AINT252 Computational Theory & AI Coursework

Question 1 & Question 2

The submission includes two versions of code for each question, this is because some aspects of the updated scripts, such as use of “”, is only available on later versions of MATLAB (>=R2017a). Both scripts generate the same results and perform have identical functionalities, however, the updated scripts use loops in order to make the code more efficient as well as be adaptable for values of K.

Question 1 – K-means clustering (Unsupervised Learning)

To view all the code in order to calculate these answers as well as to see the graphs, figures and the original dataset in full, please refer to the file ‘Question_1.m’.

Data Analysis

1) *The total number of rows (objects, cases) in your data. = 2268*

To report the total number of objects and cases in the data sample given the MATLAB function ‘size()’ can be used, for example [size(dataset,1)], returns the total number of rows for the 1st scalar dimension/rows for whatever the dataset is. In the case of our data set the total number of rows = 2268.

2) *For each column (feature) from 1 to 4, find the mean, the standard deviation and the histogram.*

The mean and standard deviation as well as histograms for each column is generated and calculated; a histogram for all columns was also created (shown in figures below). As 4 features existed, four means and standard deviations were expected. Using the MATLAB functions ‘mean()’ and ‘std()’ made this easy and put the results for each column into a table (seen in the MATLAB code file).

Column/Feature 1 Mean = 9.4937	Column/Feature 1 Standard Deviation = 3.5325
Column/Feature 2 Mean = 9.4942	Column/Feature 2 Standard Deviation = 3.5442
Column/Feature 3 Mean = 9.4806	Column/Feature 3 Standard Deviation = 3.5920
Column/Feature 4 Mean = 9.4894	Column/Feature 4 Standard Deviation = 3.5925

Figure 1 – Mean and standard deviation values returned from relative MATLAB functions

The histograms show the frequency distribution for each feature/column in the dataset.

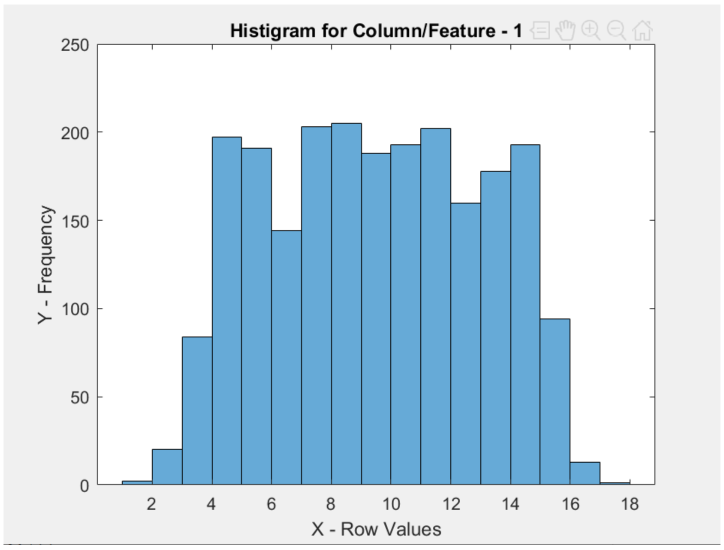


Figure 2 – Histogram for the first column/feature in the given dataset.

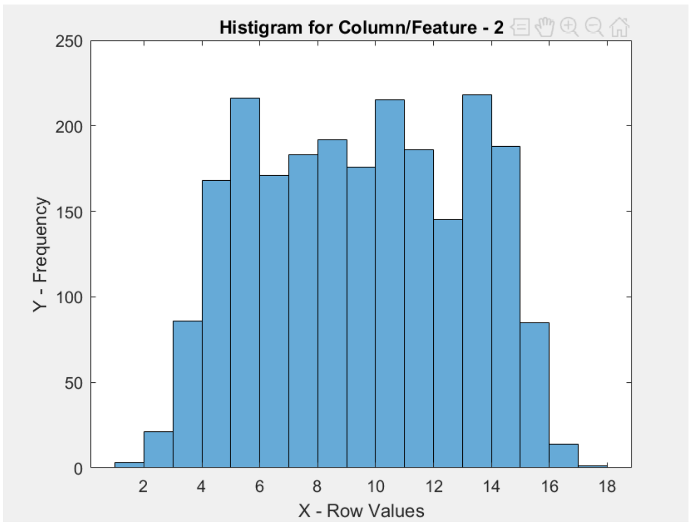


Figure 3 – Histogram for the second column/feature in the given dataset.

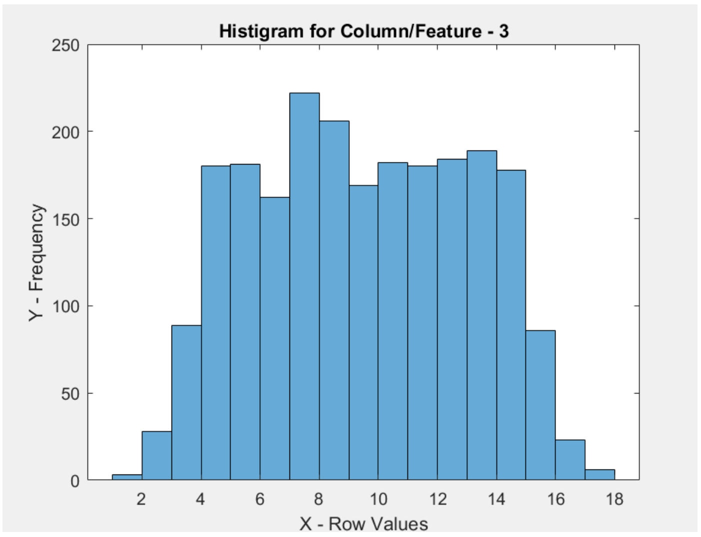


Figure 3 – Histogram for the third column/feature in the given dataset.

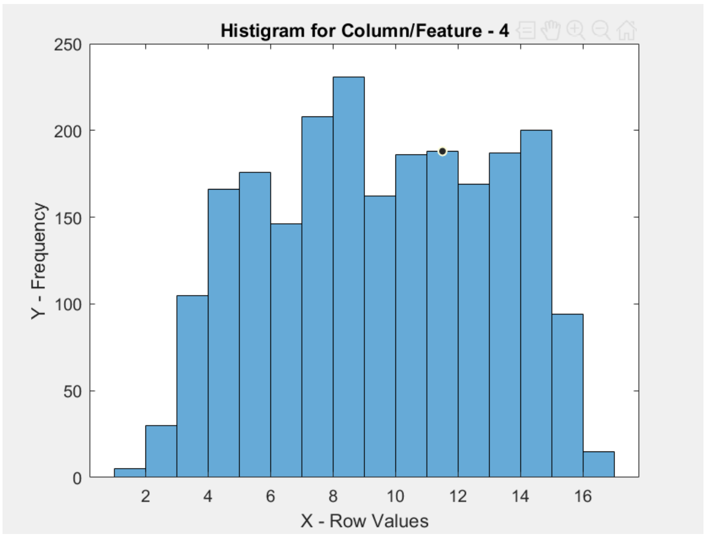


Figure 4 – Histogram for the forth column/feature in the given dataset.

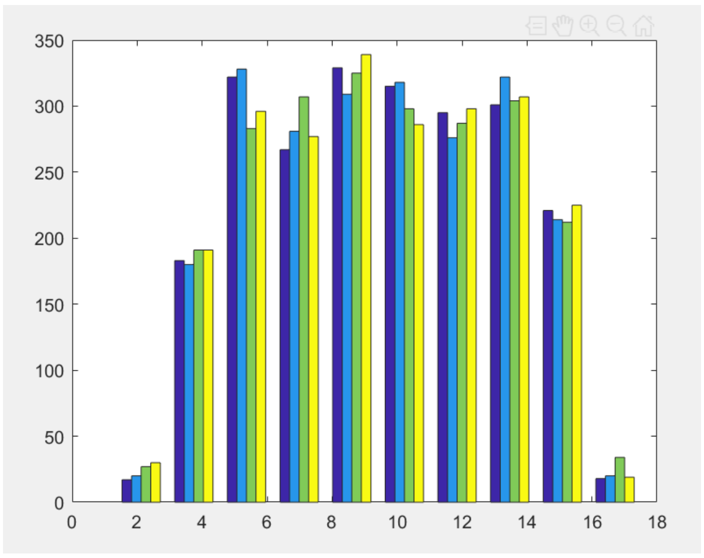
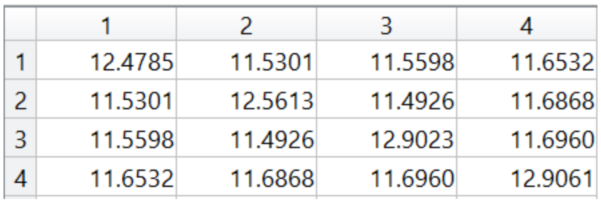


Figure 5 – Histogram for all columns/features in the given dataset.

3) The covariance matrix and the correlation matrix of the given dataset.

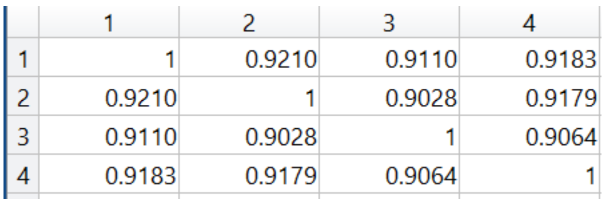
The covariance matrix (Figure 6) is a way of showing the covariance between the features, we can use this to assess the direction of the linear relationship between them, this shows how as feature x changes, how feature y changes in relation to x. With all values being positive, we expect a positive trend when plotting the data. To calculate the covariance matrix MATLAB has the function 'cov()', with this the table (Figure 6) is generated from the dataset.



	1	2	3	4
1	12.4785	11.5301	11.5598	11.6532
2	11.5301	12.5613	11.4926	11.6868
3	11.5598	11.4926	12.9023	11.6960
4	11.6532	11.6868	11.6960	12.9061

Figure 6 – Covariance Matrix for the given dataset.

The correlation matrix (Figure 7) examines the strength and direction of the linear relationship of the features, or in simpler terms how similar each feature is to each other. The correlation values can range between -1 and +1 with +1 indicating a strong relationship between features. From the generated correlation matrix, a value no lower than 0.9028 is calculated, meaning that a strong positive linear relationship exists between the features. MATLAB has the function 'corrcoef()' to calculate a correlation matrix.



	1	2	3	4
1	1	0.9210	0.9110	0.9183
2	0.9210	1	0.9028	0.9179
3	0.9110	0.9028	1	0.9064
4	0.9183	0.9179	0.9064	1

Figure 7 – Correlation Matrix for the given dataset.

From the correlation matrix and the covariance matrix we should expect a mostly positive trend in plotting the data, as well as the datapoints being fairly close together.

4) Further analysis of the data.

When looking into the previous analysis of the data, the means and standard deviations of the feature are all similar, this indicates the features should all be much the same and their retrospective values should all differ similarly too. When looking at the histograms for the distribution of the data we can see that the features very roughly follow the bell curve of normal distribution, although, the values around the middle of the data are more similar in their distribution.

When plotting the first two features with the above analysis and what previously mentioned (Figures 1-7) we would expect a positive, strong linear relationship of the data and this is shown (Figure 8). When looking at the data in respect to 'K-Means Clustering' we can roughly see 4 main clusters, this will be later proven with analysis of suitable values for K.

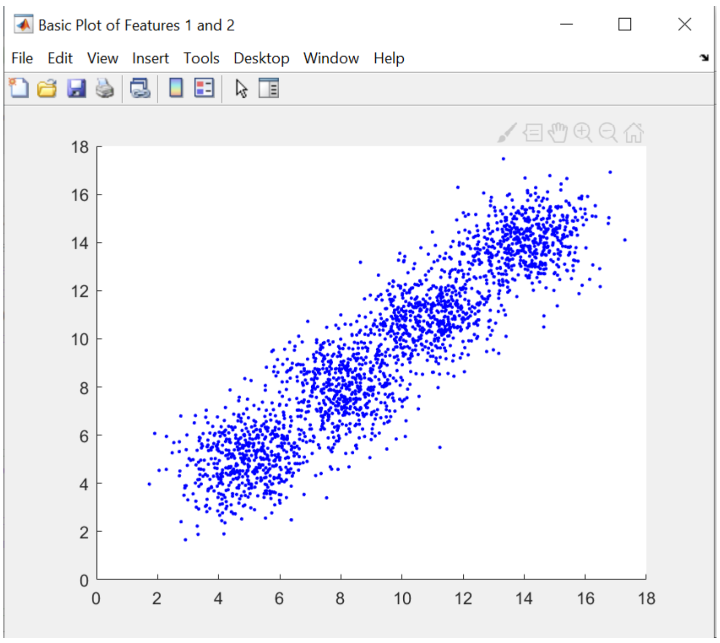


Figure 8 – Scatter plot of the first 2 features in the given dataset.

K-means algorithm: Finding the optimal number of classes.

The K-means algorithm, or the ‘kmeans()’ function in MATLAB, sorts data based upon the set number of K, with K being the number of clusters to sort the given dataset into. K-means is a unsupervised learning algorithm which means data is sorted without a teacher/teaching set and the algorithm will learn from its own actions, think of the K-means clustering algorithm as self-sorting.

With the value of K set, all datapoints in the set are firstly randomly assigned to a cluster by using an index for each row/datapoint in the dataset, with K = 4 we would expect the 4 clusters to be generated. This index is acts like a label that tells each datapoint what cluster it belongs to. The centroids/central points are then calculated by finding the mean distance (in our case Euclidean distance) between all the points for each cluster to find the central point of each cluster. Each datapoint then has its index changed based on what central point itself is closer to. The central point is calculated again and then the datapoints will be tested if they are closer to a different centroid. These steps are repeated until no more improvements can be made and the data will be sorted into K clusters. The centroids are stored in a sperate table through the ‘kmeans()’ function.

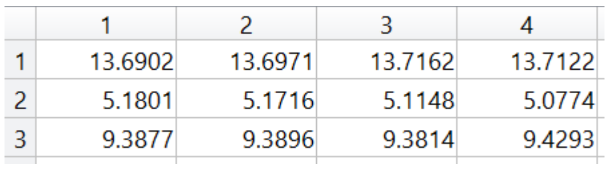
In order to find the optimal number for K or in other words the optimal number of classes, we can use silhouette plots as well as finding the mean silhouette measure. The silhouette plots measure each point for each cluster and gives a value between -1 and 1 as a measurement on how similar each point in a cluster is to its neighbouring datapoints. The mean silhouette measure is a mean average of all the datapoint silhouette values; as we change K the mean silhouette measure will change; getting this closest to 1 is the aim as this would mean for each clusters, each point is related to each other as much as it possibly can be.

Setting K to 3,4,5 and finding the optimal number for K

a) *Repeat the following procedure 3 times: select the value of the parameter k*

K=3

1) *The coordinates of the centroids of the clusters for K=3*



	1	2	3	4
1	13.6902	13.6971	13.7162	13.7122
2	5.1801	5.1716	5.1148	5.0774
3	9.3877	9.3896	9.3814	9.4293

Figure 9 – Coordinates of the centroids of clusters when K = 3

2) *The silhouette plots for K=3*

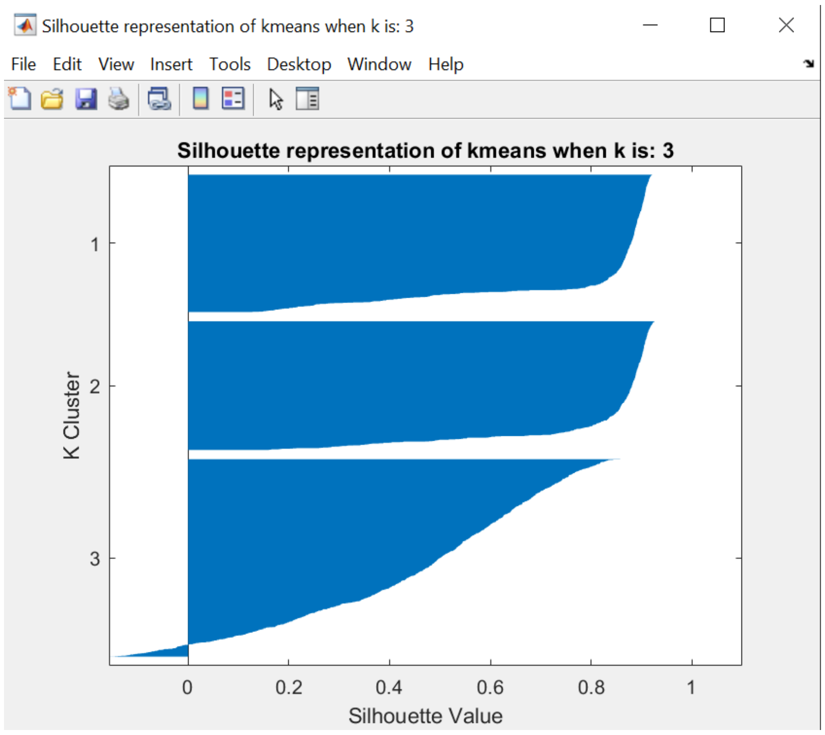


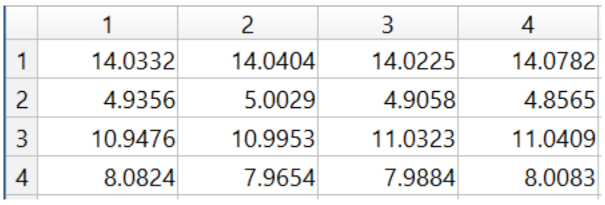
Figure 10 – Silhouette Plot for clusters when K = 3

3) *The mean silhouette measure for K=3*

From the above silhouette plot we can calculate the mean silhouette value by finding the mean for the silhouette values, as previously mentioned this will tell us how well-suited K, or the number of clusters, is for the dataset. The mean silhouette for when K=3 is: 0.6589.

K=4

1) *The coordinates of the centroids of the clusters for K=4*



	1	2	3	4
1	14.0332	14.0404	14.0225	14.0782
2	4.9356	5.0029	4.9058	4.8565
3	10.9476	10.9953	11.0323	11.0409
4	8.0824	7.9654	7.9884	8.0083

Figure 11 – Coordinates of the centroids of clusters when K = 4

2) *The silhouette plots for K=4*

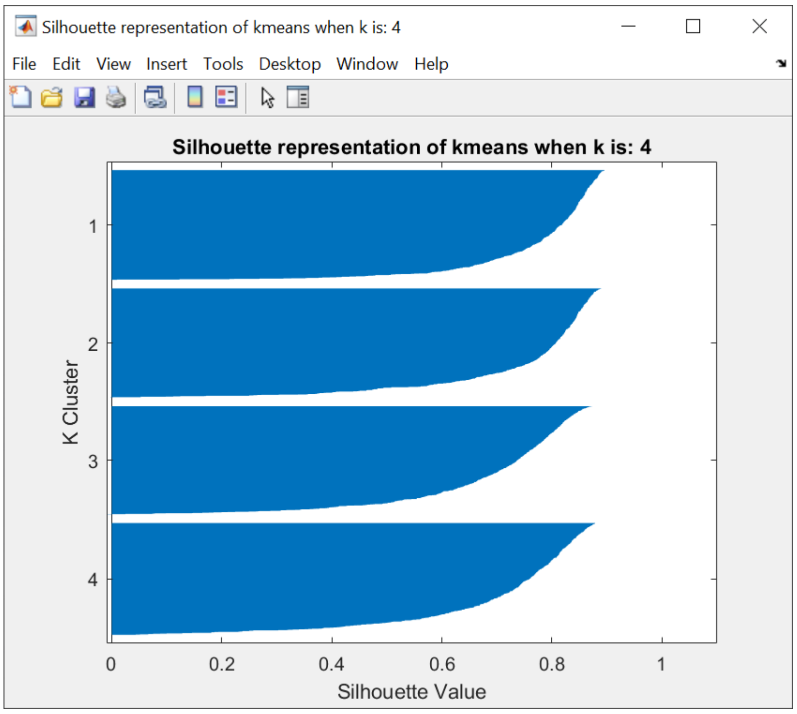


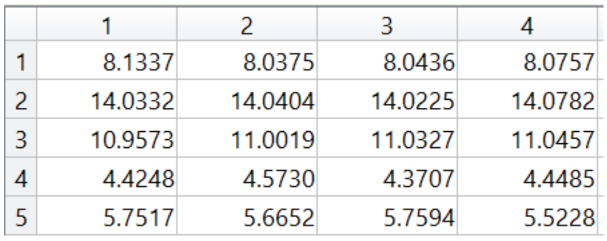
Figure 12 – Silhouette Plot for clusters when K = 4

3) *The mean silhouette measure for K=4*

From the above silhouette plot we can calculate the mean silhouette value by finding the mean for the silhouette values, as previously mentioned this will tell us how well-suited K, or the number of clusters, is for the dataset. The mean silhouette for when K=4 is: 0.7309.

K=5

1) *The coordinates of the centroids of the clusters for K=5*



	1	2	3	4
1	8.1337	8.0375	8.0436	8.0757
2	14.0332	14.0404	14.0225	14.0782
3	10.9573	11.0019	11.0327	11.0457
4	4.4248	4.5730	4.3707	4.4485
5	5.7517	5.6652	5.7594	5.5228

Figure 13 – Coordinates of the centroids of clusters when K = 5

2) *The silhouette plots for K=5*

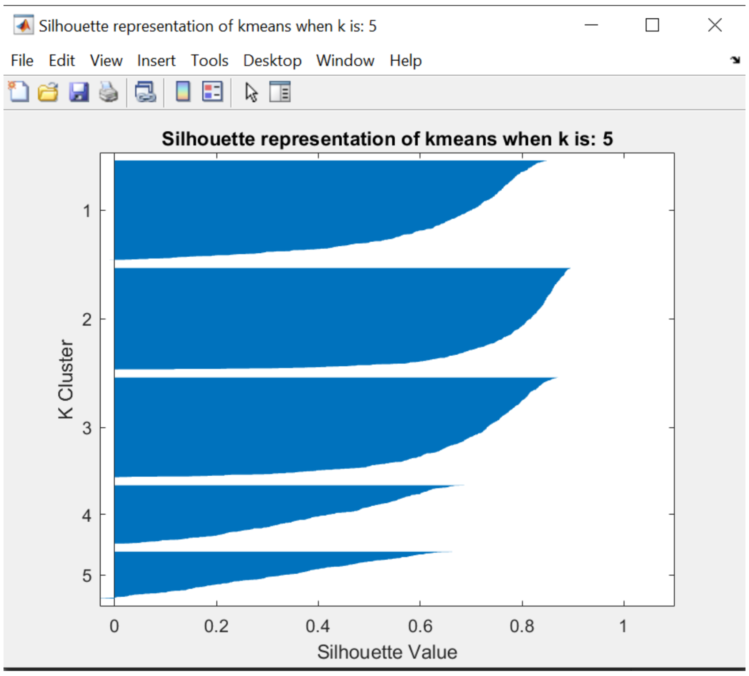


Figure 14 – Silhouette Plot for clusters when K = 5

3) *The mean silhouette measure for K=5*

From the above silhouette plot we can calculate the mean silhouette value by finding the mean for the silhouette values, as previously mentioned this will tell us how well-suited K, or the number of clusters, is for the dataset. The mean silhouette for when K=5 is: 0.6147.

b) ***Compare the results of the three classifications (K = 3, 4, 5), and select the best classification***

1) Report the optimal number of classes (K = 4)

In order to find the optimal number of classes the silhouette plots and mean silhouette values are compared (Figures 9-14). As mentioned, the higher the mean silhouette value the better suited K is for the clustering. When K = 4 the mean silhouette measure is 0.7309, the highest of all the runs for K; this indicates that K = 4 or 4 clusters is the optimal number of classes as all datapoints in each cluster are on average closer together.

With further analysis when looking at the silhouette plots themselves (Figures 10,12,14), on visual inspection with K = 3 there are negative silhouette values showing a very poor cluster 3, with this not appearing in other silhouettes you could argue K = 3 can be disregarded. When comparing the silhouette plots for K = 4 and K = 5, 4 also proves better with all cluster silhouettes being similar shapes as well as being closer to being rectangular. Being closer to rectangular tell us that each datapoint relates to other datapoints much the same as other datapoints; K = 5's clusters 4 and 5 the silhouettes don't appear right.

When plotting the clusters in 2D we can visually assess the data, even without K-Means clustering and plotting the data (Figure 8) 4 clusters can be assumed to exist. When plotting K = 3 (Figure 15) the green cluster has a centroid where less points exist, this doesn't match initial expectations nor does it match the other clusters, unlike K = 4 (Figure 16) where the clusters appear as expected, all similar in shape. When plotting the K = 5 clusters (Figure 17), its the yellow and pink clusters that fail the expectation, it appears as a merging of two clusters and not individual clear clusters.

Concluding from the above analysis, **K = 4** is the optimal number of classes

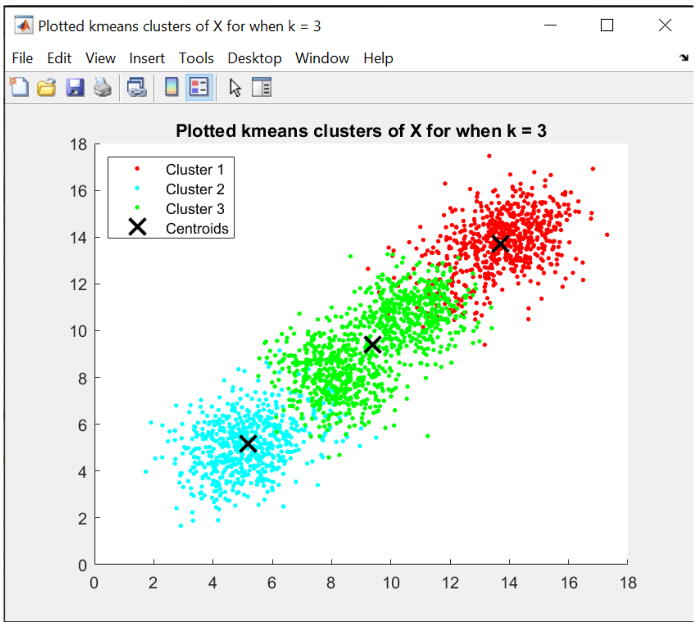


Figure 15 – Plot for features 1 and 2 to visualise clustering when K = 3

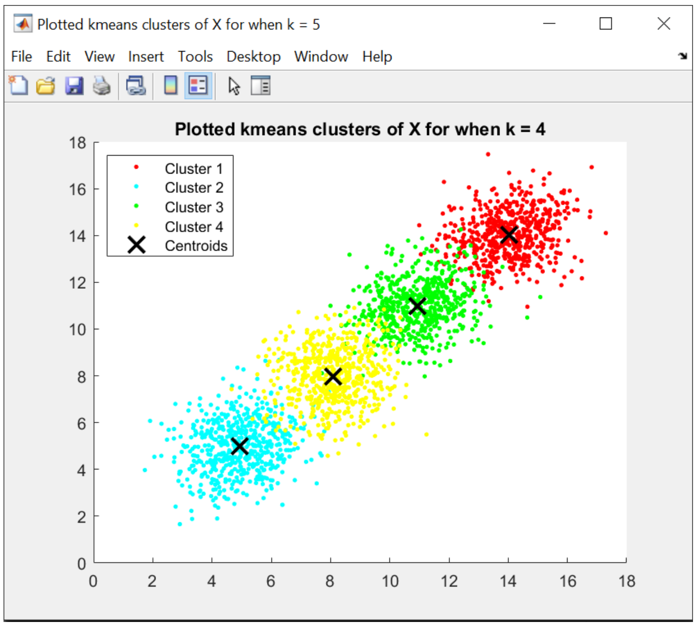


Figure 16 – Plot for features 1 and 2 to visualise clustering when K = 4

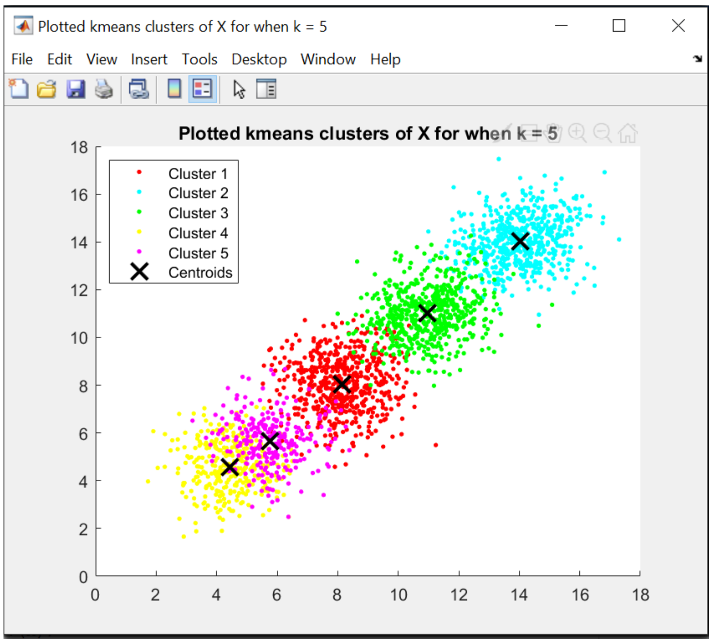


Figure 17 – Plot for features 1 and 2 to visualise clustering when K = 5

2) *Visualise the optimal number of classes in a 3D projection of the clustered data, report the MATLAB figure.*

With K = 4 being the optimal number of classes this was then plotted in 3D using the 'scatter3()' function; to do this the first 3 features of the dataset is used. In 3D, further features can be shown how they are clustered into spherical shapes further showing how the features are clustered correctly. The 3D projection shows further how the K-means clustering algorithm applies to all features. The 3D plot is seen better when run in MATLAB when it is rotated in real time.

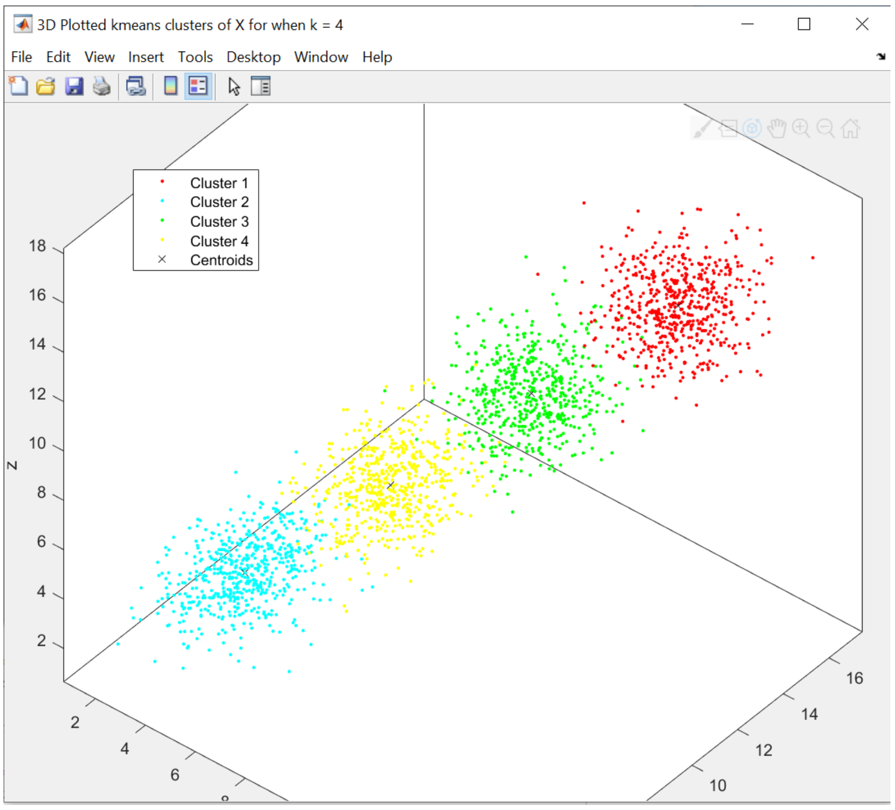


Figure 18 – Plot for features 1,2 and 3 to visualise clustering when K = 4 (OPTIMAL)

Question 2 – K Nearest Neighbour (KNN) Classifier for Supervised Learning

To view all the code in order to calculate these answers as well as to see the graphs, figures and the original dataset in full, please refer to the file 'Question_2.m'.

Data Analysis

1) *The total number of rows (objects, cases) in your data. = 3402*

Getting the total number of rows again like question 1, used the 'size()' function, with [size(dataset,1);] returning the amount of rows in the 1st scalar dimension of rows. In this case the dataset has the number of rows = 3402.

2) *For each column (feature) from 1 to 5 report the mean and standard deviation*

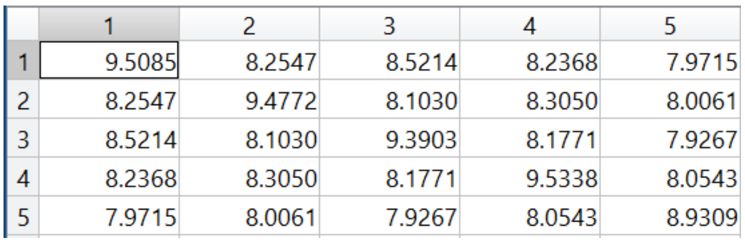
Again like 'Question 1' the MATLAB functions 'mean()' and 'std()' were used to calculate the mean and standard deviation. Although as the given dataset give the class labels in column/feature 6, these were excluded from the data and columns and features 1 to 5 are put into a separate dataset by using [dataset = Y(:,1:5)]. By putting this new dataset into 'mean()' and 'std()' we are sure only features 1 – 5 are calculated. The results are shown below (Figure 19).

Column/Feature 1 Mean = 8.9007	Column/Feature 1 Standard Deviation = 3.0836
Column/Feature 2 Mean = 8.9195	Column/Feature 2 Standard Deviation = 3.0785
Column/Feature 3 Mean = 8.9161	Column/Feature 3 Standard Deviation = 3.0644
Column/Feature 4 Mean = 8.8741	Column/Feature 4 Standard Deviation = 3.0877
Column/Feature 5 Mean = 8.9112	Column/Feature 5 Standard Deviation = 2.9885

Figure 19 – Mean and standard deviation values returned from relative MATLAB functions

3) *Covariance matrix and the correlation matrix of features 1 to 5*

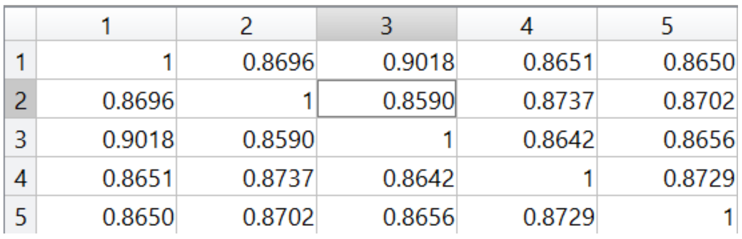
As explained in 'Question 1' the covariance matrix (Figure 20) is a way of showing covariance between the features and their linear relationship (as feature x changes, how feature y changes). With all values being positive we expect a positive trend of the data, 'cov(dataset)' was used to calculate the covariance matrix.



	1	2	3	4	5
1	9.5085	8.2547	8.5214	8.2368	7.9715
2	8.2547	9.4772	8.1030	8.3050	8.0061
3	8.5214	8.1030	9.3903	8.1771	7.9267
4	8.2368	8.3050	8.1771	9.5338	8.0543
5	7.9715	8.0061	7.9267	8.0543	8.9309

Figure 20 – Covariance Matrix for the given dataset.

Much the same as before in 'Question 1', the correlation matrix (Figure 21) is the strength and direction of the linear relationship across the features. With no value less than +0.8590 a strong, positive relationship is expected to exist; 'corrcoef(dataset)' calculated to correlation matrix.



	1	2	3	4	5
1	1	0.8696	0.9018	0.8651	0.8650
2	0.8696	1	0.8590	0.8737	0.8702
3	0.9018	0.8590	1	0.8642	0.8656
4	0.8651	0.8737	0.8642	1	0.8729
5	0.8650	0.8702	0.8656	0.8729	1

Figure 21 – Correlation Matrix for the given dataset.

4) *Report the number of classes = 6*

As the data was pre-labelled for what class each row is, finding the number of classes is as simple as finding the max value of class label as this will be the amount of classes. The MATLAB function 'max()' is used and as the class labels were split from the dataset and put into a separate matrix, we just find the max value in that matrix with the code [num_Classes = max(class_Labels)]. The number of classes for this dataset = 6.

5) *Further data analysis*

When looking at the above data analysis the means and standard deviation are all similar in value, we should expect each class or each feature to vary similarly as well as have similar values on average, we cannot be sure of the range of the data, but finding the min and max of each feature would show this if we studied further. From the correlation matrix and the covariance matrix we should expect the data to increase and stay fairly close together.

To analyse the data further as well as test to see if the above reports are correct a basic scatter plot (Figure 22) of features 1 and 2 was plotted using the 'scatter()' function. This shows that the linear relationship and other above assumptions are correct. The outlining datapoints from the centres of the data is responsible for bring the values in the correlation matrix away from +1.

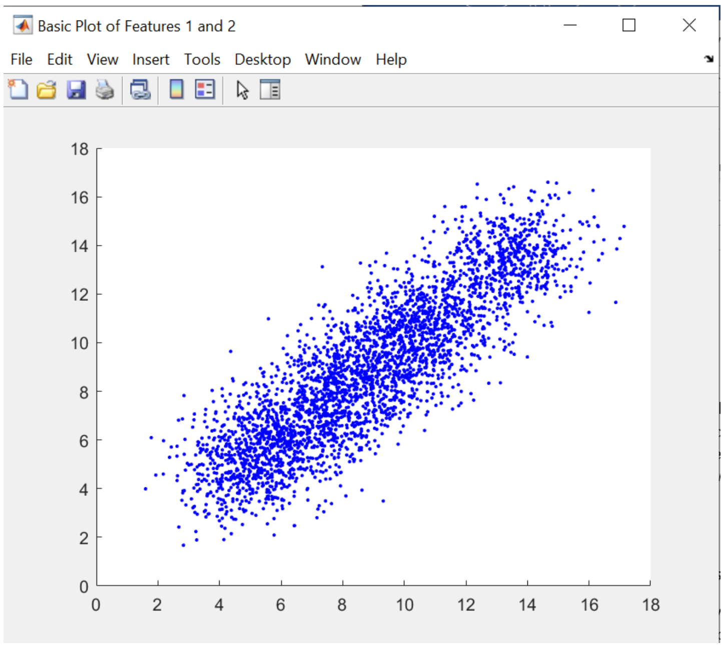


Figure 22 – Scatter plot of the first 2 features in the given dataset.

Data Pre-processing

The data for KNN classification must first be pre-processed, we must assign a training set from the original data, in our case this was 60% of the data. This was achieved by using N, which was the number of rows previously calculated in the data analysis. N now being the sample size is multiplied by 0.6 to give the Training size of the data. The data row index needs to be shuffled too, this is so the training data is allocated randomly; this is achieved by creating a random permutation of integers 1 to N(sample size) using the MATLAB function ‘randperm(N)’ that created a row vector of the shuffled indexes.

By using the randomised index’s, the training set and the testing set was created; the testing set is used after KNN classification to see how effective the current algorithm allocates its predicts class to a datapoint(row). The sets were made by iterating through the randomised indexes and copying over the relevant row to the training set until 60% of the original samples were allocated, then the rest of the samples are allocated to testing set. Class labels were split too, this was to make things clearer on what the code is working with/on, all tables here were given meaningful names to make coding and understanding easier.

To see fully how this was achieved refer to the code in ‘Question_2.m’

KNN Classifier – K Nearest Neighbour

The KNN classifier algorithm works by first generating a model from the training data for the future data’s class to use to be predicted. The model is generated by using the MATLAB function ‘fitcknn()’, this function returns a model that will predict class labels for the testing data in our case. It takes the training set, the class labels for the training set and the number of neighbours the model will work on (K) as parameters and generates a model of the training set.

The model will calculate predicted labels for the testing set with the function ‘predict()’ for each row of the testing set (or any future datapoints) to calculate the predicted class label. To predict the class label for each row of the dataset each data point is compared to K nearest neighbours (K amount of closest points) and their class labels. Whatever is the majority class in these datapoints is, the new datapoint will be assigned this class.

Confusion matrix’s show how effective the KNN classification performed, for each class you can see how much of the testing data was correctly classified, in this investigation a percentage of correct classification is used. The testing rows/objects actual classifications (original class label) are compared with the testing rows/objects predicted classifications (predicted class label). The performance of the KNN algorithm is tested in this way by using the percentage of correct classification, all others show the error percentage.

a) <u>Training:</u> use the training data set to train the KNN classifier. <u>Testing:</u> select K = 5 and test the classifier performance using the testing data. <u>Report:</u> the result of testing using the confusion matrix for testing the dataset when K = 5.

The training, testing and generation of the confusion matrix (Figure 23) was performed as explained above. In the code this is displayed in the command window.

		ACTUAL CLASS (%)					
PREDICTED CLASS (%)		*Class 1*	*Class 2*	*Class 3*	*Class 4*	*Class 5*	*Class 6*
	Class 1	84.0000	16.5138	0	0	0	0
	Class 2	16.0000	65.5963	8.9109	0	0	0
	Class 3	0	17.8899	79.7030	16.0000	0	0
	Class 4	0	0	10.8911	71.5556	13.5246	0
	Class 5	0	0	0.5450	12.4444	84.4262	2.0243
	Class 6	0	0	0	0	2.0492	97.9757

Figure 23 – Confusion matrix of KNN classifier algorithm when K = 5

When looking into the confusion matrix of KNN when K = 5, class 6 is the only very strongly correctly classified class, with all the others being mostly in the correct class however spilling over into near matched classes. This could all be due to the number of classes in the data without having much variance between the data, so the classes do somewhat overlap in a way making the training harder. However, as very distant classes are not classified (e.g. class 6 is never predicted as class 1,2,3,4) this does show how the algorithm is performing somewhat well as does most data points being classified correctly.

b) <u>Investigate how the classification result depends on K:</u> repeat the same procedure as before but using K = 7. For the testing dataset K = 7. <u>Report:</u> the result of testing using the confusion matrix.

The training, testing and generation of the confusion matrix (Figure 24) was performed as explained above. In the code this is displayed in the command window.

		ACTUAL CLASS (%)					
PREDICTED CLASS (%)		*Class 1*	*Class 2*	*Class 3*	*Class 4*	*Class 5*	*Class 6*
	Class 1	84.4444	17.8899	0	0	0	0
	Class 2	16.5556	66.0550	10.3960	0	0	0
	Class 3	0	16.0550	79.7030	16.0000	0	0
	Class 4	0	0	9.4059	71.5556	12.2951	0
	Class 5	0	0	0.4950	12.4444	85.6557	2.4291
	Class 6	0	0	0	0	2.0492	97.5709

Figure 24 – Confusion matrix of KNN classifier algorithm when K = 7

Looking at the confusion matrix for K = 7 is very similar to before, with the results being matched very closely; the above analysis applies just the same here too with the number of classes skewing the result.

c) <u>Using the overall percentage of correct classifications of testing data compare two KNN classifications:</u> For K = 5 and for K = 7.

To calculate the overall percentage of correct classifications, the total number of correct predicted classes are divided by how many predictions were made then multiplied by 100 to give a percentage; the code used = [length(find((Pred_KNN_Label-Class_Lab_Testing')==0))/length(Class_Lab_Testing)*100;] .

When K = 5 the overall percentage of correct classifications = 80.9699%
When K = 7 the overall percentage of correct classifications = 81.2638%

With the percentage being so close together, the overall performance of KNN classification is near identical, with K = 7 being slightly improved by 0.2939% of correct classifications. The result is negligible, this is proved by how close the confusion matrix's are when compared against each other.

Conclusion

Concluding, for both K = 5 and K = 7, as for performance of the KNN algorithm the difference is minimal (shown with the percentage of correct classifications), this could be due to the number of classes for the data. When plotting the first two features we can see how the classes are grouped easier as they are now visualised in a easy to visualise format (Figure 25). As suspected with the confusion matrix, the classes overlap a fair amount, this will cause difficulty with classifying new data as there isn't really a clear definition on separation of classes close together, it also makes sense why classes further apart don't get incorrectly classifies as the KNN algorithm is working as it should.

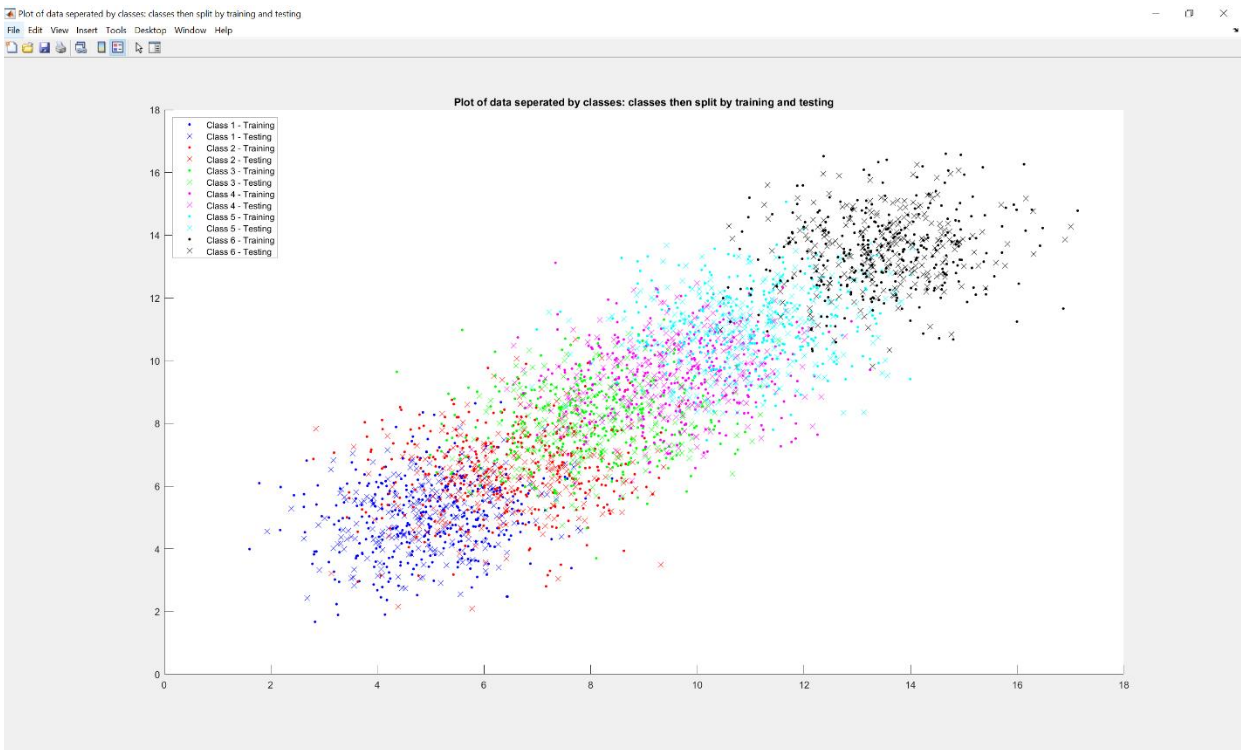


Figure 25 – Plot of features 1 and 2 of KNN classifier algorithm's training set classes and testing set generated classes when K = 7, shown better when run in MATLAB