# PYTHON FOR DATA SCIENCE

## — Predicting cycling traffic —

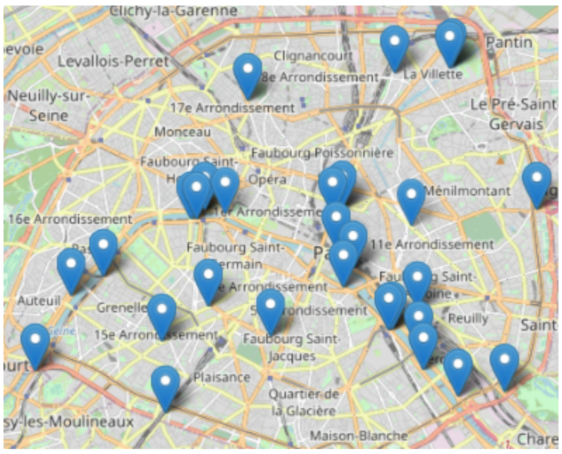
MELLERIO Antoine
VISKADOUROS Evangelos
December 2021

# Introduction

This project aims at predicting the bicycle traffic in some specific locations in Paris.

30 counting stations were installed in strategic bicycling lines of the capital. to count the number of bikes passing by over one hour. It was decided to take the logarithm of the bike counts for repartition concerns.

This information could be very useful for the administration of the city in order to provide adapted facilities to the inhabitant.



*Locations of the 30 counting stations*

In order to build the predictive model, we retrieved measurements of bike count between 01/09/2020 and 09/08/2021, along with information on the concerned stations and measurements.

| counter_id | counter_name | site_id | site_name | date | counter_installation_date | coordinates | counter_technical_id | latitude | longitude |
|---|---|---|---|---|---|---|---|---|---|

To carry this project out, we are first going to look for additional data to complete the ones discussed before. The model is going to be tested on a dataset that ranges from 01/09/2020 to 21/10/2021. Thus we aim at finding data that cover this period.

Then we will deal with filtering the data and preprocessing it.

The final step will be  to find and optimize a model in order to maximize the accuracy of the prediction.

# Data exploration

## I. Finding data

For exploring the data, we cross-validate over 4 folds in order to make comparisons in a timely manner. In order to improve our dataset, we found 3 sources giving useful information on weather (temperature, precipitations, clouds, wind, etc.) and holidays (scholar and public) :

| Dataset | Number of features | Frequency | Source |
|---|---|---|---|
| **Orly weather** | 58 | /hour | Meteo France |
| **Paris weather** | 8 | /day | https://www.visualcrossing.com/weather/weather-data-services#/login |
| **Holidays calendar** | 4 | /hour | https://www.data.gouv.fr/en/ |

The Paris weather is geographically more accurate but less precise in frequency of the measurements.

## II. Filtering the data

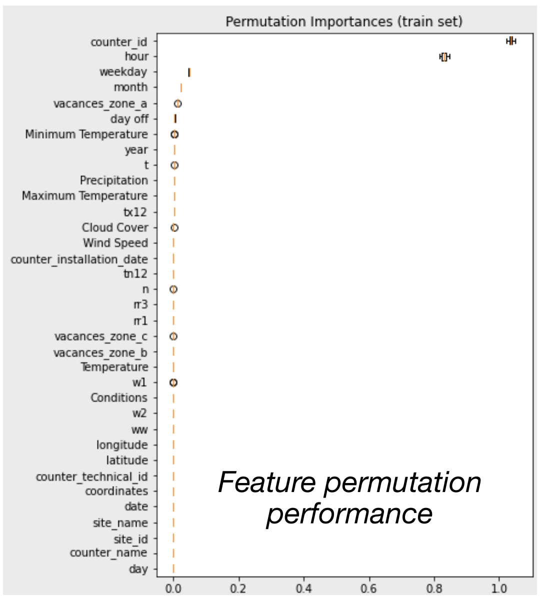
We use the whole dataset in a pipeline with HistGradientBoosting regressor to classify feature per permutation importance. We perform this rankingwith the permutation_importance module of sklearn.inspection.

Beforehand, we split the 'date' feature into 'hour', 'day', 'weekday', 'month', and 'year'.

We then decide to remove the features having 0 importance, selecting the following:

- hour
- weekday
- month
- year
- tn12
- tx12
- n
- rr3
- rr1
- w1
- w2
- t
- day off

- counter_id
- counter_installation_date
- Maximum Temperature
- Minimum Temperature
- Temperature
- Conditions
- Cloud Cover
- Wind Speed
- Precipitation
- vacances_zone_a
- vacances_zone_b
- vacances_zone_c



*Feature permutation performance*

## III. <u>Preprocessing the data</u>

<u>Missing values:</u> the features w1, w2, n, rr1, rr3, tn12 and tx12 have missing values. As their values are continuous, we replace these missing values **interpolating** linearly when datas are missing one by one, or polynomialy of order 2 when several datas are missing in a row.

Moreover, the precipitation feature 'rr1' has negative values -0.1 that are an error and hence replaces by 0.

<u>Categorical values:</u>

- vacances_zone_a, vacances_zone_b, vacances_zone_c and 'day off' are already binary values (0 or 1) hence **we do not preprocess them**.
- "counter_id", "counter_installation_date" are not-ordered thus we use a **one-hot encoder**
- "Conditions" is ordered but we cannot use OrdinalEncoder because some categories are missing in the training set (got "ValueError: Found unknown categories ['Rain, Overcast'] in column 0 during transform"). Hence we apply as well a **one-hot encoder**.
- "counter_installation_date": we decompose it in year and month to use them as numerical and so that the model expands better to predictions for a new station. We tried their influence out:
  - only year as numerical : RMSE: 0.935 ± 0.193 ; Time : 97.99 s.
  - year + month as numerical : RMSE: 0.925 ± 0.198 ; Time : 107.15 s.
  - whole date as categorical (OHEncoder) : RMSE: 0.896 ± 0.226 ; Time : 158.34 s.

**—> we keep the feature as categorical and apply a one-hot encoder.**

<u>Numerical variables:</u>

- 'Temperature', 'Maximum Temperature', 'Minimum Temperature', 'Precipitation', 'Wind Speed', 'Cloud Cover', 't', 'tn12', 'tx12', 'w1', 'w2', 'n', 'rr1', 'rr3' are all numerical and we just apply a **Standard Scaler** to them.

<u>Try out:</u> 1 - apply periodic spline or cos/sin transformations to the date variables in order to emphasis on correlations

- with cyclic month, weekday, hour : RMSE: 0.904 ± 0.224/nTime : 105.91 s.
- without : RMSE: 0.901 ± 0.225/nTime : 102.29 s.
- with sin/cos: RMSE: 0.870 ± 0.223/nTime : 104.24 s.

**—> we encode the data with the sin/cos transformations.**

2 - add a column per covid-lockdown period indicating by a 1 the dates of lockdown. The accuracy decreased by about 0.015 using these new datas, hence they are dropped.

# Model optimization

## I. Exploring different models

For the model exploration and optimisation process we will cross-validate our models with 8 folds equal to the different number of years available in the dataset.
In the beginning, we start with the given simple Linear Regression model, which uses OneHotEncoding for both date and categorical columns.
We test this model with cross validation with 8 folds and the result is slightly better with the additional data, with a mean RMSE = 0.874 over the 8 folds. For this model we are using one hot encoding for categorical and date variables and some standard scaling for numerical variables, in order for the regression model to work better and faster. However, this model doesn't have potential of improving due to lack of parameters.

Using this performance as a baseline one, we will test different models available from scikit-learn, and specifically from ensemble library which contains models usually useful in solving such problems.

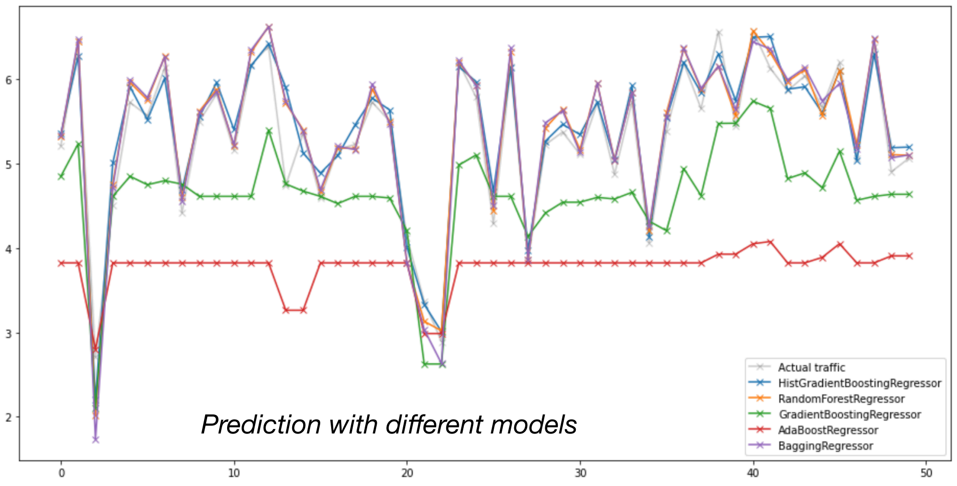
We will use the default parameters for comparing between different models and the preprocessing as described above: Standard scaling for numerical features, SIN-COS transformations for date and time features and One Hot Encoding for categorical features.

We can observe the following results for the unoptimised models:

| | RandomForestRegressor | GradientBoostRegressor | HistGradientBoostingRegressor | BaggingRegressor | AdaBoost Regressor | KNeighborsRegressor |
|---|---|---|---|---|---|---|
| **RMSE** | 0,774 | 0,831 | 0,726 | 0,792 | 1,11 | 0,89 |
| **Time*** | 48m 51s | 17min 33s | 1min 24s | 5min 15s | 9min 6s | 26min 10s |

**Time refers to time needed for training and cross-validating over 8 folds.*

For a random slice of the test data, the predictions of the above models can be viewed in the graph.



*Prediction with different models*

As we observe, HistGradientBoostingRegressor offers the best result in terms of prediction accuracy and in a timely manner as well. Hence, we will proceed to We choose gradient boosting regression as our default model since it is usually a good choice for tabular data. We will use the default parameters for comparing between different models before proceeding to parameter tuning.

## II. Hyper-parameter tuning

Using the HistGradientBoostingRegressor with the preprocessing as described above, we perform grid search on the above set of parameters :

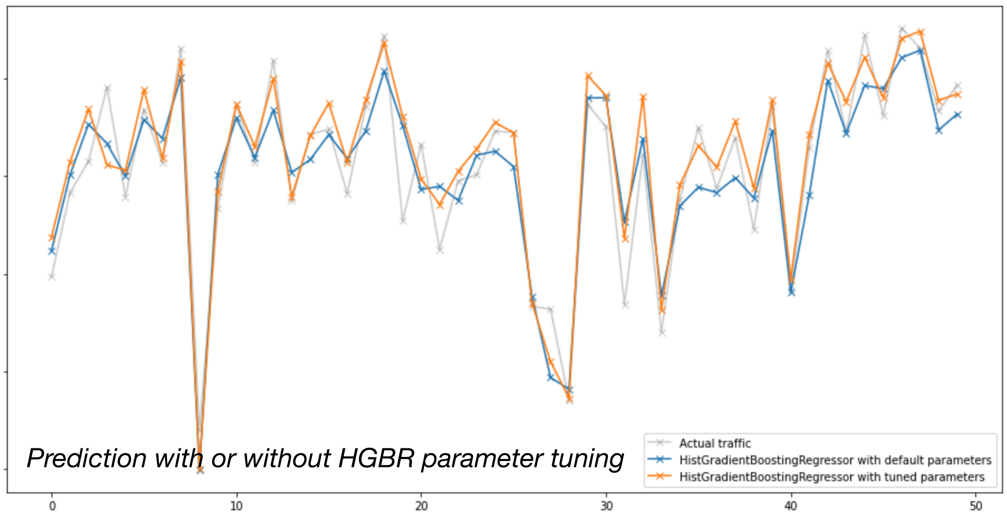
```
param_grid = {
    'histgradientboostingregressor__max_iter': [100, 500, 1000, 1200],
    'histgradientboostingregressor__max_depth': [None, 10, 20],
    'histgradientboostingregressor__min_samples_leaf': [15, 20],
    'histgradientboostingregressor__learning_rate': [0.01, 0.05, 0.1, 0.4, 0.7,
    1, 1.2]}
```

**learning_rate**
The learning rate of our regressor which is used as a multiplicative factor for the leaves values. For this parameter, the default value equal to **0.1** seems to have the best result out of different values.

**max_iter**
The maximum number of iterations of the boosting process, i.e. the maximum number of trees that are created. The default value is equal to **100**, but here the grid search shows that for our dataset a larger maximum number of trees, equal to **1000**, offers slightly better accuracy.

**min_samples_leaf**
This parameter sets the minimum number of samples per leaf. For small datasets it is recommended to lower this value since only very shallow trees would be built. Hence, since our dataset only includes it seems that min_samples_leaf = 20 gives a better result.

**max_depth**
The maximum depth of each tree. The depth of a tree is the number of edges to go from the root to the deepest leaf. Depth isn't constrained by default, so we test a couple of integer values that constrain the depth of the trees. Although, the default value set to None, meaning no constraint offered the best results.



*Prediction with or without HGBR parameter tuning*

## Conclusion

Finding and preprocessing external data, then comparing models and tuning the parameters, we manage to reach an RMSE of 0,7.

We could try to improve our result by finding Paris weather dataset per hour and keep exploring more models for example.