DSA_202101_ 17: Clustering Assignment

Use Case: Using different clustering algorithms

Supervised By: Prof. Arya.

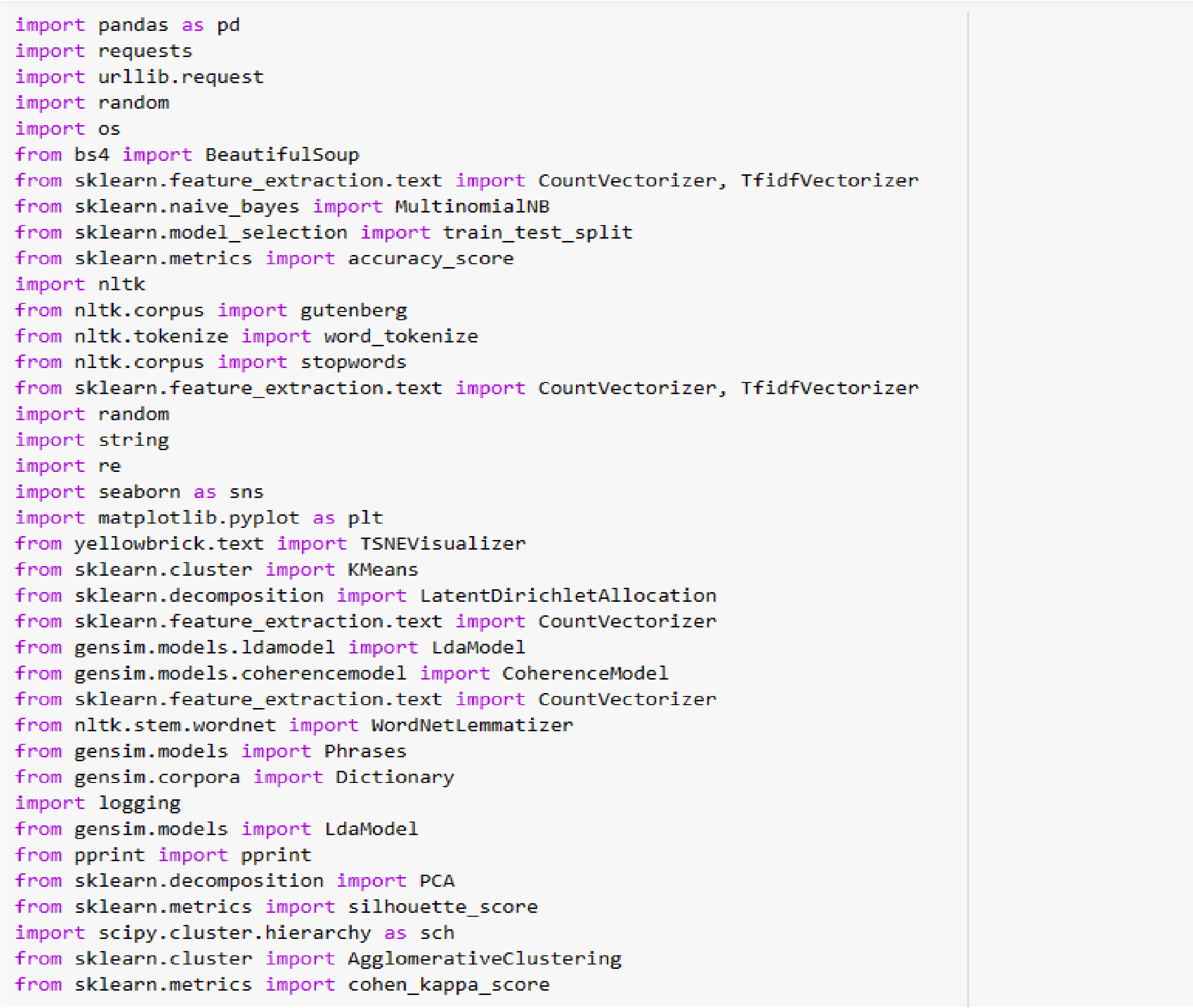
TA. Pouya Khodaee

Team Members

Last Name, First Name
Abdelazim, Sara
Abdullah, Alaa
Fahem, Noha
Mousa, Naser

Importing Libraries

We first imported the libraries we need for the project.



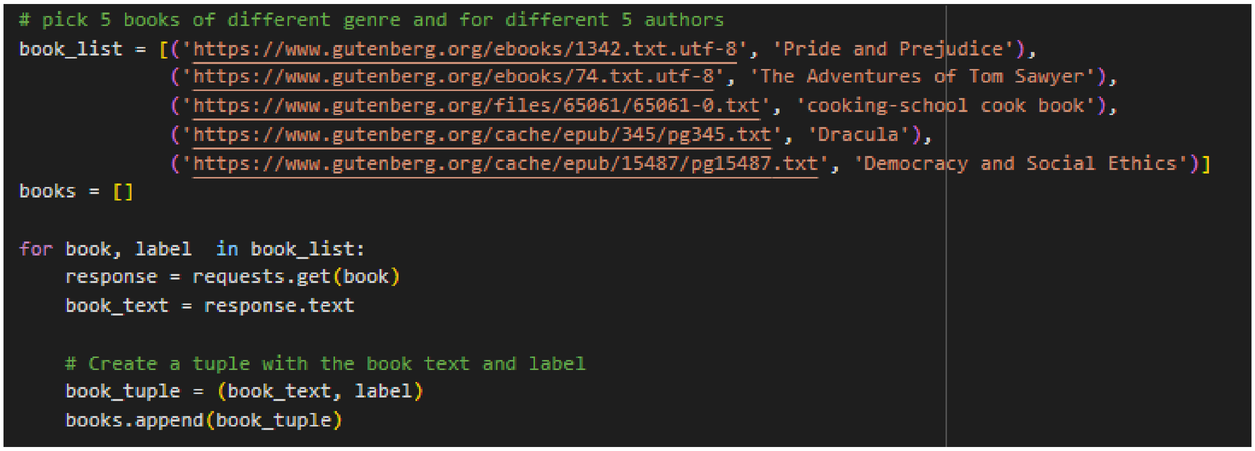
```
import pandas as pd
import requests
import urllib.request
import random
import os
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import nltk
from nltk.corpus import gutenberg
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
import random
import string
import re
import seaborn as sns
import matplotlib.pyplot as plt
from yellowbrick.text import TSNEVisualizer
from sklearn.cluster import KMeans
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
from gensim.models.ldamodel import LdaModel
from gensim.models.coherencemodel import CoherenceModel
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Phrases
from gensim.corpora import Dictionary
import logging
from gensim.models import LdaModel
from pprint import pprint
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import cohen_kappa_score
```

GuetnBerg books choice

1. Pride and Prejudice
2. The Adventures of Tom Sawyer
3. cooking-school cook-book
4. Dracula
5. Democracy and Social Ethics

Loading Data

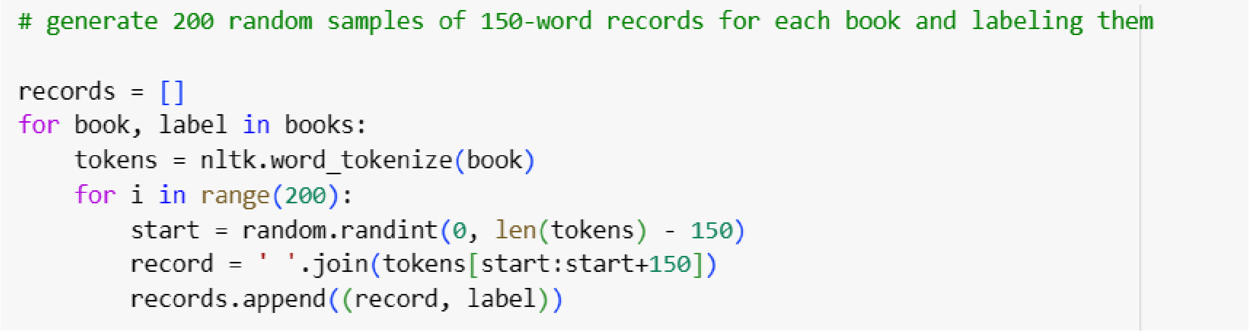
Then, we loaded our data as shown in this block of code. The list of books is specified in a separate list of tuples called book_list. Each tuple in book_list contains a URL that points to the text file of a book and a string label that identifies the book. Then there is books which is a list containing the text and the book title as the for loop iterates through each tuple in book_list, downloads the book text from the URL using the requests library, and creates a new tuple with the book text and label. This tuple is then added to the books list.



```
# pick 5 books of different genre and for different 5 authors
book_list = [('https://www.gutenberg.org/ebooks/1342.txt.utf-8', 'Pride and Prejudice'),
             ('https://www.gutenberg.org/ebooks/74.txt.utf-8', 'The Adventures of Tom Sawyer'),
             ('https://www.gutenberg.org/files/65061/65061-0.txt', 'cooking-school cook book'),
             ('https://www.gutenberg.org/cache/epub/345/pg345.txt', 'Dracula'),
             ('https://www.gutenberg.org/cache/epub/15487/pg15487.txt', 'Democracy and Social Ethics')]
books = []

for book, label  in book_list:
    response = requests.get(book)
    book_text = response.text

    # Create a tuple with the book text and label
    book_tuple = (book_text, label)
    books.append(book_tuple)
```

Creating partitions of data

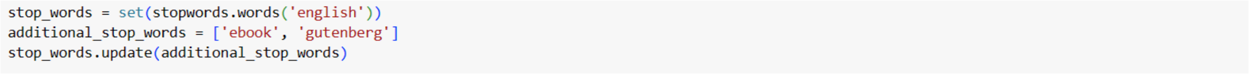
As required to generate 200 random samples of 150 words for each book. We iterated through the books as we know from the snippet above it's a tuple containing book text and its title. The code first tokenizes the text of each book using the nltk.word_tokenize() function from the Natural Language Toolkit (NLTK) library. It then selects 200 random samples from each book, each consisting of 150 consecutive tokens. For each iteration, the code creates a string by joining the tokens with spaces, and creates a tuple with the string and the label of the corresponding book. This tuple is then added to the records.



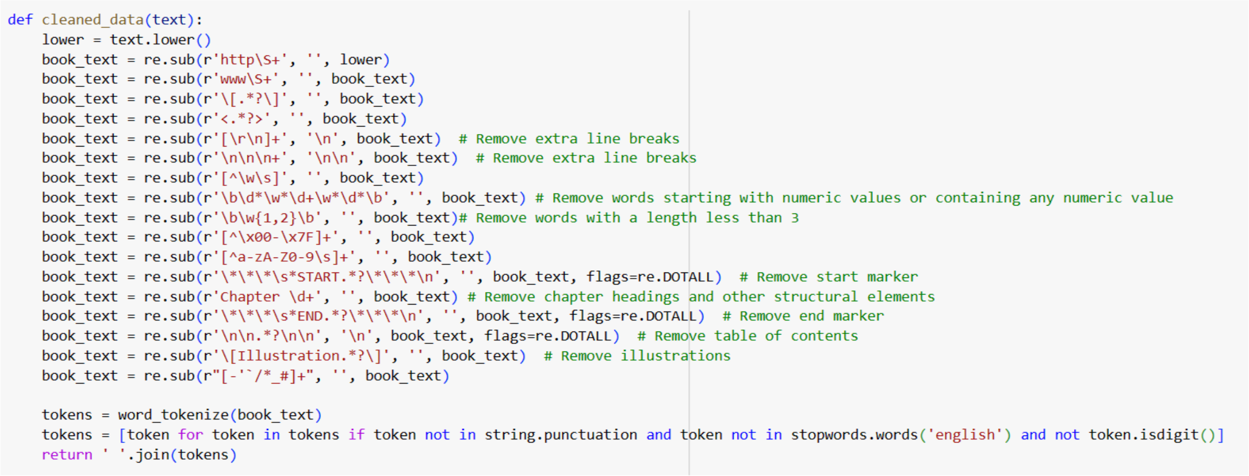
```
# generate 200 random samples of 150-word records for each book and labeling them

records = []
for book, label in books:
    tokens = nltk.word_tokenize(book)
    for i in range(200):
        start = random.randint(0, len(tokens) - 150)
        record = ' '.join(tokens[start:start+150])
        records.append((record, label))
```

Data Preprocessing and Cleaning

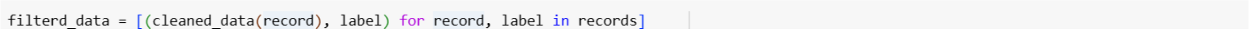
We added some additional words to the stopwords list that we thought we can safely remove as they are not useful for our case. Then there is our cleaned_data function that takes in a single parameter, text, which is a string containing text data. The function performs a series of text cleaning steps on the input text and returns the cleaned text as a string. Then we used a list comprehension that iterates through each tuple in records and applies the cleaned_data() function to the text sample to create a new list of tuples called filtered_data.



```
stop_words = set(stopwords.words('english'))
additional_stop_words = ['ebook', 'gutenberg']
stop_words.update(additional_stop_words)
```



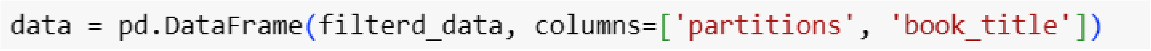
```
def cleaned_data(text):
    lower = text.lower()
    book_text = re.sub(r'http\S+', '', lower)
    book_text = re.sub(r'www\S+', '', book_text)
    book_text = re.sub(r'\[.*?\]', '', book_text)
    book_text = re.sub(r'<.*?>', '', book_text)
    book_text = re.sub(r'[\r\n]+', '\n', book_text)  # Remove extra line breaks
    book_text = re.sub(r'\n\n\n+', '\n\n', book_text)  # Remove extra line breaks
    book_text = re.sub(r'[^\w\s]', '', book_text)
    book_text = re.sub(r'\b\d*\w*\d+\w*\d*\b', '', book_text) # Remove words starting with numeric values or containing any numeric value
    book_text = re.sub(r'\b\w{1,2}\b', '', book_text)# Remove words with a length less than 3
    book_text = re.sub(r'[^\x00-\x7F]+', '', book_text)
    book_text = re.sub(r'[^a-zA-Z0-9\s]+', '', book_text)
    book_text = re.sub(r'\*\*\*\s*START.*?\*\*\*\n', '', book_text, flags=re.DOTALL)  # Remove start marker
    book_text = re.sub(r'Chapter \d+', '', book_text) # Remove chapter headings and other structural elements
    book_text = re.sub(r'\*\*\*\s*END.*?\*\*\*\n', '', book_text, flags=re.DOTALL)  # Remove end marker
    book_text = re.sub(r'\n\n.*?\n\n', '\n', book_text, flags=re.DOTALL)  # Remove table of contents
    book_text = re.sub(r'\[Illustration.*?\]', '', book_text)  # Remove illustrations
    book_text = re.sub(r"[-'`/*_#]+", '', book_text)

    tokens = word_tokenize(book_text)
    tokens = [token for token in tokens if token not in string.punctuation and token not in stopwords.words('english') and not token.isdigit()]
    return ' '.join(tokens)
```



```
filterd_data = [(cleaned_data(record), label) for record, label in records]
```

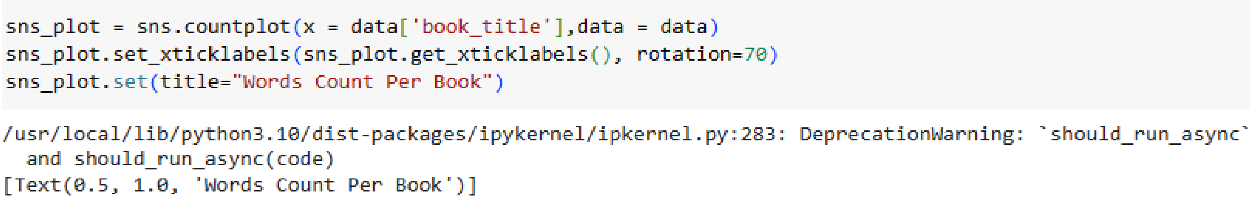
Converting to dataframe

We converted our data to dataframe as it's easier to work with in cases such as analysis, manipulation, filtering, and etc. And we give proper names to columns instead just the default 0 and 1.



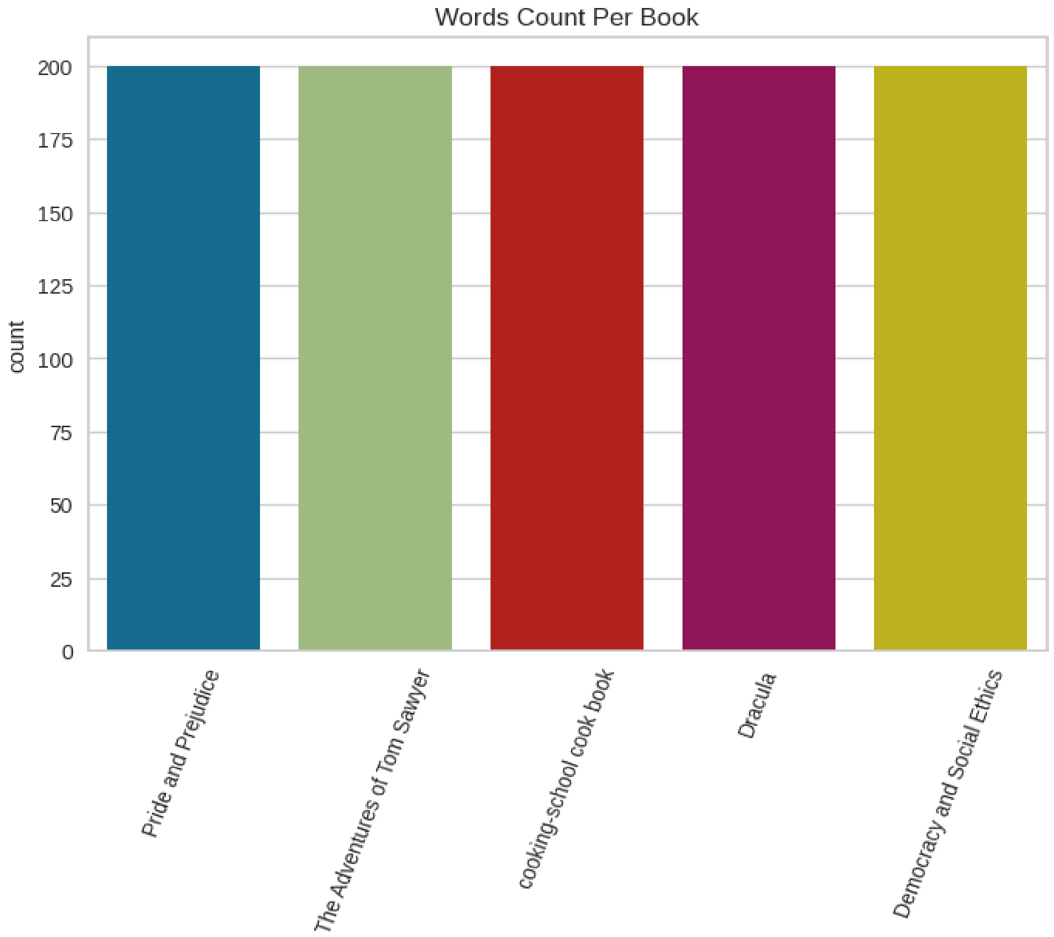
```
data = pd.DataFrame(filterd_data, columns=['partitions', 'book_title'])
```

This is our count plot showing exactly equal 5 books with 200 document.



```
sns_plot = sns.countplot(x = data['book_title'],data = data)
sns_plot.set_xticklabels(sns_plot.get_xticklabels(), rotation=70)
sns_plot.set(title="Words Count Per Book")
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async`
  and should_run_async(code)
[Text(0.5, 1.0, 'Words Count Per Book')]
```



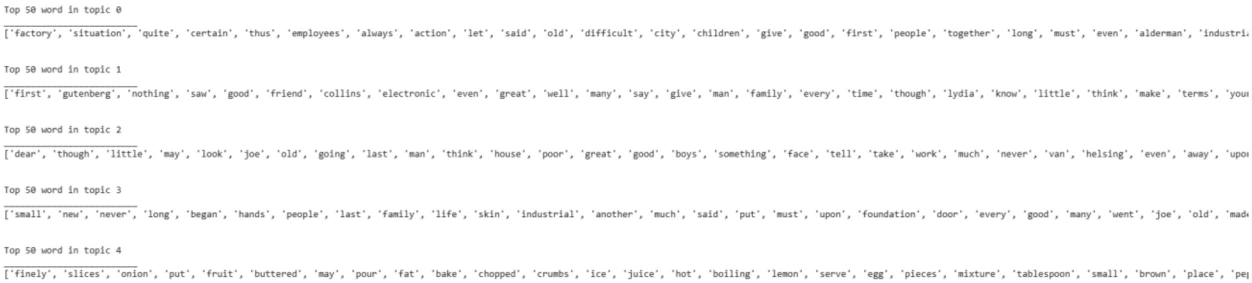
Word Embedding

Or as some call it transformation. We used Bow and TF-IDF just as the other assignment. The new transformation we applied here was LDA from LatentDirichletAllocation library.

We printed the top 50 words that have the highest probability of belonging to the topic or you could think of it the most important keywords that define each topic.

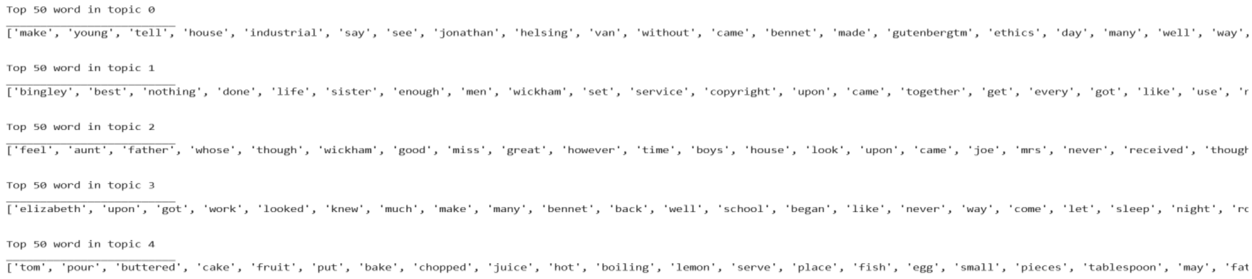
For example, looking at the top 50 words in topic 0, we see words such as "said", "together", "always", "people". From my point of view I would say this topic is for a book maybe related to personal experience or about a story, but it has the name "Darcy", so we all know it’s Pride and Prejudice.

And like that, we can get an overall view of data using LDA.



```
Top 50 word in topic 0
____________________________
['factory', 'situation', 'quite', 'certain', 'thus', 'employees', 'always', 'action', 'let', 'said', 'old', 'difficult', 'city', 'children', 'give', 'good', 'first', 'people', 'together', 'long', 'must', 'even', 'alderman', 'industri

Top 50 word in topic 1
____________________________
['first', 'gutenberg', 'nothing', 'saw', 'good', 'friend', 'collins', 'electronic', 'even', 'great', 'well', 'many', 'say', 'give', 'man', 'family', 'every', 'time', 'though', 'lydia', 'know', 'little', 'think', 'make', 'terms', 'you

Top 50 word in topic 2
____________________________
['dear', 'though', 'little', 'may', 'look', 'joe', 'old', 'going', 'last', 'man', 'think', 'house', 'poor', 'great', 'good', 'boys', 'something', 'face', 'tell', 'take', 'work', 'much', 'never', 'van', 'helsing', 'even', 'away', 'upo

Top 50 word in topic 3
____________________________
['small', 'new', 'never', 'long', 'began', 'hands', 'people', 'last', 'family', 'life', 'skin', 'industrial', 'another', 'much', 'said', 'put', 'must', 'upon', 'foundation', 'door', 'every', 'good', 'many', 'went', 'joe', 'old', 'mad

Top 50 word in topic 4
____________________________
['finely', 'slices', 'onion', 'put', 'fruit', 'buttered', 'may', 'pour', 'fat', 'bake', 'chopped', 'crumbs', 'ice', 'juice', 'hot', 'boiling', 'lemon', 'serve', 'egg', 'pieces', 'mixture', 'tablespoon', 'small', 'brown', 'place', 'pe
```

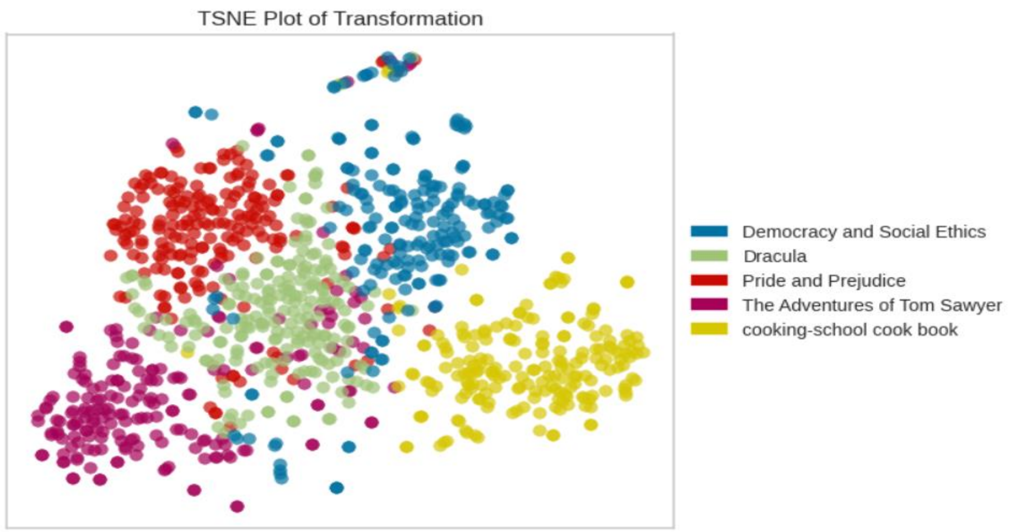
This was the output for TF-IDF



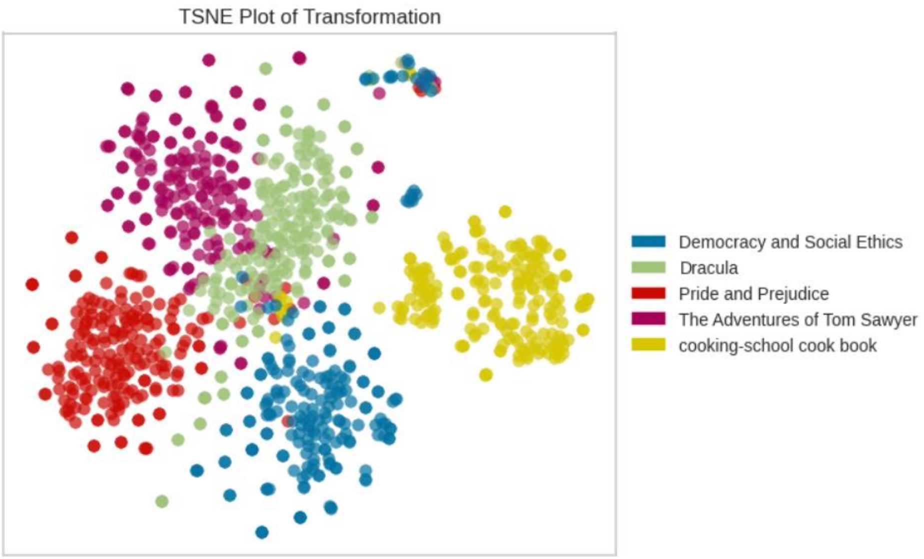
```
Top 50 word in topic 0
____________________________
['make', 'young', 'tell', 'house', 'industrial', 'say', 'see', 'jonathan', 'helsing', 'van', 'without', 'came', 'bennet', 'made', 'gutenbergtm', 'ethics', 'day', 'many', 'well', 'way',

Top 50 word in topic 1
____________________________
['bingley', 'best', 'nothing', 'done', 'life', 'sister', 'enough', 'men', 'wickham', 'set', 'service', 'copyright', 'upon', 'came', 'together', 'get', 'every', 'got', 'like', 'use', 'r

Top 50 word in topic 2
____________________________
['feel', 'aunt', 'father', 'whose', 'though', 'wickham', 'good', 'miss', 'great', 'however', 'time', 'boys', 'house', 'look', 'upon', 'came', 'joe', 'mrs', 'never', 'received', 'though

Top 50 word in topic 3
____________________________
['elizabeth', 'upon', 'got', 'work', 'looked', 'knew', 'much', 'make', 'many', 'bennet', 'back', 'well', 'school', 'began', 'like', 'never', 'way', 'come', 'let', 'sleep', 'night', 'ro

Top 50 word in topic 4
____________________________
['tom', 'pour', 'buttered', 'cake', 'fruit', 'put', 'bake', 'chopped', 'juice', 'hot', 'boiling', 'lemon', 'serve', 'place', 'fish', 'egg', 'small', 'pieces', 'tablespoon', 'may', 'fat
```

We created a function for tsne Visualization and applied it on both transformations

On BoW



On TF-IDF

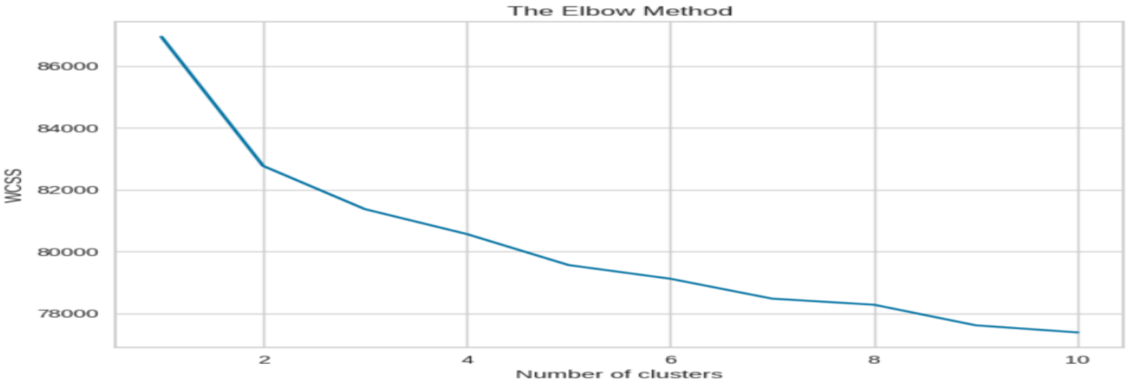


Clustering

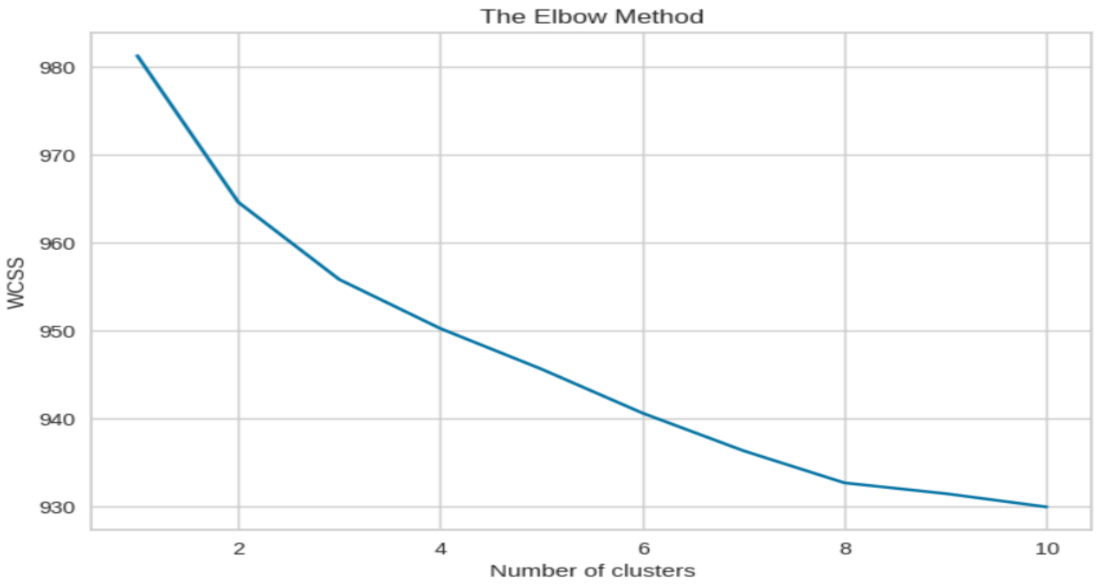
We used Kmeans, Agglomerative Clustering, and EM

First, **Kmeans** we used elbow method to figure out which number of clusters we should go with.

For bow data, we went with 6 clusters as the figure suggested.

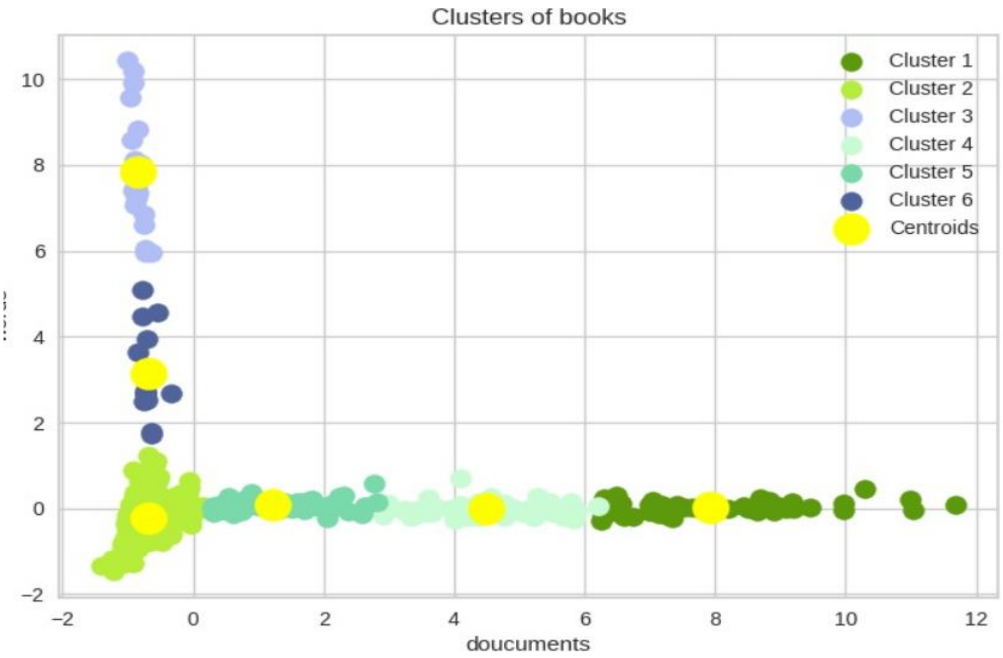


And for TF-IDF, we went with 8 clusters

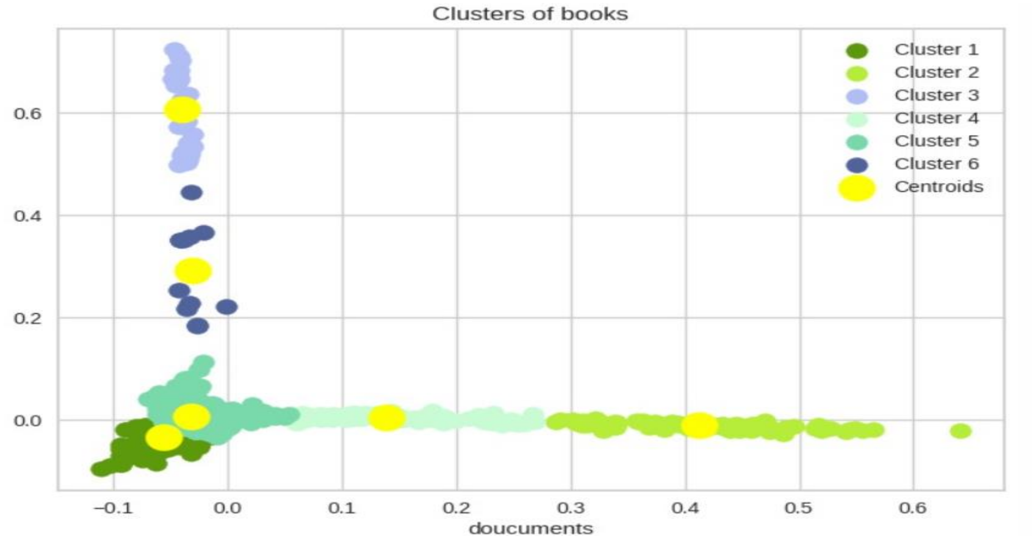


But before we trained K-means on our data, we applied PCA on both our representations of data as both BOW and TF-IDF are like a sparse matrix which is highly dimensional that could result in overfitting the model. So, PCA seemed like the proper action to take to lower the dimensions.

Visualizing the clusters for BOW

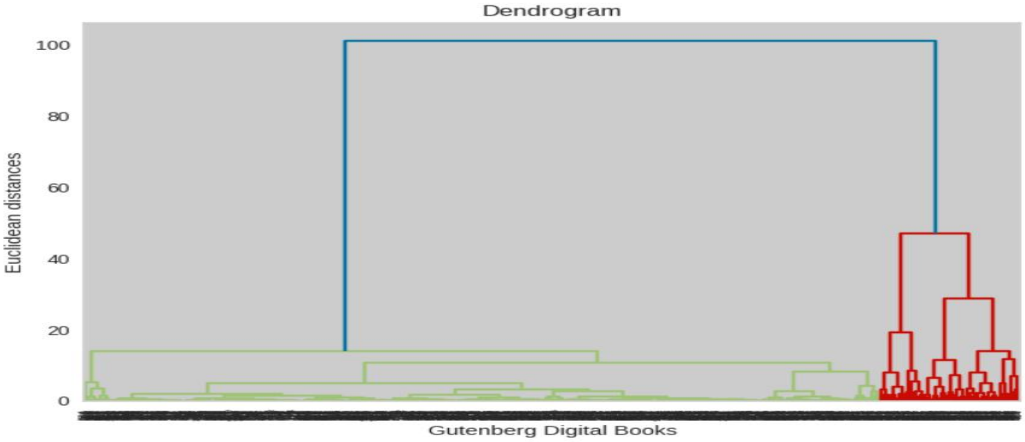


Visualizing the clusters for TF-IDF

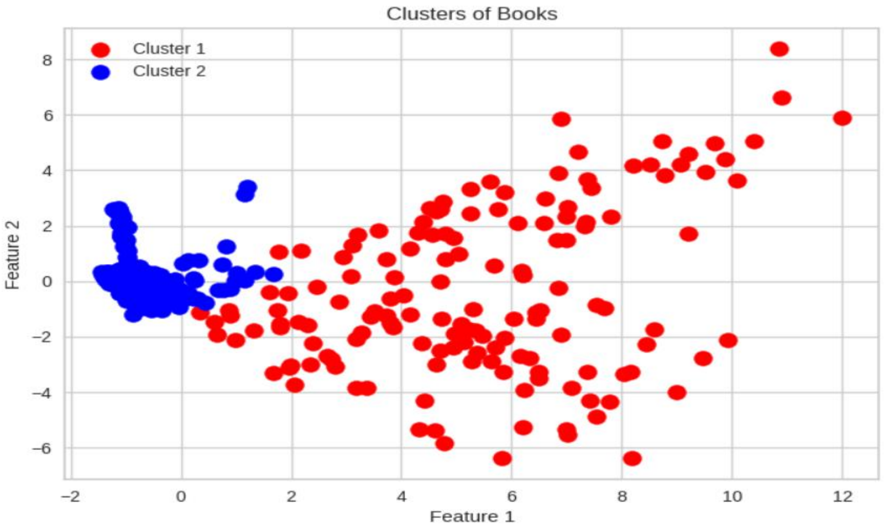


Second, with **Agglomerative Hierarchical Clustering,** we used dendrogram to find a proper number of clusters to go with.

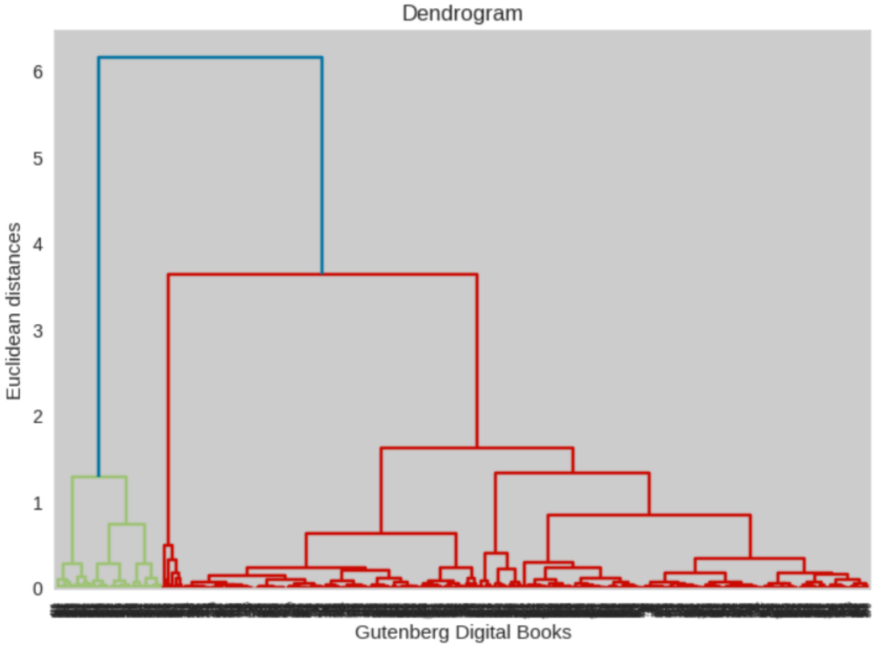
→For BOW, based on the graph and our intuition to analyze it we went with 2 clusters.



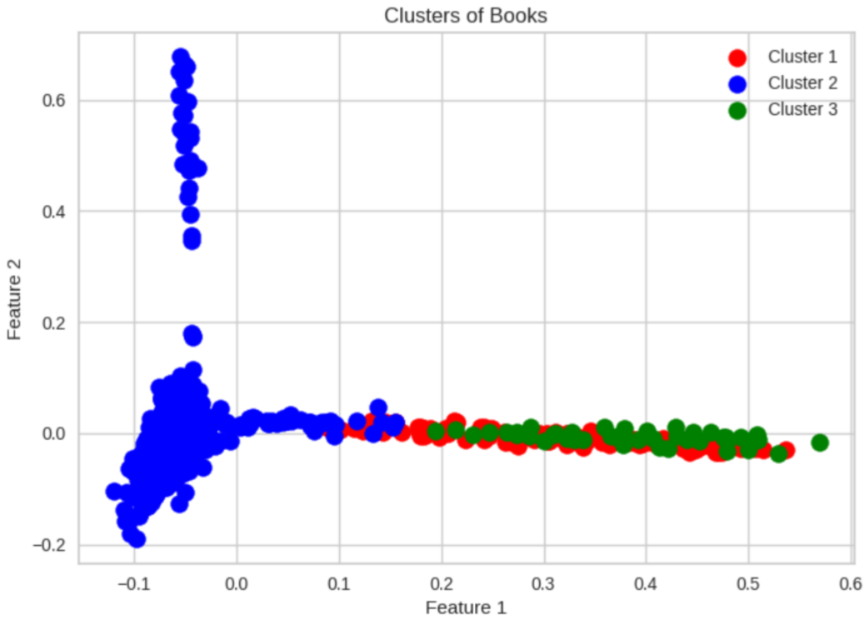
And its figure was like this



→For TF-IDF, we went with 3 clusters.

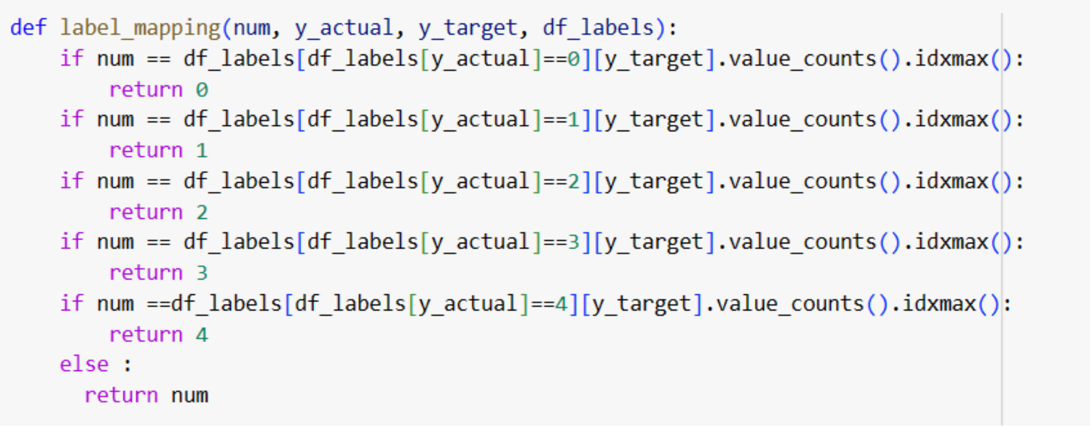


The visual was as shown



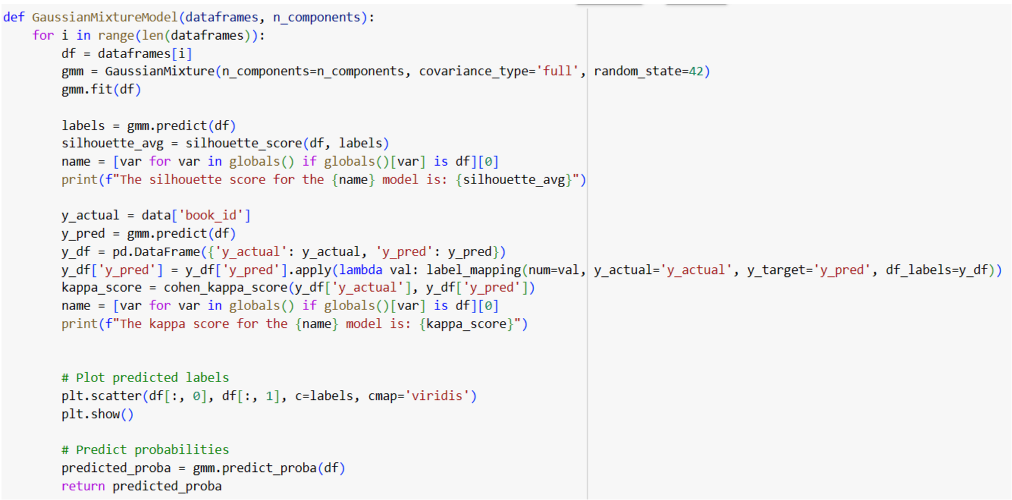
And lastly, we used **EM clustering**

We first used this function to map the labels based on the maximum occurrences for each category.



```
def label_mapping(num, y_actual, y_target, df_labels):
    if num == df_labels[df_labels[y_actual]==0][y_target].value_counts().idxmax():
        return 0
    if num == df_labels[df_labels[y_actual]==1][y_target].value_counts().idxmax():
        return 1
    if num == df_labels[df_labels[y_actual]==2][y_target].value_counts().idxmax():
        return 2
    if num == df_labels[df_labels[y_actual]==3][y_target].value_counts().idxmax():
        return 3
    if num ==df_labels[df_labels[y_actual]==4][y_target].value_counts().idxmax():
        return 4
    else :
      return num
```

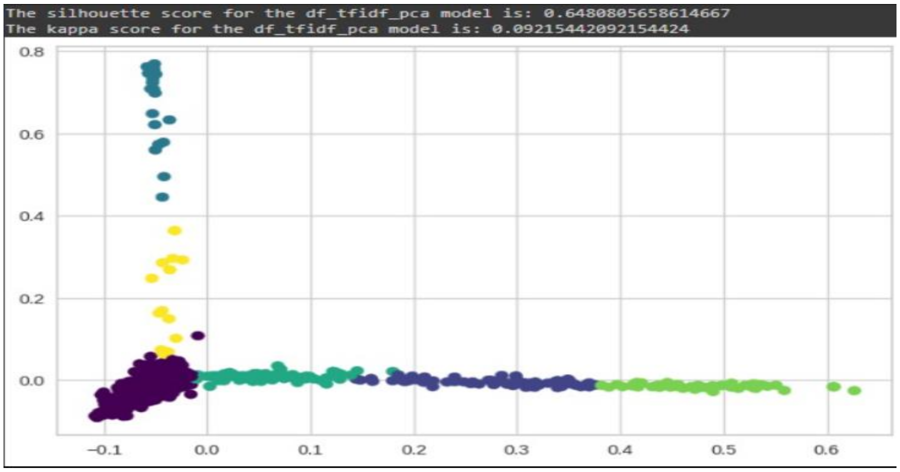
Then applied the Gaussian



```
def GaussianMixtureModel(dataframes, n_components):
    for i in range(len(dataframes)):
        df = dataframes[i]
        gmm = GaussianMixture(n_components=n_components, covariance_type='full', random_state=42)
        gmm.fit(df)

        labels = gmm.predict(df)
        silhouette_avg = silhouette_score(df, labels)
        name = [var for var in globals() if globals()[var] is df][0]
        print(f"The silhouette score for the {name} model is: {silhouette_avg}")

        y_actual = data['book_id']
        y_pred = gmm.predict(df)
        y_df = pd.DataFrame({'y_actual': y_actual, 'y_pred': y_pred})
        y_df['y_pred'] = y_df['y_pred'].apply(lambda val: label_mapping(num=val, y_actual='y_actual', y_target='y_pred', df_labels=y_df))
        kappa_score = cohen_kappa_score(y_df['y_actual'], y_df['y_pred'])
        name = [var for var in globals() if globals()[var] is df][0]
        print(f"The kappa score for the {name} model is: {kappa_score}")


        # Plot predicted labels
        plt.scatter(df[:, 0], df[:, 1], c=labels, cmap='viridis')
        plt.show()

        # Predict probabilities
        predicted_proba = gmm.predict_proba(df)
        return predicted_proba
```

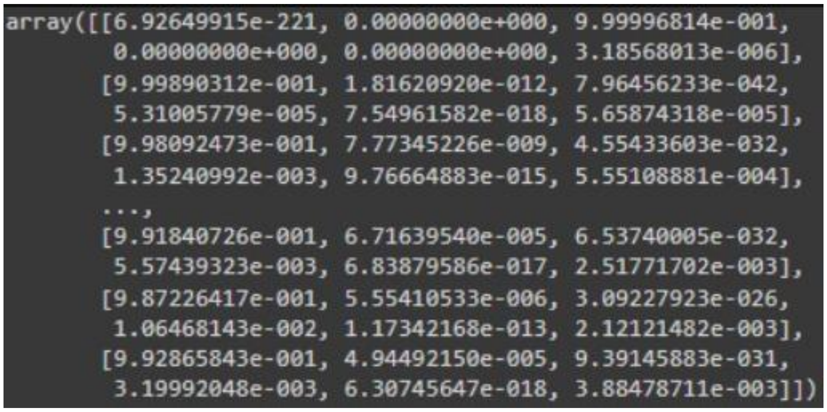
We first experimented with 6 clusters

→For TF-IDF, graph looked like this



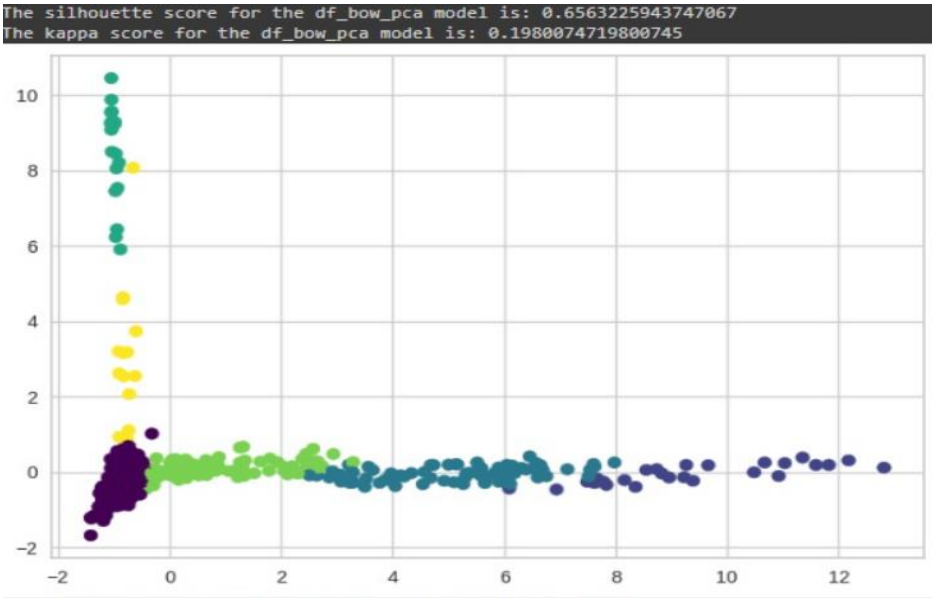
```
The silhouette score for the df_tfidf_pca model is: 0.6480805658614667
The kappa score for the df_tfidf_pca model is: 0.09215442092154424
```

Probabilities of each point indicating its weight



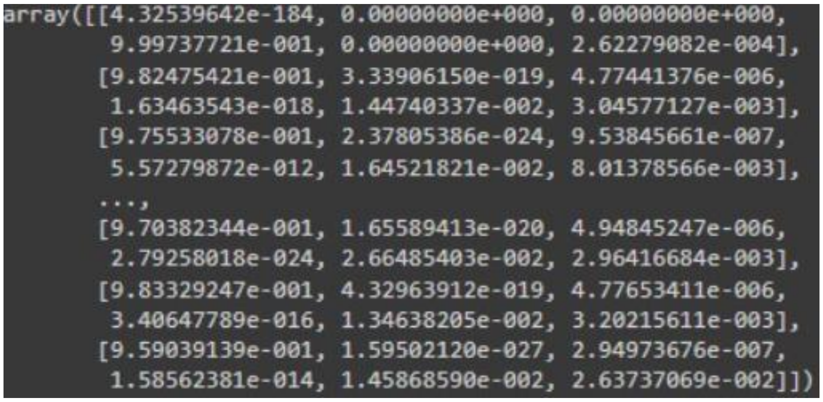
```
array([[6.92649915e-221, 0.00000000e+000, 9.99996814e-001,
         0.00000000e+000, 0.00000000e+000, 3.18568013e-006],
        [9.99890312e-001, 1.81620920e-012, 7.96456233e-042,
         5.31005779e-005, 7.54961582e-018, 5.65874318e-005],
        [9.98092473e-001, 7.77345226e-009, 4.55433603e-032,
         1.35240992e-003, 9.76664883e-015, 5.55108881e-004],
        ...,
        [9.91840726e-001, 6.71639540e-005, 6.53740005e-032,
         5.57439323e-003, 6.83879586e-017, 2.51771702e-003],
        [9.87226417e-001, 5.55410533e-006, 3.09227923e-026,
         1.06468143e-002, 1.17342168e-013, 2.12121482e-003],
        [9.92865843e-001, 4.94492150e-005, 9.39145883e-031,
         3.19992048e-003, 6.30745647e-018, 3.88478711e-003]])
```

→And for BOW, the graph was as shown below



```
The silhouette score for the df_bow_pca model is: 0.6563225943747067
The kappa score for the df_bow_pca model is: 0.1980074719800745
```

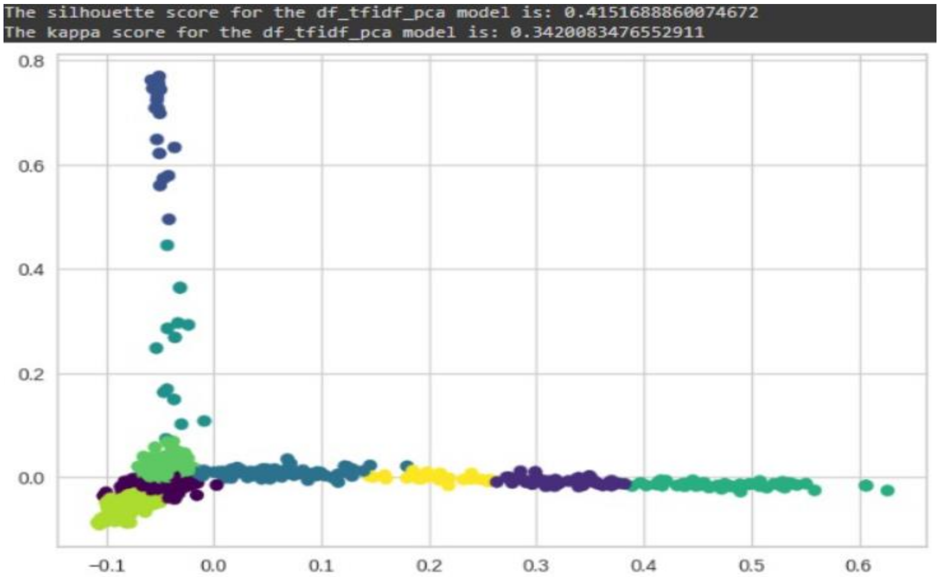
Probabilities



```
array([[4.32539642e-184, 0.00000000e+000, 0.00000000e+000,
         9.99737721e-001, 0.00000000e+000, 2.62279082e-004],
        [9.82475421e-001, 3.33906150e-019, 4.77441376e-006,
         1.63463543e-018, 1.44740337e-002, 3.04577127e-003],
        [9.75533078e-001, 2.37805386e-024, 9.53845661e-007,
         5.57279872e-012, 1.64521821e-002, 8.01378566e-003],
        ...,
        [9.70382344e-001, 1.65589413e-020, 4.94845247e-006,
         2.79258018e-024, 2.66485403e-002, 2.96416684e-003],
        [9.83329247e-001, 4.32963912e-019, 4.77653411e-006,
         3.40647789e-016, 1.34638205e-002, 3.20215611e-003],
        [9.59039139e-001, 1.59502120e-027, 2.94973676e-007,
         1.58562381e-014, 1.45868590e-002, 2.63737069e-002]])
```

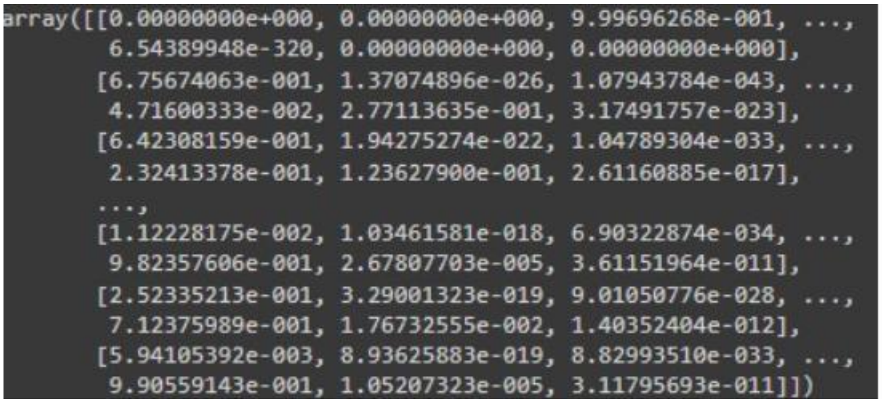
Then, we tried with 9 clusters

→For TF-IDF



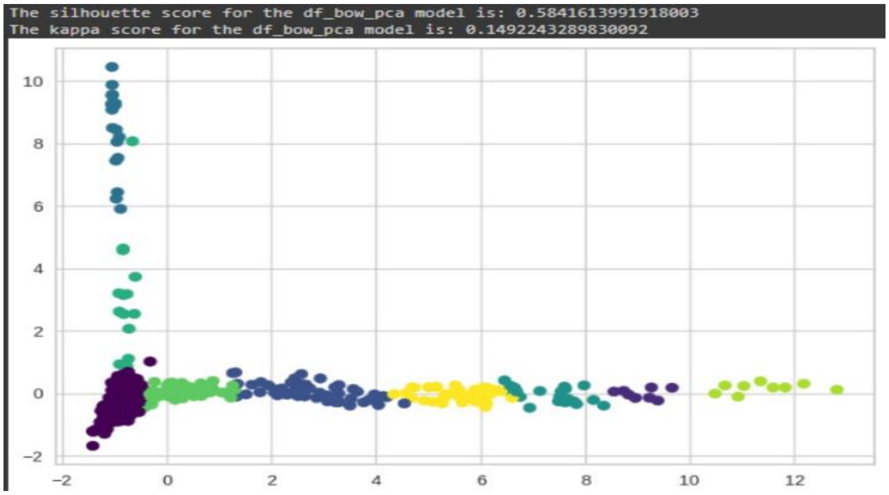
```
The silhouette score for the df_tfidf_pca model is: 0.4151688860074672
The kappa score for the df_tfidf_pca model is: 0.3420083476552911
```

Probabilities



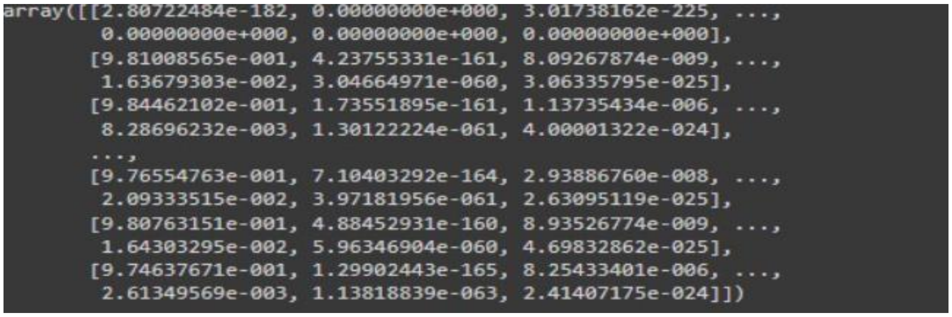
```
array([[0.00000000e+000, 0.00000000e+000, 9.99696268e-001, ...,
        6.54389948e-320, 0.00000000e+000, 0.00000000e+000],
       [6.75674063e-001, 1.37074896e-026, 1.07943784e-043, ...,
        4.71600333e-002, 2.77113635e-001, 3.17491757e-023],
       [6.42308159e-001, 1.94275274e-022, 1.04789304e-033, ...,
        2.32413378e-001, 1.23627900e-001, 2.61160885e-017],
       ...,
       [1.12228175e-002, 1.03461581e-018, 6.90322874e-034, ...,
        9.82357606e-001, 2.67807703e-005, 3.61151964e-011],
       [2.52335213e-001, 3.29001323e-019, 9.01050776e-028, ...,
        7.12375989e-001, 1.76732555e-002, 1.40352404e-012],
       [5.94105392e-003, 8.93625883e-019, 8.82993510e-033, ...,
        9.90559143e-001, 1.05207323e-005, 3.11795693e-011]])
```

→For BOW



```
The silhouette score for the df_bow_pca model is: 0.5841613991918003
The kappa score for the df_bow_pca model is: 0.1492243289830092
```

Probabilities

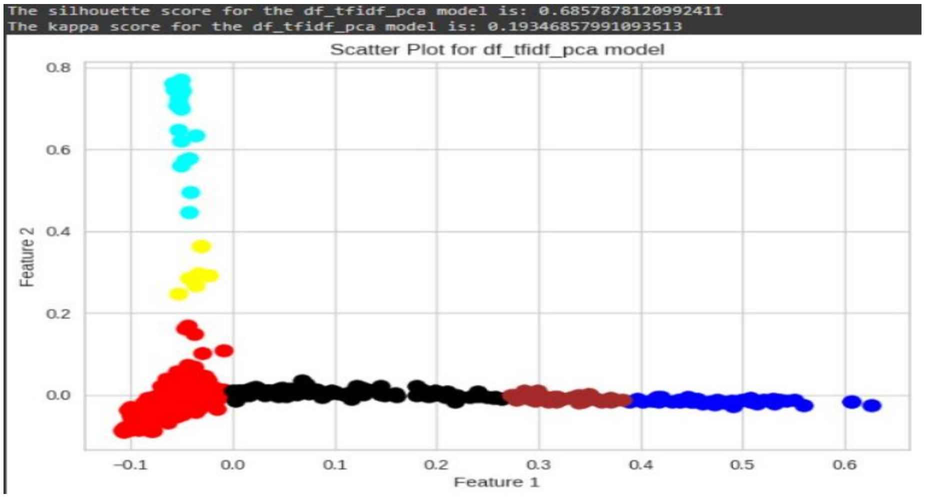


```
array([[2.80722484e-182, 0.00000000e+000, 3.01738162e-225, ...,
         0.00000000e+000, 0.00000000e+000, 0.00000000e+000],
        [9.81008565e-001, 4.23755331e-161, 8.09267874e-009, ...,
         1.63679303e-002, 3.04664971e-060, 3.06335795e-025],
        [9.84462102e-001, 1.73551895e-161, 1.13735434e-006, ...,
         8.28696232e-003, 1.30122224e-061, 4.00001322e-024],
        ...,
        [9.76554763e-001, 7.10403292e-164, 2.93886760e-008, ...,
         2.09333515e-002, 3.97181956e-061, 2.63095119e-025],
        [9.80763151e-001, 4.88452931e-160, 8.93526774e-009, ...,
         1.64303295e-002, 5.96346904e-060, 4.69832862e-025],
        [9.74637671e-001, 1.29902443e-165, 8.25433401e-006, ...,
         2.61349569e-003, 1.13818839e-063, 2.41407175e-024]])
```

Kappa on Agglomerative Clustering

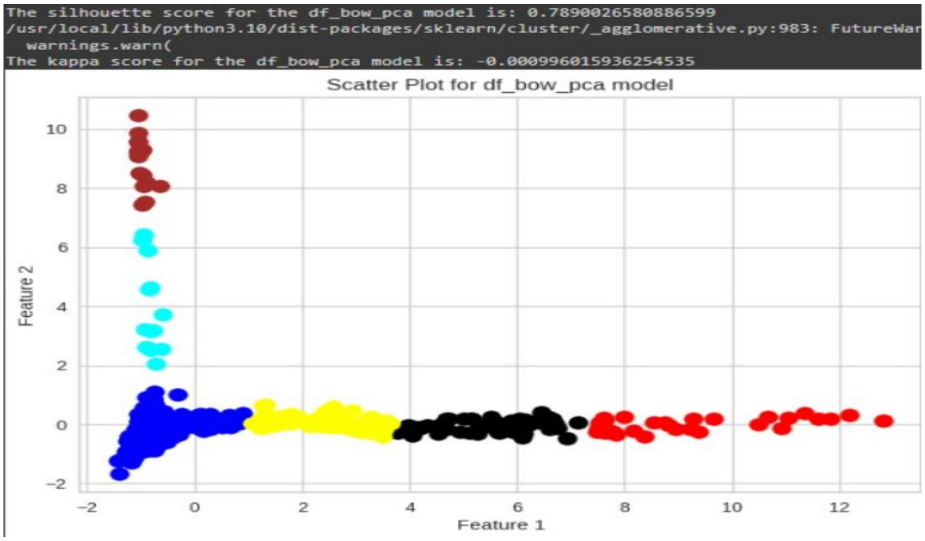
For 6 clusters

→In case of TF-IDF



```
The silhouette score for the df_tfidf_pca model is: 0.6857878120992411
The kappa score for the df_tfidf_pca model is: 0.19346857991093513
```

→In case of BOW

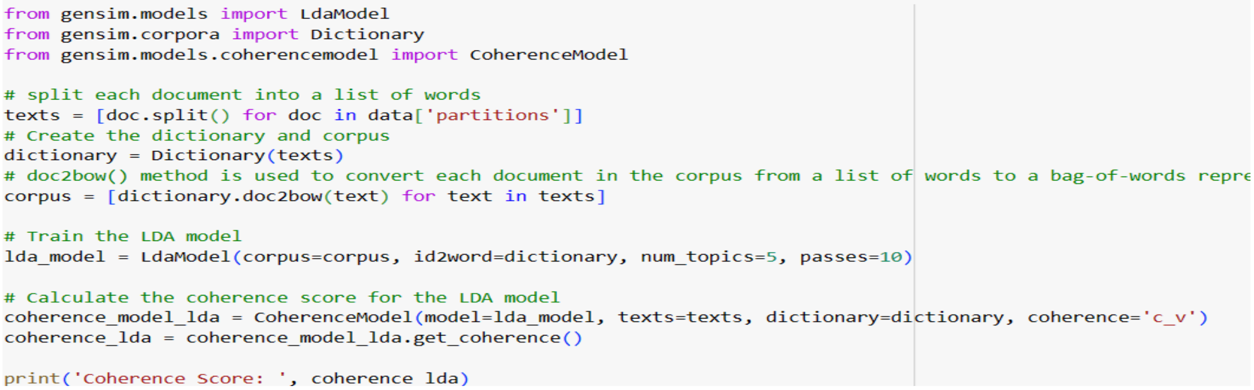


```
The silhouette score for the df_bow_pca model is: 0.7890026580886599
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_agglomerative.py:983: FutureWar
  warnings.warn(
The kappa score for the df_bow_pca model is: -0.000996015936254535
```

Evaluation

We coherence, silhouette, and kappa

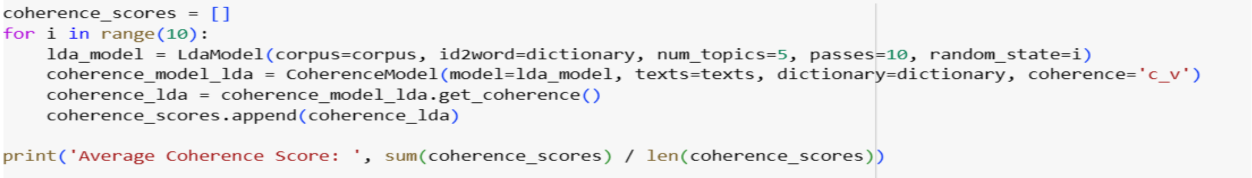
The snippet of code below is the one we used for calculating coherence. I think everything is interpreted in the block.



```
from gensim.models import LdaModel
from gensim.corpora import Dictionary
from gensim.models.coherencemodel import CoherenceModel

# split each document into a list of words
texts = [doc.split() for doc in data['partitions']]
# Create the dictionary and corpus
dictionary = Dictionary(texts)
# doc2bow() method is used to convert each document in the corpus from a list of words to a bag-of-words repre
corpus = [dictionary.doc2bow(text) for text in texts]

# Train the LDA model
lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=5, passes=10)

# Calculate the coherence score for the LDA model
coherence_model_lda = CoherenceModel(model=lda_model, texts=texts, dictionary=dictionary, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()

print('Coherence Score: ', coherence_lda)
```

Coherence Score: 0.47078331125807604 It is a moderate score implying that the topics in the LDA model may be somewhat interpretable and distinct, but we could use some improvement.

And as we noticed every time we run this snippet, the score changes. So, to obtain a more stable score we calculated the average via this code.



```
coherence_scores = []
for i in range(10):
    lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=5, passes=10, random_state=i)
    coherence_model_lda = CoherenceModel(model=lda_model, texts=texts, dictionary=dictionary, coherence='c_v')
    coherence_lda = coherence_model_lda.get_coherence()
    coherence_scores.append(coherence_lda)

print('Average Coherence Score: ', sum(coherence_scores) / len(coherence_scores))
```

And the score was Average Coherence Score: 0.5030249447801141 **Improvement!**

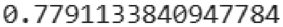
Error Analysis

We used the silhouette method to see if the number of clusters we used was good or not.

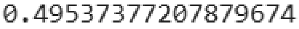
**In case of K-means clustering:

→For BOW 6 clusters, the score was Silhouette score: 0.7525550577541916

Which is a good result and indicates that the clustering is effective. This means that the clusters are well-separated and capture meaningful patterns in the data.

- ➔ When we decreased the number by just one cluster, it was 

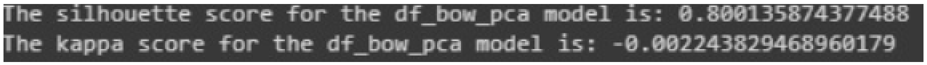
```
0.7791133840947784
```

→For TF-IDF 8 clusters, the score was, 

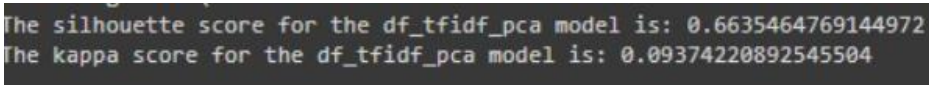
```
0.49537377207879674
```

So, we can conclude that the score decreases when we increase the number of clusters in our case.

Also, for Kappa with 6 clusters for both representatives

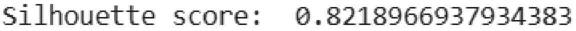


```
The silhouette score for the df_bow_pca model is: 0.800135874377488
The kappa score for the df_bow_pca model is: -0.002243829468960179
```



```
The silhouette score for the df_tfidf_pca model is: 0.6635464769144972
The kappa score for the df_tfidf_pca model is: 0.09374220892545504
```

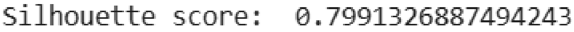
In case of Agglomerative Clustering:

→For BOW 2 clusters, the score was 

```
Silhouette score:  0.8218966937934383
```

.

Which surprisingly indicates they are more similar than we think.

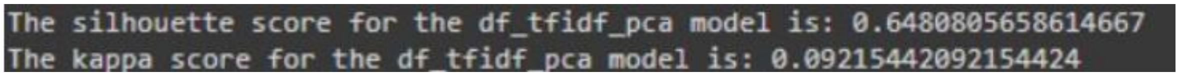
→For TF-IDF 3 clusters, the score was 

```
Silhouette score:  0.7991326887494243
```

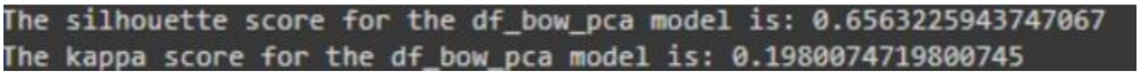
Which is still surprisingly good result!

**In case of Agglomerative Clustering

With 6 clusters, scores were for both transformations.

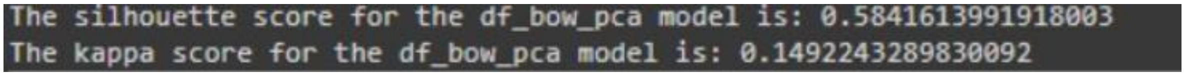


```
The silhouette score for the df_tfidf_pca model is: 0.6480805658614667
The kappa score for the df_tfidf_pca model is: 0.09215442092154424
```

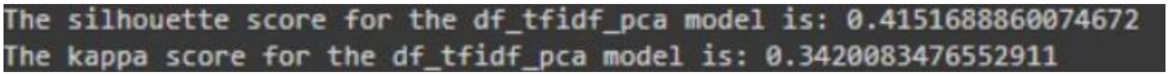


```
The silhouette score for the df_bow_pca model is: 0.6563225943747067
The kappa score for the df_bow_pca model is: 0.1980074719800745
```

With 9 clusters, scores were



```
The silhouette score for the df_bow_pca model is: 0.5841613991918003
The kappa score for the df_bow_pca model is: 0.1492243289830092
```



```
The silhouette score for the df_tfidf_pca model is: 0.4151688860074672
The kappa score for the df_tfidf_pca model is: 0.3420083476552911
```

You could notice the scores of Kappa got higher with an increased number of clusters and the silhouette just had a slight decrease which is a still valid result.

Our source code:

https://colab.research.google.com/drive/1NOnbxuAZ5bAf5i3ulHcTHLknzbkl8SvQ?usp=sharing

Resources we looked up:

Topic Modeling with Latent Dirichlet Allocation (LDA) | by Sandeep Panchal | Analytics Vidhya | Medium

Evaluate Topic Models: Latent Dirichlet Allocation (LDA) | by Shashank Kapadia | Towards Data Science