CENG463 ASSIGNMENT 2 REPORT

Members: Ceren Çağlayan & Elif Özyürek

ID: 270201059 & 280201079

Department: Computer Engineering

Lecture: Introduction to Machine Learning

Lecturer: Dr. Emrah İnan

Table of Contents

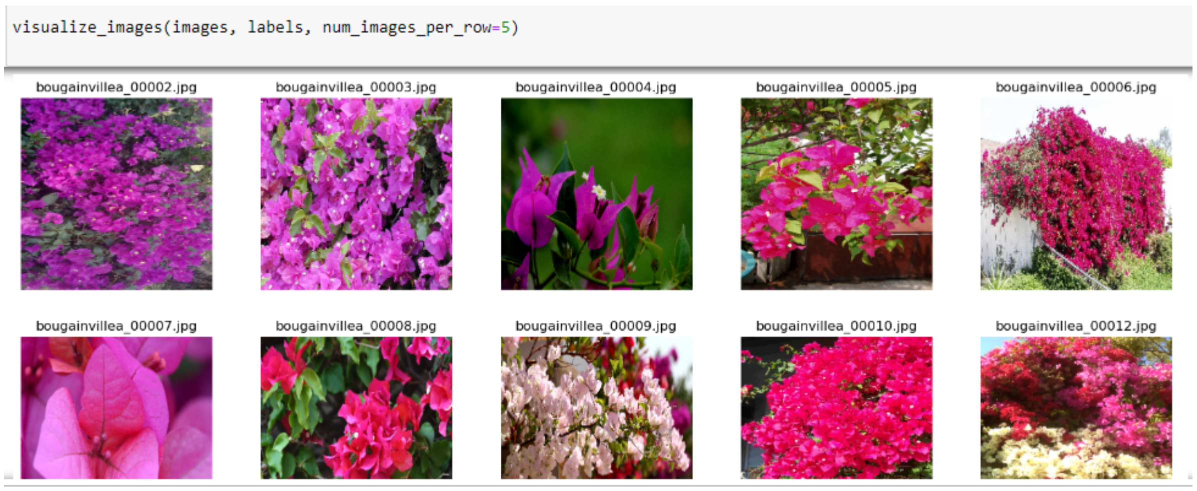
1. Describing the Shape and Features of the Dataset

1.1 Choosing the Dataset

The dataset used in all the experiments is the Flowers dataset from Kaggle. [1] The dataset has 733 images of 10 different types of flowers. The details about the contents of the dataset will be further discussed in the next sections.

To process the image data, Python's OpenCV library was used. To preserve the quality of images, the images were resized using the following line of code. The first ten images of the dataset are shown in *Figure.0*.

image = cv2.resize(image, image_size, interpolation=cv2.INTER_LANCZOS4)

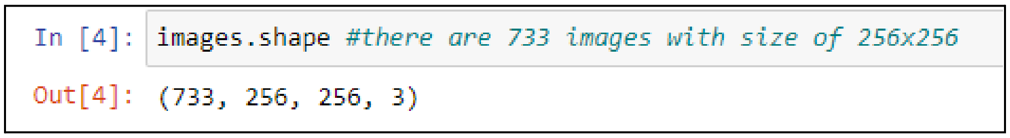


```
visualize_images(images, labels, num_images_per_row=5)
```

Figure.0

1.2 Shape and Features

The shape of the flowers dataset can be seen in *Figure.1*. There are 733 flower images of 256x256 pixel size, and all images use 3 color channels (RGB color space).

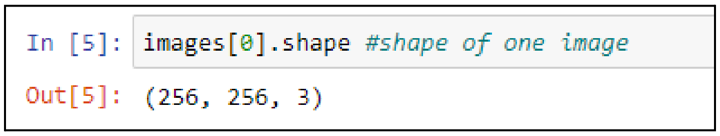


```
In [4]: images.shape #there are 733 images with size of 256x256

Out[4]: (733, 256, 256, 3)
```

Figure.1

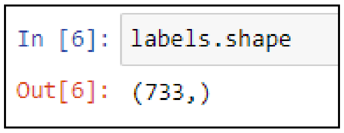
The shape of each image can be seen in *Figure.2*. All images are of 256x256 pixel size, and use 3 color channels (RGB color space).



```
In [5]:  images[0].shape #shape of one image
Out[5]:  (256, 256, 3)
```

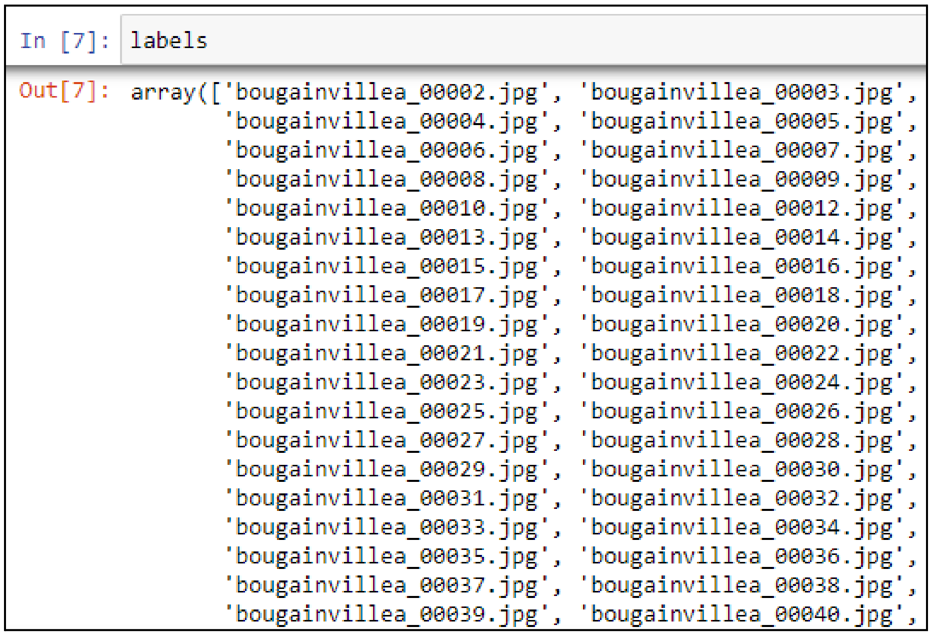
Figure.2

Labels of the 733 images are kept in a one-dimensional array with size 733. The shape of the labels array and the array itself is shown in *Figure.3* and *Figure.4* respectively.



```
In [6]:  labels.shape
Out[6]:  (733,)
```

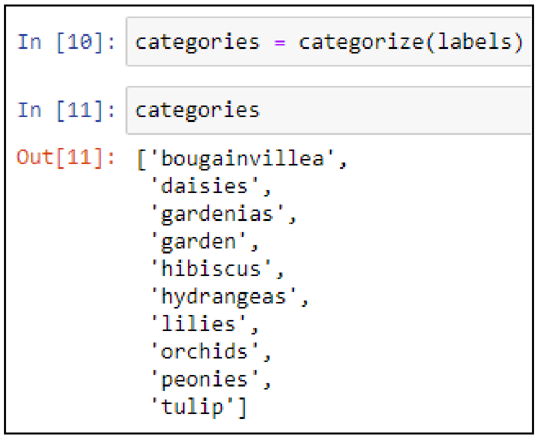
Figure.3



```
In [7]:  labels
Out[7]:  array(['bougainvillea_00002.jpg', 'bougainvillea_00003.jpg',
                'bougainvillea_00004.jpg', 'bougainvillea_00005.jpg',
                'bougainvillea_00006.jpg', 'bougainvillea_00007.jpg',
                'bougainvillea_00008.jpg', 'bougainvillea_00009.jpg',
                'bougainvillea_00010.jpg', 'bougainvillea_00012.jpg',
                'bougainvillea_00013.jpg', 'bougainvillea_00014.jpg',
                'bougainvillea_00015.jpg', 'bougainvillea_00016.jpg',
                'bougainvillea_00017.jpg', 'bougainvillea_00018.jpg',
                'bougainvillea_00019.jpg', 'bougainvillea_00020.jpg',
                'bougainvillea_00021.jpg', 'bougainvillea_00022.jpg',
                'bougainvillea_00023.jpg', 'bougainvillea_00024.jpg',
                'bougainvillea_00025.jpg', 'bougainvillea_00026.jpg',
                'bougainvillea_00027.jpg', 'bougainvillea_00028.jpg',
                'bougainvillea_00029.jpg', 'bougainvillea_00030.jpg',
                'bougainvillea_00031.jpg', 'bougainvillea_00032.jpg',
                'bougainvillea_00033.jpg', 'bougainvillea_00034.jpg',
                'bougainvillea_00035.jpg', 'bougainvillea_00036.jpg',
                'bougainvillea_00037.jpg', 'bougainvillea_00038.jpg',
                'bougainvillea_00039.jpg', 'bougainvillea_00040.jpg',
```

Figure.4

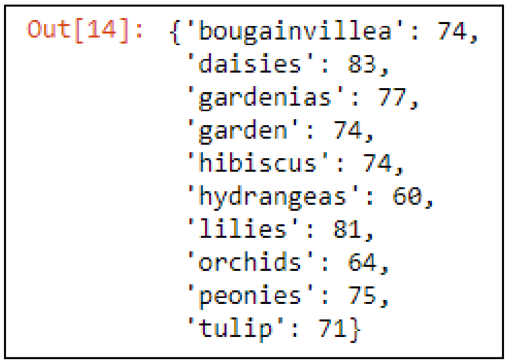
The flower types in the dataset are extracted from the labels with the function *categorize(labels)*. The different types of flowers can be seen in *Figure.5*, and the number of flowers for each flower type can be seen in *Figure.6*.



```
In [10]: categories = categorize(labels)

In [11]: categories
Out[11]: ['bougainvillea',
          'daisies',
          'gardenias',
          'garden',
          'hibiscus',
          'hydrangeas',
          'lilies',
          'orchids',
          'peonies',
          'tulip']
```

Figure.5



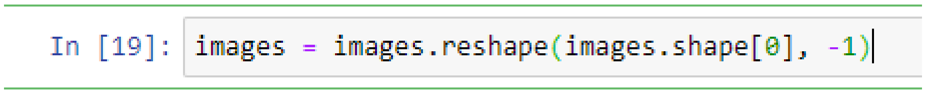
```
Out[14]: {'bougainvillea': 74,
          'daisies': 83,
          'gardenias': 77,
          'garden': 74,
          'hibiscus': 74,
          'hydrangeas': 60,
          'lilies': 81,
          'orchids': 64,
          'peonies': 75,
          'tulip': 71}
```

Figure.6

2. Setup and Choosing Clustering Algorithms

2.1 Preprocessing

For the preprocessing of data, the images in the dataset are "flattened". This flattens the images and converts each image into a vector. The code for flattening is shown in *Figure.7*.



```
In [19]: images = images.reshape(images.shape[0], -1)
```

Figure.7

The flattened data before applying any preprocessing can be seen in *Figure.8*. The figure shows that the value range of data is very high before any preprocessing is applied. To prevent this and keep the data in a certain value range, a scaling method should be used.

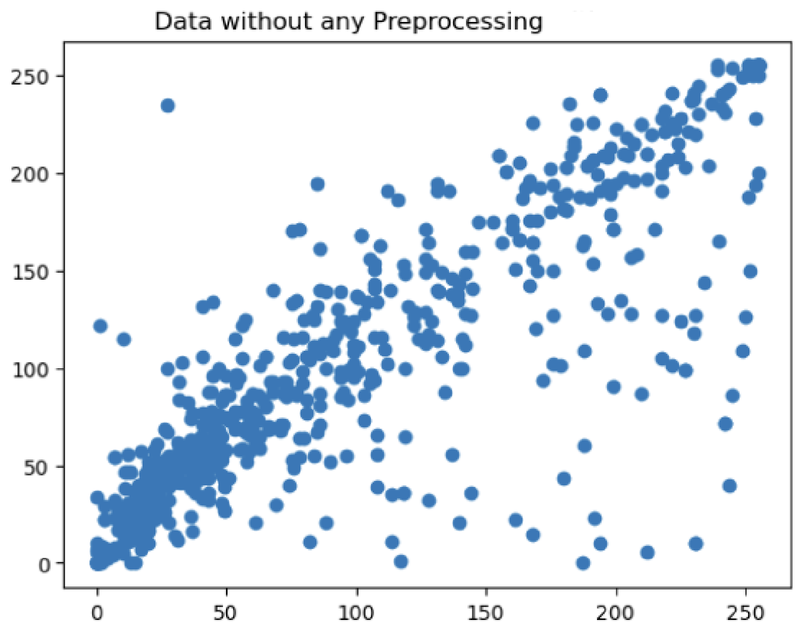


Figure.8

For the scaling of data, two scaling methods were tested with K-Means Clustering: Min-Max Scaler and Standard Scaler. Although both methods performed similarly, scaling the data in the range [0,1] was preferred. The comparison for StandardScaler and MinMaxScaler is shown in the following figures. (*Figure.9* and *Figure.10*)

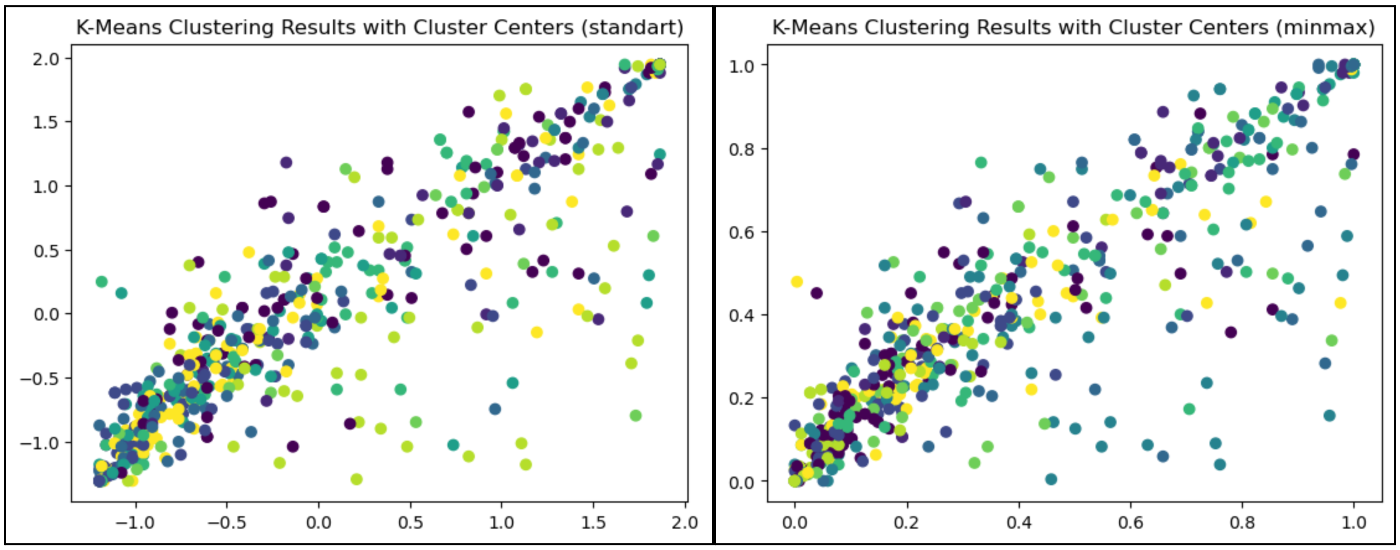


Figure.9

In *Figure.10*, the clustering obtained with the K-Means algorithm is visualized using the Manifold Learning Algorithm t-SNE. The reason for using t-SNE and the effects of scaling will be further discussed in the section *3.Results*.

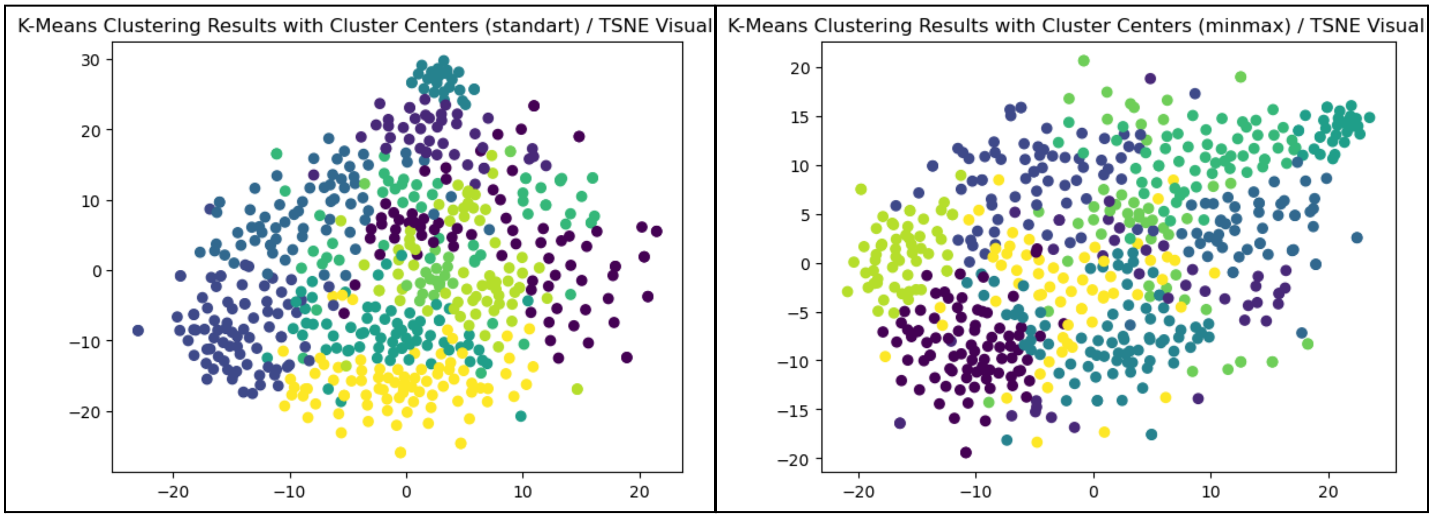


Figure.10

2.2. Choosing Clustering Algorithms

2.2.1 K-Means

It is an algorithm that works by dividing data points into a certain number of clusters. Each data point is assigned to the closest cluster center, and cluster center is set as the mean of the points assigned to it. It usually fails with complex shaped clusters.

For the K-Means algorithm, each direction is of the same importance, and the number of clusters should be specified beforehand. Since there are 10 different types of flowers in the Flowers dataset, the algorithm was initialized with n_clusters=10.

2.2.2 Agglomerative

Agglomerative clustering declares each point its own cluster, then merges two most similar clusters until a specified number of clusters are left. This algorithm also requires the number of clusters to be set beforehand. It usually fails with complex cluster shapes.

Since there are 10 different types of flowers in the Flowers dataset, the algorithm was initialized with n_clusters=10.

Hierarchical clustering provides the intermediate steps of creating clusters with agglomerative clustering. All possible clusterings can be observed using hierarchical clustering, and its visualization tool dendrograms. The dendrogram for the hierarchical clustering of the Flowers dataset can be seen in *Figure.11*. The thin black line that lies between cluster distances 200 and 400 creates 10 clusters, which is what the algorithm finds when n_clusters=10.

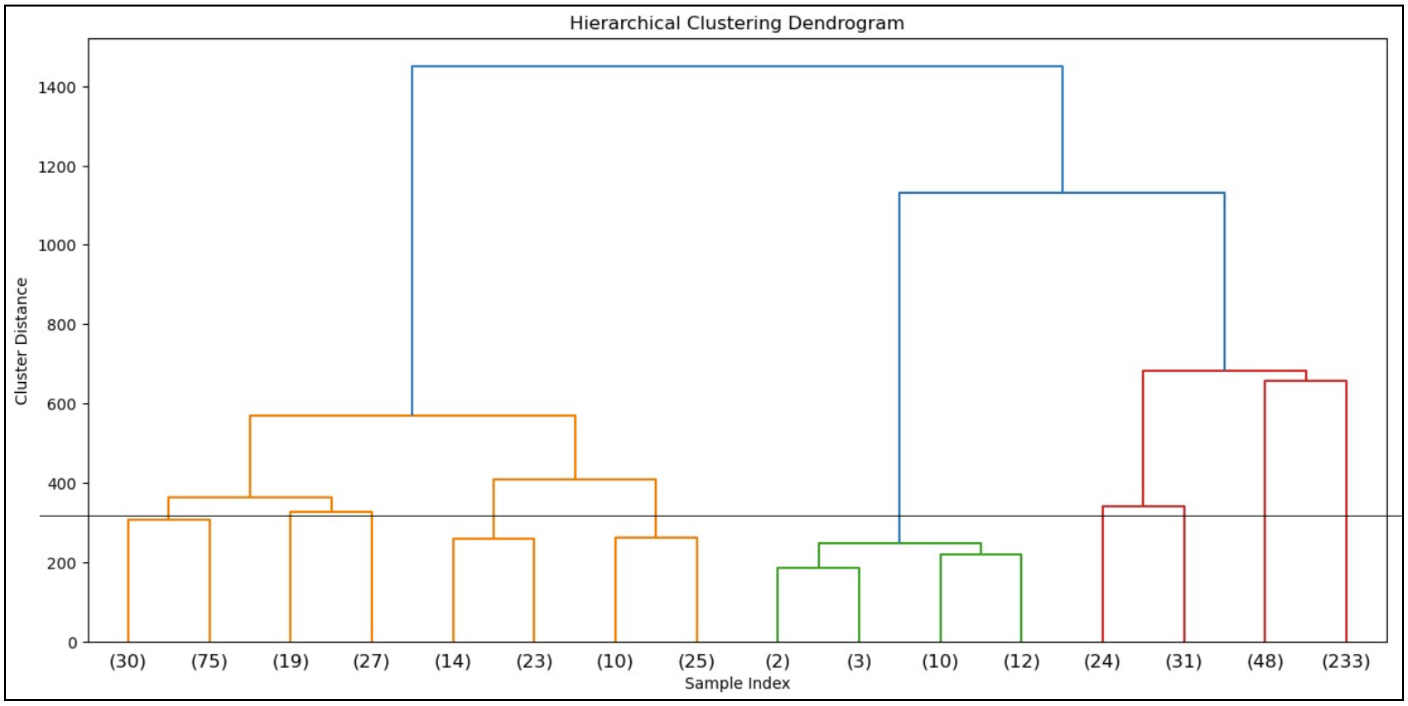


Figure.11

2.2.3 DBSCAN

DBSCAN algorithm is a density-based clustering algorithm. Unlike other algorithms, it has additional parameters called "eps" and "min_samples" and does not require setting the number of clusters. It usually works well with complex cluster shapes. If a point is labeled -1, this means that it is not a member of any cluster. In other words, the point is a “noise”.

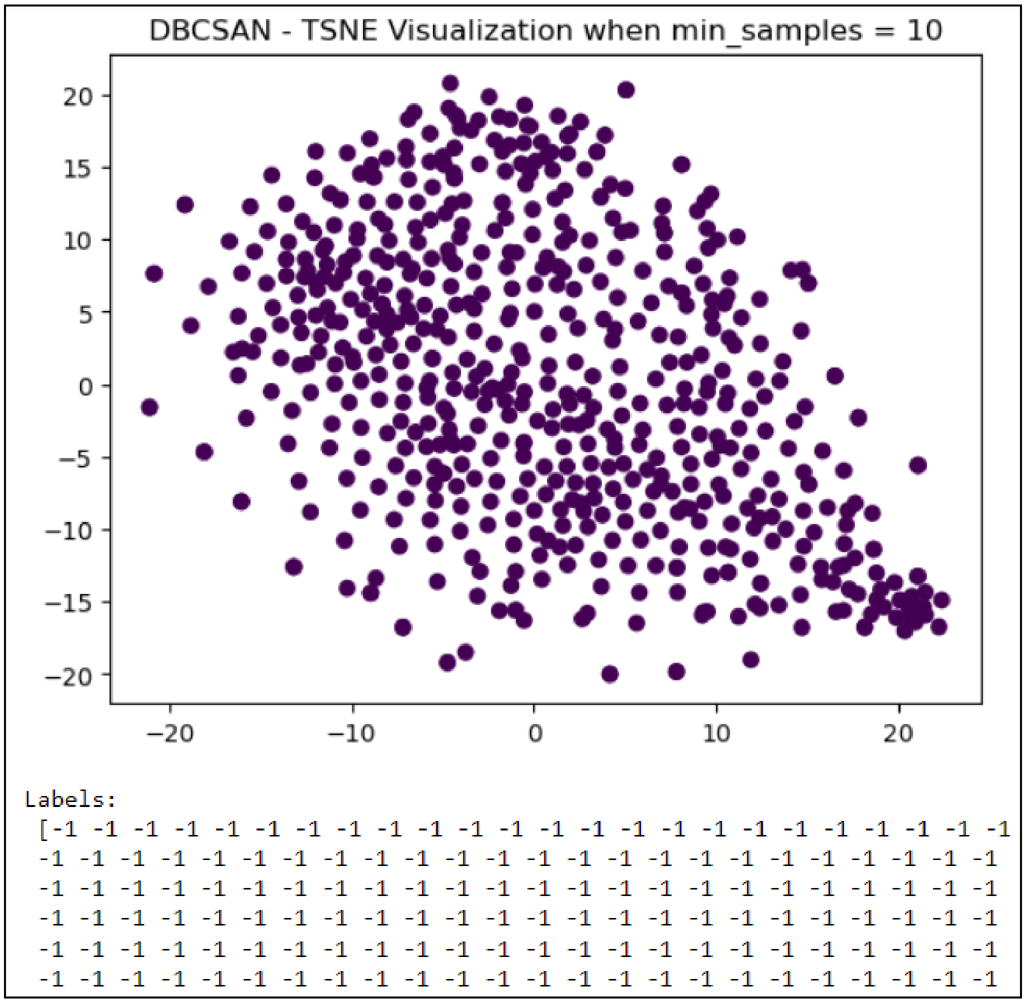
2.2.3.1 Tuning Parameters

The parameters *eps* and *min_samples* were adjusted to reach the optimal number of clusters.

eps is the parameter that decides what “points being close to each other” means, and it indirectly decides on how many clusters there will be. min_samples is the parameter that decides the minimum cluster size.

***min_samples*:** For this dataset, any value for min_samples that is greater than 1 resulted in all points being labeled -1, or in other words, noise. When min_samples=1, this issue is resolved and most of the noise values become core points or boundary points. Thus, min_samples is set to be 1.

t-SNE visualization of the DBSCAN algorithm with eps=1 and min_samples=10 can be seen in *Figure.12.* As mentioned above, all points are considered noises and their labels are -1.



```
Labels:
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Figure.12

t-SNE visualization of the DBSCAN algorithm with eps=1 and min_samples=1 can be seen in *Figure.13*. The algorithm creates 511 clusters with this setting of parameters.

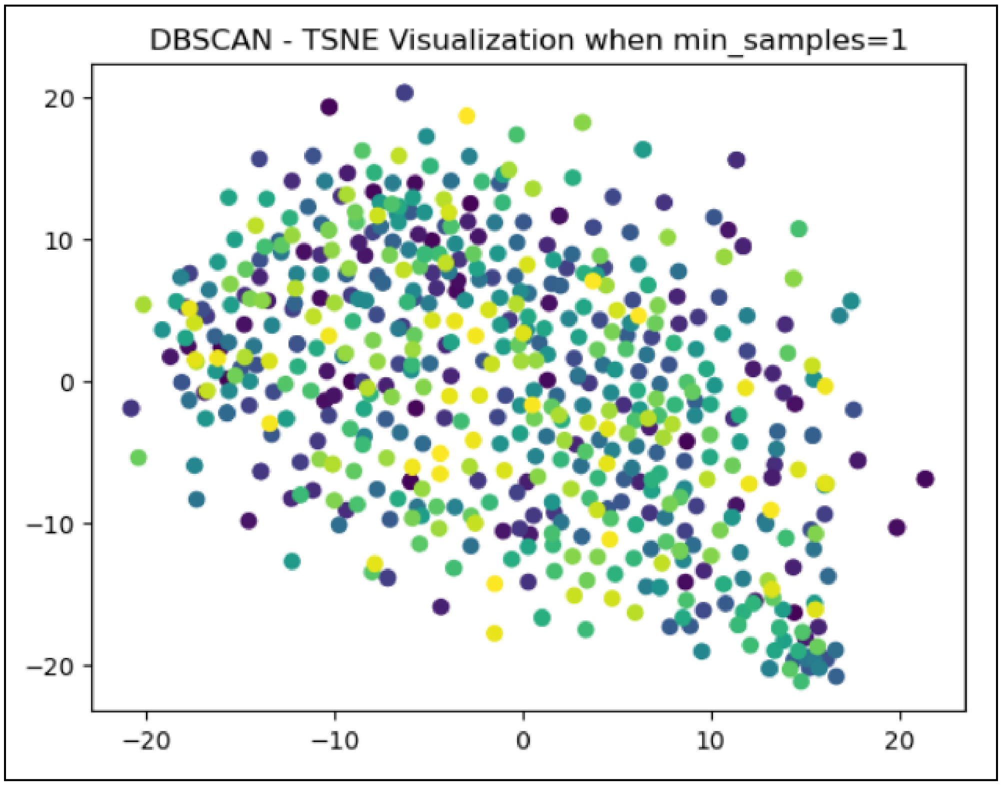


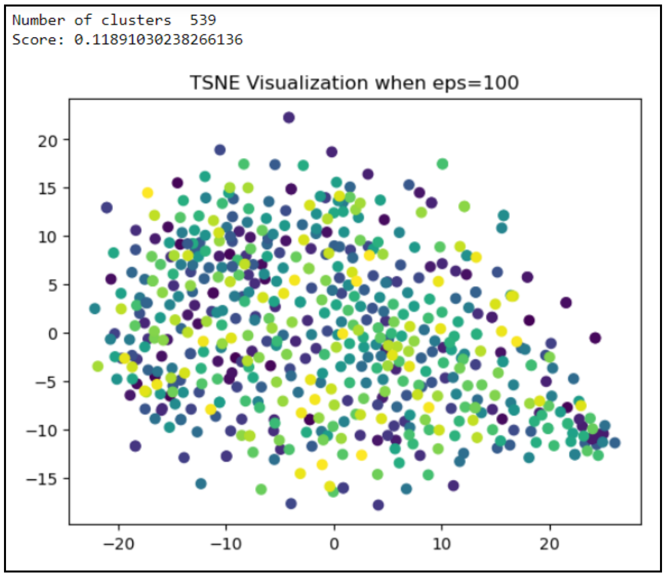
Figure.13

eps**:** It was observed that when the value of eps is increased, the number of clusters decreases. Since it is very hard to manually observe the effect of eps, it was scored with the silhouette score method.

The scores for different values of eps:

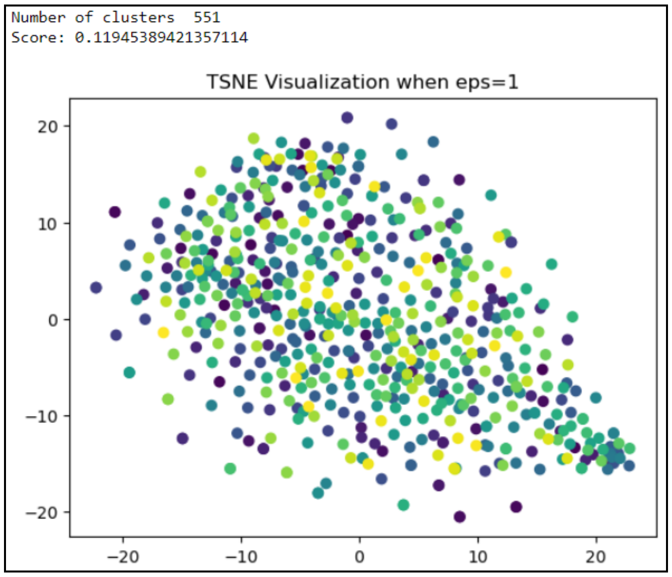
eps=100 → 0.118910…
eps=1 → 0.119454…

The scores, number of clusters, and visualization of the algorithm with different values of epc are shown in *Figure.14* and *Figure.15*. Since the score of eps=1 is higher, this setting was preferred for the final result.



```
Number of clusters  539
Score: 0.11891030238266136
```

Figure.14



```
Number of clusters  551
Score: 0.11945389421357114
```

Figure.15

3. Results

Both MinMaxScaler and StandardScaler were used for observing scores. The silhouette score obtained from the Agglomerative algorithm was slightly lower when StandardScaler was used. The reason for this can be that MinMaxScaler is more resistant to outliers. The silhouette scores and t-SNE visualizations for different scalers are shown in *Figure.16* and *Figure.17*.

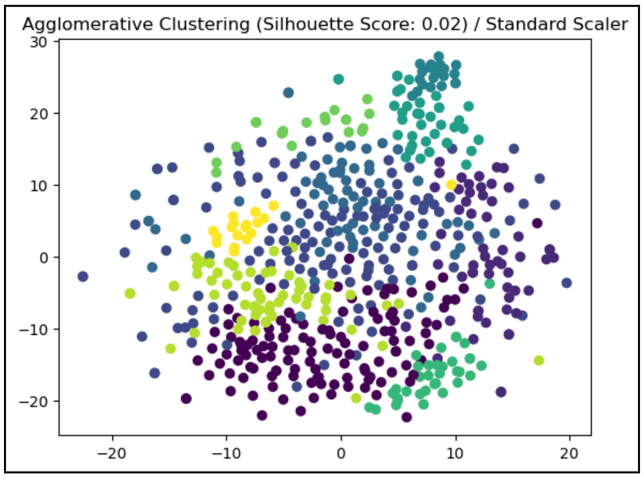


Figure.16

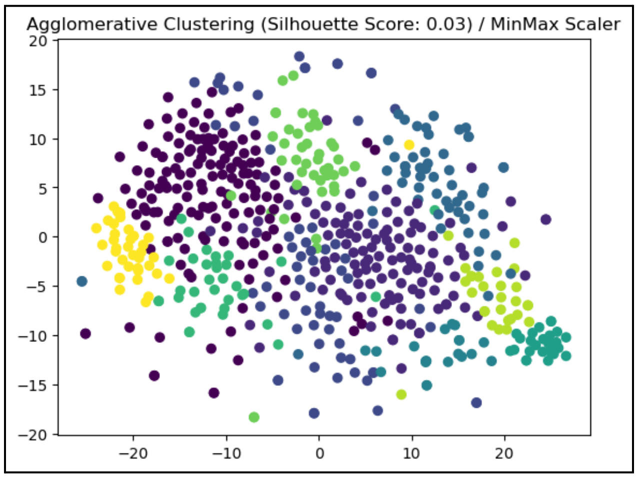


Figure.17.

For the Visualization / Dimension reduction part, the manifold algorithm t-SNE was preferred. The reason for this is that it looks at the closeness of the points in the original space, and brings the closer ones closer while taking the further ones further away. In addition, since its running time is long, it has been found that when you want to run the .ipynb file containing the codes, the running time is noticeably extended.

The final setup:

Scaler: MinMaxScaler

Visualization algorithm: t-SNE (manifold algorithms)

Number of clusters for K-means and Agglomerative clustering: 10

Eps and min_samples for DBSCAN: 1

Scoring method: Silhouette Score

Final Results:

Although the results were low as expected, better results could have been obtained with a more consistent dataset with different features.

K-Means (Silhouette Score: 0.04):

In K-Means clustering, a very low score was achieved. Since K-Means is based on a linear structure, it may have difficulty capturing non-linear structures in our data set. The visualization and score for this algorithm is shown in *Figure.18*.

Agglomerative Clustering (Silhouette Score: 0.03):

The lowest score was observed in Agglomerative. The reason for this score may be that it is difficult to distinguish between clusters in a hierarchical structure. The visualization and score for this algorithm is shown in *Figure.19*.

DBSCAN (Silhouette Score: 0.12):

Silhouette score was higher for DBSCAN than other algorithms. This is because DBSCAN has an adaptive structure for clustering and can handle different density regions in the dataset well. By setting the parameters correctly, the optimal number of clusters were achieved, and the number of noise points was reduced. The visualization and score for this algorithm is shown in *Figure.20*.

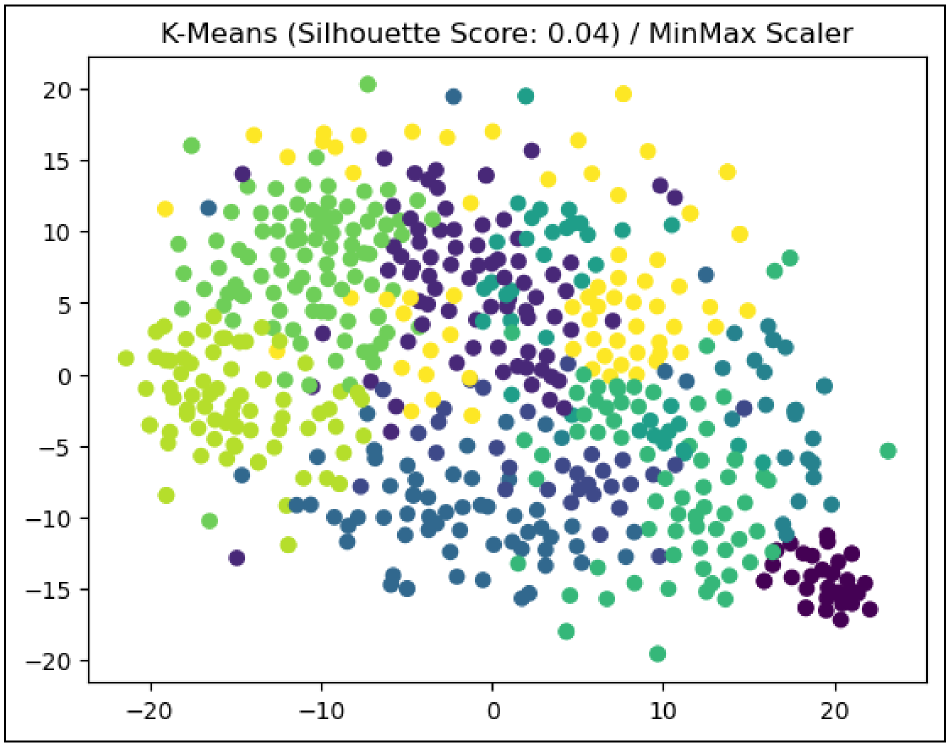


Figure.18

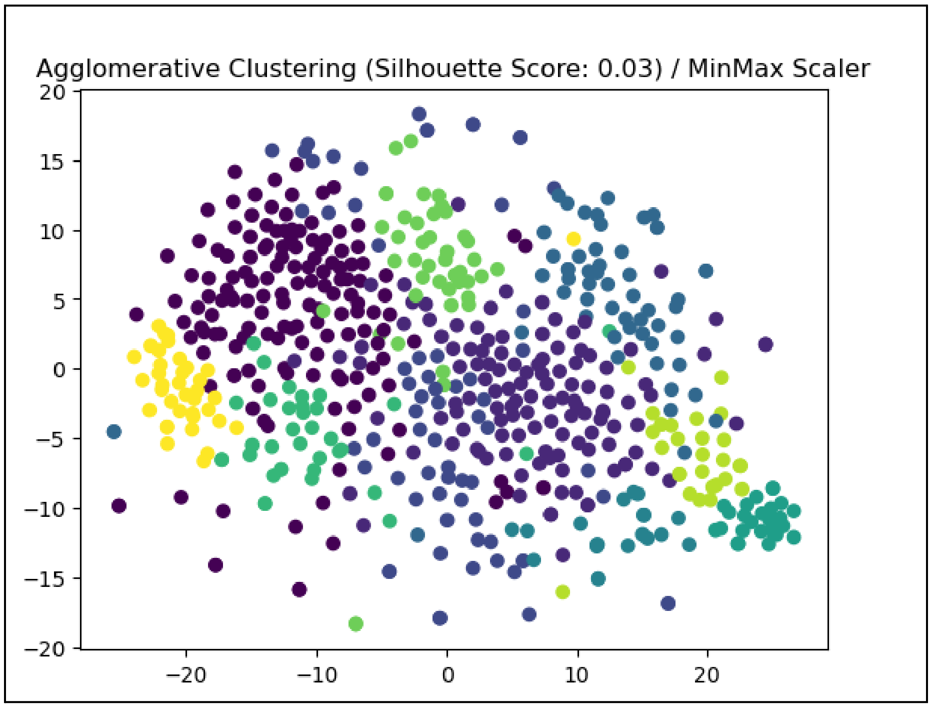


Figure.19

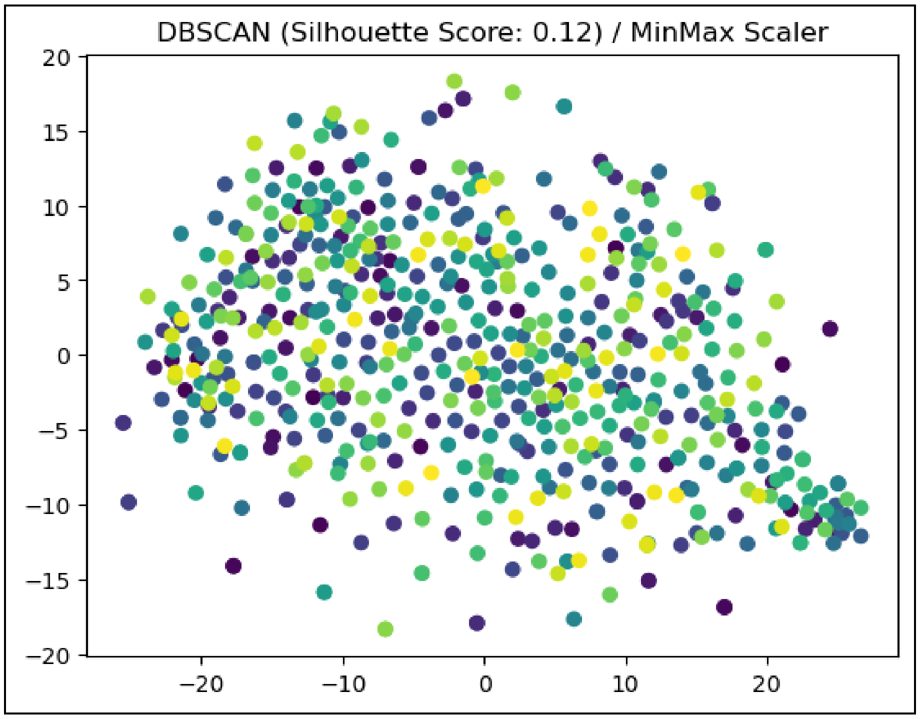
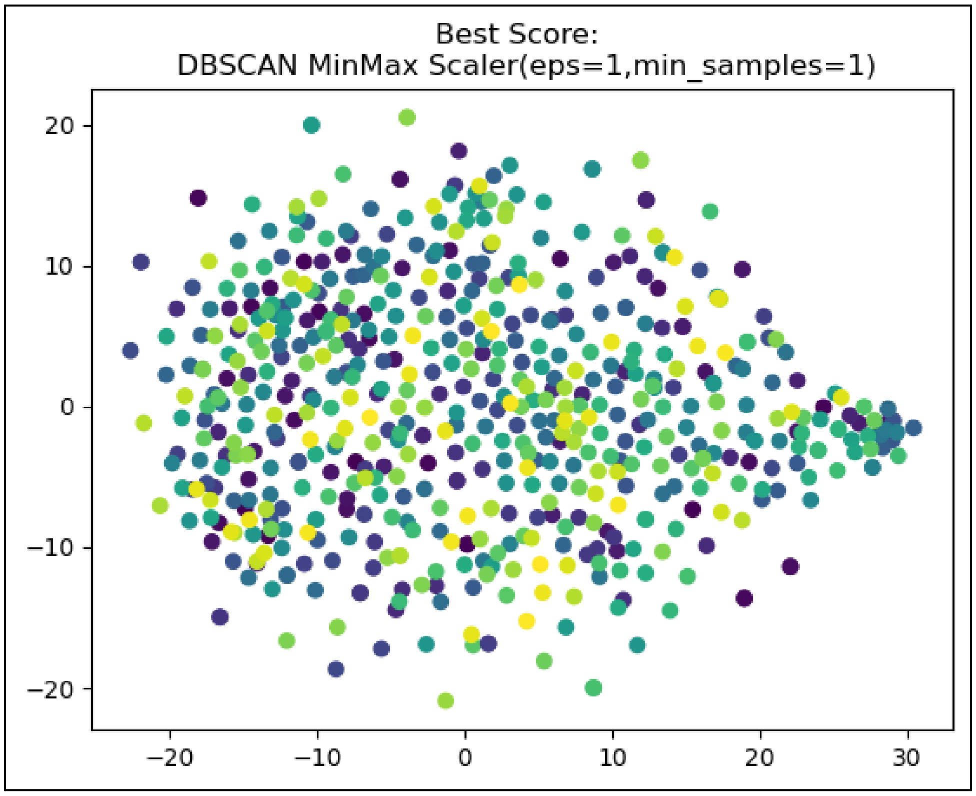


Figure.20

BEST SCORE:



References

[1]. https://www.kaggle.com/datasets/aksha05/flower-image-dataset