ESILV - Python for data analysis – Final Project

ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Estimation of obesity levels based on eating habits and physical condition Data Set

Thomas Hubert – IBO A5

UCI Machine Learning Repository

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

Introduction

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.

Problematic

What are the main parameter beside Weight and Height that influence the « Obesity Level » of the person ?

Summary

I. Data Exploration
  a. Exploring the dataset
  b. Verifying the dataset

II. Finding correlation between data

III. Modelisation
  a. Preparing the data
  b. Using and finding the best model
  c. Tuning the hyperparamaters

IV. Flask API

I. Data Exploration

a. Exploring the dataset

I loaded the data in a pandas dataframe for easy usage. Then I checked some basic information with .info(), .shape, .head() to see if the dataset is correctly loaded.

```
# exploration of the dataset :

ObesityData = pd.read_csv("Datasets/ObesityDataSet_raw_and_data_sinthetic.csv",sep=",")
ObesityData.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTR.
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.0	no	Public_Transporta
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.0	Sometimes	Public_Transporta
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.0	Frequently	Public_Transporta
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.0	Frequently	Wa
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.0	Sometimes	Public_Transporta

```
ObesityData.shape
```

```
(2111, 17)
```

```
ObesityData.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
```

Here are the meaning behind every columns name. Besides the obvious information of Age/Weight/Height/Nobeyesdad, we see mainly characteristic related to eating and physical habit, as well as in important one being Family_history.

```
ObesityData.columns

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

Gender : Male or Female

Age : Age of the person

Height : Height in "meter"

Weight : Weight in "kilogram"

Family_history : if parents/family with obesity

FAVC : Frequent consumption of high caloric food

FCVC : Frequency of consumption of vegetables

NCP : Number of main meals

CAEC : Consumption of food between meals

SMOKE : does the person smoke or not

CH20 : Consumption of water daily

SCC : Calories consumption monitoring

FAF : Physical activity frequency

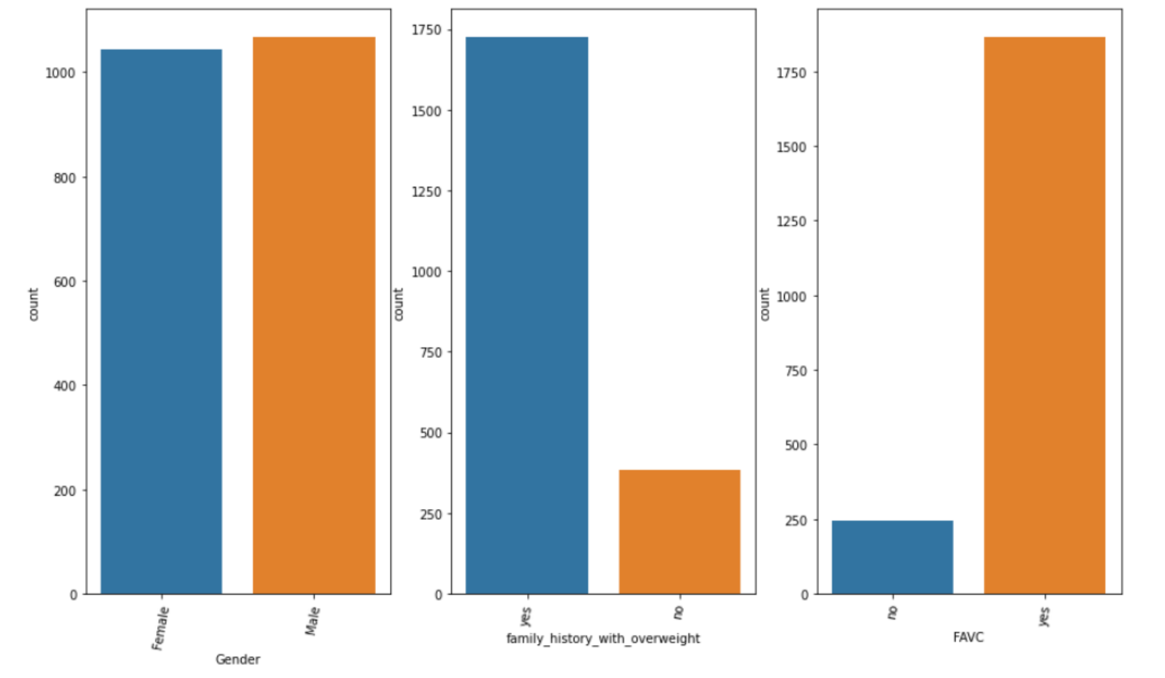
TUE : Time using technology devices

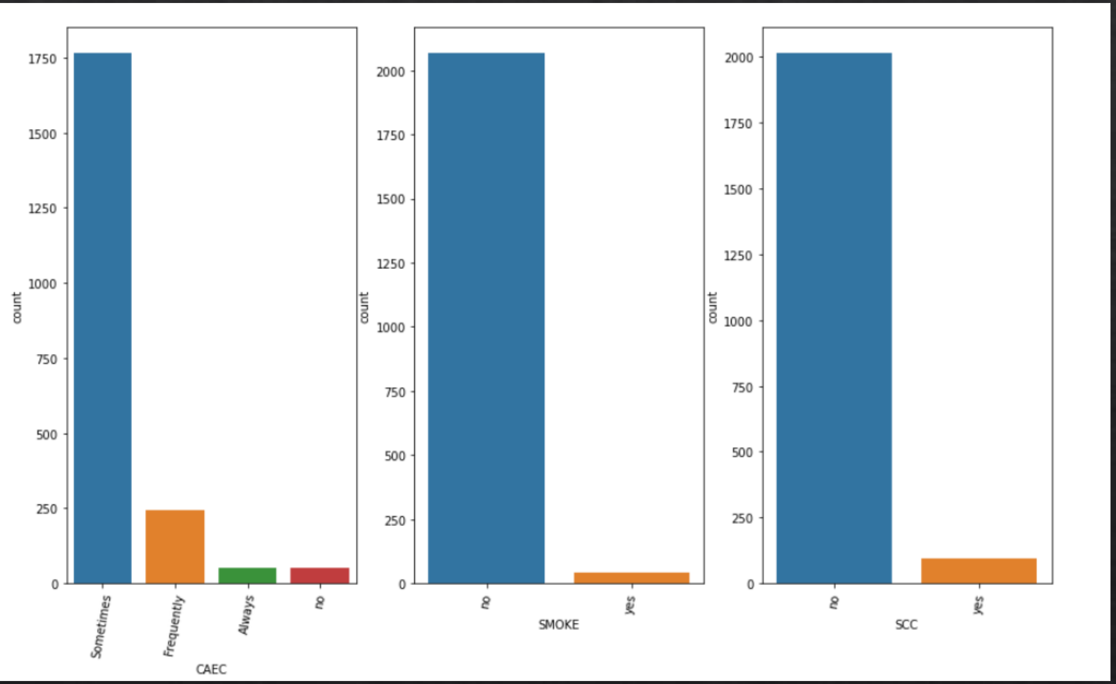
CALC : Alcool consumption

MTRANS : what kind of transportation taken

NObeyesdad : level of obesity

```python
col =['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC', 'SMOKE',
          'SCC', 'CALC', 'MTRANS', 'TUE','FAF','FCVC','NObeyesdad']

fig, ax = plt.subplots(4, 3, figsize=(15, 40))
for col, subplot in zip(col, ax.flatten()):

        seaborn.countplot(DataVizu[col], ax=subplot)
        subplot.set_xticklabels(rotation=80,labels=DataVizu[col].unique())
plt.show()
```





In order to represent TUE, FAF and FCVC, I tweaked a new dataframes based on the original to have a more discret histogram, by rounding up the values provided using a map.

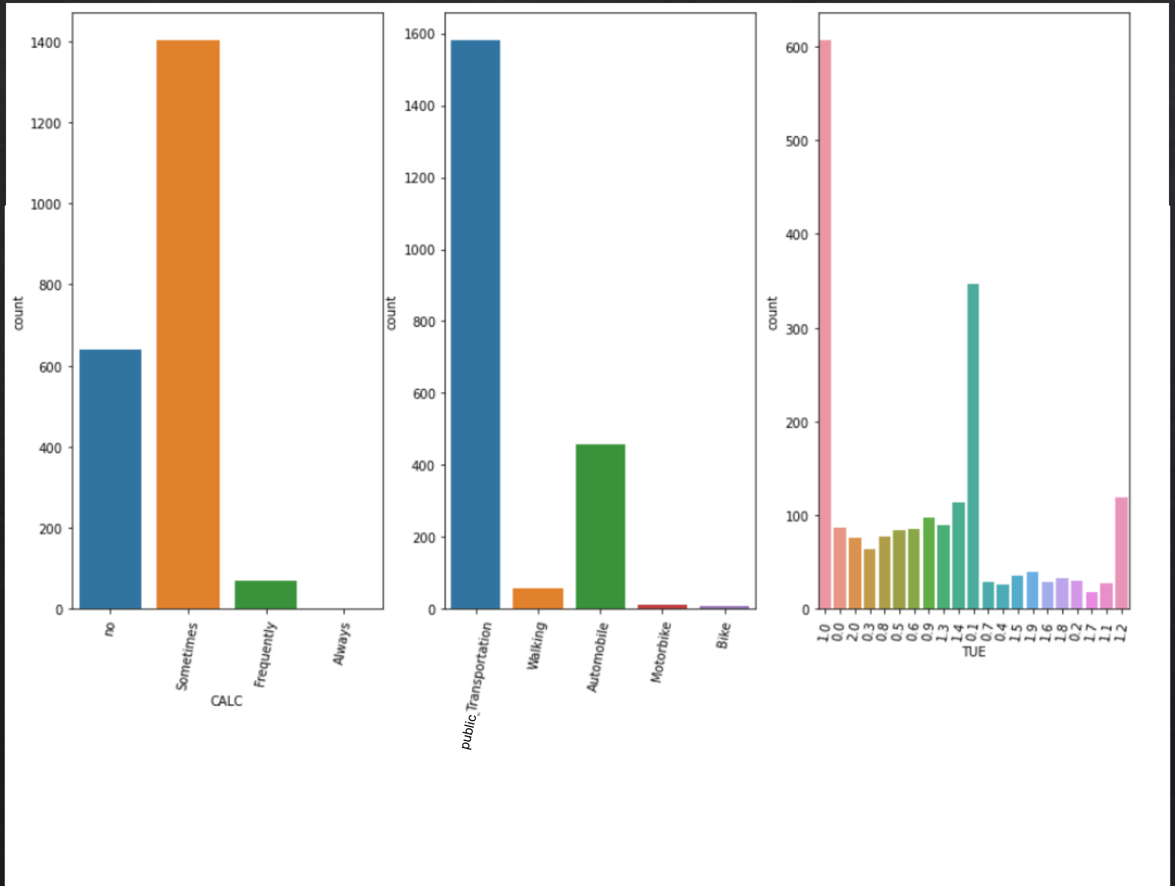
```
ObesityData.FCVC.unique()
```

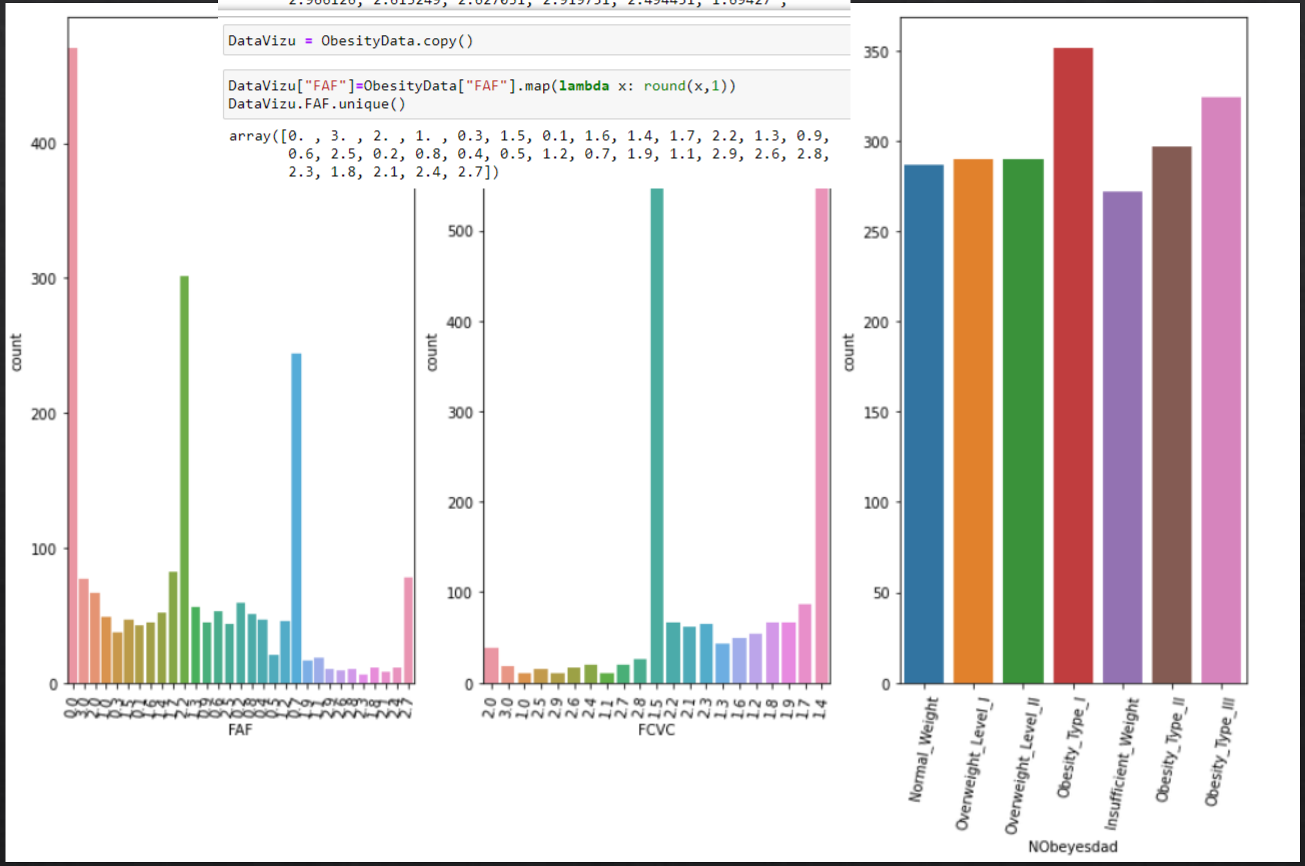
```
array([2.      , 3.      , 1.      , 2.450218, 2.880161, 2.00876 ,
       2.596579, 2.591439, 2.392665, 1.123939, 2.027574, 2.658112,
       2.88626 , 2.714447, 2.750715, 1.4925  , 2.205439, 2.059138,
       2.310423, 2.823179, 2.052932, 2.596364, 2.767731, 2.815157,
       2.737762, 2.568063, 2.524428, 2.971574, 1.0816  , 1.270448,
       1.344854, 2.959658, 2.725282, 2.844607, 2.44004 , 2.432302,
       2.592247, 2.449267, 2.929889, 2.015258, 1.031149, 1.592183,
       1.21498 , 1.522001, 2.703436, 2.362918, 2.14084 , 2.5596  ,
       2.336044, 1.813234, 2.724285, 2.71897 , 1.133844, 1.757466,
       2.979383, 2.204914, 2.927218, 2.88853 , 2.890535, 2.530066,
       2.241606, 1.003566, 2.652779, 2.897899, 2.483979, 2.945967,
       2.478891, 2.784464, 1.005578, 2.938031, 2.842102, 1.889199,
       2.943749, 2.33998 , 1.950742, 2.277436, 2.371338, 2.984425,
       2.977018, 2.663421, 2.753752, 2.318355, 2.594653, 2.886157,
       2.967853, 2.619835, 1.053534, 2.530233, 2.8813  , 2.824559,
       2.762325, 2.070964, 2.68601 , 2.794197, 2.720701, 2.880792,
       2.674431, 2.55996 , 1.212908, 1.140615, 2.562409, 2.004146,
       2.690754, 2.051283, 2.19005 , 2.21498 , 2.91548 , 2.708965,
       2.853513, 2.580872, 2.508835, 2.896562, 2.911877, 2.910733,
       2.966126, 2.613249, 2.627031, 2.919751, 2.494451, 1.69427 ,
```

```
DataVizu = ObesityData.copy()
```

```
DataVizu["FAF"]=ObesityData["FAF"].map(lambda x: round(x,1))
DataVizu.FAF.unique()
```

```
array([0. , 3. , 2. , 1. , 0.3, 1.5, 0.1, 1.6, 1.4, 1.7, 2.2, 1.3, 0.9,
       0.6, 2.5, 0.2, 0.8, 0.4, 0.5, 1.2, 0.7, 1.9, 1.1, 2.9, 2.6, 2.8,
       2.3, 1.8, 2.1, 2.4, 2.7])
```





b. Verifying the validity of the dataset

First of all, I checked to see if there were any missing or null values :

```
ObesityData.isnull().sum()

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

None were missing, the dataset is complete. Since over 75% were artificially generated from the first 22%, it is quite normal the generation was correctly filled.

Now, i wanted to see if the columns Nobeyesdad was correct by looking ip the definition from the World Health Organization, based on the BMI (Body Mass Indicator) :

(https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight)

Adults (>19 years old)

- Underweight if BMI < 18,5
- Healthy if BMI between 18,5 and 25
- Overweight if BMI > 25 and < 30
- Obesity if BMI > 30

Teenager (>5 and <= 19 years old)

- tresholds for overweight and obesity change a lot during this period, but around >25 for obesity, and between 22 and 25 for overweight for 13 years old

```
ObesityData[ObesityData.Age >= 19.0].shape
```

```
(1760, 19)
```

```
ObesityData[ObesityData.Age < 19.0].shape
```

```
(351, 19)
```

=> So we have a majority of 19 years old and more, so we can take the basis for adult to begin with with most of them

```python
# We add the BMI (IMC en français) to the model :

ObesityData['BMI'] = ObesityData['Weight']/(ObesityData['Height']**2)
ObesityData.head()
```

We add the obesity level to see if they are incoherent data later on.

```python
# We will verify the accuracy of the NObeyesdad column with the BMI

# make a copy of over 19 years old population
ObesityDataOver19 = ObesityData[ObesityData.Age >= 19.0].copy()

def verify_BMI(bmi, obe_rating):
    response = False

    all_level = ['Insufficient_Weight','Normal_Weight','Overweight_Level_I','Overweight_Level_II','Obesity_Type_I','Obesity_Type_

    if(bmi<18,5 and obe_rating == 1):
        response = True
    elif(bmi>=18.5 and bmi<25 and obe_rating == 2):
        response = True
    elif(bmi>=25 and bmi<30 and (obe_rating == 3 or obe_rating==4)):
        response = True
    elif(bmi>=25 and bmi<30 and (obe_rating == 5 or obe_rating == 6 or obe_rating == 7)):
        response = True

    return response
```

=> It appears that the data are 100% correct for the over 19 years old population

```python
compteur_true = 0
max_size = ObesityDataOver19.shape[0]
ratio_true = 0

for label, row in ObesityDataOver19.iterrows():
    if(verify_BMI(row['BMI'],row['ob_lvl'])):
        compteur_true +=1

ratio_true = (compteur_true/max_size)
print("the BMI is viable in " + str(ratio_true*100)+ "% of case")
```

```
the BMI is viable in 100.0% of case
```


ESILV

According to wikipedia, "For adults, the prevalence of overweight and obesity is 39.7 and 29.9%, respectively." (https://en.wikipedia.org/wiki/Obesity_in_Mexico#:~:text=Mexico%20passed%20the%20United%20States,39.7%20and%2029.9%25%2C%20respectively.) We are going to see if it's close to what we have in our data

```
def verify_obesity_rate(dataf):
    size_max = dataf.shape[0]
    overweight_compteur = 0
    obesity_compteur = 0
    for label, row in dataf.iterrows():
        if(row['ob_lvl']==3 or row['ob_lvl']==4):
            overweight_compteur+=1
        elif(row['ob_lvl']==5 or row['ob_lvl']==6 or row['ob_lvl']==7):
            obesity_compteur +=1
    ratio_over = overweight_compteur/size_max
    ratio_ob = obesity_compteur/size_max
    print("the ratio of overweight adult is : " + str(ratio_over*100)+ "%")
    print("the ratio of obesity for adult is : " + str(ratio_ob*100)+ "%")
```

```
verify_obesity_rate(ObesityDataOver19)
```

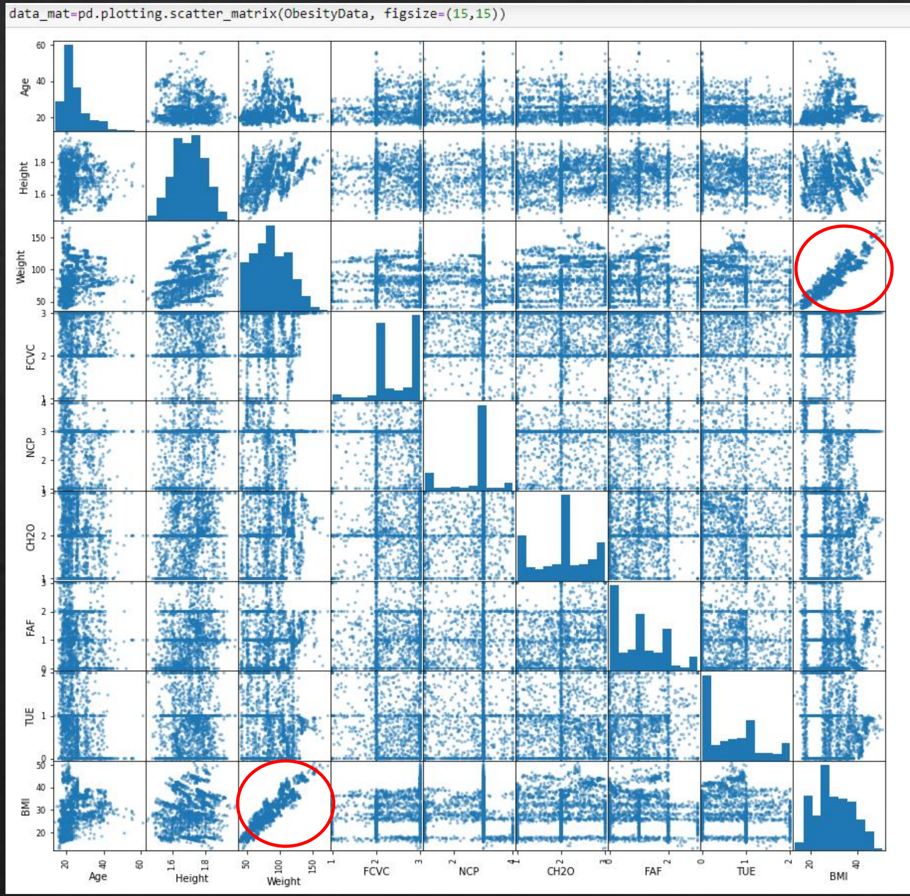
```
the ratio of overweight adult is : 28.295454545454547%
the ratio of obesity for adult is : 50.17045454545455%
```

the rate are not really close (28% for 39% and 50% for 30%), meaning it's not really representative of the whole population.

=> The ratios are not really close from reality, mostly du to a small sample given to the Weka Tool. The analysis can still be done, but the initial sample is not really representative of the population. Technically though, if the tool was used correctly, it won't really alter the answer.

II. Finding correlation between data

To begin with, we start with the scatter matrix bundled with pandas on our original dataframe :



We can easily see the almost linear relation between BMI and weight. Which is obvious, as the calculus to get the BMI involved the weight.

ESILV

I then converted non-numerical columns into « int » to use a heatmap on all the columns to see the best correlations :

```
def int_columns(df, liste_col):

    for col in liste_col:
        values = df[col].unique()
        #print(values)
        new_col_values = []
        for label, row in df.iterrows():
            temp = (np.where(values == row[col]))
            new_col_values.append(temp[0][0]+1)
        #print(new_col_values)
        new_name_col = col+" - int"
        df.insert(5,new_name_col, new_col_values)
        del df[col]
```

```
liste_col = ['Gender','family_history_with_overweight','FAVC','CAEC','SMOKE','SCC','CALC','MTRANS']
int_columns(NewData,liste_col)
```

```
NewData.head()
```

	Age	Height	Weight	Gender - int	family_history_with_overweight - int	MTRANS - int	CALC - int	SCC - int	SMOKE - int	CAEC - int	FAVC - int	FCVC	NCP	CH2O	FAF	TUE
0	21.0	1.62	64.0	1	1	1	1	1	1	1	1	2.0	3.0	2.0	0.0	1.0
1	21.0	1.52	56.0	1	1	1	2	2	2	1	1	3.0	3.0	3.0	3.0	0.0
2	23.0	1.80	77.0	2	1	1	3	1	1	1	1	2.0	3.0	2.0	2.0	1.0
3	27.0	1.80	87.0	2	2	2	3	1	1	1	1	3.0	3.0	2.0	2.0	0.0
4	22.0	1.78	89.8	2	2	1	2	1	1	1	1	2.0	1.0	2.0	0.0	0.0

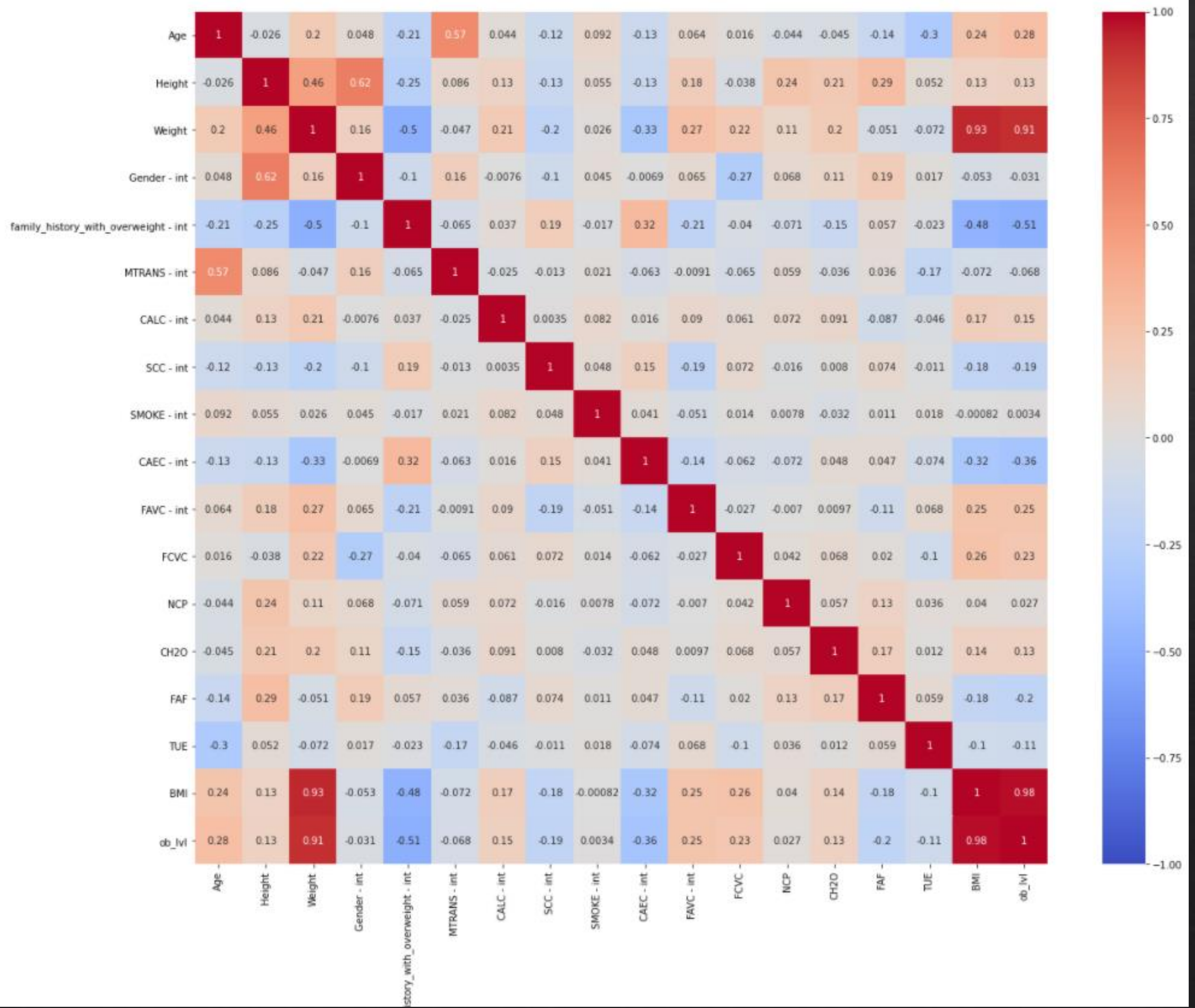
This function aims at converting « string » values into « int », then dropping the original columns. (it is done on a copied dataframe). Ob_lvl is the numerical adaptation of the Nobeyesdad columns.

Again, there is a high correlation between the obesity level and weight.

Since they pretty much means the same thing, I disregard the correlation between BMI and ob_lvl as well.

The next interesting correlation are the family history, FCVC **(Frequency of consumption of vegetables)** and FAVC **(Frequent consumption of high caloric food)**

```
plt.figure(figsize=(20,16))
seaborn.heatmap(NewData.corr("pearson"),vmin=-1, vmax=1,cmap='coolwarm',annot=True,square=True);
```



ESILV

I divided the dataframe into seperate dataframe, each one about a specific obesity level.

```
NormalW = DataVizu[DataVizu['NObeyesdad']=='Normal_Weight']
InsufW = DataVizu[DataVizu['NObeyesdad']=='Insufficient_Weight']
OverWI = DataVizu[DataVizu['NObeyesdad']=='Overweight_Level_I']
OverWII = DataVizu[DataVizu['NObeyesdad']=='Overweight_Level_II']
ObesI = DataVizu[DataVizu['NObeyesdad']=='Obesity_Type_I']
ObesII = DataVizu[DataVizu['NObeyesdad']=='Obesity_Type_II']
ObesIII = DataVizu[DataVizu['NObeyesdad']=='Obesity_Type_III']
```

```
InsufW.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTI
59	Male	20.0	1.76	55.0	yes	yes	2.0	4.0	Sometimes	no	3.0	no	2.0	2.0	no	Public_Transpo
71	Female	22.0	1.67	50.0	yes	no	3.0	3.0	no	no	3.0	yes	2.0	1.0	Sometimes	Public_Transpo
75	Female	23.0	1.63	45.0	yes	no	3.0	3.0	Sometimes	no	3.0	yes	2.0	0.0	no	Public_Transpo
76	Female	24.0	1.60	45.0	yes	no	2.0	3.0	no	no	2.0	no	1.0	0.0	no	Public_Transpo
83	Female	19.0	1.60	45.0	no	no	3.0	3.0	no	no	3.0	yes	2.0	0.0	no	W

ESILV

About FCVC

```
(InsufW['FCVC'].value_counts()/InsufW['FCVC'].count()*(100)).head(5)
```

```
3.0    33.823529
2.0    15.808824
2.9     9.191176
2.7     6.250000
2.6     5.147059
Name: FCVC, dtype: float64
```

```
(NormalW['FCVC'].value_counts()/NormalW['FCVC'].count()*(100)).head(5)
```

```
2.0    54.006969
3.0    39.721254
1.0     6.271777
Name: FCVC, dtype: float64
```

```
(OverWI['FCVC'].value_counts()/OverWI['FCVC'].count()*(100)).head(5)
```

```
2.0    42.413793
3.0    17.241379
2.3     4.482759
2.8     4.137931
2.9     4.137931
Name: FCVC, dtype: float64
```

```
(OverWII['FCVC'].value_counts()/OverWII['FCVC'].count()*(100)).head(5)
```

```
2.0    47.931034
3.0    14.482759
2.7     5.172414
2.8     4.827586
2.1     4.482759
Name: FCVC, dtype: float64
```

```
(ObesI['FCVC'].value_counts()/ObesI['FCVC'].count()*(100)).head(5)
```

```
2.0    52.136752
3.0     9.401709
2.1     5.698006
2.3     4.843305
2.9     4.273504
Name: FCVC, dtype: float64
```

```
(ObesII['FCVC'].value_counts()/ObesII['FCVC'].count()*(100)).head(5)
```

```
3.0    15.488215
2.2    10.774411
2.0     9.090909
2.9     8.754209
2.6     7.070707
Name: FCVC, dtype: float64
```

```
(ObesIII['FCVC'].value_counts()/ObesIII['FCVC'].count()*(100)).head(5)
```

```
3.0    100.0
Name: FCVC, dtype: float64
```

We observe that the frequency of vegetable consumption decrease as the BMI increase, except for the last level where the lack of data may give us incorrect result.

About FAVC

```
(InsufW['FAVC'].value_counts()/InsufW['FAVC'].count()*(100)).head(5)
```

```
yes    81.25
no     18.75
Name: FAVC, dtype: float64
```

```
(NormalW['FAVC'].value_counts()/NormalW['FAVC'].count()*(100)).head(5)
```

```
yes    72.473868
no     27.526132
Name: FAVC, dtype: float64
```

```
(OverWI['FAVC'].value_counts()/OverWI['FAVC'].count()*(100)).head(5)
```

```
yes    92.413793
no      7.586207
Name: FAVC, dtype: float64
```

```
(OverWII['FAVC'].value_counts()/OverWII['FAVC'].count()*(100)).head(5)
```

```
yes    74.482759
no     25.517241
Name: FAVC, dtype: float64
```

```
(ObesI['FAVC'].value_counts()/ObesI['FAVC'].count()*(100)).head(5)
```

```
yes    96.866097
no      3.133903
Name: FAVC, dtype: float64
```

```
(ObesII['FAVC'].value_counts()/ObesII['FAVC'].count()*(100)).head(5)
```

```
yes    97.643098
no      2.356902
Name: FAVC, dtype: float64
```

```
(ObesIII['FAVC'].value_counts()/ObesIII['FAVC'].count()*(100)).head(5)
```

```
yes    99.691358
no      0.308642
Name: FAVC, dtype: float64
```

Here, we can clearly see that the people suffering from obesity are all high caloric food eater.

About the Age

Age

```
InsufW['Age'].mean()
```

```
19.783237150735307
```

```
NormalW['Age'].mean()
```

```
21.738675958188153
```

```
OverWI['Age'].mean()
```

```
23.41767367241378
```

```
OverWII['Age'].mean()
```

```
26.996981424137942
```

```
ObesI['Age'].mean()
```

```
25.88494073219372
```

```
ObesII['Age'].mean()
```

```
28.23378532323232
```

```
ObesIII['Age'].mean()
```

```
23.4955539691358
```

=> We can see there is a tendancy of being older the bigger the obesity level is, with the insufficient weight being the younger, while the obese one are the older.

About Family History

```
(InsufW['family_history_with_overweight'].value_counts()/InsufW['family_history_with_overweight'].count()*(100)).head(5)
```

```
no     53.676471
yes    46.323529
Name: family_history_with_overweight, dtype: float64
```

```
(NormalW['family_history_with_overweight'].value_counts()/NormalW['family_history_with_overweight'].count()*(100)).head(5)
```

```
yes    54.006969
no     45.993031
Name: family_history_with_overweight, dtype: float64
```

```
(OverWI['family_history_with_overweight'].value_counts()/OverWI['family_history_with_overweight'].count()*(100)).head(5)
```

```
yes    72.068966
no     27.931034
Name: family_history_with_overweight, dtype: float64
```

```
(OverWII['family_history_with_overweight'].value_counts()/OverWII['family_history_with_overweight'].count()*(100)).head(5)
```

```
yes    93.793103
no      6.206897
Name: family_history_with_overweight, dtype: float64
```

```
(ObesI['family_history_with_overweight'].value_counts()/ObesI['family_history_with_overweight'].count()*(100)).head(5)
```

```
yes    98.005698
no      1.994302
Name: family_history_with_overweight, dtype: float64
```

```
(ObesII['family_history_with_overweight'].value_counts()/ObesII['family_history_with_overweight'].count()*(100)).head(5)
```

```
yes    99.6633
no      0.3367
Name: family_history_with_overweight, dtype: float64
```

```
(ObesIII['family_history_with_overweight'].value_counts()/ObesIII['family_history_with_overweight'].count()*(100)).head(5)
```

```
yes    100.0
Name: family_history_with_overweight, dtype: float64
```

Here, it is clear cut that family history has a direct link with the obesity level. The percentage of family with obesity increase along the obesity level of the person.

As a result, we clearly see there is a correlation between the family history, the Age, FAVC and FCVC. Regarding the context of those countries : they are not the wealthiest and junk food is most of the time cheaper and easier to acquire, making it easy for the population to reach those obesity level. A family with an history of overweight can notice this lack of wealth that will descend unto the children…

III. Modelisation

Since we are dealing with a regression problem, I will use linear regression model such as **RandomForest, KNeighborsRegressor and LinearRegression.** They are among the most used model, with linearRegression and RandomForst being supervised while Kneighbor isn't.

a. Preparing the data

First, I split the data into two : the data (NewData_X) and the label (NewData_y).
Then I split the data using sklearn library.

```
NewData_X = NewData.drop('ob_lvl', axis=1).copy()
NewData_X.head()
```

	Age	Height	Weight	Gender - int	family_history_with_overweight - int	MTRANS - int
0	21.0	1.62	64.0	1	1	1
1	21.0	1.52	56.0	1	1	1
2	23.0	1.80	77.0	2	1	1
3	27.0	1.80	87.0	2	2	2
4	22.0	1.78	89.8	2	2	1

```
NewData_y = NewData["ob_lvl"].copy()
NewData_y.head()
```

```
0    2
1    2
2    2
3    3
4    4
Name: ob_lvl, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(NewData_X, NewData_y, test_size = 0.1)
```

```
X_train.shape
```

```
(1899, 16)
```

```
y_train.shape
```

```
(1899,)
```

```
X_test.shape
```

```
(212, 16)
```

```
y_test.shape
```

```
(212,)
```

b. Finding the best model

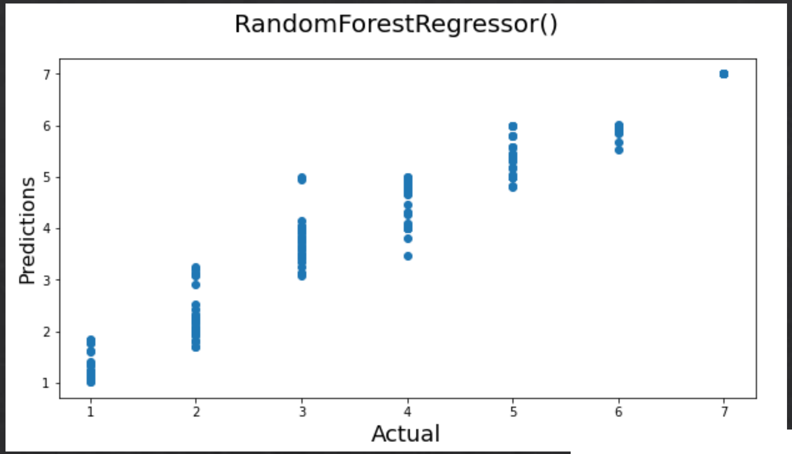
```
# I will iterate throught 3 different models : 2 supervised and 1 unsupervised

regressor_models=[RandomForestRegressor(),KNeighborsRegressor(),LinearRegression()]
all_names=['RandomForestRegressor','KNeighborsRegressor','LinearRegression']
all_rmse=[]
all_r2=[]

for i in range (len(regressor_models)):
    mod=regressor_models[i]
    mod_tuned=mod.fit(X_train,np.ravel(y_train,order='C'))
    test_pred=mod_tuned.predict(X_test)
    all_rmse.append(np.sqrt(mean_squared_error(test_pred,y_test)))
    all_r2.append(r2_score(y_test, test_pred))

    fig = plt.figure(figsize=(10,5))
    plt.scatter(y_test,test_pred)
    fig.suptitle(mod, fontsize=20)
    plt.xlabel('Actual', fontsize=18)
    plt.ylabel('Predictions', fontsize=16)

print(all_names)
print(all_r2)
print(all_rmse)
```

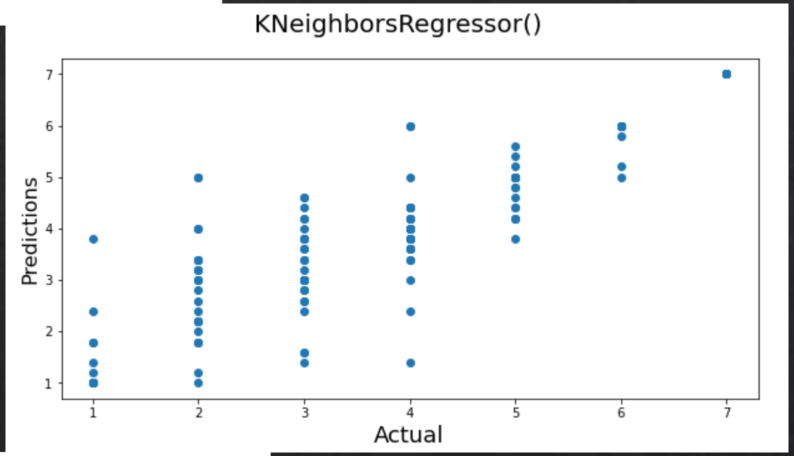
To see which model is the best, I use a for loop that will, for the same hyperparameters, plot a graph of the result to see wich one predict the better.

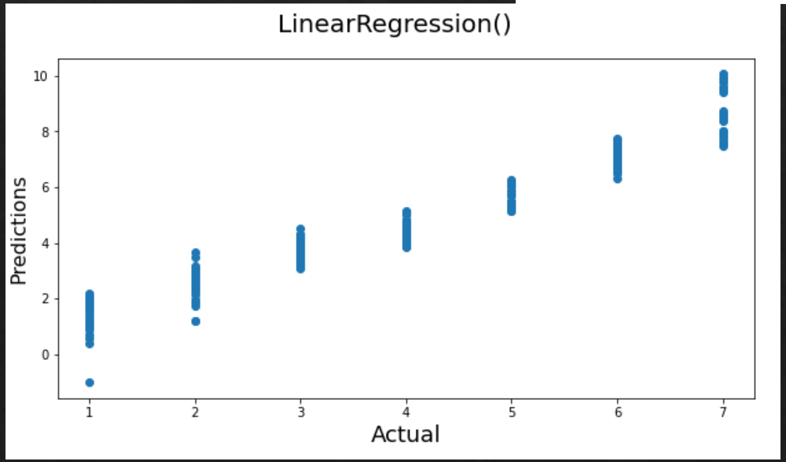
```
['RandomForestRegressor', 'KNeighborsRegressor', 'LinearRegression']
[0.953865795629884, 0.8713588173809168, 0.9109357210296658]
[0.4318403084584118, 0.7211102550927978, 0.6000168380839015]
```

According to the models, the best RMSE and R2 is the RandomForestRegressor.

This is the model i will tune from now on.







```python
from sklearn.model_selection import RandomizedSearchCV
```

```python
# We will use the random grid with the following parameters :
# estim_para (# of estimators/trees) - max_features - depth_para (max level) - min_samples_split (min # of split per leaf) -
# min_samples_leaf (min # of samples per tree) - bootstrap
# I used pretty basic and common possible parameter

estim_para = [1,5,10,15,20,25,30]
max_features = ['auto', 'sqrt']
depth_para = [1,5,10,20,30,40,50,60,70,80,90]
depth_para.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True, False]

random_grid = {'n_estimators': estim_para,
               'max_features': max_features,
               'max_depth': depth_para,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

I define potential parameters that I will store in the random_grid

I then use the RandomizedSearchCV to go through all the parameters

```python
# I used the RandomizedSearchCV to find out the best parameters

rfr_model = RandomForestRegressor()

rf_random = RandomizedSearchCV(estimator = rfr_model, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_

rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```python
# We have the best parameters here :
rf_random.best_params_
```

```
{'n_estimators': 15,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 50,
 'bootstrap': True}
```

Then I access the best parameter via the .best_params_

For my case, it would be n_estimators = 15 and max_depth = 50

ESILV

```
hyperparameter, i will create a model with them :
domForestRegressor(n_estimators=15,min_samples_split= 2, min_samples_leaf=1, max_features="auto", max_depth= 50, bootstrap= True)
```

```
# Now to fit and predict with this new model :

rfr_tuned.fit(X_train, y_train)
pred = rfr_tuned.predict(X_train)
```

```
# Let's see its R2 result :
R2_on_train = r2_score(y_train, pred)
print(R2_on_train)
```

```
0.9983591480272344
```

=> the accuracy is really good. Now, i tested it on the testing set...

```
# Usage on the test set
pred_test = rfr_tuned.predict(X_test)
R2_on_test = r2_score(y_test, pred_test)
print(R2_on_test)
```

```
0.9544891300139919
```

=> Again, the accuracy is still high, meaning there was no overfitting.

Now I just need to export the model wia pickle to use it later with the API.

When creating and using the model on the test set, I obtained a good accuracy, meaning the parameters are working.

IV. Flask API

For the API, I decided to use Flask

I exported the model with its hyperparameter using pickle to be able to load it into the separate .py used for the api.

```
import pickle
pickle.dump(rfr_tuned, open('rfr_model', 'wb'))
```

```
# I just want to see if the exported model works :
model_test = pickle.load(open('rfr_model','rb'))
predi_pickle = model_test.predict(X_test)
print(r2_score(y_test,predi_pickle))
```

```
0.9544891300139919
```

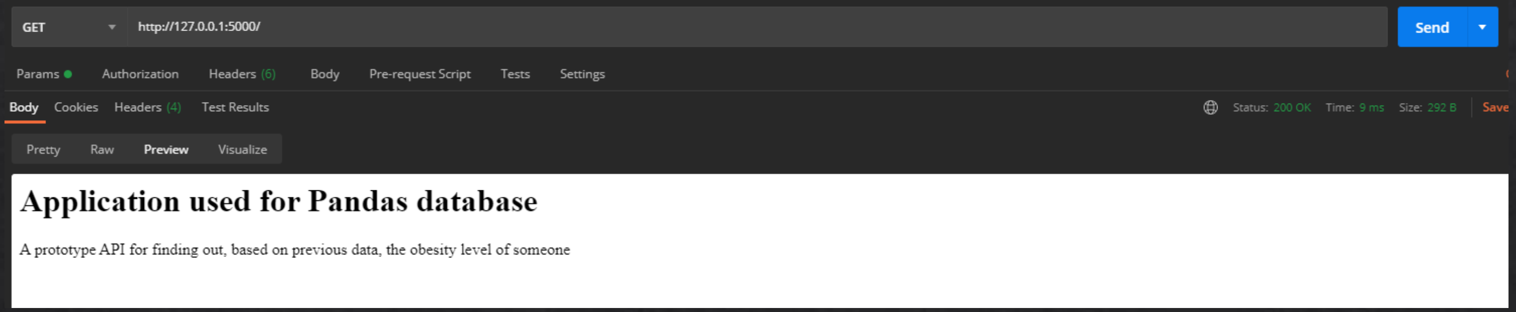
=> The model was succesfully exported and working. We can now use it for the api

ESILV

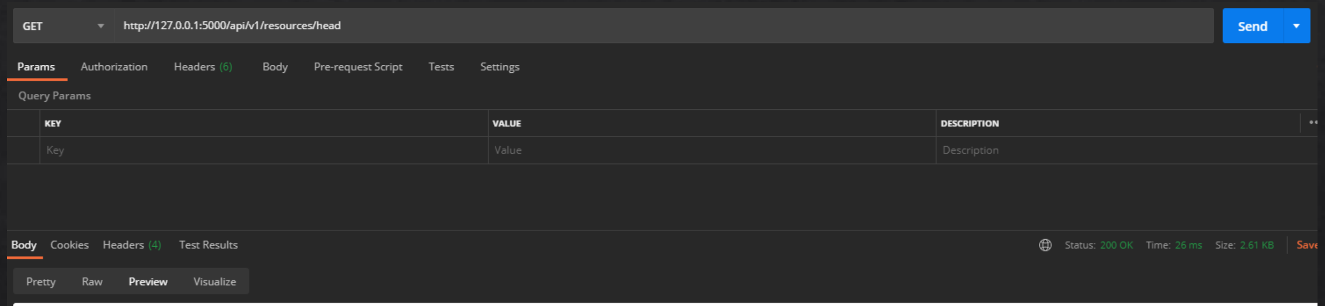
You can find the API in the app_flask.py file.

There are 3 endpoints :

- http://127.0.0.1:5000/ => basic home api



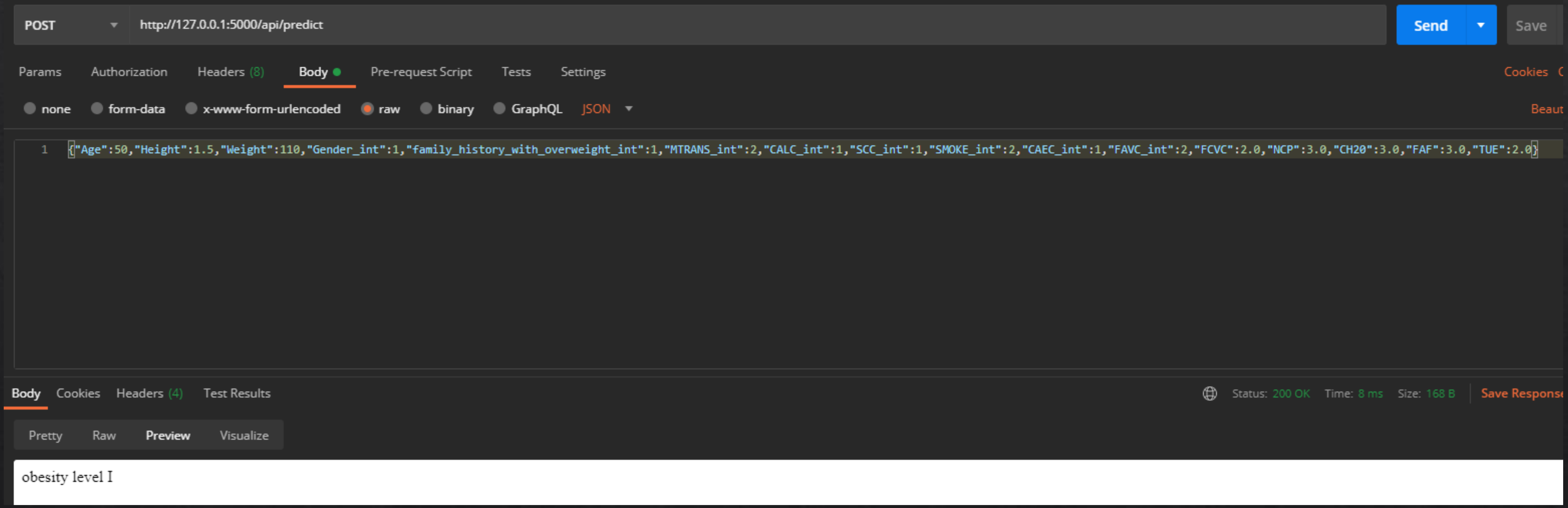
- http://127.0.0.1:5000/api/v1/ressources/head => give you the 5 first lines from the original dataset



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

http://127.0.0.1:5000/api/predict => it’s a POST endpoint where you provide a json as data that will be converted to array in order to go trhough the model and give back the result of the prediction.

In this case, we have a case of obsity level I according to our data and the model



Thank you

ESILV