherence_model_lda.get_coherence()
        coherence_values.append(coherence_lda)


    coherence ={f'2 topics and {words} words':coherence_values[0],
                f'4 topics and {words} words': coherence_values[1],
                f'19 topics and {words} words':coherence_values[2]}

    print(coherence)

lda_coherence_function(processed_text = processed_text_method_one, words = 10)
```

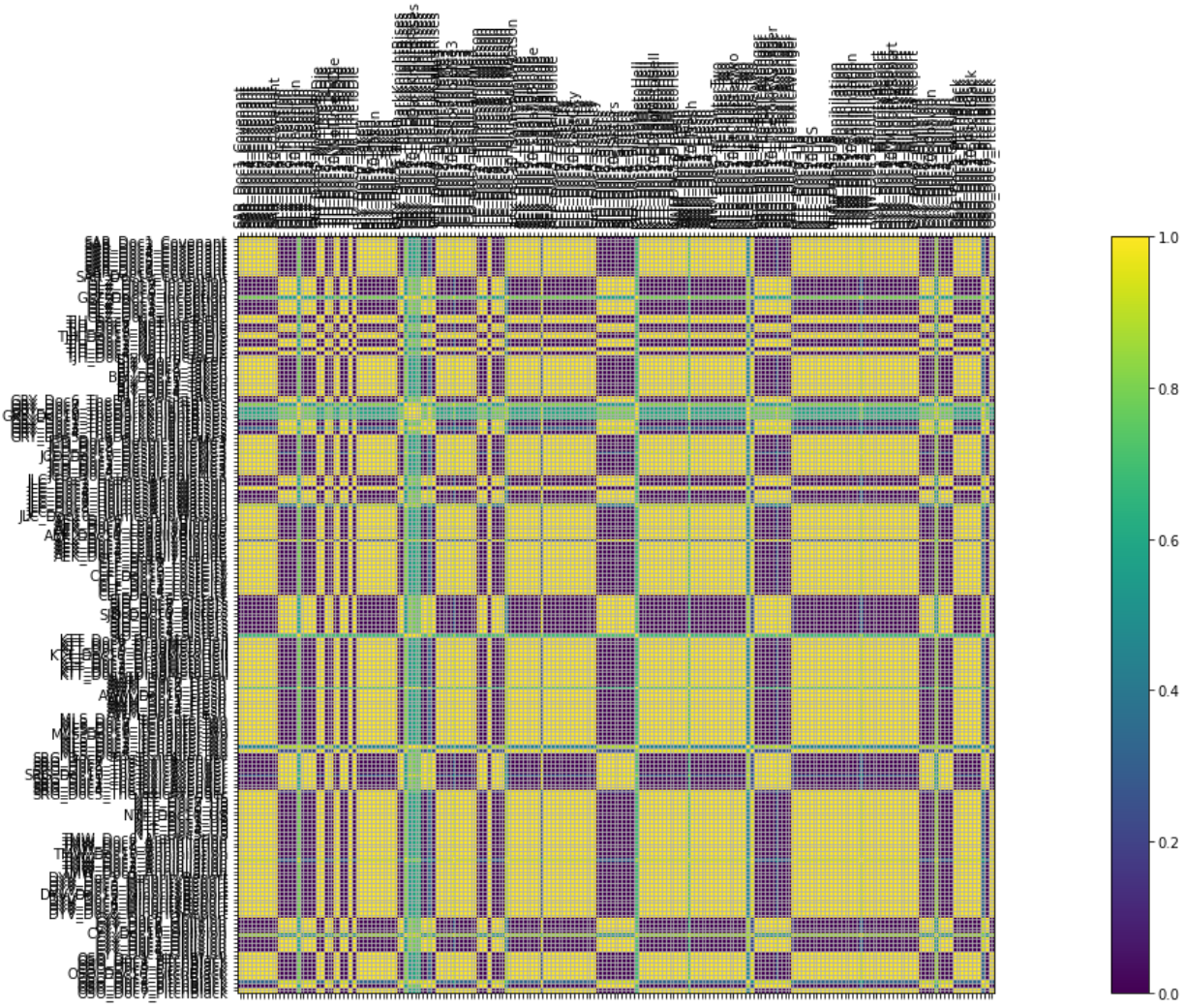
```
[(0, '0.003*"movie" + 0.003*"films" + 0.002*"first" + 0.002*"blonde" + 0.002*"people"
+ 0.002*"years" + 0.002*"would" + 0.002*"horror" + 0.002*"character" + 0.002*"legall
y"'), (1, '0.008*"movie" + 0.003*"first" + 0.003*"films" + 0.003*"story" + 0.003*"mov
ies" + 0.002*"would" + 0.002*"horror" + 0.002*"action" + 0.002*"characters" + 0.002
*"people"')]
[(0, '0.003*"films" + 0.003*"blonde" + 0.002*"first" + 0.002*"people" + 0.002*"legall
y" + 0.002*"funny" + 0.002*"little" + 0.002*"still" + 0.002*"would" + 0.002*"horro
r"'), (1, '0.008*"movie" + 0.004*"first" + 0.004*"ahmed" + 0.004*"story" + 0.003*"hol
mes" + 0.003*"kinley" + 0.003*"would" + 0.003*"movies" + 0.003*"doesnt" + 0.003*"char
acters"'), (2, '0.007*"movie" + 0.003*"action" + 0.003*"films" + 0.003*"something" +
0.002*"horror" + 0.002*"character" + 0.002*"years" + 0.002*"first" + 0.002*"would" +
0.002*"story"'), (3, '0.007*"movie" + 0.004*"films" + 0.004*"knight" + 0.004*"horror"
+ 0.003*"first" + 0.003*"toxic" + 0.002*"batman" + 0.002*"black" + 0.002*"chapter" +
0.002*"years"')]
[(0, '0.005*"loretta" + 0.004*"blonde" + 0.004*"house" + 0.003*"funny" + 0.003*"seque
nce" + 0.003*"christine" + 0.003*"first" + 0.003*"audience" + 0.003*"school" + 0.003
*"tough"'), (1, '0.006*"first" + 0.004*"watson" + 0.004*"despicable" + 0.004*"holmes"
+ 0.004*"future" + 0.004*"might" + 0.003*"story" + 0.003*"movie" + 0.003*"characters"
+ 0.003*"films"'), (2, '0.008*"movie" + 0.007*"action" + 0.005*"films" + 0.004*"neeso
n" + 0.004*"human" + 0.004*"something" + 0.003*"daughter" + 0.003*"thats" + 0.003*"ga
rland" + 0.003*"great"'), (3, '0.006*"films" + 0.004*"movie" + 0.004*"planet" + 0.003
*"first" + 0.003*"would" + 0.003*"character" + 0.003*"loretta" + 0.003*"never" + 0.00
2*"though" + 0.002*"series"'), (4, '0.009*"oblivion" + 0.008*"earth" + 0.006*"scifi"
+ 0.005*"cruise" + 0.005*"movie" + 0.005*"victoria" + 0.004*"films" + 0.004*"first" +
0.004*"planet" + 0.004*"kosinski"'), (5, '0.015*"toxic" + 0.009*"avenger" + 0.008*"mo
vie" + 0.008*"melvin" + 0.006*"horror" + 0.005*"films" + 0.005*"movies" + 0.005*"wast
e" + 0.004*"troma" + 0.004*"people"'), (6, '0.008*"holmes" + 0.006*"watson" + 0.005
*"reilly" + 0.005*"creatures" + 0.005*"people" + 0.004*"steve" + 0.004*"someone" + 0.
004*"would" + 0.003*"ferrell" + 0.003*"fresh"'), (7, '0.005*"fresh" + 0.005*"first" +
0.004*"craig" + 0.004*"steve" + 0.003*"familiar" + 0.003*"something" + 0.003*"thats"
+ 0.003*"films" + 0.003*"seems" + 0.003*"takes"'), (8, '0.011*"movie" + 0.006*"ahmed"
+ 0.005*"horror" + 0.004*"kinley" + 0.004*"action" + 0.004*"films" + 0.004*"ritchie"
+ 0.004*"first" + 0.004*"story" + 0.003*"blonde"'), (9, '0.008*"movie" + 0.006*"part
y" + 0.005*"character" + 0.005*"sisters" + 0.004*"poehler" + 0.004*"would" + 0.004*"d
aughter" + 0.003*"spielberg" + 0.003*"maura" + 0.003*"people"'), (10, '0.008*"movie"
+ 0.005*"people" + 0.005*"would" + 0.004*"holmes" + 0.004*"characters" + 0.003*"obliv
ion" + 0.003*"scene" + 0.003*"movies" + 0.003*"comedy" + 0.003*"doesnt"'), (11, '0.00
8*"sisters" + 0.006*"poehler" + 0.006*"movie" + 0.004*"years" + 0.004*"maura" + 0.003
*"theres" + 0.003*"family" + 0.003*"pitch" + 0.003*"riddick" + 0.003*"first"'), (12,
'0.008*"dreams" + 0.005*"movie" + 0.005*"inception" + 0.004*"dream" + 0.004*"adelaid
e" + 0.004*"family" + 0.004*"really" + 0.004*"nolans" + 0.003*"despicable" + 0.003*"h
orror"'), (13, '0.019*"knight" + 0.010*"rises" + 0.010*"batman" + 0.009*"gotham" + 0.
009*"wayne" + 0.009*"nolan" + 0.007*"nolans" + 0.005*"bruce" + 0.005*"selina" + 0.004
*"christopher"'), (14, '0.005*"films" + 0.004*"toxic" + 0.004*"doesnt" + 0.004*"schoo
l" + 0.004*"movie" + 0.003*"comedy" + 0.003*"warner" + 0.003*"fresh" + 0.003*"steve"
+ 0.003*"around"'), (15, '0.012*"movie" + 0.007*"kinley" + 0.006*"ahmed" + 0.005*"cha
pter" + 0.005*"covenant" + 0.004*"loretta" + 0.004*"first" + 0.004*"taliban" + 0.004
*"action" + 0.004*"tatum"'), (16, '0.008*"movie" + 0.008*"despicable" + 0.008*"minion
s" + 0.004*"first" + 0.004*"story" + 0.003*"monster" + 0.003*"movies" + 0.003*"james"
+ 0.003*"films" + 0.003*"brother"'), (17, '0.008*"derry" + 0.006*"chapter" + 0.005*"l
osers" + 0.004*"pennywise" + 0.004*"films" + 0.004*"series" + 0.004*"would" + 0.003
*"years" + 0.003*"batman" + 0.003*"knight"'), (18, '0.005*"blonde" + 0.004*"annihilat
ion" + 0.003*"films" + 0.003*"story" + 0.003*"characters" + 0.003*"character" + 0.003
*"school" + 0.003*"dream" + 0.003*"fiction" + 0.003*"future"')]
{'2 topics and 10 words': 0.24298533572424605, '4 topics and 10 words': 0.24364827409
55987, '19 topics and 10 words': 0.31434878311440023}
```

Topic Modeling Experiment 13: 2-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 2

In [162…

```
topic_word_list = []

run_lda(processed_text = processed_text_method_three,
       number_of_topics = 2,
       words = 10)
```

```
[(0, '0.004*"dream" + 0.004*"character" + 0.003*"toxic" + 0.003*"sister" + 0.003*"kni
ght" + 0.002*"people" + 0.002*"nolan" + 0.002*"first" + 0.002*"poehler" + 0.002*"holm
es"'), (1, '0.004*"character" + 0.004*"first" + 0.004*"horror" + 0.003*"story" + 0.00
3*"action" + 0.003*"people" + 0.002*"doesnt" + 0.002*"family" + 0.002*"woman" + 0.002
*"director"')]
```



In [163…

```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 2, number_of_words = 10)
```

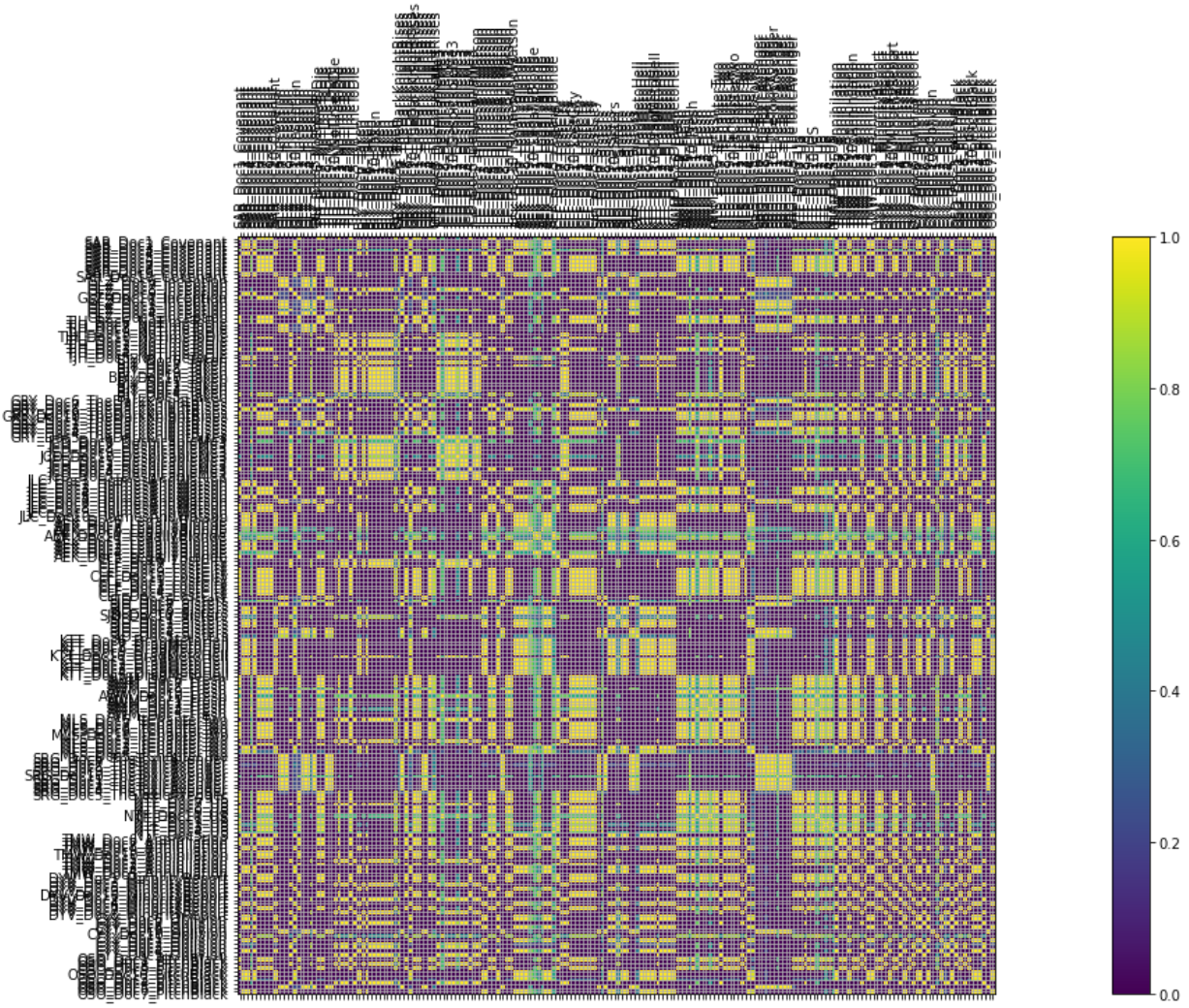
	Topic 0	Topic 1
0	dream	character
1	character	first
2	toxic	horror
3	sister	story
4	knight	action
5	people	people
6	nolan	doesnt
7	first	family
8	poehler	woman
9	holmes	director

Topic Modeling Experiment 14: 4-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 2

In [164…

```
topic_word_list = []

run_lda(processed_text = processed_text_method_three,
       number_of_topics = 4,
       words = 10)
```

```
[(0, '0.007*"dream" + 0.007*"toxic" + 0.005*"character" + 0.004*"avenger" + 0.004*"me
lvin" + 0.003*"knight" + 0.003*"nolans" + 0.003*"people" + 0.003*"nolan" + 0.003*"inc
eption"'), (1, '0.004*"horror" + 0.004*"character" + 0.004*"first" + 0.003*"story" +
0.003*"people" + 0.003*"doesnt" + 0.003*"family" + 0.003*"action" + 0.003*"director"
+ 0.002*"year"'), (2, '0.004*"despicable" + 0.004*"daughter" + 0.004*"character" + 0.
004*"first" + 0.003*"minion" + 0.003*"story" + 0.003*"action" + 0.002*"minute" + 0.00
2*"three" + 0.002*"never"'), (3, '0.004*"blonde" + 0.003*"character" + 0.003*"first"
+ 0.003*"woman" + 0.003*"sister" + 0.003*"christine" + 0.003*"legally" + 0.002*"peopl
e" + 0.002*"something" + 0.002*"horror"')]
```



In [165…

```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 4, number_of_words = 10)
```

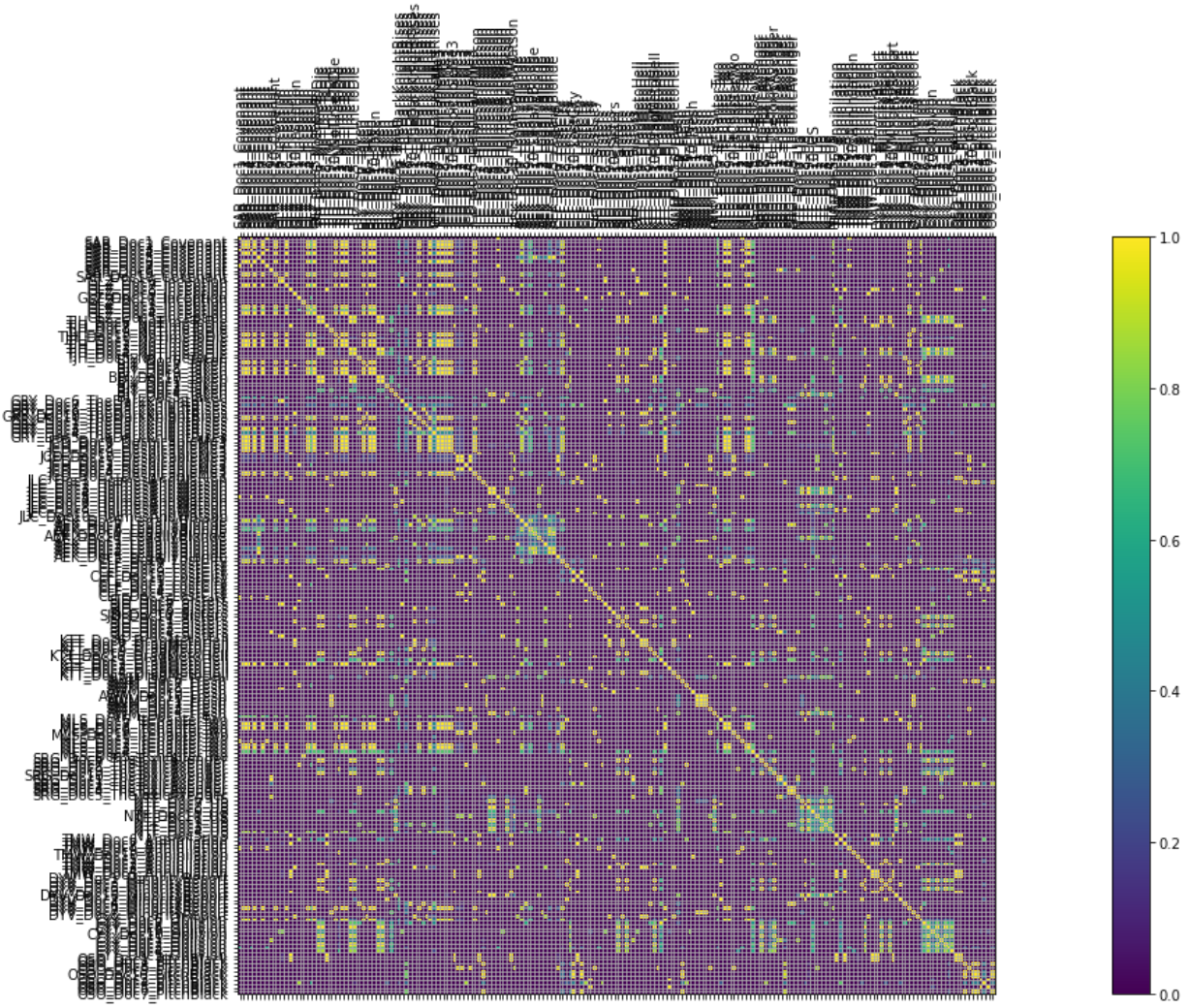
	Topic 0	Topic 1	Topic 2	Topic 3
0	dream	horror	despicable	blonde
1	toxic	character	daughter	character
2	character	first	character	first
3	avenger	story	first	woman
4	melvin	people	minion	sister
5	knight	doesnt	story	christine
6	nolans	family	action	legally
7	people	action	minute	people
8	nolan	director	three	something
9	inception	year	never	horror

Topic Modeling Experiment 15: 19-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 2

In [166…

```
topic_word_list = []

run_lda(processed_text = processed_text_method_three,
       number_of_topics = 19,
       words = 10)
```

```
[(0, '0.005*"inception" + 0.004*"dream" + 0.004*"character" + 0.003*"sister" + 0.003
*"little" + 0.003*"always" + 0.003*"party" + 0.003*"nolan" + 0.003*"story" + 0.003*"r
aimi"'), (1, '0.011*"horror" + 0.008*"holmes" + 0.007*"peele" + 0.007*"adelaide" + 0.
007*"family" + 0.005*"nyongo" + 0.005*"watson" + 0.004*"there" + 0.004*"jason" + 0.00
4*"peeles"'), (2, '0.004*"never" + 0.004*"story" + 0.004*"adaptation" + 0.004*"garlan
d" + 0.003*"minute" + 0.003*"doesnt" + 0.003*"death" + 0.003*"first" + 0.003*"daughte
r" + 0.003*"author"'), (3, '0.005*"knight" + 0.005*"action" + 0.004*"sister" + 0.004
*"christine" + 0.004*"horror" + 0.003*"theyre" + 0.003*"nolan" + 0.003*"director" +
0.003*"youre" + 0.003*"nolans"'), (4, '0.017*"toxic" + 0.012*"avenger" + 0.006*"comed
y" + 0.006*"melvin" + 0.006*"troma" + 0.005*"action" + 0.005*"horror" + 0.004*"waste"
+ 0.004*"getting" + 0.003*"place"'), (5, '0.007*"oblivion" + 0.006*"woman" + 0.005*"c
ruise" + 0.004*"action" + 0.004*"character" + 0.004*"toxic" + 0.004*"earth" + 0.004
*"going" + 0.004*"daughter" + 0.004*"victoria"'), (6, '0.006*"planet" + 0.006*"creatu
re" + 0.004*"christine" + 0.004*"though" + 0.003*"diesel" + 0.003*"might" + 0.003*"ta
tum" + 0.003*"think" + 0.003*"named" + 0.002*"turn"'), (7, '0.004*"character" + 0.004
*"shimmer" + 0.004*"without" + 0.003*"sense" + 0.003*"series" + 0.003*"garland" + 0.0
03*"holmes" + 0.003*"alien" + 0.003*"annihilation" + 0.003*"people"'), (8, '0.006*"cr
eature" + 0.005*"family" + 0.005*"loretta" + 0.004*"brother" + 0.004*"people" + 0.004
*"villain" + 0.004*"despicable" + 0.003*"horror" + 0.003*"first" + 0.003*"minion"'),
(9, '0.007*"batman" + 0.006*"knight" + 0.005*"character" + 0.005*"gotham" + 0.005*"wa
yne" + 0.004*"monster" + 0.004*"people" + 0.004*"rise" + 0.004*"bruce" + 0.003*"selin
a"'), (10, '0.005*"character" + 0.005*"horror" + 0.004*"score" + 0.004*"first" + 0.00
3*"mission" + 0.003*"chapter" + 0.003*"weird" + 0.003*"fiction" + 0.003*"another" +
0.003*"certain"'), (11, '0.006*"first" + 0.005*"fresh" + 0.005*"character" + 0.005*"w
oman" + 0.004*"holmes" + 0.004*"opening" + 0.004*"steve" + 0.003*"people" + 0.003*"sp
ielberg" + 0.003*"surprise"'), (12, '0.010*"dream" + 0.008*"point" + 0.006*"sister" +
0.006*"character" + 0.005*"child" + 0.005*"poehler" + 0.005*"fisher" + 0.004*"first"
+ 0.004*"second" + 0.004*"comedy"'), (13, '0.007*"ahmed" + 0.007*"kinley" + 0.006*"in
terpreter" + 0.005*"first" + 0.005*"ritchie" + 0.005*"annihilation" + 0.004*"covenan
t" + 0.004*"character" + 0.004*"taliban" + 0.004*"afghan"'), (14, '0.010*"character"
+ 0.009*"pitch" + 0.009*"black" + 0.006*"loretta" + 0.006*"riddick" + 0.006*"bullock"
+ 0.005*"planet" + 0.005*"comedy" + 0.005*"tatum" + 0.005*"diesel"'), (15, '0.006*"po
ehler" + 0.006*"annihilation" + 0.005*"sister" + 0.004*"screen" + 0.004*"moment" + 0.
004*"friend" + 0.004*"playing" + 0.004*"night" + 0.004*"maura" + 0.004*"inside"'), (1
6, '0.025*"blonde" + 0.018*"legally" + 0.014*"school" + 0.014*"witherspoon" + 0.012
*"warner" + 0.012*"harvard" + 0.006*"played" + 0.006*"ahmed" + 0.006*"luketic" + 0.00
6*"comedy"'), (17, '0.005*"first" + 0.005*"dream" + 0.005*"character" + 0.004*"story"
+ 0.004*"despicable" + 0.004*"little" + 0.004*"kinley" + 0.004*"year" + 0.004*"ahmed"
+ 0.003*"world"'), (18, '0.007*"people" + 0.006*"character" + 0.005*"moment" + 0.004
*"party" + 0.004*"sister" + 0.004*"story" + 0.004*"group" + 0.003*"first" + 0.003*"fr
esh" + 0.003*"world"')]
```



In [167…

```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 19, number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topi
0	inception	horror	never	knight	toxic	oblivion	planet	character	creature	batm
1	dream	holmes	story	action	avenger	woman	creature	shimmer	family	knig
2	character	peele	adaptation	sister	comedy	cruise	christine	without	loretta	charac
3	sister	adelaide	garland	christine	melvin	action	though	sense	brother	gotha
4	little	family	minute	horror	troma	character	diesel	series	people	way
5	always	nyongo	doesnt	theyre	action	toxic	might	garland	villain	mons
6	party	watson	death	nolan	horror	earth	tatum	holmes	despicable	peop
7	nolan	there	first	director	waste	going	think	alien	horror	r
8	story	jason	daughter	youre	getting	daughter	named	annihilation	first	bru
9	raimi	peeles	author	nolans	place	victoria	turn	people	minion	seli

In [168…

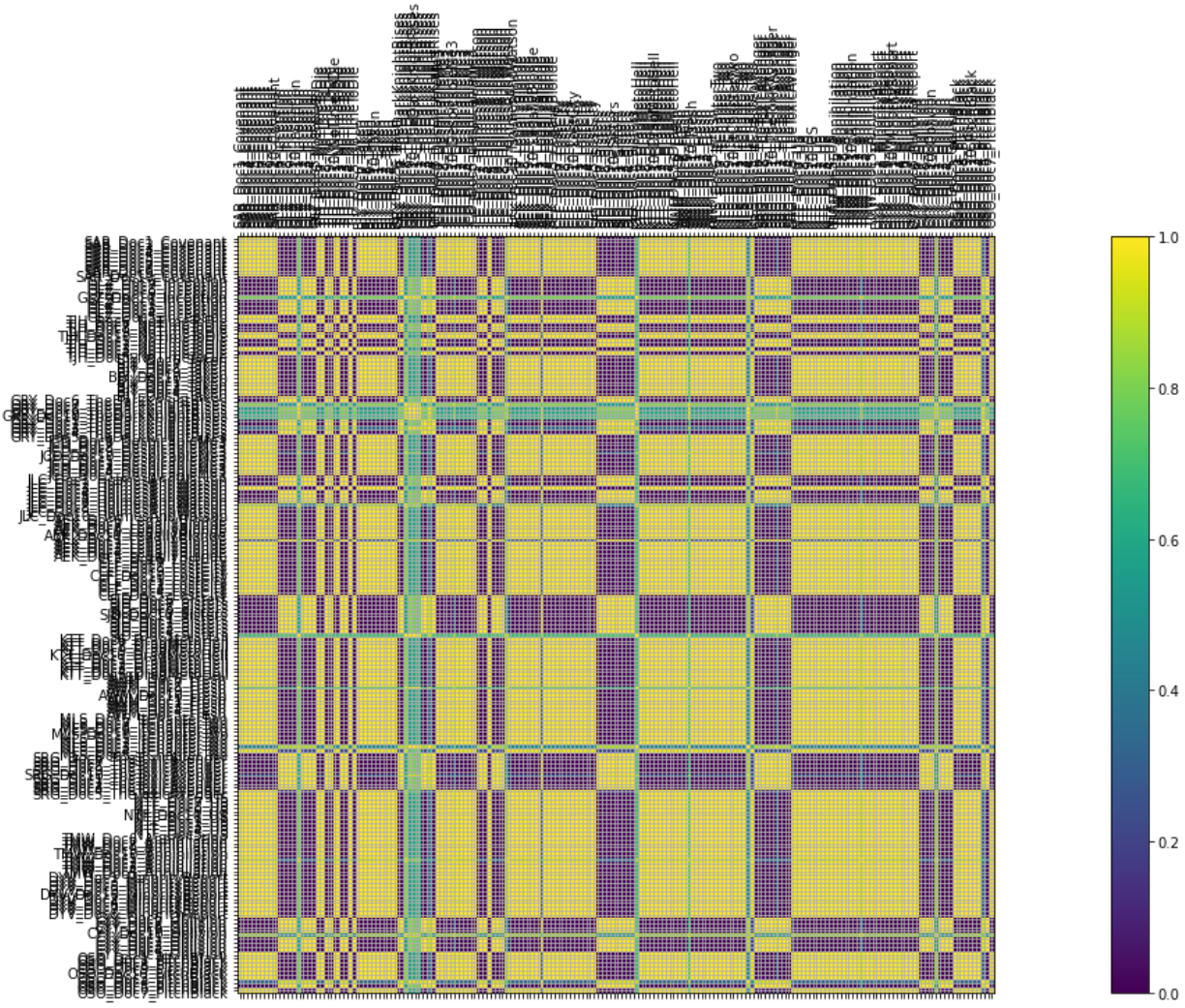
```
topics = [2, 4, 19]
coherence_values = []

lda_coherence_function(processed_text = processed_text_method_three, words = 10)
```

```
[(0, '0.004*"dream" + 0.004*"character" + 0.003*"toxic" + 0.003*"sister" + 0.003*"kni
ght" + 0.002*"people" + 0.002*"nolan" + 0.002*"first" + 0.002*"poehler" + 0.002*"holm
es"'), (1, '0.004*"character" + 0.004*"first" + 0.004*"horror" + 0.003*"story" + 0.00
3*"action" + 0.003*"people" + 0.002*"doesnt" + 0.002*"family" + 0.002*"woman" + 0.002
*"director"')]
[(0, '0.007*"dream" + 0.007*"toxic" + 0.005*"character" + 0.004*"avenger" + 0.004*"me
lvin" + 0.003*"knight" + 0.003*"nolans" + 0.003*"people" + 0.003*"nolan" + 0.003*"inc
eption"'), (1, '0.004*"horror" + 0.004*"character" + 0.004*"first" + 0.003*"story" +
0.003*"people" + 0.003*"doesnt" + 0.003*"family" + 0.003*"action" + 0.003*"director"
+ 0.002*"year"'), (2, '0.004*"despicable" + 0.004*"daughter" + 0.004*"character" + 0.
004*"first" + 0.003*"minion" + 0.003*"story" + 0.003*"action" + 0.002*"minute" + 0.00
2*"three" + 0.002*"never"'), (3, '0.004*"blonde" + 0.003*"character" + 0.003*"first"
+ 0.003*"woman" + 0.003*"sister" + 0.003*"christine" + 0.003*"legally" + 0.002*"peopl
e" + 0.002*"something" + 0.002*"horror"')]
[(0, '0.005*"inception" + 0.004*"dream" + 0.004*"character" + 0.003*"sister" + 0.003
*"little" + 0.003*"always" + 0.003*"party" + 0.003*"nolan" + 0.003*"story" + 0.003*"r
aimi"'), (1, '0.011*"horror" + 0.008*"holmes" + 0.007*"peele" + 0.007*"adelaide" + 0.
007*"family" + 0.005*"nyongo" + 0.005*"watson" + 0.004*"there" + 0.004*"jason" + 0.00
4*"peeles"'), (2, '0.004*"never" + 0.004*"story" + 0.004*"adaptation" + 0.004*"garlan
d" + 0.003*"minute" + 0.003*"doesnt" + 0.003*"death" + 0.003*"first" + 0.003*"daughte
r" + 0.003*"author"'), (3, '0.005*"knight" + 0.005*"action" + 0.004*"sister" + 0.004
*"christine" + 0.004*"horror" + 0.003*"theyre" + 0.003*"nolan" + 0.003*"director" +
0.003*"youre" + 0.003*"nolans"'), (4, '0.017*"toxic" + 0.012*"avenger" + 0.006*"comed
y" + 0.006*"melvin" + 0.006*"troma" + 0.005*"action" + 0.005*"horror" + 0.004*"waste"
+ 0.004*"getting" + 0.003*"place"'), (5, '0.007*"oblivion" + 0.006*"woman" + 0.005*"c
ruise" + 0.004*"action" + 0.004*"character" + 0.004*"toxic" + 0.004*"earth" + 0.004
*"going" + 0.004*"daughter" + 0.004*"victoria"'), (6, '0.006*"planet" + 0.006*"creatu
re" + 0.004*"christine" + 0.004*"though" + 0.003*"diesel" + 0.003*"might" + 0.003*"ta
tum" + 0.003*"think" + 0.003*"named" + 0.002*"turn"'), (7, '0.004*"character" + 0.004
*"shimmer" + 0.004*"without" + 0.003*"sense" + 0.003*"series" + 0.003*"garland" + 0.0
03*"holmes" + 0.003*"alien" + 0.003*"annihilation" + 0.003*"people"'), (8, '0.006*"cr
eature" + 0.005*"family" + 0.005*"loretta" + 0.004*"brother" + 0.004*"people" + 0.004
*"villain" + 0.004*"despicable" + 0.003*"horror" + 0.003*"first" + 0.003*"minion"'),
(9, '0.007*"batman" + 0.006*"knight" + 0.005*"character" + 0.005*"gotham" + 0.005*"wa
yne" + 0.004*"monster" + 0.004*"people" + 0.004*"rise" + 0.004*"bruce" + 0.003*"selin
a"'), (10, '0.005*"character" + 0.005*"horror" + 0.004*"score" + 0.004*"first" + 0.00
3*"mission" + 0.003*"chapter" + 0.003*"weird" + 0.003*"fiction" + 0.003*"another" +
0.003*"certain"'), (11, '0.006*"first" + 0.005*"fresh" + 0.005*"character" + 0.005*"w
oman" + 0.004*"holmes" + 0.004*"opening" + 0.004*"steve" + 0.003*"people" + 0.003*"sp
ielberg" + 0.003*"surprise"'), (12, '0.010*"dream" + 0.008*"point" + 0.006*"sister" +
0.006*"character" + 0.005*"child" + 0.005*"poehler" + 0.005*"fisher" + 0.004*"first"
+ 0.004*"second" + 0.004*"comedy"'), (13, '0.007*"ahmed" + 0.007*"kinley" + 0.006*"in
terpreter" + 0.005*"first" + 0.005*"ritchie" + 0.005*"annihilation" + 0.004*"covenan
t" + 0.004*"character" + 0.004*"taliban" + 0.004*"afghan"'), (14, '0.010*"character"
+ 0.009*"pitch" + 0.009*"black" + 0.006*"loretta" + 0.006*"riddick" + 0.006*"bullock"
+ 0.005*"planet" + 0.005*"comedy" + 0.005*"tatum" + 0.005*"diesel"'), (15, '0.006*"po
ehler" + 0.006*"annihilation" + 0.005*"sister" + 0.004*"screen" + 0.004*"moment" + 0.
004*"friend" + 0.004*"playing" + 0.004*"night" + 0.004*"maura" + 0.004*"inside"'), (1
6, '0.025*"blonde" + 0.018*"legally" + 0.014*"school" + 0.014*"witherspoon" + 0.012
*"warner" + 0.012*"harvard" + 0.006*"played" + 0.006*"ahmed" + 0.006*"luketic" + 0.00
6*"comedy"'), (17, '0.005*"first" + 0.005*"dream" + 0.005*"character" + 0.004*"story"
+ 0.004*"despicable" + 0.004*"little" + 0.004*"kinley" + 0.004*"year" + 0.004*"ahmed"
+ 0.003*"world"'), (18, '0.007*"people" + 0.006*"character" + 0.005*"moment" + 0.004
*"party" + 0.004*"sister" + 0.004*"story" + 0.004*"group" + 0.003*"first" + 0.003*"fr
esh" + 0.003*"world"')]
{'2 topics and 10 words': 0.2887710660386871, '4 topics and 10 words': 0.284540054001
85534, '19 topics and 10 words': 0.31361824462762233}
```

Topic Modeling Experiment 16: 2-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 3

In [69]:
```
topic_word_list = []

run_lda(processed_text = processed_text_method_four,
       number_of_topics = 2,
       words = 10)
```

```
[(0, '0.004*"dream" + 0.004*"character" + 0.003*"toxic" + 0.003*"sister" + 0.003*"kni
ght" + 0.002*"people" + 0.002*"nolan" + 0.002*"first" + 0.002*"poehler" + 0.002*"holm
es"'), (1, '0.004*"character" + 0.004*"first" + 0.004*"horror" + 0.003*"story" + 0.00
3*"action" + 0.003*"people" + 0.002*"doesnt" + 0.002*"family" + 0.002*"woman" + 0.002
*"director"')]
```



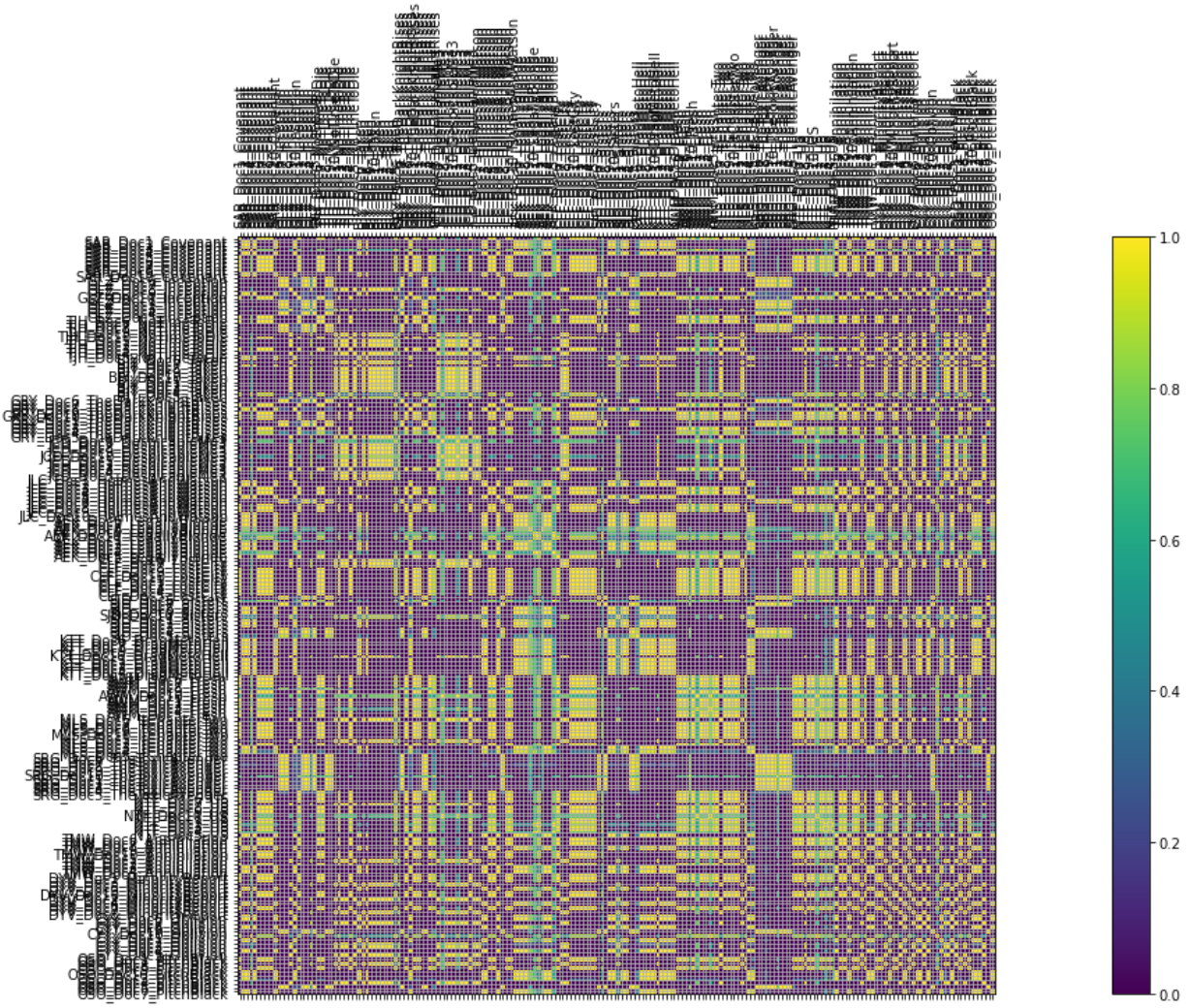
In [70]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 2, number_of_words = 10)
```

	Topic 0	Topic 1
0	dream	character
1	character	first
2	toxic	horror
3	sister	story
4	knight	action
5	people	people
6	nolan	doesnt
7	first	family
8	poehler	woman
9	holmes	director

Topic Modeling Experiment 17: 4-Topic Latent Dirichlet Allocation with Data Wrangling and Vectorization Method 3

In [71]:
```
topic_word_list = []

run_lda(processed_text = processed_text_method_four,
       number_of_topics = 4,
       words = 10)
```

```
[(0, '0.007*"dream" + 0.007*"toxic" + 0.005*"character" + 0.004*"avenger" + 0.004*"me
lvin" + 0.003*"knight" + 0.003*"nolans" + 0.003*"people" + 0.003*"nolan" + 0.003*"inc
eption"'), (1, '0.004*"horror" + 0.004*"character" + 0.004*"first" + 0.003*"story" +
0.003*"people" + 0.003*"doesnt" + 0.003*"family" + 0.003*"action" + 0.003*"director"
+ 0.002*"year"'), (2, '0.004*"despicable" + 0.004*"daughter" + 0.004*"character" + 0.
004*"first" + 0.003*"minion" + 0.003*"story" + 0.003*"action" + 0.002*"minute" + 0.00
2*"three" + 0.002*"never"'), (3, '0.004*"blonde" + 0.003*"character" + 0.003*"first"
+ 0.003*"woman" + 0.003*"sister" + 0.003*"christine" + 0.003*"legally" + 0.002*"peopl
e" + 0.002*"something" + 0.002*"horror"')]
```



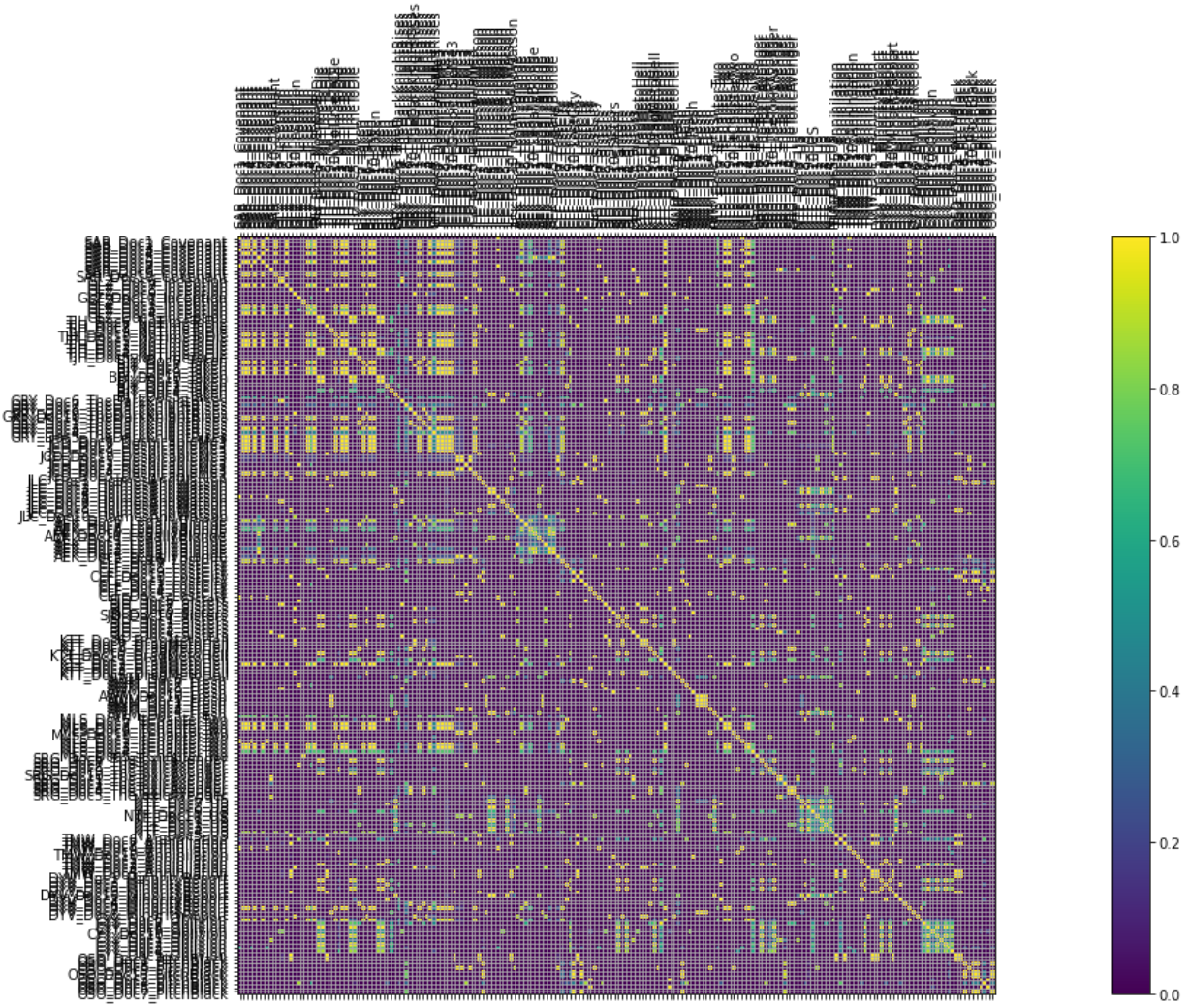
In [72]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 4, number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3
0	dream	horror	despicable	blonde
1	toxic	character	daughter	character
2	character	first	character	first
3	avenger	story	first	woman
4	melvin	people	minion	sister
5	knight	doesnt	story	christine
6	nolans	family	action	legally
7	people	action	minute	people
8	nolan	director	three	something
9	inception	year	never	horror

Topic Modeling Experiment 18: 19-Topic Latent Dirichlet Allocaiton with Data Wrangling and Vectorization Method 3

In [73]:
```
topic_word_list = []

run_lda(processed_text = processed_text_method_four,
       number_of_topics = 19,
       words = 10)
```

```
[(0, '0.005*"inception" + 0.004*"dream" + 0.004*"character" + 0.003*"sister" + 0.003
*"little" + 0.003*"always" + 0.003*"party" + 0.003*"nolan" + 0.003*"story" + 0.003*"r
aimi"'), (1, '0.011*"horror" + 0.008*"holmes" + 0.007*"peele" + 0.007*"adelaide" + 0.
007*"family" + 0.005*"nyongo" + 0.005*"watson" + 0.004*"there" + 0.004*"jason" + 0.00
4*"peeles"'), (2, '0.004*"never" + 0.004*"story" + 0.004*"adaptation" + 0.004*"garlan
d" + 0.003*"minute" + 0.003*"doesnt" + 0.003*"death" + 0.003*"first" + 0.003*"daughte
r" + 0.003*"author"'), (3, '0.005*"knight" + 0.005*"action" + 0.004*"sister" + 0.004
*"christine" + 0.004*"horror" + 0.003*"theyre" + 0.003*"nolan" + 0.003*"director" +
0.003*"youre" + 0.003*"nolans"'), (4, '0.017*"toxic" + 0.012*"avenger" + 0.006*"comed
y" + 0.006*"melvin" + 0.006*"troma" + 0.005*"action" + 0.005*"horror" + 0.004*"waste"
+ 0.004*"getting" + 0.003*"place"'), (5, '0.007*"oblivion" + 0.006*"woman" + 0.005*"c
ruise" + 0.004*"action" + 0.004*"character" + 0.004*"toxic" + 0.004*"earth" + 0.004
*"going" + 0.004*"daughter" + 0.004*"victoria"'), (6, '0.006*"planet" + 0.006*"creatu
re" + 0.004*"christine" + 0.004*"though" + 0.003*"diesel" + 0.003*"might" + 0.003*"ta
tum" + 0.003*"think" + 0.003*"named" + 0.002*"turn"'), (7, '0.004*"character" + 0.004
*"shimmer" + 0.004*"without" + 0.003*"sense" + 0.003*"series" + 0.003*"garland" + 0.0
03*"holmes" + 0.003*"alien" + 0.003*"annihilation" + 0.003*"people"'), (8, '0.006*"cr
eature" + 0.005*"family" + 0.005*"loretta" + 0.004*"brother" + 0.004*"people" + 0.004
*"villain" + 0.004*"despicable" + 0.003*"horror" + 0.003*"first" + 0.003*"minion"'),
(9, '0.007*"batman" + 0.006*"knight" + 0.005*"character" + 0.005*"gotham" + 0.005*"wa
yne" + 0.004*"monster" + 0.004*"people" + 0.004*"rise" + 0.004*"bruce" + 0.003*"selin
a"'), (10, '0.005*"character" + 0.005*"horror" + 0.004*"score" + 0.004*"first" + 0.00
3*"mission" + 0.003*"chapter" + 0.003*"weird" + 0.003*"fiction" + 0.003*"another" +
0.003*"certain"'), (11, '0.006*"first" + 0.005*"fresh" + 0.005*"character" + 0.005*"w
oman" + 0.004*"holmes" + 0.004*"opening" + 0.004*"steve" + 0.003*"people" + 0.003*"sp
ielberg" + 0.003*"surprise"'), (12, '0.010*"dream" + 0.008*"point" + 0.006*"sister" +
0.006*"character" + 0.005*"child" + 0.005*"poehler" + 0.005*"fisher" + 0.004*"first"
+ 0.004*"second" + 0.004*"comedy"'), (13, '0.007*"ahmed" + 0.007*"kinley" + 0.006*"in
terpreter" + 0.005*"first" + 0.005*"ritchie" + 0.005*"annihilation" + 0.004*"covenan
t" + 0.004*"character" + 0.004*"taliban" + 0.004*"afghan"'), (14, '0.010*"character"
+ 0.009*"pitch" + 0.009*"black" + 0.006*"loretta" + 0.006*"riddick" + 0.006*"bullock"
+ 0.005*"planet" + 0.005*"comedy" + 0.005*"tatum" + 0.005*"diesel"'), (15, '0.006*"po
ehler" + 0.006*"annihilation" + 0.005*"sister" + 0.004*"screen" + 0.004*"moment" + 0.
004*"friend" + 0.004*"playing" + 0.004*"night" + 0.004*"maura" + 0.004*"inside"'), (1
6, '0.025*"blonde" + 0.018*"legally" + 0.014*"school" + 0.014*"witherspoon" + 0.012
*"warner" + 0.012*"harvard" + 0.006*"played" + 0.006*"ahmed" + 0.006*"luketic" + 0.00
6*"comedy"'), (17, '0.005*"first" + 0.005*"dream" + 0.005*"character" + 0.004*"story"
+ 0.004*"despicable" + 0.004*"little" + 0.004*"kinley" + 0.004*"year" + 0.004*"ahmed"
+ 0.003*"world"'), (18, '0.007*"people" + 0.006*"character" + 0.005*"moment" + 0.004
*"party" + 0.004*"sister" + 0.004*"story" + 0.004*"group" + 0.003*"first" + 0.003*"fr
esh" + 0.003*"world"')]
```



In [74]:
```
topic_number_list = []
words_list = []
split_words_list = []
processed_words_list = []

create_topics_words_df(number_of_topics = 19, number_of_words = 10)
```

	Topic 0	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topi
0	inception	horror	never	knight	toxic	oblivion	planet	character	creature	batm
1	dream	holmes	story	action	avenger	woman	creature	shimmer	family	knig
2	character	peele	adaptation	sister	comedy	cruise	christine	without	loretta	charac
3	sister	adelaide	garland	christine	melvin	action	though	sense	brother	gotha
4	little	family	minute	horror	troma	character	diesel	series	people	way
5	always	nyongo	doesnt	theyre	action	toxic	might	garland	villain	mons
6	party	watson	death	nolan	horror	earth	tatum	holmes	despicable	peop
7	nolan	there	first	director	waste	going	think	alien	horror	r
8	story	jason	daughter	youre	getting	daughter	named	annihilation	first	bru
9	raimi	peeles	author	nolans	place	victoria	turn	people	minion	seli

In [75]:
```
topics = [2, 4, 19]
coherence_values = []

lda_coherence_function(processed_text = processed_text_method_four, words = 10)
```

```
[(0, '0.004*"dream" + 0.004*"character" + 0.003*"toxic" + 0.003*"sister" + 0.003*"kni
ght" + 0.002*"people" + 0.002*"nolan" + 0.002*"first" + 0.002*"poehler" + 0.002*"holm
es"'), (1, '0.004*"character" + 0.004*"first" + 0.004*"horror" + 0.003*"story" + 0.00
3*"action" + 0.003*"people" + 0.002*"doesnt" + 0.002*"family" + 0.002*"woman" + 0.002
*"director"')]
[(0, '0.007*"dream" + 0.007*"toxic" + 0.005*"character" + 0.004*"avenger" + 0.004*"me
lvin" + 0.003*"knight" + 0.003*"nolans" + 0.003*"people" + 0.003*"nolan" + 0.003*"inc
eption"'), (1, '0.004*"horror" + 0.004*"character" + 0.004*"first" + 0.003*"story" +
0.003*"people" + 0.003*"doesnt" + 0.003*"family" + 0.003*"action" + 0.003*"director"
+ 0.002*"year"'), (2, '0.004*"despicable" + 0.004*"daughter" + 0.004*"character" + 0.
004*"first" + 0.003*"minion" + 0.003*"story" + 0.003*"action" + 0.002*"minute" + 0.00
2*"three" + 0.002*"never"'), (3, '0.004*"blonde" + 0.003*"character" + 0.003*"first"
+ 0.003*"woman" + 0.003*"sister" + 0.003*"christine" + 0.003*"legally" + 0.002*"peopl
e" + 0.002*"something" + 0.002*"horror"')]
[(0, '0.005*"inception" + 0.004*"dream" + 0.004*"character" + 0.003*"sister" + 0.003
*"little" + 0.003*"always" + 0.003*"party" + 0.003*"nolan" + 0.003*"story" + 0.003*"r
aimi"'), (1, '0.011*"horror" + 0.008*"holmes" + 0.007*"peele" + 0.007*"adelaide" + 0.
007*"family" + 0.005*"nyongo" + 0.005*"watson" + 0.004*"there" + 0.004*"jason" + 0.00
4*"peeles"'), (2, '0.004*"never" + 0.004*"story" + 0.004*"adaptation" + 0.004*"garlan
d" + 0.003*"minute" + 0.003*"doesnt" + 0.003*"death" + 0.003*"first" + 0.003*"daughte
r" + 0.003*"author"'), (3, '0.005*"knight" + 0.005*"action" + 0.004*"sister" + 0.004
*"christine" + 0.004*"horror" + 0.003*"theyre" + 0.003*"nolan" + 0.003*"director" +
0.003*"youre" + 0.003*"nolans"'), (4, '0.017*"toxic" + 0.012*"avenger" + 0.006*"comed
y" + 0.006*"melvin" + 0.006*"troma" + 0.005*"action" + 0.005*"horror" + 0.004*"waste"
+ 0.004*"getting" + 0.003*"place"'), (5, '0.007*"oblivion" + 0.006*"woman" + 0.005*"c
ruise" + 0.004*"action" + 0.004*"character" + 0.004*"toxic" + 0.004*"earth" + 0.004
*"going" + 0.004*"daughter" + 0.004*"victoria"'), (6, '0.006*"planet" + 0.006*"creatu
re" + 0.004*"christine" + 0.004*"though" + 0.003*"diesel" + 0.003*"might" + 0.003*"ta
tum" + 0.003*"think" + 0.003*"named" + 0.002*"turn"'), (7, '0.004*"character" + 0.004
*"shimmer" + 0.004*"without" + 0.003*"sense" + 0.003*"series" + 0.003*"garland" + 0.0
03*"holmes" + 0.003*"alien" + 0.003*"annihilation" + 0.003*"people"'), (8, '0.006*"cr
eature" + 0.005*"family" + 0.005*"loretta" + 0.004*"brother" + 0.004*"people" + 0.004
*"villain" + 0.004*"despicable" + 0.003*"horror" + 0.003*"first" + 0.003*"minion"'),
(9, '0.007*"batman" + 0.006*"knight" + 0.005*"character" + 0.005*"gotham" + 0.005*"wa
yne" + 0.004*"monster" + 0.004*"people" + 0.004*"rise" + 0.004*"bruce" + 0.003*"selin
a"'), (10, '0.005*"character" + 0.005*"horror" + 0.004*"score" + 0.004*"first" + 0.00
3*"mission" + 0.003*"chapter" + 0.003*"weird" + 0.003*"fiction" + 0.003*"another" +
0.003*"certain"'), (11, '0.006*"first" + 0.005*"fresh" + 0.005*"character" + 0.005*"w
oman" + 0.004*"holmes" + 0.004*"opening" + 0.004*"steve" + 0.003*"people" + 0.003*"sp
ielberg" + 0.003*"surprise"'), (12, '0.010*"dream" + 0.008*"point" + 0.006*"sister" +
0.006*"character" + 0.005*"child" + 0.005*"poehler" + 0.005*"fisher" + 0.004*"first"
+ 0.004*"second" + 0.004*"comedy"'), (13, '0.007*"ahmed" + 0.007*"kinley" + 0.006*"in
terpreter" + 0.005*"first" + 0.005*"ritchie" + 0.005*"annihilation" + 0.004*"covenan
t" + 0.004*"character" + 0.004*"taliban" + 0.004*"afghan"'), (14, '0.010*"character"
+ 0.009*"pitch" + 0.009*"black" + 0.006*"loretta" + 0.006*"riddick" + 0.006*"bullock"
+ 0.005*"planet" + 0.005*"comedy" + 0.005*"tatum" + 0.005*"diesel"'), (15, '0.006*"po
ehler" + 0.006*"annihilation" + 0.005*"sister" + 0.004*"screen" + 0.004*"moment" + 0.
004*"friend" + 0.004*"playing" + 0.004*"night" + 0.004*"maura" + 0.004*"inside"'), (1
6, '0.025*"blonde" + 0.018*"legally" + 0.014*"school" + 0.014*"witherspoon" + 0.012
*"warner" + 0.012*"harvard" + 0.006*"played" + 0.006*"ahmed" + 0.006*"luketic" + 0.00
6*"comedy"'), (17, '0.005*"first" + 0.005*"dream" + 0.005*"character" + 0.004*"story"
+ 0.004*"despicable" + 0.004*"little" + 0.004*"kinley" + 0.004*"year" + 0.004*"ahmed"
+ 0.003*"world"'), (18, '0.007*"people" + 0.006*"character" + 0.005*"moment" + 0.004
*"party" + 0.004*"sister" + 0.004*"story" + 0.004*"group" + 0.003*"first" + 0.003*"fr
esh" + 0.003*"world"')]
{'2 topics and 10 words': 0.2887710660386871, '4 topics and 10 words': 0.284540054001
85534, '19 topics and 10 words': 0.31361824462762233}
```