REPUBLIQUE DU SENEGAL

Un peuple-Un but-Une foi

****

MINISTERE DE L'ECONOMIE, DU PLAN ET DE LA COOPERATION

****

(ANSD)

AGENCE NATIONALE DE LA STATISTIQUE ET DE LA DEMOGRAPHIE

****

ECOLE NATIONALE DE LA STATISTIQUE ET DE L'ANALYSE ECONOMIQUE
Pierre Ndiaye (ENSAE)

PROJET DE FIN DE MODULE DE MACHINE LEARNING EN AS3/DS

PROJET DE MACHINE LEARNING : ANALYSE, TRAITEMENT, CLASSIICATION BINAIRE AVEC NEURONE ARTIFICIEL

Rédigé par :

DEKOU VESSOU Jeff-Maruis

SOUS LA SUPERVISION DE :
M. GOMIS

ANNEE ACADEMIQUE 2022-2023

INTRODUCTION

Le naufrage du Titanic est l'un des événements les plus tragiques de l'histoire maritime, qui a coûté la vie à plus de 1500 personnes. Depuis lors, de nombreuses enquêtes ont été menées pour comprendre les raisons de cette catastrophe. Dans ce projet de machine Learning, nous allons explorer la base de données du Titanic et utiliser des techniques de modélisation pour prédire les chances de survie des passagers en fonction de différentes caractéristiques telles que l'âge, le sexe, la classe, etc. Ce projet est un excellent exemple de l'application de l'apprentissage automatique dans le domaine de l'analyse de données historiques, et nous permettra de découvrir des informations intéressantes sur les facteurs qui ont contribué à la survie des passagers du Titanic. Ce projet s'inscrit également dans un contexte d'évaluation des notre compréhension du cours de machine Learning et des différentes méthodes d'élaborations du modèle ainsi que son amélioration.

Nous allons repartir notre travail en trois (3) grandes parties :

- *La présentation de notre base de données*
- *Le traitement des données et la visualisation des variables*
- *La modélisation : Classification binaire avec neurone artificiel*

Notre projet sera effectué avec le langage python et sur google Colab. Nous allons des à présent présenter nos données.

I. PRÉSENTATION DE LA BASE DE DONNÉES

1. Premier regard sur nos données et signification de nos variables

Nos données sont recueillies dans la base de données « base.csv ». Dans cette base, chaque passager possède un identifiant allant de 1 à 891.

Tableau 1: Présentation des données

Variables	Description	Notes sur les valeurs
PassengerID	Identifiant du passager	Entier compris entre 1 et 1309
Survived	Survivant ?	1 si le passager a survécu, 0 s'il est décédé
Pclass	Classe du passager	1 = 1ère classe, 2 = 2ème classe, 3 = 3ème classe
Name	Nom et titre du passager	Style : Nom, titre. Prénoms
Sex	Sexe du passager	'male' ou 'female'
Age	Age du passager	Décimal si inférieur à 1, estimé si de la forme xx.5
SibSp	Nombre d'époux, de frères ou de sœurs présents à bord	
Parch	Nombre de parents ou d'enfants présents à bord	
Ticket	Numéro du ticket	
Fare	Prix des tickets	Le prix est indiqué en £ et pour un seul achat (peut correspondre à plusieurs tickets)
Cabin	Numéro de Cabine	Un ou plusieurs numéros de cabine, de la forme 'A123'
Embarked	Port d'embarcation	C, Q S

2. Installation et importation des packages

Pour une bonne analyse sur Python, il nous faut un certain nombre de packages. On peut énumérer entre autres :

- *Pandas* : traitement (manipulation, analyse) des tableaux de données mixtes (variables numériques, textuelles, booléens ..)
- *Numpy* : manipulation des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.
- *SciKit Learn* : librairie destinée à l'apprentissage automatique. Elle est développée par de nombreux contributeurs notamment dans le monde académique par des instituts français d'enseignement supérieur et de recherche comme l'Inria et Télécom ParisTech. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support. Elle fournit des outils simples et efficaces pour l'extraction de données et l'analyse de données, y compris la préparation des données, la sélection de modèles, l'évaluation, et plus encore. Elle est conçue

pour s'harmoniser avec d'autres bibliothèques libres Python, notamment NumPy et SciPy.

- Et pour la visualisation des données et la génération de graphique : *matplotlib et seaborn*
- TensorFlow : est une bibliothèque de logiciels open-source pour l'apprentissage automatique et l'intelligence artificielle. Il est utilisé pour créer, entraîner et déployer des modèles de machine Learning et des réseaux de neurones artificiels

```
# Pandas pour manipuler les tableaux de données
import pandas as pd
# Numpy pour les listes de données numériques et les fonctions classiques mathématiques
import numpy as np
# scipy (librairie scientifique) pour les fonctions statistiques et autres utilisaires
import scipy
#librairies pour la modélisation , machine learning , traitements et prédiction
import sklearn
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
from tensorflow.keras import layers,Sequential,losses,Input,datasets
import tensorflow as tf
from sklearn.metrics import confusion_matrix
# librairies pour la visualisation de données
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib as mpl
```

3. Importations de notre base CSV :

On importe notre base CSV.

```
#%%Importation de la base zt visualisation
data = pd.read_csv('C:/Users/DELL/Desktop/Projets_AS3/projet/projet/base.csv')
```

4. Visualisation générale de la base

Comme nous avons importés la base, nous allons faire ressortir une vue d'ensemble sur notre base en ce qui concerne la taille les variables et leur nombre de données valides et manquantes.

```
print(data.shape)    # dimension du tableau (891;12)
print(data)
```

Nous allons présenter de façon globale nos données

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0
..                                               ...     ...   ...    ...
886                            Montvila, Rev. Juozas    male  27.0      0
887                     Graham, Miss. Margaret Edith  female  19.0      0
888         Johnston, Miss. Catherine Helen "Carrie"  female   NaN      1
889                            Behr, Mr. Karl Howell    male  26.0      0
890                              Dooley, Mr. Patrick    male  32.0      0

     Parch            Ticket     Fare Cabin Embarked
0        0         A/5 21171   7.2500   NaN        S
1        0          PC 17599  71.2833   C85        C
```

Nous allons également nous intéresser à la visualisation des variables.

```
print(data.columns)
data.isnull().sum()
```

Cette visualisation nous montre des résultats suivants :

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')

Out[7]: PassengerId      0
        Survived         0
        Pclass           0
        Name             0
        Sex              0
        Age            177
        SibSp            0
        Parch            0
        Ticket           0
        Fare             0
        Cabin          687
        Embarked         2
        dtype: int64
```

On peut voir que les variables Age, Cabin et Embarked présentent des valeurs manquantes. Nous allons par la suite expliciter la manière donc nous nous mettrons pour traiter ces données aux valeurs manquantes.

Néanmoins nous devons retenir que le traitement ne peut se faire que sur les variables pertinentes que nous choisirons dans la suite de nos analyses. La partie suivante s'axera alors le choix de nos variables pertinentes, les critères de ce choix, les traitements des valeurs manquantes ou aberrantes s'il a lieu et allons également faire ressortir les motifs du rejet de certains variables.

II. TRAITEMENT ET CHOIX DES VARIABLES

Dans cette partie, nous allons mettre l'accent sur les critères du choix de nos variables ; nos variables choisies sans oublier de faire ressortir les différents traitements effectués au niveau de ces variables.

1. Critère du choix des variables pertinentes de notre modèle

Pour faire le choix de nos variables, nous avons procédé de façon graphique par une visualisation de l'association de notre variable et la variable cible (Survived dans notre cas) d'une part et d'autre part on mesure la corrélation entre les variables qualitatives à l'aide du coefficient de Cramer. Pour certaines variables on va même jusqu'à faire les tableaux croisés.

❖ Visualisation graphique

Cette approche nous permet de représenter l'association de nos variables et de la variable cible. Elle nous aide à faire ressortir les groupes ayant plus survécus ou non. Néanmoins pour confirmer cette hypothèse on s'appuie sur le coefficient de cramer que nous avons construit.

❖ Coefficient de Cramer

✓ *Un peu de théorie*

Contrairement au x^2, il reste stable si l'on augmente la taille de l'échantillon dans les mêmes proportions intermodalités. Il est basé sur le x^2 maximal que le tableau de contingence pourrait théoriquement produire : ce dernier aurait alors une seule case non nulle par ligne ou par colonne (selon que le tableau a plus de lignes ou plus de colonnes). Ce x^2 max théorique est égal à l'effectif multiplié par le plus petit côté du tableau (nombre de lignes ou de colonnes) moins 1. Par exemple un tableau de 2 × 3 avec un effectif de 100 a pour x^2 max 100 × (2 – 1) = 100.

Le V de Cramer est la racine carrée du x^2 divisé par le x^2 max.

$$V = \sqrt{\frac{\chi^2}{\chi^2_{max}}} = \sqrt{\frac{\chi^2}{n \times [\min(l,c) - 1]}}$$

V de Cramer

Plus V est proche de zéro, plus il y a indépendance entre les deux variables étudiées. Il vaut 1 en cas de complète dépendance puisque le x^2 est alors égal au x^2 max (dans un tableau 2 × 2, il prend une valeur comprise entre -1 et 1).

✓ *Mise en œuvre sous python*

```
#%%FONCTION CRAMER
# cette fonction évalue la corrélation entre variables qualitatives en
# - élaboration du tableau de contingence des valeurs
# - calcul du chi2 de cet tableau
# - calcul du coefficient de cramer qui est une normalisation du coefficient chi2

def cramers_v(x, y):
    confusion_matrix = pd.crosstab(x,y)
    chi2 = scipy.stats.chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum().sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2-((k-1)*(r-1))/(n-1))
    rcorr = r-((r-1)**2)/(n-1)
    kcorr = k-((k-1)**2)/(n-1)
    return np.sqrt(phi2corr/min((kcorr-1),(rcorr-1)))
```

- La fonction cramers_v prend en entrée deux variables catégorielles x et y et calcule le coefficient de corrélation Cramer's V entre ces deux variables. Cet indicateur mesure la force de la relation entre deux variables catégorielles et est compris entre 0 (pas de corrélation) et 1 (corrélation parfaite).
- La fonction commence par calculer la matrice de contingence (tableau croisé) entre les deux variables à l'aide de la fonction pd.crosstab de la bibliothèque Pandas. Ensuite, elle utilise la fonction chi2_contingency de la bibliothèque scipy pour calculer le test du chi2 et obtenir la valeur du chi2. En utilisant le nombre total de valeurs dans la matrice de contingence, elle calcule ensuite le phi2.
- La fonction calcule ensuite le phi2 corrigé (phi2corr) en utilisant la formule spécifique de Cramer's V. Enfin, la fonction calcule le coefficient de corrélation corrigé en utilisant les valeurs corrigées de r (nombre de lignes) et de k (nombre de colonnes) et en le divisant par la racine carrée de la plus petite valeur entre k-1 et r-1.
- La fonction retourne ensuite le résultat de la corrélation de Cramer's V calculé entre les deux variables catégorielles x et y.

Des fois pour une bonne visualisation on associe les tableaux croisés aux critères. Sur la base de ces critères nous sommes parvenus à choisir 7 variables dont Une variable créée (Title qui est le titre du passager) et 6 variables locales de notre base : La classe du passager (Pclass), l'âge (Age), le Sexe (Sex), le lieu d'embarcation (Embarked), le nombres de frères et sœur à bord (SibSp) et les parents et enfants (Parch). La partie suivante mettra la lumière sur les raisons du choix de ces variables.

2. Traitement des données et raison du choix de nos variables

Dans cette partie nous allons faire ressortir tous les variables, leur traitement et à partir des critères, faire le choix.

a. Survived : la variable cible

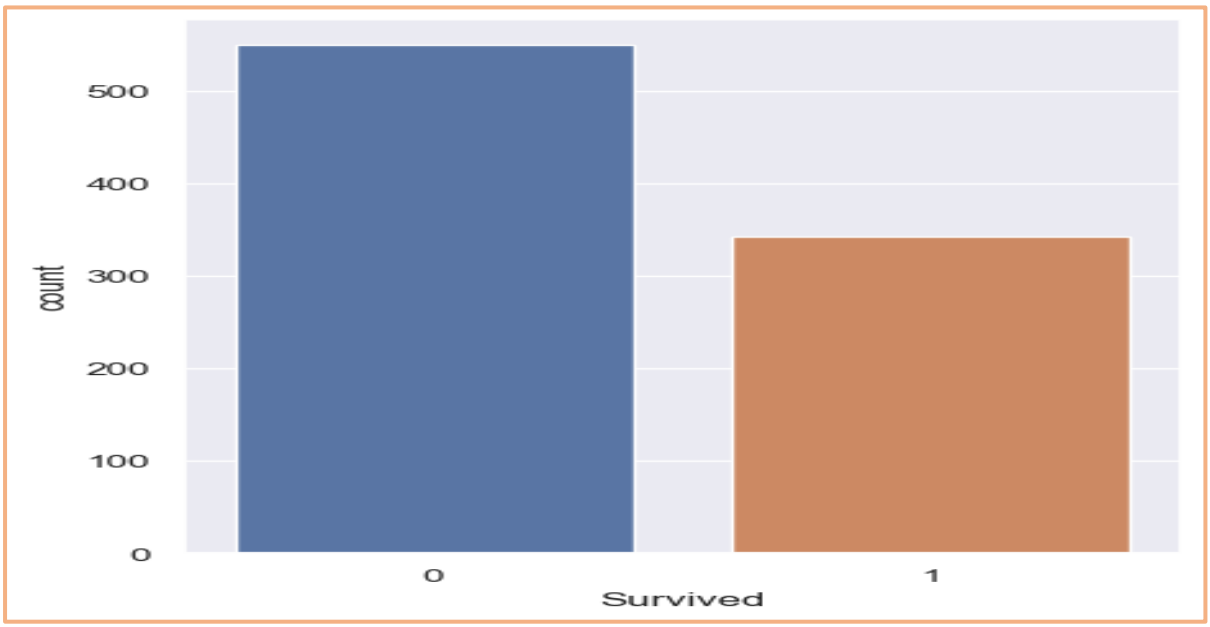
Notre objectif étant de prédire la survie des passagers, il est important avant de commencer notre analyse de visualiser cette variable qu'est la survie.

```
Entrée [17]: #Visualisation variable survived
             data.Survived.value_counts()

Out[17]: 0    549
         1    342
         Name: Survived, dtype: int64
```

Ce résultat nous montre que les données sont numériques (0 pour les passagers disparus, 1 pour les survivants), il n'y a pas de valeurs manquantes. Il n'y a pas également de valeurs aberrantes comme nous le montre visuellement à partir du code ci-dessus ; la répartition des survivants et des victimes :

```
sns.set(rc={'figure.figsize':(5,6)})#figursize pour la largeur et la hauteur de notre plot
_ = sns.countplot(x="Survived", data=data)
```

Figure 1:Repartition des survivants et des victimes



On peut voir clairement qu'il a plus de victimes que de survivants soit 62% de victimes contre 39% de survivants.

b. <u>Pclass : La classe du passager</u>

Comme un avion, il y a plusieurs classes dans un bateau, la première classe est la plus prestigieuse. Voyons quel est le taux de survie par classe, sns. barplot permet de générer un diagramme en barres de y(Survived) en fonction de la valeur x (ici Pclass).

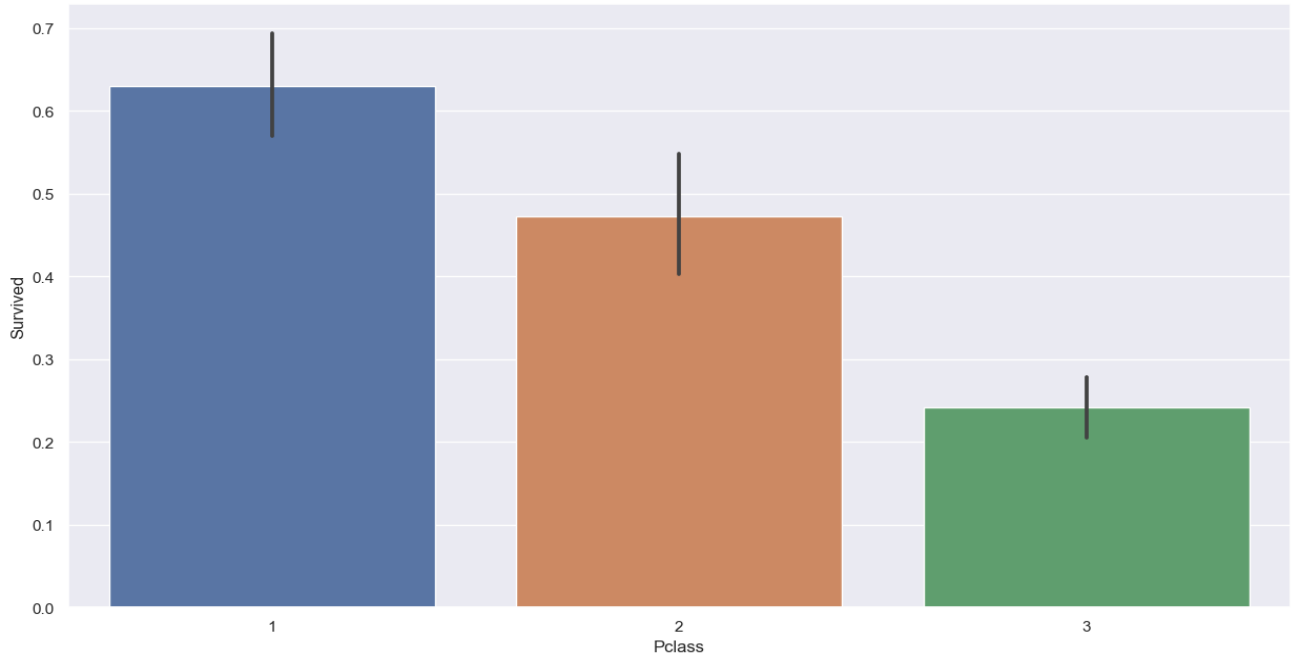
```
# liste des valeurs prises par Pclass
data.Pclass.value_counts()
```

```
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

On peut voir que Pclass prend 3 valeurs, 1,2 ou 3, il n'y a pas de valeur aberrante ni de valeurs manquantes.

```
_ = sns.barplot(x="Pclass",y="Survived", data=data)
```

Figure 2:Repartition des classes de passagers par rapport aux survivants



Visuellement il y a une forte dépendance entre classe et survie. En effet on peut voir que ceux de la classe 1 ont tendance à survivre plus que ceux des autres classes par exemple. Voyons dès à présent qu'el est le niveau de corrélation.

Nous allons juste faire appel à notre fonction de cramer que nous avons créé pour mesurer la corrélation entre nos deux variables qualitatives.

```
print(cramers_v(data['Pclass'], data['Survived']))
```

```
0.33668387622245516
```

On peut voir que le Coefficient de Cramer est de l'ordre de 33% ce qui montre une relation de dépendance entre ces variables.

Voici également visuellement sur un tableau croisé comment se reparties les données.

```
pd.crosstab(data["Survived"],data["Pclass"])
```

Survived \ Pclass	1	2	3
0	80	97	372
1	136	87	119

Au vue dette répartition on peut allons confirmer que cette variable est pertinente donc on peut le conserver.

c. *Exploitation du nom et création de la variable Title(titre)*

Nous allons visualiser le nom et dire les raisons du choix du titre.

```
##Name
data.Name[0]
```

```
'Braund, Mr. Owen Harris'
```

Le nom (champ Name) est constitué du « nom + titre + prénom », et on peut en tirer au moins 1 informations intéressantes : Le titre.

Le choix de ce dernier s'est fait d'abord d'un point de vue subjectif et ensuite par le biais de nos critères. Du point subjectif on peut dire que :

- Le colonel ou le capitaine par exemple à plus de chance de se sauver, grâce à sa formation militaire et sa force par exemple
- De même les petits enfants ou garçons ou filles (Master) ont plus de risque de mourir vu leur age, et leur force physique et mental.
- Plus encore le révérend qui est un homme de Dieu peut par exemple accepter de se sacrifier pour sauver les autres ;

Par ailleurs, nous allons extraire cette variable et faire de traitement et voir si elle vérifie nos critères.

❖ Extraction de la variable

Nous allons extraire le titre de la variable name :

```
#Extraire le titre
data['Title'] = data['Name'].str.split(", ", expand=True)[1].str.split(".", expand=True)[0]
```

Nous allons visualiser alors cette variable :

```
data.Title.value_counts()
```

```
Mr              517
Miss            182
Mrs             125
Master           40
Dr                7
Rev               6
Mlle              2
Major             2
Col               2
the Countess      1
Capt              1
Ms                1
Sir               1
Lady              1
Mme               1
Don               1
Jonkheer          1
Name: Title, dtype: int64
```

❖ Critère du choix de la variable Title

Nous allons croiser cette variable avec la variable survived :

```
pd.crosstab(data.Title,data.Sex)
```

Sex / Title	female	male
Capt	0	1
Col	0	2
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1
the Countess	1	0

Il y a beaucoup de titres peu fréquents, on va fusionner les titres pour lesquels on n'a peu de valeurs, ainsi on les regroupera en 4 catégories : Mr, Mrs, Miss, Master.

```
#Catégorisation de noms
data['Title']=data['Title'].replace(['Major','Don','Jonkheer','Capt','Sir','Rev','Col'], 'Mr')
data['Title']=data['Title'].replace(['Mlle'], 'Miss')
data['Title']=data['Title'].replace(['Lady','the Countess','Mme','Ms','Dona'], 'Mrs')
#repartition selon le sexe
data.loc[(data.Sex== "female") &(data.Title=="Dr"),'Title']="Mrs"
data.loc[(data.Sex== "male") &(data.Title=="Dr"),'Title']="Mr"
data.Title.value_counts()
```

```
Mr        537
Miss      184
Mrs       130
Master     40
Name: Title, dtype: int64
```

On va évaluer le coefficient de corrélation entre les variables .

```
print(cramers_v(data['Title'], data['Survived']))
```

```
0.5711100322192953
```

On peut voir que le coefficient de Cramer est de 0,57 qui est supérieur à 0,5 donc il existe une forte dépendance entre la variable titre et à la variable cible. On la conserve donc.

Pour une analyse plus poussée dans la suite (arbre de décision), nous allons encoder cette variable :

```
encoder = LabelEncoder()
data['Title']= encoder.fit_transform(data.Title)
data['Title']
```

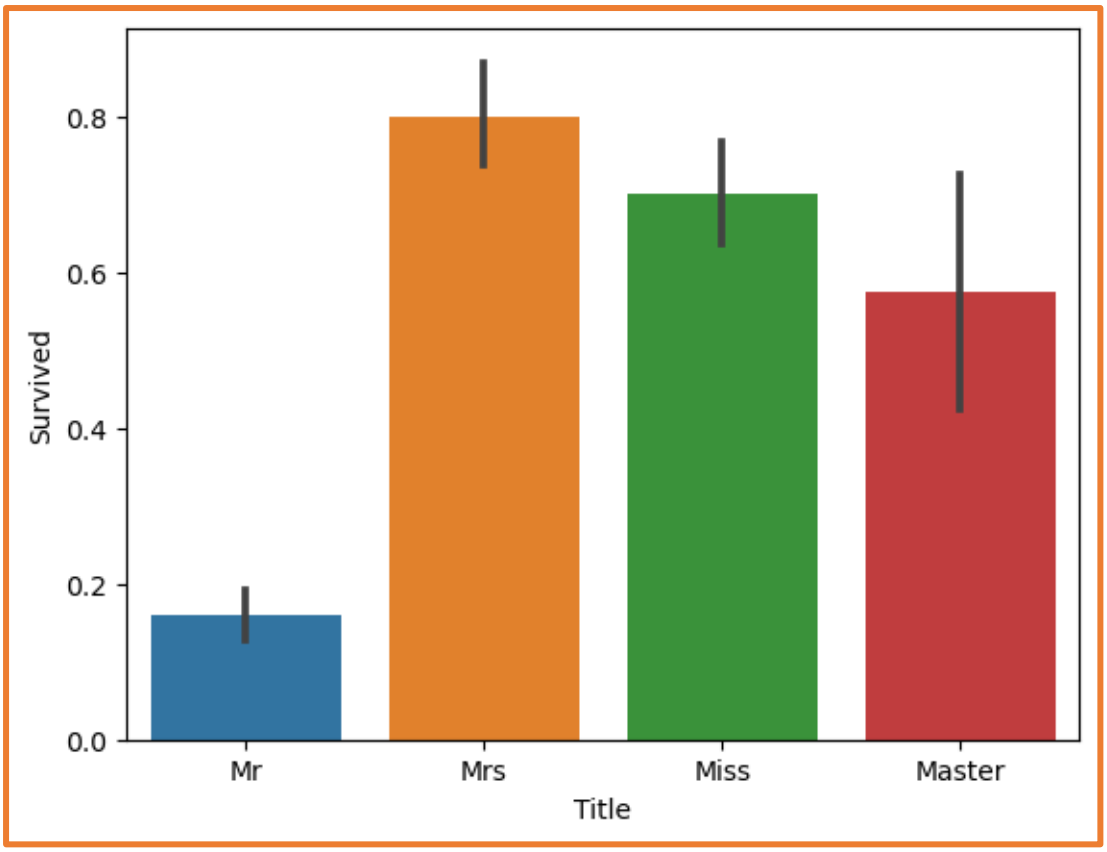
```
0       2
1       3
2       1
3       3
4       2
       ..
886     2
887     1
888     1
889     2
890     2
```

.

Ainsi on a les encodages suivants :

- 1-Miss
- 2-Mister
- 3-Mrs
- 4-Master

On peut aussi voir la répartition du titre par rapport au taux de survie :



On peut voir ici que les Mrs ont un taux de survie plus élevé que les autres.

d. *Variable Sexe (Sex)*

Une vue globale de cette variable nous montre qu'elle ne comporte pas de valeurs manquantes et aberrantes.

```
data.Sex.value_counts()
```

```
male      577
female    314
Name: Sex, dtype: int64
```

Nous allons dès à présent mesurer la corrélation entre les variables

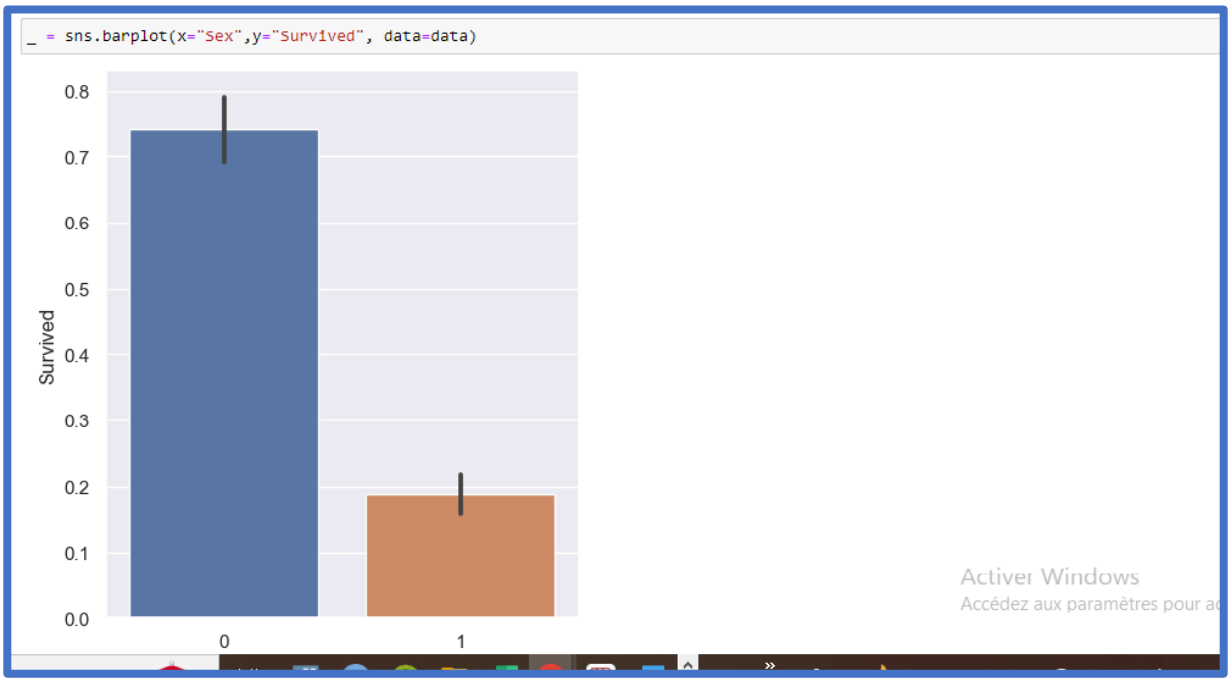
```
print(cramers_v(data['Sex'], data['Survived']))
```

```
0.5401999468101071
```

Donc le Phi de Cramer est supérieur à 0,5 donc il existe une forte dépendance entre la variable Sex et Survived donc on conserve cette variable Sex.
Pour des analyses poussées nous allons encoder la variable Sex.

```
#Sex(encoder)
encoder = LabelEncoder()
data['Sex']= encoder.fit_transform(data.Sex)
data['Sex']#Sex(encoder)
#1-Male
#0-Female
```

On peut également visualiser le nombre de survivant par rapport au sexe :



On peut voir que le taux de survie au niveau des hommes est supérieur au taux de survie au niveau des femmes.

e. *Variable Age*

- Valeurs manquantes

Cette variable rend compte comme le nom l'indique de l'âge des passagers.
Nous allons voir si elle ne comporte pas de valeurs manquantes.

```
data['Age'].isnull().sum()
```

```
177
```

Il manque beaucoup de données, et pas question de supprimer les passagers pour lesquels il manque l'âge. Nous allons donc procéder à une correction de ses valeurs.

❖ Traitement de ces valeurs manquantes

Pour traiter ces valeurs manquantes, nous pouvons opter ^pour un remplacement des valeurs manquantes par la moyenne ou la médiane mais cela pourrait biaiser beaucoup nos analyser car rien ne nous confirme que tous les âges ont les mêmes caractéristiques. Pour ce faire donc, nous avons croiser l'âge avec plusieurs variables de notre base et nous avons pris la décision de corriger cette dernière avec la variable Titre (Title).

-D'abord visualisons la répartition de cette dernière par rapport aux valeurs manquantes de l'âge dans un tableau croisé.

```
pd.crosstab(data.Age.isna(),data.Title)
```

Title Age	Master	Miss	Mr	Mrs
False	36	148	417	113
True	4	36	120	17

On peut voir que Master comporte 4 valeurs manquantes ; Miss : 36 valeurs manquantes ; Mr : 120 valeurs manquantes et Mrs 17 valeurs manquantes.

-Nous allons alors corriger ces valeurs en remplaçant pour chaque catégorie ; les valeurs manquantes par les moyennes correspondantes.

```
#Moyenne Mrs
data.loc[(data.Age.notna())&(data.Title=="Mrs"),'Age'].mean()
#35.92 --36 ans
```

```
35.92035398230089
```

```
#Moyenne Miss
data.loc[(data.Age.notna())&(data.Title=="Miss"),'Age'].mean()
#22 ans
```

```
21.804054054054053
```

```
#Moyenne Mr
data.loc[(data.Age.notna())&(data.Title=="Mr"),'Age'].mean()
#32.94 --33 ans
```

```
32.98441247002398
```

```
#Moyenne Mr
data.loc[(data.Age.notna())&(data.Title=="Master"),'Age'].mean()
  #4.57--5 ans
```

```
4.574166666666667
```

Ainsi nous allons remplacer ces valeurs manquantes par les moyennes des âges des catégories.

```
data.loc[(data.Age.isna())&(data.Title=="Mrs"),'Age']=36
```

```
data.loc[(data.Age.isna())&(data.Title=="Miss"),'Age']=22
```

```
data.loc[(data.Age.isna())&(data.Title=="Mr"),'Age']=33
```

```
data.loc[(data.Age.isna())&(data.Title=="Master"),'Age']= 5
```

```
data['Age'].isna().sum()
```

```
0
```

On peut voir avec le dernier code qu'il n'y a plus de valeurs manquantes on peut allors visualiser notre base et voir si on peut conserver la variable age.

Comme cette dernière présente des valeurs assez éloignées nous allons centrer et réduire cette variable ce qui nous permettra non seulement de faire la suite mais également de faire le test de cramer dessus (Car elle devient qualitative après recodage et les moyennes et variances n'ont plus de sens) :

```
data['Age']=StandardScaler().fit_transform(data['Age'].values.reshape(-1, 1))
```

- Critère du choix de la variable age

Nous allons tous simplement utiliser la valeur de la corrélation de cramer pour voir si l'on peut utiliser cette variable. Nous allons le prendre si le coefficient est supérieur à 0,15.

```
print(cramers_v(data['Age'], data['Survived']))
```

```
0.2707481148791006
```

On trouve 0,27 qui est supérieur à 0,15 donc on peut utiliser cette variable.

f. *Variable SibSp et Parch*

Ces variables rendent compte d'une part des sœurs, frères ('SibSp') et des parents et enfants (Parch). Ces variables ne comportent pas de valeurs manquantes comme nous le montre la ligne suivante ainsi que des valeurs aberrantes. Et elle ont un bon coefficient de corrélation de cramer.

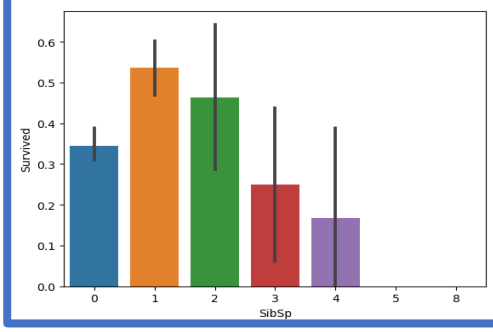
❖ *SibSp*

```
##SibSp
data.SibSp.value_counts()
```

```
0    608
1    209
2     28
4     18
3     16
8      7
5      5
Name: SibSp, dtype: int64
```

```
#Sibp
sns.barplot(x='SibSp',y='Survived',data=data)
```

```
<AxesSubplot:xlabel='SibSp', ylabel='Survived'>
```




```
print(cramers_v(data['SibSp'], data['Survived']))
```

```
0.18742816095927223
```

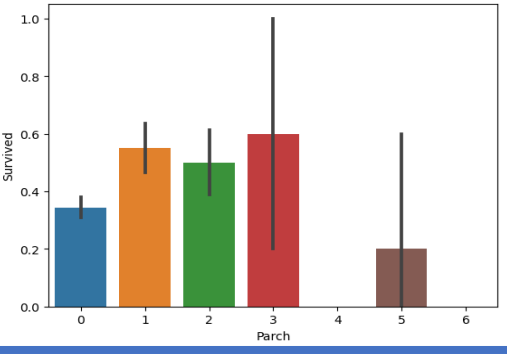
❖ Parch

```
###Parch
data.Parch.value_counts()
```

```
0    678
1    118
2     80
5      5
3      5
4      4
6      1
Name: Parch, dtype: int64
```

```
sns.barplot(x='Parch',y='Survived',data=data)
```

```
<AxesSubplot:xlabel='Parch', ylabel='Survived'>
```




```
print(cramers_v(data['Parch'], data['Survived']))
```

```
0.15693364431605167
```

On peut voir en globalité que leur coefficient de Cramer est supérieur à 0,15 donc on conserve ces variables SibSp et Parch.

g. Variable Embarked

Nous allons visualiser cette variable et voir si elle est pertinente.

```
#Embarked
data.Embarked.value_counts()
```

```
S    644
C    168
Q     77
Name: Embarked, dtype: int64
```

Il n'y a pas de valeurs aberrantes mais 2 valeurs manquantes.

❖ Traitement

Ici on a 2 valeurs manquantes. Le croisement avec la variable « Tickets » pour voir peut-être les personnes ayant les mêmes tickets pour déduire le lieu d'embarcation ; s'avère inutile car seul ces deux passagers aux valeurs manquantes ont ces tickets.

```
data[data.Embarked.isna()]
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
61	62	1	1	Icard, Miss. Amelie	0	38.0	0	0	113572	80.0	B28	NaN
829	830	1	1	Stone, Mrs. George Nelson (Martha Evelyn)	0	62.0	0	0	113572	80.0	B28	NaN

Ainsi nous allons chercher les autres personnes qui possède ce ticket :

```
pd.crosstab(data.Ticket == "113572",data.Embarked)
```

Embarked	C	Q	S
Ticket			
False	168	77	644

On a aucune information c'est à dire personne n'a ces tickets à part eux. Donc pour éviter toute supposition qui pourrait biaiser les résultats et aussi comme les valeurs manquantes ne représentent que 0,2% ce qui est négligeable alors on a décidé alors de le supprimer :

```
##Aucune informarion concernant on drop
data = data.dropna(subset=['Embarked'])
```

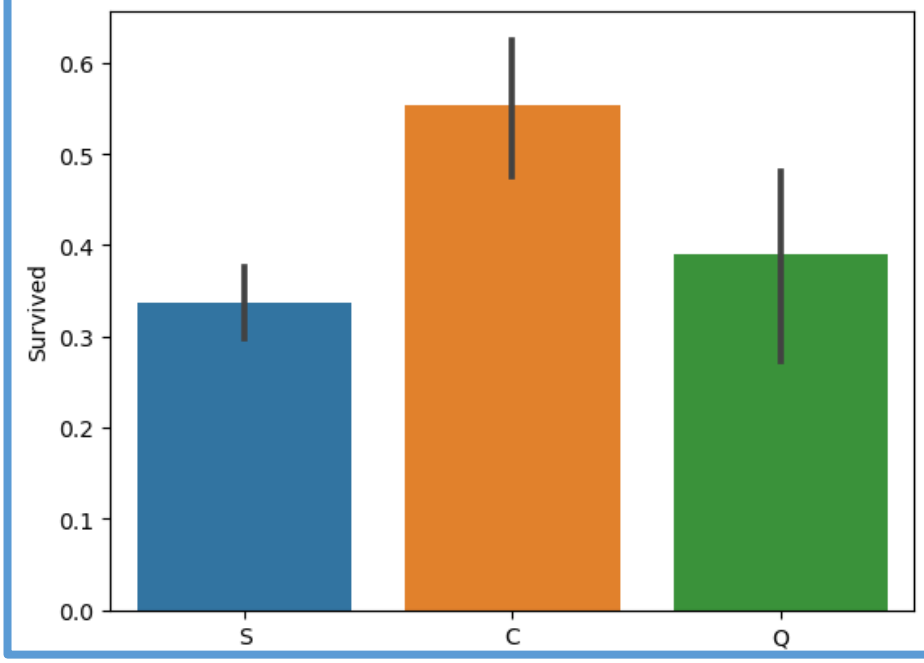
```
##Valeurs manquantes
data.Embarked.isna().sum()
##0
```

```
0
```

❖ Critère du choix

Nous allons voir la répartition du taux de survie par rapport au lieu

```
_ = sns.barplot(x="Embarked",y="Survived", data=data)
```



d'embarcation.

On peut voir que ceux qui descendent au lieu C ont plus de chance de survivre que les autres.

A présent évaluons le coefficient de corrélation entre les variables.

```
print(cramers_v(data['Embarked'], data['Survived']))
```

```
0.16605833339661635
```

On peut alors conserver cette variable. Nous l'encoderons pour la suite du travail.

```
Encoder = LabelEncoder()
data['Embarked'] = encoder.fit_transform(data.Embarked)
print(data['Embarked'])
#2= S; 0=C ; 1=Q
```

```
0      2
1      0
2      2
3      2
4      2
      ..
886    2
887    2
888    2
889    0
890    1
Name: Embarked, Length: 891, dtype: int32
```

h. La variable Fare

Nous allons visualiser cette variable, corriger cette dernière et voir si elles est pertinente.

```
[64] print(cramers_v(data['Embarked'], data['Survived']))
```

```
0.16605833339661635
```

```
##Fare
data.Fare.describe()
```

```
count    889.000000
mean      32.096681
std       49.697504
min        0.000000
25%        7.895800
50%       14.454200
75%       31.000000
max      512.329200
Name: Fare, dtype: float64
```

```
[66] data.Fare.isna().sum()
```

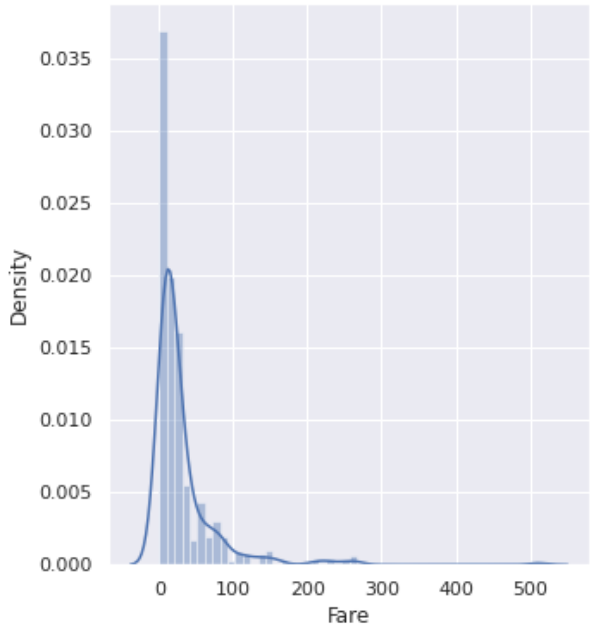
On peut remarquer cette le prix moyen des tickets est de 32 livres environ et il existe de forts écarts entre ces prix à cause de l'écart type qui est élevé soit de l'ordre de 49,69. Les prix varient entre 0 livres et 512,33 livres.

- Traitements

Ce graphique nous montre que les données sont en partie plus regroupées à gauche et moins à droite. Nous allons calculer le coefficient d'asymétrie pour vérifier

```
_= sns.distplot(data.Fare)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
```



✓ ***Calcul de l'asymétrie***

```
scipy.stats.skew(data.Fare.dropna())
#4.8>0 donc assymétrie _données rangées à gauche
```

```
4.793334993307843
```

- Cette ligne de code calcule l'asymétrie (skewness en anglais) de la distribution des valeurs dans la colonne "Fare" du DataFrame "data" en utilisant la fonction skew de la bibliothèque scipy.stats. L'asymétrie mesure l'asymétrie de la distribution de probabilité des données autour de leur moyenne.
- La méthode dropna() est utilisée pour supprimer toutes les valeurs manquantes de la colonne "Fare" avant de calculer l'asymétrie. Cela est nécessaire car la plupart des fonctions statistiques ne peuvent pas être appliquées sur des données manquantes.
- La sortie de cette ligne de code est la valeur d'asymétrie de la distribution de la colonne "Fare". Si la valeur est positive, cela indique que la distribution a une queue de droite plus longue et que la majorité des valeurs se trouvent à gauche de la moyenne. Si la valeur est négative, cela indique que la distribution a une queue de gauche plus longue et que la majorité des valeurs se trouvent à droite de la moyenne. Si la valeur est proche de zéro, cela indique que la distribution est relativement symétrique.

Dans notre cas, la valeur de l'asymétrie est de 4.8 ce qui est positif, donc la majorité de nos données se trouvent à gauche de la moyenne. Pour corriger cela, nous allons normaliser la variable Fare.

```
data['Fare']=StandardScaler().fit_transform(data['Fare'].values.reshape(-1, 1))
```

- La méthode StandardScaler() de la bibliothèque scikit-learn est utilisée pour créer un objet de normalisation qui sera appliqué à la colonne "Fare". La méthode fit_transform() est ensuite utilisée pour ajuster le transformateur sur la colonne "Fare" et pour normaliser les valeurs de cette colonne.
- La méthode reshape(-1,1) est utilisée pour changer la forme de la colonne "Fare" d'un tableau à une dimension en un tableau à deux dimensions, avec une colonne et un nombre de lignes qui s'ajuste automatiquement en fonction de la longueur de la colonne. Cette étape est nécessaire pour que la méthode fit_transform() puisse être appliquée correctement sur la colonne.

- Critere du choix

```
print(cramers_v(data['Fare'], data['Survived']))

0.446323174494435
```

On voit que le V de cramer est largement supérieur à 0.15 donc on prend la variable Fare.

i. *Récapitulatif*

Nous venons de traiter nos variables et avons choisis les 7 variables que nous allons utiliser pour notre classification et les raisons de ce choix. Ces variables ne sont rien d'autre que :

- ✓ Titre (Title) [« extraite de la variable Name]
- ✓ Classe des passagers (Pclass)
- ✓ Age des passagers (Age)
- ✓ Le nombre de frères et Sœurs à bord (SibSp)
- ✓ Le nombre de Parents et enfants à bord (Parch)
- ✓ Le Sexe (Sex)
- ✓ Le lieu d'embarcation (Embarked)
- ✓ Le prix des tickets (Fare)

La question à se poser maintenant est : Pourquoi choisir ces variables et non les autres ?

Nous allons donner les raisons dans cette dernière partie du traitement.

j. *Raison du non choix des autres variables*

Nous allons prendre chaque variable restante et donner les raisons qui seront liés à nos opinions, nos expériences et d'autre même aux chiffres.

- *Nom (Name) et Identifiant des passagers (Passenger Id)*

Ici on a choisi que titre mais on a laissé les noms et prénoms. Pourquoi ?

La raison est que l'on veut prédire la suivie et ce n'est pas parce qu'on t'appelle X ou Y que tu vas survivre. Le nom est indépendant de la survie. Il en est de même pour l'identifiant des passagers

- *Tickets (Ticket)*

Ici simplement, avec le lieu d'embarcation et la classe on peut connaitre ton ticket. C'est le processus qui est peut-être utilisé pour accorder des tickets à la gare

- Cabine (Cabin)

Il y a trop peu de données sur la cabine, par ailleurs la classe du passager définit l'emplacement de sa cabine, pour ne pas générer trop de bruit nous allons laisser cette variable.

Nous allons maintenant implémenter notre modèle de machine Learning.

III. MODELE DE MACHINE LEARNING

Dans cette partie nous allons présenter trois modèles effectués en fonction du score et de l'amélioration. Pour pouvoir trouver le bon modèle nous avons tester 3 modèles différents et le dernier modèle donnait de bonnes prédictions et de bons résultats. Dans cette partie nous allons présenter chaque modèle , expliquer le code , présenter le score et la matrice de confusion.

1. Choix de la variable dépendante et des variables indépendantes et séparation en base test et d'entrainement

Notre variable Target est la variable « Survived » et les descripteurs , les variables pertinentes choisis dans la partie précédente.

```
##Choix de target
Y=data['Survived']

X=data[['Pclass','Sex','SibSp','Parch','Embarked','Title','Age','Fare']]
```

Avant de construire notre modèle, nous allons séparer nos données en 2 parties dont 80% pour l'entrainement et 20% pour le test.

```
#separation
Xtrain,Xtest,ytrain,ytest=train_test_split(X,Y, test_size=0.2, random_state=2)
```

2. Modèles de Machine Learning

a. *TensorFlow*

Ce modèle a été le premier modèle effectué mais elle n'a pas été convainquant car elle a un score faible de 58% et n'arrive pas à prédire les données.

```
# Créer le modèle
model = tf.keras.Sequential([
  tf.keras.layers.Dense(1, input_shape=[8] , activation='sigmoid')
])

# Compiler le modèle
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


#entrainement du modele
model.fit(Xtrain,ytrain,epochs=10)
```

Notre modèle est défini comme une séquence de couches. Dans ce cas, il s'agit d'un modèle avec une seule couche dense (fully connected) qui prend en entrée un vecteur de 8 dimensions et renvoie une sortie de dimension 1. La fonction d'activation utilisée est la fonction sigmoïde, qui renvoie des valeurs comprises entre 0 et 1.

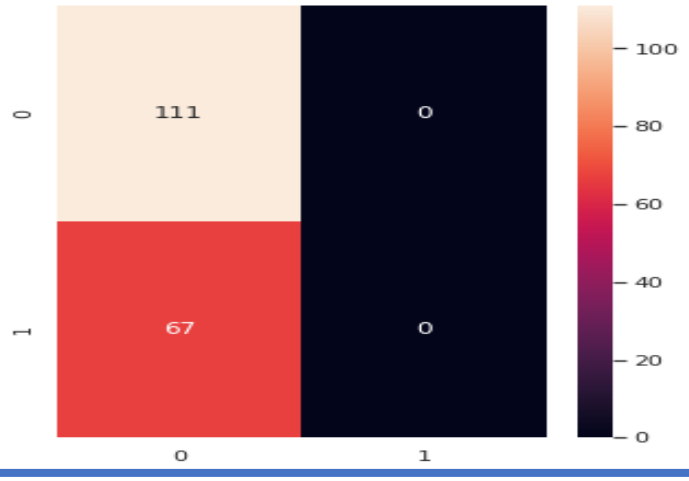
```
#Evaluation du modele sur les données du test
test_loss,test_acc=model.evaluate(Xtest, ytest)
print('Test accuracy', test_acc)
```

```
6/6 [==============================] - 0s 3ms/step - loss: 0.7812 - accuracy: 0.5843
Test accuracy 0.584269642829895
```

En plus du faible score , elle n'arrive pas à faire des prédictions.

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(ytest,predictions)
sns.heatmap(cm,annot=True,fmt='d')
```

```
<AxesSubplot:>
```



On a alors laissé ce modèle car malgré nos multiples méthodes d'amélioration ce modèle peine à bien prédire

b. *Le modèle Logistique*

Nous avons enfin opter pour le modèle logistique et cela a eu un impact car elle produit un bon score et donne de bonnes prédictions .

```
from sklearn.linear_model import LogisticRegression
model2=LogisticRegression()
model2.fit(X,Y)
print(model2.score(X,Y))
#base de test
y_pred_test = model2.predict(Xtest)
print('Accuracy1: {:.2f}'.format(accuracy_score(ytest, y_pred_test)))
#base d'entrainement
y_pred_train = model2.predict(Xtrain)
print('Accuracy2: {:.2f}'.format(accuracy_score(ytrain, y_pred_train)))
```

```
0.8042744656917885
Accuracy1: 0.80
Accuracy2: 0.81
```

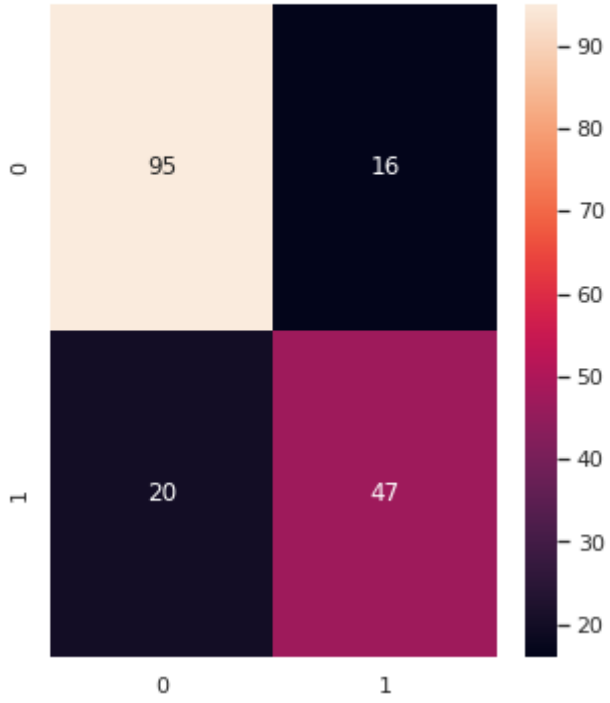
Que fait ce bout de code et que signifie ces résultats ?

- ✓ Ce code utilise la classe LogisticRegression de la bibliothèque Scikit-Learn pour entraîner un modèle de régression logistique sur les données d'entraînement.
- ✓ Le modèle est créé à l'aide de la méthode LogisticRegression(), puis entraîné sur les données d'entraînement en utilisant la méthode fit().
- ✓ La performance du modèle est ensuite évaluée à l'aide de la méthode score(), qui calcule le score de précision moyenne sur les données d'entraînement.
- ✓ Ensuite, le modèle est utilisé pour faire des prédictions sur les données de test, et la précision de ces prédictions est mesurée à l'aide de la méthode accuracy_score(). La même mesure de précision est également calculée pour les données d'entraînement.

On voit également que le score sur l'ensemble des données est de 80,42%, donc notre modèle est performant. Et sur la base de test et d'entrainement, elles sont de l'ordre de 80% et ne sont pas éloignés. Don notre modèle est performant sur la base de test et d'entrainement.

```
cm=confusion_matrix(ytest, y_pred_test)
sns.heatmap(cm,annot=True,fmt='d')
```

```
<AxesSubplot:>
```



Cette classification est obtenue sur la base de test et on peut voir que sur 105 personnes n'ayant pas survécu, le modèle réussi à prédire bien 90% des valeurs et sur 63 personnes ayant survécu elle a réussi à bien prédire environ 75% des valeurs.

c. *Le MLPClassifer*

On cherche toujours à améliorer le modèle d'où le recours à d'autres modèles pour un score plus acceptable. D'où le recours au MLPClassifier. Avec ce modèle , on obtient un score de 89% avec plus de valeurs bien prédites par rapport au modèle logistique.

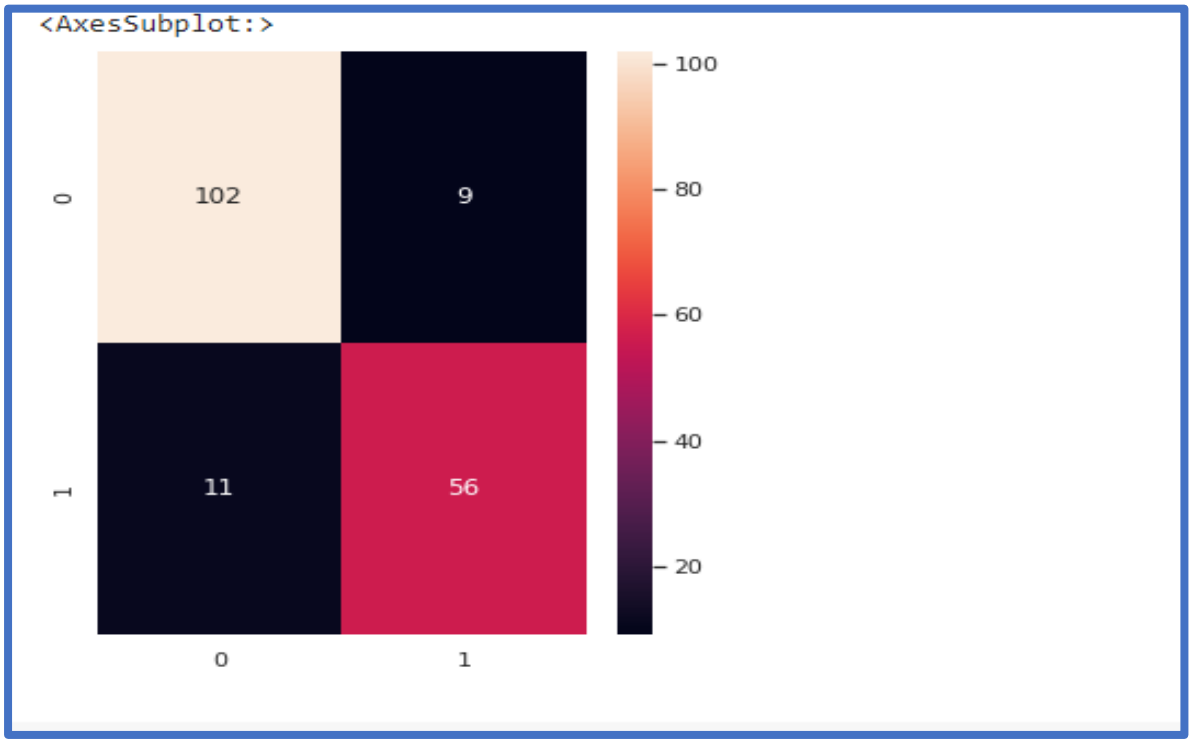
```
#%%MODELE
modele = MLPClassifier(hidden_layer_sizes=(300,500),
                       max_iter = 1000000,activation = 'relu',
                       solver = 'adam')

#%%
modele.fit(X,Y)
print(modele.score(X,Y))

#%%prédiction du modele et score du modele
#base de test
y_pred_test = modele.predict(Xtest)
print('Accuracy_train: {:.2f}'.format(accuracy_score(ytest, y_pred_test)))
#base d'entrainement
y_pred_train = modele.predict(Xtrain)
print('Accuracy_test: {:.2f}'.format(accuracy_score(ytrain, y_pred_train)))
```

- Le modèle est créé à l'aide de la classe MLPClassifier de la bibliothèque Scikit-Learn. La taille des couches cachées est définie par le paramètre hidden_layer_sizes. Le paramètre max_iter détermine le nombre maximum d'itérations de l'optimiseur qui est utilisé pour entraîner le modèle. Le paramètre activation détermine la fonction d'activation utilisée pour les neurones cachés, et le paramètre solver détermine l'algorithme d'optimisation utilisé pour minimiser la fonction de perte du modèle.
- Le modèle est entraîné sur les données d'entraînement à l'aide de la méthode fit(). La performance du modèle est ensuite évaluée à l'aide de la méthode score(), qui calcule le score de précision moyenne sur les données d'entraînement.
- Ensuite, le modèle est utilisé pour faire des prédictions sur les données de test, et la précision de ces prédictions est mesurée à l'aide de la méthode accuracy_score(). La même mesure de précision est également calculée pour les données d'entraînement.

```
0.8920134983127109
Accuracy_train:  0.89
Accuracy_test:  0.89
```

On peut voir que notre modèle est performant car ayant un bon score sur la base de test et la base d'entrainement.

On peut aller loin en visualisant la matrice de confusion de notre modelé afin de voir les classifications.



On peut voir que le modèle prédit plus de bonnes valeurs que les autres modèles.
En résumé , le modèle MLP Classifier est bien approprié pour la modélisation dans notre cas.

CONCLUSION

En définitive, après avoir étudié différents modèles de machine Learning pour la classification de survivants du Titanic, il est clair que le MLP Classifier est le meilleur modèle pour ce projet. Ce modèle a produit les résultats les plus précis avec une précision de plus de 85 %. Le MLP Classifier a été choisi en raison de ses capacités à traiter des données complexes, sa capacité à généraliser les données et à fournir des prédictions précises. Nous avons également effectué une validation croisée pour nous assurer que le modèle était bien généralisé et qu'il n'y avait pas de surajustement.
Dans l'ensemble, ce projet a démontré que le machine Learning peut être utilisé pour résoudre des problèmes complexes, en l'occurrence ici, la classification des survivants du Titanic. Les résultats obtenus peuvent être utilisés pour améliorer la sécurité dans des situations similaires.
En fin de compte, nous sommes confiants que le MLP Classifier est le meilleur modèle pour ce projet et que les résultats obtenus sont fiables et précis. Ce modèle pourrait être utilisé dans d'autres projets de classification de données similaires.

Table des matières