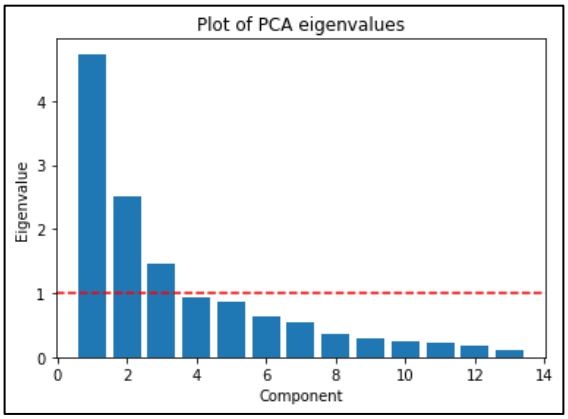
Tomisin Adeyemi - FML HW 5

Preamble

The data was first normalized using the scikit-learn's StandardScaler() class to make it compatible with the different dimensionality reduction methods. This is because the dimension reduction methods assume that the data is normalized.

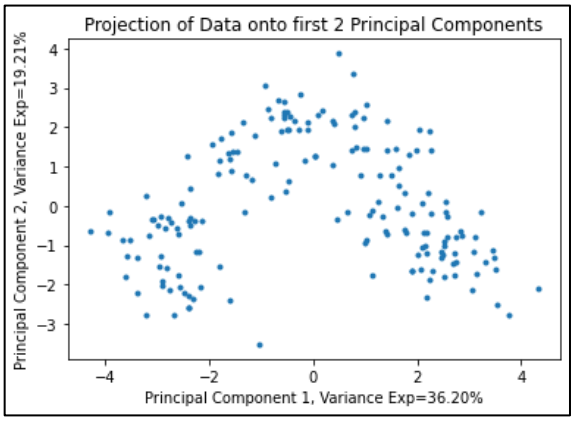
1. Do a PCA on the data. How many Eigenvalues are above 1? Plotting the 2D solution (projecting the data on the first 2 principal components), how much of the variance is explained by these two dimensions, and how would you interpret them?

I created a PCA class object and fitted it to the scaled data, then transformed the scaled data in the new lower dimensions. The eigenvalues explain the amount of variance that is accounted for in the data by a certain principal component, so I used the explain_variance_ attribute of the pca object to extract the eigenvalues. Similarly, the percentage of variance explained by a principal component can be gotten from the pca.explained_variance_ratio attribute. To interpret the first 2 dimensions/principal components, I inspected the pca.components_ attribute and plotted the values to visualize how important each feature was in the Principal Component.



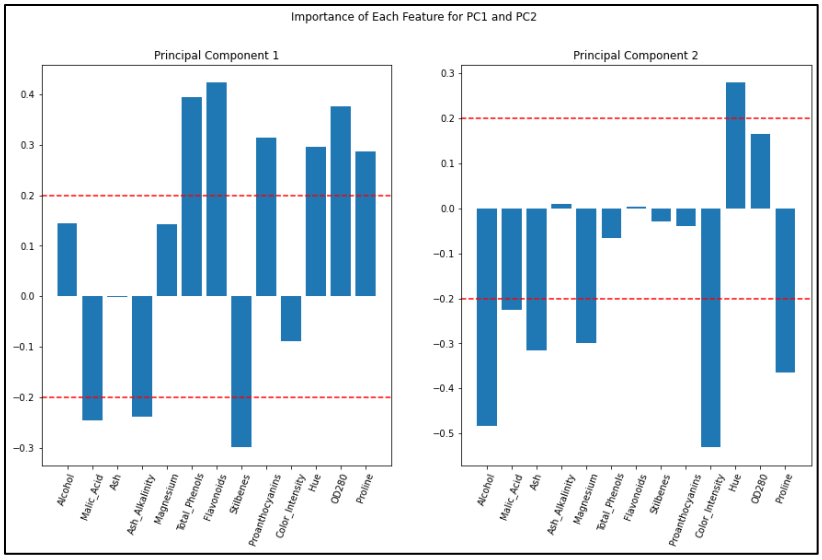
The plot to the left visualized the eigenvalues for each principal component. It shows that there are **3 eigenvalues above 1**. This means that 3 principal components account for more variance than is accounted for by a single feature in the original data.

By this criterion, retaining the first 3 principal components would yield the best results.



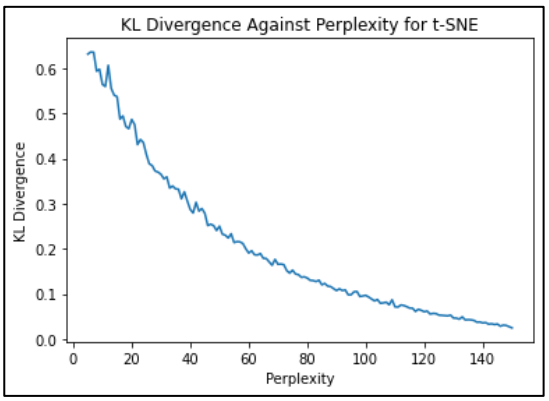
The graph to the right shows the projection of the data onto the first 2 principal components. The total variance explained from these 2 components is **55.41%**, which is not that great. Adding the third principal component (by considering eigenvalues that were above 1) accounts for 66.53% of the variance, which still is not that great. This can also be because components 4 & 5 have eigenvalues that are close to 1.

Finally, the graph below shows how important each feature was in each Principal Component. I used a threshold of 0.2 to determine if a feature was *important*. By this standard, PC1 is a combination of 9 features, and PC2 is a combination of 7. PC1 being most heavily determined by Phenols, Flavonoids and OD280, and PC2 being most heavily influenced by Color Intensity, Proline and Alcohol.



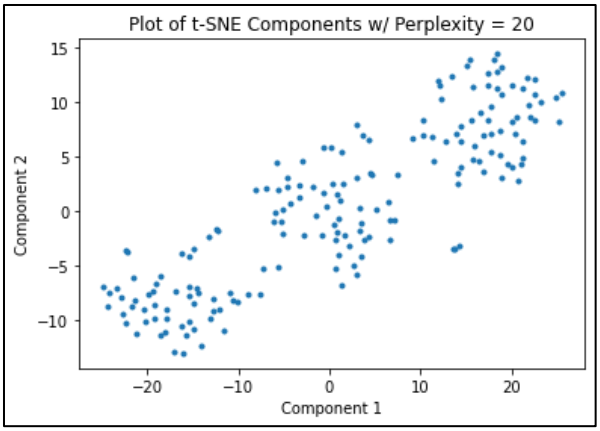
2. Use t-SNE on the data. How does KL-divergence depend on Perplexity (vary Perplexity from 5 to 150)? Make sure to plot this relationship. Also, show a plot of the 2D component with a Perplexity of 20.

First, I fit and transformed a TSNE object with a perplexity of 20 on the scaled data. Then I calculated the KL Divergence of 146 different TSNE models, varying the perplexity from 5 to 150. I then stored the values in a dataframe and plotted them on a graph.

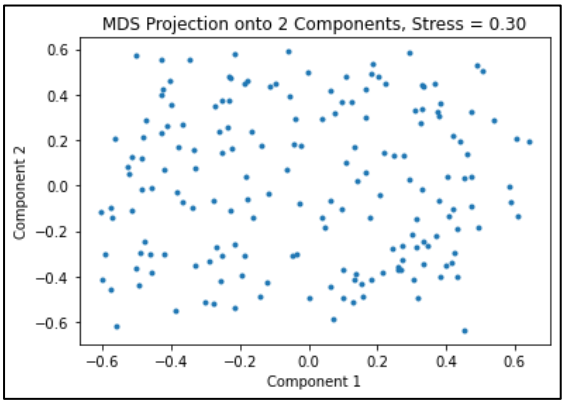


The graph to the right shows the change in KL divergence as the perplexity increased. From the graph, there seems to be an inverse exponential relationship between perplexity and KL Divergence: as the perplexity increases, the KL Divergence reduces. This is not surprising: KL Divergence serves as the loss function of t-SNE, and measures how different one probability distribution is from another. In the context of t-SNE, it measures the difference between high dimensional affinity and low dimensional affinity. Perplexity describes the number of neighbors with non-zero affinities to consider. As perplexity increases, the gaps between clusters get increasingly large: dissimilar clusters in the high dimensional space are more dissimilar in the lower dimensional space. Thus, the KL divergence reduces. Note that this is not necessarily a good thing, as increasing the number of neighbors to consider can group non-related points together.

The graph below shows a plot of the 2D components of t-SNE with a perplexity of 20. It seems like there is a clear separation between the classes, of which there are 3.



3. Use MDS on the data. Try a 2-dimensional embedding. What is the resulting stress of this embedding? Also, plot this solution and comment on how it compares to t-SNE.



I created an MDS object of the scikit-learn class with 2 components, set normalized stress to auto and the metric parameter set to False. I set the metric parameter to False because the formula for the MDS stress function learnt in class matches the formula for the non-metric version of MDS, as shown here. In addition, I opted to normalize the stress to make the stress score more interpretable.

The unnormalized stress score was around ~22k, which is a bit uninterpretable; the normalized stress score is **0.30**, which (by Scikit-learn's documentation) indicates that the MDS is a very poor fit to the data. This is emphasized in the plot shown on the right.

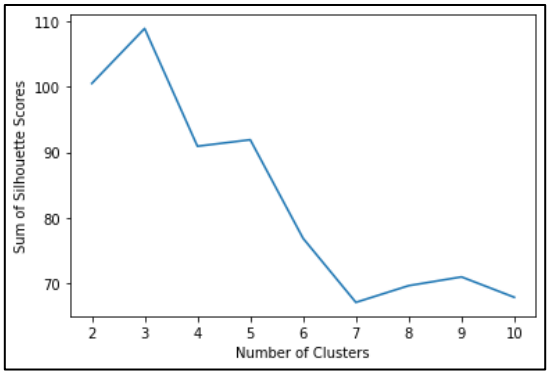
There is no clear pattern in the data points, which explains why the stress score is so poor. The MDS plot is a lot harder to decipher than the t-SNE plot, meaning that t-SNE is a better dimension reduction method than MDS for this dataset.

The poor performance of MDS shows that there is no configuration of wine properties in a 2D space that preserves the distances in 13 dimensions sufficiently.

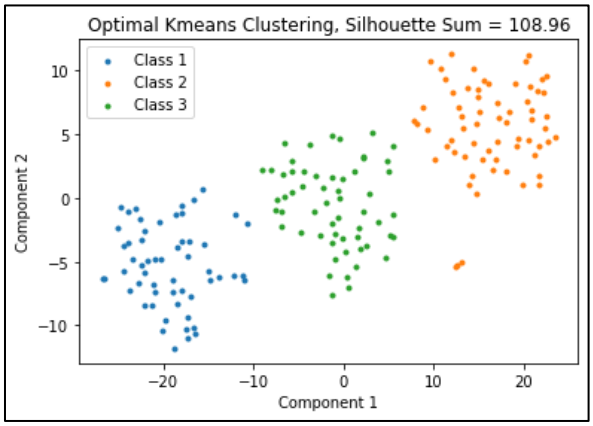
4. Building on one of the dimensionality reduction methods above that yielded a 2D solution (1-3, your choice), use the Silhouette method to determine the optimal number of clusters and then use kMeans with that number (k) to produce a plot that represents each wine as a dot in a 2D space in the color of its cluster. What is the total sum of the distance of all points to their respective clusters centers, of this solution?

I decided to build on the t-SNE method because the separation seemed to be the clearest amongst all the methods attempted. I then tried k values between 2 and 10 and used the sum of the silhouette coefficient scores to determine the best value of k. A single silhouette coefficient score calculates the average distance of a data point to all points in the nearest cluster. The closer this value is to 1, the

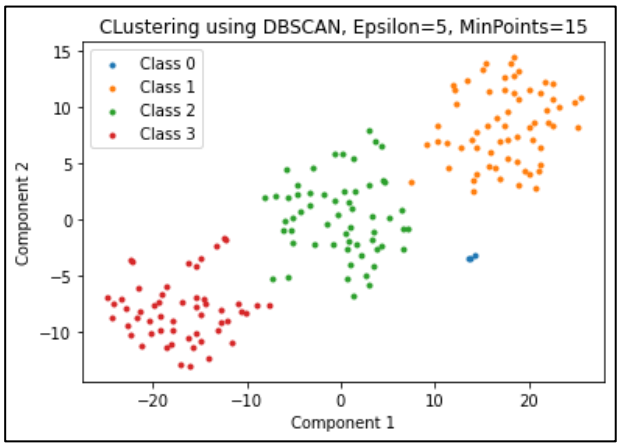
better. Thus, the sum of the silhouette coefficient scores describes unambiguous the clusters are; the higher the score, the better. The graph on the right shows the sum of the silhouette scores for each number of clusters. The highest score, **108.96**, was achieved with k = 3 clusters. Thus, the total sum of the distance of all points to their respective clusters centers here is **108.96**.



The graph below shows Kmeans clustering with the optimal k value, 3.



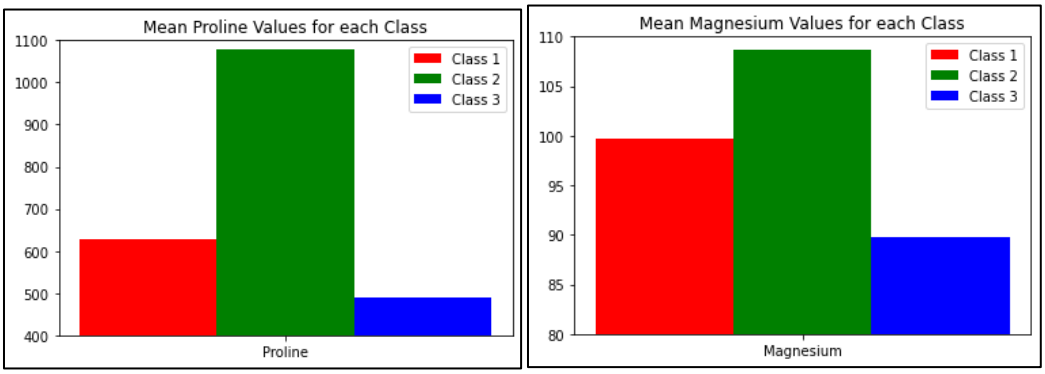
5. Building on one of the dimensionality reduction methods above that yielded a 2D solution (1-3, your choice), use dBScan to produce a plot that represents each wine as a dot in a 2D space in the color of its cluster. Make sure to suitably pick the radius of the perimeter ("epsilon") and the minimal number of points within the perimeter to form a cluster ("minPoints") and comment on your choice of these two hyperparameters.

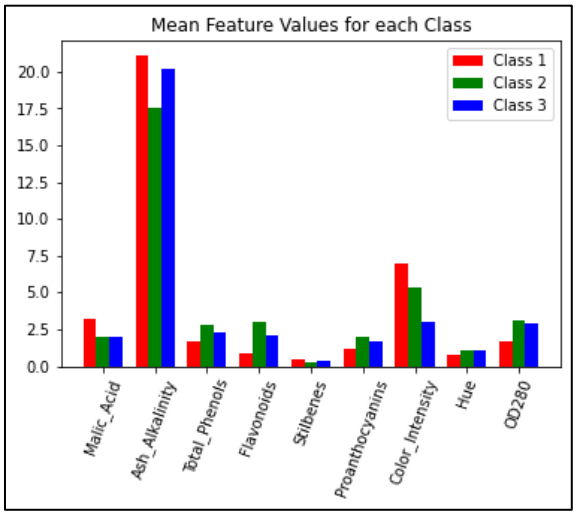


First, I created a DBSCAN object, then I fitted it to the t-SNE data. I then tried different epsilon and minPoints (represented as min_samples in sklearn) and picked the combination of values that resulted in the most visually appealing graph. As expected, the determination of epsilon and minpoints is very arbitrary, a very real problem in DBscan. In addition, small changes in the epsilon and min_points values made/break cluster boundaries, showing that DBScan relies on sharp density drops to identify clusters. The figure on the right shows the dBScan plot, I used an epsilon of 5 and 15 minpoints. The plot shows 4 different classes as opposed to the optimal 3 shown in K Means, with the fourth class showing a smalle number of outliers. This also shows DBScan's sensitivity to outliers.

Extra credit: a) Given your answers to all of these questions taken together, how many different kinds of wine do you think there are and how do they differ?

Based on questions 4 and 5 (ignoring the outliers here), the number of wines is 3. To inspect the properties of these different kinds of wine, I extracted the indices corresponding to each Class in the KMeans labels. Then I extracted those datapoints from the original wine data. After, I compared the mean of each feature in each of the classes. The graph below shows the results:





Class 2 has higher mean proline and magnesium values than 1 and 3, and 1 has higher mean proline and magnesium values than 3. Class 1 has higher color intensities than 2 and 3, and 2 has higher intensities than 3. The rest of the differences are visualized in the graph.