S1 Principles of Data Science Coursework Report

CRSiD: tmb76

University of Cambridge

Section A

Question 1

(a)

To explore the data, it needs to be visualised in some way. This dataset being high-dimensional with 500 dimensions in the feature-space, makes visualisation tough. Thus, each feature is visualised separately with density plot. Densities of the first 20 features are obtained and shown below (Figure 1). The main observation is that some features follow very similar distributions. A rough assumption is thus that there are overall 34 groups of highly correlated features in the entire dataset. The most common one has a very strong peak at 0, with a much smaller one at around 1. Another is bimodal, with peaks at 0 and 3. And the last one is also bimodal, with peaks at 0 and a stonger one at approximately 4. If the features can be grouped then one could reduce the dimensionality of the feature-space to 1 representative feature from each group.

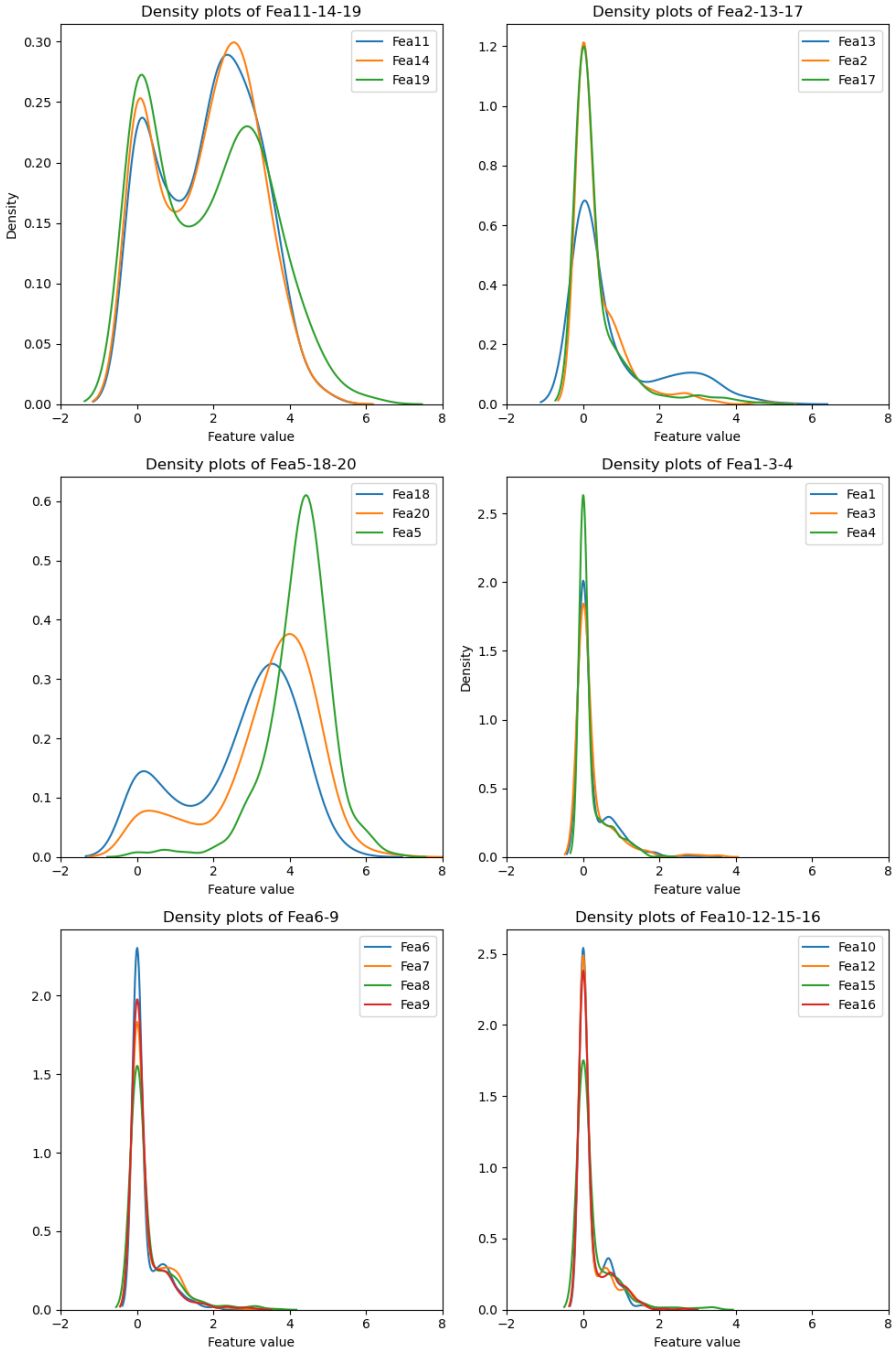

Figure 1: **Density plots of the first 20 features, grouped by similarity**. The x-axis was set to the same scale for all plots, for easier comparison. The y-axis is however different for all plots.

(b)

Thus, Principle Component Analysis (PCA) is conducted on the dataset. PCA enables one to derive a lower-dimensional set of features from the full **A** dataset. This is done by finding the direction in which the observations have the greatest variance, for the full feature space [1, pp. 255-257]. This is done on the entire dataset, not just the first 20 features. Using `scikit-learn`'s PCA function, getting the first 2 Principles Components (PC).

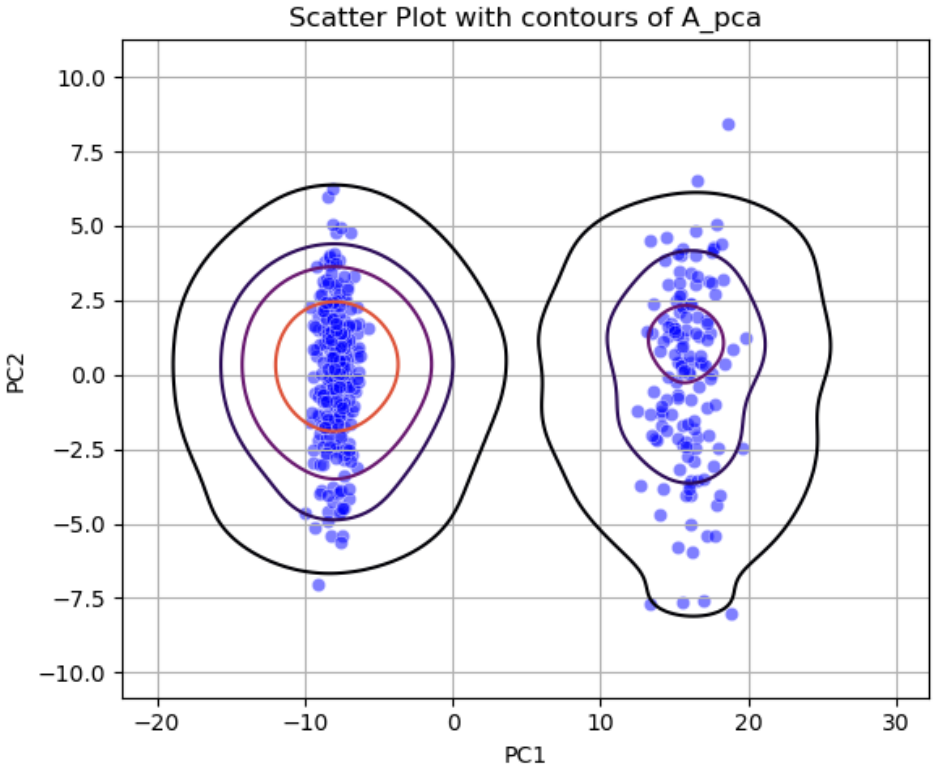

Figure 2: **Scatter plot of the dataset after PCA.** Density contours are also shown.

From Figure 2, the bimodal separation seen in some of the first 20 features is seen again in the 1^{st} PC. The 2^{nd} PC also resembles the more commonly observed feature in Figure 1, with a very strong peak and a much weaker one at a slightly higher value, more from the right-hand side group. This is hard to see in the scatter plot, but the density contours do show the negative skewness of the data's distribution for that PC. It is clear that the PCA transformed data is linearly separable into 2 groups.

(c)

From this, clustering can be run to try and see how they would compare to the distribution observed. The `scikit-learn` KMeans clustering algorithm was used, with all parameters set to their default values, apart from the random state. The default number of clusters is 8 [4]. Now from Figure 1 and Figure 2, there likely are not 8 clusters, but K-means will find 8. To assess if "default" K-means clustering performs adequatly, the data is split into 2. For both these splits, clustering is run and then applied to the other split. The reason this can be done is that K-means clustering's main result is the position of points called centroids, which are the geometrical centres of each clusters (when defining the centroids from the feature means). In terms of their significance they are each the most representative points of each cluster [2, p. 243]. One of the `scikit-learn` K-means output is those `cluster_centers_`. Then when using the `.predict` method after fitting the model, i.e. clustering one half of the data, the other half can be clustered base on the distance of each point to the centroids. This is done for both splits of the data and the results are shown in the tables below:

Clusters	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	72	0	0	0	0	0	0
2	1	0	1	0	4	1	2	2
3	0	0	2	0	0	0	0	0
4	2	0	6	0	6	8	14	11
5	1	0	1	1	2	1	1	0
6	0	65	0	0	0	0	0	0

Table 1: Contingency table for the 2 clusterings on the first split of the data.

Clusters	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	0	81	0	0	0	0	54	0
2	0	0	0	3	0	0	0	0
4	1	0	5	1	3	3	0	1
5	0	0	1	0	2	1	0	1
6	0	0	7	5	14	9	0	2
7	0	0	3	0	2	2	0	3

Table 2: Contingency table for the 2 clusterings on the second split of the data.

(d)

As can be seen, some clusters are empty. This can be explained by the initialization of the centroids. If the centroid happened to have no points closer to it than other centroids at the start, it then stays where it is for the whole clustering process [4]. Overall, the fact that a too large cluster number was selected for the model means there is little agreement between clusters (Table 1). Note the cluster numbers are assigned "randomly" and so one could have the exact same clusters but one is cluster 4 and the other cluster 7. Even considering this, we'd expect to have 8 large numbers, each on a different row and column. Seeing as there are numerous small groups of agreeing clusterings, this is not the case and the clustering is not very stable. Stability here refers to how constant the clusters are when clustering the same or very similar datasets.

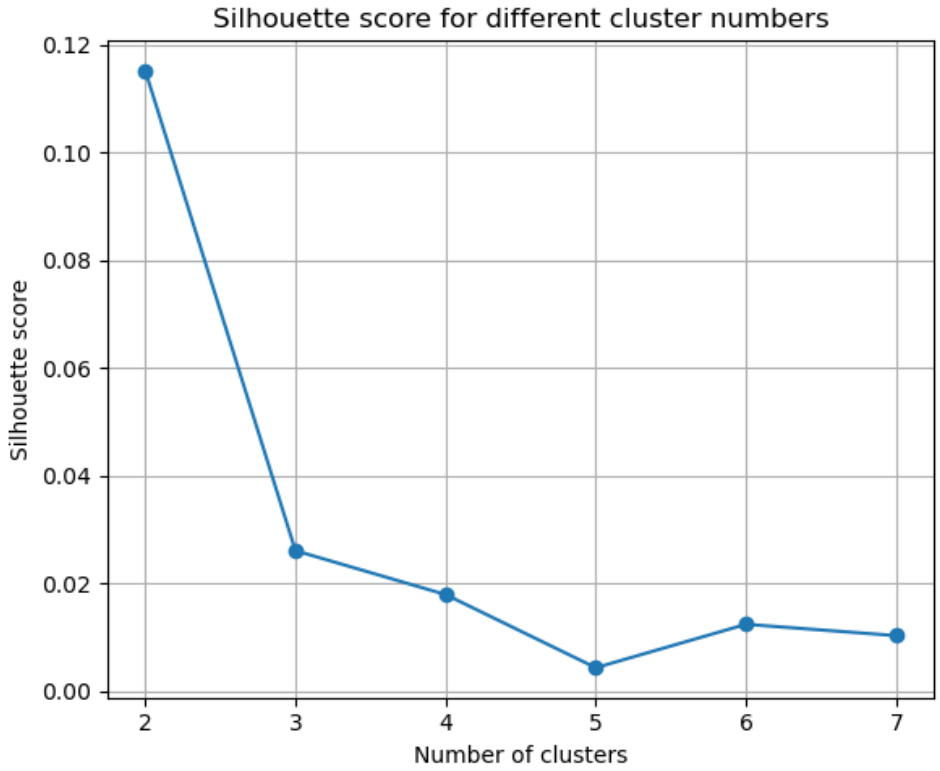

Figure 3: **Silhouette scores for a few different number of clusters**

Thus, clustering is done again for a lower number of clusters. Based on the visualisation of the data, Fig 2, and Figure 3 which shows the silhouette score (a measure of how well the data is clustered) [2, pp. 247-250], a number of 2 clusters is chosen. And the following contingency table is obtained:

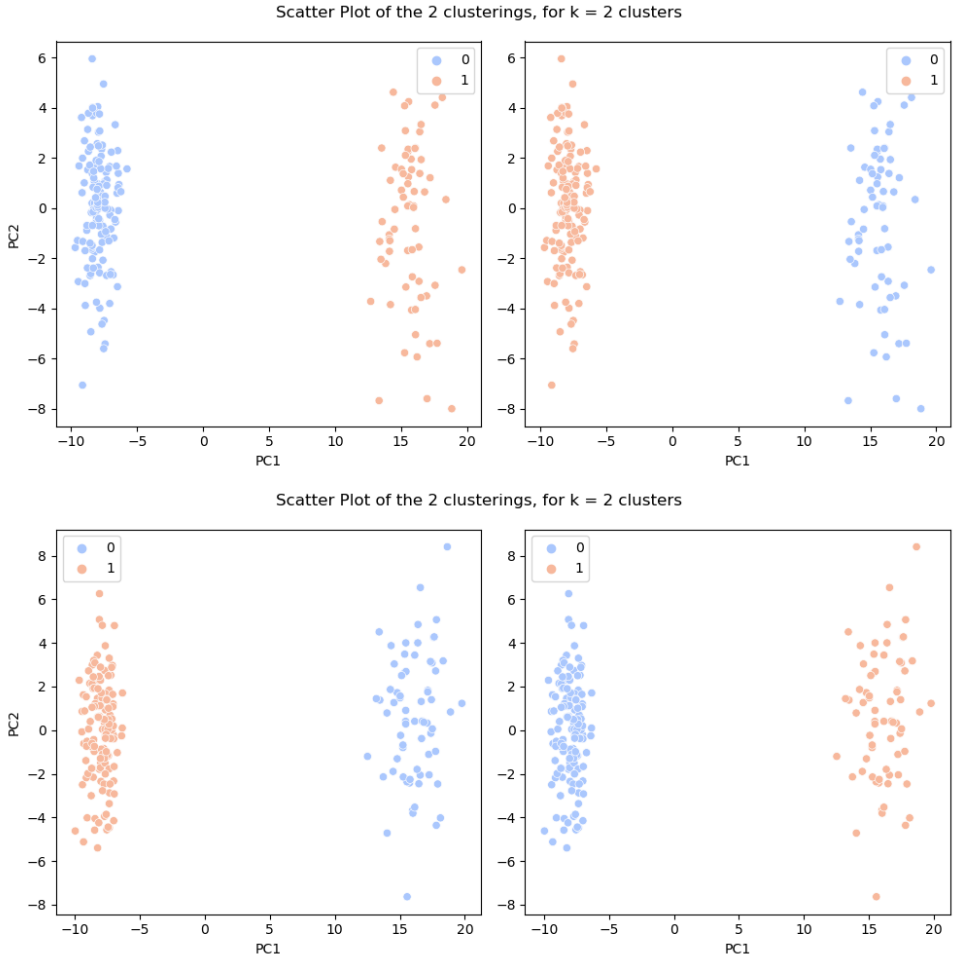

Figure 4: **Scatter plot of the 2 clusterings** The colors indicate the cluster membership.

Clusters	0	1
0	0	135
1	69	0

Table 3: **Contingency table comparing both clusterings for k=2 clusters**

(e)

Here, with the 2 clusters, both models agree on how to separate the data, as can be seen in Figure 4 and Table 3.

Conducting PCA before and after clustering is useful in both cases. One gives a clue as to how many clusters one should use in the K-means training and the other enables one to visualise the clustering results. One additional thing is that the K-means could have been done on the PCA reduced dataset, which would have been a 2D dataset. The impact on performance in that case is not completely clear, but usually in a high dimensionality case, PCA helps avoid the Curse of Dimensionality. This Curse is data becoming very sparse as the number of dimensions increases [2, p. 217]. This means distance measurements like the ones in K-means can run into some difficulty. And PCA transforming the data can help avoid some of those issues.

Question 2

The same dataset as in A is given, but with duplicated observations and missing labels introduced.

(a)

First, the frequency of the labels is summarised in the table below (Table 4):

Label	Frequency
1.0	179
2.0	157
4.0	72

Table 4: Frequency of the labels in the **B** dataset.

(b)

Taking only the feature columns, the following rows were identified as duplicates:

Samples: [27, 29, 43, 45, 65, 73, 82, 99, 100, 106, 116, 118, 119, 145, 146, 165, 172, 174, 187, 192, 197, 209, 219, 248, 252, 259, 290, 296, 304, 310, 343, 350, 351, 358, 381, 383, 388, 395, 408, 423]

Table 5: List of the 40 duplicated rows.

One way to address the dupicates is to use a supervised learning method to train a model on the non-duplicated rows' labels and then predict them for the duplicated rows. Once we obtain these label predictions, we compare them with the duplicates' labels and see which ones match. If one does, then we keep that row and discard the other. If both match, we keep only one of them to avoid having duplicates, even correctly labelled ones. And if none match, we drop both rows.

Using this method, the following rows were dropped:

Samples: [29, 82, 100, 116, 118, 146, 172, 209, 219, 259, 290, 296, 350, 358, 381, 395]

Table 6: List of mislabelled dropped rows

Leaving 16 duplicated rows but correctly labelled, meaning half of which need to be dropped. The following samples were dropped: [106, 145, 192, 248, 252, 310, 388, 423].

(c)

Missing labels cases are sometimes referred to as a semi-supervised learning case [1, p. 24]. Here are 2 ways observations with missing labels can be dealt with. The first is model based imputation. Similarly to the previous question, this relies on training a machine learning model on a complete version of the dataset (sometimes a subset), and then predicting the labels for the observations with missing labels. The choice of model here is broad since the total number of missing values is not too great, and multiple models would be appropriate. This method takes into account relationships between features though runs the risk of an invalid prediction and therefore affecting the result of any model training conducted subsequently. Another more brute-force approach is a simple removal of those observations. This has the advantage of simplicity though brings loss of information, and becomes inappropriate in a scenario where many observations have missing labels. For that second method especially, it is important to consider why the labels are missing. They can be Missing At Random (MAR), meaning there is no underlying reason for that value missing. On the other hand they could be Missing Not At Random (MNAR), meaning that the data missing, sometimes combined with the values of the features for that observation, carry an important meaning. In that case, dropping these observations would be an important loss of information [1, pp. 515-516].

(d)

Here, a model based imputation method is used, assuming the labels are MAR. A multinomial logistic regression is trained on the data without the rows with missing labels. The 20 rows identified to have missing values are dropped, and the model is trained on the remaining data. Testing returns an accuracy of 0.847. The model is then used to predict the labels for the rows missing them. We obtain new frequencies for the labels:

Label	**Frequency**
1.0	187
2.0	162
4.0	64

Table 7: Frequency of the labels in the **B** dataset, after missing label prediction.

Table 7 shows a slight tendency to predict labels 1.0 and 2.0 compared to 4.0, though the observations with missing values do not have to follow the original proportions, since we assumed the values were MAR.

Question 3

(a)

In this dataset, values in various observations and features are missing. 5 observations have missing values. These are samples: [137, 142, 230, 262, 388]. For each of these rows, the following features are missing:

Missing Features: [Fea58, Fea142, Fea150, Fea233, Fea269, Fea299, Fea339, Fea355, Fea458, Fea466, Fea491]

(b)

Again, one can deal with these missing values in a number of ways. One is to use model-based imputation, as in Q2. An example of that is using a k-Nearest Neighbour imputing method. The model looks at the k closest observations to the one with missing values and takes the mean of the values for that feature. Again, this captures relationships between the features, and one can select the number k as they wish. However, it does assume a linear relationship. Another is a simple constant value-based imputation. This usually just takes the mean or median value of that feature and imputs it where values are missing for that feature. This is a simple approach where you can avoid dropping values but it ignores any relationship between features and as a result some loss of information occurs. One way to improve this result is by conducting multiple imputations. Choosing an imputation model, one can repeat the imputation, obtaining a mean and uncertainty of their results. Moreover, imputing multiple times means greater adaptability to the data [5].

(c)

Here, a k-Nearest Neighbour imputation is used. As said above, this takes into account feature relationships while not being too computationally expensive. The number of neighbours k is set to 10. This is from training a kNN model in Q2 and found an optimal number of neighbours to be 10. Comparing the distribution within the concerned features before and after imputation, Figure 5 is obtained. As can be seen the distributions show little change, except for small dips at certain points, where the original has lower density than the imputed distribution.

(d)

To detect outliers, one can calculate the z-scores of each features by standardizing the data. This is done by shifitng/subtracting feature values by their mean and

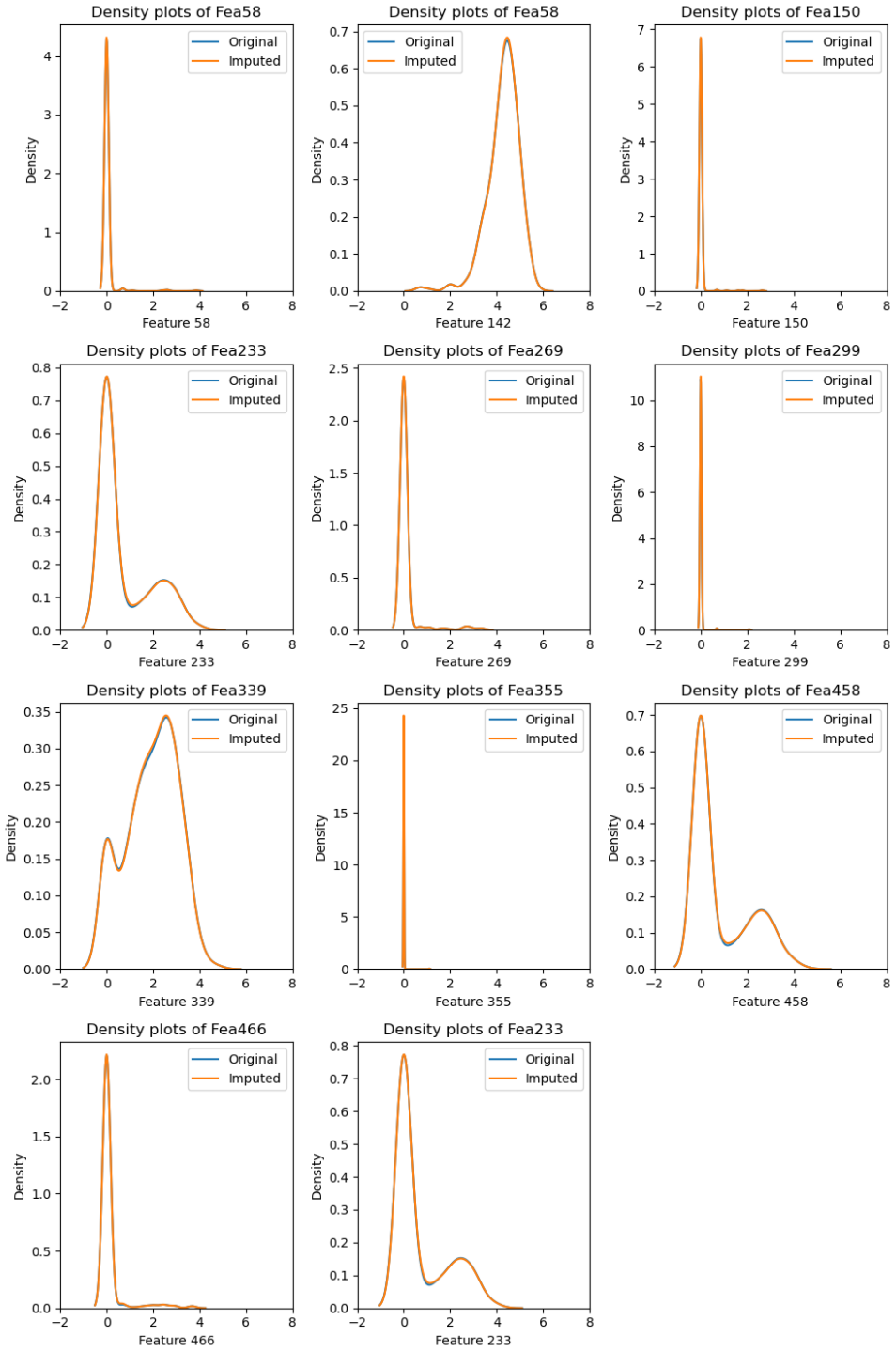

Figure 5: **Density plots of features before and after the imputation**

dividing by their standard deviation [2, p. 73]. This thus expresses each value by how many standard deviations they are from the mean. By then setting a threshold of 3, values further than 3 standard deviations can be identified. Here, outliers were found in numerous rows and over all features. Over 2900 were found in the total of 204,000 data points in the dataset.

(e)

Here, a KNN-imputer was used. By training the model on the data with the outliers still there, the outliers can then be dropped by setting them to NaN. Then using the imputer, the missing values/outliers are imputed/corrected with new values that reflect the overall dataset rather than the outliers. After imputation, the number of outliers is reduced to 1385.

Some of those distributions are not plotted as the variance of those features is now 0, after the imputation. Figure 6 shows that some distributions have been narrowed, which is what is expected from correcting outliers.

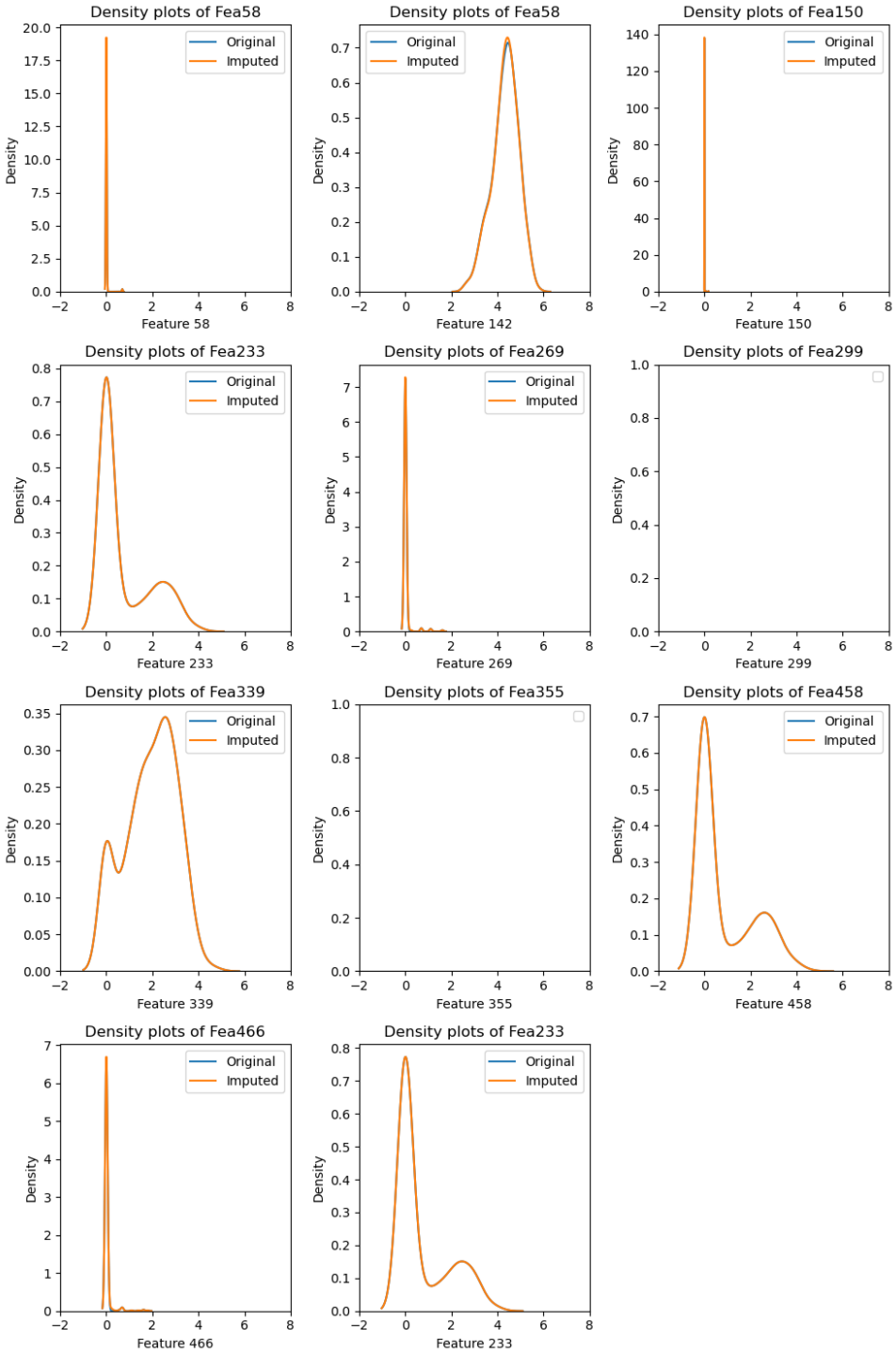

Figure 6: **Density plots of features before and after outlier correction**

Question 4

(a)

A single decision tree is grown through recursive binary splitting of the data. It is a greedy search as at every node, the splitting is only made according to what the best split is at that time [1, pp. 337-338]. That is described by some criterion, like the Gini Index. The Gini index is a measure of the node "purity", how much of one class the node contains [1, pp. 338-339]. Once the tree is fully grown down to having only totally pure nodes at the end, one can prune it, removing some of the last nodes to avoid having an overtrained tree. In bagging, multiple trees are grown independently on bootstrapped samples of the dataset and combined to reduce the high variance issues single decision trees have [1, p. 343]. Random forests are built in a similar way to bagging but as a tree is built on a bootstrapped sample, at each split the tree has to make, it is only performed on a subset of the features. This decorrelates the trees, and let's them explore a lot more possible splits [1, 354]. This sort of helps counteract the greedy nature of the tree. When using a random forest there are a few hyperparameters one can tune to control how each tree is grown. First is the criterion used, as there are options other than the Gini index. Second, the number of features that the tree has to split from at each node. It is usually set to the squareroot of the total number of features.

(b)

In preparation to training a classifier on the data, some preprocessing was done. The first thing done was to count missing values, and none were found. Then, checks were run for duplicates and none were found either. Now, checks were made for zero-variance features and the following features were identified and dropped:

0 Variance features: [Fea49, Fea66, Fea94, Fea97, Fea106, Fea109, Fea125, Fea129, Fea151, Fea152, Fea187, Fea189, Fea223, Fea224, Fea238, Fea264, Fea293, Fea324, Fea384, Fea432, Fea440, Fea450, Fea463, Fea543, Fea636, Fea666, Fea680, Fea701, Fea707, Fea728, Fea729, Fea750, Fea785, Fea787, Fea808, Fea830, Fea846, Fea930, Fea943, Fea945, Fea973, Fea982]

Further, with the average variance being 0.618, 16 rows were found with near-zero variance and were also dropped:

Near-zero Variance features: [Fea747, Fea730, Fea57, Fea544, Fea4, Fea154, Fea597, Fea610, Fea19, Fea855, Fea347, Fea843, Fea499, Fea231, Fea404, Fea546]

Outliers were then dealt with

Finally, 2 pairs of highly correlated features (¿ 0.9) were found and one of each pairs were dropped: Feature 300 and 345, and Feature 869 and 954.

Because the Random Forest classifier is robust to outliers, outliers are not dealt with [1, pp. 346-347].

(c)

The random forest classifier was trained on default parameters: 100 trees, Gini index splitting criteria, no max depth set, and the max number of features to split from set to the squareroot of the total number of features.

Training this model, a test set classification error of 0.04 is found.

(d)

This number of trees in the random forest can be optimised using an Out-Of-Bag (OOB) score. This works from the fact each tree is built for a bootstrapped subsample of the dataset. Leaving an "Out-Of-Bag" sample the trees for which that sample was OOB can be tested upon. An OOB classification error can thus be obtained in Figure 7, for each number of trees [1, p. 345].

Thus, the optimal number of trees is around 150, where the OOB error stabilizes, and going further only makes the model more computationally expensive.

Re-running the model for the optimal tree number, the test-classification error now is 0.04. Thus the improvement is minimal in this case.

(e)

One characteristic of the random forest classifier is it can give us the relative importance of features. It is computed as how much of a reduction in the Gini index (or other used criterion) is brought by that feature. Were it not for the bootstrapping and subset of features to split from, there would not be enough information for that measure to be computed accurately [1, pp. 345-346]. In this case, we obtained the plot in Figure 8, with the 50 most important features:

We see that feature importance decreases very rapidly before steadily decreasing slowly. Taking the first 20, another random forest model can be trained. With this model, a test set classification error of 0.0699 was obtained. As expected, the performance is slightly lower but considering the amount by which computational complexity was decreased, this is a very positive result.

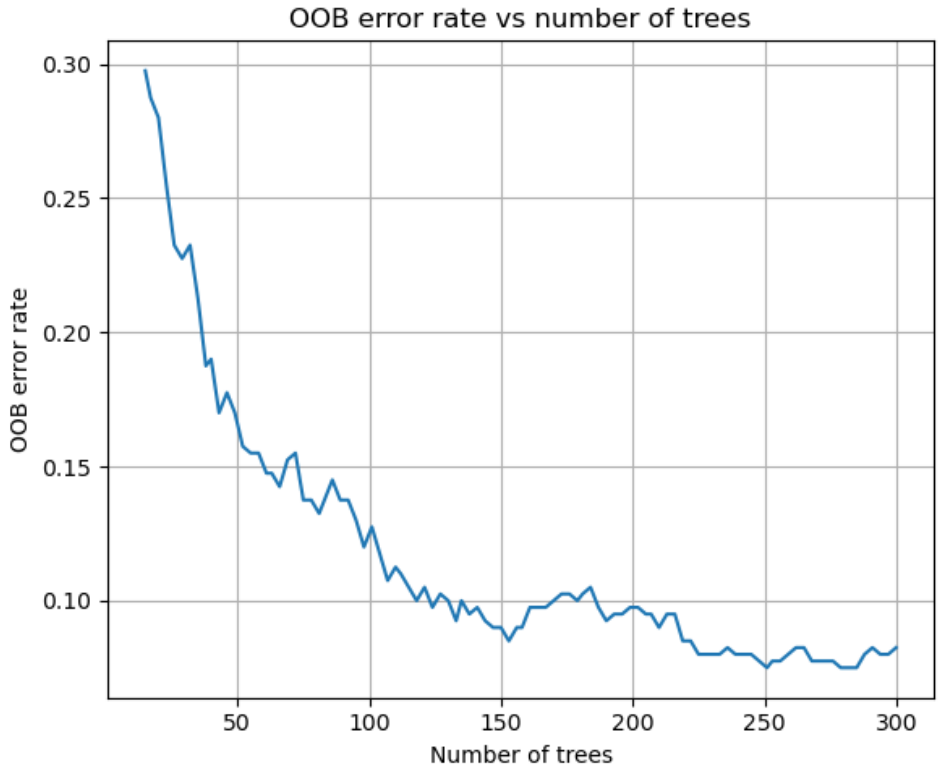

Figure 7: **OOB error vs number of trees** Density contours are also shown.

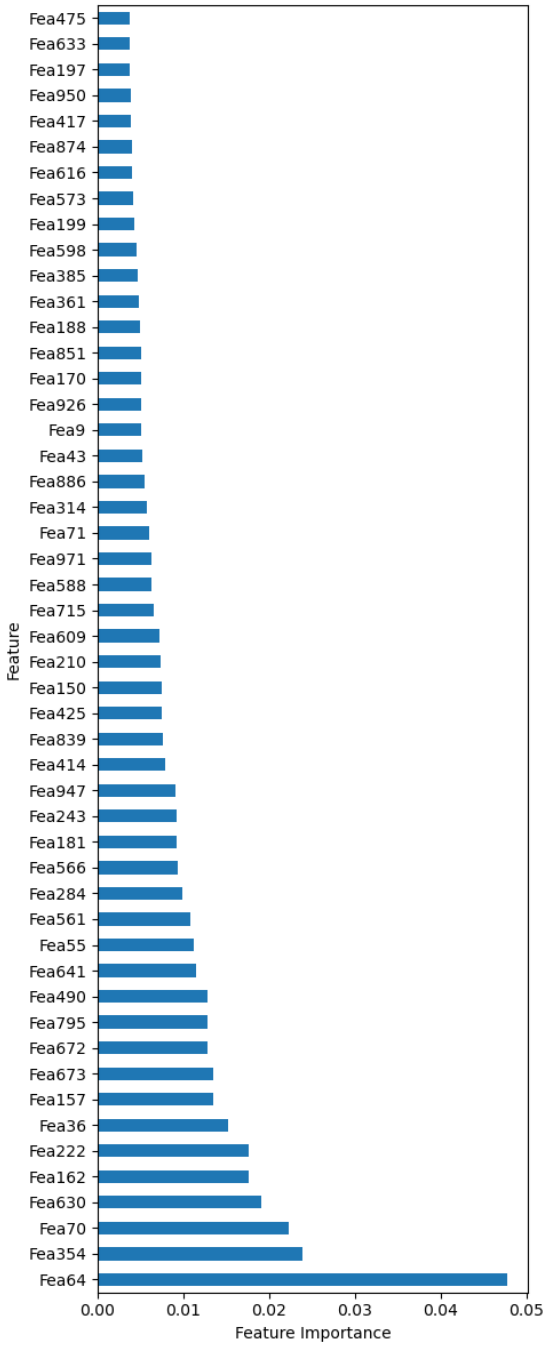

Figure 8: **Random Forest Feature Importance**, for the 50 most important features

(f)

Here, the previous questions (b, c, e) are reproduced using another supervised learning method. The chosen method is Support Vector Machines (SVM), as it is quite good when the data is high dimensional, and even when there are more dimensions than samples [3]. However, in order to get the feature importances back from the SVM model, one needs to use a linear kernel.

The data was preprocessed in this same way as for the Random Forest Classifier. Duplicates, low-variance, and highly correlated rows and features are dealt with, and outliers are left alone since the SVM algorithm is robust to them.

Running this, a test set classification error of 0.0799 was obtained. And using the `coeff_` output of the model, the feature importances were obtained and plotted in figure 9. As one cans see there is some agreement between the 2 models, with the most important feature being the same and a few features with relatively similar importance (Fea36, Fea354,. . .) Training the SVM model on just the most important feature, the test set classification error roughly doubles but stays indicative of good performance.

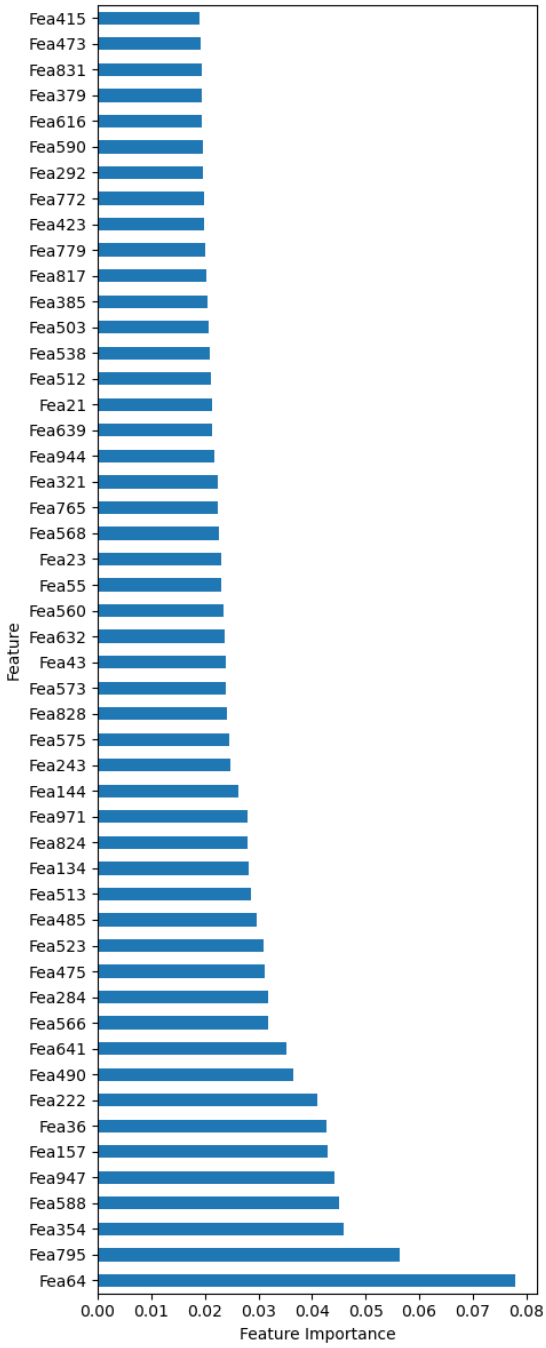

Figure 9: **SVM Feature Importance**, for the 50 most important features

Question 5

(a)

Two different clustering techniques are explored in this question. One of them will be K-means which has been already discussed partly. It will place a desired number of initial centroids in the feature space and update their position based on cluster members until convergence. The two main outputs are then the cluster centroids' final location and the cluster memberships of all the datasets' points. The second technique will Gaussian Mixture Models. The idea behind this is to assume the entire data is distributed as the sum of multiple multivariate Gaussian distributions with unknown parameters [2, p. 260]. The main outputs for this clustering technique are the parameters of those gaussians (means and covariance matrices), and the weights for each of those Gaussians to be used in the sum. Both of these techniques' output can be used on a dataset to assign cluster memberships to each observation. This is done by assigning each observation to the cluster whose centroid is closest to it for K-means, and by assigning each observation to the Gaussian which has the highest probability of generating that observation for GMM. One property of the GMM is that it can be used to generate new observations, by sampling from the weighted Gaussians. This makes the GMM a generative model.

Here, the data was preprocessed in the same way as in Q4 with the addition of dealing with outliers, using the same method as in Q3. Using these methods, clusterings are obtained both for the same data which makes comparison easier. The number of clusters is set to 3 for both methods, which is based on the visualisation of the data, see Figures 12 and 13. The results are shown in Tables 8 and 9.

For each clustering technique, label frequencies were obtained.

Label	**Frequency**
0	103
1	173
3	224

Table 8: Frequency of the labels for the K-means clustering of **Baseline** dataset

Label	**Frequency**
0	169
1	176
3	155

Table 9: Frequency of the labels for the GMM clustering of **Baseline** dataset

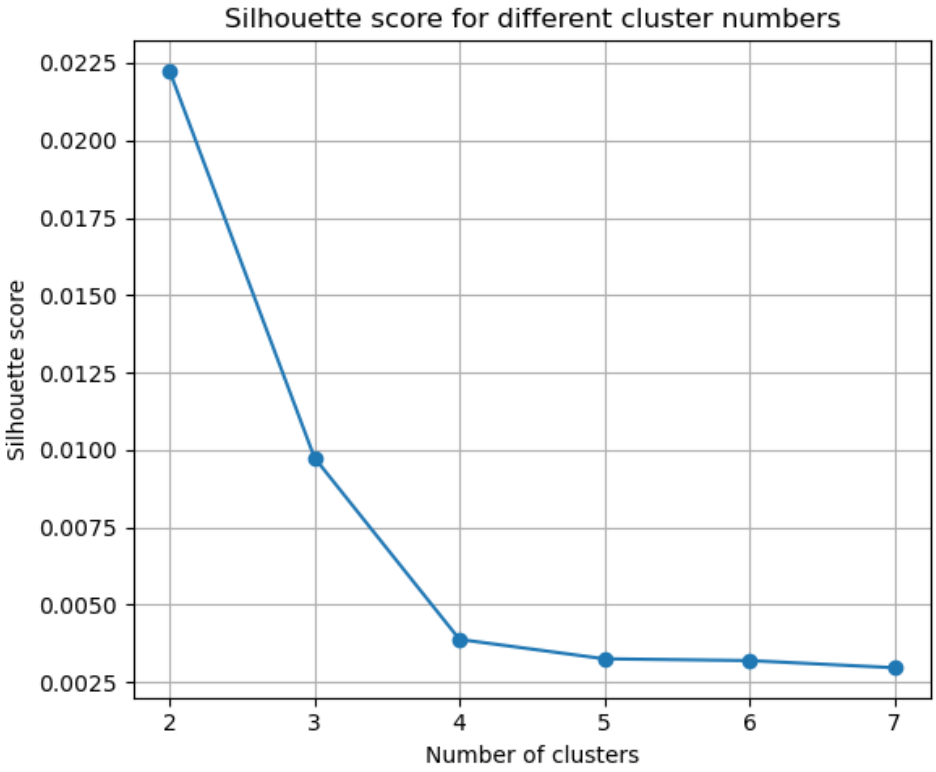

Figure 10: **Silhouette scores for the Baseline Dataset**

And the following contingency table comparing the 2 clusterings was obtained:

Clusters	**0**	**1**	**2**
0	53	45	5
1	4	68	101
2	112	63	49

Table 10: Contingency table for the 2 clusterings on the **Baseline** dataset. Rows are K-means, columns are GMM.

Thus, the clusterings are not very stable. There are 2 larger groups (112, 101) which show some agreement for the "cores" of 2 clusters though 5 groups of consequent size (53, 45, 68, 63, 49) which clearly show some ambiguity as to what cluster some points should belong to.

(b)

Similar to Q4, a random forest classifier is trained to determine feature importances. However, where the problem was supervised and the true labels were known, here the labels obtained from the clusterings are used. Thus, 2 models are trained and the following feature importances were derived.

As one can see, there is some agreement between the 2 on what features are most important. With each set of most important features, the clustering is re-run for each technique based on each top 40 features only. The following contingency tables compare the 2 top 40 features models together and for each technique, the top 40 and the total features clustering.

Clusters	**0**	**1**	**2**
0	6	103	5
1	17	53	79
2	175	0	62

Table 11: **Contingency table for the 2 clusterings on the Baseline dataset.** For the top 40 most important features. Rows are K-means and columns are GMM.

(c)

Having applied PCA to run the GMM clustering, the data can be visualised on the first 2 PCs. The following plots show the data colour coded according to cluster memberships for the total features clustering and the top 40 features clustering, each for the 2 clustering techniques.

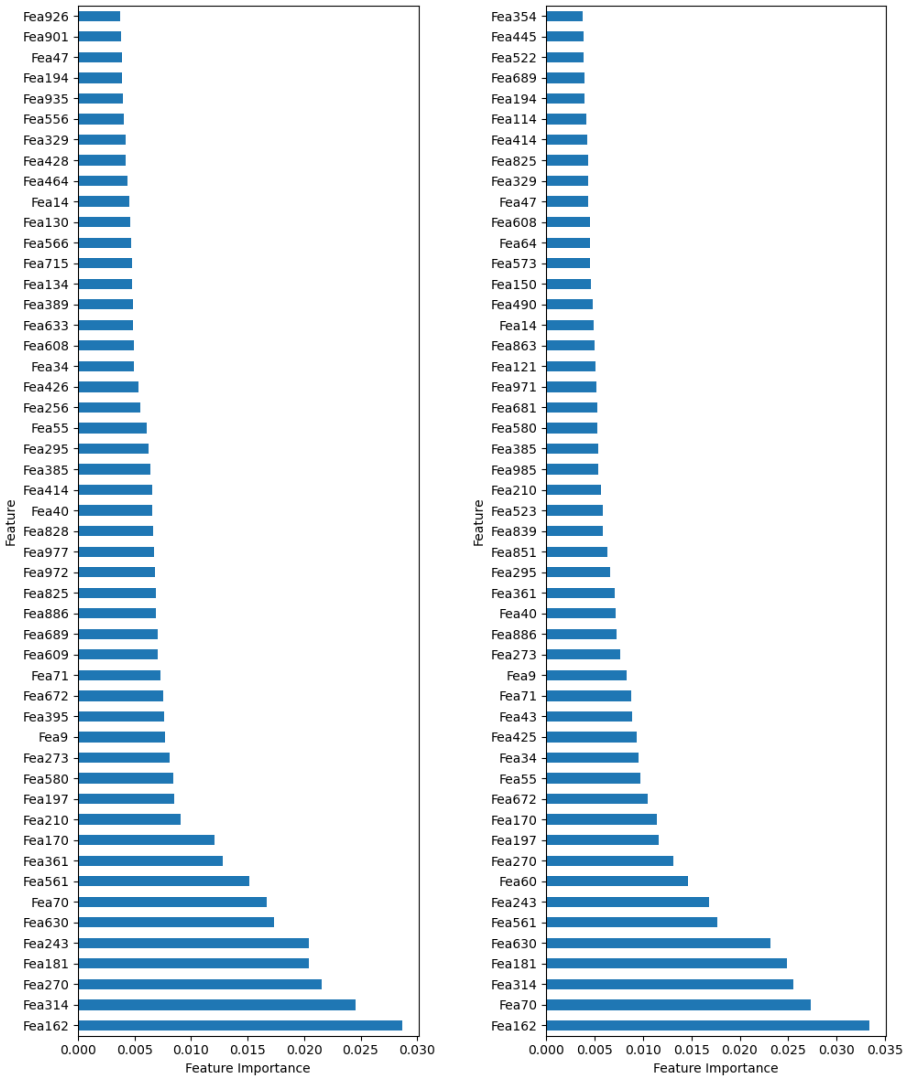

Figure 11: **Random Forest Feature Importance**, for the 50 most important features, from K-means clustering (left) and GMM clustering (right)

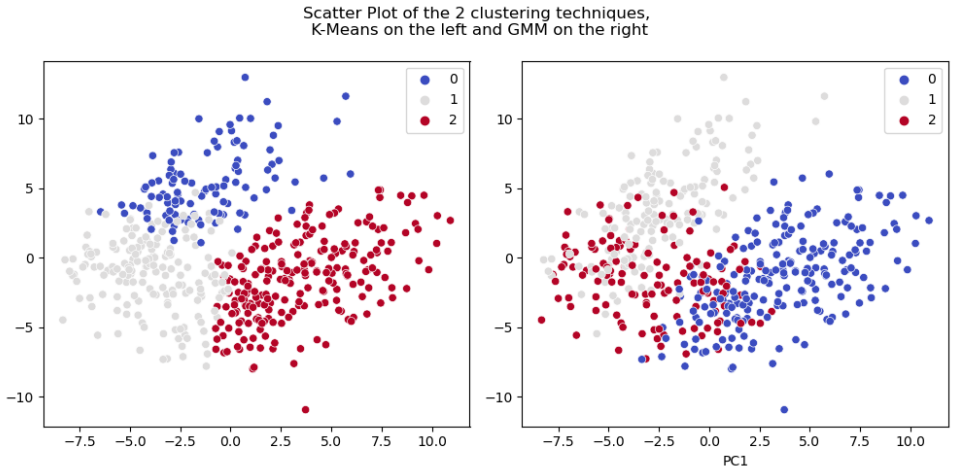

Figure 12: **Scatter plot of cluster memberships.**For the total features clustering. K-means (left) and GMM (right).

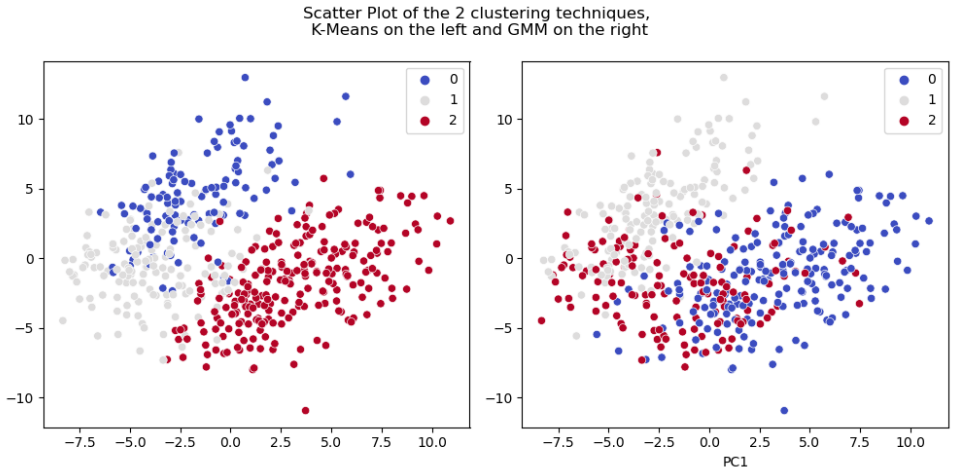

Figure 13: **Scatter plot of cluster memberships.**For the top 40 features clustering. K-means (left) and GMM (right).

Clusters	0	1	2
0	91	20	3
1	11	130	8
2	1	23	213

Table 12: **Contingency table for the top 40 and total features clustering on the Baseline dataset.** For K-means. The rows are the top 40 features clustering and the columns are the total features clustering.

Clusters	0	1	2
0	168	9	21
1	3	134	19
2	43	10	93

Table 13: **Contingency table for the top 40 and total features clustering on the Baseline dataset.** For GMM. The rows are the top 40 features clustering and the columns are the total features clustering.

Here, the agreement can be seen between the 2 techniques. The top 40 features clustering is more ambiguous, with the clusters overlapping more, especially for K-means. The main issue with the 2 clustering techniques used is that they are both at risk of falling into local minima. This is especially true for K-means, which is very sensitive to the initialisation of the centroids [2, p. 263]. This could be the reason for some of the disagreements. Futhermore, the Baseline dataset is quite high dimensional, with 1000 features.

Figure 12 and 13 show the data colour coded according to the first and second most important features for both clustering techniques. The first most important feature seems to help in differentiating between the 2 right and left clusters, and the bottom cluster, comparing with figure 10 and 11. The second most important feature seems to help in differentiating between the right and left clusters, again looking at Figure 10 and 11.

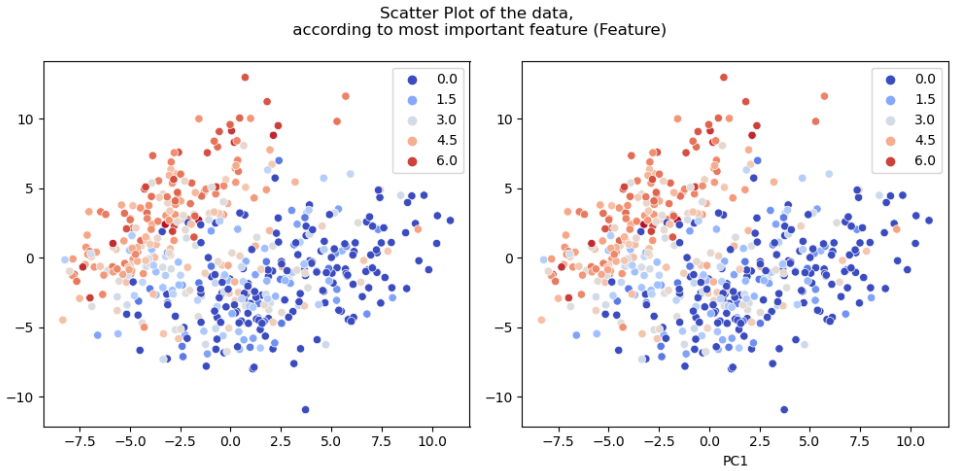

Figure 14: **Scatter plot of the data color coded according to the most important feature.** K-means (left) and GMM (right).

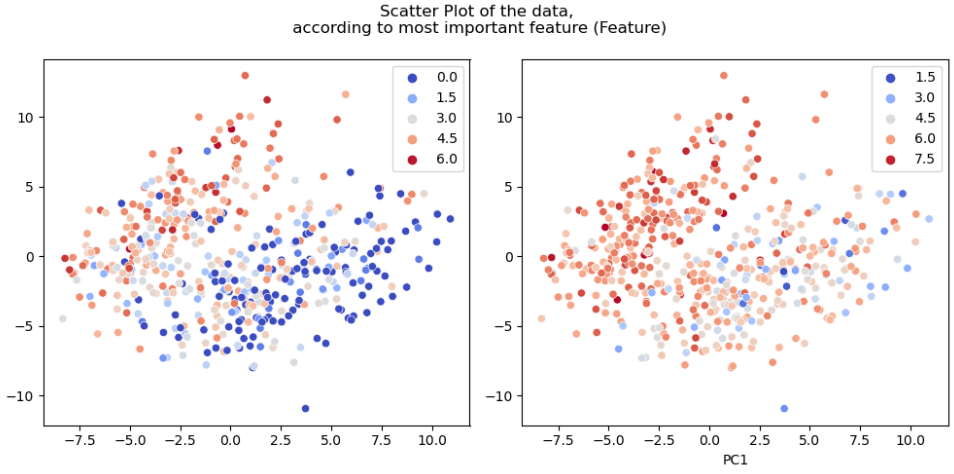

Figure 15: **Scatter plot of the data color coded according to the 2nd most important feature.** K-means (left) and GMM (right).

Appendix

Copilot's autocompletion feature was used in coding the project, when writing docstrings for the functions, and when covering repetitive parts of the code, such as the density plots in Plot_funcs. ChatGPT was used to help in debugging the code, by providing the tracebacks when an error was difficult to understand.

Bibliography

[1] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. *An Introduction to Statistical Learning with Applications in Python*. Springer Texts in Statistics. Springer, 2013.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. scikit-learn: Machine learning in python. `https://scikit-learn.org`, 2011.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. scikit-learn: Support vector machines. `https://scikit-learn.org/stable/modules/svm.html`, 2021.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. sklearn.cluster.kmeans. `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans`, 2021.

[5] Jerome P. Reiter. Statistical analysis of network data with r. `http://www2.stat.duke.edu/~jerry/ReiterUNC2020.pdf`, 2020.