جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

كلية العلوم والتقنيات
FACULTÉ DES SCIENCES ET TECHNIQUES

Master science et technique
Filière : système d'information décisionnel et imagerie
MINI PROJET DATA MINING

Thème

Extraction de la connaissance à partir d'une dataset en utilisons Python

Présenté Par :
-BOUIGADERN ABDELAZIZ

Encadré Par :
-MOHAMED SABIRI

Année Universitaire 2023-2024

I-Introduction :

-Description de dataset :

La base de données Titanic décrit le statut de survie des passagers individuels sur le Titanic.

-Variable descriptions

- Pclass : Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- Survival: Survival (0 = No; 1 = Yes)
- name :Name
- sex :Sex
- age :Age
- sibsp :Number of Siblings/Spouses Aboard
- parch :Number of Parents/Children Aboard
- ticket :Ticket Number
- fare :Passenger Fare (British pound)
- cabin :Cabin
- embarked :Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)
- boat :Lifeboat
- body Body :Identification Number
- home.dest : Home/Destination

-Notes spéciales

- Pclass : est un indicateur du statut socio-économique (SES)

  1er ~ Supérieur; 2ème ~ Moyen; 3ème ~ Basse

- Age : est en années; Fractionnel si l'âge est inférieur à un (1)

  Si l'âge est estimé, il est sous la forme xx.5

- Fare : Le prix est en livres britanniques d'avant 1970 ()

  Facteurs de conversion: 1 = 12s = 240d et 1s = 20d

En ce qui concerne les variables de la relation familiale (c'est-à-dire sibsp et parch), certaines relations :

- **Sibling**: Frère, sœur, du passager à bord du Titanic
- **Spouse**: Mari Ou femme du passager à bord du Titanic
- **Parent**: Mère ou père du passager à bord du Titanic
- **Child**: Fils, Fille, Stepson, ou belle-fille du passager à bord du Titanic

Lecture du fichier :

```
#importation des packages necessaires
from __future__ import division
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#lire dataset
df = pd.read_csv('./titanic.csv')
```

APERÇU DU FICHIER :

```
#afficher les 10 premières  lignes
df.head(10)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

II- PRE-TRAITEMENT :

Dans cette partie consiste à faire quelque modification pour donner une bonne étude statistique :

- **Manipuler les valeurs manquantes :**

En premier on va compter les valeurs manquantes dans chaque colonne par script suivant :

```
####### prétraitement ######
#le nombre des valeurs manquants de chaque colonne
df.isnull().sum()
```

```
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 remarque que les variables (Age, Embarked, Cabin) contient des valeurs manquants, et pour nettoyer notre dataset :

- ✓ on va tout d'abord calculé la moyenne de la variable ' âge', en suite on va remplacer ses valeurs manquantes par la moyenne calculé.
- ✓ Et pour vérifier on a lancé un petit script où on remarque que le nombre des valeurs manquants de la variable Age est égale à 0 parés notre nettoyage.

```
#calculer la moyenne d'age
mn=df['Age'].mean()
#remplacer les valeurs manquants d'age par la moyenne
df['Age'].fillna(mn,inplace=True)
```

```
#afficher les valeurs manquants ( age : 0 valeur manquant )
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

- ✓ Et pour la variable 'Embarked' On va grouper selon la variable en suite on va calculer la fréquence de chaque valeurs de cet variable et on a remplacé nos valeurs manquantes par valeurs la plus fréquente qui est la valeur 'S'.
- ✓ Et on suite on a effectué a lancé un script pour vérifier le bon déroulement de notre nettoyage.

```
#donner la frequence de chaque element de Embarked
df.groupby('Embarked').size()
```

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

```
#remplacer les valeurs manquants d'Embarked par mode ( valeur la plus frequente)
df['Embarked'].fillna('S', inplace=True)
```

```
#valeur manquant Embarked =0
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         0
```

✓ Même principe on traite les valeurs manquantes de variable Cabin par mode.

- **Recherche des valeurs aberrantes :**

Le but de cette partie est convertir les données en écarts-types par rapport à la moyenne pour connaitre les valeurs aberrantes

```
#Recherche de valeurs aberrantes ( convertir les données en écarts-types par rapport à la moyenne)
#stddev = La déviation de Stadard
def getdeviations(x, mean, stddev):
    return abs(x - mean) / stddev
```

```
#remplacer les valeurs manquants de Cabin par mode
df['Cabin'].fillna('C', inplace=True)
```

```
#valeur manquant Cabin =0
df.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       0
dtype: int64
```

```
#les écarts-types par rapport à la moyenne
getdeviations(df['Age'],df['Age'].mean(),df['Age'].std())
```

```
0     5.921480e-01
1     6.384304e-01
2     2.845034e-01
3     4.076970e-01
4     4.076970e-01
5     4.371893e-15
6     1.869009e+00
7     2.130371e+00
8     2.075923e-01
9     1.207437e+00
10    1.976549e+00
11    2.176654e+00
12    7.459703e-01
13    7.153416e-01
14    1.207437e+00
15    1.945920e+00
16    2.130371e+00
17    4.371893e-15
18    1.000524e-01
```

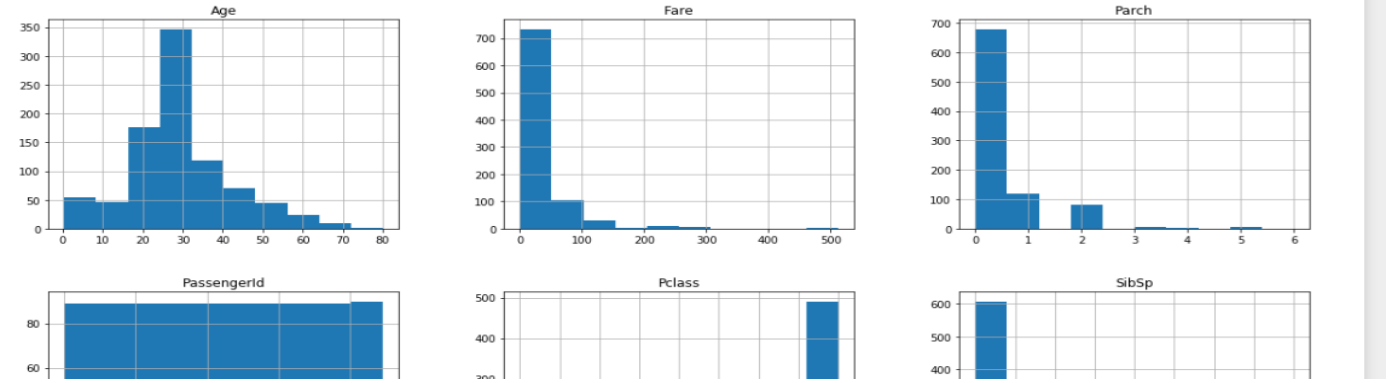
On remarque que tous les observations sont proches entre eux donc on ne possède pas des valeurs aberrants.

III-Analyse du Donnés :

- Représentations graphiques :

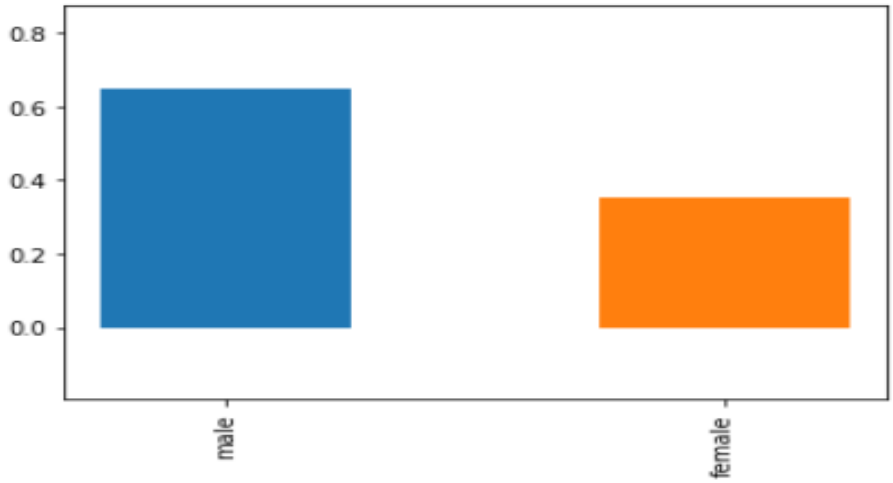
- **Pour variable quantitative (histogramme)**

```
###  représentations graphiques
# variable quantitative
# afficher l'histogramme
_=df.hist(figsize=(20,15))
```



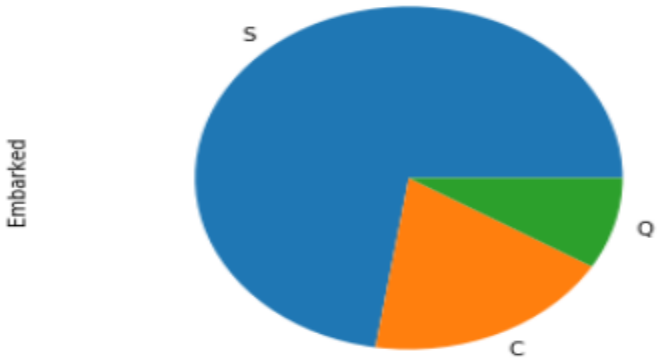
- **Pour variable qualitative**

```
# variable qualitative ( Sex )
# Diagramme
df['Sex'].value_counts(normalize=True).plot(kind='bar')
# Cette ligne assure que le pie chart est un cercle plutôt
plt.axis('equal')
plt.show() # Affiche le graphique
```



On constate que le Sexe masculin est plus fréquent que féminin.

```
# variable qualitative ( Embarked )
# Diagramme en secteurs
df['Embarked'].value_counts(normalize=True).plot(kind='pie')
plt.axis('equal')
plt.show()
```

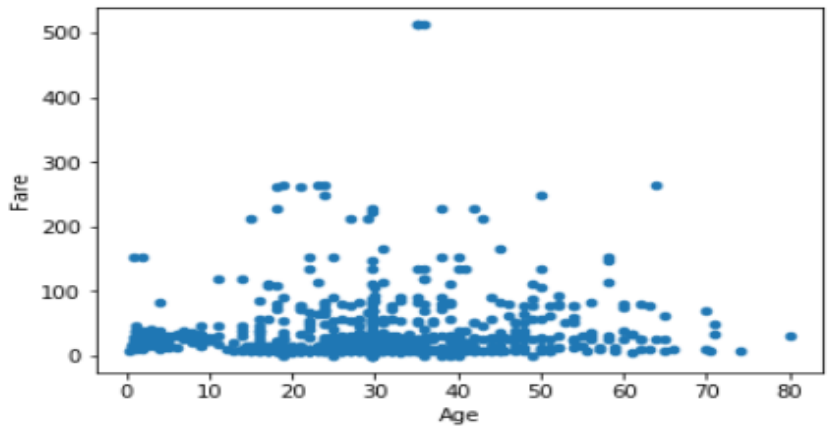


Dans ce diagramme on voit que pour la variable Embarked on trouve la valeur S après la valeur C et enfin la valeur Q

- **Deux variables quantitatives :**

```
#Deux variables quantitatives (un nuage de points (Age,Fare))
df.plot.scatter(x='Age',y='Fare')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20df2b6ea90>
```



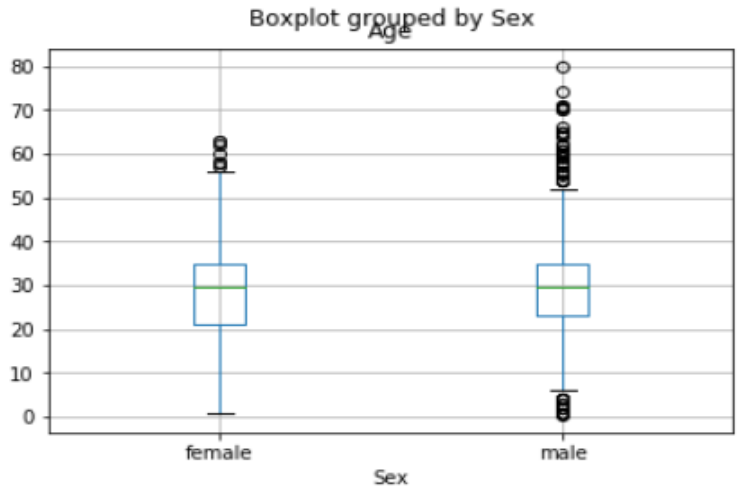
Le nuage de points indique le degré de corrélation entre deux ou plusieurs variables liées. Chaque unité représente un point dans le nuage.

L'abscisse (Age) augmente de gauche à droite tandis que l'ordonnée (Fare) augmente de bas en haut. À mesure que les valeurs augmentent, les points se déplacent de la base inférieure gauche vers la droite inférieure.

- **une variable qualitative et une variable quantitative (*boîte à moustache*)**

```
#une variable qualitative et une variable quantitative (Age,Sex)
df.boxplot(column='Age',by='Sex')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20df2b81668>
```



Dans cette feuille de travail, Age correspond à la variable de graphique et Sex correspond à la variable de catégorie pour le regroupement. Le graphique présente la loi de distribution des Age pour chaque type de Sexe.

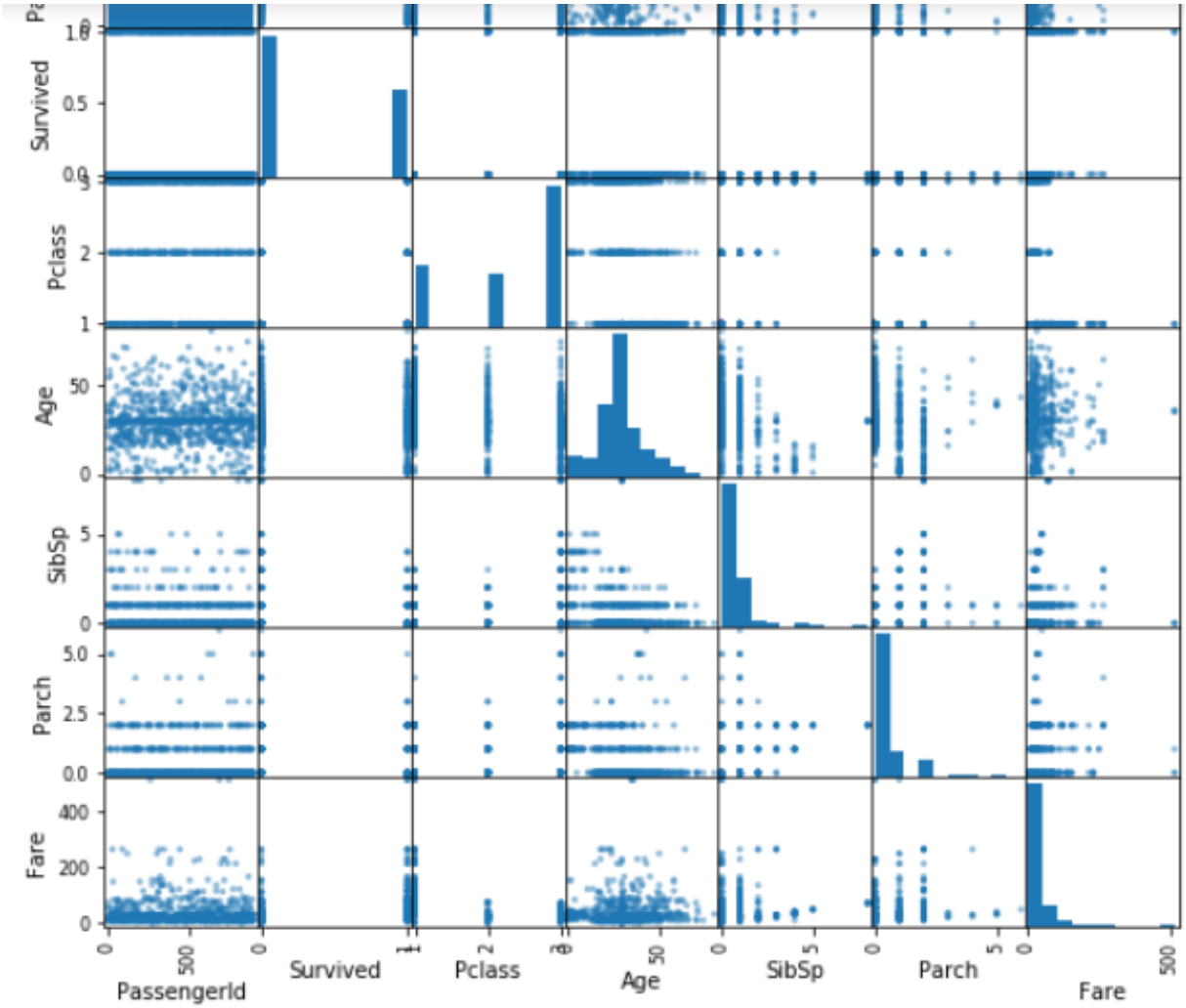
On remarque que les valeurs éloignées qui se situent entre 1,5 et 3 longueurs de boite à partir de la bordure inférieure moins que les valeurs éloignées de la bordure supérieure de la boite.

```
#les statistiques descriptives résumant la tendance centrale,
#la dispersion et la forme de la distribution d'un ensemble de données
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.002015	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.699118	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Comme vous pouvez le voir, nous obtenons pas mal d'informations ici avec juste une invocation rapide de .describe (). Nous obtenons le nombre, qui est combien de lignes nous avons pour chaque colonne. Nous obtenons alors la moyenne, ou la moyenne, de toutes les données dans cette colonne. STD est l'écart type pour chaque colonne. Min est la valeur minimale de cette ligne. 25% correspond à la marque du 25ème centile, et ainsi de suite à 75%. Enfin, nous obtenons max, qui est la valeur la plus élevée pour cette colonne.

- **Corrélation entre deux variables quantitatives**



```
#correlation avec method='kendall'
h1=df.corr(method='kendall', min_periods=1)
h1
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.004090	-0.026824	0.028424	-0.048394	0.000798	-0.008921
Survived	-0.004090	1.000000	-0.323533	-0.032690	0.085915	0.133933	0.266229
Pclass	-0.026824	-0.323533	1.000000	-0.245526	-0.039552	-0.021019	-0.573531
Age	0.028424	-0.032690	-0.245526	1.000000	-0.116728	-0.173968	0.080726
SibSp	-0.048394	0.085915	-0.039552	-0.116728	1.000000	0.425241	0.358262
Parch	0.000798	0.133933	-0.021019	-0.173968	0.425241	1.000000	0.330360
Fare	-0.008921	0.266229	-0.573531	0.080726	0.358262	0.330360	1.000000

Ici, nous obtenons la corrélation de chaque colonne par rapport à l'autre. Comme vous pouvez le voir, nous obtenons un tableau de comparaison. Évidemment, tous les variables ne sont pas tous très étroitement corrélés.

- **Corrélation avec méthode de Spearman**

```
#correlation avec method='spearman'
df.corr(method='spearman', min_periods=1)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.034091	0.041560	-0.061161	0.001235	-0.013975
Survived	-0.005007	1.000000	-0.339668	-0.039109	0.088879	0.138266	0.323736
Pclass	-0.034091	-0.339668	1.000000	-0.308875	-0.043019	-0.022801	-0.688032
Age	0.041560	-0.039109	-0.308875	1.000000	-0.147035	-0.217290	0.118847
SibSp	-0.061161	0.088879	-0.043019	-0.147035	1.000000	0.450014	0.447113
Parch	0.001235	0.138266	-0.022801	-0.217290	0.450014	1.000000	0.410074
Fare	-0.013975	0.323736	-0.688032	0.118847	0.447113	0.410074	1.000000

- **Corrélation avec méthode de pearson**

```
#correlation avec method='pearson'
df.corr(method='pearson', min_periods=1)
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
PassengerId	1.000000	-0.005007	-0.035144	0.033207	-0.057527	-0.001652	0.012658
Survived	-0.005007	1.000000	-0.338481	-0.069809	-0.035322	0.081629	0.257307
Pclass	-0.035144	-0.338481	1.000000	-0.331339	0.083081	0.018443	-0.549500
Age	0.033207	-0.069809	-0.331339	1.000000	-0.232625	-0.179191	0.091566
SibSp	-0.057527	-0.035322	0.083081	-0.232625	1.000000	0.414838	0.159651
Parch	-0.001652	0.081629	0.018443	-0.179191	0.414838	1.000000	0.216225
Fare	0.012658	0.257307	-0.549500	0.091566	0.159651	0.216225	1.000000

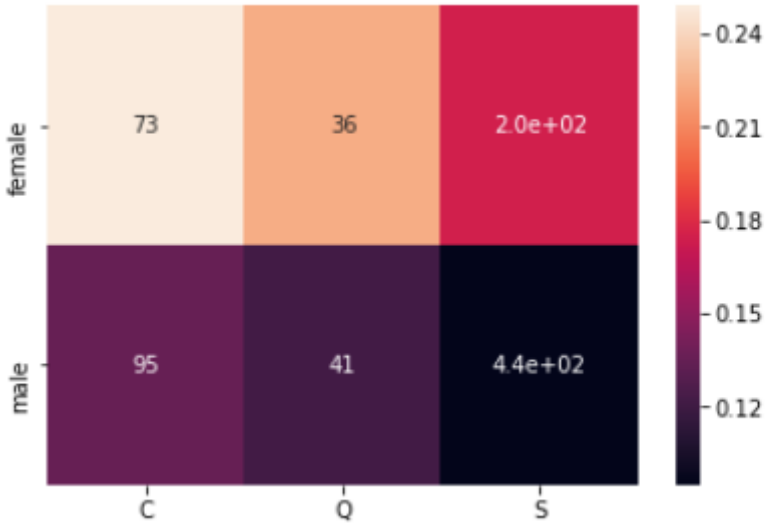
- **Corrélation entre deux variables qualitatives (tableau de contingence)**

```
#le tableau de contingence
X = "Sex"
Y = "Embarked"

c = df[[X,Y]].pivot_table(index=X,columns=Y,aggfunc=len)
cont = c.copy()

tx = df[X].value_counts()
ty = df[Y].value_counts()

cont.loc[:,"Total"] = tx
cont.loc["total",:] = ty
cont.loc["total","Total"] = len(df)
cont
```

Embarked	C	Q	S	Total
Sex				
female	73.0	36.0	205.0	314.0
male	95.0	41.0	441.0	577.0
total	168.0	77.0	646.0	891.0

La fréquence des observations présentant à la fois la caractéristique i pour la variable Sexe, et la caractéristique j pour la variable Embarked.

Apres on ajoute une mesure statistique. (Cette mesure est calculable pour chacune des cases du tableau de contingence.) Appliquer sur cette mesure un seuil au-delà duquel on dira que les 2 variables sont corrélées. (Une contribution au non indépendance)



On
constate d'après le seuil que Sexe féminin et Embarked_C sont corrélés.

III-Extraction de connaissances par « Règles d'association »

Pour faire cette Extraction il faut faire quelque modification pour bien Etablir cette connaissance :

- ✓ Remplacer la colonne de Cabine par la première lettre de son contenu :

```
#remplacer la colonne de Cabine par la premiere lettre de sa contenu
df.loc[-df.Cabin.isnull(),'Cabin'] = df.loc[-df.Cabin.isnull(),'Cabin'].apply( lambda x : x.split()[0][0] )
# remplacer les valeurs manquants de la colonne Cabine par N (NaN)
df['Cabin'].fillna('N', inplace=True)
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N	S

Apres on supprime les colonnes inutiles qui ne contient aucun information important comme PassengerId, Name , Ticket avec les script suivant :

```
#supprimer les colonnes inutiles ( ne contient aucun formation important)
data_pr= df.drop( ['PassengerId', 'Name', 'Ticket'], axis=1 )
data_pr.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Cabin	Embarked
0	0	3	male	22.0	1	0	7.2500	N	S
1	1	1	female	38.0	1	0	71.2833	C	C
2	1	3	female	26.0	0	0	7.9250	N	S
3	1	1	female	35.0	1	0	53.1000	C	S
4	0	3	male	35.0	0	0	8.0500	N	S

- ✓ Convertir les colonnes catégorielles en variables indicatrices (0 ou 1) :

Pour chaque catégorie ajoute une colonne et donner la valeur 1 pour spécifier où il est valide. On fait cette transformation pour les colonnes (Sex, Cabin ,Embarked). Avec ce script :

```
#Convertir les colonnes catégorielles en
#variables indicatrices (pour chaque categorie ajoute une colonne pour specifier ou il est valide (= 1))
data_pr = pd.get_dummies( data_pr, columns=['Sex', 'Cabin', 'Embarked'])
data_pr.head()
```

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Cabin_A	Cabin_B	Cabin_C	Cabin_D	Cabin_E	Cabin_F	Cabin_G	Cabin_N	Cabin
0	0	3	22.0	1	0	7.2500	0	1	0	0	0	0	0	0	0	1	
1	1	1	38.0	1	0	71.2833	1	0	0	0	1	0	0	0	0	0	
2	1	3	26.0	0	0	7.9250	1	0	0	0	0	0	0	0	0	1	
3	1	1	35.0	1	0	53.1000	1	0	0	0	1	0	0	0	0	0	
4	0	3	35.0	0	0	8.0500	0	1	0	0	0	0	0	0	0	1	

- ✓ On traite la variable Survived

On devise cette colonne en deux colonnes une colonne contient Survived et d'autre contient not Survived.

```
import numpy as np
[l,c]=np.shape(X)
#on ajoute deux colonne une colonne contient not survived et l'autre contient Sex_male
z = np.zeros((l,1), dtype='i')
Y=np.append(X,z,axis=1)
z1 = np.zeros((l,1), dtype='i')
Y=np.append(Y,z1,axis=1)
[l1,c1]=np.shape(Y)
```

```
for i in range(0,l1):
    for j in range(0,2):
        if Y[i,j]==1:
            Y[i,j+2]=0
        else:
            Y[i,j+2]=1
```

```
T = Y[:,1].copy()
Y[:,1] = Y[:,2]
Y[:,2] = T
Y
```

Donc après cette transformation on a la table suivant :

Y=['Survived','not_Survived','Sex_female','Sex_male'] et ajoute d'autre colonne pour faire la règle ['Cabine_A' …… , 'Cabine_T','Embarked_C', … ,'Embarked_S']

```
X1= pd.DataFrame(  [ data_pr['Cabin_A'], data_pr['Cabin_B'],data_pr['Cabin_C'],data_pr['Cabin_D'],data_pr['Ca
X1.head()
```

	Cabin_A	Cabin_B	Cabin_C	Cabin_D	Cabin_E	Cabin_F	Cabin_G	Cabin_N	Cabin_T	Embarked_C	Embarked_Q	Embarked_S
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	0	0	1	0	0	0	1
3	0	0	1	0	0	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	1	0	0	0	1

```
print(np.append(Y, X1, axis=1))
```

```
[[0 1 0 ... 0 0 1]
 [1 0 1 ... 1 0 0]
 [1 0 1 ... 0 0 1]
 ...
 [0 1 1 ... 0 0 1]
 [1 0 0 ... 1 0 0]
 [0 1 0 ... 0 1 0]]
```

-on peut calculer le nombre de Survived :

```
#nombre de survived
nbr_sur=0
for i in range(0,1):
    if Y[i,0]==1:
        nbr_sur+=1

print(nbr_sur)
```

```
342
```

- ✓ On utilise maintenant l'algorithme a priori :

L'algorithme Apriori est un algorithme d'exploration de données , dans le domaine de l'apprentissage des règles d'association. Il sert à reconnaître des propriétés qui reviennent fréquemment dans un ensemble de données et d'en déduire une catégorisation. Chaque règle d'association a deux mesures : le « support » et la « confiance ». Le support d'un ensemble d'items est définit comme la fréquence d'apparition simultanée des items figurant dans l'ensemble des données.

On calcule aussi regles_valides ,regles_non_valides, support et confiance de chaque deux variables pour déterminer la règle la plus fréquente.

```python
#algo pour calculer regles_valides et regles_non_valides et support et confiance
from collections import defaultdict
valides= defaultdict(float)
invalides = defaultdict(float)
confiance= defaultdict(float)
support = defaultdict(float)
nombre_occurances = defaultdict(int)
for i in range(0,l1):
    for j in range(0,c1):
        if Y[i,j]==1:
            nombre_occurances[j]+=1
        for k in range(0,c1):
            if j==k:
                continue
            elif Y[i,j]==1:
                if Y[i,k]==1:
                    valides[j,k]+=1
                else:
                    invalides[j,k]+=1
            if nombre_occurances[j]!=0:
                confiance[j,k] = valides[j,k]/nombre_occurances[j]
                support[j,k] = valides[j,k]
```

Donc On peut afficher les regles_valides entre chaque deux variable :

```python
#regles_valides
print(valides)
```

```
defaultdict(<class 'float'>, {(1, 0): 0.0, (1, 2): 81.0, (1, 3): 468.0, (1, 4): 8.0, (1, 5): 12.0, (1, 6): 24.0, (1, 7): 8.0,
(1, 8): 8.0, (1, 9): 5.0, (1, 10): 2.0, (1, 11): 481.0, (1, 12): 1.0, (1, 13): 75.0, (1, 14): 47.0, (1, 15): 427.0, (3, 0): 1
09.0, (3, 1): 468.0, (3, 2): 0.0, (3, 4): 14.0, (3, 5): 20.0, (3, 6): 32.0, (3, 7): 15.0, (3, 8): 17.0, (3, 9): 8.0, (3, 10):
0.0, (3, 11): 470.0, (3, 12): 1.0, (3, 13): 95.0, (3, 14): 41.0, (3, 15): 441.0, (11, 0): 206.0, (11, 1): 481.0, (11, 2): 21
7.0, (11, 3): 470.0, (11, 4): 0.0, (11, 5): 0.0, (11, 6): 0.0, (11, 7): 0.0, (11, 8): 0.0, (11, 9): 0.0, (11, 10): 0.0, (11,
12): 0.0, (11, 13): 99.0, (11, 14): 73.0, (11, 15): 515.0, (15, 0): 219.0, (15, 1): 427.0, (15, 2): 205.0, (15, 3): 441.0, (1
5, 4): 8.0, (15, 5): 25.0, (15, 6): 36.0, (15, 7): 20.0, (15, 8): 26.0, (15, 9): 11.0, (15, 10): 4.0, (15, 11): 515.0, (15, 1
2): 1.0, (15, 13): 0.0, (15, 14): 0.0, (0, 1): 0.0, (0, 2): 233.0, (0, 3): 109.0, (0, 4): 7.0, (0, 5): 35.0, (0, 6): 35.0,
(0, 7): 25.0, (0, 8): 24.0, (0, 9): 8.0, (0, 10): 2.0, (0, 11): 206.0, (0, 12): 0.0, (0, 13): 93.0, (0, 14): 30.0, (0, 15): 2
19.0, (2, 0): 233.0, (2, 1): 81.0, (2, 3): 0.0, (2, 4): 1.0, (2, 5): 27.0, (2, 6): 27.0, (2, 7): 18.0, (2, 8): 15.0, (2, 9):
5.0, (2, 10): 4.0, (2, 11): 217.0, (2, 12): 0.0, (2, 13): 73.0, (2, 14): 36.0, (2, 15): 205.0, (6, 0): 35.0, (6, 1): 24.0,
(6, 2): 27.0, (6, 3): 32.0, (6, 4): 0.0, (6, 5): 0.0, (6, 7): 0.0, (6, 8): 0.0, (6, 9): 0.0, (6, 10): 0.0, (6, 11): 0.0, (6,
12): 0.0, (6, 13): 21.0, (6, 14): 2.0, (6, 15): 36.0, (13, 0): 93.0, (13, 1): 75.0, (13, 2): 73.0, (13, 3): 95.0, (13, 4): 7.
0, (13, 5): 22.0, (13, 6): 21.0, (13, 7): 13.0, (13, 8): 5.0, (13, 9): 1.0, (13, 10): 0.0, (13, 11): 99.0, (13, 12): 0.0, (1
3, 14): 0.0, (13, 15): 0.0, (14, 0): 30.0, (14, 1): 47.0, (14, 2): 36.0, (14, 3): 41.0, (14, 4): 0.0, (14, 5): 0.0, (14, 6):
2.0, (14, 7): 0.0, (14, 8): 1.0, (14, 9): 1.0, (14, 10): 0.0, (14, 11): 73.0, (14, 12): 0.0, (14, 13): 0.0, (14, 15): 0.0,
(8, 0): 24.0, (8, 1): 8.0, (8, 2): 15.0, (8, 3): 17.0, (8, 4): 0.0, (8, 5): 0.0, (8, 6): 0.0, (8, 7): 0.0, (8, 9): 0.0, (8, 1
0): 0.0, (8, 11): 0.0, (8, 12): 0.0, (8, 13): 5.0, (8, 14): 1.0, (8, 15): 26.0, (10, 0): 2.0, (10, 1): 2.0, (10, 2): 4.0, (1
0, 3): 0.0, (10, 4): 0.0, (10, 5): 0.0, (10, 6): 0.0, (10, 7): 0.0, (10, 8): 0.0, (10, 9): 0.0, (10, 11): 0.0, (10, 12): 0.0,
(10, 13): 0.0, (10, 14): 0.0, (10, 15): 4.0, (7, 0): 25.0, (7, 1): 8.0, (7, 2): 18.0, (7, 3): 15.0, (7, 4): 0.0, (7, 5): 0.0,
```

Et le support :

```
#support
print(support)
```

```
defaultdict(<class 'float'>, {(1, 0): 0.0, (1, 2): 81.0, (1, 3): 468.0, (1, 4): 8.0, (1, 5): 12.0, (1, 6): 24.0, (1, 7): 8.0,
(1, 8): 8.0, (1, 9): 5.0, (1, 10): 2.0, (1, 11): 481.0, (1, 12): 1.0, (1, 13): 75.0, (1, 14): 47.0, (1, 15): 427.0, (3, 0): 1
09.0, (3, 1): 468.0, (3, 2): 0.0, (3, 4): 14.0, (3, 5): 20.0, (3, 6): 32.0, (3, 7): 15.0, (3, 8): 17.0, (3, 9): 8.0, (3, 10):
0.0, (3, 11): 470.0, (3, 12): 1.0, (3, 13): 95.0, (3, 14): 41.0, (3, 15): 441.0, (11, 0): 206.0, (11, 1): 481.0, (11, 2): 21
7.0, (11, 3): 470.0, (11, 4): 0.0, (11, 5): 0.0, (11, 6): 0.0, (11, 7): 0.0, (11, 8): 0.0, (11, 9): 0.0, (11, 10): 0.0, (11,
12): 0.0, (11, 13): 99.0, (11, 14): 73.0, (11, 15): 515.0, (15, 0): 219.0, (15, 1): 427.0, (15, 2): 205.0, (15, 3): 441.0, (1
5, 4): 8.0, (15, 5): 25.0, (15, 6): 36.0, (15, 7): 20.0, (15, 8): 26.0, (15, 9): 11.0, (15, 10): 4.0, (15, 11): 515.0, (15, 1
2): 1.0, (15, 13): 0.0, (15, 14): 0.0, (0, 1): 0.0, (0, 2): 233.0, (0, 3): 109.0, (0, 4): 7.0, (0, 5): 35.0, (0, 6): 35.0,
(0, 7): 25.0, (0, 8): 24.0, (0, 9): 8.0, (0, 10): 2.0, (0, 11): 206.0, (0, 12): 0.0, (0, 13): 93.0, (0, 14): 30.0, (0, 15): 2
19.0, (2, 0): 233.0, (2, 1): 81.0, (2, 3): 0.0, (2, 4): 1.0, (2, 5): 27.0, (2, 6): 27.0, (2, 7): 18.0, (2, 8): 15.0, (2, 9):
5.0, (2, 10): 4.0, (2, 11): 217.0, (2, 12): 0.0, (2, 13): 73.0, (2, 14): 36.0, (2, 15): 205.0, (6, 0): 35.0, (6, 1): 24.0,
(6, 2): 27.0, (6, 3): 32.0, (6, 4): 0.0, (6, 5): 0.0, (6, 7): 0.0, (6, 8): 0.0, (6, 9): 0.0, (6, 10): 0.0, (6, 11): 0.0, (6,
12): 0.0, (6, 13): 21.0, (6, 14): 2.0, (6, 15): 36.0, (13, 0): 93.0, (13, 1): 75.0, (13, 2): 73.0, (13, 3): 95.0, (13, 4): 7.
0, (13, 5): 22.0, (13, 6): 21.0, (13, 7): 13.0, (13, 8): 5.0, (13, 9): 1.0, (13, 10): 0.0, (13, 11): 99.0, (13, 12): 0.0, (1
3, 14): 0.0, (13, 15): 0.0, (14, 0): 30.0, (14, 1): 47.0, (14, 2): 36.0, (14, 3): 41.0, (14, 4): 0.0, (14, 5): 0.0, (14, 6):
2.0, (14, 7): 0.0, (14, 8): 1.0, (14, 9): 1.0, (14, 10): 0.0, (14, 11): 73.0, (14, 12): 0.0, (14, 13): 0.0, (14, 15): 0.0,
(8, 0): 24.0, (8, 1): 8.0, (8, 2): 15.0, (8, 3): 17.0, (8, 4): 0.0, (8, 5): 0.0, (8, 6): 0.0, (8, 7): 0.0, (8, 9): 0.0, (8, 1
0): 0.0, (8, 11): 0.0, (8, 12): 0.0, (8, 13): 5.0, (8, 14): 1.0, (8, 15): 26.0, (10, 0): 2.0, (10, 1): 2.0, (10, 2): 4.0, (1
0, 3): 0.0, (10, 4): 0.0, (10, 5): 0.0, (10, 6): 0.0, (10, 7): 0.0, (10, 8): 0.0, (10, 9): 0.0, (10, 11): 0.0, (10, 12): 0.0,
(10, 13): 0.0, (10, 14): 0.0, (10, 15): 4.0, (7, 0): 25.0, (7, 1): 8.0, (7, 2): 18.0, (7, 3): 15.0, (7, 4): 0.0, (7, 5): 0.0,
```

La confiance :

```
#confiance
print(confiance)
```

```
defaultdict(<class 'float'>, {(1, 0): 0.0, (1, 2): 0.14754098360655737, (1, 3): 0.8524590163934426, (1, 4): 0.01457194899817850
6, (1, 5): 0.02185792349726776, (1, 6): 0.04371584699453552, (1, 7): 0.014571948998178506, (1, 8): 0.014571948998178506, (1,
9): 0.009107468123861567, (1, 10): 0.0036429872495446266, (1, 11): 0.8761384335154827, (1, 12): 0.0018214936247723133, (1, 13):
0.1366120218579235, (1, 14): 0.08561020036429873, (1, 15): 0.7777777777777778, (3, 0): 0.18890814558058924, (3, 1): 0.811091854
4194108, (3, 2): 0.0, (3, 4): 0.024263431542461005, (3, 5): 0.03466204506065858, (3, 6): 0.05545927209705372, (3, 7): 0.0259965
33795493933, (3, 8): 0.029462738301559793, (3, 9): 0.01386481802426343, (3, 10): 0.0, (3, 11): 0.8145580589254766, (3, 12): 0.0
017331022530329288, (3, 13): 0.16464471403812825, (3, 14): 0.07105719237435008, (3, 15): 0.7642980935875217, (11, 0): 0.2998544
3959243085, (11, 1): 0.7001455604075691, (11, 2): 0.3158660844250364, (11, 3): 0.6841339155749636, (11, 4): 0.0, (11, 5): 0.0,
(11, 6): 0.0, (11, 7): 0.0, (11, 8): 0.0, (11, 9): 0.0, (11, 10): 0.0, (11, 12): 0.0, (11, 13): 0.14410480349344978, (11, 14):
0.10625909752547306, (11, 15): 0.7496360989810772, (15, 0): 0.33900928792569657, (15, 1): 0.6609907120743034, (15, 2): 0.317337
4613003096, (15, 3): 0.6826625386996904, (15, 4): 0.01238390092879257, (15, 5): 0.03869969040247678, (15, 6): 0.055727554179566
56, (15, 7): 0.030959752321981424, (15, 8): 0.04024767801857585, (15, 9): 0.017027863777089782, (15, 10): 0.006191950464396285,
(15, 11): 0.7972136222910217, (15, 12): 0.0015479876160990713, (15, 13): 0.0, (15, 14): 0.0, (0, 1): 0.0, (0, 2): 0.68128654970
76024, (0, 3): 0.31871345029239767, (0, 4): 0.02046783625730994, (0, 5): 0.1023391812865497, (0, 6): 0.1023391812865497, (0,
7): 0.07309941520467836, (0, 8): 0.07017543859649122, (0, 9): 0.023391812865497075, (0, 10): 0.005847953216374269, (0, 11): 0.6
023391812865497, (0, 12): 0.0, (0, 13): 0.2719298245614035, (0, 14): 0.08771929824561403, (0, 15): 0.6403508771929824, (2, 0):
0.7420382165605095, (2, 1): 0.25796178343949044, (2, 3): 0.0, (2, 4): 0.0031847133757961785, (2, 5): 0.08598726114649681, (2,
6): 0.08598726114649681, (2, 7): 0.05732484076433121, (2, 8): 0.04777070063694268, (2, 9): 0.01592356687898089, (2, 10): 0.0127
38853503184714, (2, 11): 0.6910828025477707, (2, 12): 0.0, (2, 13): 0.23248407643312102, (2, 14): 0.11464968152866242, (2, 15):
0.6528662420382165, (6, 0): 0.5932203389830508, (6, 1): 0.4067796610169492, (6, 2): 0.4576271186440678, (6, 3): 0.5423728813559
322, (6, 4): 0.0, (6, 5): 0.0, (6, 7): 0.0, (6, 8): 0.0, (6, 9): 0.0, (6, 10): 0.0, (6, 11): 0.0, (6, 12): 0.0, (6, 13): 0.3559
322033898305, (6, 14): 0.03389830508474576, (6, 15): 0.6101694915254238, (13, 0): 0.5535714285714286, (13, 1): 0.44642857142857
145, (13, 2): 0.43452380952380953, (13, 3): 0.5654761904761905, (13, 4): 0.041666666666666664, (13, 5): 0.13095238095238096, (1
```

On définit une fonction qui affiche pour chaque prémisse et conclusion en entrée, le Support et la confiance que nous venons de calculer.

Et faire quelque test :

```
#definir la regle
def regle(premise,conclusion):
    print("Si une personne ",col[premise],", elle est de  ",col[conclusion]," avec un support de ",support[premise,conclusion],'
```

```
#test la fonction
regle(1,3)
regle(1,2)
regle(0,10)
regle(1,12)
regle(5,13)
```

```
Si une personne  not_survived , elle est de   Sex_male  avec un support de  468.0  et une confiance de  0.8524590163934426 .
Si une personne  not_survived , elle est de   Sex_female  avec un support de  81.0  et une confiance de  0.14754098360655737 .
Si une personne  survived , elle est de   Cabin_G  avec un support de  2.0  et une confiance de  0.005847953216374269 .
Si une personne  not_survived , elle est de   Cabin_T  avec un support de  1.0  et une confiance de  0.0018214936247723133 .
Si une personne  Cabin_B , elle est de   Embarked_C  avec un support de  22.0  et une confiance de  0.46808510638297873 .
```

Généraliser le programme précèdent pour afficher les 10 premières règles en les triant par support avec ce script :

```
from operator import itemgetter
#trier les regles selon support
tab=sorted(support.items(), key=itemgetter(1), reverse=True)
```

```
for i in range(10):
    print(tab[i])
```

```
((6, 14), 551.0)
((14, 6), 551.0)
((1, 6), 505.0)
((6, 1), 505.0)
((3, 6), 502.0)
((6, 3), 502.0)
((1, 3), 468.0)
((3, 1), 468.0)
((3, 14), 441.0)
```

On remarque que l'indice 6 et 11 et plus fréquente il correspond à Cabin_C et Embarked_S donc la règle est suivant :

« Si une personne Cabin_C, elle est d'Embarked_S avec un support de 551.0 et une confiance de 0.7386058981233244. »

IV-Extraction de connaissances par « Classification Ascendante Hiérarchique ou d'autre type de classification»

Une première approche consiste à utiliser la Classification Ascendante Hiérarchique (CAH). Le graphe qui en résulte permet de se faire une idée visuelle des différents regroupements et d'intuiter le nombre de classes. Les variables étant toutes numériques, nous utiliserons la distance euclidienne comme mesure de dissimilarité et la distance de "Ward" comme mesure de dissimilarité inter-classe (cette distance vise à maximiser l'inertie inter-classe)

-importer les packages nécessaires avant de normaliser le dataset data_pr (cette data qui construit après Convertir les colonnes catégorielles de le dataset en variables indicatrices)

```
import sklearn
import sklearn.preprocessing
#normaliser le dataset data_pr(apres Convertir les colonnes catégorielles de le dataset en variables indicatrices)
scale=sklearn.preprocessing.scale(data_pr)
```

```
print(scale)
```

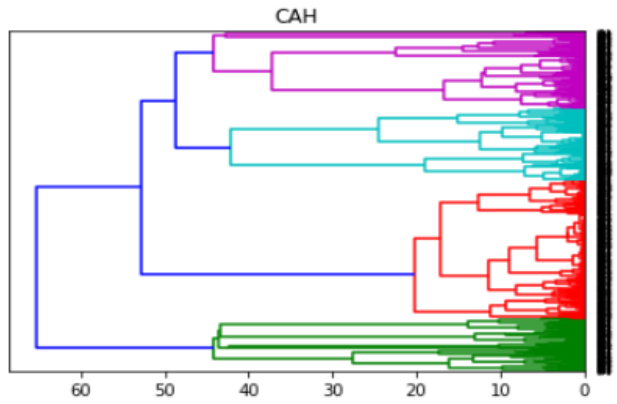
```
[[-0.78927234  0.82737724 -0.5924806  ... -0.48204268 -0.30756234
   0.61583843]
 [ 1.2669898  -1.56610693  0.63878901 ...  2.0745051  -0.30756234
  -1.62380254]
 [ 1.2669898   0.82737724 -0.2846632  ... -0.48204268 -0.30756234
   0.61583843]
 ...
 [-0.78927234  0.82737724  0.         ... -0.48204268 -0.30756234
   0.61583843]
 [ 1.2669898  -1.56610693 -0.2846632  ...  2.0745051  -0.30756234
  -1.62380254]
 [-0.78927234  0.82737724  0.17706291 ... -0.48204268  3.25137334
  -1.62380254]]
```

Apres générer la matrice des liens :

```
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy
#générer la matrice  des liens
z=scipy.cluster.hierarchy.linkage(scale,method='ward',metric='euclidean')
print(z)
```

```
[[ 451.          490.            0.            2.        ]
 [   4.          614.            0.            2.        ]
 [ 126.          196.            0.            2.        ]
 ...
 [1770.         1775.           47.34772911  356.        ]
 [1764.         1778.           50.39694953  703.        ]
 [1777.         1779.           71.72688182  891.        ]]
```

```
from scipy.cluster.hierarchy import dendrogram
plt.title("CAH")
dendrogram(z,orientation='left',color_threshold=45)
plt.show()
```



4 groupes constitués

Nous allons alors afficher la classe obtenue pour chaque observation

```
from scipy.cluster.hierarchy import fcluster
#afficher la classe obtenue pour chaque observation
groupes=fcluster(z,t=45,criterion='distance')
print(groupes)
```

```
[2 3 4 4 2 3 1 4 4 3 4 4 2 4 4 4 4 2 4 3 2 1 3 1 4 4 3 1 3 2 3 1 3 2 3 2 3
 2 4 3 4 4 3 3 3 2 3 3 3 4 4 2 1 4 1 2 4 3 4 4 3 1 2 4 3 3 4 2 4 2 2 4 2 3
 2 4 2 2 2 4 2 2 3 2 4 4 2 2 1 2 2 2 1 2 2 2 1 1 4 2 4 2 1 2 2 2 4 2 2 3 2
 3 2 4 3 2 3 2 1 4 2 2 3 1 1 3 3 2 4 2 3 2 4 4 2 3 1 2 2 1 3 4 4 3 2 2 2 4
 4 2 2 4 2 2 2 3 3 2 2 4 2 4 2 2 4 2 1 4 2 2 1 4 4 2 1 2 2 3 2 2 4 3 4 4 4
 1 3 2 3 2 4 2 4 4 1 1 3 2 3 4 2 4 2 3 2 4 2 3 3 1 2 4 2 2 3 1 4 2 1 2 2 2
 2 2 2 2 2 2 2 2 4 4 2 2 4 2 4 2 4 2 2 3 3 2 2 3 3 4 4 1 2 2 4 2 2 4 3 3 1 1
 4 3 4 1 1 3 2 4 2 4 4 4 2 2 4 3 3 1 4 2 4 4 3 2 2 2 1 3 2 2 2 3 4 1 1 4 2 3
 3 1 2 1 3 3 2 1 2 1 3 3 3 1 3 1 4 2 2 4 4 2 4 1 2 2 3 4 4 3 2 1 4 1 3 2 1
 2 4 2 2 1 2 1 4 1 2 2 2 4 4 4 2 2 2 2 3 2 3 2 1 4 3 3 4 3 3 2 3 2 1 3 3 1
 1 2 2 3 4 3 4 1 3 2 3 3 2 4 2 2 4 4 3 3 1 2 2 1 4 2 4 2 2 4 2 2 4 2 4 2 2
 2 2 4 2 3 3 2 2 4 4 4 2 4 3 3 2 4 2 2 4 4 3 1 2 4 4 2 1 1 4 4 1 2 4 2 2 4
 2 1 4 2 3 2 2 2 3 3 2 3 1 1 4 3 1 2 1 2 2 2 2 2 3 3 2 2 4 1 4 1 2 2 2 4 4
 2 2 4 1 4 4 1 2 2 2 2 2 3 2 3 1 2 1 2 2 3 3 4 1 3 4 2 2 2 3 2 1 3 2 1 4 3
 4 2 1 2 3 1 3 3 4 1 2 2 4 3 3 3 4 4 1 3 2 1 1 4 4 2 3 2 4 3 2 2 3 2 3 3 4]
```

Trier les observations par indice de la classe.

```
#trier les observations par indice de la classe.
nb=np.argsort(groupes)
print(pd.DataFrame(data_pr,groupes[nb]))
```

```
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
1        1     1  38.0     1     0  71.2833         1      0
```

En coupant à une distance d'environ 45, nous pouvons former quatre groupes. Néanmoins on voit bien que pour beaucoup de groupes le regroupement se fait à une distance élevée ce qui inciterait à former plus de groupes (pour avoir plus d'homogénéité). Si nous avions coupé à 40, nous aurions alors 11 groupes (et la encore il faudrait couper plus bas).

Néanmoins la coupure aux alentours des 45 est intéressante. En effet, même si ces groupes n'apparaissent pas parfaitement homogènes sur la figure, si je regarde les ascensions à l'intérieur de ces groupes, je retrouve bien des similarités de difficulté (évaluées selon ma propre expérience) entre les éléments qui le composent.

Nous pouvons à présent essayer d'utiliser une méthode de classification qui n'est pas basée sur des arbres, par exemple les K-Means

K-means

Initialiser une matrice k-means et trier les observations par indice de la classe :

```
from sklearn import cluster
#Initialiser une matrice k-means
kmeans=cluster.KMeans(n_clusters=4)
kmeans.fit(scale)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=4, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
#trier les observations par indice de la classe.
idk=np.argsort(kmeans.labels_)
print(pd.DataFrame(data_pr,groupes[idk]))
```

```
4          0       3  35.0       0       0   8.0500            0          1
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
4          0       3  35.0       0       0   8.0500            0          1
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
3          1       1  35.0       1       0  53.1000            1          0
4          0       3  35.0       0       0   8.0500            0          1
```

On donne le paramètre n_clusters=Nombre de classe trouve dans CAH (4).

Et enfin Afficher la correspondance entre les classes CAH et K-means

```
#la correspondance entre les classes CAH et K-means
pd.crosstab(kmeans.labels_,groupes)
```

col_0	1	2	3	4
row_0				
0	8	363	71	25
1	127	0	0	0
2	4	0	46	170
3	1	0	70	6

On remarque qu'il n y a aucun groupe correspond à k-means et a CAH en même temps, Dans cet exemple chaque méthode possède ses propres groupes.