Report Assignment 2

Text Clustering

uOttawa

Faculté de génie
Faculty of Engineering

DTI5125[EG] Data Science Applications 20219

Group (5)

Introduction

Text clustering is the application of cluster analysis to text-based documents. It uses machine learning and natural language processing (NLP) to understand and categorize unstructured, textual data. In our assignment we have books and partitions then we will cluster these partitions and combine the smeller partitions in the same cluster.

1- Choose data

The dataset used in this project are five different samples of Gutenberg digital books, which are all of five different genres and of five different authors, the books are:

1. Relativity: The Special and General Theory, Bookshelves: Physics

2. Domestic Animals, Bookshelves: Animal

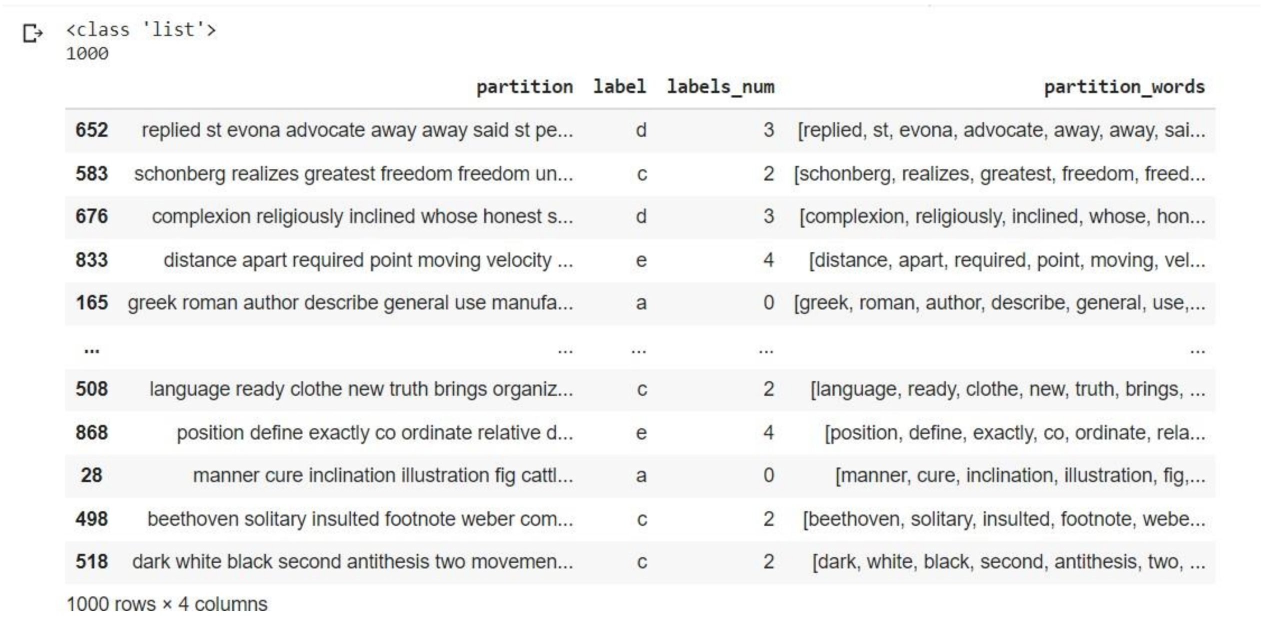
3. Murder in the Gunroom, Bookshelves: Crime fiction

4. Concerning the Spiritual in Art, Bookshelves: Art

5. A Book About Lawyers, Books in British Law

2- Preprocessing and Data Cleansing

At the beginning of the data preprocessing process, we transformed the data to lower case, then we used regular excretion to remove punctuation and special characters, after that we used tokenization, lemmatization and we removed stop words, we took a random sample from each book which consist of 200 partition and each partition has 150 word.



```
<class 'list'>
1000
```

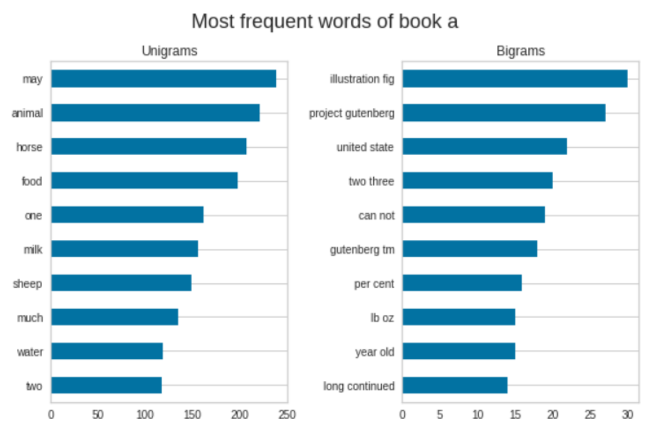
	partition	label	labels_num	partition_words
652	replied st evona advocate away away said st pe...	d	3	[replied, st, evona, advocate, away, away, sai...
583	schonberg realizes greatest freedom freedom un...	c	2	[schonberg, realizes, greatest, freedom, freed...
676	complexion religiously inclined whose honest s...	d	3	[complexion, religiously, inclined, whose, hon...
833	distance apart required point moving velocity ...	e	4	[distance, apart, required, point, moving, vel...
165	greek roman author describe general use manufa...	a	0	[greek, roman, author, describe, general, use,...
...	...	...	...	...
508	language ready clothe new truth brings organiz...	c	2	[language, ready, clothe, new, truth, brings, ...
868	position define exactly co ordinate relative d...	e	4	[position, define, exactly, co, ordinate, rela...
28	manner cure inclination illustration fig cattl...	a	0	[manner, cure, inclination, illustration, fig,...
498	beethoven solitary insulted footnote weber com...	c	2	[beethoven, solitary, insulted, footnote, webe...
518	dark white black second antithesis two movemen...	c	2	[dark, white, black, second, antithesis, two, ...

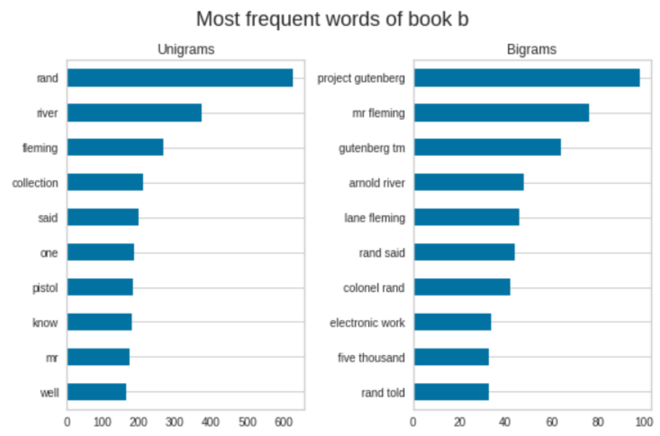
1000 rows × 4 columns

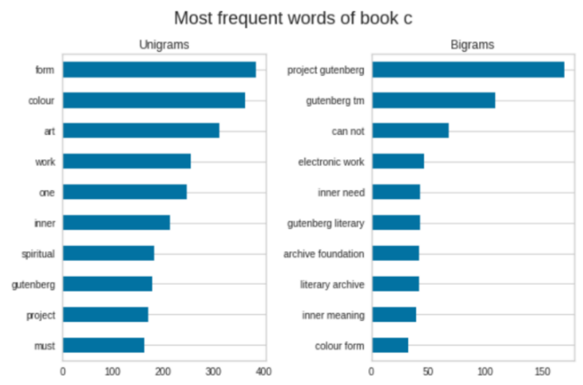
Fig (1.1)

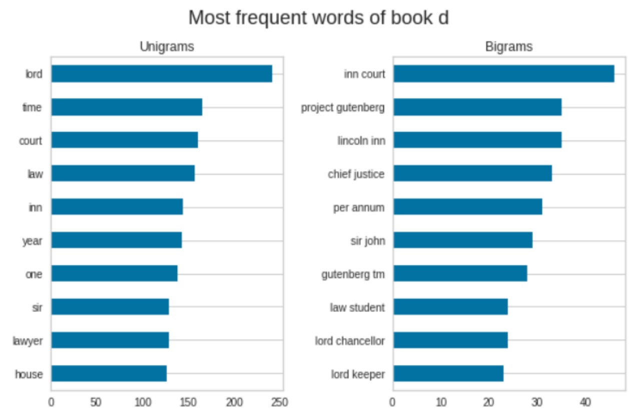
3- Data Exploration

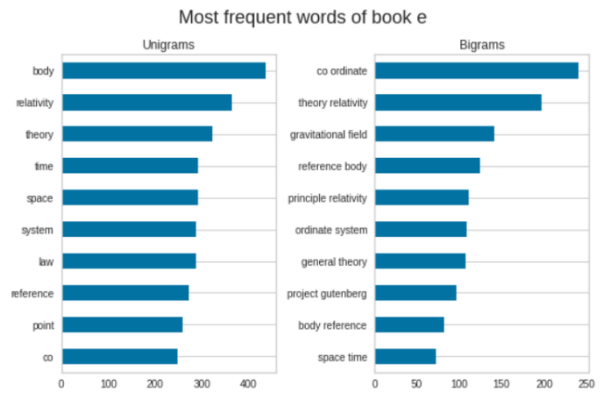
➢ Word Frequency:









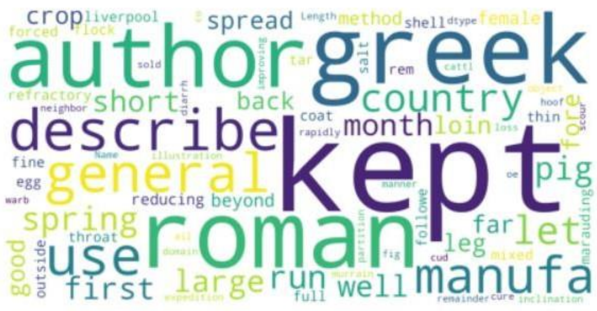


Word Cloud :

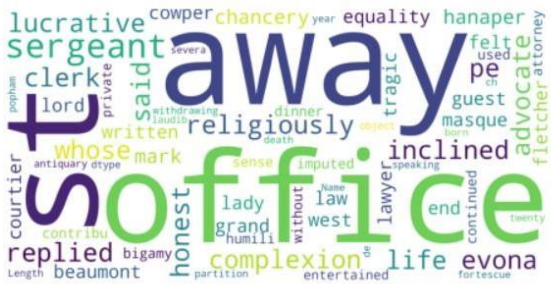
Fig (1.2)

- Word cloud:

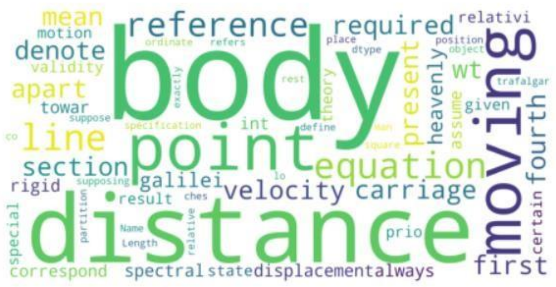
WordCloud of book a



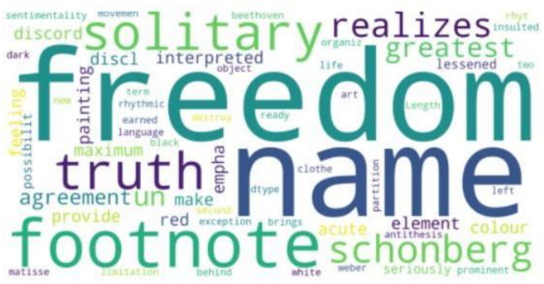
WordCloud of book d



WordCloud of book e



WordCloud of book c



WordCloud of book b

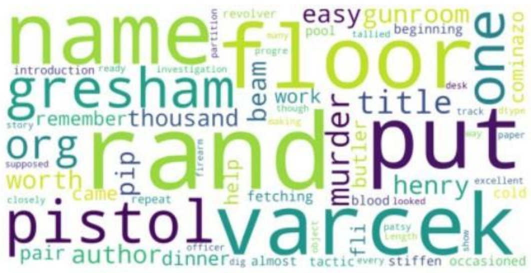
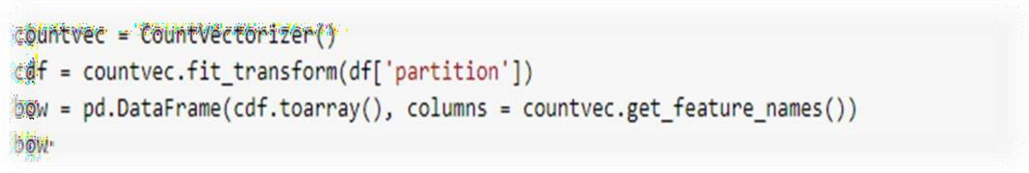


Fig (1.3)

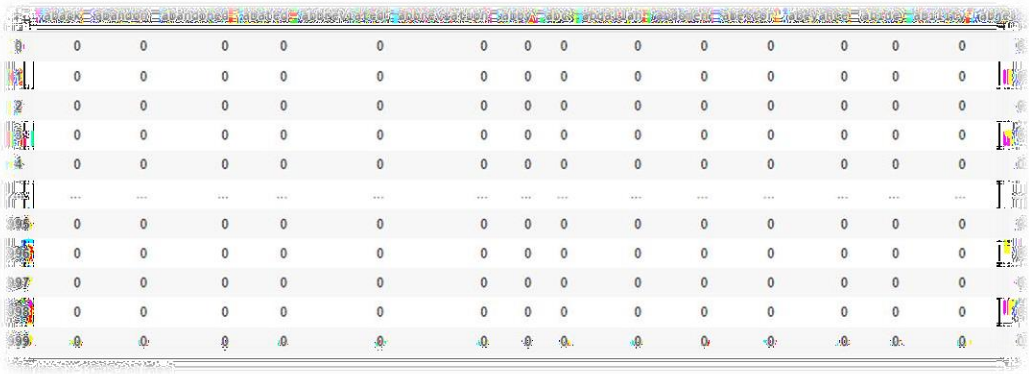
4- Feature Engineering

Machine Learning Algorithms cannot work with a raw text directly, the text must be converted into numbers specifically, vectors of numbers.
In language processing, the vectors x is derived from textual data, in order to reflect various linguistic properties of the text. To achieve this transformation there are some popular methods for that: BOW, TF_IDF, WORD2VEC and LDA.

- BOW
  In this representation, a text is represented as a bag of its words. This achieved by counting how many times each word appears, this process is often referred to as vectorization. By other words the occurrence of each word is used as a feature here:



```
countvec = CountVectorizer()
cdf = countvec.fit_transform(df['partition'])
bow = pd.DataFrame(cdf.toarray(), columns = countvec.get_feature_names())
bow
```

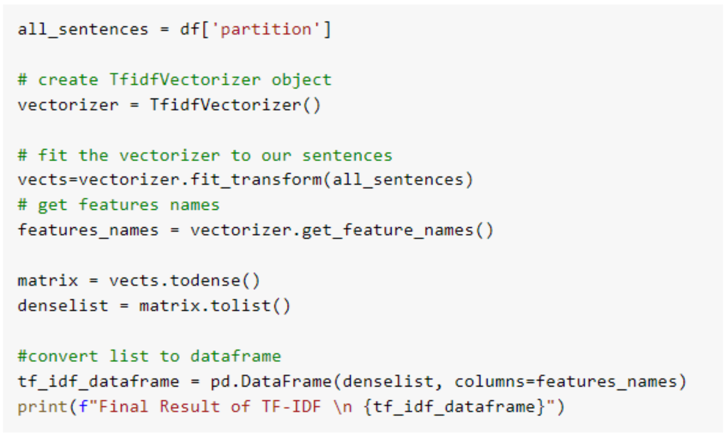


	aback	abandon	abandoned	abated	abbreviated	abbreviation	abby	abc	abdallah	abdomen	abetted	abeyance	abide	ability
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0	0	0	0	0	0	0	0	0	0	0	0	0	0
996	0	0	0	0	0	0	0	0	0	0	0	0	0	0
997	0	0	0	0	0	0	0	0	0	0	0	0	0	0
998	0	0	0	0	0	0	0	0	0	0	0	0	0	0
999	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig (4.1)

➢ TF-IDF

IT is (term frequency-inverse document frequency) , It reflects how important a word is to a document in a collection or corpus , The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, it is one of the most popular term-weighting schemes.



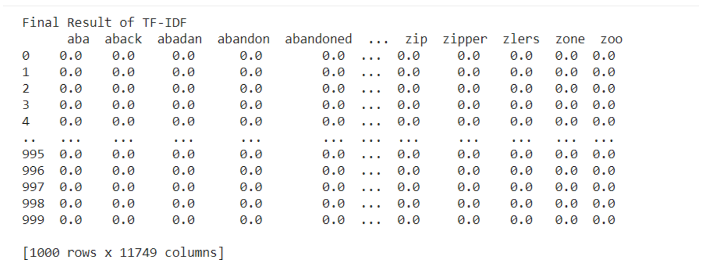
```
all_sentences = df['partition']

# create TfidfVectorizer object
vectorizer = TfidfVectorizer()

# fit the vectorizer to our sentences
vects=vectorizer.fit_transform(all_sentences)
# get features names
features_names = vectorizer.get_feature_names()

matrix = vects.todense()
denselist = matrix.tolist()

#convert list to dataframe
tf_idf_dataframe = pd.DataFrame(denselist, columns=features_names)
print(f"Final Result of TF-IDF \n {tf_idf_dataframe}")
```



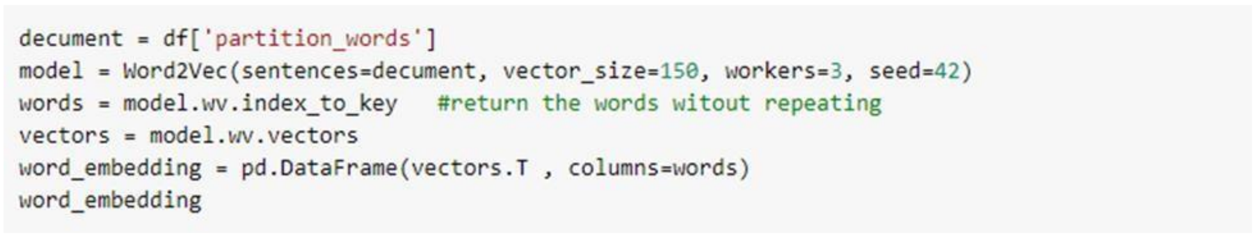
```
Final Result of TF-IDF
      aba  aback  abadan  abandon  abandoned  ...  zip  zipper  zlers  zone  zoo
0     0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
1     0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
2     0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
3     0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
4     0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
..    ...    ...     ...      ...        ...  ...  ...     ...    ...   ...  ...
995   0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
996   0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
997   0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
998   0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0
999   0.0    0.0     0.0      0.0        0.0  ...  0.0     0.0    0.0   0.0  0.0

[1000 rows x 11749 columns]
```

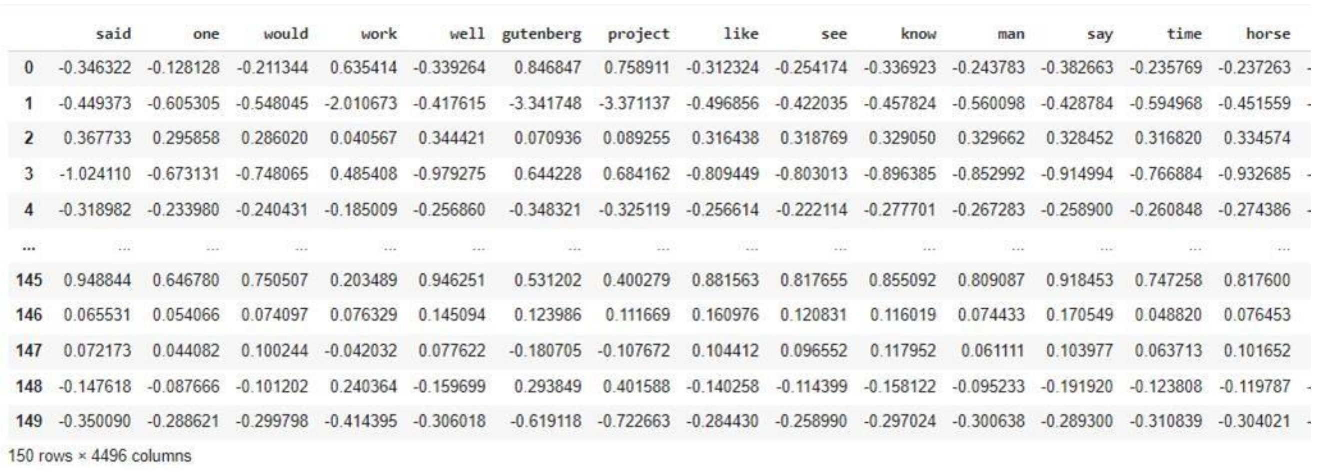
Fig (4.2)

➢ WORD2VEC

The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence, from its name it convert each word into a vector, vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.



```
decument = df['partition_words']
model = Word2Vec(sentences=decument, vector_size=150, workers=3, seed=42)
words = model.wv.index_to_key   #return the words witout repeating
vectors = model.wv.vectors
word_embedding = pd.DataFrame(vectors.T , columns=words)
word_embedding
```



	said	one	would	work	well	gutenberg	project	like	see	know	man	say	time	horse
0	-0.346322	-0.128128	-0.211344	0.635414	-0.339264	0.846847	0.758911	-0.312324	-0.254174	-0.336923	-0.243783	-0.382663	-0.235769	-0.237263
1	-0.449373	-0.605305	-0.548045	-2.010673	-0.417615	-3.341748	-3.371137	-0.496856	-0.422035	-0.457824	-0.560098	-0.428784	-0.594968	-0.451559
2	0.367733	0.295858	0.286020	0.040567	0.344421	0.070936	0.089255	0.316438	0.318769	0.329050	0.329662	0.328452	0.316820	0.334574
3	-1.024110	-0.673131	-0.748065	0.485408	-0.979275	0.644228	0.684162	-0.809449	-0.803013	-0.896385	-0.852992	-0.914994	-0.766884	-0.932685
4	-0.318982	-0.233980	-0.240431	-0.185009	-0.256860	-0.348321	-0.325119	-0.256614	-0.222114	-0.277701	-0.267283	-0.258900	-0.260848	-0.274386
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
145	0.948844	0.646780	0.750507	0.203489	0.946251	0.531202	0.400279	0.881563	0.817655	0.855092	0.809087	0.918453	0.747258	0.817600
146	0.065531	0.054066	0.074097	0.076329	0.145094	0.123986	0.111669	0.160976	0.120831	0.116019	0.074433	0.170549	0.048820	0.076453
147	0.072173	0.044082	0.100244	-0.042032	0.077622	-0.180705	-0.107672	0.104412	0.096552	0.117952	0.061111	0.103977	0.063713	0.101652
148	-0.147618	-0.087666	-0.101202	0.240364	-0.159699	0.293849	0.401588	-0.140258	-0.114399	-0.158122	-0.095233	-0.191920	-0.123808	-0.119787
149	-0.350090	-0.288621	-0.299798	-0.414395	-0.306018	-0.619118	-0.722663	-0.284430	-0.258990	-0.297024	-0.300638	-0.289300	-0.310839	-0.304021

150 rows × 4496 columns

fig (4.3)

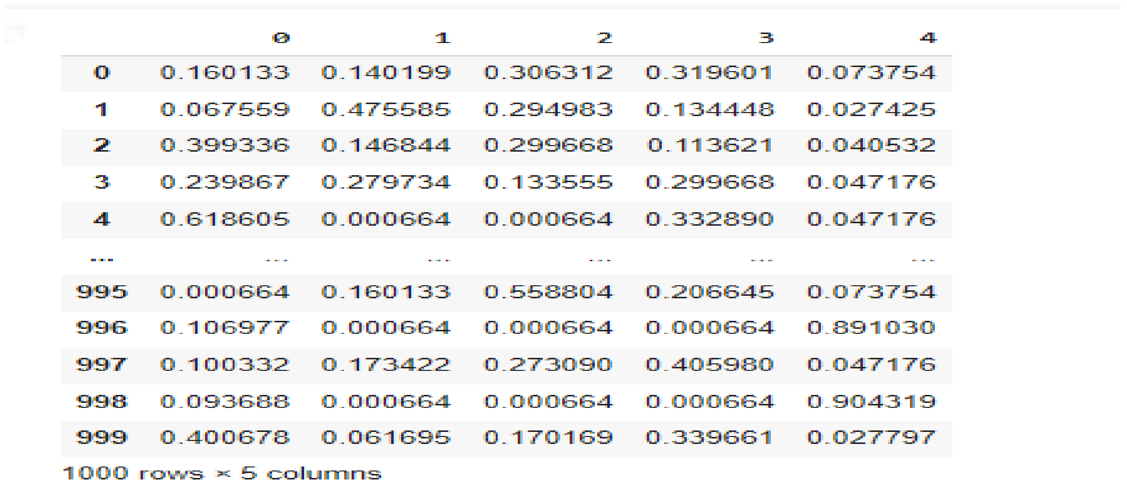
Unlike BOW, WORD2VEC represents words by vectors, which capture various characteristics of that word with regard to the overall text. Like semantic relationship of the word.

➢ LDA

The LDA model is a generative statistical model of a collection of documents. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words.

```
countvec_lda = CountVectorizer()
cdf_lda = countvec_lda.fit_transform(df['partition'])
```

```
ldamodel = lda.LDA(n_topics=5, n_iter=10, random_state=1)
ldamodel.fit(cdf_lda)
lda_representation = ldamodel.doc_topic_
```



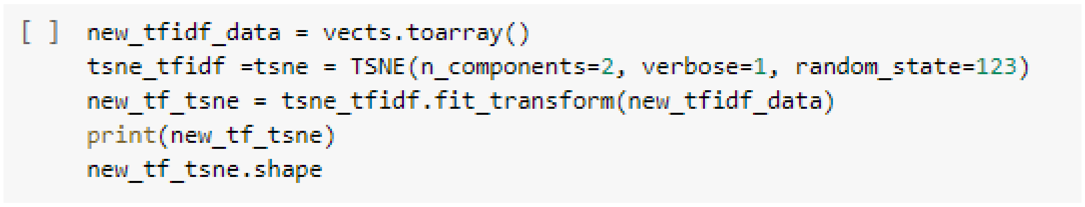
	0	1	2	3	4
0	0.160133	0.140199	0.306312	0.319601	0.073754
1	0.067559	0.475585	0.294983	0.134448	0.027425
2	0.399336	0.146844	0.299668	0.113621	0.040532
3	0.239867	0.279734	0.133555	0.299668	0.047176
4	0.618605	0.000664	0.000664	0.332890	0.047176
...	...	...	...	...	...
995	0.000664	0.160133	0.558804	0.206645	0.073754
996	0.106977	0.000664	0.000664	0.000664	0.891030
997	0.100332	0.173422	0.273090	0.405980	0.047176
998	0.093688	0.000664	0.000664	0.000664	0.904319
999	0.400678	0.061695	0.170169	0.339661	0.027797

1000 rows × 5 columns

5- Topic modeling:

After data transformation Subsequently, the clustering algorithm creates clusters, that are representative of the data contained in these clusters.

We applied some of these algorithms, but first we used dimensionality reduction algorithms like (PCA and T-SNE) for reducing the dimensionality of the data so we can get a better score rather than modeling the real data. We applied PCA technique which gives us lower score over all models with wrong clustering plots, so we used T-SNE which given us a good score and more understandable figures and plots.

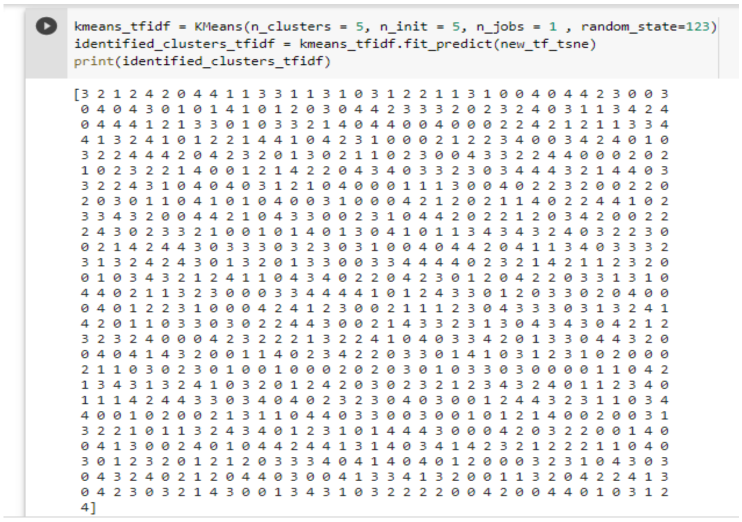


```
[ ]  new_tfidf_data = vects.toarray()
     tsne_tfidf =tsne = TSNE(n_components=2, verbose=1, random_state=123)
     new_tf_tsne = tsne_tfidf.fit_transform(new_tfidf_data)
     print(new_tf_tsne)
     new_tf_tsne.shape
```

Fig (5.1)

- **K-Means:**

  We based T-SNE result to the model with n_clusters = 5 and n_init =5, so the model uses 5 clusters and 5 centroids



```
kmeans_tfidf = KMeans(n_clusters = 5, n_init = 5, n_jobs = 1 , random_state=123)
identified_clusters_tfidf = kmeans_tfidf.fit_predict(new_tf_tsne)
print(identified_clusters_tfidf)
```

```
[3 2 1 2 4 2 0 4 4 1 1 3 3 1 1 3 1 0 3 1 2 2 1 1 3 1 0 0 4 0 4 4 2 3 0 0 3
 0 4 0 4 3 0 1 0 1 4 1 0 1 2 0 3 0 4 4 2 3 3 3 2 0 2 3 2 4 0 3 1 1 3 4 2 4
 0 4 4 4 1 2 1 3 3 0 1 0 3 3 2 1 4 0 4 4 0 0 4 0 0 0 2 2 4 2 1 2 1 1 3 3 4
 4 1 3 2 4 1 0 1 2 2 1 4 4 1 0 4 2 3 1 0 0 0 2 1 2 2 3 4 0 0 3 4 2 4 0 1 0
 3 2 2 4 4 4 2 0 4 2 3 2 0 1 3 0 2 1 1 0 2 3 0 0 4 3 3 2 2 4 4 0 0 0 2 0 2
 1 0 2 3 2 2 1 4 0 0 1 2 1 4 2 2 0 4 3 4 0 3 3 2 3 0 3 4 4 4 3 2 1 4 4 0 3
 3 2 2 4 3 1 0 4 0 4 0 3 1 2 1 0 4 0 0 0 1 1 1 3 0 0 4 0 2 2 3 2 0 0 2 2 0
 2 0 3 0 1 1 0 4 1 0 1 0 4 0 0 3 1 0 0 0 4 2 1 2 0 2 1 1 4 0 2 2 4 4 1 0 2
 3 3 4 3 2 0 0 4 4 2 1 0 4 3 3 0 0 2 3 1 0 4 4 2 0 2 2 1 2 0 3 4 2 0 0 2 2
 2 4 3 0 2 3 3 2 1 0 0 1 0 1 4 0 1 3 0 4 1 0 1 1 3 4 3 4 3 2 4 0 3 2 2 3 0
 0 2 1 4 2 4 4 3 0 3 3 3 0 3 2 3 0 3 1 0 0 4 0 4 4 2 0 4 1 1 3 4 0 3 3 3 2
 3 1 3 2 4 2 4 3 0 1 3 2 0 1 3 3 0 0 3 3 4 4 4 4 0 2 3 2 1 4 2 1 1 2 3 2 0
 0 1 0 3 4 3 2 1 2 4 1 1 0 4 3 4 0 2 2 0 4 2 3 0 1 2 0 4 2 2 0 3 3 1 3 1 0
 4 4 0 2 1 1 3 2 3 0 0 0 3 3 4 4 4 4 1 0 1 2 4 3 3 0 1 2 0 3 3 0 2 0 4 0 0
 0 4 0 1 2 2 3 1 0 0 0 4 2 4 1 2 3 0 0 2 1 1 1 2 3 0 4 3 3 3 0 3 1 3 2 4 1
 4 2 0 1 1 0 3 3 0 3 0 2 2 4 4 3 0 0 2 1 4 3 3 2 3 1 3 0 4 3 4 3 0 4 2 1 2
 3 2 3 2 4 0 0 0 4 2 3 2 2 2 1 3 2 2 4 1 0 4 0 3 3 4 2 0 1 3 3 0 4 4 3 2 0
 0 4 0 4 1 4 3 2 0 0 1 1 4 0 2 3 4 2 2 0 3 3 0 1 4 1 0 3 1 2 3 1 0 2 0 0 0
 2 1 1 0 3 0 2 3 0 1 0 0 1 0 0 0 2 0 2 0 3 0 1 0 3 3 0 3 0 0 0 0 1 1 0 4 2
 1 3 4 3 1 3 2 4 1 0 3 2 0 1 2 4 2 0 3 0 2 3 2 1 2 3 4 3 2 4 0 1 1 2 3 4 0
 1 1 1 4 2 4 4 3 3 0 3 4 0 4 0 2 3 2 3 0 4 0 3 0 0 1 2 4 4 3 2 3 1 1 0 3 4
 4 0 0 1 0 2 0 0 2 1 3 1 1 0 4 4 0 3 3 0 0 3 0 0 1 0 1 2 1 4 0 0 2 0 0 3 1
 3 2 2 1 0 1 1 3 2 4 3 4 0 1 2 3 1 0 1 4 4 4 3 0 0 0 4 2 0 3 2 2 0 0 1 4 0
 0 4 1 3 0 0 2 4 0 1 0 4 4 2 4 4 1 3 1 4 0 3 4 1 4 2 3 2 1 2 2 2 1 1 0 4 0
 3 0 1 2 3 2 0 1 2 1 2 0 3 3 3 4 0 4 1 4 0 4 0 1 2 0 0 0 3 2 3 1 0 4 3 0 3
 0 4 3 2 4 0 2 1 2 0 4 4 0 3 0 0 4 1 3 3 4 1 3 2 0 0 1 1 3 2 0 4 2 2 4 1 3
 0 4 2 3 0 3 2 1 4 3 0 0 1 3 4 3 1 0 3 2 2 2 2 0 0 4 2 0 0 4 4 0 1 0 3 1 2
 4]
```

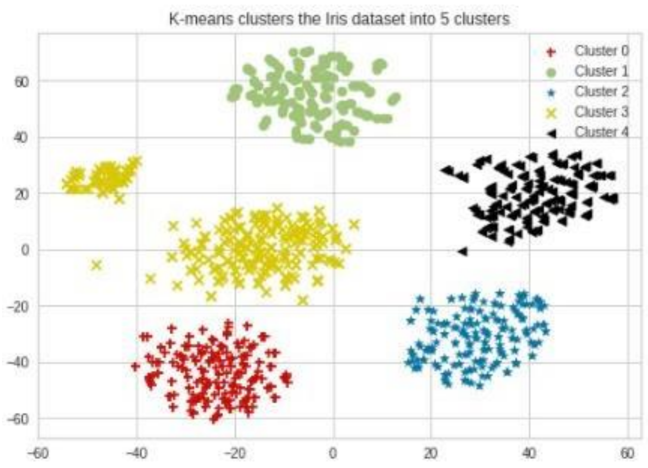
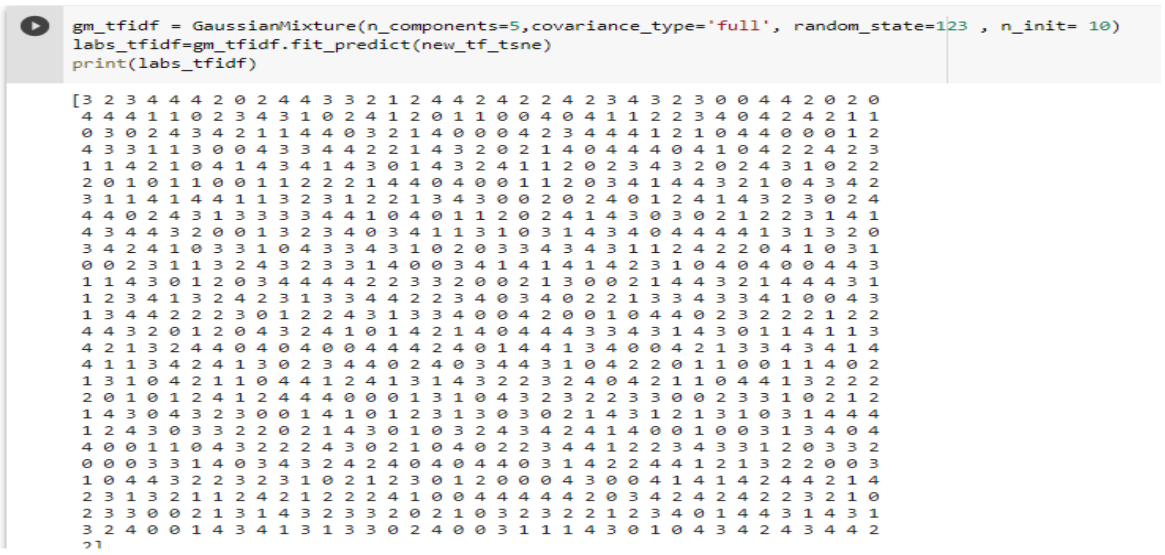


Fig (5.2)

- EM: with 5 cluster and general covariance matrix, also fit T-SNE.

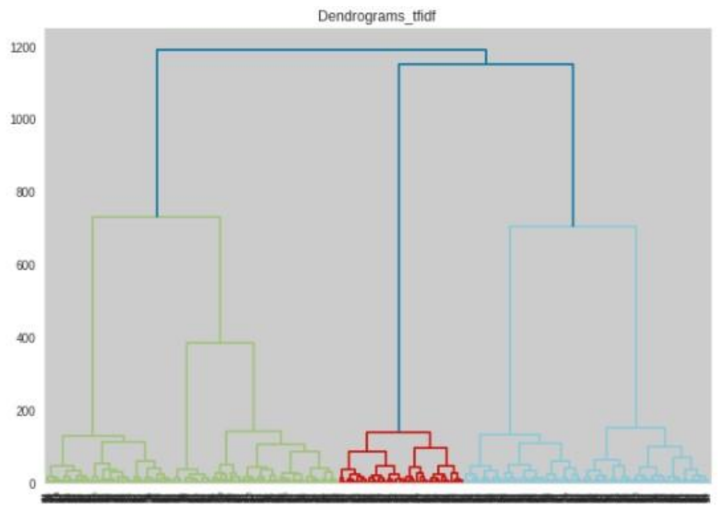


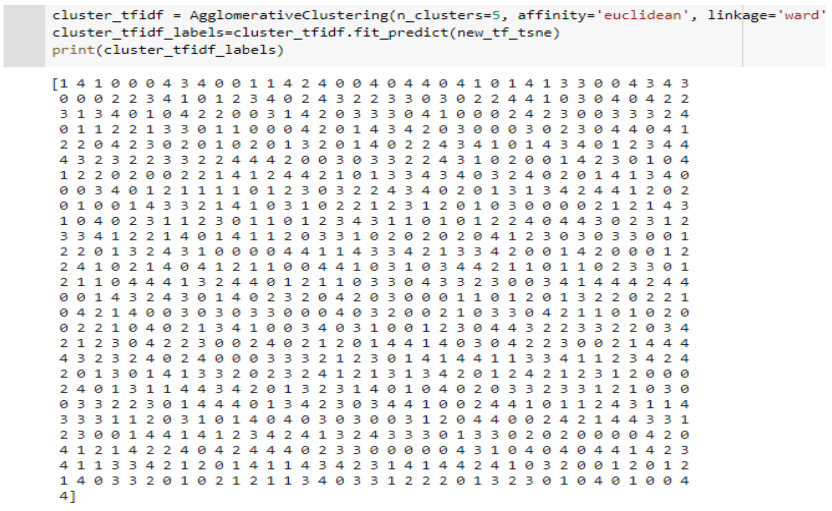
```
gm_tfidf = GaussianMixture(n_components=5,covariance_type='full', random_state=123 , n_init= 10)
labs_tfidf=gm_tfidf.fit_predict(new_tf_tsne)
print(labs_tfidf)
```

```
[3 2 3 4 4 4 2 0 2 4 4 3 3 2 1 2 4 4 2 4 2 2 4 2 3 4 3 2 3 0 0 4 4 2 0 2 0
 4 4 4 1 1 0 2 3 4 3 1 0 2 4 1 2 0 1 1 0 0 4 0 4 1 1 2 2 3 4 0 4 2 4 2 1 1
 0 3 0 2 4 3 4 2 1 1 4 4 0 3 2 1 4 0 0 0 4 2 3 4 4 4 1 2 1 0 4 4 0 0 0 1 2
 4 3 3 1 1 3 0 0 4 3 3 4 4 2 2 1 4 3 2 0 2 1 4 0 4 4 4 0 4 1 0 4 2 2 4 2 3
 1 1 4 2 1 0 4 1 4 3 4 1 4 3 0 1 4 3 2 4 1 1 2 0 2 3 4 3 2 0 2 4 3 1 0 2 2
 2 0 1 0 1 1 0 0 1 1 2 2 2 1 4 4 0 4 0 0 1 1 2 0 3 4 1 4 4 3 2 1 0 4 3 4 2
 3 1 1 4 1 4 4 1 1 3 2 3 1 2 2 1 3 4 3 0 0 2 0 2 4 0 1 2 4 1 4 3 2 3 0 2 4
 4 4 0 2 4 3 1 3 3 3 3 4 4 1 0 4 0 1 1 2 0 2 4 1 4 3 0 3 0 2 1 2 2 3 1 4 1
 4 3 4 4 3 2 0 0 1 3 2 3 4 0 3 4 1 1 3 1 0 3 1 4 3 4 0 4 4 4 4 1 3 1 3 2 0
 3 4 2 4 1 0 3 3 1 0 4 3 3 4 3 1 0 2 0 3 3 4 3 4 3 1 1 2 4 2 2 0 4 1 0 3 1
 0 0 2 3 1 1 3 2 4 3 2 3 3 1 4 0 0 3 4 1 4 1 4 1 4 2 3 1 0 4 0 4 0 0 4 4 3
 1 1 4 3 0 1 2 0 3 4 4 4 4 2 2 3 3 2 0 0 2 1 3 0 0 2 1 4 4 3 2 1 4 4 4 3 1
 1 2 3 4 1 3 2 4 2 3 1 3 3 4 4 2 2 3 4 0 3 4 0 2 2 1 3 3 4 3 3 4 1 0 0 4 3
 1 3 4 4 2 2 2 3 0 1 2 2 4 3 1 3 3 4 0 0 4 2 0 0 1 0 4 4 0 2 3 2 2 2 1 2 2
 4 4 3 2 0 1 2 0 4 3 2 4 1 0 1 4 2 1 4 0 4 4 4 3 3 4 3 1 4 3 0 1 1 4 1 1 3
 4 2 1 3 2 4 4 0 4 0 4 0 0 4 4 4 2 4 0 1 4 4 1 3 4 0 0 4 2 1 3 3 4 3 4 1 4
 4 1 1 3 4 2 4 1 3 0 2 3 4 4 0 2 4 0 3 4 4 3 1 0 4 2 2 0 1 1 0 0 1 1 4 0 2
 1 3 1 0 4 2 1 1 0 4 4 1 2 4 1 3 1 4 3 2 2 3 2 4 0 4 2 1 1 0 4 4 1 3 2 2 2
 2 0 1 0 1 2 4 1 2 4 4 4 0 0 0 1 3 1 0 4 3 2 3 2 2 3 3 0 0 2 3 3 1 0 2 1 2
 1 4 3 0 4 3 2 3 0 0 1 4 1 0 1 2 3 1 3 0 3 0 2 1 4 3 1 2 1 3 1 0 3 1 4 4 4
 1 2 4 3 0 3 3 2 2 0 2 1 4 3 0 1 0 3 2 4 3 4 2 4 1 4 0 0 1 0 0 3 1 3 4 0 4
 4 0 0 1 1 0 4 3 2 2 2 4 3 0 2 1 0 4 0 2 2 3 4 4 1 2 2 3 4 3 3 1 2 0 3 3 2
 0 0 0 3 3 1 4 0 3 4 3 2 4 2 4 0 4 0 4 4 0 3 1 4 2 2 4 4 1 2 1 3 2 2 0 0 3
 1 0 4 4 3 2 2 3 2 3 1 0 2 1 2 3 0 1 2 0 0 0 4 3 0 0 4 1 4 1 4 2 4 4 2 1 4
 2 3 1 3 2 1 1 2 4 2 1 2 2 2 4 1 0 0 4 4 4 4 4 2 0 3 4 2 4 2 4 2 2 3 2 1 0
 2 3 3 0 0 2 1 3 1 4 3 2 3 3 2 0 2 1 0 3 2 3 2 2 1 2 3 4 0 1 4 4 3 1 4 3 1
 3 2 4 0 0 1 4 3 4 1 3 1 3 3 0 2 4 0 0 3 1 1 1 4 3 0 1 0 4 3 4 2 4 3 4 4 2
 2]
```

Fig (5.3)

- Hierarchical: fit the T-SNE and using 5 clusters with Euclidean matrix which works with Ward linkage that minimizes the variance



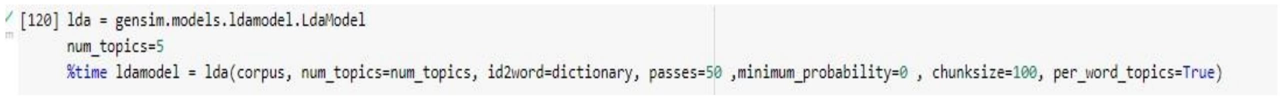


```
cluster_tfidf = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward'
cluster_tfidf_labels=cluster_tfidf.fit_predict(new_tf_tsne)
print(cluster_tfidf_labels)
```

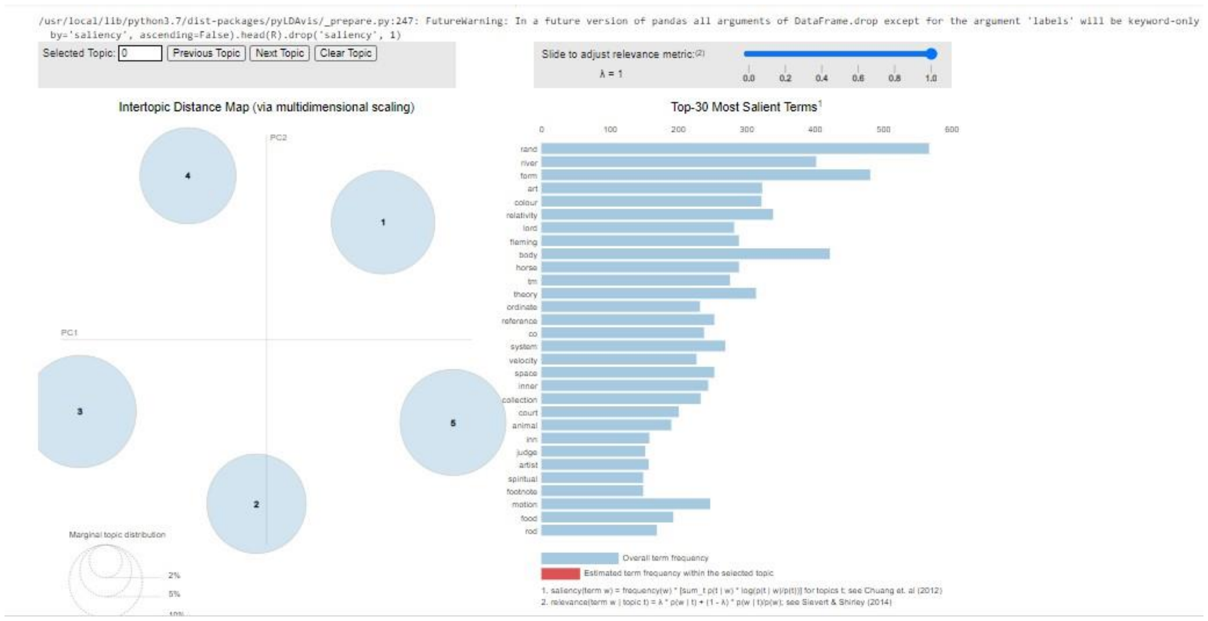
```
[1 4 1 0 0 0 4 3 4 0 0 1 1 4 2 4 0 0 4 0 4 4 0 4 1 0 1 4 1 3 3 0 0 4 3 4 3
 0 0 0 2 2 3 4 1 0 1 2 3 4 0 2 4 3 2 2 3 3 0 3 0 2 2 4 4 1 0 3 0 4 0 4 2 2
 3 1 3 4 0 1 0 4 2 2 0 0 3 1 4 2 0 3 3 3 0 4 1 0 0 0 2 4 2 3 0 0 3 3 3 2 4
 0 1 1 2 2 1 3 3 0 1 1 0 0 0 4 2 0 1 4 3 4 2 0 3 0 0 0 3 0 2 3 0 4 4 0 4 1
 2 2 0 4 2 3 0 2 0 1 0 2 0 1 3 2 0 1 4 0 2 2 4 3 4 1 0 1 4 3 4 0 1 2 3 4 4
 4 3 2 3 2 2 3 3 2 2 4 4 4 2 0 0 3 0 3 3 2 2 4 3 1 0 2 0 0 1 4 2 3 0 1 0 4
 1 2 2 0 2 0 0 2 2 1 4 1 2 4 4 2 1 0 1 3 3 4 3 4 0 3 2 4 0 2 0 1 4 1 3 4 0
 0 0 3 4 0 1 2 1 1 1 1 0 1 2 3 0 3 2 2 4 3 4 0 2 0 1 3 1 3 4 2 4 4 1 2 0 2
 0 1 0 0 1 4 3 3 2 1 4 1 0 3 1 0 2 2 1 2 3 1 2 0 1 0 3 0 0 0 0 2 1 2 1 4 3
 1 0 4 0 2 3 1 1 2 3 0 1 1 0 1 2 3 4 3 1 1 0 1 0 1 2 2 4 0 4 4 3 0 2 3 1 2
 3 3 4 1 2 2 1 4 0 1 4 1 1 2 0 3 3 1 0 2 0 2 0 2 0 4 1 2 3 0 3 0 3 3 0 0 1
 2 2 0 1 3 2 4 3 1 0 0 0 0 4 4 1 1 4 3 3 4 2 1 3 3 4 2 0 0 1 4 2 0 0 0 1 2
 2 4 1 0 2 1 4 0 4 1 2 1 1 0 0 4 4 1 0 3 1 0 3 4 4 2 1 1 0 1 1 0 2 3 3 0 1
 2 1 1 0 4 4 4 1 3 2 4 4 0 1 2 1 1 0 3 3 0 4 3 3 2 3 0 0 3 4 1 4 4 4 2 4 4
 0 0 1 4 3 2 4 3 0 1 4 0 2 3 2 0 4 2 0 3 0 0 0 1 1 0 1 2 0 1 3 2 2 0 2 2 1
 0 4 2 1 4 0 0 3 0 3 0 3 3 0 0 0 4 0 3 2 0 0 2 1 0 3 3 0 4 2 1 1 0 1 0 2 0
 0 2 2 1 0 4 0 2 1 3 4 1 0 0 3 4 0 3 1 0 0 1 2 3 0 4 4 3 2 2 3 3 2 2 0 3 4
 2 1 2 3 0 4 2 2 3 0 0 2 4 0 2 1 2 0 1 4 4 1 4 0 3 0 4 2 2 3 0 0 2 1 4 4 4
 4 3 2 3 2 4 0 2 4 0 0 0 3 3 3 2 1 2 3 0 1 4 1 4 4 1 1 3 3 4 1 1 2 3 4 2 4
 2 0 1 3 0 1 4 1 3 3 2 0 2 3 2 4 1 2 1 3 1 3 4 2 0 1 2 4 2 1 2 3 1 2 0 0 0
 2 4 0 1 3 1 1 4 4 3 4 2 0 1 3 2 3 1 4 0 1 0 4 0 2 0 3 3 2 3 3 1 2 1 0 3 0
 0 3 3 2 2 3 0 1 4 4 4 0 1 3 4 2 3 0 3 4 4 1 0 0 2 4 4 1 0 1 1 2 4 3 1 1 4
 3 3 3 1 1 2 0 3 1 0 1 4 0 4 0 3 0 3 0 0 3 1 2 0 4 4 0 0 2 4 2 1 4 4 3 3 1
 2 3 0 0 1 4 4 1 4 1 2 3 4 2 4 1 3 2 4 3 3 3 0 1 3 3 0 2 0 2 0 0 0 0 4 2 0
 4 1 2 1 4 2 2 4 0 4 2 4 4 4 0 2 3 3 0 0 0 0 0 4 3 1 0 4 0 4 0 4 4 1 4 2 3
 4 1 1 3 3 4 2 1 2 0 1 4 1 1 4 3 4 2 3 1 4 1 4 4 2 4 1 0 3 2 0 0 1 2 0 1 2
 1 4 0 3 3 2 0 1 0 2 1 2 1 1 3 4 0 3 3 1 2 2 2 0 1 3 2 3 0 1 0 4 0 1 0 0 4
 4]
```

Fig(5.4)

➢ LDA as topic modeling:



```
[120] lda = gensim.models.ldamodel.LdaModel
      num_topics=5
      %time ldamodel = lda(corpus, num_topics=num_topics, id2word=dictionary, passes=50 ,minimum_probability=0 , chunksize=100, per_word_topics=True)
```



```
/usr/local/lib/python3.7/dist-packages/pyLDAvis/_prepare.py:247: FutureWarning: In a future version of pandas all arguments of DataFrame.drop except for the argument 'labels' will be keyword-only
  by='saliency', ascending=False).head(R).drop('saliency', 1)
```

🟦 Overall term frequency

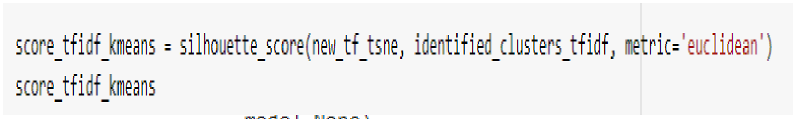
🟥 Estimated term frequency within the selected topic

1. saliency(term w) = frequency(w) * [sum_t p(t | w) * log(p(t | w)/p(t))] for topics t; see Chuang et. al (2012)
2. relevance(term w | topic t) = λ * p(w | t) + (1 - λ) * p(w | t)/p(w); see Sievert & Shirley (2014)

6- Model Evaluations:

That’s time for clustering validation and evaluation of clustering quality.
We used evaluation methods:

- **Silhouette**:
  returns the mean Silhouette Coefficient over all samples.



```
score_tfidf_kmeans = silhouette_score(new_tf_tsne, identified_clusters_tfidf, metric='euclidean')
score_tfidf_kmeans
```

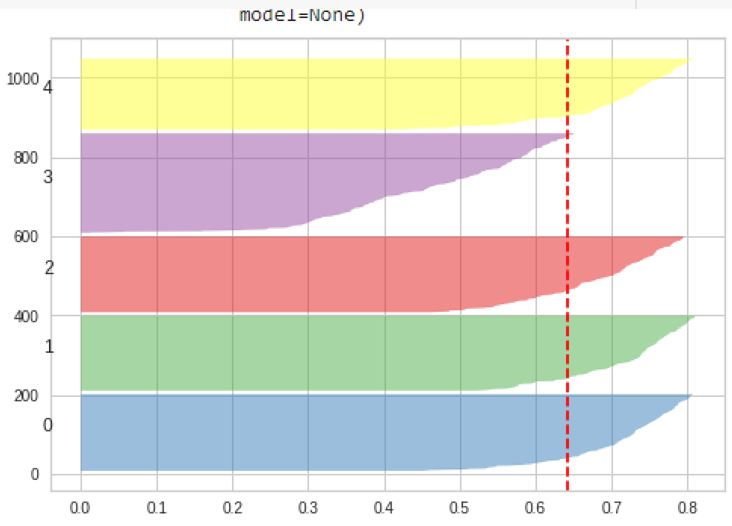
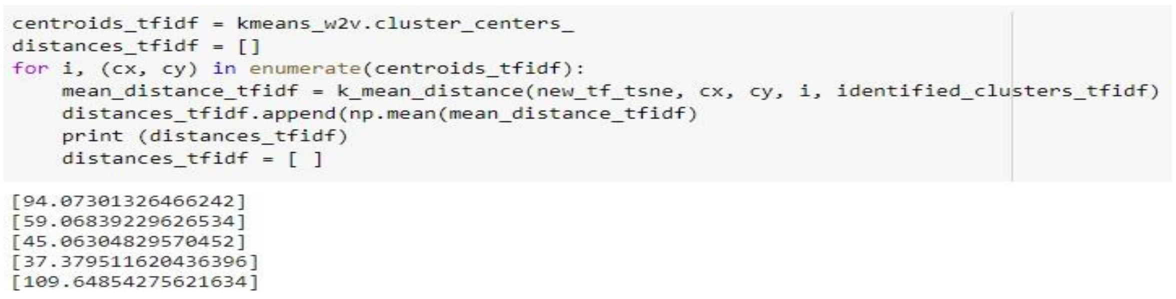


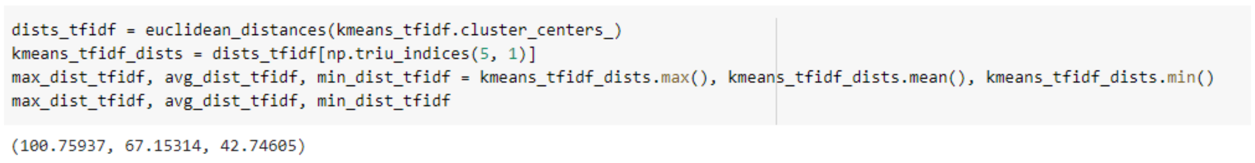
Fig (6.1)

- **Consistency**:
  We measure the distance between each cluster and each point within cluster.



```
centroids_tfidf = kmeans_w2v.cluster_centers_
distances_tfidf = []
for i, (cx, cy) in enumerate(centroids_tfidf):
    mean_distance_tfidf = k_mean_distance(new_tf_tsne, cx, cy, i, identified_clusters_tfidf)
    distances_tfidf.append(np.mean(mean_distance_tfidf)
    print (distances_tfidf)
    distances_tfidf = [ ]
```

```
[94.07301326466242]
[59.06839229626534]
[45.06304829570452]
[37.379511620436396]
[109.64854275621634]
```

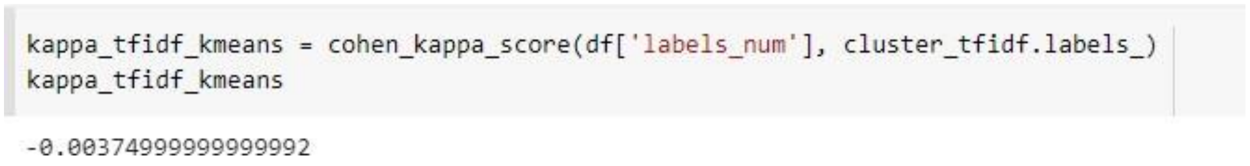


```
dists_tfidf = euclidean_distances(kmeans_tfidf.cluster_centers_)
kmeans_tfidf_dists = dists_tfidf[np.triu_indices(5, 1)]
max_dist_tfidf, avg_dist_tfidf, min_dist_tfidf = kmeans_tfidf_dists.max(), kmeans_tfidf_dists.mean(), kmeans_tfidf_dists.min()
max_dist_tfidf, avg_dist_tfidf, min_dist_tfidf
```

```
(100.75937, 67.15314, 42.74605)
```

Fig (6.2)

➢ Kappa

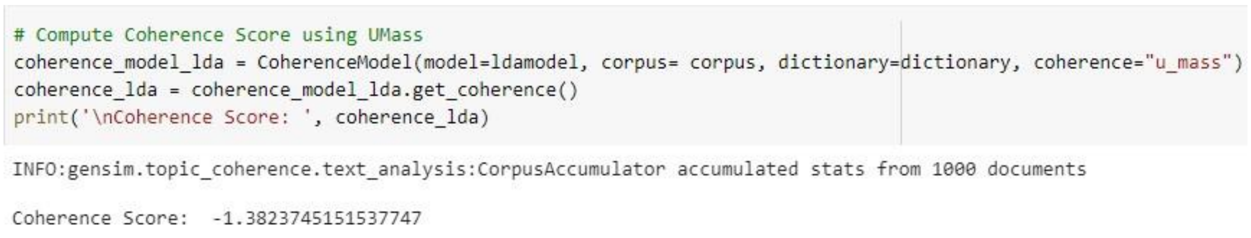


```
kappa_tfidf_kmeans = cohen_kappa_score(df['labels_num'], cluster_tfidf.labels_)
kappa_tfidf_kmeans
```

```
-0.003749999999999992
```

Fig (6.3)

➢ Coherence: it’s used with LDA as topic modeling.



```
# Compute Coherence Score using UMass
coherence_model_lda = CoherenceModel(model=ldamodel, corpus= corpus, dictionary=dictionary, coherence="u_mass")
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

```
INFO:gensim.topic_coherence.text_analysis:CorpusAccumulator accumulated stats from 1000 documents

Coherence Score:  -1.3823745151537747
```

Fig (6.4)

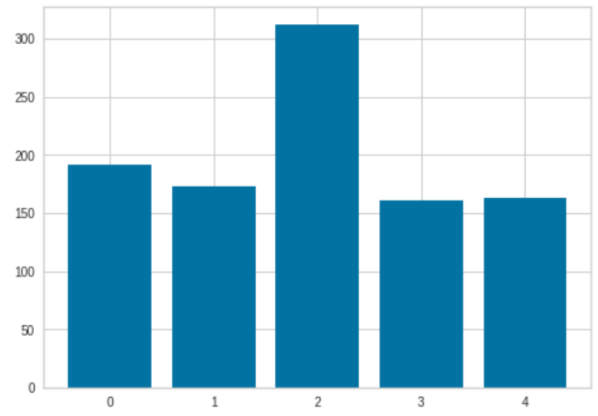
2- Compare clustering result:

- BOW
  - *EM*

    Silhouette score: 0.45908052

    Kappa: -0.2275

```
Counter({2: 312, 0: 191, 1: 173, 4: 163, 3: 161})
<BarContainer object of 5 artists>
```

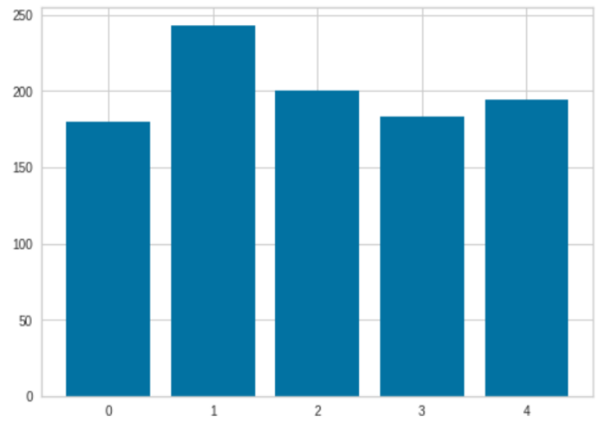


  - *K-Means*

    Silhouette score: 0.48006293

    Kappa: -0.2337499

```
Counter({1: 243, 2: 200, 4: 194, 3: 183, 0: 180})
<BarContainer object of 5 artists>
```

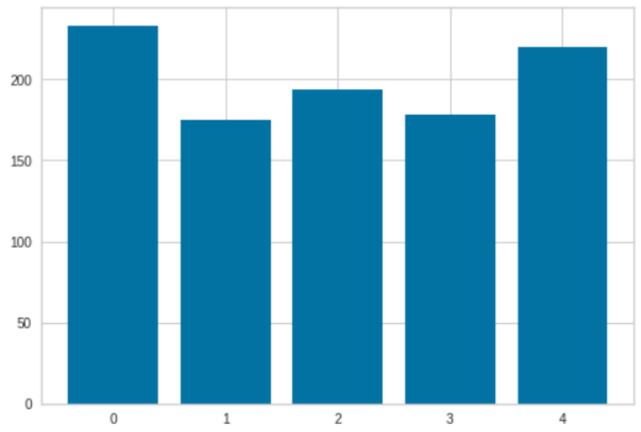


- *Hierarchical*

  Silhouette score: 0.46273592

  Kappa: -0.0149999

```
Counter({0: 233, 4: 220, 2: 194, 3: 178, 1: 175})
<BarContainer object of 5 artists>
```



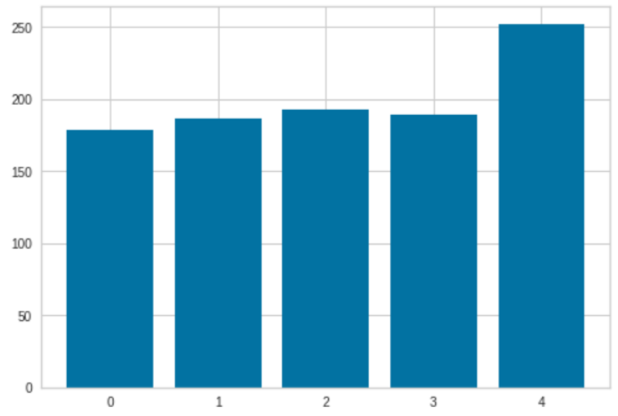
➢ TF-IDF

- *EM*

  Silhouette score: 0.63862866

  Kappa: -0.233749

```
Counter({4: 252, 2: 193, 3: 189, 1: 187, 0: 179})
<BarContainer object of 5 artists>
```

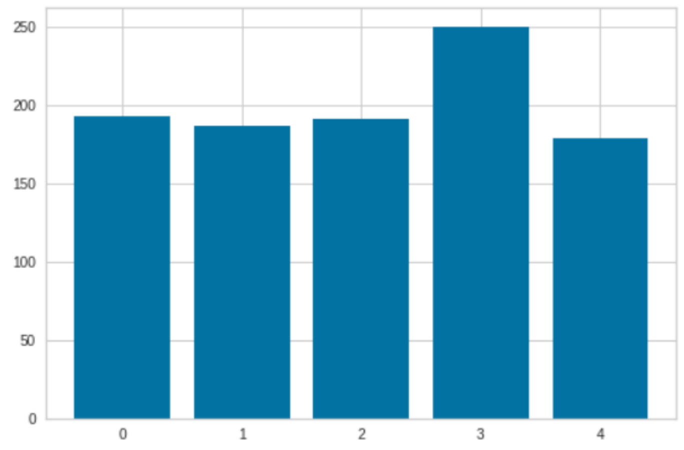


- *K-Means*

  Silhouette score: 0.6413468

  Kappa: -0.23625

```
Counter({3: 250, 0: 193, 2: 191, 1: 187, 4: 179})
<BarContainer object of 5 artists>
```

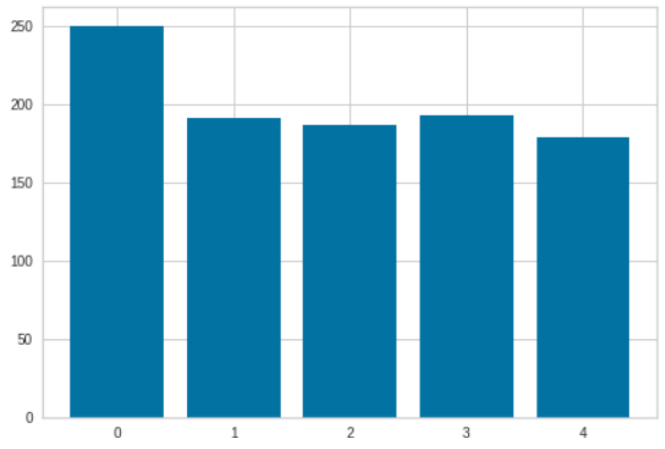


- Hierarchical

  Silhouette score: 0.6413468

  Kappa: -0.23625

```
Counter({0: 250, 3: 193, 1: 191, 2: 187, 4: 179})
<BarContainer object of 5 artists>
```



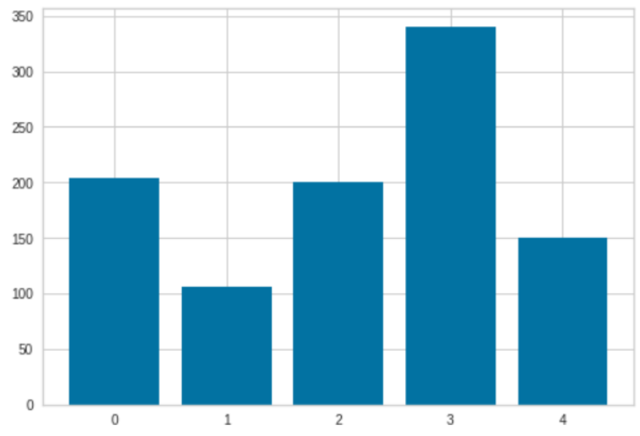
➢ LDA

- EM

  Silhouette score: 0.537854

Kappa: 0.16625

```
Counter({3: 340, 0: 204, 2: 200, 4: 150, 1: 106})
<BarContainer object of 5 artists>
```

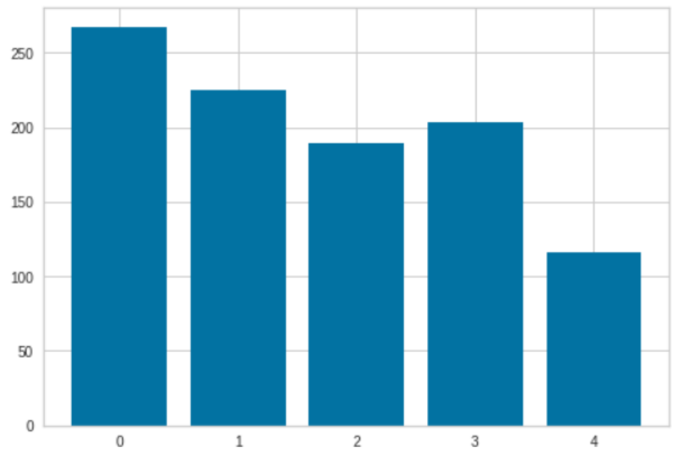


- K-Means

  Silhouette score: 0.57938975

  Kappa: -0.2250

```
<BarContainer object of 5 artists>
```

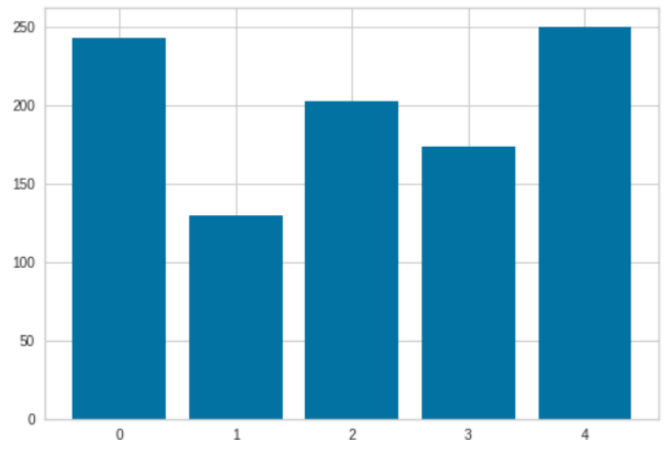


- Hierarchical

  Silhouette score: 0.5726299

Kappa: -0.20875

```
Counter({4: 250, 0: 243, 2: 203, 3: 174, 1: 130})
<BarContainer object of 5 artists>
```

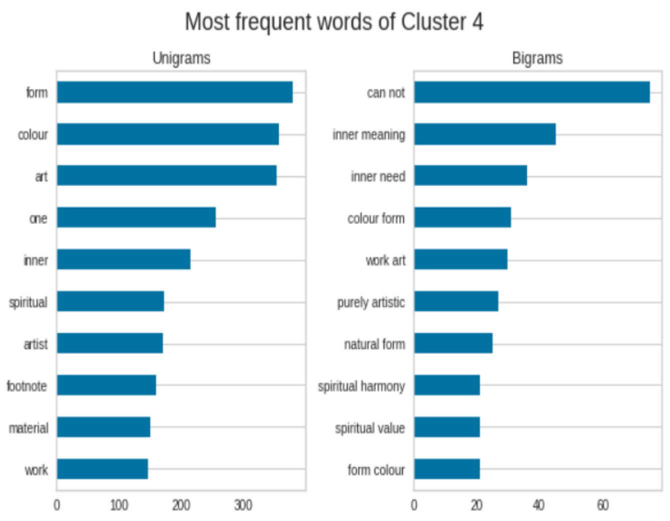


		BOW	TFIDF	Word2vec	LDA
EM	silhouette	0.45908052	0.63862866	0.60395604	0.537854
	kappa	-0.2275	-0.233749	-0.22124999999999995	0.16625
K_means	silhouette	0.48006293	0.6413468	0.6035826	0.57938975
	kappa	-0.2337499	-0.23625	0.018750000000000044	-0.2250
Hierarchical	silhouette	0.46273592	0.6413468	0.5381887	0.5726299
	kappa	-0.0149999	-0.23625	0.16374999999999995	-0.20875

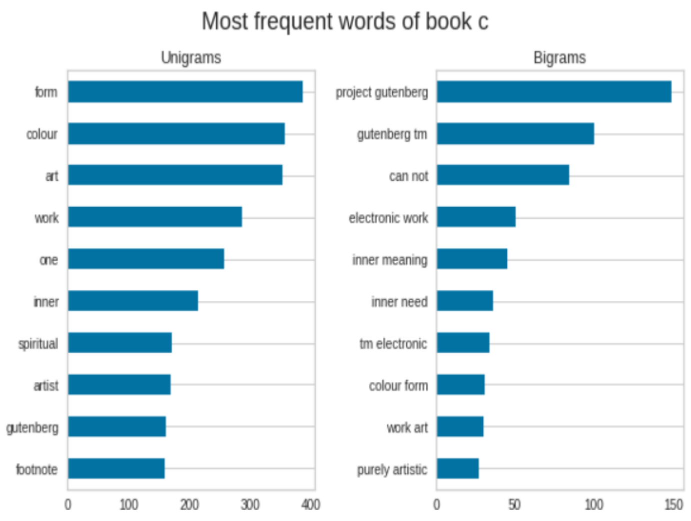
3- Error Analysis:

Error analysis is an important process at any clustering task in general and very important in NLP clustering especially as it give us more insights about our data before and after clustering, which gives us the advantage to identify the main factors that leads to misclustering and reduces the silhouette scores in all models. So, we applied some techniques to analysis and visualize our data before and after the clustering process which gives us a better vison about the data and the root causes of the errors in clustering. We decided to implement our analysis techniques on K-Means model after TF-IDF transformation as it has the highest silhouette score after performing the clustering. At the beginning we had to explore the data before and after the clustering.

After Clustering



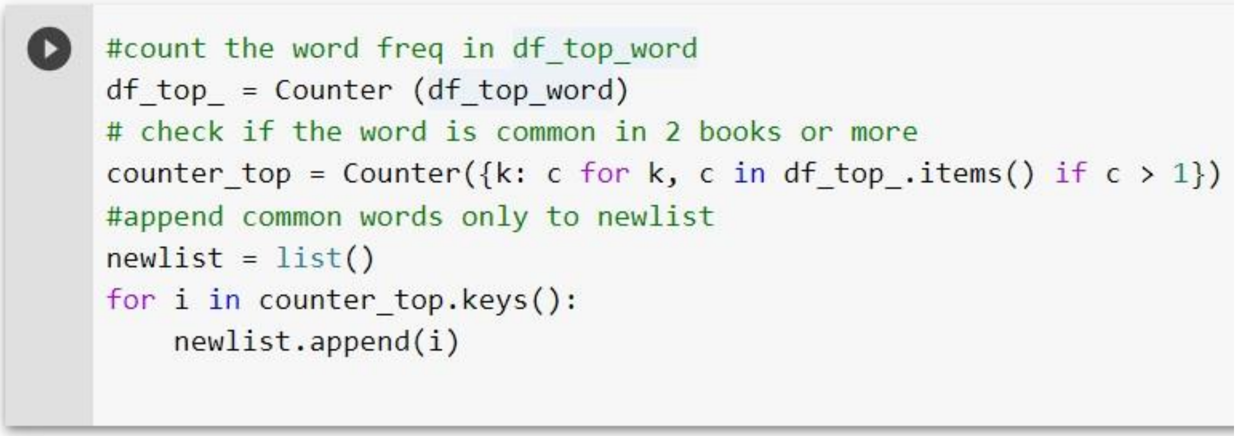
Before clustering



By comparing the word frequency of Book (C) and cluster (4) we can figure out some changes due to the errors in clustering process, from the above figure we can notice that the frequency of some words changed like the frequency of (form) before clustering was almost four hundred and after clustering decreed to almost 350 times, and example of the clustering changes is the frequency of the word (work) which decreased from almost 3 hundred times to 145 time and (Gutenberg) completely disappeared from the top frequent words list, same changes happens in unigrams lists, but after we analysis the result of the other clusters we found that words like work and Gutenberg which are decreased by a huge rate in cluster 4 and their frequency increased in another clusters. Which leads us to a question what actually the root causes of this errors and how could we overcome these errors. We reached a conclusion after exploring our data before and after clustering maybe the errors appear due to common words which made the models to be confused while they tried to cluster some partitions based on their similarity while those partitions have words which are common across more than one book. So, we have developed and implemented an approach which takes the top common words across all books and removing those words from all books then we tried to implement the selected model again. Then we calculated the silhouette score again and we noticed that the score increased and it keeps increasing if we removed more common words from the data.

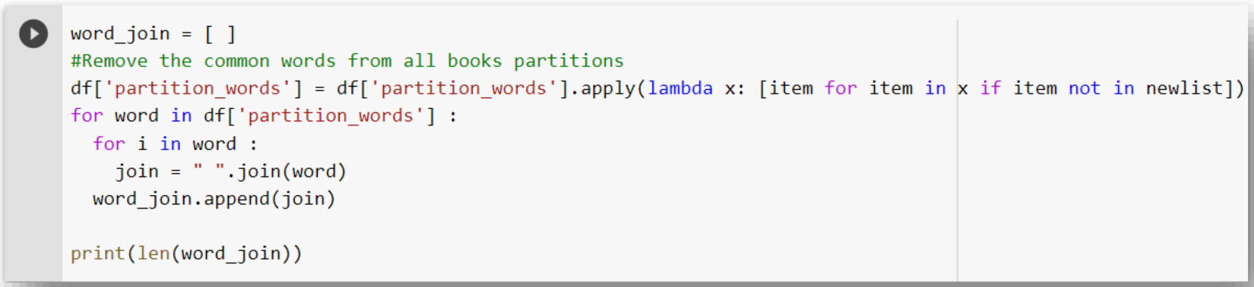
We created a list which contains the most frequented words in every book the we removed unique words and we kept only common words which they most frequent words in more than one book.

Count the top common words in books



```
#count the word freq in df_top_word
df_top_ = Counter (df_top_word)
# check if the word is common in 2 books or more
counter_top = Counter({k: c for k, c in df_top_.items() if c > 1})
#append common words only to newlist
newlist = list()
for i in counter_top.keys():
    newlist.append(i)
```

Removing the most frequent common words from all book’s partitions

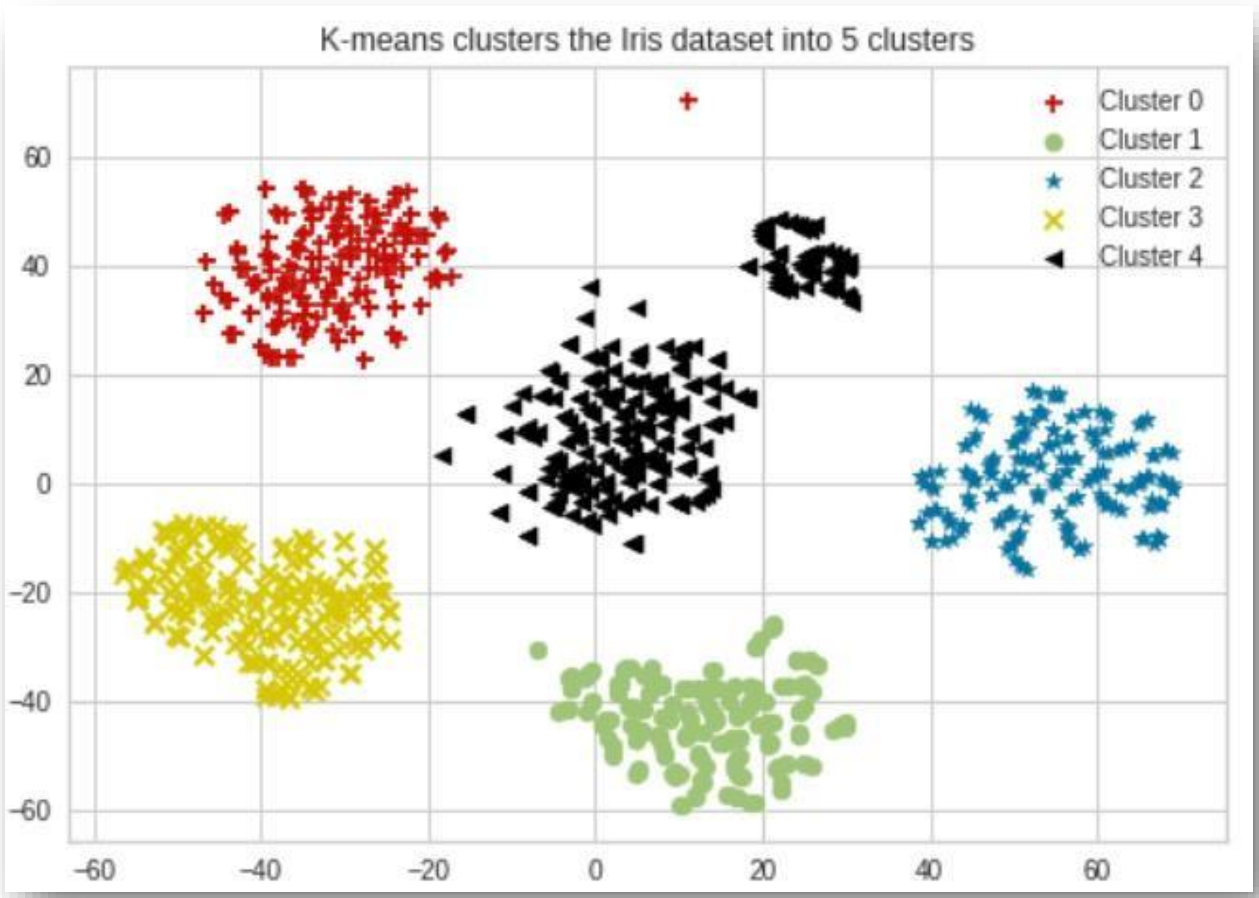


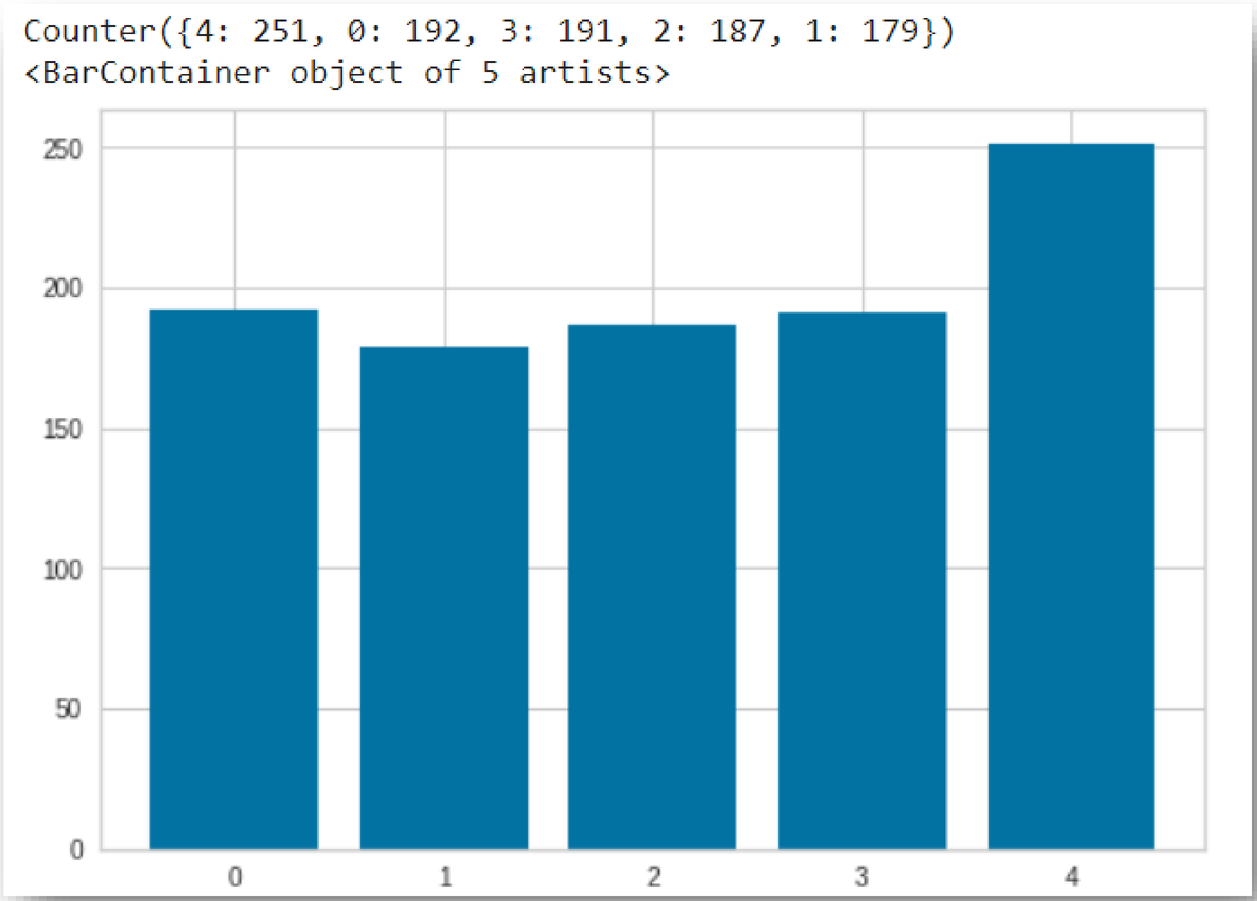
```
word_join = [ ]
#Remove the common words from all books partitions
df['partition_words'] = df['partition_words'].apply(lambda x: [item for item in x if item not in newlist])
for word in df['partition_words'] :
  for i in word :
    join = " ".join(word)
  word_join.append(join)

print(len(word_join))
```

Implementing clustering and calculate the silhouette score again

K-Means after TF-IDF transformations



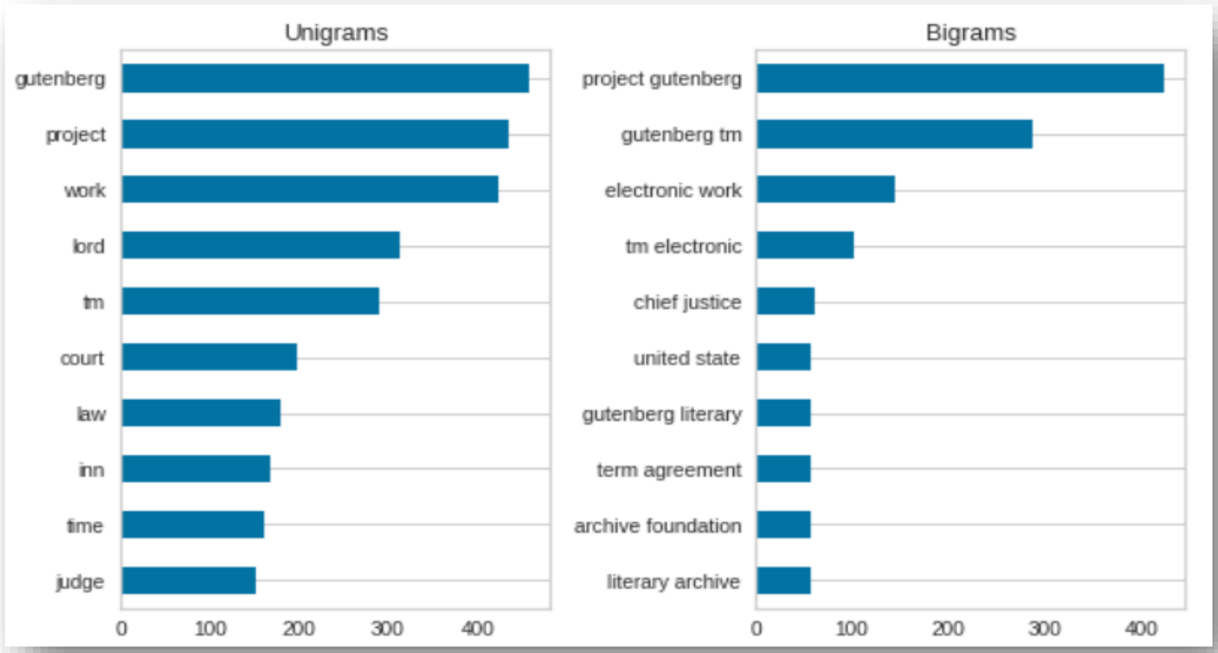


```
Counter({4: 251, 0: 192, 3: 191, 2: 187, 1: 179})
<BarContainer object of 5 artists>
```

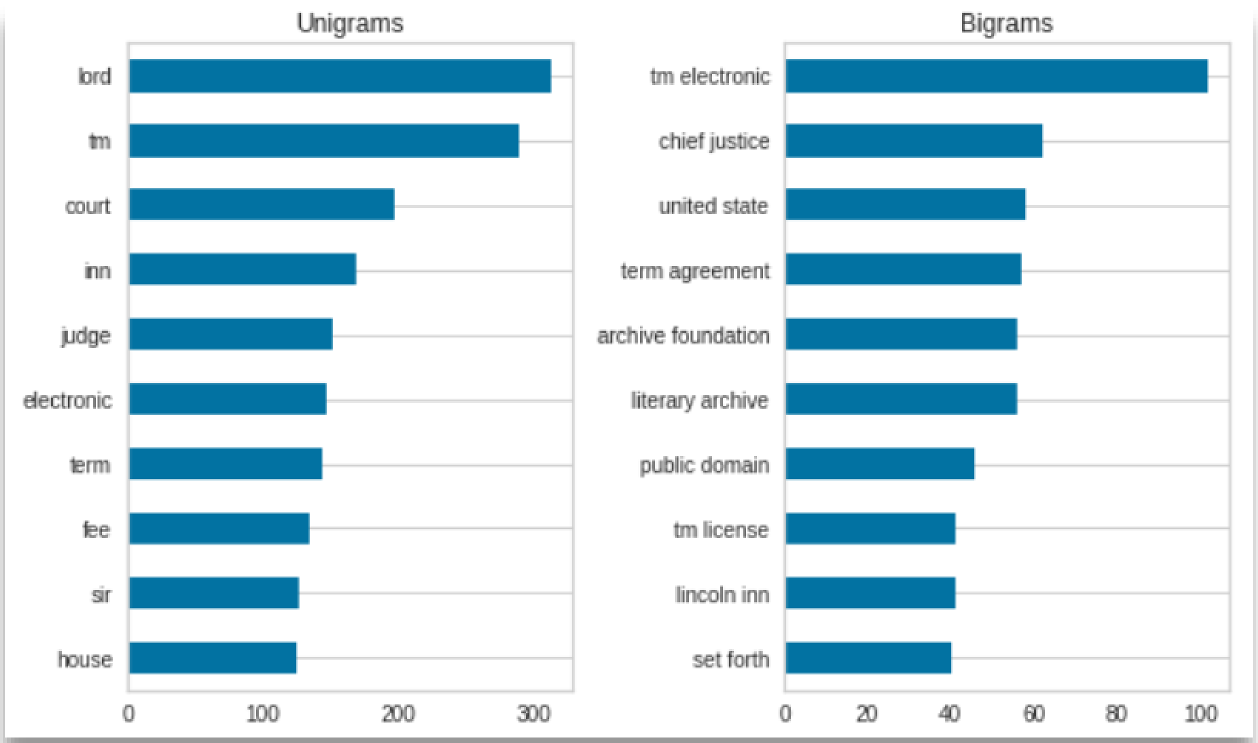
After calculating the silhouette score, we noticed that the score increased from 0.64124 To 0.65217304 and it keeps increasing if we removed more common words from the data.

Exploring the clusters data after clustering to get more insights:

Clustering without removing common words

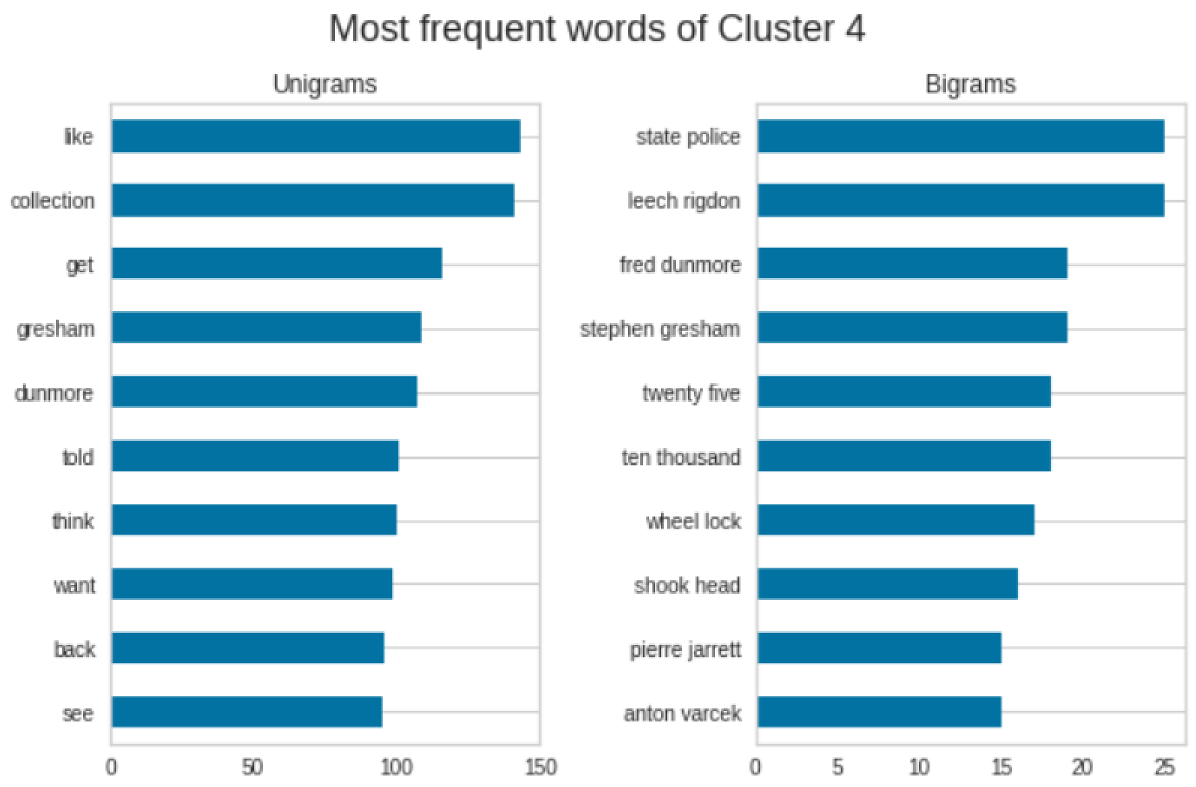


Clustering without removing common words



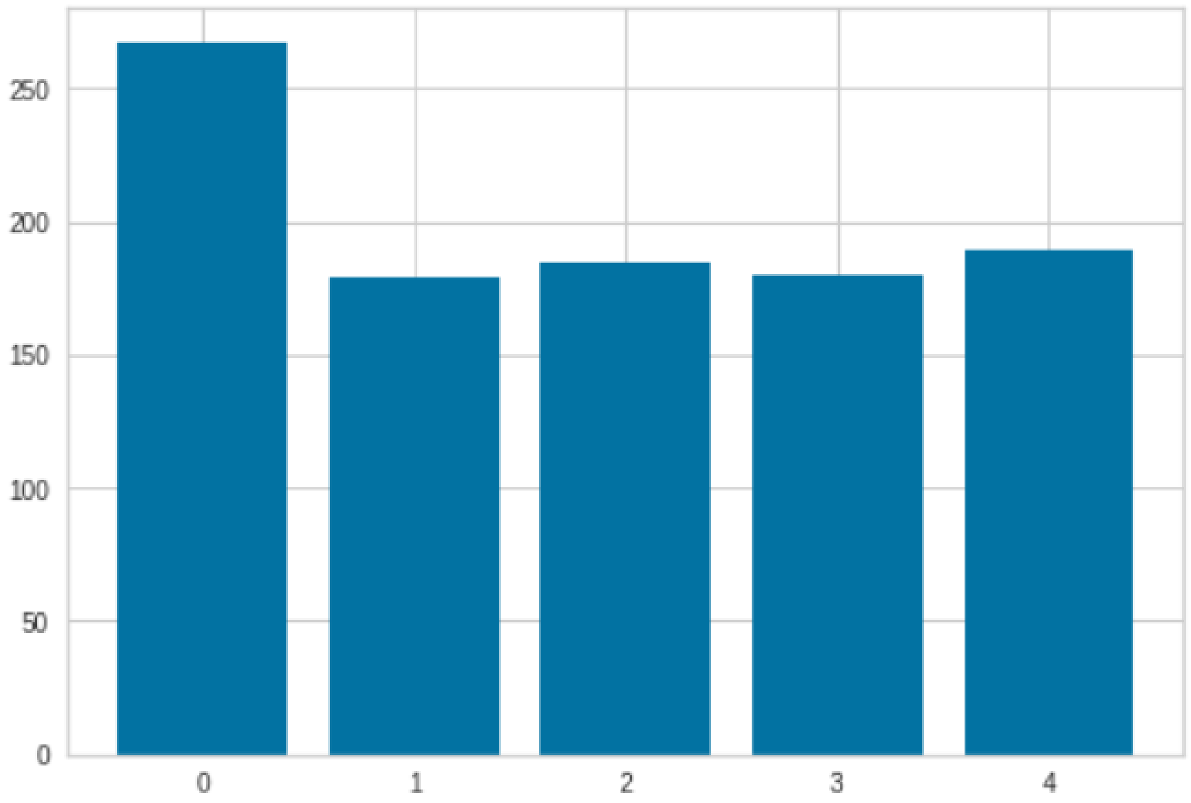
We can get some insights from the above figures which contain word frequency before and after removing the most frequent common words from the data, cluster before removing the most frequent common words contains words that not relevant to the book genre only but it common across all books which leads to some errors in clustering, while cluster after removing the most frequent common words contains words that are unique and could be considered as a good representation and main characteristics that identifying the book genre.

we tried to remove the top 10 most frequent words not just the common words but this approach achieved less silhouette score which reached 0.55856544 less than the most frequent common words silhouette score which achieved almost 0.6521 and the error increased as we removed some words which considered as main characteristics that identifying the book genre.



Another problem we faced after implementing this approach we couldn’t match every cluster to its genre by exploring clusters data and getting insights to identify the main characteristics of the cluster as we removed the most frequent words even they are unique for their genre and the above figure shows the cluster that we couldn’t match it to its right genre. The below figures shows that a specific cluster gets more partitions more the others due to the increasing of error causes as we removed some unique words from the genre data.

```
Counter({0: 267, 4: 189, 2: 185, 3: 180, 1: 179})
<BarContainer object of 5 artists>
```



Conclusion:

In summary, Text Clustering is one of the most hardest fields as it needs to be aware about the data you had and to explore your data before and after the clustering process, So, We had to prepare the data and explore it after preparing to get insights about the data we had to have the ability to understand the main characteristics of the clusters after we don the clustering process , We implemented different feature engineering techniques like BOW and TF-IDF (also we used other features LDA, Word-Embedding). And we had applied different clustering models like K-means, EM and Hierarchical clustering algorithms. We performed evaluations by calculating Kappa against true authors, Consistency, Coherence and Silhouette. And we performed some Error-Analysis approaches to Identity what were the characteristics of the instance records that
threw the machine off, using the top 10 frequent words and/or top collocations.