Clustering: K-Means and Hierarchical Clustering

Taylor Shrode

10/04/2020

Introduction

"Clustering is a technique used to group similar objects (close in terms of distance) together in the same group (cluster) (Yu-Wei, 2015)." Unlike supervised learning methods that we have used in previous assignments, clustering analysis does not use any label information. Instead, it simply uses the similarity between data features to group them into clusters (Yu-Wei, 2015). The four most common types of clustering methods are k-means clustering, hierarchical clustering, model-based clustering, and density-based clustering (Yu-Wei, 2015). We will be clustering our data using k-means clustering and hierarchical clustering.

The data set we will be using refers to the clients of a wholesale distributor and includes the annual spending in monetary units (m.u.) on diverse product categories (Margarida, 2013). The features included in this data set are (Margarida, 2013):

1. FRESH: annual spending (m.u.) on fresh products (Continuous)
2. MILK: annual spending (m.u.) on milk products (Continuous)
3. GROCERY: annual spending (m.u.)on grocery products (Continuous)
4. FROZEN: annual spending (m.u.) on frozen products (Continuous)
5. DETERGENTS_PAPER: annual spending (m.u.) on detergents and paper products (Continuous)
6. DELICATESSEN: annual spending (m.u.) on and delicatessen products (Continuous)
7. CHANNEL: customer Channel - 1 = Horeca (Hotel/Restaurant/Cafe) or 2 = Retail channel (Nominal)
8. REGION: customer Region - 1 = Lisbon, 2 = Oporto, or 3 = Other (Nominal)

We will be using the hierarchical and k-means clustering methods to cluster our data based on the features above.

Libraries

Before we begin building our models, we need to load the necessary libraries into R.

```
library(DataExplorer)
library(ggplot2)
library(caret)
library(dplyr)
library(cluster)
```

```
library("factoextra")
library(dendextend)
library(ape)
```

The **DataExplorer** library allows us to perform data exploration analysis and the **ggplot2** allows us to adjust the themes and colors in the plots. The **caret** package allows to create dummy variables for our nominal (discrete) variables. The **dplyr** package allows us to recode values in our data set (i.e. values in the REGION and CHANNEL columns). The **cluster** package allows us to use various clustering algorithms and the **factoextra** package allows us to visualize our clustering (K-means cluster analysis, n.d.). Finally, the **dendextend** package allows us to cut our dendrograms at different heights and the **ape** package allows us to view these cuts in various plots.

Load Data

To load our data into R, we can load the data directly from the URL using the **read.csv()** function.

```
wholesale_df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/00292/Wholesale customers data.csv")
head(wholesale_df)
```

```
##   Channel Region Fresh Milk Grocery Frozen Detergents_Paper Delicassen
## 1       2      3 12669 9656    7561    214              2674       1338
## 2       2      3  7057 9810    9568   1762              3293       1776
## 3       2      3  6353 8808    7684   2405              3516       7844
## 4       1      3 13265 1196    4221   6404               507       1788
## 5       2      3 22615 5410    7198   3915              1777       5185
## 6       2      3  9413 8259    5126    666              1795       1451
```

Notice, the values in the **Region** and **Channel** columns are numerical instead of categorical. To transform these values into dummy variables later, they should be categorical values. To recode these values, we can use the **recode()** function.

```
wholesale_df$Region <- recode(wholesale_df$Region, "1" = "Lisbon", "2" = "Oporto", "3" = "Other")
wholesale_df$Channel <- recode(wholesale_df$Channel, "1" = "Horeca", "2" = "Retail")
table(wholesale_df$Region)
```

```
##
## Lisbon Oporto  Other
##     77     47    316
```

```
table(wholesale_df$Channel)
```

```
##
## Horeca Retail
##    298    142
```

With our loaded data, we can perform data exploration analysis.

Data Exploration

To begin our analysis, we will view the structure and summary of our data.

```
str(wholesale_df)
```

```
## 'data.frame':    440 obs. of  8 variables:
##  $ Channel         : chr  "Retail" "Retail" "Retail" "Horeca" ...
##  $ Region          : chr  "Other" "Other" "Other" "Other" ...
##  $ Fresh           : int  12669 7057 6353 13265 22615 9413 12126 7579 5963
6006 ...
##  $ Milk            : int  9656 9810 8808 1196 5410 8259 3199 4956 3648
11093 ...
##  $ Grocery         : int  7561 9568 7684 4221 7198 5126 6975 9426 6192
18881 ...
##  $ Frozen          : int  214 1762 2405 6404 3915 666 480 1669 425 1159
...
##  $ Detergents_Paper: int  2674 3293 3516 507 1777 1795 3140 3321 1716 7425
...
##  $ Delicassen      : int  1338 1776 7844 1788 5185 1451 545 2566 750 2098
...
```

```
summary(wholesale_df)
```

```
##    Channel             Region              Fresh             Milk
##  Length:440         Length:440         Min.   :     3   Min.   :   55
##  Class :character   Class :character   1st Qu.:  3128   1st Qu.: 1533
##  Mode  :character   Mode  :character   Median :  8504   Median : 3627
##                                        Mean   : 12000   Mean   : 5796
##                                        3rd Qu.: 16934   3rd Qu.: 7190
##                                        Max.   :112151   Max.   :73498
##     Grocery          Frozen        Detergents_Paper    Delicassen
##  Min.   :    3   Min.   :   25.0   Min.   :    3.0   Min.   :    3.0
##  1st Qu.: 2153   1st Qu.:  742.2   1st Qu.:  256.8   1st Qu.:  408.2
##  Median : 4756   Median : 1526.0   Median :  816.5   Median :  965.5
##  Mean   : 7951   Mean   : 3071.9   Mean   : 2881.5   Mean   : 1524.9
##  3rd Qu.:10656   3rd Qu.: 3554.2   3rd Qu.: 3922.0   3rd Qu.: 1820.2
##  Max.   :92780   Max.   :60869.0   Max.   :40827.0   Max.   :47943.0
```

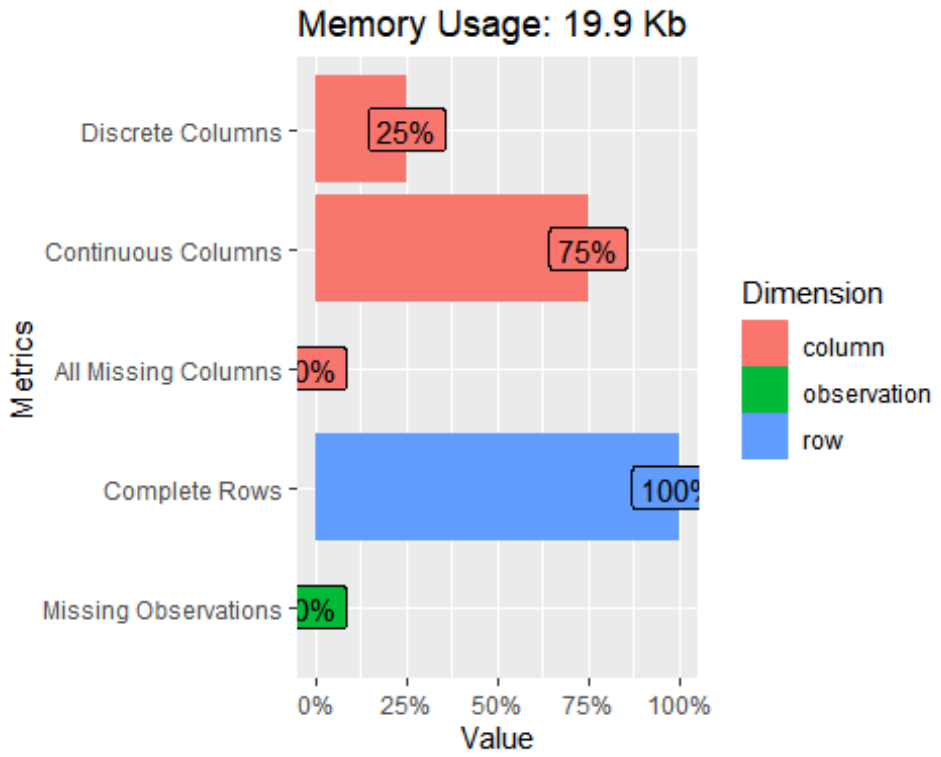
From the outputs above, we can see that our data set contains 440 rows of data and 8 different variables/features. Two of our variables are "*characters*" and the remainder of our variables are "*integers*". The summary output above suggests that each of the *integer* columns contain a wide range of values. We will need to scale these columns later so that our clustering algorithms do not depend on an arbitrary variable unit (K-means cluster analysis, n.d.).

```
introduce(wholesale_df)
```

```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1  440       8                2                  6                   0
```

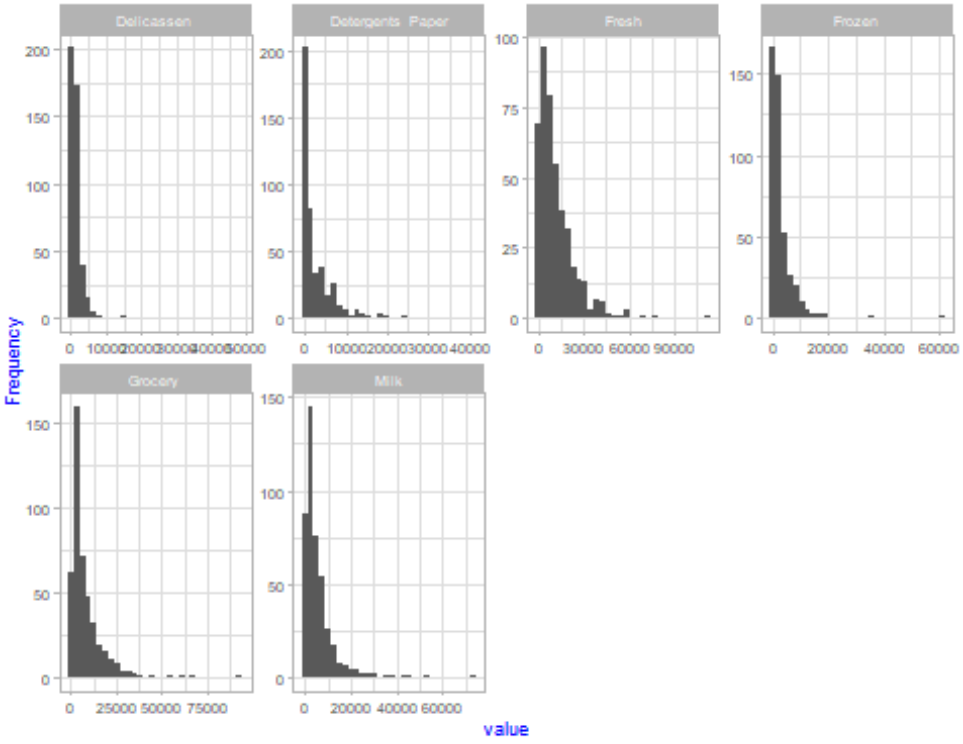
```
##   total_missing_values complete_rows total_observations memory_usage
## 1                    0           440               3520        20384
```

```
plot_intro(wholesale_df)
```



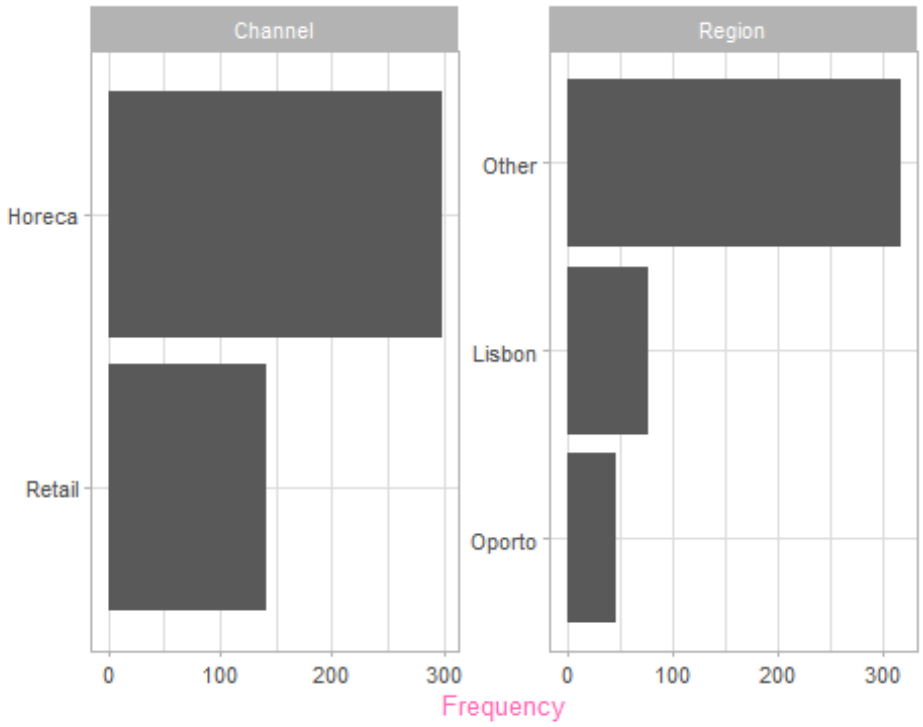
The output above suggests that we do not have any missing values on our data set and six of our eight (75%) columns contain continuous values. To view the distribution of these values, we can use the **plot_histogram()** function.

```
plot_histogram(wholesale_df, ggtheme = theme_light(base_size = 7),
theme_config = list("text" = element_text(color = "blue")))
```



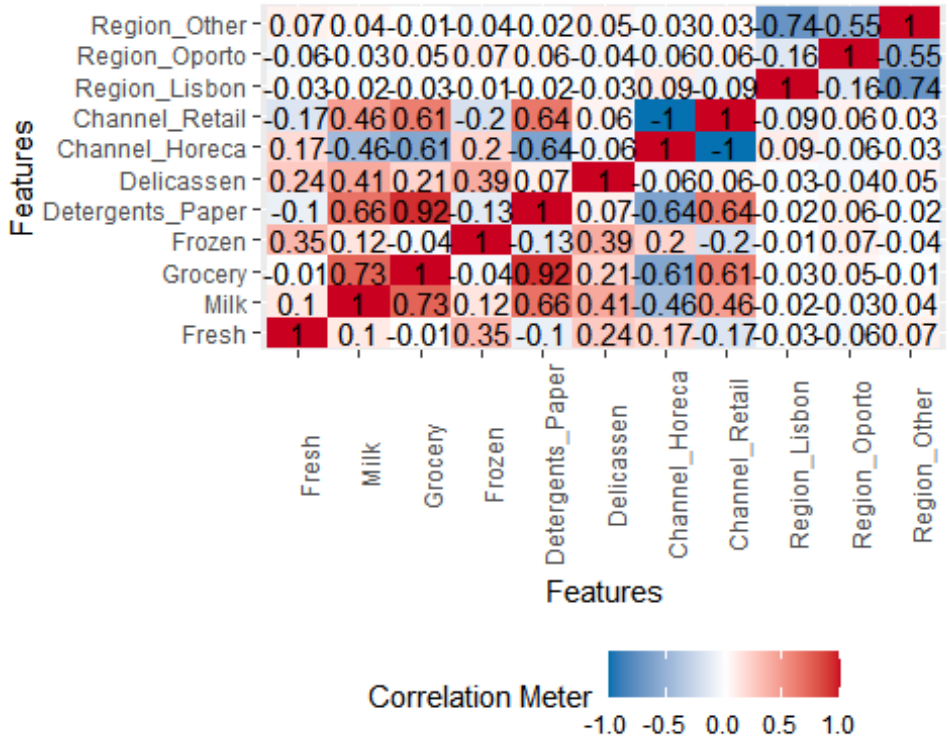
Notice, none of the continuous variables in our data set are normally distributed and the majority of the annual spending values (in monetary units (m.u.)) are “cheaper” values. Next, to view the distributions for our discrete values, we can use a bar chart.

```
plot_bar(wholesale_df, ggtheme = theme_light(base_size = 10), theme_config =
list("text" = element_text(color = "hotpink")))
```



This output suggests that most of our values are in the “Other” Region and belong to the “Horeca” Channel (i.e. Hotels/Restaurants/Cafes). Now, because clustering relies on the similarities between features, we can plot a correlation matrix for our continuous variables to determine how correlated our features are with each other (What is a correlation matrix, 2018).

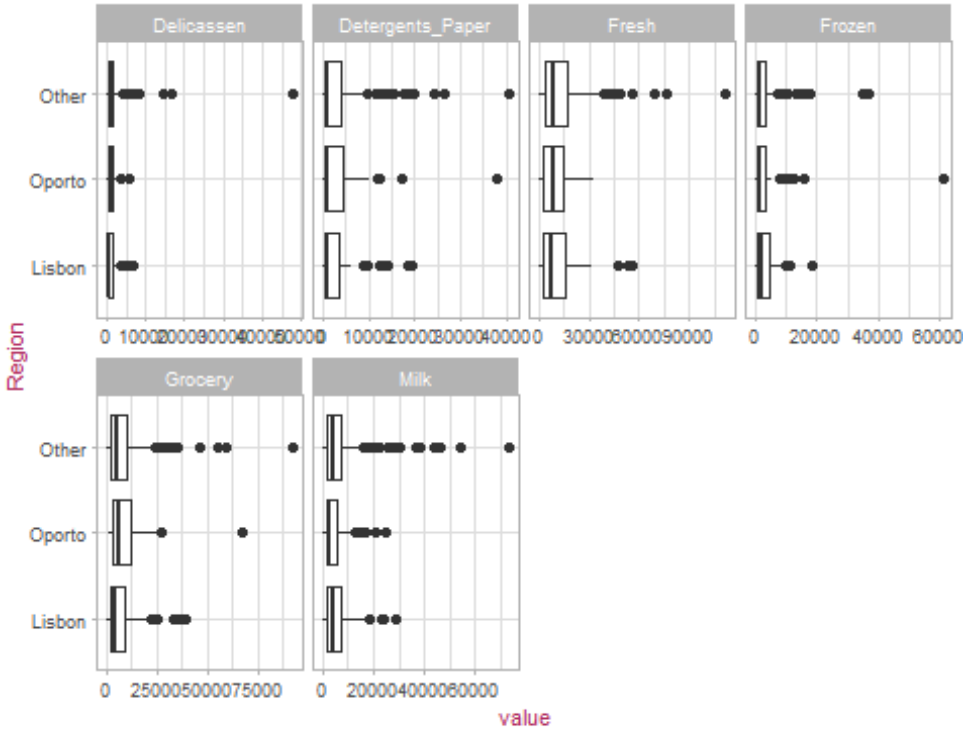
```
plot_correlation(wholesale_df)
```



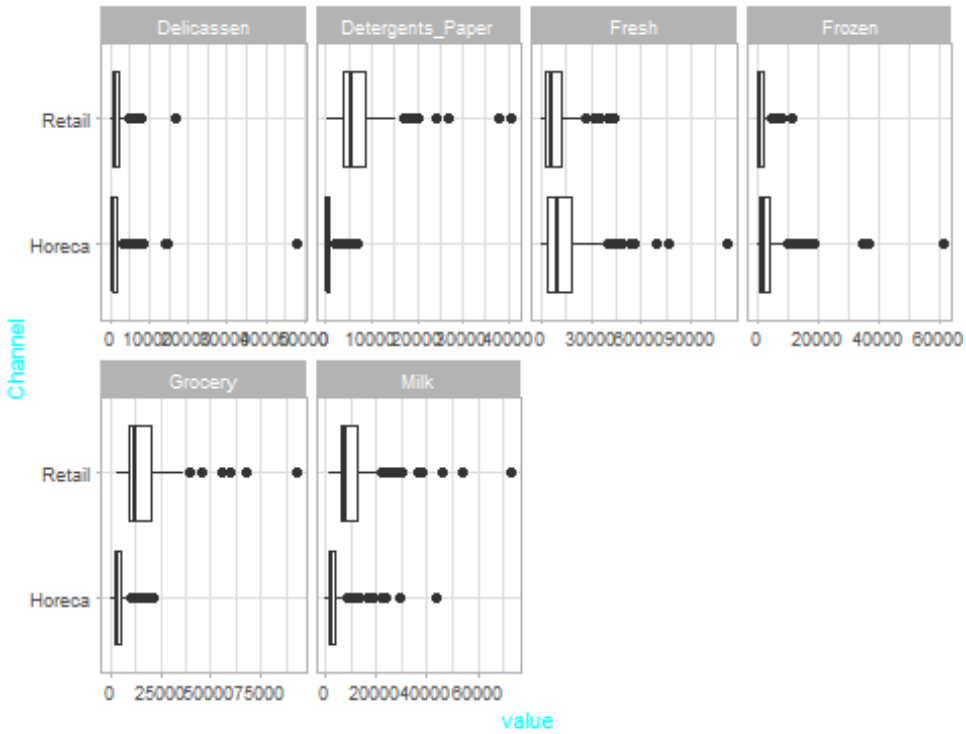
The first thing the output above suggests is that the **Grocery** feature is highly correlated with a few other features including **Milk**, **Detergents_Paper**, and the **Channel** values. Notice, the **Channel** values are also highly correlated to the **Milk** and **Detergents_Paper** features.

Finally, we will create boxplots for our continuous features and break down each continuous feature by a discrete feature.

```
plot_boxplot(wholesale_df, by = "Region", ggtheme = theme_light(base_size =
8), theme_config = list("text" = element_text(color = "maroon")))
```




```
plot_boxplot(wholesale_df, by = "Channel", ggtheme = theme_light(base_size = 8), theme_config = list("text" = element_text(color = "cyan")))
```



The outputs above suggest that the **Region** values appear to have similar outliers, medians, and minimum and maximum values. On the other hand, the **Channel** values do not demonstrate these similarities.

Data Preparation

As mentioned above, we need to convert our discrete values to dummy variables and we need to scale our continuous features so that we do not have any dominating features. First, we will transform our discrete features to dummy variables by using the **dummyVars()** function.

```
dmy <- dummyVars(" ~ Region + Channel", data = wholesale_df, fullRank = T)
trsf <- data.frame(predict(dmy, newdata = wholesale_df))
head(trsf)
```

```
##   RegionOporto RegionOther ChannelRetail
## 1            0           1             1
## 2            0           1             1
## 3            0           1             1
## 4            0           1             0
## 5            0           1             1
## 6            0           1             1
```

The dummy variables for the **Region** features can be interpreted as such:

- If **RegionOporto** = 0 and **RegionOther** = 1, then **Region** = Other.
- If **RegionOporto** = 1 and **RegionOther** = 0, then **Region** = Oporto.
- If **RegionOporto** = 0 and **RegionOther** = 0, then **Region** = Lisbon.

Similarly,

- If **ChannelRetail** = 1, then **Channel** = Retail.
- If **ChannelRetail** = 0, then **Channel** = Horeca.

Now, we can scale our continuous features by using the **scale()** function, which centers and/or scales the columns of a numeric matrix (R: scaling and centering of matrix-like objects, n.d.).

```
wholesale_scale <- scale(wholesale_df[,c(3:8)])
wholesale_final <- cbind(wholesale_scale, trsf)
head(wholesale_final)
```

```
##         Fresh        Milk     Grocery     Frozen Detergents_Paper
Delicassen
## 1  0.05287300  0.52297247 -0.04106815 -0.5886970      -0.04351919 -
0.06626363
## 2 -0.39085706  0.54383861  0.17012470 -0.2698290       0.08630859
0.08904969
## 3 -0.44652098  0.40807319 -0.02812509 -0.1373793       0.13308016
2.24074190
```

```
## 4  0.09999758 -0.62331041 -0.39253008  0.6863630      -0.49802132
0.09330484
## 5  0.83928412 -0.05233688 -0.07926595  0.1736612      -0.23165413
1.29786952
## 6 -0.20457266  0.33368675 -0.29729863 -0.4955909      -0.22787885 -
0.02619421
##   RegionOporto RegionOther ChannelRetail
## 1            0           1             1
## 2            0           1             1
## 3            0           1             1
## 4            0           1             0
## 5            0           1             1
## 6            0           1             1
```

Notice, we used the **cbind()** function to create our final dataframe that contains our scaled features and dummy variables.

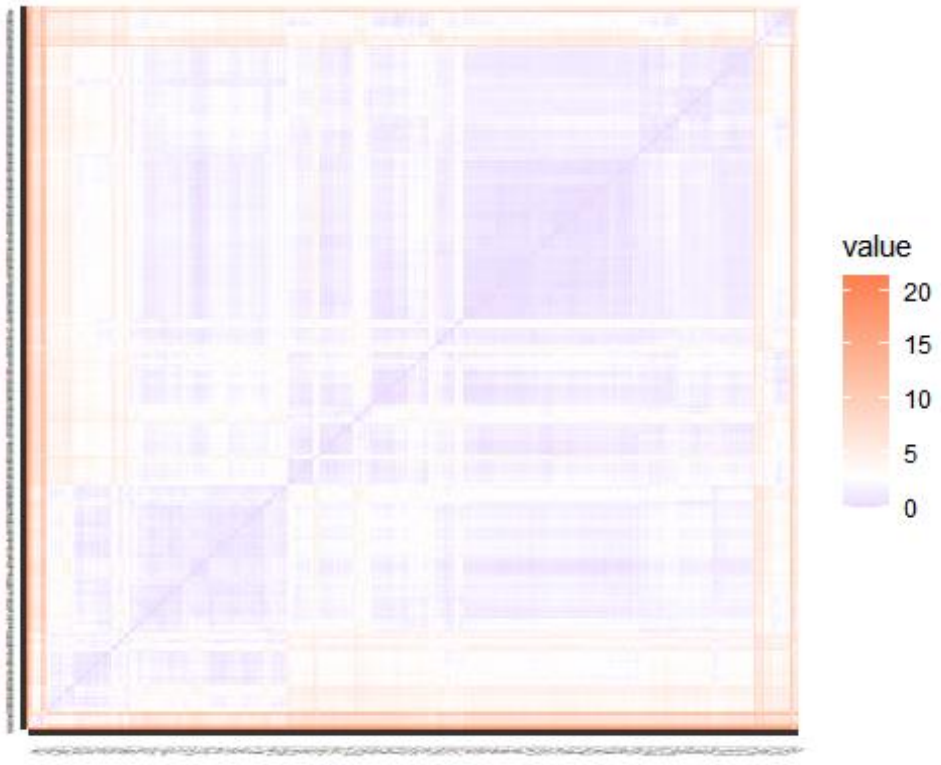
Part 1: Use K-Means Clustering

K-means clustering is "a flat clustering technique, which produces only one partition with *k* clusters (Yu-Wei, 2015)." The k-means clustering method is the simplest and the most commonly used unsupervised clustering method for partitioning a data set into a set of *k* groups, where *k* is the number of pre-defined clusters (K-means cluster analysis, n.d.). Each cluster is represented by its center (centroid) which corresponds to the mean of points assigned to the cluster (K-means cluster analysis, n.d.). The "points/objects within the same cluster are as similar (high intra-class similarity) as possible while the points from different clusters are as dissimilar (low inter-class similarity) as possible (K-means cluster analysis, n.d.)."

The basic idea behind this clustering method is that each cluster is defined so that the total intra-cluster variation (within-cluster variation) is minimized (K-means cluster analysis, n.d.). To classify each observation into groups requires the distance (similarity/dissimilarity) to be computed between each pair of observations (K-means cluster analysis, n.d.). There are many methods to do this, but the most common distance measure is the Euclidean distance. Other distance measures include the Manhattan distance, the Pearson Correlation distance, and the Spearman Correlation distance (K-means cluster analysis, n.d.).

To compute and visualize a distance matrix for our data set, we can use the **get_dist()** function to compute the distance matrix and the **fviz_dist()** function to visualize the matrix.

```
distance <- get_dist(wholesale_final, method = "euclidean")
fviz_dist(distance, gradient = list(low = "blue", mid = "white", high =
"coral"), lab_size = 0.5)
```



Now, while there are several k-means clustering algorithms, the standard is the Hartigan-Wong algorithm, "which defines the total within-cluster variation as the sum of squared distances Euclidean distances between items and the corresponding centroid (K-means cluster analysis, n.d.)."

Training Initial Model

The first step when using the k-means clustering method is to determine the the number of clusters prior to running the algorithm (Yu_Wei, 2015). For our first model, we will choose 10 for our *k* value. The algorithm begins by randomly selecting observations from the data set to serve as the initial center for the clusters (centroids) (K-means cluster analysis, n.d.). Then, each remaining observation is assigned to the closest centroid, defined using the Euclidean distance. This assignment of clusters continues for each observation and the algorithm continues to compute the new mean value of each cluster (K-means cluster analysis, n.d.). This process continues until convergence is achieved. In other words, until the algorithm has determined each observation belongs to the closest cluster centroid.

To compute our first k-means model, we can use the **kmeans()** function. We will define our *k* value to equal 10 and set our **nstart** option to 25. This will generate 25 initial configurations and then report the best one (K-means cluster analysis, n.d.).

```
set.seed(789)

wholesale_k10 <- kmeans(wholesale_final, 10, nstart = 25)
str(wholesale_k10)
```

```
## List of 9
##  $ cluster     : Named int [1:440] 5 5 7 4 6 5 5 5 3 5 ...
##   ..- attr(*, "names")= chr [1:440] "1" "2" "3" "4" ...
##  $ centers     : num [1:10, 1:9] 1.076 1.965 -0.439 0.237 -0.444 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:10] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:9] "Fresh" "Milk" "Grocery" "Frozen" ...
##  $ totss       : num 2861
##  $ withinss    : num [1:10] 93.1 0 141.7 73.5 128.2 ...
##  $ tot.withinss: num 760
##  $ betweenss   : num 2102
##  $ size        : int [1:10] 5 1 176 43 97 67 10 2 29 10
##  $ iter        : int 4
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
#wholesale_k10
```

We can gather the following information from the output above (K-means cluster analysis, n.d.):

1. Cluster: A vector of integers that indicates the cluster each point was allocated.
2. Centers: Matrix of cluster centers.
3. totss: Total sum of squares.
4. withinss: Vector of within-cluster sum of squares (one component per cluster).
5. tot.withinss: Sum of withinss (between-cluster sum of squares).
6. betweenss: Between-cluster sum of squares (difference between #3 and #5).
7. Size: Number of points in each cluster.

Now, we can evaluate our model.

Evaluate Model

First, we can print the size of our clusters and the coordinates of the cluster centroids.

```
wholesale_k10$size # size of each cluster
```

```
##  [1]   5   1 176  43  97  67  10   2  29  10
```

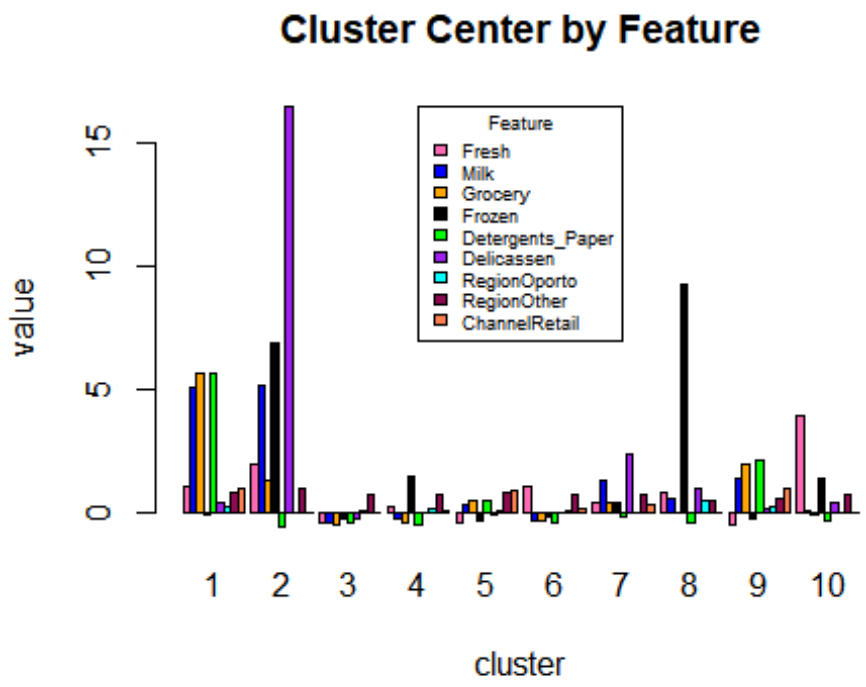
```
wholesale_k10$centers # coordinates of the cluster centroids
```

```
##         Fresh        Milk     Grocery      Frozen Detergents_Paper
Delicassen
## 1   1.0755395  5.10330749  5.63190631 -0.08979632        5.6823687
0.41981740
## 2   1.9645810  5.16961846  1.28575327  6.89275382       -0.5542311
16.45971129
## 3  -0.4393518 -0.44710991 -0.50638990 -0.28130813       -0.4557556 -
0.23474940
## 4   0.2370851 -0.29383207 -0.42497950  1.43457516       -0.4926072 -
```

```
0.01948986
## 5  -0.4437186  0.27909926  0.45038299 -0.37079244        0.4737530 -
0.08204866
## 6   1.0785217 -0.33820181 -0.37101329 -0.16869860       -0.4255396 -
0.03959479
## 7   0.3942257  1.31217333  0.40132141  0.43452320       -0.1520376
2.36495710
## 8   0.7918828  0.56104641 -0.01128859  9.24203651       -0.4635194
0.93210312
## 9  -0.5025939  1.41676382  1.92952805 -0.27734401        2.1311379
0.17668396
## 10  3.9617933  0.08968838 -0.08227548  1.38630711       -0.3260991
0.35691196
##    RegionOporto RegionOther ChannelRetail
## 1    0.20000000   0.8000000    1.00000000
## 2    0.00000000   1.0000000    0.00000000
## 3    0.09659091   0.7102273    0.01704545
## 4    0.13953488   0.6976744    0.06976744
## 5    0.09278351   0.7938144    0.91752577
## 6    0.10447761   0.7164179    0.14925373
## 7    0.00000000   0.7000000    0.30000000
## 8    0.50000000   0.5000000    0.00000000
## 9    0.20689655   0.5517241    1.00000000
## 10   0.00000000   0.7000000    0.00000000
```

We can further inspect the center of each cluster using a barplot (Yu-Wei, 2015).

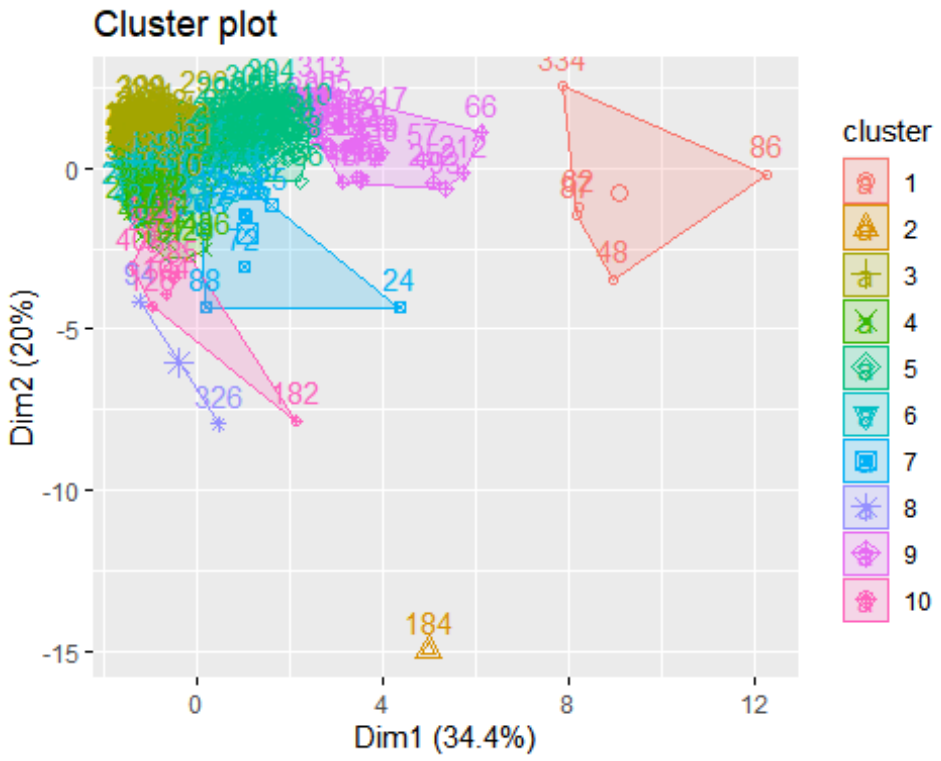
```
barplot(t(wholesale_k10$centers), beside = TRUE,
        xlab = "cluster", ylab="value",
        col = c("hotpink", "blue", "orange", "black", "green", "purple",
"cyan", "deeppink4", "coral"),
        main = "Cluster Center by Feature",
        legend = c(colnames(wholesale_k10$centers)),
        args.legend = list(title = "Feature", x = "top", cex = 0.55))
```




```
# (Yu-Wei, 2015): inspect center of each cluster
# (How can I add features or dimensions to my bar plot?, n.d.)
```

Now, we can view our cluster results using the **fviz_cluster()** function, which will perform principal componenet analysis (PCA) if there are more than two dimensions (variables) and plot accordingly (K-means cluster analysis, n.d.).

```
fviz_cluster(wholesale_k10, wholesale_final)
```

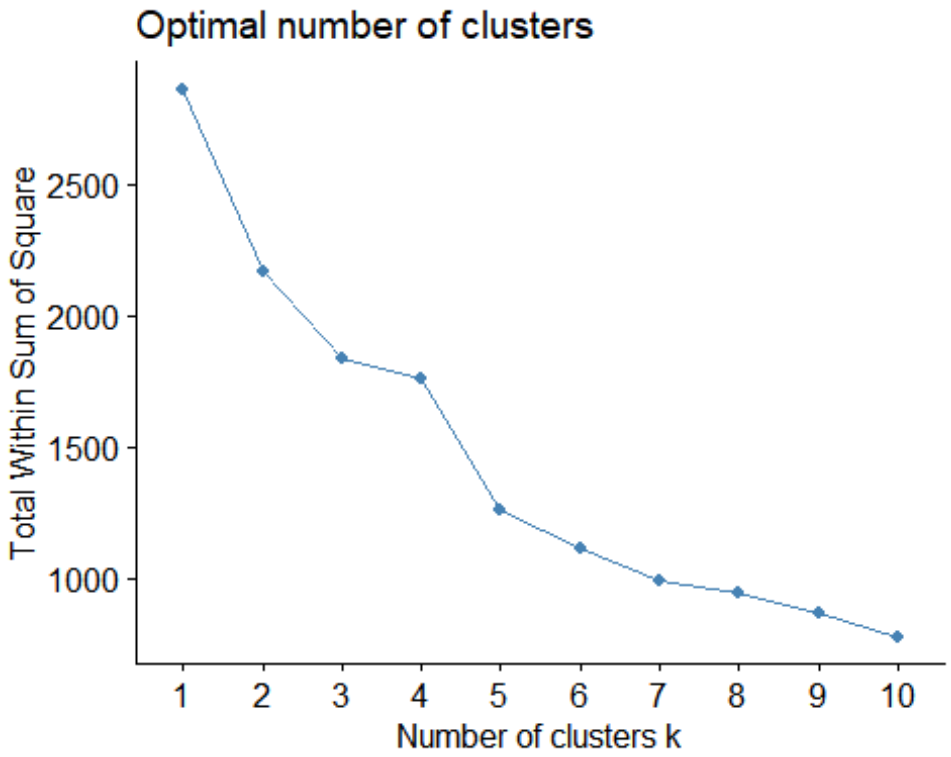


This plot suggests that there may be too many clusters for this data set because of the noise in the top left corner of our plot.

Determine Optimal Clusters

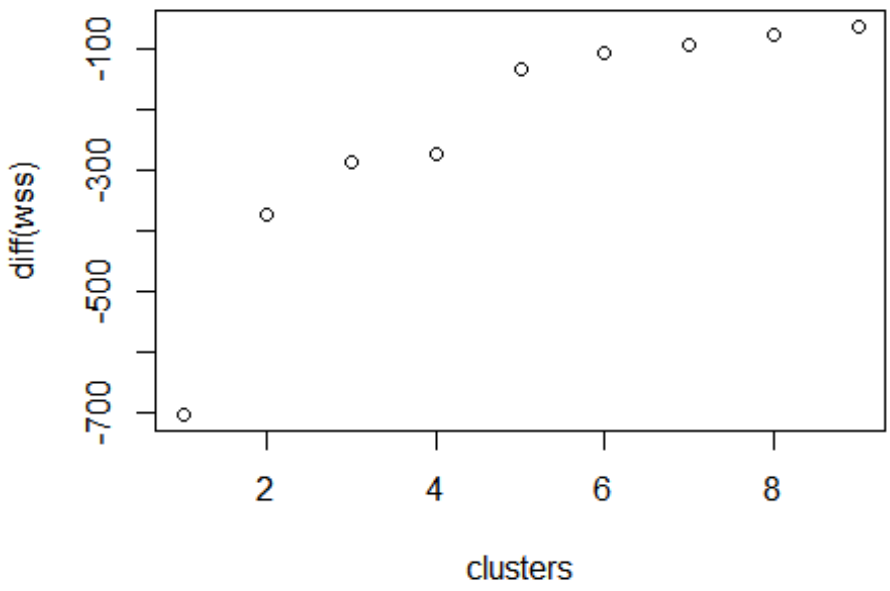
To determine the optimal number of clusters, there are various methods we can use. In particular, we will be using the Elbow method, the Silhouette method, and the Gap Statistic method. First, we will use the Elbow method. Basically, the idea is to define clusters such that the total intra-cluster variation (total within-cluster sum of square (wss)) is minimized (K-means cluster analysis, n.d.). The total within-cluster sum of square measures the compactness of the clustering. We want this value to be as small as possible. In an elbow plot, this value is where there appears to be a bend in the plot (K-means cluster analysis, n.d.). We can simply create this plot usint the **fviz_nbclust()** function and define the method as **wss**.

```
fviz_nbclust(wholesale_final, kmeans, method = "wss")
```



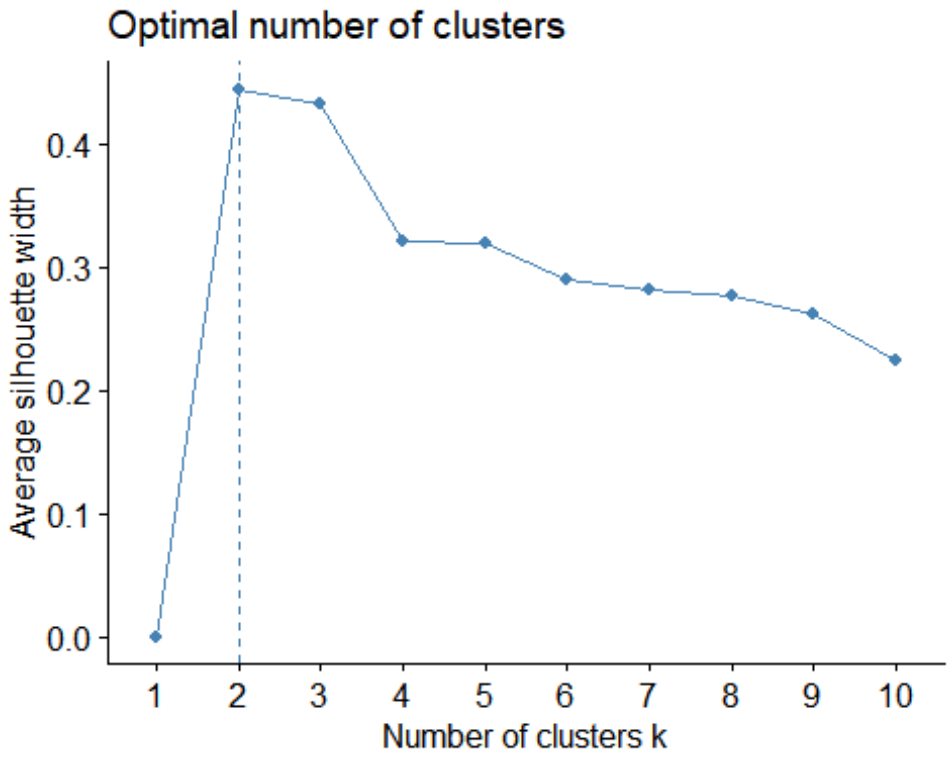
In the plot above, the "elbow" appears to occur at $k = 3$. To confirm this, we can compute the total within-cluster sum of square the *k* values 1 through ten. Then, we need to use the **unlist()** function to convert our list of wss values to a vector (R - convert named list to vector with values only, n.d.) and then we compute the difference (**diff()**) between each consecutive pair of elements in a vector (Calculate the difference, 2020). Then, from an assignment in my MSDS 682: Text Analytics Course, we can plot the **diff(wss)** values and where the plot "flattens" will indicate the best number of clusters.

```
wss <- function(k) {
  kmeans(wholesale_final, k, nstart = 35)$tot.withinss
}
k_vals <- 1:10
wss_vals <- lapply(k_vals, wss)
wss <- unlist(wss_vals, use.names = FALSE)
plot(diff(wss), xlab = "clusters")
```



From this plot, an argument can be made for *k* values 2 and 5. To further aid our decision for the optimal number of clusters, we will use the Silhouette method. This method measures the quality of clustering by determining how well each object lies within its cluster (K-means cluster analysis, n.d.). A high Silhouette value indicates "good" clustering. For this plot, we will use the argument **method = "silhouette"**.

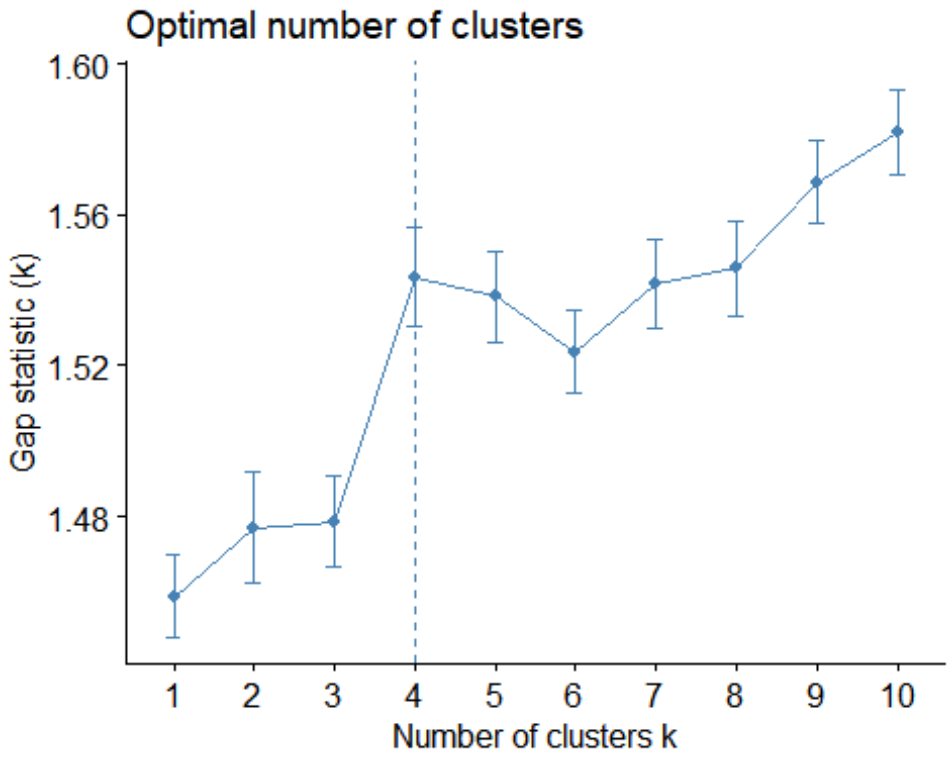
```
fviz_nbclust(wholesale_final, kmeans, method = "silhouette")
```



This plot also suggests that the number of optimal clusters is 2.

Finally, we will use the Gap Statistic method to determine the optimal number of clusters. "The gap statistic compares the total intra-cluster variation for different values of *k* with their expected values under null reference distribution of the data (i.e. a distribution with no obvious clustering) (K-means cluster analysis, n.d.)." To create this plot, we will use the argument **method = "gap"**.

```
fviz_nbclust(wholesale_final, kmeans, method = "gap")
```



This output suggests that the optimal number of clusters is 4. Now, since the majority of our plots indicate the optimal number of clusters is 2, we will use 2 for our *k* value in our final model.

Final Model

We can now create our final k-means model using a *k* value of 2.

```
wholesale_k2 <- kmeans(wholesale_final, 2, nstart = 25)
str(wholesale_k2)
```

```
## List of 9
##  $ cluster     : Named int [1:440] 1 1 1 1 1 1 1 1 1 2 ...
##   ..- attr(*, "names")= chr [1:440] "1" "2" "3" "4" ...
##  $ centers     : num [1:2, 1:9] 0.0091 -0.0779 -0.2234 1.9137 -0.2521 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:2] "1" "2"
##   .. ..$ : chr [1:9] "Fresh" "Milk" "Grocery" "Frozen" ...
##  $ totss       : num 2861
##  $ withinss    : num [1:2] 1283 874
##  $ tot.withinss: num 2157
##  $ betweenss   : num 704
##  $ size        : int [1:2] 394 46
##  $ iter        : int 1
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

Now, we will plot our clusters.

```
wholesale_k2$size

## [1] 394  46

wholesale_k2$centers

##          Fresh      Milk    Grocery       Frozen Detergents_Paper
Delicassen
## 1  0.009095778 -0.223432 -0.2521029  0.003248669       -0.2488329 -
0.07617343
## 2 -0.077907318  1.913743  2.1593160 -0.027825558        2.1313083
0.65244203
##   RegionOporto RegionOther ChannelRetail
## 1   0.09898477   0.7233503     0.2487310
## 2   0.17391304   0.6739130     0.9565217

fviz_cluster(wholesale_k2, wholesale_final)
```

A *k* value of 2 seems to better represent our data in clusters. We can extract these clusters and add to our initial data to view some descriptive statistics at the cluster level (K-means cluster analysis, n.d.).

```
wholesale_copy <- wholesale_df

wholesale_copy %>%
  mutate(Cluster = wholesale_k2$cluster) %>%
```

```
  group_by(Cluster) %>%
  summarise_all("mean")
```

```
## # A tibble: 2 x 9
##   Cluster Channel Region  Fresh   Milk Grocery Frozen Detergents_Paper
##     <int>   <dbl>  <dbl>  <dbl>  <dbl>   <dbl>  <dbl>            <dbl>
## 1       1      NA     NA 12115.  4147.   5556.  3088.            1695.
## 2       2      NA     NA 11015. 19920.  28472.  2937.           13043.
## # ... with 1 more variable: Delicassen <dbl>
```

Interpret Clusters

First, we will compare the **Region** and the **Channel** labels with the clustering result.

```
table(wholesale_df$Region, wholesale_k2$cluster) #compare region with
clustering results
```

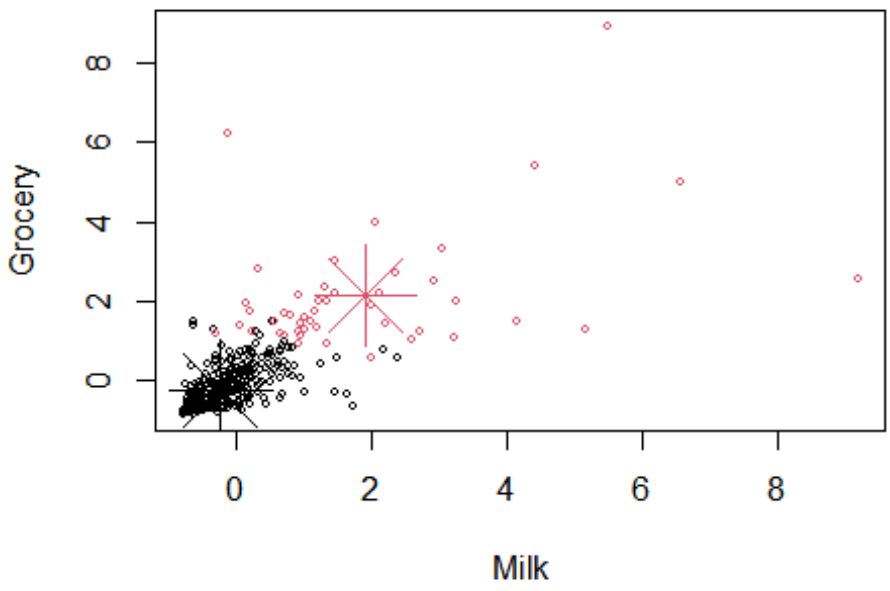
```
##
##            1   2
##   Lisbon  70   7
##   Oporto  39   8
##   Other  285  31
```

```
table(wholesale_df$Channel, wholesale_k2$cluster) #compare channel with
clustering results
```

```
##
##            1   2
##   Horeca 296   2
##   Retail  98  44
```

The output above suggests that the majority of our **Region** and **Channel** data was placed in the first cluster (K-means clustering, n.d.). Now, we can plot clusters and their centroids. Below, we will plot the clusters for **Milk** and **Grocery** and mark their cluster centers (K-means clustering, n.d.).

```
plot(wholesale_final[c("Milk", "Grocery")], col = wholesale_k2$cluster,pch=1,
cex=0.5)
points(wholesale_k2$centers[, c("Milk", "Grocery")], col=1:3,pch=8, cex=5)
```



We can also "get" the elements for each cluster by using the commands below (How to get the elements from a k means cluster in R, n.d.):

```
cluster_1 <- wholesale_df[wholesale_k2$cluster == 1,]
dim(cluster_1)

## [1] 394   8

cluster_2 <- wholesale_df[wholesale_k2$cluster == 2,]
dim(cluster_2)

## [1] 46  8
```

Notice, 394 of our 440 total observations were places in our first cluster, which we can confirm from the **size** output above. Now, we can view a summary of each cluster.

```
summary(cluster_1)

##    Channel            Region            Fresh            Milk
##  Length:394         Length:394         Min.   :    3   Min.   :   55
##  Class :character   Class :character   1st Qu.: 3324   1st Qu.: 1376
##  Mode  :character   Mode  :character   Median : 8829   Median : 3199
##                                        Mean   :12115   Mean   : 4147
##                                        3rd Qu.:17136   3rd Qu.: 6108
##                                        Max.   :76237   Max.   :23527
##     Grocery          Frozen        Detergents_Paper    Delicassen
##  Min.   :    3   Min.   :   25.0   Min.   :    3.0   Min.   :    3.0
##  1st Qu.: 2050   1st Qu.:  738.8   1st Qu.:  236.2   1st Qu.:  395.0
```

```
##  Median : 3836   Median : 1567.0   Median :  629.5   Median :  914.5
##  Mean   : 5556   Mean   : 3087.7   Mean   : 1695.1   Mean   : 1310.1
##  3rd Qu.: 8118   3rd Qu.: 3686.0   3rd Qu.: 2584.0   3rd Qu.: 1752.2
##  Max.   :22272   Max.   :60869.0   Max.   :10069.0   Max.   :14472.0
```

```
summary(cluster_2)
```

```
##    Channel            Region             Fresh             Milk
##  Length:46          Length:46          Min.   :    85   Min.   : 3737
##  Class :character   Class :character   1st Qu.:  2026   1st Qu.:11946
##  Mode  :character   Mode  :character   Median :  5407   Median :15184
##                                        Mean   : 11015   Mean   :19920
##                                        3rd Qu.: 11895   3rd Qu.:24587
##                                        Max.   :112151   Max.   :73498
##     Grocery          Frozen        Detergents_Paper   Delicassen
##  Min.   :13567   Min.   :   33.0   Min.   :  239    Min.   :    3.0
##  1st Qu.:19936   1st Qu.:  806.2   1st Qu.: 8063    1st Qu.:  719.8
##  Median :23797   Median : 1365.0   Median :11908    Median : 1437.5
##  Mean   :28472   Mean   : 2936.8   Mean   :13043    Mean   : 3364.8
##  3rd Qu.:29929   3rd Qu.: 2794.5   3rd Qu.:15312    3rd Qu.: 2925.5
##  Max.   :92780   Max.   :36534.0   Max.   :40827    Max.   :47943.0
```

The elements in our first cluster appear to be lower annual spending observations and the second cluster contains the higher annual spending observations. Recall from the histograms we created during our data exploration section, most of our observations were lower annual spending values.

Part 2: Use Hierarchical Clustering

Now, we will use Hierarchical clustering on our data, **wholesale_final**. Hierarchical clustering creates a hierarchy of clusters and presents the hierarchies in a dendrogram (Yu-Wei, 2015). Instead of specifying the number of clusters prior to building the clustering algorithm, this method does not require the number of clusters to be specified at the beginning (Yu-Wei, 2015).

Hierarchical clustering can be divided into two main types of clustering: agglomerative and divsive (Hierarchical cluster analysis, n.d.).

1. Agglomerative clustering (AGNES): Works in a bottom-up manner. Each object is initially considered as a single element cluster (leaf) (Hierarchical cluster analysis, n.d.). At each step of the algorithm, the two most similar clusters are combined into a larger cluster (nodes). This process is continued until all points are members of just on single larger cluster (root) (Hierarchical cluster analysis, n.d.). Agglomerative cluster is best at identifying small clusters.
2. Divisive clustering (DIANA): Works in a top-down manner (Hierarchical cluster analysis, n.d.). It works in an inverse order of agglomerative clustering. Instead of beginning with single elements, divisive clustering begins with the one large cluster (root) (Hierarchical cluster analysis, n.d.). At each step of the iteration, the most

heterogeneous cluster is divided into two (nodes). The process is complete when all objects are in their own cluster (leaf) (Hierarchical cluster analysis, n.d.).

Regardless of the chosen approach, both first use a distance similarity measure to combine or split clusters (Yu-Wei, 2015). Recall, to measure the similarity/dissimilarity between observations, the Euclidean distance, Manhattan distance, Pearson Correlation distance, or the Spearman Correlation distance can be used. To measure the dissimilarity between two cluster of observations, we can use a variety of different linkage methods (Hierarchical cluster analysis, n.d.). The most popular types include (World Class FTE Week 6):

1. Single Linkage: Also known as nearest neighbor clustering, finds the smallest distance between points in different clusters. Makes the assumption that two groups are close if two points are close.
2. Complete Linkage: Also known as farthest neighbor clustering, finds the largest (maximum) distance between points in different clusters.
3. Average Linkage: Finds the average distance between all pairs of data objects in different groups. This is a compromise between single and complete linkages.
4. Centroid linkage: This is the distance between the means (centroids) of the clusters.
5. Ward's Variance method: Minimizes the total within-cluster variance. At each step the pair of clusters with minimum between-cluster distance are merged (Hierarchical cluster analysis, n.d.).

First, we will perform agglomerative hierarchical clustering.

Agglomerative Clustering

To perform agglomerative hierarchical clustering, we will use the **hclust()** function, but the **agnes()** function could also be used (Hierarchical cluster analysis, n.d.). First, we need to compute the dissimilarity values using the **dist()** function and use these values in the **hclust()** function.

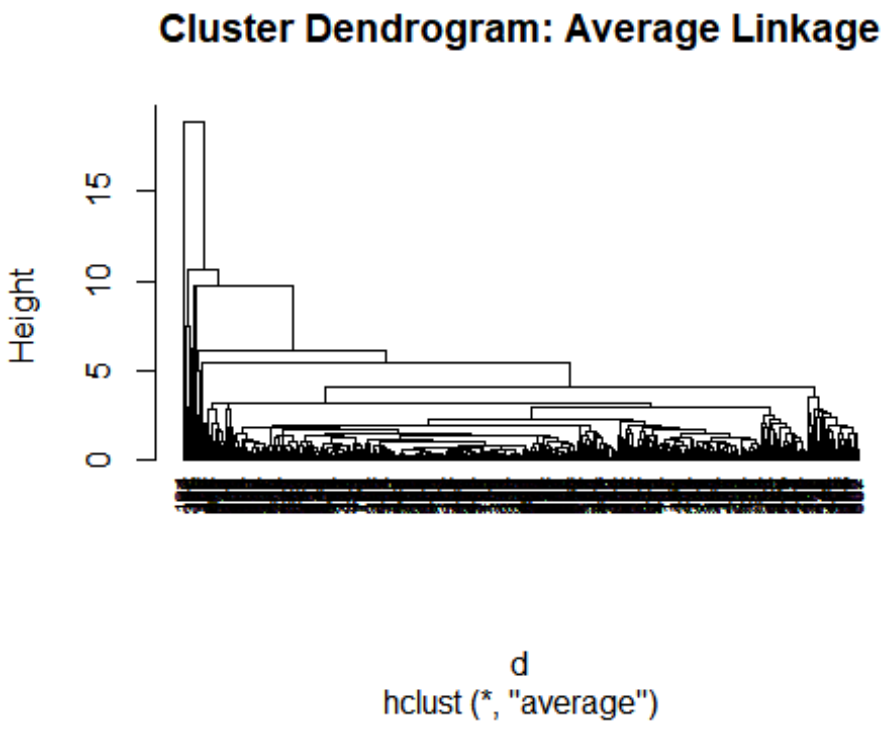
```
d <- dist(wholesale_final, method = "euclidean") #compute dissimilarity
values with dist()
```

We will specify the agglomeration method in our first agglomerative hierarchical clutering model to be "average", which tells the mdoel to use the average linkage. Then, we will plot the dendrogram. "A dendrogram is a diagram that shows the hierarchical relationship between objects (What is a dendrogram?, 2018)."

```
hc1 = hclust(d, method = "average") #average linkage
hc1

##
## Call:
## hclust(d = d, method = "average")
##
## Cluster method   : average
## Distance         : euclidean
## Number of objects: 440
```

```r
plot(hc1, cex = 0.6, hang = -1, main = "Cluster Dendrogram: Average Linkage")
#plot dendrogram
```



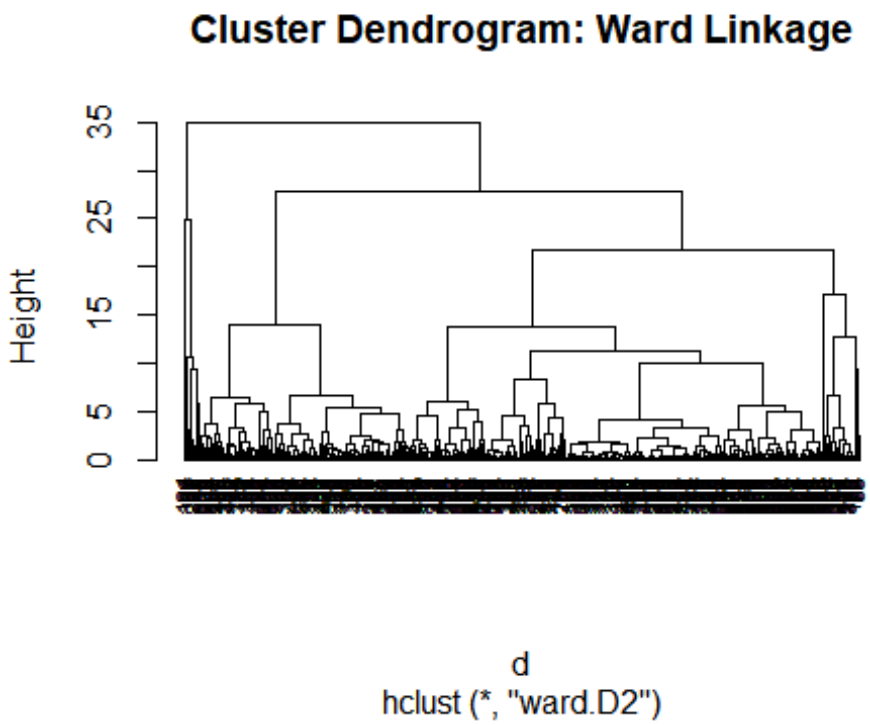

```r
#hang = -1 puts labels at the same height
```

Now, we will use the Ward linkage.

```r
hc2 = hclust(d, method = "ward.D2") #Ward linkage
hc2

##
## Call:
## hclust(d = d, method = "ward.D2")
##
## Cluster method   : ward.D2
## Distance         : euclidean
## Number of objects: 440

plot(hc2, cex = 0.6, hang = -1, main = "Cluster Dendrogram: Ward Linkage")
#plot dendrogram
```




```
#hang = -1 puts labels at the same height
```

Alternatively, we can use the **agnes()** function to identify the strongest clustering structure for this data set. The function below was used from Hierarchical Cluster Analysis (n.d.) to demonstrate how to determine the strongest clustering structure.

```
#methods
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")

ac <- function(x) {
  agnes(wholesale_final, method = x)$ac
}

lapply(m, ac)

## $average
## [1] 0.9608405
##
## $single
## [1] 0.9506065
##
## $complete
## [1] 0.9646056
##
## $ward
## [1] 0.977974
```

The output above suggests that the Ward's method identifies the strongest clustering methods.

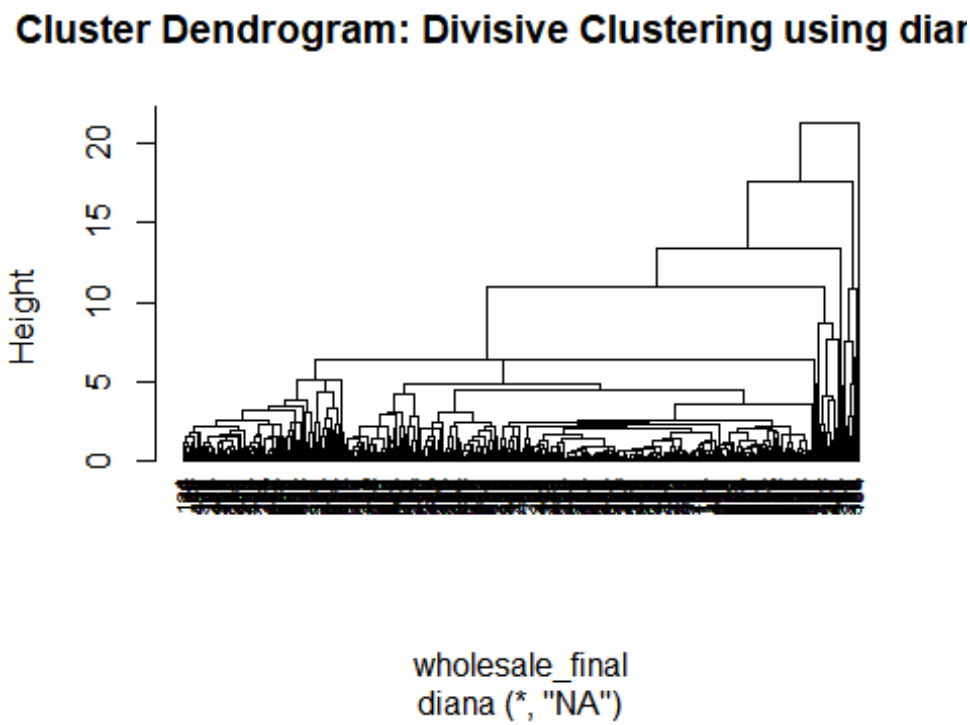
Divisive Clustering

Next, we will use divisive clustering.

```
hc3 <- diana(wholesale_final)
hc3$dc

## [1] 0.9610709

pltree(hc3, cex = 0.6, hang = -1, main = "Cluster Dendrogram: Divisive
Clustering using diana()")
```



Notice, the divisive coefficient is not as strong as our agglomerative coefficient for Ward's method.

Cutting Dendrogram

Notice that in each dendrogram above, the leaves correspond to observations and as we move up the tree, the observations that are similar are combined into branches, which are fused at a higher height (Hierarchical cluster analysis, n.d.). The dissimilarity between two observations can be represented by the height of the fusion. The higher the fusion, the less similar the observations (Hierarchical cluster analysis, n.d.). The height of the cut to the dendrogram controls the number of clusters obtained (Hierarchical cluster analysis, n.d.). To identify these clusters, we can cut the dendrogram using the **cutree()** function. We will

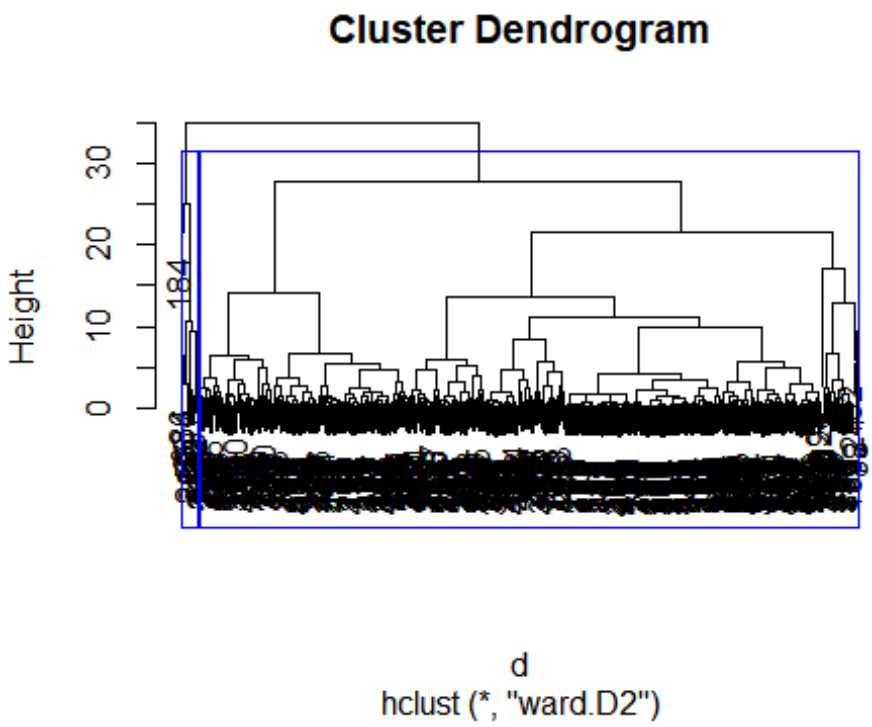
cut our Ward's dendrogram from above. We will use $k = 2$ as this was the optimal number of clusters we found above.

```
fit_ward = cutree(hc2, k = 2)
table(fit_ward)

## fit_ward
##   1   2
## 429  11
```

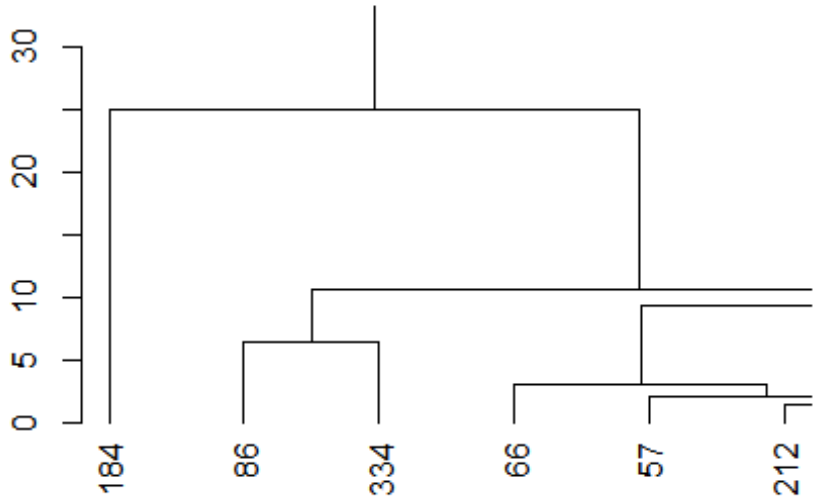
Now, we can draw a border around the two clusters.

```
plot(hc2)
rect.hclust(hc2, k=2, border = "blue")
```



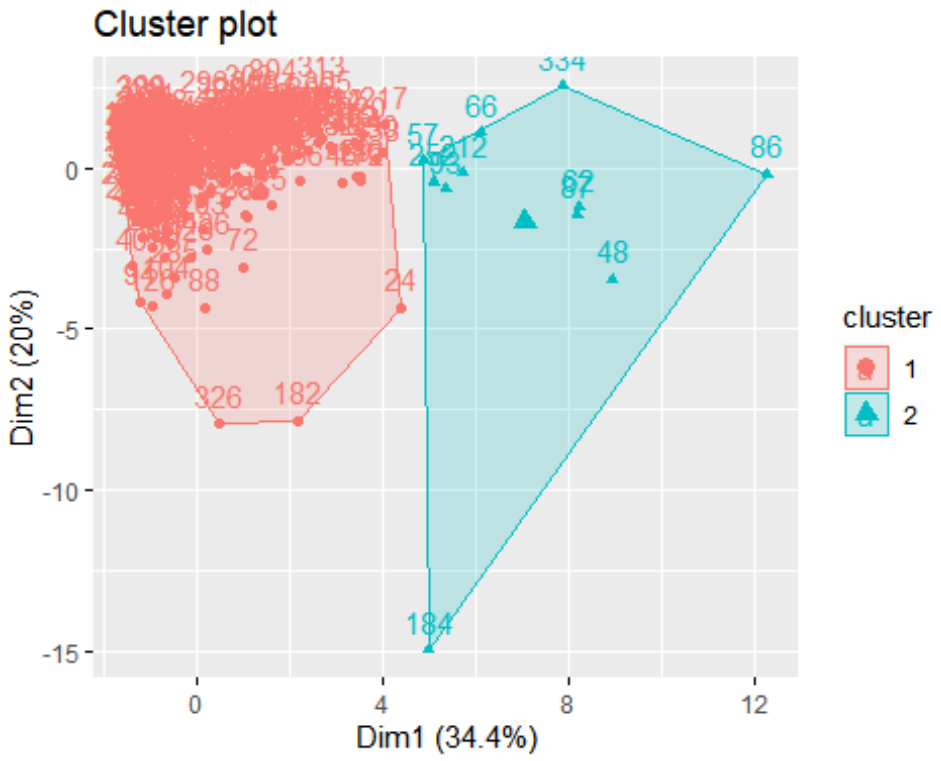
We can zoom in to our first cluster using the following command (Beautiful dendrogram visualizations in r, n.d.):

```
hcd_2 <- as.dendrogram(hc2)
plot(hcd_2, xlim = c(1,6), ylim = c(1,32)) #zoom into first cluster
```



Further, we can visualize these clusters by using the **fviz_cluster()** function (Hierarchical cluster analysis, n.d.).

```
fviz_cluster(list(data = wholesale_final, cluster = fit_ward))
```



Now, let's use $k = 10$ in our **cutree()** function.

```
fit_2 = cutree(hc2, k = 10)
table(fit_2)

## fit_2
##   1   2   3   4   5   6   7   8   9  10
##  89  52  49  48 166   4  19  10   2   1
```

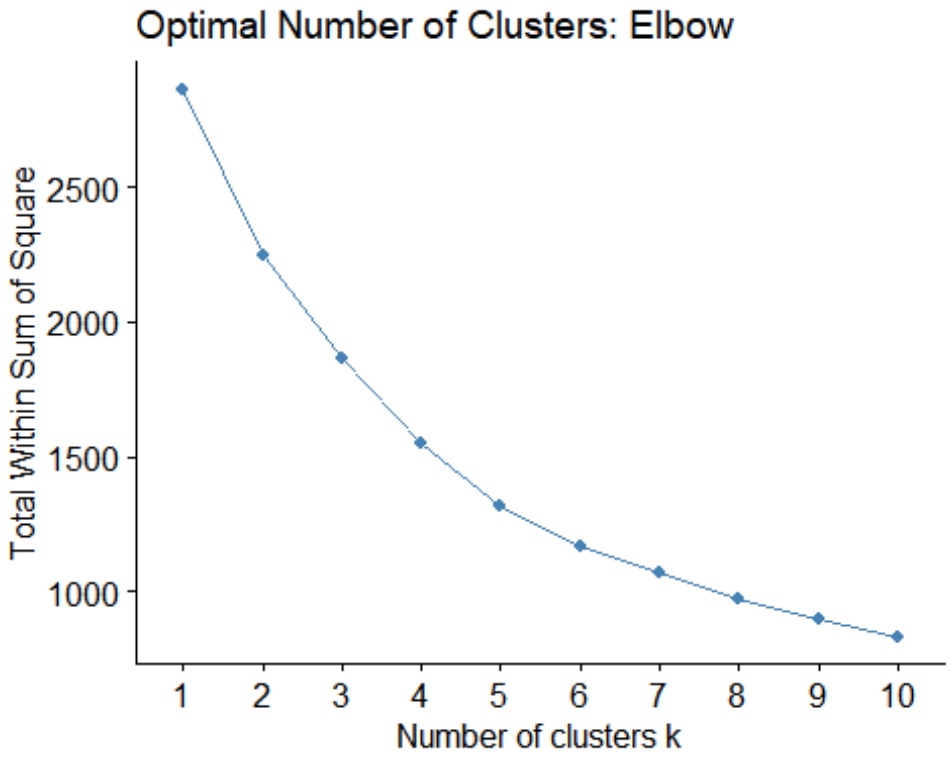
To view this cut, we will use the **as.phylo()** function to produce a more sophisticated plot (Beautiful dendrogram visualizations in r, n.d.).

```
colors = c("pink", "blue", "orange", "black", "green", "purple", "cyan",
"deeppink4", "coral", "darkolivegreen")
plot(as.phylo(hc2), type = "fan", tip.color = colors[fit_2], label.offset =
0.5, cex = 0.4, show.tip.label = TRUE)
```

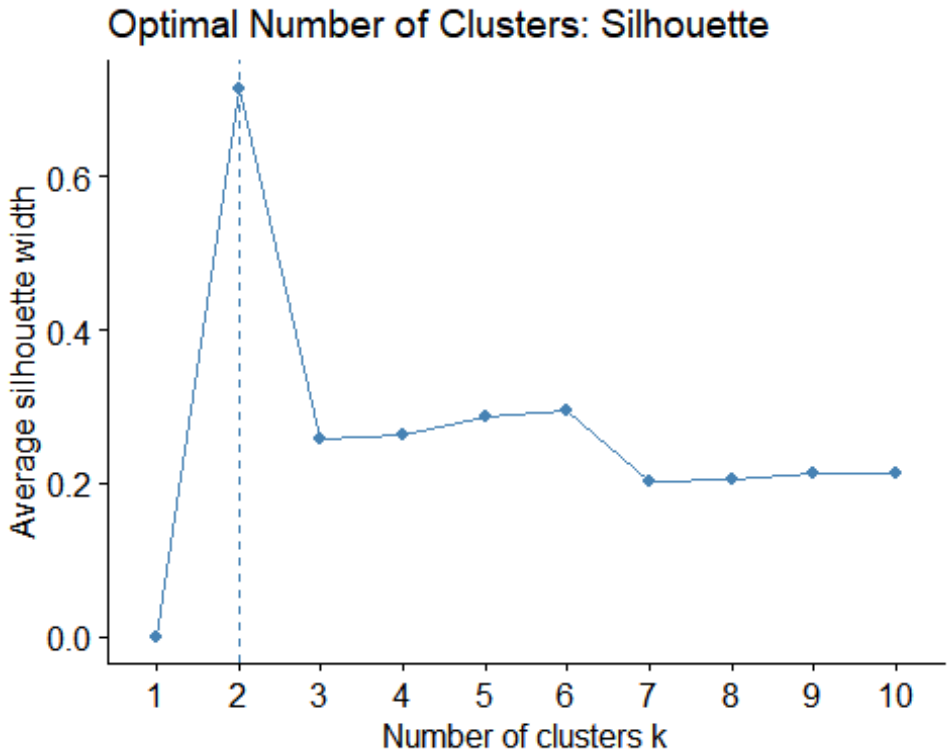
Determine Optimal Clusters

Finally, we can determine the optimal number of clusters using similar methods for k-means clustering.

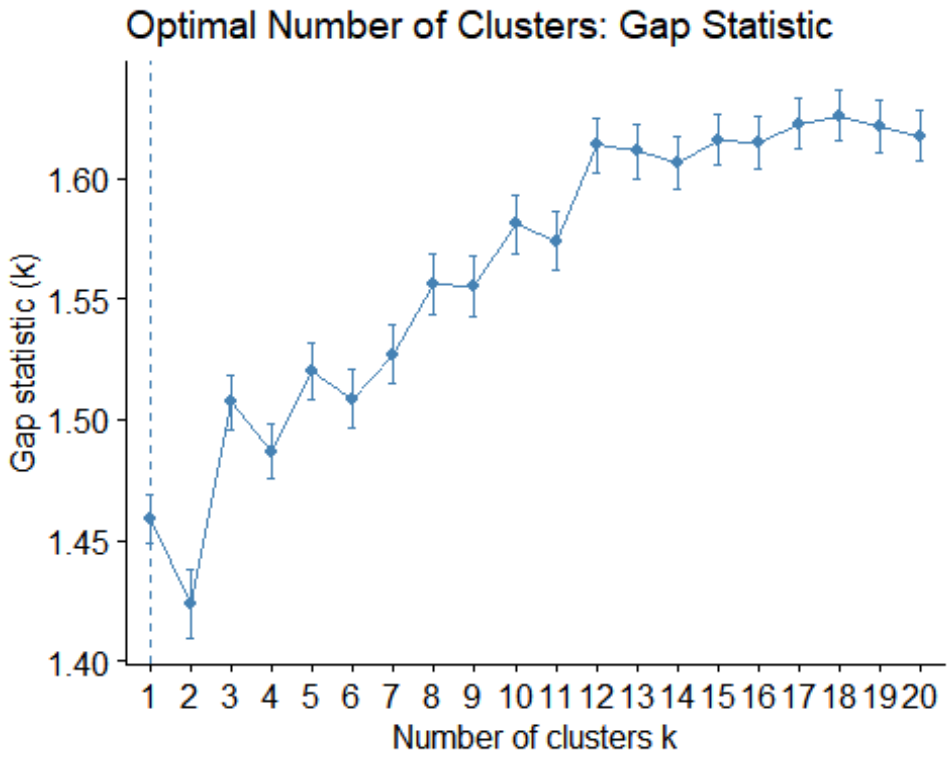
```
fviz_nbclust(wholesale_final, FUN = hcut, method = "wss")+labs(title
="Optimal Number of Clusters: Elbow")
```




```r
fviz_nbclust(wholesale_final, FUN = hcut, method = "silhouette")+labs(title
="Optimal Number of Clusters: Silhouette")
```



```
fviz_nbclust(wholesale_final, FUN = hcut, method = "gap", k.max =
20)+labs(title ="Optimal Number of Clusters: Gap Statistic")
```



From these outputs, the optimal number of cluster would be 2 according to the Silhouette and Elbow plot.

Conclusion

For this assignment, we applied two different clustering algorithms, k-means and hierarchical, to a data set that contained information regarding clients of a wholesale distributor and included their annual spending. After loading our data, we performed data exploration and prepared our data for our clustering models. This included converted our discrete values to dummy variables and scaling our continuous variables. After preparing the data, we began by applying the k-means clustering algorithm. Using our initial model, we used various methods to determine the optimal number of clusters. Using this value, we created our final k-means clustering model and created different plots and analyzed the contents of each cluster. From here, we moved on to hierarchical clustering. We began by using different methods to build our clusters. To visualize these clusters, we created dendrograms and compared the results. Finally, using similar techniques to determine the optimal number of clusters for the k-means clustering, we found found that the optimal number of clusters for our hierarchical clustering algorithm.

Resources

Beautiful dendrogram visualizations in r. (n.d.). Retrieved October 5, 2020, from http://www.sthda.com/english/wiki/beautiful-dendrogram-visualizations-in-r-5-must-known-methods-unsupervised-machine-learning

Calculate the difference . (2020, May 27). GeeksforGeeks. https://www.geeksforgeeks.org/calculate-the-difference-between-consecutive-pair-of-elements-of-a-vector-in-r-programming-diff-function/

Hierarchical cluster analysis . (n.d.). Retrieved October 5, 2020, from http://uc-r.github.io/hc_clustering

How can I add features or dimensions to my bar plot? (n.d.). Retrieved October 5, 2020, from https://stats.idre.ucla.edu/r/faq/how-can-i-add-features-or-dimensions-to-my-bar-plot/

How to get the elements from a k means cluster in R. (n.d.). Stack Overflow. Retrieved October 5, 2020, from https://stackoverflow.com/questions/21955571/how-to-get-the-elements-from-a-k-means-cluster-in-r

K-means cluster analysis. (n.d.). Retrieved October 5, 2020, from https://uc-r.github.io/kmeans_clustering

K-means clustering . (n.d.). Retrieved October 5, 2020, from http://www.rdatamining.com/examples/kmeans-clustering

Margarida , C. (2013). Uci machine learning repository: Wholesale customers data set. https://archive.ics.uci.edu/ml/datasets/Wholesale+customers

R - convert named list to vector with values only. (n.d.). Stack Overflow. Retrieved October 5, 2020, from https://stackoverflow.com/questions/17094774/convert-named-list-to-vector-with-values-only

R: scaling and centering of matrix-like objects. (n.d.). Retrieved October 5, 2020, from https://stat.ethz.ch/R-manual/R-devel/library/base/html/scale.html

What is a correlation matrix? (2018, August 16). Displayr. https://www.displayr.com/what-is-a-correlation-matrix/

What is a dendrogram? . (2018, March 21). Displayr. https://www.displayr.com/what-is-dendrogram/

World Class From the Expert: Week 6

Yu-Wei, C. (2015). Machine Learning with R Cookbook. Packt Publishing.