

Projet

Machine Learning

Telco

 4 DataScience 2

Réalisé par:

BELAIDI Nada


BEN HMIDA Nizar

CHARAABI Khaled

KHECHANA Fares

LAKHAL Abir

MAGHERBI Racha



Compréhension du problème métier	2
problématique:	2
Objectif:	2
Compréhension des données	2
Préparation des données	4
Modélisation	7
1-K Nearest Neighbors	7
2-Arbre de décision :CART	8
3-Random Forest:	11
4- Machine à vecteurs de support SVM:	13
5- Naive Bayes:	13
Conclusion	14

I. Compréhension du problème métier

1. problématique:

Il va sans dire que l'un des problèmes majeurs auquel et souvent confronter n'importe quelle entreprise est celui de satisfaire sa clientèle pour pouvoir la maintenir. pour se faire, elle doit prévoir une stratégie commerciale efficace et rentable

Une telle stratégie doit tenir compte des prévisions des taux de désabonnement et parer à une telle éventualité, du coup la facilité de changer d'opérateur est l'un des défis majeurs auxquels elle doit faire face. Pour fidéliser les clients existants, les organisations doivent améliorer le service-client, hausser la qualité des produits et être en mesure de savoir à l'avance quels clients susceptibles de se désabonner. En prévoyant le taux de désabonnement des clients, les entreprises peuvent immédiatement prendre des mesures pour les fidéliser.

La prédiction peut être effectuée en analysant les données des clients à l'aide de techniques d'exploration.

2. Objectif:

Au cours de notre projet, notre objectif est de tester et trouver l'algorithme d'apprentissage adéquat pour résoudre un problème de machine learning qui consiste à vérifier si le client peut se désabonner de son opérateur Telco ou non.

II. Compréhension des données

```
: #afficher les premiers données  
print(data1.head())
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	
0	7590-VHVEG	Female	0	Yes	No	1	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	
4	9237-HQITU	Female	0	No	No	2	Yes	

	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	
0	No phone service	DSL	No	...	No	
1	No	DSL	Yes	...	Yes	
2	No	DSL	Yes	...	No	
3	No phone service	DSL	Yes	...	Yes	
4	No	Fiber optic	No	...	No	

	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	
0	No	No	No	Month-to-month	Yes	
1	No	No	No	One year	No	
2	No	No	No	Month-to-month	Yes	
3	Yes	No	No	One year	No	
4	No	No	No	Month-to-month	Yes	

	PaymentMethod	MonthlyCharges	TotalCharges	Churn
0	Electronic check	29.85	29.85	No
1	Mailed check	56.95	1889.5	No
2	Mailed check	53.85	108.15	Yes
3	Bank transfer (automatic)	42.30	1840.75	No
4	Electronic check	70.70	151.65	Yes

On peut visualiser ici les 5 premières lignes de notre DataFrame.

```
Entrée [7]: #vérifier qu'il n'ya pas des valeurs manquantes
datal.isna().values.any()
```

```
Out[7]: False
```

```
Entrée [8]: #vérifier qu'il n'ya pas des valeurs manquantes
datal.isna().values.any()
```

```
Out[8]: False
```

```
Entrée [9]: datal.shape
```

```
Out[9]: (7043, 21)
```

```
Entrée [10]: datal["Churn"].unique()
```

```
Out[10]: array(['No', 'Yes'], dtype=object)
```

```
Entrée [11]: datal.isnull().sum().sum()
```

```
Out[11]: 0
```

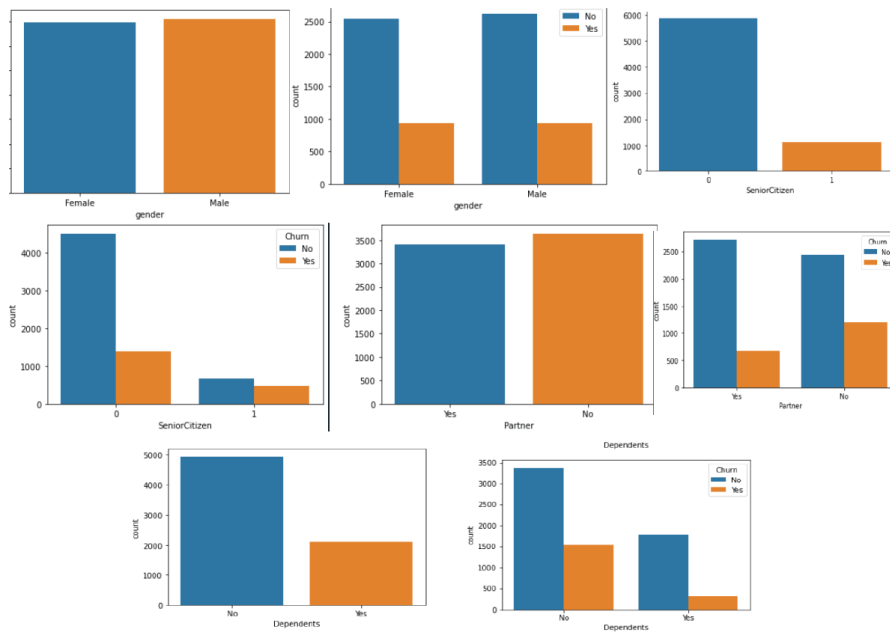
```
Entrée [12]: datal.notnull().values.any()
```

```
Out[12]: True
```

```
Entrée [13]: datal.notnull().sum().sum()
```

```
Out[13]: 147903
```

Ici ,on a testé l'existence des valeurs Nan .Pas de valeurs manquantes ou bien nulles.



Ces représentations graphiques montrent les relations entre le “Churn” et le reste des fonctionnalités .

III. Préparation des données

```
Entrée [3]: df.isnull().sum().sum()
```

```
Out[3]: 0
```

```
Entrée [4]: df.isna().sum().sum()
```

```
Out[4]: 0
```

Pas de valeurs manquantes ou bien nulles, on a pas besoin d'effectuer l'imputation.

```
Entrée [5]: #Remplacez les colonnes de texte par des entiers.  
#Les colonnes ci-dessous incluent des valeurs de texte similaires, je les ai donc modifiées une fois.  
df1=df  
df1.gender = [1 if each == "Male" else 0 for each in df1.gender]
```

```
columns_to_convert = ['Partner',  
                      'Dependents',  
                      'PhoneService',  
                      'MultipleLines',  
                      'OnlineSecurity',  
                      'OnlineBackup',  
                      'DeviceProtection',  
                      'TechSupport',  
                      'StreamingTV',  
                      'StreamingMovies',  
                      'PaperlessBilling',  
                      'Churn']  
  
for item in columns_to_convert:  
    df1[item] = [1 if each == "Yes" else 0 if each == "No" else -1 for each in df1[item]]
```

```
Entrée [6]: #CONTACT  
df1["Contract"].replace(["Month-to-month", "Two year", "One year"], [0, 1, 2], inplace = True)  
df1["PaymentMethod"].unique()
```

```
Out[6]: array(['Electronic check', 'Mailed check', 'Bank transfer (automatic)',  
              'Credit card (automatic)'], dtype=object)
```

```
Entrée [7]: df1["InternetService"].replace(["Fiber optic", "DSL", "No"], [0, 1, 2], inplace = True)
```

```
Entrée [8]: #PAYMENT METHOD  
df1["PaymentMethod"].replace(["Electronic check", "Mailed check", "Bank transfer (automatic)", "Credit card (automatic)"], [0, 1, 2, 3],  
                             < >
```

```
Entrée [10]: pd.to_numeric(df1['MonthlyCharges'])
```

```
Out[10]: 0      29.85  
1      56.95  
2      53.85  
3      42.30  
4      70.70  
...  
7038    84.80  
7039   103.20  
7040    29.60  
7041    74.40  
7042   105.65  
Name: MonthlyCharges, Length: 7043, dtype: float64
```

```
Entrée [27]: del df1['customerID']
```

Ici, on a changé les colonnes non numériques en d'autres à valeurs permettant d'effectuer l'ACP.

La valeur 489 ème de la colonne 'TotalCharges' et un espace qui ne peut pas être transformé en une valeur numérique, on le remplace par la 0.

```
Entrée [11]: df1['TotalCharges'] = df1['TotalCharges'].replace([' '], '0')
```

```
Entrée [13]: pd.to_numeric(df1['MonthlyCharges'])
```

On se retrouve avec cette dataframe:

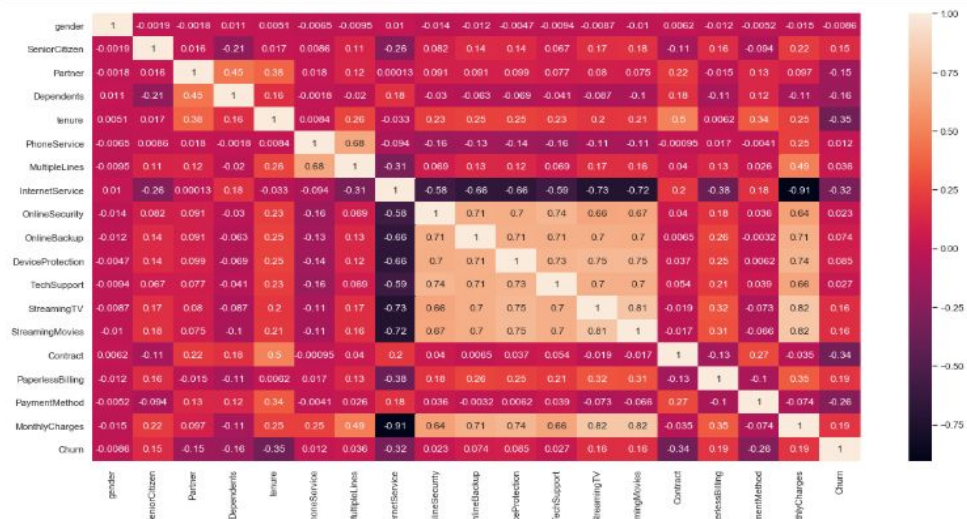
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport
0	0	0	1	0	1	0	-1	1	0	1	0	0
1	1	0	0	0	34	1	0	1	1	0	1	1
2	1	0	0	0	2	1	0	1	1	1	1	0
3	1	0	0	0	46	0	-1	1	1	0	0	1
4	0	0	0	0	2	1	0	0	0	0	0	0
...
7038	1	0	1	1	24	1	1	1	1	0	1	1
7039	0	0	1	1	72	1	1	0	0	1	1	1
7040	0	0	1	1	11	0	-1	1	1	0	0	0
7041	1	1	1	0	4	1	1	0	0	0	0	0
7042	1	0	0	0	66	1	0	0	1	0	1	1

7043 rows x 20 columns

Feature Selection:

On a opté pour une méthode de sélection basée sur la corrélation comme on nous l'a demandé dans le document du projet et on a affiché notre heat map pour bien voir nos résultats.

```
Entrée [23]: def show_correlations(dataframe, show_chart = True):
fig = plt.figure(figsize = (20,10))
corr = dataframe.corr(method='pearson')
if show_chart == True:
sns.heatmap(corr,
xticklabels=corr.columns.values,
yticklabels=corr.columns.values,
annot=True)
return corr
correlation_df = show_correlations(df1,show_chart=True)
```



On remarque que la corrélation maximale qu'on a obtenue est 0.19. Donc, on a choisi d'éliminer les corrélations négatives.

```
Entrée [24]: show_correlations(df1, show_chart=False)["Churn"].sort_values(ascending=False)
```

```
Out[24]: Churn      1.000000
MonthlyCharges  0.193356
PaperlessBilling 0.191825
StreamingTV      0.164673
StreamingMovies  0.163220
SeniorCitizen    0.150889
DeviceProtection 0.084654
OnlineBackup     0.074205
MultipleLines    0.036310
TechSupport      0.027037
OnlineSecurity   0.023309
PhoneService     0.011942
gender           -0.008612
Partner          -0.150448
Dependents       -0.164221
PaymentMethod    -0.262818
InternetService  -0.316846
Contract         -0.341504
tenure           -0.352229
Name: Churn, dtype: float64

<Figure size 1440x720 with 0 Axes>
```

```
Entrée [49]: df2=df1
del df2['customerID']
del df2['tenure']
del df2['Contract']
del df2['InternetService']
del df2['PaymentMethod']
del df2['Dependents']
del df2['Partner']
del df2['gender']
```

On passe maintenant à l'application de l'ACP:

```
Entrée [51]: #feature extraction(ACP)
from sklearn.decomposition import PCA
# define a matrix

array = df2.values
X = array[:,0:13]

X
```

```
Out[51]: array([[0, 0, -1, ..., 29.85, '29.85', 0],
 [0, 1, 0, ..., 56.95, '1889.5', 0],
 [0, 1, 0, ..., 53.85, '108.15', 1],
 ...,
 [0, 0, -1, ..., 29.6, '346.45', 0],
 [1, 1, 1, ..., 74.4, '306.6', 1],
 [0, 1, 0, ..., 105.65, '6844.5', 0]], dtype=object)
```

```
Entrée [52]: # create the PCA instance
pca = PCA(13)
pca
```

```
Out[52]: PCA(copy=True, iterated_power='auto', n_components=13, random_state=None,
          svd_solver='auto', tol=0.0, whiten=False)
```

```
Entrée [53]: pca.fit(X)
```

```
Out[53]: PCA(copy=True, iterated_power='auto', n_components=13, random_state=None,
          svd_solver='auto', tol=0.0, whiten=False)
```

```
Entrée [54]: #Les composantes principale: Vecteurs propres
C=pca.components_
print(C)

#Proportions de variance associées aux axes: Les valeurs propres
propvar=pca.explained_variance_
print(propvar)

A = pca.transform(X)
print(A)
```

Et voici une partie de ce qu'on a obtenu:


```
print(A)
```

[1.67524404e-05	1.47731901e-05	1.16851345e-04	1.50331434e-04
	1.74994548e-04	1.77595484e-04	1.54684204e-04	1.78729004e-04
	1.80024221e-04	3.43849527e-05	8.64441407e-03	9.99962541e-01
	-3.86240011e-05]			
[3.25307468e-03	2.94971916e-03	8.22056602e-03	1.30967698e-02
	1.53702049e-02	1.62612735e-02	1.38465603e-02	2.05997905e-02
	2.04588566e-02	7.05454133e-03	9.99006616e-01	-8.65418831e-03
	8.21087710e-03]			
[-2.20627433e-02	-2.23942581e-01	-4.34537883e-01	4.10827261e-01
	3.58768151e-01	3.60795481e-01	4.13058293e-01	2.80859706e-01
	2.86184635e-01	9.17046726e-03	-2.97277320e-02	-4.39808478e-05
	-2.24492942e-02]			
[1.70660669e-01	-1.75575503e-01	-4.22823356e-01	-4.13542863e-01
	-1.64664698e-01	-3.18704238e-02	-2.85137671e-01	2.51379342e-01
	2.45523917e-01	5.18597443e-01	-4.48824373e-04	9.82988335e-05
	2.97321583e-01]			
[-9.36540808e-02	-6.40978317e-02	-3.54644892e-01	-2.17055376e-01
	-4.00057947e-01	5.59655676e-02	-1.17557177e-01	1.56669253e-01
	1.81783952e-01	-7.56932294e-01	1.21886225e-02	1.16509490e-05
	-8.09347237e-02]			
[2.38719465e-01	-4.87535891e-02	-1.52828886e-01	-1.11839119e-01
	7.20663185e-01	-1.78520020e-01	-3.53036368e-01	-1.09259628e-01
	-1.18611386e-01	-3.43225041e-01	3.51012212e-03	2.52009233e-05
	2.91701678e-01]			
[2.69033976e-01	-8.76767121e-03	3.44037977e-01	8.17266109e-02
	-1.81949910e-01	6.22280324e-01	-5.38826185e-02	-5.36016696e-02
	4.24210570e-02	-1.31667475e-01	-1.50799099e-02	3.27033936e-05
	5.97514985e-01]			
[2.11203711e-02	-1.65583095e-02	-1.12659292e-01	5.77516284e-01
	-2.79838501e-01	-5.78676391e-01	3.59503529e-02	1.47186067e-02
	4.24000105e-02	-5.29411020e-02	1.79127040e-03	6.71654213e-05
	4.83501630e-01]			
[2.40457007e-01	8.66030720e-03	2.33705417e-01	4.86600307e-01

IV. Modélisation

1-K Nearest Neighbors

En appliquant le KNN avec une plage allant de 1 vers 40 On a obtenu comme suit le graph donnant le taux d'erreur pour les différentes valeurs de K.

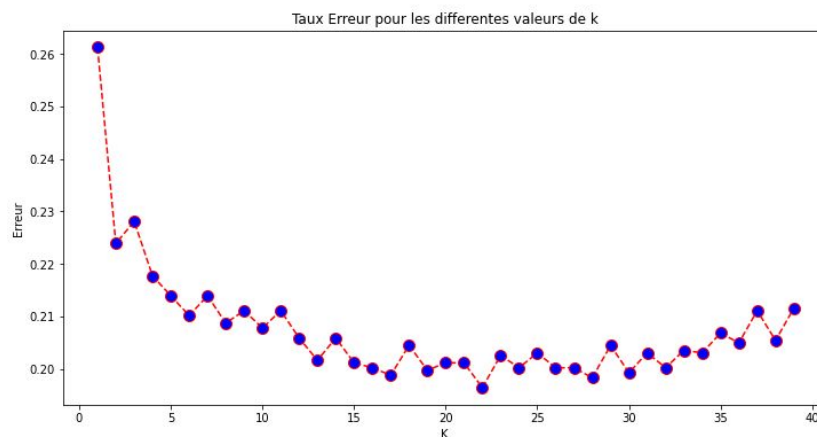
On a choisi la valeur de K la plus petite, c.à.d ayant le taux d'erreur le plus petit d'où on a choisit K=23.

Knearest neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
```

```
error = []
# Calculer l'erreur pour k entre 1 et 40
# Pour chaque itération, l'erreur moyenne pour les valeurs prédites
# de l'ensemble de test est calculée et sauvegardée ds la liste Erreur.
for i in range(1, 40):
    knn = KNeighborsClassifier(i)
    knn_model = knn.fit(X_train, y_train)
    pred_i = knn_model.predict(X_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Taux Erreur pour les différentes valeurs de k')
plt.xlabel('K')
plt.ylabel('Erreur')
```

Text(0, 0.5, 'Erreur')



On passant maintenant notre train et test data frames on a obtenu comme suit des résultats de **80% accuracy** pour le **training set** et **80% accuracy** pour le **test set**

```
: knn = KNeighborsClassifier(23)
knn_model = knn.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)

: print('Accuracy of K-NN classifier on training set: {:.2f}'
      .format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
      .format(knn.score(X_test, y_test)))
```

```
Accuracy of K-NN classifier on training set: 0.80
Accuracy of K-NN classifier on test set: 0.80
```

2-Arbre de décision :CART

On a divisé l'ensemble des observations X et l'ensemble de classe Y, chacun en deux sous-ensemble :

- un sous-ensemble d'apprentissage : 70% de l'ensemble initial
- un sous-ensemble de test : 30% de l'ensemble initial

```
Entrée [40]: from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=0)
```

```
Entrée [41]: classifier = tree.DecisionTreeClassifier()
classifier.fit(X_train, y_train)
```

```
Out[41]: DecisionTreeClassifier()
```

```
Entrée [42]: from sklearn.metrics import accuracy_score
```

```
Entrée [43]: Y_pred = classifier.predict(X_test)
```

```
Entrée [44]: accuracy_score(y_test, Y_pred)
```

```
Out[44]: 0.7330809275911027
```

```
Entrée [45]: confusion_matrix(y_test, Y_pred)
```

```
Out[45]: array([[1272, 288],
               [ 276, 277]])
```

```
Entrée [46]: print(classification_report(y_test, Y_pred))
```

	precision	recall	f1-score	support
0	0.82	0.82	0.82	1560
1	0.49	0.50	0.50	553
accuracy			0.73	2113
macro avg	0.66	0.66	0.66	2113
weighted avg	0.73	0.73	0.73	2113

l'application de l'arbre de décision a donné une performance plus ou moins bonne 0.73 , concernant les précisions on remarque une très bonne prédiction pour les personnes qui ne vont se désabonner (0.82) , par contre il y a une baisse de précision pour les personnes qui vont se désabonner 0.49 (à améliorer)

L'arbre est trop long et difficile à interpréter. Pour faire face à ce problème on va essayer de régler les hyperparamètre de notre modèle

```
Entrée [41]: from sklearn.model_selection import GridSearchCV
```

```
Entrée [42]: grid = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5)
```

```
Entrée [43]: grid.fit(X_train, y_train)
```

```
Out[43]: GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=1),
                    param_grid={'criterion': ['gini', 'entropy'],
                                'max_depth': array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])})
```

```
Entrée [45]: grid.best_params_
```

```
Out[45]: {'criterion': 'gini', 'max_depth': 3}
```

```
Entrée [46]: final_model = DecisionTreeClassifier(random_state=0, criterion='gini', max_depth=3 )
```

```
Entrée [47]: final_model.fit(X_train, y_train)
```

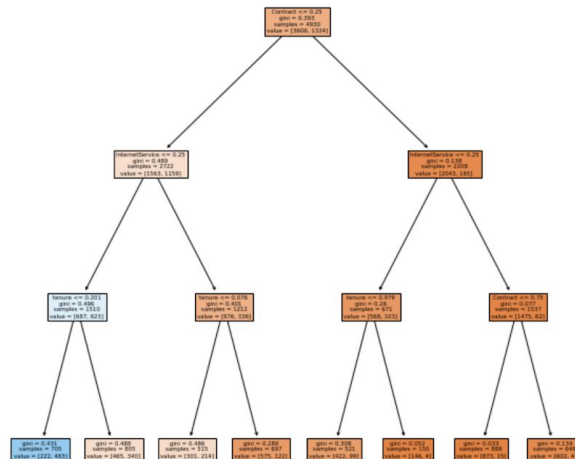
```
Out[47]: DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
Entrée [48]: print('train score = ', final_model.score(X_train, y_train))
print('test score = ', final_model.score(X_test, y_test))
```

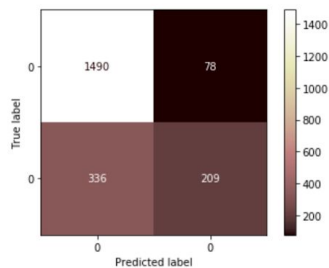
```
train score = 0.7843813387423935
test score = 0.804070042593469
```

```
Entrée [49]: importance = dt.feature_importances_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()
```

```
Entrée [54]: import matplotlib.pyplot as plt
plt.figure(figsize=(10,10))
plot_tree(final_model,feature_names = list(data1.columns[:-1]),filled=True)
plt.show()
```



```
Entrée [51]: from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(final_model, X_test, y_test,display_labels=data1.Churn,cmap=plt.cm.pink)
plt.show()
```



```
Entrée [52]: ypred = final_model.predict(X_test)
print(classification_report(y_test, ypred))
```

	precision	recall	f1-score	support
0	0.82	0.95	0.88	1568
1	0.73	0.38	0.50	545
accuracy			0.80	2113
macro avg	0.77	0.67	0.69	2113
weighted avg	0.79	0.80	0.78	2113

```
Entrée [65]: from sklearn.tree import export_text
print(export_text(final_model))
```

```

--- feature_11 <= 0.25
|--- feature_4 <= 0.25
|   |--- feature_3 <= 0.20
|   |   |--- class: 1
|   |   |--- feature_3 > 0.20
|   |   |--- class: 0
|   |--- feature_4 > 0.25
|   |   |--- feature_3 <= 0.08
|   |   |--- class: 0
|   |   |--- feature_3 > 0.08
|   |   |--- class: 0
|--- feature_4 > 0.25
|   |--- feature_3 <= 0.98
|   |   |--- class: 0
|   |   |--- feature_3 > 0.98
|   |   |--- class: 0
|   |--- feature_4 > 0.25
|   |   |--- feature_11 <= 0.75
|   |   |   |--- class: 0
|   |   |   |--- feature_11 > 0.75
|   |   |   |--- class: 0
|   |   |--- class: 0
|   |--- class: 0
|--- class: 0
  
```

Après application de SearchGridCV qui nous a aidé à déterminer les meilleures valeurs des hyperparamètres réglés et après évaluation de la performance de final_model sur le sous-ensemble de données approprié on a trouvé 0.82 pour les personnes qui ne vont se désabonner et 0.73 pour les personnes qui vont se désabonner avec une performance de 0.80

train_score	test_score
78%	80%

3-Random Forest:

```
Entrée [56]: data["Churn"] = data["Churn"].astype(int)

Y_train = data["Churn"]
X_train = data.drop(labels = ["Churn"],axis = 1)
```

```
Entrée [57]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score,GridSearchCV

Rfcf = RandomForestClassifier(random_state=3)
Rfcf.fit(X_train, Y_train)
```

```
Out[57]: RandomForestClassifier(random_state=3)
```

```
Entrée [58]: clf_score = cross_val_score(Rfcf, X_train, Y_train, cv=5)
print(clf_score)
clf_score.mean()
```

```
[0.79134138 0.78353442 0.76579134 0.78196023 0.78196023]
```

```
Out[58]: 0.7809175188721854
```

```
Entrée [80]: from sklearn.metrics import classification_report
print(classification_report( Y_train, y_pred))
```

```

              precision    recall  f1-score   support

     0       0.86         0.87         0.86         5174
     1       0.63         0.61         0.62         1869

 accuracy          0.80         0.80         0.80         7043
 macro avg          0.74         0.74         0.74         7043
 weighted avg          0.80         0.80         0.80         7043
```

```
Entrée [44]: from sklearn.ensemble import RandomForestClassifier
Rf=RandomForestClassifier(n_estimators=10).fit(X_train, y_train)
y_pred_RF=Rf.predict(X_test)
```

```
Entrée [46]: print("confusion matrix : ")
print(confusion_matrix(y_test, y_pred_RF))
print(classification_report(y_test, y_pred_RF))
print("Accuracy score is : ", accuracy_score(y_test, y_pred_RF))
```

```
confusion matrix :
[[1404  164]
 [ 289  256]]
      precision    recall  f1-score   support

      0       0.83       0.90       0.86       1568
      1       0.61       0.47       0.53        545

   accuracy       0.79       2113
  macro avg       0.72       2113
weighted avg       0.77       2113

Accuracy score is : 0.7856128726928537
```

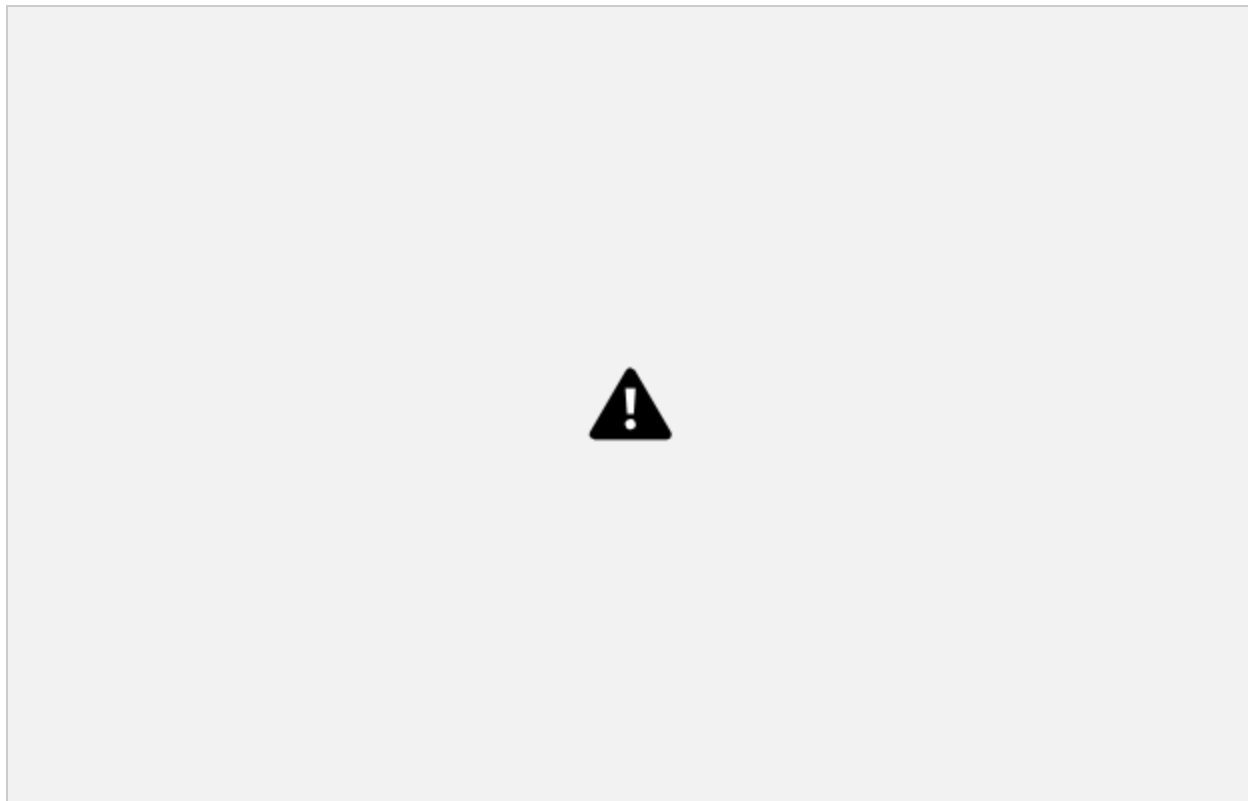
```
Entrée [48]: print('train score = ', Rf.score(X_train, y_train))
print('test_score = ', Rf.score(X_test, y_test))
```

```
train score = 0.9572008113590263
test_score = 0.7856128726928537
```

Après traitement de l’algorithme Random Forest, il nous donné une performance de 0.78 , concernant les précisions on remarque une très bonne prédiction pour les personnes qui ne vont se désabonner (0.83) , pour les personnes qui vont se désabonner 0.61

train_score	test_score
95%	78%

4- Machine à vecteurs de support SVM:



```
[125]: print('Accuracy of SVM classifier on training set: {:.2f}'  
        .format(svclassifier.score(X_train, y_train)))  
        print('Accuracy of SVM classifier on test set: {:.2f}'  
              .format(svclassifier.score(X_test, y_test)))
```

```
Accuracy of SVM classifier on training set: 0.90  
Accuracy of SVM classifier on test set: 0.74
```

l'algorithme Random SVM, il nous donné une performance de 0.80 , concernant les précisions on remarque une très bonne prédiction pour les personnes qui ne vont se désabonner (0.84) , pour les personnes qui vont se désabonner 0.65

train_score	test_score
90%	74%

5- Naive Bayes:

On a commencé par diviser notre dataset en deux parties:

-Partie entraînement (75%) .

-Partie test (25%) .

```
Entrée [57]: X= df2.iloc[:,1:12].values
             y= df2['Churn']

Entrée [58]: from sklearn.model_selection import train_test_split
             X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, random_state=0)
```

Puis on a appliqué notre modèle sur ces deux derniers.

```
Entrée [59]: import mpmath
             from sklearn import datasets
             from sklearn import metrics
             from sklearn.naive_bayes import GaussianNB

Entrée [60]: model = GaussianNB()
             model.fit(X_train, y_train)

Out[60]: GaussianNB(priors=None, var_smoothing=1e-09)

Entrée [61]: expected = y_val
             predicted = model.predict(X_val)
```

Notre modèle nous a donné les résultats suivants:

```
Entrée [62]: metrics.classification_report(expected, predicted)

Out[62]: '
precision    recall  f1-score   support\n
0.48      0.72      0.58      463\n
0.69      1761\nweighted avg          0.78      0.72      0.74      1761\n'

Entrée [63]: metrics.confusion_matrix(expected, predicted)

Out[63]: array([[941, 357],
               [129, 334]], dtype=int64)

Entrée [64]: from sklearn.metrics import accuracy_score

Entrée [65]: accuracy_score(y_val, predicted)

Out[65]: 0.7240204429301533
```

V. Conclusion

La finalité et l'objectif de cet ensemble des processus qu'on a effectué dans ce projet est d'avoir les plus grands pourcentages possible. En appliquant plusieurs algorithmes différents après avoir passé par les étapes de la méthodologie CRISP on a trouvé que le Classifieur **K Nearest Neighbors (KNN)** nous a donné la plus haute accuracy avec **80% pour le Test set est aussi pour le train set**. Ce résultat n'est pas définitif et peut encore être amélioré.