# Spaceship Titanic: an Analysis for Transported Passengers

**Fundamentos de Aprendizagem Automática**
**Dr. Pétia Georgieva**

João António
*Mestrado em Ciência de Dados*
*DETI*
*Universidade de Aveiro*
*Aveiro, Portugal*
*N.Mec. 76558, joaoantonio@ua.pt*

Tiago Freitas
*Mestrado em Ciência de Dados*
*DETI*
*Universidade de Aveiro*
*Aveiro, Portugal*
*N.Mec. 76748, tiagofreitas79@ua.pt*

***Abstract*—A predictive analyses of four different models were performed in the scope of a Kaggle competition "Spaceship Titanic". The specific purpose of this study is to evaluate which passengers where Transported based on some initial conditions /features. This features were scraped from the damaged computer, resulting in around one forth of the passengers have at least one not a number (Nan) in their features.**

**Using minimal hyper parameter search, we have reached the top 7% of the leaderboard, place number 167 at the time of writing this.**

***Index Terms*—Kaggle, Spaceship Titanic, LGBM, CatBoost**

November 17, 2022

## 1. Introduction

The dataset is a 'GettingStarted Prediction Competition' from Kaggle [1] where we have a classification problem that has inspired on the widely known Titanic dataset.

One of the main differences between these two datasets is that the results are only known by Kaggle, while on the original Titanic competition, some users were able to reverse engineer the problem by watching the movie, and creating a a script that memorized the names, and predicted the correct fate for every person (resulting in 100% accuracy with no machine learning models).

The Introduction that we found on Kaggle [1] is:

"In the year 2912, an interstellar vessel named *Spaceship Titanic* was transporting nearly 13 000 passengers from our solar system to three newly habitable exoplanets orbiting nearby stars, when it collided with a spacetime anomaly hidden within a dust cloud, meeting a similar fate as its namesake. Though the ship stayed intact, almost half of the passengers were transported to an alternate dimension. To help rescue crews and retrieve the lost passengers, we will try to predict which passengers were transported by the anomaly using records recovered from the spaceship's damaged computer system".

## 2. Problem Definition

### 2.1. Initial Dataset

The data set we will be working on contains the following personal records for about two-thirds (8693) of the passengers:

- PASSENGERID - A unique Id for each passenger of form gggg_pp where gggg indicates a group the passenger is travelling with and pp is their number within the group. People in a group are often family members, but not always.
- HOMEPLANET - The planet the passenger departed from, typically their planet of permanent residence.
- CRYOSLEEP - Indicates whether the passenger elected to be placed into suspended animation for the duration of the voyage. Passengers in cryosleep are confined to their cabins.
- CABIN - The cabin number where the passenger is staying. Takes the form deck/num/side, where side can be either P for Port or S for Starboard.
- DESTINATION - The planet the passenger will be debarking to.
- AGE - The age of the passenger.
- VIP - Whether the passenger has paid for special VIP service during the voyage.
- ROOMSERVICE, FOODCOURT, SHOPPINGMALL, SPA, VRDECK - Amount the passenger has billed at each of the Spaceship Titanic's many luxury amenities.
- NAME - The first and last names of the passenger.
- TRANSPORTED - Whether the passenger was transported to another dimension. This is the target, the column we will be trying to predict.

Almost every column of our dataset has some null values (except PASSENGERID and TRANSPORTED). These values are considered to be unknown and are roughly 2% (around 200 values) for each column, as presented in Table 1. As we can see in this pie chart that nearly 1/4 of the passenger

have at least 1 NaN in their features, with some passenger having 4 NaNs in their features.

TABLE 1. TOTAL NULL VALUES OF EACH FEATURE.

| Feature | Null Values |
|---|---|
| CryoSleep | 217 |
| ShoppingMall | 208 |
| VIP | 203 |
| HomePlanet | 201 |
| Name | 200 |
| Cabin | 199 |
| VRDeck | 188 |
| FoodCourt | 183 |
| Spa | 183 |
| Destination | 182 |
| RoomService | 181 |
| Age | 179 |
| PassengerId | 0 |
| Transported | 0 |

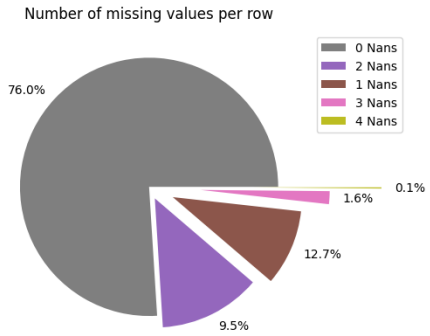


Figure 1. Number of missing values per row.

We can also see that most rows (passengers) don't have any missing data, some have missing data for only one or two of the features, while a very small percentage has missing data for three or four features, as shown in Figure 1.

## 2.2. Evaluation

Lastly, the metric Kaggle decided to use to evaluate performance for competitors of this model was Accuracy, which is given by (True Positive + True Negative) / Total.

Later on we will talk more about metrics when we show the ROC and Precision vs Recall curve.

## 2.3. Data Processing and Feature Engineering

If we look at the explanation of the dataset above, we can see that some of the features might not be very easy to work with the way they are defined right now, and it might be helpful to either transform, split, combine or even create some new features that are easier to visualize and easier to understand their possible relation with our target column.

The final features that we worked on are:

- Splitting PASSENGERID into new features GROUP (first part gggg) and GROUP_ID (second part pp).
- Splitting CABIN (with form deck/num/side) into new features DECK, CABINNUMBER and CABINSIDE.
- Splitting the NAME feature and retaining only the SURNAME (While a given surname won't make you more prone to suffer space anomaly disasters, being a family member of someone that wants to be in a given location of the spaceship might influence whether you get Transported of not).
- Transforming the HOMEPLANET and DESTINATION features into categorical by assigning a categorical value 0,1,2 to each of the 3 different home/destination planet.
- Transforming the AGE feature in to categorical (AGE) by splitting the values into bins corresponding to ranges of 10 years.
- Creating new feature TOTAL corresponding to the total amount spent on the spaceship's amenities and transforming this new feature into categorical (TOTAL) by splitting its values into 10 bins of equal width.
- Maintaining ROOMSERVICE, FOODCOURT, SHOPPINGMALL, SPA, VRDECK as numerical features.
- Maintaining CRYOSLEEP and VIP as boolean fetures.

After the feature engineering, we will have 2 Boolean features (CRYOSLEEP and VIP), 6 Numeric features (ROOMSERVICE, FOODCOURT, SHOPPINGMALL, SPA, VRDECK and CABINNUMBER) and 9 Categorical features (HOMEPLANET, DESTINATION, GROUP, GROUP_ID, SURNAME, DECK, CABINSIDE, AGE, TOTAL).

For the AGE, TOTAL we used both discrete (no_bins) or the continuous form, and we got different results based on this, more about it in later sections.

To finish the data processing we will apply a pipeline that will perform different processing steps depending on the type of feature it is dealing with. For Boolean features it will replace the null values by the most common result (in our case, False for both Boolean features). For Numeric features it will replace the null values with the average value of the feature and the apply a StandardScaler. For Categorical features it will replace the null values with the most common category and then change the category names to numbers (for example, for the feature HOMEPLANET it replaces the null values with Earth and then changes {Earth, Europa, Mars} for {0,1,2}).

## 2.4. Pandas Processing

In order to divide some of the columns, like it is shown above, we have created a function that does the split of the mentioned columns, the sum of total spend, and the discretization of age and total. We considered placing this function as one of the steps of the pipeline (see next section) but ended up leaving it a function, since it is simpler.

So for the train data, we apply this pandas processing function, just like we did for the Train, this function will

not change any values, other than just convert 2 columns to discrete values.

## 2.5. Pipeline

To simplify all of these feature engineering, we decided to use pipelines: function inside of sklearn that allow us to quickly define a pipeline and then abstract from it. Besides, this also allows us to use the ”fit”, ”transform” and ”fit_transform” of the pipeline. With this we can save the pipeline already transformed and later on use it to predict the y.

Since we have 3 types of data, boolean, objects (will be converted to categorical) and numerical ones, we defined one subpipeline for each. The Boolean subpipeline is the simplest, we use a simpler imputer with strategy equal to the most frequent, so any missing values or Nan will be replaced for the most common value in that column. The objects pipeline will do the same, but after wards it includes a ordinal encoder, so that every class in no longer an object (string) but a encoded class represented by a integer (like we have explained on the previous subsection), every new string that the encoder has never seen before will be transformed into a -1. Lastly for numerical columns, we will replace the missing/nan values by the median of the column, and then apply a standard scale to the column, so that all the numeric values have similar means and variances.

We joined this 3 subpipelines into one pipeline that we have called preprocessing_pipe. The main advantage of this method is that we can now call fit to any new data, and it will be scaled and encoded using the training data. This guarantees that the data is treated equally, and all the transformations applied to the train dataset will also be applied to the test dataset.

## 2.6. Data Visualization

Now that we have the features we want to work with, it may be important to try to visualize not only them but their possible relation with our target Transported. For the Boolean features and some of the Categorical ones, we can use bar plots of that feature and compare the results between passengers who were transported and passengers who weren't, as shown in Figure 2. We can also use more detailed bar plots paired with box plots to visualize Numerical features, like the AGE (before processing, as seen in Figure 3) and the TOTAL spent (Figure 4).

Looking at the graphs presented, it is very hard to see any difference between the passengers that were transported and those who weren't for any of the features (except maybe CRYOSLEEP). These types of graphs sometimes allow us to quickly delete some features, especially ones where the difference for the transported and not transported is very small, analyzing the Figure 2, the VIP seems to be one of the features that we could delete, we decided to keep it still, since most models can change the weight for it.

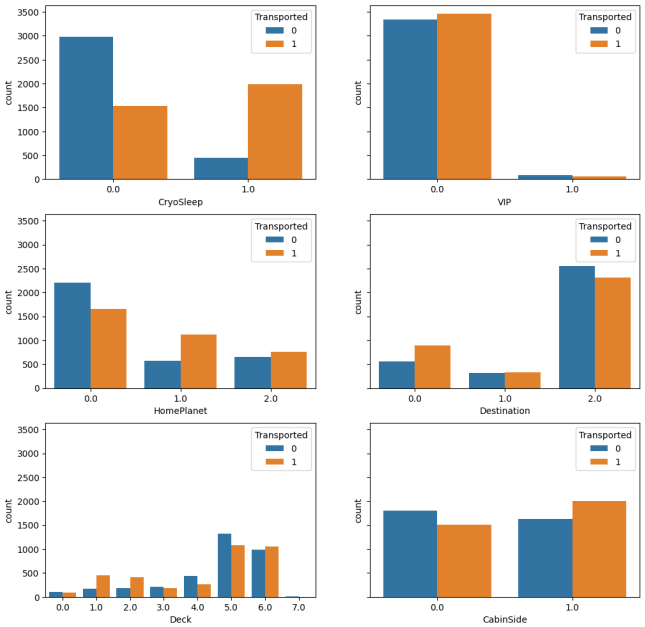


Figure 2. Bar plot for various categorical features for Transported and not Transported passengers.

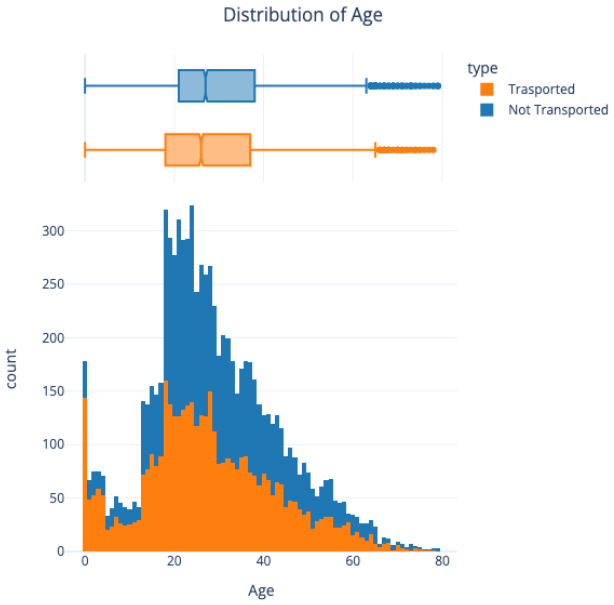


Figure 3. Box plot and histogram of the ages for Transported and not Transported passengers.

## 2.7. Correlation

As we can see in Figure 5 there is not one variable that has a high correlation with the Transported column. Also we can see that, aside from the HOMEPLANET to DECK, the entire matrix shows very small correlations between the features, so even the features that we created are not

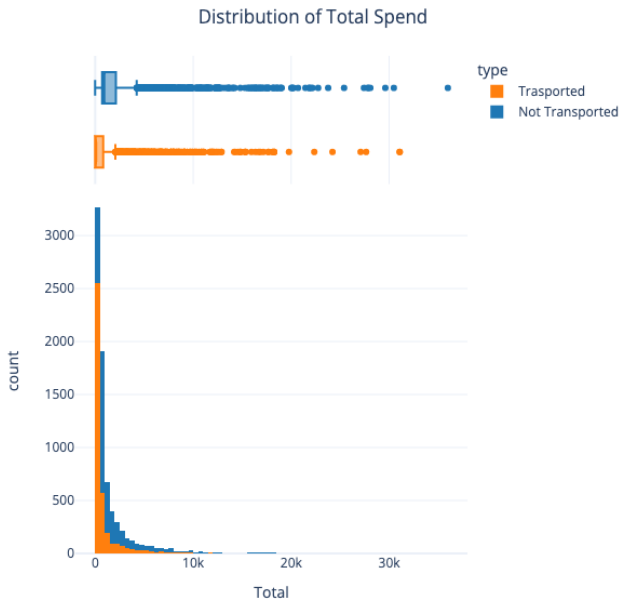


Figure 4. Box plot and histogram of the total spend for Transported and not Transported passengers.

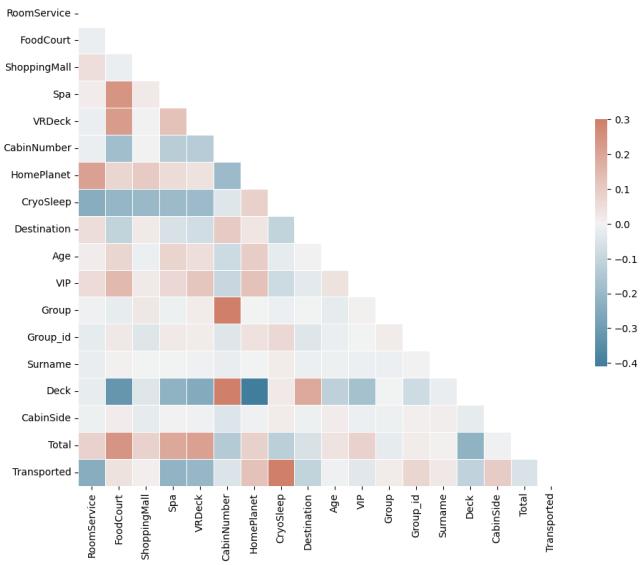


Figure 5. Correlation between all the variables that we use.

that correlated with the original features, suggesting that the features that we created might give some extra insights to the problem. Creating more low correlation features might improve our result even more.

In an attempt to better visualize some of the pairs of features that had a higher correlation, we created the Figure 6. Here we can see that this pair of features almost separates the entirety of transported and not transported passengers (this is just one the graphs that we can see in 7).

Looking at the full pairplot of the continuous variables

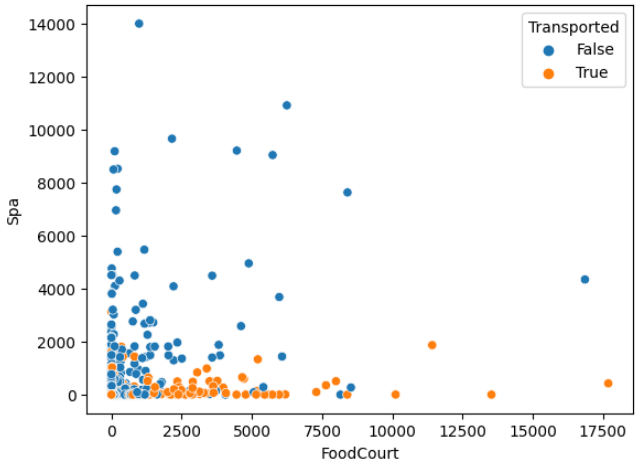


Figure 6. Transported passenger using FOODCOURT vs SPA.

(Figure 7), we can see that the separation between the two categories is not that simple, suggesting that will need models that are more complex than just a linear regression. Besides, we can also see that we don't have any numerical variables that are a clear function of another variable, meaning we shouldn't remove any features, since those effects can't be justified by another feature.

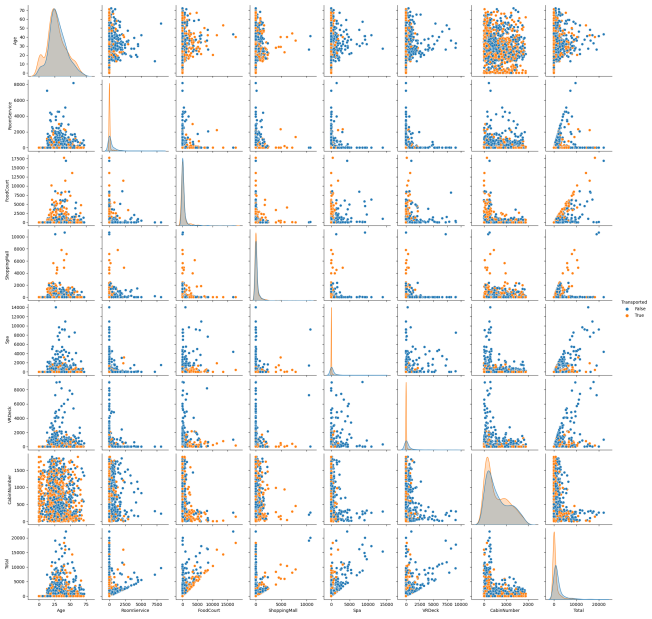

Figure 7. Pair plot for the continuous variables.

## 3. Models

Using the Lazypredict library [2], we can see from Figure 8 that the best models are SVC, NuSVC, Extra Tree Classifers, Random Forest, LGBM and XGBoosting Classifer (Note that Figure 8, doesn't include all the graphs).

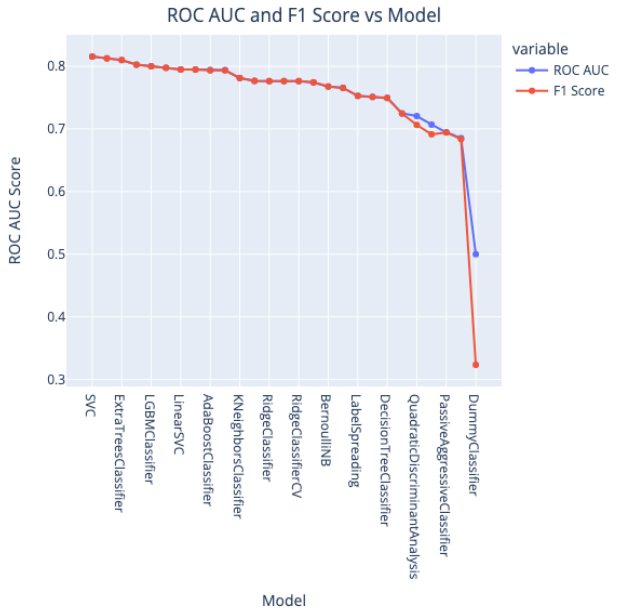


Figure 8. A lazy predict for the models, in function of the ROC and F1 Score.

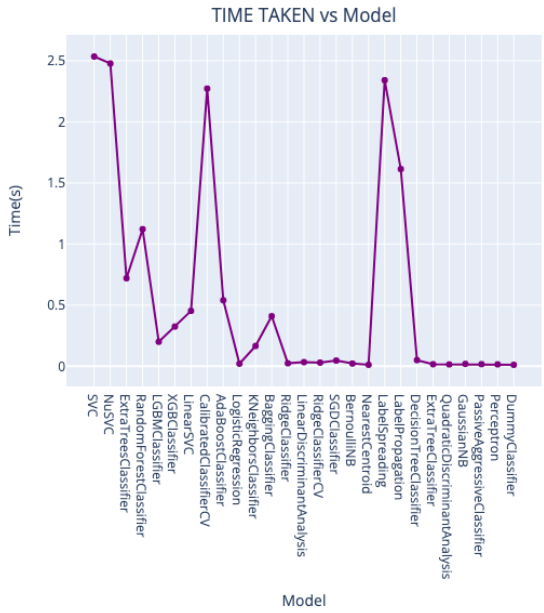


Figure 9. The time elapsed for every model for the Lazy Predict.

We decided to not use the NuSVC and the Random Forest since they are very similar to the SVC and the Extra Tree Classifier, respectively.

Here the Lazy Predict used the full data from the train.csv that we have, all the other tests that we did with the models were done only after splitting the train dataset into train and test using cross validation (CV) on the train, so dividing our train dataset into multiple parts and then using one for evaluation and other for training, changing the one used for training, testing all the parts as the training subset and then averaging the results.

This way we are reserving the test dataset for the final evaluation, it is also worth noting that kaggle gave us both the train.csv and the evaluation.csv (the correct name is test.csv, but we have renamed it to avoid confusion), where the evaluation.csv doesn't have any label, the true labels are only known by Kaggle, this way they have a fair way to evaluate the models.

## 3.1. SVC

This was the model that, with default settings, got us the best result according to the Lazy predict. It is a classifier that comes from the family of the Support Vector Machines (SVM). The main objective is to find a hyperplane that can separate the data in the feature space. It is a quite famous model that tends to produce good results, despite being computationally efficient, it can be quite heavy to train. Around 5 to 10 years ago it was common to see this model in some kaggle competitions, in our days is becoming less and less common, especially for competitions that belong to the Getting started, specially because it takes a long time for it to train and parameter search.

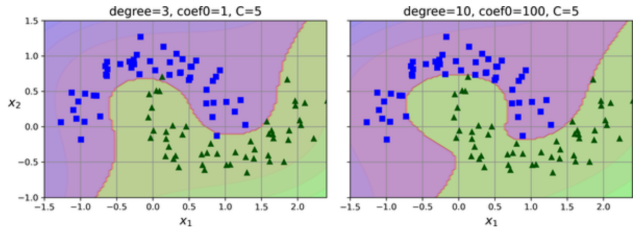


Figure 10. Example of a Polynomial Kernel, extracted from [3].

Figure 10 shows one example of a Polynomial kernel, allowing us to see the effects of the polynomial degree. As we can see the image on the left has a lower degree so the curves are less "pronounced" when comparing with the image on the right. As we know, the degree contributes a lot to more complexity, and a very high degree tends to lead to overfitting.

For larger datasets the recommendation of SKlearn is to choose the Linear SVC or the SGD Classifer, both also represented on the models list that we trained using the Lazy Classifiers (visible in Figure 8), and had a score of 0.79 and 0.77 respectively, so we decided to stick with the SVC. It is also worth noting that this is one the models that takes the most time to train as we can see in Figure 9.

For our model we started by assuming a polynomial kernel (has more parameters than the other kernels) and we tested for a few degree levels and for a coef0 (independent term of the poly and sigmoid kernels) and C (regularization function). To do this search for parameters we have used HalvingGridSearchCV, with 5 cross validations, and scoring

for the accuracy. After some iterations we obtained the model SVC(C=11, kernel=poly, degree=7 and coef0=5). We also decided to test for different types of kernels, and the second best that we got was the linear one with the accuracy of 75%, so the poly kernel is a clear winner.

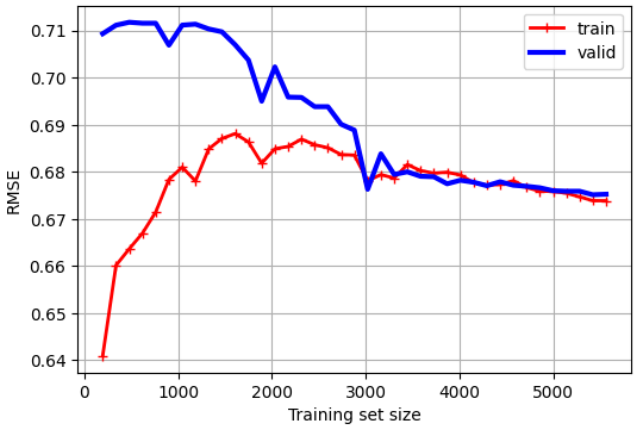


Figure 11. The learning curve for the SVC model.

As we can see in Figure 11 as we increase the size of the training set, both the error of the train and validation start coming down, this is what we expect since more training allows for better definition of the limits of the feature space, so the separating plane is more accurate with more data.

Due to the time it takes to make any change to this models, we have decided to move into our second best model from the Lazy Predict, the Extra Tree Classifier. Also just by checking the RMSE and comparing it to teh next sections, this leads us to loose even more confidence on the Lazy Predict.

## 3.2. Extra Tree Classifier

This is a ensemble model and it's a type of random forest (ensemble of decision trees) but with a lot more randomness in inserted to the model. Basically, every feature can use different random thresholds, resulting in extremely randomized trees (or for short extra-tree), hence the name. The name XTRee is also quite common for this type of model, and during Tables we will refer to it using Xtree.

After we did all the training that it requires we were able to reach Figure 12, on this four part figure we can see at the top the confusion matrices, in the left the un-normalized and the most common one, and the normalized one in the right. This was made using cross validation prediction, so it was always evaluated on data that it had never seen before. We can see that it predicted True for True cases 84% of the time, and it predicted False for False cases 77% of the time. On the bottom part of the figure we can see the Precision Recall curve. Recall refers to the percentage of total relevant results correctly classified by your algorithm, while precision is the percentage of results which are relevant. So a algorithm that predicts all the events as True has a 100% recall, with 50%

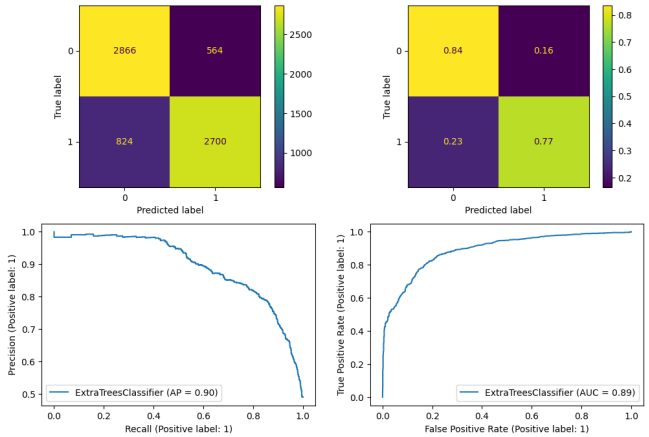


Figure 12. The confusion matrix (normalized and not normalized), the ROC curve and the Precision recall curve.

precision, so it is possible to increase precision or recall at the cost of the other. This is one of the reasons why in Kaggle competitions we tend to not evaluate these 2 metrics but other ones like f1 score or accuracy (like this case) or even custom metrics. Lastly at the bottom right corner we have the ROC curve, this curve relates the True positive rate with the False positive rate. The area under the curve (AUC) is also one of the metrics that can be used to evaluate a model, a model with 100% of correct predictions will have an AUC of 1, while a model that predicts 0% of the results correctly will have an AUC of 0.

Like we did to the previous model, we started by trying to improve the 0,80 F1 score that we obtained with the Lazy Predict. Again, we used a HalvingGridSearchCV, this time covering the number of estimators (n_estimators), the max of features that can be used, and the minimum impurity that must be decreased at every step of the way.

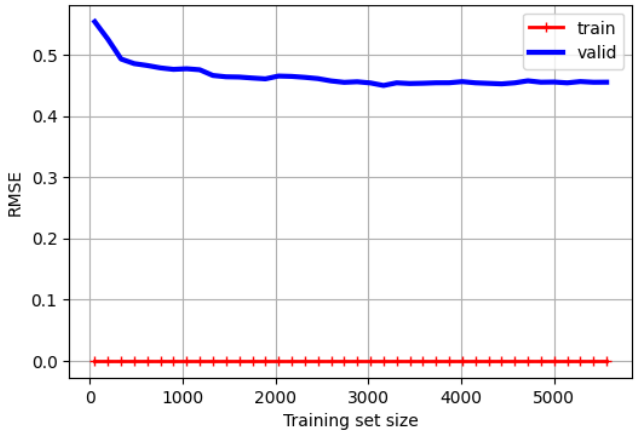


Figure 13. The learning curve for the Extra Tree Classifier model.

We also decided to repeat the Learning curve for the Extra Tree classifier, that we did for the SVC, obtaining Figure 13. Here we can see that the error in the train dataset

in always zero, so if we evaluated using the training dataset the result would be perfect. Of course, this is not the case for the validation curve. We can also see that the RMSE of the validation tends to plateau quite fast, so the Extra Tree Classifier was able to quickly determine the structure that it needed to predict, and after roughly 2000 of the training set size it reached the minimum RMSE that it was able to obtain.

With the best model that we obtained, we were able to obtain the confusion matrix visible at Figure 12. This model has a number of features of 750, a max features of log2 of the initial training size and a min impurity decrease of 0.0.

### 3.3. XGBoost

XGBoost is a open source predictive machine learning library based on the paper "Greedy Function Approximation: A Gradient Boosting Machine", by Friedman [4], and document at [5]. It is a high-performance implementation of gradient boosted decision trees, considered by many one of the most accurate models. Besides, due to the fact that it quite well optimized, it makes it a very good initial model. It also belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core.

For comparison, we can obtain similar graphs to the previous models for the XGBoost, resulting in Figures 14 and 15.

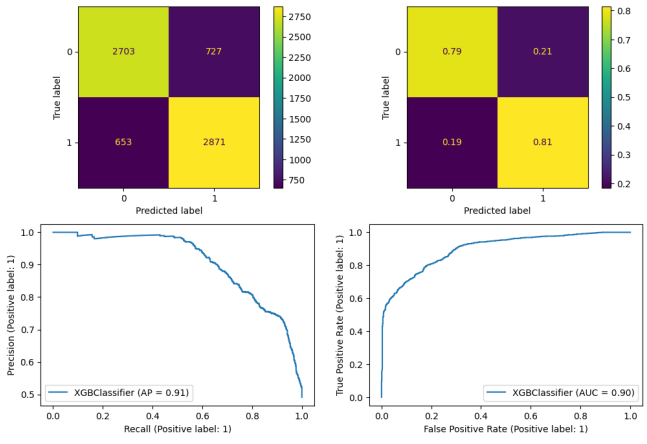


Figure 14. The confusion matrix (normalized and not normalized), the ROC curve and the Precision recall curve for the XGBoost.

We can see that the results obtained are overall very similar to the one obtained using the XTree classifier.

XGBoost is a model that is rather frequently on the top of the Kaggle competitions, being described as "The most accurate modeling technique for structured data.", with a dedicated class from their courses to it https://www.kaggle.com/code/alexisbcook/xgboost. Despite all of this, the documentation can also be quite daunting, as it has many possible variables to a new user.

Besides all this advantages it is a model that was made be hard to overfit with, so most times its learning curves

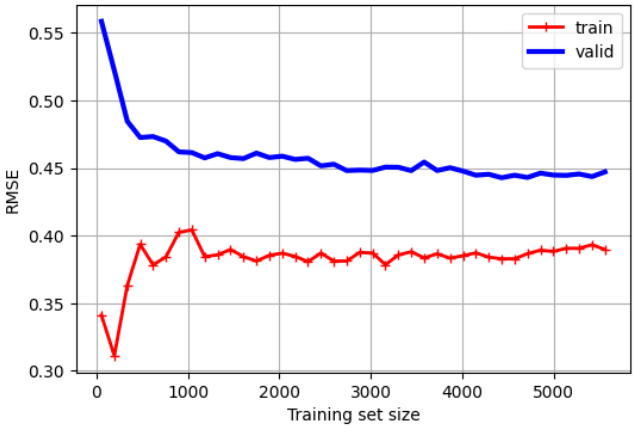


Figure 15. The learning curve for the XGBoost model.

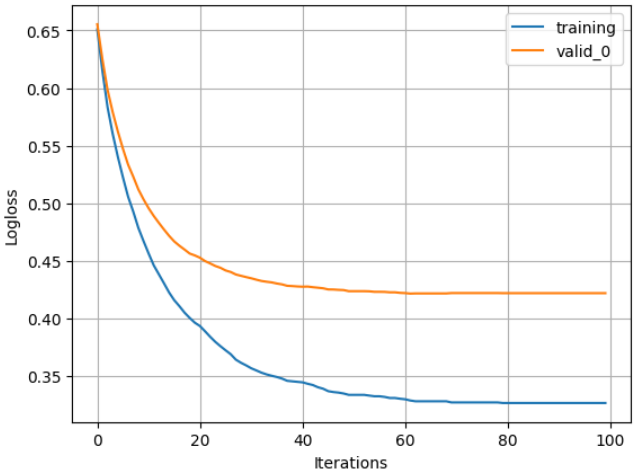


Figure 16. The learning curve for the XGBoosting model.

are always decreasing, like the one that we obtained for our XGBoost model in Figure 16.

### 3.4. LGBM Classifier

The LGBM Classifier has recently gained a lot of popularity, it is also a gradient boosting framework (just like XGBoost). It is slowly stealing the spotlight from XGboost, as it was created by Microsoft and it is also quite daunting by looking at the documentation, present at [6].

On Figure 17 and Figure 18 we can see the confusion matrix, ROC curve and Precision vs Recall curve and the learning curve respectively. Just by looking at all the Confusion matrix, we can see that this is the best one that we got so far, so this leads us to think that this will be the best model from all that we have trained so far. The AUC curve continues to be in the 0.90 just like we had on Figure 14.

Lastly for the learning curve this is one of the most interesting ones, where the error of the training is increasing as the training size increase, while the test error is decreasing.

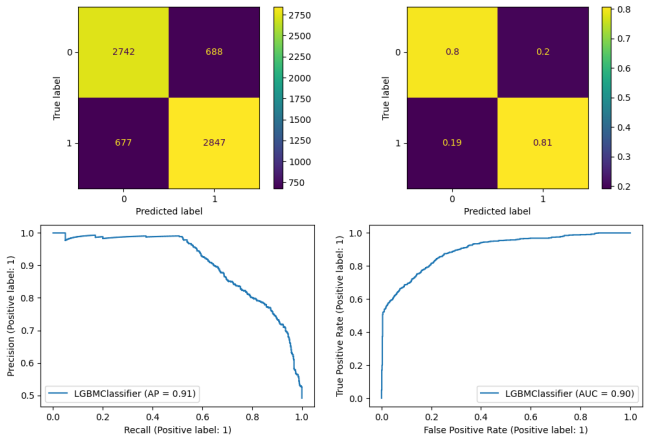


Figure 17. The confusion matrix (normalized and not normalized), the ROC curve and the Precision recall curve for the LGBM.

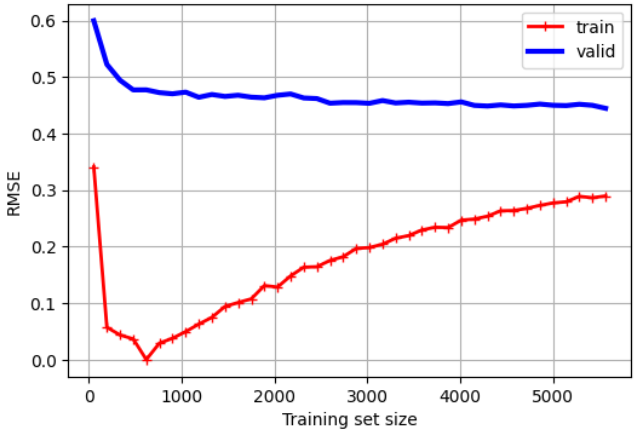


Figure 18. The learning curve for the LGBM model.

Here we we can see that the learning curve is not that simple as the learning curves for the XGboost, this is one of the cases where it makes sense to do a early stopping, on the point where the error of the valid starts to grow again.

## 3.5. CatBoost Classifier

Lastly we decided to also experiment with Cat Boost Classifier, this is probably the most recent library from all the ones that we are considering, it has the doumentation present at [7],

On Figure 20 and 21 we can see the already common to this work, The confusion matrix (normalized and not normalized), the ROC curve and the Precision recall curve and the learning curve respectevily. Just by doing a quick comparison we can see that this one of the highest ROC that we have so far, and it is also one of the best confusion matrix, suggesting that this might be our best model yet.

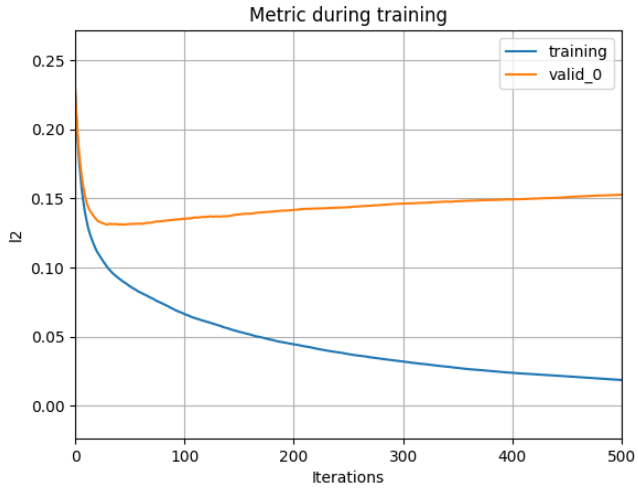


Figure 19. The learning curve for the LGBM model.

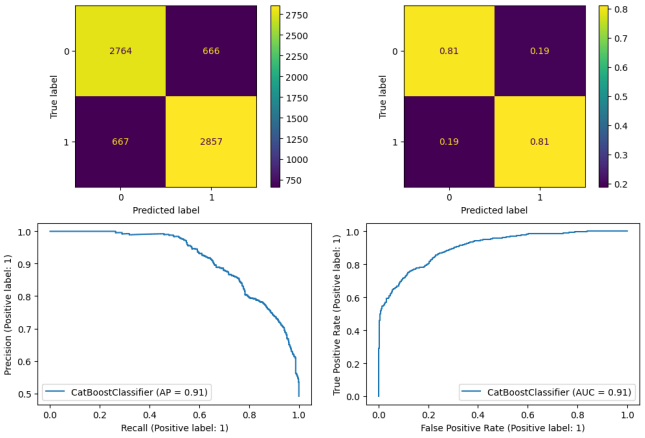


Figure 20. The confusion matrix (normalized and not normalized), the ROC curve and the Precision recall curve for the CatBoost.

## 3.6. Model comparison

Overall, most of the results obtained in our confusion matrices were quite similar, with LGBM and Cat Boost appearing to be overall slightly superior. It is worth noting that sometimes we might choose sub-optimal models, because we are looking for the ability to explain and understand, or because we want to maximize a different metric, other than just accuracy. For example, for a cancer detection we might want to lower false positives results, so maximize precision, or in a case where we prefer to analyze some extra noise at the cost of not loosing any important object, cases where we want to maximize recall.

One quite interesting quality of random forest (all the models that we have selected are from this family) in general is that they measure the relative importance of every feature, which is quite useful for feature selection. For example if we wanted to do a bigger and more in-depth study using the SVC, we could decrease the computational needs of that

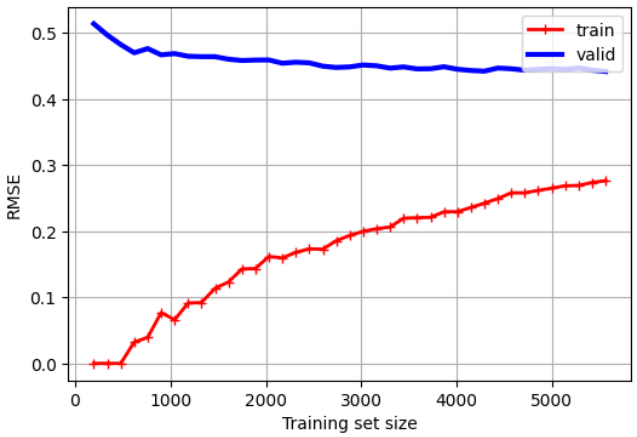


Figure 21. The learning curve for the CatBoost model.

TABLE 2. IMPORTANCE FOR EVERY VARIABLE ACCORDING TO THE EXTRA TREE CLASSIFIER, XGBOOST, LBGM AND CAT BOOST.

| CAT Boost | LBGM | XBG | Xtree | Variables |
|---|---|---|---|---|
| 5.4% | 1.4% | 48.2% | 17.4% | CryoSleep |
| 8.9% | 12.8% | 2.1% | 9.3% | CabinNumber |
| 5.8% | 11.2% | 1.9% | 9.0% | Group |
| 5.3% | 11.9% | 1.1% | 8.0% | Surname |
| 9.4% | 9.3% | 5.6% | 6.9% | Spa |
| 5.8% | 4.3% | 2.5% | 6.5% | age |
| 8.3% | 8.9% | 4.8% | 6.3% | VRDeck |
| 6.8% | 7.0% | 7.7% | 6.2% | RoomService |
| 5.9% | 8.1% | 3.6% | 5.9% | FoodCourt |
| 11.0% | 6.8% | 3.0% | 5.4% | Deck |
| 3.5% | 6.9% | 3.0% | 5.1% | ShoppingMall |
| 10.7% | 2.7% | 8.3% | 3.7% | HomePlanet |
| 1.3% | 2.0% | 2.1% | 3.5% | Group_id |
| 2.2% | 1.9% | 1.5% | 2.4% | Destination |
| 5.2% | 1.5% | 1.8% | 2.2% | total |
| 4.3% | 3.2% | 2.6% | 1.8% | CabinSide |
| 0.1% | 0.2% | 0.0% | 0.4% | VIP |

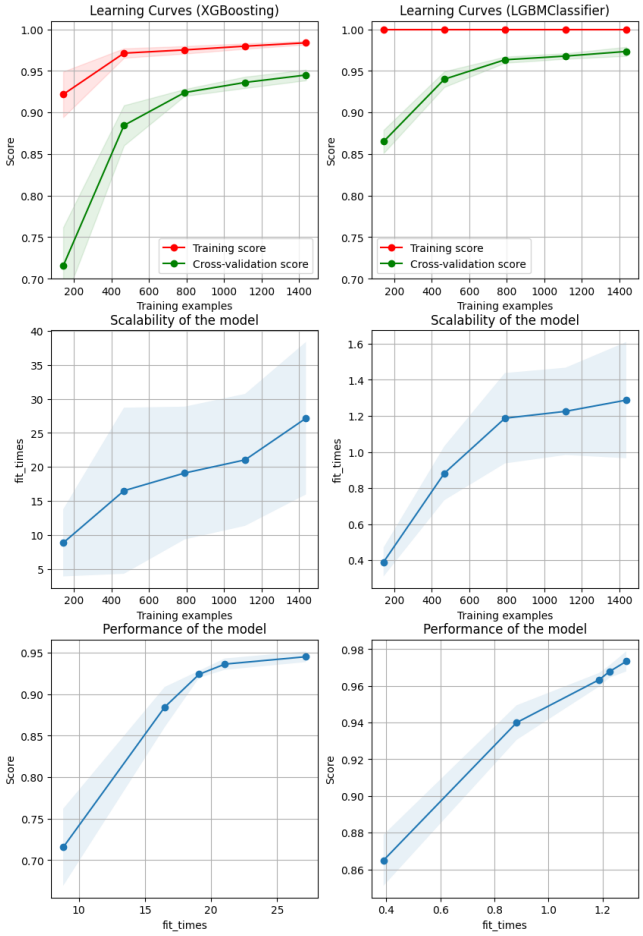


Figure 22. Poly SVC vs ExtraTree Classifier.

model by using the information from Table 3.6 to select only the most relevant features. For example, if we had to reduce some of the features, the VIP would be the first that we would remove from the dataset, while the CryoSleep seems to be the most important, at least for our Extra Tree classifier.

We decided to also include on this Table (3.6) the importance for the XGboost and LGBM too, here we can see that VIP was not used by XGboost at all. The Table is ordered in descending order of importance for the Xtree Classifier, it is curious that the order is not maintained to the other two models, not even between them.

In our case, assuming we want to rescue both the passengers that remained in the ship as well as the ones transported to another dimension, we might be interested in false positive and false negative, so accuracy is a good metric, besides, this is also the metric that Kaggle will use to classify the results at the end.

In Figure 22 and Figure 23 we can see the summary of the 4 models that we have used. In terms of the learning curves vs Training size. The XGBoost was the unique model where the training score was not 1 at the begining, so with lower training set it can't even have a perfect accuracy, and around the 1200 it continues to be the worst model (by comparison with the others), while the other 3 are quite similar between them, so if we could only work with a subset of our training dataset this ExtraTree Classifier would perform better than the XGboosting.

Discussing the middle part of the Figures, we can see the scalability of the model, here the y scale is quite different between them, as you can se ethe XGboosting is the one with a higher y scale, going to almost 40 fit_times this justifies in part why is this model so slower when compared to the other ones, on teh otehr end of the spectrum we have the LBGM, which had 1.6 fit_times at the most, once again consistent with our training times, while the Cat Boost and Extra Tree Classifier are very similar.

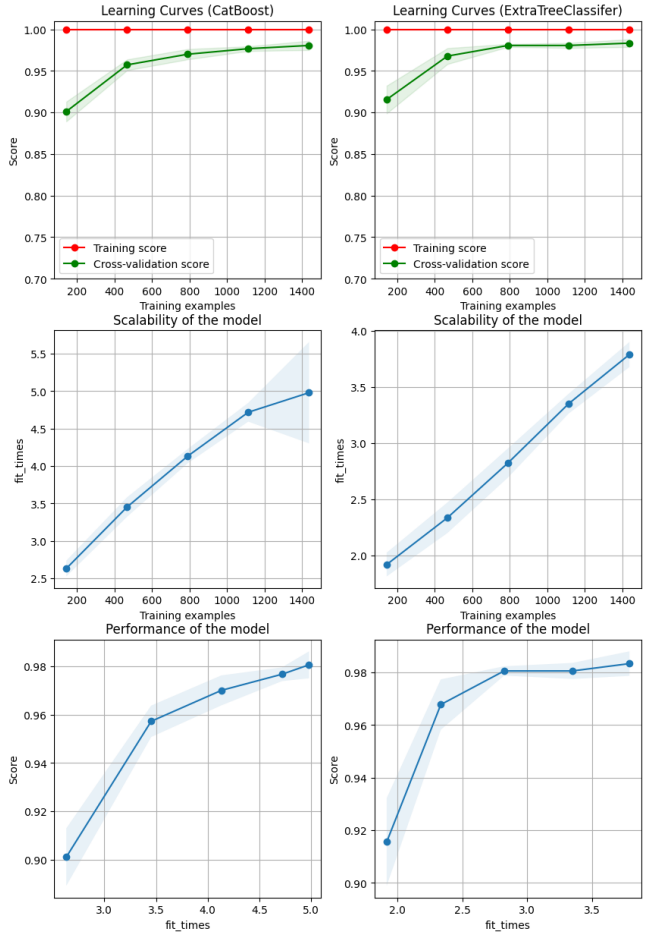


Figure 23. XGBoosting Classifer vs LGBM Classifer.

# 4. Kaggle

## 4.1. Other Kaggle submissions

Since this is a ongoing competition and there are a lot of submissions, we decided to focus on just a few that we found most interesting.

**4.1.1. Spaceship Titanic: A complete guide.** This notebook [8] contains a very detailed data analysis and very interesting idea of using log transform to the 5 spend categories, given their distribution (similar to total spend in Fig 4). Also contains an interesting idea of combining 2 classifiers and using cross fold validation to obtain a probability of a given passenger being transported which results in a graph of predicted probabilities (confidence of prediction), and then calculating a threshold for the probability to be assigned True or False to transported column based on the assumption that the train and test datasets have the same distribution.

It is also worth revisiting the figure 8, where we had a higher than 81% with the base SVC model, it had a score of 82%, so some of our feature engineering are producing features that can better predict, or the author didn't do a good scanning of possible models (since SVM tend to be quite slow, like we have discussed previously, they tend to be ignored). This notebook was awarded a Gold medal according to Kaggle.

TABLE 3. OUR SUBMISSION TO KAGGLE.

| Model | no bins | Score | Leaderboard | Top % |
|---|---|---|---|---|
| CatBoost(depth=7 iter=300) | X | 0.80874 | 167 | 7 % |
| CatBoost(depth=6 iter=600) | X | 0.80547 | 315 | 14 % |
| LGBM(dart, 500) | X | 0.80336 | 475 | 21 % |
| LGBM | - | 0.79985 | 720 | 31 % |
| XGBoost | - | 0.79915 | - | - |
| XTree | - | 0.79565 | - | - |
| XTree | X | 0.79401 | - | - |
| XGBoost | X | 0.78863 | - | - |

**4.1.2. (0.81669) misaelcribeiro solution+Modularity FE.** On this notebook [9] the user just created a the total sum of spending like we did, but instead of then passing it into discrete intervals, he submitted it just like this. He used a Cat Boost Classifier and with it was able to obtain a 0,8169 score. This is the best score on public notebooks, rendering the author a spot on the leaderboard, and also the notebook itself was evaluated with a Silver medal.

**4.1.3. CatBoost — EDA + Feature engineering.** Lastly, on this particular notebook [10], they made a very good analysis by category, dropping some that graphically didn't seem too relevant, while also creating new features. For example they created a feature called 'is_solo'that controls if a user is flying solo or not, and apparently flying solo is a lot safer for this space anomaly.

The feature engineering here, despite being too graphical and subjective was quite in depth and interesting. Their public score is the exact same as ours, so there's a lot to learn from their work.

## 4.2. Kaggle submission

On Kaggle we are allowed to 10 submission per day, we decided to do introduce our own submissions, the results can be found in 4.2. The scripts that have 'no_bins' in their parameters, didn't had the binarization of Age and Total Spend during the pipeline.

We decided to test no_bins for all the models, it was a suprise for us both, that the no_bins that produced the best results also resulted in the worst results that we got, suggesting that probably Xtree and XGboost are to sensitive to this extra computation from the original dataset, while the CatBoost and LBGM actually preform better without the descritization.

# 5. Conclusions

The best Kaggle submission has 0.87654 of accuracy suggesting that we could still improve our models by quite

a lot. Still this should not discourage us, since our models actually got us close.

The feature engineering that we did was refreshing, most of the real competitions on kaggle already come with data prepared, without missing values and with little to no information in between lines, here we had the opportunity to build a few features based on others, and seeing other works suggested that we could have done even more of this, like IS_SOLO that [9].

Our experience with Lazy Predict was not the best, it gave us a initial direction to choose the models but the results don't inspire much confidence, with it we saw that the best results came from the SVC and XTree Classifier, and on the submission to Kaggle and on Train/test evaluation we saw that was not completely right. They probably cut some corners in order to make a quick analysis, these cuts might make a difference in competitions like this, where every single 0,01 makes a difference. This is also how they managed to do 29 models in less than 20 seconds. Overall we understand now why the library is not that used.

Our main goal was just to compete, to place in practice some of the models that we have developed during classes and explore a problem for ourselves. Also the idea our processing of features and feature engineering was interesting and allow us to produce some good results but a lot more about them could have been explored, we were constantly going back and forth testing new ideas and new features. We tried to come up with our own features, but just from a small search we can already see a lot of creative ways to express the features that we had, and the ones that we could have created.

The ability to reach the top 7 % was unexpected for both of us, we wanted to participate on the competition from the beginning, but we were not expecting such a good result, with such a minor parameter search, and without using any type of neural network. Still it is a good boost of motivation to keep dwelling between models and exploring data sets.

## References


[1] Kaggle, "Spaceship titanic," https://www.kaggle.com/competitions/spaceship-titanic/.

[2] "Lazypredict library," https://github.com/shankarpandala/lazypredict.

[3] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[4] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[5] "Xgboost documentation," https://xgboost.readthedocs.io/en/stable/.

[6] "Light gbm documentation," https://lightgbm.readthedocs.io/en/v3.3.2/.

[7] "Cat boost documentation," https://catboost.ai/.

[8] Samuel Cortinhas, "Spaceship titanic: A complete guide," https://www.kaggle.com/code/samuelcortinhas/spaceship-titanic-a-complete-guide.

[9] Jim Liu, "(0.81669) misaelcribeiro solution+modularity fe," https://www.kaggle.com/code/jimliu/0-81669-misaelcribeiro-solution-modularity-fe.

[10] "Catboost — eda + feature engineering," https://www.kaggle.com/code/yefimsokolov/catboost-eda-feature-engineering/notebook?scriptVersionId=110485775.