Exploratory Obesity Data and Classification via Machine Learning

Dinh Viet Tuan - 21041406

***Abstract* - Since 1980, obesity has become a major social and health problem that needs more and more attention. Because of this, new research comes out every day, including studies that look at obesity rates, especially the factors that matter and how to predict the problem when these factors are present. In this research, various classifying algorithms were used to estimate levels of obesity. Several machine-learning approaches were compared based on how well they met the evaluation criteria. The Random Forest method was 95.4% accurate when the right problem-specific characteristics were chosen, which is the highest accuracy.**

Keywords – Machine Learning; Classification; Linear Regression; Random Forest; Visualization; Bagging; Obesity; Data Exploratory.

INTRODUCTION

The World Health Organization (WHO) (OMS, 2016) defines obesity and overweight as excessive fat accumulation in certain body areas that can be dangerous to health. Since 1980, the number of individuals suffering from obesity has massively increased. In 2014, over 1.9 billion adults, 18 years and older, were overweight. Of these, over 600 million were obese. Obesity may be caused by biological risk factors such as the family background. Other risk variables include social, psychological, and nutrition issues. On either side, researchers offer other predictors of obesity, including "being an only child, family issues such as divorce, anxiety, and depressed mood" [1].

Based on the above assumptions and the literature found in several studies that looked at factors that affect obesity, several data mining methods have been created. Experts have researched the diseases and made digital tools to measure a person's level of obesity. However, these methods can only estimate the body mass index and do not consider essential things like a person's family history or how much time they spend on the condition. Based on it, the researchers believe an intelligent technology capable of detecting obesity levels in individuals more effectively is necessary [1].

The goal of this study was to use different types of machine learning to determine if a person is obese. The research is structured as follows: the relevant studies are presented in the second section. In the third section, the data set and its characteristics are described. In the fourth section, the application of categorization algorithms is discussed. The findings and comments are presented in the last section.

LITERATURE REVIEW

In recent years, many studies on obesity have been carried out with different methods and results. In different ways, researchers have identified many factors that influence obesity. Below are brief reviews of the

studies conducted using data mining techniques on data relevant to this health issue.

In the study of Adnan and Husain [2], information regarding childhood obesity was collected from parents, caregivers, and the children themselves. The author has found that parental education level, lifestyle and eating habits, and environmental factors are all risk factors for obesity. The authors used Naïve Bayes and Decision Tree methods and optimized parameters to predict obesity in children. As a result, they obtained 19 related parameters with a high degree of accuracy. The accuracy of the prediction is 75%.

Authors Adnan and Husain also have another related study[3] that predicts childhood obesity. This study attempts to show the advantages and disadvantages of different approaches, including Naïve Bayes, Decision Tree and Neural Networks. The goal is to provide parents with the information they need about obesity so that they can prevent childhood obesity.

MyHealthyKids [4] is a study by Adnan and Hussain that uses three integrated modules in US elementary schools to try to stop kids from becoming overweight. The model is based on Naïve Bayes, and the tests give about 73% accuracy. The system's three modules are obesity prediction, persuasion, and recipe suggestion for health maintenance.

Dugan et al. [5] studied obesity in children older than two years. Machine learning methods in this research include Naïve Bayes, ID3, J48, Random Tree and Random Forest. The results showed that ID3 was the most effective method, with 85% accuracy and 89% sensibility.

METHODOLOGY

A. Data Set

This journal contains statistics that may be used to estimate the rate of obesity in Peru, Colombia and Mexico.The people in the data range from 14 to 61 years old and have different eating habits, health conditions, and physical conditions, as previously stated. The information was gathered with the help of a digital survey that anonymous users filled out. The data is numerical and continuous. Therefore, it may be analyzed using classification, prediction, segmentation, and association algorithms [6].

Target Variable: NObeysedad

Data are listed from lowest to highest BMI, which includes types:

- Insufficient weight
- Normal Weight
- Overweight Level 1
- Overweight Level 2
- Obesity type 1
- Obesity type 2
- Obesity type 3

The 6 features related with eating habits are:

- FAVC: Consumption of high caloric food
- FCVC: Regular intake of vegetables
- NCP : Number of main meals
- CAEC:Regular intake of food between meals
- CH2O: Consumption of water
- CALC: Intake of alcohol

The 5 features related to Physical Conditioning:

- SCC: Monitoring Calorie
- FAF: Frequency of physical exercise per week

- MTRANS: Transportation
- TUE: Daily time spent using technology devices
- SMOKE: Smoking or not

And the 5 features of Responder Charater - istics:

- family_history_with_overweight: Family history of obesity
- Gender: Male and female
- Age: Float
- Height: Float, in meters

Weight: Float, in kilograms

After all the data has been gathered, it has been preprocessed so that it can be used for different data mining techniques.

B. Methods

In this study, Logistic Regression, Bagging and Random Forest are used to classify obesity levels.

1. Logistic Regression:

Logistic regression could also be utilised to handle classification challenges. Generally, logistic regression classifiers could use a linear combination of many input data or predictor variables as the sigmoid function argument. The result of the sigmoid function correlates to a number between 0 and 1. The median value represents the dividing line between class 1 and 0 representatives. Specifically, an output generating a result larger than 0.5 is classified as belonging to class 1. In contrast, if the output is less than 0.5, the matching input is classified as class 0 (Harrington, 2012).

2. Random Forest:

Leo Breiman proposed this approach based on a regression tree in 2001 [7]. The random forest method is often used for classification difficulties [8]. The Random Forest [9] is chosen because it was strong and could handle a wide range of characteristics with a small number of samples. As illustrated in Figure 1, each tree in the training data set is trained using the bootstrapping sample approach. This method searches for a random subset of variables to divide at each node. The input vector of each unit is given to the RF for classification, and each tree votes for a class. The RF was selected based on the target with the most votes. Unlike other models, it can handle massive amounts of input data. [10].

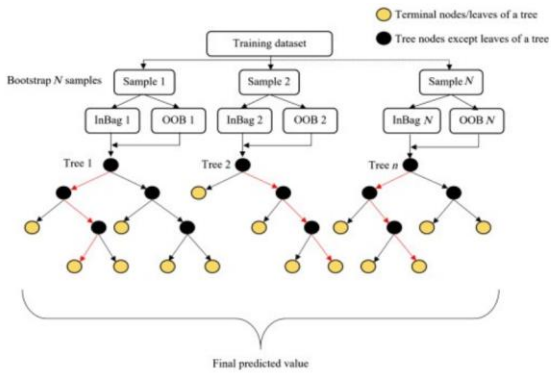

Figure 1: Structure of a random forest

ANALYSIS

A. Data Preparation

1. Import libraries

The research will use the libraries Pandas, NumPy,sklearn.preprocessing,Seaborn,matp -lotlib.pyplot,skicit-learn, ..

2. Calculate BMI

After all the data has been collected, the columns BMI will be generated to do exploratory data process. BMI caculated by using this equation:

$$BMI = \frac{weight}{(weight * height)}$$

(*See* ***Apendix 2***)

The results were compared with data provided by the WHO for levels of obesity.

3. Check shape of data and missing values

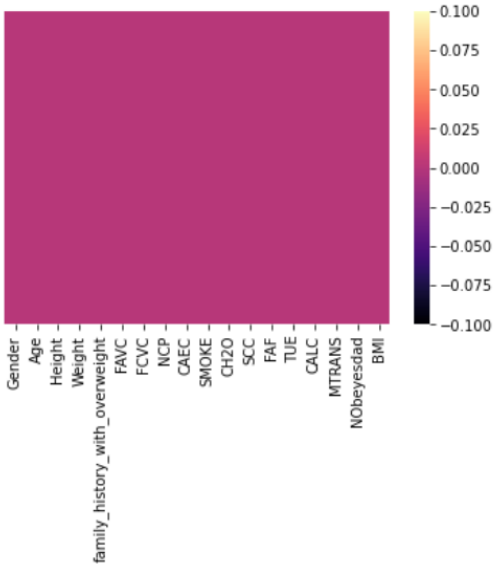

Figure 2: Checking Missing Values

(*See* ***Apendix 5***)

The .shape function accurately returned (2111,17) and the heatmap validated that no missing data exist.

B. Exploratory Data by Visualization

1. How do respondents react to questions with yes-or-no options?

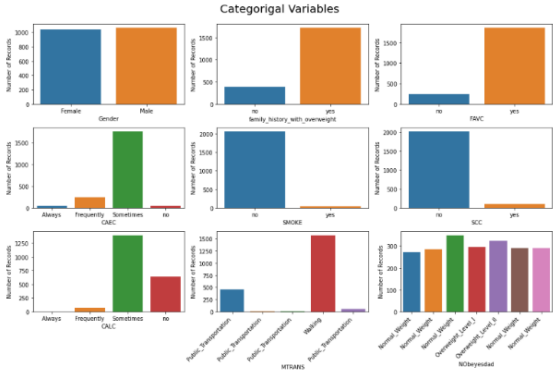

Figure 3: Categorical Variables Distribution

((See ***Apendix 12***)

Observing the distribution of variables is very important in data analysis. For example, less than 1% of people in the survey could be found to be smokers, because if smoking affected obesity, this would be a noticeable effect to pay attention to. The same is valid for determining how many calories people eat since less than 5% of respondents do it.

In contrast, most survey participants had a family history of obesity. They commonly consume high-calorie items; therefore, drawing inferences from these facts may be more accurate. Variations from the actual population may also result from these data.

2. What is the connection between BMI and other variables?

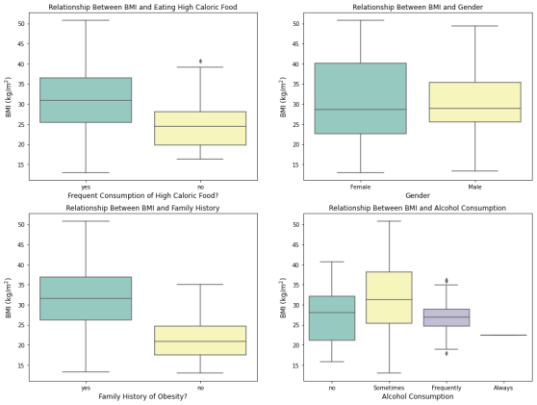

Figure 4: Relationship between Obesity and Factors

(*See* ***Apendix 8***)

It is helpful to make a chart that shows how categorical variables can change based on it and which factors have an effect.

The first subplot demonstrates that people who regularly eat high-calorie foods have a median BMI of about 6 kg/m2 higher than those who do not. This implies that calorie consumption likely leads to increased body fat.

Men and women had the same median BMI, but women's BMI was more spread out based

on the quartiles. This fits the earlier conclusion that women's weight varied more than men's.

As demonstrated by the third subplot, the median BMI of persons with a family history of obesity is higher by about 11 kg/m2.

The fourth subplot shows that there is not much of a link between alcoholism and BMI because people who drink alcohol often have the same median BMI as people who do not drink. Also, because only one person said "always," more information may be needed to answer this question.

3. How do a person's movement routine affect their BMI?

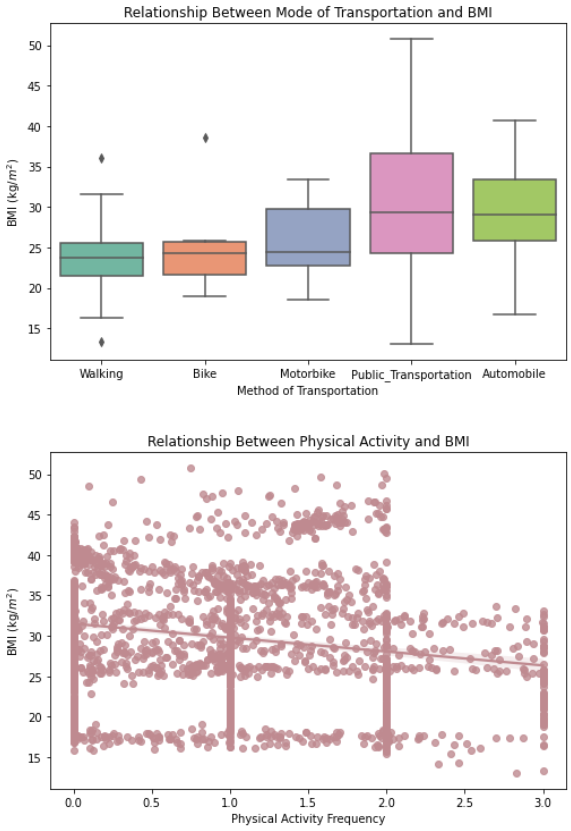

Figure 5: Relationship Between BMI and Movement Habits

(*See* ***Apendix 11***)

This graph illustrates the relationship between two factors linked to movement and BMI. The regression line of the first subplot shows that the number of times a person is physically active is related to their BMI in a negative way. However, several points are far from the regression line, which shows that the number of times a person is physically active may not be related.

The second subplot demonstrates the connection between various means of transportation and BMI. Putting them in this order shows that intense physical activity, like walking and riding, is linked to a lower BMI

C. Using Machine Learning

1. Correlation and Drop Variable

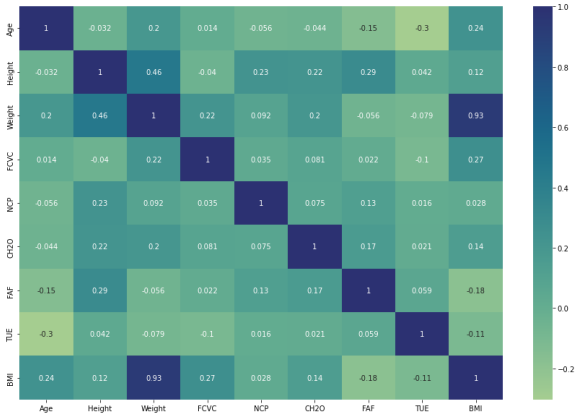

Figure 6: Correlation of Attributes

(*See* ***Apendix 7***)

Knowing that weight is so highly correlated (0.93/1) with obesity, “Weight” variable should be droped to help the model can learn from the other variables.

Some variables have correlated with the target variable showed is: Age, Height, FCV, CW and NMM.

2. Normalize Data Set

Since the ranges of the dataset's properties fluctuate, the data must be adjusted before it can be separated to run models. This may be problematic since a little change in one trait may not affect the other. Thus, ranges are normalized to the interval 0–1.

Moreover, critical thing is "obesity" variable will be converted to a binomial variable, with Not Obese as 0 and Obese as 1.

(See ***Appendix 18***)

3. Target Variable Distribution

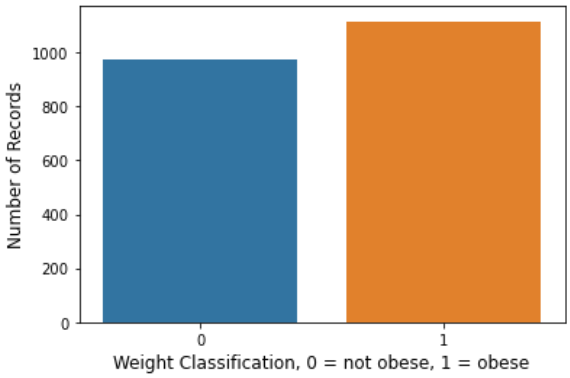

Figure 7: Target Variable Distribution

(*See* ***Apendix 15***)

In this dataset, there are about as many cases of obesity as there are cases of non-obesity. Then the target variable is a binary one, which will guide our further model selections.

4. Splitting test and training data

Before building the models, the dataset is split into training (75%) and testing (25%). The shape function is used to make sure the data is split correctly. (See ***Appendix 19***)

FINDINGS

1. Baseline classification accuracy

Calculating the baseline classification accuracy is preferable due to the simplest, most reasonable forecast. This gives us a great place to start thinking about how to make more accurate models, which will help us get a higher score.

Baseline Accuracy = 0.5457

(See ***Appendix 20***)

The result is a low accuracy of 54.57% for the baseline, so our next models should be more accurate than this.

2. Model 1: Logistic Regression and hyperparameter tuning(See ***Appendix 21, 22***)

The first model to be trained in the logistic regression model, which sorts variables into groups and predicts how likely they are to be in each group. The data set will be separated consistently using default parameters and random_state. The model is then fitted using data, and predictions and evaluations are performed. The model is then cross-validated using a usual 10-fold cross-validation process. The data is split into ten subsets, and each subset is set aside as test data to see how well the model fits the data.

The first accuracy of predictions of the logistic regression is 76.43% and cross-validation is 71%.

The cross-validation score of 71% was dramatically lower than the model's accuracy by about 5.43%. This suggests that our model may overfit and not reflect the actual relationship between variables. The model should be tuned using hyperparameters like GridSearchCV. GridSearchCV is regarded as one of the most precise algorithms owing to its extensive number of iterations, which traverse every hyperparameter value combination

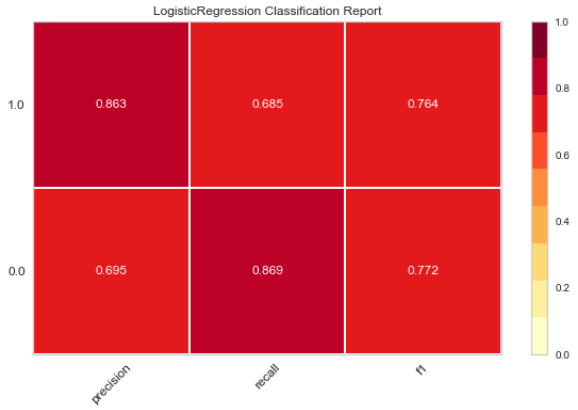

Figure 8: Logistic Regression Model Report

(*See* ***Apendix 23***)

The fact that our precision, recall and f1 is about 70-80 per cent, which shows that our model has been better. However, it is widely considered that it might be improved.

3. Model 2: Random Forest

N_estimator is the number of trees that need to built before taking the maximum voting or averages of predictions.

To determine which n_estimator should be used for the Random Forest model, graphs of n_estimators and their testing accuracy were constructed.

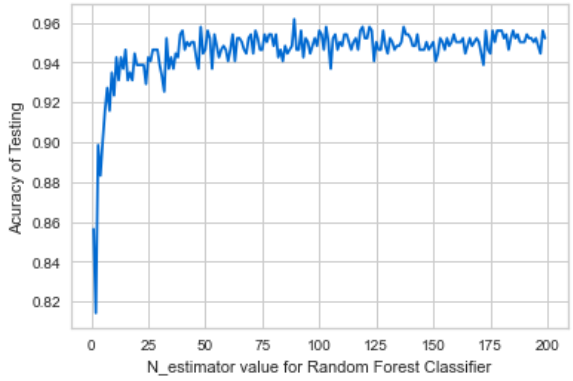

Figure 9: Number of trees

(*See* ***Apendix 24***)

This graph implies that a n_estimator of around 88 might help achieve high accuracy.

When a node is split, Max_features is the number of random subsets of features that will be looked at. Here is the Classification Report of Random Forest Model:

Classification Report:

	precision	recall	f1-score	support
0.0	0.95	0.94	0.95	236
1.0	0.95	0.96	0.96	286
accuracy			0.95	522
macro avg	0.95	0.95	0.95	522
weighted avg	0.95	0.95	0.95	522

Figure 8: Random Forest Report

(*See* ***Apendix 28***)

The Random Forest model is the best so far. The model has an accuracy of 95.4%, which is better than linear regression and the baseline method.(See ***Appendix 25)***

Cross-validation is 93.6%, not far from 95%, so this model may not be overfitting.

Moreover, precision is nearly equal to recall. It is about 95%

Furthermore, the F1 score is near 1, which is 0.96. This is a good result.

4. Most Influential Features

A simple bar graph from Random Forest Model to visualize most significant features

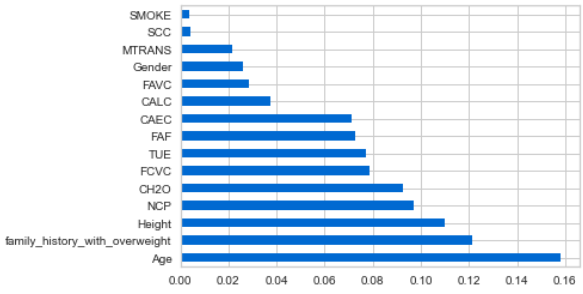

Figure 10: Most Influential Features

(*See* ***Apendix 26***)

This graph reveals that using the Random Forest model, "Age", "Family history", and "height" were the most effective predictors of obesity, which are 15.8%, 12.1% and 11%, respectively(See ***Appendix 27***). However, smoking and calorie intake monitoring were simply noise.

CONCLUSION

From the baseline accuracy of 54.57% to the Random Forest model's accuracy of 95.4%, which was reached by combining many models and changing things like n_estimators, it was a jump. Even though the model is very accurate, it is not overfitting because the cross-validation scores for each ensemble approach were all within 1% of the model's accuracy.

The estimates of each feature's importance show that *age*, *family history*, and *height* had the most effect on the above models. One of these models was also affected by how much food was eaten before meals, how often people worked out, and how many big meals they ate. Factors like *smoking*, *keeping track of calories*, *gender*, and mode of *transportation* were not always influential.

REFERENCES

[1] E.De-La-Hoz-Correa,F.E.Mendoza-Palechor, A. De-La-Hoz-Manotas, R.C. Morales-Ortega, and S. H. B. Adriana, "Obesity Level Estimation Software Based on Decision Trees," J. Comput. Sci., pp: 67-77, 2019.

[2] M.H.B.M. Adan, and W. Husain, "A hybrid approach using Naive Bayes and genetic algorithm for childhood obesity prediction", IEEE Xplore Press, pp: 281-285, 2012.

[3] M.H.B.M. Adnan, W. Husain, and F. Damanhoori, "A survey on utilization of data mining for childhood obesity prediction", IEEE Xplore Press, pp: 1–6, 2010.

[4] W. Husain, M. H.M. Adan, L.K. Ping, J. Poh, and L.K. Meng, "My healthy kids: Intelligent obesity intervention system for primary school children", The Society of Digital Information and Wireless Communication, pp: 627–633, 2013.

[5] T.M. Dugan, S. Mukhopadhyay, A.Carroll, and SDowns, "Machine learning techniques for prediction of early childhood obesity", Applied Clin. Inform., pp: 506-520, 2015.

[6] archive.ics.uci.edu. (n.d.). *Index of /ml/machine-learning-databases/00544*. [online].Available at: https://archive.ics.uci.edu/ml/machine-learning-databases/00544/.

[7] YI Qi, “Random forest for bioinformatics”, Springer, 2012, pp/307-323.

[8] P.M.Chakraborty Sounak, Khalilia Mohammed, “Predicting disease risks from highly imbalanced data using random forest,” vol. 11, no 1, p. 51, 2011.

[9] A.F in A.N.L Sarica, Alessia, Cerasa, Antonio; Quattrone, “Random Forest algorithm for the classification of neroimaging data in Alzheimer’s disease: A systematic review,” 2017.

[10] A. Hemmati-Sarapardeh, A. Larestani, M.Nait Amar, *Chapter 2 – Inteligent models*. 2020.

APPENDICES

Appendix 1: Import library and loading data

```
#Import libraries necessary for data analysis
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')


#Load data into dataframe df
df = pd.read_csv ('D:\LONDON MET\Data Analysis and
Visualization\CW\ObesityDataSet_raw_and_data_sinthetic.csv')
df = pd.DataFrame(df)
```

Appendix 2: Calculate BMI and checking shape of the data

```
#Body fat classification is determined by body mass index (BMI), so I calculated
BMI and added each individual's BMI as a new column
df['BMI'] = df['Weight']/(df['Height']**2)
```

Appendix 3: Check Duplicate Rows and Drop Duplicate

```
#Check how many duplicate rows there are
duplicate_data = df[df.duplicated()]
print(duplicate_data.shape)

#Drop duplicates
df = df.drop_duplicates(keep='last')
df.shape
```

Appendix 4: Check data again

```
#Check if the new column was calculated and placed correctly
df.head()
```

Result:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad	BMI
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transportation	Normal_Weight	24.386526
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transportation	Normal_Weight	24.238227
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transportation	Normal_Weight	23.765432
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Walking	Overweight_Level_I	26.851852
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transportation	Overweight_Level_II	28.342381

Appendix 5: Check missing values

```
#Check if there are any missing values

sns.heatmap(df.isnull(), cbar=True, yticklabels=False, cmap='magma')
```

Appendix 6: Checking info of the data set

```
# Checking info
df.info()
```

Result:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2087 entries, 0 to 2110
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          2087 non-null   object
 1   Age                             2087 non-null   float64
 2   Height                          2087 non-null   float64
 3   Weight                          2087 non-null   float64
 4   family_history_with_overweight  2087 non-null   object
 5   FAVC                            2087 non-null   object
 6   FCVC                            2087 non-null   float64
 7   NCP                             2087 non-null   float64
 8   CAEC                            2087 non-null   object
 9   SMOKE                           2087 non-null   object
 10  CH2O                            2087 non-null   float64
 11  SCC                             2087 non-null   object
 12  FAF                             2087 non-null   float64
 13  TUE                             2087 non-null   float64
 14  CALC                            2087 non-null   object
 15  MTRANS                          2087 non-null   object
 16  NObeyesdad                      2087 non-null   object
 17  BMI                             2087 non-null   float64
dtypes: float64(9), object(9)
memory usage: 309.8+ KB
```

```
### Note:
Eating Habit:
- `FAVC` : Frequent consumption of high caloric food
- `FCVC` : Frequency of consumption of vegetables
- `NCP` :  Number of main meals
- `CAEC`: Consumption of food between meals
- `CH20`: Consumption of water daily
- `CALC`: Consumption of alcohol

Physical condition:
- `SCC`: Calories consumption monitoring
- `FAF`: Physical activity frequency
- `TUE`: Time using technology devices
- `MTRANS`:Transportation used

Type of Data:
- Categorical: `Gender`,`family_history_with_overweight`, `FAVC`, `FCVC`, `NCP`, `CAEC`, `SMOKE`, `CH2O`, `SCC`, `FAF`, `TUE`, `CALC`, `MTRANS`, `NObeyesdad`.

- Numerical:  `Age`, `Height`, `BMI`, `Weight`

- Mix types of data: No

- Contain Error/Typo: No

- Blank or Null: No

- Various Data Type: String, Int, Float
```

Appendix 7: Checking Correlation and Plot HeatMap

```
#Correlation matrix
df.corr()

#Correlation heatmap
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),cmap="crest", annot = True)
```

Appendix 8: Describe Dataframe

```
# Describe Data Frame
```

```
df.describe()
```
Result:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	BMI
count	2087.000000	2087.000000	2087.000000	2087.000000	2087.000000	2087.000000	2087.000000	2087.000000	2087.000000
mean	24.353090	1.702674	86.858730	2.421466	2.701179	2.004749	1.012812	0.663035	29.765758
std	6.368801	0.093186	26.190847	0.534737	0.764614	0.608284	0.853475	0.608153	8.024934
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000	0.000000	12.998685
25%	19.915937	1.630178	66.000000	2.000000	2.697467	1.590922	0.124505	0.000000	24.368897
50%	22.847618	1.701584	83.101100	2.396265	3.000000	2.000000	1.000000	0.630866	28.896224
75%	26.000000	1.769491	108.015907	3.000000	3.000000	2.466193	1.678102	1.000000	36.095538
max	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000	2.000000	50.811753

Appendix 9: Check target varible distribution

```
#Check target Variable distribution:

df["NObeyesdad"].value_counts().to_frame()
```

Result:

	NObeyesdad
Obesity_Type_I	351
Obesity_Type_III	324
Obesity_Type_II	297
Overweight_Level_II	290
Normal_Weight	282
Overweight_Level_I	276
Insufficient_Weight	267

Appendix 10: What is the connection between BMI and other variables?

```
plt.figure(figsize=(16,12))

#subplot 1: high caloric food
plt.subplot(2, 2, 1)
sns.boxplot(x = "FAVC", y = 'BMI', order=["yes", "no"],data = df, palette =
"Set3").set_title('Relationship Between BMI and Eating High Caloric Food')
```

```
plt.xlabel("Frequent Consumption of High Caloric Food?", size=12)
plt.ylabel("BMI (kg/$m^2$)", size=12)

#subplot 2: gender
plt.subplot(2, 2, 2)
sns.boxplot(x = 'Gender', y = 'BMI', data = df, palette =
"Set3").set_title('Relationship Between BMI and Gender')
plt.xlabel("Gender", size=12)
plt.ylabel("BMI (kg/$m^2$)", size=12)

#subplot 3: family history
plt.subplot(2, 2, 3)
sns.boxplot(x = 'family_history_with_overweight', y = 'BMI', data = df, palette =
"Set3").set_title('Relationship Between BMI and Family History')
plt.xlabel("Family History of Obesity?", size=12)
plt.ylabel("BMI (kg/$m^2$)", size=12)

#subplot 4: alcohol consumption
plt.subplot(2, 2, 4)
sns.boxplot(x = 'CALC', y = 'BMI', data = df, palette =
"Set3").set_title('Relationship Between BMI and Alcohol Consumption')
plt.xlabel("Alcohol Consumption", size=12)
plt.ylabel("BMI (kg/$m^2$)", size=12)
plt.show()
```

Appendix 11: What is relationship Between BMI and Mode of Transportation?

```
# Relationship Between BMI and Mode of Transportation
plt.figure(figsize=(18,12))
#subplot 1: transportation and BMI
plt.subplot(2, 2, 1)
sns.boxplot(x = 'MTRANS', y = 'BMI', data = df,
order=["Walking","Bike","Motorbike","Public_Transportation", "Automobile"],
palette = "Set2").set_title('Relationship Between Mode of Transportation and
BMI')
plt.xlabel("Method of Transportation")
plt.ylabel("BMI (kg/$m^2$)")

#subplot 2: physical activity and BMI
plt.figure(figsize=(18,12))
plt.subplot(2, 2, 2)
sns.regplot(x=df.FAF, y=df.BMI, color='#bf8a90').set_title('Relationship Between
Physical Activity and BMI')
```

```
plt.xlabel("Physical Activity Frequency")
plt.ylabel("BMI (kg/$m^2$)")
```

Appendix 12: Check distribution of Catergorical Variables

```
# columns of interest
columns = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE',
           'SCC', 'CALC', 'MTRANS', 'NObeyesdad']

fig, ax = plt.subplots(3, 3, figsize=(18, 16))
for col, subplot in zip(columns, ax.flatten()):
    sns.countplot(df[col], ax=subplot)

    if col=="MTRANS":
        sns.countplot(df[col],ax=subplot)
        plt.xticks(rotation=45)
        subplot.yaxis.label.set_text("Number of Records")
    elif col=="NObeyesdad":
        sns.countplot(df[col],ax=subplot)
        plt.xticks(rotation = 45)
        subplot.yaxis.label.set_text("Number of Records")
    else:
        sns.countplot(df[col],ax=subplot)
        subplot.yaxis.label.set_text("Number of Records")

# show figure & plots
fig.suptitle("Categorigal Variables", fontsize=20)
plt.tight_layout(pad=7, w_pad=0.5, h_pad=1.5)
plt.show()
```

Appendix 13: Check distribution of Ordinal Variables

```
# columns of interest
columns = ["FCVC", "NCP", "CH2O", "FAF", "TUE"]

fig, ax = plt.subplots(1, 5, figsize=(15, 4))
for col, subplot in zip(columns, ax.flatten()):
    sns.countplot(df[col], ax=subplot)
    subplot.yaxis.label.set_text("Number of Records")

# show figure & plots
```

```
fig.suptitle("Ordinal Variables", fontsize=20)
plt.tight_layout(pad=5, w_pad=0.7, h_pad=0.5)
plt.show()
```

Appendix 14: Transform Data to numerical data

```
df['NObeyesdad'] = df['NObeyesdad'].replace("Overweight_Level_I", int("1"))
df['NObeyesdad'] = df['NObeyesdad'].replace("Overweight_Level_II", int("1"))
df['NObeyesdad'] = df['NObeyesdad'].replace("Overweight_Level_III", int("1"))
df['NObeyesdad'] = df['NObeyesdad'].replace("Insufficient_Weight", int("1"))
df['NObeyesdad'] = df['NObeyesdad'].replace("Normal_Weight", int("1"))

df['NObeyesdad'] = df['NObeyesdad'].replace("Obesity_Type_I", int("0"))
df['NObeyesdad'] = df['NObeyesdad'].replace("Obesity_Type_II", int("0"))
df['NObeyesdad'] = df['NObeyesdad'].replace("Obesity_Type_III", int("0"))

df['Gender'] = df['Gender'].replace("Female", int('0'))
df['Gender'] = df['Gender'].replace("Male", int('1'))

df['family_history_with_overweight'] =
df['family_history_with_overweight'].replace("yes", int('1'))
df['family_history_with_overweight'] =
df['family_history_with_overweight'].replace("no", int('0'))

df['FAVC'] = df['FAVC'].replace("yes", int('1'))
df['FAVC'] = df['FAVC'].replace("no", int('2'))

df['FCVC'] = df['FCVC'].replace("Never", int('1'))
df['FCVC'] = df['FCVC'].replace("Sometimes", int('2'))
df['FCVC'] = df['FCVC'].replace("Always", int('3'))

df['CAEC'] = df['CAEC'].replace("no", int('1'))
df['CAEC'] = df['CAEC'].replace("Sometimes", int('2'))
df['CAEC'] = df['CAEC'].replace("Frequently", int('3'))
df['CAEC'] = df['CAEC'].replace("Always", int('4'))

df['SMOKE'] = df['SMOKE'].replace("yes", int('1'))
df['SMOKE'] = df['SMOKE'].replace("no", int('2'))

df['SCC'] = df['SCC'].replace("yes", int('1'))
df['SCC'] = df['SCC'].replace("no", int('2'))

df['CALC'] = df['CALC'].replace("no", int('1'))
df['CALC'] = df['CALC'].replace("Sometimes", int('2'))
```

```
df['CALC'] = df['CALC'].replace("Frequently", int('3'))
df['CALC'] = df['CALC'].replace("Always", int('4'))

df['MTRANS'] = df['MTRANS'].replace("Automobile", int('1'))
df['MTRANS'] = df['MTRANS'].replace("Motorbike", int('2'))
df['MTRANS'] = df['MTRANS'].replace("Bike", int('3'))
df['MTRANS'] = df['MTRANS'].replace("Public_Transportation", int('4'))
df['MTRANS'] = df['MTRANS'].replace("Walking", int('5'))
```

Appendix 15: Distribution of the target variable via countplot

```
#Distribution of the target variable via countplot

sns.countplot(x='NObeyesdad', data=df)
plt.xlabel("Weight Classification, 0 = not obese, 1 = obese", size=12)
plt.ylabel("Number of Records", size=12)
plt.show()
```

Appendix 16: Drop the weight and BMI variable

```
#Drop the weight and BMI variable
df.drop(['Weight','BMI'],axis = 1, inplace = True)
```

Appendix 17: Import libraries for maching learning

```
#import libraries
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import
(classification_report,recall_score,precision_score,accuracy_score)
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
from yellowbrick.features import FeatureImportances
from sklearn import metrics
from yellowbrick.classifier import ClassificationReport
```

Appendix 18: Normalize Data

```
#Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
Data = scaler.fit_transform(df)
cols = ['Gender', 'Age', 'Height', 'family_history_with_overweight',
        'FAVC','FCVC','NCP','CAEC','SMOKE','CH2O','SCC', 'FAF','TUE','CALC',
        'MTRANS','NObeyesdad']
Data = pd.DataFrame(Data, columns=cols)
print(Data.head())
```

Result:

```
    Gender       Age    Height  family_history_with_overweight  FAVC  FCVC  \
0      0.0  0.148936  0.320755                             1.0   1.0   0.5
1      0.0  0.148936  0.132075                             1.0   1.0   1.0
2      1.0  0.191489  0.660377                             1.0   1.0   0.5
3      1.0  0.276596  0.660377                             0.0   1.0   1.0
4      1.0  0.170213  0.622642                             0.0   1.0   0.5

        NCP      CAEC  SMOKE  CH2O  SCC       FAF  TUE      CALC  MTRANS  \
0  0.666667  0.333333    1.0   0.5  1.0  0.000000  0.5  0.000000    0.75
1  0.666667  0.333333    0.0   1.0  0.0  1.000000  0.0  0.333333    0.75
2  0.666667  0.333333    1.0   0.5  1.0  0.666667  0.5  0.666667    0.75
3  0.666667  0.333333    1.0   0.5  1.0  0.666667  0.0  0.666667    1.00
4  0.000000  0.333333    1.0   0.5  1.0  0.000000  0.0  0.333333    0.75

   NObeyesdad
0         1.0
1         1.0
2         1.0
3         1.0
4         1.0
```

Appendix 19: Split Data to Training set and Test set

```
#Store the feature and target variable
X = Data.iloc[:, :-1]
y = Data.iloc[:, -1]

print(Data.shape)
print(X.shape)
print(y.shape)

#Partition the data into training and test sets (75/25)
X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size=0.25,random_state=24)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
Result:
 (2087, 16)
 (2087, 15)
 (2087,)
 (1565, 15)
 (522, 15)
 (1565,)
 (522,)
```

Appendix 20: Calculate the baseline classification

```
# Compute the baseline classification accuracy for X_train
dummy_classifier = DummyClassifier(strategy='most_frequent')
dummy_classifier.fit(X_train,y_train)
baseline_acc = dummy_classifier.score(X_test,y_test)
print("Baseline Accuracy = ", baseline_acc)
```

Appendix 21: Logistic Regression

```
#Logistic regression accuracy
logReg = LogisticRegression(random_state=42)
logReg.fit(X_train, y_train)
y_pred = logReg.predict(X_test)
print("Accuracy of predictions:", metrics.accuracy_score(y_test, y_pred))

#Cross-validation
cv1 = cross_val_score(logReg, X, y, scoring='accuracy', cv=10)
cv1 = pd.Series(cv1)
print("Cross-validation: ", cv1.mean())
```

Result:

```
Accuracy of predictions: 0.764367816091954
Cross-validation:  0.7100225432462275
```

Appendix 22: GridSearchCV

```
from sklearn.model_selection import RepeatedStratifiedKFold
#GridSearchCV
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l1', 'l2', 'elasticnet']
c_values = [150, 100, 75, 50, 10, 1.0, 0.1, 0.01, 0.001, 0.0001]
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=12, n_repeats=10, random_state=42)
grid_search = GridSearchCV(estimator=logReg, param_grid=grid, n_jobs=-1, cv=cv,
scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X, y)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

Result:

```
Best: 0.757830 using {'C': 1.0, 'penalty': 'l2', 'solver': 'liblinear'}
```

Appendix 23: Log Reg with GridSearch

```
#Logistic reg with grid search hyperparameters
```

```
gridLogReg = LogisticRegression(random_state = 42, penalty = "l2", C = 100,
solver = "liblinear")
gridLogReg.fit(X_train, y_train)
grid_pred = gridLogReg.predict(X_test)
print("Accuracy of predictions:", metrics.accuracy_score(y_test, grid_pred))
#cross-validation
gridcv = cross_val_score(gridLogReg, X, y, scoring='accuracy', cv=10)
gridcv = pd.Series(gridcv)
print("Cross-validation: ", gridcv.mean())

from yellowbrick.classifier import ClassificationReport
#Log reg classification report visualizer
visualizer = ClassificationReport(gridLogReg, size=(600, 400), color = '#bf8abf')
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.poof()
```

Appendix 24: Calculate Number of Tree

```
#Random forest n_estimator and testing accuracy
scoresrf =[]
for k in range(1, 200):
    rfc = RandomForestClassifier(n_estimators=k)
    rfc.fit(X_train, y_train)
    y_predrf = rfc.predict(X_test)
    scoresrf.append(accuracy_score(y_test, y_predrf))
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(range(1, 200), scoresrf, color = '#0069d1')
plt.xlabel('N_estimator value for Random Forest Classifier')
plt.ylabel('Acuracy of Testing')
```

Appendix 25: Random Forest

```
#Random Forest model
model_rf = RandomForestClassifier(n_estimators=135, max_features= 6,
random_state=42)
model_rf.fit(X_train, y_train)
predict_rf = model_rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, predict_rf)
print("Accuracy:", accuracy_rf)
```

```
#cross-validation
rfcv = cross_val_score(model_rf, X, y, scoring='accuracy', cv=10)
rfcv = pd.Series(rfcv)
print("Cross-validation: ", rfcv.mean())
```

Appendix 26: Finding Features Most Important

```
#Random forest feature importance
feat_importances = pd.Series(model_rf.feature_importances_, index=X.columns)
feat_importances.nlargest(20).plot(kind='barh',color = '#0069d1' )
```

Appendix 27: Top 3 important features

```
# Top 3 features for RandomForest
rimp=pd.DataFrame(zip(X_train.columns, model_rf.feature_importances_))
rfimp = (rimp.sort_values(1, ascending = False))
print("Three most important features: \n", rfimp.iloc[:3])
```

Result:

```
Three most important features:
                                  0         1
1                               Age  0.158304
3  family_history_with_overweight  0.121149
2                            Height  0.110173
```

Appendix 28: Classification Report

```
print('Classification Report:\n')
print(classification_report(y_test, y_predrf))
```

Classification Report:

	precision	recall	f1-score	support
0.0	0.95	0.94	0.95	236
1.0	0.95	0.96	0.96	286
accuracy			0.95	522
macro avg	0.95	0.95	0.95	522
weighted avg	0.95	0.95	0.95	522