Spaceship Titanic

Auteurs : *PRUDHOMME Pierre, HMADOUCH Ayoub, LECONTE Victorien, JESSEN Nathan.*

Ecole nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Sommaire

1 Introduction au défi

Le défi sur lequel nous avons travaillé, un défi *Kaggle*, ayant pour but de créer un modèle permettant de prévoir le transport de passagers d'une planète à une autre. Le défi s'inspirant du fameux paquebot *Titanic*, mais remis aux goûts du jour il s'intitule *Spaceship Titanic*.

Notre dataset fourni nous donne plusieurs informations sur les passagers. Notre objectif va être de prédire la colonne *'Transported'* pour chacun des passengers.
On repère alors certaines colonnes qui vont nous être utiles :

- **Passenger_Id** la colonne qui va nous renseigner sur la taille et le numéro du groupe.
- **Homeplanet** et **Destination** nous renseignant sur la planète de départ et celle d'arrivée.
- **CryoSleep** et **VIP** nous donnant des informations sur les conditions de vie des passagers.
- **Cabin** le numéro de cabine de chaque passager contenant la position de chaque cabine sur le vaisseau.
- **Age** l'âge du passager.
- **Gold (RoomService FoodCourt ShoppingMall Spa VRDeck)** le montant dépensé par chaque passager.
- **Name** le nom de chaque passager
- **Transported** la colonne que l'on cherche à prédire nous indiquant si le passager a été transporté ou pas.

On connait maintenant les différentes colonnes sur lesquelles on va devoir travailler pour essayer de prédire de la manière la plus précise possible la colonne *Transported* du jeu de données test.

2 Introduction au Dataset

2.1 Le dataset en quelques graphiques

On va d'abord essayer de représenter le dataset via quelques graphiques afin d'avoir un premier regard sur ce jeu de données.

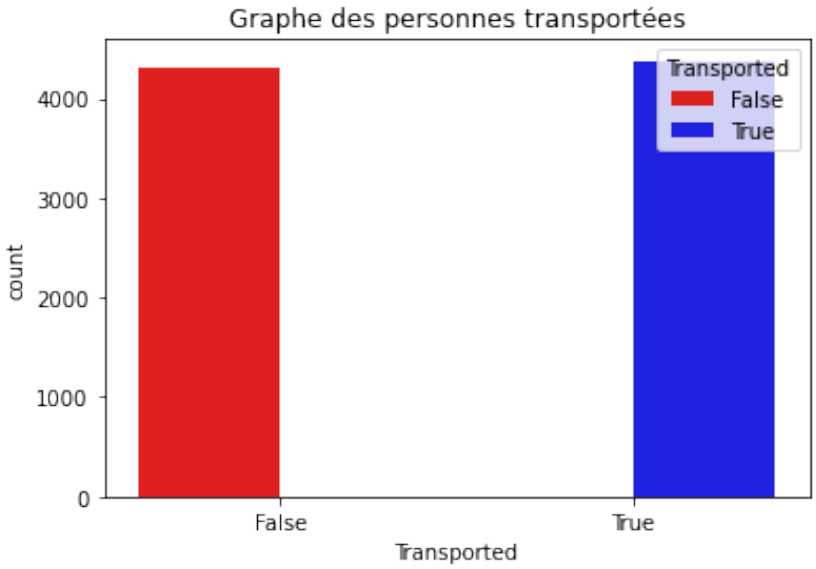
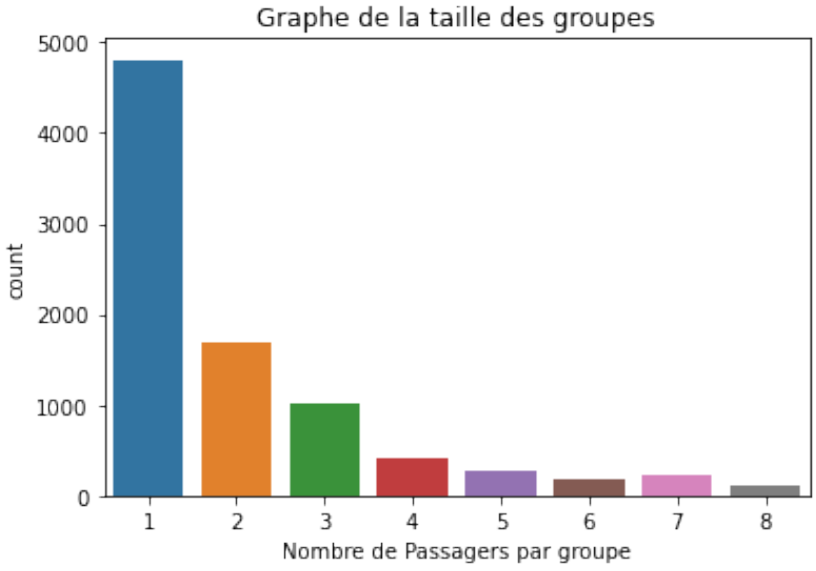


FIGURE 1 – Répartition des passagers transportés.

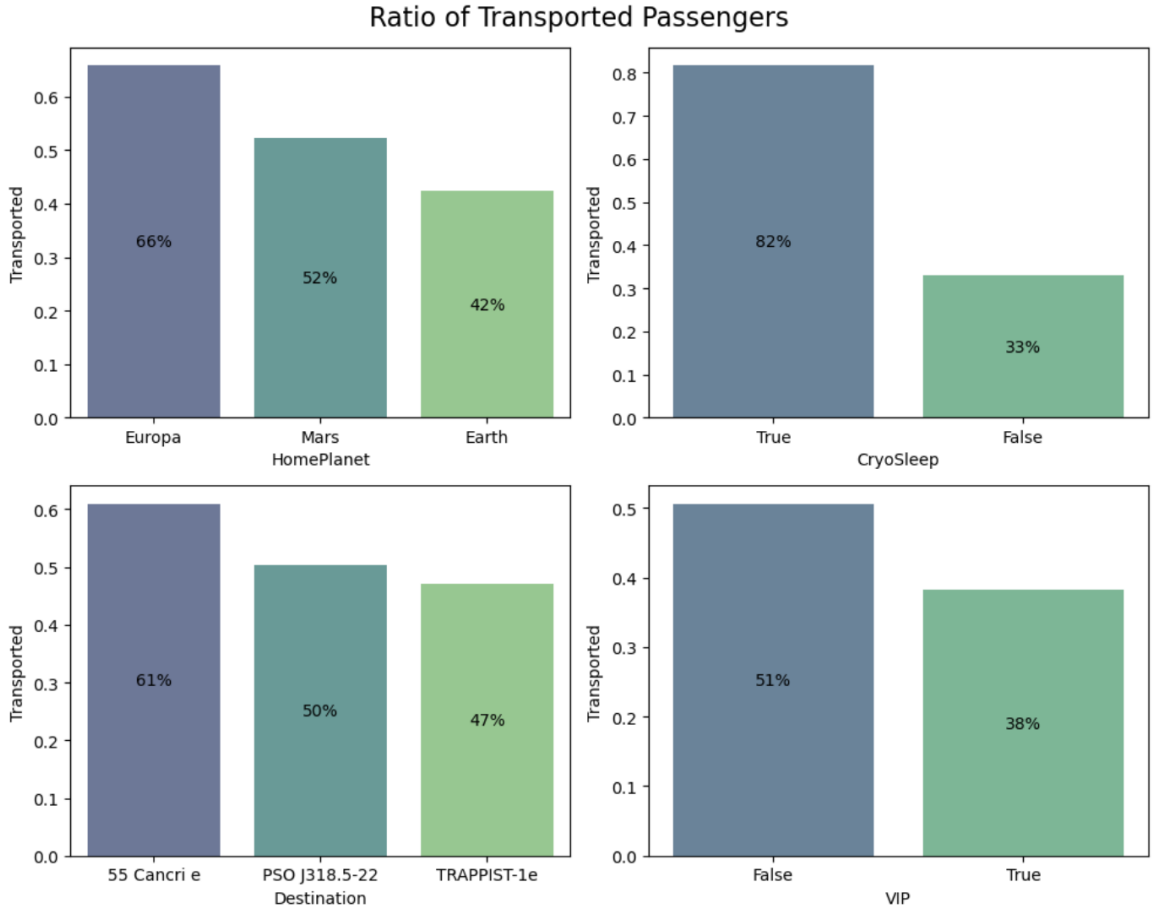
On voit donc qu'environ 1 personne sur 2 a été transportée. Ce graphique ne nous apporte pas énormément d'informations, il reste intéressant car on aurait pu apercevoir une autre tendance via ce graphe mais on observe simplement deux groupes distincts et presque équivalents.

On s'intéresse aussi rapidement aux passagers et notamment si ils voyagent seuls ou en groupe. Pour cela grâce à leur numéro de passager on va pouvoir retracer leur groupe et ainsi avoir une information sur la taille de chaque groupe.



2.2 Première idée sur les passagers transportés

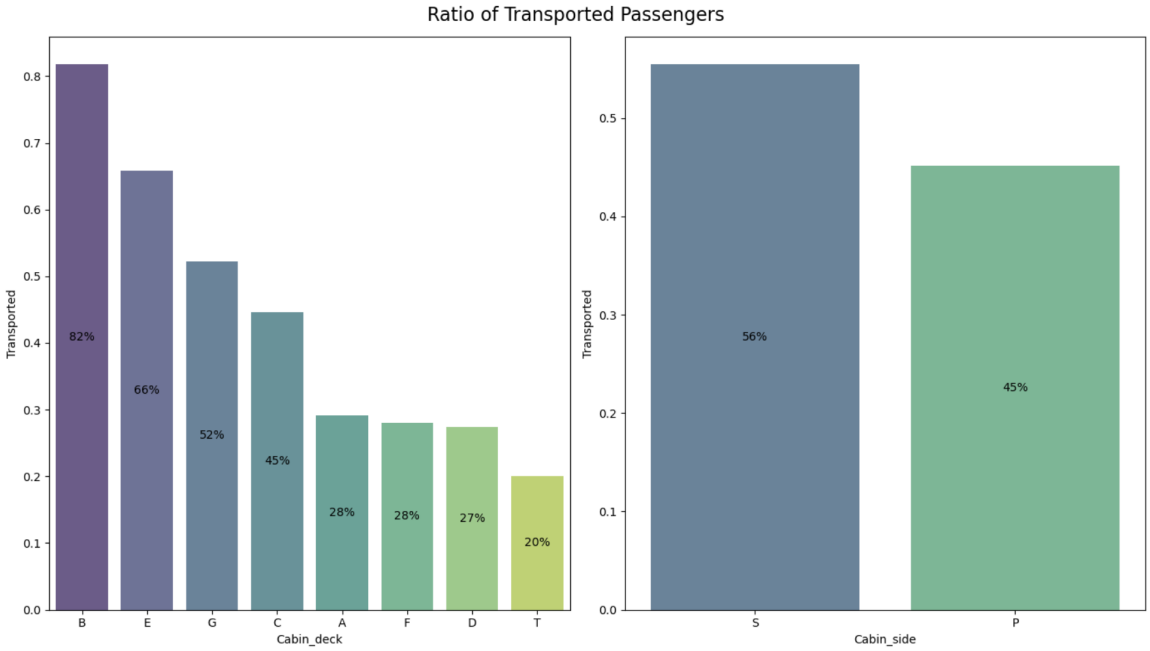
Il est important pour la suite d'avoir un premier regard sur les personnes qui ont été tranportés par rapport à ceux qui n'ont pas été transportés. Regardons donc quelques caractéristiques permettant de distinguer des éventuelles priorisations des transportés :



Grâce à ces premiers graphiques, nous pouvons remarquer plusieurs choses :

- Il y a un réel impact de la planète de provenance sur la possibilité d'être tranporté vers la destination
- On a beaucoup plus de chance d'être tansporté quand on est cryogénisé, en effet 80% des gens cryogénisés ont été transportés contrairement à 30% pour les non cryogénisés
- la destination à un impact assez faible sur la probabilité d'être transporté à l'exception peut être d'un léger avantage si on se déplace vers "55 Cancri e"
- Le fait d'être VIP ne favorise pas le transport vers sa destination

Finalement regardons l'impact du placement sur le bateau par rapport au fait d'être transporté ou non :



On remarque que les personnes sur le deck B et E ont beaucoup plus été transportés que ceux sur le deck D ou T. On remarque également que certains pourcentages sont identiques par exemple 82% du deck B a été tranporté et 82% des cryogénisés ont été tranportés, y a t il donc un étage reservé au cryogénisés ? 66% des gens provenant de Europa ont été tranportés et 66% du deck E a été transporté, y a t'il un deck reservé aux personnes provenant de Europa ?

Le dernier aspect que nous regardons pour nous faire une première idée est l'aspect pécunier : est ce que ceux qui ont dépensé plus d'argent ont plus de chance d'être transportés que ceux qui avaient peu dépensé ?

3 Traitement des données

3.1 Appropriation des valeurs manquantes

On va utiliser quelques graphiques afin de comprendre la disposition des valeurs manquantes et chercher éventuellement des liens entre celles-ci.

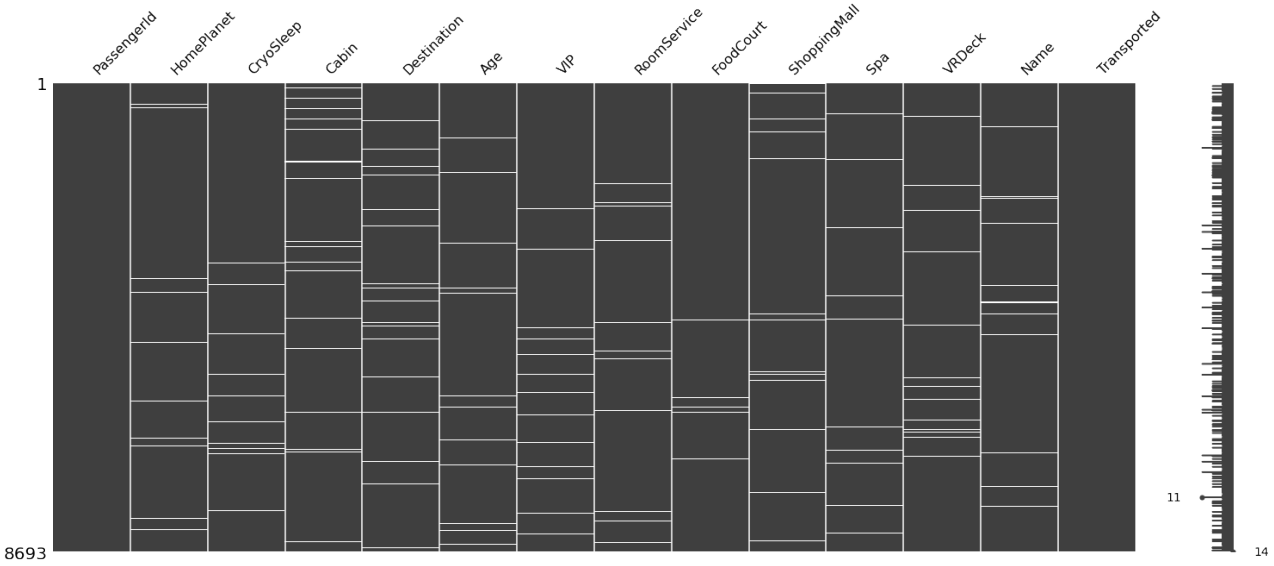


FIGURE 2 – Heatmap des valeurs manquantes

On peut ainsi représenter toutes les valeurs manquantes et avoir une première idée du travail qu'on va devoir effectuer afin de traiter au mieux la totalité des données. Et d'avoir le moins possible de données manquantes.

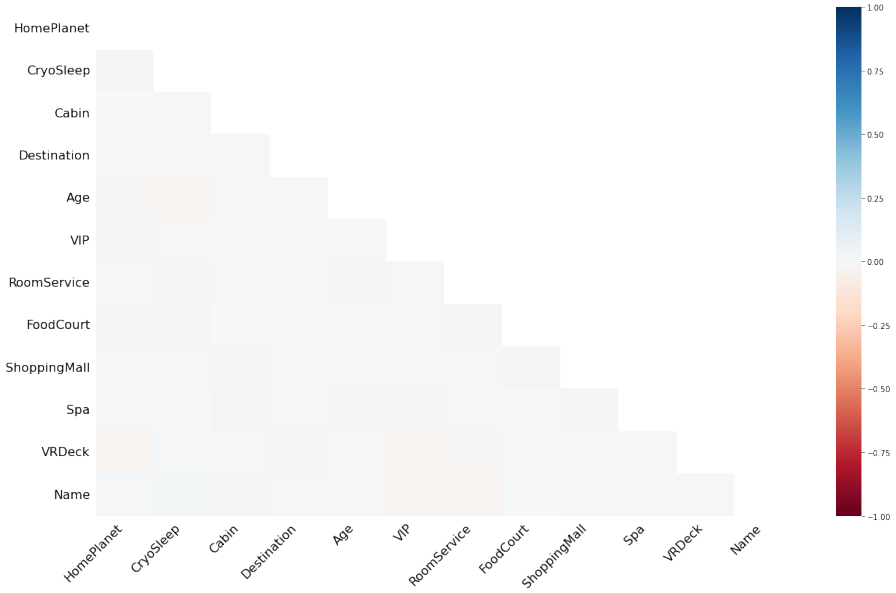
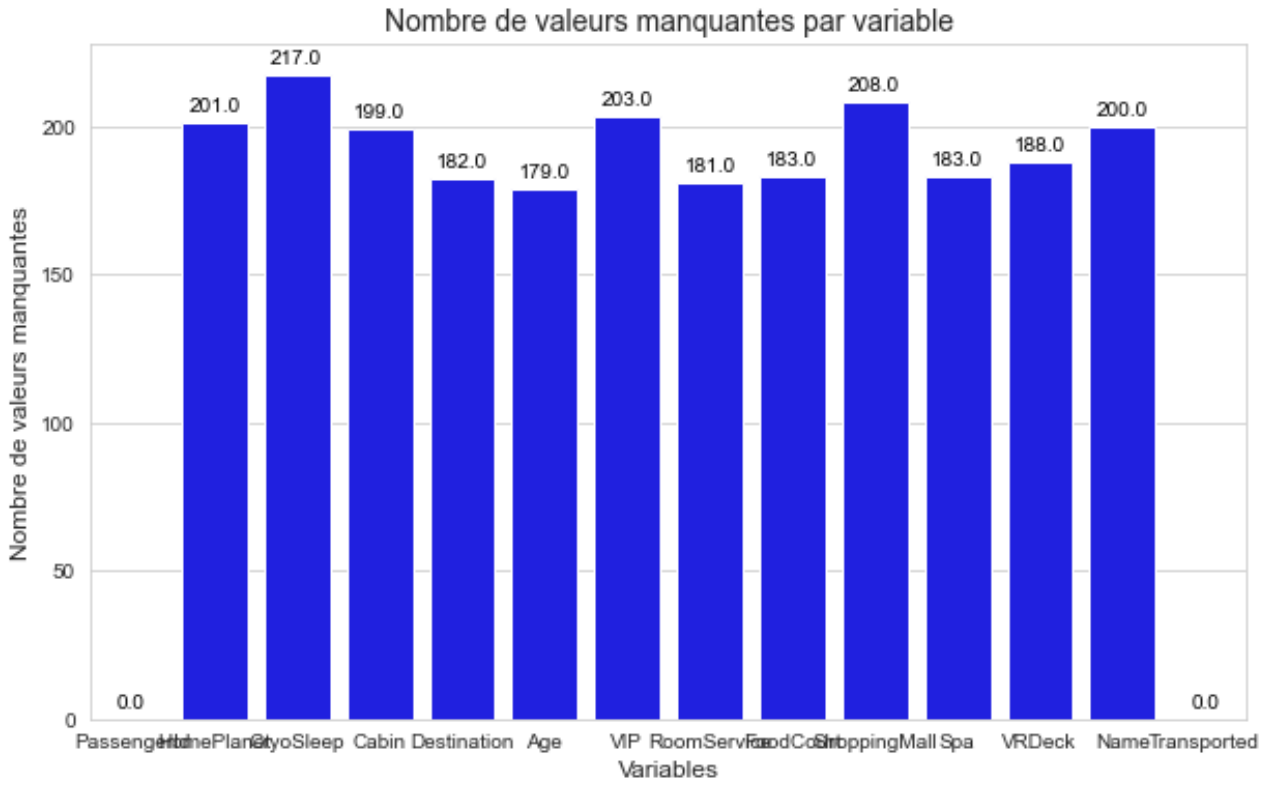


FIGURE 3 – Corrélation des valeurs manquantes

On sait grâce à ce graphique que les valeurs manquantes ne sont en aucun cas corrélées et que se sont simplement des valeurs manquantes aléatoires dans le dataset.



On voit alors qu'on a environ 200 valeurs manquantes par colonne soit environ 2% des valeurs de chaque colonne. Il est donc important d'essayer de réduire au maximum ces données afin de pouvoir les exploiter au mieux. Dans le cas ou on séparerait les différentes variables, ce pourcentage de valeurs fausse se répercuterait dans toutes les colonnes du dataset et ainsi on aurait sûrement des erreurs entraînant une diminution de la précision de nos modèles.

3.2 Traitement des valeurs manquantes

Le traitement des données est crucial pour obtenir un modèle fiable et précis. Lorsque des données sont collectées, il est fréquent que certaines valeurs soient manquantes. Cela peut être dû à diverses raisons telles que des erreurs dans la collecte de données, des problèmes techniques ou encore des données manquantes de manière intentionnelle. Cependant, il est important de traiter ces valeurs manquantes pour obtenir un modèle fiable. En effet, si ces valeurs manquantes ne sont pas traitées, cela peut entraîner une baisse significative de la précision du modèle, car les données incomplètes peuvent introduire des biais dans l'analyse et fausser les résultats. Afin de traiter des différentes valeurs manquantes, on se retrouve face à différentes règles. En effet, il s'avère que la plupart des colonnes possèdent des règles implicites

Nous allons voir pour chaque variable concernée par des valeurs manquantes ce que nous allons faire pour remplacer celles-ci par des valeurs cohérentes.

Dépenses supplémentaires (RoomService, FoodCourt, ShoppingMall, VRDeck, Spa) :

- Si le passager est en Cryosleep alors, il n'a aucune dépense supplémentaire.
- Si la personne n'est pas en Cryosleep alors, on remplit la valeur manquante par la médiane de chaque option en prenant uniquement les dépenses des personnes qui ne sont pas en Cryosleep.

Cryosleep :

- Si la somme des dépenses est positive alors la personne n'est pas en Cryosleep
- Si la somme des dépenses est nulle alors, il y a une probabilité de 83% que la personne soit en Cryosleep, on implémenterait donc ça.

VIP :

- Si la personne a moins de 18 ans alors, elle n'est pas VIP.
- Si la personne vient de la planète Earth alors, elle n'est pas VIP
- Si la personne vient de Mars en direction de 55 Cancri e alors, elle n'est pas VIP
- Si la personne est sur les deck G ou T alors, elle n'est pas VIP
- Les personnes qui ne sont pas sur les decks A, B, C, D et qui ne sont pas en Cryosleep sont des VIP
- Finalement pour les dernières valeurs manquantes on suppose que ces derniers ne sont pas VIP, car globalement, il y a un nombre très faible de VIP

Age : On va implémenter l'âge médian selon différentes catégories :

- Les personnes qui sont VIP
- Les personnes qui ont des dépenses
- Les personnes qui n'ont pas de dépenses, mais qui ne sont pas en Cryosleep
- Pour le reste des valeurs manquantes, on remplace par la valeur médiane globale des âges.

HomePlanet :

- Les VIP qui vont à 55 Cancri e viennent uniquement de Europa
- Les gens qui sont sur les decks A, B, C, T vient de Europa
- Les gens qui sont sur le deck G viennent uniquement de Earth
- Pour le restant des valeurs manquantes, on impose que les personnes viennent de Earth car c'est le plus probable.

Destination :

- On impose que les gens qui ont une valeur manquante à la place de leur planète d'arrivée se dirigent vers TRAPPIST-1e car c'est le plus probable

Cabin :

- C'était assez complexe de retrouver la cabine d'où provenait le passager ainsi, dans le codage one hot, nous mettons tous les decks à 0 et nous comptons sur la puissance du modèle en utilisant les autres données pour savoir si la personne est transportée ou non.
- Concernant le Side, nous imposons que toutes les personnes qui ont une valeur manquante se retrouvent sur le Side P.

Pour les deux dernières variables, c'est un risque que l'on prend, car cela peut augmenter le biais. En revanche les résultats sont très satisfaisants en appliquant cette méthode.

3.3 Encodage One Hot

L'encodage à chaud ou One Hot encoding en anglais est une méthode d'encodage permettant de séparer une variable avec un nombre fini d'états (n) en colonnes ajoutées à notre dataset. Ainsi on peut séparer les colonnes contenant par exemple 3 valeurs comme la colonne '*Destination*' ou encore '*HomePlanet*'.Il faut effectuer ces opérations après avoir traité le problème des valeurs manquantes, dans le cas contraire ces valeurs manquantes vont être interprétées comme des "0" par Python et on va perdre de l'information. On va alors créer des colonnes '*Destination_XXXX*' ou '*HomePlanet_XXXX*' et les remplir de 0 et de 1.

3.4 Comparaison avec d'autres résultats

Le traitement des données n'est pas le seul moyen d'obtenir un bon résultat, néanmoins il est l'une des méthodes permettant d'obtenir des résultats justes basés sur des valeurs cohérentes remplis de manière sensées. Après avoir parcouru les différentes règles du dataset nous avons réussi à remplir au maximum les valeurs manquantes du jeu de données. Avec une moyenne d'environ 200 valeurs manquantes par colonne, nous avons alors remplacé celles-ci en accord avec les règles énoncées précédemment sauf pour la destination que nous avons complété par "Trappist-1e". Mais après avoir traité ces valeurs, le résultat obtenu n'est pas à la hauteur du temps passé sur ces dernières. En effet, en traitant ces données de manière précise et une à une, le résultat obtenu est du même ordre de grandeur qu'un résultat obtenu en remplissant chaque colonne avec des "0". Nous avons quand même décidé de maintenir cette méthode car à nos yeux elle explicitait mieux chacune des valeurs données.
Le traitement des valeurs manquantes n'est pas le seul facteur à prendre en compte dans la recherche du meilleur score. En effet le modèle est aussi à prendre en compte, certains modèles sont plus performants que d'autre. Mais un autre facteur sur lequel on peut agir directement est la séparation des données et le nombre de variables d'entrée que l'on va donner à notre modèle. Selon les différentes répartitions des valeurs les résultats vont varier.

4 Application des différents modèles

4.1 RandomForest

Le random forest est l'un des algorithmes les plus utilisés à ce jour en machine learning. Plutôt intuitif et rapide à entraîner, il n'est pas le plus précis mais offre une bonne base de départ.

Son principe de fonctionnement est assez basique : l'idée est de regrouper un nombre non négligeable d'estimateurs assez faibles,(il s'agit d'arbres de décisions) et ainsi d'en créer un beaucoup plus précis. Chacun de ses estimateurs ne voit qu'une partie des données (vision parcellaire) et leur réunion permet d'obtenir des prédictions plus que satisfaisantes.

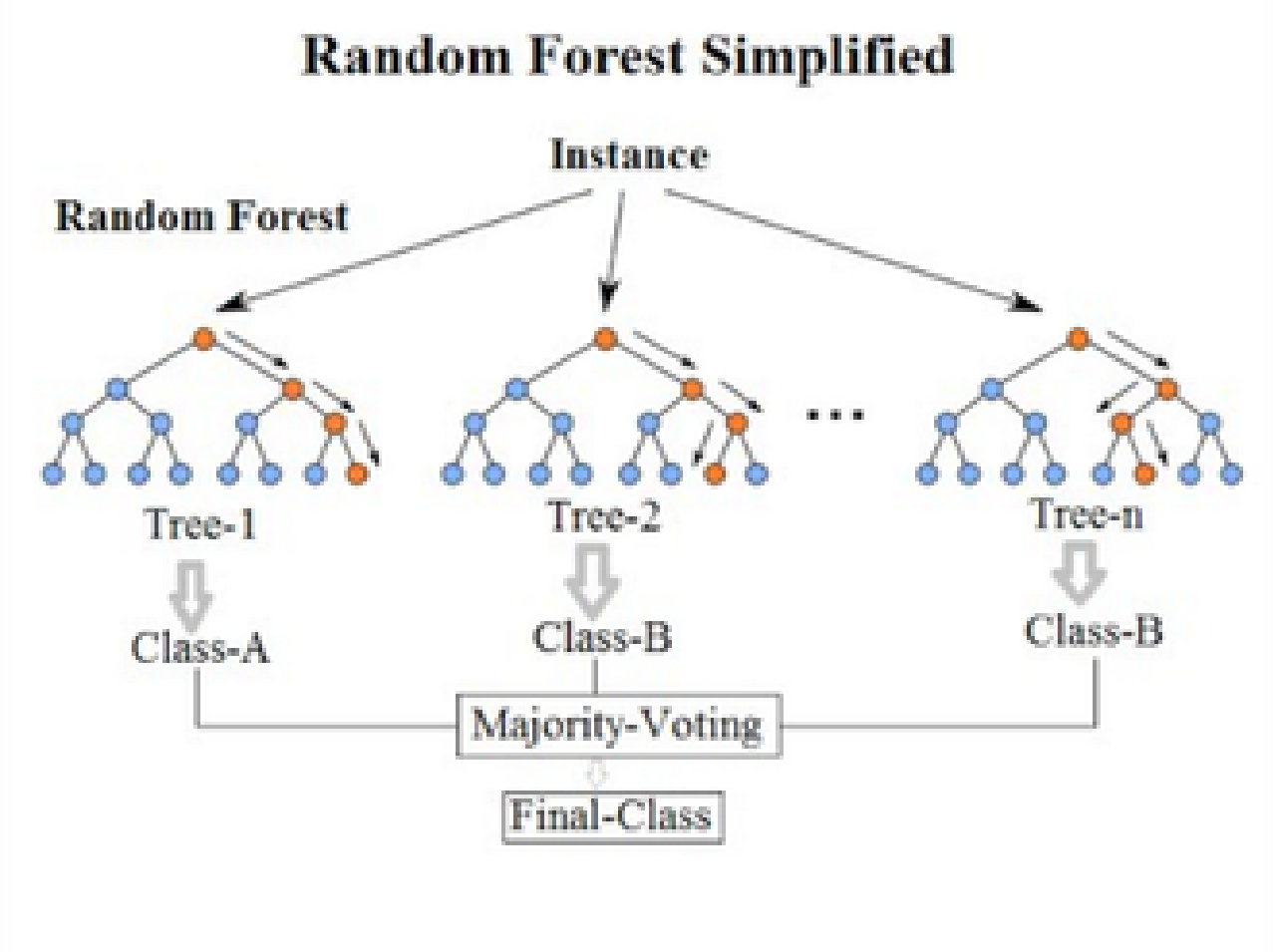


FIGURE 4 – Diagramme d'un Random forest

Ce modèle nous a donné des résulats assez rapidement, nous avons bien sûr utilisé l'outil GridSearchCV mais nous avons vite compris que le fait que nous ne connaissions pas d'encadrement précis des valeurs optimales des hyperparamètres allait fortement nous handicaper. Nous avons donc décidé d'utiliser un RandomizedSearchCV : au lieu de tester toutes les combinaisons d'hyperparamètres possibles on en teste un nombre déterminé au préalable et on ressort celle qui nous donne la meilleure accuracy. Les hyperparamètres que nous avons tuné sont les suivants : n estimators, max depth, min samples split, min samples leaf, max features et max leaf nodes. Certains sont très importants pour éviter l'overfitting tels que max depth et min samples split d'autres impactent fortement la temps de compilation tels que n estimators.

4.2 XGBoost

L'algorithme XGBoost (eXtreme Gradient Boosting) est une technique d'ensemble d'arbres de décision basée sur le boosting, qui combine plusieurs modèles faibles pour créer un modèle fort capable de prédire avec précision. XGBoost est connue pour sa capacité à produire des modèles précis, même avec de grands ensembles de données et des variables complexes. Elle utilise une méthode de gradient boosting pour améliorer la précision du modèle en augmentant progressivement la complexité de l'arbre de décision.

Pour déterminer la meilleure combinaison des paramètres pour ce modèle, nous avons utilisé la méthode **GridSearchCV** qui consiste à tester toutes les combinaisons possibles des hyperparamètres spécifiés et à évaluer les performances du modèle pour chacune de ces combinaisons.

Après l'application de cette méthode sur notre jeu de données, nous avons résultats suivants :

— **gamma = 1.5** : le paramètre de réduction de perte minimum requis pour qu'un nœud de l'arbre de décision soit divisé. Une valeur plus grande peut permettre de réduire le sur-apprentissage, en éliminant les divisions qui ne conduisent pas à une amélioration significative de la performance.

— **subsample = 0.8** : la fraction des données d'entraînement à utiliser pour chaque arbre de décision. Une valeur plus petite peut réduire le sur-apprentissage, tandis qu'une valeur plus grande peut augmenter la variance du modèle.

— **max_depth = 3** : la profondeur maximale de chaque arbre de décision de l'algorithme. Une valeur plus grande peut permettre d'obtenir un modèle plus complexe, mais peut également entraîner un sur-apprentissage.

— **colsample_bytree = 0.6** : la proportion de fonctionnalité à utiliser pour chaque arbre. Une valeur inférieure signifie que le modèle est moins susceptible de sur-apprendre, mais il peut manquer des informations importantes.

— **n_estimators = 200** : le nombre d'arbres à créer dans le modèle. Plus le nombre est élevé, plus le modèle sera complexe et précis, mais aussi plus, il sera susceptible de sur-apprendre (overfitting).

Pour mesurer la performance de notre modèle, nous avons utilisé **ROC-AUC**. C'est une métrique de performance couramment utilisée pour évaluer les modèles de classification. ROC signifie Receiver Operating Characteristic, qui est une courbe tracée en représentant le taux de vrais positifs (TPR) par rapport au taux de faux positifs (FPR) pour différents seuils de classification.

La matrice de confusion ci-dessous représente les performances de notre modèle qui prédit soit "positif" ou "négatif" pour chaque exemple dans l'ensemble de données.

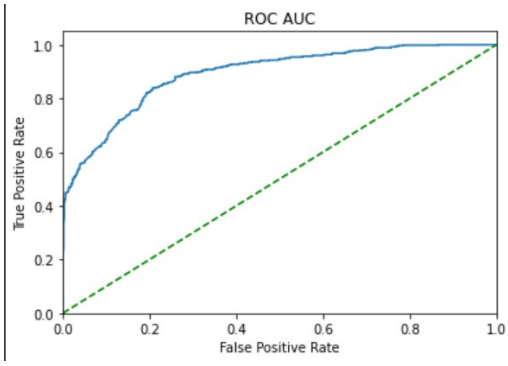
Les valeurs de la matrice de confusion sont les suivantes :

— **0.76** : La proportion de prédictions correctes pour les exemples négatifs. Cela signifie que 76% des exemples qui sont réellement négatifs ont été correctement prédits comme étant négatifs par le modèle.

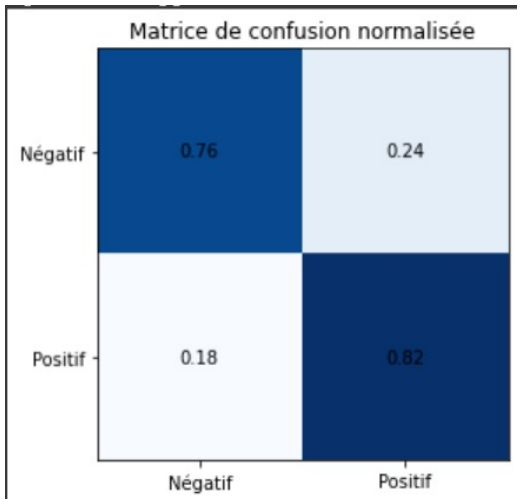
— **0.82** : La proportion de prédictions correctes pour les exemples positifs. Cela signifie que 82% des exemples qui sont réellement positifs ont été correctement prédits comme étant positifs par le modèle.

— **0.24** : La proportion d'exemples négatifs qui ont été incorrectement prédits comme étant positifs. Cela signifie que 24% des exemples qui sont réellement négatifs ont été incorrectement prédits comme étant positifs par le modèle.

— **0.18** : La proportion d'exemples positifs qui ont été incorrectement prédits comme étant négatifs. Cela signifie que 18% des exemples qui sont réellement positifs ont été incorrectement prédits comme étant négatifs par le modèle.



(a) Courbe ROC-AUC pour Xgboost



(b) Matrice de confusion

FIGURE 5 – Xgboost

4.3 CatBoost

CatBoost est un modèle très connu et capable de capturer des relations complexes entre les variables, même si elles ne sont pas linéaires. Si vous pensez que les relations entre vos variables sont complexes, CatBoost peut être un choix judicieux. Ce modèle repose sur 5 étapes toutes cruciales, nous allons les lister et expliquer chacune d'entre elle :

1. Le traitement des données : CatBoost est capable de gérer les données catégorielles (c'est-à-dire des données qui prennent des valeurs de texte ou de symboles). Il est également capable de gérer les données manquantes et les valeurs aberrantes.

2 : La construction d'arbres de décision : CatBoost crée un arbre de décision en utilisant des techniques de division des données et de recherche de la meilleure valeur de division.

3. L'entraînement des arbres : CatBoost entraîne les arbres de décision en utilisant un algorithme de descente de gradient.

4. La combinaison des arbres : CatBoost combine les arbres entraînés en un modèle en utilisant un processus de pondération et de vote.

5. La prédiction : Une fois que le modèle est créé, il peut être utilisé pour faire des prédictions sur de nouvelles données.

Pour la mise en place du modèle nous avons décidé de ne tuner que certains des nombreux hyperparamètres du CatBoost. Nous avons modifié uniquement depth, iterations, learning rate, random strength et l2 leaf reg. Notre meilleur score, qui est de 80,967 a été obtenu avec le jeu de paramètres suivant : depth=6, iterations=1000, learning rate=0.0415, random strength=9, l2 leaf reg=8. Ce qui nous a beaucoup surpris c'est que ce jeu de paramètres a été choisi quasiment en entier de manière complètement arbitraire et même après une RandomizedSearchCV de plusieurs heures les paramètres trouvés ne nous ont pas donné de meilleur score. Ce score a été obtenu en modifiant les paramètres à la main.

4.4 SVM

Le modèle de Machine Learning SVM (Support Vector Machines) est une méthode de classification supervisée qui permet de trouver une frontière de décision optimale entre deux classes dans notre cas.

Le principe de base de SVM est de trouver un hyperplan qui sépare les données en deux classes de manière optimale. Un hyperplan est une frontière de décision qui permet de séparer les points de deux classes dans un espace à n dimensions, où n est le nombre de caractéristiques de chaque point.

Le SVM recherche l'hyperplan qui maximise la marge, qui est la distance entre l'hyperplan et les points les plus proches de chaque classe. Les points les plus proches de chaque classe sont appelés vecteurs de support, d'où le nom de "Support Vector Machines".

Le modèle SVM peut être utilisé pour résoudre des problèmes de classification linéaire, où les données peuvent être séparées par un hyperplan, mais il peut également être étendu à des problèmes de classification non linéaire, en utilisant des techniques de noyau (kernel trick) qui permettent de projeter les données dans un espace de dimension supérieure.

L'entraînement du modèle SVM implique la résolution d'un problème d'optimisation convexe, qui consiste à trouver les poids optimaux pour chaque caractéristique des données et les coefficients qui définissent l'hyperplan. Cette optimisation est généralement réalisée à l'aide d'un algorithme d'optimisation tel que le Lagrangien augmenté.

Une fois le modèle entraîné, il peut être utilisé pour prédire la classe de nouveaux points en les projetant dans l'espace des caractéristiques et en les classant en fonction de leur position par rapport à l'hyperplan.

Avec ce modèle nous avons eu de bon résultats mais pas aussi bon qu'en utilisant le modèle catboost. En effet nous avons eu un score de 0.79378 en utilisant un noyau rbf(Radial Bases Function)

5 Conclusion

L'objectif de ce projet était de prédire si le voyage d'un passager de vaisseau spatial va bien se dérouler ou non. Pour cela on a à notre disposition un jeu de données d'entraînement ainsi qu'un jeu de données de test. On va alors préparer les données et entraîner différents modèles sur ce jeu, dans le but d'être le plus précis possible. Dans notre cas nous arrivons à obtenir un score de 80,967 correspondant au pourcentage de bonnes prédictions.

Pour obtenir un tel résultat nous avons effectué plusieurs étapes détaillées au dessus. Ce projet nous a permi de comprendre l'importance du travail en amont avant d'effectuer une quelconque utilisation des ressources. Le travail de Data Cleaning est une des parties les plus longues dans la mise en place de ce projet. Pour cela nous avons utilisé plusieurs outils et règles implicites au dataset. Néanmoins il reste impossible de traiter les données endommagées ou manquantes de tout le dataset.

L'utilisation du Random forest nous a semblé un choix judicieux, mais nous n'avons pas réussi à dépasser un score de 80%. Nous avons donc décidé de changer de modèle et nous avons eu l'idée d'en utiliser un autre dont l'efficacité n'est plus à prouver : le catboost. Ce modèle est connu pour être très précis et efficace, même avec des ensembles de données complexes et de grandes tailles. Il est également facile à utiliser et ne nécessite pas de réglage complexe des paramètres. Grâce à ce dernier nous avons obtenu un score de 80,967% et avons décroché la 114ème place.

Nous avons choisi d'utiliser XGBoost pour sa capacité à traiter les données manquantes, ce qui nous a permis d'obtenir un modèle robuste et fiable même en présence de valeurs manquantes dans nos données. Pour améliorer encore la performance du modèle, nous avons utilisé GridSearchCV pour optimiser les valeurs des paramètres, ce qui nous a permis de trouver la meilleure combinaison pour notre jeu de données. Nous avons évalué la performance du modèle en utilisant la métrique AUC-Roc. Finalement, nous avons obtenu une accuracy de 0,79217, ce qui est un bon résultat mais le Catboost reste le meilleur.

Axes d'amélioration

Afin d'améliorer nos prédictions nous avons identifié quelques points d'amélioration dans nos méthodes. Dans le Data Cleaning, on peut consacrer plus de temps à l'étude des données afin de déceler de nouvelles règles implicites. Au lieu de remplir certaine colonnes numérique avec des médianes on peut éventuellement mettre une distribution normale afin de conserver la variance. Une colonne a aussi échapper à notre traitement, on pourrait donc plus se pencher sur cette colonne et chercher des règles internes au jeu de données.

On pourrait aussi chercher à expliciter au mieux les valeurs disponibles en sortie de modèle grâce à des packages python comme SHAP ou encore ELI5. Permettant de préciser l'impact de chaque variable dans la construction de la sortie du modèle.

A Code source

spaceship_off_3

March 30, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib
import seaborn as sns
import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

[188]:
```
data_train = pd.read_csv("train.csv")

data_test = pd.read_csv("test.csv")
data_train

df = data_train.copy(deep=True)
```

[153]:
```
data_train.isna().sum()
```

[153]:
```
PassengerId       0
HomePlanet      201
CryoSleep       217
Cabin           199
Destination     182
Age             179
VIP             203
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Name            200
Transported       0
dtype: int64
```

[154]:
```
# If passenger is in cryosleep then all the amenities are 0
col_list = ['RoomService', 'FoodCourt', 'ShoppingMall', 'VRDeck','Spa']
df.loc[df.CryoSleep.eq(True), col_list] = 0
```

[155]:
```
# Where summ of amenities is 0 we also impute 0 for missing values
zero_amenities = df[col_list].sum(axis=1).eq(0)
df.loc[zero_amenities, col_list] = 0
```

[156]:
```
# If the passenger is not in cryosleep then we put the median ofthe expenses of␣
 ↪each amenitie for people not in cryosleep
for i in col_list:
    df.loc[df[i].isna(), i] = df.loc[df[i].gt(0), i].median()
```

[157]:
```
#Summ of all the expenses
df['Gold'] = df[col_list].sum(axis = 1)
```

[158]:
```
## If Gold>0 and we don not know if he is in cryolseep then we set cryolseep to␣
 ↪false
df.loc[(df.CryoSleep.isna() & df.Gold.gt(0)), 'CryoSleep']=False
```

[160]:
```
#in the data we have 3653 people with gold==0 and out of these 3653, about 3000␣
 ↪of the are in cryosleep which is about 83% so
#we simulate a random distribuition where if it s greater than 0.83 it is not␣
 ↪in cryosleep otherwise it is
def fonct_cryo(x):
    if x>=0.83:
        return False
    return True

df.loc[(df.CryoSleep.isna()& df.Gold==0),'CryoSleep']=fonct_cryo(np.random.
 ↪random())
```

[196]:
```
df.groupby(['HomePlanet', 'Destination']).VIP.value_counts()
```

[196]:
```
HomePlanet  Destination    VIP
Earth       55 Cancri e    False     668
            PSO J318.5-22  False     693
            TRAPPIST-1e    False    3030
Europa      55 Cancri e    False     807
                           True       63
            PSO J318.5-22  True       10
                           False       9
            TRAPPIST-1e    False    1107
                           True       56
Mars        55 Cancri e    False     189
            PSO J318.5-22  False      39
                           True        8
            TRAPPIST-1e    False    1383
                           True       55
Name: VIP, dtype: int64
```

[ ]:
```
df.groupby('VIP').Age.min()
```

On remarque ici que Il n'y a aucun VIP qui viennent de Earth Il n'y a pas de vip de Mars vers Cancri Que le plus jeune VIP a 18 ans

[161]:
```
# We know that the youngest VIP is 18 years old
df.loc[(df.VIP.isna() & (df.Age < 18)), 'VIP'] = False
```

[162]:
```
# We know that there are no VIP's comming from earth
df.loc[(df.VIP.isna() & (df.HomePlanet == 'Earth')), 'VIP'] = False

# We know that there are no VIP's comming Mars and going to Cancri
df.loc[(df.VIP.isna()
        & (df.HomePlanet.eq('Mars'))
        & (df.Destination.eq('55 Cancri e'))), 'VIP'] = False
```

[163]:
```
df[['Cabin_deck', 'Cabin_num', 'Cabin_side']] = df.Cabin.str.split("/",␣
 ↪expand=True)
```

[ ]:
```
df.groupby(['VIP', 'CryoSleep']).Deck.value_counts()
```

On remarque ici qu'il n'y a pas de VIP sur les deck G et T

Les VIP qui sont en Cryosleep sont seulement sur les decks A, B, C, D

[164]:
```
# We know that there are no VIP passengers on deck G and T
df.loc[(df.VIP.isna()
        & (df.Cabin_deck.isin(['G', 'T']))), 'VIP'] = False

# We Know that people who are not on decks A to D and not in CryoSleep are VIP's
df.loc[df.VIP.isna()
       & df.CryoSleep.eq(False)
       & ~df.Cabin_deck.isin(['A', 'B', 'C', 'D']), 'VIP'] = True


#For the rest just put nan VIPs to false as probability is low to  be a vip
df.loc[df.VIP.isna() , 'VIP'] = False
```

[165]:
```
# The VIPs going to Cancri are just from Europa
df.loc[(df.HomePlanet.isna()
        & df.VIP.eq(True)
        & df.Destination.eq('55 Cancri e')), 'HomePlanet'] = 'Europa'
```

[166]:
```
# Impute HomePlanet based on previous distribution analysis
df.loc[(df.HomePlanet.isna()
        & df.Cabin_deck.isin(['T', 'A', 'B', 'C',])), 'HomePlanet'] = 'Europa'

df.loc[(df.HomePlanet.isna() & df.Cabin_deck.eq('G')), 'HomePlanet'] = 'Earth'
```

```
df.loc[(df.HomePlanet.isna() & df.Cabin_deck.eq('G')), 'HomePlanet'] = 'Earth'
```

[167]:
```
df.loc[df.HomePlanet.isna(), 'HomePlanet']
```

[167]:
```
59      NaN
186     NaN
225     NaN
291     NaN
405     NaN
       ...
8489    NaN
8515    NaN
8613    NaN
8666    NaN
8674    NaN
Name: HomePlanet, Length: 107, dtype: object
```

[168]:
```
df.groupby('Destination').count()
```

[168]:

Destination	PassengerId	HomePlanet	CryoSleep	Cabin	Age	VIP
55 Cancri e	1800	1788	1800	1750	1766	1800
PSO J318.5-22	796	790	796	778	782	796
TRAPPIST-1e	5915	5827	5915	5788	5787	5915

Destination	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name
55 Cancri e	1800	1800	1800	1800	1800	1757
PSO J318.5-22	796	796	796	796	796	779
TRAPPIST-1e	5915	5915	5915	5915	5915	5779

Destination	Transported	Gold	Cabin_deck	Cabin_num	Cabin_side
55 Cancri e	1800	1800	1750	1750	1750
PSO J318.5-22	796	796	778	778	778
TRAPPIST-1e	5915	5915	5788	5788	5788

[169]:
```
# Impute median Age for people with VIP status
df.loc[((df.VIP == True) & df.Age.isna()), 'Age'] = df.loc[(df.VIP == True),␣
 ↪'Age'].median()

# Impute median Age for people that have expenses
df.loc[(df.Age.isna() & df.Gold.gt(0)), 'Age'] = df.loc[df.Gold.gt(0), 'Age'].
 ↪median()

# Impute median Age for people with no expenses and not in CryoSleep
```

```
df.loc[(df.Age.isna() & df.Gold.eq(0) & df.CryoSleep.eq(False)), 'Age'] = df.
 ↪loc[(df.Gold.eq(0) & df.CryoSleep.eq(False)), 'Age'].median()

# Impute an overall median Age for people not included in previous groups
df.Age.fillna(df.Age.median(), inplace=True)
```

[170]:
```
df.groupby('HomePlanet').count()
```

[170]:

HomePlanet	PassengerId	CryoSleep	Cabin	Destination	Age	VIP	\
Earth	4663	4663	4568	4561	4663	4663	
Europa	2164	2164	2103	2127	2164	2164	
Mars	1759	1759	1722	1717	1759	1759	

HomePlanet	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name	\
Earth	4663	4663	4663	4663	4663	4555	
Europa	2164	2164	2164	2164	2164	2118	
Mars	1759	1759	1759	1759	1759	1718	

HomePlanet	Transported	Gold	Cabin_deck	Cabin_num	Cabin_side
Earth	4663	4663	4568	4568	4568
Europa	2164	2164	2103	2103	2103
Mars	1759	1759	1722	1722	1722

[171]:
```
# We drop these variables to avoid overfitting of cabin num as it is not␣
 ↪relevant
df = df.drop(['PassengerId', 'Name','Cabin_num'], axis = 1)
```

[172]:
```
# For the remaining missing values Of Home Planet we input the most likely␣
 ↪planet of destination and home planet
df.loc[df.HomePlanet.isna(),'HomePlanet']='Earth'
df.loc[df.Destination.isna(),'Destination']='TRAPPIST-1e'
```

[174]:
```
#Encodage One hot

df['HomePlanet_Europa'] = df.apply(lambda x : 1 if x['HomePlanet'] == 'Europa'␣
 ↪else 0 , axis= 1)
df['HomePlanet_Earth'] = df.apply(lambda x : 1 if x['HomePlanet'] == 'Earth'␣
 ↪else 0 , axis= 1)
df['HomePlanet_Mars'] = df.apply(lambda x : 1 if x['HomePlanet'] == 'Mars' else␣
 ↪0 , axis= 1)
df['Destination_TRAPPIST'] = df.apply(lambda x : 1 if x['Destination'] ==␣
 ↪'TRAPPIST-1e' else 0 , axis= 1)
df['Destination_55_Cancri'] = df.apply(lambda x : 1 if x['Destination'] == '55␣
 ↪Cancri e' else 0 , axis= 1)
```

```
df['Destination_MPSO'] = df.apply(lambda x : 1 if x['Destination'] == 'PSO J318.
 ↪5-22' else 0 , axis= 1)

decks = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'T']

#Ainsi ici on n'a pas trouvé de moyen de retrouver les cabins manquantes, on␣
 ↪les mets donc toutes à 0 dans l'encodage One hot
for deck in decks :
    df[f'Cabin_deck{deck}'] = df.apply(lambda x: 1 if x['Cabin_deck'] ==␣
 ↪f'{deck}' else 0, axis = 1)
```

[177]:
```
#Encodage one hot
df['CryoSleep'] = LabelEncoder().fit_transform(df['CryoSleep'])
df['VIP'] = LabelEncoder().fit_transform(df['VIP'])

# 0 for side P and 1 for side S, if it is a missning value we put it to 0 for␣
 ↪side P
df['Cabin_side'] = LabelEncoder().fit_transform(df['Cabin_side'])
```

[180]:
```
df = df.drop([ 'Destination', 'HomePlanet', 'Cabin', 'Cabin_deck',␣
 ↪'Transported'], axis = 1)
```

[189]:
```
#We resume all our data treatment in this function

def treating_data(data):
    df=df = data.copy(deep=True)

    #Taking care of the missing values in the amenities
    col_list = ['RoomService', 'FoodCourt', 'ShoppingMall', 'VRDeck','Spa']
    df.loc[df.CryoSleep.eq(True), col_list] = 0

    zero_amenities = df[col_list].sum(axis=1).eq(0)
    df.loc[zero_amenities, col_list] = 0

    for i in col_list:
        df.loc[df[i].isna(), i] = df.loc[df[i].gt(0), i].median()

    df['Gold'] = df[col_list].sum(axis = 1)

    #Taking care of people in Cryolseep

    df.loc[(df.CryoSleep.isna() & df.Gold.gt(0)), 'CryoSleep']=False

    df.loc[(df.CryoSleep.isna()& df.Gold==0),'CryoSleep']=fonct_cryo(np.random.
 ↪random())

    #Taking care of the VIP's
```

```
   df.loc[(df.VIP.isna() & (df.Age < 18)), 'VIP'] = False

   df.loc[(df.VIP.isna() & (df.HomePlanet == 'Earth')), 'VIP'] = False

   df.loc[(df.VIP.isna()
           & (df.HomePlanet.eq('Mars'))
           & (df.Destination.eq('55 Cancri e'))), 'VIP'] = False


   df[['Cabin_deck', 'Cabin_num', 'Cabin_side']] = df.Cabin.str.split("/",␣
↪expand=True)

   df.loc[(df.VIP.isna()
       & (df.Cabin_deck.isin(['G', 'T']))), 'VIP'] = False

   df.loc[df.VIP.isna()
      & df.CryoSleep.eq(False)
      & ~df.Cabin_deck.isin(['A', 'B', 'C', 'D']), 'VIP'] = True



   df.loc[df.VIP.isna() , 'VIP'] = False


   #Taking care of the Age
   df.loc[((df.VIP == True) & df.Age.isna()), 'Age'] = df.loc[(df.VIP ==␣
↪True), 'Age'].median()

   df.loc[(df.Age.isna()
       & df.Gold.gt(0)), 'Age'] = df.loc[df.Gold.gt(0), 'Age'].median()

   df.loc[(df.Age.isna()
       & df.Gold.eq(0)
       & df.CryoSleep.eq(False)), 'Age'] = df.loc[(df.Gold.eq(0)
                                                   & df.CryoSleep.eq(False)),␣
↪'Age'].median()

   df.Age.fillna(df.Age.median(), inplace=True)

   df = df.drop(['PassengerId', 'Name','Cabin_num'], axis = 1)

   #Taking care of the HomePlanet and Destination

   df.loc[(df.HomePlanet.isna()
       & df.VIP.eq(True)
       & df.Destination.eq('55 Cancri e')), 'HomePlanet'] = 'Europa'
```

```
    df.loc[(df.HomePlanet.isna()
        & df.Cabin_deck.isin(['T', 'A', 'B', 'C',])), 'HomePlanet'] = 'Europa'

    df.loc[(df.HomePlanet.isna() & df.Cabin_deck.eq('G')), 'HomePlanet'] =␣
↪'Earth'

    df.loc[(df.HomePlanet.isna() & df.Cabin_deck.eq('G')), 'HomePlanet'] =␣
↪'Earth'

    df.loc[df.HomePlanet.isna(),'HomePlanet']='Earth'
    df.loc[df.Destination.isna(),'Destination']='TRAPPIST-1e'


    #encodage One hot
    df['HomePlanet_Europa'] = df.apply(lambda x : 1 if x['HomePlanet'] ==␣
↪'Europa' else 0 , axis= 1)
    df['HomePlanet_Earth'] = df.apply(lambda x : 1 if x['HomePlanet'] ==␣
↪'Earth' else 0 , axis= 1)
    df['HomePlanet_Mars'] = df.apply(lambda x : 1 if x['HomePlanet'] == 'Mars'␣
↪else 0 , axis= 1)
    df['Destination_TRAPPIST'] = df.apply(lambda x : 1 if x['Destination'] ==␣
↪'TRAPPIST-1e' else 0 , axis= 1)
    df['Destination_55_Cancri'] = df.apply(lambda x : 1 if x['Destination'] ==␣
↪'55 Cancri e' else 0 , axis= 1)
    df['Destination_MPSO'] = df.apply(lambda x : 1 if x['Destination'] == 'PSO␣
↪J318.5-22' else 0 , axis= 1)

    decks = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'T']
    for deck in decks :
        df[f'Cabin_deck{deck}'] = df.apply(lambda x: 1 if x['Cabin_deck'] ==␣
↪f'{deck}' else 0, axis = 1)


    df['CryoSleep'] = LabelEncoder().fit_transform(df['CryoSleep'])
    df['VIP'] = LabelEncoder().fit_transform(df['VIP'])
    df['Cabin_side'] = LabelEncoder().fit_transform(df['Cabin_side'])

    df = df.drop([ 'Destination', 'HomePlanet', 'Cabin', 'Cabin_deck'], axis =␣
↪1)


    return df
```

[190]:
```
X_train=treating_data(data_train)
```

[192]:
```
Y_train=X_train['Transported']

X_train=X_train.drop(['Transported'],axis=1)
```

[193]:
```
X_test=treating_data(data_test)
```

[194]:
```
from sklearn.ensemble import GradientBoostingClassifier

gb1 = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1,␣
 ↪random_state=0)


gb1.fit(X_train, Y_train)


Y_test_gradboost1=gb1.predict(X_test)


df_gradboost1 = pd.DataFrame(data_test['PassengerId'])

df_gradboost1 = df_gradboost1.assign(Transported=Y_test_gradboost1)

df_gradboost1['Transported']=df_gradboost1['Transported'].
 ↪replace([0,1],[False,True])
# Enregistrer le DataFrame en CSV
df_gradboost1.to_csv('test_spaceship_gradboost_personnal_cleaning.csv',␣
 ↪index=False)

#Avec ca on a un score de 0.80196
```

[195]:
```
X_train.to_csv("trained_data.csv", index= False)
Y_train.to_csv("Y_trained_data.csv", index= False)
X_test.to_csv("test_for_trained_data.csv", index= False)
X_train.isna().sum()
```

[195]:
```
CryoSleep       0
Age             0
VIP             0
RoomService     0
FoodCourt       0
ShoppingMall    0
Spa             0
VRDeck          0
```

```
Gold                    0
Cabin_side              0
HomePlanet_Europa       0
HomePlanet_Earth        0
HomePlanet_Mars         0
Destination_TRAPPIST    0
Destination_55_Cancri   0
Destination_MPSO        0
Cabin_deckA             0
Cabin_deckB             0
Cabin_deckC             0
Cabin_deckD             0
Cabin_deckE             0
Cabin_deckF             0
Cabin_deckG             0
Cabin_deckT             0
dtype: int64
```

[ ]:

Projet_graphique

March 30, 2023

Projet Machine Learning - Graphique

0.1 Bibliothèques

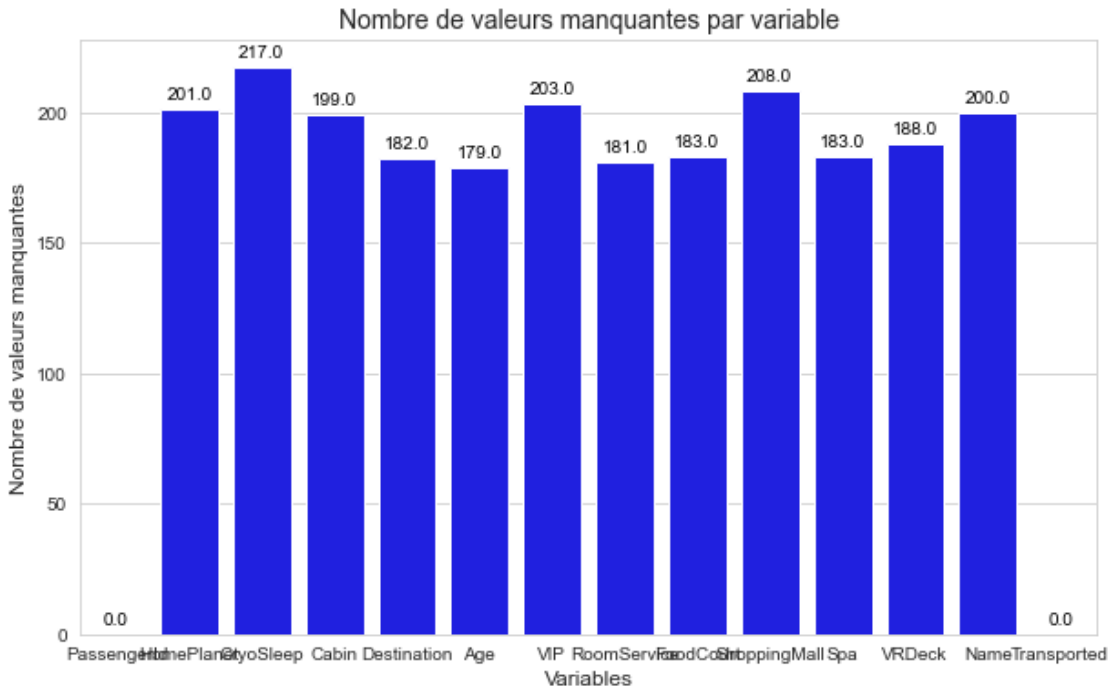
[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import missingno as msno
import configparser
```

0.2 Ouverture et Etude des Données

[2]:
```
data_train = pd.read_csv("train.csv")

data_test = pd.read_csv("test.csv")
```

[13]:
```
# Calculer le nombre de valeurs manquantes dans chaque colonne
missing_values = data_train.isnull().sum()

# Créer un graphique à barres avec seaborn
sns.set_style("whitegrid")
plt.figure(figsize=(10,6))
barplot = sns.barplot(x=missing_values.index, y=missing_values.values,␣
 ↪color='blue')

# Ajouter une étiquette pour l'axe des x
plt.xlabel("Variables", fontsize=12)

# Ajouter une étiquette pour l'axe des y
plt.ylabel("Nombre de valeurs manquantes", fontsize=12)

# Ajouter un titre
plt.title("Nombre de valeurs manquantes par variable", fontsize=14)

# Ajouter la valeur de chaque barre
```

```
for bar in barplot.patches:
    plt.annotate(format(bar.get_height()),
                 (bar.get_x() + bar.get_width() / 2, bar.get_height()),
                 ha='center', va='center',
                 xytext=(0, 8),
                 textcoords='offset points', fontsize=10, color='black')

# Afficher le graphique
plt.show()
```

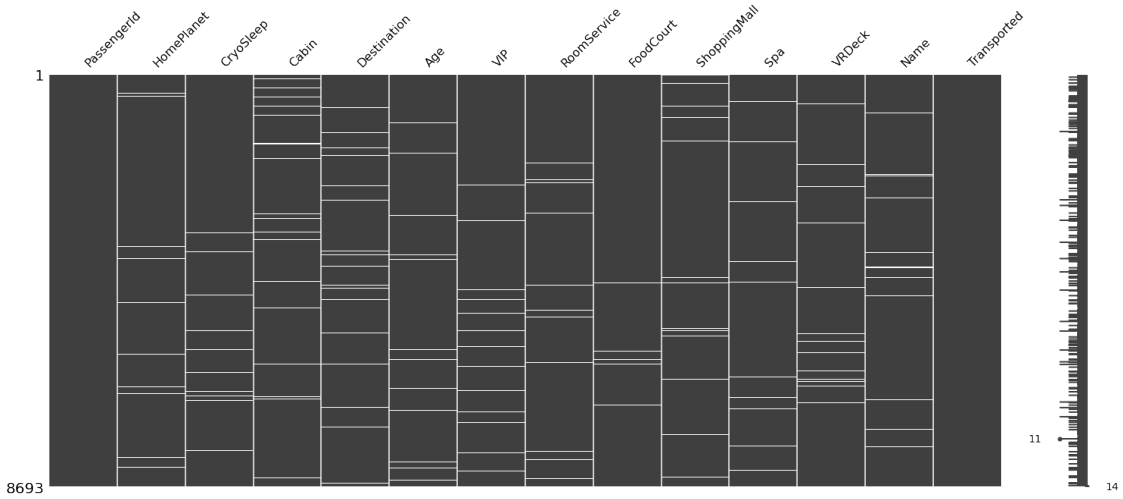


[7]:
```
display(data_train)
```

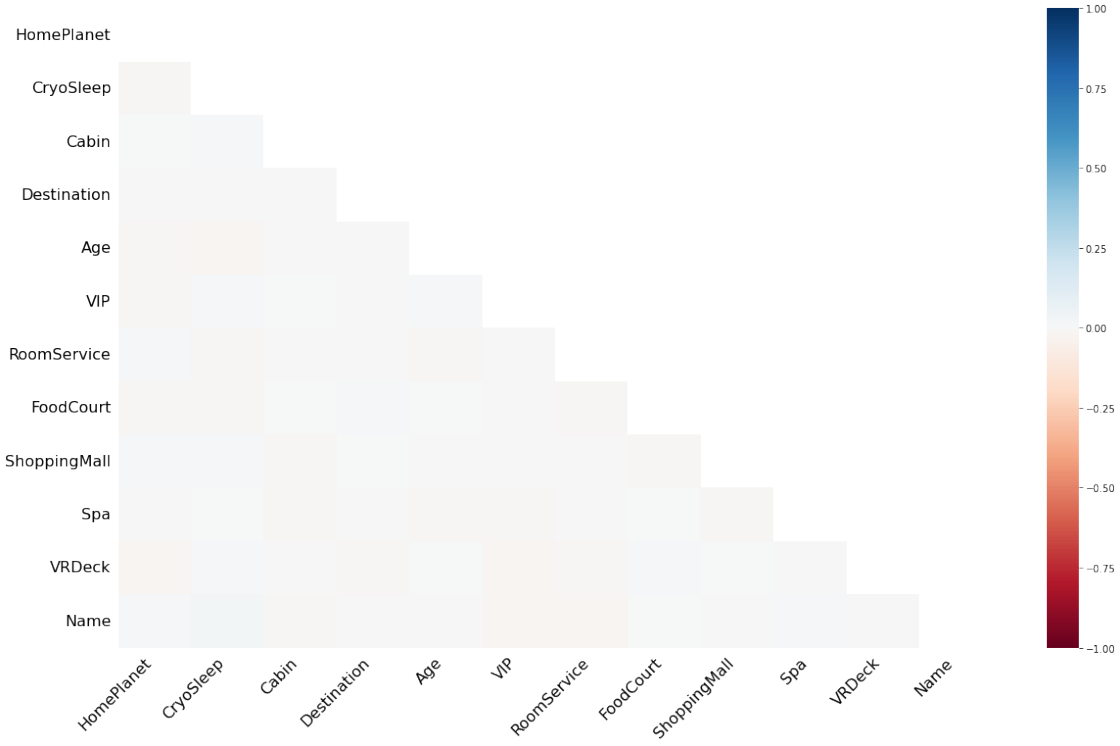
```
     PassengerId HomePlanet CryoSleep     Cabin    Destination   Age    VIP  \
0        0001_01     Europa     False     B/0/P    TRAPPIST-1e  39.0  False
1        0002_01      Earth     False     F/0/S    TRAPPIST-1e  24.0  False
2        0003_01     Europa     False     A/0/S    TRAPPIST-1e  58.0   True
3        0003_02     Europa     False     A/0/S    TRAPPIST-1e  33.0  False
4        0004_01      Earth     False     F/1/S    TRAPPIST-1e  16.0  False
…            …          …         …         …             …    …    …    …
8688     9276_01     Europa     False    A/98/P    55 Cancri e  41.0   True
8689     9278_01      Earth      True   G/1499/S  PSO J318.5-22  18.0  False
8690     9279_01      Earth     False   G/1500/S    TRAPPIST-1e  26.0  False
8691     9280_01     Europa     False    E/608/S    55 Cancri e  32.0  False
8692     9280_02     Europa     False    E/608/S    TRAPPIST-1e  44.0  False
```

```
      RoomService  FoodCourt  ShoppingMall     Spa  VRDeck              Name  \
0             0.0        0.0           0.0     0.0     0.0   Maham Ofracculy
1           109.0        9.0          25.0   549.0    44.0      Juanna Vines
2            43.0     3576.0           0.0  6715.0    49.0     Altark Susent
3             0.0     1283.0         371.0  3329.0   193.0      Solam Susent
4           303.0       70.0         151.0   565.0     2.0  Willy Santantines
…             …          …             …       …       …                 …
8688          0.0     6819.0           0.0  1643.0    74.0  Gravior Noxnuther
8689          0.0        0.0           0.0     0.0     0.0   Kurta Mondalley
8690          0.0        0.0        1872.0     1.0     0.0      Fayey Connon
8691          0.0     1049.0           0.0   353.0  3235.0   Celeon Hontichre
8692        126.0     4688.0           0.0     0.0    12.0   Propsh Hontichre

      Transported
0           False
1            True
2           False
3           False
4            True
…             …
8688        False
8689        False
8690         True
8691        False
8692         True

[8693 rows x 14 columns]
```

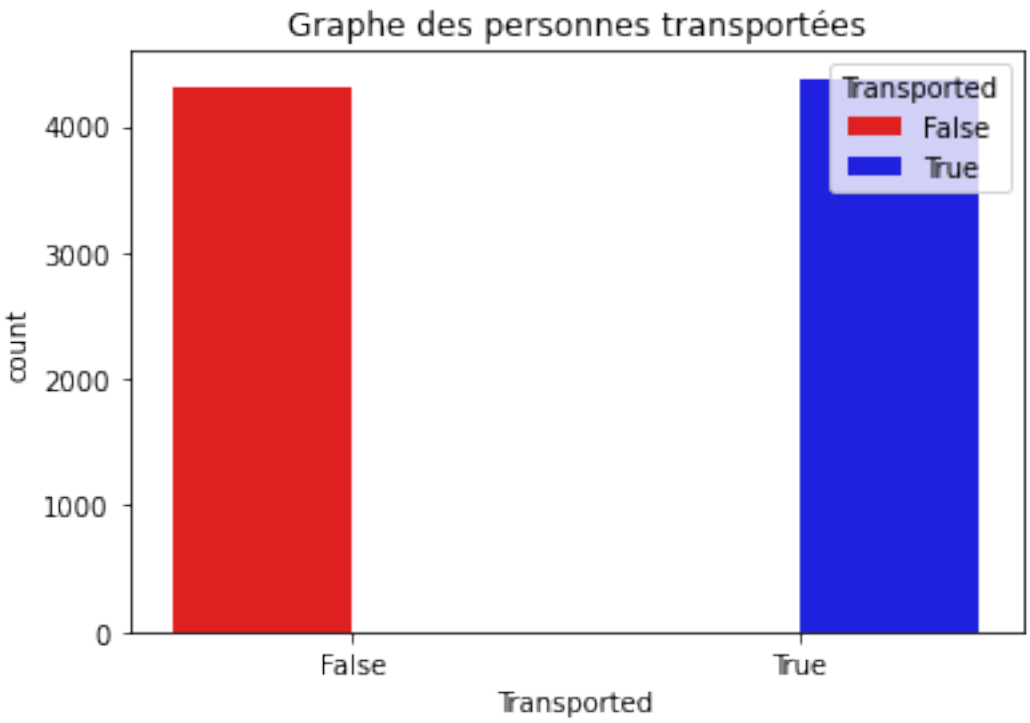
[5]:
```
msno.matrix(data_train)
plt.show()
```



[6]:
```
# correlation map of missing features
msno.heatmap(data_train)
plt.show()
# The visualizations show that there does not appear to be any correlation␣
↪among the missing features, indicating that they are missing at random.
```



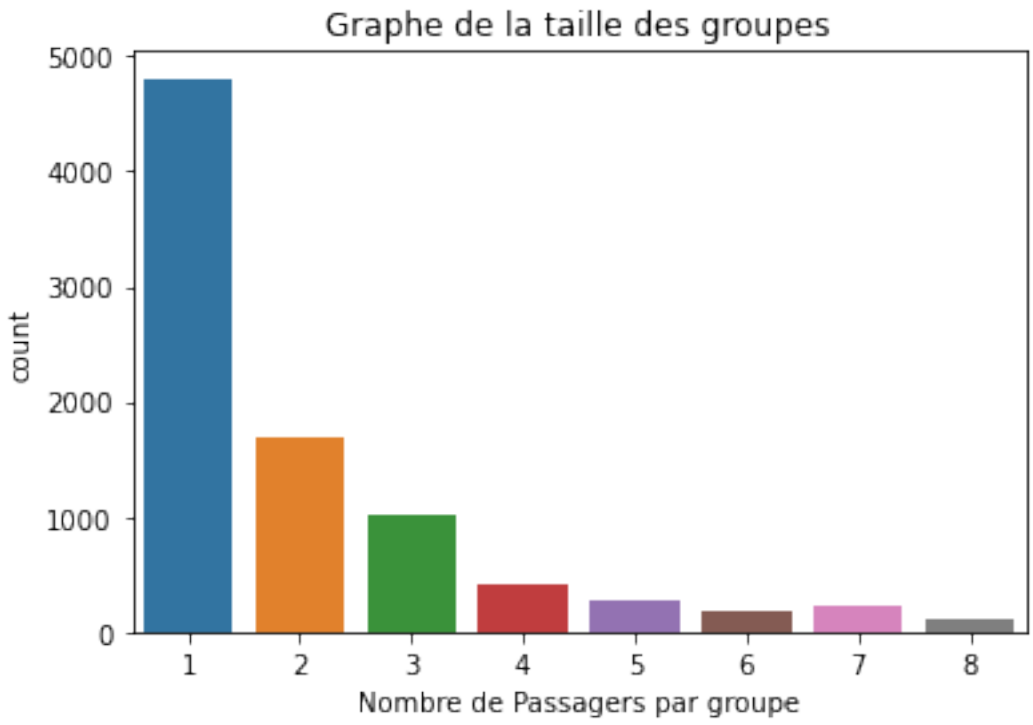
[7]:
```
sns.countplot(x=data_train['Transported'], hue = data_train['Transported']␣
↪,orient='h', palette=['red', 'blue'])
plt.title('Graphe des personnes transportées')
plt.show()
```



On peut voir que la moitié des passagers est transporté et l'autre moitié ne l'est pas.
On va maintenant extraire le groupe de chaque passager et ensuite compter le nombre de passagers présents par groupe.

[8]:
```
data_train['Passenger_group'] = data_train['PassengerId'].apply(lambda x : x.
 ↪split('_')[0]).astype(str)
data_test['Passenger_group'] = data_test['PassengerId'].apply(lambda x : x.
 ↪split('_')[0]).astype(str)
unique_group = data_train['Passenger_group'].nunique()
#6217

data_train['Group_Size'] = data_train['Passenger_group'].map(lambda x: pd.
 ↪concat([data_train['Passenger_group'], data_test['Passenger_group']]).
 ↪value_counts()[x])
```

[9]:
```
sns.countplot(x=data_train['Group_Size'],orient='h')
plt.title('Graphe de la taille des groupes')
plt.xlabel('Nombre de Passagers par groupe')
plt.show()
```



Nombre de passagers par groupe

[19]:
```
df = data_train.copy(deep=True)
col_list = ['RoomService', 'FoodCourt', 'ShoppingMall', 'VRDeck']

df['Gold'] = df[col_list].sum(axis = 1)

df[['Cabin_deck', 'Cabin_num', 'Cabin_side']] = df.Cabin.str.split("/",␣
 ↪expand=True)
df[['Group_id', 'People_id']] = df.PassengerId.str.split("_", expand = True)
# Pour l'age on va ajouter la moyenne de l'age dans chaque NA
moy_age = df.Age.mean()
df.Age.fillna(moy_age, inplace = True)

#On ajoute juste un nom 'Inconnu' a chaque valeur manquante
df.Name.fillna('Inconnu', inplace=True)

#People in a group have the same HomePlanet
#link between HomePlanet and CabinDeck:
#Earth on E, F,G
#Mars on D, E, F
#Europa on A, B, C, D, E, T
def fill_home_planet(row):
    if row.Cabin_deck == "E":
```

```
        return "Earth"
    if row.Cabin_deck == "D":
        return "Mars"
    if row.Cabin_deck == "A" or row.Cabin_deck == "B" or row.Cabin_deck == "C"␣
 ↪or row.Cabin_deck == "T":
        return "Europa"
    return row.HomePlanet

df.HomePlanet = df.apply(fill_home_planet, axis=1)

#People not Children and not CryoSleep with no bill all have Destination␣
 ↪TRAPPIST-1e
#People under 25 and coming from europa are NOT going to PSO ...
def fill_destination(row) :
    if row.Age >= 12  and  row.CryoSleep == False and row.Gold == 0 :
        return "TRAPPIST-1e"
    if row.HomePlanet == 'Europa' and row.Age < 25 :
        return "TRAPPIST-1e"
    return row.Destination

df.Destination = df.apply(fill_destination, axis=1)

#On regarde les différentes colonnes contenant des informations sur les␣
 ↪dépenses effectuées au cours du voyage
#Si la personne est cryogénisé alors on sait qu il ne va rien dépenser
#Si la personnes en question a moins de 12 ans alors on sait qu'il ne peut pas␣
 ↪dépenser d'argent et ainsi on remplit la case correspondante avec un 0.
def fill_RoomService(row) :
    if row.Age <= 12 and pd.isnull(row.RoomService) :
        return 0
    if row.CryoSleep == True :
        return 0
    return row.RoomService
df.RoomService = df.apply(fill_RoomService, axis=1)

def fill_ShoppingMall(row) :
    if row.Age <= 12 and pd.isnull(row.ShoppingMall) :
        return 0
    if row.CryoSleep == True :
        return 0
    return row.ShoppingMall
df.ShoppingMall = df.apply(fill_ShoppingMall, axis=1)

def fill_FoodCourt(row) :
    if row.Age <= 12 and pd.isnull(row.FoodCourt) :
        return 0
    if row.FoodCourt == True :
```

```
        return 0
    return row.FoodCourt
df.FoodCourt = df.apply(fill_FoodCourt, axis = 1)

def fill_Spa(row) :
    if row.Age <= 12 and pd.isnull(row.Spa) :
        return 0
    if row.CryoSleep == True :
        return 0
    return row.Spa
df.Spa = df.apply(fill_Spa, axis = 1)

def fill_VRDeck(row) :
    if row.Age <= 12 and pd.isnull(row.VRDeck) :
        return 0
    if row.CryoSleep == True :
        return 0
    return row.VRDeck
df.VRDeck = df.apply(fill_VRDeck, axis = 1)

#in groups with several CabinDeck, people with no bill are necessarily on␣
 ↪CabinDeck:
#Earth on G
#Europa on B
#Mars on E or F
def fill_deck(row) :
    if row.Gold == 0 and row.HomePlanet == 'Earth' :
        return "G"
    if row.Gold == 0 and row.HomePlanet == 'Europa' :
        return "B"
    if row.Gold == 0 and row.HomePlanet == 'Mars' :
        return "E"
    return row.Cabin_deck
df.Cabin_deck = df.apply(fill_deck, axis = 1)

#Earth has no VIP
def fill_VIP(row) :
    if row.HomePlanet == 'Earth' :
        return False
    return row.VIP
df.VIP = df.apply(fill_VIP, axis = 1)

df['CryoSleep'] = LabelEncoder().fit_transform(df['CryoSleep'])
df['VIP'] = LabelEncoder().fit_transform(df['VIP'])
df['Transported'] = LabelEncoder().fit_transform(df['Transported'])

df['HomePlanet'].fillna('Europa', inplace = True)
```

```
df['RoomService'].fillna(0, inplace = True)
df['FoodCourt'].fillna(0, inplace = True)
df['ShoppingMall'].fillna(0, inplace = True)
df['Spa'].fillna(0, inplace = True)
df['VRDeck'].fillna(0, inplace = True)
df['Destination'].fillna("TRAPPIST-1e", inplace = True)

msno.matrix(df)
plt.show()
```