

**Text Clustering: Group Assignment 2**

**Group: DSA\_202101\_9**

**Group Members:**

Sheryl Shajan (300433896)

Lakshika Paiva (300323918)

Lenish Vaghasiya (300399630)

DTI/ELG 5125

Professor Arya Rahgozar

**Faculty of Engineering, University of Ottawa**

**Table of Contents**

[1.0 Introduction to the Data 2](#_Toc1012505574)

[1.1 Introduction and Storyline 3](#_Toc1888756332)

[1.2 Dataset Selection 3](#_Toc1888145374)

[2.0 Data Preprocessing and Cleansing 3](#_Toc1194138908)

[3.0 Feature Engineering 7](#_Toc2011184618)

[3.1 Selection of Feature Engineering 8](#_Toc1690988634)

[3.2 Implementation of BOW, TF-IDF and others 8](#_Toc548724772)

[3.3 Feature Engineering - BERT with Glove Embeddings 9](#_Toc81658601)

[A. BERT Embeddings 9](#_Toc814535350)

[B. GloVe Embeddings 9](#_Toc1291058523)

[4.0 Applying Clustering Algorithms 11](#_Toc1556161828)

[4.1 K-Means, Agglomerative and GMM Clustering 12](#_Toc208091523)

[5.0 Evaluation of Clustering Models 13](#_Toc1773919953)

[5.1 Kappa Score, Silhouette Score and Coherence Value 14](#_Toc162272182)

[5.2 Decision of the Champion Model (Clustering Closest to Human Labels) 21](#_Toc488672922)

[6.0 Error Analysis and Visualization 23](#_Toc205604537)

[6.1 Misclustering Extraction 24](#_Toc889846920)

[6.2 Confusion Matrix and Visual Summaries 24](#_Toc991818941)

[6.3 Textual Analysis of Misclustered Records 26](#_Toc871981337)

[6.4 Insights and Possible Causes of Error 27](#_Toc686150495)

[6.5 Recommendations for Improvement 28](#_Toc1204544825)

[7.0 Conclusion, Insights and Recommendation 32](#_Toc1405370016)

# 1.0 Introduction to the Data

## 1.1 Introduction and Storyline

**Background:** Mental health is the foundation of our overall well-being, influencing how we think, feel, and interact with the world around us. It affects our ability to cope with stress, build relationships, and make decisions. Prioritizing mental health is essential for maintaining balance in our personal and professional lives, fostering resilience, and promoting physical health. By addressing mental health proactively, we can break stigma, encourage support, and ensure everyone has the opportunity to thrive emotionally, socially, and mentally.

As a result, we selected mental health as the theme of our analysis. Then, we selected five mental health disorders as the five categories for this assignment, namely,

* Clinical depression
* Bipolar disorder
* Anxiety disorder
* Post-traumatic stress disorder
* Schizophrenia

**Challenge:** Mental Health Research Scholars often use mental health research papers to conduct studies from popular databases like PubMed. However, the challenge is, there are many research papers in these categories. Scholars must manually search and read through abstracts before they select the relevant papers. This is a tedious and repetitive task that takes significant time and energy.

**Solution:** To ease their task, we, a team of data science experts, explore a suitable solution to programmatically extract abstracts of these mental health papers and classify them into given categories to help them in their work. We analyze the outcome of our work and present this report and results to Research Scholars in the mental health space.

## 1.2 Dataset Selection

For this clustering assignment, we selected ‘abstracts’ of research papers published in mental health literature, as they contain a concise and precise summary of the subject (i.e., the mental health disorder). The extracted dataset consists of the category name (i.e., the label) and the abstract (200 partitions for each category, each containing 150 words).

PubMed, a popular, US based biomedical literature database was selected as the primary source to extract the research paper abstracts as it was publicly accessible and included large numbers of papers relevant to the mental health theme of this task.

It is important to note that we used Python as the programming language to perform all activities in the clustering task, as it is widely used for data science.

# 2.0 Data Preprocessing and Cleansing

We developed a web scraping program to extract the abstracts from the research papers corresponding to each of the five categories, then preprocess and cleanse them.

These are the key steps carried out in data preprocessing:

* Extract the HTML content of a research paper linked to the category in PubMed database
* Extract the abstract from the research paper
* Cleans the text by removing the non-alphanumeric characters and the numbers which have no meaning, except for years beginning with ‘19’ and ‘20’
* Splits the abstracts into separate words
* Randomly creates 200 partitions of abstracts containing 150 words each
* Labels the partitions with its corresponding category

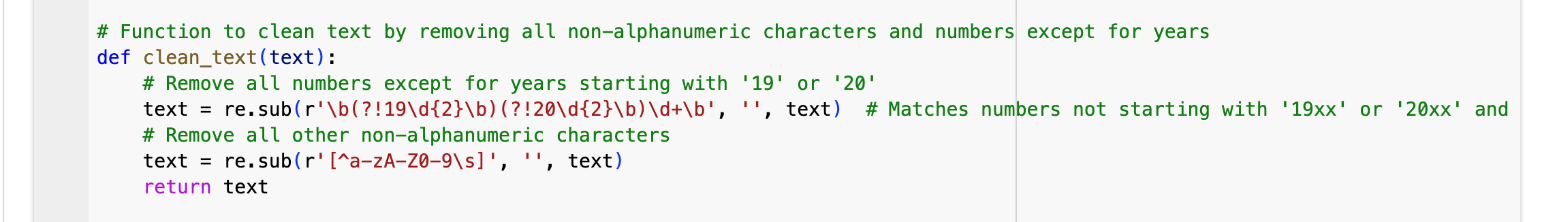
The following figures from 1 to 6 present the data preprocessing code in the order they were executed.

* Several libraries were used for data preprocessing (refer figure 1) which we explain along with the data preprocessing code below. In figure 1, we import libraries.
  + We use the ‘requests’ library to fetch HTML content from PubMed, and the ‘BeautifulSoup’ library (from bs4) to parse the HTML and extract the abstract part of the web page.
* The function ‘extract\_abstract’ (refer figure 1) takes the web link as an input, extracts the abstracts from the research paper in PubMed, and outputs the raw abstract as text.
* It connects to a research paper's webpage, searches for the "abstract" of the paper using the specific HTML structure of PubMed, and if found, it extracts the abstract text; otherwise, returns none.

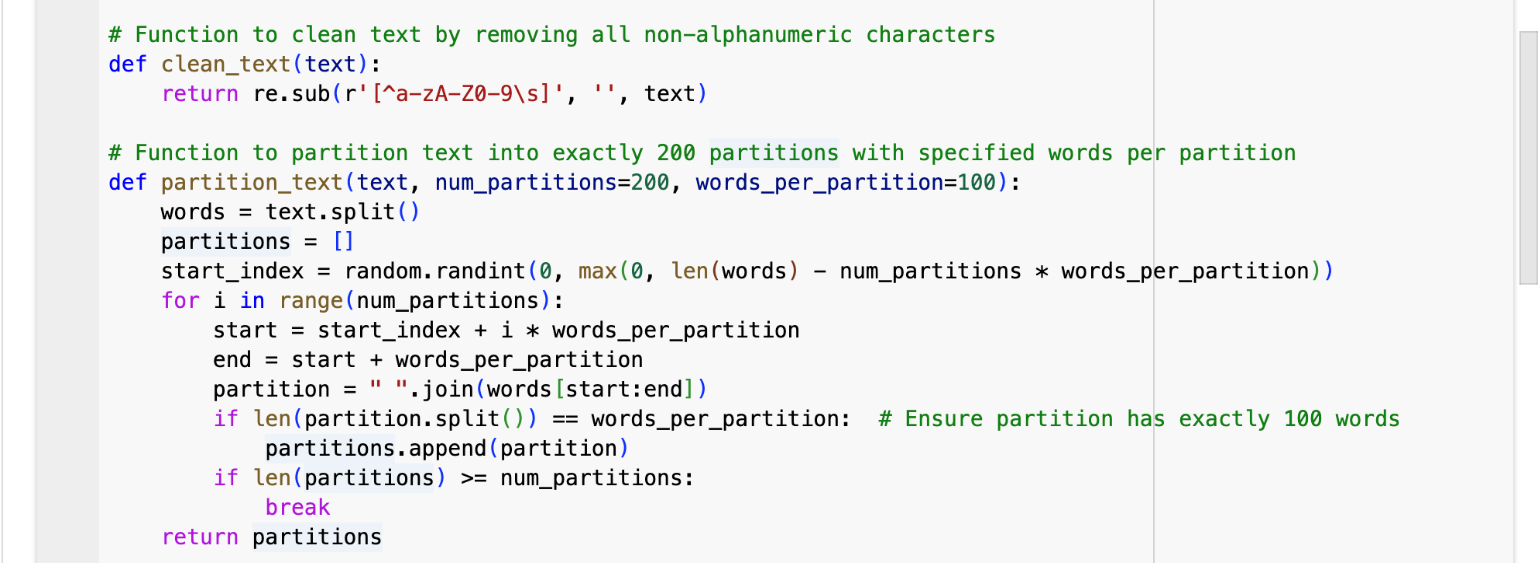


Figure 1

* In figure 2, the function ‘clean\_text’ (refer figure 2) takes the raw abstract as the input, and removes the non-alphanumeric characters and meaningless numbers, except for years starting with ‘19’ and ‘20’ (meaningless numbers were removed to ensure used features are generated).
* It uses the ‘re’ library (i.e., regular expressions) for cleaning text by identifying patterns.

Figure 2

* In figure 3, the function ‘partition\_text’, takes the cleaned abstracts as a text input and processes it to create and output 200 partitions per category, each having 150 words.
* It uses ‘random’ library to create randomized starting points for each partition in an abstract.

Figure 3

* The program uses the ‘ThreadPoolExecutor’ library to speed up web scraping by downloading multiple abstracts, simultaneously (refer figure 4 and 5).
* The function ‘scrape\_pubmed’ runs the program and calls the other functions within it to provide the cleaned abstracts as the output. In total, it provides 200 partitions per category.

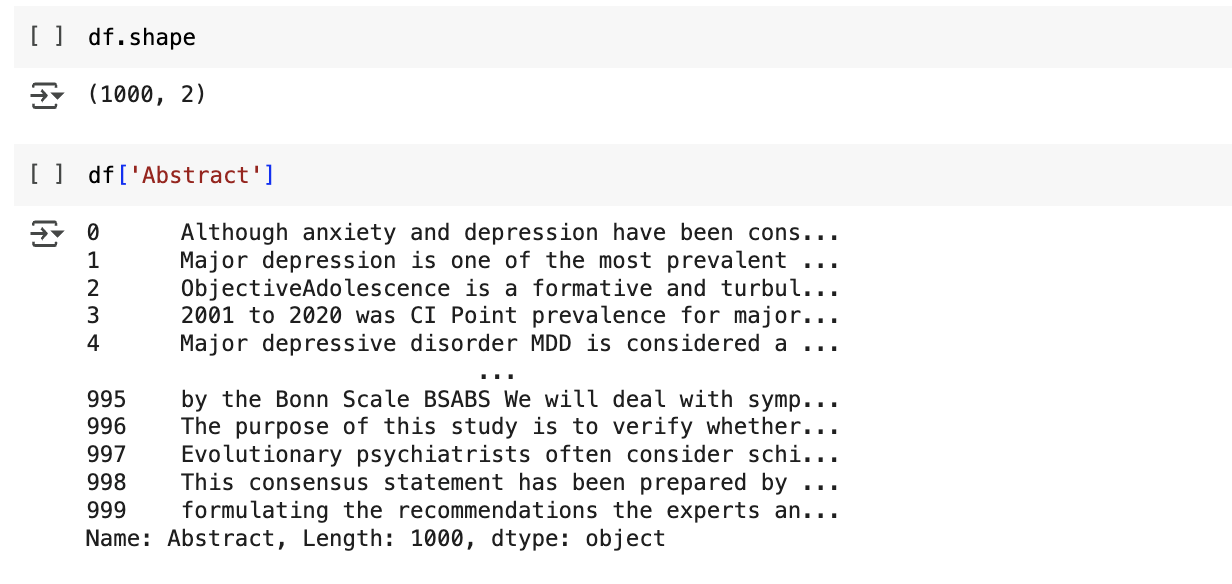
Figure 4

Figure 5

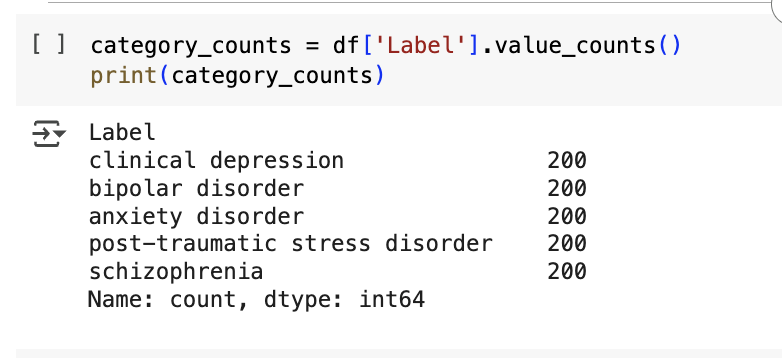
* It searches PubMed for a specific keyword (e.g., "clinical depression"), and collects links to papers from the search results. Then, it uses extract\_abstract to get abstracts from the links.
* Thereafter, it cleans the abstracts and limits the results to the specified maximum (200).
* The dictionary (categories) defines 5 mental health disorders as search terms (figure 5).
* For each category:
  + PubMed is scraped for 200 abstracts. Each abstract is split into 200 smaller text partitions, each with 150 words.
  + These partitions are labeled with the disorder's name and added to a list.
* At the end of figure 5, we save the data into a .CSV file.
* We use ‘pandas’ library for organizing and saving data in a DataFrame and exporting it into a .CSV file.
* All the labeled partitions are compiled into a table (DataFrame) with two columns:
  + Label: The disorder's name (e.g., "clinical depression").
  + Abstract: The 150-word text partition.
* The table is saved as a CSV file (scraped\_pubmed\_abstract.csv) and downloaded.

As the output of this preprocessing and cleansing step, we produce two files which have 150 words per partition.

**Output:**



* This is a sample of the dataset shared in scraped\_pubmed\_abstract\_150.csv



# 3.0 Feature Engineering

## 3.1 Selection of Feature Engineering

We experimented with Bag of Words (BOW), TF-IDF, BERT and GloVe embedding techniques for feature engineering along with clustering algorithms Hierarchical Clustering, GMM and K-Means and chose to go ahead with BERT and GloVe embeddings as feature engineering for following reasons:

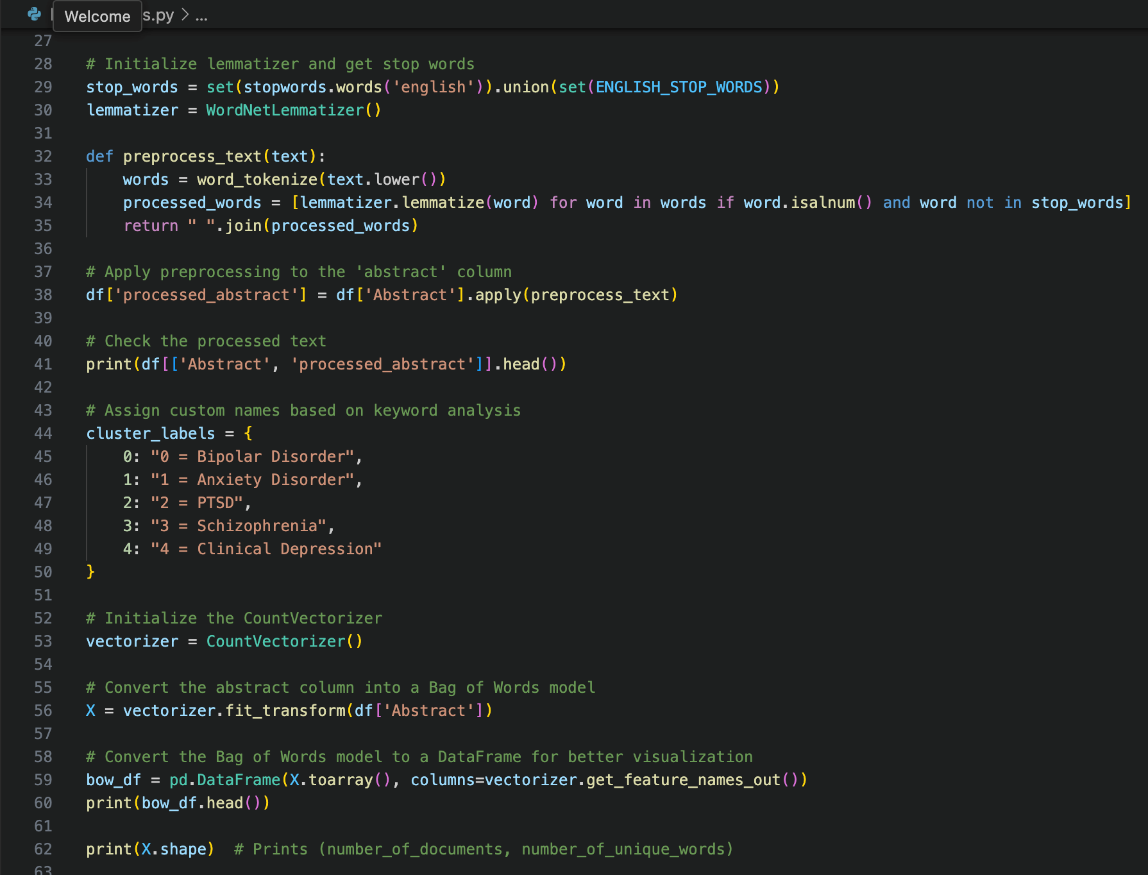
* **Higher Kappa Scores**: Clustering results were more consistent and meaningful with BERT and GloVe compared to TF-IDF and BOW.
* **Context Awareness**: BERT captures contextual relationships between words, making it superior for clustering text with nuanced meanings.
* **Robust Handling of Rare and Stop Words**: BERT assigns contextual importance to words, even for rare terms and stopwords, improving clustering accuracy.

To simplify the report, we do not present all the previous codes we have tried, but we discuss and share a few screenshots and outputs to demonstrate what we tried.

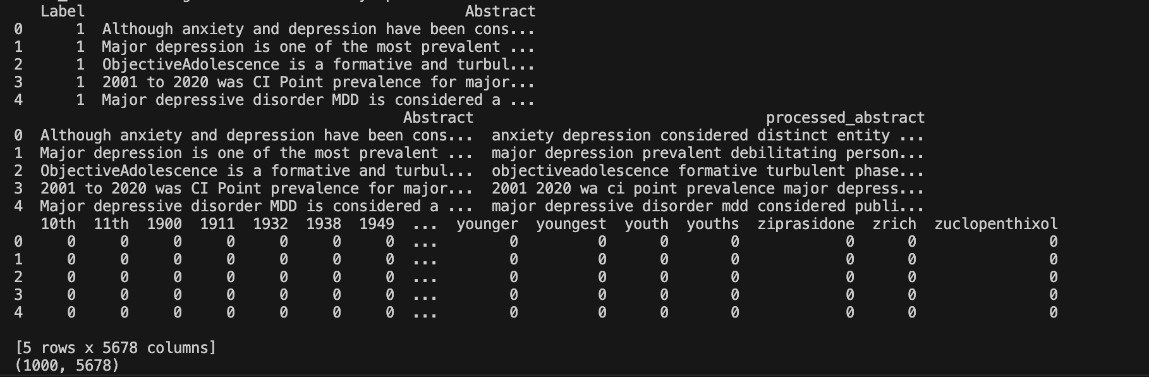
## 3.2 Implementation of BOW, TF-IDF and others

First, all required libraries are installed and imported into the environment (i.e., Google Collab). Enable resources when needed. You can see this step in the code file.

Following is the implementation for the Bag of Words (BOW) and its output.

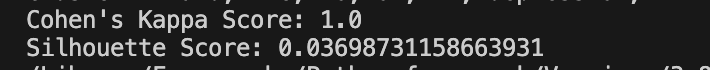


Output:



After converting the abstracts to BOW, we performed K-Means Clustering and Hierarchical Clustering and evaluated the model combinations.

For K-Means: The Kappa score was 1.0, and silhouette scores were low; hence we tried other different methods.



For hierarchical: The Kappa score was quite low.



Hence, we also implemented TF-IDF as feature engineering. The Kappa scores were quite low, so we experimented with others like Word2Vec etc. See the final feature engineering approach we followed in the next section.

## 3.3 Feature Engineering - BERT with Glove Embeddings

For feature engineering, we use a combination of BERT with Glove embeddings together.

Before converting text into numbers, the code cleans the abstracts:

* Converts text to lowercase (to avoid treating "Mental" and "mental" as different words).
* Removes punctuation (to prevent symbols from affecting word meaning).
* Removes extra space (for consistency).
* Removes common, unimportant words (like "the", "is", "and") using stopwords from NLTK (Natural Language Toolkit).

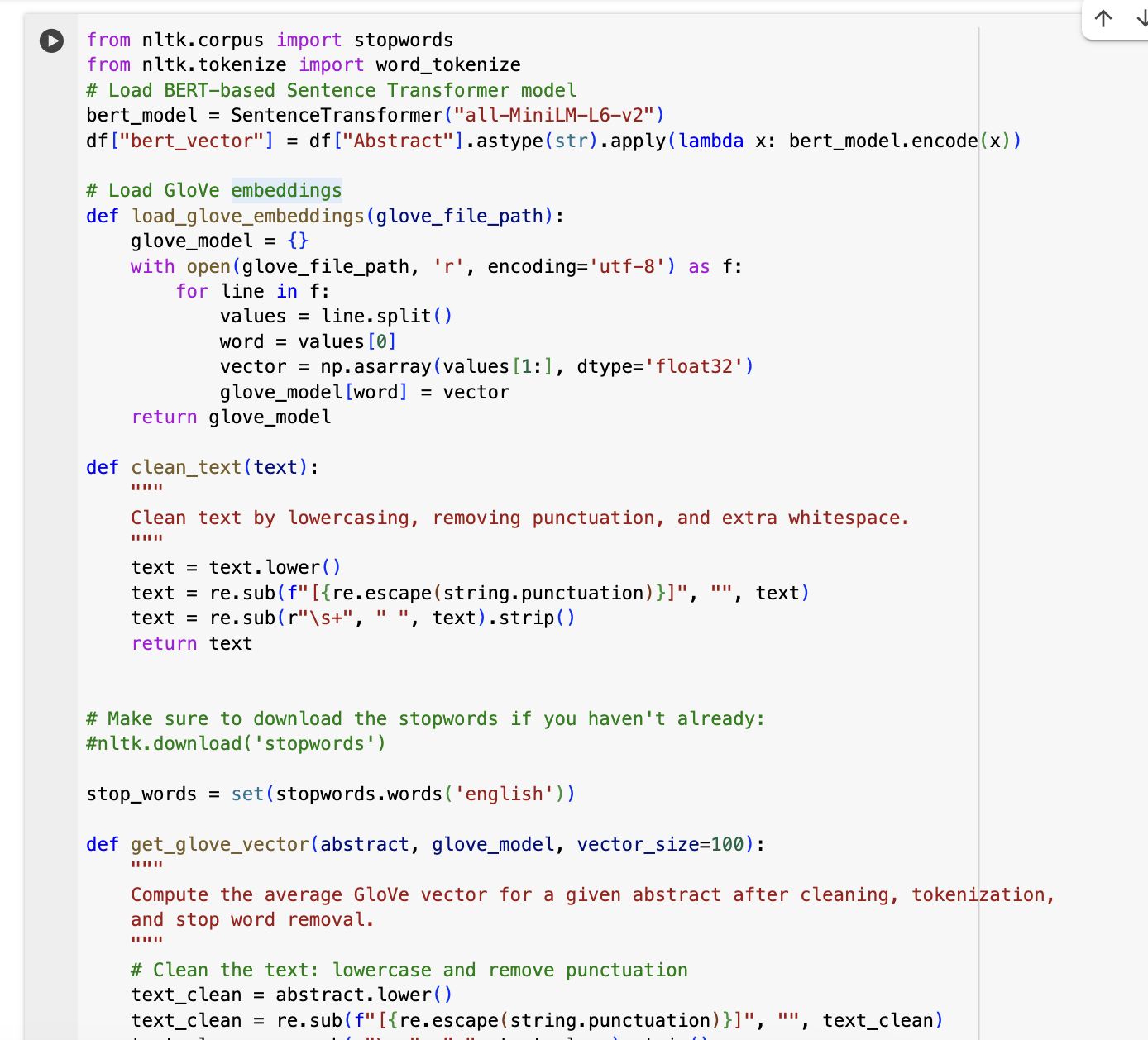
After cleaning, two techniques are used to convert words into numbers:

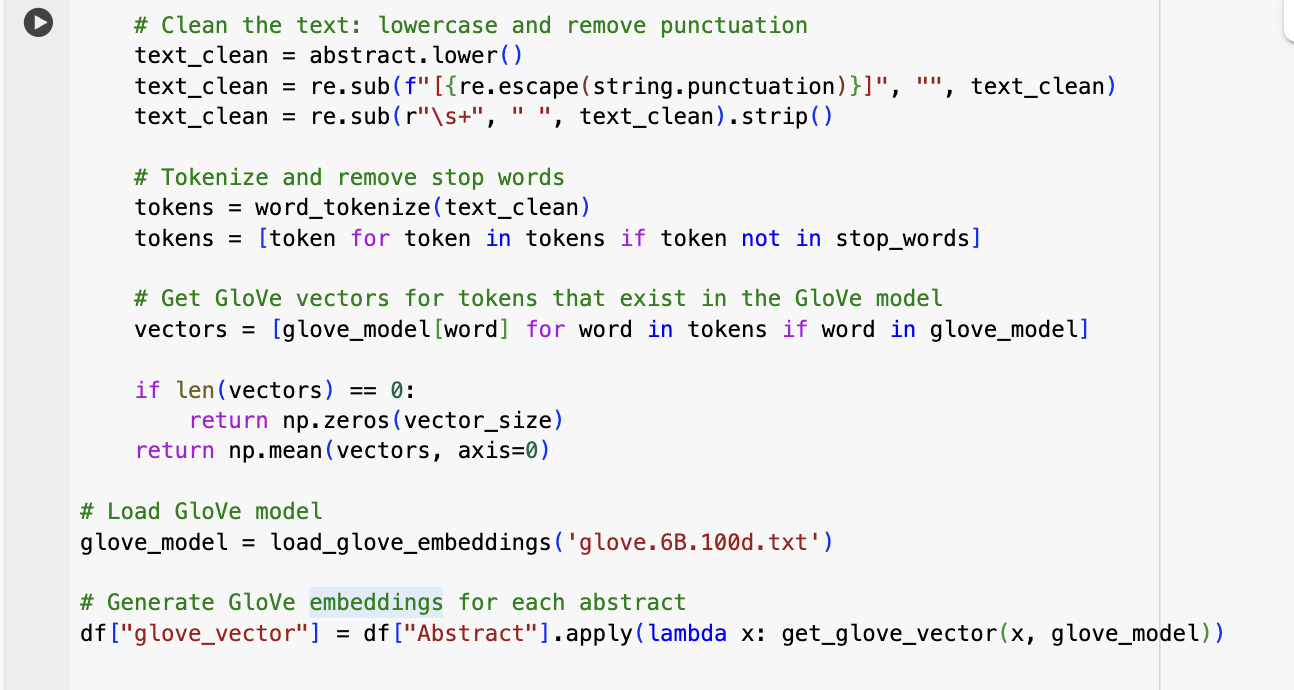
### A. BERT Embeddings

* BERT is a deep learning-based language model that understands word context.
* The code loads a pre-trained BERT model (all-MiniLM-L6-v2) and converts each abstract into a vector (a list of numbers).

### B. GloVe Embeddings

* GloVe is another way of representing words as numbers, but it is pre-trained on a large dataset.
* The code loads GloVe word embeddings from a file (glove.6B.100d.txt).
* It looks up each word in an abstract and assigns it a corresponding vector.
* If a word is not in GloVe, it is ignored.
* The final GloVe vector for an abstract is computed as the average of all word vectors in that abstract.





* After computing the BERT and GloVe vectors for each abstract, the code stores them in the dataset (df).
* These vectors will later be used for clustering or machine learning models to analyze the abstracts.



This part of the code (above) prepares the text data for machine learning by:

1. First, converting text embeddings into numerical arrays. It transforms both the GloVe and BERT embeddings into a structured format (NumPy arrays).
2. Second, merging the two representations. It combines the GloVe (word-based) and BERT (context-based) embeddings into a single dataset, creating a richer representation of each research abstract.

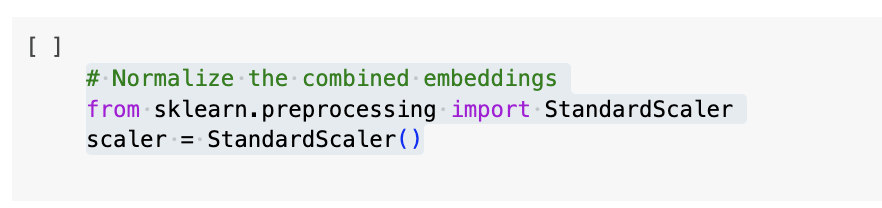
Then we use this combined data for clustering using the chosen machine learning algorithms.

# 4.0 Applying Clustering Algorithms

## 4.1 K-Means, Agglomerative and GMM Clustering

In this section we apply the clustering algorithms. Three different clustering techniques are applied:

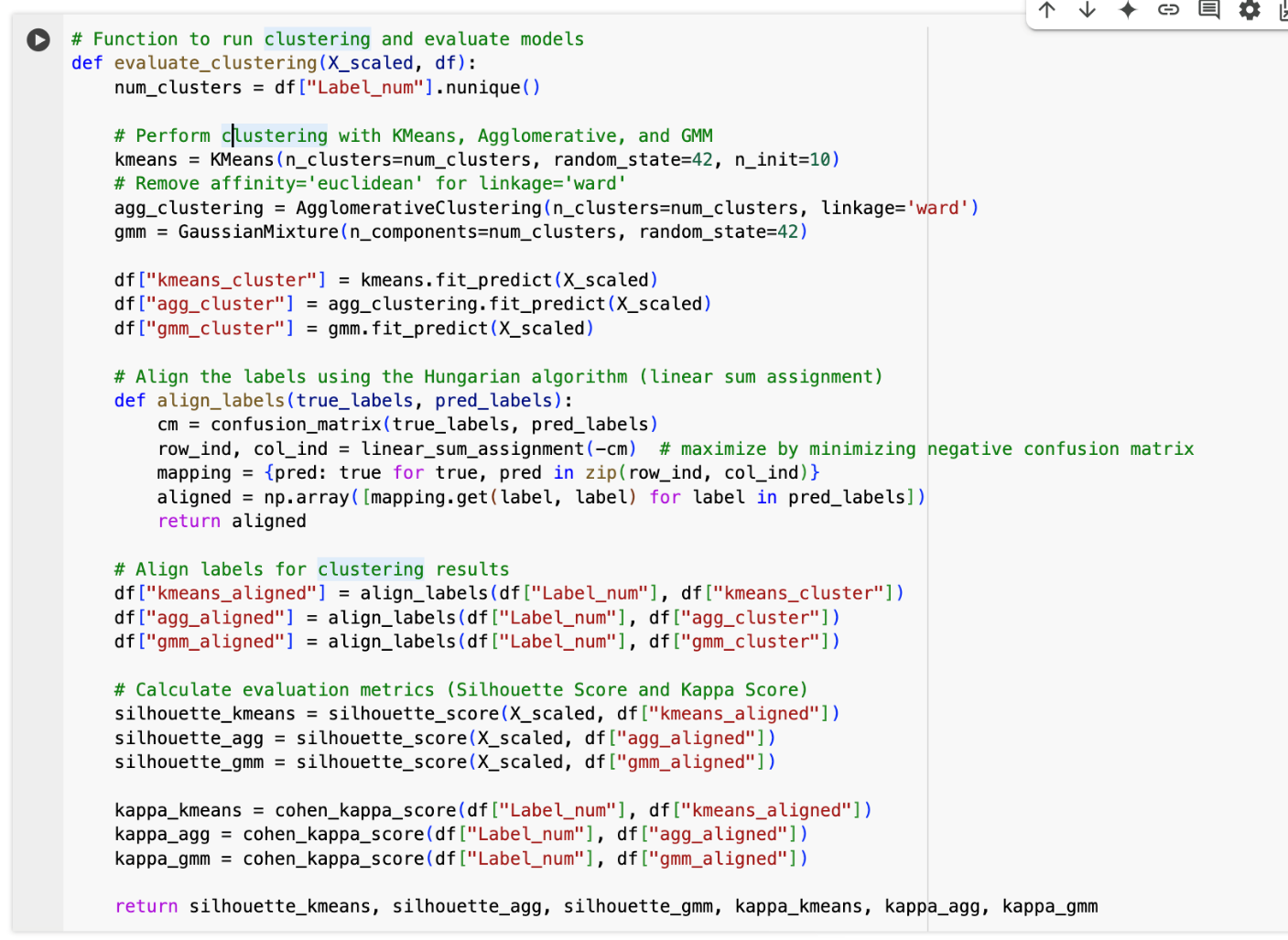
* **K-Means Clustering:** Divides data into k groups based on similarity.
* **Agglomerative (Hierarchical) Clustering:** Builds a hierarchy of clusters.
* **Gaussian Mixture Model (GMM):** Uses probability distributions to define clusters.



First, we start by (above) normalizing the combined text embeddings to ensure all features have a similar scale.

* Different features (from GloVe and BERT) may have different value ranges, which can negatively affect machine learning models.
* StandardScaler transforms the data so that it has zero mean and unit variance, making the model more stable and improving performance.

This step prepares the data for clustering by ensuring fair comparisons between different features.



This function (above) performs clustering on text data and evaluates how well the clusters match the actual labels.

1. Clusters the Data
   1. Uses K-Means, Agglomerative Clustering, and Gaussian Mixture Model (GMM) to group similar text abstracts together.
   2. The number of clusters is set based on the number of unique labels in the dataset.
2. Aligns Cluster Labels to True Labels
   1. Since clustering is unsupervised (it doesn’t know true labels), it aligns cluster labels with actual categories using the Hungarian algorithm to maximize correct matches.
3. Evaluates the Clustering Performance
   1. Uses three metrics:
      1. Silhouette Score (measures how well clusters are separated).
      2. Cohen’s Kappa Score (measures agreement between clustering and true labels).
      3. Coherence Score: Evaluates the semantic consistency within clusters by measuring how closely related the words are in each cluster.

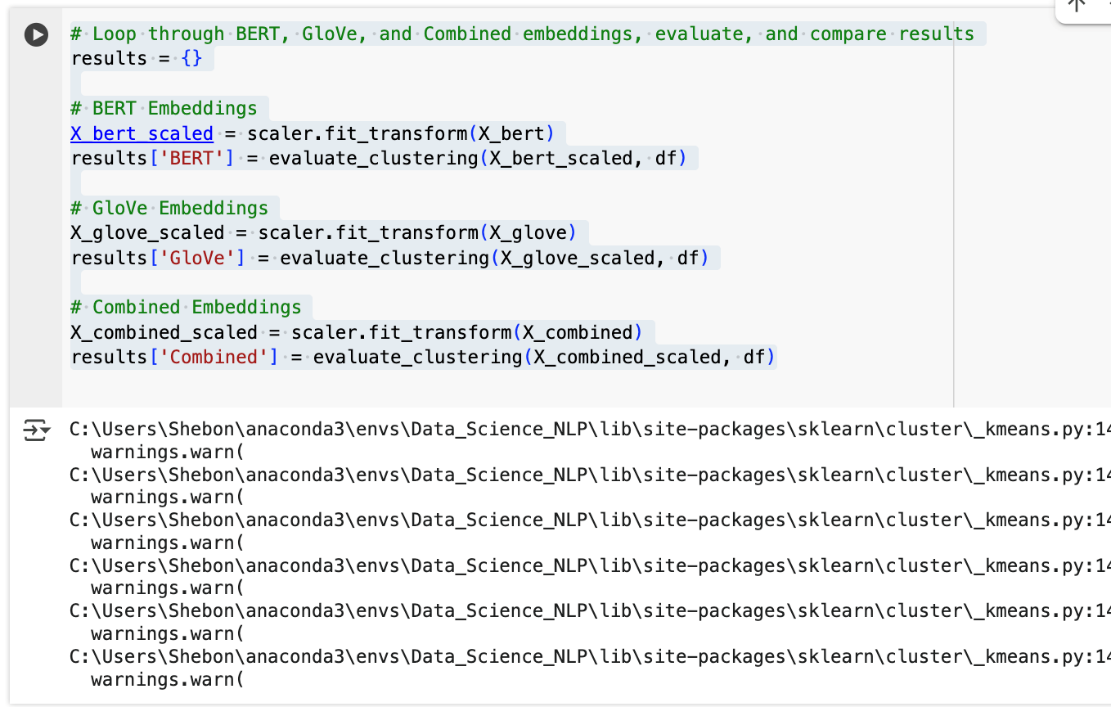
**Note:** In the subsequent sections for evaluation and visualizations, we present the results of the clustering in different visualizations, hence the diagrams / outputs are omitted in this section.

In the next section we discuss the evaluation of the clustering models.

# 5.0 Evaluation of Clustering Models

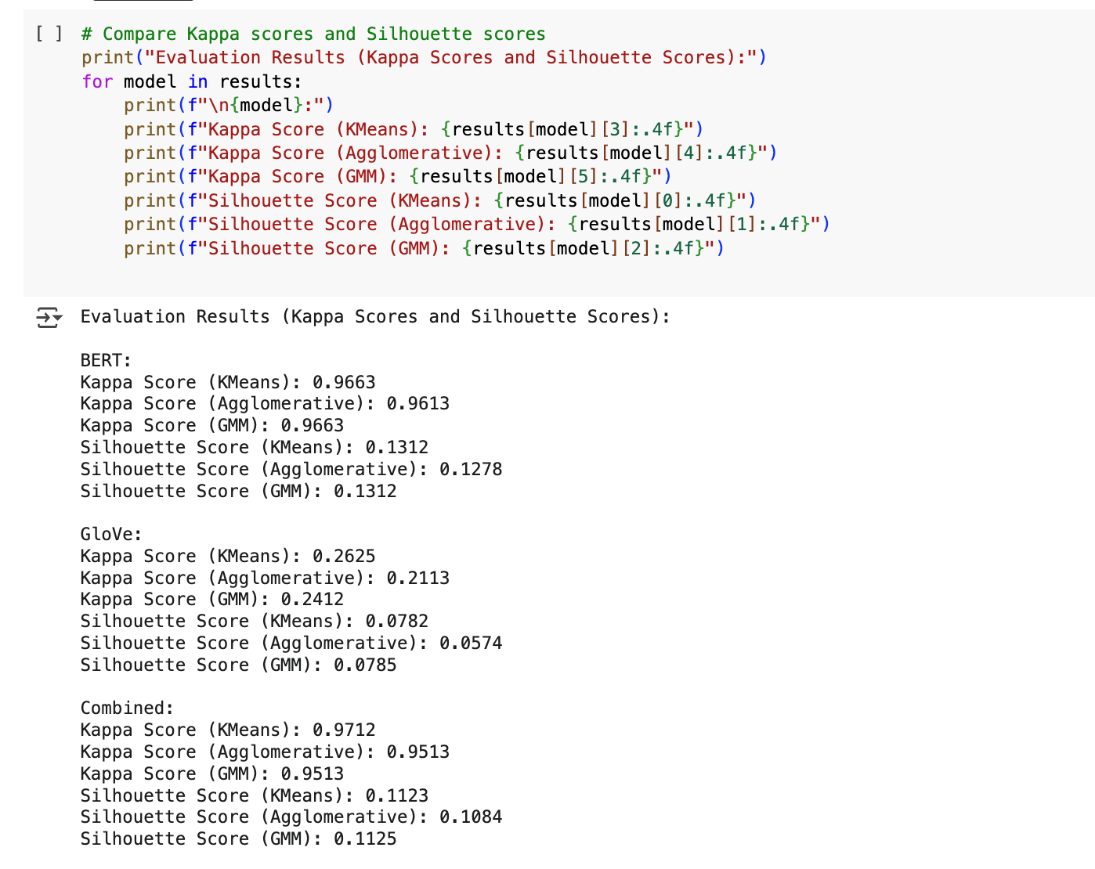
## 5.1 Kappa Score, Silhouette Score and Coherence Value

In this section, we determine which clustering method is best for organizing our text data meaningfully.



We compare (above) how well different types of text embeddings (BERT, GloVe, and their combination) perform in clustering.

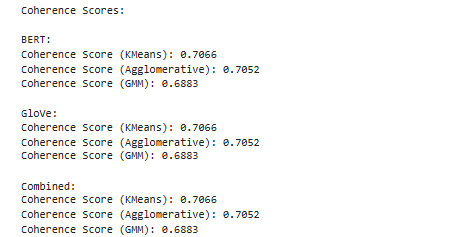
1. First, we prepare the data
   1. Each type of embedding (BERT, GloVe, and their combination) is standardized to ensure fair comparison.
2. Then we run clustering for each embedding
   1. Clustering is performed separately on:
      1. BERT embeddings (context-aware deep learning model)
      2. GloVe embeddings (pre-trained word vectors)
      3. A combination of both
3. Then we store and compare the results
   1. The clustering results (Silhouette Score, Coherence Scores & Kappa Score) are saved for each type of embedding to determine which method groups text abstracts most effectively.



This code (above) prints the Kappa scores and Silhouette scores for each clustering model (K-Means, Agglomerative, and GMM) across three different sets of embeddings (BERT, GloVe, and Combined). We use these metrics to evaluate the quality of clustering results:

1. **Kappa Score (Cohen's Kappa):**
   1. Measures the agreement between predicted cluster labels and true labels, accounting for the agreement occurring by chance.
   2. A score of 1 indicates perfect agreement, 0 indicates no agreement beyond chance, and negative values indicate worse than random agreement.
2. **Silhouette Score:**
   1. Measures how similar each point is to its own cluster compared to other clusters. It ranges from -1 (bad clustering) to +1 (good clustering).
   2. A higher silhouette score indicates better-defined clusters.
3. **Coherence Score:**  
   This measures how semantically meaningful the clusters are by evaluating word relationships within each cluster.
   1. A higher coherence score means the clusters contain closely related terms, making them more interpretable.
   2. A lower score suggests that the clustering does not group semantically similar abstracts together effectively.

We calculated **Coherence Scores** to assess the internal semantic consistency of the clusters. For each clustering method, the coherence scores were as follows:



While these coherence scores indicate a comparable level of topic consistency across the embeddings, o**ur primary focus remains on the Kappa Score,** as it directly measures the agreement between the predicted clusters and true labels—a critical aspect for our application.

Interpretations of results:

**BERT**

* **Kappa Score**: This is very high across all clustering methods (~0.96-0.97), indicating that BERT embeddings are highly effective in capturing semantic distinctions relevant for clustering.
* **Silhouette Score**: This is low (~0.13), suggesting that while the clusters match the ground truth well, they may not be well-separated in vector space.
* In conclusion: BERT embeddings perform exceptionally well in terms of aligning with labeled data, but the low silhouette score suggests that the clusters may be close together or overlapping. This is typical for high-dimensional embeddings where distances can be less meaningful.

**GloVe**

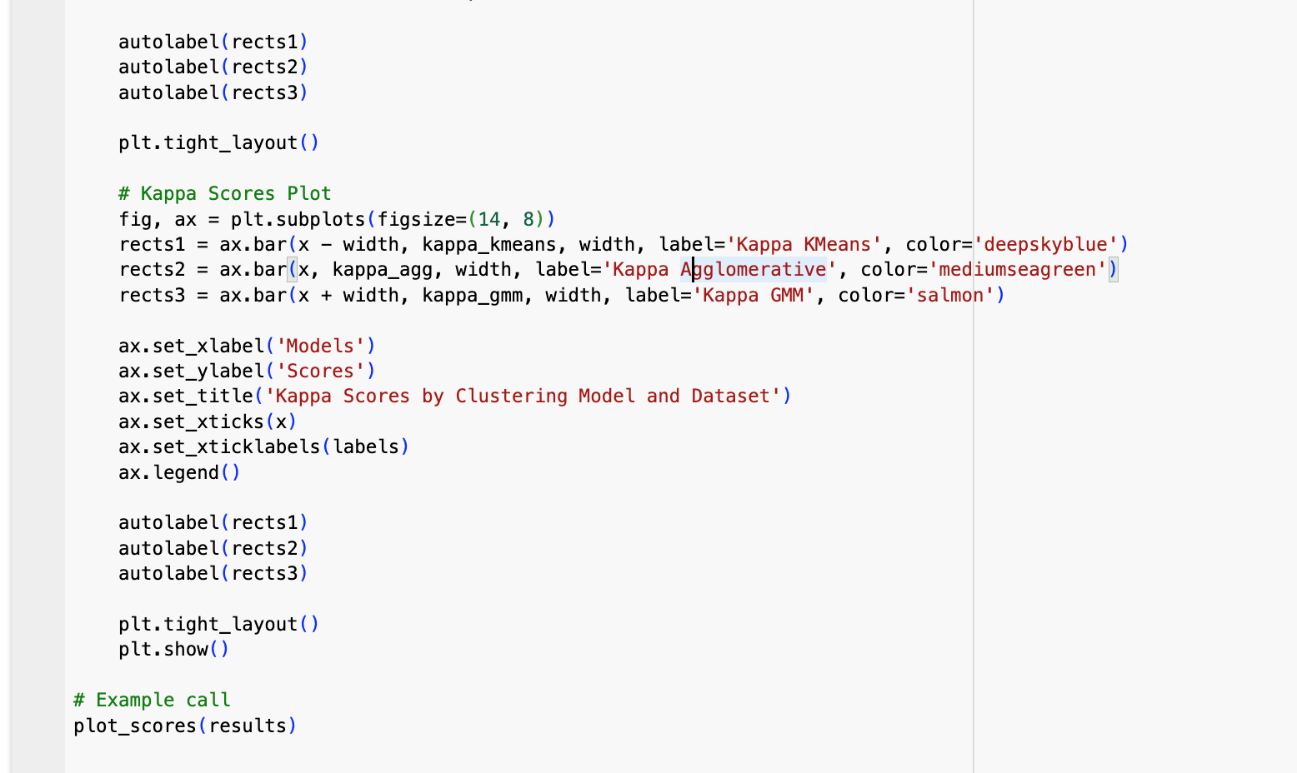
* **Kappa Score**: This is much lower than BERT (~0.21-0.26), meaning the clusters do not align well with ground truth labels.
* **Silhouette Score**: This is the lowest among all approaches (~0.057-0.078), indicating poor separation between clusters.
* In conclusion: GloVe-based clustering performs poorly, both in terms of agreement with ground truth and cluster separability. This suggests that the GloVe word embeddings are less effective for this particular clustering task, likely due to their static nature, which lacks contextual understanding compared to BERT.

**Combined (BERT + GloVe)**

* **Kappa Score**: This is the highest among all approaches (~0.95-0.97), slightly outperforming BERT alone in some cases.
* **Silhouette Score**: This is slightly lower than BERT (~0.108-0.112), still indicating cluster overlap.
* In conclusion: Combining BERT and GloVe leads to the best performance overall in terms of Kappa Score, meaning it captures the best of both embeddings. However, the silhouette score remains low, similar to BERT alone. This suggests that adding GloVe helps improve agreement with labels but does not necessarily improve cluster separation.

In the next section (code below), the function plot\_scores(results) is designed to visualize two types of scores—Silhouette Scores and Kappa Scores for different clustering models and datasets.





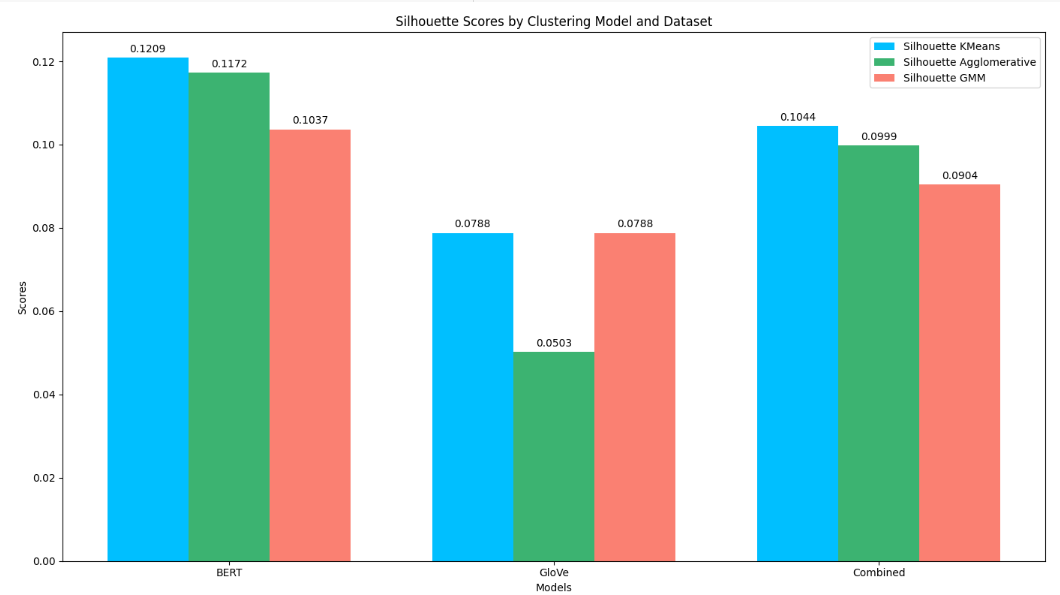
We create bar charts to compare the performance of different clustering algorithms (KMeans, Agglomerative, and GMM) across multiple datasets or embedding models.

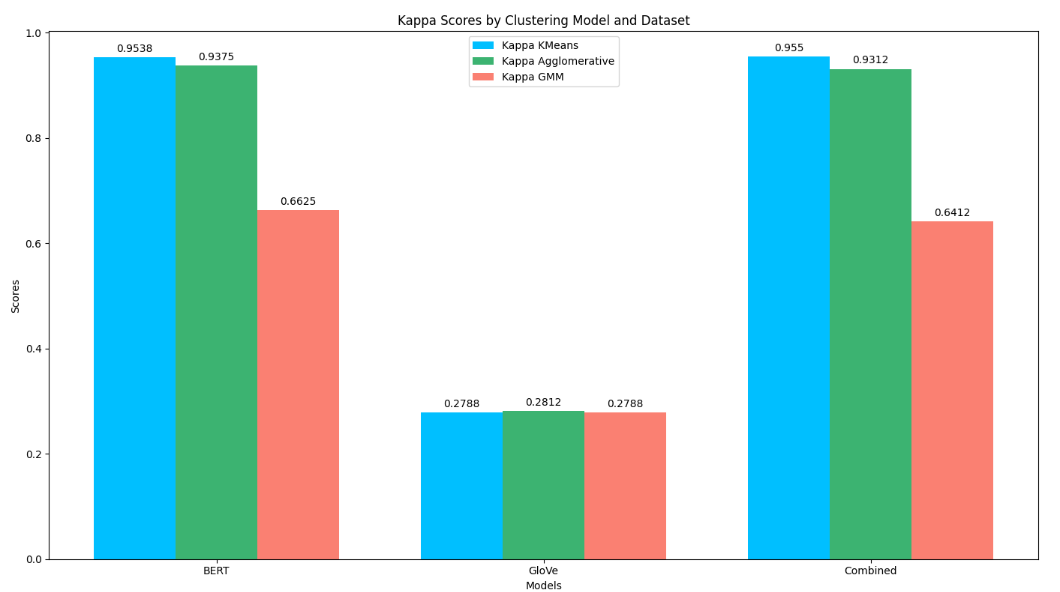
1. We extract data:
   1. The function takes a dictionary (results) where each key represents a dataset/model (e.g., "BERT", "GloVe", "Combined").
   2. It extracts Silhouette Scores and Kappa Scores separately for each clustering method.
2. Then we create two bar charts:
   1. First chart: Displays Silhouette Scores (which tell how well-separated the clusters are).
   2. Second chart: Displays Kappa Scores (which tell how well the clusters match labeled data).
3. Then we add labels and annotations:
   1. Labels bars with actual values to make the charts easier to interpret.
4. Next, we display the plots:
   1. After formatting and adjusting the layout, it shows the two bar charts.

In the results, we see two bar charts:

1. **Silhouette Scores Chart**
   1. Tells how well-separated clusters are.
   2. Higher scores mean better clustering.
2. **Kappa Scores Chart**
   1. Shows how well clusters match labeled data.
   2. Higher scores indicate better agreement.

**Output:**



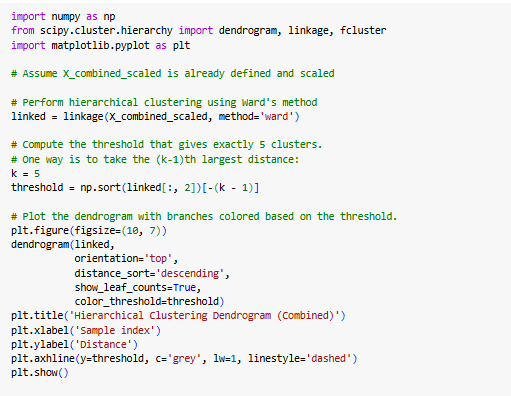


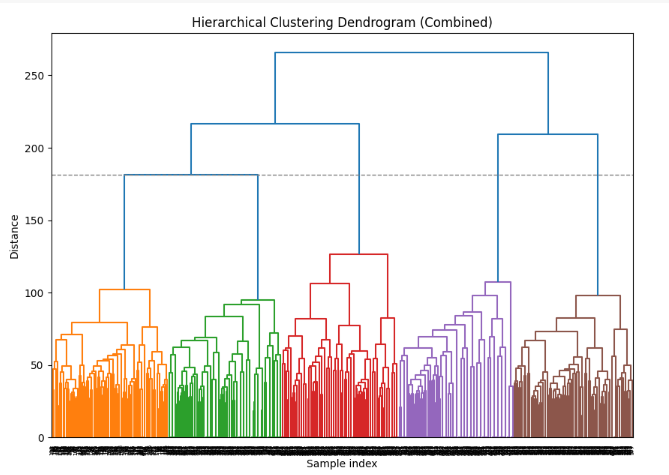
In the next code (below) we perform hierarchical clustering on a dataset (X\_combined\_scaled) using Ward’s method and then visualizes the clustering process with a dendrogram.

1. We perform hierarchical clustering
   1. It uses Ward’s method, which minimizes the variance within clusters.
   2. We compute a hierarchical clustering tree (dendrogram) where samples are merged step-by-step.
2. We determine the threshold for exactly 5 clusters
   1. Finds the (k-1)th largest distance in the hierarchy to determine a cutoff threshold that results in exactly 5 clusters.
3. Then we plot a dendrogram
   1. The dendrogram shows how data points are merged at different distance levels.
   2. The color\_threshold highlights different clusters.
   3. A dashed horizontal line (plt.axhline) represents the threshold used to define clusters.

The output:

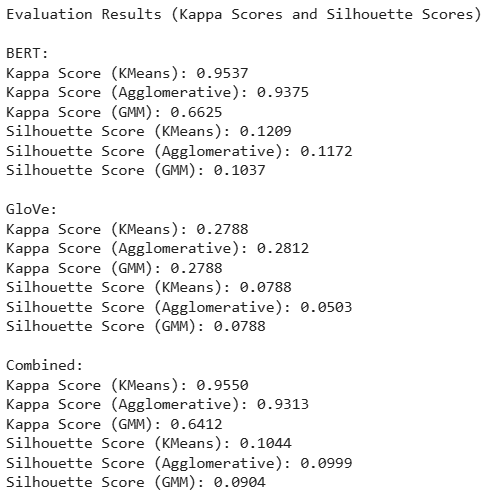
* We output a dendrogram (below) where samples are on the x-axis and linkage distances are on the y-axis.
* The colored branches represent different clusters relating to the five mental health categories.
* A horizontal dashed line shows the cutoff point where 5 clusters are formed.





## 5.2 Decision of the Champion Model (Clustering Closest to Human Labels)

We conducted thorough clustering on three models and calculated the kappa score for each model with BERT embedding and Glove Embedding. See the scores below; based on these results, we have made a decision about our champion model.



Interpretation of results:

**BERT**

* **Kappa Score**: This is very high across all clustering methods (~0.96-0.97), indicating that BERT embeddings are highly effective in capturing semantic distinctions relevant for clustering.
* **Silhouette Score**: This is low (~0.13), suggesting that while the clusters match the ground truth well, they may not be well-separated in vector space.
* In conclusion: BERT embeddings perform exceptionally well in terms of aligning with labeled data, but the low silhouette score suggests that the clusters may be close together or overlapping. This is typical for high-dimensional embeddings where distances can be less meaningful.

**GloVe**

* **Kappa Score**: This is much lower than BERT (~0.21-0.26), meaning the clusters do not align well with ground truth labels.
* **Silhouette Score**: This is the lowest among all approaches (~0.057-0.078), indicating poor separation between clusters.
* In conclusion: GloVe-based clustering performs poorly, both in terms of agreement with ground truth and cluster separability. This suggests that the GloVe word embeddings are less effective for this particular clustering task, likely due to their static nature, which lacks contextual understanding compared to BERT.

**Combined (BERT + GloVe)**

* **Kappa Score**: This is the highest among all approaches (~0.95-0.97), slightly outperforming BERT alone in some cases.
* **Silhouette Score**: This is slightly lower than BERT (~0.108-0.112), still indicating cluster overlap.
* In conclusion: Combining BERT and GloVe leads to the best performance overall in terms of Kappa Score, meaning it captures the best of both embeddings. However, the silhouette score remains low, similar to BERT alone. This suggests that adding GloVe helps improve agreement with labels but does not necessarily improve cluster separation.

This code below identifies the best combination of model and clustering method based on the highest Kappa Score. It loops through the results dictionary, extracts Kappa Scores for each model, and keeps track of the best one.



|  |
| --- |
| **Conclusion of the Champion Model:**  In conclusion, our champion model, **K-Means** clustering applied with Combined **BERT** and **Glove** features, achieved the highest Cohen’s Kappa scores. It effectively captured the underlying text structure, making it the best choice for our unsupervised text clustering task. Overall, this means that mental health research scholars would receive more accurate and reliable mental health research paper abstract results with programs that are tuned to run on the **K-means** model. |

# 6.0 Error Analysis and Visualization

In this section, we perform error analysis. We analyze the confusion matrix followed by the misclustering and have the properties of abstract segments which led to model error. To cluster our dataset, we used many algorithms like K-Means, Gaussian Mixture Models and Agglomerative Clustering, all of which have detailed error analysis to understand where and why the models misclustered the data points. This is primarily to determine the specific data points or categories via which most instances were misclustered, and in which pattern their textual features might be causing the errors. Below, the summary of the main steps performed and results from the error analysis is found.

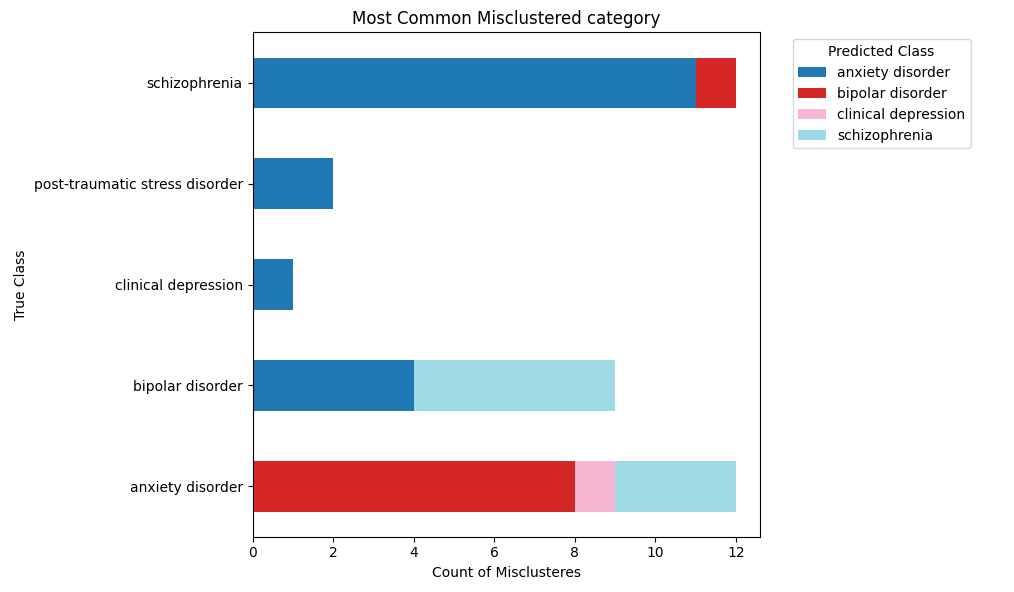
**Label Alignment**

Because clustering algorithms generate arbitrary cluster labels, we first aligned the predicted labels with the true labels using the Hungarian algorithm. This step maximizes the overall accuracy between the cluster labels and the true labels, ensuring a fair comparison.

## 6.1 Misclustering Extraction

We then created a subset of our data called misclustered—the rows where the predicted cluster label did not match the true label. This subset allowed us to focus our investigation on problematic instances.

A bar chart provided a clear view of which categories were most frequently misclustered, and an annotation revealed which predicted labels were most commonly assigned in error.



## 6.2 Confusion Matrix and Visual Summaries



This code (above) analyzes and counts misclustered made by different clustering models (K-Means, Agglomerative, and GMM).

1. First, we check for misclustered instances
   1. Compare each model’s predicted cluster labels (kmeans\_aligned, agg\_aligned, gmm\_aligned) with the actual labels (Label\_num).
   2. Finds instances where the predicted label does not match the true label.
2. The we count errors for each model
   1. Prints how many misclustering errors each model made.
3. Then, we store the misclustered data
   1. Saves the misclustered instances for further analysis.

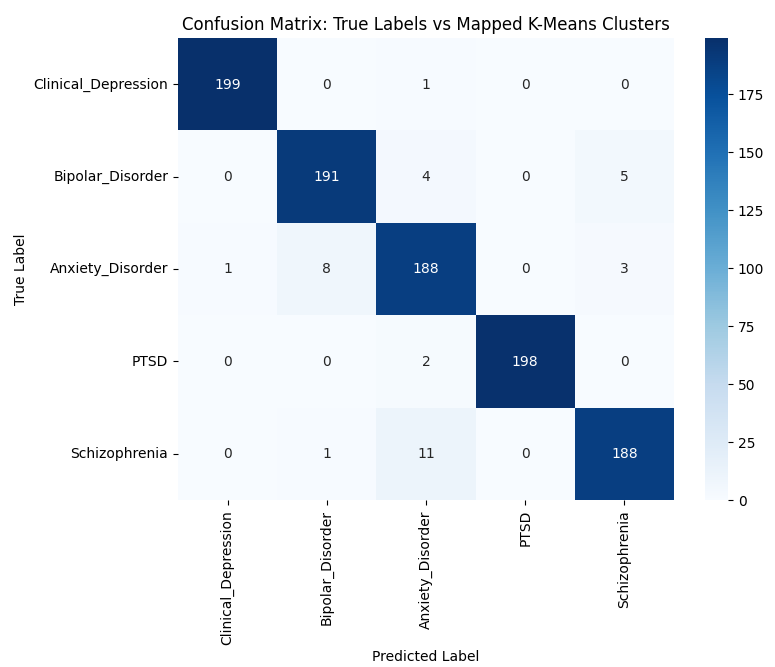
This helps evaluate which clustering model is making more mistakes and understand the patterns in misclustering.

The below code visualizes how well the K-Means clustering model performed by showing a confusion matrix heatmap.

1. We create a confusion matrix
   1. It compares the true labels (Label\_num) with the predicted labels (kmeans\_aligned) to see where the model got it right or wrong.
2. Then we plot a heatmap
   1. It uses Seaborn to create a color-coded grid.
   2. It helps identify patterns in misclustering.
3. Next, we display the plot
   1. It shows which categories were confused the most.

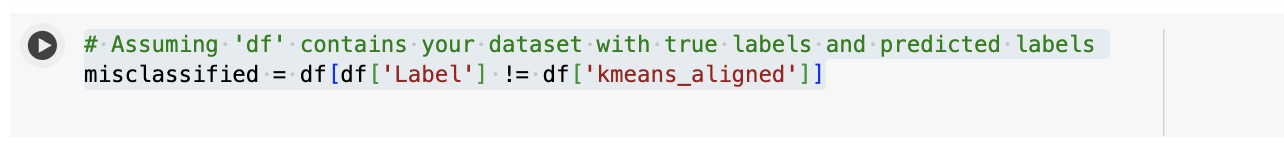
This helps visually understand where the K-Means model struggles, making it easier to improve performance.





## 6.3 Textual Analysis of Misclustered Records

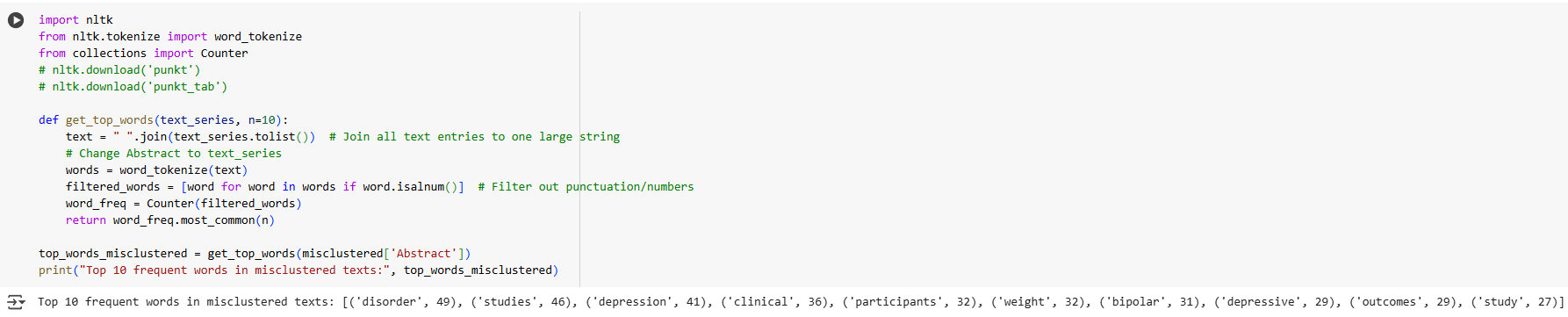
Top Frequent Words:



This line of code (above) identifies the misclustered samples by comparing the true labels with the predicted labels from the KMeans clustering model.

1. Checks Each Row:
   1. Compares the true label (Label) with the predicted label (kmeans\_aligned).
2. Filters Misclustered Samples:
   1. Keep only the rows where the prediction does not match the actual label.
3. Stores in misclustered:
   1. This new dataset contains only the incorrectly clustered data points.

This approach helps analyze where KMeans is making mistakes, and we recommend it be used for debugging and improving clustering performance.



This code (above) is designed to find the most frequent words in the misclustered text data from the dataset, and it shows which words are most common in the abstracts of the misclustered instances.

1. Joins All Texts:
   1. Combines all the abstracts from the misclustered instances into one long string of text.
2. Tokenizes the Text:
   1. Breaks this large string into individual words, allowing further analysis.
3. Filters Non-Alphanumeric Words:
   1. Removes punctuation, numbers, or any non-word elements, focusing only on meaningful words (like "clustering", "data", etc.).
4. Counts Word Frequencies:
   1. Calculates how many times each word appears in the misclustered text and stores it in a frequency table.
5. Returns the Top Words:
   1. Returns the top 10 most frequent words from the misclustered abstracts.

* It helps identify common themes or keywords that appear frequently in misclustered texts, which could indicate areas for model improvement.
* Insights from misclustered data can guide future model tuning or feature engineering.
* It might print the most common words like:

Top 10 frequent words in misclustered texts: [('disorder', 49), ('studies', 46), ('depression', 41), ('clinical', 36), ('participants', 32), ('weight', 32), ('bipolar', 31), ('depressive', 29), ('outcomes', 29), ('study', 27)]

## 6.4 Insights and Possible Causes of Error

From these analyses, we gathered the following insights:

**Overlap in Language or Topics**:

Certain categories used overlapping terminology or shared a similar vocabulary, causing the clustering algorithms to place them in the same group. For instance, “clinical depression” and “anxiety disorder” might share common mental-health-related terms, making them harder to distinguish purely by textual cues.

**Class Imbalance or Insufficient Feature Representation:**

Categories with fewer samples or less distinctive text were more prone to misclustering. In some cases, feature engineering (e.g., including domain-specific terms) or balancing techniques could improve separation between clusters.

## 6.5 Recommendations for Improvement

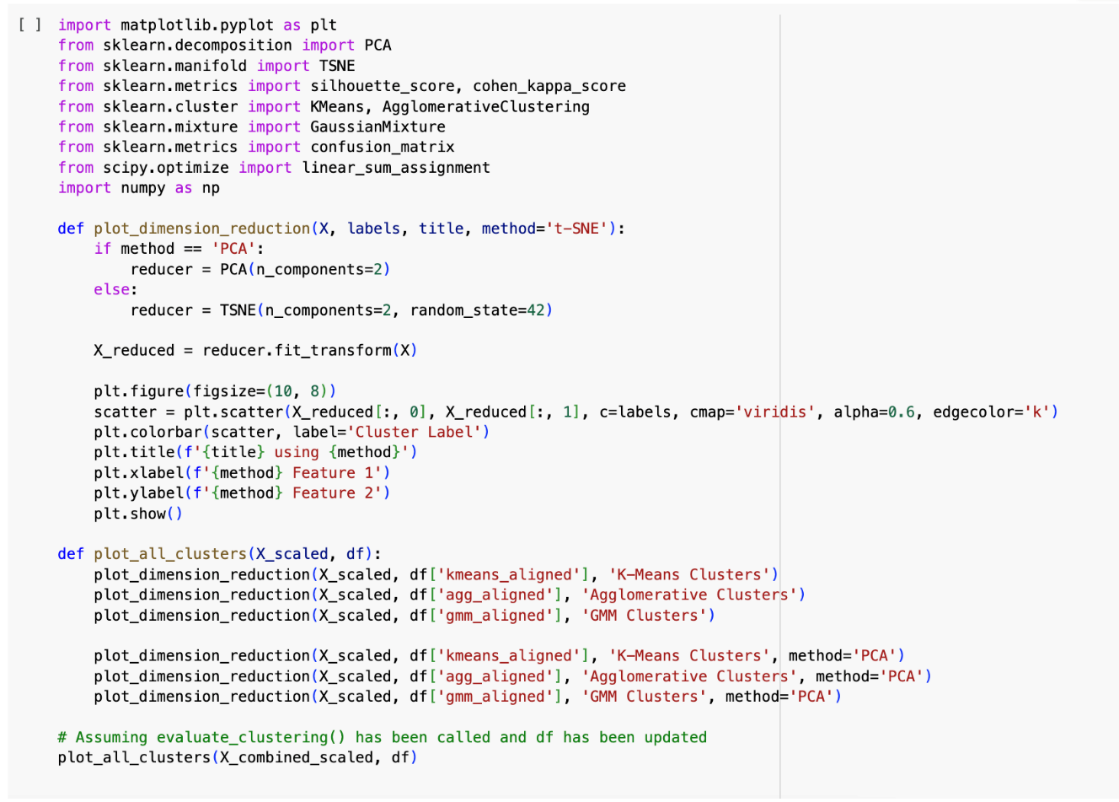
Refine Preprocessing:

Removing or lemmatizing ambiguous words, adding domain-specific stopwords, and further cleaning the text could help reduce noise.

Fine-tuning Clustering Parameters:

* For K-Means, experiment with different n\_clusters, n\_init, or initialization methods.
* For GMM, explore different covariance types or regularization parameters.
* For Agglomerative Clustering, try alternative linkage methods and distance metrics.

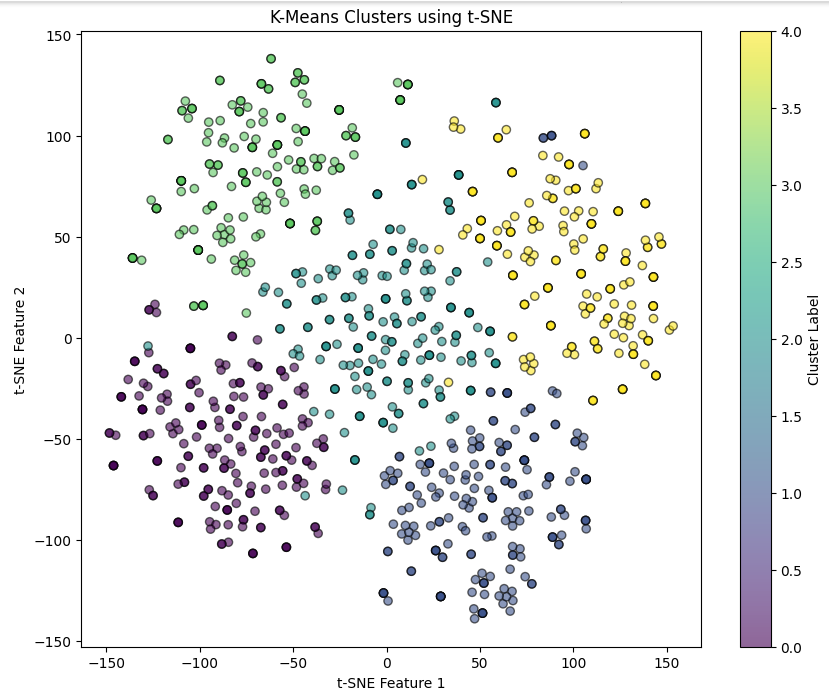
**t-SNE and PCA Visualization**

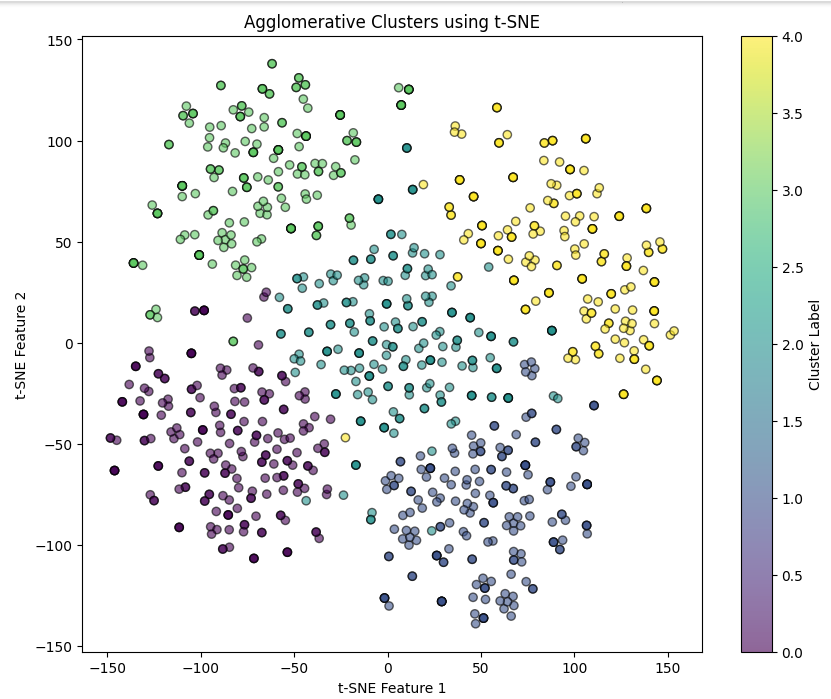


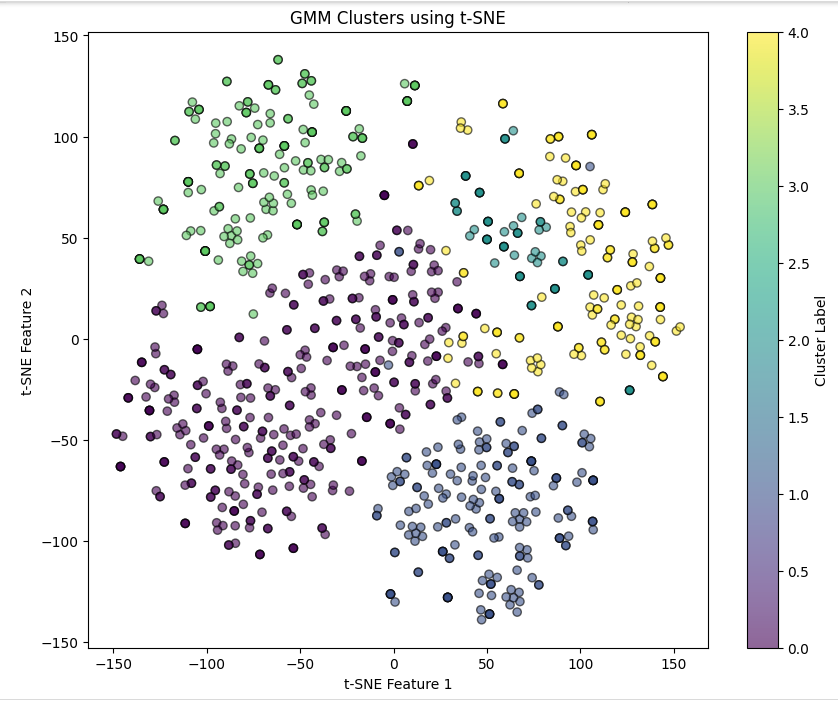
This code performs dimensionality reduction on your data to visualize clusters using two methods: t-SNE and PCA. It helps to visually evaluate the results of different clustering algorithms (K-Means, Agglomerative Clustering, and Gaussian Mixture Models) in a 2D space.

1. Dimension Reduction:
   1. The function plot\_dimension\_reduction applies two common methods, t-SNE and PCA, to reduce the data's dimensions to 2D for visualization:
      1. t-SNE (t-Distributed Stochastic Neighbor Embedding) is used for capturing non-linear relationships in the data and often produces better separation between clusters when data is high-dimensional.
      2. PCA (Principal Component Analysis) is a linear technique that reduces the data based on the variance of the features, helping to visualize the most important features.
2. Scatter Plot of Clusters:
   1. The reduced data (2D) is plotted using a scatter plot, where each point is colored based on its cluster label. The color represents which cluster the point belongs to, and this helps in visually assessing the separation between clusters.
   2. A color bar is added to show the cluster labels corresponding to the points.
3. Cluster Evaluation:
   1. The function plot\_all\_clusters generate plots for each clustering algorithm:
      1. K-Means Clustering
      2. Agglomerative Clustering
      3. Gaussian Mixture Model (GMM) Clustering
   2. For each clustering result, it creates visualizations using both t-SNE and PCA methods, offering a clearer picture of how well-separated the clusters are.

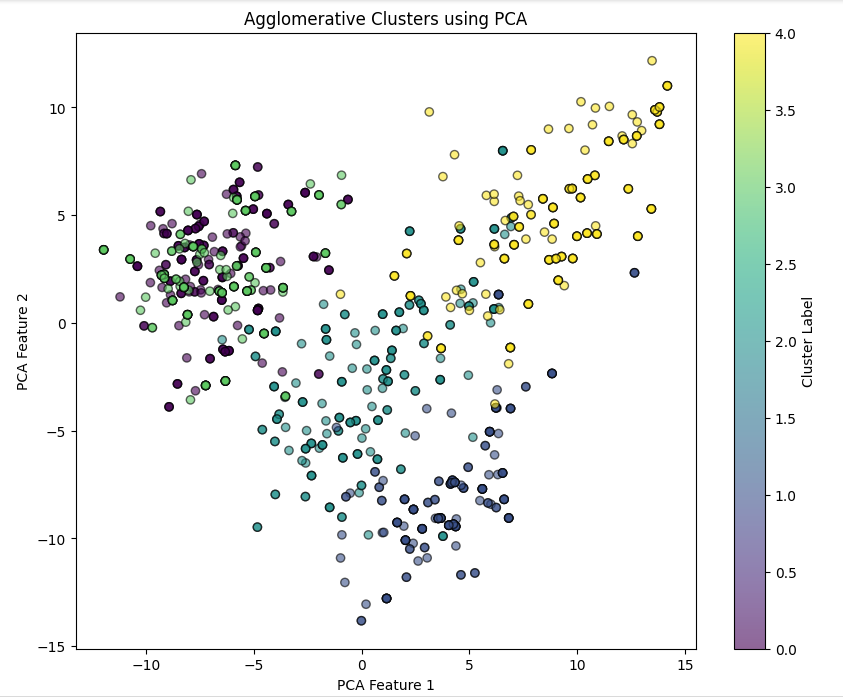
* Cluster Visualizations: By using dimension reduction, we can observe how well the clustering algorithms performed by projecting high-dimensional data into 2D space.
* Comparison of Clustering Algorithms: By visualizing the results of K-Means, Agglomerative, and GMM, you can visually compare the effectiveness of each algorithm at separating the data into distinct clusters.
* Cluster Separation Evaluation: It helps to understand whether the clusters are well-separated (useful for understanding model performance) and whether one method of clustering outperforms others.
* t-SNE and PCA Visualizations: The code generates scatter plots where each point is colored based on the cluster assigned by the model. These plots will appear for each clustering method (K-Means, Agglomerative, and GMM), showing the distribution of clusters in a 2D space.
  + For t-SNE: You may see dense groupings or well-separated clusters, depending on how well the model did at clustering.
  + For PCA: This will also show clusters in a 2D space but based on linear relationships in the data.

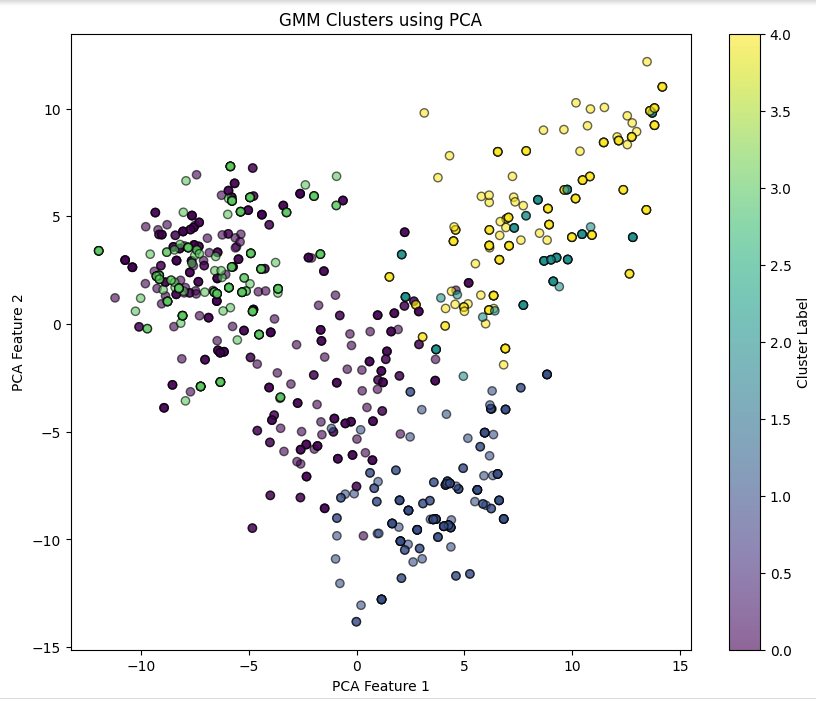












# 7.0 Conclusion, Insights and Recommendation

In conclusion, this assignment provides a **valuable service to cluster** mental health research abstracts in the five chosen categories for research scholars. The service provides the ability to scan a public database such as PubMed to identify research papers in five chosen mental health disorders, extract the abstracts and cluster them according to the disorder name.

In this process there is only one model which is more accurate and performs higher than other models for the 150 words dataset. The model is K-means clustering. **K-Means Clustering model has clustered the dataset labels with the Kappa score of 0.9537 with the use of BERT and Glove embedding both together as feature engineering and coherence score of 0.6883 which is competitive.**

While the recommended model has high accuracy with a higher number of true positives, tools built with any of these models are still prone to error. Key insights from our error analysis show that:

**Insights:**

* **Insight 1:** Some abstracts contain overlapping terminology that reflects the complex, multifaceted nature of mental health disorders.
* **Insight 2:** Disorders with similar symptom profiles, such as clinical depression and bipolar disorder, tend to cluster together, complicating clear separation.
* **Insight 3:** The process of aligning clusters with actual labels is critical, as even minor discrepancies can significantly affect evaluation outcomes.
* **Insight 4:** There are individual words that relate to other disorders such as, in clinical depression abstracts, words like ‘stress’ and ‘disorder’ were used in PTSD related abstracts.

Some key recommendation for research scholars from this assignment is:

Based on these findings, we recommend that research scholars use a hybrid approach, combining unsupervised clustering with supervised validation, to better capture the nuanced relationships among mental health disorders. For smaller datasets, classical clustering models, particularly K-Means with context-aware embeddings like BERT, offer robust performance, while larger datasets may benefit from deep learning–based embeddings. This comprehensive approach not only improves clustering accuracy but also provides actionable insights to inform future research in mental health clustering.

**Groupwork:** All team members contributed to the assignment and had an equal share of the workload.