UNIVERSITÉ IBA DER THIAM

UFR Sciences Et Technologie

DÉPARTEMENT INFORMATIQUE,

Rapport Final de Projet

Travaux Pratiques Python pour DATA SCIENCE

Premiére année de Master Informatique

Spécialité Intelligence Artificielle et Smart Tech

Sous le thème :

Pré-traitement des Données et Les méthodes de structuration et de classification en apprentissage supervisé

Présenté par :

1. Khadim NGOM

2. Abdoul Wahab LY

Année universitaire 2024-2025

Table des matières

INTRODUCTION

Ce rapport présente une étude pratique centrée sur les concepts fondamentaux du pré-traitement des données, de leur visualisation et de l'apprentissage automatique, avec un accent particulier sur la classification supervisée. Au cours de cette démarche, nous avons exploré une base de données réelle en procédant à son chargement, son nettoyage et sa visualisation afin d'en extraire des informations pertinentes. Par la suite, nous avons implémenté, entraîné et évalué trois algorithmes de classification complémentaires : les K-Plus Proches Voisins (KNN), les Arbres de Décision, et les Forêts Aléatoires.

L'objectif principal de ce travail pratique est de développer une compréhension approfondie des techniques essentielles de manipulation, d'analyse et de modélisation des données en utilisant Python. Nous cherchons également à analyser le fonctionnement de chaque modèle, à évaluer leurs performances, ainsi qu'à explorer leurs forces et limitations, notamment en termes de capacité à éviter le surapprentissage. Enfin, cette étude met en lumière l'impact de l'optimisation des hyperparamètres et de l'utilisation de méthodes d'ensemble sur la robustesse et la capacité de généralisation des modèles. Cette démarche s'inscrit dans une logique d'analyse exploratoire essentielle en data science, et ce rapport constitue une synthèse structurée des étapes parcourues, ainsi que des observations clés issues de cette expérience.

I Pré-traitement des Données : les méthodes de visualisation et de description

I.1 Chargement de la Base de Données

La base de données a été chargée à l'aide de la bibliothèque pandas, en spécifiant le séparateur de colonnes (;) car il s'agit d'un fichier CSV.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

hopital = pd.read_csv("satisfaction_hopital.csv", sep=";")
```

	service	sexe	age	profession	amelioration.sante	amelioration.moral	recommander	score.relation	score,information
0	3	0	41.0	4.0	1.0	0.0	1.0	36.0	22.0
1	3	1	29.0	8.0	2.0	3.0	2.0	33.0	36.0
2	3	1	83.0	2.0	2.0	1.0	2.0	40.0	37.0
3	3	0	66.0	3.0	2.0	0.0	2.0	32.0	35.0
4	3	1	84.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
529	8	1	81.0	NaN	NaN	NaN	NaN	NaN	NaN
530	8	0	69.0	NaN	NaN	NaN	NaN	NaN	NaN
531	8	1	30.0	4.0	2.0	1.0	1.0	32.0	NaN
532	8	1	30.0	8.0	1.0	2.0	2.0	30.0	34.0
533	8	0	50.0	NaN	NaN	NaN	NaN	NaN	NaN

534 rows × 9 columns

I.2 Pré-traitement des Données

Dans cette section, les étapes initiales de pré-traitement ont été réalisées.

I.2.1 Renommage des colonnes

Afin de faciliter la manipulation des données et d'améliorer la lisibilité des noms de colonnes, certaines colonnes contenant des points ou des virgules ont été renommées.

```
# Renommer les colonnes existantes
hopital.rename(columns={'amelioration.sante': 'amelioration_sante',
                        'amelioration.moral': 'amelioration_moral',
                       'score.relation': 'score_relation',
                        'score,information': 'score_information'}, inplace=True)
```

	service	sexe	age	profession	amelioration_sante	amelioration_moral	recommander	score_relation	score_information
0	3	0	41.0	4.0	1.0	0.0	1.0	36.0	22.0
1	3	1	29.0	8.0	2.0	3.0	2.0	33.0	36.0
2	3	1	83.0	2.0	2.0	1.0	2.0	40.0	37.0
3	3	0	66.0	3.0	2.0	0.0	2.0	32.0	35.0
4	3	1	84.0	NaN	NaN	NaN	NaN	NaN	NaN

I.2.2 Vérification de la taille du dataset

La fonction .shape a été utilisée pour obtenir les dimensions de la base de données, confirmant qu'elle contient 534 lignes et 9 colonnes.

I.2.3 Vérification des types de variables

Les types de données de chaque colonne ont été vérifiés pour s'assurer qu'ils sont appropriés pour les analyses futures. La plupart des variables sont de type numérique (float64 et int64).

I.2.4 Identification des valeurs manquantes

Il est crucial d'identifier la présence de valeurs manquantes (NaN) dans le dataset, car elles peuvent affecter les analyses. La méthode .isna() a été utilisée pour cela. Les résultats indiquent la présence de valeurs manquantes dans plusieurs colonnes.

hopital.isna()

	service	sexe	age	profession	amelioration_sante	amelioration_moral	recommander	score_relation	score_information
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	True	True	True	True	True	True
...	...	...	...	...	...	...	...	...	...
529	False	False	False	True	True	True	True	True	True
530	False	False	False	True	True	True	True	True	True
531	False	False	False	False	False	False	False	False	True
532	False	False	False	False	False	False	False	False	False
533	False	False	False	True	True	True	True	True	True

534 rows × 9 columns

Donc il y'a des valeurs manquantes.
Ensuite, on calcule le pourcentage des valeurs manquantes dans chaque colonne.

Pour savoir le pourcentage des valeurs manquantes en le triant
(hopital.isna().sum()/hopital.shape[0]).sort_values(ascending=True)

```
service                0.000000
sexe                   0.000000
age                    0.011236
profession             0.200375
recommander            0.241573
amelioration_moral     0.282772
amelioration_sante     0.295880
score_information      0.329588
score_relation         0.346442
dtype: float64
```

Les colonnes ayant des valeurs manquantes représentent moins de 50%, il n'est donc pas nécessaire de les éliminer. En revanche, nous allons supprimer les lignes qui contiennent des valeurs manquantes.

I.2.5 Eliminer les valeurs manquantes

Avant d'éliminer les valeurs manquantes on fait une copie du dataset avec **df_copy = hopital.copy()** Supprimer les lignes ou on observe les données manquantes en utilisant la fonction dropna

df_clean = df_copy.dropna(axis=0)
df_clean

	service	sexe	age	profession	amelioration_sante	amelioration_moral	recommander	score_relation	score_information
0	3	0	41.0	4.0	1.0	0.0	1.0	36.0	22.0
1	3	1	29.0	8.0	2.0	3.0	2.0	33.0	36.0
2	3	1	83.0	2.0	2.0	1.0	2.0	40.0	37.0
3	3	0	66.0	3.0	2.0	0.0	2.0	32.0	35.0
5	3	0	84.0	6.0	3.0	3.0	2.0	39.0	28.0
...	...	...	...	...	...	...	...	...	...
522	8	1	77.0	8.0	2.0	2.0	2.0	39.0	25.0
524	8	1	58.0	3.0	3.0	1.0	2.0	37.0	35.0
526	8	1	69.0	4.0	3.0	1.0	2.0	40.0	31.0
527	8	1	67.0	4.0	3.0	3.0	2.0	38.0	35.0
532	8	1	30.0	8.0	1.0	2.0	2.0	30.0	34.0

287 rows × 9 columns

On a utilisé shape pour vérifier que des lignes ont été supprimées, passant de 534 à 287 après l'exécution.

I.3 Réaliser un aperçu des données

I.3.1 Calculer les statistiques de bases

df_clean.describe()

	service	sexe	age	profession	amelioration_sante	amelioration_moral	recommander	score_relation	score_information
count	287.000000	287.000000	287.000000	287.000000	287.000000	287.000000	287.000000	287.000000	287.000000
mean	4.470383	0.445993	54.013937	4.379791	2.247387	1.686411	1.658537	35.038328	31.832753
std	2.255916	0.497943	16.669551	1.694915	0.769696	0.953054	0.543675	4.761654	6.719654
min	1.000000	0.000000	18.000000	1.000000	0.000000	0.000000	0.000000	13.000000	13.000000
25%	3.000000	0.000000	43.000000	3.000000	2.000000	1.000000	1.000000	33.000000	28.000000
50%	5.000000	0.000000	55.000000	4.000000	2.000000	1.000000	2.000000	36.000000	33.000000
75%	6.000000	1.000000	66.000000	5.000000	3.000000	3.000000	2.000000	39.000000	38.000000
max	8.000000	1.000000	89.000000	8.000000	3.000000	3.000000	2.000000	40.000000	40.000000

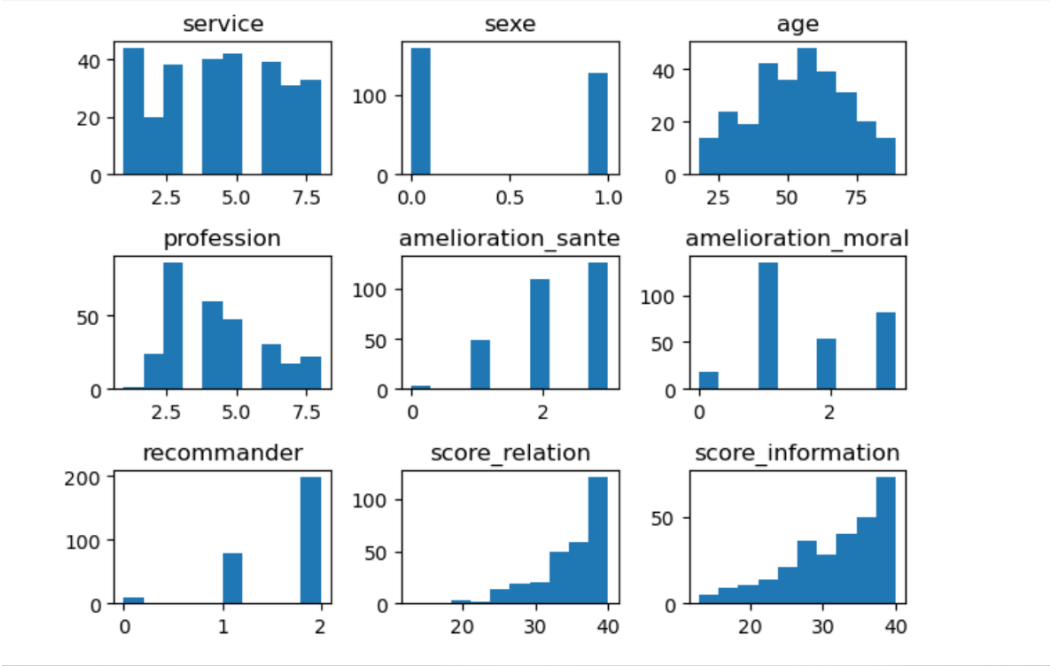
Le tableau suivant présente un résumé statistique des différentes variables issues de la base de données (287 observations) :

- **service** : variable discrète variant de 1 à 8.
  - Moyenne : 4,47 ; Médiane : 5 ; Écart-type : 2,26
  - Répartition globalement centrée, légère dispersion
- **sexe** : variable binaire (0 ou 1)
  - Moyenne : 0,45 ; Médiane : 0
  - Environ 45% des individus sont de sexe codé «1»
- **âge** : variable continue de 18 à 89 ans
  - Moyenne : 54 ans ; Médiane : 55 ; Écart-type : 16,67
  - Répartition assez étendue mais centrée
- **profession** : variable discrète
  - Moyenne : 4,38 ; Médiane : 4 ; Écart-type : 1,69
  - Données centrées
- **amélioration santé** : échelle de 0 à 3
  - Moyenne : 2,25 ; Médiane : 2
  - Les réponses sont concentrées vers des valeurs élevées
- **amélioration moral** : échelle de 0 à 3
  - Moyenne : 1,69 ; Médiane : 1
  - Répartition plus étalée, centrée sur une amélioration modérée
- **recommander** : variable binaire ou ternaire
  - Moyenne : 1,66 ; Médiane : 2
  - Tendance à recommander plutôt positive
- **score relation** : score de 13 à 40
  - Moyenne : 35 ; Médiane : 36 ; Écart-type : 4,76
  - Relations perçues globalement comme bonnes
- **score information** : score de 13 à 40
  - Moyenne : 31,83 ; Médiane : 33 ; Écart-type : 6,72
  - Informations jugées relativement complètes par les répondants

Toutes les variables sont complètes (aucune valeur manquante). Les données montrent une tendance globalement positive concernant la santé, les relations et les informations reçues.

I.3.2 .Visualiser les données en traçant les histogrammes de toutes les variables

```
for i in np.arange(1,10):
    plt.subplot(3, 3, i)
    plt.title(df_clean.columns[i−1])
    plt.hist(df_clean[df_clean.columns[i−1]])
    plt.tight_layout()
```



- **service** : Répartition du nombre de services, avec une distribution relativement uniforme, mais une légère concentration autour de 4 ou 5 services.
- **sexe** : Variable binaire, avec un grand nombre d'observations pour une catégorie (probablement homme ou femme) et beaucoup moins pour l'autre.
- **âge** : Variable continue avec une distribution approximativement normale, concentrée autour de 45 à 50 ans.
- **profession** : Répartition des différentes professions, avec une catégorie (probablement la plus courante) en nombre nettement plus élevé.
- **amelioration_sante** : Réponses ou évaluations liées à l'amélioration de la santé, avec des valeurs généralement faibles mais quelques variations.
- **amelioration_moral** : Réponses ou évaluations concernant l'amélioration morale, avec une prédominance de la valeur la plus basse.
- **recommander** : Mesure des recommandations, avec un grand nombre d'observations dans la catégorie la plus haute.
- **score_relation** : Score ou évaluation des relations, avec une distribution répartie mais concentrée davantage vers les scores faibles à moyens.
- **score_information** : Score ou évaluation de l'information, avec une distribution un peu dispersée mais tendance vers des scores élevés.

I.3.3 Étudier la répartition des âges selon le sexe par les boites à moustaches boxplot.

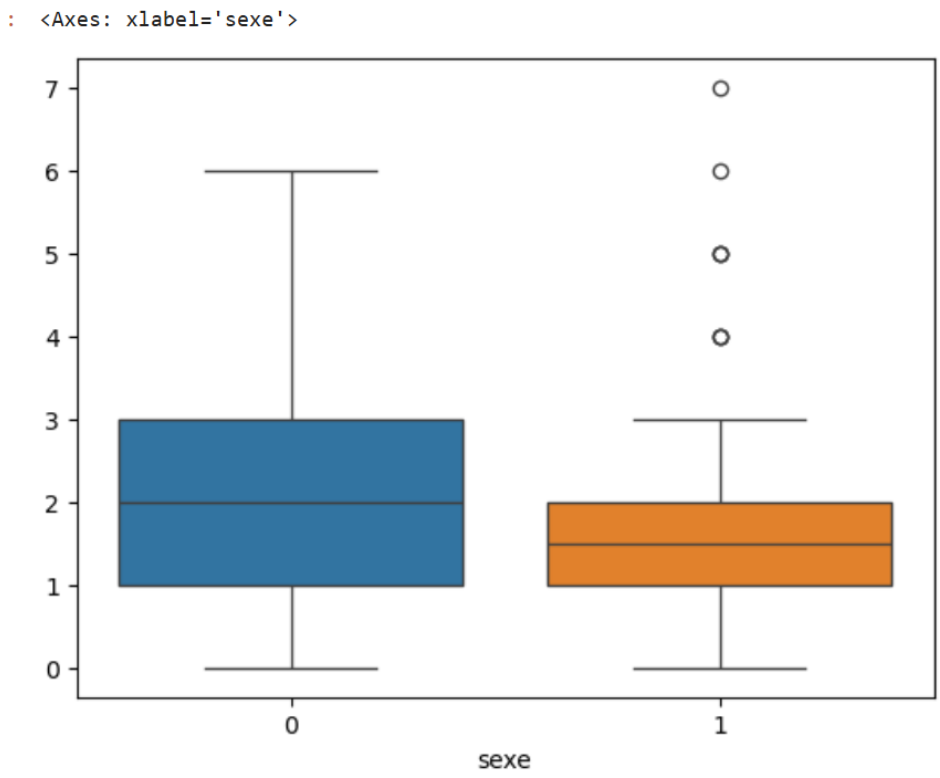
Utilisons crosstab pour calculer la répartition des âges selon le sexe avant de tracer le boxplot.

```
rep_age_sex = pd.crosstab(df_clean['age'],df_clean['sexe'])
rep_age_sex
```

sexe	0	1
age		
18.0	1	2
19.0	0	1
20.0	1	0
21.0	1	2
22.0	1	1
...	...	...
84.0	3	3
85.0	0	1
86.0	1	0
88.0	0	2
89.0	1	0

70 rows × 2 columns

sns.boxplot(rep_age_sex)



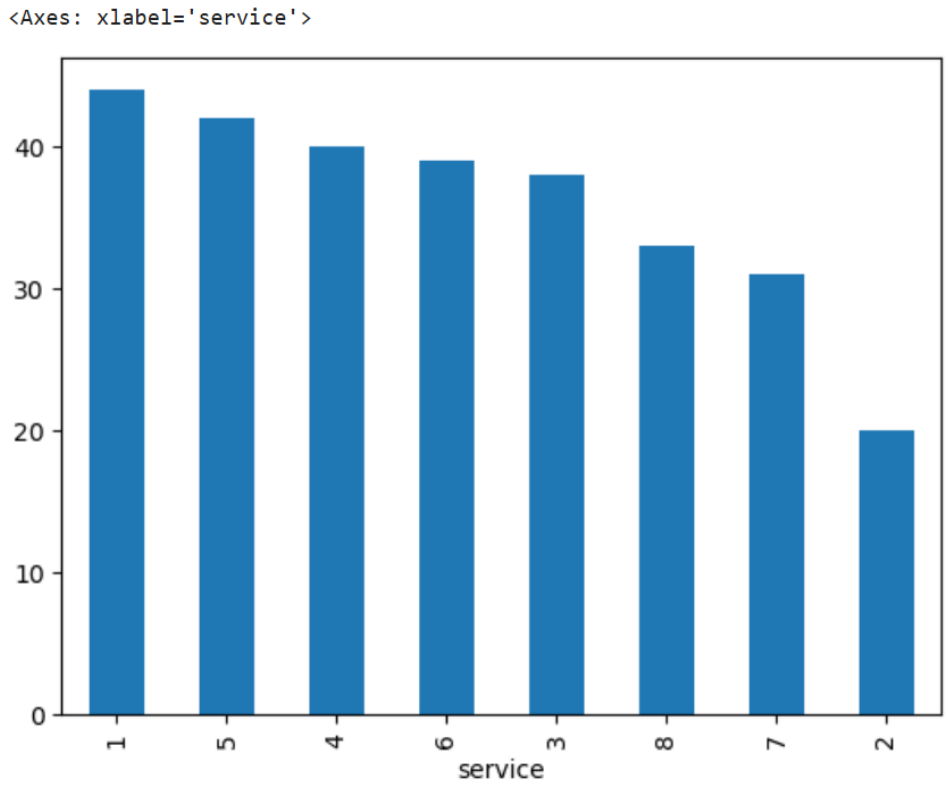
L'image présente un boxplot comparant la distribution des âges selon le sexe. Voici une analyse succincte :

— La distribution des âges semble différente entre les deux sexes.
— La médiane (ligne à l'intérieur des boîtes) indique que la majorité des âges pour le sexe 0 est légèrement inférieure à celle du sexe 1.

— La plage interquartile (largeur de la boîte) indique une distribution plus concentrée pour l'un des sexes, probablement celui avec la boîte plus petite.
— Il y a quelques points en dehors des whiskers (points en haut), suggérant la présence d'outliers, en particulier du côté du sexe 1, avec des valeurs de l'âge plus élevées.

I.3.4 Tracer le diagramme en barre de l'effectif de chaque numéro de service

df_clean['service'].value_counts().plot.bar()



Principales observations :

Le service 1 a la valeur la plus élevée, environ 45. Les services 5 et 4 suivent, avec des valeurs légèrement inférieures à 45 et 40.
Les autres services montrent une tendance décroissante, allant du service 3 au service 2.
Le service 2 a la valeur la plus basse, autour de 20.

Globalement, on observe une tendance décroissante des valeurs du service 1 au service 2, avec le service 1 étant le plus élevé et le service 2 le plus bas.

I.4 Traiter les données en remplaçant les données manquantes de chaque variable par sa moyenne.

Avant tout, on fait une copie du dataset original.

data = hopital.copy()

On remplace les données manquantes de chaque variable par la moyenne en utilisant fillna

data_clean = data.fillna(data[data.columns].mean())
data_clean

	service	sexe	age	profession	amelioration_sante	amelioration_moral	recommander	score_relation	score_information
0	3	0	41.0	4.000000	1.000000	0.000000	1.000000	36.000000	22.000000
1	3	1	29.0	8.000000	2.000000	3.000000	2.000000	33.000000	36.000000
2	3	1	83.0	2.000000	2.000000	1.000000	2.000000	40.000000	37.000000
3	3	0	66.0	3.000000	2.000000	0.000000	2.000000	32.000000	35.000000
4	3	1	84.0	4.430913	2.231383	1.678851	1.624691	35.217765	31.910615
...	...	...	...	...	...	...	...	...	...
529	8	1	81.0	4.430913	2.231383	1.678851	1.624691	35.217765	31.910615
530	8	0	69.0	4.430913	2.231383	1.678851	1.624691	35.217765	31.910615
531	8	1	30.0	4.000000	2.000000	1.000000	1.000000	32.000000	31.910615
532	8	1	30.0	8.000000	1.000000	2.000000	2.000000	30.000000	34.000000
533	8	0	50.0	4.430913	2.231383	1.678851	1.624691	35.217765	31.910615

534 rows × 9 columns

I.5 Effectuer une réduction de dimension de la base par l'application de l'Analyse en composantes principales

I.5.1 Analyser le nuage des variables (cercle des corrélations)

On importe les bibliothéque sklearn pour faire de l'ACP

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Standardisons les données

```
scaler = StandardScaler()
data_clean_scaled = scaler.fit_transform(data_clean)
```

```
array([[-6.80687025e-01, -9.96261669e-01, -9.72868445e-01, ...,
        -1.28119282e+00,  2.09574017e-01, -1.78425392e+00],
       [-6.80687025e-01,  1.00375236e+00, -1.65113588e+00, ...,
         7.69728490e-01, -5.94176884e-01,  7.36231044e-01],
       [-6.80687025e-01,  1.00375236e+00,  1.40106758e+00, ...,
         7.69728490e-01,  1.28124189e+00,  9.16265685e-01],
       ...,
       [ 1.51693614e+00,  1.00375236e+00, -1.59461360e+00, ...,
        -1.28119282e+00, -8.62093851e-01,  6.39611530e-16],
       [ 1.51693614e+00,  1.00375236e+00, -1.59461360e+00, ...,
         7.69728490e-01, -1.39792778e+00,  3.76161763e-01],
       [ 1.51693614e+00, -9.96261669e-01, -4.64167867e-01, ...,
         4.55396011e-16,  0.00000000e+00,  6.39611530e-16]])
```

Appliquons l'ACP avec deux composantes

```
pca = PCA(n_components=2)
data_clean_pca = pca.fit_transform(data_clean_scaled)
```

```
data_clean_pca
```

```
[302]:

array([[-2.92507255, -0.99082133],
       [ 0.73656203,  2.63165782],
       [ 0.84480793, -1.13293614],
       ...,
       [-1.48959562,  0.79897987],
       [-0.75710759,  2.80448062],
       [ 0.18739836, -0.20675023]])

[304]:
```

Créeons un dataframe du résultat

```
df_pca = pd.DataFrame(data_clean_pca, columns=["C1", "C2"],
        index=data_clean.index)
print(df_pca)
```

```
           C1        C2
0   -2.925073 -0.990821
1    0.736562  2.631658
2    0.844808 -1.132936
3   -0.789972 -1.806087
4   -0.101407  0.183824
..        ...       ...
529  0.077023  0.625975
530  0.206303 -0.405210
531 -1.489596  0.798980
532 -0.757108  2.804481
533  0.187398 -0.206750

[534 rows x 2 columns]
```

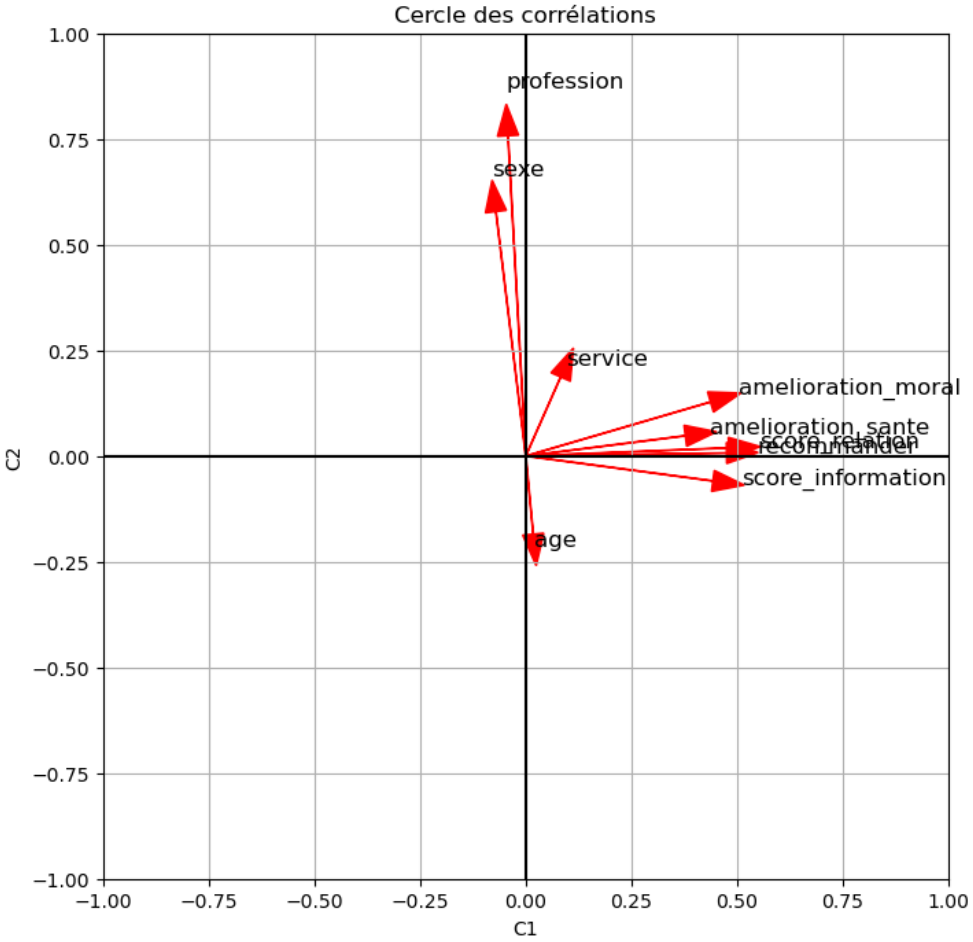
Cercle des corrélations

```
def plot_correlation_circle(pca, features):
   pcs = pca.components_
   plt.figure(figsize=(8, 8))
   for i, (x, y) in enumerate(zip(pcs[0, :], pcs[1, :])):
      plt.arrow(0, 0, x, y, head_width=0.05, color='r')
```

```
        plt.text(x * 1.15, y * 1.15, features[i], fontsize=12)

    plt.xlabel("C1")
    plt.ylabel("C2")
    plt.title("Cercle des corrélations")
    plt.grid()
    plt.axhline(0, color='black')
    plt.axvline(0, color='black')
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.show()

plot_correlation_circle(pca, data_clean.columns)
```



— **Variables bien représentées (près du bord du cercle) ?** Les variables bien représentées sont celles qui ont une longueur de vecteur proche de 1, c'est-à-dire situées près du bord du cercle. Elles sont fortement corrélées avec le plan formé par la composante 1 (C1) et la composante 2 (C2).

amelioration_moral, amelioration_sante, recommander, score_relation et score_information sont bien représentées. Leurs flèches sont longues et pointent vers la droite, ce qui montre une forte contribution au premier **axe (C1)**.

sexe et profession sont bien représentées. Leurs flèches sont longues et pointent vers le haut, ce qui montre une forte contribution au deuxieme **axe (C2)**.

— **Variables moins bien représentées (près du centre) ?**

Les variables mal représentées sont courtes ou proches du centre du cercle. Elles ont une faible contribution à C1 et C2, donc leur interprétation dans ce plan est limitée.

age et service sont mal représentées.

— **Variables fortement corrélées aux axes ?**

Axe 1 (C1) :

Variables fortement corrélées positivement sont **amelioration_moral, amelioration_sante, recommander, score_relation, score_information**

Axe 2 (C2) :

Variables corrélées positivement sont **sexe, profession**

age corrélées negativement

— **Interprétation qualitative des axes ? Composante 1 (C1)** : Représente une dimension "bénéfice perçu" ou "efficacité perçue"

Les variables **amelioration_moral, santé, score_information, recommander, score_relation** indiquent que cet axe capte la perception positive de l'intervention ou du dispositif évalué.

Plus une observation est positionnée à droite de C1, plus elle a probablement perçu des effets bénéfiques.

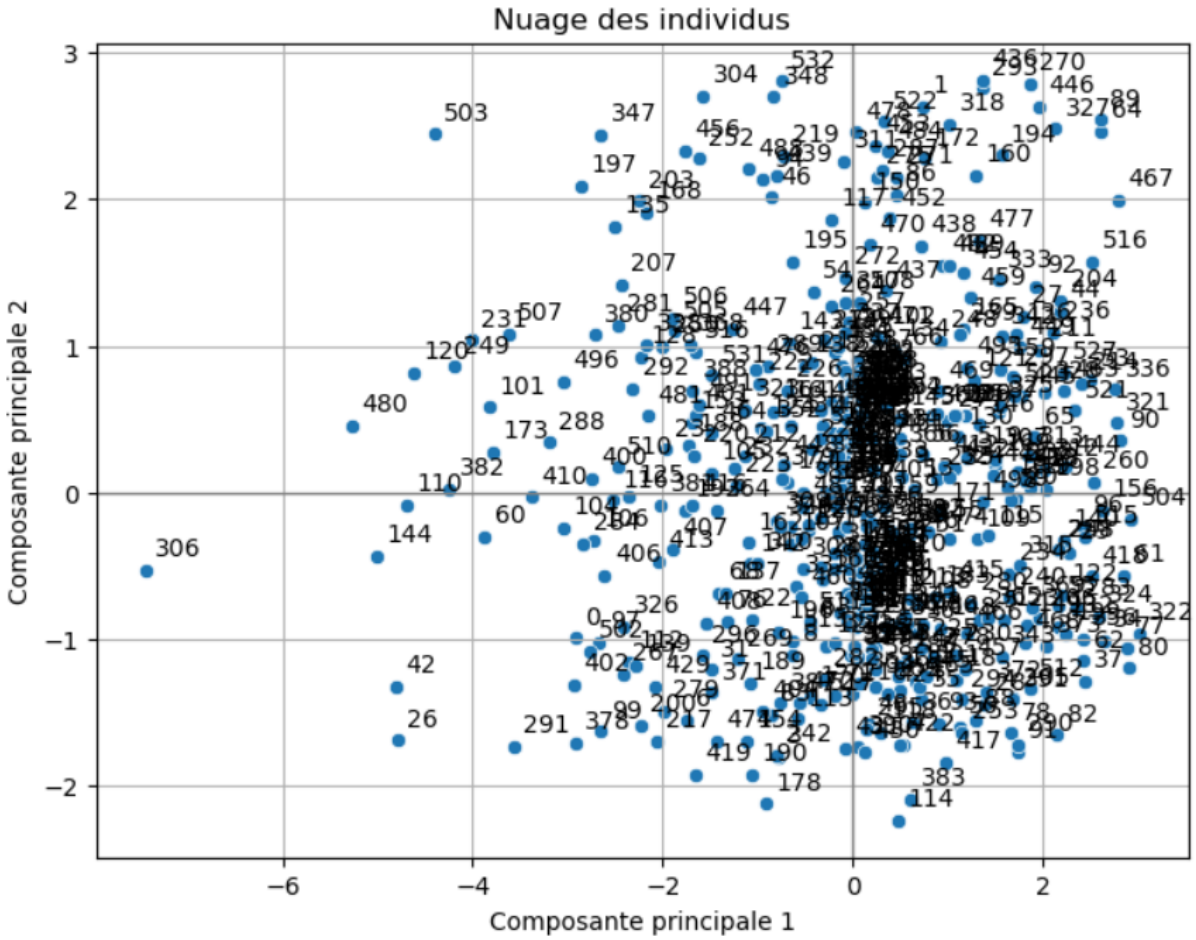
Composante 2 (C2) : Semble capter une dimension sociodémographique

âge (corrélé négativement), sexe et profession (positivement) suggèrent que les caractéristiques personnelles ou sociales influencent cette dimension.

Cela peut représenter par exemple une opposition entre **jeunes/hommes/-certaines professions et personnes plus âgées/femmes/autres professions**.

I.5.2 Analyser le nuage d'individus

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x="C1", y="C2", data=df_pca)
for i in df_pca.index:
   plt.text(df_pca.loc[i, "C1"]+0.1, df_pca.loc[i, "C2"]+0.1, i)
plt.axhline(0, color='gray', lw=1)
plt.axvline(0, color='gray', lw=1)
plt.title("Nuage des individus")
plt.xlabel("Composante principale 1")
plt.ylabel("Composante principale 2")
plt.grid()
plt.show()
```



— **Aspect général du nuage ?** Le nuage a une forme allongée verticalement, avec une forte concentration des individus autour de la **composante 1 = 0**.

Cela indique que la **composante 1 (axe horizontal)** n'explique qu'une partie modérée de la variabilité globale.

La majorité des individus ont une valeur similaire sur l'axe 1, mais une variabilité importante sur l'axe 2 (vertical).

— **Des groupes semblent-ils se former ?** On observe une forte densité centrale, mais aucun groupe bien distinct ne ressort visuellement.

Cela peut indiquer :

— soit une population homogène sans sous-groupes nets,

— soit une structure latente plus complexe non révélée par les deux premiers axes.

Cependant, quelques points isolés aux extrémités (ex. : à gauche ou en haut) peuvent appartenir à des sous-groupes rares ou atypiques.

— **Individus mal représentés ?** Les individus proches du centre du graphique (autour de (0,0)) sont mal représentés dans ce plan factoriel.

Cela signifie que leur qualité de représentation ($\cos^2$) est faible sur les deux premiers axes.

Ces individus contribuent peu à la structure révélée par l'ACP → leur profil est mieux décrit par d'autres axes (PC3, PC4…).

— **Proximité en projection mais pas dans les données initiales ?** Oui, c'est possible :

— L'ACP projette les individus dans un espace réduit (2D), donc certaines distances sont approximées.

— Deux individus proches dans le plan (1,2) peuvent en réalité être différents sur les variables peu représentées par ces deux axes.

Par exemple : Les points numérotés 101 et 103 peuvent sembler proches ici, mais peuvent être différents dans les variables comme "âge" ou "sexe", peu représentées par C1/C2.

I.5.3 Les variables qui contribuent pour la composition de chaque composante principale

```
# Contributions (carrés des composantes)
contrib = pd.DataFrame(np.square(pca.components_),
  columns=data_clean.columns, index=["C1", "C2"])
print(contrib.T.sort_values("C1", ascending=False))
```

	C1	C2
score_relation	0.230586	0.000369
recommander	0.225692	0.000051
score_information	0.196123	0.003474
amelioration_moral	0.191140	0.016338
amelioration_sante	0.142584	0.002290
service	0.006815	0.034945
sexe	0.004986	0.334384
profession	0.001765	0.573998
age	0.000310	0.034151

score_relation, recommander, score_information, amelioration_moral, amelioration_sante contribuent pour la composante principale **C1**

sexe, profession pour la composante **C2**

I.5.4 Déduction des variables les plus pertinentes

```
contrib["Total"] = contrib.sum(axis=1)
total_contrib = contrib.sort_values("Total", ascending=False)
print(total_contrib.T)
```

	C1	C2
service	0.006815	0.034945
sexe	0.004986	0.334384
age	0.000310	0.034151
profession	0.001765	0.573998
amelioration_sante	0.142584	0.002290
amelioration_moral	0.191140	0.016338
recommander	0.225692	0.000051
score_relation	0.230586	0.000369
score_information	0.196123	0.003474
Total	1.000000	1.000000

Les variables les plus pertinentes sont **score_relation, recommander, score_information, amelioration_moral, amelioration_sante,sexe, profession**

I.5.5 Synthèse générale des pré-traitements

1. **Nettoyage des données**

   Suppression et remplacement des valeurs manquantes : Des vérifications ont été faites pour identifier les NaN. Les lignes ou colonnes avec des valeurs manquantes non pertinentes ont été éliminées et ensuite remplaçés par leurs moyennes.

   Correction de noms de variables incohérents : Exemple : Les variables ont été renommées correctement si besoin pour garantir l'uniformité dans le traitement

2. **Statistiques de base**

   Utiliser describe pour calculer les statistiques de bases

3. **Visualisations**

   Visualiser les données avec des histogrammes et boxplot

4. **Centrage et réduction des variables**

   Étape indispensable pour l'ACP :

   Toutes les variables ont été centrées (moyenne = 0) et réduites (écart-type = 1) afin d'avoir le même poids dans l'analyse, peu importe leur échelle initiale.

5. **Vérification de la qualité des variables**

   Analyse de la variance et des corrélations pour évaluer si certaines variables étaient trop peu informatives ou redondantes.

6. **Préparation à l'ACP**

   Calcul des contributions des variables aux axes principaux (C1 et C2).

   Interprétation des valeurs propres et choix du nombre d'axes à conserver.

   Réduction potentielle de la base aux variables les plus contributives pour simplifier l'analyse

II Les méthodes de structuration et de classification en apprentissage supervisé

Chargeons les donnéés

```
sonar = pd.read_csv("sonar.all−data.csv",
                    names=["F1", "F2", "F3", "F4", "F5",
                    "F6", "F7", "F8", "F9", "F10", "F11",
                     "F12", "F13", "F14", "F15", "F16", "F17",
                      "F18", "F19", "F20", "F21", "F22", "F23",
                       "F24", "F25", "F26", "F27", "F28", "F29",
                        "F30", "F31", "F32", "F33", "F34",
                         "F35",  "F36", "F37", "F38", "F39",
                         "F40", "F41", "F42", "F43", "F44", "F45",
                         "F46",   "F47", "F48", "F49","F50,
                         "F51","F52", "F53", "F54", "F55",
                         "F56", "F57", "F58", "F59", "F60", "OBJECT"])
```

```
sonar.head()
```

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	...	F52	F53	F54	F55	F56	F57	F58	F59	F60	OBJECT
0	0.0200	0.0371	0.0428	0.0207	0.0954	0.0986	0.1539	0.1601	0.3109	0.2111	...	0.0027	0.0065	0.0159	0.0072	0.0167	0.0180	0.0084	0.0090	0.0032	R
1	0.0453	0.0523	0.0843	0.0689	0.1183	0.2583	0.2156	0.3481	0.3337	0.2872	...	0.0084	0.0089	0.0048	0.0094	0.0191	0.0140	0.0049	0.0052	0.0044	R
2	0.0262	0.0582	0.1099	0.1083	0.0974	0.2280	0.2431	0.3771	0.5598	0.6194	...	0.0232	0.0166	0.0095	0.0180	0.0244	0.0316	0.0164	0.0095	0.0078	R
3	0.0100	0.0171	0.0623	0.0205	0.0205	0.0368	0.1098	0.1276	0.0598	0.1264	...	0.0121	0.0036	0.0150	0.0085	0.0073	0.0050	0.0044	0.0040	0.0117	R
4	0.0762	0.0666	0.0481	0.0394	0.0590	0.0649	0.1209	0.2467	0.3564	0.4459	...	0.0031	0.0054	0.0105	0.0110	0.0015	0.0072	0.0048	0.0107	0.0094	R

5 rows × 61 columns

- **Combien de classes ?**
  Nous avons deux classes : **M** et **R**
- **Combien de caractéristiques descriptives ? De quels types ?**

  sonar.dtypes.value_counts()

```
float64    60
object      1
Name: count, dtype: int64
```

On a 60 Variables quantitatives et 1 Variables qualitatives

- **Calculer les statistiques de base des variables 2 à 7**

  (sonar[sonar.columns[2:8]]).describe()

	F3	F4	F5	F6	F7	F8
count	208.000000	208.000000	208.000000	208.000000	208.000000	208.000000
mean	0.043832	0.053892	0.075202	0.104570	0.121747	0.134799
std	0.038428	0.046528	0.055552	0.059105	0.061788	0.085152
min	0.001500	0.005800	0.006700	0.010200	0.003300	0.005500
25%	0.018950	0.024375	0.038050	0.067025	0.080900	0.080425
50%	0.034300	0.044050	0.062500	0.092150	0.106950	0.112100
75%	0.057950	0.064500	0.100275	0.134125	0.154000	0.169600
max	0.305900	0.426400	0.401000	0.382300	0.372900	0.459000

- **Combien d'exemples ?**

  sonar.shape

```
(208, 61)
```

208 exemples

- **Combien d'exemples de chaque classe ?**

  sonar['OBJECT'].value_counts()

```
OBJECT
M    111
R     97
Name: count, dtype: int64
```

classe M : 111 exemples

classe R : 97 exemples

- **Comment sont organisés les exemples ?**

  Les exemples sont organisés dans un format tabulaire, où :

  Chaque **ligne** représente un seul exemple ou une observation.

  Chaque **colonne** représente une caractéristique spécifique (caractéristique descriptive) ou la classe cible ('OBJECT').

  Les caractéristiques sont numériques, et la dernière colonne contient l'étiquette de classe pour chaque exemple.

II.1 Séparation des données en bases d'apprentissage et de test

Avant de commencer, faisons l'encodage de la variable objet en remplaçant M par 1 et R par 0.

```
code = {
      'M': 1,
      'R': 0,
   }
for col in sonar.select_dtypes('object'):
   sonar[col] = sonar[col].map(code)
```

Ensuite, donnons X comme les features et y comme la target.

```
y=sonar['OBJECT']
X=sonar[sonar.columns[:-1]]
```

Enfin, séparons les données en bases d'apprentissage et de test

```
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=5)
print('X_train_set:',X_train.shape)
print('X_test_set:',X_test.shape)
print('y_train_set:',y_train.shape)
print('y_test_set:',y_test.shape)
```

```
X_train_set: (145, 60)
X_test_set: (63, 60)
y_train_set: (145,)
y_test_set: (63,)
```

II.2 Modélisaation avec les k plus proches voisins

II.2.1 Entraînement du Modèle Initial

Un modèle KNeighborsClassifier a été initialisé et entraîné sur les données d'entraînement (X_train,y_train).

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train,y_train)
```

Les performances initiales du modèle sur les ensembles d'entraînement et de test ont été évaluées :

Score d'entraînement : 0.834

Score de test : 0.746

Ces scores indiquent une performance correcte, mais un certain degré de surapprentissage (overfitting) peut être suspecté, étant donné que le score d'entraînement est significativement plus élevé que le score de test.

La matrice de confusion pour les prédictions sur l'ensemble de test est la suivante :

```
from sklearn.metrics import confusion_matrix
confusion_matrix(expected,predicted)
```

```
array([[16, 12],
       [ 4, 31]], dtype=int64)
```

Cette matrice nous montre que :

- 16 observations de la classe 0 ont été correctement classifiées (vrais négatifs).
- 12 observations de la classe 0 ont été incorrectement classifiées comme classe 1 (faux positifs).
- 4 observations de la classe 1 ont été incorrectement classifiées comme classe 0 (faux négatifs).
- 31 observations de la classe 1 ont été correctement classifiées (vrais positifs).

II.2.2 Optimisation des Hyperparamètres avec GridSearchCV

Afin d'améliorer les performances du modèle et de réduire potentiellement le sur-apprentissage, une recherche par grille (GridSearchCV) a été effectuée pour trouver les meilleurs hyperparamètres pour le KNeighborsClassifier. Les hyperparamètres testés étaient le nombre de voisins (k) allant de 1 à 90, et les métriques de distance ('euclidean' et 'manhattan').

```
from sklearn.model_selection import GridSearchCV
param_grid={'n_neighbors':np.arange(1,90),'metric': ['euclidean','manhattan']}
grid=GridSearchCV(KNeighborsClassifier(),param_grid,cv=4)
grid.fit(X_train,y_train)
```

Les meilleurs hyperparamètres trouvés par GridSearchCV sont :

'metric' : 'euclidean'

'n_neighbors' : 2 Le meilleur score obtenu lors de cette recherche par grille était de **0.841**. Le modèle best_estimator_ de la grille, qui est le KNeighborsClassifier entraîné avec ces meilleurs hyperparamètres, a été sélectionné pour l'analyse ultérieure.

```
model=grid.best_estimator_
```

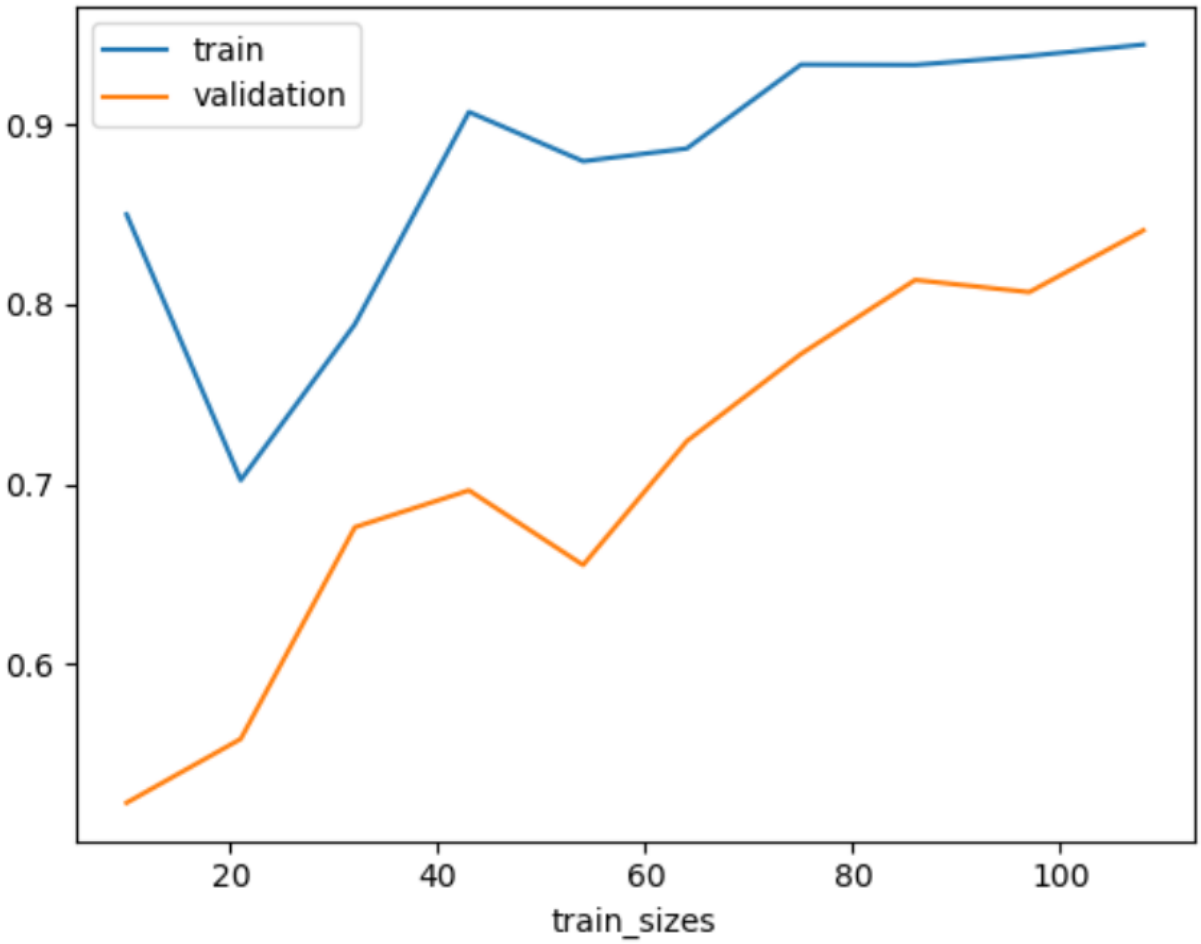
II.2.3 Analyse de la Courbe d'Apprentissage

Une courbe d'apprentissage a été générée pour évaluer comment les performances du modèle évoluent avec l'augmentation de la taille de l'ensemble d'entraînement.

```
from sklearn.model_selection import learning_curve
N,train_score,val_score=learning_curve(model,X_train, y_train,
      train_sizes=np.linspace(0.1,1.0,10),cv=4)

plt.plot(N,train_score.mean(axis=1),label='train')
plt.plot(N,val_score.mean(axis=1),label='validation')
plt.xlabel('train_sizes')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x24d36414cb0>
```



L'observation de la courbe d'apprentissage révèle que :

- **Le score d'entraînement (courbe bleue)** : Démarre à un niveau élevé et diminue légèrement à mesure que la taille de l'ensemble d'entraînement augmente. Cela est typique, car le modèle a plus de difficultés à parfaitement mémoriser un ensemble de données plus grand.
- **Le score de validation (courbe orange)** : Commence à un niveau plus bas et augmente progressivement avec l'augmentation de la taille de l'ensemble d'entraînement. C'est un bon signe, indiquant que le modèle généralise mieux avec plus de données.
- **L'écart entre les courbes** : Il y a un écart significatif entre le score d'entraînement et le score de validation, ce qui suggère toujours un certain degré de surapprentissage. Cependant, les deux courbes semblent converger lentement, indiquant que l'ajout de plus de données pourrait potentiellement améliorer davantage la performance de généralisation.

II.3 Analyse et Interprétation d'un Modèle d'Arbre de Décision

II.3.1 Entraînement du Modèle et Évaluation Initiale

Un modèle DecisionTreeClassifier a été initialisé et entraîné sur les données d'entraînement (X_train,y_train).

from sklearn.tree import DecisionTreeClassifier, plot_tree

```
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

Les performances initiales du modèle sur les ensembles d'entraînement et de test sont les suivantes :

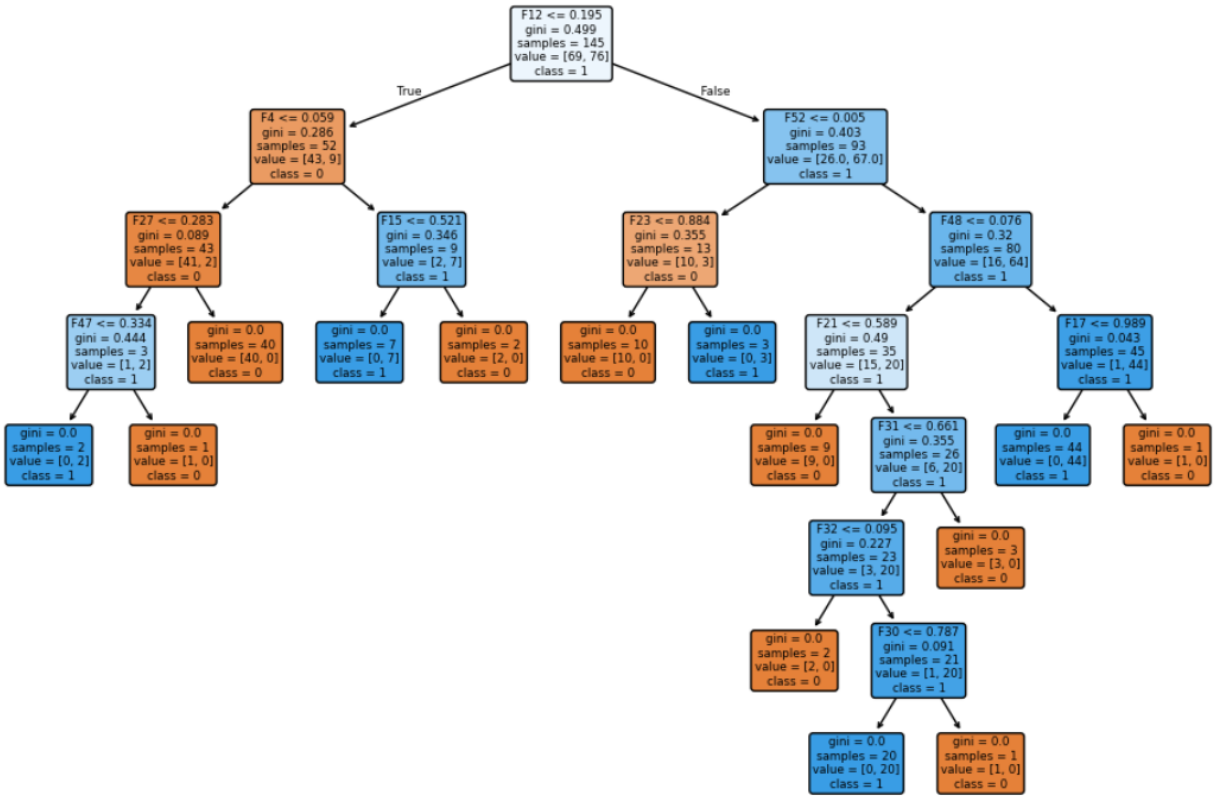
Score d'entraînement : 1.0

Score de test : 0.650

Le score d'entraînement de 1.0 indique que le modèle a parfaitement appris les données d'entraînement. Cependant, le score de test significativement plus bas (0.650) révèle un problème de surapprentissage (overfitting). Cela signifie que le modèle est devenu trop complexe et a mémorisé le bruit et les spécificités des données d'entraînement, le rendant moins performant sur de nouvelles données non vues.

II.3.2 Visualisation et Interprétation de l'Arbre de Décision

```
plt.figure(figsize=(12,8))
plot_tree(model,
          feature_names=X.columns,
          class_names=list(map(str, pd.Categorical(y).categories)),
          filled=True,
          rounded=True
)
```



Observations et Caractéristiques de l'Arbre :

Profondeur de l'arbre : L'arbre généré est relativement profond et complexe, avec de nombreuses ramifications. Cette complexité est la cause probable du surapprentissage observé. Un arbre profond suggère qu'il a appris des motifs très spécifiques dans les données d'entraînement, plutôt que des relations plus générales.

Conditions de division (Splitting Conditions) : Chaque nœud interne (nœud de décision) est basé sur une caractéristique (par exemple, F12, F4, F52, etc.) et un seuil

(par exemple, F12 <= 0.195, F4 <= 0.059). C'est ainsi que l'arbre divise les données à chaque étape.

Impureté Gini : Le "gini" affiché dans chaque nœud est l'indice d'impureté de Gini. Il mesure la probabilité qu'un élément choisi au hasard dans l'ensemble soit mal classé si classé au hasard selon la distribution des classes dans le nœud. Un gini de 0.0 indique un nœud pur (tous les échantillons appartiennent à la même classe), ce qui est l'objectif des divisions.

Nombre d'échantillons (samples) : Le nombre d'échantillons qui arrivent à chaque nœud est indiqué. Cela permet de voir comment les données sont réparties à travers l'arbre.

Valeur (value) : La section "value" indique la distribution des classes pour les échantillons qui atteignent ce nœud. Par exemple, value = [69, 76] signifie qu'il y a 69 échantillons de la classe 0 et 76 échantillons de la classe 1 dans ce nœud.

Classe prédite (class) : La "class" indique la classe majoritaire à ce nœud (la prédiction de l'arbre pour ce nœud).

Interprétation des Caractéristiques Importantes :

En examinant la structure de l'arbre, nous pouvons identifier les caractéristiques les plus discriminantes :

F12 et F52 : Ces deux caractéristiques sont au premier niveau de l'arbre (nœud racine et son premier split droit). Cela signifie qu'elles sont les caractéristiques les plus importantes car elles sont utilisées pour la division initiale et affectent le plus grand nombre d'échantillons. F12 semble être la caractéristique la plus importante car elle est au tout premier nœud.

F4 : Est également très important car c'est le premier split après F12 sur le chemin "True".

F27, F15, F23, F48, F21, F17 : Ces caractéristiques sont importantes à un niveau secondaire, car elles divisent des sous-ensembles de données après les premières divisions.

F47, F31, F32, F30 : Ces caractéristiques semblent être importantes pour des divisions plus spécifiques et plus profondes, affectant des sous-ensembles plus petits d'échantillons.

II.4 Analyse et Évaluation Forêt Aléatoire

II.4.1 Entraînement et Évaluation du Modèle de Forêt Aléatoire

Conformément aux recommandations pour adresser le surapprentissage de l'arbre de décision, un modèle RandomForestClassifier a été entraîné avec n_estimators=100 (100 arbres dans la forêt).

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(n_estimators=100)
model.fit(X_train, y_train)
```

Les performances du modèle de Forêt Aléatoire sont les suivantes :

Score d'entraînement : 1.0

Score de test : 0.793

Bien que le score d'entraînement reste parfait, le score de test **(0.793)** est nettement supérieur à celui de l'Arbre de Décision unique **(0.650)**, ce qui démontre l'efficacité des forêts aléatoires pour réduire le surapprentissage.

Le rapport de classification détaillé sur l'ensemble de test est présenté ci-dessous :

```
from sklearn.metrics import classification_report
print(classification_report(expected, predicted))
```

	precision	recall	f1-score	support
0	0.80	0.71	0.75	28
1	0.79	0.86	0.82	35
accuracy			0.79	63
macro avg	0.79	0.79	0.79	63
weighted avg	0.79	0.79	0.79	63

Précision (Precision) : Pour la classe 0, 80

Rappel (Recall) : Pour la classe 0, 71

F1-score : C'est la moyenne harmonique de la précision et du rappel. Il est de 0.75 pour la classe 0 et 0.82 pour la classe 1, indiquant un bon équilibre entre précision et rappel pour les deux classes, avec une performance légèrement meilleure pour la classe 1.

Support : Le nombre réel d'occurrences de chaque classe dans l'ensemble de test (28 pour la classe 0, 35 pour la classe 1).

Accuracy (Exactitude) : L'exactitude globale du modèle sur l'ensemble de test est de 0.79 (79

Macro Avg et Weighted Avg : Ces moyennes confirment la performance équilibrée du modèle pour les deux classes.

II.5 Conclusion Générale et Comparaison des Modèles

Le modèle **KNN** a montré une performance initiale solide (score de test de 0.746) et a été légèrement amélioré par l'optimisation des hyperparamètres (meilleur score de validation croisée de 0.841). Il sert de bonne référence pour les modèles basés sur la distance.

L'Arbre de Décision simple, bien que très interprétable, a souffert d'un surapprentissage sévère (score de test de 0.650), en raison de sa profondeur et de sa capacité à mémoriser les données d'entraînement. L'analyse visuelle a permis d'identifier les caractéristiques clés.

La **Forêt Aléatoire** a démontré une amélioration significative par rapport à l'arbre de décision unique, atteignant un score de test de 0.793. Cela confirme son efficacité à réduire le surapprentissage grâce à l'agrégation de multiples arbres de décision et à l'introduction de la randomisation. Les métriques du rapport de classification pour la Forêt Aléatoire sont également très bonnes, avec un bon équilibre entre précision et rappel pour les deux classes.

En conclusion, parmi les modèles évalués, le **RandomForestClassifier** offre la meilleure performance de généralisation, démontrant sa robustesse face au surapprentissage par rapport à un arbre de décision unique. Pour des travaux futurs, une optimisation des hyperparamètres pour le RandomForestClassifier via **GridSearchCV** ou **RandomizedSearchCV** pourrait encore améliorer ses performances. Il serait également pertinent de comparer ces résultats avec d'autres algorithmes d'apprentissage automatique pour identifier le modèle le plus adapté à la tâche.

CONCLUSION

Ce travail pratique nous a permis d'acquérir une compréhension approfondie et concrète des étapes essentielles de l'analyse de données et de l'apprentissage automatique. Nous avons débuté par une phase de pré-traitement et de visualisation des données, qui s'est avérée cruciale pour explorer la structure de la base, identifier d'éventuelles anomalies et mieux comprendre les variables en jeu. Ces étapes ont posé les fondations nécessaires pour une modélisation efficace.

Ensuite, la mise en œuvre de trois algorithmes de classification — K-Plus Proches Voisins, Arbres de Décision et Forêts Aléatoires — a illustré l'importance de la sélection et de l'optimisation des modèles. Le KNN, simple mais performant, a servi de référence solide, tandis que l'Arbre de Décision a montré sa transparence mais aussi ses limites face au surapprentissage. Enfin, l'utilisation des Forêts Aléatoires a permis de tirer parti des méthodes d'ensemble pour améliorer la robustesse et la généralisation, confirmant leur supériorité dans notre contexte.

Ce travail a ainsi renforcé nos compétences en manipulation, visualisation et modélisation des données, tout en soulignant l'équilibre nécessaire entre la complexité du modèle, la performance et la capacité à éviter le surapprentissage. Il a aussi mis en lumière l'impact positif des méthodes d'ensemble pour répondre aux défis du machine learning. En définitive, cette expérience nous rappelle que la réussite en data science repose autant sur une analyse préalable rigoureuse que sur une sélection judicieuse des modèles, leur calibration et leur évaluation.