**Ramp Competition: Number of Air Passengers**

Project Report

**By Paul-Antoine Girard & Adrien Toulouse**

**Username: toulouse\_girard**

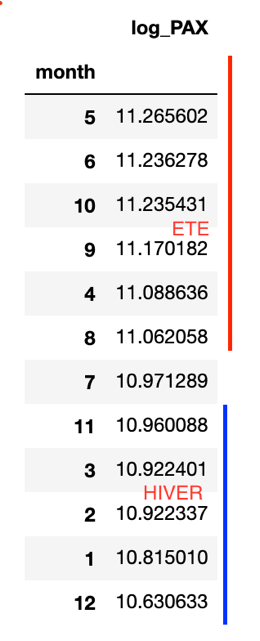
**Final submission name: ???**

***The aim of this report is to narrate our workflow, to detail everything we tried and explain the choices we made to build our best model and predict the number of air passengers.***

## **Starting point, descriptive statistics and data preprocessing**

Understanding the task and how to proceed was our first challenge, as it was for both of us our first data science competition. Our starting point was to look at the data we were given. We tried to find patterns, to see the correlations between variables, to detect missing values and to see which variables could help us predict best the number of air passengers.

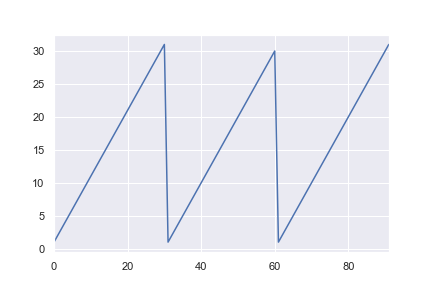
Une image contenant capture d’écran

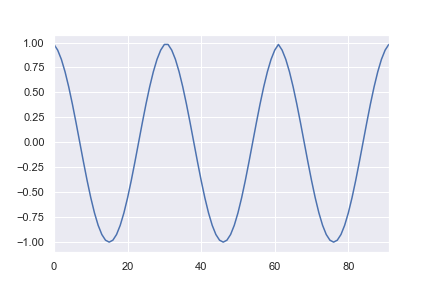
Description générée automatiquement**A. *‘Date’***

We started by looking closely at the date after its one-hot encoding. Our correlation analysis showed us that there was a difference in the number of air passengers for the different weekdays and the different months. Therefore, we built a variable ‘*weekend’* and a variable ‘*season’* to better capture this information. Creating these variables improved our score when we used specific regressors such as SVM. However, they didn’t improve our score in our gradient boosting model, so we didn’t keep them in our final submission.

We also remarked that the variables which represent the date (‘*month’*, ‘*day’*…) weren’t continuous. In fact, for the variable ‘*month*’, December corresponds to 12 and January to 1. So, the algorithm doesn’t know that January follows December. Therefore, we designed a model in which we transform the different variables (‘*day’*, ‘*month’*, ‘*week*’, and ‘*weekday*’) into two dimensions using cosines and sinus functions to represent the data continuously and this allowed us to improve our score. We also kept the column summarizing the year of the flight and the number of the day from a particular date, but we subtracted 2011 to the ‘year’ variable to have an encoding that goes from 0 to 2, and we computed the number of the day from the first date of the dataset.

The first graph shows that if we represent days with their number in a month, every 30 or 31 days, the graph isn’t continuous. However, when we use our transformation (second and third plots), the values go from -1 to 1 and it’s always continuous.

 A picture containing sky

Description automatically generated

**B. *‘Airports’***

Then, we concentrated our work on how to encode the airports. The first method was to use a one-hot encoding. However, this technique leads to 40 columns (two columns per airport) and we figured that we didn’t need that many. Our idea was to create a column per airport and put the value 1 if the airplane leaves this airport or -1 if it arrives there. The number of columns was divided by 2. Lastly, we decided to implement a mean encoding method (we multiplied the airport columns by the average number of passengers respectively leaving or arriving in that airport). This encoding allowed to improve the score of our final model.

**C. *‘Reservations’***

Finally, we focused on the reservation variables (‘*WeeksToDeparture’* and ‘*std\_wtd’*). We found a strong correlation between these two and tried different methods to handle the information contained in the two columns. We tried to replace the two variables by a single product column of the two. We also tried to use a PCA algorithm with one principal component, but in the end, we decided to simply drop the variable ‘*std\_wtd’* and to create different categories for the variable ‘*WeeksToDeparture’*. We created four groups of equal sizes (using *pd.cut*) by ordering the variable ‘*WeeksToDeparture’*, and we encoded these groups from 0 to 3.

## **Adding and preprocessing external data**

Secondly, we looked at the external weather dataset. We analyzed the correlation between the weather variables and the number of air passengers and tried to:

* regroup variables using a PCA algorithm with a smaller number of principal components,
* delete some redundant information (minimum and maximum of a variable),
* do operations between them (like multiplying humidity and temperature),

We ended up only keeping the variable ‘*Events’* which we thought could have an influence on the number of passengers*.* As this variable was categorical, we first replaced missing values by 0 and we encoded into three numerical values: 0 when there were no particular events, 1 for common events such as rain or fog, 2 for more important events such as thunderstorms.

Overall, we thought that the weather data wasn’t necessarily the best thing to work on to really improve our model. In fact, the weather on the day of arrival will not necessarily determine the weather of the following days for a week of holiday for example.

As the external data we initially had didn’t allow us to explain enough the number of air passengers, we decided to add new external data.

Overall, we added the following four variables:

* Distance in kilometer between the airports: ‘Distance’
* Number of inhabitants per city: ‘Population’
* Revenue per state: ‘Revenue’
* Crude oil price: ‘Oil\_Price’

Every time we added a new variable, our technique was to also to challenge the other variables. We didn’t want to keep information if they weren’t relevant. Therefore, each time we added data to our model, we tested to remove others to see if they still had an impact on our model or if the accuracy was still the same without them.

**A. *‘Distance’***

Our first idea was to add the distance between the airports. We had the intuition that longer flights would carry more passenger and we thought that the distance information could help our algorithm to make a difference between long flights with more passengers and shorter flights. We found the geographical longitude and latitude of each airport and used a function to calculate the distance between each of the twenty airports. This information consequently improved our score, so we decided to keep this variable. We tried at some point to regroup the distance between airports into three categories of flights that we could encode: long flights, medium flights and short flights. However, this encoding did not allow us to improve or score and we lost a bit of precision, so we decided to stick with the distance variable.

**B. *‘Population’***

Our second idea was that the number of passengers on a flight could be explained by the population of the city of arrival and that flights going to big cities would carry more passengers. Therefore, we created a column with the number of passengers in the city of arrival. We kept it during a long period of our tests, but in the end, we realized that it didn’t explain significantly our response variable and didn’t keep it in our final model.

**C. *‘Revenue’***

Related to the number of inhabitants of a city, we also thought that the per Capita income could represent the resources of a state. The richer the state, the more its infrastructure is developed, and the more travelers and workers are coming and going. However, the score was the same with and without this variable. So, we tried to use it differently and we created 3 categories of revenues. We affected 0 to the airports of a low-income state, 1 to the airports of a standard income state and 2 to airports of richer states. Sadly, this didn’t help our algorithm either and we also dropped this variable.

**D. *‘Oil\_price’***

Lastly, looking at the bigger picture, we focused on the expenses of an airline company. Most of them are related to the airplanes and their maintenance, the crews, and oil. Since this last expense has a cost that varies enormously, its variations may be reflected in air tickets’ prices. So, we added this variable to our external dataset and tried different techniques to use it. When we first added this variable, it wasn’t relevant. The oil price of the day of the flight isn’t influential on the tickets’ prices since they are established a long time before. So, we tried another approach and we associated with a flight the oil price three months before it. After adjusting several times the number of months before the flight, the variable still wasn't relevant. So, we decided to drop it too.

## **Choosing a machine learning algorithm: ‘Honey badger don’t give a crap’**

After comparing the results from several regressors among which Random forest, SVM and Logistic regression, we decided to go for a gradient boosting algorithm. More particularly, we opted for the *XGBRegressor* as it fitted the best our data and gave us our best score.

Some advantages of this algorithm are:

* Speed: XGBoost is quick to execute thanks to parallel processing.
* Performance: XGBoost outperformed the other algorithms we tried.
* Regularization: XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization parameters that we can adjust to prevent the model from overfitting.
* Not too dependent on the number of variables.

After several tries, we wanted to find another regressor that would perform as well but go faster. We tried to use the *LGBMRegressor*, which is another gradient boosting regressor. This new algorithm was performant and a lot faster when we used it with the local test (RAMP\_submission\_test). However, when we submitted our work using this regressor, we didn’t get results as good as with the *XGBRegressor*, and the time wasn’t reduced. It went only 0.5 seconds faster. Therefore, we dropped *LGBMRegressor* and kept *XGBRegressor*.

## **Parameters tuning**

Every time we tried a new model or tried an existing model on new data, we tuned and adapted the parameters of our regressor to fit the data as well as possible. To do so, we used a cross-validation GridSearch based on minimizing the RMSE.

For our best model, the most important parameters are:

* *reg\_alpha = 0.001*: regularization parameter to reduce overfitting
* *max\_depth = 7:* Maximum depth of a tree. We Increased this value to make the model more complex and even though it is more likely to overfit we got better results.
* *n\_estimator = 2000*: number of boosted trees. We set a high value to improve our model
* *learning\_rate = 0.3:* learning rate of our model
* *min\_child\_weight = 5:* Minimum sum of instance weight needed in a child. We increased it too to make our model more conservative.

Overall, our parameters tend to make our model overfit the train data (we noticed a quite important difference in our results between the train and the test set) but these are the parameters that gave us the best score for the competition.

## **Score obtained, lessons learnt and conclusion**

***Finally, our model obtained a RMSE of ??? and ranked ??? in the RAMP competition.***

Our good score was a combination of all the things we tried during these several weeks of competition: adding new useful variables to our dataset, preprocessing all our variables, choosing the best machine learning algorithm and tuning its parameters to fit our data.

Our workflow was very circular: every time we added new data or did relevant feature preprocessing, we would challenge other variables in the actual model, and analyze how our score was impacted. We also tried to choose the best regressor and adapt it each time so that it would fit the dataset. We tried to improve our score at every step of our workflow and would continuously go back and forth all the steps.

Overall, this project was particularly interesting to understand the importance of data preparation and the particularities of the different regressors. It was also fun to look for new external data and see how they could improve our model. Undoubtedly our model could still be improved by adding more data and going deeper into parameter tuning.