Obesity Level Predictor

1. Descriptive Statistics, Simple Exploration and Data Cleaning

In [ ]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn import metrics
```

In [ ]:

```
# Reading the file
df = pd.read_csv('/Users/azamrahman/Desktop/Projects/Capstone/ObesityDataSet.csv

# Display the first 5 rows
df.head()
```

Out[ ]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes

In [ ]:

```
# Display descriptive statistics of each numerical attribute
df.describe()
```

Out[ ]:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF
count	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000	2111.000000
mean	24.312600	1.701677	86.586058	2.419043	2.685628	2.008011	1.010298
std	6.345968	0.093305	26.191172	0.533927	0.778039	0.612953	0.850592
min	14.000000	1.450000	39.000000	1.000000	1.000000	1.000000	0.000000
25%	19.947192	1.630000	65.473343	2.000000	2.658738	1.584812	0.124505
50%	22.777890	1.700499	83.000000	2.385502	3.000000	2.000000	1.000000
75%	26.000000	1.768464	107.430682	3.000000	3.000000	2.477420	1.666678
max	61.000000	1.980000	173.000000	3.000000	4.000000	3.000000	3.000000

In [ ]:

```
# Display the total null values in each attribute
df.isnull().sum()

# There are no missing values present in this dataset
```

Out[ ]:

```
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

In [ ]:

```
# Number of total duplicated rows
duprows = df.duplicated(subset = None, keep = 'first').sum()
print("There are", duprows, "duplicated rows" )

#Drop the duplicated rows
df = df.drop_duplicates()

# Re-count the total number of rows and display the type of data attributes
df.info()

# Total number of rows dropped from 2110 to 2087 rows
```

```
There are 24 duplicated rows
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2087 entries, 0 to 2110
Data columns (total 17 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
```

```
dtypes: float64(8), object(9)
memory usage: 293.5+ KB
```

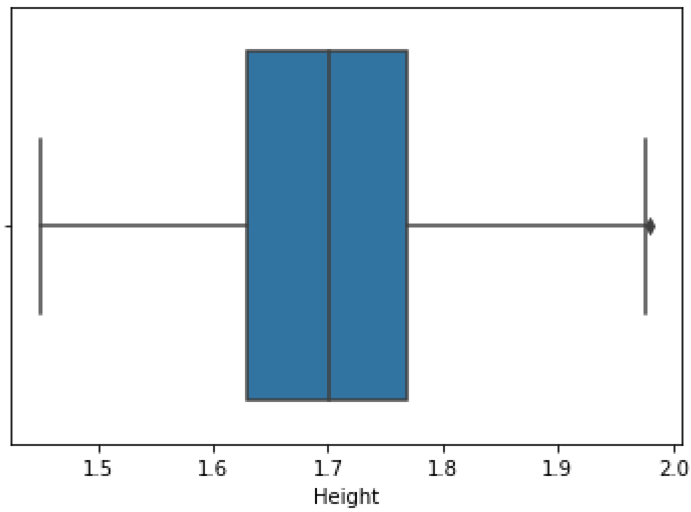
In [ ]:

```python
# Display the boxplot for Height to identify any outliers
sns.boxplot(df['Height'])
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/se
aborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword
arg: x. From version 0.12, the only valid positional argument will be `data`, an
d passing other arguments without an explicit keyword will result in an error or
misinterpretation.
  FutureWarning
```

Out[ ]:

```
<AxesSubplot:xlabel='Height'>
```



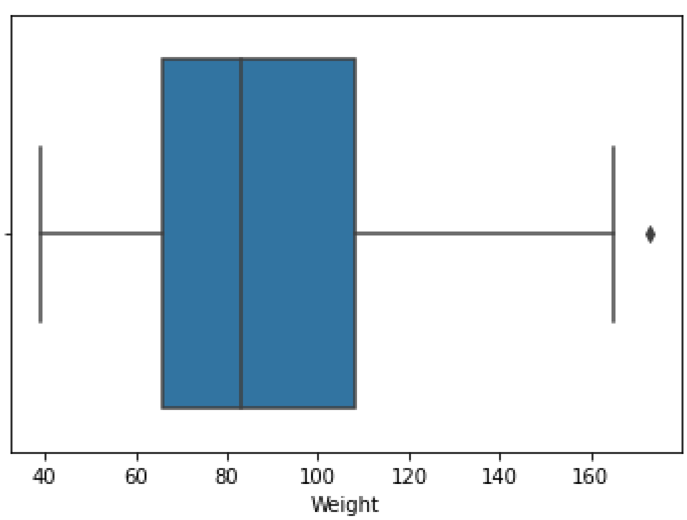
In [ ]:

```python
# Display the boxplot for Weight to identify any outliers
sns.boxplot(df['Weight'])
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/se
aborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword
arg: x. From version 0.12, the only valid positional argument will be `data`, an
d passing other arguments without an explicit keyword will result in an error or
misinterpretation.
  FutureWarning
```

Out[ ]:

```
<AxesSubplot:xlabel='Weight'>
```

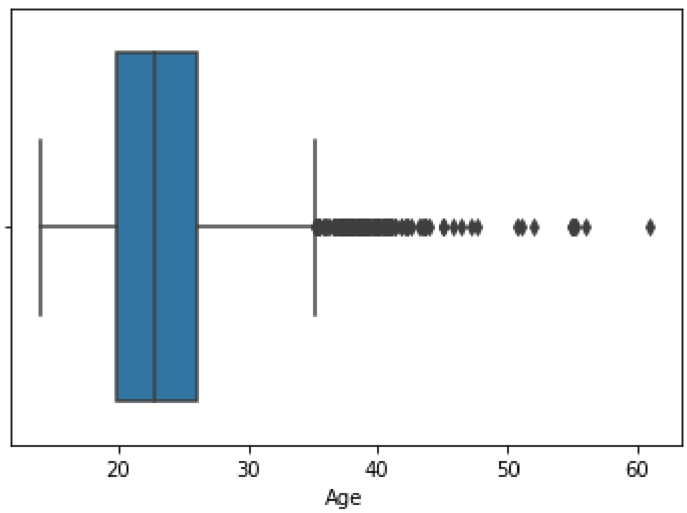


In [ ]:

```
# Display the boxplot for Age to identify any outliers
sns.boxplot(df['Age'])
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/se
aborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword
arg: x. From version 0.12, the only valid positional argument will be `data`, an
d passing other arguments without an explicit keyword will result in an error or
misinterpretation.
  FutureWarning
```

Out[ ]: <AxesSubplot:xlabel='Age'>



In [ ]:

```
# Display the count of outliers for each attribute
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).sum()

print(outliers)
```

```
/Users/azamrahman/Library/Python/3.7/lib/python/site-packages/ipykernel_launche
r.py:5: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons i
s deprecated and will raise ValueError in a future version.  Do `left, right = l
eft.align(right, axis=1, copy=False)` before e.g. `left == right`
  """
Age                               167
CAEC                                0
CALC                                0
CH2O                                0
FAF                                 0
FAVC                                0
FCVC                                0
Gender                              0
Height                              1
MTRANS                              0
NCP                               577
NObeyesdad                          0
SCC                                 0
SMOKE                               0
TUE                                 0
Weight                              1
```

```
family_history_with_overweight      0
dtype: int64
```

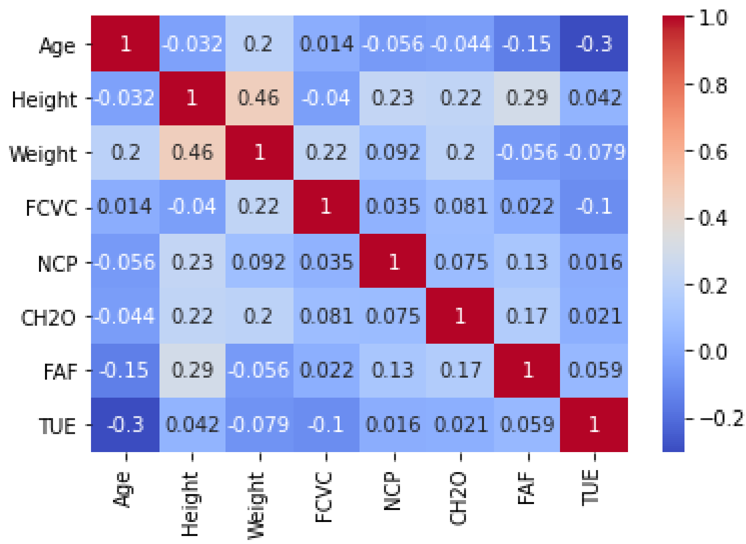
We can see that NCP has a very high count of outliers. Since the attributes acts as an ordinal variable, we can keep it for now and remove the attribute if needed when developing the model. The most important attributes are Height and Weight for clustering, and we can see that it only has one outlier which shows that it does not need manipulation at the moment.

2. Deep Data Exploration and Preparation for Cluster Analysis

In [ ]:

```
# plotting correlation heatmap
sns.heatmap(df.corr(),annot=True, cmap='coolwarm')

# displaying heatmap
plt.show()
```



The correlation heatmap indicates that none of the variables have a siginificant correlation to each other that needs to be dealt with. Height and Weight have the strongest positive correlation of 46%. Since we know both variables are crucial in determining the Obesity Level, it will remain in the dataset.

In [ ]:

```
# Create a copy of the original dataframe for data exploration
df_explore = df.copy()
# Compute BMI using the original formula and Weight and Height columns
df_explore["BMI"] = df_explore["Weight"] / df_explore["Height"]**2
# Display first 5 rows
df_explore.head()
```

Out[ ]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes

In [ ]:

```
# Apply logic to create another column named "Truth" to display if the BMI level

df_explore["Truth"] = np.where(((df_explore["BMI"] < 18.5) & (df_explore["NObeye
((df_explore["BMI"] >= 18.5) & (df_explore["BMI"] < 25) & (df_explore["NObeyesda
((df_explore["BMI"] >= 25) & (df_explore["BMI"] < 30) & ((df_explore["NObeyesdad
((df_explore["BMI"] >= 30) & (df_explore["BMI"] < 35) & (df_explore["NObeyesdad"
((df_explore["BMI"] >= 35) & (df_explore["BMI"] < 40) & (df_explore["NObeyesdad"
((df_explore["BMI"] >= 40) & (df_explore["NObeyesdad"] == "Obesity_Type_III")) ,
# Calculate the accuracy of BMI when compared to the class label.
Acc_of_BMI = (df_explore["Truth"].sum() / len(df_explore))*100
print("The Accuracy of BMI and Nobeyesdad is", Acc_of_BMI)
```

```
The Accuracy of BMI and Nobeyesdad is 95.11260182079539
```

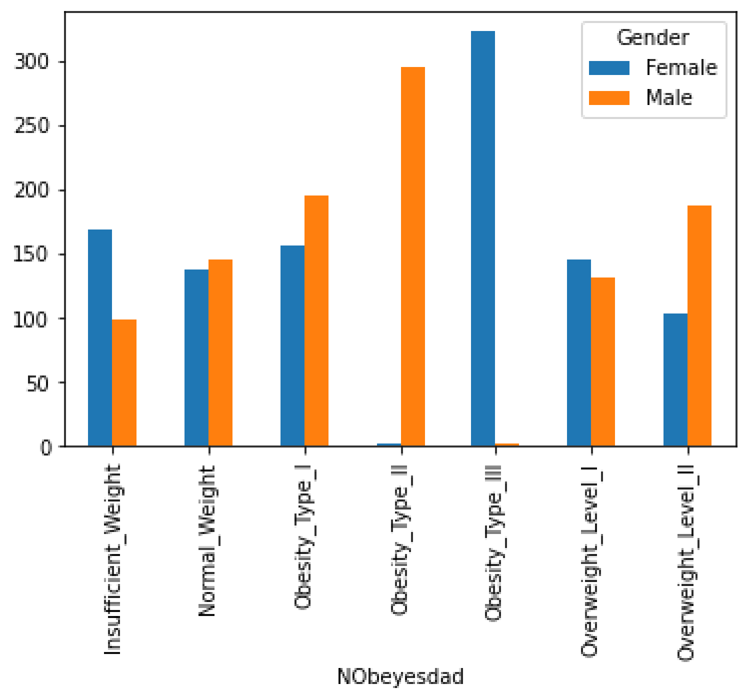
This shows that BMI is not directly correlated to the Obesity level since not all levels are True. This means that some of the other columsn come intp the equation into determining the proper diagnosis for Obesity Level. The class label is not completely determined by the BMI (Height vs Weight) but have a very accurate representation.

In [ ]:

```
# Break the Data into Gender and Class label to make sure there is accurate repr
df.groupby(["NObeyesdad", "Gender"]).size().unstack(level=1).plot(kind='bar')
```

Out[ ]:

```
&lt;AxesSubplot:xlabel='NObeyesdad'&gt;
```



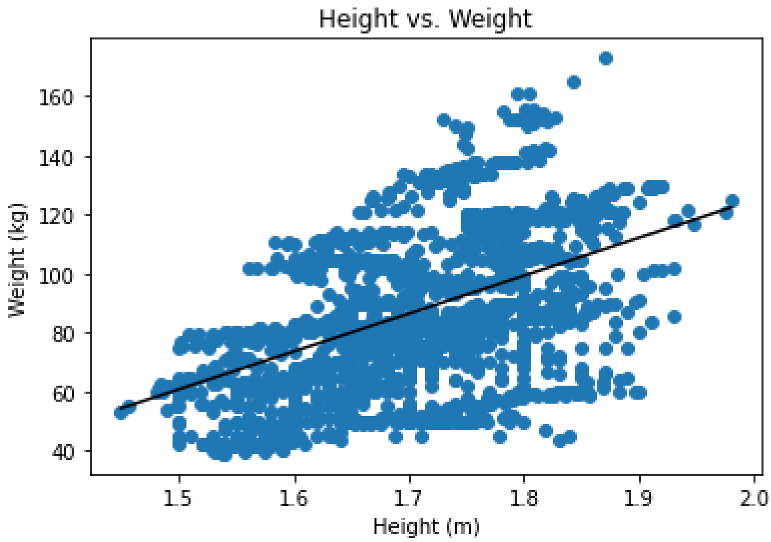
As we can see, most of the data is accurately represented for each gender except for Obesity_Type_I and II. Both categories lack severe representation of opposite genders. This could be an issue if other factors such as body fat percentage and muscle mass into play since it can distort the class label. On the other hand, we do have Height and Weight attributes that can contribute into lowering the errors of the other factors.

In [ ]:

```
# Plot Weight vs. Height to prepare for clustering. Having a visual can show us
plt.scatter(df_explore["Height"], df_explore["Weight"])
plt.xlabel("Height (m)")
plt.ylabel("Weight (kg)")
plt.title("Height vs. Weight")
# Add best line of fit
plt.plot(np.unique(df_explore["Height"]), np.poly1d(np.polyfit(df_explore["Heigh
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fde17b68e90>]
```



Completing the Cluster Analysis

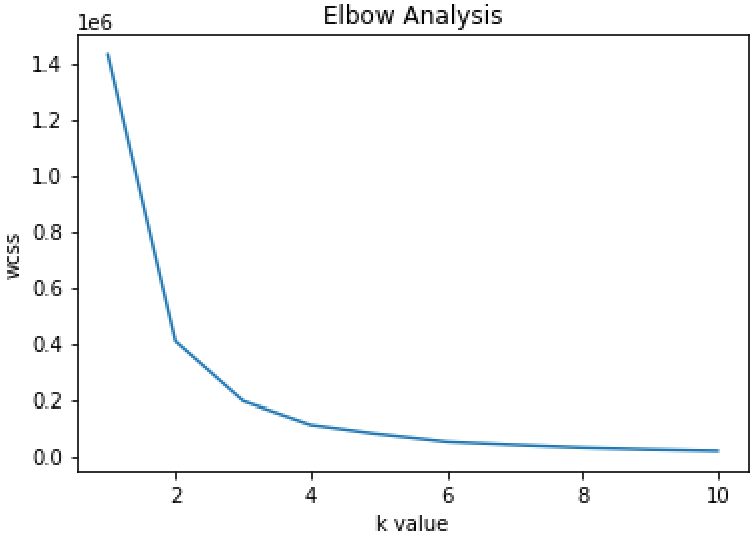
In [ ]:

```
# How to check the right number of clusters for annual income vs spending score:
x = df.iloc[:,[2,3]].values
wcss = []

for i in range(1,11):
    km = KMeans(n_clusters = i, init = 'k-means++', max_iter=300, n_init = 10, r
    km.fit(x)
    wcss.append(km.inertia_)

plt.plot(range(1,11), wcss)
plt.xlabel("k value")
plt.ylabel("wcss")
plt.title("Elbow Analysis")
plt.show()

# Best number of clusters is 4 as the curve smoothens after 4.
# Keep in mind that this is only for height vs weight
```



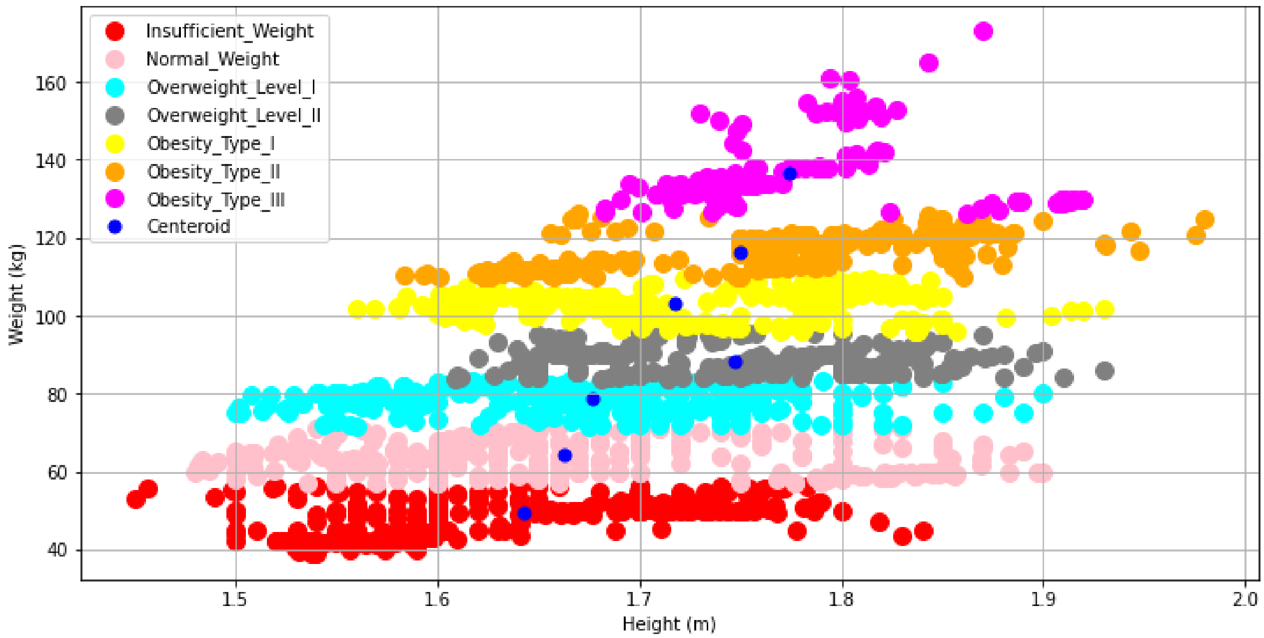
According to the elbow method, the best values of k is 4 clusters as the curve smoothens. However, we will still use 7 since we want to cluster based on each Obesity Level. On a more realistic note, having only 4 clusters that represent Underweight, Normal Weight, Overweight, and Obese would be the most effective way to develop the most accurate model based on clustering.

In [ ]:

```
# Number of Clusters:
K = 7

km = KMeans(n_clusters = K, init = 'k-means++', max_iter=300, n_init = 10, rando
y_means = km.fit_predict(x)

# Select random observations as centroids
fig=plt.figure(figsize=(12,6))
plt.scatter(x[y_means == 5, 0], x[y_means == 5, 1], s = 100, c = 'red', label =
plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s = 100, c = 'pink', label =
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s = 100, c = 'cyan', label =
plt.scatter(x[y_means == 6, 0], x[y_means == 6, 1], s = 100, c = 'grey', label =
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s = 100, c = 'yellow', label
plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s = 100, c = 'orange', label
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s = 100, c = 'magenta', labe
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:, 1], s = 50, c = 'bl
plt.xlabel("Height (m)")
plt.ylabel("Weight (kg)")
plt.legend()
plt.grid()
plt.show()
```



Comparing original Class Label and Clustering to measure Performance

In [ ]:

```
# Export the clustered results to a columns
df["Cluster"] = y_means
df.head()
```

Out[ ]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes

In [ ]:

```
# Apply logic to create another column named "Truth" to display if the Cluster f

df["Truth"] = np.where(((df["Cluster"] == 0) & (df["NObeyesdad"] == "Insufficien
((df["Cluster"] == 1) & (df["NObeyesdad"] == "Normal_Weight")) |
((df["Cluster"] == 2) & (df["NObeyesdad"] == "Overweight_Level_I")) |
((df["Cluster"] == 3) & (df["NObeyesdad"] == "Overweight_Level_II")) |
((df["Cluster"] == 4) & (df["NObeyesdad"] == "Obesity_Type_I")) |
((df["Cluster"] == 5) & (df["NObeyesdad"] == "Obesity_Type_II")) |
((df["Cluster"] == 6) & (df["NObeyesdad"] == "Obesity_Type_III")) , True, False)
# Calculate the accuracy of Cluster when compared to the class label.
Acc_of_Cluster = (df["Truth"].sum() / len(df))*100
print("The Accuracy of Cluster when compared to original class label is", Acc_of
```

```
The Accuracy of Cluster when compared to original class label is 8.2894106372783
9 %
```

When comparing the cluster to the original class label, we can see that the K-means cluster analysis did not do a good job predicting the level of obesity correctly using 7 clusters. This is perhaps the amount of clusters are too much, or the height and weight were not the best indicators to diagnose obesity. In the Final results submission. I will include a code that runs the K-means Algorithm with just 4 clusters to see if it can develop a more accurate class label. For example, the 4 clusters can be Underweight, Normal, Overweight, and Obese. Since the elbow method supported this method, it will be interesting to see how well it will do.

Data Preparation for Supervised Learning

In [ ]:

```
# Columns of all numerical variables:
num_cols = df._get_numeric_data().columns.tolist()
# Columns of all categorical variables:
cat_cols = df.select_dtypes(include=['object']).columns.tolist()
cat_cols.remove('NObeyesdad')
```

In [ ]:

```
# Creating a copy of the dataframe
df_onehot=df.copy()
# Converting  categorical variables to dummy variables to complete encoding
df_onehot = pd.get_dummies(df, columns=cat_cols, prefix = cat_cols)

df_onehot
```

Out[ ]:

	Age	Height	Weight	FCVC	NCP	CH2O	FAF	TUE	NObey
0	21.000000	1.620000	64.000000	2.0	3.0	2.000000	0.000000	1.000000	Normal_
1	21.000000	1.520000	56.000000	3.0	3.0	3.000000	3.000000	0.000000	Normal_
2	23.000000	1.800000	77.000000	2.0	3.0	2.000000	2.000000	1.000000	Normal_
3	27.000000	1.800000	87.000000	3.0	3.0	2.000000	2.000000	0.000000	Overweight_
4	22.000000	1.780000	89.800000	2.0	1.0	2.000000	0.000000	0.000000	Overweight_L
...	...	...	...	...	...	...	...	...	
2106	20.976842	1.710730	131.408528	3.0	3.0	1.728139	1.676269	0.906247	Obesity_T
2107	21.982942	1.748584	133.742943	3.0	3.0	2.005130	1.341390	0.599270	Obesity_T
2108	22.524036	1.752206	133.689352	3.0	3.0	2.054193	1.414209	0.646288	Obesity_T
2109	24.361936	1.739450	133.346641	3.0	3.0	2.852339	1.139107	0.586035	Obesity_T
2110	23.664709	1.738836	133.472641	3.0	3.0	2.863513	1.026452	0.714137	Obesity_T

2087 rows × 34 columns

Model Building for Random Forest to predict Obesity Level

In [ ]:

```
# Train test set split
class_col_name="NObeyesdad"

# Make sure that the class label is not included
one_hot_feature_names=df_onehot.columns[df_onehot.columns != class_col_name]
```

```
# Split dataset into training set and test set: 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(df_onehot.loc[:, one_hot_fea
```

In [ ]:

```
# Create Random Forest Model
clf = RandomForestClassifier(n_estimators=100)

# Train the model
clf.fit(X_train, y_train)

# Create the Predictions
y_pred = clf.predict(X_test)
```

Initial Performance Measures for Random Forest

In [ ]:

```
# Show the classification Report to view the results
print(classification_report(y_test, y_pred))
```

```
                     precision    recall  f1-score   support

Insufficient_Weight       0.97      0.95      0.96        82
      Normal_Weight       0.81      0.91      0.86        70
     Obesity_Type_I       0.99      0.97      0.98       122
    Obesity_Type_II       0.99      0.98      0.98        84
   Obesity_Type_III       0.99      1.00      0.99        98
 Overweight_Level_I       1.00      0.88      0.94        95
Overweight_Level_II       0.89      0.97      0.93        76

           accuracy                           0.95       627
          macro avg       0.95      0.95      0.95       627
       weighted avg       0.96      0.95      0.95       627
```

In our initial Results, we can see that the Random Forest model can predict the obesity level at an accuracy of 95%. However, we can see through the precision of Normal_Weight shows that it does have some trouble predicting the true positives of the obesity level.

In the final results, we will investigate feature engineering, overtraining, and the best possible parameters for the model with in detailed classification results.