FINAL PROJECT REPORT

TOPIC: DATA ANALYSIS - IMPACT OF LIFESTYLE ON OBESITY

HA HAI VU

1. Introduction

1.1. Data Description:

As we know, a dramatic increase in obesity rates has occurred in the last few decades and becoming a public health concern in over the world. Some studies from the WHO indicate that obesity rates have increased every decade in all age groups and in both genders. To have better understanding of what aspects of an individual's daily life can they focus on to maintain or change their obesity level, I used the "Estimation of obesity levels UCI Dataset" from Kaggle for this project. This dataset includes data for the estimation of obesity levels in individuals from three Latin America countries which are Mexico, Peru and Colombia, based on their eating habits and physical condition. 77% of the data was generated synthetically using the Weka tool and the SMOTE filter, while 23% of the data was collected directly from users through a web platform. Source: https://www.kaggle.com/datasets/jayitabhattacharyya/estimation-of-obesity-levels-uci-dataset

The dataset contains 17 attributes and 2111 records, the records are labeled with the class variable NObeyesdad (Obesity Level), which 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. The independent attributes include 13 categorical variables and 3 numeric variables. The table below shows the attributes, data type and category description used for the analysis:

No.	Attribute	Data type	Categories
1	Gender	Categorical	Male; Female
2	Age	Numeric	
3	Height	Numeric	
4	Weight	Numeric	
5	family_history_with_overweight	Categorical	Yes/No
6	FAVC (eating high calorie foods frequently)	Categorical	Yes/No
7	FCVC (eating vegetables in meals)	Categorical	1 - Never 2 - Sometimes 3 - Always
8	NCP (number of main meals daily)	Numeric	
9	CAEC (eating food between meals)	Categorical	No Sometimes Frequently Always
10	Smoke	Categorical	Yes/No
11	CH2O (water intake per day in liters)	Numeric	
12	SCC (monitor calories on a daily basis)	Categorical	Yes/No

13	FAF (physical activity in number of days)	Numeric	0 – "I do not have" 1 – "1 or 2 days" 2 – "2 or 4 days" 3 – 4 or 5 days"
14	TUE (time spent on technology)	Numeric	0 – "0 – 2 hours" 1 – "3 – 5 hours" 2 – "More than 5 hours"
15	CALC (alcohol intake)	Categorical	I do not drink Sometimes Frequently Always
16	MTRANS (means of transportation)	Categorical	Automobile Motor Bike Bike Public-Transportation Walking
17	NObeyesdad (Obesity Level)	Categorical	Insufficient_Weight Normal_Weight Overweight_Level_I Overweight_Level_II Obesity_Type_I Obesity_Type_II Obesity_Type_III

1.2. Purpose of Analysis

This project analyzes a dataset containing obesity levels among adults from three countries in Latin America to determine which specific eating habits or daily activities most affect the classification of obesity levels. Some studies have shown that eating high-calorie foods frequently, eating between meals, and having lower days of physical activity are the most salient attributes to determining obesity levels. Besides, gender and age also may play a role in rising obesity trends.

2. Exploratory Analysis

During my exploratory analysis, the following visual was created to make a simple view of the Obesity Type Distribution. As we can see from this pie chart, it is almost equally distributed among all the elements of the obesity categorical types.

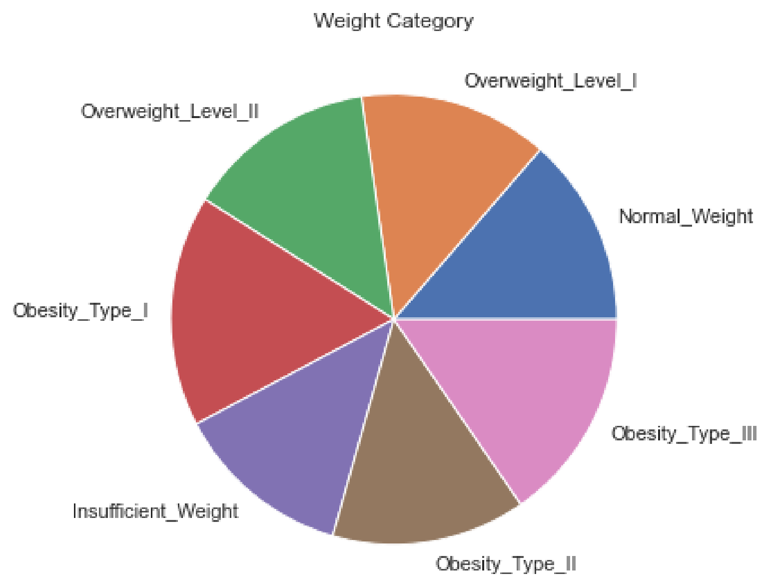


Figure 1.1 - Obesity Type Distribution Pie Chart

The pie charts below give us a clear view of the distribution of obesity types between men and women. We can see that Obesity Type III is the most prevalent type of obesity in women. While in men, the higher BMI is reflected by the large piece of Obesity Type II. Besides, we can see a higher proportion of Insufficient Weight in females compared to males. These results could be explained by eating the diet and maintaining the body are usually in women.

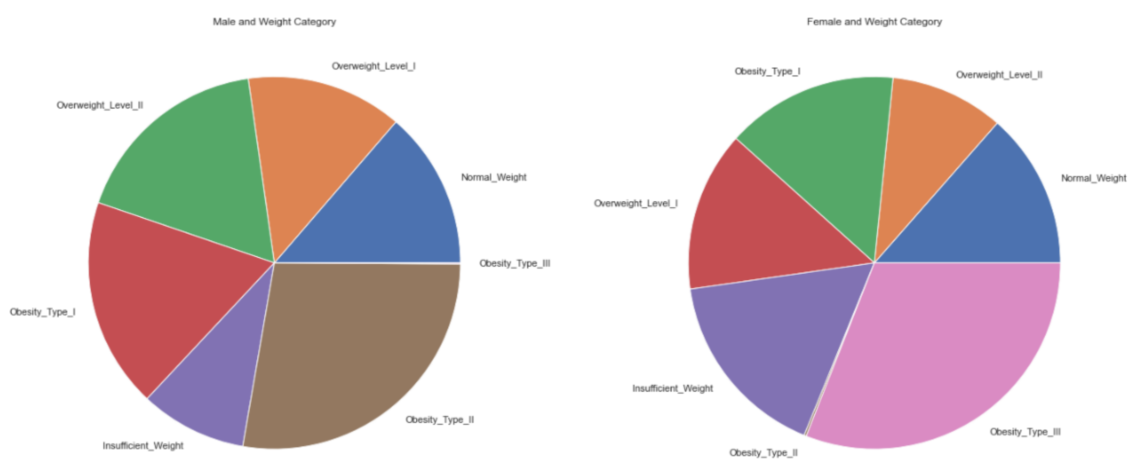


Figure 1.2 - Pie Charts Comparing Distribution of Obesity Types based on Gender

Gender can be explored as visualization by height and weight. We can see from the boxplot below that males and females are similarly distributed in height. While males are usually taller than females, both males and females have a similar average weight.

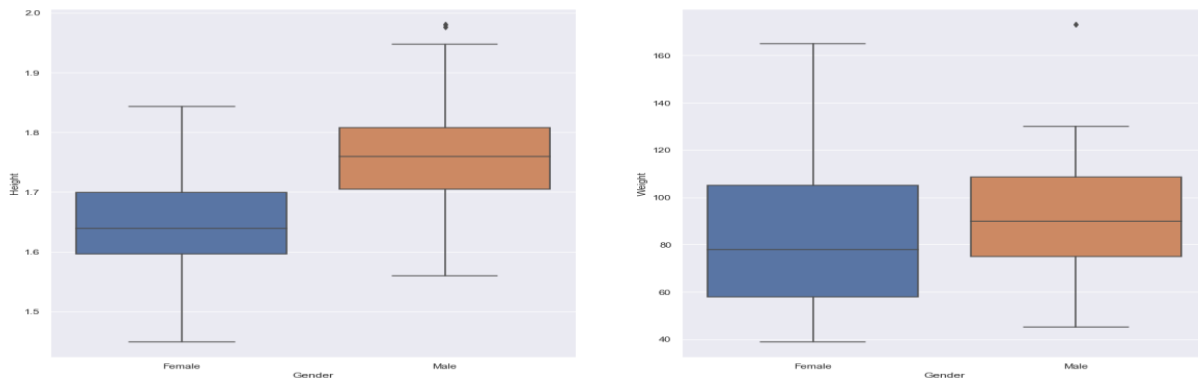


Figure 1.3 – Box Plot Comparing Height and Weight between Males and Females

3. Cluster Analysis

To perform the cluster analysis using K-Means algorithm, firstly I remove the NObeyesdad variable from the dataset. Then dummy variables are created for the categorical features. The K-Means algorithm looks at the nearest neighbor based on distance to group datapoints into clusters. The standard Euclidean distance function is used for the K-means clustering. Min-max normalization scales all values of the data between 0 and 1. K-means requires selecting a number for clusters (K number), so the values of K in this part are chosen including K = 5, K = 3, and K = 2. The maximum iterations used is 500. For each value of K, the cluster centroids were examined to determine if any pattern exists in the data. Next step, a silhouette analysis is performed to evaluate the separation between the resulting clusters and determine the quality of the clusters. The mean silhouette value is calculated and used as a threshold when determining the cluster quality.

The plot below shows the results of the silhouette analysis for K = 5. We can see that cluster 2 and cluster 4 performed well with many of their coefficients above the mean silhouette value. Three clusters 0, 1 and 3 did not perform since most of their coefficients are below the mean silhouette value. Four clusters display negative values which indicate that 5 clusters are not fit for the dataset.

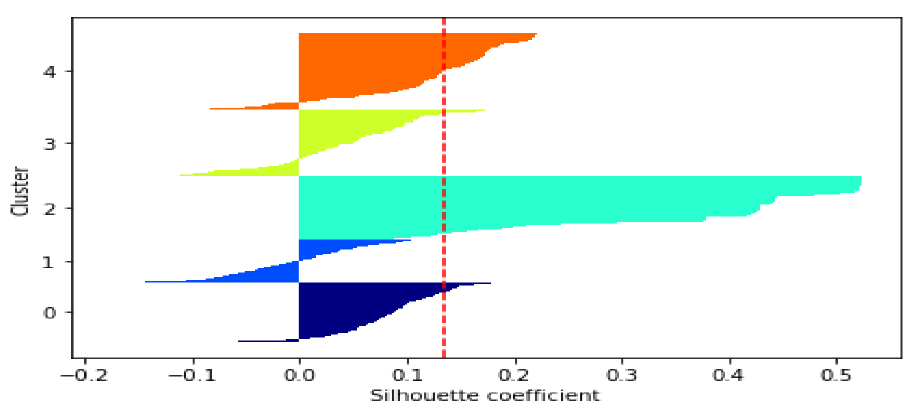


Figure 2.1 – Silhouettes Plot at K=5

At K=3, the plot of the silhouettes shows that cluster 2 outperformed better than the other clusters with all its coefficients above the mean silhouette value. Cluster 1 performed the worst and did not have any coefficients above the mean silhouette value and had negative coefficients. We may conclude that three clusters are still too high for this dataset.

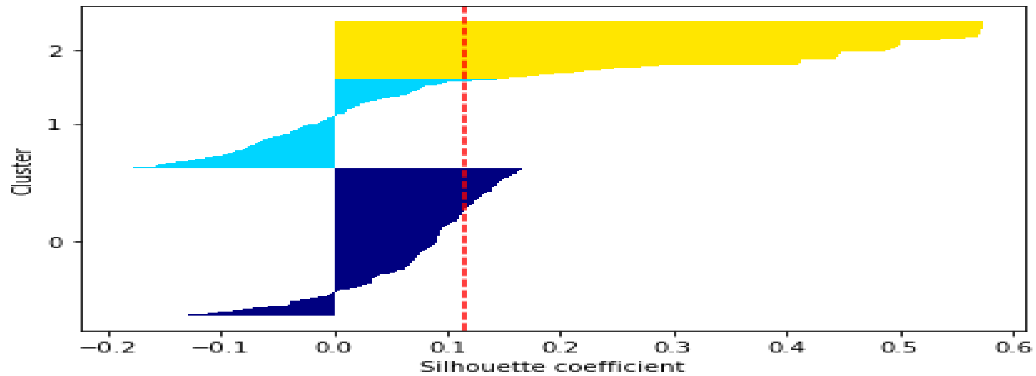


Figure 2.2 – Silhouettes Plot at K=3

At K = 2, this silhouette plot shows that both clusters 0 and 1 have coefficients that are above the mean silhouette value and none of the coefficients are negative. When looking at the centroids, the two features that most likely represent the clusters are Gender_Male and Gender_Female. In cluster 0, Gender_Female has a value of 1.00 while Gender_Male has a value 0.00. In cluster 1, Gender_Male has a value of 1.00 while Gender_Female has a value of -0.00. Besides, from the silhouette plots, we can conclude that cluster 0 represents females and cluster 1 represents males. This evaluation shows that gender may play a role in determining the classification of obesity levels.

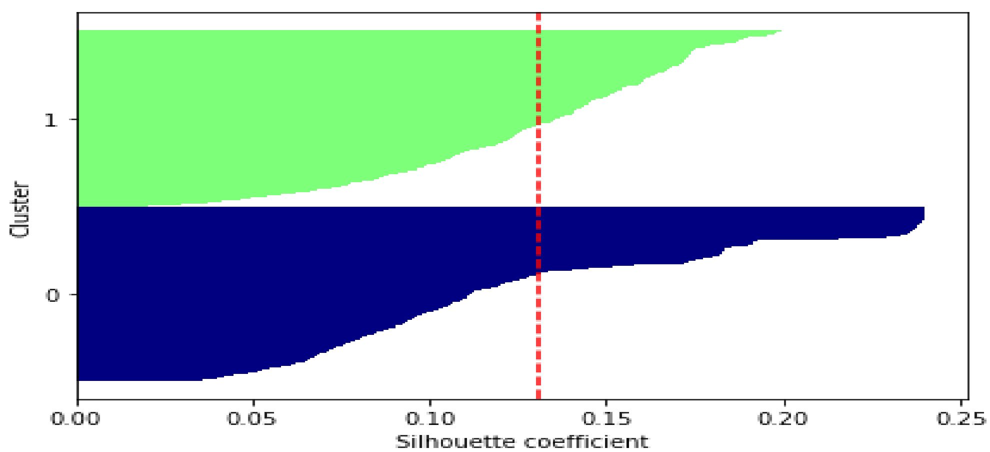


Figure 2.3 - Silhouettes Plot at K=2

Next, I am going to create age groups and separate the age based on generation to re-evaluate the clusters and determine if a pattern exists also within the age group for classification. The generation groups are divided into four groups related to the age period:

Boomer Generation in age 57 - 66 (1946 – 1964)

X Generation in age 41 - 56 (1965 – 1980)

Millennials Generation in age 25 - 40 (1981 – 1996)

Z Generation in age 9 - 24 (1997 – 2012)

The age groups are created by binning the Age attribute and then transforming the age group attribute into dummy variables. For exploratory purposes, K-means is performed on the dataset first without min-max normalization and second with min-max normalization at K = 3 with the three generational age groups: Gen-X and Boomers, Millennials, Gen-Z. The results of cluster analysis without normalization show a silhouette plot with all coefficients above the mean silhouette value. When looking at the centroids, cluster 2 shows Gen-Z at 0.9 while Millennials at .10 and Gen-X and Boomers at 0.00. Most likely Gen-Z is represented in cluster 2.

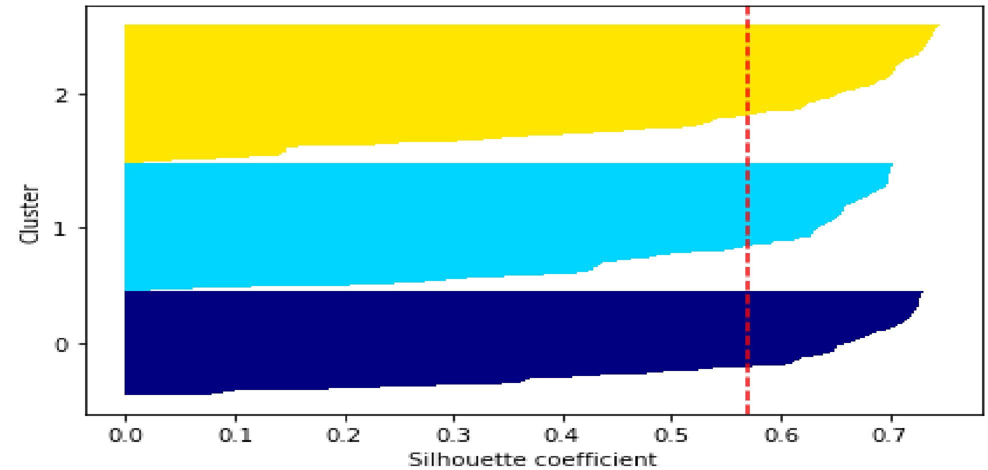


Figure 2.4 - Silhouettes Plot for Age Grouped at K=3

The completeness and homogeneity scores were calculated for clusters since the class labels exist. The completeness score was 0.70 which shows that members of a given class are assigned to the same cluster 70% of the time. The completeness score is positive and confirms that the clusters captured most of one class. The homogeneity score was much lower at 0.39 which shows that the clusters are not pure. These results may indicate that age group may be a factor in deciding the clusters for the data, but it may not be the main factor that affects obesity level for classification. The silhouette plots display that a pattern exists, but we must take into consideration that the data was not scaled.

Now I perform K-means again with the data normalized for K=3 to validate the results. We can see that Cluster 0 has coefficients above the mean silhouette value. Cluster 2 performed adequately with many of its coefficients above the mean silhouette value and only a few of its coefficients in negative. Cluster 1 did not perform as well as many of the coefficients are in negative and none of them are above the mean silhouette value. When looking at the centroids, the values of the age group do not directly correspond to the silhouette plots.

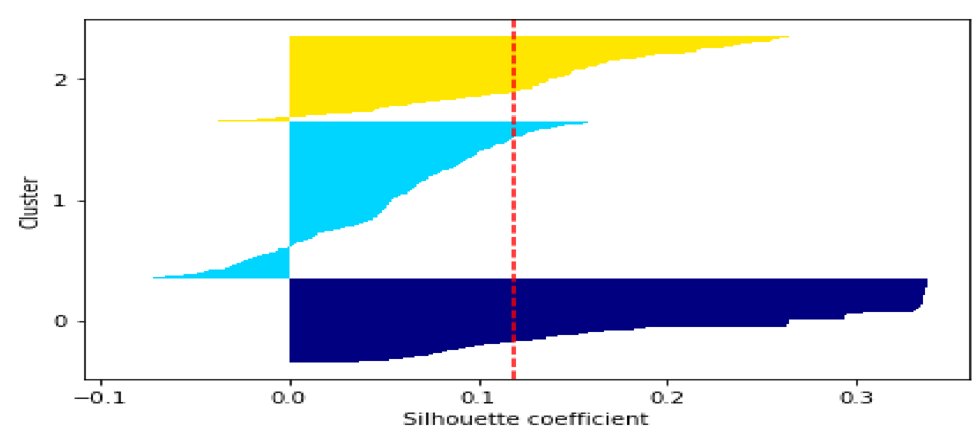


Figure 2.4 - Silhouettes Plot for Age Grouped at K=3 (normalized)

These results show that with the normalized data, a pattern may not necessarily appear in the age groups. When evaluating the completeness and homogeneity scores, we see both results are low scores. The completeness score was around 0.34 and the homogeneity score is 0.18. These scores show that grouping by age is not the main determining factor for the classification of obesity levels.

4. Classification with Decision Tree

In this part, I use the Decision Tree classifier model to evaluate the full dataset and each age-group dataset, then perform feature selection. The top features will determine which features ultimately affect obesity levels the most.

View class label names and numeric association:

0: 'Insufficient_Weight', 1: 'Normal_Weight', 2: 'Obesity_Type_I',

3: 'Obesity_Type_II', 4: 'Obesity_Type_III', 5: 'Overweight_Level_I',

6: 'Overweight_Level_II'

```
Accuracy:0.946

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        61
           1       0.93      0.91      0.92        45
           2       0.89      0.95      0.92        79
           3       0.95      0.96      0.95        54
           4       1.00      1.00      1.00        63
           5       0.92      0.93      0.93        61
           6       0.95      0.87      0.90        60

    accuracy                           0.95       423
   macro avg       0.95      0.94      0.95       423
weighted avg       0.95      0.95      0.95       423

Average Test Accuracy:  0.9456264775413712
Average Train Accuracy:  1.0
```

The classifier model on the full dataset performed very well with an accuracy of 94.6%. Class 0 and Class 4 had a 100% accurate prediction. Class 1, 3, 5, 6 achieved above 90% accuracy. The accuracy for the training set is 100% and the accuracy for the test set is 94.56%. The model is performing well and not overfitting since the accuracy for the test set is quite close to the training set and not experiencing high variance.

```
Age       470.510134679508
Weight    11390.601482312912
Gender_Female     274.57777589368993
Gender_Male       262.4874450895646
family_history_with_overweight_no         405.00183379903723
FCVC_Always       542.9949158091111
CAEC_Frequently           348.88961093191773
```

With the feature selection, using the top 15% of features, the classifier still performed well with an accuracy of 86.3%. Although the accuracy was reduced from the original feature set, the reduced feature set contains only seven features and still achieved a high level of accuracy. Class 1 had the lowest accuracy

Accuracy:0.863

Classification Report

	precision	recall	f1-score	support
0	0.93	0.93	0.93	61
1	0.71	0.78	0.74	45
2	0.82	0.82	0.82	79
3	0.91	0.98	0.95	54
4	1.00	1.00	1.00	63
5	0.83	0.80	0.82	61
6	0.80	0.72	0.75	60
accuracy			0.86	423
macro avg	0.86	0.86	0.86	423
weighted avg	0.86	0.86	0.86	423

score at 71%. Class 4 still maintained 100% accuracy. Moreover, for the full dataset, the top features that are associated with obesity levels are Age, Weight, Gender, family_history_with_overweight as ‘no’, FCVC as ‘Always’ and CAEC as ‘Frequently.’ These results confirm that gender is a feature in the classification of obesity levels. Besides gender, no history of obesity in the family is also an important feature when classifying obesity levels. Finally, always eating vegetables with meals (FCVC) and frequently eating food between meals (CAEC) are also the top features.

Similarly, the classifier model on the Gen-Z dataset performed with slightly lower accuracy at 91.1%. Class 3 had a 100% accurate prediction. Class 0 and 5 achieved above 90% accuracy. Class 1,2,4 and 6 achieved above 84% accuracy. With feature selection using the top 15% of features, the accuracy of the classifier decreased to 81.9%. This model with feature selection does not perform as well as the model using the full dataset. Class 4 had the highest accuracy at 95%, which is comparable to the full dataset which predicted class 4 at 100%. Class 6 had the lowest accuracy at 55%. This shows that for the Gen-Z age group, the model is unable to classify Class 6 using the top 15% of features. The top 15% of features include weight, gender as male, family_history_with_obesity as ‘no’, always eating vegetables with meals (FCVC) as ‘always’, and frequently eating food between meals.

The classifier model on the Millennials dataset did not perform as well as the model for the Gen-Z or full dataset. The model achieved an accuracy of 90.3%. Class 3 and Class 4 had a prediction accuracy of 100%. Class 6 performed better in this model with an accuracy of 95%. Class 0 had an accuracy of 40%, which is too lower in accuracy compared to the previous two models. With the feature selection using the top 15% of features, the model’s accuracy dropped to 80.6%. Class 4 again had the highest accuracy at 100%. Class 2 and Class 5 had the lowest accuracy at 58% and 56% respectively. Class 6 has a significant drop in accuracy, which prior to feature selection had a 95% prediction, and after feature selection has a 76% prediction. This shows that the features necessary to predict Class 6 are not included in the top 15% features. The top 15% of features include weight, gender both male and female, family_history_with_ obesity as ‘no’, always eating vegetables with meals (FCVC), and frequently eating food between meals. These features are the same top features from the model using the full dataset.

The classifier model for the Gen-X and Boomers dataset performed worse compared to all previous models. This model had the lowest accuracy score compared to the previous models, at 66.7%. This dataset is significantly smaller than the previous two datasets, so not all classes are represented in this model. This model was able to predict Class 1 at 100% accuracy, and Class 4 at 50%. This model was unable to predict Class 0 and Class 3. With the feature selection using the top 15% of features, the model maintained its accuracy at 66.7%. This model underperformed compared to all previous models and again, the model was unable to predict Class 0. The top 15% of features include weight and always eating vegetables with meals (FCVC) which are similar to full dataset, Gen-Z dataset, and Millennial dataset. Because the sample size is significantly lower, more data would be needed for this population to perform a more detailed and determine what key features affect the classification of obesity for the Gen-X and Boomers age group.

5. Conclusion

In this analysis, clustering analysis was explored on the dataset to discover patterns by groups in the data. The result shows that a pattern was only found with gender and not with age. The model with the best accuracy from the Decision Tree classifier is the full dataset. The top 15% features for this model include age, weight, gender, family history with obesity, always eating vegetables with meals (FCVC) and frequently eating food between meals (CAEC). With these top features, the model still performed well with an accuracy of 86.3%. Biological features and family history with obesity are top features that are associated with classifying obesity. With the full dataset, only two additional eating habit features were top features. Gen-Z age group is represented more in the full dataset compared to Millennials and Gen-X and Boomers. For the Gen Z and Millennials age group, biological and hereditary features are more associated with obesity levels. Besides, eating habits are additional top features, specifically eating vegetables with meals and eating between meals. Gender appears to play a role with Gen-Z and Millennials age group. This is expected due to biological factors such as difference in weight, height, and calorie intake. For Gen-X and Boomers age group, weight and direct eating habits such as water intake, calorie intake, direct physical activity, and mode of transportation are top features. More data is needed for Gen-X and Boomers to be able to analysis and evaluate the models and determine which features affect classification of obesity levels best.

In general, factors that affect the classification of obesity levels the most are age, gender, weight, and family history with obesity. Besides, always eating vegetables with meals and frequently eating between meals are also the most important factors, while physical activity attributes surprisingly were not the important factors in classifying obesity levels. Although age is considered, an individual’s generational group does not play an important role in classifying obesity levels.

Appendix A: Data Exploring

```
import numpy as np
import pylab as pl
import pandas as pd
import importlib
import matplotlib.pyplot as plt

import sklearn
from sklearn.model_selection import train_test_split

from sklearn.cluster import KMeans
from sklearn import decomposition
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import completeness_score, homogeneity_score
from sklearn.metrics import silhouette_samples

from sklearn import model_selection
from sklearn import tree
from sklearn import feature_selection
from sklearn import preprocessing
from sklearn import metrics
```

Loading dataset to Pandas dataframe

```
df = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programming M
achine Learning Applications/Final Project/Obeysity Project/obesity.csv', header=0
)
```

```
df
```

```
      Gender        Age    Height      Weight family_history_with_overweight  \
0     Female  21.000000  1.620000   64.000000                            yes
1     Female  21.000000  1.520000   56.000000                            yes
2       Male  23.000000  1.800000   77.000000                            yes
3       Male  27.000000  1.800000   87.000000                             no
4       Male  22.000000  1.780000   89.800000                             no
...      ...        ...       ...         ...                            ...
2106  Female  20.976842  1.710730  131.408528                            yes
2107  Female  21.982942  1.748584  133.742943                            yes
2108  Female  22.524036  1.752206  133.689352                            yes
2109  Female  24.361936  1.739450  133.346641                            yes
2110  Female  23.664709  1.738836  133.472641                            yes

     FAVC  FCVC  NCP       CAEC SMOKE      CH2O  SCC       FAF       TUE  \
0      no   2.0  3.0  Sometimes    no  2.000000   no  0.000000  1.000000
1      no   3.0  3.0  Sometimes   yes  3.000000  yes  3.000000  0.000000
2      no   2.0  3.0  Sometimes    no  2.000000   no  2.000000  1.000000
3      no   3.0  3.0  Sometimes    no  2.000000   no  2.000000  0.000000
4      no   2.0  1.0  Sometimes    no  2.000000   no  0.000000  0.000000
...   ...   ...  ...        ...   ...       ...  ...       ...       ...
2106  yes   3.0  3.0  Sometimes    no  1.728139   no  1.676269  0.906247
```

```
2107  yes   3.0  3.0  Sometimes    no  2.005130   no  1.341390  0.599270
2108  yes   3.0  3.0  Sometimes    no  2.054193   no  1.414209  0.646288
2109  yes   3.0  3.0  Sometimes    no  2.852339   no  1.139107  0.586035
2110  yes   3.0  3.0  Sometimes    no  2.863513   no  1.026452  0.714137

            CALC                 MTRANS           NObeyesdad
0             no  Public_Transportation        Normal_Weight
1      Sometimes  Public_Transportation        Normal_Weight
2     Frequently  Public_Transportation        Normal_Weight
3     Frequently                Walking   Overweight_Level_I
4      Sometimes  Public_Transportation  Overweight_Level_II
...          ...                    ...                  ...
2106   Sometimes  Public_Transportation     Obesity_Type_III
2107   Sometimes  Public_Transportation     Obesity_Type_III
2108   Sometimes  Public_Transportation     Obesity_Type_III
2109   Sometimes  Public_Transportation     Obesity_Type_III
2110   Sometimes  Public_Transportation     Obesity_Type_III

[2111 rows x 17 columns]
```

```
df.shape
```

```
(2111, 17)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 17 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2111 non-null   object
 1   Age                             2111 non-null   float64
 2   Height                          2111 non-null   float64
 3   Weight                          2111 non-null   float64
 4   family_history_with_overweight  2111 non-null   object
 5   FAVC                            2111 non-null   object
 6   FCVC                            2111 non-null   float64
 7   NCP                             2111 non-null   float64
 8   CAEC                            2111 non-null   object
 9   SMOKE                           2111 non-null   object
 10  CH2O                            2111 non-null   float64
 11  SCC                             2111 non-null   object
 12  FAF                             2111 non-null   float64
 13  TUE                             2111 non-null   float64
 14  CALC                            2111 non-null   object
 15  MTRANS                          2111 non-null   object
 16  NObeyesdad                      2111 non-null   object
dtypes: float64(8), object(9)
memory usage: 280.5+ KB
```

```
df.describe()
```

```
               Age       Height       Weight         FCVC          NCP  \
count  2111.000000  2111.000000  2111.000000  2111.000000  2111.000000
mean     24.312600     1.701677    86.586058     2.419043     2.685628
std       6.345968     0.093305    26.191172     0.533927     0.778039
min      14.000000     1.450000    39.000000     1.000000     1.000000
25%      19.947192     1.630000    65.473343     2.000000     2.658738
50%      22.777890     1.700499    83.000000     2.385502     3.000000
75%      26.000000     1.768464   107.430682     3.000000     3.000000
max      61.000000     1.980000   173.000000     3.000000     4.000000

              CH2O          FAF          TUE
count  2111.000000  2111.000000  2111.000000
mean      2.008011     1.010298     0.657866
std       0.612953     0.850592     0.608927
min       1.000000     0.000000     0.000000
25%       1.584812     0.124505     0.000000
50%       2.000000     1.000000     0.625350
75%       2.477420     1.666678     1.000000
max       3.000000     3.000000     2.000000
```

```
df.columns
```

```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE', 'CH2O', 'SCC', 'FAF', 'TUE',
       'CALC', 'MTRANS', 'NObeyesdad'],
      dtype='object')
```

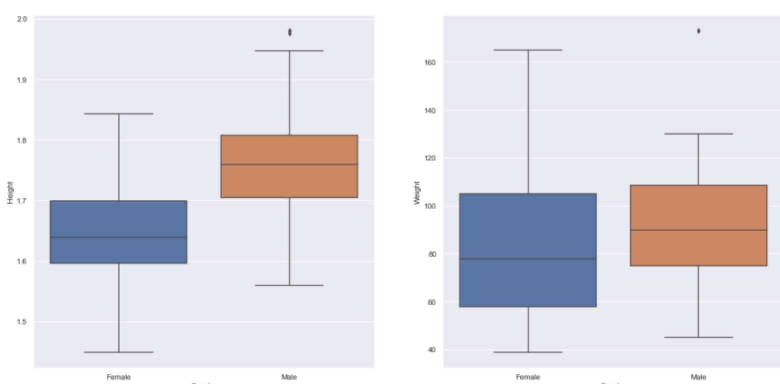
```
df.columns = ['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight
','FAVC', 'FCVC', 'NCP', 'CAEC', 'SMOKE',
              'CH2O', 'SCC', 'FAF', 'TUE','CALC', 'MTRANS', 'NObeyesdad']
df
```

```
      Gender        Age    Height      Weight family_history_with_overweight  \
0     Female  21.000000  1.620000   64.000000                            yes
1     Female  21.000000  1.520000   56.000000                            yes
2       Male  23.000000  1.800000   77.000000                            yes
3       Male  27.000000  1.800000   87.000000                             no
4       Male  22.000000  1.780000   89.800000                             no
...      ...        ...       ...         ...                            ...
2106  Female  20.976842  1.710730  131.408528                            yes
2107  Female  21.982942  1.748584  133.742943                            yes
2108  Female  22.524036  1.752206  133.689352                            yes
2109  Female  24.361936  1.739450  133.346641                            yes
2110  Female  23.664709  1.738836  133.472641                            yes

     FAVC  FCVC  NCP       CAEC SMOKE      CH2O  SCC       FAF       TUE  \
0      no   2.0  3.0  Sometimes    no  2.000000   no  0.000000  1.000000
1      no   3.0  3.0  Sometimes   yes  3.000000  yes  3.000000  0.000000
2      no   2.0  3.0  Sometimes    no  2.000000   no  2.000000  1.000000
3      no   3.0  3.0  Sometimes    no  2.000000   no  2.000000  0.000000
4      no   2.0  1.0  Sometimes    no  2.000000   no  0.000000  0.000000
...   ...   ...  ...        ...   ...       ...  ...       ...       ...
2106  yes   3.0  3.0  Sometimes    no  1.728139   no  1.676269  0.906247
2107  yes   3.0  3.0  Sometimes    no  2.005130   no  1.341390  0.599270
```

```
2108  yes   3.0  3.0  Sometimes    no  2.054193   no  1.414209  0.646288
2109  yes   3.0  3.0  Sometimes    no  2.852339   no  1.139107  0.586035
2110  yes   3.0  3.0  Sometimes    no  2.863513   no  1.026452  0.714137

            CALC                 MTRANS           NObeyesdad
0             no  Public_Transportation        Normal_Weight
1      Sometimes  Public_Transportation        Normal_Weight
2     Frequently  Public_Transportation        Normal_Weight
3     Frequently                Walking   Overweight_Level_I
4      Sometimes  Public_Transportation  Overweight_Level_II
...          ...                    ...                  ...
2106   Sometimes  Public_Transportation     Obesity_Type_III
2107   Sometimes  Public_Transportation     Obesity_Type_III
2108   Sometimes  Public_Transportation     Obesity_Type_III
2109   Sometimes  Public_Transportation     Obesity_Type_III
2110   Sometimes  Public_Transportation     Obesity_Type_III

[2111 rows x 17 columns]
```

```
import seaborn as sns

sns.set()
fig = plt.figure(figsize=(20,10))
plt.subplot(1, 2, 1)
sns.boxplot(x='Gender', y='Height', data=df)
plt.subplot(1, 2, 2)
sns.boxplot(x='Gender', y='Weight', data=df)
```

```
<AxesSubplot:xlabel='Gender', ylabel='Weight'>
```

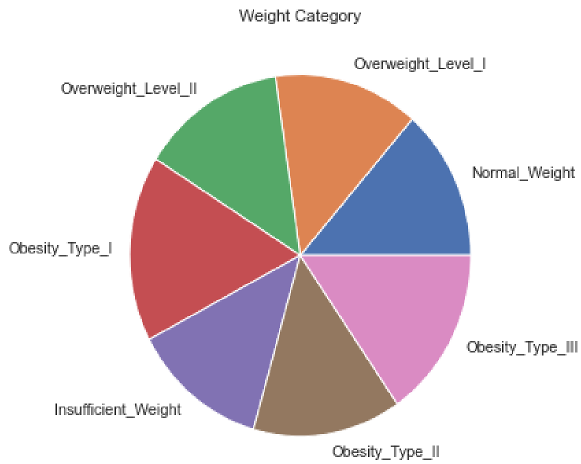



```
import collections
from collections import Counter

c = Counter(df['NObeyesdad'])
print(c)
```

```
Counter({'Obesity_Type_I': 351, 'Obesity_Type_III': 324, 'Obesity_Type_II': 297, '
Overweight_Level_I': 290, 'Overweight_Level_II': 290, 'Normal_Weight': 287, 'Insuf
ficient_Weight': 272})
```

```
fig = plt.figure(figsize=(6,6))
plt.pie([float(c[v]) for v in c], labels=[str(k) for k in c], autopct=None)
```

```
plt.title('Weight Category')
plt.tight_layout()
```



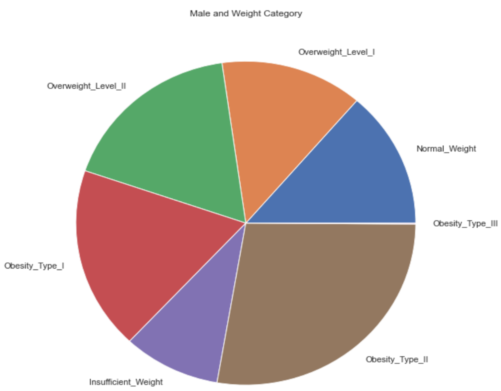

```
filt = df['Gender'] == 'Male'
c_m = Counter(df.loc[filt, 'NObeyesdad'])
print(c_m)
```

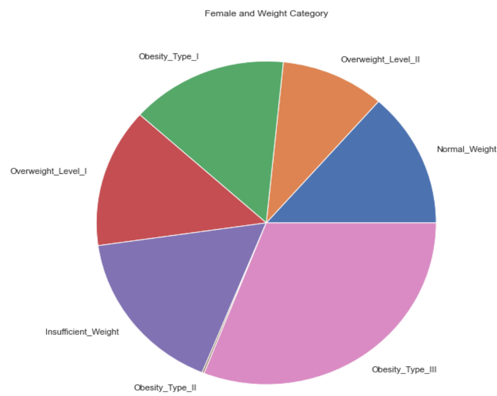
```
Counter({'Obesity_Type_II': 295, 'Obesity_Type_I': 195, 'Overweight_Level_II': 187
, 'Normal_Weight': 146, 'Overweight_Level_I': 145, 'Insufficient_Weight': 99, 'Obe
sity_Type_III': 1})
```

```
c_f = Counter(df.loc[~filt, 'NObeyesdad'])
print(c_f)
```

```
Counter({'Obesity_Type_III': 323, 'Insufficient_Weight': 173, 'Obesity_Type_I': 15
6, 'Overweight_Level_I': 145, 'Normal_Weight': 141, 'Overweight_Level_II': 103, 'O
besity_Type_II': 2})
```

```
fig = plt.figure(figsize=(20,8))
plt.subplot(1, 2, 1)
plt.pie([float(c_m[v]) for v in c_m], labels=[str(k) for k in c_m], autopct=None)
plt.title('Male and Weight Category')
plt.tight_layout()

plt.subplot(1, 2, 2)
plt.pie([float(c_f[v]) for v in c_f], labels=[str(k) for k in c_f], autopct=None)
plt.title('Female and Weight Category')
plt.tight_layout()
```





Appendix B: Clustering K-Mean

```
import numpy as np
import pylab as pl
import pandas as pd
import importlib
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn import decomposition
from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import completeness_score, homogeneity_score
from sklearn.metrics import silhouette_samples

df = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programming M
achine Learning Applications/Final Project/Obeysity Project/obesity.csv', header=0
)
```

To perform the cluster analysis using K-Means algorithm, firstly I remove the NObeyesdad variable from the dataset.

```
df2 = df.iloc[:,:16]


# Convert FCVC, NCP, CH20, FAF, and TUE into a Categorical Feature by first, conve
rting it from Float to Integer:
cleaned_data['FCVC'] = cleaned_data['FCVC'].astype('int')
cleaned_data['NCP'] = cleaned_data['NCP'].astype('int')
cleaned_data['CH2O'] = cleaned_data['CH2O'].astype('int')
cleaned_data['FAF'] = cleaned_data['FAF'].astype('int')
cleaned_data['TUE'] = cleaned_data['TUE'].astype('int')

# Convert Age from Float to Integer:
cleaned_data['Age'] = cleaned_data['Age'].astype('int')
cleaned_data.dtypes
```

```
Gender                            object
Age                                int32
Height                           float64
Weight                           float64
family_history_with_overweight    object
FAVC                              object
FCVC                               int32
NCP                                int32
CAEC                              object
SMOKE                             object
CH2O                               int32
SCC                               object
FAF                                int32
TUE                                int32
CALC                              object
MTRANS                            object
dtype: object
```

```
# Rename values in FCVC into Categorical Names:
cleaned_data['FCVC'] = cleaned_data['FCVC'].replace({1: 'Never'})
cleaned_data['FCVC'] = cleaned_data['FCVC'].replace({2: 'Sometimes'})
cleaned_data['FCVC'] = cleaned_data['FCVC'].replace({3: 'Always'})

# Rename values in NCP into Categorical Names:
cleaned_data['NCP'] = cleaned_data['NCP'].replace({1: '1'})
cleaned_data['NCP'] = cleaned_data['NCP'].replace({2: '2'})
cleaned_data['NCP'] = cleaned_data['NCP'].replace({3: '3'})
cleaned_data['NCP'] = cleaned_data['NCP'].replace({4: '3+'})

# Rename values in CH2O into Categorical Names:
cleaned_data['CH2O'] = cleaned_data['CH2O'].replace({1: 'Less than a liter'})
cleaned_data['CH2O'] = cleaned_data['CH2O'].replace({2: 'Between 1 and 2 L'})
cleaned_data['CH2O'] = cleaned_data['CH2O'].replace({3: 'More than 2 L'})

# Rename values in FAF into Categorical Names:
cleaned_data['FAF'] = cleaned_data['FAF'].replace({0: 'I do not have'})
cleaned_data['FAF'] = cleaned_data['FAF'].replace({1: '1 or 2 days'})
cleaned_data['FAF'] = cleaned_data['FAF'].replace({2: '2 or 4 days'})
cleaned_data['FAF'] = cleaned_data['FAF'].replace({3: '4 or 5 days'})

# Rename values in TUE into Categorical Names:
cleaned_data['TUE'] = cleaned_data['TUE'].replace({0: '0-2 Hours'})
cleaned_data['TUE'] = cleaned_data['TUE'].replace({1: '3-5 Hours'})
cleaned_data['TUE'] = cleaned_data['TUE'].replace({2: 'More than 5 Hours'})

# Save Numeric Dataframe for future use:
data_numeric.to_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programmin
g Machine Learning Applications/Final Project/Obeysity Project/obesity_numeric.csv
', index = False)

# Normalize the numeric dataset with Min-Max Scaling:
df_min_max_scaled = data_numeric.copy()
for column in df_min_max_scaled.columns:
```

```
    df_min_max_scaled[column] = (df_min_max_scaled[column] - df_min_max_scaled[col
umn].min()) / (df_min_max_scaled[column].max() - df_min_max_scaled[column].min())
```

```
# View class labels:
labels_df =  df['NObeyesdad']
labels_df
```

```
0              Normal_Weight
1              Normal_Weight
2              Normal_Weight
3         Overweight_Level_I
4        Overweight_Level_II
                ...
2106        Obesity_Type_III
2107        Obesity_Type_III
2108        Obesity_Type_III
2109        Obesity_Type_III
2110        Obesity_Type_III
Name: NObeyesdad, Length: 2111, dtype: object
```

```
# Transform class label into numeric:
le = preprocessing.LabelEncoder()
labels_num = le.fit_transform(labels_df)
labels_num
```

```
array([1, 1, 1, ..., 4, 4, 4])
```

```
# View class label names and numeric association:
label_names = dict(zip(le.transform(le.classes_), le.classes_))
print(label_names)
```

```
{0: 'Insufficient_Weight', 1: 'Normal_Weight', 2: 'Obesity_Type_I', 3: 'Obesity_Ty
pe_II', 4: 'Obesity_Type_III', 5: 'Overweight_Level_I', 6: 'Overweight_Level_II'}
```

The K-Means algorithm looks at the nearest neighbor based on distance to group datapoints into clusters. The standard Euclidean distance function is used for the K-means clustering. Min-max normalization scales all values of the data between 0 and 1. K-means requires selecting a number for clusters (K number), so the values of K in this part are chosen including K = 5, K = 3, and K = 2. The maximum iterations used is 500. For each value of K, the cluster centroids were examined to determine if any pattern exists in the data. Next step, a silhouette analysis is performed to evaluate the separation between the resulting clusters and determine the quality of the clusters. The mean silhouette value is calculated and used as a threshold when determining the cluster quality.

```
#initialize k-means with n = 5
kmeans = KMeans(n_clusters=5, max_iter=500, verbose=1)
kmeans.fit(df_min_max_scaled)
```

```
Initialization complete
Iteration 0, inertia 11066.101640170213
Iteration 1, inertia 8360.414436554367
Iteration 2, inertia 8186.470992220824
Iteration 3, inertia 8106.25615858154
Iteration 4, inertia 8062.842443331067
```

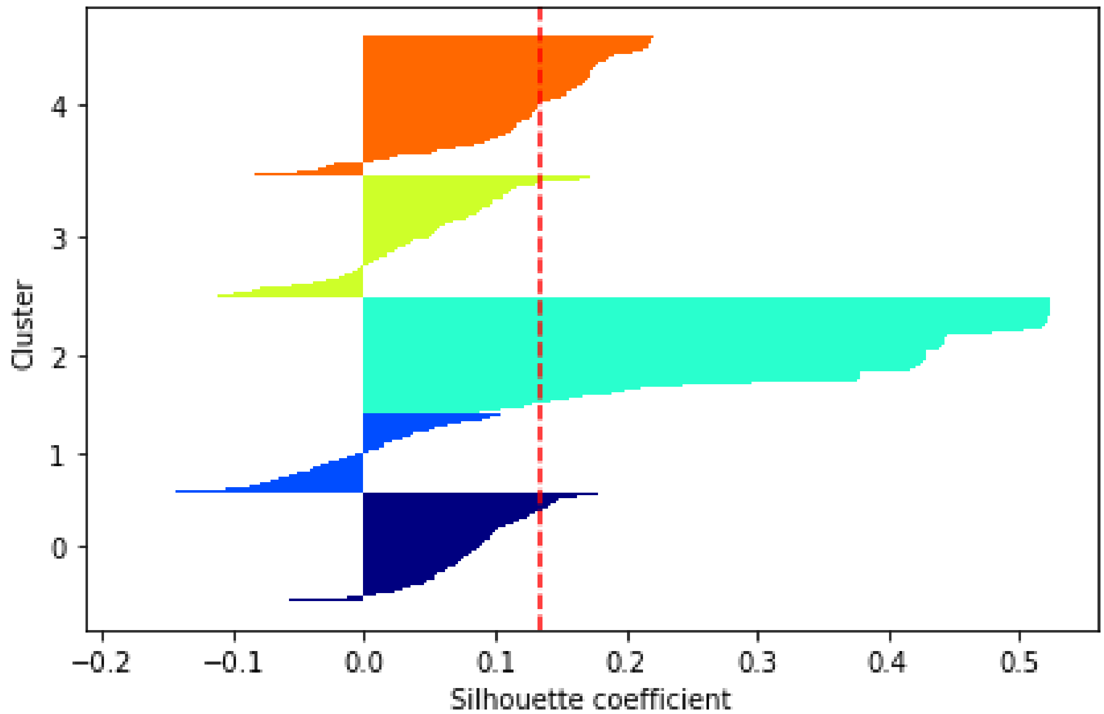
```
Iteration 5, inertia 8030.075683527646
Iteration 6, inertia 8015.152418512467
Iteration 7, inertia 8005.362074057147
Iteration 8, inertia 7998.6319272701785
Iteration 9, inertia 7989.325169352731
Iteration 10, inertia 7985.293744865411
Iteration 11, inertia 7962.842499132133
Iteration 12, inertia 7948.710932438474
…
Iteration 23, inertia 7867.807454464741
Iteration 24, inertia 7866.794860734273
Iteration 25, inertia 7866.580445432262
Iteration 26, inertia 7866.479112486189
Iteration 27, inertia 7866.430261428807
Converged at iteration 27: strict convergence.

KMeans(max_iter=500, n_clusters=5, verbose=1)

clusters5 = kmeans.predict(df_min_max_scaled)

pd.DataFrame(clusters5, columns=["Cluster"])

      Cluster
0           3
1           2
2           3
3           1
4           4
...       ...
2106        2
2107        2
2108        2
2109        2
2110        2

[2111 rows x 1 columns]

def cluster_sizes(clusters):
    size = {}
    cluster_labels = np.unique(clusters)
    n_clusters = cluster_labels.shape[0]

    for c in cluster_labels:
        size[c] = len(df[clusters == c])
    return size

size5 = cluster_sizes(clusters5)

for c5 in size5.keys():
    print("Size of Cluster", c5, ": ", size5[c5])

Size of Cluster 0 :  406
Size of Cluster 1 :  298
Size of Cluster 2 :  430
```

```
Size of Cluster 3 :  460
Size of Cluster 4 :  517
```

```
# The centroids provide an aggregate representation and a characterization of each cluster:
pd.options.display.float_format='{:,.2f}'.format

centroids5 = pd.DataFrame(kmeans.cluster_centers_, columns=df_min_max_scaled.columns.values)


# Silhouette Analysis at n = 5:
c5_silhouette = metrics.silhouette_samples(df_min_max_scaled, clusters5)
print('Mean Silhouette Value :', c5_silhouette.mean())
```

```
Mean Silhouette Value : 0.1340150852248124
```

```
def plot_silhouettes(data, clusters, metric='euclidean'):

    from matplotlib import cm
    from sklearn.metrics import silhouette_samples

    cluster_labels = np.unique(clusters)
    n_clusters = cluster_labels.shape[0]
    silhouette_vals = metrics.silhouette_samples(data, clusters, metric='euclidean')
    c_ax_lower, c_ax_upper = 0, 0
    cticks = []
    for i, k in enumerate(cluster_labels):
        c_silhouette_vals = silhouette_vals[clusters == k]
        c_silhouette_vals.sort()
        c_ax_upper += len(c_silhouette_vals)
        color = cm.jet(float(i) / n_clusters)
        pl.barh(range(c_ax_lower, c_ax_upper), c_silhouette_vals, height=1.0,
                       edgecolor='none', color=color)

        cticks.append((c_ax_lower + c_ax_upper) / 2)
        c_ax_lower += len(c_silhouette_vals)

    silhouette_avg = np.mean(silhouette_vals)
    pl.axvline(silhouette_avg, color="red", linestyle="--")

    pl.yticks(cticks, cluster_labels)
    pl.ylabel('Cluster')
    pl.xlabel('Silhouette coefficient')

    pl.tight_layout()
    pl.show()

    return

# Plot and Evaluate the Silhouettes:
plot_silhouettes(df_min_max_scaled, clusters5)
```



The plot above shows the results of the silhouette analysis for K = 5. We can see that cluster 2 and cluster 4 performed well with many of their coefficients above the mean silhouette value. Three clusters 0, 1 and 3 did not perform since most of their coefficients are below the mean silhouette value. Four clusters display negative values which indicate that 5 clusters are not fit for the dataset.

```
# k-means with n = 3
kmeans3 = KMeans(n_clusters=3, max_iter=500, verbose=1)

kmeans3.fit(df_min_max_scaled)
```

```
Initialization complete
Iteration 0, inertia 14135.315132533431
Iteration 1, inertia 9320.214367519184
Iteration 2, inertia 8989.279144478374
Iteration 3, inertia 8887.939027582415
Iteration 4, inertia 8864.666671331554
Iteration 5, inertia 8853.909806854133
Iteration 6, inertia 8840.47688918553
Iteration 7, inertia 8815.620472638182
……

Iteration 39, inertia 8765.670664641966
Iteration 40, inertia 8765.423219001728
Iteration 41, inertia 8765.398780834896
Converged at iteration 41: strict convergence.

KMeans(max_iter=500, n_clusters=3, verbose=1)
```

```
clusters3 = kmeans3.predict(df_min_max_scaled)

size3 = cluster_sizes(clusters3)
```

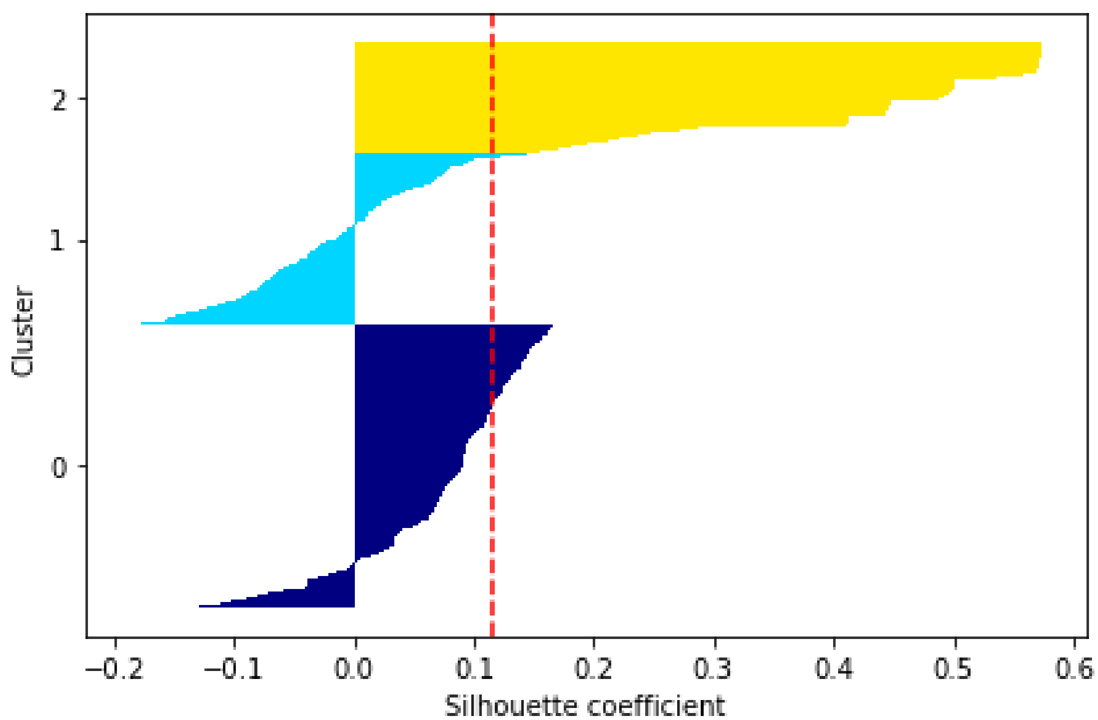
```
for c in size3.keys():
    print("Size of Cluster", c, ": ", size3[c])
```

```
Size of Cluster 0 :  1058
Size of Cluster 1 :  636
Size of Cluster 2 :  417
```

```
# View centroids for an aggregate representation and a characterization of each cl
uster:
pd.options.display.float_format='{:,.2f}'.format

centroids3 = pd.DataFrame(kmeans3.cluster_centers_, columns=df_min_max_scaled.colu
mns.values)


# Silhouette Analysis at n = 3:
c3_silhouette = metrics.silhouette_samples(df_min_max_scaled, clusters3)
print('Mean Silhouette Value :', c3_silhouette.mean())
```

```
Mean Silhouette Value : 0.11634874352766442
```

```
# Plot and Evaluate the Silhouettes:
plot_silhouettes(df_min_max_scaled, clusters3)
```



At K=3, the plot of the silhouettes shows that cluster 2 outperformed better than the other clusters with all its coefficients above the mean silhouette value. Cluster 1 performed the worst and did not have any coefficients above the mean silhouette value and had negative coefficients. We may conclude that three clusters are still too high for this dataset.

```
# k-means with n = 2
kmeans2 = KMeans(n_clusters=2, max_iter=500, verbose=1)
```

```
kmeans2.fit(df_min_max_scaled)
```

```
Initialization complete
Iteration 0, inertia 13960.75307590449
Iteration 1, inertia 9578.434933304481
Iteration 2, inertia 9454.434799667557
Iteration 3, inertia 9440.386447690518
Iteration 4, inertia 9439.746651906486
Iteration 5, inertia 9439.703145546142
Converged at iteration 5: strict convergence.
Initialization complete
Iteration 0, inertia 15761.320024178247
Iteration 1, inertia 9963.709082711048
Iteration 2, inertia 9849.34245965224
Iteration 3, inertia 9825.99785802669
Iteration 4, inertia 9820.123866787919
Iteration 5, inertia 9818.034274050513
```

…….

```
Iteration 6, inertia 9439.746651906486
Iteration 7, inertia 9439.703145546142
Converged at iteration 7: strict convergence.
```

```
KMeans(max_iter=500, n_clusters=2, verbose=1)
```

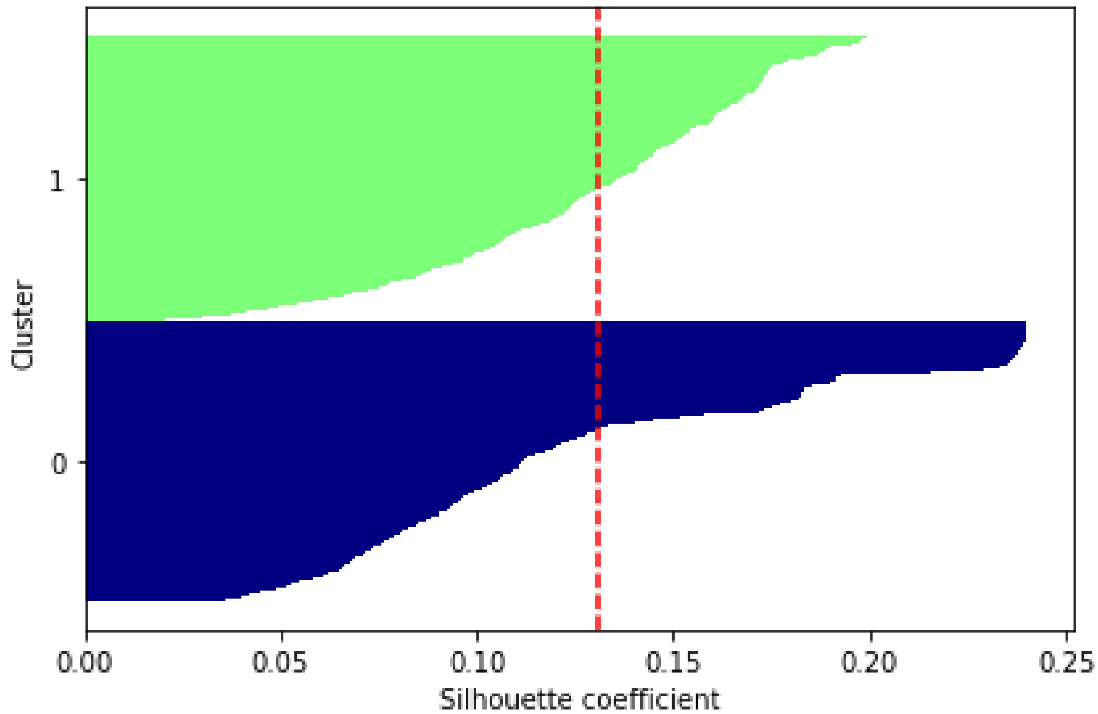
```
clusters2 = kmeans2.predict(df_min_max_scaled)

size2 = cluster_sizes(clusters2)

for c in size2.keys():
    print("Size of Cluster", c, ": ", size2[c])
```

```
Size of Cluster 0 :  1044
Size of Cluster 1 :  1067
```

```
# View centroids for an aggregate representation and a characterization of each cl
uster:
pd.options.display.float_format='{:,.2f}'.format

centroids2 = pd.DataFrame(kmeans2.cluster_centers_, columns=df_min_max_scaled.colu
mns.values)


# Silhouette Analysis at n = 2:
c2_silhouette = metrics.silhouette_samples(df_min_max_scaled, clusters2)
print('Mean Silhouette Value :', c2_silhouette.mean())
```

```
Mean Silhouette Value : 0.13093478332005926
```

```
# Plot and Evaluate the Silhouettes:
plot_silhouettes(df_min_max_scaled, clusters2)
```



At K = 2, this silhouette plot shows that both clusters 0 and 1 have coefficients that are above the mean silhouette value and none of the coefficients are negative. When looking at the centroids, the two features that most likely represent the clusters are Gender_Male and Gender_Female. In cluster 0, Gender_Female has a value of 1.00 while Gender_Male has a value 0.00. In cluster 1, Gender_Male has a value of 1.00 while Gender_Female has a value of -0.00. Besides, from the silhouette plots, we can conclude that cluster 0 represents females and cluster 1 represents males. This evaluation shows that gender may play a role in determining the classification of obesity levels.

Next, we will create age groups and seperate the age of each individual based on generation. Exploring age groups will allow us to re-evaluate the clusters and determine if a pattern exists also within age group for classification. Discretize the Age attribute into 4 seperate age groups and re-run K-Means Clustering: Gen-Z (9 – 24); Millennials (25 – 40); Gen-X (41 – 56); Boomers (57 - 66)

```
age_bins = pd.qcut(data_numeric.Age, [0, .61, .972, 1])
age_bins.head(5)
```

```
0    (13.999, 24.0]
1    (13.999, 24.0]
2    (13.999, 24.0]
3      (24.0, 40.0]
4    (13.999, 24.0]
Name: Age, dtype: category
Categories (3, interval[float64, right]): [(13.999, 24.0] < (24.0, 40.0] < (40.0, 61.0]]
```

```
age_bins = pd.qcut(data_numeric.Age, [0, .61, .972, 1], labels = ['Gen-Z', 'Millenials', 'Gen-X & Boomers'])
age_df = pd.concat([age_bins, df2['Age']], axis=1)
age_df.columns = ['Age Group', 'Age']
age_df.head(10)
```

```
    Age Group  Age
0       Gen-Z   21
1       Gen-Z   21
2       Gen-Z   23
3  Millenials   27
4       Gen-Z   22
5  Millenials   29
6       Gen-Z   23
7       Gen-Z   22
8       Gen-Z   24
9       Gen-Z   22
```

```
data_age_groups = data_numeric
data_age_groups["Age"] = age_df['Age Group']

# Create Dummy Variables for Binned Dataset:
df_age_groups = pd.get_dummies(data_age_groups)
```

The age groups are created by binning the Age attribute and then transforming the age group attribute into dummy variables. For exploratory purposes, K-means is performed on the dataset first without min-max normalization and second with min-max normalization at K = 3 with the three generational age groups: Gen-X and Boomers, Millennials, Gen-Z.

```
# Perform K-Means Clustering with N = 3:
kmeans = KMeans(n_clusters=3, max_iter=500, verbose=1)
kmeans.fit(df_age_groups)

Initialization complete
Iteration 0, inertia 243646.6387142869
Iteration 1, inertia 210543.42499733614
Iteration 2, inertia 209002.86484584527
Iteration 3, inertia 208962.08181874343
Converged at iteration 3: center shift 0.0011258354680577789 within tolerance 0.001
5358503717230955.
Initialization complete
Iteration 0, inertia 334512.84451211605
Iteration 1, inertia 239585.93414120717
Iteration 2, inertia 229845.02398452605
Iteration 3, inertia 221314.8380126595
……

Iteration 4, inertia 209703.02658441666
Iteration 5, inertia 209362.03202101548
Iteration 6, inertia 209013.56709803344
Iteration 7, inertia 208954.93374886585
Iteration 8, inertia 208952.94787903145
Converged at iteration 8: center shift 0.0008306717121226769 within tolerance 0.00
15358503717230955.

KMeans(max_iter=500, n_clusters=3, verbose=1)

age_clusters = kmeans.predict(df_age_groups)
```

```
size = cluster_sizes(age_clusters)

for c in size.keys():
    print("Size of Cluster", c, ": ", size[c])
```

```
Size of Cluster 0 :  591
Size of Cluster 1 :  731
Size of Cluster 2 :  789
```

```
# View centroids for an aggregate representation and a characterization of each cl
uster:
pd.options.display.float_format='{:,.2f}'.format

centroids = pd.DataFrame(kmeans.cluster_centers_, columns=df_age_groups.columns.va
lues)


centroids['Age_Gen-Z'] #clusters containing Gen-Z
```

```
0   0.90
1   0.43
2   0.64
Name: Age_Gen-Z, dtype: float64
```

```
centroids['Age_Millenials'] #clusters containing Millenials
```

```
0   0.10
1   0.56
2   0.31
Name: Age_Millenials, dtype: float64
```

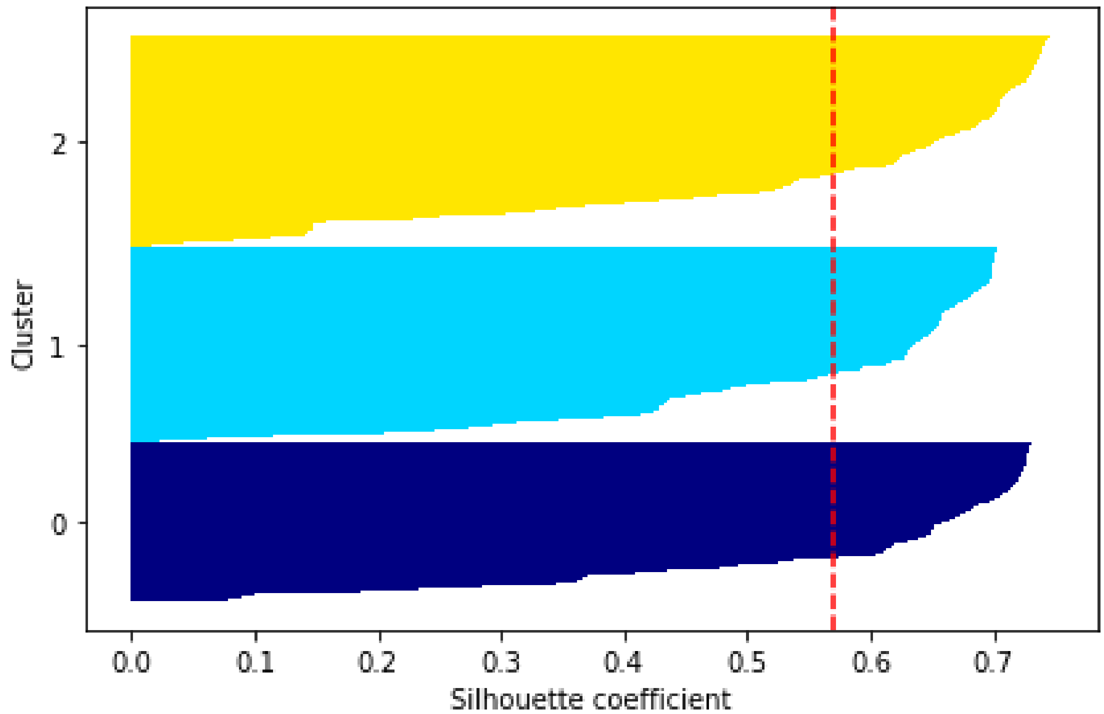
```
centroids['Age_Gen-X & Boomers'] #clusters containing Gen-X and Boomers
```

```
0   0.00
1   0.01
2   0.04
Name: Age_Gen-X & Boomers, dtype: float64
```

```
# Silhouette Analysis at n = 3:
age_silhouette = metrics.silhouette_samples(df_age_groups, age_clusters)
print('Mean Silhouette Value :', age_silhouette.mean())
```

```
Mean Silhouette Value : 0.5691256560102319
```

```
# Plot and Evaluate the Silhouettes:
plot_silhouettes(df_age_groups, age_clusters)
```



The results of cluster analysis without normalization show a silhouette plot with all coefficients above the mean silhouette value. When looking at the centroids, cluster 2 shows Gen-Z at 0.9 while Millennials at .10 and Gen-X and Boomers at 0.00. Most likely Gen-Z is represented in cluster 2.

```
# Calculate Completeness and Homogeneity for the clusters:
complete = completeness_score(labels_num, age_clusters)
print(f"Completeness Score for Clusters: {complete}")
homogene = homogeneity_score(labels_num, age_clusters)
print(f"Homogeneity Score for Clusters: {homogene}")
```

```
Completeness Score for Clusters: 0.7020884578966542
Homogeneity Score for Clusters: 0.39448267211636195
```

The completeness and homogeneity scores were calculated for clusters since the class labels exist. The completeness score was 0.70 which shows that members of a given class are assigned to the same cluster 70% of the time. The completeness score is positive and confirms that the clusters captured most of one class. The homogeneity score was much lower at 0.39 which shows that the clusters are not pure. These results may indicate that age group may be a factor in deciding the clusters for the data, but it may not be the main factor that affects obesity level for classification. The silhouette plots display that a pattern exists, but we must take into consideration that the data was not scaled.

Now I perform K-means again with the data normalized for K=3 to validate the results.

```
# Normalize the dataset with Min-Max Scaling:
df_age_groups_norm = df_age_groups.copy()
for column in df_age_groups_norm.columns:
    df_age_groups_norm[column] = (df_age_groups_norm[column] - df_age_groups_norm[
column].min()) / (df_age_groups_norm[column].max() - df_age_groups_norm[column].mi
n())
```

```
# Perform K-Means Clustering with N = 3:
kmeans3 = KMeans(n_clusters=3, max_iter=500, verbose=1)

kmeans3.fit(df_age_groups_norm)

Initialization complete
Iteration 0, inertia 15649.062601909736
Iteration 1, inertia 10036.064245859783
Iteration 2, inertia 9863.46995200902
Iteration 3, inertia 9778.366713234653
Iteration 4, inertia 9752.208153425938
Iteration 5, inertia 9704.588050516266
Iteration 6, inertia 9654.95487199607
… …

Iteration 5, inertia 9666.966201469197
Iteration 6, inertia 9665.314471689406
Iteration 7, inertia 9665.118337318918
Iteration 8, inertia 9665.07483024741
Converged at iteration 8: strict convergence.

KMeans(max_iter=500, n_clusters=3, verbose=1)

clusters_norm3 = kmeans3.predict(df_age_groups_norm)

size3 = cluster_sizes(clusters_norm3)

for c in size3.keys():
    print("Size of Cluster", c, ": ", size3[c])

Size of Cluster 0 :  514
Size of Cluster 1 :  1015
Size of Cluster 2 :  582

# View centroids for an aggregate representation and a characterization of each cl
uster:
pd.options.display.float_format='{:,.2f}'.format

centroids3 = pd.DataFrame(kmeans3.cluster_centers_, columns=df_age_groups_norm.col
umns.values)


centroids3['Age_Gen-Z'] #clusters containing Gen-Z Normalized

0   0.61
1   0.99
2   0.06
Name: Age_Gen-Z, dtype: float64

centroids3['Age_Millenials'] #clusters containing Millenials Normalized
```

```
0    0.39
1    0.01
2    0.87
Name: Age_Millenials, dtype: float64
```

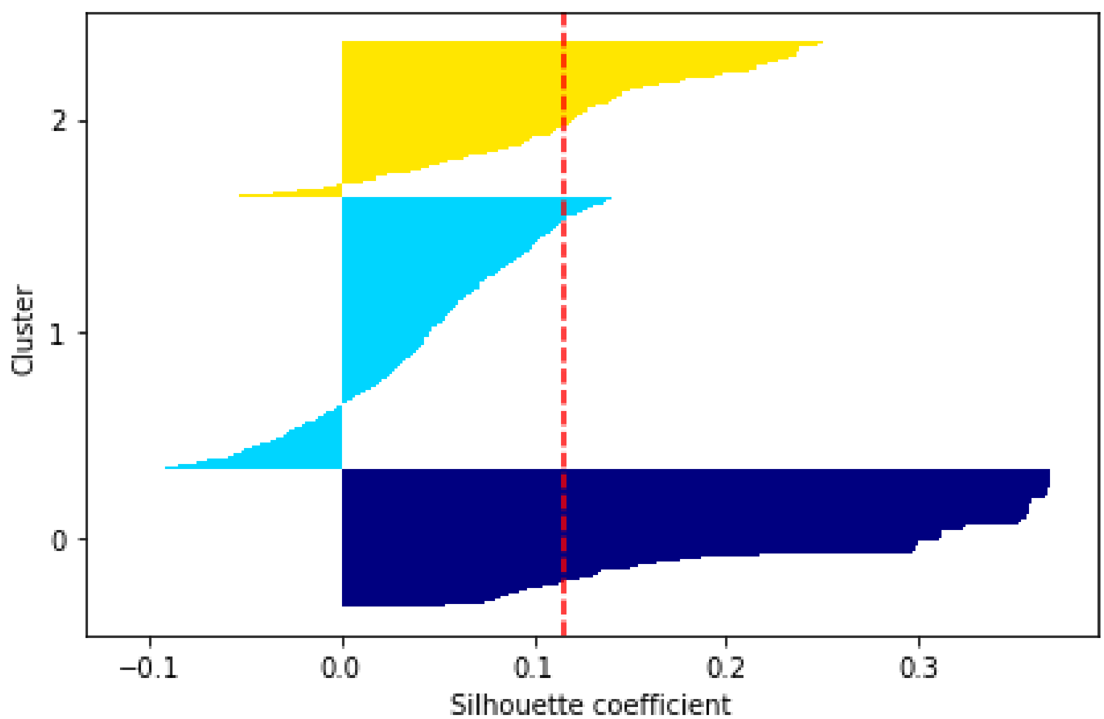
```
centroids3['Age_Gen-X & Boomers'] #clusters containing Gen-X and Boomers Normalize
d
```

```
0    0.01
1    0.00
2    0.06
Name: Age_Gen-X & Boomers, dtype: float64
```

```
# Silhouette Analysis at n = 3:
age_norm_silhouette = metrics.silhouette_samples(df_age_groups_norm, clusters_norm
3)
print('Mean Silhouette Value :', age_norm_silhouette.mean())
```

```
Mean Silhouette Value : 0.11611406296849332
```

```
# Plot and Evaluate the Silhouettes:
plot_silhouettes(df_age_groups_norm, clusters_norm3)
```



We can see that Cluster 0 has coefficients above the mean silhouette value. Cluster 2 performed adequately with many of its coefficients above the mean silhouette value and only a few of its coefficients in negative. Cluster 1 did not perform as well as many of the coefficients are in negative and none of them are above the mean silhouette value. When looking at the centroids, the values of the age group do not directly correspond to the silhouette plots.

```
# Calculate Completeness and Homogeneity for the clusters:
complete_norm = completeness_score(labels_num, clusters_norm3)
print(f"Completeness Score for Clusters: {complete_norm}")
```

```
homogene_norm = homogeneity_score(labels_num, clusters_norm3)
print(f"Homogeneity Score for Clusters: {homogene_norm}")
```

```
Completeness Score for Clusters: 0.340494223096423
Homogeneity Score for Clusters: 0.18426693761791008
```

When evaluating the completeness and homogeneity scores, we see both results are low scores. The completeness score was around 0.34 and the homogeneity score is 0.18. These scores show that grouping by age is not the main determining factor for the classification of obesity levels.

Save Output of Data-Set (non-normalized) based on Age-Groups for Classifier Use:

```
# Create a copy of the data with the Age Groups:
data_age_groups = data_numeric
data_age_groups["Age"] = age_df['Age Group']

# Add the class labels as a column to the dataset:
data_age_groups['NObeyesdad'] = labels_df


genz_df = data_age_groups[data_age_groups["Age"] == 'Gen-Z']


#Save Gen-Z dataframe to CSV:
genz_df.to_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programming Mac
hine Learning Applications/Final Project/Obeysity Project/genz_dataframe.csv', ind
ex = False)

millen_df = data_age_groups[data_age_groups["Age"] == 'Millenials']


# Save Millenials dataframe to CSV:
millen_df.to_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programming M
achine Learning Applications/Final Project/Obeysity Project/millenials_dataframe.c
sv', index = False)

genxboomers_df = data_age_groups[data_age_groups["Age"] == 'Gen-X & Boomers']


# Save Gen-X and Boomers dataframe to CSV:
genxboomers_df.to_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programm
ing Machine Learning Applications/Final Project/Obeysity Project/genxboomers_dataf
rame.csv', index = False)
```

Appendix C: Decision Tree

```
import numpy as np
import pylab as pl
import pandas as pd
import importlib
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import model_selection
from sklearn import tree
from sklearn import feature_selection
from sklearn import preprocessing
from sklearn import metrics

# Load original dataset to Pandas dataframe:
df = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programming M
achine Learning Applications/Final Project/Obeysity Project/obesity.csv', header=0
)
df.head()

   Gender   Age  Height  Weight family_history_with_overweight FAVC  FCVC  \
0  Female  21.0    1.62    64.0                            yes   no   2.0
1  Female  21.0    1.52    56.0                            yes   no   3.0
2    Male  23.0    1.80    77.0                            yes   no   2.0
3    Male  27.0    1.80    87.0                             no   no   3.0
4    Male  22.0    1.78    89.8                             no   no   2.0

   NCP       CAEC SMOKE  CH2O  SCC  FAF  TUE        CALC  \
0  3.0  Sometimes    no   2.0   no  0.0  1.0          no
1  3.0  Sometimes   yes   3.0  yes  3.0  0.0   Sometimes
2  3.0  Sometimes    no   2.0   no  2.0  1.0  Frequently
3  3.0  Sometimes    no   2.0   no  2.0  0.0  Frequently
4  1.0  Sometimes    no   2.0   no  0.0  0.0   Sometimes

                  MTRANS           NObeyesdad
0  Public_Transportation        Normal_Weight
1  Public_Transportation        Normal_Weight
2  Public_Transportation        Normal_Weight
3                Walking   Overweight_Level_I
4  Public_Transportation  Overweight_Level_II

# Load transformed dataset with numeric values only:
data_numeric = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Pro
gramming Machine Learning Applications/Final Project/Obeysity Project/Obesity_nume
ric.csv', header=0)
# Load Gen-Z Dataframe:
genz_df = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Programm
ing Machine Learning Applications/Final Project/Obeysity Project/genz_dataframe.cs
v', header=0)
# Load Millenials Dataframe:
millen_df = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - Progra
mming Machine Learning Applications/Final Project/Obeysity Project/millenials_data
```

```
frame.csv', header=0)
# Load Gen-X and Boomers Dataframe:
genxboomers_df = pd.read_csv('D:/Study/Master - Data Science/Study/10. DSC 478 - P
rogramming Machine Learning Applications/Final Project/Obeysity Project/genxboomer
s_dataframe.csv', header=0)
```

Decision Tree and Feature Selection with Full Dataset:

```
# Obtain the class label from original dataset:
labels_df =  df['NObeyesdad']
labels_df
```

```
0              Normal_Weight
1              Normal_Weight
2              Normal_Weight
3         Overweight_Level_I
4        Overweight_Level_II
                ...
2106        Obesity_Type_III
2107        Obesity_Type_III
2108        Obesity_Type_III
2109        Obesity_Type_III
2110        Obesity_Type_III
Name: NObeyesdad, Length: 2111, dtype: object
```

```
# Transform class label into numeric:
le = preprocessing.LabelEncoder()
labels_num = le.fit_transform(labels_df)
labels_num
```

```
array([1, 1, 1, ..., 4, 4, 4])
```

```
# View class label names and numeric association:
label_names = dict(zip(le.transform(le.classes_), le.classes_))
print(label_names)
```

```
{0: 'Insufficient_Weight', 1: 'Normal_Weight', 2: 'Obesity_Type_I', 3: 'Obesity_Ty
pe_II', 4: 'Obesity_Type_III', 5: 'Overweight_Level_I', 6: 'Overweight_Level_II'}
```

```
# Build training and test sets:
x_train, x_test, label_train, label_test = train_test_split(data_numeric, labels_n
um, test_size=0.2, random_state=1)

# Train Decision tree Classifier on the Training Data:
d_tree = tree.DecisionTreeClassifier()
dt_all = d_tree.fit(x_train, label_train)

# Function for Measure Performance:
def measure_performance(X, y, clf, show_accuracy=True, show_classification_report=
True, show_confussion_matrix=True):
    y_pred = clf.predict(X)
    if show_accuracy:
        print ("Accuracy:{0:.3f}".format(metrics.accuracy_score(y, y_pred)),"\n")
```

```
    if show_classification_report:
        print ("Classification Report")
        print (metrics.classification_report(y, y_pred, zero_division=0),"\n")

    if show_confussion_matrix:
        print ("Confussion Matrix")
        print (metrics.confusion_matrix(y, y_pred),"\n")

# Predict on Test Set, View Performance, and Accuracy of Decision Tree Model:
measure_performance(x_test, label_test, dt_all, show_confussion_matrix=True, show_
classification_report=True)
```

```
Accuracy:0.946

Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        61
           1       0.93      0.91      0.92        45
           2       0.89      0.95      0.92        79
           3       0.95      0.96      0.95        54
           4       1.00      1.00      1.00        63
           5       0.92      0.93      0.93        61
           6       0.95      0.87      0.90        60

    accuracy                           0.95       423
   macro avg       0.95      0.94      0.95       423
weighted avg       0.95      0.95      0.95       423


Confussion Matrix
[[60  1  0  0  0  0  0]
 [ 0 41  0  0  0  4  0]
 [ 0  0 75  3  0  0  1]
 [ 0  0  2 52  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  2  0  0  0 57  2]
 [ 0  0  7  0  0  1 52]]
```

```
# View the Accuracy of the Test and Training Sets:
print('Average Test Accuracy: ', d_tree.score(x_test, label_test))
print('Average Train Accuracy: ', d_tree.score(x_train, label_train))
```

```
Average Test Accuracy:  0.9456264775413712
Average Train Accuracy:  1.0
```

The classifier model on the full dataset performed very well with an accuracy of 94.6%. Class 0 and Class 4 had a 100% accurate prediction. Class 1, 3, 5, 6 achieved above 90% accuracy. The accuracy for the training set is 100% and the accuracy for the test set is 94.56%. The model is performing well and not overfitting since the accuracy for the test set is quite close to the training set and not experiencing high variance

```python
# Perform feature selection for top 15%
fs = feature_selection.SelectPercentile(feature_selection.chi2, percentile=15)
x_train_fs = fs.fit_transform(x_train, label_train)

# View the top 15% of the most important features:
print(data_numeric.columns[fs.get_support()].values)
```

```
['Age' 'Weight' 'Gender_Female' 'Gender_Male'
 'family_history_with_overweight_no' 'FCVC_Always' 'CAEC_Frequently']
```

```python
# View scores for each top feature:
for i in range(len(data_numeric.columns.values)):
    if fs.get_support()[i]:
        print(data_numeric.columns.values[i], '\t', fs.scores_[i])
```

```
Age 	 470.510134679508
Weight 	 11390.601482312912
Gender_Female 	 274.57777589368993
Gender_Male 	 262.4874450895646
family_history_with_overweight_no 	 405.00183379903723
FCVC_Always 	 542.9949158091111
CAEC_Frequently 	 348.88961093191773
```

```python
# Evaluate the Classifier with the top 15% feature set:
d_tree.fit(x_train_fs, label_train)
x_test_fs = fs.transform(x_test)
measure_performance(x_test_fs, label_test, d_tree, show_confussion_matrix=True, sh
ow_classification_report=True)
```

```
Accuracy:0.863

Classification Report
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        61
           1       0.71      0.78      0.74        45
           2       0.82      0.82      0.82        79
           3       0.91      0.98      0.95        54
           4       1.00      1.00      1.00        63
           5       0.83      0.80      0.82        61
           6       0.80      0.72      0.75        60

    accuracy                           0.86       423
   macro avg       0.86      0.86      0.86       423
weighted avg       0.86      0.86      0.86       423


Confussion Matrix
[[57  4  0  0  0  0  0]
 [ 4 35  0  0  0  5  1]
 [ 0  0 65  5  0  2  7]
 [ 0  0  1 53  0  0  0]
 [ 0  0  0  0 63  0  0]
 [ 0  9  0  0  0 49  3]
```

```
 [ 0  1 13  0  0  3 43]]
```

With the feature selection, using the top 15% of features, the classifier still performed well with an accuracy of 86.3%. Although the accuracy was reduced from the original feature set, the reduced feature set contains only seven features and still achieved a high level of accuracy. Class 1 had the lowest accuracy score at 71%. Class 4 still maintained 100% accuracy. Moreover, for the full dataset, the top features that are associated with obesity levels are Age, Weight, Gender, family_history_with_overweight as 'no', FCVC as 'Always' and CAEC as 'Frequently.' These results confirm that gender is a feature in the classification of obesity levels. Besides gender, no history of obesity in the family is also an important feature when classifying obesity levels. Finally, always eating vegetables with meals (FCVC) and frequently eating food between meals (CAEC) are also the top features.

Decision Tree and Feature Selection with Gen-Z Dataset:

```
#Remove the age and class label column for Gen-Z DF:
data_genz = genz_df.iloc[:,1:43]


# View Class Labels for Gen-Z DF:
labels_genz =  genz_df['NObeyesdad']
labels_genz.head()

0          Normal_Weight
1          Normal_Weight
2          Normal_Weight
3    Overweight_Level_II
4          Normal_Weight
Name: NObeyesdad, dtype: object

# Transform class label into numeric:
le_z = preprocessing.LabelEncoder()
genz_labels = le_z.fit_transform(labels_genz)
genz_labels

array([1, 1, 1, ..., 4, 4, 4])

# Build training and test sets for Gen-Z:
genz_train, genz_test, genz_label_train, genz_label_test = train_test_split(data_g
enz, genz_labels, test_size=0.2, random_state=1)

# Train Decision tree Classifier on the Training Data:
dt_genz = d_tree.fit(genz_train, genz_label_train)

# Predict on Gen-Z Test Set, View Performance, and Accuracy of Decision Tree Model
:
measure_performance(genz_test, genz_label_test, dt_genz, show_confussion_matrix=Tr
ue, show_classification_report=True)

Accuracy:0.911

Classification Report
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        49
           1       0.89      0.85      0.87        48
           2       0.89      1.00      0.94        51
           3       1.00      0.72      0.84        18
           4       0.87      0.95      0.91        21
           5       0.94      0.85      0.90        55
           6       0.84      0.90      0.87        29

    accuracy                           0.91       271
   macro avg       0.91      0.90      0.90       271
weighted avg       0.91      0.91      0.91       271


Confussion Matrix
[[49  0  0  0  0  0  0]
 [ 2 41  0  0  0  2  3]
 [ 0  0 51  0  0  0  0]
 [ 0  0  2 13  3  0  0]
 [ 0  0  1  0 20  0  0]
 [ 0  5  1  0  0 47  2]
 [ 0  0  2  0  0  1 26]]
```

```
# View the Accuracy of the Test and Training Sets:
print('Average Test Accuracy: ', d_tree.score(genz_test, genz_label_test))
print('Average Train Accuracy: ', d_tree.score(genz_train, genz_label_train))
```

```
Average Test Accuracy:  0.9114391143911439
Average Train Accuracy:  1.0
```

Similarly, the classifier model on the Gen-Z dataset performed with slightly lower accuracy at 91.1%. Class 3 had a 100% accurate prediction. Class 0 and 5 achieved above 90% accuracy. Class 1,2,4 and 6 achieved above 84% accuracy.

```
# Perform feature selection for top 15% of Gen-Z DF:
fs_genz = feature_selection.SelectPercentile(feature_selection.chi2, percentile=15
)
genz_train_fs = fs_genz.fit_transform(genz_train, genz_label_train)
```

```
# View the top 15% of the most important features for Gen-Z:
print(data_genz.columns[fs_genz.get_support()].values)
```

```
['Weight' 'Gender_Male' 'family_history_with_overweight_no' 'FAVC_no'
 'FCVC_Always' 'NCP_2' 'CAEC_Frequently']
```

```
# View scores for each top feature:
for i in range(len(data_genz.columns.values)):
    if fs_genz.get_support()[i]:
        print(data_genz.columns.values[i], '\t', fs_genz.scores_[i])
```

```
Weight       9715.93028639861
Gender_Male        119.37576369900033
```

```
family_history_with_overweight_no    230.20882848759723
FAVC_no      135.28900924705363
FCVC_Always        292.28229522019336
NCP_2        167.46598969723757
CAEC_Frequently    202.1009043591279
```

```python
# Evaluate the Classifier with the top 15% feature set for Gen-Z DF:
d_tree.fit(genz_train_fs, genz_label_train)
genz_test_fs = fs_genz.transform(genz_test)
measure_performance(genz_test_fs, genz_label_test, d_tree, show_confussion_matrix=
True, show_classification_report=True)
```

```
Accuracy:0.819

Classification Report
              precision    recall  f1-score   support

           0       0.98      0.88      0.92        49
           1       0.77      0.85      0.81        48
           2       0.86      0.86      0.86        51
           3       0.80      0.67      0.73        18
           4       0.95      1.00      0.98        21
           5       0.80      0.80      0.80        55
           6       0.55      0.59      0.57        29

    accuracy                           0.82       271
   macro avg       0.82      0.81      0.81       271
weighted avg       0.82      0.82      0.82       271


Confussion Matrix
[[43  6  0  0  0  0  0]
 [ 1 41  0  0  0  2  4]
 [ 0  0 44  3  0  1  3]
 [ 0  0  5 12  1  0  0]
 [ 0  0  0  0 21  0  0]
 [ 0  4  0  0  0 44  7]
 [ 0  2  2  0  0  8 17]]
```

With feature selection using the top 15% of features, the accuracy of the classifier decreased to 81.9%. This model with feature selection does not perform as well as the model using the full dataset. Class 4 had the highest accuracy at 95%, which is comparable to the full dataset which predicted class 4 at 100%. Class 6 had the lowest accuracy at 55%. This shows that for the Gen-Z age group, the model is unable to classify Class 6 using the top 15% of features. The top 15% of features include weight, gender as male, family_history_with_obesity as ‘no’, always eating vegetables with meals (FCVC) as ‘always’, and frequently eating food between meals.

Decision Tree and Feature Selection with Millenials Dataset:

```
#Remove the age and class label column for Millenials DF:
data_millen = millen_df.iloc[:,1:43]


# View Class Labels for Millenials DF:
labels_millen =  millen_df['NObeyesdad']
labels_millen.head()
```

```
0     Overweight_Level_I
1          Normal_Weight
2         Obesity_Type_I
3    Overweight_Level_II
4         Obesity_Type_I
Name: NObeyesdad, dtype: object
```

```
# Transform class label into numeric:
le_m = preprocessing.LabelEncoder()
millen_labels = le_m.fit_transform(labels_millen)
millen_labels
```

```
array([5, 1, 2, 6, 2, 6, 5, 6, 6, 3, 6, 6, 6, 3, 1, 2, 1, 1, 6, 2, 1, 1,
       1, 1, 5, 2, 1, 2, 1, 1, 5, 6, 1, 1, 5, 6, 1, 6, 3, 1, 1, 1, 5, 2,
       2, 2, 1, 5, 4, 2, 1, 1, 3, 6, 3, 6, 6, 1, 5, 0, 6, 1, 1, 1, 5, 1,
       1, 1, 1, 1, 5, 1, 5, 1, 1, 5, 1, 2, 5, 6, 5, 6, 1, 6, 1, 2, 1, 1,
       6, 4, 1, 1, 1, 1, 6, 2, 5, 2, 6, 5, 1, 1, 1, 6, 6, 1, 4, 4, 4, 4,
       0, 0, 0, 0, 0, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
       6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 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, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
```

```
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4])

# Build training and test sets for Millenials:
mil_train, mil_test, mil_label_train, mil_label_test = train_test_split(data_mille
n, millen_labels, test_size=0.2, random_state=1)

# Train Decision tree Classifier on the Training Data:
dt_mil = d_tree.fit(mil_train, mil_label_train)

# Predict on Millenials Test Set, View Performance, and Accuracy of Decision Tree
Model:
measure_performance(mil_test, mil_label_test, dt_mil, show_confussion_matrix=True,
show_classification_report=True)

Accuracy:0.903

Classification Report
              precision    recall  f1-score   support

           0       0.40      1.00      0.57         2
           1       0.83      0.62      0.71        16
           2       0.77      0.89      0.83        19
           3       1.00      0.91      0.95        43
           4       1.00      1.00      1.00        30
           5       0.79      0.85      0.81        13
           6       0.95      1.00      0.98        21

    accuracy                           0.90       144
   macro avg       0.82      0.90      0.84       144
weighted avg       0.92      0.90      0.91       144


Confussion Matrix
[[ 2  0  0  0  0  0  0]
 [ 3 10  0  0  0  2  1]
 [ 0  1 17  0  0  1  0]
 [ 0  0  4 39  0  0  0]
 [ 0  0  0  0 30  0  0]
 [ 0  1  1  0  0 11  0]
 [ 0  0  0  0  0  0 21]]


# View the Accuracy of the Test and Training Sets:
print('Average Test Accuracy: ', d_tree.score(mil_test, mil_label_test))
print('Average Train Accuracy: ', d_tree.score(mil_train, mil_label_train))

Average Test Accuracy:  0.9027777777777778
Average Train Accuracy:  1.0
```

The classifier model on the Millennials dataset did not perform as well as the model for the Gen-Z or full dataset. The model achieved an accuracy of 90.3%. Class 3 and Class 4 had a prediction accuracy of 100%. Class 6 performed better in this model with an accuracy of 95%. Class 0 had an accuracy of 40%, which is too lower in accuracy compared to the previous two models.

```
# Perform feature selection for top 15% of Millenials DF:
fs_mil = feature_selection.SelectPercentile(feature_selection.chi2, percentile=15)
mil_train_fs = fs_mil.fit_transform(mil_train, mil_label_train)

# View the top 15% of the most important features for Millenails:
print(data_millen.columns[fs_mil.get_support()].values)
```

```
['Weight' 'Gender_Female' 'Gender_Male'
 'family_history_with_overweight_no' 'FCVC_Always' 'CAEC_Frequently'
 'MTRANS_Automobile']
```

```
# View scores for each top feature:
for i in range(len(data_millen.columns.values)):
    if fs_mil.get_support()[i]:
        print(data_millen.columns.values[i], '\t', fs_mil.scores_[i])
```

```
Weight 	 1599.4360572768592
Gender_Female 	 173.42692440859898
Gender_Male 	 146.10242506447887
family_history_with_overweight_no 	 169.8920540299759
FCVC_Always 	 249.67035685056916
CAEC_Frequently 	 166.34169934064465
MTRANS_Automobile 	 110.75284152840752
```

```
# Evaluate the Classifier with the top 15% feature set for Millenials DF:
d_tree.fit(mil_train_fs, mil_label_train)
mil_test_fs = fs_mil.transform(mil_test)
measure_performance(mil_test_fs, mil_label_test, d_tree, show_confussion_matrix=True, show_classification_report=True)
```

```
Accuracy:0.806

Classification Report
              precision    recall  f1-score   support

           0       0.50      1.00      0.67         2
           1       0.83      0.62      0.71        16
           2       0.58      0.74      0.65        19
           3       0.95      0.86      0.90        43
           4       1.00      1.00      1.00        30
           5       0.56      0.77      0.65        13
           6       0.76      0.62      0.68        21

    accuracy                           0.81       144
   macro avg       0.74      0.80      0.75       144
weighted avg       0.83      0.81      0.81       144


Confussion Matrix
[[ 2  0  0  0  0  0  0]
 [ 2 10  0  0  0  3  1]
 [ 0  0 14  2  0  2  1]
 [ 0  0  4 37  0  1  1]
 [ 0  0  0  0 30  0  0]
```

```
 [ 0  1  1  0  0 10  1]
 [ 0  1  5  0  0  2 13]]
```

With the feature selection using the top 15% of features, the model's accuracy dropped to 80.6%. Class 4 again had the highest accuracy at 100%. Class 2 and Class 5 had the lowest accuracy at 58% and 56% respectively. Class 6 has a significant drop in accuracy, which prior to feature selection had a 95% prediction, and after feature selection has a 76% prediction. This shows that the features necessary to predict Class 6 are not included in the top 15% features. The top 15% of features include weight, gender both male and female, family_history_with_ obesity as 'no', always eating vegetables with meals (FCVC), and frequently eating food between meals. These features are the same top features from the model using the full dataset.

Decision Tree and Feature Selection with Gen-X & Boomers Dataset:

```
#Remove the age and class label column for Gen-X & Boomers:
data_genxb = genxboomers_df.iloc[:,1:43]


# View Class Labels for Gen-X & Boomers DF:
labels_genxb =  genxboomers_df['NObeyesdad']
labels_genxb.head()

0        Obesity_Type_I
1        Obesity_Type_I
2    Overweight_Level_I
3         Normal_Weight
4        Obesity_Type_I
Name: NObeyesdad, dtype: object

# Transform class label into numeric:
le_x = preprocessing.LabelEncoder()
genxb_labels = le_m.fit_transform(labels_genxb)
genxb_labels

array([1, 1, 3, 0, 1, 4, 4, 2, 1, 0, 4, 2, 4, 4, 3, 3, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2])

# Build training and test sets for Gen-X and Boomers:
xb_train, xb_test, xb_label_train, xb_label_test = train_test_split(data_genxb, ge
nxb_labels, test_size=0.2, random_state=1)

# Train Decision tree Classifier on the Training Data:
dt_xb = d_tree.fit(xb_train, xb_label_train)

# Predict on Gen-X and Boomers Test Set, View Performance, and Accuracy of Decisio
n Tree Model:
measure_performance(xb_test, xb_label_test, dt_xb, show_confussion_matrix=True, sh
ow_classification_report=True)

Accuracy:0.667

Classification Report
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       1.00      1.00      1.00         4
           3       0.00      0.00      0.00         1
           4       0.50      0.67      0.57         3

    accuracy                           0.67         9
   macro avg       0.38      0.42      0.39         9
weighted avg       0.61      0.67      0.63         9


Confussion Matrix
[[0 0 0 1]
 [0 4 0 0]
 [0 0 0 1]
 [0 0 1 2]]
```

```
# View the Accuracy of the Test and Training Sets:
print('Average Test Accuracy: ', d_tree.score(xb_test, xb_label_test))
print('Average Train Accuracy: ', d_tree.score(xb_train, xb_label_train))
```

```
Average Test Accuracy:  0.6666666666666666
Average Train Accuracy:  1.0
```

The classifier model for the Gen-X and Boomers dataset performed worse compared to all previous models. This model had the lowest accuracy score compared to the previous models, at 66.7%. This dataset is significantly smaller than the previous two datasets, so not all classes are represented in this model. This model was able to predict Class 1 at 100% accuracy, and Class 4 at 50%. This model was unable to predict Class 0 and Class 3.

```
# Perform feature selection for top 15% of Gen-X and Boomers DF:
fs_xb = feature_selection.SelectPercentile(feature_selection.chi2, percentile=15)
xb_train_fs = fs_xb.fit_transform(xb_train, xb_label_train)
```

```
# View the top 15% of the most important features for Millenails:
print(data_genxb.columns[fs_xb.get_support()].values)
```

```
['Weight' 'FCVC_Always' 'CH2O_More than 2 L' 'SCC_yes' 'FAF_2 or 4 days'
 'FAF_4 or 5 days' 'MTRANS_Public_Transportation']
```

```
# View scores for each top feature:
for i in range(len(data_genxb.columns.values)):
    if fs_xb.get_support()[i]:
        print(data_genxb.columns.values[i], '\t', fs_xb.scores_[i])
```

```
Weight       59.13977455691617
FCVC_Always        9.282051282051283
CH2O_More than 2 L       9.376068376068377
SCC_yes      31.0
FAF_2 or 4 days    15.333333333333334
FAF_4 or 5 days    15.0
MTRANS_Public_Transportation   31.0
```

```python
# Evaluate the Classifier with the top 15% feature set for Gen-X and Boomers DF:
d_tree.fit(xb_train_fs, xb_label_train)
xb_test_fs = fs_xb.transform(xb_test)
measure_performance(xb_test_fs, xb_label_test, d_tree, show_confussion_matrix=True
, show_classification_report=True)
```

```
Accuracy:0.667

Classification Report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         1
           1       1.00      0.50      0.67         4
           3       0.50      1.00      0.67         1
           4       0.60      1.00      0.75         3

    accuracy                           0.67         9
   macro avg       0.53      0.62      0.52         9
weighted avg       0.70      0.67      0.62         9


Confussion Matrix
[[0 0 1 0]
 [0 2 0 2]
 [0 0 1 0]
 [0 0 0 3]]
```

With the feature selection using the top 15% of features, the model maintained its accuracy at 66.7%. This model underperformed compared to all previous models and again, the model was unable to predict Class 0. The top 15% of features include weight and always eating vegetables with meals (FCVC) which are similar to full dataset, Gen-Z dataset, and Millennial dataset. Because the sample size is significantly lower, more data would be needed for this population to perform a more detailed and determine what key features affect the classification of obesity for the Gen-X and Boomers age group.