Salaheddine HOUKMI M2 Data Engineering
Clément ROUANET

Rapport Heart Disease

SOMMAIRE

Introduction

Voici les champs d'un dataset représentant des maladies cardiaques :

- **age :** age de la personne (en années)
- **sex :** sexe de la personne (0 = femme, 1 = homme)
- **cp :** type de douleur thoracique
  - ➢ Valeur 0 : asymptomatique
  - ➢ Valeur 1 : angine atypique
  - ➢ Valeur 2 : douleur non angineuse
  - ➢ Valeur 3 : angine typique
- **trestbps :** tension artérielle au repos de la personne (mmHg à l'admission à l'hôpital)
- **chol :** mesure du cholestérol de la personne en mg/dl
- **fbs :** glycémie à jeun de la personne (> 120 mg/dl, 1 = vrai ; 0 = faux)
- **restecg :** résultats électrocardiographiques au repos
  - ➢ Valeur 0 : montrant une hypertrophie ventriculaire gauche probable ou certaine selon les critères d'Estes
  - ➢ Valeur 1 : normale
  - ➢ Valeur 2 : présentant une anomalie de l'onde ST-T (inversions de l'onde T et/ou élévation ou dépression ST > 0,05 mV)
- **thalach :** fréquence cardiaque maximale atteinte par la personne
- **exang :** angine induite par l'exercice (1 = oui ; 0 = non)
- **oldpeak :** dépression ST induite par l'exercice par rapport au repos (« ST » se rapporte aux positions sur le tracé ECG)
- **slope:** la pente du segment ST de l'exercice de pointe
  - ➢ 0 : pente descendante, 1 : plat, 2 : ascendant
- **ca :** Le nombre de navires majeurs (0–3)
- **thal :** trouble sanguin appelé thalassémie
  - ➢ Valeur 0 : NULL
  - ➢ Valeur 1 : défaut corrigé (pas de flux sanguin dans une partie du cœur)
  - ➢ Valeur 2 : flux sanguin normal
  - ➢ Valeur 3 : défaut réversible (un flux sanguin est observé mais il n'est pas normal)
- **target :** Maladie cardiaque (0 = non, 1 = oui)

I - Exploitation des données

1 - Exploration des données

Nous avons commencé par importer les données du fichier CSV, nommé *heart.csv* grâce à la fonction *read_csv* de la librairie *pandas*.

Voici les premières lignes du dataframe obtenu :

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52.0	1	0	125.0	212	0	1	168	0	1.0	2	2	3	0
1	53.0	1	0	140.0	203	1	0	155	1	3.1	0	0	3	0
2	70.0	1	0	NaN	174	0	1	125	1	2.6	0	0	3	0
3	61.0	1	0	148.0	203	0	1	161	0	0.0	2	1	3	0
4	62.0	0	0	138.0	294	1	1	106	0	1.9	1	3	2	0

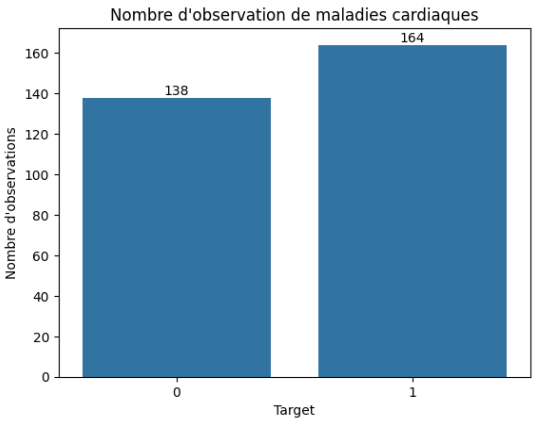
Le dataframe comprend 1025 lignes et 14 colonnes de type float et int. Il n'y a pas de colonnes catégorielles.

II - Nettoyage des données

1 - Valeurs dupliquées

Grâce à l'attribut *duplicated()* du dataframe, on obtient le nombre de lignes dupliquées dans le dataframe ; à savoir 723 lignes dupliquées parmi les 1025 que compose le dataframe. Ainsi, en prévention d'un éventuel overfitting et de l'amélioration de la qualité des données, on a décidé de supprimer les lignes dupliquées. Après cette opération, le dataframe ne comporte plus que 302 lignes.
Ces 302 lignes sont équilibrées pour chaque classe (0 et 1), cela nous permettra d'avoir une meilleure performance de modèle, éviter le biais du modèle, évaluation plus précise et limitation de la sur-représentation pour éviter un éventuel overfitting.

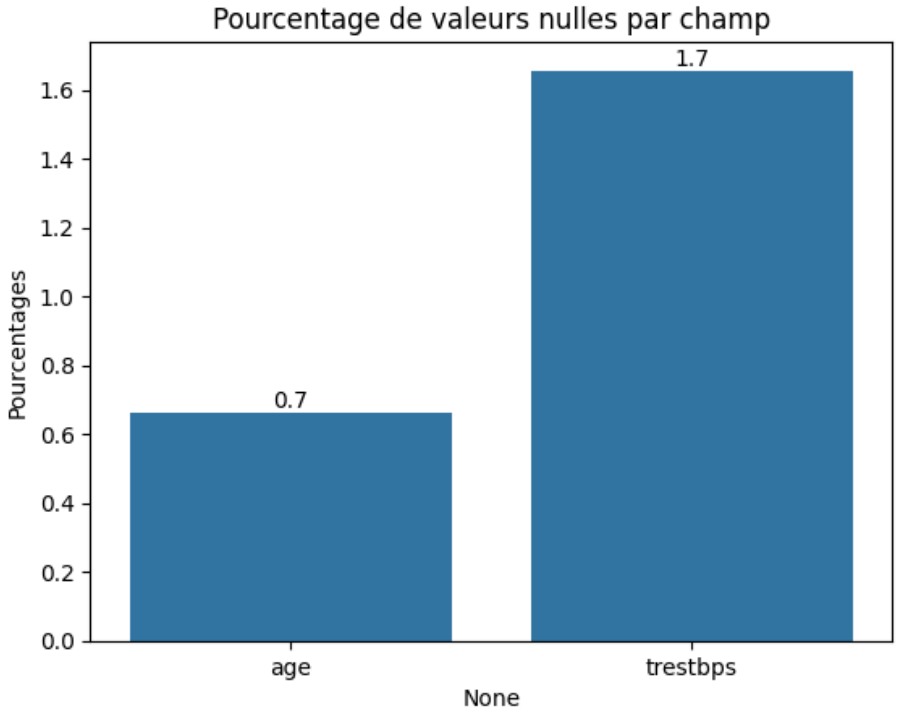
Voici un graphique qui résume le nombre de maladie ou non dans le dataframe :



2 - Valeurs nulles

Grâce à l'attribut *isnull()* du dataframe, on obtient le nombre de lignes avec des valeurs à nulles dans le dataframe. On trouve dans notre cas deux champs ayant des valeurs nulles, à savoir *âge avec 2 valeurs* et *trestbps* avec 5 valeurs.

Voici un graphique qui résume le pourcentage de valeurs nulles par champ :



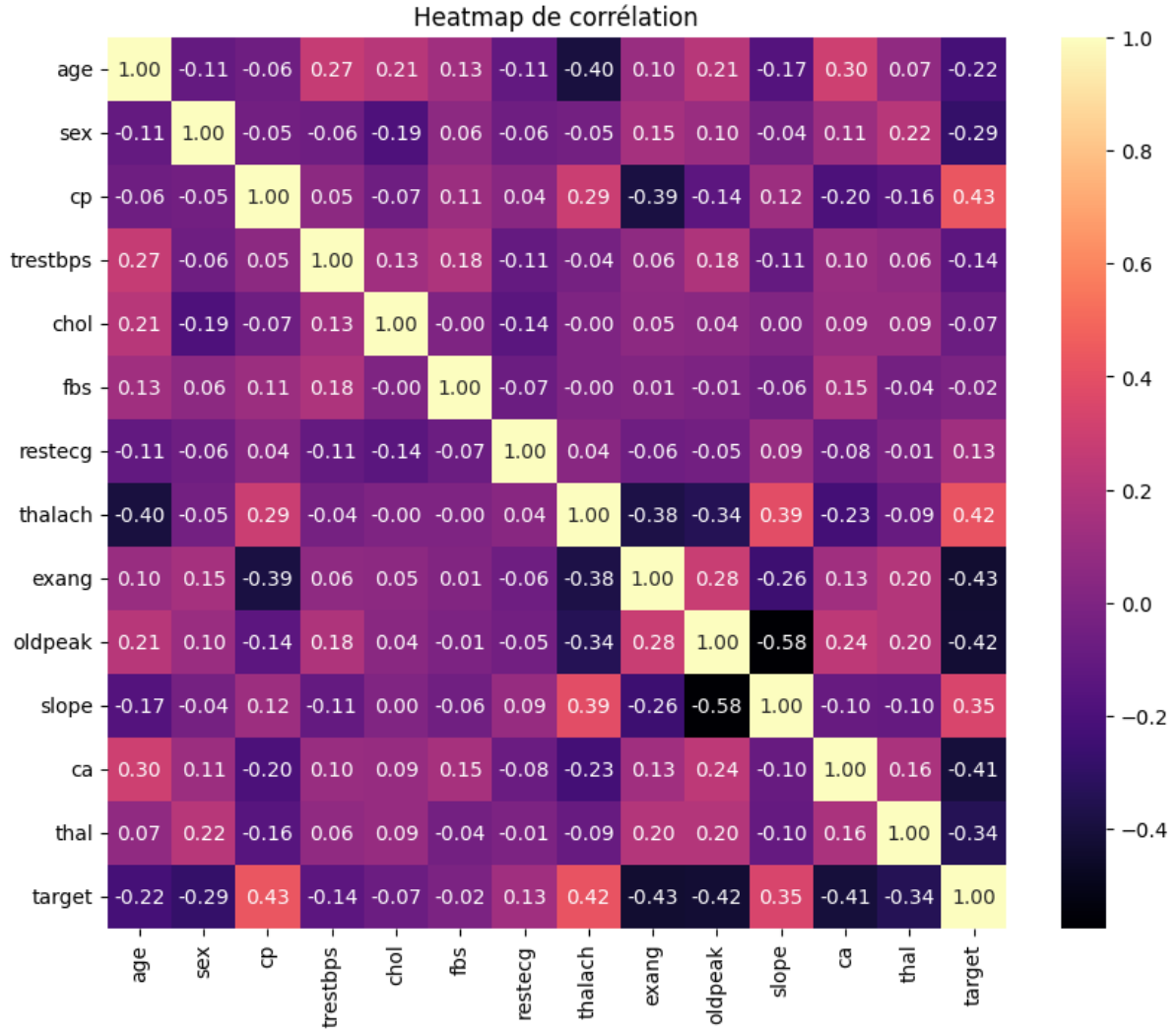
Nous avons remplacé les valeurs nulles du champ *'trestbps'* par la moyenne du champ qui est d'environ 130.97. Parce qu'on a remarqué que les lignes nulles ont des valeurs comprises entre 60 et 70 dans la colonne age et la plupart d'entre eux sont du sex masculin donc après avoir sollicité des études sur le métier on a constaté que la tension artérielle moyenne chez les hommes âgés de 60 à 70 ans peut être d'environ 130-135 mm Hg e qui présente la moyenne calculée.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
2	70.0	1	0	NaN	174	0	1	125	1	2.6	0	0	3	0
24	61.0	0	0	NaN	307	0	0	146	1	1.0	1	0	3	0
103	60.0	1	0	NaN	282	0	0	142	1	2.8	1	2	3	0
246	63.0	1	3	NaN	233	1	0	150	0	2.3	0	0	1	1
282	64.0	1	0	NaN	212	0	0	132	0	2.0	1	2	1	0

Puis, pour le champ *age*, nous avons décidé de supprimer les lignes associées au vu du peu de données manquantes (<10%) et de la difficulté de les remplacer. A la fin de cette étape, le dataframe ne comporte plus de valeurs nulles et est composé de 300 lignes.

3 - valeurs aberrantes

En premier lieu, traçons le heatmap de corrélation entre différentes variables de l'ensemble de données. Il permet d'identifier les relations linéaires entre les caractéristiques et de comprendre la structure sous-jacente des données.

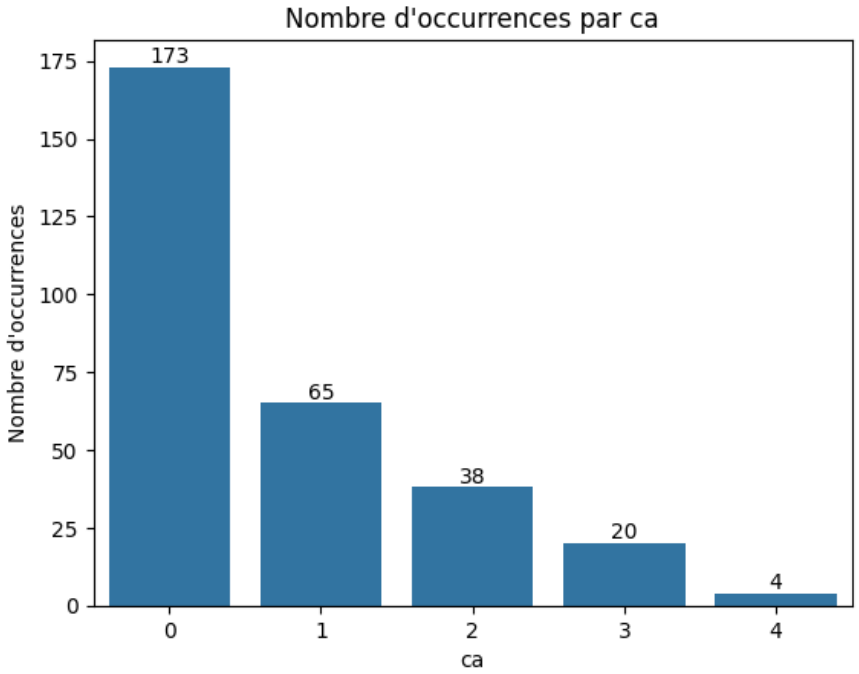


On remarque ainsi que les champs les plus corrélés au champ target sont :

- cp avec 0.43
- thalach avec 0.42
- slope avec 0.35

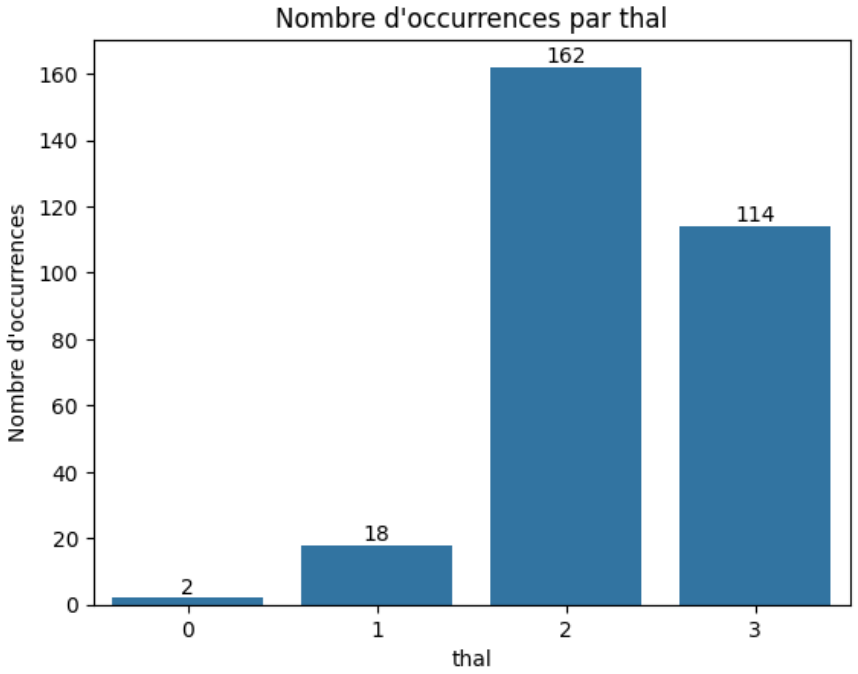
On a remarqué des valeurs aberrantes pour les champs 'ca' et 'thal' qui ne correspondent pas avec la documentation des données.

Voici, ci-dessous le nombre d'occurrences de chaque valeur pour le champ ‘ca’ :



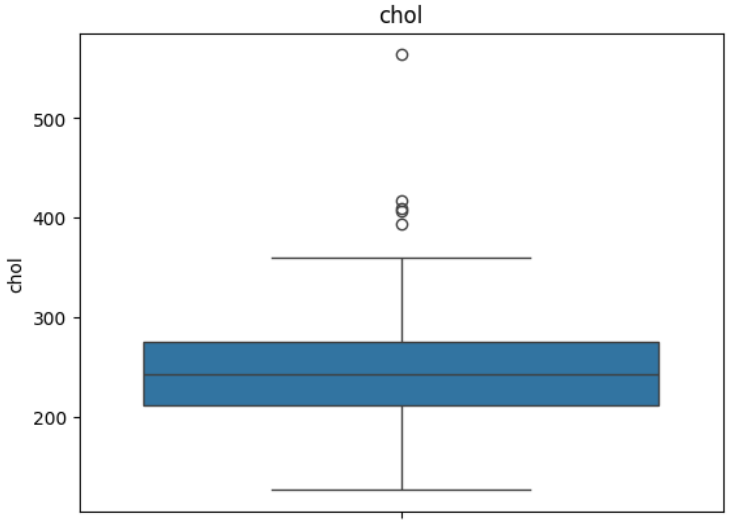
Le champ ‘ca’ ne prenant que des valeurs allant de 0 à 3, nous avons décidé de supprimer ces lignes du fait du petit nombre de valeurs et de corrélation faible.

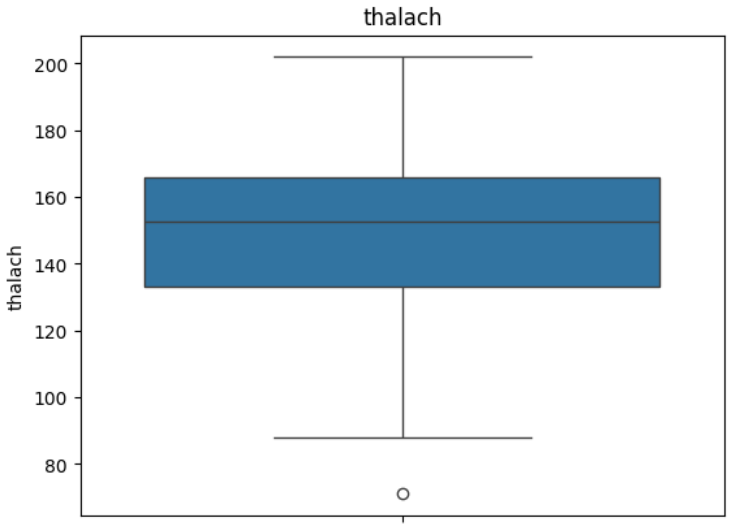
De même voici, ci-dessous le nombre d'occurrences de chaque valeur pour le champ thal’ :



Le champ ‘thal’ ne prenant que des valeurs allant de 1 à 3, les valeurs à 0 étant des valeurs nulles. Nous avons ainsi décidé de les supprimer.

Nous avons ensuite tracé les boîtes à moustaches de chaque champ afin de trouver de potentielles valeurs aberrantes. Nous avons ainsi pu trouver des valeurs aberrantes pour deux champs ‘chol’ et ‘thalach’ ; dont voici les boîtes à moustaches :





Afin de supprimer les valeurs aberrantes de ces deux champs, nous avons supprimé les lignes correspondant aux valeurs supérieures ou inférieures des limites supérieures et inférieures des boîtes à moustaches ci-dessus.

A la fin de cette étape, le dataframe ne comporte plus de valeurs aberrantes et est composé de 262 lignes, qui seront ensuite celles sur qui l’apprentissage sera effectué.

III - Traitements des modèles

1 - Preprocessing

La première étape consiste à séparer le dataframe préparé précédemment en deux : un appelé *y* contenant que la colonne des résultats ‘target’ et un autre *X* contenant toutes les autres colonnes.

Nous avons ensuite utilisé la fonction *train_test_split* de la librairie *sklearn*, pour séparer les deux dataframes *X* et *y* en deux avec une partie pour entraîner le modèle et une seconde pour tester le modèle.
Finalement, nous avons utilisé la fonction *StandardScaler* qui est une méthode de prétraitement pour mettre à l'échelle les caractéristiques de l’ensemble de données en centrant les données et réduisant l’échelle. Ainsi, tous les champs et valeurs auront le même poids pour les futures modèles de prédiction, améliorant ainsi leurs précisions.

2 - SVM (Support Vector Machine)

Nous avons utilisé le modèle SVC (Support Vector Classification), qui est une implémentation spécifique de SVM pour la classification dans la bibliothèque scikit-learn en Python.

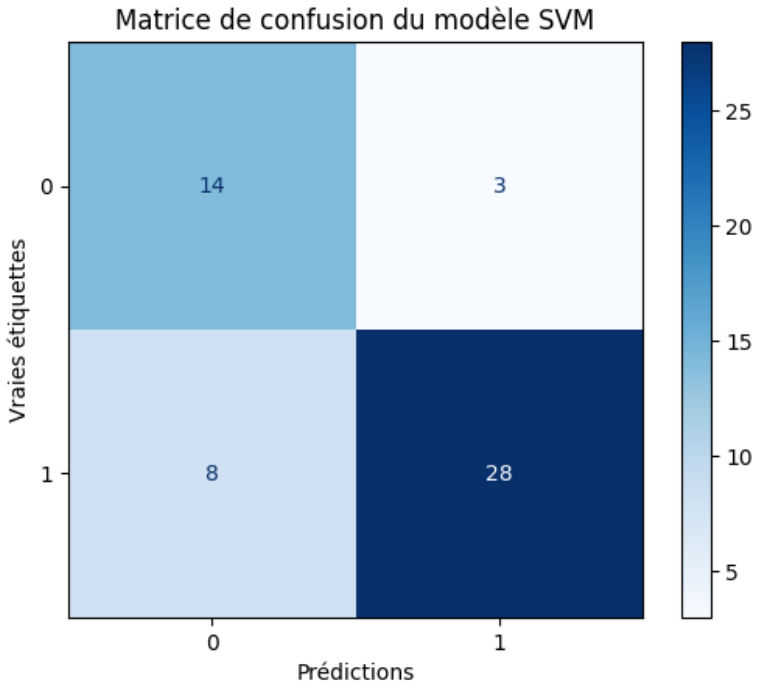
Puis, nous avons cherché les meilleurs paramètres pour le modèle afin d'avoir une précision plus importante. Pour cela, nous avons utilisé la fonction *GridSearchCV* de scikit-learn pour effectuer une recherche systématique des hyperparamètres optimaux pour le modèle d'apprentissage. Elle effectue une recherche exhaustive à travers une grille spécifiée d'hyperparamètres, évaluant chaque combinaison par validation croisée et sélectionnant celle qui produit les meilleures performances.

Nous avons trouvé les hyperparamètres optimaux suivants :

- *C* : qui contrôle la régularisation dans les modèles SVC ⇒ 0.5 ;
- *gamma* : qui influence un exemple unique sur l'ensemble de la frontière de décision ⇒ auto

On entraîne ensuite le modèle SVC avec le dataframe X d'entraînement mis à l'échelle et le y de test. Puis, on effectue une prédiction du résultat avec le dataframe X de test mis à l'échelle avec le modèle entraîné.

On peut à présent afficher la matrice de confusion du *y* prédit et du y de test afin d'évaluer les performances d'un modèle de classification ; que voici :



Finalement, voici le rapport avec les différentes valeurs de test de précision de notre modèle SVM :

	precision	recall	f1-score	support
0	0.64	0.82	0.72	17
1	0.90	0.78	0.84	36
accuracy			0.79	53
macro avg	0.77	0.80	0.78	53
weighted avg	0.82	0.79	0.80	53

3 - KNN (Nearest Neighbour classification)

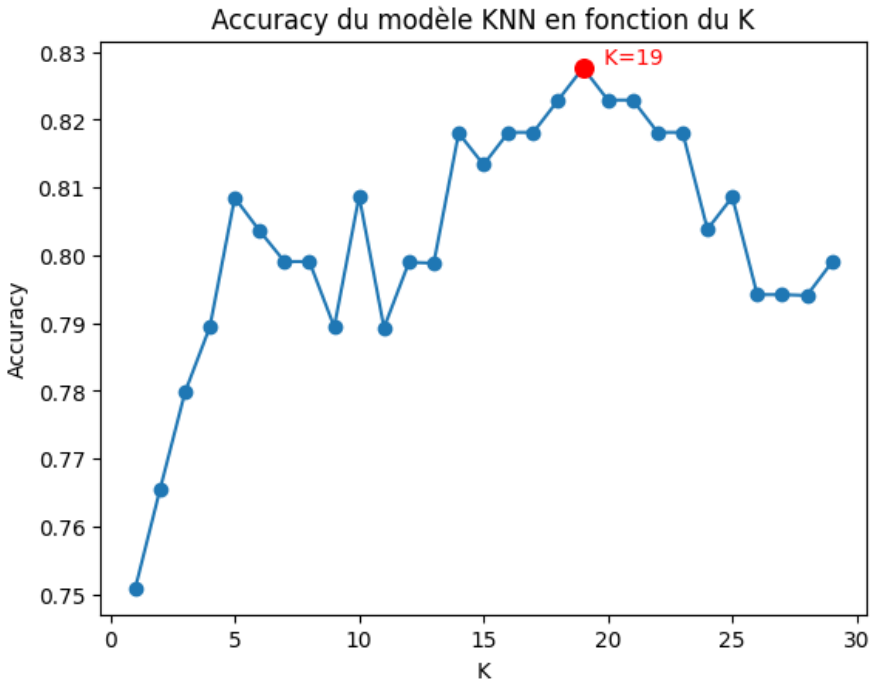
Nous avons utilisé le modèle KNN (Nearest Neighbour classification), qui est un algorithme d'apprentissage automatique supervisé utilisé à la fois pour la classification et la régression.

Puis, nous avons, comme précédemment, cherché les meilleurs hyperparamètres avec la fonction *GridSearchCV* de scikit-learn.

Nous avons trouvé l'hyperparamètre optimal suivant :

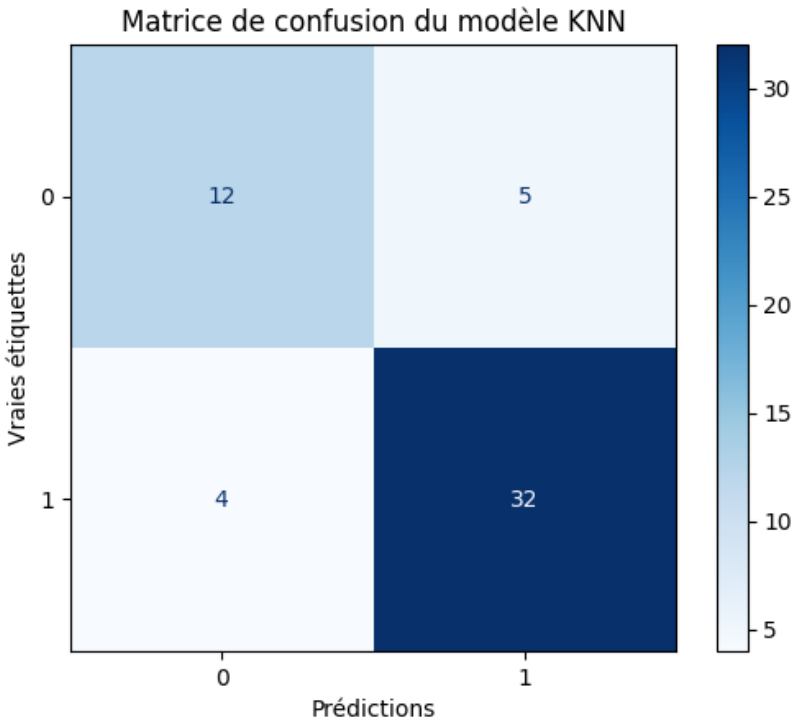
- *n_neighbors* : qui représente le nombre de voisins à considérer lors de la prédiction d'une nouvelle observation ⇒ 19

On peut voir sur le schéma Elbow suivant le K choisi est celui avec l'accuracy la plus élevée:



On entraîne ensuite le modèle KNN avec le dataframe X d'entraînement mis à l'échelle et le y de test. Puis, on effectue une prédiction du résultat avec le dataframe X de test mis à l'échelle avec le modèle entraîné.

On peut à présent afficher la matrice de confusion du *y* prédit et du y de test afin d'évaluer les performances d'un modèle de classification ; que voici :



Finalement, voici le rapport avec les différentes valeurs de test de précision de notre modèle KNN :

	precision	recall	f1-score	support
0	0.75	0.71	0.73	17
1	0.86	0.89	0.88	36
accuracy			0.83	53
macro avg	0.81	0.80	0.80	53
weighted avg	0.83	0.83	0.83	53

4 - Random Forest

Nous avons utilisé le modèle de Random Forest, qui est est un algorithme d'apprentissage automatique appartenant à la catégorie des modèles d'ensemble.

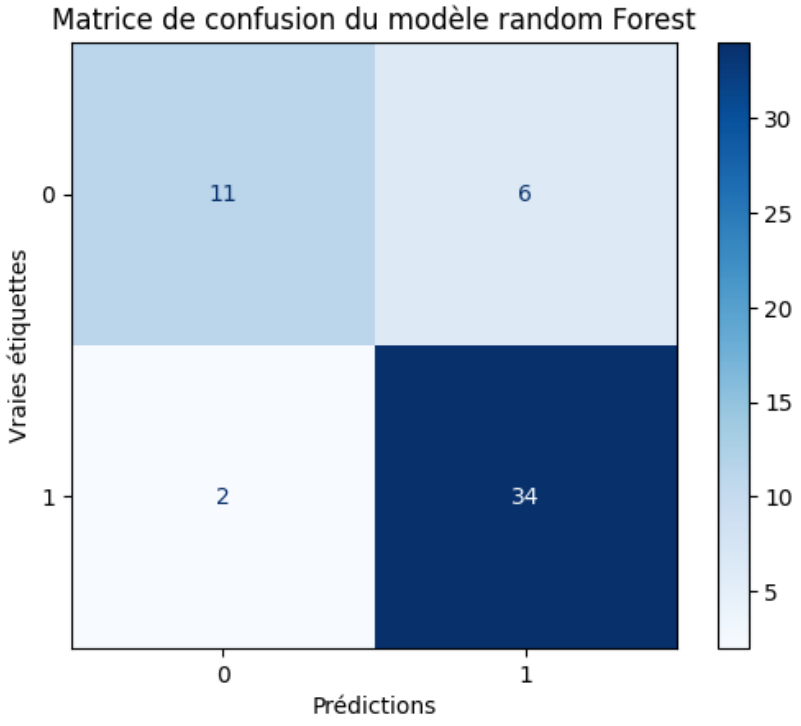
Puis, nous avons, comme précédemment, cherché les meilleurs hyperparamètres avec la fonction *RandomizedSearchCV* de scikit-learn.

Nous avons trouvé l'hyperparamètre optimal suivant :

- *max_depth* : qui contrôle la profondeur maximale de chaque arbre de décision dans la forêt ⇒ 2
- *n_estimators* : qui spécifie le nombre d'arbres dans la forêt ⇒ 659

On entraîne ensuite le modèle de Random Forest avec le dataframe X d'entraînement mis à l'échelle et le y de test. Puis, on effectue une prédiction du résultat avec le dataframe X de test mis à l'échelle avec le modèle entraîné.

On peut à présent afficher la matrice de confusion du *y* prédit et du y de test afin d'évaluer les performances d'un modèle de classification ; que voici :



Finalement, voici le rapport avec les différentes valeurs de test de précision de notre modèle Random Forest :

	precision	recall	f1-score	support
0	0.85	0.65	0.73	17
1	0.85	0.94	0.89	36
accuracy			0.85	53
macro avg	0.85	0.80	0.81	53
weighted avg	0.85	0.85	0.84	53

IV - Comparaisons des modèles & Conclusion

Afin de comparer les trois modèles et voir lequel est le plus efficace pour notre dataset, nous avons calculé deux scores ; qui sont l'accuracy (évaluation du modèle) et l'accuracy moyenne de la validation croisée.

Les valeurs de chaque modèle sont obtenues grâce à la fonction :

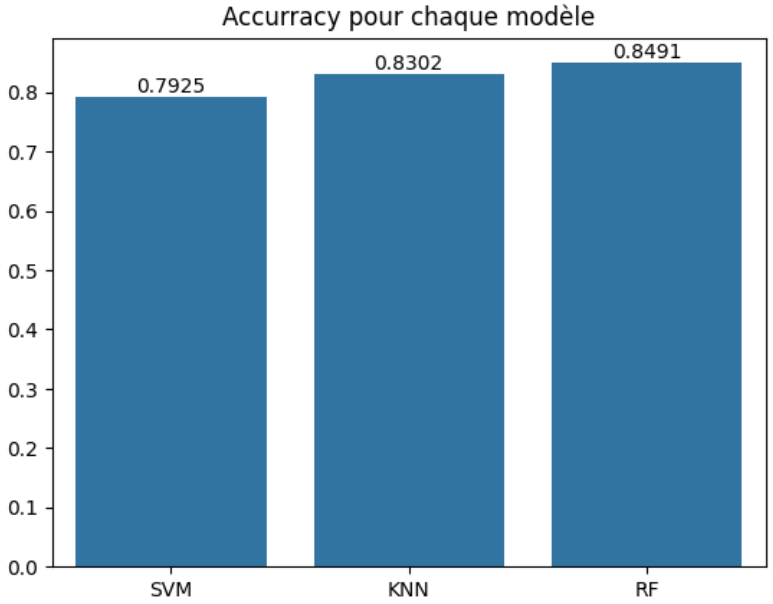
- score qui est un attribut du modèle (accuracy) ;
- *cross_val_score* fonction de sklearn (accuracy de la validation croisée).

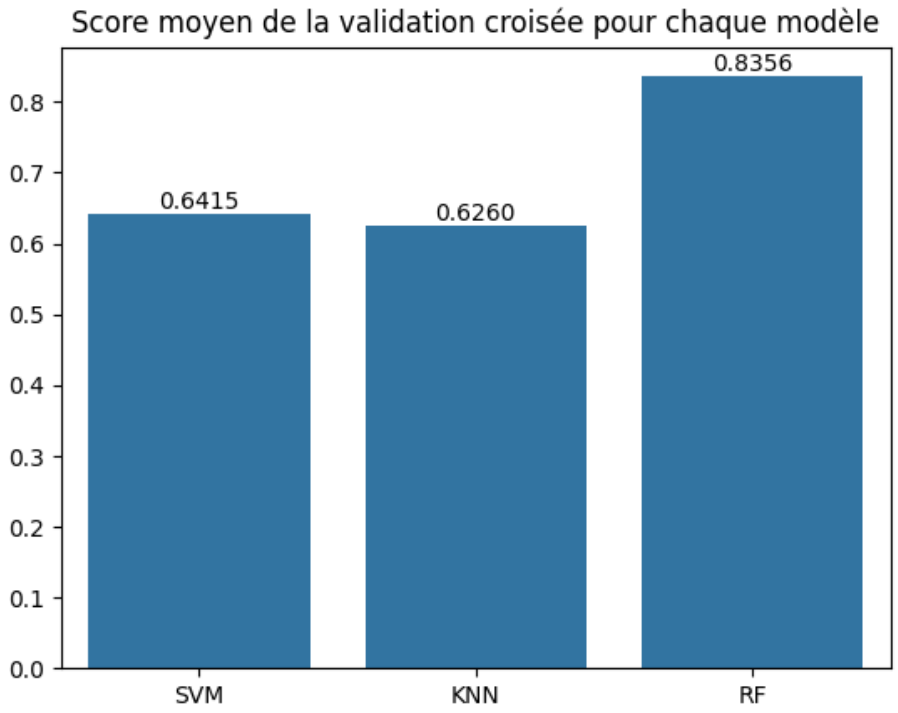
Voici donc les valeurs obtenues :

```
Accuracy for SVM: 0.7925
Accuracy for KNN: 0.8302
Accuracy for RF: 0.8491
```

```
Mean accuracy for SVM: 0.6415
Mean accuracy for KNN: 0.6260
Mean accuracy for RF: 0.8434
```

Pour une meilleure comparaison, voici les schémas propres à chaque mesure :





On peut ainsi aisément conclure que le modèle de Random Forest est le plus efficace et précis pour traiter notre dataset.

Voici pourquoi Random Forest est le modèle le plus adapté à notre problème :

- **Gestion des relations non linéaires :** Si les relations entre les caractéristiques et la variable cible (la colonne 'target') sont non linéaires ou complexes, Random Forest peut être efficace pour capturer ces relations grâce à son approche basée sur des arbres de décision.

- **Robustesse aux valeurs aberrantes :** Les données médicales peuvent parfois contenir des valeurs aberrantes. Random Forest peut être plus robuste à ces valeurs aberrantes car il agrège plusieurs arbres, atténuant ainsi les effets de surajustement causés par des observations atypiques.

- **Prise en compte automatique des interactions entre les caractéristiques :** Random Forest a la capacité intrinsèque de prendre en compte les interactions entre les caractéristiques, ce qui peut être important dans le domaine médical où plusieurs facteurs peuvent contribuer à un résultat.

- **Peu de prétraitement des données nécessaire :** Random Forest est généralement moins sensible aux problèmes de mise à l'échelle des caractéristiques ou de normalisation, ce qui peut être avantageux si les caractéristiques de votre ensemble de données ont des échelles différentes.