Ecole Supérieure Privée
d'Ingénierie et de Technologies

Rapport Projet Python

Réalisé par :

Mohamed Iheb Bousnina

Montassar Thabti

Fedi Bayoudh

Wissal Bayoudh

Khalil Ben romdhane

Mhadheb Ben Mahmoud

Mustapha Bouattour

Crédit scoring lors de l'octroi d'un prêt

Réalisé au sein de ESPRIT

Ecole Supérieure Privée
d'Ingénierie et de Technologies

Année Universitaire 2019 - 2020

Rapport du Projet Python

Par

Mohamed Iheb Bousnina

Montassar Thabti

Fedi Bayoudh

Wissal Bayoudh

Khalil Ben romdhane

Mhadheb Ben Mahmoud

Mustapha Bouattour

Crédit scoring lors de l'octroi d'un prêt

Réalisé au sein de ESPRIT

Année Universitaire 2019 - 2020

Table des matières

Table des figures

Liste des abréviations

— **CAH** = **C**lassification- **A**scendante **H**iérarchique

— **KNN** = **K**- **N**earest **N**eighbor

— **SVM** = **S**upport- **V**ector **M**achines

Introduction générale

Plusieurs types de risques peuvent affecter la survie d'une banque. Parmi ces risques, on trouve notamment le risque de marché, d'option, de crédit, opérationnel, etc. Le risque de crédit, appelé également risque de contrepartie est le risque le plus répandu. S'il existe plusieurs types de risques de crédit, celui de non remboursement est un risque majeur. La crise financière actuelle trouve son origine principale dans ce type de risque, on peut prendre à titre d'exemple la crise des subprimes liée au problème du non remboursement des crédits immobiliers aux Etats-Unis. Plusieurs travaux de recherche ont été réalisés pour détecter à l'avance les emprunteurs qui seront défaillants de ceux qui ne seront pas. Ces travaux sont basés essentiellement sur l'analyse des comptes annuels des emprunteurs.

Chapitre 1

Cadre général du Projet

Plan

1.1 Introduction

A la recherche d'un moyen convivial pour quantifier le risque représenté par les demandeurs de prêt, ils nous ont été chargés de développer un modèle de scoring des clients qui devra assurer une classification fiable des clients selon plusieurs critères de solvabilité.

Ce projet sera centré sur la résolution de ce problème à partir de trois jeux de données chacun représente une banque différente. De ce fait, la première étape du projet consiste à comprendre les données « data understanding». Puis la phase de nettoyage et la préparation des données pour créer ensuite les modèles et enfin analyser les résultats et créer des rapports et des graphes de visualisation.

1.2 Contexte et problématique

L'appréciation du risque crédit est devenue pour la banque une préoccupation première, ainsi les politiques de risque crédit sont érigées de manière à essayer le plus possible d'encadrer ce risque et de pouvoir le quantifier.

Afin d'atteindre cet objectif, les banques ont recours à plusieurs outils d'aide à la prise de décision concernant le risque, tel le scoring.

cela nous mènera à répondre à une problématique qui s'impose aux banques en général lors d'octroi des crédits aux particuliers.

Comment peut-on évaluer le risque du crédit à partir des modéles de prédiction ?

1.2.1 Outils et technologies

Anaconda

Figure 1.1: Logo Anaconda

Anaconda est une distribution libre et open source3 des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique (traitement de données à grande échelle, analyse prédictive, calcul scientifique), qui vise à simplifier la gestion des paquets et de déploiement4. Les versions de paquetages sont gérées par le système de gestion de paquets conda5. La distribution Anaconda est utilisée par plus de 6 millions d'utilisateurs et comprend plus de 250 paquets populaires en science des données adaptés pour Windows, Linux et MacOS.

Jupyter Notebook

Figure 1.2: Logo Jupyter Notebook

Jupyter est une application web utilisée pour programmer dans plus de 40 langages de programmation, dont Python, Julia, Ruby, R, ou encore Scala2. Jupyter est une évolution du projet IPython. Jupyter permet de réaliser des calepins ou notebooks, c'est-à-dire des programmes contenant à la fois du texte en markdown et du code en Julia, Python, R... Ces notebooks sont utilisés en science des données pour explorer et analyser des données.

Python

Figure 1.3: Logo Python

Python est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Visual Code

Figure 1.4: Logo Visual Code

Visual Studio Code est présenté lors de la conférence des développeurs Build d'avril 2015 comme un éditeur de code multi-plateforme, open source et gratuit, supportant une dizaine de langages.

Django

Figure 1.5: Logo Django

Django est un cadre de développement web open source en Python. Il a pour but de rendre le développement web 2.0 simple et rapide. Pour cette raison, le projet a pour slogan « Le framework pour les perfectionnistes avec des deadlines. ». Développé en 2003 pour le journal local de Lawrence (Kansas), Django a été publié sous licence BSD à partir de juillet 2005.

Power BI

Figure 1.6: Logo Power BI

Power BI est une solution d'analytique métier qui vous permet de consulter vos données et de partager des insights au sein de votre organisation, ou de les intégrer à votre application ou à votre site web. Connectez-vous à des centaines de sources de données et donnez vie à vos données avec des tableaux de bord et rapports live.

1.3 Conclusion

Le credit scoring a fortement réformé l'analyse risque crédit des prêteurs. En rendant possible un traitement massif des demandes, il a rendu le crédit accessible à une plus large frange de la population. Sa nature statistique en fait un outil prédictif fiable, pour autant que les profils des clients ciblés soient similaires à ceux des clients ayant composé l'échantillon d'origine. Mais même établit sur une base large, l'échantillon n'est toutefois pas construit pour garantir la représentativité de la population dans son ensemble.

Chapitre 2

La banque Américaine

Plan

2.1 Compréhension des données

La première étape consiste à bien comprendre les aspects métier. Cette phase vise à déterminer précisément les données à analyser, à identifier la qualité des données disponibles et à faire le lien entre les données et leur signification d'un point de vue métier.

On a commencé par la banque américaine on a alors importé le jeu de données Bank-of-America-data.csv.

```
[3]: Bank_America.head()
```

[3]:

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
0	1	1100	25860.0	39025.0	HomeImp	Other	10.5	0.0	0.0	94.366667	1.0	9.0	NaN
1	1	1300	70053.0	68400.0	HomeImp	Other	7.0	0.0	2.0	121.833333	0.0	14.0	NaN
2	1	1500	13500.0	16700.0	HomeImp	Other	4.0	0.0	0.0	149.466667	1.0	10.0	NaN
3	1	1500	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	0	1700	97800.0	112000.0	HomeImp	Office	3.0	0.0	0.0	93.333333	0.0	14.0	NaN

Figure 2.1: Les 5 premières lignes du jeu de données Bank of America data

Ce jeu données comprend les 13 variables suivantes :

— BAD 1 = client en défaut de paiement 0 = prêt remboursé

— LOAN Montant de la demande de prêt

— MORTDUE Montant dû sur l'hypothèque existante

— VALUE Valeur de la propriété actuelle

— REASON DebtCon = consolidation de dettes HomeImp = amélioration de l'habitat

— JOB Six catégories professionnelles

— YOJ Années à l'emploi actuel

— DEROG Nombre de rapports dérogatoires majeurs

— DELINQ Nombre de lignes de crédit en souffrance

— CLAGE Âge de la ligne commerciale la plus ancienne en mois

— NINQ Nombre de lignes de crédit récentes

— CLNO Nombre de lignes de crédit

— DEBTINC Ratio dette-revenu

2.2 Nettoyage et Préparation des données

Cette phase de préparation des données regroupe les activités liées à la construction de l'ensemble précis des données à analyser, faite à partir des données brutes. Elle inclut ainsi le classement des données en fonction de critères choisis, le nettoyage des données, et surtout leur recodage pour les rendre compatibles avec les algorithmes qui seront utilisés.

Pour ce faire, on a commencé par la suppression des lignes qui contiennent moins de 6 valeurs non manquantes. Puis On a remplacé les valeurs manquantes par le job Other et les valeurs manquantes de la colonne MORTDUE par 0.8*la valeur de la colonne VALUE du meme ligne (parcequ'on a remarqué que c'est le cas pour les autres valeurs présentent). Pour terminer avec ces deux variables On a supprimé les 15 lignes que nous avons pas pu remplacer leurs colonne puisque ces deux colonnes n'ont ni une valeur dans MORTDUE ni dans VALUE.

```
Bank_America_1=Bank_America.dropna(thresh=6)

values = {'JOB': 'Other'}
Bank_America_1=Bank_America_1.fillna(value=values)

Bank_America_1['MORTDUE'] = Bank_America_1['MORTDUE'].fillna(value=Bank_America_1['VALUE']*0.8)

Bank_America_1['VALUE'] = Bank_America_1['VALUE'].fillna(value=Bank_America_1['MORTDUE']*1.2)
Bank_America_1.dropna(subset=["MORTDUE"],inplace=True)
Bank_America_1.dropna(subset=["VALUE"],inplace=True)
```

L'étape suivante consiste à faire un traitement qui nous permettera de remplir les valeurs manquantes REASON et ceci en cherchant la majorité des clients ayant un profil similaire au profil de ce client et ensuite on affecte la valeur de REASON de la majorité à ce champ vide

```
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["REASON"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        d=Bank_America_1.loc[(Bank_America_1["LOAN"]<x*1.2)&(x*0.8<Bank_America_1["LOAN"])
                             &(Bank_America_1["MORTDUE"]<y*1.2)&(y*0.8<Bank_America_1["MORTDUE"])
                             &(Bank_America_1["VALUE"]<z*1.2)&(z*0.8<Bank_America_1["VALUE"])]
        d1=d.loc[(d["REASON"]=="HomeImp")]
        s1=d1.shape[0]
        d2=d.loc[(d["REASON"]=="DebtCon")]
        s2=d2.shape[0]
        if(s1>s2):
            Bank_America_1["REASON"][index]=d1.iloc[0]["REASON"]
        elif (s2>s1):
            Bank_America_1["REASON"][index]=d2.iloc[0]["REASON"]
```

Une seule ligne n'a pas eu de REASON parcequ'il y a pas un profil proche de ce client donc on va supprimer cette ligne

On a fait un traitement qui nous permet de remplir les valeurs manquantes de YOJ et ceci

en cherchant la majorité des clients ayant un profil similaire au profil de ce client et on a calculé leurs mediane ensuite on a affecté la valeur de la mediane à ce champ vide

```
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["YOJ"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        t=Bank_America_1["REASON"][index]
        v=Bank_America_1["JOB"][index]
        d=Bank_America_1.loc[(pd.notna(Bank_America_1["YOJ"]))
                             &(Bank_America_1["JOB"]==v)&(Bank_America_1["REASON"]==t)&
                             (Bank_America_1["LOAN"]<x*1.2)&(x*0.8<Bank_America_1["LOAN"])&
                             (Bank_America_1["MORTDUE"]<y*1.2)&(y*0.8<Bank_America_1["MORTDUE"])&
                             (Bank_America_1["VALUE"]<z*1.2)&(z*0.8<Bank_America_1["VALUE"])]
        l=np.median(d["YOJ"])
        if np.isnan(l):
            l=2.0
        Bank_America_1["YOJ"][index]=round(l, 1)
```

Le même principe pour toutes les autres variables

```
# Meme principe que la précédente
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["DELINQ"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        t=Bank_America_1["REASON"][index]
        v=Bank_America_1["JOB"][index]
        w=Bank_America_1["YOJ"][index]
        d=Bank_America_1.loc[(pd.notna(Bank_America_1["DELINQ"]))&(Bank_America_1["JOB"]==v)&
                             (Bank_America_1["REASON"]==t)&(Bank_America_1["LOAN"]<x*1.5)&
                             (x*0.5<Bank_America_1["LOAN"])&(Bank_America_1["MORTDUE"]<y*1.5)&
                             (y*0.5<Bank_America_1["MORTDUE"])&(Bank_America_1["VALUE"]<z*1.5)
                             &(z*0.5<Bank_America_1["VALUE"])]
        if len(d)!=0:
            l=d["DELINQ"].iloc[0]
        else:
            l=0.0
        Bank_America_1["DELINQ"][index]=round(l, 1)
```

```
# Meme principe que la précédente
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["CLAGE"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        t=Bank_America_1["REASON"][index]
        v=Bank_America_1["JOB"][index]
        w=Bank_America_1["YOJ"][index]
        d=Bank_America_1.loc[(pd.notna(Bank_America_1["CLAGE"]))&(Bank_America_1["JOB"]==v)&
                             (Bank_America_1["REASON"]==t)&(Bank_America_1["LOAN"]<x*1.5)&
                             (x*0.5<Bank_America_1["LOAN"])&(Bank_America_1["MORTDUE"]<y*1.5)&
                             (y*0.5<Bank_America_1["MORTDUE"])&(Bank_America_1["VALUE"]<z*1.5)
                             &(z*0.5<Bank_America_1["VALUE"])]
        if len(d)!=0:
            l=np.median(d["CLAGE"])
        else:
            l=173.466667
        Bank_America_1["CLAGE"][index]=round(l, 6)
```

```
# Meme principe que la précédente
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["CLNO"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        t=Bank_America_1["REASON"][index]
        v=Bank_America_1["JOB"][index]
        d=Bank_America_1.loc[(pd.notna(Bank_America_1["CLNO"]))&(Bank_America_1["JOB"]==v)&
                             (Bank_America_1["REASON"]==t)&(Bank_America_1["LOAN"]<x*1.5)&
                             (x*0.5<Bank_America_1["LOAN"])&(Bank_America_1["MORTDUE"]<y*1.5)&
                             (y*0.5<Bank_America_1["MORTDUE"])&(Bank_America_1["VALUE"]<z*1.5)
                             &(z*0.5<Bank_America_1["VALUE"])]
        if len(d)!=0:
            l=np.median(d["CLNO"])
        else:
            l=20.0
        Bank_America_1["CLNO"][index]=round(l, 1)
```

```
# Meme principe que la précédente
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["NINQ"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        t=Bank_America_1["REASON"][index]
        v=Bank_America_1["JOB"][index]
        d=Bank_America_1.loc[(pd.notna(Bank_America_1["NINQ"]))&(Bank_America_1["JOB"]==v)&
                             (Bank_America_1["REASON"]==t)&(Bank_America_1["LOAN"]<x*1.5)&
                             (x*0.5<Bank_America_1["LOAN"])&(Bank_America_1["MORTDUE"]<y*1.5)&
                             (y*0.5<Bank_America_1["MORTDUE"])&(Bank_America_1["VALUE"]<z*1.5)
                             &(z*0.5<Bank_America_1["VALUE"])]
        if len(d)!=0:
            l=np.median(d["NINQ"])
        else:
            l=1.0
        Bank_America_1["NINQ"][index]=round(l, 1)
```

```
# Meme principe que la précédente
for index,row in Bank_America_1.iterrows():
    if (pd.isnull(row["DEBTINC"])):
        x=Bank_America_1["LOAN"][index]
        y=Bank_America_1["MORTDUE"][index]
        z=Bank_America_1["VALUE"][index]
        t=Bank_America_1["REASON"][index]
        v=Bank_America_1["JOB"][index]
        d=Bank_America_1.loc[(pd.notna(Bank_America_1["DEBTINC"]))&(Bank_America_1["JOB"]==v)
                             &(Bank_America_1["REASON"]==t)&(Bank_America_1["LOAN"]<x*1.5)&
                             (x*0.5<Bank_America_1["LOAN"])&(Bank_America_1["MORTDUE"]<y*1.5)&
                             (y*0.5<Bank_America_1["MORTDUE"])&(Bank_America_1["VALUE"]<z*1.5)
                             &(z*0.5<Bank_America_1["VALUE"])]
        if len(d)!=0:
            l=np.median(d["DEBTINC"])
        else:
            l=34.898413
        Bank_America_1["DEBTINC"][index]=round(l, 6)
```

```
# one hot encoding
df1 = pd.get_dummies(Bank_America_1.REASON)
df2 = pd.get_dummies(Bank_America_1.JOB)
Bank_America_After_Encoding=pd.concat([Bank_America_1,df1,df2],axis=1)
Bank_America_After_Encoding.drop(labels=['REASON','JOB'], axis =1, inplace = True)


# Mettre la column cible dans un vecteur à part et la supprimer de la dataframe
cible_America=Bank_America_After_Encoding["BAD"]
Bank_America_Final=Bank_America_After_Encoding.drop(labels=['BAD'], axis =1)

# DataFrame surlaquelle on va appliquer nos algorithmes:
Bank_America_Final
```

Les données aprés le nettoyage et l'encodage

	LOAN	MORTDUE	VALUE	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC	DebtCon	HomeImp	Mgr	Office	Other	ProfExe	Sales	Self
0	1100	25860.0	39025.0	10.5	0.0	0.0	94.366667	1.0	9.0	34.898413	0	1	0	0	1	0	0	0
1	1300	70053.0	68400.0	7.0	0.0	2.0	121.833333	0.0	14.0	34.898413	0	1	0	0	1	0	0	0
2	1500	13500.0	16700.0	4.0	0.0	0.0	149.466667	1.0	10.0	34.898413	0	1	0	0	1	0	0	0
4	1700	97800.0	112000.0	3.0	0.0	0.0	93.333333	0.0	14.0	30.635165	0	1	0	1	0	0	0	0
5	1700	30548.0	40320.0	9.0	0.0	0.0	101.466002	1.0	8.0	37.113614	0	1	0	0	1	0	0	0
6	1800	48649.0	57037.0	5.0	3.0	2.0	77.100000	1.0	17.0	35.891654	0	1	0	0	1	0	0	0
7	1800	28502.0	43034.0	11.0	0.0	0.0	88.766030	0.0	8.0	36.884894	0	1	0	0	1	0	0	0
8	2000	32700.0	46740.0	3.0	0.0	2.0	216.933333	1.0	12.0	36.388274	0	1	0	0	1	0	0	0
9	2000	49800.0	62250.0	16.0	0.0	0.0	115.800000	0.0	13.0	34.898413	0	1	0	0	0	0	1	0
11	2000	20627.0	29800.0	11.0	0.0	1.0	122.533333	1.0	9.0	34.898413	0	1	0	1	0	0	0	0
12	2000	45000.0	55000.0	3.0	0.0	0.0	86.066667	2.0	25.0	36.388274	0	1	0	0	1	0	0	0
13	2000	64536.0	87400.0	2.5	0.0	0.0	147.133333	0.0	24.0	38.263601	0	1	1	0	0	0	0	0
14	2100	71000.0	83850.0	8.0	0.0	1.0	123.000000	0.0	16.0	35.891654	0	1	0	0	1	0	0	0
15	2200	24280.0	34687.0	7.2	0.0	1.0	300.866667	0.0	8.0	36.636584	0	1	0	0	1	0	0	0
16	2200	90957.0	102600.0	7.0	2.0	6.0	122.900000	1.0	22.0	39.189639	0	1	1	0	0	0	0	0
18	2300	28192.0	40150.0	4.5	0.0	0.0	54.600000	1.0	16.0	36.636584	0	1	0	0	1	0	0	0
19	2300	102370.0	120953.0	2.0	0.0	0.0	90.992533	0.0	13.0	31.588503	0	1	0	1	0	0	0	0
20	2300	37626.0	46200.0	3.0	0.0	1.0	122.266667	1.0	14.0	36.512429	0	1	0	0	1	0	0	0
21	2400	50000.0	73395.0	5.0	1.0	0.0	253.182301	1.0	0.0	42.909997	0	1	0	0	0	1	0	0
22	2400	28000.0	40800.0	12.0	0.0	0.0	67.200000	2.0	22.0	19.394050	0	1	1	0	0	0	0	0
23	2400	18000.0	21600.0	22.0	0.0	2.0	121.733333	0.0	10.0	0.524499	0	1	1	0	0	0	0	0
24	2400	13744.0	17180.0	18.0	0.0	0.0	14.566667	3.0	4.0	34.898413	0	1	0	0	1	0	0	0
25	2400	34863.0	47471.0	12.0	0.0	0.0	70.491080	1.0	21.0	38.263601	0	1	1	0	0	0	0	0
26	2400	98449.0	117195.0	4.0	0.0	0.0	93.811775	0.0	13.0	29.681827	0	1	0	1	0	0	0	0
27	2500	15000.0	20200.0	18.0	0.0	0.0	136.066667	1.0	19.0	34.898413	0	1	0	0	1	0	0	0
28	2500	25116.0	36350.0	10.0	1.0	2.0	276.966667	0.0	9.0	36.512429	0	1	0	0	1	0	0	0
29	2500	7229.0	44516.0	2.0	0.0	0.0	208.000000	0.0	12.0	34.898413	0	1	0	0	0	0	0	1

Figure 2.2: La table de la banque Américaine encodée

2.3 Création des modeles

C'est la phase de Data Science proprement dite. La modélisation comprend le choix, le paramétrage et le test de différents algorithmes ainsi que leur enchaînement, qui constitue un modèle. Ce processus est d'abord descriptif pour générer de la connaissance, en expliquant pourquoi les choses se sont passées. Il devient ensuite prédictif en expliquant ce qu'il va se passer, puis prescriptif en permettant d'optimiser une situation future.

Pour ce faire, on a choisi d'appliquer plusieurs algorithmes afin de determiner l'algorithme le plus adequat possible en utilisant la courbe de Roc pour la détermination et la comparaison des performances diagnostiques de nos modéles.

Avant tout application d'algorithmes on a divisé les données en deux parties données cibles et données sans cible.

	BAD	LOAN	MORTDUE	VALUE	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC	DebtCon	HomeImp	Mgr	Office	Other	ProfExe	Sales	Self
0	1	1100	25860.0	39025.0	10.5	0.0	0.0	94.366667	1.0	9.0	34.898413	0	1	0	0	1	0	0	0

```
Bank_America.drop(['Unnamed: 0'],axis=1,inplace=True)
```

```
dataCible=Bank_America.iloc[:,0]
dataSansCible=Bank_America.iloc[:,1:]
```

2.3.1 Apprentissage supervisé

Avec ce type d'apprentissage on va récupérer des données annotées/ labélisées de leurs sorties pour entraîner le modèle.

Quand la variable à prédire prend une valeur discrète, on parle d'un problème de classification. Parmi les algorithmes de classification, on retrouve : Support Vector Machine (SVM), Réseaux de neurones, Naïve Bayes, Logistic Regression...

K-Nearest Neighbor

K-Nearest Neighbour (K-NN) est un algorithme simple qui stocke tous les cas disponibles et classe les nouvelles données ou cas en fonction d'une mesure de similarité.

La première étape de cet algorithme consiste à diviser les données en données de test et train. Puis, sélectionnez une valeur K nombre de voisin. Enfin il faut Déterminer la fonction de distance à utiliser.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=0)
```

```
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 differentes valeurs de k')
plt.xlabel('K ')
plt.ylabel('Erreur')
```

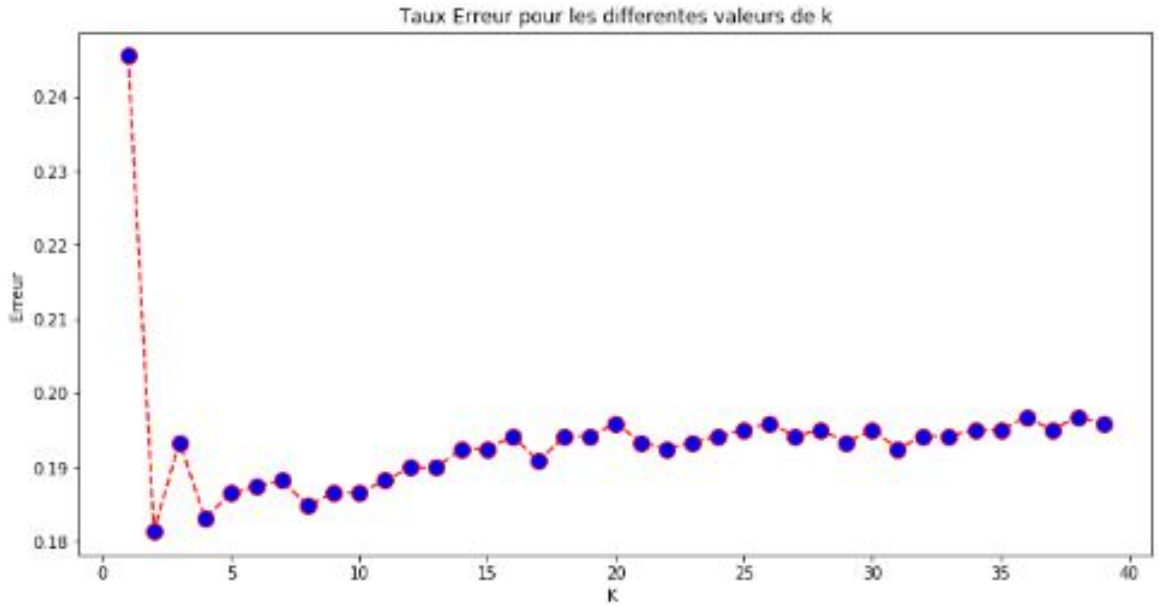


Figure 2.3: Taux Erreur pour les differentes valeurs de k (KNN)

La premiére valeur minimal est k=1 donc 1 est le nombre de voisins.

```
knn = KNeighborsClassifier(1)
knn_model = knn.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

```
acc_knn = accuracy_score(y_test, y_pred_knn)
acc_knn
```

```
0.7544910179640718
```

Nous avons obtenu 0,7544, ce qui signifie que notre modèle mesure environs 75% de données correctement.

Decision Trees (CART)

Un arbre de décision est une technique de modélisation d'apprentissage automatique efficace non paramétrique largement utilisée pour les problèmes de régression et de classification. Un arbre de classification est obtenu par un partitionnement récursif de l'espace d'entrée. Le but de cet algorithme est de démélanger les labels Pour produire la distribution la plus pure possible des étiquettes dans chaque noeud.

— Application du CART

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc_model = dtc.fit(X_train, y_train)
y_pred_dtc = dtc_model.predict(X_test)
print('Accuracy of CART classifier on training set: {:.2f}'
     .format(dtc.score(X_train, y_train)))
print('Accuracy of CART classifier on test set: {:.2f}'
     .format(dtc.score(X_test, y_test)))
```

— Evaluation des résultats

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_dtc))
```

	precision	recall	f1-score	support
0	0.86	0.34	0.48	927
1	0.24	0.79	0.36	242
accuracy			0.43	1169
macro avg	0.55	0.56	0.42	1169
weighted avg	0.73	0.43	0.46	1169

Support Vector Machines (SVM)

L'objectif de l'algorithme de machine à vecteur de support est de trouver un hyperplan dans un espace à N dimensions (N - le nombre d'entités) qui classe distinctement les points de données.
Pour séparer les deux classes de points de données, il existe de nombreux hyperplans possibles qui pourraient être choisis représenter dans la figure 2.4 . Notre objectif est de trouver un plan qui a la marge maximale, c'est-à-dire la distance maximale entre les points de données des deux classes. L'optimisation de la distance de marge fournit un certain renforcement afin que les futurs points de données puissent être classés avec plus de confiance.

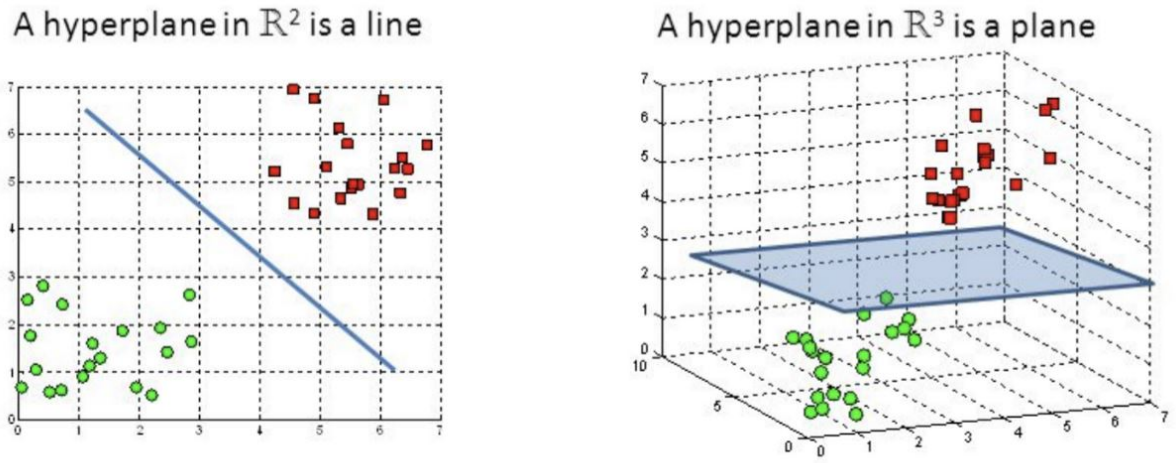


Figure 2.4: Hyperplans dans l'espace d'entités 2D et 3D

— Application de l'algorithme

```
from sklearn.svm import SVC, LinearSVC
svc = SVC(probability=True)
svc.fit(X_train, y_train)
predictions_svm = svc.predict(X_test)
acc_svc = accuracy_score(y_test, predictions_svm)
acc_svc
```

— Evaluation des résultats

Nous avons obtenu0.7929854576561164 , ce qui signifie que notre modèle mesure environs 79% de données correctement.

Perceptron

Perceptron est un classificateur linéaire (binaire). Il est utilisé dans l'apprentissage supervisé. Il aide à classer les données d'entrée données.

```
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, y_train)
predictions0 = perceptron.predict(X_test)
acc_perceptron = accuracy_score(y_test, predictions0)
acc_perceptron
```

```
0.7929854576561164
```

Déscente du gradient

La descente de gradient est un algorithme d'optimisation utilisé pour minimiser certaines fonctions en se déplaçant de manière itérative dans le sens de la descente la plus raide définie par le négatif du gradient. En apprentissage automatique, nous utilisons la descente de gradient pour mettre à jour les paramètres de notre modèle. Les paramètres se réfèrent aux coefficients de régression linéaire et aux poids dans les réseaux de neurones.

Application de cet algorithme :

```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd.fit(X_train, y_train)
predictionss = sgd.predict(X_test)
acc_sgd = accuracy_score(y_test, predictionss)
acc_sgd
```

Nous avons obtenu 0.7929854576561164 , ce qui signifie que notre modèle mesure environs 79% de données correctement.

Random Forest

La Random Forest, comme son nom l'indique, se compose d'un grand nombre d'arbres de décision individuels qui fonctionnent comme un ensemble. Chaque arbre individuel dans la forêt aléatoire crache une prédiction de classe et la classe avec le plus de votes devient la prédiction de notre modèle.

— Application du random Forest

```
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier, VotingClassifier, ExtraTreesClassifier, GradientBoostingClassifier

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions_rf = random_forest.predict(X_test)
print(random_forest.score(X_train, y_train))
acc_random_forest = accuracy_score(y_test, predictions_rf)
acc_random_forest
```

— Evaluation des résultats

```
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions_rf))
```

	precision	recall	f1-score	support
0	0.91	1.00	0.95	927
1	0.97	0.62	0.75	242
accuracy			0.92	1169
macro avg	0.94	0.81	0.85	1169
weighted avg	0.92	0.92	0.91	1169

Nous avons obtenu 0.92 , ce qui signifie que notre modèle mesure environs 92% de données correctement.

2.3.2 Apprentissage non supervisé

K-means

Le clustering K-means est l'un des algorithmes d'apprentissage automatique les plus simples et les plus populaires.

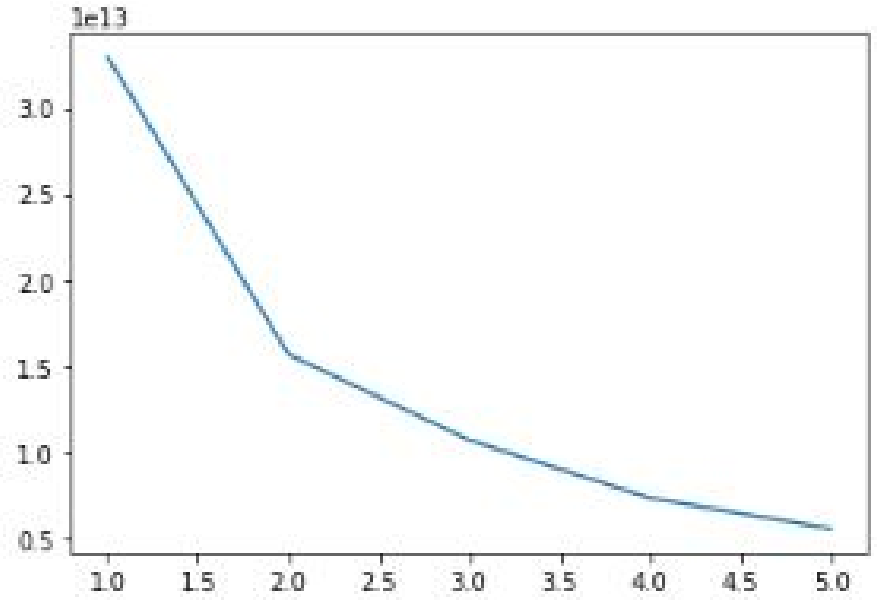
Pour traiter les données d'apprentissage, l'algorithme K-means commence par un premier groupe de centroïdes sélectionnés au hasard, qui sont utilisés comme points de départ pour chaque cluster, puis effectue des calculs itératifs (répétitifs) pour optimiser les positions des centroïdes.

— Application du kmeans

Etape 1 : Détermination de nombre de clusters

```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
```

```
# Déterminer Le nombre de cluster (L-Bow)
L = []
for i in range(1,6):
    model = KMeans(n_clusters=i)
    model.fit(dataSansCible)
    L.append(model.inertia_)
plt.plot(range(1,6),L)
```



Etape 2 :Application de l'algorithme

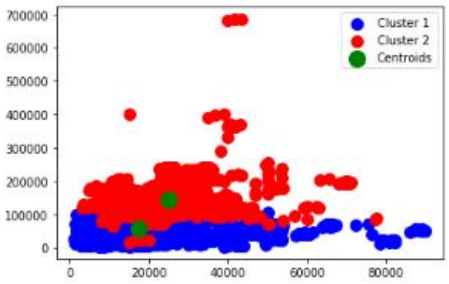
```
# Nombre de cluster = 2
kmeans = KMeans(n_clusters=2, precompute_distances='auto')
kmeans.fit(dataSansCible)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
y_kmeans = kmeans.fit_predict(dataSansCible)
```

Etape 3 : Représentation des clusters

```
plt.scatter(dataSansCible.values[y_kmeans == 0, 0], dataSansCible.values[y_kmeans == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
plt.scatter(dataSansCible.values[y_kmeans == 1, 0], dataSansCible.values[y_kmeans == 1, 1], s = 100, c = 'red', label = 'Cluster 2')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 200, c = 'green', label = 'Centroids')
plt.legend()
plt.show()
```



— Evaluation des résultats

```
idk = np.argsort(kmeans.labels_)
pd.crosstab(dataCible,kmeans.labels_)
```

col_0	0	1
BAD		
0	3812	872
1	956	203

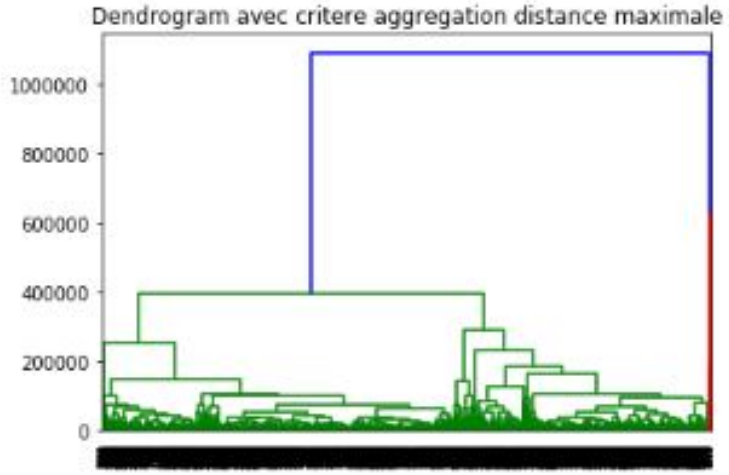
Classification Ascendante Hiérarchique (CAH)

l'objectifs de cet algorithme est de produire une structure (arborescence) permettant :

— La mise en évidence de liens hiérarchiques entre individus ou groupes d'individus

— La détection d'un nb de classes « naturel » au sein de la population

```
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
matrice1=linkage(dataSansCible, 'complete')
dendrogram(matrice1)
plt.title('Dendrogram avec critere aggregation distance maximale')
plt.show()
```




```
groupes_cah = fcluster(matrice1,t=700000,criterion='distance')
idg = np.argsort(groupes_cah)
# CrossTable
pd.crosstab(dataCible,groupes_cah)
```

col_0	1	2
BAD		
0	4675	9
1	1149	10

2.4 Analyse et évaluation des résultats

L'évaluation vise à vérifier les modèles ou les connaissances obtenues afin de s'assurer qu'ils répondent aux objectifs formulés au début du processus. Elle contribue aussi à la décision de déploiement du modèle ou, si besoin est, à son amélioration. A ce stade, on teste notamment la robustesse et la précision des modèles obtenus.

On a appliqué deux méthodes d'évaluation l'accurancy et la courbe de ROC

2.4.1 Tableau Comparatif des résultats

Le tableau compartif permet d'évaluer l'accurancy ou la précision des modéles appliqués et de déterminer le modéle le plus adéquat.

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'Score': [acc_svc, acc_knn,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_decision_tree]})
models.sort_values(by="Score",ascending=False)
```

	Model	Score
2	Random Forest	0.917023
3	Naive Bayes	0.805817
0	Support Vector Machines	0.792985
4	Perceptron	0.792985
5	Stochastic Gradient Decent	0.788708
1	KNN	0.754491
6	Decision Tree	0.436270

D'après le tableau de comparaison donné ci-dessus on a conclu que le randomForest est le modèle le plus approprié parce qu'il mesure environ 92% de données correctement prédites.

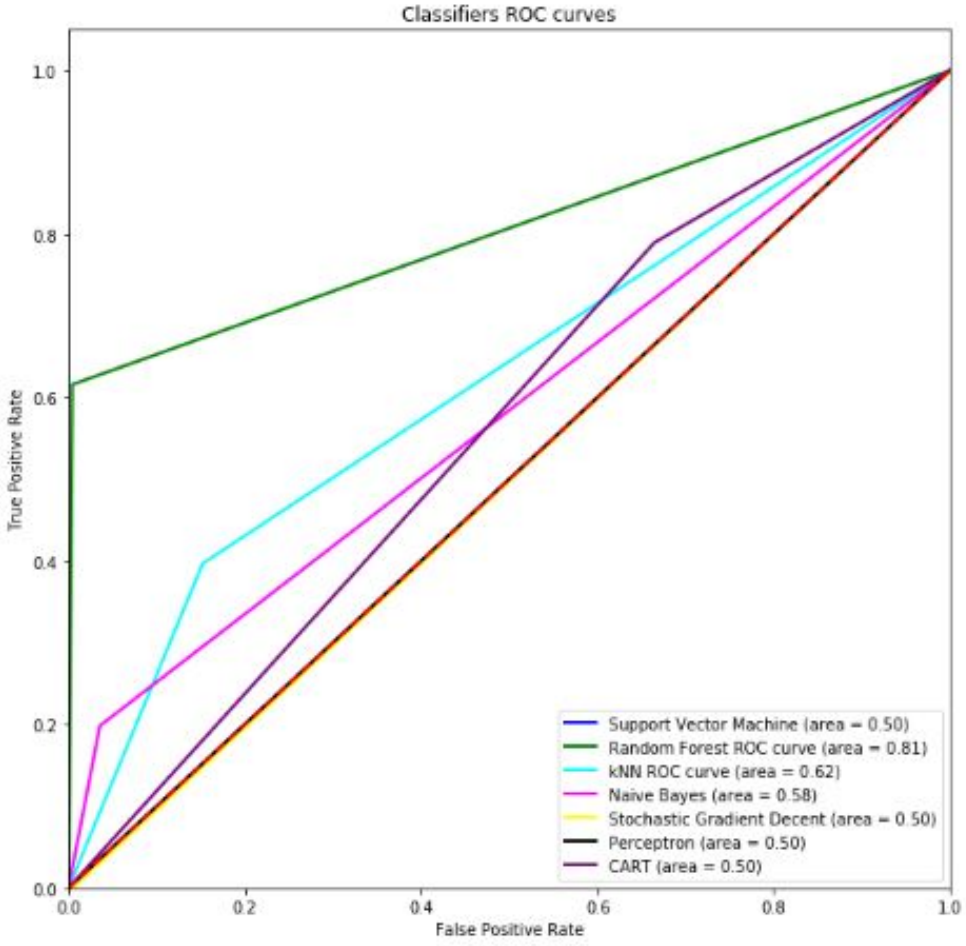
2.4.2 Courbe de ROC

Une courbe ROC (receiver operating characteristic) est un graphique représentant les performances d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs en fonction du taux de faux positifs.

```
from sklearn.metrics import roc_curve, auc
```

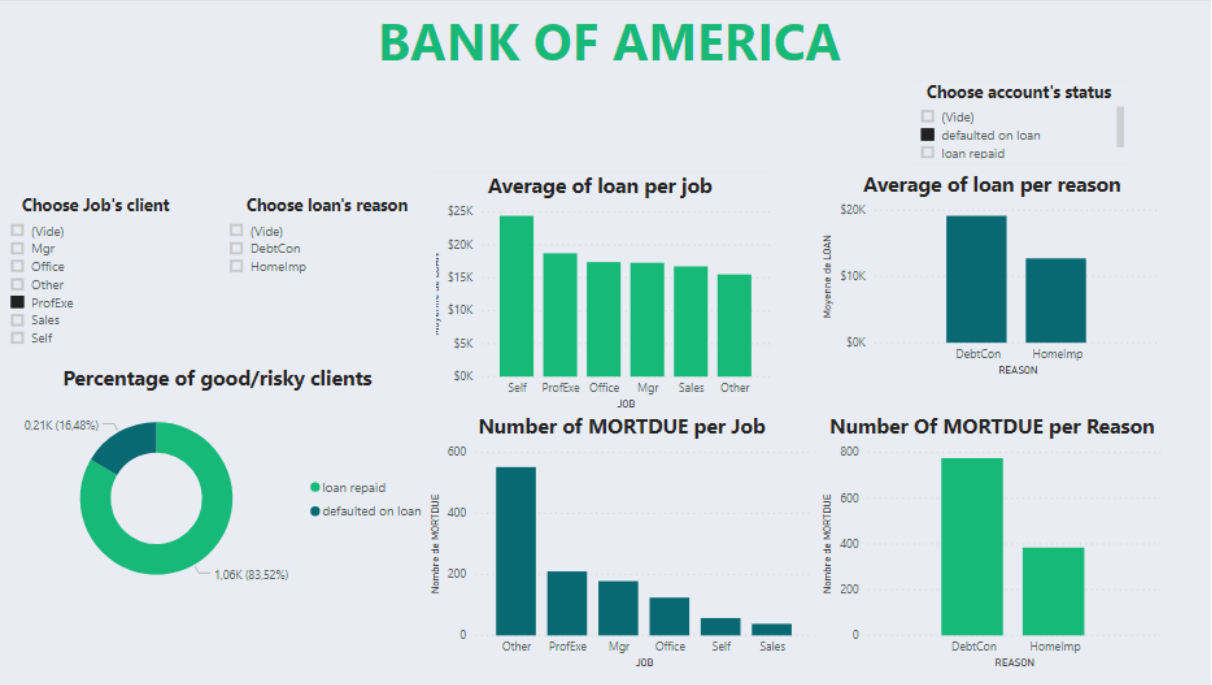
```
fpr1, tpr1, threshold1 = roc_curve(y_test, predictions_svm)
roc_auc1 = auc(fpr1, tpr1)
fpr2, tpr2, threshold2 = roc_curve(y_test, predictions_rf)
roc_auc2 = auc(fpr2, tpr2)
fpr3, tpr3, threshold3 = roc_curve(y_test, y_pred_knn)
roc_auc3 = auc(fpr3, tpr3)
fpr4, tpr4, threshold4 = roc_curve(y_test, predictions_B)
roc_auc4 = auc(fpr4, tpr4)
fpr5, tpr5, threshold5 = roc_curve(y_test, predictionss)
roc_auc5 = auc(fpr5, tpr5)
fpr6, tpr6, threshold6 = roc_curve(y_test, predictions0)
roc_auc6 = auc(fpr6, tpr6)
fpr7, tpr7, threshold7 = roc_curve(y_test, y_pred_dtc)
roc_auc7 = auc(fpr7, tpr7)
```

```
plt.figure(figsize=(10,10))
plt.plot(fpr1, tpr1, color='blue', lw=2, label='Support Vector Machine (area = %0.2f)'% roc_auc1)
plt.plot(fpr2, tpr2, color='green', lw=2, label='Random Forest ROC curve (area = %0.2f)'% roc_auc2)
plt.plot(fpr3, tpr3, color='cyan', lw=2, label='kNN ROC curve (area = %0.2f)'% roc_auc3)
plt.plot(fpr4, tpr4, color='magenta', lw=2, label='Naive Bayes (area = %0.2f)'% roc_auc4)
plt.plot(fpr5, tpr5, color='yellow', lw=2, label='Stochastic Gradient Decent (area = %0.2f)'% roc_auc5)
plt.plot(fpr6, tpr6, color='black', lw=2, label='Perceptron (area = %0.2f)'% roc_auc6)
plt.plot(fpr7, tpr7, color='#770080', lw=2, label='CART (area = %0.2f)'% roc_auc6)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Classifiers ROC curves')
plt.legend(loc = "lower right")
plt.show()
```



On a obtenu un résultat identique à celui du tableau comparatif. De ce fait le RandomForest est le modèle le plus adéquat.

2.5 Déploiement



Chapitre 3

La banque de Taiwan

Plan

3.1 Compréhension des données

Cette recherche a utilisé une variable binaire, le paiement par défaut (Oui = 1, Non = 0), comme variable de réponse. Cette étude a examiné la littérature et utilisé les 23 variables suivantes comme variables explicatives :

```
taiwan_Bank.head()
```

ID	LIMIT BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY 0	PAY 2	PAY 3	PAY 4	PAY 5	...	BILL AMT4	BILL AMT5	BILL AMT6	PAY AMT1	PAY AMT2	PAY AMT3	PAY AMT4	PAY AMT5	PAY AMT6	default payment next month
1	20000	2	2	1	24	2	2	-1	-1	-2	...	0	0	0	0	689	0	0	0	0	1
2	120000	2	2	2	26	-1	2	0	0	0	...	3272	3455	3261	0	1000	1000	1000	0	2000	1
3	90000	2	2	2	34	0	0	0	0	0	...	14331	14948	15549	1518	1500	1000	1000	1000	5000	0
4	50000	2	2	1	37	0	0	0	0	0	...	28314	28959	29547	2000	2019	1200	1100	1069	1000	0
5	50000	1	2	1	57	-1	0	-1	0	0	...	20940	19146	19131	2000	36681	10000	9000	689	679	0

Figure 3.1: Les 5 premières lignes du jeu de données Taiwan-bank

— X1 : Montant du crédit donné (en dollars NT) : il comprend à la fois le crédit à la consommation individuel et le crédit (supplémentaire) familial.

— X2 : Sexe (1 = homme ; 2 = femme).

— X3 : Éducation (1 = école supérieure ; 2 = université ; 3 = école secondaire ; 4 = autres).

— X4 : État matrimonial (1 = marié ; 2 = célibataire ; 3 = autres).

— X5 : Âge (année).

— X6 - X11 : Historique des paiements antérieurs. Nous avons suivi les enregistrements de paiements mensuels antérieurs (d'avril à septembre 2005) comme suit : X6 = l'état du remboursement en septembre 2005 ; X7 = l'état du remboursement en août 2005 ; . . . ; X11 = l'état de remboursement en avril 2005. L'échelle de mesure pour l'état de remboursement est : -1 = payer en temps voulu ; 1 = retard de paiement d'un mois ; 2 = retard de paiement de deux mois ; . . . ; 8 = retard de paiement de huit mois ; 9 = retard de paiement de neuf mois et plus.

— X12-X17 : Montant du relevé de facture (dollar NT). X12 = montant du relevé de facture en septembre 2005 ; X13 = montant du relevé de facture en août 2005 ; . . . ; X17 = montant du relevé de facture en avril 2005.

— X18-X23 : Montant du paiement précédent (en dollars NT). X18 = montant payé en septembre 2005 ; X19 = montant payé en août 2005 ; . . . ; X23 = montant payé en avril 2005.

3.2 Nettoyage et Préparation des données

Pour la banque de Taiwan il y a pas de données manquantes Donc on va faire directement l'encodage (One Hot Encoder)

```
taiwan_Bank.isnull().sum()
```

```
LIMIT_BAL                   0
SEX                         0
EDUCATION                   0
MARRIAGE                    0
AGE                         0
PAY_0                       0
PAY_2                       0
PAY_3                       0
PAY_4                       0
PAY_5                       0
PAY_6                       0
BILL_AMT1                   0
BILL_AMT2                   0
BILL_AMT3                   0
BILL_AMT4                   0
BILL_AMT5                   0
BILL_AMT6                   0
PAY_AMT1                    0
PAY_AMT2                    0
PAY_AMT3                    0
PAY_AMT4                    0
PAY_AMT5                    0
PAY_AMT6                    0
default payment next month  0
dtype: int64
```

Figure 3.2: Vérification des valeurs manquantes

```
taiwan_Bank_1=taiwan_Bank
# Il ya des valeurs autres que 1,2 et 3 alors on a remplacer ces valeurs par "Others" qui vaut 3
taiwan_Bank_1.MARRIAGE.replace([0], [3], inplace=True)
# Il ya des valeurs autres que 1,2,3 et 4 dans la colonne EDUCATION alors on a remplacer ces valeurs par "Other" qui vaut 4
taiwan_Bank_1.EDUCATION.replace([0], [4], inplace=True)
taiwan_Bank_1.EDUCATION.replace([5], [4], inplace=True)
taiwan_Bank_1.EDUCATION.replace([6], [4], inplace=True)
# one hot encoding
df1 = pd.get_dummies(taiwan_Bank_1.SEX)
df1.rename(columns = {1:'Male',2:'Female'}, inplace = True)
df2 = pd.get_dummies(taiwan_Bank_1.MARRIAGE)
df2.rename(columns = {1:'Mar_Married',2:'Mar_Single',3:'Mar_Others'}, inplace = True)
df3 = pd.get_dummies(taiwan_Bank_1.EDUCATION)
df3.rename(columns = {1:'Ed_GS',2:'Ed_HS',3:'Ed_Univ',4:'Ed_Other'}, inplace = True)
taiwan_Bank_1 = pd.concat([taiwan_Bank_1,df1,df2,df3], axis = 1)
taiwan_Bank_1.drop(labels=['SEX','MARRIAGE','EDUCATION'], axis =1, inplace = True)

# Mettre la column cible dans un vecteur à part et la supprimer de la dataframe
cible_taiwan=taiwan_Bank_1["default payment next month"]
taiwan_Bank_Final=taiwan_Bank_1.drop(labels=['default payment next month'], axis =1)

# DataFrame surlaquelle on va appliquer nos algorithmes:
taiwan_Bank_Final
```

Figure 3.3: Encodage de la table taiwan banque

3.3 Création des modeles

3.3.1 Apprentissage supervisé

K-Nearest Neighbor

On a commencé par chercher tout d'abord la valeur de K à l'aide de la courbe du taux d'erreur pour les differentes valeurs de K (K=21). Puis on a appliqué cet algorithme afin de predir les valeurs de la variable cible.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(taiwan_Bank_Final, cible_taiwan, random_state = 0)
```

```
# Centrage et réduction
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

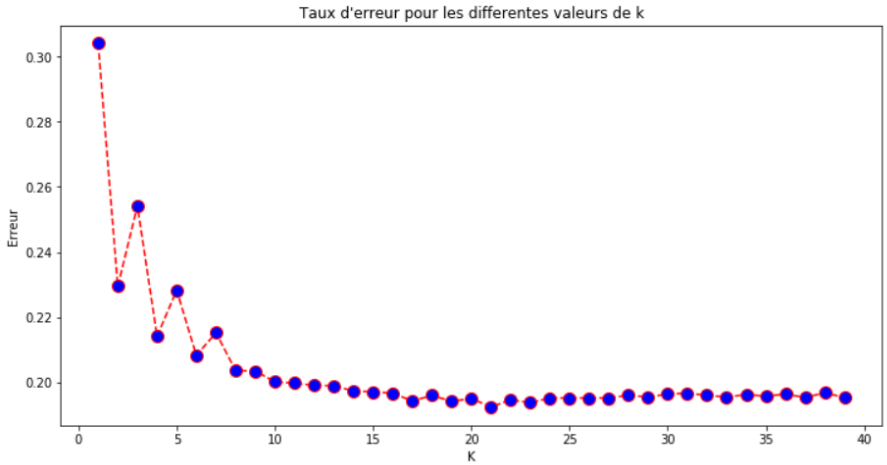
```
# Choisir le meilleur k
error = []

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 d\'erreur pour les differentes valeurs de k')
plt.xlabel('K')
plt.ylabel('Erreur')
```

```
Text(0, 0.5, 'Erreur')
```

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Taux d\'erreur pour les differentes valeurs de k')
plt.xlabel('K')
plt.ylabel('Erreur')
```

```
Text(0, 0.5, 'Erreur')
```



```
# Le meilleur k est 21 (min erreur)
knn = KNeighborsClassifier(21)
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)))

# Matrice de confusion
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_knn))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_knn))
```

```
Accuracy of K-NN classifier on training set: 0.82
Accuracy of K-NN classifier on test set: 0.81
[[5618  250]
 [1193  439]]
              precision    recall  f1-score   support

           0       0.82      0.96      0.89      5868
           1       0.64      0.27      0.38      1632

    accuracy                           0.81      7500
   macro avg       0.73      0.61      0.63      7500
weighted avg       0.78      0.81      0.78      7500
```

Decision Trees (CART)

Les arbres de décision sont des méthodes d'apprentissage non paramétrique utilisés pour des problèmes de classification et de régression. Le but est de créer un modèle qui prédit les valeurs de la variable cible, en se basant sur un ensemble de séquences de règles de décision déduites a partir des données d'apprentissage.

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc_model = DecisionTreeClassifier().fit(X_train, y_train)
y_pred_dtc = dtc_model.predict(X_test)
```

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

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_dtc))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_dtc))
```

```
Accuracy of CART classifier on training set: 0.82
Accuracy of CART classifier on test set: 0.81
[[3668 2200]
 [ 845  787]]
              precision    recall  f1-score   support

           0       0.81      0.63      0.71      5868
           1       0.26      0.48      0.34      1632

    accuracy                           0.59      7500
   macro avg       0.54      0.55      0.52      7500
weighted avg       0.69      0.59      0.63      7500
```

Support Vector Machines (SVM)

On a appliqué le SVM qui sert à résoudre des problèmes de discrimination et de régression. Les SVM sont une généralisation des classifieurs linéaires. On a opté à utiliser deux types de SVM :

SVM Linéaire

```
from sklearn.svm import SVC, LinearSVC
svc1 = SVC(kernel="linear")
svc1_model=svc1.fit(X_train, y_train)
y_pred_svc1 = svc1_model.predict(X_test)
```

```
print('Accuracy of SVM on training set: {:.2f}'
     .format(svc1.score(X_train, y_train)))
print('Accuracy of SVM on test set: {:.2f}'
     .format(svc1.score(X_test, y_test)))
```

```
Accuracy of SVM on training set: 0.79
Accuracy of SVM on test set: 0.80
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_svc1))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_svc1))
```

```
[[5405  463]
 [1043  589]]
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      5868
           1       0.56      0.36      0.44      1632

    accuracy                           0.80      7500
   macro avg       0.70      0.64      0.66      7500
weighted avg       0.78      0.80      0.78      7500
```

SVM Polynomiale

gamma ="scale"

```
from sklearn.svm import SVC, LinearSVC
svc3 = SVC(kernel="poly",gamma='scale')
svc3_model=svc3.fit(X_train, y_train)
y_pred_svc3 = svc3_model.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_svc3))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_svc3))
```

```
[[5703  165]
 [1279  353]]
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      5868
           1       0.68      0.22      0.33      1632

    accuracy                           0.81      7500
   macro avg       0.75      0.59      0.61      7500
weighted avg       0.79      0.81      0.77      7500
```

SGDClassifier

Cette methode est utilisée pour la minimisation d'une fonction objectif qui est écrite comme une somme de fonctions différentiables.

```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd_model=sgd.fit(X_train, y_train)
y_pred_sgd = sgd_model.predict(X_test)
```

```
print('Accuracy of SGDClassifier on training set: {:.2f}'
     .format(sgd.score(X_train, y_train)))
print('Accuracy of SGDClassifier on test set: {:.2f}'
     .format(sgd.score(X_test, y_test)))
```

```
Accuracy of SGDClassifier on training set: 0.80
Accuracy of SGDClassifier on test set: 0.81
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_sgd))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_sgd))
```

```
[[5524  344]
 [1091  541]]
              precision    recall  f1-score   support

           0       0.84      0.94      0.89      5868
           1       0.61      0.33      0.43      1632

    accuracy                           0.81      7500
   macro avg       0.72      0.64      0.66      7500
weighted avg       0.79      0.81      0.79      7500
```

Preceptron

Le Preceptron s'agit d'un neurone formel muni d'une règle d'apprentissage qui permet de déterminer automatiquement les poids synaptiques de manière à séparer un problème d'apprentissage supervisé.

```
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron_model=perceptron.fit(X_train, y_train)
y_pred_preceptron = perceptron_model.predict(X_test)
```

```
print('Accuracy of Preceptron on training set: {:.2f}'
     .format(perceptron.score(X_train, y_train)))
print('Accuracy of Preceptron on test set: {:.2f}'
     .format(perceptron.score(X_test, y_test)))
```

```
Accuracy of Preceptron on training set: 0.48
Accuracy of Preceptron on test set: 0.55
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_preceptron))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_preceptron))
```

```
[[2852 3016]
 [ 393 1239]]
              precision    recall  f1-score   support

           0       0.88      0.49      0.63      5868
           1       0.29      0.76      0.42      1632

    accuracy                           0.55      7500
   macro avg       0.59      0.62      0.52      7500
weighted avg       0.75      0.55      0.58      7500
```

Random Forest

Cet algorithme combine les concepts de sous-espaces aléatoires et de bagging. Il effectue un apprentissage sur de multiples arbres de décision entraînés sur des sous-ensembles de données légèrement différents.

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc_model = rfc.fit(X_train, y_train)
y_pred_rfc = rfc_model.predict(X_test)
```

```
print('Accuracy of Random Forest classifier on training set: {:.2f}'
     .format(rfc.score(X_train, y_train)))
print('Accuracy of Random Forest classifier on test set: {:.2f}'
     .format(rfc.score(X_test, y_test)))

from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred_rfc))

from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_rfc))
```

```
Accuracy of Random Forest classifier on training set: 0.82
Accuracy of Random Forest classifier on test set: 0.81
[[5664  204]
 [1380  252]]
              precision    recall  f1-score   support

           0       0.80      0.97      0.88      5868
           1       0.55      0.15      0.24      1632

    accuracy                           0.79      7500
   macro avg       0.68      0.56      0.56      7500
weighted avg       0.75      0.79      0.74      7500
```

3.3.2 Apprentissage non supervisé

K-means

Le K-means est un algorithme qui sert à diviser les points en k groupes, appelés clusters, de façon à minimiser une certaine fonction. On considère la distance d'un point à la moyenne des points de son cluster . La fonction à minimiser est la somme des carrés de ces distances.

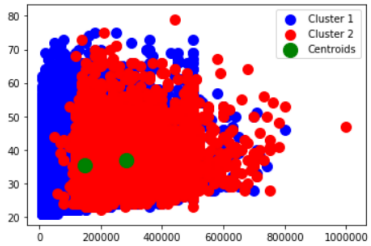
```
x=taiwan_Bank_Final.values
x
```

```
array([[ 20000,     24,      2, ...,      1,      0,      0],
       [120000,     26,     -1, ...,      1,      0,      0],
       [ 90000,     34,     -1, ...,      1,      0,      0],
       ...,
       [ 30000,     37,      4, ...,      1,      0,      0],
       [ 80000,     41,      1, ...,      0,      1,      0],
       [ 50000,     46,     -1, ...,      1,      0,      0]], dtype=int64)
```

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2, init = "k-means++", max_iter = 300, n_init = 10, random_state = 0)
kmeans.fit(taiwan_Bank_Final)
y_kmeans= kmeans.fit_predict(taiwan_Bank_Final)
pd.crosstab(cible_taiwan, kmeans.labels_)
```

col_0	0	1
default payment next month		
0	19983	3381
1	5862	774

```
plt.scatter(taiwan_Bank_Final.values[y_kmeans == 0, 0], taiwan_Bank_Final.values[y_kmeans == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
plt.scatter(taiwan_Bank_Final.values[y_kmeans == 1, 0], taiwan_Bank_Final.values[y_kmeans == 1, 1], s = 100, c = 'red', label = 'Cluster 2')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 200, c = 'green', label = 'Centroids')
plt.legend()
plt.show()
```




```
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn import metrics
metrics.adjusted_rand_score(cible_taiwan,kmeans.labels_)
```

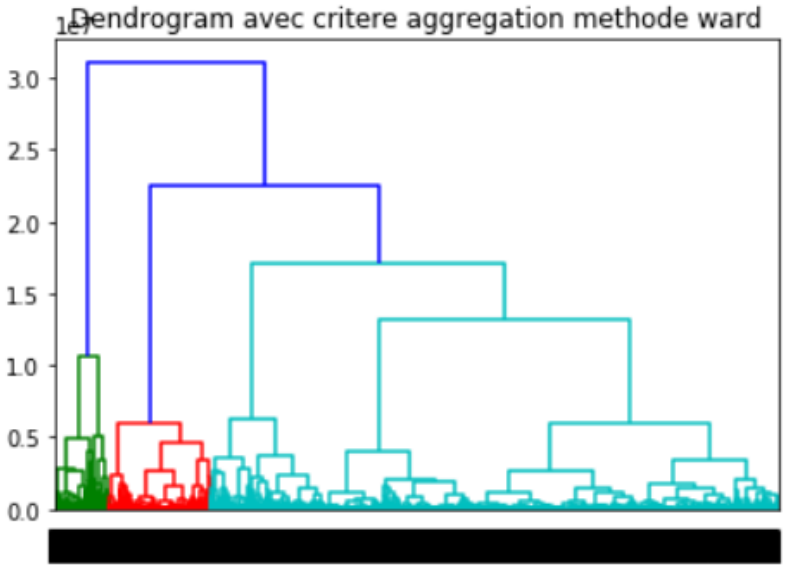
```
-0.018190949541530502
```

Classification Ascendante Hiérarchique (CAH)

C'est une méthode de classification automatique qui a pour but de répartir ces individus dans un certain nombre de classes.

```
#Librairies pour la CAH
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
```

```
matrice=linkage(taiwan_Bank_Final,method='ward',metric='euclidean')
dendrogram(matrice);
plt.title('Dendrogram avec critere aggregation methode ward')
plt.show()
```




```
#Couper le dendrogramme
groupes_cah = fcluster(matrice,t=25000000,criterion='distance')
print(groupes_cah)
```

```
[2 2 2 ... 2 2 2]
```

```
#correspondance les vrais labels avec les groupes de la CAH
pd.crosstab(cible_taiwan,groupes_cah)
```

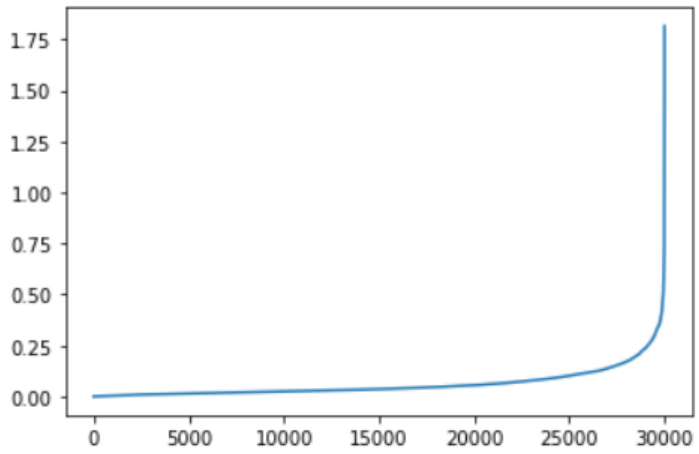
col_0	1	2
default payment next month		
0	1737	21627
1	461	6175

DBScan

L'algorithme DBSCAN utilise 2 paramètres : la distance epsilon et le nombre minimum de points MinPts qui doivent se trouver dans un rayon epsilon pour que ces points soient considérés comme un cluster.

```
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import MinMaxScaler

min_max_scaler=MinMaxScaler()
taiwan_Bank_Final_Scaled=min_max_scaler.fit_transform(taiwan_Bank_Final)
nbrs=NearestNeighbors(n_neighbors=2,algorithm='ball_tree').fit(taiwan_Bank_Final_Scaled)
distances, indices=nbrs.kneighbors(taiwan_Bank_Final_Scaled)
distances=np.sort(distances,axis=0)
distances=distances[:,1]
plt.plot(distances)
```

```
[<matplotlib.lines.Line2D at 0x1a5a45b0ba8>]
```




```
from sklearn.cluster import DBSCAN
db=DBSCAN(eps=0.3,min_samples=3).fit(taiwan_Bank_Final)
labels=db.labels_
pd.crosstab(cible_taiwan,labels)
```

col_0	-1	0	1	2	3
default payment next month					
0	23357	2	2	1	2
1	6631	1	1	2	1

3.4 Analyse et évaluation des résultats

3.4.1 Tableaux comparatifs

par Scort

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
```

```
acc_svc_line=accuracy_score(y_test, y_pred_svc1)
acc_svc_poly=accuracy_score(y_test, y_pred_svc3)
acc_NB=accuracy_score(y_test, y_pred_gaussian)
acc_knn=accuracy_score(y_test, y_pred_knn)
acc_preceptron=accuracy_score(y_test, y_pred_preceptron)
acc_rfc=accuracy_score(y_test, y_pred_rfc)
acc_sgd=accuracy_score(y_test, y_pred_sgd)
acc_dtc=accuracy_score(y_test, y_pred_dtc)
models = pd.DataFrame({
    'Model': ['Support Vector Machines Lineaire', 'Support Vector Machines Polynomial','KNN',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'Score': [acc_svc_line,acc_svc_poly, acc_knn,
              acc_rfc, acc_NB, acc_preceptron,
              acc_sgd, acc_dtc]})
models.sort_values(by="Score",ascending=False)
```

	Model	Score
6	Stochastic Gradient Decent	0.808667
2	KNN	0.807600
1	Support Vector Machines Polynomial	0.807467
0	Support Vector Machines Lineaire	0.799200
3	Random Forest	0.788800
4	Naive Bayes	0.758267
7	Decision Tree	0.594000
5	Perceptron	0.545467

On constate à travers la comparaison entre les differents Scores de nos modéles que le meilleur modéle à appliquer sur les données de la banque de Taiwan est le **sgdClassifier**.

tableau comparatif F1-score

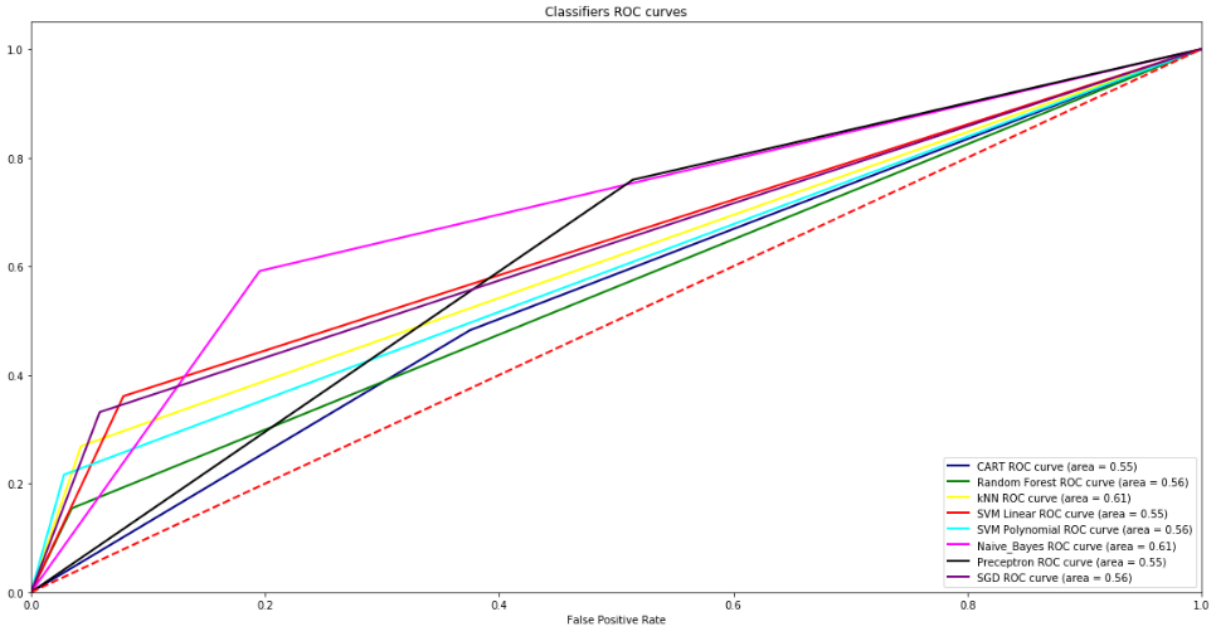
```
f1s_svc_line=f1_score(y_test, y_pred_svc1)
f1s_svc_poly=f1_score(y_test, y_pred_svc3)
f1s_NB=f1_score(y_test, y_pred_gaussian)
f1s_knn=f1_score(y_test, y_pred_knn)
f1s_preceptron=f1_score(y_test, y_pred_preceptron)
f1s_rfc=f1_score(y_test, y_pred_rfc)
f1s_sgd=f1_score(y_test, y_pred_sgd)
f1s_dtc=f1_score(y_test, y_pred_dtc)

models = pd.DataFrame({
    'Model': ['Support Vector Machines Lineaire', 'Support Vector Machines Polynomial','KNN',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'F1Score': [f1s_svc_line,f1s_svc_poly, f1s_knn,
              f1s_rfc, f1s_NB, f1s_preceptron,
              f1s_sgd, f1s_dtc]})
models.sort_values(by="F1Score",ascending=False)
```

	Model	F1Score
4	Naive Bayes	0.515629
0	Support Vector Machines Lineaire	0.438897
6	Stochastic Gradient Decent	0.429877
5	Perceptron	0.420927
2	KNN	0.378285
7	Decision Tree	0.340766
1	Support Vector Machines Polynomial	0.328372
3	Random Forest	0.241379

A partir des comparatifs entre les F1Scores, le meilleur modéle est le **Naive Bayes**

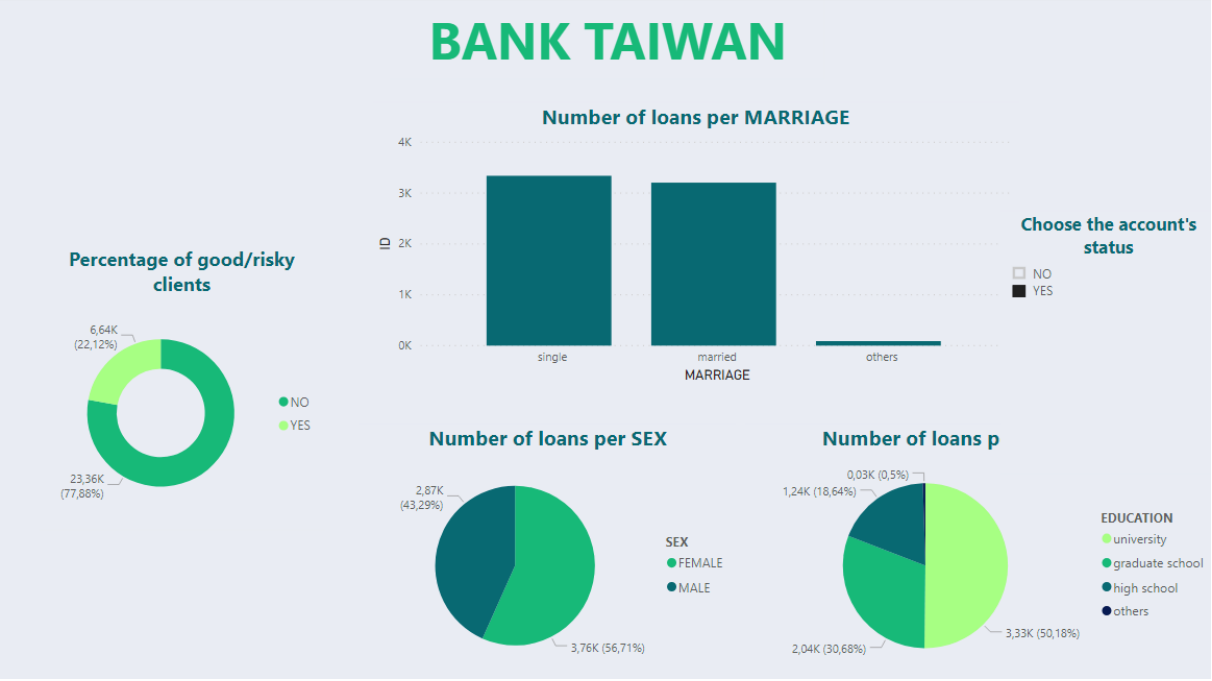
3.4.2 Courbe de ROC

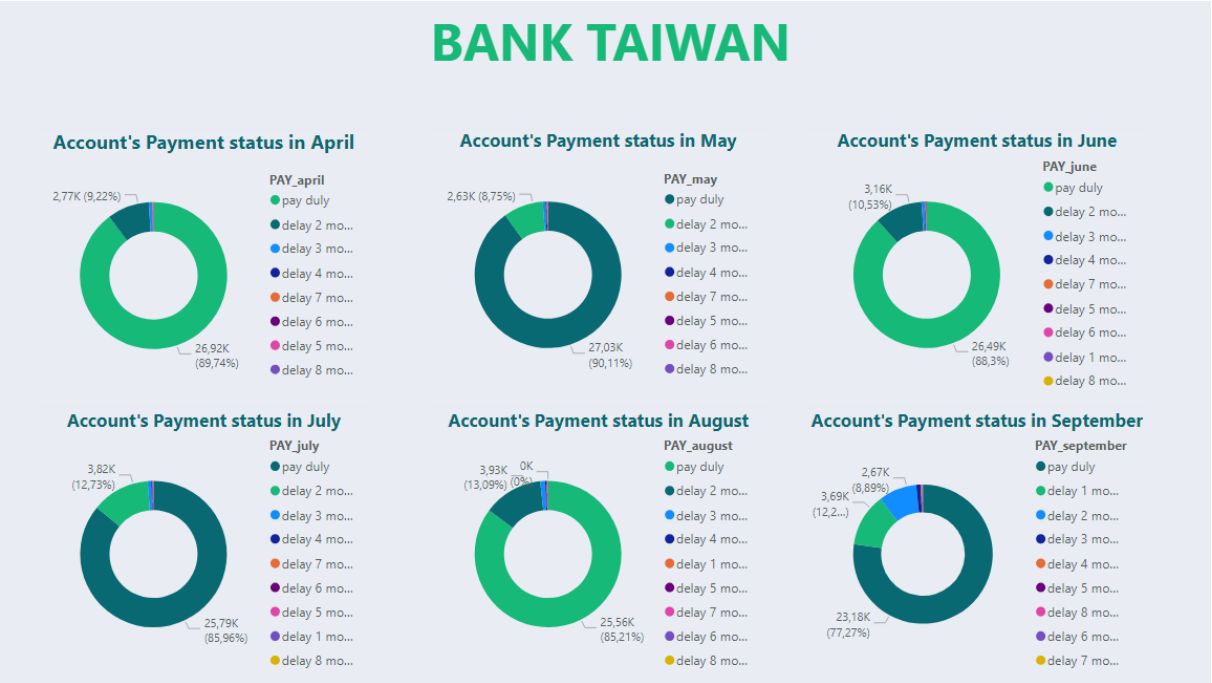
```
#Affichage avec la bibliothèque graphique intégrée à Notebook
%matplotlib inline

from sklearn.metrics import roc_curve, auc

fpr1, tpr1, threshold1 = roc_curve(y_test, y_pred_dtc)
roc_auc1 = auc(fpr1, tpr1)
fpr2, tpr2, threshold2 = roc_curve(y_test, y_pred_rfc)
roc_auc2 = auc(fpr2, tpr2)
fpr3, tpr3, threshold3 = roc_curve(y_test, y_pred_knn)
roc_auc3 = auc(fpr3, tpr3)
fpr4, tpr4, threshold4 = roc_curve(y_test, y_pred_svc1)
roc_auc4 = auc(fpr4, tpr4)
fpr5, tpr5, threshold5 = roc_curve(y_test, y_pred_svc3)
roc_auc5 = auc(fpr5, tpr5)
fpr6, tpr6, threshold6 = roc_curve(y_test, y_pred_gaussian)
roc_auc6 = auc(fpr6, tpr6)
fpr7, tpr7, threshold7 = roc_curve(y_test, y_pred_preceptron)
roc_auc7 = auc(fpr7, tpr7)
fpr8, tpr8, threshold8 = roc_curve(y_test, y_pred_sgd)
roc_auc8 = auc(fpr8, tpr8)


plt.figure(figsize=(20,10))
plt.plot(fpr1, tpr1, color='navy', lw=2, label='CART ROC curve (area = %0.2f)'% roc_auc1)
plt.plot(fpr2, tpr2, color='green', lw=2, label='Random Forest ROC curve (area = %0.2f)'% roc_auc2)
plt.plot(fpr3, tpr3, color='yellow', lw=2, label='kNN ROC curve (area = %0.2f)'% roc_auc3)

plt.plot(fpr4, tpr4, color='red', lw=2, label='SVM Linear ROC curve (area = %0.2f)'% roc_auc1)
plt.plot(fpr5, tpr5, color='cyan', lw=2, label='SVM Polynomial ROC curve (area = %0.2f)'% roc_auc2)
plt.plot(fpr6, tpr6, color='magenta', lw=2, label='Naive_Bayes ROC curve (area = %0.2f)'% roc_auc3)
plt.plot(fpr7, tpr7, color='black', lw=2, label='Preceptron ROC curve (area = %0.2f)'% roc_auc1)
plt.plot(fpr8, tpr8, color='#770080', lw=2, label='SGD ROC curve (area = %0.2f)'% roc_auc2)


plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Classifiers ROC curves')
plt.legend(loc = "lower right")
plt.show()
```



à travers la courbe de ROC, on peut constater que le meilleur algorithme est le **sgdClassifier** (le plus loin de la courbe rouge discontinue) en se basant sur leurs accuracy.

3.5 Déploiement






BANK TAIWAN
Selectionner le client par ID
9
NO
Client's education degree
high school
Loan amount
Client's Age
28
$140K
$0K
$280K
Client's marital status
married
Client's Sex
FEMALE
PAY_september
PAY_august
PAY_july
PAY_june
PAY_may
PAY_april
pay duly
pay duly
delay 2
pay duly
pay duly
pay duly

Chapitre 4

La banque Allemande

Plan

4.1 Compréhension des données

On ce qui concerne la 3éme banque la banque allemande Deux jeux de données sont fournis. L'ensemble de données d'origine contient des attributs catégoriques / symboliques et se trouve dans le fichier "german.data".

Pour les algorithmes qui nécessitent des attributs numériques, on a un deuxième fichier "german.data-numeric". Ce fichier a été édité et plusieurs variables d'indicateur ajoutées pour le rendre approprié pour Algorithmes qui ne peuvent pas faire face aux variables catégorielles. Nombreuses les attributs classés par catégorie (comme l'attribut 17) ont été codé comme un entier.

```
german_Bank.head().T
```

	0	1	2	3	4
Status	A11	A12	A14	A11	A11
Duration	6	48	12	42	24
credit_history	A34	A32	A34	A32	A33
Purpose	A43	A43	A46	A42	A40
Credit_amount	1169	5951	2096	7882	4870
Saving_amount	A65	A61	A61	A61	A61
YOJ	A75	A73	A74	A74	A73
Installment_rate	4	2	2	2	3
Status_SEX	A93	A92	A93	A93	A93
debtors_guarantors	A101	A101	A101	A103	A101
Residence	4	2	3	4	4
Property	A121	A121	A121	A122	A124
Age	67	22	49	45	53
installment_plans	A143	A143	A143	A143	A143
Housing	A152	A152	A152	A153	A153
Nb_credit	2	1	1	1	2
Job	A173	A173	A172	A173	A173
persons_liable	1	1	2	2	2
Telephone	A192	A191	A191	A191	A191
Foreign_warker	A201	A201	A201	A201	A201
Cost	1	2	1	1	2

Figure 4.1: Les 5 premières lignes du jeu de données Bank of german

```
german_Bank.dtypes
```

```
Status                object
Duration               int64
credit_history        object
Purpose               object
Credit_amount          int64
Saving_amount         object
YOJ                   object
Installment_rate       int64
Status_SEX            object
debtors_guarantors    object
Residence              int64
Property              object
Age                    int64
installment_plans     object
Housing               object
Nb_credit              int64
Job                   object
persons_liable         int64
Telephone             object
Foreign_worker        object
Cost                   int64
dtype: object
```

Figure 4.2: Vérification des types des attributs

Nombre d'attributs allemand : 20 (7 numériques, 13 catégoriques)

4.2 Nettoyage et Préparation des données

Pour la banque allemande il y a pas de données manquantes Donc on a fait directement l'encodage (One Hot Encoder)

```
german_Bank_1=german_Bank
# one hot encoding
df1 = pd.get_dummies(german_Bank_1.Status)
df1.rename(columns = {'A11':'Status_A11','A12':'Status_A12','A13':'Status_A13','A14':'Status_A14'}, inplace = True)
df2 = pd.get_dummies(german_Bank_1.credit_history)
df2.rename(columns = {'A30':'CredHist_A30','A31':'CredHist_A31','A32':'CredHist_A32',
                      'A33':'CredHist_A33','A34':'CredHist_A34'}, inplace = True)
df3 = pd.get_dummies(german_Bank_1.Purpose)
df3.rename(columns = {'A40':'Purpose_A40','A41':'Purpose_A41','A42':'Purpose_A42','A43':'Purpose_A43',
                      'A44':'Purpose_A44','A45':'Purpose_A45','A46':'Purpose_A46','A47':'Purpose_A47',
                      'A48':'Purpose_A48','A49':'Purpose_A49','A410':'Purpose_A410'}, inplace = True)
df4 = pd.get_dummies(german_Bank_1.Saving_amount)
df4.rename(columns = {'A61':'SaveAccount_A61','A62':'SaveAccount_A62','A63':'SaveAccount_A63',
                      'A64':'SaveAccount_A64','A65':'SaveAccount_A65'}, inplace = True)
df5 = pd.get_dummies(german_Bank_1.YOJ)
df5.rename(columns = {'A71':'YOJ_A71','A72':'YOJ_A72','A73':'YOJ_A73','A74':'YOJ_A74','A75':'YOJ_A75'}, inplace = True)
df6 = pd.get_dummies(german_Bank_1.Status_SEX)
df6.rename(columns = {'A91':'Status&Sex_A91','A92':'Status&Sex_A92','A93':'Status&Sex_A93','A94':'Status&Sex_A94',
                      'A95':'Status&Sex_A95'}, inplace = True)
df7 = pd.get_dummies(german_Bank_1.debtors_guarantors)
df7.rename(columns = {'A101':'DebGuarant_A101','A102':'DebGuarant_A102',
                      'A103':'DebGuarant_A103'}, inplace = True)
df9 = pd.get_dummies(german_Bank_1.Property)
df9.rename(columns = {'A121':'Property_A121','A122':'Property_A122','A123':'Property_A123',
                      'A124':'Property_A124'}, inplace = True)
df10 = pd.get_dummies(german_Bank_1.installment_plans)
df10.rename(columns = {'A141':'InstallPlan_A141','A142':'InstallPlan_A142','A143':'InstallPlan_A143'}, inplace = True)
df11 = pd.get_dummies(german_Bank_1.Housing)
df11.rename(columns = {'A151':'Housing_A151','A152':'Housing_A152','A153':'Housing_A153'}, inplace = True)
df12 = pd.get_dummies(german_Bank_1.Job)
df12.rename(columns = {'A171':'Job_A171','A172':'Job_A172','A173':'Job_A173','A174':'Job_A174'}, inplace = True)
df13 = pd.get_dummies(german_Bank_1.Telephone)
df13.rename(columns = {'A191':'Tel_None','A192':'Tel_Registred'}, inplace = True)
df14 = pd.get_dummies(german_Bank_1.Foreign_worker)
df14.rename(columns = {'A201':'ForeignWorker_Yes','A202':'ForeignWorker_No'}, inplace = True)

german_Bank_1 = pd.concat([german_Bank_1,df1,df2,df3,df4,df5,df6,df7,df9,df10,df11,df12,df13,df14], axis = 1)
german_Bank_1.drop(labels=['Status','credit_history','Purpose','Saving_amount','YOJ','Status_SEX',
                           'debtors_guarantors','Property','installment_plans',
                           'Housing','Job','Telephone','Foreign_worker'], axis =1, inplace = True)

# Mettre la column cible dans un vecteur à part et la supprimer de la dataframe
cible_german=german_Bank_1["Cost"]
german_Bank_Final=german_Bank_1.drop(labels=['Cost'], axis =1)

# DataFrame surLaquelle on va appliquer nos algorithmes:
german_Bank_Final
```

4.3 Création des modeles

4.3.1 Apprentissage supervisé

K-Nearest Neighbor (KNN)

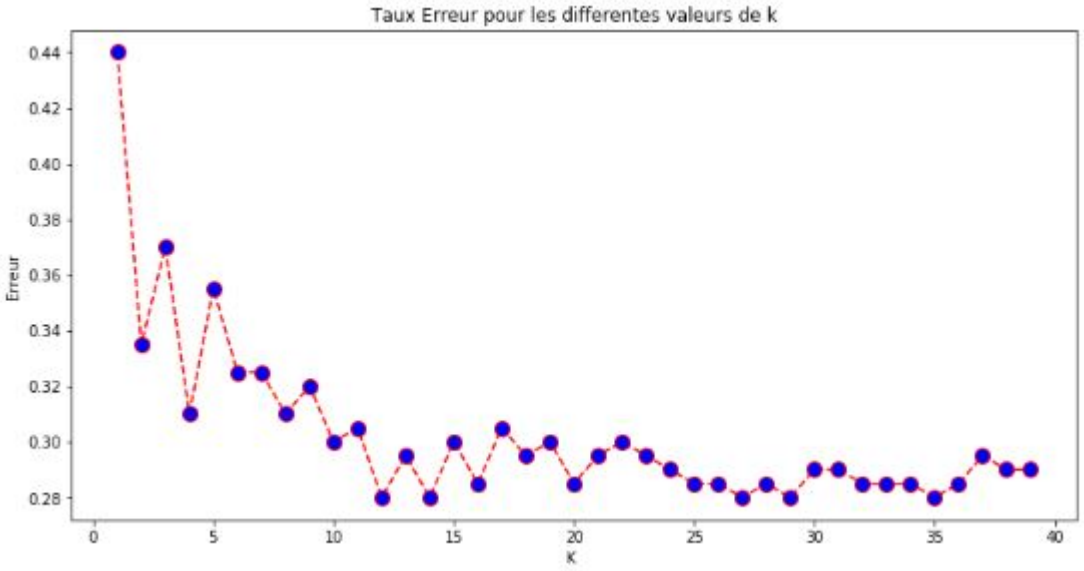
```
print(german_Bank.groupby('Cost').size())
```

```
Cost
1    700
2    300
dtype: int64
```

```
X = dataSansCible
y= pd.factorize(dataCible)[0]
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2, random_state=0)
```

```
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 differentes valeurs de k')
plt.xlabel('K ')
plt.ylabel('Erreur')
```



```
knn = KNeighborsClassifier(12)
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: 0.72
Accuracy of K-NN classifier on test set: 0.72
```

```
acc_knn = accuracy_score(y_test, y_pred_knn)
acc_knn
```

```
0.72
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.73      0.96      0.83       142
           1       0.57      0.14      0.22        58

    accuracy                           0.72       200
   macro avg       0.65      0.55      0.53       200
weighted avg       0.68      0.72      0.65       200
```

Decision Trees (CART)

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc_model = dtc.fit(X_train, y_train)
y_pred_dtc = dtc_model.predict(X_test)
print('Accuracy of CART classifier on training set: {:.2f}'
     .format(dtc.score(X_train, y_train)))
print('Accuracy of CART classifier on test set: {:.2f}'
     .format(dtc.score(X_test, y_test)))
```

```
Accuracy of CART classifier on training set: 1.00
Accuracy of CART classifier on test set: 0.65
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_dtc))
```

```
              precision    recall  f1-score   support

           0       0.76      0.75      0.75       142
           1       0.40      0.41      0.41        58

    accuracy                           0.65       200
   macro avg       0.58      0.58      0.58       200
weighted avg       0.65      0.65      0.65       200
```

Perceptron

```
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, y_train)
predictions0 = perceptron.predict(X_test)
acc_perceptron = accuracy_score(y_test, predictions0)
acc_perceptron
```

```
0.71
```

Déscente du gradient

```
from sklearn.linear_model import SGDClassifier

sgd = SGDClassifier()
sgd.fit(X_train, y_train)
predictionss = sgd.predict(X_test)
acc_sgd = accuracy_score(y_test, predictionss)
acc_sgd
```

```
0.31
```

Support Vector Machines (SVM)

```
from sklearn.svm import SVC, LinearSVC
svc1 = SVC(kernel="linear")
svc1_model=svc1.fit(X_train, y_train)
y_pred_svc1 = svc1_model.predict(X_test)
```

```
acc_svc = accuracy_score(y_test, y_pred_svc1)
acc_svc
```

```
0.75
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier, VotingClassifier, ExtraTreesClassifier, GradientBoostingClassifier

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, y_train)
predictions_rf = random_forest.predict(X_test)
print(random_forest.score(X_train, y_train))
acc_random_forest = accuracy_score(y_test, predictions_rf)
acc_random_forest
```

```
1.0
0.73
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions_rf))
```

```
              precision    recall  f1-score   support

           0       0.78      0.87      0.82       142
           1       0.55      0.40      0.46        58

    accuracy                           0.73       200
   macro avg       0.66      0.63      0.64       200
weighted avg       0.71      0.73      0.72       200
```

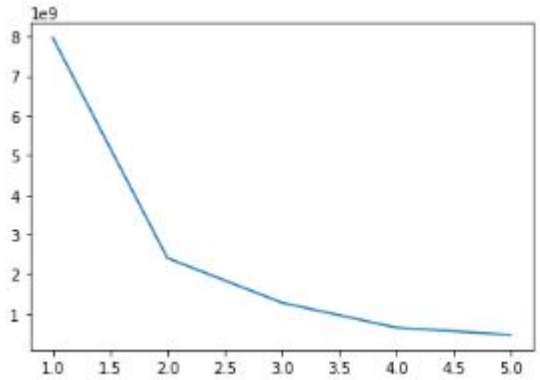
4.3.2 Apprentissage non supervisé

K-means

```
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
```

```
# Déterminer le nombre de cluster (L-Bow)
L = []
for i in range(1,6):
    model = KMeans(n_clusters=i)
    model.fit(dataSansCible)
    L.append(model.inertia_)
plt.plot(range(1,6),L)
```

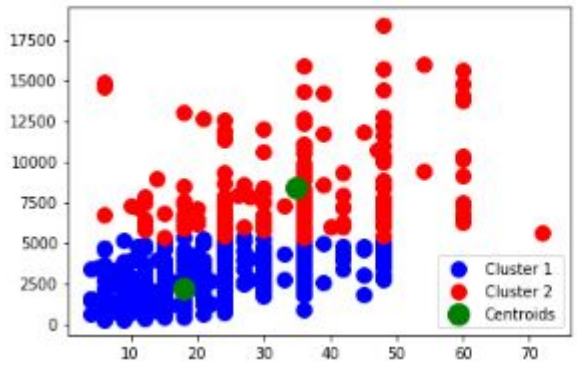
```
[<matplotlib.lines.Line2D at 0x1d5aa301128>]
```



```
# Nombre de cluster = 2
kmeans = KMeans(n_clusters=2, precompute_distances='auto')
kmeans.fit(dataSansCible)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
y_kmeans = kmeans.fit_predict(dataSansCible)
```

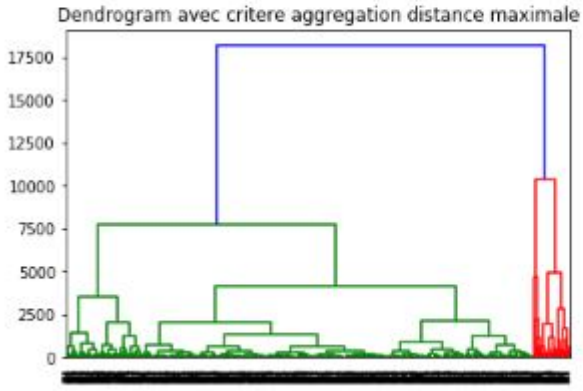



```
idk = np.argsort(kmeans.labels_)
pd.crosstab(dataCible,kmeans.labels_)
```

col_0	0	1
Cost		
1	599	101
2	228	72

Classification Ascendante Hiérarchique (CAH)

```
from scipy.cluster.hierarchy import dendrogram, linkage,fcluster
matrice1=linkage(dataSansCible, 'complete')
dendrogram(matrice1)
plt.title('Dendrogram avec critere aggregation distance maximale')
plt.show()
```



```
groupes_cah = fcluster(matrice1,t=12500,criterion='distance')
idg = np.argsort(groupes_cah)
# CrossTable
pd.crosstab(dataCible,groupes_cah)
```

col_0	1	2
Cost		
1	668	32
2	262	38

4.4 Analyse et évaluation des résultats

4.4.1 Tableau Comparatif des résultats

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent'],
    'Score': [acc_svc, acc_knn,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd]})
models.sort_values(by="Score",ascending=False)
```

	Model	Score
0	Support Vector Machines	0.75
2	Random Forest	0.73
1	KNN	0.72
3	Naive Bayes	0.71
4	Perceptron	0.71
5	Stochastic Gradient Decent	0.31

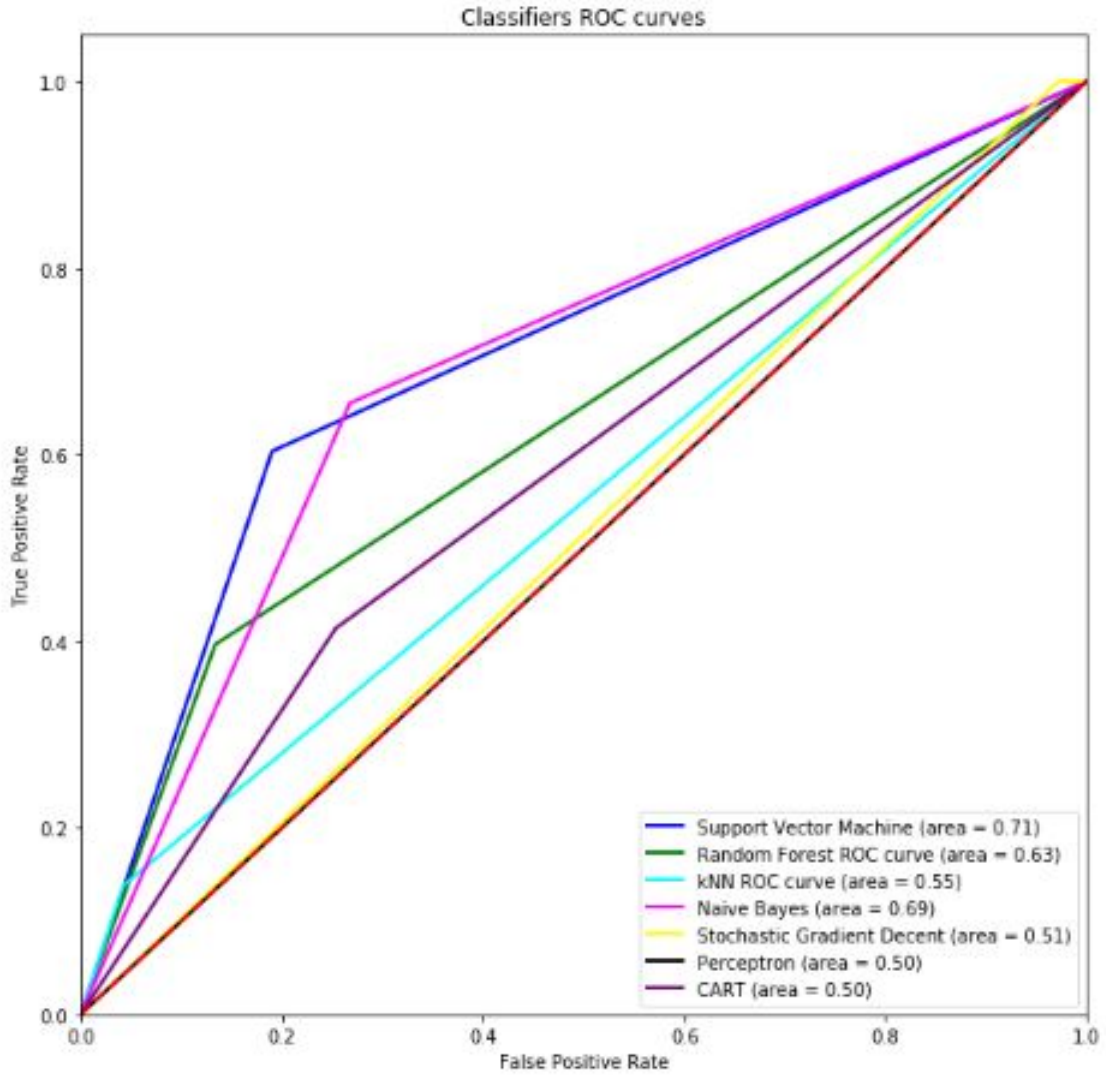
D'aprés le tableau de comparaison donné ci-dessus on a conclu que le SVM est le modèle leplus approprié parce qu'il mesure environ 75% de données correctement prédites.

4.4.2 Courbe de ROC

```
from sklearn.metrics import roc_curve, auc
```

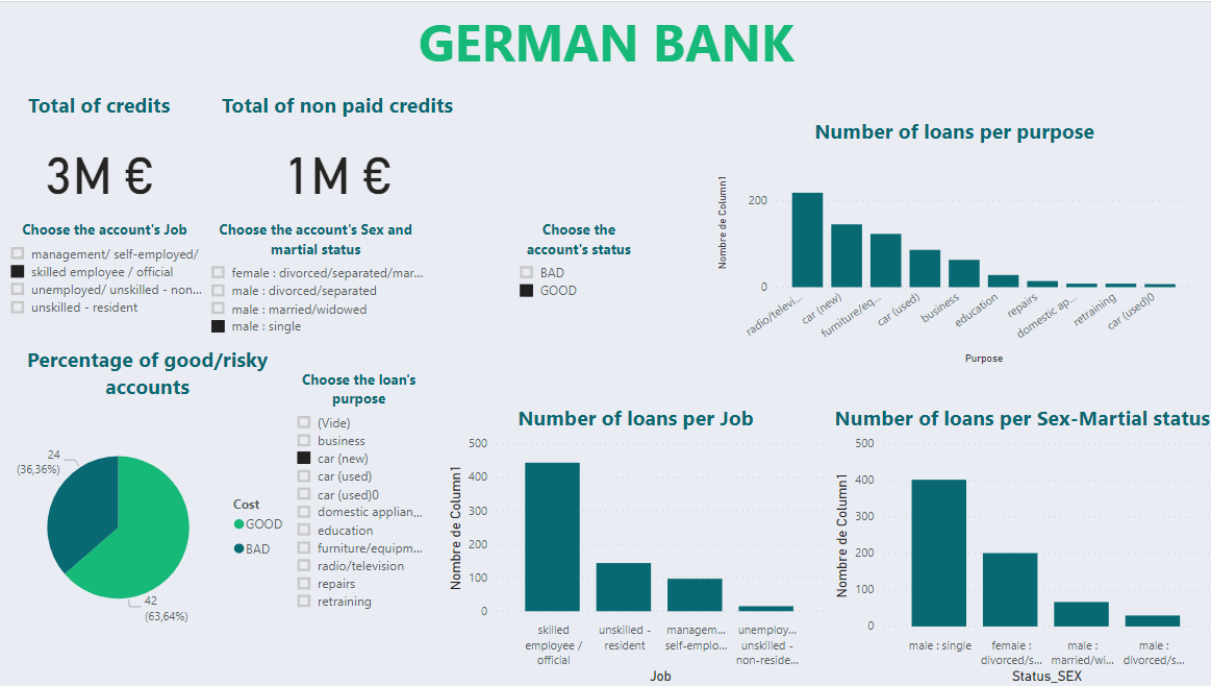
```
fpr1, tpr1, threshold1 = roc_curve(y_test, y_pred_svc1)
roc_auc1 = auc(fpr1, tpr1)
fpr2, tpr2, threshold2 = roc_curve(y_test, predictions_rf)
roc_auc2 = auc(fpr2, tpr2)
fpr3, tpr3, threshold3 = roc_curve(y_test, y_pred_knn)
roc_auc3 = auc(fpr3, tpr3)
fpr4, tpr4, threshold4 = roc_curve(y_test, predictions_B)
roc_auc4 = auc(fpr4, tpr4)
fpr5, tpr5, threshold5 = roc_curve(y_test, predictionss)
roc_auc5 = auc(fpr5, tpr5)
fpr6, tpr6, threshold6 = roc_curve(y_test, predictions0)
roc_auc6 = auc(fpr6, tpr6)
fpr7, tpr7, threshold7 = roc_curve(y_test, y_pred_dtc)
roc_auc7 = auc(fpr7, tpr7)
```

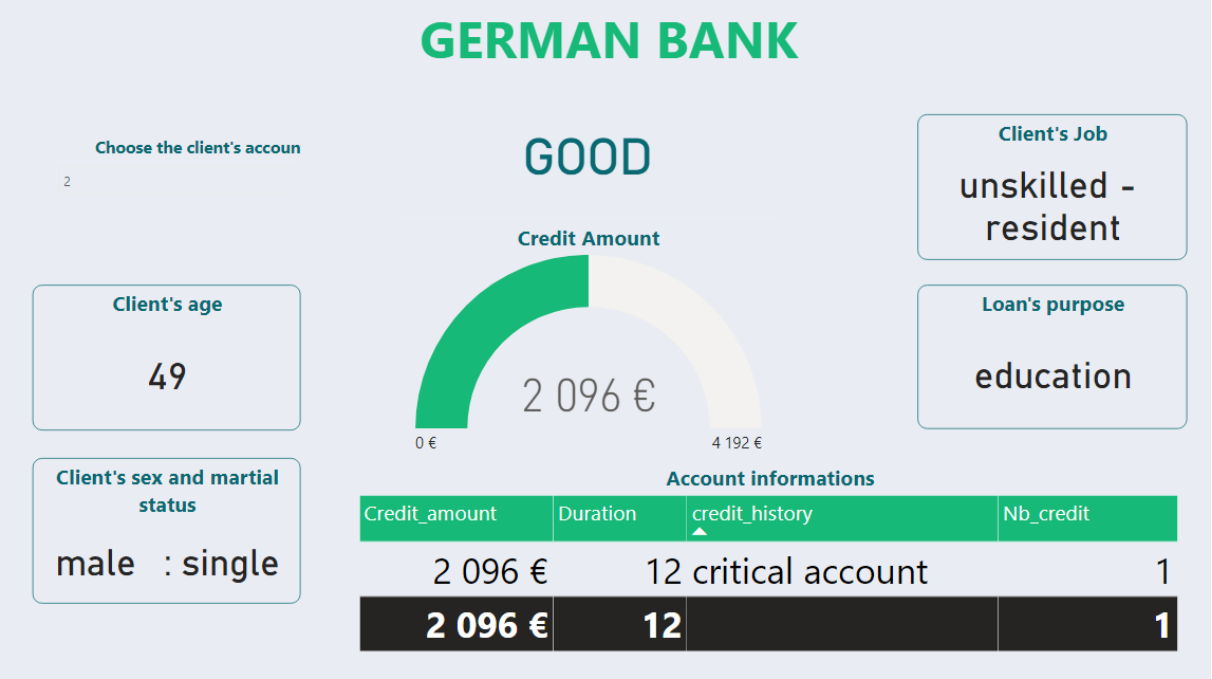
```
plt.figure(figsize=(10,10))
plt.plot(fpr1, tpr1, color='blue', lw=2, label='Support Vector Machine (area = %0.2f)'% roc_auc1)
plt.plot(fpr2, tpr2, color='green', lw=2, label='Random Forest ROC curve (area = %0.2f)'% roc_auc2)
plt.plot(fpr3, tpr3, color='cyan', lw=2, label='kNN ROC curve (area = %0.2f)'% roc_auc3)
plt.plot(fpr4, tpr4, color='magenta', lw=2, label='Naive Bayes (area = %0.2f)'% roc_auc4)
plt.plot(fpr5, tpr5, color='yellow', lw=2, label='Stochastic Gradient Decent (area = %0.2f)'% roc_auc5)
plt.plot(fpr6, tpr6, color='black', lw=2, label='Perceptron (area = %0.2f)'% roc_auc6)
plt.plot(fpr7, tpr7, color='#770088', lw=2, label='CART (area = %0.2f)'% roc_auc6)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Classifiers ROC curves')
plt.legend(loc = "lower right")
plt.show()
```



On a obtenu un résultat identique à celui du tableau comparatif. De ce fait le SVM est le modèle le plus adéquat.

4.5 Déploiement





Conclusion générale

Il est indispensable que les scores, d'une manière ou d'une autre , apprennent à reconnaître ces publics et à distinguer parmi ceux-ci les différents niveaux de risque... Car sans cela, ils resteront exclus pour de mauvaises raisons, simplement parce qu'ils n'ont pas intégré l'échantillon d'origine. Et cela, n'est-ce pas de la discrimination indirecte ?