Python for Data Analysis Project

Clément MUFFAT-JOLY Jean MARCHAND – DIA4

INTRODUCTION

We worked on the Dataset for estimation of **obesity levels** based on eating habits and physical condition in individuals from Colombia, Peru and Mexico.
The information has 17 features and 2111 records.

About the features of the dataset

The **attributes** related with **eating habits** are:

- Frequent consumption of high caloric food (FAVC)
- Frequency of consumption of vegetables (FCVC)
- Number of main meals (NCP)
- Consumption of food between meals (CAEC)
- Consumption of water daily (CH20)
- Consumption of alcohol (CALC)

The **attributes** related with the **physical condition** are:

- Calories consumption monitoring (SCC)
- Physical activity frequency (FAF)
- Time using technology devices (TUE)
- Transportation used (MTRANS)

The **targert** is NObesity with **differents** values:

- Insufficient Weight
- Normal Weight
- Overweight Level I
- Overweight Level II
- Obesity Type I
- Obesity Type II
- Obesity Type III

Import Dataset and EDA

We first **import** the dataset :

1. Import Dataset

```
df = pd.read_csv('ObesityDataSet.csv', encoding='latin-1')
```

Then we started to **explore** the dataset with the head function:

2 Exploration Dataset

```
# Preview of the dataset
df.head()
```

We had to **convert** all **quantitative** features to int type:

```
df['Age'] = round(df['Age']).astype(int)
df['FCVC'] = round(df['FCVC']).astype(int)
df['CH2O'] = round(df['CH2O']).astype(int)
df['FAF'] = round(df['FAF']).astype(int)
df['TUE'] = round(df['TUE']).astype(int)
```

And we had to **convert** all **qualitative** features by replace with int values

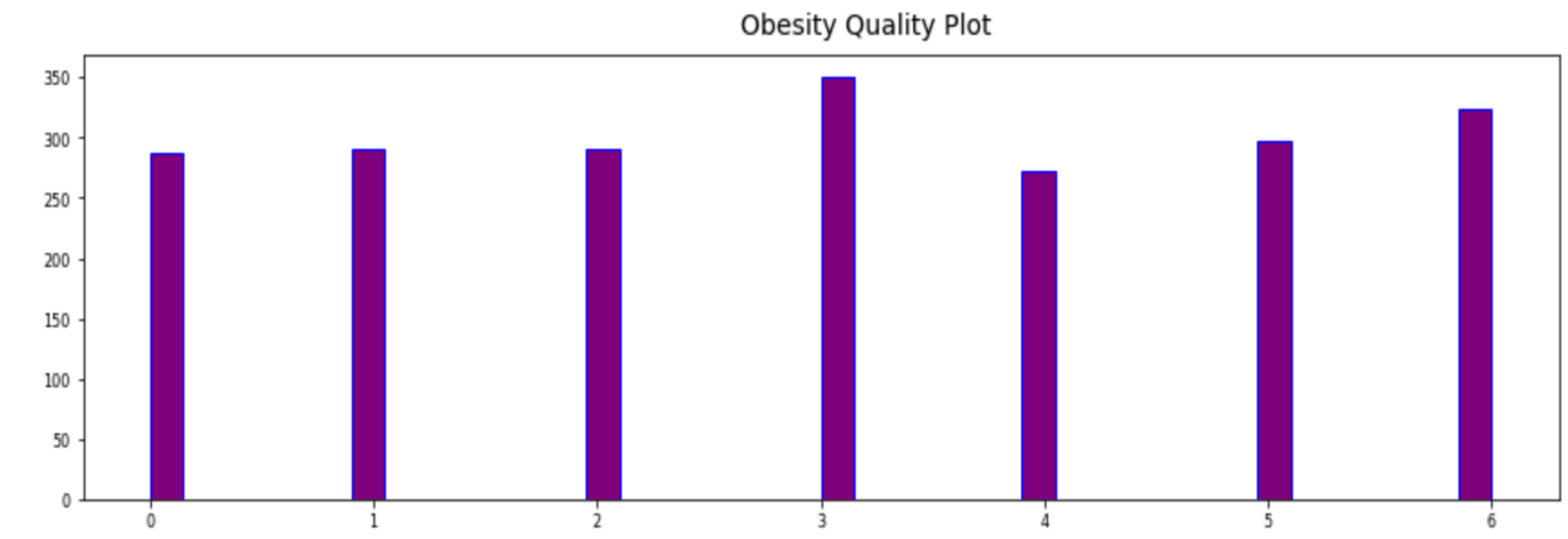
Replace qualitative features by numerical values

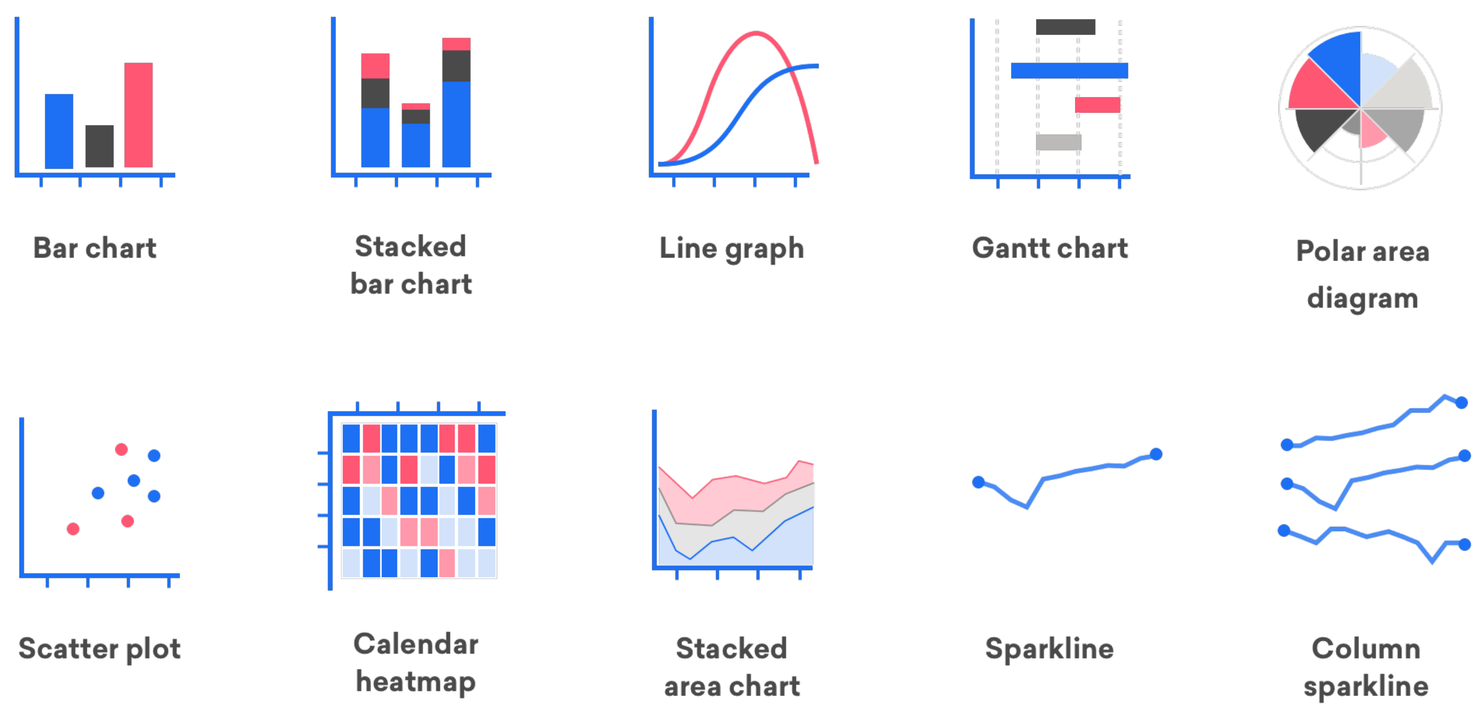
```
df["family_history_with_overweight"].replace(['yes', 'no'],[1,0],inplace=True)
```

```
df["FAVC"].replace(['yes', 'no'],[1,0],inplace=True)
```

Data Visualization

Distribution of obesity categories

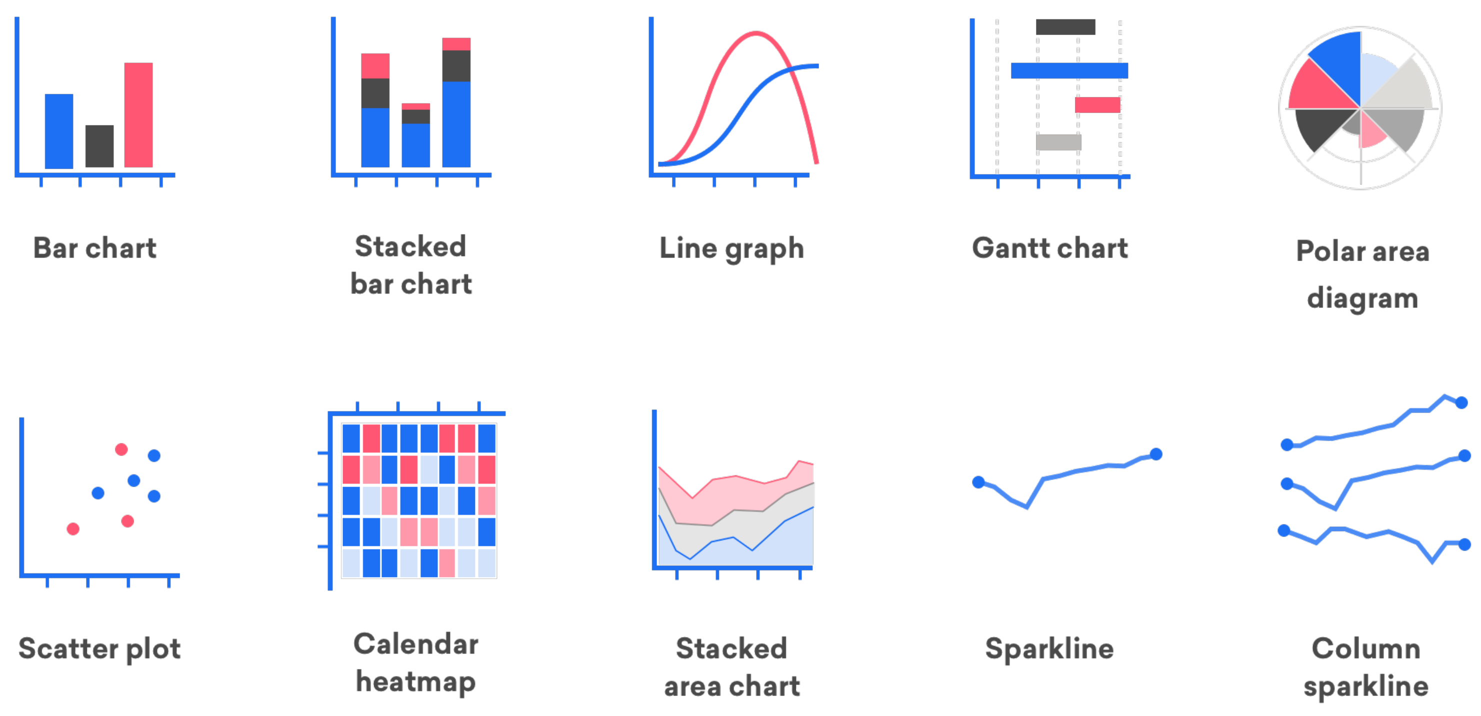


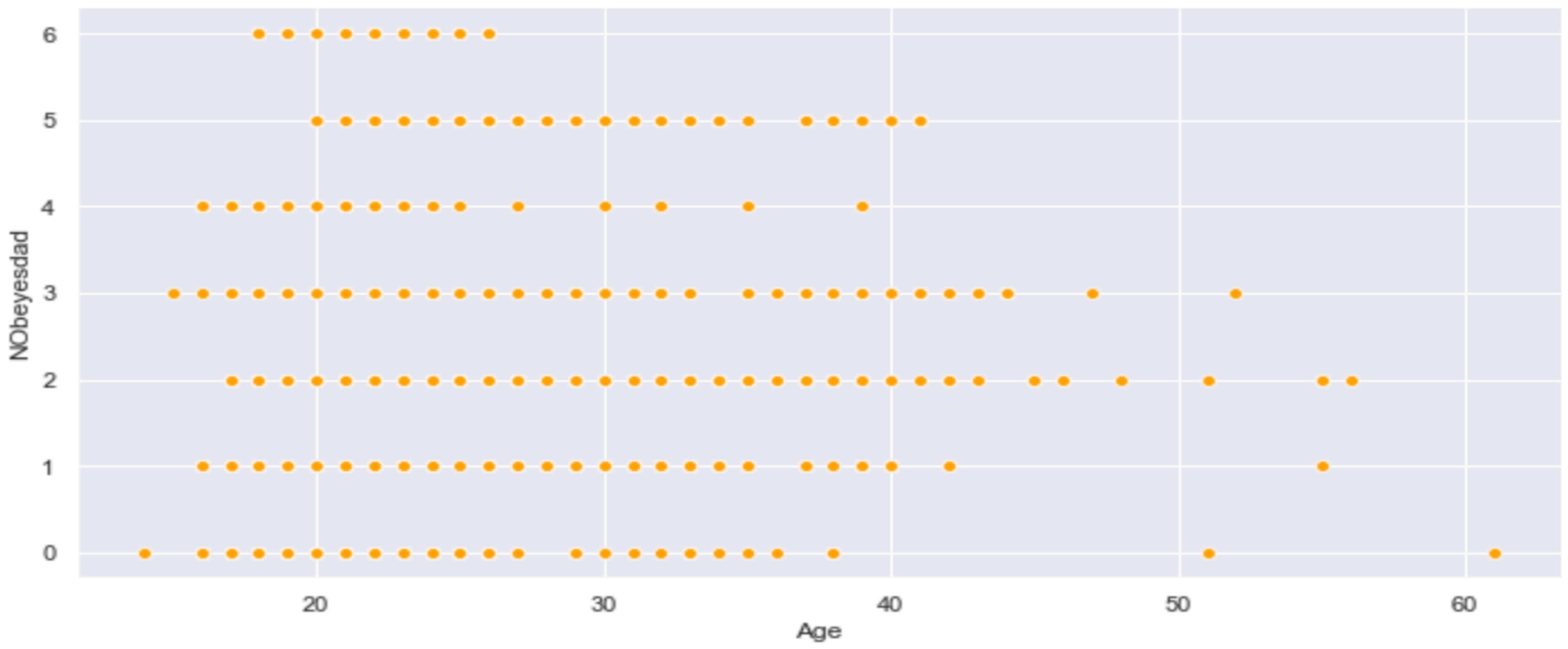


We have **explored** several data **visualizations** in the notebook. Here are 3 interesting graphs that we present to you. First, here is the **distribution** of the **number** of values (people) for **each level of obesity.** Since the qualitative values have been transformed into numbers, 0 corresponds to the most underweight people and 6 the most overweight. In 1 these are the so-called **normal weight** people. We can see that this category includes the **most people** is the 3 corresponds to overweight level 2. However, we can see that the more **extreme weight categories** are **very represented**, which is not a guarantee of good health for most of the people who participated in this survey. Let's try to study two other graphs to **better understand** these data

Data Visualization

Distribution of obesity categories according to age

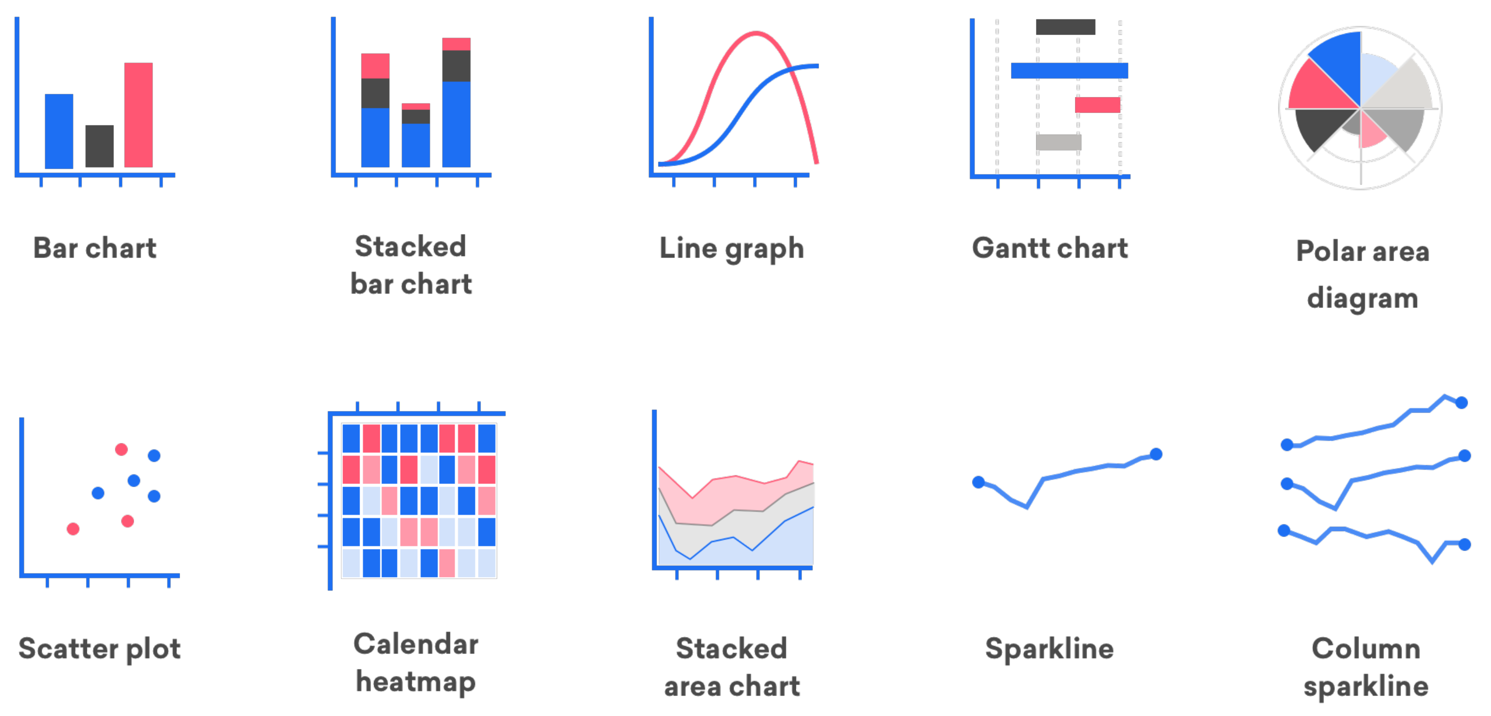


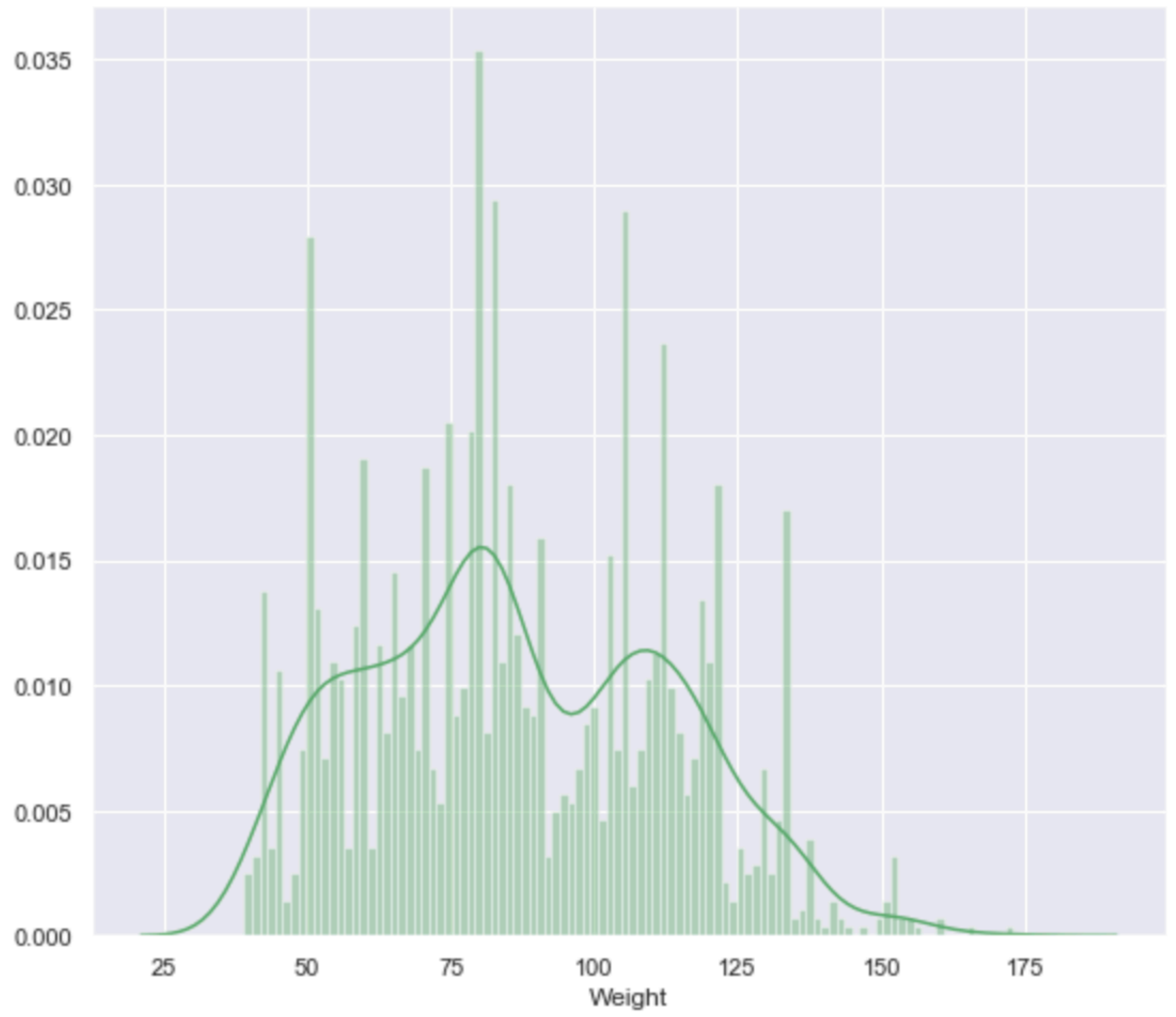


In this second graph we can see how the **categories of obesity** are **distributed** according to **age**. At first glance the data are rather well distributed but we can observe that **no person over 26 years old** is at **level 3 of obesity** (the highest). We also note that **the majority of obesity** levels 1 and 2 are made up **of young people**. Conversely, the **oldest people** are divided between the **level of overweight 2** and the **level of underweight**. Is **age** therefore a **determining** variable in **the level of obesity**? This is what we will see next with the matrix of

Data Visualization

Percentage distribution of the population weight distribution





We can distinguish **3 bumps** on the curve which could imply that the population is **divided** into **3** large **weight groups**. We can see that there are about **as many people** in the **underweight** part as **overweight**. An **interesting** data is the high distribution, about **15%** of the population towards an **average** weight of **75 kg** which is **more** than the **average weight** and confirms the **tendency** that this population is **overweight**.

Pre-Processing

We **drop** features using the **calculation** of the predict value :

Prepocessing data

Now, we can drop some values because they were used for the calculation of the predict value.

```
]: df.drop(['Height','Weight'],1, inplace=True)
```

Then we **split** the data **without** '*Nobeyesdad*' which is the **target** value.

We split our data in training and test data:

```
]: x,y = df.loc[:,df.columns != 'NObeyesdad'], df.loc[:,'NObeyesdad']
   x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 1)
   print(df)
```

We **scale** the value using **StandartScaler** function on **x_train** and **x_test.**

Now we can scale our data:

```
]: sc = StandardScaler()
   x_train = sc.fit_transform(x_train)
   x_test = sc.transform(x_test)
```

We **display** the **correlation map** with sns.heatmap to see the **correlation between features**

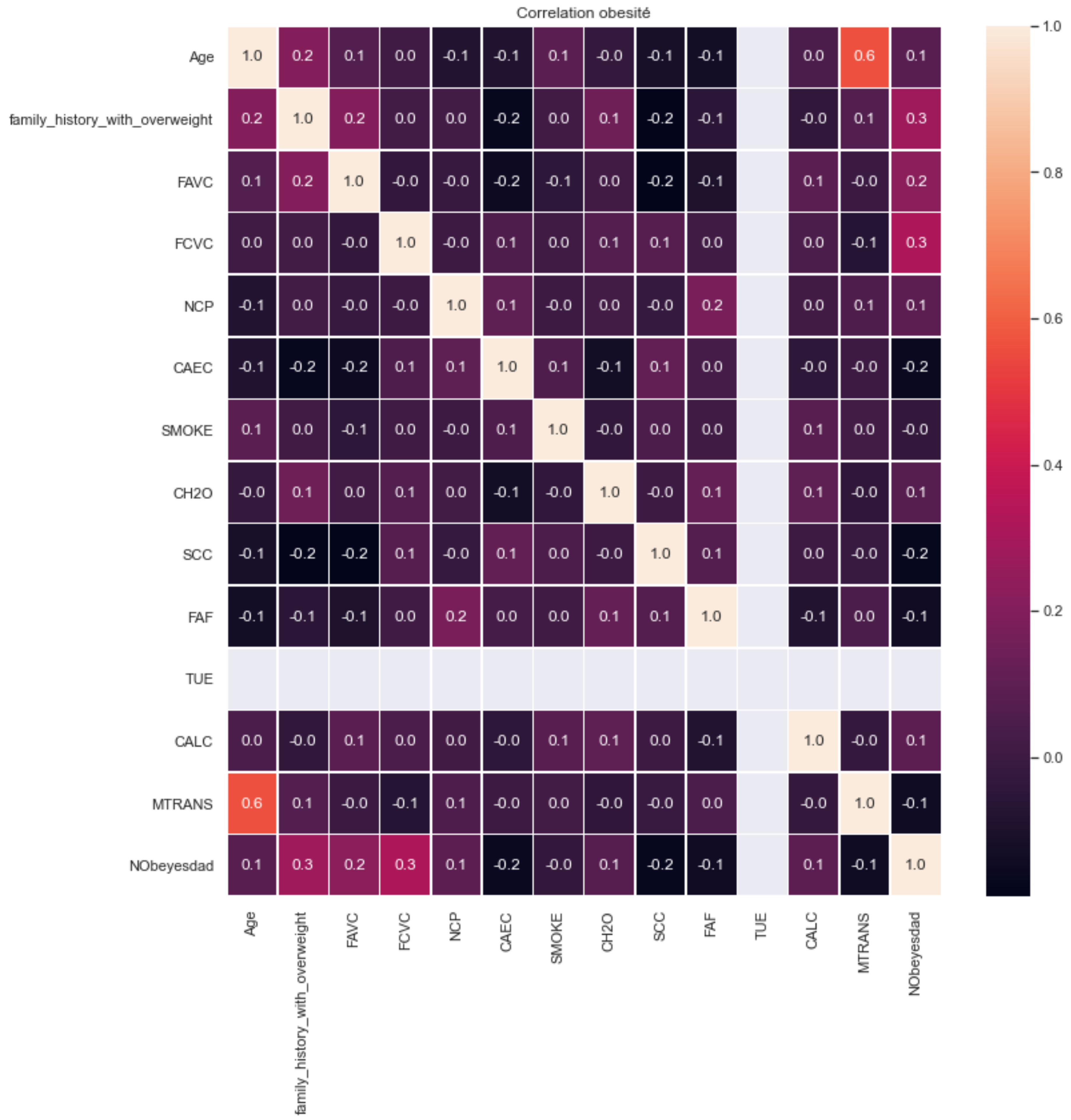
```
: #correlation map
  f,ax = plt.subplots(figsize=(18, 18))
  sns.heatmap(df.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax).set(title="Correlation obesité")
  plt.show()
```

Pre-Processing

Correlation map

Important features to predict obesity:

- We can see that the **family history of obesity** and **(FCVC) the frequency of vegetable consumption** are the **two** variables **most correlated** with the **level of obesity (**with a correlation of **0.3**). They are therefore **decisive**.
- **Next** comes a **correlation** of **0.2** (**FAVC**) for the f**requency of consuming high calorie foods**.
- And **finally** we have a correlation of **0.1**: **age**, the (**NCP**) the **number of meat-based meals**, (**CH20**) the **water consumption** per day and finally (**CALC**) the **alcohol consumption .**

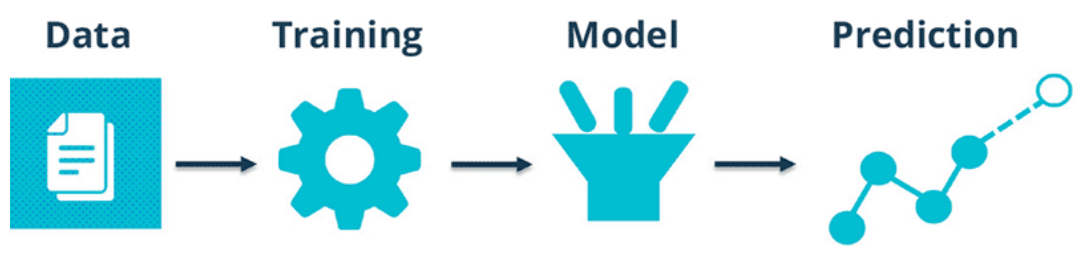


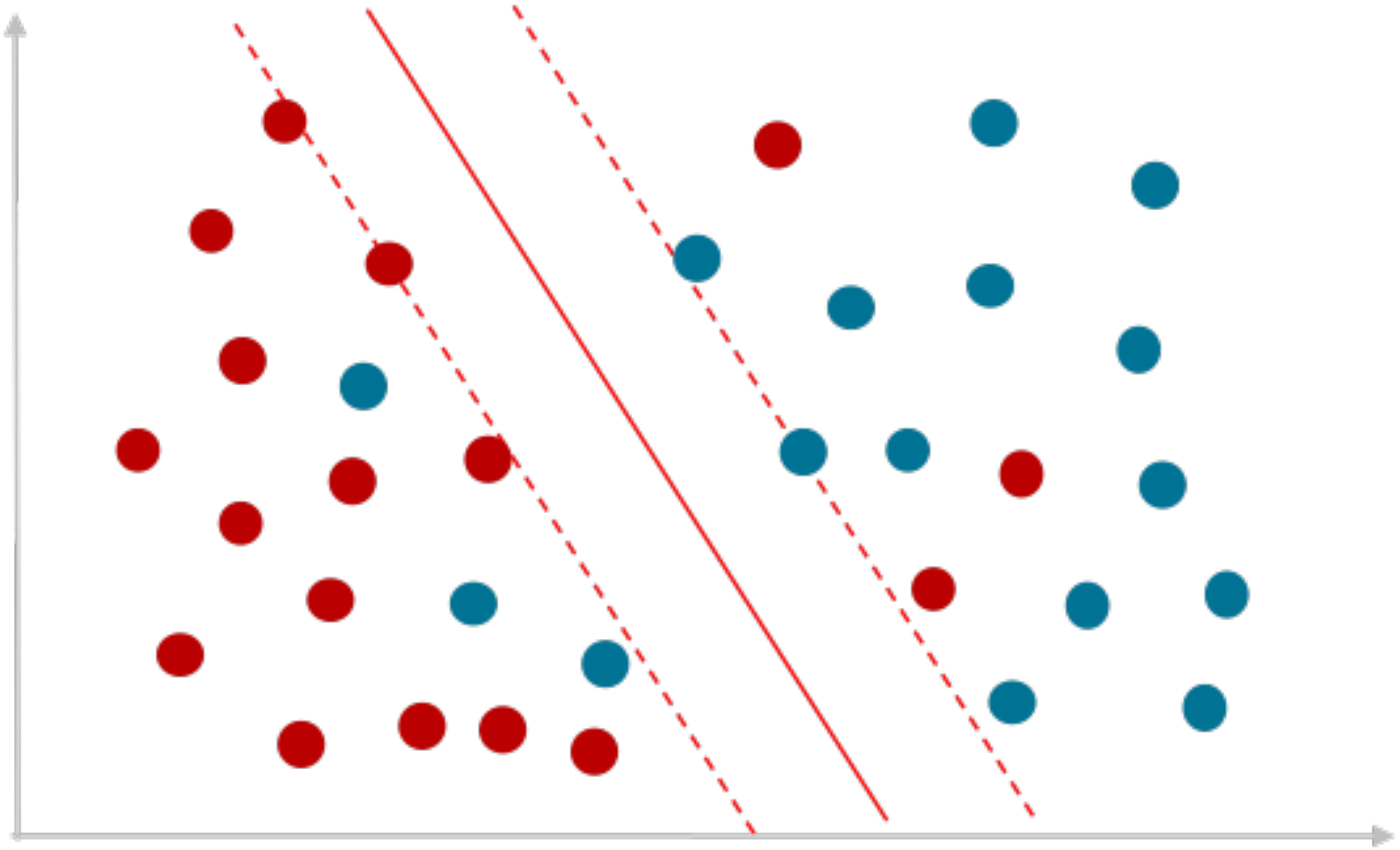
Machine Learning Models

In this part we try to **define** a **model** capable of **best predicting** the **level of obesity** based on certain variables.
We have tested **several models** that we will **present to you** one by one and you show at the end the **classification between them**.

We did :

- KNN
- Logistic Regression
- LDA
- Classification Tree
- Random Forest
- Bagging
- Boosting





Machine Learning

KNN

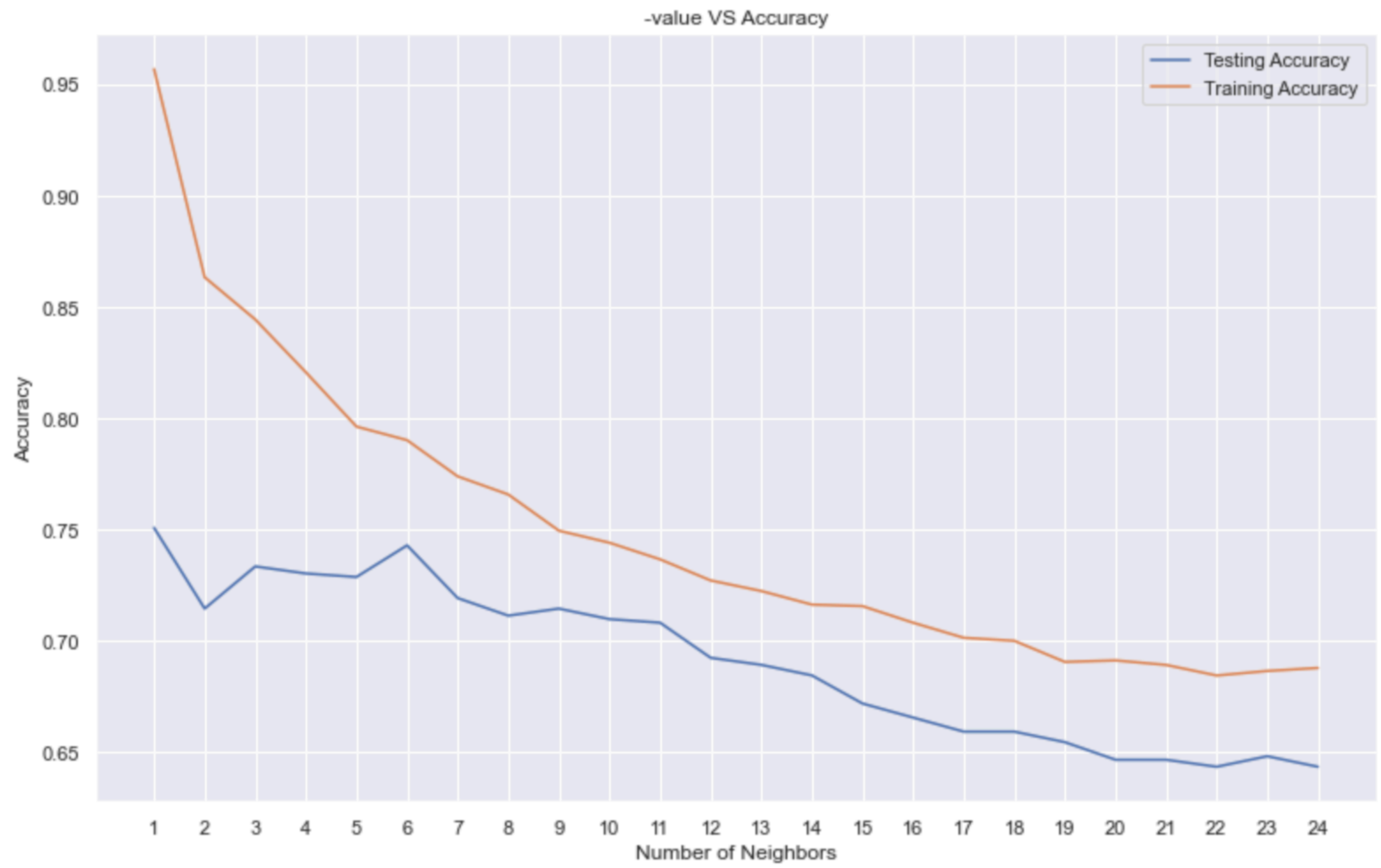
We fit the **KNN model** using **KneighborsClassifier** from sklearn collection. We set to **3** the number of **neighbors**. We can see we have **0,73 accuracy**.

KNN

```
# train test split
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train,y_train)
prediction = knn.predict(x_test)
print('With KNN (K=3) accuracy is: ',knn.score(x_test,y_test)) # accuracy
```

```
With KNN (K=3) accuracy is:  0.7334384858044164
```

We chose to **display** the **accuracy** of **training_set** and **test_set** according to the **number of neighbors** defined in the knn.
We can see that the **best model** is with **1 neighbor**, which gives an **accuracy of 0.75.**




```
Best accuracy is 0.750788643533123 with K = 1
```

Machine Learning

Logistic Regression

We fit the **Logistic Regression model** using **LogisticRegression** from sklearn collection.

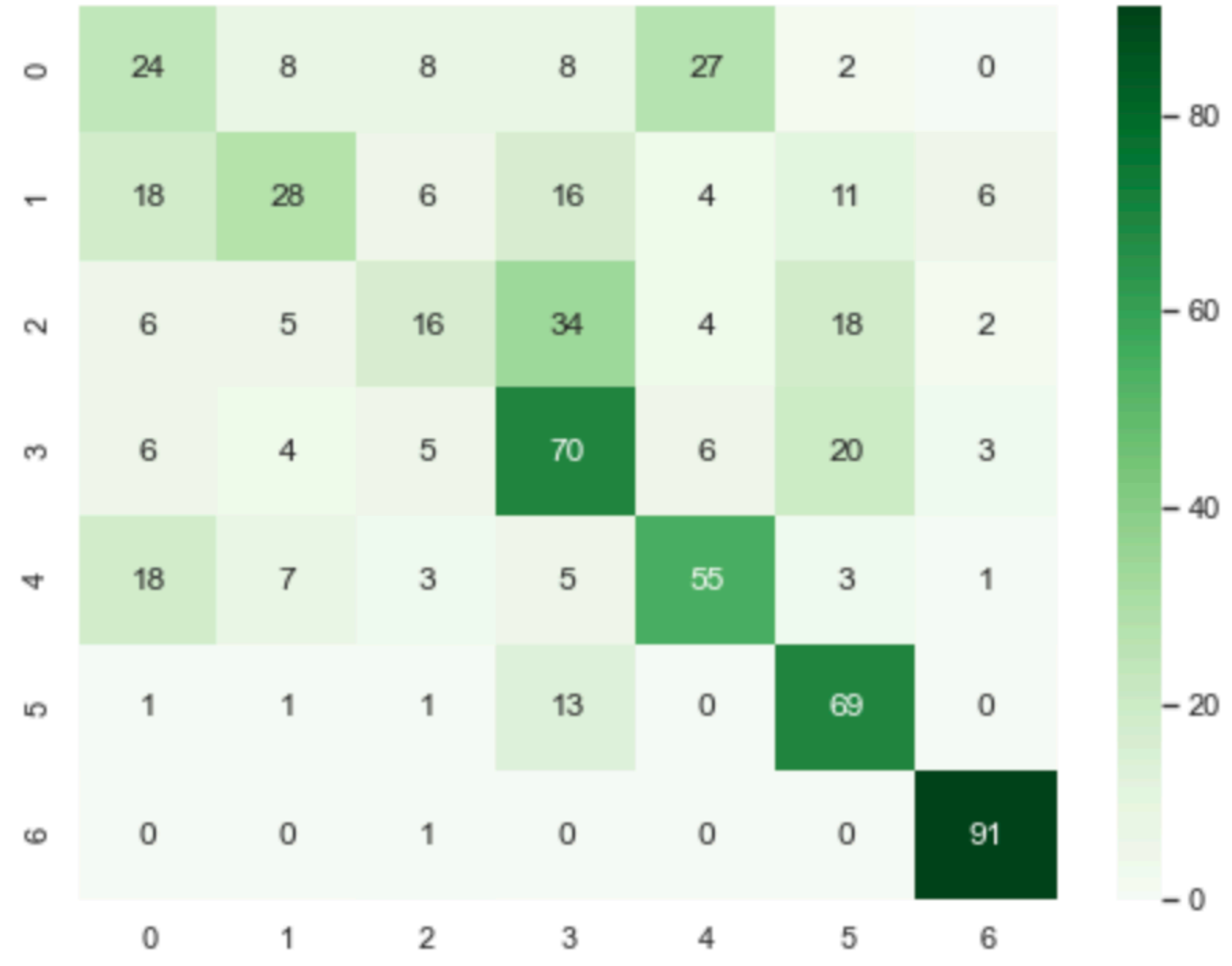
Thanks to the **confusion matrix** we see that the **accuracy** of the model is **0.55**. It's **less** than the **KNN** model.

Logistic Regression

```
]: from sklearn.linear_model import LogisticRegression
logreg= LogisticRegression(max_iter= 2000, random_state=10)
logreg.fit(x_train,y_train)
y_pred= logreg.predict(x_test)
logreg.score(x_train,y_train)
```

```
0.6161137440758294
```

```
Accuracy Score : 0.556782334384858
```



Machine Learning

LDA

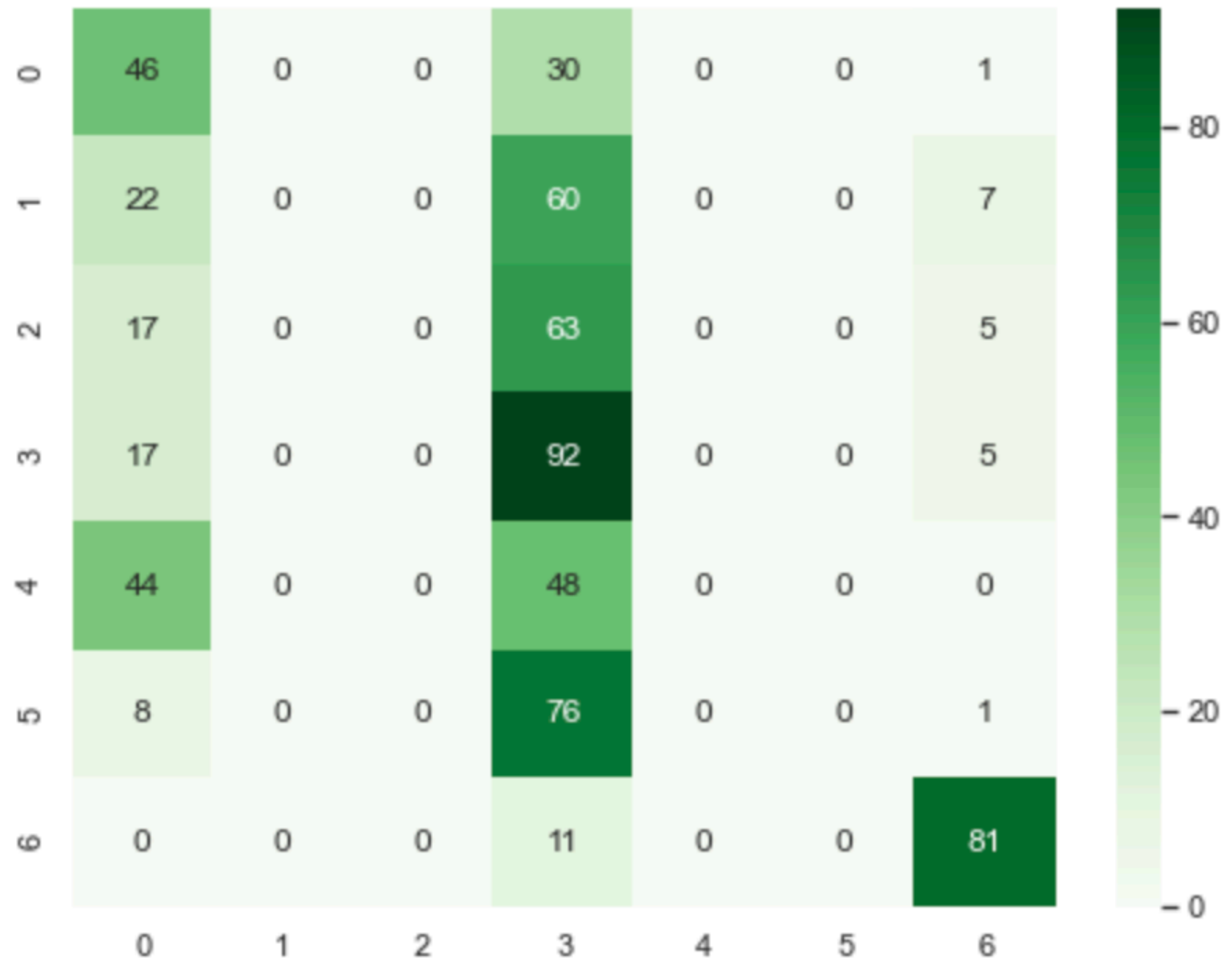
We fit the **LDA model** using **LinearDiscriminantAnalysis** from sklearn collection.

LDA

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=1)
x_train_new = lda.fit_transform(x_train, y_train)
x_test_new = lda.transform(x_test)


classifier = RandomForestClassifier(max_depth=2, random_state=4)

classifier.fit(x_train_new, y_train)
y_pred = classifier.predict(x_test_new)
```

```
Accuracy0.34542586750788645
```

Thanks to the **confusion matrix** we see that the **accuracy** of the model is **0.34**. It's **less** than the **KNN** and **the Logistic Regression** model.



Machine Learning

Decision Tree Classifier

We fit the **Decision Tree model** using **DecisionTreeClassifier** from sklearn collection. We use the ‘*gini*’ criterion.

Decision Tree Classifier

```
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)


# fit the model
clf_gini.fit(x_train, y_train)
y_pred_gini = clf_gini.predict(x_test)
```

```
Accuracy0.485804416403785 47
```

Thanks to the **confusion matrix** we see that the **accuracy** of the model is **0,48**. It’s **less** than the **KNN** and **the Logistic Regression** model but **better** than **LDA** model.

Machine Learning

Random Forest

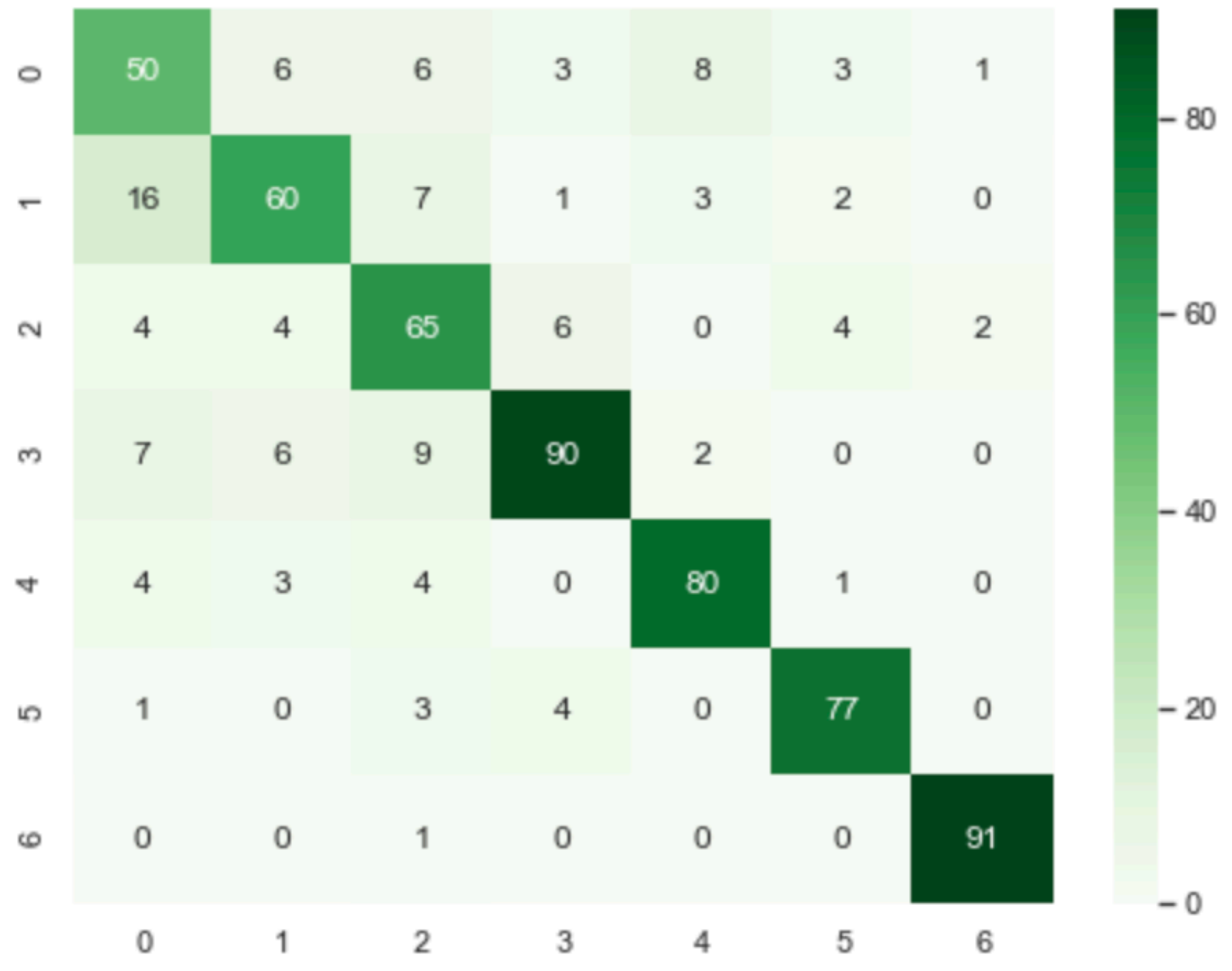
We fit the **Random Forest model** using **RandomForestClassifier** from sklearn collection.

Thanks to the **confusion matrix** we see that the **accuracy** of the model is **0,81**. It's **better** than the **KNN**, the **Logistic Regression, LDA** and the **Decision Tree** model.

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=300, random_state=50)
RF.fit(x_train, y_train)
y_pred = RF.predict(x_test)
```

```
Accuracy0.8091482649842271
```



Machine Learning

Bagging

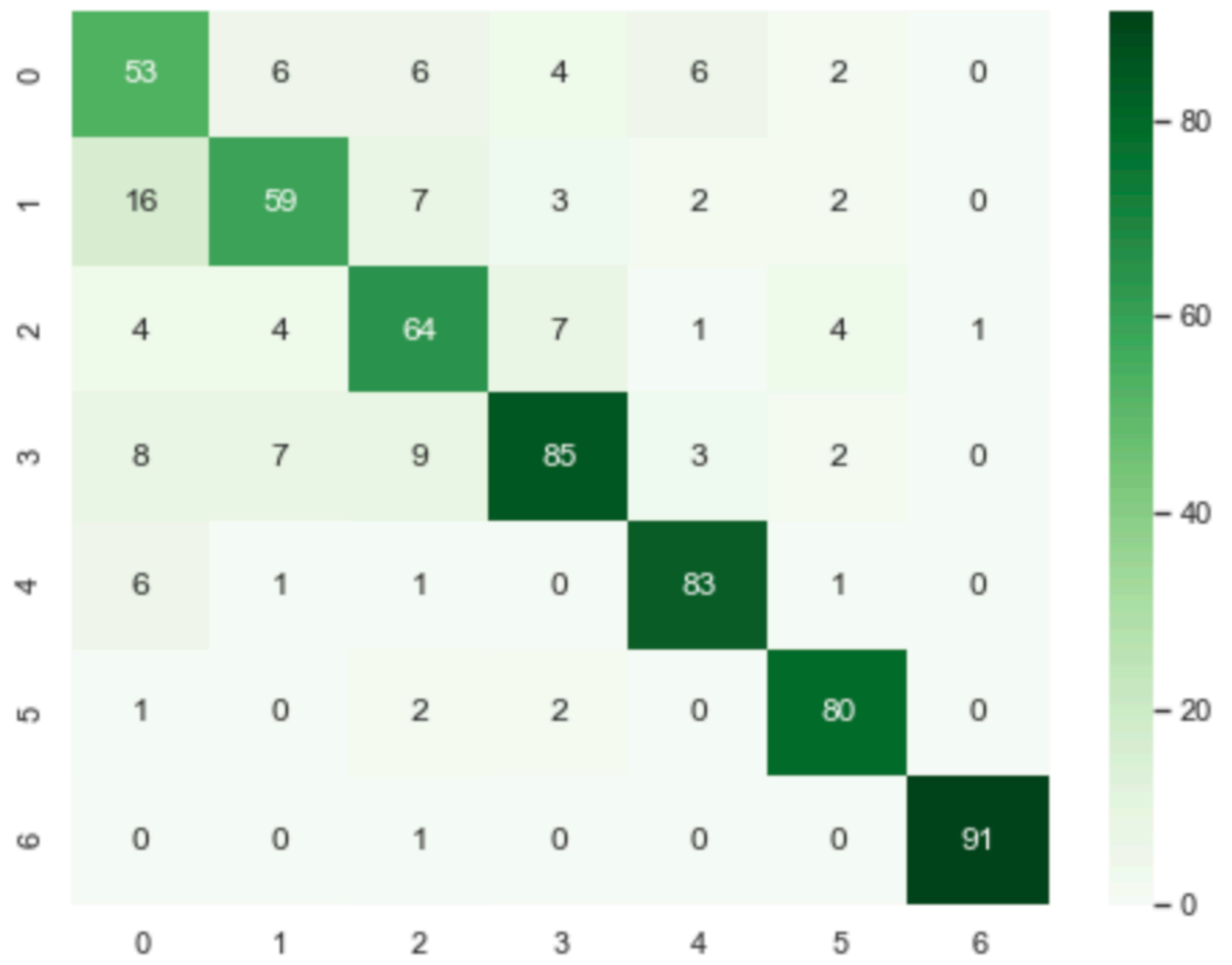
We fit the **Bagging model** using **BaggingClassifier** from sklearn collection.

Thanks to the **confusion matrix** we see that the **accuracy** of the model is **0,81**. It’s **better** than the **KNN**, the **Logistic Regression, LDA, Decision Tree** and the **Random Forest** model.

Bagging

```
from sklearn.ensemble import BaggingClassifier

bagging = BaggingClassifier(n_estimators=300, random_state=50)
bagging.fit(x_train, y_train)

y_pred = bagging.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
```

```
Accuracy0.8123028391167192
```



Machine Learning

Boosting

We fit two **Boosting model.** The first one using **HistGrandientBoostingClassifier** and the second using **GradientBoostingClassifier**.

Thanks to the **confusion matrix** we see that the **accuracy** of the two model is **0,80**. It's **better** than the **KNN**, the **Logistic Regression, LDA, Decision Tree**, **Random Forest** model but **less** than **Bagging**

Histogram-based Gradient Boosting

```
]: from sklearn.experimental import enable_hist_gradient_boosting
   from sklearn.ensemble import HistGradientBoostingClassifier

   boost = HistGradientBoostingClassifier(random_state=50)
   boost.fit(x_train, y_train)
```

```
Accuracy0.804416403785489
```

Gradient Boosting

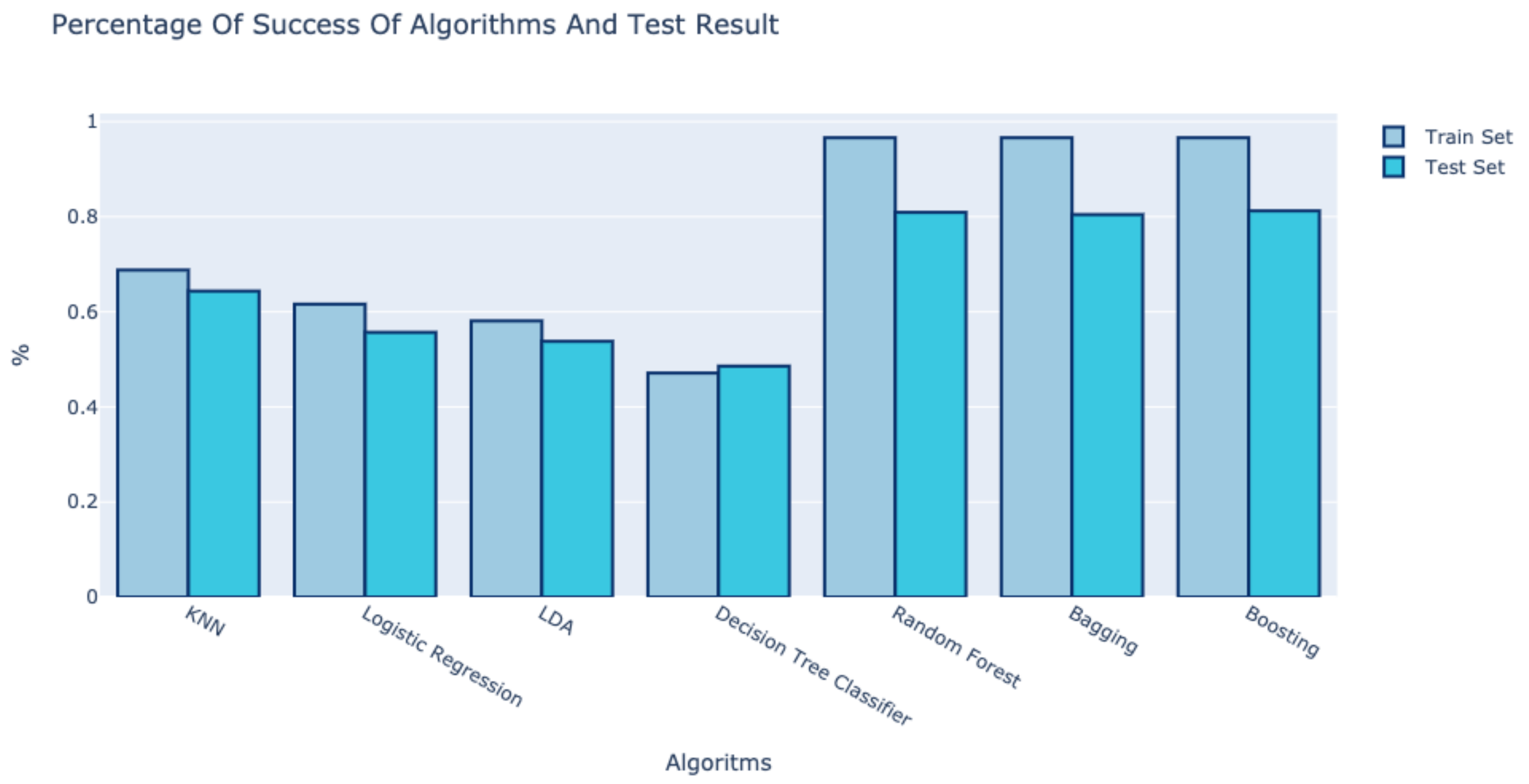
```
]: from sklearn.ensemble import GradientBoostingClassifier

   gboost = GradientBoostingClassifier(n_estimators=300, random_state=50)
   gboost.fit(x_train, y_train)

   y_pred = gboost.predict(x_test)
```

```
Accuracy0.804416403785489
```

Machine Learning

Results

We can see on this graph all the **accuracies scores** of **each** of the **models** on the **train_set** and on the **test_set.** We have decided to keep the **RandomForest** model which offers very **good results.**



Machine Learning

Features Selection

Before finalizing the model to be used on our API, we wanted to **simplify** it by **removing** the very **unimportant** variables in the role of predicting the level of obesity. So here are **the least important** values on the graph.

Finally we **keep** these variables:

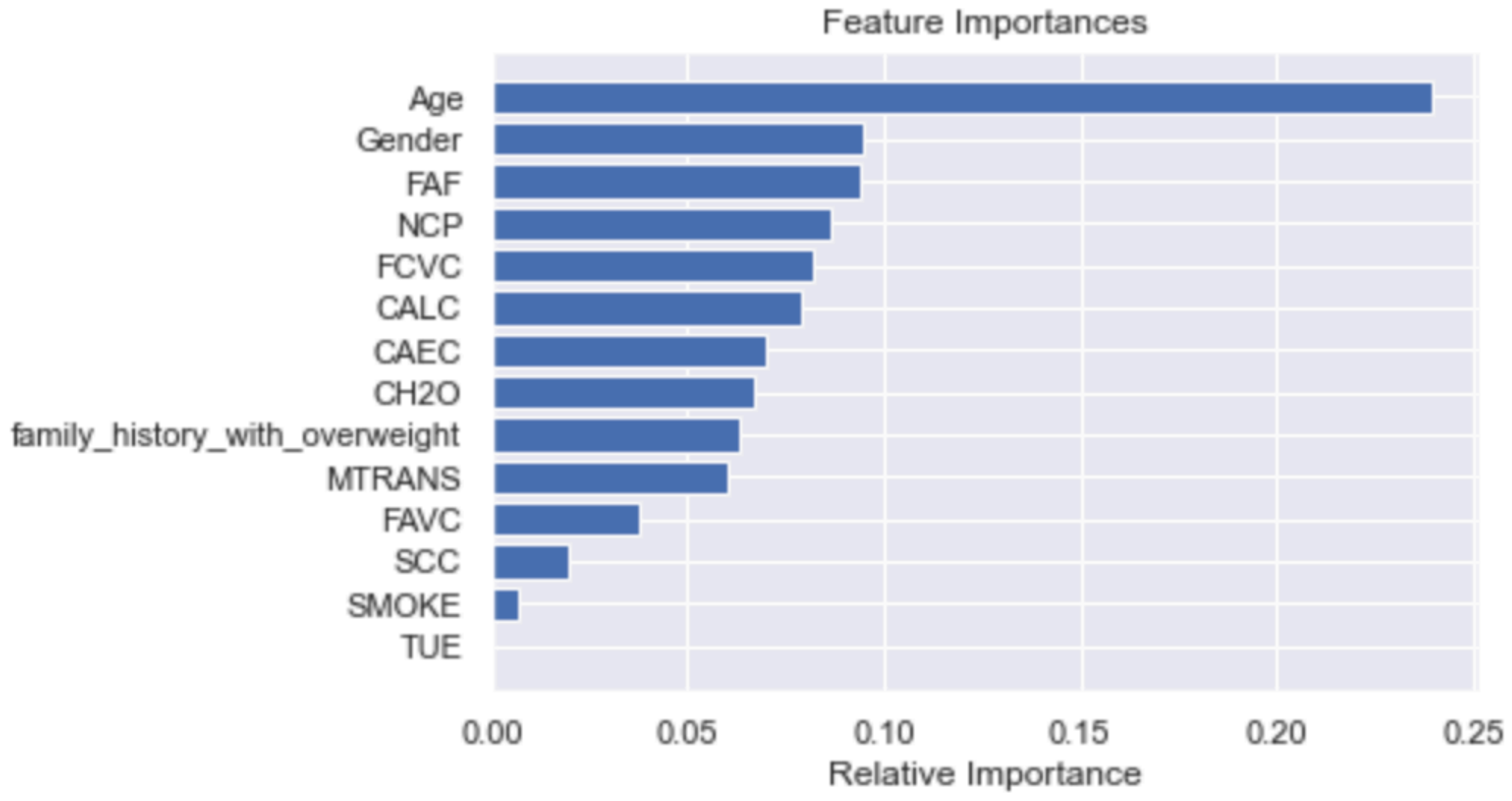
- Gender
- Age
- Family_history_with_overweight
- FCVC
- NCP
- CAEC
- CH2O
- FAF
- CALC

The **accuracy drops by 2%** but the model is simpler.

Features selection

We have selected Random Forest algorithm because it is the best here.

```
]: indices = RF.feature_importances_.argsort()
   plt.title('Feature Importances')
   plt.barh(range(len(indices)), RF.feature_importances_[indices], color='b', align='center')
   plt.yticks(range(len(indices)), [x.columns[i] for i in indices])
   plt.xlabel('Relative Importance')
   plt.show()
```



We can keep only keep features with high importance (here we keep feature importance >0.05) to simplify our model:

API

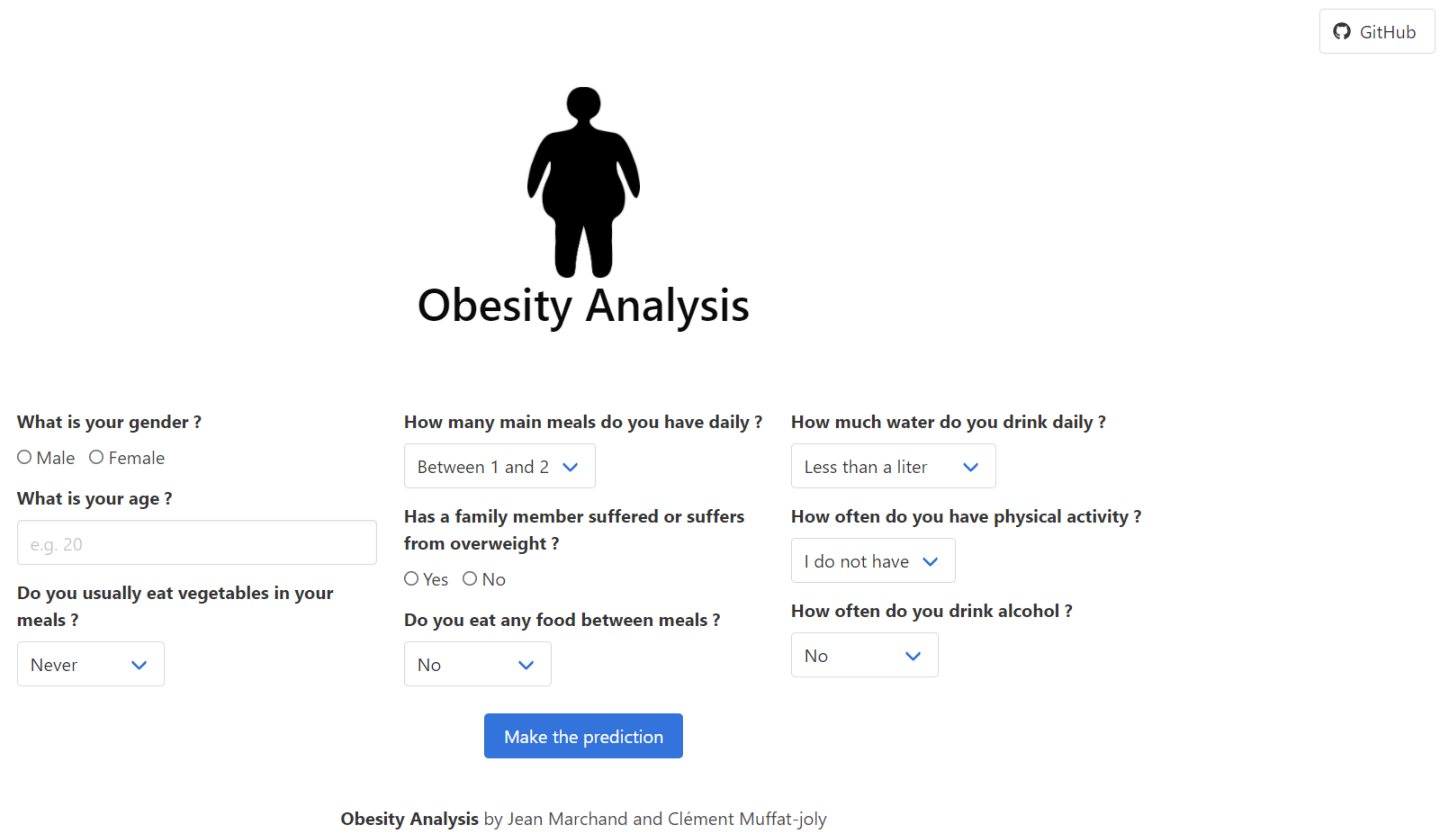
Framework used

To develop our **API** with python, we had **two choices**: **Django** or **Flask**. We chose **Flask** because it is **lightweight** than Django. And we didn't need complicated structures to our API, user just need to make a **single POST request** to obtain the prediction.

API

Website

Moreover, the **Flask** project is composed in **two parts**. One is the **API** that **any program** can **call**. And the second one is the **website** where **any user** can **interact** with to obtain a **prediction**. To predict the obesity level, we used the **model's dump** obtained by the **notebook**. Also, we used **Bulma** for the **CSS** of our website and **Pickle** to use the **model's dump** for the **website** and the **API**.



BULMA