Machine Learning Project

Kaggle Competition - Spaceship Titanic

4th April 2024

1 Team Composition

For this competition, our team name is "**Fiv3**" consisting of 4 members, namely:

- CHEAM Richard, LY Sovanleng, NOUV Ratanakmuny, THUO Menghor

2 Introduction

In this competition, our task is to predict whether a passenger was transported to an alternate dimension by utilizing a given dataset, which it provides personal records that recovered from the ship's damaged computer system. Speaking of which, the dataset was split into two, namely, `train` and `test`, comprising two-thirds and one-third, respectively, of a total of approximately `13000` personal records.

3 Data Analysis

This phase is a fundamental and indispensable step within this project which involves the process of inspecting, cleansing, transforming data, and so on, with the goal of discovering useful information in order to subsequently support the decision-making.

3.1 Exploratory Data Analysis (EDA)

As for `Train` dataset, it consists of `8693` observations, whereas, `4277` observations lie within the `Test` dataset. Each dataset consists of `13` explanatory variables ($X_{i=1,2,...,13}$) and only the `Train` dataset contains a target variable (Y).

`13` predictors are described as follows:

- `PassengerId`: a unique `ID` for each passenger.
- `HomePlanet`: planet which the passenger departed from.
- `CryoSleep`: an indication whether the passenger elected to be put into suspended animation.
- `Cabin`: cabin number where the passenger is staying in the form of `deck/num/side`.
- `Destination`: planet which the passenger will be debarking to.
- `Age`: age of the passenger.
- `VIP`: an indication whether the passenger has paid for special `VIP` service.
- `RoomService, FoodCourt, ShoppingMall, Spa VRDeck`: amount which the passenger has billed for each ship's amenity.
- `Name`: first and last names of the passenger.

`1` target variable is described as follows:

- `Transported`: an indication of whether the passenger was transported to another dimension.

In addition, the given features correspond to the following types:

- `Object`: PassengerId, HomePlanet, CryoSleep, Cabin, Destination, VIP, Name.
- `Float`: Age, RoomService, FoodCourt, ShoppingMall, Spa, VRDeck.
- `Boolean`: Transported.

Hence, the categorical data will be transformed into numerical so that the machine learning algorithms can be fitted as the `scikit-learn` library are designed to work with numerical data. To deal with those qualitative predictors, a method called "Data Encoding" will be used.

3.1.1 Inspection of Missing Values

It is essential to check for null or missing values within the dataset in order to handle them later on so that we can ensure data integrity and model performance. Figures below are the visualization of missing values contain in the `Test` and `Train` dataset in which `PassengerId` has no missing values for both dataset.

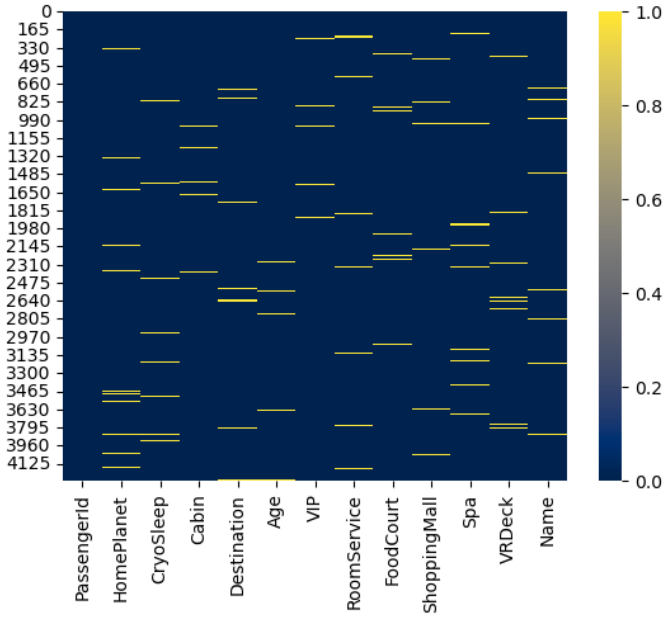

Figure 1: Heatmap of test dataset

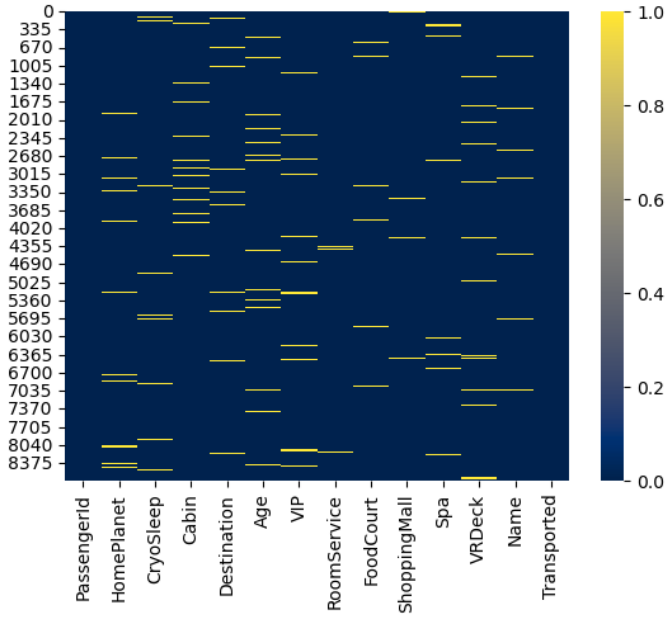

Figure 2: Heatmap of train dataset

Visibly, there are 2324 missing values in total for `Train` dataset, and 1117 for `Test` dataset, for which they will be handled later in Data Preprocessing section.

3.1.2 Identifying Patterns and Trends

Regarding this section, the given dataset will be further explored to discover patterns and trends lie within the dataset with the aid of data visualization and summary statistics.

In the previous section, different types of data were identified; here, each categorical feature will be scrutinized. However, `PassengerId` and `Name` will not be taken into consideration since it does not have a strong meaning towards the analysis. Below is a list of each feature with its categorical values.

- `HomePlanet : 'Europa', 'Earth', 'Mars'.`
- `Destination : 'TRAPPIST-1e', 'PSO J318.5-22', '55 Cancri e'.`
- `Cabin: 'B/0/P', 'F/0/S', 'A/0/S', ..., 'G/1499/S', 'G/1500/S', 'E/608/S'.`

Missing Values		
Predictor	`Train`	`Test`
`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
Total	2324	1117

Table 1: Number of missing values in train and test dataset

- `CryoSleep, VIP, Transported : 'True', 'False'`.

Notably, for `Cabin` feature, the only part of the cabin form that is helpful for the analysis is its `Deck`, since cabin's side can either be *Port* or *Starboard* which are equivalent in terms of occurrence.

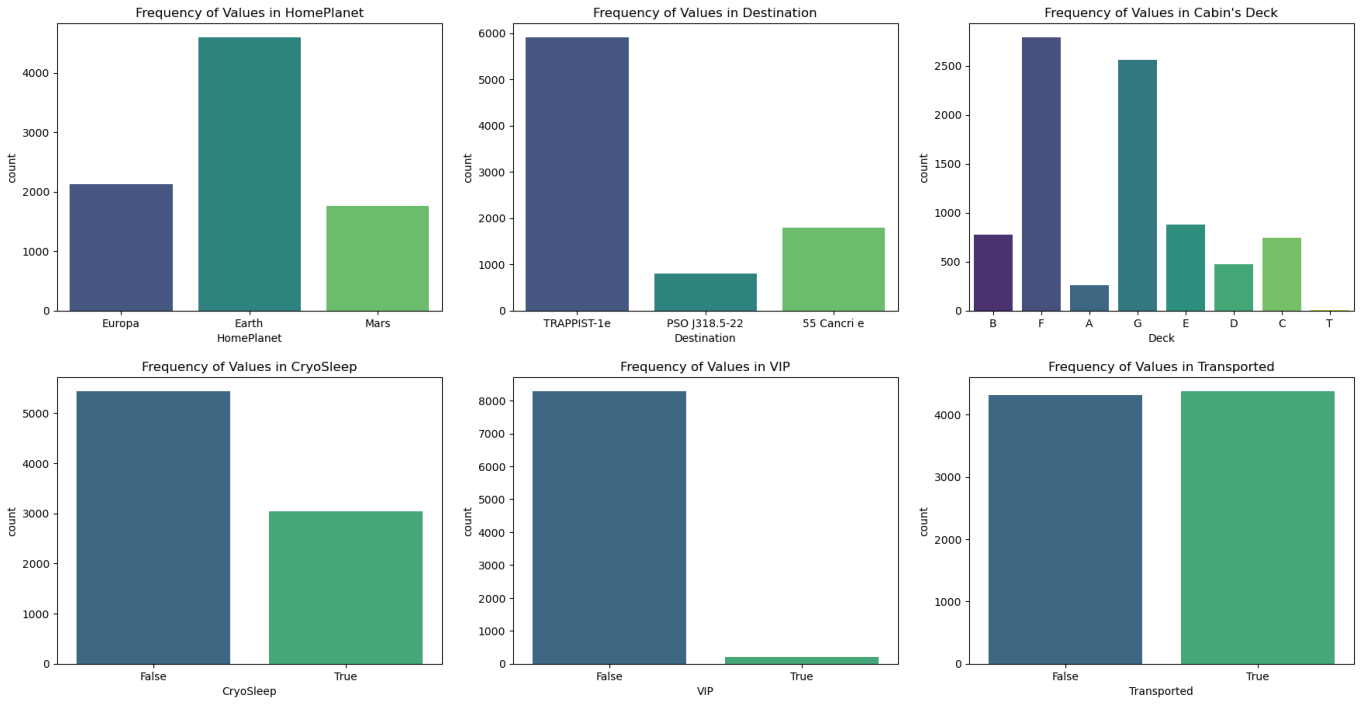

Figure 3: Frequency of categorical features in training dataset

According to Figure 3, passengers are most likely chose to depart from `Earth` to `TRAPPIST-1e` without a `VIP` status. Moreover, the majority of passengers are presumably staying at deck `F` and `G` since these two decks outnumbered the others and very little at deck `T`. Last but not least, twice as passengers preferred to be not in `CryoSleep` as to those who preferred to be in suspended animation and the number of passengers who are successfully `Transported` is relatively equal.

	HomePlanet	CryoSleep	Destination	VIP	Deck	Transported
count	8492	8476	8511	8490	8494	8693
unique	3	2	3	2	8	2
top	Earth	False	TRAPPIST-1e	False	F	True
freq	4602	5439	5915	8291	2794	4378

Table 2: Descriptive statistics for categorical features in training dataset

Based on Table 2, *count* indicates that there are missing values since each categorical feature does not reach the total observation of `8693`. Also, *unique* tells us the possible values each feature could take, while *top* is the most common value with its number of occurrences below labeled by *freq*. For instance, out of `8693` passengers, `5915` people are destined to `TRAPPIST-1e`, while `2596` headed to different destinations, and the rest (`182` observations) are missing data.

On top of that, numerical features will be examined as well in order to get additional information from the dataset by making use of histograms rather than bar charts.

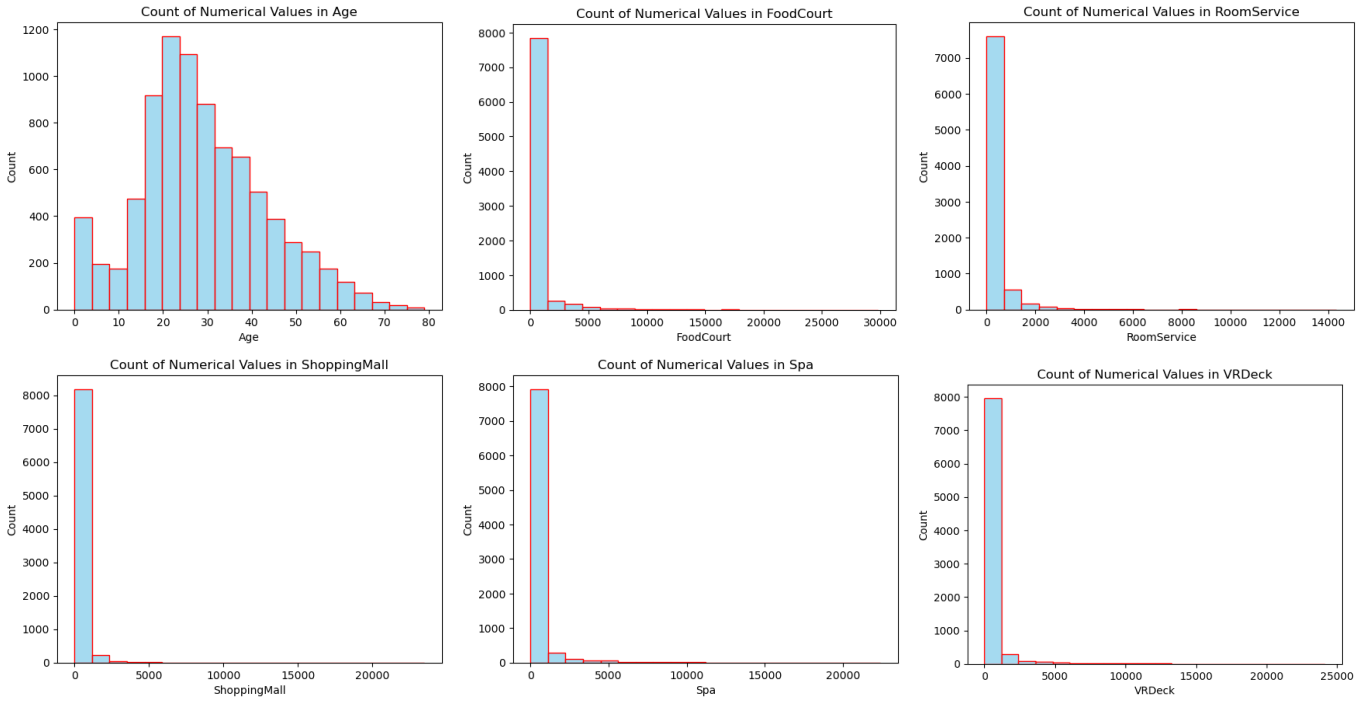

Figure 4: Distribution of each numerical feature in training dataset

In accordance to Figure 4, evidently, passengers on the spaceship rather not paying for provided luxury amenities since the expenditure of services are positively skewed (tail is more pronounced on the right side). Plus, most of individuals' age are range from `15-30` as the distribution fluctuates at the start then reaches its peak at `20s` and falls off steeply ever since. Plus,

	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
count	8514.000	8512.000	8510.000	8485.000	8510.000	8505.000
mean	28.827	224.688	458.077	173.729	311.139	304.855
std	14.489	666.718	1611.489	604.696	1136.706	1145.717
min	0.000	0.000	0.000	0.000	0.000	0.000
25%	19.000	0.000	0.000	0.000	0.000	0.000
50%	27.000	0.000	0.000	0.000	0.000	0.000
75%	38.000	47.000	76.000	27.000	59.000	46.000
max	79.000	14327.000	29813.000	23492.000	22408.000	24133.000

Table 3: Descriptive statistics for numerical features in training dataset

As demonstrated by Table 3, the age of passengers range up to `79`, with an average of approximately `29`, and *25%* (Q1) of the passengers are 19 years old or younger. Since standard deviation (*std*) quantifies the amount of variation or dispersion in a set of values, service features such as `FoodCourt, Spa, VRDeck` and so on have a high variability compared to the average which is reasonable as only few people were willing to pay for such amenities by virtue of Figure 4; nevertheless, *50%* (Q2) being `0` for all services implies that half of the observations did not willing to spend and *75%* (Q3) are significantly lower than their *max* values in which such discrepancy suggests outliers in the dataset. With that being said, passengers are either have very minimal spending or no spending at all, while only few individuals have a large amount of expense on spaceship's services.

3.1.3 Correlation Analysis

For this section, the primarily concern is to find out whether relationships exist between variables. Due to the fact that the dataset contains both categorical and numerical variables, hence, two cases of analysis will be considered: quantitative variables analysis and qualitative variables analysis.

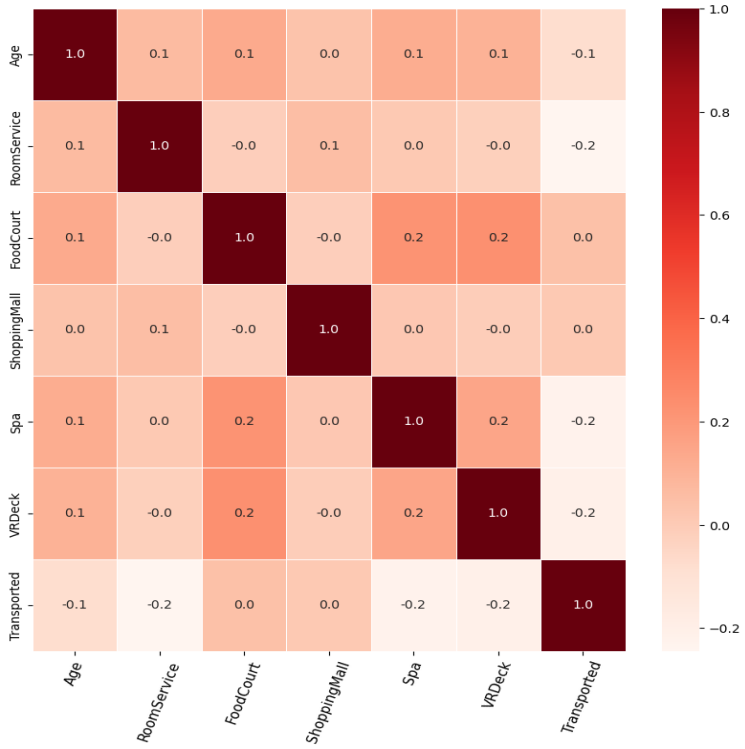

Figure 5: Correlation matrix of quantitative predictors and target variable in training dataset

Figure 5 suggests that there is no correlation exists between input variables (X) as well as with the target

variable (Y) which is a good news as the classification later will not be affected by relationship within variables.

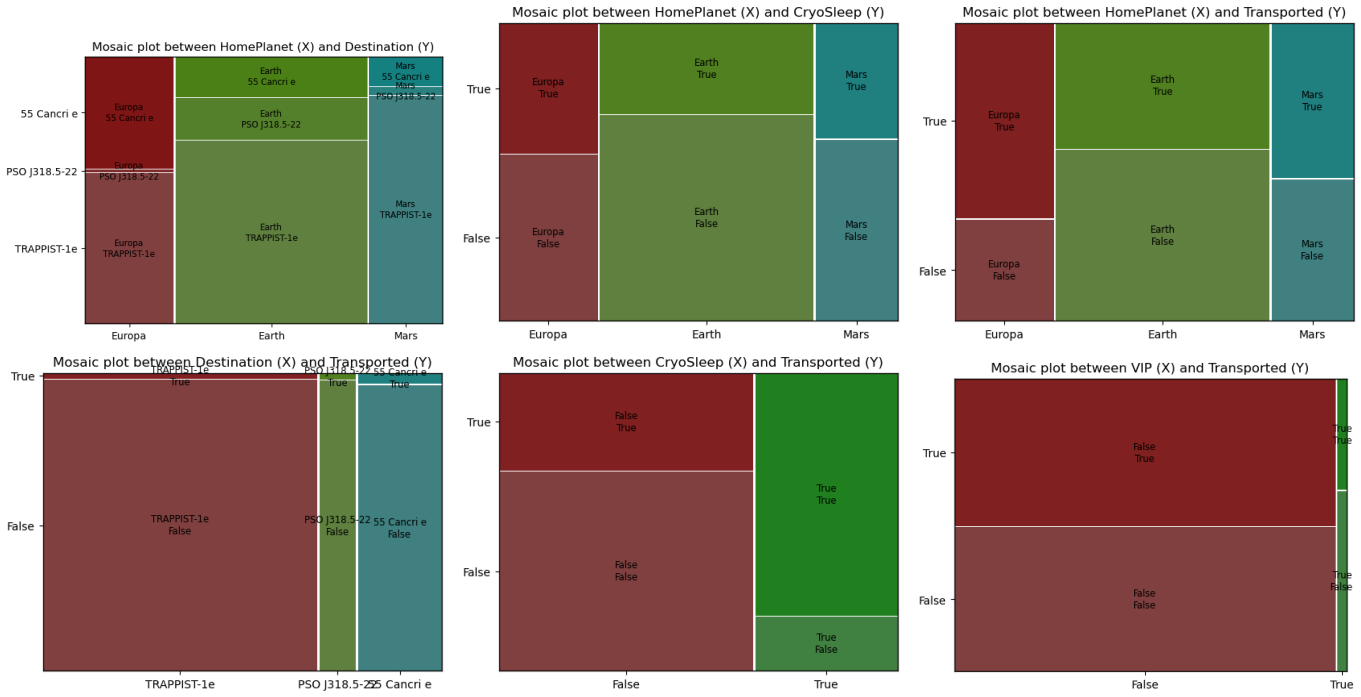

Figure 6: Mosaic plots between certain qualitative predictors and target variable in training dataset

Figure 6 indicates several inferences such as:

- Most of the passengers' destination is `TRAPPIST-1e` regardless of their `HomePlanet`.
- Majority of the passengers preferred to not be in `CryoSleep`.
- Passengers from `Europa` are likely to not be `Transported`, while `Earth` seems to have a successful rate of 40% and `Mars` is balance.
- Being successfully `Transported` does not take into account the `Destination` which demonstrates that they are not dependent.
- Individuals who were in `CryoSleep` are likely to be successfully `Transported` which suggests the dependency.
- Whether or not passengers hold `VIP` status, it does not affect the state of being successfully `Transported`, which indicates that they are independent to one another.

4 Data Preprocessing

Thus far, the Data Preparation and Exploration steps gave us a better overview on the nature of the given dataset, hence, in this section, a crucial step will be taken in order to make the data suitable for modeling later on such as Feature Engineering, Data Encoding, Handling Missing Values, and so on.

4.1 Feature Engineering

This process is also known as **Data Transformation** in which it is a machine learning technique that extract relevant information from a single predictor in order to create new predictors that are not in the initial training set.

For this project, variable `Cabin` takes form `deck/num/side`, as a result, this predictor will be split into

3 different predictors: `Deck, Num, Side`, which `Deck` and `Side` are categories and `Num` is numeric. Also, since there is a noticeable different in terms of expenditure, a column called `TotalExpense` will be created by summing all expense columns.

Therefore, such transformation will then increase the number of predictors from `13-1` (deletion of `Cabin`) to `(13-1)+4` (addition of `Deck, Num, Side` and `TotalExpense`). In this case, the dataset now possesses in total of `16` predictors.

4.2 Data Encoding

Since the algorithms to be used require numerical data as input, so it is a must to transform categorical values to numerical. In Data Analysis section, certain categorical data were identified, however, variables `Name` and `PassengerId` will not be taken into account for further procedures since they have no signification on the prediction afterward. Thus, the remaining features will be encoded by utilizing different encoding techniques, which leads to the dataset with `14` quantitative predictors.

4.2.1 Label Encoding

Label encoding is a simple and effective way to convert categorical variables into numerical form in which they have no inherent order. For this project, `LabelEncoder`[1] from `scikit-learn` will be used, which this transformer should be used to encode target value Y, rather than input X.

Therefore, the target variable `Transported` will then have an `Int` type rather than `Bool`, which `1` stands for `True`, and `0` stands for `False`.

4.2.2 Ordinal Encoding

In ordinal encoding or integer encoding, each unique category value is assigned as an integer value; this results in a single column of integers per feature in which they take value in a finite set, meaning, $\forall X_{i=1,2,...,13} \in [0, n-1]$, where n is number of categories of a feature. In the same way, `OrdinalEncoder`[2] of `scikit-learn` will be responsible for such conversion.

Consequently, as an instance, explanatory variable `HomePlanet` will then uniquely take values in a set `{0, 1, 2}`, which each one of the integer represents its own categorical information: `{Europa, Earth, Mars}` respectively, and likewise for other qualitative predictors.

4.3 Handling Missing Values

As stated above, to ensure the quality of the data and the fitted models , we will have to inevitably take into account the existence of null values within the dataset. To do so, we use the imputation method which refers to the replacement of missing values with substitute values to retain most of the information of the dataset. The imputation rules are:

- Impute missing value of numerical variables with `KNNImputer`[3] based on KNN algorithm
- Impute missing value of categorical variables by the most frequent class. To do so, we use `SimpleImputer`[4] from `scikit-learn` and set the parameter `strategy='most_frequent'`

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

5 Model Selection

5.1 Testing Techniques

In order to evaluate our models, we test several models that we have learnt and some that are most used for classification problem. Plus, we use `GridSearchCV` to get the best parameters for each selected model below. To compare the models we use two methods:

- Train-Test Split: a traditional method that divide training dataset into two to estimate how the model will generalize to new, unseen data.
- Cross-Validation: we perform 10 folds in order to evaluate the robustness of the model across multiple folds and to avoid overfitting.

5.2 LogisticRegression

Logistic regression is a statistical method that is used to model a binary response variable based on predictor variables. We apply the logistic regression with the l_2 regularization to avoid overfitting by penalizing large coefficient, promoting model simplicity and robustness. For the parameter, $max\_iter = 10000$ for the optimization algorithm to converge, accommodating the regularization and complexity of the data. In addition, we tested various value with split dataset for scaled and unscaled and cross-validation. We managed to obtain the best performance by cross-validation when the parameter $C = 0.005$.

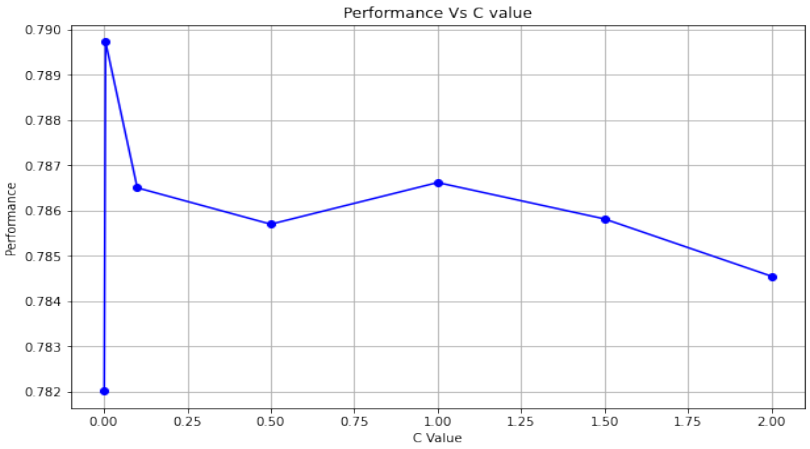

Figure 7: Cross-validation performance with respect to different values of regularization parameter

5.3 Support Vector Machine

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

$$K(x, x') = exp(-\lambda \|x - x'\|)$$

For the λ parameter of the kernel function, we use the default value, which euqals 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 $C = 9$.

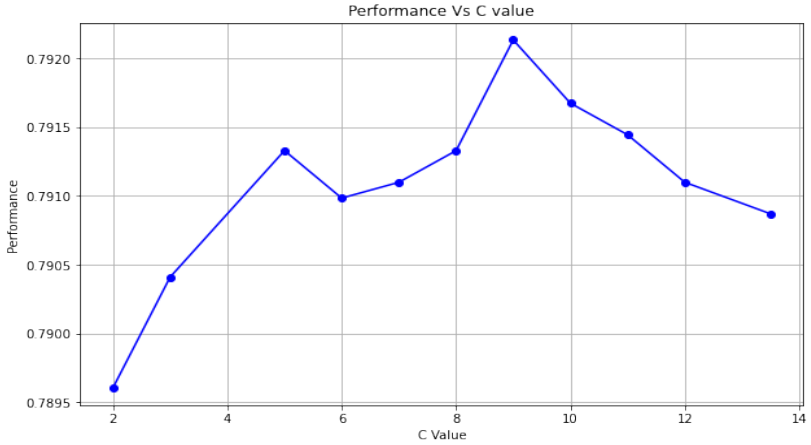

Figure 8: Cross-validation performance with respect to different values of regularization parameter

5.4 DecisionTreeClassifier

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

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

where Q_m represents data at node m with n_m samples. Splits in the tree are based on this impurity measure. In term 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 set to 6.

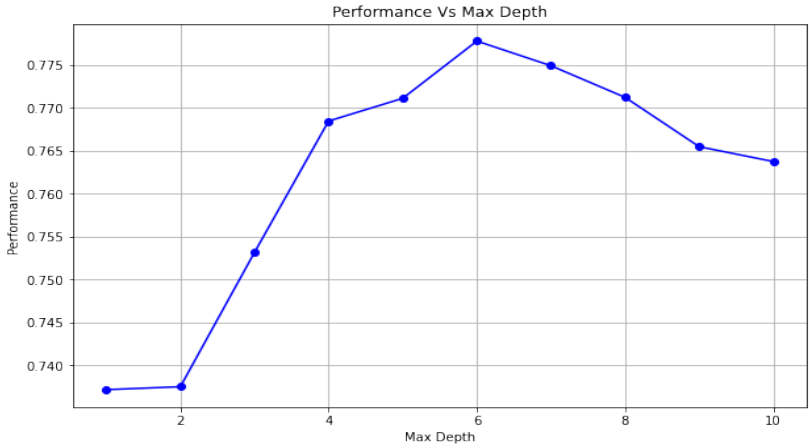

Figure 9: Cross-validation performance with respect to different values of regularization parameter

5.5 Random Forest

The ensemble method we chose to implement in this project is the Random Forest algorithm. This technique constructs a multitude of decision trees during training and outputs the majority vote or average prediction of the individual trees for classification and regression tasks, respectively. In our exploration, we prioritized the optimization of the number of trees and the size of the sample data used to train each tree. After a series of cross-validation experiments, we found that setting the number of trees to 200 optimized our model's performance. To ensure a diverse set of trees, we used bootstrapping, randomly selecting samples from the dataset equal in size to the original dataset for each tree. Furthermore, the measures for purity and depth of the trees were maintained as per the standard decision tree algorithm, which base their splits on the maximization of information gain or minimization of entropy.

5.6 Gradient Boosting classifier

Gradient Boosting is another ensemble learning technique utilized in our study. It incrementally builds an ensemble of weak learners, typically decision trees, and improves the model through the optimization of a loss function. We tested various learning rates and found that a learning rate of 0.05, with 400 estimators, yielded the most robust model performance. The trees' depth was kept at a default value of 3 to prevent overfitting and maintain computational efficiency.
Boosting models commence with the initial learners making predictions and iteratively correcting them through subsequent models. Specifically, we used the formula:

$$L(y, f) = -y \log(g) - (1 - y) \log(1 - g)$$

where y is the actual value, g is the predicted probability, and f is the prediction model. We iteratively updated our model by training weak learners on the residual errors of the previous learners, as per the equation:

$$f_m(x) = f_{m-1}(x) + \gamma_m h_m(x)$$

with $f_m(x)$ representing the updated model's prediction, γ_m the learning rate, and $h_m(x)$ the weak learner focused on the residual errors. The process of finding the optimal learning rate γ_m can be automated using libraries like Scikit-learn's 'GridSearchCV'.

5.7 XGBoostClassifier

XGBoost is an optimized distributed gradient boosting framework designed to be highly efficient, flexible, and portable, making it particularly suitable for handling large datasets. It incorporates Lasso (l_1) and Ridge (l_2) regularization techniques to prevent overfitting and enhance generalization, thereby improving the model's robustness and performance.

5.8 LGBMClassifier

LightGBM is a gradient boosting framework that uses tree based learning algorithm, and designed to optimized speed and memory usage with better accuracy. It utilizes histogram-based algorithms for efficient tree construction, complemented by optimization techniques such as leaf-wise tree growth and efficient data storage strategies.

5.9 CatBoostClassifier

CatBoost has two main features, it works with categorical data (the Cat) and it uses gradient boosting (the Boost), making it another ensemble method that is used in this project. It 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. Since the target variable has only two different values, "Logloss" metric will be used which is given by:

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

Where `N` is the number of samples, y_i represents the true label of the i-th instance and $\hat{y}_i$ represents the predicted probability that the i-th instance belongs to the positive class. The aim of this classifier is to minimize this Logloss function, so the value of Logloss function is reduced at each iteration.

5.10 AdaBoostClassifier

AdaBoost, short for Adaptive Boosting, a supervised learning algorithm that is utilised as an Ensemble Method by combining multiple weak or base learners (e.g., decision trees) into a strong learner. More precisely, this approach constructs a model and assigns equal weights to all data points. It then applies larger weights to incorrectly categorised points. It will continue to train models until a smaller error is returned.

6 Model Assessment

Performance of each model is summarized in the following table:

Model	Train-Test Split	Cross-Validation
`LogisticRegression`	0.777458	0.789721
`SVC`	0.775733	0.792135
`DecisionTreeClassifier`	0.771708	0.781898
`RandomForestClassifier`	0.787809	0.796853
`GradientBoostingClassifier`	0.791834	0.80019
`XGBClassifier`	0.80046	0.783510
`LGBMClassifier`	0.796435	0.797199
`AdaBoostClassifier`	0.783784	0.788227
`CatBoostClassifier`	0.798959	0.801686

Table 4: Performance of different models

Based on the evaluation above, `CatBoostClassifier` outperformed other models. In order to increase the accuracy and reduce the complexity of the model, important feature selection will be further conisdered.

7 Feature Selection

For this step, Sequential Feature Selection[5] (SFS) of Scikit-learn library will be utilized. This selector adds (forward selection) or removes (backward selection) features to form a feature subset in a greedy fashion. At each stage, it chooses the best feature to add or remove based on the cross-validation score of an estimator (model).

For our case, 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 in order to prevent from removing too many variables. Moreover, the `direction` used is "forward" with "5" `cv` which based solely on "Accuracy". As a result, features obtained are: `CryoSleep, RoomService, Spa, VRDeck, Deck, Side`, and `TotalExpense`. As a result, we obtained a better accuracy than before which is `81.225` (score on Kaggle).

8 Further Experiment

Ensemble models performed fairly well in predicting Y; however, the performance did not vary significantly due to identical feature engineering. Therefore, we believe that different feature engineering methods and ensemble models could yield an improvement. To do so, we can use the results from different feature engineering and use OR operator to identify data points that are difficult to predict. In this case we select the available submissions on Kaggle and combine with ours, namely, `sub`.

Score: 0.81903

[5] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html

```
sub1=pd.read_csv("sub1.csv")
sub2=pd.read_csv("sub2.csv")
sub3=pd.read_csv("sub3.csv")
sub_mix['Transported']= sub["Transported"] | sub1['Transported'] | sub2['Transported'] |
                                             sub3['Transported']
```

Then we repeat the process and play around with the combination which actually give us a better score: `0.81926`

9 Performance on Kaggle

Our best submission score is `0.81926`, which put us in the 45th position out of 2,605 teams of this competition. Plus, this position corresponds to the top 2%.

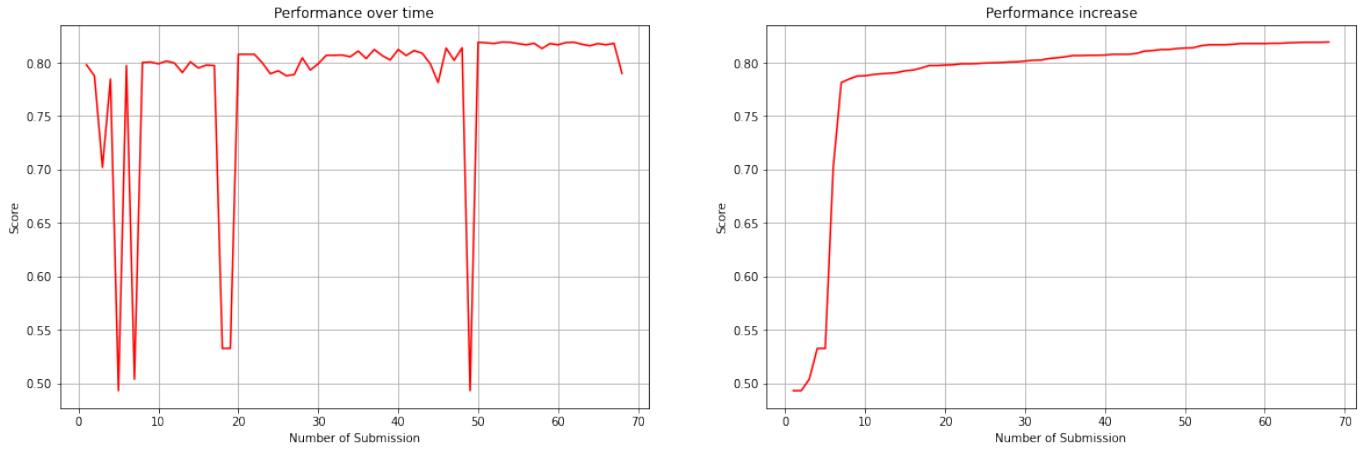

Figure 10: Performance on kaggle

The plots above demonstrate our performance over 69 submissions. The one on the left suggests a fluctuation which due to the fact that we tried many different appraoches from feature engineering to model. Besides, for the one on the right, apart from the mistakes we made, most of our submission scores were around 75-80.

10 Conclusion

In conclusion, data preparation and methodologies used proved to be effective for predicting the target variable. Starting from imputation techniques to selecting the revelant features which enabled us to build a parsimonious model which consists powerful predictors. Moreover, we found that using different features engineering combination approach actually did yield a good result as well despite the numerous of attempts since we have to play around with the ensemble.

Nevertheless, there are teams out there who scored above 0.82, which indicates that there may be alternative methods regarding this issue that differ from our project's methodology. Thus, we believe that expanding the range of tested models should be the main strategy to improve our score. Also, putting weights on the combination of models in section 8 would be a great idea as well.

A Appendix: code

Import Libraries

```
import pandas as pd
from sklearn.preprocessing import OrdinalEncoder, LabelEncoder
from sklearn.impute import SimpleImputer, KNNImputer
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier,
                                                RandomForestClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

Function to scale numerical features

```
def scale(X):
    num_cols = ["Age", "RoomService", "FoodCourt", "ShoppingMall", "Spa", "VRDeck", "
                                                    TotalExpense"]
    scaler = preprocessing.StandardScaler().fit(X[num_cols])
    X_scaled = pd.DataFrame(scaler.transform(X[num_cols]))
    X_scaled.columns = num_cols
    X = X.drop(num_cols, axis='columns')
    X = pd.concat([X, X_scaled], axis='columns')
    return X
```

Preparation of Training and Testing dataset

```
def data_prep(df):
    #split and delete Cabin
    df[['Deck', 'Num', 'Side']] = df['Cabin'].str.split('/', expand = True)

    #drop unwanted X
    df = df.drop(["PassengerId", "Cabin", "Name"], axis = 'columns')

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

    #Num is also Categorical as we split from Cabin
    categ_cols = ["HomePlanet", "Destination", "CryoSleep", "VIP", "Deck", "Side", "Num"]
    num_cols = ["Age", "RoomService", "FoodCourt", "ShoppingMall", "Spa", "VRDeck", "
                                                    TotalExpense"]

    #encoding
    encoder = OrdinalEncoder().fit(df[categ_cols])
    df[categ_cols] = encoder.transform(df[categ_cols])

    #imputation
    knn_imputer = KNNImputer(n_neighbors=2, weights="uniform")
    df[num_cols] = knn_imputer.fit_transform(df[num_cols])

    imp_freq = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
    imp_freq.fit(df[categ_cols])
    df[categ_cols]= imp_freq.transform(df[categ_cols])

    if "Transported" in df.columns:
        target = LabelEncoder().fit(df["Transported"])
        df["Transported"] = target.transform(df["Transported"])
    return df
```

```
# read csv file
df_train = pd.read_csv("train.csv")
df_test = pd.read_csv("test.csv")

# id index for submssion
PassengerId_test = df_test['PassengerId']

# data pipeline
df_train = data_prep(df_train)
df_test = data_prep(df_test)
```

Hyperparameter Tuning

```
def get_best_param(model, X_train, Y_train):
    # define hyperparameter grids for different models
    hyperparameter_grids = {
        'RandomForestClassifier()': {
            'n_estimators': [100, 200, 300],
            'max_depth': [None, 10, 20],
            'min_samples_split': [2, 5, 10],
            'min_samples_leaf': [1, 2, 4],
            'bootstrap': [True, False]
        },
        'GradientBoostingClassifier()': {
            'n_estimators' : [100, 200, 300, 400],
            'learning_rate': [0.1, 0.05, 0.01, 0.001],
            'max_depth': [4, 8],
            'min_samples_leaf': [100, 150],
            'max_features': [0.3, 0.2, 0.1]
        },
        'CatBoostClassifier()': {
            'iterations': [1000],
            'depth': [1,2,3,4,5,6,7,8,9,10],
            'learning_rate': [0.001, 0.005, 0.01, 0.05, 0.1],
            'l2_leaf_reg': [0,1,2,3,4,5,6,7,8]
        }
        #more models to be added
    }
    param_grid = hyperparameter_grids[model.__class__.__name__ + '()']

    if param_grid is None:
        raise ValueError("Model not supported or hyperparameter grid not defined")

    grid_search = GridSearchCV(model, param_grid=param_grid, scoring="accuracy", cv=5)
    grid_search.fit(X_train, Y_train)

    return grid_search.best_score_, grid_search.best_estimator_, grid_search.best_params_
```

Cross-Validation (Train-Test Split is implemented with the same logic)

```
def get_CV_score(X_train, Y_train):
    print('------- Instantiating Models -------')
    models = {
        'Logistic Regression': LogisticRegression(penalty='l2', C=0.005, max_iter=10000),
        'SVC': SVC(C=9, kernel ='rbf'),
        'CatBoost': CatBoostClassifier(depth=6, iterations=700, l2_leaf_reg=4,
                                                    learning_rate=0.01),
        'AdaBoost': AdaBoostClassifier(learning_rate=0.05, n_estimators=2000),
        'XGBoost': XGBClassifier(objective='binary:logistic',
                                 colsample_bytree=0.4603, gamma=0.0468,
                                 learning_rate=0.05, max_depth=10,
                                 min_child_weight=1.7817, n_estimators=3000,
                                 reg_alpha=4.5, reg_lambda=8.5,
```

```
                                subsample=0.5213,
                                random_state=42),
        'GradientBoosting': GradientBoostingClassifier(learning_rate=0.01, max_depth=8,
                                                       max_features=0.3, min_samples_leaf=
                                                       100, n_estimators=400),
        'LGBM': LGBMClassifier(n_estimators=100,
                                max_depth=7,
                                learning_rate=0.05,
                                subsample=0.2,
                                colsample_bytree=0.56,
                                reg_alpha=0.25,
                                reg_lambda=5e-08,
                                objective='binary',
                                metric='accuracy',
                                boosting_type='gbdt',
                                device='cpu',
                                random_state=1),
        'Random Forest': RandomForestClassifier(n_estimators=200, max_depth=10,
                                                       min_samples_split=10,
                                                       min_samples_leaf=1, bootstrap=True,
                                                       random_state=42),
        'DecisionTreeClassifier ' : DecisionTreeClassifier(criterion='entropy', max_depth
                                                       = 6),
    }
    # perform 10 folds cross-validation and store results
    CV_res = {}
    for name, model in models.items():
        print(f'------- Fitting {name} -------')
        score = cross_val_score(model, X_train, Y_train, cv=10).mean()
        CV_res[name] = score

    # create dataframe
    df = pd.DataFrame(list(CV_res.items()), columns=['Model', 'Score'])

    print(f'------- Printing Dataframe -------')
    return df
```

Feature Selection for CatBoostClassifier

```
model_ = CatBoostClassifier(depth=6, iterations=700, l2_leaf_reg=4, learning_rate=0.01,
                                                verbose=0)
sfs = SequentialFeatureSelector(model_, scoring='accuracy', direction='backward',
                                                n_features_to_select='auto', tol=None)
sfs.fit(X_train, Y_train)
best_features = list(sfs.get_feature_names_out())
```

Submission

```
cat = CatBoostClassifier(depth=6, iterations=700, l2_leaf_reg=4, learning_rate=0.01,
                                                verbose=0)
cat.fit(X_train[best_features], Y_train)
cat_Y = cat.predict(df_test[best_features])

res = pd.DataFrame(
        {
            'PassengerId': list(PassengerId_test),
            'Transported': [(p == 1) for p in list(cat_Y)]
        }
    )

res.to_csv('submission.csv', index = False)
```