KMeans and DBSCAN Algorithims

2023-12-11

K-MEANS

Install packages and read in data

```
#install.packages('tidyverse') #only install once
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ dplyr     1.1.2     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.4.4     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
#install.packages('factoextra') #only install once
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

Reading the data

```
# read in data (change to your path)
clustering_input1 <- read_rds( "city.rds")
```

Removing city, region and province from the data

```
clustering_input2 <- clustering_input1 %>%
        select(-city, -region, -province )

str(clustering_input2)
```

```
## tibble [257 × 18] (S3: tbl_df/tbl/data.frame)
##  $ size                      : num [1:257] 762 1031 394 902 853 ...
##  $ promo_units_per           : num [1:257] 0.412 0.309 0.34 0.373 0.377 ...
##  $ high_med_gp               : num [1:257] 0.631 1 0 0.981 0.604 ...
##  $ velocityA_units_per       : num [1:257] 0.62 0.594 0.688 0.608 0.63 ...
##  $ velocityB_units_per       : num [1:257] 0.216 0.227 0.209 0.233 0.221 ...
##  $ velocityC_units_per       : num [1:257] 0.0729 0.0903 0.0381 0.0761 0.0683 ...
##  $ velocityD_units_per       : num [1:257] 0.0772 0.079 0.0571 0.0715 0.0673 ...
##  $ velocityNEW_units_per     : num [1:257] 0.00346 0.00217 0.0027 0.00219 0.0048 ...
##  $ energy_units_per          : num [1:257] 0.14 0.17 0.16 0.17 0.176 ...
##  $ regularBars_units_per     : num [1:257] 0.0948 0.0683 0.1067 0.0664 0.0884 ...
##  $ gum_units_per             : num [1:257] 0.05 0.0296 0.0703 0.0404 0.064 ...
##  $ bagpegCandy_units_per     : num [1:257] 0.0436 0.0292 0.0223 0.0433 0.0364 ...
##  $ isotonics_units_per       : num [1:257] 0.0603 0.0529 0.1019 0.0635 0.0592 ...
##  $ singleServePotato_units_per: num [1:257] 0.0323 0.0443 0.0325 0.0313 0.0347 ...
##  $ takeHomePotato_units_per  : num [1:257] 0.0227 0.0221 0 0.0244 0.02 ...
##  $ kingBars_units_per        : num [1:257] 0.0412 0.04 0.0735 0.0352 0.0396 ...
##  $ flatWater_units_per       : num [1:257] 0.0985 0.1116 0.0885 0.1041 0.0884 ...
##  $ psd591Ml_units_per        : num [1:257] 0.0347 0.0509 0.057 0.0537 0.0287 ...
```

```
summary(clustering_input2)
```

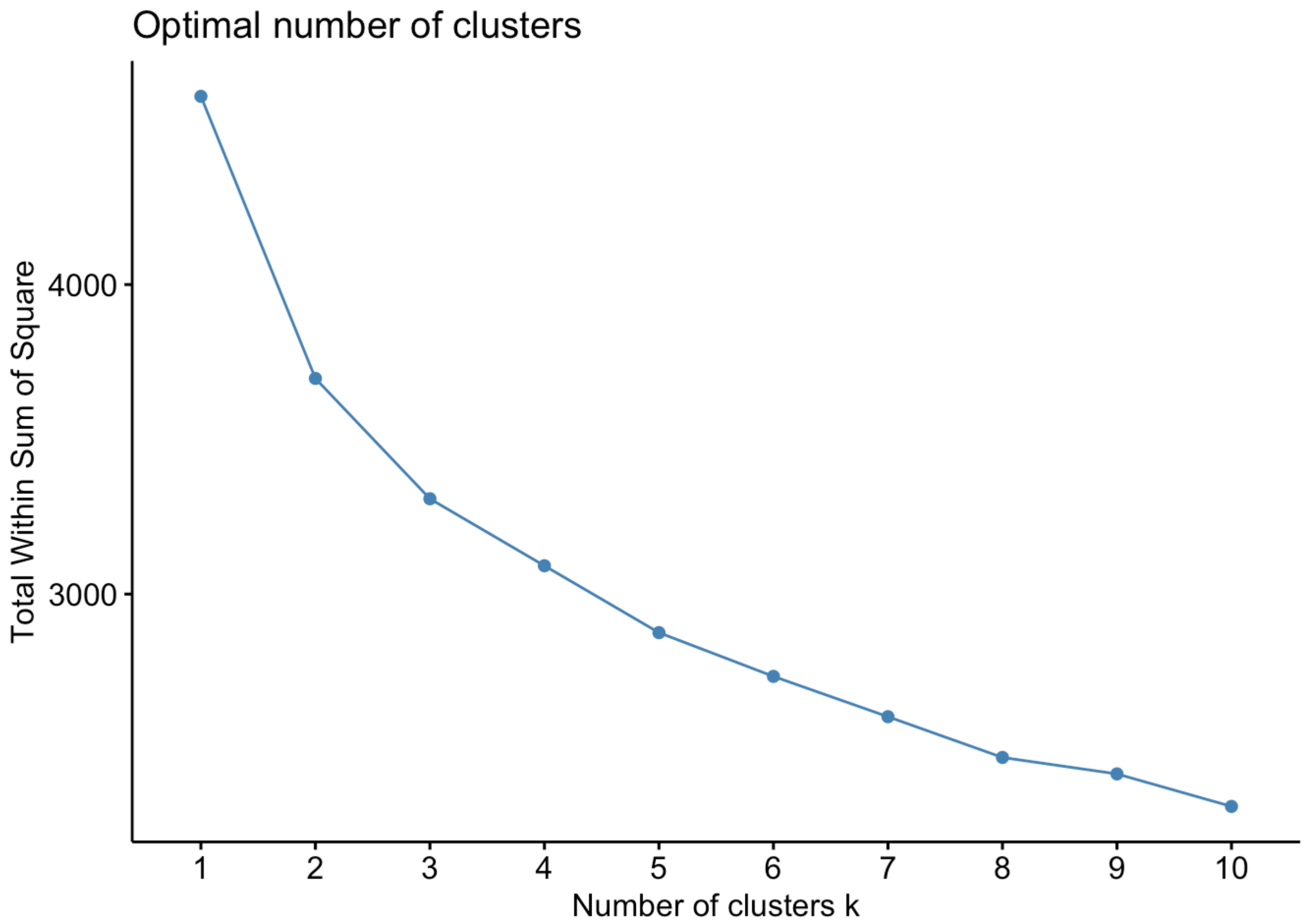
```
##       size        promo_units_per   high_med_gp     velocityA_units_per
##  Min.   : 297.0   Min.   :0.2198   Min.   :0.0000   Min.   :0.5129
##  1st Qu.: 679.2   1st Qu.:0.3355   1st Qu.:0.1154   1st Qu.:0.6189
##  Median : 782.5   Median :0.3688   Median :0.5504   Median :0.6378
##  Mean   : 758.0   Mean   :0.3678   Mean   :0.5399   Mean   :0.6412
##  3rd Qu.: 869.8   3rd Qu.:0.3954   3rd Qu.:0.9904   3rd Qu.:0.6638
##  Max.   :1119.0   Max.   :0.5541   Max.   :1.0000   Max.   :0.7470
##  velocityB_units_per velocityC_units_per velocityD_units_per
##  Min.   :0.1524      Min.   :0.02724     Min.   :0.03778
##  1st Qu.:0.1962      1st Qu.:0.06197     1st Qu.:0.06308
##  Median :0.2089      Median :0.06952     Median :0.06900
##  Mean   :0.2074      Mean   :0.06848     Mean   :0.07012
##  3rd Qu.:0.2189      3rd Qu.:0.07687     3rd Qu.:0.07685
##  Max.   :0.2491      Max.   :0.10014     Max.   :0.14839
##  velocityNEW_units_per energy_units_per  regularBars_units_per
##  Min.   :0.0009289     Min.   :0.08248   Min.   :0.04800
##  1st Qu.:0.0028859     1st Qu.:0.13755   1st Qu.:0.07187
##  Median :0.0039880     Median :0.16549   Median :0.08104
##  Mean   :0.0040320     Mean   :0.17033   Mean   :0.08247
##  3rd Qu.:0.0049742     3rd Qu.:0.19413   3rd Qu.:0.09102
##  Max.   :0.0148708     Max.   :0.31297   Max.   :0.16850
##  gum_units_per     bagpegCandy_units_per isotonics_units_per
##  Min.   :0.02019   Min.   :0.00000       Min.   :0.02492
##  1st Qu.:0.04470   1st Qu.:0.02524       1st Qu.:0.05490
##  Median :0.05877   Median :0.03011       Median :0.06035
##  Mean   :0.06004   Mean   :0.02991       Mean   :0.06072
##  3rd Qu.:0.07372   3rd Qu.:0.03520       3rd Qu.:0.06640
##  Max.   :0.15876   Max.   :0.05245       Max.   :0.10191
##  singleServePotato_units_per takeHomePotato_units_per kingBars_units_per
##  Min.   :0.01384             Min.   :0.00000          Min.   :0.01114
##  1st Qu.:0.03122             1st Qu.:0.01809          1st Qu.:0.03272
##  Median :0.03592             Median :0.02250          Median :0.03946
##  Mean   :0.03721             Mean   :0.02321          Mean   :0.03939
##  3rd Qu.:0.04157             3rd Qu.:0.02790          3rd Qu.:0.04494
##  Max.   :0.08420             Max.   :0.05575          Max.   :0.08355
##  flatWater_units_per psd591Ml_units_per
##  Min.   :0.04537     Min.   :0.01577
##  1st Qu.:0.08269     1st Qu.:0.03275
##  Median :0.09457     Median :0.04133
##  Mean   :0.09726     Mean   :0.04204
##  3rd Qu.:0.11114     3rd Qu.:0.05049
##  Max.   :0.16152     Max.   :0.08940
```

Z-score standardization

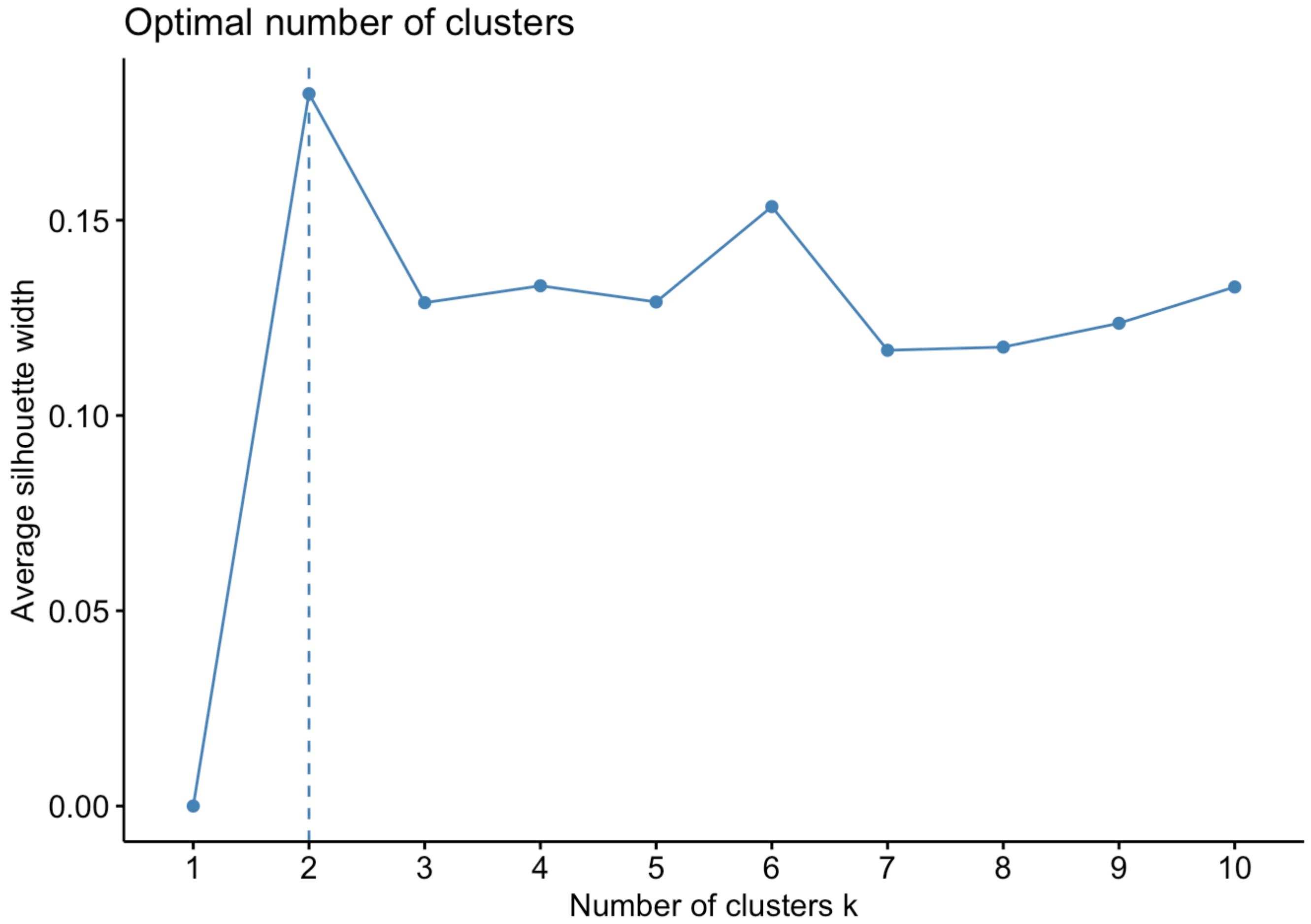
```
clustering_input2 <- as.data.frame(scale(clustering_input2))
```

Finding a suitable k using two different methods

```
# Within-cluster sum of square method
set.seed(42)
factoextra::fviz_nbclust(clustering_input2, kmeans, method = "wss")
```




```
# Silhouette approach
set.seed(42)
factoextra::fviz_nbclust(clustering_input2, kmeans, method = "silhouette")
```



Running the model

```
set.seed(42)
clusters <- kmeans(clustering_input2, centers=3, iter.max=10, nstart=10)
```

Checking the size of the k clusters

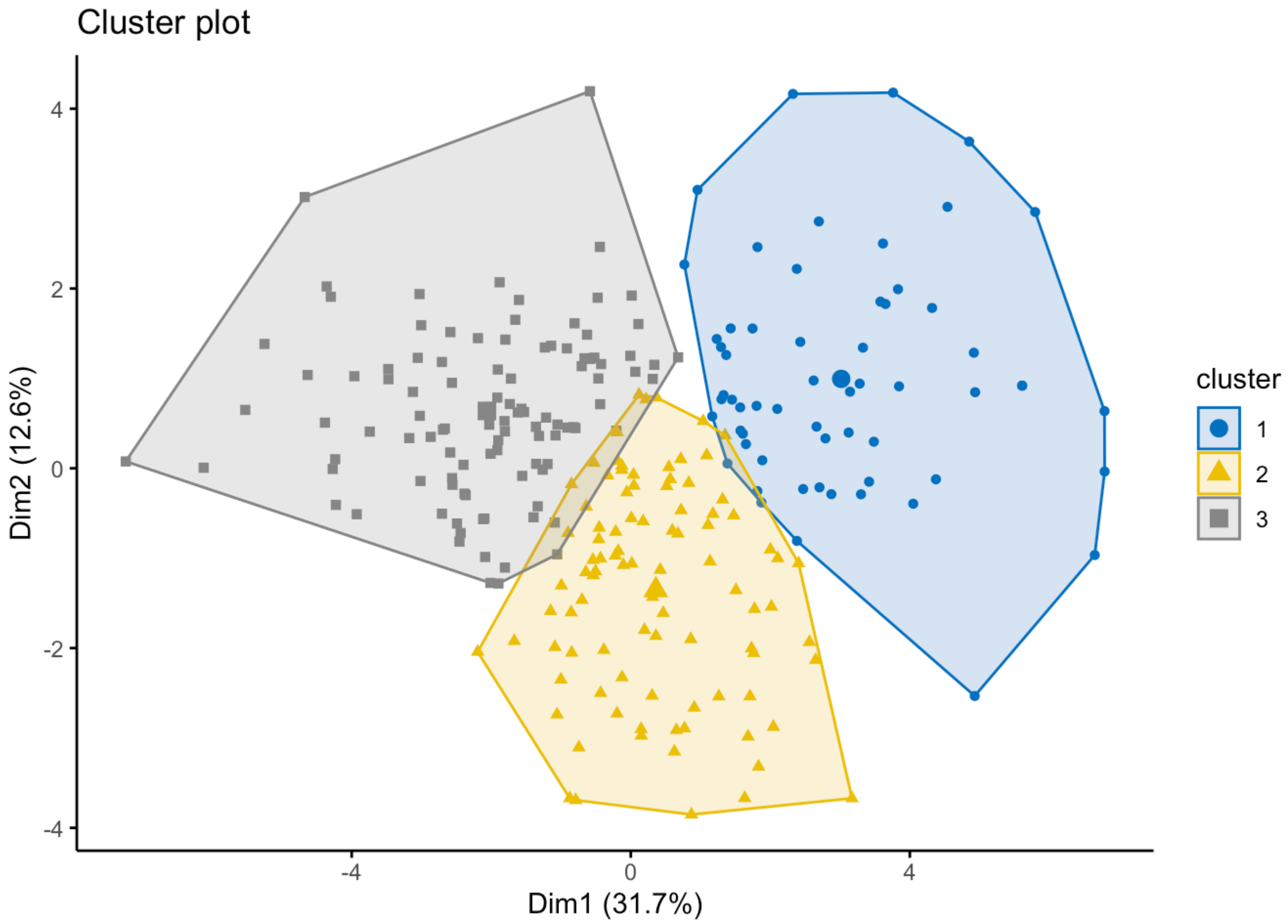
```
clusters$size
```

```
## [1]  60  93 104
```

Visualizing the clustering

(Uses principal components to collapse the dimensions of the data down to two dimensions)

```
fviz_cluster(clusters, clustering_input2,  geom = "point", show.clust.cent = TRUE, palette = "jco", ggtheme = the
me_classic())
```



Number of clustersfor K-Means Algorithm Based on Data

After a thorough analysis of the data using both the “wss” and “silhouette” methods for determining the optimal number of clusters in the K-means algorithm, we have decided to utilize 3 clusters. The “wss” method suggests that any score from 3 to 7 could be suitable. This suggests that 3 clusters provide a good balance between capturing the variance in the data and avoiding unnecessary complexity. Moreover, the silhouette method, which measures the quality of clustering by assessing cohesion within clusters and separation between clusters, also supports the choice of 3 clusters. The highest silhouette score was obtained when setting k closer to 2, but the third-highest score was achieved with k=3. When we visually inspected the clusters in a two-dimensional plot, the decision to use 3 clusters was further reinforced. Cluster 3 exhibited almost non-overlapping classes and effectively separated data points in space. Additionally, examining the size of clusters, we found that they were reasonably balanced, with 60, 93, and 104 data points within clusters 1, 2, and 3, respectively. This balanced distribution suggests that 3 clusters offer a representative segmentation of the data without skewing towards any specific subset. In summary, the combination of the wss and silhouette methods, visual inspection, and along with number of data pints within clusters led us to confidently choose 3 clusters for our K-means algorithm.

KMeans with 6 Centers

Running the model with 6 centers

```
set.seed(42)
clusters <- kmeans(clustering_input2, centers=6, iter.max=10, nstart=10)
```

Checking the size of the k clusters

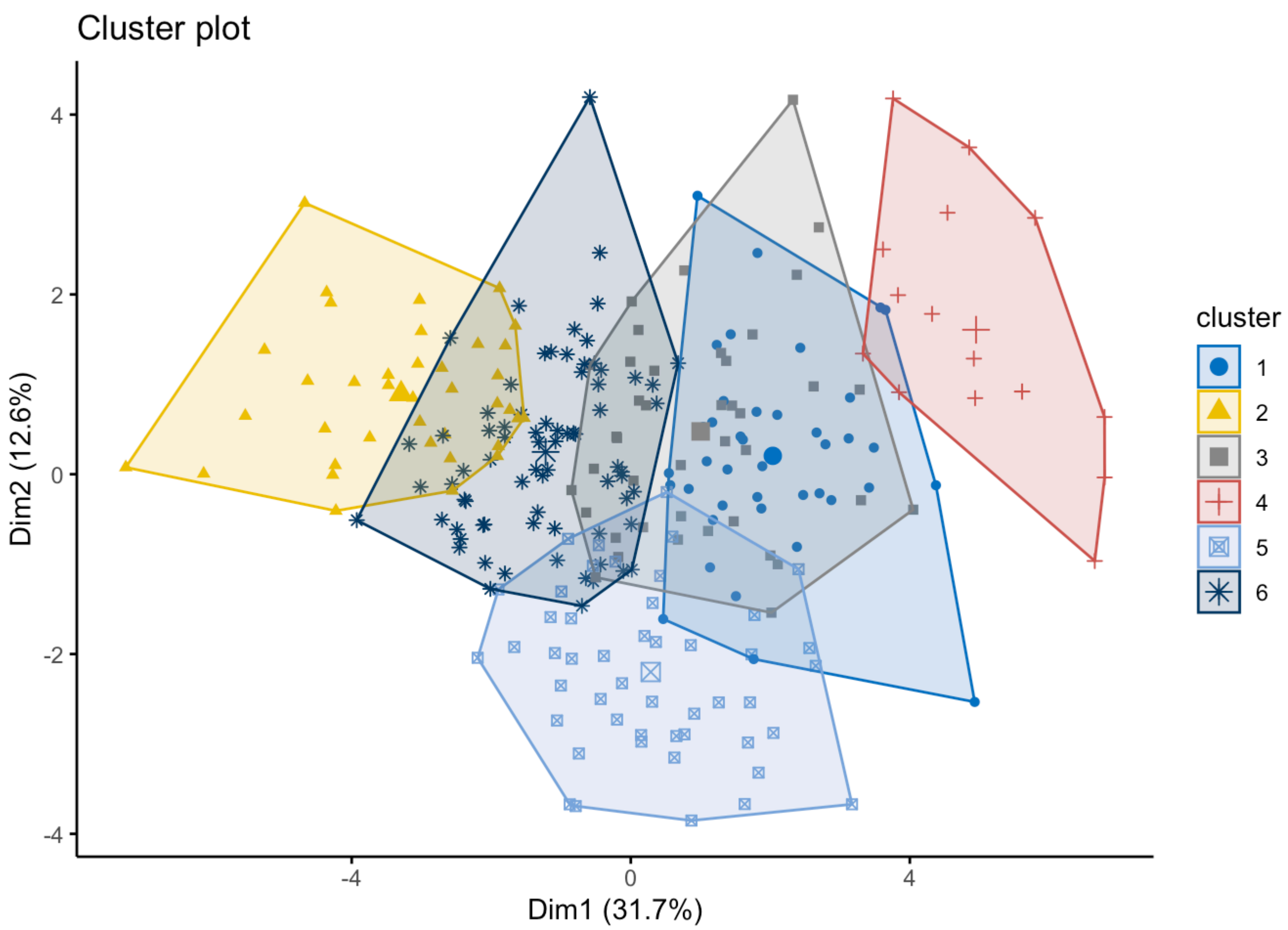
```
clusters$size
```

```
## [1] 40 38 41 16 48 74
```

Visualizing the clustering

(Uses principal components to collapse the dimensions of the data down to two dimensions)

```
fviz_cluster(clusters, clustering_input2,  geom = "point", show.clust.cent = TRUE, palette = "jco", ggtheme = the
me_classic())
```



Matrix indicating the mean values for each feature and cluster combination

```
clusters$centers
```

```
##        size promo_units_per high_med_gp velocityA_units_per
## 1 -0.8334802      0.85415068  -0.8530105           0.5675765
## 2  0.9730808     -0.96508940   0.9416705          -1.0999058
## 3 -0.4937076     -0.47571554  -0.5933514           0.3385144
## 4 -2.0769160      0.20998372  -1.3027639           1.9791388
## 5  0.5418821      0.35512582   0.6126291           0.4247053
## 6  0.3219522      0.02170173   0.1905726          -0.6329434
##   velocityB_units_per velocityC_units_per velocityD_units_per
## 1         -0.04868204        -0.896639723         -0.45935193
## 2          0.71493366         1.122293108          0.94452311
## 3         -0.50137236        -0.204722528          0.08575534
## 4         -1.21351814        -2.054344441         -1.32062712
## 5         -0.88946447         0.000469068         -0.01792813
## 6          0.77630614         0.465663111          0.01292992
##   velocityNEW_units_per energy_units_per regularBars_units_per gum_units_per
## 1            -0.3691618        0.2473944            0.69499660     0.3451923
## 2            -0.5259638       -0.9314910           -0.63590796    -1.2042494
## 3             0.2288860       -0.3266658            0.48639102     0.8768447
## 4            -0.8481852        0.7494395           -0.18801582     0.7576097
## 5             0.5612940        1.2090704           -0.39211615     0.5756207
## 6             0.1621300       -0.4207058           -0.02361574    -0.5911945
##   bagpegCandy_units_per isotonics_units_per singleServePotato_units_per
## 1            -0.2703180           0.6581372                 -0.71121731
## 2             0.7115161          -0.5592933                  0.60595393
## 3            -0.3718517          -0.6420075                  0.16954328
## 4            -2.0313090           1.0181920                  0.32429546
## 5            -0.1321374          -0.1862150                 -0.27042736
## 6             0.5116834           0.1878002                  0.08463452
##   takeHomePotato_units_per kingBars_units_per flatWater_units_per
## 1               -0.4939842          0.4229639        -0.661240307
## 2                0.7229799         -0.1074965         1.163693701
## 3               -0.6531160          0.1524344         0.752925614
## 4               -1.6795812          1.3379553         0.007204408
## 5                0.7027466         -0.9978398        -0.319939451
## 6                0.1649371          0.1000747        -0.451336155
##   psd591Ml_units_per
## 1          0.2376672
## 2          0.3601325
## 3         -1.0608783
## 4          0.2383237
## 5         -0.3122990
## 6          0.4254251
```

Naming the clusters

```
clustering_input1$cluster <- clusters$cluster

clustering_input1 <- clustering_input1 %>%
        mutate(cluster_labels = case_when(
                cluster==1 ~ 'Small Stores, Low Profits, Love Promos',
                cluster==2 ~ 'Largest Stores, Most Profitable, Love Flat Water',
                cluster==3 ~ 'Small Stores, Low Profits, Love Flat Water',
                cluster==4 ~ 'Smallest Stores, Lowest Profit, That Like Energy Drinks, Isotonics, King Bars',
                cluster==5 ~ 'Large Stores, Profitable, Lots of Energy Drinks, Big Chip Bags',
                cluster==6 ~ 'Mid Size, Bags of Candy'))

slice_sample(clustering_input1, n=50)
```

```
## # A tibble: 50 × 23
##    city    province region  size promo_units_per high_med_gp velocityA_units_per
##    <chr>   <chr>    <chr>  <dbl>           <dbl>       <dbl>               <dbl>
##  1 WESTBA… BC       WEST    850            0.343       0.980               0.623
##  2 OTTAWA  ON       ONTAR…  645.           0.378       0.535               0.657
##  3 LOGAN … BC       WEST    764            0.394       0.808               0.638
##  4 ALDERS… AB       WEST   1046            0.285       1                   0.583
##  5 NANAIMO BC       WEST    529            0.438       0                   0.662
##  6 MONTRÃ… QC       QUEBEC  797.           0.336       0.725               0.649
##  7 LONGUE… QC       QUEBEC  794.           0.415       0.469               0.670
##  8 ACTON   ON       ONTAR…  394            0.340       0                   0.688
##  9 MAIDST… ON       ONTAR…  475            0.350       0                   0.702
## 10 OAK BL… MB       WEST    930            0.397       1                   0.601
## # ℹ 40 more rows
## # ℹ 16 more variables: velocityB_units_per <dbl>, velocityC_units_per <dbl>,
## #   velocityD_units_per <dbl>, velocityNEW_units_per <dbl>,
## #   energy_units_per <dbl>, regularBars_units_per <dbl>, gum_units_per <dbl>,
## #   bagpegCandy_units_per <dbl>, isotonics_units_per <dbl>,
## #   singleServePotato_units_per <dbl>, takeHomePotato_units_per <dbl>,
## #   kingBars_units_per <dbl>, flatWater_units_per <dbl>, …
```

Explanation of each name

The clusters have been assigned labels based on certain characteristics. Let’s break down the cluster names:

- **Small Stores, Low Profits, Love Promos (Cluster 1):**
  - Features indicative of small value in size.
  - Low profitability suggested by the negative value in high_med_gp.
  - Affinity for promotions, as implied by high positive values in promo_units_per.
- **Largest Stores, Most Profitable, Love Flat Water (Cluster 2):**
  - Largest stores indicated by the highest positive value in size.
  - Most profitable, as suggested by the highest positive value in high_med_gp.
  - Affection for flat water, as indicated by the high positive value in flatWater_units_per.
- **Small Stores, Low Profits, Love Flat Water (Cluster 3):**
  - Similar to Cluster 1, these are small stores with low profitability.
  - Love for flat water, as indicated by the high positive value in flatWater_units_per.
- **Smallest Stores, Lowest Profit, That Like Energy Drinks, Isotonics, King Bars (Cluster 4):**
  - Smallest stores indicated by the lowest value in size.
  - Lowest profitability, as suggested by the largest negative value in high_med_gp.

  - Preference for energy drinks, isotonics, and king bars, indicated by high positive values in relevant features.
- **Large Stores, Profitable, Lots of Energy Drinks, Big Chip Bags (Cluster 5):**
  - Large stores, indicated by the high positive value in size.
  - Profitable, as suggested by the high positive value in high_med_gp.
  - Specializing in energy drinks and big chip bags, as indicated by high positive values in energy_units_per and takeHomePotato_units_per.
- **Mid Size, Bags of Candy (Cluster 6):**
  - Mid-size stores, suggested by the moderate value in size.
  - Specializing in bags of candy, as implied by high positive values in bagpegCandy_units_per.

Largest and most profitable cluster cities Cluster 2

Recommendation NANSE should focus on cluster 4. These are the smallest stores, which have the lowest profit margin. To increase the profit margin, NANSE should consider having a promotion that if a customer buys a certain number of specific items, they get a percentage discount on another item (ensuring the discount is more than made up for by the profit on the other items). To stimulate increased purchases of higher-profit items, it is recommended to implement targeted promotions. For instance, offering customers a percentage discount on energy drinks when they buy a specified quantity of higher-profit items could incentivize them to diversify their purchases. Importantly, the discount on energy drinks should be strategically set to ensure that the increased sales of higher-margin items compensate for the discount, thereby fostering both customer engagement and improved profitability for these small stores.

Parry Sound and Trenton

```
# PARRY SOUND
parry_sound <- clustering_input1 %>%
    filter(city == "PARRY SOUND")
print(parry_sound$cluster_labels)
```

```
## [1] "Largest Stores, Most Profitable, Love Flat Water"
```

```
# TRENTON
parry_sound <- clustering_input1 %>%
    filter(city == "TRENTON")
print(parry_sound$cluster_labels)
```

```
## [1] "Smallest Stores, Lowest Profit, That Like Energy Drinks, Isotonics, King Bars"
```

Findings

- **Parry Sound:** This city has the largest stores with the highest profit margin. Customers love buying flat water in this city.
- **Trenton:** This city has the smallest stores with the lowest profit margin. Customers love buying energy drinks, isotonics, and King Bars in this city.

DBSCAN

Installing dbscan package and reading data

```
#install.packages('dbscan') #only install once
library(dbscan)
```

```
##
## Attaching package: 'dbscan'
```

```
## The following object is masked from 'package:stats':
##
##     as.dendrogram
```

```
# read in data (change to your path)
clustering_input1 <- read_rds("city.rds")

clustering_input2 <- clustering_input1 %>%
        select(-city, -region, -province )
```

Standardizing the data using z-score standardization.

```
clustering_input2 <- as.data.frame(scale(clustering_input2))
```

Running the DBSCAN algorithm

```
set.seed(42)
clusters_db <- dbscan::dbscan(clustering_input2, eps = 5, minPts = 4)
```

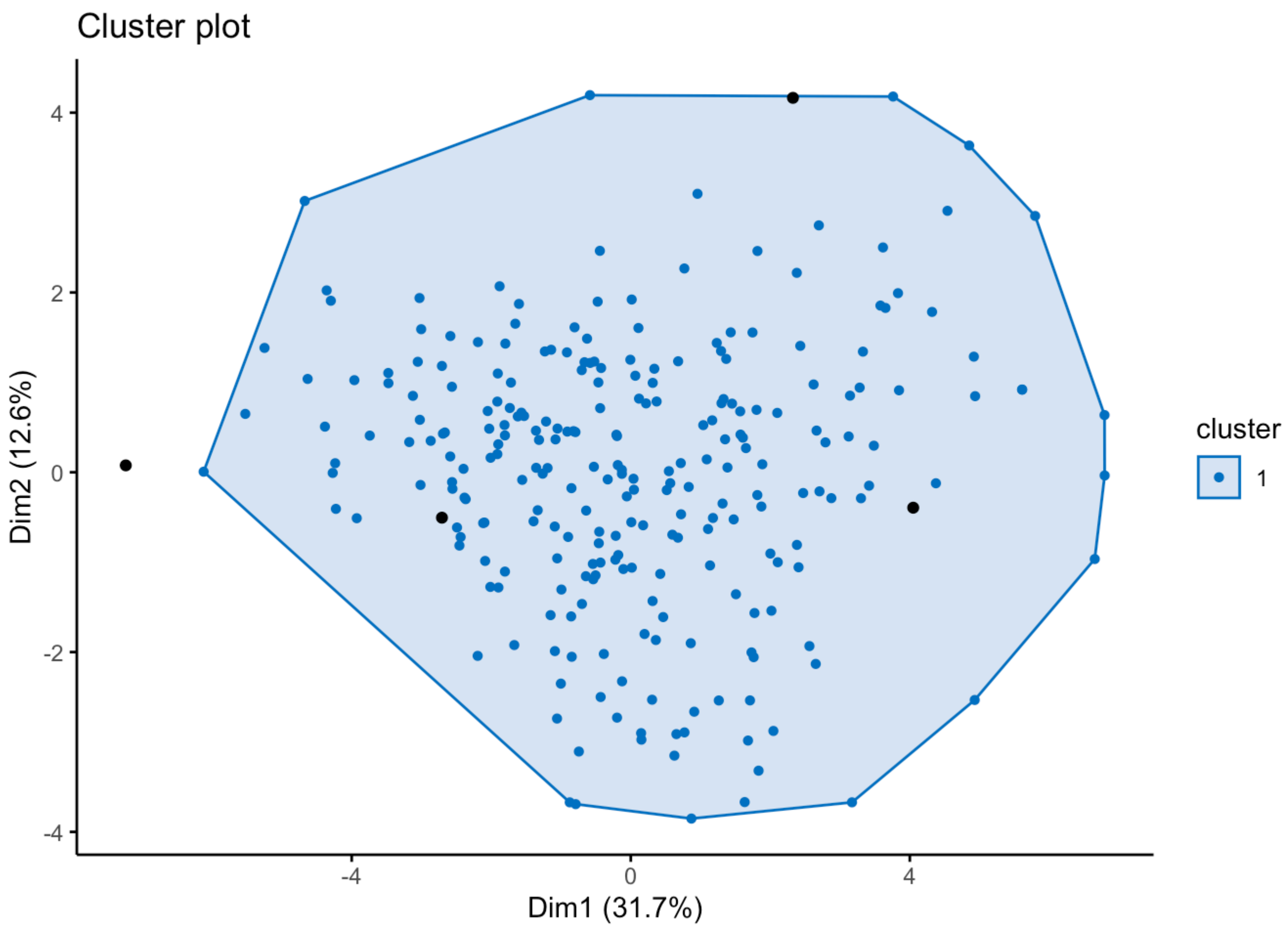
Printing the size of the clusters.

```
table(clusters_db$cluster)
```

```
##
##   0   1
##   4 253
```

Visualizing the clusters

```
fviz_cluster(clusters_db, clustering_input2,  geom = "point", show.clust.cent = FALSE, palette = "jco", ggtheme =
theme_classic())
```



There is 1 cluster, and 253 cities are in this cluster.

Why KMeans works better for this dataset * K-means proves to be a more helpful clustering method for this dataset compared to DBSCAN due to its capacity to clearly break down the dataset into six distinct clusters. The output of K-means provides valuable insights into the grouping of stores, allowing for a comprehensive understanding of different segments within the data. This breakdown into clusters facilitates a more granular analysis, enabling NANSE to identify specific stores that share similar characteristics. Unlike DBSCAN, which may not provide as explicit cluster assignments, K-means produces well-defined clusters that can guide targeted strategies. Moreover, it's worth noting that DBSCAN is particularly effective on dense datasets, and since our dataset is not dense, it may not be as helpful in this context. * Therefore, we recommend that NANSE leverages the K-means method for its clustering analysis, as it not only groups stores effectively but also facilitates a more nuanced approach to store management and optimization.

How each method deals with outliers K-means did not remove any outliers but rather created clusters that incorporated the outliers. DBSCAN actually did remove 4 outlier data points.

The handling of outliers differs significantly between K-means and DBSCAN clustering methods. In the case of K-means, the algorithm typically does not explicitly remove outliers but instead incorporates them into the clusters it forms. K-means assigns each data point to the nearest cluster center, which means outliers can influence the centroid positions and potentially impact the overall structure of the clusters.

On the other hand, DBSCAN takes a different approach. DBSCAN identifies outliers as noise points, treating them as data points that do not belong to any cluster. This method effectively removes outliers by categorizing them as noise during the clustering process.