

Rapport de projet

Apprentissage automatique

Compétition Kaggle « Spaceship Titanic »

GHAITH Sarahnour, TOUHAMI Wided, BELAIDI Saber et ROISEUX Thomas

30 mars 2023

1 Introduction

Ce projet se base sur le dataset `spaceship-titanic` de Kaggle. Il a pour but de prédire les survivants d'un vol spatial entre différentes planètes.

Pour cela, nous avons à notre disposition un dataset contenant des informations sur les passagers, incluant par exemple leur identité, leur planète de départ et d'arrivée, leur emplacement dans le vol. . .

Le tout est alors d'utiliser toutes ces informations pour prédire au mieux quels sont les passagers qui vont survivre lorsque l'accident va se produire. Pour cela, on va utiliser divers algorithmes de machine learning afin de classer au mieux les passagers qui vont survivre ainsi que ceux qui vont périr durant le voyage. Nous allons donc mettre en œuvre diverses techniques :

1. Régression logistique.
2. Arbre de décision.
3. Machine à support de vecteur.
4. Algorithme de boosting.

Mais pour que ces algorithmes soient efficaces, il est primordial de préparer les données et de les nettoyer, afin de ne garder que les variables explicatives importantes.

2 Analyse exploratoire des données

Nous avons un dataset d'entraînement et un dataset de test. Chacun d'entre eux possède les variables suivantes :

- PassengerId.
- HomePlanet.
- CryoSleep.
- Cabin.
- Destination.
- Age.
- VIP.
- RoomService.
- FoodCourt.
- FoodCourt.
- ShoppingMall.
- Spa.
- VRDeck.
- Name.
- Transported.

Au regard de toutes ces variables, certaines auront un impact dans la prédiction, tandis que d'autres pourront être écartées directement, dès l'étape de prétraitement des données (identifiant et nom).

En tout, nous avons 6 variables continues, 4 variables de catégories et 3 variables descriptives.

2.1 Dataset d'entraînement

Le dataset d'entraînement possède la forme suivante :

```
Number of rows in train data: 8693
Number of columns in train data: 14
Number of values in train data: 119378
Number missing values in train data: 2324
```

FIGURE 1 – Forme du dataset d'entraînement

On constate alors qu'il y a de nombreuses valeurs manquantes.

	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
count	8514.000000	8512.000000	8510.000000	8485.000000	8510.000000	8505.000000
mean	28.827930	224.687617	458.077203	173.729169	311.138778	304.854791
std	14.489021	666.717663	1611.489240	604.696458	1136.705535	1145.717189
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	27.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	38.000000	47.000000	76.000000	27.000000	59.000000	46.000000
max	79.000000	14327.000000	29813.000000	23492.000000	22408.000000	24133.000000

FIGURE 2 – Analyse descriptive du dataset d’entraînement

La figure 2 donne des informations statistiques sur les différentes variables explicatives numériques, avant transformation des autres variables.

2.2 Data set de test

Le dataset de test possède la forme suivante :

```
Number of rows in test data: 4277
Number of columns in test data: 13
Number of values in train data: 54484
Number of rows with missing values in test data: 1117
```

FIGURE 3 – Forme du dataset de test

Ce dataset possède également de nombreuses valeurs manquantes.

Au regard de tous ces éléments, il faudra donc considérer une solution pour les valeurs manquantes.

2.3 Visualisation des données

Nous pouvons désormais visualiser certaines données.

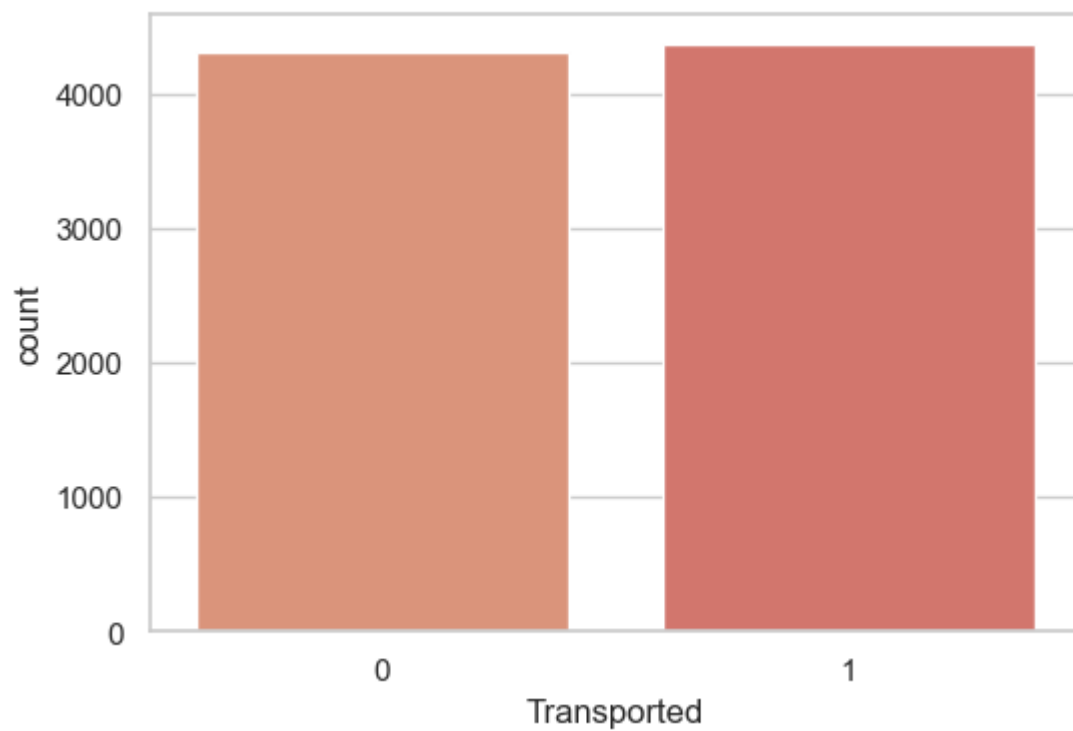
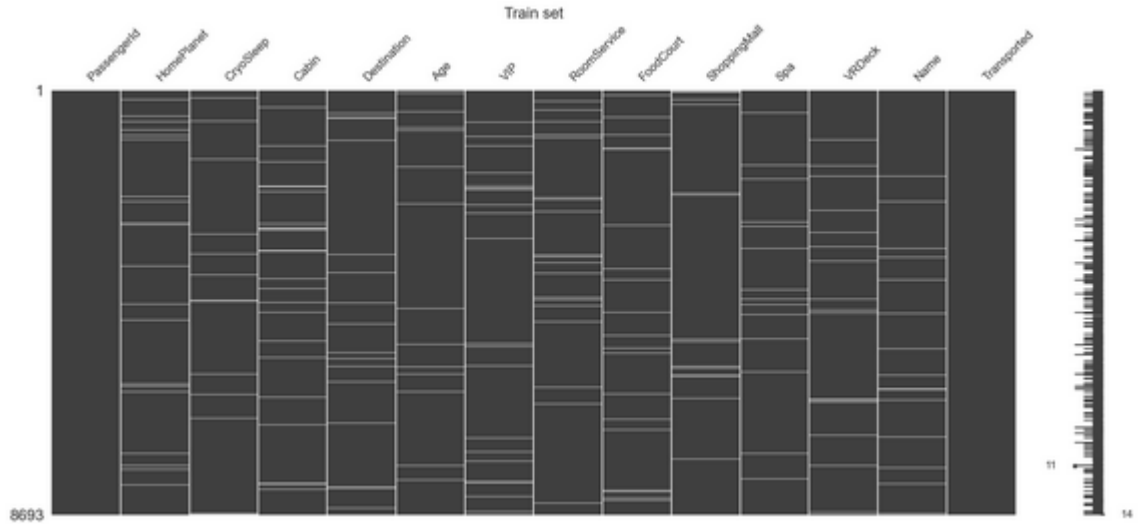


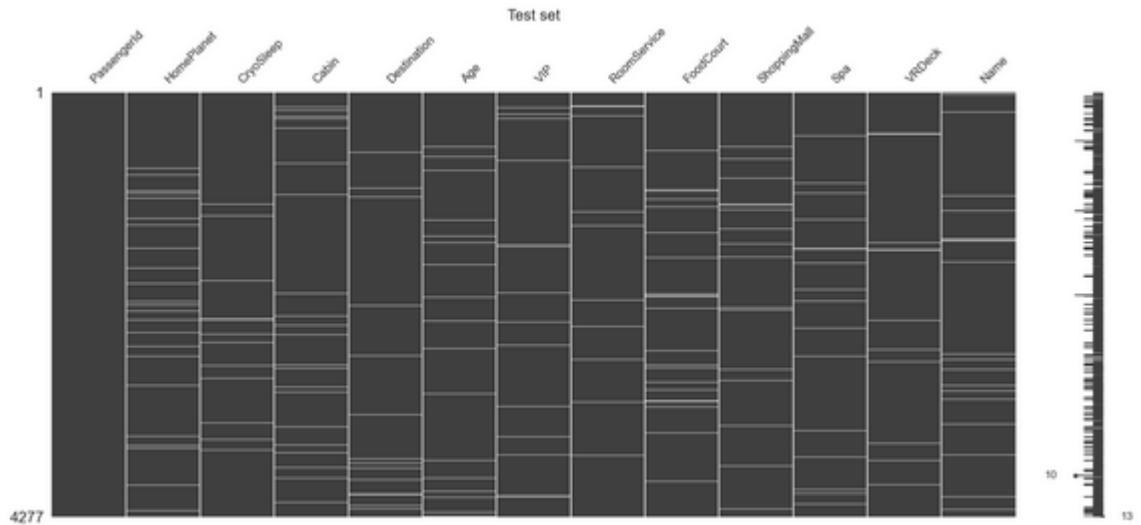
FIGURE 4 – Répartition des voyageurs arrivés ou non à destination

La figure 4 montre la répartition de personnes dont le voyage a réussi par rapport à celles dont le voyage a échoué. On constate qu'on a une répartition assez équilibrée, ce qui est prometteur pour entraîner un modèle qui prédit efficacement si le voyage réussit ou non.

2.4 Visualisation des valeurs manquantes



(a) Entrainement



(b) Test

FIGURE 5 – Visualisation des valeurs manquantes

Ces deux graphiques montrent la répartition des valeurs manquantes suivant les variables considérées. Cela permet alors de voir si une variable possède trop de valeurs manquantes pour pouvoir les remplacer. On constate ici qu'aucune variable ne présente de gros manque de valeur.

2.5 Matrice de corrélation

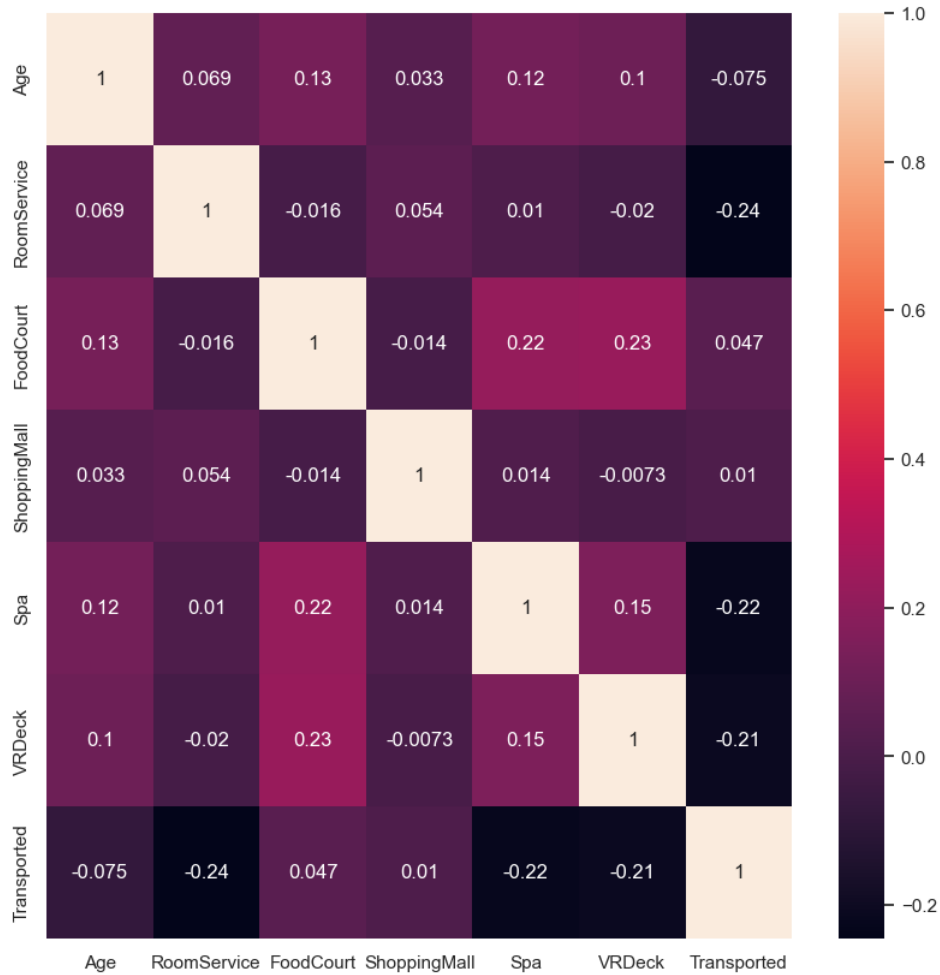


FIGURE 6 – Matrice de corrélation

La figure 6 montre la corrélation entre les différentes variables numériques. On remarque que peu de variables sont corrélées entre elles donc une technique comme la PCA ne serait pas d'une grande aide. Au contraire, il vaut mieux ici conserver toutes les variables.

La figure 7 (page 7) montre la corrélation entre toutes les variables. On remarque seulement une forte corrélation entre le nom, l'identifiant et la cabine des passagers, ce qui est assez cohérent. Il peut donc être pertinent de ne garder que l'une de ces deux variables, en l'occurrence la cabine plutôt que le nom.

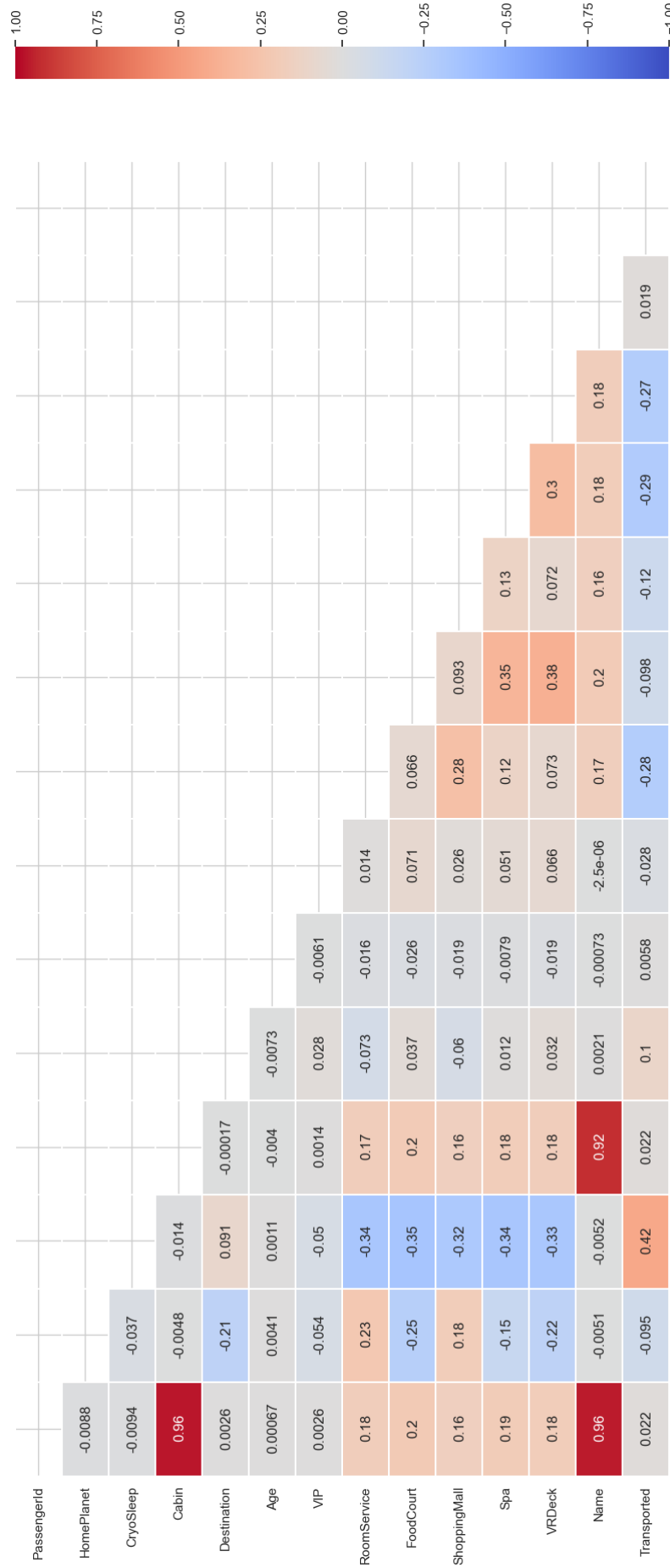


FIGURE 7 – Table de corrélation

La figure 8 montre que la plupart des passagers choisissent de ne pas dépenser plus d'argent, et que cette distribution diminue exponentiellement. On constate également quelques outliers. Mais les passagers qui arrivent à bon port semblent avoir moins dépensé que les autres.

La figure 9 montre également la répartition par catégories. On y constate que l'écrasante majorité des passagers ne sont pas VIP.

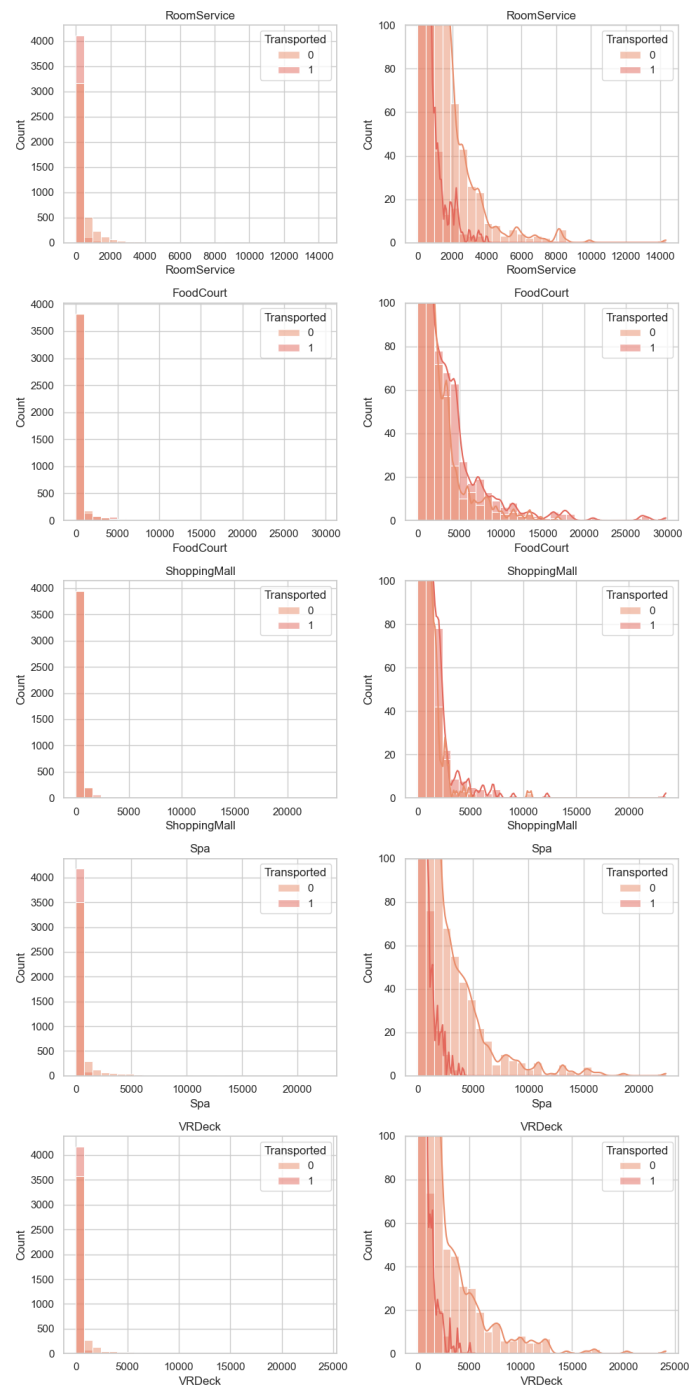


FIGURE 8 – Variables de loisir avant transformation

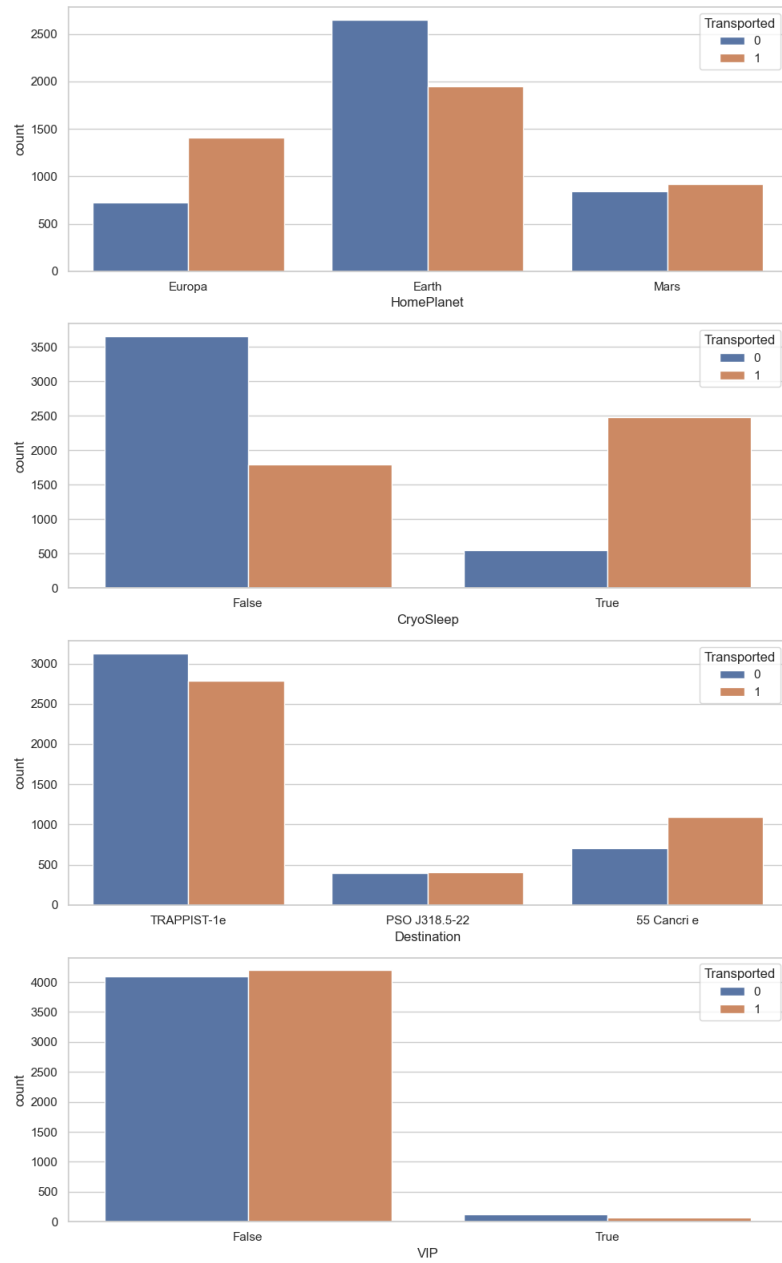


FIGURE 9 – Représentation des variables de catégories

3 Data preprocessing

Après avoir visualisé les données, il est temps de les préparer avant d'entraîner les modèles.

1. Nous avons choisi de retirer la **nom** en raison de sa forte corrélation avec la cabine, ainsi que le fait qu'elle ne sera pas utile pour prédire le succès ou non du transport.
2. Ensuite, nous avons transformé la variable **Transported** qui avait pour valeurs **True** et **False** en variable numérique, afin de pouvoir faire de la classification.
3. La visualisation des données a montré que la variable **Cabin** avait la forme suivante : $n/n/n$ avec n un entier. On a alors choisi de séparer cette variable en trois nouvelles : **deck**, **num** et **side**. Puis nous avons supprimé la variable initiale **Cabin**, celle-ci n'ayant plus d'utilité.
4. De plus, la variable **num** représentant le numéro de la cabine n'a pas une grande importance dans la prédiction, contrairement au numéro du pont ou au côté, donc nous l'avons supprimée.
5. Enfin, nous avons regroupé les variables **RoomService**, **FoodCourt**, **ShoppingMall**, **Spa**, **VRDeck** en les sommant, celles-ci étant toutes relatives au divertissement et n'impactant pas directement la réussite du transport. Cela donnera un indice de la somme d'argent dépensée.
6. Nous allons créer une variable binaire indiquant si le passager a ou non dépensé.
7. Nous allons prendre le logarithme de cela pour réduire l'écart.
8. En ce qui concerne l'âge, nous avons constaté que les passagers avaient des âges très diversifiés. Nous avons donc choisi de les catégoriser en 4 : enfant, adolescent, adulte et vétéran. Ensuite, nous avons supprimé la variable **Age**.
9. Pour éviter tout overfitting, au regard de la figure 9 sur les passagers VIP, nous avons choisi de supprimer cette variable, pour ne pas déséquilibrer le modèle.
10. De l'identifiant des passagers, nous avons extrait la valeur centrale pour les regrouper. Cela a laissé apparaître des groupes de plusieurs tailles (figure 10) donc nous avons transformé cela en variable binaire qui détermine s'il y a ou non une seule personne dans le groupe.
11. L'utilisation de boxplots a permis de détecter les différents outliers.
12. Comme de nombreuses valeurs sont manquantes, nous avons choisi de les imputer. Pour cela, nous avons utilisé l'imputeur simple fourni

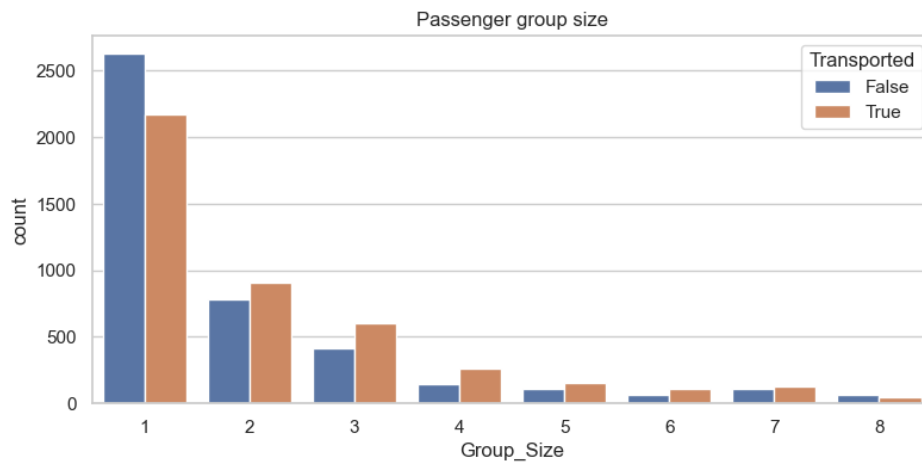


FIGURE 10 – Taille des groupes

par `scikit-learn` mais nous avons adapté la stratégie suivant les variables à imputer.

- Pour les variables nominatives, nous avons utilisé la stratégie du plus fréquent. Cela a permis de conserver la distribution globale.
- Pour les variables numériques, nous avons utilisé la stratégie de la médiane, afin de conserver une valeur médiane constante tout en ajoutant les valeurs manquantes.

4 Modèles utilisés

Afin de s'assurer de l'exactitude des prédictions en production d'un modèle en apprentissage automatique on est emmené à le tester sur de nouvelles données jamais vu auparavant.

Le processus de validation est une étape cruciale pour but de décider à quel point les résultats obtenus sont acceptables et pour ce faire nous avons opté pour la technique **Train-Test Split**.

L'approche **sklearn** suppose de commencer par créer une instance du modèle, puis de lui appliquer la méthode `.fit` avec les données et les labels d'entraînement. Nous appelons ensuite la méthode `.predict` en fournissant le modèle ajusté. Il reste ensuite à évaluer le modèle avec la méthode `.score` à laquelle nous fournissons les données et les labels de test.

4.1 Technique du Train-Test Split

C'est parmi les méthodes les plus utilisés pour l'évaluation d'un modèle en machine learning. Elle consiste principalement à diviser les données en deux ensembles différents, un ensemble pour entraîner le modèle (70 % à 80 %) et un ensemble pour le tester (20 % à 30 %).

4.2 Régression logistique

Notre problème étant un problème de classification nous avons choisi comme modèle de baseline la régression logistique. Malgré leur manque en complexité et le peu de puissance prédictive leur inclusion est une nécessité pour contextualiser les résultats des modèles entraînés.

La régression logistique est un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien, son principe est d'étudier le lien entre les variables dépendantes et les variables indépendantes.

Pour cela, cette méthode cherche toujours à maximiser la vraisemblance via un algorithme de maximisation. Dans notre cas, nous avons utilisé celui de **scikit-learn**. Ce modèle a fourni une première approche de la prédiction des survivants du vol.

Exemple d'utilisation du modèle

```

In [32]: model_Log = LogisticRegression(max_iter=10000)
          model_Log.fit(X_train1,y_train1)
          model_Log.score(X_train1,y_train1)

Out[32]: 0.7861662352602818

In [33]: y_pred_log= model_Log.predict(X_test1)

In [34]: model_Log.score(X_test1, y_test1)

Out[34]: 0.7791834387579069

```

FIGURE 11 – LogisticRegression

Les prédictions résultantes de l'application de ce modèle ne sont pas satisfaisantes.

L'attribut `.coef` du modèle ajusté permet de connaître les coefficients de la fonction de décision.

```

In [35]: model_Log.coef_

Out[35]: array([[ 0.19902899,  0.71553953, -0.19209709, -0.0726186 , -0.87305497,
                  1.36253917,  0.47007151, -1.76372805, -1.62858714, -0.53539642,
                  0.26459526, -0.69690125,  0.0188659 ,  0.07641501]])

```

FIGURE 12 – LogisticRegression Coefficients

Pour trouver un meilleur score, nous avons utilisé un autre modèle de prédiction qui donne des résultats vraiment satisfaisants.

4.3 Random forest

Une *random forest* est un algorithme d'apprentissage automatique qui est composé d'un assemblage d'arbres de décisions. Il s'agit alors de séparer l'ensemble d'apprentissage en plusieurs sous-ensembles et d'entraîner un arbre de décision sur chacun d'entre eux. Le modèle global utilise alors un système de vote pour établir la prédiction de la forêt. Dans notre cas, nous avons utilisé celui de `scikit-learn`. Concernant les paramètres de cette méthode, ils sont nombreux nous allons exposer ceux que nous avons utilisé :

- `n_estimators` : correspond au nombre d'arbres.
- `min_samples_leaf` : correspond au nombre minimal d'échantillons par feuille.

Nous avons utilisé `GridSearchCv` pour rechercher les valeurs des paramètres spécifiées pour Random Forest. Les paramètres sélectionnés sont ceux qui maximisent le score. La figure ci-dessous montre la méthode utilisée.

```

In [41]: # fine tuning random forest
model_forest = RandomForestClassifier()

forest_params = [{'n_estimators': 500},
                  {'min_samples_leaf': 4}]

forest_grid = GridSearchCV(estimator=model_forest, param_grid=forest_params, cv=5)
forest_grid.fit(X_train, y_train)
forest_grid.best_estimator_

Out[41]: * RandomForestClassifier
RandomForestClassifier(min_samples_leaf=4, n_estimators=500)

```

FIGURE 13 – Optimisation par les hyperparamètres

Random forest calculent l'importance des caractéristiques en cherchant la diminution moyenne d'impureté de chacune d'elles, autrement dit une caractéristique qui diminue l'incertitude de classification obtient un meilleur score. Comme la figure ci-dessous montre le tableau d'importances de Gini.

```

In [45]: model_forest_final.feature_importances_

Out[45]: array([0.04546948, 0.10140377, 0.01823344, 0.00088449, 0.11282245,
                0.10203256, 0.08655131, 0.11884455, 0.1015871 , 0.06262753,
                0.01963247, 0.18901791, 0.02182126, 0.01907168])

```

FIGURE 14 – Importance des caractéristiques

Les prédictions résultantes de l'application de cette méthode sont satisfaisantes.

4.4 Classificateur par boosting de gradient

Le classificateur par boosting de gradient est un algorithme d'apprentissage automatique, se basant sur des algorithmes faibles, comme les arbres de décision. Il les améliore en supposant que la fonction de perte est une fonction dérivable, rendant ainsi les arbres sous-jacents beaucoup plus efficaces.

Pour optimiser au mieux notre algorithme, nous avons utilisé la fonction `RandomizedSearchCV`, qui crée une grille d'hyperparamètres et cherche ceux qui vont donner la meilleure précision pour le modèle, en les testant successivement.

XGBoost est une variante de cet algorithme, utilisant des arbres décisionnels parallèles pour améliorer grandement l'efficacité de l'algorithme.

Catboost est un algorithme par boosting de gradient, mais qui utilise d'autres alternatives pour effectuer la classification, notamment en se basant sur des permutations.

AdaBoost est un algorithme d'apprentissage automatique, qui comme tout algorithme d'amplification de gradient, se base sur l'itération d'algorithmes faibles. Il effectue alors une somme pondérée de tous ces algorithmes faibles pour effectuer une prédiction plus précise.

5 Résultats

La figure ci-dessous montre les résultats que nous avons obtenu

	Model	Test Accuracy
0	CatBoost Classifier	0.81
1	LightGBM Classifier	0.80
2	Exterme Gradient Boost Classifier	0.80
3	Random Forest Classifier	0.79
4	Gradient Boost Classifier	0.79
5	SVM Classifier	0.79
6	AdaBoost Classifier	0.78
7	Logistic Regression	0.78
8	K-Neighbors	0.77

FIGURE 15 – Résultats

6 Performance sous Kaggle

La figure 16 illustre la performance obtenue sous Kaggle

Submission and Description		Public Score ⓘ
✓	submission.csv Complete · TOUHAMI WIDED AHLEM · 2d ago	0.80664
✓	submission.csv Complete · TOUHAMI WIDED AHLEM · 2d ago	0.80664
✓	submission.csv Complete · TOUHAMI WIDED AHLEM · 2d ago	0.78559

FIGURE 16 – Performance sous Kaggle

Et la figure 17 illustre le classement obtenue sous Kaggle (240 parmi 2371)








238	Veysel Sapan		0.80664	1	3d
239	Michael Felman		0.80664	3	9h
240	Groupe Alpha	   	0.80664	2	4m
 Your Best Entry! Your most recent submission scored 0.80664, which is an improvement of your previous score of 0.78559. Great job! Tweet this					

FIGURE 17 – Classement sous Kaggle

Nos résultats ne nous ont certes pas permis d'atteindre le haut du classement sur Kaggle mais nous avons appris à utiliser des méthodes de traitements de données et aussi des méthodes de prédiction.