



Mémoire de certification

Conduire un projet de sciences de données

RS 1527

Predicting response times of the Paris Fire Brigade Vehicles

QUACH Nai Minh

Data Science SAP3

DS SAP3 – 2019/2020

Predicting response times of the Paris Fire Brigade Vehicles

Nai Minh QUACH, October 30, 2020



Source: <https://www.epsu.org/article/epsu-deplores-tragic-loss-3-italian-firefighters>

Machine learning is called to save life. It comes to the rescue of Paris Fire Brigade to predict the duration required for emergency vehicles to arrive at their destination.

The challenge is a regression problem proposed by the prestigious Ecole Normal Supérieure: <https://challengedata.ens.fr/participants/challenges/21/>

In a megacity of 12 million inhabitants, burgeoning of traffic, constantly and endlessly transformed by public works and popular events, the invocation of “prediction” may sound like a kind of black magic power. The challenge will be burning for those who want to step up to the first place on the leaderboard.

Table of Contents

1	Overview	5
2	Data Exploration	6
2.1	Global view	6
2.2	Response variables	6
2.3	Stationarity and i.i.d.....	9
2.4	Principal Component Analysis.....	11
2.5	Feature variables.....	12
2.5.1	“OSRM” features	12
2.5.2	Spatial features	13
2.6	Trajectories.....	15
2.7	Categorical features	16
3	Features engineering	18
3.1	Time features.....	18
3.2	Spatial features	18
3.3	Mean speed and Traffic map.....	19
3.4	Sequential data	20
3.5	Feature decomposition.....	21
4	Models comparison	22
4.1	Ordinary Least Squares Regression.....	22
4.2	Ridge regression	22
4.3	Random Forest	23
4.4	Gradient Boosting.....	24
4.5	First Results.....	24
4.5.1	OLS model results	24
4.5.2	Models Comparison results.....	26
5	Gradient Boosting with Catboost.....	28
5.1	Catboost	28
5.1.1	Ordered Boosting	28
5.2	Categorical variable encoding.....	28
5.3	Strategy to overcome outliers.....	29
5.3.1	Early stopping	29
5.3.2	Hubert Loss function	29
5.3.3	Tree depth	30

5.3.4	Learning rate	31
5.4	Multitarget model – “nested” models	32
5.5	Train models.....	33
6	Results with Catboost.....	34
6.1	Leaderboard.....	34
6.2	General view on the model performance	35
6.3	Analyzing the R2 metric	37
6.4	Outliers analysis	39
6.4.1	Example1: 6 hours for 14 km.....	40
6.4.2	Example2: 2.5 hours for 3.1 km.....	41
6.4.3	How are outliers distributed in time?	42
6.4.4	How are outliers distributed geographically?	43
6.4.5	Are outliers detectable by machine learning?	44
6.5	Understand the model	45
7	“All models are wrong”.....	50
7.1	Predicting Confidence Interval with quantile-loss	50
7.2	Bayesian Model with MCMC	52
7.2.1	OLS Bayesian Model.....	53
7.2.2	Heteroscedastic Linear Model.....	55
7.2.3	Linear model with Student-T	56
7.2.4	Inliers and Outliers mixture model – Hogg method.....	58
8	Conclusion.....	62
9	Github repository	63

1 Overview

Our journey is an invitation to the land of supervised machine learning, a regression problem, with a dataset of 219 thousand records: <https://challengedata.ens.fr/participants/challenges/21>

The goal of the challenge is to predict the response time of intervention, i.e. the delay in second for an emergency vehicle between the moment it was selected and the time it arrived at the destination place. The target is decomposed into three dependent variables:

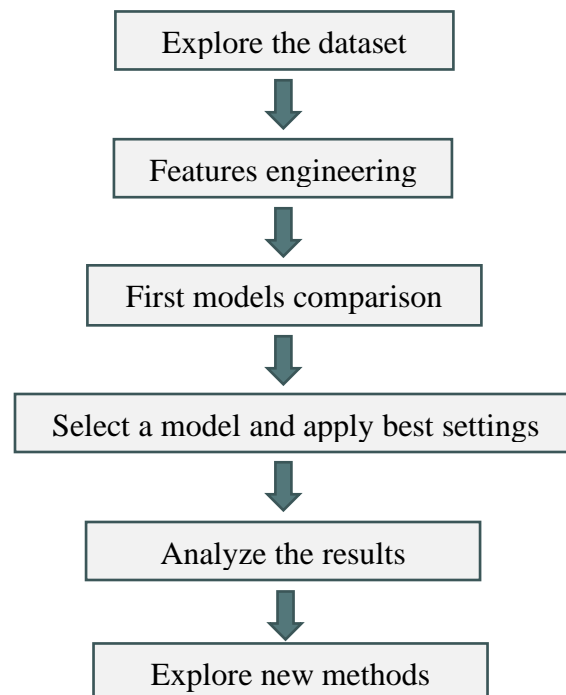
- **Y0**: the elapsed time between the selection and the departure of the emergency vehicle.
- **Y1**: the elapsed time between the departure and the arrival to the intervention place.
- **Y2**: the time between the selection and the arrival to the intervention place, $Y2 = Y0 + Y1$.

The feature data revolve around:

- Spatial data: GPS coordinates of departure and arrival, distances, street names,
- Temporal data: intervention date and time, estimated duration,
- Others: vehicle type, intervention type, etc.

The metric of the public leader board is R2 (R-square).

The following chapters in is document will present the journey in this order:



2 Data Exploration

2.1 Global view

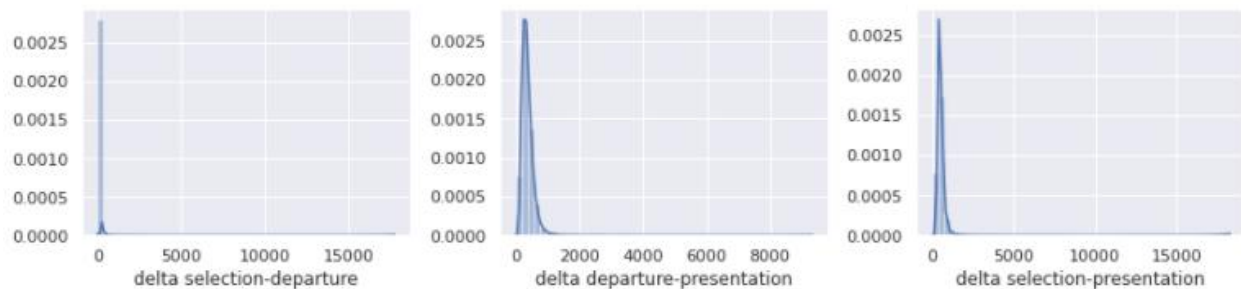
The given datasets are composed of:

- A training data is a cross sectional dataset of **219337 rows**, with **26 feature variables**, split into 2 files, and 3 response variables.
In addition to the usual scalar data types (integer, float, string, ...), we can also find fields in JSON format. They contain either a scalar or a list such as a list of street names, or GPS coordinates.
- A test dataset for submission, of **108003 rows**, i.e. 1/3 of the training dataset. It will be used for the challenge submission.

2.2 Response variables

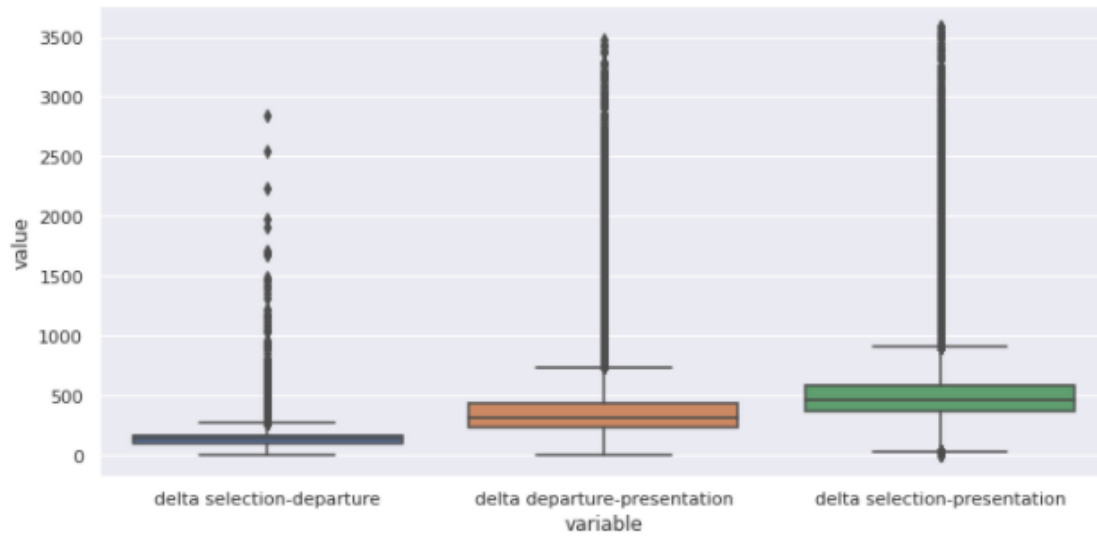
The response variables (Y0, Y1 and Y2) distributions exhibit high asymmetry:

Histograms of response variables - Figure 2-1



Values are very dispersed along the right tail.

Distributions of the response variables - Figure 2-2



The high values of skewness and kurtosis (see table below) already suggest some concern as we will have very little training data for those extreme values. The resulting models will have to predict them with low confidence. The presence of these **outliers** is going to prove our models in robustness.

Table of skewness and Kurtosis of response variables - Table 2-1

	Y	Skew	Kurtosis
0	delta selection-departure	74.323	14623.849
1	delta departure-presentation	17.157	1089.749
2	delta selection-presentation	17.169	1025.978

The tableau shows the skewness (skew=0 for Normal distribution) and kurtosis (“Excess-kurtosis”, 0 for Normal distribution) for the three response variables. They are all high values beyond 0.

If we exclude the extreme values and look at those are under 20 minutes. The three response variables are distributed as below:

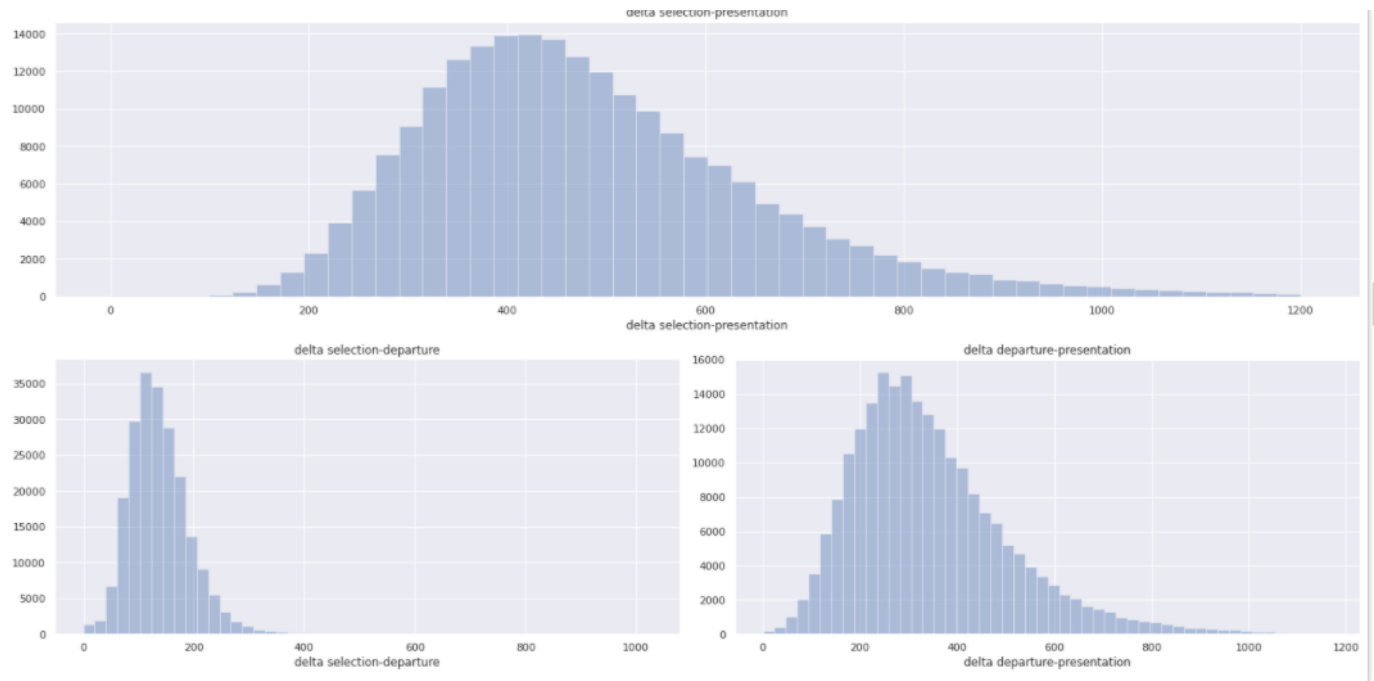
Distributions the response variables (< 20 minutes)

Top chart: Y2, “Delta selection-presentation”

Bottom left: Y0, “Delta selection-departure”

Bottom right: Y1

Figure 2-3



The top chart is the total time (Y2: delta selection-presentation). The bottom left, “delta selection-departure” (Y1). The bottom right: “delta departure-presentation” (Y2).

Right skewness is observed on these curves. That would suggest the **log transformation** to stabilize the variance.

The Pearson correlation matrix suggests there is no correlation between Y0 and Y1:

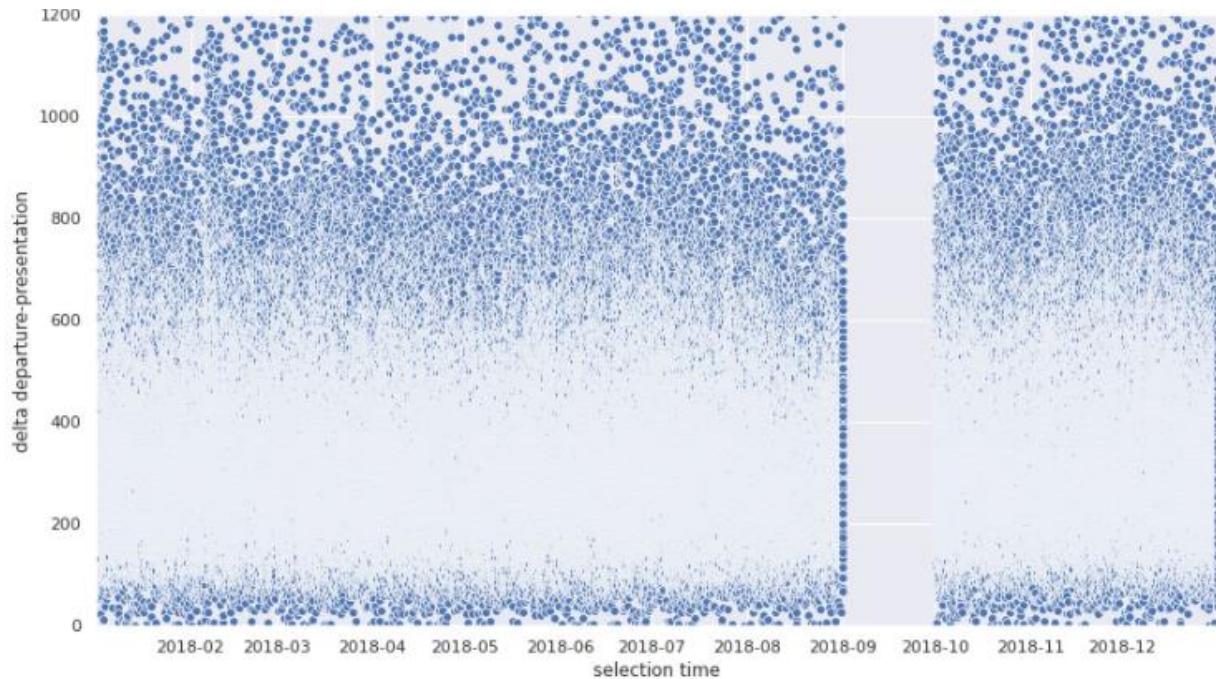
	delta selection-departure	delta departure-presentation
delta selection-departure	1.000	0.027
delta departure-presentation	0.027	1.000

2.3 Stationarity and i.i.d

The dataset is provided for the year 2018. It is important to check if there was any disruptive event that would break the assumption of i.i.d.: independent and identically distributed randomness.

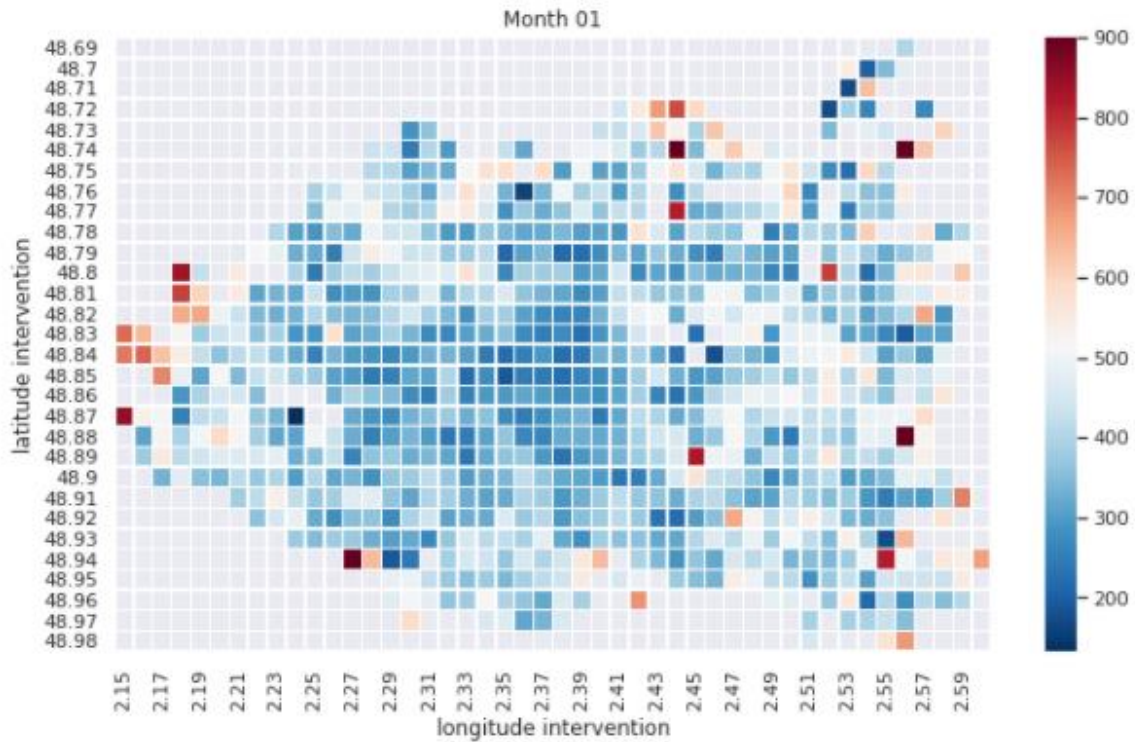
The chart below shows Y1 (“delta departure-presentation”) per “selection time”:

Response variable Y1 “delta departure-presentation” over time - Figure 2-4

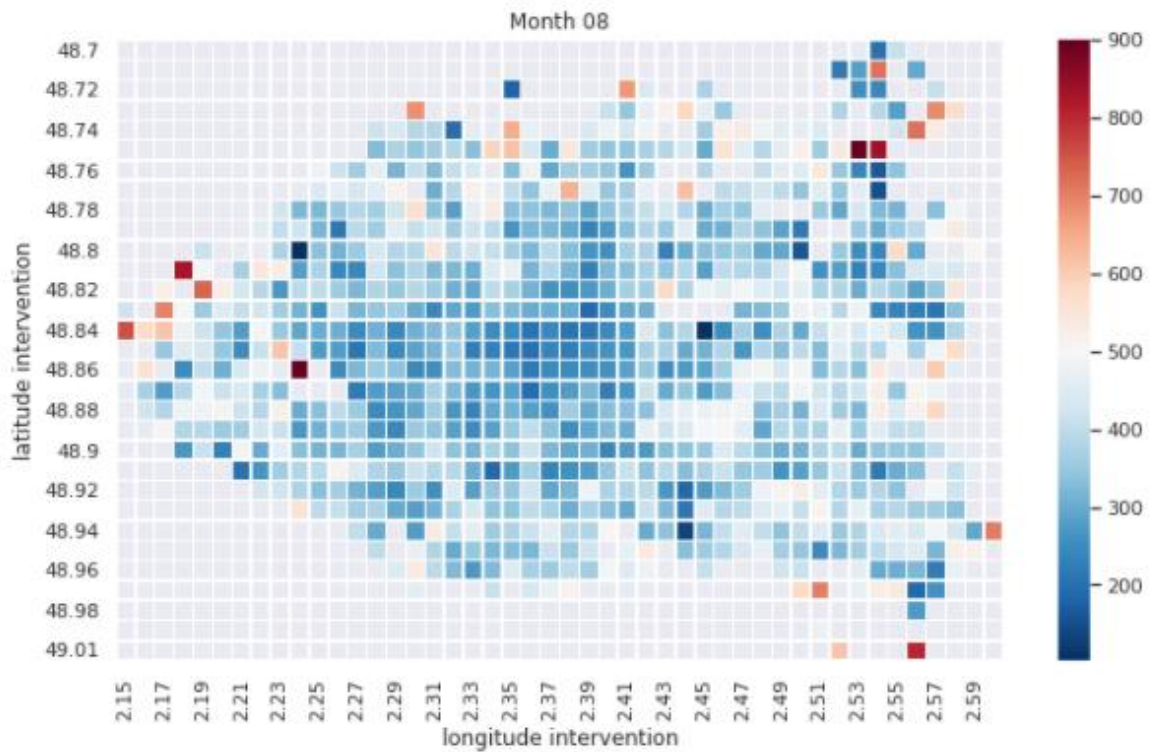


Except for disruption in September, the values seem to be stationary. We can double-check by comparing the response times, by areas, between January and August when activity is low.

Average response time (“delta departure-presentation”) on January over different coordinates -
Figure 2-5



Average response time (“delta departure-presentation”) on August - Figure 2-6

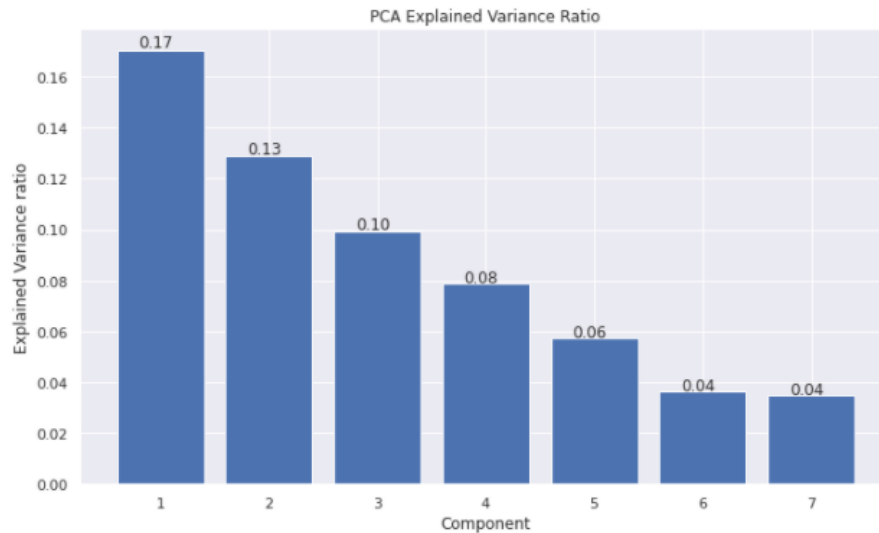


We don't see obvious disruption. The i.i.d. assumption seems to be respected.

2.4 Principal Component Analysis

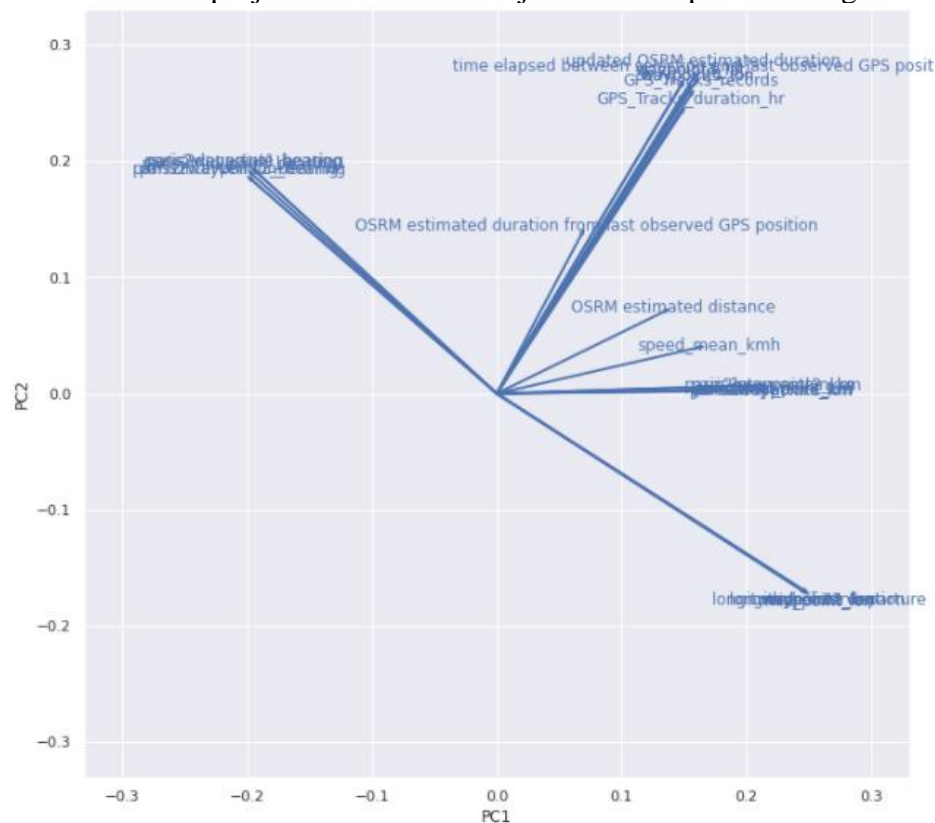
PCA is useful for showing how data is spanned in the feature space. The three top PCA components account for 40% of variance. The rest of variance is very dispersed.

PCA Explained Variance Ratio - Figure 2-7



If we projected the feature variables on the two first PCA components, we can get an idea how the different features contributes to the two PCA major components.

Feature variables projected on the two major PCA components - Figure 2-8

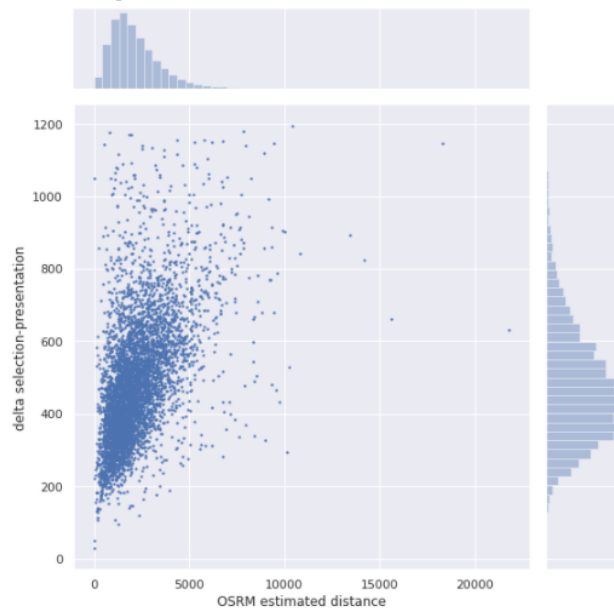


2.5 Feature variables

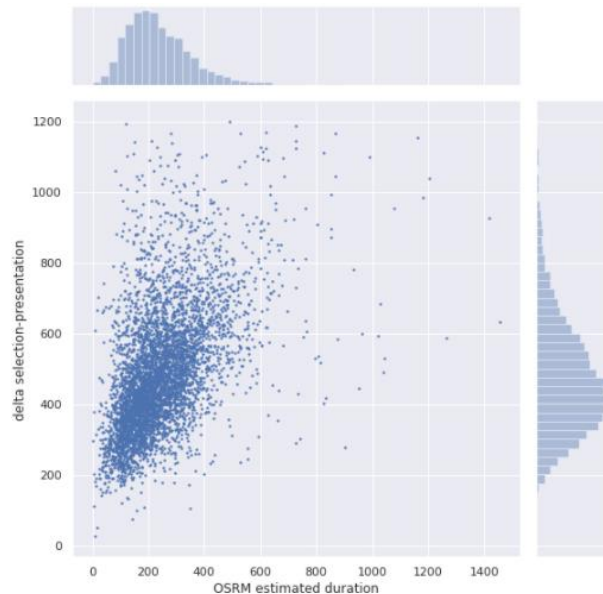
2.5.1 “OSRM” features

“OSRM” is a routing facility that allows estimating distance and duration. As one might expect, they are significantly correlated to the response variable. It should be a valuable variable for the future models.

Response variable “delta selection-presentation” per “OSRM distance” - Figure 2-9



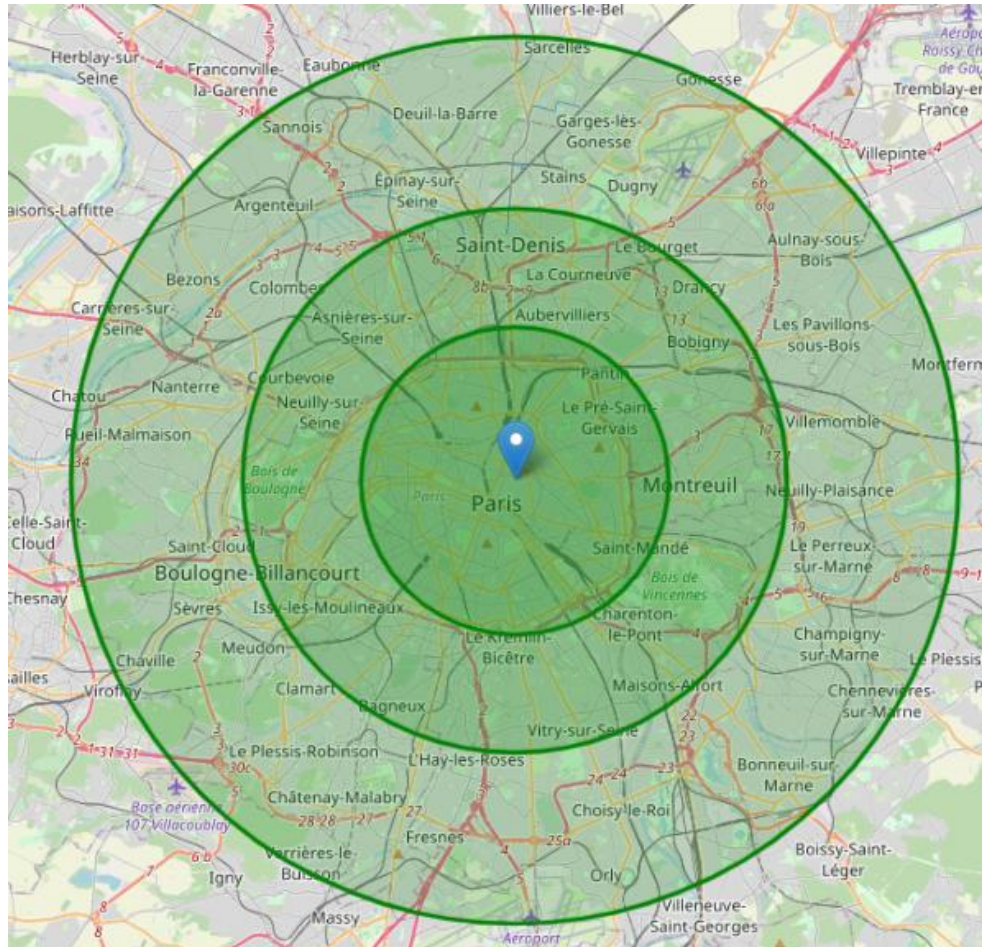
Response variable “delta selection-presentation” per “OSRM duration” - Figure 2-10



2.5.2 Spatial features

The GPS coordinates, longitude, latitude, are given for the departure and the intervention places. Using the python “folium” library, we can display their cumulative distribution over Paris. In the figure below, the concentric circles represent respectively 95%, 85% and 75%.

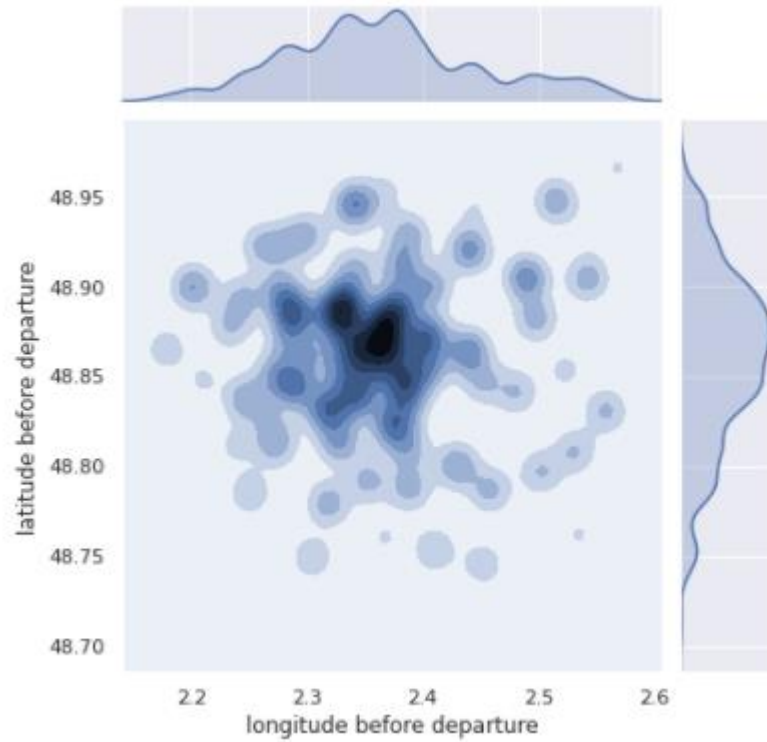
Distribution of the occurrence of interventions per area - Figure 2-11



More precisely, we can see the distribution of departures and destinations.

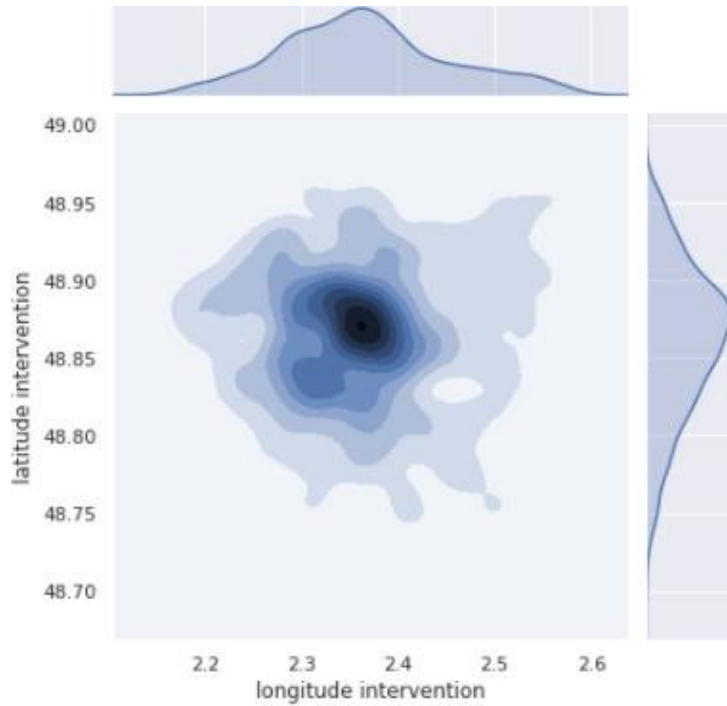
Departure distribution: this map locates all the fire stations in Paris.

Distribution of the departures' coordinates - Figure 2-12



Distribution of the interventions' coordinates

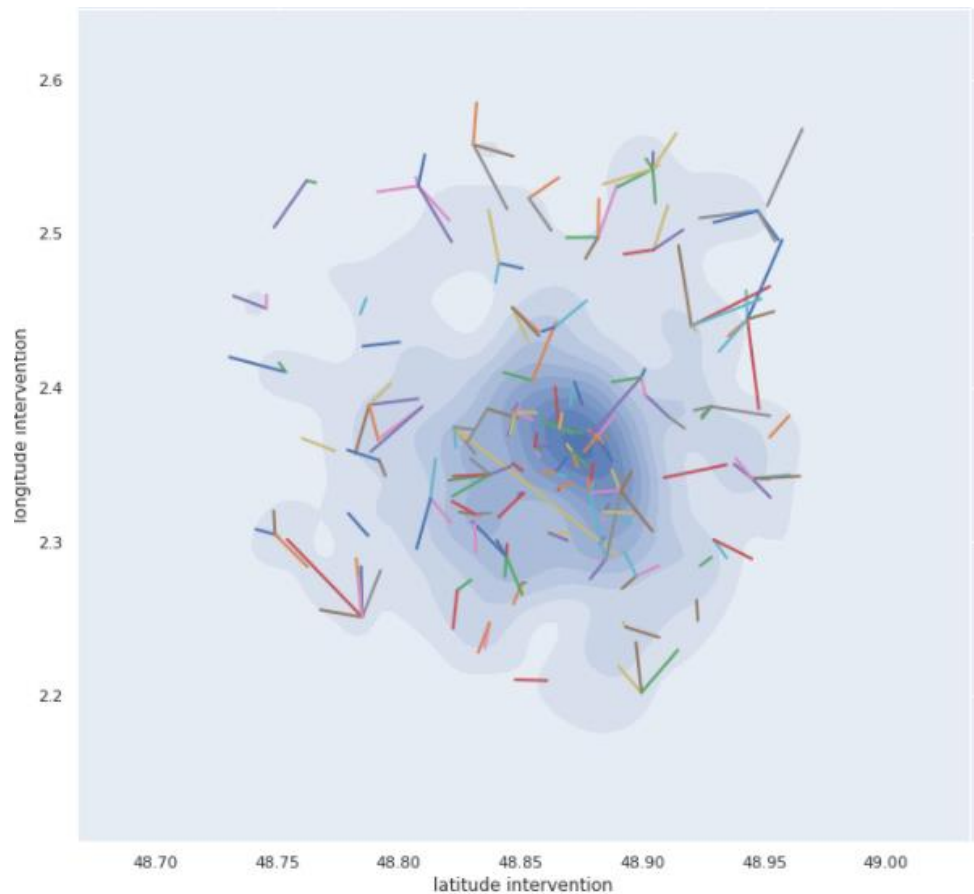
Intervention distribution: interventions are well spread over the city. Figure 2-13



2.6 Trajectories

The trajectories are mostly of small distances as the lines represented below:

Intervention trajectories - Figure 2-14

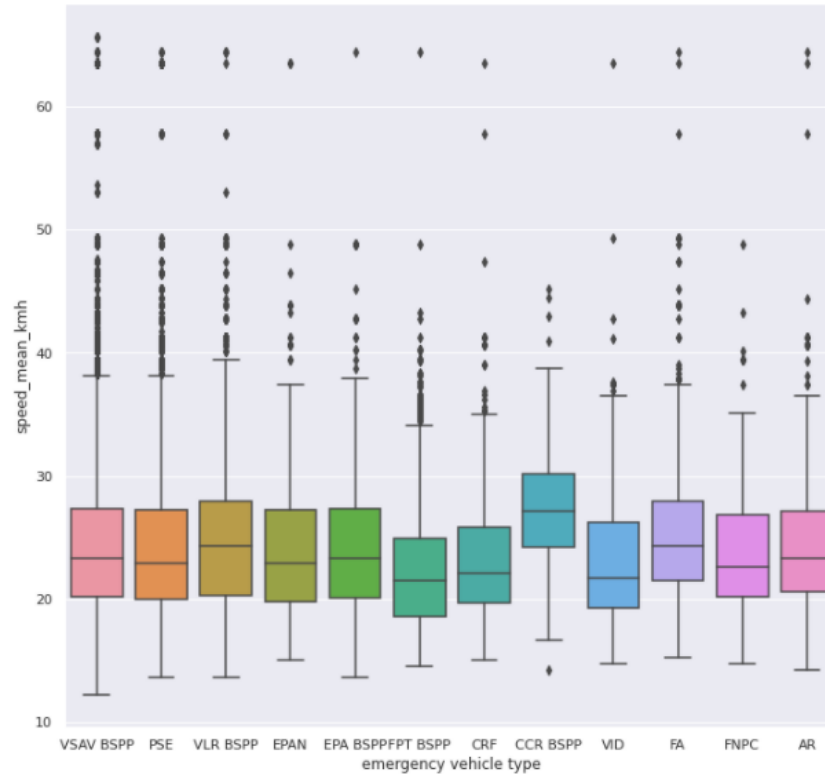


Given the times and distances, we can deduce the **average speeds**. This will be presented in the chapter 3.3.

2.7 Categorical features

To check if there is some correlation between categorical variables and the response times, the average speeds are computed per category.

Average of speed (km/h) per vehicle type - Figure 2-15



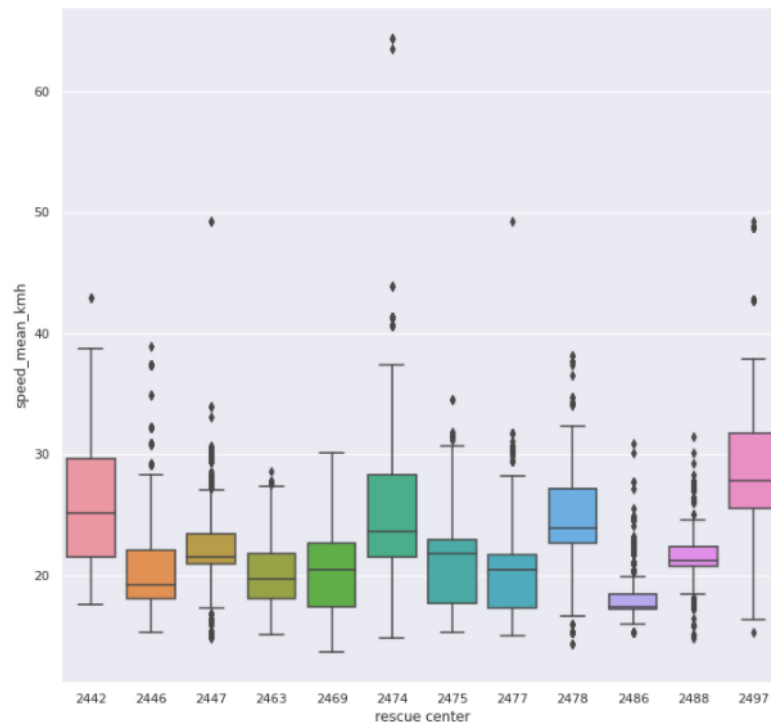
The vehicle types are generally little impacting, except for “CRR BSPP” (see picture below) which has an average speed slightly higher. This vehicle is a chemical intervention truck. The dangerous aspect may justify its speediness.

“CRR BSPP” vehicle type - Figure 2-16



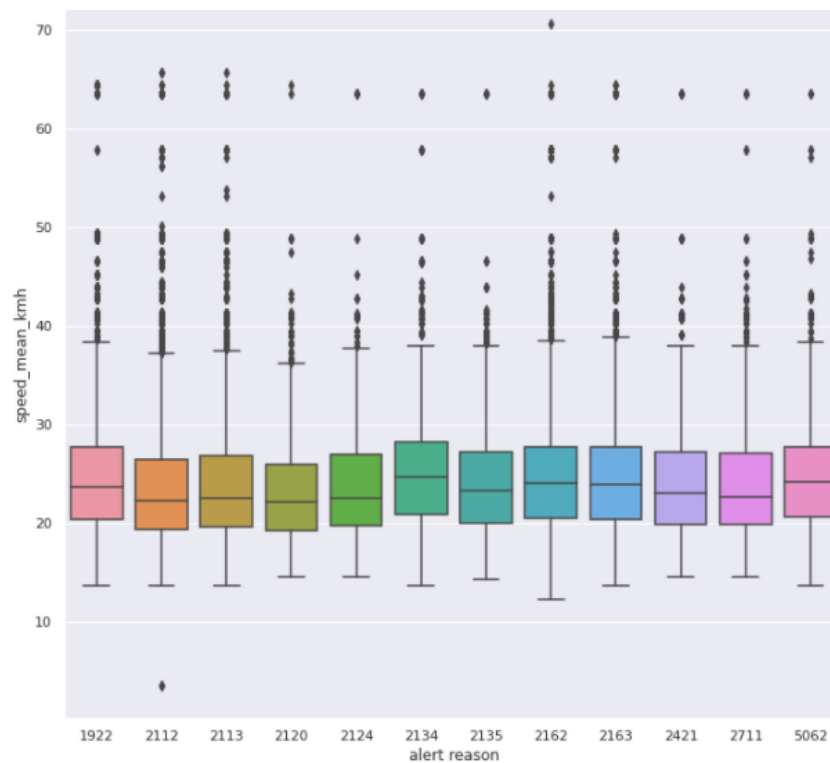
The rescue center seems also having a certain correlation with the average vehicle speed:

Average of speed (km/h) per “rescue center” - Figure 2-17



Other categorical variables do not show an obvious correlation:

Average of speed (km/h) per “alert reason” - Figure 2-18



3 Features engineering

From the initial 26 features, 30 new features are generated. The sections below will describe these new features.

3.1 Time features

From the selection date-time, we generate the time features below that may express some causal relation to the traffic state.

- The month of year,
- Day of week,
- Hour of day,
- Holiday (Boolean).

3.2 Spatial features

It is natural to suppose speeds are usually slower when it is closer to Paris center. It could also vary depending on the direction (east Paris looks easier than west Paris). We can extract the following features from the GPS coordinates:

- Distance from Paris center:
Distance in km from Paris center which is Cathédrale Notre-Dame de Paris.
- Bearing (direction):
We can also provide the direction of the intervention place, i.e. the bearing, the angle measure clockwise from the north direction.

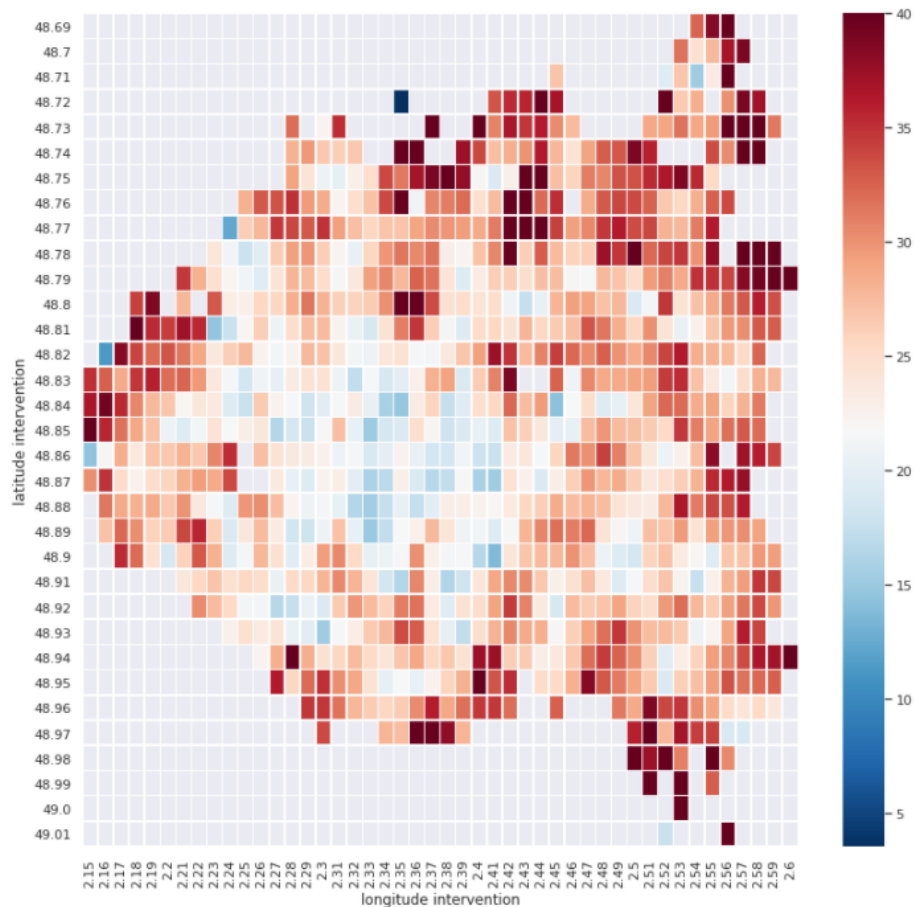
Given the coordinates of the departure and the arrival, to have the best guess of “where did the emergency vehicle pass by?”, we can compute the **midpoint** of the trajectory and its relative distance and bearing (angle) from Paris center.

3.3 Mean speed and Traffic map

For every intervention place, we try to determine how fast vehicles can run in average. We can estimate an average speed map of Paris. We are going to grid the city in tiles. These tiles have a dimension of 0.01° longitude and 0.01° latitude ($\sim 0.81 \text{ km}^2$). The mean speed is then estimated with the training dataset.

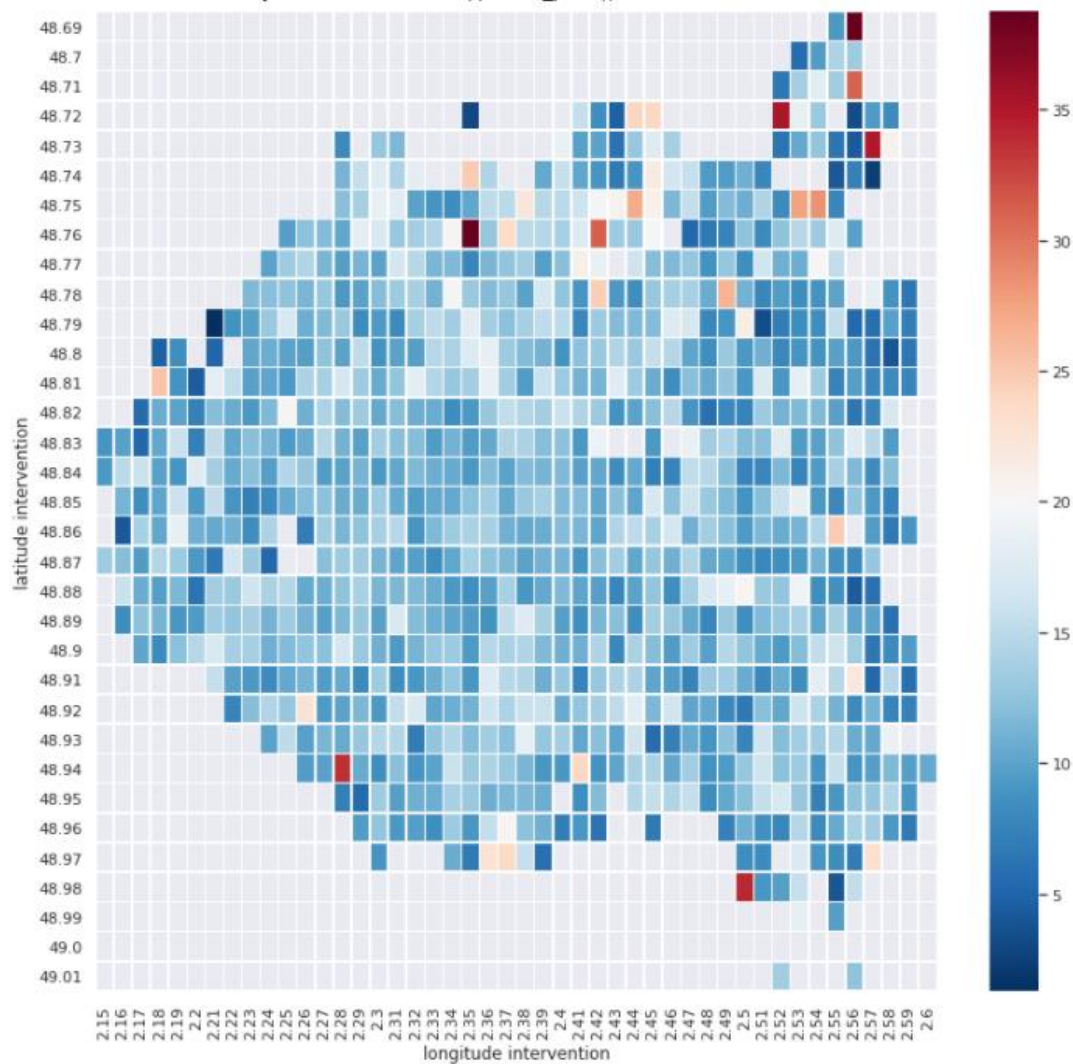
The resulting traffic map is below. Vehicles mean speeds do vary per region.

Average speed (km/h) per GPS coordinates - Figure 3-1



The variance of the speeds looks homogenous, mostly below 20km/h:

Standard deviation of speed (km/h) per GPS coordinates - Figure 3-2



3.4 Sequential data

Some fields in the dataset are given as lists of data. These lists are composed of a variable number of elements. We can find:

- List of GPS coordinates,
- List of date-times,
- List of street names (named as “waypoints” in the dataset).

Let’s consider the case of the fields “**GPS Track ***”. These fields are usually empty. They are none in **70% of the time**. But their role turned out to be crucial. They are provided by a tracking system which is triggered when the vehicle is slowed down. During the wait time, valuable information was given about the sequential wait times and positions.

The following features are extracted:

- Number of elements of the tracking list,
- Total duration: the sum of durations deduced from the list of date-times.
- Total distance: the sum of distances deduced from the list of GPS coordinates.

Alternatively, we call for text mining technics. It is particularly meaningful for analyzing a list of street names. We build a corpus made of 12311 street names (number of distinct street names found). To avoid the curse of dimensionality, we summarize them by “topic” extraction, i.e. using **Latent Semantic Analysis**.

We can generalize the technic to a list of durations or coordinates. Each duration or coordinate will correspond to a word.

3.5 Feature decomposition

The original feature “vehicle type” is composed of two words.

The first one corresponds to a real vehicle type ('VSAV BSPP', 'PSE', ...). See <https://www.pompiers.fr/pompiers/nous-connaitre/vehicules-des-sapeurs-pompiers>.

The second one corresponds to the owner of the vehicle:

- BSPP: Brigade de sapeurs-pompiers de Paris
- SSLIA: Service de sauvetage et de lutte contre l'incendie des aéronefs,
- Etc

We can separate them for more informative features.

4 Models comparison

Since there are many methods for regression, we can first quickly try them out with default settings. We will then be able to select one that seems the most promising and turn it.

But before implementing the different types of models, we can summarily analyze them to estimate their suitability to our problem.

4.1 Ordinary Least Squares Regression

The great classic of regression models would be the linear model. The most basic variant is the OLS estimator (Ordinary Least Squares). It allows expressing the response variable as a simple linear relationship with the feature variables:

$$y = X \beta^t + \epsilon$$

where $\epsilon \sim N(0, \sigma)$, σ is constant (homoscedasticity)

This simplicity allows a very easy interpretation, such as the contribution of the feature variables. It is capable to **extrapolate** more easily than the trees methods which would have difficulty predicting values that they had not seen. But this aspect is not needed in our case: we only need interpolation as our test dataset is distributed in the same way as the training dataset. The test dataset covers the same time range (2018 from January to December) and the same locations.

The [Gauss Markov Theorem](#) tells us that if the assumptions of OLS are met, the ordinary least squares estimate for regression coefficients gives us the **best linear unbiased estimate (BLUE)**. These assumptions are:

1. Linearity: the dependent variable is a linear function of the variables specified in the model.
2. Random: our data must have been randomly sampled from the population
3. Non-collinearity: the regressors are not perfectly correlated with each other.
4. Exogeneity: the regressors are not correlated to the error term.
5. Homoscedasticity; the variance of the error is constant for all predicted values.

Nothing can guarantee these assumptions are true in our problem. In particular, the 1) and 5) are probably difficult to meet in practice. Nevertheless, the linear model is worth being tried as the analysis of its results will provide insight for our next experiments.

4.2 Ridge regression

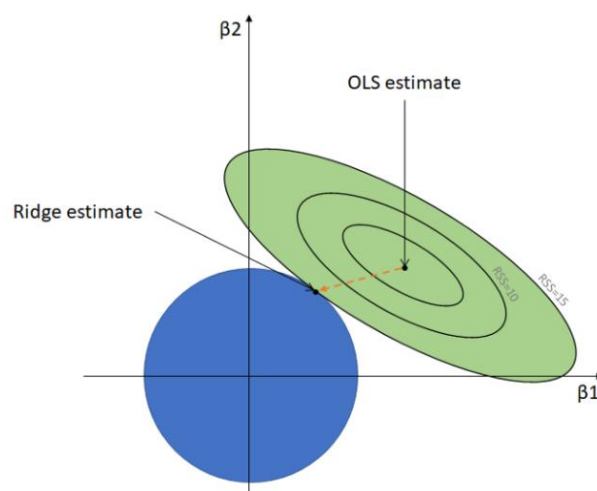
When we visualized the data in the section above (*Distributions of the response variables - Figure 2-2*), we had noticed the distribution of the response variables have lots of extreme and dispersed values (high skew and kurtosis). That suggests we will have to face **outliers**. These outliers will certainly augment the **variance** of our model. A model of high variance means that the variance of its estimated parameter, $Var(\widehat{\beta})$, is high: every time, we fit the model with a sample of the population, we get a very different parameter, $\hat{\beta}$, due to a small number of extreme values. Consequently, we can hardly get the true parameters. The errors of the predictions are then also impacted.

To combat this variance, we can extend the linear regression with the regularization concept. In a frequentist point of view, regularization simply consists in adding a penalty term to the squared loss in the loss function.

$$L_{Ridge}(\hat{\beta}) = \|y - X\hat{\beta}^t\|^2 + \lambda \|\hat{\beta}\|^2$$

The consequence of this new term is to restrict the search space for the parameters, ending up with a lower variance:

Ridge Regression - Figure 4-1



Source: [Medium](#).

One might wonder about the weirdness of this Ridge penalty. From a Bayesian point of view, the Ridge regression is equivalent to consider a **gaussian prior for the parameter β** :

$$\begin{aligned}\beta &\sim N(0, \lambda I) \\ Y &\sim N(X\beta, \sigma)\end{aligned}$$

where λ denotes the Ridge hyperparameter and I the identity matrix.

The estimation of the parameter consists then to get the **MAP (Maximum A Posteriori)** estimate:

$$\hat{\beta} = \operatorname{argmax}_{\beta} (\text{Likelihood}(\beta) * \text{Prior}(\beta)) / \text{constant}$$

Applying log transformation, the term likelihood (of a gaussian) is proportional to the squared loss, $((Y - X\beta)^T(Y - X\beta))$, and the prior term is simply the ridge term, $\lambda\beta^t\beta$.

4.3 Random Forest

The two main concerns we had sensed for the linear model are:

- The true model is not linear.
- The model **variance** will be high as there are many outliers.
This problem is also characterized as **overfitting**.

Random forest may provide an appreciable advantage:

- Its tree base learners are capable to treat nonlinear problems.
- By the principles of “Central Limit Theorem” and “Law of Large Number”, when the number of trees in the model increases, the variance of the “Ensemble” will decrease.

The principle is quite simple. It is an additive model with many base models (CART trees), f_i :

$$g(x) = \sum_i f_i(x)$$

Those base models are constructed so that they are not correlated to each other, otherwise the principle of the Central Limit Theorem won't work. For that, each of them is built with a different subsample of features and rows of data (bagging). These independent trees are individually weak learners, not complex, usually very biased. But the aggregation of all of them will make a good job.

4.4 Gradient Boosting

If Random Forest seems to be a good way to combat variance, reducing bias is another challenge. To combat also bias, Gradient Boosting is good as well for the job.

Alike Random Forest, Gradient Boosting is also an additive algorithm:

$$g(x) = \sum_i \alpha_i f_i(x)$$

But this time, the base models, f_i , are not anymore independent. Each new tree, f_i , is trained to correct the errors left by the previous one. Seen differently, a new tree aims at descending a step in the gradient of the loss. Its target variable is equivalent to a gradient. The coefficient α_i , is a learning rate. It tends to limit the gradient descend step.

4.5 First Results

In the rest of the document, all tests are done with the dataset randomly split. 2/3 (146955 rows) for the training dataset, 1/3 for the test (72382 rows).

4.5.1 OLS model results

Using the python statsmodels library, the model for Y1 ("delta departure-presentation") is below. The model is statistically significant as the probability(F-statistic) is 0. However, the assumption of gaussian error is far unlikely (Prob (Omnibus) = 0).

OLS Results - Table 4-1

OLS Regression Results			
Dep. Variable:	delta departure-presentation	R-squared:	0.402
Model:	OLS	Adj. R-squared:	0.399
Method:	Least Squares	F-statistic:	118.0
Date:	Wed, 29 Apr 2020	Prob (F-statistic):	0.00
Time:	15:12:06	Log-Likelihood:	-9.6652e+05
No. Observations:	146955	AIC:	1.935e+06
Df Residuals:	146120	BIC:	1.943e+06
Df Model:	834		
Covariance Type:	nonrobust		

- Prob(F-statistic): 0. The linear model is significant in overall.
- R-squared: 0.402, 40% of the variance is explained by the predictors.

Residuals analysis - Table 4-2

Omnibus:	343433.582	Durbin-Watson:	2.002
Prob(Omnibus):	0.000	Jarque-Bera (JB):	16542035361.331
Skew:	22.544	Prob(JB):	0.00
Kurtosis:	1646.026	Cond. No.	1.95e+24

The residuals are very unlikely to be gaussian:

- High Omnibus: the errors are very skew compared to the normal curve.
- Prob (Omnibus) = 0 = probability that the residuals are normally distributed.
- Kurtosis is far above 3. That means heavy tail and much more outliers than the normal.
- Skew is far above 0. That means the residuals are highly asymmetric. The tail is heavier on the right side
- Durbin-Watson tests for homoscedasticity. We hope to have a value between 1 and 2. In this case, the data is close, but within limits.
- Cond. No. = $\sqrt{\frac{\max(\text{eigenvalue}(X))}{\min(\text{eigenvalue}(X))}}$. It takes a very large value (normally ≤ 30). There might be multicollinearity, making the model very unstable.

OLS Coefficients - Table 4-3

	coef	std err	t	P> t	[0.025	0.975]
intervention on public roads	-2.5670	0.891	-2.881	0.004	-4.313	-0.821
longitude intervention	-64.4894	32.815	-1.965	0.049	-128.807	-0.172
latitude intervention	30.2556	45.492	0.665	0.506	-58.909	119.420
emergency vehicle	-0.0013	0.000	-6.206	0.000	-0.002	-0.001
date key selection	0.0004	0.000	1.405	0.160	-0.000	0.001
time key selection	-0.0002	2.9e-06	-63.749	0.000	-0.000	-0.000
delta status preceding selection-selection	8.475e-05	4.32e-06	19.596	0.000	7.63e-05	9.32e-05
departed from its rescue center	-1.4475	2.494	-0.580	0.562	-6.336	3.441
longitude before departure	-23.0661	109.732	-0.210	0.834	-238.138	192.006
latitude before departure	-195.3615	118.633	-1.647	0.100	-427.880	37.158
delta position gps previous departure-departure	0.3652	0.108	3.396	0.001	0.154	0.576
OSRM estimated distance	-0.0013	0.000	-6.372	0.000	-0.002	-0.001
OSRM estimated distance from last observed GPS position	-0.0012	0.000	-7.652	0.000	-0.002	-0.001
time elapsed between selection and last observed GPS position	0.3141	0.003	96.584	0.000	0.308	0.320
selection_weekday	0.3882	0.093	4.159	0.000	0.205	0.571
selection_day	0.0342	0.021	1.619	0.105	-0.007	0.076
selection_is_holiday	0.5976	1.084	0.551	0.581	-1.527	2.723
OSRM_estimated_speed	-0.2271	0.168	-1.351	0.177	-0.556	0.102
departure2intervention_bearing	-0.0051	0.002	-2.097	0.036	-0.010	-0.000
waypoint3_lon	15.9498	5.229	3.050	0.002	5.700	26.199
waypoint3_lat	-1.9433	0.253	-7.667	0.000	-2.440	-1.447

Most of the coefficients have p-value ($p > |t|$) under 0.05 (5%). They are said to be statistically significant.

4.5.2 Models Comparison results

The table below shows the results of the different models with the response variables Y1.

Models comparison Table 4-4

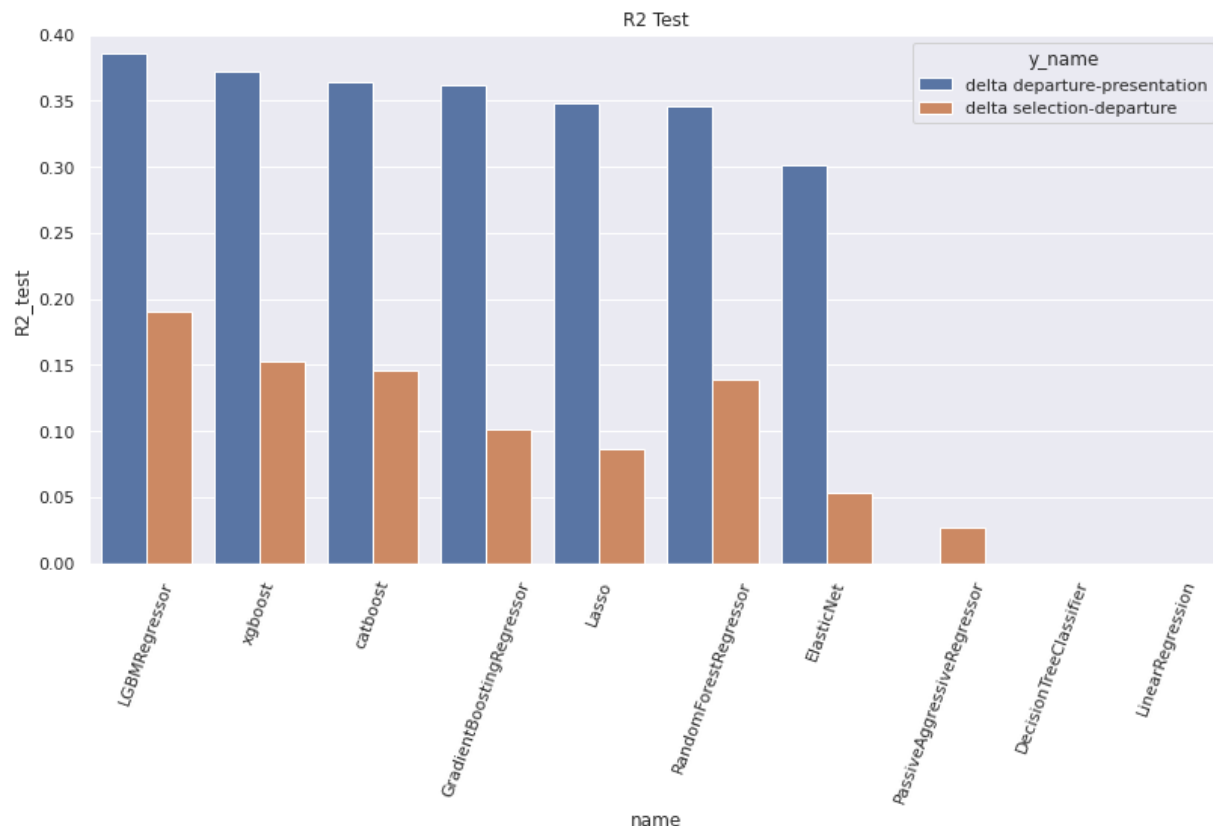
	name	y_name	mse_train	R2_train	mse_test	R2_test	fit_time
0	LinearRegression	delta departure-presentation	8.942668e+09	0.411048	8.942668e+09	-161424.824914	34.363681
1	Lasso	delta departure-presentation	3.607889e+04	0.363949	3.607889e+04	0.348733	143.974838
2	ElasticNet	delta departure-presentation	3.871577e+04	0.321248	3.871577e+04	0.301134	141.181175
3	RandomForestRegressor	delta departure-presentation	3.623539e+04	0.915944	3.623539e+04	0.345908	2233.367075
4	GradientBoostingRegressor	delta departure-presentation	3.533983e+04	0.495268	3.533983e+04	0.362074	369.455270
5	xgboost	delta departure-presentation	3.479908e+04	0.464300	3.479908e+04	0.371835	284.869950
6	LGBMRegressor	delta departure-presentation	3.402136e+04	0.506784	3.402136e+04	0.385874	16.512115
7	catboost	delta departure-presentation	3.521822e+04	0.496765	3.521822e+04	0.364269	63.352263

We can notice models that fit well on the training dataset, i.e. yielding high R2_train value, do not necessarily predict well on the test set. This is the problem of **overfitting**. It is true for LinearRegression which has a good R2-train (0.411) but completely failed on generalization, which R2-test is overly negative.

ElasticNet (Lasso + Ridge) and Random Forest show their capability to combat variance with a R2-test above 0.3.

The gradient boosting models, xgboost, LGBM, Catboost, are on the top of the podium. The chart below shows the results of the different models with the two response variables, Y0 and Y1.

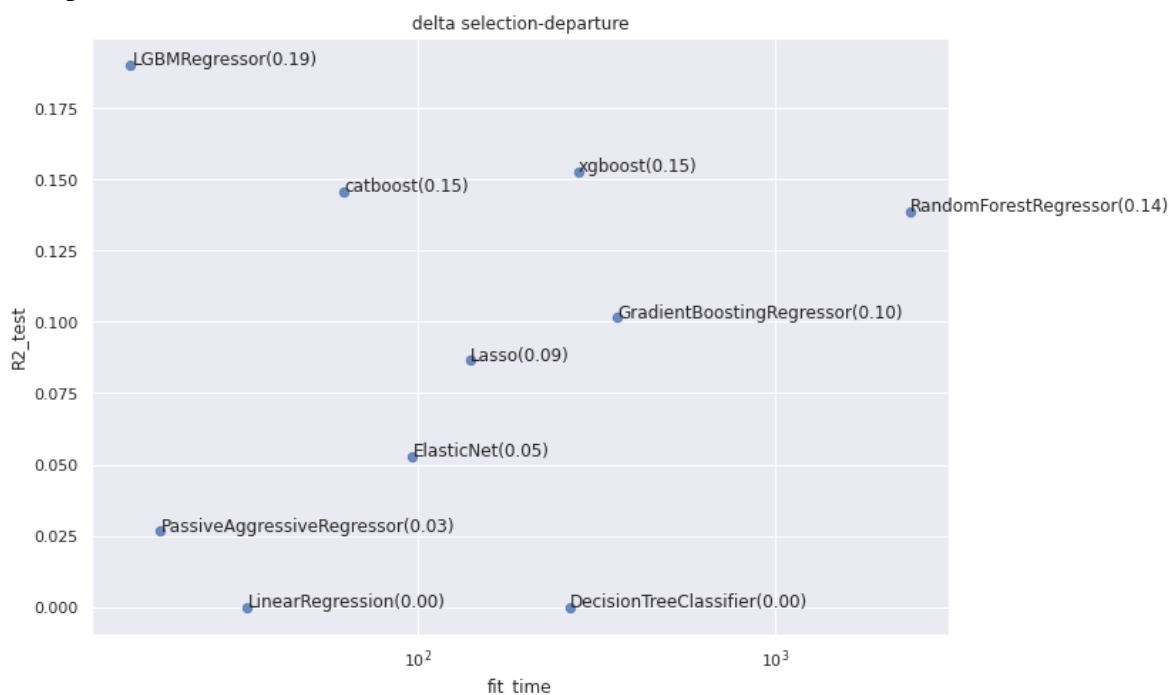
R-squared on test set with different estimators - Figure 4-2



The computation times vary a lot between models. The charts below show the R2-test and execution time in second for the different models.

Training time [second] and R-squared of different estimators - Figure 4-3

Y0 as response variable:



Y1 as response variable:



The gradient boosting models are the best ones in term of accuracy and speed.

5 Gradient Boosting with Catboost

After a fast comparison of different types of models, I decided to go on with **Catboost** for its potential ability in fighting overfitting and hyperparameters setting, especially for the loss function and variable encoding.

5.1 Catboost

Catboost is a Russian implementation of Gradient Boosting. Among its many nice features, some are particularly worth of attention for our problem.

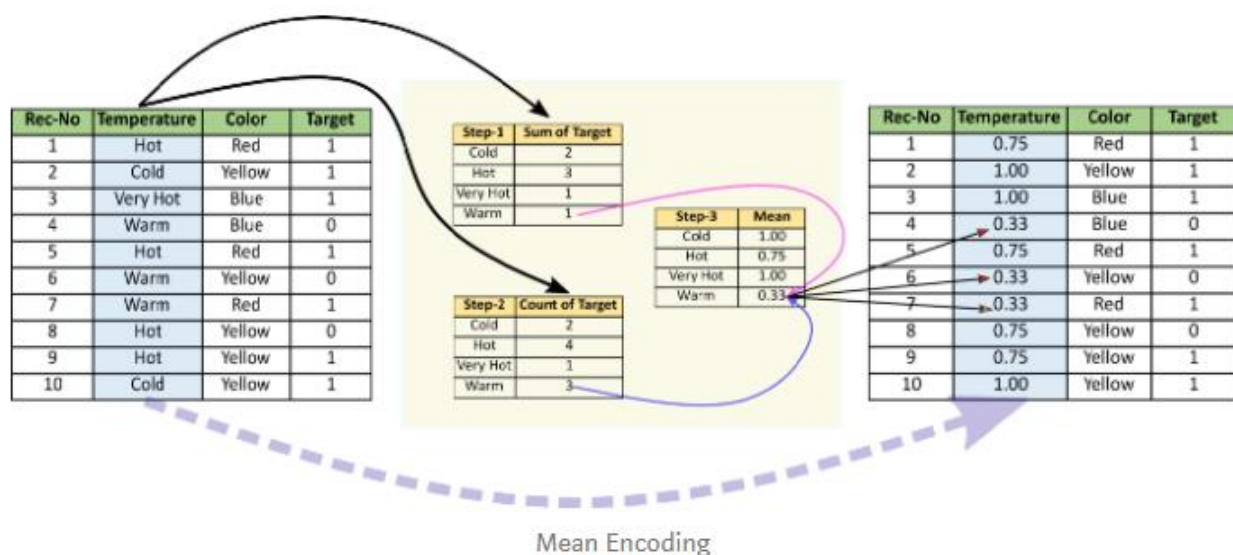
5.1.1 Ordered Boosting

A noticeable innovation of Catboost, in the goal of fighting overfitting, is the principle of “**ordered boosting**”. That influences the way how the residuals are computed at each iteration. The intuition is that, on every iteration of boosting, when we evaluate the residual of a datapoint, we must use a model that has not been trained with this datapoint. That means there are many more base models trained than in the classical gradient boosting. A strategy was deployed to control the computation cost: the number of models is $\log(\text{number_of_datapoint})$.

5.2 Categorical variable encoding

Catboost offers also a great convenience in encoding automatically categorical variables with **target-mean coding**: a category is encoded as the mean value of the target variable over the category.

Categorical variable – target encoding - Figure 5-1



Source: <https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02>

To avoid **target leakage**, the so-called **prediction shift**, the idea of “ordered boosting” is again applied. The target-mean of a category assigned to an example is not anymore, a constant evaluated only once on all the dataset. But it is evaluated on the datapoints before that example. Please refer to <https://towardsdatascience.com/https-medium-com-talperetz24-mastering-the-new-generation-of-gradient-boosting-db04062a7ea2> for more insight.

5.3 Strategy to overcome outliers

As already noticed, the response variables contain lots of extreme values. That corroborates the fact we observed relatively low R2-test values in the comparison of models given above. Whatever type of model is chosen, the R2-test for Y1 is under 0.4.

To overcome this difficulty, special settings are applied to the model. The goal is to construct a model with a minimum of bias and variance. That means **creating a maximum of trees but under the constraint of not falling into overfitting**.

The following sections illustrate the settings that resulted from this strategy.

5.3.1 Early stopping

Setting many trees (aka iterations) allows a gradient boosting model to gradually reduce its loss (gradient descent). But that is done with respect to the loss provided by the training dataset. The model could fall into overfitting the data, i.e. get excessively close to the given data including its noisy values and outliers.

One technic is to tell the model to stop when the model starts losing performance evaluated on a test dataset (also called the validation set).

In Catboost, we can set the following parameters:

- **early_stopping_rounds**: 200,
- **eval_metric**: "R2",
- **eval_set**: the dataset used for validation.

5.3.2 Hubert Loss function

Every iteration, i.e. new tree, consists of doing gradient descent of a loss function. But if the loss function was based on squared error such as MSE (Mean Squared Error) which is the usual default setting in regression, the model would be particularly vulnerable to noisy data. More the errors are extreme, more the gradient of the square loss is strong.

Therefore, instead of measuring the errors with a quadratic function, i.e. using the **L2-norm**:

$$SumOfSquaredError = L_2(Error) = \sum e_i^2 \text{ where } e_i = (\hat{y}_i - y_i)$$

One might think of **L1-norm** which is known to be resistant to outliers.

$$L_1(Error) = \sum |e_i|$$

But L2-norm has the advantage over the L1-norm of being smoother around zero and accelerating convergence above 1.0.

An ideal trade-off is to get a loss function that meets both the advantages of L1 and L2 norms, something that respond to the following intuitions:

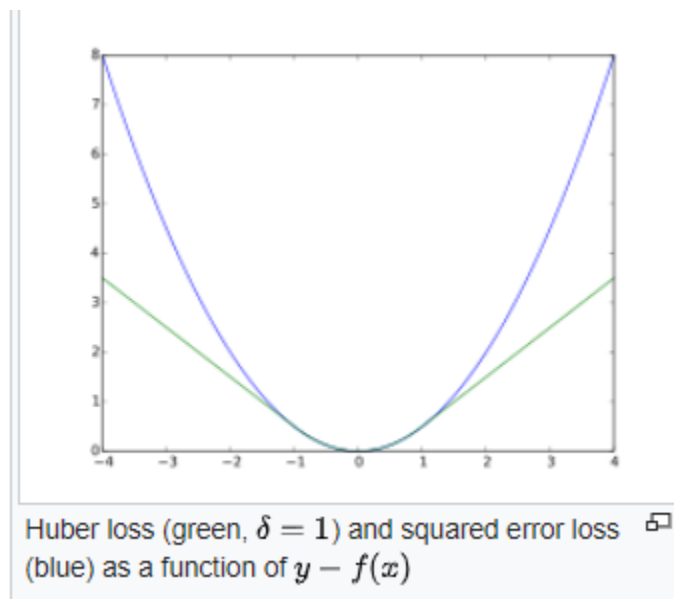
- Reduce the importance of high errors. For instance, if the model makes an error that exceeds 10 minutes, do not overreact. Go slowly....
- Keep a reasonable convergence speed when an error is under 10 minutes. I don't want to spend too much time waiting for the final model.
- Soften down the gradient descent where the errors get close to zero.

That loss function is named **Huber loss**:

$$L_{\delta}(y, \hat{y}) = \frac{1}{2} (y - \hat{y})^2 \quad \text{if } |y - \hat{y}| \leq \delta$$

$$= \delta |y - \hat{y}| - \frac{1}{2} \delta^2 \quad \text{if } |y - \hat{y}| > \delta$$

Huber loss function - Figure 5-2



Source: Wikipedia.

The Huber loss function is displayed on the chart above. We can see for high values, the squared error loss has a much sharper slope than the $y - f(x)$ function.

In Catboost, one can set the Huber function as loss function:

- `loss_function='Huber:delta=600'`

The hyperparameter delta is still to be tuned. After a few tries, the value of **600**, aka 10 minutes, turns out to be a good setting.

5.3.3 Tree depth

If we want many trees, we have to keep the base models as “weak learner” as possible (slightly better than the random guess), i.e. not too complex. Consequently, the tree depth must be sufficiently small (shallow trees). But it must also allow a minimum of splits to make sense.

The common value of **depth is 6**. Experiments proved it works fine.

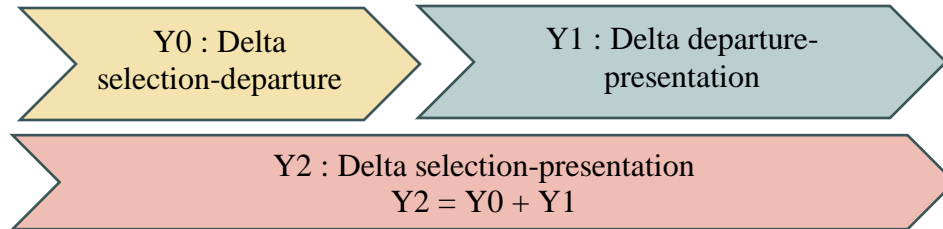
5.3.4 Learning rate

Extreme values are going to cause high gradients. To limit this effect, a common technic is to limit the learning rate to a small value. The price to pay is the slowdown in convergence, hence increasing the number of trees. But that is why I want. The more there are trees, the less bias and variance are. However, I don't want to spend days in waiting. A learning rate of 0.05 turns out to be good.

- **learning_rate=0.05**

5.4 Multitarget model – “nested” models

Let's remember there are three response variables that are linked together:



We can proceed in different ways:

1. **Create two independent models for Y0 and Y1.**

Y2 is just the sum of Y0 and Y1.

But doing so, we can notice the resulted of performance for Y0 is relatively poor. For instance, the table below shows the R2 metric for the resulted modeling:

Before using nested models - Table 5-1

Response variable	R2 on test set
Y0 (delta selection-departure)	0.200
Y1 (delta departure-presentation)	0.425
Y2 (delta selection-presentation)	0.434

The available features can't well explain Y0.

The alternative approach below tends to overcome this drawback.

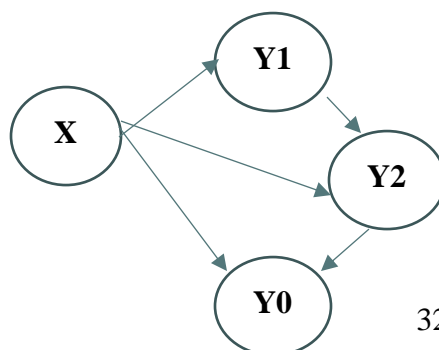
2. **Create a set of “nested” models:**

- Create the first model for $\hat{Y}_1 = f(\mathbf{X})$,
- Create the second model for **Y2** by reinjecting \hat{Y}_1 as an explainable variable, $\hat{Y}_2 = g(\mathbf{X}, \hat{Y}_1)$
- Create the third model for $\hat{Y}_0 = h(\mathbf{X}, \hat{Y}_1, \hat{Y}_2)$ by reinjecting \hat{Y}_1 and \hat{Y}_2 as explainable variables.

The intuition behind this is to create 3 models which are “linked” together:

- The available features should not be able to well explain Y0.
- So, rather than predicting Y0 directly from X, it could be easier to predict first Y2 (= Y0 + Y1) by reusing \hat{Y}_1 and X as input.
- Finally, we can “predict” Y0 by reusing X, \hat{Y}_1 and \hat{Y}_2 as inputs.

The idea could be presented by the following graphical model:



The results are: *R2 with Nested models - Table 5-2*

Response variable	R2 on test set
Y0 (delta selection-departure)	0.234
Y1 (delta departure-presentation)	0.425
Y2(delta selection-presentation)	0.429

The 2nd solution seems to improve slightly the performance for Y0 (Before using nested models - Table 5-1).

5.5 Train models

Using the settings described above, three Catboost regression models are trained, one per response variable (target). The results are below - Table 5-3

Target	Fit time [hour]	Number of trees (best iteration)	R2 / Test
Y0 (delta selection-departure)	0.53	200	0.234
Y1 (delta departure-presentation)	2.10	13673	0.425
Y2(delta selection-presentation)	0.47	3056	0.429

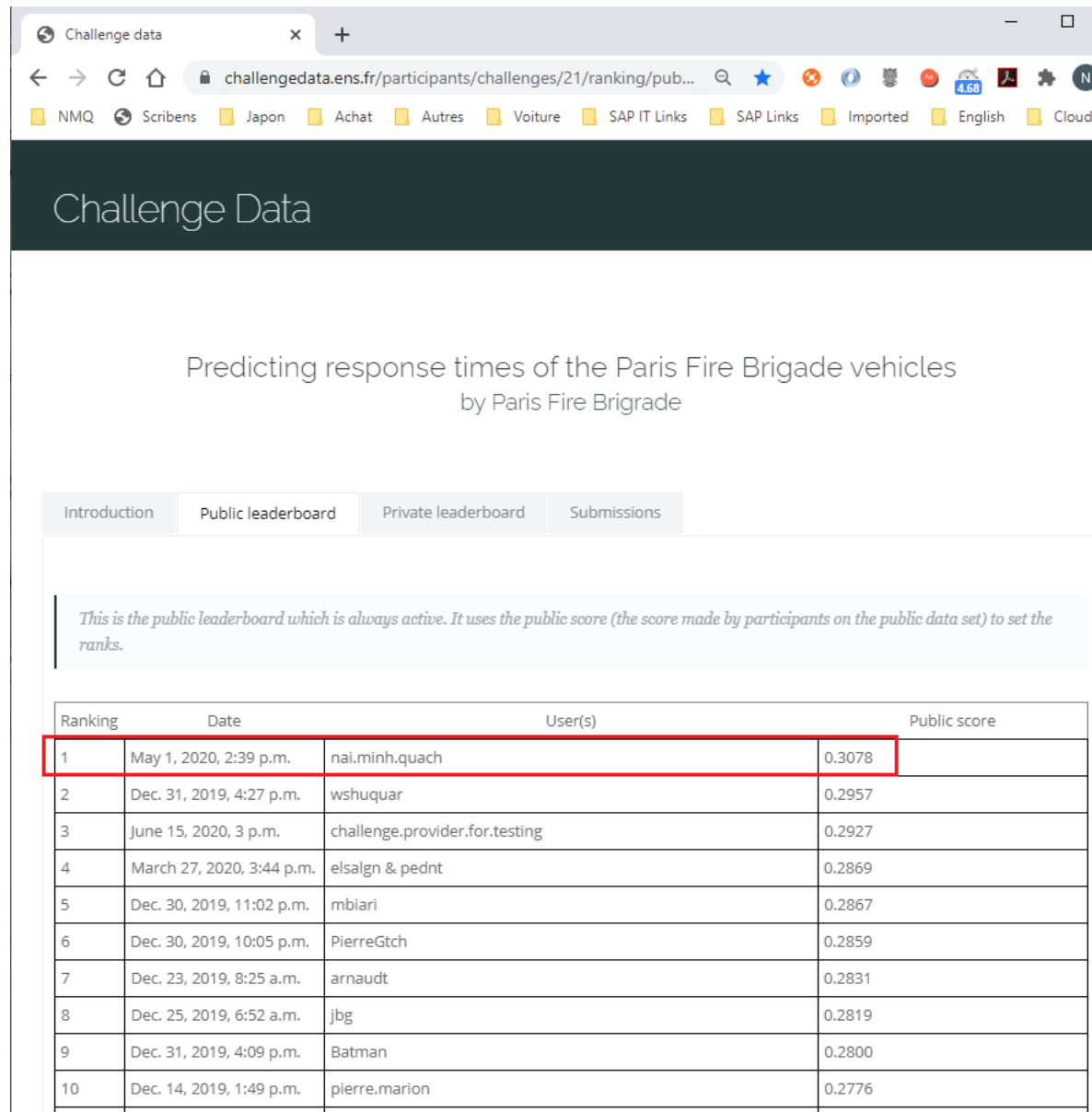
We can notice the large number of trees in the model for Y1: **13673**. It results from the intention of minimizing the bias and the variance of the model. But that comes with a price: more than 2 hours are needed. The performance in R2-test is 0.425 for Y1. It does outperform all those we had seen in the first comparison (the best one was 0.39). We will analyze in the next section if all this job is worth it.

6 Results with Catboost

6.1 Leaderboard

The submission has been ranked **first out of 196** on the [Leaderboard](#) which was opened since January of 2020.

Public Leader board – screenshot on October 04, 2020 (ranking among 198 submissions) Table 6-1



The screenshot shows a web browser window with the URL challengedata.ens.fr/participants/challenges/21/ranking/pub.... The page title is "Challenge Data". The challenge name is "Predicting response times of the Paris Fire Brigade vehicles by Paris Fire Brigade". The page has tabs for "Introduction", "Public leaderboard", "Private leaderboard", and "Submissions". A note states: "This is the public leaderboard which is always active. It uses the public score (the score made by participants on the public data set) to set the ranks." Below this is a table with the following data:

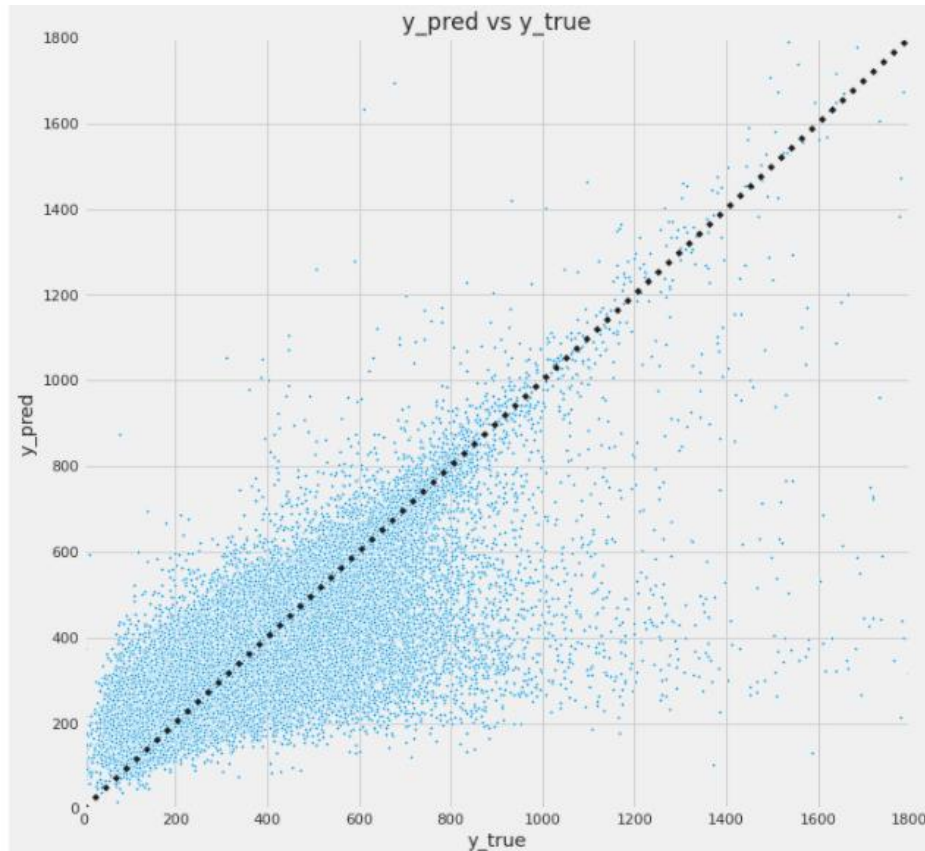
Ranking	Date	User(s)	Public score
1	May 1, 2020, 2:39 p.m.	nai.minh.quach	0.3078
2	Dec. 31, 2019, 4:27 p.m.	wshuquar	0.2957
3	June 15, 2020, 3 p.m.	challenge.provider.for.testing	0.2927
4	March 27, 2020, 3:44 p.m.	elsalgn & pednt	0.2869
5	Dec. 30, 2019, 11:02 p.m.	mbiari	0.2867
6	Dec. 30, 2019, 10:05 p.m.	PierreGtch	0.2859
7	Dec. 23, 2019, 8:25 a.m.	arnaudt	0.2831
8	Dec. 25, 2019, 6:52 a.m.	jbg	0.2819
9	Dec. 31, 2019, 4:09 p.m.	Batman	0.2800
10	Dec. 14, 2019, 1:49 p.m.	pierre.marion	0.2776

One can notice that the achieved performance, **$R^2 = 0.3078$** , is still very far from 1.0. In another word, only 30% of the variance of the prediction can be explained. Later in the document, we will see how to interpret this number.

6.2 General view on the model performance

A first appreciation could be done by plotting the predictions (y_{pred}) against the actual values (y_{true}). The following chart is about Y1 (“Delta departure-presentation”) with the test dataset:

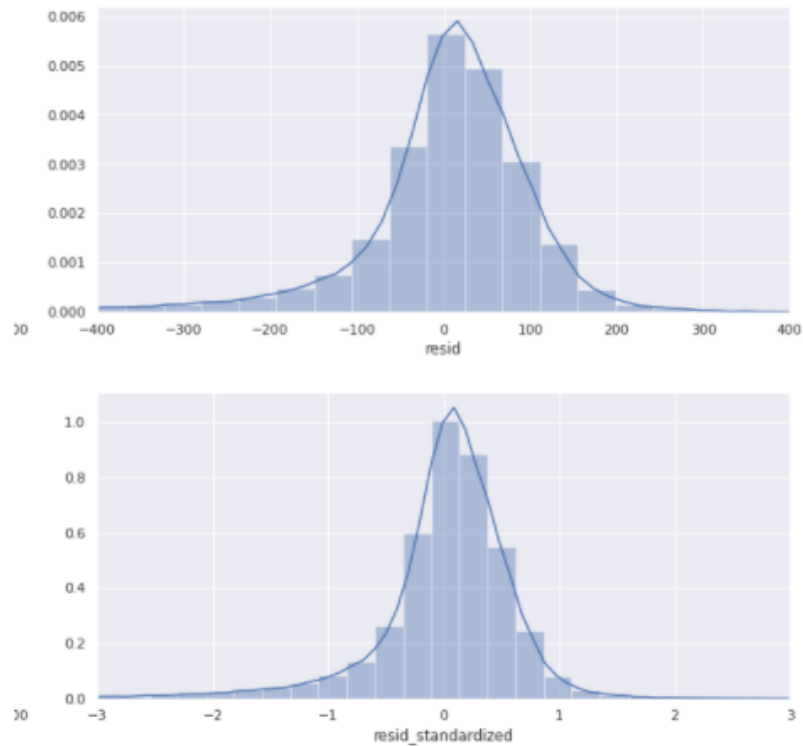
Predicted Y1 (“Delta departure-presentation”) vs Actual - Figure 6-1



At the first glance, it looks like the predictions are very dispersed far from the diagonal. Many predictions are extremely under-estimated. But in fact, a large majority of the datapoints are well estimated.

The charts below show respectively the distributions of the residuals ($\epsilon_i = \hat{y}_i - y_i$) and the standardized residuals ($\frac{\epsilon_i}{\text{std}(\epsilon)}$).

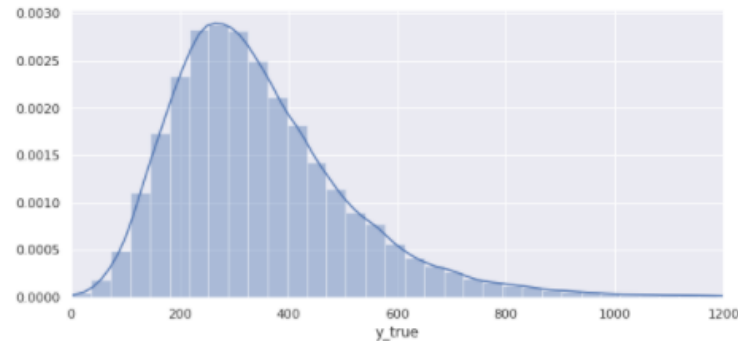
Y1 Standardized Residuals distribution - Figure 6-2



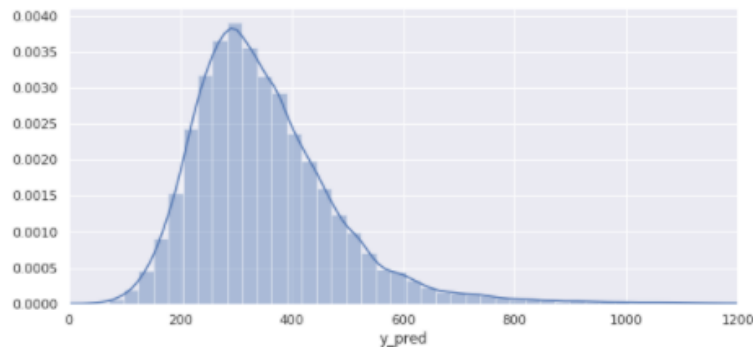
Most of the residuals in absolute value are below 300s (5 minutes), i.e. under 3 times the standard deviation. However, this distribution is skewed: the predictions tend to under-estimate for very few examples.

If we compare the distributions of the true values and the predictions, we can notice some divergence: the model tends to concentrate more mass around the mode value (~300s). It did not manage to predict the long stretched right tail.

Y1 actual values distribution - Figure 6-3



Y1 predicted values distribution



6.3 Analyzing the R2 metric

R2 is a measurement of goodness of fit:

$$R^2 = 1 - \frac{\text{Unexplained variance}}{\text{Total variance}} = 1 - \frac{\sum(\hat{y}_i - y_i)^2}{\sum(y_i - \bar{y})^2}$$

We try now to understand how this number is composed.

By taking inspiration from the [ROC curves](#) or AUC (Area Under Curve) in binary classification, we can create a similar chart for our regression model. The idea is to see how R2 is distributed over the amplitude of residuals: what are the subsets of the population that are the most responsible for R2 drop?

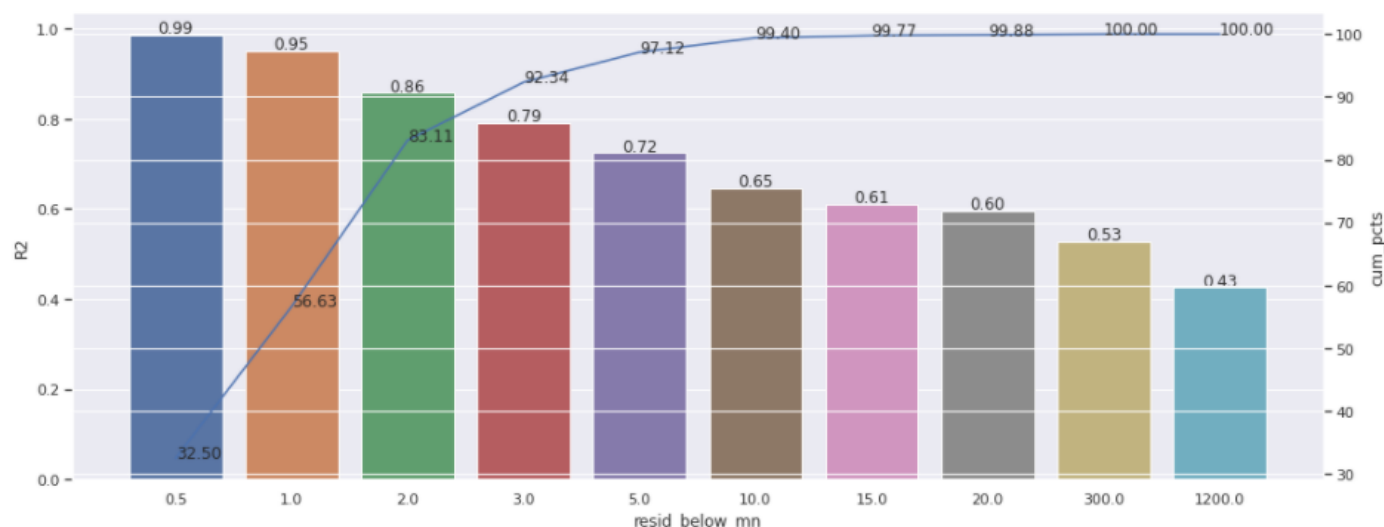
Concretely, we apply first the model on the validation set, i.e. 1/3 of the train data that was not used for the model training. The resulting predictions are then sorted by the absolute value of the residuals in ascending order. We then create a cumulative chart with:

- X-axis: the higher bound of absolute residuals.
- Y-axis: the R2 metric measured on the subset of data which residuals are under the X-value.

We can then display a chart showing how R2 evolves when we evaluated the test dataset by accumulating the samples ordered by their residual values.

The chart below is about Y1 (“delta departure-presentation”).

R-squared breakdown per residual upper bound - Figure 6-4



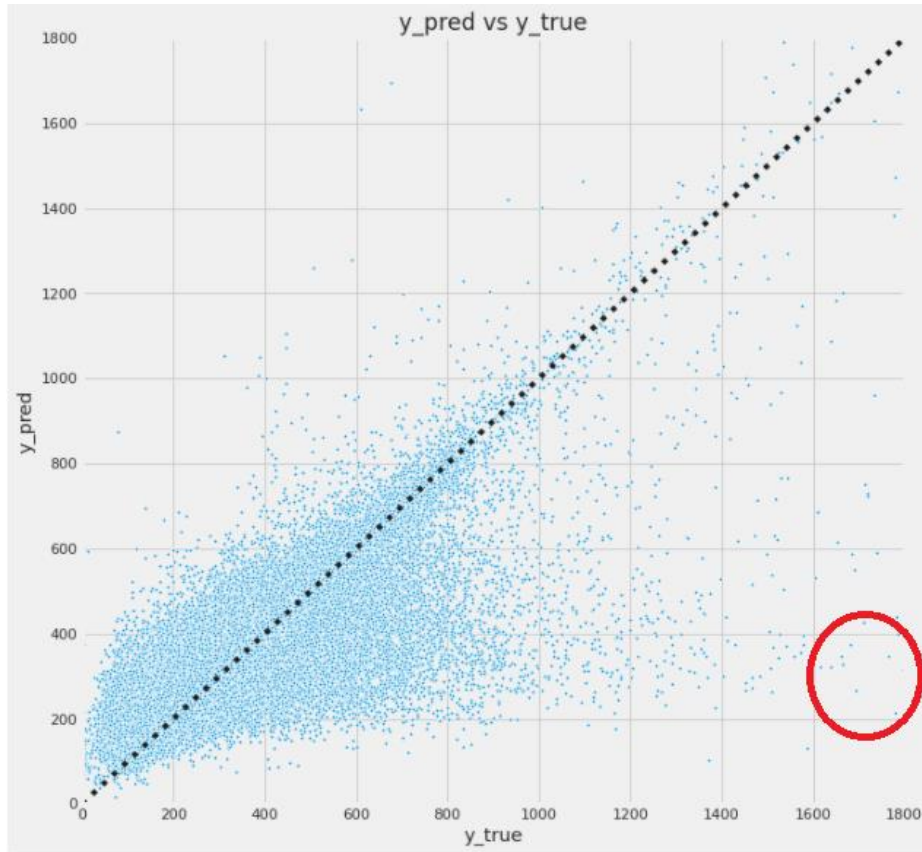
The colored bars represent the R2 values by cumulative bins of test data ordered by the absolute value of residuals. For example, the first bar (blue) indicated the R2 on the test set if we only took samples having residuals under 0.5 minutes. The 2nd bar (orange) indicated the R2 on the test set if we only took samples having residuals under 1 minute.

The blue line indicates the corresponding cumulative percentage in the dataset. So, for example, if we take only the samples having residuals below 2 minutes (green bar), they represent 83.11% of the test dataset. That corresponds to R2=0.86.

That means the global R2 (0.42) is essentially degraded by the very few last portions of data. **The last 8% of data (after the green bar) is responsible for decreasing the R2 by 50%** (from 0.86 to 0.41). There are few outliers in the dataset responsible for the high discrepancy.

Another way to see these outliers is to plot the true response variable (X-axis) against the prediction (Y-axis). We can see, even though the majority of points lie on the proximity of the diagonal line, few ones were far in the bottom-right side. For instance, in the red circle, the model has predicted around 300s, but the true values range around 1700 seconds.

Y1, actual vs predicted values - Figure 6-5



In the following section, we try to shed light on those outliers.

6.4 Outliers analysis

Let's see some worse predictions. Given the GPS coordinates, we can use Google Maps URL to estimate the path and time.

6.4.1 Example1: 6 hours for 14 km

Line #208599 in the training dataset. **6 hours were needed for 14 km.** This is just unexplainable. The model predicted 17minutes, Google-maps 20 minutes.

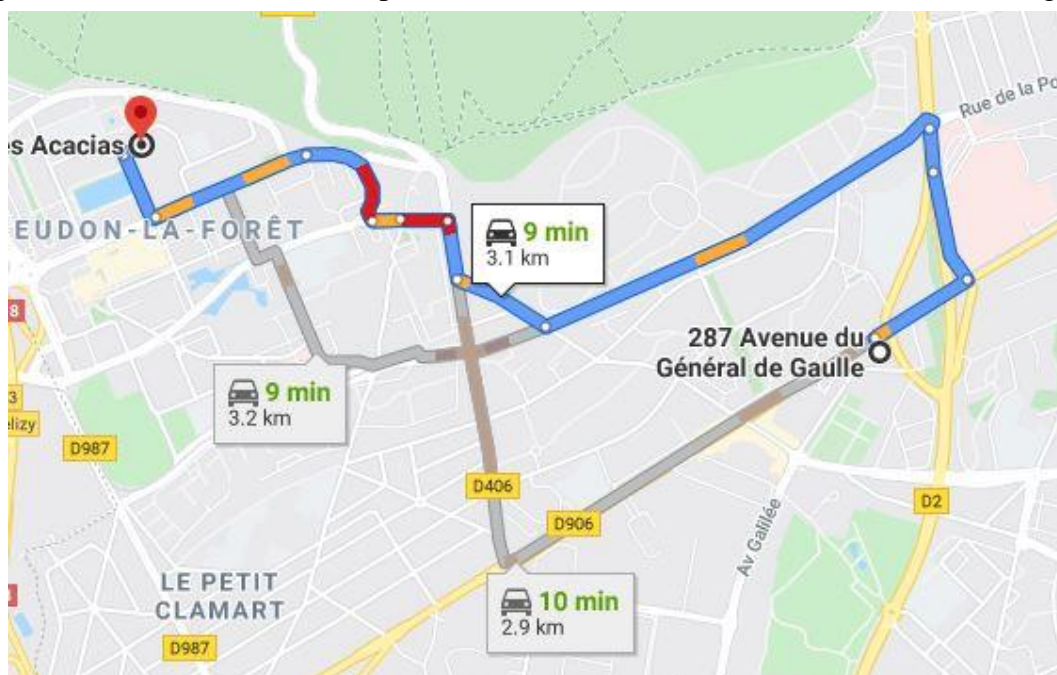
Unexplainable extreme value in response time (**6 hours were needed for 14 km**) - Figure 6-6



6.4.2 Example2: 2.5 hours for 3.1 km

Line #183804. **2.5 hours was needed for 3.1 km.** The model predicted 10 minutes, Google-maps 9 minutes.

Unexplainable extreme value in response time (**2.5 hours was needed for 3.1 km**) - Figure 6-7

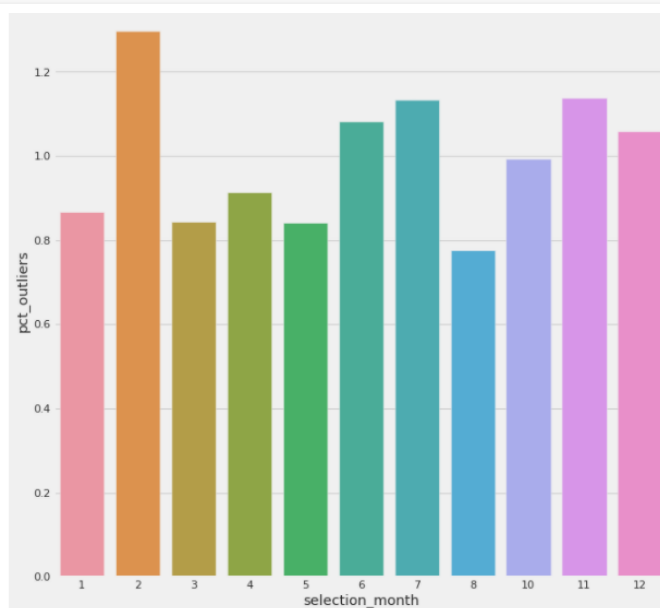


6.4.3 How are outliers distributed in time?

If we defined outliers as the datapoints which residual absolute value is 3 times above the standard deviation (i.e. 481 seconds). It would be helpful to see how they are distributed in time. The table and the chart below show how those outliers are distributed per month.

Outliers distribution over time (month) - Figure 6-8

	selection_month	nb_outliers	nb_total	pct_outliers
0	1	58	6691	0.867
1	2	78	6013	1.297
2	3	57	6764	0.843
3	4	57	6237	0.914
4	5	57	6781	0.841
5	6	73	6750	1.081
6	7	80	7055	1.134
7	8	44	5668	0.776
8	10	67	6750	0.993
9	11	76	6685	1.137
10	12	74	6988	1.059



Surprisingly, we can notice outliers occur on a regular frequency: about one 1% per month. There should be some “regular” events that are not captured in the dataset.

6.4.4 How are outliers distributed geographically?

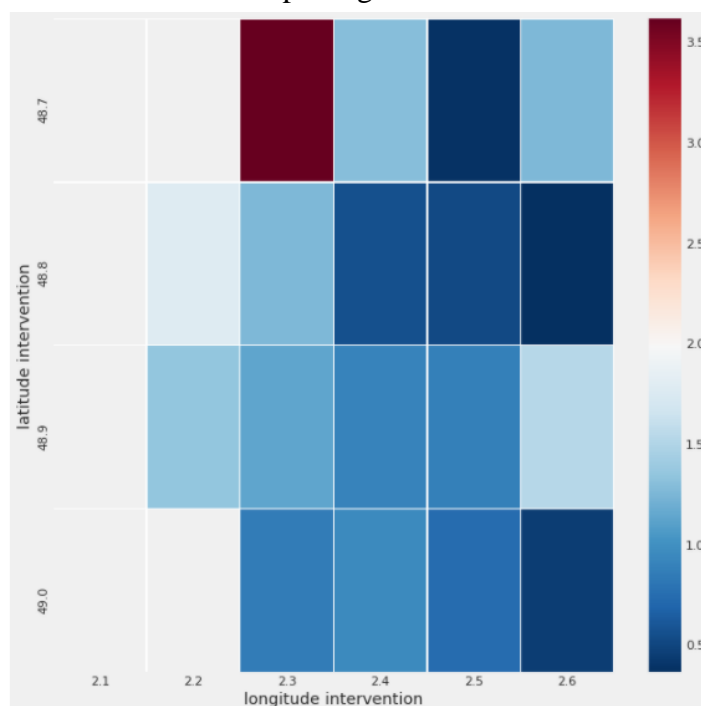
The number of outliers per area (per tile of 0.1 longitude x 0.1 latitude) - Table 6-2

longitude intervention	2.200	2.300	2.400	2.500	2.600
latitude intervention					
48.700	nan	11.000	8.000	2.000	1.000
48.800	40.000	134.000	54.000	21.000	3.000
48.900	47.000	170.000	151.000	53.000	11.000
49.000	nan	5.000	6.000	3.000	1.000

The proportion of outliers per area (percentage per tile of 0.1 longitude x 0.1 latitude):

longitude intervention	2.100	2.200	2.300	2.400	2.500	2.600
latitude intervention						
48.700	nan	nan	3.618	1.299	0.380	1.266
48.800	nan	1.784	1.255	0.567	0.526	0.365
48.900	nan	1.354	1.135	0.909	0.885	1.532
49.000	nan	nan	0.858	0.951	0.743	0.448

Outliers distribution per segment of GPS coordinates

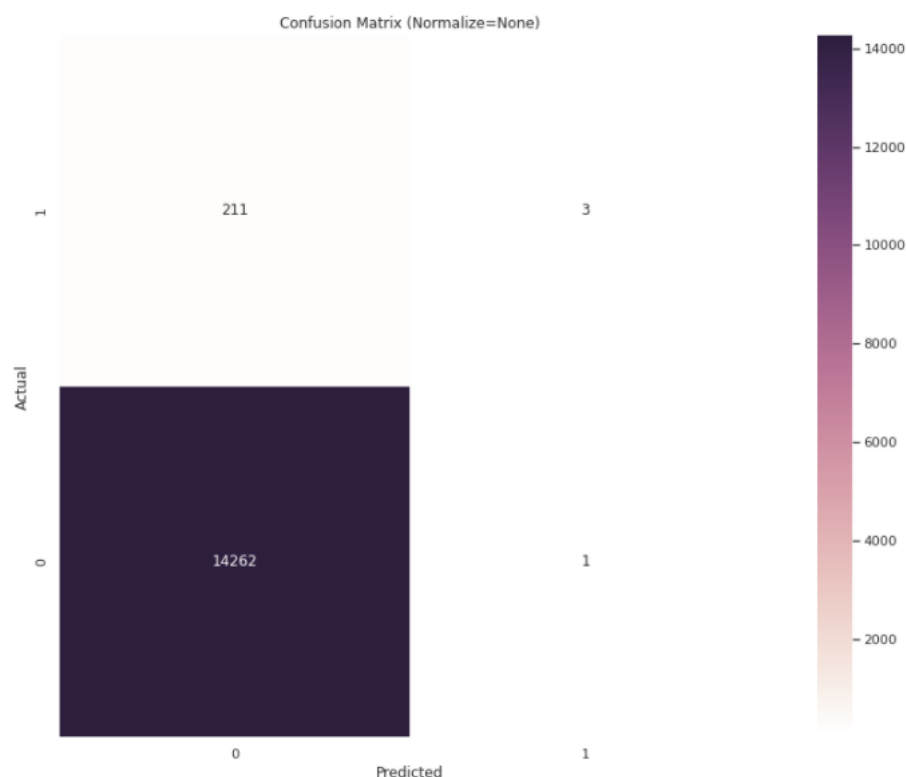


The outliers are relatively well dispatched over Ile de France, except a small concentration in Paris North-West: 3.618% versus 1.10% on average. But 3.6% is still a small percentage. We cannot use this information to characterize outliers.

6.4.5 Are outliers detectable by machine learning?

One may ask: given the features, can we predict the outliers using a classification model? Using **Catboost as a classifier** (to detect outliers), we get the following **confusion matrix**:

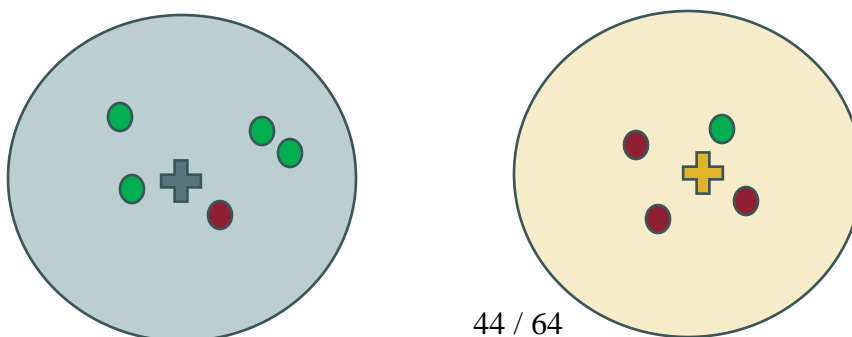
Confusion Matrix for outlier's detection - Table 6-3



The classifier performs very poorly in accuracy: **only 3 out of 211 positive cases are detected**.

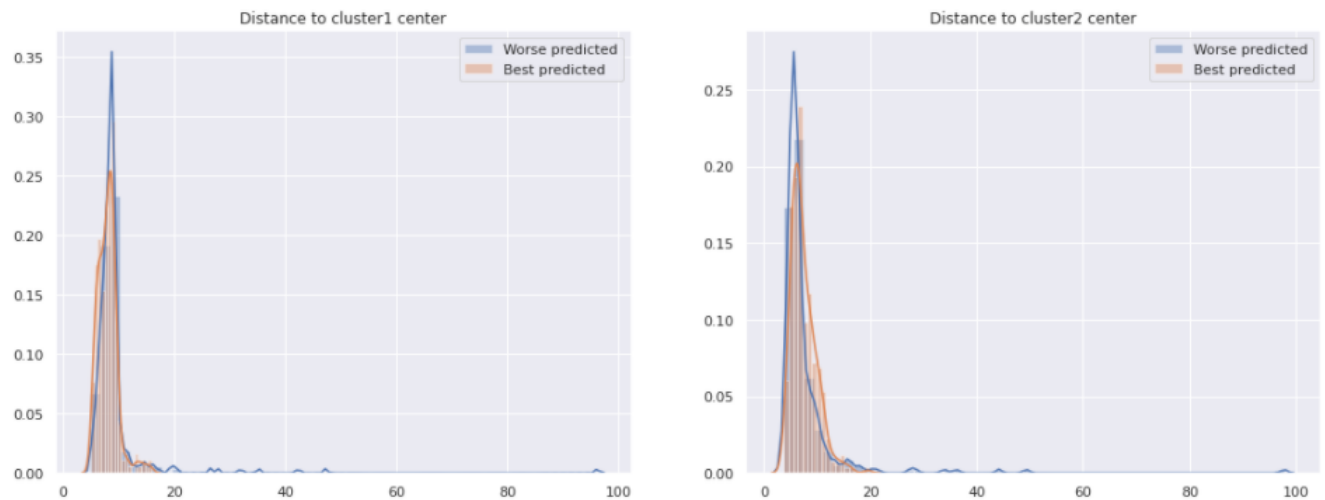
We can also use a **clustering** algorithm to check whether the outliers are distinguishable from the inlier points. To do this, we can study the distance of those outliers from the centroids of the dataset. Using K-Means, we fit the model with 2 clusters. We can then compare the distribution of distances to the 2 centroids, between inliers and outliers. In the example below, let's suppose the outliers are the red points, the inlier the green points. If they are clustered differently, we will their distances to the centroids (crosses) are distributed differently.

Using clustering for outliers detection - Figure 6-9



The charts below display the effective distributions. We can notice the inliers and the outliers do have almost the same distributions of their distances to the two centroids.

Distances distribution to clusters' centroids - Figure 6-10



That leads us to think the outliers and the inliers are not distinguishable in the feature space: if one tried to compare an inlier X_i and an outlier X_o in the features space, no difference could be made.

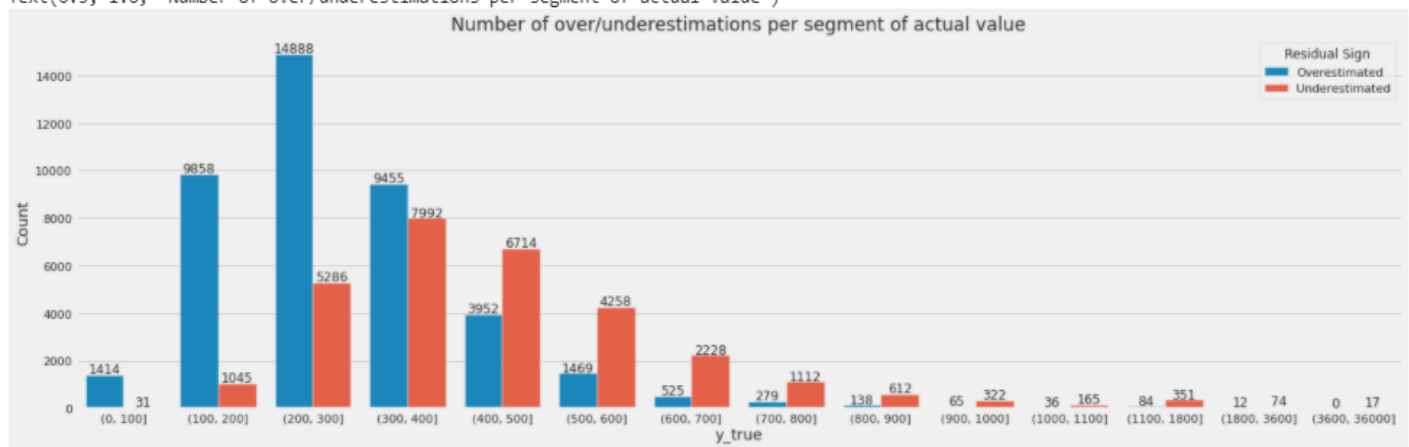
6.5 Understand the model

We can wonder:

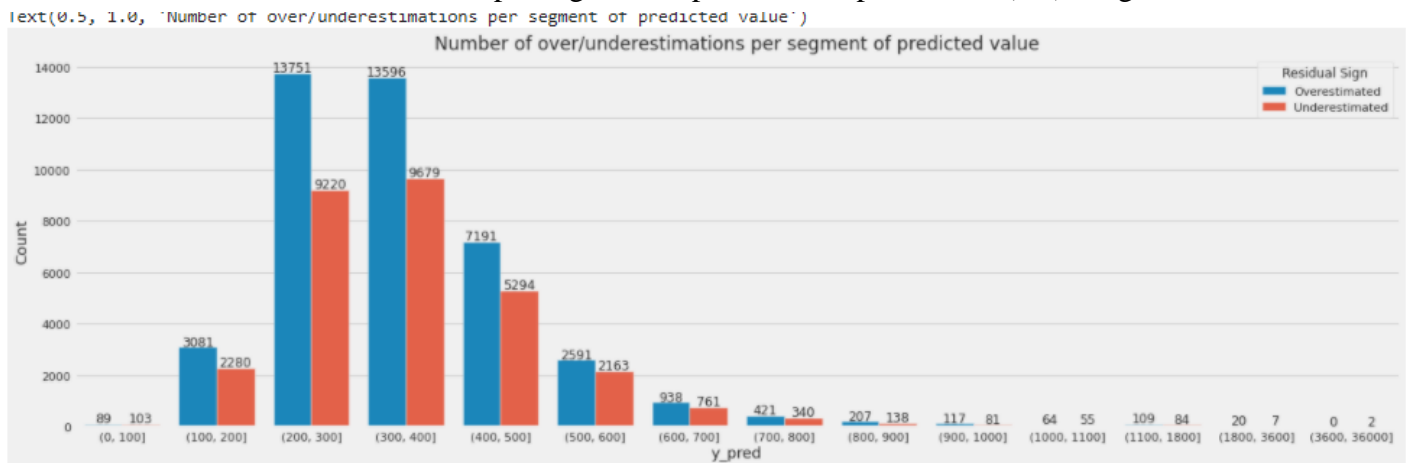
- Did the model mostly underestimate or overestimate the response times?
Based on the chart [actual time vs prediction](#), one may wonder if the model tends towards underestimating the response variable.
- Did the model fail to differentiate the outliers from the inliers?

We can count the number of underestimations and overestimations, aka looking at the sign of the residuals, per segment of value. We can do this per actual time (y_{true}) or per predicted time (y_{pred}):

Number of over/under estimations per segment of actual response time (Y1) - Figure 6-11



Number of over/under estimations per segment of predicted response time (Y1) - Figure 6-12



Interestingly, these two charts said the opposite things:

- The first chart said, for a given actual value (y_{true}), when it is above 400 seconds, the model tends to underestimate (blue bars are lower after (400,500]).
- The second chart said, indeed, for a given predicted value (y_{pred}), the model mostly tries to overestimate (blue bars are always higher).

One explanation could be: the model has learned that it should deal with high response times during the training. It tried then to frequently overestimate. However, the dispersion is so extreme, the model has to make the best trade-off to keep the loss minimum on average.

Another question is to check whether the model can, in some way, make the difference about outliers from inliers. For that, we can compare the way the features contribute between two subsets of data:

- A subset of datapoints having the lowest residuals: the **best-predicted** points.
- And another with the highest residuals: the **best-predicted** points.

We look at the **SHAP value** of each these two sets:

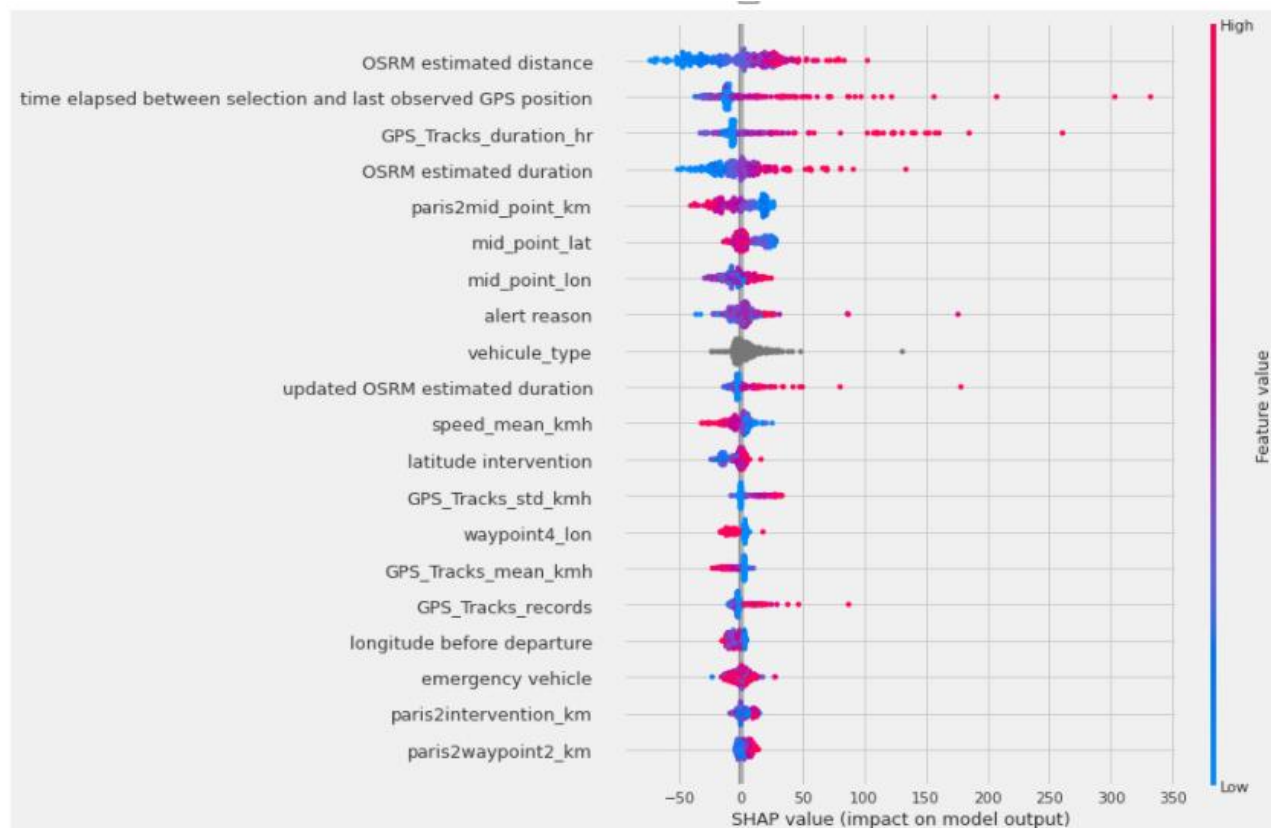
- Best predicted

- Worse predicted.

Shap values of the 300-**best** predicted subset - Table 6-4

	count	mean	std	min	25%	50%	75%	max
GPS_Tracks_duration_hr	300.000	19.432	34.746	0.258	6.325	7.706	13.951	260.800
time elapsed between selection and last observed GPS position	300.000	21.983	33.090	0.138	10.261	12.298	24.430	332.165
OSRM estimated distance	300.000	26.090	18.822	0.111	11.857	23.325	36.451	102.660
OSRM estimated duration	300.000	16.878	17.072	0.064	5.802	12.140	22.156	133.780
alert reason	300.000	8.151	13.165	0.016	2.936	5.561	9.003	176.019
updated OSRM estimated duration	300.000	6.668	12.721	0.021	2.338	3.369	7.966	178.395
vehicule_type	300.000	7.899	10.389	0.034	2.548	5.122	9.432	130.301
GPS_Tracks_std_kmh	300.000	6.050	9.728	0.003	0.216	0.690	6.781	33.347
mid_point_lat	300.000	9.129	9.135	0.025	2.049	4.253	18.308	28.440
paris2mid_point_km	300.000	14.068	8.215	0.063	6.255	16.213	20.134	40.668

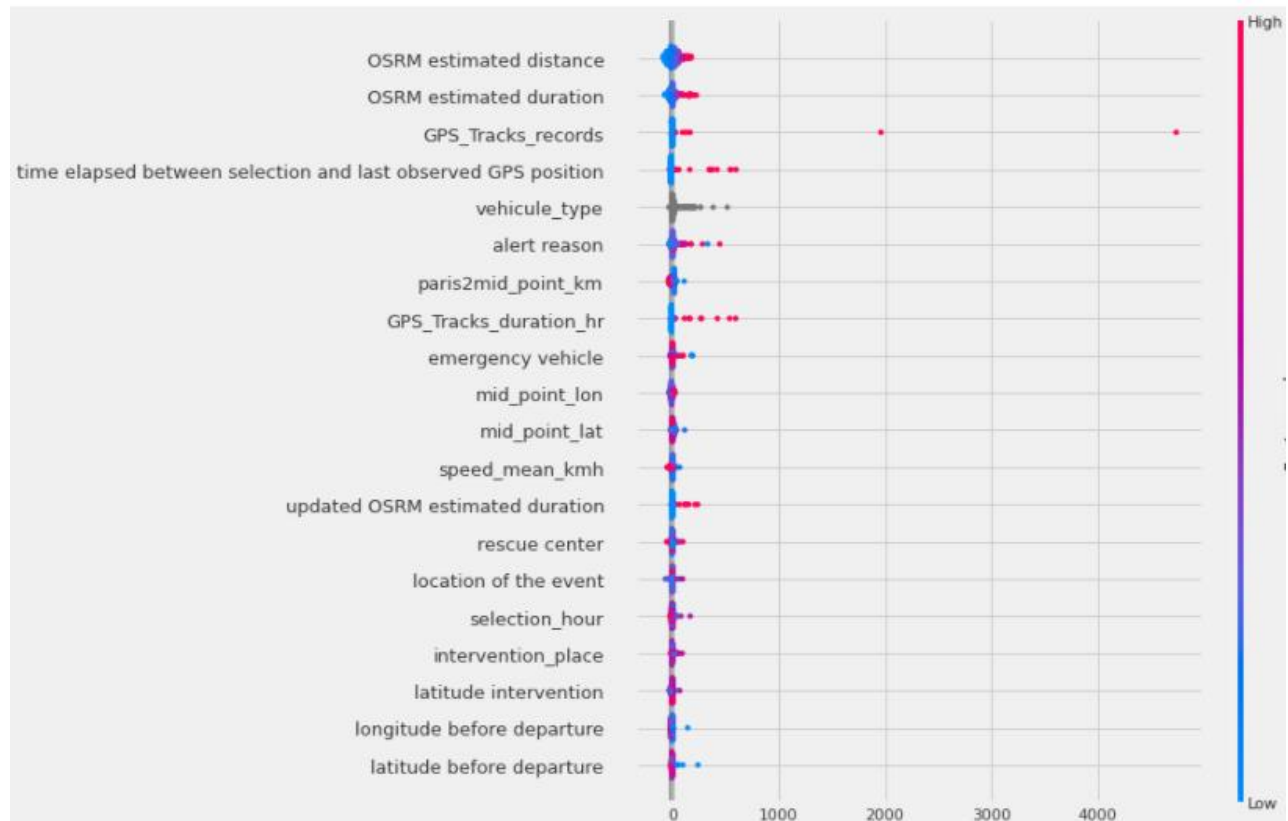
Shap values of the 300-best predicted subset – summary plot - Figure 6-13



Shap values of the 300-worse predicted subset – summary plot - Table 6-5

	count	mean	std	min	25%	50%	75%	max
GPS_Tracks_records	300.000	26.133	295.569	0.125	1.651	2.264	2.689	4735.127
time elapsed between selection and last observed GPS position	300.000	23.137	72.442	0.173	9.589	10.991	12.205	599.397
GPS_Tracks_duration_hr	300.000	15.156	57.404	0.481	6.031	6.990	7.644	595.129
vehicule_type	300.000	22.676	55.726	0.007	2.937	5.508	11.170	515.524
alert reason	300.000	19.287	42.169	0.105	4.577	8.770	16.498	446.657
OSRM estimated duration	300.000	30.347	40.798	0.224	7.266	16.151	33.459	225.989
OSRM estimated distance	300.000	42.635	33.893	0.742	19.513	35.354	55.426	176.785
updated OSRM estimated duration	300.000	7.277	25.004	0.112	2.043	2.654	3.573	239.547
emergency vehicle	300.000	9.552	21.366	0.038	2.109	4.616	9.339	192.191
latitude before departure	300.000	5.907	15.826	0.008	1.986	3.565	6.166	241.008

Shap values of the 300-worse predicted subset – summary plot - Figure 6-14



The answer is yes: the model did behave differently for inliers and outliers. The feature importance is not the same. In the “worse predicted” case, the standard deviations of the Shap value of the top features are much higher, though the mean remains in the same order of magnitude. The model tends to overfit for outliers. That can be also a sign of the variability of the error variance (heteroskedasticity). The errors do vary over different subsets of the population.

7 “All models are wrong”

Until here, we have always considered pointwise estimation: we are interested in one final model with one instance of estimated parameters. But no matter how optimal they look like, there is always uncertainty. In the section “Models comparison”, we had already seen different types of model tell different predictions. But, even for the same type of model, different settings, different samples of the same population, also can tell different things.

For instance, if we did a K-fold validation, we would see the model changes on every fold iteration. For instance, the following table shows the change of feature importance for 5 folds:

Feature importances – variations in 5 folds estimation - Table 7-1

	Feature Id	Importances	Feature Id	Importances	Feature Id	Importances	Feature Id	Importances	Feature Id	Importances
0	updated OSRM estimated duration	23.692631	time elapsed between selection and last observ...	15.672128	OSRM estimated distance	16.678117	OSRM estimated distance	14.831754	OSRM estimated distance	15.383812
1	OSRM estimated distance	14.647387	OSRM estimated distance	14.940610	GPS_Tracks_duration_hr	15.113871	GPS_Tracks_duration_hr	14.539872	time elapsed between selection and last observ...	15.066833
2	GPS_Tracks_duration_hr	10.036896	GPS_Tracks_duration_hr	12.883960	time elapsed between selection and last observ...	14.383991	time elapsed between selection and last observ...	14.427717	GPS_Tracks_duration_hr	13.415484
3	OSRM estimated duration	8.349626	GPS_Tracks_records	12.837318	OSRM estimated duration	10.985448	OSRM estimated duration	11.825836	OSRM estimated duration	11.140396
4	time elapsed between selection and last observ...	7.531315	OSRM estimated duration	11.869980	GPS_Tracks_records	9.772263	GPS_Tracks_records	9.425469	GPS_Tracks_records	10.502910
5	GPS_Tracks_records	7.142200	vehicle_type	2.194288	updated OSRM estimated duration	2.556363	updated OSRM estimated duration	2.704970	updated OSRM estimated duration	2.636033

[George Edward Pelham Box](#), a great statistician, stated “[all models are wrong](#)”. Therefore, as our model has the right to be wrong, we like then to know how much the model thinks it is wrong. Equivalently, how much confidence it thinks to have about its prediction.

Two approaches are studied,

- Estimate a **confidence interval**, more exactly pointwise-estimate the two ends of a confidence interval.

For instance, for a predicted value, \hat{y} , estimate the low end, \hat{y}_L , and the high end, \hat{y}_H , such that the model thinks there is 90% of chance that \hat{y} lies in $[\hat{y}_L, \hat{y}_H]$.

- Create a **probabilistic model** by [Bayesian inference](#).

We can construct a **Bayesian** model which can tell us, not only how the model parameters should be distributed (posterior distribution of the parameters, $P(\theta | X, Y)$), but also how prediction is distributed ([posterior predictive distribution](#)).

But it allows us to go even further by bringing tremendous flexibility in the modeling. This is particularly interesting in dealing with outliers.

Our goal now is no longer to improve our pointwise estimation (i.e improve R2 metric) but to get more insight about the model’s uncertainty and how outliers can be dealt with.

7.1 Predicting Confidence Interval with quantile-loss

It is known that by minimizing the **square loss**, a model equivalently seeks to estimate the **expectation** of the response variable conditioned on the predictor variable:

$$\hat{y} = E[y | X = x]$$

We can generalize this concept by setting a special **loss function** so that the model will estimate, at this time, not the expectation, but a **quantile** (or percentile). For instance, we can build a model for the 90-percentile of the response variable, i.e.

$$\text{Estimate a value } \hat{y} \text{ such that } \text{Proba}(Y \leq \hat{y} | x) = 0.9$$

This loss function is precisely the **quantile loss**:

$$L_q(\hat{y}, y) = (q - 1) \sum_{i=y_i < \hat{y}_i} |y_i - \hat{y}_i| + q \sum_{i=y_i \geq \hat{y}_i} |y_i - \hat{y}_i| \text{ where } q \in]0, 1[$$

The model is encouraged to further minimize overestimates with respect to a limit provided by the hyperparameter **q**. In another word, we will tell the model to care more about finding a q-percentile of the response variable. This function is available in Catboost.

For a given response variable, for instance, Y1 (“Delta departure-presentation”), supposing we want to estimate the 90% confidence interval, we can create 3 models with 3 quantile-loss functions:

- Quantile-loss **q=0.05** for the model that will predict the **lower end** of the **90%-confidence interval**,
- Quantile-loss **q=0.5** for the **median**,
- Quantile-loss **q=0.95** for the **upper end**.

For the response variable Y1 (“Delta departure-presentation”), the table below displays the statistics of the predicted values, respectively, the actual value (y_true), the estimated lower end confidence interval (y_predL), the higher end (y_predH), the median (y_predM) and the interval length (= y_predH - y_predL):

Y1 prediction and confidence interval - Table 7-2

	y_true	y_predL	y_predM	y_predH	interval_len
count	72382.000000	72382.000000	72382.000000	72382.000000	72382.000000
mean	356.130944	221.419976	332.624245	549.629841	328.209865
std	235.369420	124.458969	143.310792	182.716174	153.262930
min	1.000000	1.553758	20.656753	164.813022	-135.890145
25%	231.000000	142.726196	240.507146	450.798182	196.570864
50%	319.000000	193.508082	307.387199	534.525047	361.640384
75%	433.000000	264.989992	393.852250	633.562879	410.791174
max	22722.000000	1985.041754	4626.830650	5350.794478	3735.585117

Our model provides an estimated **95%-confidence interval equal to 328s (~5mn) on average**.

This information could be helpful to reassure a caller in distress. It can also reveal “abnormal” situations when an extreme level of uncertainty is estimated. For instance, when the confidence interval was estimated over a 1-hour amplitude, there should be something wrong...

7.2 Bayesian Model with MCMC

The pointwise estimations we have done so far consist of a **frequentist** approach that treats the outputs (model parameters or predictions) as fixed numbers. Even though with the quantile-loss, we can evaluate a confidence interval, it is still a point-estimation. As a result, we are always a little bit hungry as how to figure out the notion of **probability**:

- What is the posterior probability of the model parameters given the observations and our prior beliefs?
- What is the posterior probability of the predictions?

This is where the **Bayesian** approach comes into play: Bayesian inference. The principle lies simply in the **Bayes' theorem**:

$$Posterior = \frac{Likelihood * Prior}{Marginal Likelihood}$$

So, we just need to know how to put together the three components, prior, likelihood, and marginal likelihood, the posterior can be computed. This is the art of “[probabilistic programming](#)”. This paradigm is experiencing rapid advances in the recent years. Many libraries such as Stan, PyMC3, and Tensorflow Probability are available. I will use [PyMC3](#) for our regression problem. As its name suggests, PyMC3 mainly uses Markov Chain Monté Carlo ([MCMC](#)) to fit Bayesian models. In the current state of PyMC3, the computational cost remains expensive for high dimensionality. Fitting directly a PyMC3 model with our 64-dependent-variables dataset will make the runtime run out of memory. To make experimentation easy, rather than using the original explanatory variables, I replace them by the prediction given by the Caboost model:

$$\hat{y} = f(g(X))$$

where g is the Catboost model. f is the Bayesian model we will create.

\hat{y} estimation of $Y1$ (Delta departure-presentation) after the Bayesian model correction.

On top of assessing uncertainty, the Bayesian model allows creating multiple variants of a model when initial assumptions are broken by outliers. Our problem can then be more robust to outliers. By taking examples from PyMC3, four Bayesian models will be created:

1. The first Bayesian model is a basic OLS (Ordinary Least Square) model.
2. The 2nd one considers the error variance in the likelihood is not constant but stochastic.
3. The 3rd one uses the Student T-distribution instead of the Gaussian for the likelihood.

Reference: <https://pymc3.readthedocs.io/en/latest/notebooks/GLM-robust.html>.

4. The last one uses the **Hogg** Method (Hogg paper 2010) to build a mixture of sub-models for inliers and outliers. Reference: <https://arxiv.org/abs/1008.4686>.

7.2.1 OLS Bayesian Model

An OLS model can simply be formulated as:

$$\hat{y} \sim N(b_0 + b_1 x, \sigma^2)$$

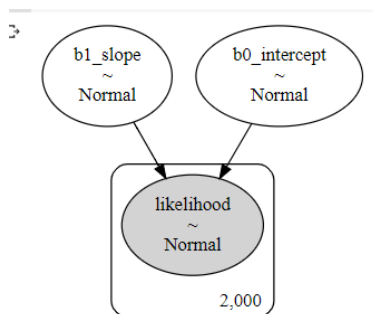
Where $(b_0, b_1) = (\text{intercept}, \text{slope})$,

$b_0 \sim N(0, 120)$ = our prior on intercept

$b_1 \sim N(0, 5.0)$ = our prior on the slope

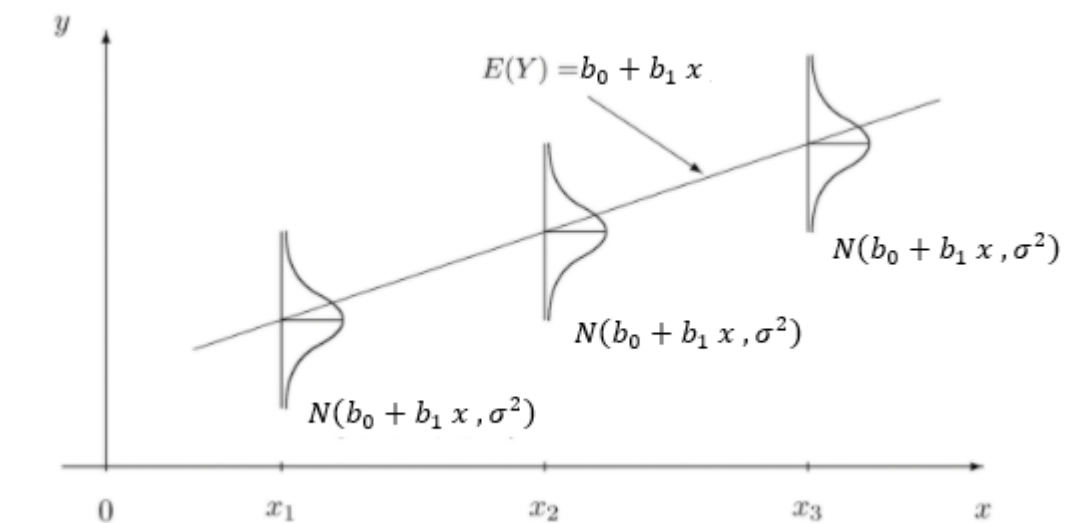
$\sigma = \text{constant standard deviation (homoscedasticity)} = \text{sample std dev}$

The direct graph is like this:



In another word, in a frequentist OLS framework, we supposed the model parameters (b_0, b_1, σ) are fixed:

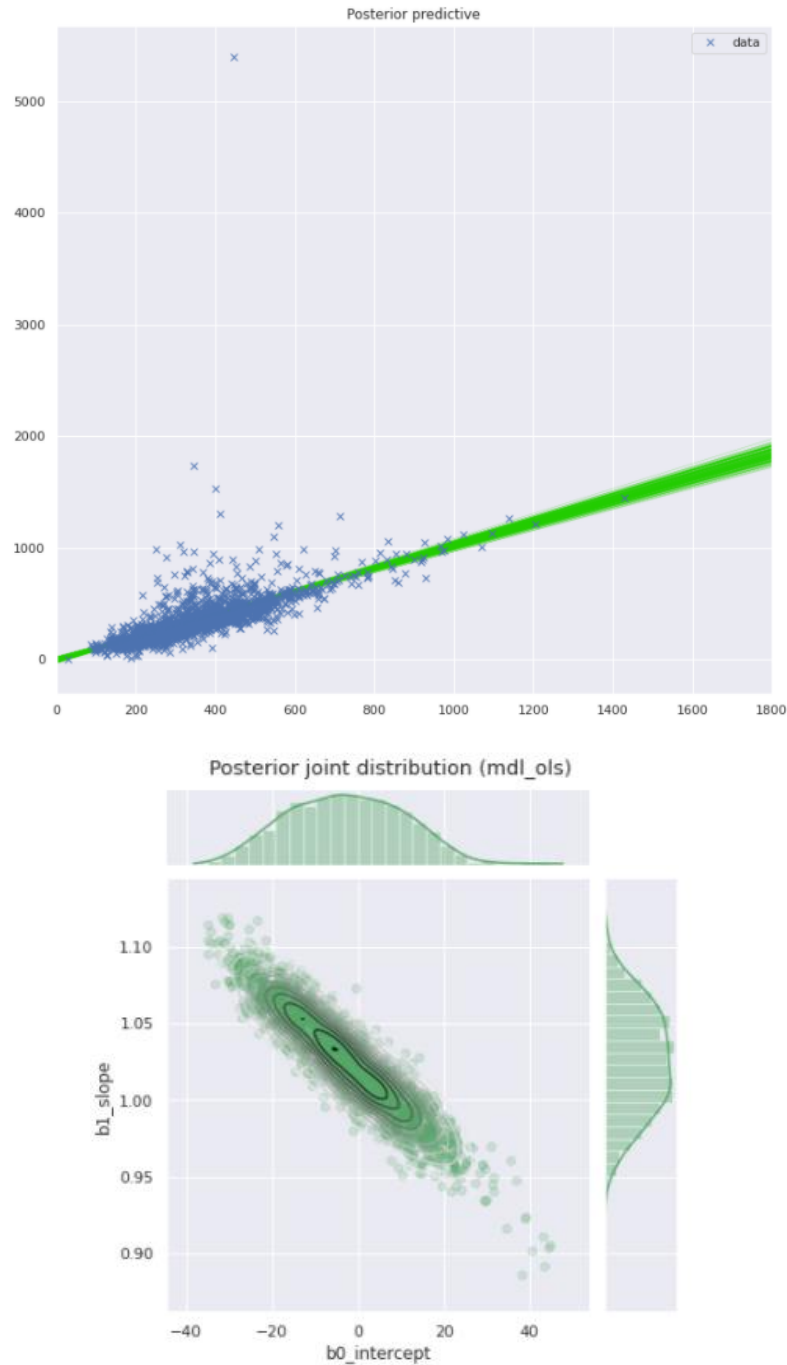
OLS Bayesian Model - Figure 7-1



Whereas in Bayesian settings, these parameters are random. The goal is to estimate their posterior distributions based on our beliefs on their likelihood and priors.

After having fit the model with MCMC, the posterior predictive distribution is below:

Predictive Posterior - Figure 7-2



On the right chart, the observed y are blue crosses. The green lines are different samples of the predictive function according to the posterior distribution.

On the right side, the parameter sample distribution. They follow Gaussian distributions. The dispersion of those green lines and green points informs us about the uncertainty of the model and the predictions.

7.2.2 Heteroscedastic Linear Model

In the previous OLS model, the error was supposed to be constant in variance (σ). This assumption is not true if we look at the “Posterior Predictive” chart above. We can then treat σ as a random variable. A usual prior for variance is the Half Cauchy distribution.

$$\hat{y} \sim N(b_0 + b_1 x, \sigma)$$

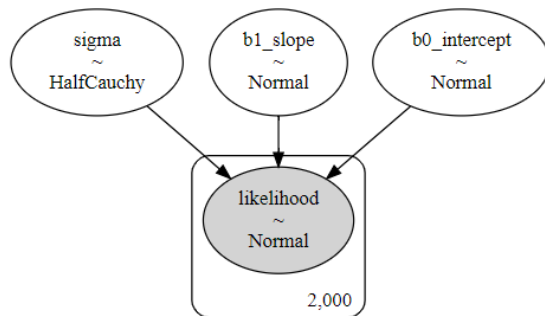
Where $(b_0, b_1) = (\text{intercept}, \text{slope})$,

$b_0 \sim N(0, 120)$ = our prior on intercept

$b_1 \sim N(0, 5.0)$ = our prior on the slope

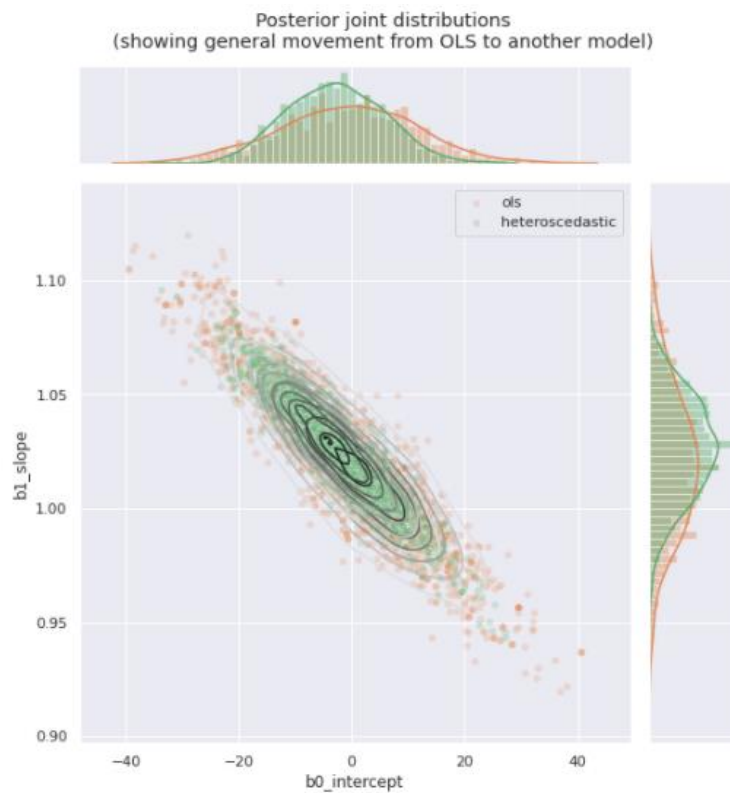
$\sigma \sim \text{HalfCauchy}(\beta)$, $\beta = 10$

Model directed graph:



The resulted posterior distributions for the slope and coefficient are:

Posterior Joint Distribution with heteroscedasticity - Figure 7-3

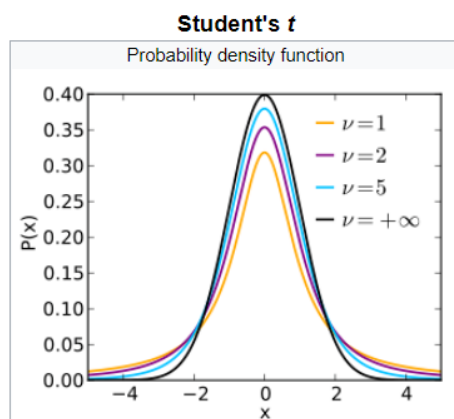


The green curve corresponds to the new model (heteroscedastic), the orange curve OLS. We get a model with a lower variance for the parameters, hence less uncertainty for the model.

7.2.3 Linear model with Student-T

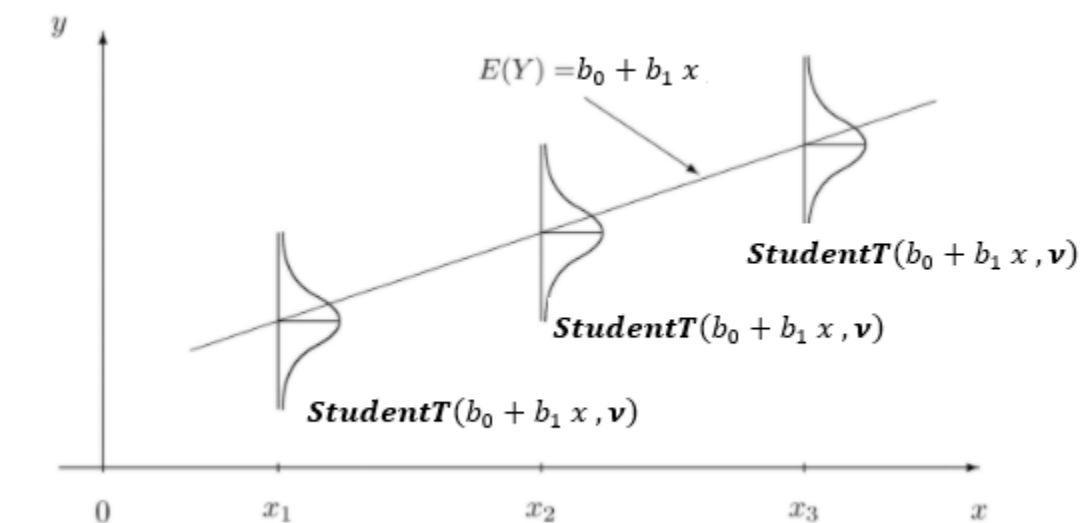
When analyzing the Catboost model, we had noticed the residuals are distributed with heavier tail – i.e. higher kurtosis than a gaussian. That means the residuals tend towards being either closer to the mean, or farther from the mean compared to a gaussian distribution. This shape looks a lot like a Student's T-distribution with a low degree of freedom.

Student-t distribution - Figure 7-4



Source: https://en.wikipedia.org/wiki/Student%27s_t-distribution

Therefore, we can use Student's T-distribution for the likelihood - Figure 7-5



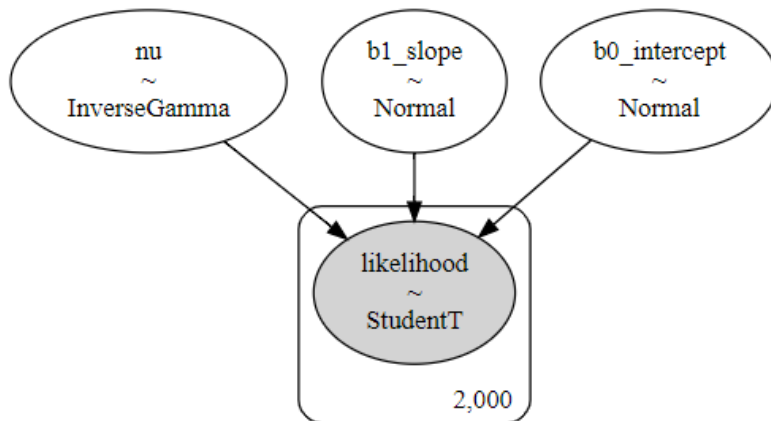
The Bayesian model is expressed as:

$$\hat{y} \sim \text{StudentT}(b_0 + b_1 x, \nu)$$

Where $(b_0, b_1) = (\text{intercept}, \text{slope})$,

$b_0 \sim N(0, 120)$ = our prior on intercept

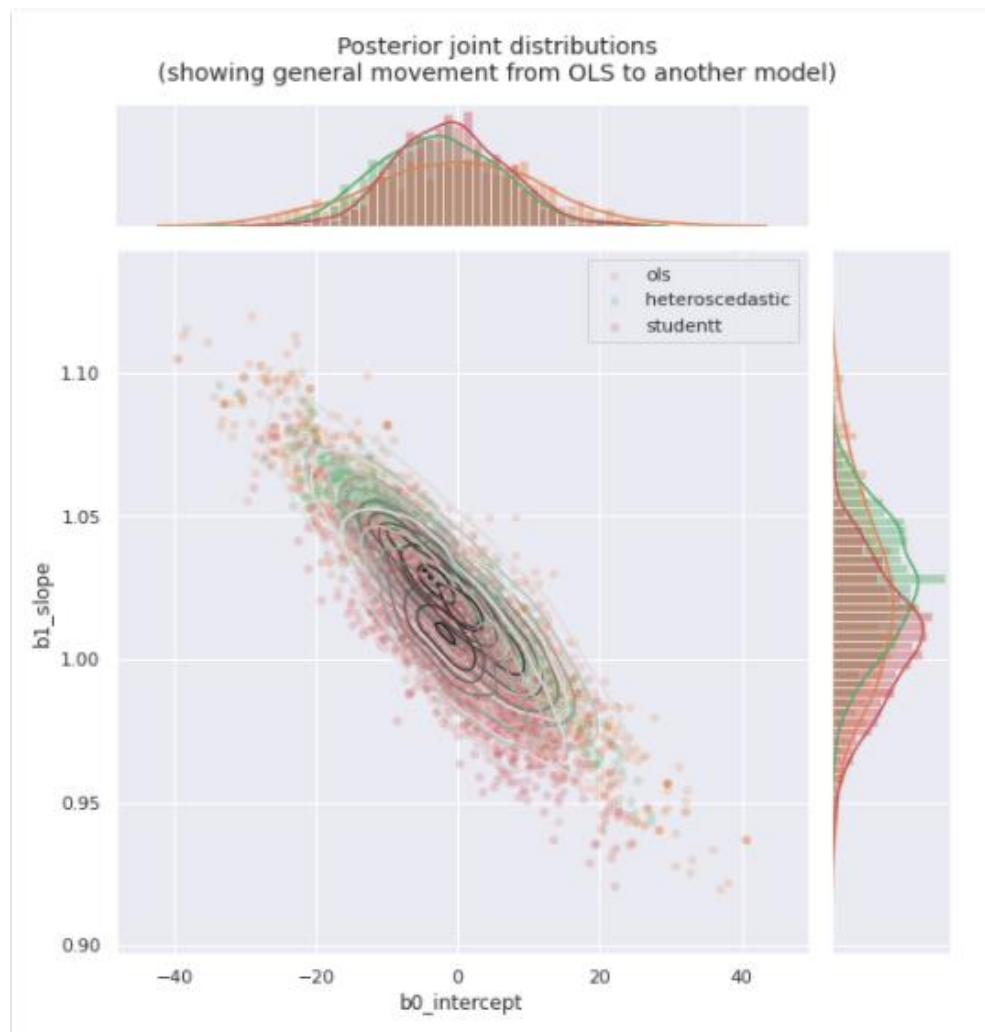
$b_1 \sim N(0, 5.0)$ = our prior on the slope
 $\nu \sim \text{InverseGamma}(\alpha, \beta), \quad \alpha = \beta = 1$



The posteriors distributions are plotted in the graph below:

- Orange: OLS
- Green: Error gaussian with random variance.
- Red: Student T likelihood with a random degree of freedom.

Posterior Joint Distribution with Student-t likelihood - Figure 7-6



Interestingly, the posterior mean of Student-t corresponds to Catboost model:

- Mean intercept = 0
- Mean slope = 1.0

Catboost is already a robust model...

7.2.4 Inliers and Outliers mixture model – Hogg method

The idea comes from Hogg paper 2010 and Jake Vanderplas's notebook. We can consider our data is composed of two latent groups: **inliers** and **outliers**. Each of them follows a different linear model with different mean and variance. Like the Gaussian Mixture model, Hogg proposed a mixture of Gaussian distributions for the likelihood. But unlike the Gaussian Mixture, the component weights are reduced to a boolean variable following a Bernoulli distribution, i.e. a datapoint can belong only to one group:

The log-likelihood is written as:

Hogg Model - Equation 1

$$\log \mathcal{L} = \sum_i^{i=N} \log \left[\frac{(1 - B_i)}{\sqrt{2\pi\sigma_{in}^2}} \exp \left(-\frac{(x_i - \mu_{in})^2}{2\sigma_{in}^2} \right) \right] + \sum_i^{i=N} \log \left[\frac{B_i}{\sqrt{2\pi(\sigma_{in}^2 + \sigma_{out}^2)}} \exp \left(-\frac{(x_i - \mu_{out})^2}{2(\sigma_{in}^2 + \sigma_{out}^2)} \right) \right]$$

where:

\mathbf{B} is Bernoulli-distributed $B_i \in [0_{(inlier)}, 1_{(outlier)}]$

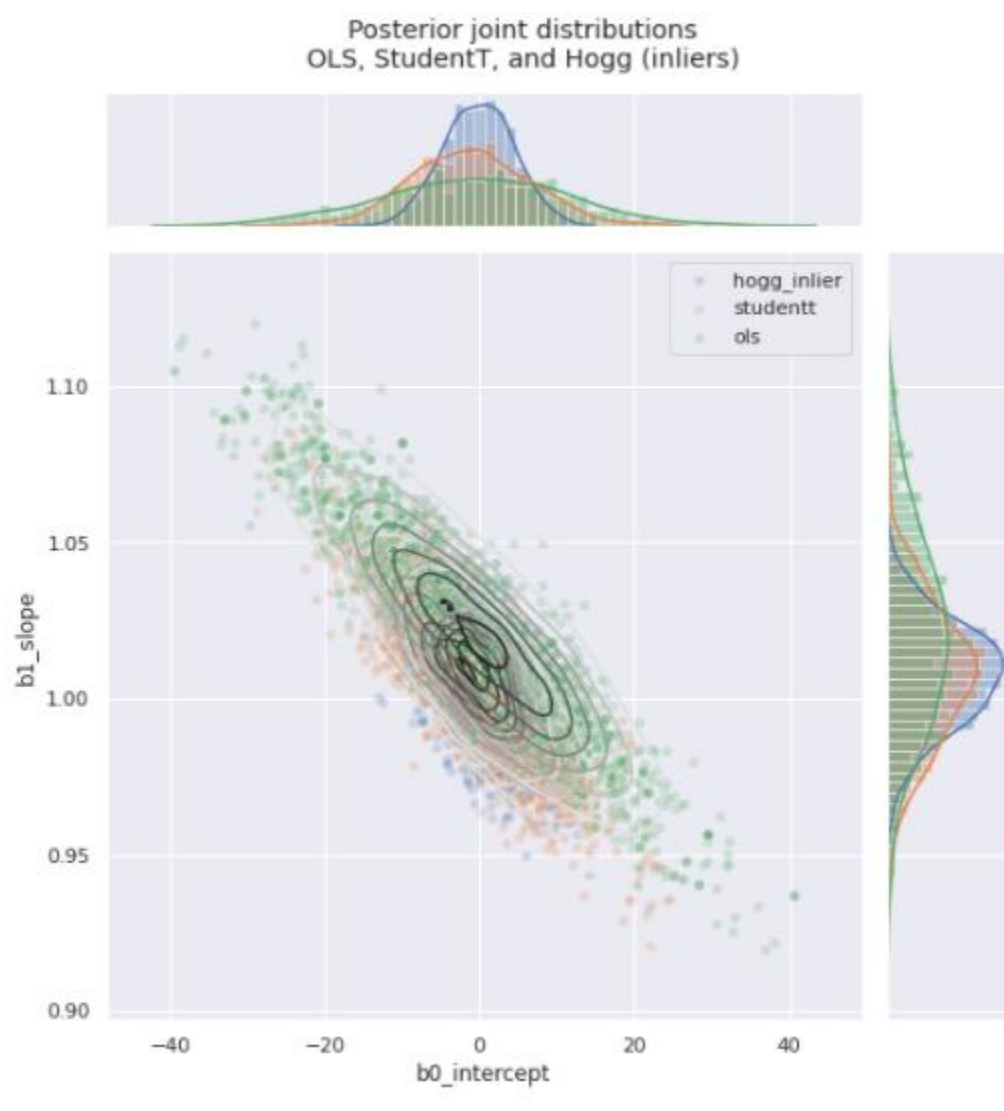
$\mu_{in}, \mu_{out}, \sigma_{in}^2, \sigma_{out}^2$ are respectively the means and variances of inliers and outliers.

We will put together the two components:

Inlier likelihood	Outlier likelihood
<p>OLS model: $\widehat{y}_{in} \sim N(\mu_{in}, \sigma_{in})$ Where: $\mu_{in} = b_0 + b_1 x$ $b_0 \sim N(0, 5.0) = \text{our prior on intercept}$ $b_1 \sim N(1, 5.0) = \text{our prior on the slope}$ $\sigma_{in} = \text{constant} = \text{sample std dev}$</p>	<p>Heteroscedastic model: $\widehat{y}_{out} \sim N(\mu_{out}, \sigma_{out})$ Where: $\mu_{out} \sim N(1800, 3600) = \text{weakly informative prior for outlier's mean}$ $\sigma_{out} \sim \text{HalfNormal}(1800, 3600) = \text{weakly informative prior for outlier's variance}$</p>
<p>Log-likelihood of Y = $\log P(Y X, \theta) = (1 - \pi) \log P(Y_{in}) + \pi \log P(Y_{out})$ Where $\pi \sim \text{Bernoulli}(p)$, $p = \text{constant} = \text{fraction of outliers estimated in sample} = 1\%$</p>	

The resulted posterior distribution is below:

Posterior Joint Distribution with Hogg method - Figure 7-7

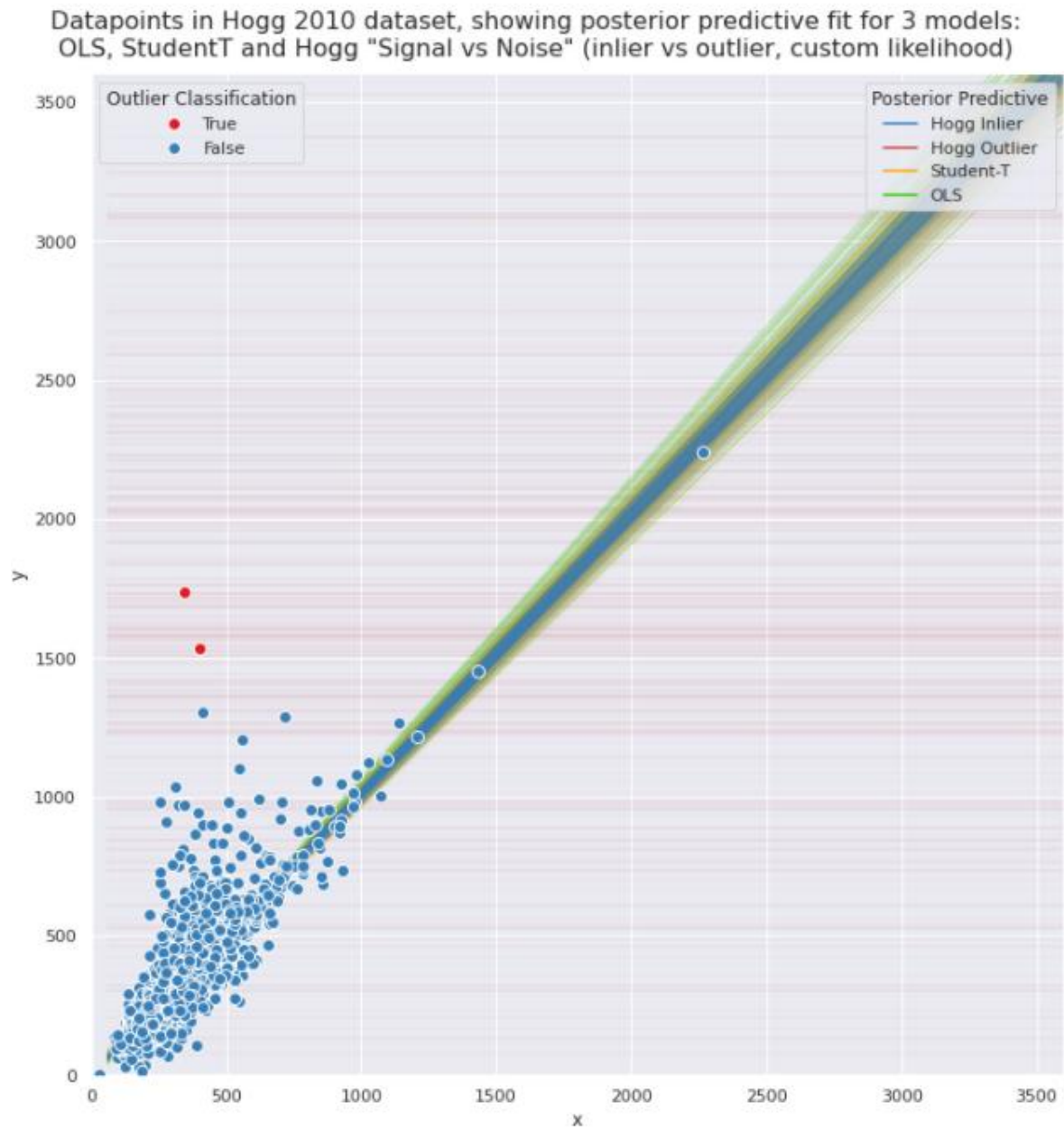


This new model provides two interesting outcomes:

- We have now a posterior distribution for the inlier exclusively (blue curve). It allows us to get a more accurate estimation of the uncertainty for inliers. We can notice its variance is less important in the chart above (blue curve).
- As now we know the outliers' posterior distribution, for any new datapoint, we can estimate its probability of being an outlier.

In the chart below, we display the posterior predictive distribution along with the different models and components (inliers & outliers). The red points are those whose outlier probability is greater than 0.5.

Posterior Predictive function with Hogg method - Figure 7-8



8 Conclusion

This journey in the world of regression allowed me to practice and deepen what Ecole Polytechnique had taught me. I'm thankful to all instructors who trained me.

I really enjoyed all steps in this project:

- The first contact with the data in “**Data Exploration**” did provide helpful insight and intuitions for the rest of the project.
- **Feature engineering** was an important step for winning the challenge. There is no good model without good data. For instance, from GPS coordinates and times, speeds can be computed. I had seen some data that may be missing most of the time, but they turned out to be extremely informative when I carefully made them speak.
- The central parts of the pleasure are to **create**, **analyze**, and **tune** models. It is not to solely try out well-established algorithms or iterate on parameter settings by brute force. Results analysis and intuitions provide fermentation of new ideas. For instance, the choice of a special setting in **gradient boosting**, such as for the loss function, was premediated based on the dispersion of the response variable.
- Explaining the model's results took a large part in my works, providing me the opportunity to discover the notion of **outliers**, using the method of **SHAP** values for interpretability, **classification** and **clustering** for the “detectability” these outliers.
- Lastly, winning the challenge did not stop me from continuing to learn. Other questions, other alternative methods, have opened my curiosity on the vast horizon of **Bayesian inference** which I truly enjoyed.

9 Annex

9.1 Code source

The current project is implemented in Python. The code source is available in

https://github.com/quachn/X_PFB

9.2 List of figures and tables

Histograms of response variables - Figure 2-1.....	6
Distributions of the response variables - Figure 2-2.....	7
Figure 2-3.....	8
Response variable Y1 “delta departure-presentation” over time - Figure 2-4.....	9
Average response time (“delta departure-presentation”) on January over different coordinates - Figure 2-5.....	10
Average response time (“delta departure-presentation”) on August - Figure 2-6.....	10
PCA Explained Variance Ratio - Figure 2-7.....	11
Feature variables projected on the two major PCA components - Figure 2-8.....	11
Response variable “delta selection-presentation” per “OSRM distance” - Figure 2-9.....	12
Response variable “delta selection-presentation” per “OSRM duration” - Figure 2-10.....	12
Distribution of the occurrence of interventions per area - Figure 2-11.....	13
Distribution of the departures’ coordinates - Figure 2-12.....	14
Intervention distribution: interventions are well spread over the city. Figure 2-13.....	14
Intervention trajectories - Figure 2-14.....	15
Average of speed (km/h) per vehicle type - Figure 2-15.....	16
“CRR BSPP” vehicle type - Figure 2-16.....	16
Average of speed (km/h) per “rescue center” - Figure 2-17.....	17
Average of speed (km/h) per “alert reason” - Figure 2-18.....	17
Average speed (km/h) per GPS coordinates - Figure 3-1.....	19
Standard deviation of speed (km/h) per GPS coordinates - Figure 3-2.....	20
Ridge Regression - Figure 4-1.....	23
R-squared on test set with different estimators - Figure 4-2.....	26
Training time [second] and R-squared of different estimators - Figure 4-3.....	27
Categorical variable – target encoding - Figure 5-1.....	28
Huber loss function - Figure 5-2.....	30
Predicted Y1 (“Delta departure-presentation”) vs Actual - Figure 6-1.....	35
Y1 Standardized Residuals distribution - Figure 6-2.....	36
Y1 actual values distribution - Figure 6-3.....	37
R-squared breakdown per residual upper bound - Figure 6-4.....	38
Y1, actual vs predicted values - Figure 6-5.....	39
Unexplainable extreme value in response time (6 hours were needed for 14 km) - Figure 6-6 ...	40
Unexplainable extreme value in response time (2.5 hours was needed for 3.1 km) - Figure 6-7.	41
Outliers distribution over time (month) - Figure 6-8.....	42
Using clustering for outliers detection - Figure 6-9.....	44

Distances distribution to clusters' centroids - Figure 6-10	45
Number of over/under estimations per segment of actual response time (Y1) - Figure 6-11.....	46
Number of over/under estimations per segment of predicted response time (Y1) - Figure 6-12	46
Shap values of the 300- best predicted subset – summary plot - Figure 6-13.....	48
Shap values of the 300- worse predicted subset – summary plot - Figure 6-14.....	49
OLS Bayesian Model - Figure 7-1	53
Predictive Posterior - Figure 7-2	54
Posterior Joint Distribution with heteroscedasticity - Figure 7-3.....	55
Student-t distribution - Figure 7-4.....	56
Therefore, we can use Student's T-distribution for the likelihood - Figure 7-5	56
Posterior Joint Distribution with Student-t likelihood - Figure 7-6	58
Posterior Joint Distribution with Hogg method - Figure 7-7	60
Posterior Predictive function with Hogg method - Figure 7-8.....	61