PROJECT REPORT

«Optimisation of supply network for importing coffee beans»

Group 12:

1. Mikhail Mironov u211361

2. Mikhail Martyanov u211360

3. Sofia Tarasova u211359

UPF Barcelona
2023

Contents

1 Problem Statement

We are glad to be of service to you. We carefully read the report that you sent to us, we understand that being able to predict engine-off time (time when a driver drops off the delivery) is essential as it allows to respect delivery times. Therefore, we will make sure to create the best predictive model for you which will give you the best estimates of engine-off time. Estefania, as you mentioned, previous year was tough for the company to do route planning because of unpredictable engine-off times. In order to tackle this, you decided to use 3 minutes as an approximation of engine-off time for each delivery. Such approach in some way helped you to manage deliveries but it was still not enough, the error of such prediction was significant. We propose using Machine Learning algorithms, this way we will be able to uncover hidden to the naked eye patterns in the data and train the appropriate model that is able to handle predicting of engine-off time way better than the current estimate. Machine learning allows to train a model on existing data and use this model to make future predictions that if done properly are very close to reality. This allows to know beforehand how much time it is going to take to unload the car which in its turn allows to have more predictable delivery times for clients.

Speaking of the data, you provided us with a dataset containing 9000 labeled observations. The dataset contains the following variables Table (1).

Table 1: Variable description

Variable	**Description**	**Data Type**
client_name	Name of the client	String
truck_size	Contains the type of car used for delivery. Could be either *Van*, *Truck*, *Combi*	Categorical: String
truck_origin_warehouse	Defines the warehouse where this delivery started from	Categorical: str
delivery_timestamp	At what date and time was the delivery done (defined as the moment the engine-off time starts)	Datetime
total_weight	Total weight of the goods delivery	Numeric: Float

Table 1: Variable description

Variable	Description	Data Type
brand_1_coffee_proportion	what percentage of the delivery was of Beanie's brand #1	Numeric: Float
brand_2_coffee_proportion	what percentage of the delivery was of Beanie's brand #2	Numeric: Float
brand_3_coffee_proportion	what percentage of the delivery was of Beanie's brand #3	Numeric: Float
driver_id	the ID of the driver that was driving the route	Categorical: String
is_fresh_client	whether the client was fresh at the date of the delivery. Fresh clients are clients that have been doing business with Beanie for less than 30 days	Boolean
postcode	the postcode of the client location	Categorical: Int
business_category	whether the client is a hotel, a cafe or restaurant or a coffee retailer	Categorical: String
floor	the physical position of the client location	Categorical: String
box_count	how many distinct boxes were delivered to the client. The coffee beans bags are grouped into boxes for delivery	Numeric: Int
final_time	the engine-off time, measured in seconds	Numeric: Float

The dataset you provided has no missing values, as well as the data itself is clean, meaning that we do not have any obvious outliers that will skew the results. Now we will start discussing what features might be relevant for prediction of engine-off time.

1.1 Feature selection and engineering

In this part we will try to understand which of the variables you provided might have an impact on the engine-off time. We will go over variables that we believe are significant and explain our reasoning behind it.

First is either *total_weight* or *box_count*. These are the obvious choice since they directly represent the amount of work a driver has to perform to do the delivery, so we should definitely include either of these variables into our analysis. Second one is *delivery_timestamp* namely the time of the day of the delivery which could be either morning, day or evening. We believe that time of the day might influence how busy the client location is which could get in the way of fast delivery, or simply due to behavioural patterns of the drivers that tend to work slower during early hours. From Figure (1) we can see that *delivery_hour* seems to have an effect on the *final_time* whilst others do not since changes in *delivery_hour* result in greater changes in *final_time*.

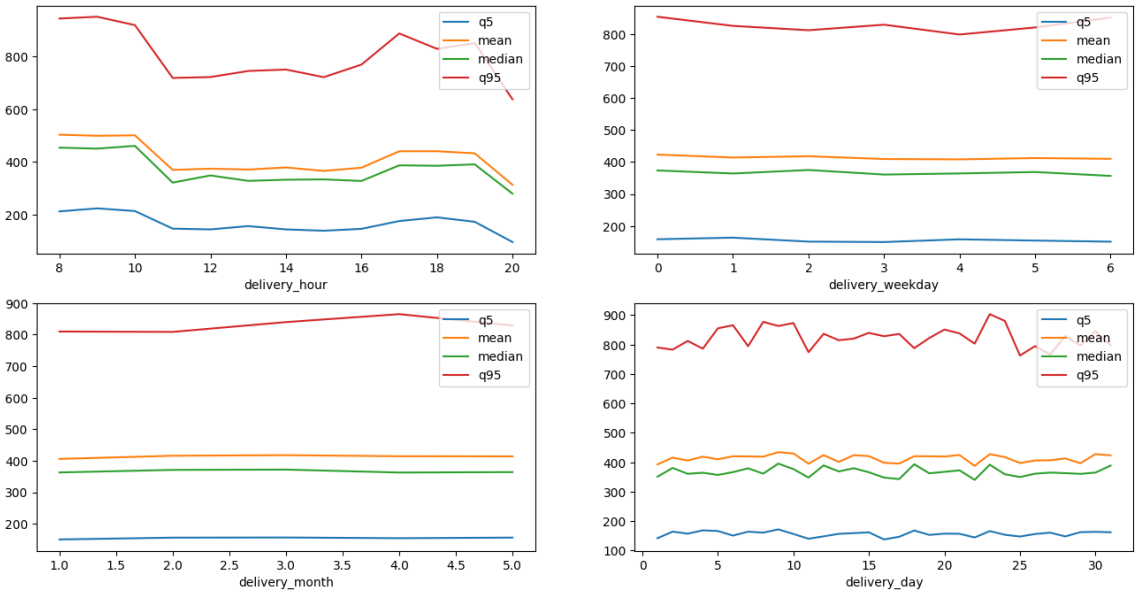

Figure 1: Final time vs time variable.

Moreover, we believe that *truck_size* variable is also significant. To see this we calculated mean values of *final_time* for each of type of truck, results are presented in the Table (2) below.

Table 2: Final time across types of truck

truck_size	q5	mean	median	q95
Combi	143.20	388.60	346.69	793.86
Truck	163.56	**432.90**	**375.55**	875.64
Van	157.96	419.01	372.83	829.07

As we can mean and median values are slightly different, this result might be statistically insignificant, but we believe that unloading a truck and a small van is different in terms of effort. Truck itself might be higher above the ground, so it takes additional effort to handle boxes especially heavy ones.

The next variable is *floor*. This also makes perfect sense since it measures the amount of work performed by the driver as well. If you need to go to higher floors to deliver the package especially a heavy one, this will naturally take more time than just dropping it off at the ground floor. Below is the Table (3) with final times for each type of floor.

Table 3: Final time across types of truck

floor	q5	mean	median	q95
Ground Floor	178.49	453.12	410.37	898.68
Other	211.45	**522.42**	**481.03**	1012.76
Street level	146.66	390.88	344.97	788.75

We are also considering using *driver_id* column. We think that this variable is one of the most influential as it can describe how the specific driver handles deliveries. Some of the drivers might be more physically capable to handle heavy boxes and therefore deliver them faster or some might be just more dedicated and focused on work. From Table (4) below we can see that despite having more or less the same work, they perform differently on average, this corresponds to similar values in *mean_weight_handled* columns whilst different values in *mean* and *median* columns.

Table 4: Final time across types of truck

driver_id	q5	mean	median	q95	mean_weight_handled
D13	252.55	548.64	505.96	1019.13	62.84
D16	231.38	496.70	450.58	941.21	60.51
D18	197.25	456.15	403.40	885.89	62.14
D27	203.63	459.28	420.81	840.73	63.35
D33	105.70	285.27	256.07	550.54	61.55
D49	158.87	372.63	336.18	705.68	62.39
D58	192.82	433.85	400.27	793.25	62.50
D61	157.84	399.21	349.02	782.83	62.72
D63	285.77	608.66	553.59	1117.50	63.05
D64	189.69	435.96	389.12	825.89	62.76
D84	187.53	416.54	371.34	807.96	62.52
D9	181.24	416.42	371.79	806.13	64.17
D98	107.55	281.26	251.34	556.26	62.88

This implies that indeed there are individual effects on engine-off time, therefore, to account for these we will include them into our analysis. Also we believe that *is_fresh_client* variable should be helpful. New clients might be not used to the whole process of receiving the order, hence it takes slightly more time to drop off the delivery for them.

As a result, after feature selection we ended up with the following variables that will be used for analysis: *box_count*, *is_fresh_client*, *driver_id*, *truck_size*, *floor*, *delivery_timestamp* (to account for the time of the day when delivery occurs).

2 Modelling

2.1 Model training and testing

We will use 80-20 train test split. First 80 % of the data is a train set that is used for training of the model. The other 20 % are used to test the model, and it should not be used for any hyperparamter tuning and comparison of the models. Otherwise we are just adjusting our models so it yields the best results which creates bias meaning our model is not generalising. This might lead to the model being completely wrong once run in the production. In order to tune models and choose the best ones we will do Repeated K-Fold crossvalidation. This will

allow us to save up on training data as we will not have to create another separate validation set. Also such approach allows to control overfitting as choosing models and tuning their hyperparameters on some predefined validation set might result in overfitting because left-out validation set might be not representative of the general population. Moreover, we will use repeated crossvalidation meaning that we will create multiple variations of K-Folds which eliminates risk of being just lucky with folds we get for validation.

2.2 Loss function and evaluation

Before building a model we have to decide on loss function we will use that is the most suitable for our task. We might go for a simple MSE (Mean Squared Error) which heavily penalises being wrong by a huge margin. This way our model will be more influenced by outliers which might hurt its performance for regular observations (inliers), but this is necessary as it is better to make smaller mistakes rather than huge ones making clients wait way more than expected. We might also consider using MAE (Mean Absolute Error) loss to make our models more robust to outliers, this solution is the opposite, we will ignore outliers but improve predictions for regular observations, meaning that MAE will make no mistakes for inliers and once in a while a giant mistake for outliers. We believe that the first scenario is more in line with Beanie Limited standards, so we will use MSE as loss function.

For evaluation we will choose MAPE (Mean Absolute Percentage Error) as it is easy to interpret. MAPE describes how significant the error is compared to the absolute value of the engine-off time, it allows us to see the scale of the error itself. Lower MAPE indicates that on average model is able to make small errors relative to final engine-off time whilst absolute metrics such MSE, MAE and others are not able to pick up this effect.

2.3 Evaluating previous strategy

As you Estefania mentioned, you used 3 minute estimate of engine-off time for every delivery. Now we can evaluate perfomance of this estimate using metrics discussed above.

Table 5: Performance of current Beanie Limited strategy

	MAE	MSE	RMSE	MAPE	R2
$\hat{y}_i = 180$	239.72	100664.18	317.28	0.49	-1.18

2.4 Baseline model

It is generally a good practice to come up with a simple model in order to compare other models against it. This way, we will be able to see how added complexity to the model improves its performance if it does any in the first place. In our case, we trained multiple linear regression:

$$\hat{y_i} = \alpha + \beta_1 \cdot box\_count + \beta_2 \cdot is\_fresh\_client + B_{3..15} \cdot is\_driver + \\ B_{16..18} \cdot is\_truck\_size + B_{19..21} \cdot is\_time\_of\_day + B_{22..24} \cdot is\_floor \tag{1}$$

$B_{n..k}$ denotes a column vector of β estimates while variable that follows indicate a one-hot vector. Therefore, the model that we have, include both numeric and binary variables. We used Repeated K-Fold cross-validation same as explained in Section (2.1) to get the results below in Table (6).

Table 6: Cross-validated results for Baseline Linear Regression

	MAE	MSE	RMSE	MAPE	R2
q5	72.64	9768.36	98.83	0.21	0.79
mean	69.97	8835.52	93.96	0.20	0.81
median	69.88	8795.79	93.79	0.20	0.81
q95	67.55	8046.18	89.70	0.19	0.82

As we see from above baseline algorithm already beats the estimate of 3 minutes used previously. We are now able to improve on MAPE almost by 2.5 times. MAE improved by about 160 seconds which means that on average our predictions of engine-off are now within almost 1 minute error not 4 as it used to be. Further, we will see that we are able to do way better than this.

3 Finding the best model

3.1 LASSO Regression

Now we will improve on results obtained with Baseline Linear Regression. As a proxy model, we used LASSO regression to check if variables that we chose are the most significant ones and we do not lose out on much by omitting other variables from the data. LASSO regression includes a penalty on having high values of coefficients, therefore LASSO model only picks up

the most significant variables. We ran cross-validation for LASSO model on the same folds to see how it compares to the baseline. Results are presented in the Table (7) below:

Table 7: Cross-validated results for LASSO Regression

	MAE	MSE	RMSE	MAPE	R2
q5	72.60	9761.01	98.80	0.20	0.79
mean	69.91	8827.42	93.92	0.20	0.81
median	69.81	8791.49	93.76	0.20	0.81
q95	67.50	8037.37	89.65	0.19	0.82

As we see, we were right about the selection of the variables, LASSO puts highest coefficients on the most significant variables same as we discussed earlier while having basically the same stats. Since we didn't do any scaling of the data, we have perfect interpretability with LASSO regression. Therefore, now we are safe to use the same set of variables that we used for Baseline model.

3.2 RandomForest Regressor

In our data we might have non-linear patterns, in order to tackle this non-linearity we could use Decision Trees that are able to learn these non-linear patterns, but they are prone to overfitting. In order to solve this, we might want to use bagging (bootstrap aggregating) like RandomForest where we train multiple relatively shallow Decision Trees and pool their predictions to create a combined prediction. RandomForest also has multuple hyperparameters that control the training process as well as the quality of predictions. Therefore, we might want to optimize for these as well. As a result, we trained RandomForest multiple times and chose the parameters that get the highest crossvalidated MAPE score. Then, we used these best parameters to train the model and crossvalidated it on the same folds as previous models, so it is a fair comparison. In Table (8), there are results of crossvalidation:

Table 8: Cross-validated results for RandomForest Regression

	MAE	MSE	RMSE	MAPE	R2
q5	68.25	9123.19	95.52	0.18	0.80
mean	65.76	8412.78	91.68	0.17	0.81
median	65.85	8391.93	91.61	0.17	0.81
q95	63.60	7688.40	87.68	0.16	0.83

3.3 Gradient Boosting Regression

Another approach is using Boosting algorithms, where week learners are trained sequentially so that the next one is attempting to learn the error of the previous learner. We expect that boosting will result in superior results. For boosting we used GradientBoostingRegressor, we as well fine-tuned the model to have the lowest crossvalidated MAPE score. The following Table (9) is a result of crossvalidation:

Table 9: Cross-validated results for GradientBoosting Regression

	MAE	MSE	RMSE	MAPE	R2
q5	64.60	8178.08	90.43	0.17	0.82
mean	62.57	7576.29	87.01	0.16	0.83
median	62.28	7677.75	87.62	0.16	0.83
q95	60.70	6878.33	82.93	0.16	0.85

Boosting allowed us to improve even further, now we have MAPE of 16% which is compared to current 49% is a 3 times improvement. This will allow to substantially improve deliveries.

3.4 Comparing obtained models

Below is the graph where we plotted predictions of all models on same crossvalidation folds. Blue dots are scattered across y=x line, they represent the true values of engine-off time whilst other points are predictions of the models against the actual values.

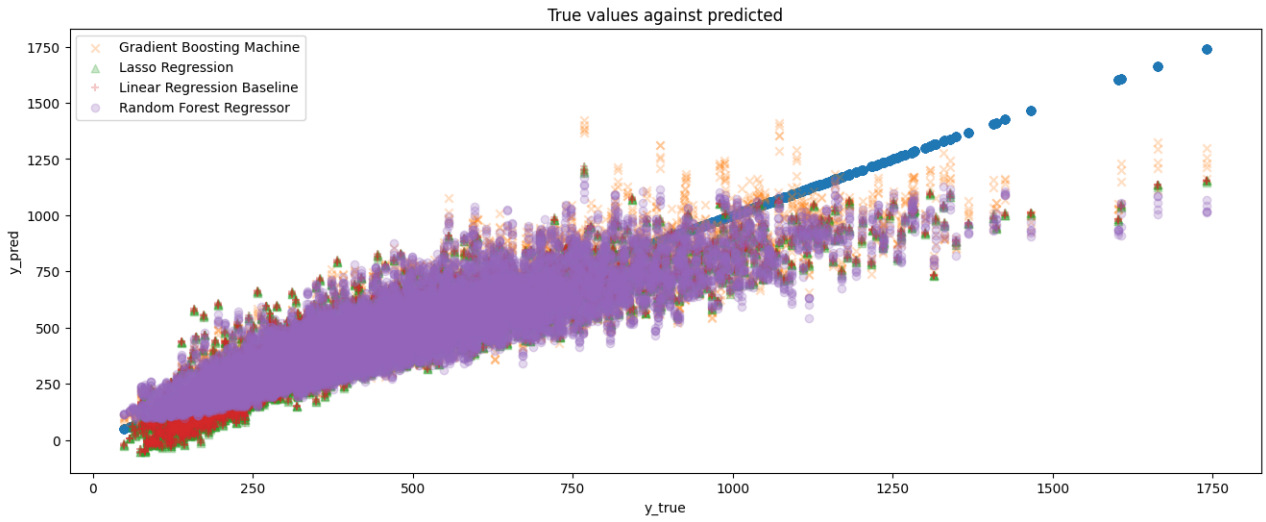

Figure 2

The closer a dot is to the blue line the more accurate the prediction is. Therefore, according to this, we see that RandomForest (purple circles) and Gradient Boosting Machine (orange crosses) are scattered the closest to the blue line, this implies that they are better than other models. It is more interesting to see how models behave for higher values of real engine-off time. Figure (3) below shows how model compare to each other when predicting higher values of engine-off time.

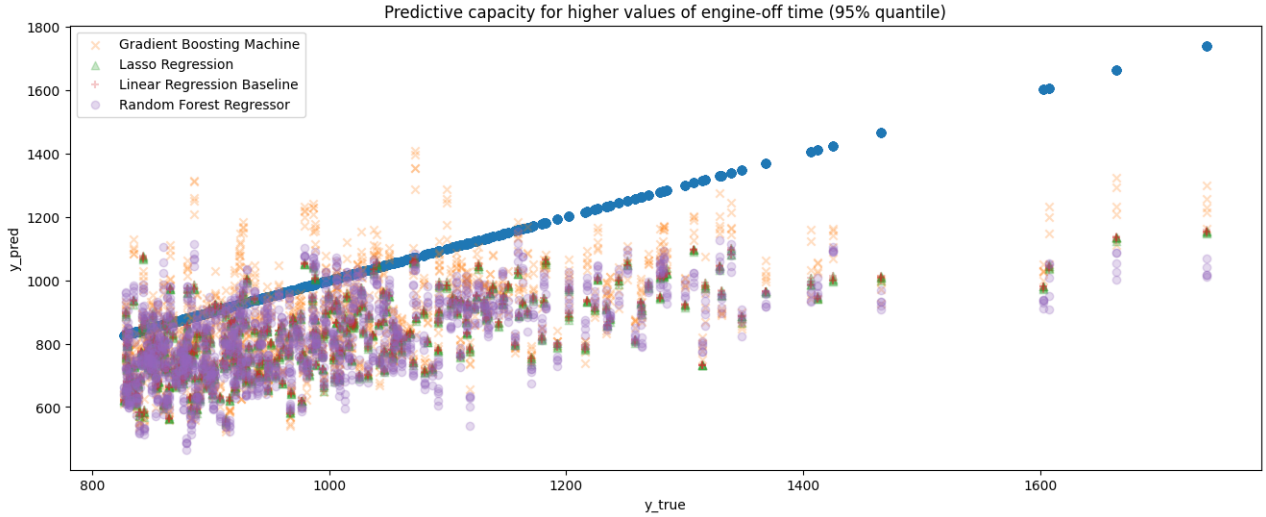

Figure 3

We see that Gradient Boosting Machine outperforms others here as well. Summing up, after seeing that boosting gave us the lowest MAE, MAPE, MSE and highest R2 for the same of variables, we should definitely choose Gradient Boosting Machine (GBM) as our final model.

4 Interpreting the model

We are blessed to have decision trees as underlying learners because it is possible to produce good predictions with them as well as have good interpretability, which usually hurts one another.

4.1 Feature Importance

We might want to use Feature Importance which are defined as an overall impact of the variable towards improving the splits in the decision trees. By default squared error is used to measure how good the split is. Therefore, sum of all changes in squared errors relative to overall decrease

in sum of squared errors (SSE) is defined as impact of the variables. The most significant variables will allow to decrease SSE the most.

Below Table (10) is the calculated features importance for our best model GBM. In order to eliminate chance of just being lucky, model was retrained with multiple resampled train sets and for each we calculated the feature importances, then we just took the average.

Table 10: Feature importances for Gradient Boosting Machine

	mean_feature_importance
box_count	0.636724
D98	0.080611
is_fresh_client	0.050145
D33	0.041209
Street level	0.037382
D13	0.034054
D63	0.027552
morning	0.026901
Other	0.013234
D16	0.013119
Combi	0.008428
D49	0.005751

Additionally, we could have calculated **permutation importances** when features are randomly shuffled and then we check how worse the model has become in terms of the scoring metric. If it became worse than the feature is significant, otherwise it is just random noise that we should neglect. Since we hand-picked our features and double checked their validity with LASSO regression (they are not thrown away with regularisation), we don't need to perform this, results will coincide with feature importances.

4.2 Future improvements

We might consider choosing boosting algorithms that use different approaches of building trees from GBM like LightGBM, XGBoost and CatBoost. We might also go heavy on hyperparameter tuning especially for CatBoost to find the best set of parameters this will allow to achieve an improvement in results by 3-5% but it will be computationally difficult we might need to get a GPU to train multiple models especially if we get more data.

Also as Pablo suggested we can use cyclical time encoding (represent time in terms of sine and cosine), previously we saw that hour column is significant meaning it has a lot of impact when predicting final engine-off time. By doing cyclical encoding we preserve measure of how close timestamps are to one another.

As you saw, MAPE is a very good metric, it is very powerful and easy to understand at the same time. Currently, we are optimising models to get the lowest MSE and then tune the hyperparamters for the best MAPE. But instead we could optimize MAPE directly as a loss function, for instance, in CatBoost we can do it.