Machine Learning Project 2023

Kaggle competition: Spaceship Titanic

1 Team composition

In this Kaggle competition, our team is named **"PJoBoDo"**, which consists of 4 members:

- LE Do Thanh Dat
- ZARATE GUEVARA John Sabastian
- YOU Borachhun
- KHENG Piseth

2 Exploratory data analysis

2.1 General information

This Kaggle competition aims to predict whether passengers on a spaceship were transported to an alternate dimension. Two datasets, a training set and a testing set, are provided, each containing 13 explanatory variables. The training set includes 8693 observations, while the testing set has 4277 observations. The explanatory variables are:

- `PassengerId`: the unique ID of each passenger, in the form `gggg_pp` where `gggg` indicates a group and `pp` indicates a number within the group
- `HomePlanet`: the departing planet
- `CryoSleep`: indicates whether the passenger is in cryo sleep
- `Cabin`: the cabin number of the passenger, in the form `deck/num/side`, where `side` can be `P` or `S`
- `Destination`: the destination planet
- `Age`: the age of the passenger
- `VIP`: indicates whether the passenger has paid for VIP service
- `RoomService`, `FoodCourt`, `ShoppingMall`, `Spa`, `VRDeck`: amount the passenger has billed for each amenities
- `Name`: the first and last name of the passenger.

Both datasets come with the explanatory variables, whereas only the training set includes the target variable, `Transported`, which indicates whether the passenger was transported to an alternate dimension.

In addition, we see that the features of the datasets contain two types of data: numerical data and categorical data. Since certain algorithms are only capable of processing numerical data, we would need to use encoding methods to convert all categorical features to numerical features.

2.2 Checking null values

We notice that there are missing values contained in the datasets. It is important to keep in mind that some machine learning algorithms cannot handle such values and will produce errors or incorrect results if the data contains missing values.

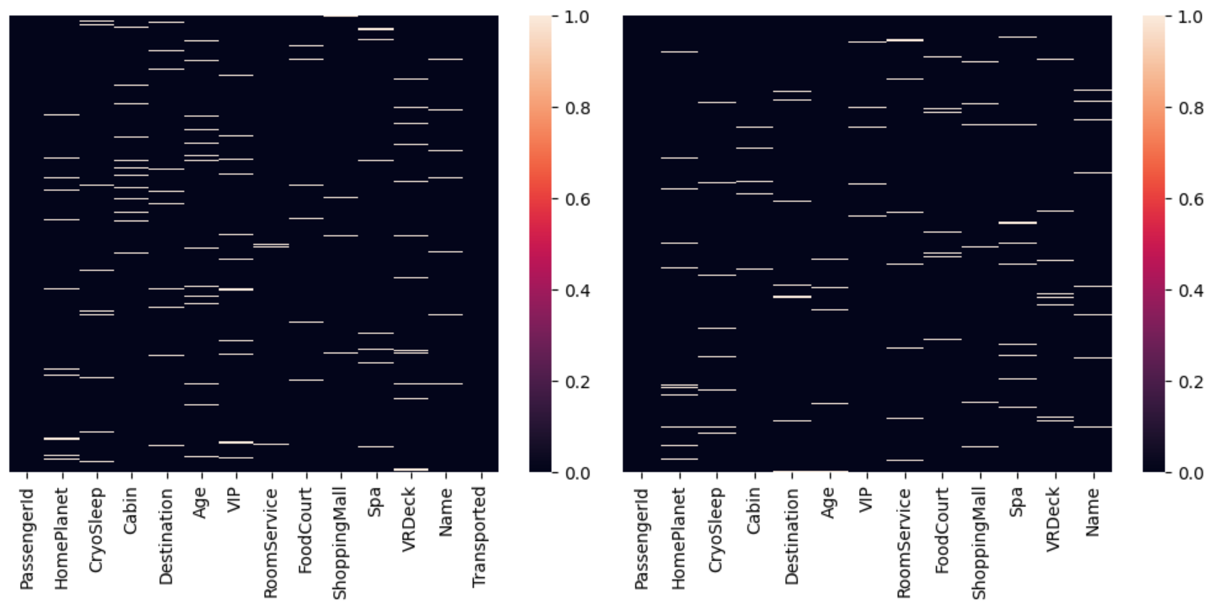


Figure 1: Missing values in the training set (left) and the testing set (right)

Figure 1 illustrates missing values in the training and testing dataset. We can clearly see that all the features contain missing values, except `PassengerId`. Table 1 shows the number of missing values for each feature in both datasets.

Feature	**Missing values**	
	Training set	**Testing set**
`PassengerId`	0	0
`HomePlanet`	201	87
`CryoSleep`	217	93
`Cabin`	199	100
`Destination`	182	92
`Age`	179	91
`VIP`	203	93
`RoomService`	181	82
`FoodCourt`	183	106
`ShoppingMall`	208	98
`Spa`	183	101
`VRDeck`	188	80
`Name`	200	94

Table 1: Number of missing values for each feature in the training and testing dataset

We discover that in total the training dataset has 2324 missing values whereas the testing dataset has 1117. Further analysis reveals that these missing values are present in 2087 rows (observations) and 996 rows for the training and testing dataset, respectively. In fact, if we remove all the missing values, it may result in imprecise statistical analysis and a biased machine learning model.

2.3 Data analysis

In this section, we would like to analyze some features in order to have greater precision in performing the imputing process. Firstly, we start by analyzing the data of categorical features.

After exploring the categorical features, we figure out that `HomePlanet` consists of three unique values: `Europa`, `Earth` and `Mars`, while `Destination` has three distinct values: `TRAPPIST-1e`, `PSOJ318.5-22` and `55 Cancri e`. `VIP` and `CryoSleep`, on the other hand, has two unique values: `True` and `False`.

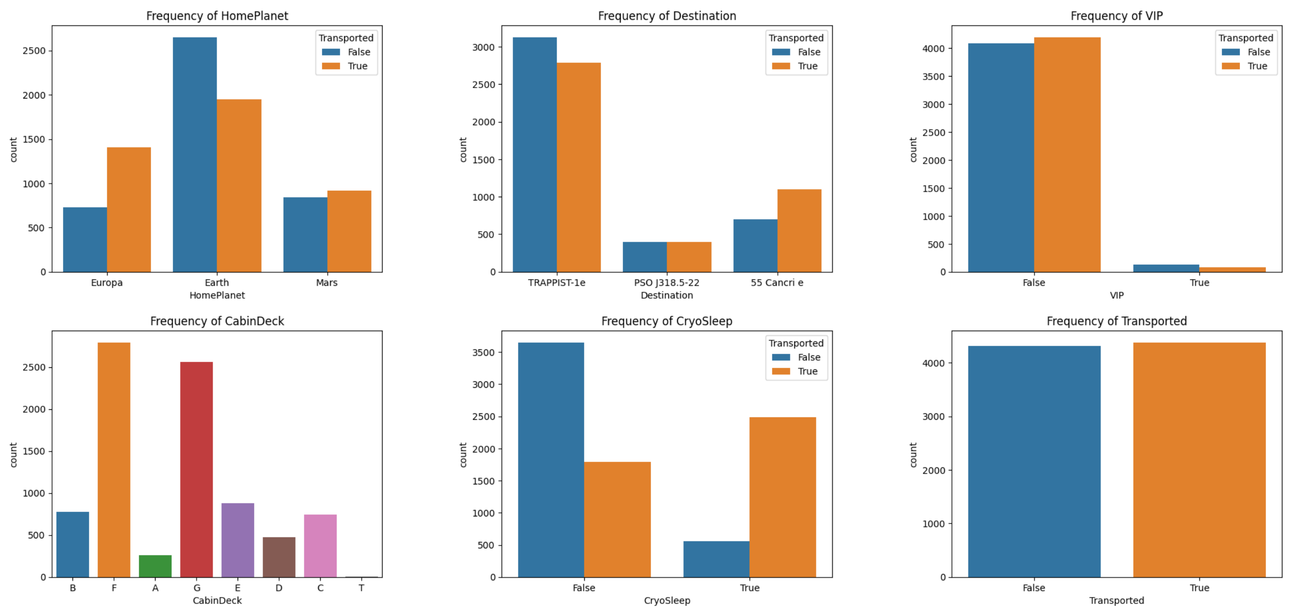


Figure 2: Frequency of categorical features in the training dataset

According to Figure 2, we can see that there is some information about the categorical variables:

- Passengers are mostly from `Earth`.
- Passengers are more likely to not take the `CryoSleep`.
- The destination `TRAPPIST-1e` is more likely to be the choice of passengers.
- Most passengers do not have VIP status.
- `CabinDeck F` and `G` have a high number of passengers, while `CabinDeck T` has only a few.
- There is a balance between the number of passengers who are transported and those who are not in the dataset.

Additionally, we aim to deduce insights from the numerical features, which we represent through histograms to illustrate the distribution of the numerical variables.

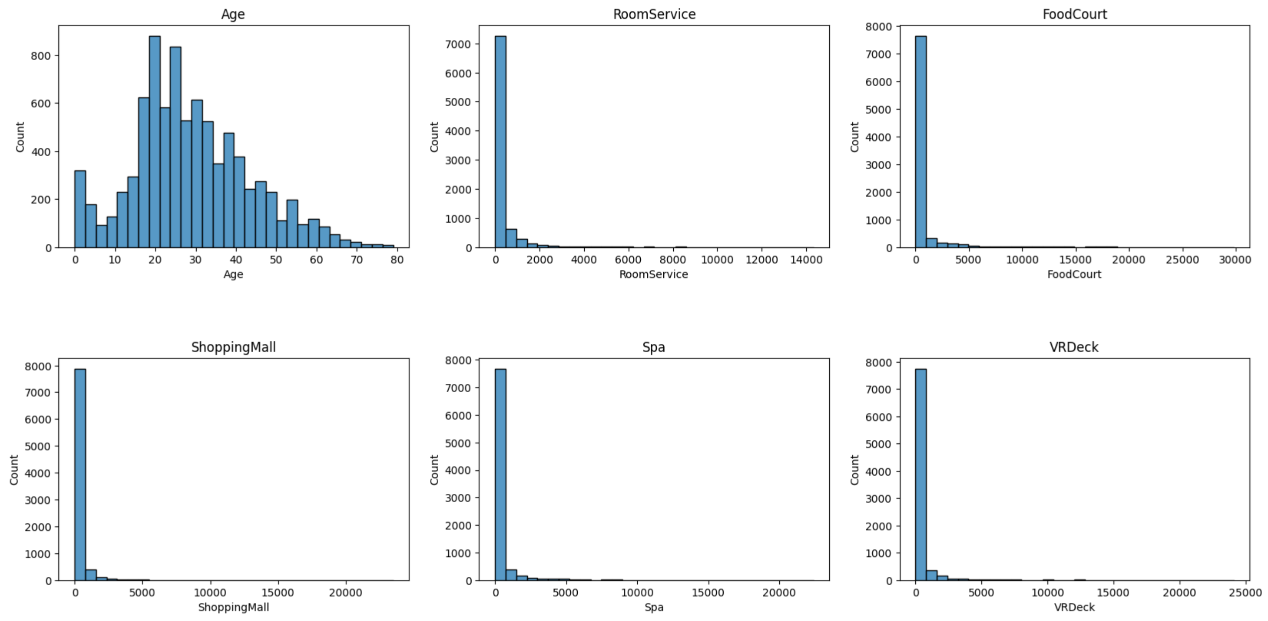
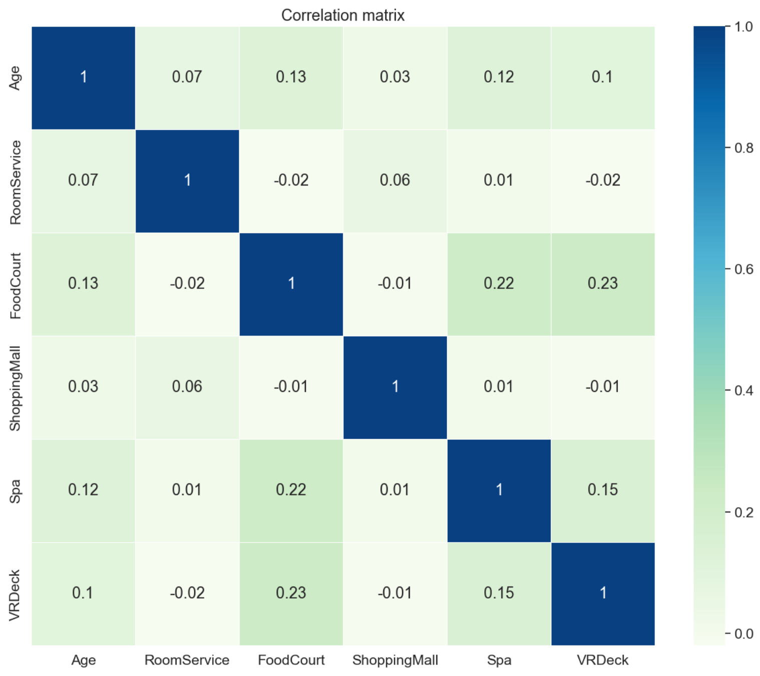


Figure 3: Distribution of numerical variables in the training dataset

Based on Figure 3, it is apparent that the distribution of all numerical variables does not follow a normal distribution. Furthermore:

- The expenses for `RoomService`, `FoodCourt`, `ShoppingMall`, `Spa`, and `VRDeck` have a skewed distribution towards the right.
- Most of the passengers on the spaceship have a range of age between 20 to 30.
- Numerous individuals on the spaceship are unlikely to spend money on those particular services.



	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
Age	1	0.07	0.13	0.03	0.12	0.1
RoomService	0.07	1	-0.02	0.06	0.01	-0.02
FoodCourt	0.13	-0.02	1	-0.01	0.22	0.23
ShoppingMall	0.03	0.06	-0.01	1	0.01	-0.01
Spa	0.12	0.01	0.22	0.01	1	0.15
VRDeck	0.1	-0.02	0.23	-0.01	0.15	1

Figure 4: Correlation matrix of numerical features of the training dataset

Finally, we create a correlation matrix to gain a better understanding of the data. This analysis allows us to verify whether there is high correlation between the numerical variables. This is generally a positive indication when searching for parsimonious models, as it eliminates causal relationships that would make the model redundant. Additionally, if we decide to take a statistical approach for the prediction, this information can be useful.

3 Data preprocessing

As we explore the datasets in the previous section, we can see that there exists multiple categorical features as well as various missing values. Therefore, it is essential to preprocess the datasets in order for learning algorithms to be able to process them.

3.1 Adding and removing features

Firstly, for the `Cabin` feature, we think that in order for the data to be meaningful, it is better that the feature is split into 3 different features: `CabinDeck`, `CabinNum` and `CabinSide`, by the slash (/) symbol. The original `Cabin` feature is not needed anymore and thus is removed.

Next, we create a new feature named `TotalExpense`, which is the result of the sum of all the amenity values, specifically `RoomService`, `FoodCourt`, `ShoppingMall`, `Spa` and `VRDeck`, for each of the passenger.

Finally, `PassengerId` and `Name` features are dropped. This is because in our opinion, both features would not be useful in predicting the target variable since they are just identifications for the samples.

3.2 Encoding data

Since some learning algorithms only take numerical data as input, it is a good idea to convert all categorical data into numerical data. In the datasets, the features that are to be encoded are `HomePlanet`, `CryoSleep`, `Destination`, `VIP`, `CabinDeck` and `CabinSide`.

In order to encode the data, we use ordinal encoding technique. This technique assigns each category to an integer value. More specifically, with Scikit-learn, `OrdinalEncoder`[1] assigns each category to an integer in $[\![0, n-1]\!]$ where n is the number of categories of a feature.

3.3 Imputing data

In this step, we fill the missing values in the datasets. To do so, we have 2 strategies:

1. Imputing missing values by hand based on the insights of data that we found out in the data analysis section. The imputation rules are:
   - The passengers are mostly from `Earth` so we impute all missing data of `HomePlanet` with `Earth`
   - The destination `TRAPPIST-1e` is more likely to be the choice of passengers so we impute all missing data of `Destination` with `TRAPPIST-1e`
   - Most passengers do not have VIP status so we impute all missing data of `VIP` with `False`
   - Passengers are more likely to not take the Cryosleep so we impute all missing data of `CryoSleep` with `False`
   - We use the median value to impute missing data of `Age` but we divide the passengers into two groups: VIP and non-VIP passengers
   - We impute all missing data of others numerical variables with the `KNNImputer`[2] based on KNN algorithm.
2. Imputing missing values using the mean value. With this strategy, for each column, we calculate the mean value of available data and impute the missing data with that value. To do so, we use `SimpleImputer`[3] from Scikit-learn and set the parameter `strategy='mean'`.

We tried both strategies and the second one with the `SimpleImputer` that uses the mean value of each column returns a higher accuracy. Thus, we decided to choose the second strategy. Furthermore, with the first strategy that imputs missing values by hand, it could lead to overfitting.

4 Model comparison

4.1 Testing techniques

For the development of this project, we test several models that are available and are currently most used in classification tasks. To compare the models we use two methods:

- The first method consists of dividing the initial training set into two subsets: a training set (the set that is truly used to fit the model) and a validating set. The first set is used for model training and the second set is for precision measurement. The training set had 75% of the initial data while the validating set had the remaining 25%.
- The second method, we use was cross-validation with 5 folds.

4.2 List of models

The models that we use in this project:

- k-nearest neighbors: `KNeighborsClassifier`
- Logistic regression: `LogisticRegression`

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html
[2] https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html
[3] https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

- Support vector machine: `SVC`
- Decision tree: `DecisionTreeClassifier`
- Random forest: `RandomForestClassifier`
- Boosting: `GradientBoostingClassifier`, `XGBClassifier`, `LGBMClassifier`, `CatBoostClassifier`

4.3 k-nearest neighbors

This supervised learning method consists of assigning to an element a classification of its k nearest neighbors. For this we first need to define a distance on $\mathbb{R}^d$ where d is the number of variables. The distance that is used for this is the Euclidean distance.

After that we define the number of neighbors that we want to use for the classification. Choosing a small k may make the model unable to capture general trends from the data while a very large one means that important details are not captured when predicting. We tested multiple values of k with cross-validation and the one that gave the best performance was $k = 100$.

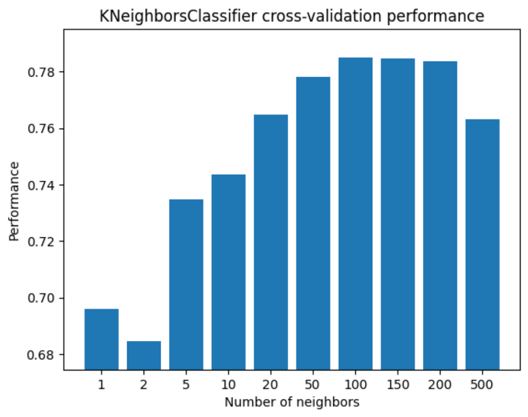


Figure 5: Cross-validation performance with respect to different number of neighbors

This model offers the advantage that it is easy to implement and understand conceptually. However, we consider that due to its lack of robustness it is not the most appropriate option for our case.

4.4 Logistic regression

In this project, we use the logistic regression with the l_2 penalty and thus, we have to consider the regularization parameter. For the parameter, we tested various values with cross-validation, and we managed to obtain the best performance when the parameter $C = 0.05$.

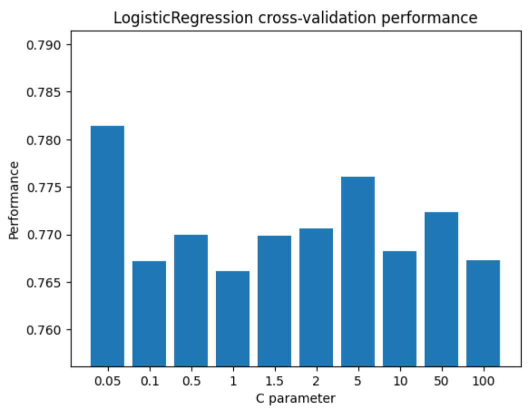


Figure 6: Cross-validation performance with respect to different values of regularization parameter

4.5 Support vector machine

With support vector machines, there are two important criteria that we need to consider: the kernel function and the regularization parameter. For this project, we use RBF kernel function, which is defined as:

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

For the γ parameter of the kernel function, we use the default value, which equals to 1 / (number of features $\times$ variance of feature data). As for the regularization parameter C, we tested several of the its values with cross-validation to compare the performances. Based on the test, we managed to get the best performance when $C = 1$.

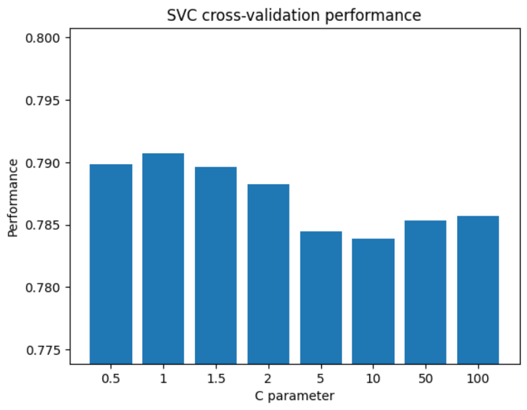


Figure 7: Cross-validation performance with respect to different values of regularization parameter

4.6 Decision tree

When working with the decision tree algorithm, we mainly consider on the impurity measure and the depth of the tree. The impurity measure that is used is entropy, which is given by:

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk}) \text{ with } p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} \mathbb{I}(y = k)$$

where Q_m represents data at node m with n_m samples. Splits in the tree are based on this impurity measure. In terms of the depth of the tree, nodes are expanded until all leaves are pure or until all leaves contain less than a number of samples, which is by default set to 2.

4.7 Random forest

One of the ensemble methods that is used in this project is random forest. With this learning technique, we focus on the number of trees and the number of samples to train each tree, on top of what we have already focused on with decision tree. For the number of trees, we tested various values with cross-validation, and the number of trees that gave the best performance was 100. Furthermore, for each tree, we randomly choose samples with a size that equals to the original sample size. Finally, the impurity measure and the depth of each tree remain the same as the decision tree algorithm.

4.8 Gradient boosting

Gradient boosting is another ensemble method that is used. For this algorithm, we mainly consider the learning rate along with the number of trees and the depth of the trees. We tested multiple values of the learning rate with cross-validation and the value that gave the best performance was 0.05. As for the number of trees, 100 remains the optimal number, which is the same as random forest. Furthermore, we use the default value of the maximum depth of each tree, which is 3.

Specifically, boosting models work from the training of what are called weak learners. These can be simple decision trees or in general models that, although they can be trained quickly, do not reach a very high level

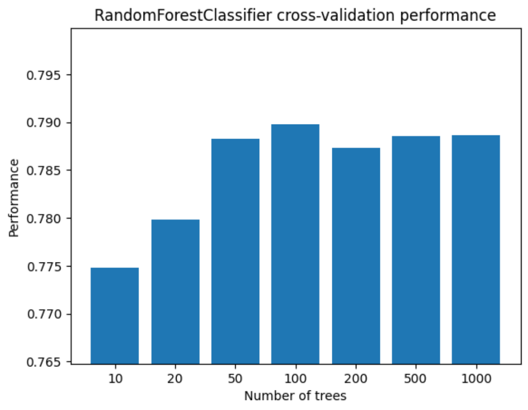


Figure 8: Cross-validation performance with respect to different number of trees

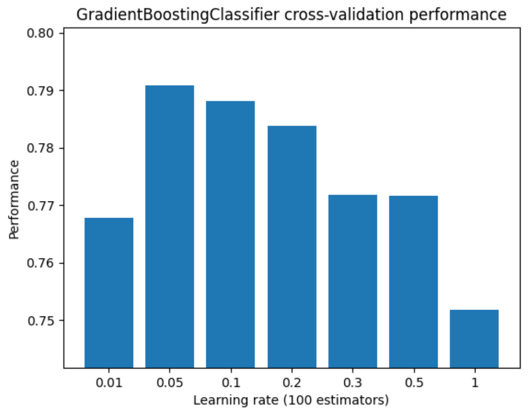


Figure 9: Cross-validation performance with respect to different learning rates (100 estimators/trees)

of precision. The first step is to define the loss function, for classification models the most used is the cross entropy loss function.

$$L(y, \hat{y}) = -(ylog(\hat{y}) + (1-y)log(1-\hat{y}))$$

where y is the true label, and $\hat{y}$ is the predicted probability of the positive class. The next step is to initialize the model as thee constant that minimize this loss function. After this the process consists of correcting the model iteratively through weak learners. That is, we solve the problem

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right)$$

where $F_{m-1}(x_i)$ is the model in the previous step and $h_m(x_i)$ is the weak learner trained over the residues of the model F_{m-1}. Finally we update the model as

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

4.9 XGBClassifier

Gradient Boosting with regularization, or regularized gradient boosting, is a variation on the traditional boosting model that introduces l_1 and l_2 penalties to the loss function. This regularization helps to prevent overfitting and improve generalization performance. Additionally, regularized gradient boosting can also use early stopping to avoid overfitting.

4.10 LGBMClassifier

LightGBM is a gradient boosting algorithm that uses gradient-based one-side sampling to reduce the number of samples used for each tree, histogram-based approach to discretize continuous features, leaf-wise tree growth, and GPU acceleration. These features allow for faster training and higher accuracy, particularly on large and complex datasets. However, LightGBM may require more hyperparameter tuning than traditional gradient boosting and may not always outperform traditional methods on small and simple datasets

4.11 CatBoostClassifier

CatBoost is a variant of gradient boosting, making it another ensemble method that is used in this project. CatBoost is a third-party library developed by Yandex that provides an efficient implementation of the gradient boosting algorithm on decision trees. During training, a set of decision trees is built consecutively. Each successive tree is built with reduced loss compared to the previous trees. It is designed to work perfectly on categorical input variables. We use the CatBoostClassifier() with the metric is "Logloss" and the number of iterations (number of trees built) is 1000 trees. The Logloss function is given below:

$$\text{Logloss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

The aims of the CatBoostClassifier is to minimize this Logloss function. The value of Logloss function is reduced by each iteration. We let the learning rate is default so the learning rate is defined automatically depending on the number of iterations and the input dataset. This rate can be tuned automatically to get the best possible quality. Based on the evaluation metric values (Logloss value) on each iteration to tune the learning rate:

- Decrease the learning rate if overfitting is observed.
- Increase the learning rate if there is no overfitting and the error on the evaluation dataset still reduces on the last iteration.

4.12 Summary of performance

The performance of the models are summarized in the following table.

	Training/validating set	**Cross-validation**
`KNeighborsClassifier`	0.7884	0.7850
`LogisticRegression`	0.7925	0.7814
`SVC`	0.7902	0.7908
`DecisionTreeClassifier`	0.7466	0.7389
`RandomForestClassifier`	0.7971	0.7898
`GradientBoostingClassifier`	0.7948	0.7909
`XGBClassifier`	0.8017	0.7740
`LGBMClassifier`	0.8040	0.7848
`CatBoostClassifier`	0.8146	0.7925

Table 2: Performance of different tested models

Regarding the result of the testing, we decided to choose the `CatBoostClassifier` model because it has the best accuracy. In order to increase the accuracy and reduce the dimension of the model, we decided to do the feature selection in order to select the important features (variables) to put into the `CatBoostClassifier` model.

4.13 Sequential feature selection

We use the Sequential Feature Selection method of the Scikit-learn library to select features in the CatBoost model. The idea of this method is to add (forward selection) or remove (backward selection) one feature at each stage to find the best feature subset. At each stage, it chooses the best feature to add or to remove based on the cross-validation score (we chose the accuracy in this case). For this project, we use the code below:

```python
from catboost import CatBoostClassifier
from sklearn.feature_selection import SequentialFeatureSelector

# Feature selection
model_fs = CatBoostClassifier(verbose=False)
```

```
sf = SequentialFeatureSelector(model_fs, scoring='accuracy', direction='backward',
    n_features_to_select='auto', tol=None)
sf.fit(X_train_data, Y_train_data)
best_features = list(sf.get_feature_names_out())
```

We choose the backward direction and the score for selection is accuracy. Therefore, starting from the full `CatBoostClassifier` model with all features (variables), at each stage, it removes each feature sequentially and compares the accuracy of those new models to choose the model with the highest accuracy corresponding to the best feature to be removed. The `n_features_to_select` is set to `auto` and `tol` is set to `None`, which means at least half of the features are selected. This is to avoid removing too many variables from the model. After using this method, we obtain the best feature subset for training the `CatBoostClassifier` model. It includes: `CryoSleep`, `RoomService`, `Spa`, `VRDeck`, `CabinDeck`, `CabinSide`, and `TotalExpense`.

5 Performance on Kaggle

At the time of writing this report, the best score that we got was 0.81271, which leaves us in position 60 out of 2 365 participants. This corresponds to the top 3%.

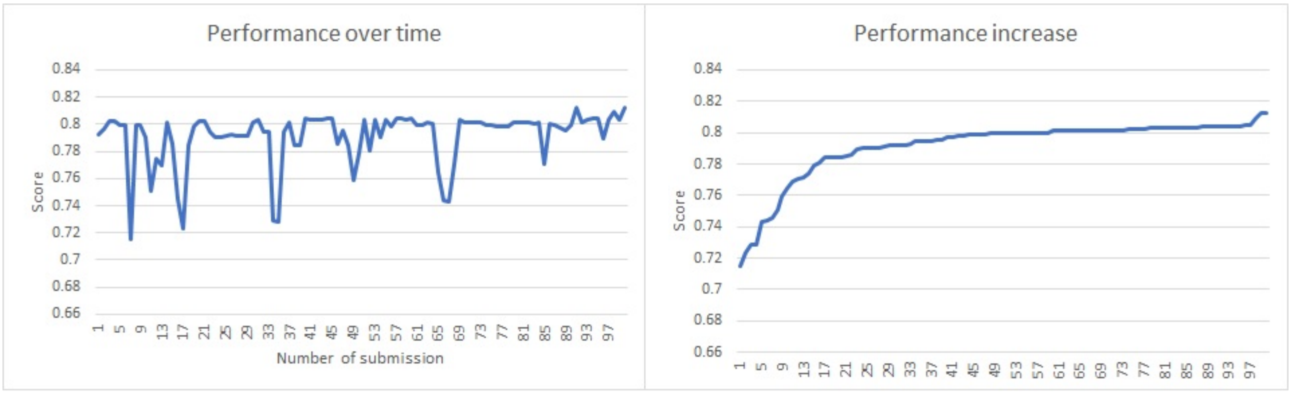


Figure 10: Performance on Kaggle

Figure 10 depicts our progress on Kaggle over 100 attempts. As shown in the left figure, our progress was far from linear. The lower peaks in performance were due to suboptimal model choices and, in most cases, overly complex models that led to overfitting.

In the second graph, we can see a gradual increase in our score until we reached 0.79, after which the increase was marginal until the final jump from 0.8 to 0.81. This last improvement was due to two major variables: the use of CatBoost and a decision to opt for simpler imputation methods. In our opinion, overcomplicating the imputation by hand methods was leading to overfitting, which was resolved by using a simple imputer.

6 Conclusion and further improvements

In conclusion, our models and data treatment methodology proved to be effective tools for predicting the objective variable. Firstly, we emphasize the importance of using an adequate selection of variables, which enabled us to build a parsimonious model that avoided overfitting.

Moreover, we found that using simple imputation methods yielded the best results, despite our attempts to use more specific methods that resulted in lower model scores. This highlights the potential drawbacks of overly complex models, which may struggle to detect the most general behaviors of the data.

It is worth noting that three contestants scored above 0.82, which suggests that there may be alternative approaches to this problem that differ from our project's methodology. Thus, we believe that expanding the range of tested models should be the main strategy for improving our score. Hyperparameter tuning method such as Random Search, Grid Search for optimizing the parameters of training model is also a good choice to research and try.

A Appendix: code

Import libraries

```python
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
```

Training data preparation

```python
# Read data from file
train_data = pd.read_csv('train.csv')

# Check missing values
print(train_data.isnull().sum())

# Preprocess data
def data_pipeline(df):

    # Split column "Cabin" into 3 columns: "CabinDeck", "CabinNum", "CabinSide"
    df[['CabinDeck', 'CabinNum', 'CabinSide']] = df.Cabin.str.split('/', expand = True)

    # Drop 3 columns "PassengerId", "Name" and "Cabin"
    df = df.drop(['PassengerId', 'Name', 'Cabin'], axis='columns')

    # Create a new feature "TotalExpense"
    amenities = ['RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck']
    df['TotalExpense'] = df[amenities].sum(axis=1)

    # Encode categorical data
    cat_cols = ['HomePlanet', 'CryoSleep', 'Destination', 'VIP', 'CabinDeck', 'CabinSide']
    encoder = OrdinalEncoder().fit(df[cat_cols])
    df[cat_cols] = encoder.transform(df[cat_cols])

    if 'Transported' in df.columns:
        df['Transported'] = list(map(int, df['Transported']))

    # Impute missing values
    miss_cols = df.isnull().sum()
    miss_cols = list(miss_cols[miss_cols>0].index)
    tf = ColumnTransformer([("imp", SimpleImputer(strategy='mean'), miss_cols)])
    df[miss_cols] = tf.fit_transform(df[miss_cols])

    return df

train_data = data_pipeline(train_data)

# Separate data into features and labels
X_train_data = train_data.drop('Transported', axis='columns')
Y_train_data = train_data['Transported']
```

Testing data preparation

```python
# Read data from file
X_test_data = pd.read_csv('test.csv')

```

```python
# PassengerId to write to submission file
PassengerIdTest = X_test_data['PassengerId']

# Check missing values
print(test_data.isnull().sum())

# Preprocess data
X_test_data = data_pipeline(X_test_data)
```

Comparing different models

KNeighborsClassifier

```python
n_neighbors_list = [1, 2, 5, 10, 20, 50, 100, 150, 200, 500]
score_list = []
for n_neighbors_ in n_neighbors_list:
    clf = KNeighborsClassifier(n_neighbors=n_neighbors_)
    scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
    score_list.append(np.mean(scores))
print(score_list)
```

LogisticRegression

```python
C_list = [0.05, 0.1, 0.5, 1, 1.5, 2, 5, 10, 50, 100]
score_list = []
for C_ in C_list:
    clf = LogisticRegression(penalty='l2', C=C_)
    scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
    score_list.append(np.mean(scores))
print(score_list)
```

SVC

```python
C_list = [0.5, 1, 1.5, 2, 5, 10, 50, 100]
score_list = []
for C_ in C_list:
    clf = SVC(C=C_, kernel='rbf', gamma='scale')
    scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
    score_list.append(np.mean(scores))
print(score_list)
```

DecisionTreeClassifier

```python
clf = DecisionTreeClassifier(criterion='entropy', splitter='best', max_depth=None,
    min_samples_split=2)
scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
print(np.mean(scores))
```

RandomForestClassifier

```python
n_estimators_list = [10, 20, 50, 100, 200, 500, 1000]
score_list = []
for n_estimators_ in n_estimators_list:
    clf = RandomForestClassifier(n_estimators=n_estimators_, criterion='entropy',
        max_depth=None, min_samples_split=2, bootstrap=True, max_samples=None)
    scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
    score_list.append(np.mean(scores))
print(score_list)
```

GradientBoostingClassifier

```python
learning_rate_list = [0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 1]
n_estimators_list = [10, 20, 50, 100, 200, 500, 1000]
score_list = []
for learning_rate_ in learning_rate_list:
    score_list.append([])
    for n_estimators_ in n_estimators_list:
        clf = GradientBoostingClassifier(n_estimators=n_estimators_,
            learning_rate=learning_rate_, max_depth=3)
        scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
        score_list[-1].append(np.mean(scores))
print(score_list)
```

XGBClassifier

```python
clf = XGBClassifier()
scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
print(np.mean(scores))
```

LGBMClassifier

```python
clf = LGBMClassifier()
scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
print(np.mean(scores))
```

CatBoostClassifier

```python
clf = CatBoostClassifier(verbose=False)
scores = cross_val_score(clf, X_train_data, Y_train_data, cv=5)
print(np.mean(scores))
```

Training and predicting with the best model (CatBoostClassifier)

```python
# Feature selection
model_fs = CatBoostClassifier(verbose=False)
sf = SequentialFeatureSelector(model_fs, scoring='accuracy', direction='backward',
    n_features_to_select='auto', tol=None)
sf.fit(X_train_data, Y_train_data)
best_features = list(sf.get_feature_names_out())

# Model training
clf = CatBoostClassifier(verbose=False)
clf.fit(X_train_data[best_features], Y_train_data)

# List all parameters of the trained model
print(clf.get_all_params())

# Prediction
prediction = clf.predict(X_test_data[best_features])

# Write prediction to submission file
res = pd.DataFrame(
        {
            'PassengerId': list(PassengerIdTest),
            'Transported': [(p == 1) for p in list(prediction)]
        }
    )
res.to_csv('submission.csv', index=False)
```