Ecole Polytechnique

MAP536 - Python for Data Science

RAMP Project: Predicting Cyclist Traffic in Paris

Master Data science for Business X-HEC, 2021

Authors

Hermine Tranie Benjamin Montagnes

Teachers

Mathurin Massias

Hicham Janati

Roman Yurchak

December 8, 2021

Contents

Introduction

Goal: Best predict cyclist traffic in Paris
Given data:

- Explanatory Variables:
  - 455163 observations from 2020-09-01 01:00:00 to 2021-08-09 23:00:00 in the train dataset
  - 30 different bike sites with localisation (latitude, longitude)
  - External data containing weather and traffic features
- Response Variable:
  - For each timepoint, the bike count at the given station

Based on the starting kit, we have loaded the data and produced a few exploratory data analysis plots. In the following, we will go deeper into the EDA, introduce feature engineering, perform model selection and hyperparameter tuning in order to improve our model.

Data Engineering

2.1 Variables from initial dataset

2.1.1 Bike count

The bike count is our target variable in this model. We need to look at its distribution in order to check for different kinds of model assumptions that might be needed. As per the

starting kit explanation, we are performing a log transformation on it to give it a more central node. Indeed, its distribution pre-transformation is highly skewed and least square error is not appropriate as it is designed for normal error distributions. In the following analysis, we are keeping the log bike count as the target variable to avoid domination of a small number of points.

2.1.2 Date encoding

In the initial dataset, the date is given as a string, which we transfer to a datetime object in order for python to treat it better. We then create columns for each of the aspects of a datetime object: year, month, day, weekday, hour. The data was collected every 3H, for granularity purposes and to have even more points when we import hourly data, we have decided to duplicate the data to make it hourly by keeping the same data points.

2.1.3 Spline transformation

Now that we have encoded the data, there is still an improvement that can be made on dates encoding. In fact, time data is cyclical, as in one hour there are 60 minutes, and one day has 24 hours. But when an hour is complete or when a day is complete, the next one starts counting minutes and hours from 0. This makes minute 1 and 2 strongly connected, but also minute 59 and 1, as there is only one minute difference, and not 59 as the model would assume. To tackle this issue, we can use sklearn.preprocessing.SplineTransormer using ‘extrapolation = “periodic‘. Thanks to the use of the extrapolation="periodic" parameter, we observe that the feature encoding stays smooth when extrapolating beyond midnight.

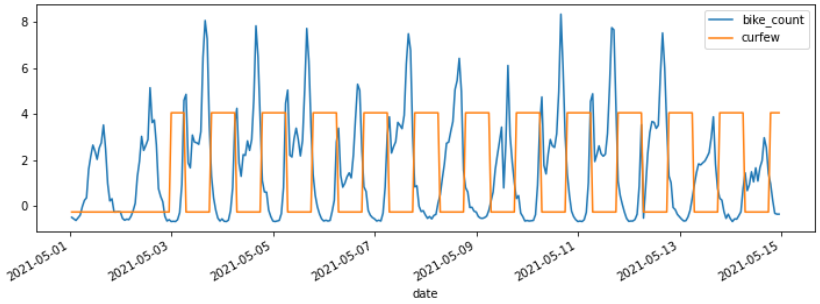
2.1.4 Categorical encoding

We then performed ordinal encoding on categorical data in the dataframe: counter name and site name: we basically mapped each unique label to an integer value from 1 to 30.

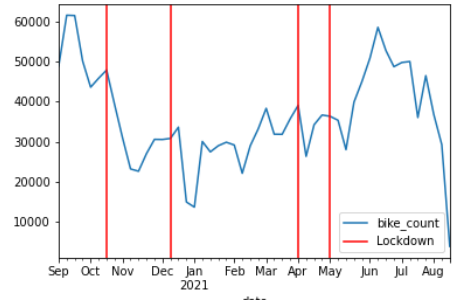
Data Visualization

NOTE: All the data that was used for the following graphs were Standard Scaled before so that we could see their evolution over time on the same graph.

To have an even more precise model we needed to fetch external data in addition to the weather data that was given to us. We thus needed to find data that was related to the traffic of bikes in Paris during the period we had. Our first intuition was that the COVID-19 pandemic must have impacted how people use bikes whether it's from working from home, being in lockdown or even the curfew situation that happened in Paris. We thus created a new column in the dataframe that we had already and placed a 1 if the corresponding date and hour was during lockdown or curfew or else 0. We get the following results :



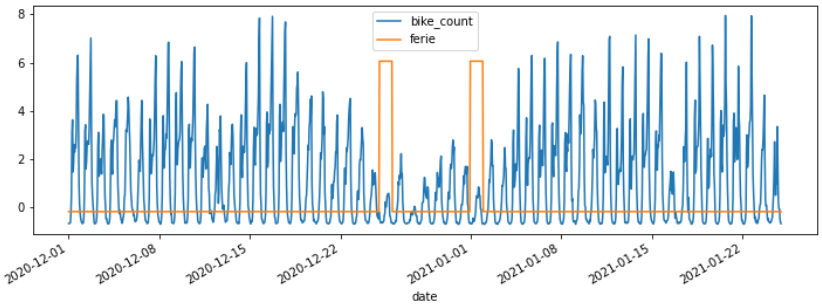
(a) Curfew



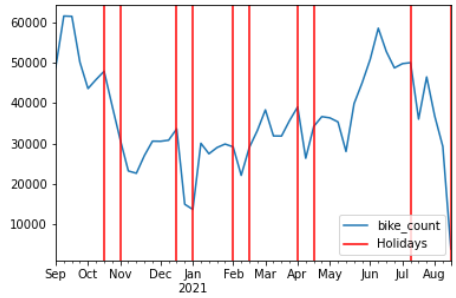
(b) Lockdown

We can clearly see how curfew and bike count are inversely correlated as people do not use bikes during these hours so it is good external data to include. Then, we can also clearly see on the year graph that during the 2 lockdowns that happened in Paris there is a clear local minima during these periods.

However, we noticed on the year grah a clear drop around the end of december as well as in mid august. We then linked these drops with the public holidays and the bank holidays and added them in our dataframe the same way we did for the lockdown and the curfew and this is the results we obtained:



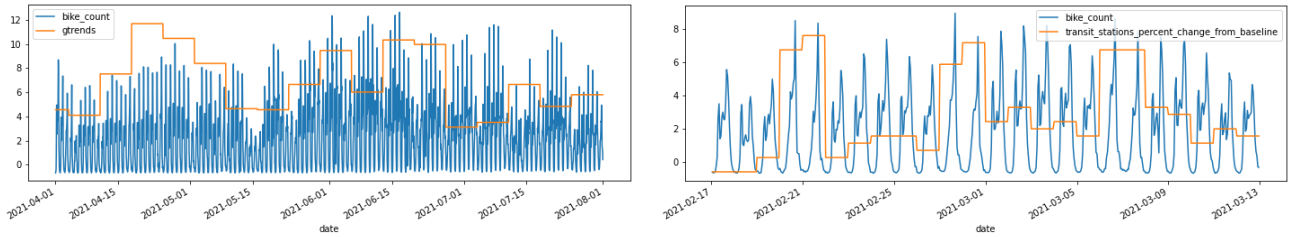
(a) Bank Holidays



(b) Public Holidays

We can see on the date we selected for the graph that clearly people during bank holidays use the bikes less than on other days. Moreover, during each holiday, a clear drop can be observed in the bike count data points. Those two external features are then going to be useful to improve a RMSE score.

We then wanted to broaden our field of approach. After searching keywords on Google trends that could be linked to bicycles in Paris, we clearly saw a trend in the words that Parisians would search linked to how they will move around. With a bit more research, we also found a Google Mobility Report that recorded the mobility trends for places like public transport hubs such as subway, bus, and train stations during the Covid pandemic. We thus used Google trends to sum up weekly the number of searches of a list of keywords (searches = ["velo electrique", "velib", "velib paris", "borne velib", "velo electrique prime", "velo electrique subvention", "velo tout chemin", "piste cyclable", "velo decathlon", "achat de velo","greve transport", "station velib", "greve metro"]) Note that the words “greve transport” and “greve metro” were included as we observed that people uses more bikes when transport strikes occured. We got the following graphs:



(a) Google Trends

(b) Mobility Trends

We can see that both features give interesting results. For instance we can clearly see a correlation between the mobility trends during weekends and the high bike count peaks on weekends. Google trends gives us weekly trends on people’s bike usage.

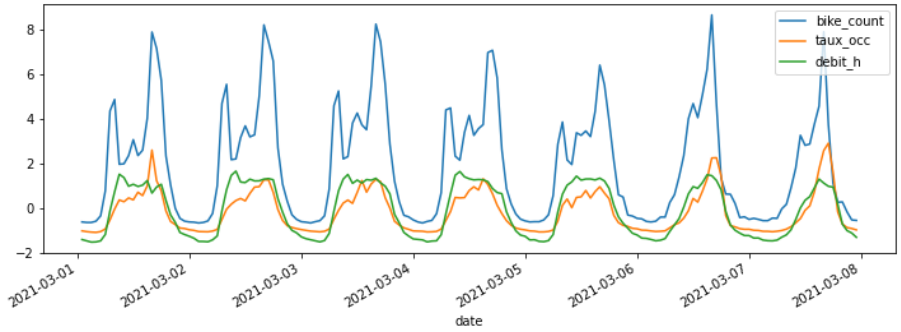
Finally, we noticed that metro traffic and car traffic in Paris were very correlated with bike traffic, as well as the rush hours when people go and come back from work explained the 2 daily spikes that we can observe on the graphs. We thus have :

From the graphs above we can clearly see the two rush hour peaks, as well as we can easily identify from the metro traffic the weekdays and weekends.

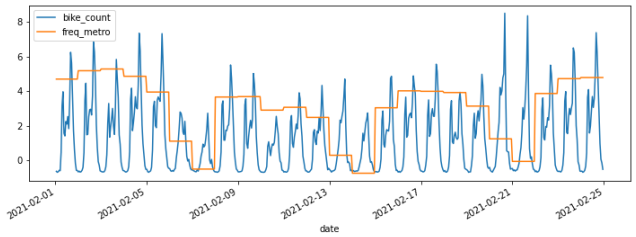
Data Modelling

4.1 Implement different models

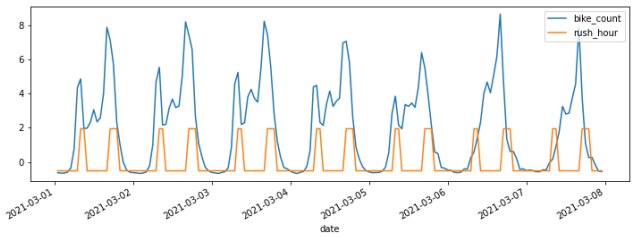
In the starting kit, a ridge regression was implemented with a RMSE score on the test set of 1.12. In order to improve this baseline model we have decided to implement different machine learning



(a) Google Trends



(b) Mobility Trends



(c) Mobility Trends

models notably tree based methods including: Random Forest, Extreme Gradient Boosting (XGBoost), Histogram-based Gradient Boosting (HistBoost), Light Gradient Boosting Machine (LightGBM) and CatBoost. The interesting aspect of boosting is that it is a better trade-off for bias-variance. Looking at the RMSE of all these models, pre-tuning we have that LightGBM and CatBoost were the most efficient and faster than the others. For example RF had a good RMSE in the end but took 120 times longer than LightGBM.

4.2 Select best model based on RMSE

Model :	Test RMSE	Time
Ridge	1.12	1s
RandomForest	0.44	3m 57s
XGBoost	0.49	15s
HistBoost	0.53	4s
LightGBM	0.52	2s
CatBoost	0.45	24s

Figure 4.1: Table of comparison of the different RMSE for the different models

4.3 Tune hyperparameters of best model

All these tests were run without any tuning or cross validation so there is a clear possibility of overfitting. We can do the same by testing with the estimator.py file locally to see how they behave with cross validation (still without tuning). However note that CatBoost updates its learning rate on its own so tuning might not be that useful.

Hence, we have decided to tune hyperparameters for LightGBM and XGBoost models. For the LightGBM model we get the following after 218 minutes of tuning using RandomizedSearch:

- number of leaves: 36
- lambda: 5
- min child samples: 350
- min child weight: 0.001

This yields a best RMSE of 0.711 on the test data on the RAMP platform.

For the XGBoost model we get the following after 222 minutes of tuning using RandomizedSearch:

- number of estimators: 1000
- subsample: 0.80
- max depth: 15
- learning rate: 0.1

This yields a best RMSE of 0.725 on the test data on the RAMP platform. Eventually, thanks to cross validation, we have made sure that there is no overfitting with the best parameters that were chosen. So we decided to pick the LightGBM model.

Conclusion

In a nutshell, in order to best predict the cyclist traffic in Paris, we have built a model through the following 3 steps:

- **EDA & Data Engineering:** Thanks to EDA, we have spotted potential engineering to be performed in order to improve features and make them more relevant through: encoding and spline transformer.
- **Data Visualization:** By adding external data, we needed to make sure they had some kind of similar behavior as the bike count in order to add insights to the model. This is why visualization are important and interesting to compare variables and spot patterns, for eg. When there is a curfew, the bike count is near zero and when it isn't there it's quite high.
- **Data Modelling:** The prediction part is achieved in this step. We compared 6 different models: 1 (the baseline) linear and 5 tree-based models. In the first-hand analysis, tree-based models outperformed any linear model due to the presence of high nonlinear and complex relationships between dependent and independent variables. The best model, from our analysis, with the lowest RMSE of 0.711 is using the LightGBM regressor.

5.1 Extension & Future Work

For future work, to improve the prediction of cyclist traffic, we could try other methods:

- Prophet model: split the data by site name (there are 30) and perform a prophet model on each of them, especially good for time series with strong seasonal effects and several seasons of historical data, as in our case.
- add lag feature: split the data by site name and add a lag feature of 1 day and 7 day to each timepoint then merge the data into one big dataset and perform the models in part 2 again.
- LSTM: deep neural network that can learn long term sequences of observations.

What we tried/learnt:

- we tried different date transformers: sine-cosine and spline and found that spline worked better for us
- hyperparameter tuning is extremely time consuming especially gridsearch, so we need to try a few ranges in randomized search and refine our search after