Ziqi Zhao, CSCI S-108 Data Mining, Discovery and Exploration with Professor Stephen Elston

FINAL PROJECT REPORT - AI VS HUMAN ESSAY CLASSIFCATION

I. Abstract/Executive Summary

A combination of non-supervised machine learning techniques were used build a classifier that attempts to classify AI vs human written essays without any labelling.

The best technique, a combination of either K-means or agglomerative clustering (k=2) with cosine similarity applied to linear PCA (components =2) results yielded a binary classification accuracy of 97%, a result that is surprising given initially low expectations. It is uncertain whether this can scale with larger datasets given the compute limitations of the author.

The dataset consisted of 44,868 essays compiled in one single .csv file, with around 60% written by precollege students from grades 6-12 and 40% generated by 16 different AI large language models. All AI and human written essays were written on topics from 15 different prompt topics. The dataset turned out to be too compute heavy for the compute resources available, thus a reduced dataset with 3,998 essays were used with approximately 50% human and 50% AI written essays.

The essay data was preprocessed using stopwords filtering and TF-IDF vectorization was used prior to classification. Node2vec was attempted to preserve non-Euclidean distances but was again too compute heavy and abandoned.

Initial naive K-means clustering with k =2 showed a classifcation accuracy of only 26%, with the confusion matrix showing a large number of false negatives (positive = 1, or AI written), incorrectly classifying AI written-essays as human written.

K-means clustering with 2 PCA components improved accuracy to 73%, with almost 97% of all AI-written essays being classified correctly and nearly all incorrect classifications being false positives, incorrectly classifying human written essays as AI written. We hypothesize the high skew to false positives may be due to skew in the reduced dataset, whether out of the 15 different prompt topics only 2 had human written essays, while AI written essays spanned all 15 topics.

Optimal Hyperparameter Selection in Clustering was then applied with WCSS, BCSS, Silhoutte Scores and Calinski-Harabasz Scores calculated to find the optimal cluster size. The Silhouette Score is quite low across all cluster numbers, with values ranging up to 0.08. These low values indicate poor clustering quality. In addition to k=2, multiple peaks were seen around k=11, k=16, and k=15 was applied and plotted (still with PCA applied), showing concentrated and well defined clusters. We hypothesize this may be due to the 15 different prompt topics, having distinct concentrations of unqiue words, and a simple LDA (LatentDirichletAllocation) was applied to show the top 10 words in

top 10 topics, which looked highly similar to the prompt topics and further confirmed our hypothesis.

In terms of non-Euclidean distances, CLARA (k-mediod approximation) with UMAP applied were then attempted, including an optimal parameter search for CLARA. The resulting Silhouette Coefficient was again very low at below 0.1, suggesting poor cluster quality. These methods did not seem to improve the classification accuracy and when plotting the dimensionality reduction of the UMAP and CLARA clustering, two clusters are always evident, but the false positive cluster was also always clearly evident in its own cluster neighbouring the AI-written essays cluster, similar to results from the naive K-means with PCA, as this seems to be a trend.

Finally, agglomerative clustering (k=2) with cosine similarity applied to linear PCA (components =2) results was done. This succesfully reduced the false positives to nearly 0 and resulted in a 97% accuracy. The cosine similarity matrix seemed to be key as replacing aggolmeative clustering with simple K-means also results in 97% accuracy. Note the cosine metric was also used in UMAP, and a Kernal PCA with a cosine kernal were also tried but did not improve the accuracy further from 73%. It appears that a separate cosine similarity matrix must be generated post dimensionality reduction, then fitted to the clustering method in order to achieve this effect. See discussion for further details.

Given the high accuracy achieved at 97%, no further methods were attempted. Theoretically sound but highly compute intensive methods such as spectral clustering were noted but not attempted.

II. Problem Definition

In the rapidly evolving field of generative AI, distinguishing AI-generated text from human-written content remains a critical challenge with implications in academia, content moderation, and digital content creation. While considerable progress has been made identifying human generated fake vs real content such as fake vs real news, distinguishing AI-generated text from human-written content remains a challenge.

In January 2023, OpenAI had released an AI classifier for indicating AI-written text. However, it was shut down in July 2023 due to low accuracy rates[1]. This hints at the challenging nature of this problem when applied on a very large scale.

This project aims to apply non-supervised machine learning techniques to build a classifier that attempts to classify AI vs human produced essays, without any labelling.

[1] https://openai.com/index/new-ai-classifier-for-indicating-ai-written-text/

III. Initial Exploratory Data Analysis (EDA) and Data Preparation

There are many datasets available that collect human vs AI generated data. Upon exploration, I have chosen the DAIGT V2 Train Dataset[2]. The dataset employed in this project comprises approximately 44,868 essays compiled in one single .csv file, roughly equally distributed between AI-generated and human-written texts.

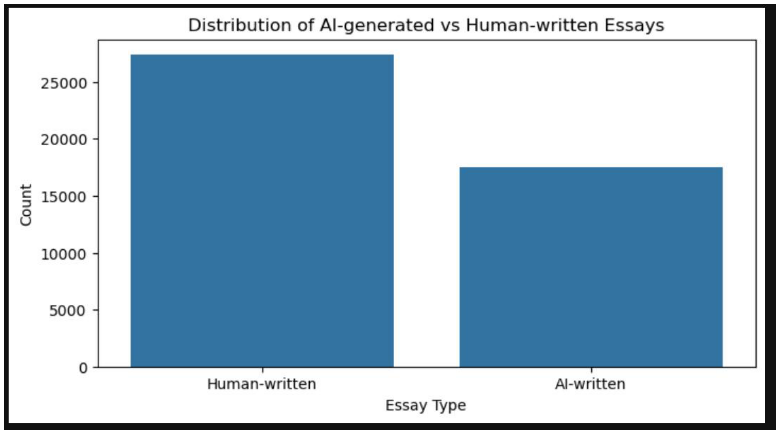


Figure 1. Distribution of AI-written vs Human-written Essays

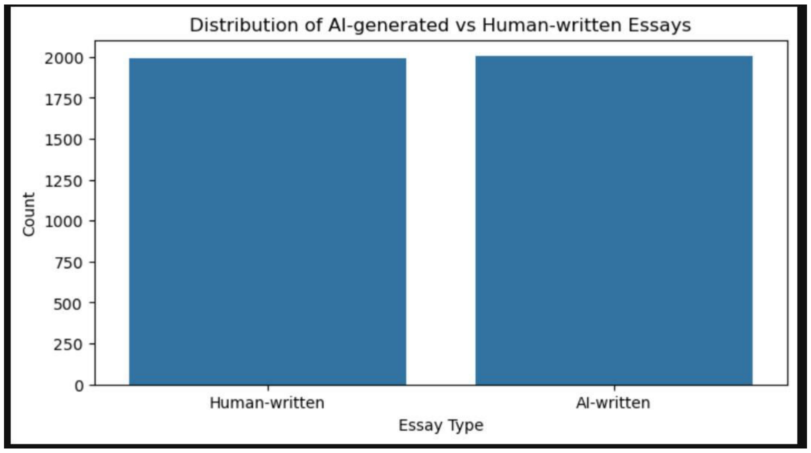


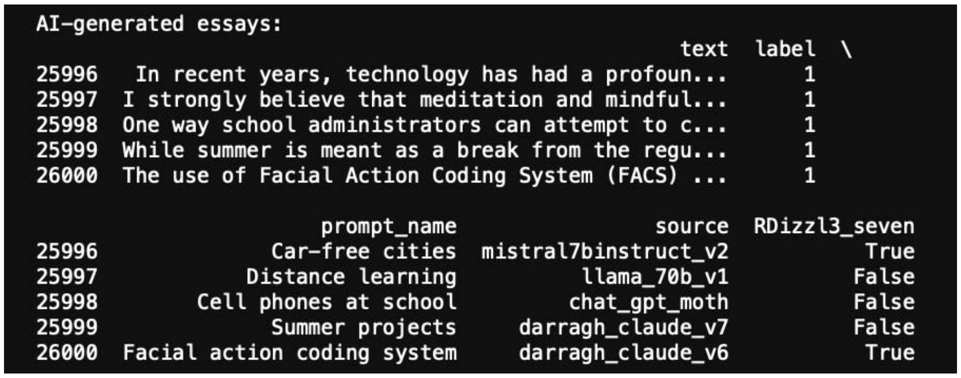
Figure 2. Distribution of AI-written vs Human-written Essays for the reduced size dataset used

Given the large dataset and local compute limitations and that a simple 2-component PCA run on pre-processed text takes 10+ minutes for a single run. This project uses a “reduced dataset” that has 3,998 essays, with around 50% human and 50% AI written.

Each essay is associated with the original metadata shown below including its label (1 for AI and 0 for human), prompt, source dataset, and length. This dataset's balanced

2 https://www.kaggle.com/datasets/thedrcat/daigt-v2-train-dataset

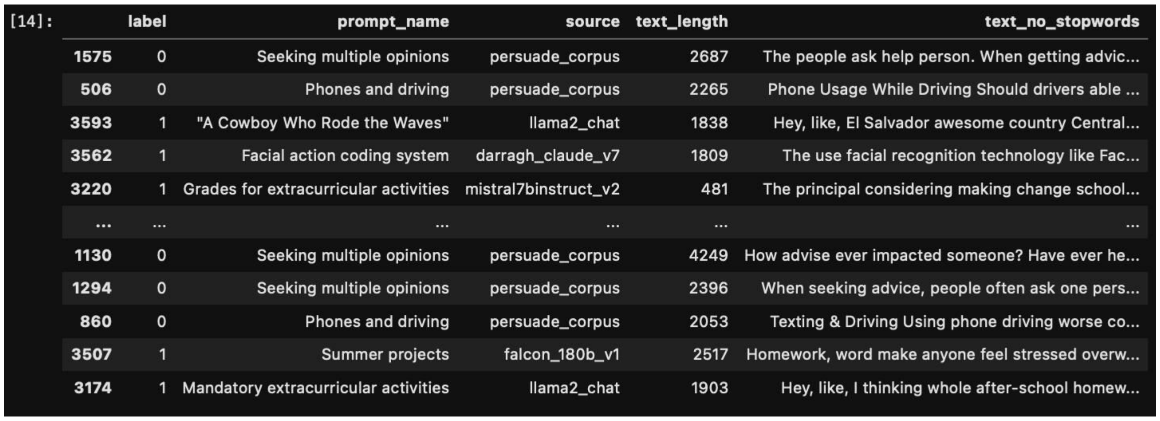
nature and rich metadata makes it ideal for exploration with unsupervised learning methods. (**Note: all labels were ignored in applying techniques used only in validation.)**



```
AI-generated essays:
                                                    text  label  \
25996   In recent years, technology has had a profoun...      1
25997  I strongly believe that meditation and mindful...      1
25998  One way school administrators can attempt to c...      1
25999  While summer is meant as a break from the regu...      1
26000  The use of Facial Action Coding System (FACS) ...      1

                    prompt_name                source  RDizzl3_seven
25996             Car-free cities  mistral7binstruct_v2           True
25997           Distance learning           llama_70b_v1          False
25998        Cell phones at school          chat_gpt_moth          False
25999             Summer projects      darragh_claude_v7          False
26000  Facial action coding system      darragh_claude_v6           True
```

Figure 3. Snapshot of original dataset (AI-generated essays) pre-processing

(Side Note: RDizzl3_seven feature — This is a boolean value feature present in the DAIGT-V2 dataset that determines whether essays were written based on prompts included in the hidden test set of the original Kaggle competition that this dataset was originally used for. We will ignore this column completely in this project.)



[14]:

	label	prompt_name	source	text_length	text_no_stopwords
1575	0	Seeking multiple opinions	persuade_corpus	2687	The people ask help person. When getting advic...
506	0	Phones and driving	persuade_corpus	2265	Phone Usage While Driving Should drivers able ...
3593	1	"A Cowboy Who Rode the Waves"	llama2_chat	1838	Hey, like, El Salvador awesome country Central...
3562	1	Facial action coding system	darragh_claude_v7	1809	The use facial recognition technology like Fac...
3220	1	Grades for extracurricular activities	mistral7binstruct_v2	481	The principal considering making change school...
...	...	...	...	...	...
1130	0	Seeking multiple opinions	persuade_corpus	4249	How advise ever impacted someone? Have ever he...
1294	0	Seeking multiple opinions	persuade_corpus	2396	When seeking advice, people often ask one pers...
860	0	Phones and driving	persuade_corpus	2053	Texting & Driving Using phone driving worse co...
3507	1	Summer projects	falcon_180b_v1	2517	Homework, word make anyone feel stressed overw...
3174	1	Mandatory extracurricular activities	llama2_chat	1903	Hey, like, I thinking whole after-school homew...

Figure 4. Snapshot of processed dataset with unused or repeat columns removed

As shown in Figure 4:

- The data was split in 80%/20% but the entire dataset was used for almost all tasks.
- "Stopwords" were filtered out, or common English words that contributes little to sentence meaning using the NLTK Python package.
- The essays were then vectorized using a TF-IDF vectorizer, and **only** the vectorized essay data was passed into subsequent clustering/dimensionality reduction, etc.

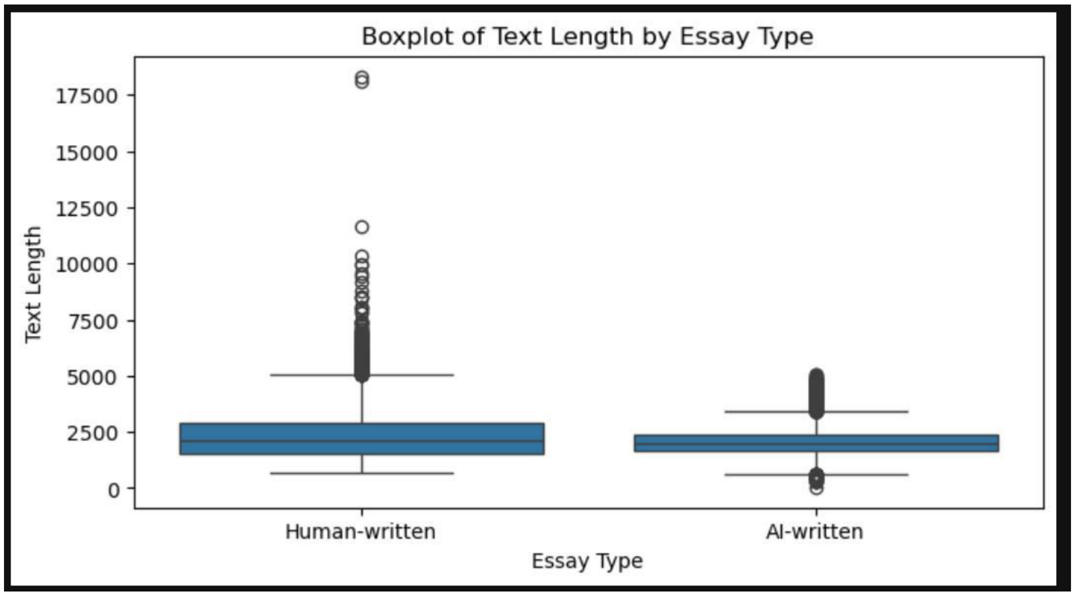


Figure 5. Boxplot of AI vs human written essay length.
The longer the essay length the more likely it was written by a human.

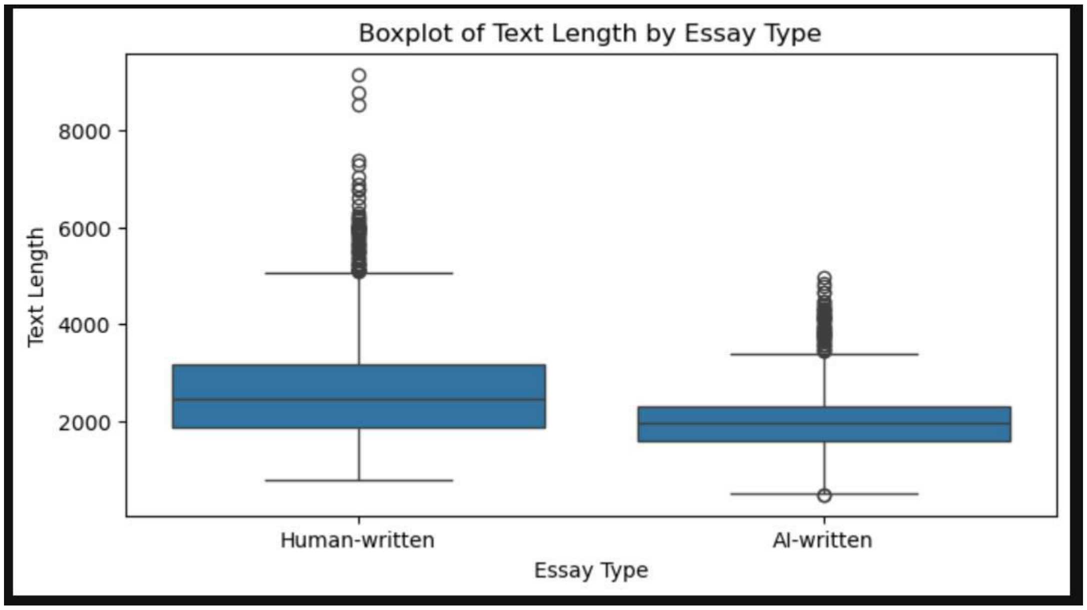


Figure 6. Boxplot of AI vs human written essay length for reduced dataset.
There are less outliers with less extremes in terms of long essay lengths.

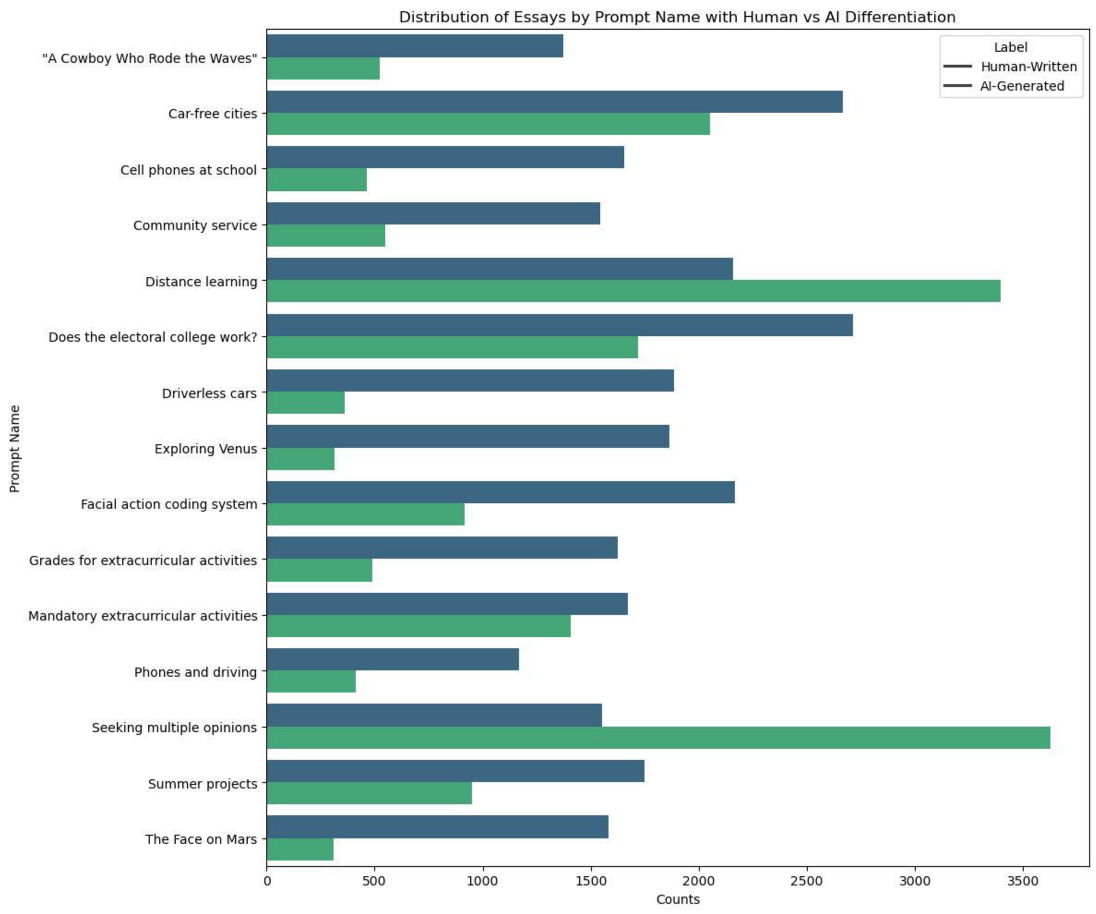


Figure 7. There are 15 different prompts used in essay topic, with some having more human-written essays and some more AI-written essays.

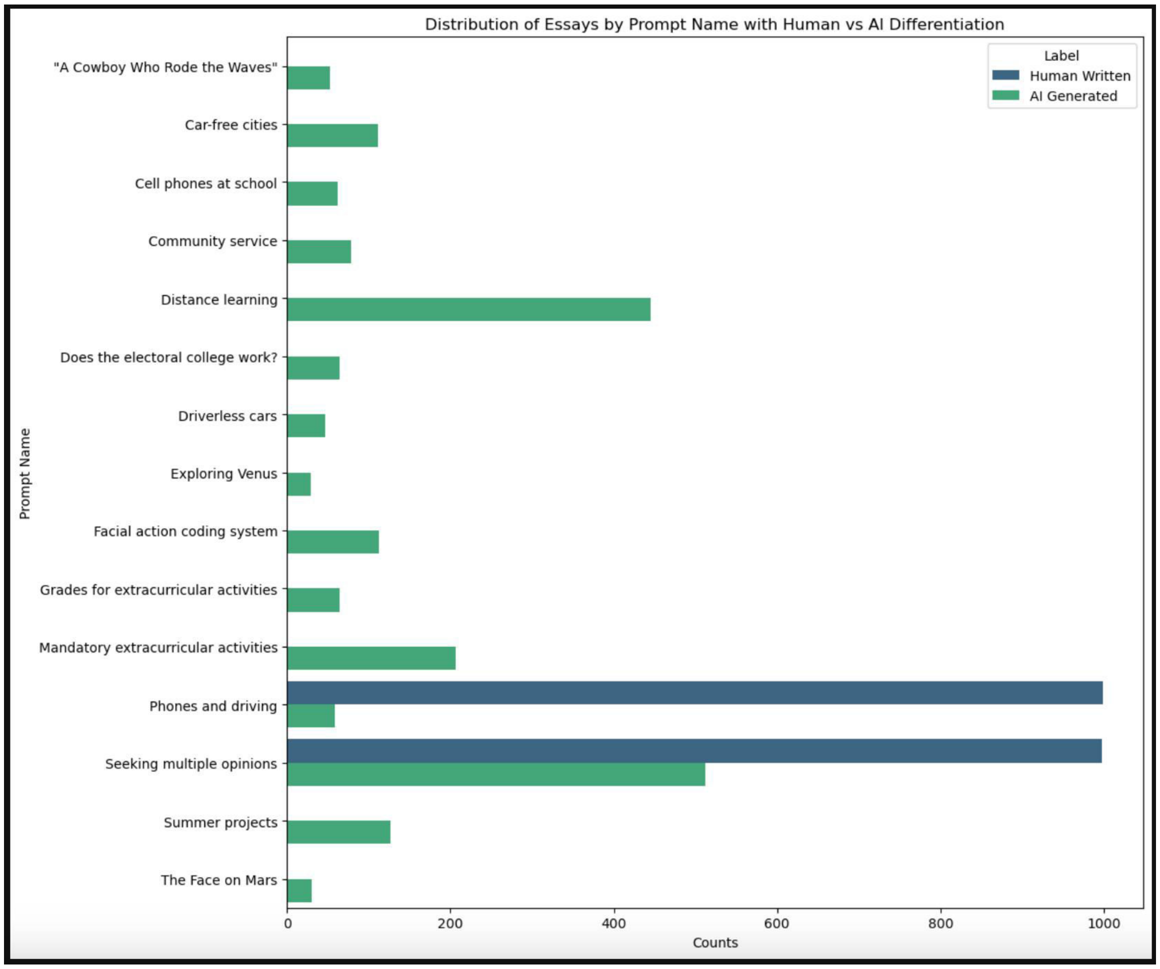


Figure 8. All 15 different prompts are still seen in the reduced dataset. Now only 2 of the 15 different prompts have human-written essays.

Figures 7,8 - Human-written essays were taken from the Persuasive Essays for Rating, Selecting, and Understanding Argumentative and Discourse Elements (PERSUADE) corpus. The PERSUADE corpus is large-scale corpus of writing with annotated discourse elements[3]. The below is a summary of the PERSUADE corpus used:

“The PERSUADE corpus comprises two sub-corpora consisting of source-based essays (n = 12,875) and independent essays (n = 13,121). Source-based writing requires the student to refer to a text while independent writing excludes this requirement. The source-based set was derived from seven unique writing prompts and related sources. The writing reflects students in grades 6 through 10. The independent set reflects writing

[3] https://www.sciencedirect.com/science/article/pii/S1075293522000630

where background knowledge of the topic was not a requirement, and no sources were required to produce the texts. The independent sub-corpus was collected from students in grades 8 through 12, and the collection was derived from eight unique writing prompts. All prompts and sources are available within the PERSUADE corpus.

The PERSUADE corpus was limited to essays with a minimum of 150 words of which 75% had to be correctly spelled American English words."

There are many types of large language models ("LLM") represented in the dataset, for example:

- ChatGPT by openAI, the most well-known and most adopted LLM
- Llama, a mostly open source LLM lead by Meta
- Mistral, an LLM that uses a fairly new "Mixture of Experts" model architechture
- Cohere Command, an LLM that uses retrieval-augmented generation to better handle longer texts.

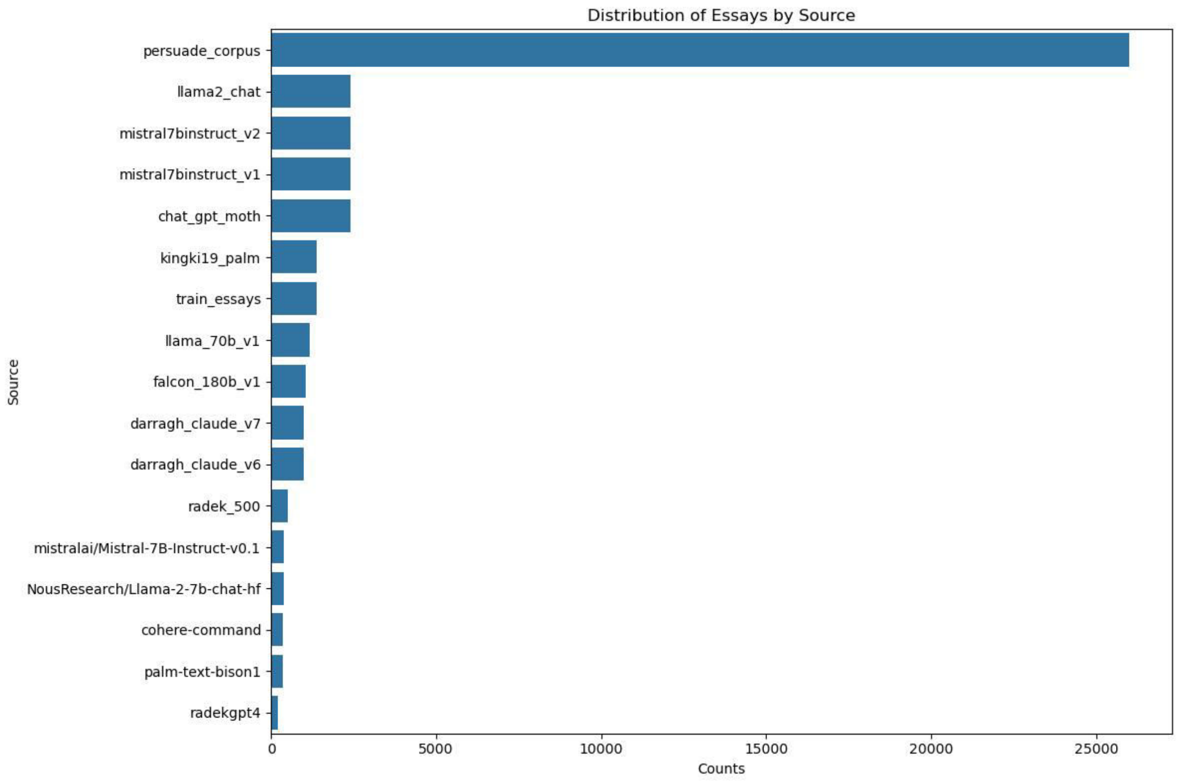


Figure 9. There are over 15 different large language models (LLMs) represented, while the human-written essays are all from the persuade_corpus.

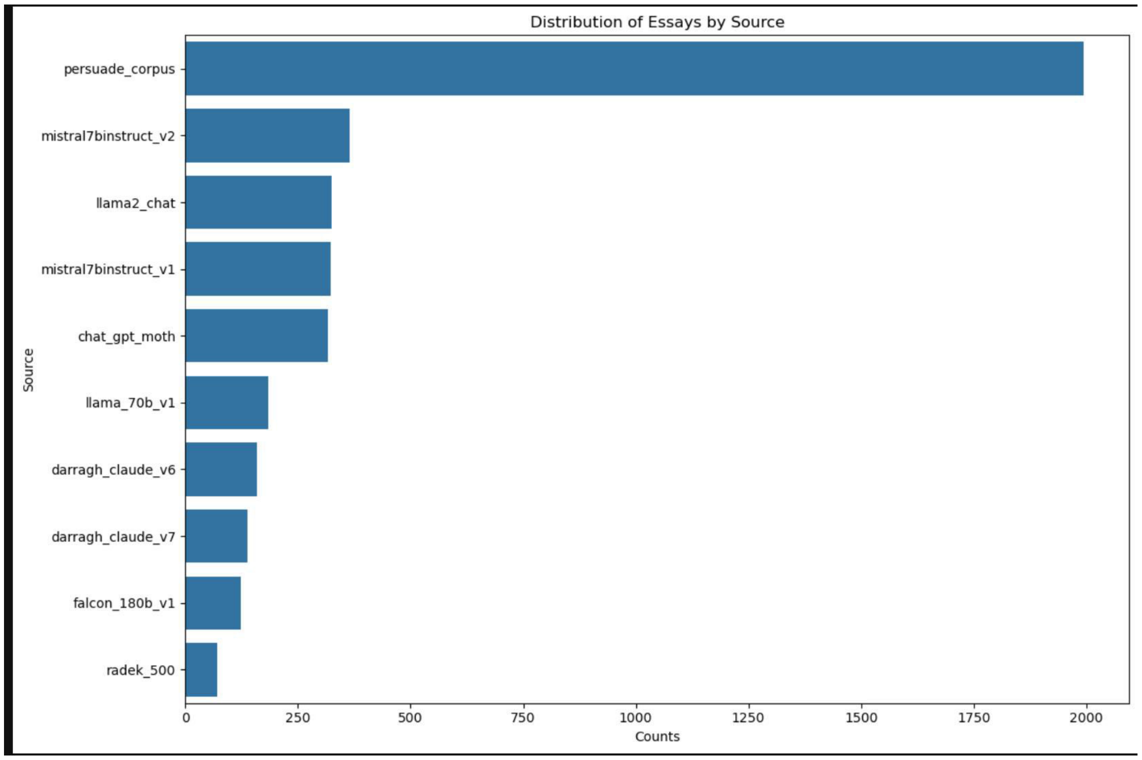
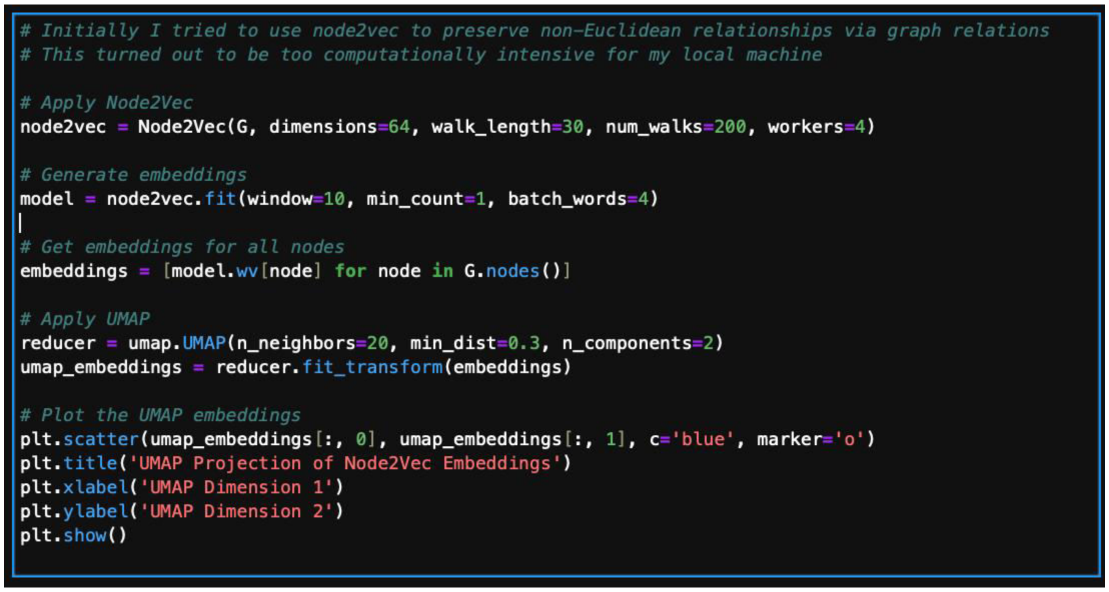


Figure 10. Only 9 different large language models (LLMs) are represented in the reduced dataset.

IV. Clustering with Dimensionality Reduction Methodology and Results

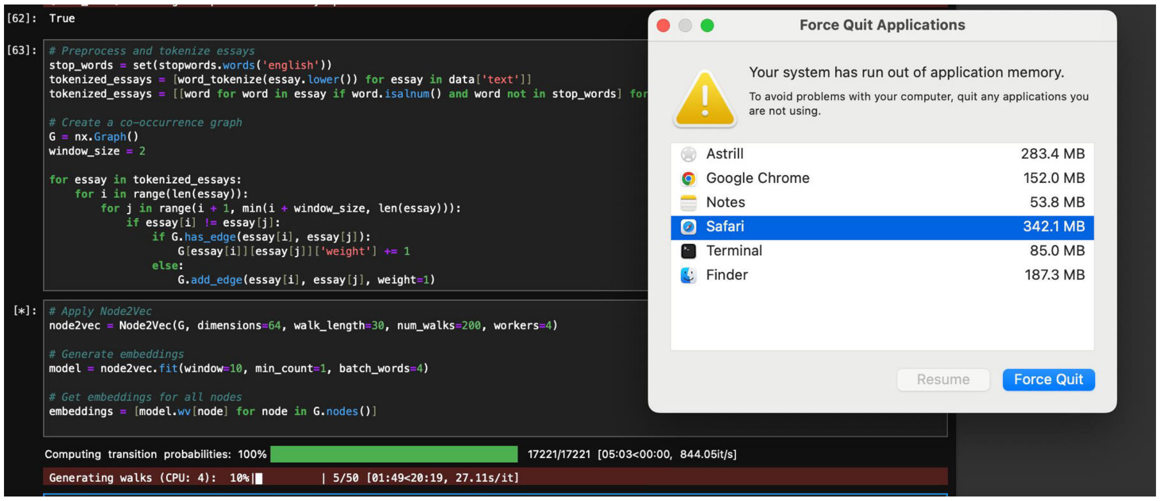
First, a precursory note on the high-dimensionality and data nature. The advice given on the proposal was noted, which was that PCA would likely not work well and methods such as UMAP are better with potential non-Euclidean distances being used. However, given that the dataset used is over 3,000 text essays only, standard vectorization processing techniques that convert these essays to numerical representations are nearly all Euclidean.

I did some research and found Node2Vec, which generates vector representations of nodes on a graph, but is also well known in applications on text corpus given advantages in semantic linkages via graph for text meaning. Node2Vec would have given non-Euclidean embeddings and could also be combined with K-means clustering for visualization and UMAPs.



```
# Initially I tried to use node2vec to preserve non-Euclidean relationships via graph relations
# This turned out to be too computationally intensive for my local machine

# Apply Node2Vec
node2vec = Node2Vec(G, dimensions=64, walk_length=30, num_walks=200, workers=4)

# Generate embeddings
model = node2vec.fit(window=10, min_count=1, batch_words=4)

# Get embeddings for all nodes
embeddings = [model.wv[node] for node in G.nodes()]

# Apply UMAP
reducer = umap.UMAP(n_neighbors=20, min_dist=0.3, n_components=2)
umap_embeddings = reducer.fit_transform(embeddings)

# Plot the UMAP embeddings
plt.scatter(umap_embeddings[:, 0], umap_embeddings[:, 1], c='blue', marker='o')
plt.title('UMAP Projection of Node2Vec Embeddings')
plt.xlabel('UMAP Dimension 1')
plt.ylabel('UMAP Dimension 2')
plt.show()
```

Figure 11. Some initial code I wanted to run with Node2Vec (synthetic graph generation not shown). The hyperparameters could potentially been reduced further for efficient computation, but at the potential cost of mis-representation and overcompression.



```
[62]:  True

[63]:  # Preprocess and tokenize essays
       stop_words = set(stopwords.words('english'))
       tokenized_essays = [word_tokenize(essay.lower()) for essay in data['text']]
       tokenized_essays = [[word for word in essay if word.isalnum() and word not in stop_words] for

       # Create a co-occurrence graph
       G = nx.Graph()
       window_size = 2

       for essay in tokenized_essays:
           for i in range(len(essay)):
               for j in range(i + 1, min(i + window_size, len(essay))):
                   if essay[i] != essay[j]:
                       if G.has_edge(essay[i], essay[j]):
                           G[essay[i]][essay[j]]['weight'] += 1
                       else:
                           G.add_edge(essay[i], essay[j], weight=1)

 [*]:  # Apply Node2Vec
       node2vec = Node2Vec(G, dimensions=64, walk_length=30, num_walks=200, workers=4)

       # Generate embeddings
       model = node2vec.fit(window=10, min_count=1, batch_words=4)

       # Get embeddings for all nodes
       embeddings = [model.wv[node] for node in G.nodes()]

      Computing transition probabilities: 100%                    17221/17221 [05:03<00:00, 844.05it/s]
      Generating walks (CPU: 4):  10%|█          | 5/50 [01:49<20:19, 27.11s/it]
```

Force Quit Applications

Your system has run out of application memory.

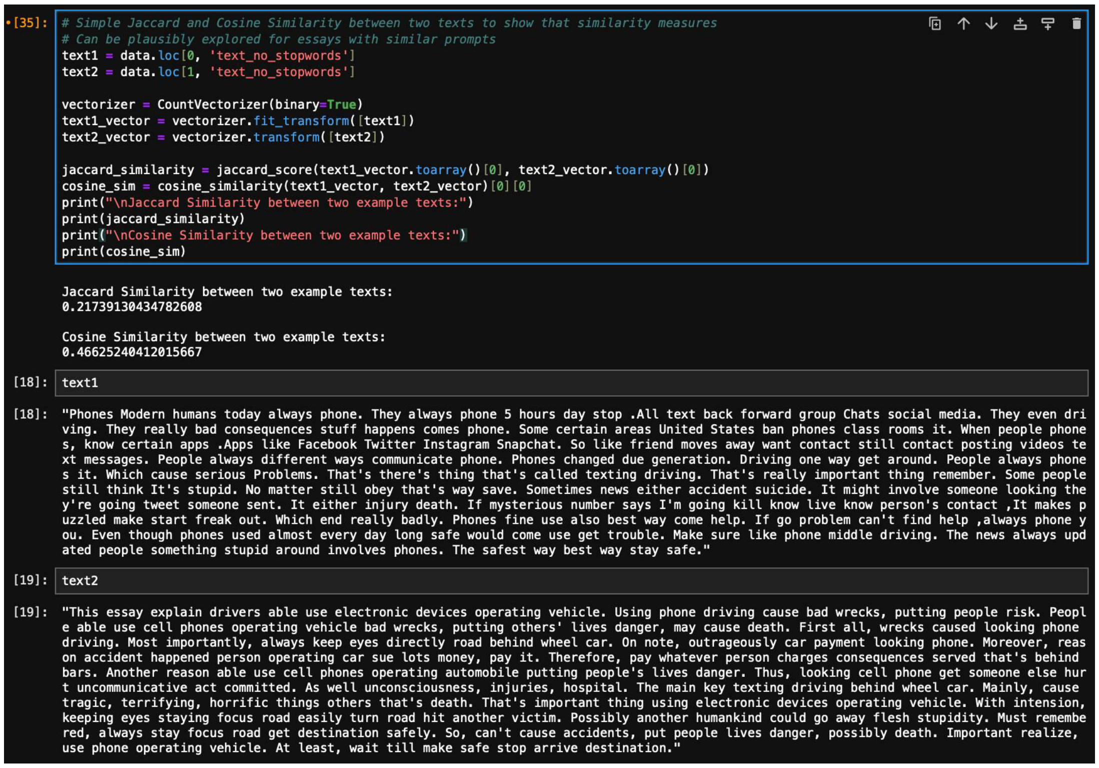
To avoid problems with your computer, quit any applications you are not using.

```
Astrill          283.4 MB
Google Chrome    152.0 MB
Notes             53.8 MB
Safari           342.1 MB
Terminal          85.0 MB
Finder           187.3 MB
```

Figure 12. My Apple Silicon was not powerful enough to support Node2Vec for the dataset and system ran out of application memory upon running the code in Figure 11.

Unfortunately, my local hardware did not support this process which seemed to be very computationally expensive. The Node2Vec conversion ran for a while on the reduced dataset, then my system ran out of application memory and the conversion never finished, as shown in Figure 11 and Figure 12. So PCA and TF-IDF Euclidean vectorization was used in this project, albeit having some theoretical imperfection.

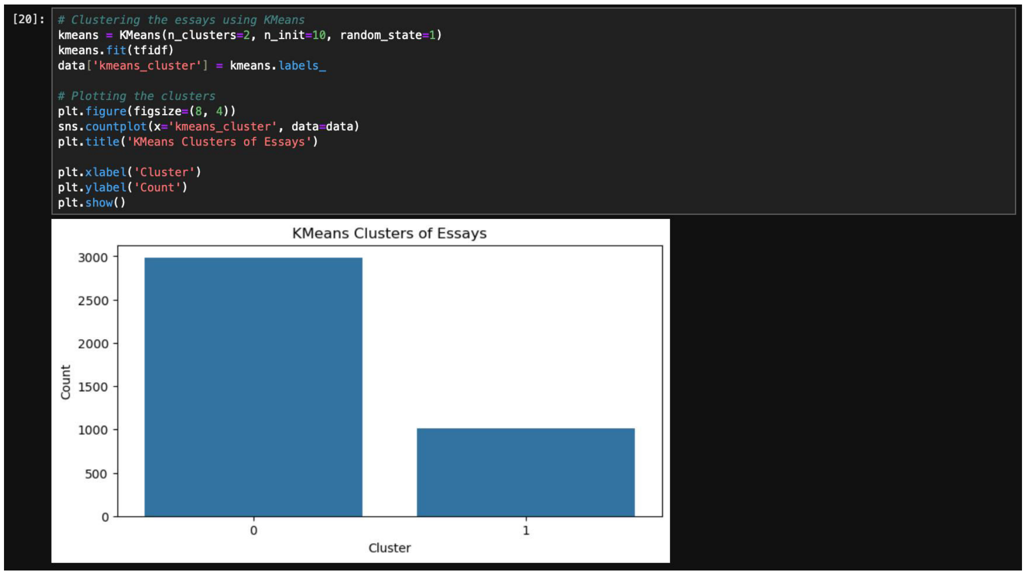
Preliminary Checks with Jaccard and Cosine Similarity



```
[35]: # Simple Jaccard and Cosine Similarity between two texts to show that similarity measures
      # Can be plausibly explored for essays with similar prompts
      text1 = data.loc[0, 'text_no_stopwords']
      text2 = data.loc[1, 'text_no_stopwords']

      vectorizer = CountVectorizer(binary=True)
      text1_vector = vectorizer.fit_transform([text1])
      text2_vector = vectorizer.transform([text2])

      jaccard_similarity = jaccard_score(text1_vector.toarray()[0], text2_vector.toarray()[0])
      cosine_sim = cosine_similarity(text1_vector, text2_vector)[0][0]
      print("\nJaccard Similarity between two example texts:")
      print(jaccard_similarity)
      print("\nCosine Similarity between two example texts:")
      print(cosine_sim)

      Jaccard Similarity between two example texts:
      0.21739130434782608

      Cosine Similarity between two example texts:
      0.46625240412015667

[18]: text1

[18]: "Phones Modern humans today always phone. They always phone 5 hours day stop .All text back forward group Chats social media. They even dri
      ving. They really bad consequences stuff happens comes phone. Some certain areas United States ban phones class rooms it. When people phone
      s, know certain apps .Apps like Facebook Twitter Instagram Snapchat. So like friend moves away want contact still contact posting videos te
      xt messages. People always different ways communicate phone. Phones changed due generation. Driving one way get around. People always phone
      s it. Which cause serious Problems. That's there's thing that's called texting driving. That's really important thing remember. Some people
      still think It's stupid. No matter still obey that's way save. Sometimes news either accident suicide. It might involve someone looking the
      y're going tweet someone sent. It either injury death. If mysterious number says I'm going kill know live know person's contact ,It makes p
      uzzled make start freak out. Which end really badly. Phones fine use also best way come help. If go problem can't find help ,always phone y
      ou. Even though phones used almost every day long safe would come use get trouble. Make sure like phone middle driving. The news always upd
      ated people something stupid around involves phones. The safest way best way stay safe."

[19]: text2

[19]: "This essay explain drivers able use electronic devices operating vehicle. Using phone driving cause bad wrecks, putting people risk. Peopl
      e able use cell phones operating vehicle bad wrecks, putting others' lives danger, may cause death. First all, wrecks caused looking phone
      driving. Most importantly, always keep eyes directly road behind wheel car. On note, outrageously car payment looking phone. Moreover, reas
      on accident happened person operating car sue lots money, pay it. Therefore, pay whatever person charges consequences served that's behind
      bars. Another reason able use cell phones operating automobile putting people's lives danger. Thus, looking cell phone get someone else hur
      t uncommunicative act committed. As well unconsciousness, injuries, hospital. The main key texting driving behind wheel car. Mainly, cause
      tragic, terrifying, horrific things others that's death. That's important thing using electronic devices operating vehicle. With intension,
      keeping eyes staying focus road easily turn road hit another victim. Possibly another humankind could go away flesh stupidity. Must remembe
      red, always stay focus road get destination safely. So, can't cause accidents, put people lives danger, possibly death. Important realize,
      use phone operating vehicle. At least, wait till make safe stop arrive destination."
```

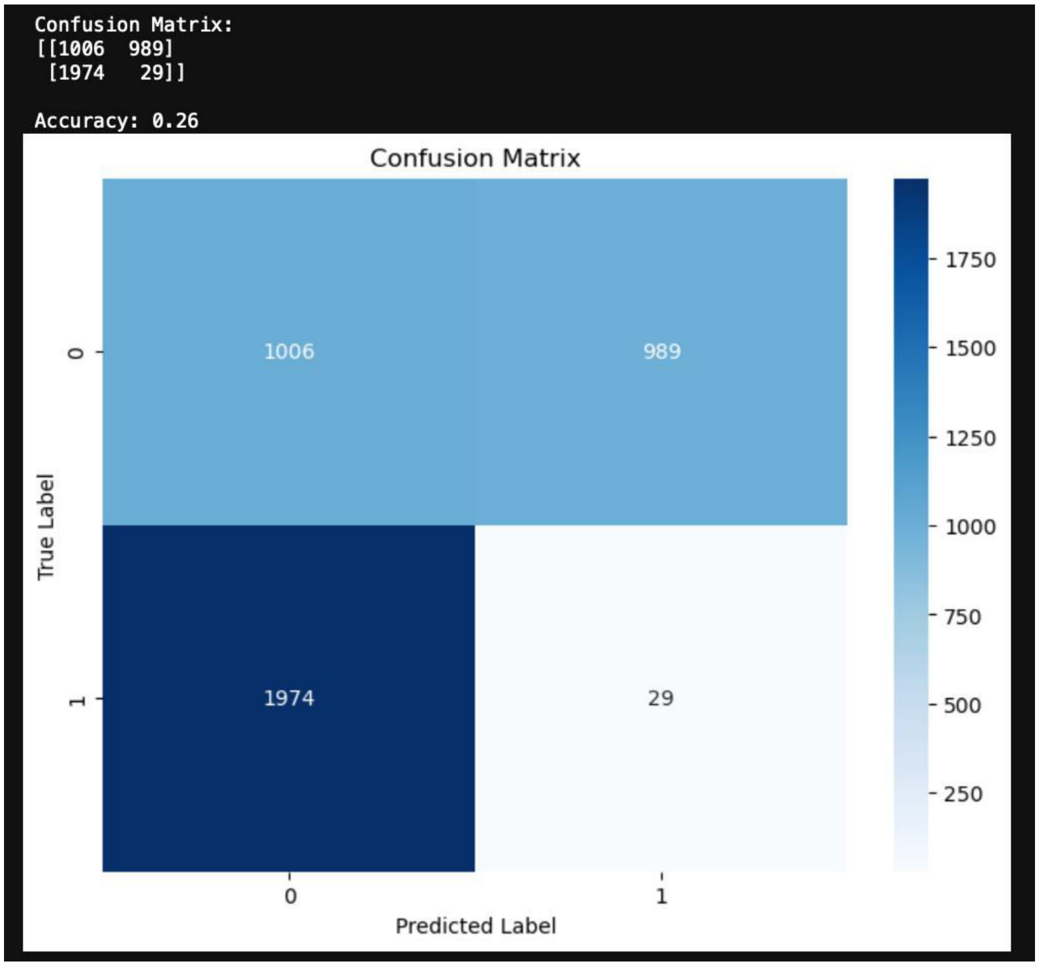
Figure 13. Jaccard and Cosine Similarity scores are quite high, 0.217 and 0.466 respectively for two sample essays that are on the prompt topic “Phone and Driving”

Naive K-means clustering with N=2



```
[20]: # Clustering the essays using KMeans
      kmeans = KMeans(n_clusters=2, n_init=10, random_state=1)
      kmeans.fit(tfidf)
      data['kmeans_cluster'] = kmeans.labels_

      # Plotting the clusters
      plt.figure(figsize=(8, 4))
      sns.countplot(x='kmeans_cluster', data=data)
      plt.title('KMeans Clusters of Essays')

      plt.xlabel('Cluster')
      plt.ylabel('Count')
      plt.show()
```

Figure 14. Naive K-means clustering with clusters = 2 show a high number of AI-written essays being classified as human written. (1 = AI written)



```
Confusion Matrix:
[[1006  989]
 [1974   29]]

Accuracy: 0.26
```

Figure 15. Naive K-means clustering with clusters = 2 showns dismal accuracy at 0.26, with a large number of false negatives (positive = 1 or AI written)

In Figure 13, Jaccard and Cosine Similarity scores were tested on sample essays with the same prompt. A Cosine similarity of 0.466 was observed and was higher than the Jaccard similarity score, perhaps since it ignores word frequency and words well with sparse matrices. However, our hypothesis is that these similiarity measures would not be precise enough for our classification, thus we move on and use them as simple checks only.

As our goal is to specify whether an essay is AI or human written, the naive starting point is a K-means clustering algorithm with clusters = 2. The results are dismal with an accuracy of 0.26 as shown in Figures 14 and 15, with a large number of false negatives, or AI written-essays were incorrectly classified as human written.

K-means with PCA

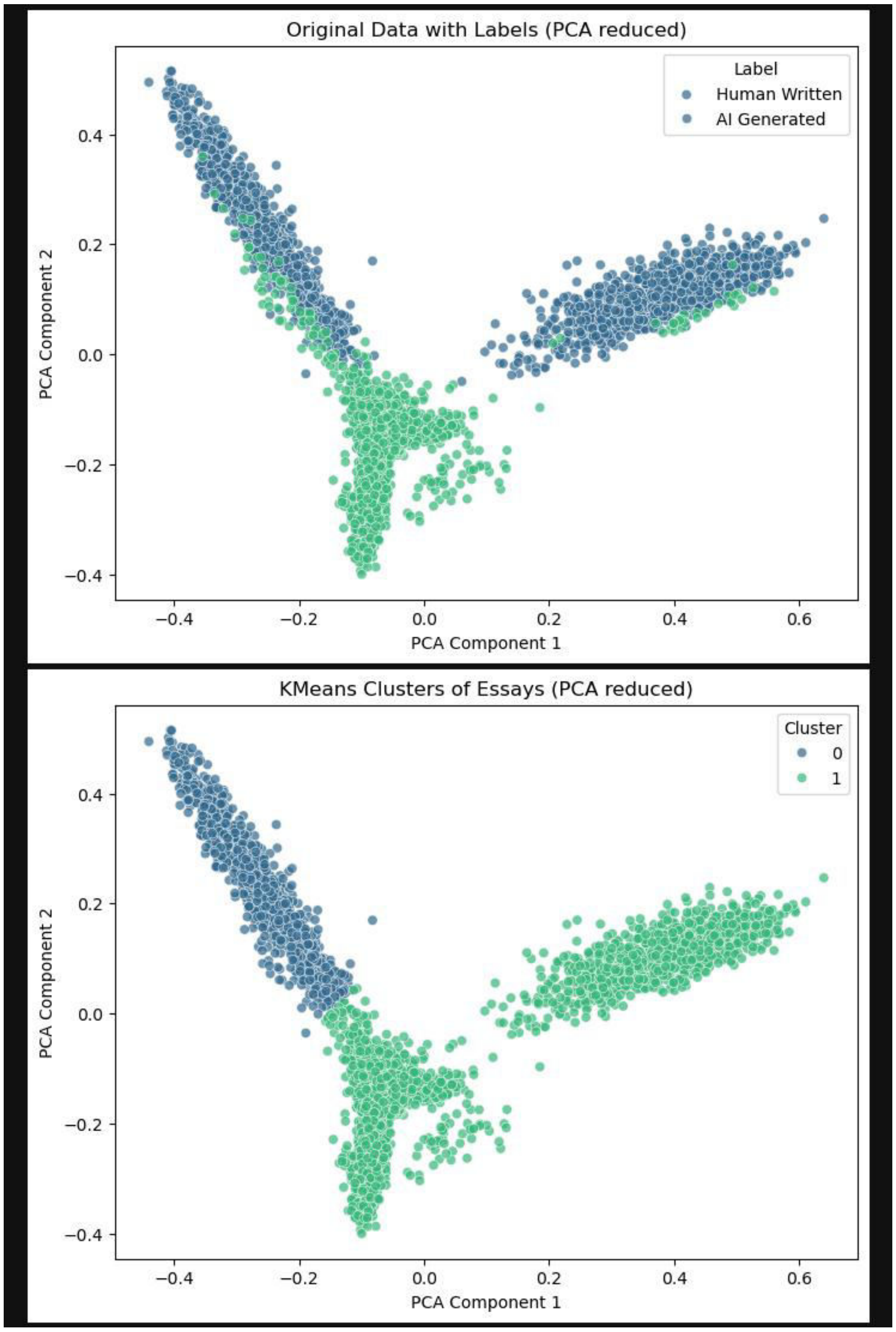
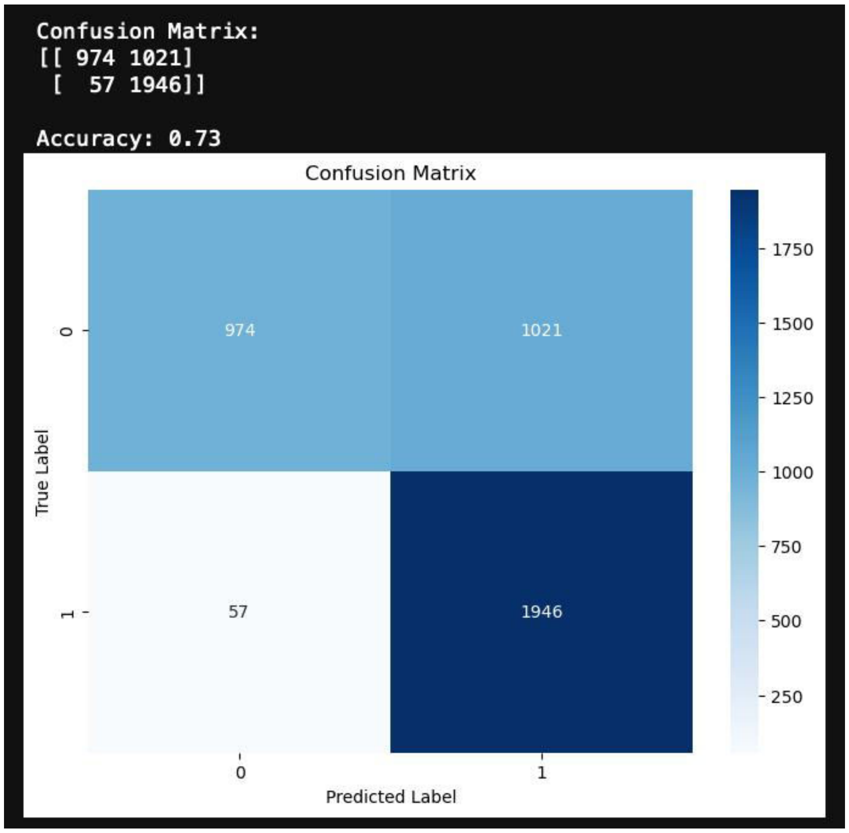


Figure 16. Top is original labels and bottom is the K-Means clustering with k=2 post simple PCA transform with 2 components. We can clearly see mis-classification of the "wing" on the right side as well as slight blending on the left "wing" (1=AI written)



```
Confusion Matrix:
[[ 974 1021]
 [  57 1946]]

Accuracy: 0.73
```

Figure 17. Confusion matrix for K-Means clustering with k=2 post simple PCA transform with 2 components. A large number of false positives are present and the false negatives have now been correctly classified compared to the naive K-means.

Shown in Figures 16 and 17 is K-means applied with a simple 2 component PCA. Higher PCA levels were tested but as contributions drop off quickly past the first 2 components and the results were not significantly different and thus were not shown.

Figures 16 and 17 should be viewed together. The 1,021 false negatives seen in the confusion matrix can be clearly seen as the right “wing” of the PCA plot. A result that is quite clear, which I was not expecting given the known difficulty of this classification problem.

My hypothesis is that the lack of variety in human written essay prompts (only 2) compared to AI written essays prompts (15) in the reduced dataset, which was shown prior in Figure 8, may be responsible for these false negatives, as almost all (1946 out of 2003 or 97%+) AI written essays were classified correctly.

Optimal Hyperparameter Selection in Clustering

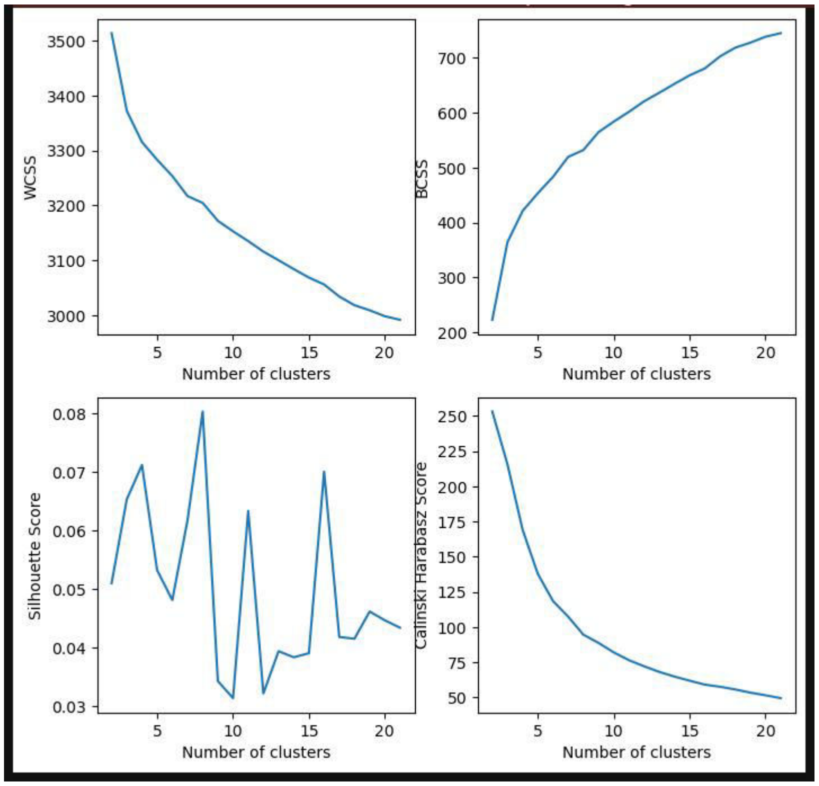


Figure 18. Optimal cluster number selection metric plots for K-means clustering

WCSS, BCSS, Silhoutte Scores and Calinski-Harabasz Scores were calculated and plotted to find the optimal cluster number. The implicit assumption here is we did not define the problem as a binary classification problem initially given the data.

There are no kinks in the Calinski-Harabasz score and the max is at a very low number of clusters, suggesting that 2 clusters is indeed optimal, and WCSS and BCSS plots support this as well. However, we see in the Silhoutte Scores that one peak does occur at k= 2 or 3, but also at k=8, 11, 16 clusters roughly. The Silhouette Score is quite low across all cluster numbers, with values ranging up to 0.08. These low values indicate poor clustering quality, where the clusters are either not well-separated or the points within each cluster are not particularly similar to each other.

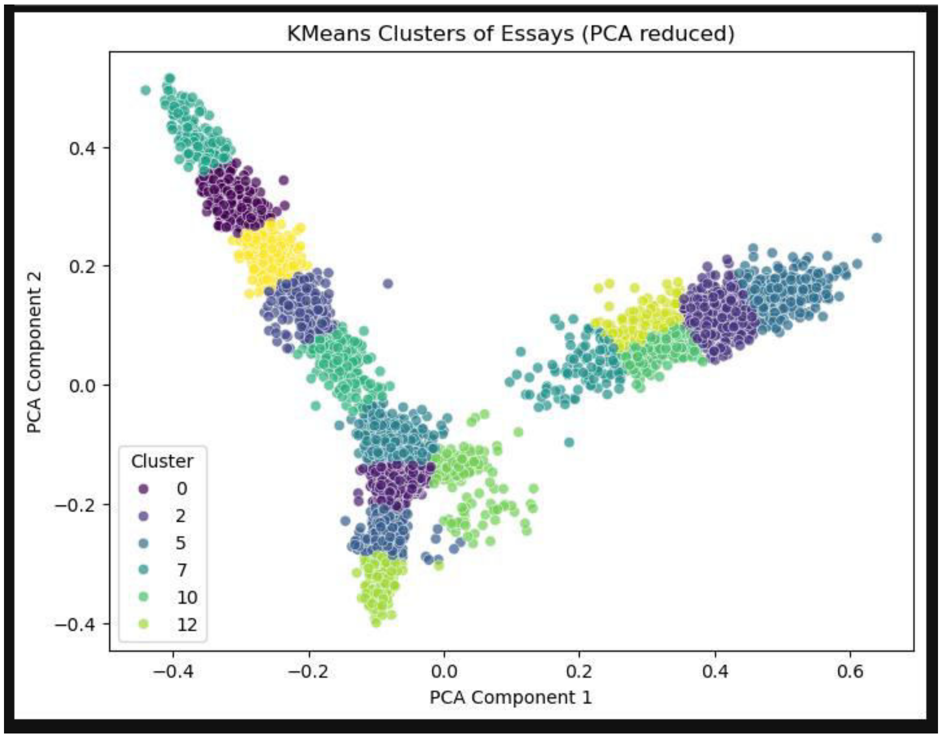
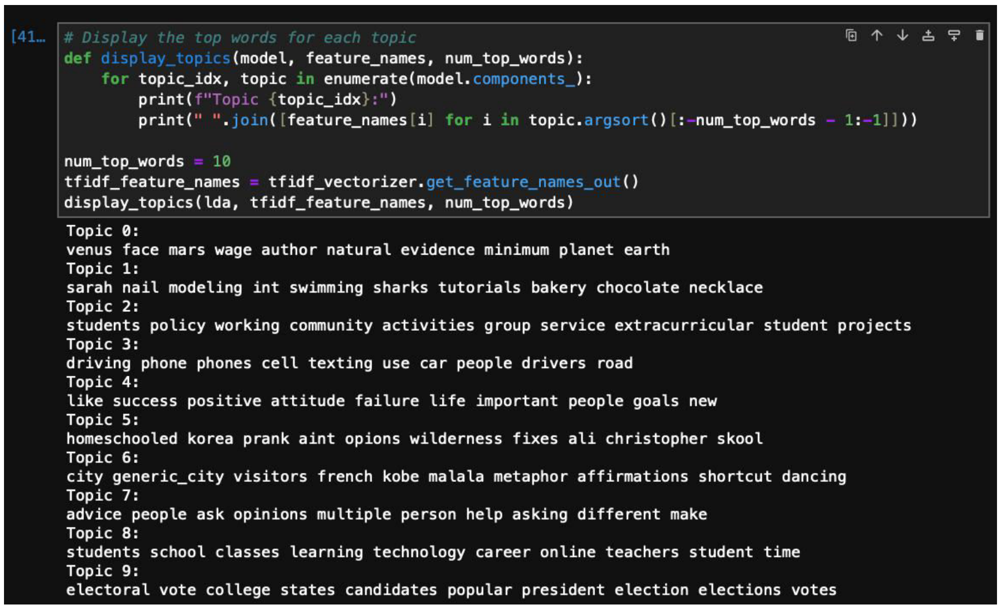


Figure 19. K-Means clustering with k=15 post simple PCA transform with 2 components. Note that the right “wing” and the centre of the left “wing” are different clusters/colors despite the colors looking very similar.

In Figure 19, k=15 is shown. We see that the clusters are still very clearly defined and do not blend. One explanation for this clear distinction is that there are around 15 different prompt topics used in the dataset, previously shown in Figure 7, which should be the most likely explanation given the vocabulary and words used in different prompt topics cluster naturally. I show this briefly in Figure 20, where I implemented a simple LDA using LatentDirichletAllocation which listed top 10 words used for top 10 topics in the vectorized essays, and clear themes can be seen from each topic’s top 10 words.

However, the left and right “wings” are still separated distance wise and this is not not explained in the metrics above.



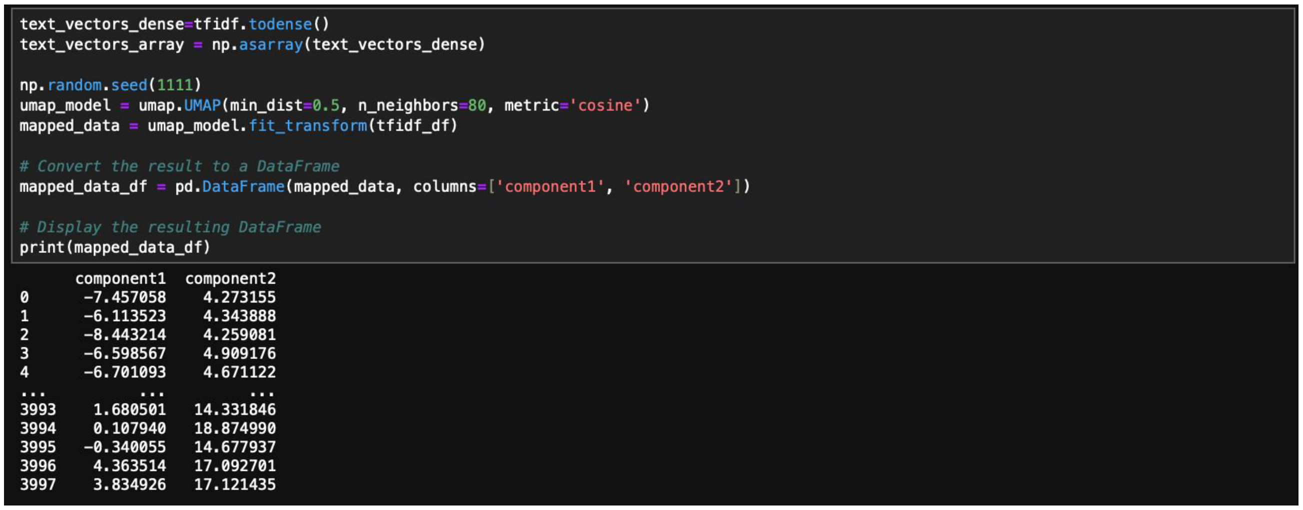
```
[41…
# Display the top words for each topic
def display_topics(model, feature_names, num_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print(f"Topic {topic_idx}:")
        print(" ".join([feature_names[i] for i in topic.argsort()[:-num_top_words - 1:-1]]))

num_top_words = 10
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
display_topics(lda, tfidf_feature_names, num_top_words)
```

```
Topic 0:
venus face mars wage author natural evidence minimum planet earth
Topic 1:
sarah nail modeling int swimming sharks tutorials bakery chocolate necklace
Topic 2:
students policy working community activities group service extracurricular student projects
Topic 3:
driving phone phones cell texting use car people drivers road
Topic 4:
like success positive attitude failure life important people goals new
Topic 5:
homeschooled korea prank aint opions wilderness fixes ali christopher skool
Topic 6:
city generic_city visitors french kobe malala metaphor affirmations shortcut dancing
Topic 7:
advice people ask opinions multiple person help asking different make
Topic 8:
students school classes learning technology career online teachers student time
Topic 9:
electoral vote college states candidates popular president election elections votes
```

Figure 20. LDA listing top 10 words for top 10 topics in the essays. Clear themes can be seen from each topic's top 10 words.

Note that I have also tried Kernel PCA with 2 components and a cosine similarity kernel for text similarity. However, the results were almost exactly the same as with regular (linear) PCA and thus results are not re-shown here. I suspect this may be due to the essay data represented as TF-IDF vectors is high-dimensional and sparse. Thus, the linear structure captured by PCA might already be a good approximation of the variance in the data, leading to similar results when using Kernal PCA.

UMAP Projections



```
text_vectors_dense=tfidf.todense()
text_vectors_array = np.asarray(text_vectors_dense)

np.random.seed(1111)
umap_model = umap.UMAP(min_dist=0.5, n_neighbors=80, metric='cosine')
mapped_data = umap_model.fit_transform(tfidf_df)

# Convert the result to a DataFrame
mapped_data_df = pd.DataFrame(mapped_data, columns=['component1', 'component2'])

# Display the resulting DataFrame
print(mapped_data_df)
```

```
      component1  component2
0      -7.457058    4.273155
1      -6.113523    4.343888
2      -8.443214    4.259081
3      -6.598567    4.909176
4      -6.701093    4.671122
...          ...         ...
3993    1.680501   14.331846
3994    0.107940   18.874990
3995   -0.340055   14.677937
3996    4.363514   17.092701
3997    3.834926   17.121435
```

Figure 21. UMAP projections with original labels shown.

The UMAP parameter setup is shown in Figure 21. I chose to map the text vectors using UMAP with a min_dist = 0.5 to avoid artifical separation, while setting n_neighbors = 80 to have compact clusters shown if possible. Again cosine metric selected for its ease of use in assessing text similarity (though not without drawbacks, as frequency of words are not considered, which may be a factor in classifying human vs AI written essays).

In the plotted UMAP results in Figure 22, we can again see two rough areas, one in bottom left and the other in top right, with AI generated labels being less compact and more distributed (expected given the higher variety of prompt topics for AI in the dataset used, again shown in Figure 8).

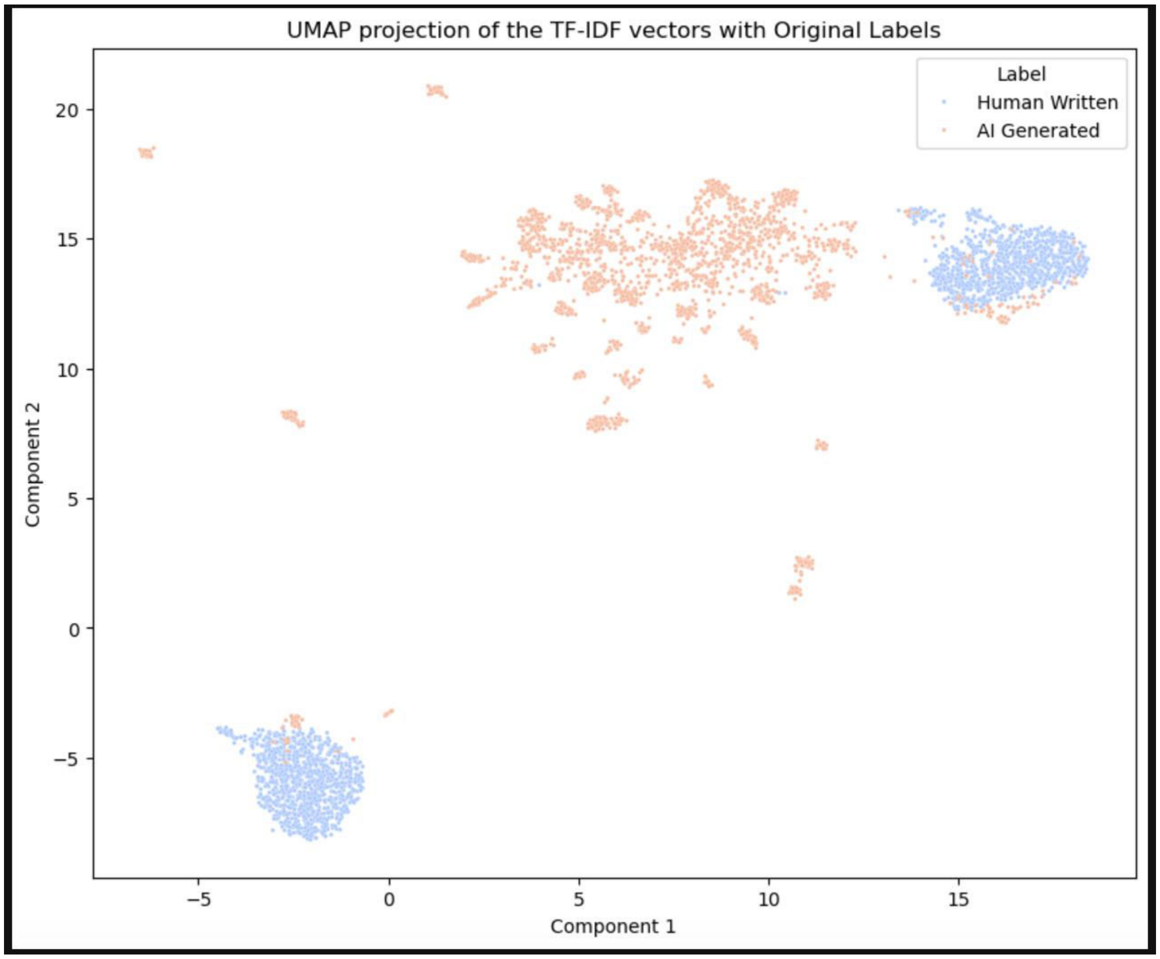
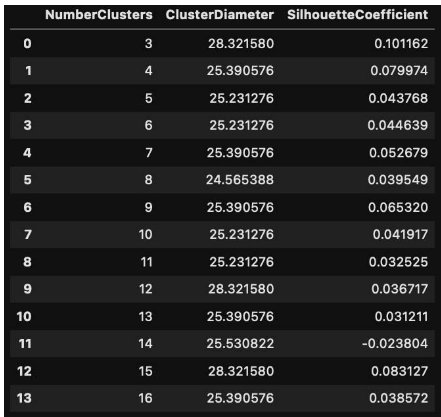


Figure 22. UMAP projections with original labels shown.

The false positives seen prior in the K-means with PCA can be clearly seen in the top-right corner, which are Human Written essays, but are very close to the AI Generated cluster and it was likely this cluster that was misclassified as AI Generated in the confusion matrix previously shown in Figure 17.

CLARA (K-mediods approximation)



	NumberClusters	ClusterDiameter	SilhouetteCoefficient
0	3	28.321580	0.101162
1	4	25.390576	0.079974
2	5	25.231276	0.043768
3	6	25.231276	0.044639
4	7	25.390576	0.052679
5	8	24.565388	0.039549
6	9	25.390576	0.065320
7	10	25.231276	0.041917
8	11	25.231276	0.032525
9	12	28.321580	0.036717
10	13	25.390576	0.031211
11	14	25.530822	-0.023804
12	15	28.321580	0.083127
13	16	25.390576	0.038572

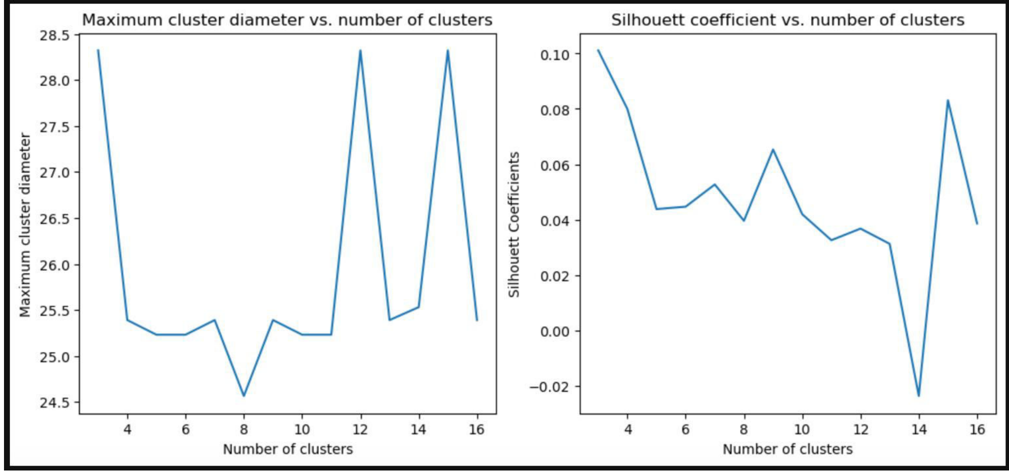


Figure 23. CLARA assessment metrics for optimal number of clusters

- **Maximum Cluster Diameter vs. Number of Clusters:**

  The maximum cluster diameter fluctuates more dramatically as the number of clusters increases. There is no clear trend of decreasing diameter, which might indicate some instability or irregularity in the clustering process. Max diameter generally hovers around a high value (around 25.2 to 28.3), which suggests that the clusters are not particularly tight or well-separated.

- **Silhouette Coefficient vs. Number of Clusters:**

  The silhouette coefficient is quite low across all cluster numbers, with values ranging from around -0.02 to 0.1. These low values indicate poor clustering quality, where the clusters are either not well-separated or the points within each cluster are not particularly similar to each other. The highest silhouette coefficient is at 3 clusters (around 0.1), but this is still quite low, indicating that even at this number of clusters, the quality of clustering is not strong.

Given the unpromising metrics seen in Figure 23, CLARA was applied on top of the UMAP projections seen in Figure 22 in an attempt to improve results, as UMAP reduces the dimensionality of the data and thus potentially improves clustering.

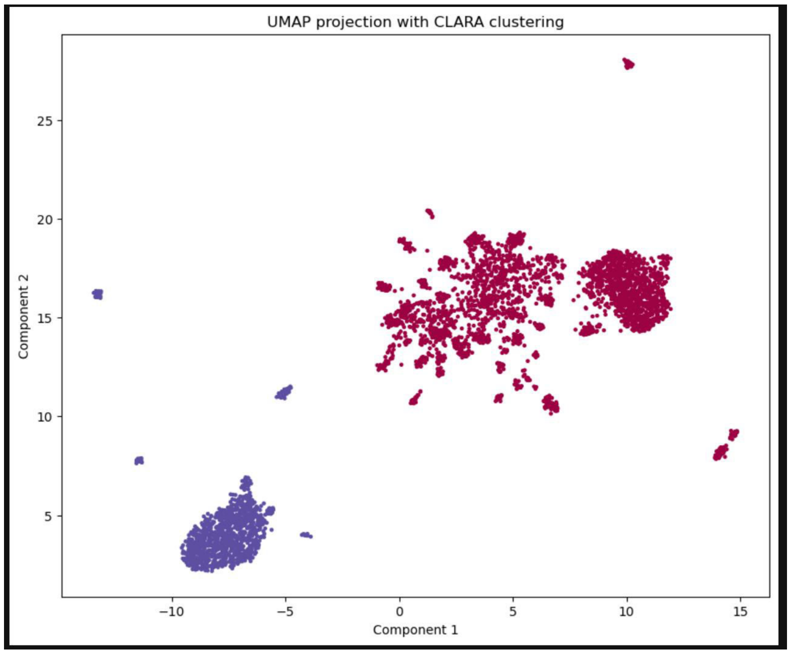


Figure 24. CLARA approximation plot metrics for optimal number of clusters, clusters=2

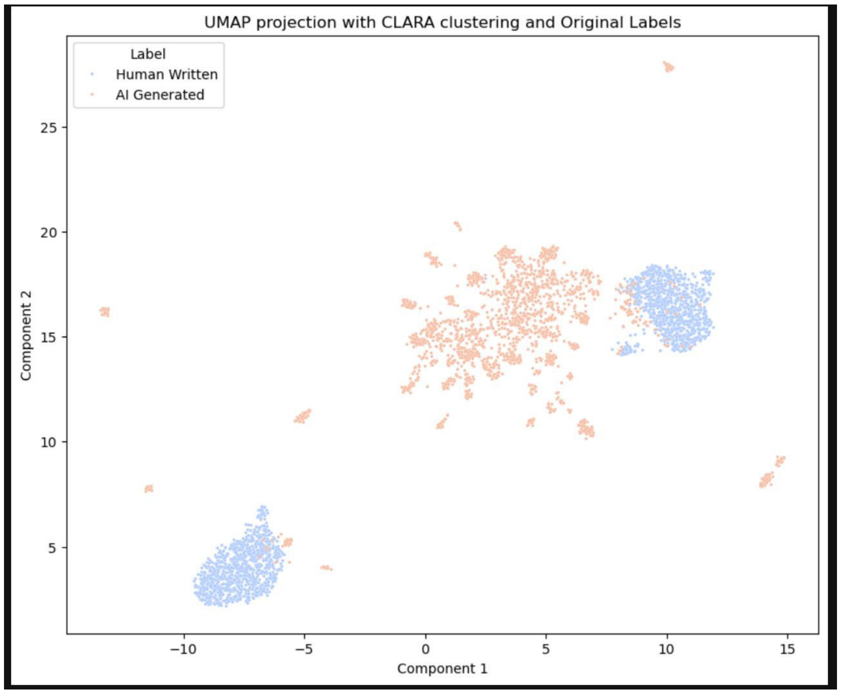


Figure 25. CLARA approximation plot with actual Labels shown

Again, comparing Figures 24 and 25, we see in Figure 25 the false positives seen prior in the UMAP (Figure 22) and K-means with PCA (Figure 17). This seems to be a trend I that I cannot get rid of and improve upon regardless of the type of dimensionality reduction and clustering method used. Figure 26 shows the CLARA with UMAP with clusters=3, which looks similar to Figure 25 with the false positives being its own cluster and mixed in with the true AI Generated essays.

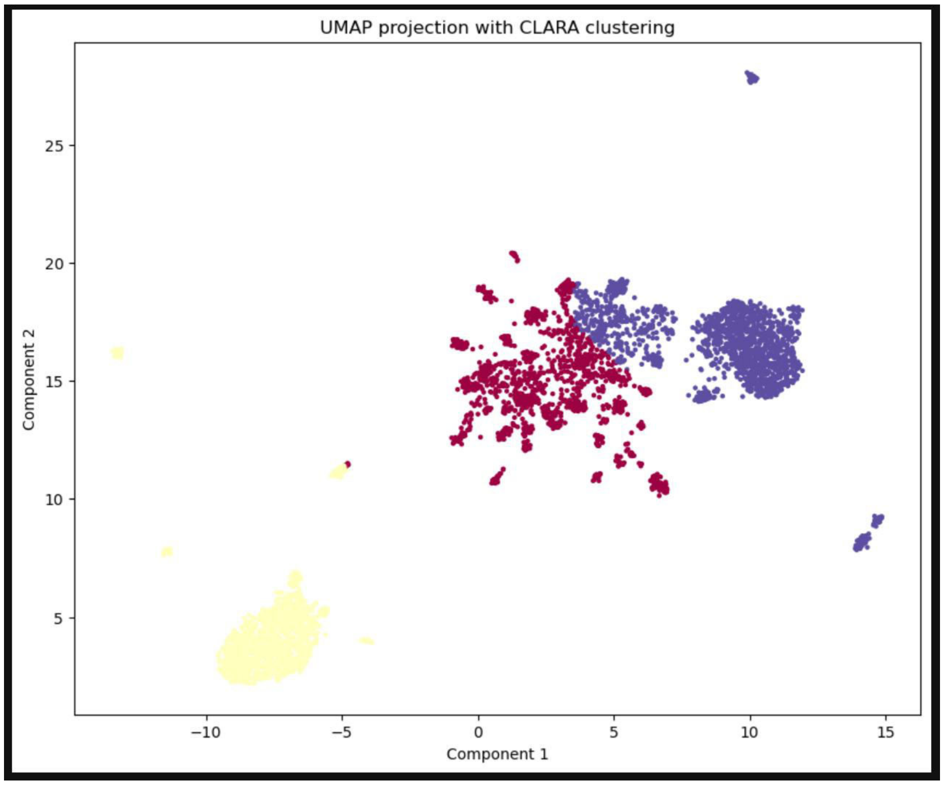
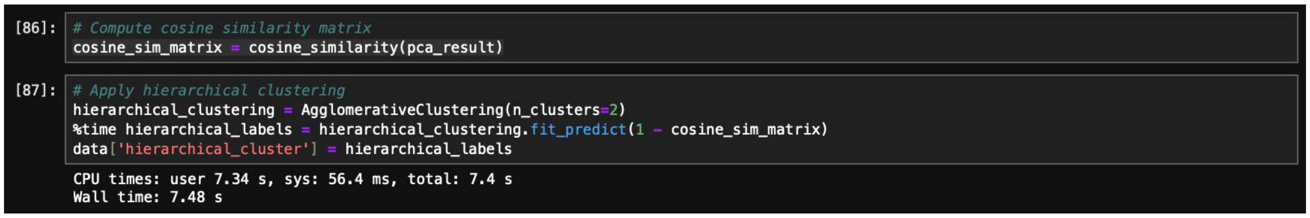


Figure 26. CLARA approximation plot metrics for optimal number of clusters, clusters=3

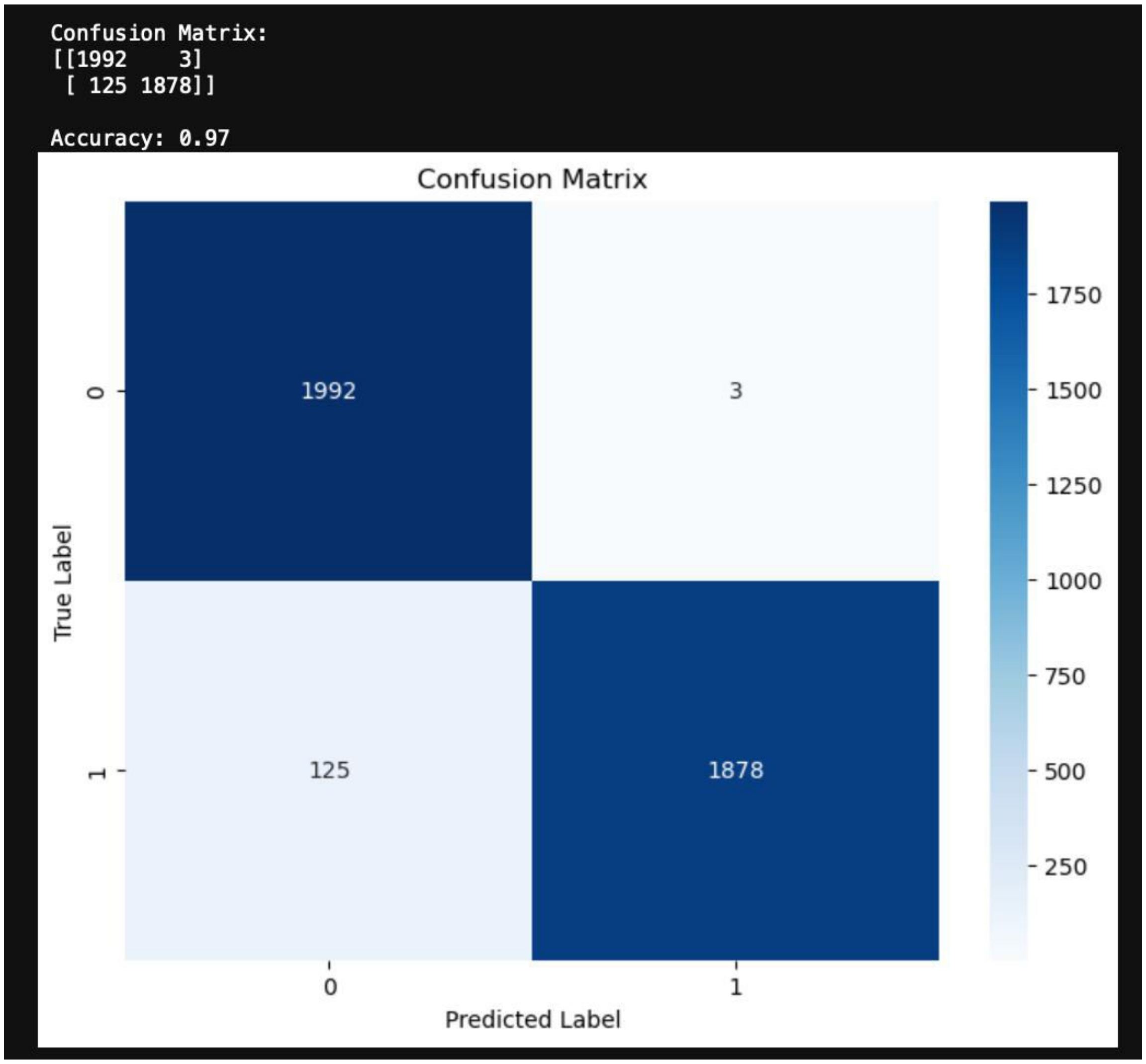
Spectral Clustering

I avoided this completely given that it is heavily compute intensive with theoretically ~O(n^3) complexity, though in theory it would work very well with high dimensional complex cluster shapes.

Hierachical (Agglomerative) Clustering



```
[86]: # Compute cosine similarity matrix
      cosine_sim_matrix = cosine_similarity(pca_result)

[87]: # Apply hierarchical clustering
      hierarchical_clustering = AgglomerativeClustering(n_clusters=2)
      %time hierarchical_labels = hierarchical_clustering.fit_predict(1 - cosine_sim_matrix)
      data['hierarchical_cluster'] = hierarchical_labels

      CPU times: user 7.34 s, sys: 56.4 ms, total: 7.4 s
      Wall time: 7.48 s
```



```
Confusion Matrix:
[[1992    3]
 [ 125 1878]]

Accuracy: 0.97
```

Figure 27. Agglomerative clustering (k=2) with cosine similarity applied to linear PCA (components =2) results was done. Confusion matrix is shown with a 97% accuracy.

The Confusion matrix now shows 97% accuracy in Figure 27. I had finally gotten rid of the false positives! However, the key was not the agglomerative clustering. Instead, the key seemed to be pre-calculating a cosine similarity matrix using the PCA reduced results, as simple K-means with cosine similarity applied to PCA results also yielded an accuracy of 97%, with the full plot shown in Figure 28.

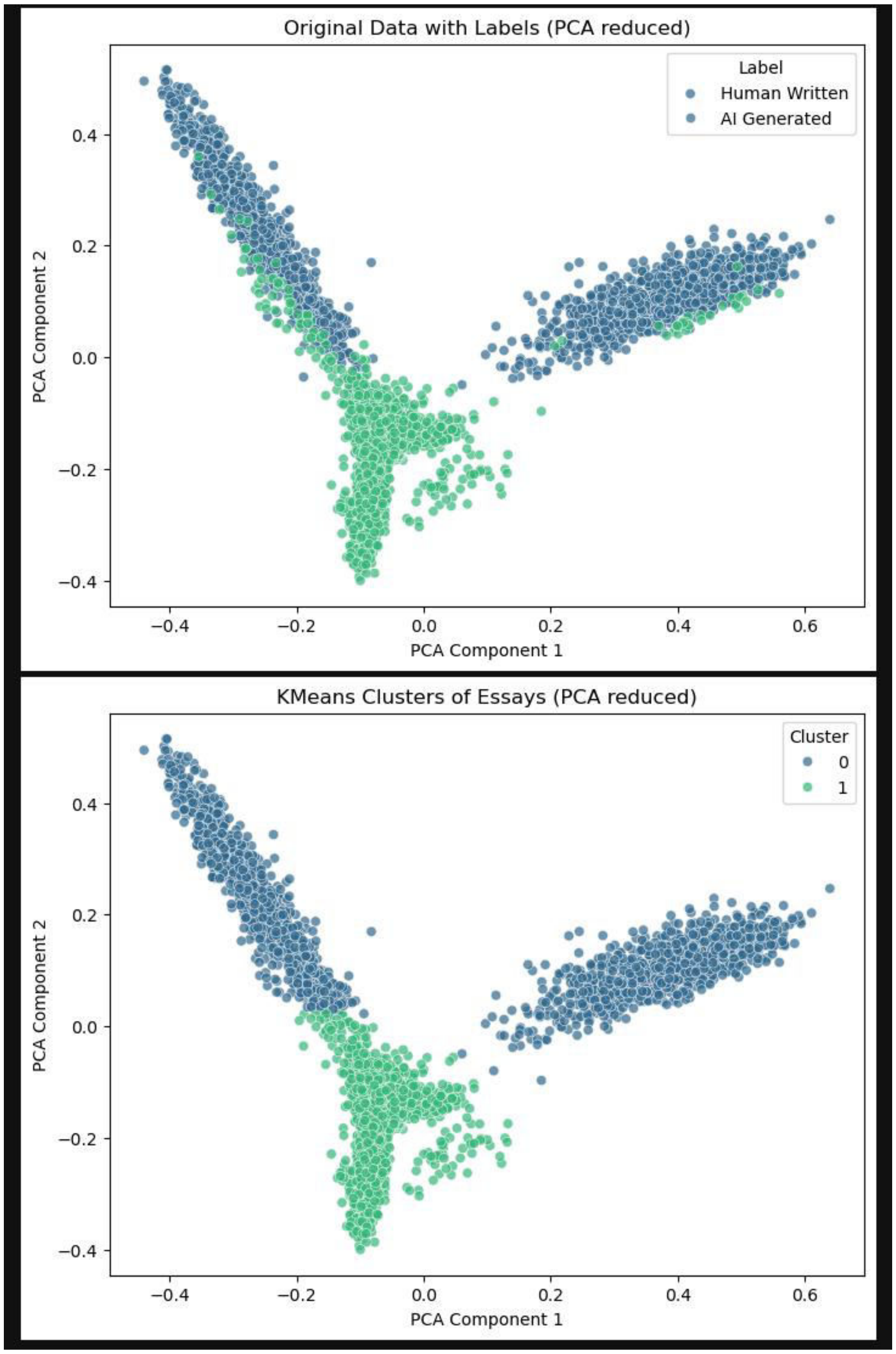
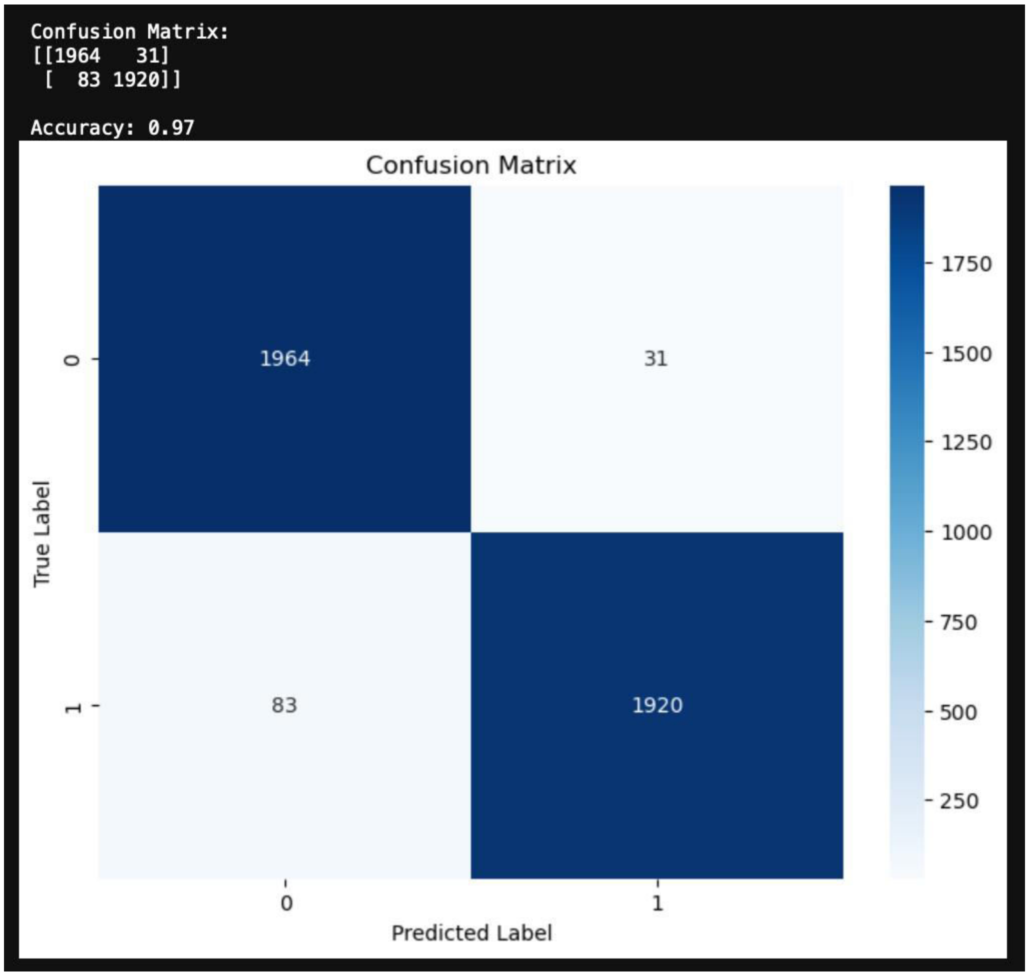


Figure 28. Final Results: K-means with cosine similarity applied to PCA results. The classification accuracy has also reached 97%.



```
Confusion Matrix:
[[1964   31]
 [  83 1920]]

Accuracy: 0.97
```

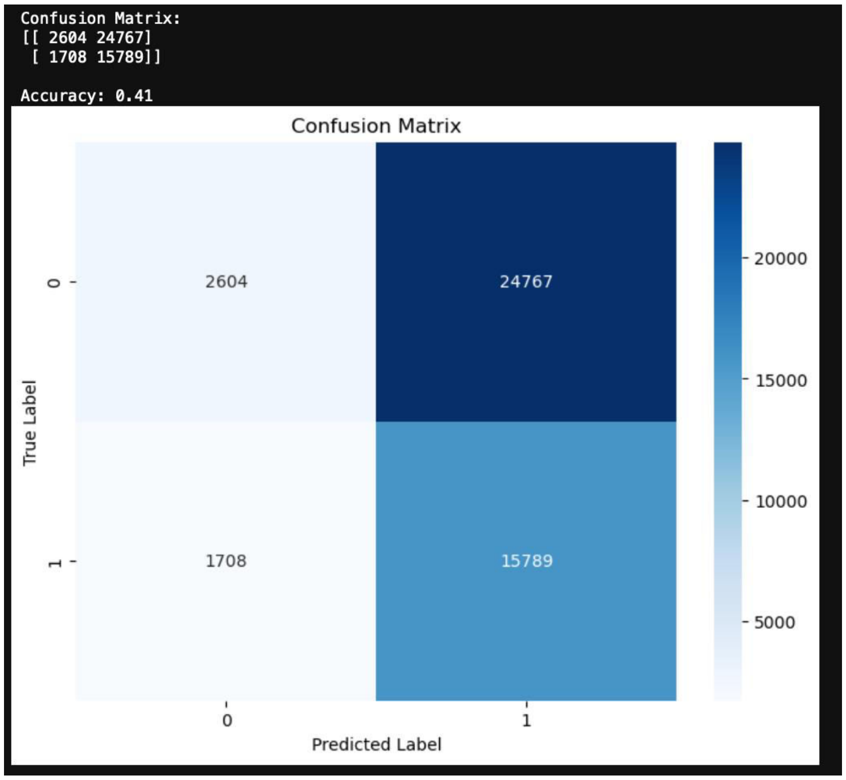
Figure 28. Final Results: K-means with cosine similarity applied to PCA results. The classification accuracy has also reached 97%.

Note that previously Kernal PCA with a cosine kernal did not yield such a high accuracy. My hypothesis is that cosine similarity used directly on the PCA-transformed data clusters based on how similar the angles of the data vectors are, which is particularly effective when given good data structure alignment with cosine similarity metric.

Comparatively. In KPCA using a cosine kernel, the data is non-linearly mapped to a higher-dimensional space where only the inner products represent cosine similarity, since the TF-IDF vectors were already linear, there wasn't much clustering improvement to be seen.

The insight here is that I knew from the start of project that cosine similiarity was a good metric for use on text related tasks and vectors, I just needed to think deeper to apply it properly.

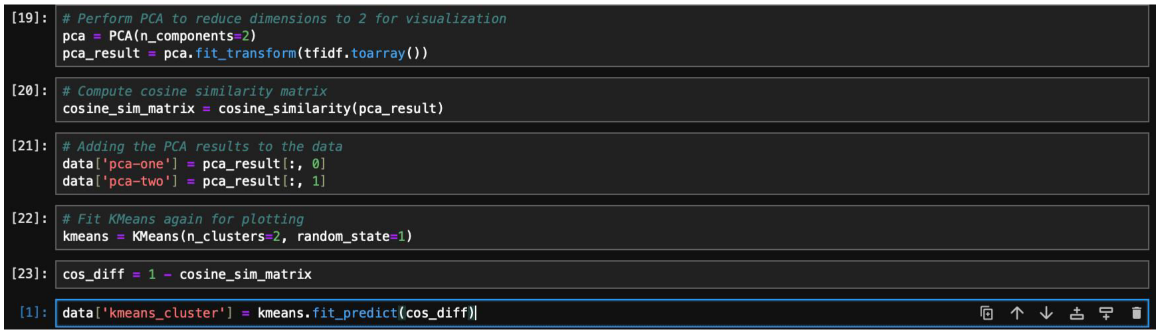
PCA with Clustering applied to Full Dataset (Without Cosine Similarity)



```
Confusion Matrix:
[[ 2604 24767]
 [ 1708 15789]]

Accuracy: 0.41
```

Figure 30. Confusion matrix of PCA, components = 2, with K-means, clusters = 2 Applied to full dataset.

Note that I really wanted to apply cosine similarity to the dataset. However, my Macbook Pro with an M3 Pro chip could not handle this. I even went as far as breaking up the calculations in cells. It was the Kmeans fit_predict function that failed, initially with system application memory runout, then after some tweaking on my end memory-wise, the Kernel just dies at around 10 minutes of runtime each time. So I could not obtain the accuracy using a cosine similarity matrix on the full dataset.



```
[19]: # Perform PCA to reduce dimensions to 2 for visualization
      pca = PCA(n_components=2)
      pca_result = pca.fit_transform(tfidf.toarray())

[20]: # Compute cosine similarity matrix
      cosine_sim_matrix = cosine_similarity(pca_result)

[21]: # Adding the PCA results to the data
      data['pca-one'] = pca_result[:, 0]
      data['pca-two'] = pca_result[:, 1]

[22]: # Fit KMeans again for plotting
      kmeans = KMeans(n_clusters=2, random_state=1)

[23]: cos_diff = 1 - cosine_sim_matrix

 [1]: data['kmeans_cluster'] = kmeans.fit_predict(cos_diff)
```

Figure 31. Breaking up the code into as small cells as possible in an attempt to run PCA with K-means clustering applied with a cosine similarity matrix on the full dataset.

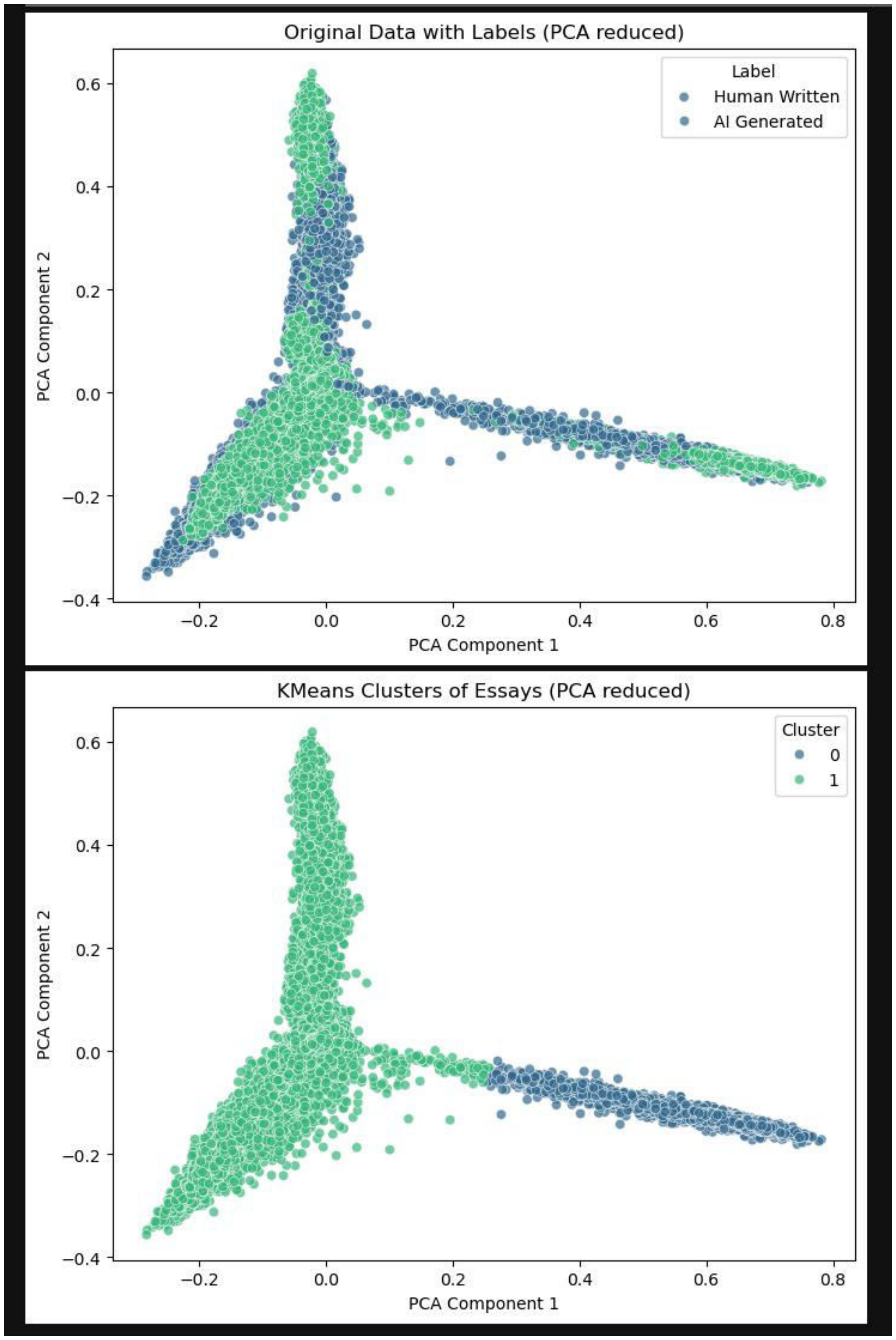


Figure 32. Top is original labels and bottom is the K-Means clustering with k=2 post simple PCA transform with 2 components on the full dataset, with no cosine similarity applied.

The naive K-means before and after PCA did not change the accuracy much, unlike in the reduced dataset, both yielding only 41% accuracy. However, we should not be discouraged too early. Upon further inspection, the classifier almost classifies all AI generated essays correctly again. It is the false positives that have greatly increated in number, as the full dataset has disproportionally more human essays vs AI generated essays as shown in Figure 1. We find that the classifier is unable to classify human written essays properly and defaults to essays being AI-written without a cosine similarity matrix.

V. Conclusion

The best technique, a combination of either K-means or agglomerative clustering (k=2) with cosine similarity applied to linear PCA (components =2) results yielded a binary classification accuracy of 97%. It is uncertain whether this can scale with larger datasets and should be further explored given appropriate compute resources.

This best technique contrasts with initial naive K-means clustering with k =2 showed a classifcation accuracy of only 26%, with the confusion matrix showing a large number of false negatives (positive = 1, or AI written), incorrectly classifying AI written-essays as human written.

K - means clustering with 2 PCA components improved accuracy to 73%, with almost 97% of all AI-written essays being classified correctly and nearly all incorrect classifications being false positives, incorrectly classifying human written essays as AI written. We hypothesize the high skew to false positives may be due to skew in the reduced dataset, whether out of the 15 different prompt topics only 2 had human written essays, while AI written essays spanned all 15 topics.

Optimal hyperparameter selection was performed with low silhouette scores < 0.1, which indicated poor clustering quality. CLARA with UMAP was also applied and did not achieve a higher accuracy, with a false positive cluster clearly seen as a trend which also appeared in K-means with PCA.

Kernal PCA and agglomerative clustering were also performed with no significant improvements to the binary classification. Theoretically sound but highly compute intensive methods such as spectral clustering were noted but not attempted.

The findings of this project should be tested and scaled on large datasets. Supervised methods such as deep learning with RNN or LSTM models may also be attempted to tackling the problem of AI vs Human written test classification given labelled data.

VI. Appendix - Code Used in Project

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import numpy.random as nr
import umap

from sklearn.metrics import jaccard_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import silhouette_score, calinski_harabasz_score
from sklearn.metrics.pairwise import pairwise_distances, cosine_similarity,
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.decomposition import PCA, KernelPCA
from sklearn_extra.cluster import CLARA, KMedoids, CommonNNClustering
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import confusion_matrix, accuracy_score, precision_scor

from nltk.corpus import stopwords
# nltk.download('stopwords')
```

In [3]:
```
# Load the dataset, note that since running the notebook on my machine takes
# I only kept slightly less than 10% of the original dataset at 3,998 essays
# Can be downloaded at https://www.kaggle.com/datasets/thedrcat/daigt-v2-tra
data = pd.read_csv('train_v2_drcat_02_reduced.csv')
```

In [4]:
```
# Display structure of the dataset
print("Dataset Info:")
print(data.info())

# Check for missing values
print("\nMissing Values:")
print(data.isnull().sum())
```

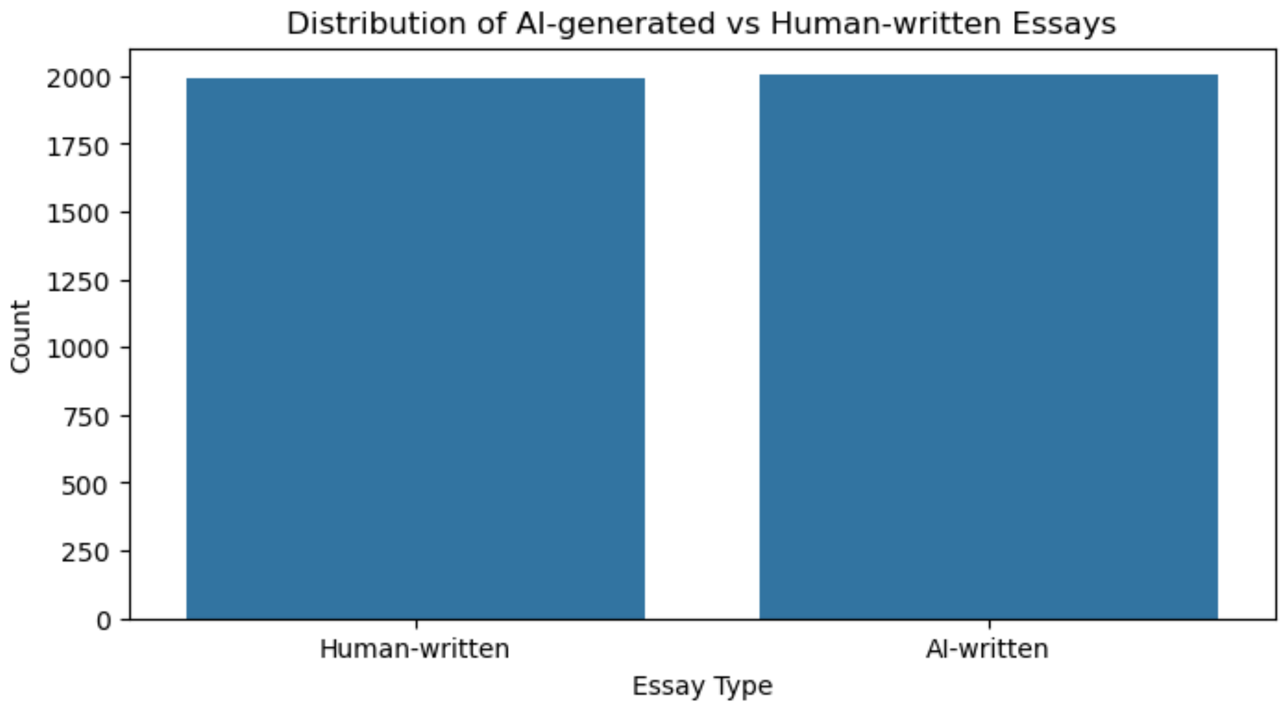
```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3998 entries, 0 to 3997
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   text           3998 non-null   object
 1   label          3998 non-null   int64 
 2   prompt_name    3998 non-null   object
 3   source         3998 non-null   object
 4   RDizzl3_seven  3998 non-null   bool  
dtypes: bool(1), int64(1), object(3)
memory usage: 129.0+ KB
None

Missing Values:
text             0
label            0
prompt_name      0
source           0
RDizzl3_seven    0
dtype: int64
```

In [5]:

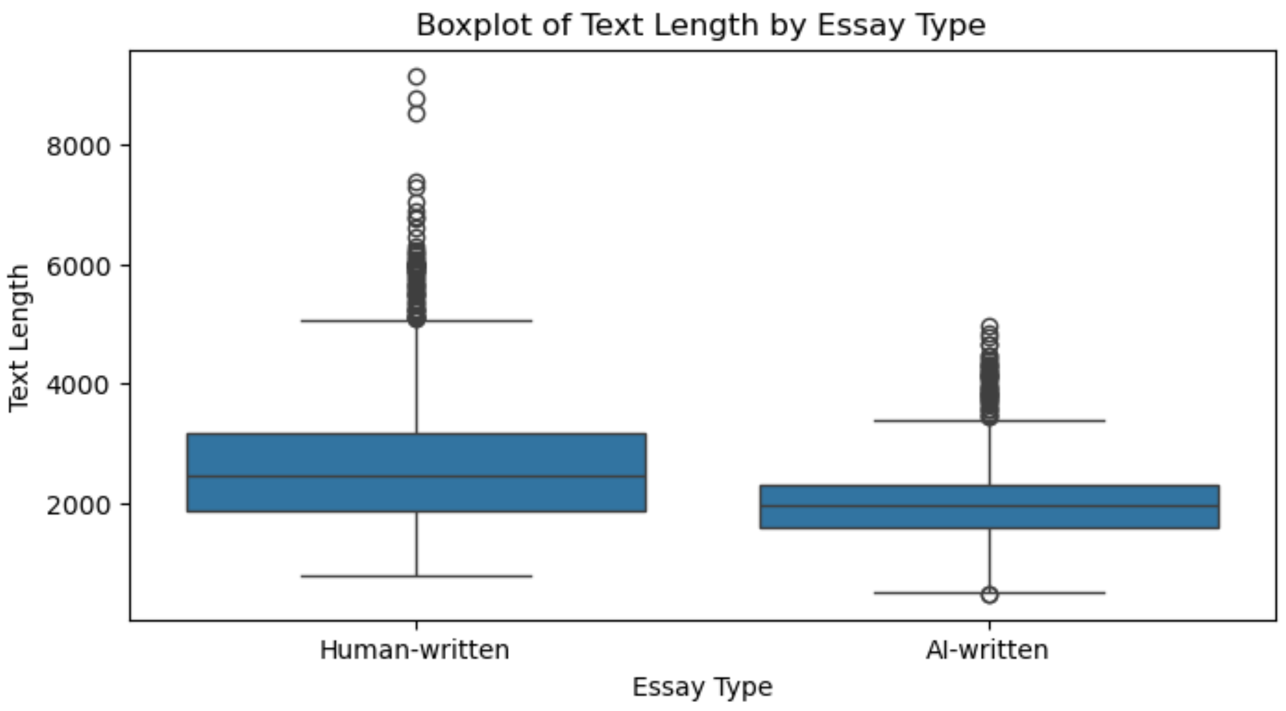
```
# Examine the distribution of the 'label' column
print("\nLabel Distribution:")
print(data['label'].value_counts())

# Separate the data into essays generated by AI (label=1) and written by hum
human_written = data[data['label'] == 0]
ai_generated = data[data['label'] == 1]

# Display the first few rows of human-written essays
print("\nHuman-written essays:")
print(human_written.head())

# Display the first few rows of AI-generated essays
print("\nAI-generated essays:")
print(ai_generated.head())
```

```
Label Distribution:
label
1    2003
0    1995
Name: count, dtype: int64

Human-written essays:
                                                text  label  \
0  Phones\n\nModern humans today are always on th...      0
1  This essay will explain if drivers should or s...      0
2  Driving while the use of cellular devices\n\nT...      0
3  Phones & Driving\n\nDrivers should not be able...      0
4  Cell Phone Operation While Driving\n\nThe abil...      0

          prompt_name           source  RDizzl3_seven
0  Phones and driving  persuade_corpus          False
1  Phones and driving  persuade_corpus          False
2  Phones and driving  persuade_corpus          False
3  Phones and driving  persuade_corpus          False
4  Phones and driving  persuade_corpus          False

AI-generated essays:
                                                   text  label  \
1995   In recent years, technology has had a profoun...      1
1996  I strongly believe that meditation and mindful...      1
1997  One way school administrators can attempt to c...      1
1998  While summer is meant as a break from the regu...      1
1999  The use of Facial Action Coding System (FACS) ...      1

                     prompt_name               source  RDizzl3_seven
1995            Car-free cities  mistral7binstruct_v2           True
1996          Distance learning           llama_70b_v1          False
1997        Cell phones at school          chat_gpt_moth          False
1998            Summer projects       darragh_claude_v7          False
1999  Facial action coding system     darragh_claude_v6           True
```

In [6]:
```python
# Plot the number of AI-generated vs human-written essays
plt.figure(figsize=(8, 4))
sns.countplot(x='label', data=data)
plt.title('Distribution of AI-generated vs Human-written Essays')
plt.xlabel('Essay Type')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['Human-written', 'AI-written'])

plt.show()
```

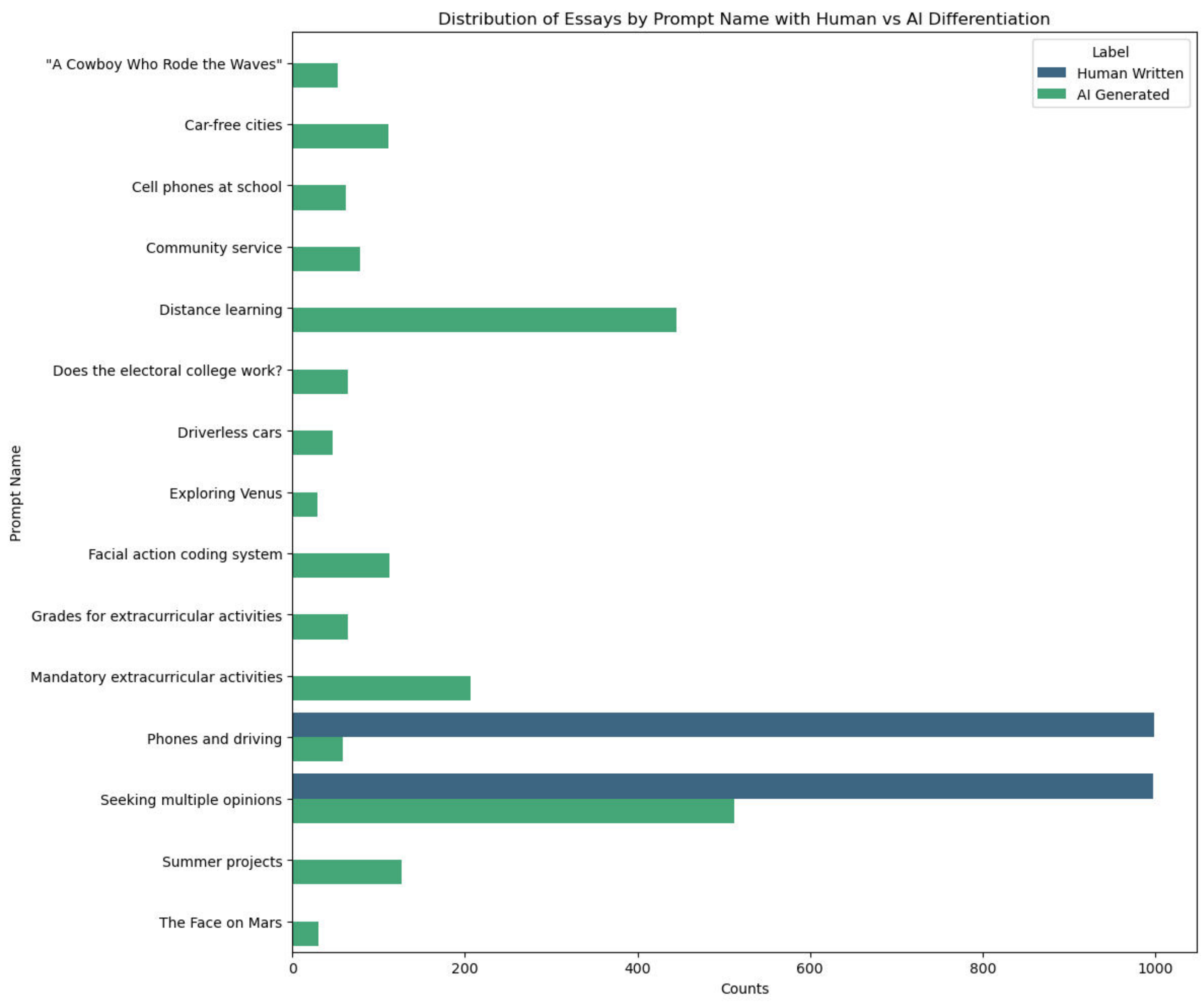


In [7]:
```
# Additional analysis: length of essays boxplot
data['text_length'] = data['text'].apply(len)
plt.figure(figsize=(8, 4))
sns.boxplot(x='label', y='text_length', data=data)
plt.title('Boxplot of Text Length by Essay Type')
plt.xlabel('Essay Type')
plt.ylabel('Text Length')
plt.xticks(ticks=[0, 1], labels=['Human-written', 'AI-written'])
plt.show()
```

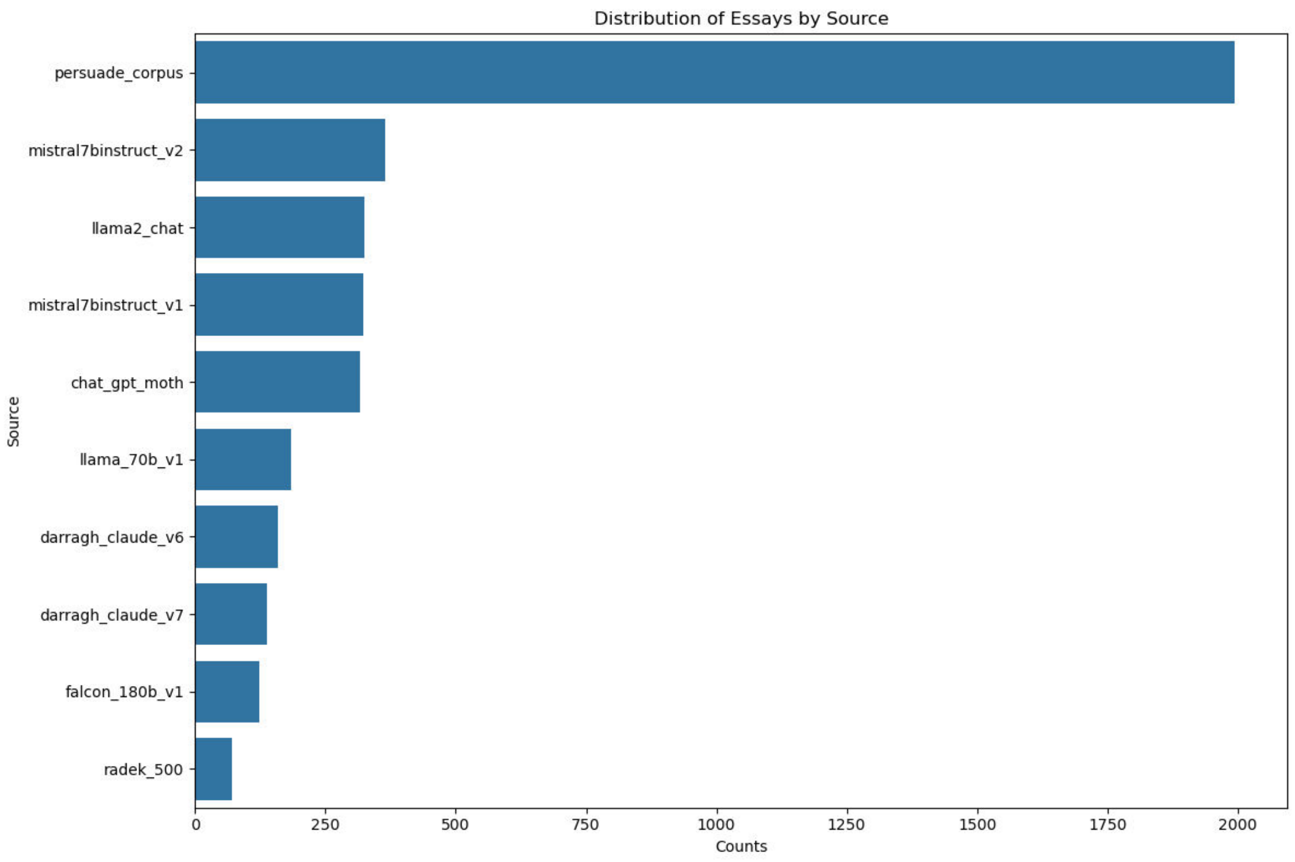


In [8]:

```
# Group data by 'prompt_name' and 'label', and count occurrences
prompt_label_group = data.groupby(['prompt_name', 'label']).size().reset_inc

# Sorting the data by prompt name and counts to better visualize in the plot
prompt_label_sorted = prompt_label_group.sort_values(['prompt_name', 'counts

# Plotting the distribution of essays by prompt name, with differentiation b
plt.figure(figsize=(12, 10))
scatter = sns.barplot(x='counts', y='prompt_name', hue='label', data=prompt_
plt.title('Distribution of Essays by Prompt Name with Human vs AI Differenti
plt.xlabel('Counts')
plt.ylabel('Prompt Name')
plt.legend(title='Label', labels=['Human-Written', 'AI-Generated'])
handles, labels = scatter.get_legend_handles_labels()
scatter.legend(handles=handles, labels=['Human Written', 'AI Generated'], ti
plt.tight_layout()
plt.show()
```



In [9]:
```
# Group data by 'source' and count occurrences
source_group = data.groupby('source').size().reset_index(name='counts')

# Sort the data by counts before plotting
source_sorted = source_group.sort_values('counts', ascending=False)

# Plotting the distribution of essays by source
plt.figure(figsize=(12, 8))
sns.barplot(x='counts', y='source', data=source_sorted)
plt.title('Distribution of Essays by Source')
plt.xlabel('Counts')
plt.ylabel('Source')
plt.tight_layout()  # Adjusts plot to ensure everything fits without overlap
plt.show()
```



In [10]:
```
# Preliminary stopword removal function
def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word not in stop_words]
    return " ".join(filtered_words)

# Apply stopword removal as an added column
data['text_no_stopwords'] = data['text'].apply(remove_stopwords)

# Get rid of original text and unused columns for performance
data.drop(columns=['text', 'RDizzl3_seven'], inplace=True)
```

In [11]:
```
# Split the data into training and validation sets - was not used subsequent
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42

# Display the shape of the training and validation sets
print(f'Training set shape: {train_data.shape}')
print(f'Validation set shape: {val_data.shape}')
```

```
Training set shape: (3198, 5)
Validation set shape: (800, 5)
```

In [12]:
```
# Check the train data
train_data
```

Out[12]:

	label	prompt_name	source	text_length	text_no_stopwords
1575	0	Seeking multiple opinions	persuade_corpus	2687	The people ask help person. When getting advic...
506	0	Phones and driving	persuade_corpus	2265	Phone Usage While Driving Should drivers able ...
3593	1	"A Cowboy Who Rode the Waves"	llama2_chat	1838	Hey, like, El Salvador awesome country Central...
3562	1	Facial action coding system	darragh_claude_v7	1809	The use facial recognition technology like Fac...
3220	1	Grades for extracurricular activities	mistral7binstruct_v2	481	The principal considering making change school...
...	...	...	...	...	...
1130	0	Seeking multiple opinions	persuade_corpus	4249	How advise ever impacted someone? Have ever he...
1294	0	Seeking multiple opinions	persuade_corpus	2396	When seeking advice, people often ask one pers...
860	0	Phones and driving	persuade_corpus	2053	Texting & Driving Using phone driving worse co...
3507	1	Summer projects	falcon_180b_v1	2517	Homework, word make anyone feel stressed overw...
3174	1	Mandatory extracurricular activities	llama2_chat	1903	Hey, like, I thinking whole after-school homew...

3198 rows × 5 columns

In [13]:

```
# Vectorize the text data using TF-IDF
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, stop_words='englis
tfidf = tfidf_vectorizer.fit_transform(data['text_no_stopwords'])
```

In [14]:

```
# Chnaging the TF-IDF array into a dataframe for future use
tfidf_array = tfidf.toarray()
```

```
tfidf_df = pd.DataFrame(tfidf_array, columns=tfidf_vectorizer.get_feature_na
```

In [15]:
```
# Simple Jaccard and Cosine Similarity between two texts to show that simila
# Can be plausibly explored for essays with similar prompts
# Cosine is expected and seen to work well on text vectors
text1 = data.loc[0, 'text_no_stopwords']
text2 = data.loc[1, 'text_no_stopwords']

vectorizer = CountVectorizer(binary=True)
text1_vector = vectorizer.fit_transform([text1])
text2_vector = vectorizer.transform([text2])

jaccard_similarity = jaccard_score(text1_vector.toarray()[0], text2_vector.t
cosine_sim = cosine_similarity(text1_vector, text2_vector)[0][0]
print("\nJaccard Similarity between two example texts:")
print(jaccard_similarity)
print("\nCosine Similarity between two example texts:")
print(cosine_sim)
```

```
Jaccard Similarity between two example texts:
0.21739130434782608

Cosine Similarity between two example texts:
0.46625240412015667
```

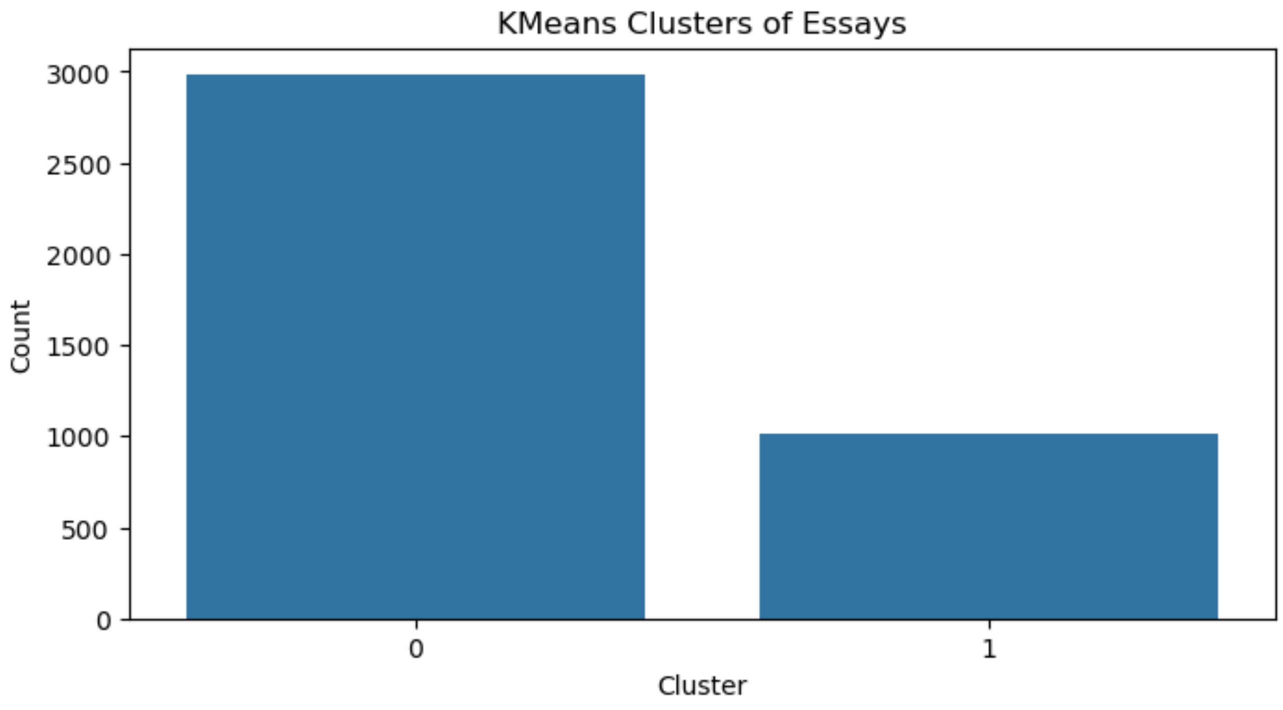
In [16]:
```
text1
```

Out[16]:
```
"Phones Modern humans today always phone. They always phone 5 hours day sto
p .All text back forward group Chats social media. They even driving. They
really bad consequences stuff happens comes phone. Some certain areas Unite
d States ban phones class rooms it. When people phones, know certain apps .
Apps like Facebook Twitter Instagram Snapchat. So like friend moves away wa
nt contact still contact posting videos text messages. People always differ
ent ways communicate phone. Phones changed due generation. Driving one way
get around. People always phones it. Which cause serious Problems. That's t
here's thing that's called texting driving. That's really important thing r
emember. Some people still think It's stupid. No matter still obey that's w
ay save. Sometimes news either accident suicide. It might involve someone l
ooking they're going tweet someone sent. It either injury death. If mysteri
ous number says I'm going kill know live know person's contact ,It makes pu
zzled make start freak out. Which end really badly. Phones fine use also be
st way come help. If go problem can't find help ,always phone you. Even tho
ugh phones used almost every day long safe would come use get trouble. Make
sure like phone middle driving. The news always updated people something st
upid around involves phones. The safest way best way stay safe."
```

In [17]:
```
text2
```

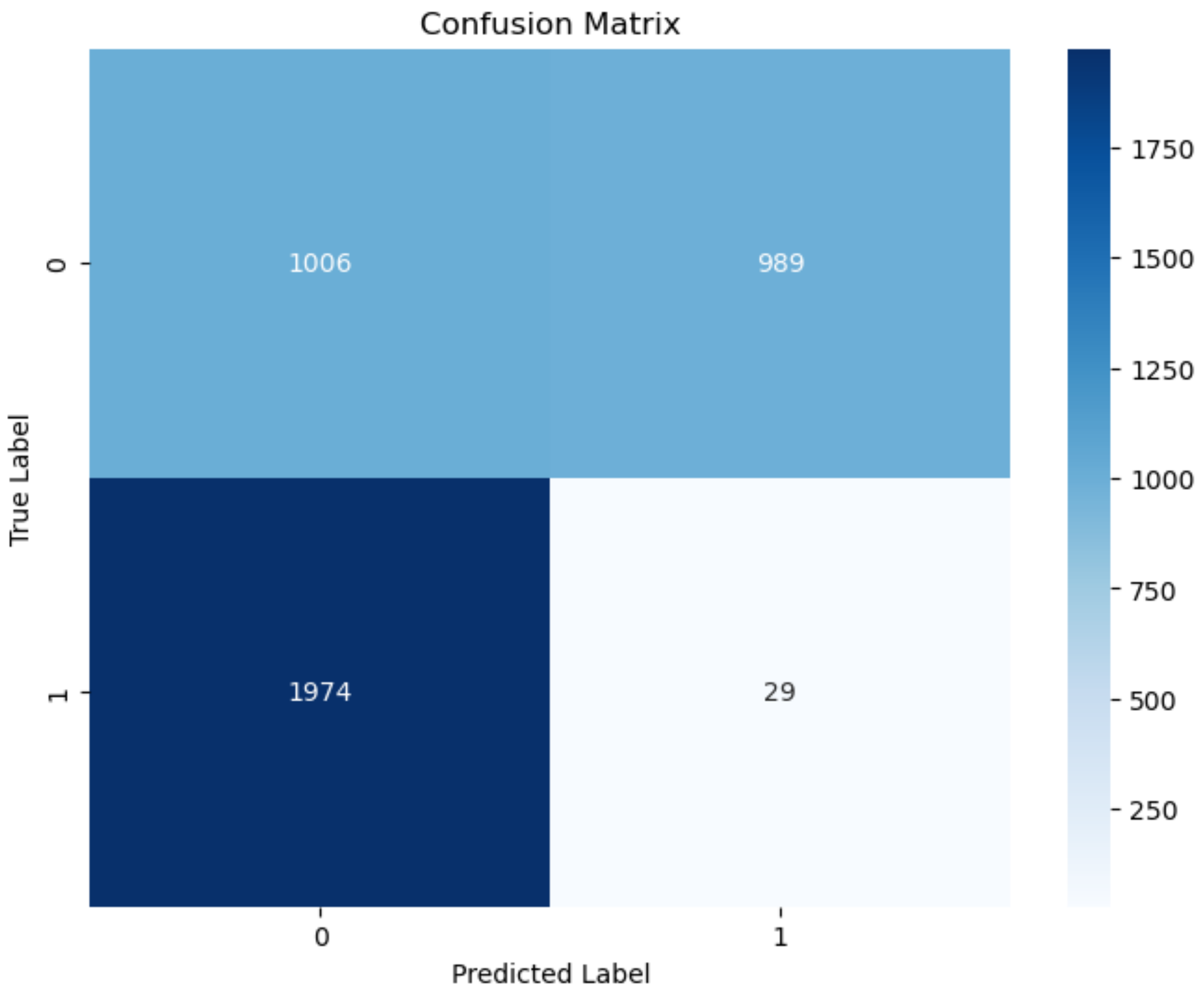
Out[17]:
```
"This essay explain drivers able use electronic devices operating vehicle.
Using phone driving cause bad wrecks, putting people risk. People able use
cell phones operating vehicle bad wrecks, putting others' lives danger, may
cause death. First all, wrecks caused looking phone driving. Most important
ly, always keep eyes directly road behind wheel car. On note, outrageously
car payment looking phone. Moreover, reason accident happened person operat
ing car sue lots money, pay it. Therefore, pay whatever person charges cons
equences served that's behind bars. Another reason able use cell phones ope
rating automobile putting people's lives danger. Thus, looking cell phone g
et someone else hurt uncommunicative act committed. As well unconsciousnes
s, injuries, hospital. The main key texting driving behind wheel car. Mainl
y, cause tragic, terrifying, horrific things others that's death. That's im
portant thing using electronic devices operating vehicle. With intension, k
eeping eyes staying focus road easily turn road hit another victim. Possibl
y another humankind could go away flesh stupidity. Must remembered, always
stay focus road get destination safely. So, can't cause accidents, put peop
le lives danger, possibly death. Important realize, use phone operating veh
icle. At least, wait till make safe stop arrive destination."
```

In [18]:
```python
# Clustering the essays using naive KMeans
kmeans = KMeans(n_clusters=2, n_init=10, random_state=1)
kmeans.fit(tfidf)
data['kmeans_cluster'] = kmeans.labels_

# Plotting the clusters
plt.figure(figsize=(8, 4))
sns.countplot(x='kmeans_cluster', data=data)
plt.title('KMeans Clusters of Essays')

plt.xlabel('Cluster')
plt.ylabel('Count')
plt.show()
```

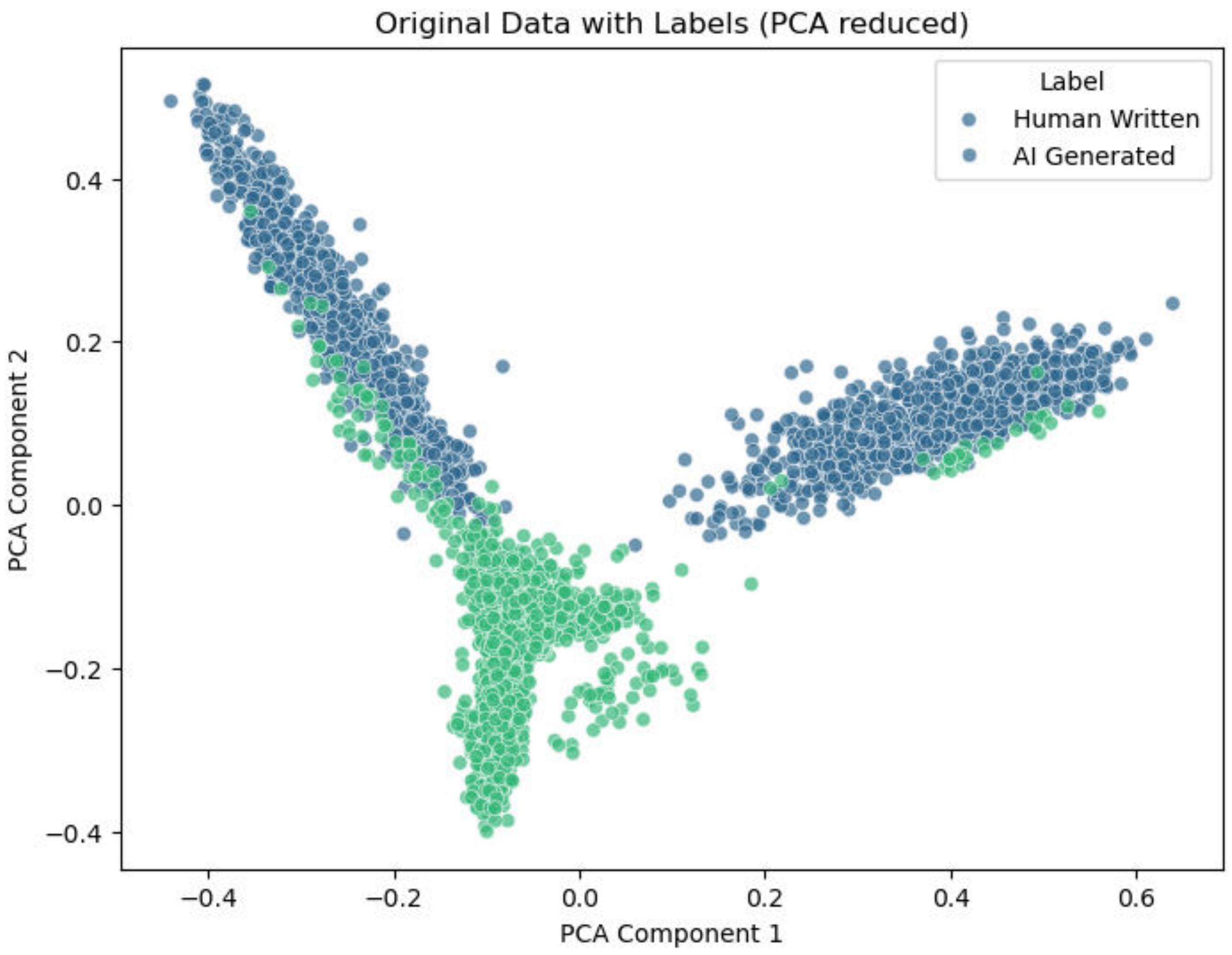


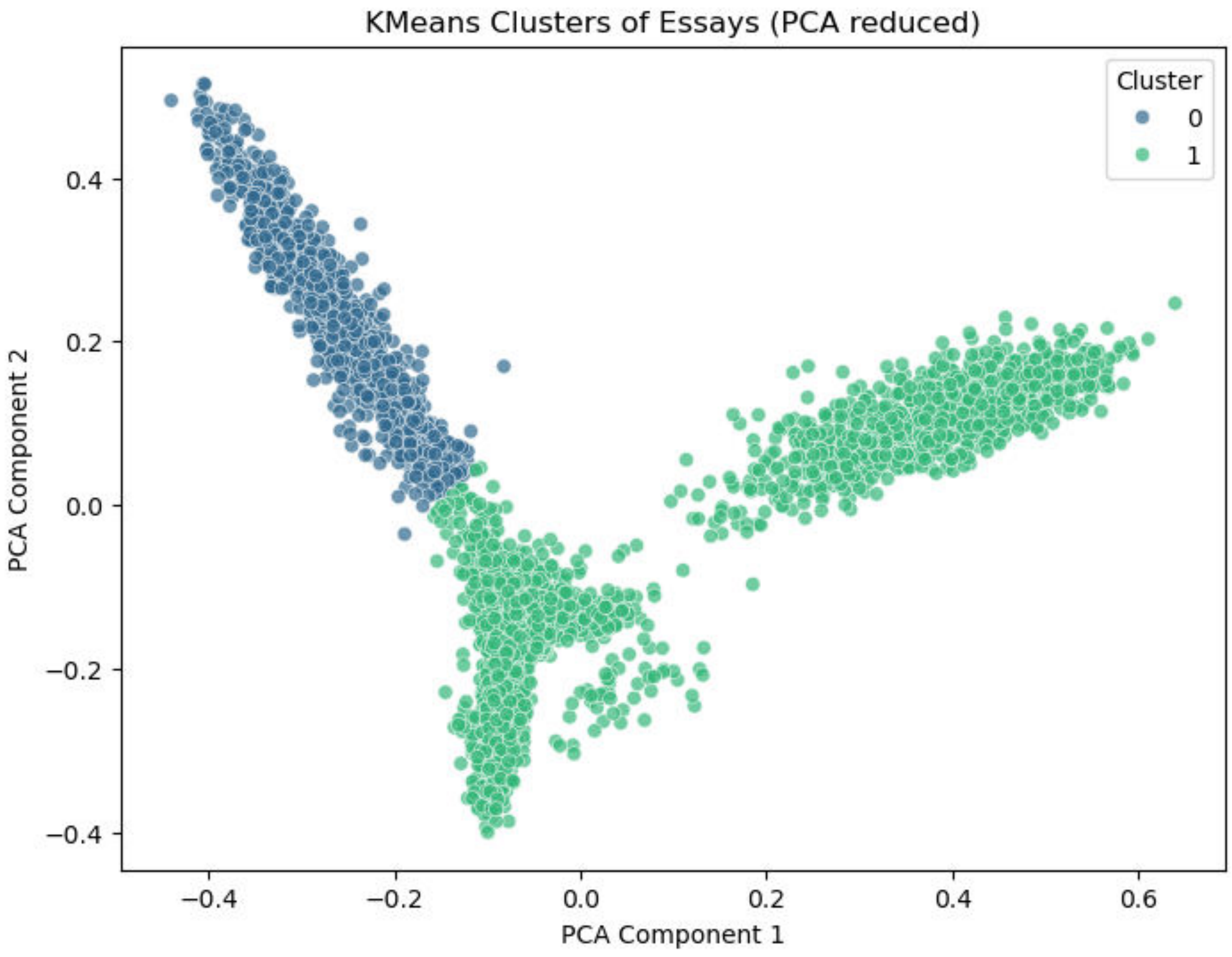
In [19]:
```
# Calculate confusion matrix to compare actual labels with cluster labels
conf_mat = confusion_matrix(data['label'], data['kmeans_cluster'])
print("\nConfusion Matrix:")
print(conf_mat)

# Determine accuracy of the clustering
accuracy = accuracy_score(data['label'], data['kmeans_cluster'])
print(f"\nAccuracy: {accuracy:.2f}")

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
[[1006  989]
 [1974   29]]

Accuracy: 0.26
```



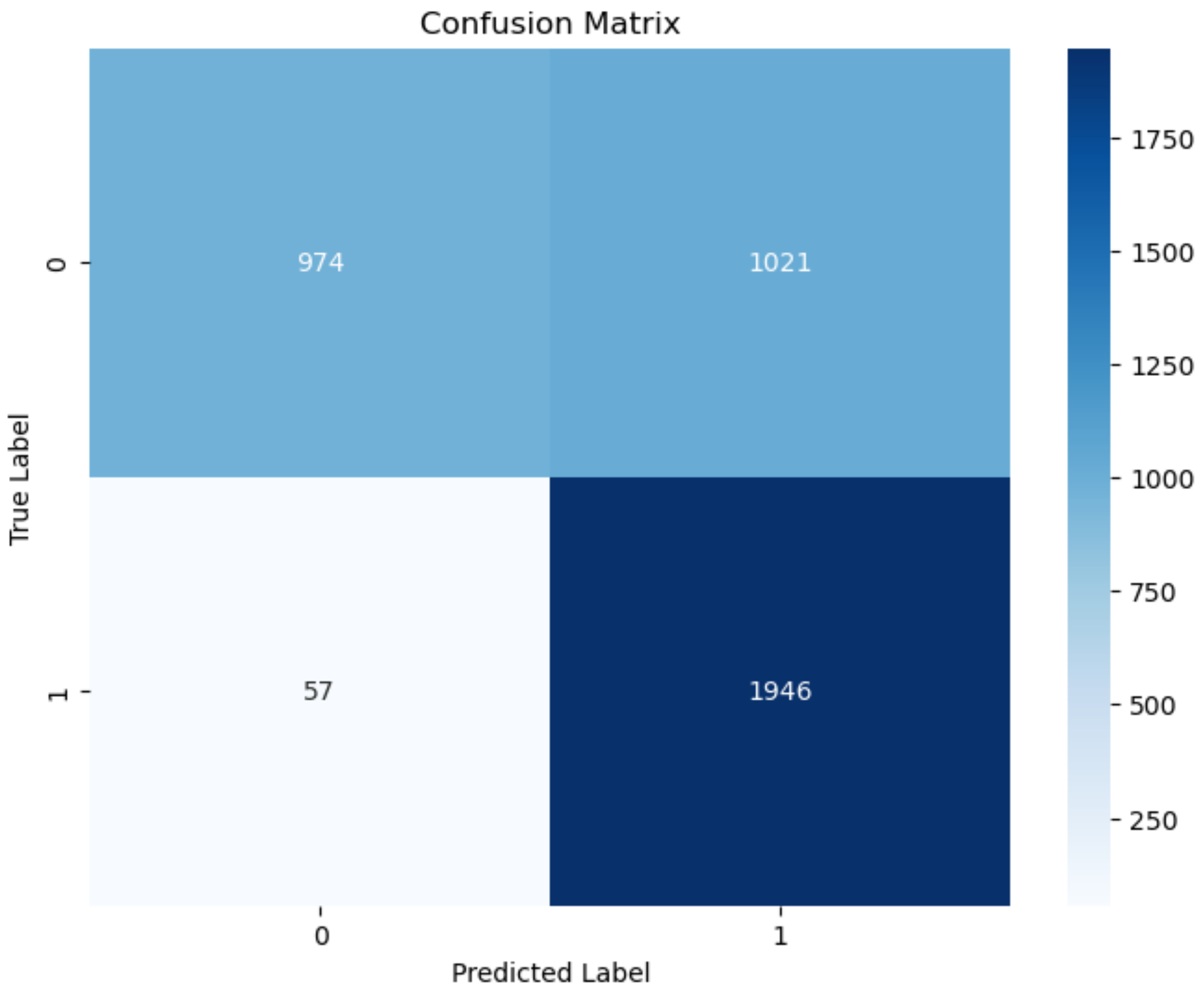
In [20]:
```
# Perform PCA to reduce dimensions to 2 for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(tfidf.toarray())

# Adding the PCA results to the data
data['pca-one'] = pca_result[:, 0]
data['pca-two'] = pca_result[:, 1]

# Fit KMeans again for plotting
kmeans = KMeans(n_clusters=2, random_state=1)
data['kmeans_cluster'] = kmeans.fit_predict(pca_result)

# Plotting the original labeled data
plt.figure(figsize=(8, 6))
sns.scatterplot(x='pca-one', y='pca-two', hue='label', palette='viridis', da
plt.title('Original Data with Labels (PCA reduced)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Label', labels=['Human Written', 'AI Generated'])
plt.show()

# Plotting the K-means clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='pca-one', y='pca-two', hue='kmeans_cluster', palette='vir
plt.title('KMeans Clusters of Essays (PCA reduced)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.show()
```





```
In [21]: # Calculate confusion matrix to compare actual labels with cluster labels
         conf_mat = confusion_matrix(data['label'], data['kmeans_cluster'])
         print("\nConfusion Matrix:")
         print(conf_mat)

         # Determine accuracy of the clustering
         accuracy = accuracy_score(data['label'], data['kmeans_cluster'])
         print(f"\nAccuracy: {accuracy:.2f}")

         # Plot confusion matrix
         plt.figure(figsize=(8, 6))
         sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
         plt.xlabel('Predicted Label')
         plt.ylabel('True Label')
         plt.title('Confusion Matrix')
         plt.show()
```

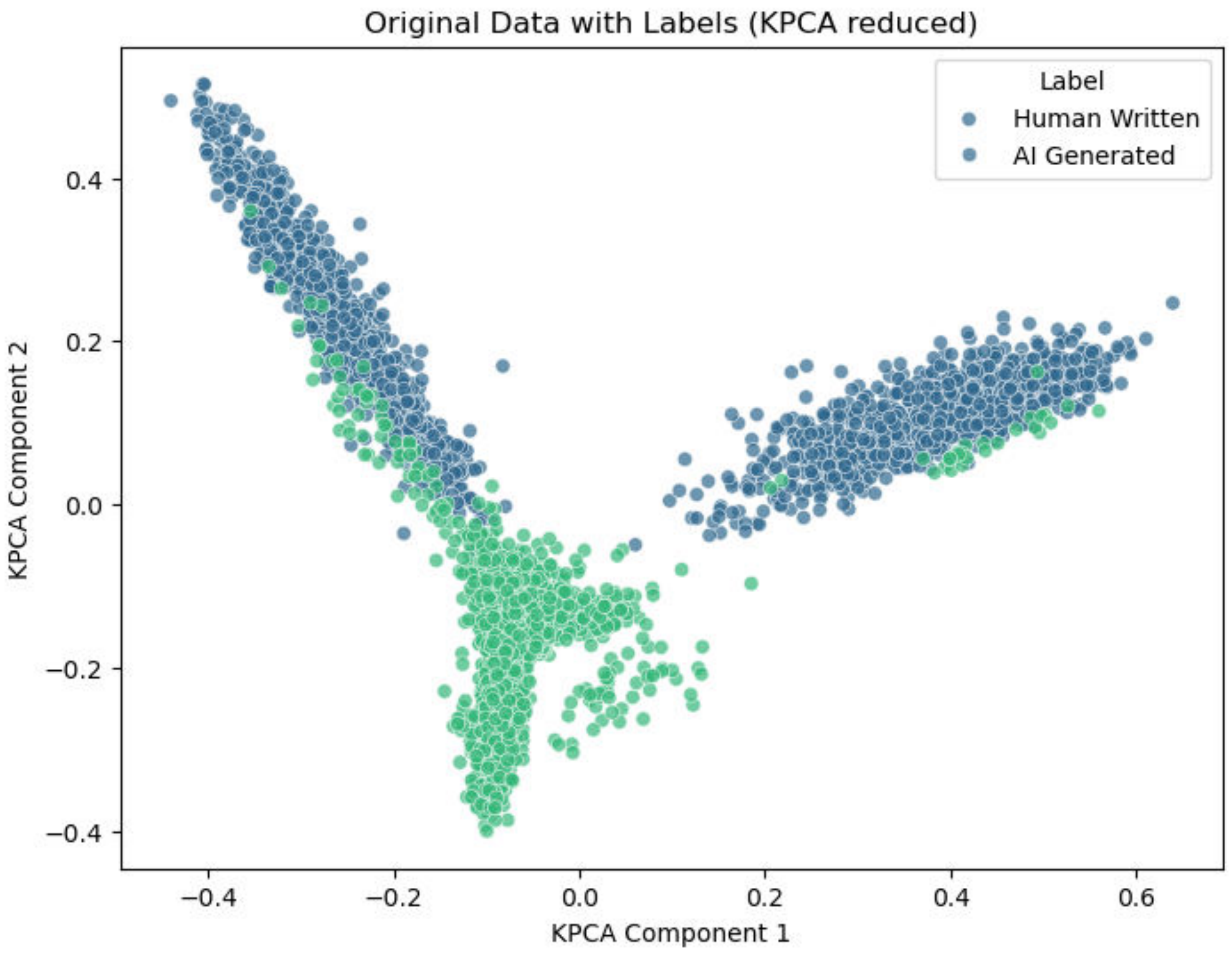
```
Confusion Matrix:
[[ 974 1021]
 [  57 1946]]

Accuracy: 0.73
```

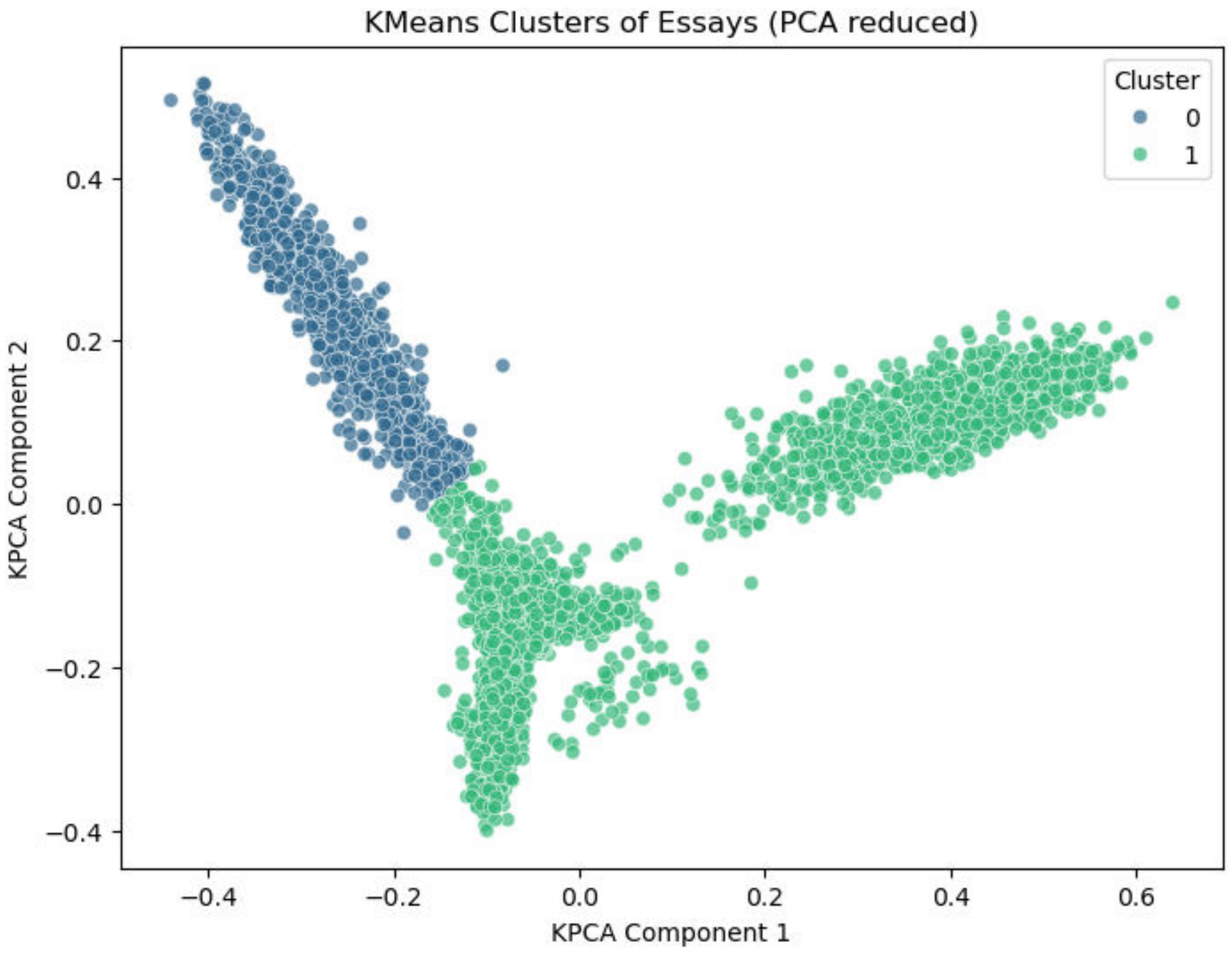


In [22]:
```
# Even I am amazed at what just simple PCA + K means can do.
# The right "Wing" seen in the component plots is basically the top right cc
# I.e. False positives, which are human written essays incorrectly classifie
```

In [23]:
```
# Trying KPCA instead, results weren't much different with a different kerne
kpca = KernelPCA(n_components=2, kernel='cosine')
kpca_result = kpca.fit_transform(tfidf.toarray())

# Adding the PCA results to the data
data['kpca-one'] = kpca_result[:, 0]
data['kpca-two'] = kpca_result[:, 1]

# Fit KMeans again for plotting
kmeans = KMeans(n_clusters=2, random_state=1)
```

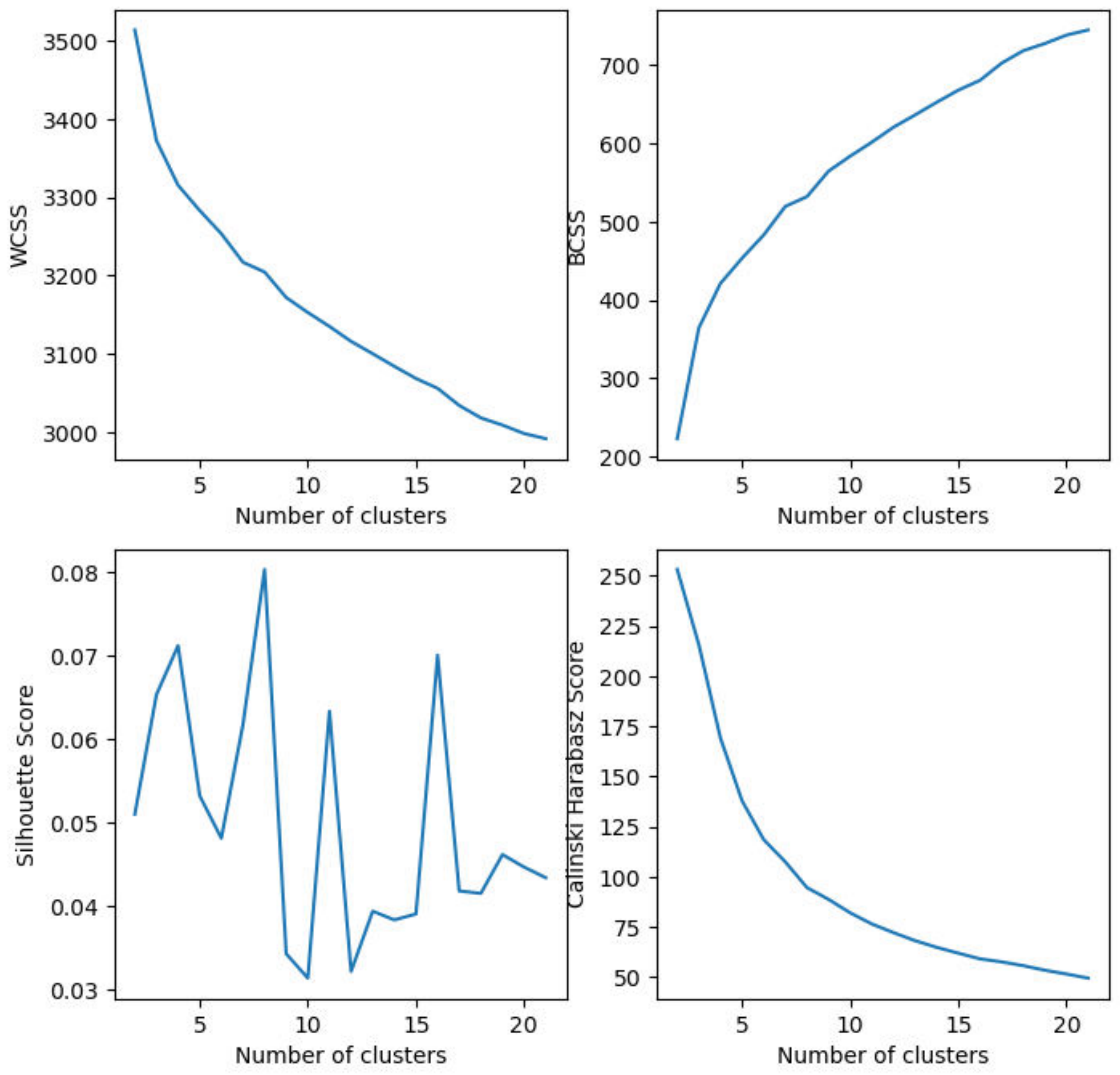
```
data['kmeans_cluster_kpca'] = kmeans.fit_predict(kpca_result)

# Plotting the original labeled data
plt.figure(figsize=(8, 6))
sns.scatterplot(x='kpca-one', y='kpca-two', hue='label', palette='viridis',
plt.title('Original Data with Labels (KPCA reduced)')
plt.xlabel('KPCA Component 1')
plt.ylabel('KPCA Component 2')
plt.legend(title='Label', labels=['Human Written', 'AI Generated'])
plt.show()

# Plotting the Kernal PCA K-means clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='kpca-one', y='kpca-two', hue='kmeans_cluster_kpca', palet
plt.title('KMeans Clusters of Essays (PCA reduced)')
plt.xlabel('KPCA Component 1')
plt.ylabel('KPCA Component 2')
plt.legend(title='Cluster')
plt.show()
```





In [24]:
```
# A cluster search to kind optimal K - code from class
def cluster_search_kmeans(df, nclusts=(2,21), n_init=10):
    ## If there are cluster assignments in the data frame remove them.
    if 'cluster_assignments' in df.columns: df.drop(columns='cluster_assignm
    WCSS=[]
    BCSS=[]
    silhouette=[]
    CH_index = []
    ## Compute total sum of squares
    x_bar = np.mean(df, axis=0)
    TSS = np.sum(np.sum(np.square(df - x_bar)))
    for n in range(nclusts[0],nclusts[1]+1):
        temp_model = KMeans(n_clusters=n, n_init=n_init).fit(df)
        WCSS.append(temp_model.inertia_)
        BCSS.append(TSS - temp_model.inertia_)
        assignments = temp_model.predict(df)
        silhouette.append(silhouette_score(df, assignments))
        CH_index.append(calinski_harabasz_score(df, assignments))
    _, ax = plt.subplots(2,2, figsize=(8,8))
    ax = ax.flatten()
    ax[0].plot(range(nclusts[0],nclusts[1]+1),WCSS)
    ax[0].set_xlabel('Number of clusters')
    ax[0].set_ylabel('WCSS')
    ax[1].plot(range(nclusts[0],nclusts[1]+1),BCSS)
    ax[1].set_xlabel('Number of clusters')
    ax[1].set_ylabel('BCSS')
    ax[2].plot(range(nclusts[0],nclusts[1]+1),silhouette)
    ax[2].set_xlabel('Number of clusters')
    ax[2].set_ylabel('Silhouette Score')
    ax[3].plot(range(nclusts[0],nclusts[1]+1),CH_index)
    ax[3].set_xlabel('Number of clusters')
    ax[3].set_ylabel('Calinski Harabasz Score')

nr.seed(9966)
cluster_search_kmeans(tfidf_df)
```

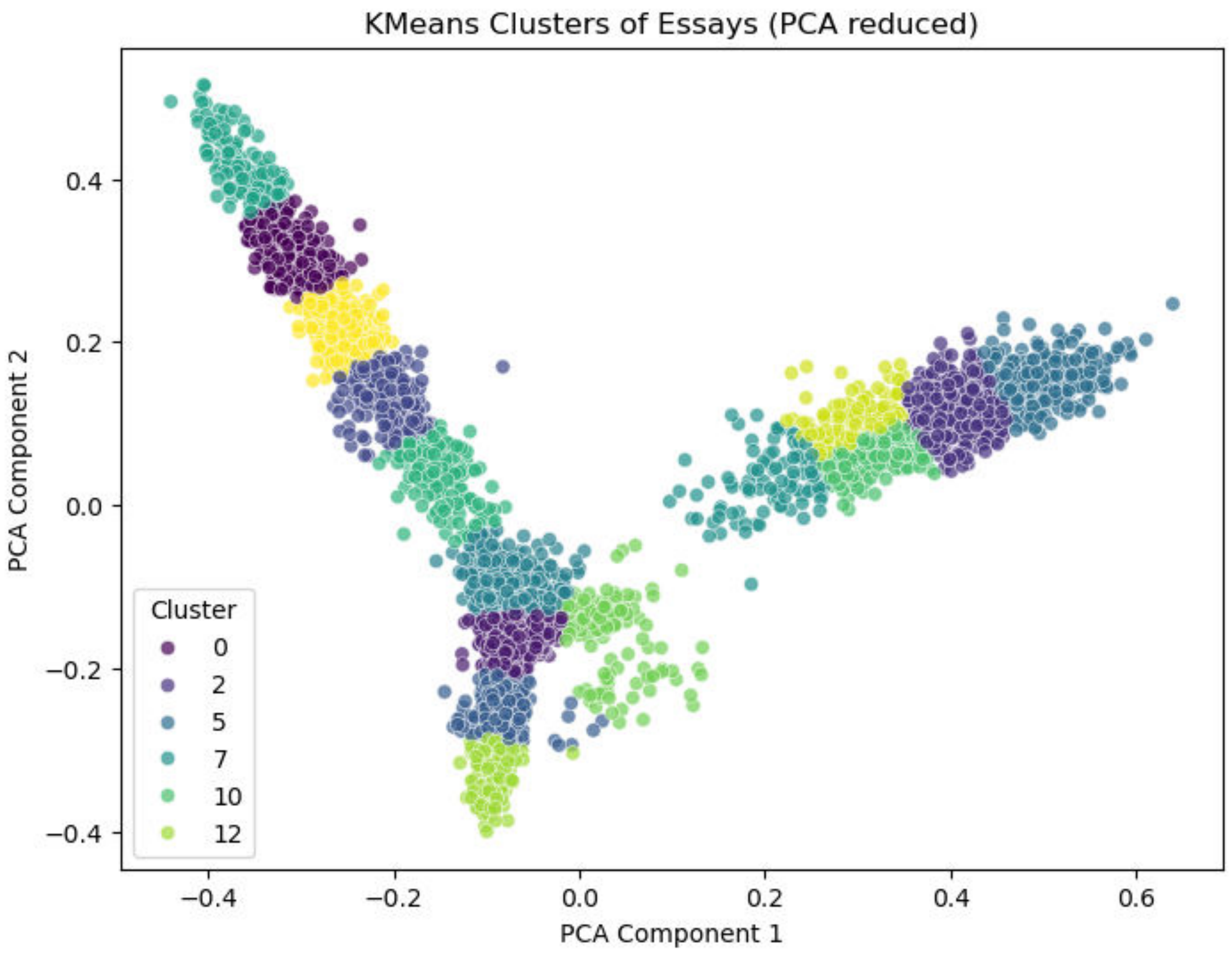
```
/opt/anaconda3/envs/tensorflow_env/lib/python3.10/site-packages/numpy/core/f
romnumeric.py:86: FutureWarning: The behavior of DataFrame.sum with axis=Non
e is deprecated, in a future version this will reduce over both axes and ret
urn a scalar. To retain the old behavior, pass axis=0 (or do not pass axis)
  return reduction(axis=axis, out=out, **passkwargs)
```



In [25]:
```
# CH Score drops quickly, indicating very low number of clusters.
# Sihouette Score is very low at <10, and shows spikes below around 2-3 but
```

In [26]:
```
# Trying KPCA instead, results weren't much different
kpca = KernelPCA(n_components=2, kernel='cosine')
kpca_result = kpca.fit_transform(tfidf.toarray())

# Adding the PCA results to the data
data['kpca-one'] = kpca_result[:, 0]
data['kpca-two'] = kpca_result[:, 1]

# Fit KMeans again for plotting
kmeans = KMeans(n_clusters=15, random_state=1)
data['kmeans_cluster'] = kmeans.fit_predict(kpca_result)
```

```
# Plotting the K-means clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='kpca-one', y='kpca-two', hue='kmeans_cluster', palette='v
plt.title('KMeans Clusters of Essays (PCA reduced)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.show()
```



In [27]:
```
#  LDA model for topic modeling to verify that the 15 clusters are based on
lda = LatentDirichletAllocation(n_components=10, random_state=0)
lda.fit(tfidf)
```

Out[27]:
```
▾      LatentDirichletAllocation
LatentDirichletAllocation(random_state=0)
```

In [28]:
```
# Display the top words for each topic, shows potential clustering based on
def display_topics(model, feature_names, num_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print(f"Topic {topic_idx}:")
        print(" ".join([feature_names[i] for i in topic.argsort()[:-num_top_

num_top_words = 10
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
display_topics(lda, tfidf_feature_names, num_top_words)
```

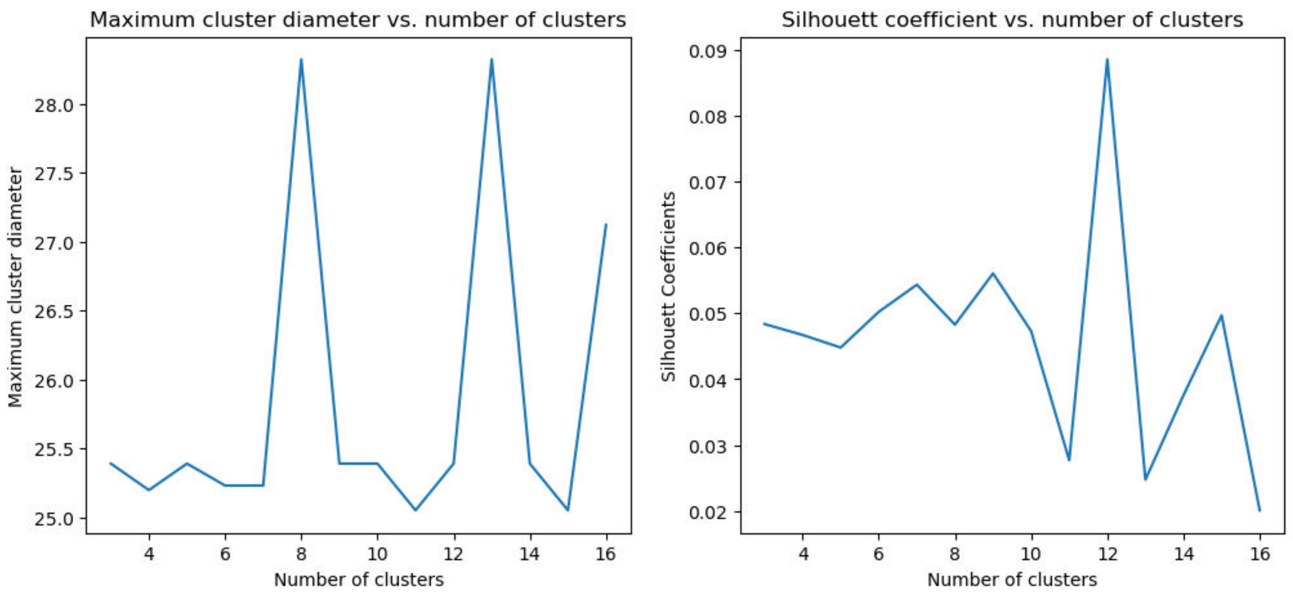
```
Topic 0:
venus face mars wage author natural evidence minimum planet earth
Topic 1:
sarah nail modeling int swimming sharks tutorials bakery chocolate necklace
Topic 2:
students policy working community activities group service extracurricular s
tudent projects
Topic 3:
driving phone phones cell texting use car people drivers road
Topic 4:
like success positive attitude failure life important people goals new
Topic 5:
homeschooled korea prank aint opions wilderness fixes ali christopher skool
Topic 6:
city generic_city visitors french kobe malala metaphor affirmations shortcut
dancing
Topic 7:
advice people ask opinions multiple person help asking different make
Topic 8:
students school classes learning technology career online teachers student t
ime
Topic 9:
electoral vote college states candidates popular president election election
s votes
```

In [29]:

```
# CLARA evaluation code from class used here for optimized hyperparameter ex

def find_max_diameter(df, metric='manhattan'):
    max_diameters = []
    ## Put your code below
    for i in df.loc[:,'cluster_assignments'].unique():
        temp = df[df.loc[:,'cluster_assignments']==i].drop(columns='cluster_
        max_diameters.append(np.amax(pairwise_distances(temp, metric=metric)
    return np.amax(max_diameters)

def evaluate_CLARA(df, metric='manhattan', linkage='complete', nclusts=(3,16
    silhouette_coefficients = []
    max_diameters = []
    for k in range(nclusts[0],nclusts[1]+1):
        ## Put your code below
        ## First compute the cluster assignmenets for the number of clusters
        if 'cluster_assignments' in df.columns: df.drop(columns='cluster_ass
        temp_mod=CLARA(n_clusters=k, metric=metric)
        cluster_assignments = temp_mod.fit_predict(df)

        ## Compute and append the silhouette coefficeint to the list
        silhouette_coefficients.append(silhouette_score(df, cluster_assignme

        ## Find the max diameter of the clusters
        ## First add the cluster assignment column to the data frame
        df['cluster_assignments'] = cluster_assignments
        max_diameters.append(find_max_diameter(df, metric=metric))

    ## Plot the results
    _, ax = plt.subplots(1,2, figsize=(12,5))
    ax[0].plot(range(nclusts[0],nclusts[1]+1), max_diameters);
    ax[0].set_xlabel('Number of clusters')
    ax[0].set_ylabel('Maximum cluster diameter')
    ax[0].set_title('Maximum cluster diameter vs. number of clusters')
    ax[1].plot(range(nclusts[0],nclusts[1]+1), silhouette_coefficients);
    ax[1].set_xlabel('Number of clusters')
    ax[1].set_ylabel('Silhouett Coefficients')
    ax[1].set_title('Silhouett coefficient vs. number of clusters')
    return pd.DataFrame({'NumberClusters':range(nclusts[0],nclusts[1]+1),
                         'ClusterDiameter':max_diameters ,
                         'SilhouetteCoefficient':silhouette_coefficients})

evaluate_CLARA(tfidf_df)
```

Out[29]:

	NumberClusters	ClusterDiameter	SilhouetteCoefficient
0	3	25.390576	0.048365
1	4	25.198059	0.046711
2	5	25.390576	0.044798
3	6	25.231276	0.050210
4	7	25.231276	0.054334
5	8	28.321580	0.048270
6	9	25.390576	0.056041
7	10	25.390576	0.047293
8	11	25.051834	0.027713
9	12	25.390576	0.088495
10	13	28.321580	0.024770
11	14	25.390576	0.037580
12	15	25.051834	0.049683
13	16	27.121196	0.020127

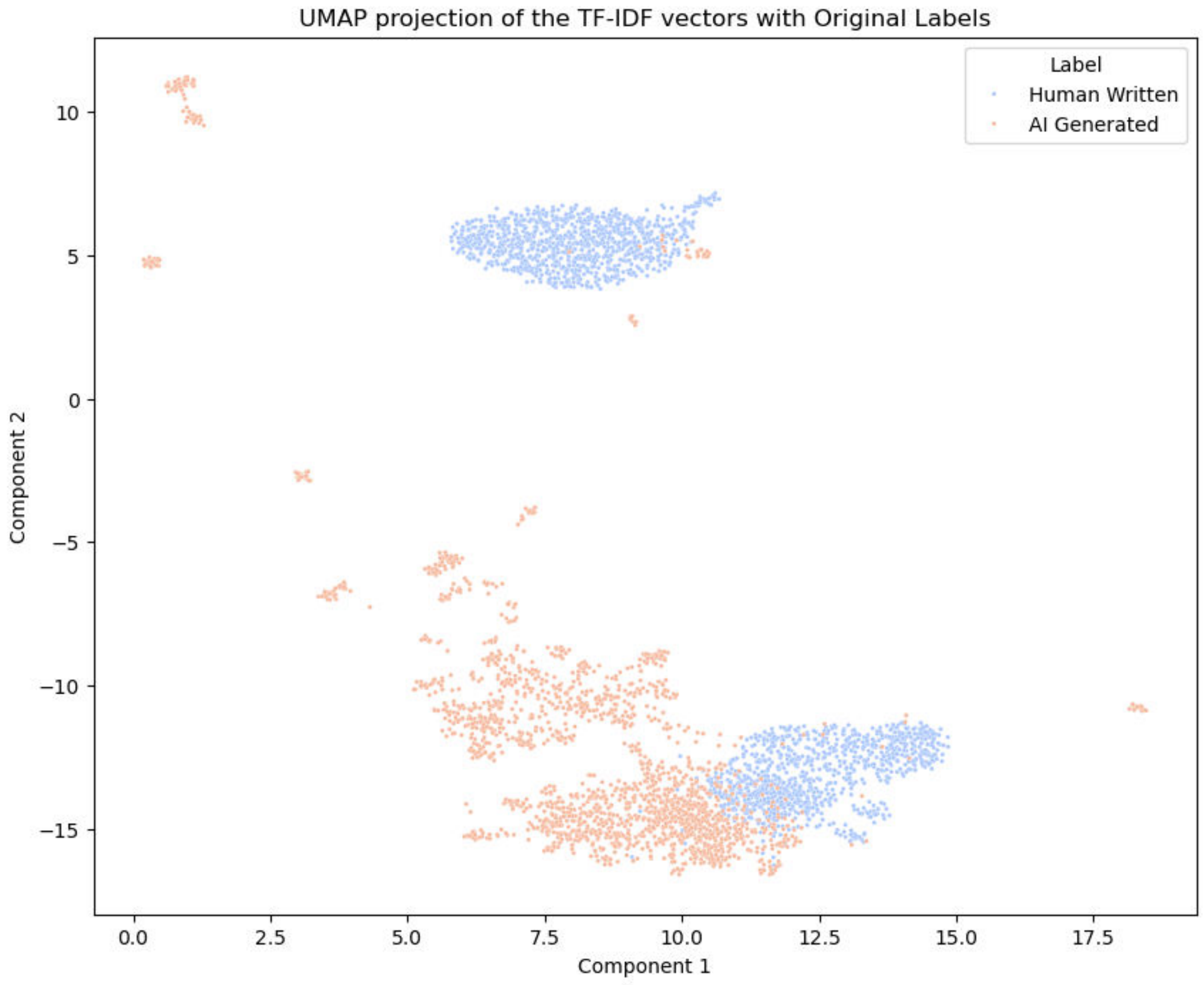


In [30]:
```
# UMAP setup
text_vectors_dense=tfidf.todense()
text_vectors_array = np.asarray(text_vectors_dense)

np.random.seed(1111)
umap_model = umap.UMAP(min_dist=0.5, n_neighbors=80, metric='cosine')
```

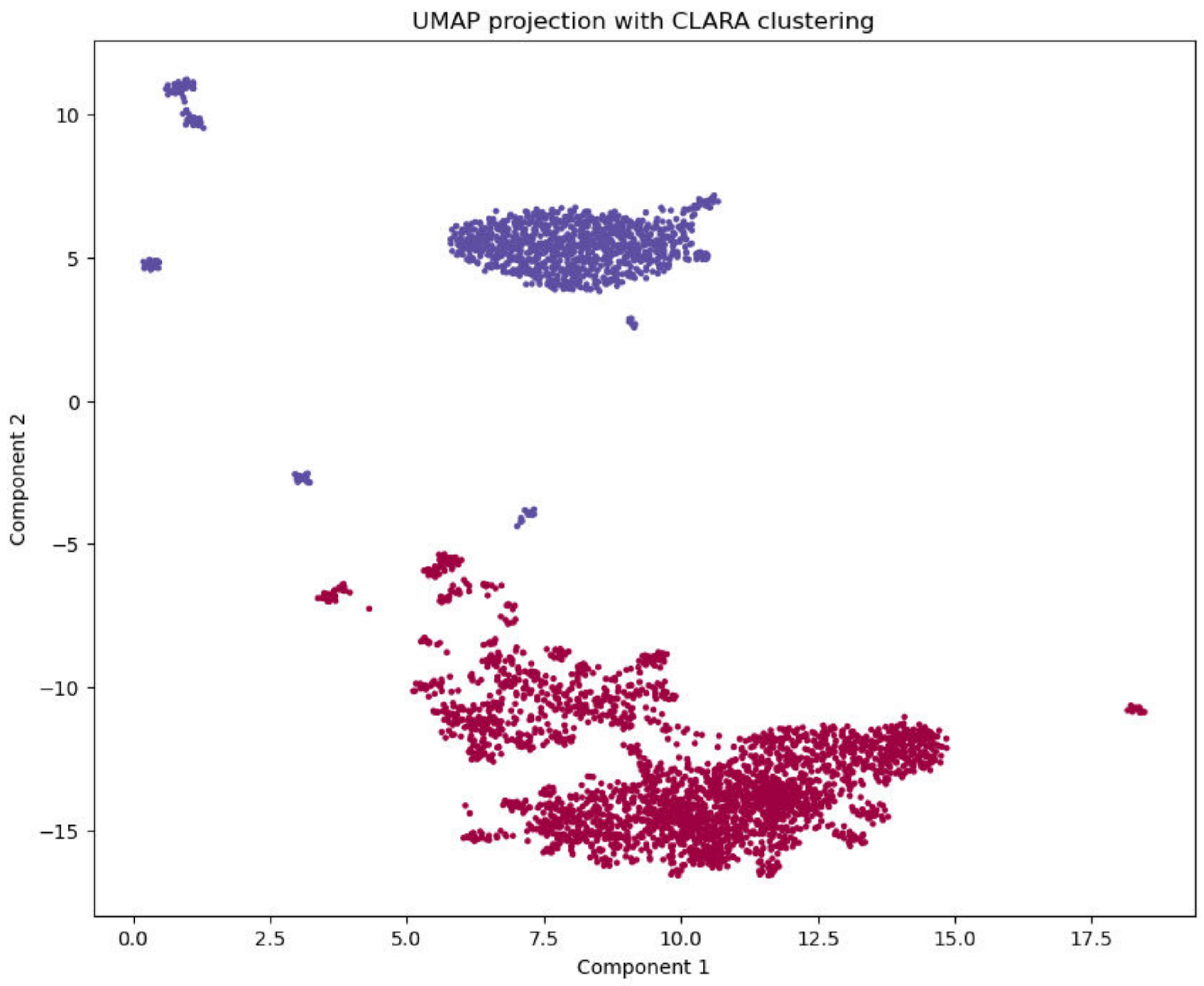
```
mapped_data = umap_model.fit_transform(tfidf_df)

# Convert the result to a DataFrame
mapped_data_df = pd.DataFrame(mapped_data, columns=['component1', 'component
print(mapped_data_df)
```

```
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_ac
tive_levels instead.
      component1  component2
0       8.408147    5.409148
1       9.368107    4.565830
2       7.577414    6.105409
3       9.549308    5.201487
4       9.561337    5.427907
...          ...         ...
3993    8.776796  -15.604988
3994    5.317032   -9.996903
3995    7.600809  -13.717668
3996    9.574719  -12.825543
3997    9.407278  -15.076341

[3998 rows x 2 columns]
```

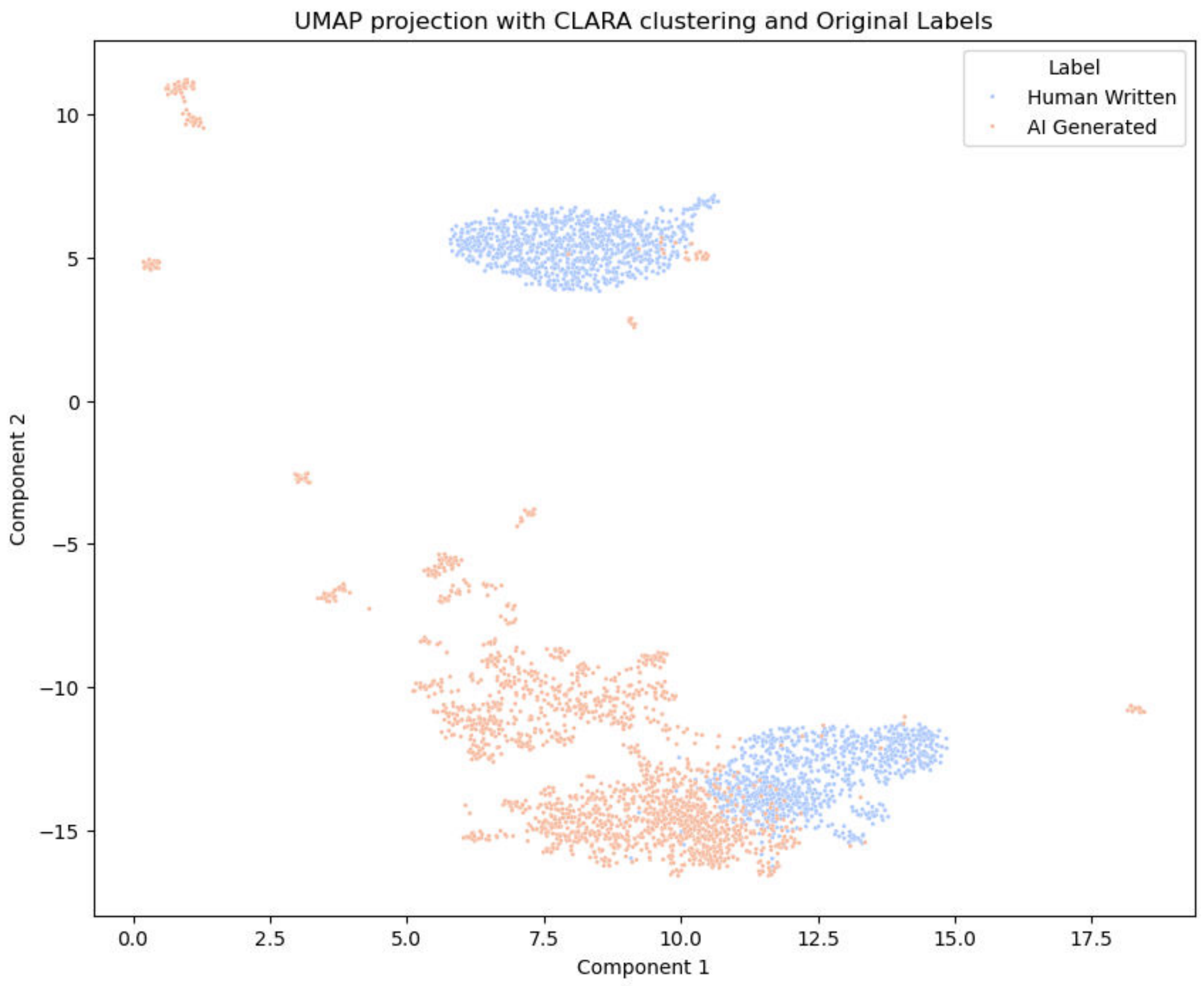
In [31]:
```
# Plot the UMAP results with the original labels
mapped_data_df['label'] = data['label'].values

plt.figure(figsize=(10, 8))
scatter = sns.scatterplot(x='component1', y='component2', hue='label', palet
plt.title('UMAP projection of the TF-IDF vectors with Original Labels')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.legend(title='Label', labels=['Human Written', 'AI Generated'])
handles, labels = scatter.get_legend_handles_labels()
scatter.legend(handles=handles, labels=['Human Written', 'AI Generated'], ti
plt.show()
```



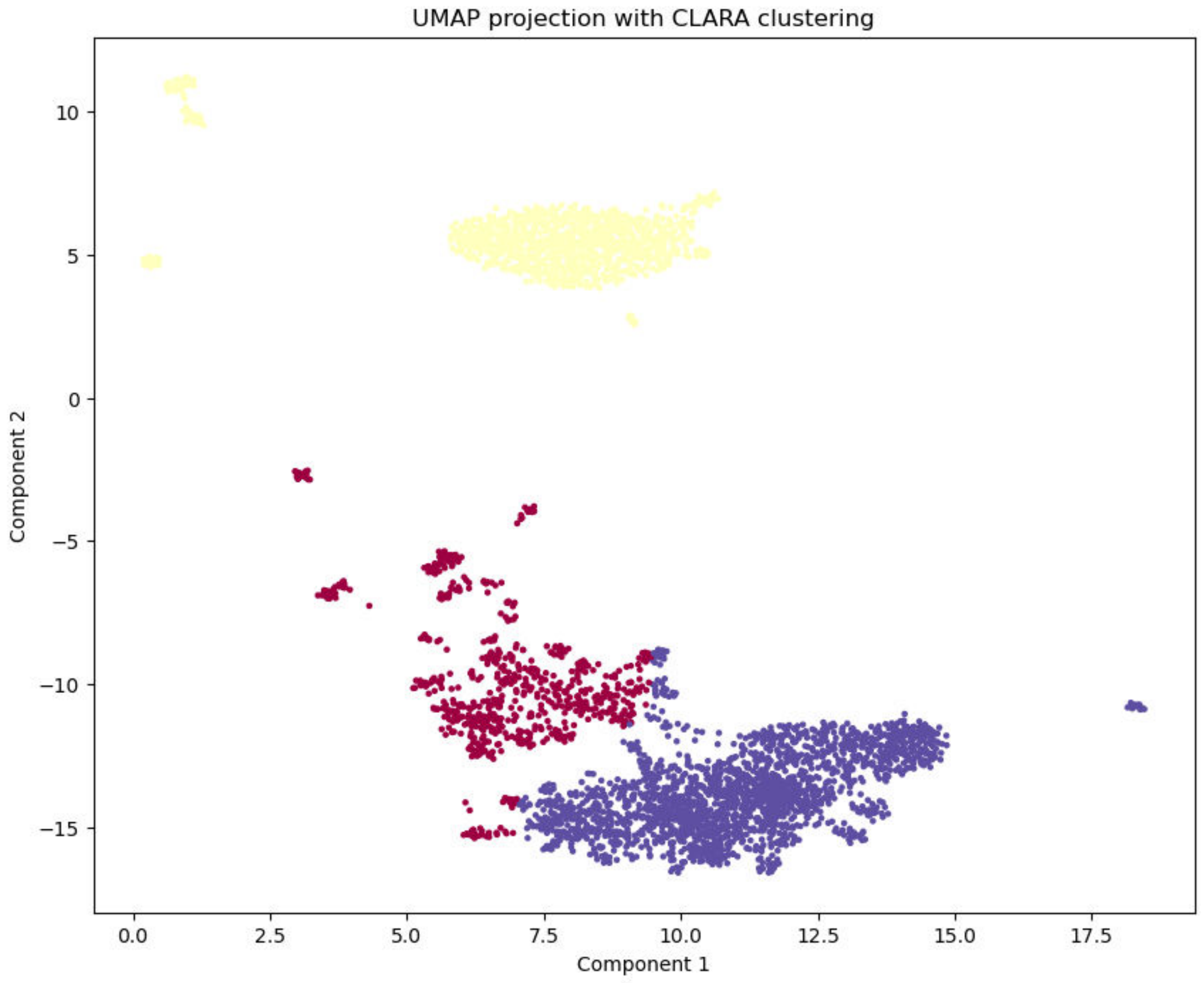
In [32]:
```
# Apply the CLARA algorithm with n = 2
n_clusters = 2
clara_model = CLARA(n_clusters=n_clusters, metric='manhattan', random_state=
cluster_assignments = clara_model.fit_predict(mapped_data)

# Plot the results with cluster labels
plt.figure(figsize=(10, 8))
plt.scatter(mapped_data[:, 0], mapped_data[:, 1], c=cluster_assignments, s=5
plt.title('UMAP projection with CLARA clustering')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.show()
```



In [33]:
```
# Add the CLARA cluster assignments to your DataFrame
mapped_data_df['label'] = data['label'].values

# Plot the UMAP projection with CLARA clusters and original labels
plt.figure(figsize=(10, 8))
scatter=sns.scatterplot(x='component1', y='component2', hue='label',
                palette='coolwarm', data=mapped_data_df, s=5)
plt.title('UMAP projection with CLARA clustering and Original Labels')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.legend(title='Label', labels=['Human Written', 'AI Generated'])
handles, labels = scatter.get_legend_handles_labels()
scatter.legend(handles=handles, labels=['Human Written', 'AI Generated'], ti
plt.show()
```



In [34]:
```
# Apply the CLARA algorithm with n = 3 for comparison and given the hyperpar
n_clusters = 3
clara_model = CLARA(n_clusters=n_clusters, metric='manhattan', random_state=
cluster_assignments = clara_model.fit_predict(mapped_data)

# Plot the results with cluster labels
plt.figure(figsize=(10, 8))
plt.scatter(mapped_data[:, 0], mapped_data[:, 1], c=cluster_assignments, s=5
plt.title('UMAP projection with CLARA clustering')
plt.xlabel('Component 1')
plt.ylabel('Component 2')
plt.show()
```



```
In [35]: # Compute cosine similarity matrix with simple pca results
         cosine_sim_matrix = cosine_similarity(pca_result)
```

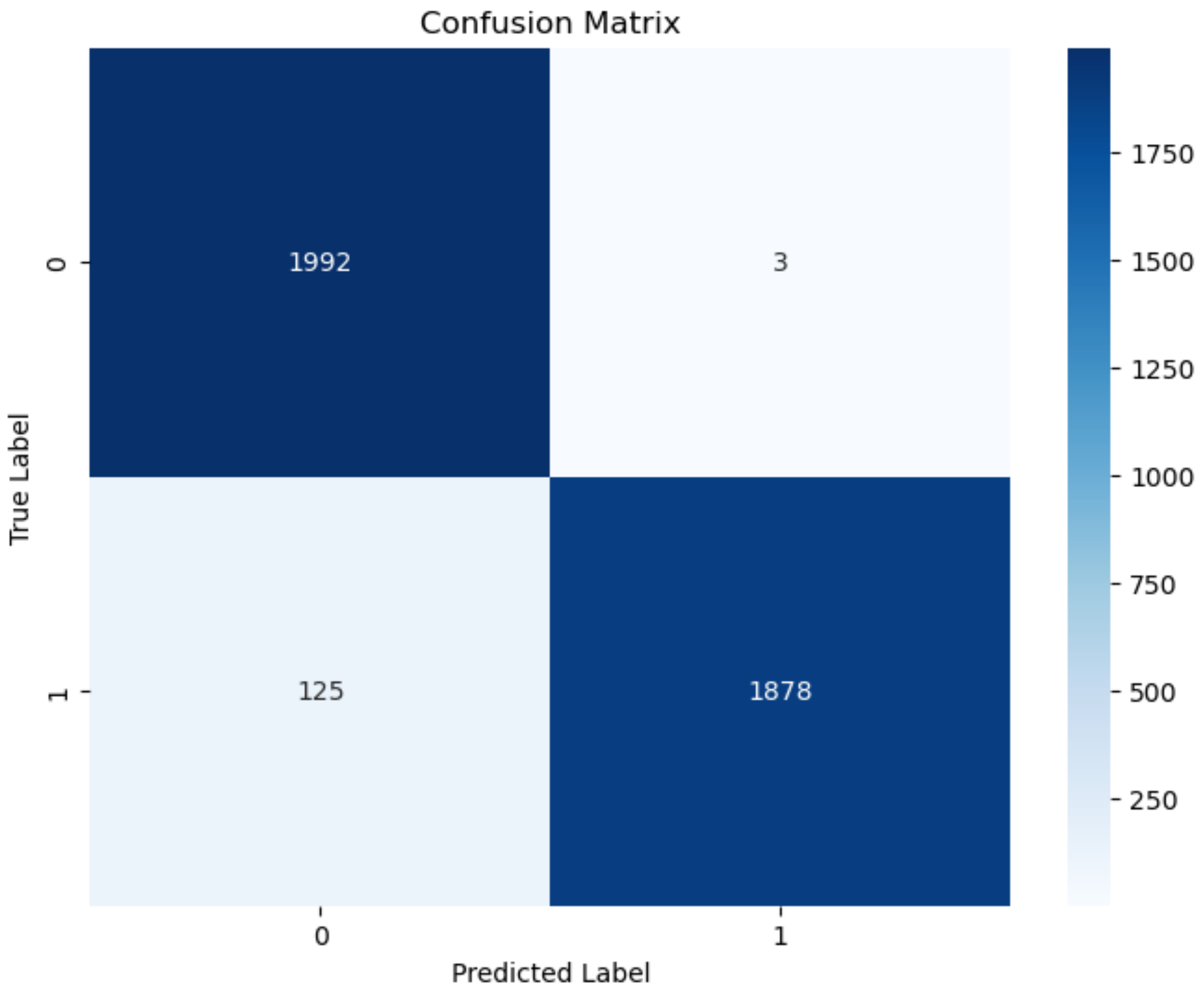
```
In [36]: # Apply hierarchical clustering with prediction on 1 - cosine similarity
         hierarchical_clustering = AgglomerativeClustering(n_clusters=2)
         %time hierarchical_labels = hierarchical_clustering.fit_predict(1 - cosine_s
         data['hierarchical_cluster'] = hierarchical_labels
```

```
CPU times: user 7.32 s, sys: 2.22 s, total: 9.55 s
Wall time: 7.21 s
```

```
In [37]: # Calculate confusion matrix to compare actual labels with cluster labels
         conf_mat = confusion_matrix(data['label'], data['hierarchical_cluster'])
         print("\nConfusion Matrix:")
         print(conf_mat)

         # Determine accuracy of the clustering
         accuracy = accuracy_score(data['label'], data['hierarchical_cluster'])
         print(f"\nAccuracy: {accuracy:.2f}")
```

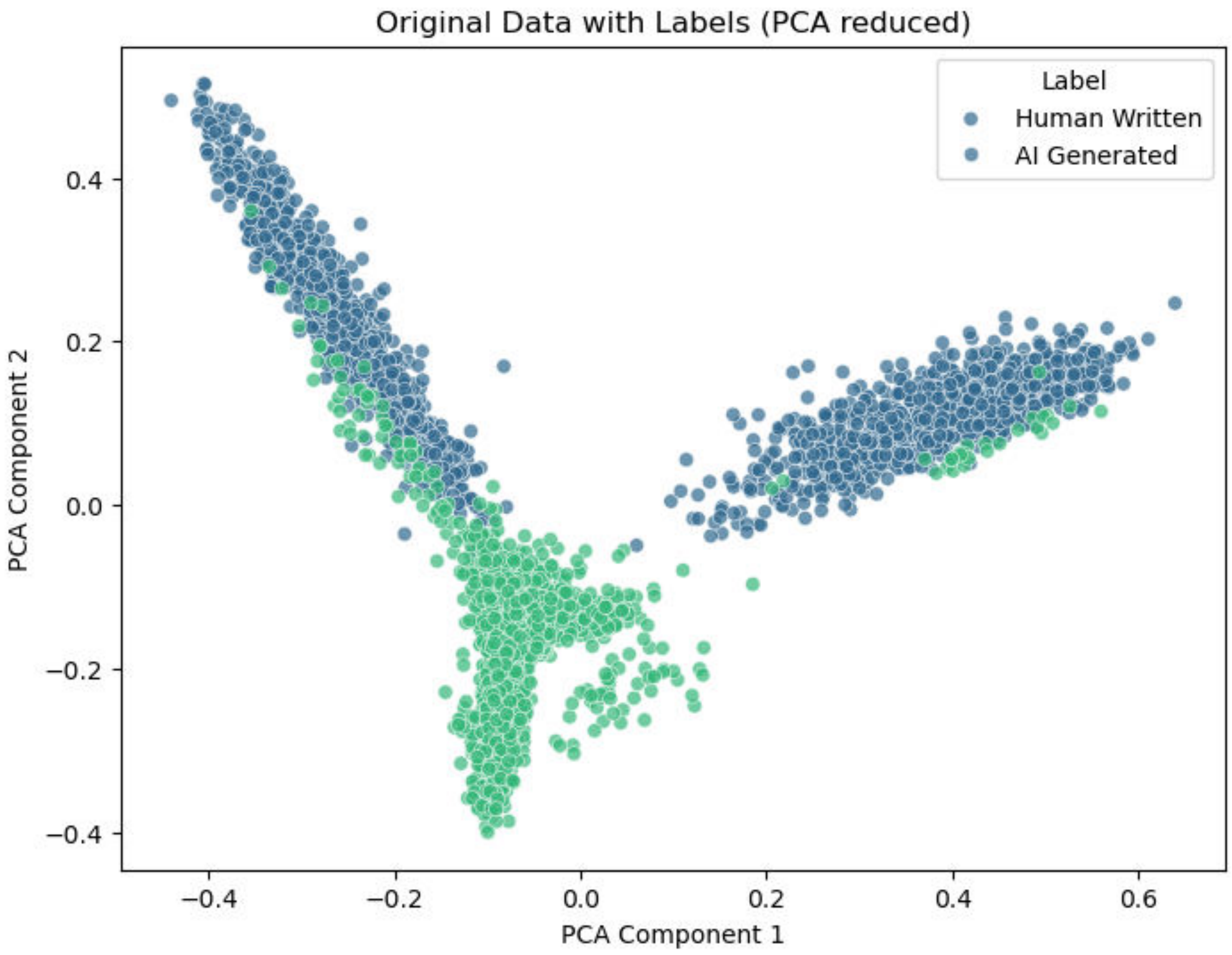
```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

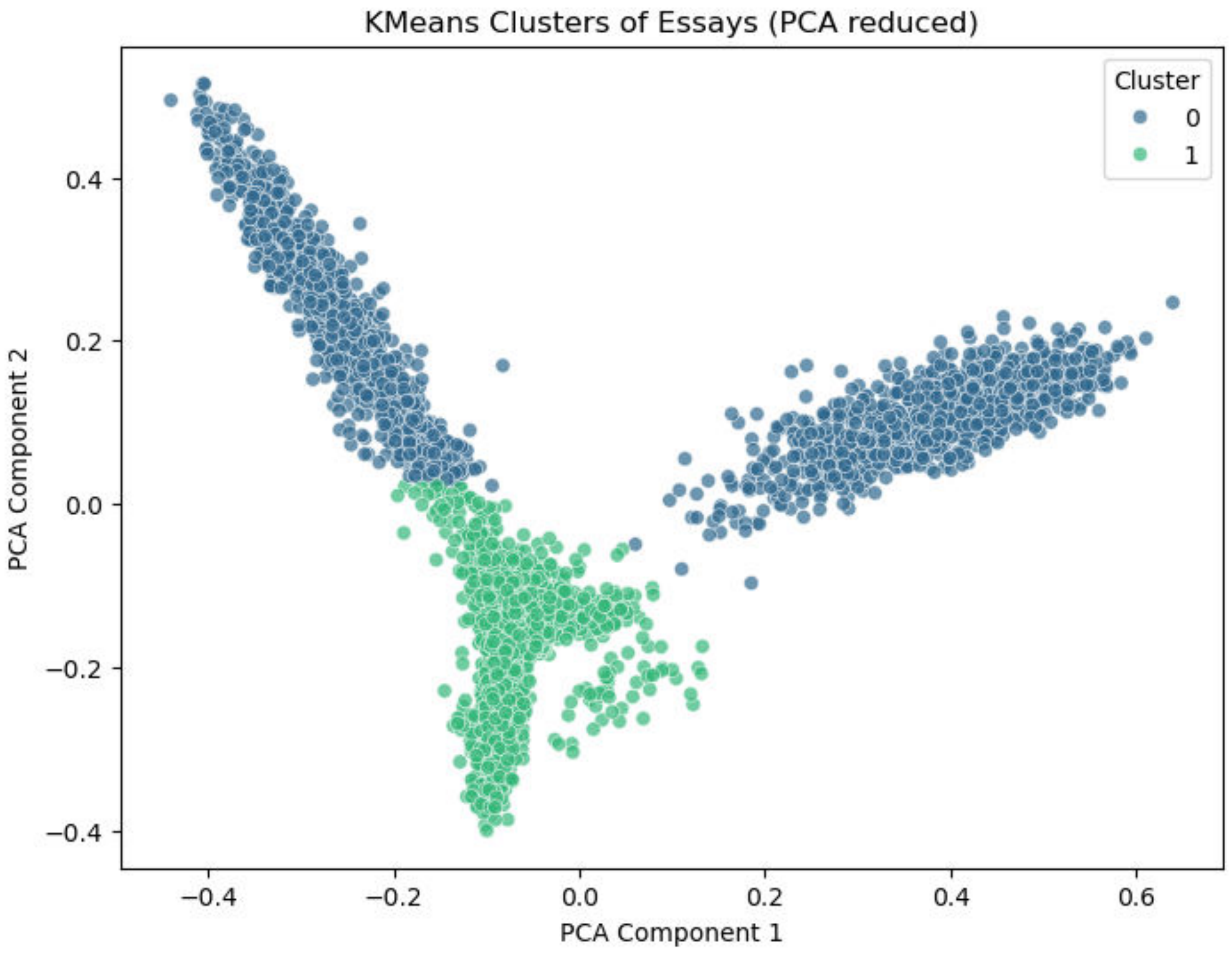
```
Confusion Matrix:
[[1992    3]
 [ 125 1878]]

Accuracy: 0.97
```



In [39]:
```
# Perform PCA to reduce dimensions to 2 for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(tfidf.toarray())

# Compute cosine similarity matrix
cosine_sim_matrix = cosine_similarity(pca_result)
```

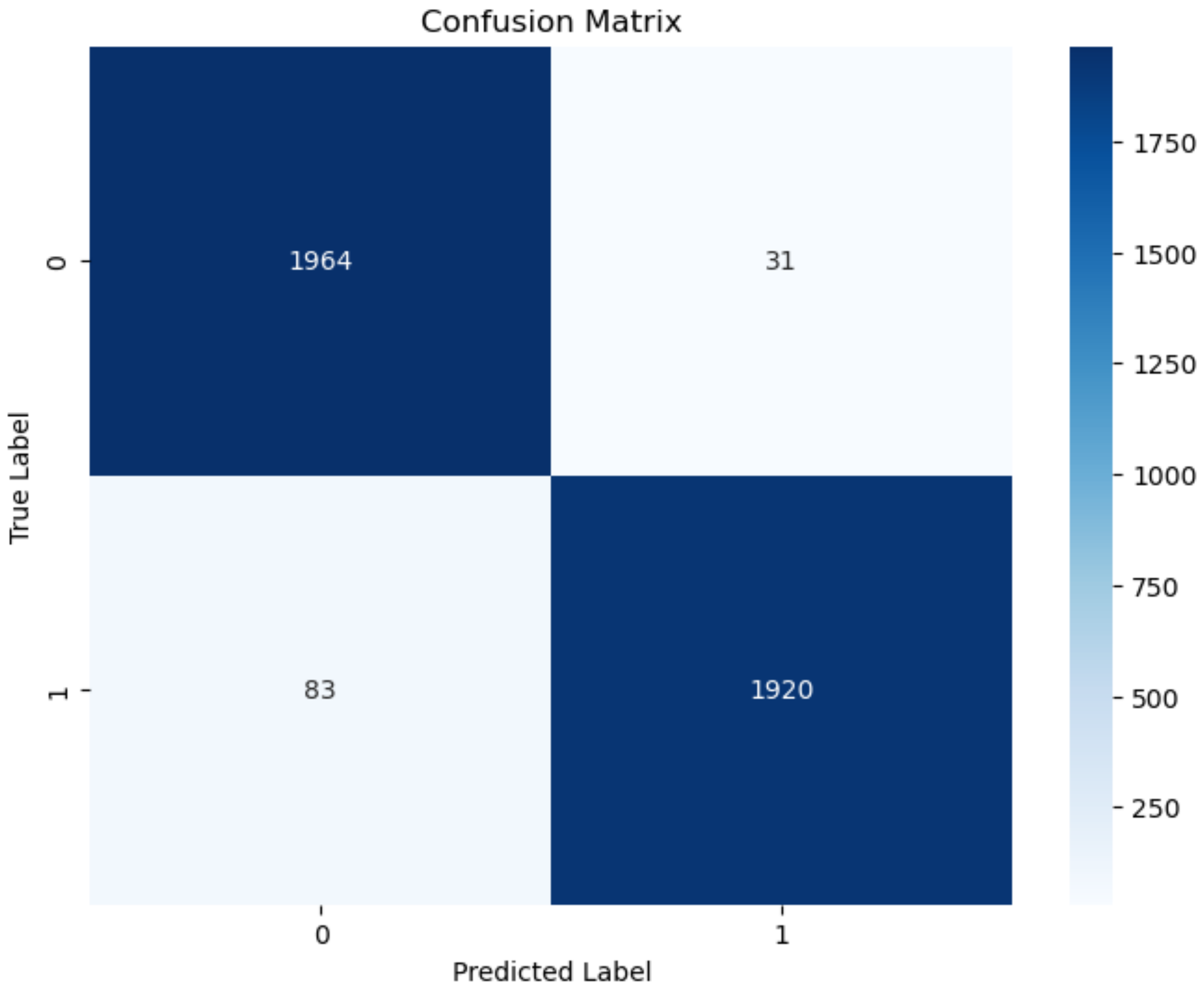
```
# Adding the PCA results to the data
data['pca-one'] = pca_result[:, 0]
data['pca-two'] = pca_result[:, 1]

# Fit KMeans again for plotting on 1 - cosine similarity this time
kmeans = KMeans(n_clusters=2, random_state=1)
data['kmeans_cluster'] = kmeans.fit_predict(1 - cosine_sim_matrix)

# Plotting the original labeled data
plt.figure(figsize=(8, 6))
sns.scatterplot(x='pca-one', y='pca-two', hue='label', palette='viridis', da
plt.title('Original Data with Labels (PCA reduced)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Label', labels=['Human Written', 'AI Generated'])
plt.show()

# Plotting the K-means clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='pca-one', y='pca-two', hue='kmeans_cluster', palette='vir
plt.title('KMeans Clusters of Essays (PCA reduced)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.show()
```


Original Data with Labels (PCA reduced)
Label
Human Written
AI Generated
PCA Component 1
PCA Component 2
−0.4
−0.2
0.0
0.2
0.4
0.6



In [40]:
```
# Calculate confusion matrix to compare actual labels with cluster labels
conf_mat = confusion_matrix(data['label'], data['kmeans_cluster'])
print("\nConfusion Matrix:")
print(conf_mat)

# Determine accuracy of the clustering
accuracy = accuracy_score(data['label'], data['kmeans_cluster'])
print(f"\nAccuracy: {accuracy:.2f}")

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
[[1964   31]
 [  83 1920]]

Accuracy: 0.97
```


Confusion Matrix
True Label
Predicted Label