Project 3

Matt Briskey

10/7/23

See the file "Project_3.pdf" for the project scenario and description.

Exercises

Exercise 1

When conducting statistical analysis, particularly so for supervised and unsupervised statistical learning, it is prudent to complete an exploratory data analysis (EDA).4 After reading the data frames Project3_prodhier.RDS and Project3_segment.RDS into RStudio (or another R application), make custid the index for each row of Project3_segment.RDS.Unless otherwise stated, all analytics requested below are to be completed using the data of Project3_segment.RDS.

(a) Use the skim function of skimr to create a summary of the data. (6 points) One of the benefits of using this function is the output of a histogram spark graph for each numerical variable. While one may export the output of skim as txt file using the sink function, reading this output into a tool to create the report is not so easy.

This said, for your report provide the output of the function skim without charts, where the output will not contain the spark graphs. Using the quantiles of the results of skim, we will create multiple contingency tables. Specification of these tables follow.

Run the following chunk to load data.

```
# load data frames
# Set the working directory
setwd("C:/Users/16145/Google Drive/MSDA/DATA621 Advanced Analytics/Project 3")
df0 <- readRDS("Project3_segment.RDS")

# Make custid row index
df1 <- data.frame(df0[,-1], row.names = df0[,1])
```

NOTE: use the chunk below to use skim to produce a summary of the data.

```
eda1 <- skim_without_charts(df1)
eda1
```

Table 1: Data summary

Name	df1
Number of rows	2558
Number of columns	20
Column type frequency:	
numeric	20
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
p1	0	1.00	36.03	50.13	0	0	11.02	93.21	163.65
p2	0	1.00	24.80	48.38	0	0	0.00	19.89	197.19
p3	0	1.00	13.75	14.06	0	0	10.30	25.61	57.39
p4	0	1.00	12.00	16.40	0	0	0.00	23.23	72.96
p5	0	1.00	10.40	14.72	0	0	0.00	21.66	56.49
p6	0	1.00	9.11	19.35	0	0	0.00	8.50	112.34
p7	0	1.00	11.72	16.61	0	0	4.53	21.29	74.01
p8	0	1.00	9.31	12.15	0	0	0.00	16.96	60.55
p9	0	1.00	7.36	14.40	0	0	0.00	6.04	60.04
p10	0	1.00	4.17	8.65	0	0	0.00	0.00	49.13
p11	0	1.00	4.96	7.30	0	0	0.00	7.20	41.60
p12	0	1.00	4.80	9.49	0	0	0.00	7.62	49.49
p13	0	1.00	15.10	16.71	0	0	11.74	26.21	77.49
p14	0	1.00	7.02	8.32	0	0	4.02	13.01	43.65
p15	0	1.00	2.22	3.58	0	0	0.00	3.96	25.89
age	182	0.93	42.85	14.76	18	31	42.00	53.00	90.00
hhincome	165	0.94	79.83	37.43	18	49	77.00	106.00	196.00
gender	211	0.92	1.03	0.42	0	1	1.00	1.00	2.00
tenure	0	1.00	37.75	19.35	1	24	37.00	51.00	96.00
loyalty	0	1.00	0.75	0.43	0	0	1.00	1.00	1.00

(b) For the value of age of each custid, map it to one of following five groups where $Q_{age,p}$ is the empirical quantile of age for percentile p; $Q_{age,0}$ is the minimum and $Q_{age,1}$ is the maximum.

- Age Group 1: age $\in [Q_{age,0}, Q_{age,0.25}]$.
- Age Group 2: age $\in (Q_{age,0.25}, Q_{age,0.5}]$.
- Age Group 3: age $\in (Q_{age,0.5}, Q_{age,0.75}]$.
- Age Group 4: age $\in (Q_{age,0.75}, Q_{age,1}]$.
- Age Group 5: age is missing.

Complete the same for hhincome and tenure using their empirical quantiles. Note tenure will only have 4 groups since there are no missing values. Using the groupings for age and hhincome, create a contingency table that displays frequencies for the demographic variables age, hhincome and gender. Ensure totals are provided for each row and column; this request holds for contingency tables created and reported for this project. (6 points) Run the following code chunk.

```
# function to format as dollars
format.money  <- function(x) {
  paste0("$", formatC(as.numeric(x), format="f", digits=0, big.mark=","))
}

# make separate df for variables we want to summarize
df2 <- df1[c("gender","loyalty","age","tenure","hhincome")]

# create age group labels based on percentiles
age_1 <- paste(c(eda1$numeric.p0[16],"to",
                 eda1$numeric.p25[16]),collapse = " ")
age_2 <- paste(c(eda1$numeric.p25[16]+1,"to",
                 eda1$numeric.p50[16]),collapse = " ")
age_3 <- paste(c(eda1$numeric.p50[16]+1,"to",
                 eda1$numeric.p75[16]),collapse = " ")
age_4 <- paste(c(eda1$numeric.p75[16]+1,"to",
                 eda1$numeric.p100[16]),collapse = " ")

# group ages, including a category for missing ages
df2$agecohort <- ifelse(df2$age <= eda1$numeric.p25[16],
                        age_1,
                        ifelse(df2$age <= eda1$numeric.p50[16],
                               age_2,
                               ifelse(df2$age <= eda1$numeric.p75[16],
                                      age_3,
                                      ifelse(df2$age <= eda1$numeric.p100[16],
                                             age_4,"Missing"))))

# apply money format function
hhincomep0 <- format.money(eda1$numeric.p0[17])
hhincomep25 <- format.money(eda1$numeric.p25[17])
hhincomep50 <- format.money(eda1$numeric.p50[17])
hhincomep75 <- format.money(eda1$numeric.p75[17])
hhincomep100 <- format.money(eda1$numeric.p100[17])
hhincomep25p <- format.money(eda1$numeric.p25[17]+1)
hhincomep50p <- format.money(eda1$numeric.p50[17]+1)
hhincomep75p <- format.money(eda1$numeric.p75[17]+1)
```

```
# formatted income group labels
hhincome_1 <- paste(c(hhincomep0,"to",hhincomep25),
                    collapse = " ")
hhincome_2 <- paste(c(hhincomep25p,"to",hhincomep50),
                    collapse = " ")
hhincome_3 <- paste(c(hhincomep50p,"to",hhincomep75),
                    collapse = " ")
hhincome_4 <- paste(c(hhincomep75p,"to",hhincomep100),
                    collapse = " ")

# group the incomes, including a category for missing
df2$hhincomecohort <- ifelse(df2$hhincome <= eda1$numeric.p25[17],
                         hhincome_1,
                         ifelse(df2$hhincome <= eda1$numeric.p50[17],
                                hhincome_2,
                                ifelse(df2$hhincome <= eda1$numeric.p75[17],
                                       hhincome_3,
                                       ifelse(df2$hhincome <= eda1$numeric.p100[17],
                                              hhincome_4,"Missing"))))

# create tenure group labels
tenure_1 <- paste(c(eda1$numeric.p0[19],"to",
                    eda1$numeric.p25[19]),collapse = " ")
tenure_2 <- paste(c(eda1$numeric.p25[19]+1,"to",
                    eda1$numeric.p50[19]),collapse = " ")
tenure_3 <- paste(c(eda1$numeric.p50[19]+1,"to",
                    eda1$numeric.p75[19]),collapse = " ")
tenure_4 <- paste(c(eda1$numeric.p75[19]+1,"to",
                    eda1$numeric.p100[19]),collapse = " ")

# group tenure
df2$tenurecohort <- ifelse(df2$tenure <= eda1$numeric.p25[19],
                         tenure_1,
                         ifelse(df2$tenure <= eda1$numeric.p50[19],
                                tenure_2,
                                ifelse(df2$tenure <= eda1$numeric.p75[19],
                                       tenure_3,
                                       ifelse(df2$tenure <= eda1$numeric.p100[19],
                                              tenure_4,"Missing"))))
```

NOTE: Fill in code to group gender and loyalty cohorts. Use the names that are given in the code chunk below.

```
# create gender groupings here
df2$gendercohort <- ifelse(df2$gender == 0,
                           "male",
                           ifelse(df2$gender == 1,
                                  "female",
                                  ifelse(df2$gender == 2,
                                         "other", "missing")))


df2$loyaltycohort <- ifelse(df2$loyalty == 1,
                           "yes","no")
```

NOTE: Place your code at the bottom of the following code chunk. Create the formatted summary table using xtabs() and ftable(), similar to what was done in Project 2.

```
# create df for grouped variables only
df3 <- subset(df2, select = c(agecohort,hhincomecohort,
                              gendercohort))

# rename NA's as Missing
df3[is.na(df3)] <- "Missing"


# Exposure Tables
ct1 <- xtabs(~agecohort+hhincomecohort+gendercohort,
             addNA = TRUE, data=df3)

ct1_export <- ftable(addmargins(ct1, margin = 1:3,
                                list(Total = sum)))
```

```
## Margins computed over dimensions
## in the following order:
## 1: agecohort
## 2: hhincomecohort
## 3: gendercohort
```

```
print("Age, Income, Gender Frequency Contingency Table")
```

```
## [1] "Age, Income, Gender Frequency Contingency Table"
```

```
ct1_export
```

```
##                          gendercohort female male Missing other Total
## agecohort hhincomecohort
## 18 to 31  $107 to $196                   34    2       0     4    40
##           $18 to $49                    209   31       5    32   277
##           $50 to $77                    128   25       2    21   176
```

```
##           $78 to $106   103    9       1     9   122
##           Missing         1    0       2     0     3
##           Total         475   67      10    66   618
## 32 to 42  $107 to $196   97    9       2     9   117
##           $18 to $49    167   14       2    17   200
##           $50 to $77    125   10       1    16   152
##           $78 to $106   115   13       3    14   145
##           Missing         0    0       5     0     5
##           Total         504   46      13    56   619
## 43 to 53  $107 to $196   138    9       2    10   159
##           $18 to $49     79    4       2    10    95
##           $50 to $77    117    8       3    16   144
##           $78 to $106   124    9       0    22   155
##           Missing         2    0       2     0     4
##           Total         460   30       9    58   557
## 54 to 90  $107 to $196   231    6       3    30   270
##           $18 to $49     31    1       0     3    35
##           $50 to $77     85    5       3    15   108
##           $78 to $106   140    8       3    15   166
##           Missing         1    0       2     0     3
##           Total         488   20      11    63   582
## Missing   $107 to $196     2    0       5     0     7
##           $18 to $49       3    0       5     0     8
##           $50 to $77       1    0       3     1     5
##           $78 to $106      1    0      11     0    12
##           Missing          4    1     144     1   150
##           Total           11    1     168     2   182
## Total     $107 to $196   502   26      12    53   593
##           $18 to $49     489   50      14    62   615
##           $50 to $77     456   48      12    69   585
##           $78 to $106    483   39      18    60   600
##           Missing          8    1     155     1   165
##           Total         1938  164     211   245  2558
```

(c) Using the groupings for age and hhincome, create a contingency table that displays proportions for age, hhincome and gender. Four decimal points will be sufficient for all proportions in all tables and illustrations. (6 points) NOTE: You can produce a formatted table of proportions simply by taking the object that you stored the formatted table from part 1b in and dividing it by the overall sample size. For example, if you named that table ct1_format, then you can convert all the numbers in it to proportions with ct1_format/n, where n is holding the sample size.

```
prop_table_ct1 <-  round(prop.table(ct1),4)

prop_table_ct1_export <- ftable(addmargins(prop_table_ct1,
```

```
                                                margin = 1:3,
                                                list(Total = sum)))
```

```
## Margins computed over dimensions
## in the following order:
## 1: agecohort
## 2: hhincomecohort
## 3: gendercohort
```

```
print("Age, Income, Gender Proportion Contingency Table")
```

```
## [1] "Age, Income, Gender Proportion Contingency Table"
```

```
prop_table_ct1_export
```

```
##                          gendercohort female   male Missing  other  Total
## agecohort hhincomecohort
## 18 to 31  $107 to $196               0.0133 0.0008  0.0000 0.0016 0.0157
##           $18 to $49                 0.0817 0.0121  0.0020 0.0125 0.1083
##           $50 to $77                 0.0500 0.0098  0.0008 0.0082 0.0688
##           $78 to $106                0.0403 0.0035  0.0004 0.0035 0.0477
##           Missing                    0.0004 0.0000  0.0008 0.0000 0.0012
##           Total                      0.1857 0.0262  0.0040 0.0258 0.2417
## 32 to 42  $107 to $196               0.0379 0.0035  0.0008 0.0035 0.0457
##           $18 to $49                 0.0653 0.0055  0.0008 0.0066 0.0782
##           $50 to $77                 0.0489 0.0039  0.0004 0.0063 0.0595
##           $78 to $106                0.0450 0.0051  0.0012 0.0055 0.0568
##           Missing                    0.0000 0.0000  0.0020 0.0000 0.0020
##           Total                      0.1971 0.0180  0.0052 0.0219 0.2422
## 43 to 53  $107 to $196               0.0539 0.0035  0.0008 0.0039 0.0621
##           $18 to $49                 0.0309 0.0016  0.0008 0.0039 0.0372
##           $50 to $77                 0.0457 0.0031  0.0012 0.0063 0.0563
##           $78 to $106                0.0485 0.0035  0.0000 0.0086 0.0606
##           Missing                    0.0008 0.0000  0.0008 0.0000 0.0016
##           Total                      0.1798 0.0117  0.0036 0.0227 0.2178
## 54 to 90  $107 to $196               0.0903 0.0023  0.0012 0.0117 0.1055
##           $18 to $49                 0.0121 0.0004  0.0000 0.0012 0.0137
##           $50 to $77                 0.0332 0.0020  0.0012 0.0059 0.0423
##           $78 to $106                0.0547 0.0031  0.0012 0.0059 0.0649
##           Missing                    0.0004 0.0000  0.0008 0.0000 0.0012
##           Total                      0.1907 0.0078  0.0044 0.0247 0.2276
## Missing   $107 to $196               0.0008 0.0000  0.0020 0.0000 0.0028
##           $18 to $49                 0.0012 0.0000  0.0020 0.0000 0.0032
##           $50 to $77                 0.0004 0.0000  0.0012 0.0004 0.0020
##           $78 to $106                0.0004 0.0000  0.0043 0.0000 0.0047
##           Missing                    0.0016 0.0004  0.0563 0.0004 0.0587
```

```
##           Total                      0.0044 0.0004  0.0658 0.0008 0.0714
## Total     $107 to $196               0.1962 0.0101  0.0048 0.0207 0.2318
##           $18 to $49                 0.1912 0.0196  0.0056 0.0242 0.2406
##           $50 to $77                 0.1782 0.0188  0.0048 0.0271 0.2289
##           $78 to $106                0.1889 0.0152  0.0071 0.0235 0.2347
##           Missing                    0.0032 0.0004  0.0607 0.0004 0.0647
##           Total                      0.7577 0.0641  0.0830 0.0959 1.0007
```

(d) Using the groupings for tenure, create a contingency table that displays frequencies for the retailer supplied variables tenure and loyalty. (6 points) NOTE: Use xtabs() and ftable().

```
df4 <- subset(df2, select = c(tenurecohort, loyaltycohort))

# Exposure Tables
ct1 <- xtabs(~tenurecohort+loyaltycohort,
             addNA = TRUE, data=df4)

ct1_export <- ftable(addmargins(ct1, margin = 1:2,
                                list(Total = sum)))
```

```
## Margins computed over dimensions
## in the following order:
## 1: tenurecohort
## 2: loyaltycohort
```

```
print("Tenure and Loyalty Frequency Contingency Table")
```

```
## [1] "Tenure and Loyalty Frequency Contingency Table"
```

```
ct1_export
```

```
##              loyaltycohort   no  yes Total
## tenurecohort
## 1 to 24                     371  301   672
## 25 to 37                    172  487   659
## 38 to 51                     81  516   597
## 52 to 96                     23  607   630
## Total                       647 1911  2558
```

(e) Using the groupings for tenure, create a contingency table that displays proportions for the retailer supplied variables tenure and loyalty. (6 points) NOTE: You can produce a formatted table of proportions simply by taking the object that you stored the formatted table from part 1b in and dividing it by the overall sample size.

```
prop_table_ct1 <-  round(prop.table(ct1),4)
```

```
prop_table_ct1_export <- ftable(addmargins(prop_table_ct1,
                                           margin = 1:2,
                                           list(Total = sum)))
```

```
## Margins computed over dimensions
## in the following order:
## 1: tenurecohort
## 2: loyaltycohort
```

```
print("Tenure and Loyalty Proportion Contingency Table")
```

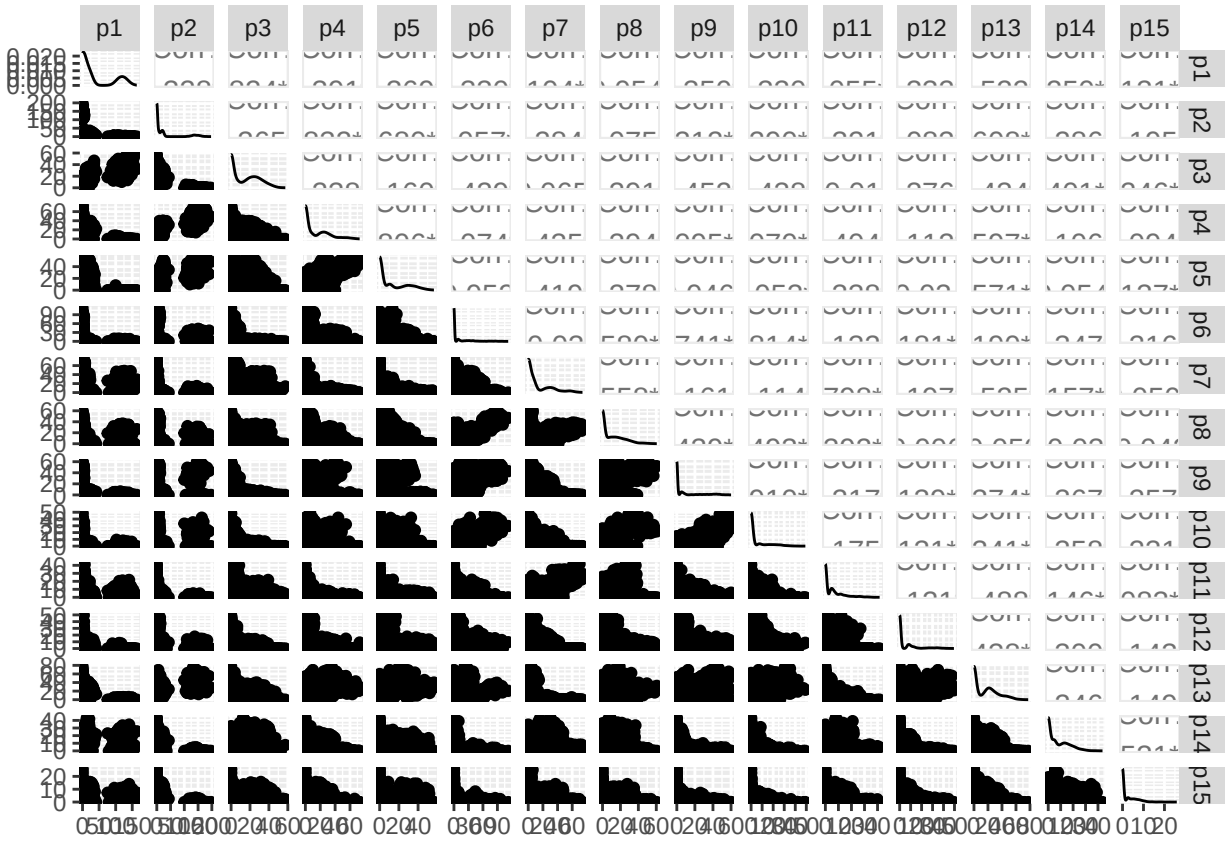
```
## [1] "Tenure and Loyalty Proportion Contingency Table"
```

```
prop_table_ct1_export
```

```
##              loyaltycohort     no    yes  Total
## tenurecohort
## 1 to 24                    0.1450 0.1177 0.2627
## 25 to 37                   0.0672 0.1904 0.2576
## 38 to 51                   0.0317 0.2017 0.2334
## 52 to 96                   0.0090 0.2373 0.2463
## Total                      0.2529 0.7471 1.0000
```

(f) The ggpairs() function from the GGally package allows one to build a scatter plot matrix. With defaults settings, the function provides a matrix with Pearson correlations, bivariate scatter plots, and a smoothed density univariate density estimate on the diagonal. Create a ggpairs scatter plot matrix using p1, p2, . . . , p15. (6 points) NOTE: Place your code at the bottom of the following code chunk. ggpairs can take a few minutes to run for a larger number of variables. Be patient here.

```
# create df with only p1 through p15
df4 <- subset(df1, select = c(p1,p2,p3,p4,p5,
                              p6,p7,p8,p9,p10,p11,
                              p12,p13,p14,p15))

# make scatterplot matrix
ggpairs(df4)
```



(g) Provide a paragraph on how the provided EDA results may be used to determine the number of segments and segment specification using clustering algorithms. (6 points) There are multiple ways to determine the optimal number of segments to use in a clustering algorithm. Gap statistics, elbow method, silhouette coefficient, dendrogram, and Bayesian information criterion can all be used to determine the optimal number of segments for a clustering algorithm. None of these methods were specifically used and we instead relied on exploratory data analysis.

Exercise 2

k-means is in the set of clustering techniques which produce a partition of the observations into a specified number of groups, by either minimizing or maximizing some numerical criterion. Such optimization methods differ from other frequently used methods, such as agglomerative and divisive hierarchical clustering, in not necessarily forming hierarchical classifications of the data. Differences between the methods in this class arise both because of the variety of clustering criteria that might be optimized and the various optimization algorithms that might be used.

As described in the textbook, the basic idea behind the numerical criterion optimization methods is that associated with each partition of the n subjects into the required number of groups, g, is an index c(n, g), the value of which measures some aspect of the "quality" of

this particular partition. For some indices high values are associated with a desirable cluster solution, whereas for others a low value is sought. Associating an index with each partition allows them to be compared. A variety of such clustering criteria have been suggested. Some operate on the basis of the inter-subject dissimilarities; others use the original data matrix.

One of the earliest hill-climbing algorithms, Ball and Hall (1967), consisted of iteratively updating a partition by simultaneously relocating each subject to the group to whose mean it was closest and then recalculating the group means. Although not explicitly stated, it can be shown that, under some regularity conditions, this is equivalent to minimizing the trace of the within-group dispersion matrix when Euclidean distances are used to define "closeness". Such algorithms, involving the calculation of the mean (centroid) of each cluster, are often referred to as k-means algorithms.

While k-means algorithms are most suitable for data where all variables are measured on a continuous scale sans point-specific inflated data (e.g., zero-inflated data), k-means algorithms are still used to identify business segments even though many, if not all, continuous variables are point-specific inflated. Though the variables p1, p2, . . . , p15 are zero-inflated, we will use them, and them only, to conduct a behavioral segmentation using k-means.

An EDA that is typically completed prior to commencing a k-means clustering exercise is to identify outliers in the variables that are to be used. The method of Filzmoser et al. (2005) is frequently used for multivariate outlier detection. It is able to distinguish between extreme values of a normal distribution and values originating from a different distribution (outliers). The function pcout() of the R package "mvoutlier" implements their method. While we will not be completing an outlier detection exercise (since we have zero-inflated data which is surely not normally distributed), you are encouraged to complete such exercises when conducting k-means clustering using data that is (approximately) normally distributed.

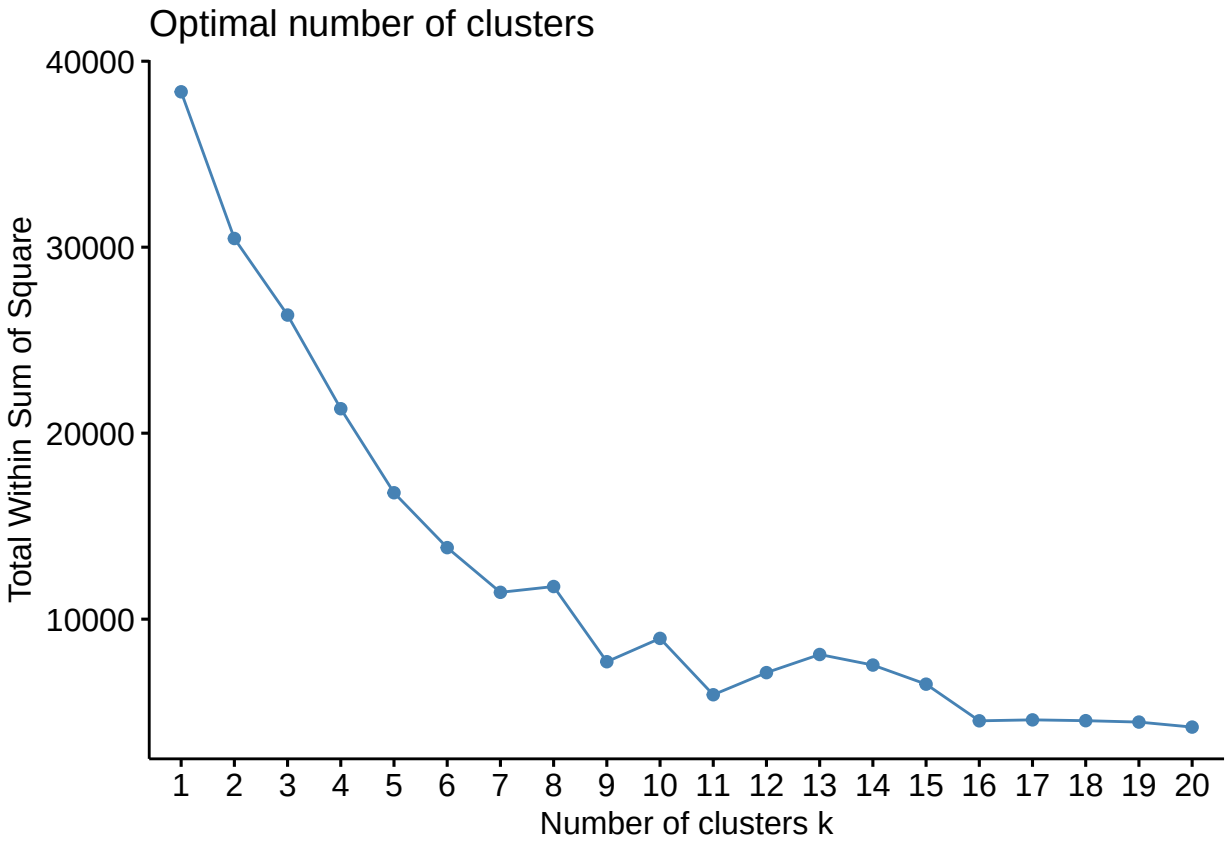
Use the function mutate of the dplyr package to normalize p1, p2, . . . , p15. Once this has been done, we will use these 15 normalized variables to determine the number of k-means clusters.

(a) Originating from Thorndike (1953), the elbow method is a heuristic that is frequently used to determine the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters and choosing the elbow of the curve as the number of clusters to use.

Using the function fviz_nbclust with FUNcluster=kmeans, method = "wss" and k.max = 20, and set.seed(621), create a plot of the best number of clusters using the elbow method. (6 points) NOTE: Place your code for the plot using the elbow method at the bottom of the following code chunk.

```
# first scale the data
df5 <- subset(df1, select=c(p1,p2,p3,p4,p5,p6,
                            p7,p8,p9,p10,p11,p12,
                            p13,p14,p15))
```

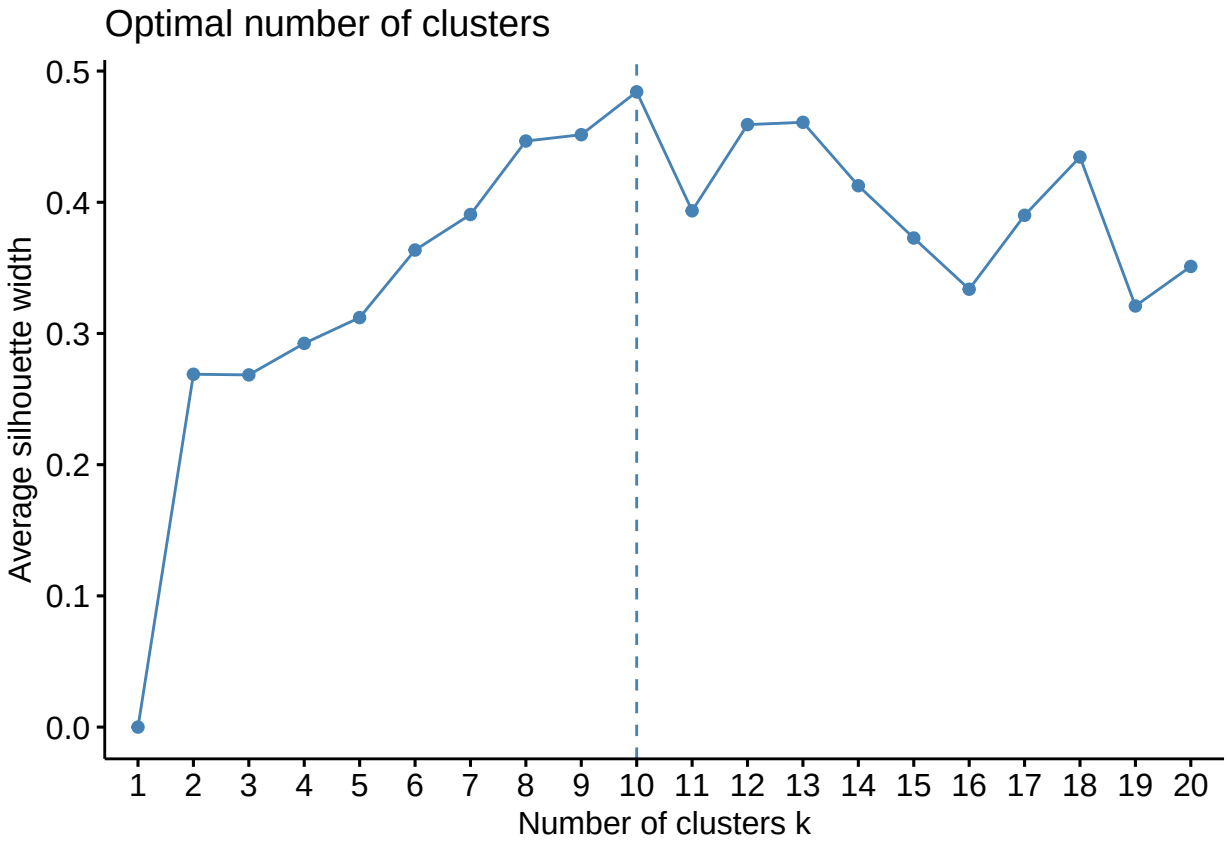
```r
# use dplyr to scale numeric vars
df6 <- df5 %>% mutate(across(where(is.numeric), scale))

# for repeatability
set.seed(621)


#  elbow plot
fviz_nbclust(df6, FUNcluster=kmeans, method = "wss", k.max=20)
```



(b) The average silhouette approach of Rousseeuw (1987) is also frequently used to determine the number of clusters in a data set. Upon completing a k-means clustering exercise, each cluster is represented by a silhouette, which is based on the comparison of its tightness and separation. The silhouette shows which objects are well within their cluster, and which ones are merely somewhere in between clusters. The entire clustering is displayed by combining the silhouettes into a single plot, which allows an appreciation of the relative quality of the clusters and an overview of the data configuration. The average silhouette width, or simply just average silhouette, provides an evaluation of clustering validity, and may be used to select an "appropriate" number of clusters.

Using the function fviz_nbclust with FUNcluster=kmeans, method = "silhouette" and k.max = 20, and set.seed(622), create a plot of the best number of clusters using average silhouette. (6 points) NOTE: Place your code for the plot using the silhouette method at the bottom of the following code chunk.

```
# for reproducibility
set.seed(622)

# Avg Silhouette plot
fviz_nbclust(df6, FUNcluster=kmeans, method = "silhouette", k.max=20)
```



(c) The gap statistic of Tibshirani et al. (2001) compares the total intracluster variation for different numbers of clusters k with their expected values under null reference distribution of the data. For each number of clusters and using squared Euclidean distance as the distance between observations, it compares the natural logarithm of the pooled within-cluster sum of squares around the cluster means to the expected value of the natural logarithm of the pooled within-cluster sum of squares around the cluster means where the reference distribution for calculating the expectation is that there is no obvious clustering. Calculation of the expectation is done by bootstrapping a uniform distribution on the hypercube determined by the ranges of the variables used for clustering.

Given large data and a large number k, calculating the gap statistics and their standard

errors can be computationally expensive. In addition, convergence issues typically arise. Despite these potential computational challenges, we will calculate gap statistics. Use the clusGap function with FUN = kmeans, nstart = 25, K.max = 20, and B = 50, along with set.seed(623), to estimate the gap statistics. Then use the fviz_gap_stat function to plot the gap statistics, where the optimal number of clusters is identified by the dotted line.

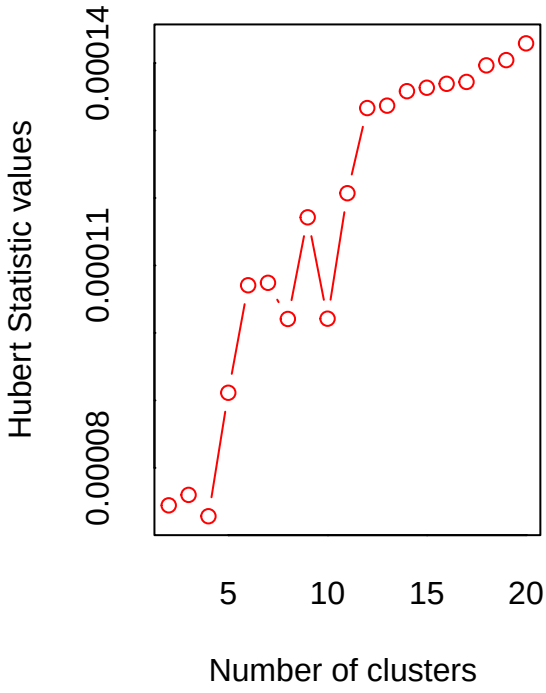
Provide the plot in your report and address the following question. Should the gap statistic be used for this project to identify the best number of clusters? Provide a reason for you answer. (6 points) NOTE: Run the code chunk below. Be patient with this one as well. You will see a lot of warnings, so "warning=FALSE" was added to the chunk options so they won't take up space in your output.

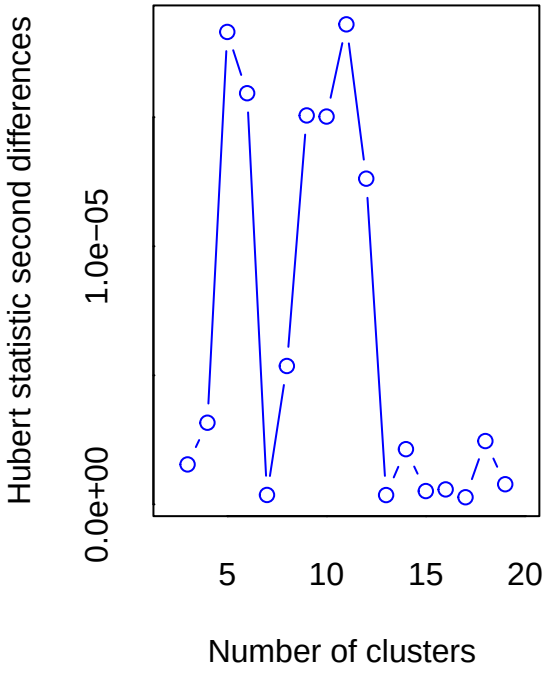
```
# for reproducibility
set.seed(623)

# use gap statistic for finding optimal k
gap_stat <- clusGap(df6, FUN = kmeans, nstart = 25,
                    K.max = 20, B = 50)

gap2 <- fviz_gap_stat(gap_stat)
```

(d) The above methods and their resulting indices for enacting the method for a given data set are just a few of the many techniques used to determine the optimal number of clusters. Chiang and Mirkin (2010) provides a review of these methods, and many other methods, used to determine the optimal number of clusters. The function NBClust of the package of the same name provides up 30 indices for determining the optimal number of clusters. The majority rule, an automatic output of the function upon its successful completion, is at times used to determine the best number of clusters.

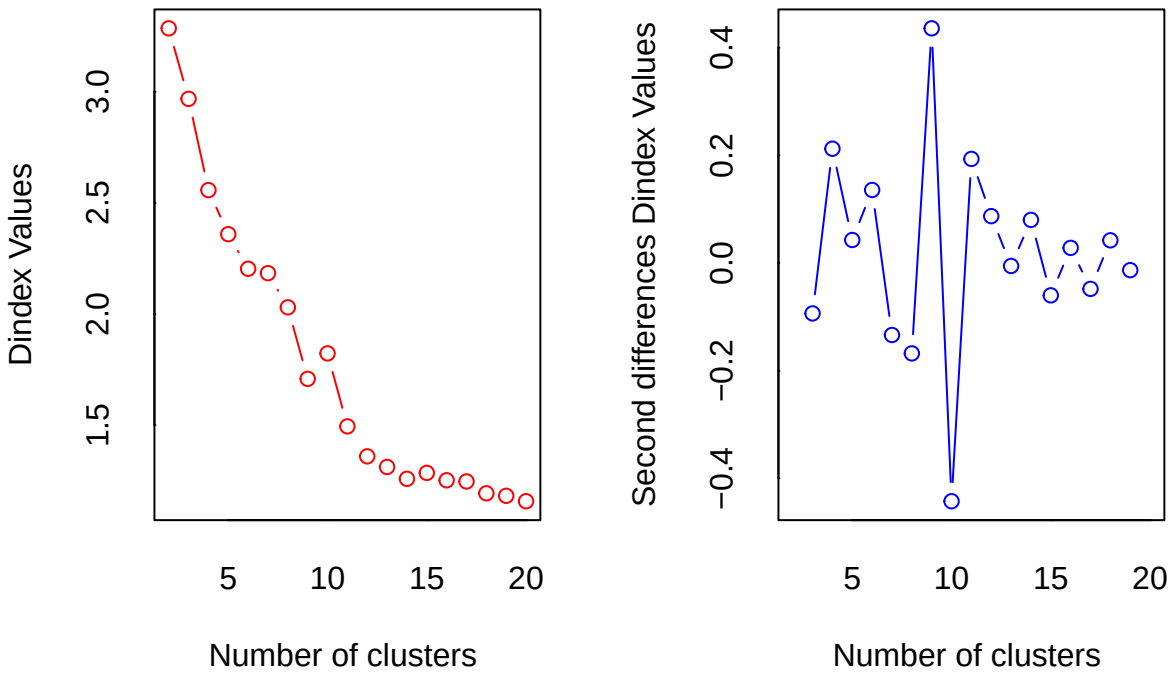
Using the function NbClust with diss = NULL, distance = "euclidean", min.nc = 2, max.nc = 20, and method = "kmeans", and set.seed(624), provide the methods and indices of Best.nc. (6 points) NOTE: Run the code chunk below.

```
# for reproducibility
set.seed(624)

# optimal cluster numbers k-means clustering
nb <- NbClust(data = df6, diss = NULL,
              distance = "euclidean",
              min.nc = 2, max.nc = 20,
              method = "kmeans")
```






```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that correspo
##                 significant increase of the value of the measure i.e the significant
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant p
##                 second differences plot) that corresponds to a significant increase o
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 4 proposed 2 as the best number of clusters
## * 3 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
## * 2 proposed 11 as the best number of clusters
## * 1 proposed 13 as the best number of clusters
## * 7 proposed 14 as the best number of clusters
## * 1 proposed 16 as the best number of clusters
## * 2 proposed 20 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  14
```

```
##
##
## *****************************************************************
```

```
print("NbClust Best.nc output for k-means")
```

```
## [1] "NbClust Best.nc output for k-means"
```

```
nb$Best.nc
```

```
##                      KL      CH Hartigan      CCC    Scott      Marriot  TrCovW
## Number_clusters 16.0000   14.00   10.000  20.0000    11.00 3.000000e+00       3
## Value_Index     44.1338 1385.54 1394.065 227.5002 11369.49 5.645493e+44 3323162
##                    TraceW Friedman   Rubin  Cindex      DB Silhouette   Duda
## Number_clusters     9.000  11.0000 14.0000 13.0000 14.0000    14.0000 2.0000
## Value_Index      4786.454  82.0812 -1.7738  0.2145  0.9546     0.4872 1.5183
##                   PseudoT2   Beale Ratkowsky     Ball PtBiserial Frey McClain
## Number_clusters     2.0000  2.0000    4.0000    3.000    14.0000    1  2.0000
## Value_Index      -661.9031 -3.5215    0.3419 6525.516     0.6104   NA  0.5677
##                     Dunn Hubert SDindex Dindex    SDbw
## Number_clusters 14.0000      0 14.0000      0 20.0000
## Value_Index      0.0797      0  1.5663      0  0.1776
```

(e) Setting aside the gap statistic results, since convergence and computationally complexity are challenges given the size of the data, choose the best number of clusters using the other calculated indices or some other reason. Explain why you chose this number of clusters. Are there any concerns with choosing this number of clusters? (6 points) From the Elbow chart, 11 appears to be the optimal number of clusters. The silhouette chart shows 10 to be the optimal number of clusters. Using the NbClust function, 14 appears to be the optimal number of clusters. For the remainder of the project, I will use 14 as the number of clusters since that is the largest of the three and I want to make sure I get the best statistics I can.

Exercise 3

For the number of clusters you chose and set.seed(625), use the kmeans function with nstart = 80 to conduct clustering. Identify the output of the call to kmeans as kmeans80.

NOTE: Use the name "km_true80" to store the k-means output, as indicated in the following chunk.

```
# for reproducibility
set.seed(625)

# k-means clustering with k=14
km_true80 <-  kmeans(df6, centers = 14, nstart = 80)
```

```
km_true80$size
```

(a) Report the size of each cluster for the sample. (4 points)

```
##  [1] 263 365 176 110 396 171  55  59 182 173 255 126  42 185
```

(b) Report the expected size of each cluster for the population of custid. (4 points) Since the sample is 0.01% of the population, we can multiply each cluster size by 100 and we get the below:

Cluster 1: 26,300 Cluster 2: 36,500 Cluster 3: 17,600 Cluster 4: 11,000 Cluster 5: 39,600 Cluster 6: 17,100 Cluster 7: 5,500 Cluster 8: 5,900 Cluster 9: 18,200 Cluster 10: 17,300 Cluster 11: 25,500 Cluster 12: 12,600 Cluster 13: 4,200 Cluster 14: 18,500 Total Population: 255,800

(c) Report for each cluster the means for p1, p2, . . . , p15. These values are to

Run the code chunk below.

```r
# get centroids
centroids <- data.frame(aggregate(df5,
                                  by = list(cluster =
                                  km_true80$cluster),
                                  mean))

# store indices
i <- c(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)

centroids[,i] <- apply(centroids[ , i],
                       2, # Specify own function within apply
                     function(x) as.numeric(as.character(x)))
centroids$p1 <- currency(centroids$p1)
centroids$p2 <- currency(centroids$p2)
centroids$p3 <- currency(centroids$p3)
centroids$p4 <- currency(centroids$p4)
centroids$p5 <- currency(centroids$p5)
centroids$p6 <- currency(centroids$p6)
centroids$p7 <- currency(centroids$p7)
centroids$p8 <- currency(centroids$p8)
centroids$p9 <- currency(centroids$p9)
centroids$p10 <- currency(centroids$p10)
```

```
centroids$p11 <- currency(centroids$p11)
centroids$p12 <- currency(centroids$p12)
centroids$p13 <- currency(centroids$p13)
centroids$p14 <- currency(centroids$p14)
centroids$p15 <- currency(centroids$p15)

print("Clusters Centroids")
```

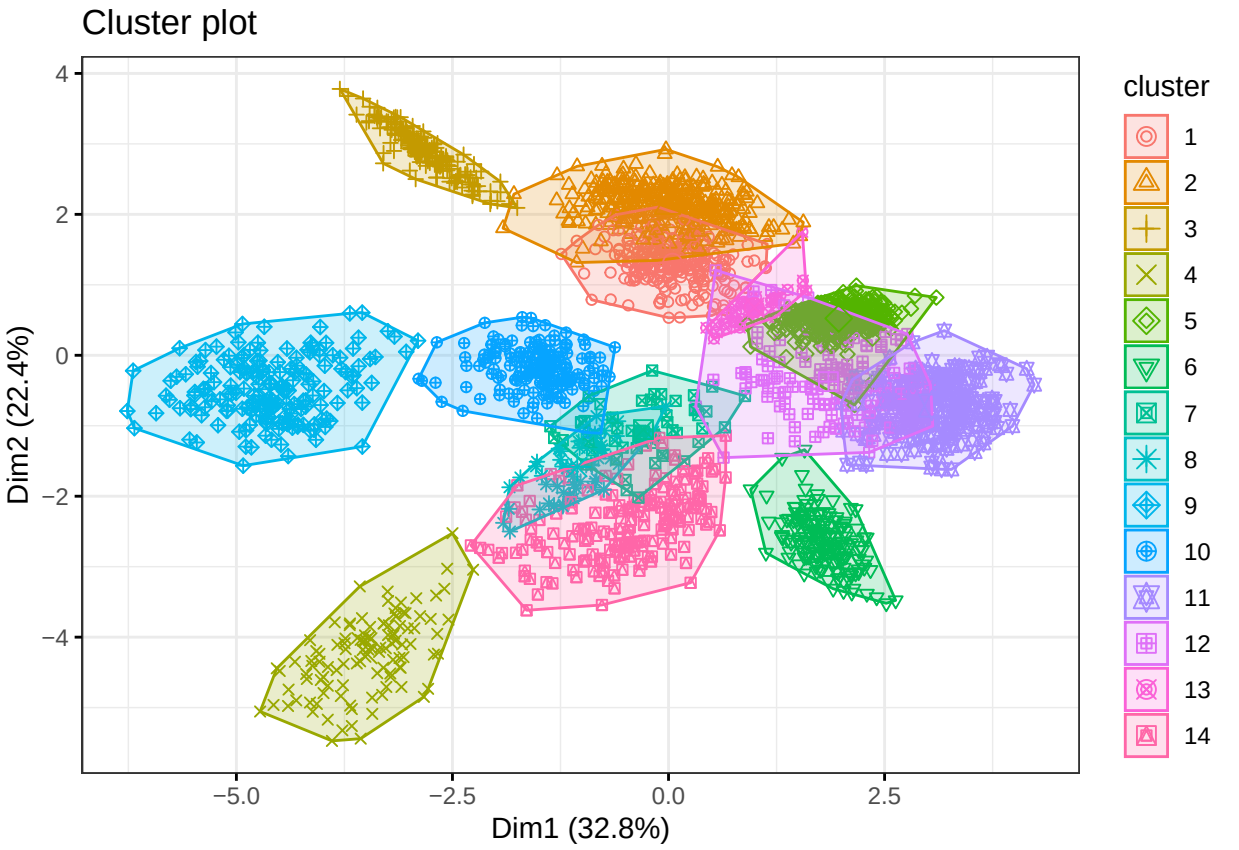
```
## [1] "Clusters Centroids"
```

```
centroids
```

```
##    cluster      p1      p2     p3     p4     p5     p6     p7     p8     p9
## 1        1  $17.33  $16.20 $15.79 $23.13  $5.43  $0.11  $0.58  $0.17  $2.86
## 2        2  $20.31  $19.57 $22.36 $21.50 $32.09  $0.25  $0.53  $0.13  $0.14
## 3        3   $0.16 $145.31  $0.15 $52.51 $39.87  $0.14  $0.68  $0.11  $0.08
## 4        4   $0.27   $3.70  $0.24  $3.73  $5.91 $78.54  $0.74 $41.07 $43.45
## 5        5 $118.88   $0.13 $34.24  $0.07  $0.00  $0.17  $5.36  $0.24  $0.18
## 6        6   $2.66   $0.03  $3.01  $0.07  $0.00  $0.18 $54.40 $26.09  $0.14
## 7        7   $0.79   $0.20  $0.43  $0.56  $0.15  $7.28  $5.17  $3.52  $3.80
## 8        8   $0.23   $0.31  $0.07  $0.09  $0.00  $6.28  $5.95  $3.11 $41.02
## 9        9   $0.58 $138.98  $0.10 $32.09 $23.67 $26.33  $5.40 $17.88 $38.57
## 10      10   $0.20   $3.31  $0.14  $3.75  $4.27  $7.21  $0.74  $8.61  $4.86
## 11      11 $121.22   $0.10 $27.16  $0.10  $0.03  $0.14 $31.41 $23.77  $0.03
## 12      12  $11.35   $0.21 $10.98  $0.31  $1.00  $0.61 $23.18  $8.43  $0.46
## 13      13   $0.67   $0.32  $0.84  $0.46  $0.21  $0.17  $0.90  $0.00  $0.00
## 14      14   $0.25   $0.12  $1.76  $2.51  $3.19 $40.56 $28.61 $12.95 $13.86
##       p10    p11    p12    p13    p14    p15
## 1   $0.13  $0.21  $0.12 $20.34 $13.59  $0.44
## 2   $0.21  $2.68  $4.29 $23.26 $12.56  $6.37
## 3   $0.12  $0.16  $0.09 $27.66  $0.19  $0.18
## 4  $28.50  $0.07  $4.30 $22.60  $0.21  $0.17
## 5   $0.13  $2.88  $0.06  $0.15  $2.57  $0.26
## 6   $0.04 $24.72  $0.00  $0.06  $2.51  $0.29
## 7   $5.65 $13.71 $26.95 $13.28  $0.86  $0.61
## 8  $13.49  $2.61  $3.60  $0.18  $0.08  $0.24
## 9  $21.89  $0.07  $5.87 $47.97  $0.14  $0.17
## 10  $1.00  $0.34 $32.44 $41.78  $0.20  $0.26
## 11  $0.19 $10.28  $0.06  $0.12 $19.34  $6.29
## 12  $0.64  $9.91  $0.18  $1.02 $21.97  $3.71
## 13  $0.00  $0.29  $0.00  $2.42  $3.58 $14.78
## 14 $10.51  $7.53  $9.49  $2.17  $1.78  $1.22
```

```
fviz_cluster(object=km_true80,data=df6,geom = "point", ellipse.type = "convex", ggtheme
```

(d) To visualize the clusters, we will use principal component analysis. Specifically, the clusters will be plotted using the first two principal components. Use the fviz cluster function with object=kmeans80, the data frame used to produce kmeans80, geom = "point", ellipse.type = "convex", and ggtheme = theme bw(). Save the resulting graph, and provide it in your report. (4 points)



(e) For the 4 largest clusters in terms of custid count, provide a "business description" of the cluster. These descriptions could use the attributes of Project3_prodhier.RDS, the gross sales centroid of each cluster, and demographic, loyalty program and tenure summary statistics of each cluster. If you chose fewer than 4 clusters, then complete this exercise for each of your clusters. (4 points) The largest 4 clusters in terms of custid count are cluster #5 (396), cluster #2 (365), cluster #1 (263), and cluster #11 (255)

Cluster #5 - Body Care-Bath and Shower-Vie Quotidienne gross sales of centroid is $118.88
Cluster #2 - Body Care-Bath and Shower-Vie Quotidienne gross sales of centroid is $20.31
Cluster #1 - Body Care-Bath and Shower-Vie Quotidienne gross sales of centroid is $17.33
Cluster #11 - Body Care-Bath and Shower-Vie Quotidienne gross sales of centroid is $121.22

Exercise 4

In a hierarchical classification the data are not partitioned into a particular number of classes or clusters at a single step. Instead the classification consists of a series of partitions, which may run from a single cluster containing all subjects, to n clusters each containing a single subject. Hierarchical clustering techniques are subdivided into agglomerative methods, which proceed by a series of successive fusions of the n subjects into groups, and divisive methods, which separate the n subjects successively into finer groupings. Agglomerative procedures are probably the most widely used of the hierarchical methods. They produce a series of partitions of the data: the first consists of n single subject "clusters"; the last consists of a single group containing all n subjects. Hierarchical classifications produced by either the agglomerative or divisive route may be represented by a two-dimensional diagram known as a dendrogram, which illustrates the fusions or divisions made at each stage of the analysis.

As an illustrative example of hierarchical clustering, where visualizing a dendrogram standard empirical practice, take a 2% random sample of the custid using set.seed(626). The data frame after sampling will have 51 custids and the normalized values of p1, p2, . . . , p15. Then use this data frame in the dist function with method="euclidean" to return the distance matrix computed by using Euclidean distance to compute the distances between the rows of the input matrix. Use the function hclust with the distance matrix from the previous step and method = "average".

Run the following code chunk.

```
# for reproducibility
set.seed(626)

# randomly sample 2% of data frame
df7 <- sample_n(df6,round(dim(df6)[1]/50))
```

NOTE: Use this data frame df7 in the dist function with method="euclidean" to return the distance matrix computed by using Euclidean distance to compute the distances between the rows of the input matrix.

```
mat<-dist(df7, method="euclidean")
mat
```

```
##              VxLWZNg39jd2 p9R0s4hifwCL THscVqv9A2dL 7mskDeCfUXUu 6JGBfc2igrey
## p9R0s4hifwCL    5.0340944
## THscVqv9A2dL    4.5670707    5.1693555
## 7mskDeCfUXUu    3.0360178    4.1465586    3.4451556
## 6JGBfc2igrey    5.6532466    7.3026958    7.2954272    5.6290172
## AxE8m64pR3Cr    4.6941328    3.8447167    2.5571096    3.7937618    6.9700043
## ldSNHx3DBSrb    4.5213268    7.6137580    7.4187541    5.5706393    5.5118848
## TINTcMJTAyDw    3.2483970    5.8742658    5.6679822    4.1114785    5.0203194
## DrPxteZyuZfS    4.1453800    4.3186209    1.2177804    2.8899196    6.8304040
```

```
## i013JbphkzdZ    3.0535386    3.8193979    3.1871147    0.6259050    5.8525714
## zGadHigIDv6z    4.4343129    5.2043934    1.2381926    3.2992076    6.6543444
## kpej7M5rrkJt    4.4366294    4.8216574    1.4257807    3.4597740    6.9174763
## sij8JzPPvqW1    2.5225228    4.3856445    4.1653004    2.7055386    5.7575348
## Kzcj7AjWcK4i    3.5952184    6.1344736    6.0706690    4.2992077    4.8195919
## IRQYtmVYmDz0    5.9591928    7.5822247    7.6781566    6.2743041    1.7560554
## x1rCQxul53sT    6.4714606    5.1242358    6.3306757    6.2121359    7.7874328
## N8NdoNXifb0E    3.9137360    2.5773435    3.8941875    2.9663732    6.2681393
## mcF3ZYA08Otk    3.2950010    4.0701259    4.9107892    3.8295425    6.4528190
## 0I15liRzEWsY    2.5656112    3.7201289    3.9593498    1.3207907    5.6228062
## JLrRAGZNDhsa    2.4482847    3.7899170    4.1699318    3.1087166    6.6051739
## 7hz2WvdmnFyz    4.6087444    2.7586733    4.1664458    3.7284687    6.5233090
## vRb4qBGZYWJ5    4.6657954    5.3960392    0.8157626    3.6291599    7.1441657
## NIsa6HfyyDJG    3.7179823    6.6105571    6.4751793    4.8122097    4.8182819
## nPfUP9LQGjN5    2.0695342    4.1647413    4.4044487    2.8612227    5.9824883
## I8UeXsVicrNl    4.5606230    2.0396078    4.0853170    3.6774939    6.9218483
## AAoVQNFVGEYX    5.1008358    2.5256917    4.8842303    4.4679576    7.4088993
## 1DgI8w7LHJKO    4.5986544    5.4196618    0.9334724    3.5434508    7.0864184
## Fl4D8tyQt6JD    5.3170763    3.6776926    3.6945037    4.6350498    7.3685674
## cMxBm0i4DQ5e    2.5388090    3.9855344    3.5610858    2.6181499    6.0606900
## RriYEaRbFplB    4.1733460    2.3189697    3.7245174    3.1062130    7.2178422
## iUOWKy71j72s    4.8342687    5.4833326    0.7876924    3.8882574    7.5770804
## lSwlRJFMkWaL    2.7630395    4.4129306    4.7380236    3.5690924    6.1085312
## S7pIj7jJ5sYd    4.4602909    5.1475596    1.2720149    3.3779894    6.8805495
## j3ao6E3OOfh6    4.1904076    2.1581429    3.6942429    3.1298628    6.5861584
## 8099hdnTqjgV    2.4832365    4.6864671    3.7438848    1.9402825    5.2633419
## iIE5hTCkcKXT    5.4321411    3.8033894    3.6919945    4.6886481    7.5974870
## cMZJBv6hI69I    3.1558464    3.4965589    3.5728126    1.1893378    5.9001735
## fWBii4xp1GzE    4.4909656    4.9218420    0.5934783    3.4451029    7.2643791
## c8iiBzBoWUTo    3.7119149    3.7558161    4.1069239    2.0452367    5.6585023
## ONHXGmNh8dIs    4.9087421    2.4561351    4.5016844    4.0087552    7.6648596
## 139Jnx308F75    5.4047013    4.0661954    4.9864736    4.6421879    6.1341020
## Jf6emLIOfmdj    6.6788885    4.5071795    5.2308332    6.0075914    8.8813086
## OrPHld09mheD    6.9289837    6.9696739    7.1337306    6.1538182    4.2057794
## 1mca8JBV02cD    6.2087483    7.9004649    7.8138783    6.2467645    2.1760187
## ItjbeSEfdbcL    5.5072134    7.6723736    7.6088553    5.7043809    2.2940490
## Ipryv7a7GXCB    2.0735096    4.3792792    4.5937949    2.8918493    5.3619955
## f0feu6GNkLXl    4.8094168    5.8735901    5.7642609    4.6056540    5.7650425
## 1Koe5mOVjSO1    6.4504602    3.9156743    5.3225613    5.8649172    8.5280637
## pdFmvb7S0FVG    4.4406047    5.8574008    5.6726057    4.0776756    5.2236323
## ITaszdywMT4z    4.4766165    6.1134827    5.9366642    4.6640511    5.4890537
## XxbmRluFogMr    6.0571593    3.7965209    4.6022191    5.4233322    8.3308608
##              AxE8m64pR3Cr ldSNHx3DBSrb TINTcMJTAyDw DrPxteZyuZfS i013JbphkzdZ
## p9R0s4hifwCL
## THscVqv9A2dL
```

```
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb    7.4651802
## TINTcMJTAyDw    5.6737428    2.3093890
## DrPxteZyuZfS    2.4704965    6.9619734    5.0585360
## i013JbphkzdZ    3.5065362    5.6724095    4.0757277    2.5737309
## zGadHigIDv6z    2.8592651    7.0530105    5.2053143    1.1763308    3.0842387
## kpej7M5rrkJt    2.3510768    7.1662587    5.3337972    1.1548818    3.2307517
## sij8JzPPvqW1    4.5006963    5.2292027    3.8348660    3.3694817    2.7306458
## Kzcj7AjWcK4i    6.1273178    1.9646890    1.1159522    5.4285104    4.3275671
## IRQYtmVYmDz0    7.2186835    5.9094326    5.3897706    7.2417859    6.4655110
## x1rCQxul53sT    4.5619767    8.2937449    6.6405583    5.8270774    6.0762217
## N8NdoNXifb0E    3.0348334    6.7806396    4.8755325    3.2076581    2.6250991
## mcF3ZYA08Otk    4.8483557    5.5458372    4.2132615    4.0583635    3.7437978
## 0I15liRzEWsY    3.8981274    5.0584559    3.5524139    3.3153674    1.2342058
## JLrRAGZNDhsa    4.1608734    5.9984537    4.5922629    3.6771319    2.9349406
## 7hz2WvdmnFyz    2.4192168    7.1169045    5.1832028    3.4942209    3.4467993
## vRb4qBGZYWJ5    2.6709383    7.2700590    5.4722256    1.2725419    3.4008217
## NIsa6HfyyDJG    6.3915855    1.6153221    1.2443667    5.9490706    4.8429291
## nPfUP9LQGjN5    4.4831504    4.9298950    3.6187895    3.7161327    2.8421813
## I8UeXsVicrNl    2.4644455    7.4255305    5.6189615    3.4264555    3.3709208
## AAoVQNFVGEYX    3.0599576    7.8817086    6.1861839    4.3556853    4.2099866
## 1DgI8w7LHJKO    2.8344778    7.2133197    5.4205462    1.2592628    3.3096280
## Fl4D8tyQt6JD    1.8985111    8.0318402    6.3647280    3.4592948    4.3988515
## cMxBm0i4DQ5e    3.9061840    5.6068936    3.9843898    2.8012970    2.4902475
## RriYEaRbFplB    3.1251260    7.3424724    5.5439966    3.0895044    2.7351358
## iUOWKy71j72s    2.5896799    7.6958951    6.0195015    1.7725283    3.6673231
## lSwlRJFMkWaL    4.9196588    5.6498893    4.3855120    4.0181289    3.5225334
## S7pIj7jJ5sYd    2.7378840    7.0111780    5.1484907    1.1574009    3.1235387
## j3ao6E3OOfh6    2.5294167    7.0389209    5.1209058    2.8625140    2.7906906
## 8099hdnTqjgV    4.0763210    4.2809286    2.5827116    3.1027418    1.9685587
## iIE5hTCkcKXT    2.1820421    8.2080285    6.6139156    3.4732534    4.4533714
## cMZJBv6hI69I    3.6022036    5.4837529    3.8907062    2.9055883    0.8112113
## fWBii4xp1GzE    2.1952060    7.3882244    5.6211403    1.2629510    3.1779527
## c8iiBzBoWUTo    3.9774851    6.1750675    4.3932203    3.2904415    1.9848309
## ONHXGmNh8dIs    3.2444653    7.8774631    6.1630655    4.0908098    3.6957694
## 139Jnx308F75    3.8070269    7.3194127    5.4040404    4.3138619    4.4408241
## Jf6emLIOfmdj    4.1611566    9.4203029    8.0594500    5.0793654    5.7920885
## OrPHld09mheD    6.4559918    8.1370835    6.9592598    6.6574224    6.2754420
## 1mca8JBV02cD    7.6214050    5.8249175    5.4847231    7.3894501    6.4808442
## ItjbeSEfdbcL    7.4388135    4.6910231    4.6464941    7.1666146    5.9854820
## Ipryv7a7GXCB    4.6883135    4.8163611    3.3865545    3.8237700    2.9071550
## f0feu6GNkLXl    5.5799265    7.0849173    5.6456938    5.1657135    4.6356575
## 1Koe5mOVjSO1    3.6199779    9.1382662    7.7085057    5.0864753    5.6436109
```

```
## pdFmvb7S0FVG    5.5998176    6.4999446    5.2003479    5.0605945    4.1981278
## ITaszdywMT4z    5.8671530    6.6903204    5.3654065    5.3549176    4.7382022
## XxbmRluFogMr    2.9765591    8.8075270    7.3127476    4.4392019    5.1870852
##              zGadHigIDv6z kpej7M5rrkJt sij8JzPPvqW1 Kzcj7AjWcK4i IRQYtmVYmDz0
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt    1.3208704
## sij8JzPPvqW1    3.9463193    3.7482951
## Kzcj7AjWcK4i    5.6178346    5.6806594    3.7773531
## IRQYtmVYmDz0    7.1402509    7.2980048    6.2066443    5.3115507
## x1rCQxul53sT    6.1302522    5.4159811    6.5286643    7.1134600    7.8648282
## N8NdoNXifb0E    3.9265796    3.8379683    3.8382067    5.3680230    6.4317076
## mcF3ZYA08Otk    4.7385759    4.3013246    1.9803500    4.0946814    6.7914714
## 0I15liRzEWsY    3.7934251    3.9681782    2.6236411    3.8121682    6.2030239
## JLrRAGZNDhsa    4.3378452    4.0528844    2.7659518    4.7696531    6.9173322
## 7hz2WvdmnFyz    4.0597718    3.6076934    4.4641198    5.6978451    6.7409123
## vRb4qBGZYWJ5    0.8872871    1.2445979    4.1975448    5.8880274    7.5349964
## NIsa6HfyyDJG    6.0748074    6.1558666    4.4567026    1.1251750    5.1688983
## nPfUP9LQGjN5    4.3064700    4.0850460    1.5908351    3.6396165    6.3685121
## I8UeXsVicrNl    4.1951095    3.5479064    4.1885738    5.9831479    7.1674942
## AAoVQNFVGEYX    5.0343901    4.3292967    5.0213099    6.5651853    7.6042165
## 1DgI8w7LHJKO    0.7226232    1.2744014    4.1238608    5.8178252    7.5120694
## Fl4D8tyQt6JD    4.0678612    3.1939013    4.7940237    6.6778865    7.4485122
## cMxBm0i4DQ5e    3.4481018    3.0825934    1.7151997    4.0730347    6.4683279
## RriYEaRbFplB    3.9049196    3.5150588    3.9201586    5.9085998    7.5724225
## iUOWKy71j72s    1.6727937    1.5015420    4.5286762    6.4063960    7.9369184
## lSwlRJFMkWaL    4.6660654    4.4322723    1.2881411    4.2690436    6.4360382
## S7pIj7jJ5sYd    0.6987514    1.3382223    3.9602936    5.5652246    7.3181844
## j3ao6E3OOfh6    3.6499237    3.1139782    3.6590604    5.4995100    6.8916151
## 8099hdnTqjgV    3.2870634    3.5036466    2.5208200    3.0197419    5.7686595
## iIE5hTCkcKXT    4.1290987    3.2142676    4.6664026    6.8321763    7.7840464
## cMZJBv6hI69I    3.4432021    3.6133090    2.8975692    4.1839188    6.4877967
## fWBii4xp1GzE    1.3536818    1.2572823    4.1629286    6.0333215    7.6394150
## c8iiBzBoWUTo    3.7216215    3.8851656    3.2928283    4.6147576    6.3020246
## ONHXGmNh8dIs    4.8639040    4.4853676    4.9839992    6.6234080    7.8773389
## 139Jnx308F75    4.5874048    4.4872150    5.1305596    5.9488579    6.4374872
## Jf6emLIOfmdj    5.8501180    5.0494892    5.7193875    8.1354907    9.0228122
```

```
## OrPHld09mheD    6.3706708    6.6386436    6.6473595    7.1059836    4.9110148
## 1mca8JBV02cD    7.2267701    7.5022801    6.4268988    5.3239287    1.9794463
## ItjbeSEfdbcL    7.0766054    7.3079309    5.8214008    4.3837301    2.5413991
## Ipryv7a7GXCB    4.3316160    4.2581731    1.2777431    3.3336965    5.7954352
## f0feu6GNkLXl    5.3061810    5.3698809    4.3659740    5.7906779    6.2113682
## 1Koe5mOVjSO1    5.8277512    4.9591618    5.7607124    7.8680747    8.6330463
## pdFmvb7S0FVG    5.1405798    5.2931175    3.9886304    5.2556670    5.9093609
## ITaszdywMT4z    5.4431331    5.5751789    4.2130729    5.5043629    5.8783771
## XxbmRluFogMr    5.1574009    4.3365331    5.4114470    7.5192644    8.4676732
##              x1rCQxul53sT N8NdoNXifb0E mcF3ZYA08Otk 0I15liRzEWsY JLrRAGZNDhsa
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt
## sij8JzPPvqW1
## Kzcj7AjWcK4i
## IRQYtmVYmDz0
## x1rCQxul53sT
## N8NdoNXifb0E    5.1612054
## mcF3ZYA08Otk    6.4015408    4.2811922
## 0I15liRzEWsY    6.1966713    2.7051226    3.4700620
## JLrRAGZNDhsa    6.3895404    3.3696664    2.5273587    2.8080998
## 7hz2WvdmnFyz    3.3941973    2.1155108    4.7046503    3.6439729    4.2277828
## vRb4qBGZYWJ5    6.1571814    4.1134832    4.9659602    4.0930368    4.5361635
## NIsa6HfyyDJG    7.2471664    5.6428944    4.7814154    4.2671236    5.1226034
## nPfUP9LQGjN5    6.4692338    3.7051757    1.5044707    2.4072757    1.7585787
## I8UeXsVicrNl    3.9567798    2.2894401    4.1311506    3.6543273    3.4862997
## AAoVQNFVGEYX    3.4018721    3.1263471    4.7939083    4.4056542    4.0505197
## 1DgI8w7LHJKO    6.3091811    4.1644878    4.9004214    4.0091721    4.4750179
## Fl4D8tyQt6JD    4.2448708    3.5475346    4.7076277    4.7392336    4.3448731
## cMxBm0i4DQ5e    5.9859441    3.3720467    1.8497927    2.6575022    1.7134417
## RriYEaRbFplB    4.9895226    2.3567830    3.9584393    3.1745758    2.8097372
## iUOWKy71j72s    6.2282030    4.3530223    5.1509477    4.4228930    4.3849817
## lSwlRJFMkWaL    7.0337591    4.1646856    1.9227165    3.2586649    2.6030341
## S7pIj7jJ5sYd    6.0792334    3.9168378    4.7282442    3.7513833    4.4084950
## j3ao6E3OOfh6    4.2280750    1.9073307    3.7861250    3.1658493    3.2851582
## 8099hdnTqjgV    6.0164937    3.4577465    3.4507686    1.8364105    3.3971886
## iIE5hTCkcKXT    5.0951757    3.9248625    4.5502711    4.8497613    4.2342777
```

```
## cMZJBv6hI69I    6.0764440    2.5360606    3.7273324    0.9816779    3.0497550
## fWBii4xp1GzE    5.8934468    3.7915045    4.7874852    3.8982479    4.0408977
## c8iiBzBoWUTo    5.6906130    2.6864448    4.1991695    2.0439704    3.8166387
## ONHXGmNh8dIs    4.7213568    2.4305624    5.0539780    3.9297991    3.7804222
## 139Jnx308F75    3.9111298    3.1032108    5.4861038    4.5618663    5.4237953
## Jf6emLIOfmdj    6.8249304    5.2202480    5.3244203    6.0985963    5.0153379
## OrPHld09mheD    7.2726540    6.0717046    7.1837744    6.3057948    7.2681067
## 1mca8JBV02cD    8.4146176    6.7273304    6.8570942    6.1959269    7.0152736
## ItjbeSEfdbcL    8.1643584    6.4976055    6.3411656    5.5796669    6.5677903
## Ipryv7a7GXCB    6.5468274    3.7255040    2.1939603    2.4057973    2.6398651
## f0feu6GNkLXl    6.5420796    5.0259374    5.8600372    4.6838908    5.9110537
## 1Koe5mOVjSO1    5.5725634    4.7743402    5.3230077    5.8645632    4.9680348
## pdFmvb7S0FVG    6.6433049    4.9767210    5.5372808    4.1734619    5.6421182
## ITaszdywMT4z    6.8701547    5.2555962    5.7194326    4.6682230    5.8015316
## XxbmRluFogMr    5.3699386    4.4249136    5.0822707    5.4343770    4.6603923
##              7hz2WvdmnFyz vRb4qBGZYWJ5 NIsa6HfyyDJG nPfUP9LQGjN5 I8UeXsVicrNl
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt
## sij8JzPPvqW1
## Kzcj7AjWcK4i
## IRQYtmVYmDz0
## x1rCQxul53sT
## N8NdoNXifb0E
## mcF3ZYA08Otk
## 0I15liRzEWsY
## JLrRAGZNDhsa
## 7hz2WvdmnFyz
## vRb4qBGZYWJ5    4.1995951
## NIsa6HfyyDJG    5.9431039    6.3045425
## nPfUP9LQGjN5    4.4397476    4.5339784    4.1585582
## I8UeXsVicrNl    1.4911649    4.3187405    6.3416473    4.0305896
## AAoVQNFVGEYX    2.0597615    5.1136337    6.8487492    4.7287550    1.2080936
## 1DgI8w7LHJKO    4.2903201    0.4253614    6.2602163    4.4664870    4.3830421
## Fl4D8tyQt6JD    2.8983125    3.8645598    7.0060967    4.7207506    2.3515953
## cMxBm0i4DQ5e    3.9386746    3.7110714    4.6396470    1.4344302    3.4624531
## RriYEaRbFplB    2.6733306    4.1035808    6.3664524    3.6903958    1.7598410
```

```
## iUOWKy71j72s    4.4119483    1.1551442    6.7848700    4.7030468    4.2467883
## lSwlRJFMkWaL    4.9522282    4.8633695    4.8511790    1.7785633    4.4793985
## S7pIj7jJ5sYd    4.0179242    0.7248547    6.0261881    4.3151173    4.2014515
## j3ao6E3OOfh6    1.4173409    3.8688180    5.9242242    3.6479504    0.9156875
## 8099hdnTqjgV    3.9111473    3.6529573    3.5160169    2.5804608    4.1790814
## iIE5hTCkcKXT    3.3378438    3.9550421    7.2023855    4.6329789    2.5503208
## cMZJBv6hI69I    3.3828197    3.7601769    4.7104079    2.8909946    3.3322076
## fWBii4xp1GzE    3.8914821    0.9551055    6.4340786    4.3280604    3.7749585
## c8iiBzBoWUTo    3.3129830    4.0760682    5.1035557    3.5359952    3.6346480
## ONHXGmNh8dIs    2.5837269    4.9004140    6.8879503    4.6624310    1.9616445
## 139Jnx308F75    2.5338556    4.8495646    6.1712318    5.2889335    3.4445784
## Jf6emLIOfmdj    5.1119684    5.6557618    8.5461225    5.5770643    4.0057271
## OrPHld09mheD    5.9025223    6.9789653    7.2641478    6.8854841    6.3987157
## 1mca8JBV02cD    7.2580067    7.6716852    5.2978295    6.3955998    7.5994099
## ItjbeSEfdbcL    7.0378109    7.4641961    4.3111365    5.7554314    7.3943834
## Ipryv7a7GXCB    4.4887377    4.5963696    3.8700399    1.4664182    4.3584568
## f0feu6GNkLXl    5.0445133    5.5718410    6.1343495    5.4571748    5.5216042
## 1Koe5mOVjSO1    4.2414269    5.6603759    8.2037938    5.5400402    3.2081510
## pdFmvb7S0FVG    5.1000461    5.4767053    5.6466444    5.0313157    5.5448426
## ITaszdywMT4z    5.3922163    5.7497676    5.8206063    5.2781832    5.8147060
## XxbmRluFogMr    3.9711024    4.9296485    7.8682050    5.2115745    3.0224132
##              AAoVQNFVGEYX 1DgI8w7LHJKO Fl4D8tyQt6JD cMxBm0i4DQ5e RriYEaRbFplB
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt
## sij8JzPPvqW1
## Kzcj7AjWcK4i
## IRQYtmVYmDz0
## x1rCQxul53sT
## N8NdoNXifb0E
## mcF3ZYA08Otk
## 0I15liRzEWsY
## JLrRAGZNDhsa
## 7hz2WvdmnFyz
## vRb4qBGZYWJ5
## NIsa6HfyyDJG
## nPfUP9LQGjN5
```

```
## I8UeXsVicrNl
## AAoVQNFVGEYX
## 1DgI8w7LHJKO    5.2023581
## Fl4D8tyQt6JD    2.6073347    4.0553042
## cMxBm0i4DQ5e    4.2453775    3.6327509    4.1562850
## RriYEaRbFplB    2.2767289    4.1092968    3.2814537    3.0145752
## iUOWKy71j72s    4.9018080    1.3182930    3.5582159    3.8409286    3.9387146
## lSwlRJFMkWaL    5.2522612    4.8029127    5.0251825    2.1379564    4.2042631
## S7pIj7jJ5sYd    5.0392099    0.5200352    4.0116279    3.5245325    3.9663135
## j3ao6E3OOfh6    2.0021164    3.8931659    2.7813693    2.9633821    1.6453331
## 8099hdnTqjgV    4.9402596    3.5616486    4.9696047    2.6072066    3.8766704
## iIE5hTCkcKXT    3.0660847    4.0885827    1.4395022    4.0511411    3.5016753
## cMZJBv6hI69I    4.1454878    3.6712345    4.4805474    2.7880123    2.7395339
## fWBii4xp1GzE    4.4858440    1.0989867    3.3066748    3.4889360    3.4558615
## c8iiBzBoWUTo    4.3925912    4.0051297    4.8234605    3.1962627    3.3295561
## ONHXGmNh8dIs    1.9852158    5.0252186    3.2361791    4.2563991    1.7743943
## 139Jnx308F75    3.8407739    4.9977220    4.0474671    4.8735792    4.1455177
## Jf6emLIOfmdj    4.3503252    5.7785165    3.1480142    5.2147658    4.6098599
## OrPHld09mheD    6.9224837    6.9198396    6.8799780    6.6930792    7.0543674
## 1mca8JBV02cD    8.0465550    7.6187915    7.8403257    6.5613540    7.7649955
## ItjbeSEfdbcL    7.8323100    7.4280031    7.7018057    6.1011039    7.5153262
## Ipryv7a7GXCB    5.1196570    4.5218983    5.0957736    1.9724263    4.1273262
## f0feu6GNkLXl    6.1014072    5.5174796    6.2750324    5.1780703    5.6459977
## 1Koe5mOVjSO1    3.2574645    5.8054971    2.3866836    5.1953775    4.1863353
## pdFmvb7S0FVG    6.1422513    5.4011595    6.3276617    4.8674116    5.5724719
## ITaszdywMT4z    6.3869224    5.6778561    6.5654268    5.1268745    5.8410590
## XxbmRluFogMr    3.1456467    5.1029718    1.8053143    4.8298172    3.8110010
##              iUOWKy71j72s lSwlRJFMkWaL S7pIj7jJ5sYd j3ao6E3OOfh6 8099hdnTqjgV
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt
## sij8JzPPvqW1
## Kzcj7AjWcK4i
## IRQYtmVYmDz0
## x1rCQxul53sT
## N8NdoNXifb0E
## mcF3ZYA08Otk
```

```
## 0I15liRzEWsY
## JLrRAGZNDhsa
## 7hz2WvdmnFyz
## vRb4qBGZYWJ5
## NIsa6HfyyDJG
## nPfUP9LQGjN5
## I8UeXsVicrNl
## AAoVQNFVGEYX
## 1DgI8w7LHJKO
## Fl4D8tyQt6JD
## cMxBm0i4DQ5e
## RriYEaRbFplB
## iUOWKy71j72s
## lSwlRJFMkWaL    5.0669434
## S7pIj7jJ5sYd    1.7117137    4.6680928
## j3ao6E3OOfh6    3.9808901    4.0916221    3.6679699
## 8099hdnTqjgV    4.1670704    3.5007885    3.3103672    3.5622587
## iIE5hTCkcKXT    3.6041289    4.8034563    4.1038536    2.9166600    5.1268932
## cMZJBv6hI69I    4.0617654    3.6037743    3.4270867    2.7931103    1.9561842
## fWBii4xp1GzE    0.6641380    4.7214333    1.3247253    3.4581947    3.7282317
## c8iiBzBoWUTo    4.5937410    3.9849128    3.6674822    3.0106560    2.8566016
## ONHXGmNh8dIs    4.6637477    5.1585227    4.8943549    2.3492004    4.7662186
## 139Jnx308F75    5.2721595    5.6836106    4.7012616    3.2099373    4.5202667
## Jf6emLIOfmdj    5.1538701    5.4927497    5.8523260    4.4707463    6.6675666
## OrPHld09mheD    7.4215228    7.1668222    6.7088610    6.1085919    6.1160725
## 1mca8JBV02cD    8.0740973    6.7665304    7.4294883    7.2751021    5.7818602
## ItjbeSEfdbcL    7.8737290    6.2207216    7.2318646    7.0611054    5.1796133
## Ipryv7a7GXCB    4.9775921    1.4666412    4.2985210    3.8673427    2.6157441
## f0feu6GNkLXl    6.1112592    4.7981759    5.2504492    5.0545374    4.7593879
## 1Koe5mOVjSO1    5.2108540    5.6394493    5.8025038    3.8869343    6.4318491
## pdFmvb7S0FVG    6.0305343    4.5560014    5.1280756    5.0450882    4.2768260
## ITaszdywMT4z    6.2795597    4.5800409    5.4187347    5.3402661    4.6086953
## XxbmRluFogMr    4.4601240    5.3540901    5.1075238    3.6330289    5.9635759
##              iIE5hTCkcKXT cMZJBv6hI69I fWBii4xp1GzE c8iiBzBoWUTo ONHXGmNh8dIs
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt
```

```
## sij8JzPPvqW1
## Kzcj7AjWcK4i
## IRQYtmVYmDz0
## x1rCQxul53sT
## N8NdoNXifb0E
## mcF3ZYA08Otk
## 0I15liRzEWsY
## JLrRAGZNDhsa
## 7hz2WvdmnFyz
## vRb4qBGZYWJ5
## NIsa6HfyyDJG
## nPfUP9LQGjN5
## I8UeXsVicrNl
## AAoVQNFVGEYX
## 1DgI8w7LHJKO
## Fl4D8tyQt6JD
## cMxBm0i4DQ5e
## RriYEaRbFplB
## iUOWKy71j72s
## lSwlRJFMkWaL
## S7pIj7jJ5sYd
## j3ao6E3OOfh6
## 8099hdnTqjgV
## iIE5hTCkcKXT
## cMZJBv6hI69I    4.5686907
## fWBii4xp1GzE    3.4138932    3.5231602
## c8iiBzBoWUTo    5.0534906    2.1893155    4.0354707
## ONHXGmNh8dIs    3.6918950    3.6008260    4.2164094    4.0935829
## 139Jnx308F75    4.7530718    4.3937814    4.7981458    3.9922969    3.9541264
## Jf6emLIOfmdj    2.2305205    5.8445213    5.0475039    6.5134699    4.6085294
## OrPHld09mheD    6.9769433    6.3491699    7.1019757    5.9223024    7.4256984
## 1mca8JBV02cD    8.2467670    6.5074319    7.7844669    6.3665861    8.2091110
## ItjbeSEfdbcL    8.1040888    6.0123162    7.5735577    5.9028868    7.9327193
## Ipryv7a7GXCB    5.0720516    2.9990133    4.5569814    3.0091383    5.0384748
## f0feu6GNkLXl    6.4533042    4.8024819    5.7192420    3.9158545    6.2155815
## 1Koe5mOVjSO1    1.8447342    5.6373086    5.0183951    6.1619264    3.9087401
## pdFmvb7S0FVG    6.5032066    4.3874378    5.6326195    3.4678200    6.2267789
## ITaszdywMT4z    6.7347771    4.8678531    5.8984683    4.1849816    6.4682534
## XxbmRluFogMr    1.5061002    5.1960884    4.2681894    5.7347017    3.5611410
##              139Jnx308F75 Jf6emLIOfmdj OrPHld09mheD 1mca8JBV02cD ItjbeSEfdbcL
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
```

ldSNHx3DBSrb
TINTcMJTAyDw
DrPxteZyuZfS
i013JbphkzdZ
zGadHigIDv6z
kpej7M5rrkJt
sij8JzPPvqW1
Kzcj7AjWcK4i
IRQYtmVYmDz0
x1rCQxul53sT
N8NdoNXifb0E
mcF3ZYA08Otk
0I15liRzEWsY
JLrRAGZNDhsa
7hz2WvdmnFyz
vRb4qBGZYWJ5
NIsa6HfyyDJG
nPfUP9LQGjN5
I8UeXsVicrNl
AAoVQNFVGEYX
1DgI8w7LHJKO
Fl4D8tyQt6JD
cMxBm0i4DQ5e
RriYEaRbFplB
iUOWKy71j72s
lSwlRJFMkWaL
S7pIj7jJ5sYd
j3ao6E3OOfh6
8099hdnTqjgV
iIE5hTCkcKXT
cMZJBv6hI69I
fWBii4xp1GzE
c8iiBzBoWUTo
ONHXGmNh8dIs
139Jnx308F75
Jf6emLIOfmdj 6.3771022
OrPHld09mheD 5.1343706 8.3456466
1mca8JBV02cD 6.7886568 9.4433129 5.0376461
ItjbeSEfdbcL 6.6097676 9.3149948 5.4893695 1.5484661
Ipryv7a7GXCB 5.1241245 6.1572166 6.5739604 6.0128973 5.3466839
f0feu6GNkLXl 5.3622381 7.9240111 6.4801435 7.0799810 6.6894368
1Koe5mOVjSO1 5.6160333 1.5014535 7.9193341 9.1307063 8.9870581
pdFmvb7S0FVG 5.2751102 7.9750404 6.0499216 6.5404123 6.0202726
ITaszdywMT4z 5.7086473 8.1649737 6.5631338 6.7712205 6.3718288
XxbmRluFogMr 5.2695149 1.7574644 7.8503081 8.8981065 8.7222568

```
##              Ipryv7a7GXCB f0feu6GNkLXl 1Koe5mOVjSO1 pdFmvb7S0FVG ITaszdywMT4z
## p9R0s4hifwCL
## THscVqv9A2dL
## 7mskDeCfUXUu
## 6JGBfc2igrey
## AxE8m64pR3Cr
## ldSNHx3DBSrb
## TINTcMJTAyDw
## DrPxteZyuZfS
## i013JbphkzdZ
## zGadHigIDv6z
## kpej7M5rrkJt
## sij8JzPPvqW1
## Kzcj7AjWcK4i
## IRQYtmVYmDz0
## x1rCQxul53sT
## N8NdoNXifb0E
## mcF3ZYA08Otk
## 0I15liRzEWsY
## JLrRAGZNDhsa
## 7hz2WvdmnFyz
## vRb4qBGZYWJ5
## NIsa6HfyyDJG
## nPfUP9LQGjN5
## I8UeXsVicrNl
## AAoVQNFVGEYX
## 1DgI8w7LHJKO
## Fl4D8tyQt6JD
## cMxBm0i4DQ5e
## RriYEaRbFplB
## iUOWKy71j72s
## lSwlRJFMkWaL
## S7pIj7jJ5sYd
## j3ao6E3OOfh6
## 8099hdnTqjgV
## iIE5hTCkcKXT
## cMZJBv6hI69I
## fWBii4xp1GzE
## c8iiBzBoWUTo
## ONHXGmNh8dIs
## 139Jnx308F75
## Jf6emLIOfmdj
## OrPHld09mheD
## 1mca8JBV02cD
## ItjbeSEfdbcL
```

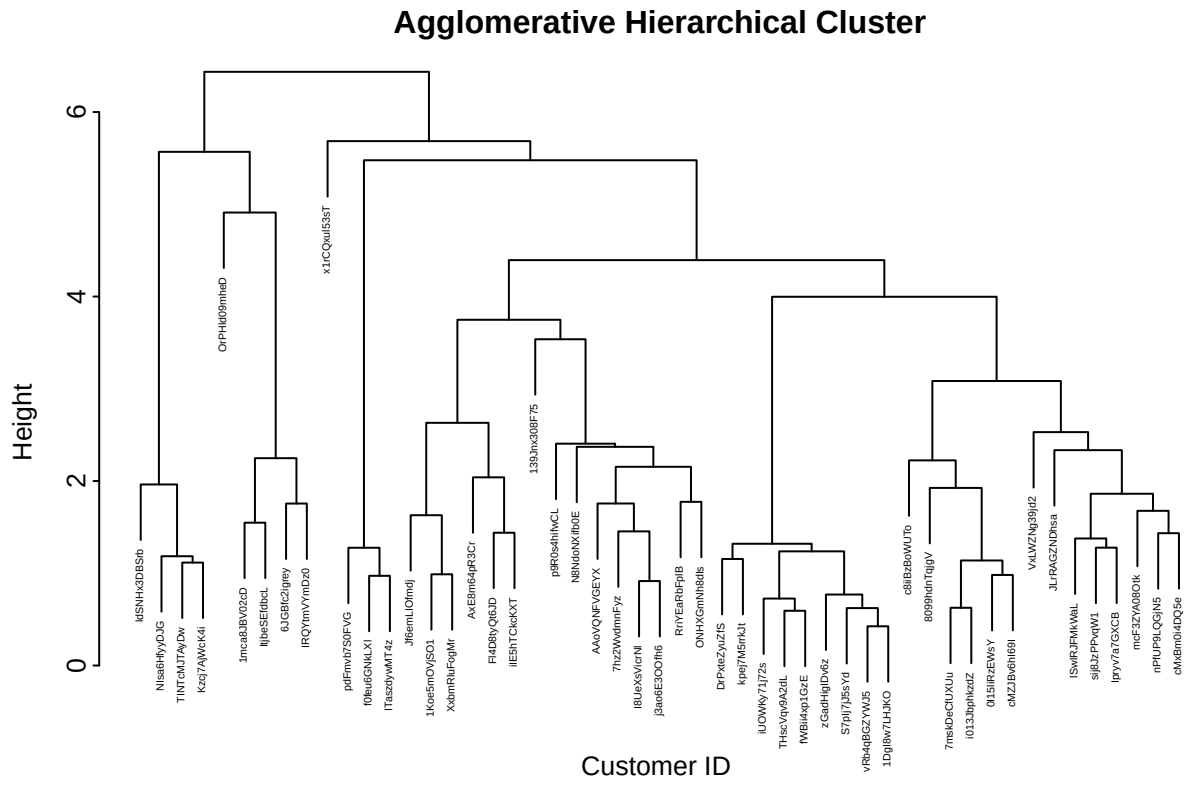
```
## Ipryv7a7GXCB
## f0feu6GNkLXl    4.3135630
## 1Koe5mOVjSO1    6.0707018    7.5442482
## pdFmvb7S0FVG    3.8910820    1.1586565    7.6228335
## ITaszdywMT4z    4.1106492    0.9726442    7.8213245    1.3946292
## XxbmRluFogMr    5.7191321    7.2163138    0.9889668    7.2740718    7.4818211
```

(a) Plot the dendrogram and provide it in your report. height is the value of the criterion associated with the clustering method for the particular agglomeration. (3 points) NOTE: Use the name in the chunk below to store the output from hclust. Complete the code.

```r
# create dendrogram for clustering
Hierar_cl0 <- hclust(mat, method = "average")
```

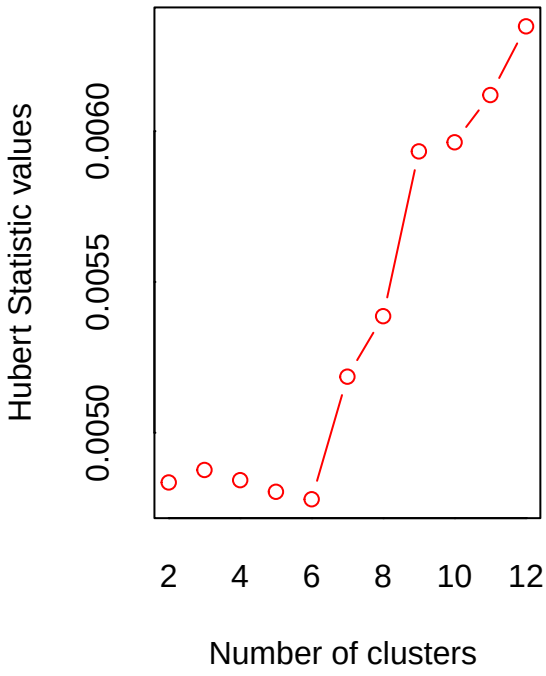
Run the following chunk to produce the dendrogram (formatted so it is easier to read).

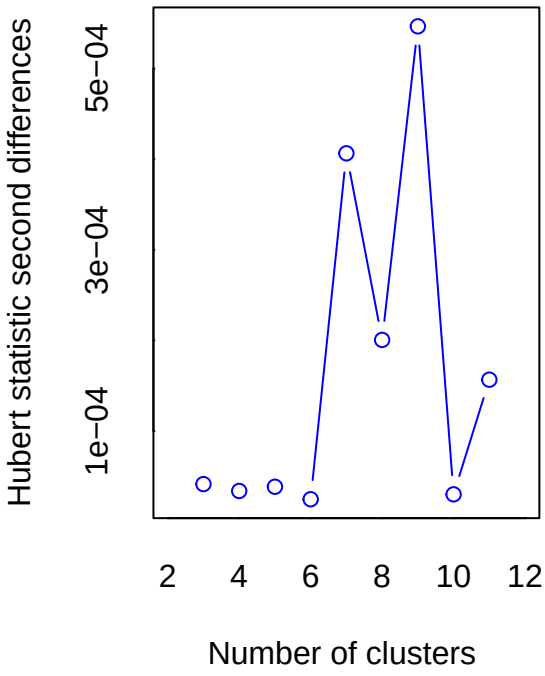
```r
# set plot parameters
par(cex=0.3, mar=c(6, 10, 4, 1))

# plot dendrogram
hcplot <- plot(Hierar_cl0, xlab="", ylab="", main="",
               sub="", axes=FALSE)
par(cex=0.8)
title(xlab="Customer ID \n \n", ylab="Height \n ",
      main="Agglomerative Hierarchical Cluster",
      sub="Unweighted Pair Group Method with Arithmetic Mean")
axis(2)
```



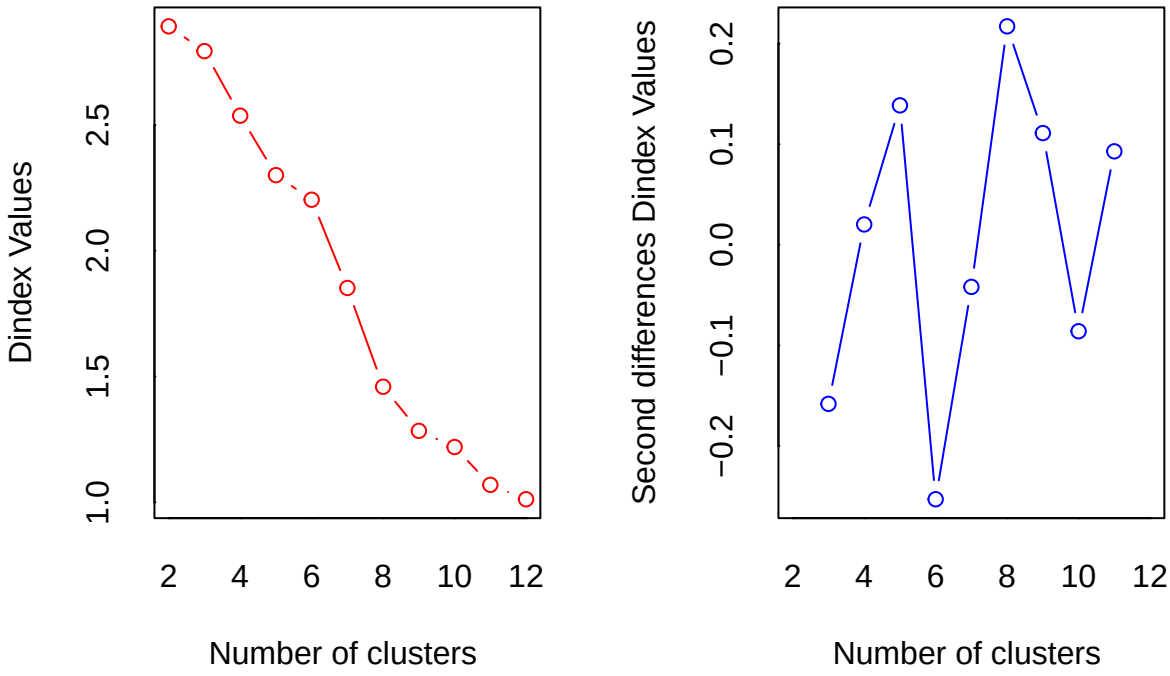
(b) Using the function NbClust with diss = NULL, distance = "euclidean", min.nc = 2, max.nc = 12, method = "average", and the data frame used to create the distance matrix, provide the methods and indices of Best.nc. (3 points)
NOTE: See part 2d for an example of NbClust. The method is different in this case, however.

```
num_clu<-NbClust(df7,diss=NULL,distance="euclidean",min.nc=2, max.nc=12,method="average"
```






```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that correspo
##                 significant increase of the value of the measure i.e the significant
##                 index second differences plot.
##
```




```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant p
##                 second differences plot) that corresponds to a significant increase o
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 6 proposed 2 as the best number of clusters
## * 1 proposed 3 as the best number of clusters
## * 2 proposed 4 as the best number of clusters
## * 2 proposed 5 as the best number of clusters
## * 4 proposed 6 as the best number of clusters
## * 1 proposed 7 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
## * 2 proposed 11 as the best number of clusters
## * 3 proposed 12 as the best number of clusters
##
##                     ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
```

```
##
##
## *****************************************************************
```

```
num_clu$Best.nc
```

```
##                      KL      CH Hartigan     CCC    Scott      Marriot
## Number_clusters    2.000 12.0000   6.0000 12.0000   4.0000 4.000000e+00
## Value_Index     1951.467 32.6788  16.8968 17.3589 172.4977 1.469881e+16
##                    TrCovW  TraceW Friedman   Rubin Cindex     DB Silhouette
## Number_clusters   7.0000  5.0000   6.0000  9.0000 8.0000 6.0000     11.000
## Value_Index     694.6724 45.5835  77.5495 -0.9401 0.3498 0.6279      0.514
##                    Duda PseudoT2  Beale Ratkowsky    Ball PtBiserial Frey
## Number_clusters 2.0000   2.0000 2.0000     2.000  3.0000     5.0000    1
## Value_Index     0.9969   0.1234 0.0311     0.341 84.6492     0.7018   NA
##                  McClain    Dunn Hubert SDindex Dindex    SDbw
## Number_clusters  2.0000 11.0000      0   6.000      0 12.0000
## Value_Index      0.2648  0.5275      0   0.467      0  0.1192
```

(c) Choose the best number of clusters using the above results or some other reason. Explain why you chose this number of clusters. (2 points) Based on the Best.nc from above, the optimal number of clusters is 12 which is very close to what the other methods (elbow and silhouette) calculated.