Section 1 Investigate the biological data set and perform dimensionality reduction. (AA4, 7 marks)

The attributes have 8 dimensions (excluding the category attribute). In this section, you will investigate whether the dimensions be reduced without losing too much information.

Task:

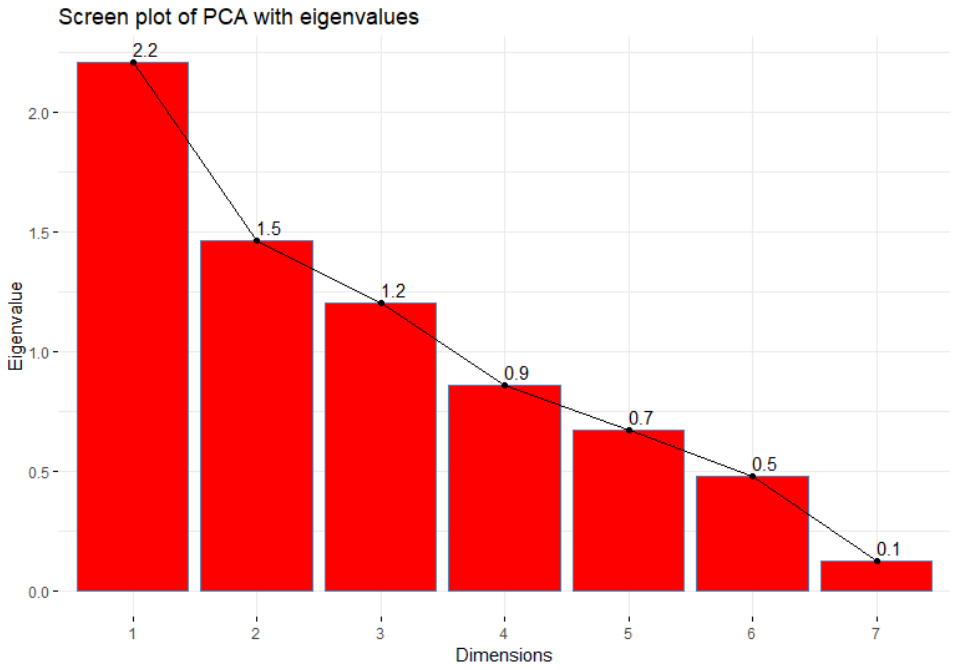
a) Perform PCA on the first 8 attributes, write down the results. (2 marks)

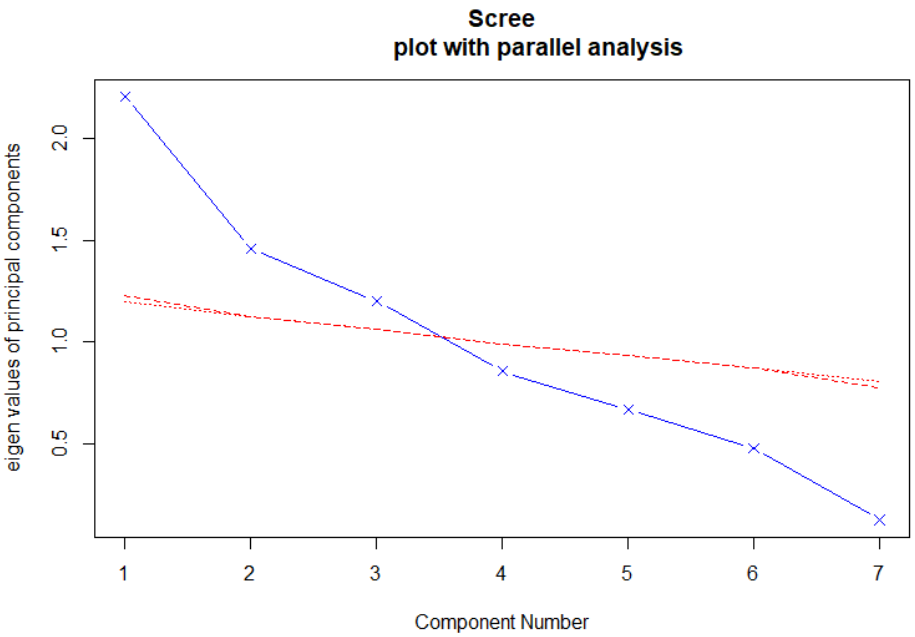
```
Principal Components Analysis
> #Preforming PCA
> pca_ecoli <- prcomp(ecoli.scaled, center = TRUE,scale. = TRUE)
> pca_ecoli
Standard deviations (1, .., p=7):
[1] 1.4851348 1.2087972 1.0961308 0.9258289 0.8181948 0.6918498 0.3555607

Rotation (n x k) = (7 x 7):
            PC1         PC2          PC3         PC4          PC5         PC6          PC7
mcg  0.43301833 -0.37265574  0.292199211 -0.06707282  0.004909846 -0.75965086  0.08154845
gvh  0.22398568 -0.47029644  0.552886681 -0.14655068 -0.050196250  0.60363588  0.18599716
lip  0.10951653 -0.48177748 -0.492232610  0.20125402  0.671280372  0.10724976  0.10478843
chg  0.01816912 -0.45560856 -0.545179203 -0.20553058 -0.671631593  0.03639389 -0.01462969
aac  0.34590722  0.04745278  0.008181552  0.88426143 -0.294591756  0.09557701 -0.01385432
alm1 0.60707010  0.16428058 -0.110458413 -0.24341844  0.093650818  0.16949665 -0.70393508
alm2 0.51167924  0.41354610 -0.239070354 -0.22384663 -0.013968426  0.08867463  0.67219322
>
```

b) Write down the eigenvalues for each of the attributes, sorted in descending order. (1 mark)

2.2, 1.5, 1.2, 0.9, 0.7, 0.5, 0.1

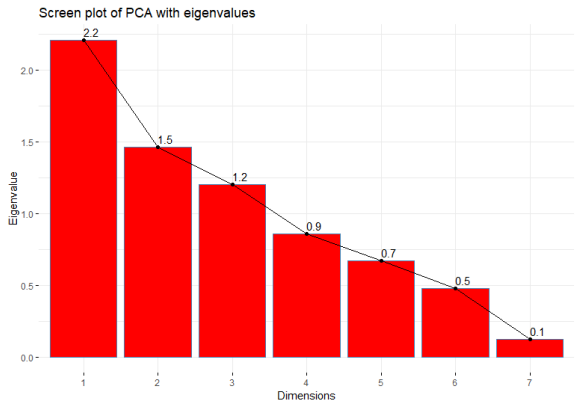
c) Create a Scree plot using the eigenvalues found in (b). (1 mark)

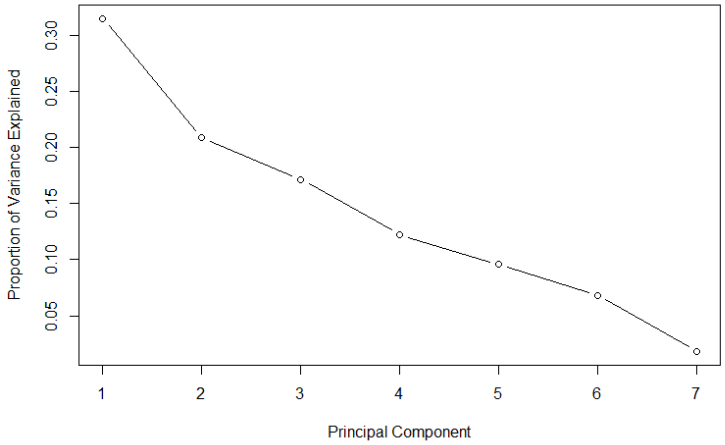




d) Chose the number of dimensions to keep, given a sound explanation as to how you arrived at this decision. Which of the dimensions did you choose? Why?

I chose to keep the first 3 dimensions because the they have eigenvalues greater than 1, and a factor with an eigenvalue greater than 1 explains as much variance as a single variable, and it is known that only factors that explain at least the same amount of variance as a single variable are worth keeping. You can see the plot with eigen values below as well as variance:





Paste the first 10 rows of the data after dimensionality reduction has been performed. (3 marks)

```
> #first 10 rows after dimensionality reduction
> ecoli_dr <- ecoli.scaled[c(-4,-5,-6,-7)]
> head(ecoli_dr, n=10)
    mcg  gvh  lip
1  0.49 0.29 0.48
2  0.07 0.40 0.48
3  0.56 0.40 0.48
4  0.59 0.49 0.48
5  0.23 0.32 0.48
6  0.67 0.39 0.48
7  0.29 0.28 0.48
8  0.21 0.34 0.48
9  0.20 0.44 0.48
10 0.42 0.40 0.48
>
```

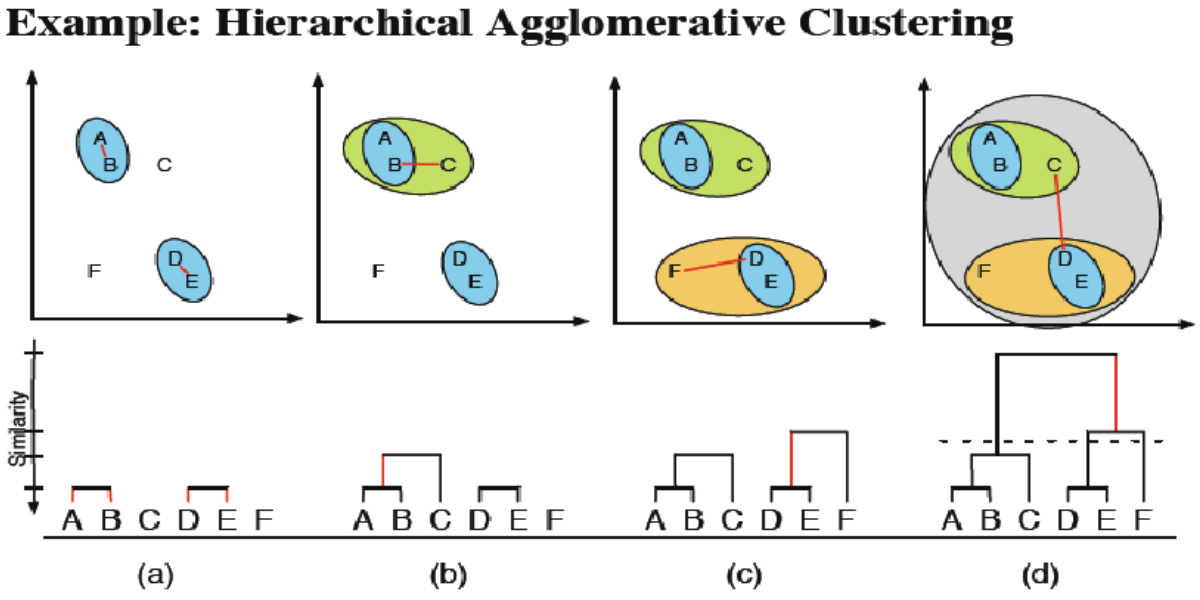
Section 2 Find and analyse clusters in the data set. (KU2, 5 marks) (KU6, 5 marks) (KU7, 5 marks) (AA5, 7 marks) (SE3, 10 marks)

Task 1: In a paragraph, describe the k-means algorithm (1 mark).

The k-means clustering is a type of unsupervised learning and it is a proper classification algorithm for compact clusters, however it is sensitive to outliers and noise. It only uses numerical attributes and it accepts various inputs such as: 1. an input which is a set of feature vectors, 2. The number of clusters to be detected (k), 3. A convergence threshold. It's goal is to find groups in the data, with the number of groups represented as K(mentioned above). The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Usually, clustering is used when you want to learn and explore your dataset.

In a paragraph, describe hierarchical clustering (1 mark).

Clustering algorithms have a goal to create clusters in which entities within such clusters should be as similar as possible whilst entities in one cluster should not be similar at all from entities in another. Hierarchical clustering's aim is to combine the two nearest clusters into a larger cluster repeatedly. It starts by treating each observation as a separate cluster, then it repeatedly executes the following two steps: 1. Find the two clusters which are closest together, 2. Merge the two most similar clusters. The process repeats itself until all clusters are merged together.



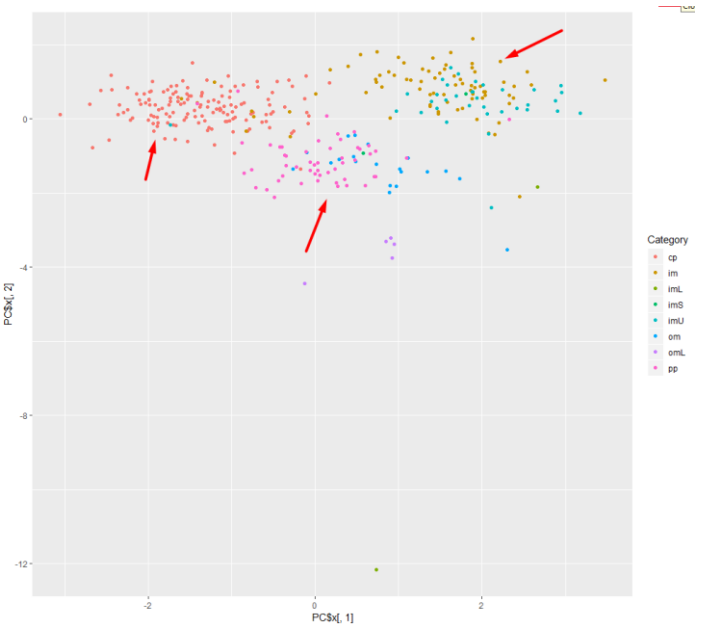
Describe at least 3 differences between k-means and hierarchical clustering. Describe also how both k-means and hierarchical clustering both make use of metrics. (2 marks).

1. K-means clustering can handle big data well, but hierarchical clustering can't handle big data well.
2. K-means requires the user to know what K is (number of clusters you want to divide in your data), however with hierarchical clustering this is not necessary, and the user can stop at whatever number of clusters they find appropriate as the dendrogram will provide more information.
3. K-means begins with a random choice of clusters and thus the result produced by running the algorithm repeatedly may differ, whilst with hierarchical clustering the results are reproducible.

K-means uses the Euclidean distance as a metric whilst Hierarchical clustering can use any of the following metrics: Euclidean distance, Squared Euclidean distance, Manhattan distance, Maximum distance, Mahalanobis distance. The choice of an appropriate metric (for hierarchical clustering) will influence the shape of the clusters. In hierarchical clustering, the metric will take as input pairs of clusters (a single element can be considered to be a cluster of size one), where X is the set of all clusters, subject to some conditions.

For k-means (only), describe the expected results on your data set. (1 mark)

After performing PCA, I now know I only need to use the first 3 attributes from my data. When plotting the categories with my 3 PCA scores, I can see a clear pattern of how the categories will be clustered into 3 separate clusters as seen below:



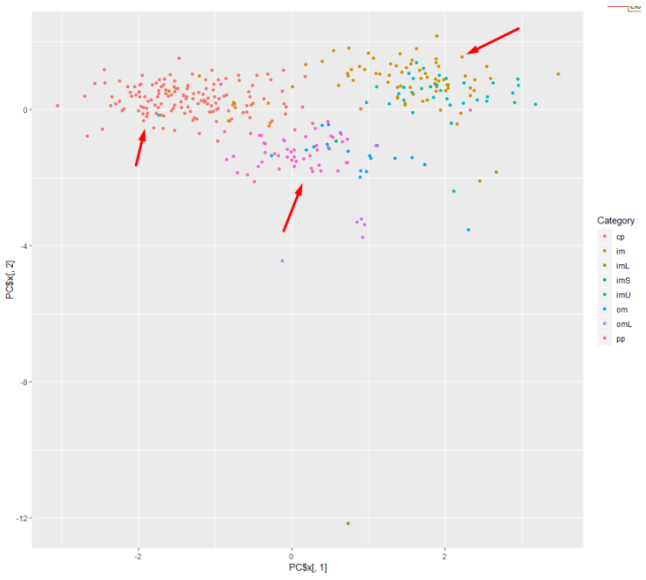
So, I think that the attributes which have the least results such as imL and imS, omL and om will all be grouped in one cluster, imU and im will be in another cluster and cp and pp will be in another cluster.

Task 2: Identify specific issues and problems that you will face when applying k-means to the scenario. (2 marks)

It is possible to end up with overfit of data. As well as, not choosing the correct amount of clusters. In addition, the number of classes of k-means are not equal and k-means does not care about cluster cardinalities. Thus, for this scenario, where we may have varying density in the data set, then two clusters of the same area may not have the same number of elements.

Discuss how you will choose the value of k (the number of clusters), justifying your choice. (1 marks)

I will choose the number of clusters by using the previous result obtained from the principal component analysis, which resulted in performing dimensionality reduction and only keeping the first 3 dimensions, due to their variance. I will choose those same 3 dimensions since they explain the majority of the variance and when looking at the image below which displays the categories, it is evident that there are 3 main clusters.



I also used NBClust to check what the optimal number of clusters is and the result was 3, as seen below:

```
package 'flexclust' was built under R version 3.5.2
> res<-NbClust(ecoli_kmeans[c(-1,-9)], diss=NULL, distance = "euclidean", min.nc=2, max.nc=8,
+              method = "kmeans", index = "kl")
> res$All.index
      2       3       4       5       6       7       8
 1.6936 12.0493  0.6456  5.9029  0.1047  0.7416  3.3778
> res$Best.nc #3
Number_clusters     Value_Index
         3.0000         12.0493
>
```

Discuss and describe how you will initialise the algorithm. Give justification of your initialisation method. (2 marks)

I will initialise the algorithm by using the nstart, which attempts multiple initial configurations and reports on the best one. In my case I am going to set nstart as 25 which will generate 25 initial configurations. This is done because the initial centres can converge badly.

Discuss the metric that you will be using for the analysis. Give justification of the metric used. (2 marks)

The metric being used is Error sum of squares (SSE). This is due to kmeans using the Hartigan and Wong algorithm by default, and this algorithm uses the within-cluster sum of squares of errors (SSE). This is a measure of variation within a cluster, and if all cases in a cluster turn out to be identical, then the SSE will be equal to 0.

$$SSE = \sum_{i=1}^{n} (x_i - \bar{x})^2$$

https://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html <- This is the link I used that states that the method 'kmeans' in R uses Hartigan and Wong by default.

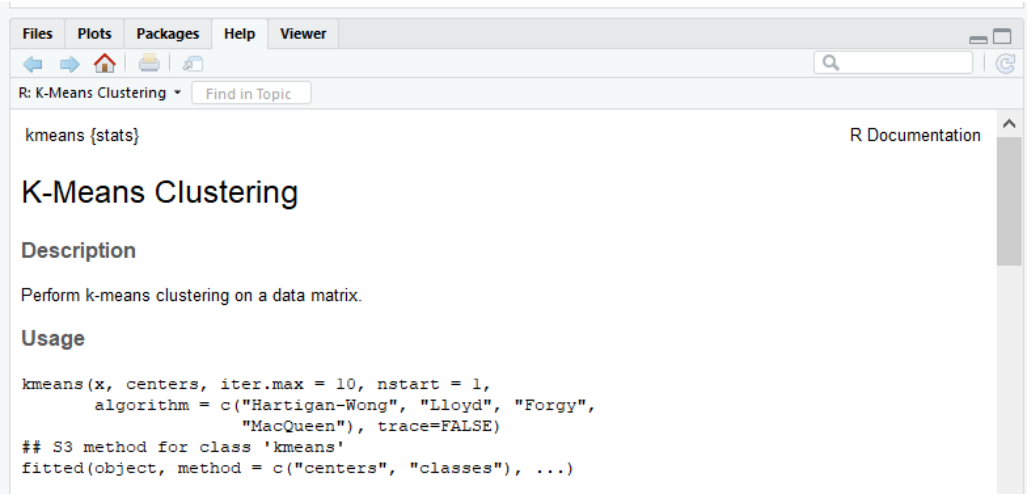
Task 3: Using R, or otherwise, implement if necessary, and apply k-means to the scenario.

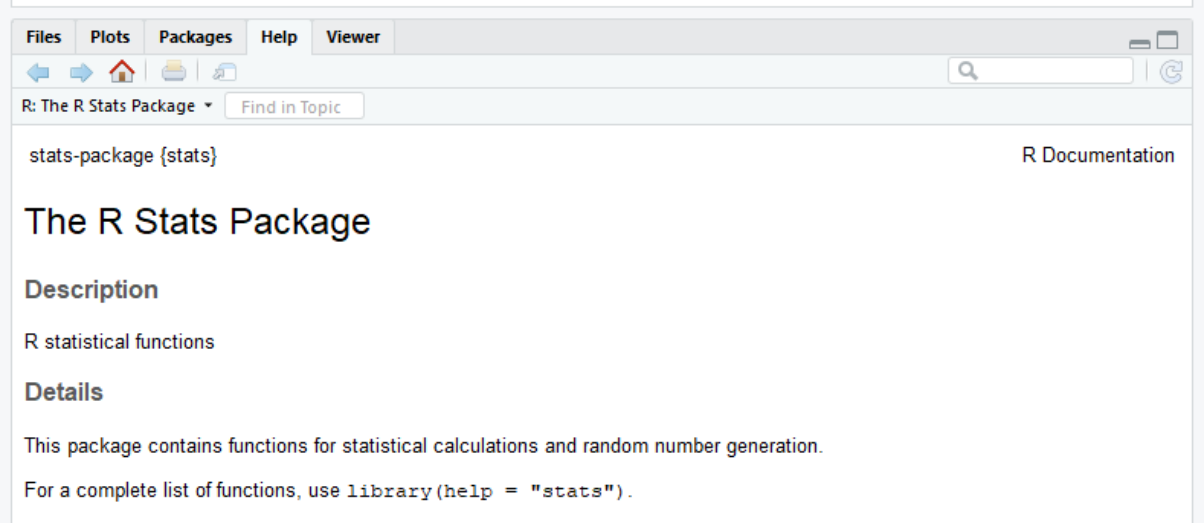
If you are using an existing library or tool that supplies the actual code, then for the purposes of the assignment, the implementation is the code/script/formula that calls this method.

Add screenshots of the k-means implementation. There is no need to take screenshots of the supporting code, only the k-means and it initialisation.

The R inbuilt library called stats was used to implement kmeans, in addition to the kmeans method.

```
#Applying Kmeans algorithm
help(kmeans)
set.seed(235)
ecolicluster <- kmeans(PC$x[,1:3], 3, nstart = 25)
```





Apply your implementation to the data set, and describe your resulting clusters, giving screenshots of the cluster visualisation.

```
library(ClusterR)
library(ggplot2)
library(factoextra)
library(cluster)
library(NbClust)

#TASK 3
#actual dataset being tampered with
ecoli_kmeans <- read.csv('C:/Users/Fran/Documents/Bsc/3BSC/3bsc_pendrive/FirstSem/StatisticsForCS/MonteCarlo_PCA_Home/data.csv')

names(ecoli_kmeans) <- c("SequenceName", "mcg", "gvh", "lip","chg", "aac","alm1", "alm2", "Category")

head(ecoli_kmeans)
```

First, I removed the dimensions I don't need such as Sequence name and Category, and only kept the first three dimensions after that, since my PCA results lead me to choosing the first 3 dimensions. I then set a seed and created the cluster by using the kmeans method and passed the dataset as the first parameter, I set the centres (number of clusters) to 3, and I set the nstart (which is stating how many random sets should be chosen) to 25 just to specify a random starting point (this is further explained below).

```
#Applying Kmeans algorithm
help(kmeans)
set.seed(235)
ecolicluster <- kmeans(PC$x[,1:3], 3, nstart = 25)
```

```
K-means clustering with 3 clusters of sizes 162, 65, 109

Cluster means:
        mcg         gvh        lip
1 -0.8873777 -0.57246643 -0.1748815
2  0.8691763  1.54098657  0.7291215
3  0.8005387 -0.06811527 -0.1748815

Clustering vector:
  [1] 1 1 3 3 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 1 3 1 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1
 [65] 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 1 1 1 1 1 2 1 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1
[129] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 3 1 3 1 3 1 1 3 3 1 3 1 2 3 3 1 3 3 3 3 1 1 3 1 1 3 1 3 3 3 1 3 3 2 1 2 1 3 3 1 3 1 2
[193] 1 1 3 3 1 3 3 1 3 1 1 1 1 2 3 1 1 3 3 3 3 1 1 1 1 3 2 3 3 3 2 2 2 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 1 3 3 3 3 3 2 3 3 3 3
[257] 3 3 3 2 2 2 2 2 2 2 2 3 2 2 2 3 2 3 2 3 3 2 3 2 2 2 2 2 3 3 2 2 3 2 2 2 3 2 3 2 2 2 2 2 1 3 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3
[321] 2 2 2 2 2 1 2 3 2 2 1 3 3 3 3 2

Within cluster sum of squares by cluster:
[1] 104.38044 337.08580  66.21834
 (between_SS / total_SS =  49.5 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

It is known that we have 8 different categories (the protein localization sites), however as you can see below, the class distribution is skewed, and the bulk of proteins which reside in the same group are in the first three: cytoplasm, inner membrane without signal sequence and periplasm. The rest have few in each category group, especially in the last three.

```
9. Class Distribution. The class is the localization site. Please see Nakai &
                       Kanehisa referenced above for more details.

   cp  (cytoplasm)                                 143
   im  (inner membrane without signal sequence)     77
   pp  (perisplasm)                                 52
   imU (inner membrane, uncleavable signal sequence)  35
   om  (outer membrane)                             20
   omL (outer membrane lipoprotein)                  5
   imL (inner membrane lipoprotein)                  2
   imS (inner membrane, cleavable signal sequence)   2
```

In addition, since the starting assignments are random, I specified the nstart to be 25, meaning that it will be run 25 different times with random starting assignments and then proceed the select the one with the lowest within the cluster variation.

```
----------------------------------------
> ecolicluster
K-means clustering with 3 clusters of sizes 151, 79, 106

Cluster means:
         PC1         PC2         PC3
1 -1.3665575  0.3093815 -0.2270837
2  0.2795354 -1.5204840  0.9039345
3  1.7383668  0.6924682 -0.3501999
```

```
Within cluster sum of squares by cluster:
[1] 109.6213 440.0190 139.5040
 (between_SS / total_SS =  57.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "with
> str(ecoliCluster)
List of 9
 $ cluster     : int [1:336] 1 1 1 2 1 1 1 1 1 1 ...
 $ centers     : num [1:3, 1:3] -1.367 0.28 1.738 0.30
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "1" "2" "3"
  .. ..$ : chr [1:3] "PC1" "PC2" "PC3"
 $ totss       : num 1631
 $ withinss    : num [1:3] 110 440 140
 $ tot.withinss: num 689
 $ betweenss   : num 942
 $ size        : int [1:3] 151 79 106
 $ iter        : int 2
 $ ifault      : int 0
 - attr(*, "class")= chr "kmeans"
> table(ecoliCluster$cluster, ecoli_kmeans$Category)

     cp  im imL imS imU  om omL  pp
  1 138   8   0   0   1   0   0   4
  2   4   1   1   1   0  20   5  47
  3   1  68   1   1  34   0   0   1
>
```

The above image shows the 3 clusters and how they group the 8 categories into the 3 clusters. I validated that this made sense because I know that the data should have 143 instances of cp (cytoplasm) and as you can see above, cp is grouped into 138,4,1 which make 143. We can then visualize the newly created clusters in the images below:

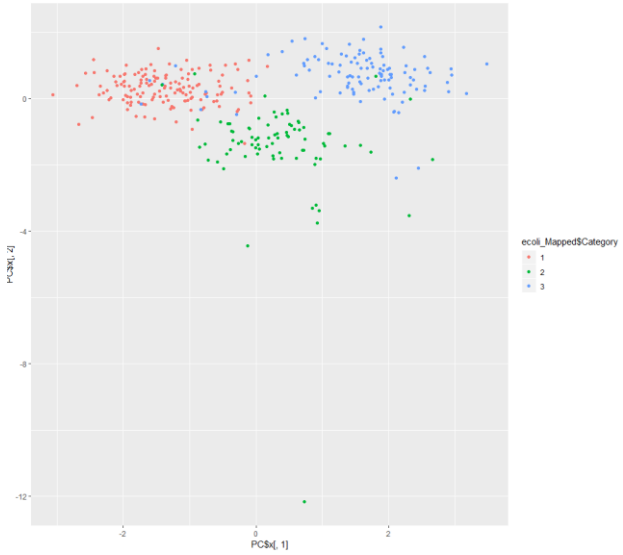


Figure 1: With label

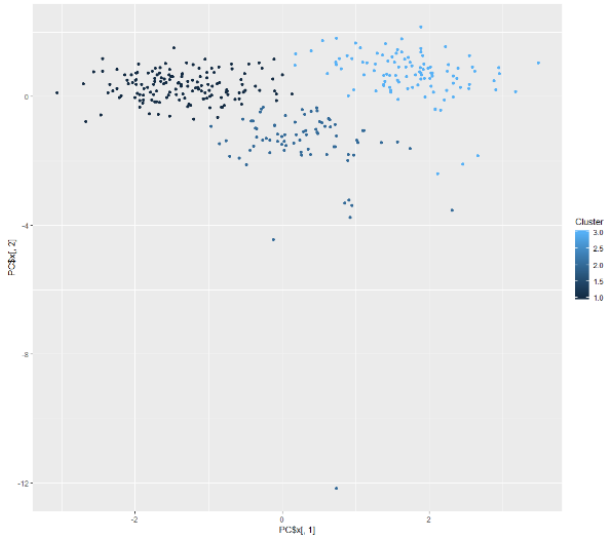
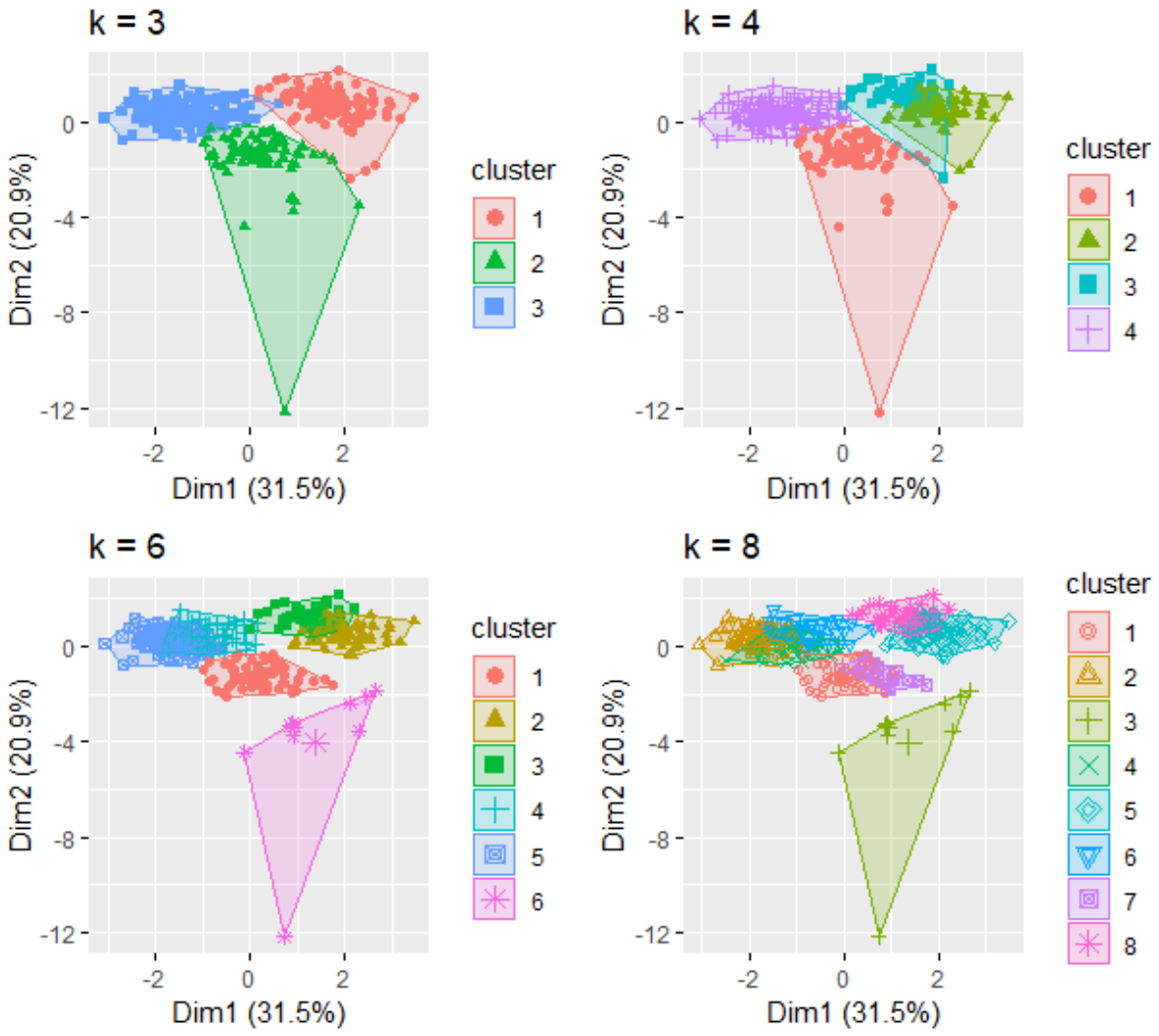


Figure 2: With cluster number

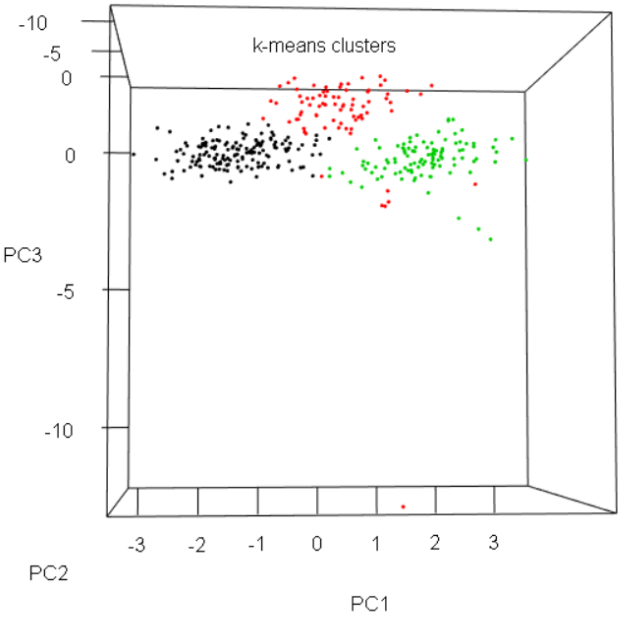
Task 4: Visually inspect your cluster visualisation. Comment whether the resulting clusters appear to be good, explaining why. (1 mark)

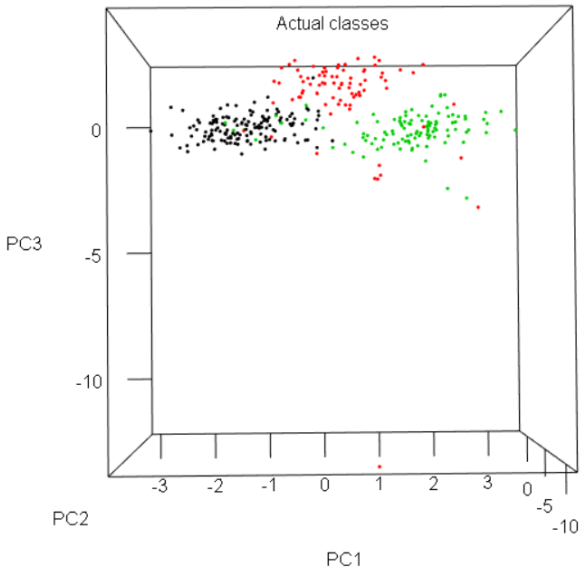
The clusters appear to be good and they make sense since I manually checked if there should be 143 instances of cluster 1 (which is the cp) and the data confirms that this is true. Additionally, I did this for the rest of the 7 remaining clusters and confirmed the validity.

I also created some comparisons, in 2D and 3D, of different k clusters and from the visualization (as seen below), it is notable that setting k to 3 was the best choice, as it groups the categories into 3 clusters, and this can be confirmed in the 3D plot.



As well as 3D plots:





What is the accuracy of the cluster for identifying the correct variety for the given sample? (2 mark).

Note: due to the way that the problem is worked out, the clusters identified by k-means will require to be mapped in such a way as to reflect the categories in the dataset.

I mapped each category to cluster 1, 2 and 3 by using a method I created to map the category accordingly by group and created a crosstable with the ecoli_mapped$category and ecolic$Cluster

```
Total Observations in Table:  336

                     | ecoliCluster$cluster
ecoli_Mapped$Category |         1 |         2 |         3 | Row Total |
----------------------|-----------|-----------|-----------|-----------|
                    1 |       138 |         4 |         1 |       143 |
----------------------|-----------|-----------|-----------|-----------|
                    2 |         4 |        74 |         3 |        81 |
----------------------|-----------|-----------|-----------|-----------|
                    3 |         9 |         1 |       102 |       112 |
----------------------|-----------|-----------|-----------|-----------|
         Column Total |       151 |        79 |       106 |       336 |
----------------------|-----------|-----------|-----------|-----------|
```

As you can see above, 4 + 1 + 4 + 9 +1 + 3 are the outliers. The accuracy can be calculated simply with this cross table by adding the outliers -> 23 and saying 1 –(23/336), which is 93% accuracy. This can be confirmed below by the use of a confusion matrix:

```
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3
         1 138   4   1
         2   4  74   3
         3   9   1 102

Overall Statistics

               Accuracy : 0.9345
                 95% CI : (0.9025, 0.9585)
    No Information Rate : 0.4494
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.8988
 Mcnemar's Test P-Value : 0.06018

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.9139   0.9367   0.9623
Specificity            0.9730   0.9728   0.9565
Pos Pred Value         0.9650   0.9136   0.9107
Neg Pred Value         0.9326   0.9804   0.9821
Prevalence             0.4494   0.2351   0.3155
Detection Rate         0.4107   0.2202   0.3036
Detection Prevalence   0.4256   0.2411   0.3333
Balanced Accuracy      0.9434   0.9547   0.9594
> |
```

Accuracy: 93%.

Sensitivity for class one is 0.91, class 2 is 0.94 and class 3 is 0.96.

Specificity for class one is 0.97, class 2 is 0.97 and class 3 is 0.95.

The accuracy identified above might not reflect the accuracy for new kernels that were not used in the original sample to build the clusters.

Split the original data set into a training set and a test set and perform the same test again.

Find the sensitivity, specificity and accuracy of the clusters. (4 marks)

I first split the data into 75% training and 25% test as seen in the image below:

```
newEcoli.rows <- nrow(newEcoli)
newEcoli.sample <- sample(newEcoli.rows, newEcoli.rows*0.75)

newEcoli.train <- newEcoli[newEcoli.sample,]
newEcoli.test <- newEcoli[-newEcoli.sample,]

newEcoli.train <- kcca(newEcoli.train[,2:8], k=3, kccaFamily("kmeans"))
```

I then proceeded to creating my model using a cross table to obtain my predictions:

	newEcoli.test.pred			
newEcoli.test$Category	1	2	3	Row Total
1	34	22	12	68
2	1	10	4	15
3	0	1	0	1
Column Total	35	33	16	84

And confusion matrix results:

```
Overall Statistics

               Accuracy : 0.5238
                 95% CI : (0.4119, 0.634)
    No Information Rate : 0.4167
    P-Value [Acc > NIR] : 0.03071

                  Kappa : 0.1933
 Mcnemar's Test P-Value : 3.262e-07

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.9714   0.3030   0.0000
Specificity            0.3061   0.9020   0.9853
Pos Pred Value         0.5000   0.6667   0.0000
Neg Pred Value         0.9375   0.6667   0.8072
Prevalence             0.4167   0.3929   0.1905
Detection Rate         0.4048   0.1190   0.0000
Detection Prevalence   0.8095   0.1786   0.0119
Balanced Accuracy      0.6388   0.6025   0.4926
>
```

So, from 93%, the **Accuracy** is now 52%. This is to be expected since we are now using test data. The **Sensitivity** for class one is 0.97, class 2 is 0.30 and class 3 is 0.

Specificity for class one is 0.31, class 2 is 0.90 and class 3 is 0.98.

Apart from the statistics mentioned above, research another method to evaluate the resulting clusters. Write down information about the selected method. (1 mark)

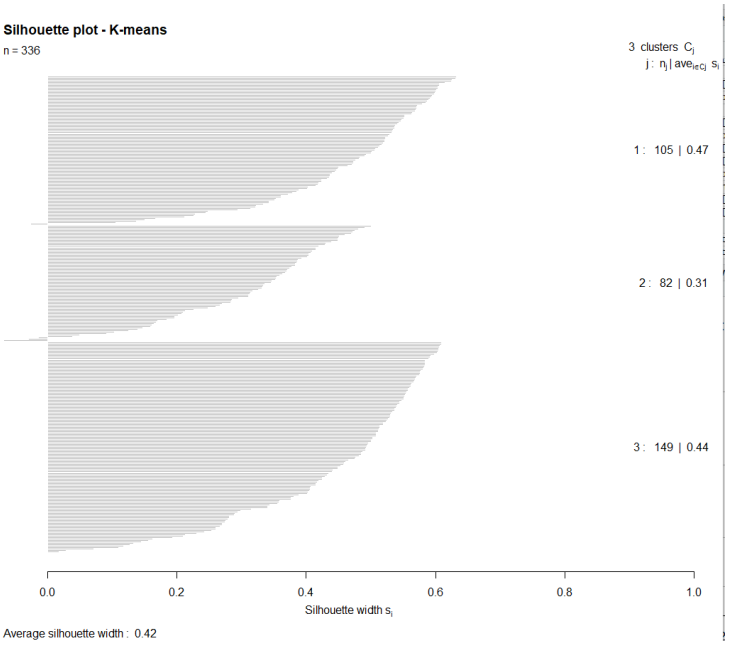
Another method is to use the Silhouette method which measures how well an observation is clustered and it estimates the average distance between clusters. The silhouette plot displays a measure of how close each point in one cluster is to points in the neighbouring clusters.

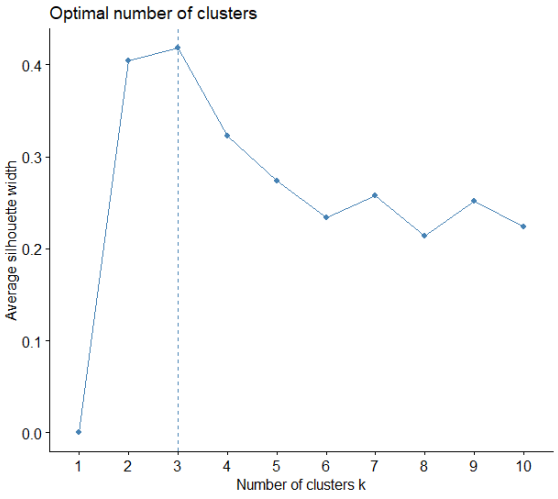
In short, the silhouette method determines how well each object lies within its cluster. A high average (closer to 1 since a silhouette ranges from -1 to 1) silhouette width indicates a good clustering.

Use the selected method on your clusters. (2 mark)

My silhouette method shows that the clusters are good as they have an average width of 0.42.

I displayed the silhouette method in two ways:





Part 2 The Length of a Random Line

Section 1

Task 1: Implement a Simple Monte Carlo algorithm to estimate the mean length of a line in the unit square (2-dimensional scenario), using a sample size of 20,000. (2 marks)

For both cases, find the 99% confidence interval. (3 marks)

```
/// <summary>
/// Returns the length l, of a line in 2-dimensions, using the Euclidean distance,
/// Which is given by this equation : Math.Sqrt(Math.Pow((x1 - x2), 2) + Math.Pow((y1 - y2), 2));
/// </summary>
/// <param name="x1"></param>
/// <param name="x2"></param>
/// <param name="y1"></param>
/// <param name="y2"></param>
/// <returns></returns>
private static double TwoD(double x1, double x2, double y1, double y2)
{
    return Math.Sqrt(Math.Pow((x1 - x2), 2) + Math.Pow((y1 - y2), 2));
}
```

```
/// <summary>
/// Returns the length l, of a line in 3-dimensions, using the Euclidean distance,
/// Which is given by this equation : Math.Sqrt(Math.Pow((x1 - x2), 2) + Math.Pow((y1 - y2), 2) + Math.Pow((z1 - z2), 2));
/// </summary>
/// <param name="x1"></param>
/// <param name="x2"></param>
/// <param name="y1"></param>
/// <param name="y2"></param>
/// <param name="z1"></param>
/// <param name="z2"></param>
/// <returns></returns>
private static double ThreeD(double x1, double x2, double y1, double y2, double z1, double z2)
{
    return Math.Sqrt(Math.Pow((x1 - x2), 2) + Math.Pow((y1 - y2), 2) + Math.Pow((z1 - z2), 2));
}
```

I created two methods which calculate the length of a 2D and 3D line, as seen above.

I then created a method to calculate the mean length of a line (3D and 2D) as seen below:

```
public static void SimpleMonteCarlo()
{
    int n = 20000; //n is the number of simulations
    double l = 0;  //length of a line
    double l_three_d = 0; //length of a line
    double[] distances = new double[n]; // here the distance is stored in an array for each iteration of n
    double[] distancesThreeD = new double[n]; // here the distance is stored in an array for each iteration of n

    //2D --> repeat the experiment n (20k) times
    for (int i = 0; i < n; i++)
    {
        //get a point (x1,x2,y1,y2) at random
        double x1 = r.NextDouble();
        double x2 = r.NextDouble();
        double y1 = r.NextDouble();
        double y2 = r.NextDouble();

        //2D
        double result = TwoD(x1, x2, y1, y2);
        l = result;
        distances[i] = l;
    }

    //3D --> repeat the experiment n (20k) times
    for (int i = 0; i < n; i++)
    {
        //get a point (x1,x2,y1,y2) at random
        double x1 = r.NextDouble();
        double x2 = r.NextDouble();
        double y1 = r.NextDouble();
        double y2 = r.NextDouble();
        double z1 = r.NextDouble();
        double z2 = r.NextDouble();

        //3D
        double result2 = ThreeD(x1, x2, y1, y2, z1, z2);
        l_three_d = result2;
        distancesThreeD[i] = l_three_d;
    }
```

```
#region average
double averageOfLine = Queryable.Average(distances.AsQueryable());
double averageOfLineThreeD = Queryable.Average(distancesThreeD.AsQueryable());
#endregion

#region standard dev
//2D
double sum2D = distancesThreeD.Sum(d => Math.Pow(d - averageOfLine, 2));
double sd = Math.Sqrt((sum2D) / (distances.Count() - 1));

//3D
double sum3D = distancesThreeD.Sum(d => Math.Pow(d - averageOfLineThreeD, 2));
double sd_3d = Math.Sqrt((sum3D) / (distancesThreeD.Count() - 1));
#endregion

#region confidence interval
//2D
//x = average, theta = sd, and n = 20k
//2.58 * (sd / sqrt(20k) ==[141.421356237]) = 2.58 * sd (0.sd) = margin of error
double marginOfError = 2.58 * (sd / Math.Sqrt(n));
double lowerEndOfInterval = averageOfLine - marginOfError;//lower end of the interval is average - margin of error
double UpperEndOfInterval = averageOfLine + marginOfError;//and upper end is average + margin of error

//3D
double marginOfError_3d = 2.58 * (sd_3d / Math.Sqrt(n));
double lowerEndOfInterval_3d = averageOfLineThreeD - marginOfError_3d;//lower end of the interval is average - margin of error
double UpperEndOfInterval_3d = averageOfLineThreeD + marginOfError_3d;//and upper end is average + margin of error
#endregion


Console.WriteLine("n = " + n);
Console.WriteLine("2D Results: \n");
Console.WriteLine("Average: " + averageOfLine);
Console.WriteLine("Standard Deviation: " + sd);
Console.WriteLine("μ: " + averageOfLine + " ± " + marginOfError);
Console.WriteLine("lower end of interval = " + lowerEndOfInterval);
Console.WriteLine("upper end of interval = " + UpperEndOfInterval);
Console.WriteLine("You can be 99% confident that the population mean (μ) falls between" + lowerEndOfInterval + " and " + UpperEndOfInterval);
```

And the results:

```
Simple monte carlo:

n = 20000
2D Results:

Average: 0.521530737450397
Standard Deviation: 0.288497456675645
μ: 0.521530737450397 ± 0.00526316150563686
lower end of interval = 0.51626757594476
upper end of interval = 0.526793898956034
You can be 99% confident that the population mean (μ) falls between0.51626757594476 and 0.526793898956034

****

3D Results:

Average of 3d: 0.661729461477541
Standard Deviation 3d: 0.252139083555535
μ: 0.661729461477541 ± 0.00459986279923442
lower end of interval = 0.657129598678307
upper end of interval = 0.666329324276775
You can be 99% confident that the population mean (μ) falls between0.657129598678307 and 0.666329324276775

****

Press any key to continue ...
```

Section 2

Task 1: a) Implement a Simple Monte Carlo algorithm to analyse the variation in this scenario. (5 marks)

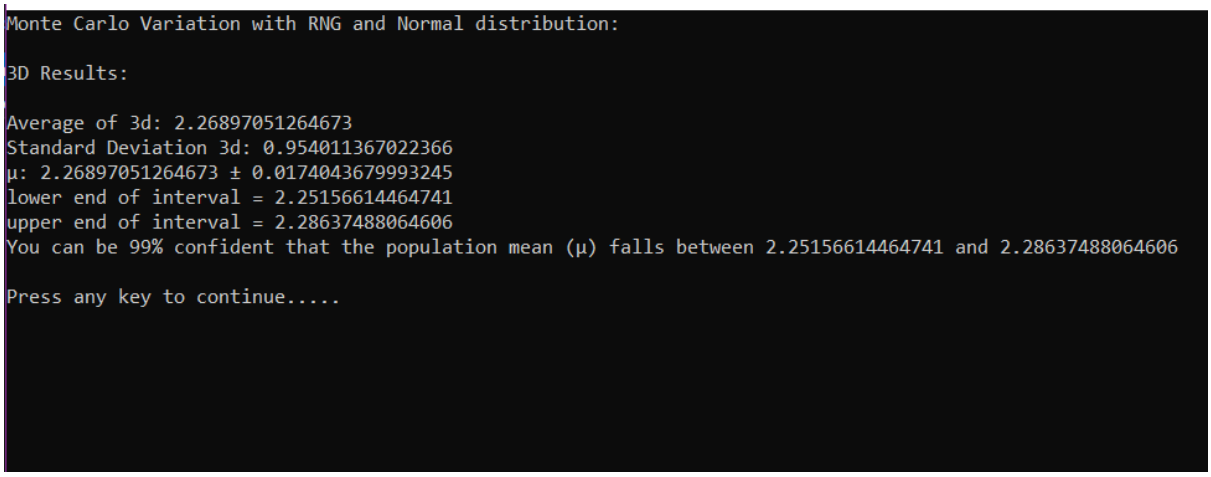
```
public static void MonteCarlo_Variation()
{
    int n = 20000; //n is the number of simulations
    double l_three_d = 0; //length of a line
    double[] distancesThreeD = new double[n]; // here the distance is stored in an array for each iteration of n

    BoxMullerNormal b = new BoxMullerNormal();
    //repeat the experiment n (20k) times
    for (int i = 0; i < n; i++)
    {
        //get a point (x1,x2,y1,y2) at random
        double x1 = b.GetRandom();
        double x2 = b.GetRandom();
        double y1 = b.GetRandom();
        double y2 = b.GetRandom();
        double z1 = b.GetRandom();
        double z2 = b.GetRandom();

        //3D
        double result2 = ThreeD(x1, x2, y1, y2, z1, z2);
        l_three_d = result2; //get length of a 3d line
        distancesThreeD[i] = l_three_d;
    }

    #region average
    double averageOfLineThreeD = Queryable.Average(distancesThreeD.AsQueryable());
    #endregion

    #region standard dev
    double sum = distancesThreeD.Sum(d => Math.Pow(d - averageOfLineThreeD, 2));
    double sd_3d = Math.Sqrt((sum) / (distancesThreeD.Count() - 1));
    #endregion

    #region confidence interval
    double marginOfError_3d = 2.58 * (sd_3d / Math.Sqrt(n));
    double lowerEndOfInterval_3d = averageOfLineThreeD - marginOfError_3d;//lower end of the interval is average - margin of error
    double UpperEndOfInterval_3d = averageOfLineThreeD + marginOfError_3d;//and upper end is average + margin of error
    #endregion

    display results
```

In the method above, I am doing more or less the same as in the Simple version, however instead I am using a Box Muller random number generator.

```
public class BoxMullerNormal : MonteCarlo
{
    private MathNet.Numerics.Distributions.Normal normal;

    public BoxMullerNormal(double mean = 0, double std = 1.0)
    {
        normal = new MathNet.Numerics.Distributions.Normal(mean, std);
    }

    public override dynamic GetRandom()
    {
        return normal.Sample();
    }
```

b) Find the sample mean, estimate the standard deviation and produce the 99% confidence interval for this scenario for a given sample size. (5 marks)

The variation produces the following results:



Task 2: The mean length for the scenario needs to be estimated to within [$\hat{\mu}n - \varepsilon$, $\hat{\mu}n + \varepsilon$] error with a significance level α, where $\varepsilon > 0$ and $0 < \alpha \leq 1$ are passed as a parameters.

a) Design and describe an algorithm that is able to find c such that: $\mu \in [\ \hat{\mu}n - c\ ,\ \hat{\mu}n + c\ ]$ with a significance level α Such that $\varepsilon \geq c > 0$ The mean number of executions of the Monte Carlo simulations is O(1).

Provide screenshots showing the different problem sizes analysed and the final confidence interval obtained.

```
public static string FindMaxError(double e, double alpha, int n=100)
{
    Console.WriteLine("Default problem size = " + n);
    double l_three_d = 0; //length of a line
    double[] distancesThreeD = new double[n]; // here the distance is stored in an array for each iteration of n
    double c = 0;
    int k = 0;
    BoxMullerNormal b = new BoxMullerNormal();

    //repeat the experiment n times
    for (int i = 0; i < n; i++)
    {
        //get a point (x1,x2,y1,y2,z1,z2) at random
        double x1 = b.GetRandom();
        double x2 = b.GetRandom();
        double y1 = b.GetRandom();
        double y2 = b.GetRandom();
        double z1 = b.GetRandom();
        double z2 = b.GetRandom();

        //3D
        double result2 = ThreeD(x1, x2, y1, y2, z1, z2);
        l_three_d = result2; //get length of a 3d line
        distancesThreeD[i] = l_three_d;
    }

    double averageOfLineThreeD = Queryable.Average(distancesThreeD.AsQueryable());
    double sum = distancesThreeD.Sum(d => Math.Pow(d - averageOfLineThreeD, 2));
    double sd_3d = Math.Sqrt((sum) / (distancesThreeD.Count() - 1));
    //double sd_3d = Math.Sqrt(distancesThreeD.Average(v => Math.Pow(v - averageOfLineThreeD, 2)));


    c = alpha * (sd_3d/ Math.Sqrt(n));
```

```
if(c > e)
{
    Console.WriteLine("c is less than e, so try again...");
    k = 1;
    n = (int)Math.Ceiling(Math.Pow((alpha * sd_3d / e), 2) + k);
   return FindMaxError(alpha, e, n);
}
else
{
    Console.WriteLine("c is greater than e, Done!");

    #region confidence interval
    //3D
    double marginOfError_3d = 2.58 * (sd_3d / Math.Sqrt(n));
    double lowerEndOfInterval_3d = averageOfLineThreeD - marginOfError_3d;//lower end of the interval is average - margin of error
    double UpperEndOfInterval_3d = averageOfLineThreeD + marginOfError_3d;//and upper end is average + margin of error
    Console.WriteLine("μ: " + averageOfLineThreeD + " ± " + marginOfError_3d);
    Console.WriteLine("lower end of interval = " + lowerEndOfInterval_3d);
    Console.WriteLine("upper end of interval = " + UpperEndOfInterval_3d);
    Console.WriteLine("You can be 99% confident that the population mean (μ) falls between" + lowerEndOfInterval_3d + " and " + UpperEndOfInterval_3d);
    #endregion
    return "Max error: " + c + " , Ideal n = " + n;
}
```

```
Console.WriteLine("Finding max error and ideal n: \n");
double alpha = 0.95;
double e = 0.01;
string maxerror = MonteCarlo.FindMaxError(e, alpha);
Console.WriteLine(maxerror);
```

First, I set alpha to 0.95, and the error to 0.01, and default problem size is 100. These are the results:

```
Finding max error and ideal n:

Default problem size = 100
c is less than e, so try again...
Default problem size = 11370
c is greater than e, Done!
μ: 2.26660819814551 ± 0.0230199495481292
lower end of interval = 2.24358824859738
upper end of interval = 2.28962814769364
You can be 0.95% confident that the population mean (μ) falls between2.24358824859738 and 2.28962814769364
Max error: 0.00847633801190804 , Ideal n = 11370
```

The above results show that it only took 2 rounds for the ideal n to be found. The final confidence interval is show above.

Now, if I set the problem size to be 1500, and change alpha to 0.99, let's see the difference:

```
Finding max error and ideal n:

Default problem size = 1500
c is less than e, so try again...
Default problem size = 8959
c is less than e, so try again...
Default problem size = 8981
c is greater than e, Done!
μ: 2.26905339945317 ± 0.0258076785181586
lower end of interval = 2.24324572093501
upper end of interval = 2.29486107797133
You can be 0.99% confident that the population mean (μ) falls between2.24324572093501 and 2.29486107797133
Max error: 0.00990294640813063 , Ideal n = 8981
```

This actually took longer because it took 3 rounds not 2.

b) Give a valid argument that the mean number of executions of the Monte Carlo simulation is O(1). (5 marks)

The process of finding the ideal n and max error is a fairly easy and light weight task. Basically, we have a 50% chance of finding a low standard deviation on our population, and a 50% chance of finding a high one. If the error is greater than the max error, then we repeat the process again, if not we are done. This repetition only happens up to 3 to 4 times, as we are bound to find an error smaller or equal to the max error in maximum ~3-4 rounds. In my case it takes two rounds most of the time, one time the error is larger than the max error, and the next it is smaller (using alpha of 0.99, starting n = 100 and max error = 0.01) and the ideal n is usually over 1000. Thus, the Monte Carlo simulation should take constant time as it never exceeds more than 3 minutes, no matter the amount of data in the data set. Whether I have a small number of items in my collection, or if I have a big number, the Monte Carlo Simulation will take the same amount of time to find the maximum error and ideal n, making the simulation O(1) because the time taken is constant and not based on the size of the number of items in the collection. Big O(1) means that it doesn't matter how many items you have in a data structure, it should still perform in constant time. This is the case for Monte Carlo Simulation, as no matter how big my problem size is, it always performs in constant time. Also, we need to keep in mind that we are assuming that the true standard deviation is finite, if not the Monte Carlo simulation will not work. I created an integer called repeats and passed it in my find max error method, setting it as a default of 1. Every time the max error is less than e, it is counted as a repeated. Below you may see my implementation and results, in which the results confirm that the Monte Carlo Simulation is in fact O(1) because no matter how large the problem size, the repeats are always between 1-3, therefore E(X) = 1 + 2 = 3 E O(1).

```
public static string FindMaxError(double e, double alpha, int n=100, int repeats=1)
{
    Console.WriteLine("Default problem size = " + n);
```

```
if(c > e)
{
    Console.WriteLine("c is less than e, so try again...");
    k = 1;
    n = (int)Math.Ceiling(Math.Pow((alpha * sd_3d / e), 2) + k);
    repeats++;
    return FindMaxError(e, alpha, n, repeats);
}
else
{
    Console.WriteLine("c is greater than e, Done!");

    confidence interval

                                I call the repeats here when it is done.
    Console.WriteLine("Repeats: " + repeats);
    return "Max error: " + c + " , Ideal n = " + n;
}
//finding a 1 in a random array
```

```
Trial 0: Finding max error and ideal n:
Default problem size = 100
c is less than e, so try again...
Default problem size = 9910
c is greater than e, Done!
μ: 2.24965107822686 ± 0.0244800481067402
lower end of interval = 2.22517103012012
upper end of interval = 2.2741311263336
You can be 0.99% confident that the population mean (μ) falls between2.22517103012012 and 2.2741311263336
Repeats: 2
Max error: 0.0093935068316561 , Ideal n = 9910
'
Trial 1: Finding max error and ideal n:
Default problem size = 100
c is less than e, so try again...
Default problem size = 10628
c is greater than e, Done!
μ: 2.25779765017724 ± 0.0240106147650994
lower end of interval = 2.23378703541214
upper end of interval = 2.28180826494234
You can be 0.99% confident that the population mean (μ) falls between2.23378703541214 and 2.28180826494234
Repeats: 2
Max error: 0.00921337543311955 , Ideal n = 10628
'
```

```
Trial 2: Finding max error and ideal n:
Default problem size = 100
c is less than e, so try again...
Default problem size = 8169
c is less than e, so try again...
Default problem size = 8997
c is greater than e, Done!
μ: 2.26891750124779 ± 0.0260544989019517
lower end of interval = 2.24286300234584
upper end of interval = 2.29497200014974
You can be 0.99% confident that the population mean (μ) falls between2.24286300234584 and 2.29497200014974
Repeats: 3
Max error: 0.00999765655540005 , Ideal n = 8997
'
Trial 3: Finding max error and ideal n:
Default problem size = 100
c is less than e, so try again...
Default problem size = 9217
c is greater than e, Done!
μ: 2.27585118735229 ± 0.0258672261512456
lower end of interval = 2.24998396120105
upper end of interval = 2.30171841350354
You can be 0.99% confident that the population mean (μ) falls between2.24998396120105 and 2.30171841350354
Repeats: 2
Max error: 0.0099257960812919 , Ideal n = 9217
```