

Dataset : Audio features of songs ranging from 1922 to 2011

INTRODUCTION : This dataset is a subset of the Million Song Dataset (MSD). It contains audio features of songs with their release year. Based on the audio features, we want to be able to predict the release year of the songs (this is our target variable). First, we are going to study the different variables of our dataset, then we will look for correlations and links between them by using data-visualization. Then, we will try different prediction models and compare the results in order to be able to choose the best model.

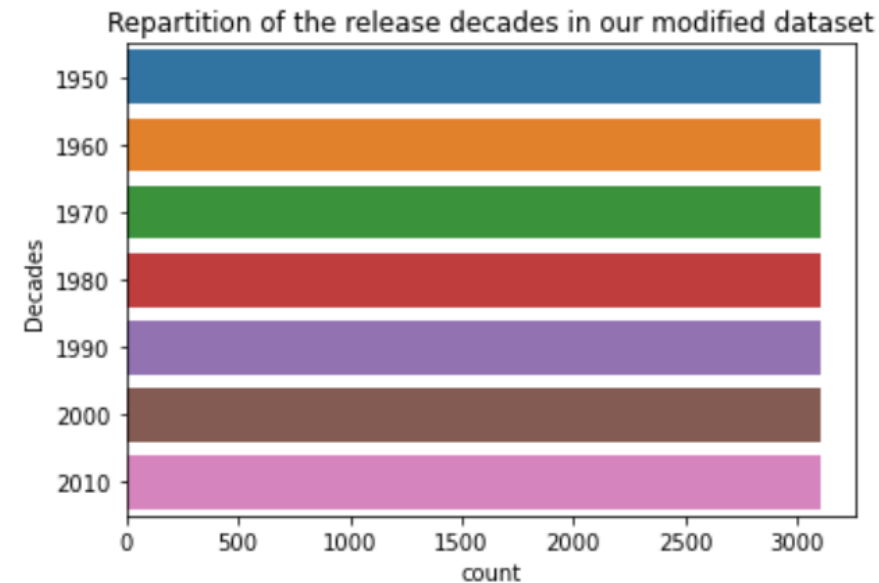
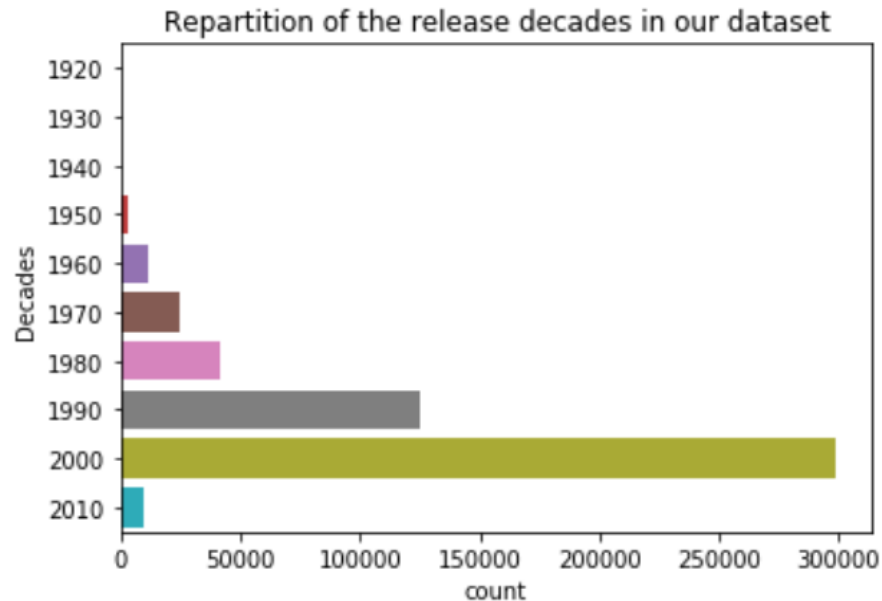
DESCRIPTION OF THE DATASET : We have 515345 rows and 91 variables (1 target value + 90 attributes)

- The first column is the target value : **Year** (of release of the song) - between 1922 and 2011
- The 90 attributes can be of 2 types :
 - **Timbre_Average**
There are 12 Timbre_Average. A timbre refers to the set of sound characteristics that identify an instrument. In our dataset, we only have the average of each timbre for our songs.
 - **Timbre_Covariance**
There are 78 Timbre_Covariance. A Timbre_Covariance is the value of a covariance between 2 Timbre_Average. As we have 12 Timbre_Average, we should have $12 \times 12 = 144$ Timbre_Covariance.
But as $\text{Cov}(\text{Timbre_Average1}, \text{Timbre_Average2}) = \text{Cov}(\text{Timbre_Average2}, \text{Timbre_Average1})$, we can take the half of these informations so we have $144/2 = 78$ Timbre_Covariance.

Data visualizations (1/4)

The intervals in which we can find songs is really big (1922-2011). We decided to add another attribute : Decade, which represents the release decade of the song. For example, the decade 1950 contains the songs which were released between 1950 and 1959.

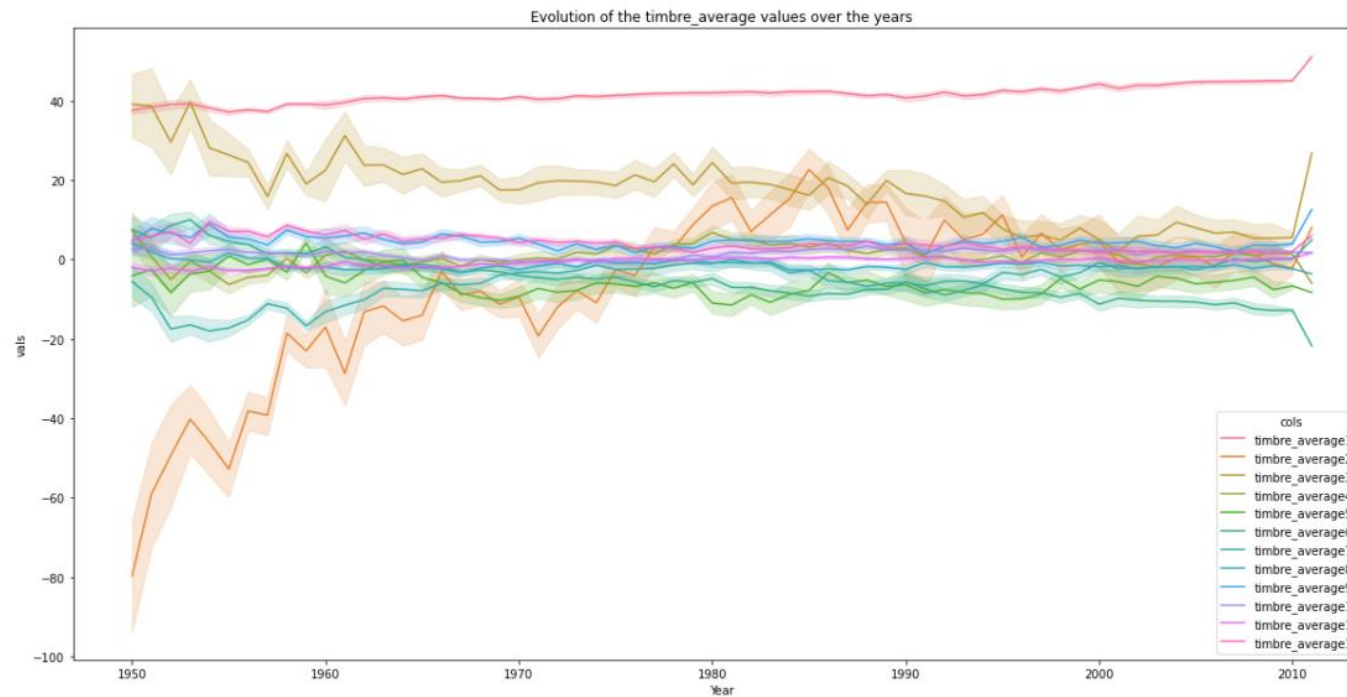
If we look at the amount of values we have in each decade, we can see that they vary a lot. We have less than 3000 values in decades 1920, 1930 and 1940 whereas we have almost 300000 values for decade 2000. So we decided to take a random sample of 3100 songs for each decade (but we removed decades 1920, 1930 and 1940), for our data visualizations.



Data visualizations (2/4)

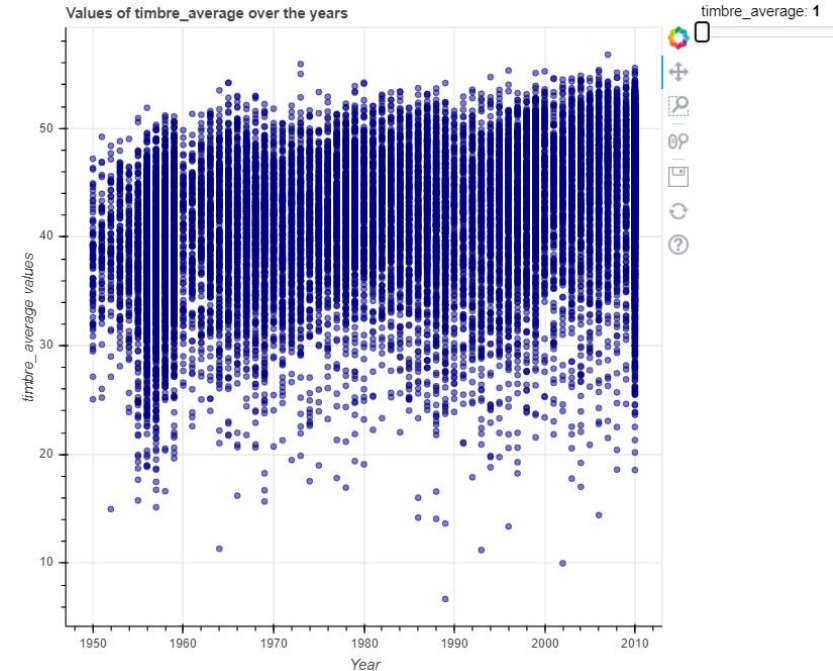
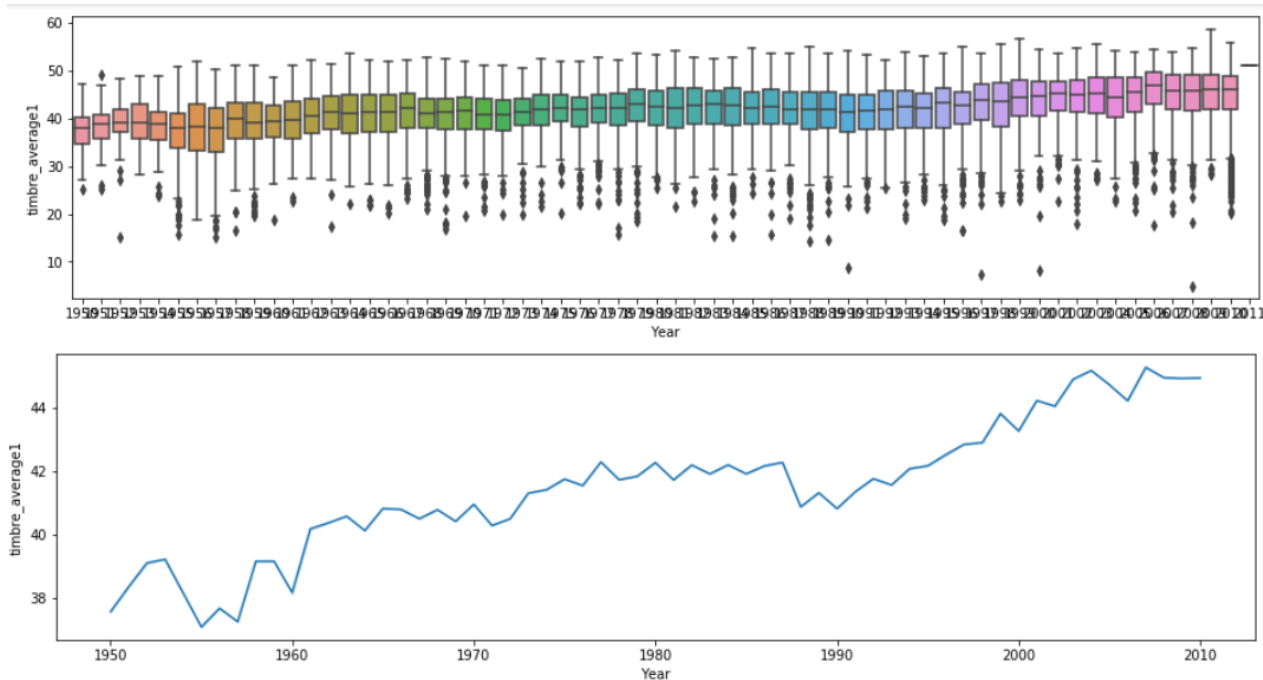
We use our modified dataset for the visualisations so we study here 21700 rows.

This plot shows that each `timbre_average` have really different variation intervals. For example, `timbre_average1` has values around 40 every year, whereas the values of `timbre_average12` vary a lot (between -80 and 0). This plot only shows



Data visualizations (3/4)

We wanted to study each `timbre_average` separately so we made different plots for each `timbre_average` in function of the year. For example, if we only look at `timbre_average1` :



We can see that `timbre_average` is linked with the year: it increases with the year. So it can help to predict the release year of the songs.

The plots for the other `timbre_average` are presented in our Jupyter Notebook.

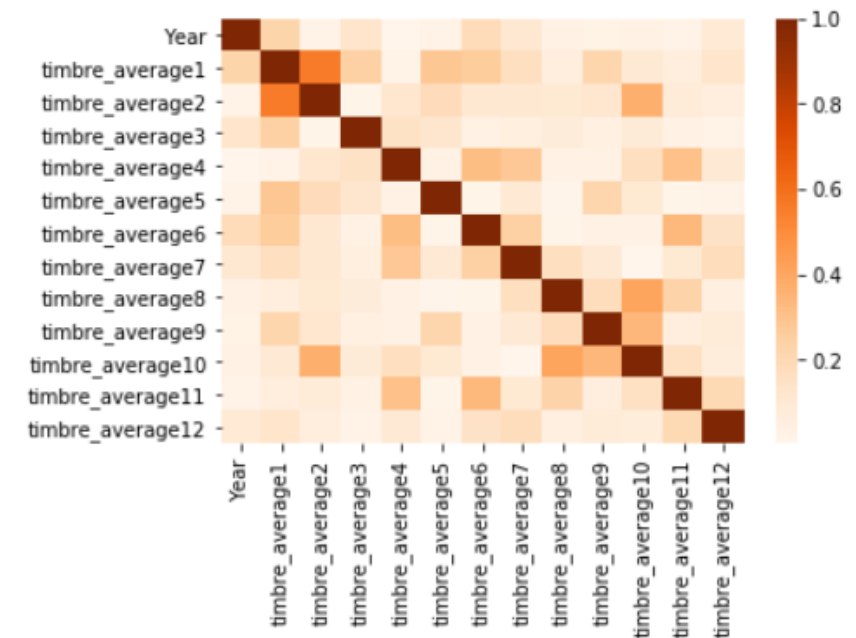
Data visualizations (4/4)

Conclusion :

Our target value, the year, is linked with all the timbre_average but the correlation is not high. We can see it in our correlation matrix.

So finally, we will use all the timbre_average attributes in our prediction models: from timbre_average1 to timbre_average12.

We are not considering the timbre_covariance values of our dataset as they only give further information about links between 2 timbre_average and not information about links between timbre_average and release year (so they can't help for prediction).



Modelisation (1/3)

Our problem is a **Classification** problem. The different categories are the different years between 1922 and 2011, which represent a lot of categories ! That's why we decided to transform our problem in : "**Predict release decade of songs**". Thanks to this transformation, we only have 10 categories : 1920 (1920 to 1929), 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000 and 2010.

We use all the rows of our dataset (515345).

After splitting our dataset into training and testing sets, we tried a lot of different Classification models and computed the accuracy.

| Classifier | Accuracy |
|---|---|
| Decision Tree Classifier We obtained the best accuracy with max_depth=10 : <i>59.48% on the test set</i> | Accuracy Score on train data: 0.6082230741723286 Accuracy Score on test data: 0.5947840935551474 |
| Random Forest Classifier We obtained the best accuracy with max_depth=15 : <i>60.35% on the test set</i> | Accuracy Score on df_train: 0.6839255865011186 Accuracy Score on df_test: 0.603548420480712 |

Modelisation (2/3)

| Classifier | Accuracy |
|---|---|
| Logistic Regression We obtained an accuracy of 58.6% on the test set. We didn't vary any parameters. | <pre>Accuracy Score on df_train: 0.5867506050046987 Accuracy Score on df_test: 0.5860585754572973</pre> |
| Gaussian Naïve Bayes We obtained an accuracy of 55.44% on the test set. We didn't vary any parameters. | <pre>Accuracy Score on train data: 0.5574997020022675 Accuracy Score on test data: 0.5544552534216449</pre> |
| Gradient Boosting We obtained an accuracy of 47.81% on the test set. We didn't vary any parameters. | <pre>Accuracy Score on df_train: 0.6073692760179741 Accuracy Score on df_test: 0.47811828930687433</pre> |
| Support Vector Classifier We could only try this model on a smaller dataset (the one we used for our data visualizations) because it took too much time. The accuracy is small as expected (36.6%) but could have been way higher if we would have used the entire dataset. | <pre>Accuracy Score on df_train_small: 0.4057541641977747 Accuracy Score on df_test_small: 0.36661035171248657</pre> |

Modelisation (3/3)

Our best model was finally : **Random Forest Classifier**.

To optimize it at most, we decided to make a **GridSearch** and vary several parameters in order to try several combinations.

We decided to vary : - **max_depth** (between 14 and 16)

- **criterion** (between « gini » and « entropy »)

- **min_sample_leafs** (between 1 and 3).

We didn't make a K-fold cross validation because our dataset is really big so we don't need this.

We finally obtained : `RandomForestClassifier(max_depth=15, min_samples_leaf=2)`

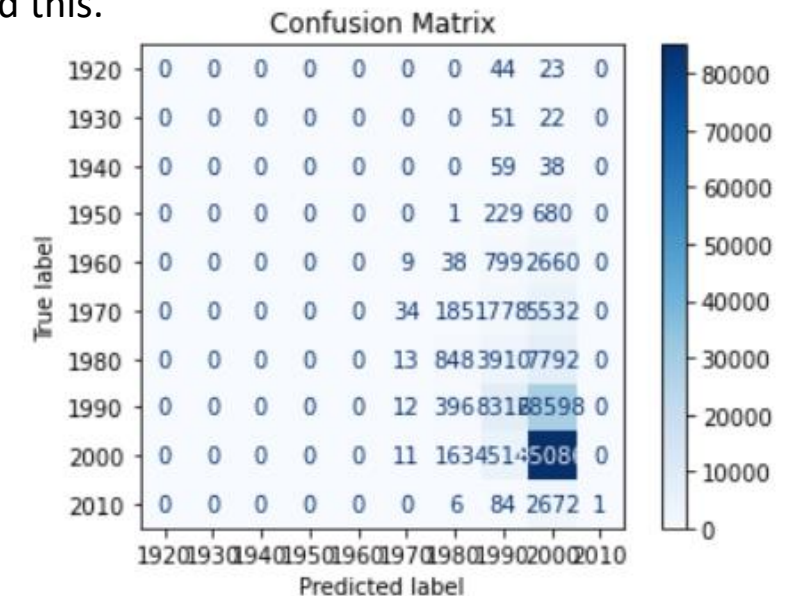
This didn't gave us indications about the parameter criterion, so we kept the default parameter of criterion.

We finally fitted our best model, the one provided by our GridSearch :

Accuracy Score on df_train: 0.6811119334924502

Accuracy Score on df_test: 0.6098483868463946

Final accuracy : 60.98%

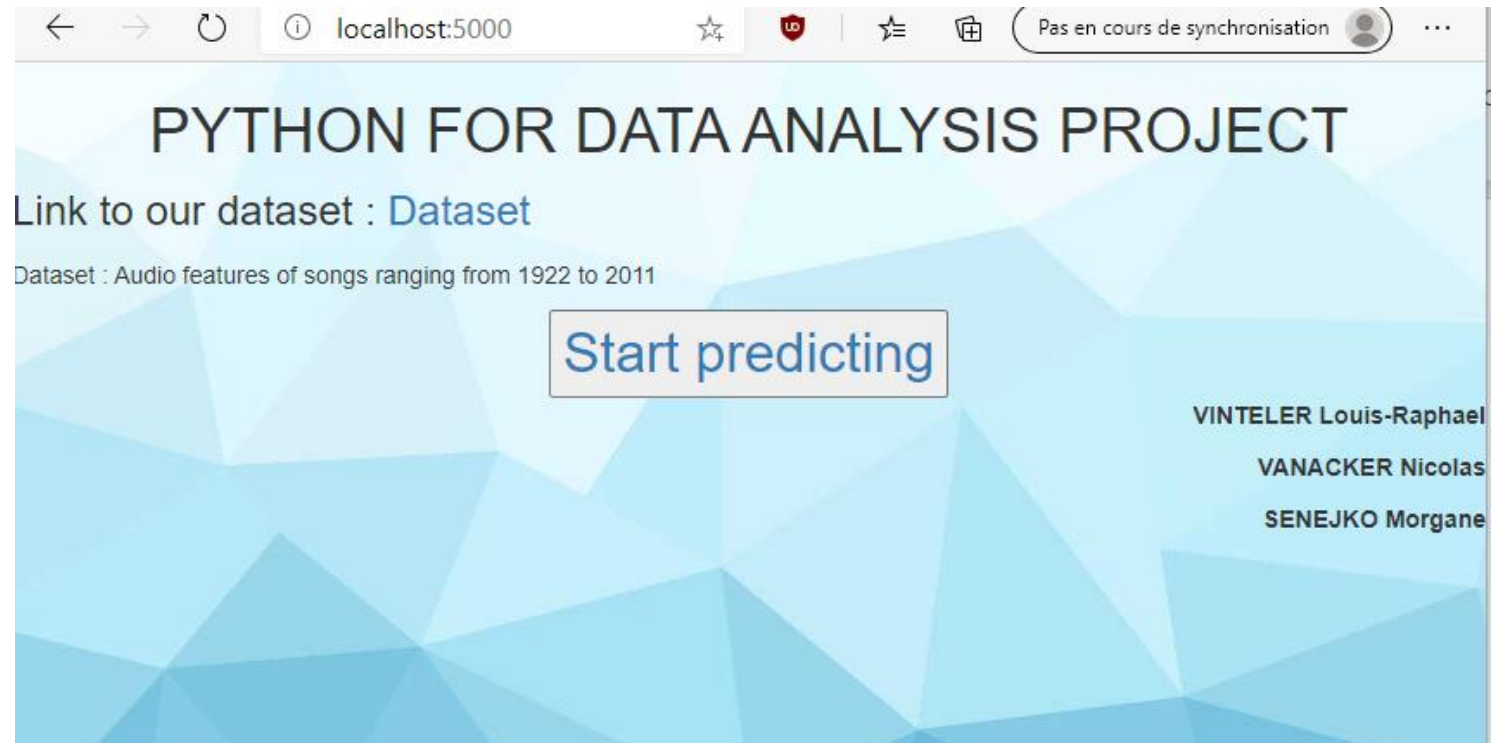


API (1/3)

To visualize our model, we chose to develop the API on flask. The API is developed on a new conda environment. The API takes a bit longer to start because we load our .txt which contains the initial dataset. We do that because on the prediction part, we chose a random observation from our dataset (the .txt is only used for this, not on prediction).

The API contains 3 html pages : home, predict and results. The API files contains also a requirements.txt to get all the packages for our new environment and a joblib file four the model.

On the home page, there are a link to our dataset and a brief presentation, and a button to start the prediction.



API (2/3)

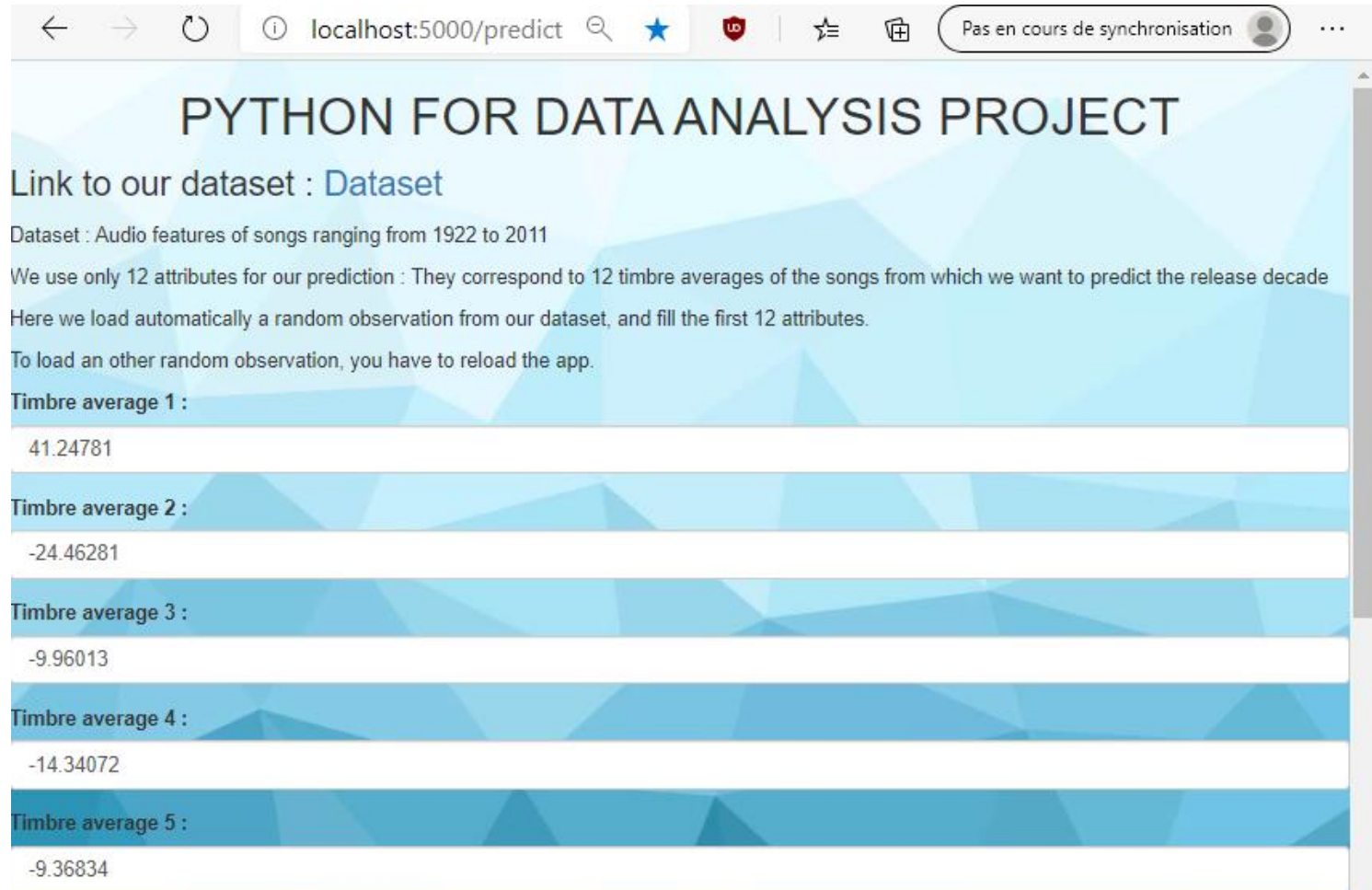
On the predict page, we ask the user to enter 12 values : timbre average 1, timbre average 2 ... until timbre average 12.

These values are filled automatically by a random observation from our dataset because it's not easy to invent that many values (which have to be coherent). They now corresponds to the values of a real song.

But we can also enter them manually because we put limitations: the values can only be between max and min values of each features.

After entering the 12 values, you can click on the button "submit" and the API will load the model from joblib and then predict a result.

To fill the fields with another random observation, you have to reload the API.



The screenshot shows a web browser at localhost:5000/predict. The page has a blue geometric background and contains the following text:

- PYTHON FOR DATA ANALYSIS PROJECT**
- Link to our dataset : [Dataset](#)
- Dataset : Audio features of songs ranging from 1922 to 2011
- We use only 12 attributes for our prediction : They correspond to 12 timbre averages of the songs from which we want to predict the release decade
- Here we load automatically a random observation from our dataset, and fill the first 12 attributes.
- To load an other random observation, you have to reload the app.

Below the text, there are five input fields, each with a label and a value:

| Label | Value |
|--------------------|-----------|
| Timbre average 1 : | 41.24781 |
| Timbre average 2 : | -24.46281 |
| Timbre average 3 : | -9.96013 |
| Timbre average 4 : | -14.34072 |
| Timbre average 5 : | -9.36834 |

API (3/3)

On the results page, we display the result from our prediction. It corresponds to the release decade of the songs for which we entered the different values of timbre average.

When the different timbre average corresponds to a random observation of our dataset, we can also display the real release decade of the song and even the release year of the song.

So we can compare the prediction with the real value.

A green text is displayed if the model predicted right or a red text if it didn't.

In our example : the model predicted the good decade 2000.

