Estimation of obesity levels based on eating habits and physical condition

link to the dataset and important information :

https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+con
(https://archive.ics.uci.edu/ml/datasets/Estimation+of+obesity+levels+based+on+eating+habits+and+physical+co

sciencedirect.com/science/article/pii/S2352340919306985?via%3Dihub

"This dataset include data for the estimation of obesity levels in individuals from the countries of Mexico, Peru and Colombia, based on their eating habits and physical condition. The data contains 17 attributes and 2111 records, the records are labeled with the class variable NObesity (Obesity Level), that allows classification of the data using the values of Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III. 77% of the data was generated synthetically using the Weka tool and the SMOTE filter, 23% of the data was collected directly from users through a web platform."

Packages

Entrée [1]:

```
import pandas as pd
import numpy as np
import sklearn as skl
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output, State
import statistics
from sklearn import preprocessing
from sklearn import svm
from sklearn.model_selection import StratifiedKFold,cross_val_score,RandomizedSearchCV,Grid
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
import xgboost as xgb
```

```
<ipython-input-1-535d7a68412c>:11: UserWarning:
The dash_core_components package is deprecated. Please replace
`import dash_core_components as dcc` with `from dash import dcc`
  import dash_core_components as dcc
<ipython-input-1-535d7a68412c>:12: UserWarning:
The dash_html_components package is deprecated. Please replace
`import dash_html_components as html` with `from dash import html`
  import dash_html_components as html
```

Variables

Entrée [3]:

```
#df = pd.read_csv('ObesityDataSet_raw_and_data_sinthetic.csv')
df= pd.read_csv("ObesityDataSet.csv")
```

Description

Entrée [4]:

```
df.head()
```

Out[4]:

	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

The dataset contains 16 variables. 15 features:

- Gender
- Age
- Height
- Weight
- Family History with Overweight
- Attributes related with eating habits (6)
- Attributes related with the physical condition (4)

1 target:

- NObeyesdad, renamed NObesity

Another revelant information is that most of these variables are category, except for Age, Height and Weight.

Problematic

Based on all the data, the objective of this project is to classify individuals according to their obesity level. This variables will be named: **NObesity**. This variable will take differents values among:

- Insufficient_Weight
- Normal_Weight
- Overweight_Level_I
- Overweight_Level_II
- Obesity_Type_I
- Obesity_Type_II
- Obesity_Type_III

Data cleaning : part 1

We will perform data cleansing in two parts. In the first part we will process the data so that they are the most usable for graphic analysis. And in the second part we then transform the variables so that it is usable for the algorithm

Missing values

First, let's drop all the row with missing value

Entrée [5]:

```
df_clean = df.dropna()
print(df_clean.shape, df.shape)
```

```
(2111, 17) (2111, 17)
```

We can see that they have the same shape, so there is no missing value: we keep **df** for our analyse

Non-numeric values

Entrée [6]:

```
df.info()
```

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

Outliers

For non-categorical variables, we need to check if there is any outliers. There 3 concerned variables are Age, Height and Weight.

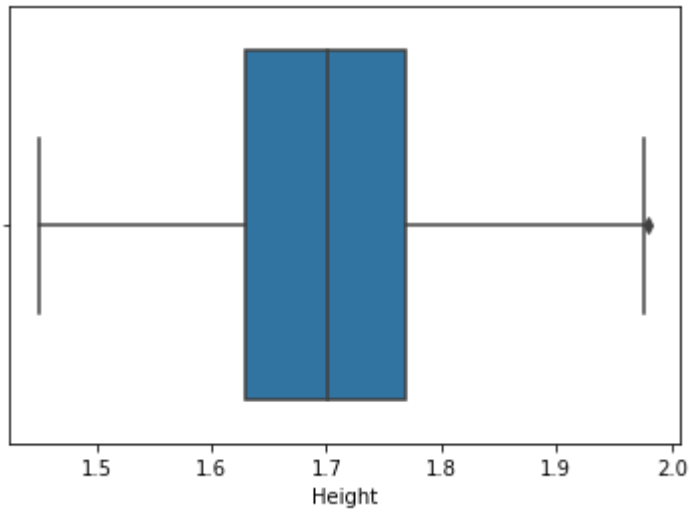
Age is not a problem, because the study has been done with subjects between 14 and 61 years.

Entrée [7]:

```
sns.boxplot(x=df['Height'])
```

Out[7]:

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



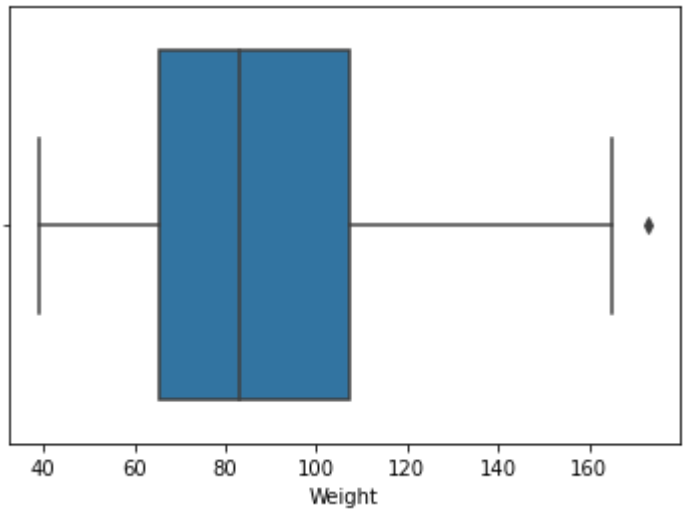
Only few point are outside the box for the Height

Entrée [8]:

```
sns.boxplot(x=df['Weight'])
```

Out[8]:

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



Same for the Weight, we will exclude the extrem values. We set the terminales by using the quantiles values.

Entrée [25]:

```
df_categorized = df.copy()
```

Entrée [27]:

```
q1, q2 = df_categorized["Height"].quantile(q=0.25), df_categorized["Height"].quantile(q=0.7
q3, q4 = df_categorized["Weight"].quantile(q=0.25), df_categorized["Weight"].quantile(q=0.7

IQR1, IQR2 = q2-q1, q4-q3

#Height selection
df_categorized = df_categorized[df_categorized["Height"] >  q1 - 1.5*IQR1]
df_categorized = df_categorized[df_categorized["Height"] <  q2 + 1.5*IQR1]

#Weight selection
df_categorized = df_categorized[df_categorized["Weight"] >  q3 - 1.5*IQR2]
df_categorized = df_categorized[df_categorized["Weight"] <  q4 + 1.5*IQR2]

df_categorized.shape
```

Out[27]:

```
(2108, 17)
```

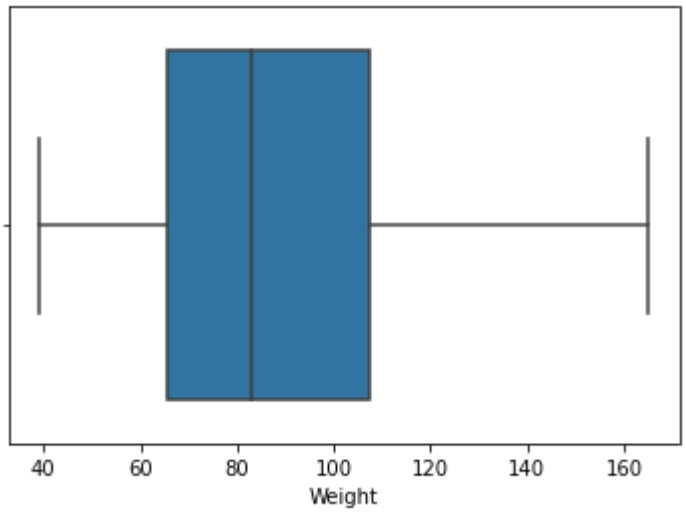
```
Only 3 lines were deleted.
```

Entrée [15]:

```
sns.boxplot(x=df_categorized['Weight'])
```

Out[15]:

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

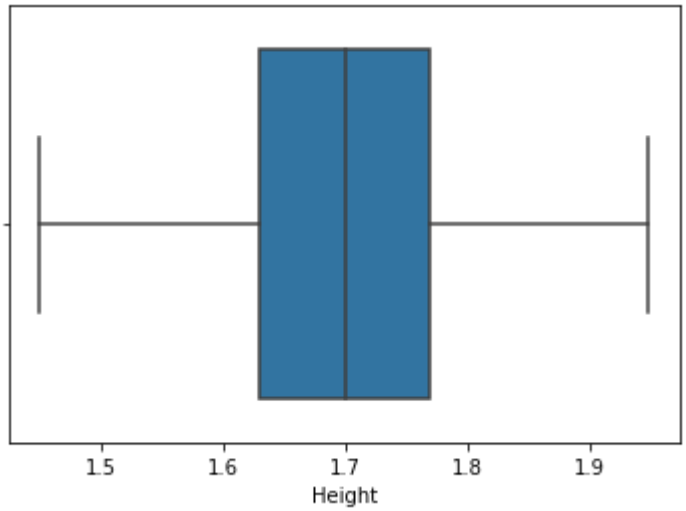


Entrée [28]:

```
sns.boxplot(x=df_categorized['Height'])
```

Out[28]:

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



Data Analysis

Features

Entrée [29]:

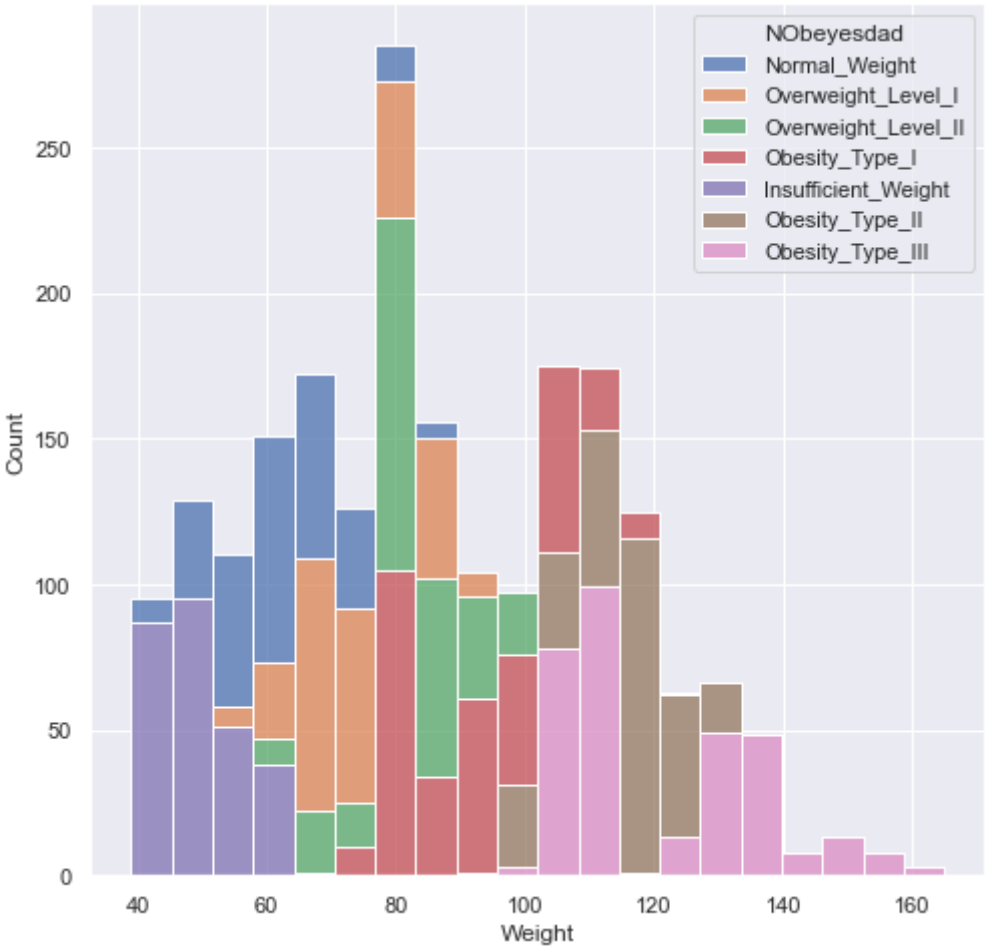
```
df.describe().T.style.bar(subset=['mean'], color='#606ff2').background_gradient(subset=['st
```

Out[29]:

	count	mean	std	min	25%	50%	75%	
Age	2111.000000	24.312600	6.345968	14.000000	19.947192	22.777890	26.000000	61.00
Height	2111.000000	1.701677	0.093305	1.450000	1.630000	1.700499	1.768464	1.98
Weight	2111.000000	86.586058	26.191172	39.000000	65.473343	83.000000	107.430682	173.00
FCVC	2111.000000	2.419043	0.533927	1.000000	2.000000	2.385502	3.000000	3.00
NCP	2111.000000	2.685628	0.778039	1.000000	2.658738	3.000000	3.000000	4.00
CH2O	2111.000000	2.008011	0.612953	1.000000	1.584812	2.000000	2.477420	3.00
FAF	2111.000000	1.010298	0.850592	0.000000	0.124505	1.000000	1.666678	3.00
TUE	2111.000000	0.657866	0.608927	0.000000	0.000000	0.625350	1.000000	2.00

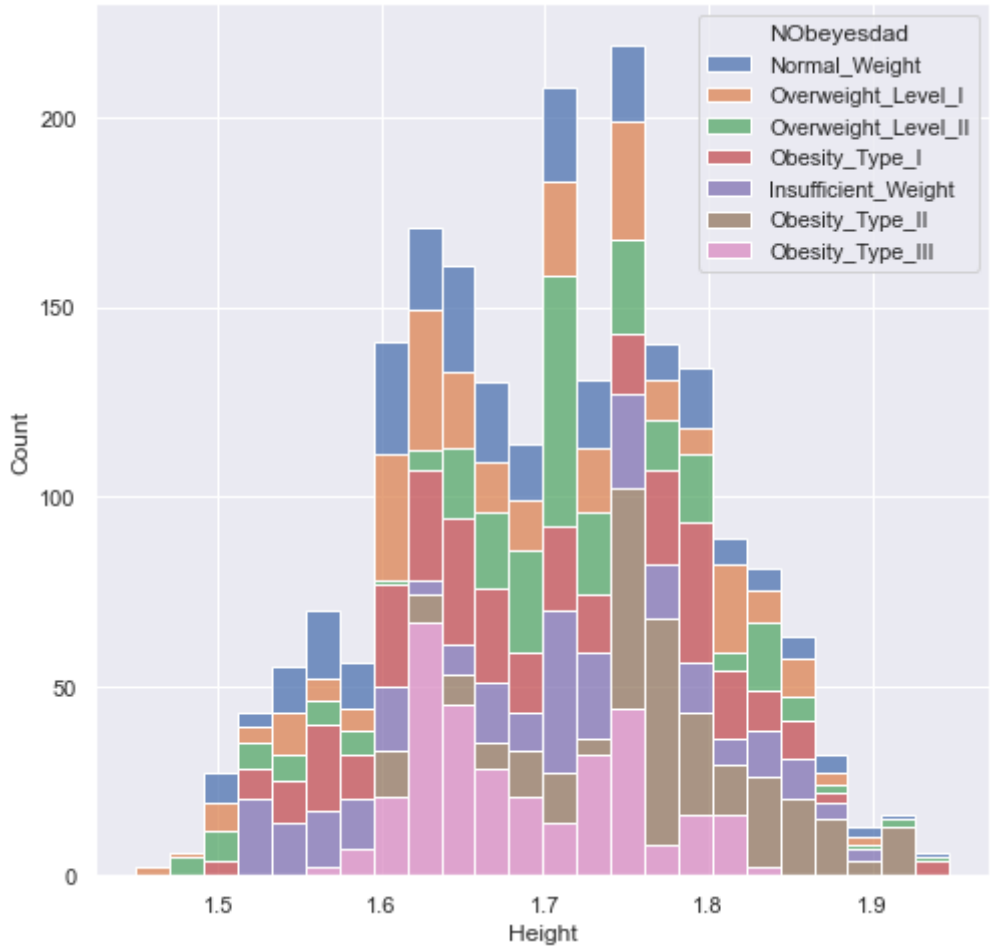
Entrée [32]:

```
sns.histplot(data = df_categorized, x = df_categorized['Weight'], hue = df_categorized['NOb
sns.set(rc = {'figure.figsize':(8,8)})
```



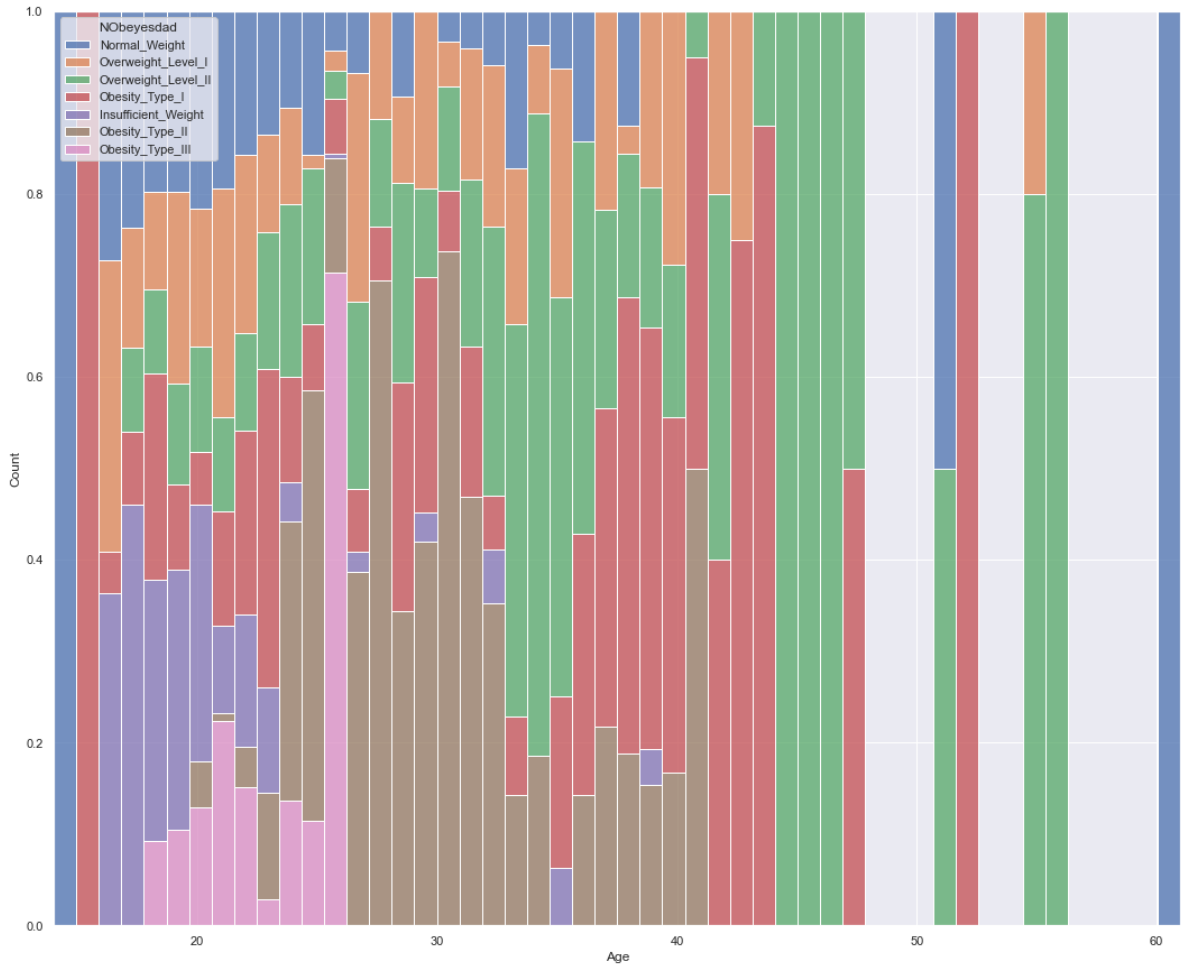
Entrée [31]:

```
ax = sns.histplot(data = df_categorized, x = df_categorized['Height'], hue = df_categorized
sns.set(rc = {'figure.figsize':(8,8)})
```



Entrée [181]:

```
sns.histplot(data = df_categorized, x = df_categorized['Age'], hue = df_categorized['NObeye
sns.set(rc = {'figure.figsize':(18,15)})
```



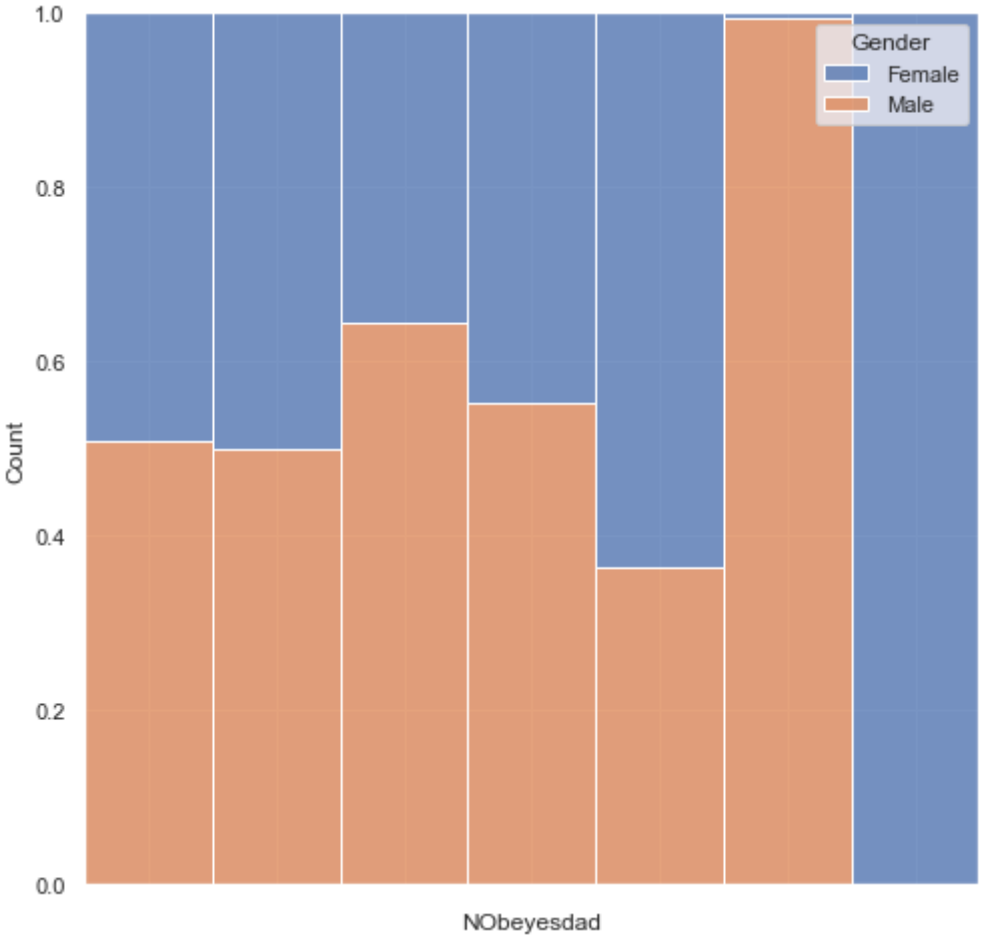
As we can see, Weight has a huge impact on the classification: an individual under 80kg cannot be considered as obese, which makes sense, but it considerably reduce the possibility choice. For the feature age, we can see that Obesity type 3 person have a life expectency way less higher than other: no case of type 3 are register over 30 years old.

Impact on target

Entrée [33]:

```
ax = sns.histplot(data = df_categorized, x = df_categorized['NObeyesdad'], hue = df_categor
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
sns.set(rc = {'figure.figsize':(8,8)})
```

```
<ipython-input-33-07af83f6c73f>:2: UserWarning: FixedFormatter should only b
e used together with FixedLocator
  ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
```

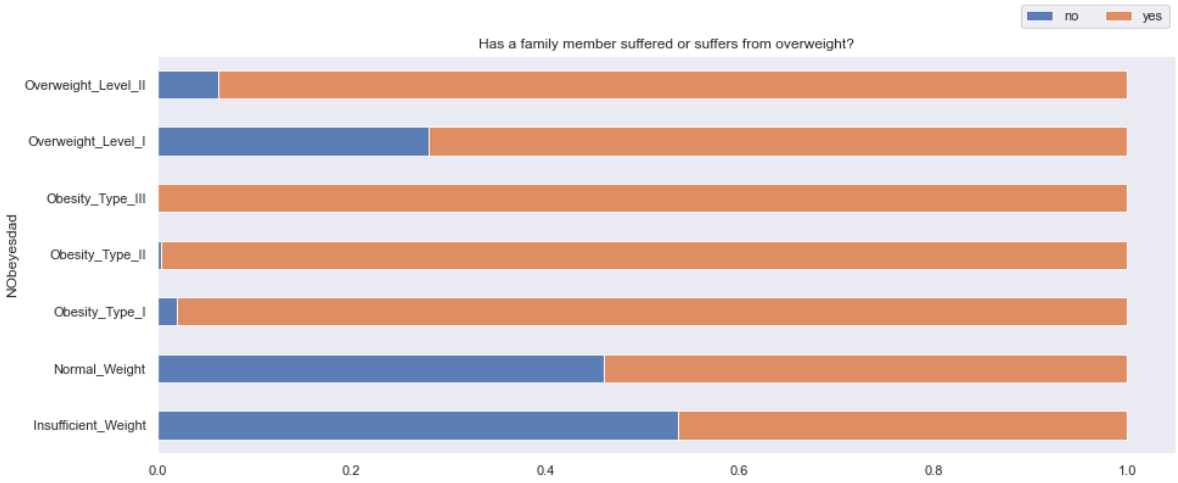


Entrée [34]:

```
big_count = pd.crosstab(df_categorized['NObeyesdad'], df_categorized['family_history_with_o
big_pct = big_count.div(big_count.sum(1), axis=0)
big_pct.plot.barh(stacked=True, figsize=(15, 6), alpha=0.9, grid=False)
plt.legend(loc="right", bbox_to_anchor=(1, 1.1), ncol=2)
plt.title("Has a family member suffered or suffers from overweight?")
```

Out[34]:

```
Text(0.5, 1.0, 'Has a family member suffered or suffers from overweight?')
```

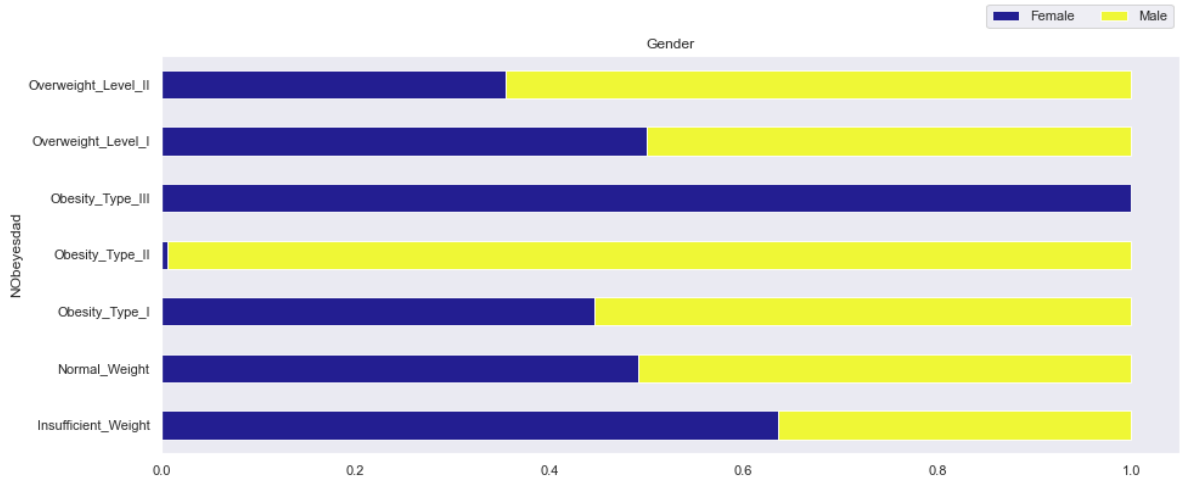


Entrée [35]:

```
big_count = pd.crosstab(df_categorized['NObeyesdad'], df_categorized['Gender'])
big_pct = big_count.div(big_count.sum(1), axis=0)
big_pct.plot.barh(stacked=True, figsize=(15, 6), alpha=0.9, grid=False, cmap='plasma')
plt.legend(loc="right", bbox_to_anchor=(1, 1.1), ncol=2)
plt.title("Gender")
```

Out[35]:

```
Text(0.5, 1.0, 'Gender')
```

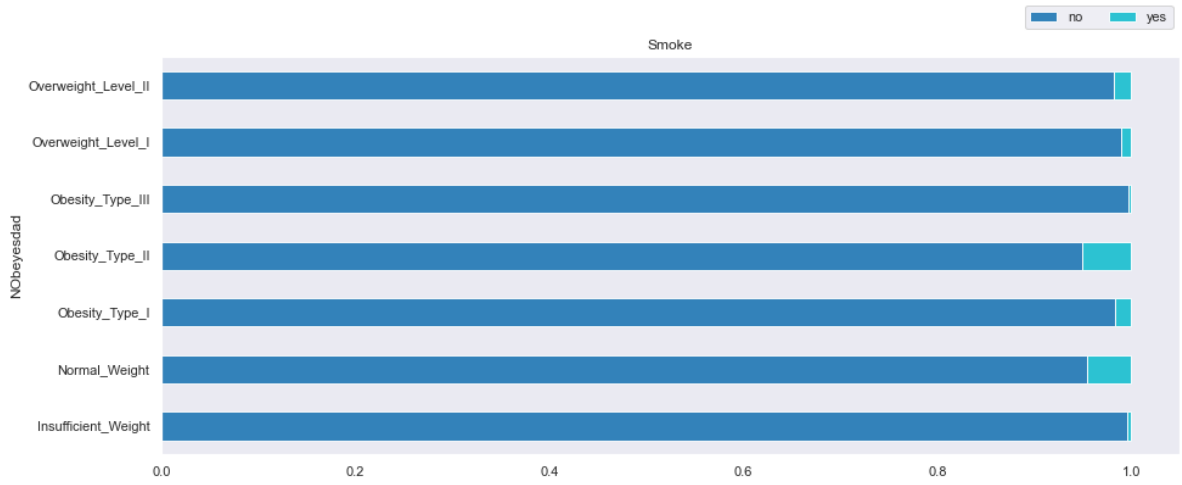


Entrée [36]:

```
big_count = pd.crosstab(df_categorized['NObeyesdad'], df_categorized['SMOKE'])
big_pct = big_count.div(big_count.sum(1), axis=0)
big_pct.plot.barh(stacked=True, figsize=(15, 6), alpha=0.9, grid=False, cmap = sns.color_pa
plt.legend(loc="right", bbox_to_anchor=(1, 1.1), ncol=2)
plt.title("Smoke")
```

Out[36]:

```
Text(0.5, 1.0, 'Smoke')
```

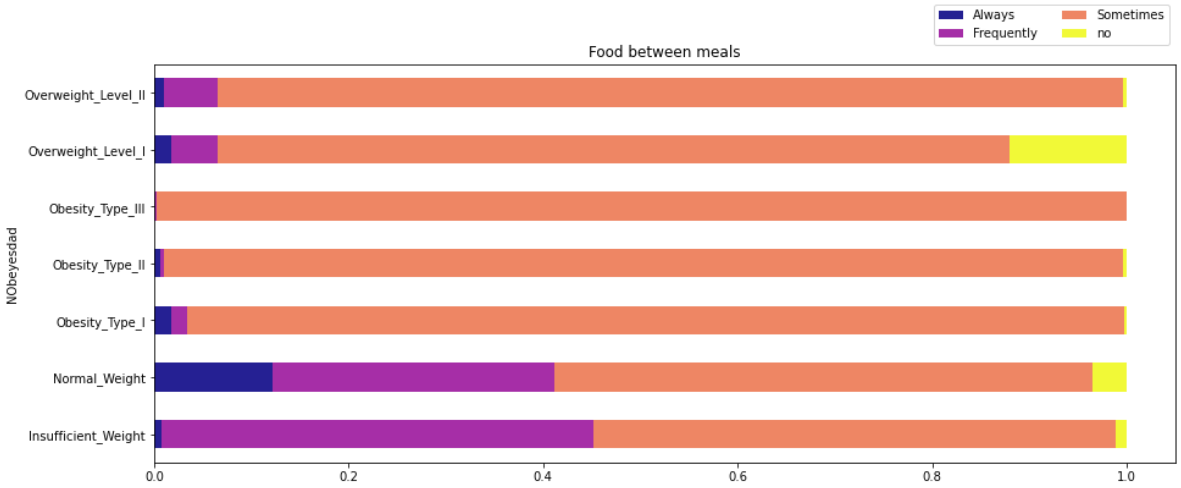


Entrée [248]:

```
big_count = pd.crosstab(df_categorized['NObeyesdad'], df_categorized['CAEC'])
big_pct = big_count.div(big_count.sum(1), axis=0)
big_pct.plot.barh(stacked=True, figsize=(15, 6), alpha=0.9, grid=False, cmap='plasma')
plt.legend(loc="right", bbox_to_anchor=(1, 1.1), ncol=2)
plt.title("Food between meals")
```

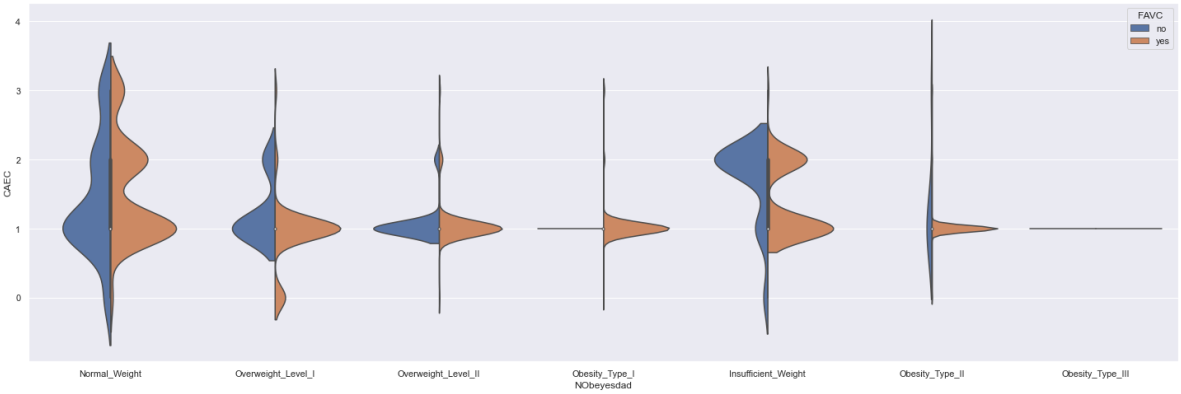
Out[248]:

```
Text(0.5, 1.0, 'Food between meals')
```



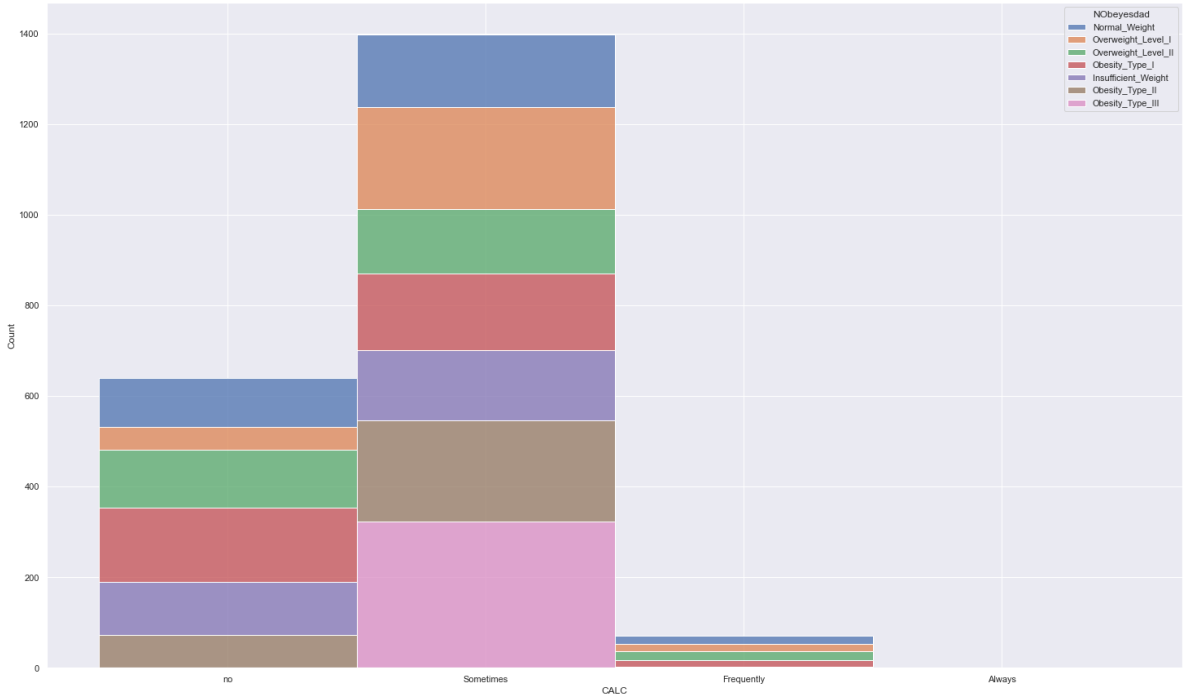
Entrée [40]:

```
dict_CAEC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
sns.violinplot(data = df_categorized, x = df_categorized['NObeyesdad'], y = df_categorized[
sns.set(rc = {'figure.figsize':(25,15)})
```



Entrée [41]:

```
sns.histplot(data = df_categorized, x = df_categorized['CALC'], hue = df_categorized['NObey
sns.set(rc = {'figure.figsize':(8,8)})
```

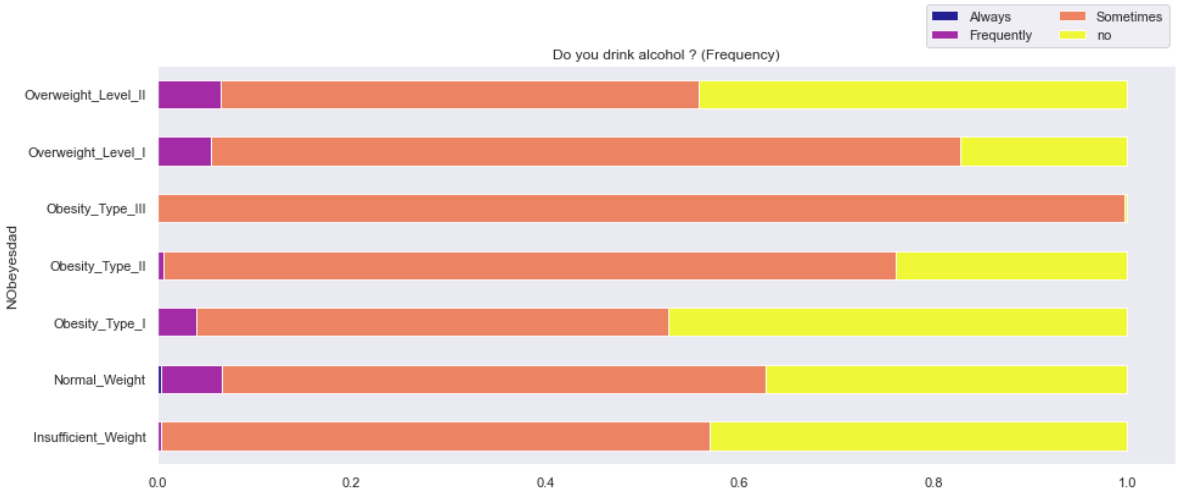


Entrée [42]:

```
big_count = pd.crosstab(df_categorized['NObeyesdad'], df_categorized['CALC'])
big_pct = big_count.div(big_count.sum(1), axis=0)
big_pct.plot.barh(stacked=True, figsize=(15, 6), alpha=0.9, grid=False, cmap='plasma')
plt.legend(loc="right", bbox_to_anchor=(1, 1.1), ncol=2)
plt.title("Do you drink alcohol ? (Frequency)")
```

Out[42]:

```
Text(0.5, 1.0, 'Do you drink alcohol ? (Frequency)')
```

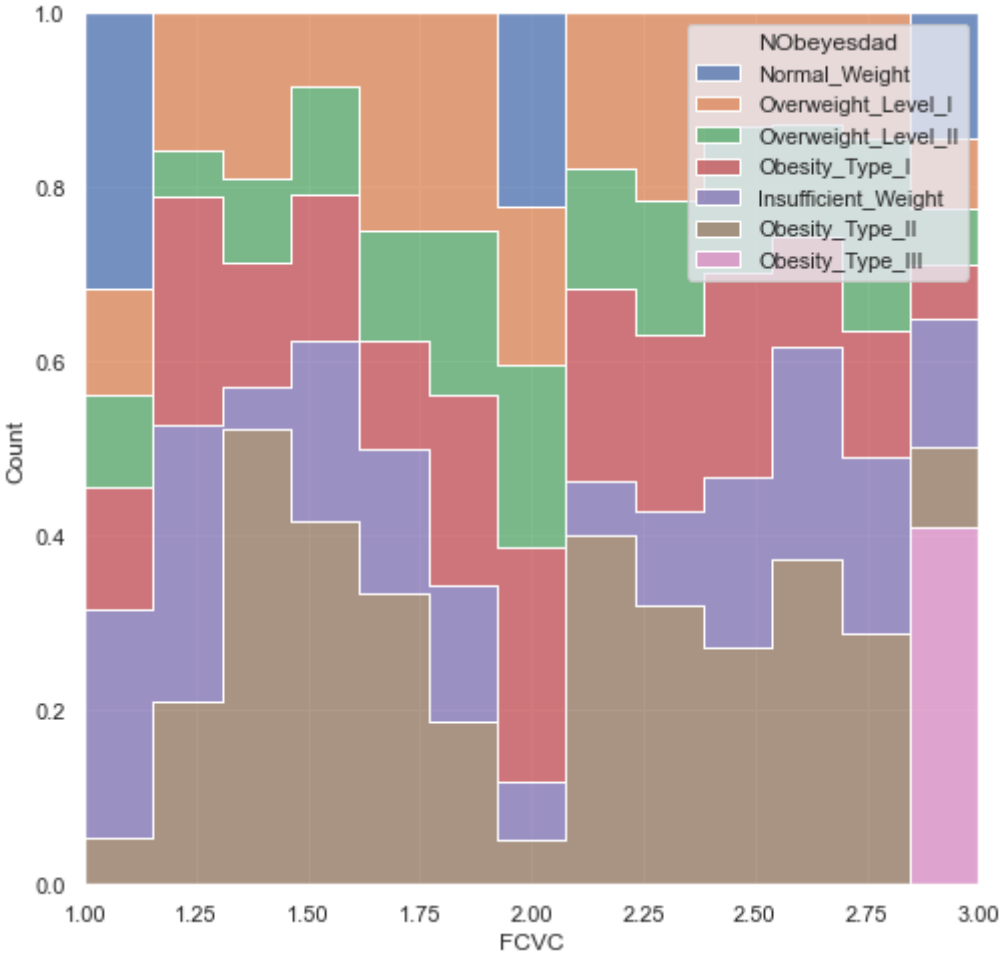


Entrée [43]:

```
sns.histplot(data = df_categorized, x = df_categorized['FCVC'], hue = df_categorized['NObey
```

Out[43]:

```
<AxesSubplot:xlabel='FCVC', ylabel='Count'>
```

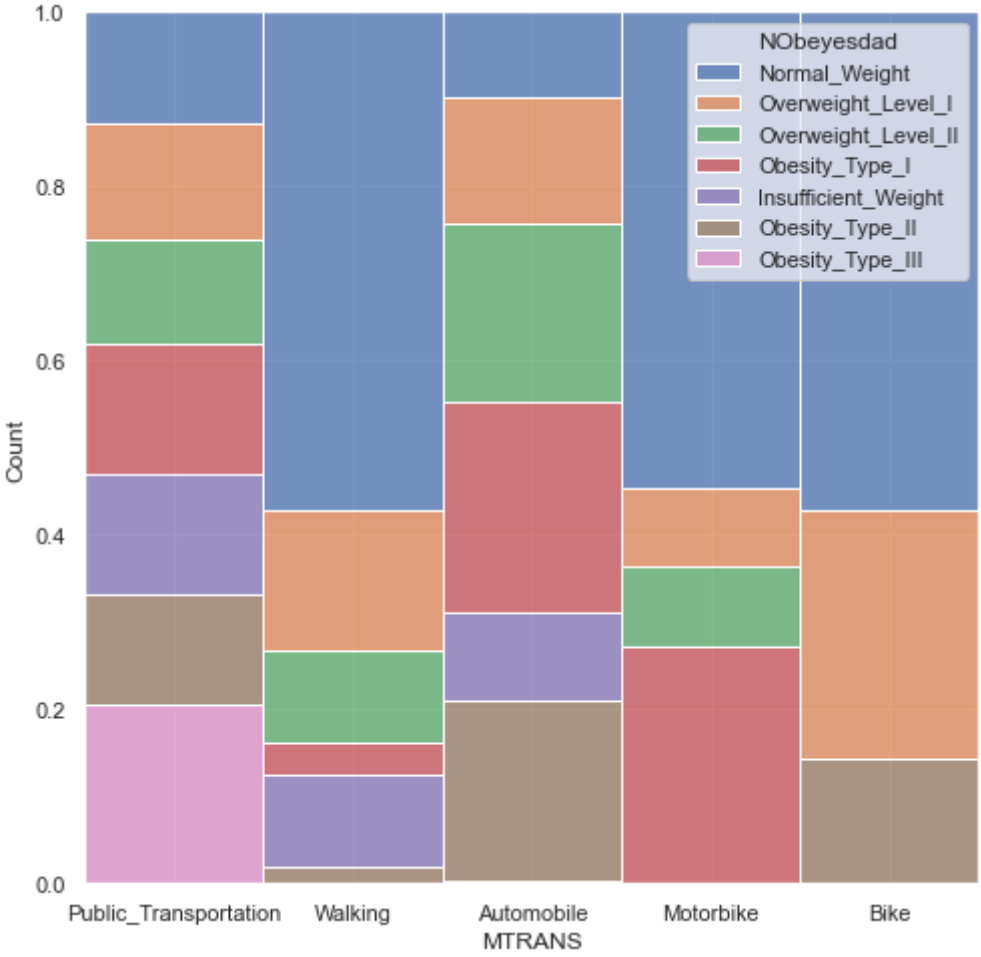


Entrée [44]:

```
sns.histplot(data = df_categorized, x = df_categorized['MTRANS'], hue = df_categorized['NOb
```

Out[44]:

```
<AxesSubplot:xlabel='MTRANS', ylabel='Count'>
```

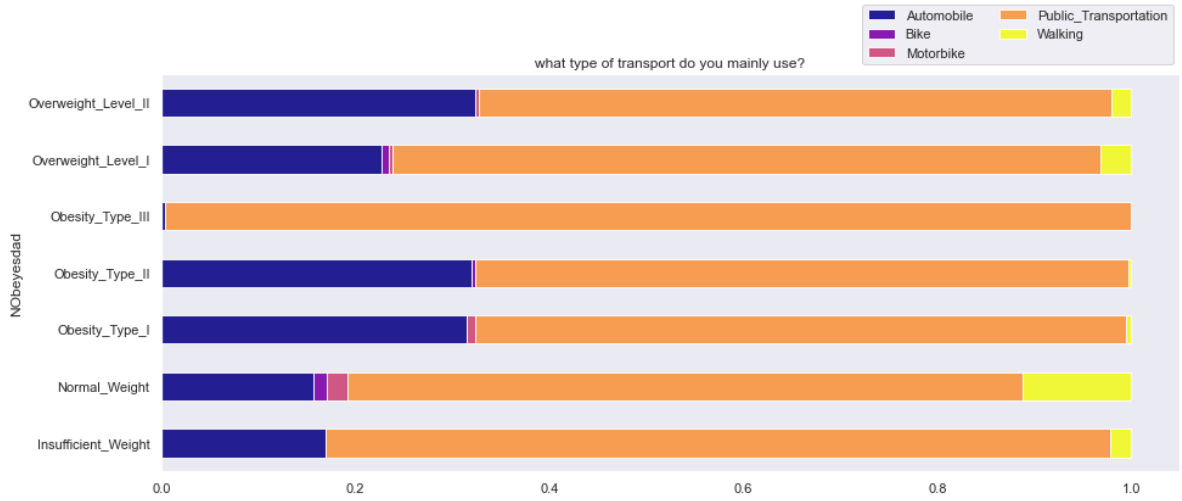


Entrée [48]:

```
big_count = pd.crosstab(df_categorized['NObeyesdad'], df_categorized['MTRANS'])
big_pct = big_count.div(big_count.sum(1), axis=0)
big_pct.plot.barh(stacked=True, figsize=(15, 6), alpha=0.9, grid=False, cmap='plasma')
plt.legend(loc="right", bbox_to_anchor=(1, 1.1), ncol=2)
plt.title("what type of transport do you mainly use?")
```

Out[48]:

```
Text(0.5, 1.0, 'what type of transport do you mainly use?')
```



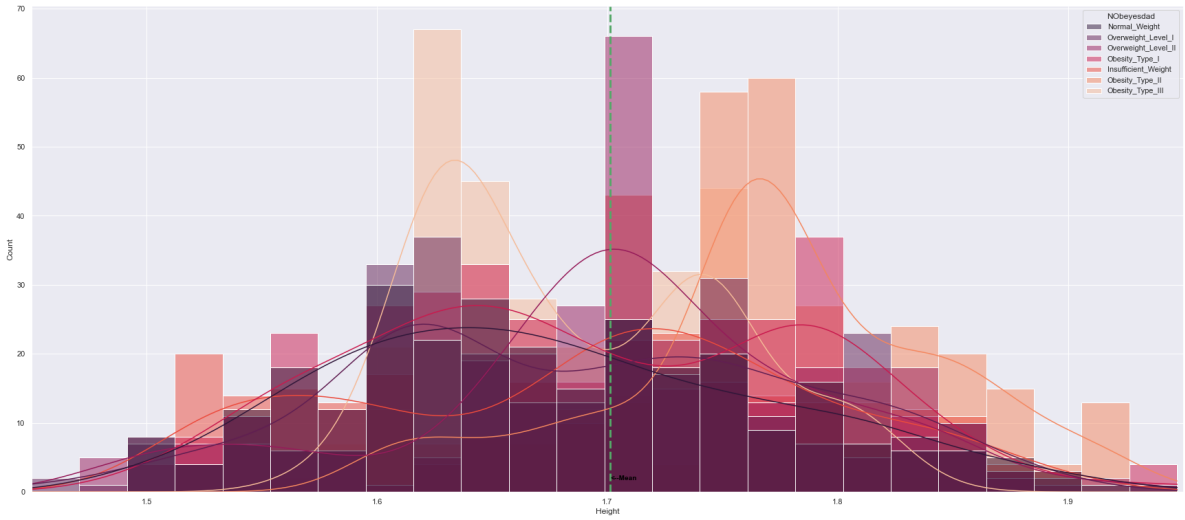
Entrée [49]:

```
def super_plot_for_WH(value, hue = 'NObeyesdad'):
    fig, ax = plt.subplots(1, 1, figsize=(30, 13))
    sns.histplot(x=value, hue=hue, data=df_categorized, kde=True, palette='rocket')
    ax.axvline(x=df_categorized[value].mean(), color='g', linestyle='--', linewidth=3)
    ax.text(df_categorized[value].mean(), df_categorized[value].mean(), "<--Mean", horizont
    if value=="Height":
        ax.set_xlim([1.45,1.95])
    elif value == 'Weight':
        ax.set_xlim([39,166])
    else :
        ax.set_xlim([14,62])

    sns.despine()
```

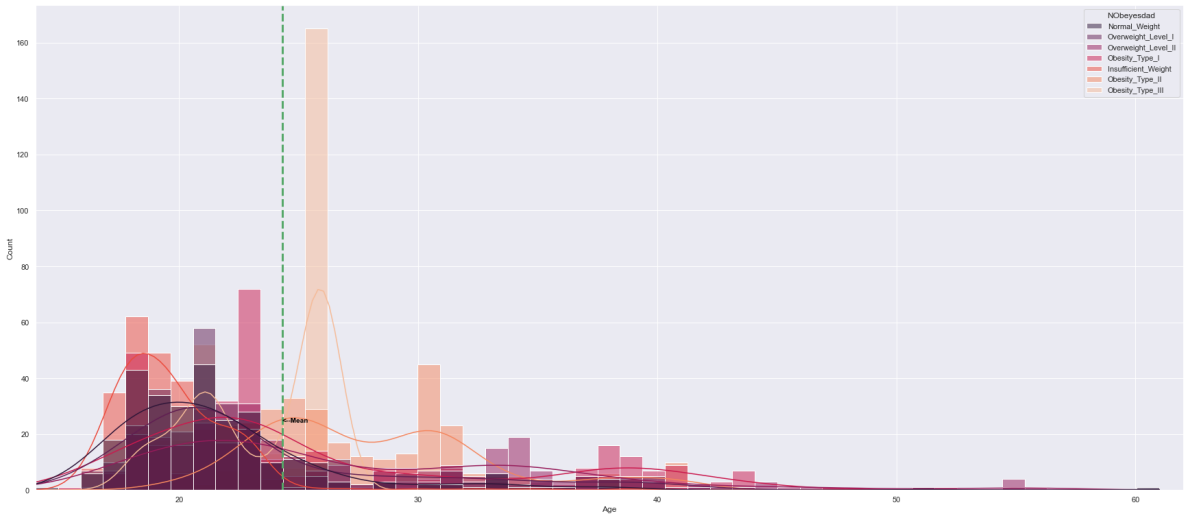
Entrée [50]:

```
super_plot_for_WH('Height')
```



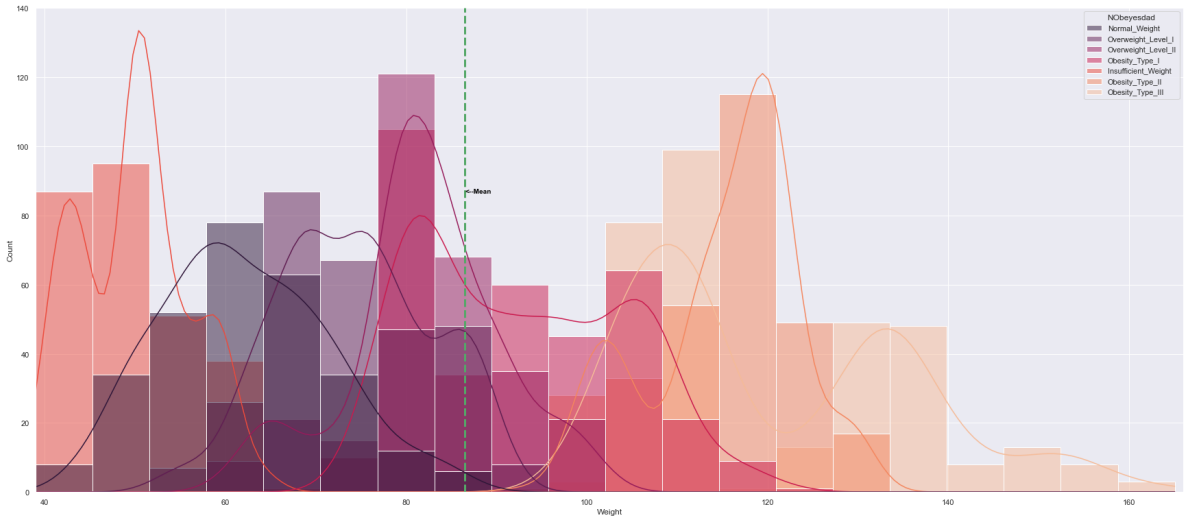
Entrée [51]:

```
super_plot_for_WH('Age')
```



Entrée [52]:

```
super_plot_for_WH('Weight')
```



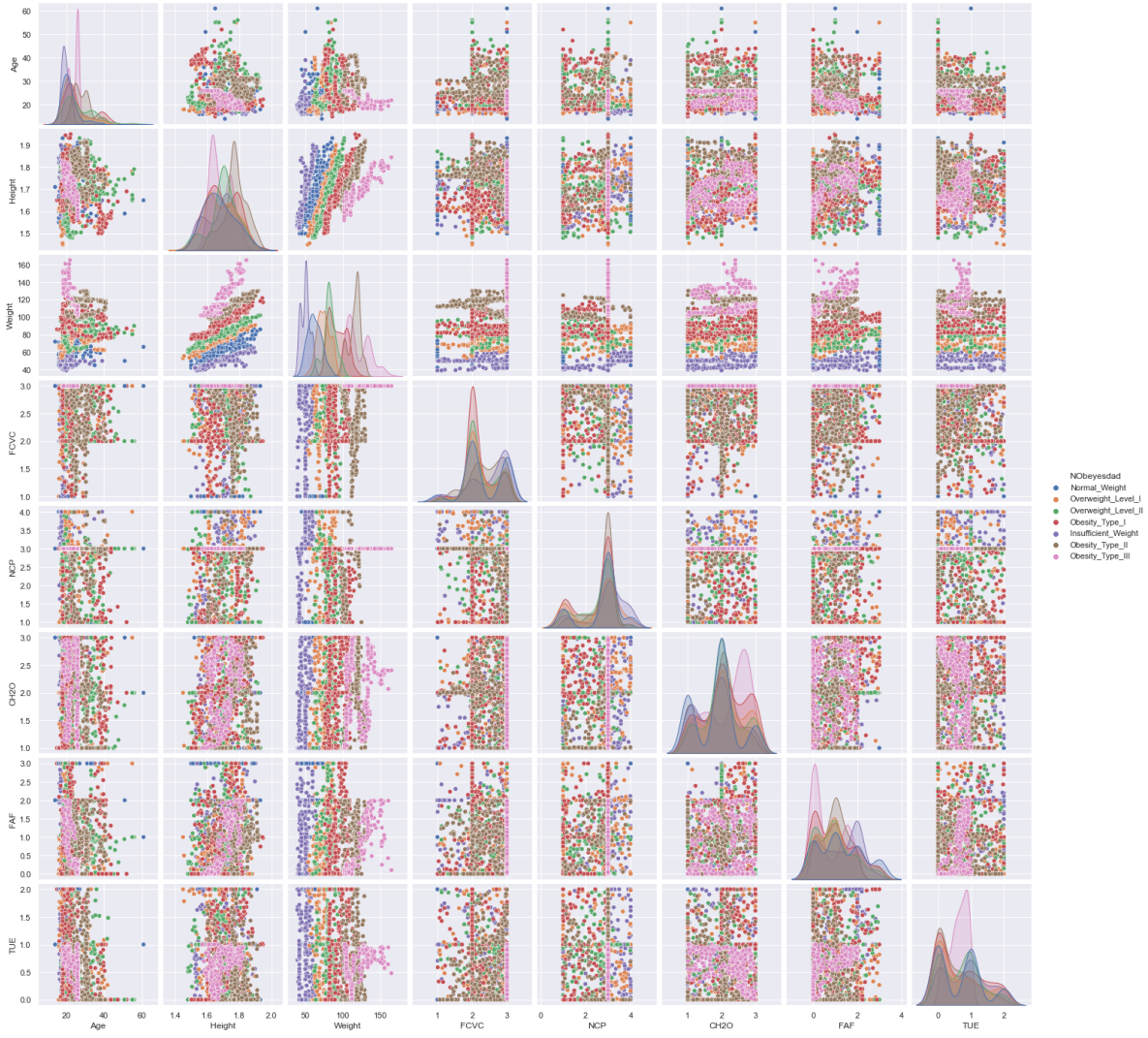
Entrée [53]:

```
sns.pairplot(df_categorized,hue='NObeyesdad')
```

```
C:\Users\fanny\anaconda3\lib\site-packages\seaborn\distributions.py:306: Use
rWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
C:\Users\fanny\anaconda3\lib\site-packages\seaborn\distributions.py:306: Use
rWarning: Dataset has 0 variance; skipping density estimate.
  warnings.warn(msg, UserWarning)
```

Out[53]:

```
<seaborn.axisgrid.PairGrid at 0x209f64f9b80>
```

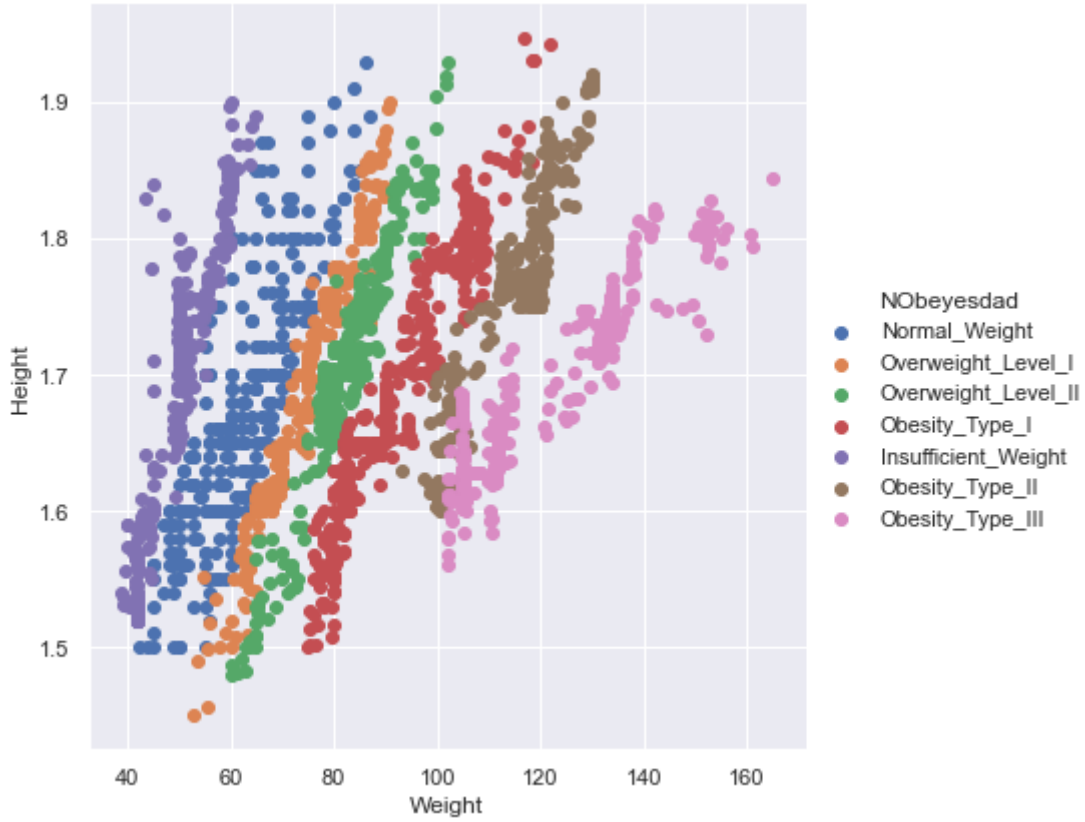


Entrée [54]:

```
sns.FacetGrid(df_categorized,hue="NObeyesdad",height=6).map(plt.scatter,"Weight","Height").
```

Out[54]:

```
<seaborn.axisgrid.FacetGrid at 0x209f67fcd30>
```



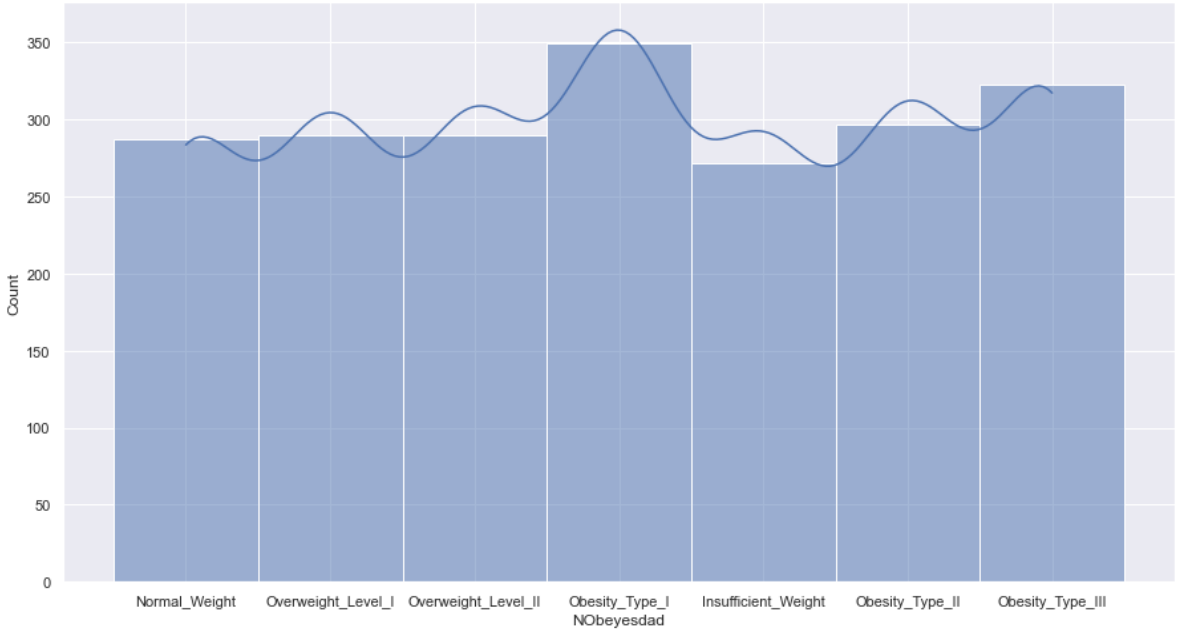
Target

Entrée [55]:

```
target = pd.DataFrame(df_categorized, columns = ['NObeyesdad'])
```

Entrée [57]:

```
sns.histplot(target['NObeyesdad'], kde=True)
sns.set(rc = {'figure.figsize':(15,15)})
```

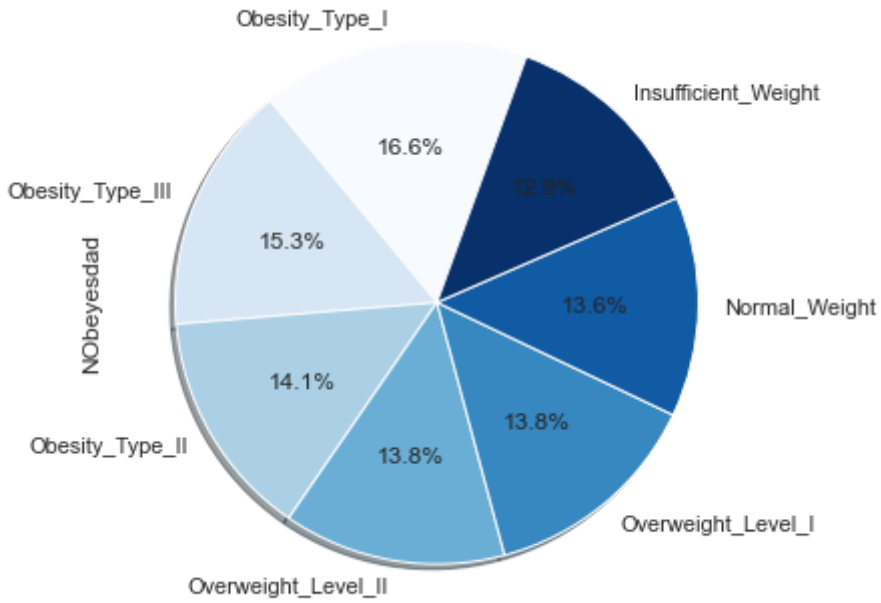


Entrée [58]:

```
fig, ax = plt.subplots(1, 1, figsize=(18, 6))
target['NObeyesdad'].value_counts().plot.pie(autopct='%1.1f%%',shadow=True, cmap="Blues", s
```

Out[58]:

```
<AxesSubplot:ylabel='NObeyesdad'>
```



The target variable is well balanced.

Data cleaning : part 2

Now, we can convert all our string variables into numeric ones (through categories): Let's first check what are the different possibilities for each string variables (Gender, family_history_with_overweight, 'FAVC', CAEC, SMOKE, SCC, CALC, MTRANS, NObesitydad)

Entrée [59]:

```
list_var = ['Gender','family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CAL
for x in list_var:
    print(df_categorized[x].unique())
```

```
['Female' 'Male']
['yes' 'no']
['no' 'yes']
['Sometimes' 'Frequently' 'Always' 'no']
['no' 'yes']
['no' 'yes']
['no' 'Sometimes' 'Frequently' 'Always']
['Public_Transportation' 'Walking' 'Automobile' 'Motorbike' 'Bike']
```

Create dictionnary with each string value and its numeric value

Entrée [63]:

```
dict_Gender = {'Female' : 0, 'Male' : 1}
dict_family_history_with_overweight = {'no' : 0, 'yes' : 1}
dict_FAVC = {'no' : 0, 'yes' : 1}
dict_CAEC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
dict_SMOKE = {'no' : 0, 'yes' : 1}
dict_SCC = {'no' : 0, 'yes' : 1}
dict_CALC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
dict_MTRANS = {'Public_Transportation' : 0, 'Walking' : 1, 'Automobile' : 2, 'Motorbike' :
dict_NObeyesdad = {'Normal_Weight' : 0, 'Overweight_Level_I' : 1, 'Overweight_Level_II' : 2
```

Entrée [61]:

```
for x in list_var:
    exec("df_categorized['"+ x +"'] = df_categorized['"+ x +"'].replace(dict_"+ x +")") #tr

df_categorized.head()
```

Out[61]:

ge	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	S
.0	1.62	64.0	1	0	2.0	3.0	1	0	2.0	
.0	1.52	56.0	1	0	3.0	3.0	1	1	3.0	
.0	1.80	77.0	1	0	2.0	3.0	1	0	2.0	
.0	1.80	87.0	0	0	3.0	3.0	1	0	2.0	
.0	1.78	89.8	0	0	2.0	1.0	1	0	2.0	

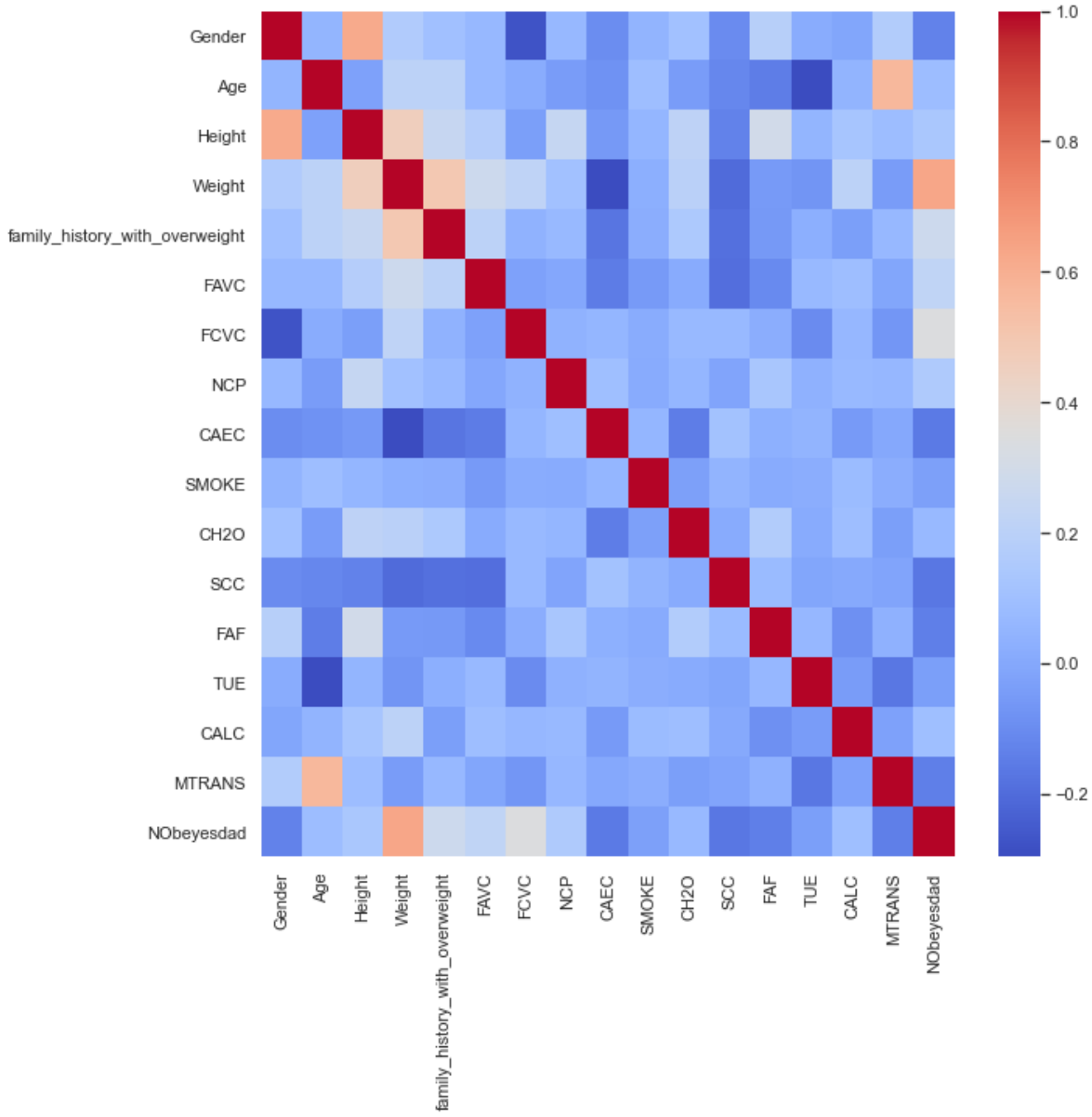
As we can see, each string values have been convert into numeric value according to their refered dictionnary.

Entrée [67]:

```
dfbis = df_categorized
dfbis["NObeyesdad"] = dfbis["NObeyesdad"].replace(dict_NObeyesdad)
fig, ax = plt.subplots(figsize = (10,10))
sns.heatmap(dfbis.corr(), cmap = 'coolwarm')
```

Out[67]:

<AxesSubplot:>



We can see that every variable is globaly lineraly independant from each other. Because of that, ACP will not be efficient, so we do not perform it. We will select the variable correlated with less than 30%.

Data Processing

Data Normalisation

We perform a normalisation of the dataset, because some model will required the dataset to be formatted

Entrée [68]:

```
df_features = df_categorized.drop(columns = 'NObeyesdad')
```

Entrée [69]:

```
scaler = preprocessing.StandardScaler().fit(df_features)
df_scaled = pd.DataFrame(scaler.transform(df_features))
df_scaled['NObesity'] = df['NObeyesdad']
df_scaled.columns = df.columns.to_list()
df_scaled.head()
```

Out[69]:

P	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NC
7	-0.298887	-0.146006	-0.011550	-0.218435	-1.187188	0.562523	-1.417696	-0.563018	Norr
7	-0.298887	6.849021	1.621152	4.578027	2.339294	-1.079300	0.521631	-0.563018	Norr
7	-0.298887	-0.146006	-0.011550	-0.218435	1.163800	0.562523	2.460958	-0.563018	Norr
7	-0.298887	-0.146006	-0.011550	-0.218435	1.163800	-1.079300	2.460958	0.588138	Overweig
9	-0.298887	-0.146006	-0.011550	-0.218435	-1.187188	-1.079300	0.521631	-0.563018	Overweig

We now have our dataset scaled

Modeling

Entrée [70]:

```
df_scaled_features = df_scaled.drop(columns = 'NObeyesdad')
```

Training / Testing Sets

Entrée [73]:

```
X_train, X_test, Y_train, Y_test = train_test_split(df_scaled_features, df_scaled['NObeyesd
```

We used the above method, but we cannot find the same data as when we developed the models. Since we had the best performances with the first sets we have tried, we decided to import them thanks to the csv that we had created at the time. But you can choose to use the sets from the above code. However, there is a 3% loss of accuracy on average. We are aware that this problem could have been fixed with rd.seed(32) but we did not think that our first set would have such a difference in performance with the other sets.

Entrée [79]:

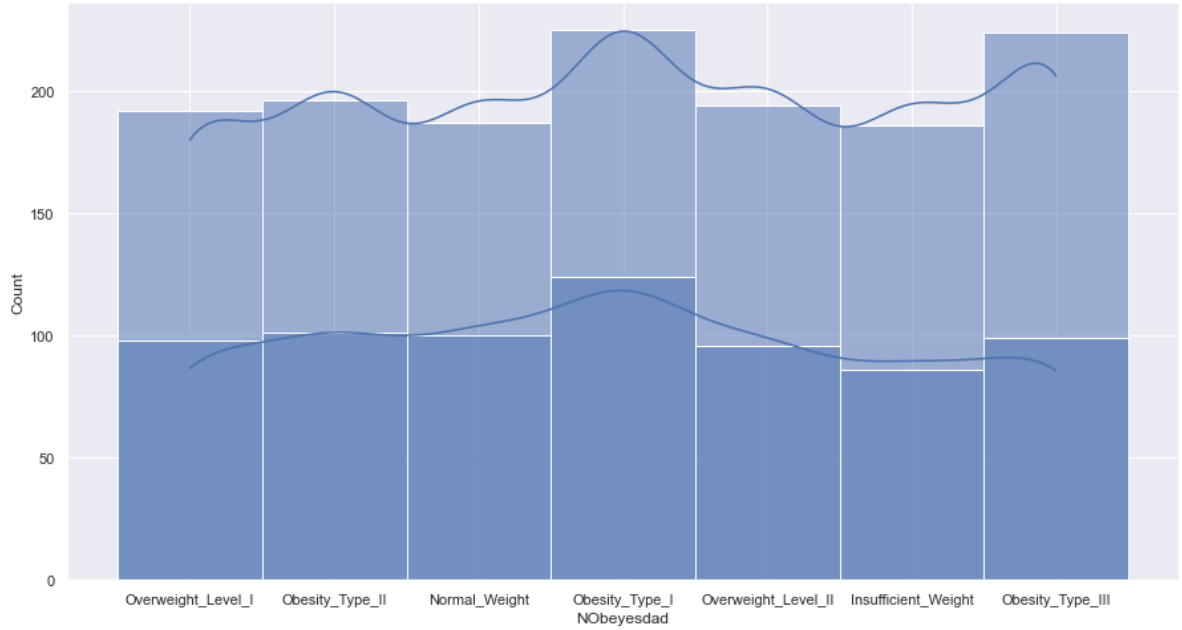
```
X_train = pd.read_csv("X_train.csv")
Y_train = pd.read_csv("Y_train.csv")
X_test = pd.read_csv("X_test.csv")
Y_test = pd.read_csv("Y_test.csv")
Y_train = Y_train['NObeyesdad']
Y_test = Y_test['NObeyesdad']
```

Entrée [80]:

```
sns.set(rc = {'figure.figsize':(15,8)})
sns.histplot(Y_train, kde=True)
sns.histplot(Y_test, kde=True)
```

Out[80]:

```
<AxesSubplot:xlabel='NObeyesdad', ylabel='Count'>
```



The split is correct, because it is still well balanced.

Model

These models and their performance will be classified and stock in the list "perf"

Entrée [81]:

```
perf=[]
```

Logistic regression

Entrée [82]:

```
logisticRegr = LogisticRegression(solver='lbfgs', max_iter=1000)
logisticRegr.fit(X_train, Y_train)
y_pred = logisticRegr.predict(X_test)
```

Entrée [83]:

```
print("confusion matrix")
print(confusion_matrix(Y_test, y_pred))
print("model summary : ")
print(classification_report(Y_test, y_pred))
score = accuracy_score(Y_test, y_pred)
print("accuracy : "+str(score))
perf.append([score,"Logistic regression"])
```

```
confusion matrix
[[ 82   4   0   0   0   0   0]
 [  8  67   0   0   0  15  10]
 [  0   0 109   7   0   1   7]
 [  0   0   1 100   0   0   0]
 [  0   0   1   0  98   0   0]
 [  0   3   0   0   0  81  14]
 [  0   1  12   0   0  10  73]]
model summary :
                    precision    recall  f1-score   support

Insufficient_Weight       0.91      0.95      0.93        86
      Normal_Weight       0.89      0.67      0.77       100
     Obesity_Type_I       0.89      0.88      0.88       124
    Obesity_Type_II       0.93      0.99      0.96       101
   Obesity_Type_III       1.00      0.99      0.99        99
 Overweight_Level_I       0.76      0.83      0.79        98
Overweight_Level_II       0.70      0.76      0.73        96

           accuracy                           0.87       704
          macro avg       0.87      0.87      0.87       704
       weighted avg       0.87      0.87      0.87       704

accuracy : 0.8664772727272727
```

Gradient Boosting Classifier

Entrée [84]:

```
gbdt = GradientBoostingClassifier(random_state=123)
gbdt.fit(X_train,Y_train)
gbdt_pre = gbdt.predict(X_test)
```

Entrée [85]:

```
print("confusion matrix")
print(confusion_matrix(Y_test, gbdt_pre))
print("model summary : ")
print(classification_report(Y_test, gbdt_pre))
score = accuracy_score(Y_test, gbdt_pre)
print("accuracy : "+str(score))
perf.append([score,"Gradient Boosting Classifier"])
```

```
confusion matrix
[[ 84   2   0   0   0   0   0]
 [  0  93   0   0   0   7   0]
 [  0   0 120   1   0   0   3]
 [  0   0   2  99   0   0   0]
 [  0   0   0   1  98   0   0]
 [  0   1   0   0   0  96   1]
 [  0   2   0   0   0   2  92]]
model summary :
                    precision    recall  f1-score   support

Insufficient_Weight       1.00      0.98      0.99        86
      Normal_Weight       0.95      0.93      0.94       100
     Obesity_Type_I       0.98      0.97      0.98       124
    Obesity_Type_II       0.98      0.98      0.98       101
   Obesity_Type_III       1.00      0.99      0.99        99
 Overweight_Level_I       0.91      0.98      0.95        98
Overweight_Level_II       0.96      0.96      0.96        96

           accuracy                           0.97       704
          macro avg       0.97      0.97      0.97       704
       weighted avg       0.97      0.97      0.97       704

accuracy : 0.96875
```

KNN

Entrée [86]:

```
#function to determine the best possible k
stat=[]
k=[]
for i in range(2,20):#les k sont testé de 2 à 20 car il est très peu probable que k>20

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,Y_train)
    pred_knn = knn.predict(X_test)
    stat.append([accuracy_score(Y_test, pred_knn),i])
k=sorted(stat, reverse = True)
  #  print(k)

print(k)
```

```
[[0.8295454545454546, 2], [0.8196022727272727, 3], [0.796875, 5], [0.7911931
818181818, 4], [0.78125, 6], [0.7784090909090909, 7], [0.7642045454545454,
9], [0.7642045454545454, 8], [0.7571022727272727, 12], [0.7556818181818182,
10], [0.7485795454545454, 11], [0.7471590909090909, 13], [0.742897727272727
3, 14], [0.7386363636363636, 16], [0.7372159090909091, 18], [0.734375, 17],
[0.7329545454545454, 15], [0.7301136363636364, 19]]
```

Entrée [87]:

```
#According to our last result k=2 is the best choice for our model
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train,Y_train)
pred_knn = knn.predict(X_test)
```

Entrée [88]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test, pred_knn))
print("model summary : ")
print(classification_report(Y_test, pred_knn))
score = accuracy_score(Y_test, pred_knn)
print("accuracy : "+str(score))
perf.append([score,"KNN"])
```

```
matrice de confusion
[[ 80   4   0   0   0   2   0]
 [ 31  47   3   1   0  10   8]
 [  1   1 112   4   0   3   3]
 [  0   0   3  98   0   0   0]
 [  0   1   0   0  97   1   0]
 [  2   6   7   1   0  80   2]
 [  2   6  10   2   0   6  70]]
model summary :
                    precision    recall  f1-score   support

Insufficient_Weight       0.69      0.93      0.79        86
      Normal_Weight       0.72      0.47      0.57       100
     Obesity_Type_I       0.83      0.90      0.86       124
    Obesity_Type_II       0.92      0.97      0.95       101
   Obesity_Type_III       1.00      0.98      0.99        99
 Overweight_Level_I       0.78      0.82      0.80        98
Overweight_Level_II       0.84      0.73      0.78        96

           accuracy                           0.83       704
          macro avg       0.83      0.83      0.82       704
       weighted avg       0.83      0.83      0.82       704

accuracy : 0.8295454545454546
```

Support vector machine

Entrée [89]:

```
clf = svm.SVC()
clf.fit(X_train, Y_train)
pred_svm = clf.predict(X_test)
```

Entrée [90]:

```
scores=cross_val_score(clf,X_train,Y_train,n_jobs=-1)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
[0.84341637 0.87544484 0.85053381 0.86476868 0.83214286]
0.85 accuracy with a standard deviation of 0.02
```

Entrée [92]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test,pred_svm))
print("model summary : ")
print(classification_report(Y_test,pred_svm))
score = accuracy_score(Y_test, pred_svm)
print("accuracy : "+str(score))
perf.append([score,"SVM"])
```

```
matrice de confusion
[[ 76  10   0   0   0   0   0]
 [  8  71   0   0   0  13   8]
 [  0   6 112   2   0   2   2]
 [  0   1   1  99   0   0   0]
 [  0   1   0   0  97   0   1]
 [  0  11   1   0   0  81   5]
 [  0   5   7   0   0   6  78]]
model summary :
                     precision    recall  f1-score   support

Insufficient_Weight       0.90      0.88      0.89        86
      Normal_Weight       0.68      0.71      0.69       100
     Obesity_Type_I       0.93      0.90      0.91       124
    Obesity_Type_II       0.98      0.98      0.98       101
   Obesity_Type_III       1.00      0.98      0.99        99
 Overweight_Level_I       0.79      0.83      0.81        98
Overweight_Level_II       0.83      0.81      0.82        96

           accuracy                           0.87       704
          macro avg       0.87      0.87      0.87       704
       weighted avg       0.87      0.87      0.87       704

accuracy : 0.8721590909090909
```

Entrée [93]:

```
parameters = {'C':[4,2,3,5,6], "degree":[1,3,5,2,4],'gamma':[0.5,0.1,0.15,0.6,0.8,0.9],"ker
grid=GridSearchCV(svm.SVC(),parameters,n_jobs=-1)
grid.fit(X_train,Y_train)
print(grid.best_score_,grid.best_estimator_)
```

```
0.9579613624809354 SVC(C=6, degree=1, gamma=0.9, kernel='poly')
```

Entrée [94]:

```
clf2 = svm.SVC(C=6, degree=1, gamma=0.9, kernel='poly')
clf2.fit(X_train, Y_train)
pred_svm2= clf2.predict(X_test)
```

Entrée [95]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test,pred_svm2))
print("model summary : ")
print(classification_report(Y_test,pred_svm2))
score = accuracy_score(Y_test, pred_svm2)
print("accuracy : "+str(score))
perf.append([score,"SVM2"])
```

```
matrice de confusion
[[ 86   0   0   0   0   0   0]
 [  2  88   0   0   0  10   0]
 [  0   0 120   1   0   0   3]
 [  0   0   1 100   0   0   0]
 [  0   0   0   1  98   0   0]
 [  0   1   0   0   0  93   4]
 [  0   0   3   0   0   6  87]]
model summary :
                     precision    recall  f1-score   support

Insufficient_Weight       0.98      1.00      0.99        86
      Normal_Weight       0.99      0.88      0.93       100
     Obesity_Type_I       0.97      0.97      0.97       124
    Obesity_Type_II       0.98      0.99      0.99       101
   Obesity_Type_III       1.00      0.99      0.99        99
 Overweight_Level_I       0.85      0.95      0.90        98
Overweight_Level_II       0.93      0.91      0.92        96

           accuracy                           0.95       704
          macro avg       0.96      0.95      0.95       704
       weighted avg       0.96      0.95      0.95       704

accuracy : 0.9545454545454546
```

Naive Bayes

Entrée [96]:

```
gnb = GaussianNB()
y_gnb = gnb.fit(X_train, Y_train).predict(X_test)
```

Entrée [97]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test,y_gnb))
print("model summary : ")
print(classification_report(Y_test,y_gnb))
score = accuracy_score(Y_test, y_gnb)
print("accuracy : "+str(score))
perf.append([score,"Naive Bayes"])
```

```
matrice de confusion
[[ 85   1   0   0   0   0   0]
 [ 79   6   4   2   0   6   3]
 [  0   1  51  64   0   4   4]
 [  0   1   0 100   0   0   0]
 [  0   1   0   0  97   0   1]
 [ 39   0  31  21   0   7   0]
 [ 16   2  26  38   0   3  11]]
model summary :
                    precision    recall  f1-score   support

Insufficient_Weight       0.39      0.99      0.56        86
      Normal_Weight       0.50      0.06      0.11       100
     Obesity_Type_I       0.46      0.41      0.43       124
    Obesity_Type_II       0.44      0.99      0.61       101
   Obesity_Type_III       1.00      0.98      0.99        99
 Overweight_Level_I       0.35      0.07      0.12        98
Overweight_Level_II       0.58      0.11      0.19        96

           accuracy                           0.51       704
          macro avg       0.53      0.52      0.43       704
       weighted avg       0.53      0.51      0.43       704

accuracy : 0.5071022727272727
```

Random Forest

Entrée [98]:

```
rf = RandomForestClassifier(n_estimators = 500)
rf.fit(X_train, Y_train)
rf_pred=rf.predict(X_test)
```

Entrée [99]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test,rf_pred))
print("model summary : ")
print(classification_report(Y_test,rf_pred))
score = accuracy_score(Y_test, rf_pred)
print("accuracy : "+str(score))
perf.append([score,"Random Forest"])
```

```
matrice de confusion
[[ 80   6   0   0   0   0   0]
 [  0  88   0   0   0   8   4]
 [  0   1 118   0   0   1   4]
 [  0   0   1 100   0   0   0]
 [  0   0   0   0  99   0   0]
 [  0   5   0   0   0  91   2]
 [  0   4   1   0   0   3  88]]
model summary :
                     precision    recall  f1-score   support

Insufficient_Weight       1.00      0.93      0.96        86
      Normal_Weight       0.85      0.88      0.86       100
     Obesity_Type_I       0.98      0.95      0.97       124
    Obesity_Type_II       1.00      0.99      1.00       101
   Obesity_Type_III       1.00      1.00      1.00        99
 Overweight_Level_I       0.88      0.93      0.91        98
Overweight_Level_II       0.90      0.92      0.91        96

           accuracy                           0.94       704
          macro avg       0.94      0.94      0.94       704
       weighted avg       0.95      0.94      0.94       704

accuracy : 0.9431818181818182
```

XGboost

Entrée [100]:

```
dict_NObeyesdad = {'Normal_Weight' : 0, 'Overweight_Level_I' : 1, 'Overweight_Level_II' : 2
Y_test_int=Y_test.replace(dict_NObeyesdad)
Y_train_int=Y_train.replace(dict_NObeyesdad)
```

Entrée [101]:

```
model = XGBClassifier()
model.fit(X_train, Y_train_int)
y_xgb=model.predict(X_test)
```

```
C:\Users\fanny\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarn
ing: The use of label encoder in XGBClassifier is deprecated and will be rem
oved in a future release. To remove this warning, do the following: 1) Pass
option use_label_encoder=False when constructing XGBClassifier object; and
2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [n
um_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[21:26:10] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'multi:softprob' was changed from 'merror' to
'mlogloss'. Explicitly set eval_metric if you'd like to restore the old beha
vior.
```

Entrée [102]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test_int,y_xgb))
print("model summary : ")
print(classification_report(Y_test_int,y_xgb))
score = accuracy_score(Y_test_int, y_xgb)
print("accuracy : "+str(score))
perf.append([score,"XGBoost"])
```

```
matrice de confusion
[[ 87  12   1   0   0   0   0]
 [  2  95   1   0   0   0   0]
 [  3   1  89   3   0   0   0]
 [  0   0   2 122   0   0   0]
 [  2   0   0   0  84   0   0]
 [  0   0   0   1   0 100   0]
 [  0   0   0   0   0   1  98]]
model summary :
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       100
           1       0.88      0.97      0.92        98
           2       0.96      0.93      0.94        96
           3       0.97      0.98      0.98       124
           4       1.00      0.98      0.99        86
           5       0.99      0.99      0.99       101
           6       1.00      0.99      0.99        99

    accuracy                           0.96       704
   macro avg       0.96      0.96      0.96       704
weighted avg       0.96      0.96      0.96       704

accuracy : 0.9588068181818182
```

Entrée [103]:

```
dtrain = xgb.DMatrix(X_train, label=Y_train_int)
dvalid = xgb.DMatrix(X_test, label=Y_test_int)
watchlist = [(dvalid, 'valid'), (dtrain, 'train')]
num_class=7
param = {'objective':'multi:softmax',
         'eta':'0.3', 'max_depth':10,
         'silent':1, 'nthread':-1,
         'num_class':num_class,
         'eval_metric':'merror'}
model = xgb.train(param, dtrain, 60, watchlist, early_stopping_rounds=20,
                  verbose_eval=10)
y_xgb2=model.predict(dvalid)
y_xgb2 = [int(x) for x in y_xgb2]
```

```
[21:26:15] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actua
lly being used
  but getting flagged wrongly here. Please open an issue if you find any suc
h cases.


[0]     valid-merror:0.08807    train-merror:0.03561
[10]    valid-merror:0.05114    train-merror:0.00000
[20]    valid-merror:0.03835    train-merror:0.00000
[27]    valid-merror:0.03835    train-merror:0.00000
```

Entrée [104]:

```
print("matrice de confusion")
print(confusion_matrix(Y_test_int,y_xgb2))
print("model summary : ")
print(classification_report(Y_test_int,y_xgb2))
score = accuracy_score(Y_test_int, y_xgb2)
print("accuracy : "+str(score))
perf.append([score,"XGBoost2"])
```

```
matrice de confusion
[[ 92   7   1   0   0   0   0]
 [  4  94   0   0   0   0   0]
 [  3   2  88   3   0   0   0]
 [  0   0   2 122   0   0   0]
 [  2   0   0   0  84   0   0]
 [  0   0   0   1   0 100   0]
 [  0   0   0   0   0   1  98]]
model summary :
              precision    recall  f1-score   support

           0       0.91      0.92      0.92       100
           1       0.91      0.96      0.94        98
           2       0.97      0.92      0.94        96
           3       0.97      0.98      0.98       124
           4       1.00      0.98      0.99        86
           5       0.99      0.99      0.99       101
           6       1.00      0.99      0.99        99

    accuracy                           0.96       704
   macro avg       0.96      0.96      0.96       704
weighted avg       0.96      0.96      0.96       704

accuracy : 0.9630681818181818
```

Ranking of the best prediction models

Entrée [105]:

```
perf=sorted(perf, reverse = True)
for i in range(len(perf)):
    print(str(i+1)+"-" + str(perf[i][1])+" with an accuracy of :"+ str(round(perf[i][0]*100
```

```
1-Gradient Boosting Classifier with an accuracy of :96.88%
2-XGBoost2 with an accuracy of :96.31%
3-XGBoost with an accuracy of :95.88%
4-SVM2 with an accuracy of :95.45%
5-Random Forest with an accuracy of :94.32%
6-SVM with an accuracy of :87.22%
7-SVM with an accuracy of :87.22%
8-Logistic regression with an accuracy of :86.65%
9-KNN with an accuracy of :82.95%
10-Naive Bayes with an accuracy of :50.71%
```

For our model and API we will use the gradient boosting classifier model because it is the one that has the best

performance.

Dash and API

After ranking the best algorithms, we now know that we must use gradient boosting. We have handed over all the lines of codes you need to launch the API. It may seem redundant with code segments that you may have seen before, but you just have to run this part there for the API to work (this was in case you were only running the API in our code). .

Entrée [107]:

```
#df = pd.read_csv('ObesityDataSet_raw_and_data_sinthetic.csv')
dt= pd.read_csv("ObesityDataSet.csv")
df= pd.read_csv("ObesityDataSet.csv")
dict_Gender = {'Female' : 0, 'Male' : 1}
dict_family_history_with_overweight = {'no' : 0, 'yes' : 1}
dict_FAVC = {'no' : 0, 'yes' : 1}
dict_CAEC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
dict_SMOKE = {'no' : 0, 'yes' : 1}
dict_SCC = {'no' : 0, 'yes' : 1}
dict_CALC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
dict_MTRANS = {'Public_Transportation' : 0, 'Walking' : 1, 'Automobile' : 2, 'Motorbike' :
list_var = ['Gender','family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE', 'SCC', 'CAL
for x in list_var:
    exec("dt['"+ x +"'] = dt['"+ x +"'].replace(dict_"+ x +")") #transformation for each st
dt = dt.drop(columns = 'NObeyesdad')
dt_col = dt.copy
```

Entrée [108]:

```
X_train = pd.read_csv("X_train.csv")
Y_train = pd.read_csv("Y_train.csv")
X_test = pd.read_csv("X_test.csv")
Y_test = pd.read_csv("Y_test.csv")
Y_train = Y_train['NObeyesdad']
Y_test = Y_test['NObeyesdad']
X = pd.concat([X_train,X_test], ignore_index=True)
Y = pd.concat([Y_train,Y_test], ignore_index=True)
```

Entrée [109]:

```
gbdt = GradientBoostingClassifier(random_state=123)
gbdt.fit(X,Y)
```

Out[109]:

```
GradientBoostingClassifier(random_state=123)
```

Entrée [110]:

```
dict_Gender = {'Female' : 0, 'Male' : 1}
dict_family_history_with_overweight = {'no' : 0, 'yes' : 1}
dict_FAVC = {'no' : 0, 'yes' : 1}
dict_FCVC = {'Never' : 1, 'Sometimes' : 2, 'Always' : 3}
dict_CAEC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
dict_SMOKE = {'no' : 0, 'yes' : 1}
dict_SCC = {'no' : 0, 'yes' : 1}
dict_CALC = {'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3}
dict_MTRANS = {'Public_Transportation' : 0, 'Walking' : 1, 'Automobile' : 2, 'Motorbike' :
dict_NCP = {'Between 1 and 2' : 1, '3' : 3, 'more than 3' : 4}
dict_FAF = {'I do not have' : 0, '1 or 2 days' : 1, '2 or 4 days' : 2, '4 or 5 days' : 3}
dict_TUE = {'0-2 hours' : 0, '3-5 hours' : 1, 'More than 5 hours' : 2}
dict_CH2O = {'Less than a liter' : 0, 'Between 1 and 2 L' : 1, 'More than 2 L' : 2}

list_feat = ['Gender','family_history_with_overweight', 'FAVC','FCVC', 'CAEC', 'SMOKE', 'SC
```

Entrée [111]:

```
def predict_value(n,selected_gender,selected_age,selected_height,selected_weight,selected_f
    data = pd.DataFrame([[selected_gender,selected_age,selected_height,selected_weight,sele
    data.columns = dt.columns
    for x in list_feat:
        exec("data['"+ x +"'] = data['"+ x +"'].replace(dict_"+ x +")") #transformation for
    dataset = pd.concat([dt,data], ignore_index=True)
    scaler = preprocessing.StandardScaler().fit(dataset)
    dataset_scaled = pd.DataFrame(scaler.transform(dataset))
    dataset_scaled.columns = dt.columns.to_list()
    pred = gbdt.predict(dataset_scaled.tail(1))
    return pred[0]
```

Entrée [*]:

```
pred_value = ''

if __name__ == '__main__':

    app = dash.Dash(__name__)

    app.layout = html.Div(children=[
                            #Title
                            html.H1(
                                children='Health check',
                                className = 'header'
                            ),

                            #gender
                            html.Div(children =[
                                html.P(children = 'What is your gender ?'),

                                dcc.Input(
                                id = 'selected_gender',
                                type = 'text',
                                list = 'liste_gender',
                                placeholder = 'Gender',
                                debounce = False,
                                spellCheck = True,
                                ),
                            ]),

                            #age
                            html.Div(children =[
                                html.P(children = 'What is your age?'),

                                dcc.Input(
                                id = 'selected_age',
                                type = 'text',
                                placeholder = 'Age',
                                debounce = False,
                                spellCheck = True,
                                inputMode = 'numeric'
                                ),
                            ]),

                            #height
                            html.Div(children =[
                                html.P(children = 'What is your height?'),

                                dcc.Input(
                                id = 'selected_height',
                                type = 'text',
                                placeholder = 'Height (meter)',
                                debounce = False,
                                spellCheck = True,
                                inputMode = 'numeric'
                                ),
                            ]),

                            #weight
                            html.Div(children =[
                                html.P(children = 'What is your weight?'),
```

```
                    dcc.Input(
                    id = 'selected_weight',
                    type = 'text',
                    placeholder = 'Weight (kg)',
                    debounce = False,
                    spellCheck = True,
                    inputMode = 'numeric'
                    ),
                ]),

                #family overweight
                html.Div(children =[
                    html.P(children = 'Has a family member suffered or suffers

                    dcc.Input(
                    id = 'selected_fam_hist',
                    type = 'text',
                    list = 'liste_family_history',
                    placeholder = 'Family history with overweight',
                    debounce = False,
                    spellCheck = True,
                    ),
                ]),

                #facv
                html.Div(children =[
                    html.P(children = 'Do you eat high caloric food frequently?

                    dcc.Input(
                    id = 'selected_favc',
                    type = 'text',
                    list = 'liste_favc',
                    placeholder = 'Consumption of high caloric food',
                    debounce = False,
                    spellCheck = True,
                    ),
                ]),

                #fccv
                html.Div(children =[
                    html.P(children = 'Do you usually eat vegetables in your me

                    dcc.Input(
                    id = 'selected_fcvc',
                    type = 'text',
                    list = 'liste_fcvc',
                    placeholder = 'Frequency of consumption of vegetables',
                    debounce = False,
                    spellCheck = True,
                    inputMode = 'numeric'
                    ),
                ]),

                #ncp
                html.Div(children =[
                    html.P(children = 'How many main meals do you have daily?')

                    dcc.Input(
                    id = 'selected_ncp',
                    type = 'text',
                    list = 'liste_ncp',
```

```
                placeholder = 'Number of main meals',
                debounce = False,
                spellCheck = True
                ),
            ]),

            #caec
            html.Div(children =[
                html.P(children = 'Do you eat any food between meals?'),

                dcc.Input(
                id = 'selected_caec',
                type = 'text',
                list = 'liste_caec',
                placeholder = 'Consumption of food between meals',
                debounce = False,
                spellCheck = True,
                ),
            ]),

            #smoke
            html.Div(children =[
                html.P(children = 'Do you smoke?'),

                dcc.Input(
                id = 'selected_smoke',
                type = 'text',
                list = 'liste_smoke',
                placeholder = 'Smoker',
                debounce = False,
                spellCheck = True,
                ),
            ]),

            #water
            html.Div(children =[
                html.P(children = 'How much water do you drink daily?'),

                dcc.Input(
                id = 'selected_ch2o',
                type = 'text',
                list = 'liste_ch2o',
                placeholder = 'Consumption of water daily',
                debounce = False,
                spellCheck = True,
                inputMode = 'numeric'
                ),
            ]),

            #calories monitoring
            html.Div(children =[
                html.P(children = 'Do you monitor the calories you eat dail

                dcc.Input(
                id = 'selected_scc',
                type = 'text',
                list = 'liste_scc',
                placeholder = 'Calories consumption monitoring',
                debounce = False,
                spellCheck = True,
                ),
```

```
    ]),

    #physical activities
    html.Div(children =[
        html.P(children = 'How often do you have physical activity?

        dcc.Input(
        id = 'selected_faf',
        type = 'text',
        list = 'liste_faf',
        placeholder = 'Physical activity frequency',
        debounce = False,
        spellCheck = True,
        inputMode = 'numeric'
        ),
    ]),

    #technologie usage
    html.Div(children =[
        html.P(children = 'How much time do you use technological d

        dcc.Input(
        id = 'selected_tue',
        type = 'text',
        list = 'liste_tue',
        placeholder = 'Time using technology devices',
        debounce = False,
        spellCheck = True,
        inputMode = 'numeric'
        ),
    ]),



    #alcohol
    html.Div(children =[
        html.P(children = 'how often do you drink alcohol?'),

        dcc.Input(
        id = 'selected_calc',
        type = 'text',
        list = 'liste_calc',
        placeholder = 'Consumption of alcohol',
        debounce = False,
        spellCheck = True,
        ),
    ]),

    #transport mode
    html.Div(children =[
        html.P(children = 'Which transportation do you usually use?

        dcc.Input(
        id = 'selected_mtrans',
        type = 'text',
        list = 'liste_mtrans',
        placeholder = 'Transportation used',
        debounce = False,
        spellCheck = True,
        ),
    ]),
```

```
                    #list Tool
                    html.Div(children = [
                        #Gender
                        html.Datalist(
                                id = 'liste_gender',
                                children = [
                                html.Option(value = x) for x in df['Gender'].unique
                            ),

                        #Family history
                        html.Datalist(
                                id = 'liste_family_history',
                                children = [
                                html.Option(value = x) for x in df['family_history_
                            ),

                        #FAVC
                        html.Datalist(
                                id = 'liste_favc',
                                children = [
                                html.Option(value = x) for x in df['FAVC'].unique()
                            ),

                        #CAEC
                        html.Datalist(
                                id = 'liste_caec',
                                children = [
                                html.Option(value = x) for x in df['CAEC'].unique()
                            ),

                        #SMOKE
                        html.Datalist(
                                id = 'liste_smoke',
                                children = [
                                html.Option(value = x) for x in df['SMOKE'].unique(
                            ),

                        #SCC
                        html.Datalist(
                                id = 'liste_scc',
                                children = [
                                html.Option(value = x) for x in df['SCC'].unique()]
                            ),

                        #CALC
                        html.Datalist(
                                id = 'liste_calc',
                                children = [
                                html.Option(value = x) for x in df['CALC'].unique()
                            ),

                        #MTRANS
                        html.Datalist(
                                id = 'liste_mtrans',
                                children = [
                                html.Option(value = x) for x in df['MTRANS'].unique
                            ),

                        html.Datalist(
                                id = 'liste_ncp',
```

```
                                    children = [
                                    html.Option(value = 'Between 1 and 2'),
                                    html.Option(value = '3'),
                                    html.Option(value = 'more than 3')]
                                ),

                                html.Datalist(
                                        id = 'liste_faf',
                                        children = [
                                        html.Option(value = 'I do not have'),
                                        html.Option(value = '1 or 2 days'),
                                        html.Option(value = '2 or 4 days'),
                                        html.Option(value = '4 or 5 days')]
                                ),
                                html.Datalist(
                                        id = 'liste_tue',
                                        children = [
                                        html.Option(value = '0-2 hours'),
                                        html.Option(value = '3-5 hours'),
                                        html.Option(value = 'More than 5 hours')]
                                ),

                                html.Datalist(
                                        id = 'liste_ch2o',
                                        children = [
                                        html.Option(value = 'Less than a liter'),
                                        html.Option(value = 'Between 1 and 2 L'),
                                        html.Option(value = 'More than 2 L')]
                                ),

                                html.Datalist(
                                        id = 'liste_fcvc',
                                        children = [
                                        html.Option(value = 'Never'),
                                        html.Option(value = 'Sometimes'),
                                        html.Option(value = 'Always')]
                                )

                            ]),
                        html.Button(
                            id='calcul_button',
                            n_clicks=0,
                            children='Calculate'
                        ),

                        html.Div(children =[
                            html.H3(children = 'Estimation of overweigth:'),
                            html.P(
                                id = 'prediction',
                                children = pred_value
                            )
                        ])
    ]
    )

    @app.callback(
    Output('prediction', 'children'),

    Input('calcul_button', 'n_clicks'),
    State('selected_gender', 'value'),
    State('selected_age', 'value'),
```

```
    State('selected_height', 'value'),
    State('selected_weight', 'value'),
    State('selected_fam_hist', 'value'),
    State('selected_favc', 'value'),
    State('selected_fcvc', 'value'),
    State('selected_ncp', 'value'),
    State('selected_caec', 'value'),
    State('selected_smoke', 'value'),
    State('selected_ch2o', 'value'),
    State('selected_scc', 'value'),
    State('selected_faf', 'value'),
    State('selected_tue', 'value'),
    State('selected_calc', 'value'),
    State('selected_mtrans', 'value')
    )

    def predict(n_clicks,selected_gender,selected_age,selected_height,selected_weight,selec
        pred_value = predict_value(n_clicks,selected_gender,selected_age,selected_height,se
        return pred_value

    app.run_server(debug = False)
```

```
Dash is running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/)

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production depl
oyment.
   Use a production WSGI server instead.
 * Debug mode: off


 * Running on http://127.0.0.1:8050/ (http://127.0.0.1:8050/) (Press CTRL+C
 to quit)
```

The end, thank you !