AY 2024/25 SEMESTER 1

SC4020 DATA ANALYTICS & MINING

Project 1 - Report

By Group 9

Name	Matric No.	Contributions
Wong Zi Lun	U2240220D	Researching and finding code for methods Methods section of report of report
Javen Ong Jing Hian	U2240610E	Researched on Silhouette method Running the notebooks Experiment Section of report
Thejesvi Ramesh	U2240554H	Abstract, Introduction and Conclusion of report
Florian Goering	U2220036A	Extensively looked up for suitable datasets for the experiment

Table of Contents

Abstract

This report evaluates the performance of three clustering algorithms — K-means, K-means++, and DBSCAN — on three diverse datasets: Weather Type Classification, Comprehensive Diabetes Clinical, and Enron Fraud Email. These datasets span different sizes and feature types, including numeric, categorical, and binary variables. The algorithms are compared in terms of clustering accuracy, scalability, and computational efficiency, revealing strengths and limitations in handling complex real-world data. The analysis highlights the best-suited algorithm for each type of dataset, providing insights into optimal use cases for clustering in various domains.

Introduction

Clustering is an unsupervised learning technique widely used in data analytics and mining to group data points based on their features without prior knowledge of their labels. It has applications in various fields, from weather pattern classification to healthcare diagnostics. Despite its broad utility, selecting the right clustering algorithm for a particular dataset can be challenging, as performance varies based on dataset size and composition, among other qualities.

This report aims to compare three widely-used clustering algorithms — K-means, K-means++ and DBSCAN — across three distinct datasets. The Weather Type Classification dataset provides a mix of numerical and categorical variables, as is typical in environmental data, while the Comprehensive Diabetes Clinical dataset presents a combination of medical and demographic variables. Finally, the Enron Fraud Email dataset, with over 400,000 records, presents a challenge for clustering large-scale, primarily categorical data.

By evaluating the performance of these algorithms, we aim to identify their strengths, limitations, and optimal use cases, and to gain a deeper insight on how to make informed decisions when choosing clustering techniques for various real-world problems.

List of Datasets

Weather Type Classification	A synthetically generated dataset that mimics weather data. Size of dataset: 19,000 Variables ● Temperature (numeric)

	● Humidity (numeric) ● Wind Speed (numeric) ● Precipitation (%) (numeric) ● Cloud Cover (categorical) ● Atmospheric Pressure (numeric) ● UV Index (numeric) ● Season (categorical) ● Visibility (km) (numeric) ● Location (categorical) ● Weather Type (categorical) [1] Found from: https://www.kaggle.com/datasets/nikhil7280/weather-type-classification
Comprehensive Diabetes Clinical	A dataset of the health and demographic data of individuals Size of dataset: 100,000 Variables ● Gender (categorical) ● Age (numeric) ● Location (categorical) ● Race (categorical) ● Hypertension (binary) ● Heart Disease (binary) ● Smoking History (categorical) ● BMI (numeric) ● HbA1c Level (numeric) ● Blood Glucose Level (numeric) ● Diabetes Status (binary) [2] Found from: https://www.kaggle.com/datasets/priyamchoksi/100000-diabetes-clinical-dataset
Enron Fraud Email	A dataset of emails of US company Enron, consisting of internal emails, marketing emails, spam and fraud attempts. Size of dataset: 447,417 Variables ● Folder-User (categorical) ● Folder-Name (categorical) ● Message-ID (text)

	● Date (datetime) ● From (categorical) ● To (categorical) ● Subject (categorical) ● Mime-Version (numeric) ● Content-Type (categorical) ● Content-Transfer-Encoding (categorical) ● X-From (categorical) ● X-To (categorical) ● X-cc (categorical) ● X-bcc (categorical) ● X-Folder (categorical) ● X-Origin (categorical) ● X-FileName (categorical) ● Body (text) ● Cc (categorical) ● Bcc (categorical) ● Time (numeric) ● Attendees (categorical) ● Re (categorical) ● Source (categorical) ● Mail-ID (categorical) ● POI-Present (binary) ● Suspicious-Folders (binary) ● Sender-Type (categorical) ● Unique-Mails-From-Sender (numeric) ● Low-Comm (binary) ● Contains-Reply-Forwards (binary) ● Label (categorical) [3] Found from: https://www.kaggle.com/datasets/advaithsrao/enron-fraud-email-dataset

Methods

K-means

K-means is an algorithm that allows us to sort a dataset into *k* different clusters based on each data point's distance from the center of each cluster.

It works by first randomly assigning *k* cluster centroids, and then assigning each data point to a cluster based on their distance from the centroid of each cluster. Then, new cluster centroids are assigned from the data points in each cluster. The process then repeats until the cluster centroids do not change, which usually means we have found a good cluster. [4]

Algorithm

```
1. Specify the number $k$ of clusters to assign.
2. Randomly initialize $k$ centroids.
3. repeat
4.         expectation: Assign each point to its closest centroid.
5.         maximization: Compute the new centroid (mean) of each cluster.
6. until The centroid positions do not change.
```

Parameters Settings

The value *k* is found using the Silhouette Coefficient, which measures the quality of clusters by evaluating how well each data point fits within its assigned cluster. [5]

Two factors are considered:

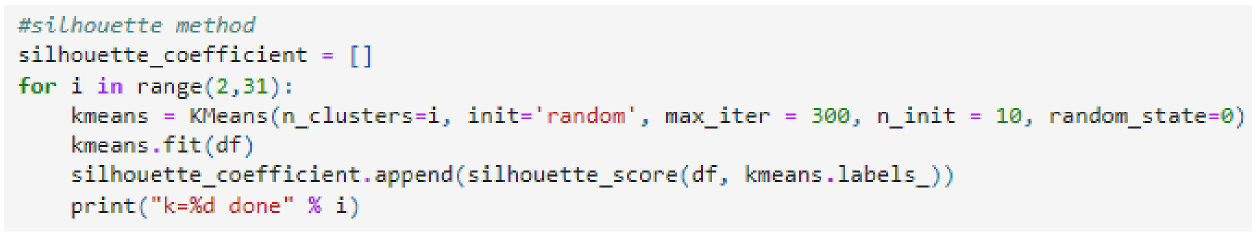
1. Cohesion, a: how close a point is to other points in its cluster.
2. Separation, b: how far the point is from points in the nearest neighboring cluster.

The coefficient is then calculated as follows:

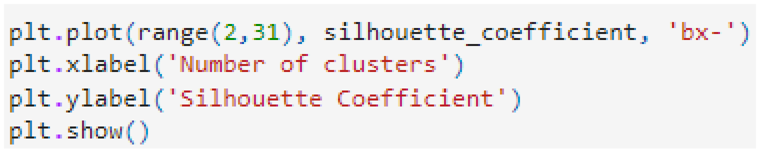
$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))}$$

The values range from -1, which indicates that it is misclustered, to 1, meaning it is well clustered.

Silhouette Method Code:

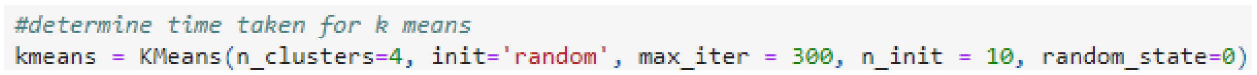


```
#silhouette method
silhouette_coefficient = []
for i in range(2,31):
    kmeans = KMeans(n_clusters=i, init='random', max_iter = 300, n_init = 10, random_state=0)
    kmeans.fit(df)
    silhouette_coefficient.append(silhouette_score(df, kmeans.labels_))
    print("k=%d done" % i)
```

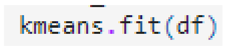


```
plt.plot(range(2,31), silhouette_coefficient, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.show()
```

K-Means Code:



```
#determine time taken for k means
kmeans = KMeans(n_clusters=4, init='random', max_iter = 300, n_init = 10, random_state=0)
```



```
kmeans.fit(df)
```

Strengths:

- **Simple**: K-Means is straightforward and easy to implement, allowing for easy use in grouping datasets
- **Efficient**: K-Means is one of the fastest clustering algorithms

Weaknesses

- **Requires an appropriate k value to be chosen**: Choosing a k value too high or too low will lead to a poor quality of clusters found
- **Sensitive to initialization of centroids**: The final clusters found by K-Means is heavily dependent on the initial cluster centroids, and poor initialization may lead to suboptimal clusters
- **Sensitive to outliers**: K-Means is sensitive to outliers and noise, and may mistaken them for a cluster
- **Poor performance on data with non-spherical shape**: K-Means has poor performance on datasets with clusters of varying shapes, sizes and densities

K-means++

K-means++ is an extension of the K-means algorithm, and similarly allows us to sort a dataset into *k* different clusters based on each data point's distance from the center of each cluster.

K-means++ differs from the standard K-means in its initialization of the *k* cluster centroids. First, randomly assign a data point to be a cluster centroid. Then, compute the shortest distance of each data point from the current cluster centroid(s), and assign the data points with the maximum shortest distance to be a new central centroid. The process then repeats until we have obtained *k* cluster centroids. Then the K-means algorithm proceeds as usual. [6]

Algorithm

```
1. Specify the number k of clusters to assign.
2. Initialize k centroids.
3. repeat
4.         expectation: Assign each point to its closest centroid.
5.         maximization: Compute the new centroid (mean) of each cluster.
6. until The centroid positions do not change.
```

Parameters Settings

The value of k is found by the exact same method as K-Means which is the Silhouette Method.

Silhouette Method Code:

```python
#hard to determine through elbow method so resort to silhouette method
silhouette_coefficient = []
for i in range(2,31):
    kmeanspp = KMeans(n_clusters=i, init='k-means++', max_iter = 300, n_init = 10, random_state=0)
    kmeanspp.fit(df)
    silhouette_coefficient.append(silhouette_score(df, kmeanspp.labels_))
    print("k=%d done" % i)
```

```python
plt.plot(range(2,31), silhouette_coefficient, 'bx-')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.show()
```

K-Means++ Code:

```python
#determine time taken for k means++
kmeanspp = KMeans(n_clusters=4, init='k-means++', max_iter = 300, n_init = 10, random_state=0)
```

```python
kmeanspp.fit(df)
```

Strengths

- Similar to K-Means Algorithm
- Has the added benefit of a smarter initialisation of cluster centroids compared to K-means which uses a random initialization of centroids, which could possibly lead to suboptimal results.

Weaknesses

- Similar to K-Means Algorithm, less the choosing of initial cluster centroids

Density-Based Spatial Clustering Of Applications With Noise (DBSCAN)

DBSCAN is an algorithm that identifies clusters as regions with a high density of data points, separated by regions of low density of data points.

In DBSCAN, clusters are considered a set of data points in regions of high density of other data points, and are found by taking a data point in a high density region, and then recursively finding all neighboring data points which are also in a high density region. For DBSCAN, 2 parameters, **eps** and **MinPts**, need to be defined. [7]

1. **eps**: the distance determining the neighborhood around a data point
2. **MinPts**: the minimum number of neighboring data points within the eps radius

Algorithm

```
1. Specify the parameters eps and MinPts.
2. Find all neighboring data points within the eps radius and more than MinPts
   neighbors
3. repeat
4.         Assign a cluster to a data point in a high density region not already
   assigned to a cluster.
5.         Recursively find all its neighboring data points and assign them to the
   same cluster
6. until There are no unvisited data points.
```

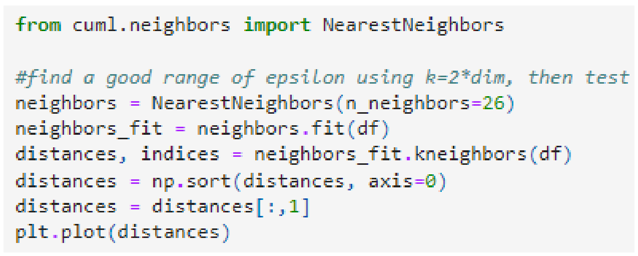
Parameters Settings

Finding eps and minPts[8][9]:

1. Set $k = 2 * dimensionality\ of\ the\ data.$
2. The k-nearest neighbors algorithm is applied to compute the distance to the k-th nearest neighbor for each point in the dataset.

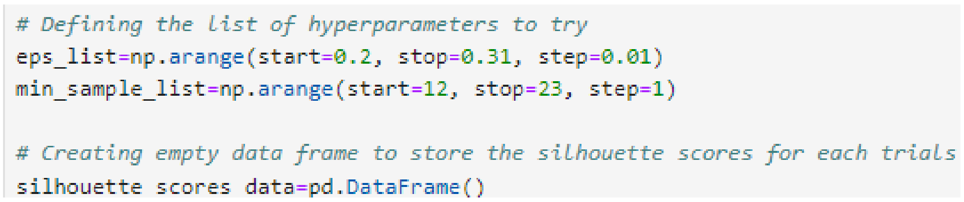
3. These distances are sorted and plotted. The elbow in the plot indicates a good choice for the range of eps to test, as it separates dense regions from noise.
4. Take the combination of the above range of eps and a range of minPts to get the combination with the highest silhouette score

Finding epsilon range:

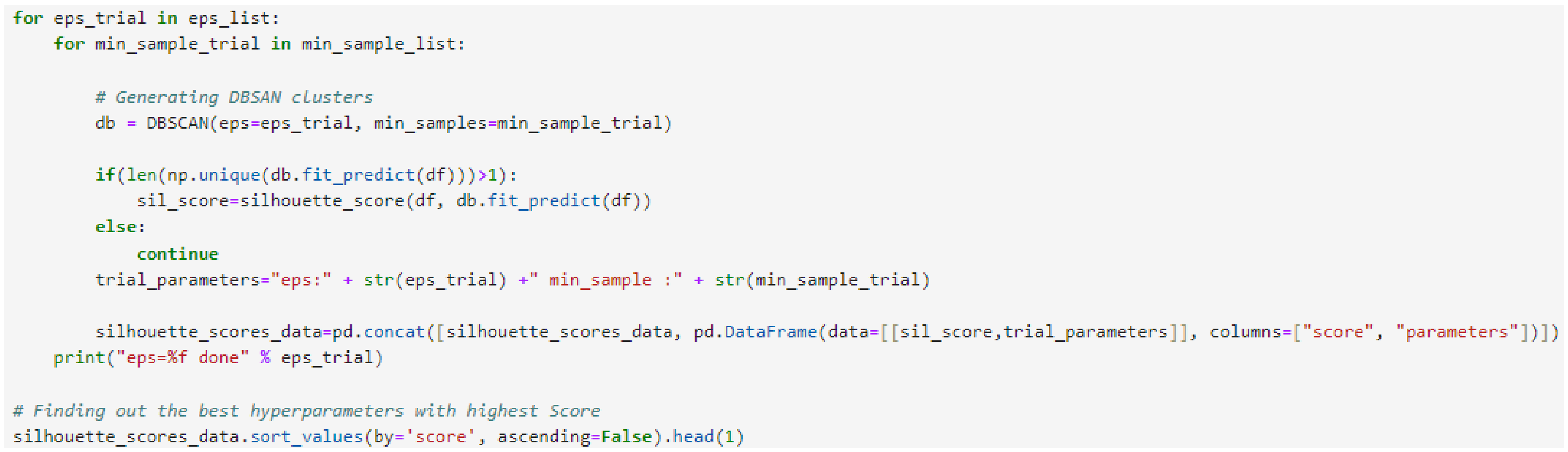


```python
from cuml.neighbors import NearestNeighbors

#find a good range of epsilon using k=2*dim, then test
neighbors = NearestNeighbors(n_neighbors=26)
neighbors_fit = neighbors.fit(df)
distances, indices = neighbors_fit.kneighbors(df)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

Silhouette Method:

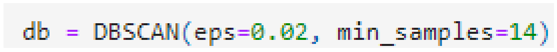


```python
# Defining the list of hyperparameters to try
eps_list=np.arange(start=0.2, stop=0.31, step=0.01)
min_sample_list=np.arange(start=12, stop=23, step=1)

# Creating empty data frame to store the silhouette scores for each trials
silhouette_scores_data=pd.DataFrame()
```

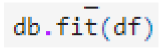


```python
for eps_trial in eps_list:
    for min_sample_trial in min_sample_list:

        # Generating DBSAN clusters
        db = DBSCAN(eps=eps_trial, min_samples=min_sample_trial)

        if(len(np.unique(db.fit_predict(df)))>1):
            sil_score=silhouette_score(df, db.fit_predict(df))
        else:
            continue
        trial_parameters="eps:" + str(eps_trial) +" min_sample :" + str(min_sample_trial)

        silhouette_scores_data=pd.concat([silhouette_scores_data, pd.DataFrame(data=[[sil_score,trial_parameters]], columns=["score", "parameters"])])
    print("eps=%f done" % eps_trial)

# Finding out the best hyperparameters with highest Score
silhouette_scores_data.sort_values(by='score', ascending=False).head(1)
```

DBSCAN Function:



```python
db = DBSCAN(eps=0.02, min_samples=14)
```



```python
db.fit(df)
```

Strengths

- **Can work with data of varying shapes**: DBSCAN can find clusters with varying shapes and densities
- **Outlier identification**: DBSCAN is able to identify outliers and noise, improving performance in datasets with a lot of noise
- **Does not require number of clusters to be specified**: Useful when the optimal number of clusters is unknown

Weaknesses

- **Less effective with clusters of varying densities**: DBSCAN may be unable to identify smaller clusters within a larger, sparser area
- **Less effective with high dimension data**: The distance between points become less meaningful in higher dimensions, preventing DBSCAN from effectively identifying clusters
- **Parameter Sensitivity**: DBSCAN's performance is highly reliant on its parameter settings, and poorly chosen parameters can lead to suboptimal clustering

Key factors that affect the performance

K-Means	**Initial cluster centroids** The clusters found using K-Means are determined by its initial choice of cluster centroids, and poor initialization can lead to suboptimal clustering. **Parameter setting (k value)** The quality of the clusters is determined by the choice of k, and can lead to underfitting or overfitting of clusters if a poor k value is used **Presence of outliers and noise in data** K-Means is sensitive to outliers and noise, and can skew the cluster centroids **Data scaling** The scaling of each attribute of the dataset used may disproportionately affect the choice of clusters, if they are not standardized **Dimensionality of data** Distances between data points become less meaningful with high-dimension data, which will influence the clusters found
K-Means++	Similar to K-Means, less the issue of random initial cluster centroids
DBSCAN	**Parameter setting (ε and MinPts)** The parameters of DBSCAN determine how it identifies clusters, and a poor choice of parameters can lead to suboptimal clustering. **Data density and shape** DBSCAN has poor performance when it comes to data with varying densities or has clusters in extremely close proximity, potentially leading to the misclassification of clusters **Dimensionality of data**

	Distances between data points become less meaningful with high-dimension data, which will influence the clusters found

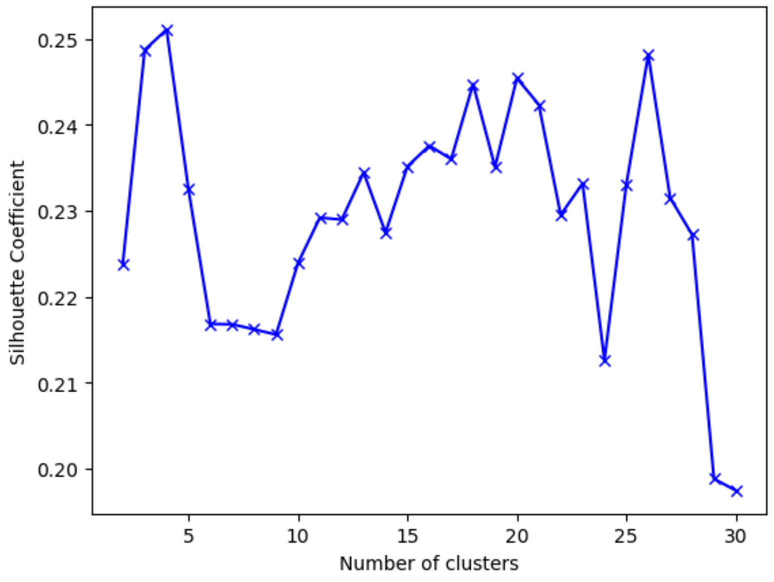
Experiments

To compare the performances for the algorithms above, we used the aforementioned 3 datasets which have differing dataset sizes so that obvious comparisons can be made between the performance of the algorithms with respect to differing dataset sizes.

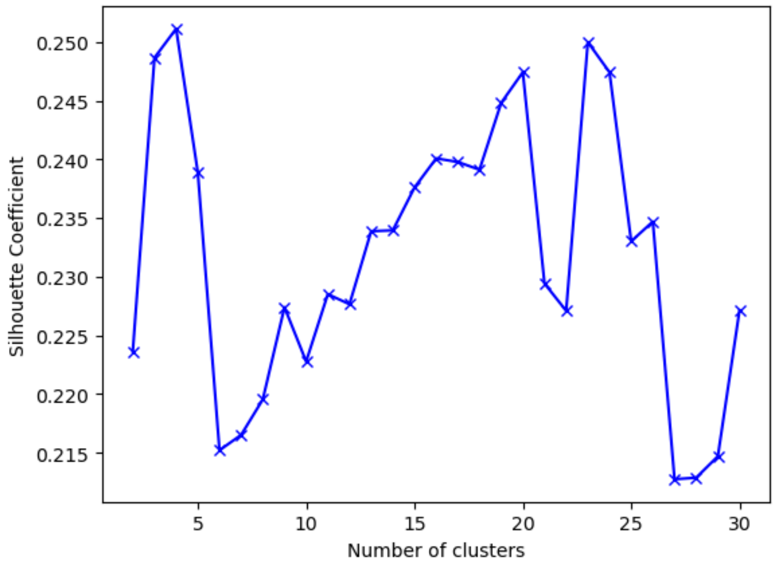
All the dataset exploration performed solely on the K-means Jupyter notebooks and then appropriate data cleaning will be applied to all Jupyter notebooks. Generally, we have removed class labels and removed columns that have too many unique values to be meaningful to use for clustering. Then, for datasets with remaining non-numerical columns, we used the SciKit Learn library's label encoder to transform the columns to become numerical with each number representing a specific label. Finally, we would then use SciKit Learn's MinMaxScaler to normalize the columns for clustering to be performed.

Weather Type Classification:

This dataset class label is ‘Weather Type’ which we promptly removed and has 3 non-numerical data columns: ‘Cloud Cover’, ‘Season’ and ‘Location’ with the first 2 having 4 unique values and the last having 3. There was no null data in the dataset leaving us with 13200 data rows with 10 columns, so we proceeded to apply label encoder on these 3 columns and normalize the data. Then we proceeded with finding the best parameters for each method as shown with the plots below.

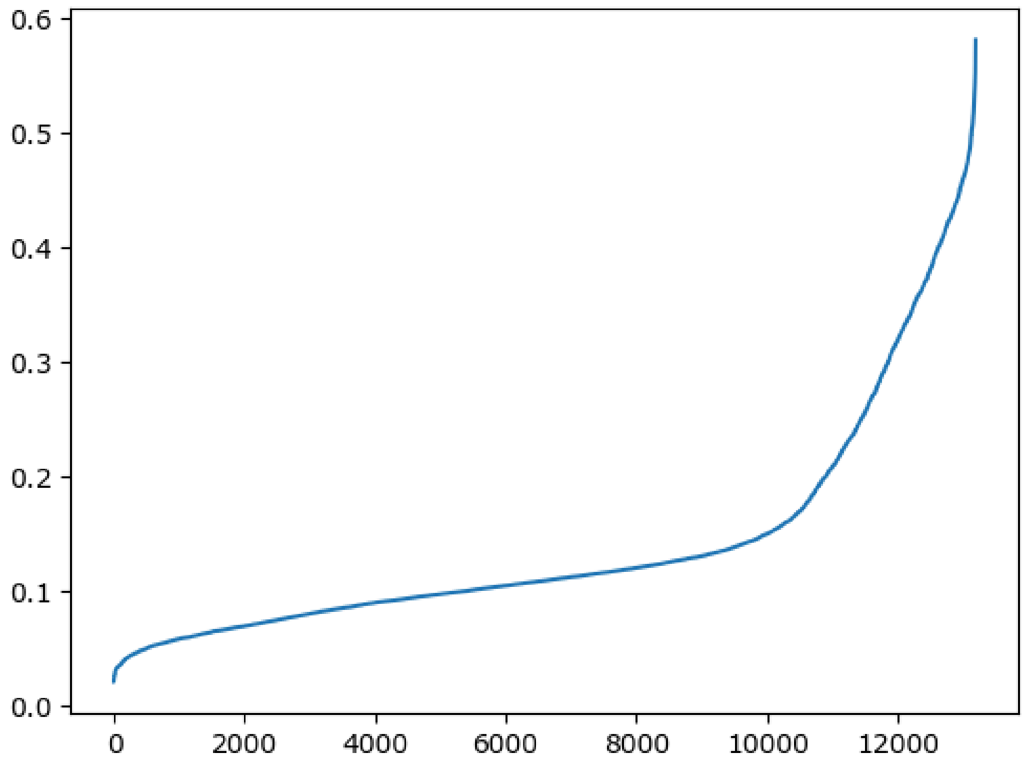


Silhouette method for K-Means



Silhouette method for K-Means++

Best silhouette scores for both K-means and K-means++ are produced by k=4.



20-NN distance plot

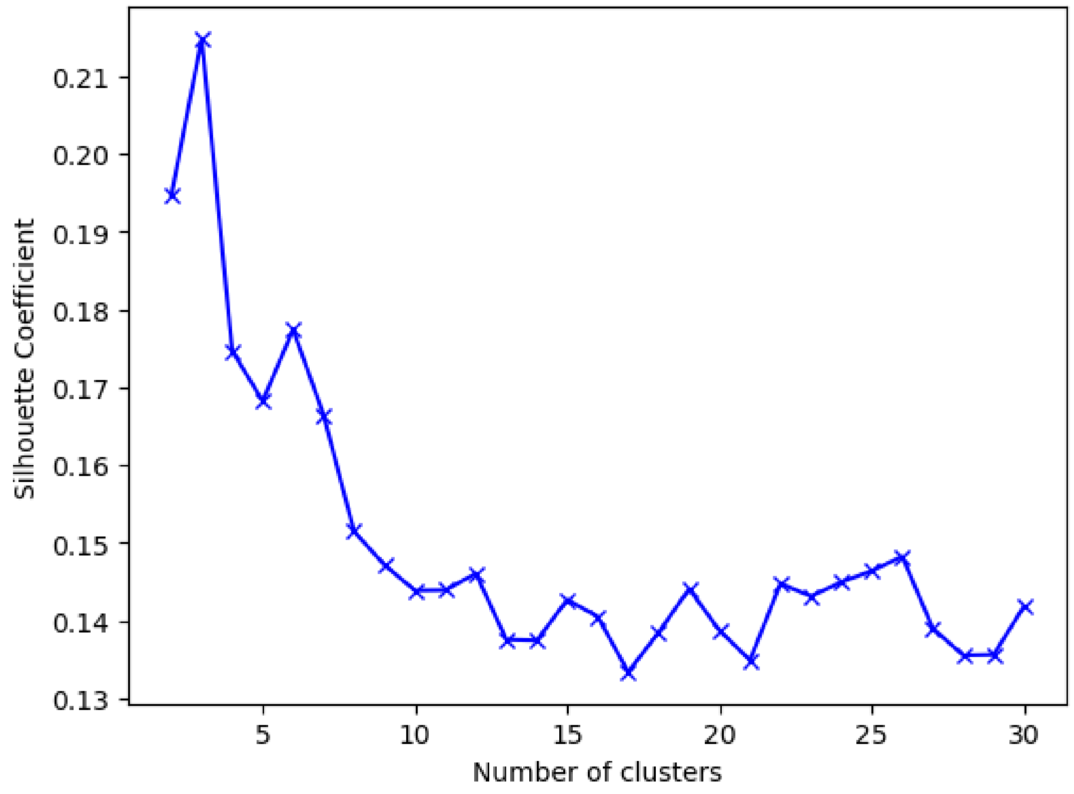
We take the 2nd elbow point which is around 0.4 to 0.5. So we tested the silhouette scores of epsilon from 0.4 to 0.5 in intervals of 0.01 and minimum sample points from 11 to 20 and ended up with epsilon of 0.5 and minimum sample of 13 with the best silhouette score of 0.116401

Performance Comparison:

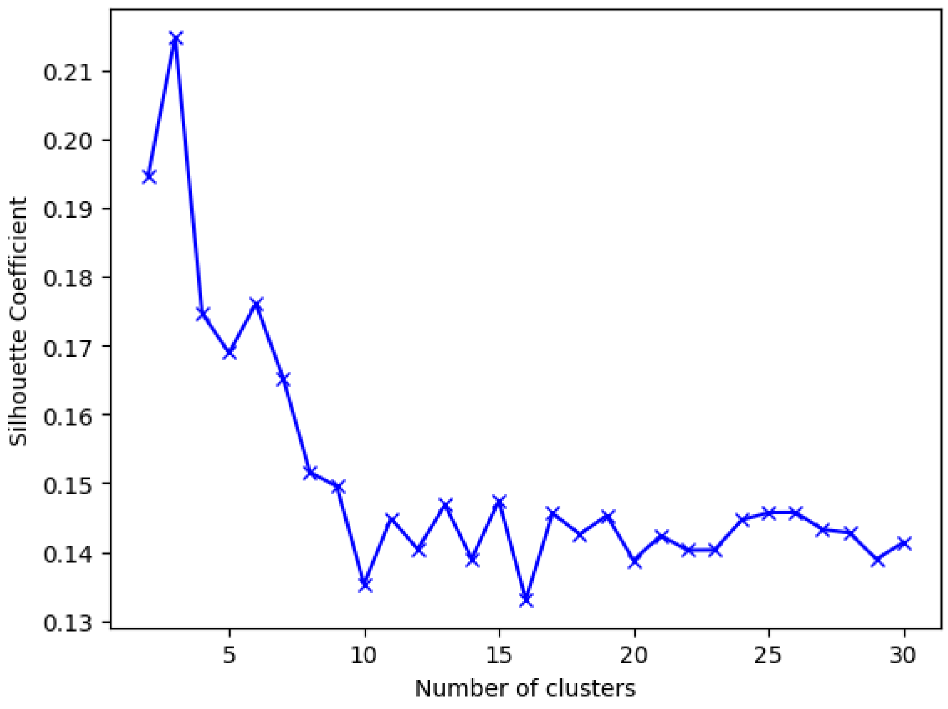
Clustering Methods	K-means	K-means++	DBSCAN
Chosen Parameters	k=4		Eps:0.5, Min sample:13
Number of Clusters/Noise points	4		Clusters:3, Noise Points:556
Time taken (in seconds)	0.254578	0.353435	0.275501
Inertia (Sum of squared L2 distance)	5105.994864	5105.979211	NA
Silhouette Score	0.251117		Before removing noise: 0.116401 After: 0.125480

Comprehensive Diabetes Clinical:

This dataset class label is ‘diabetes’ which we promptly removed. Then there are 5 race label binary columns but it is such that every row has only 1 race label as 1 and everything else as 0 so a new column, ‘race_label’ was made where it just has the race label for each row of data and then removed the 5 binary columns. The dataset has no null data and 3 other non-numerical columns: ‘gender’, ‘location’ and smoking_history‘, ‘location’ has 55 unique categories and ‘smoking_history’ has 6 unique categories which are good enough to use for clustering, leaving us with 100000 data rows with 10 columns. Then we applied the label encoder on the 3 columns and the newly created ‘race_label’ column and normalized the data with MinMaxScaler and proceeded with finding the parameters for the clustering methods as shown with the plots below.

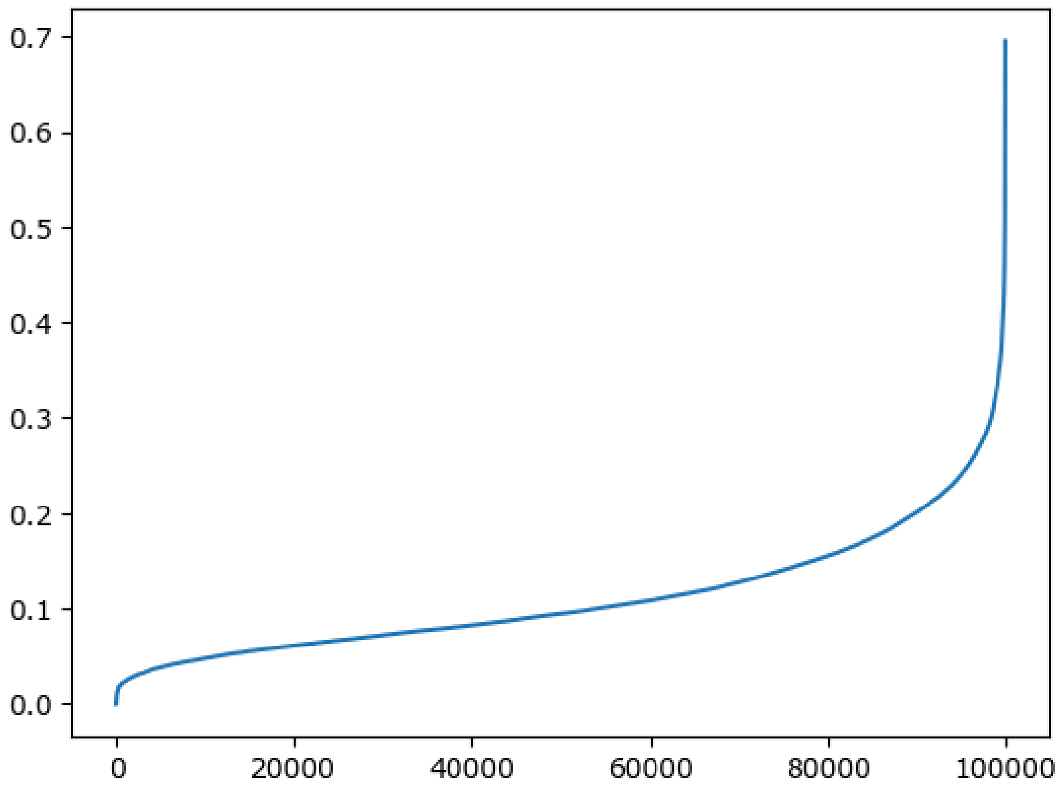


Silhouette method for K-Means



Silhouette method for K-Means

Best silhouette scores for both K-means and K-means++ are produced by k=3.



22-NN distance plot

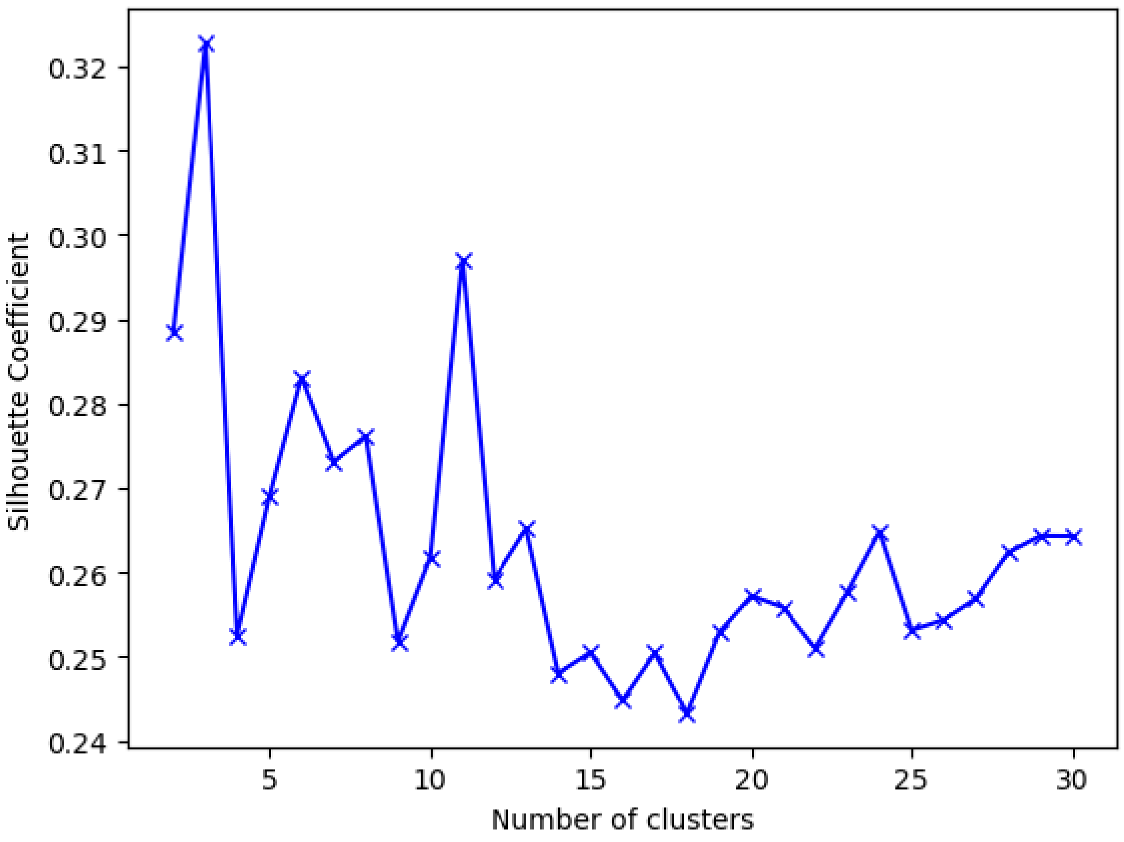
Taking elbow point which is around 0.2 to 0.3. So we tested the silhouette scores of epsilon from 0.2 to 0.3 in intervals of 0.01 and minimum sample points from 12 to 23 and ended up with epsilon of 0.3 and minimum sample of 20 with the best silhouette score of 0.083822

Performance Comparison:

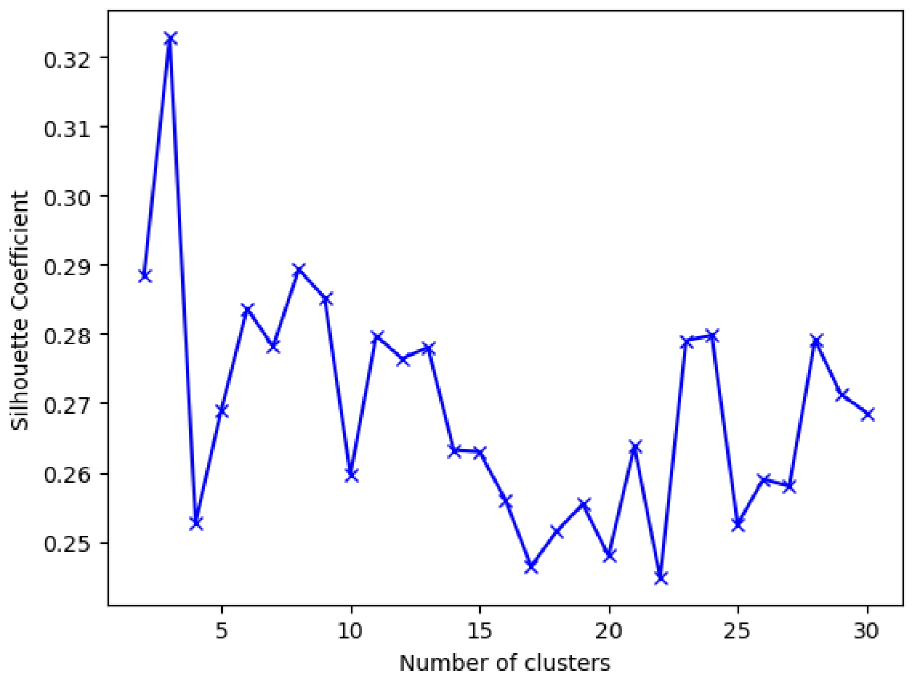
Clustering Methods	K-means	K-means++	DBSCAN
Chosen Parameters	k=3		Eps:0.3, Min sample:20
Number of Clusters/Noise points	3		Clusters:9, Noise Points:8107
Time taken (in seconds)	0.475535	0.518494	5.066284
Inertia (Sum of squared L2 distance)	50313.874713	50313.828257	NA
Silhouette Score	0.214782		Before removing noise: 0.083822 After: 0.118926

Enron Fraud Email:

This dataset class label is ‘Label’ which we promptly removed. This dataset with 30 columns has mostly non-numerical data and hence we had to do extensive exploring of data for this dataset. Firstly, we removed ‘Folder-User’, ‘Message-ID’, ‘Date’, ‘From’, ‘To’, ‘Subject’, ‘X-From’, ‘X-cc’, ‘X-Folder’, ‘Body’, ‘Cc’, ‘Bcc’, ‘Re, ‘Source’, ‘Mail-ID’, ‘Suspicious-Folders’ were removed either because they had only 1 unique value, too many unique values. We also removed ‘Mime-Version’ at the same time due to the label encoder not being able to encode multiple types. Then, we found that ‘X-bcc’ columns had 447240 null data out of 447417 data rows so those 2 columns were removed and got rid of any remaining data rows with null data, leaving us with 436571 data rows with 13 columns. Afterwards, we proceeded to apply label encoders to most of the columns and normalized the dataset to find parameters for clustering as shown with the plots below.

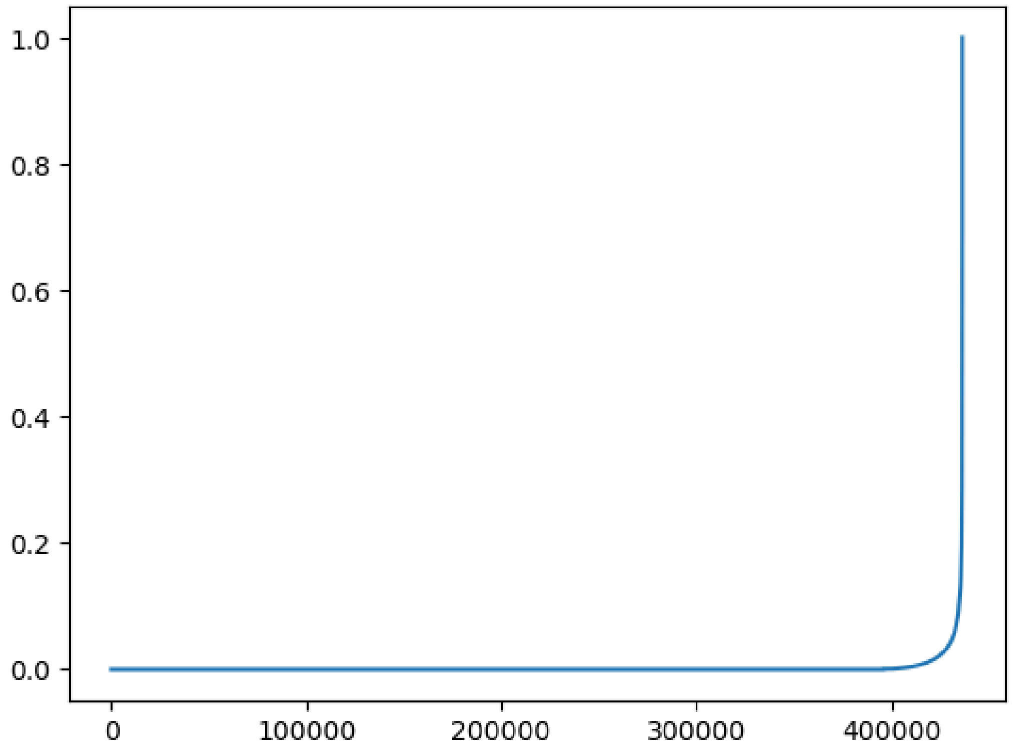


Silhouette Method for K-Means



Silhouette Method for K-Means++

Best silhouette scores for both K-means and K-means++ are produced by k=3.



26-NN Distance Plot

Taking elbow point which is around 0.02 to 0.05. So we tested the silhouette scores of epsilon from 0.02 to 0.05 in intervals of 0.01 and minimum sample points from 14 to 26 and ended up with epsilon of 0.02 and minimum sample of 14 with the best silhouette score of 0.173855

Performance Comparison:

Clustering Methods	K-means	K-means++	DBSCAN
Chosen Parameters	k=3		Eps:0.02, Min sample:14
Number of Clusters/Noise points	3		Clusters:4311, Noise Points:98396
Time taken (in seconds)	1.101637	1.227193	94.400563
Inertia (Sum of squared L2 distance)	221808.665369	194178.938047	NA
Silhouette Score	0.252784		Before removing noise: 0.173855 After: 0.487056

Comparison of the performance of the methods

From all the results of the experiments, all the K-means and K-means++ clustering had the exact same silhouette scores with K-means++ having better inertia although only slight improvements for small data points and only more pronounced with the email dataset. This could be due to the fact that K-Means++ has an algorithm to initialize centroids versus the random initialization of K-Means.

The silhouette scores of K-Means/K-Means++ versus DBSCAN had varying results amongst the 3 datasets. The first 2 datasets had better silhouette scores for K-Means/K-Means++ versus DBSCAN but it was different for the Enron fraud email dataset. The weather type classification and comprehensive diabetes clinical datasets had relatively low DBSCAN's silhouette scores before noise points were removed and them having a small percentage of points being noise made the removal of noise very insignificant in improving the silhouette score, possibly indicating that the parameters may have been too lax and should possibly be stricter to possibly produce better fitting clusters with better silhouette scores. On the other hand, Enron fraud email dataset had very substantial amount of noise and hence had its silhouette score improve by more than 2 times before noise was removed, this also indicates that the remaining data points of the dataset are more non-spherical but in small enough clusters to be better clustering models than K-Means and K-Means++.

Putting comparisons between the silhouette scores aside, most of the silhouette scores are relatively low with most of them being below 0.3 other than Enron fraud email's DBSCAN silhouette score after removing noise. This probably indicates that any of these three clustering methods are not very suitable to determine clusters for the weather type classification and comprehensive diabetes clinical datasets and K-Means/K-Means++ is not suitable for Enron fraud email.

Time taken increases by dataset size for all three methods but differences between the methods are more pronounced as the dataset size increases. For the weather dataset, the time difference is generally insignificant due to its relatively small size. Then for the other 2 datasets, it is clear that time taken for K-means is always the fastest followed by a slight increase with K-means++ probably due to extra time taken for centroid initialization versus random initialization in K-means clustering and then very significant time increase for DBSCAN with the increase being very significant with the biggest dataset.

Conclusion

In this project, we conducted a comparison of K-means, K-means++, and DBSCAN algorithms across three distinct datasets. Our experiments have shown how effective these 3 algorithms may be, determined by their Silhouette Score and Time taken, when it comes to conducting clustering on different datasets, varies and each algorithm has its strengths and weaknesses.

Limitations we encountered included a lack of datasets that suited our needs. We attempted to additionally assess the scalability of each clustering method, however the results we obtained do not give a good reflection of each algorithm's performance on increasingly large datasets. This problem could be alleviated if we could have found multiple datasets on the same data, but with varying sizes. Additionally, more experiments with a larger number of datasets would be useful in demonstrating each algorithm's scalability.

Further work could include exploring the use of more advanced parameter optimization techniques, so as to demonstrate the significance in choosing appropriate starting parameters. The impact of dimensionality reduction techniques on clustering performance when exploring datasets with large numbers of attributes could also be examined.

References

[1] Nikhil7280. (2021). *Weather type classification*. Kaggle. Retrieved from https://www.kaggle.com/datasets/nikhil7280/weather-type-classification

[2] Priyam Choksi. (2021). *100,000 diabetes clinical dataset*. Kaggle. Retrieved from https://www.kaggle.com/datasets/priyamchoksi/100000-diabetes-clinical-dataset

[3] Advaith Srao. (2021). *Enron fraud email dataset*. Kaggle. Retrieved from https://www.kaggle.com/datasets/advaithsrao/enron-fraud-email-dataset

[4] GeeksForGeeks. (n.d.). *K-means clustering: Introduction*. Retrieved from https://www.geeksforgeeks.org/k-means-clustering-introduction/

[5] Tomar, A. (2021). *How to determine the optimal number of clusters in K-means using the elbow method*. Built In. Retrieved from https://builtin.com/data-science/elbow-method

[6] Khosla, S. (2021). *ML - K-means algorithm*. GeeksForGeeks. Retrieved from https://www.geeksforgeeks.org/ml-k-means-algorithm/

[7] Dey, D. (2021). *DBSCAN clustering in ML: Density-based clustering*. GeeksForGeeks. Retrieved from https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/

[8] Masoud, A. (2022). *How to determine epsilon and MinPts parameters of DBSCAN clustering*. Retrieved from https://www.sefidian.com/2022/12/18/how-to-determine-epsilon-and-minpts-parameters-of-dbscan-clustering/

[9] Hashmi, F. (2021). *How to create clusters using DBSCAN in Python*. Thinking Neuron. Retrieved from https://thinkingneuron.com/how-to-create-clusters-using-dbscan-in-python/