Final Project

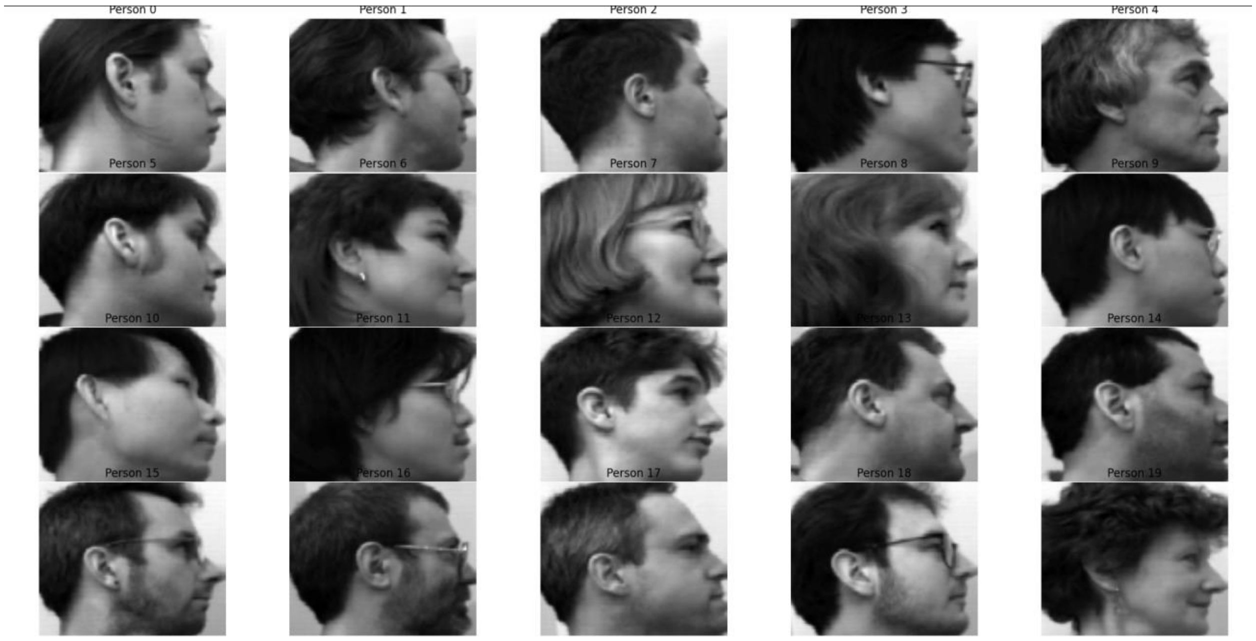
Problem Description:

The goal of this project is to create a classifier that can reliably classify individuals by their face.

Data:

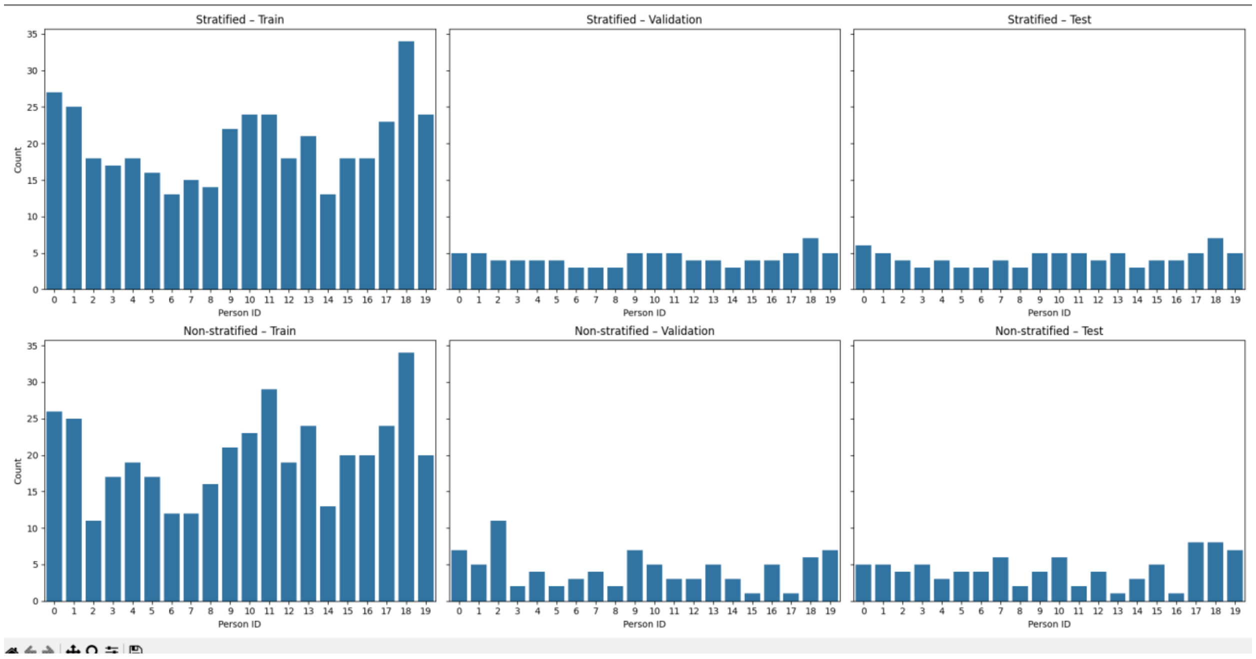
The dataset we are using contains images for 20 different people. The main variable is facedat which contains a 1x20 array where each entry corresponds to 1 person. Each entry is a 3D array of shape (h, w, n_images) where h and w are width and height and n_images is the number of images available for each person. All of these numbers are different between people, so in processing the data we need to resize the images to be all of the same size, and ensure that when we split the data we account for the fact that there are different numbers of images for each person.

Ex image :



Methodology/Code Explanation:

To achieve this, we will be using a combination of clustering algorithms and neural networks trained on a cropped version of the UMIST dataset. We load in the data to a pandas data frame, and ensure that we resize all of the images to ensure that they are all the same size. We use stratified train test split as there are a different number of images for each person, so we want to split each persons images into training testing and validation proportionally to how many images there are of them to make sure we have an even distribution of each person in the datasets. Had we not done this we would have an uneven distribution.

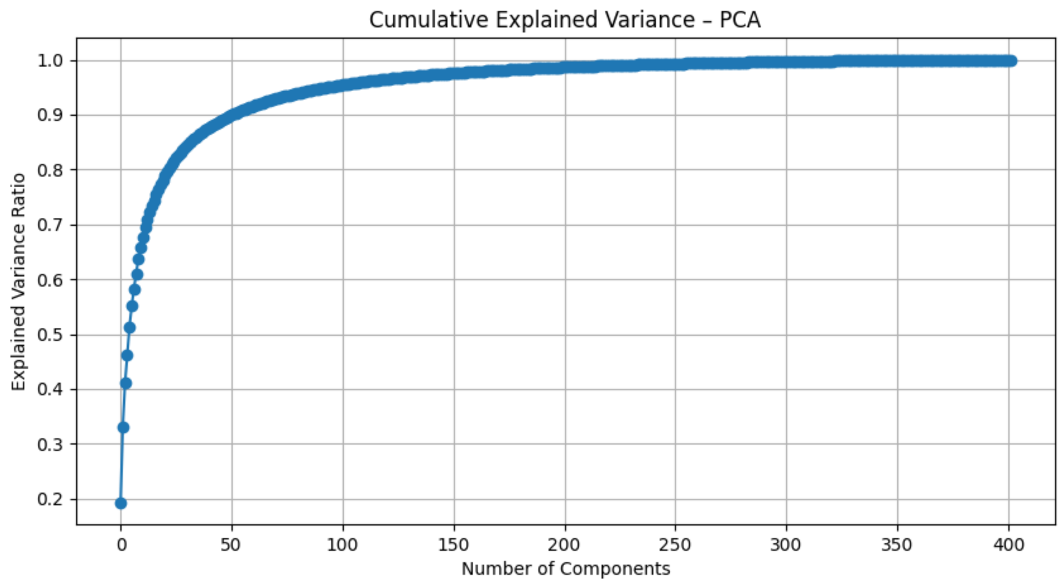


This is the comparison of using stratified vs non-stratified splitting to show how when we do not stratify we don’t get even splitting.

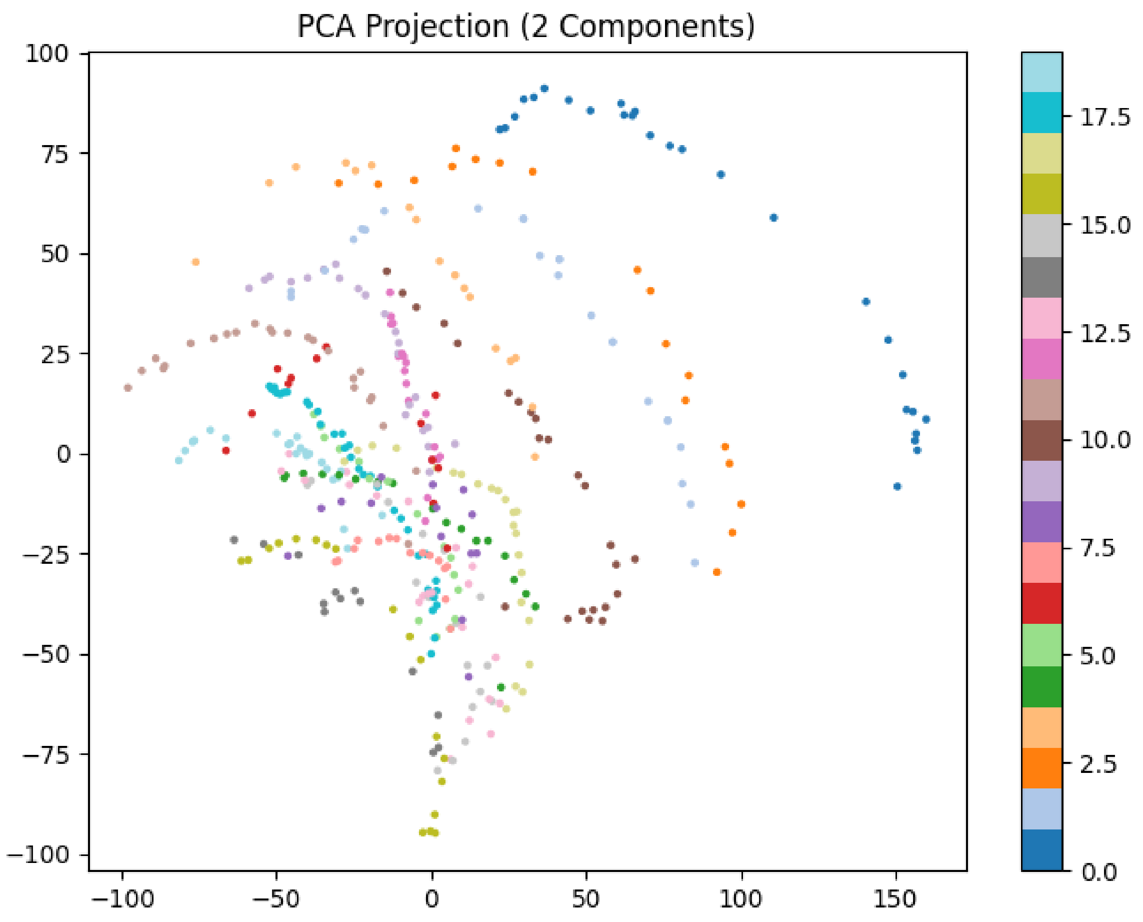
We use a 70/15/15 split as it gives the model enough data for each step without overfitting it or not having enough data to perform reliable tests.

Next we perform a comparison of 4 different dimensionality reduction methods.

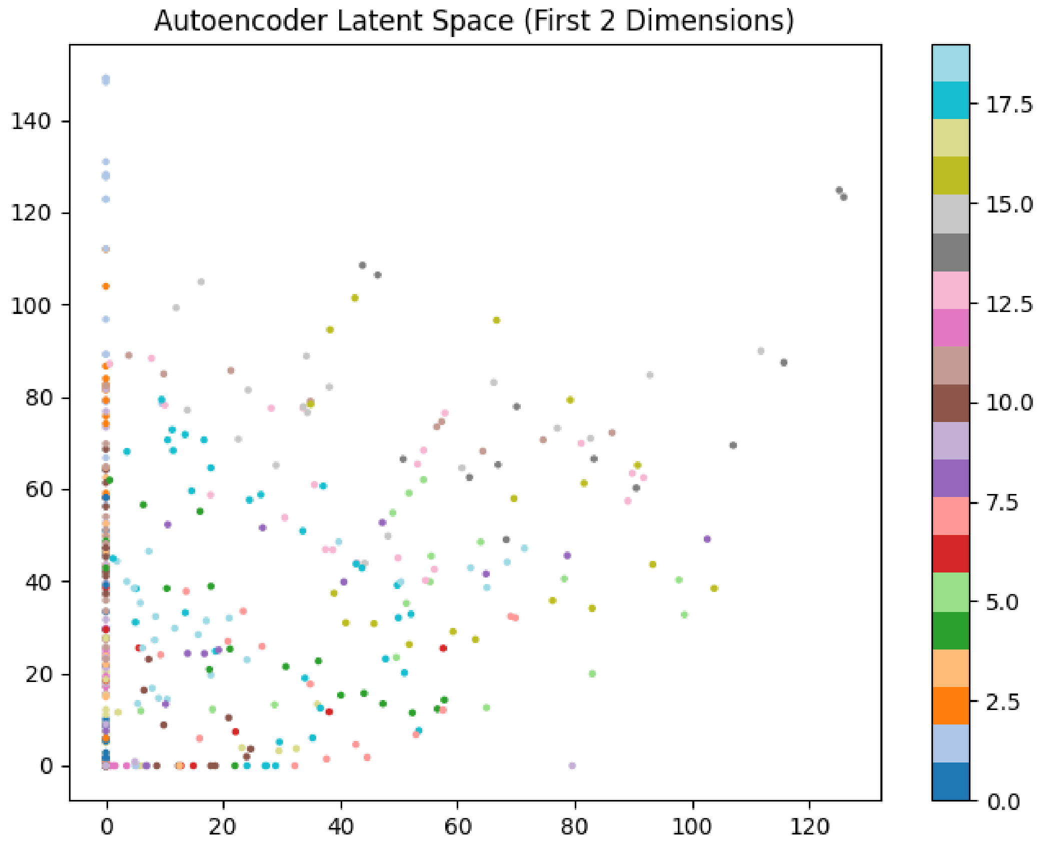
PCA:



This graph shows that 50 components explain around 90% of the variance, and if we want 100% explained variance we would need to use around 200 components.

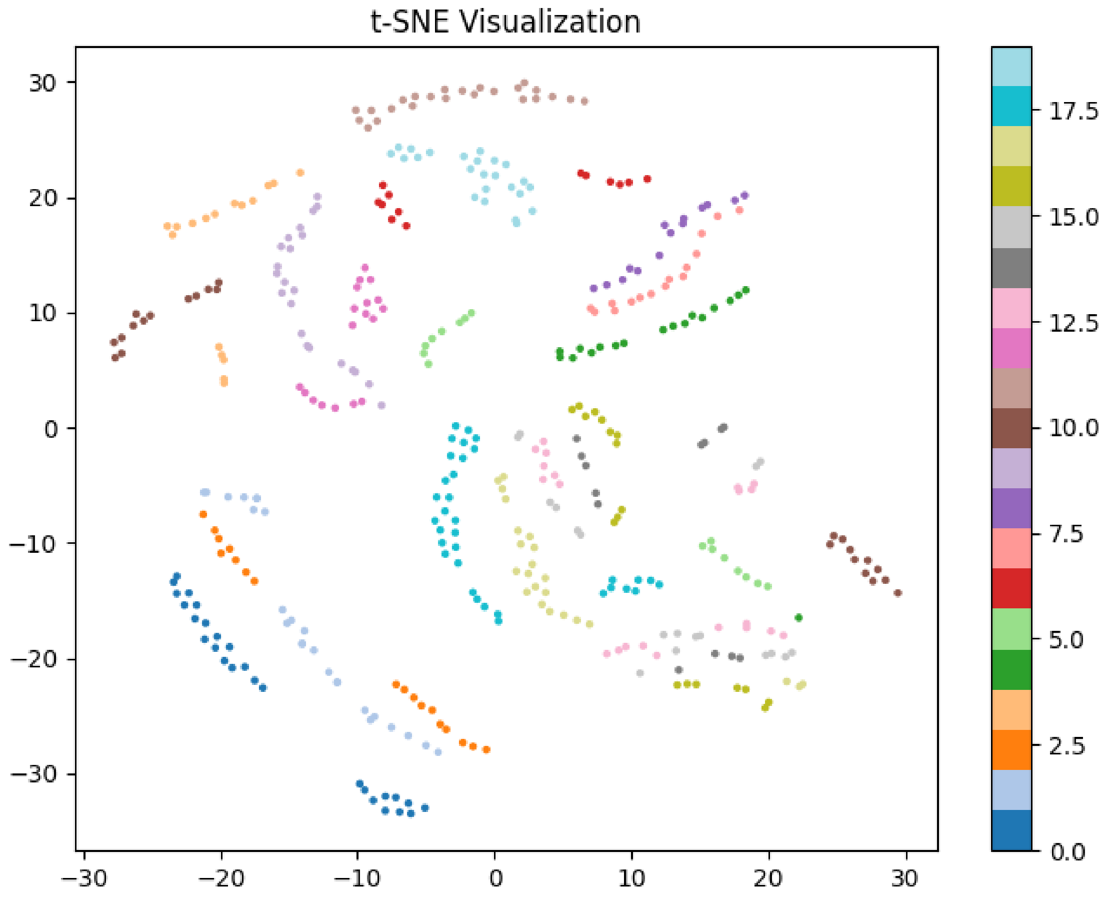


Auto Encoder:



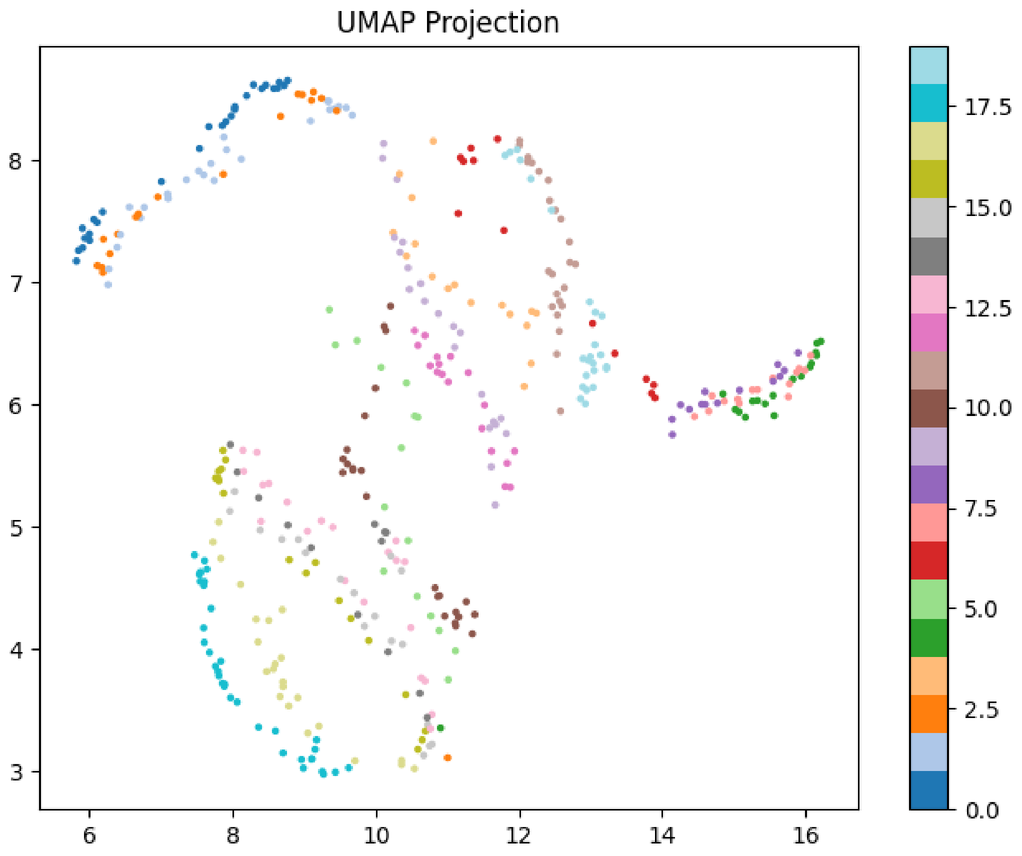
Autoencoder Latent Space (First 2 Dimensions)
0
20
40
60
80
100
120
140
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

t-SNE:



t-SNE Visualization
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0

UMAP:



All of the graphs above show the 2D representation of the compressed data. The different colors each represent the persons pictures, and the clearer they are grouped shows how well the dimensionality technique preserved the difference between people. From these graphs we can see that UMAP and t-SNE did the best at preserving the variance. However, this can be misleading because UMAP and t-SNE are built specifically for visualizing high dimensional data in lower dimensions, but they are not good for classification as they cannot consistently transform the validation and test data. So, despite its graph not showing the variance very well, we will be using the autoencoder as it is the best for capturing non-linear data like we have. We will not be using PCA for this reason as it is not good for capturing non-linear data so it will lose a lot of the finer differences in the images, like changing in lighting and pose.

For the autoencoder, we use ReLU on the hidden layers as it is computationally cheap, and Linear activation on the output layer as our data is not binary, so we want to preserve reconstructions without forcing numbers into a narrow range. We use a latent dimension of size 50 as it provides a good trade off between strong compression and preserving important information for classification. We also use the Adam optimizer to help with training the encoder.

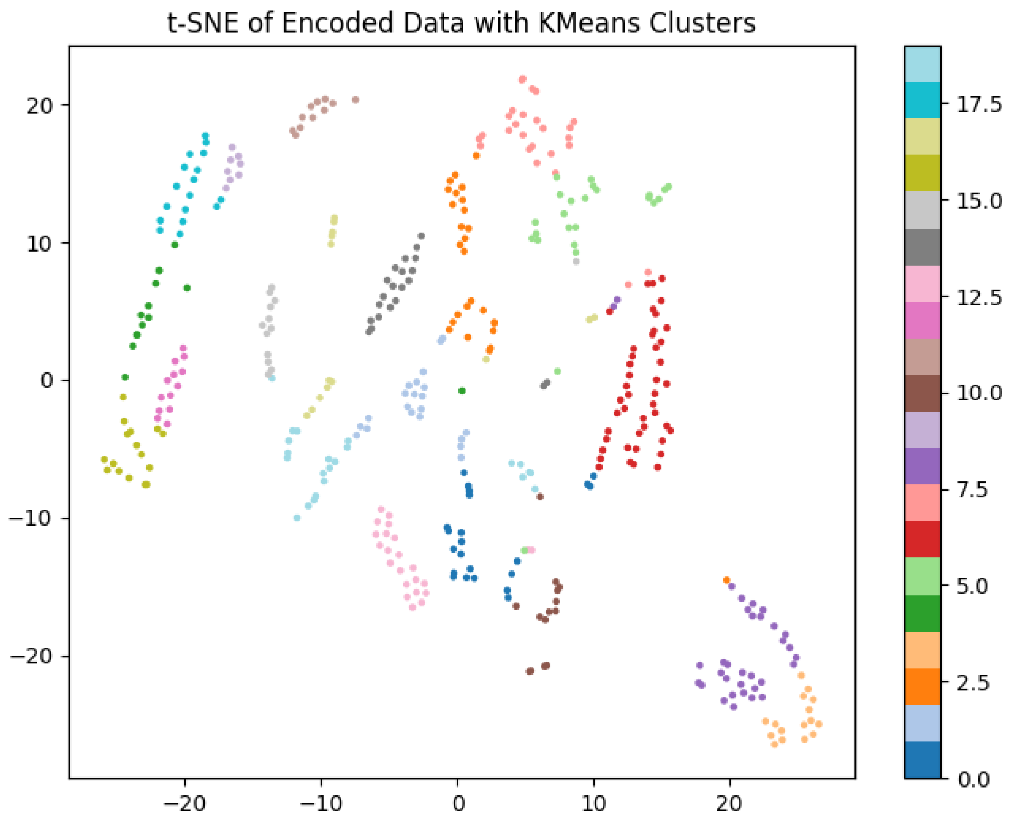
For the clustering, we compare DBSCAN, KMeans and Hierarchical clustering to see which one works best for our dataset. For KMeans and Hierarchical we set the number of clusters to 20, as we have 20 people and we selected the ward method for the Hierarchical algorithm as it is best for creating the most compact clusters. I added in DBSCAN with random hyperparameters just for comparison and to show that it is not good for this dataset as it is not good with clusters with varying densities (we have a different number of images for each person so the clusters would all be different sizes) and it is not good with high dimensional data. Our final purity scores are:

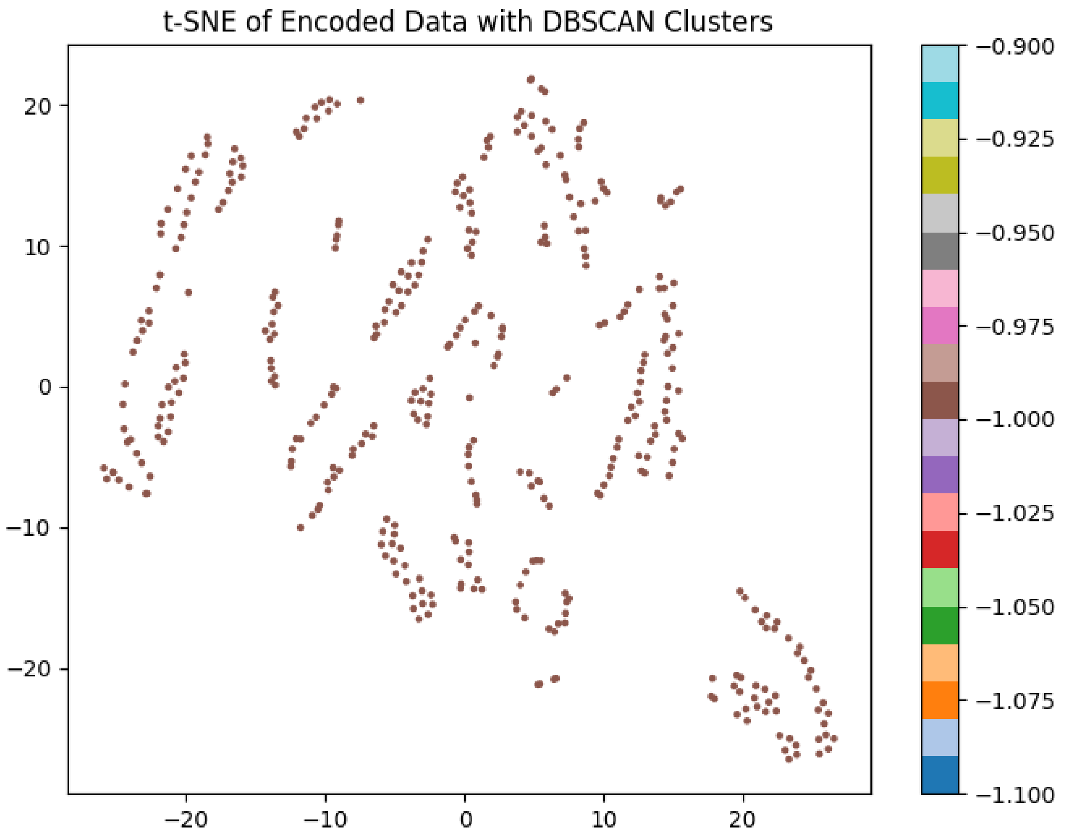
K-Means Purity: 0.5174129353233831

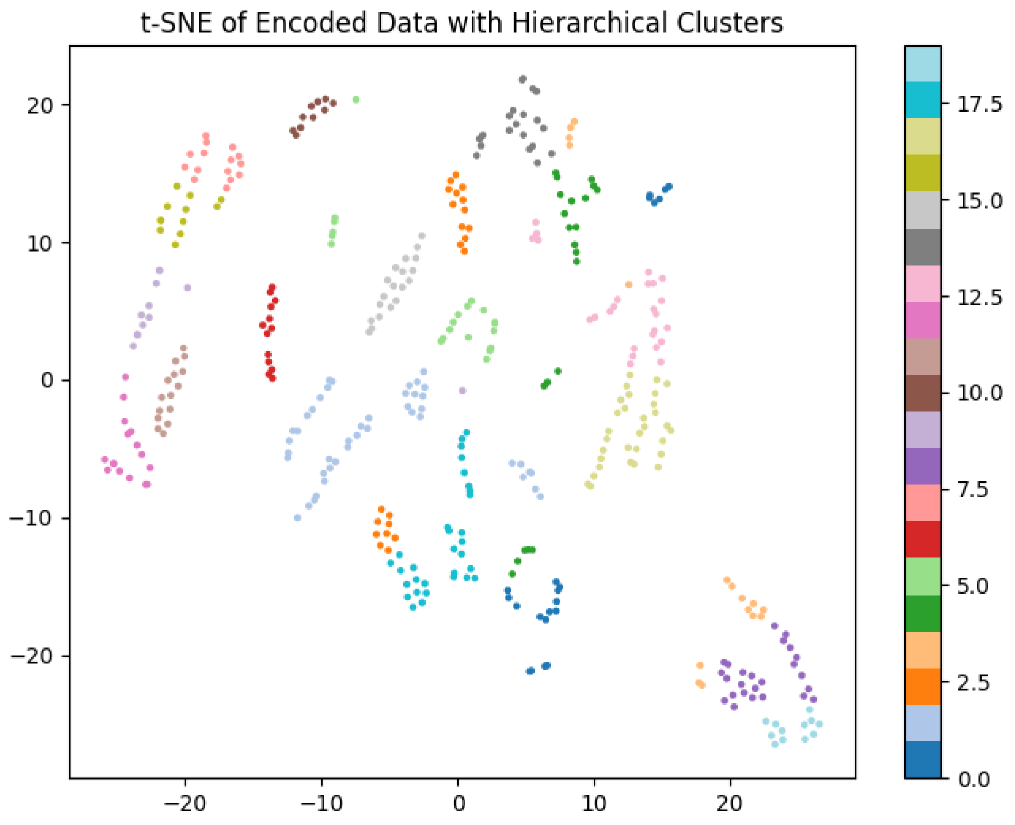
DBSCAN Purity: 0.0845771144278607

Hierarchical Clustering Purity: 0.5771144278606966

And finally we use t-SNE to visualize our clusters:




t-SNE of Encoded Data with DBSCAN Clusters
20
10
0
−10
−20
−20
−10
0
10
20
−0.900
−0.925
−0.950
−0.975
−1.000
−1.025
−1.050
−1.075
−1.100



We can see clearly from the graphs and the purity scores that DBSCAN performs worse and Hierarchical cluster did the best.

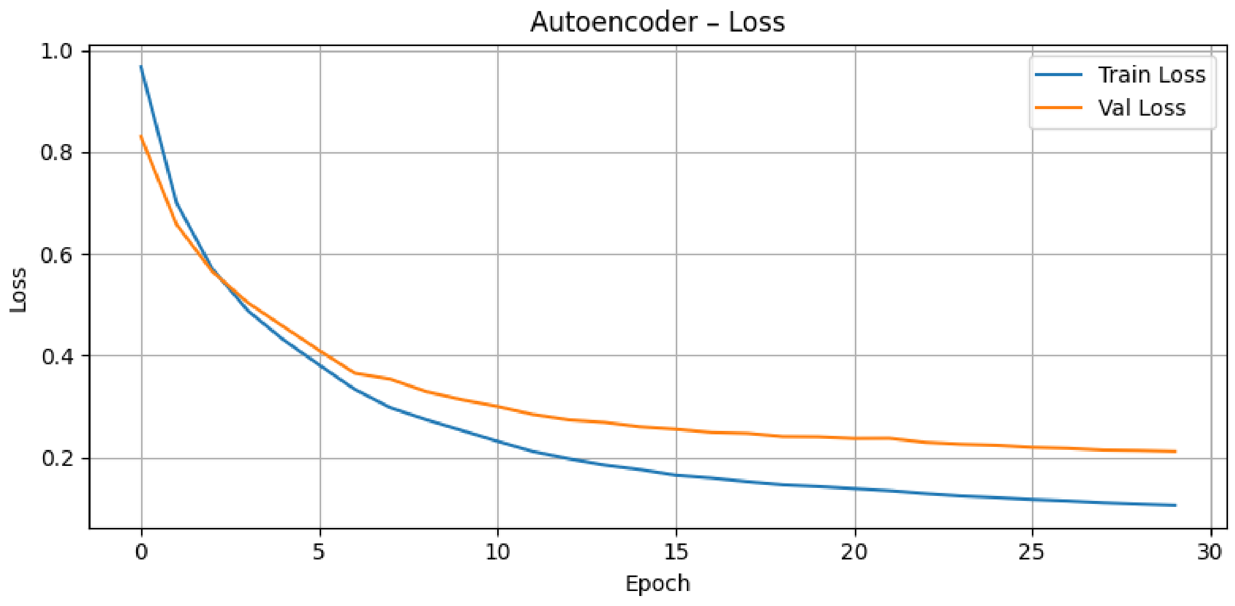
For the neural network we use the Keras sequential API because it is good for building simple classifiers, and because of how we have represented the data, we don’t need to use something like a CNN. The model consists a 50 dimensional input layer, matching the 50 dimensional latent code given by the auto encoder. Then we have 2 hidden layers using ReLU activation because it is computationally cheap and is good at mitigating vanishing gradients, and in between these layers we have a Dropout of 0.2, that helps reduce overfitting. Then on the output layer we have 20 outputs, 1 for each person using Softmax activation which is the standard for classification and outputs normalized class probabilities. I also did not use the clustering as extra features because in my testing I found it actually gave me a lower accuracy, so I think something went wrong with the clustering and it acted more as noise the classifier had to sort through as opposed to extra data to help it. For my optimizer I used Adam with a learning rate of 1x10^-3, which adjusts the learning rate per parameter and helps the model converge faster than regular stochastic gradient descent. We train the classifier on 30 epochs with a batch size of 32 as it provides a good balance between having enough training time and avoiding excessive overfitting. To tune the hyper parameters there was a bit of guess and check to get them right, in combination with previous knowledge from other projects. I chose 128/64 sized hidden layers as they are large enough to find non linear patters, but not too big to avoid over fitting. I chose a dropout of 0.2 because anymore and we risk over regularizing the model and lowering accuracy.

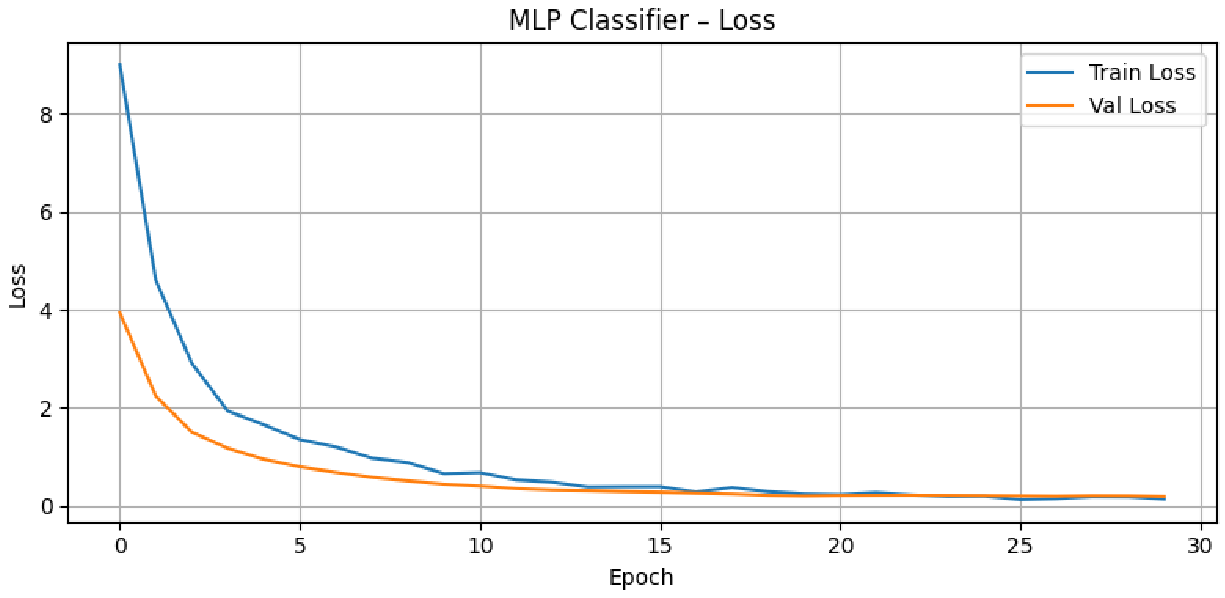
Results:

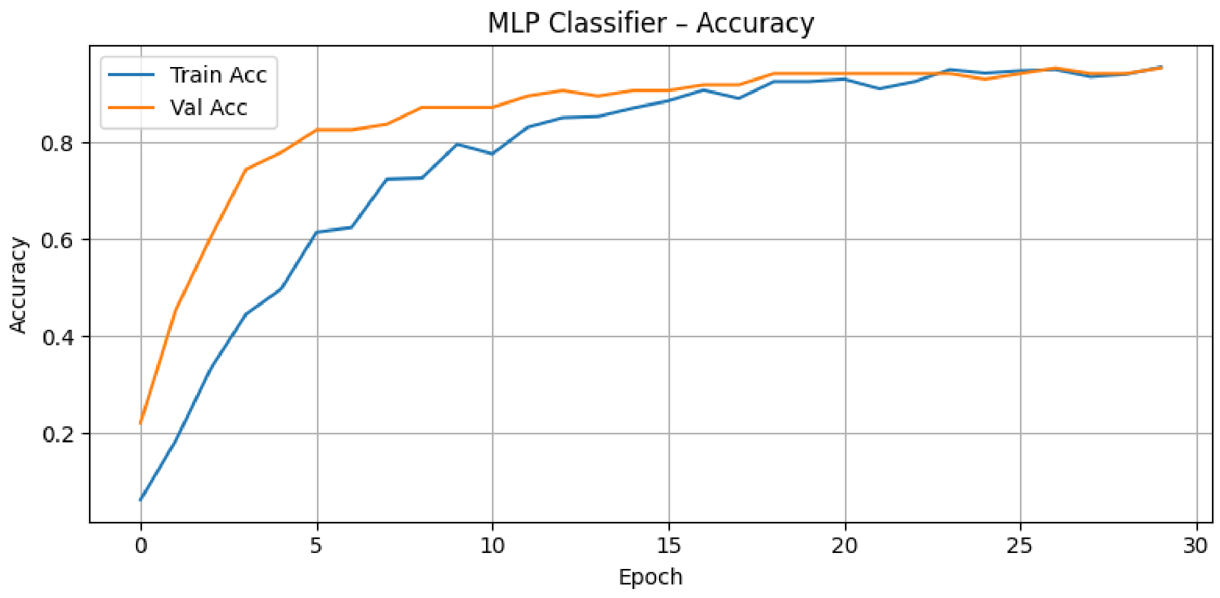
In the end I ended up with a model that had a 95% accuracy. Below are the graphs of the accuracy and loss functions, as well as the confusion matrix and the precision scores for each person.

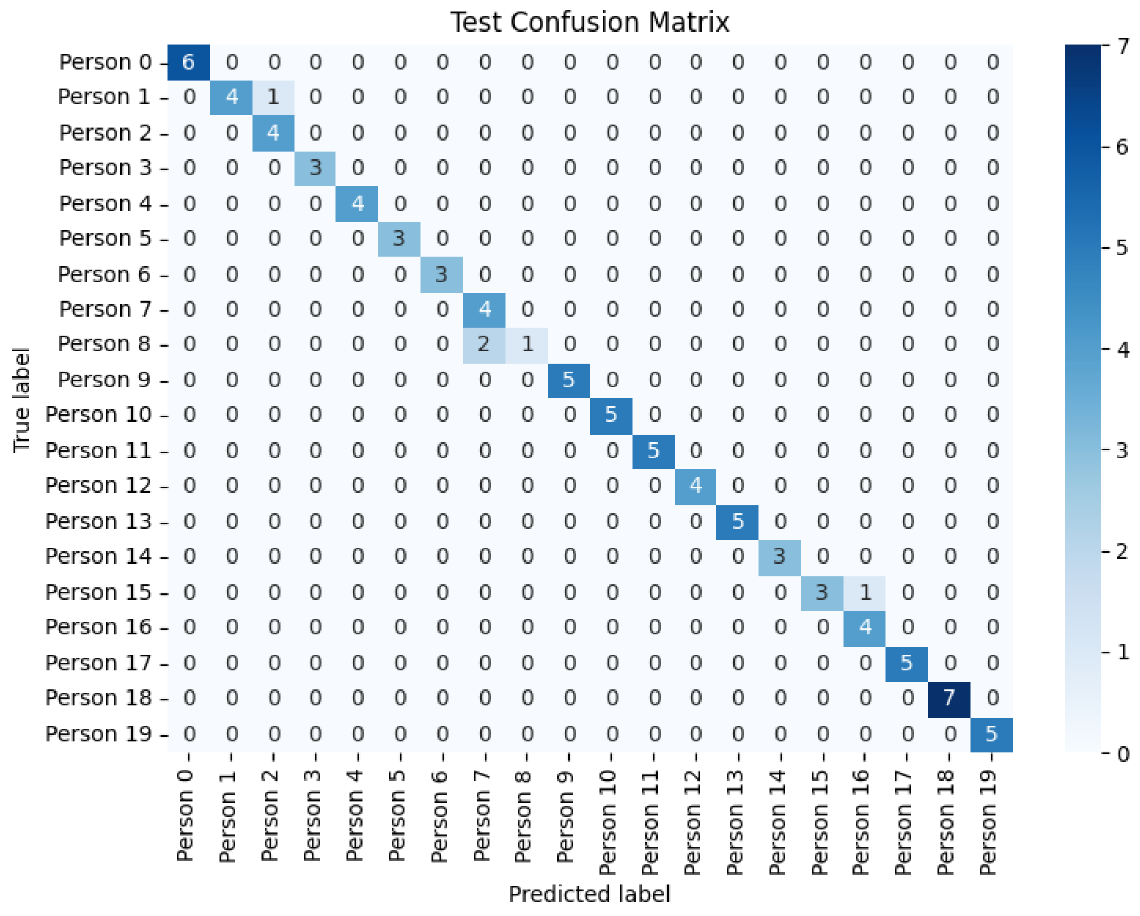
```
      precision   recall  f1-score   support


    0     1.0000    1.0000    1.0000        6
    1     1.0000    0.8000    0.8889        5
    2     0.8000    1.0000    0.8889        4
    3     1.0000    1.0000    1.0000        3
    4     1.0000    1.0000    1.0000        4
    5     1.0000    1.0000    1.0000        3
    6     1.0000    1.0000    1.0000        3
    7     0.6667    1.0000    0.8000        4
    8     1.0000    0.3333    0.5000        3
    9     1.0000    1.0000    1.0000        5
   10     1.0000    1.0000    1.0000        5
   11     1.0000    1.0000    1.0000        5
   12     1.0000    1.0000    1.0000        4
   13     1.0000    1.0000    1.0000        5
   14     1.0000    1.0000    1.0000        3
   15     1.0000    0.7500    0.8571        4
```

```
          16     0.8000    1.0000    0.8889         4
          17     1.0000    1.0000    1.0000         5
          18     1.0000    1.0000    1.0000         7
          19     1.0000    1.0000    1.0000         5

    accuracy                         0.9540        87
   macro avg     0.9633    0.9442    0.9412        87
weighted avg     0.9663    0.9540    0.9504        87
```



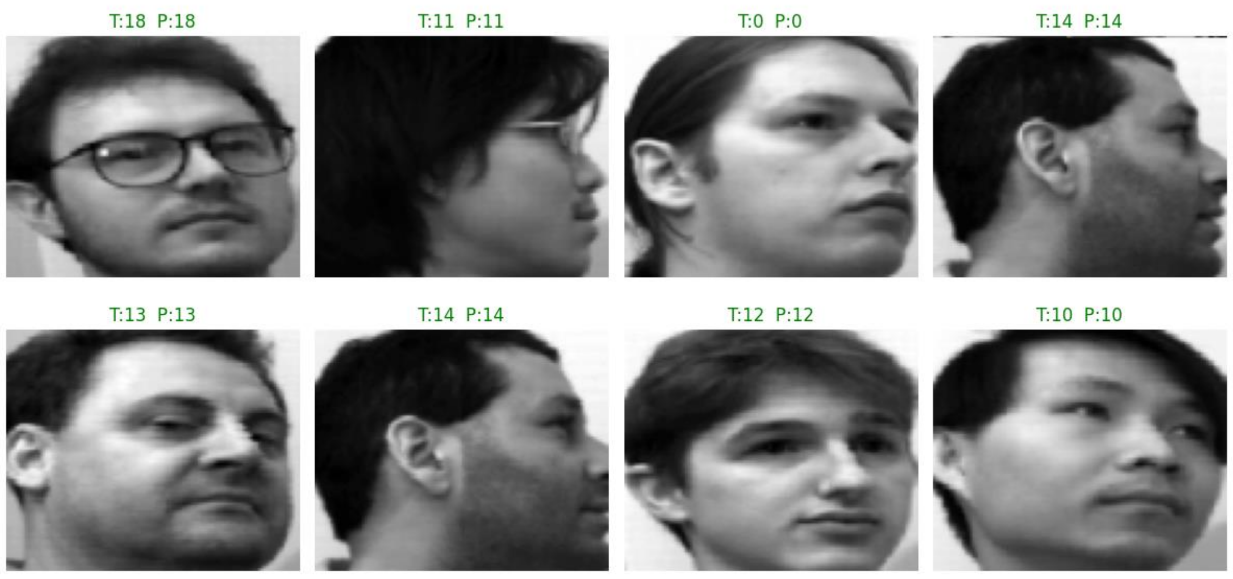

MLP Classifier – Loss
Train Loss
Val Loss
Loss
Epoch
0
2
4
6
8
0
5
10
15
20
25
30


MLP Classifier – Accuracy
Train Acc
Val Acc
Accuracy
Epoch
0.2
0.4
0.6
0.8
0
5
10
15
20
25
30



True label \ Predicted label	Person 0	Person 1	Person 2	Person 3	Person 4	Person 5	Person 6	Person 7	Person 8	Person 9	Person 10	Person 11	Person 12	Person 13	Person 14	Person 15	Person 16	Person 17	Person 18	Person 19
Person 0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 1	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 2	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 3	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 4	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 5	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 6	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
Person 7	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
Person 8	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0
Person 9	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0
Person 10	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0
Person 11	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
Person 12	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
Person 13	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0
Person 14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0
Person 15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	0
Person 16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0
Person 17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
Person 18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
Person 19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5

Here is a visual showing the expected and predicted label for test images



Overall, the network was able to predict the images with a high degree of accuracy.

References:

https://stackoverflow.com/questions/874461/read-mat-files-in-python

https://www.geeksforgeeks.org/machine-learning/umap-uniform-manifold-approximation-and-projection/

https://www.datacamp.com/tutorial/introduction-t-sne

https://medium.com/@adnan.mazraeh1993/a-comprehensive-guide-to-dimensionality-reduction-from-basic-to-super-advanced-techniques-9-3e3f3eeb8814

https://medium.com/@laakhanbukkawar/unlocking-the-power-of-hierarchical-clustering-a-journey-from-basics-to-advanced-concepts-bce8ab72f3d1