Data Science : Santé, Assurance et Finance

DATA CAMP
ASHRAE - Great Energy Predictor III

Étudiants :
DJIOMOU NGONGANG Cédric
LE Son Tung
NGUYEN Thi Ha Giang
TRAORE Babou

Encadré par :
GUILLOUX Agathe
BUSSY Simon

March 31, 2021

Contents

1 Introduction

1.1 Competition overview

Created by Antony Goldbloom in 2010, Kaggle is one of the most popular online communities for data scientists and machine learning practitioners. Hosted by ASHRAE on Kaggle, the competition our project worked on is "*ASHRAE - Great Energy Predictor III*".

Given data of over 1000 buildings during a three-year period, the purpose of this competition is predicting the efficiency of building energy usage in the areas: chilled, electric, hot water, water meter.

The competition's evaluation metric is Root Mean Squared Logarithmic Error, which is defined by following formula:

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(log(s_i+1) - log(a_i+1))^2}$$

where:
- n is the total number of observations in the (public/private) data set,
- s_i is our prediction of target, and
- a_i is the actual target for i.

1.2 Project summary

In this report, we describe briefly our work in the order of our steps for approaching the problem:

- *Firstly*, we study the data structure and relation between given data for future merge.

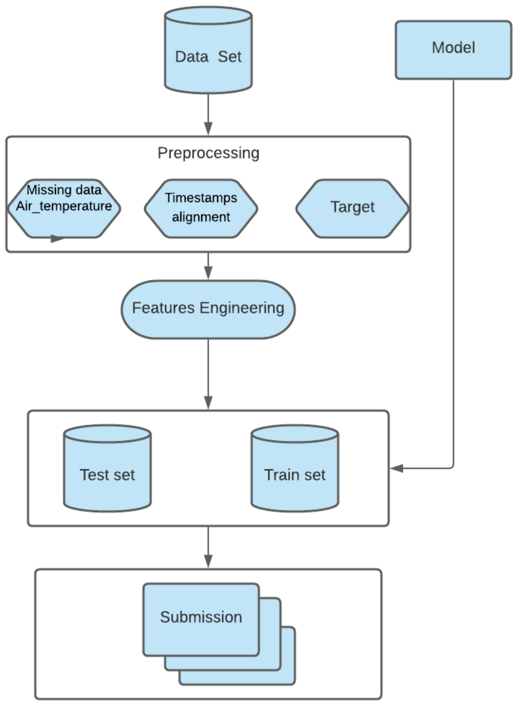
- *Secondly*, for explanatory data analysis (EDA) and preprocessing, we try to understand the variables, work on the missing data, target variable, outliers, etc. then feature engineering.

- *Thirdly*, we present our models, motivations behind and difficulties we encountered for each. Then we shows the results and our scores on the Leaderboard.

- *Lastly*, we conclude our work and give out possible improvement for future work.

For more information on our project and insight into our progress, please see *here*.

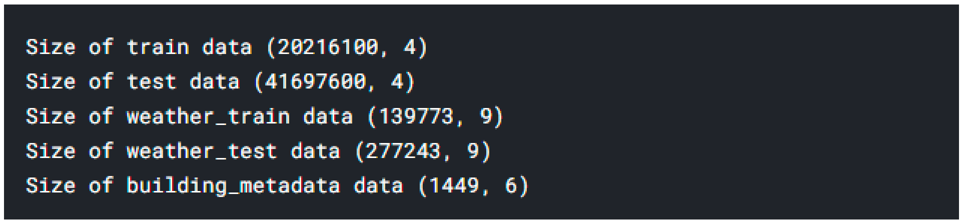
The figure below summarizes our framework for developping an accurate model for the competition:



2 Data description

The data of this competition can be found *here*. As written on the data description by Kaggle: "The data set includes three years of hourly meter readings from over one thousand buildings at several different sites around the world."

To be more specific, the data contains the information about the buildings and the weather corresponding to consumption time. There are five files in total (excluding the submission file), which the sizes are:



```
Size of train data (20216100, 4)
Size of test data (41697600, 4)
Size of weather_train data (139773, 9)
Size of weather_test data (277243, 9)
Size of building_metadata data (1449, 6)
```

The train file: The `train.csv` file containing four collumns: `building_id` (foreign key for the building metadata), `meter` (the meter id code), `timestamp` (when the measurement was taken) and `meter_reading` (the target variable).

In `meter`, we have four different types of energy: 0 - electricity, 1 - chilledwater, 2 - steam, 3 - hotwater.

It is noted that the units for the target variable are kWh, except for site 0 where the unit of electric meter reading are in kBTU. So we must converted the units to kWh.

The test file: This file is quite similar to the `train.csv` excluding the target variable. It is also include a `row_id` for the submission file so that predictions follow the correct order.

The weather files: These files contain the information about the weather from a meteorological station as close as possible to the site in train and test, including: air temperature, cloud coverage, dew temperature, trace precipitation, sea level pressure, wind direction and wind speed.

The building_meta file: This file contains the information of buildings, like the primary category of activities for the building (`primary_use`), the gross floor area (`square_feet`) or the number of floors (`floor_count`) of the building and the year building was opened (`year_built`). It also has the foreign keys for the train and the weather files (`site_id`).

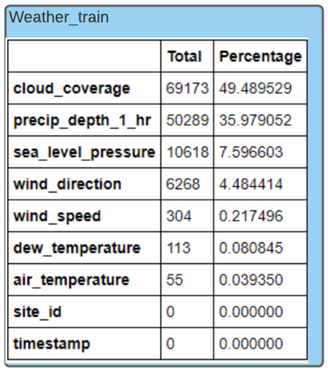
3 Explanatory data analysis and Preprocessing

3.1 Explanatory data analysis

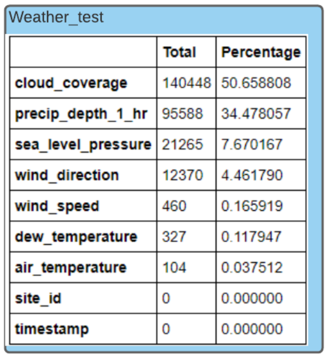
3.1.1 Detection of missing data

After the data loading, our first step is to check for missing values in the given data.

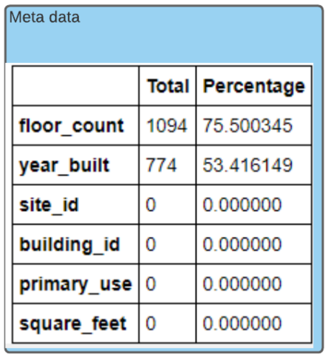
There are no missing data in the `train.csv` and `test.csv` files. For the others, we have the following results:


Weather_train

	Total	Percentage
cloud_coverage	69173	49.489529
precip_depth_1_hr	50289	35.979052
sea_level_pressure	10618	7.596603
wind_direction	6268	4.484414
wind_speed	304	0.217496
dew_temperature	113	0.080845
air_temperature	55	0.039350
site_id	0	0.000000
timestamp	0	0.000000


Weather_test

	Total	Percentage
cloud_coverage	140448	50.658808
precip_depth_1_hr	95588	34.478057
sea_level_pressure	21265	7.670167
wind_direction	12370	4.461790
wind_speed	460	0.165919
dew_temperature	327	0.117947
air_temperature	104	0.037512
site_id	0	0.000000
timestamp	0	0.000000


Meta data

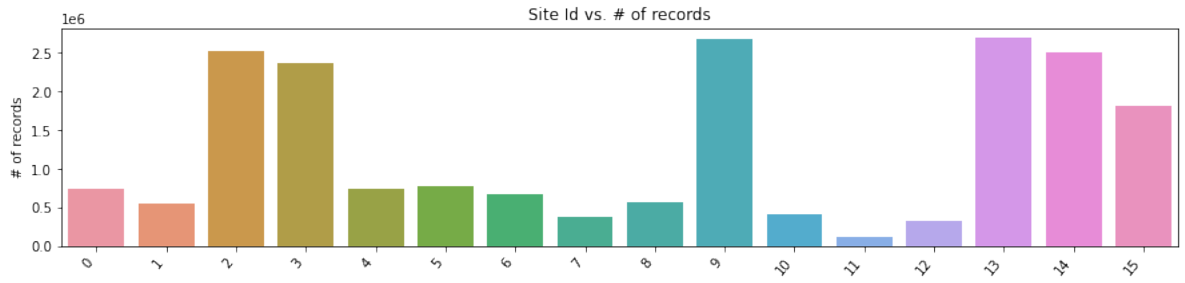
	Total	Percentage
floor_count	1094	75.500345
year_built	774	53.416149
site_id	0	0.000000
building_id	0	0.000000
primary_use	0	0.000000
square_feet	0	0.000000

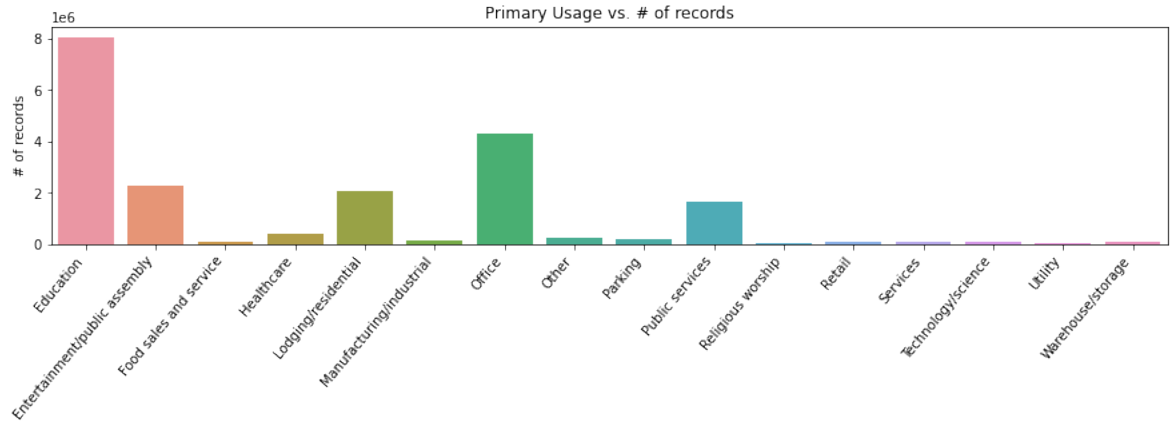
We can see the number of missing value for some variables are quite large, such as `cloud_coverage`, `precip_depth_1_hr` or `floor_count` and *year_built*.

After studying more about the variables, we will try to treat this problem.

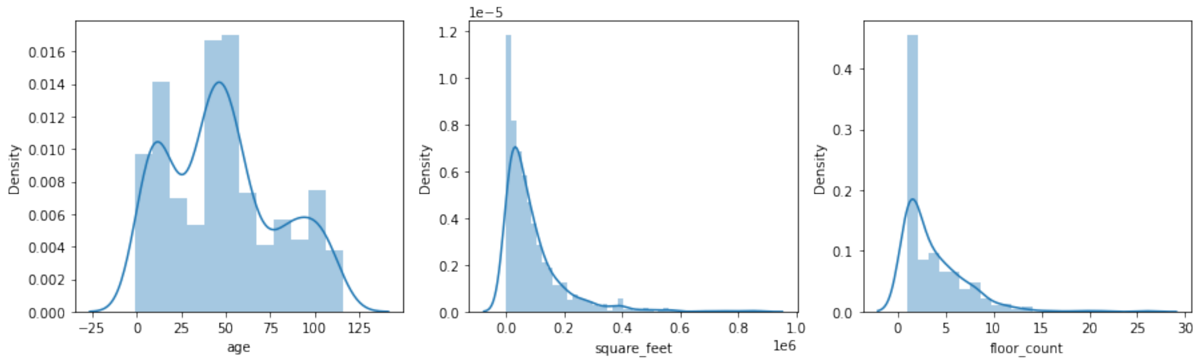
3.1.2 Building variables

The meta_data has 6 columns in total and has 1449 buildings (which is same as the number of building in train data). We visualize the distribution of each variable in this data for better understanding.





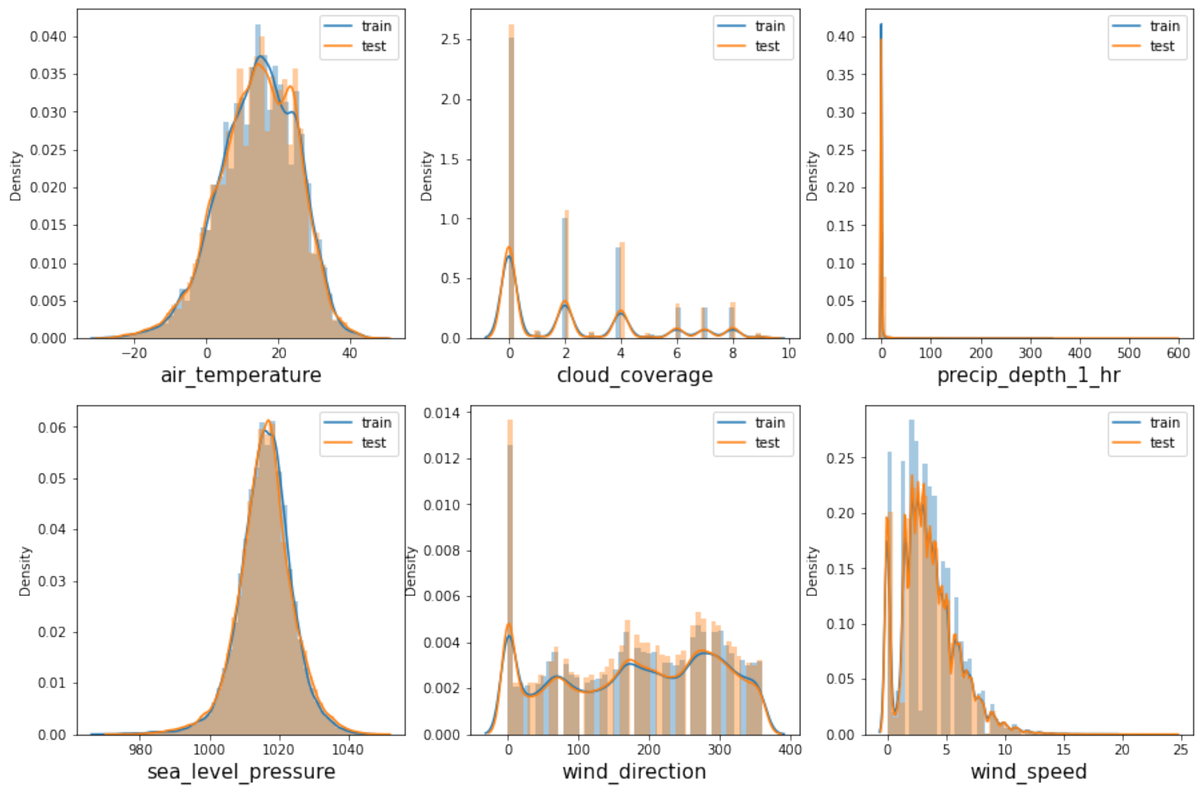
With the total of 16 site, the most popular primary usage is education, then office.



The age of these building (at 2016) range from 0 to around 115. Most of the building has les than 5 floors.

3.1.3 Weather variables

We continue with visualizing the `weather_train` and `weather_test`'s variables.

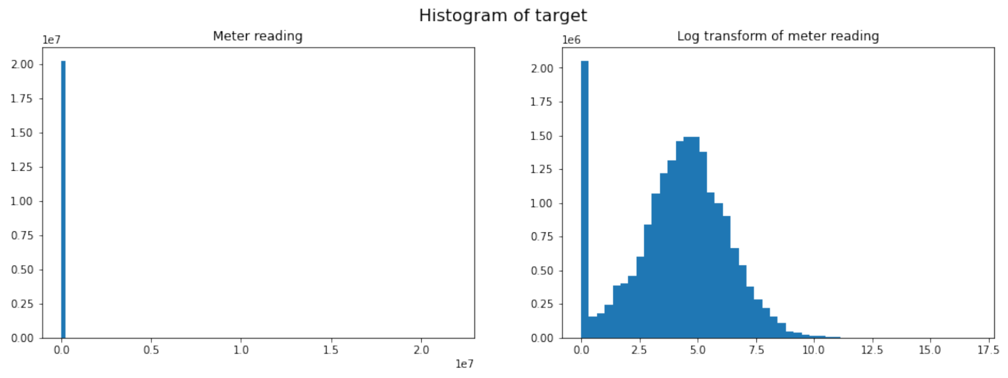


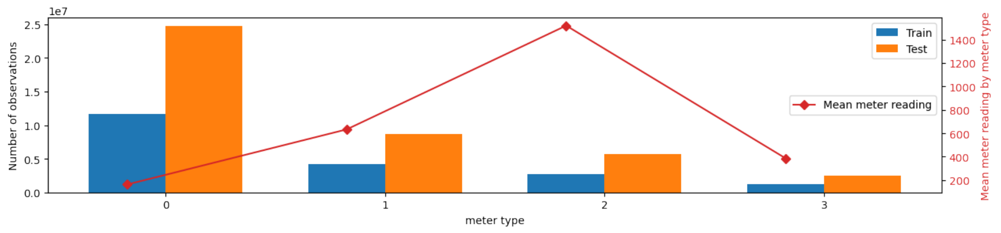
We see that the distribution of these variables on test set and train set are quite similar.

3.1.4 Target variable and Outliers

Our target variable is `meter_reading`. As noted *here* by the competition's host, the electric meter readings for site 0 is kBTU while other sites' are in kWh. So we convert all model inputs into kWh and then later we will get back to kBTU for scoring.

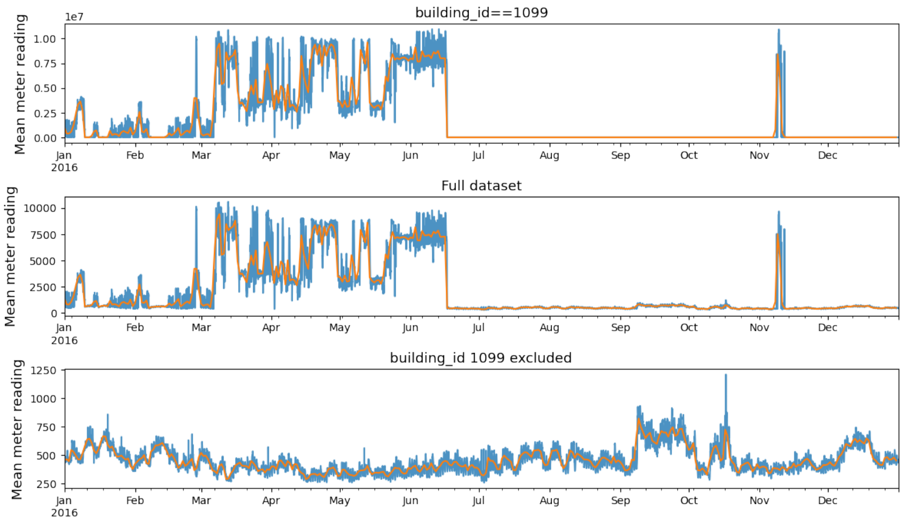
Because of the distribution of `meter_reading` and the evaluation of the competition (RMSLE), we transform the original target variable to obtain a new target variable `log1p(meter_reading)`.





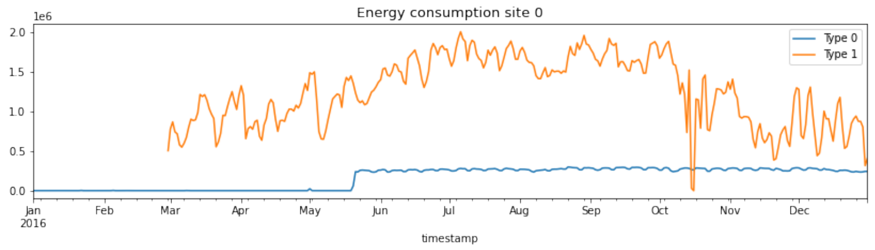
Outliers

There are some very large values in training set, which may cause noise to our model. We try to detect and remove them.



We found building 1099 consumes lots of energy, which affects the overall picture of our data. So we consider it as an outlier.

Another finding is about site 0, all the record of chilled water meter reading in this site before 20/05/2016 is 0. This might happen because site 0 was under maintainance during this time. So we should remove them for better prediction.



3.2 Preprocessing

3.2.1 Missing data treatment

Due to the difference characteristics of each variables, we have adopted different strategies to fill these missing values.

For the `meta_data` dataset, we consider `floor_count` and `year_built` seperately:

Year_build: There are 53,41% of data in `year_build` not available. As this represents the year building was opened, we believe it is rational to fill the missing ones by the *mean* of existed `year_build`.

Floor_count: According to the above table, 75.50% of `floor_count` are not available. As this is the number of floors of the building, it is reasonable to fill the NAs by 1, which is equal to the *mode* of the available ones.

As for the weather data, we have:

Air_temperature: There are 55 and 104 missing values in the column of `air_temperature` respectively in `weather_train` and `weather_test`. We notice that there's no NA from the same hour in two consecutive days of the same site. Therefore, we decided to fill the missing data with the value from the closest previous day without NAs, at the same time of the day, of the same site.

Others: The rest are `cloud_coverage`, `dew_temperature`, `precip_depth_1_hr`, `wind_speed`, `wind_direction`, `sea_level_pressure`. With these variables, since the weather's behaviour hardly changes in 12 hours, we choose to make a forward and backward filling with limit equal 12.

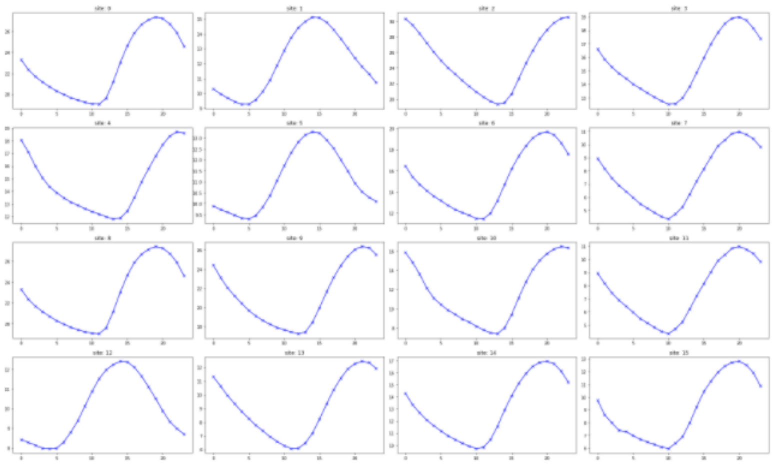
After that there are still some missing values in `cloud_coverage`, `precip_depth_1_hr`, `sea_level_pressure`. So, we group the rows of these columns by `site_id` and fill the missing values with the *mean* of its group.

Additionnal missing data: After merging the files and obtain new train and test, new missing values appear because some merging keys in the original train are not in the `weather_train` (same case for test and `weather_test`).

3.2.2 Timestamp Alignement

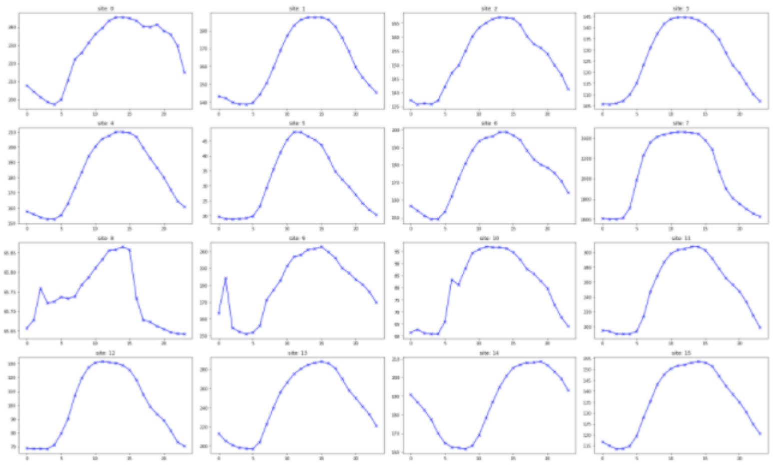
Motivation: Based on the suggestion of this *thread*, there's a probability that the timestamps of both `weather_train` and `weather_test` are not in local time format. To validate the existence of the problem, we follow the recommendation of this *kernel*.

First we look at the `air_temperature` in both data by site by hour with the assumption that highest air temperature should appear at around 14.



We see that the peak temperature seems off, for some site it even occurs at night. This means that the timestamp in this data is not the local time.

Then we look at the energy consumption by site by hour in the train data:



This makes sense. Site 14 seems to be the only difference here, with the energy consumption starts at around 10, however it might due to this site used mainly for education. So the timestamp in this data is more aligned to the local time.

Therefore, it is needed to align before merging these datasets by timestamps.

Alignment: We find the (mean rank of) hour of the day when each site has the highest `air_temperature` then obtain the time offset, which is the difference between 14 and said hour. For aligning, we shift all timestamp of the site by the calculated offset.

3.3 Feature Engineering

Belows are our process of feature engineering:

- Use `log1p` to transform target variable `meter_reading` and `square_feet`.

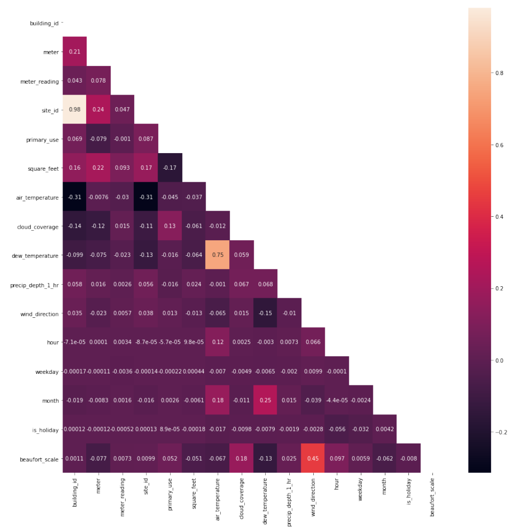
- Create some useful variables: month, holiday, date and hour (with the problem of the form of time series)

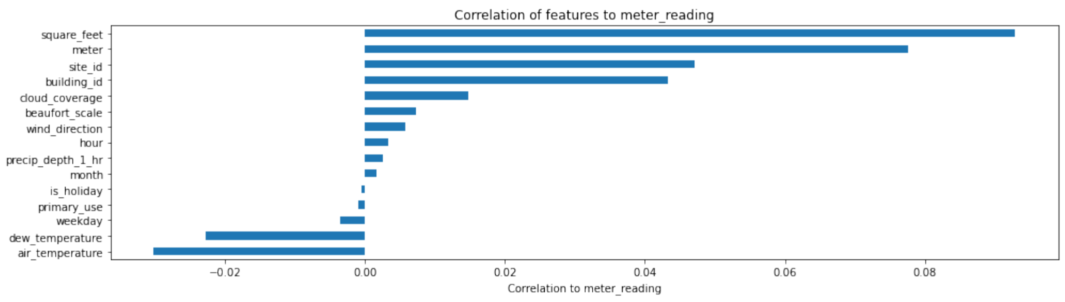
- Use label encoding for non-numeric categorical variable `primary_use`: label encoding assigns each class of `primary_use` to an unique number.

- Discretize two continuous variables as following:
  `wind_direction`: we divide the compass direction (0-360) into 16 main directions 0-22.5, 22.5-45,... and apply this to transfrom `wind_direction`.
  `wind_speed`: based on 12 level Beaufort wind force scale.

- Drop some variables that are not used in modelling: `sea_level_pressure`, `floor_count` and `year_built`.

- Measure the correlation between variables then zoom in the correlation between target variable and others:





4 Models and results

In this part we mention 3 models we have tried for this problem: Decision Tree Regressor, Gradient Boosting Machine and Light Gradient Boosting Machine. Then we made submissions to see the scores and compare them.

4.1 Decision Tree Regressor

4.1.1 Method description

The Decision Tree Regressor is a non-parametric supervised learning method (no assumption is made on the data distribution). It takes two enteries one the features and one the target. It learns the rules decision inferred from the data features in the goal to predict the value of a new observation target. In addition it works with a set of hyperparameters such as: criterion, splitter, max_depth, min_samples_split, min_samples_leaf, etc...

Mathematical formulation

Let Y $\in \mathbb{R}^d$ set of a quantitatives observations and X $\in \mathbb{R}^{n\times d}$ data of qualitives and/or quantitatives observations. For a given node m with data X_m, the goal is to minimizise the following function:

$$G(X_m, \theta) = \frac{n_{left}}{m} MSE(X_{left}) + \frac{n_{right}}{m} MSE(X_{right})$$

Where:

- θ give the thresold of spliting,
- $MSE(X_m) = \frac{1}{m} \sum\limits_{i \in N_m} (y_i - c_m)^2$,
- $c_m = \frac{1}{m} \sum\limits_{i \in N_m} y_i$.

4.1.2 Application and Results

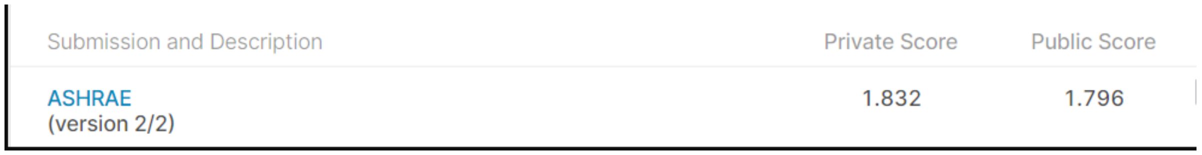
Firstly, the goal is to predict the variable *reading_meter* which is quantitative by accross other variables like square feet floor, count year build, sea level pesure, wind direction wind speed, dew temperature, cloud coverage, air temperature. So we are in situation of regression because the target is quantitative and the features are qualitative and/or qualitative.

We have choosen the Decision Tree Regressor as the first model. The goal was to make a first submission and learn the differents functionnality on kaggle. So the model trained on a small data taked after:

- conerting of type "Electricity" in site 0 is kBTU to kWh.
- Removing some very large values of target variable using IQR.

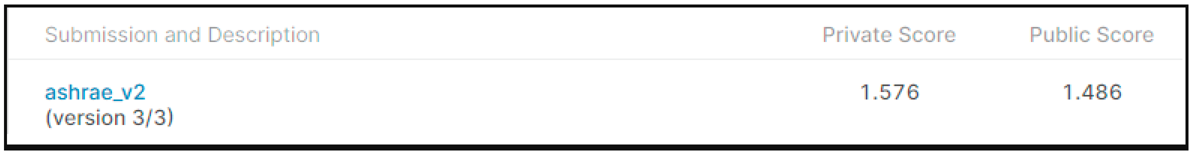
- Removing all variables contain NAs.
- Create one hot vector for ”primary use”.

In below is the score after training mode min_samples_split = 200, min_samples_leaf = 100 and the other hyperparameter was defined by default:



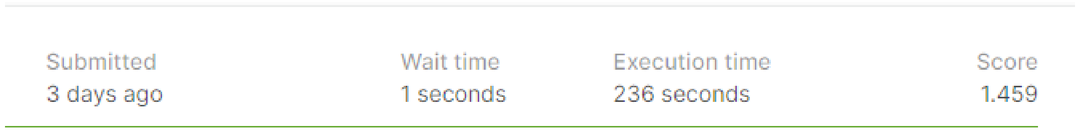
Submission and Description	Private Score	Public Score
ASHRAE (version 2/2)	1.832	1.796

This score is bad, to improve it we had judge to make differents traitements on the data set wich were to to fill the data Air temperature, timesamps alignement and target transformation. The second score was:



Submission and Description	Private Score	Public Score
ashrae_v2 (version 3/3)	1.576	1.486

This still is a litlle bad, we made a gridsearch cross validation. To do this we construct a pipeline where the parameter contains just two elements min_samples_split, min_samples_leaf with respectively possibility [100, 200, 250] and [100, 150, 200]. The optimal parameter we found is min_samples_split= 200 and min_samples_leaf=150. When we put this changements in ours DecisionTreeRegressor model we get a litlle good score which is:



Submitted	Wait time	Execution time	Score
3 days ago	1 seconds	236 seconds	1.459

For more information about all score, see *Here*.

4.2 Extreme Gradient Boost(XGBOOST)

4.2.1 Model description

Extreme Gradient Boost(XGBoost) is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework (GBM). GBM and XGBoost are ensemble tree methods. They use the gradient descent architecture to apply the principle to boost poor learners (CARTs.). XGBoost significantly improves the GBM framework by providing a parallel tree boosting that solve many data science problems in a fast and accurate way.
To learn more about XGBOOST: *here*

The difference between XGBoost and LightGBM(which is present below) lies in the time of execution. there’s a big difference within the execution time for the training procedure. Light GBM is more faster than XGBOOST and may be the better approach when handling large datasets in limited time competitions.

Mathematical formulation

For a given Y$\in \mathbb{R}^N$ and X$_i \in \mathbb{R}^D$ $i = 1, ..., N$; the goal is to find and compute the split parameters $\theta_m \in \mathbb{R}$ by following way:

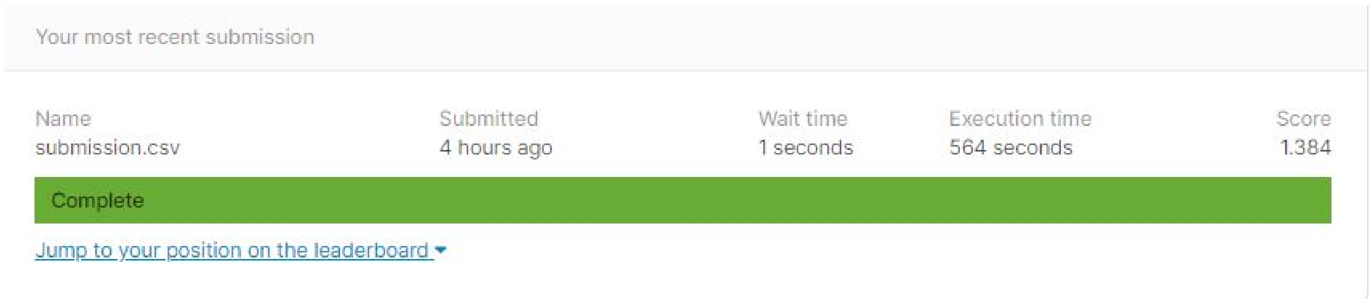
$$\{\hat{c_m}, \hat{\theta_m}\} = \min_{c_m, \theta_m} \sum_{i=1}^{N} L\left(Y_i, c_0 + \sum_{m=1}^{M} c_m T(X_i, \theta_m)\right)$$

Where:

- L(.,.) is the loss function,
- function T(.,.) is a Decision Tree Regressor,
- c_m is the aggregation parameters for $1 \leq m \leq M$,
- D is number of features and M is number of tree.

4.2.2 Application and Results

After applying a regression decision tree, the next step was to try to improve the performance by an ensemble method in this case the xgboost. We took as input parameters of the model: `n_estimators` = 20, `max_depth` = 10 and `reg_lambda` =0.5. To train the model, we took an `early_stopping_round` =50. We could have used a gridsearch to find the right parameters but memory space problems did not allow it. So we did it by hand. Thus, with some features engineering, and with these parameters, we managed to greatly improve the error although it is still higher compared to the other errors of the competition and therefore remains to be improved by LightGBM for example.



Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	4 hours ago	1 seconds	564 seconds	1.384

Complete

Jump to your position on the leaderboard

4.3 Light Gradient Boosting Machine (LightGBM)

4.3.1 Model description

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed for faster training speed, higher efficiency and lower the memory usage. It allows us to handle large data.

LightGBM use leaf-wise tree growth for optimization in accuracy, instead of level-wise like most of the decision tree learning algorithms. Moreover, the advancement of LightGBM lies in the use of the Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) techniques, which allows us to search faster than other gradient boosting decision tree method and still be reliable on large data set.

The detailed ideas and novel techniques in this method can be found in the original *paper*.

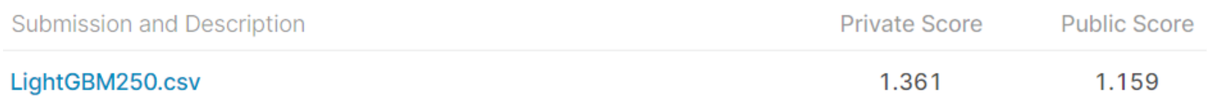
It is noted that LightGBM has an advantage of converging faster, however it is best used on data with more than 10000 records. On small dataset, using this method might result in overfitting.

4.3.2 Application and Results

After starting with the baseline model Decision tree regression, we try with boosting tree methods. These methods bring us some improvements in predition. But the data is too huge, bagging and boosting methods take a lot of time to run and usually we got the crash because of memory limitation on Kaggle. LightGBM has the most advantage in this case, so we decide to use it for final model.

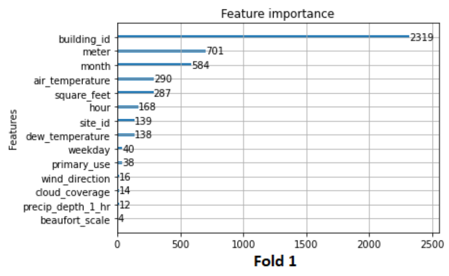
We use 3-folds cross-validation stratified on `building_id`. And set some parameters for tree base model: `learning_rate` = 0.1, `num_leaves` = 20, `feature_fraction` = 0.85, `lambda_l1` = 1, `lambda_l2` = 1, and try with different `num_boost_round`.

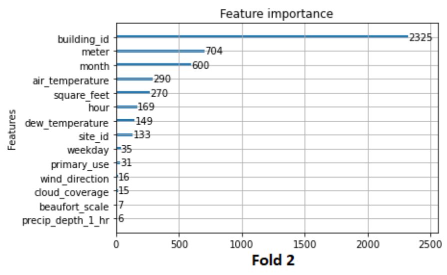
The best result we got is with `num_boost_round` = 250.

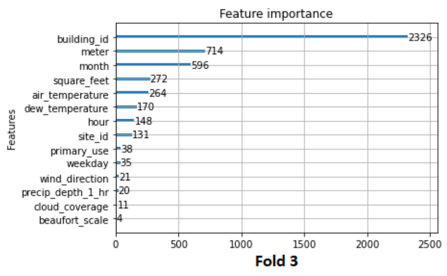


Submission and Description	Private Score	Public Score
LightGBM250.csv	1.361	1.159

Feature importance evaluation







We see that features do not contribute equally to the prediction of the target: `building_id` plays the most importance role in our LightGBM model, follow by `meter` and `month`. Aside from `air_temperature` and `dew_temperature`, the contributions to the prediction of other weather variables are quite small .

5 Conclusion

In conclusion, this project is an opportunity for us to work on a large set of real data. One of our most important part is EDA and processing. We dedicate a great amount of time for studying the structure of variables, filling missing values, finding outliers, etc. Then, we applied a number of tree based models: Decision Resgression Tree, XGBoost, LightGBM and compare the results.

Taking into account the large size of data set and the limitation of compution platform and time of execution, the performance of LightGBM is the best. As a perspective, we suggest to this algorithsm further in order to increase the scores. Other suggestions are studying further the outliers and using data leakage.

References

[1] Ashrae great energy predictor iii kaggle's competition. https://www.kaggle.com/c/ashrae-energy-prediction/overview.

[2] Lightgbm github repository. https://github.com/microsoft/LightGBM.

[3] Lightgbm's documentation. https://lightgbm.readthedocs.io/en/latest/.

[4] Xgboost's documentation. https://xgboost.readthedocs.io/en/latest/index.html.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.