



TITANIC - MACHINE LEARNING FROM DISASTER

Predict which passengers survived the Titanic shipwreck using machine learning



Sommaire

I.	Introduction	2
1.	Objectif.....	2
2.	État de l’art.....	2
3.	Méthodologie.....	4
II.	Traitement et analyse des données.....	4
1.	Présentation des données	5
2.	Analyse exploratoire des données.....	6
3.	Traitement des données	9
III.	Modélisation	12
1.	Modèle Baseline.....	12
2.	Modèles implémentés	13
IV.	Présentation des résultats	17
1.	Métriques.....	17
2.	Synthèse des résultats de la modélisation.....	18
V.	Conclusion.....	19
	Sources bibliographiques :	20

I. Introduction

Le Titanic fut un paquebot britannique qui a coulé dans l'océan Atlantique Nord aux premières heures du matin le 15 avril 1912, après avoir heurté un iceberg lors de son voyage inaugural de Southampton à New York. On estimait à 2 224 le nombre de passagers et de membres d'équipage à bord du navire ; et à plus de 1 500 le nombre de morts, ce qui en fait l'une des catastrophes maritimes commerciales les plus meurtrières de l'histoire moderne en temps de paix. Le Titanic était aussi le plus grand navire à flot au moment de son entrée en service et était le deuxième des trois paquebots de classe olympique exploités par la « **White Star Line** ». Le Titanic a été construit par le chantier naval « **Harland and Wolff** » à Belfast. **Thomas Andrews**, son architecte, est mort dans la catastrophe. Avec la puissance de calcul des ordinateurs et le développement des algorithmes de machine learning, pouvons-nous développer un modèle de prédiction qui nous permettra de prédire les passagers du Titanic qui sont décédés lors du naufrage ?

Pour répondre à une telle question, **Kaggle** qui est une plateforme web organisant des compétitions en data science, a ouvert une compétition sur le Titanic à laquelle nous avons pris part.

1. Objectif

L'objectif principale de notre projet est de construire un modèle robuste de prédiction qui nous permettra de prédire les passagers du Titanic qui sont décédés lors du naufrage avec une meilleure précision possible. Afin d'atteindre notre objectif principal, nous allons nous fixer deux objectifs secondaires :

- 🔗 Obtenir une compréhension claire de notre jeu de données à travers une analyse exploratoire.
- 🔗 Implémenter de multiples modèles afin de les comparer et d'en choisir le plus performant

2. État de l'art

La prédiction des passagers du Titanic qui sont décédés lors du naufrage est une problématique liée à l'apprentissage supervisé du machine learning. En effet, les données utilisées pour l'apprentissage comportent des étiquettes ou labels qui permettent d'identifier si un passager du Titanic est décédé lors du naufrage ou ne l'est pas. Nous

avons à faire plus précisément à un problème de classification. Ce problème se décompose en trois parties :

- ✚ Les **données** en entrée du système.
- ✚ Le **classifieur** qui apprend à reconnaître les passagers décédés ;
- ✚ Les **classes** qui sont les différentes réponses possibles du classifieur : **1 pour passager décédé** ou **0 pour passager survécu**.

a) Les Kernels de Kaggle

Sur le site de **kaggle**, nous avons pu trouver des kernels¹ qui proposent un workflow dans la résolution du problème du Titanic. Ces kernels sont des notebooks interactifs qui nous permettent de suivre les étapes du travail de l'auteur.

L'un des kernels qui a beaucoup retenu notre attention est le « **Titanic Data Science EDA with meme+Solution** » (Soham Mukherjee, juin 2020). Dans son travail, Soham a implémenté huit (08) modèles de classification binaire à savoir : la régression logistique, le support vecteur machine linéaire, le random forest, K plus proche voisins, le Gaussian Naive Bayes, le Perceptron, le stochastic gradient descent et l'arbre de décision. Nous allons beaucoup nous inspirer de ce kernel pour notre travail. Cependant, la particularité de notre travail sera l'implémentation de l'ensemble learning qui est une technique permettant d'agréger des modèles afin d'obtenir un modèle beaucoup plus robuste.

En plus du travail de Soham, nous avons parcouru aussi d'autres kernels. Le travail de Medaxone² nous a beaucoup inspiré pour la partie traitement des données. En outre il aborde les notions de pipeline et de cross validation. Le kernel de Tom³ nous a permis d'explorer d'autres techniques de traitement de données.

Tous ces kernels ont une particularité bien donnée. Cependant nous n'avons pas pu trouver un kernel qui utilise l'ensemble learning, plus précisément le **voting classifier**, pour agréger des modèles de prédiction afin d'obtenir un modèle plus robuste.

¹ Les Kernels sont des notebooks interactives sous python.

² <https://www.kaggle.com/giseokryu/titanic-using-pipeline-gridsearchcv>

³ <https://www.kaggle.com/toshimelonhead/titanic-top-15>





b) Articles associés au Titanic

En dehors du site de [Kaggle](#), nous avons parcouru le web à la recherche d'articles en rapport avec notre problématique. D'une part, il est très important de comprendre l'histoire du Titanic afin d'être guidé dans le traitement des données. Nous avons eu recours à Wikipedia⁴ pour retracer l'histoire du Titanic. D'autre part, le travail de Niklas Donga⁵, a retenu notre attention quant à la qualité du traitement des données et à la diversité des modèles utilisés. En effet, à travers une analyse univariée, il fait ressortir des informations clés sur le jeu de données. Le feature engineering met l'accent sur l'augmentation de features par un croisement judicieux de features existants.

A partir de tous ces travaux déjà réalisés, nous allons construire une méthodologie de travail pour notre projet.

3. Méthodologie

Notre travail va se dérouler en quatre (04) étapes :

-  Récupérer les données et les charger dans notre environnement de travail,
-  Explorer les données,
-  Nettoyer les données,
-  Modéliser

Les trois premières étapes concernent le traitement et l'analyse des données. Après la modélisation, nous allons synthétiser les résultats obtenus afin de pouvoir prendre une bonne décision quant au modèle à retenir pour l'implémentation.

II. Traitement et analyse des données

Cette partie est cruciale dans le travail du data scientist. En effet, elle regroupe toutes les étapes nécessaires avant la modélisation. Elle affecte directement les résultats de la modélisation et de ce fait doit être traité avec la plus grande précaution possible.

⁴ https://fr.wikipedia.org/wiki/Naufrage_du_Titanic

⁵ <https://towardsdatascience.com/predicting-the-survival-of-titanic-passengers-30870ccc7e8>

1. Présentation des données

Sur le site de [Kaggle](https://www.kaggle.com/c/titanic/data), chaque compétition héberge les données nécessaires. Les données pour la compétition du Titanic sont alors disponibles sur **Kaggle**⁶.

Les données ont déjà été scindées en deux groupes :

- Les données d'entraînement stockées dans le fichier **train.csv**.
- Les données de test stockées dans le fichier **test.csv**.

Les données d'entraînement seront utilisées pour construire nos modèles de machine learning. Pour ces données, nous connaissons le résultat (également connu sous le nom de « **ground truth** ») pour chaque passager. Le dictionnaire de données est récapitulé dans le tableau 1 suivant :

Tableau 1: Dictionnaire de données

Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Les données de test seront utilisées pour vérifier les performances de généralisation de notre modèle sur des données inconnues lors de l'entraînement. A la différence du jeu de données d'entraînement, le jeu de test ne comporte pas de résultat pour chaque passager. D'où l'absence de la variable « **Survival** ». Notre objectif est de prédire ces résultats.

⁶ <https://www.kaggle.com/c/titanic/data>

Sur le site de la compétition, nous avons également un fichier de données **gender_submission.csv** qui regroupe un ensemble de prédictions supposant que seulement les passagers de sexe féminin ont survécus au naufrage.

2. Analyse exploratoire des données

Dans le but de pouvoir évaluer nos modèles de machine learning, nous allons prendre 25% de la base de données d'entraînement pour en faire une base de données de validation. L'analyse exploratoire ne concernera que les 75% de la base de données d'entraînement initialement hébergée dans **Kaggle**.

a) Les données manquantes

Dans notre base de données d'entraînement seules les variables **Cabin**, **Age** et **Embarked** comportent des valeurs manquantes.

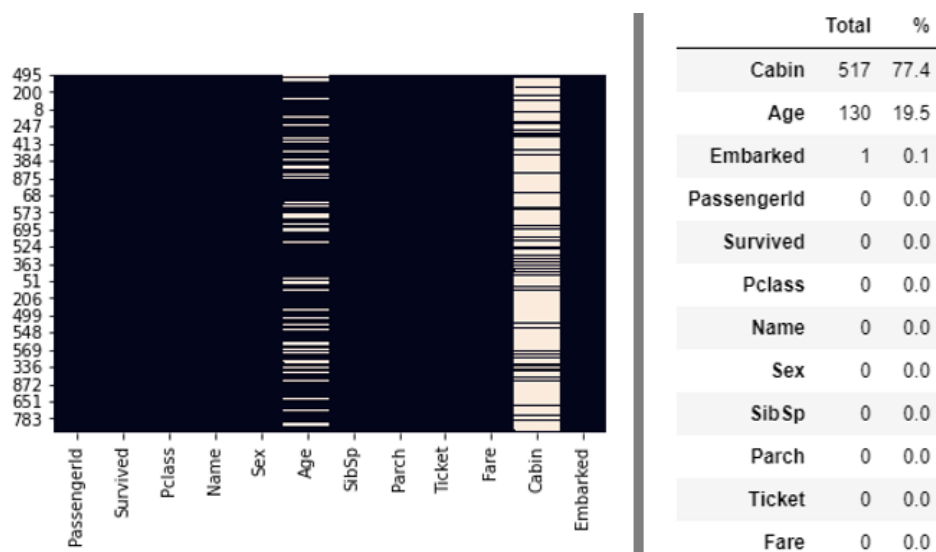


Figure 1: Données manquantes

La variable **Cabin** comporte 77.4% de données manquantes alors que la variables **Age**, respectivement **Embarked** en comporte 19.5% respectivement 0.1%.

b) Analyse uni-variée

Le naufrage du Titanic a été une des plus meurtrière dans l'histoire maritime. En effet, plus de la moitié des passagers du Titanic (62,7%) ont périé lors du naufrage. En outre il y avait plus d'hommes (64,4%) que de femmes à bord. La majorité des passagers à bord n'avait ni d'enfants (69%) ni de parents (76,5%). Les passagers avaient majoritairement

payé les tickets de la troisième classe (55,7%). Nous pouvons facilement conclure que le transport maritime n'était pas le plus utilisé par haute bourgeoisie (**Annexe 1**).

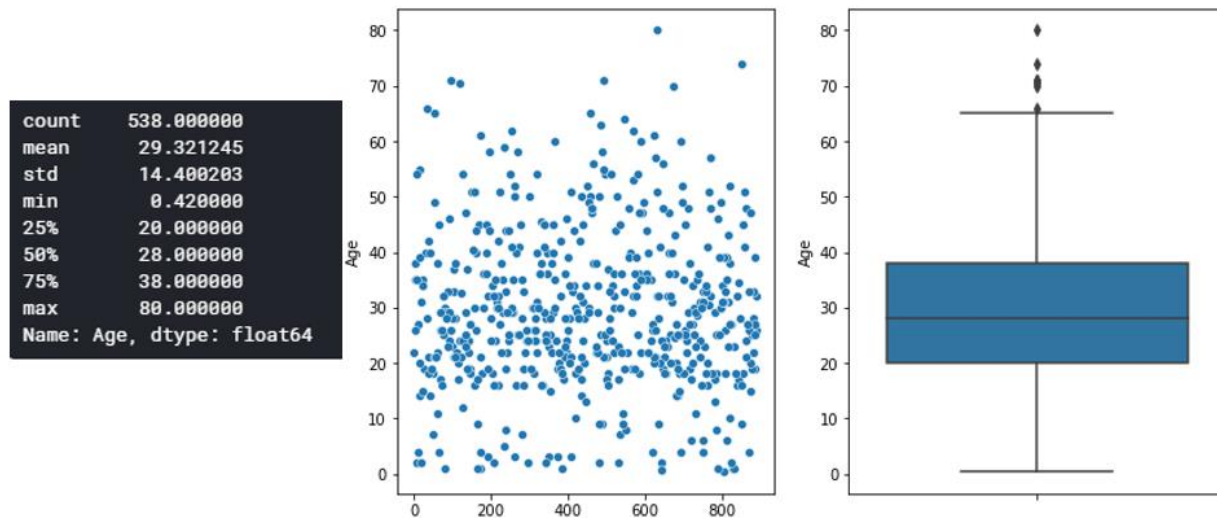


Figure 2 : Distribution de l'âge

Bien vrai que la tranche d'âge des passagers variait entre moins de 1an à 80 ans, la moyenne d'âge était de 29 ans et plus de 75% des passagers avaient moins de 40 ans (**fig2**). Les passagers du Titanic étaient de ce fait, majoritairement jeunes. Par conséquent, une telle catastrophe a impacté beaucoup de jeunes.

Le cout du billet variait entre 0 et 512.33 dollars. Plus de 75% des passagers ont payé des billets de moins de 40 dollars. On peut alors déduire que le Titanic n'était pas uniquement réservé que pour les riches (**fig3**).

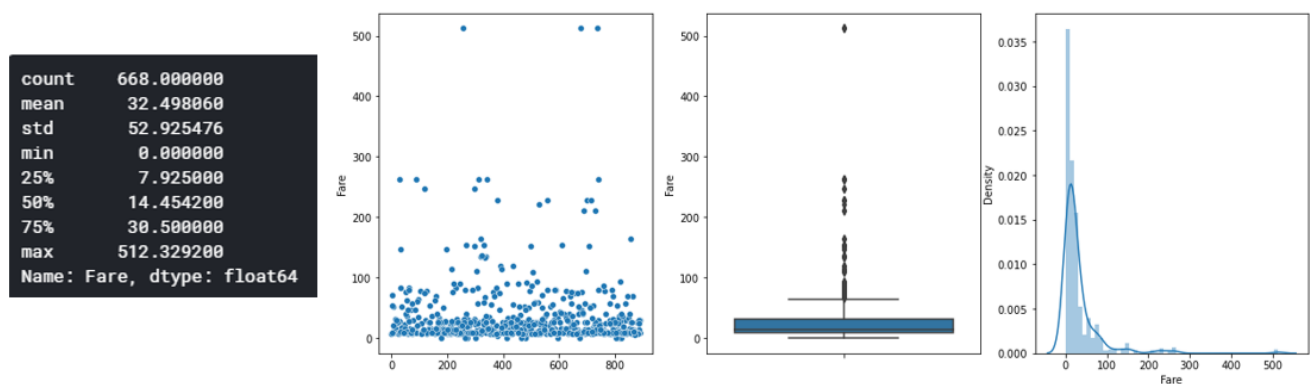


Figure 3: Distribution de la variable Fare

A partir de ces analyses, nous avons pu dégager une première caractéristique des passagers du Titanic. Afin de mieux appréhender d'autres caractéristiques, nous allons devoir effectuer une analyse multivariée.

c) Analyse multivariée

Le décès des passagers du Titanic suivant le genre est une analyse intéressante à ressortir. Le pourcentage de survie des hommes est moins de 20% alors que celle des femmes est plus de 70% (**fig4**). Nous pouvons conclure que le genre était bien déterminant dans le sauvetage des passagers lors du naufrage.

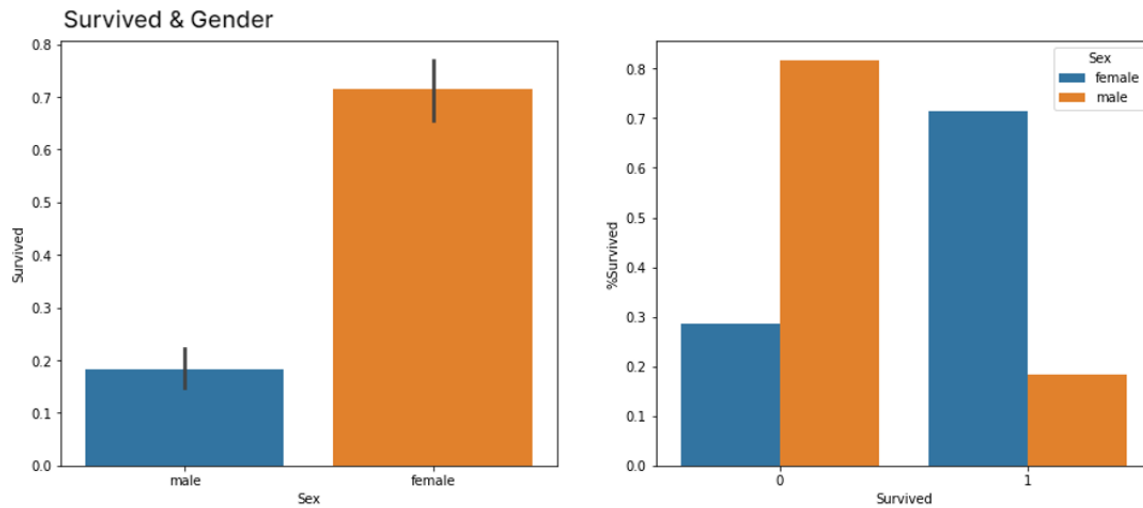


Figure 4: Analyse des décès suivant le genre

En outre, l'analyse des passagers qui ont survécu suivant la classe des tickets, nous révèle que leur proportion décroît en fonction de la classe du ticket (**fig5**). Pendant que plus de 60% des passagers de la première classe ont survécu au naufrage, moins de 30% seulement des passagers de la troisième classe y ont survécu.

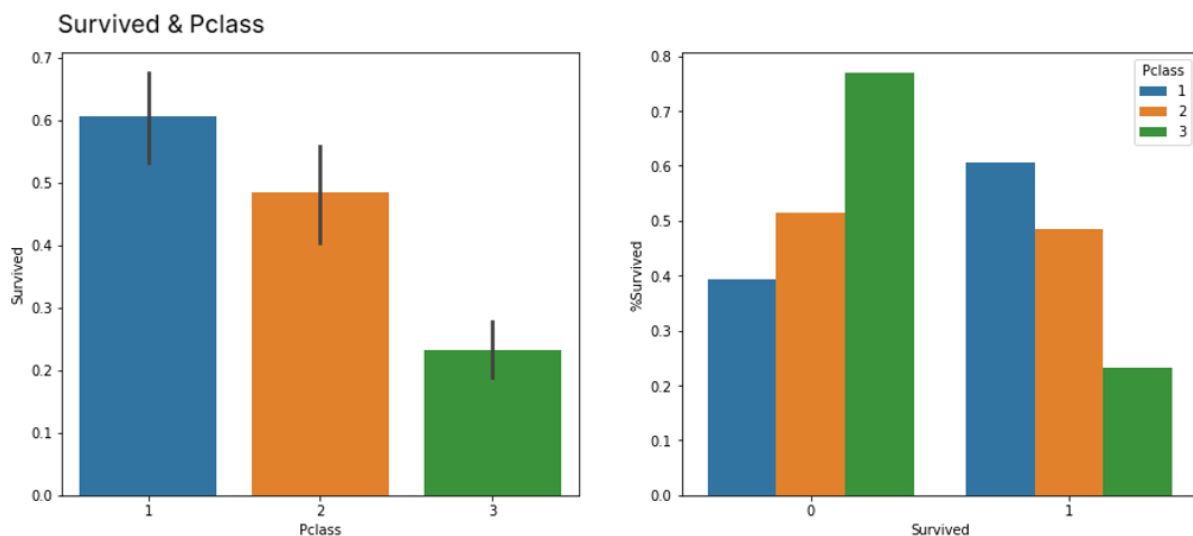


Figure 5: Analyse des décès suivant la classe du ticket

L'âge aussi a été un critère très déterminant particulièrement chez les hommes. En effet, nous voyons sur la figure 6 que les femmes de façon générale ont survécu indépendamment de l'âge alors que chez les hommes, seuls les moins âgés (< 10 ans) ont eu plus de chance de survivre.

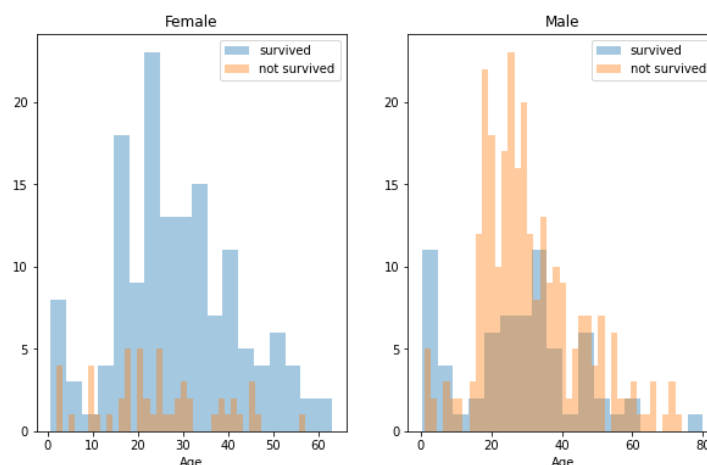


Figure 6: Analyse des décès par Age et par genre

Les passagers qui avaient plus de quatre (04) parents à bord du Titanic ont eu vraiment moins de chance de survivre au naufrage. En outre, plus un passager avait des enfants, plus il avait de forte chance de ne pas survivre (**Annexe2**).

3. Traitement des données

Après l'analyse exploratoire des données, nous devons effectuer le traitement des données afin de rendre la base de données opérationnelle pour la modélisation. En effet, la plupart des modèles de machine learning n'admettent pas des variables qualitatives. En outre, certains algorithmes sont très sensibles aux valeurs manquantes. Cette partie sera consacré au nettoyage de la base de données et aux features engineering.

a) Traitement des données manquantes

Comme nous l'avons évoqué dans la **section 2 a)** seules les variables **Cabin**, **Embarked** et **Age** comportent des données manquantes. Le traitement des données manquantes ne va concerner que les variables **Embarked** et **Age** car la variable **Cabin** sera par la suite supprimer de notre base de données finale.

Les valeurs manquantes de la variable **Embarked** seront imputées par la valeur modale de la variable. Pour la variable **Age**, nous allons imputer les valeurs manquantes par des estimations aléatoires de valeurs comprises entre la moyenne moins l'écart type et la moyenne plus l'écart type.

b) Encodage de variables

Les variables **Sex** et **Embarked** doivent être converties en variables quantitatives afin de pouvoir être considérées dans la modélisation.

En ce qui concerne la variable **Sex**, nous allons codifier tous les hommes par zéro (0) et toutes les femmes par un (01). Pour la variable **Embarked**, le port « **S** » sera codifié par zéro (0), le port « **C** » par un (01) et enfin le port « **Q** » par deux (02).

c) Catégorisation de variables continues

Elle permet de découvrir une certaine structure dans les variables continues, qui peuvent s'avérer difficiles à percevoir autrement. De plus, la variable catégorisée ainsi obtenue est plus facile à analyser et à interpréter. Cependant nous devons garder à l'esprit que cela entraîne également une perte d'information et une perte de puissance.

Dans notre étude, seules les variables **Age** et **Fare** seront catégorisées. Avant de catégoriser une variable continue, nous allons lui appliquer la transformation quantile⁷ afin d'obtenir une distribution normale. A partir de cette distribution normale, nous allons regrouper les passagers en quatre (04) groupes comme le montre la figure 7 ci-dessous.

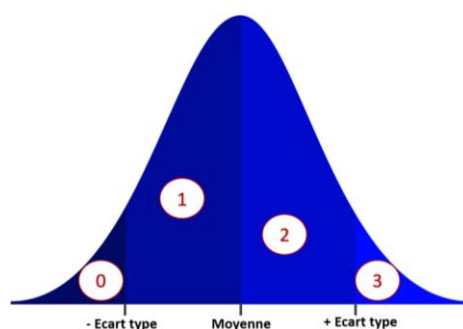


Figure 7: Principe de catégorisation des variables continues

⁷ <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html>

Les nouvelles variables obtenues seront appelées **AgeCat** pour les cas de la variable **Age** et **FareCat** pour **Fare**.

d) Création de nouvelles variables

Nous avons créé au total onze (09) variables regroupées en cinq (04) groupes :

A partir de la variable **Cabin**

Nous avons trois (03) nouvelles variables créées. La première variable appelée **InfoCabin** prend la valeur une (01) quand nous avons une information sur la cabine du passager et zéro (0) sinon. La seconde appelée **NbCabin** caractérise le nombre de cabines que possède un passager donné. Enfin, la variable **Deck** caractérise le pont du bateau occupé par le passager.

A partir de la variable **Name**

Nous allons extraire le titre des passagers à partir de leur nom. Les titres obtenus seront regroupés en cinq (05) classes : « **Mlle** », « **Miss** », « **Ms** », « **Mme** », et « **Mrs** ». Cette information sera stockée dans la variable **Title**.

A partir des variables **SibSp** et **Parch**

A partir des variables **SibSp** et **Parch**, nous allons créer la variable **FamilySize** qui caractérise la taille de la famille à bord du Titanic pour un passager donné. Cette variable est obtenue en faisant la somme des variables **SibSp** et **Parch**.

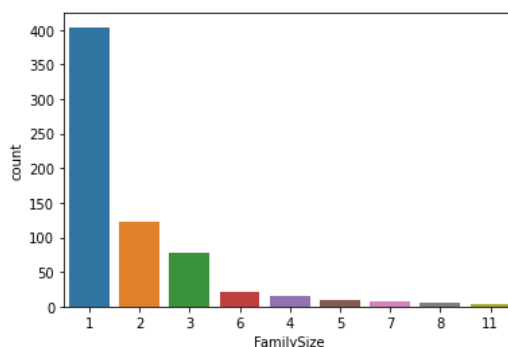


Figure 8: Diagramme de la taille de la famille à bord du Titanic

Afin d'éviter la sous représentativité de certaines modalités de la variable **FamilySize**, nous allons créer la variable **FamilySizeCat** en regroupant ensemble tous les passagers ayant plus de 4 membres de leur famille à bord.

Enfin, nous allons créer la variable **IsAlone** qui prendra la valeur zéro (0) si le passager n'a aucun membre de sa famille à bord et un (01) sinon.

Combinaison artificielle

Nous allons multiplier la variable **Age** et **Pclass** pour créer ainsi la variable **Age_Class**. La variable **Fare_per_Person** sera créée en divisant la variable **Fare** par la taille de la famille à bord.

e) Suppression de variables

Après avoir créé les nouvelles variables, nous allons maintenant supprimer toutes les variables qualitatives de notre base de données. Elles sont au nombre de trois (03) : **Name**, **Ticket**, et **Cabin**. En outre, la variable **PassengerId** ne contribue pas à apporter une information sur le décès ou non d'un passager. Par conséquent, nous allons aussi la supprimer de notre dataset.

Après la phase de traitement de données, nous avons une base de données prête à la modélisation.

III. Modélisation

Nous sommes maintenant prêts à former un modèle et à prédire la solution requise. Il existe une multitude d'algorithmes de modélisation prédictifs parmi lesquels choisir. Dans cette partie, nous allons présenter les algorithmes que nous avons retenus pour notre étude.

1. Modèle Baseline

Notre modèle baseline sera le modèle qui prédit que seulement les femmes du Titanic ont survécu au naufrage. Sur le site de **Kaggle**, les prédictions du modèle sur la base de données test sont stockées dans le fichier **gender_submission.csv**.

2. Modèles implémentés

Nous avons retenu huit (08) modèles que nous entraînerons sur notre jeu de données afin de comparer leurs performances entre eux. Deux cas d'implémentation seront abordés dans notre étude. Le premier cas consiste à entraîner les modèles avec les valeurs par défaut des paramètres de chaque modèle sur une base de données d'entraînement et utiliser une base de données de validation pour capter les performances en termes de généralisation des modèles. Nous pourrons alors comparer les performances des modèles. Le deuxième cas consiste à tuner les hyper-paramètres afin de les optimiser et obtenir ainsi de meilleures performances. Nous utiliserons la méthode **GridSearchCV** de la classe **model_selection** du module **scikit learn** de python afin d'implémenter ce cas.

a) Stochastic Gradient Descent (SGD)

Le modèle SGD (**SGDClassifier**) est implémenté dans la classe **linear_model** du module **scikit learn**. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées sur la figure 9

```
class sklearn.linear_model.SGDClassifier(loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal',
eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False,
average=False)
```

[\[source\]](#)

Figure 9: Modèle SGDClassifier

Les hyper-paramètres que nous avons choisi de tuner sont : **alpha**, **loss**, **tol** et **penalty** (**fig10**).

```
param_grid = {'alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1e0, 1e1, 1e2, 1e3], # learning rate
              'tol': [1e-4, 1e-3, 1e-2, 1e-1],
              'loss': ['hinge', 'log', 'modified_huber', 'squared_hinge', 'perceptron'], # logistic regression,
              'penalty': ['l2', 'l1', 'elasticnet'],
              'random_state': [1]}
```

Figure 10: Hyper paramètres à tuner du modèle SGDClassifier

b) Random Forest

Le random forest (**RandomForestClassifier**) est implémenté dans la classe **ensemble** du module **scikit learn**. L'ensemble de ses hyper-paramètres et leurs valeurs par défaut sont décrits sur la figure 11.

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
```

[\[source\]](#)

Figure 11: Modèle random forest

Les hyper-paramètres que nous avons choisi de tuner sont : **criterion**, **n_estimators**, **max_depth** et **min_samples_split** (fig12).

```
param_grid = { "criterion": ["gini", "entropy"],
               "n_estimators": [i for i in range(50, 250, 50)],
               "max_depth": [i for i in range(5, 25, 2)],
               "min_samples_split": [12, 16, 18, 25, 35],
               "oob_score": [True],
               "random_state": [1],
               "n_jobs": [-1]}
```

Figure 12: Hyper paramètres à tuner du modèle random forest

c) La régression logistique

La régression logistique (**LogisticRegression**) est implémentée dans la classe **linear_model** du module **scikit learn**. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées sur la figure 13.

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)
```

[\[source\]](#)

Figure 13: Modèle logistique

Les hyper-paramètres que nous avons choisis de tuner sont : **tol**, **C**, **penalty**, **solver** et **max_iter** (fig14).

```
param_grid={ 'tol': [1e-4, 1e-3, 1e-2, 1e-1],
             "C": [i for i in range(7, 20, 2)],
             "penalty": ["l1", "l2"], # l1 lasso l2 ridge
             "solver": ['lbfgs', 'liblinear'],
             "max_iter": [i for i in range(100, 1000, 100)],
             "random_state": [1]
           }
```

Figure 14: Hyper paramètres à tuner du modèle logistique

d) K plus proche voisin

L'algorithme des K plus proche voisin (***KNeighborsClassifier***) est implémenté dans la classe ***neighbors*** du module ***scikit learn***. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées sur la figure 15.

```
class sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2,
metric='minkowski', metric_params=None, n_jobs=None, **kwargs)
```

[\[source\]](#)

Figure 15: Modèle *KNeighborsClassifier*

Les hyper-paramètres que nous avons choisis de tuner sont : ***weights*** et ***n_neighbors*** (**fig16**).

```
param_grid = {
    "weights": ["uniform", "distance"],
    'n_neighbors': [3, 5, 7, 9, 11, 13, 15] # usually odd numbers
}
```

Figure 16: Hyper paramètres à tuner du modèle *KNeighborsClassifier*

e) Perceptron

Le perceptron (***Perceptron***) est implémenté dans la classe ***linear_model*** du module ***scikit learn***. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées dans la figure 17.

```
class sklearn.linear_model.Perceptron(*, penalty=None, alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
tol=0.001, shuffle=True, verbose=0, eta0=1.0, n_jobs=None, random_state=0, early_stopping=False, validation_fraction=0.1,
n_iter_no_change=5, class_weight=None, warm_start=False)
```

[\[source\]](#)

Figure 17: Modèle perceptron

Les hyper-paramètres que nous avons choisis de tuner sont : ***eta0*** et ***max_iter*** (**fig18**)

```
param_grid = {'eta0': [1e-4, 1e-3, 1e-2, 1e-1, 1.0, 10],
    'max_iter': [i for i in range(500, 2000, 500)],
    'random_state': [1]
}
```

Figure 18: Hyper paramètres à tuner du modèle perceptron

f) Support vecteur machine linéaire

Le support vecteur machine linéaire (**LinearSVC**) est implémenté dans la classe **svm** du module **scikit learn**. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées sur la figure 19

```
class sklearn.svm.LinearSVC(penalty='l2', loss='squared_hinge', *, dual=True, tol=0.0001, C=1.0, multi_class='ovr',
fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0, random_state=None, max_iter=1000)
```

[\[source\]](#)

Figure 19: Modèle support vecteur machine linéaire

Les hyper-paramètres que nous avons choisis de tuner sont : **C**, **loss** et **tol** (fig20).

```
param_grid = {'C': np.arange(0.01, 100, 10),
              'loss': ['hinge', 'squared_hinge'],
              'tol': [1e-4, 1e-3, 1e-2, 1e-1],
              'random_state': [1]}
}
```

Figure 20: Hyper paramètres à tuner du modèle LinearSVC

g) Decision Tree

Le modèle decision tree (**DecisionTreeClassifier**) est implémenté dans la classe **tree** du module **scikit learn**. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées sur la figure 21

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, ccp_alpha=0.0)
```

[\[source\]](#)

Figure 21: Modèle decision tree

Les hyper-paramètres que nous avons choisis de tuner sont : **criterion**, **max_depth**, **max_leaf_nodes** et **min_samples_split** (fig22).

```
param_grid = {'criterion': ['gini', 'entropy'],
              'max_depth': [i for i in range(7, 20, 2)],
              'max_leaf_nodes': list(range(2, 100)),
              'min_samples_split': [2, 3, 4]}
```

Figure 22: Hyper paramètres à tuner du modèle DecisionTreeClassifier

h) Voting classifier

Le modèle voting classifier (**VotingClassifier**) est implémenté dans la classe **ensemble** du module **scikit learn**. Ses hyper-paramètres ainsi que leurs valeurs par défaut sont présentées dans la figure 23

```
class sklearn.ensemble.VotingClassifier(estimators, *, voting='hard', weights=None, n_jobs=None, flatten_transform=True, verbose=False)
```

[\[source\]](#)

Figure 23: Modèle VotingClassifier

Le paramètre « **estimators** » prendra la liste des sept (07) classificateurs que nous avons déjà citée ci-dessus chacun implémenté avec ses meilleurs paramètres après tuning (**fig24**).

```
sgd = SGDClassifier(alpha = 0.001, loss = 'log', penalty = 'l2', tol = 0.0001, random_state = 1)
rf = RandomForestClassifier(n_estimators = 50, criterion = 'gini', min_samples_split = 12, oob_score = True, random_state = 1, n_jobs = -1)
logreg = LogisticRegression(C = 9.0, penalty = 'l2', max_iter = 100, solver = 'lbfgs', tol = 0.1, random_state = 1)
knn = KNeighborsClassifier(n_neighbors = 5, weights = 'uniform')
perceptron = Perceptron(eta0 = 0.0001, max_iter = 500, random_state = 1)
linear_svc = LinearSVC(C = 60.01, loss = 'squared_hinge', random_state = 1, tol = 0.0001)
decision_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 19, max_leaf_nodes = 62, min_samples_split = 2)

voting_clf = VotingClassifier(estimators=[('SGD', sgd), ('RF', random_forest), ('LR', logreg), ('KNN', knn), ('NaiveBayes', gaussian), ('Perceptron',
('SVC', linear_svc), ('DT', decision_tree)], voting='hard')
```

Figure 24: Implémentation du modèle VotingClassifier

IV. Présentation des résultats

1. Métriques

Les métriques nous permettent de mesurer la performance des modèles et de les comparer ainsi entre eux.

La métrique que nous choisirons d'utiliser afin de mesurer la performance de nos modèles sera le score **f1_macro**.

Le score F1 peut être interprété comme une moyenne pondérée de la **précision** et du **rappel**, et il atteint sa meilleure valeur à 1 et son pire score à 0. La contribution relative de la **précision** et du **rappel** au score F1 est égale. La formule pour le score F1 est la suivante :

$$F1 = 2 * (precision * recall) / (precision + recall)$$

Figure 25: formule de la f1 score

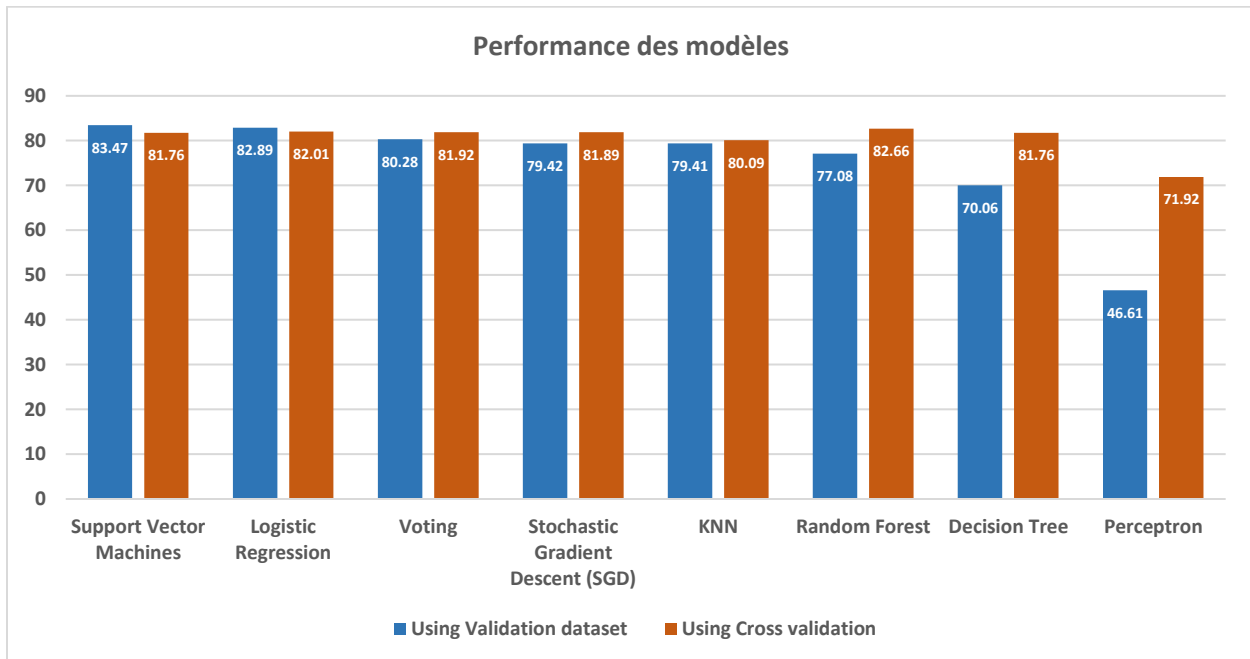
2. Synthèse des résultats de la modélisation

Les valeurs optimales des hyper-paramètres pour les différents modèles sont récapitulées dans sur la figure 26.

```
sgd = SGDClassifier(alpha = 0.001, loss = 'log', penalty = 'l2', tol = 0.0001, random_state = 1)
rf = RandomForestClassifier(n_estimators = 50, criterion = 'gini', min_samples_split = 12, oob_score = True, random_state = 1, n_jobs = -1)
logreg = LogisticRegression(C = 9.0, penalty = 'l2', max_iter = 100, solver = 'lbfgs', tol = 0.1, random_state = 1)
knn = KNeighborsClassifier(n_neighbors = 5, weights = 'uniform')
perceptron = Perceptron(eta0 = 0.0001, max_iter = 500, random_state = 1)
linear_svc = LinearSVC(C = 60.01, loss = 'squared_hinge', random_state = 1, tol = 0.0001)
decision_tree = DecisionTreeClassifier(criterion = 'entropy', max_depth = 19, max_leaf_nodes = 62, min_samples_split = 2)
```

Figure 26: Meilleures valeurs des hyper paramètres des modèles

Les performances obtenues après entraînement ont été représentées dans la figure ci-dessous (**fig27**).



La moyenne des performances des modèles dans le cas de l'utilisation de la base de données de validation est de 74,90% ($\pm 11,38\%$). Celle dans le cas de l'utilisation de la cross validation est de 80.50% ($\pm 3,31\%$). Nous pouvons déduire que l'utilisation de la base de données de validation a généralement sous-estimé les performances de nos modèles.

Nous pouvons alors nous intéresser aux résultats de la cross-validation et choisir le **voting classifieur** comme modèle à déployer car ses performances tournent autour de la moyenne des performances de tous les modèles.

Notre soumission au site de compétition **Kaggle** a obtenu un score de 78,23% avec un classement de 4 918 sur 23 592 dans la compétition ; alors que le modèle baseline donnait un score de 77,51%. Ce résultat est indicatif pendant le déroulement de la compétition. Il représente aussi le meilleur résultat que nous avons obtenu sur les huit (08) soumissions dans **Kaggle**.

V. Conclusion

Le site de **Kaggle** nous a permis de tester nos compétences en tant qu'ingénieur machine learning dans la compétition « **Titanic – Machine Learning from Disaster** ». Nous avons commencé par l'exploration des données, qui nous a permis de nous faire une idée de l'ensemble des données. Pendant ce processus, nous avons utilisé **seaborn** et **matplotlib** pour faire les visualisations. Pendant la partie de prétraitement des données, nous avons imputer les valeurs manquantes, converti les caractéristiques en valeurs numériques, regroupé les valeurs en catégories et créé de nouvelles caractéristiques. Ensuite, nous avons entraîné 8 modèles de machine learning différents et choisi la prédiction de l'un d'entre eux pour soumettre dans **Kaggle**.

Il excite des perspectives d'amélioration du résultat que nous avons obtenu. Par exemple, nous pouvons effectuer une **feature engineering** plus poussée, effectuer un réglage plus approfondi des hyper-paramètres sur les modèles de machine learning ou prospector d'autres modèles de classification de machine learning.

Sources bibliographiques :

1. A Journey through Titanic <https://www.kaggle.com/omarelgabry/a-journey-through-titanic>
2. Getting Started with Pandas: Kaggle's Titanic Competition <https://www.kaggle.com/c/titanic>
3. Titanic best working Classifier <https://www.kaggle.com/sinakhorami/titanic-best-working-classifier>
4. Niklas Donges May14, 2018 : Predicting the Survival of Titanic Passengers <https://towardsdatascience.com/predicting-the-survival-of-titanic-passengers-30870ccc7e8>
5. Titanic <https://fr.wikipedia.org/wiki/Titanic>

ANNEXES

Annexe 1 : Analyse univariée

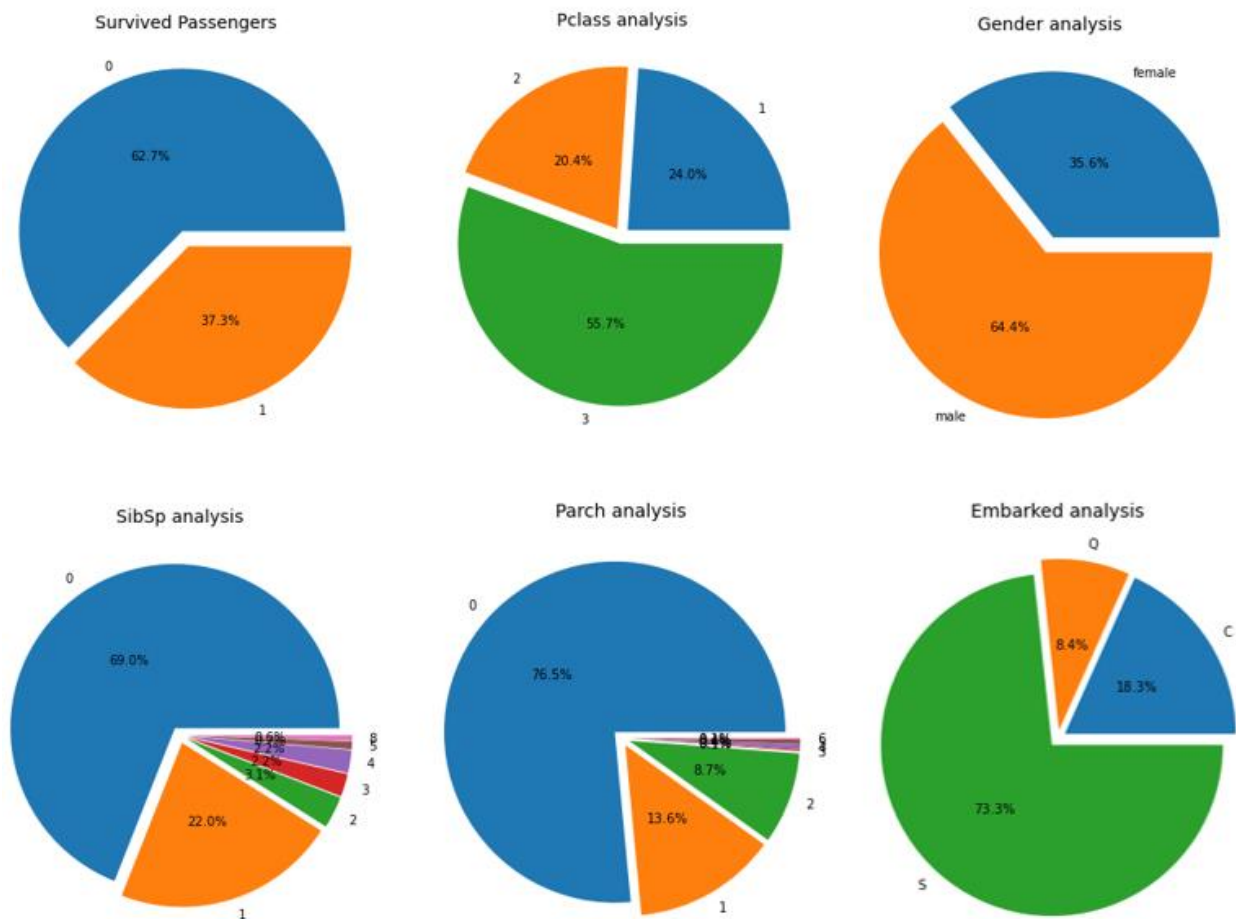


Figure 27: Diagramme des variables catégorielles

Annexe 2 : Analyse multivariée

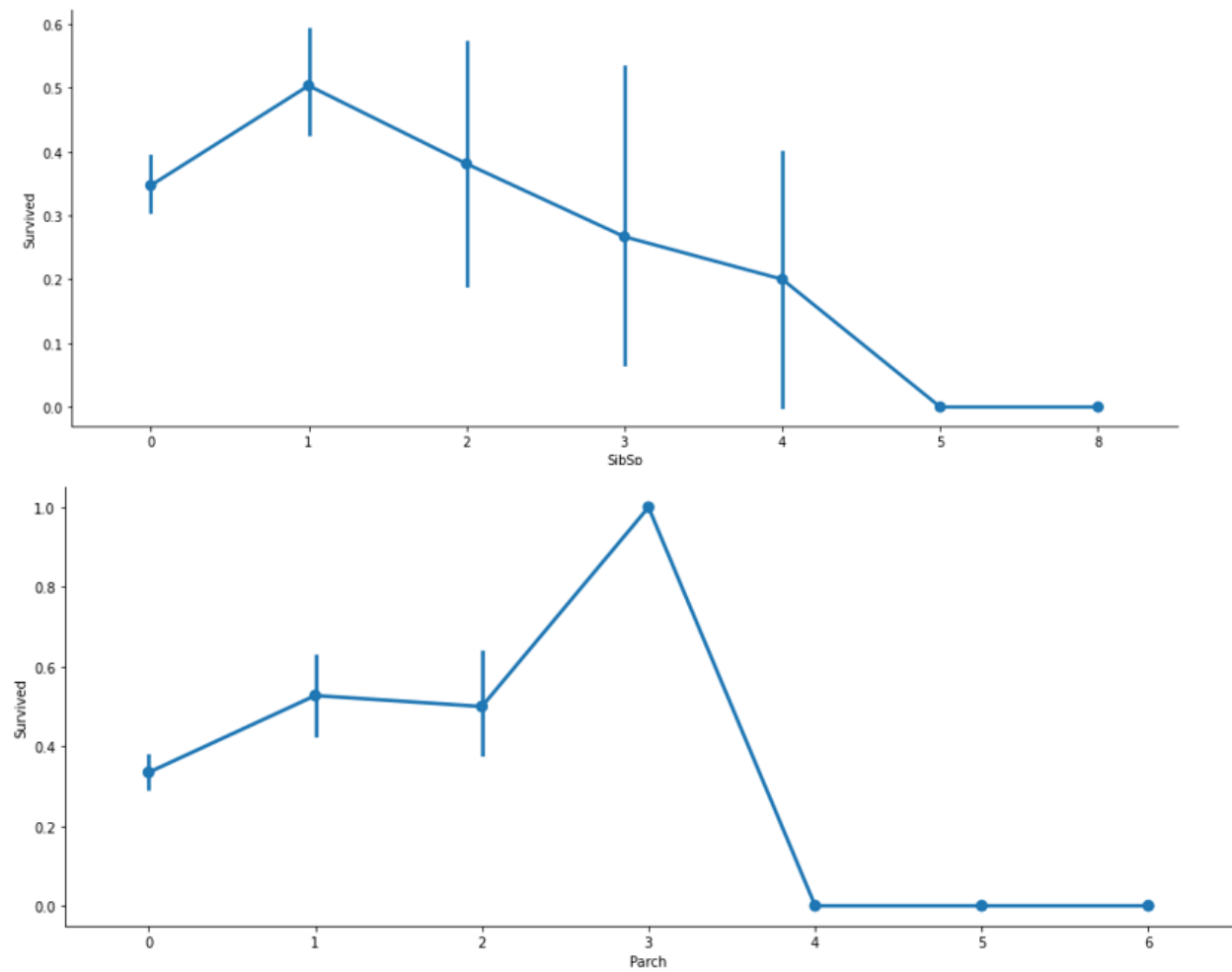


Figure 28: Proportion de Survived par rapport à Sibsp et Parch