

# YearPredictionMSD Data Set

Python for data analysis

**Théophile PUISEUX & Nieves RODRIGUEZ ALVAR**

# Objectives

## Prediction of the release year of a song from audio features

The Million Song Dataset (MSD) is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. The purpose being to predict the release year of a song from audio features.

Songs in the dataset are mostly western, commercial tracks ranging from 1922 to 2011, with a peak in the year 2000s.

Due to the large spectrum of possible values (89 years) and the imbalanced data, we decide that it was better to predict the decade of the release year.

# Three main steps

From understanding to implementation

## Data Analysis



Inspecting, cleaning, and modeling data to attain informing conclusions, and supporting decision-making.

## Models Selection



Design experiments to select a statistical model from a set of candidate models.

## Deployment



Integrate the selected model into an existing production and stable environment where a client request the model

# Data Analysis

Python for data analysis

# Dataset description

## 90 features...

Before starting to code, we wanted to fully understand the dataset we were given and its purpose.

There are 90 attributes :

- TimbreAverage[1-12]
- TimbreCovariance[1-78]

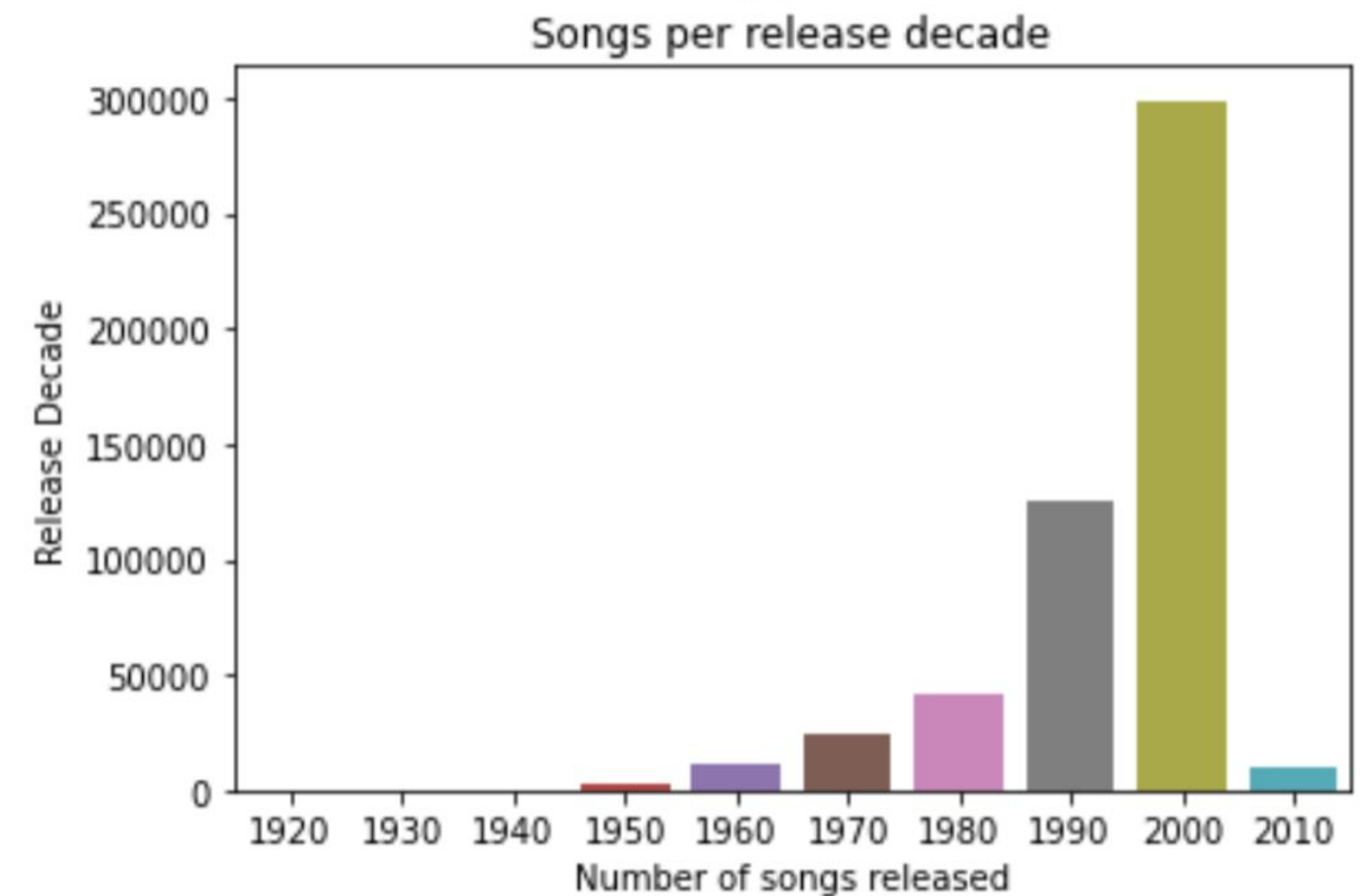
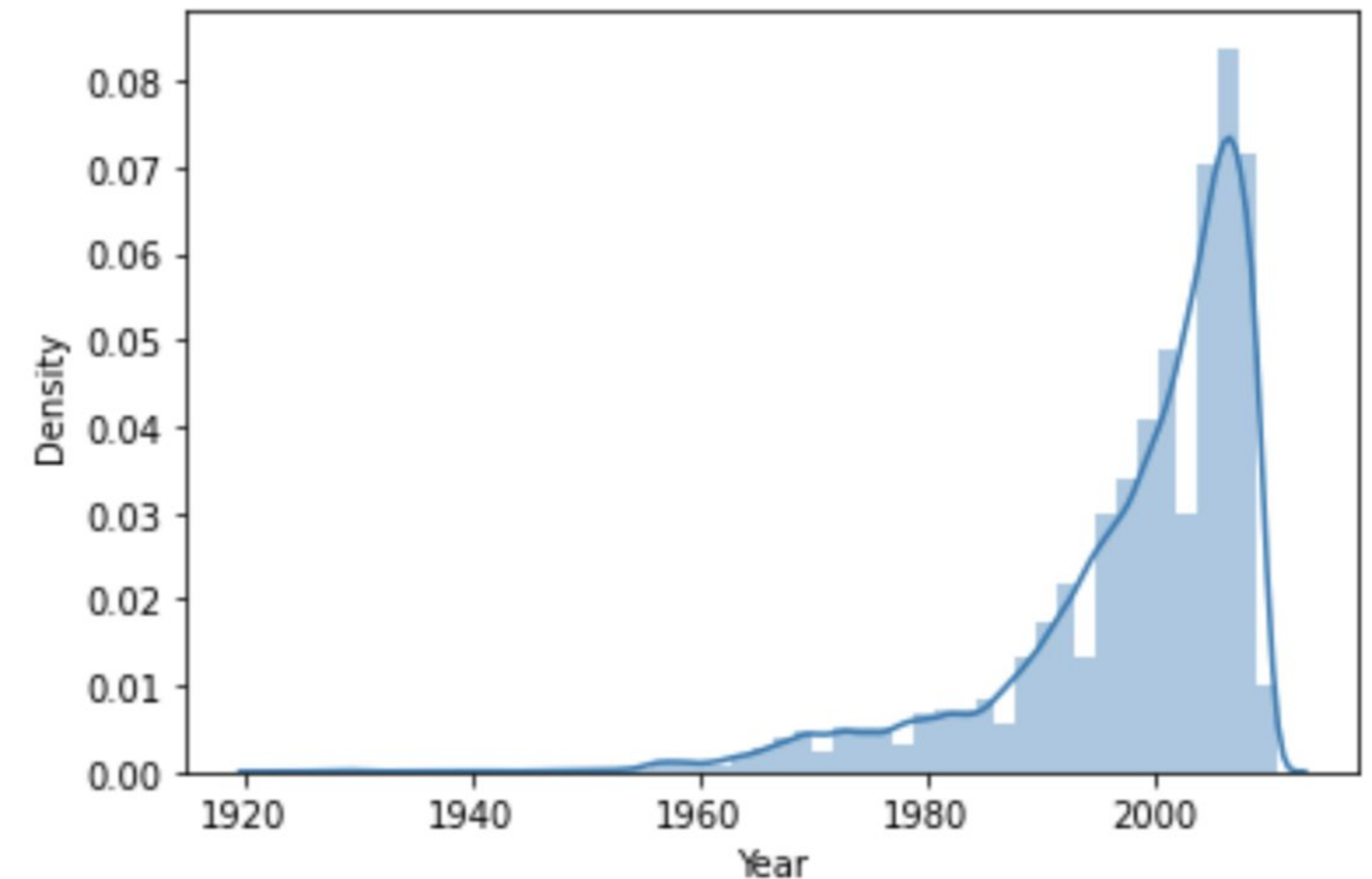
These features were extracted from the 'timbre' features from The Echo Nest API. The authors took the average and covariance over all 'segments' and each segment was described by a 12-dimensional timbre vector.

# Year Distribution

## Target

As explained in the dataset description, we can clearly see that there is a peak in the 2000s.

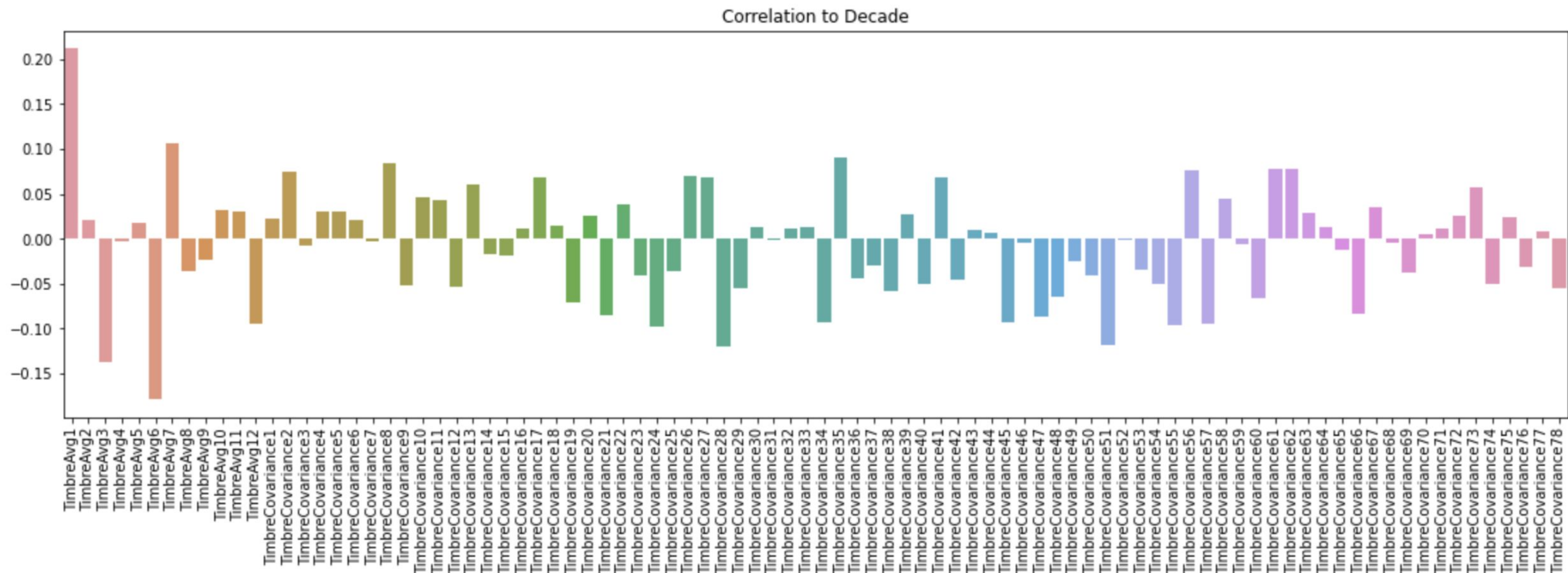
In order to prevent any bias in our future models, we decided to downsample our train set, to have equal data for each decade.





# Correlation analysis

Decade

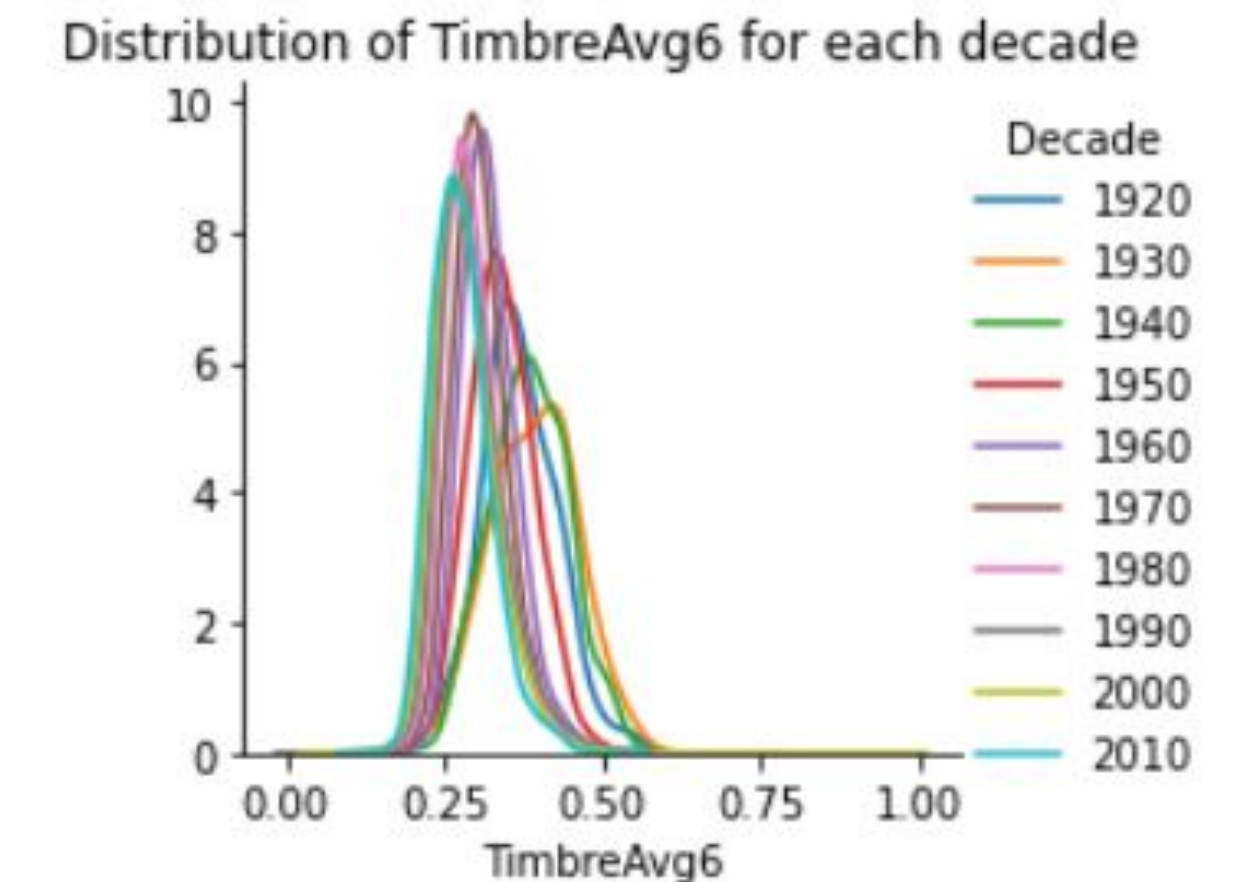
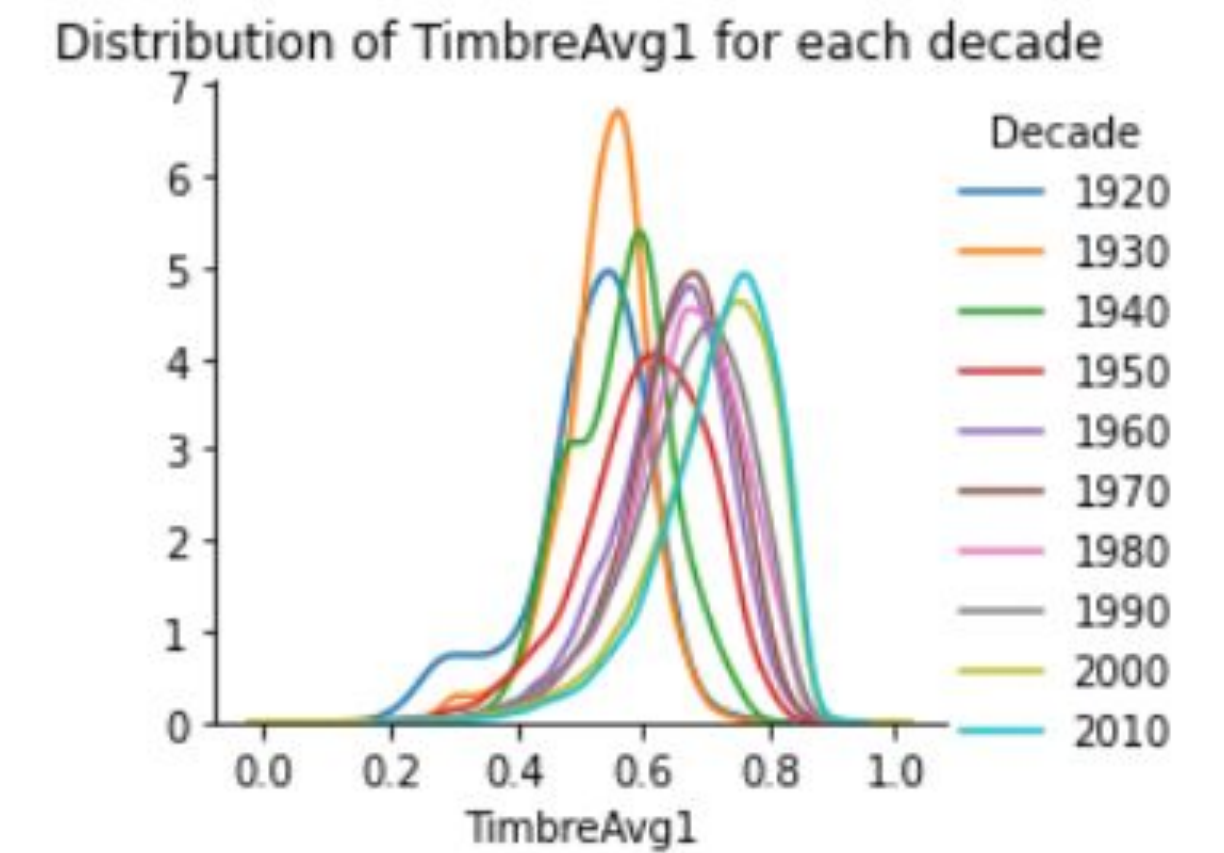
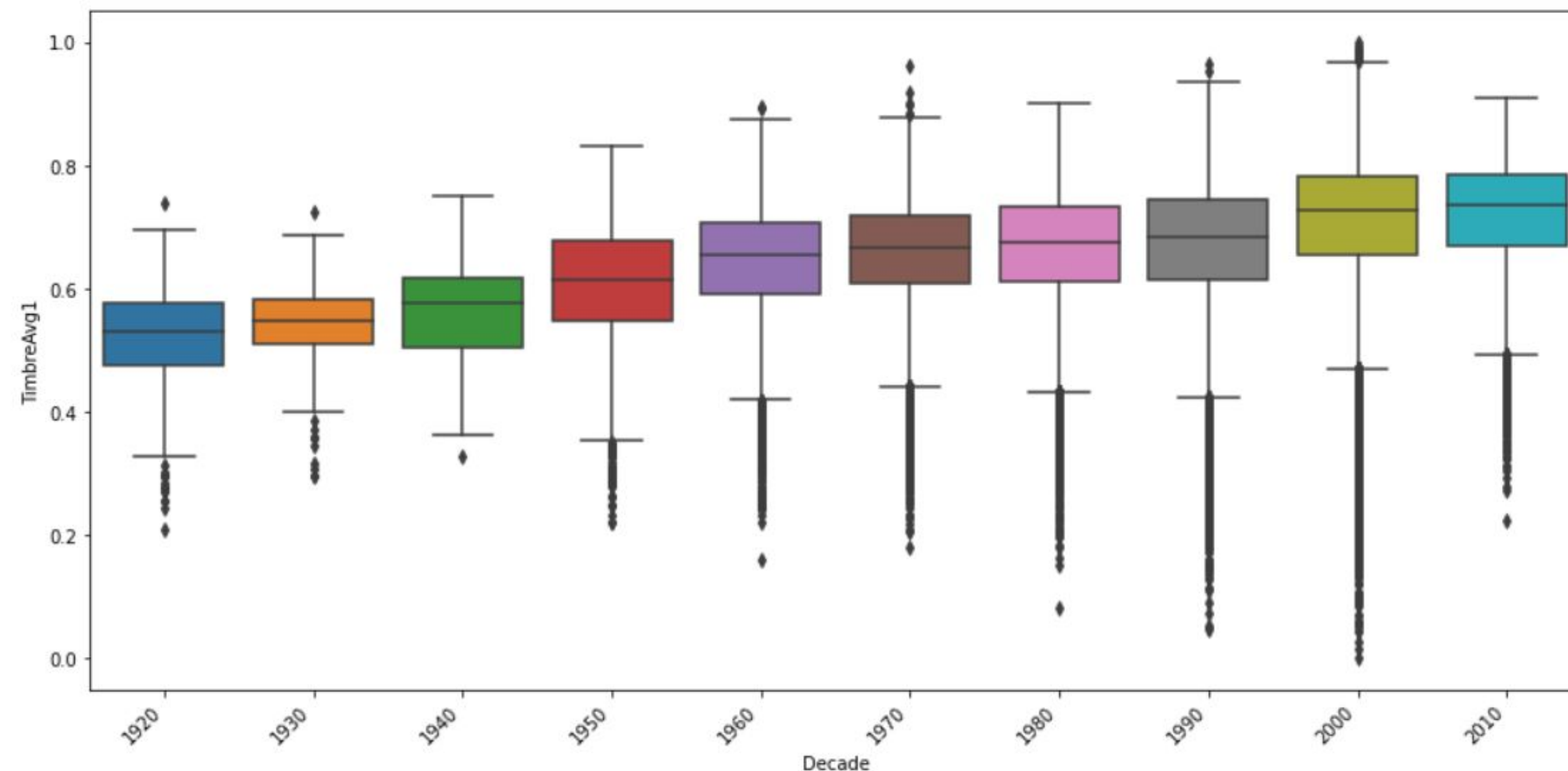


In this bar graph, we can see our correlation results for ‘Decade’ of the independent variables.

We can see that there isn’t a feature that really stands out from the others.

# Distribution for each feature

## By decade



Due to the large amount of feature in this dataset, we couldn't find a link between the variables and the target. The distribution of features by decade is uniform.



# Data preparation

## What we used

### MinMax Scaling

Normalize the data and bring every feature in the same footing without any upfront importance.

### PCA

Reduce the dimension of our dataset. From 90 feature we reduced it to only 20.

### Downsampling

To train our models on all decades equally and therefore deal with the unbalanced dataset.

# Models Selection

Python for data analysis

# Hyperparameters Tuning

## Method we used

To tune our models and find the best hyperparameters, we either used the GridSearchCV method or a manual GridSearch with for loops.

```
grid_search = GridSearchCV(svm.SVC(),
                            {'kernel':['linear', 'rbf', 'poly'],
                             'C': [1, 5, 10, 15, 20, 25],
                             'gamma' : [1, 5, 10, 15, 20]
                            },
                            cv=None)

grid_search.fit(X_train, y_train)
clf = grid_search.best_estimator_
print(clf)
```

```
depths = [5, 10, 15, 20]
n_est=[10, 100, 200]
for d in depths:
    for n in n_est:
        model = RandomForestClassifier(n_estimators=n, max_depth=d, random_state=0).fit(X_train, y_train)
        score = model.score(X_test, y_test)
        pred_y = model.predict(X_test)
        print("N_estimators:{0:.0f}, Depth:{1:.0f}, Score:{2:.4f}"
              .format(n, d, score))
```

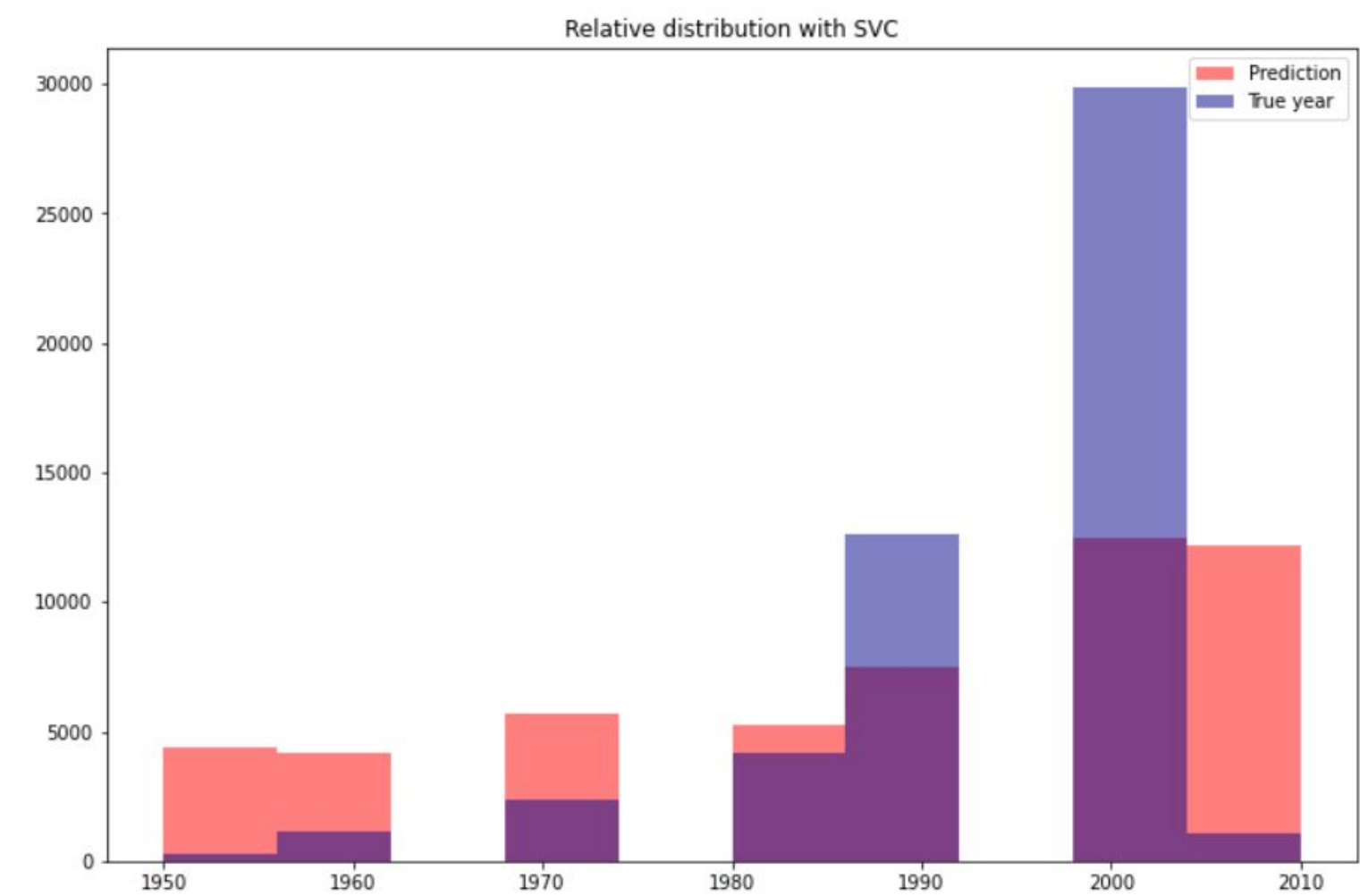
# Classification Models

## Models we used

To answer our problem we used the following classification algorithms :

- Decision Tree Classifier
- SVC
- Random Forest Classifier
- Ridge Classifier

Models	Accuracy
Decision Tree Classifier	23.6%
SVC	31.5%
Random Forest Classifier	28.4%
Ridge Classifier	18.3%

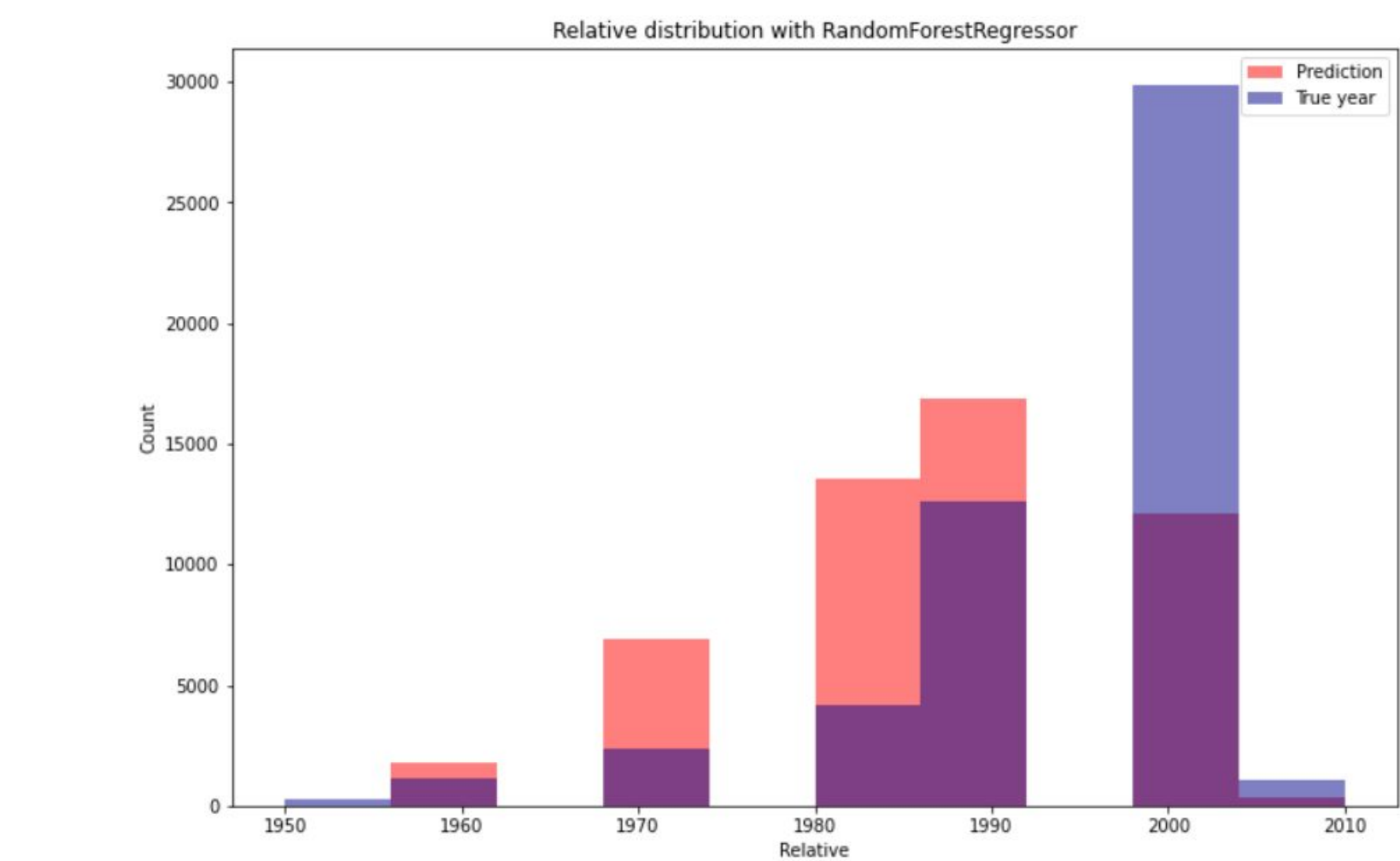


# Regression Models

## Models we used

To answer our problem we used the following regression algorithms :

- Random Forest Regressor
- Linear Regression
- Lasso Regularization
- ElasticNet Regularization



Models	RMSE
Random Forest Regressor	13.91
Linear Regression	15.16
Lasso	14.79
ElasticNet	14.70



# Model Selection

## Decision-making factor

A RMSE around 15 means that our model is predicting correctly within a 15 years range around the prediction.

For our API, we thus decided to select the :

- SVC (Highest accuracy)
- Random Forest Classifier (Second best accuracy)
- Random Forest Regressor (Lowest RMSE)

# Deployment of the API

Python for data analysis

# Transformation of the model into an API

What does it consist of ?

## Front-end : Bootstrap



Bootstrap is a collection of useful tools for creating web design and web applications. It is a set that contains HTML and CSS codes, forms, buttons, navigation tools and other interactive elements, as well as optional JavaScript extensions.

## Back-end : Flask



Flask is an open-source micro framework for web development in Python. It is classified as micro framework because it is very light. Flask aims to keep the kernel simple but extensible.

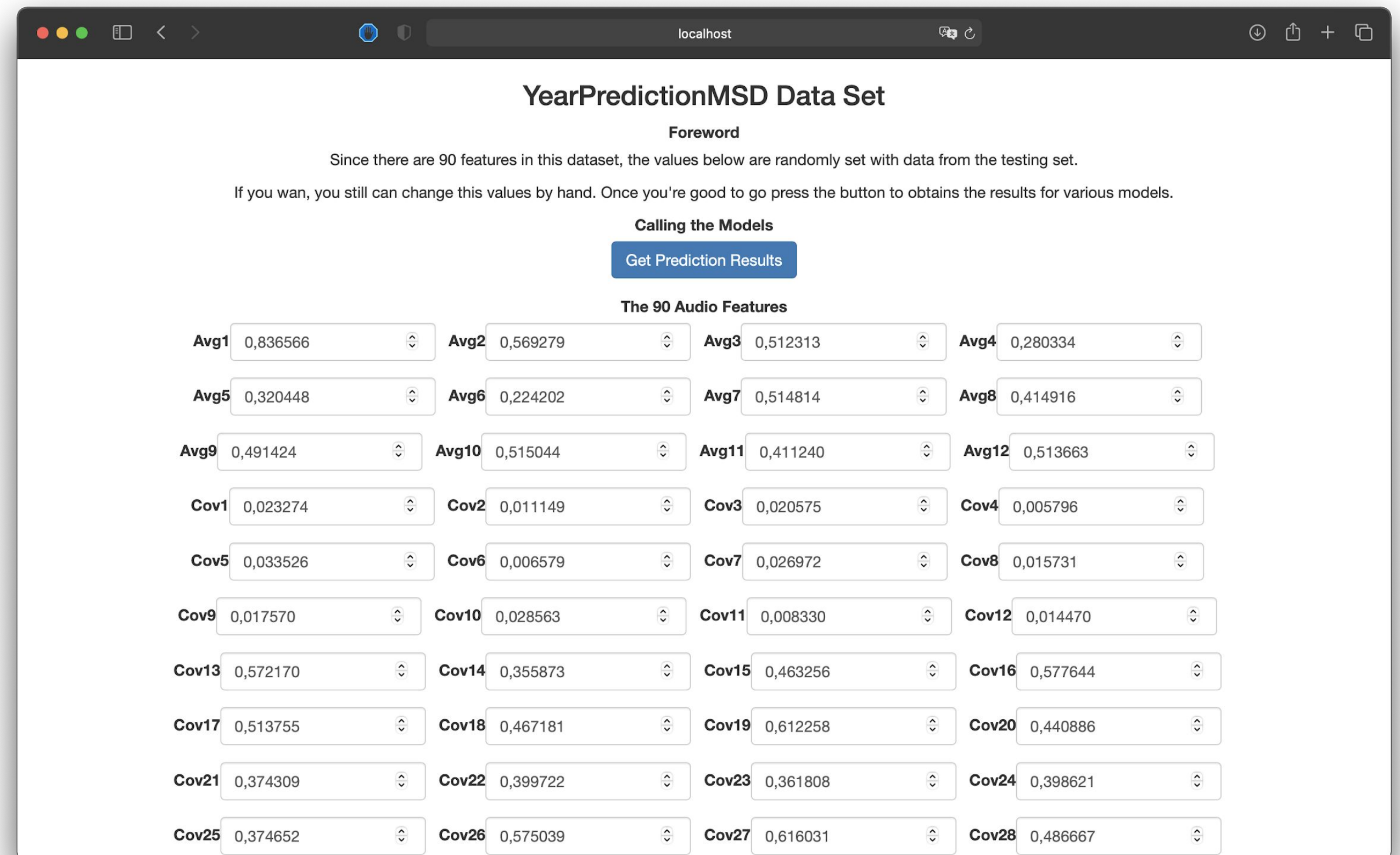
# Flask API

## Easy to use

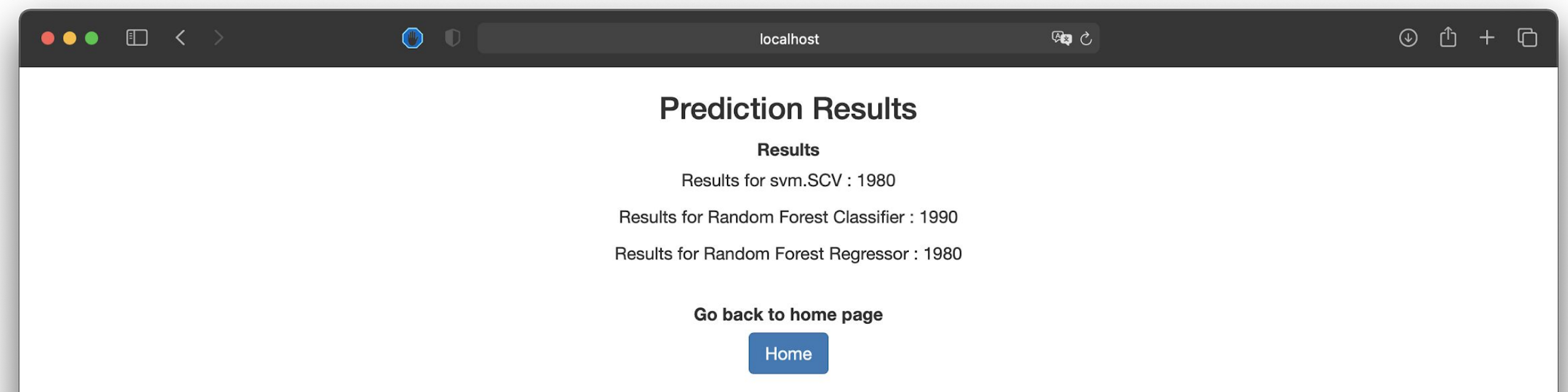
To make it easy to use for any user, we already placed a value in the 90 feature inputs. The user can access it and modify any value.

These values are randomly selected from the test set.

Once the user is satisfied with the values, he presses the button and the API goes on to the next page with the results



The screenshot shows a web browser window with the URL 'localhost'. The page title is 'YearPredictionMSD Data Set'. Below the title is a 'Foreword' section explaining that 90 features are randomly set from the testing set and that users can manually change these values. A 'Calling the Models' section contains a 'Get Prediction Results' button. The main part of the page is titled 'The 90 Audio Features' and displays a grid of 90 input fields, each with a label (Avg1 through Cov28) and a numerical value. The values are: Avg1: 0,836566, Avg2: 0,569279, Avg3: 0,512313, Avg4: 0,280334, Avg5: 0,320448, Avg6: 0,224202, Avg7: 0,514814, Avg8: 0,414916, Avg9: 0,491424, Avg10: 0,515044, Avg11: 0,411240, Avg12: 0,513663, Cov1: 0,023274, Cov2: 0,011149, Cov3: 0,020575, Cov4: 0,005796, Cov5: 0,033526, Cov6: 0,006579, Cov7: 0,026972, Cov8: 0,015731, Cov9: 0,017570, Cov10: 0,028563, Cov11: 0,008330, Cov12: 0,014470, Cov13: 0,572170, Cov14: 0,355873, Cov15: 0,463256, Cov16: 0,577644, Cov17: 0,513755, Cov18: 0,467181, Cov19: 0,612258, Cov20: 0,440886, Cov21: 0,374309, Cov22: 0,399722, Cov23: 0,361808, Cov24: 0,398621, Cov25: 0,374652, Cov26: 0,575039, Cov27: 0,616031, Cov28: 0,486667.



The screenshot shows a web browser window with the URL 'localhost'. The page title is 'Prediction Results'. Below the title is a 'Results' section showing the following information: 'Results for svm.SCV : 1980', 'Results for Random Forest Classifier : 1990', and 'Results for Random Forest Regressor : 1980'. At the bottom, there is a 'Go back to home page' link and a 'Home' button.

# Test the API

Only three steps

## Clone the project



Download the project  
or clone it directly from  
git.

## Start the flask API



On your terminal  
launch the app with  
the following  
command :  
`python app.py`

## Go to localhost:5000



On your browser go to  
the address :  
`http://localhost:5000`