Exploring Variations in Clustering and Predictive Analysis

Ekaterina Galkina

Big Data & Analytics

The George Washington University

Introduction

The objective of this project is to determine which features are the most useful in predicting obesity.

In order to achieve this goal, we are going to study the Obesity Data set which includes data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labeled with the class variable Nobesity (Obesity Level), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III.

First we are going to do pairwise plotting to find any possible linear relationships between pairs of attributes consisting of the first 16 columns (gender, age, height, …) and the last column (Obesity level). Then, we are going to select the features with linear relationships, then extract the simplified data set containing only the most relevant attributes and split the final table into training sets and test sets. Finally we are going to predict obesity level using GLM and Predict for subsets of the data using these features and compare which variations of variables work best to do so.

Question 1 : Plotting Data Set & Identifying possible relationships

First, let's import our data set using read.table() pre-implemented-function, as it is shown on the screenshot below (figure 1.1).

```
> path <- file.choose()
>
> data <- read.table(path, sep = ",", header = TRUE, stringsAsFactors = FALSE)
> data[1:10, 1:17]
   Gender Age Height Weight family_history_with_overweight FAVC FCVC NCP      CAEC SMOKE CH2O SCC FAF TUE
1  Female  21   1.62   64.0                           yes   no    2   3 Sometimes    no    2  no   0   1
2  Female  21   1.52   56.0                           yes   no    3   3 Sometimes   yes    3 yes   3   0
3    Male  23   1.80   77.0                           yes   no    2   3 Sometimes    no    2  no   2   1
4    Male  27   1.80   87.0                            no   no    3   3 Sometimes    no    2  no   2   0
5    Male  22   1.78   89.8                            no   no    2   1 Sometimes    no    2  no   0   0
6    Male  29   1.62   53.0                            no  yes    2   3 Sometimes    no    2  no   0   0
7  Female  23   1.50   55.0                           yes  yes    3   3 Sometimes    no    2  no   1   0
8    Male  22   1.64   53.0                            no   no    2   3 Sometimes    no    2  no   3   0
9    Male  24   1.78   64.0                           yes  yes    3   3 Sometimes    no    2  no   1   1
10   Male  22   1.72   68.0                           yes  yes    2   3 Sometimes    no    2  no   1   1
         CALC                MTRANS          NObeyesdad
1          no Public_Transportation       Normal_Weight
2   Sometimes Public_Transportation       Normal_Weight
3  Frequently Public_Transportation       Normal_Weight
4  Frequently               Walking  Overweight_Level_I
5   Sometimes Public_Transportation Overweight_Level_II
6   Sometimes            Automobile       Normal_Weight
7   Sometimes             Motorbike       Normal_Weight
8   Sometimes Public_Transportation       Normal_Weight
9  Frequently Public_Transportation       Normal_Weight
10         no Public_Transportation       Normal_Weight
>
```

Figure 1.1 : Data set's import

Translating alphanumeric values into numeric values

To make it easier to work with the dataset, we need to convert alphanumeric values into integers. Table 2.1 below shows the integer encoding for each attribute that originally has string values in a fixed set.

The encoding follows these conventions:

- Yes/No and Female/Male values are converted to 1 or 0.
- Frequency-based attributes are assigned values from 0 to 3, with 0 representing the least frequent (No/Never) and 3 representing the most frequent (Always).
- Transportation methods are ranked from 0 to 4, where 0 corresponds to the least healthy option (Car) and 4 to the healthiest (Walking).
- Obesity levels are numbered from 0 to 6, in increasing order, starting from Insufficient Weight (0) to Obesity Type III (6).

Variable	Encoding
Gender	Female → 1 Male → 0
family_history_with_overweight	yes → 1 no → 0
FAVC (Frequent high-caloric food consumption)	yes → 1 no → 0
CAEC (Consumption of food between meals)	No → 0 Sometimes → 1 Frequently → 2 Always → 3
SMOKE	yes → 1 no → 0
SCC (Calories monitoring)	yes → 1 no → 0
CALC (Alcohol consumption)	No → 0 Sometimes → 1 Frequently → 2 Always → 3

MTRANS (Usual transportation method)	Automobile → 0 Motorbike → 1 Public_Transportation → 2 Bike → 3 Walking → 4
NObeyesdad	Insufficient_Weight → 0 Normal_Weight → 1 Overweight_Level_I → 2 Overweight_Level_II → 3 Obesity_Type_I → 4 Obesity_Type_II → 5 Obesity_Type_III → 6

Table 1.2 : Mapping alphanumeric values

The code in Figure 1.3 modifies the dataset based on the convention outlined in Table 1.2.

```
> # Convert categorical variables to numeric
> data <- data %>%
+   mutate(
+     Gender = ifelse(Gender == "Female", 1, 0),
+
+     family_history_with_overweight = ifelse(family_history_with_overweight == "yes", 1, 0),
+
+     FAVC = ifelse(FAVC == "yes", 1, 0),
+
+     CAEC = case_when(
+       CAEC == "no" ~ 0,
+       CAEC == "Sometimes" ~ 1,
+       CAEC == "Frequently" ~ 2,
+       CAEC == "Always" ~ 3
+     ),
+
+     SMOKE = ifelse(SMOKE == "yes", 1, 0),
+
+     SCC = ifelse(SCC == "yes", 1, 0),
+
+     CALC = case_when(
+       CALC == "no" ~ 0,
+       CALC == "Sometimes" ~ 1,
+       CALC == "Frequently" ~ 2,
+       CALC == "Always" ~ 3
+     ),
+
+     MTRANS = case_when(
+       MTRANS == "Automobile" ~ 0,
+       MTRANS == "Motorbike" ~ 1,
+       MTRANS == "Public_Transportation" ~ 2,
+       MTRANS == "Bike" ~ 3,
+       MTRANS == "Walking" ~ 4
+     ),
+
+     NObeyesdad = case_when(
+       NObeyesdad == "Obesity_Type_III" ~ 6,
+       NObeyesdad == "Obesity_Type_II" ~ 5,
+       NObeyesdad == "Obesity_Type_I" ~ 4,
+       NObeyesdad == "Overweight_Level_II" ~ 3,
+       NObeyesdad == "Overweight_Level_I" ~ 2,
+       NObeyesdad == "Normal_Weight" ~ 1,
+       NObeyesdad == "Insufficient_Weight" ~ 0
+     )
+   )
>
```

Figure 1.3 : Code for mapping each character field into numeric

The screenshot below (figure 1.4) illustrates the dataset after this change. This output was obtained by running the following command : data[1:10,], which prints the first 10 lines on the dataset.

```
   Gender Age Height Weight family_history_with_overweight FAVC FCVC NCP CAEC SMOKE CH2O SCC FAF TUE CALC
1       1  21   1.62   64.0                               1    0    2   3    1     0    2   0   0   1    0
2       1  21   1.52   56.0                               1    0    3   3    1     1    3   1   3   0    1
3       0  23   1.80   77.0                               1    0    2   3    1     0    2   0   2   1    2
4       0  27   1.80   87.0                               0    0    3   3    1     0    2   0   2   0    2
5       0  22   1.78   89.8                               0    0    2   1    1     0    2   0   0   0    1
6       0  29   1.62   53.0                               0    1    2   3    1     0    2   0   0   0    1
7       1  23   1.50   55.0                               1    1    3   3    1     0    2   0   1   0    1
8       0  22   1.64   53.0                               0    0    2   3    1     0    2   0   3   0    1
9       0  24   1.78   64.0                               1    1    3   3    1     0    2   0   1   1    2
10      0  22   1.72   68.0                               1    1    2   3    1     0    2   0   1   1    0
   MTRANS NObeyesdad
1       2          1
2       2          1
3       2          1
4       4          2
5       2          3
6       0          1
7       1          1
8       2          1
9       2          1
10      2          1
>
```

Figure 1.3 : Code for mapping each character field into numeric

Furthermore, let's plot our data set to repair possible correlations.

Pairwise plotting and identification of relationships between attributes

First and easiest method is to simply apply the plot() function, as it is shown on figure 1.4. The result of this command is given below (figure 1.5).

```
> # First we print the whole matrix of pairwise plotting, but it is not very
> # usefull as everything is too small to perceive possible linearity
> plot(data)
```

Figure 1.4 : plot() on data set : source code

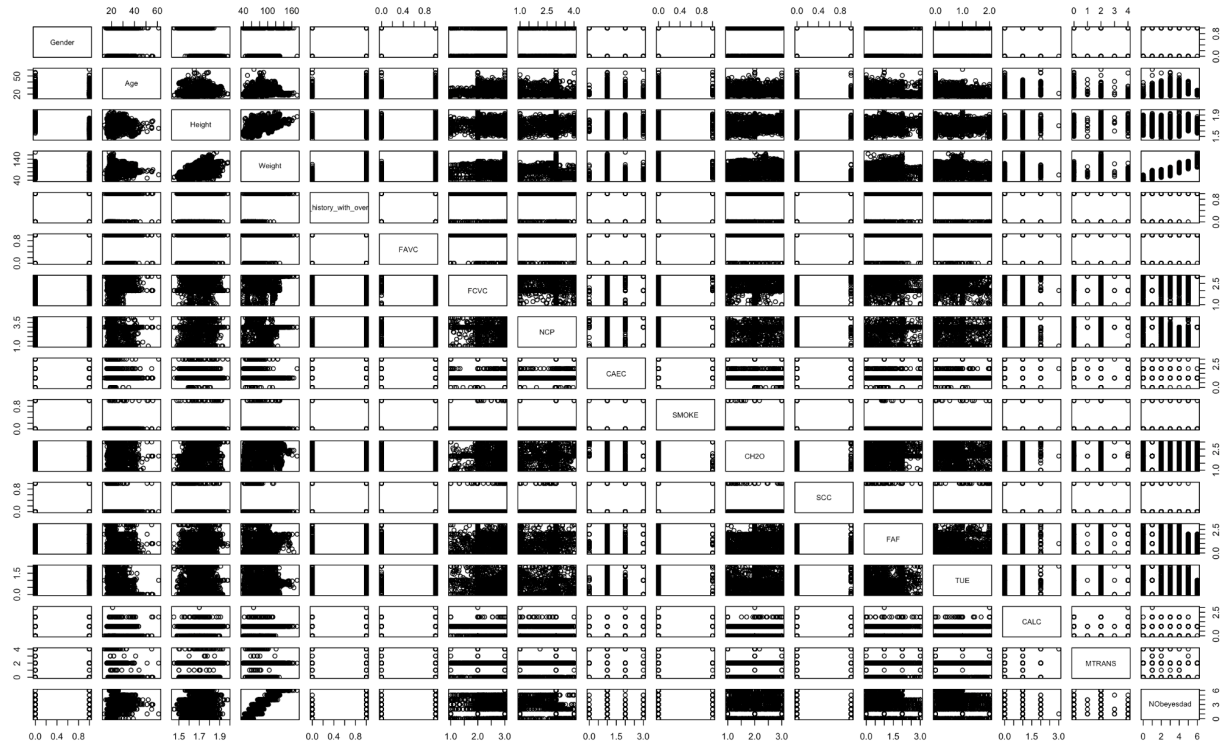


Figure 1.4 : plot() on data set : result

Figure 1.4 is a correlation matrix that shows pairwise relationships between each attribute in the dataset.

However, in this illustration, it is difficult to clearly identify correlations, except for the most obvious : the one between weight and obesity level. The image contains too much information in a small format, making it hard to interpret and nearly useless.

Let’s try another method by plotting each of the first 16 attributes separately with obesity level. The figure 1.5 shows the code to run to obtain these. In this code, we also adjust the opacity of the points to prevent overlapping, which is particularly useful since we are working with

categorical integers and not continuous variables. By doing so, when multiple points overlap, we can still distinguish areas with high concentrations from those with fewer points.

```
> # Let's plot each of th 16 values with the 17 to see possible linearity
> # But still not very visible
> plot_titles <- colnames(data)[1:16]  # Titles of the first 16 columns
> for (i in 1:16) {
+   plot(data[, i], data[, 17], main = paste("Correlation between", plot_titles[i], "and Obesity Level"),
+        xlab = plot_titles[i], ylab = "Obesity Level", col = rgb(0, 0, 0, 0.2), pch = 16)
+ }
>
```

Figure 1.5 : Source code for pairwise plotting each of the 16 attributes with the last one

Below you can find 16 figures (Figures 1.6.1 - 1.6.16) returned by the code from figure 1.5.

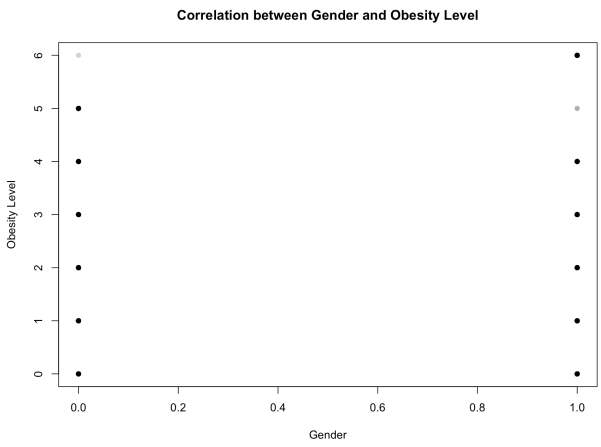


Figure 1.6.1 : Gender vs Obesity Level

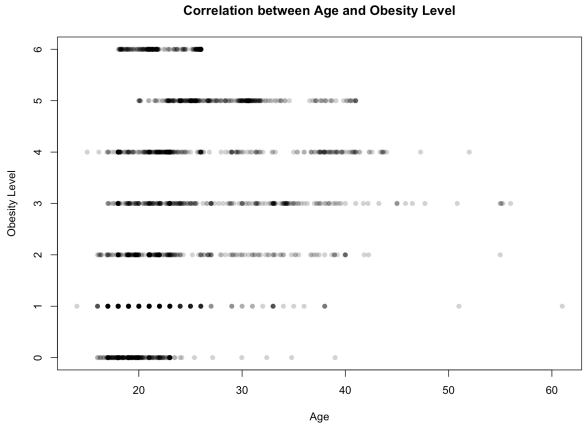


Figure 1.6.2 : Age vs Obesity Level

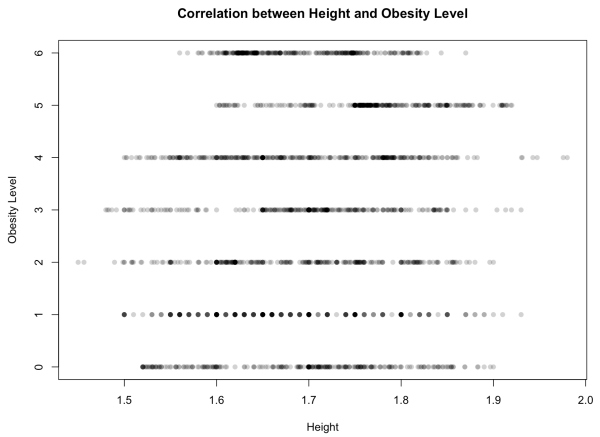


Figure 1.6.3 : Height vs Obesity Level

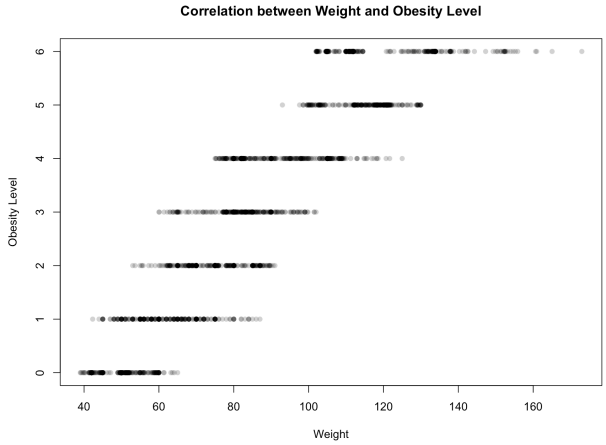


Figure 1.6.4 : Weight vs Obesity Level

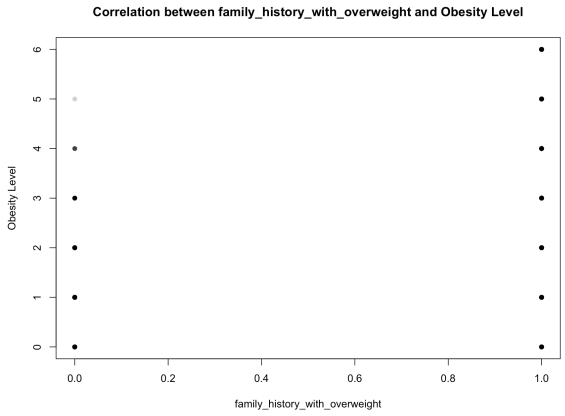


Figure 1.6.5 : family_history_with_overweight vs Obesity Level

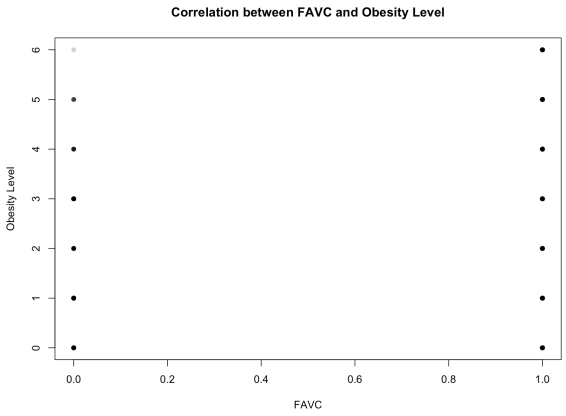
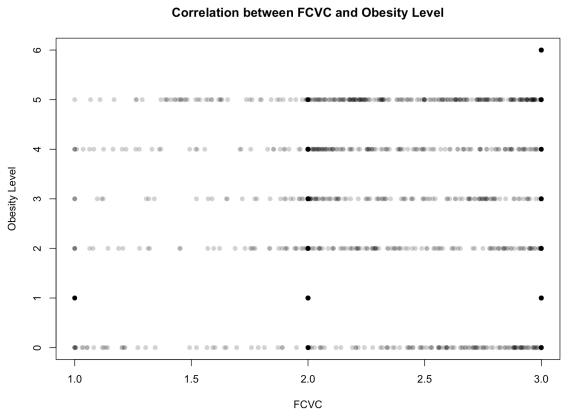


Figure 1.6.6 : FAVC vs Obesity Level



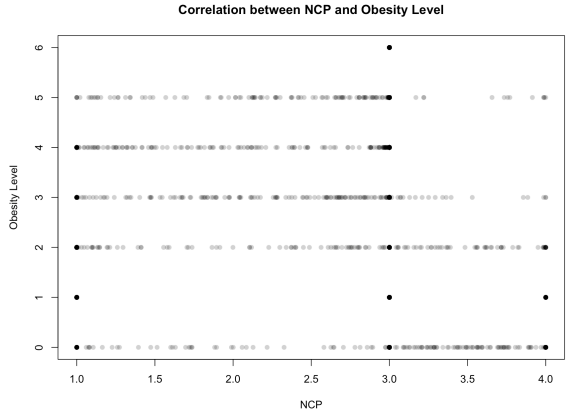
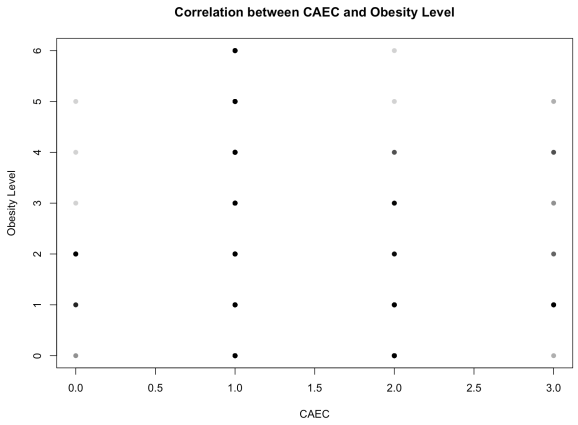


Figure 1.6.7 : FCVC vs Obesity Level

Figure 1.6.8 : NCP vs Obesity Level



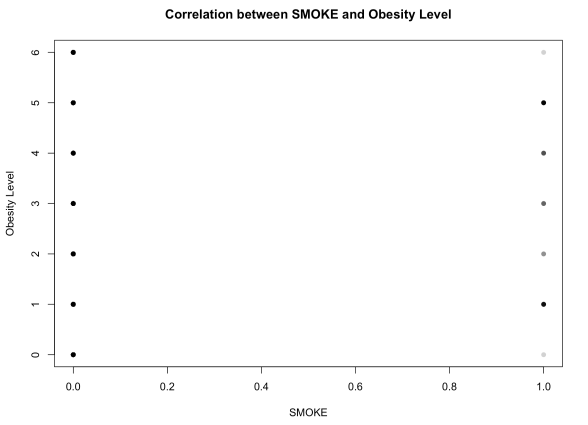
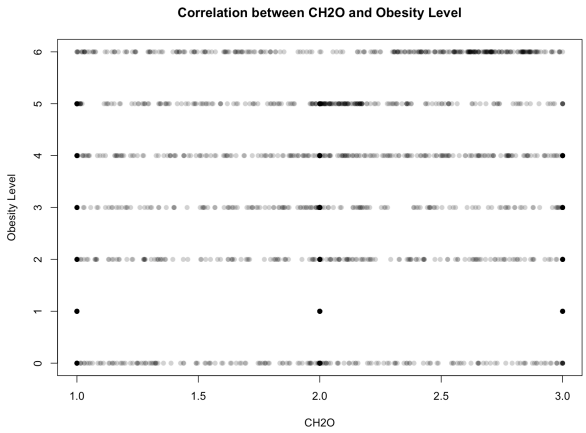


Figure 1.6.9 : CAEC vs Obesity Level

Figure 1.6.10 : SMOKE vs Obesity Level



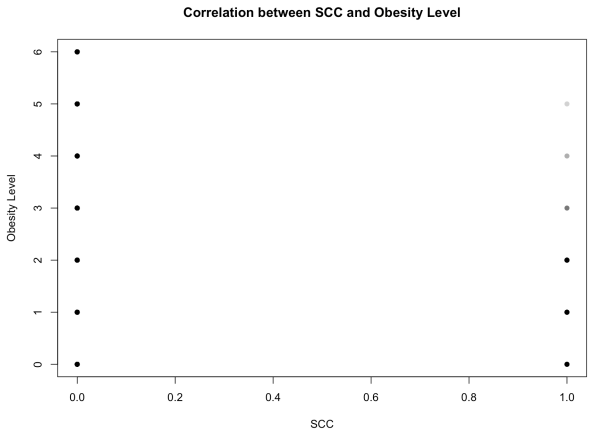


Figure 1.6.11 : CH2O vs Obesity Level

Figure 1.6.12 : SCC vs Obesity Level

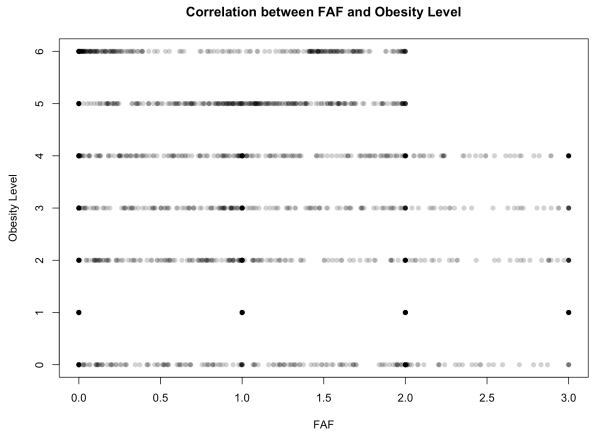


Figure 1.6.13 : FAF vs Obesity Level

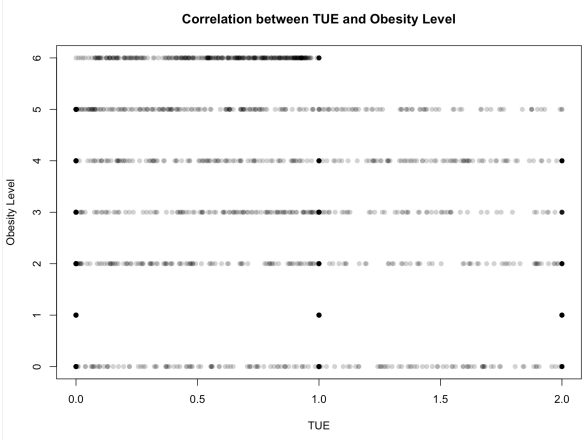


Figure 1.6.14 : TUE vs Obesity Level

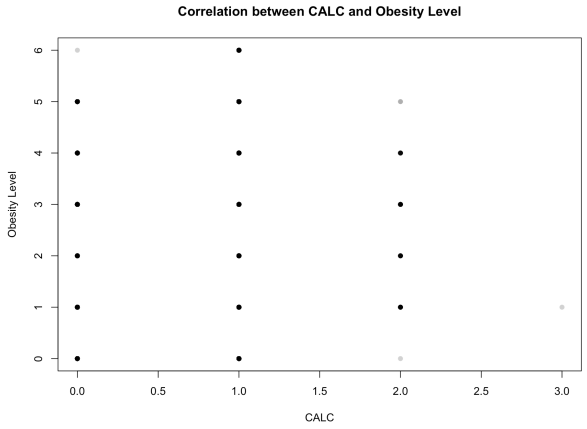


Figure 1.6.15 : CALC vs Obesity Level

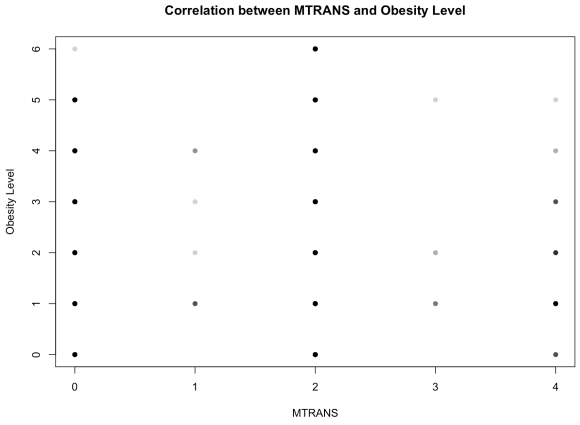


Figure 1.6.16 : MTRANS vs Obesity Level

From these graphs (Figures 1.6.1 - 1.6.16), we can more clearly see patterns.

First, in figure 1.6.1, which shows the correlation between gender and obesity level, the points are almost evenly distributed. However, the opacity of the points representing women with Obesity_Type_III is lower than for men, suggesting a slight association.

In figure 1.6.5, we can more clearly see that people without a family history of overweight tend to have lower obesity levels compared to those who do. Similarly, in figure 1.6.6, which compares FAVC (indicating whether a person frequently eats high-caloric food) with obesity level, we observe that Obesity_Type_III is more common among those who answered "yes."

figure 1.6.10, which compares SMOKE (whether a person smokes) with obesity level, shows that smokers appear less frequently in the Obesity_Type_III and Insufficient_Weight categories and are more concentrated in the middle weight categories. In contrast, non-smokers are more evenly distributed across all obesity levels.

Finally, in figure 1.6.12, which examines the relationship between obesity and SCC (whether a person monitors their calorie intake), we see that those who do not track their calories tend to have higher counts in Obesity_Type_I, Obesity_Type_II, and Obesity_Type_III compared to those who do.

However, these observations may not be entirely reliable, as the number of categories is limited, and some points still overlap.

Let’s use another plot method to study correlation, namely corrplot() function. Figure 1.7 below shows the code needed to generate the visual representation. Before plotting, we modified the name of one attribute because, in R, long names take up more space and can shrink the graph itself.

```
> # Change the column name to better see
> colnames(data.znorm)[colnames(data.znorm) == "family_history_with_overweight"] <- "family_history"
>
> # Here we can better see possible correlation
> cor_matrix <- cor(data.znorm)
>
> # Initial way to plot
> corrplot(cor_matrix)
>
```

Figure 1.7 : Source code for corrplot() : First way to do it

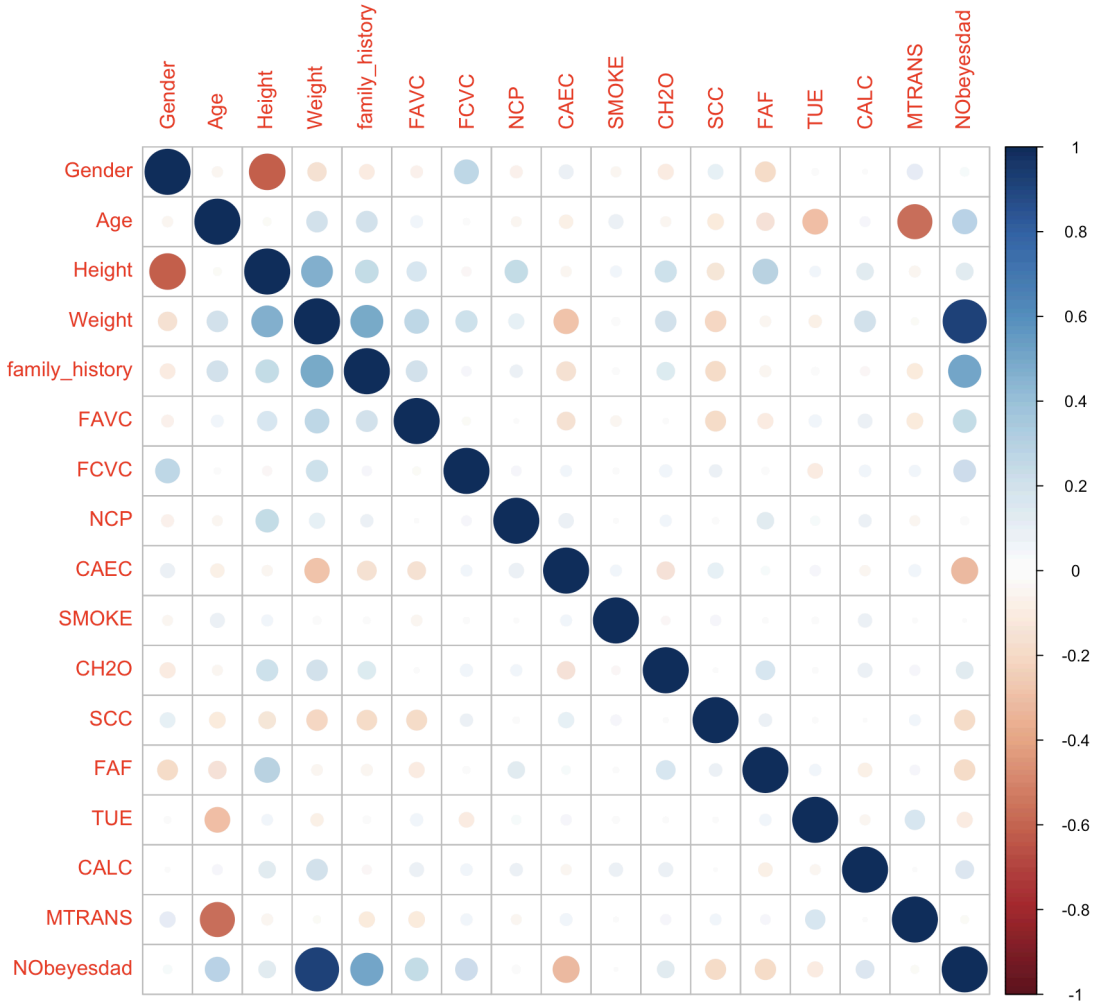


Figure 1.8 : corrplot() applied to the data set : First way

From figure 1.8, we can see correlations between attribute pairs much more clearly than with the previous plotting method. However, we can further improve the visualization by displaying the correlation coefficients. Figure 1.9 shows the code needed to achieve this and figure 1.10 shows the result.

```
# With some modifications to better visualize
corrplot(cor_matrix,
         tl.srt = 45, addCoef.col = "black")
```

Figure 1.9 : Source code for corrplot() : Second way to do it

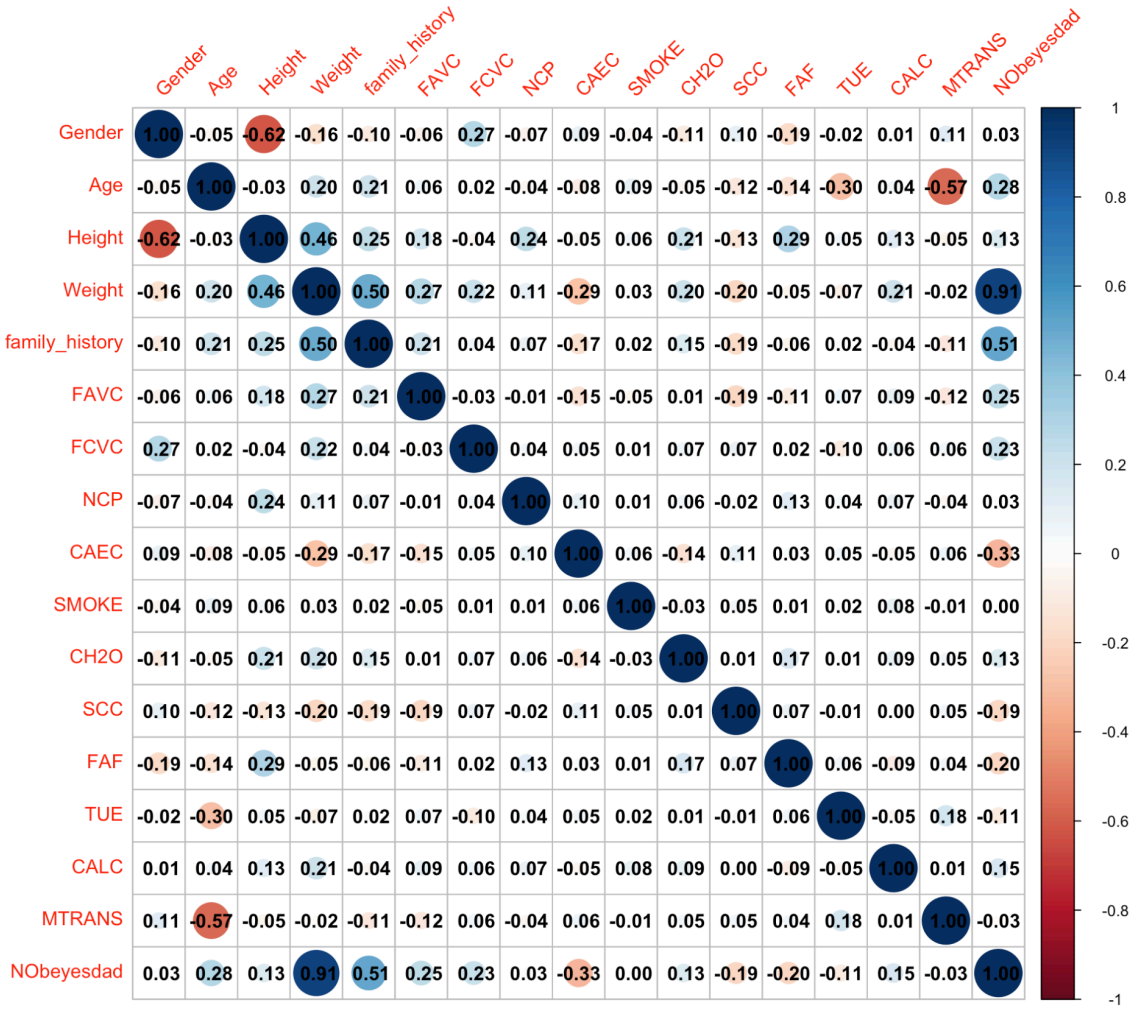


Figure 1.10 : corrplot() applied to the data set : Second way

In this matrix, if the value is positive, it means that the higher the attribute's value, the higher the obesity level, indicating a linear relationship. On the opposite, if the value is negative, a higher attribute value corresponds to a lower obesity level, meaning high negative values show an inverse correlation.

By observing figure 1.10, we can see that the attributes most correlated with obesity level are: Weight (0.91), Family History (0.51), Age(0.28), FAVC (0.25), CALC (frequency of alcohol consumption) (0.15), CH2O (daily water intake) (0.13), and Height(0.13).

On the other hand, the attributes with the most negative correlation are: CAEC (-0.33), FAF (frequency of physical activity) (-0.20), and SCC (-0.19).

3-D plotting of the most correlated attributes previously found

As we previously identified from figure 1.10, the attributes most correlated with obesity level are Weight, Family History, Age, and FAVC. However, since weight directly influences obesity level, including it in a 3D plot would not be particularly interesting.

Therefore, we will create two 3D plots:

1. Family History, Age, and Obesity Level
2. Family History, FAVC, and Obesity Level

On the other hand, we have also identified that the attributes with the most negative correlation are CAEC and FAF. Therefore, we will create a third 3D plot using CAEC, FAF, and Obesity Level.

Below, you can find the source code (figures 1.11, 1.13, and 1.15) to generate these 3D visualizations. After each code chunk, the corresponding graph is displayed (figures 1.12, 1.14, and 1.16).

```
> #-----------------------------
> # 3D Plots
> #-----------------------------
>
> # Age, family_history, and Obesity Level
> plot_ly(data, x = ~Age, y = ~family_history_with_overweight, z = ~NObeyesdad, color = ~NObeyesdad,
colors = c('#636EFA', 'green', 'red')) %>%
+   add_markers() %>%
+   layout(scene = list(
+     xaxis = list(title = 'Age'),
+     yaxis = list(title = 'family_history_with_overweight'),
+     zaxis = list(title = 'Obesity Level')))
>
```

Figure 1.11 : Source code for 3D plot : Family History, Age, and Obesity Level

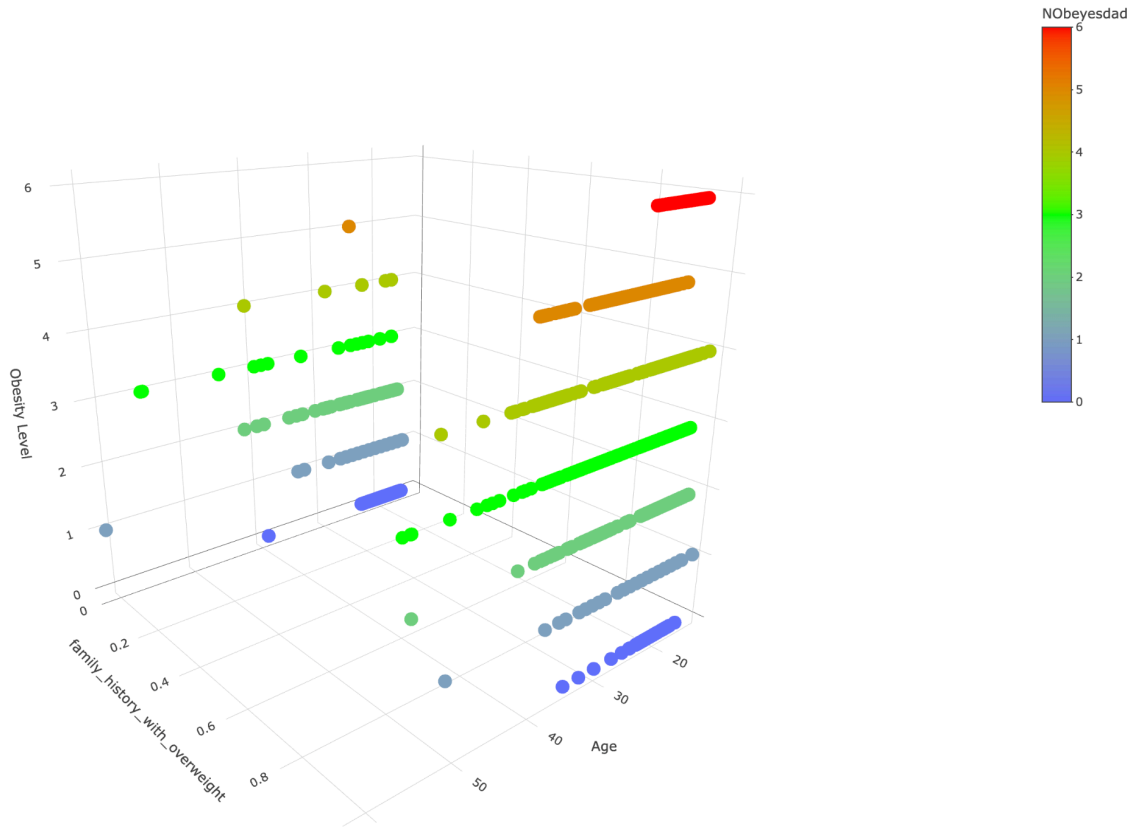


Figure 1.12 : 3D plot : Family History, Age, and Obesity Level

```
> # FAVC, family_history_with_overweight, and Obesity Level
> plot_ly(data, x = ~FAVC, y = ~family_history_with_overweight, z = ~NObeyesdad, color = ~NObeyesdad,
colors = c('#636EFA', 'green', 'red')) %>%
+   add_markers() %>%
+   layout(scene = list(
+     xaxis = list(title = 'FAVC'),
+     yaxis = list(title = 'family_history_with_overweight'),
+     zaxis = list(title = 'Obesity Level')))
>
```

Figure 1.13 : Source code for 3D plot : Family History, FAVC, and Obesity Level

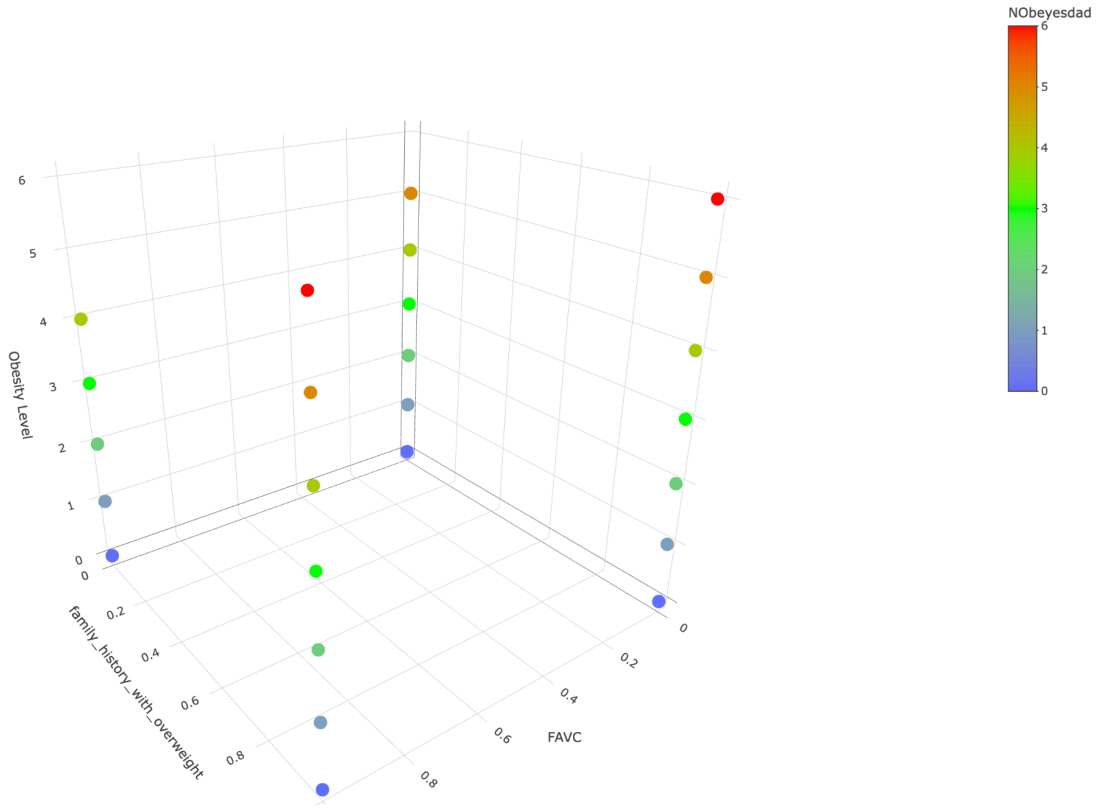


Figure 1.14 : 3D plot : Family History, FAVC, and Obesity Level

```
> # Example 3: FAVC, FAF, and Obesity Level
> plot_ly(data, x = ~CAEC, y = ~FAF, z = ~NObeyesdad, color = ~NObeyesdad, colors = c('#636EFA',
'green', 'red')) %>%
+   add_markers() %>%
+   layout(scene = list(
+     xaxis = list(title = 'CAEC'),
+     yaxis = list(title = 'FAF'),
+     zaxis = list(title = 'Obesity Level')))
>
```

Figure 1.15 : Source code for 3D plot : CAEC, FAF, and Obesity Level

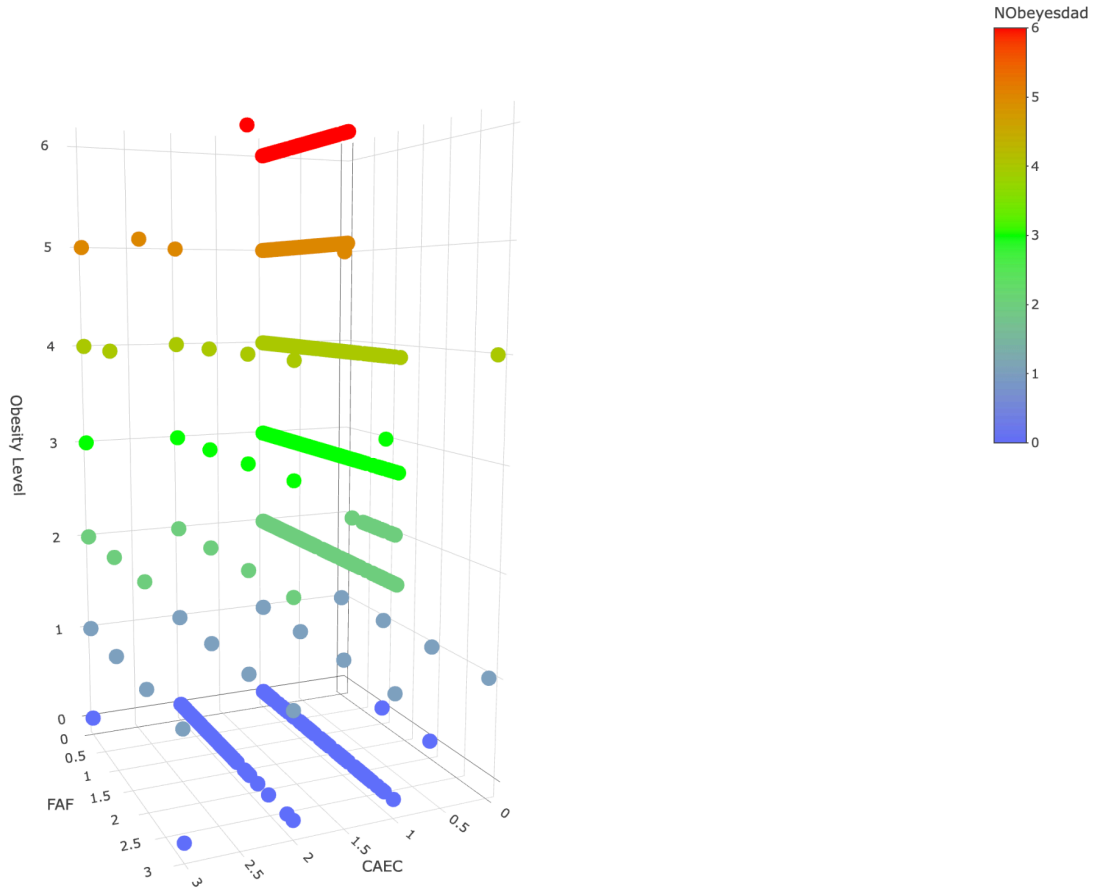


Figure 1.16 : 3D plot : CAEC, FAF, and Obesity Level

Question 2 : Data preparation

Data investigation

Let's study our data set more. To do this, we can use pre-build R functions : summary() and describe(). First, the function summary() applied to the data set (figure 2.1) reveals some interesting statistics about each attribute's distribution in the data set. Namely, this shows the min, max, first quantile, 3rd quantile, median and mean.

For example, by analyzing this output, we can determine the age distribution in the dataset:

- The youngest person is 14 years old (min = 14).
- The oldest person is 61 years old (max = 61).
- 25% of the individuals are younger than 19.95 years (1st Qu. = 19.95).
- 75% of the individuals are younger than 26 years (3rd Qu. = 26).
- Half of the individuals are younger than 22.78 years (median = 22.78).
- The average age in the dataset is 24.31 years (mean = 24.31).

We can also determine the obesity level distribution :

- The lowest obesity level is Insufficient_Weight (min = 0).
- The highest obesity level is Obesity_Type_III (max = 6).
- 25% of individuals have an obesity level of Normal_Weight (1) or lower (1st Qu. = 1).
- 75% of individuals have an obesity level of Obesity_Type_II (5) or lower (3rd Qu. = 5).
- Half of the individuals have an obesity level of Overweight_Level_II (3) or lower (median = 3).
- The average obesity level is approximately Overweight_Level_II (mean = 3.112).

```
> summary(data)
     Gender            Age            Height          Weight
 Min.   :0.0000   Min.   :14.00   Min.   :1.450   Min.   : 39.00
 1st Qu.:0.0000   1st Qu.:19.95   1st Qu.:1.630   1st Qu.: 65.47
 Median :0.0000   Median :22.78   Median :1.700   Median : 83.00
 Mean   :0.4941   Mean   :24.31   Mean   :1.702   Mean   : 86.59
 3rd Qu.:1.0000   3rd Qu.:26.00   3rd Qu.:1.768   3rd Qu.:107.43
 Max.   :1.0000   Max.   :61.00   Max.   :1.980   Max.   :173.00
 family_history_with_overweight      FAVC             FCVC            NCP
 Min.   :0.0000                 Min.   :0.0000   Min.   :1.000   Min.   :1.000
 1st Qu.:1.0000                 1st Qu.:1.0000   1st Qu.:2.000   1st Qu.:2.659
 Median :1.0000                 Median :1.0000   Median :2.386   Median :3.000
 Mean   :0.8176                 Mean   :0.8839   Mean   :2.419   Mean   :2.686
 3rd Qu.:1.0000                 3rd Qu.:1.0000   3rd Qu.:3.000   3rd Qu.:3.000
 Max.   :1.0000                 Max.   :1.0000   Max.   :3.000   Max.   :4.000
      CAEC           SMOKE              CH2O            SCC               FAF
 Min.   :0.000   Min.   :0.00000   Min.   :1.000   Min.   :0.00000   Min.   :0.0000
 1st Qu.:1.000   1st Qu.:0.00000   1st Qu.:1.585   1st Qu.:0.00000   1st Qu.:0.1245
 Median :1.000   Median :0.00000   Median :2.000   Median :0.00000   Median :1.0000
 Mean   :1.141   Mean   :0.02084   Mean   :2.008   Mean   :0.04548   Mean   :1.0103
 3rd Qu.:1.000   3rd Qu.:0.00000   3rd Qu.:2.477   3rd Qu.:0.00000   3rd Qu.:1.6667
 Max.   :3.000   Max.   :1.00000   Max.   :3.000   Max.   :1.00000   Max.   :3.0000
      TUE              CALC            MTRANS        NObeyesdad
 Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.000
 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:2.000   1st Qu.:1.000
 Median :0.6253   Median :1.0000   Median :2.000   Median :3.000
 Mean   :0.6579   Mean   :0.7314   Mean   :1.618   Mean   :3.112
 3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:2.000   3rd Qu.:5.000
 Max.   :2.0000   Max.   :3.0000   Max.   :4.000   Max.   :6.000
>
```

Figure 2.1 : summary() applied to the data set

The describe() function provides additional statistical insights about the dataset. It includes also measures like mean, median, min, and max, but at the same time introduces new ones:

- vars : the index number of the attribute in the dataset. It goes from 1 to 17.
- n : The number of rows in the dataset for each attribute. In our case it is 2111 for each.
- sd (standard deviation): Indicates how much the values deviate from the mean on average.

- trimmed (trimmed mean): The mean after removing a percentage of extreme values from both ends.
- mad (median absolute deviation): A robust measure of variability, showing how much values deviate from the median.
- range (data range): The difference between the maximum and minimum values.
- skew (skewness): Measures the asymmetry of the data distribution. Here, the value of 0 indicates a perfectly symmetric distribution.
- kurtosis (peakedness): Indicates whether the data has heavy tails or is more flat. Here, higher values mean more extreme outliers.
- se (standard error): The standard deviation divided by the square root of n, which indicates the precision of the mean estimate.

For example, by taking the Obesity level, we can deduce from this table that for this attribute :

- sd = 1.99
- trimmed = 3.14
- mad = 2.97
- range = 6
- skew = -0.08
- kurtosis = -1.22
- se = 0.04

```
> describe(data)
                              vars    n  mean    sd median trimmed   mad   min    max  range  skew
Gender                           1 2111  0.49  0.50   0.00    0.49  0.00  0.00   1.00   1.00  0.02
Age                              2 2111 24.31  6.35  22.78   23.34  4.78 14.00  61.00  47.00  1.53
Height                           3 2111  1.70  0.09   1.70    1.70  0.10  1.45   1.98   0.53 -0.01
Weight                           4 2111 86.59 26.19  83.00   85.82 32.22 39.00 173.00 134.00  0.26
family_history_with_overweight   5 2111  0.82  0.39   1.00    0.90  0.00  0.00   1.00   1.00 -1.64
FAVC                             6 2111  0.88  0.32   1.00    0.98  0.00  0.00   1.00   1.00 -2.40
FCVC                             7 2111  2.42  0.53   2.39    2.46  0.57  1.00   3.00   2.00 -0.43
NCP                              8 2111  2.69  0.78   3.00    2.77  0.00  1.00   4.00   3.00 -1.11
CAEC                             9 2111  1.14  0.47   1.00    1.05  0.00  0.00   3.00   3.00  1.90
SMOKE                           10 2111  0.02  0.14   0.00    0.00  0.00  0.00   1.00   1.00  6.70
CH2O                            11 2111  2.01  0.61   2.00    2.01  0.67  1.00   3.00   2.00 -0.10
SCC                             12 2111  0.05  0.21   0.00    0.00  0.00  0.00   1.00   1.00  4.36
FAF                             13 2111  1.01  0.85   1.00    0.94  1.19  0.00   3.00   3.00  0.50
TUE                             14 2111  0.66  0.61   0.63    0.59  0.72  0.00   2.00   2.00  0.62
CALC                            15 2111  0.73  0.52   1.00    0.75  0.00  0.00   3.00   3.00 -0.24
MTRANS                          16 2111  1.62  0.91   2.00    1.70  0.00  0.00   4.00   4.00 -0.67
NObeyesdad                      17 2111  3.11  1.99   3.00    3.14  2.97  0.00   6.00   6.00 -0.08
                              kurtosis   se
Gender                           -2.00 0.01
Age                               2.81 0.14
Height                           -0.57 0.00
Weight                           -0.70 0.57
family_history_with_overweight    0.70 0.01
FAVC                              3.74 0.01
FCVC                             -0.64 0.01
NCP                               0.38 0.02
CAEC                              5.38 0.01
SMOKE                            42.95 0.00
CH2O                             -0.88 0.01
SCC                              17.02 0.00
FAF                              -0.62 0.02
TUE                              -0.55 0.01
CALC                             -0.33 0.01
MTRANS                            0.39 0.02
NObeyesdad                       -1.22 0.04
>
```

Figure 2.2 : describe() applied to the data set

Subset with the most correlated attributes

As mentioned in Question 1, the attributes most correlated with obesity level are Weight, Family History, Age, FAVC, and CALC. Therefore, we will select these attributes for our subset.

Figure 2.3 shows the source code to create this subset and displays the first 10 rows of the new dataset.

```
> # Creating the subset of the data set with the most correlated attributes
> subset_data <- data %>%
+   select(Weight, family_history_with_overweight, Age, FAVC, CALC, NObeyesdad)
> subset_data[1:10,]
   Weight family_history_with_overweight Age FAVC CALC NObeyesdad
1    64.0                             1  21    0    0          1
2    56.0                             1  21    0    1          1
3    77.0                             1  23    0    2          1
4    87.0                             0  27    0    2          2
5    89.8                             0  22    0    1          3
6    53.0                             0  29    1    1          1
7    55.0                             1  23    1    1          1
8    53.0                             0  22    0    1          1
9    64.0                             1  24    1    2          1
10   68.0                             1  22    1    0          1
>
```

Figure 2.3 : Source code to create a subset of the most correlated attributes

Data splitting into training and testing data sets

Below (figure 2.4), you can find the source code for splitting the data set into training and testing sets. The dataset is divided in three different ways: 70-30%, 60-40%, and 50-50%.

```
#-------------------------------------------
# d - Creating training and testing sets
#-------------------------------------------

# Normalizing our initial subset
subset_data.norm <- as.data.frame(lapply(subset_data, normalize))

# Dividing into training and testing sets
subset_data.norm.rows = nrow(subset_data.norm)

# First training and testing sets with 70-30% division
subset_data.norm.sample1 = 0.7
subset_data.rows1 = subset_data.norm.sample1 * subset_data.norm.rows

subset_data.train.index1 = sample(subset_data.norm.rows, subset_data.rows1)

subset_data.train1 = subset_data.norm[subset_data.train.index1,]
subset_data.test1 = subset_data.norm[-subset_data.train.index1,]

# Second training and testing sets with 60-40% division
subset_data.norm.sample2 = 0.6
subset_data.rows2 = subset_data.norm.sample2 * subset_data.norm.rows

subset_data.train.index2 = sample(subset_data.norm.rows, subset_data.rows2)

subset_data.train2 = subset_data.norm[subset_data.train.index2,]
subset_data.test2 = subset_data.norm[-subset_data.train.index2,]

# Third training and testing sets with 50-50% division
subset_data.norm.sample3 = 0.5
subset_data.rows3 = subset_data.norm.sample3 * subset_data.norm.rows

subset_data.train.index3 = sample(subset_data.norm.rows, subset_data.rows3)

subset_data.train3 = subset_data.norm[subset_data.train.index3,]
subset_data.test3 = subset_data.norm[-subset_data.train.index3,]
```

Figure 2.4 : Source code to create training and testing data sets

Question 3 : Clustering on the whole Data Set

First, let's upload our data set using read.table() function, and print some information about this data using str() function. The screenshot below (figure 3.1) shows the code to perform these actions and the result.

```
> path <- file.choose()
>
> data <- read.table(path, sep = ",", header = TRUE, stringsAsFactors = FALSE)
>
> str(data)
'data.frame':   2111 obs. of  17 variables:
 $ Gender                        : chr  "Female" "Female" "Male" "Male" ...
 $ Age                           : num  21 21 23 27 22 29 23 22 24 22 ...
 $ Height                        : num  1.62 1.52 1.8 1.8 1.78 1.62 1.5 1.64 1.78 1.72 ...
 $ Weight                        : num  64 56 77 87 89.8 53 55 53 64 68 ...
 $ family_history_with_overweight: chr  "yes" "yes" "yes" "no" ...
 $ FAVC                          : chr  "no" "no" "no" "no" ...
 $ FCVC                          : num  2 3 2 3 2 2 3 2 3 2 ...
 $ NCP                           : num  3 3 3 3 1 3 3 3 3 3 ...
 $ CAEC                          : chr  "Sometimes" "Sometimes" "Sometimes" "Sometimes" ...
 $ SMOKE                         : chr  "no" "yes" "no" "no" ...
 $ CH2O                          : num  2 3 2 2 2 2 2 2 2 2 ...
 $ SCC                           : chr  "no" "yes" "no" "no" ...
 $ FAF                           : num  0 3 2 2 0 0 1 3 1 1 ...
 $ TUE                           : num  1 0 1 0 0 0 0 0 1 1 ...
 $ CALC                          : chr  "no" "Sometimes" "Frequently" "Frequently" ...
 $ MTRANS                        : chr  "Public_Transportation" "Public_Transportation" "Public_Transportatio
n" "Walking" ...
 $ NObeyesdad                    : chr  "Normal_Weight" "Normal_Weight" "Normal_Weight" "Overweight_Level_I"
...
```

Figure 3.1 : Data set's import

Then, as before for convenience, let's convert alphanumeric values into numeric values using the same system as before (See mapping table 2.1 for these values).

To do so, we need to run the code below (figure 3.2) :

```r
# Converting categorical variables to numeric
data <- data %>%
  mutate(
    Gender = ifelse(Gender == "Female", 1, 0),

    family_history_with_overweight = ifelse(family_history_with_overweight == "yes", 1, 0),

    FAVC = ifelse(FAVC == "yes", 1, 0),

    CAEC = case_when(
      CAEC == "no" ~ 0,
      CAEC == "Sometimes" ~ 1,
      CAEC == "Frequently" ~ 2,
      CAEC == "Always" ~ 3
    ),

    SMOKE = ifelse(SMOKE == "yes", 1, 0),

    SCC = ifelse(SCC == "yes", 1, 0),

    CALC = case_when(
      CALC == "no" ~ 0,
      CALC == "Sometimes" ~ 1,
      CALC == "Frequently" ~ 2,
      CALC == "Always" ~ 3
    ),

    MTRANS = case_when(
      MTRANS == "Automobile" ~ 0,
      MTRANS == "Motorbike" ~ 1,
      MTRANS == "Public_Transportation" ~ 2,
      MTRANS == "Bike" ~ 3,
      MTRANS == "Walking" ~ 4
    ),

    NObeyesdad = case_when(
      NObeyesdad == "Obesity_Type_III" ~ 6,
      NObeyesdad == "Obesity_Type_II" ~ 5,
      NObeyesdad == "Obesity_Type_I" ~ 4,
      NObeyesdad == "Overweight_Level_II" ~ 3,
      NObeyesdad == "Overweight_Level_I" ~ 2,
      NObeyesdad == "Normal_Weight" ~ 1,
      NObeyesdad == "Insufficient_Weight" ~ 0
    )
  )
```

Figure 3.2 : Mapping each character field into numeric

Next, let's write two different normalization functions. The first one will convert all values to a range between 0 and 1, while the second one will normalize the values so that each attribute has a mean of 0 and a variance close to 1.

The following screenshot (figure 3.5) shows the code used to implement these functions and applies the first one to the dataset. From the output, we can see that all data is now within the range of 0 and 1.

```
> # Normalization functions
> normalize <- function(x) {((x-min(x))/(max(x) - min(x)))}
>
> zscore <- function(x){(x-mean(x))/sd(x)}
>
> data.norm <- as.data.frame(lapply(data, normalize))
> data.norm[1:10,]
   Gender       Age     Height    Weight family_history_with_overweight FAVC FCVC       NCP      CAEC
1       1 0.1489362 0.32075472 0.1865672                                1    0  0.5 0.6666667 0.3333333
2       1 0.1489362 0.13207547 0.1268657                                1    0  1.0 0.6666667 0.3333333
3       0 0.1914894 0.66037736 0.2835821                                1    0  0.5 0.6666667 0.3333333
4       0 0.2765957 0.66037736 0.3582090                                0    0  1.0 0.6666667 0.3333333
5       0 0.1702128 0.62264151 0.3791045                                0    0  0.5 0.0000000 0.3333333
6       0 0.3191489 0.32075472 0.1044776                                0    1  0.5 0.6666667 0.3333333
7       1 0.1914894 0.09433962 0.1194030                                1    1  1.0 0.6666667 0.3333333
8       0 0.1702128 0.35849057 0.1044776                                0    0  0.5 0.6666667 0.3333333
9       0 0.2127660 0.62264151 0.1865672                                1    1  1.0 0.6666667 0.3333333
10      0 0.1702128 0.50943396 0.2164179                                1    1  0.5 0.6666667 0.3333333
   SMOKE CH2O SCC       FAF TUE      CALC MTRANS NObeyesdad
1      0  0.5   0 0.0000000 0.5 0.0000000   0.50  0.1666667
2      1  1.0   1 1.0000000 0.0 0.3333333   0.50  0.1666667
3      0  0.5   0 0.6666667 0.5 0.6666667   0.50  0.1666667
4      0  0.5   0 0.6666667 0.0 0.6666667   1.00  0.3333333
5      0  0.5   0 0.0000000 0.0 0.3333333   0.50  0.5000000
6      0  0.5   0 0.0000000 0.0 0.3333333   0.00  0.1666667
7      0  0.5   0 0.3333333 0.0 0.3333333   0.25  0.1666667
8      0  0.5   0 1.0000000 0.0 0.3333333   0.50  0.1666667
9      0  0.5   0 0.3333333 0.5 0.6666667   0.50  0.1666667
10     0  0.5   0 0.3333333 0.5 0.0000000   0.50  0.1666667
>
```

Figure 3.5 : Normalisation functions and application of normalization function on the data set

In the next screenshot (figure 3.6), we can see the application of the z-normalization function on the dataset. From the output, we can observe that the mean is around 0 and the standard deviation is approximately 1, which confirms that our function worked correctly and as we wanted.

```
> data.znorm <- as.data.frame(lapply(data, zscore))
> data.znorm[1:10,]
       Gender         Age     Height      Weight family_history_with_overweight       FAVC       FCVC
1   1.0116740 -0.52200070 -0.8753819 -0.86235386                      0.4721794 -2.7591155 -0.7848327
2   1.0116740 -0.52200070 -1.9471379 -1.16780030                      0.4721794 -2.7591155  1.0880839
3  -0.9879925 -0.20683997  1.0537789 -0.36600341                      0.4721794 -2.7591155 -0.7848327
4  -0.9879925  0.42348149  1.0537789  0.01580463                     -2.1168357 -2.7591155  1.0880839
5  -0.9879925 -0.36442034  0.8394277  0.12271089                     -2.1168357 -2.7591155 -0.7848327
6  -0.9879925  0.73864222 -0.8753819 -1.28234271                     -2.1168357  0.3622633 -0.7848327
7   1.0116740 -0.20683997 -2.1614891 -1.20598110                      0.4721794  0.3622633  1.0880839
8  -0.9879925 -0.36442034 -0.6610307 -1.28234271                     -2.1168357 -2.7591155 -0.7848327
9  -0.9879925 -0.04925961  0.8394277 -0.86235386                      0.4721794  0.3622633  1.0880839
10 -0.9879925 -0.36442034  0.1963741 -0.70963065                      0.4721794  0.3622633 -0.7848327
         NCP       CAEC      SMOKE        CH2O        SCC        FAF        TUE       CALC     MTRANS
1   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203 -1.1877577  0.5618636 -1.4188354  0.4177850
2   0.404057 -0.3002744  6.8523733  1.61837509  4.5803537  2.3391959 -1.0803687  0.5210361  0.4177850
3   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203  1.1635447  0.5618636  2.4609075  0.4177850
4   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203  1.1635447 -1.0803687  2.4609075  2.6062321
5  -2.166509 -0.3002744 -0.1458657 -0.01307017 -0.2182203 -1.1877577 -1.0803687  0.5210361  0.4177850
6   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203 -1.1877577 -1.0803687  0.5210361 -1.7706621
7   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203 -0.0121065 -1.0803687  0.5210361 -0.6764385
8   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203  2.3391959 -1.0803687  0.5210361  0.4177850
9   0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203 -0.0121065  0.5618636  2.4609075  0.4177850
10  0.404057 -0.3002744 -0.1458657 -0.01307017 -0.2182203 -0.0121065  0.5618636 -1.4188354  0.4177850
    NObeyesdad
1  -1.06408228
2  -1.06408228
3  -1.06408228
4  -0.56031962
5  -0.05655696
6  -1.06408228
7  -1.06408228
8  -1.06408228
9  -1.06408228
10 -1.06408228
>
```

Figure 3.6 : Application of z-normalization function on the data set

Next, let's use the corrplot() function from the rubric to identify possible correlation patterns between attributes. But, as in question 1, before plotting, we will modify the name of one

attribute because long names in R take up more space and can shrink the graph. The source code to perform these actions is shown in figure 3.7 and the result obtained is in figure 3.8.

```
colnames(data)[colnames(data) == "family_history_with_overweight"] <- "family_history"
cor_matrix <- cor(data)
corrplot(cor_matrix)
colnames(data)[colnames(data) == "family_history"] <- "family_history_with_overweight"
```

Figure 3.7 : Code for corrplot on the data set

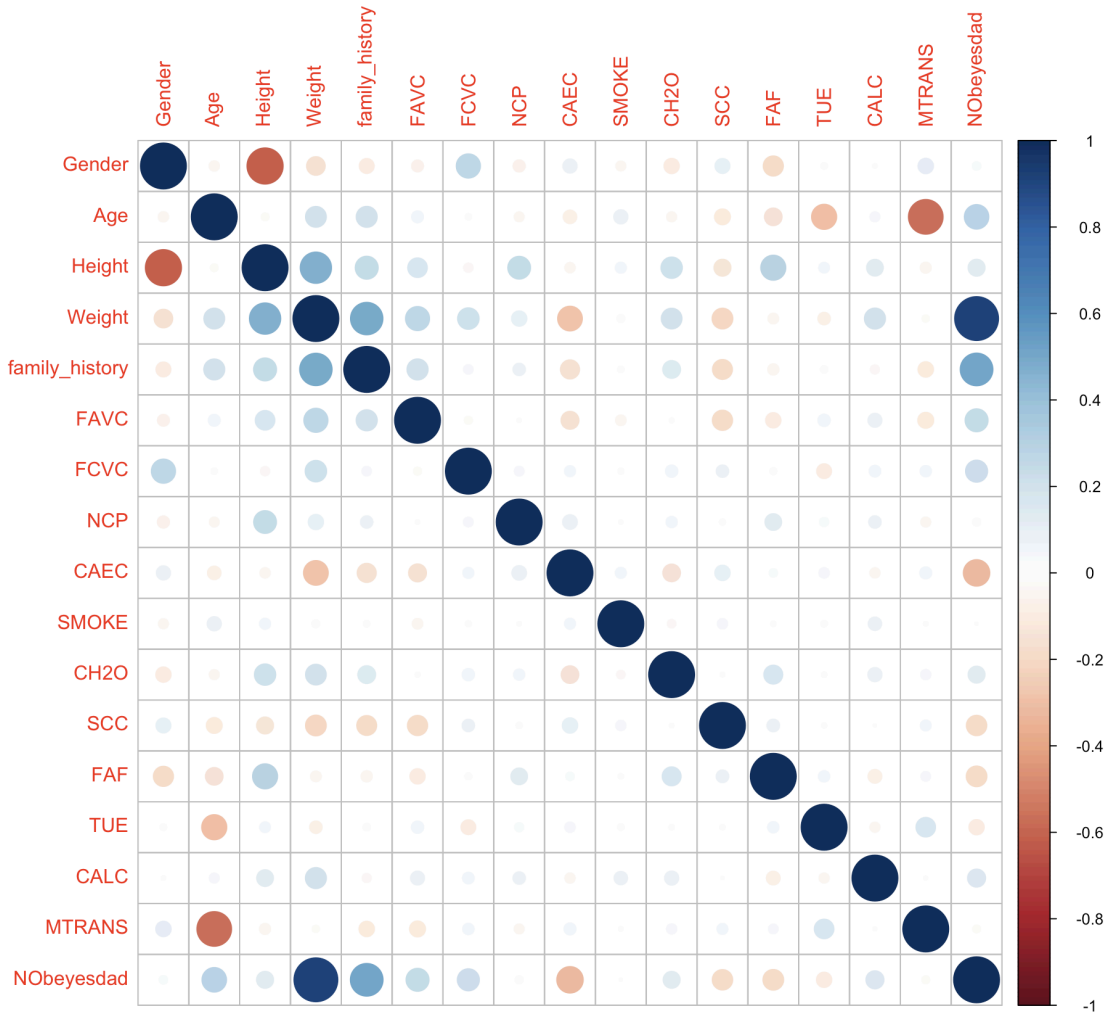


Figure 3.8 : Corrplot illustration

From figure 3.8, we can see a positive correlation between weight and height, age and weight, having family history with overweight and weight, among others (indicated by blue circles). There is also a negative correlation between MTRANS (which indicates the most frequently used mode of transport, ranging from the most unhealthy to the healthiest) and age (meaning that as people get older, they tend to choose more unhealthy and less physically active transportation methods), as well as between CAEC and weight, and so on (indicated by red circles). Additionally, we can observe that, in general, men tend to be taller than women.

Next, let's apply the kmeans() function to our dataset to observe how it divides the values into classes. First, we'll try using 3 centers, as shown in figure 3.9.

```
> # Clustering on the whole data set :
> object <- kmeans(data.norm, centers = 3, iter.max = 10, nstart = 25)
>
> str(object)
List of 9
 $ cluster     : int [1:2111] 1 1 3 2 2 2 1 2 3 3 ...
 $ centers     : num [1:3, 1:17] 1 0.606 0 0.232 0.159 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:3] "1" "2" "3"
  .. ..$ : chr [1:17] "Gender" "Age" "Height" "Weight" ...
 $ totss       : num 2689
 $ withinss    : num [1:3] 645 484 661
 $ tot.withinss: num 1790
 $ betweenss   : num 899
 $ size        : int [1:3] 811 383 917
 $ iter        : int 3
 $ ifault      : int 0
 - attr(*, "class")= chr "kmeans"
>
```

Figure 3.9 : kmeans() function applied to normalized data set with 3 centers

Below, in figure 3.10, we can see the object returned by the kmeans() function from figure 3.9. It shows the sizes of the 3 clusters (811, 383, and 917), the cluster means for each attribute, the clustering vector (which indicates the cluster associated with each row in the dataset), the measurement of how well the association of clusters was achieved (through the within-cluster sum of squares for each cluster), and the overall score, which is 33.4%. The overall score here is quite low. In fact here, the dataset has 17 attributes and 2,111 points, while we are trying to associate them with only 3 centers, thus, many points will likely be far from their associated centers.

```
> object
K-means clustering with 3 clusters of sizes 811, 383, 917

Cluster means:
     Gender       Age    Height    Weight family_history_with_overweight      FAVC      FCVC       NCP
1 1.0000000 0.2320370 0.3837716 0.3888693                      1.000000 0.9001233 0.7988805 0.5533436
2 0.6057441 0.1594562 0.3810112 0.1478477                      0.000000 0.7415144 0.6876814 0.5219427
3 0.0000000 0.2332995 0.5946233 0.4118422                      0.997819 0.9291167 0.6396138 0.5861009
       CAEC      SMOKE      CH2O        SCC       FAF       TUE      CALC    MTRANS NObeyesdad
1 0.3711467 0.01726264 0.5133061 0.03699137 0.2678564 0.3167783 0.2412659 0.4180025  0.6502261
2 0.4369017 0.01566580 0.4065706 0.12793734 0.3709760 0.3131755 0.2558747 0.4588773  0.1623151
3 0.3645947 0.02617230 0.5364758 0.01853871 0.3834215 0.3462640 0.2410033 0.3699564  0.5512541

Clustering vector:
   [1] 1 1 3 2 2 2 1 2 3 3 3 1 2 3 3 1 3 2 1 1 3 1 1 1 3 3 3 2 3 2 2 1 2 2 2 1 2 2 3 1 1 2 1 3 2 1 3 3 1 2
  [51] 1 2 2 1 2 2 3 3 1 3 1 2 2 3 1 1 3 3 3 3 1 1 1 3 3 1 1 1 1 3 2 3 3 2 3 3 3 3 1 1 2 1 3 2 2 3 1 2 2 2
 [101] 2 3 2 1 1 2 2 3 3 1 3 1 1 1 1 1 1 3 1 1 3 3 3 2 3 2 2 3 2 1 2 1 2 3 3 3 3 3 2 3 2 2 2 3 2 2 1 2 1
 [151] 3 1 1 3 3 1 1 3 1 3 3 2 1 2 3 3 1 3 1 1 2 3 2 2 2 1 1 3 3 2 2 1 2 1 2 2 3 3 3 1 2 3 3 2 1 3 3 3 1 1
 [201] 1 1 1 3 3 1 3 3 1 1 3 2 1 3 3 3 1 2 1 1 1 3 1 1 1 3 2 2 2 3 3 1 1 1 1 1 2 1 2 1 2 2 1 3 1 2 2 2 3 3
 [251] 1 3 3 1 1 3 3 2 2 1 2 1 2 2 3 2 2 2 2 3 3 2 2 3 2 1 3 2 1 2 2 3 1 2 3 3 2 1 2 2 3 2 3 3 3 2 2 2 3 2
 [301] 2 2 3 2 3 3 1 1 2 2 3 2 2 3 2 2 3 2 2 2 3 3 2 3 1 2 3 3 3 1 2 3 3 2 2 2 2 3 2 2 2 2 3 2 3 2 1 3 3 3
 [351] 3 2 2 1 2 2 2 1 3 1 2 1 1 2 3 2 3 2 1 2 2 3 2 2 3 3 1 2 2 3 2 3 1 3 1 2 2 1 3 1 3 2 2 2 1 1 1 2 3 2
 [401] 3 2 3 1 2 2 2 1 3 1 2 2 2 2 3 3 2 3 2 3 3 3 2 3 2 1 2 2 2 3 2 2 2 1 2 1 2 1 2 2 2 1 3 3 3 2 3 3 1 1
 [451] 2 3 2 2 2 2 2 2 2 3 1 3 3 3 2 1 3 3 2 2 2 1 2 1 2 2 1 1 2 2 2 3 1 2 1 3 2 3 3 2 2 2 2 2 2 2 3 3 2 1 1
 [501] 1 1 1 1 1 1 1 3 3 3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 2 2 2 2 2 2 1 1 1 2 2 2 1
 [551] 1 1 2 2 2 2 2 2 3 3 3 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
 [601] 2 2 2 2 2 2 3 3 3 3 1 2 2 2 2 2 1 3 1 2 2 1 2 1 2 2 3 2 3 1 2 2 2 3 1 2 2 3 3 3 2 2 3 3 3 1 1 1 1 2
 [651] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 3 3 3 3 1 3 2 2 2 2 2 2 1 3 1 2 2 2 1 1 1 2 2 2 2 3 3 3 3 3 2 2 2
 [701] 3 3 3 1 3 1 2 2 2 2 2 2 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 2 2 2 2 2 2 3 3 3 2 2 2 3 2
 [751] 3 2 3 3 1 3 1 1 2 2 1 1 2 2 2 1 3 3 3 3 3 1 3 1 1 3 3 1 3 3 3 2 1 3 2 1 2 1 3 1 2 1 1 2 1 2 1 3 3 1
 [801] 1 1 2 2 2 2 2 2 3 1 1 3 3 1 3 3 3 3 3 3 3 1 1 1 1 2 1 1 1 1 1 2 2 2 2 2 1 1 3 3 3 3 3 3 1 1 1 1 3 3
 [851] 3 1 1 1 3 3 1 3 3 3 3 2 1 1 3 2 2 1 2 2 1 3 3 1 1 1 1 1 1 2 2 1 2 1 1 1 3 3 1 1 1 1 2 2 2 2 2 3 3 3
 [901] 1 3 2 2 3 3 3 1 3 3 1 1 1 2 2 1 1 1 1 1 2 2 2 2 1 1 1 3 3 3 1 1 3 3 3 1 1 1 1 3 3 3 1 3 3 3 3 2 1 3
 [951] 3 2 2 1 2 1 3 1 2 1 1 3 2 2 1 1 2 1 1 3 3 3 1 1 1 1 2 3 3 3 3 3 3 3 3 3 1 3 3 1 1 3 1 1 3 3 1 1 3 3
 [ reached getOption("max.print") -- omitted 1111 entries ]

Within cluster sum of squares by cluster:
[1] 644.8771 483.6816 661.3125
 (between_SS / total_SS =  33.4 %)

Available components:

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

Figure 3.10 : the object returned by kmeans() function applied to normalized data set with 3 centers

Next, the rubric file also demonstrates how to print the centers and the classes of each row of the dataset using the built-in fitted() function. The code to perform this action, as well as the result, are illustrated in figures 3.11 and 3.12, respectively.

```
> # Get cluster centers for each point
> fitted(object, method = "centers")
     Gender       Age    Height    Weight family_history_with_overweight      FAVC      FCVC       NCP
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
3 0.0000000 0.2332995 0.5946233 0.4118422                       0.997819 0.9291167 0.6396138 0.5861009
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
1 1.0000000 0.2320370 0.3837716 0.3888693                       1.000000 0.9001233 0.7988805 0.5533436
2 0.6057441 0.1594562 0.3810112 0.1478477                       0.000000 0.7415144 0.6876814 0.5219427
```

Figure 3.11 : The output of the pre-implemented function fitted() to print all points' centers

```
> # Get cluster assignments (which cluster each point belongs to)
> fitted(object, method = "classes")
   [1] 1 1 3 2 2 2 1 2 3 3 3 1 2 3 3 1 3 2 1 1 3 1 1 1 3 3 3 2 3 2 2 1 2 2 2 1 2 2 3 1 1 2 1 3 2 1 3 3 1 2
  [51] 1 2 2 1 2 2 3 3 1 3 1 2 2 3 1 1 3 3 3 3 1 1 1 3 3 1 1 1 1 3 2 3 3 2 3 3 3 3 1 1 2 1 3 2 2 3 1 2 2 2
 [101] 2 3 2 1 1 2 2 3 3 1 3 1 1 1 1 1 1 1 3 1 1 3 3 3 2 3 2 2 3 2 1 2 1 2 3 3 3 3 3 2 3 2 2 2 3 2 2 1 2 1
 [151] 3 1 1 3 3 1 1 3 1 3 3 2 1 2 3 3 1 3 1 1 2 3 2 2 2 1 1 3 3 2 2 1 2 1 2 2 3 3 3 1 2 3 3 2 1 3 3 3 1 1
 [201] 1 1 1 3 3 1 3 3 1 1 3 2 1 3 3 3 1 2 1 1 1 3 1 1 1 3 2 2 2 3 3 1 1 1 1 1 2 1 2 1 2 2 1 3 1 2 2 2 3 3
 [251] 1 3 3 1 1 3 3 2 2 1 2 1 2 2 3 2 2 2 2 3 3 2 2 3 2 1 3 2 1 2 2 3 1 2 3 3 2 1 2 2 3 2 3 3 3 2 2 2 3 2
 [301] 2 2 3 2 3 3 1 1 2 2 3 2 2 3 2 2 3 2 2 2 3 3 2 3 1 2 3 3 3 1 2 3 3 2 2 2 2 3 2 2 2 2 3 2 3 2 1 3 3 3
 [351] 3 2 2 1 2 2 2 1 3 1 2 1 1 2 3 2 3 2 1 2 2 3 2 2 3 3 1 2 2 3 2 3 1 3 1 2 2 1 3 1 3 2 2 2 1 1 1 2 3 2
 [401] 3 2 3 1 2 2 2 1 3 1 2 2 2 2 3 3 2 3 2 3 3 3 2 3 2 1 2 2 2 3 2 2 2 1 2 1 2 1 2 2 2 1 3 3 3 2 3 3 1 1
 [451] 2 3 2 2 2 2 2 2 2 3 1 3 3 3 2 1 3 3 2 2 2 1 2 1 2 2 1 1 2 2 2 3 1 2 1 3 2 3 3 2 2 2 2 2 2 3 3 2 1 1
 [501] 1 1 1 1 1 1 1 3 3 3 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 2 2 2 2 2 2 1 1 1 2 2 2 1
 [551] 1 1 2 2 2 2 2 2 3 3 3 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3
 [601] 2 2 2 2 2 2 3 3 3 3 1 2 2 2 2 2 1 3 1 2 2 1 2 1 2 2 3 2 3 1 2 2 2 3 1 2 2 3 3 3 2 2 3 3 3 1 1 1 1 2
 [651] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 3 3 3 3 1 3 2 2 2 2 2 2 1 3 1 2 2 2 1 1 1 2 2 2 2 3 3 3 3 3 2 2 2
 [701] 3 3 3 1 3 1 2 2 2 2 2 2 2 2 2 3 3 3 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 3 3 3 2 2 2 3 2
 [751] 3 2 3 3 1 3 1 1 2 2 1 1 2 2 2 1 3 3 3 3 3 1 3 1 1 3 3 1 3 3 3 2 1 3 2 1 2 1 3 1 2 1 1 2 1 2 1 3 3 1
 [801] 1 1 2 2 2 2 2 2 3 1 1 3 3 1 3 3 3 3 3 3 1 1 1 1 2 1 1 1 1 1 2 2 2 2 2 1 1 3 3 3 3 3 3 3 1 1 1 1 3 3
 [851] 3 1 1 1 3 3 1 3 3 3 3 2 1 1 3 2 2 1 2 2 1 3 3 1 1 1 1 1 1 2 2 1 2 1 1 1 3 3 1 1 1 1 2 2 2 2 2 3 3 3
 [901] 1 3 2 2 3 3 3 1 3 3 1 1 1 2 2 1 1 1 1 1 2 2 2 2 1 1 1 3 3 3 1 1 3 3 3 1 1 1 1 3 3 3 1 3 3 3 3 2 1 3
 [951] 3 2 2 1 2 1 3 1 2 1 1 3 2 2 1 1 2 1 1 3 3 3 1 1 1 1 2 3 3 3 3 3 3 3 3 3 1 3 3 1 1 3 1 1 3 3 1 1 3 3
 [ reached getOption("max.print") -- omitted 1111 entries ]
>
```

Figure 3.12 : The output of the pre-implemented function fitted() to print all points' classes

Furthermore, by using the fviz_cluster() function from the factoextra package in R, we can obtain a visual representation of the cluster division of our dataset in 2D, where the x and y axes represent the two attributes with the highest variance in the dataset.

Figure 3.13 shows the command to execute to obtain this representation.

```
factoextra::fviz_cluster(object, data.norm)
```

Figure 3.13 : The fviz_cluster() command to get representation of cluster division on the data set

Below, on figure 3.14, you can see the result of the fviz_cluster() function.

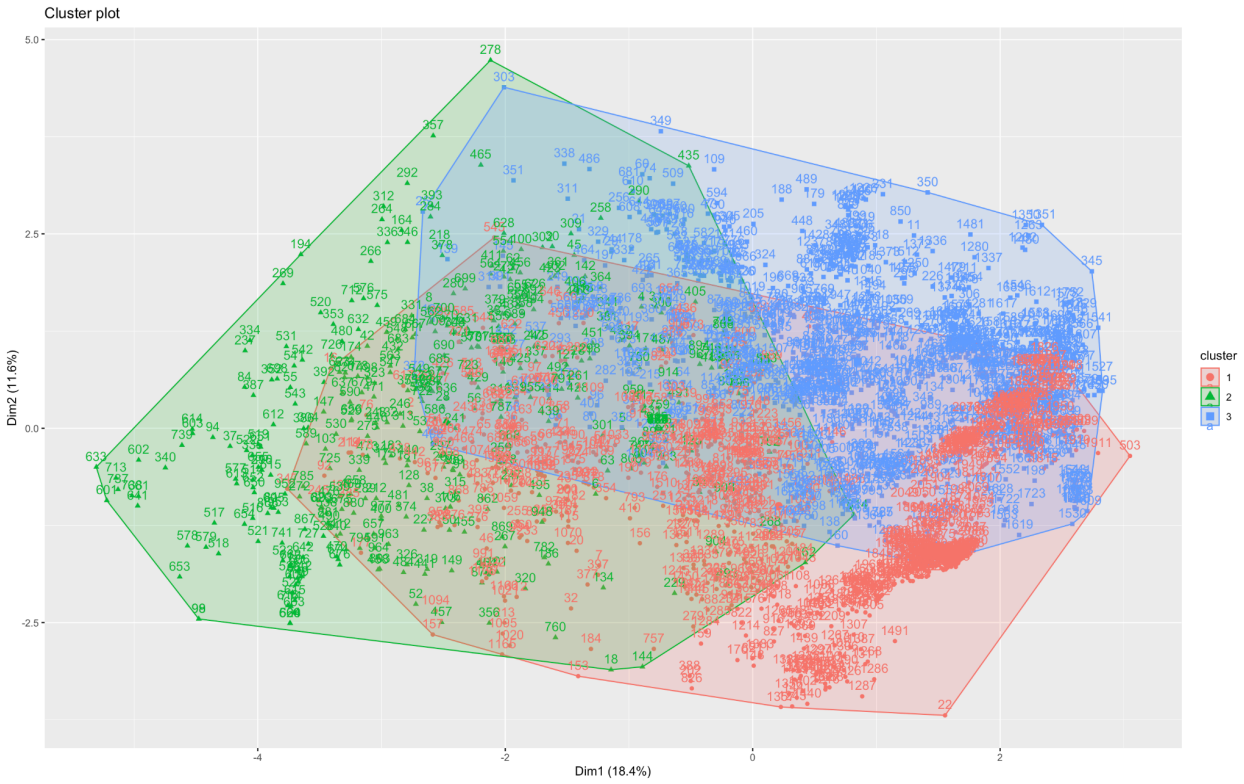


Figure 3.14 : The output of the command represented on the figure 3.13

The next figure (3.15) illustrates how to obtain the center points of each attribute for each cluster.

```
> object$centers
     Gender       Age    Height    Weight family_history_with_overweight      FAVC      FCVC       NCP
1 1.0000000 0.2320370 0.3837716 0.3888693                      1.000000 0.9001233 0.7988805 0.5533436
2 0.6057441 0.1594562 0.3810112 0.1478477                      0.000000 0.7415144 0.6876814 0.5219427
3 0.0000000 0.2332995 0.5946233 0.4118422                      0.997819 0.9291167 0.6396138 0.5861009
       CAEC      SMOKE      CH2O       SCC       FAF       TUE      CALC    MTRANS NObeyesdad
1 0.3711467 0.01726264 0.5133061 0.03699137 0.2678564 0.3167783 0.2412659 0.4180025  0.6502261
2 0.4369017 0.01566580 0.4065706 0.12793734 0.3709760 0.3131755 0.2558747 0.4588773  0.1623151
3 0.3645947 0.02617230 0.5364758 0.01853871 0.3834215 0.3462640 0.2410033 0.3699564  0.5512541
>
```

Figure 3.15 : The centers of the object returned by kmeans() function on the figure 3.9

As we can see, using only 3 clusters is likely not the best choice. To improve the results, let's implement the wssplot() function from the rubric. This function generates a visual graph to help determine the optimal number of clusters for a given dataset. It does so by iterating through different numbers of clusters (ranging from 1 to a specified maximum, which defaults to 15) and computing the within-cluster sum of squares error for each. By analyzing the resulting plot, we can identify the point where adding more clusters results in diminishing improvements.

The source code for the implementation of this function and command to run to obtain this visual representation are shown in Figure 3.16.

```
wssplot <- function(data, nc = 15, seed = 1234) {
  wss <- numeric(nc)
  wss[1] <- (nrow(data) - 1) * sum(apply(data, 2, var))

  for (i in 2:nc) {
    set.seed(seed)
    kmeans_result <- kmeans(data, centers = i)
    wss[i] <- kmeans_result$tot.withinss
  }

  plot(1:nc, wss, type = "b",
       xlab = "Number of clusters",
       ylab = "Within groups sum of squares")
}

wssplot(data.norm, nc = 15)
```

Figure 3.16 : wssplot() function implementation and the command to run on out data set

Below, in Figure 3.17, we can see the graph itself. The plot shows that the curvature of the line starts to flatten around 10 clusters, suggesting that this number would be a better choice for cluster separation.

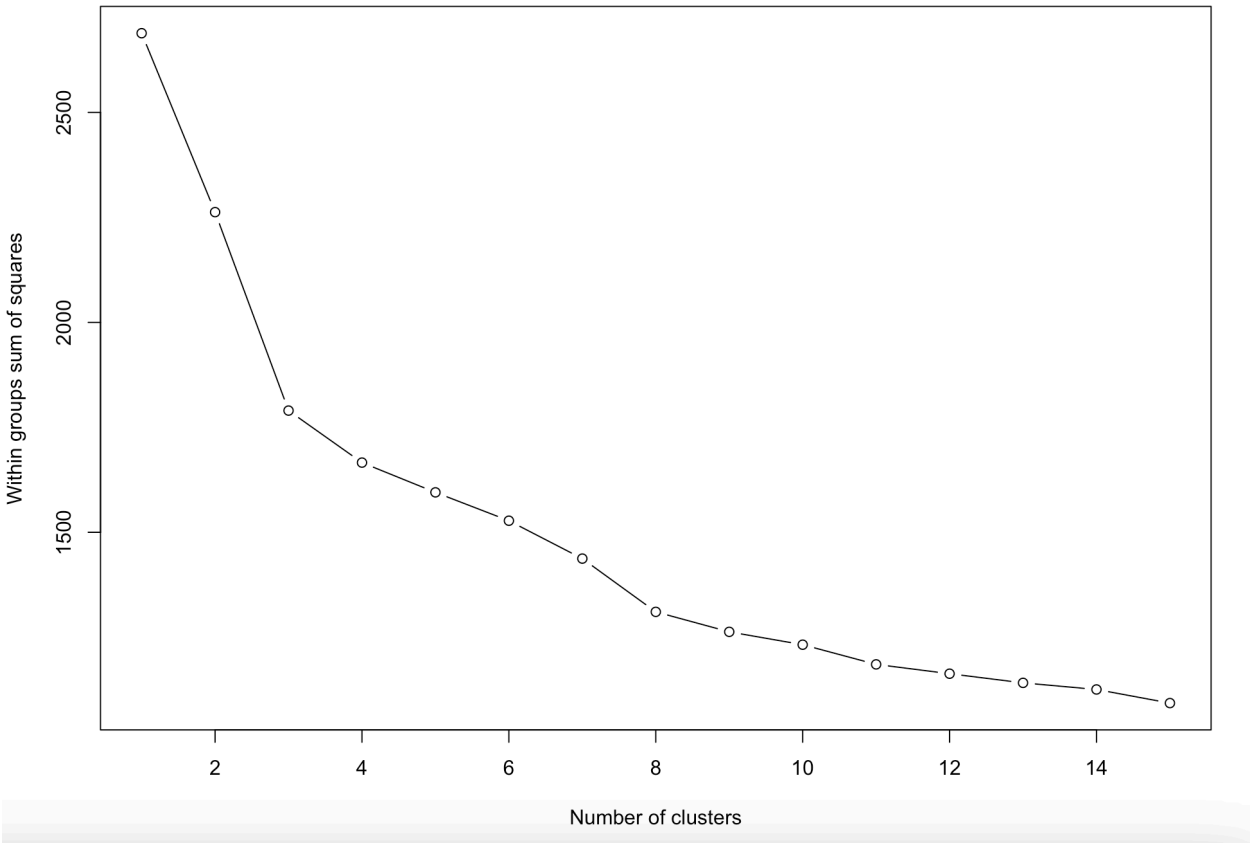


Figure 3.17 : The result of the wssplot() function from figure 3.16

The factoextra package also has a pre-implemented function called fviz_nbclust(), which helps determine the optimal number of clusters using visual representation. This function supports different methods, such as the Elbow method, Silhouette method, and Gap statistic, to help us find the optimal cluster number based on different characteristics. Here, we use the WSS method, just as before, but with some pre-built adjustments.

On figure 3.18 you can find the command to run to get this second visual representation and on figure 3.19 the result of the execution of this command.

```
> factoextra::fviz_nbclust(data.norm, FUNcluster = kmeans, method = "wss", k.max = 20, verbose = TRUE)
>
```

Figure 3.18 : The command to obtain the optimal number of clusters for the data set

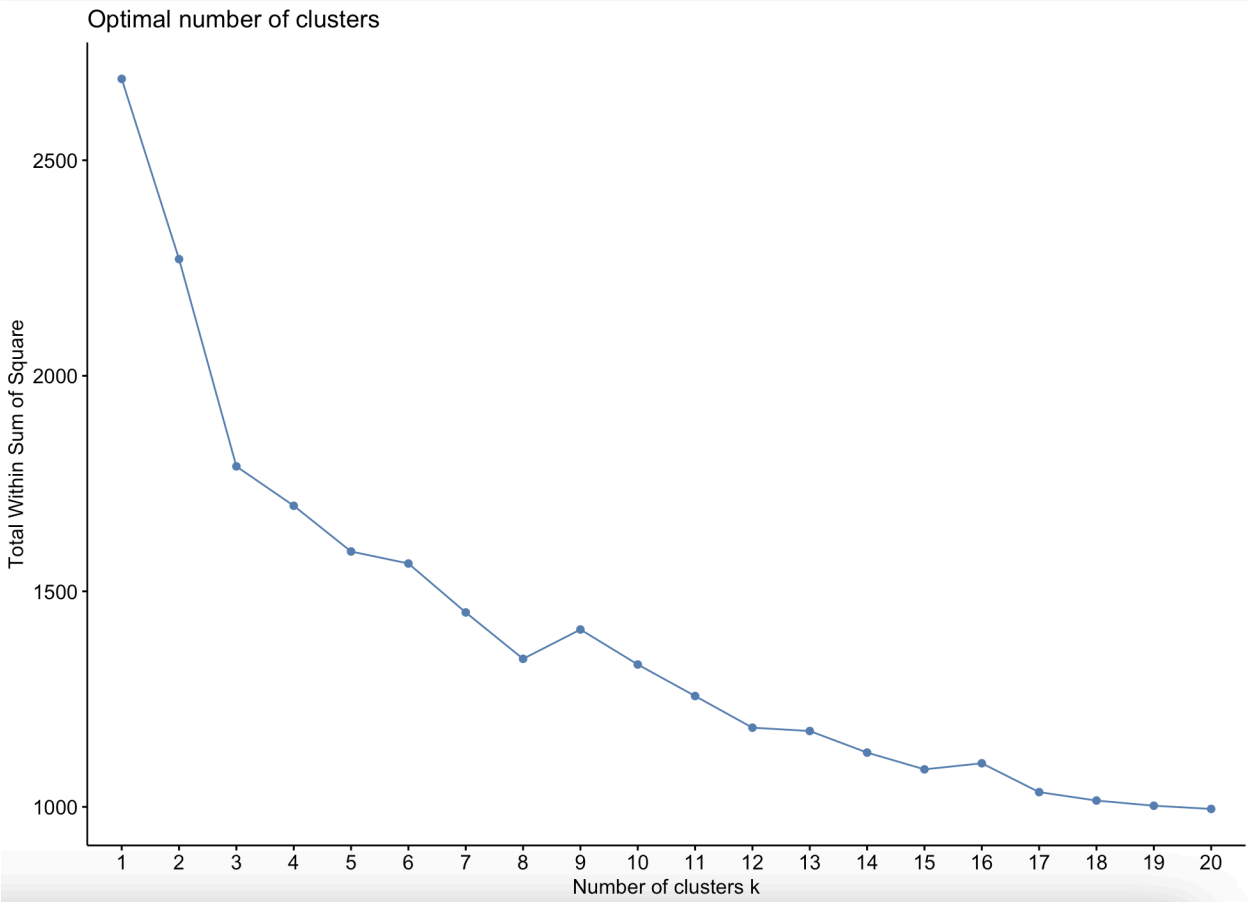


Figure 3.19 : The result of the command on figure 3.18

From this graph (figure 3.19), it also seems that 10 could be the most optimal number of clusters. Therefore, let's run the kmeans() function again, this time specifying 10 centers. The screenshots

below (Figures 3.20.1 and 3.20.2) illustrate the updated function call and the resulting k-means clustering object.

```
> object2 <- kmeans(data.norm, centers = 10, iter.max = 15, nstart = 30)
> object2
K-means clustering with 10 clusters of sizes 233, 329, 170, 146, 153, 410, 168, 256, 184, 62

Cluster means:
      Gender       Age    Height     Weight family_history_with_overweight      FAVC      FCVC       NCP
1  1.0000000 0.3391441 0.3324931 0.26360584                              1 1.0000000 0.6566634 0.4689398
2  1.0000000 0.2011649 0.4467744 0.60472807                              1 1.0000000 0.9973742 0.6666667
3  1.0000000 0.1428219 0.2925517 0.09536840                              0 1.0000000 0.6831831 0.4981756
4  0.5547945 0.2192036 0.3805499 0.25239559                              1 0.0000000 0.6878937 0.5602793
5  0.0000000 0.1834147 0.5106476 0.23468220                              0 0.7581699 0.6203726 0.5389641
6  0.0000000 0.2961870 0.6108401 0.49836396                              1 1.0000000 0.6257283 0.5719151
7  1.0000000 0.1499193 0.3936519 0.23188491                              1 1.0000000 0.6317362 0.4649484
8  0.0000000 0.1986943 0.5608128 0.40100540                              1 1.0000000 0.6154282 0.4978945
9  0.0000000 0.1448655 0.6249124 0.26580632                              1 1.0000000 0.7153093 0.7305885
10 1.0000000 0.1531531 0.3132731 0.08809493                              0 0.0000000 0.8600624 0.5497747
       CAEC       SMOKE      CH2O         SCC       FAF       TUE      CALC    MTRANS NObeyesdad
1  0.3662375 0.030042918 0.4264704 0.017167382 0.2460752 0.0905097 0.2031474 0.2339056 0.47424893
2  0.3343465 0.003039514 0.6038044 0.000000000 0.2248376 0.3024759 0.3333333 0.5000000 0.99189463
3  0.4352941 0.000000000 0.2862203 0.141176471 0.3169256 0.4054096 0.2803922 0.4529412 0.11666667
4  0.3858447 0.061643836 0.5180932 0.109589041 0.4163965 0.3000362 0.1986301 0.4554795 0.39611872
5  0.3747277 0.032679739 0.5407375 0.058823529 0.4278483 0.2635578 0.2549020 0.4362745 0.25381264
6  0.3495935 0.029268293 0.4829849 0.009756098 0.3448983 0.1212225 0.2691057 0.3213415 0.70284553
7  0.4444444 0.011904762 0.4899504 0.089285714 0.3227205 0.6667892 0.1488095 0.5000000 0.35615079
8  0.3736979 0.023437500 0.4861211 0.003906250 0.2092999 0.6610887 0.2460938 0.4355469 0.54687500
9  0.3768116 0.005434783 0.6972619 0.038043478 0.6894307 0.4242509 0.1702899 0.3532609 0.26539855
10 0.5913978 0.016129032 0.4165523 0.258064516 0.3776184 0.1887458 0.1989247 0.5241935 0.07795699
```

Figure 3.20.1 : The kmeans() function applied to the data set with better number of clusters (Part 1)

```
Clustering vector:
   [1]  4  4  4  5  5  5  1  5  9  9  9  7  5  5  8  4  6  3  1  4  4  1  1  7  8  4  8  3  9  5  5  4 10  5
  [35]  5  4  3  3  8  1  1  5  1  8  5  4  9  9  4  3  7  3  3  4 10  5  8  4  7  9  7  5  5  8  7  7  8  8
  [69]  6  4  7  4  4  4  8  4  4  1  7  4 10  6  8 10  8  4  9  6  1  1 10  4  4 10  3  4  4 10 10  5  3  4
 [103]  3  1  1  3  3  8  4  7  8  1  1  4  1  7  7  4  8  4  7  8  4  6  5  6  5  3  6  5  7  5  7  3  6  6
 [137]  8  4  4  5  9  5  5  3  4  5 10  4  3  1  8  1  1  4  6  1  4  9  4  6  4  5  1  5  4  4  4  4  1  1
 [171]  5  9 10  5  5  1  1  9  9  5  3  1  3  1  5 10  4  4  9  7  5  6  4  5  2  8  4  6  7  7  4  1  2  4
 [205]  9  1  6  4  7  7  8  3  4  9  9  8  4  5  4  1  4  8  7  1  1  4  3  3 10  4  9  1  4  4  1  4  3  7
 [239]  3  7  3  5  7  6  4 10  3  3  9  9  7  4  4  7  4  4  4  5  5  7  5  4  5  5  9  5  3  5  5  8  5  3
 [273]  5  6 10  7  9  5  1  5  5  6  7  5  8  6  3  1  3  5  9  5  8  4  8 10  3  5  8  5  5  5  9  3  4  6
 [307]  7  7  5 10  9  5  3  8  3  5  4  5  3  3  8  4 10  9  1  3  9  4  4  2  3  9  6 10  3  5  5  9  3 10
 [341]  5  5  6  5  6  5  4  8  9  8  4 10  3  1  3  3  5  7  4  4  5  1  4  5  8  5  6  3  1  5  5  8  3  3
 [375]  9  6  4  5  5  8  5  9  4  4  7  3 10  1  4  1  8 10  5  5  7  7  4  5  6  5  4  5  8  4  5  5  5  7
 [409]  4  7  5 10  5  5  8  9  5  8  5  6  9  9  5  8  5  7  5  3  5  6  5  3  5  7  5  7  3  7  3  3  3  7
 [443]  8  8  9  3  9  9  7  7  5  4  5  3  3  5  3  3  5  8  7  8  6  4  5  7  6  6  3  3  3  1 10  1  5 10
 [477]  1  7  5  3  3  9  7  3  7  9  5  4  9  3  5  5  5  5  3  6  8  5  2  2  2  2  2  2  2  2  9  9  9
 [511]  7  7  7 10 10 10 10 10 10  3  3  3  3  3  3  3  3 10 10 10  9  9  9  9  9  3  3  3 10 10 10  7
 [545]  7  7  3  3  3  1  1  1  5  5  5  5  5  5  9  9  9  3  3  3  9  9  9  7  7  1  3  3  3  3  3  3 10 10
 [579] 10  9  9  9  1  7  7  3  3  3  3  3  3  9  9  9  9  9  9  8  8  8 10 10 10  3  3  3  8  9  8  9  7 10
 [613] 10 10  3  3  4  9  7  3 10  7  3  1  5  5  9  3  9  7  3  3 10  9  7  3  3  9  9  8 10  3  8  9  9  7
 [647]  7  7  7 10 10 10 10 10 10  3  3  3  3  3 10  3  3  3  7  1  3  9  9  9  9  7  9  3  3  3  3 10 10  7
 [681]  9  7  3  3  3  7  1  1  5  3  5  5  9  9  9  9  9  3  3  3  9  9  9  7  9  1  3  3  3  3  3  3 10  3
 [715]  3  9  9  9  1  7  7  3  3  3  3  3  3  9  9  9  9  9  9  8  8  8 10 10 10  3  3  3  8  9  8  5  5  5
 [749]  6  5  6  3  8  6  7  8  4  7  5  3  1  1  5  5  5  1  9  9  9  9  9  7  9  1  1  9  6  1  8  8  6  3
 [783]  1  8  3  7  3  4  8  1  5  7  1  3  7  5  1  6  9  7  7  7  3  5  5  5  5  5  6  1  7  6  6  1  8  8
 [817]  8  8  8  6  1  1  7  7  5  1  1  1  1  1  5  5  5  5  5  1  1  6  6  9  9  9  9  9  1  1  7  7  9  9
 [851]  9  1  1  7  6  6  1  8  6  8  9  3  1  1  8  3  3  7  3  3  4  8  8  1  7  1  7  7  1  3  3  1  5  1
 [885]  7  7  9  9  7  7  7  1  3  5  5  5  5  6  6  6  1  9  3  3  8  9  6  7  8  8  1  1  7  5  5  1  1  1
 [919]  1  1  5  5  5  5  1  1  7  9  9  9  1  1  8  8  9  1  1  7  1  9  6  6  1  8  8  8  9  3  1  8  8  3
 [953]  3  7  3  1  8  1  5  7  7  6  3  3  7  7  5  1  1  6  9  9  7  7  7  7  3  8  6  8  9  6  6  8  9  8
 [987]  4  8  6  4  4  9  4  4  8  8  7  7  8  8
 [ reached getOption("max.print") -- omitted 1111 entries ]

Within cluster sum of squares by cluster:
 [1] 133.07975  69.68336 129.25732 161.49217 147.77624 180.09861 118.70498 127.61068 102.34070  47.52246
 (between_SS / total_SS =  54.7 %)

Available components:

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

Figure 3.20.2 : The kmeans() function applied to the data set with better number of clusters (Part 2)

As shown in Figure 3.20.2, the total score has now improved to 54.7%, which is better than before. Additionally, we can visualize the new cluster distribution by running the fviz_cluster() function. The source code for this command is shown in figure 3.21, and its resulting visualization can be seen in figure 3.22.

```
> factoextra::fviz_cluster(object2, data.norm)
>
```

Figure 3.21 : The fviz_cluster() command line to obtain visual representation of new cluster separation

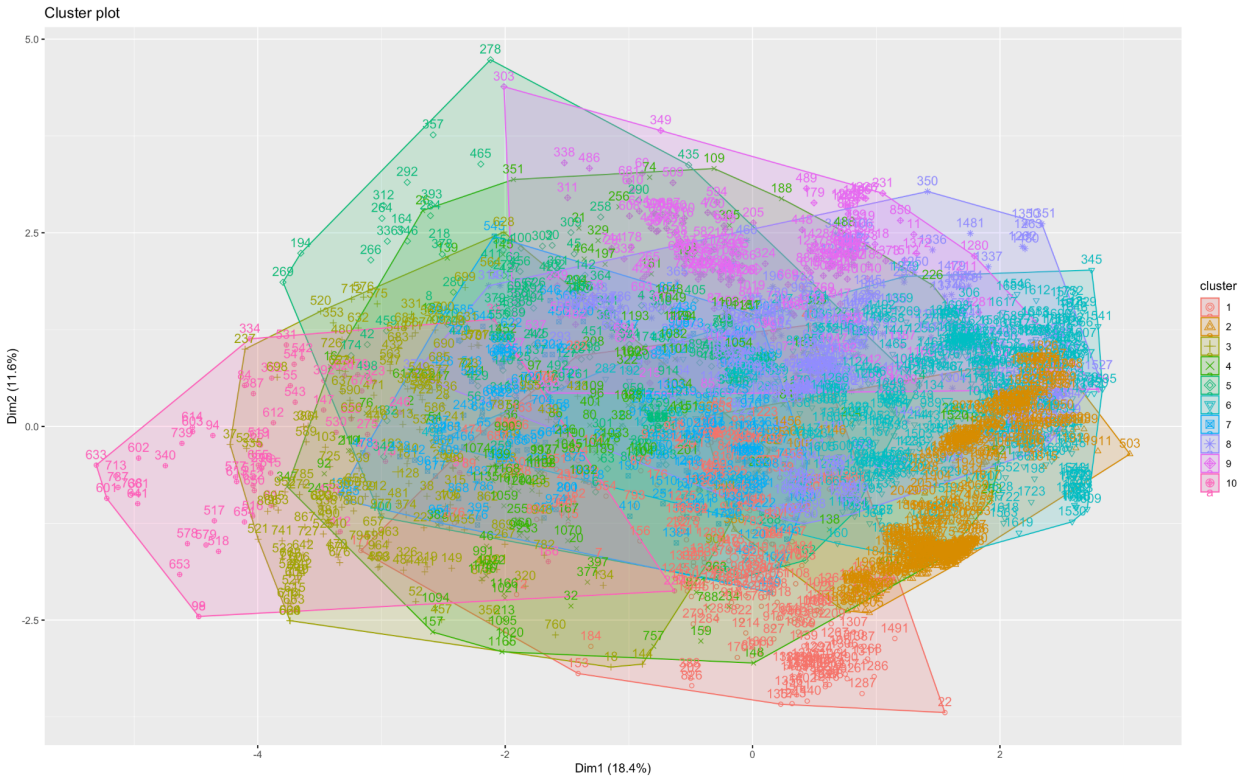


Figure 3.22 : The result of the command line on the figure 3.21

Question 4 : Prediction

Let's take a subset of the attributes that are most correlated with obesity level, as identified in questions 1 and 2. Then, using the function wssplot() we defined earlier, we will determine the optimal number of clusters. Figure 4.1 shows the source code for this process.

```
> wssplot(subset_data.norm, nc = 15)
>
```

Figure 4.1 : wssplot() on the subset : source code

Figure 4.2 illustrates the result produced. From this graph, we can see that starting from 6, the line becomes flat, indicating that 6 is the optimal number of clusters.

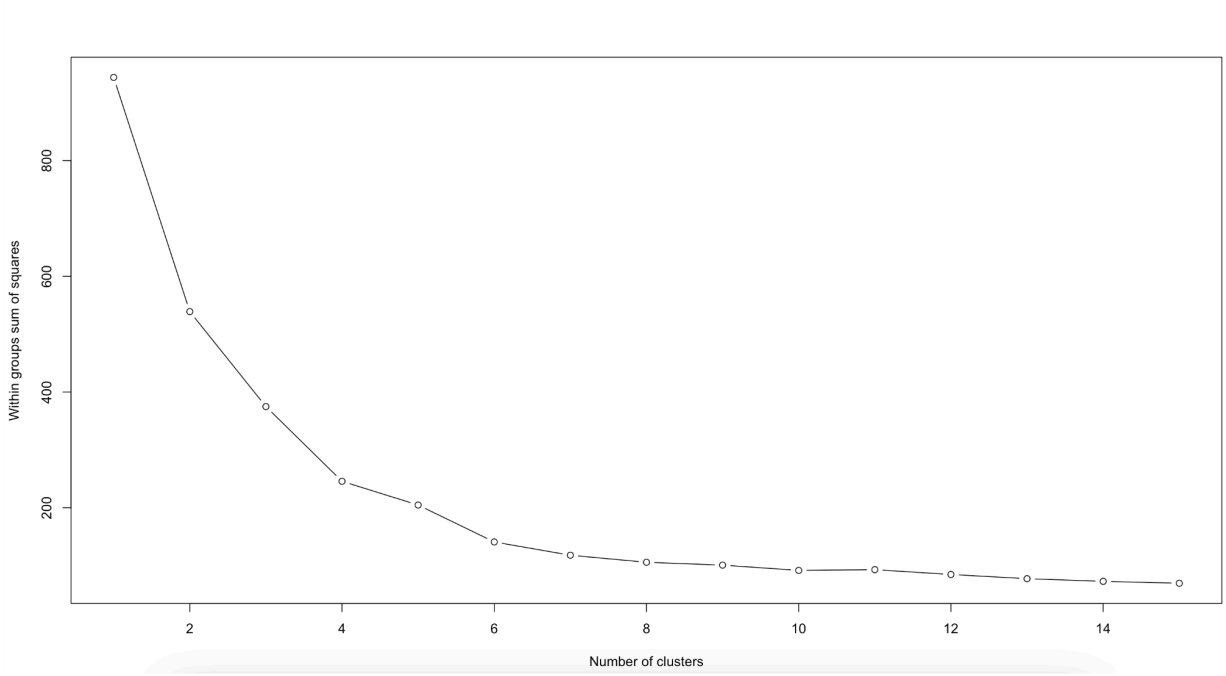


Figure 4.2 : wssplot() on the subset : result

Furthermore, let's perform a generalized linear model (GLM) analysis on our subset. Here, the response variable is obesity level, while the explanatory variables are weight, family history of overweight, age, FAVC, and CALC. We use a normal distribution for linear regression and apply the model to the first training dataset (split 70-30%).

Figure 4.3 shows the command to run the glm() function and the summary of the returned object.

```
> subset_data.train1.glm = glm(NObeyesdad ~ Weight + family_history_with_overweight + Age + FAVC + CALC,
+                              family = gaussian, data = subset_data.train1)
>
> summary(subset_data.train1.glm)

Call:
glm(formula = NObeyesdad ~ Weight + family_history_with_overweight +
    Age + FAVC + CALC, family = gaussian, data = subset_data.train1)

Coefficients:
                               Estimate Std. Error t value Pr(>|t|)
(Intercept)                   -0.07587    0.01237  -6.133 1.11e-09 ***
Weight                         1.49114    0.02070  72.037  < 2e-16 ***
family_history_with_overweight  0.04014    0.01007   3.987 7.03e-05 ***
Age                            0.27199    0.02592  10.492  < 2e-16 ***
FAVC                          -0.01933    0.01095  -1.765   0.0777 .
CALC                          -0.05069    0.02028  -2.500   0.0125 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.01643602)

    Null deviance: 163.910  on 1476  degrees of freedom
Residual deviance:  24.177  on 1471  degrees of freedom
AIC: -1868.4

Number of Fisher Scoring iterations: 2

>
```

Figure 4.3 : glm() on the subset

Furthermore, by running the plot() function on the object returned by the glm() function (Figure 4.4), we obtain diagnostic plots that help analyze model performance and detect potential issues.

- Residuals vs. Fitted (Figure 4.5.1): This plot checks for non-linearity and unequal variance in residuals. In the best case scenario, residuals should be randomly scattered around zero without any clear pattern. But here the red line showing the residuals movement, at the beginning is in fact around the zero value, but then starts to decline from it.
- Q-Q Plot (Figure 4.5.2): This plot assesses if the residuals follow a normal distribution. If residual points align with the diagonal line, it indicates that residuals are normally

distributed. In our case, we can see that first values are almost perfectly following the line, but at the end these tend to be lower down the line, which suggests non-normality.

- Scale-Location Plot (Figure 4.5.3): This graph helps check homoscedasticity (constant variance of residuals). Ideally, the red trend line should be flat. In our case, we can see some curves, which indicates slight heteroscedasticity, and means that the model's variance changes across fitted values a little bit.
- Residuals vs. Leverage (Figure 4.5.4): This plot helps identify influential data points that may affect the model too much.

```
> plot(subset_data.train1.glm)
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
Hit <Return> to see next plot:
>
```

Figure 4.4 : Plot of glm() Result

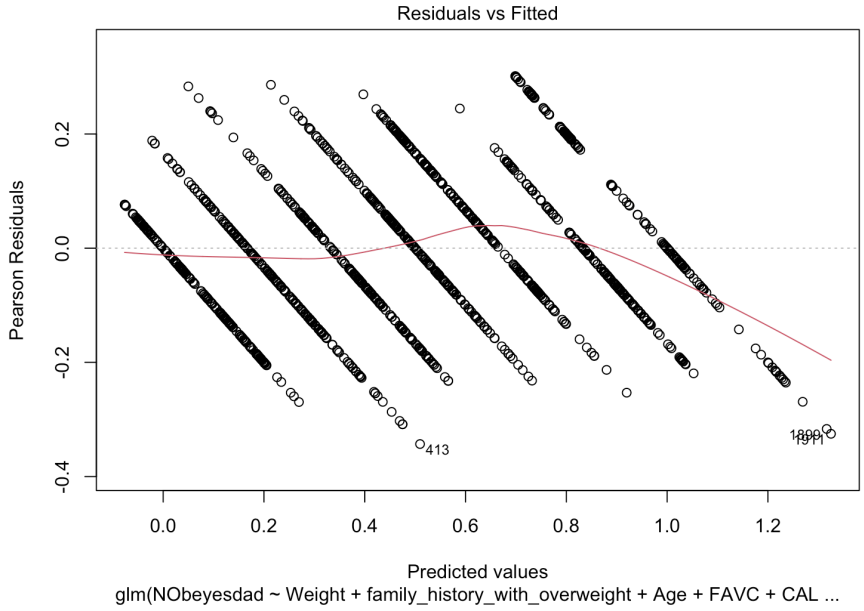


Figure 4.5.1 : Residuals vs Fitted graph

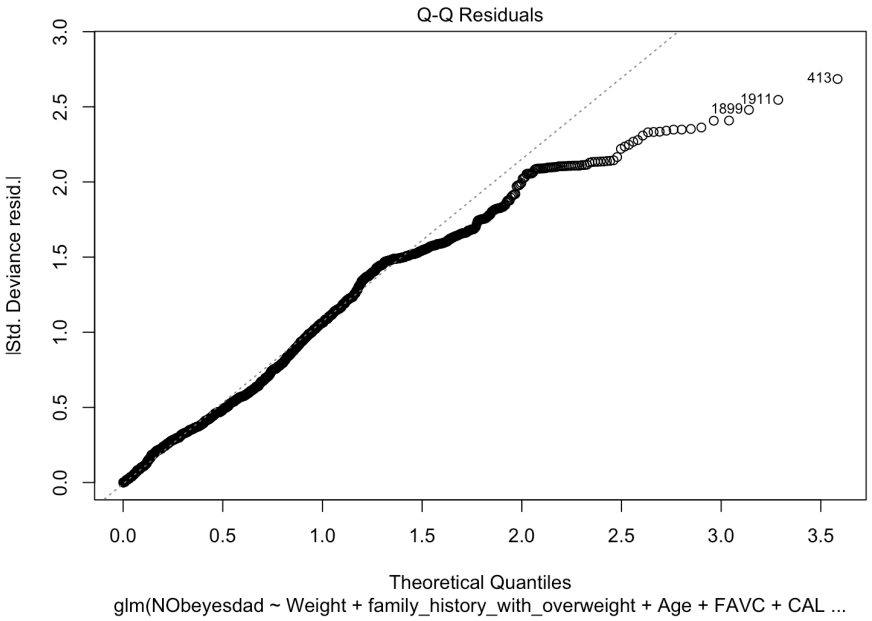


Figure 4.5.2 : Q-Q Residuals graph

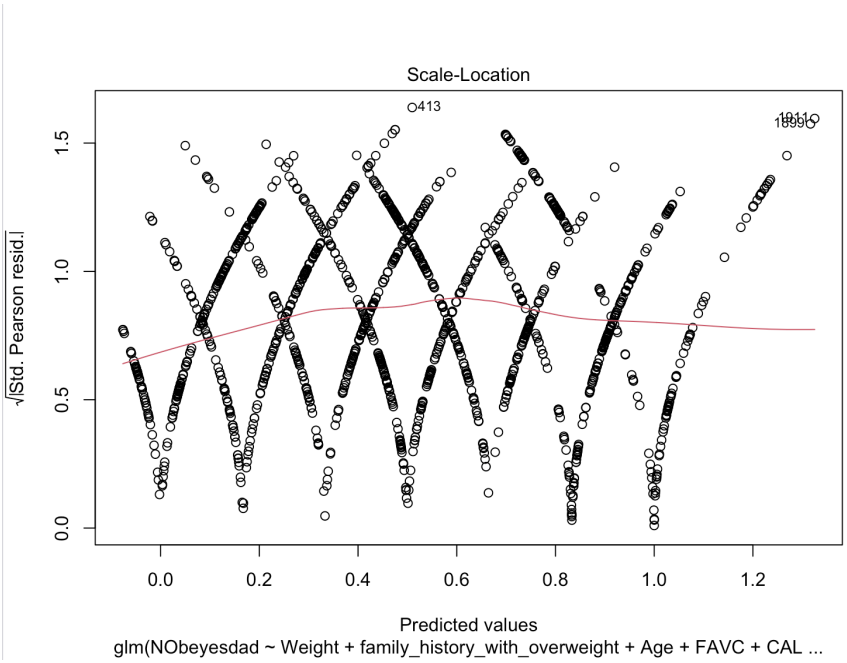


Figure 4.5.3 : Scale-Location graph

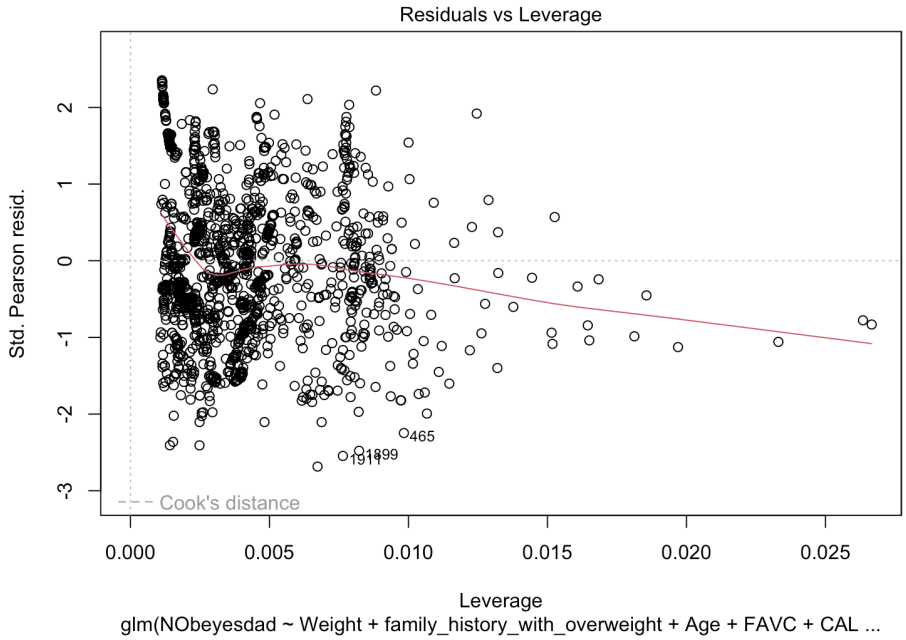


Figure 4.5.4 : Residuals vs Leverage graph

The next screenshot (Figure 4.6) displays the confidence intervals for each explanatory variable.

```
> confint(subset_data.train1.glm)
Waiting for profiling to be done...
                                    2.5 %       97.5 %
(Intercept)                   -0.10011281 -0.051619320
Weight                         1.45057280  1.531714594
family_history_with_overweight  0.02040912  0.059880259
Age                            0.22118017  0.322792596
FAVC                          -0.04080156  0.002134069
CALC                          -0.09043371 -0.010954070
>
```

Figure 4.6 : Source code to obtain confidence interval and the result

The next screenshot (Figure 4.7) shows the anova() command applied to the previously obtained GLM result and its output. This function performs a Chi-square test to evaluate the significance of each predictor in the model.

```
> subset_data.train1.glm.anova = anova(subset_data.train1.glm, test = "Chisq")
> subset_data.train1.glm.anova
Analysis of Deviance Table

Model: gaussian, link: identity

Response: NObeyesdad

Terms added sequentially (first to last)


                             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                                          1476    163.910
Weight                        1  137.299      1475     26.612 < 2.2e-16 ***
family_history_with_overweight  1    0.492      1474     26.120  4.53e-08 ***
Age                           1    1.780      1473     24.340 < 2.2e-16 ***
FAVC                          1    0.060      1472     24.280   0.05586 .
CALC                          1    0.103      1471     24.177   0.01241 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

Figure 4.7 : anova() function and the result

Next, the screenshot below (figure 4.8) shows the predict() function applied to the training and testing data sets to make predictions for the dependent variable based on the training set.

```
> subset_data.test1.pred = predict(subset_data.train1.glm, newdata = subset_data.test1)
> subset_data.test1.pred
          1           3           4           7           8           9
0.282985684 0.405426841 0.499709311 0.158176422 0.109322659 0.247216846
         12          18          22          24          31          37
0.424800985 0.425593938 0.698989787 0.452843805 0.481233628 0.062375125
         44          46          47          56          69          70
0.207600137 0.294113622 0.335777481 0.215261045 0.816079538 0.233149857
         73          74          76          81          85          89
0.255834764 0.265641770 0.083128750 0.164499479 0.481332576 0.280566822
         90          93          98          99         101         104
0.431942700 0.668504624 -0.018871604 -0.018871604 0.195481350 0.185773292
        105         108         111         112         114         121
0.523179038 0.541867310 0.369161295 0.202671255 0.121853557 0.397633146
        122         126         130         134         139         140
0.586395981 0.390542189 0.380224124 0.443444973 0.463262445 0.291166781
        143         144         146         147         149         150
0.529559386 0.404246883 0.273376917 0.081742708 0.207947140 0.599736753
        152         154         157         158         167         172
0.243196785 0.580774720 0.152572081 0.202242224 0.401406779 0.162625515
        173         179         185         186         190         193
0.181877231 0.481349496 0.273376917 0.328324228 0.310839395 0.493987476
        195         198         211         212         217         218
0.486656651 0.963401516 0.814261897 0.078844057 0.266533671 0.331452388
        219         230         232         234         236         238
0.041509428 0.969806601 0.269009852 0.485981191 0.338179424 0.129670732
        240         244         248         253         255         257
0.252078051 0.705675139 0.084185050 0.741059197 0.199749124 0.482413587
        260         261         262         264         265         266
0.529847471 0.345914570 0.406747773 0.112673821 0.262776958 0.033457323
        270         271         272         273         274         285
0.280583741 0.696684513 0.039244267 0.161634666 0.347797320 0.307734660
```

Figure 4.8 : predict() function on the data set

```
> summary(subset_data.test1.pred)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.07341  0.29518  0.50593  0.53491  0.78642  1.44234
>
```

Figure 4.9 : The summary of the object returned by the predict() function

Then, in the next screenshot, we create two clusterings: the first one is based on predicted values (subset_data.test1.pred.k6), and the second one is based on the original test dataset (subset_data.test1.kmeans.k6). After that, we generate a cross table to evaluate the accuracy of the predictions. The source code and the results are shown below in Figures 4.10.1 and 4.10.2. From this cross table, we can see some interesting insights about the predictions. For instance, the model correctly predicted the members of cluster 1 only 2 times, with the rest of the points being classified into the wrong clusters. On the other hand, the best predictions were for cluster 5, where 46 points were classified correctly.

```
> subset_data.test1.pred.k6 = kmeans(subset_data.test1.pred, centers = 6)
> subset_data.test1.kmeans.k6 = kmeans(subset_data.test1, centers = 6)
>
> subset_data.test1.ct.k6 = CrossTable(subset_data.test1.pred.k6$cluster,
+                                      subset_data.test1.kmeans.k6$cluster,
+                                      prop.chisq = TRUE)


   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  634
```

Figure 4.10.1 : CrossTable() function to compare predictions with true values (Part 1)

	subset_data.test1.kmeans.k6$cluster						
subset_data.test1.pred.k6$cluster	1	2	3	4	5	6	Row Total
1	2	0	98	0	0	12	112
	4.785	5.300	145.575	19.785	13.956	14.009	
	0.018	0.000	0.875	0.000	0.000	0.107	0.177
	0.043	0.000	0.560	0.000	0.000	0.063	
	0.003	0.000	0.155	0.000	0.000	0.019	
2	1	0	50	0	0	0	51
	2.045	2.413	91.668	9.009	6.355	15.364	
	0.020	0.000	0.980	0.000	0.000	0.000	0.080
	0.021	0.000	0.286	0.000	0.000	0.000	
	0.002	0.000	0.079	0.000	0.000	0.000	
3	2	0	27	0	1	72	102
	4.091	4.826	0.047	18.019	10.788	55.431	
	0.020	0.000	0.265	0.000	0.010	0.706	0.161
	0.043	0.000	0.154	0.000	0.013	0.377	
	0.003	0.000	0.043	0.000	0.002	0.114	
4	18	3	0	29	8	107	165
	2.720	2.960	45.544	0.001	7.673	66.032	
	0.109	0.018	0.000	0.176	0.048	0.648	0.260
	0.383	0.100	0.000	0.259	0.101	0.560	
	0.028	0.005	0.000	0.046	0.013	0.169	
5	5	17	0	30	46	0	98
	0.706	32.959	27.050	9.298	93.493	29.524	
	0.051	0.173	0.000	0.306	0.469	0.000	0.155
	0.106	0.567	0.000	0.268	0.582	0.000	
	0.008	0.027	0.000	0.047	0.073	0.000	
6	19	10	0	53	24	0	106
	15.798	4.953	29.259	62.734	8.817	31.934	
	0.179	0.094	0.000	0.500	0.226	0.000	0.167
	0.404	0.333	0.000	0.473	0.304	0.000	
	0.030	0.016	0.000	0.084	0.038	0.000	
Column Total	47	30	175	112	79	191	634
	0.074	0.047	0.276	0.177	0.125	0.301	

Figure 4.10.2 : CrossTable() function to compare predictions with true values (Part 2)

But for convenience, let's calculate all performance metrics (True Positive: TP, False Positive: FP, etc.) to better assess the accuracy of the predictions. First, we need to extract the confusion matrix itself. Below (Figure 4.11), you can find the command to obtain the confusion matrix from the object returned by the previously executed CrossTable() function.

```
confusion_matrix1 <- subset_data.test1.ct.k6$t
```

Figure 4.11 : Getting confusion matrix from the object returned by CrossTable() function

Next, we can define a function that calculates and prints all performance metrics based on the given confusion matrix. The source code for this function is shown below in Figure 4.12.

```r
print_measurements <- function(confusion_matrix) {
  matrix_values <- as.matrix(confusion_matrix)

  num_clusters <- nrow(matrix_values)

  TP <- numeric(num_clusters)
  FP <- numeric(num_clusters)
  FN <- numeric(num_clusters)
  TN <- numeric(num_clusters)

  for (i in 1:num_clusters) {
    TP[i] <- matrix_values[i, i]
    FP[i] <- sum(matrix_values[, i]) - TP[i]
    FN[i] <- sum(matrix_values[i, ]) - TP[i]
    TN[i] <- sum(matrix_values) - TP[i] - FP[i] - FN[i]
  }

  accuracy <- (TP + TN) / (TP + FP + FN + TN)
  error <- (FP + FN) / (TP + FP + FN + TN)
  sensivity <- TP / (TP + FN)
  specificity <- TN / (FP + TN)
  precision <- TP / (TP + FP)
  f_measure <- 2 * (precision * sensivity) / (precision + sensivity)

  cat("False Positive: ", FP, "\n")
  cat("False Negative: ", FN, "\n")
  cat("True Positive: ", TP, "\n")
  cat("True Negative: ", TN, "\n")

  cat("Accuracy: ", accuracy, "\n")
  cat("Error: ", error, "\n")
  cat("Sensivity: ", sensivity, "\n")
  cat("Specificity: ", specificity, "\n")
  cat("Precision: ", precision, "\n")
  cat("F - measure: ", f_measure, "\n")
}
```

Figure 4.12 : Implementation of the function printing all needed measurements of prediction

Finally, in Figure 4.13, we can see the print_measurements() function applied to the confusion matrix of our results and its output.

```
> print_measurements(confusion_matrix1)
False Positive:  45 30 148 83 33 191
False Negative:  110 51 75 136 52 106
True Positive:  2 0 27 29 46 0
True Negative:  477 553 384 386 503 337
Accuracy:  0.7555205 0.8722397 0.648265 0.6545741 0.8659306 0.5315457
Error:  0.2444795 0.1277603 0.351735 0.3454259 0.1340694 0.4684543
Sensivity:  0.01785714 0 0.2647059 0.1757576 0.4693878 0
Specificity:  0.9137931 0.948542 0.7218045 0.8230277 0.9384328 0.6382576
Precision:  0.04255319 0 0.1542857 0.2589286 0.5822785 0
F - measure:  0.02515723 NaN 0.1949458 0.2093863 0.519774 NaN
>
```

Figure 4.13 : Performance measurements of the first prediction results

Here, we have such measurements as :

- False Positive (FP): Points predicted to be in a cluster but in reality belong to a different cluster.
- False Negative (FN): Points that belong to a cluster but are predicted to be in a different cluster
- True Positive (TP): Points that belong to the correct cluster and are predicted to be in that same cluster.
- True Negative (TN): Points that do not belong to the cluster and are correctly predicted as not belonging to it.
- Accuracy: Proportion of correct predictions.
- Error: Proportion of incorrect predictions.
- Sensitivity (Recall): Proportion of actual positives correctly identified.
- Specificity: Proportion of points that do not belong to a cluster and are correctly predicted as not belonging to it.

- Precision: Proportion of predicted positives that are correct.
- F-Measure: Harmonic mean of precision and recall.

From our results, we can glean that, for example, the accuracy is close to one in some clusters, meaning the model predicts well at times. This is also reflected in the error, which is not far from 0. However, precision and sensitivity are quite low, indicating that the model struggles to correctly identify the points' correct clusters (for clusters 1 and 6, these values are 0). On the other hand, specificity is often close to 1, suggesting that the model performs well when predicting that a point does not belong to a specific cluster.

Conclusion