# Capstone 3 Final Report

## Sample Data Cleaning and Exploration:

This project started with four documents as sample data to test our cleaning and exploration methods.

For natural language processing, data cleaning is the most important step and where we spend most of our time and energy.

```
# Load text files
file_paths = ["Aspen_Barricades_of_Culture_Wars.txt",
              "Douglas_Murray_IQ_Politics_and_the_Left.txt",
              "Nina_Paley_Animator_Extraordinaire.txt",
              "DR_Iain_McGilchrist.txt"]
texts = []
for file_path in file_paths:
    with open(file_path, "r") as file:
        texts.append(file.read())
```

There are 24797 words in total in these four documents. And there is what a sample of the first document looks like:

```
def count_total_words(text_list):
    total_words = 0
    for text in text_list:
        words = text.split()
    re
print(count_total_words(texts))
print(texts[0][:100])
```

(function) def count_total_words(text_list: Any) -> int

✓ 0.0s

```
24797
Well, I think you don’t want to underestimate the role that technological transformation has played
```

We proceed to divide the texts into smaller documents, each containing only 25 sentences. Why 25 sentences?

I manually counted a sample of conversations inside the documents. and 25 is the median number of sentences to cover a topic.

```python
# Divide into smaller documents with 25 sentences each
max_sentences_per_document = 25
all_smaller_documents = []
for text in texts:
    smaller_documents = divide_into_smaller_documents(text, max_sentences_per_document)
    all_smaller_documents.extend(smaller_documents)

# Load every document into a NumPy array
num_documents = len(all_smaller_documents)
array_documents = np.empty(num_documents, dtype=object)
for i, doc in enumerate(all_smaller_documents):
    array_documents[i] = '\n'.join(doc)

# Save the array as a .npy file
np.save("array_documents.npy", array_documents)
```

This yields us 69 documents, each containing 25 sentences.

```python
print(array_documents[0])
print(len(array_documents))
```

✓ 0.0s

```
Well, I think you don't want to underestimate the role that technological transformation has played in this.
I've been thinking about YouTube and podcasts quite intensely for about two years.
I started putting my university lectures on YouTube in 2013.
I did that for a variety of reasons—mostly curiosity and the drive to learn.
I've found that, if I want to learn a technology, the best way to do it is to use it.
I'm always learning new technologies—not that that makes me particularly unique—and I had some success with my lectures on public television in Canada.
I did some lectures with this series called "Big Ideas" on Canadian public television.
There's about 200 of those lectures, and I did five of them.
Two hundred done by 200 different people, but I did five of them, and they were regularly in the top 10 of the most viewed lectures.
So I knew there was some broader market for, let's say, ideas.
I thought, "well, I might as well put my lectures up on YouTube, and see what happens."
And then, by April of 2016, I had a million views.
I thought, "huh.
The only reason people are watching these is because they want to watch them, because they're actually really hard."
A million of something is a lot.
If you sell a million copies of your book—well, first of all, that never happens, right?
It's very, very rare.
You never have your scientific papers cited a million times.
You rarely have a million dollars.
It's a very large number.
Well, fair enough.
It's, of course, not as uncommon as it once was, but it's still a significant number, and I didn't really have any way of calibrating that.
I thought, "well, what am I supposed to do, now that I hit a million views?
How am I supposed to conceptualize that?
What is this YouTube thing anyways, that was once a repository for cute cat videos?
69
```

## Text Preprocessing

This step is long and iterative. We have to keep returning to the sample text and find the anomalies. Every NLP project is different.

I eventually arrive at these things to take care of:

1. lower case
2. punctuations in different coding method
3. contraction

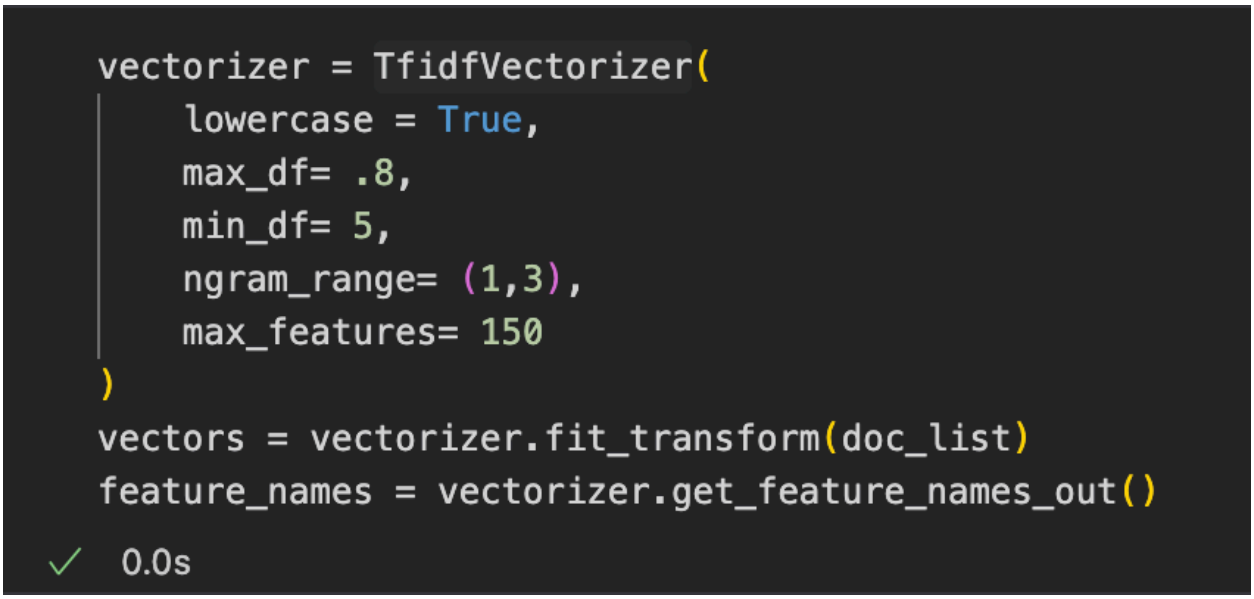
4. english stop words
5. lemmatization

and this is what the processed text looks like:


It's obviously quite hard to understand the text now after preprocessing. But all non-important words are removed. This is crucial before apply Scikit-Learn's vectorization.

**Vectorization**

The vectorization step sets the foundation for our modeling. There are few tunable parameters:




It's impossible to decide the exact settings for these parameters.  the max_df and min_df are commonly set as such among text recognition projects. ngram means the vectorization will recognize unigram, bigram and trigram occurrences. And the max_features means the maximum number of features to be included as vectors. This means that ALL DOCUMENTS will only be represented by a maximum of 150 words. I think 150 is a reasonable number for our current number of documents. As we expand our database, the number of features need to be increased accordingly.

```
print(feature_names)
```

✓ 0.0s

```
['absolutely' 'actually' 'allow' 'also' 'answer' 'anything' 'ask' 'become'
 'believe' 'big' 'biological' 'bit' 'book' 'call' 'cannot' 'case' 'cent'
 'certainly' 'chaos' 'claim' 'come' 'could' 'course' 'degree' 'difference'
 'element' 'error' 'especially' 'even' 'evidence' 'exactly' 'example'
 'fact' 'far' 'film' 'find' 'first' 'form' 'fundamental' 'get' 'go' 'god'
 'good' 'great' 'group' 'happen' 'hell' 'hierarchy' 'human' 'idea'
 'identity' 'intellectual' 'issue' 'keep' 'kind' 'know' 'law' 'learn'
 'least' 'leave' 'lecture' 'let' 'let say' 'life' 'look' 'lot' 'make'
 'manifest' 'maybe' 'mean' 'men' 'men woman' 'might' 'move' 'much' 'music'
 'never' 'new' 'obviously' 'oh' 'ok' 'one' 'one thing' 'order' 'part'
 'people' 'per' 'per cent' 'personality' 'place' 'play' 'political'
 'problem' 'produce' 'put' 'question' 'quite' 'radical' 'read' 'really'
 'reason' 'right' 'risk' 'say well' 'see' 'seem' 'sense' 'social'
 'society' 'something' 'something like' 'sort' 'speak' 'start' 'state'
 'story' 'student' 'suppose' 'system' 'take' 'talk' 'tell' 'test' 'theory'
 'there' 'theres' 'theyre' 'think well' 'time' 'true' 'try' 'two'
 'understand' 'university' 'use' 'want' 'wasnt' 'watch' 'way' 'whats'
 'woman' 'work' 'world' 'would' 'would say' 'write' 'wrong' 'yeah' 'year'
 'yes']
```

This printout shows all the features that will be used as vectors. We can immediately spot some features such as “absolutely”, “actually” — adverbs that don’t really help with identifying the meaning of the text. Then we have “oh”, “ok”, modal particles that don’t yield much meaning.

One option is to removal them from the “cleaned_text”. But before we apply the clustering model, completely removing these words can be premature. At this point, we choose not to remove them.

This is an example of:

1. original text

> Well, I think you don’t want to underestimate the role that technological transformation has played in this.
> I’ve been thinking about YouTube and podcasts quite intensely for about two years.
> I started putting my university lectures on YouTube in 2013.
> I did that for a variety of reasons—mostly curiosity and the drive to learn.
> I’ve found that, if I want to learn a technology, the best way to do it is to use it.
> I’m always learning new technologies—not that that makes me particularly unique—and I had some success with my lectures on public television in Canada.
> I did some lectures with this series called "Big Ideas" on Canadian public television.
> There’s about 200 of those lectures, and I did five of them.
> Two hundred done by 200 different people, but I did five of them, and they were regularly in the top 10 of the most viewed lectures.
> So I knew there was some broader market for, let’s say, ideas.
> I thought, "well, I might as well put my lectures up on YouTube, and see what happens."
> …
> I thought, "well, what am I supposed to do, now that I hit a million views?
> How am I supposed to conceptualize that?

2. Cleaned text

   What is this YouTube thing anyways, that was once a repository for cute cat videos?
   well think want underestimate role technological transformation play think youtube podcasts quite intensely two year start put university lecture youtube 2013 variety reason mostly curiosity drive learn find want learn technology best way use always learn new technology make particularly unique success lecture public television canada lecture series call big idea canadian public television theres 200 lecture five two hundred do 200 different people five regularly top 10 viewed lecture know broad market let say idea think well might well put lecture youtube see happen april 2016 million view think huh reason people watch want watch theyre actually really hard million something lot sell million copy book well first never happen right rare never scientific paper cite million time rarely million dollar large number well fair enough course uncommon still significant number really way calibrate think well suppose hit million view suppose conceptualize youtube thing anyways repository cute cat video

3. “vectorized” text

   ['actually', 'big', 'book', 'call', 'course', 'find', 'first', 'happen', 'idea', 'know', 'learn', 'lecture', 'let', 'let say', 'lot', 'make', 'might', 'never', 'new', 'people', 'play', 'put', 'quite', 'really', 'reason', 'right', 'see', 'something', 'start', 'suppose', 'theres', 'theyre', 'think well', 'time', 'two', 'university', 'use', 'want', 'watch', 'way', ‘year’]

## Modeling

We apply K-Means clustering. We need to decide another important parameter: the number of total clusters.

We set the K = 5,6,7,8,10, which is the number of potential topics touched on by each document.

```
true_k = 10
model = KMeans(n_clusters = true_k, init = 'k-means++', max_iter = 100, n_init=1, random_state=1)
model.fit(vectors)
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names_out()

with open (f'sample_topic_result_{true_k}.txt', 'w', encoding = 'utf-8') as f:
    for i in range(true_k):
        f.write(f"\nCluster {i}: ")
        for ind in order_centroids[i, :10]:
            f.write(f'{terms[ind]} ')
```

The more topics, or the closer the number of topics is to the total number of documents, the less meaningful is our clustering. This is because, on the extreme end, if there are 69 topics, i.e. every document has a topic of its own, then the reader might as well just read every document. Therefore, around or fewer than10 topics is a reasonable number to try to cluster the documents around.

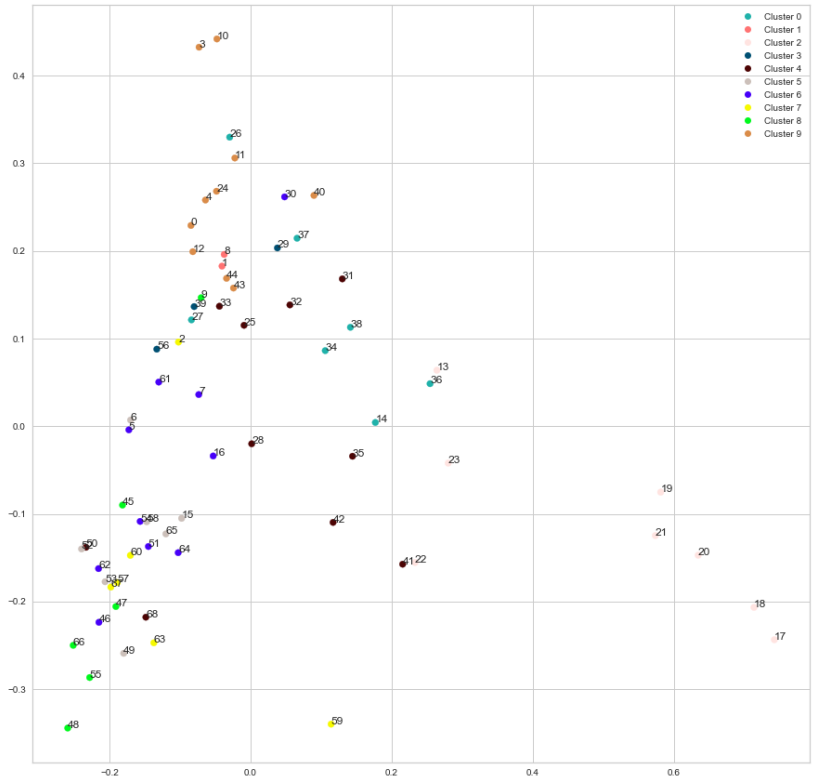
There is an example of what 10 clusters look like:

```
Cluster 0: leave radical know oh actually answer good happen exactly problem
Cluster 1: woman men difference men woman social would people society test say well
Cluster 2: question idea get theory right start yeah would happen people
Cluster 3: music film really ok right make work first watch see
Cluster 4: write book big yes talk time would much happen read
Cluster 5: hierarchy system people go chaos test order risk case error
Cluster 6: sense yes cannot right use form group idea people something
Cluster 7: people law political theyre wrong student reason hell would want
Cluster 8: per per cent cent leave error risk identity university there people
Cluster 9: mean learn story god part also sense something know go
```

This feel like a group of intelligible topics. The “meaningless” words we identified earlier are still here, “actually”, “oh”, “ok”, but they do not affect guessing the general meaning of the clusters.

E.g. cluster 0 could mean that the radical approach of answering problems; cluster 1 could mean the social difference between men and women.

To decide the number of clusters that provides the best separation of documents, we plot a PCA plot to identify the separation of the the documents

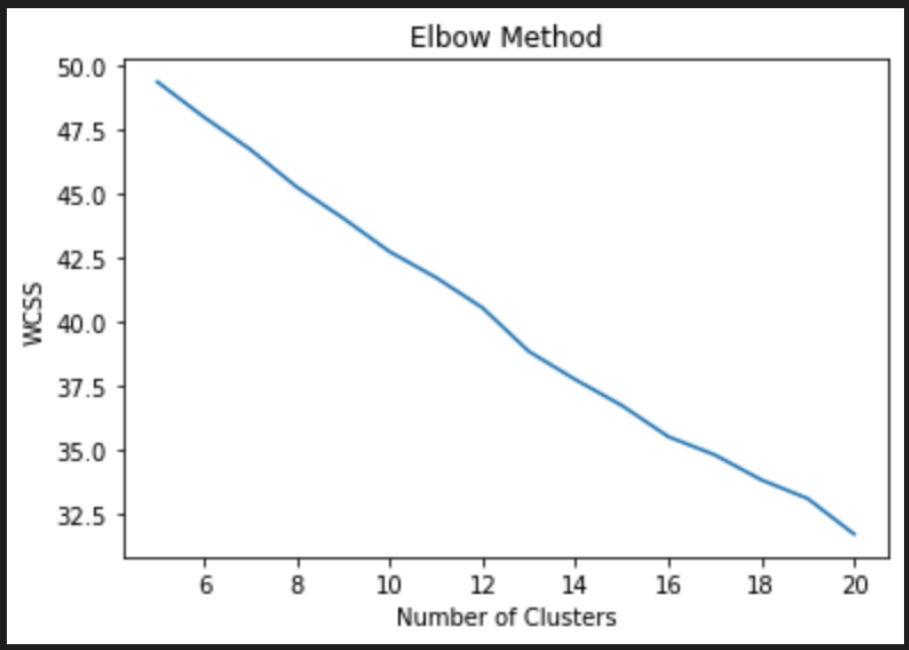



This PCA graph does not look very convincing. The different colors overlap on the PCA graph. The PCA graphs for other number of clusters are similar to this. No number of clusters is significantly better than others.

Another method of determining the ideal number of clustering is the “elbow method”, which is to find the within-cluster-sum-of-squares. The place with the significant dip is the sweet spot.

```
#Use the "Elbow Method" to determine the optimal true_k value
#withincluster sum of squares
wcss = []
for k in range (5,21):
    k_model = KMeans(n_clusters = k, init = 'k-means++', max_i
    k_model.fit(vectors)
    wcss.append(k_model.inertia_)
plt.plot(range(5,21), wcss)
plt.title("Elbow Method")
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
```

✓ 0.2s

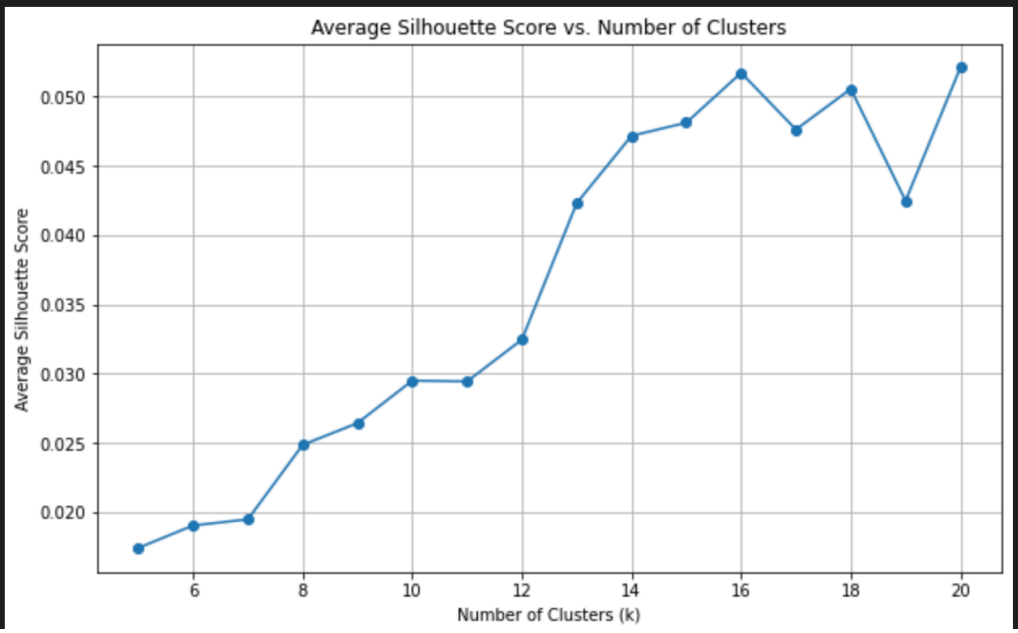



Unfortunately, we find no dip in the graph.

We lastly consult the average silhouette_score for each number of clusters

```python
from sklearn.metrics import silhouette_score
clusters_list =[]
silhouette_scores_list = []
for k in range(5,21):
    km = KMeans(n_clusters=k, init='k-means++', n_init=1, max_iter=100, random_state =1)
    cluster_labels = km.fit_predict(vectors)
    avg_silhouette_score = silhouette_score(vectors, cluster_labels)

    clusters_list.append(k)
    silhouette_scores_list.append(avg_silhouette_score)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(clusters_list, silhouette_scores_list, marker='o')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Average Silhouette Score")
plt.title("Average Silhouette Score vs. Number of Clusters")
plt.grid(True)
plt.show()
```

✓ 0.3s




The average silhouette score barely breaks 0.050, while the target for a good clustering is 0.5 or more.

There can only be two reasons for such a failure of deriving meaningful clustering:

1. The 25 sentence per document strategy is not working.
2. There is not meaningful cluster in these documents.

To address these two assessments, we need to:
1. Come up with new document generating strategy.
2. Expand our database to hopefully have meaningful clusters

## New Document Generating Strategy

An intuitively better way of generating documents is to break the text by natural paragraphs.

The transcripts themselves already have paragraphs, and each paragraph naturally is about one topic.

Thank you all very much for coming. It's really shocking to me that you don't have anything better to do on a Tuesday night. Seriously though, it's very strange, in some sense, that so many of you are here to listen to a sequence of lectures on the psychological significance of the Bible stories. It's something I've wanted to do for a long time, but it still does surprise me that there's a ready audience for it. That's good. We'll see how it goes.

I'll start with this, because it's the right question: why bother doing this? And I don't mean why should I bother—I have my own reasons for doing it—but you might think, 'why bother with this strange old book at all?' That's a good question. It's a contradictory document that's been cobbled together over thousands of years. It's outlasted many, many kingdoms. It's really interesting that it turns out a book is more durable than stone. It's more durable than a castle. It's more durable than an empire. It's really interesting that something so evanescent can be so long-living. So there's that; that's kind of a mystery.

I'm approaching this whole scenario, the Biblical stories, as if they're a mystery, fundamentally because they are. There's a lot we don't understand about them. We don't understand how they came about. We don't really understand how they were put together. We don't understand why they had such an unbelievable impact on civilization. We don't understand how people could have believed them. We don't understand what it means that we don't believe them now, or even what it would mean if we did believe them. On top of all that, there's the additional problem—which isn't specific to me, but is certainly relevant to me—that, no matter how educated you are, you're not educated enough to discuss the psychological significance of the Biblical stories. But I'm going to do my best, partly because I want to learn more about them. One of the things I've learned is that one of the best ways to learn about something is to talk about it. When I'm lecturing, I'm thinking. I'm not trying to tell you what I know for sure to be the case, because there's lots of things that I don't know for sure to be the case. I'm trying to make sense out of this, and I have been doing this for a long time.

You may know, you may not, that I'm an admirer of Nietzsche. Nietzsche was a devastating critic of dogmatic Christianity—Christianity as it was instantiated in institutions. Although, he is a very paradoxical thinker. One of the things Nietzsche said was that he didn't believe the scientific revolution would have ever got off the ground if it hadn't been for Christianity—and, more specifically, for Catholicism. He believed that, over the course of a thousand years, the European mind had to train itself to interpret everything that was known within a single coherent framework—coherent if you accept the initial axioms. Nietzsche believed that the Catholicization of the phenomena of life and history produced the kind of mind that was then capable of transcending its dogmatic foundations, and concentrating on something else. In this particular case, it happened to be the natural world.

Nietzsche believed that Christianity died of its own hand, and that it spent a very long time trying to attune people to the necessity of the truth, absent the corruption, and all that—that's always part of any human endeavour. The truth—the spirit of truth—that was developed by Christianity turned on the roots of Christianity. Everyone woke up and said, or thought, something like, 'how is it that we came to believe any of this?' It's like waking up one day and noting that you really don't know why you put a Christmas tree up, but you've been doing it for a long time, and that's what people do. There are reasons Christmas trees came about. The ritual lasts long after the reasons have been forgotten.

Nietzsche was a critic of Christianity, and also a champion of its disciplinary capacity. The other thing that Nietzsche believed was that it was not possible to be free unless you had been a slave. By that, he meant that you don't go from childhood to full-fledged adult individuality: you go from child to a state of discipline, which you might think is akin to self-imposed slavery. That would be the best scenario, where you have to discipline yourself to become something specific, before you might be able to reattain the generality you had as a child. He believed that Christianity had played that role for Western civilization. But, in the late 1800s, he announced that God was dead.

You often hear of that as something triumphant, but, for Nietzsche, it wasn't. He was too nuanced a thinker to be that simpleminded. Nietzsche understood—and this is something I'm going to try to make clear—that there's a very large amount that we don't know about the structure of experience—that we don't know about reality—and we have our articulated representations of the world. Outside of that, there are things we know absolutely nothing about. There's a buffer between them, and those are things we sort of know something about. But we don't know them in an articulated way.

So we devise a plan to use each paragraph as a separate document. But there is an important exception: each paragraph needs to be more than 10 sentences. This is because we don't want to use "yes" or "ok" as a single document, as they can often be in a paragraph of their own in interview transcript.

This is how we will divide the documents:

```python
def divide_paragraphs(input_text, sentences_threshold=10):
    # Tokenize the input text into paragraphs
    paragraphs = input_text.split("\n")

    # Initialize variables
    current_document = ""
    document_count = 1
    paragraph_list = []

    for paragraph in paragraphs:
        current_document += paragraph + "\n"
        num_sentences = len(nltk.sent_tokenize(current_document))
        if num_sentences >= sentences_threshold:
            #save_to_file(current_document, document_count)
            #print(document_count,':',current_document)
            paragraph_list.append(current_document)
            document_count += 1
            current_document = ""

    return paragraph_list
```

✓ 0.0s

# Expanding Data Base

We import all the available transcripts from Dr. Peterson's website, 26 in total.

```python
# List of file names you want to direct to the folder
file_names = ['Abraham_Father_of_Nations.txt',
'Adam_and_Eve.txt',
'Aspen_Barricades_of_Culture_Wars.txt',
'Aubrey_Marcus_Truth_Responsibility.txt',
'Ben_Shapiro.txt',
'Cain_and_Abel.txt',
'Call_to_Abraham.txt',
'Chaos_Order.txt',
'DR_Iain_McGilchrist.txt',
'Death_Resurrection.txt',
'Douglas_Murray_IQ_Politics_and_the_Left.txt',
'God_and_Hierarchy.txt',
'Great_Sacrifice_Abraham_Isaac.txt',
'Jocab_Wresting_with_God.txt',
'Jocabs_Ladder.txt',
'Joseph_Coat_of_Many_Colors.txt',
'Lewis_Howes_Pain_Suffering.txt',
'Lewis_Howes_Responsibility_Meaning.txt',
'Noah_and_the_Flood.txt',
'Oxford_Union_Address.txt',
'Phenomenology_of_the_Divine.txt',
'Psychology_of_the_Flood.txt',
'Russell_Brand_Freedom_Tyranny.txt',
'Sodom_Gomorrah.txt',
'The_Idea_of_God.txt',
'ideology_logos_belief.txt']
```

Now we have 1638 documents, each containing a paragraph that is at least ten sentences

```
# Divide into smaller documents with at least 10 sentences each
all_smaller_documents = []
for text in texts:
    smaller_documents = divide_paragraphs(text)
    all_smaller_documents.extend(smaller_documents)
print(len(all_smaller_documents))
```

✓ 0.8s

```
1638
```

The preprocessing step remains the same. But for the vectorization step, I decide to include 300 feature words instead of 150 because of the much larger data size.

```
vectorizer = TfidfVectorizer(
    lowercase = True,
    max_df= .8,
    min_df= 5,
    ngram_range= (1,3),
    max_features= 300
)
vectors = vectorizer.fit_transform(new_doc_list)
feature_names = vectorizer.get_feature_names_out()
print(feature_names)
```

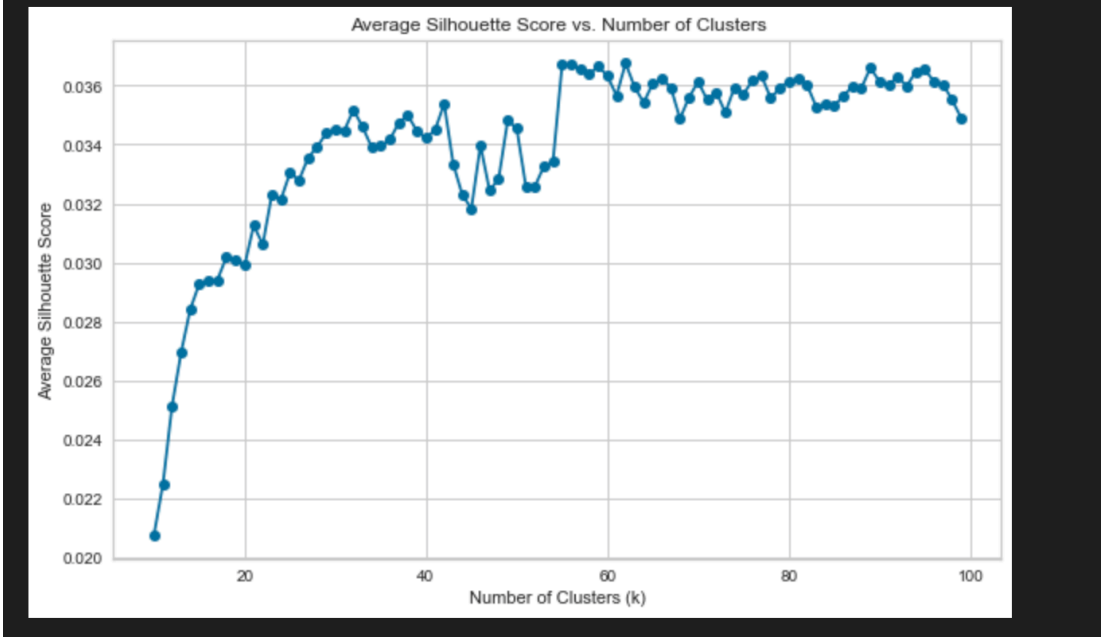
✓ 1.1s

```
['able' 'abraham' 'absolutely' 'across' 'act' 'actually' 'adam' 'aim'
 'along' 'already' 'also' 'although' 'always' 'animal' 'another' 'answer'
 'anything' 'anyways' 'around' 'ask' 'associate' 'away' 'back' 'bad'
 'become' 'begin' 'being' 'believe' 'best' 'big' 'bit' 'bloody' 'body'
 'book' 'bring' 'brother' 'build' 'cain' 'call' 'cannot' 'care' 'case'
 'catastrophe' 'certainly' 'chaos' 'child' 'christ' 'come' 'consciousness'
 'consequence' 'could' 'course' 'culture' 'damn' 'day' 'deal' 'death'
 'degree' 'develop' 'die' 'difference' 'different' 'difficult' 'do'
 'dream' 'earth' 'easy' 'eat' 'element' 'else' 'end' 'enough' 'especially'
 'even' 'ever' 'every' 'everyone' 'everything' 'evil' 'exactly' 'example'
 'exist' 'experience' 'eye' 'face' 'fact' 'fall' 'family' 'far' 'father'
 'figure' 'find' 'first' 'form' 'forward' 'fundamental' 'future' 'game'
 'get' 'give' 'go' 'god' 'good' 'great' 'guy' 'hand' 'happen' 'happy'
 'hard' 'hell' 'here' 'hierarchy' 'high' 'house' 'human' 'human being'
 'idea' 'ideal' 'imagine' 'important' 'individual' 'interest'
 'interesting' 'issue' 'jacob' 'joseph' 'jung' 'keep' 'kid' 'kind' 'know'
 'land' 'large' 'last' 'lay' 'learn' 'least' 'leave' 'lecture' 'let'
 'let say' 'level' 'life' 'like' 'line' 'little' 'live' 'long' 'look'
 'lord' 'lot' 'make' 'man' 'manifest' 'manner' 'many' 'matter' 'maybe'
 'mean' 'men' 'might' 'mind' 'moral' 'mother' 'move' 'much' 'name'
 'nature' 'need' 'never' 'new' 'next' 'noah' 'nothing' 'number'
 'obviously' 'often' 'oh' 'ok' 'old' 'one' 'one thing' 'open' 'order'
 'part' 'partly' 'people' 'perhaps' 'person' 'perspective' 'place' 'play'
 'point' 'possible' 'potential' 'power' 'pretty' 'probably' 'problem'
 'produce' 'proper' 'properly' 'psychological' 'put' 'question' 'quite'
...
 'turn' 'two' 'understand' 'unto' 'upon' 'use' 'value' 'walk' 'want'
 'wasnt' 'watch' 'water' 'way' 'well' 'whatever' 'whats' 'whole' 'wife'
 'within' 'without' 'woman' 'word' 'work' 'world' 'would' 'would say'
 'write' 'wrong' 'yeah' 'year' 'yes']
```

We then check the average silhouette score:

```
from sklearn.metrics import silhouette_score
clusters_list =[]
silhouette_scores_list = []
for k in range(10,100):
    km = KMeans(n_clusters=k, init='k-means++', n_init=1, max_iter=100, random_state =1)
    cluster_labels = km.fit_predict(vectors)
    avg_silhouette_score = silhouette_score(vectors, cluster_labels)

    clusters_list.append(k)
    silhouette_scores_list.append(avg_silhouette_score)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(clusters_list, silhouette_scores_list, marker='o')
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Average Silhouette Score")
plt.title("Average Silhouette Score vs. Number of Clusters")
plt.grid(True)
plt.show()
```

✓ 18.1s




The result is rather surprising. The average silhouette score became worse than from the sampling data.

This means that KMeans is not an appropriate method of topic modeling for this data. It's either that the text is highly individual or we need more advanced method of differentiating the documents.

## Advanced Modeling

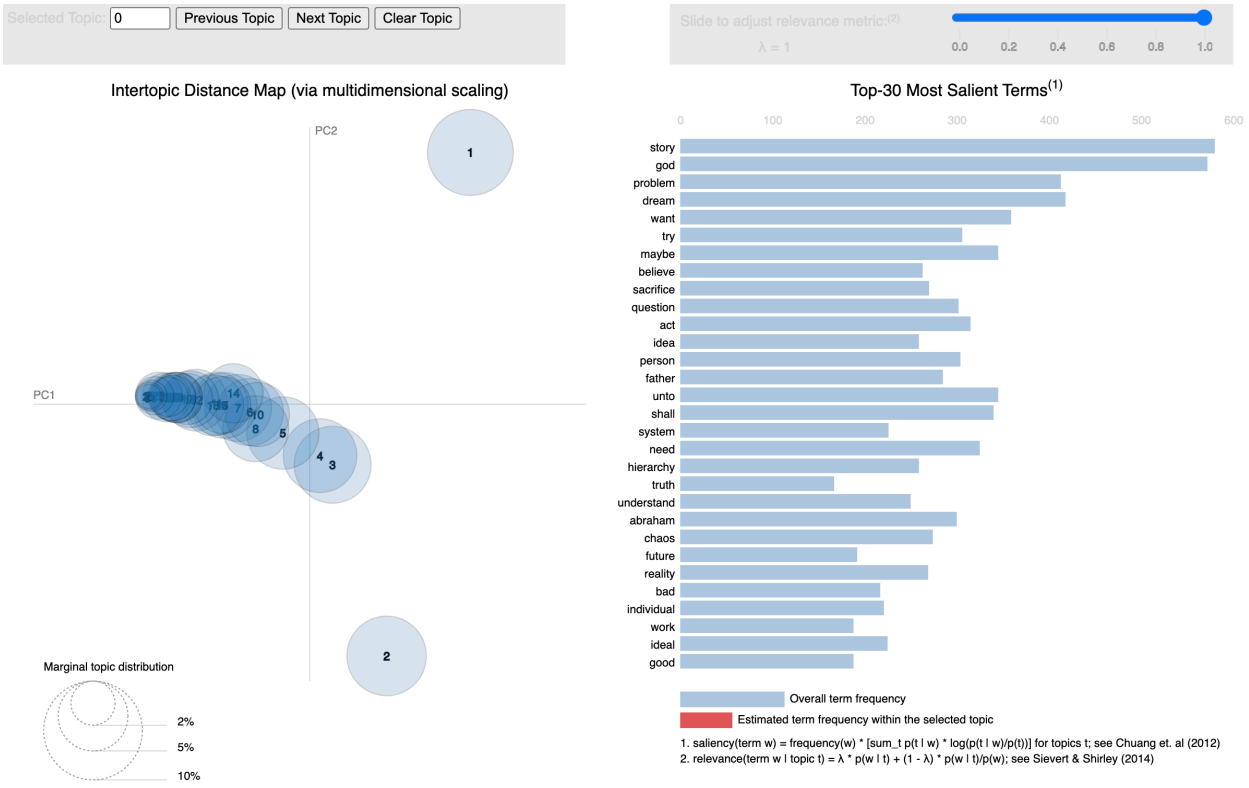
A more advanced modeling method is Latent Dirichlet Allocation Modeling. We won't have average silhouette score as a simple metric to validate the result. But we can use pyLDAvis package to visualize the data on a PCA graph to judge the document separation.

```
#TF-IDF REMOVAL
from gensim.models import TfidfModel
id2word = corpora.Dictionary(doc_bigrams_trigrams)

corpus = [id2word.doc2bow(doc) for doc in doc_bigrams_trigrams]

tfidf = TfidfModel(corpus, id2word= id2word)

#Filter low value words and also words missing in tfidf models.

low_value = 0.1

for i in range(0, len(corpus)):
    bow = corpus[i]
    low_value_words = [] #reinitialize to be safe. You can skip this.
    tfidf_ids = [id for id, value in tfidf[bow]]
    bow_ids = [id for id, value in bow]
    low_value_words = [id for id, value in tfidf[bow] if value < low_value]
    words_missing_in_tfidf = [id for id in bow_ids if id not in tfidf_ids] # The words with tf-idf so

    new_bow = [b for b in bow if b[0] not in low_value_words and b[0] not in words_missing_in_tfidf]

    # reassign
    corpus[i] = new_bow
```

✓ 0.7s

```
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=30,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=20,
                                            alpha="auto")
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(lda_model, corpus, id2word)
vis
```

We can see that vectorization part in Gensim is very similar to Scikit-Learn's.
The result, however, seems equally bad.




This is the PCA graph when the number of topics is set to 30. There is so much overlap for most of the topics. This is obviously not a good way of categorizing the documents.

# Conclusion:

Although we don't arrive at a convenient topic modeling result, this is still a meaningful experiment in natural language processing. We employed numerous methods to clean the data, met with the interesting challenge of breaking a long transcript into smaller chunks, and applied many different tests to evaluate the Kmeans clustering result. We also experimented with the more sophisticated Gensim LDA model.

Not having a clear topic definition only means that the text we are dealing with is highly individualistic, and all the tools we have at hand are not capable of finding appropriate topics for it. However, that is not to say that there isn't any other unsupervised topic modeling method out there that can achieve what we set out to do in the beginning. And I'm eager to find out about it in the future.