Python for Data Analysis

Obesity Dataset

Yanis BENABDESSLAM

DIA 2

Summary

1. Dataset Overview
2. Visuals & Conversion
    1. First Visuals
    2. Data conversion
    3. Other plots
3. Prediction Model
    1. Used libraries
    2. Models' implementation
    3. Best model
4. Flask API
    1. For Python console view
        1. App_test.py file
        2. Request.py file
        3. Results
    2. For Explorer view
        1. App file
        2. Html page
        3. Results

Dataset Overview

- *For this project, we used the Jupyter notebook open-source web app to go through the databse, and to train our prediction models.*

- *These first rows allow us to load the database into our notebook, using python's library "pandas":*

Loading data

```
Entrée [130]: import pandas as pd

Entrée [131]: dataset = pd.read_csv("ObesityDataSet_raw_and_data_sinthetic.csv", sep = ',')

Entrée [132]: dataset
```

Out[132]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	
0	Female	21.000000	1.620000	64.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	0.000000	1.000000	
1	Female	21.000000	1.520000	56.000000	yes	no	3.0	3.0	Sometimes	yes	3.000000	yes	3.000000	0.000000	S
2	Male	23.000000	1.800000	77.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	2.000000	1.000000	F
3	Male	27.000000	1.800000	87.000000	no	no	3.0	3.0	Sometimes	no	2.000000	no	2.000000	0.000000	F
4	Male	22.000000	1.780000	89.800000	no	no	2.0	1.0	Sometimes	no	2.000000	no	0.000000	0.000000	S
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2106	Female	20.976842	1.710730	131.408528	yes	yes	3.0	3.0	Sometimes	no	1.728139	no	1.676269	0.906247	S
2107	Female	21.982942	1.748584	133.742943	yes	yes	3.0	3.0	Sometimes	no	2.005130	no	1.341390	0.599270	S
2108	Female	22.524036	1.752206	133.689352	yes	yes	3.0	3.0	Sometimes	no	2.054193	no	1.414209	0.646288	S
2109	Female	24.361936	1.739450	133.346641	yes	yes	3.0	3.0	Sometimes	no	2.852339	no	1.139107	0.586035	S
2110	Female	23.664709	1.738836	133.472641	yes	yes	3.0	3.0	Sometimes	no	2.863513	no	1.026452	0.714137	S

2111 rows × 17 columns

Dataset Overview

- *Our dataset describes around 2000 people through some features, and tells their physical state (insufficient weight, normal weight or the type of obesity in case of overweight).*
- *Here is how the "ObesityDataSet_raw_and_data_synthetic" looks like:*

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
0	Female	21.000000	1.620000	64.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	0.000000	1.000000	no	Public_Transportation	Normal_Weight
1	Female	21.000000	1.520000	56.000000	yes	no	3.0	3.0	Sometimes	yes	3.000000	yes	3.000000	0.000000	Sometimes	Public_Transportation	Normal_Weight
2	Male	23.000000	1.800000	77.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	2.000000	1.000000	Frequently	Public_Transportation	Normal_Weight
3	Male	27.000000	1.800000	87.000000	no	no	3.0	3.0	Sometimes	no	2.000000	no	2.000000	0.000000	Frequently	Walking	Overweight_Level_I
4	Male	22.000000	1.780000	89.800000	no	no	2.0	1.0	Sometimes	no	2.000000	no	0.000000	0.000000	Sometimes	Public_Transportation	Overweight_Level_II
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2106	Female	20.976842	1.710730	131.408528	yes	yes	3.0	3.0	Sometimes	no	1.728139	no	1.676269	0.906247	Sometimes	Public_Transportation	Obesity_Type_III
2107	Female	21.982942	1.748584	133.742943	yes	yes	3.0	3.0	Sometimes	no	2.005130	no	1.341390	0.599270	Sometimes	Public_Transportation	Obesity_Type_III
2108	Female	22.524036	1.752206	133.689352	yes	yes	3.0	3.0	Sometimes	no	2.054193	no	1.414209	0.646288	Sometimes	Public_Transportation	Obesity_Type_III
2109	Female	24.361936	1.739450	133.346641	yes	yes	3.0	3.0	Sometimes	no	2.852339	no	1.139107	0.586035	Sometimes	Public_Transportation	Obesity_Type_III
2110	Female	23.664709	1.738836	133.472641	yes	yes	3.0	3.0	Sometimes	no	2.863513	no	1.026452	0.714137	Sometimes	Public_Transportation	Obesity_Type_III

2111 rows × 17 columns

Dataset Overview

- *The following table describes each feature used in this dataset:*

- *Once we know what each feature means, we can try to analyze the dataset by drawing some graphs, that allow us to see the relationships between them.*

Feature	Description
Gender	Represents if the person is a man or a woman
Age	Defines the age of the person
Height	The height of the person
Weight	The weight of the person
Family_history_ with_overweight	Shows if a family member suffers from overweight
FAVC	Shows if the person frequently eats high caloric food
FCVC	Shows if the person usually eats vegetables in his/her meals
NCP	The number of main meals that the person daily has
CAEC	The frequency of feeding between meals
SMOKE	Shows if the person is used to smoke or not
CH2O	The quantity of drunk water daily
SCC	Tells if the person monitors the calories he/she eats
FAF	Shows how often the person has a physical activity
TUE	Daily time spent using technological devices
CALC	The frequency of drinking alcohol
MTRANS	Used transportation
NObeyesdad	Shows if the person is obese, normal, or skinny

Visuals & Conversion

1. First Visuals

- *Average weight per gender:*

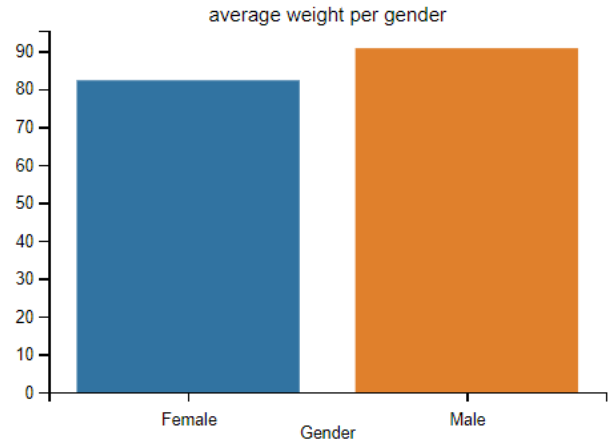
- *First, I created a “data_gen_wei” variable that contains the two genders and the average weight of each one.*

```
Entrée [20]: data_gen_wei = dataset.groupby("Gender").Weight.mean().sort_values()
             data_gen_wei

Out[20]: Gender
         Female    82.302364
         Male      90.769478
         Name: Weight, dtype: float64
```

- *Then, I used the “seaborn” library to make a bar chart using the data contained in the “data_gen_wei” variable.*

- *We can notice that, in this dataset, men are “heavier” than women*

```
Entrée [32]: sns.barplot(x = data_gen_wei.index, y = data_gen_wei.values).set_title("average weight per gender")

Out[32]: Text(0.5, 1.0, 'average weight per gender')
```



Visuals & Conversion

1. First Visuals

- *Average weight per obesity type*

- *First, I created a "data_tar_wei" variable that contains all the obesity types and the average weight of people that are in each category.*

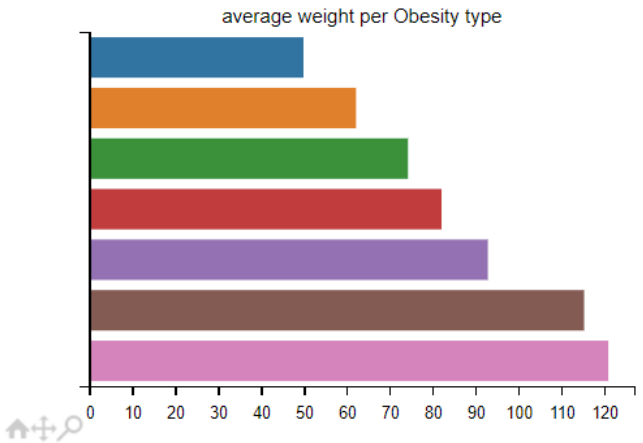
```
Entrée [22]: data_tar_wei = dataset.groupby("NObeyesdad").Weight.mean().sort_values()
             data_tar_wei
```

```
Out[22]: NObeyesdad
         Insufficient_Weight     49.906330
         Normal_Weight           62.155052
         Overweight_Level_I      74.266828
         Overweight_Level_II     82.085271
         Obesity_Type_I          92.870198
         Obesity_Type_II        115.305311
         Obesity_Type_III       120.941114
         Name: Weight, dtype: float64
```

```
Entrée [34]: sns.barplot(y = data_tar_wei.index, x = data_tar_wei.values).set_title("average weight per Obesity type")
Out[34]: Text(0.5, 1.0, 'average weight per Obesity type')
```

- *Like the previous visual, we used the "seaborn" library to plot the bar chart associated to the "data_tar_wei" variable.*

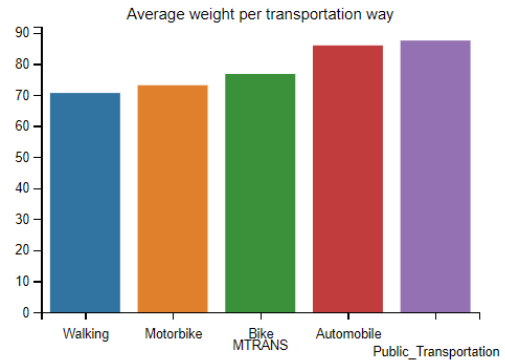
- *We can notice that the fatter you are, the higher your type of obesity is. In fact, this is an obvious result, but I've decided to show it to be sure that the dataset is logical on this point.*



Visuals & Conversion

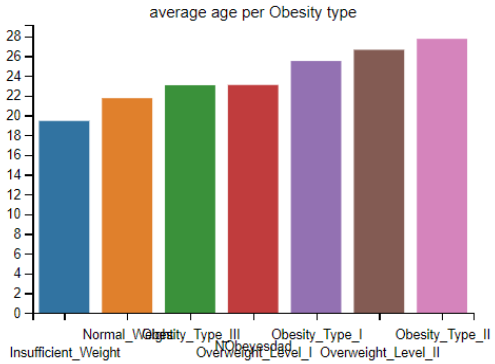
1. First Visuals

- *Using the same method as before, I've plotted the following graphs:*



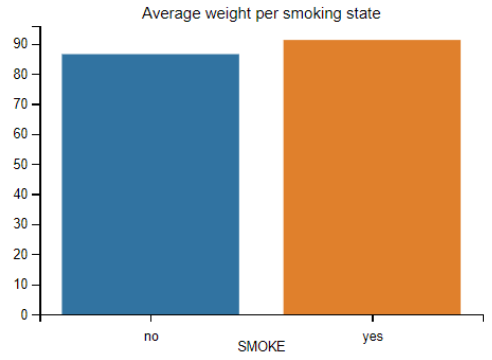
Observation:

- *The fattest people use to travel using public transportation ways. However, the skinniest use to walk.*



Observation:

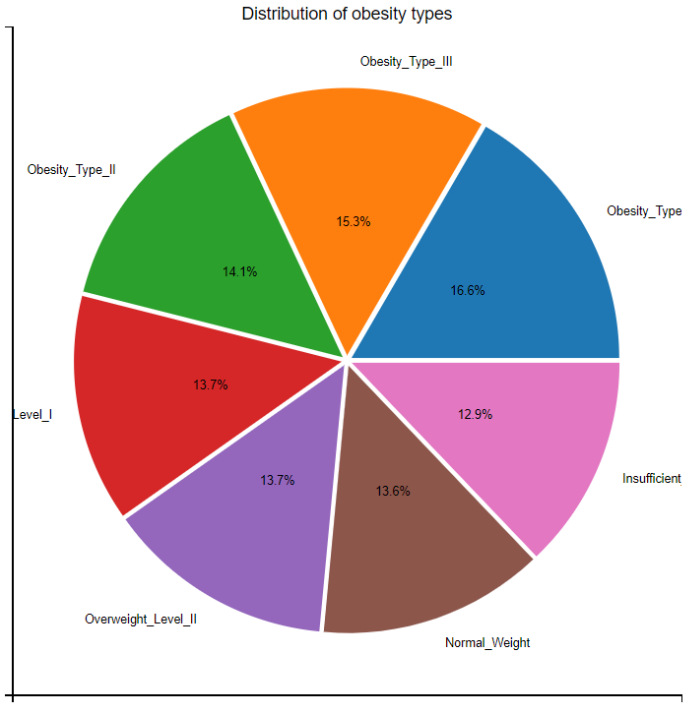
- *The obesity type worsens with age: the older you are, the more likely you are to be obese*



Observation:

- *According to our database, the more you smoke, the more you gain weight.*

- *For the pie chart, I used the matplotlib library*



Visuals & Conversion

2. Data Conversion

- *As most of the dataset values that are in “string” type are few for each string column (ex: CAEC values are "no", "sometimes", always ", frequently"), we decided to convert these values types into “int” type, so it will be easier to use for our future prediction models.*

```
Entrée [104]: dataset.CAEC.value_counts()

Out[104]: Sometimes     1765
          Frequently     242
          Always          53
          no              51
          Name: CAEC, dtype: int64
```

- *For example, concerning the CAEC column, we applied the following:*

```
Entrée [105]: dataset.CAEC = dataset.CAEC.map({'no' : 0, 'Sometimes' : 1, 'Frequently' : 2, 'Always' : 3})
              dataset.head()
```

Visuals & Conversion

2. Data Conversion

- *After applying the previous line to all the “non-int” (or float) values, we obtained the following dataset :*

Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
2	21	1.62	64.0	1	0	2.0	3.0	1	0	2.0	0	0.0	1.0	0	0	Normal_Weight
2	21	1.52	56.0	1	0	3.0	3.0	1	1	3.0	1	3.0	0.0	1	0	Normal_Weight
1	23	1.80	77.0	1	0	2.0	3.0	1	0	2.0	0	2.0	1.0	2	0	Normal_Weight
1	27	1.80	87.0	0	0	3.0	3.0	1	0	2.0	0	2.0	0.0	2	2	Overweight_Level_I
1	22	1.78	89.8	0	0	2.0	1.0	1	0	2.0	0	0.0	0.0	1	0	Overweight_Level_II

- *Then, we could plot other graphs, that allowed us to see the impact of each feature on our taget.*

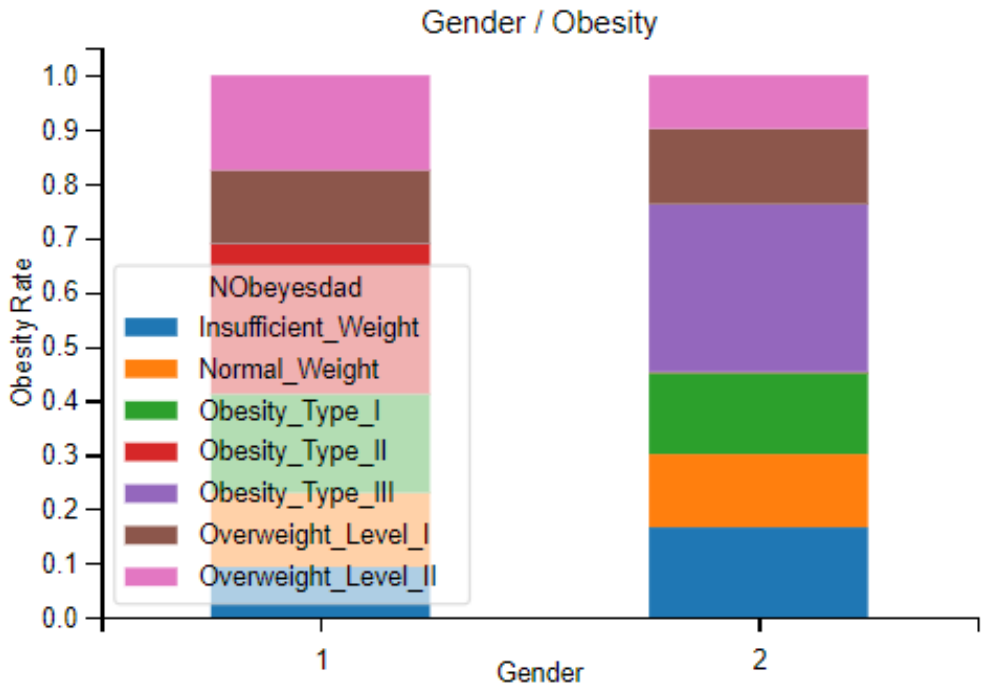
Visuals & Conversion

3. Other plots

- *The next plots that I have done allows us to see if each feature has an impact or not on our target value. If there are some features that have no impact on the target, we can get rid of them so we can simplify our prediction models (it will need less features values to work correctly).*

- *To see these impacts, I have chosen to plot (here, for the gender column) in different colors the obesity types for each gender, on a scale going from 0 to 1. Then, I can visually compare the bar plots for each column to see if there is a quite clear difference or not, according to the column value, so that we can remove or not the feature for our prediction model.*

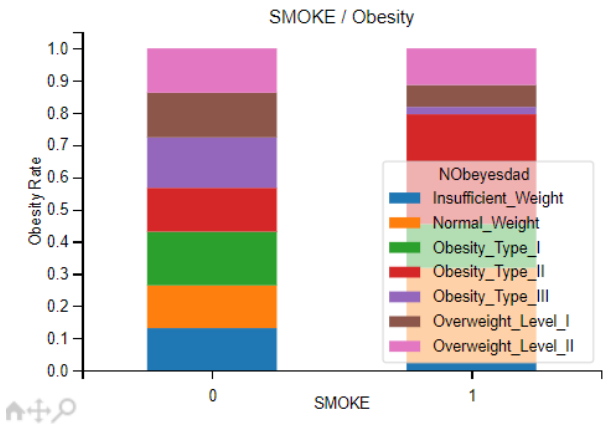
```
Entrée [114]: table = pd.crosstab(dataset.Gender,dataset.NObeyesdad)
              table.div(table.sum(1).astype(float),axis=0).plot(kind='bar', stacked=True)
              plt.title("Gender / Obesity")
              plt.xlabel("Gender")
              plt.ylabel("Obesity Rate")

Out[114]: Text(0, 0.5, 'Obesity Rate')
```

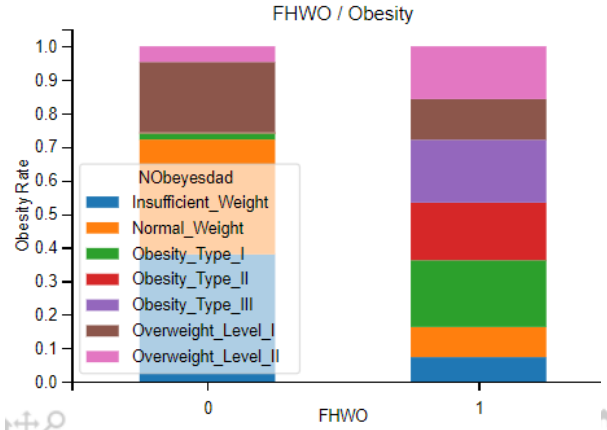


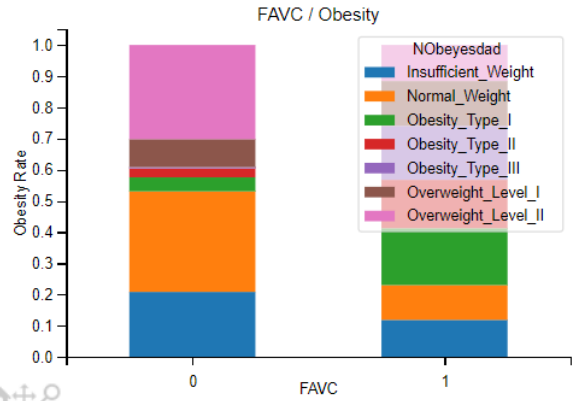
Visuals & Conversion

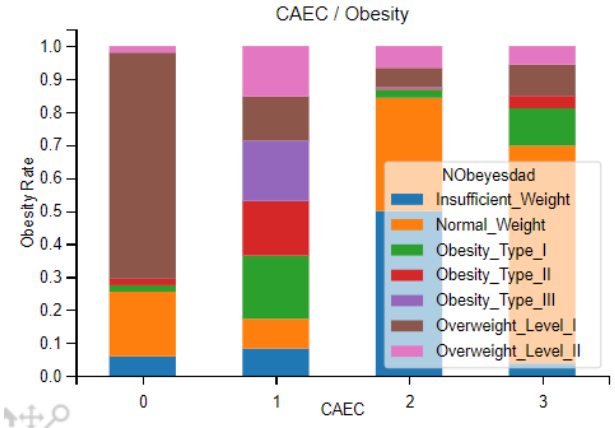
3. Other plots

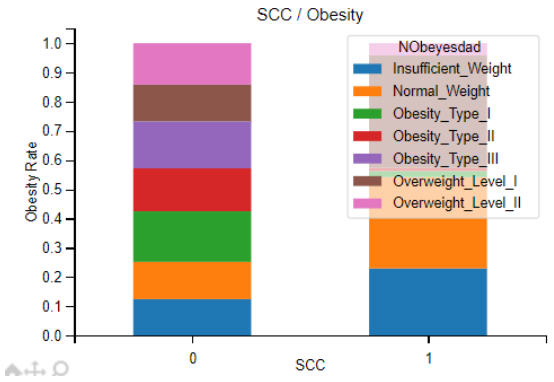
- *By proceeding the same way for each feature, we obtain the following plots:*

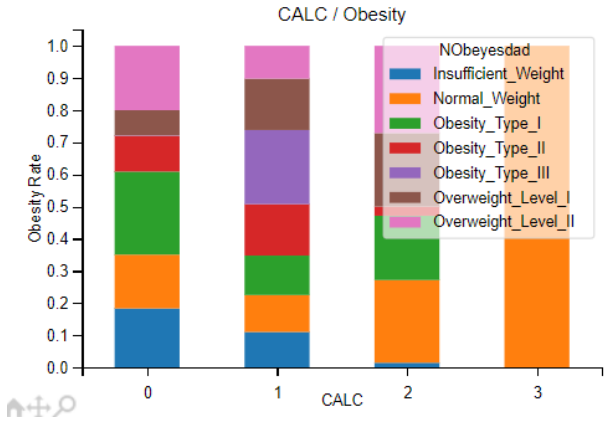


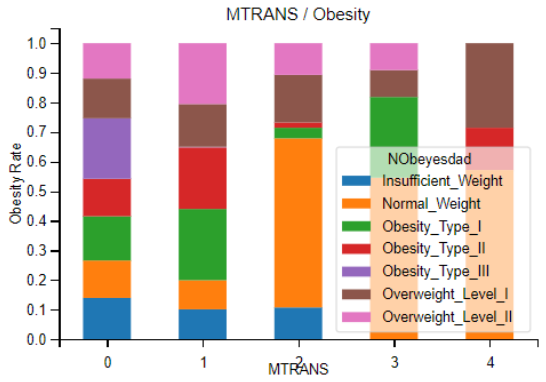












Visuals & Conversion

3. Other plots

- *I didn't make this plot on all the features : there are some features that have a lot of different values (Height, weight, etc.) : then some of these plots are not readable.*

ex :

- *When we look at the previous graphs, we can see that each feature has quite a clear impact on the target, as the repartition on the [0,1] scale according to the feature value, is not the same.*

- *So, for our prediction models, I have decided to keep all the 16 features.*

Prediction model

1. Used libraries

- *Recall : I used the following converted dataset to train the prediction models:*

Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	NObeyesdad
2	21	1.62	64.0	1	0	2.0	3.0	1	0	2.0	0	0.0	1.0	0	0	Normal_Weight
2	21	1.52	56.0	1	0	3.0	3.0	1	1	3.0	1	3.0	0.0	1	0	Normal_Weight
1	23	1.80	77.0	1	0	2.0	3.0	1	0	2.0	0	2.0	1.0	2	0	Normal_Weight
1	27	1.80	87.0	0	0	3.0	3.0	1	0	2.0	0	2.0	0.0	2	2	Overweight_Level_I
1	22	1.78	89.8	0	0	2.0	1.0	1	0	2.0	0	0.0	0.0	1	0	Overweight_Level_II

- *To make our predictions, I have mainly used the **"Scikit learn"** library : it contains a lot of "machine learning" models, and a train/test splitter.*

- *The models that I used are the following ones:*
  - *Logistic Regression*
  - *Random Forest*
  - *SVM*
  - *Bagging*
  - *Ada Boosting*
  - *Stochastic gradient Boosting*
  - *ExtraTrees*

Entrée [86]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
```

Prediction model

2. Models' implementation

- *First, I have splitted the dataset into 2 arrays X & y :*
  - *X contains all of the 16 features values,*
  - *Y contains the target values.*

```
Entrée [94]: X = data.iloc[:, :-1].values
             y = data.iloc[:, -1].values
```

- *Then, I used the "train_test_split" from scikit learn to split the dataset into training & testing sets :*

```
Entrée [95]: # Splitting data into train and test sets

             X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

- *I decided to split the dataset so that we have 70% of the data representing our training set, and 30% representing our testing set.*

Prediction model

2. Models' implementation

- *First, I have chosen to try to make a **logistic regression** on our data :*
  - *I imported the needed function from scikit learn as shown here :*

```
Entrée [96]: from sklearn.linear_model import LogisticRegression
```

  - *Then, I defined the **"classifier"** variable as the logistic regression model, setting a **"random_state"** parameter so the result doesn't change each time we rerun the cell.*
  - *After this, I fitted this classifier to our **"train set"** and then used the score function to see the accuracy of our prediction model.*

```
Entrée [97]: # Log Reg model
             classifier = LogisticRegression(solver = 'newton-cg', random_state = 5)
             classifier.fit(X_train, y_train)
             y_pred = classifier.predict(X_test)
             classifier.score(X_test,y_test)

Out[97]: 0.8296529968454258
```

- *As shown in this screenshot, the **logistic regression** model returned us around **83% of accuracy**, when testing it on the **"test set"***

Prediction model

3. Best model

- *Afterwards, I repeated **the same process** for all the previously mentioned models.*

- *Once we had all our classifiers, we could start **to compare** them, to see which one is the **most accurate** in our case :*
  - *First, I created **2 lists** that would be used to draw a table, containing in the 1st column the models' names, and the models' scores in the 2nd one :*

Entrée [110]:
```
all_models = [classifier,
              classifier2,
              classifier3,
              classifier4,
              classifier5,
              classifier6,
              classifier7]

models_names = ["Logistic Regression",
                "Random Forest",
                "SVM",
                "Bagging",
                "AdaBoosting",
                "Stochastic Gradient Boosting",
                "Extra Trees"]
```

  - *Then, I created a **dictionary** with 2 keys : **"Model"** and **"Score"**, as explained in the previous point. I filled this dictionary with all the models' names, and the score of each one. After this, I converted this dictionary into a **"pandas"** data frame, so that it is easier to show & read.*

Entrée [111]:
```
dico = {"Model" : [],
        "Score" : []}

for i in range(len(all_models)):
    dico["Model"] += [models_names[i]]
    dico["Score"] += [all_models[i].score(X_test,y_test)]

pd.DataFrame.from_dict(dico)
```

Prediction model

3. Best model

- *After finishing all the previous steps, we obtained the following table:*

	Model	Score
0	Logistic Regression	0.829653
1	Random Forest	0.941640
2	SVM	0.563091
3	Bagging	0.917981
4	AdaBoosting	0.258675
5	Stochastic Gradient Boosting	0.968454
6	Extra Trees	0.925868

- *As we can see, the best model in our case is the **"Stochastic Gradient Boosting"** classifier, with an accuracy of **96,85%**.*

- *Now we know which model is the best, we can use the **"pickle"** library to export it, so we can use it later for our **Flask API** :*

Entrée [83]:
```
# export pickle files

import pickle

pickle.dump(classifier6, open('pickle files/my_preds.pickle','wb'))
```

Flask API

1. For Python console view

1. App_test.py file

- *Now we have our pickle file containing the model, we can create a new python file that I called **"app_test.py"** which will contain all we need **to predict the obesity type** according to the features in input. We used the **"Spyder"** environment.*

- *First, we imported the following libraries (all of them are not specially needed, but I did this just in case):*

```
from flask import Flask, request, redirect, url_for, flash, jsonify, render_template
import numpy as np
import pickle as p
```

- *Now, we load the pickle file of the model by selecting its file location, and the **pickle.load()** function, and then we use the **Flask()** function to define the app :*

```
file_model = 'C:/Users/yYaNn/Documents/A4/Data Analysis Python/ObesityDataSet_raw_and_data_sinthetic (2)/pickle files/my_preds.pickl
model = p.load(open(file_model, 'rb'))
app = Flask(__name__)
```

Flask API

1. For Python console view

1. App_test.py file

```python
@app.route('/api/', methods=['POST'])
def obes_predict():

    v_data = request.get_json()
    prediction = model.predict(v_data)
    res = prediction[0]
    return jsonify(res)

if __name__ == '__main__':
    app.run(debug=True)
```

- *Then, I created the "obes_predict" function which uses the "model" to predict the "v_data" that we put in the input of the "request.py" file that we created later.*
- *Afterwards, I set the result of the prediction in the "res" variable that I returned using the "jsonify()" function.*
- *At the end, I ran the app by using the "run" function from the Flask library.*

- *The next step was to run our app: for this, I used the **Anaconda Prompt** software to host the app. First, we have to set our **app_test.py** file location using the **"cd"** prefix, and then run the python file with the **"python"** prefix. In our case, this hosts the app on **http://127.0.0.1:5000/***

```
(base) C:\Users\yYaNn>cd C:\Users\yYaNn\Documents\A4\Data Analysis Python\ObesityDataSet_raw_and_data_sinthetic (2)\pickle files

(base) C:\Users\yYaNn\Documents\A4\Data Analysis Python\ObesityDataSet_raw_and_data_sinthetic (2)\pickle files>python app_test.py
 * Serving Flask app "app_test" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 232-512-683
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Flask API

1. For Python console view

2. request.py file

- *Now, we can move to the request.py file:*
  - *First, I imported the 2 following libraries:*

```
import requests
import json
```

  - *Afterwards, I defined the data that we want to use to test our model on. We can see here that I entered the data in a 2D array type: each number corresponds to the value of a feature, according to the "converted dataset" that we made from the raw one. To simplify the selection of the values, I decided to add a description next to each value, so that we can know which value to enter.*

```
data = [[1, #   - Gender: Male = 1, Female = 2
         20, #   - Age
         1.91, #   - Height
         60.0, #   - Weight
         1, #   - family_history_with_overweight: yes = 1, no = 0
         1, #   - FAVC: yes = 1, no = 0
         0.0, #   - FCVC - consumption of vegetables
         2.0, #   - NCP - Number of main meals
         3, #   - CAEC: no = 0, Sometimes = 1, Frequently = 2, Always = 3
         0, #   - SMOKE: yes = 1, no = 0
         2.0, #   - CH2O - consumption of water
         0, #   - SCC: yes = 1, no = 0
         0.0, #   - FAF - Physical activity frequency
         2.0, #   - TUE - Time using technology devices
         0, #   - CALC: no = 0, Sometimes = 1, Frequently = 2, Always = 3
         0]] #   - MTRANS: Public_Transportation = 0, Automobile = 1, Walki
```

- *Then, I changed the format of the data using the json.dumps() function, so it can be used in the request demand. Concerning the requests, we used the requests.post() function, which contain the url of the host that we have previously seen in the Anaconda Prompt software, the data that we want to predict the associated obesity type from, and the headers which will say that our data has been loaded in json type.*

- *In this case, we tried our model on 2 people: one who is clearly under normal weight, and another one who is slightly overweight.*

```
headers = {'content-type': 'application/json', 'Accept-Charset': 'UTF-8'}
j_data   = json.dumps(data)
j_data_2 = json.dumps(data_2)

r   = requests.post('http://127.0.0.1:5000//api/', data = j_data, headers = headers)
r_2 = requests.post('http://127.0.0.1:5000//api/', data = j_data_2, headers = headers)

print()
print("Physical state: ", r.text)
print()
print("Physical state 2: ", r_2.text)
```

Flask API

1. For Python console view

3. Results

- *So, I have tested the model with these 2 data, and we can see that the first one has been detected as “insufficient weight”, and the second one as “Overweight level II”, which seems to be logical, regarding the features of each person*

```
data = [[1, #   -
         20, #   -
         1.91, #
         60.0, #
         1, #    -
         1, #    -
         0.0, #
         2.0, #
         3, #    -
         0, #    -
         2.0, #
         0, #    -
         0.0, #
         2.0, #
         0, #    -
         0]] #   -
```

```
data_2 = [[1, #
           25, #
           1.74, #
           90.0, #
           1, #
           1, #
           0.0, #
           2.0, #
           3, #
           1, #
           2.0, #
           0, #
           0.0, #
           2.0, #
           0, #
           0]] #
```

- *Result :*

```
Physical state:  "Insufficient_Weight"

Physical state 2:  "Overweight_Level_II"
```

Flask API

2. For Explorer view

1. App.py file

- *The problem with our previous API is that it is quite hard to know which value we want to be in input, as the values must be entered in an "int" or "float" type. So, I have decided to make another API which would be clearer.*

- *The app.py file is almost the same as the previous "app_test.py" : the main difference is that I added the following lines so that the program renders the API using the html file "index.html" which contains the page that will be shown in the explorer.*

```python
@app.route('/')
def home():
    return render_template('index.html')
```

- *The next step (as the previous API) was to run our app: for this, I used the **Anaconda Prompt** software to host the app. First, we need to set our **app.py** file location using the **"cd"** prefix, and then run the python file with the **"python"** prefix. In our case, this hosts the app on **http://127.0.0.1:5000/***

```python
if __name__ == "__main__":
    app.run(debug=True, host='127.0.0.1')
```

```
(base) C:\Users\yYaNn>cd C:\Users\yYaNn\Documents\A4\Data Analysis Python\ObesityDataSet_raw_and_data_sinthetic (2)\pickle fi
les

(base) C:\Users\yYaNn\Documents\A4\Data Analysis Python\ObesityDataSet_raw_and_data_sinthetic (2)\pickle files>python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with windowsapi reloader
 * Debugger is active!
 * Debugger PIN: 232-512-683
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Flask API

2. For Explorer view

2. Html page

- *Let's see in more detail how the html page looks like:*
  - *First, I decided to put some boxes where we can enter our values, for each feature. Here are the first html input rows :*

```
<input type="text" name="Gender" placeholder="Gender" required="required" />   
<input type="text" name="Age" placeholder="Age" required="required" />   
<input type="text" name="Height" placeholder="Height" required="required" />   
<input type="text" name="Weight" placeholder="Weight" required="required" />   
```

  - *Then, I had to make a "predict" button, that will return the obesity type of the person that corresponds to the data we gave in input of the model :*

```
<button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
```

  - *Finally, to help the user know what to enter in each zone, I added under these boxes a category named "How to fill the inputs", where I made an html table describing the features, and the values that we need to enter for each one :*

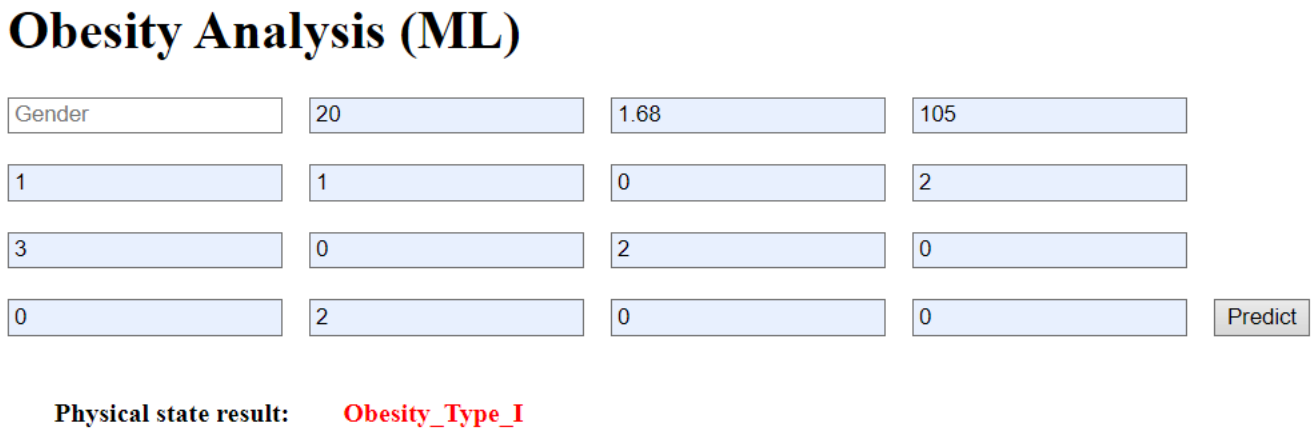
```
<tr>
        <td> <br> - FCVC </td>
        <td> <br> Do you usually eat vegetables in your meals? </td>
        <td> <br>
             <b> 1 </b> : Never
             <br>
             <b> 2 </b> : Sometimes
             <br>
             <b> 3 </b> : Always
        </td>
</tr>
```

Flask API

2. For Explorer view

3. Results

- *This is how the final page looks like :*



- *As we can see, all the needed information are below the prediction zone : each input is explained in the "description" column, and the values that we need to enter in each zone are shown in the "input values" column.*

- *Here, I filled the features for someone who is 20 years old, 1.68m tall and 105Kg weight. We can notice that the model returned us that this person was in the case of a **Type I obesity**.*

How to fill the inputs:

Input name	Description	Input values
- Gender	Gender	**1** : Man **2** : Woman
- Age	Age in years	Your age (in **years**)
- Height	Height in meters	Your height (in **meters**)
- Weight	Weight in Kg	Your weight (in **Kg**)
- FHWO	Has a family member suffers from overweight ?	**0** : no **1** : yes
- FAVC	Do you eat high caloric food frequently ?	**0** : no **1** : yes

Thank you !