

## Rapport de Projet

---

# Projet Titanic Machine Learning

---



**Mhalla Chadha**

M1 Big Data

Promo 2020

*Prof :* **Bouchhima Bochra**

20 Juin 2020

# Table des matières

0.1	Introduction générale : . . . . .	2
0.2	Compréhension les données : . . . . .	3
0.3	Exploration et analyse des données : . . . . .	4
0.3.1	Combien de personne a survécu ? : . . . . .	4
0.3.2	Le sex est-il un facteur sur la survie des passagers? . . . . .	5
0.3.3	La Pclasse est elle un facteur sur la survie des passagers? . . . . .	6
0.3.4	Relation entre Pclass, sex et survived : . . . . .	7
0.3.5	Embarked est elle un facteur sur la survie? . . . . .	8
0.3.6	Relation entre Pcalss,sex,Embarked et survived . . . . .	8
0.3.7	Parch est il un facteur sur la survie? . . . . .	9
0.3.8	SibSp est il un facteur sur la survie? . . . . .	10
0.3.9	Age est il un facteur sur la survie des passagers? . . . . .	10
0.3.10	Relation entre Age, sex et survie : . . . . .	11
0.4	Missing Data et manipulation des données : . . . . .	12
0.4.1	Etude des descripteurs : . . . . .	12
0.5	Prédiction de l'age : . . . . .	16
0.6	Prediction de la survie : . . . . .	17
0.6.1	Avec l'age prédit : . . . . .	17
0.6.2	Classement des meilleures classificateurs pour la prédiction : . . . . .	22
0.6.3	Importance des descripteurs : . . . . .	22
0.6.4	Avec la médiane de l'age : . . . . .	23
0.6.5	Classement des classificateurs pour la deuxieme méthode de prédiction : . . . . .	27
0.7	Evaluation approfondie : . . . . .	27
0.7.1	Precision & Recall : . . . . .	27
0.7.2	ROC AUC : . . . . .	27
0.7.3	Score ROC AUC : . . . . .	28
0.8	Soumettre le résultat sur Kaggle! . . . . .	28
0.9	Testons si vous allez survivre ou non ? :D . . . . .	28
0.10	Résumé : . . . . .	29

## 0.1 Introduction générale :

Le RMS Titanic était connu comme le navire insubmersible, le plus luxueux et le plus grand navire à passagers de son temps.

Malheureusement, ce paquebot a coulé le 15 Avril 1912 , il y avait environ 2 224 passagers et membres d'équipage à bord du navire, et plus de 1 500 sont morts, ce qui en fait l'une des catastrophes maritimes commerciales en temps de paix les plus meurtrières de l'histoire moderne.

Dans ce rapport nous analysons l'ensemble des données Titanic et faisons deux prédictions la première s'agit de pouvoir détecter si un passager X ayant les caractéristiques (PassengerId, Age, Pclasse, Sexe, Survived...) aurait survécu ou non et la deuxième c'est pour voir si nous aurions survécu.



## 0.2 Compréhension les données :

Dans la Machine Learning, les données sont principalement divisées en deux parties :train et test. Les données de formation sont destinées à la formation de notre algorithme et les données de test visent à vérifier la performance de notre algorithme. Pour avoir l'accès aux différentes données et afin de charger et effectuer une manipulation très basique des données, j'utilise Pandas , une bibliothèque de manipulation de données en python.

Nos dataset sont sous cette forme :

```
In [1024]: print(Data)
PassengerId  Survived  Pclass  ...  Fare  Cabin  Embarked
0            1         0       3  ...   7.2500   NaN        S
1            2         1       1  ...  71.2833   C85        C
2            3         1       3  ...   7.9250   NaN        S
3            4         1       1  ...  53.1000  C123        S
4            5         0       3  ...   8.0500   NaN        S
..          ...      ...    ...  ...   ...    ...      ...
886          887         0       2  ...  13.0000   NaN        S
887          888         1       1  ...  30.0000  B42        S
888          889         0       3  ...  23.4500   NaN        S
889          890         1       1  ...  30.0000  C148        C
890          891         0       3  ...   7.7500   NaN        Q

[891 rows x 12 columns]
```

```
In [1027]: print(test)
PassengerId  Pclass  ...  Cabin  Embarked
0           892       3  ...   NaN        Q
1           893       3  ...   NaN        S
2           894       2  ...   NaN        Q
3           895       3  ...   NaN        S
4           896       3  ...   NaN        S
..          ...    ...  ...   ...    ...
413         1305       3  ...   NaN        S
414         1306       1  ...  C105        C
415         1307       3  ...   NaN        S
416         1308       3  ...   NaN        S
417         1309       3  ...   NaN        C

[418 rows x 11 columns]
```

```
In [1029]: Data.shape #retourne le nombre des lignes et le nombre des colonnes
Out[1029]: (891, 12)
```

```
In [1030]: test.shape #retourne le nombre des lignes et le nombre des colonnes
Out[1030]: (418, 11)
```

Ainsi, nous avons ici un total de 891 lignes pour le "train" et 417 lignes pour le "test".

La signification de chaque colonne est :

PassengerId : int : Id

Survived : int : Survival (0 = No ; 1 = Yes)

Pclass : int : Passenger Class

Name : object : Name

Sex : object : Sex

Age : float : Age

SibSp : int : Nombre de frères et sœurs / conjoints à bord

Parch : int : Nombre de parents / enfants à bord

Billet : objet : Numéro de billet

Tarif : float : Tarif passager

Cabine : objet : Cabine

embarquée : objet : Port d'embarquement (C = Cherbourg; Q = Queenstown; S = Southampton)

```
In [1028]: Data.describe(include='all') #retourne les différentes valeurs (means..)
```

```
Out[1028]:
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
count	891.000000	891.000000	891.000000	...	891.000000	204	889
unique	NaN	NaN	NaN	...	NaN	147	3
top	NaN	NaN	NaN	...	NaN	G6	S
freq	NaN	NaN	NaN	...	NaN	4	644
mean	446.000000	0.383838	2.308642	...	32.204208	NaN	NaN
std	257.353842	0.486592	0.836071	...	49.693429	NaN	NaN
min	1.000000	0.000000	1.000000	...	0.000000	NaN	NaN
25%	223.500000	0.000000	2.000000	...	7.910400	NaN	NaN
50%	446.000000	0.000000	3.000000	...	14.454200	NaN	NaN
75%	668.500000	1.000000	3.000000	...	31.000000	NaN	NaN
max	891.000000	1.000000	3.000000	...	512.329200	NaN	NaN

## 0.3 Exploration et analyse des données :

### 0.3.1 Combien de personne a survécu ? :



Nous pouvons voir ici que malheureusement 38% seulement des passagers ont survécu.

#### **Soyons plus précis sur les informations des personnes qui ont survécu !**

Maintenant que nous avons chargé nos données et compris à quoi elles ressemblent, nous allons passer à mesurer l'impact de chaque caractéristique sur notre production, c'est-à-dire si un passager a survécu ou non.

Tous les types de données ne vont pas forcément être pertinents, le nom du passager par exemple n'a aucune importance sur le fait qu'il ait survécu ou non. C'est pour cela nous allons analyser par la suite les données qui peuvent être un facteur sur la survie du passager.

### 0.3.2 Le sex est-il un facteur sur la survie des passagers ?

Après avoir eu le nombre total des personnes survivantes et décédées, j'ai cherché ensuite la relation entre le sex et la survie des passagers, pour trouver enfin le résultat suivant

```
Sex      Survived
female  1         233
        0         81
male    0         468
        1         109
Name: Survived, dtype: int64
```

Autrement dit, le nombre des femmes qui ont survécu est 233 alors que les hommes c'est 109. Par contre les femmes qui n'ont pas survécu sont 81 et les hommes c'est 468.

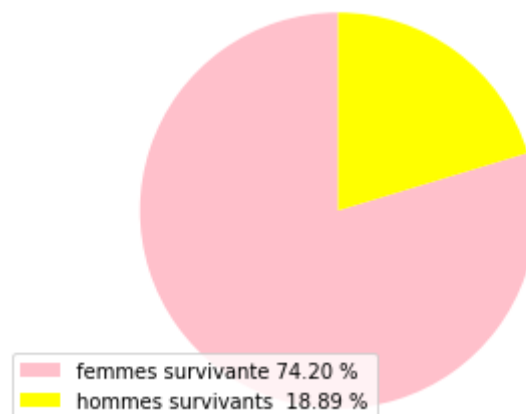
**Il est bien clair donc que le nombre femmes survivantes est très élevé par rapport au nombre des hommes.**

Si on passe au calcul des pourcentages de la survie des deux sex, on retrouve :

```
In [1038]: print('Pourcentage des femmes qui ont survécu: ',float((Data[Data.Sex ==
'female'].Survived.sum()/(Data[Data.Sex == 'female']
...:           .Survived.count()*100.0), '%')) #pourcentage des femmes qui ont survécu
par rapport au nombre total des femmes
Pourcentage des femmes qui ont survécu: 74.20382165605095 %

In [1039]: print('Pourcentage des hommes qui ont survécu: ',float((Data[Data.Sex ==
'male'].Survived.sum()/(Data[Data.Sex == 'male']
...:           .Survived.count()*100.0), '%'))#pourcentage des hommes qui ont survécu
par rapport au nombre total des hommes
Pourcentage des hommes qui ont survécu: 18.890814558058924 %
```

Pourcentage de la survie des femmes&hommes



Le sexe est important et directement proportionnel au taux de survie. Les femmes et les enfants ont été sauvés en premier lors de cette tragédie. Nous pouvons voir que 74% de toutes les femmes ont été sauvées et seulement 18% de tous les hommes ont été sauvés. Encore une fois, cela aura un impact sur nos résultats.

### 0.3.3 La Pclasse est elle un facteur sur la survie des passagers ?

```

In [1040]: Data.Pclass.value_counts()
Out[1040]:
3    491
1    216
2    184
Name: Pclass, dtype: int64

In [1041]: w=Data.groupby('Pclass').Survived.value_counts()

In [1042]: print(w)
Pclass  Survived
1        1      136
         0       80
2        0       97
         1       87
3        0      372
         1      119
Name: Survived, dtype: int64

```

On a pour la classe 1 136 passagers qui sont resté en vie et 80 sont morts, pour la classe 2 on trouve 97 passagers morts et 87 en vie par contre la classe 3 on a 372 morts et 119 ont survécu.

Pourcentage des survivants de chaque classe

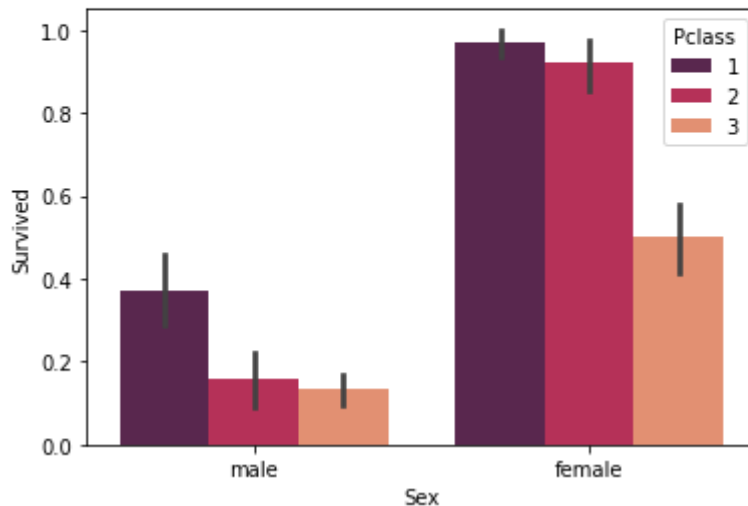


Il est évident que la classe de passagers est directement proportionnelle au taux de survie. Si l'importance d'une personne est plus importante que d'autres, elle sortira d'abord du désastre. Et nos données racontent la même histoire. 63% des personnes ont survécu à la classe 1. Par conséquent, cette fonctionnalité a vraiment un impact.

### 0.3.4 Relation entre Pclass, sex et survived :

Sex	female	male
Pclass		
1	94	122
2	76	108
3	144	347

```
In [1048]: sb.barplot('Sex', 'Survived', hue='Pclass',palette="rocket",data=Data)
Out[1048]: <matplotlib.axes._subplots.AxesSubplot at 0x238a365ffc8>
```



On retrouve ici que pour la classe1 on a 94 femmes et 122 hommes, pour la classe2, on a 76 femmes et 108 hommes et pour la classe 3 on a 144 femmes et 347 hommes.

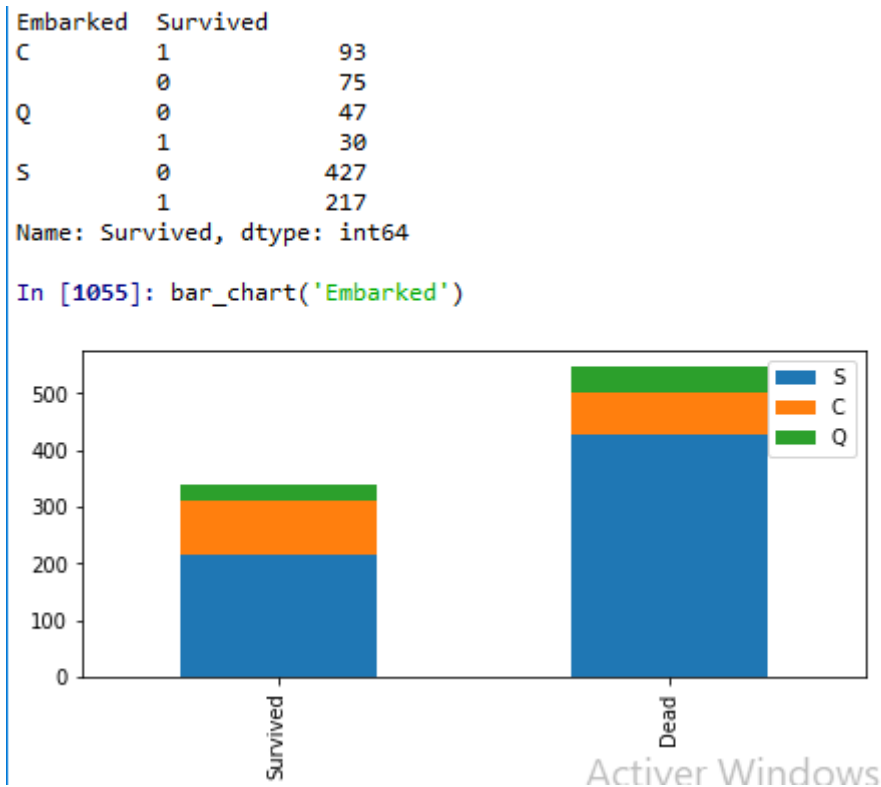
Le plot ci-dessus nous montre le taux de survie de chaque sex selon sa classe.

Il est bien clair que le taux de la survie des femmes dans les trois classes est beaucoup plus élevé que celui des hommes. De meme le décalage entre les 3 classes selon les deux catégories de sex est aussi clair, la classe 1 est la plus élevée chez les femmes et les hommes.

Ceci prouve que les femmes de la première classe ont plus de chance de survivre et d'une façon moins importante les femmes de la classe 2 et d'une façon beaucoup moins importante celles de la classe3. Puis on retrouve les hommes de la première classe et d'une manière moins importante ceux de la 2ème classe et finalement les hommes de la 3ème classe sont est les plus menacés et ont moins de chance pour survivre.

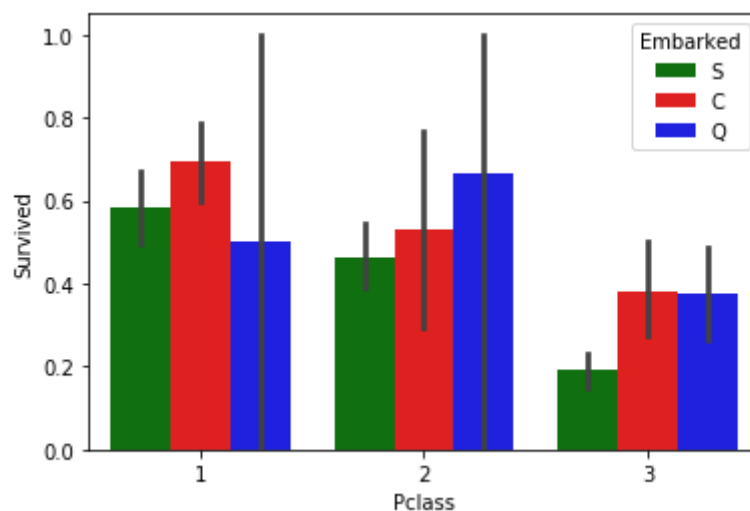


### 0.3.5 Embarked est elle un facteur sur la survie ?

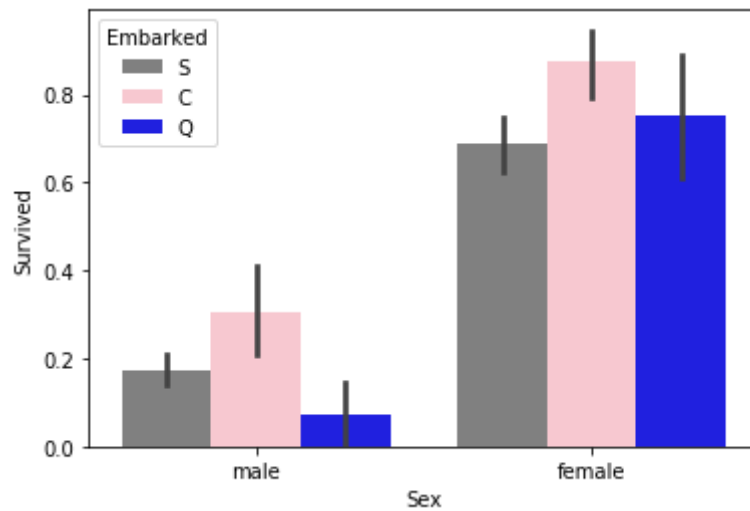


L'embarked 'S' a le nombre le plus élevé des passagers qui n'ont pas survécu, l'écart est bien clair entre les 3 types d'embarked.

### 0.3.6 Relation entre Pclass,sex,Embarked et survived



Selon la survie des passagers, pour la classe 1 on a le type "C" a plus de chance à survivre , pour la classe 2 le type "Q" a plus de chance à survivre et pour la classe3, les deux types "C" et "Q" ont la meme chance et d'une façon moins importante le type " S"

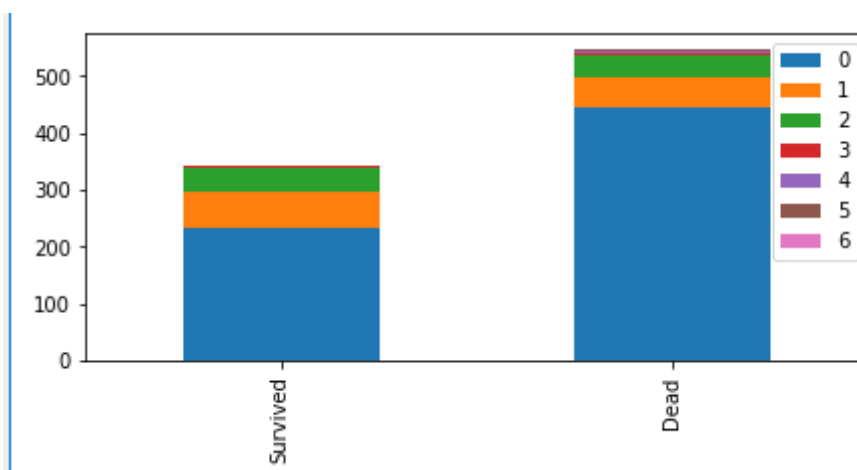


Dans ce plot, on remarque que les femmes de type embarked 'C' ont plus de chance de survivre par rapport aux femmes ayant embarked 'S' et 'Q'. Le meme principe s'applique pour les hommes, tout en sachant que les femmes de type embarked C ont beaucoup plus de chance de survivre que les hommes de type C aussi.

**La conclusion qu'on peut obtenir ici, c'est que le type embarked "C" contient le nombre le plus élevé des passagers qui ont la chance de survivre.**

### 0.3.7 Parch est il un facteur sur la survie ?

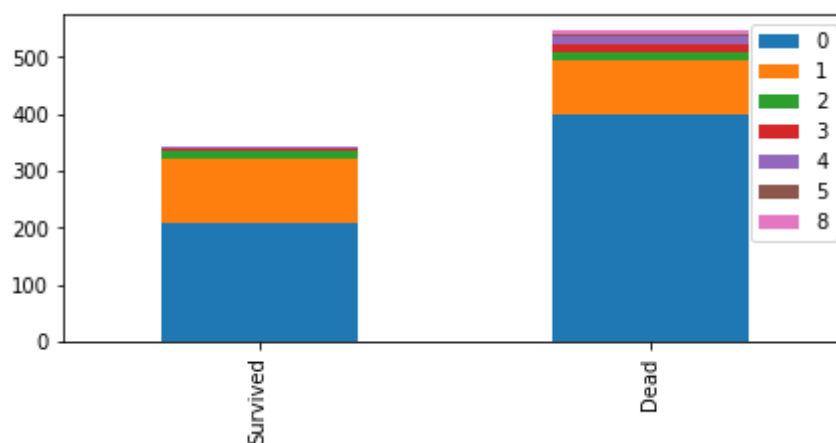
```
Parch  Survived
0      0         445
      1         233
1      1          65
      0          53
2      0          40
      1          40
3      1           3
      0           2
4      0           4
5      0           4
      1           1
6      0           1
Name: Survived, dtype: int64
```



Le parch ayant la valeur 0 occupe le nombre le plus élevé des passagers dans le cas de survie ou décès, mais le nombre de décès est plus grand que celui de la survie par contre Parch =1 on trouve que le nombre de survie est plus grand que le nombre des décès. On peut dire donc que les passagers ayant les critère de Parch=1 ont plus de chance de survivre que les autres.

### 0.3.8 SibSp est il un facteur sur la survie ?

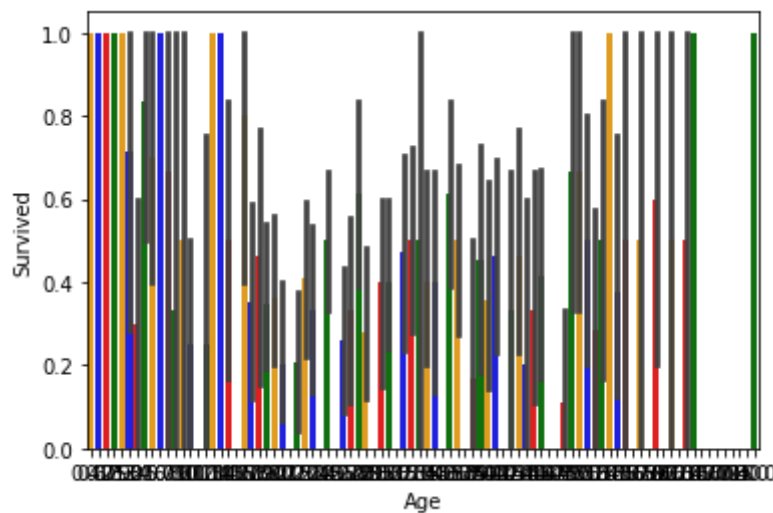
```
SibSp  Survived
0      0      398
      1      210
1      1      112
      0       97
2      0       15
      1       13
3      0       12
      1        4
4      0       15
      1        3
5      0        5
8      0        7
Name: Survived, dtype: int64
```



Les résultats ci dessus montrent que les passagers ayant 1 seul SibSp et d'une façon moins importantes ceux qui ont 2 SibSp, ont plus de chances de survivre que les autres.

### 0.3.9 Age est il un facteur sur la survie des passagers ?

```
Age      Survived
0.42     1         1
0.67     1         1
0.75     1         2
0.83     1         2
0.92     1         1
..
70.00    0         2
70.50    0         1
71.00    0         2
74.00    0         1
80.00    1         1
```



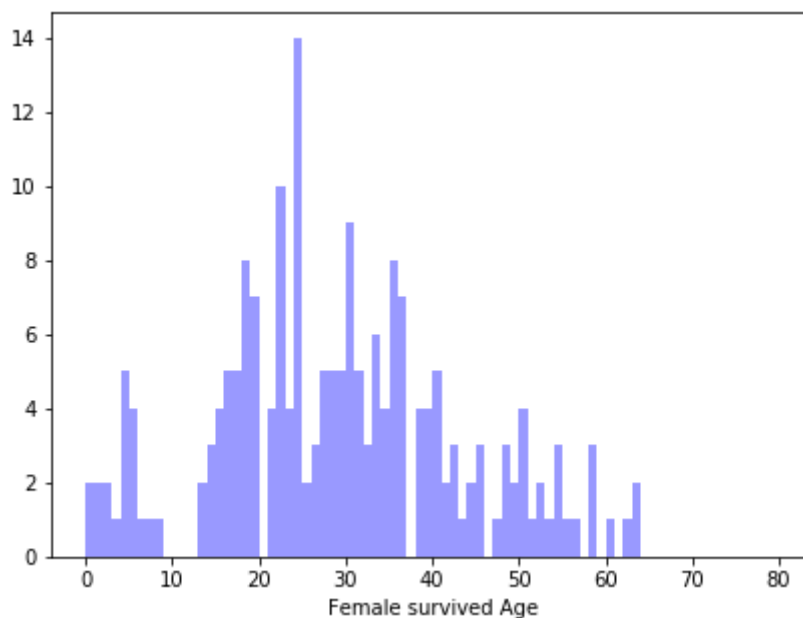
La figure n'est pas du tout claire, on ne peut rien conclure concernant le facteur age sur la survie des passagers!

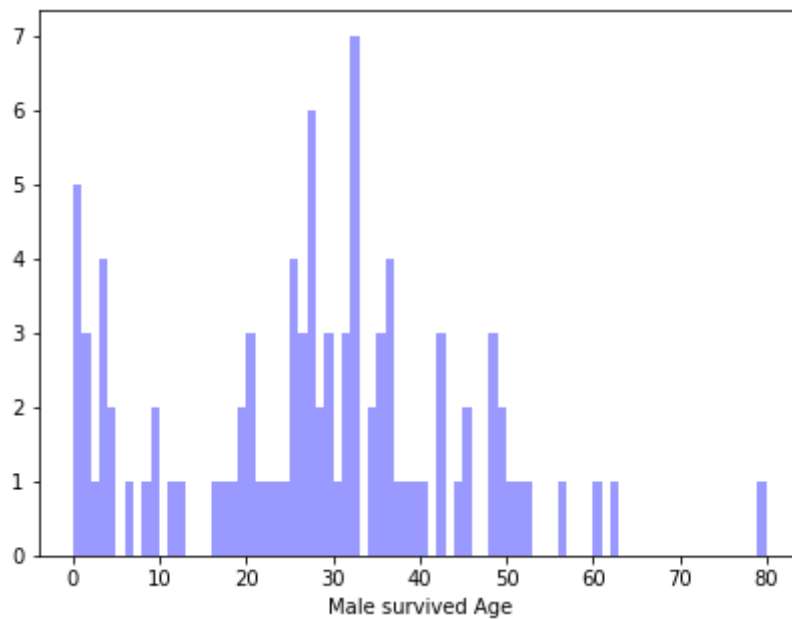
**Alors c'est quoi la cause?**

On a des données qui manquent! Par exemple le passager ayant pour "PassengerId" 6 n'est pas renseigné on retrouve NaN au lieu de son age.

### 0.3.10 Relation entre Age, sex et survie :

Même avec les données qui manquent dans la colonne age, nous pouvons quand même étudier la relation entre l'âge, le sexe et la survie





Vous pouvez voir que les hommes ont une forte probabilité de survie lorsqu'ils ont entre 20 et 35 ans, ce qui est également un peu vrai pour les femmes mais pas complètement. Pour les femmes, les chances de survie sont plus élevées entre 14 et 40.

Pour les hommes, la probabilité de survie est très faible entre 5 et 18 ans, mais ce n'est pas vrai pour les femmes.

Puisqu'il semble y avoir certains âges, qui ont augmenté les chances de survie nous étudions plus tard le traitement des données manquantes dans la colonne age.

**Comment traiter donc les données manquantes ?**

## 0.4 Missing Data et manipulation des données :

	Total	%
Cabin	687	77.1
Age	177	19.9
Embarked	2	0.2
Fare	0	0.0
Ticket	0	0.0

Embarked n'a que 2 valeurs manquantes, qui peuvent facilement être remplies. Il sera beaucoup plus délicat de gérer la fonction 'Age', qui a 177 valeurs manquantes. Le descripteur "Cabine" nécessite une enquête plus approfondie, mais il semble que nous pourrions vouloir la supprimer du jeu de données, car 77% d'entre elles sont manquantes.

### 0.4.1 Etude des descripteurs :

Avant de passer à l'étape de prédiction, nous avons besoin de bien étudier notre base de donnée, autrement dit, nous devons tout d'abord :

1/Trouver une solution pour les données manquantes.

2/Ajouter des nouvelles données à partir des données qui existent déjà.

3/Convertir certaines données catégorielles en numérique.

4/Nettoyer les descripteurs nécessaires pour la prédiction.

5/Se débarrasser des données qui nous paraissent inutiles pour la prédiction.

Tout d'abord dans toute cette partie j'ai créé une nouvelle dataset que j'ai nommé all\_data qui contient les deux dataset train et test.

### 1/Solution pour les données manquantes :

#### \*Age :

Nous rappelons que la colonne age contient 177 valeurs manquantes Pour ce fait, j'ai créé donc un tableau contenant des nombres aléatoires calculés entre la moyenne, l'écart type et is\_null ( notons que is\_null retourne le nombre des valeurs NaN dans notre nouvelle dataset) puis remplir les valeurs NaN dans la colonne Age avec des valeurs aléatoires générées.

```
In [1091]: dataset["Age"].isnull().sum()
Out[1091]: 0

In [1092]: Data["Age"].isnull().sum()
Out[1092]: 0

In [1093]: test["Age"].isnull().sum()
Out[1093]: 0
```

Nous obtenons enfin 0 valeurs manquantes dans la colonne age.

#### \*Embarked :

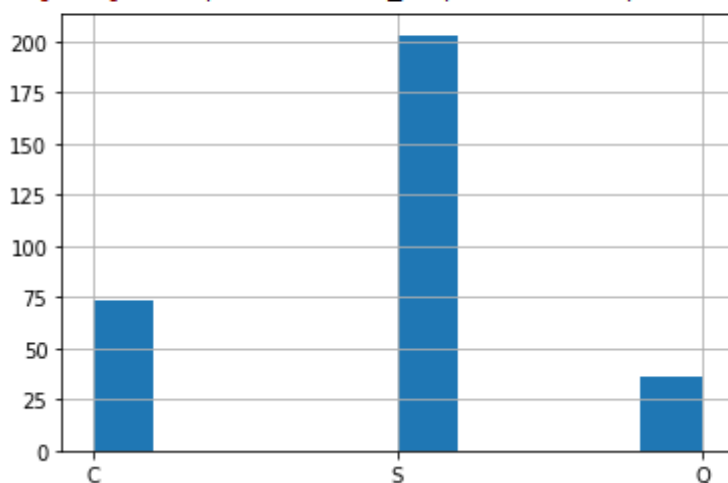
Nous rappelons aussi, que dans la colonne Embarked nous avons 2 valeurs manquantes

```
In [1094]: print(Data[Data.Embarked.isnull()])
   PassengerId  Survived  Pclass  ...  Fare  Cabin  Embarked
61             62         1      1  ...   80.0    B28        NaN
829            830         1      1  ...   80.0    B28        NaN
```

Ces deux valeurs manquantes se trouvent alors dans la colonne où Data['Ticket'] == '113572' et Data['Cabin'] == 'B28'

```
In [1096]: EmbarkedFemale = Data[(Data['Embarked'].notnull()) & ((Data['Sex'] ==
'female'))]
```

```
In [1097]: EmbarkedFemale['Embarked'].hist() #S is the most used by Females
Out[1097]: <matplotlib.axes._subplots.AxesSubplot at 0x238ab509a08>
```



On remarque ici que embarked = 'S' est le plus utilisé pour les femmes, donc il suffit de remplacer les valeurs NaN de Embarked par 'S'.

Nous obtenons enfin 0 valeurs manquantes dans Embarked

```
In [1098]: Data['Embarked'].fillna('S',inplace=True)
...: Data['Embarked'].isnull().sum()
Out[1098]: 0
```

#### \*Cabin :

Dans la colonne Cabin nous avons 77% des données manquantes ce nécessite une enquête plus approfondie,nous pourrions vouloir la supprimer du jeu de données.

```
Data = Data.drop(['Cabin'], axis=1)
```

```
test = test.drop(['Cabin'], axis=1)
```

#### \*Fare :

Concernant la colonne Fare, nous remarquons qu'on a une valeur manquante dans notre dataset, il suffit de la remplacer par la valeur 0.

```
In [1102]: for dataset in all_data:
...:     dataset['Fare'] = dataset['Fare'].fillna(0)
...:     dataset['Fare'] = dataset['Fare'].astype(int)

In [1103]: dataset["Fare"].isnull().sum()
Out[1103]: 0
```

**Nous n'avons plus donc des valeurs manquantes dans notre dataset.**

**2/ Ajouter des nouvelles données :**

#### FamilySize :

Les deux colonnes SibSp et Parch ne sont pas directement liées à la survie ou non d'un passager X. C'est là que l'idée de créer une nouvelle fonctionnalité est apparue.

Pour chaque passager, nous déterminerons la taille de sa famille en ajoutant SibSp + Parch + 1 (lui-même). La taille de la famille varie d'un minimum de 1 à un maximum de 11, la taille de la famille de 4 ayant le taux de survie le plus élevé de 72%

	FamilySize	Survived
0	1	0.303538
1	2	0.552795
2	3	0.578431
3	4	0.724138
4	5	0.200000
5	6	0.136364
6	7	0.333333
7	8	0.000000
8	11	0.000000

**Cela semble avoir un bon effet sur notre prédiction!**

#### Title :

Tout d'abord, nous obtenons le titre du nom et les stockons dans une nouvelle liste appelée titles.

```
titles = "Mr" : 1, "Miss" : 2, "Mrs" : 3, "Master" : 4, "Rare" : 5
```

À partir du nom, nous devons récupérer le titre associé à ce nom, à savoir Mr, Miss, Mrs, Master, Rare.

#### 3/Conversion des données :

Nous devons maintenant sélectionner les données à utiliser et les convertir tous en valeur numérique

```
In [1112]: print(pd.crosstab(Data['Title'], Data['Sex']))
Sex      female  male
Title
1           0    517
2         185     0
3         126     0
4           0    40
5           3    20

In [1113]: print(Data[['Title', 'Survived']].groupby(['Title'], as_index =
False).mean())
Title  Survived
0      1  0.156673
1      2  0.702703
2      3  0.793651
3      4  0.575000
4      5  0.347826
```

en utilisant la méthode `astype(int)`.

Les données que j'ai du les convertir sont, le sex, l'embarked.

J'ai attribué les valeurs 0 et 1 pour le sex

`genders = "male" : 0, "female" : 1`

0,1,2 pour l'embarked :

`ports = "S" : 0, "C" : 1, "Q" : 2`

**Nous obtenons enfin que des valeurs numériques.**

#### 4/Nettoyage de la dataset :

Dans cette partie nous allons attribuer que des des valeurs comprises entre 1 et 7 pour chacun des 'Fare' et 'Age'.

J'ai diviser chaque colonne en un intervalle précis et chaque intervalle je l'ai attribué une valeur correspondante.

Ceci aide beaucoup pour notre prédiction pour la suite.

Par exemple pour le fare < 7.91 j'ai donné la valeur 0

Pour fare compris entre 7.91 et 14.45 j'ai donné la valeur 1

De meme pour l'age, après avoir rempli les valeurs manquantes, pour l'age <11 j'ai attribué la valeur 0 pour l'age compris entre 11 et 18 j'ai attribué la valeur 1 et ainsi de suite.

**5/Suppression des données inutiles pour la prédiction :** Nous avons déjà supprimé la colonne Cabin qui contient 77% des valeurs manquantes et je vais aussi supprimer «PassengerId» car cela ne contribue pas à la probabilité de survie d'une personne. Meme chose pour "Ticket" et "Name" car elle sont inutiles pour nos prédictions.

**Notre dataset finale est donc :**

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	Title
0	0	3	0	2	1	0	0	0	2	1
1	1	1	1	5	1	0	3	1	2	3
2	1	3	1	3	0	0	0	0	1	2
3	1	1	1	5	1	0	3	0	2	3
4	0	3	0	5	0	0	1	0	1	1
5	0	3	0	3	0	0	1	2	1	1
6	0	1	0	6	0	0	3	0	1	1
7	0	3	0	0	3	1	2	0	5	4
8	1	3	1	3	0	2	1	0	3	3
9	1	2	1	1	1	0	2	1	2	3



**C'est BON! Nous avons terminé la partie du nettoyage de notre dataset qui nous servira pour notre prédiction.**

**Maintenant, nous n'avons qu'à prédire notre résultat**

## 0.5 Prédiction de l'âge :

Avant tout, on a besoin de toute cette partie d'importer la bibliothèque sklearn.

Ensuite, il est nécessaire de séparer les données et le label. Dans notre cas, le label(colonne cible) correspond à la colonne Age. Il est donc nécessaire de séparer les données contenues dans la colonne Age (après les modifications qu'elle a subi) des autres données afin d'avoir 2 tableaux : un tableau contenant les données que nous sélectionnons(Toutes les variables de notre nouvelles Dataset sauf l'âge) et un tableau contenant uniquement le label "Age".

Un modèle de machine learning n'a de sens que s'il est possible de l'évaluer. La méthode classique pour effectuer cette évaluation est de séparer les données de départ (les 891 passagers) en 2 groupes : un groupe que l'on nomme souvent "train" qui servira à "entraîner" l'algorithme de machine learning et un groupe "test" qui servira à évaluer la capacité de l'algorithme à prévoir des résultats corrects. Après cette séparation nous devons avoir les tableaux suivants : X\_train (les données d'entraînement), y\_train (les labels correspondant aux données d'entraînement), X\_test (les données de test) et y\_test (les labels correspondant aux données de test).

```
columns_cible=["Age"]
columns_Data=["Survived ","Pclass ","Sex", "SibSp", "Parch",
"Fare", "Embarked", "FamilySize", "Title"]
```

```
X=Data[columns_Data]
y=Data[columns_cible]
```

```
X_train , X_test , y_train , y_test=train_test_split(X,y,
test_size=0.2 , random_state=0)
```

**L'étape qui suit la séparation des données est faire la prédiction de l'âge categorie.**

La méthode fit de scikit learn nous permet de procéder à l'entraînement de l'algorithme (en utilisant les données Xtrain et ytrain).

La méthode predict, toujours de scikit learn, nous permet de tester l'entraînement de notre algorithme.

Cette méthode va permettre de générer un tableau que nous nommons y\_predictions, contiendra les labels prévus par l'algorithme pour les données contenues dans Xtest.

Pour évaluer l'entraînement de l'algorithme, il suffira de comparer ypredictions et ytest.

Il y'a différentes méthode pour faire la prédiction, telles qu'on cite (SVM,KNN,DecisionTree,RandomForest, Gauss, logregression.)

Nous allons essayer dans cette partie la méthode de RandomForest pour prédire l'âge.

Le score de notre age prédit est de 74.02%

```
In [610]: print(classification_report(y_test,forest_predictions))
```

	precision	recall	f1-score	support
0	0.77	0.83	0.80	12
1	0.41	0.39	0.40	18
2	0.32	0.20	0.24	30
3	0.25	0.27	0.26	30
4	0.21	0.26	0.24	34
5	0.25	0.17	0.20	24
6	0.38	0.48	0.43	31
7	0.00	0.00	0.00	0
accuracy			0.33	179
macro avg	0.32	0.33	0.32	179
weighted avg	0.33	0.33	0.32	179

```
In [611]: print("the train Age score is : ", acc_forest)
the train Age score is : 74.02
```

## 0.6 Prediction de la survie :

### 0.6.1 Avec l'âge prédit :

Notre colonne cible dans cette partie c'est "Survived" nous essayons de prédire cette dernière à partir des résultats trouvés précédemment, c'est à dire à partir des colonnes que nous avons déjà crée(toutes les colonnes sauf "survived")

Nous importons les bibliothèques suivantes qu'on nous utilisons par la suite :

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
```

```
#prediction de la survie en utilisant l'age pr dit
columns_cible=["Survived"]
columns_Data=["Pclass","Sex","Age","SibSp","Parch","Fare",
"Embarked","FamilySize","Title"]
```

```
X=Data[columns_Data]
y=Data[columns_cible]
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,
test_size=0.2,random_state=0)
```

Afin d'avoir une prédiction précise, nous allons utiliser les différentes méthodes de machine learning nommées précédemment.

On commence par la méthode de **RandomForest** :

Le score de la prédiction de la survie en utilisant la méthode de random forest est 95.79%

```
In [626]: acc_forest = round (forest.score (X_train, y_train) * 100, 2)
...: print("the survived score is : ", acc_forest)
the survived score is : 95.79
```

```
In [627]: print(classification_report(y_test, forest_predictions))
           precision    recall  f1-score   support

    0         0.85        0.90        0.88         110
    1         0.83        0.75        0.79          69

 accuracy          0.84
 macro avg         0.84        0.83        0.83         179
weighted avg         0.84        0.84        0.84         179
```

```
In [628]: confusion_matrix (y_test, forest_predictions)
Out[628]:
array([[99, 11],
       [17, 52]], dtype=int64)
```

### Interprétation de la matrice de confusion :

La première ligne concerne les prédictions de non-survie : 99 passagers ont été correctement classés comme non survécus (appelés vrais négatifs) et 11 lorsqu'ils ont été classés à tort comme non survécus (faux positifs)

La deuxième rangée concerne les prévisions de survie : 17 passagers ont été classés à tort comme ayant survécu (faux négatifs) et 52 ont été correctement classés comme ayant survécu (vrais positifs)

La matrice de confusion nous donne beaucoup d'informations sur la performance de notre modèle.

### Cherchons le score avec la validation croisée

#### Qu'est ce qu'une validation croisée?

La validation croisée désigne le processus qui permet tester la précision prédictive d'un modèle dans un échantillon test par rapport à la précision prédictive de l'échantillon d'apprentissage.

Elle divise au hasard les données d'apprentissage en K sous-ensembles appelés plis. Imaginons que nous divisons nos données en 4 fois (K = 4). Notre modèle de forêt aléatoire serait formé et évalué 4 fois, en utilisant un pli différent pour l'évaluation à chaque fois, tandis qu'il serait formé sur les 3 plis restants.

La validation croisée K-Fold sur notre modèle de forêt aléatoire, en utilisant 10 plis (cv= 10). Par conséquent, il génère un tableau avec 10 scores différents.

```
In [1134]: print ("Scores using cross validation:", scores)
Scores using cross validation: [0.81944444 0.77777778 0.84722222 0.85915493 0.78873239
0.83098592
0.81690141 0.8028169 0.81690141 0.81428571]

In [1135]: print ("Mean:", scores.mean ())
Mean: 0.8174223116476638

In [1136]: print ("Standard Deviation:", scores.std ())
Standard Deviation: 0.02331567239170287
```

Cela semble beaucoup plus réaliste qu'auparavant. Notre modèle a une précision moyenne de 82 avec un écart type de 2

L'écart type nous montre à quel point les estimations sont précises. Cela signifie dans notre cas que la précision de notre modèle peut différer de  $\pm 2\%$ . Je pense que la précision est toujours très bonne et comme la forêt aléatoire est un modèle facile à utiliser, nous allons essayer d'augmenter encore ses performances dans la section suivante.

**Prédiction avec decisionTree :**

```

In [638]: confusion_matrix (y_test, y_predage)
Out[638]:
array([[95, 15],
       [20, 49]], dtype=int64)

In [639]: accuracy_score(y_test, forest_predictions)
Out[639]: 0.8435754189944135

In [640]: acc_tree= round (decisiontree.score (X_train, y_train) * 100, 2)

In [641]: print("l'accuracy of age_categories is : " , acc_tree)
l'accuracy of age_categories is : 95.79

In [642]: scores = cross_val_score (decisiontree, X_train, y_train, cv = 7)

In [643]: print ("Scores:", scores)
Scores: [0.74509804 0.76470588 0.7254902  0.75490196 0.80392157 0.7029703
 0.83168317]

In [644]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.7612530160579082

In [645]: print ("l'ecart type du score est :", scores.std ())
l'ecart type du score est : 0.04100541396539263

```

Le score trouvé par DecisionTree est le meme que celui du RandomForest c'est 95.79 mais la moyenne du score trouvée par la validation croisée diffère c'est 76% et l'écart type c'est 0.04

**Prédiction avec GaussianNB :**

```

In [690]: gauss_predictions = gauss.predict (X_test)

In [691]: accuracy_score(y_test,gauss_predictions)
Out[691]: 0.770949720670391

In [692]: confusion_matrix (y_test, gauss_predictions)
Out[692]:
array([[84, 26],
       [15, 54]], dtype=int64)

In [693]: print(classification_report(y_test,gauss_predictions))
          precision    recall  f1-score   support

     0           0.85       0.76       0.80         110
     1           0.68       0.78       0.72          69

   accuracy                   0.77         179
  macro avg           0.76       0.77       0.76         179
 weighted avg           0.78       0.77       0.77         179

In [694]: acc_gauss = round (gauss.score (X_train, y_train) * 100, 2)

In [695]: print("the train score is : " , acc_gauss)
the train score is : 81.32

```

```

In [697]: print ("Scores:", scores)
Scores: [0.83333333 0.73611111 0.84722222 0.84507042 0.77464789 0.83098592
0.78873239 0.76056338 0.88732394 0.82857143]

In [698]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.8132562038900067

In [699]: print ("Lecart type du score est :", scores.std ())
Lecart type du score est : 0.04407241000085448

```

### Prédiction avec KNN :

```

In [664]: knn_predictions = knn.predict (X_test)

In [665]: accuracy_score(y_test, knn_predictions)
Out[665]: 0.7486033519553073

In [666]: confusion_matrix (y_test, knn_predictions)
Out[666]:
array([[89, 21],
       [24, 45]], dtype=int64)

In [667]: print(classification_report(y_test,knn_predictions))
           precision    recall  f1-score   support

    0           0.79       0.81       0.80         110
    1           0.68       0.65       0.67          69

   accuracy                   0.75         179
  macro avg           0.73       0.73       0.73         179
 weighted avg           0.75       0.75       0.75         179

In [668]: acc_knn = round (knn.score (X_train, y_train) * 100, 2)

In [669]: print("the train score is : " , acc_knn)
the train score is : 82.87

```

```

In [671]: print ("Scores:", scores)
Scores: [0.73611111 0.72222222 0.76388889 0.77464789 0.76056338 0.74647887
0.64788732 0.76056338 0.78873239 0.78571429]

In [672]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.7486809747373128

In [673]: print ("Lecart type du score est :", scores.std ())
Lecart type du score est : 0.038968965868915725

```

## Prédiction avec Logistic Regression :

```
In [677]: log_predictions = log.predict (X_test)

In [678]: accuracy_score(y_test,log_predictions)
Out[678]: 0.776536312849162

In [679]: confusion_matrix (y_test, log_predictions)
Out[679]:
array([[88, 22],
       [18, 51]], dtype=int64)

In [680]: print(classification_report(y_test,log_predictions))
           precision    recall  f1-score   support

      0       0.83       0.80       0.81       110
      1       0.70       0.74       0.72        69

   accuracy                   0.78       179
  macro avg       0.76       0.77       0.77       179
 weighted avg       0.78       0.78       0.78       179


In [681]: acc_log = round (log.score (X_train, y_train) * 100, 2)

In [682]: print("the train score is : " , acc_log)
the train score is : 82.02

In [684]: print ("Scores:", scores)
Scores: [0.81944444 0.73611111 0.83333333 0.84507042 0.8028169  0.83098592
 0.78873239 0.8028169  0.88732394 0.85714286]

In [685]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.8203778224904987

In [686]: print ("Lecart type du score est :", scores.std ())
Lecart type du score est : 0.039298227910625275
```

## Prédiction avec SVM :

```
In [650]: print(classification_report(y_test,svm_predictions))
           precision    recall  f1-score   support

      0       0.84       0.79       0.82       110
      1       0.70       0.77       0.73        69

   accuracy                   0.78       179
  macro avg       0.77       0.78       0.77       179
 weighted avg       0.79       0.78       0.78       179


In [651]: accuracy_score(y_test, svm_predictions)
Out[651]: 0.7821229050279329

In [652]: confusion_matrix (y_test, svm_predictions)
Out[652]:
array([[87, 23],
       [16, 53]], dtype=int64)

In [653]: acc_svc_lin = round (svc_lin.score (X_train, y_train) * 100, 2)

In [654]: print("the train score is : " , acc_svc_lin)
the train score is : 82.87
```

```

In [657]: confusion_matrix (y_test, svm_predictions)
Out[657]:
array([[87, 23],
       [16, 53]], dtype=int64)

In [658]: print ("Scores:", scores)
Scores: [0.81944444 0.77777778 0.81944444 0.85915493 0.8028169  0.84507042
         0.8028169  0.78873239 0.88732394 0.85714286]

In [659]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.8259725016767272

In [660]: print ("l'ecart type du score est :", scores.std ())
l'ecart type du score est : 0.03326942486770457

```

## 0.6.2 Classement des meilleures classificateurs pour la prédiction :

Parmis les score trouvé, j'ai choisi une méthode qui me permet de trier par ordre croissant les meilleures méthode pour faire ma prédiction de la survie d'un passager.

```

In [701]: results = pd.DataFrame ({
...:     'Model': ['SVM', 'KNN', 'Random Forest', 'DecisionTree', 'Log', 'Gauss'],
...:     'Score': [acc_svc_lin, acc_knn,
...:               acc_forest, acc_tree, acc_log, acc_gauss]})
...: results.sort_values(by='Score')
Out[701]:
   Model  Score
5   Gauss  81.32
4    Log  82.02
0    SVM  82.87
1    KNN  82.87
2 Random Forest  95.79
3 DecisionTree  95.79

```

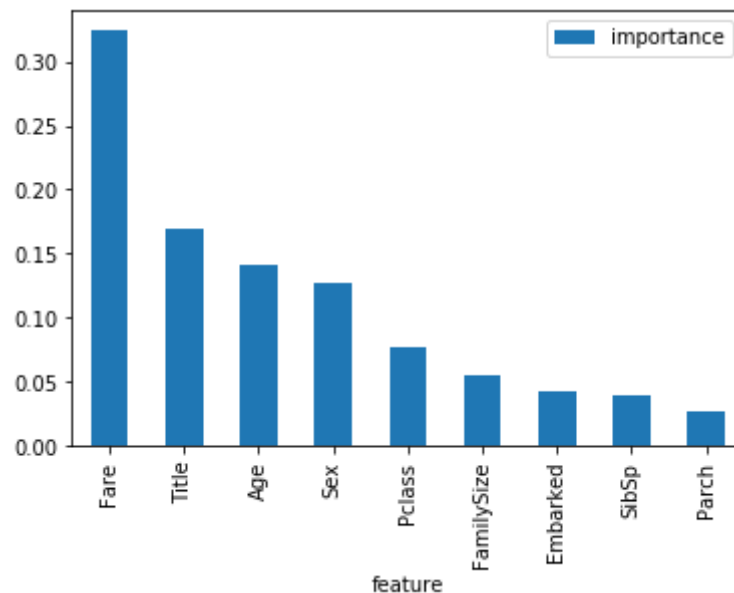
Comme nous pouvons le voir, le classificateur Random Forest et Decision tree passent en premier lieu.

## 0.6.3 Importance des descripteurs :

Une autre grande qualité de la forêt aléatoire est qu'elle permet de mesurer très facilement l'importance relative de chaque entité. Sklearn mesure l'importance des caractéristiques en examinant dans quelle mesure les nœuds treee, qui utilisent cette fonction, réduisent l'impureté en moyenne (sur tous les arbres de la forêt). Il calcule ce score automatiquement pour chaque fonctionnalité après la formation et met à l'échelle les résultats de sorte que la somme de toutes les importances soit égale à 1

```
In [702]: importances =
pd.DataFrame({'feature':X_train.columns,'importance':np.round(forest.feature_importances_,3)})
...: importances =
importances.sort_values('importance',ascending=False).set_index('feature')
...: importances.head(15)
Out[702]:
```

feature	importance
Fare	0.324
Title	0.169
Age	0.141
Sex	0.127
Pclass	0.077
FamilySize	0.055
Embarked	0.042
SibSp	0.039
Parch	0.027



#### 0.6.4 Avec la médiane de l'âge :

#je garde les memes changements sur les autres descripteurs  
sauf l'age je remplace les valeurs NaN par la mediane

```
Data['Age']=Data['Age'].fillna(Data['Age'].median())
test['Age']=test['Age'].fillna(test['Age'].median())
Data["Age"] = Data["Age"].astype(int)
```

Notre Dataset est donc la suivante :



```
In [1155]: print(Data.head(10))
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FamilySize	Title
0	0	3	0	22	1	0	0	0	2	1
1	1	1	1	38	1	0	3	1	2	3
2	1	3	1	26	0	0	0	0	1	2
3	1	1	1	35	1	0	3	0	2	3
4	0	3	0	35	0	0	1	0	1	1
5	0	3	0	28	0	0	1	2	1	1
6	0	1	0	54	0	0	3	0	1	1
7	0	3	0	2	3	1	2	0	5	4
8	1	3	1	27	0	2	1	0	3	3
9	1	2	1	14	1	0	2	1	2	3

Je refais exactement le meme travail que j'ai fait précédemment pour la prédiction de la survie mais cette fois ci avec l'age médiane.

## Random Forest

```
array([[100, 10],
       [ 21, 48]], dtype=int64)
```

```
In [1169]: print(classification_report(y_test,forest_predictions))
```

	precision	recall	f1-score	support
0	0.83	0.91	0.87	110
1	0.83	0.70	0.76	69
accuracy			0.83	179
macro avg	0.83	0.80	0.81	179
weighted avg	0.83	0.83	0.82	179

```
In [1170]: print("the train score is : ", acc_forest)
the train score is : 94.94
```

```
In [1171]: print ("Scores:", scores)
Scores: [0.79166667 0.73611111 0.84722222 0.91549296 0.77464789 0.81690141
0.77464789 0.77464789 0.81690141 0.8 ]
```

```
In [1172]: print ("Mean:", scores.mean ())
Mean: 0.8048239436619719
```

```
In [1173]: print ("Standard Deviation:", scores.std ())
Standard Deviation: 0.04688986657120149
```

## DecisionTree

```
array([[96, 14],
       [21, 48]], dtype=int64)

In [1180]: accuracy_score(y_test, forest_predictions)
Out[1180]: 0.8268156424581006

In [1181]: acc_tree= round (decisiontree.score (X_train, y_train) * 100, 2)

In [1182]: print("l'accuracy of age_categories is : " , acc_tree)
l'accuracy of age_categories is : 94.94

In [1183]: confusion_matrix (y_test, y_predage)
Out[1183]:
array([[96, 14],
       [21, 48]], dtype=int64)

In [1184]: print ("Scores:", scores)
Scores: [0.69607843 0.7745098 0.79411765 0.7745098 0.7745098 0.76237624
        0.78217822]

In [1185]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.7654685636630889

In [1186]: print ("l'ecart type du score est :", scores.std ())
l'ecart type du score est : 0.029688303462536706
```

## Log

```
In [1220]: print(classification_report(y_test,log_predictions))
           precision    recall  f1-score   support

    0       0.84        0.80        0.82         110
    1       0.70        0.75        0.73          69

 accuracy          0.78         0.78         0.78         179
 macro avg         0.77         0.78         0.77         179
 weighted avg         0.79         0.78         0.78         179

In [1221]: acc_log = round (log.score (X_train, y_train) * 100, 2)

In [1222]: print("the train score is : " , acc_log)
the train score is : 82.72

In [1223]: print ("Scores:", scores)
Scores: [0.80555556 0.76388889 0.83333333 0.85915493 0.81690141 0.81690141
        0.78873239 0.78873239 0.90140845 0.85714286]

In [1224]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.8231751620836129

In [1225]: print ("Lecart type du score est :", scores.std ())
Lecart type du score est : 0.0386634518078401
```

## Gaussian

```
In [1234]: acc_gauss = round (gauss.score (X_train, y_train) * 100, 2)

In [1235]: print("the train score is : " , acc_gauss)
the train score is : 80.34

In [1236]: print ("Scores:", scores)
Scores: [0.80555556 0.72222222 0.83333333 0.76056338 0.77464789 0.83098592
        0.77464789 0.77464789 0.90140845 0.82857143]

In [1237]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.8006583948133243

In [1238]: print ("Lecart type du score est :", scores.std ())
Lecart type du score est : 0.04767660472117795
```

## KNN

```
In [1207]: print(classification_report(y_test,knn_predictions))
              precision    recall  f1-score   support

         0       0.79      0.85      0.82        110
         1       0.73      0.65      0.69         69

 accuracy          0.77          179
 macro avg          0.76      0.75      0.75          179
 weighted avg          0.77      0.77      0.77          179

In [1208]: acc_knn = round (knn.score (X_train, y_train) * 100, 2)

In [1209]: print("the train score is : " , acc_knn)
the train score is : 85.25

In [1210]: print ("Scores:", scores)
Scores: [0.72222222 0.68055556 0.81944444 0.81690141 0.83098592 0.76056338
 0.73239437 0.71830986 0.87323944 0.82857143]

In [1211]: print ("Lecart type du score est :", scores.std ())
Lecart type du score est : 0.06018211929495524

In [1212]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.7783188016990834
```

## SVM

```
              precision    recall  f1-score   support

         0       0.84      0.79      0.82        110
         1       0.70      0.77      0.73         69

 accuracy          0.78          179
 macro avg          0.77      0.78      0.77          179
 weighted avg          0.79      0.78      0.78          179

In [1194]: confusion_matrix (y_test, svm_predictions)
Out[1194]:
array([[87, 23],
       [16, 53]], dtype=int64)

In [1195]: acc_svc_lin = round (svc_lin.score (X_train, y_train) * 100, 2)

In [1196]: print ("Scores:", scores)
Scores: [0.81944444 0.77777778 0.81944444 0.85915493 0.78873239 0.83098592
 0.77464789 0.78873239 0.90140845 0.85714286]

In [1197]: print ("La moyenne du score est :", scores.mean ())
La moyenne du score est : 0.8217471495640509

In [1198]: print ("l'ecart type du score est :", scores.std ())
l'ecart type du score est : 0.03927553309364122
```

### 0.6.5 Classement des classificateurs pour la deuxième méthode de prédiction :

	Model	Score
5	Gauss	80.34
3	DecisionTree	80.45
0	SVM	82.16
4	Log	82.72
1	KNN	85.25
2	Random Forest	94.94

Comme toujours RandomForest garde toujours le premier rang.

## 0.7 Evaluation approfondie :

Maintenant que nous avons un modèle approprié, nous pouvons commencer à évaluer sa performance d'une manière plus précise.

Mise à part la matrice de confusion qui joue un rôle important sur la performance de notre modèle il existe un moyen d'en obtenir encore plus, comme calculer la précision des classificateurs.

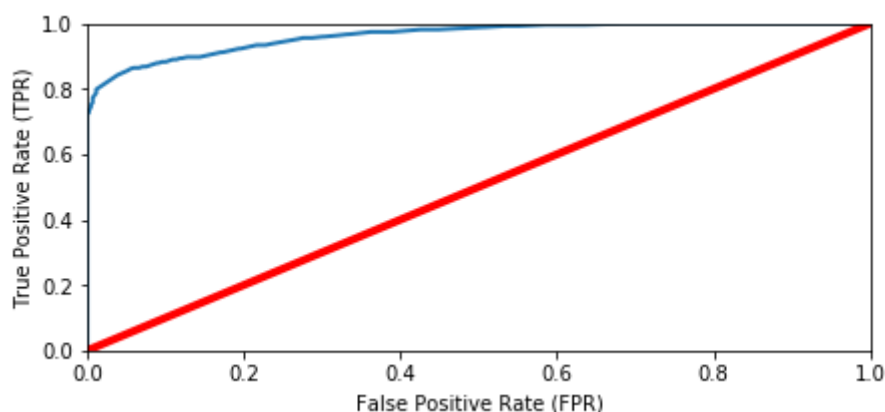
### 0.7.1 Precision & Recall :

Precision: 0.801948051948  
Recall: 0.722222222222

Notre modèle prédit 81% du temps, une survie correcte des passagers (précision). Le 'recall' nous dit qu'il prédit la survie de 73% des personnes qui ont réellement survécu

### 0.7.2 ROC AUC :

Une autre façon d'évaluer et de comparer votre classificateur binaire est fournie par la courbe ROC AUC. Cette courbe représente le vrai taux positif (également appelé rappel) par rapport au taux de faux positifs (ratio d'instances négatives mal classées)



### 0.7.3 Score ROC AUC :

Le score ROC AUC est le score correspondant à la courbe ROC AUC. Il est simplement calculé en mesurant l'aire sous la courbe, qui est appelée AUC.

Un classificateur 100% correct aurait un score ROC AUC de 1 et un classifieur complètement aléatoire aurait un score de 0,5.

```
In [65]: from sklearn.metrics import roc_auc_score

In [66]: r_a_score = roc_auc_score(y_train, y_scores)
...: print("ROC-AUC-Score:", r_a_score)
ROC-AUC-Score: 0.9632322878336547
```

Agréable! Je pense que ce score est assez bon pour soumettre les prédictions de l'ensemble de test sur Kaggle.


## 0.8 Soumettre le résultat sur Kaggle!

Your most recent submission

Name	Submitted	Wait time	Execution time	Score
cha (1).csv	3 minutes ago	1 seconds	0 seconds	0.83732

Complete

[Jump to your position on the leaderboard](#)

332	Chadha Mhalla		0.83732	10	12m
-----	---------------	---	---------	----	-----

Your Best Entry

**Super! Notre prediction est bien 83%!!**

## 0.9 Testons si vous allez survivre ou non? :D

Un simple code que j'ai mis à la fin de ce projet, nous permet de savoir si vraiment un passager X va survivre ou non selon les descripteurs que nous allons choisir!

Choisissons par exemple que le passager soit :

pclasse est 3

sex est male donc 0

age est 6 cad age entre 44 et 60

Sisbp choisi c'est 0 et Parch aussi 0

Fare choisi c'est 1

familysize 1

title est 1 un Mr

```

In [1019]: my_survival = [[3,0,6,0, 0, 1, 0,1,1]]

In [1020]: pred = forest.predict(my_survival)
...: print(pred)
[0]

In [1021]: if pred == 0:
...:     print('Oh non! Tu ne survivras pas malheureusement! :( ' )
...: else:
...:     print('OUIIIIII! Tu survivras!! :D ')
Oh non! Tu ne survivras pas malheureusement! :(

```

Voilà donc pour ces descripteurs choisis, notre modele nous informe que ce passager ne survivrait pas malheureusement.

Testons une deuxieme fois avec d'autres descripteurs :

pclasse est 1

sex est femelle donc 1

age est 6 cad age entre 44 et 60

Sisbp choisi c'est 0 et Parch aussi 0

Fare choisi c'est 1

familysize 1

title est 2 une Mme

```

In [79]: my_survival = [[1,1,6,0, 0, 1, 0,1,2]]

In [80]: pred = forest.predict(my_survival)
...: print(pred)
[1]

In [81]: if pred == 0:
...:     print('Oh non! Tu ne survivras pas malheureusement! :( ' )
...: else:
...:     print('OUIIIIII! Tu survivras!! :D ')
OUIIIIII! Tu survivras!! :D

```

## 0.10 Résumé :

Nous avons commencé par l'exploration des données où nous avons eu une idée de l'ensemble des données, vérifié les données manquantes et appris quelles fonctionnalités étaient importantes. Au cours de ce processus, nous avons utilisé seaborn et matplotlib pour effectuer les visualisations.

Au cours de la partie de prétraitement des données, nous avons calculé les valeurs manquantes, converti les entités en valeurs numériques, regroupé les valeurs en catégories et créé quelques nouvelles entités. Ensuite, nous avons commencé à former des différents modèles d'apprentissage, en avons choisi le meilleur parmi ces classifieurs (Random Forest) et y avons appliqué une validation croisée. Ensuite, nous avons discuté du fonctionnement du Random Forest, jeté un coup d'oeil à l'importance qu'elle attribue aux différentes fonctionnalités. Enfin, nous avons examiné sa matrice de confusion et calculé la précision, le recall et le score ROC AUC pour arriver finalement à avoir un meilleur score de notre modèle de prédiction et pour pouvoir à la fin tester si un passager X aurait la chance de survivre ou non.

*\*\* Bonne lecture ! :) \*\**