# DANA 4840_Group Project

Part 1 : New York College Offenses Clustering

## New York College Offenses Clustering

## 1. Data Introduction and purpose of choosing this dataset

### Data introduction

Dataset for Hierarchical Clustering

Name : New York Offenses Known to Law

Enforcement by University and College, 2019

Data Source : FBI Uniform Crime Reporting (UCR) Program (https://shorturl.at/unSho)

Number of Observations : 27 rows & 12 columns

### Purpose

This dataset is incredibly useful for understanding crime trends on college campuses in New York state. Here are some of its key applications:

1. Campus Safety Assessment:

- For students and parents: The data allows prospective students and their families to compare crime rates across different colleges and universities. This information can help them make informed decisions about where to attend.
- For administrators: Colleges and universities can use this data to identify areas of concern and prioritize safety initiatives. They can use the data to target resources to high-crime areas or develop new security measures.

2. Research & Analysis:

- Crime trends: Researchers can analyze the data to identify trends in campus crime over time. For example, they can study whether certain types of crime are increasing or decreasing, and they can explore potential causes.
- Risk factors: Researchers can investigate whether there is a correlation between crime rates and factors such as campus size, student demographics, or the surrounding community.

3. Policy Development: Informed policy: Law enforcement agencies and policymakers can use this data to develop effective strategies for preventing and responding to crime on college campuses. For example, they might use the data to target crime prevention programs or to allocate law enforcement resources.

4. Public Awareness: Transparency and accountability: By making this data publicly available, colleges and universities promote transparency and accountability in their security measures.

Community engagement: The data can foster a dialogue between law enforcement, college administrators, and the campus community about crime prevention and safety.

Objective: To analyze crime statistics across different universities and colleges in New York and uncover patterns or groupings using hierarchical clustering.

```
library(readxl)

## Warning: package 'readxl' was built under R version 4.3.2

data <- read_excel("file_show.xlsx")

head(data)

## # A tibble: 6 × 12
##   Campus    `Student\nenrollment1` `Violent\ncrime` Murder and\nnonnegli…¹
Rape2
##   <chr>                      <dbl>            <dbl>                  <dbl>
<dbl>
## 1 <NA>                        7036               12                      0
11
## 2 Alfred                      4060                2                      0
1
## 3 Binghamt…                  18892                4                      0
3
## 4 Brockport                   9539                2                      0
2
## 5 Buffalo                    34183               12                      0
6
## 6 Buffalo …                  11048                7                      0
2
## # ℹ abbreviated name: ¹`Murder and\nnonnegligent\nmanslaughter`
## # ℹ 7 more variables: Robbery <dbl>, `Aggravated\nassault` <dbl>,
## #   `Property\ncrime` <dbl>, Burglary <dbl>, `Larceny-\ntheft` <dbl>,
## #   `Motor\nvehicle\ntheft` <dbl>, Arson <dbl>

# Remove the first row of the data because it's the sum of all campus.
data <- data[-1, ]

# Convert tibble to a base R data frame
data <- as.data.frame(data)
```

```r
# Set 'Campus' column as row names
rownames(data) <- data$Campus

# Remove the 'Campus' column
df <- data[, -1]

# Print the first few rows of the updated dataframe to confirm
head(df)
```

```
##                        Student\nenrollment1 Violent\ncrime
## Alfred                                 4060              2
## Binghamton                            18892              4
## Brockport                              9539              2
## Buffalo                               34183             12
## Buffalo State College                 11048              7
## Canton                                 4981              1
##                        Murder and\nnonnegligent\nmanslaughter Rape2 Robbery
## Alfred                                                      0     1       0
## Binghamton                                                  0     3       0
## Brockport                                                   0     2       0
## Buffalo                                                     0     6       1
## Buffalo State College                                       0     2       2
## Canton                                                      0     1       0
##                        Aggravated\nassault Property\ncrime Burglary
## Alfred                                   1              40        3
## Binghamton                               1             158       12
## Brockport                                0              47        5
## Buffalo                                  5             208       18
## Buffalo State College                    3             122        3
## Canton                                   0              26        3
##                        Larceny-\ntheft Motor\nvehicle\ntheft Arson
## Alfred                              36                     1     0
## Binghamton                         146                     0     1
## Brockport                           41                     1     0
## Buffalo                            189                     1     0
## Buffalo State College              118                     1     0
## Canton                              23                     0     0
```

```r
# Get the dimensions of the dataset
dataset_dimensions <- dim(df)

# Print the dimensions
cat("Number of rows:", dataset_dimensions[1], "\n")
```

```
## Number of rows: 25
```

```r
cat("Number of columns:", dataset_dimensions[2], "\n")
```

```
## Number of columns: 11
```

```r
summary(df)

##  Student\nenrollment1  Violent\ncrime Murder
and\nnonnegligent\nmanslaughter
##  Min.   :  413        Min.   : 0.00   Min.   :0
##  1st Qu.: 3632        1st Qu.: 1.00   1st Qu.:0
##  Median : 5490        Median : 2.00   Median :0
##  Mean   : 8783        Mean   : 2.56   Mean   :0
##  3rd Qu.:10014        3rd Qu.: 4.00   3rd Qu.:0
##  Max.   :34183        Max.   :12.00   Max.   :0
##  NA's   :1
##      Rape2          Robbery       Aggravated\nassault Property\ncrime
##  Min.   :0.00   Min.   :0.00   Min.   :0.00        Min.   :  3.00
##  1st Qu.:0.00   1st Qu.:0.00   1st Qu.:0.00        1st Qu.: 26.00
##  Median :1.00   Median :0.00   Median :0.00        Median : 39.00
##  Mean   :1.44   Mean   :0.24   Mean   :0.88        Mean   : 55.28
##  3rd Qu.:2.00   3rd Qu.:0.00   3rd Qu.:1.00        3rd Qu.: 47.00
##  Max.   :6.00   Max.   :2.00   Max.   :6.00        Max.   :208.00
##
##    Burglary      Larceny-\ntheft  Motor\nvehicle\ntheft     Arson
##  Min.   : 0.00   Min.   :  2.00   Min.   :0.00          Min.   :0.00
##  1st Qu.: 1.00   1st Qu.: 19.00   1st Qu.:0.00          1st Qu.:0.00
##  Median : 3.00   Median : 33.00   Median :0.00          Median :0.00
##  Mean   : 4.64   Mean   : 50.28   Mean   :0.36          Mean   :0.16
##  3rd Qu.: 6.00   3rd Qu.: 44.00   3rd Qu.:1.00          3rd Qu.:0.00
##  Max.   :18.00   Max.   :189.00   Max.   :3.00          Max.   :1.00
##
```

Here are the insights from the dataset of 25 different campuses:

```
The average number of students enrolled at each campus is 8783.
The average number of violent crimes reported is 3.
There are no murders and nonnegligent manslaughters reported.
The average number of reported rapes is almost 1.5.
The average number of reported robberies is almost 0.2.
The average number of reported aggravated assaults is almost 1.
The average number of property crimes reported is 55.
The average number of reported burglaries is almost 5.
The average number of reported larceny-thefts is 50.
The average number of reported motor vehicle thefts is almost 0.4.
The average number of reported arson incidents is almost 0.2.
```

# 3. EDA

## Checking for Missing Values

```r
sum(is.na(df))

## [1] 1
```

```r
# Check for missing values in the entire dataset
missing_values_summary <- sapply(df, function(x) sum(is.na(x)))

# Print the summary of missing values
print(missing_values_summary)
```

```
##                    Student\nenrollment1
Violent\ncrime
##                                       1
0
## Murder and\nnonnegligent\nmanslaughter
Rape2
##                                       0
0
##                                 Robbery
Aggravated\nassault
##                                       0
0
##                       Property\ncrime
Burglary
##                                       0
0
##                       Larceny-\ntheft
Motor\nvehicle\ntheft
##                                       0
0
##                                   Arson
##                                       0
```

```r
# Drop rows with missing values
df <- na.omit(df)
```

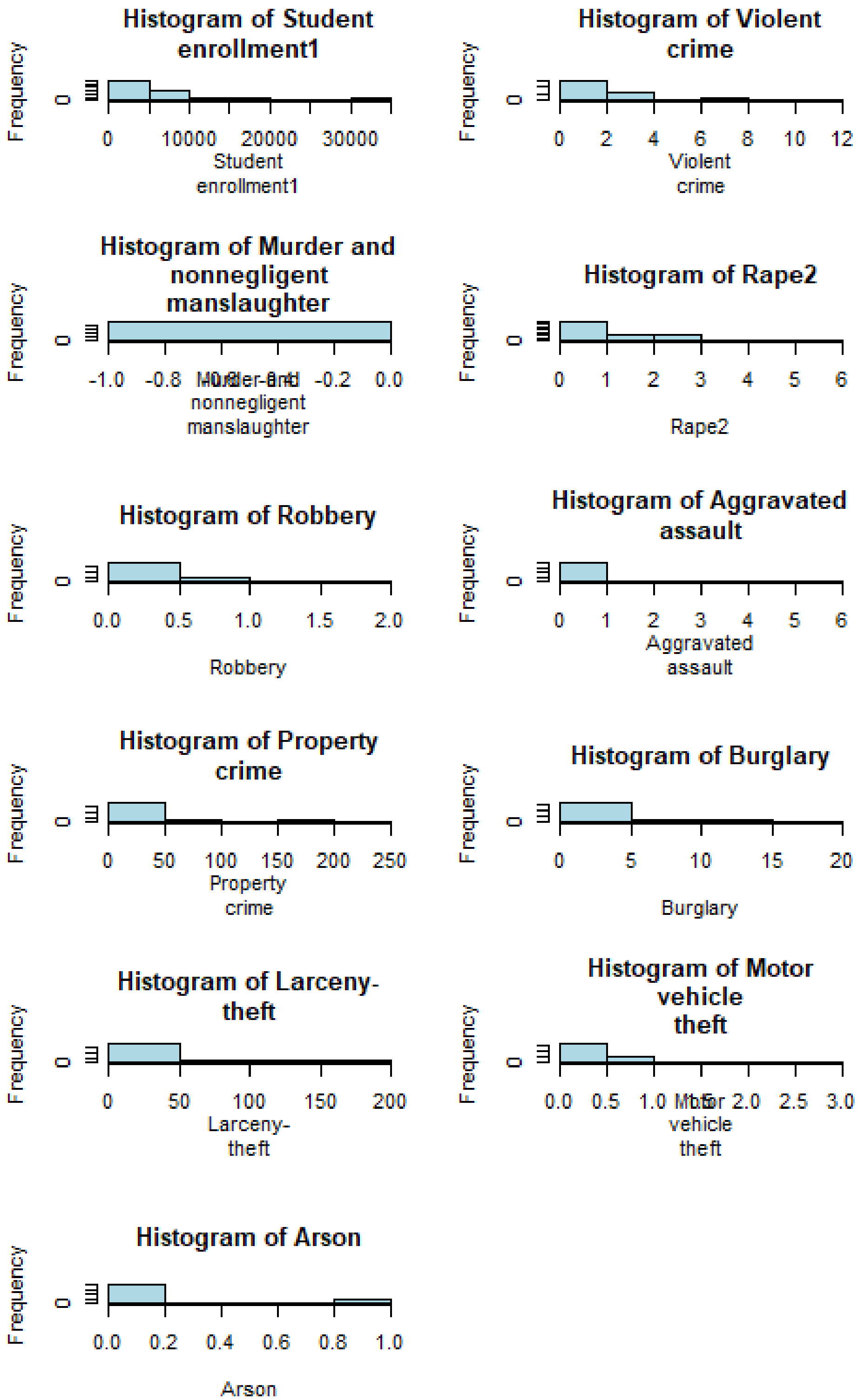
Student enrollment figures of campus Nanoscale Science and Engineering were not available, so we drop this row.

### Histograms of numerical variables

```r
# Load necessary library for plotting
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.3
```

```r
# Set up plotting area
par(mfrow=c(3, 2))  # Adjust the layout as needed, here 2 rows and 4 columns

# Plot histograms
for (col_name in names(df)) {
  hist(df[[col_name]], main=paste("Histogram of", col_name),
       xlab=col_name, col='lightblue', border='black')
}
```


Histogram of Student enrollment1
Frequency
0
0 10000 20000 30000
Student enrollment1
Histogram of Violent crime
Frequency
0
0 2 4 6 8 10 12
Violent crime
Histogram of Murder and nonnegligent manslaughter
Frequency
0
-1.0 -0.8 -0.6 -0.4 -0.2 0.0
Murder and nonnegligent manslaughter
Histogram of Rape2
Frequency
0
0 1 2 3 4 5 6
Rape2
Histogram of Robbery
Frequency
0
0.0 0.5 1.0 1.5 2.0
Robbery
Histogram of Aggravated assault
Frequency
0
0 1 2 3 4 5 6
Aggravated assault
Histogram of Property crime
Frequency
0
0 50 100 150 200 250
Property crime
Histogram of Burglary
Frequency
0
0 5 10 15 20
Burglary
Histogram of Larceny-theft
Frequency
0
0 50 100 150 200
Larceny-theft
Histogram of Motor vehicle theft
Frequency
0
0.0 0.5 1.0 1.5 2.0 2.5 3.0
Motor vehicle theft
Histogram of Arson
Frequency
0
0.0 0.2 0.4 0.6 0.8 1.0
Arson

summary of the histograms:

2. **Student Enrollment**: The histogram shows a right-skewed distribution, with the majority of schools having student enrollments clustered at the lower end of the range, and a few schools having significantly higher enrollments.

3. **Violent Crime and Its Components**:

   - **Violent Crime**: The majority of schools have a low number of violent crimes, with a steep drop-off as the number increases.
   - **Murder and Non-negligent Manslaughter**: Most schools report very low to no incidents.
   - **Rape**: The data is right-skewed with most values near zero.
   - **Robbery**: Similar to rape, most schools report low incidents.
   - **Aggravated Assault**: Most schools report very few incidents, with a steep decline as incidents increase.
4. **Property Crime and Its Components**:

   - **Property Crime**: Generally low across schools, with a small number reporting higher incidents.
   - **Burglary**: Most schools report very few incidents.
   - **Larceny-Theft**: Few schools report high incidents, but most are low.
   - **Motor Vehicle Theft**: Most schools report very few incidents.
   - **Arson**: Most schools report very low or no incidents, with a few exceptions.
5. **Frequency Distribution**: Across all categories, the data is typically right-skewed, indicating that most schools have low to moderate crime rates, with a few schools experiencing higher rates.

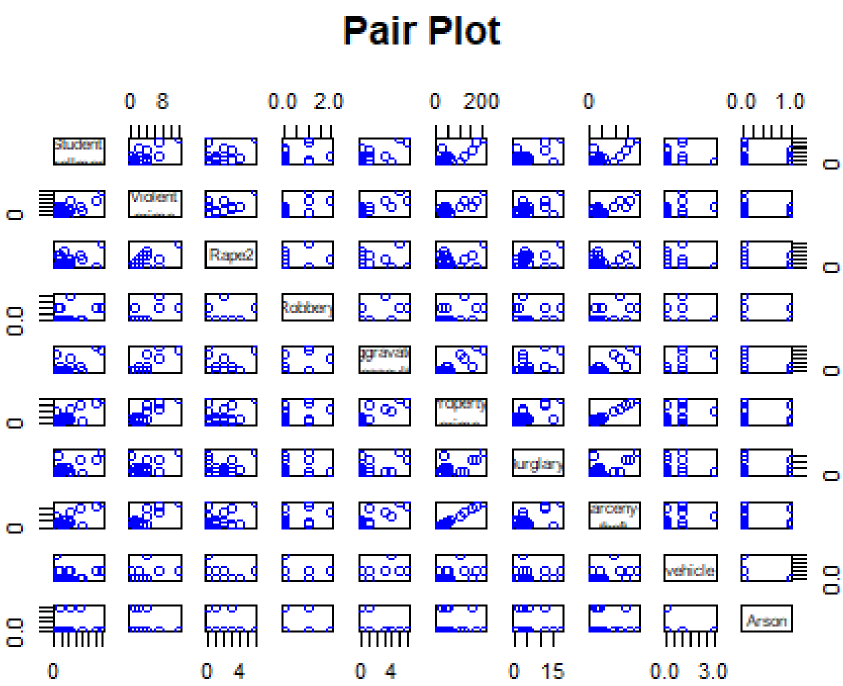
6. **Overall Trends**: The histograms illustrate that while most schools experience low levels of both violent and property crimes, certain types of crimes like larceny-theft and property crime tend to have more frequent higher incidents compared to others like murder, motor vehicle theft, and arson.

The column Murder and nonnegligent manslaughter has all value of 0 across the campus so we’re going to drop it. We decided to drop this column.

```r
# Drop the column "Murder and nonnegligent manslaughter"
df <- df[, -3]
```

### Create a Pair Plot with Color by Type

```r
# Create the pair plot
pairs(df, main = "Pair Plot", col = "blue")
```




```r
# Save the pair plot as a PNG file with higher resolution
#png("pair_plot.png", width = 2000, height = 2000, res = 300)
```

Property Crime and Larceny-Theft are extremely strongly correlated, and there is a significant positive relationship between several types of crimes and student enrollment.

### Check for Correlations Between Variables

```r
# Compute the correlation matrix
correlation_matrix <- cor(df, use = "complete.obs")

# Find pairs with correlation > 0.7 or < -0.7
high_cor_pairs <- which(abs(correlation_matrix) > 0.7 &
abs(correlation_matrix) < 1, arr.ind = TRUE)

# Print the pairs with high correlation
if (nrow(high_cor_pairs) > 0) {
  for (i in 1:nrow(high_cor_pairs)) {
    var1 <- rownames(correlation_matrix)[high_cor_pairs[i, 1]]
    var2 <- colnames(correlation_matrix)[high_cor_pairs[i, 2]]
    cor_value <- correlation_matrix[high_cor_pairs[i, 1], high_cor_pairs[i,
2]]
    cat(var1, "&", var2, ": ", cor_value, "\n")
  }
} else {
  cat("No pairs with correlation > 0.7 or < -0.7 found.")
}
```

```
## Violent
## crime & Student
## enrollment1 :  0.7425941
## Property
## crime & Student
## enrollment1 :  0.7951593
## Larceny-
## theft & Student
## enrollment1 :  0.7900293
## Student
## enrollment1 & Violent
## crime :  0.7425941
## Aggravated
## assault & Violent
## crime :  0.8250811
## Property
## crime & Violent
## crime :  0.8491228
## Larceny-
## theft & Violent
## crime :  0.8446972
## Violent
## crime & Aggravated
## assault :  0.8250811
## Property
## crime & Aggravated
## assault :  0.8580986
## Larceny-
## theft & Aggravated
## assault :  0.857747
## Student
## enrollment1 & Property
## crime :  0.7951593
## Violent
## crime & Property
## crime :  0.8491228
## Aggravated
## assault & Property
## crime :  0.8580986
## Larceny-
## theft & Property
## crime :  0.9978073
## Student
## enrollment1 & Larceny-
## theft :  0.7900293
## Violent
## crime & Larceny-
## theft :  0.8446972
## Aggravated
## assault & Larceny-
```

```
## theft :  0.857747
## Property
## crime & Larceny-
## theft :  0.9978073
```

Summary of the pair plot with the provided correlation values:

7. **Student Enrollment vs. Crime Types**:
    - **Student Enrollment and Violent Crime**: There is a strong positive correlation (0.7426) between student enrollment and violent crime, indicating that schools with higher enrollments tend to report more violent crimes.
    - **Student Enrollment and Property Crime**: An even stronger positive correlation (0.7952) suggests that larger schools also tend to have higher property crime rates.
    - **Student Enrollment and Larceny-Theft**: A strong positive correlation (0.7900) exists between student enrollment and larceny-theft, indicating that this type of property crime is more prevalent in larger schools.
8. **Violent Crime Correlations**:
    - **Violent Crime and Aggravated Assault**: There is a strong positive correlation (0.8251), indicating that schools with higher violent crime rates also tend to have higher rates of aggravated assault.
    - **Violent Crime and Property Crime**: A strong positive correlation (0.8491) suggests that schools with higher violent crime rates also report higher property crime rates.
    - **Violent Crime and Larceny-Theft**: There is a strong positive correlation (0.8447) between violent crime and larceny-theft.
9. **Property Crime Correlations**:
    - **Property Crime and Aggravated Assault**: A very strong positive correlation (0.8581) indicates that schools with higher property crime rates also experience higher rates of aggravated assault.
    - **Property Crime and Larceny-Theft**: There is an extremely high positive correlation (0.9978), indicating that larceny-theft is a significant component of property crime.
    - **Larceny-Theft and Aggravated Assault**: A strong positive correlation (0.8577) exists between larceny-theft and aggravated assault.
10. **Inter-Crime Correlations**:
    - The pair plot and the correlation values indicate significant interdependencies among different crime types, with strong correlations observed between violent and property crimes, as well as between specific types of crimes such as aggravated assault and larceny-theft.

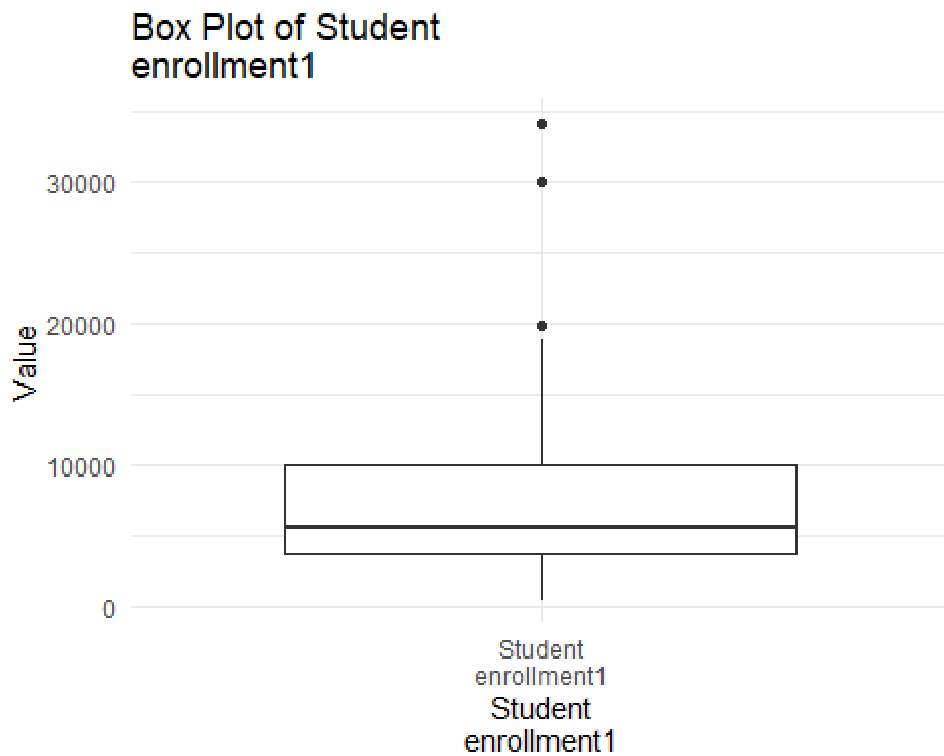
### Create the box plot

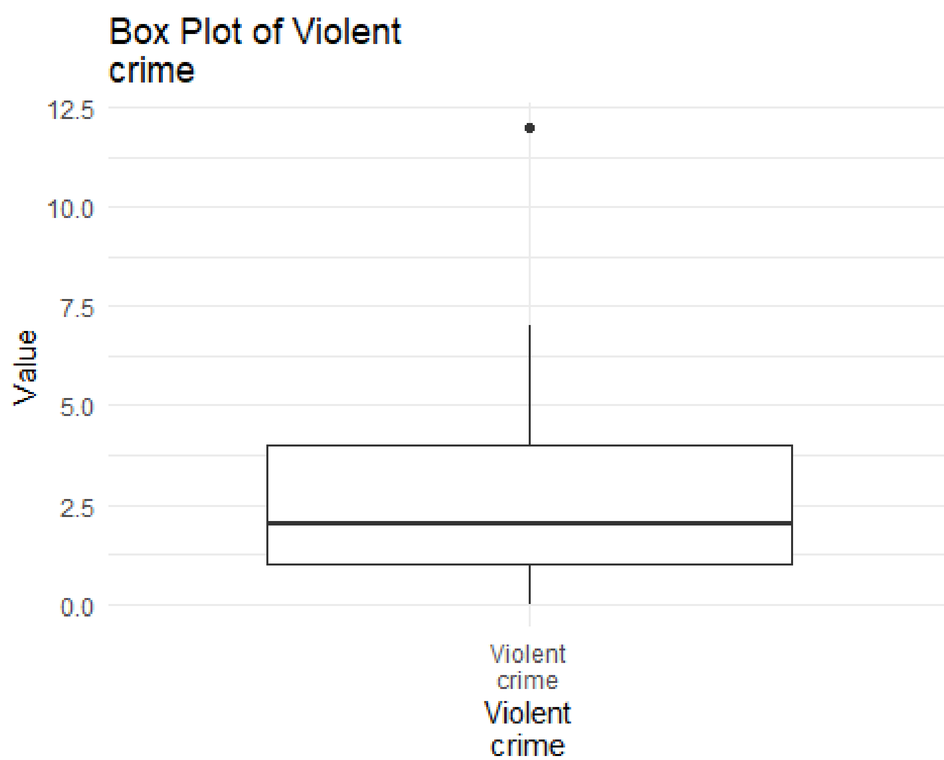
```r
library(tidyr)

## Warning: package 'tidyr' was built under R version 4.3.2
```

```r
library(dplyr)
```

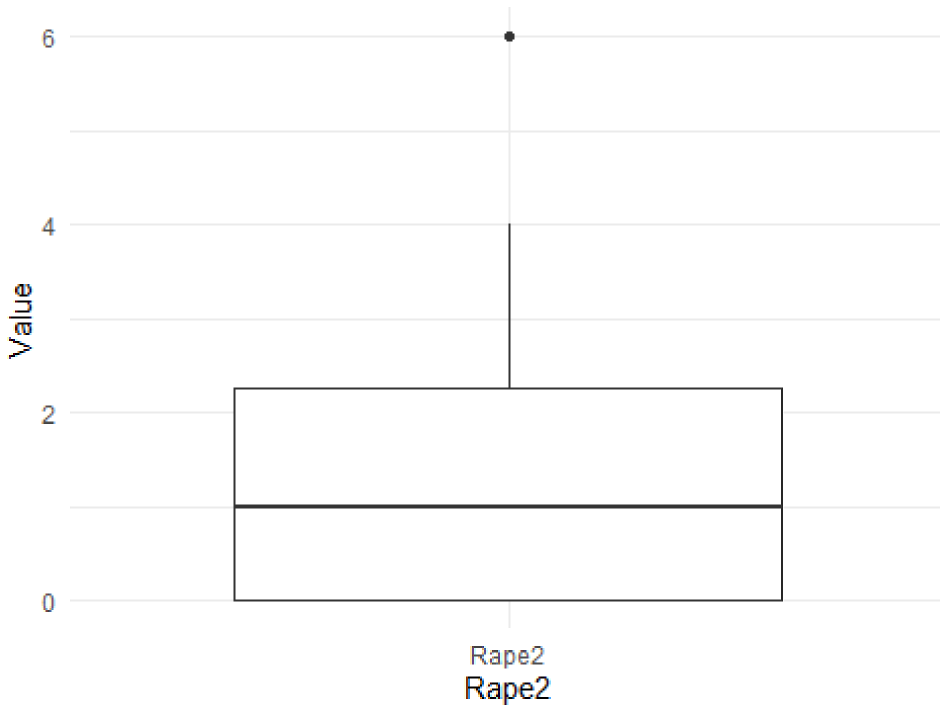
```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Reshape the data for ggplot2
subset_df_long <- pivot_longer(df, cols = everything(), names_to =
"variable", values_to = "value")

# List of unique variables
variables <- unique(subset_df_long$variable)

# Create and print separate box plots for each variable
for (var in variables) {
  # Filter data for the current variable
  data <- subset_df_long %>% filter(variable == var)

  # Create the box plot
  p <- ggplot(data, aes(x = variable, y = value)) +
    geom_boxplot() +
    labs(title = paste("Box Plot of", var), x = var, y = "Value") +
    theme_minimal()

  # Print the plot
  print(p)
}
```

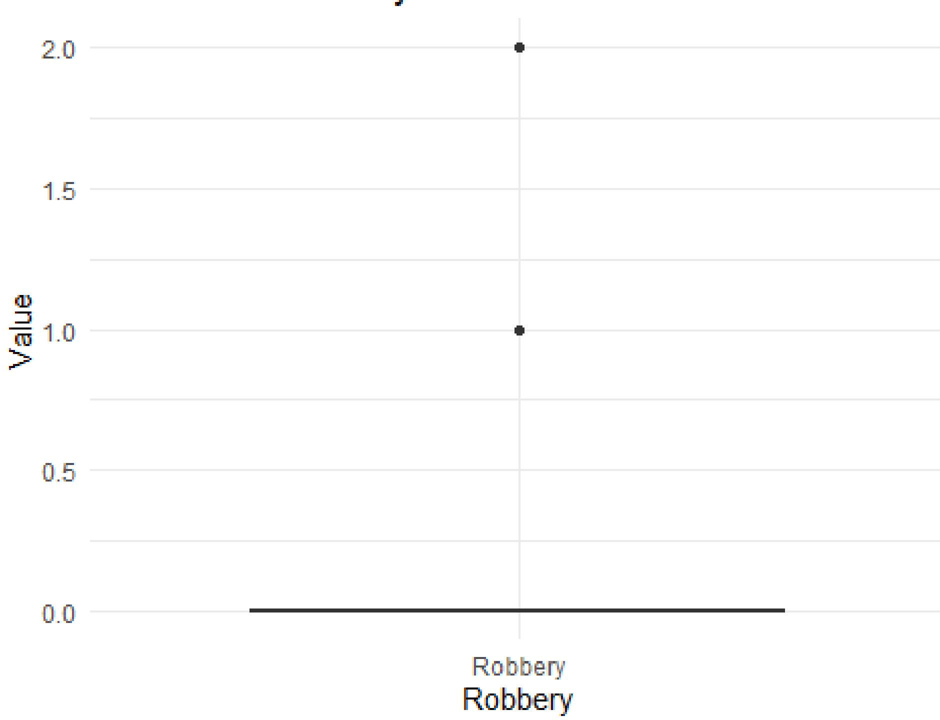

Box Plot of Student
enrollment1
Value
30000
20000
10000
0
Student
enrollment1
Student
enrollment1

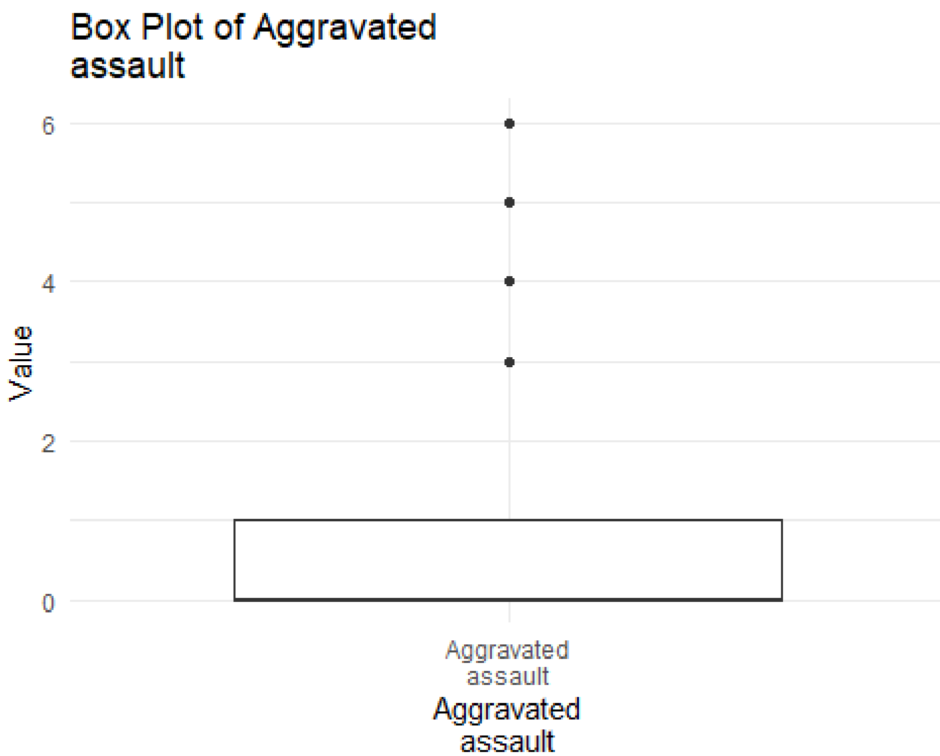

Box Plot of Violent
crime
Value
12.5
10.0
7.5
5.0
2.5
0.0
Violent
crime
Violent
crime

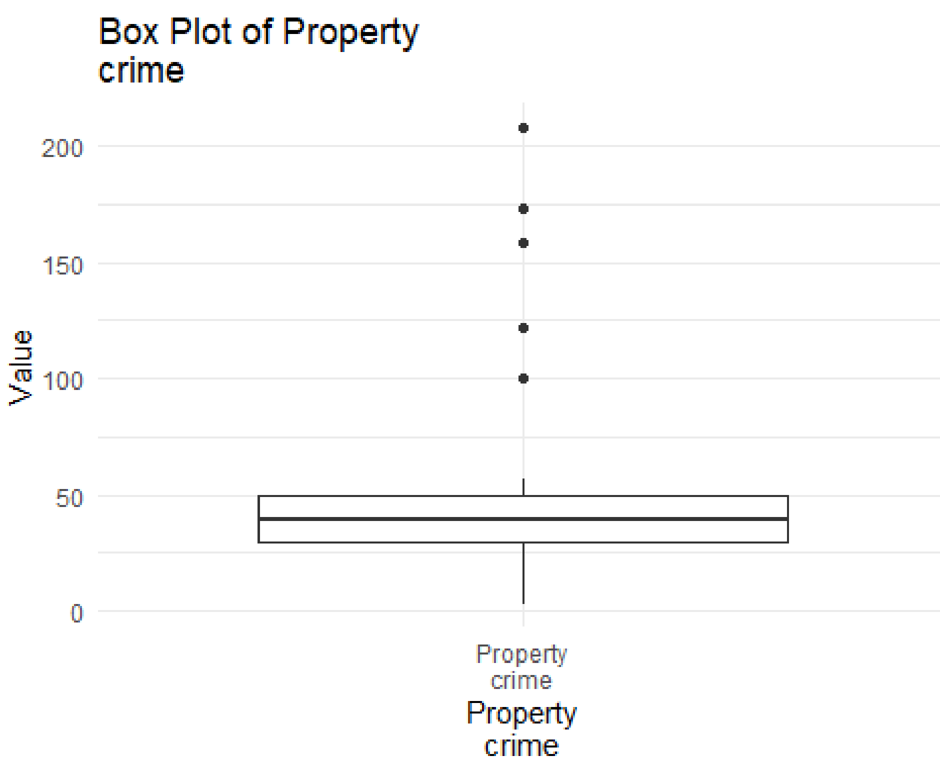
Box Plot of Rape2

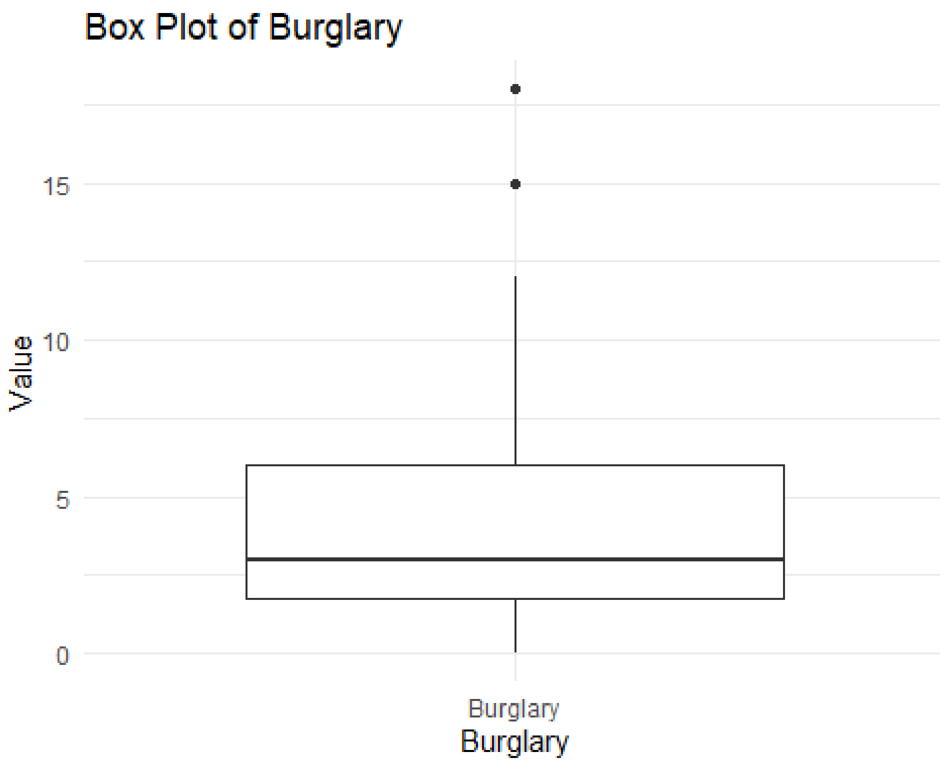


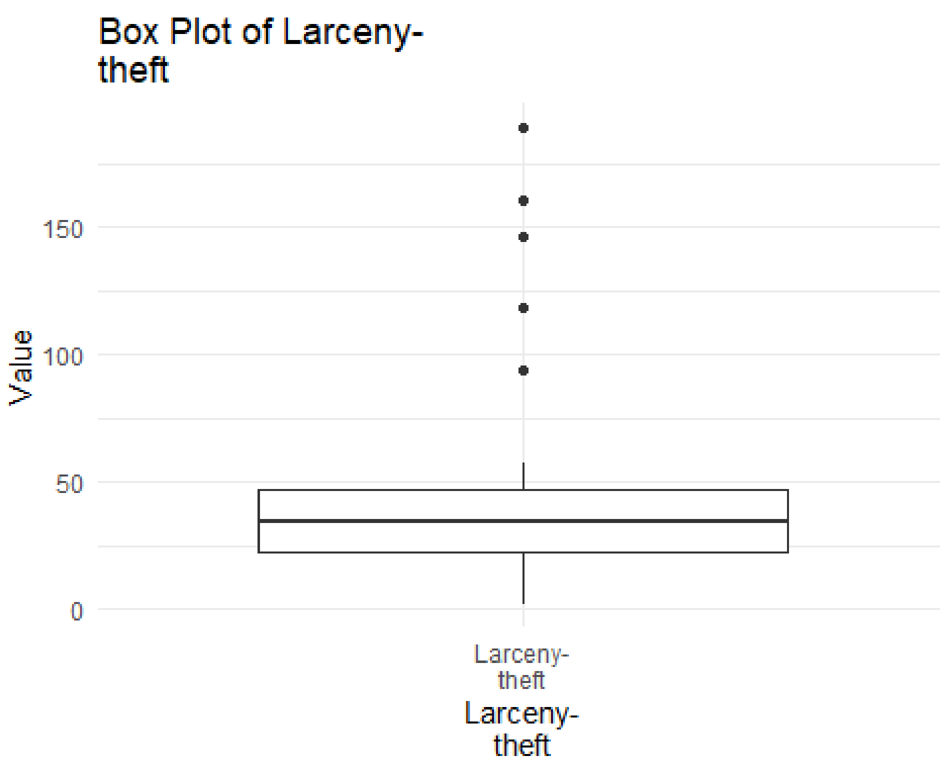

Box Plot of Robbery

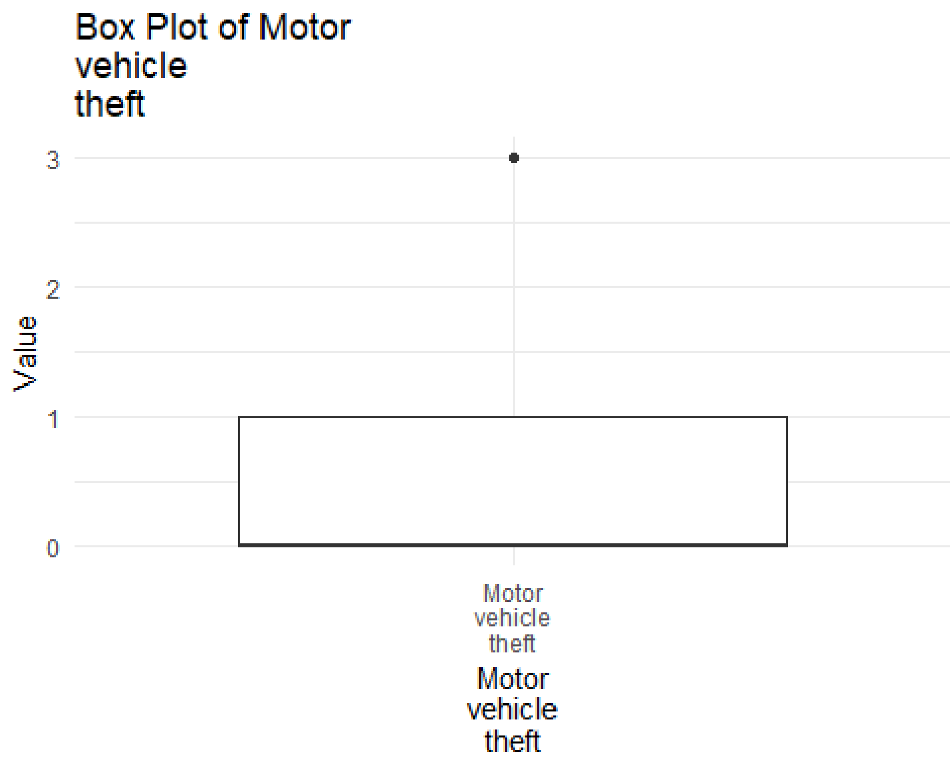


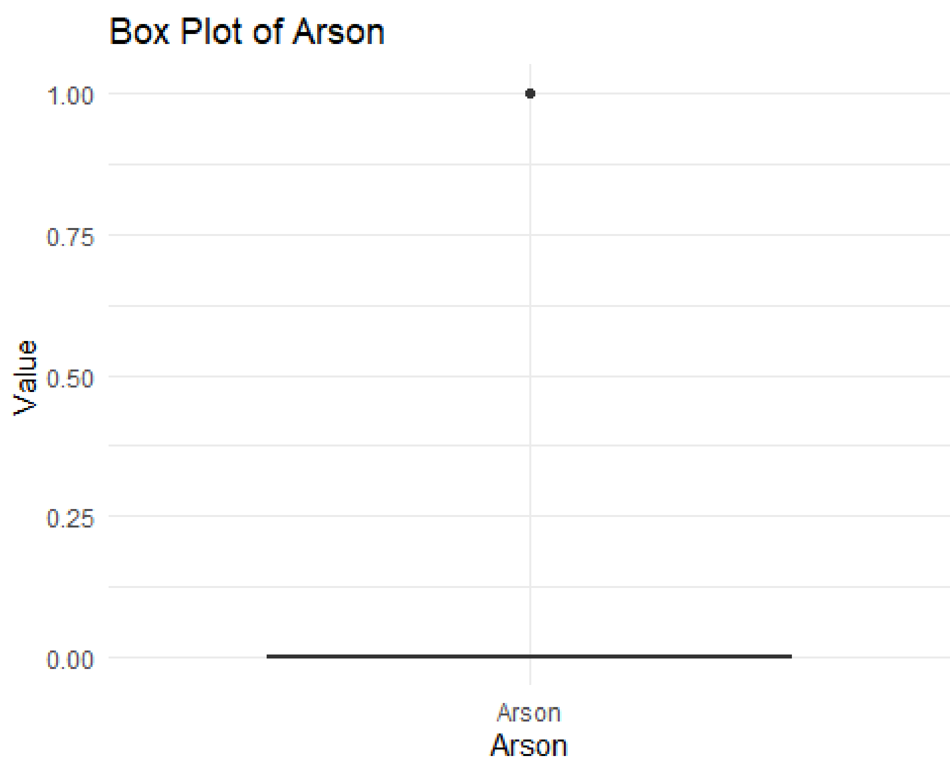

Box Plot of Aggravated
assault
Value
6
4
2
0
Aggravated
assault
Aggravated
assault


Box Plot of Property
crime
Value
200
150
100
50
0
Property
crime
Property
crime


Box Plot of Burglary
Value
15
10
5
0
Burglary
Burglary


Box Plot of Larceny-theft
Value
150
100
50
0
Larceny-theft
Larceny-theft




There are a few outliers for each variable, but this is because certain campuses have higher crime rates than others, and it is important to retain these for further analysis."

This revised sentence provides a clearer explanation of the reasoning behind keeping the outliers for further analysis.

```
# Function to identify outliers based on IQR
find_outliers <- function(x) {
  Q1 <- quantile(x, 0.25)
  Q3 <- quantile(x, 0.75)
  IQR <- Q3 - Q1
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR
  x[x < lower_bound | x > upper_bound]
}

# Create an empty list to store outlier data
outliers_list <- list()

# Identify outliers for each column and store in the list
for (i in seq_along(df)) {
  outliers <- find_outliers(df[[i]])
  if (length(outliers) > 0) {
    outliers_list[[names(df)[i]]] <- outliers
  }
}
outliers_list

## $`Student\nenrollment1`
## [1] 34183 19903 30012
##
## $`Violent\ncrime`
## [1] 12
##
## $Rape2
## [1] 6
##
## $Robbery
## [1] 1 2 1 1 1
##
## $`Aggravated\nassault`
## [1] 5 3 6 4
##
## $`Property\ncrime`
## [1] 158 208 122 173 100
##
## $Burglary
## [1] 18 15
##
## $`Larceny-\ntheft`
## [1] 146 189 118 160  94
##
## $`Motor\nvehicle\ntheft`
```

```
## [1] 3
##
## $Arson
## [1] 1 1 1 1
```

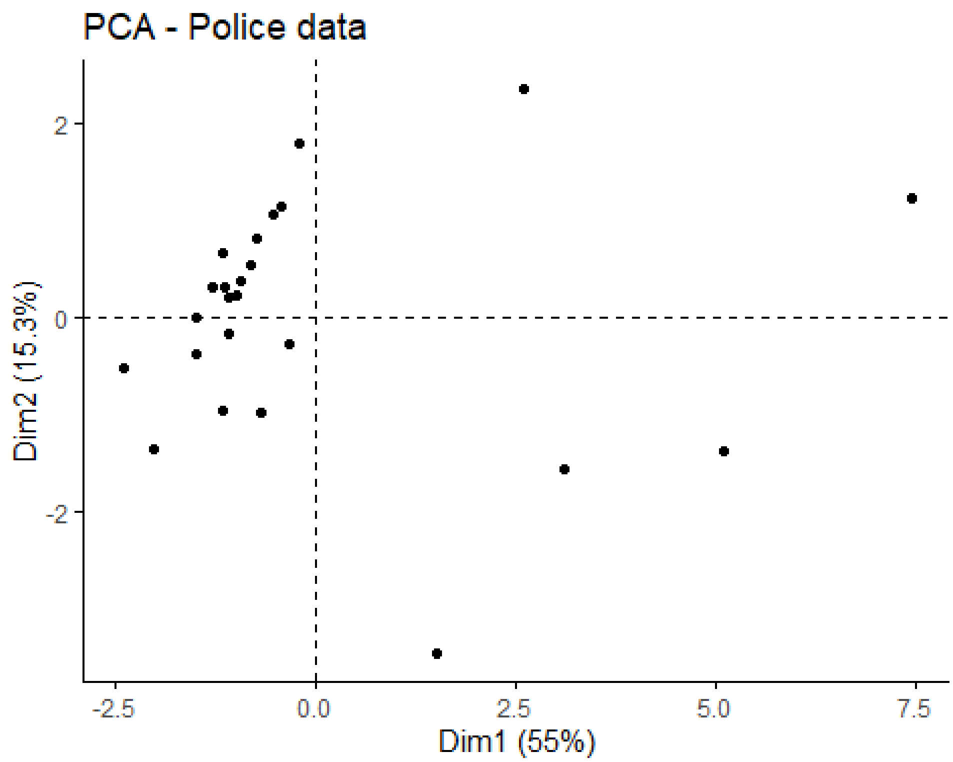
### 3. Assessing clustering tendency

```
df.scaled <- scale(df)

head(df.scaled)

##                       Student\nenrollment1 Violent\ncrime      Rape2
Robbery
## Alfred                         -0.54274019     -0.2387493 -0.3204350 -
0.47027
## Binghamton                      1.16156331      0.4774986  0.9613049 -
0.47027
## Brockport                       0.08683632     -0.2387493  0.3204350 -
0.47027
## Buffalo                         2.91860922      3.3424905  2.8839148
1.41081
## Buffalo State College           0.26023128      1.5518706  0.3204350
3.29189
## Canton                         -0.43691067     -0.5968733 -0.3204350 -
0.47027
##                       Aggravated\nassault Property\ncrime    Burglary
## Alfred                         0.04782459      -0.3187533 -0.38288005
## Binghamton                     0.04782459       1.8564499  1.49671293
## Brockport                     -0.52607053      -0.1897158  0.03480728
## Buffalo                        2.34340507       2.7781462  2.74977492
## Buffalo State College          1.19561483       1.1928286 -0.38288005
## Canton                        -0.52607053      -0.5768283 -0.38288005
##                       Larceny-\ntheft Motor\nvehicle\ntheft      Arson
## Alfred                     -0.3166248             0.8791186 -0.4377975
## Binghamton                  1.8488917            -0.5274711  2.1889876
## Brockport                  -0.2181922             0.8791186 -0.4377975
## Buffalo                     2.6954118             0.8791186 -0.4377975
## Buffalo State College       1.2976693             0.8791186 -0.4377975
## Canton                     -0.5725494            -0.5274711 -0.4377975

library("factoextra")

## Warning: package 'factoextra' was built under R version 4.3.3

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

*Visual inspection of the data*

```
fviz_pca_ind(prcomp(df.scaled), title = "PCA - Police data",
             habillage = "none",  palette = "jco",
```

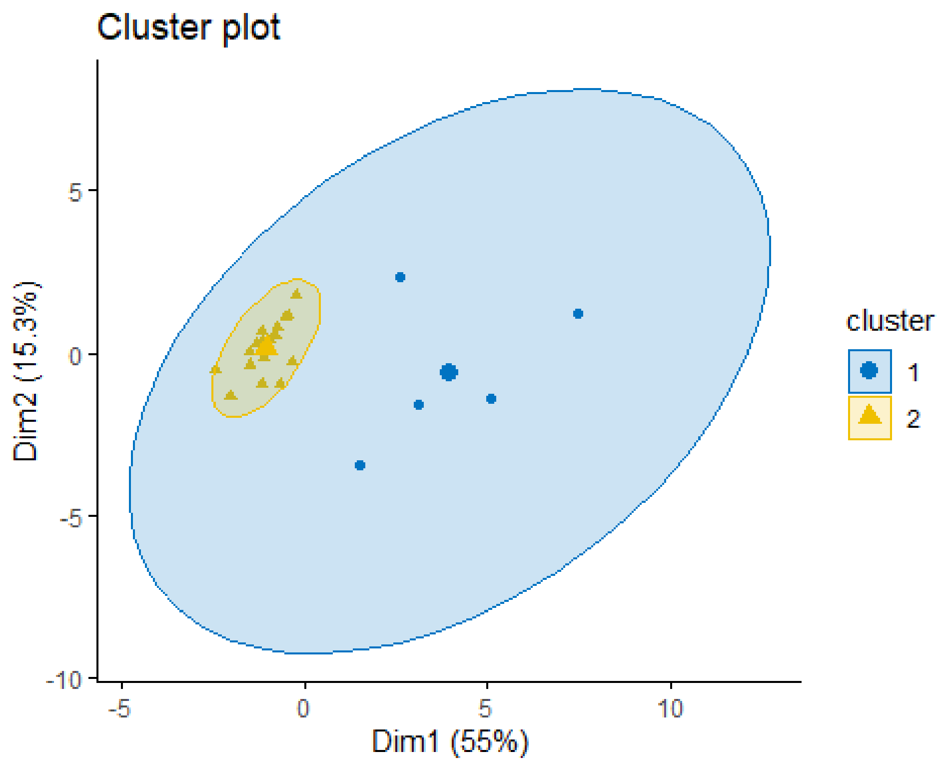
```r
           geom = "point", ggtheme = theme_classic(),
          legend = "bottom")
```




11. **Principal Components**: The first principal component (Dim1) explains 55% of the variance in the data, while the second principal component (Dim2) explains 15.3% of the variance. Together, these two components capture 70.3% of the total variance, indicating that a significant amount of the data's structure can be understood through these two dimensions.

12. **Data Distribution**: The plot shows a cluster of data points around the origin, indicating that most schools have similar characteristics regarding the variables studied. A few data points are outliers, particularly in the positive direction along Dim1, suggesting that some schools have unique characteristics or higher incidences of certain variables compared to the rest.

```r
#library(factoextra)
# K-means on iris dataset
km.res <- kmeans(df.scaled, 2)
fviz_cluster(list(data = df.scaled, cluster = km.res$cluster),
            ellipse.type = "norm", geom = "point", stand = FALSE,
           palette = "jco", ggtheme = theme_classic())
```




The plot displays two distinct clusters in the PCA-reduced space. Cluster 1 (blue circles) is more widely spread, indicating greater variability among its data points. In contrast, Cluster 2 (yellow triangles) is more tightly grouped, suggesting higher similarity among its members.

### *Hopkins Statistic*

We can conduct the Hopkins Statistic test iteratively, using 0.5 as the threshold to reject the alternative hypothesis. That is, if H < 0.5, then it is unlikely that D has statistically significant clusters.

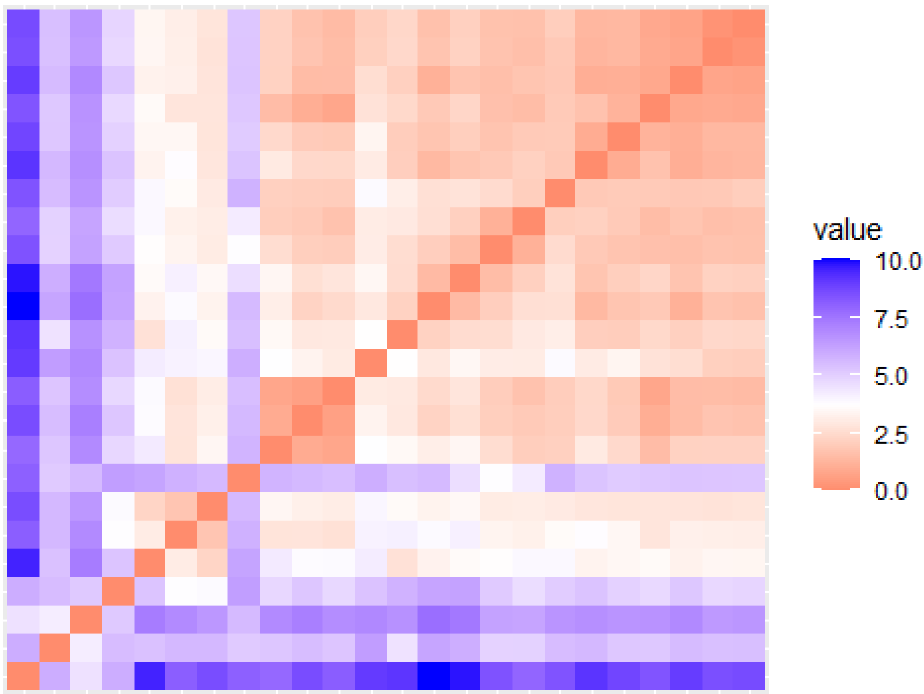
```
library(hopkins)

## Warning: package 'hopkins' was built under R version 4.3.3

hopkins::hopkins(df.scaled)

## [1] 0.9996203
```

### *Visual methods*

Compute the dissimilarity (DM) matrix between the objects in the data set using the Euclidean distance measure

```
fviz_dist(dist(df.scaled), show_labels = FALSE)+
  labs(title = "Crime data")
```

Crime data

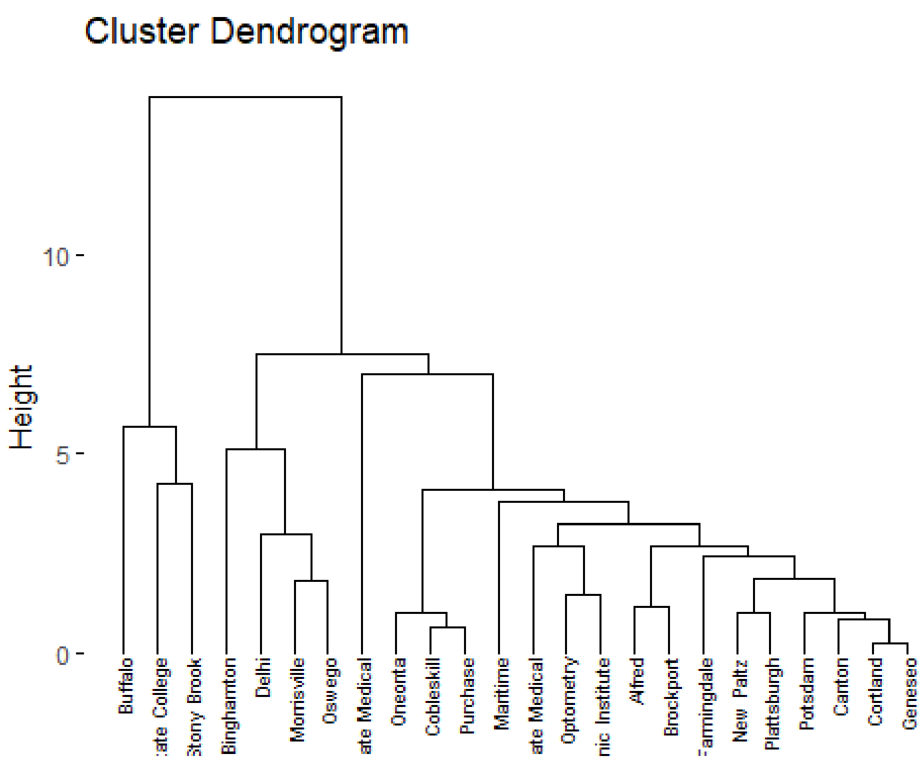



## 4. Hierarchical clustering

```
# Compute the dissimilarity matrix
# df = the standardized data
res.dist <- dist(df.scaled, method = "euclidean")

as.matrix(res.dist)[1:6, 1:6]

##                        Alfred Binghamton Brockport  Buffalo
## Alfred               0.000000   5.185832  1.156390 8.506404
## Binghamton           5.185832   0.000000  4.677971 5.984542
## Brockport            1.156390   4.677971  0.000000 7.991016
## Buffalo              8.506404   5.984542  7.991016 0.000000
## Buffalo State College 4.962247   5.568147  4.973747 5.998421
## Canton               1.606052   5.281922  1.800461 9.085469
##                      Buffalo State College   Canton
## Alfred                            4.962247 1.606052
## Binghamton                        5.568147 5.281922
## Brockport                         4.973747 1.800461
## Buffalo                           5.998421 9.085469
## Buffalo State College             0.000000 5.589255
## Canton                            5.589255 0.000000
```

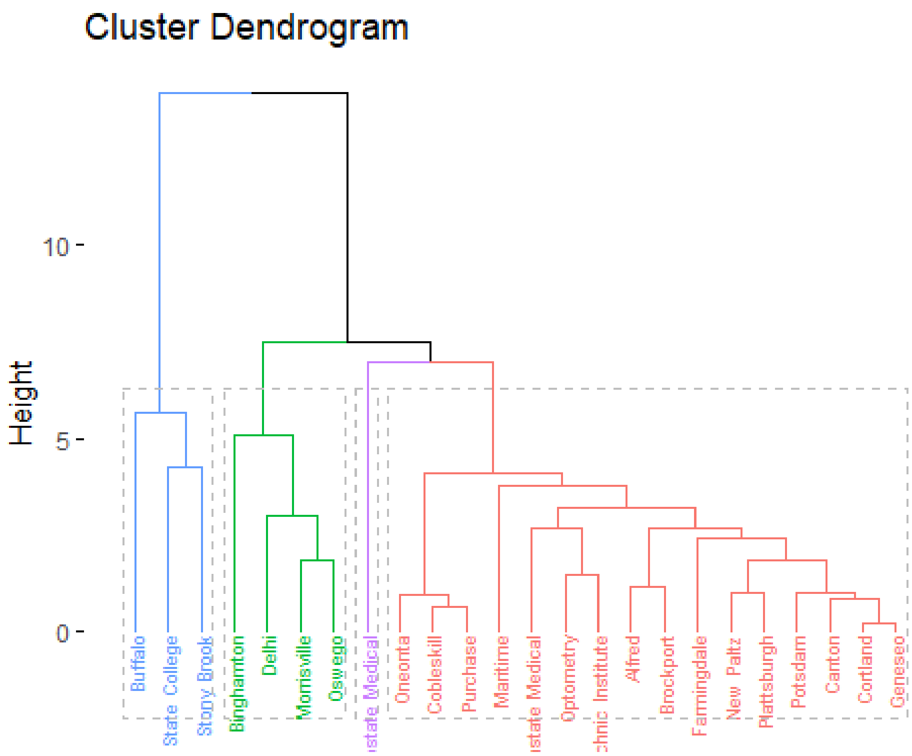
*Ward Linkage*

```r
res.hc <- hclust( d= res.dist, method = "ward.D2")

#  "ward.D2", "single", "complete"

library(factoextra)

fviz_dend(res.hc, cex =0.5)

## Warning: The `<scale>` argument of `guides()` cannot be `FALSE`. Use
"none" instead as
## of ggplot2 3.3.4.
## ℹ The deprecated feature was likely used in the factoextra package.
##   Please report the issue at
<https://github.com/kassambara/factoextra/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```r
# Compute cophentic distance

res.coph <- cophenetic(res.hc)

# Correlation between cophentic distance and the original distance

cor(res.dist, res.coph)
```

```
## [1] 0.8786548

# Cut tree into 4 groups

grp <- cutree(res.hc, k =4)

head(grp, n =4)

##      Alfred Binghamton  Brockport    Buffalo
##           1          2          1          3

# Number of members in each cluster

table(grp)

## grp
##  1  2  3  4
## 16  4  3  1

# Get the names for the members of cluster 1

rownames(df)[grp ==1]

##  [1] "Alfred"                 "Brockport"              "Canton"
##  [4] "Cobleskill"             "Cortland"               "Downstate Medical"
##  [7] "Farmingdale"            "Geneseo"                "Maritime"
## [10] "New Paltz"              "Oneonta"                "Optometry"
## [13] "Plattsburgh"            "Polytechnic Institute" "Potsdam"
## [16] "Purchase"

# Create a data frame with the scaled data and cluster assignments
df_clustered <- data.frame(df.scaled, cluster = factor(grp))

# Define a common color palette
color_palette <- c("#619CFF", "#00BA38", "#C77CFF", "#F8766D")

# Create a named vector for cluster colors
cluster_colors <- color_palette[as.numeric(df_clustered$cluster)]

# Plot the dendrogram with consistent colors
fviz_dend(res.hc, k = 4,
          cex = 0.5,
          k_colors = color_palette,  # Use the defined color palette
          color_labels_by_k = TRUE,
          rect = TRUE)
```

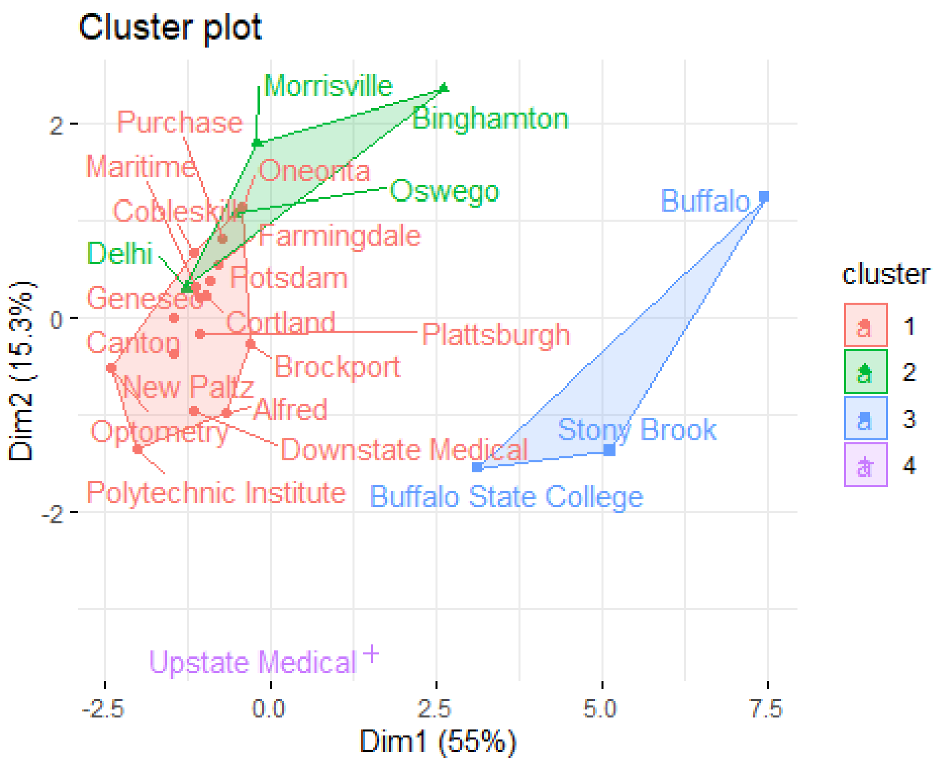



Group 2: Includes Binghamton, Delhi, Morrisville, and Oswego, which are moderately similar to each other.

Group 3: Contains Upstate Medical alone, suggesting it has unique characteristics distinct from the other clusters.

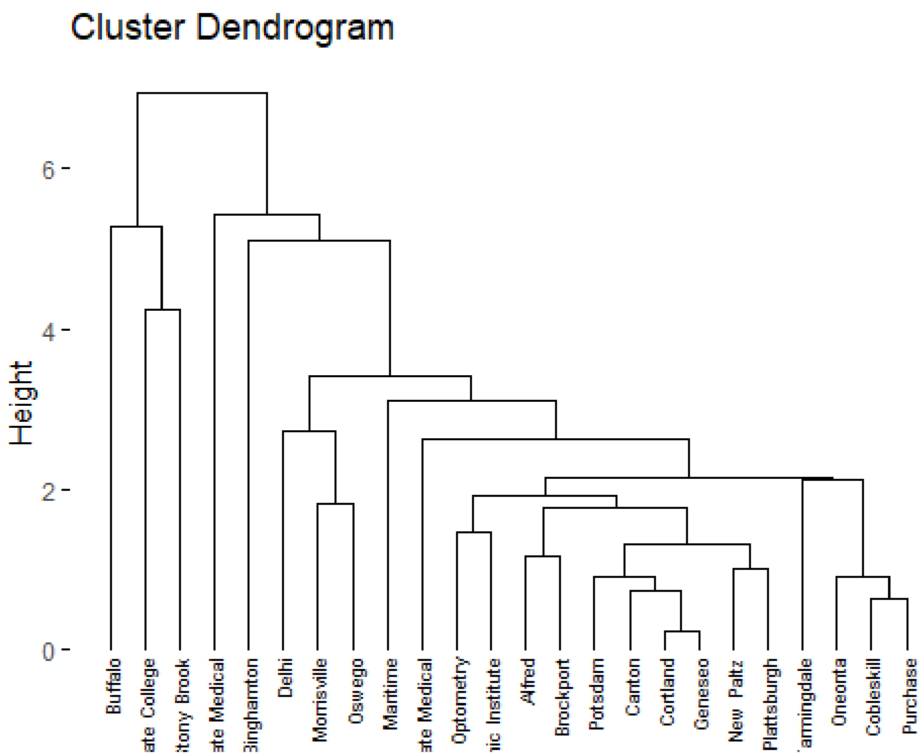
Group 4: Consists of Oneonta, Cobleskill, Purchase, Maritime, State Medical, Optometry, Tech Institute, Alfred, Brockport, Farmingdale, New Paltz, Plattsburgh, Potsdam, Canton, Cortland, and Geneseo, forming a larger and more diverse cluster with greater internal variability.

```
fviz_cluster(list(data = df.scaled, cluster = grp),
             ellipse.type = "convex",
             repel = TRUE,
             show.clust.cent = FALSE,
             ggtheme = theme_minimal(),
             palette = c("#F8766D", "#00BA38", "#619CFF", "#C77CFF"))  # Use
the defined color palette
```

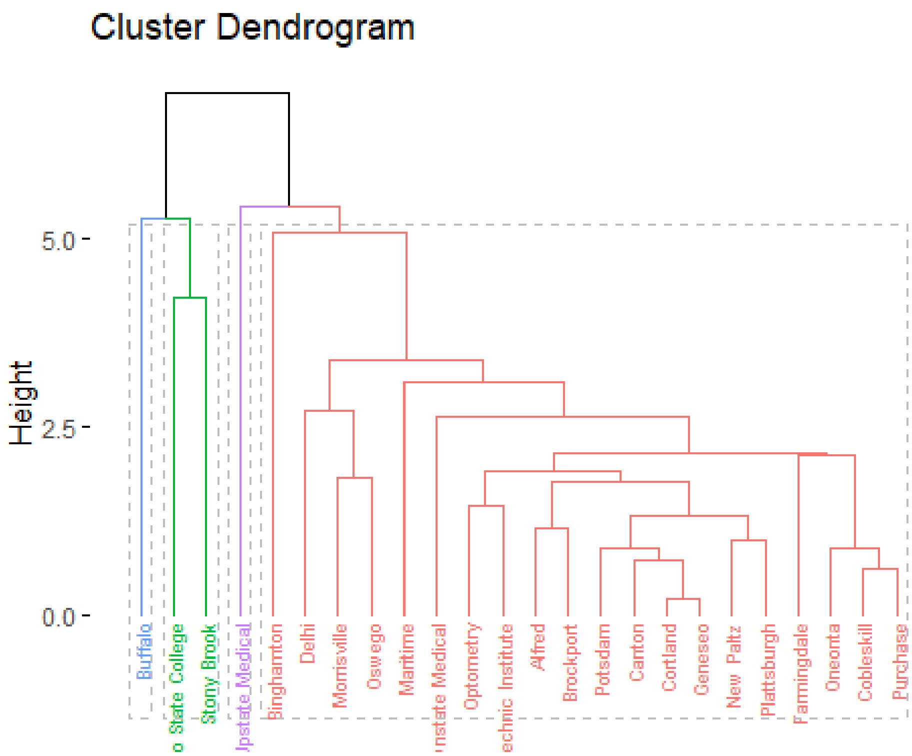



*Average Linkage*

```
res.hc2 <- hclust(res.dist, method = "average")

# Correlation between cophentic distance and the original distance

cor(res.dist, cophenetic(res.hc2))

## [1] 0.9292204

fviz_dend(res.hc2, cex =0.5)
```




```
# Cut in 4 groups and color by groups

fviz_dend(res.hc2, k =4,
          cex =0.5,
          color_labels_by_k = TRUE,
          k_colors = color_palette,
          rect = TRUE)
```



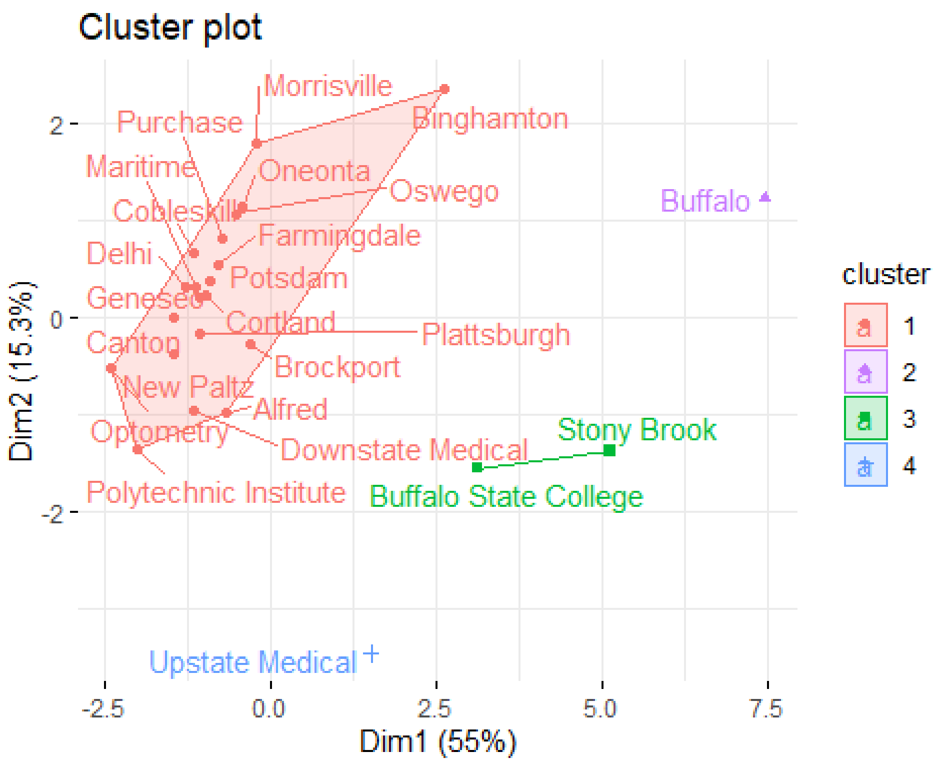

Group 1: Comprises Buffalo alone, indicating it has unique characteristics separating it significantly from the others.

Group 2: Includes State College and Stony Brook, which are moderately similar to each other.

Group 3: Contains Upstate Medical alone, suggesting it has unique characteristics distinct from the other clusters.

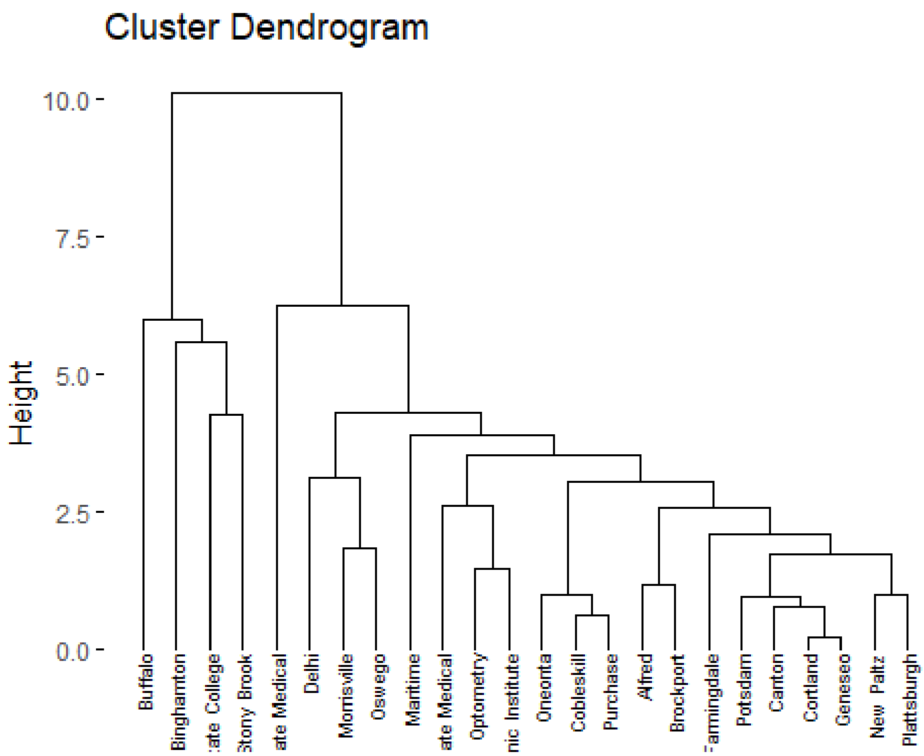
Group 4: Consists of the remaining institutions: Binghamton, Delhi, Morrisville, Oswego, Maritime, Downstate Medical, Optometry, Polytechnic Institute, Alfred, Brockport, Farmingdale, New Paltz, Plattsburgh, Potsdam, Canton, Cortland, Geneseo, Oneonta, Cobleskill, and Purchase. This larger cluster indicates greater internal variability and includes institutions with more similar characteristics within this group.

```r
fviz_cluster(list(data = df.scaled, cluster = cutree(res.hc2, k =4)),
             ellipse.type = "convex",
             repel = TRUE,
             show.clust.cent = FALSE,
             palette = c("#F8766D","#C77CFF", "#00BA38", "#619CFF"),
             ggtheme = theme_minimal())
```

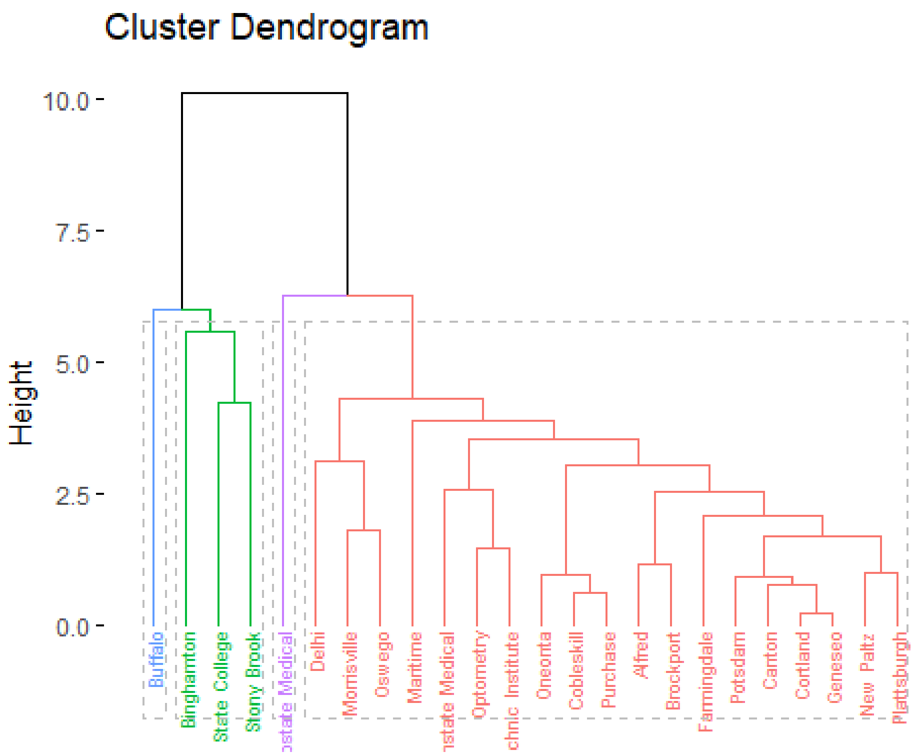



*Complete Linkage*

```
res.hc3 <- hclust(res.dist, method = "complete")

# Correlation between cophentic distance and the original distance

cor(res.dist, cophenetic(res.hc3))

## [1] 0.8890733

fviz_dend(res.hc3, cex =0.5)
```




```
# Cut in 4 groups and color by groups

fviz_dend(res.hc3, k =4,
          cex =0.5,
          color_labels_by_k = TRUE,
          k_colors = color_palette,
          rect = TRUE)
```



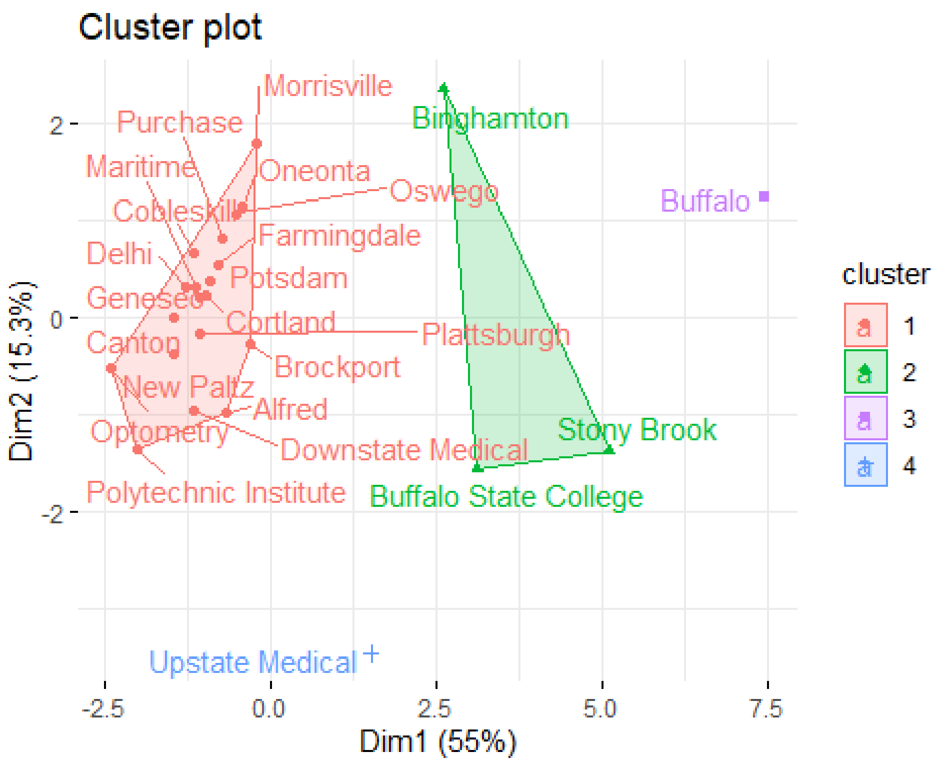

Group 1: Comprises Buffalo alone, indicating it has unique characteristics separating it significantly from the others.

Group 2: Includes State College, Binghamton, and Stony Brook, indicating these three institutions have moderately similar characteristics.

Group 3: Contains Upstate Medical alone, suggesting it has unique characteristics distinct from the other clusters.

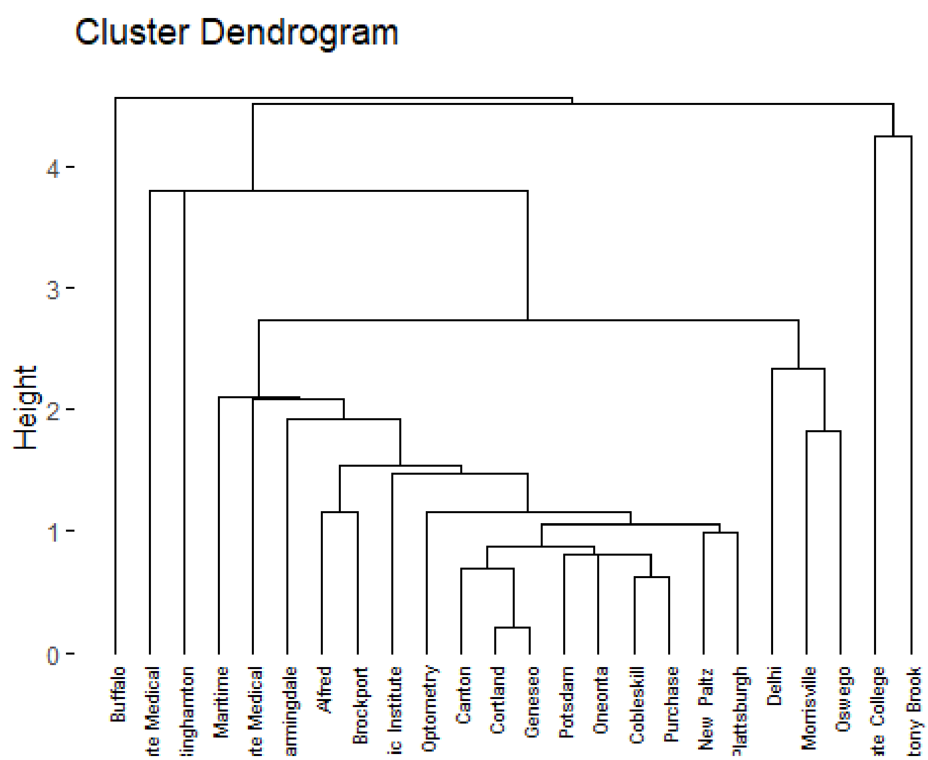
Group 4: Consists of the remaining institutions: Delhi, Morrisville, Oswego, Maritime, Downstate Medical, Optometry, Polytechnic Institute, Alfred, Brockport, Farmingdale, New Paltz, Plattsburgh, Potsdam, Canton, Cortland, Geneseo, Oneonta, Cobleskill, and Purchase. This larger cluster indicates greater internal variability and includes institutions with more similar characteristics within this group.

```
fviz_cluster(list(data = df.scaled, cluster = cutree(res.hc3, k =4)),
             ellipse.type = "convex",
             repel = TRUE,
             show.clust.cent = FALSE,
             palette = c("#F8766D","#00BA38","#C77CFF", "#619CFF"),
             ggtheme = theme_minimal())
```

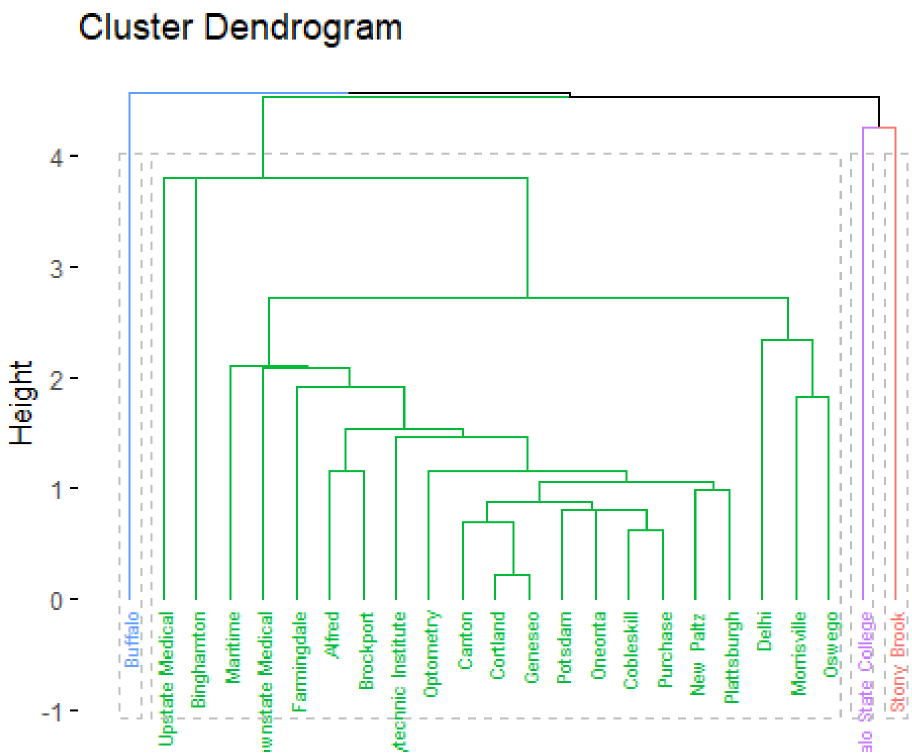



*Single Linkage*

```
res.hc4 <- hclust(res.dist, method = "single")

# Correlation between cophentic distance and the original distance

cor(res.dist, cophenetic(res.hc4))

## [1] 0.9121947

fviz_dend(res.hc4, cex =0.5)
```




```r
# Cut in 4 groups and color by groups

fviz_dend(res.hc4, k =4,
          cex =0.5,
          color_labels_by_k = TRUE,
          k_colors = color_palette,
          rect = TRUE)
```

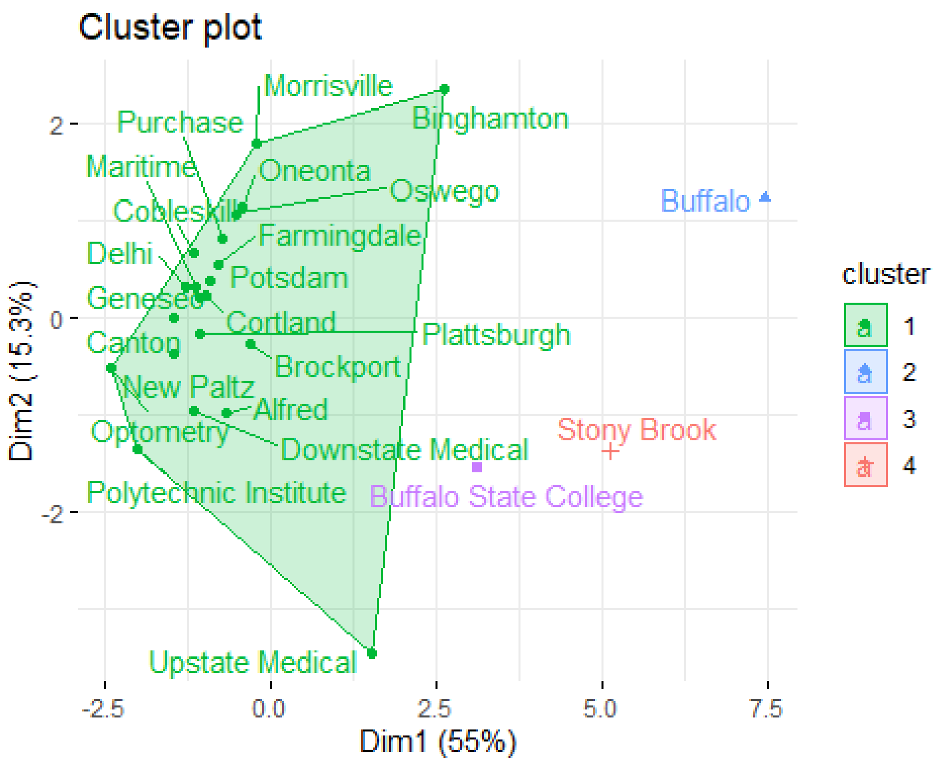



Group 2: Includes Binghamton, Maritime, Downstate Medical, Farmingdale, Alfred, Brockport, Polytechnic Institute, Optometry, Canton, Cortland, Geneseo, Potsdam, Oneonta, Cobleskill, Purchase, New Paltz, Plattsburgh, Delhi, Morrisville, and Oswego. This large group indicates a high level of internal variability and includes institutions with similar characteristics.

Group 3: Contains Buffalo State College alone, suggesting it has unique characteristics distinct from the other clusters.

Group 4: Includes only Stony Brook, indicating it has distinct characteristics separating it from the others.

```r
fviz_cluster(list(data = df.scaled, cluster = cutree(res.hc4, k =4)),
             ellipse.type = "convex",
             repel = TRUE,
             show.clust.cent = FALSE,
             palette = c("#00BA38","#619CFF","#C77CFF","#F8766D"),
             ggtheme = theme_minimal())
```



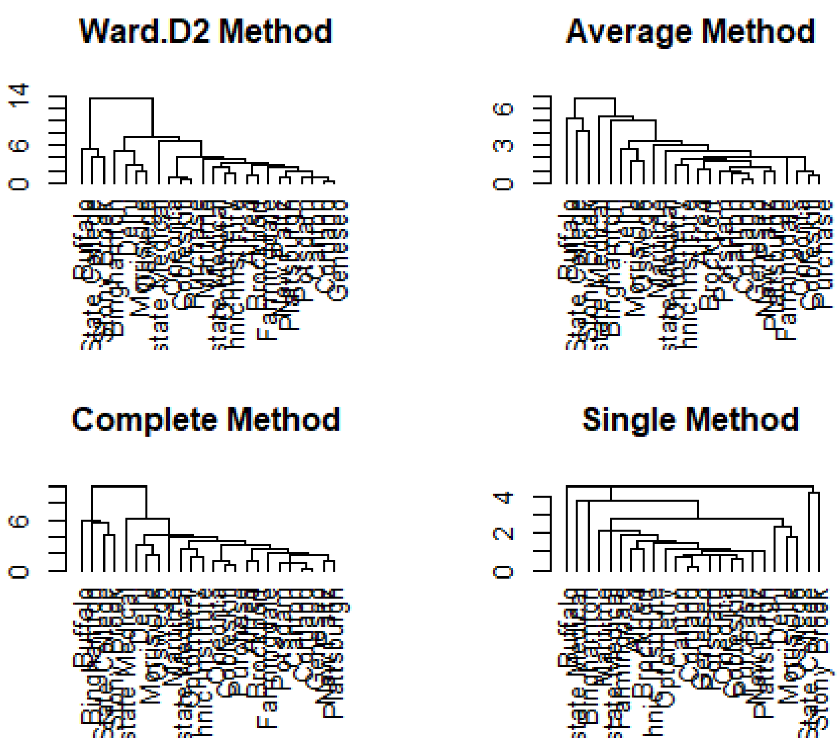

## 5. Compare dendrograms

*Visual Comparison of two dendrograms*

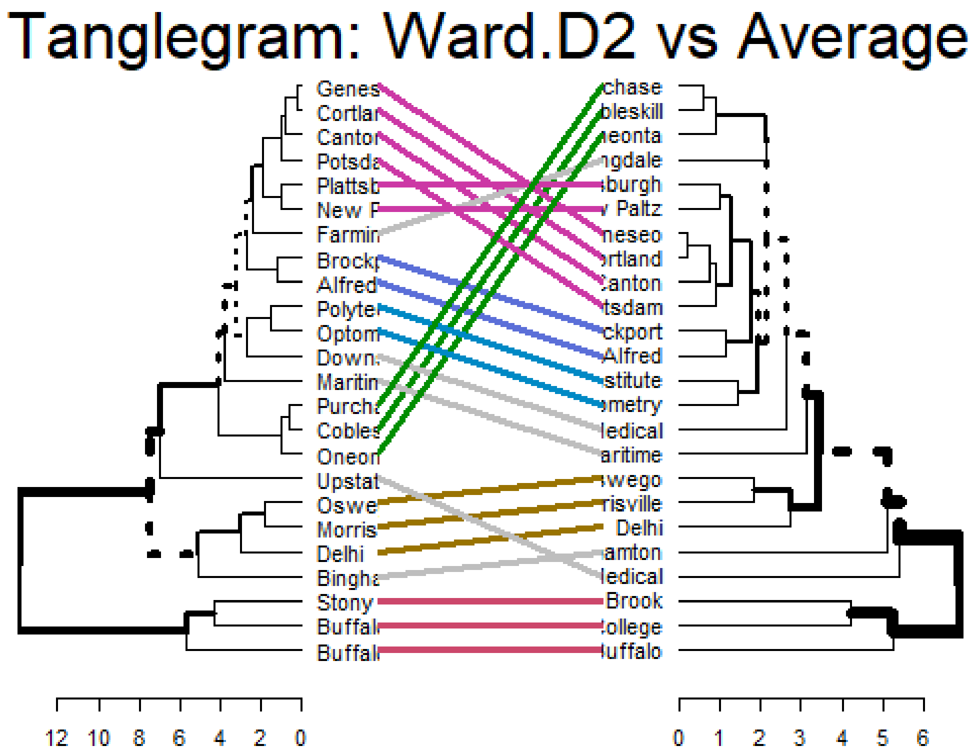
```
library(dendextend)

## Warning: package 'dendextend' was built under R version 4.3.3

##
## ---------------------
## Welcome to dendextend version 1.17.1
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at:
https://github.com/talgalili/dendextend/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:
suppressPackageStartupMessages(library(dendextend))
## ---------------------

##
## Attaching package: 'dendextend'
```

```
## The following object is masked from 'package:stats':
##
##     cutree
```

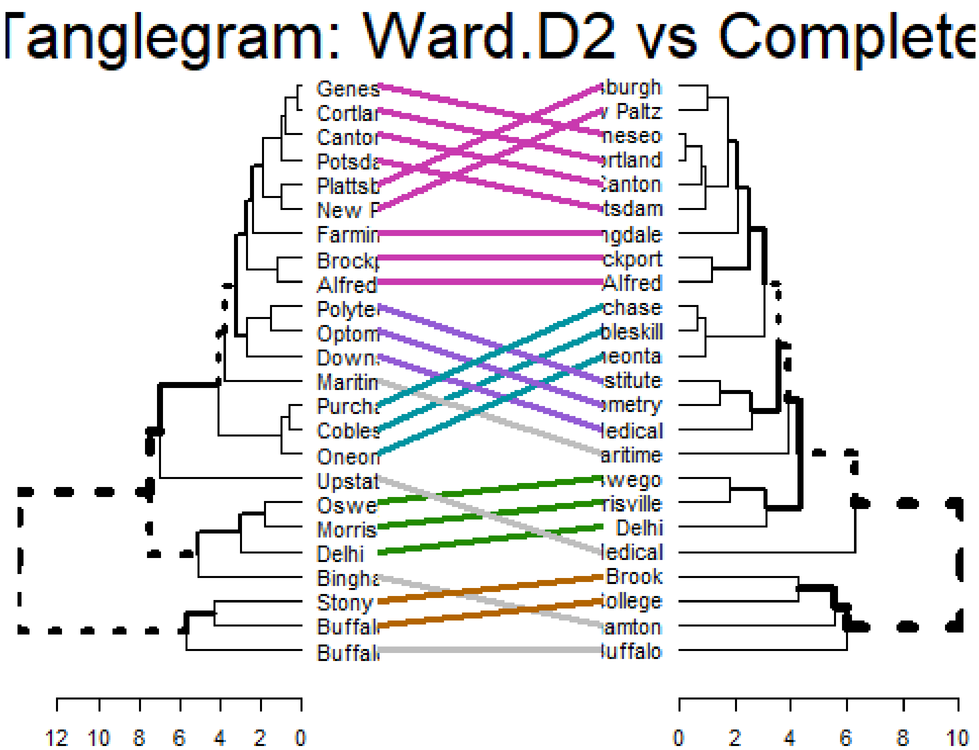
```
# Create two dendrograms
dend1 <- as.dendrogram (res.hc) # Ward.D2
dend2 <- as.dendrogram (res.hc2) # Average
dend3 <- as.dendrogram (res.hc3) # Comple
dend4 <- as.dendrogram (res.hc4) # Single

# Plot dendrograms
par(mfrow = c(2, 2))  # Set layout for multiple plots

# Plot each dendrogram
plot(dend1, main = "Ward.D2 Method")
plot(dend2, main = "Average Method")
plot(dend3, main = "Complete Method")
plot(dend4, main = "Single Method")
```



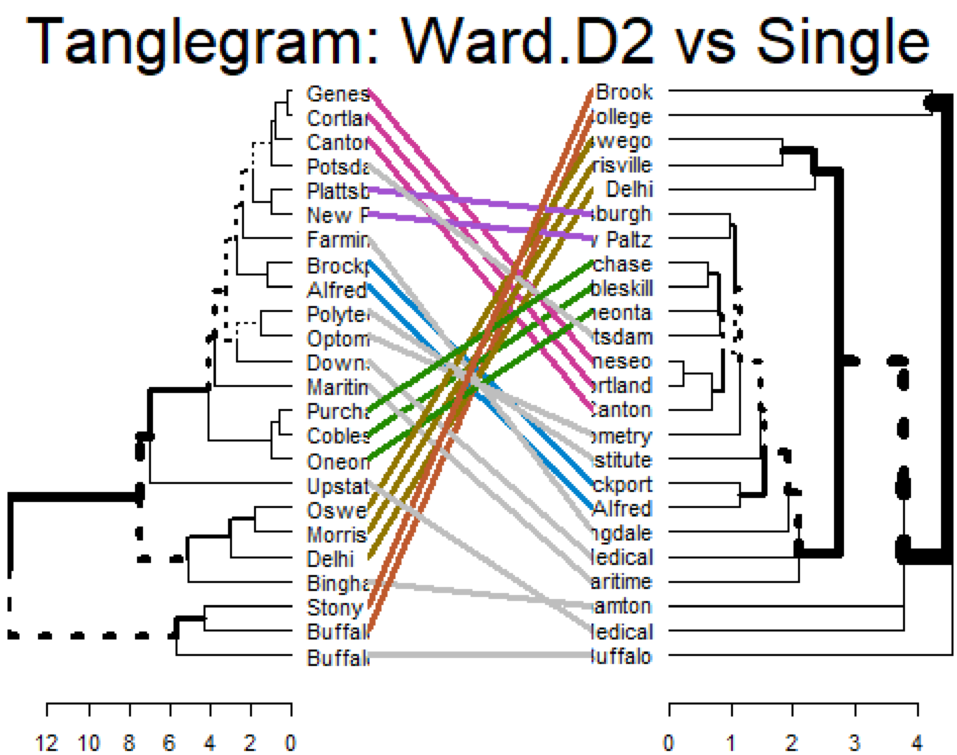

```
# Create tanglegrams and compute entanglement
tanglegram(dend1, dend2, main = "Tanglegram: Ward.D2 vs Average")
```



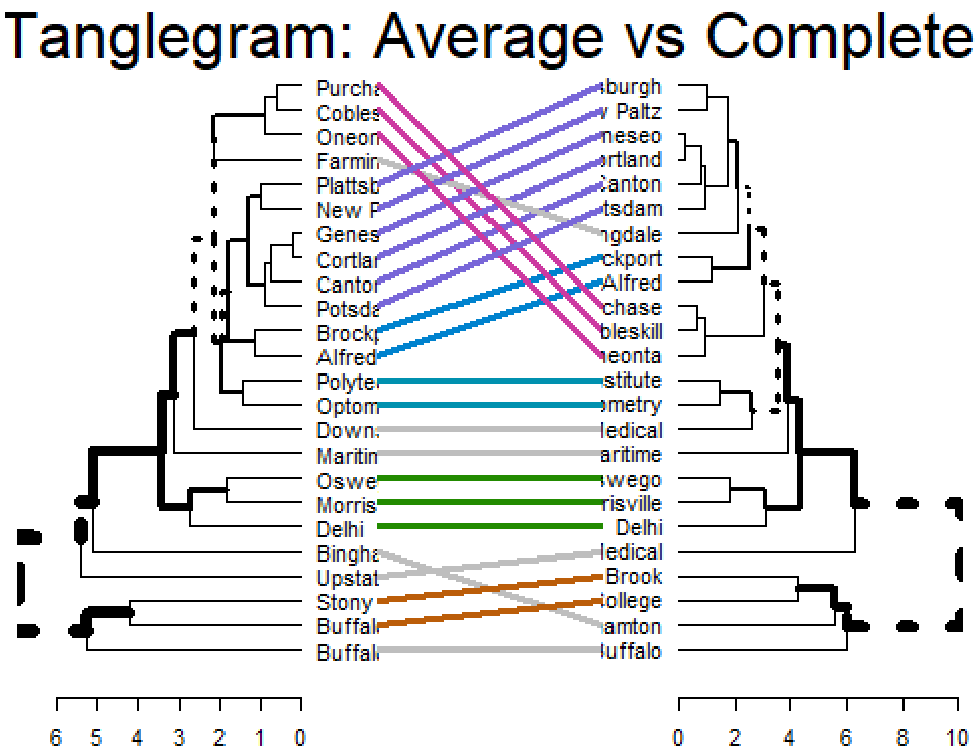

```r
entanglement_value <- entanglement(dend1, dend2)
print(paste("Entanglement between Ward.D2 and Average:", entanglement_value))

## [1] "Entanglement between Ward.D2 and Average: 0.21973997966606"

tanglegram(dend1, dend3, main = "Tanglegram: Ward.D2 vs Complete")
```



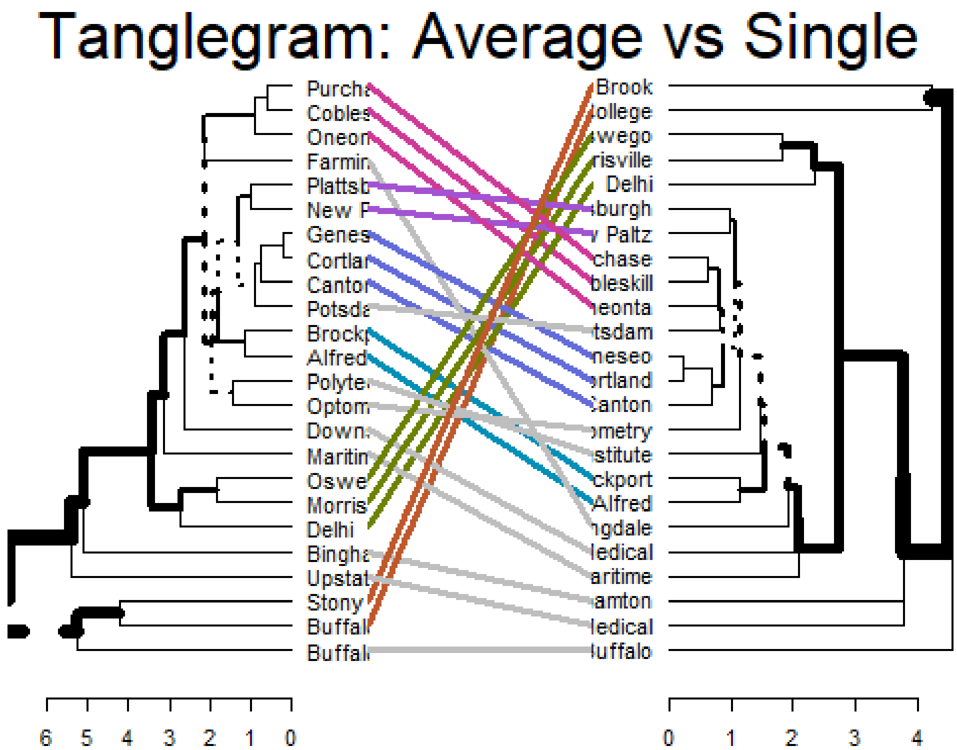

```r
entanglement_value <- entanglement(dend1, dend3)
print(paste("Entanglement between Ward.D2 and Complete:",
entanglement_value))

## [1] "Entanglement between Ward.D2 and Complete: 0.0754907825219036"

tanglegram(dend1, dend4, main = "Tanglegram: Ward.D2 vs Single")
```



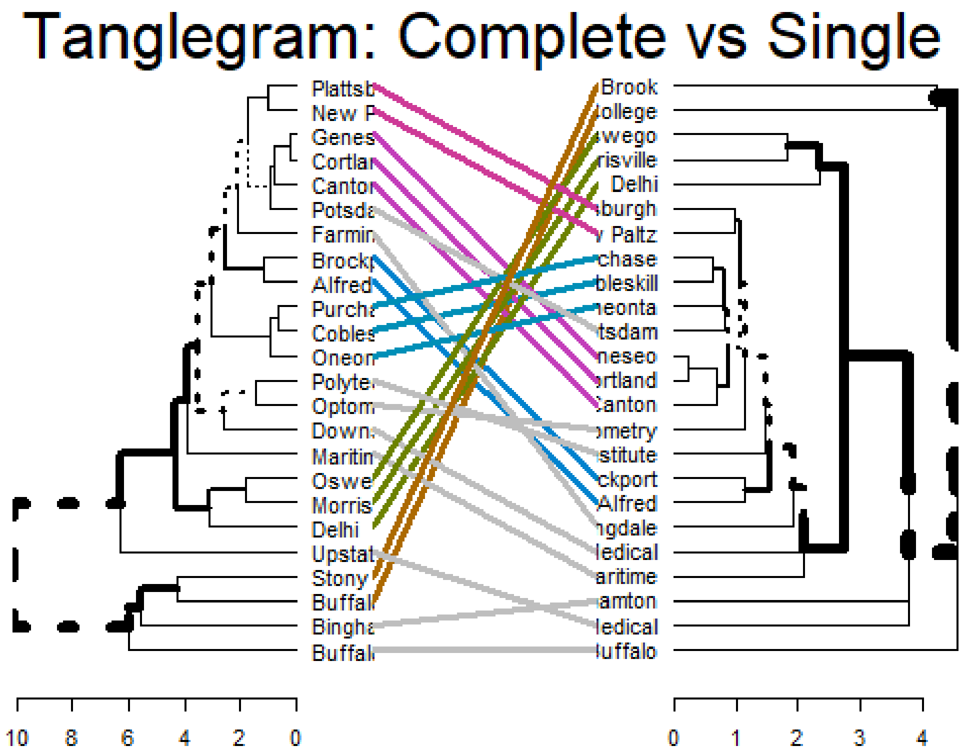

```r
entanglement_value <- entanglement(dend1, dend4)
print(paste("Entanglement between Ward.D2 and Single:", entanglement_value))

## [1] "Entanglement between Ward.D2 and Single: 0.63855476683203"

tanglegram(dend2, dend3, main = "Tanglegram: Average vs Complete")
```




```
entanglement_value <- entanglement(dend2, dend3)
print(paste("Entanglement between Average and Complete:",
entanglement_value))

## [1] "Entanglement between Average and Complete: 0.135496206654804"

tanglegram(dend2, dend4, main = "Tanglegram: Average vs Single")
```




```r
entanglement_value <- entanglement(dend2, dend4)
print(paste("Entanglement between Average and Single:", entanglement_value))

## [1] "Entanglement between Average and Single: 0.499667671863545"

tanglegram(dend3, dend4, main = "Tanglegram: Complete vs Single")
```




```r
entanglement_value <- entanglement(dend3, dend4)
print(paste("Entanglement between Complete and Single:", entanglement_value))

## [1] "Entanglement between Complete and Single: 0.522691624001376"
```

Based on the entanglement values calculated between different pairs of dendrograms, we can derive the following insights:

```
Ward.D2 vs Average (Entanglement: 0.22):
    This indicates a relatively low level of entanglement, suggesting that
the Ward.D2 and Average linkage methods produce somewhat similar clustering
structures.

Ward.D2 vs Complete (Entanglement: 0.08):
    This very low entanglement value indicates that the clustering structures
produced by the Ward.D2 and Complete linkage methods are very similar.

Ward.D2 vs Single (Entanglement: 0.64):
    This higher entanglement value suggests significant differences in the
clustering structures produced by the Ward.D2 and Single linkage methods.

Average vs Complete (Entanglement: 0.14):
    This indicates a low level of entanglement, suggesting that the Average
and Complete linkage methods produce similar clustering structures.

Average vs Single (Entanglement: 0.50):
    This moderate entanglement value indicates noticeable differences between
```

```
the clustering structures produced by the Average and Single linkage methods.

Complete vs Single (Entanglement: 0.52):
    This moderate entanglement value suggests that the clustering structures
produced by the Complete and Single linkage methods are different.
```

Summary:

- Most Similar Methods: Ward.D2 and Complete linkage methods show the most similar clustering structures with an entanglement of 0.08.
- Most Different Methods: Ward.D2 and Single linkage methods show the most different clustering structures with an entanglement of 0.64.
- Overall Trends: Methods like Ward.D2 and Complete, as well as Average and Complete, tend to produce similar clustering results, while the Single linkage method tends to produce clustering results that are more different from the other methods.

### *Correlation matrix between a list of dendrograms*

```r
# Compute cophenetic correlation matrix
dend_list <- dendlist(dend1, dend2, dend3, dend4)
coph_cor <- cor.dendlist(dend_list, method = "cophenetic")
print("Cophenetic correlation matrix",coph_cor)

## [1] "Cophenetic correlation matrix"

print(coph_cor)

##           [,1]      [,2]      [,3]      [,4]
## [1,] 1.0000000 0.9419843 0.8785501 0.8993881
## [2,] 0.9419843 1.0000000 0.9420799 0.9743336
## [3,] 0.8785501 0.9420799 1.0000000 0.9150298
## [4,] 0.8993881 0.9743336 0.9150298 1.0000000
```

The highest correlation (0.974) is between the dendrograms created by the Average and Single linkage methods, indicating they produce very similar clustering structures.

The lowest correlation (0.878) is between the dendrograms created by the Ward.D2 and Complete linkage methods, suggesting these methods produce somewhat different clustering structures compared to the other pairings.

```r
# Compute Baker correlation matrix
baker_cor <- cor.dendlist(dend_list, method = "baker")
print("Baker correlation matrix", baker_cor)

## [1] "Baker correlation matrix"

print(baker_cor)

##           [,1]      [,2]      [,3]      [,4]
## [1,] 1.0000000 0.9258940 0.9089954 0.8697014
## [2,] 0.9258940 1.0000000 0.9531899 0.9653639
```

```
## [3,] 0.9089954 0.9531899 1.0000000 0.9279202
## [4,] 0.8697014 0.9653639 0.9279202 1.0000000
```

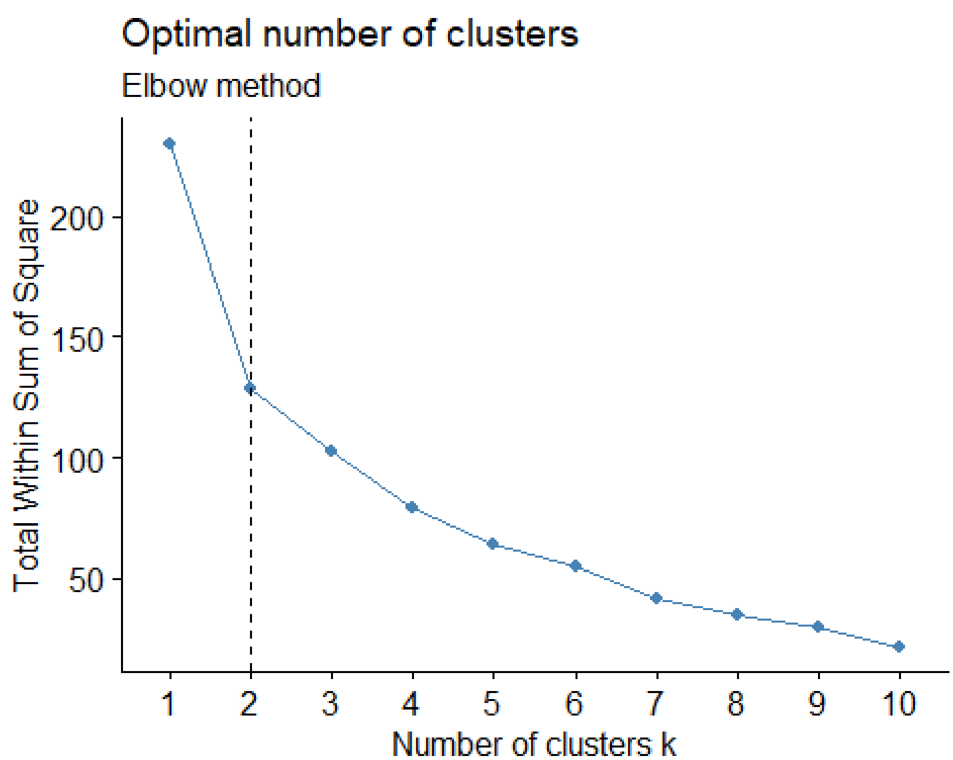
The highest correlation (0.965) is between the Average and Single linkage methods, consistent with the cophenetic correlation matrix.

The lowest correlation (0.870) is between the Ward.D2 and Single linkage methods, indicating some difference in clustering structures between these methods, but still relatively high correlation.

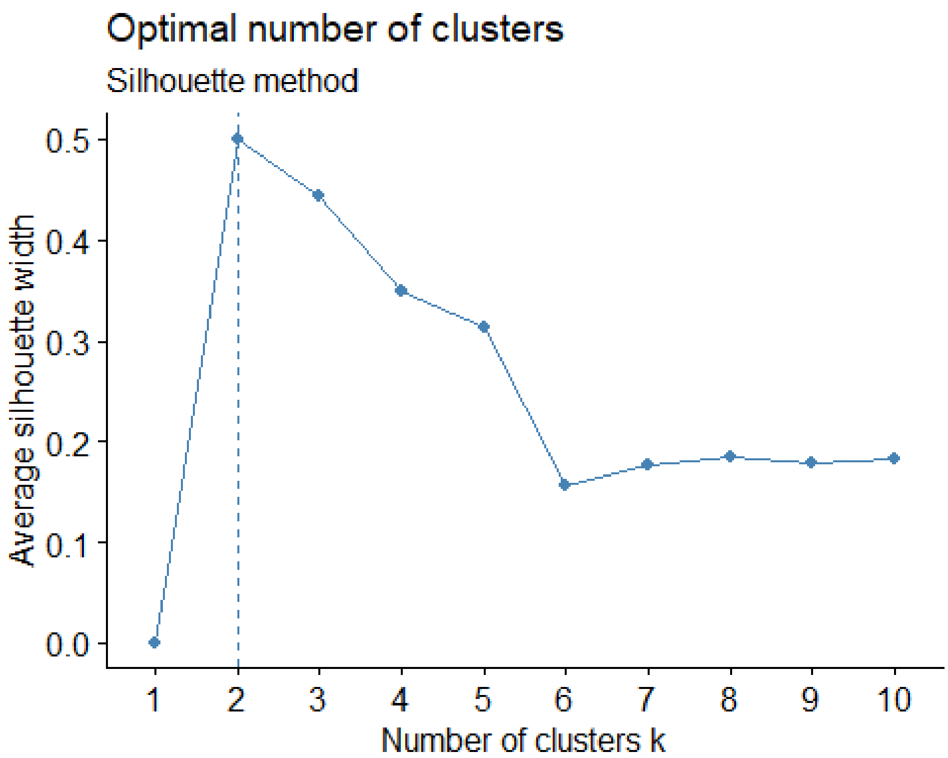
## 6. Choosing the best number of clusters

*Elbow method*

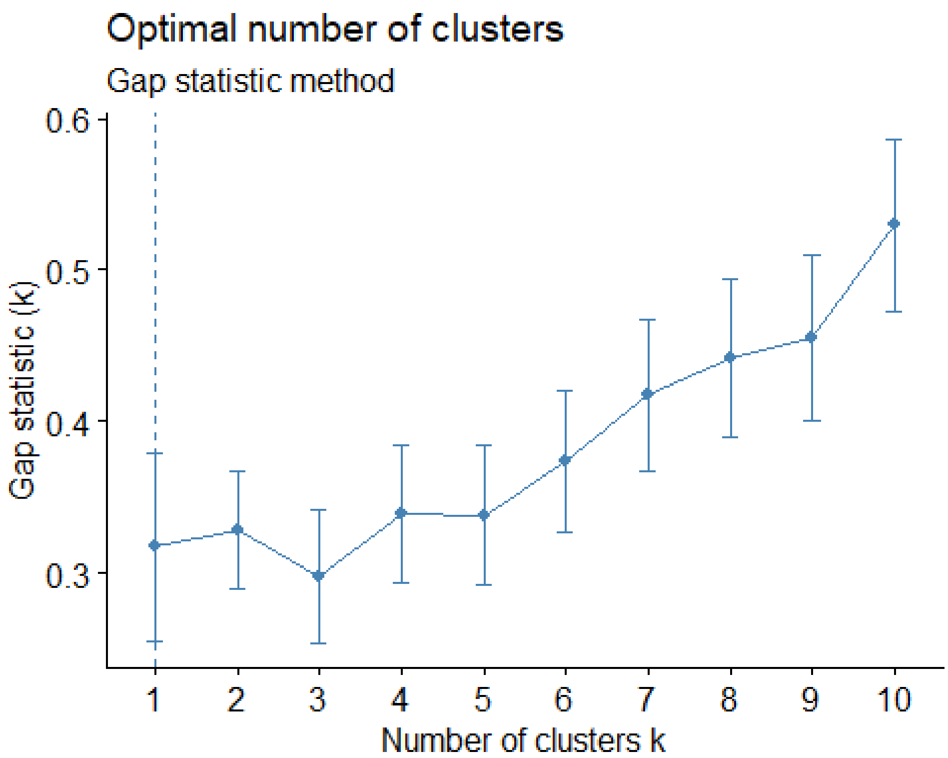
```
fviz_nbclust(df.scaled, kmeans, iter.max = 10, nstart = 25, method = "wss") +
    geom_vline(xintercept = 2, linetype = 2)+
  labs(subtitle = "Elbow method")
```




```
fviz_nbclust(df.scaled, kmeans, iter.max = 10, nstart = 25, method =
"silhouette")+
  labs(subtitle = "Silhouette method")
```




```
# nboot = 50 to keep the function speedy.
# recommended value: nboot= 500 for your analysis.
# Use verbose = FALSE to hide computing progression.
set.seed(123)
fviz_nbclust(df.scaled, kmeans,iter.max = 10, nstart = 25,  method =
"gap_stat", nboot = 50)+
  labs(subtitle = "Gap statistic method")
```




- Elbow method: 2 clusters solution suggested
- Silhouette method: 2 clusters solution suggested
- Gap statistic method: 1 clusters solution suggested

## 7. Clustering validation

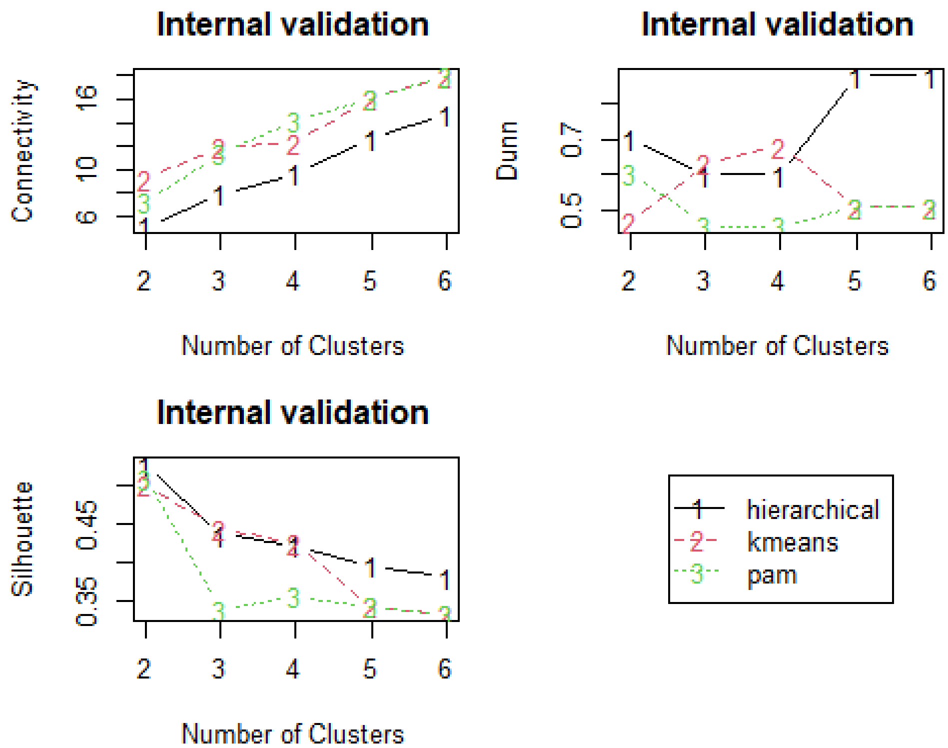
```
library(clValid)

## Warning: package 'clValid' was built under R version 4.3.3

## Loading required package: cluster

# Compute clValid
clmethods <- c("hierarchical","kmeans","pam")

intern <- clValid(df.scaled, nClust = 2:6,
              clMethods = clmethods, validation = "internal")
# Summary
summary(intern)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
```

```
##  2 3 4 5 6
##
## Validation Measures:
##                                 2       3       4       5       6
##
## hierarchical Connectivity   5.1536  7.8825  9.7159 12.6448 14.6448
##              Dunn           0.7050  0.6018  0.6018  0.8808  0.8808
##              Silhouette     0.5281  0.4361  0.4212  0.3954  0.3802
## kmeans       Connectivity   9.1615 11.9115 12.3115 16.0317 18.0317
##              Dunn           0.4663  0.6332  0.6806  0.5106  0.5106
##              Silhouette     0.5001  0.4439  0.4237  0.3419  0.3314
## pam          Connectivity   7.2492 11.4694 14.1984 16.0317 18.0317
##              Dunn           0.6060  0.4549  0.4549  0.5106  0.5106
##              Silhouette     0.5103  0.3379  0.3555  0.3419  0.3314
##
## Optimal Scores:
##
##              Score  Method       Clusters
## Connectivity 5.1536 hierarchical 2
## Dunn         0.8808 hierarchical 5
## Silhouette   0.5281 hierarchical 2
```

It can be seen that hierarchical clustering with two clusters performs the best in each case for connectivity and Silhouette measures).

However, the number of clusters is appropriate based on the Dunn index is 5 and 6.

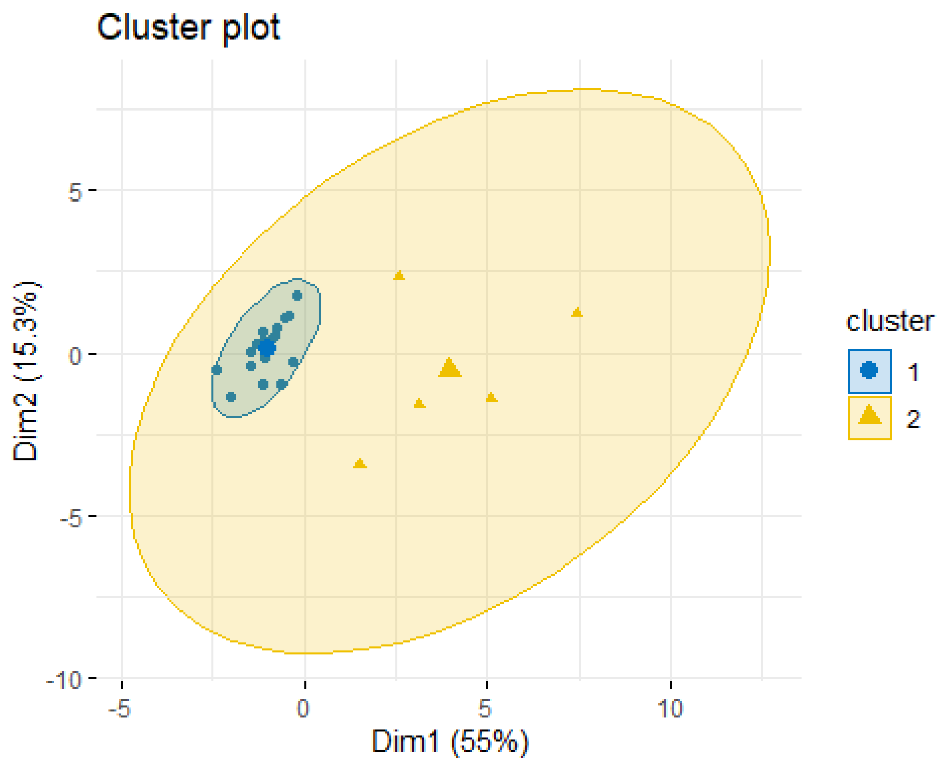
```
op <- par(no.readonly=TRUE)
par(mfrow=c(2,2), mar=c(4,4,3,1))
plot(intern, legend=FALSE)
plot(nClusters(intern), measures(intern, "Dunn")[,,1], type="n", axes=FALSE,
xlab="", ylab="")
legend("center", clusterMethods(intern), col=1:9, lty=1:9, pch=paste(1:9))
```



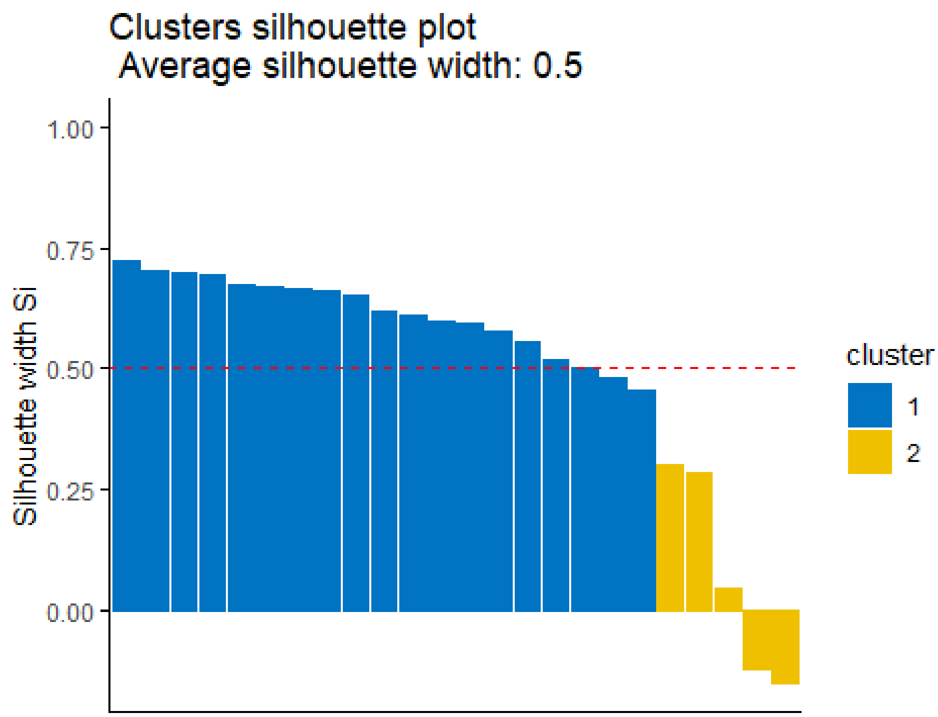

```r
par(op)
```

*Visualization:*

```r
# K-means clustering
km.res1 <- eclust(df.scaled, "kmeans", k = 2, nstart = 25, graph = FALSE)
# Visualize k-means clusters
fviz_cluster(km.res1, geom = "point", ellipse.type = "norm",
             palette = "jco", ggtheme = theme_minimal())
```




```
fviz_silhouette(km.res1, palette = "jco",
                ggtheme = theme_classic())

##   cluster size ave.sil.width
## 1       1   19          0.61
## 2       2    5          0.07
```




- A value of Si close to 1 indicates that the object is well clustered. In the other words, the object i is similar to the other objects in its group.
- A value of Si close to -1 indicates that the object is poorly clustered, and that assignment to some other cluster would probably improve the overall results.

```
# Silhouette information
silinfo <- km.res1$silinfo
 names(silinfo)

## [1] "widths"          "clus.avg.widths" "avg.width"

# Silhouette widths of each observation
 head(silinfo$widths[, 1:3], 10)

##                        cluster neighbor sil_width
## Canton                       1        2 0.7225224
## Geneseo                      1        2 0.7016625
## Potsdam                      1        2 0.6982284
## Cortland                     1        2 0.6961125
## Optometry                    1        2 0.6719168
## New Paltz                    1        2 0.6674752
## Plattsburgh                  1        2 0.6629632
## Cobleskill                   1        2 0.6610889
## Purchase                     1        2 0.6533439
## Polytechnic Institute        1        2 0.6209247
```

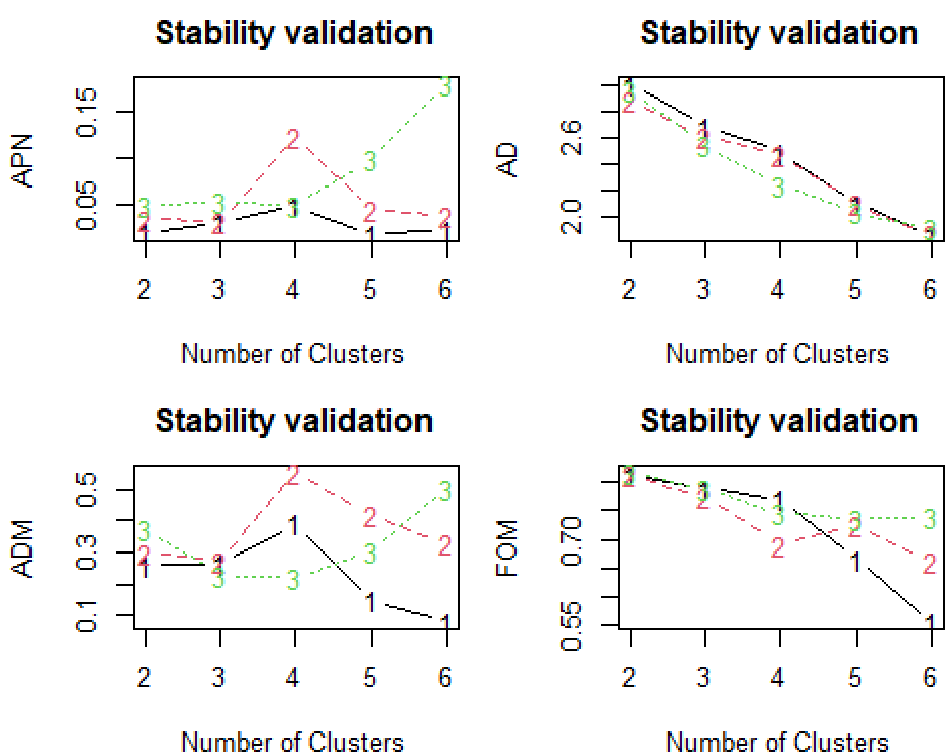
```r
# Silhouette width of observation
sil <- km.res1$silinfo$widths[, 1:3]

# Objects with negative silhouette
neg_sil_index <- which(sil[, 'sil_width'] < 0)
sil[neg_sil_index, , drop = FALSE]

##                  cluster neighbor  sil_width
## Binghamton             2        1 -0.1210743
## Upstate Medical        2        1 -0.1509450

# Average silhouette width of each cluster
silinfo$clus.avg.widths

## [1] 0.61278715 0.07177466

# The total average (mean of all individual silhouette widths)
 silinfo$avg.width

## [1] 0.5000762
```

## 8. Stability Measures

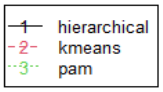
```r
# Stability measures
clmethods <- c("hierarchical","kmeans","pam")
stab <- clValid(df.scaled, nClust = 2:6, clMethods = clmethods,
                validation = "stability")

# Display only optimal Scores
summary(stab)

##
## Clustering Methods:
##  hierarchical kmeans pam
##
## Cluster sizes:
##  2 3 4 5 6
##
## Validation Measures:
##                          2      3      4      5      6
##
## hierarchical APN  0.0190 0.0301 0.0475 0.0167 0.0211
##              AD   3.0111 2.6853 2.4883 2.0990 1.8581
##              ADM  0.2584 0.2623 0.3909 0.1511 0.0765
##              FOM  0.8130 0.7892 0.7695 0.6668 0.5557
## kmeans       APN  0.0333 0.0306 0.1231 0.0437 0.0375
##              AD   2.8881 2.6066 2.4602 2.0894 1.8570
##              ADM  0.2969 0.2740 0.5552 0.4189 0.3271
##              FOM  0.8131 0.7721 0.6899 0.7230 0.6609
## pam          APN  0.0471 0.0534 0.0481 0.0990 0.1807
##              AD   2.9553 2.5384 2.2278 2.0254 1.9062
```

```
##              ADM  0.3746 0.2248 0.2202 0.3045 0.5038
##              FOM  0.8164 0.7866 0.7440 0.7363 0.7390
##
## Optimal Scores:
##
##     Score  Method       Clusters
## APN 0.0167 hierarchical 5
## AD  1.8570 kmeans       6
## ADM 0.0765 hierarchical 6
## FOM 0.5557 hierarchical 6
```

```
par(mfrow=c(2,2), mar=c(4,4,3,1))
plot(stab, measure=c("APN", "AD", "ADM", "FOM"), legend=FALSE)
```




```
plot(nClusters(stab), measures(stab, "APN")[,,1], type="n", axes=FALSE,
xlab="", ylab="")
legend("center", clusterMethods(stab), col=1:9, lty=1:9, pch=paste(1:9))
```




## 9. DBSCAN

```
# Load necessary library
library(dbscan)

## Warning: package 'dbscan' was built under R version 4.3.3

##
## Attaching package: 'dbscan'

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

**EPS** defines the radius of the neighborhood around a point. If the distance between two points is less than or equal to eps, they are considered neighbors. This neighborhood concept is crucial for determining core points, border points, and noise points:

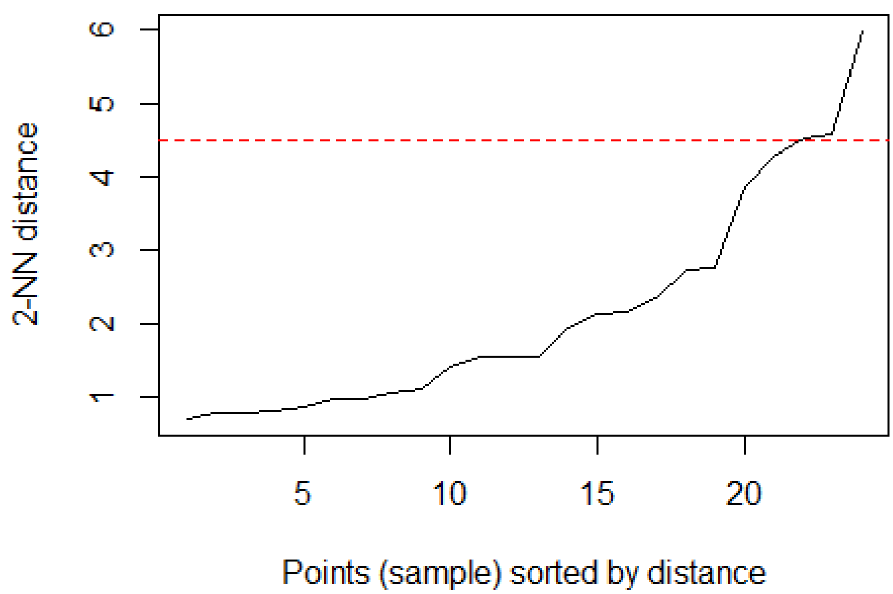
**Core Point**: A point that has at least minPts neighbors within a distance eps. **Border Point**: A point that has fewer than minPts neighbors but lies within the eps distance of a core point. **Noise Point**: A point that is neither a core point nor a border point.

#The k-distance plot is a method to help choose a suitable eps value:

k-distance: For each point, calculate the distance to its k-th nearest neighbor. Typically, k is set to minPts - 1. Sort and Plot: Sort these k-distances in ascending order and plot them. The “elbow” point in this plot, where the k-distance sharply increases, helps to identify a suitable eps. This point indicates the transition from dense to sparse regions.

#Why We Choose eps This Way: **Density Definition**: The k-distance plot helps us visualize the density of points. The “elbow” point represents the maximum distance where most points are still within a dense region. **Avoiding Noise**: By choosing eps at the elbow, we ensure that points within this distance are considered part of a cluster, while points beyond this distance are treated as noise or separate clusters. **Cluster Separation*: This method helps in separating clusters by ensuring that the eps value is not too small (which would result in too many small clusters) or too large (which would merge distinct clusters).

```r
# Convert df.scaled to a matrix
df.matrix <- as.matrix(df.scaled)

# Compute the distance to the k-th nearest neighbor (k = minPts - 1)
k <- 2  # Since minPts is 5
kNNdistplot(df.matrix, k = k)

# Add a horizontal line at a potential eps value
abline(h = 4.5, col = "red", lty = 2)  # Adjust based on your plot
```




```r
# Load necessary library
library(dbscan)

# Define parameters
eps <- 4.5  # Chosen based on the elbow in the k-distance plot
minPts <- 3  # Example value for minPts

# Apply DBSCAN with the selected parameters
db <- dbscan(df.matrix, eps = eps, minPts = minPts)

# Print the clustering results
print(db)
```

```
## DBSCAN clustering for 24 objects.
## Parameters: eps = 4.5, minPts = 3
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 1 cluster(s) and 3 noise points.
##
##  0  1
##  3 21
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

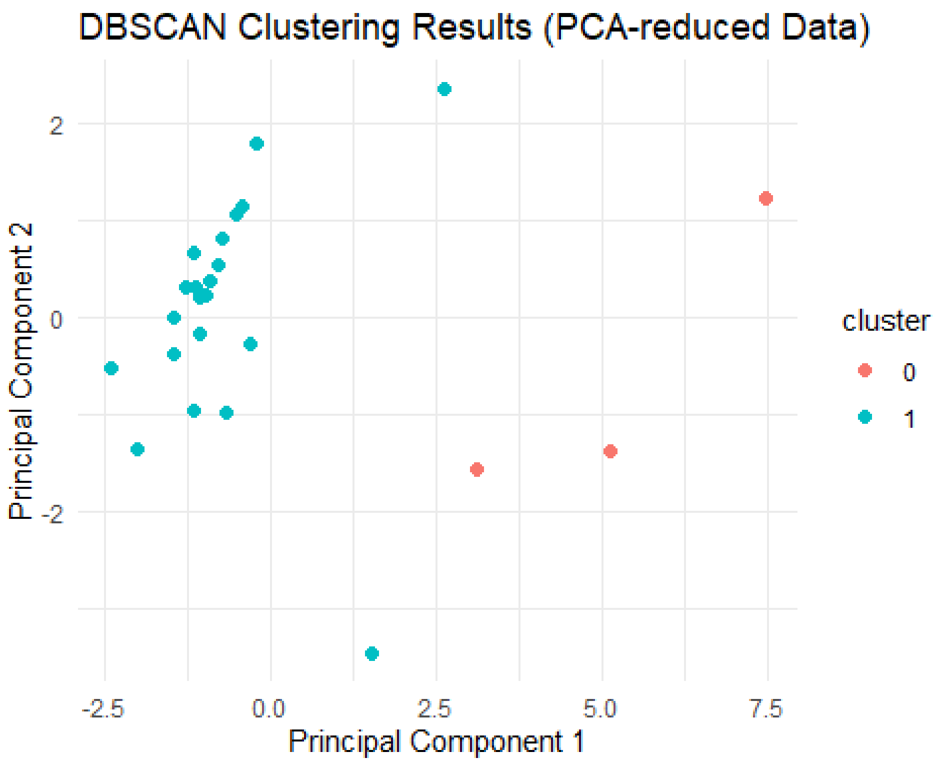
```
# Add the cluster labels to the dataset
df.scaled$cluster <- db$cluster
```

```
## Warning in df.scaled$cluster <- db$cluster: Coercing LHS to a list
```

```
# View the dataset with cluster labels
head(df.scaled)
```

```
## [[1]]
## [1] -0.5427402
##
## [[2]]
## [1] 1.161563
##
## [[3]]
## [1] 0.08683632
##
## [[4]]
## [1] 2.918609
##
## [[5]]
## [1] 0.2602313
##
## [[6]]
## [1] -0.4369107
```

```
# Load necessary library for PCA and plotting
library(ggplot2)

# Perform PCA to reduce to 2 dimensions
pca <- prcomp(df.matrix, scale. = TRUE)
pca_data <- data.frame(PC1 = pca$x[,1], PC2 = pca$x[,2], cluster =
as.factor(db$cluster))

# Plot the PCA result
ggplot(pca_data, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point(size = 2) +
  labs(title = "DBSCAN Clustering Results (PCA-reduced Data)",
       x = "Principal Component 1",
       y = "Principal Component 2") +
  theme_minimal()
```




#Results:

Number of Clusters: The DBSCAN algorithm identified a single cluster. Noise Points: There are 3 points that are considered noise (outliers) because they do not meet the criteria to be included in any cluster. Cluster Distribution:

Cluster 0: Contains 3 points. These are noise points, as indicated by the cluster label 0. Cluster 1: Contains 21 points. These points form the single cluster identified by DBSCAN. Details of Noise Points:

Noise points are those that do not have enough neighboring points (at least minPts points within eps distance) to form a dense region. In this case, 3 points did not meet this criterion and are thus labeled as noise. Detailed Output:

#Comparison of K-Means and DBSCAN Based on PCA Plots

13. **K-Means Clustering**:
    - Shows two clusters.
    - This method does not inherently detect noise.
    - All points are assigned to one of the clusters, including outliers and noise.
14. **DBSCAN Clustering**:
    - Shows one cluster and identifies noise points.
    - This method is better at handling noise and outliers, explicitly marking them.

- Based on the PCA plot from DBSCAN, we can clearly see that there are 3 outliers which are away from the cluster. DBSCAN was able to detect these outliers, while K-Means was not.

#10. Davies-Bouldin

```
library(clusterSim)

## Warning: package 'clusterSim' was built under R version 4.3.3

## Loading required package: MASS

## Warning: package 'MASS' was built under R version 4.3.2

##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select

# Calculate Davies-Bouldin score for K-Means
kmeans_davies_bouldin <- index.DB(df.matrix, km.res$cluster)$DB
print(paste("Davies-Bouldin Index for K-Means:", kmeans_davies_bouldin))

## [1] "Davies-Bouldin Index for K-Means: 1.07548402938478"
```

As DBSCAN detected only one valid cluster (and noise), the Davies-Bouldin index cannot be computed as it requires at least two clusters to measure the distances between them.

The Davies-Bouldin Index (DBI) is a measure of clustering quality, where a lower value indicates better-defined clusters.

#Range The Davies-Bouldin Index is non-negative and has a minimum value of 0. A DBI of 0 indicates perfect clustering with completely distinct clusters. In practical applications, DBI values typically range from 0 to a few units

DBI < 1: Generally indicates good clustering quality. DBI between 1 and 2: Indicates reasonably good clustering quality, though there might be room for improvement. DBI > 2: Suggests that clustering quality might be poor and the clusters are not well-separated.

A DBI of 1.075: Suggests that the clusters formed by the K-Means algorithm are relatively well-defined but not perfect.

Key Difference between Silhouette Score and Davies-Bouldin Index: Silhouette Score is more intuitive and interpretable for individual points and overall clustering. DBI focuses on cluster separations and compactness, providing a single overall score for clustering quality. Both metrics are useful for evaluating and comparing clustering results, with Silhouette Score offering more detailed insight and DBI providing a holistic measure.

Part 2 : E-commerce Customer Clustering

# Customer clustering

## 1. Data cleaning

### 1.1 load data

```
# Data manipulation and cleaning

# load packages
library(dplyr)

## Warning: package 'dplyr' was built under R version 4.3.3

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(tidyr)

## Warning: package 'tidyr' was built under R version 4.3.3

library(lubridate)

## Warning: package 'lubridate' was built under R version 4.3.3

##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

# Visualization
library(ggplot2)
library(factoextra)

## Warning: package 'factoextra' was built under R version 4.3.3

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

library(fmsb)

## Warning: package 'fmsb' was built under R version 4.3.3
```

```
# Clustering
library(cluster)
library(dbscan)

## Warning: package 'dbscan' was built under R version 4.3.3

##
## Attaching package: 'dbscan'

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

library(clValid)

## Warning: package 'clValid' was built under R version 4.3.3

# Miscellaneous
library(broom)

## Warning: package 'broom' was built under R version 4.3.3

library(reshape2)

## Warning: package 'reshape2' was built under R version 4.3.3

##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths

library(hopkins)

## Warning: package 'hopkins' was built under R version 4.3.3

# Load customer transaction data

df = read.csv('C:\\Users\\wudan\\OneDrive - Langara
College\\DANA_4840\\PROJECT\\DANA4840_Shared_folder\\customerData\\transactio
n_data.csv\\transaction_data.csv')

#check dataset structure

str(df)

## 'data.frame':    1083818 obs. of  8 variables:
##  $ UserId                : int  278166 337701 267099 380478 -1 285957
345954 -1 339822 328440 ...
##  $ TransactionId         : int  6355745 6283376 6385599 6044973 6143225
6307136 6162981 6143225 6255403 6387425 ...
##  $ TransactionTime       : chr  "Sat Feb 02 12:50:00 IST 2019" "Wed Dec 26
09:06:00 IST 2018" "Fri Feb 15 09:45:00 IST 2019" "Fri Jun 22 07:14:00 IST
```

```
2018" ...
##  $ ItemCode              : int  465549 482370 490728 459186 1733592
1787247 471576 447867 1783845 494802 ...
##  $ ItemDescription       : chr  "FAMILY ALBUM WHITE PICTURE FRAME" "LONDON
BUS COFFEE MUG" "SET 12 COLOUR PENCILS DOLLY GIRL " "UNION JACK FLAG LUGGAGE
TAG" ...
##  $ NumberOfItemsPurchased: int  6 3 72 3 3 12 9 120 36 36 ...
##  $ CostPerItem           : num  11.73 3.52 0.9 1.73 3.4 ...
##  $ Country               : chr  "United Kingdom" "United Kingdom" "France"
"United Kingdom" ...

head(df)

##   UserId TransactionId              TransactionTime ItemCode
## 1 278166       6355745 Sat Feb 02 12:50:00 IST 2019   465549
## 2 337701       6283376 Wed Dec 26 09:06:00 IST 2018   482370
## 3 267099       6385599 Fri Feb 15 09:45:00 IST 2019   490728
## 4 380478       6044973 Fri Jun 22 07:14:00 IST 2018   459186
## 5     -1       6143225 Mon Sep 10 11:58:00 IST 2018  1733592
## 6 285957       6307136 Fri Jan 11 09:50:00 IST 2019  1787247
##                        ItemDescription NumberOfItemsPurchased CostPerItem
## 1  FAMILY ALBUM WHITE PICTURE FRAME                      6       11.73
## 2              LONDON BUS COFFEE MUG                      3        3.52
## 3 SET 12 COLOUR PENCILS DOLLY GIRL                       72        0.90
## 4       UNION JACK FLAG LUGGAGE TAG                       3        1.73
## 5               WASHROOM METAL SIGN                       3        3.40
## 6  CUT GLASS T-LIGHT HOLDER OCTAGON                      12        3.52
##          Country
## 1 United Kingdom
## 2 United Kingdom
## 3         France
## 4 United Kingdom
## 5 United Kingdom
## 6 United Kingdom

# rename columns
names(df)

## [1] "UserId"                 "TransactionId"          "TransactionTime"
## [4] "ItemCode"               "ItemDescription"
"NumberOfItemsPurchased"
## [7] "CostPerItem"            "Country"

names(df) =
c('CustomerID','InvoiceNo','InvoiceDate','StockCode','Description','Quantity'
,'UnitPrice','Country')
names(df)

## [1] "CustomerID"  "InvoiceNo"   "InvoiceDate" "StockCode"   "Description"
## [6] "Quantity"    "UnitPrice"   "Country"
```

| Variable | Description |
|---|---|
| **InvoiceNo** | Code representing each unique transaction. |
| **StockCode** | Code uniquely assigned to each distinct product. |
| **Description** | Description of each product. |
| **Quantity** | The number of units of a product in a transaction. |
| **InvoiceDate** | The date and time of the transaction. |
| **UnitPrice** | The unit price of the product in sterling. |
| **CustomerID** | Identifier uniquely assigned to each customer. |
| **Country** | The country of the customer. |

```
summary(df)

##    CustomerID       InvoiceNo       InvoiceDate         StockCode
##  Min.   :     -1   Min.   :5900015   Length:1083818     Min.   :      -1
##  1st Qu.:259392   1st Qu.:6026856   Class :character   1st Qu.: 460908
##  Median :302022   Median :6166611   Mode  :character   Median : 475293
##  Mean   :241016   Mean   :6159417                      Mean   : 658269
##  3rd Qu.:341355   3rd Qu.:6289569                      3rd Qu.: 488943
##  Max.   :384027   Max.   :6397457                      Max.   :1894494
##  Description           Quantity              UnitPrice           Country
##  Length:1083818     Min.   :-242985.00   Min.   : -15265.6
Length:1083818
##  Class :character   1st Qu.:      3.00   1st Qu.:      1.7   Class
:character
##  Mode  :character   Median :      9.00   Median :      2.9   Mode
:character
##                     Mean   :     28.66   Mean   :      9.5
##                     3rd Qu.:     30.00   3rd Qu.:      5.7
##                     Max.   : 242985.00   Max.   :1696285.4

# check the nunique of each columns
library(dplyr)

nunique_df <- df %>% summarise(across(everything(), n_distinct))
print(nunique_df)

##   CustomerID InvoiceNo InvoiceDate StockCode Description Quantity
UnitPrice
## 1       4373     25900       23260      3407        4224      722
1631
##   Country
## 1      38
```

### 1.2 data imputation (missing value, duplicates,)

#### *1.2.1 Missing value*

```
# missing value function
replace_na_values <- function(x) {
```

```r
  na_values <- c(" ", "?", "_", "-1")
  x[x %in% na_values] <- NA
  return(x)
}

# Apply the custom function to each column
df <- df %>% mutate(across(everything(), replace_na_values))

# Count NA values in each column
na_count_per_column <- df %>% summarise(across(everything(), ~
sum(is.na(.))))
print(na_count_per_column)

##   CustomerID InvoiceNo InvoiceDate StockCode Description Quantity
UnitPrice
## 1     270160         0           0      5592          94        0
0
##   Country
## 1       0

# Count total NA values in the entire dataframe
total_na_count <- sum(is.na(df))
print(total_na_count)

## [1] 275846
```

There are 275846 missing values in the df. CustomerID : 270160 NA StockCode : 5592 NA Description : 94 NA

```r
# percentage of missing value in each column
print(na_count_per_column/nrow(df))

##   CustomerID InvoiceNo InvoiceDate   StockCode  Description Quantity
UnitPrice
## 1  0.2492669         0           0 0.005159538 8.673043e-05        0
0
##   Country
## 1       0
```

15. The `CustomerID` column contains nearly a quarter of missing data. This column is essential for clustering customers . Imputing such a large percentage of missing values might introduce significant bias or noise into the analysis. Since the clustering is based on customer behavior and preferences, it's crucial to have accurate data on customer identifiers. Therefore, removing the rows with missing `CustomerIDs` seems to be the most reasonable approach to maintain the integrity of the clusters and the analysis.
16. 'StockCode' and 'Description' colunns have missing values rates lower than 1%. So we remove the rows with missing values in 'StockCode' and 'Description' colunns.

```r
# Drop rows with any NA values
df_clean <- na.omit(df)
```

```r
summary(df_clean)
```

```
##    CustomerID       InvoiceNo       InvoiceDate          StockCode     
##  Min.   :259266   Min.   :5900015   Length:810086      Min.   :      42  
##  1st Qu.:293349   1st Qu.:6040430   Class :character   1st Qu.: 462609  
##  Median :318339   Median :6180603   Mode  :character   Median : 475986  
##  Mean   :321194   Mean   :6166424                      Mean   : 645977  
##  3rd Qu.:352674   3rd Qu.:6292726                      3rd Qu.: 488628  
##  Max.   :384027   Max.   :6397457                      Max.   :1894494  
##  Description           Quantity             UnitPrice           Country         
##  Length:810086      Min.   :-242985.00   Min.   :      0.0   Length:810086     
##  Class :character   1st Qu.:       6.00   1st Qu.:      1.7   Class  
:character  
##  Mode  :character   Median :      15.00   Median :      2.7   Mode   
:character  
##                     Mean   :      36.31   Mean   :      8.2                     
##                     3rd Qu.:      36.00   3rd Qu.:      5.2                     
##                     Max.   : 242985.00   Max.   :1696285.4
```

```r
# Count total NA values in the entire dataframe
total_na_count <- sum(is.na(df_clean))
print(total_na_count)
```

```
## [1] 0
```

### 1.2.2 Duplicate rows

```r
# Check for duplicate rows
duplicate_rows <- df_clean[duplicated(df_clean), ]

# Display duplicate rows
print("number of Duplicate rows:")
```

```
## [1] "number of Duplicate rows:"
```

```r
print(nrow(duplicate_rows))
```

```
## [1] 410298
```

```r
# Remove duplicate rows
df_clean <- df_clean %>% distinct()

# Display the dataframe with duplicates removed
print("Number of rows after removing duplicates:")
```

```
## [1] "Number of rows after removing duplicates:"
```

```r
print(nrow(df_clean))
```

```
## [1] 399788
```

There are 410298 duplicates rows, which suggests that this dataset may have data recording errors. We remove all the duplicates rows and then the number of rows of the cleaned dataframe is 399788.

### *1.2.3 Cancelled Transactions*

```
summary(df_clean)

##    CustomerID       InvoiceNo       InvoiceDate          StockCode
##  Min.   :259266   Min.   :5900015   Length:399788      Min.   :     42
##  1st Qu.:293139   1st Qu.:6040628   Class :character   1st Qu.: 462609
##  Median :318150   Median :6180009   Mode  :character   Median : 475986
##  Mean   :321057   Mean   :6166153                      Mean   : 646057
##  3rd Qu.:352611   3rd Qu.:6292385                      3rd Qu.: 488607
##  Max.   :384027   Max.   :6397457                      Max.   :1894494
##  Description           Quantity            UnitPrice          Country
##  Length:399788      Min.   :-242985.00   Min.   :      0.0   Length:399788
##  Class :character   1st Qu.:      6.00   1st Qu.:      1.7   Class
:character
##  Mode  :character   Median :     15.00   Median :      2.7   Mode
:character
##                     Mean   :     36.68   Mean   :      8.3
##                     3rd Qu.:     36.00   3rd Qu.:      5.2
##                     Max.   : 242985.00   Max.   :1696285.4
```

From summary, we can notice that there are negative Quantity values. From dataset introduction, we know that there are Cancelled Transactions.

```
# Select rows with negative Quantity values
cancel <- df_clean %>% filter(Quantity < 0)

# Display the rows with negative Quantity values
print("Cancel transactions:")

## [1] "Cancel transactions:"

print(nrow(cancel))

## [1] 8506

df_clean <- df_clean %>%
  mutate(Transaction_Status = ifelse(Quantity < 0, 0, 1))

print(head(df_clean))

##   CustomerID InvoiceNo                  InvoiceDate StockCode
## 1     278166   6355745 Sat Feb 02 12:50:00 IST 2019    465549
## 2     337701   6283376 Wed Dec 26 09:06:00 IST 2018    482370
## 3     267099   6385599 Fri Feb 15 09:45:00 IST 2019    490728
## 4     380478   6044973 Fri Jun 22 07:14:00 IST 2018    459186
## 5     285957   6307136 Fri Jan 11 09:50:00 IST 2019   1787247
## 6     345954   6162981 Fri Sep 28 10:51:00 IST 2018    471576
```

```
##                              Description Quantity UnitPrice        Country
## 1  FAMILY ALBUM WHITE PICTURE FRAME        6     11.73 United Kingdom
## 2             LONDON BUS COFFEE MUG        3      3.52 United Kingdom
## 3 SET 12 COLOUR PENCILS DOLLY GIRL        72      0.90         France
## 4       UNION JACK FLAG LUGGAGE TAG        3      1.73 United Kingdom
## 5  CUT GLASS T-LIGHT HOLDER OCTAGON       12      3.52 United Kingdom
## 6   NATURAL SLATE CHALKBOARD LARGE         9      6.84 United Kingdom
##   Transaction_Status
## 1                  1
## 2                  1
## 3                  1
## 4                  1
## 5                  1
## 6                  1
```

## 2. feature engineering

In this part, we apply feature engineering[1] to transform customer transaction data into customer centralized data.

[1]https://www.kaggle.com/code/xunbch/customer-segmentation-recommendation-system/input?select=data.csv

### 2.1 RFM features: recency, frequency, monetary

RFM is a method used for analyzing customer value and segmenting the customer base. It is an acronym that stands for:

Recency (R):This metric indicates how recently a customer has made a purchase. A lower recency value means the customer has purchased more recently, indicating higher engagement with the brand.

Frequency (F):This metric signifies how often a customer makes a purchase within a certain period. A higher frequency value indicates a customer who interacts with the business more often, suggesting higher loyalty or satisfaction.

Monetary (M):This metric represents the total amount of money a customer has spent over a certain period. Customers who have a higher monetary value have contributed more to the business, indicating their potential high lifetime value.

Together, these metrics help in understanding a customer's buying behavior and preferences. #### 2.1.1 Recency Days Since Last Purchas: This feature represents the number of days that have passed since the customer's last purchase. A lower value indicates that the customer has purchased recently, implying a higher engagement level with the business, whereas a higher value may indicate a lapse or decreased engagement. By understanding the recency of purchases, businesses can tailor their marketing strategies to re-engage customers who have not made purchases in a while, potentially increasing customer retention and fostering loyalty.

```r
#install.packages("lubridate")
library(lubridate)


# Remove timezone information and then parse the datetime
df_clean <- df_clean %>%
  mutate(InvoiceDate_no_tz = gsub(" IST", "", InvoiceDate)) %>%
  mutate(InvoiceDay = as.Date(parse_date_time(InvoiceDate_no_tz, orders = "a
b d H:M:S Y")))
# Remove the InvoiceDay column
df_clean <- df_clean %>% select(-InvoiceDate_no_tz)
# Display the first few rows to check the conversion
print(head(df_clean))
```

```
##   CustomerID InvoiceNo                  InvoiceDate StockCode
## 1     278166   6355745 Sat Feb 02 12:50:00 IST 2019    465549
## 2     337701   6283376 Wed Dec 26 09:06:00 IST 2018    482370
## 3     267099   6385599 Fri Feb 15 09:45:00 IST 2019    490728
## 4     380478   6044973 Fri Jun 22 07:14:00 IST 2018    459186
## 5     285957   6307136 Fri Jan 11 09:50:00 IST 2019   1787247
## 6     345954   6162981 Fri Sep 28 10:51:00 IST 2018    471576
##                          Description Quantity UnitPrice        Country
## 1  FAMILY ALBUM WHITE PICTURE FRAME         6     11.73 United Kingdom
## 2             LONDON BUS COFFEE MUG         3      3.52 United Kingdom
## 3 SET 12 COLOUR PENCILS DOLLY GIRL         72      0.90         France
## 4        UNION JACK FLAG LUGGAGE TAG         3      1.73 United Kingdom
## 5  CUT GLASS T-LIGHT HOLDER OCTAGON        12      3.52 United Kingdom
## 6   NATURAL SLATE CHALKBOARD LARGE          9      6.84 United Kingdom
##   Transaction_Status InvoiceDay
## 1                  1 2019-02-02
## 2                  1 2018-12-26
## 3                  1 2019-02-15
## 4                  1 2018-06-22
## 5                  1 2019-01-11
## 6                  1 2018-09-28
```

```r
# Remove rows with InvoiceDate after 2019
df_clean <- df_clean %>% filter(InvoiceDay <= as.Date("2019-12-31"))

# Find the most recent purchase date for each customer
customer_data <- df_clean %>% group_by(CustomerID) %>% summarise(InvoiceDay =
max(InvoiceDay)) %>% ungroup()
print(head(customer_data))
```

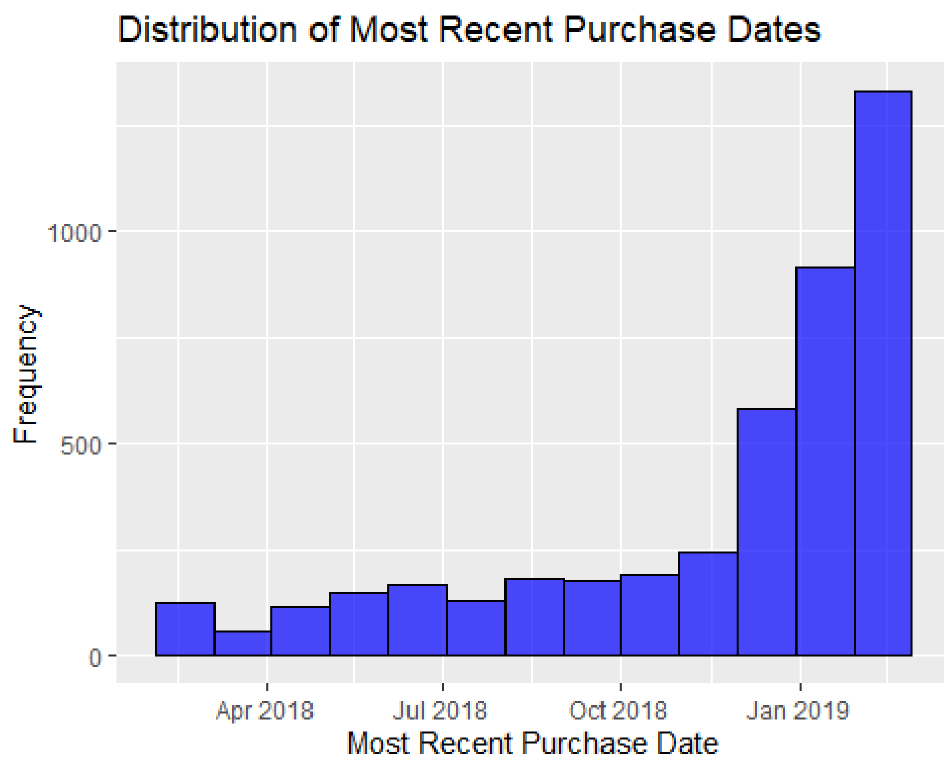
```
## # A tibble: 6 × 2
##   CustomerID InvoiceDay
##        <int> <date>
## 1     259266 2018-04-01
## 2     259287 2019-02-18
## 3     259308 2018-12-07
```

```
## 4      259329 2019-02-02
## 5      259350 2018-04-16
## 6      259392 2019-01-15
```

```r
# Find the most recent date in the entire dataset
most_recent_date <- max(df_clean$InvoiceDay)
print(head(most_recent_date))
```

```
## [1] "2019-02-20"
```

```r
# Plot the distribution of most recent purchase dates
ggplot(customer_data, aes(x = InvoiceDay)) +
  geom_histogram(binwidth = 30, fill = "blue", color = "black", alpha = 0.7)
+
  labs(title = "Distribution of Most Recent Purchase Dates", x = "Most Recent
Purchase Date", y = "Frequency")
```




```r
# Calculate the number of days since the last purchase for each customer
customer_data <- customer_data %>%
  mutate(Days_Since_Last_Purchase = as.numeric(difftime(most_recent_date,
InvoiceDay, units = "days")))

# Remove the InvoiceDay column
customer_data <- customer_data %>% select(-InvoiceDay)

# Display the resulting dataframe
print(head(customer_data))
```

```
## # A tibble: 6 × 2
##   CustomerID Days_Since_Last_Purchase
##        <int>                    <dbl>
## 1     259266                      325
## 2     259287                        2
## 3     259308                       75
## 4     259329                       18
## 5     259350                      310
## 6     259392                       36

print(nrow(customer_data))

## [1] 4358
```

### *2.1.2 Frequency*

Total Transactions: This feature represents the total number of transactions made by a customer. It helps in understanding the engagement level of a customer with the retailer.

Total Products Purchased: This feature indicates the total number of products (sum of quantities) purchased by a customer across all transactions. It gives an insight into the customer's buying behavior in terms of the volume of products purchased.

```
# Calculate the total number of transactions made by each customer
total_transactions <- df_clean %>%
  group_by(CustomerID) %>%
  summarise(Total_Transactions = n_distinct(InvoiceNo)) %>%
  ungroup()

# Calculate the total number of products purchased by each customer
total_products_purchased <- df_clean %>%
  group_by(CustomerID) %>%
  summarise(Total_Products_Purchased = sum(Quantity)) %>%
  ungroup()

# Merge the new features into the customer_data dataframe
customer_data <- customer_data %>%
  left_join(total_transactions, by = "CustomerID") %>%
  left_join(total_products_purchased, by = "CustomerID")

print(head(customer_data))

## # A tibble: 6 × 4
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
```

```
## 3     259308                      75                   4
6996
## 4     259329                      18                   1
1890
## 5     259350                     310                   1
588
## 6     259392                      36                   8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
```

From above table, we can find an outlier, customerID:259266 with a transaction of Quantity 0, Which could be an order by mistake, then it was cancelled.

```
print(df %>% filter(CustomerID == 259266))

##   CustomerID InvoiceNo                   InvoiceDate StockCode
## 1     259266   5955763 Sun Apr 01 06:17:00 IST 2018    486486
## 2     259266   5955763 Sun Apr 01 06:17:00 IST 2018    486486
## 3     259266   5955741 Sun Apr 01 06:01:00 IST 2018    486486
## 4     259266   5955741 Sun Apr 01 06:01:00 IST 2018    486486
##                   Description Quantity UnitPrice        Country
## 1 MEDIUM CERAMIC TOP STORAGE JAR  -222645      1.44 United Kingdom
## 2 MEDIUM CERAMIC TOP STORAGE JAR  -222645      1.44 United Kingdom
## 3 MEDIUM CERAMIC TOP STORAGE JAR   222645      1.44 United Kingdom
## 4 MEDIUM CERAMIC TOP STORAGE JAR   222645      1.44 United Kingdom
```

#### *2.1.3 Monetary*

Total Spend: This feature represents the total amount of money spent by each customer. It is calculated as the sum of the product of `UnitPrice` and `Quantity` for all transactions made by a customer. This feature is crucial as it helps in identifying the total revenue generated by each customer, which is a direct indicator of a customer's value to the business.

Average Transaction Value: This feature is calculated as the **Total Spend** divided by the **Total Transactions** for each customer. It indicates the average value of a transaction carried out by a customer. This metric is useful in understanding the spending behavior of customers per transaction, which can assist in tailoring marketing strategies and offers to different customer segments based on their average spending patterns.

```
# Calculate the total spend by each customer
df_clean <- df_clean %>%
  mutate(Total_Spend = UnitPrice * Quantity)

total_spend <- df_clean %>%
  group_by(CustomerID) %>%
  summarise(Total_Spend = sum(Total_Spend)) %>%
  ungroup()

# Calculate the average transaction value for each customer
average_transaction_value <- total_spend %>%
```

```
  left_join(total_transactions, by = "CustomerID") %>%
  mutate(Average_Transaction_Value = Total_Spend / Total_Transactions)

# Merge the new features into the customer_data dataframe
customer_data <- customer_data %>%
  left_join(total_spend, by = "CustomerID") %>%
  left_join(average_transaction_value %>% select(CustomerID,
Average_Transaction_Value), by = "CustomerID")

# Display the first few rows of the customer_data dataframe
print(head(customer_data))
```

```
## # A tibble: 6 × 6
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 2 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>
```

### 2.2 Product Diversity

**Unique Products Purchased**: This feature represents the number of distinct products bought by a customer. A higher value indicates that the customer has a diverse taste or preference, buying a wide range of products, while a lower value might indicate a focused or specific preference. Understanding the diversity in product purchases can help in segmenting customers based on their buying diversity, which can be a critical input in personalizing product recommendations.

```
# get the sum of unique stockcode of each customer
total_unique_product <- df_clean %>%
  group_by(CustomerID) %>%
  summarise(total_unique_product = n_distinct(StockCode)) %>%
  ungroup()

customer_data <- customer_data %>%
  left_join(total_unique_product, by = "CustomerID")
```

```
print(head(customer_data))

## # A tibble: 6 × 7
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 3 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>
```

### 2.3 Behavioural Features

**Average Days Between Purchases**: This feature represents the average number of days a customer waits before making another purchase. Understanding this can help in predicting when the customer is likely to make their next purchase, which can be a crucial metric for targeted marketing and personalized promotions.

**Favorite Shopping Day**: This denotes the day of the week when the customer shops the most. This information can help in identifying the preferred shopping days of different customer segments, which can be used to optimize marketing strategies and promotions for different days of the week.

**Favorite Shopping Hour**: This refers to the hour of the day when the customer shops the most. Identifying the favorite shopping hour can aid in optimizing the timing of marketing campaigns and promotions to align with the times when different customer segments are most active.

```
# Extract day of the week and hour from InvoiceDate
df_clean <- df_clean %>%
  mutate(InvoiceDate_no_tz = gsub(" IST", "", InvoiceDate)) %>%
  mutate(InvoiceDate = parse_date_time(InvoiceDate_no_tz, orders = "a b d
H:M:S Y"))

# Extract day of the week and hour from InvoiceDate
df_clean <- df_clean %>%
```

```r
  mutate(Day_Of_Week = wday(InvoiceDate, label = TRUE, week_start = 1),
         Hour = hour(InvoiceDate))

library(tidyr)
# Calculate the average number of days between consecutive purchases
days_between_purchases <- df_clean %>%
  group_by(CustomerID) %>%
  arrange(CustomerID, InvoiceDate) %>%
  mutate(Days_Between = as.numeric(difftime(InvoiceDate, lag(InvoiceDate),
units = "days"))) %>%
  ungroup()

average_days_between_purchases <- days_between_purchases %>%
  group_by(CustomerID) %>%
  summarise(Average_Days_Between_Purchases = mean(Days_Between, na.rm =
TRUE)) %>%
  ungroup()

# Replace NA values with 0
average_days_between_purchases <- average_days_between_purchases %>%
  mutate(Average_Days_Between_Purchases =
replace_na(Average_Days_Between_Purchases, 0))

# Find the favorite shopping day of the week
favorite_shopping_day <- df_clean %>%
  group_by(CustomerID, Day_Of_Week) %>%
  summarise(Count = n()) %>%
  ungroup() %>%
  arrange(CustomerID, desc(Count)) %>%
  group_by(CustomerID) %>%
  slice(1) %>%
  select(CustomerID, Day_Of_Week)

## `summarise()` has grouped output by 'CustomerID'. You can override using
the
## `.groups` argument.

# Find the favorite shopping hour of the day
favorite_shopping_hour <- df_clean %>%
  group_by(CustomerID, Hour) %>%
  summarise(Count = n()) %>%
  ungroup() %>%
  arrange(CustomerID, desc(Count)) %>%
  group_by(CustomerID) %>%
  slice(1) %>%
  select(CustomerID, Hour)

## `summarise()` has grouped output by 'CustomerID'. You can override using
the
## `.groups` argument.
```

```
# Merge the new features into the customer_data dataframe
customer_data <- customer_data %>%
  left_join(average_days_between_purchases, by = "CustomerID") %>%
  left_join(favorite_shopping_day, by = "CustomerID") %>%
  left_join(favorite_shopping_hour, by = "CustomerID")

head(customer_data)

## # A tibble: 6 × 10
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 6 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>, Average_Days_Between_Purchases <dbl>,
## #   Day_Of_Week <ord>, Hour <int>
```

### 2.4 Geographic features

**Country**: This feature identifies the country where each customer is located. Including the country data can help us understand region-specific buying patterns and preferences. Different regions might have varying preferences and purchasing behaviors which can be critical in personalizing marketing strategies and inventory planning. Furthermore, it can be instrumental in logistics and supply chain optimization, particularly for an online retailer where shipping and delivery play a significant role.

```
# Calculate the normalized value counts for the 'Country' column
country_counts <- df_clean %>%
  group_by(Country) %>%
  summarise(Count = n()) %>%
  mutate(Proportion = Count / sum(Count)) %>%
  arrange(desc(Proportion)) %>%
  ungroup()

# Display the top results
head(country_counts)
```

```
## # A tibble: 6 × 3
##   Country         Count Proportion
##   <chr>           <int>      <dbl>
## 1 United Kingdom 355100    0.891
## 2 Germany          9065    0.0227
## 3 France           8095    0.0203
## 4 EIRE             7469    0.0187
## 5 Spain            2463    0.00618
## 6 Netherlands      2330    0.00584
```

Given that a substantial portion (**89%**) of transactions are originating from the **United Kingdom**, we might consider creating a binary feature indicating whether the transaction is from the UK or not. This approach can potentially streamline the clustering process without losing critical geographical information, especially when considering the application of algorithms like K-means which are sensitive to the dimensionality of the feature space.

```
# Group by CustomerID and Country to get the number of transactions per
country for each customer
customer_country <- df_clean %>%
  group_by(CustomerID, Country) %>%
  summarise(Number_of_Transactions = n()) %>%
  ungroup()

## `summarise()` has grouped output by 'CustomerID'. You can override using
the
## `.groups` argument.

# Get the country with the maximum number of transactions for each customer
customer_main_country <- customer_country %>%
  arrange(CustomerID, desc(Number_of_Transactions)) %>%
  group_by(CustomerID) %>%
  slice(1) %>%
  ungroup()

# Create a binary column indicating whether the customer is from the UK or
not
customer_main_country <- customer_main_country %>%
  mutate(Is_UK = ifelse(Country == 'United Kingdom', 1, 0))

# Assuming customer_data is already defined
customer_data <- customer_data %>%
  left_join(customer_main_country %>% select(CustomerID, Is_UK), by =
"CustomerID")

print(head(customer_data))

## # A tibble: 6 × 11
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
```

```
##        <int>                    <dbl>             <int>
<int>
## 1     259266                     325                 2
0
## 2     259287                       2                 6
6417
## 3     259308                      75                 4
6996
## 4     259329                      18                 1
1890
## 5     259350                     310                 1
588
## 6     259392                      36                 8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 7 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>, Average_Days_Between_Purchases <dbl>,
## #   Day_Of_Week <ord>, Hour <int>, Is_UK <dbl>
```

### 2.5 Cancellation Insights

**Cancellation Frequency**: This metric represents the total number of transactions a customer has canceled. Understanding the frequency of cancellations can help us identify customers who are more likely to cancel transactions. This could be an indicator of dissatisfaction or other issues, and understanding this can help us tailor strategies to reduce cancellations and enhance customer satisfaction.

**Cancellation Rate**: This represents the proportion of transactions that a customer has canceled out of all their transactions. This metric gives a normalized view of cancellation behavior. A high cancellation rate might be indicative of an unsatisfied customer segment. By identifying these segments, we can develop targeted strategies to improve their shopping experience and potentially reduce the cancellation rate.

```r
# Calculate the number of cancelled transactions for each customer
cancelled_transactions <- df_clean %>%
  filter(Transaction_Status == 0) %>%
  group_by(CustomerID) %>%
  summarise(Cancellation_Frequency = n_distinct(InvoiceNo)) %>%
  ungroup()

# Merge the Cancellation Frequency data into the customer_data dataframe
customer_data <- customer_data %>%
  left_join(cancelled_transactions, by = "CustomerID")

# Replace NA values with 0 (for customers who have not cancelled any
transaction)
customer_data <- customer_data %>%
  mutate(Cancellation_Frequency = ifelse(is.na(Cancellation_Frequency), 0,
Cancellation_Frequency))
```

```r
print(head(customer_data))
```

```
## # A tibble: 6 × 12
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 8 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>, Average_Days_Between_Purchases <dbl>,
## #   Day_Of_Week <ord>, Hour <int>, Is_UK <dbl>, Cancellation_Frequency
<dbl>
```

```r
# Calculate the Cancellation Rate = cancel count /total transaction count
customer_data <- customer_data %>%
  mutate(Cancellation_Rate = Cancellation_Frequency / Total_Transactions)

print(head(customer_data))
```

```
## # A tibble: 6 × 13
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
```

```
## # ℹ 9 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>, Average_Days_Between_Purchases <dbl>,
## #   Day_Of_Week <ord>, Hour <int>, Is_UK <dbl>, Cancellation_Frequency
<dbl>,
## #   Cancellation_Rate <dbl>
```

### 2.6 Seasonality & Trends

**Monthly_Spending_Mean**: This is the average amount a customer spends monthly. It helps us gauge the general spending habit of each customer. A higher mean indicates a customer who spends more, potentially showing interest in premium products, whereas a lower mean might indicate a more budget-conscious customer.

**Monthly_Spending_Std**: This feature indicates the variability in a customer's monthly spending. A higher value signals that the customer's spending fluctuates significantly month-to-month, perhaps indicating sporadic large purchases. In contrast, a lower value suggests more stable, consistent spending habits. Understanding this variability can help in crafting personalized promotions or discounts during periods they are expected to spend more.

**Spending_Trend**: This reflects the trend in a customer's spending over time, calculated as the slope of the linear trend line fitted to their spending data. A positive value indicates an increasing trend in spending, possibly pointing to growing loyalty or satisfaction. Conversely, a negative trend might signal decreasing interest or satisfaction, highlighting a need for re-engagement strategies. A near-zero value signifies stable spending habits. Recognizing these trends can help in developing strategies to either maintain or alter customer spending patterns, enhancing the effectiveness of marketing campaigns.

```r
#install.packages("broom")
library(broom)

# Extract month and year from InvoiceDate
df_clean <- df_clean %>%
  mutate(Year = year(InvoiceDate),
         Month = month(InvoiceDate))

# Assuming Total_Spend is already calculated in df_clean
monthly_spending <- df_clean %>%
  group_by(CustomerID, Year, Month) %>%
  summarise(Total_Spend = sum(Total_Spend)) %>%
  ungroup()

## `summarise()` has grouped output by 'CustomerID', 'Year'. You can override
## using the `.groups` argument.

# Calculate seasonal buying patterns
seasonal_buying_patterns <- monthly_spending %>%
  group_by(CustomerID) %>%
  summarise(Monthly_Spending_Mean = mean(Total_Spend, na.rm = TRUE),
            Monthly_Spending_Std = sd(Total_Spend, na.rm = TRUE)) %>%
```

```
  ungroup()

# Replace NA values in Monthly_Spending_Std with 0
seasonal_buying_patterns <- seasonal_buying_patterns %>%
  mutate(Monthly_Spending_Std = ifelse(is.na(Monthly_Spending_Std), 0,
Monthly_Spending_Std))

# Define a function to calculate the spending trend
calculate_trend <- function(spend_data) {
  if (nrow(spend_data) > 1) {
    model <- lm(Total_Spend ~ Month + Year, data = spend_data)
    slope <- coef(model)["Month"]
    return(slope)
  } else {
    return(0)
  }
}

# Apply the calculate_trend function to find the spending trend for each
customer
spending_trends <- monthly_spending %>%
  group_by(CustomerID) %>%
  do(Spending_Trend = calculate_trend(.)) %>%
  unnest(cols = c(Spending_Trend))

# Assuming customer_data is already defined
customer_data <- customer_data %>%
  left_join(seasonal_buying_patterns, by = "CustomerID") %>%
  left_join(spending_trends, by = "CustomerID")


customer_data <- customer_data %>%
  mutate(Spending_Trend = ifelse(is.na(Spending_Trend), 0, Spending_Trend))

# Display the first few rows of the customer_data dataframe
head(customer_data)

## # A tibble: 6 × 16
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
```

```
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 12 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>, Average_Days_Between_Purchases <dbl>,
## #   Day_Of_Week <ord>, Hour <int>, Is_UK <dbl>, Cancellation_Frequency
<dbl>,
## #   Cancellation_Rate <dbl>, Monthly_Spending_Mean <dbl>,
## #   Monthly_Spending_Std <dbl>, Spending_Trend <dbl>
```

```
print(str(customer_data))
```

```
## tibble [4,358 × 16] (S3: tbl_df/tbl/data.frame)
##  $ CustomerID                    : int [1:4358] 259266 259287 259308
259329 259350 259392 259413 259434 259455 259476 ...
##  $ Days_Since_Last_Purchase      : num [1:4358] 325 2 75 18 310 36 204 232
214 22 ...
##  $ Total_Transactions            : int [1:4358] 2 6 4 1 1 8 1 1 1 3 ...
##  $ Total_Products_Purchased      : int [1:4358] 0 6417 6996 1890 588 1389
60 1590 720 4719 ...
##  $ Total_Spend                   : num [1:4358] 0 14928 5991 6044 1222 ...
##  $ Average_Transaction_Value     : num [1:4358] 0 2488 1498 6044 1222 ...
##  $ total_unique_product          : int [1:4358] 1 85 21 71 16 57 4 58 11
52 ...
##  $ Average_Days_Between_Purchases: num [1:4358] 0.0111 2.1004 10.8751 0 0
...
##  $ Day_Of_Week                   : Ord.factor w/ 7 levels
"Mon"<"Tue"<"Wed"<..: 7 6 2 6 1 7 2 2 6 7 ...
##  $ Hour                          : int [1:4358] 6 8 15 5 12 10 13 9 9 5
...
##  $ Is_UK                         : num [1:4358] 1 0 0 0 0 0 0 0 0 0 ...
##  $ Cancellation_Frequency        : num [1:4358] 1 0 0 0 0 1 0 0 0 0 ...
##  $ Cancellation_Rate             : num [1:4358] 0.5 0 0 0 0 0.125 0 0 0 0
...
##  $ Monthly_Spending_Mean         : num [1:4358] 0 2488 1498 6044 1222 ...
##  $ Monthly_Spending_Std          : num [1:4358] 0 1538 846 0 0 ...
##  $ Spending_Trend                : Named num [1:4358] 0 -93.3 -102.4 0 0
...
##   ..- attr(*, "names")= chr [1:4358] "" "Month" "Month" "" ...
## NULL
```

After this step, we build up all the features needed for the customer clustering. We have 15 features, covering from Purchase recency, frequency, monetart, product diversity, behavioural features, geographic features, cancellation insights and seasonality and trends.

## 3. Outlier Detection - DBSCAN

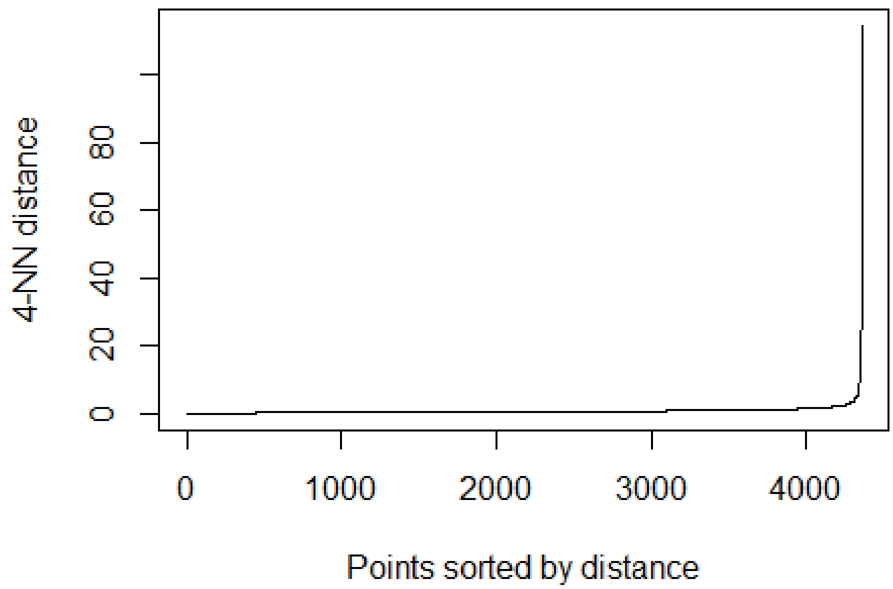
```
# convert day of week to numeric
```

```r
# Check the levels of Day_Of_Week
levels(customer_data$Day_Of_Week)

## [1] "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"

customer_data <- customer_data %>%
  mutate(Day_Of_Week = factor(Day_Of_Week, levels = c("Mon", "Tue", "Wed",
"Thu", "Fri", "Sat", "Sun"), ordered = TRUE))

# Convert Day_Of_Week to numeric ordinal values
customer_data <- customer_data %>%
  mutate(Day_Of_Week = as.numeric(Day_Of_Week))

head(customer_data)

## # A tibble: 6 × 16
##   CustomerID Days_Since_Last_Purchase Total_Transactions
Total_Products_Purcha…¹
##        <int>                    <dbl>              <int>
<int>
## 1     259266                      325                  2
0
## 2     259287                        2                  6
6417
## 3     259308                       75                  4
6996
## 4     259329                       18                  1
1890
## 5     259350                      310                  1
588
## 6     259392                       36                  8
1389
## # ℹ abbreviated name: ¹Total_Products_Purchased
## # ℹ 12 more variables: Total_Spend <dbl>, Average_Transaction_Value <dbl>,
## #   total_unique_product <int>, Average_Days_Between_Purchases <dbl>,
## #   Day_Of_Week <dbl>, Hour <int>, Is_UK <dbl>, Cancellation_Frequency
<dbl>,
## #   Cancellation_Rate <dbl>, Monthly_Spending_Mean <dbl>,
## #   Monthly_Spending_Std <dbl>, Spending_Trend <dbl>
```

Use dbscan to detect outliers

```r
library(dbscan)

# Remove the CustomerID column
customer_data_no_id <- customer_data %>%
  select(-CustomerID)

# Scale the data
```

```r
scaled_customer_data <- scale(customer_data_no_id)
kNNdistplot(scaled_customer_data, 4)
```




```r
summary(scaled_customer_data)

##  Days_Since_Last_Purchase Total_Transactions  Total_Products_Purchased
##  Min.   :-0.9083          Min.   :-0.439059   Min.   :-0.30508
##  1st Qu.:-0.7493          1st Qu.:-0.439059   1st Qu.:-0.20734
##  Median :-0.4113          Median :-0.218556   Median :-0.16158
##  Mean   : 0.0000          Mean   : 0.000000   Mean   : 0.00000
##  3rd Qu.: 0.5033          3rd Qu.: 0.001948   3rd Qu.:-0.03446
##  Max.   : 2.7996          Max.   :26.241881   Max.   :42.10826
##   Total_Spend        Average_Transaction_Value total_unique_product
##  Min.   :-0.03580   Min.   :-0.02532          Min.   :-0.7131
##  1st Qu.:-0.02583   1st Qu.:-0.01629          1st Qu.:-0.5431
##  Median :-0.02348   Median :-0.01571          Median :-0.3123
##  Mean   : 0.00000   Mean   : 0.00000          Mean   : 0.0000
##  3rd Qu.:-0.01718   3rd Qu.:-0.01483          3rd Qu.: 0.1979
##  Max.   :65.90248   Max.   :65.99977          Max.   :19.9241
##  Average_Days_Between_Purchases  Day_Of_Week          Hour
##  Min.   :-0.2963                Min.   :-1.2504   Min.   :-2.3728
##  1st Qu.:-0.2963                1st Qu.:-0.7939   1st Qu.:-0.6432
##  Median :-0.1962                Median :-0.3374   Median :-0.2108
##  Mean   : 0.0000                Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.:-0.0276                3rd Qu.: 1.0321   3rd Qu.: 0.6539
##  Max.   :27.0151                Max.   : 1.4886   Max.   : 3.2482
##      Is_UK         Cancellation_Frequency Cancellation_Rate
```

```
##  Min.   :-3.0617   Min.   :-0.3950        Min.   :-0.6031
##  1st Qu.: 0.3265   1st Qu.:-0.3950        1st Qu.:-0.6031
##  Median : 0.3265   Median :-0.3950        Median :-0.6031
##  Mean   : 0.0000   Mean   : 0.0000        Mean   : 0.0000
##  3rd Qu.: 0.3265   3rd Qu.: 0.1149        3rd Qu.: 0.5242
##  Max.   : 0.3265   Max.   :22.5499        Max.   : 5.0335
##  Monthly_Spending_Mean Monthly_Spending_Std Spending_Trend
##  Min.   :-0.02619      Min.   :-0.28756     Min.   :-30.54207
##  1st Qu.:-0.01695      1st Qu.:-0.28756     1st Qu.:  0.01675
##  Median :-0.01612      Median :-0.19077     Median :  0.04960
##  Mean   : 0.00000      Mean   : 0.00000     Mean   :  0.00000
##  3rd Qu.:-0.01496      3rd Qu.: 0.02122     3rd Qu.:  0.07761
##  Max.   :65.99847      Max.   :29.20810     Max.   : 27.72974
```

```
# Apply DBSCAN
dbscan_result <- dbscan(scaled_customer_data, eps = 2, minPts = 5)

dbscan_result
```

```
## DBSCAN clustering for 4358 objects.
## Parameters: eps = 2, minPts = 5
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 4 cluster(s) and 138 noise points.
##
##    0    1    2    3    4
##  138 3812  381   21    6
##
## Available fields: cluster, eps, minPts, metric, borderPoints
```

```
138/4358
```

```
## [1] 0.0316659
```

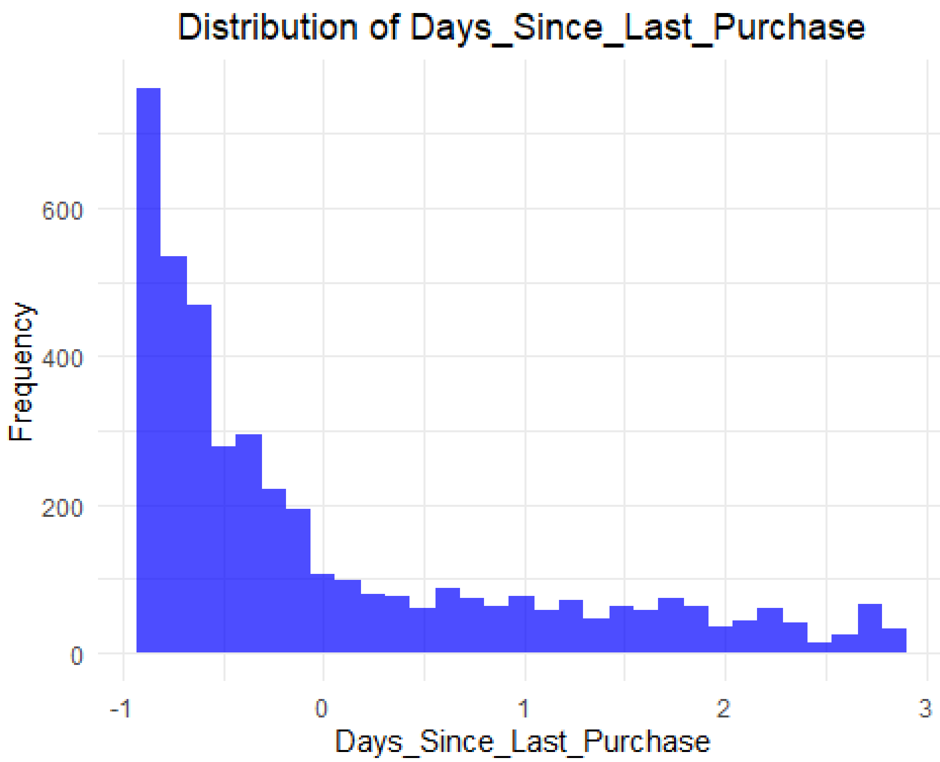
From dbscan result, we detect 138 outliers, outliers percentage 3.17%.

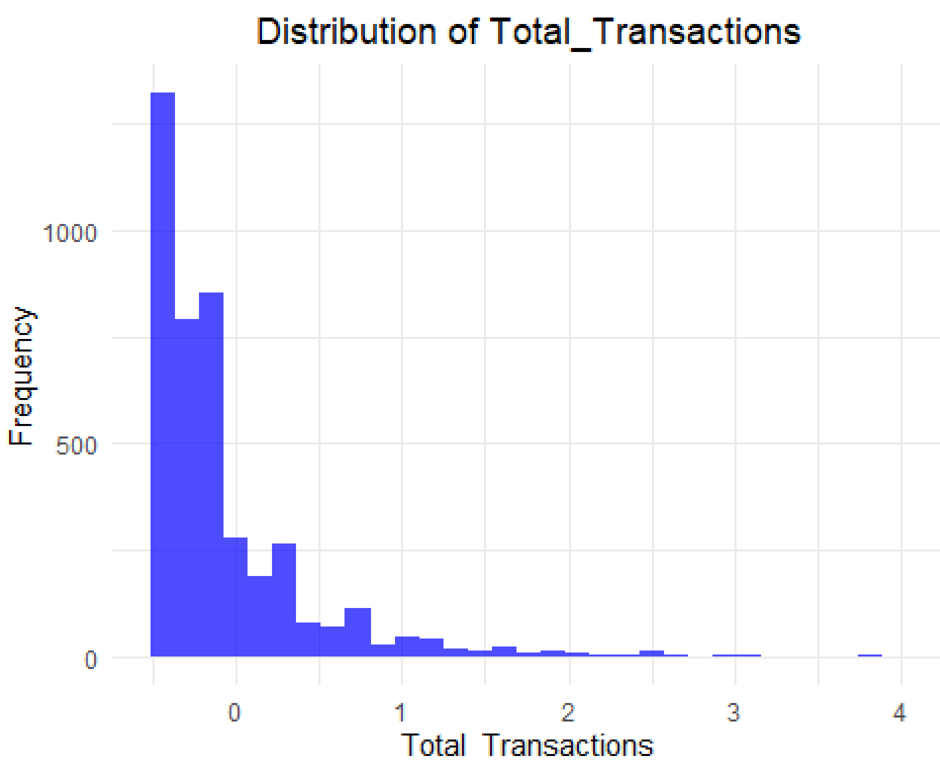
```
# Convert the scaled data back to a dataframe
scaled_customer_data_df <- as.data.frame(scaled_customer_data)

# Add the cluster assignment to the scaled_customer_data dataframe
scaled_customer_data_with_cluster <- scaled_customer_data_df %>%
  mutate(Cluster = dbscan_result$cluster)

# Filter out the outliers (Cluster 0)
scaled_customer_data_no_outliers <- scaled_customer_data_with_cluster %>%
  filter(Cluster != 0) %>%
  select(-Cluster)  #  remove the Cluster column
```

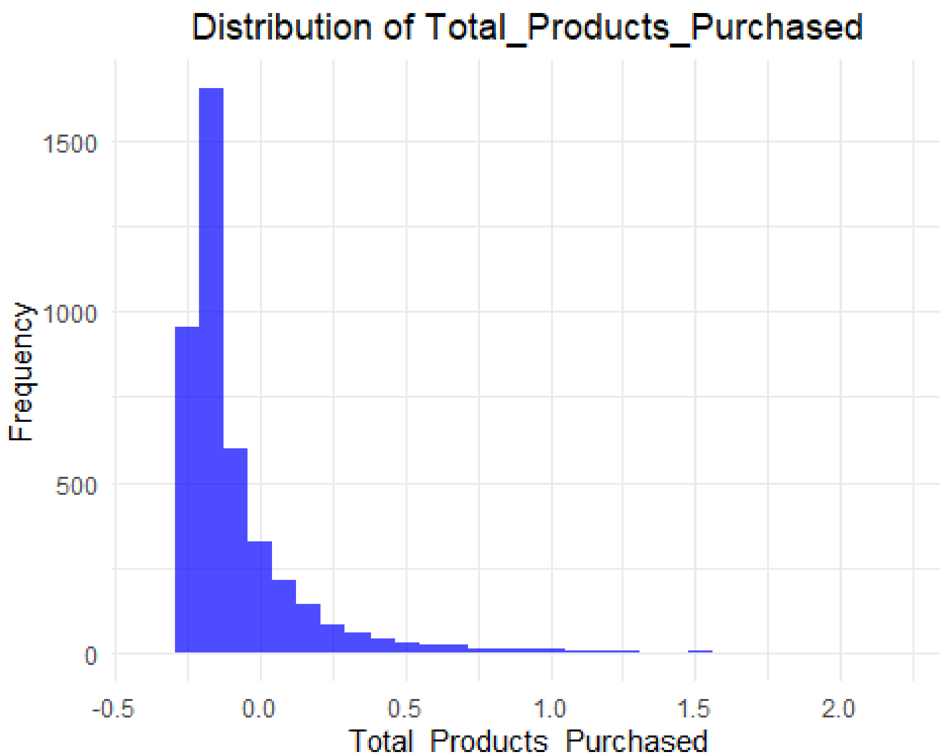
## 4. EDA (distriburion, corr)

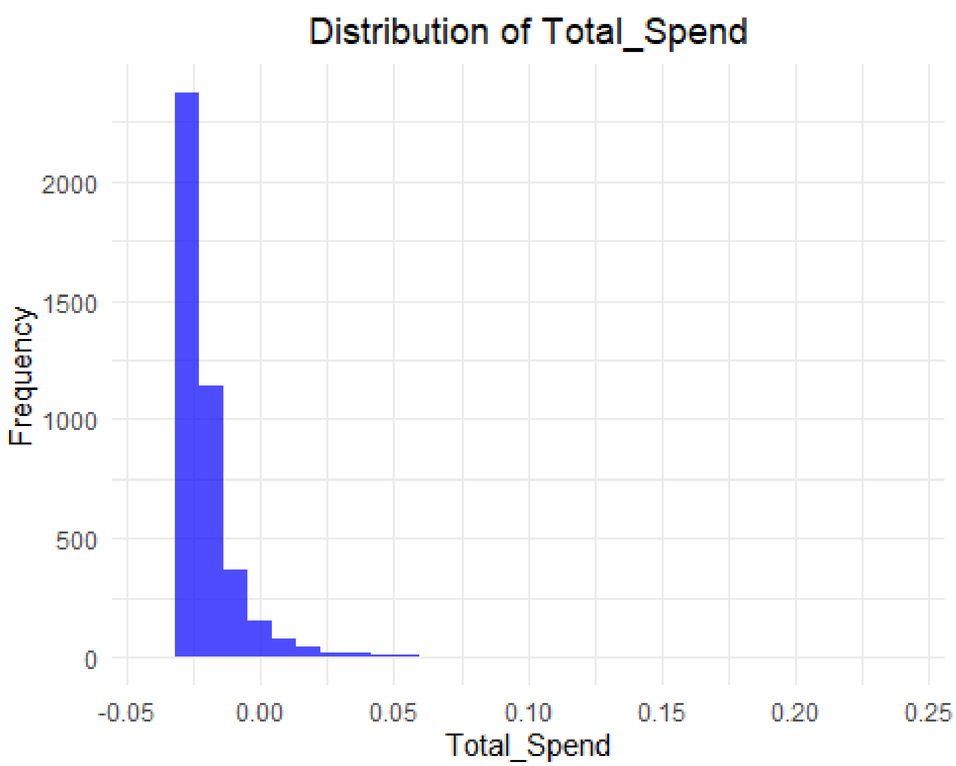
### 4.1 Distribution of each column

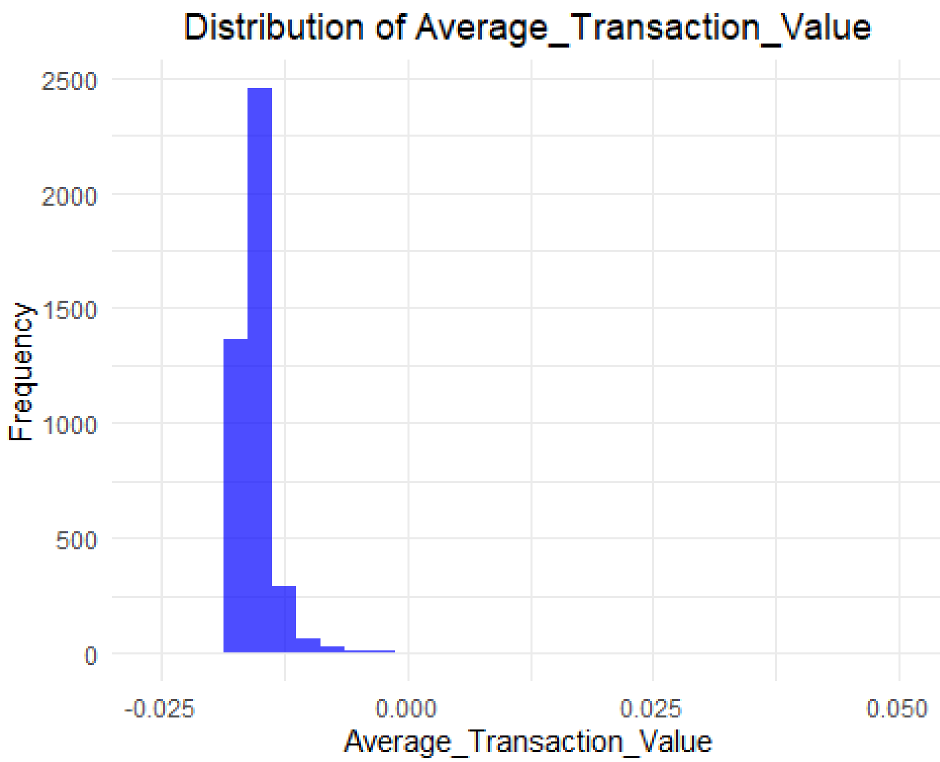
```
# draw plots showing the distribution of each columns

for (col in names(scaled_customer_data_no_outliers)) {
  # Calculate dynamic binwidth
  data_range <- range(scaled_customer_data_no_outliers[[col]], na.rm = TRUE)
  binwidth <- (data_range[2] - data_range[1]) / 30
  p <- ggplot(scaled_customer_data_no_outliers, aes_string(x=col)) +
    geom_histogram(binwidth = binwidth, fill = 'blue', alpha = 0.7) +
    theme_minimal() +
    labs(title = paste("Distribution of", col), x = col, y = "Frequency") +
    theme(plot.title = element_text(hjust = 0.5))
  print(p)
}

## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## ℹ Please use tidy evaluation idioms with `aes()`.
## ℹ See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```


Distribution of Days_Since_Last_Purchase
Frequency
600
400
200
0
-1
0
1
2
3
Days_Since_Last_Purchase

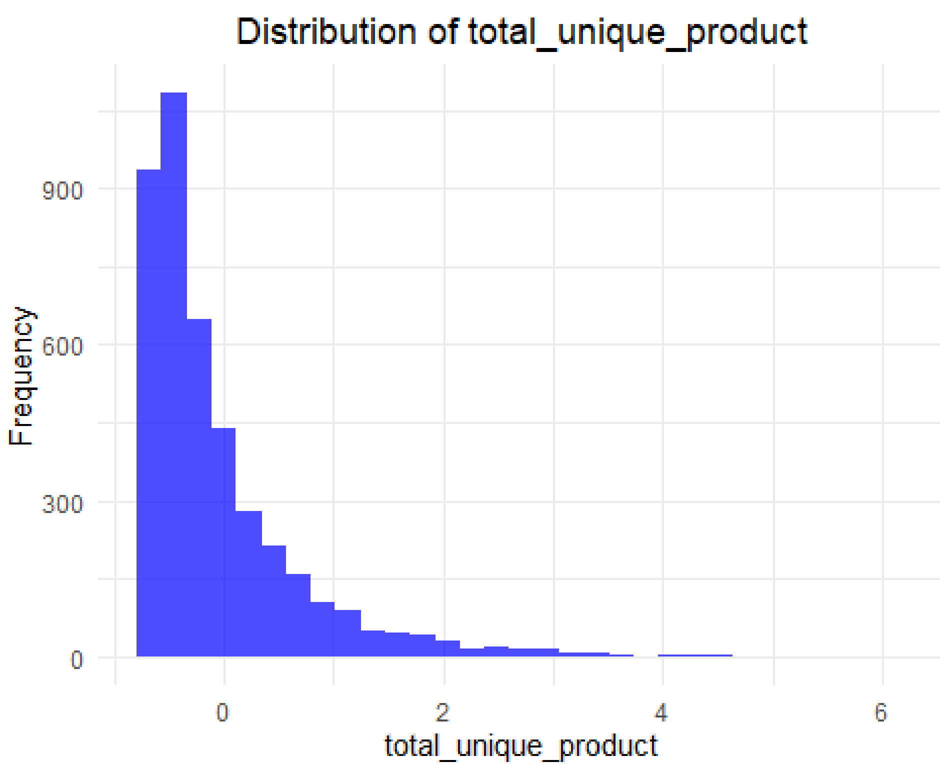

Distribution of Total_Transactions
Frequency
1000
500
0
0
1
2
3
4
Total_Transactions


Distribution of Total_Products_Purchased
Frequency
1500
1000
500
0
-0.5
0.0
0.5
1.0
1.5
2.0
Total_Products_Purchased

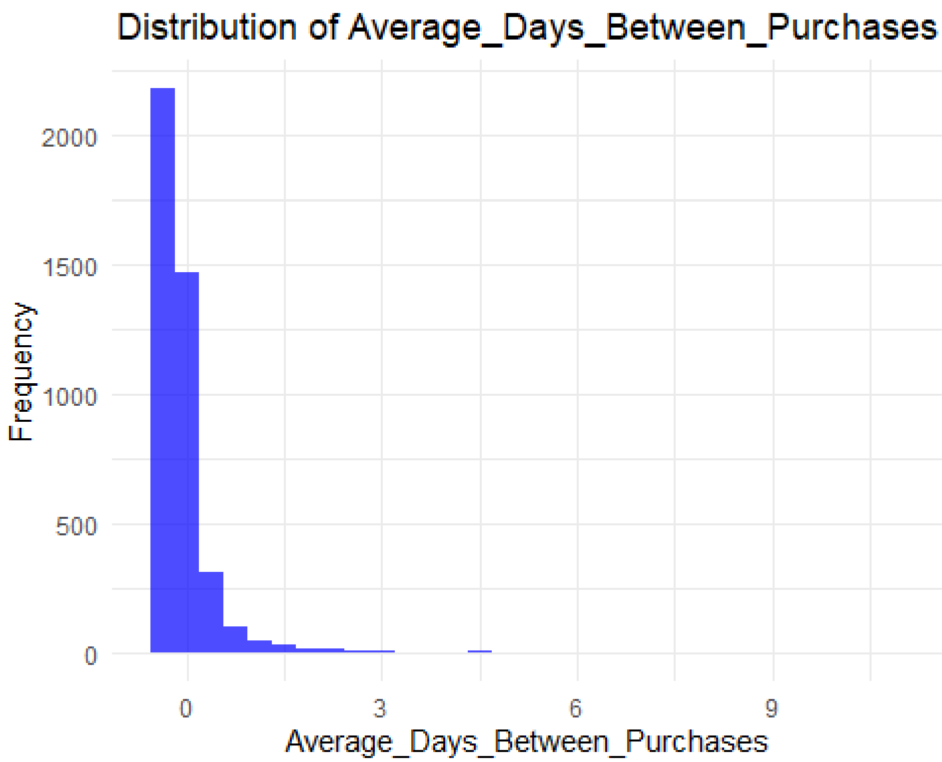

Distribution of Total_Spend
Frequency
2000
1500
1000
500
0
-0.05
0.00
0.05
0.10
0.15
0.20
0.25
Total_Spend

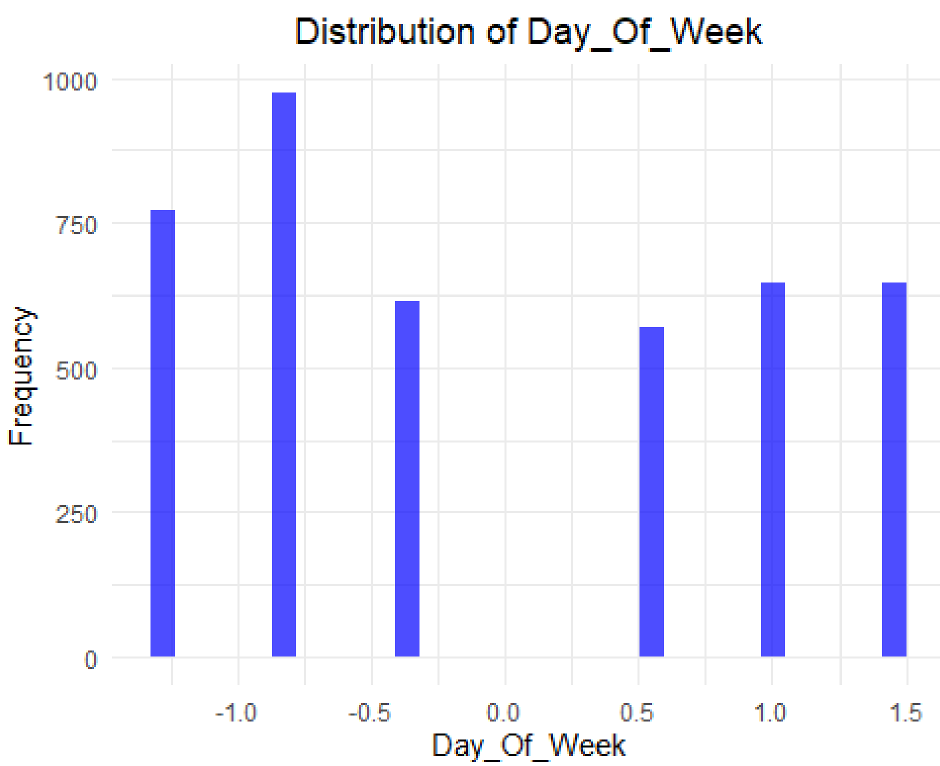


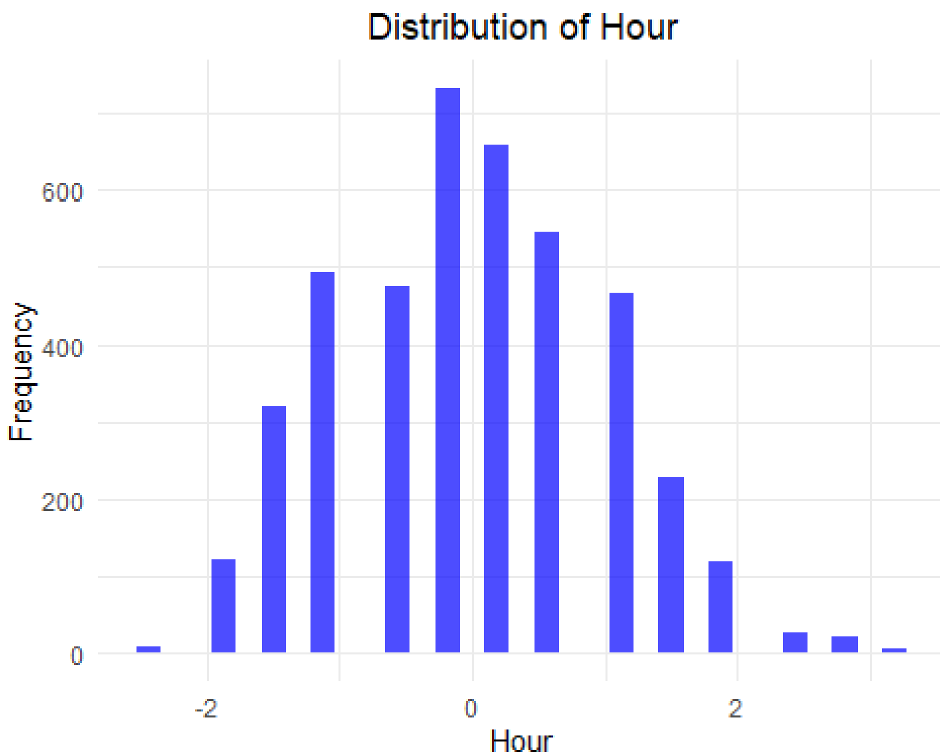
Distribution of Average_Transaction_Value
Frequency
2500
2000
1500
1000
500
0
-0.025
0.000
0.025
0.050
Average_Transaction_Value

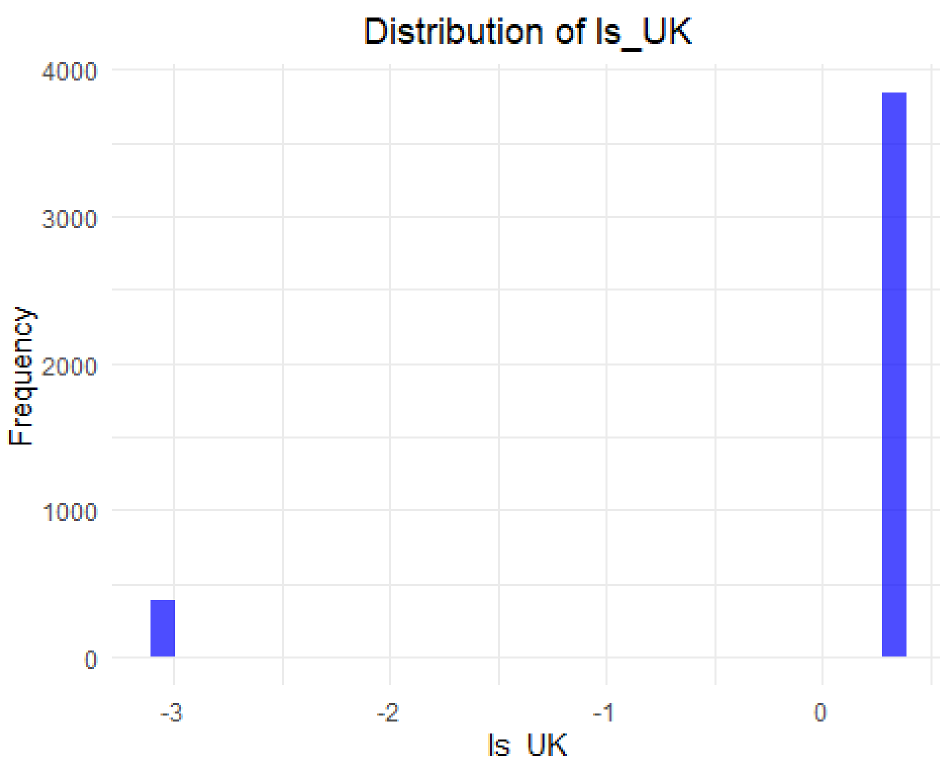



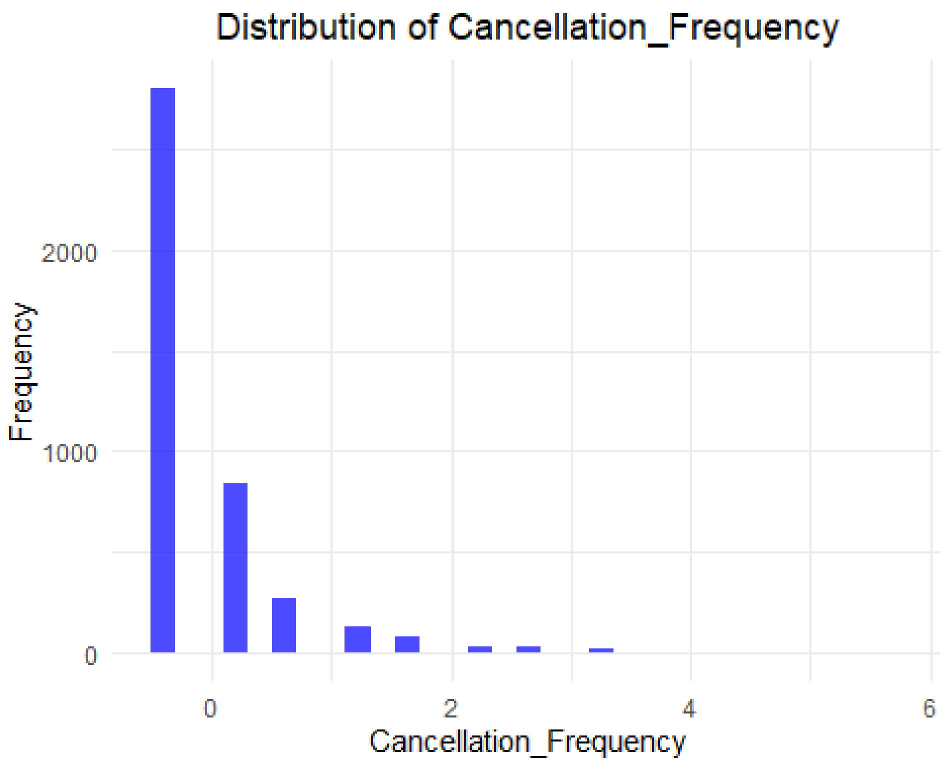
Distribution of total_unique_product
Frequency
900
600
300
0
0
2
4
6
total_unique_product

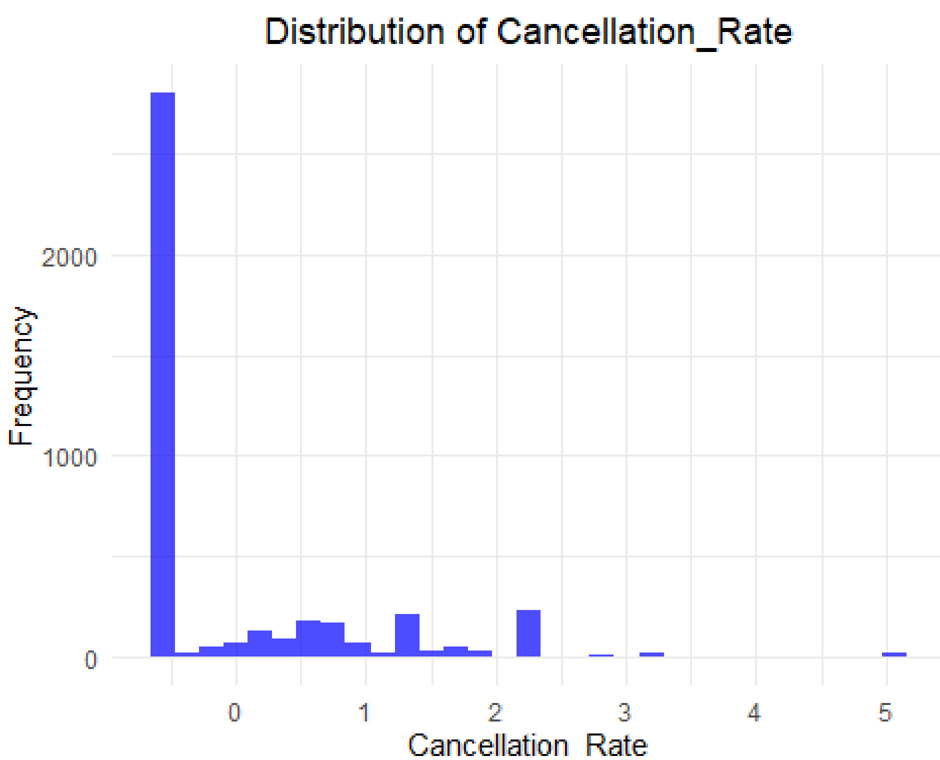

Distribution of Average_Days_Between_Purchases
Frequency
2000
1500
1000
500
0
0
3
6
9
Average_Days_Between_Purchases

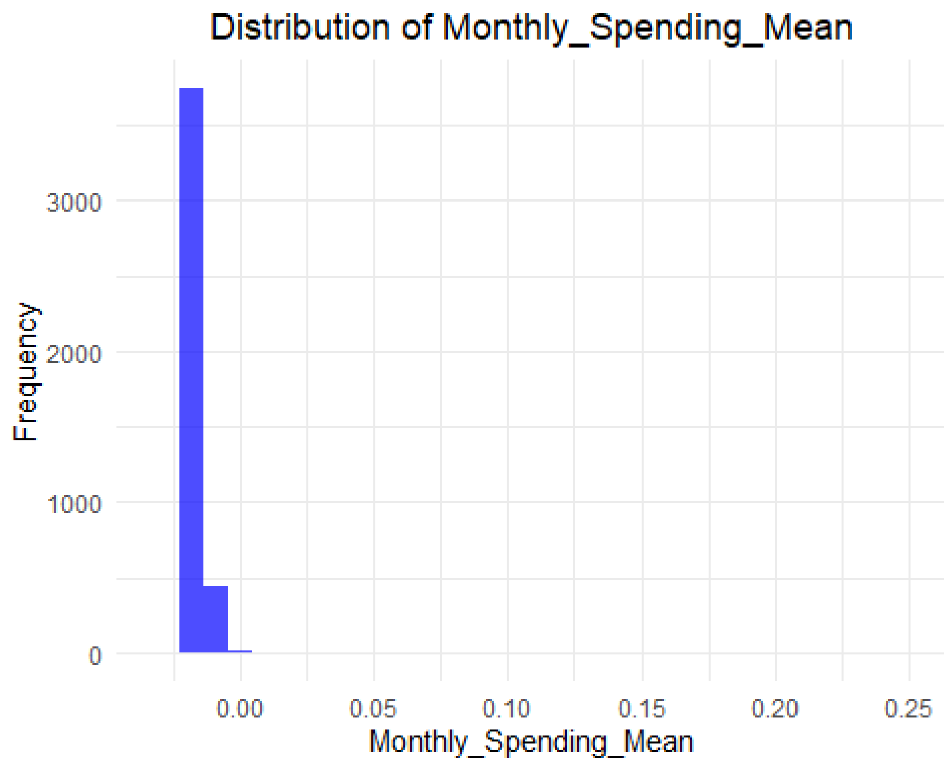

Distribution of Day_Of_Week
Frequency
1000
750
500
250
0
-1.0
-0.5
0.0
0.5
1.0
1.5
Day_Of_Week

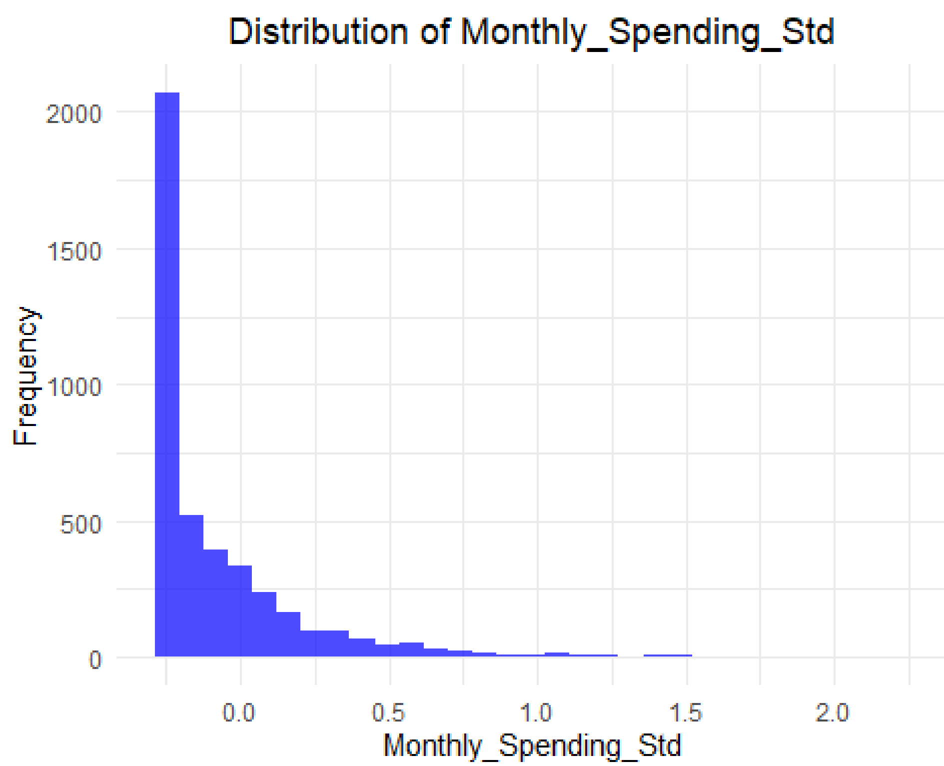

Distribution of Hour
Frequency
Hour
600
400
200
0
-2
0
2

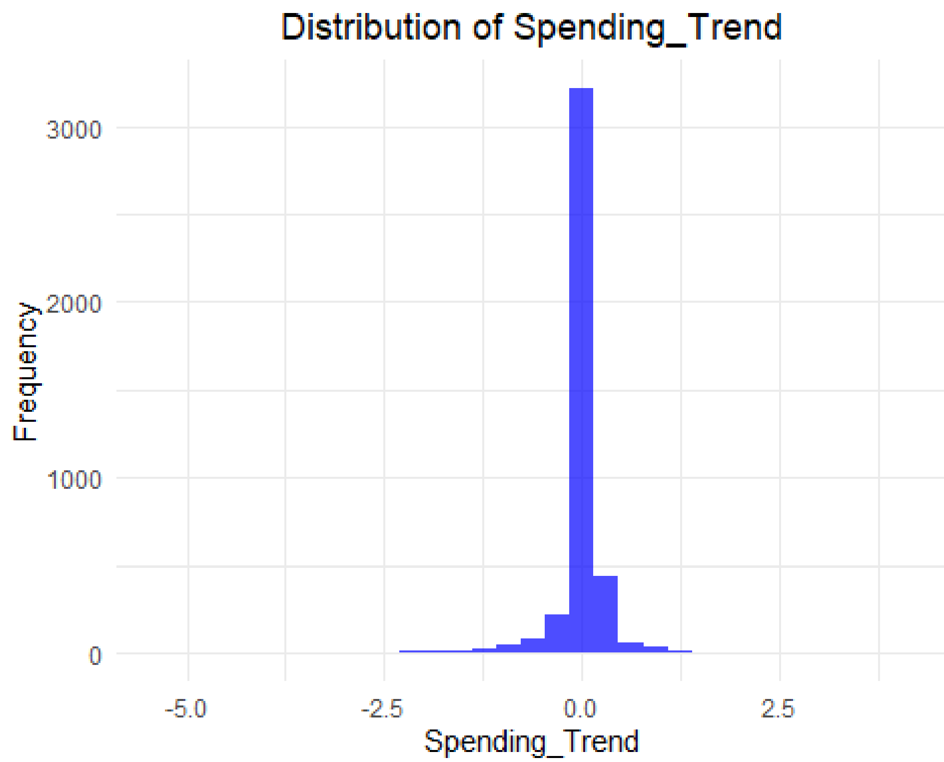

Distribution of Is_UK
Frequency
Is_UK
4000
3000
2000
1000
0
-3
-2
-1
0


Distribution of Cancellation_Frequency
Frequency
0
1000
2000
0
2
4
6
Cancellation_Frequency


Distribution of Cancellation_Rate
Frequency
0
1000
2000
0
1
2
3
4
5
Cancellation_Rate


Distribution of Monthly_Spending_Mean
Frequency
3000
2000
1000
0
0.00
0.05
0.10
0.15
0.20
0.25
Monthly_Spending_Mean


Distribution of Monthly_Spending_Std
Frequency
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
2.0
Monthly_Spending_Std




### 4.2 Correlations

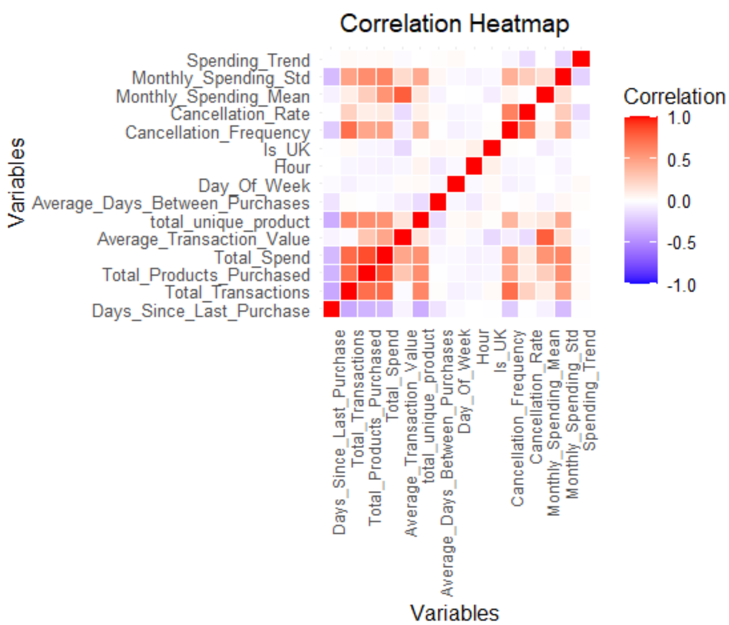
```
# Install and load necessary libraries
#install.packages("reshape2")
library(reshape2)
library(ggplot2)
library(dplyr)

data <- scaled_customer_data_no_outliers

# Calculate the correlation matrix
numeric_columns <- data %>% select_if(is.numeric)
correlation_matrix <- cor(numeric_columns, use = "complete.obs")

correlation_matrix <- round(correlation_matrix,2)
# Print the correlation matrix
print(correlation_matrix)

##                               Days_Since_Last_Purchase Total_Transactions
## Days_Since_Last_Purchase                          1.00              -0.37
## Total_Transactions                               -0.37               1.00
## Total_Products_Purchased                         -0.32               0.72
## Total_Spend                                      -0.30               0.74
## Average_Transaction_Value                        -0.05              -0.02
## total_unique_product                             -0.35               0.61
## Average_Days_Between_Purchases                   -0.12               0.01
## Day_Of_Week                                      -0.02              -0.06
```

```
## Hour                                            0.00                 -0.04
## Is_UK                                          -0.01                  0.03
## Cancellation_Frequency                         -0.22                  0.72
## Cancellation_Rate                              -0.01                  0.24
## Monthly_Spending_Mean                          -0.06                  0.09
## Monthly_Spending_Std                           -0.29                  0.48
## Spending_Trend                                 -0.01                  0.03
##                                 Total_Products_Purchased Total_Spend
## Days_Since_Last_Purchase                       -0.32       -0.30
## Total_Transactions                              0.72        0.74
## Total_Products_Purchased                        1.00        0.85
## Total_Spend                                     0.85        1.00
## Average_Transaction_Value                       0.30        0.46
## total_unique_product                            0.58        0.56
## Average_Days_Between_Purchases                 -0.01       -0.03
## Day_Of_Week                                    -0.03       -0.03
## Hour                                           -0.05       -0.06
## Is_UK                                          -0.04       -0.05
## Cancellation_Frequency                          0.46        0.49
## Cancellation_Rate                               0.09        0.11
## Monthly_Spending_Mean                           0.26        0.55
## Monthly_Spending_Std                            0.58        0.62
## Spending_Trend                                  0.02        0.03
##                                 Average_Transaction_Value
total_unique_product
## Days_Since_Last_Purchase                        -0.05                -
0.35
## Total_Transactions                              -0.02
0.61
## Total_Products_Purchased                         0.30
0.58
## Total_Spend                                      0.46
0.56
## Average_Transaction_Value                        1.00
0.14
## total_unique_product                             0.14
1.00
## Average_Days_Between_Purchases                  -0.07                -
0.15
## Day_Of_Week                                      0.02
0.03
## Hour                                            -0.04
0.06
## Is_UK                                           -0.16
0.01
## Cancellation_Frequency                          -0.07
0.38
## Cancellation_Rate                               -0.15
0.08
## Monthly_Spending_Mean                            0.79
```

```
0.13
## Monthly_Spending_Std                                      0.19
0.44
## Spending_Trend                                           -0.02
0.00
##                                  Average_Days_Between_Purchases Day_Of_Week
Hour
## Days_Since_Last_Purchase                                  -0.12       -0.02
0.00
## Total_Transactions                                         0.01       -0.06
-0.04
## Total_Products_Purchased                                  -0.01       -0.03
-0.05
## Total_Spend                                               -0.03       -0.03
-0.06
## Average_Transaction_Value                                 -0.07        0.02
-0.04
## total_unique_product                                      -0.15        0.03
0.06
## Average_Days_Between_Purchases                             1.00       -0.04
-0.09
## Day_Of_Week                                               -0.04        1.00
-0.03
## Hour                                                      -0.09       -0.03
1.00
## Is_UK                                                      0.04        0.03
0.08
## Cancellation_Frequency                                    -0.01       -0.06
-0.04
## Cancellation_Rate                                          0.02       -0.04
-0.03
## Monthly_Spending_Mean                                     -0.05        0.00
-0.01
## Monthly_Spending_Std                                       0.04       -0.03
-0.05
## Spending_Trend                                             0.01        0.03
0.00
##                                  Is_UK Cancellation_Frequency
Cancellation_Rate
## Days_Since_Last_Purchase       -0.01                  -0.22             -
0.01
## Total_Transactions              0.03                   0.72
0.24
## Total_Products_Purchased       -0.04                   0.46
0.09
## Total_Spend                    -0.05                   0.49
0.11
## Average_Transaction_Value      -0.16                  -0.07             -
0.15
## total_unique_product            0.01                   0.38
```

```
0.08
## Average_Days_Between_Purchases  0.04                 -0.01            
0.02
## Day_Of_Week                      0.03                 -0.06            -
0.04
## Hour                             0.08                 -0.04            -
0.03
## Is_UK                            1.00                  0.01            
0.00
## Cancellation_Frequency           0.01                  1.00            
0.64
## Cancellation_Rate                0.00                  0.64            
1.00
## Monthly_Spending_Mean           -0.07                  0.06            -
0.01
## Monthly_Spending_Std            -0.03                  0.41            
0.27
## Spending_Trend                   0.00                 -0.04            -
0.15
##                                 Monthly_Spending_Mean Monthly_Spending_Std
## Days_Since_Last_Purchase                        -0.06                -0.29
## Total_Transactions                               0.09                 0.48
## Total_Products_Purchased                         0.26                 0.58
## Total_Spend                                      0.55                 0.62
## Average_Transaction_Value                        0.79                 0.19
## total_unique_product                             0.13                 0.44
## Average_Days_Between_Purchases                  -0.05                 0.04
## Day_Of_Week                                      0.00                -0.03
## Hour                                            -0.01                -0.05
## Is_UK                                           -0.07                -0.03
## Cancellation_Frequency                           0.06                 0.41
## Cancellation_Rate                               -0.01                 0.27
## Monthly_Spending_Mean                            1.00                 0.17
## Monthly_Spending_Std                             0.17                 1.00
## Spending_Trend                                   0.00                -0.19
##                                 Spending_Trend
## Days_Since_Last_Purchase                 -0.01
## Total_Transactions                        0.03
## Total_Products_Purchased                  0.02
## Total_Spend                               0.03
## Average_Transaction_Value                -0.02
## total_unique_product                      0.00
## Average_Days_Between_Purchases            0.01
## Day_Of_Week                               0.03
## Hour                                      0.00
## Is_UK                                     0.00
## Cancellation_Frequency                   -0.04
## Cancellation_Rate                        -0.15
## Monthly_Spending_Mean                     0.00
```

```
## Monthly_Spending_Std                          -0.19
## Spending_Trend                                 1.00

# Melt the correlation matrix to long format
melted_corr_matrix <- melt(correlation_matrix)

# Create the heatmap
heatmap <- ggplot(data = melted_corr_matrix, aes(x = Var1, y = Var2, fill =
value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
                       midpoint = 0, limit = c(-1, 1), space = "Lab",
                       name = "Correlation") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 1,
                                   size = 8, hjust = 1)) +
  coord_fixed() +
  labs(title = "Correlation Heatmap",
       x = "Variables",
       y = "Variables") +
  theme(plot.title = element_text(hjust = 0.5))

# Display the heatmap
print(heatmap)
```




Here is a table for displaying the strong correlations:

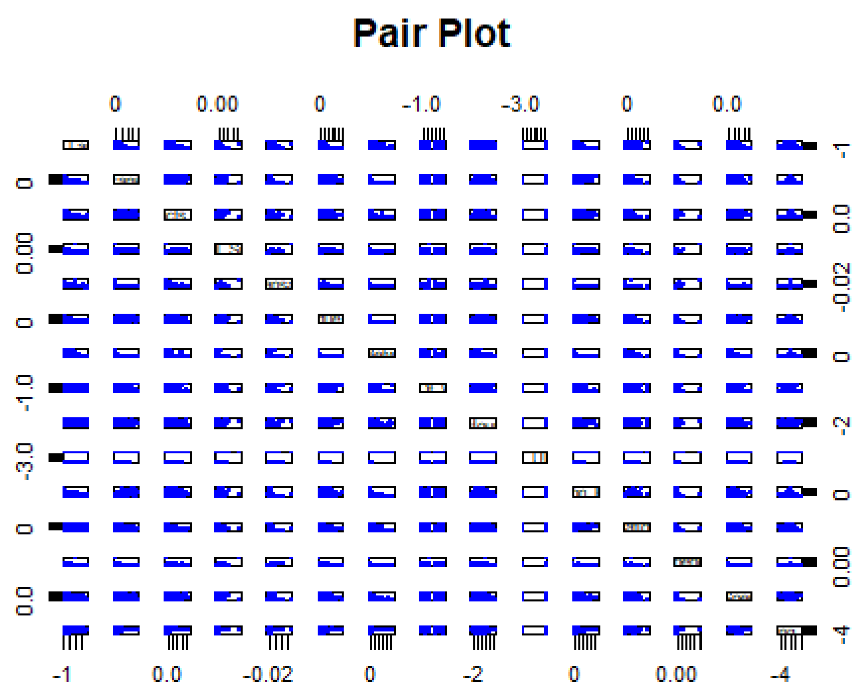
*Strong Correlations (|correlation| > 0.7)*

| Variable 1 | Variable 2 | Correlation |
|---|---|---|
| Total_Transactions | Total_Products_Purchased | 0.72 |

| Variable 1 | Variable 2 | Correlation |
|---|---|---|
| Total_Transactions | Total_Spend | 0.74 |
| Total_Products_Purchased | Total_Spend | 0.85 |
| Cancellation_Frequency | Total_Transactions | 0.72 |

From correlation result, we can find there are multicollinearity between the columns in the dataset.

```r
# Create the pair plot with smaller point size
pairs(scaled_customer_data_no_outliers,
      main = "Pair Plot",
      col = 'blue',
      pch = 19,  # Solid circle
      cex = 0.05)  # Adjust the point size
```




## 5. Dimension Reduction

**Multicollinearity Detected**: In the previous steps, we identified that our dataset contains multicollinear features. Dimensionality reduction can help us remove redundant information and alleviate the multicollinearity issue.

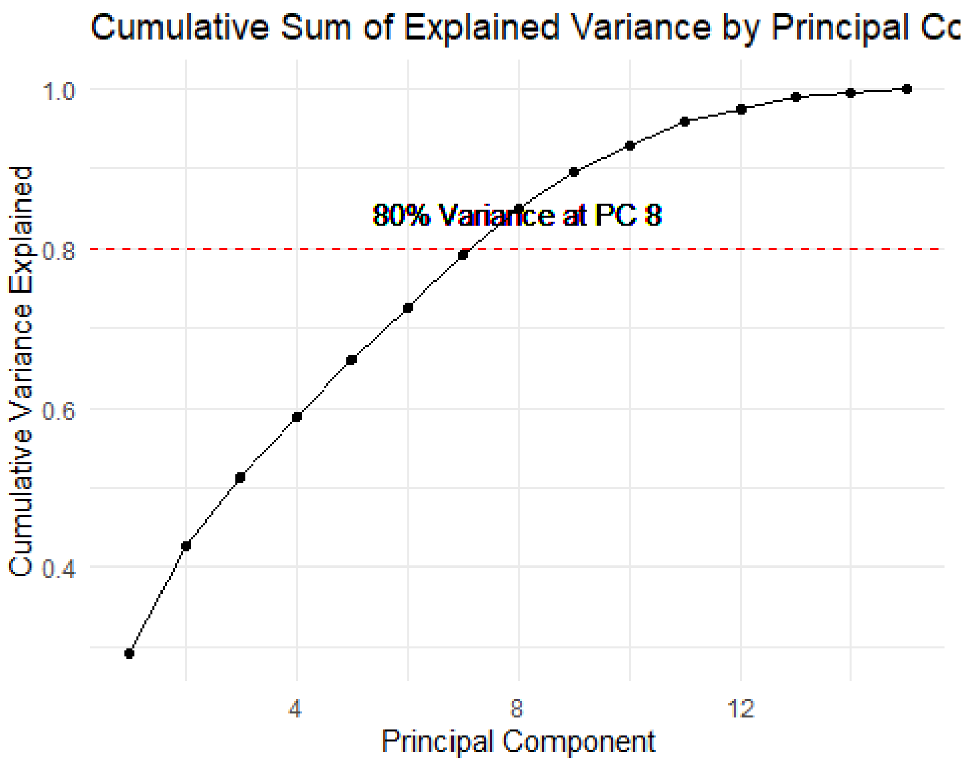
**Better Clustering with K-means**: Since K-means is a distance-based algorithm, having a large number of features can sometimes dilute the meaningful underlying patterns in the data. By reducing the dimensionality, we can help K-means to find more compact and well-separated clusters.

**Noise Reduction**: By focusing only on the most important features, we can potentially remove noise in the data, leading to more accurate and stable clusters.

**Enhanced Visualization**: In the context of customer segmentation, being able to visualize customer groups in two or three dimensions can provide intuitive insights. Dimensionality reduction techniques can facilitate this by reducing the data to a few principal components which can be plotted easily.

**Improved Computational Efficiency**: Reducing the number of features can speed up the computation time during the modeling process, making our clustering algorithm more efficient.

```
# Perform PCA
pca_result <- prcomp(scaled_customer_data_no_outliers, center = TRUE, scale.
= TRUE)

# Summary of PCA
pca_summary <- summary(pca_result)

# Extract the proportion of variance explained by each principal component
explained_variance <- pca_summary$importance[2, ]

# Calculate the cumulative sum of explained variance
cumulative_variance <- cumsum(explained_variance)

# Create a data frame for plotting
cumsum_df <- data.frame(
  Principal_Component = seq_along(cumulative_variance),
  Cumulative_Variance = cumulative_variance
)

# Plot the cumulative sum of explained variance
ggplot(cumsum_df, aes(x = Principal_Component, y = Cumulative_Variance)) +
  geom_line() +
  geom_point() +
  geom_hline(yintercept = 0.80, linetype = "dashed", color = "red") +
  geom_text(aes(x = which(cumulative_variance >= 0.80)[1],
                y = 0.80,
                label = paste("80% Variance at PC", which(cumulative_variance
>= 0.80)[1])),
            vjust = -1) +
  labs(title = "Cumulative Sum of Explained Variance by Principal
Components",
       x = "Principal Component",
       y = "Cumulative Variance Explained") +
  theme_minimal()
```




```
# Determine the number of principal components that explain at least 80% of
the variance
num_pcs <- which(cumulative_variance >= 0.80)[1]
print(paste("Number of principal components explaining at least 80%
variance:", num_pcs))

## [1] "Number of principal components explaining at least 80% variance: 8"

# Transform the data using the first 8 principal components
pca_transformed_data <- as.data.frame(pca_result$x[, 1:num_pcs])

head(pca_transformed_data)

##          PC1        PC2         PC3         PC4        PC5        PC6
PC7
## 1 -1.6156444 -1.4917160 -1.96635660  0.55776328  1.8894643 -1.1560469
1.597996
## 2  2.0227160  1.6552167  0.11973135 -0.75817022  1.3474102 -0.5139558 -
2.531017
## 3  0.3771541  0.8657006  0.10097042 -0.02673209 -0.7090584  1.8653254 -
2.297445
## 4  0.4109115  3.9024948 -0.99405719 -1.04756773  2.1340802 -0.6271568 -
1.522682
## 5 -1.8567904  0.9810792 -1.10346344  0.99466934  0.9700485  2.3772288 -
1.833942
## 6  0.4529421 -0.1756536  0.05558197 -0.13477555  1.7888538 -0.3491593 -
1.770549
```

```
##           PC8
## 1  0.2776453
## 2 -1.3530979
## 3 -2.8745446
## 4 -1.2309946
## 5 -1.2733967
## 6 -2.5438438
```

## 6. Assessing Clustering Tendency

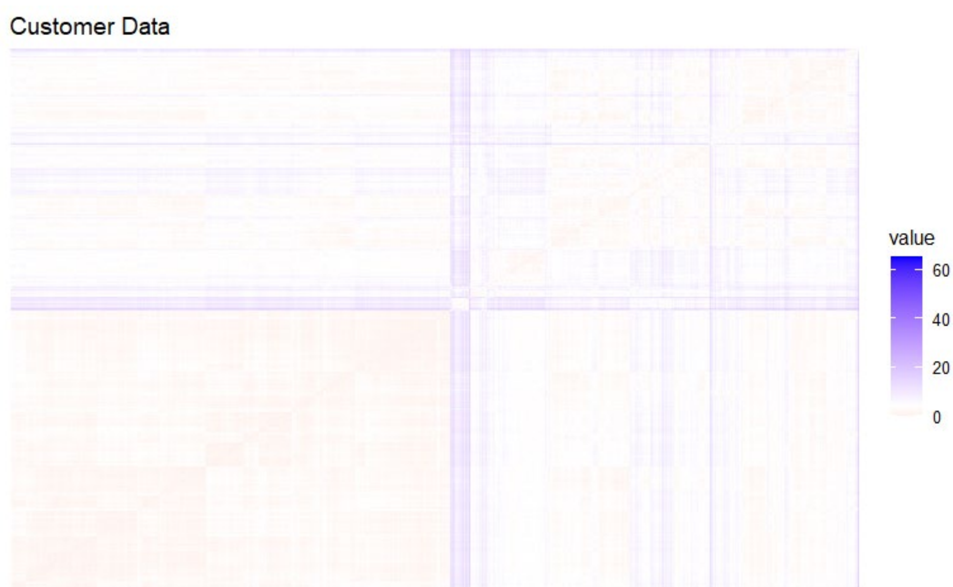
### 6.1 Hopkins analysis

check pca transformed customer data clustering tendency.

```
library(hopkins)
hopkins::hopkins(pca_transformed_data)

## [1] 1
```

### 6.2 VAT, visual assessment of cluster tendency

```
library(factoextra)
plot_vat = fviz_dist(dist(pca_transformed_data),show_labels = FALSE)+
  labs(title = 'Customer Data')
```



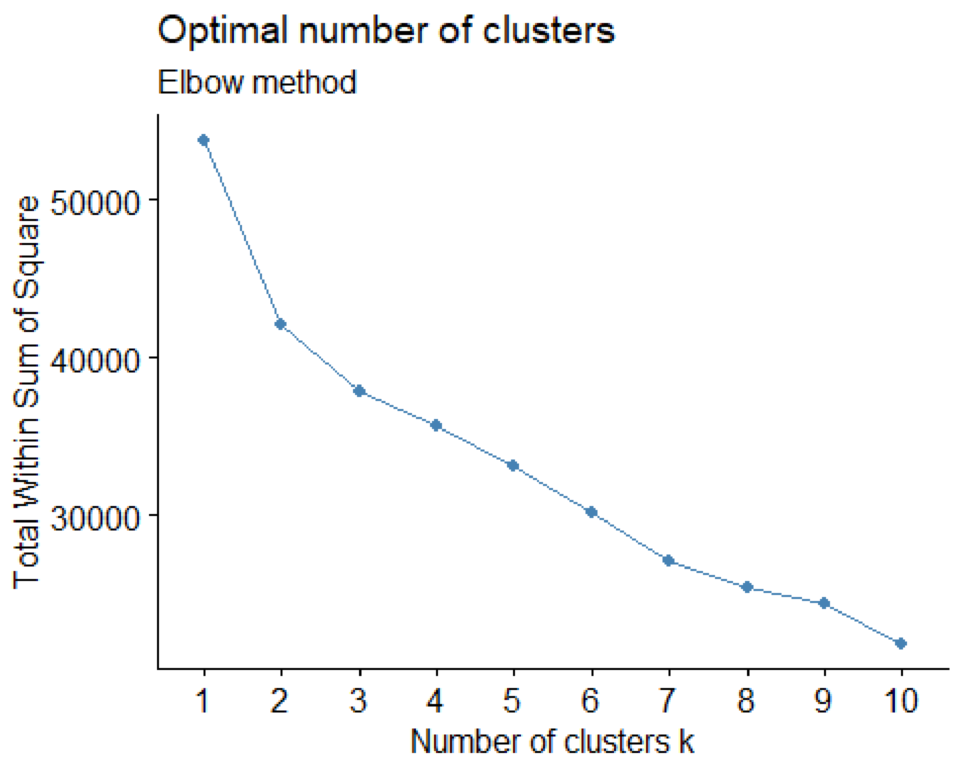

We get hopkins statistics of 1 which means this dataset is highly clusterable. From the VAT, we can see there are clear separated areas, suggesting this dataset is clusterable.

## 7. kmeans

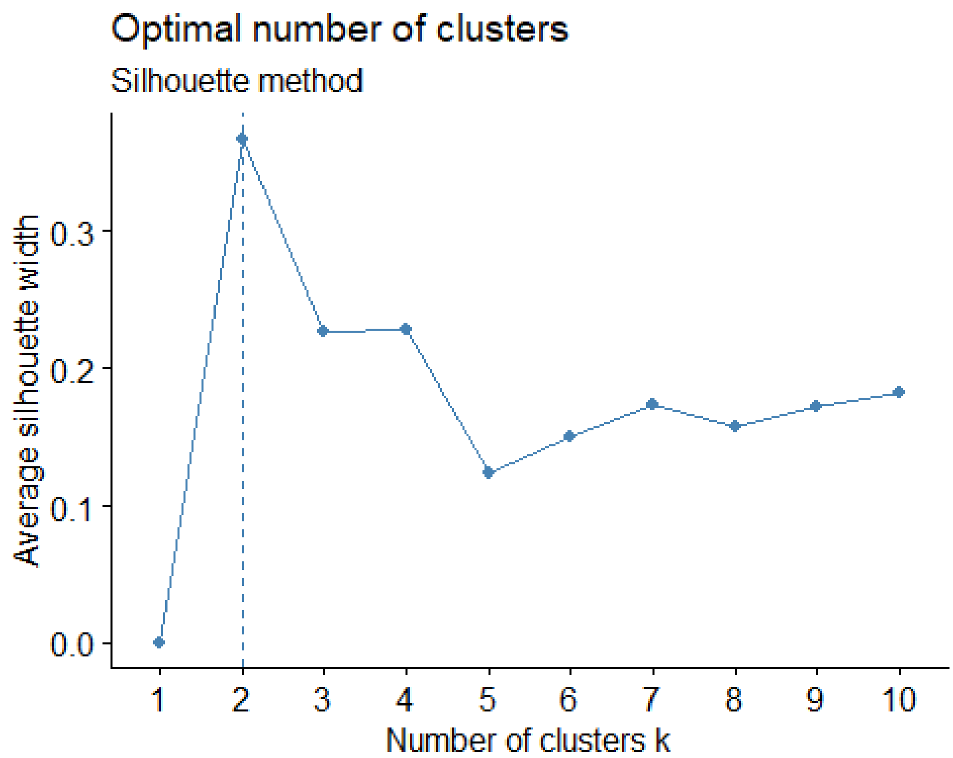
### 7.1 Optimal cluster numbers

Use Elbow method, Silhouette method and Gap to find the optimal K.

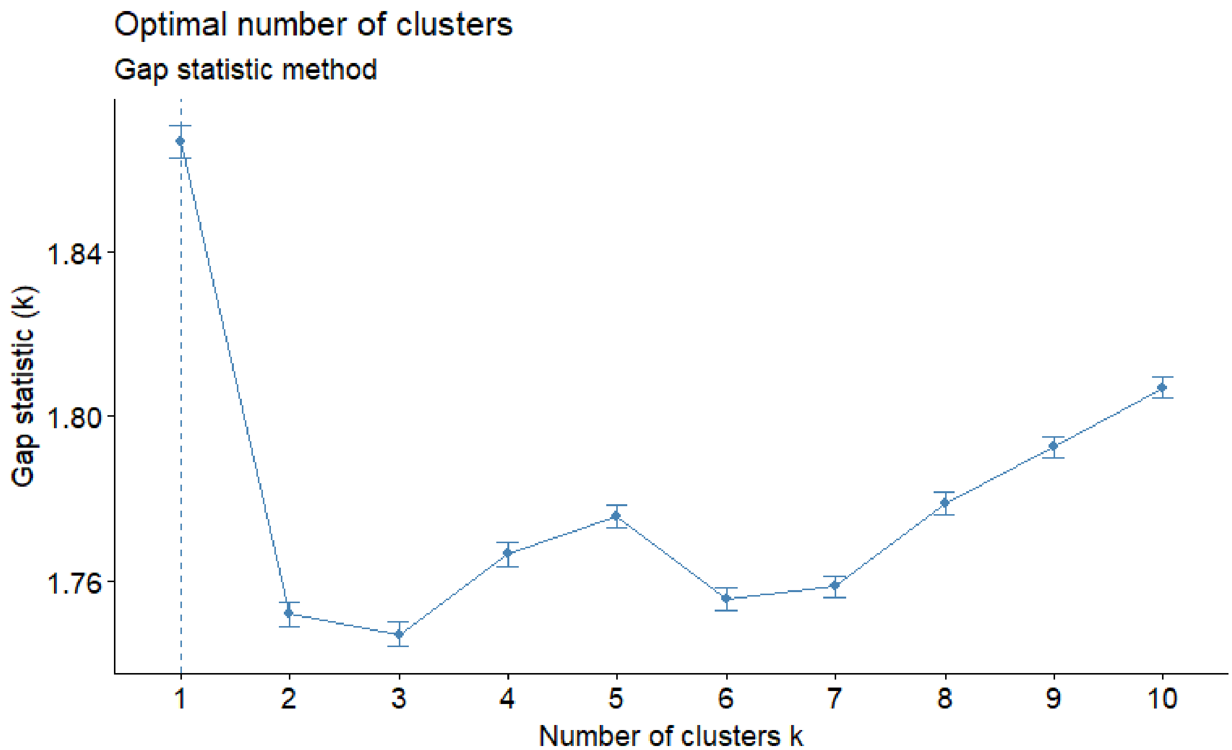
```
# Elbow method
fviz_nbclust(pca_transformed_data, kmeans, method = "wss") +
labs(subtitle = "Elbow method")
```




```
# Silhouette method
fviz_nbclust(pca_transformed_data, kmeans, method = "silhouette")+
labs(subtitle = "Silhouette method")
```




```
# Gap statistic
set.seed(123)
fviz_nbclust(pca_transformed_data, kmeans, nstart = 25, method = "gap_stat",
nboot = 10)+
labs(subtitle = "Gap statistic method")
```




Elbow method suggests k=2; Silhouette method suggests k=2; Gap statistics suggests k=1.

### 7.2. Kmeans clustering result plot

use kmeans to cluser the dataset and draw 2-D cluster plot

```
km.res <- kmeans(pca_transformed_data, 2, nstart = 25)
print(km.res)

## K-means clustering with 2 clusters of sizes 3444, 776
##
## Cluster means:
##           PC1         PC2          PC3         PC4           PC5
PC6
## 1 -0.7872819  0.04858112  0.005518949 -0.02555075 -0.0004397299
0.0004493253
## 2  3.4940707 -0.21561001 -0.024493892  0.11339791  0.0019515850 -
0.0019941704
##          PC7         PC8
## 1  0.03044076 -0.02228919
## 2 -0.13510050  0.09892265
##
## Clustering vector:
##    [1] 1 2 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 2 1 1 1
1 1 1
##   [38] 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 2 1 2 1 2 2 1
1 2 1
```

```
##   [75] 1 1 1 2 1 2 1 1 1 2 2 2 1 1 1 1 1 1 1 2 2 1 2 1 1 2 2 2 2 1 1 2 1 1
1 2 1
##  [112] 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 1
##  [149] 1 1 1 2 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1
2 2 1
##  [186] 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1
1 1 1
##  [223] 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 1 2 1 2 1 2 1 2 1 2 1
1 1 1
##  [260] 1 1 1 2 1 1 1 1 2 1 2 1 2 1 2 1 1 1 1 1 2 2 1 1 1 1 2 2 1 1 2 1 1 1
1 1 1
##  [297] 1 1 1 2 1 2 1 2 1 2 2 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
##  [334] 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1
2 1 1
##  [371] 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1
2 1 1
##  [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 1 2 1 2 1 1 1
1 2 1
##  [445] 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 2 1 1
2 2 1
##  [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 2
##  [519] 1 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2
2 2 1
##  [556] 2 1 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 2 2
1 2 1
##  [593] 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1
1 1 1
##  [630] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2
1 1 1
##  [667] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 1 2 1 2 1 1 1 1 2 1
1 1 2
##  [704] 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 2 1 2 1 1
1 1 1
##  [741] 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 2 1 2 1 1 1 2
1 1 2
##  [778] 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1
##  [815] 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1
1 1 2
##  [852] 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2
1 1 1
##  [889] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1
1 1 2
##  [926] 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 2 1
1 1 1
##  [963] 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 2
1 1 1
```

```
## [1000] 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 1
## [1037] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 2 2 2 1 1 1 1 2 1 1 1 1 2 2 2
1 1 1
## [1074] 1 1 1 2 2 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 2 1 2 2 1 1 1 1 1 1 2 1
1 1 1
## [1111] 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2
1 1 1
## [1148] 1 1 2 1 2 1 2 1 1 1 1 2 1 1 2 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1
1 2 2
## [1185] 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2 1 2 2 1 1 1 1 2 1 1 1
1 1 1
## [1222] 1 1 1 1 2 1 1 1 2 1 1 2 1 2 2 1 2 2 1 1 2 1 2 2 1 1 1 1 2 1 2 1 1 1
1 1 2
## [1259] 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1
2 1 1
## [1296] 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 2 2 1 1
2 1 1
## [1333] 2 1 1 1 2 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1
1 1 2
## [1370] 2 2 1 1 2 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1
1 1 1
## [1407] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 2
1 2 1
## [1444] 1 1 2 1 1 2 1 1 1 2 2 1 2 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 2 2 1 1
1 1 1
## [1481] 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1
1 1 1
## [1518] 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1
1 1 1
## [1555] 2 1 2 2 1 1 1 1 1 1 2 1 2 2 1 2 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2
1 1 1
## [1592] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1
1 1 1
## [1629] 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1
1 2 2
## [1666] 1 2 1 1 2 1 2 1 2 1 1 1 2 1 1 1 1 1 1 2 1 2 2 2 1 1 2 1 1 1 1 1 1 1
1 1 2
## [1703] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 1 2
## [1740] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 2 1 1 1 1 1
1 1 1
## [1777] 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1 1 1
1 1 1
## [1814] 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1
1 1 1
## [1851] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1
1 1 1
## [1888] 2 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1
1 1 1
```

```
## [1925] 2 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
1 2 1
## [1962] 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1
1 1 1
## [1999] 2 1 1 1 1 1 2 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1
## [2036] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 1
## [2073] 1 1 1 1 1 1 2 1 1 1 1 2 2 1 2 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2
1 1 1
## [2110] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2
1 1 2
## [2147] 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
2 1 1
## [2184] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1
## [2221] 2 1 2 2 1 1 2 1 2 2 1 2 1 1 1 2 1 1 1 1 1 1 2 2 1 1 2 1 2 1 1 1 1 1
2 1 2
## [2258] 1 2 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2 1 1
1 1 1
## [2295] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 2 1 1
2 1 2
## [2332] 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2
1 2 1
## [2369] 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1
1 1 1
## [2406] 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1
## [2443] 2 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 2 1 1 2 1 1 1 1 1 1 1 1
2 1 1
## [2480] 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1
1 1 1
## [2517] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1
1 2 1
## [2554] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 2 1 1 1 1 1
1 1 1
## [2591] 2 2 2 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [2628] 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2
1 1 1
## [2665] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1
1 1 1
## [2702] 2 1 1 2 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1
1 1 2
## [2739] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2
1 1 1
## [2776] 1 1 2 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 2 1
1 2 1
## [2813] 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1
1 1 2
```

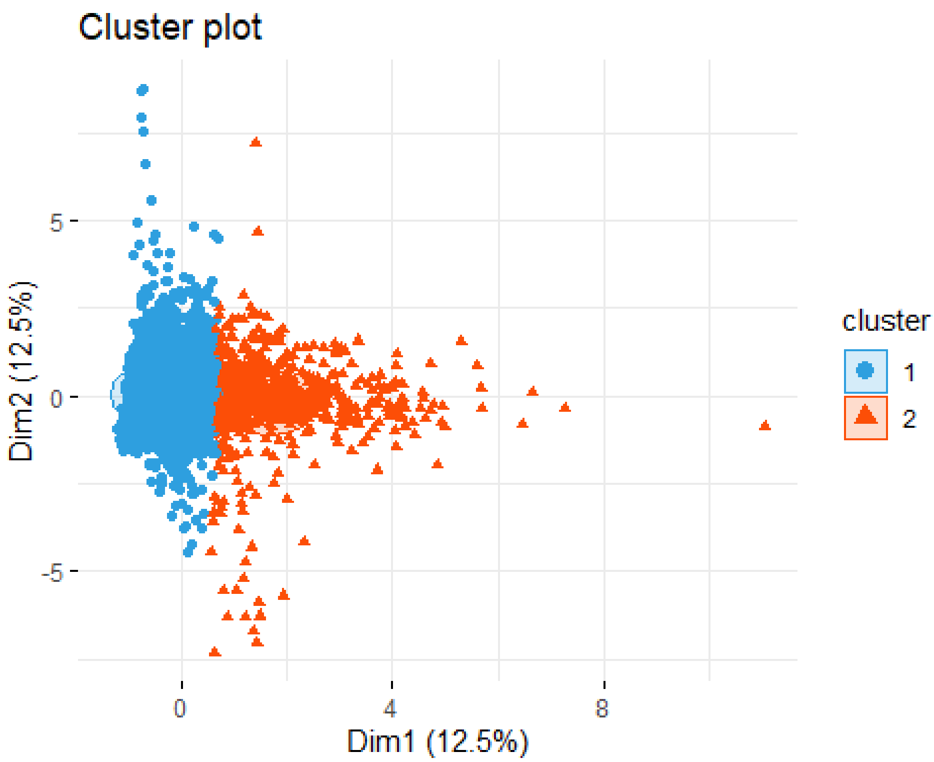
```
## [2850] 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 2 1 1
1 1 1
## [2887] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1
1 1 1
## [2924] 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [2961] 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 2 2 1 2 1 2
1 1 1
## [2998] 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1
1 1 1
## [3035] 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1
1 1 1
## [3072] 1 2 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 2 1 2 2 1 1 1 2 1
1 1 1
## [3109] 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2
1 1 2
## [3146] 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2
## [3183] 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3220] 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 1 1
1 1 1
## [3257] 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 2 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1
1 1 2
## [3294] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1
1 1 1
## [3331] 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 2 1 2 1 2 2 1
2 1 1
## [3368] 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3405] 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1
1 1 2
## [3442] 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1
1 1 1
## [3479] 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [3516] 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 1
1 1 1
## [3553] 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3590] 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 2 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1
2 2 1
## [3627] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2
1 1 1
## [3664] 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [3701] 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1
1 1 1
## [3738] 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 2 1
1 1 1
```

```
## [3775] 1 2 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 1 1 2 2 2 1 1 1 1 1 2 1 1 2 1
1 1 1
## [3812] 1 2 1 1 1 1 1 1 1 2 1 2 1 2 1 1 2 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2
1 1 2
## [3849] 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1
1 2 1
## [3886] 2 1 1 2 1 2 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1 2 1 2 1 1
1 2 1
## [3923] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 2
## [3960] 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 1
1 1 1
## [3997] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [4034] 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1
2 1 1
## [4071] 2 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1
1 1 1
## [4108] 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1
1 1 1
## [4145] 1 1 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1
2 2 1
## [4182] 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 1
## [4219] 2 1
##
## Within cluster sum of squares by cluster:
## [1] 24754.89 17292.82
##  (between_SS / total_SS =  21.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
table(km.res$cluster)
```

```
##
##    1    2
## 3444  776
```

```
fviz_cluster(km.res, data = pca_transformed_data,
             palette = c("#2E9FDF",  "#FC4E07"),
             ellipse.type = "euclid", # Concentration ellipse
             ggtheme = theme_minimal(),
             geom = "point" # Only show points, no text labels
)
```

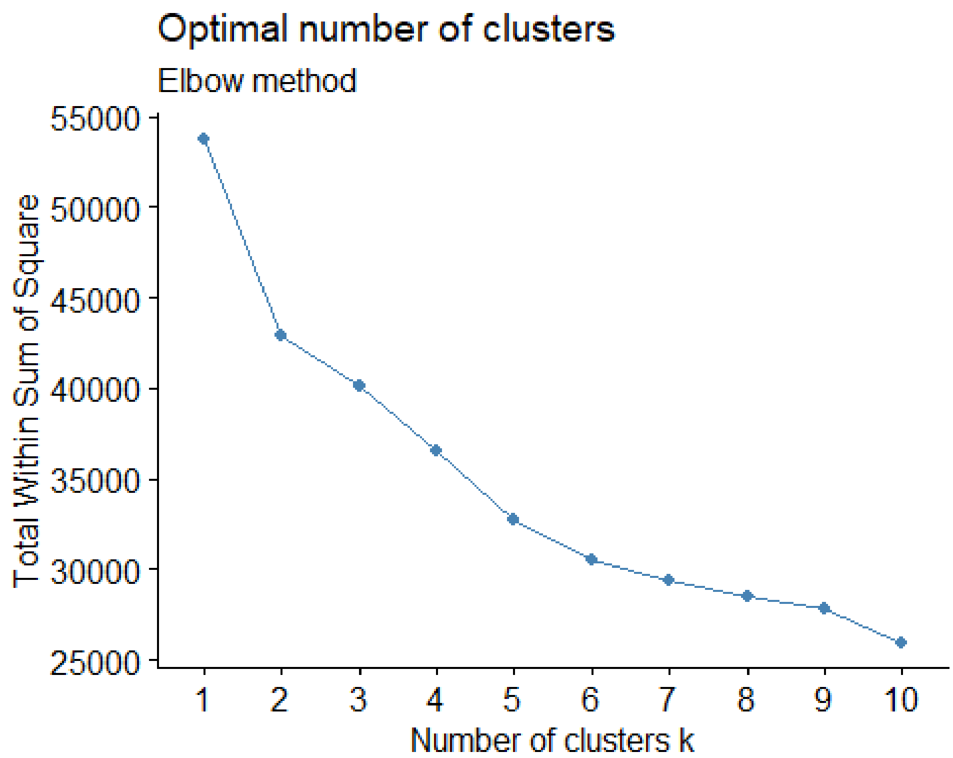



## 8. Pam, Partitioning Around Medoids

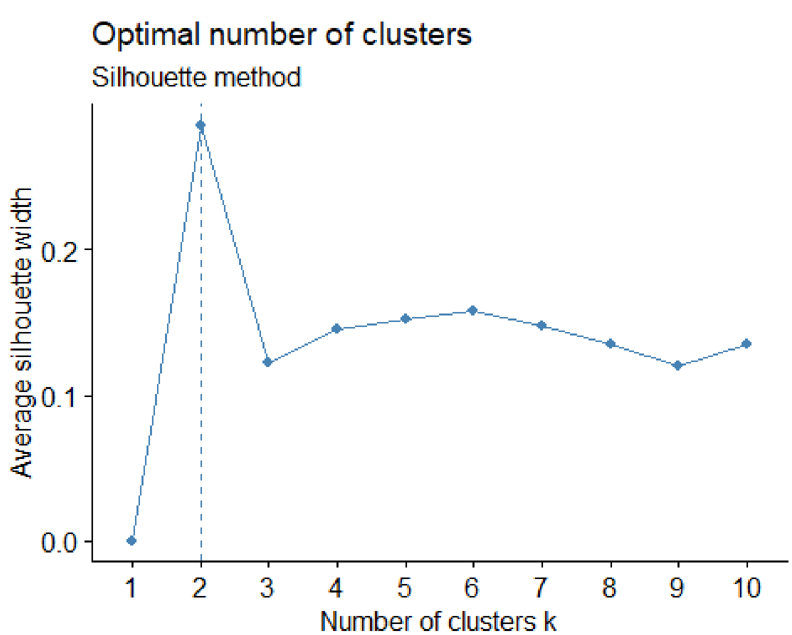
### 8.1 Optimal cluster numbers

Use Elbow method, Silhouette method and Gap to find the optimal K.

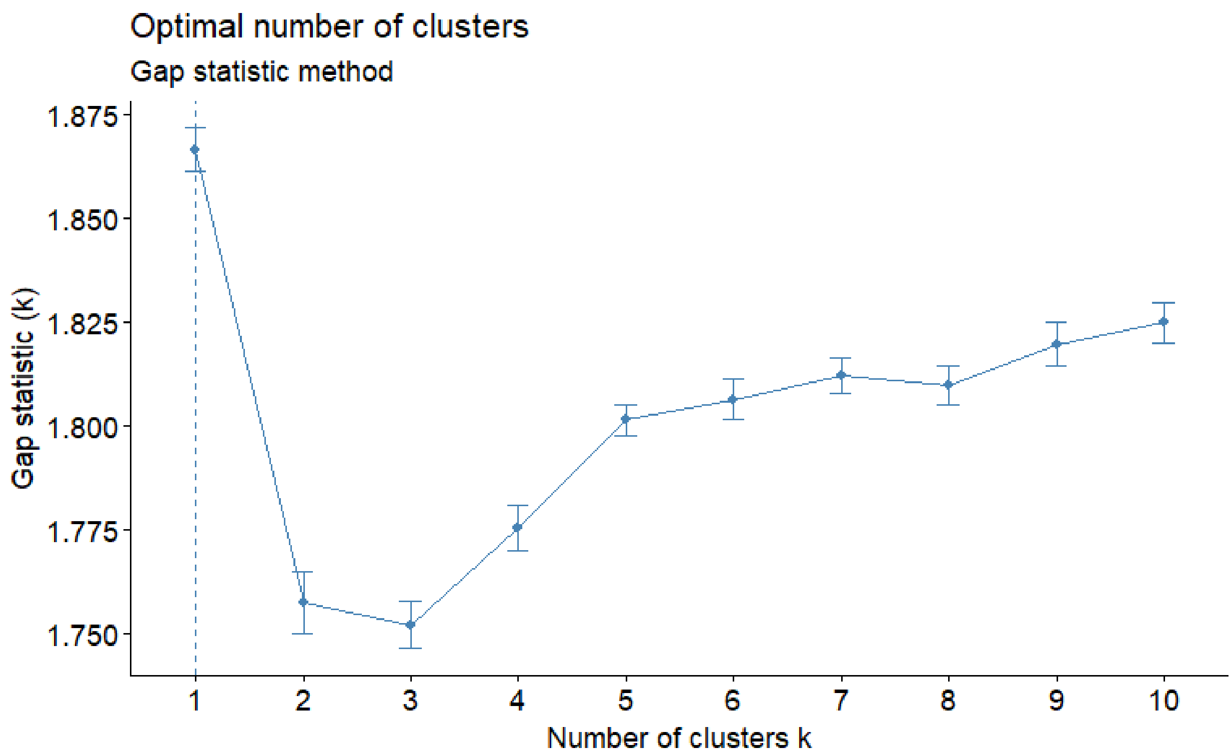
```
library(factoextra)
# Elbow method
plot_pam_elb = fviz_nbclust(pca_transformed_data, pam, method = "wss") +
labs(subtitle = "Elbow method")

print(plot_pam_elb)
```




```
# Silhouette method
plot_pam_sil = fviz_nbclust(pca_transformed_data, pam, method =
"silhouette")+
labs(subtitle = "Silhouette method")
print(plot_pam_sil)
```



```
# Gap statistic
set.seed(123)
fviz_nbclust(pca_transformed_data, pam, nstart = 25, method = "gap_stat",
nboot = 10)+
labs(subtitle = "Gap statistic method")
```




For PAM algorithm: Elbow method suggests k=2; Silhouette method suggests k=2; Gap statistics suggests k=1.

### 8.2 Pam cluster plot

```
pam.res <- pam(x = pca_transformed_data, k=2, diss = F)
print(pam.res)

## Medoids:
##         ID       PC1        PC2        PC3         PC4         PC5
PC6
## [1,] 1871 -1.279192  0.3739744  0.4187333 -0.08688125 -0.18084005
0.1364513
## [2,]  371  2.142303 -0.7776144 -0.1311005  0.42167966 -0.08453638 -
0.0457923
##               PC7       PC8
## [1,] -0.102819269 0.3356964
## [2,]  0.006281306 0.1665504
## Clustering vector:
##    [1] 1 2 1 1 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 2 2 2 1 1 2
1 1 1
##   [38] 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 2 2 1 2 2 2 1 2 2 1
```

```
1 2 1
##   [75] 1 1 1 2 1 2 1 1 1 2 2 2 1 1 2 2 1 1 1 2 2 1 2 1 2 2 2 2 2 1 1 2 1 1
1 2 1
##  [112] 1 2 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 2 2 1 1 2 2 1 1 1 1 1 2 1 1 1
2 1 1
##  [149] 1 1 1 2 1 1 1 2 1 2 1 1 2 2 1 1 1 1 2 2 1 1 1 1 2 2 1 1 1 1 1 1 2 1
2 2 1
##  [186] 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 2 1 2 1 1 1 2 2 2 1 1 1 2 1 2 1 1 1
1 1 1
##  [223] 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 2 1 2 1 1 2 1 2 1 2 1 2 1 2 1
1 1 1
##  [260] 1 1 2 2 1 1 1 1 2 1 2 2 2 1 2 1 1 1 1 1 2 2 2 1 1 1 2 2 1 1 2 1 1 1
1 1 1
##  [297] 1 1 1 2 2 2 1 1 1 2 2 1 1 1 2 1 1 1 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1
1 1 2
##  [334] 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 2 2 2 1 2 1 1 1 1 1 1
2 2 1
##  [371] 2 1 1 1 2 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 2 2 2 1
2 1 2
##  [408] 1 2 1 1 1 1 1 1 1 2 2 2 2 1 1 2 2 1 2 1 1 2 2 2 1 2 2 1 2 1 2 1 1 1
1 2 1
##  [445] 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 1 2 1 1 1 1 2 2 2 2 1 1
2 2 1
##  [482] 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1
1 1 2
##  [519] 1 1 2 1 1 2 1 2 1 1 2 1 1 1 2 1 2 2 2 1 1 2 1 1 1 2 2 1 1 1 1 1 2 1
2 2 1
##  [556] 2 1 2 1 1 1 1 1 2 1 2 1 1 2 1 2 1 1 2 1 1 1 2 1 1 2 1 2 1 1 2 1 2 2
1 2 1
##  [593] 1 1 1 1 1 2 2 2 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1
1 1 1
##  [630] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 2 1 1 2
1 1 1
##  [667] 1 1 2 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 2 2 1 1 1 2 1 2 1 2 1 1 2 1
2 1 2
##  [704] 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 1 1 2 1 2 1 1
1 1 1
##  [741] 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 2 2 1 2 1 1 1 2 1 1 2 1 2 1 1 1 2
1 1 2
##  [778] 1 1 2 2 2 1 1 1 1 2 1 2 1 2 1 1 1 2 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2
1 1 2
##  [815] 2 1 1 1 2 1 2 1 1 2 1 1 1 1 1 2 1 1 2 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1
1 1 2
##  [852] 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 1 1 2 2 1 2 1 1 2 1 1 1 2
1 1 1
##  [889] 1 1 1 2 2 1 1 1 1 1 2 1 2 2 2 2 2 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 2 1
1 1 2
##  [926] 2 2 1 1 1 2 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 2 2 1 1 2 1 2 1 1 1 1 2 1
1 1 2
##  [963] 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 2
```

```
1 1 1
## [1000] 1 1 2 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 1 1
1 1 1
## [1037] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 2 2 2 1 1 1 2 2 1 1 1 1 2 2 2
1 1 1
## [1074] 1 1 1 2 2 2 2 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 2 1 2 2 1 1 1 1 1 1 2 2
1 1 1
## [1111] 1 1 2 1 1 1 2 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2
1 1 1
## [1148] 1 2 2 1 2 1 2 1 1 2 1 2 2 2 2 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1
1 2 2
## [1185] 2 1 1 1 1 2 1 2 1 1 1 1 2 2 2 1 2 1 1 1 1 2 2 1 2 2 2 1 1 1 2 1 1 1
1 1 1
## [1222] 1 2 1 2 2 1 1 1 2 1 1 2 1 2 2 1 2 2 1 1 2 1 2 2 1 1 1 1 2 1 2 1 1 1
1 1 2
## [1259] 2 2 1 1 2 2 1 1 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 2 1
2 1 1
## [1296] 2 2 2 1 2 2 1 2 1 2 1 1 1 1 2 1 2 1 1 2 2 1 2 1 1 1 2 1 1 1 2 2 2 2
2 2 1
## [1333] 2 1 1 1 2 1 1 1 2 2 1 1 1 2 2 1 2 2 1 1 2 1 1 2 1 1 1 1 2 1 2 1 1 1
1 1 2
## [1370] 2 2 1 2 2 1 2 1 2 2 2 2 2 1 1 2 1 1 1 2 1 1 2 1 1 1 2 1 1 2 2 1 2 2
1 1 1
## [1407] 1 1 1 1 1 1 1 1 2 1 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 2
2 2 1
## [1444] 2 1 2 2 2 2 1 1 1 2 2 1 2 1 1 1 2 1 1 2 1 2 1 2 1 1 1 1 1 1 2 2 2 1
1 1 1
## [1481] 1 1 1 1 2 1 1 1 2 2 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 2 2 1 1 2 1
1 1 1
## [1518] 2 1 1 1 2 2 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 2 2 1 1 1 1
1 1 1
## [1555] 2 1 2 2 1 1 1 1 1 1 2 1 2 2 1 2 2 1 2 1 1 2 2 1 1 1 1 1 2 1 2 2 1 2
1 1 1
## [1592] 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 2 1 2 1 1 1 1 1 2 2 1 1 1 2 1 1
1 1 1
## [1629] 2 2 1 2 1 2 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1
1 2 2
## [1666] 1 2 1 1 2 1 2 1 2 1 1 1 2 1 1 1 1 1 1 2 1 2 2 2 1 2 2 2 1 2 1 1 2 1
1 1 2
## [1703] 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1
1 1 2
## [1740] 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 2 2 1 1 2 1 1 2 1 2 1 1 2 2 1
1 1 1
## [1777] 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 1 1 2 1 2 1
1 1 2
## [1814] 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1
1 1 1
## [1851] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 2
1 1 1
## [1888] 2 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 2 2 1 1 1 2 2 1 1 1 1 1 1
```

```
1 1 1
## [1925] 2 1 1 1 1 2 2 1 1 1 2 2 1 1 2 1 2 1 1 2 2 2 1 1 1 2 1 2 1 1 2 1 1 1
1 2 1
## [1962] 1 1 2 2 1 1 1 1 1 2 1 2 1 1 2 1 2 2 1 1 2 2 1 1 1 2 1 1 1 1 1 1 1 1
1 1 1
## [1999] 2 1 1 1 1 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 1 1 1 1
1 1 1
## [2036] 2 1 1 1 2 1 2 2 1 2 2 1 1 2 1 1 2 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1
2 2 1
## [2073] 1 1 1 1 1 1 2 1 1 1 1 2 2 1 2 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2
2 1 1
## [2110] 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 2 1 2 1 2 1 2
1 1 2
## [2147] 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 2 1 1 2 2 1 1 1 2 1 1 1 1 1 1
2 1 1
## [2184] 1 1 2 1 1 2 1 1 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 2
1 1 1
## [2221] 2 1 2 2 1 1 2 1 2 2 2 2 1 1 1 2 1 1 1 1 1 1 2 2 1 1 2 1 2 1 1 1 2 1
2 1 2
## [2258] 1 2 2 2 2 1 1 1 1 1 2 1 2 2 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2 1 1
1 1 1
## [2295] 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 1 1 1 2 1 1 1 1 2 1 2 1 1 2 2 2 1 1
2 1 2
## [2332] 2 2 1 1 1 2 1 2 1 2 1 1 1 2 1 1 1 1 1 2 1 1 2 1 2 1 1 2 2 1 1 1 1 2
2 2 1
## [2369] 2 1 1 1 1 2 1 2 1 1 1 1 2 1 2 1 2 2 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 2
1 1 1
## [2406] 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1
1 1 1
## [2443] 2 1 2 1 1 1 2 1 1 2 1 1 1 2 1 2 2 1 1 2 1 2 2 1 1 2 1 1 1 1 1 1 1 2
2 1 1
## [2480] 1 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 2 1 2 1 1 1 2 1 1 2 1 1 2 1 1 1 2 2
1 2 1
## [2517] 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1
1 2 1
## [2554] 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 2 2 1 2 2 1 1 1 1
1 1 2
## [2591] 2 2 2 1 1 2 1 1 1 2 1 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1
1 1 1
## [2628] 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 1 2
1 1 1
## [2665] 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 2 1 2 1 1 2 1 2 1
1 1 1
## [2702] 2 1 1 2 1 1 2 2 1 1 1 2 2 2 1 1 2 1 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1
1 1 2
## [2739] 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 2
1 1 1
## [2776] 1 1 2 2 2 1 2 1 1 1 2 1 1 1 2 1 1 1 1 2 2 1 1 2 1 1 1 2 1 2 1 1 2 1
2 2 1
## [2813] 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1
```

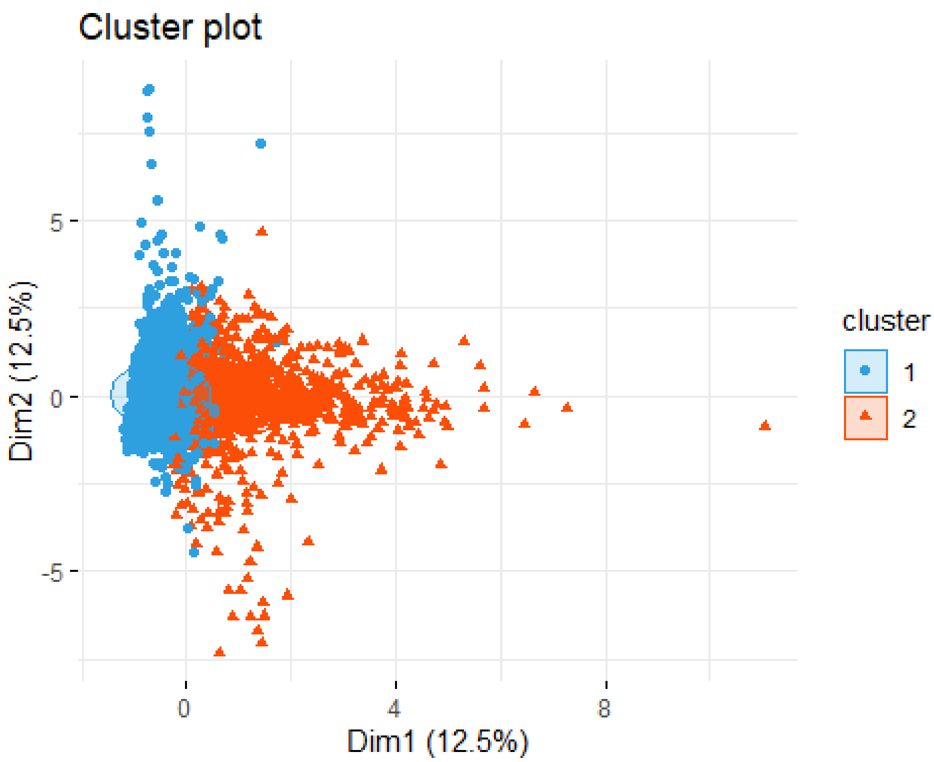
```
1 1 2
## [2850] 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 2 1 2 2 1 2 1 1 2 1 1
1 1 1
## [2887] 2 2 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1
1 1 1
## [2924] 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 2 1 1 1
1 1 1
## [2961] 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 2 2 1 1 2 2 1 2 1 2
2 1 1
## [2998] 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1
1 1 1
## [3035] 1 2 1 1 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 1
1 1 1
## [3072] 1 2 1 2 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1 2 2 1 2 2 1 1 1 2 1
2 1 1
## [3109] 2 1 1 1 1 1 2 2 1 1 2 1 1 2 2 2 2 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2
1 1 2
## [3146] 2 1 2 2 1 1 1 1 1 1 2 1 1 2 1 1 1 2 2 1 1 2 1 1 1 2 1 1 1 1 1 1 2 1
1 1 2
## [3183] 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2
1 2 1
## [3220] 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 2 1 1 2 2 1 1 1 2 2 1 1
1 1 1
## [3257] 2 2 1 2 1 1 1 2 1 1 2 1 2 1 1 2 2 2 1 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1
1 1 2
## [3294] 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 2 2 2 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1
1 1 1
## [3331] 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 2 1 2 1 2 1 2 1 2 2 1
2 2 1
## [3368] 1 1 1 1 1 2 1 2 2 2 1 2 2 1 1 2 2 2 1 1 1 1 2 2 1 1 2 2 1 1 2 1 1 1
1 1 2
## [3405] 1 1 2 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 2 2 2 1 2 1 1 2 1 1 2 1 2 1 1
2 1 2
## [3442] 1 1 2 1 1 2 2 1 1 1 1 2 2 1 1 2 1 2 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1 1
1 1 1
## [3479] 1 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 2
1 1 1
## [3516] 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 1
2 1 2
## [3553] 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1
1 2 1
## [3590] 1 2 2 1 1 1 1 1 1 1 2 1 2 1 2 2 1 2 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1
2 2 1
## [3627] 1 1 1 1 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2
1 1 1
## [3664] 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1
2 1 2
## [3701] 1 1 1 1 1 2 1 1 2 1 2 1 1 1 2 1 1 1 2 1 2 1 1 2 1 2 2 1 1 2 1 1 1 2
1 1 2
## [3738] 2 2 2 1 1 1 1 1 1 2 1 1 2 1 2 1 1 2 2 1 1 1 1 1 1 2 1 1 2 1 1 2 2 2
```

```
1 2 1
## [3775] 2 2 1 2 1 1 1 1 2 1 2 2 2 1 2 1 1 2 1 2 1 2 2 2 1 1 2 1 1 2 1 1 2 1
2 1 1
## [3812] 2 2 1 1 1 1 1 1 1 2 1 2 1 2 1 1 2 2 2 2 1 1 1 2 1 1 2 2 1 2 1 1 1 2
1 2 2
## [3849] 2 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1
1 2 1
## [3886] 2 1 1 2 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 1 2 1 2
1 2 1
## [3923] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
1 1 2
## [3960] 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2 1 1 2 1
1 1 1
## [3997] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1
1 1 1
## [4034] 1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 2 2 1
2 2 1
## [4071] 2 1 1 1 1 1 2 2 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 2 1 1 2 1 2 1 1 1
2 1 1
## [4108] 1 1 2 1 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 2 2 1 1 2 2 2 1
2 1 2
## [4145] 1 1 2 2 1 1 1 2 2 1 2 1 1 1 2 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 2 1 2
2 2 2
## [4182] 1 1 1 1 1 1 1 2 2 2 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 2 1 2 2 1 1
1 1 1
## [4219] 2 1
## Objective function:
##    build     swap
## 2.684214 2.681462
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```

```r
table(pam.res$cluster)
```

```
##
##    1    2
## 2961 1259
```

```r
pam_cluster_fviz = fviz_cluster(pam.res, data = pca_transformed_data,
             palette = c("#2E9FDF",  "#FC4E07"),
             ellipse.type = "euclid", # Concentration ellipse
             ggtheme = theme_minimal(),
             geom = "point" # Only show points, no text labels
)

print(pam_cluster_fviz)
```




## 9. Cluster Validation

### 9.1 Cluster internal validation

```
library(clValid)
clmethods = c('kmeans','pam')
intern = clValid(pca_transformed_data,nClust = 2:10,clMethods =
clmethods,validation = "internal",maxitems = 5000)
summary(intern)
plot(intern)
```

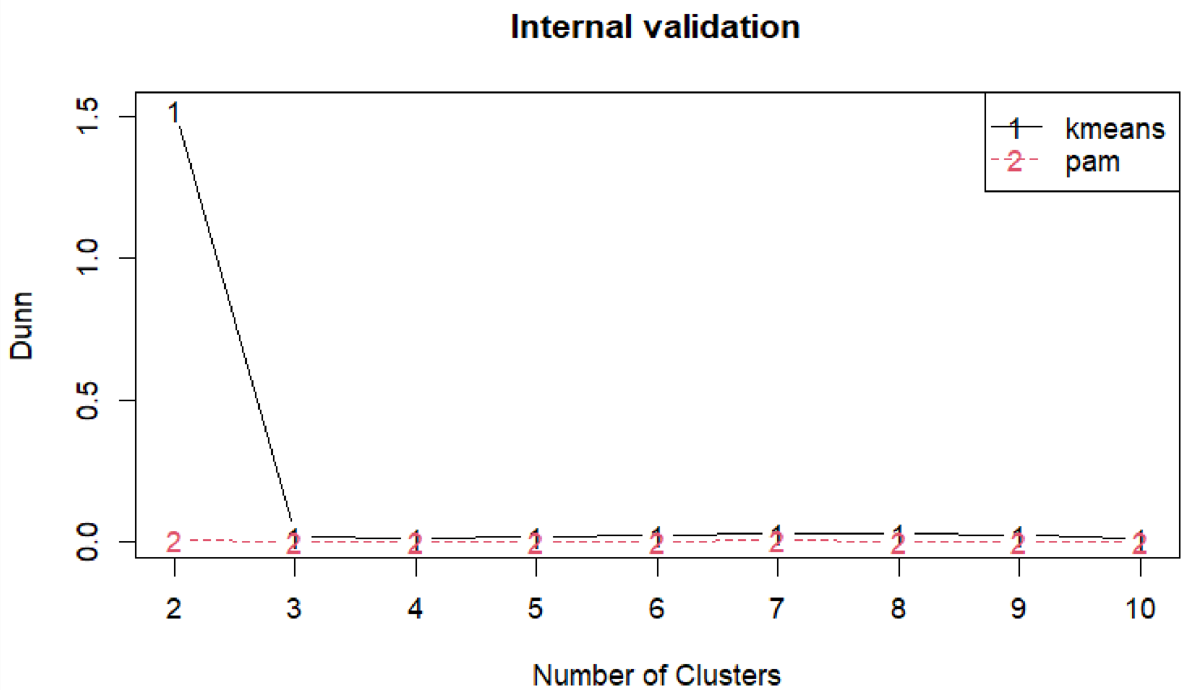
```
Warning: rownames for data not specified, using 1:nrow(data)
Clustering Methods:
 kmeans pam

Cluster sizes:
 2 3 4 5 6 7 8 9 10

Validation Measures:
                                 2         3         4         5         6
7         8         9         10

kmeans Connectivity      2.9290   83.1552  531.4802  438.5115  530.2405
480.6274  416.6631  538.4210  912.2714
       Dunn              1.5234    0.0203    0.0127    0.0192    0.0215
0.0289    0.0309    0.0249    0.0135
       Silhouette        0.9284    0.3083    0.3373    0.2287    0.3499
0.3262    0.2490    0.2390    0.2384
pam    Connectivity    559.9452  778.3401  631.1833  655.3607  767.2897
1032.6143 1318.0020 1429.9710 1596.1329
       Dunn              0.0049    0.0026    0.0036    0.0027    0.0019
0.0046    0.0019    0.0027    0.0023
```

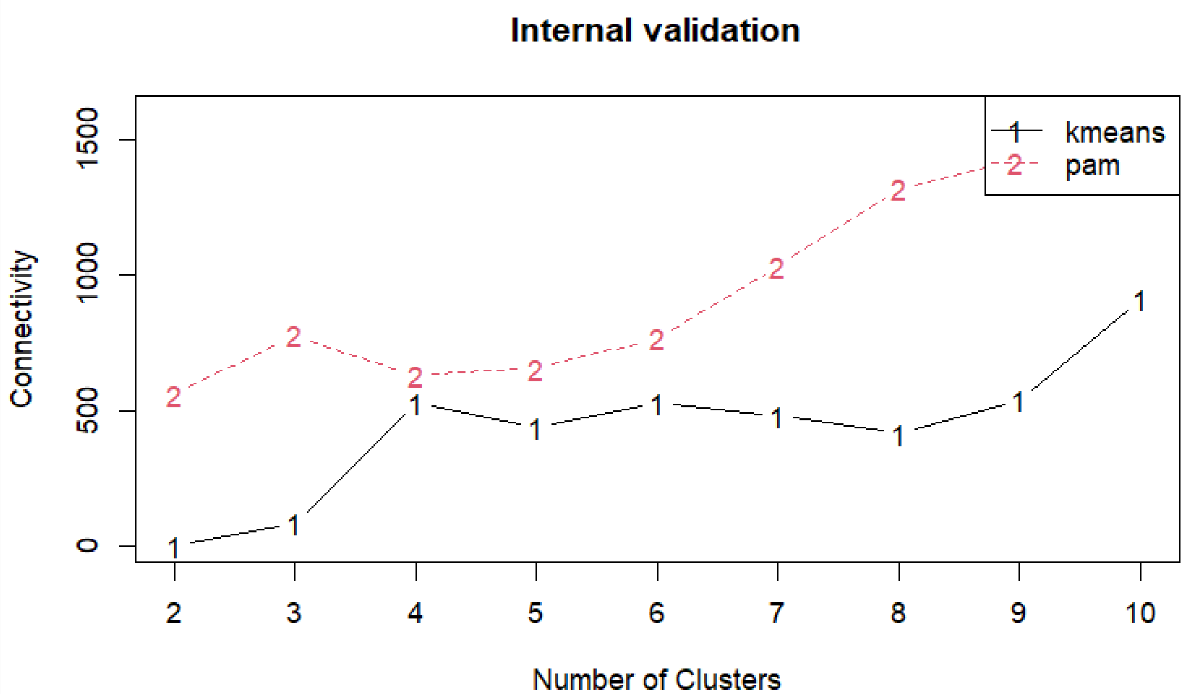
```
      Silhouette        0.2853    0.1220    0.1451    0.1517    0.1576
0.1474    0.1354    0.1204    0.1356

Optimal Scores:
```

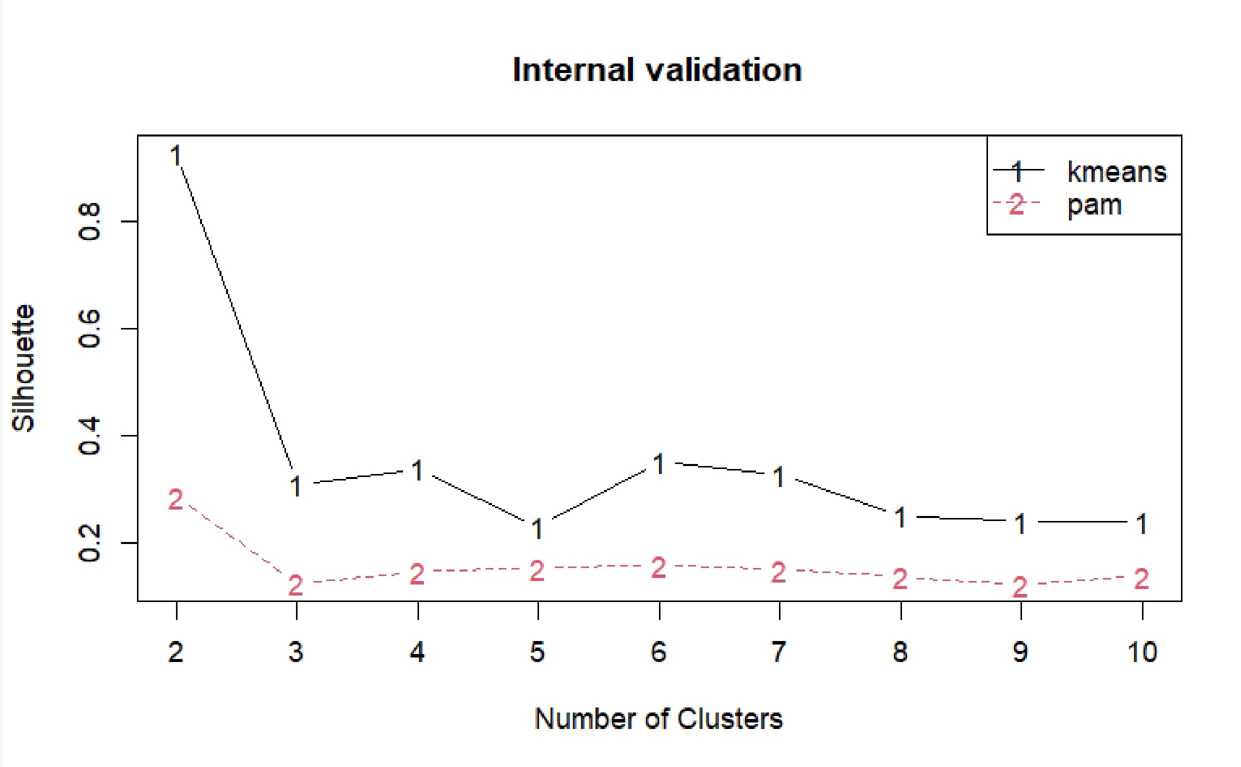
Description: df [3 × 3]

| | Score | Method | Clusters |
|---|---|---|---|
| | <dbl> | <chr> | <chr> |
| *Connectivity* | *2.9290* | *kmeans* | *2* |
| *Dunn* | *1.5234* | *kmeans* | *2* |
| *Silhouette* | *0.9284* | *kmeans* | *2* |

*3 rows*


Internal validation
Dunn
Number of Clusters
kmeans
pam
0.0
0.5
1.0
1.5
2
3
4
5
6
7
8
9
10


Internal validation
Connectivity
Number of Clusters
kmeans
pam
0
500
1000
1500
2
3
4
5
6
7
8
9
10




All of Connectivity,Dunn,Silhouette results suggesting that kmeans with k=2 is the optimal clustering method and cluster number.

### 9.2 Cluster stability validation

```
library(clValid)
clmethods = c('kmeans','pam')
stab = clValid(pca_transformed_data,nClust = 2:10,clMethods =
clmethods,validation = "stability", maxitems = 5000)
summary(stab)
plot(stab)
```

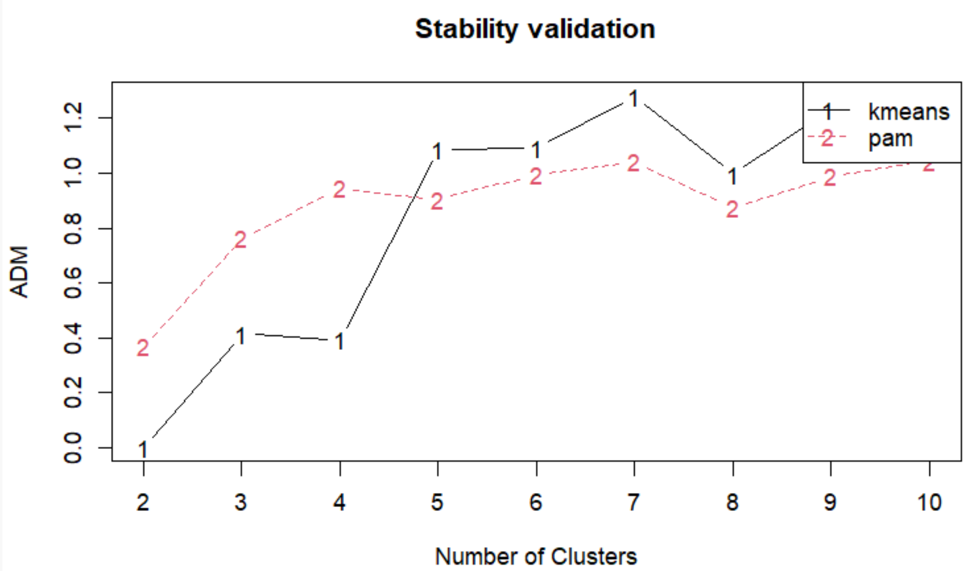
```
Warning: did not converge in 10 iterationsWarning: rownames for data not
specified, using 1:nrow(data)
Clustering Methods:
 kmeans pam

Cluster sizes:
 2 3 4 5 6 7 8 9 10

Validation Measures:
                 2      3      4      5      6      7      8      9     10

kmeans APN    0.0000 0.0823 0.0698 0.1328 0.3160 0.3915 0.3327 0.4315 0.4158
       AD     4.3143 4.2590 3.8205 3.8194 3.7800 3.6627 3.3984 3.3954 3.2489
       ADM    0.0000 0.4134 0.3920 1.0848 1.0919 1.2798 0.9962 1.2331 1.1250
       FOM    1.1671 1.1589 1.1568 1.1517 1.1451 1.1138 1.0870 1.0711 1.0464
pam    APN    0.0841 0.2048 0.2592 0.2528 0.3208 0.3596 0.3130 0.3384 0.3657
       AD     3.9101 3.7969 3.6390 3.4601 3.3603 3.3042 3.1813 3.1537 3.0981
       ADM    0.3673 0.7632 0.9450 0.9036 0.9937 1.0407 0.8726 0.9881 1.0443
```

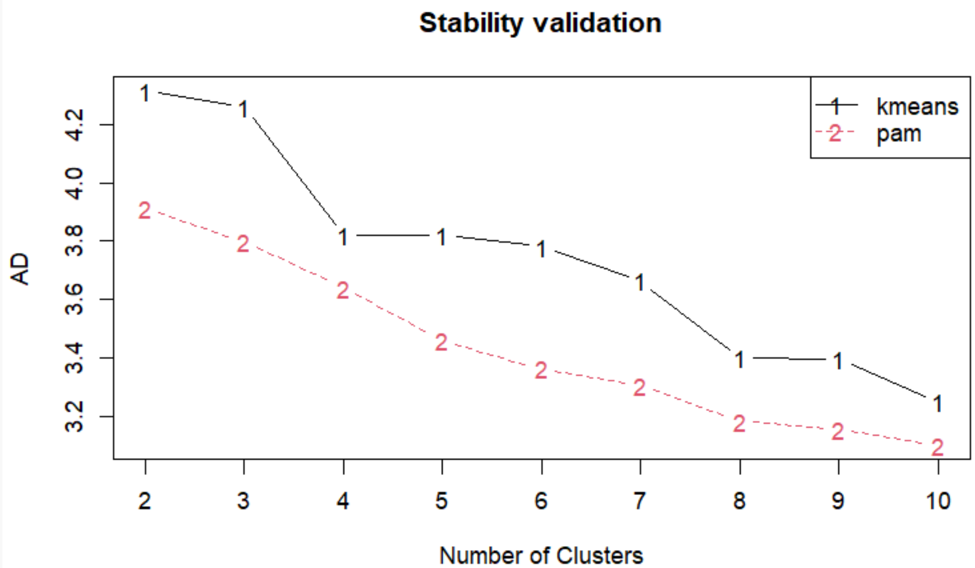
```
       FOM   1.2003 1.1968 1.1789 1.1752 1.1519 1.1414 1.1249 1.1198 1.1131

Optimal Scores:
```

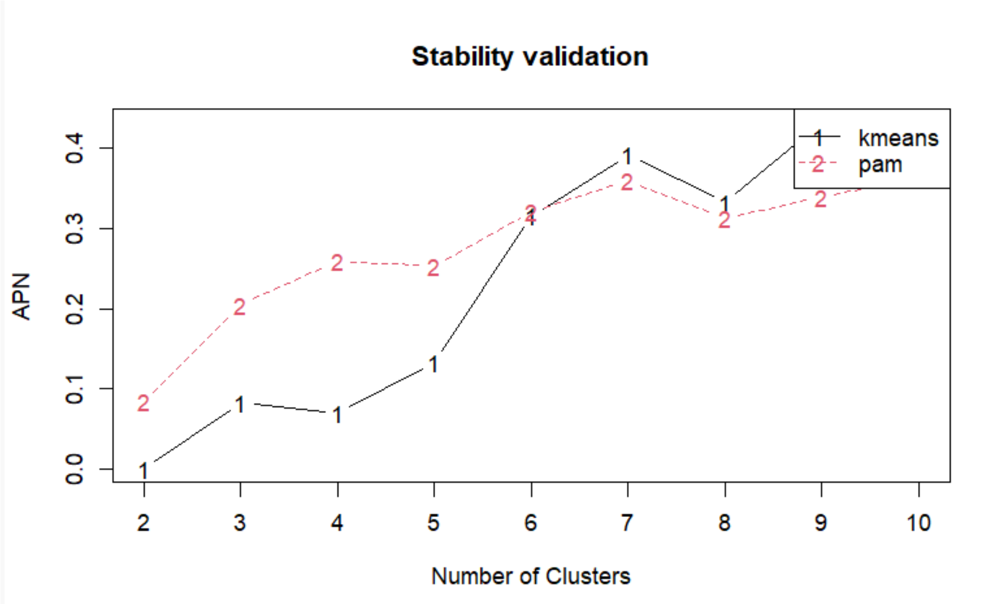
Description: *df [4 × 3]*

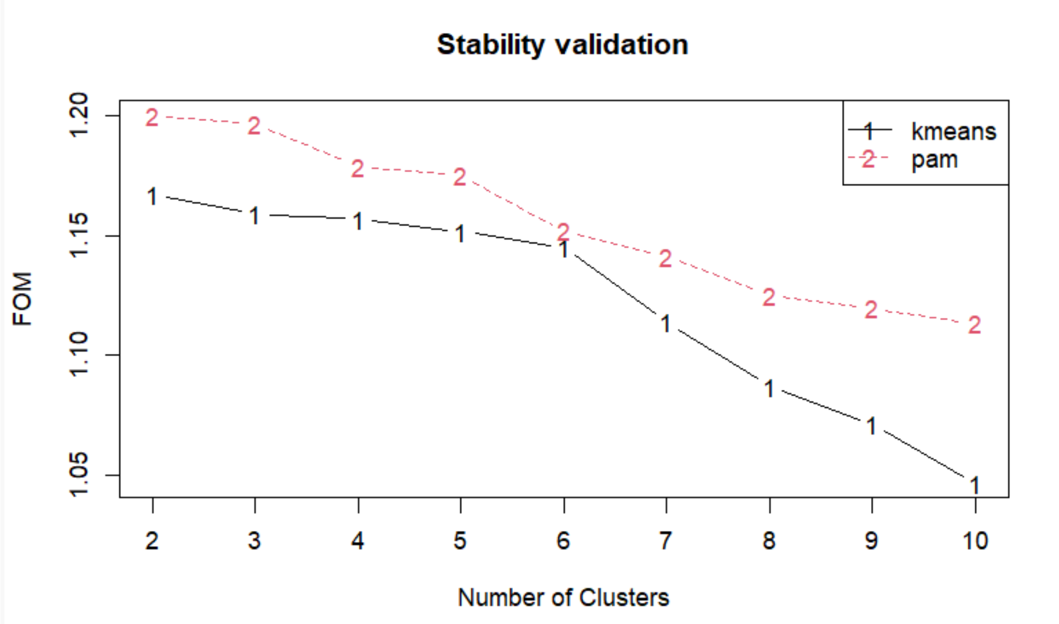
| | Score | Method | Clusters |
|---|---|---|---|
| | <dbl> | <chr> | <chr> |
| *APN* | *0.0000* | *kmeans* | *2* |
| *AD* | *3.0981* | *pam* | *10* |
| *ADM* | *0.0000* | *kmeans* | *2* |
| *FOM* | *1.0464* | *kmeans* | *10* |

*4 rows*






APN scores suggests kmeans with k=2 is optimal;

AD scores suggests pam with k=10 is optimal;

ADM scores suggests kmeans with k=2 is optimal;

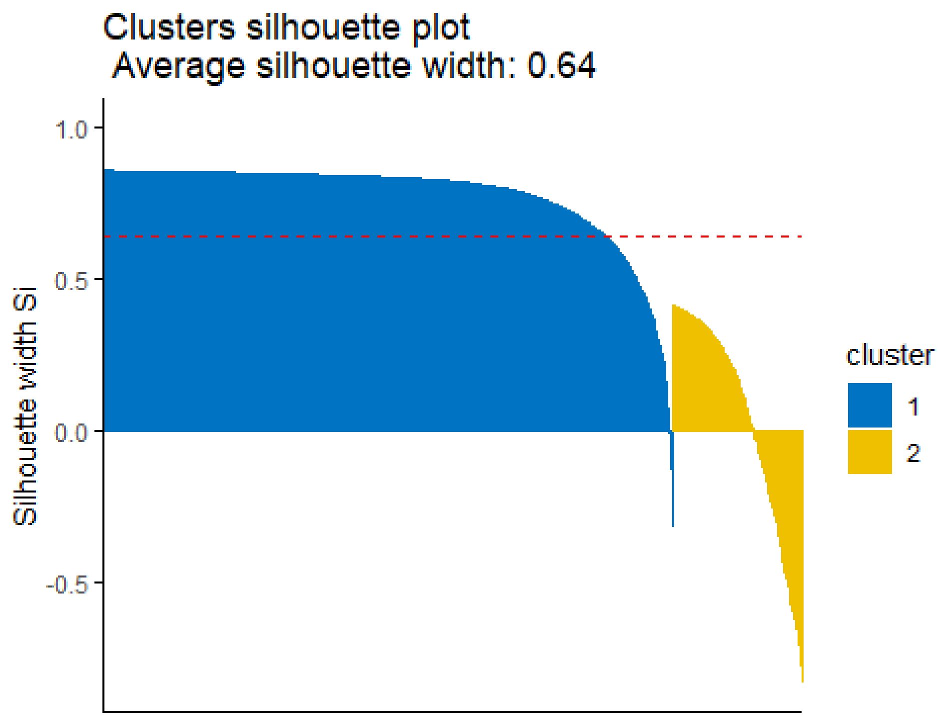
FOM scores suggests kmeans with k=10 is optimal;

# 10 Cluster Distribution Visualization

## 10.1 Silhouette score

Prepare a dataset with kmeans k=2 clustering result and original customer data

```r
library(dplyr)
# add dbscan result to the original customer data
customer_data_with_cluster <- customer_data %>%
  mutate(Cluster = dbscan_result$cluster)

# Filter out the outliers (Cluster 0)
customer_data_no_outliers <- customer_data_with_cluster %>%
  filter(Cluster != 0) %>%
  select(-Cluster)  #  remove the Cluster column

customer_data_no_outliers_with_kmCLust  <- customer_data_no_outliers %>%
  mutate(Cluster = km.res$cluster) %>%
  select(-CustomerID)

library(cluster)
# Split data by clusters
cluster_data <- customer_data_no_outliers_with_kmCLust %>%
  group_by(Cluster) %>%
  summarise(across(everything(), mean, na.rm = TRUE))

## Warning: There was 1 warning in `summarise()`.
## ℹ In argument: `across(everything(), mean, na.rm = TRUE)`.
## ℹ In group 1: `Cluster = 1`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))

# Convert kmeans results to a silhouette object
sil.km <- silhouette(km.res$cluster,
dist(customer_data_no_outliers_with_kmCLust))

# Visualize the silhouette plot
fviz_silhouette(sil.km, palette = "jco", ggtheme = theme_classic())

##   cluster size ave.sil.width
## 1       1 3444          0.77
## 2       2  776          0.04
```




Cluster 1: Size: 776 Average Silhouette Width: 0.04 Interpretation: This indicates that cluster 1 has a low silhouette width, suggesting that the points in this cluster are not well separated from other clusters and may be misclassified.

Cluster 2: Size: 3444 Average Silhouette Width: 0.77 Interpretation: This cluster has a higher average silhouette width, indicating that the points in this cluster are better clustered and more distinct from points in other clusters.

Conclusion: Cluster Quality: Cluster 2 is relatively well-defined with an average silhouette width of 0.64, indicating good clustering quality. However, Cluster 1 has a very low silhouette width of 0.04, indicating poor clustering quality and potential issues with the clustering algorithm’s ability to distinguish this cluster from others.

Overall Clustering: The overall average silhouette width of 0.64 suggests that the clustering solution is moderate.

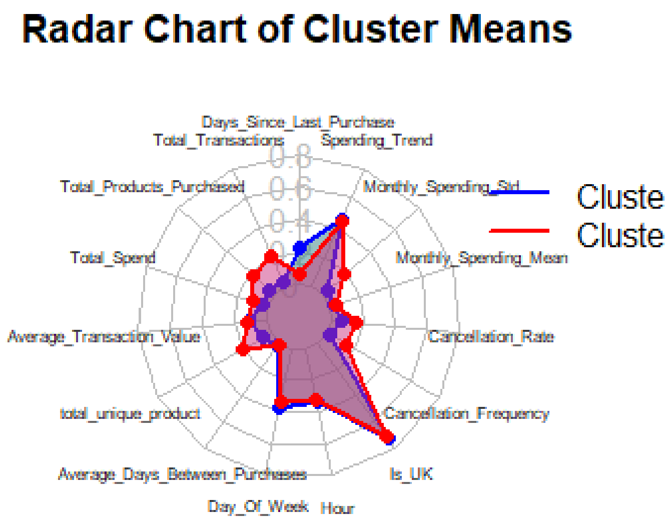
### 10.2 Cluster analysis and profiling

Draw a radar chart for each cluster to display the mean of each feature of these 2 clusters.

```
#install.packages("fmsb")
library(fmsb)
```

```
selected_columns = setdiff(names(customer_data), "CustomerID")

# Create data for radar plot
radar_data <- cluster_data %>%
  select(all_of(selected_columns)) %>%
  as.data.frame()


max_data <- customer_data_no_outliers_with_kmCLust %>%
  summarise(across(all_of(selected_columns), max, na.rm = TRUE))

min_data <- customer_data_no_outliers_with_kmCLust %>%
  summarise(across(all_of(selected_columns), min, na.rm = TRUE))

c1_scaled_radar_data = (radar_data[1,] - min_data[1,]) / (max_data[1,] -
min_data[1,] )
c2_scaled_radar_data = (radar_data[2,] - min_data[1,]) / (max_data[1,] -
min_data[1,] )
scaled_radar_data = rbind(c1_scaled_radar_data,c2_scaled_radar_data)
# Add max and min values for the radar chart
scaled_radar_data <- rbind(
  max = rep(1, ncol(scaled_radar_data)),  # Max values (scaled to 1)
  min = rep(0, ncol(scaled_radar_data)),  # Min values (scaled to 0)
  scaled_radar_data
)

# Set row names for clarity
row.names(scaled_radar_data) <- c("Max", "Min", "Cluster 1", "Cluster 2")

# Print the scaled radar data for verification
print(scaled_radar_data)

##           Days_Since_Last_Purchase Total_Transactions
Total_Products_Purchased
## Max                     1.00000000         1.00000000
1.00000000
## Min                     0.00000000         0.00000000
0.00000000
## Cluster 1               0.28749459         0.04179007
0.05902781
## Cluster 2               0.07642478         0.25931057
0.22341210
##           Total_Spend Average_Transaction_Value total_unique_product
## Max        1.00000000                 1.0000000           1.00000000
## Min        0.00000000                 0.0000000           0.00000000
## Cluster 1  0.04506843                 0.1316436           0.06531751
## Cluster 2  0.12627742                 0.1489589           0.24128829
##           Average_Days_Between_Purchases Day_Of_Week      Hour     Is_UK
```

```
## Max                                1.00000000   1.0000000 1.0000000 1.0000000
## Min                                0.00000000   0.0000000 0.0000000 0.0000000
## Cluster 1                          0.02512528   0.4660279 0.4247967 0.9146341
## Cluster 2                          0.01685408   0.4181701 0.4097938 0.8878866
##           Cancellation_Frequency Cancellation_Rate Monthly_Spending_Mean
## Max                   1.00000000         1.0000000            1.00000000
## Min                   0.00000000         0.0000000            0.00000000
## Cluster 1             0.02465641         0.0816678            0.03679383
## Cluster 2             0.17740550         0.1902648            0.04751639
##           Monthly_Spending_Std Spending_Trend
## Max                 1.00000000      1.0000000
## Min                 0.00000000      0.0000000
## Cluster 1           0.04654589      0.5647319
## Cluster 2           0.22782753      0.5547339

# Plot the radar chart with adjusted label sizes
radarchart(scaled_radar_data,
           axistype = 1,
           pcol = c("blue", "red"),
           pfcol = c(rgb(0.2, 0.5, 0.5, 0.5), rgb(0.8, 0.2, 0.5, 0.5)),
           plwd = 2,
           plty = 1,
           title = "Radar Chart of Cluster Means",
           cglcol = "grey", cglty = 1, axislabcol = "grey", caxislabels =
seq(0, 1, 0.2),
           cglwd = 0.5, vlcex = 0.5,  # Adjust the vlcex parameter for
variable label size
            cex.lab = 0.3  # Adjust the cex.lab parameter for axis label size
)

# Add a legend
legend(x = 1, y = 1, legend = c("Cluster 1", "Cluster 2"), col = c("blue",
"red"), lty = 1, lwd = 2, bty = "n")
```




### *Summary of profile of clusters*

| Key Attribute | Cluster 1: “High Spend Frequent Buyers” | Cluster 2: “Low Spend Infrequent Buyers” |
|---|---|---|
| Size | 776 | 3444 |
| Recent Purchases | 28.5 days since last purchase | 107.2 days since last purchase |
| Transaction Count | 11.4 transactions | 2.7 transactions |
| Total Spend | $16,398 | $2,676 |
| Variety in Purchases | 136 unique products | 38 unique products |
| UK Customers | 88.8% | 91.5% |
| Cancellation Rate | 19% | 8% |

Cluster 1: “High Spend Frequent Buyers”: High spending and frequent purchases with a variety of products. Cluster 2: “Low Spend Infrequent Buyers”: Low spending and infrequent purchases with limited product variety.