I will do a case study on Obesity Data Set Obtained from Kaggle. The data contains information about 2112 Individuals and their habbits. The goal is to create a predictive model which will predict the Obesity Level

The flow of the case study is as below :

- Reading the data in python
- Defining the problem statement
- Identifying the Target variable
- Looking at the distribution of Target variable
- Basic Data exploration
- Rejecting useless columns
- Visual Exploratory Data Analysis for data distribution (Histogram and Barcharts)
- Feature Selection based on data distribution
- Outlier treatment
- Missing Values treatment
- Visual correlation analysis
- Statistical correlation analysis (Feature Selection)
- Converting data to numeric for ML
- Sampling and K-fold cross validation
- Trying multiple classification algorithms
- Selecting the best Model
- Deploying the best model in production

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Reading the data in python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandoc
import nbconvert
ObesityData= pd.read_csv("C:/Users/RITWIK/OneDrive/Desktop/IVY/Ivy ML/Project/ObesityDat

# Deleting Duplicates
ObesityData=ObesityData.drop_duplicates()
ObesityData.shape
```

Out[2]:
```
(2087, 17)
```

In [3]:
```
ObesityData.head()
```

Out[3]:

	Gender	Age	Height	Weight	family_history_with_overweight	Frequent consumption of high caloric food	Frequency of consumption of vegetables	Number of main meals	Consump of f betw m
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Someti
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Someti
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Someti
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Someti
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Someti

In [4]:
```python
ObesityData.info()
```

```
<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   Frequent consumption of high caloric food     2087 non-null   object 
 6   Frequency of consumption of vegetables        2087 non-null   float64
 7   Number of main meals                          2087 non-null   float64
 8    Consumption of food between meals            2087 non-null   object 
 9   Smoke                                         2087 non-null   object 
 10  Consumption of water daily                    2087 non-null   float64
 11  Calories consumption monitoring               2087 non-null   object 
 12  Physical activity frequency                   2087 non-null   float64
 13  Time using technology devices                 2087 non-null   float64
 14  Consumption of Alcohol                        2087 non-null   object 
 15  MTRANS                                        2087 non-null   object 
 16  Obese Data                                    2087 non-null   object 
dtypes: float64(8), object(9)
memory usage: 293.5+ KB
```

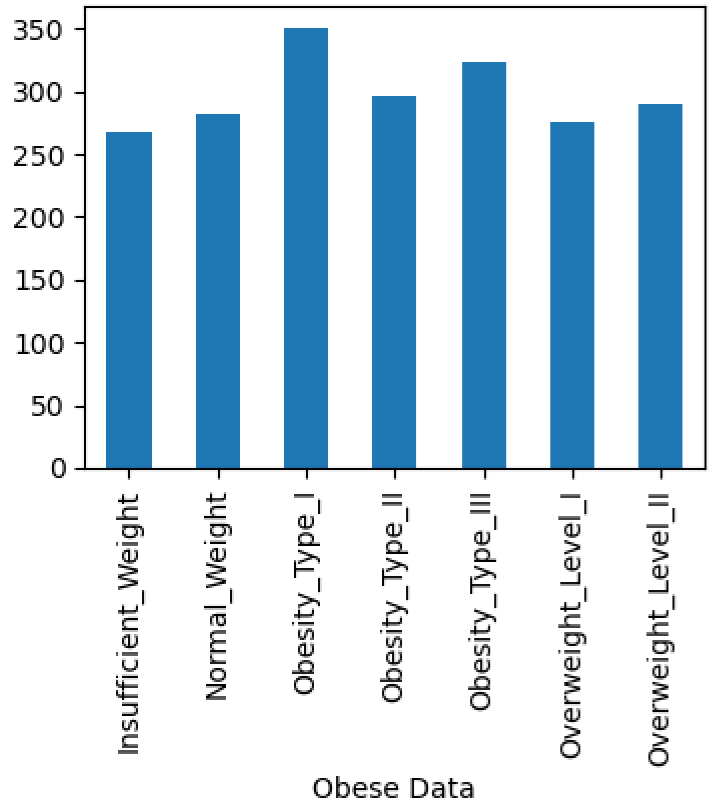
Defining The Problem Statement

Whether a person is Obese or not

- Target Variable : Obese Data
- Predictors " Gender, Smoke, Number of main meals etc

In [5]:
```python
%matplotlib inline
# Creating Bar chart as the Target variable is Categorical
GroupedData = ObesityData.groupby("Obese Data").size()
GroupedData.plot(kind="bar", figsize=(4,3))
```

Out[5]:
```
<Axes: xlabel='Obese Data'>
```



Basic Data Exploration

In [6]:
```
# Observing the summarized information of data
# Data types, Missing values based on number of non-null values Vs total rows etc.
# Remove those variables from data which have too many missing values (Missing Values >
# Remove Qualitative variables which cannot be used in Machine Learning
ObesityData.info()
```

```
<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   Frequent consumption of high caloric food 2087 non-null   object 
 6   Frequency of consumption of vegetables    2087 non-null   float64
 7   Number of main meals                      2087 non-null   float64
 8    Consumption of food between meals        2087 non-null   object 
 9   Smoke                                     2087 non-null   object 
 10  Consumption of water daily                2087 non-null   float64
 11  Calories consumption monitoring           2087 non-null   object 
 12  Physical activity frequency               2087 non-null   float64
 13  Time using technology devices             2087 non-null   float64
 14  Consumption of Alcohol                    2087 non-null   object 
 15  MTRANS                                    2087 non-null   object 
 16  Obese Data                                2087 non-null   object 
dtypes: float64(8), object(9)
memory usage: 293.5+ KB
```

In [7]:
```
ObesityData.describe(include="all")
```

Out[7]:

	Gender	Age	Height	Weight	family_history_with_overweight	Frequent consumption	Frequency consumptic

						of high caloric food	vegetabl
count	2087	2087.000000	2087.000000	2087.000000	2087	2087	2087.00000
unique	2	NaN	NaN	NaN	2	2	Na
top	Male	NaN	NaN	NaN	yes	yes	Na
freq	1052	NaN	NaN	NaN	1722	1844	Na
mean	NaN	24.353090	1.702674	86.858730	NaN	NaN	2.42146
std	NaN	6.368801	0.093186	26.190847	NaN	NaN	0.53473
min	NaN	14.000000	1.450000	39.000000	NaN	NaN	1.00000
25%	NaN	19.915937	1.630178	66.000000	NaN	NaN	2.00000
50%	NaN	22.847618	1.701584	83.101100	NaN	NaN	2.39626
75%	NaN	26.000000	1.769491	108.015907	NaN	NaN	3.00000
max	NaN	61.000000	1.980000	173.000000	NaN	NaN	3.00000

In [8]:

```
# Finging unique values for each column
# TO understand which column is categorical and which one is Continuous
# Typically if the numer of unique values are < 20 then the variable is likely to be a c
ObesityData.nunique()
```

Out[8]:

```
Gender                                     2
Age                                     1402
Height                                  1574
Weight                                  1525
family_history_with_overweight             2
Frequent consumption of high caloric food        2
Frequency of consumption of vegetables     810
Number of main meals                     635
 Consumption of food between meals         4
Smoke                                      2
Consumption of water daily              1268
Calories consumption monitoring            2
Physical activity frequency             1190
Time using technology devices           1129
Consumption of Alcohol                     4
MTRANS                                     5
Obese Data                                 7
dtype: int64
```

Basic Data Exploration

Based on the basic exploration above, you can now create a simple report of the data, noting down your observations regaring each column. Hence, creating a initial roadmap for further analysis.

The selected columns in this step are not final, further study will be done and then a final list will be created

Age , Height, Weight,'Frequency of consumption of vegetables','Number of main meals ','Consumption of water daily','Physical activity frequency','Time using technology devices ' are **continous variable**.

Gender, family_history_with_overweight ,Obese Data,Frequent consumption of high caloric food , Consumption of food between meals , Smoke, Calories consumption monitoring , Consumption of Alcohol, MTRANS are **categorical variable**.

Visual Exploratory Data Analysis

- Categorical variables: Bar plot
- Continuous variables: Histogram

Visualize distribution of all the Categorical Predictor variables in the data using bar plots

In [9]:
```
ObesityData.columns
```

Out[9]:
```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'Frequent consumption of high caloric food',
       'Frequency of consumption of vegetables', 'Number of main meals ',
       ' Consumption of food between meals', 'Smoke',
       'Consumption of water daily', 'Calories consumption monitoring',
       'Physical activity frequency', 'Time using technology devices ',
       'Consumption of Alcohol', 'MTRANS', 'Obese Data'],
      dtype='object')
```

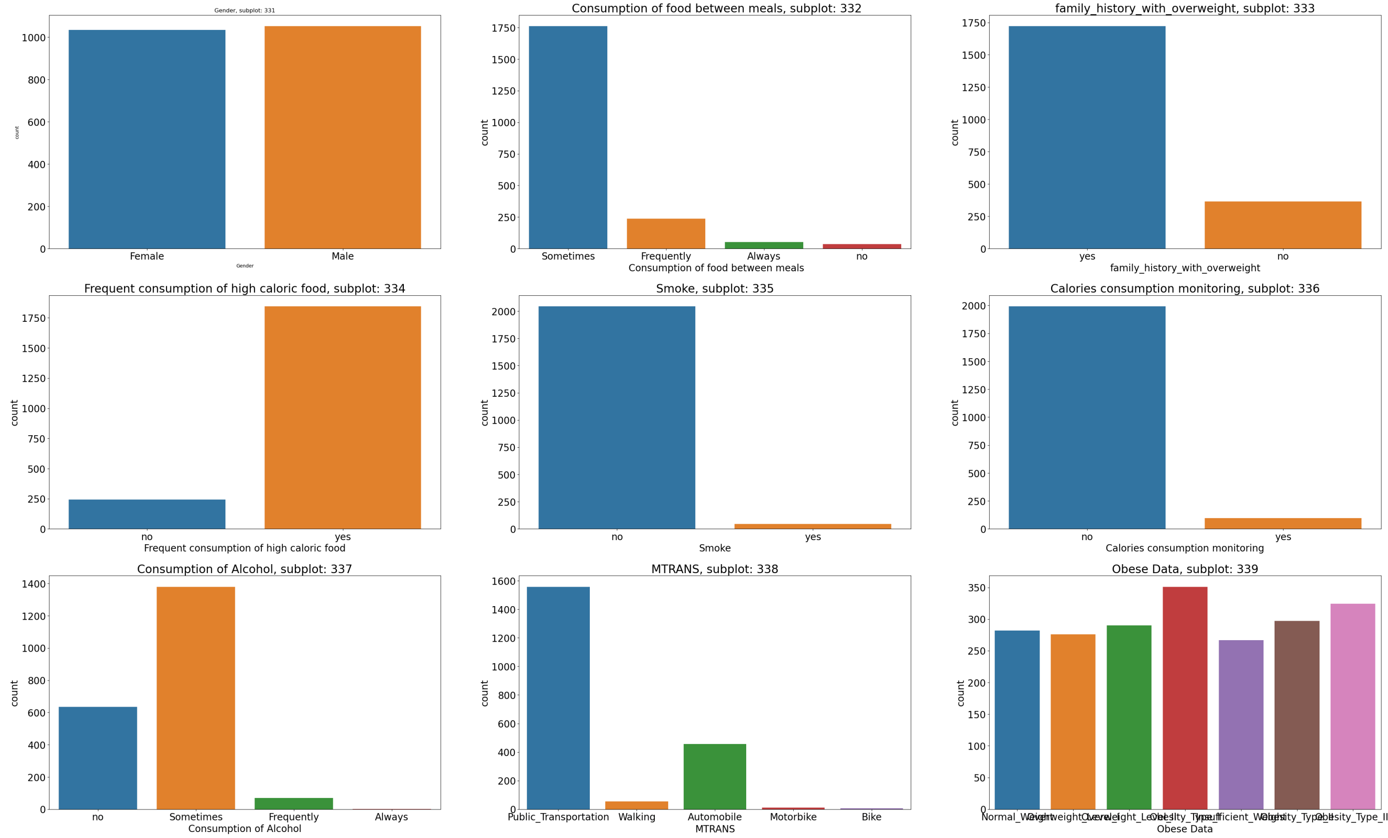
Plotting all the categorical columns in Box Plot

In [ ]:
```

```

In [10]:
```
cat = ObesityData.dtypes[ObesityData.dtypes=='object'].index
num = ObesityData.dtypes[ObesityData.dtypes!='object'].index
```

In [11]:
```
cat= ["Gender"," Consumption of food between meals",'family_history_with_overweight','Fr
```

In [12]:
```
#bivariate plot-political.knowledge
fig = plt.figure(figsize=(50,30))
c = 1
for i in cat:
    plt.subplot(3, 3, c)
    plt.title('{}, subplot: {}{}{}'.format(i, 3, 3, c))
    plt.xlabel(i)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    #sns.barplot(el_df[i],el_df['age'])
    sns.countplot(x= ObesityData[i], data=ObesityData)
    plt.rcParams.update({'font.size': 20})
    c = c + 1

plt.show()
```



Bar Charts Interpretation

These bar charts represent the frequencies of each category in the Y-axis and the category names in the X-axis.

In the ideal bar chart each category has comparable frequency. Hence, there are enough rows for each category in the data for the ML algorithm to learn.

If there is a column which shows too skewed distribution where there is only one dominant bar and the other categories are present in very low numbers. These kind of columns may not be very helpful in machine learning. We confirm this in the correlation analysis section and take a final call to select or reject the column.

In this data, all the categorical columns except have satisfactory distribution to be considered for machine learning.

Selected Categorical Variables: All the categorical variables are selected for further analysis.

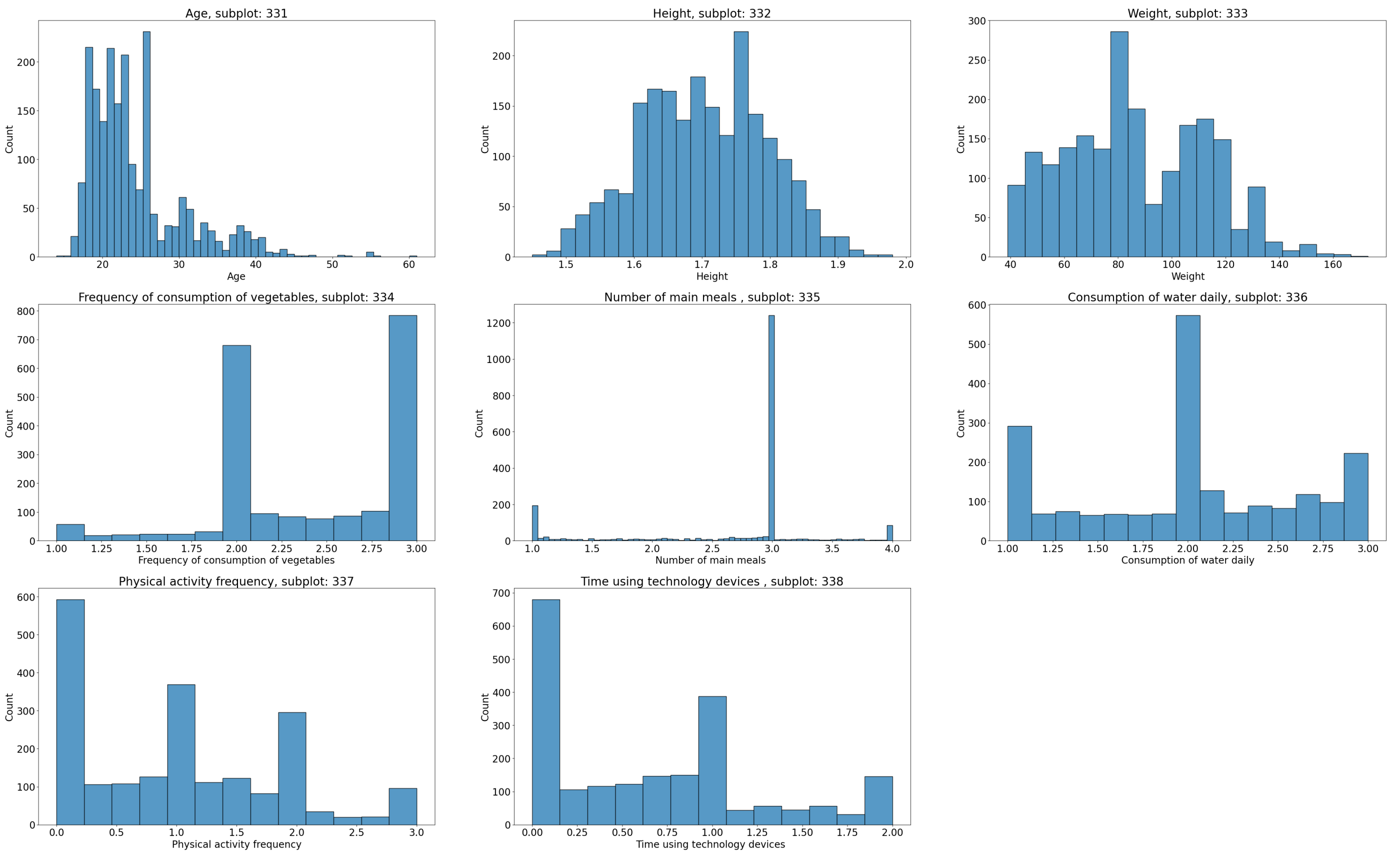
Converting Target Variable into Numeric

In [13]:
```
#ObesityData['Obese Data'].replace({'Normal_Weight':0, 'Overweight_Level_I':1,'Overweigh
```

Visualize distribution of all the Continuous Predictor variables in the data using histograms

In [14]:
```
#bivariate plot-political.knowledge using Seaborn lib
fig = plt.figure(figsize=(50,30))
c = 1
for i in num:
```

```
    plt.subplot(3, 3, c)
    plt.title('{}, subplot: {}{}{}'.format(i, 3, 3, c))
    plt.xlabel(i)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    #sns.barplot(el_df[i],el_df['age'])
    sns.histplot(x= ObesityData[i], data=ObesityData)
    plt.rcParams.update({'font.size': 20})
    c = c + 1

plt.show()
```



In [15]:

```
# Plotting histograms of multiple columns together
# Observe that ApplicantIncome and CoapplicantIncome has outliers
#ObesityData.hist(['Age','Height', 'Weight','Frequency of consumption of vegetables','Nu
```

Histogram Interpretation

Histograms shows us the data distribution for a single continuous variable.

The X-axis shows the range of values and Y-axis represent the number of values in that range.

The ideal outcome for histogram is a bell curve or slightly skewed bell curve. If there is too much skewness, then outlier treatment should be done and the column should be re-examined, if that also does not solve the problem then only reject the column.

In [16]:

```
# Checking Null Values
ObesityData.isnull().sum()
```

Out[16]:

```
Gender                                    0
Age                                       0
Height                                    0
Weight                                    0
family_history_with_overweight            0
Frequent consumption of high caloric food 0
```

```
Frequency of consumption of vegetables       0
Number of main meals                         0
 Consumption of food between meals           0
Smoke                                        0
Consumption of water daily                   0
Calories consumption monitoring              0
Physical activity frequency                  0
Time using technology devices                0
Consumption of Alcohol                       0
MTRANS                                       0
Obese Data                                   0
dtype: int64
```

Feature Selection

Now its time to finally choose the best columns(Features) which are correlated to the Target variable. This can be done directly by measuring the correlation values or ANOVA/Chi-Square tests. However, it is always helpful to visualize the relation between the Target variable and each of the predictors to get a better sense of data.

I have listed below the techniques used for visualizing relationship between two variables as well as measuring the strength statistically.

Visual exploration of relationship between variables

- Continuous Vs Continuous ---- Scatter Plot
- Categorical Vs Continuous---- Box Plot
- Categorical Vs Categorical---- Grouped Bar Plots

Statistical measurement of relationship strength between variables

- Continuous Vs Continuous ---- Correlation matrix
- Categorical Vs Continuous---- ANOVA test
- Categorical Vs Categorical--- Chi-Square test

In this case study the Target variable is categorical, hence below two scenarios will be present

- Categorical Target Variable Vs Continuous Predictor
- Categorical Target Variable Vs Categorical Predictor

Relationship exploration: Categorical Vs Continuous -- Box Plots

When the target variable is Categorical and the predictor variable is Continuous we analyze the relation using bar plots/Boxplots and measure the strength of relation using Anova test

In [17]:
```
cat
```

Out[17]:
```
['Gender',
```

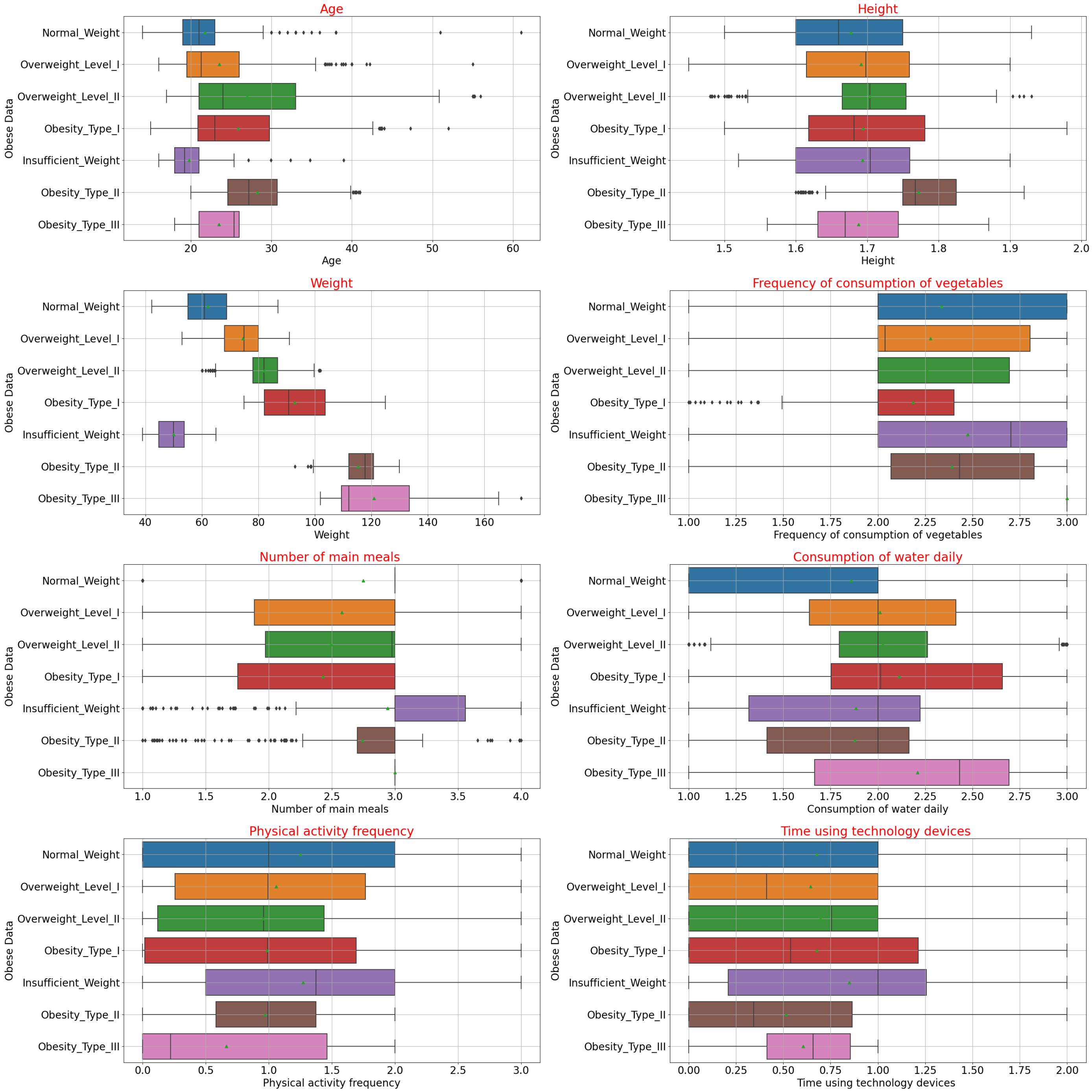
```
 ' Consumption of food between meals',
 'family_history_with_overweight',
 'Frequent consumption of high caloric food',
 'Smoke',
 'Calories consumption monitoring',
 'Consumption of Alcohol',
 'MTRANS',
 'Obese Data']
```

In [18]:
```
num
```

Out[18]:
```
Index(['Age', 'Height', 'Weight', 'Frequency of consumption of vegetables',
       'Number of main meals ', 'Consumption of water daily',
       'Physical activity frequency', 'Time using technology devices '],
      dtype='object')
```

In [19]:
```
ContinuousColsList=['Age','Height', 'Weight','Frequency of consumption of vegetables','N
```

In [30]:
```
#**Check for Box plots, Correlation plots for the continuous columns**
import matplotlib.pyplot as plt
import seaborn as sns
data_plot=ContinuousColsList
fig=plt.figure(figsize=(30,30))
for i in range(0,len(ContinuousColsList)):
    ax=fig.add_subplot(4,2,i+1)
    sns.boxplot(x=data_plot[i],y='Obese Data', data=ObesityData,showmeans=True, orient="
    ax.set_title(ContinuousColsList[i],color='Red')
    plt.grid()
plt.tight_layout()
```



Box-Plots interpretation

What should you look for in these box plots?

These plots gives an idea about the data distribution of continuous predictor in the Y-axis for each of the category in the X-Axis.

If the distribution looks similar for each category(Boxes are in the same line), that means the the continuous variable has NO effect on the target variable. Hence, the variables are not correlated to each other

The other chart exhibit opposite characteristics. Means the the data distribution is different(the boxes are not in same line!) for each category of survival. It hints that these variables might be correlated with Survived.

We confirm this by looking at the results of ANOVA test below

Statistical Feature Selection (Categorical Vs Continuous) using ANOVA test

Analysis of variance(ANOVA) is performed to check if there is any relationship between the given continuous and categorical variable

- Assumption(H0): There is NO relation between the given variables (i.e. The average(mean) values of the numeric Predictor variable is same for all the groups in the categorical Target variable)
- ANOVA Test result: Probability of H0 being true

In [19]:
```
# Defining a function to find the statistical relationship with all the categorical vari
def FunctionAnova(inpData, TargetVariable, ContinuousPredictorList):
    from scipy.stats import f_oneway

    # Creating an empty list of final selected predictors
    SelectedPredictors=[]

    print('##### ANOVA Results ##### \n')
    for predictor in ContinuousPredictorList:
        CategoryGroupLists=inpData.groupby(TargetVariable)[predictor].apply(list)
        AnovaResults = f_oneway(*CategoryGroupLists)

        # If the ANOVA P-Value is <0.05, that means we reject H0
        if (AnovaResults[1] < 0.05):
            print(predictor, 'is correlated with', TargetVariable, '| P-Value:', AnovaRe
            SelectedPredictors.append(predictor)
        else:
            print(predictor, 'is NOT correlated with', TargetVariable, '| P-Value:', Ano

    return(SelectedPredictors)
```

In [20]:
```
# Calling the function to check which categorical variables are correlated with target
ContinuousVariables=['Age','Height', 'Weight','Frequency of consumption of vegetables','
FunctionAnova(inpData=ObesityData, TargetVariable='Obese Data', ContinuousPredictorList=
```

```
##### ANOVA Results #####

Age is correlated with Obese Data | P-Value: 3.246861907985217e-86
Height is correlated with Obese Data | P-Value: 2.5185012901787443e-43
Weight is correlated with Obese Data | P-Value: 0.0
Frequency of consumption of vegetables is correlated with Obese Data | P-Value: 3.796507
103383786e-121
Number of main meals  is correlated with Obese Data | P-Value: 7.1320021657369e-31
Consumption of water daily is correlated with Obese Data | P-Value: 4.297246743274628e-1
7
Physical activity frequency is correlated with Obese Data | P-Value: 1.1554196575987729e
-20
Time using technology devices  is correlated with Obese Data | P-Value: 1.77238032713701
e-08
```

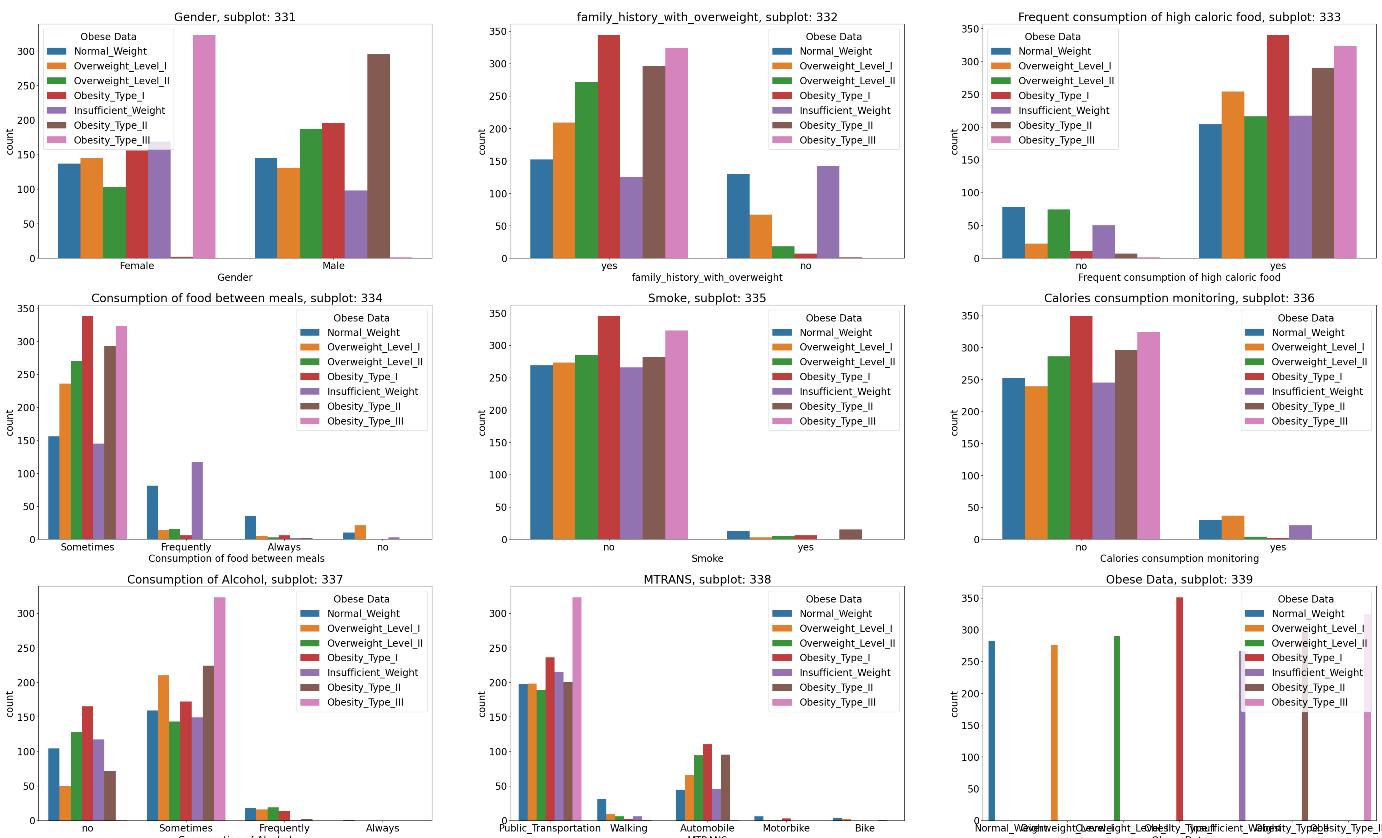
Out[20]:
```
['Age',
 'Height',
 'Weight',
 'Frequency of consumption of vegetables',
 'Number of main meals ',
 'Consumption of water daily',
 'Physical activity frequency',
 'Time using technology devices ']
```

Relationship exploration: Categorical Vs

Categorical -- Grouped Bar Charts

When the target variable is Categorical and the predictor is also Categorical then we explore the correlation between them visually using barplots and statistically using Chi-square test

In [21]:
```
#bivariate plot-political.knowledge
fig = plt.figure(figsize=(50,30))
c = 1
for i in cat:
    plt.subplot(3, 3, c)
    plt.title('{}, subplot: {}{}{}'.format(i, 3, 3, c))
    plt.xlabel(i)
    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    #sns.barplot(el_df[i],el_df['age'])
    sns.countplot(x= ObesityData[i], data=ObesityData, hue = "Obese Data")
    plt.rcParams.update({'font.size': 20})
    c = c + 1

plt.show()
```



Grouped Bar charts Interpretation

What to look for in these grouped bar charts?

These grouped bar charts show the frequency in the Y-Axis and the category in the X-Axis. If the ratio of bars is similar across all categories, then the two columns are not correlated.

Here we can see all the categorical variables the bar chart is different. Hence the two columns are corellated to each other. We confirm this analysis in below section by using Chi-Square Tests.

Statistical Feature Selection (Categorical Vs Categorical) using Chi-Square Test

Chi-Square test is conducted to check the correlation between two categorical variables

- Assumption(H0): The two columns are NOT related to each other
- Result of Chi-Sq Test: The Probability of H0 being True

In [22]:

```
# Writing a function to find the correlation of all categorical variables with the Targe
def FunctionChisq(inpData, TargetVariable, CategoricalVariablesList):
    from scipy.stats import chi2_contingency

    # Creating an empty list of final selected predictors
    SelectedPredictors=[]

    for predictor in CategoricalVariablesList:
        CrossTabResult=pd.crosstab(index=inpData[TargetVariable], columns=inpData[predic
        ChiSqResult = chi2_contingency(CrossTabResult)

        # If the ChiSq P-Value is <0.05, that means we reject H0
        if (ChiSqResult[1] < 0.05):
            print(predictor, 'is correlated with', TargetVariable, '| P-Value:', ChiSqRe
            SelectedPredictors.append(predictor)
        else:
            print(predictor, 'is NOT correlated with', TargetVariable, '| P-Value:', Chi

    return(SelectedPredictors)
```

In [23]:

```
CategoricalVariables=["Gender"," Consumption of food between meals",'family_history_with

# Calling the function
FunctionChisq(inpData=ObesityData,
              TargetVariable='Obese Data',
              CategoricalVariablesList= CategoricalVariables)
```

```
Gender is correlated with Obese Data | P-Value: 9.357967638720868e-139
 Consumption of food between meals is correlated with Obese Data | P-Value: 6.3012578973
18031e-142
family_history_with_overweight is correlated with Obese Data | P-Value: 3.52415610870361
1e-130
Frequent consumption of high caloric food is correlated with Obese Data | P-Value: 4.089
7083168071957e-47
Smoke is correlated with Obese Data | P-Value: 1.7397906082796266e-05
Calories consumption monitoring is correlated with Obese Data | P-Value: 3.3385130651942
555e-25
Consumption of Alcohol is correlated with Obese Data | P-Value: 2.22093967310428e-60
MTRANS is correlated with Obese Data | P-Value: 3.3319887895526285e-47
```

Out[23]:

```
['Gender',
 ' Consumption of food between meals',
 'family_history_with_overweight',
 'Frequent consumption of high caloric food',
 'Smoke',
 'Calories consumption monitoring',
 'Consumption of Alcohol',
 'MTRANS']
```

Finally selected Categorical variables:

['Gender', ' Consumption of food between meals', 'family_history_with_overweight', 'Frequent consumption of high caloric food', 'Smoke', 'Calories consumption monitoring', 'Consumption of Alcohol', 'MTRANS']

Selecting final predictors for Machine Learning

Based on the above tests, selecting the final columns for machine learning.

For this Data, all columns are selected

In [24]:
```
ObesityData.columns
```

Out[24]:
```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'Frequent consumption of high caloric food',
       'Frequency of consumption of vegetables', 'Number of main meals ',
       ' Consumption of food between meals', 'Smoke',
       'Consumption of water daily', 'Calories consumption monitoring',
       'Physical activity frequency', 'Time using technology devices ',
       'Consumption of Alcohol', 'MTRANS', 'Obese Data'],
      dtype='object')
```

In [25]:
```
SelectedColumns =['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'Frequent consumption of high caloric food',
       'Frequency of consumption of vegetables', 'Number of main meals ',
       ' Consumption of food between meals', 'Smoke',
       'Consumption of water daily', 'Calories consumption monitoring',
       'Physical activity frequency', 'Time using technology devices ',
       'Consumption of Alcohol', 'MTRANS']
```

In [26]:
```
DataForML = ObesityData[SelectedColumns]
DataForML.head()
```

Out[26]:

	Gender	Age	Height	Weight	family_history_with_overweight	Frequent consumption of high caloric food	Frequency of consumption of vegetables	Number of main meals	Consump of f betw m
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Someti
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Someti
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Someti
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Someti
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Someti

Data Pre-processing for Machine Learning

List of steps performed on predictor variables before data can be used for machine learning

1. Converting each Ordinal Categorical columns to numeric
2. Converting Binary nominal Categorical columns to numeric using 1/0 mapping
3. Converting all other nominal categorical columns to numeric using pd.get_dummies()
4. Data Transformation (Optional): Standardization/Normalization/log/sqrt. Important if you are using distance based algorithms like KNN, or Neural Networks

Converting the binary nominal variable to numeric using 1/0 mapping

In [27]:
```
# Converting the binary nominal variable sex to numeric
```

```
DataForML['Gender'].replace({'Female':0, 'Male':1}, inplace=True)
DataForML['family_history_with_overweight'].replace({'yes':0, 'no':1}, inplace=True)
DataForML['Frequent consumption of high caloric food'].replace({'yes':0, 'no':1}, inplac
DataForML['Smoke'].replace({'yes':0, 'no':1}, inplace=True)
DataForML['Calories consumption monitoring'].replace({'yes':0, 'no':1}, inplace=True)
```

Converting the nominal variable to numeric using get_dummies()¶

In [28]:
```
DataForML.head()
```

Out[28]:

	Gender	Age	Height	Weight	family_history_with_overweight	Frequent consumption of high caloric food	Frequency of consumption of vegetables	Number of main meals	Consump of f betw m
0	0	21.0	1.62	64.0	0	1	2.0	3.0	Someti
1	0	21.0	1.52	56.0	0	1	3.0	3.0	Someti
2	1	23.0	1.80	77.0	0	1	2.0	3.0	Someti
3	1	27.0	1.80	87.0	1	1	3.0	3.0	Someti
4	1	22.0	1.78	89.8	1	1	2.0	1.0	Someti

In [29]:
```
# Treating all the nominal variables at once using dummy variables
DataForML_Numeric=pd.get_dummies(DataForML)

# Adding Target Variable to the data
DataForML_Numeric['Obese Data']=ObesityData['Obese Data']

# Printing sample rows
DataForML_Numeric.head()
```

Out[29]:

	Gender	Age	Height	Weight	family_history_with_overweight	Frequent consumption of high caloric food	Frequency of consumption of vegetables	Number of main meals	Smoke
0	0	21.0	1.62	64.0	0	1	2.0	3.0	1
1	0	21.0	1.52	56.0	0	1	3.0	3.0	0
2	1	23.0	1.80	77.0	0	1	2.0	3.0	1
3	1	27.0	1.80	87.0	1	1	3.0	3.0	1
4	1	22.0	1.78	89.8	1	1	2.0	1.0	1

5 rows × 27 columns

Changing the target variables into numeric

In [30]:
```
print(DataForML_Numeric["Obese Data"].unique())
```

```
['Normal_Weight' 'Overweight_Level_I' 'Overweight_Level_II'
 'Obesity_Type_I' 'Insufficient_Weight' 'Obesity_Type_II'
 'Obesity_Type_III']
```

In [18]:
```
DataForML_Numeric['Obese Data'].replace({'Normal_Weight':0, 'Overweight_Level_I':1,'Over
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[18], line 1
----> 1 DataForML_Numeric['Obese Data'].replace({'Normal_Weight':0, 'Overweight_Level_I'
:1,'Overweight_Level_II':2,'Obesity_Type_I':3,'Insufficient_Weight':4,'Obesity_Type_II':
5,'Obesity_Type_III':6}, inplace=True)

NameError: name 'DataForML_Numeric' is not defined
```

In [32]:
```
DataForML_Numeric.head()
```

Out[32]:

	Gender	Age	Height	Weight	family_history_with_overweight	Frequent consumption of high caloric food	Frequency of consumption of vegetables	Number of main meals	Smoke
0	0	21.0	1.62	64.0	0	1	2.0	3.0	1
1	0	21.0	1.52	56.0	0	1	3.0	3.0	0
2	1	23.0	1.80	77.0	0	1	2.0	3.0	1
3	1	27.0	1.80	87.0	1	1	3.0	3.0	1
4	1	22.0	1.78	89.8	1	1	2.0	1.0	1

5 rows × 27 columns

Machine Learning: Splitting the data into Training and Testing sample

We dont use the full data for creating the model. Some data is randomly selected and kept aside for checking how good the model is. This is known as Testing Data and the remaining data is called Training data on which the model is built. Typically 70% of data is used as Training data and the rest 30% is used as Tesing data.

In [33]:
```
# Printing all the column names for our reference
DataForML_Numeric.columns
```

Out[33]:
```
Index(['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'Frequent consumption of high caloric food',
       'Frequency of consumption of vegetables', 'Number of main meals ',
       'Smoke', 'Consumption of water daily',
       'Calories consumption monitoring', 'Physical activity frequency',
       'Time using technology devices ',
       ' Consumption of food between meals_Always',
       ' Consumption of food between meals_Frequently',
       ' Consumption of food between meals_Sometimes',
       ' Consumption of food between meals_no',
       'Consumption of Alcohol_Always', 'Consumption of Alcohol_Frequently',
       'Consumption of Alcohol_Sometimes', 'Consumption of Alcohol_no',
       'MTRANS_Automobile', 'MTRANS_Bike', 'MTRANS_Motorbike',
       'MTRANS_Public_Transportation', 'MTRANS_Walking', 'Obese Data'],
      dtype='object')
```

In [34]:
```
# Separate Target Variable and Predictor Variables
TargetVariable='Obese Data'
Predictors=['Gender', 'Age', 'Height', 'Weight', 'family_history_with_overweight',
       'Frequent consumption of high caloric food',
       'Frequency of consumption of vegetables', 'Number of main meals ',
```

```
        'Smoke', 'Consumption of water daily',
        'Calories consumption monitoring', 'Physical activity frequency',
        'Time using technology devices ',
        ' Consumption of food between meals_Always',
        ' Consumption of food between meals_Frequently',
        ' Consumption of food between meals_Sometimes',
        ' Consumption of food between meals_no',
        'Consumption of Alcohol_Always', 'Consumption of Alcohol_Frequently',
        'Consumption of Alcohol_Sometimes', 'Consumption of Alcohol_no',
        'MTRANS_Automobile', 'MTRANS_Bike', 'MTRANS_Motorbike',
        'MTRANS_Public_Transportation', 'MTRANS_Walking']

X=DataForML_Numeric[Predictors].values
y=DataForML_Numeric[TargetVariable].values

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
# Split X and Y into training and test set in 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=20
```

In [35]:

```
# Sanity check for the sampled data
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(1460, 26)
(1460,)
(627, 26)
(627,)
```

Logistic Regression

In [36]:

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
# choose parameter Penalty='l1' or C=1
# choose different values for solver 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'
clf = LogisticRegression(C=5,penalty='l2', solver='newton-cg')

# Printing all the parameters of logistic regression
# print(clf)

# Creating the model on Training Data
LOG=clf.fit(X_train,y_train)

# Generating predictions on testing data
prediction=LOG.predict(X_test)
# Printing sample values of prediction in Testing data
TestingData=pd.DataFrame(data=X_test, columns=Predictors)
TestingData['Obese Data']=y_test
TestingData['Predicted_Obese Data']=prediction
print(TestingData.head())


# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(prediction, y_test))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))
```

```
# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(LOG, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
   Gender        Age    Height      Weight  family_history_with_overweight  \
0     0.0  25.000000  1.560000   45.000000                             1.0
1     1.0  32.610018  1.742538   83.390983                             0.0
2     1.0  25.137087  1.772045  114.067936                             0.0
3     0.0  20.979254  1.756550   78.721696                             0.0
4     1.0  30.022598  1.747739   83.314157                             0.0

   Frequent consumption of high caloric food  \
0                                        0.0
1                                        0.0
2                                        0.0
3                                        0.0
4                                        0.0

   Frequency of consumption of vegetables  Number of main meals   Smoke  \
0                                2.000000              3.000000    1.0
1                                2.247704              3.053598    1.0
2                                1.624366              3.000000    1.0
3                                2.000000              3.000000    1.0
4                                2.011656              3.165837    1.0

   Consumption of water daily  ...  Consumption of Alcohol_Frequently  \
0                    1.000000  ...                                0.0
1                    1.919629  ...                                0.0
2                    2.081719  ...                                0.0
3                    2.813234  ...                                1.0
4                    1.844645  ...                                0.0

   Consumption of Alcohol_Sometimes  Consumption of Alcohol_no  \
0                               1.0                        0.0
1                               0.0                        1.0
2                               1.0                        0.0
3                               0.0                        0.0
4                               0.0                        1.0

   MTRANS_Automobile  MTRANS_Bike  MTRANS_Motorbike  \
0                0.0          0.0               0.0
1                1.0          0.0               0.0
2                0.0          0.0               0.0
3                0.0          0.0               0.0
4                1.0          0.0               0.0

   MTRANS_Public_Transportation  MTRANS_Walking  Obese Data  \
0                           1.0             0.0           0
1                           0.0             0.0           2
2                           1.0             0.0           5
3                           1.0             0.0           1
4                           0.0             0.0           2

   Predicted_Obese Data
0                     4
1                     2
2                     5
3                     2
4                     2

[5 rows x 28 columns]
```

```
              precision    recall  f1-score   support

           0       0.94      0.68      0.79        92
           1       0.75      0.79      0.77        98
           2       0.69      0.67      0.68        70
           3       0.88      0.93      0.90        86
           4       0.86      1.00      0.92        85
           5       0.93      0.97      0.95       102
           6       1.00      1.00      1.00        94

    accuracy                           0.87       627
   macro avg       0.87      0.86      0.86       627
weighted avg       0.87      0.87      0.87       627

[[63  3  1  0  0  0  0]
 [13 77 11  1  0  0  0]
 [ 2 17 47  2  0  0  0]
 [ 0  1  7 80  0  3  0]
 [14  0  0  0 85  0  0]
 [ 0  0  4  3  0 99  0]
 [ 0  0  0  0  0  0 94]]
Accuracy of the model on Testing Sample Data: 0.87

Accuracy values for 10-fold Cross Validation:
 [0.71182476 0.76355378 0.85952116 0.90121874 0.86043777 0.8890112
 0.92771743 0.91821896 0.91184658 0.89670778]

Final Average Accuracy of the model: 0.86
```

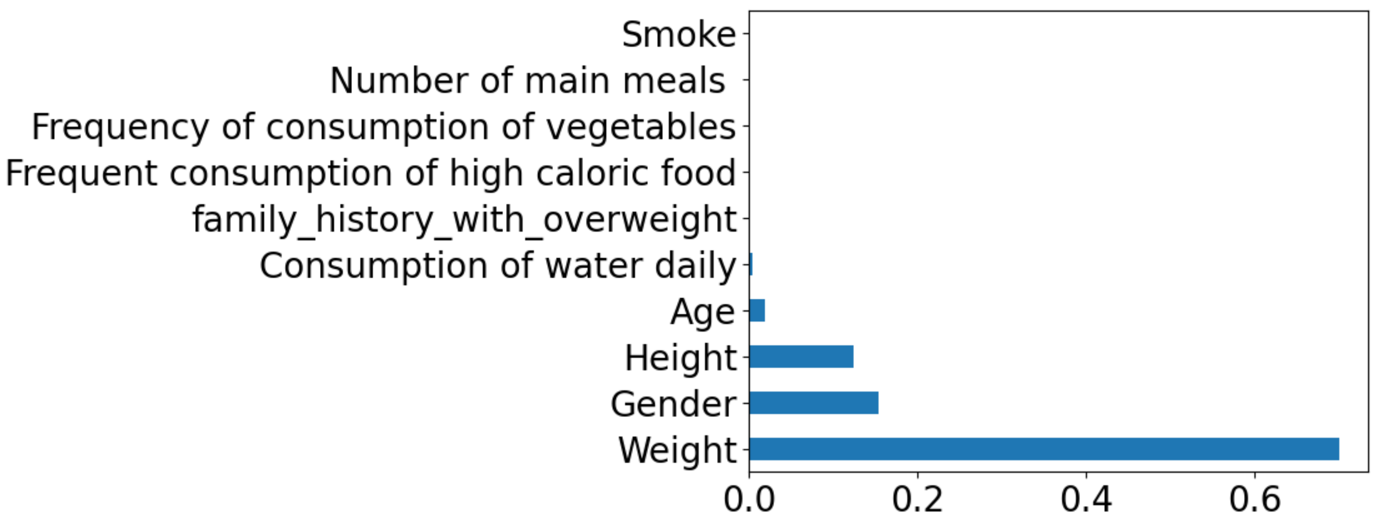
Decision Tree

In [37]:

```
#Decision Trees
from sklearn import tree
#choose from different tunable hyper parameters
clf = tree.DecisionTreeClassifier(max_depth=4,criterion='entropy')

# Printing all the parameters of Decision Trees
print(clf)

# Creating the model on Training Data
DTree=clf.fit(X_train,y_train)
prediction=DTree.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(y_test, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(DTree.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(DTree, X , y, cv=10, scoring='f1_weighted')
```

```
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
              precision    recall  f1-score   support

           0       0.57      0.83      0.67        92
           1       0.66      0.26      0.37        98
           2       0.41      0.64      0.50        70
           3       0.73      0.78      0.75        86
           4       0.89      0.67      0.77        85
           5       0.96      0.91      0.93       102
           6       1.00      0.99      0.99        94

    accuracy                           0.73       627
   macro avg       0.75      0.72      0.71       627
weighted avg       0.76      0.73      0.72       627

[[76  2  7  0  7  0  0]
 [30 25 42  1  0  0  0]
 [ 0 10 45 15  0  0  0]
 [ 0  1 15 67  0  3  0]
 [28  0  0  0 57  0  0]
 [ 0  0  0  9  0 93  0]
 [ 0  0  0  0  0  1 93]]
Accuracy of the model on Testing Sample Data: 0.72

Accuracy values for 10-fold Cross Validation:
 [0.68081193 0.70114962 0.70639918 0.70760132 0.72467543 0.72032559
 0.64233312 0.70920665 0.72730817 0.70917789]

Final Average Accuracy of the model: 0.7
```



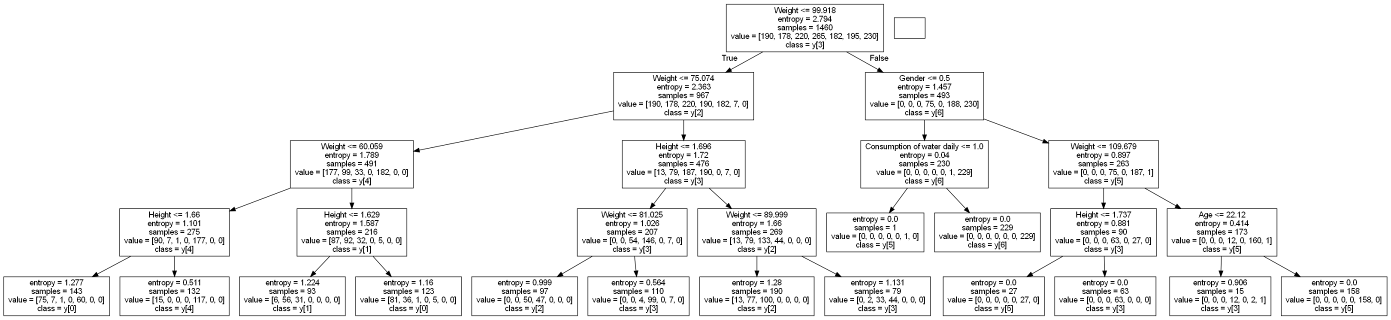
Plotting A Decision Tree

In [38]:

```
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data
dot_data = tree.export_graphviz( DTree,out_file=None,
                                feature_names=Predictors, class_names= True )

# printing the rules
#print(dot_data)
```

```python
# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=3000,height=3000)
# Double click on the graph to zoom in
```

Out[38]:



Random Forest

In [39]:

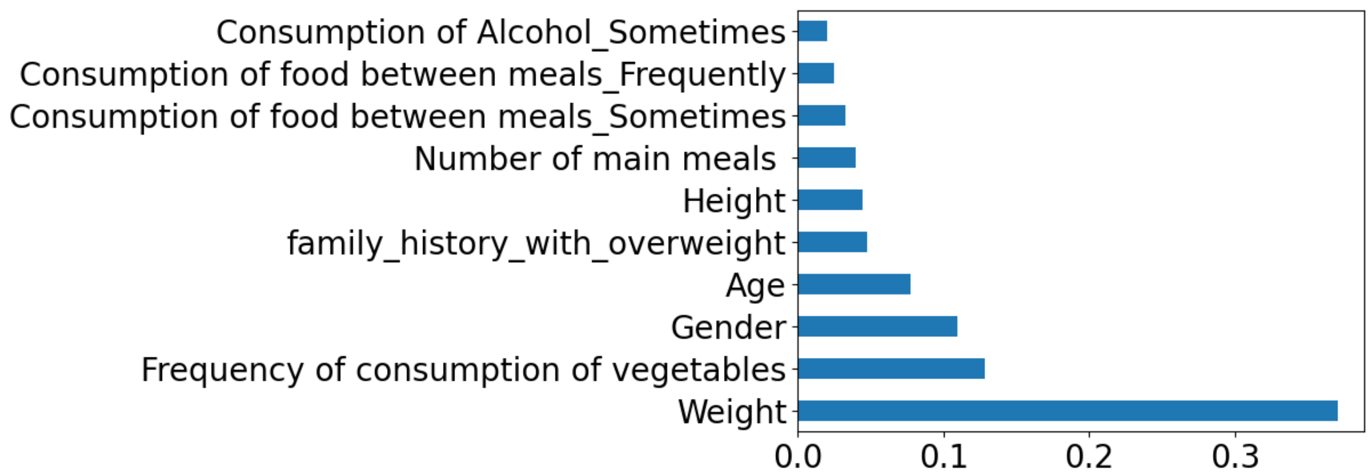
```python
# Random Forest (Bagging of multiple Decision Trees)
from sklearn.ensemble import RandomForestClassifier
# Choose different hyperparameter values of max_depth, n_estimators and criterion to tun
clf = RandomForestClassifier(max_depth=4, n_estimators=100,criterion='gini')

# Printing all the parameters of Random Forest
print(clf)

# Creating the model on Training Data
RF=clf.fit(X_train,y_train)
prediction=RF.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(y_test, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(RF, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))


# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(RF.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
```

```
RandomForestClassifier(max_depth=4)
              precision    recall  f1-score   support

           0       0.73      0.67      0.70        92
           1       0.85      0.29      0.43        98
           2       0.61      0.79      0.69        70
           3       0.65      0.87      0.75        86
           4       0.82      0.98      0.89        85
```

```
           5       0.90      0.92      0.91       102
           6       0.95      1.00      0.97        94

    accuracy                           0.78       627
   macro avg       0.79      0.79      0.76       627
weighted avg       0.80      0.78      0.76       627

[[62  5  5  3 17  0  0]
 [13 28 28 24  1  0  4]
 [ 6  0 55  5  0  3  1]
 [ 2  0  2 75  0  7  0]
 [ 2  0  0  0 83  0  0]
 [ 0  0  0  8  0 94  0]
 [ 0  0  0  0  0  0 94]]
Accuracy of the model on Testing Sample Data: 0.76

Accuracy values for 10-fold Cross Validation:
 [0.53715351 0.64060553 0.7784573  0.80178577 0.77704091 0.78411361
 0.78222003 0.83283003 0.80897329 0.8156191 ]

Final Average Accuracy of the model: 0.76
```

Out[39]:

```
<Axes: >
```

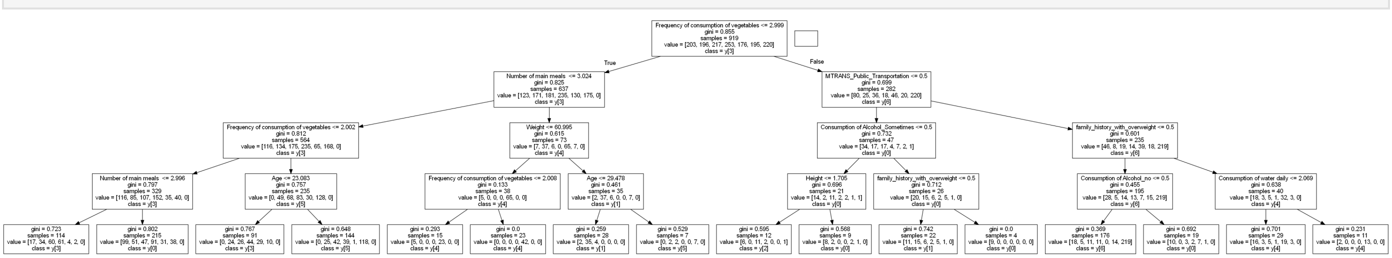


Plotting one of the Decision Trees in Random Forest

In [40]:

```python
# Plotting a single Decision Tree from Random Forest
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Random Forest
dot_data = tree.export_graphviz(clf.estimators_[5] , out_file=None, feature_names=Predic

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=3000,height=4000)
# Double click on the graph to zoom in
```

Out[40]:



Adaboost

In [41]:

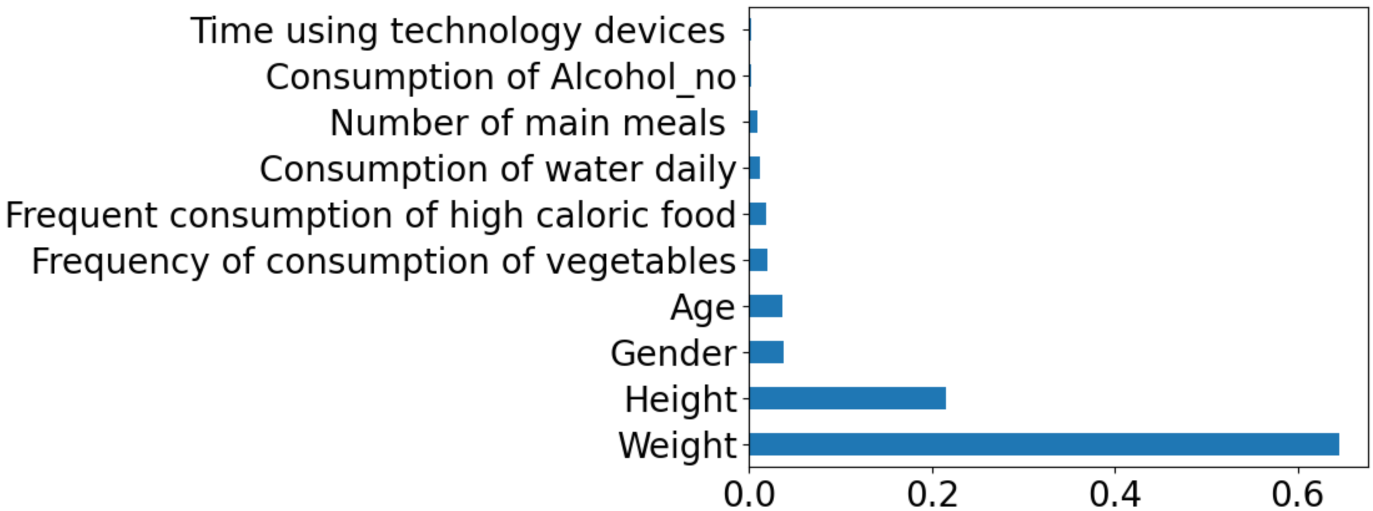
```
# Adaboost
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

# Choosing Decision Tree with 1 level as the weak learner
DTC=DecisionTreeClassifier(max_depth=4)
clf = AdaBoostClassifier(n_estimators=100, base_estimator=DTC ,learning_rate=0.01)

# Printing all the parameters of Adaboost
print(clf)

# Creating the model on Training Data
AB=clf.fit(X_train,y_train)
prediction=AB.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(y_test, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(AB, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(AB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
```

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=4),
                   learning_rate=0.01, n_estimators=100)
              precision    recall  f1-score   support

           0       0.89      0.79      0.84        92
           1       0.83      0.86      0.84        98
           2       0.71      0.93      0.81        70
           3       0.91      0.79      0.84        86
           4       0.96      0.94      0.95        85
           5       0.98      0.97      0.98       102
           6       1.00      1.00      1.00        94

    accuracy                           0.90       627
   macro avg       0.90      0.90      0.89       627
weighted avg       0.90      0.90      0.90       627

[[73 16  0  0  3  0  0]
 [ 4 84 10  0  0  0  0]
 [ 0  1 65  4  0  0  0]
 [ 0  0 16 68  0  2  0]
 [ 5  0  0  0 80  0  0]
 [ 0  0  0  3  0 99  0]
 [ 0  0  0  0  0  0 94]]
```

```
Accuracy of the model on Testing Sample Data: 0.9

Accuracy values for 10-fold Cross Validation:
 [0.86117139 0.91880678 0.9474009  0.8965942  0.93754239 0.94253207
 0.94291739 0.93358752 0.91143228 0.94200654]

Final Average Accuracy of the model: 0.92
```

Out[41]:

```
<Axes: >
```

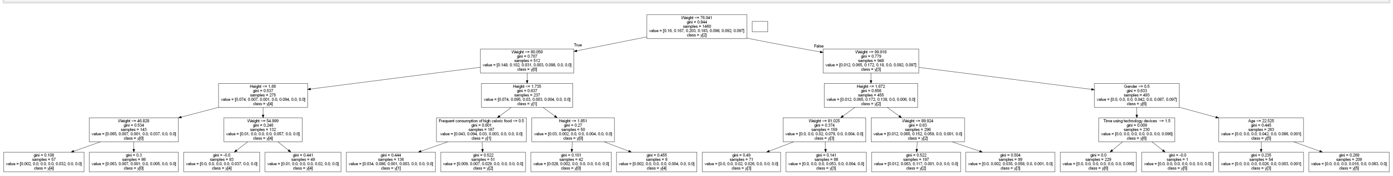


Plotting one of the Decision trees from Adaboost

In [42]:

```python
# PLotting 5th single Decision Tree from Adaboost
# Load libraries
from IPython.display import Image
from sklearn import tree
import pydotplus

# Create DOT data for the 6th Decision Tree in Adaboost
dot_data = tree.export_graphviz(clf.estimators_[5] , out_file=None, feature_names=Predic

# Draw graph
graph = pydotplus.graph_from_dot_data(dot_data)

# Show graph
Image(graph.create_png(), width=5000,height=5000)
# Double click on the graph to zoom in
```

Out[42]:



XGBOOST

In [43]:

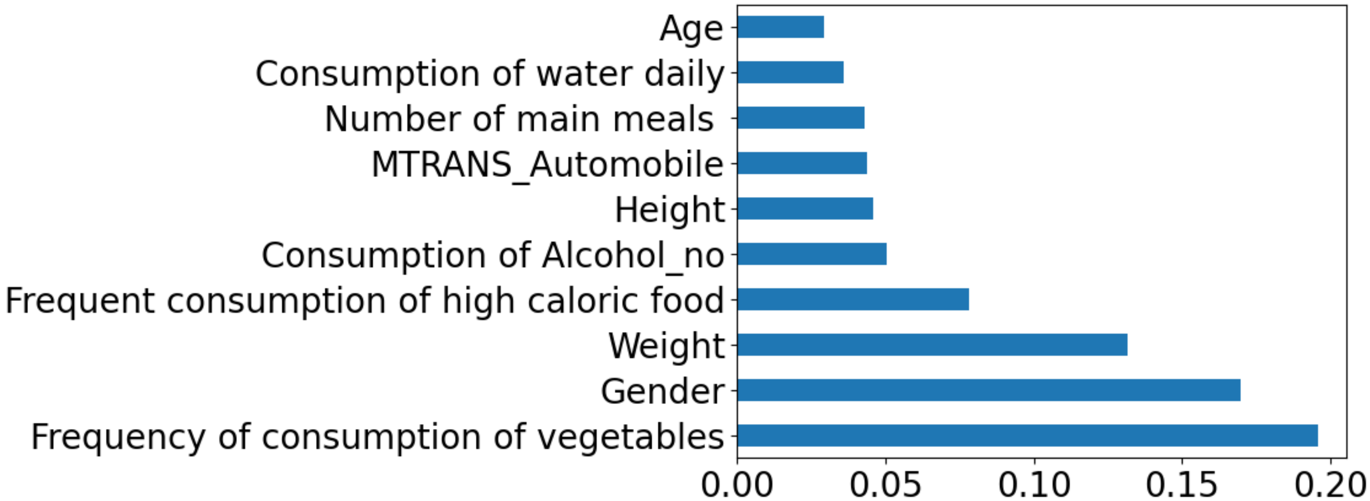
```python
# Xtreme Gradient Boosting (XGBoost)
from xgboost import XGBClassifier
clf=XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=200, objective='binary:lo

# Printing all the parameters of XGBoost
print(clf)

# Creating the model on Training Data
XGB=clf.fit(X_train,y_train)
prediction=XGB.predict(X_test)
```

```python
# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(y_test, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(XGB, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
feature_importances = pd.Series(XGB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
```

```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=200, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
              precision    recall  f1-score   support

           0       0.96      0.95      0.95        92
           1       0.95      0.95      0.95        98
           2       0.97      0.93      0.95        70
           3       0.94      0.97      0.95        86
           4       0.98      1.00      0.99        85
           5       0.99      0.98      0.99       102
           6       0.99      1.00      0.99        94

    accuracy                           0.97       627
   macro avg       0.97      0.97      0.97       627
weighted avg       0.97      0.97      0.97       627

[[ 87   3   0   0   2   0   0]
 [  4  93   1   0   0   0   0]
 [  0   1  65   4   0   0   0]
 [  0   1   1  83   0   1   0]
 [  0   0   0   0  85   0   0]
 [  0   0   0   1   0 100   1]
 [  0   0   0   0   0   0  94]]
Accuracy of the model on Testing Sample Data: 0.97

Accuracy values for 10-fold Cross Validation:
 [0.88062387 0.92125314 0.97129187 0.99043062 0.98075533 0.99522119
 1.         0.99518934 0.98075142 0.98069272]

Final Average Accuracy of the model: 0.97
```

Out[43]:

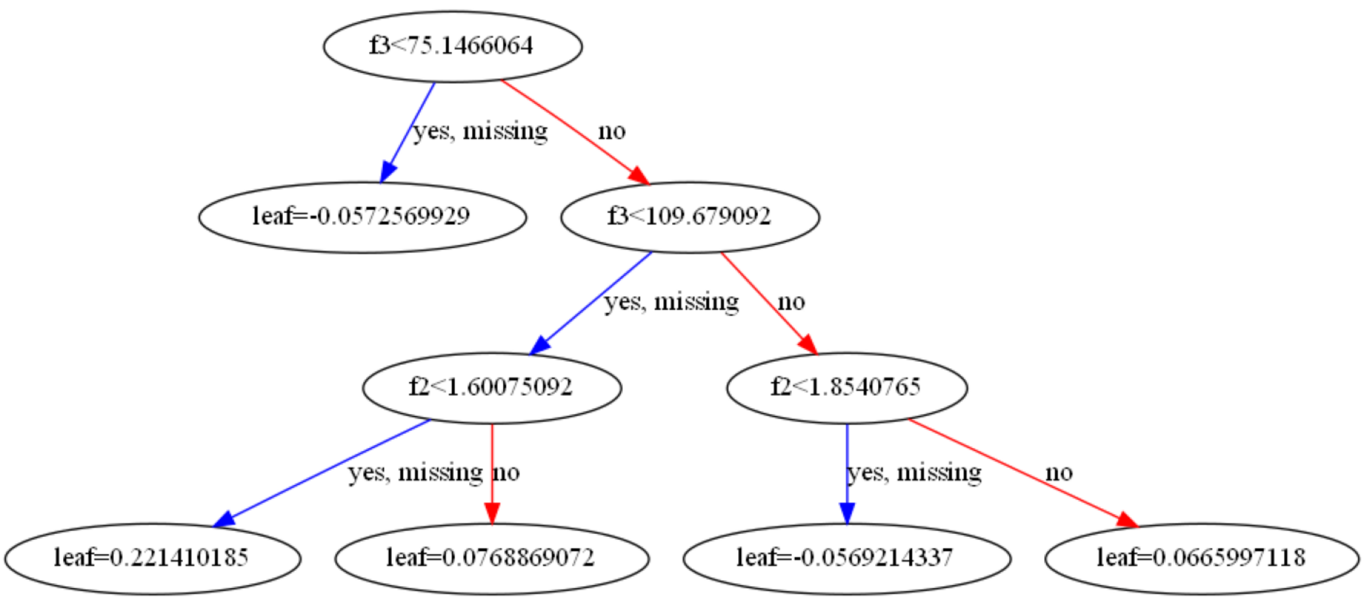
```
<Axes: >
```



Plotting a single decision tree out of XGBoost

In [44]:
```
from xgboost import plot_tree
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(20, 8))
plot_tree(XGB, num_trees=10, ax=ax)
```

Out[44]:
```
<Axes: >
```



Standardization/Normalization of data

You can choose not to run this step if you want to compare the resultant accuracy of this transformation with the accuracy of raw data.

However, if you are using KNN or Neural Networks, then this step becomes necessary.

In [45]:
```
### Sandardization of data ###
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardization or Normalization
# On this data Min Max Normalization produced better results

# Choose between standardization and MinMAx normalization
#PredictorScaler=StandardScaler()
```

```
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X
X=PredictorScalerFit.transform(X)

# Split the data into training and testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

KNN

In [46]:

```
# K-Nearest Neighbor(KNN)
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=4)

# Printing all the parameters of KNN
print(clf)

# Creating the model on Training Data
KNN=clf.fit(X_train,y_train)
prediction=KNN.predict(X_test)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y_test, prediction))
print(metrics.confusion_matrix(y_test, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y_test, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(KNN, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))


# Plotting the feature importance for Top 10 most important columns
# There is no built-in method to get feature importance in KNN
```

```
KNeighborsClassifier(n_neighbors=4)
              precision    recall  f1-score   support

           0       0.51      0.43      0.46        61
           1       0.76      0.71      0.74        55
           2       0.68      0.78      0.72        49
           3       0.78      0.83      0.81        70
           4       0.88      0.85      0.86        59
           5       0.90      0.97      0.93        64
           6       1.00      1.00      1.00        60

    accuracy                           0.80       418
   macro avg       0.79      0.79      0.79       418
weighted avg       0.79      0.80      0.79       418

[[26  6  9  9  7  4  0]
 [ 7 39  4  4  0  1  0]
```

```
 [ 6  2 38  3  0  0  0]
 [ 4  3  3 58  0  2  0]
 [ 8  1  0  0 50  0  0]
 [ 0  0  2  0  0 62  0]
 [ 0  0  0  0  0  0 60]]
Accuracy of the model on Testing Sample Data: 0.79

Accuracy values for 10-fold Cross Validation:
 [0.48732263 0.64137211 0.83714206 0.81445411 0.7849824  0.85927902
 0.84691784 0.8492802  0.84157391 0.8421592 ]

Final Average Accuracy of the model: 0.78
```

Deployment of the Model

Based on the above trials you select that algorithm which produces the best average accuracy. In this case, multiple algorithms have produced similar kind of average accuracy. Hence, we can choose any one of them.

I am choosing **XGBoost** as the final model since it is very fast for this data!

In order to deploy the model we follow below steps

1. Train the model using 100% data available
2. Save the model as a serialized file which can be stored anywhere
3. Create a python function which gets integrated with front-end(Tableau/Java Website etc.) to take all the inputs and returns the prediction

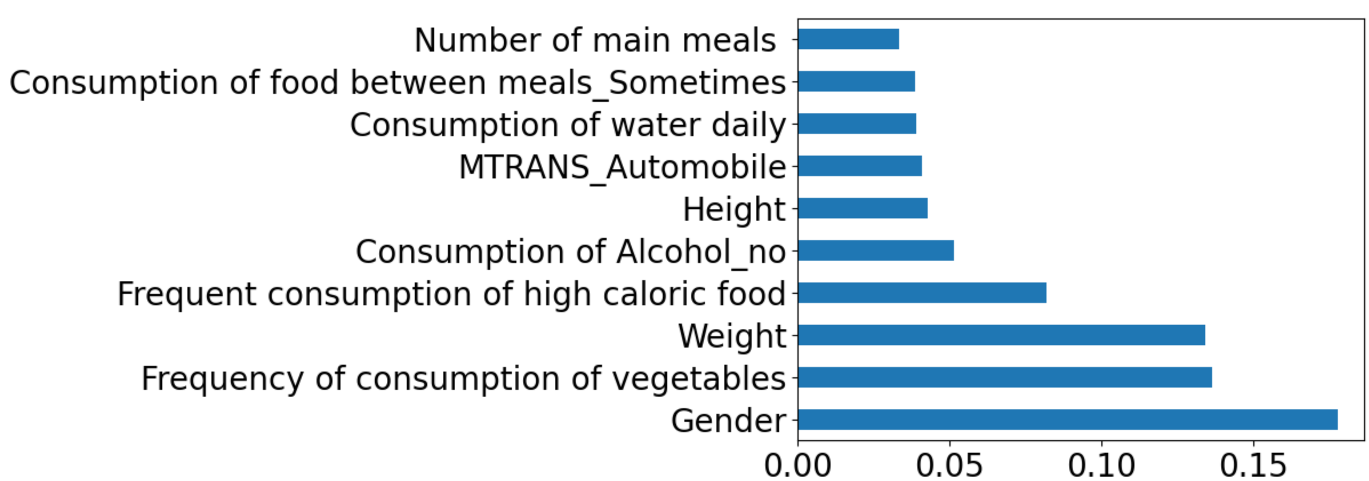
In [47]:

```python
# Xtreme Gradient Boosting (XGBoost)
from xgboost import XGBClassifier
clf=XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=200, objective='binary:lo

# Printing all the parameters of XGBoost
print(clf)

# Creating the model on Full Data
X=DataForML_Numeric[Predictors].values
y=DataForML_Numeric[TargetVariable].values
XGB=clf.fit(X,y)
prediction=XGB.predict(X)

# Measuring accuracy on Testing Data
from sklearn import metrics
print(metrics.classification_report(y, prediction))
print(metrics.confusion_matrix(y, prediction))

# Printing the Overall Accuracy of the model
F1_Score=metrics.f1_score(y, prediction, average='weighted')
print('Accuracy of the model on Testing Sample Data:', round(F1_Score,2))

# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(XGB, X , y, cv=10, scoring='f1_weighted')
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))

# Plotting the feature importance for Top 10 most important columns
%matplotlib inline
```

```
feature_importances = pd.Series(XGB.feature_importances_, index=Predictors)
feature_importances.nlargest(10).plot(kind='barh')
```

```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=3, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=200, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       282
           1       1.00      1.00      1.00       276
           2       1.00      1.00      1.00       290
           3       1.00      1.00      1.00       351
           4       1.00      1.00      1.00       267
           5       1.00      1.00      1.00       297
           6       1.00      1.00      1.00       324

    accuracy                           1.00      2087
   macro avg       1.00      1.00      1.00      2087
weighted avg       1.00      1.00      1.00      2087

[[282   0   0   0   0   0   0]
 [  0 276   0   0   0   0   0]
 [  0   0 290   0   0   0   0]
 [  0   0   0 351   0   0   0]
 [  0   0   0   0 267   0   0]
 [  0   0   0   0   0 297   0]
 [  0   0   0   0   0   0 324]]
Accuracy of the model on Testing Sample Data: 1.0

Accuracy values for 10-fold Cross Validation:
 [0.88062387 0.92125314 0.97129187 0.99043062 0.98075533 0.99522119
 1.         0.99518934 0.98075142 0.98069272]

Final Average Accuracy of the model: 0.97
```

Out[47]: <Axes: >



Ploting a Decision Tree with the whole data

In [48]:

```
X=DataForML_Numeric[Predictors].values
y=DataForML_Numeric[TargetVariable].values
```

```
### Sandardization of data ###
from sklearn.preprocessing import StandardScaler, MinMaxScaler
# Choose either standardization or Normalization
# On this data Min Max Normalization produced better results

# Choose between standardization and MinMAx normalization
#PredictorScaler=StandardScaler()
PredictorScaler=MinMaxScaler()

# Storing the fit object for later reference
PredictorScalerFit=PredictorScaler.fit(X)

# Generating the standardized values of X
X=PredictorScalerFit.transform(X)

print(X.shape)
print(y.shape)
```

```
(2087, 26)
(2087,)
```

Step 1. Retraining the model using 100% data

In [49]:

```
#Decision Trees
from sklearn import tree
#choose from different tunable hyper parameters
clf = tree.DecisionTreeClassifier(max_depth=4,criterion='entropy')

# Training the model on 100% Data available
FinalDecisionTreeModel=clf.fit(X,y)
```

Cross validating the final model accuracy with less predictors

In [50]:

```
# Importing cross validation function from sklearn
from sklearn.model_selection import cross_val_score

# Running 10-Fold Cross validation on a given algorithm
# Passing full data X and y because the K-fold will split the data and automatically cho
Accuracy_Values=cross_val_score(FinalDecisionTreeModel, X , y, cv=10, scoring='f1_weight
print('\nAccuracy values for 10-fold Cross Validation:\n',Accuracy_Values)
print('\nFinal Average Accuracy of the model:', round(Accuracy_Values.mean(),2))
```

```
Accuracy values for 10-fold Cross Validation:
 [0.68081193 0.70114962 0.70639918 0.70760132 0.72467543 0.72032559
 0.64233312 0.70920665 0.72730817 0.70917789]

Final Average Accuracy of the model: 0.7
```

Insights from Data

- Obesity_Type III is maximum in Females and Obesity_Type II in Males.
- People with Family History of Obesity has a higher chances of becoming obese.
- People consuming high caloric food has a higher chances of becoming obese.
- People who smoke and consumes food between meals has a higher chance of becoming obese.
- People consuming Alcohol sometimes has higher chance of developing Obesity .
- Number of people whose age is above 35 are obese.
- Factors which helps in reducing obesity are : Higher Physical Activity Frequency, Keeping number of mean meals less than 3 , Not smoking , Maintaining a proper diet and calory consumption monitoring .

In [ ]: