Python for data analysis Report

Célina Ben Mansour

Thomas Ariza

I. Introduction

The objective of this work is to predict a target variable from a given dataset. The dataset gives an estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico. It was created in August 2019, with 77% of data generated synthetically (with the Weka tool and SMOTE filter) and 23% collected on a web platform from users.

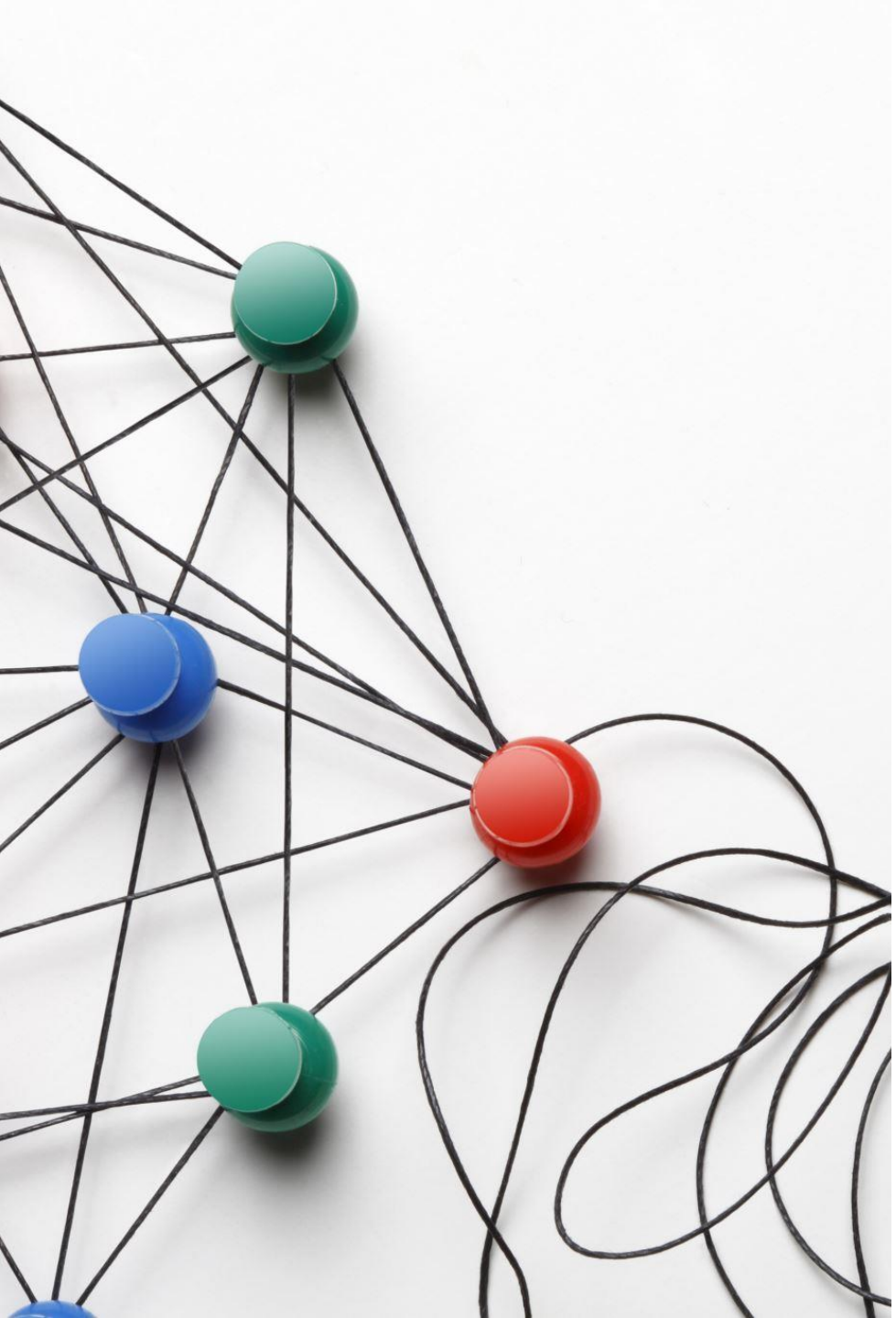
We aim to find a model based on the data, which will be used to make predictions on new data to know the category a person belongs to.

The dataset contains 17 attributes and 2111 records.

Attributes

- Gender : Male/Female
- Age
- Height
- Weight
- Family history with overweight : yes/no
- Eating high caloric food frequently (FAVC) : yes/no
- Frequence of eating vegetables in meals (FCVC) : neer/ sometimes/ always
- Number of main meals per day (NCP) : Between 1 and 2 / 3 / more than 3
- Frequence of eating food between meals (CAEC) : no/ sometimes/ frequently/ always
- Smoke : yes/no
- Quantity of water drunk per day (CH2O) : less than 1L / between 1 and 2L / more than 2L
- Monitoring the daily calories (SCC) : yes/no
- Frequence of physical activity (FAF) : I do not have / 1 or 2 days / 2 or 4 days / 4 or 5 days
- Use of technological devices (TUE) : 0-2h / 2-5h / more than 5h
- Drink alcohol (CALC) : I do not / sometimes / frequently / always
- Transportation (MTRANS) : automobile / motorbike / bike / public transportation / walking

Target variable : Obesity level (Nobeyesdad) : Insufficient Weight, Normal Weight, Overweight Level I, Overweight Level II, Obesity Type I, Obesity Type II and Obesity Type III



Since we are dealing with a classification problem (with seven different classes), it is necessary for our model to use a classification algorithm.

Moreover, we use here a dataset labeled with the response given for the target variable.

In these conditions, we had to choose a model of classification for supervised learning.

This operation requires the use of defferent python libraries.

Libraries

- Pandas : for the crating and processing the dataset
- Numpy : to process the data
- Seaborn : for data-visualization
- Matplotlib : for data-visualization
- Pylab : for data-visualization
- Scikit Learn : for the implementation of machine learning models

II. Data-visualization

To understand the given dataset, it is necessary to check some statistics about the different variables.

The first step consists of visualizing the set's shape and the number of values in each category of the target variable.

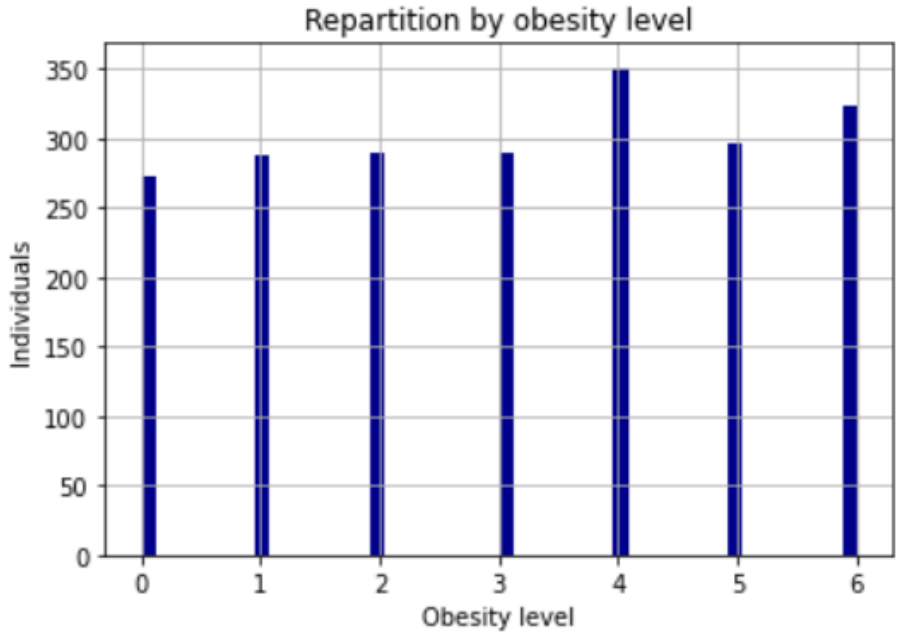
We have a dataset containing 17 attributes of different types. To use all the variables, it is necessary to transform some of them into int type. We will then use an encoder that creates a hierarchy between some of the different answers (for example : no/ sometimes / frquently / always transformed into 0/1/2/3)

	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
2	Male	23.000000	1.800000	77.000000	yes	no	2.0	3.0	Sometimes	no	2.000000	no	2.000000	1.000000
3	Male	27.000000	1.800000	87.000000	no	no	3.0	3.0	Sometimes	no	2.000000	no	2.000000	0.000000
4	Male	22.000000	1.780000	89.800000	no	no	2.0	1.0	Sometimes	no	2.000000	no	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
2106	Female	20.976842	1.710730	131.408528	yes	yes	3.0	3.0	Sometimes	no	1.728139	no	1.676269	0.906247
2107	Female	21.982942	1.748584	133.742943	yes	yes	3.0	3.0	Sometimes	no	2.005130	no	1.341390	0.599270
2108	Female	22.524036	1.752206	133.689352	yes	yes	3.0	3.0	Sometimes	no	2.054193	no	1.414209	0.646288
2109	Female	24.361936	1.739450	133.346641	yes	yes	3.0	3.0	Sometimes	no	2.852339	no	1.139107	0.586035
2110	Female	23.664709	1.738836	133.472641	yes	yes	3.0	3.0	Sometimes	no	2.863513	no	1.026452	0.714137

9 variables transformed into integer variables

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

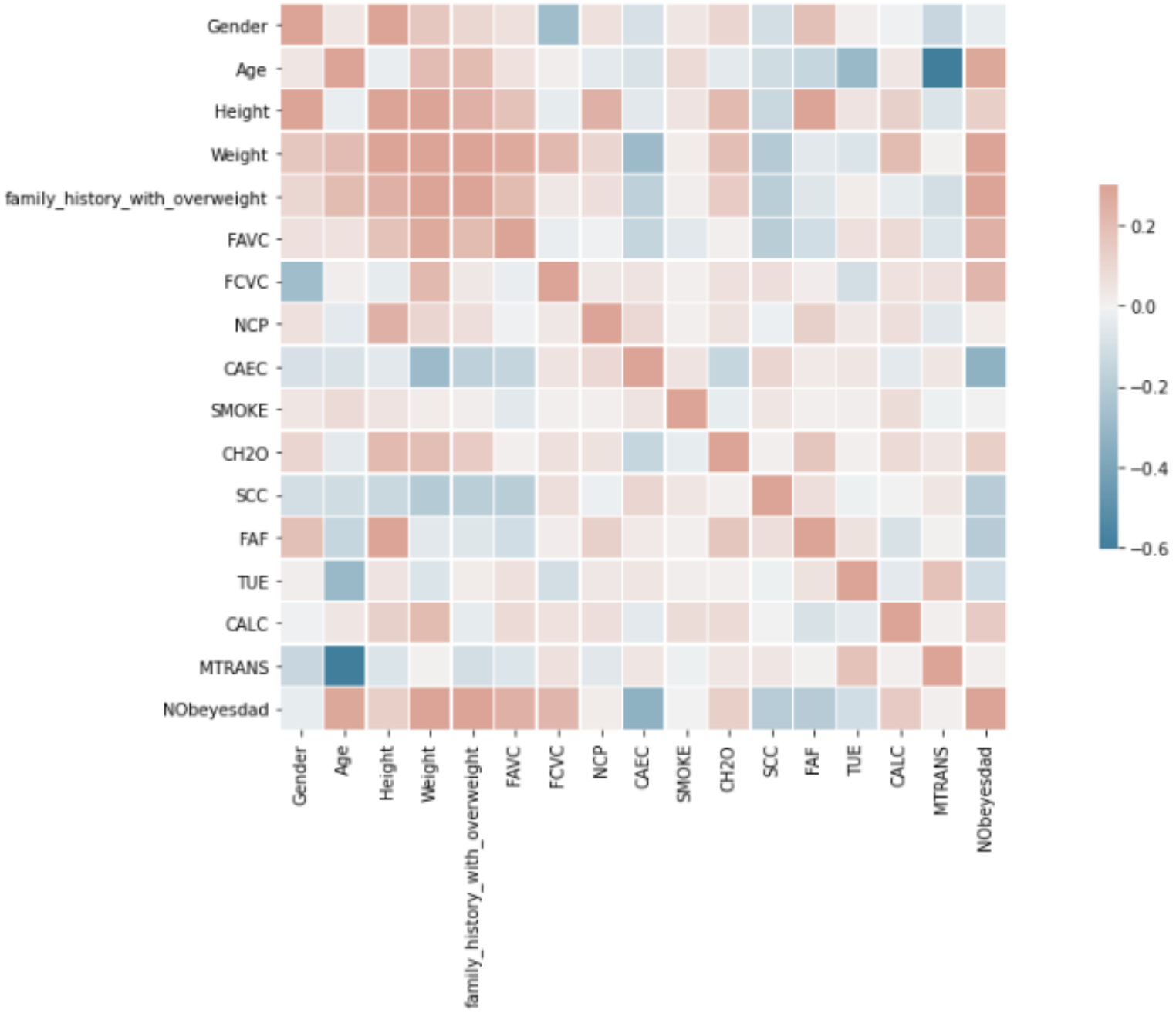
2111 rows × 17 columns



- Proportion of individuals from each category

To process the data, we create a heatmap to see the correlation between the different variables and their impact on the obesity level

We can assume that some of the variables are correlated without showing any statistics (For example the weight is directly related to the obesity level)

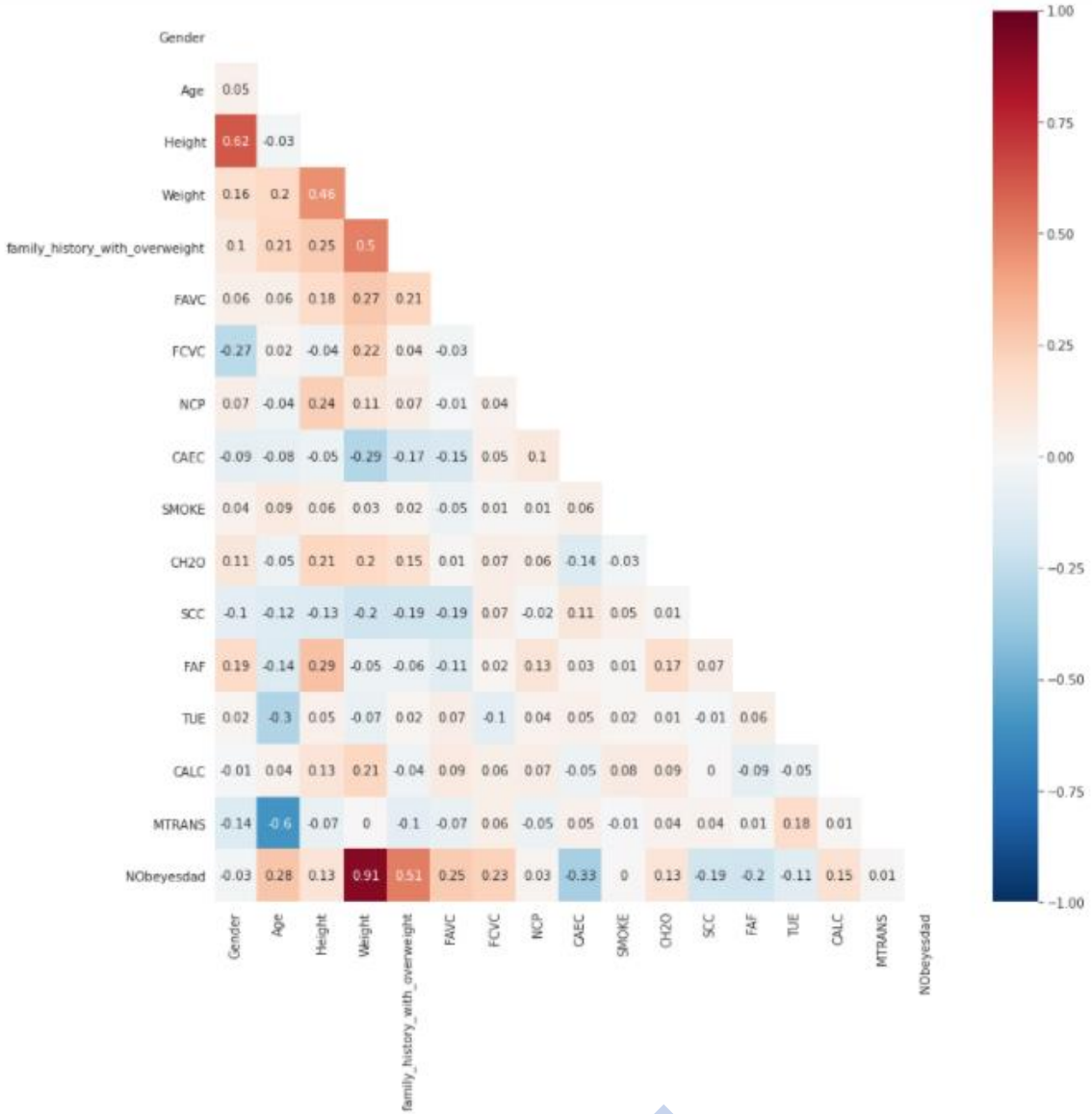


Heatmap of the correlations

We’re using for our models all the variables where |Correlation| ≥ 0,19

Variables most correlated with the obesity level :

- Age
- Weight
- Height
- Family history with overweight
- FAVC
- FCVC
- CAEC
- FAF
- CALC



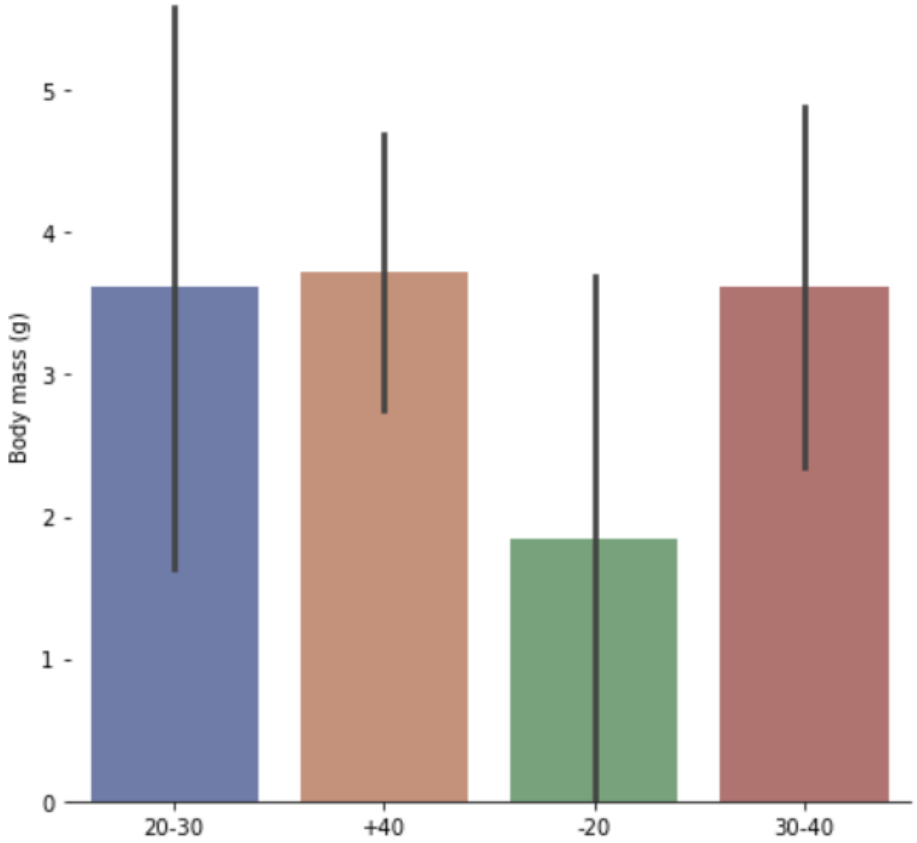
Plots and interpretations

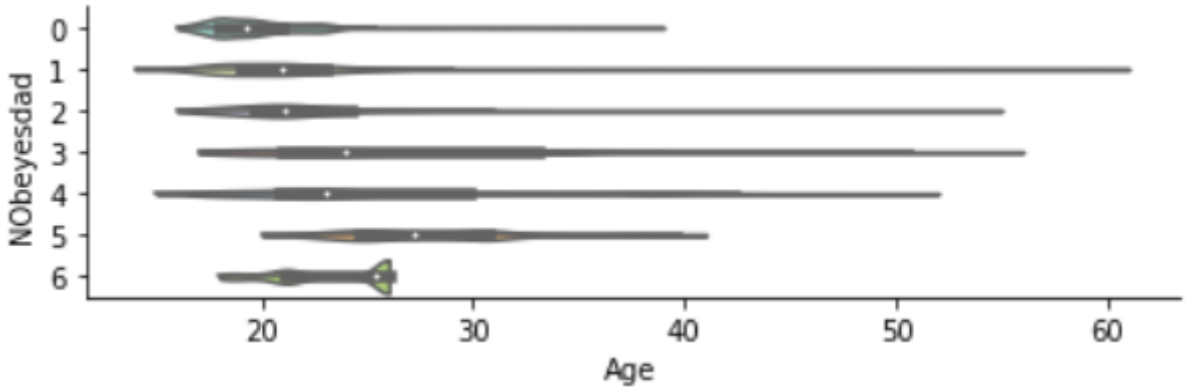
- All the plots we created involving the Age variable are using a class variable (not continuous) representing the age class.
- It was added as a new column in the dataframe « Age_categories »

```
df_cleaned["Age_categories"].value_counts()
```

```
20-30    1162
-20       537
30-40     293
+40        57
Name: Age_categories, dtype: int64
```

- Average obesity level for each age category : the third category (less than 20 y shows that the average level is between normal weight and overweight level I

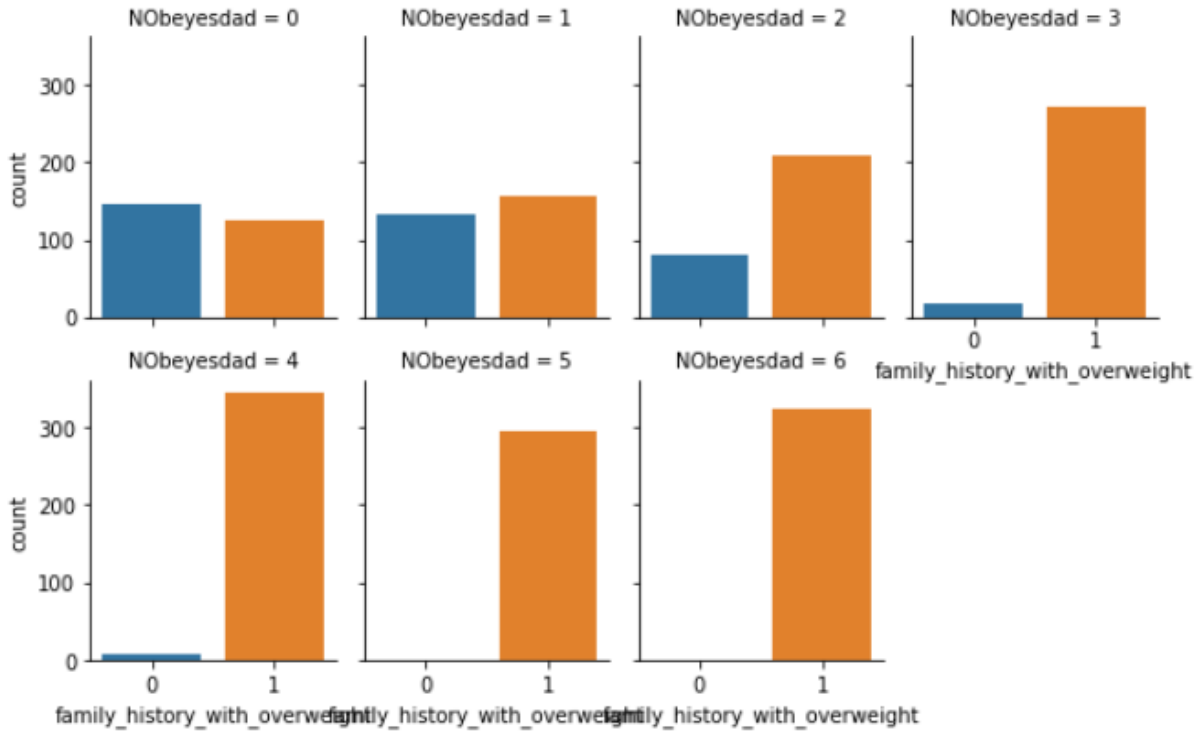




- Age representation for each obesity level

Family history of overweight

Presence of family history of overweight for each obesity level. Blue bars represent the « no » answers and orange « yes ».



- Levels 0 and 1 (Insufficient and normal weight) : balanced répartition between yes and no

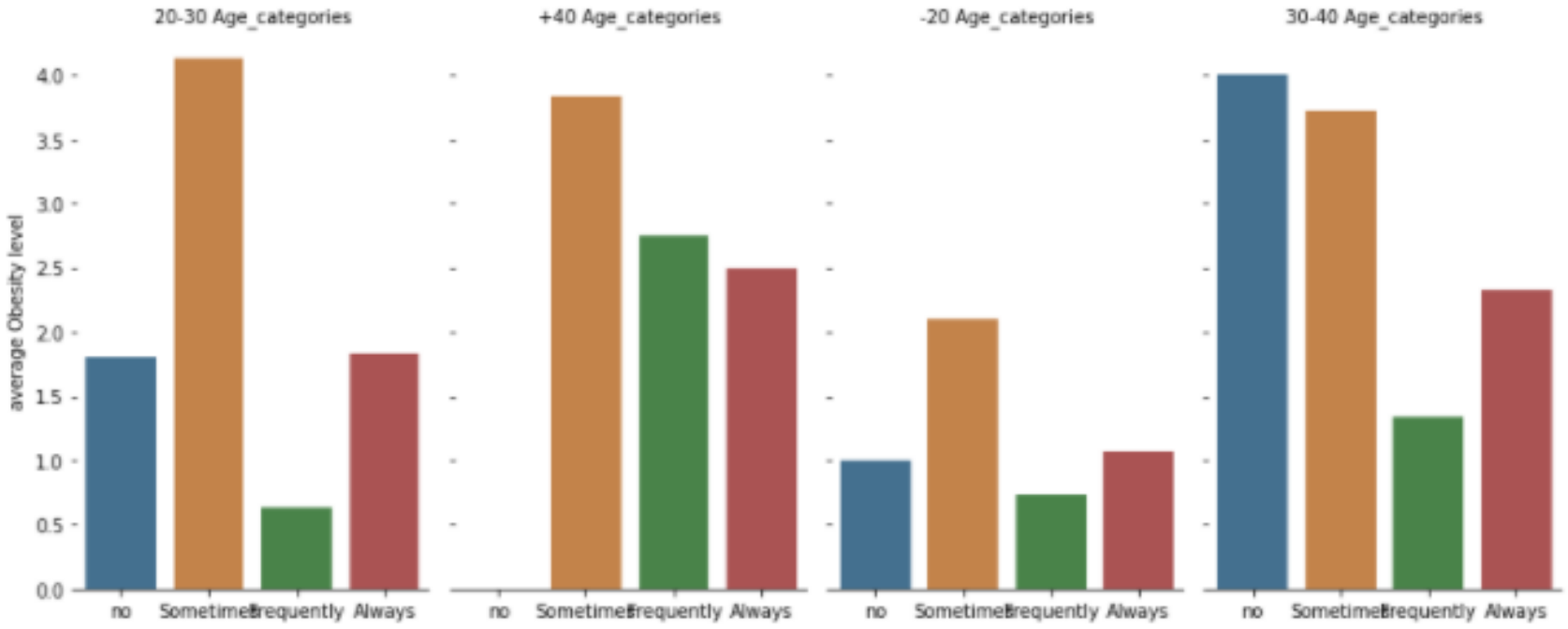
- Levels 2 and 3 (overweight level 1 and overweight level 2) : large proportion of persons with family history of overweight

- Level 4,5 and 6 (obesity types 1, 2 and 3) : no or very little proportion of persons without family history of overweight

- It shows us that the physical condition is very correlated with the family history of overweight

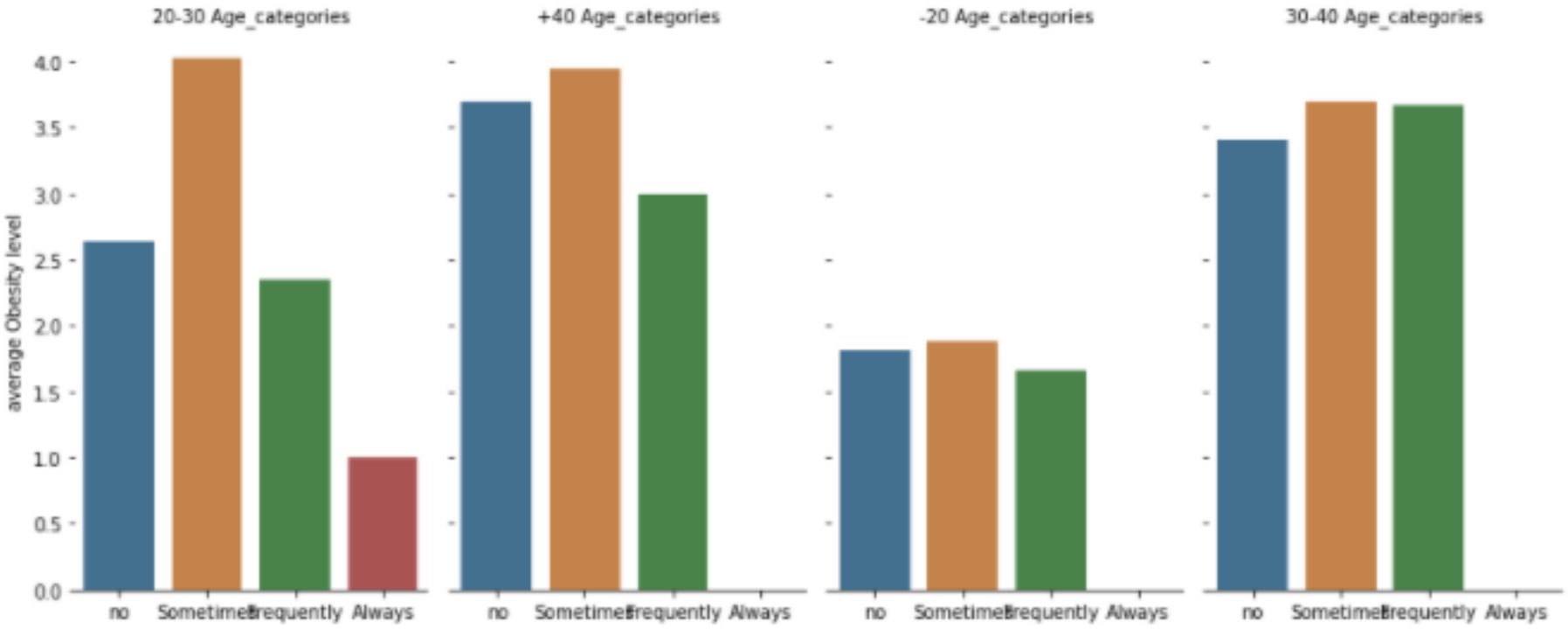
Frequence of eating food between meals (CAEC)

- Representation of the average obesity level for each age category in function of the answer to the question : Do you eat any food between meals? (4 answers possible : no/ sometimes / frequently / always)



- It shows an average obesity level of 4 for people of the 30-40 age class answering no or sometimes. This result means that for this class, the CAEC variable does not explain all the variation, but the ages can explain some behaviors.
- We can understand that the three variables “Age”, “CAEC” and “Nobeyesdad” (Obesity level) are very related.

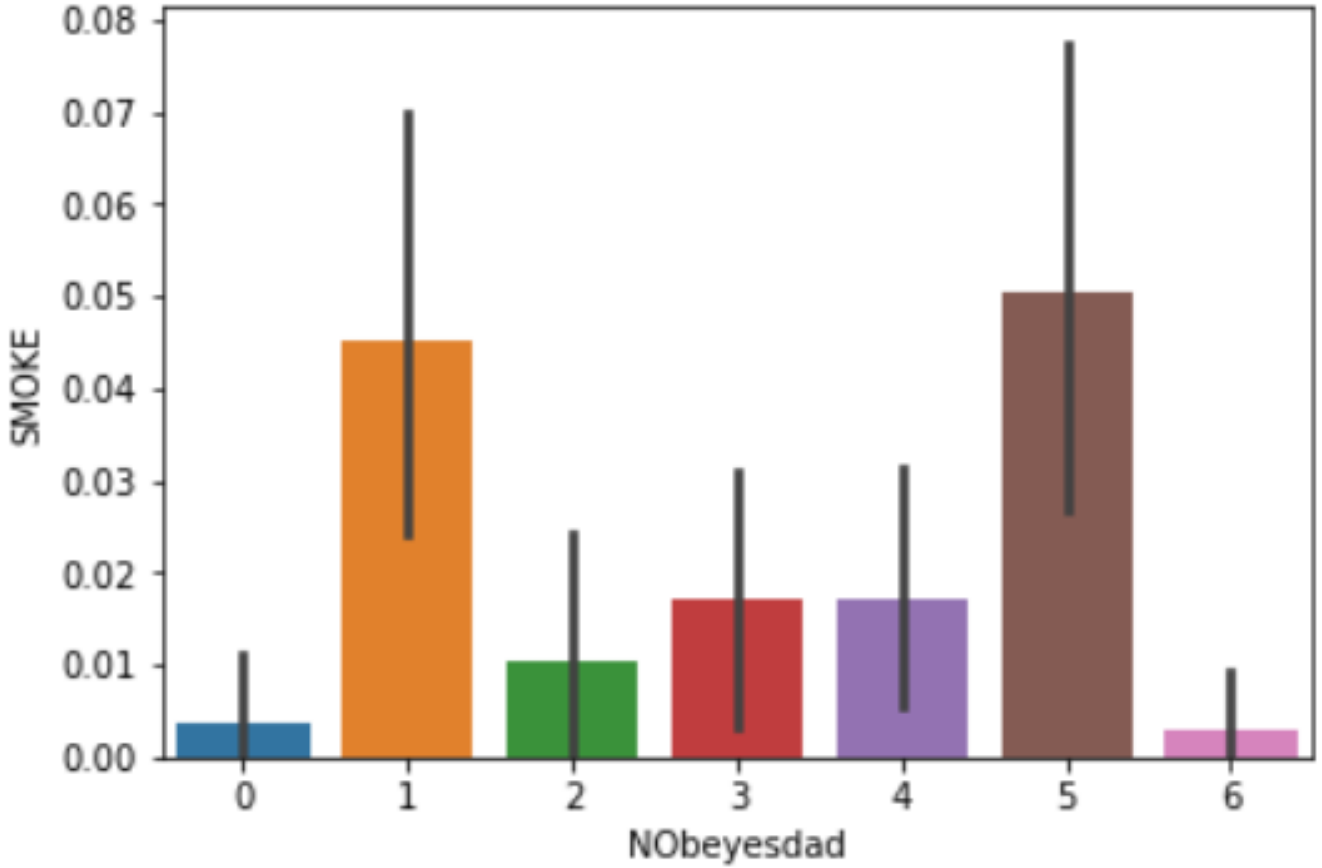
Drink alcohol (CALC)



- Representation of the average obesity level for each age category in function of the answer to the question : How often do you drink alcohol? (4 answers possible : no/ sometimes / frequently / always)

- We want now to study the influence of the alcohol on the obesity level. The heatmap of correlations showed a correlation between CALC and Obesity level of 0,15. This result can be interpreted as high. The graph shows a repartition of the four categories (no/ sometimes/ frequently/ always) more or less balanced. The average obesity level is significatively different between the four categories only for the 20-30 age category. This variable may not be the most useful for our model.

Study of the SMOKE variable



To ensure this variable has no big relation to the obesity level, we plot the proportion of smokers for each level.

We can assume there is no big link between the two variables since the répartition doesn’t follow any logical pattern.

Variables used to create a model

- Age
- Weight
- Height
- Family history
- FAVC
- FCVC
- CAEC
- FAF

These are the variables with the highest correlation with the obesity level, and the most significative.

We add the « Height » variable which is used to calculate the obesity level of a person.

II. Machine learning models

In this part of the project, we implement multiple models of prediction, and we compare the results to obtain the best prediction model.

Processing the data

- Creation of a dataset that contains only the columns we chose in the previous part.

	Age	Age_categories	Weight	Height	family_history_with_overweight	FAVC	FCVC	FAF	CAEC	NObeyesdad
0	21.000000	20-30	64.000000	1.620000	1	0	2.0	0.000000	1	1
1	21.000000	20-30	56.000000	1.520000	1	0	3.0	3.000000	1	1
2	23.000000	20-30	77.000000	1.800000	1	0	2.0	2.000000	1	1
3	27.000000	20-30	87.000000	1.800000	0	0	3.0	2.000000	1	2
4	22.000000	20-30	89.800000	1.780000	0	0	2.0	0.000000	1	3
...	...	...	...	...	...	...	...	...	...	...
2106	20.976842	20-30	131.408528	1.710730	1	1	3.0	1.676269	1	6
2107	21.982942	20-30	133.742943	1.748584	1	1	3.0	1.341390	1	6
2108	22.524036	20-30	133.689352	1.752206	1	1	3.0	1.414209	1	6
2109	24.361936	20-30	133.346641	1.739450	1	1	3.0	1.139107	1	6
2110	23.664709	20-30	133.472641	1.738836	1	1	3.0	1.026452	1	6

2111 rows × 10 columns

	Age	Weight	Height	family_history_with_overweight	FAVC	FAF	FCVC	CAEC
0	21.000000	64.000000	1.620000	1	0	0.000000	2.0	1
1	21.000000	56.000000	1.520000	1	0	3.000000	3.0	1
2	23.000000	77.000000	1.800000	1	0	2.000000	2.0	1
3	27.000000	87.000000	1.800000	0	0	2.000000	3.0	1
4	22.000000	89.800000	1.780000	0	0	0.000000	2.0	1
...	...	...	...	...	...	...	...	...
2106	20.976842	131.408528	1.710730	1	1	1.676269	3.0	1
2107	21.982942	133.742943	1.748584	1	1	1.341390	3.0	1
2108	22.524036	133.689352	1.752206	1	1	1.414209	3.0	1
2109	24.361936	133.346641	1.739450	1	1	1.139107	3.0	1
2110	23.664709	133.472641	1.738836	1	1	1.026452	3.0	1

2111 rows × 8 columns

- Separation of the target variable (Nobeyesdad : obesity level) from the dataset :

- x contains our features and y contains the target

```
y

0        1
1        1
2        1
3        2
4        3
       ..
2106     6
2107     6
2108     6
2109     6
2110     6
Name: NObeyesdad
```

Separation of the two subsets into a train and a test set

```
X_train, X_test,Y_train,Y_test=train_test_split(x, y, test_size= 0.3)
```

```
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(1477, 8)
(634, 8)
(1477,)
(634,)
```

- This operation allows us to train a model and then to test it on another set (with labels) and calculate the accuracy of the model.

- 70% of the data belongs to the train set and 30% to the test set.

- The next step consist of scaling the data : it reduces the distance between data in order to avoid it to vary widely. The algorithms which use the calculation of a distance are more likely to make less mistakes.

- The scaler we use here is a MinMaxScaler() from scikit learn library that makes a normalization of the data. We fit the scaler to the train set (only) to permit the model to adapt to a similar problem of prediction but to avoid overfitting (by adapting too much to this data).

```
minmax_scale = MinMaxScaler().fit(X_train)
```

```
X_train = minmax_scale.transform(X_train)
X_test = minmax_scale.transform(X_test)
```

First model : DecisionTreeClassifier()

The Decision tree model is used for classification problems. It is based on a nodes of a tree which are separated regarding the value of the different features used.

```
model=DecisionTreeClassifier( )
model.fit(X_train,Y_train)
print(model.score(X_train,Y_train))
print(model.predict(X_test))
print(model.score(X_test,Y_test))
```

With this model, we have a score (based on the accuracy calculation) of ~95% on the test set.

```
0.9526813880126183
```

Second model : RandomForestClassifier()

RandomForestClassifier() consists of a number of iteration of classification trees. It is often more precise than a decision tree classifier.

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

#print(random_forest.score(X_train, Y_train))
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
acc_random_forest_test = round(random_forest.score(X_test, Y_test) * 100, 2)
print(acc_random_forest_test)
```

Here our model gives a result of 95,9% of accuracy : it is more precise than the first one

```
95.9
```

Third model : The K-Nearest Neighbors Classifier (KNeighborsClassifier())

- This model consists of a classification of the data based on the calculation of the distance to a point. The class affected to the data is the same class as the nearest point already defined.

- Here, we can see that the KNN classifier gives us a lower accuracy score than the two other models. It is not the most adapted to our problem.

```
modelKNN_1 = KNeighborsClassifier(n_neighbors = 3)
modelKNN_1.fit(X_train, Y_train)
print('train score', modelKNN_1.score(X_train,Y_train))
print('test score', modelKNN_1.score(X_test,Y_test))
```

```
train score 0.8957345971563981
test score 0.807570977917981
```

Finding the best parameters : GridSearch

- GridSearchCV() is a tool that offers the possibility to find the best parameters between a set of given parameters to the algorithm.
- We can use it to refine our search but we have to choose the set of parameters we want to try.
- Here, we chose to change the following :

```
from sklearn.model_selection import GridSearchCV
random_grid = {'bootstrap': [True, False],
 'max_depth': [10, 20, 30, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5],
 'n_estimators': [200, 400, 600]}
rf = RandomForestClassifier()
rf_random = GridSearchCV(estimator = rf,
                         param_grid= random_grid,
                          cv = 3, verbose=2,
                        n_jobs = -1)

rf_random.fit(X_train, Y_train)

print(rf_random.best_params_)
```

- GridSearchCV() gives us the best parameters of the set we gave to the algorithm :

```
rf_random.best_params_
```

```
{'bootstrap': False,
 'max_depth': 30,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 600}
```

```
rf_random.best_score_
```

```
0.9323056668700561
```

Parameters

- GridSearchCV() gives us the best parameters of the set we gave to the algorithm :
- n_estimators = number of trees
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points

We are now able to choose parameters that will improve the best accuracy score

```
rf_random.best_params_
```

```
{'bootstrap': False,
 'max_depth': 30,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 600}
```

```
rf_random.best_score_
```

```
0.9323056668700561
```

•This gridSearch gave us as the best score ~0,9323. However, we have found a better score without changing those parameters. Then, we will keep the best of the previous models.

The best model : RandomForestClassifier()

- Since we obtain the best accuracy results for the RandomForestClassifier, we can conclude that this is our best model to make predictions on our target variable : Obesity Level.

(accuracy of 95,9%)

Comparaison with a model implementing all the variables of the dataset and conclusion

To verify the corectness of our best model, we compared it with a model implementing all the variables of the dataset. We didn't remove any variable and observed changes.

The best accuracy score that we obtain for the different model is of 94% which is lower than our best score

We can understand that the models created with all the data were all less precise than our best model lade with processed data. (randomForestClassifier())

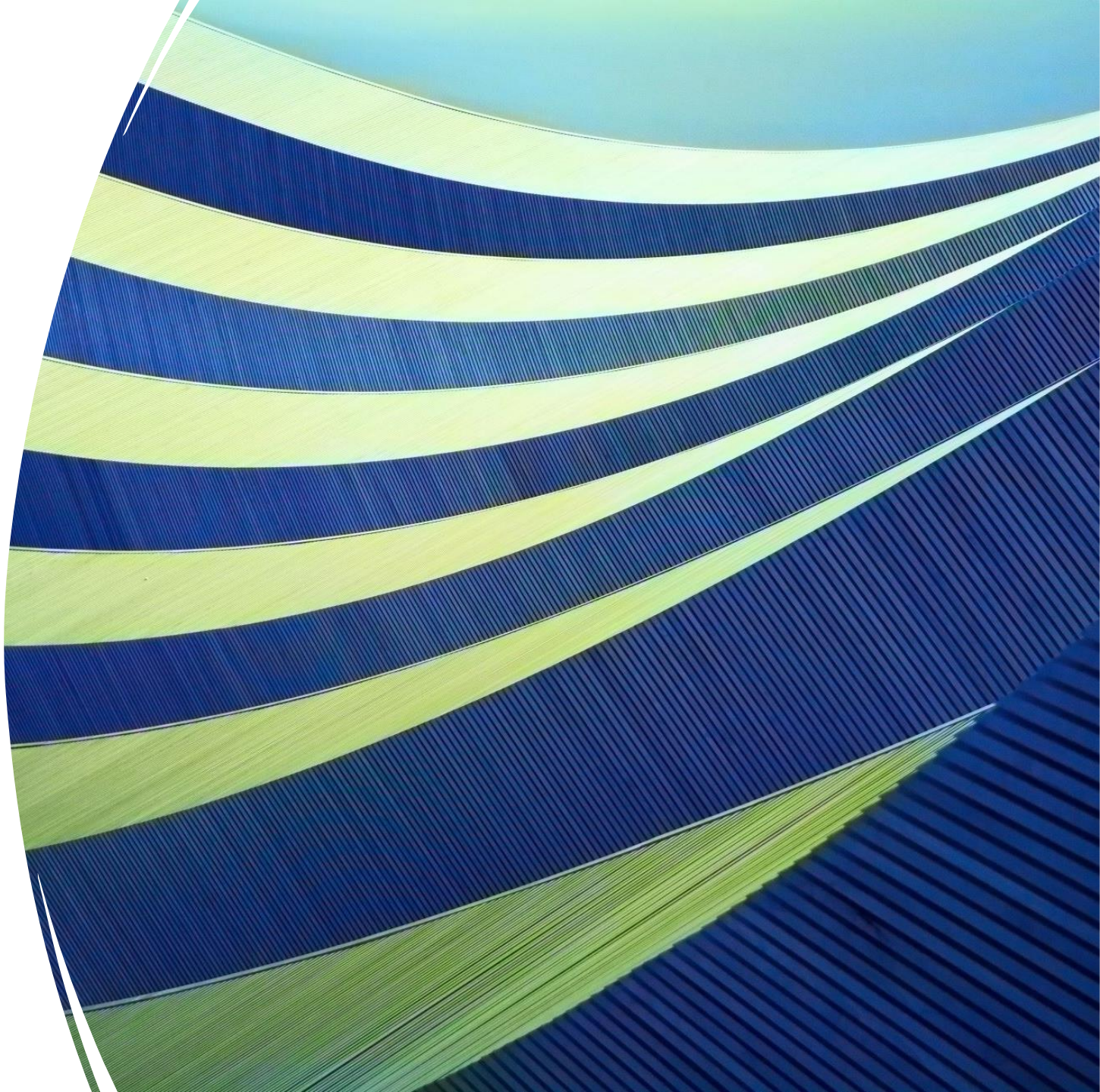
94.16 vs 95.9

To go further

Since we had a dataset that implied a classification problem, we found another way to interpret the set :

We transformed our problem to a regression problem, by using the variable « Weight » as the target variable.

We implemented a model that can predict our target data with a MSE (mean squared error) more or less correct.



Models and Accuracy

Implementation of 4 different models :

- BayesianRidge()
- DecisionTreeRegressor()
- LinearRegression()
- RandomForestRegressor()

- The cost function used here is the MSE (Mean Squared Error), for regression problems.

Best score obtained

Study made first on a dataset with 6 variables, then 8, then 9.

The 9 most correlated variables to Weight :

"FAVC","CAEC","family_history_with_overweight", "NObeyesdad","FCVC","CALC","CH2O","SCC","NCP"

RandomForestRegressor() : gives the best results, with the set of 9 variables

A MSE of 77,95 vs 115,4 for our first model

Conclusion of the second study

It is an interesting study that shows we can use variables to predict another one, but we can also invert the roles of the variables to select another target variable. It changes the dimension of the problem, converting it from a classification problem to a regression problem.