

Projet de Machine Learning

Application des méthodes de Machine Learning pour l'évaluation de la potabilité de l'eau



Réalisé par : BERRAMOU Imane
LYOUSFI Imane
SAKKALI Youssra
SAOUD Oumaima

Réalisé par : Mr. TAMIM Ayoub



Projet de Machine Learning

Application des méthodes de Machine Learning pour l'évaluation de la potabilité de l'eau

Sommaire

I.	Introduction sur Le Machine Learning	3
II.	Description de la base de données	5
1.	Description des variables et des cibles	5
1.1	Contexte	5
1.2	Contenu	5
2.	Description des variables et des cibles a l'aide de spider	8
2.1	Importation des librairies	8
2.2	Importation de la base de données	8
2.3	Visualisation des données	8
2.4	La taille de Matrice	9
2.5	Les titres de colonnes (Target and Features)	9
2.6	Visualisation des informations sur les données	9
2.7	La matrice de corrélation	10
2.8	Probabilité de l'eau potable ou non potable	11
2.9	Distribution des données	12
2.10	Proportionnalité des données	13
2.11	Valeurs uniques	15
3.	Valeurs manquantes	15
3.1	Visualiser des valeurs manquantes	15
3.2	Remplissage des valeurs manquantes	17
III.	Application des méthodes :	18
1.	Séparation des données et importation des librairies	18
1.1	Séparation des données	18
1.2	Importation des standardscaler vers l'échelle de performance	18
1.3	Entraînement du model	18
2.	La régression logistique	19



2.1	Principe	19
2.2	Application de la régression logistique selon python	20
3.	Méthode K-nearest neighbors (kNN)	23
3.1	Principe	23
3.2	Application de la méthode K-nearest neighbors (kNN)selon python	24
4.	Méthode Support Vector Machine (SVM).....	27
4.1	Principe	27
4.2	Application de la méthode Support Vector Machines (SVMs) selon python	29
IV.	Procédure commune de mesure de la performance d'un algorithme : 32	
1.	Méthodes de mesure de performance	32
1.1	La matrice de confusion	33
1.2	Principe Cross validation	34
1.3	Accuracy	35
1.4	Représentation graphique de données.....	35
2.	Comparaison des accuacy-score (obtenus à partir de la matrice de confusion) résultants des trois algorithmes de classification utilisés à l'aide de Python	36
3.	Comparaison des accuacy-score obtenus à partir de la cross-validation appliquée aux trois algorithmes de classification utilisés :.....	39
V.	Les annexes.....	40



I. Introduction sur Le Machine Learning

Le machine Learning est une technique de programmation informatique qui utilise des probabilités statistiques pour donner aux ordinateurs la capacité d'apprendre par eux-mêmes sans programmation explicite.

Pour son objectif de base, le machine Learning « apprend à apprendre » aux ordinateurs, et par la suite, à agir et réagir comme le font les humains, en améliorant leur mode d'apprentissage et leurs connaissances de façon autonome sur la durée. L'objectif ultime serait que les ordinateurs agissent et réagissent sans être explicitement programmés pour ces actions et réactions. Le machine Learning utilise des programmes de développement qui s'ajustent chaque fois qu'ils sont exposés à différents types de données en entrée. La clé du machine Learning réside dans l'entrée de volumes considérables de données dans l'ordinateur. Pour apprendre, la machine a besoin de consommer des big data.

Le machine Learning peut être classé en trois grandes catégories (Machine Learning avec supervision, sans supervision et renforcement).

Dans notre cas, il s'agit de problème de machine Learning avec supervision (Apprendre par l'exemple) avec un modèle **de classification binaire (juger la potabilité ou la non potabilité de l'eau)**. Cet apprentissage supervisé consiste en des variables d'entrée (X) et une variable de sortie (Y). Nous utilisons un algorithme pour apprendre la fonction de mapping de l'entrée à la sortie ($Y = f(X)$). Le but est d'appréhender si bien la fonction de mapping que, lorsque nous avons de nouvelles données d'entrée (X), nous pouvons prédire les variables de sortie (Y) pour ces données.

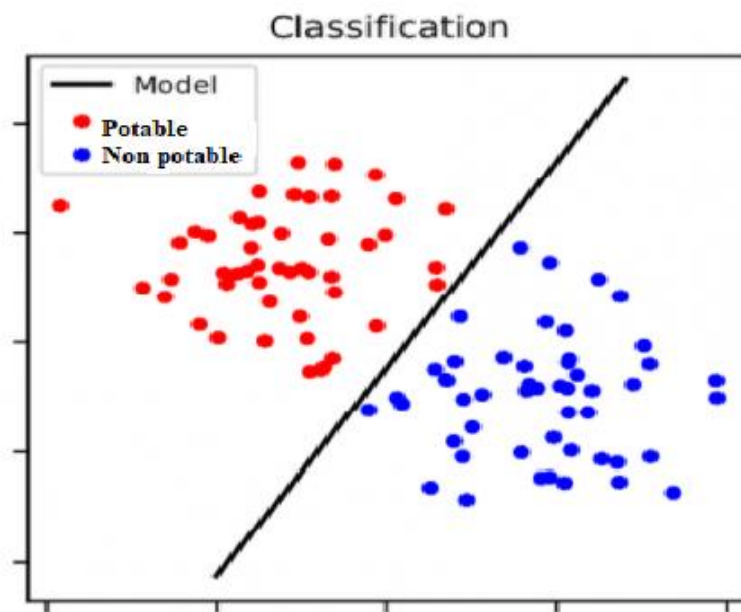
C'est ce qu'on appelle l'apprentissage supervisé, car le processus d'un algorithme tiré de l'ensemble de données de formation (training set) peut être considéré comme un enseignant supervisant le processus d'apprentissage. Nous connaissons les réponses correctes, l'algorithme effectue des prédictions itératives sur les données d'apprentissage et est corrigé par l'enseignant. L'apprentissage s'arrête lorsque l'algorithme atteint un niveau de performance acceptable.

Pour maîtriser l'apprentissage supervisé, il faut absolument comprendre et connaître 4 notions suivantes :

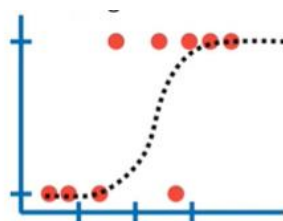


- **Dataset** (Regroupe un ensemble de données. Celles-ci dépendent d'une variable associée aux valeurs. Leur accès peut se produire de manière individuelle ou collective).
- **Modèle et ses paramètres** (Il peut s'agir d'un modèle linéaire ou non linéaire).
- **Fonctions coût** (Mesure de la performance).
- **Algorithme d'apprentissage** (Trouver les paramètres qui minimisent la fonction Coût).

Dans notre cas, le problème de classification binaire survient lorsque la variable de sortie (target =potabilité) est une catégorie, telle que « eau potable (1) », « eau non potable (0) ».



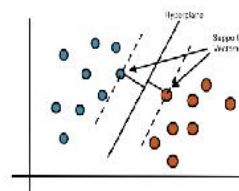
Dans notre cas, les méthodes d'apprentissage supervisé utilisées dans les problèmes de classification (**SVM, régression logistique et K-NN**).



Regression logistique



K-NN



SVM



II. Description de la base de données

1. Description des variables et des cibles

1.1 Contexte

L'accès à l'eau potable est essentiel à la santé, un droit humain fondamental et une composante d'une politique efficace de protection de la santé. Il s'agit d'une question de santé et de développement importante au niveau national, régional et local. Dans certaines régions, il a été démontré que les investissements dans l'approvisionnement en eau et l'assainissement peuvent apporter un bénéfice économique net, puisque la réduction des effets néfastes sur la santé et des coûts des soins de santé est supérieure aux coûts des interventions.

Il est important de noter qu'une **eau considérée comme potable n'est pas forcément totalement exempte de matières polluantes**. C'est juste que sa concentration en matières polluantes est considérée comme relativement faible pour ne pas nuire à la santé des consommateurs.

Le machine Learning intervient avec ses différentes méthodes pour trouver une solution et développer des applications et apprendre à la machine comment juger la potabilité ($y = 1$) ou la non potabilité d'une masse d'eau ($y = 0$), en créant, entraînant, utilisant et évaluant le modèle sur différents échantillons de masses d'eau caractérisés par différents variables (9 features : La valeur du pH, La dureté en mg/L, Solides (Total des solides dissous - TDS) en ppm, Chloramines en ppm, Sulfate en mg/L, Conductivité en $\mu\text{S}/\text{cm}$, Carbone organique ppm, Trihalométhanes $\mu\text{g}/\text{L}$, Turbidité en NTU (Nephelometric Turbidity Units)).

1.2 Contenu

Le fichier water_potability.csv contient des mesures de la qualité de l'eau pour **3276** masses d'eau différentes.

Les critères qui définissent une eau potable sont divers et vont bien au-delà de la mesure du taux de pH ! On retrouve en effet plus de 60 critères que l'on peut organiser en 7 groupes : Organoleptiques, Physico-chimiques, Substances indésirables, Substances toxiques, Microbiologiques, Pesticides et produits apparentés et Eaux adoucies.



Dans cette base, les mesures ont concerné le pH, la dureté, le total des solides dissous, les chloramines, le sulfate, la conductivité, le carbone organique, les Trihalométhanes et la turbidité.

a. La valeur du pH

Le PH est un paramètre important pour évaluer l'équilibre acide-base de l'eau. C'est également l'indicateur de l'état acide ou alcalin de l'eau. L'OMS a recommandé une limite maximale admissible de pH de 6,5 à 8,5. L'enquête actuelle a révélé des valeurs de 6,52 à 6,83, ce qui est conforme aux normes de l'OMS.

b. La dureté en mg/L

La dureté est principalement causée par les sels de calcium et de magnésium. Ces sels sont dissous à partir de dépôts géologiques que l'eau traverse. La durée pendant laquelle l'eau est en contact avec des matériaux produisant de la dureté aide à déterminer la quantité de dureté présente dans l'eau brute. La dureté a été définie à l'origine comme la capacité de l'eau à précipiter le savon causé par le calcium et le magnésium

c. Solides (Total des solides dissous - TDS) en ppm

L'eau a la capacité de dissoudre une large gamme de minéraux ou de sels inorganiques et certains organiques tels que le potassium, le calcium, le sodium, les bicarbonates, les chlorures, le magnésium, les sulfates, etc. Ces minéraux produisent un goût indésirable et une couleur diluée dans l'apparence de l'eau. Il s'agit d'un paramètre important pour l'utilisation de l'eau. L'eau avec une valeur TDS élevée indique que l'eau est hautement minéralisée. La limite souhaitable pour le TDS est de **500 mg/l** et la limite maximale est de **1000 mg/l**, ce qui est prescrit pour la consommation.

d. Chloramines en ppm

Le chlore et les chloramines sont les principaux désinfectants utilisés dans les systèmes d'eau publics. Les chloramines se forment le plus souvent lorsque de l'ammoniac est ajouté au chlore pour traiter l'eau potable. Des niveaux de chlore allant jusqu'à **4 mg/L** ou **4 parties par million (ppm)** sont considérés comme sûrs dans l'eau potable.



e. Sulfate en mg/L

Les sulfates sont des substances naturelles que l'on trouve dans les minéraux, le sol et les roches. Ils sont présents dans l'air ambiant, les eaux souterraines, les plantes et les aliments. La principale utilisation commerciale du sulfate est dans l'industrie chimique. La concentration de sulfate dans l'eau de mer est d'environ **2 700 milligrammes par litre (mg/L)**. Elle varie de **3 à 30 mg/L** dans la plupart des réserves d'eau douce, bien que l'on trouve des concentrations beaucoup plus élevées (1000 mg/L) dans certains endroits géographiques.

f. Conductivité en $\mu\text{S}/\text{cm}$

L'eau pure n'est pas un bon conducteur de courant électrique, c'est plutôt un bon isolant. L'augmentation de la concentration en ions améliore la conductivité électrique de l'eau. En général, la quantité de solides dissous dans l'eau détermine la conductivité électrique. La conductivité électrique (CE) mesure en fait le processus ionique d'une solution qui lui permet de transmettre le courant. Selon les normes de l'OMS, la valeur de la CE ne doit pas dépasser **400 $\mu\text{S}/\text{cm}$** .

g. Carbone organique ppm

Le carbone organique total (COT) dans les eaux de source provient de la matière organique naturelle en décomposition (NOM) ainsi que de sources synthétiques. Le COT est une mesure de la quantité totale de carbone dans les composés organiques dans l'eau pure. Selon l'US **EPA** **< 2 mg/L** de COT dans l'eau traitée / potable, et **< 4 mg/l** dans l'eau de source qui est utilisée pour le traitement.

h. Trihalométhanes $\mu\text{g}/\text{L}$

Les THM sont des produits chimiques que l'on peut trouver dans l'eau traitée au chlore. La concentration de THM dans l'eau potable varie en fonction du niveau de matière organique dans l'eau, de la quantité de chlore nécessaire pour traiter l'eau et de la température de l'eau qui est traitée. Une concentration de THM allant jusqu'à **80 ppm** est considérée comme sûre dans l'eau potable.

i. Turbidité en NTU (Nephelometric Turbidity Units)

La turbidité de l'eau dépend de la quantité de matières solides présentes à l'état de suspension. C'est une mesure des propriétés d'émission de lumière de l'eau et le test est



utilisé pour indiquer la qualité du rejet des déchets en ce qui concerne les matières colloïdales.

j. Potabilité

Elle indique si l'eau est propre à la consommation humaine, où 1 signifie Potable et 0 signifie Non potable.

2. Description des variables et des cibles à l'aide de spider

2.1 Importation des librairies

Avant de commencer tout code il faut d'abord importer les librairies dont on a besoin :

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

2.2 Importation de la base de données

Pour importer la base de données il suffit de spécifier le chemin du fichier water_potability.csv. Dans cet exemple le chemin suivi est :

```
#importation de la base de donnee
data = pd.read_csv("C:\\Users\\hp\\Desktop\\water_potability.csv")
```

2.3 Visualisation des données

La fonction **Head ()** sert à afficher les données, par défaut elle affiche les 5 premiers éléments (échantillon), sinon c'est possible de spécifier le nombre des éléments à afficher : **data.head (10)** pour afficher 10 par exemple.

```
   ph  Hardness  Solids  ...  Trihalomethanes  Turbidity  Potability
0  7.080795  204.890455  20791.318981  ...      86.990970    2.963135         0
1  3.716080  129.422921  18630.057858  ...      56.329076    4.500656         0
2  8.099124  224.236259  19909.541732  ...      66.420093    3.055934         0
3  8.316766  214.373394  22018.417441  ...     100.341674    4.628771         0
4  9.092223  181.101509  17978.986339  ...      31.997993    4.075075         0

[5 rows x 10 columns]
```



2.4 La taille de Matrice

La fonction **shape ()** sert à afficher la taille de la matrice (nombre de Lignes, nombre de Colonnes).

```
(3276, 10)
```

2.5 Les titres de colonnes (Target and Features)

La fonction **print (data.columns)** permet d'afficher les titres des colonnes.

```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',  
      'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],  
      dtype='object')
```

2.6 Visualisation des informations sur les données

a. La fonction describe()

Donne des informations générales sur les données (le nombre des éléments, la moyenne, écart type, le minimum, les quantités et le maximum)

	ph	Hardness	...	Turbidity	Potability
count	3276.000000	3276.000000	...	3276.000000	3276.000000
mean	7.080795	196.369496	...	3.966786	0.390110
std	1.469956	32.879761	...	0.780382	0.487849
min	0.000000	47.432000	...	1.450000	0.000000
25%	6.277673	176.850538	...	3.439711	0.000000
50%	7.080795	196.967627	...	3.955028	0.000000
75%	7.870050	216.667456	...	4.500320	1.000000
max	14.000000	323.124000	...	6.739000	1.000000

```
[8 rows x 10 columns]
```



b. La Fonction info ()

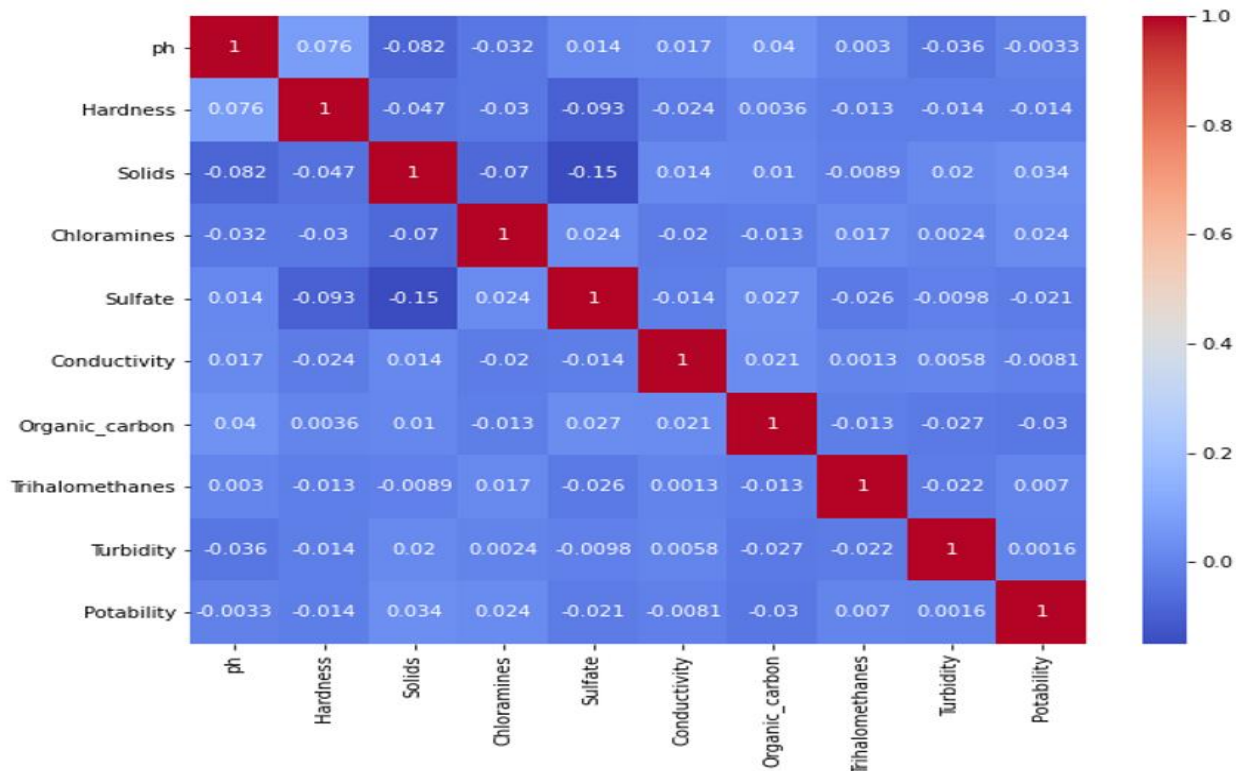
Méthode imprime des informations sur le DataFrame. Les informations contiennent le nombre de colonnes, les étiquettes de colonne, les types de données de colonne, l'utilisation de la mémoire, l'index de plage et le nombre de cellules dans chaque colonne (valeurs non nulles)

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   ph                     3276 non-null   float64
1   Hardness               3276 non-null   float64
2   Solids                 3276 non-null   float64
3   Chloramines            3276 non-null   float64
4   Sulfate                3276 non-null   float64
5   Conductivity           3276 non-null   float64
6   Organic_carbon         3276 non-null   float64
7   Trihalomethanes       3276 non-null   float64
8   Turbidity              3276 non-null   float64
9   Potability             3276 non-null   int64   
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

2.7 La matrice de corrélation

```
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')
```

Dans le code ci-dessus, nous allons représenter une matrice de corrélation à l'aide d'une carte thermique en Python



La carte thermique est utilisée pour représenter graphiquement les valeurs de la matrice avec différentes nuances de couleurs pour différentes valeurs. Il visualise très clairement la matrice globale.

On s'intéresse plus à la colonne et la ligne de 'potability' (Target).

→ Toutes les valeurs sont de même importance par égalité pour entrainer notre modèle.

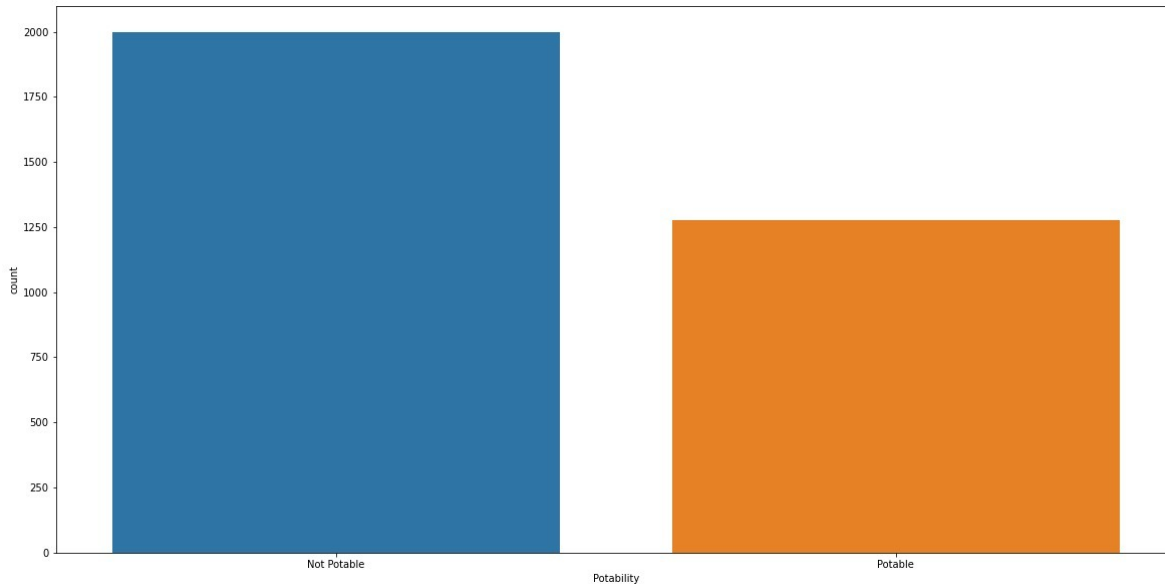
2.8 Probabilité de l'eau potable ou non potable

Le code ci-dessus nous permet de visualiser sur un graphe la probabilité de l'eau potable et de l'eau non potable.

```
ax = sns.countplot(x = "Potability", data= data, saturation=0.8)
plt.xticks(ticks=[0, 1], labels = ["Not Potable", "Potable"])
plt.show()
```



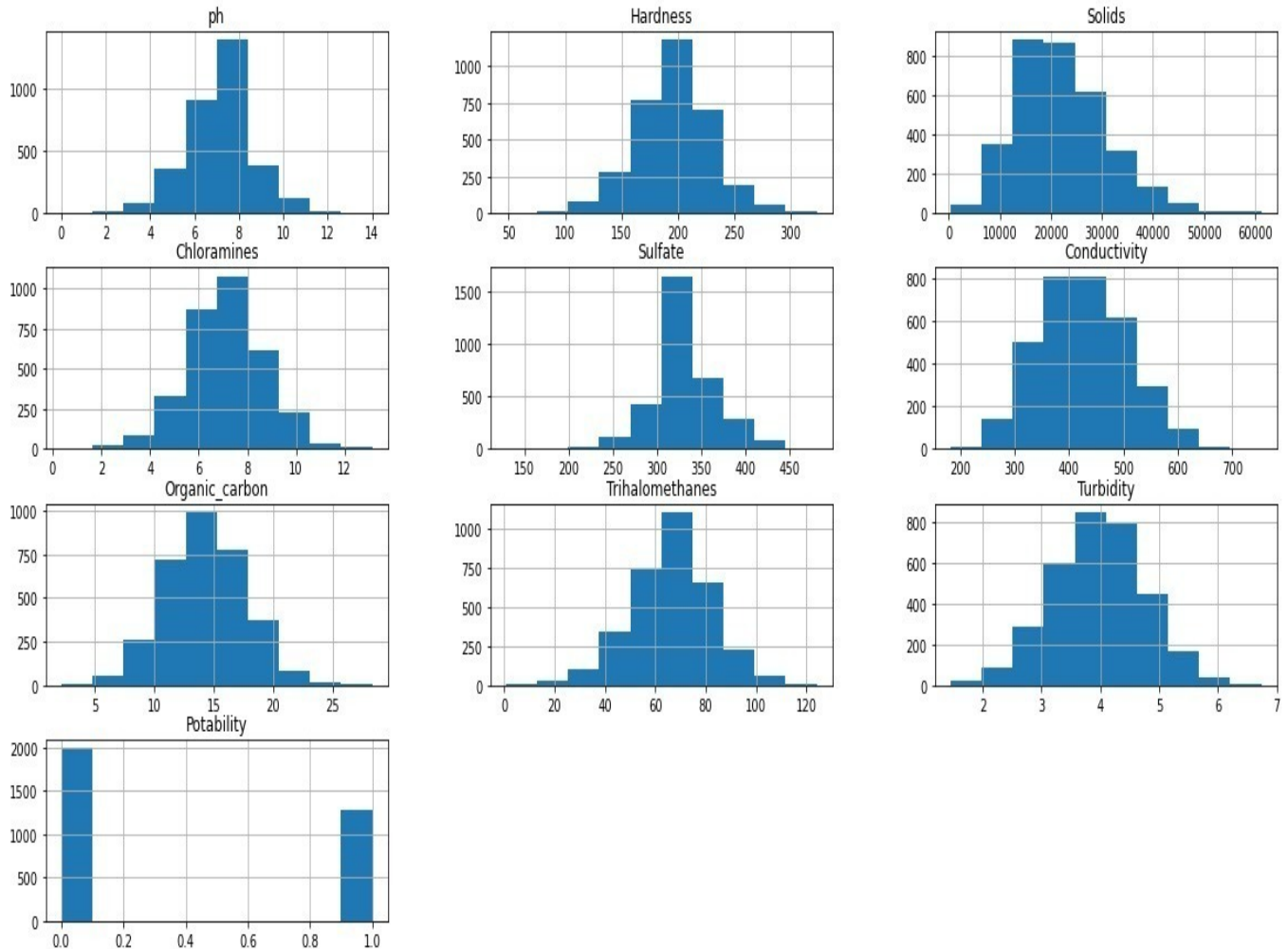
Selon le graphe si dessous on peut déduire que la probabilité d'un faux positif en jugeant une eau potable, non potable est très grande alors que la probabilité d'un faux positif en jugeant une eau non potable, potable est relativement faible.



2.9 Distribution des données

```
plt.rcParams['figure.figsize'] = [20,10]  
data.hist()  
plt.show()
```

Le code ci-dessus nous a permis de mieux explorer et visualiser la distribution de nos données. Ce sont des histogrammes qui illustrent la distribution des valeurs de chaque feature.



2.10 Proportionnalité des données

Violin Plot est une méthode pour visualiser la distribution des données numériques de différentes variables.

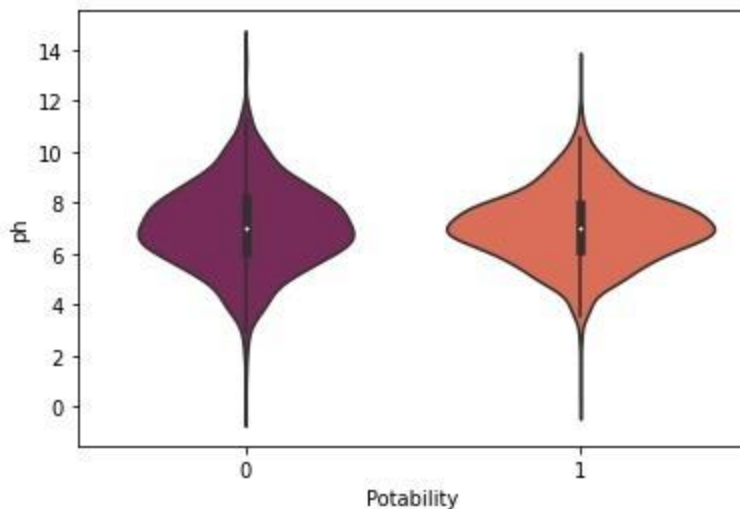
Ce plot n'est pas applicable pour toutes les valeurs, au contraire de l'histogramme.

On visualise dans le **Violinplot** la proportionnalité des 'Features' par rapport à 'Target'.

Exemple 1 : pH

```
sns.violinplot(x='Potability', y='ph', data=data, palette='rocket')
```

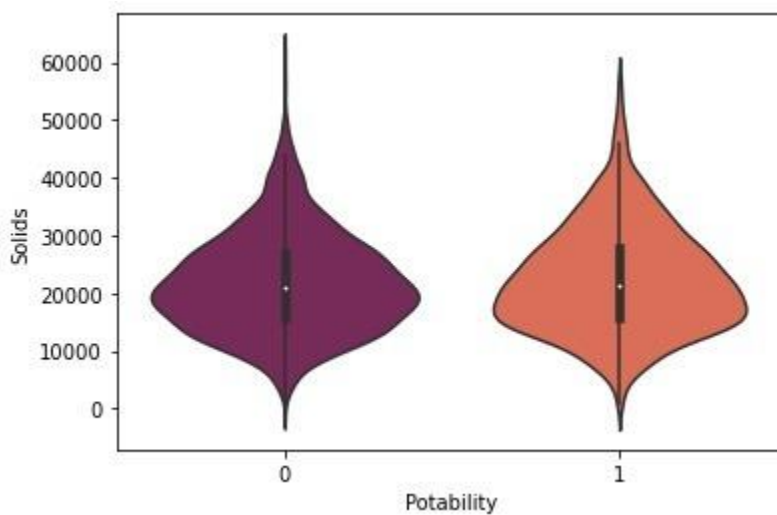
Selon le graphe ci-dessous, La proportionnalité la plus grande → pH=7 de l'eau potable (1) et non potable (0) est :



Exemple 2 : Solids

```
sns.violinplot(x='Potability', y='Solids', data=data, palette='rocket')
```

Selon le graphe ci-dessous, La proportionnalité la plus grande → Solids = 20 000 de l'eau potable (1) et non potable(0).





2.11 Valeurs uniques

```
print(data.nunique())
```

La fonction **data.nunique()** pour chercher les valeurs uniques de chaque paramètre.

```
ph                2785
Hardness          3276
Solids            3276
Chloramines       3276
Sulfate           2495
Conductivity      3276
Organic_carbon    3276
Trihalomethanes   3114
Turbidity         3276
Potability        2
dtype: int64
```

3. Valeurs manquantes

3.1 Visualiser des valeurs manquantes

```
print(data.isnull().sum())
```

Afin de savoir le nombre des valeurs manquantes de chaque features, on applique le code ci-dessus qui consiste à afficher les valeurs manquantes de chaque paramètre.

le resultat obtenu est le suivant

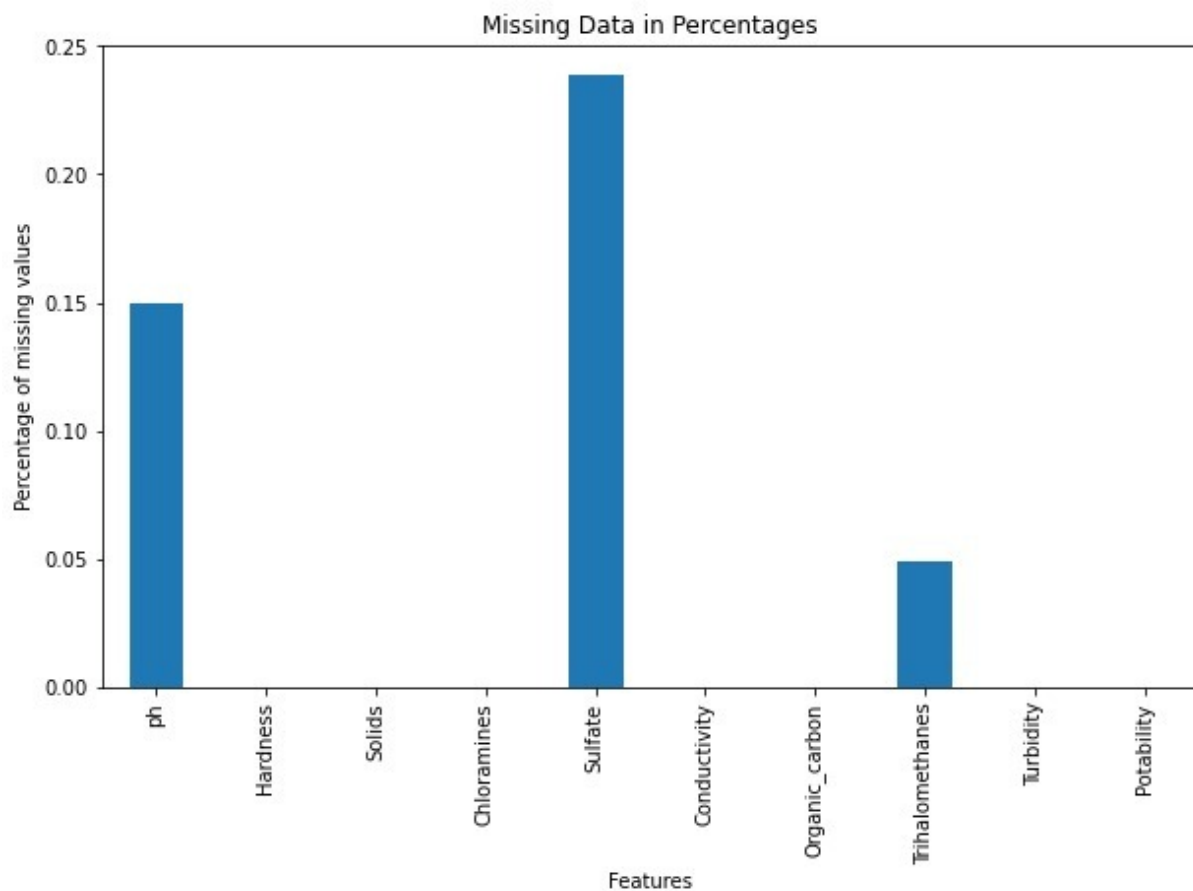
```
ph                491
Hardness          0
Solids            0
Chloramines       0
Sulfate           781
Conductivity      0
Organic_carbon    0
Trihalomethanes   162
Turbidity         0
Potability        0
dtype: int64
```




Pour afficher le pourcentage de données manquantes on applique le code suivant :

```
data.isnull().mean().plot.bar(figsize=(10,6))  
plt.ylabel('Percentage of missing values')  
plt.xlabel('Features')  
plt.title('Missing Data in Percentages')
```

Et on obtient le graphe ci-dessous qui montre qu'on a des valeurs manquantes pour le ph, le sulfate et le trihalomethanes d'un pourcentage qui respectivement de 15% , 24% et 5%





3.2 Remplissage des valeurs manquantes

On remarque que pour certains échantillons il y a des valeurs manquantes, dans ce cas il n'y a que les solutions suivantes :

1. Effacer tous les échantillons qui représentent des valeurs manquantes,
- 2. Compléter et remplir ces valeurs par la moyenne de chaque paramètre,**
3. Remplir les valeurs manquantes par une valeur comprise entre **le min** et **le max** de chaque paramètre.

Dans notre cas, on essayera la deuxième,

Par exemple si on trouve un élément avec une valeur manquante de pH

→ **Remplissage de cette valeur par la moyenne des PH des autres éléments.**

```
data [ 'ph' ] = data [ 'ph' ] . fillna ( data [ 'ph' ] . mean ())  
data [ 'Sulfate' ] = data [ 'Sulfate' ] . fillna ( data [ 'Sulfate' ] . mean ())  
data [ 'Trihalomethanes' ] = data [ 'Trihalomethanes' ] . fillna ( data [ 'Trihalomethanes' ].mean ())
```

→ Pour s'assurer qu'il n'y a plus de valeurs manquantes, on utilise la fonction **data.isnull().sum()** et on visualise le résultat suivant : (0 valeurs manquantes)

```
ph          0  
Hardness    0  
Solids      0  
Chloramines 0  
Sulfate     0  
Conductivity 0  
Organic_carbon 0  
Trihalomethanes 0  
Turbidity   0  
Potability  0
```



III. Application des méthodes :

1. Séparation des données et importation des librairies

Après la caractérisation des variables, on suit les 3 étapes suivantes commun entre les 3 méthodes avant de commencer à appliquer chaque méthode.

1.1 Séparation des données

Elle consiste à séparer les variables (features) X (x_1, x_2, x_3, \dots) de la variable Y (target ou label). Par conséquent la potabilité sera la valeur qu'on cherche à prédire en fonction des x (tous les autres colonnes).

```
X = data.drop ( 'Potability' , axis= 1 )
y = data ['Potability']
X .shape , y . shape
```

1.2 Importation des standardscaler vers l'échelle de performance

StandardScaler (échelle des normes) : elle recalibre les données pour des répartitions normales.

La fonction `fit_transform ()` est utilisée sur les données d'apprentissage afin que nous puissions mettre à l'échelle les données d'apprentissage et également apprendre les paramètres d'échelle de ces données. Ici, le modèle que nous avons construit va apprendre la moyenne et la variance des caractéristiques de l'ensemble d'apprentissage. Ces paramètres appris sont ensuite utilisés pour mettre à l'échelle nos données de test.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

1.3 Entraînement du model

Le but est de séparer le dataset entre les données d'apprentissage (70%) et les données du test (30%).

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```



2. La régression logistique

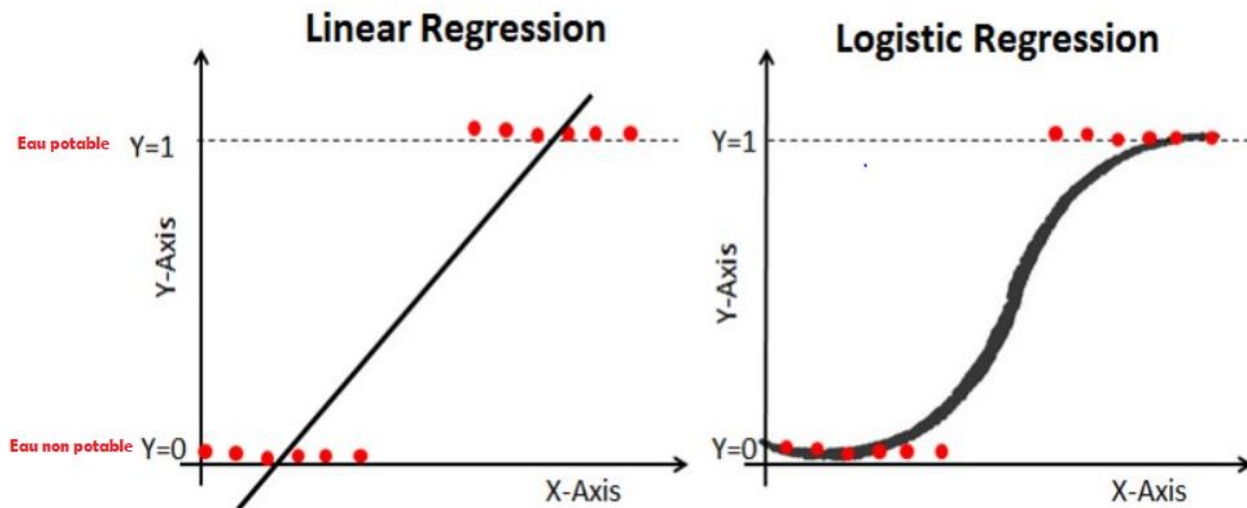
2.1 Principe

La régression logistique est un algorithme supervisé de classification. Il mesure la relation entre la variable dépendante catégorielle et une ou plusieurs variables indépendantes en donnant une estimation à la probabilité d'occurrence d'un événement à travers l'usage de sa fonction logistique.

Un modèle de régression logistique permet de résoudre les problèmes de classification binaires, et donc dans notre cas, de prédire dans notre cas **la probabilité** qu'un événement arrive (1) ou non (0) à partir de l'optimisation des coefficients de régression.

$$Y = \{1 \text{ si } f(X) \geq \text{Seuil} ; 0 \text{ si } f(X) < \text{Seuil}\}$$

La fonction qui remplit le mieux ces conditions est la fonction sigmoïde qui est schématisé de la manière suivante :



Le modèle de régression logistique donne des résultats plus pertinents, avec une certaine précision, comme il permet de résoudre les problèmes de la régression linéaire en termes de sensibilité aux outliers (erreurs), et de donner des probabilités. Son objectif est de trouver les valeurs optimales des paramètres qui font partie du modèle.



2.2 Application de la régression logistique selon python

a. Application de la régression logistique

Nous allons appliquer une régression logistique avec python en utilisant le package scikit-learn. Cette bibliothèque d'apprentissage automatique fournit pour la régression logistique le module `sklearn_model`.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
```

b. Création du modèle

Nous créons notre modèle de logistique de régression linéaire.

```
model_lg = LogisticRegression(max_iter=120, random_state=0, n_jobs=20)
```

c. Entrainement du modèle

Nous entraînons notre modèle en utilisant les données d'apprentissage (70 %).

```
model_lg.fit(X_train, y_train)
```

d. Test du modèle

Nous testons notre modèle de regression logistique sur les données de test (30%).

```
pred_lg = model_lg.predict(X_test)
```

e. Calcul du résultat de précision (Accuracy)

Le but c'est de savoir la pertinence de notre modèle.

```
lg = accuracy_score(y_test, pred_lg)
print(lg)
print(classification_report(y_test, pred_lg))
```

`Classification_report ()` c'est pour mesurer de prédiction



C'est la précision du modèle (Accuracy)

0.6286876907426246				
	precision	recall	f1-score	support
0	0.63	1.00	0.77	617
1	1.00	0.00	0.01	366
accuracy			0.63	983
macro avg	0.81	0.50	0.39	983
weighted avg	0.77	0.63	0.49	983

Cette méthode nous donne un nombre d'exactitude de 49% (basé sur le score F1) et 73% du pourcentage de rappel, tandis que la moyenne pondérée de précision a le nombre le plus élevé parmi les autres avec 77%.

NB : Après avoir ajuster les différents paramètres de logistique régression. l'accuracy obtenue est la meilleure trouvée après réglage.

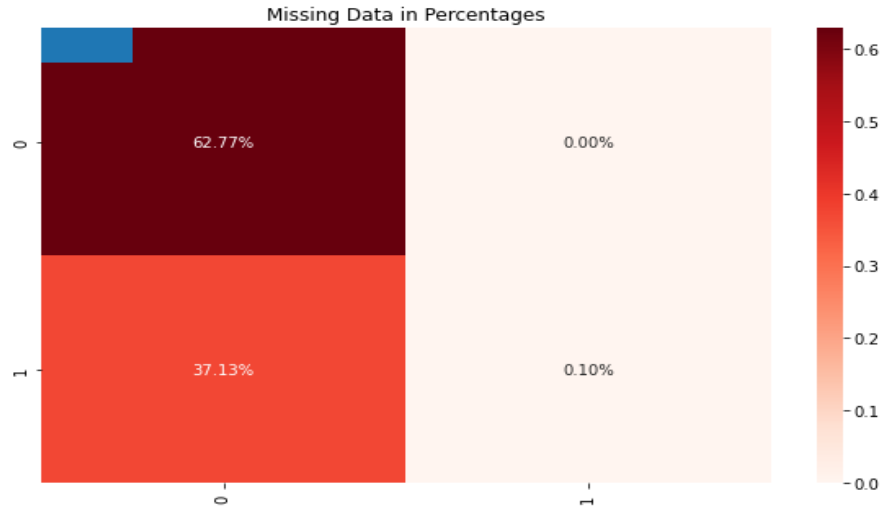
f. Matrice de confusion

Affichage de la matrice de confusion entre les données de sortie réelles et les données de sortie de notre modèle.

```
cm1 = confusion_matrix(y_test, pred_lg)
sns.heatmap(cm1/np.sum(cm1), annot = True, fmt= '0.2%', cmap = 'Reds')
```

A savoir qu'une Confusion Matrix (matrice de confusion) ou tableau de contingence permet de mesurer les performances d'un modèle de Machine Learning notamment la logistique de régression en vérifiant notamment à quelle fréquence ses prédictions sont exactes par rapport à la réalité dans des problèmes de classification.

TP : 62.77% ; TP :0.10% ; FN :37.13 ; TN : 0.00%



g. Application du Cross validation

Elle permet de mesurer la précision de notre modèle

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model_lg, X, y, cv=50)
print(scores)
print("%.2f accuracy with a standard deviation of %.2f" % (scores.mean(), scores.std()))
```

Cette méthode consiste à savoir si l'utilisation des premiers 75 % des données pour la formation et des derniers 25 % des données pour les tests est la meilleure façon de diviser les données. Plutôt que de trop se préoccuper de savoir quel bloc serait le meilleur pour les tests, la validation croisée les utilise tous, un par un, et résume les résultats à la fin. Par exemple, la validation croisée commencerait par utiliser les 3 premiers blocs pour entraîner la méthode. Ensuite, il utilise le dernier bloc pour tester la méthode, etc. A la fin, chaque bloc de données est utilisé pour les tests.

```
[0.60606061 0.60606061 0.60606061 0.62121212 0.60606061 0.60606061
0.60606061 0.60606061 0.60606061 0.60606061 0.60606061 0.60606061
0.60606061 0.60606061 0.60606061 0.60606061 0.60606061 0.60606061
0.60606061 0.60606061 0.61538462 0.61538462 0.61538462 0.61538462
0.61538462 0.61538462 0.63076923 0.61538462 0.61538462 0.61538462
0.61538462 0.61538462 0.61538462 0.61538462 0.61538462 0.61538462
0.61538462 0.6          0.61538462 0.61538462 0.61538462 0.61538462
0.6          0.6          ]
0.61 accuracy with a standard deviation of 0.01
```



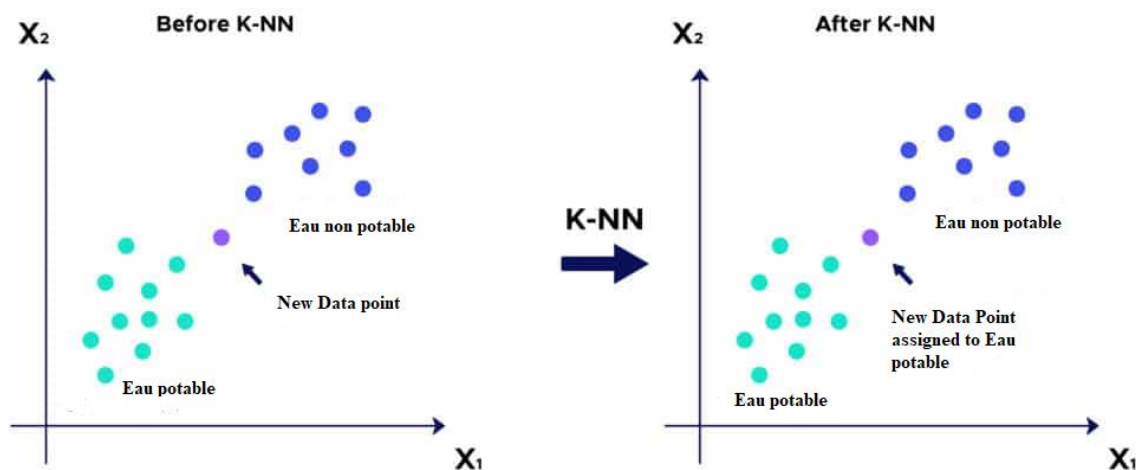
3. Méthode K-nearest neighbors (kNN)

3.1 Principe

La méthode des **K plus proches voisins (KNN)** a pour but de classer des points cibles (classe méconnue) en fonction de leurs distances par rapport à des points constituant un échantillon d'apprentissage (c'est-à-dire dont la classe est connue a priori).

K-NN est bien adapté aux problèmes de classification, où la relation entre les **features** est très **complexe** et difficile à comprendre.

L'algorithme de Nearest Neighbor (voisin le plus proche) permet de résoudre des problèmes de classification à plusieurs classes de façon simple et très efficace.



Les features sont généralement transformées dans une fourchette de nombres avant l'application de l'algorithme k-NN.

La formule de calcul de la distance dépend de la façon dont les features sont mesurées. Si certaines features ont des valeurs beaucoup plus grandes que d'autres. Les mesures de distance seront fortement affectées par des valeurs plus grandes.

Nous devons redimensionner les features pour qu'elles soient comparables sur une échelle commune. Chaque feature contribue de manière relativement égale à la formule de distance (La normalisation et la standardisation). **Min-Max normalisation**



peut être appliqué quand les données varient dans des échelles différentes. À l'issue de cette transformation, les features seront comprises dans un intervalle fixe $[0, 1]$. Le but d'avoir un tel intervalle restreint est de réduire l'espace de variation des valeurs d'une feature et par conséquent réduire l'effet des outliers. **La standardisation** quant à elle peut également être appliquée quand les features ont des unités différentes. La standardisation est le processus de transformer une feature en une autre qui répondra à la loi normale (Gaussian Distribution).

3.2 Application de la méthode K-nearest neighbors (kNN) selon python

a. Importer les librairies

En utilisant le code ci-dessous, on importe d'abord les librairies.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
```

b. Création du modèle

On crée un objet modèle où les paramètres indiqués dans le code :

```
model_kn = KNeighborsClassifier(n_neighbors=9, leaf_size=20)
```

PS :

- Pour Le Neighbors ou k, on a trouvé que 9 est la valeur où on a une meilleure Accuracy.
- Pour Le Leaf_size, on a trouvé que 20 est la valeur où on a une meilleure Accuracy.

c. Entraîner le modèle:

```
model_kn.fit(X_train, y_train)
```



d. Prédiction sur l'ensemble de test

```
pred_kn = model_kn.predict(X_test)
```

e. Accuracy

```
kn = accuracy_score(y_test, pred_kn)  
print(kn)
```

En utilisant le code ci-dessus, on calcul l'accuracy et on trouve :

```
0.6534195933456562
```

Classification_report() est utilisé pour mesurer la qualité de prédiction.

```
print(classification_report(y_test, pred_kn))
```

```
In [59]: print(classification_report(y_test, pred_kn))
```

	precision	recall	f1-score	support
0	0.69	0.82	0.75	680
1	0.55	0.37	0.44	402
accuracy			0.65	1082
macro avg	0.62	0.60	0.59	1082
weighted avg	0.64	0.65	0.63	1082

Cette méthode nous donne un nombre d'exactitude de 63% (basé sur le score F1) et 64% de la moyenne pondérée de précision, tandis que le pourcentage de rappel a le nombre le plus élevé parmi les autres avec 65%.

f. Confusion matrix

D'abord, on importe les librairies :

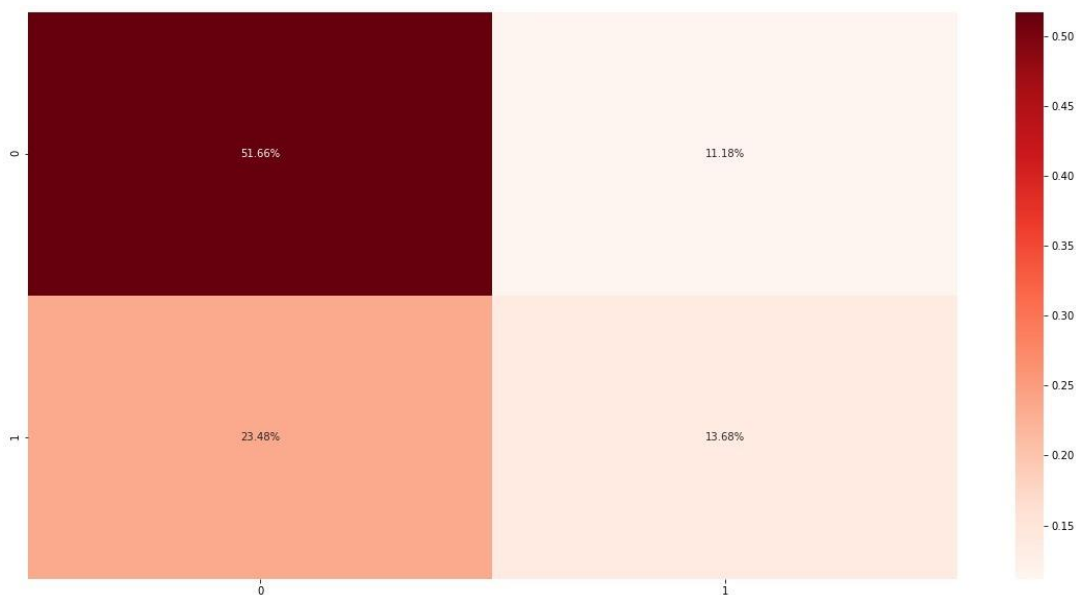
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report  
import seaborn as sns
```



La fonction **confusion_matrix ()**, c'est un tableau qui sert à décrire les performances d'un modèle de classification sur un ensemble de données de test dont les vraies valeurs sont connues.

```
cm1 = confusion_matrix(y_test, pred_kn)
sns.heatmap(cm1/np.sum(cm1), annot = True, fmt= '0.2%', cmap = 'Reds')
```

TP : 51.66 % ; TN : 23.48 % ; FP : 11.18 % ; FN : 13.68 %



a. Application du Cross validation

Elle permet de mesurer la précision de notre modèle

```
#Application de cross validation

scores = cross_val_score(model_kn, X, y, cv=50)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

Cette méthode consiste à savoir si l'utilisation des premiers 75 % des données pour la formation et des derniers 25 % des données pour les tests est la meilleure façon de diviser les données. Plutôt que de trop se préoccuper de savoir quel bloc serait le meilleur pour



les tests, la validation croisée les utilise tous, un par un, et résume les résultats à la fin. Par exemple, la validation croisée commencerait par utiliser les 3 premiers blocs pour entraîner la méthode. Ensuite, il utilise le dernier bloc pour tester la méthode, etc. A la fin, chaque bloc de données est utilisé pour les tests.

Et 0,61 présente la moyenne des 50 valeurs

```
[0.60606061 0.60606061 0.60606061 0.62121212 0.60606061 0.60606061
0.60606061 0.60606061 0.60606061 0.60606061 0.60606061 0.60606061
0.60606061 0.60606061 0.60606061 0.60606061 0.60606061 0.60606061
0.60606061 0.60606061 0.60606061 0.60606061 0.60606061 0.60606061
0.60606061 0.60606061 0.61538462 0.61538462 0.61538462 0.61538462
0.61538462 0.61538462 0.63076923 0.61538462 0.61538462 0.61538462
0.61538462 0.61538462 0.61538462 0.61538462 0.61538462 0.61538462
0.61538462 0.61538462 0.61538462 0.61538462 0.61538462 0.61538462
0.6 0.6 ]
0.61 accuracy with a standard deviation of 0.01
```

4. Méthode Support Vector Machine (SVM)

4.1 Principe

Support Vector Machines (SVMs) sont un ensemble de méthodes d'apprentissage supervisé utilisées pour la classification, la régression et la détection des valeurs aberrantes.

Les avantages des machines à vecteurs de support sont :

- Efficace dans les espaces de grande dimension.
- Toujours efficace dans les cas où le nombre de dimensions est supérieur au nombre d'échantillons.
- Utilise un sous-ensemble de points d'apprentissage dans la fonction de décision (appelés vecteurs de support), il est donc également efficace en mémoire.
- Polyvalent : différentes fonctions du noyau peuvent être spécifiées pour la fonction de décision. Des noyaux communs sont fournis, mais il est également possible de spécifier des noyaux personnalisés.

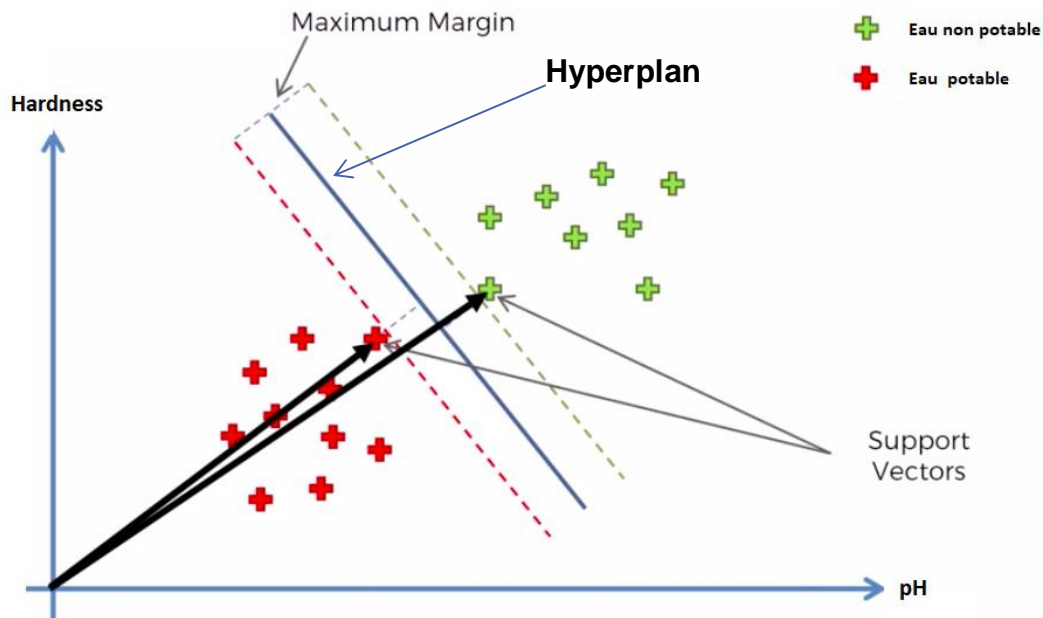
Les inconvénients des machines à vecteurs de support incluent :

- Si le nombre de caractéristiques est bien supérieur au nombre d'échantillons, évitez le sur-ajustement dans le choix des fonctions du noyau et le terme de régularisation est crucial.



- Les SVM ne fournissent pas directement d'estimations de probabilité, celles-ci sont calculées à l'aide d'une validation croisée quintuple coûteuse

On prend l'exemple de 2 features dans notre cas (pH et Hardness) : hyperplan



Intuitivement, plus nos points de données sont éloignés de l'hyperplan, plus nous sommes convaincus qu'ils ont été bien classés.

Il faut donc maximiser **la marge** (la distance entre l'hyperplan et le point de données le plus proche de l'un des ensembles).

→ Le but est donc que nos points de données soient aussi éloignés que possible de **l'hyperplan**, ce qui augmente les chances que les nouvelles données soient classées correctement.

→ C'est la solution avec la marge maximale (Maximum margin).

Support vectors : les points de données de chaque classe les plus proches de l'hyperplan.

+ Chaque classe (potability) a au moins 1 support vector.



Avec les supports vectors, il est possible de reconstruire l'hyperplan.

+ Nous pouvons stocker le modèle de classification même lorsque nous avons des millions de Features (ce cas 9 features mais on a représenté juste 2).

4.2 Application de la méthode *Support Vector Machines (SVMs)* selon python

a. Importer les librairies

En utilisant le code ci-dessous, on importe les librairies.

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import seaborn as sns
from sklearn.svm import SVC
```

b. Création du modèle

On crée un objet modèle où les paramètres indiqués dans le code :

```
model_svm = SVC(kernel='rbf', random_state = 42)
```

c. Entraîner le modèle

```
model_svm.fit(X_train, y_train)
```

d. Prédiction sur l'ensemble de test

```
pred_svm = model_svm.predict(X_test)
```

e. Accuracy :

```
sv = accuracy_score(y_test, pred_svm) print(sv)
```

En utilisant le code ci-dessus, on calcul l'accuracy et on trouve :



0.6927772126144456

Classification_report() est utilisé pour mesurer la qualité de prédiction.

```
print(classification_report(y_test, pred_svm))
```

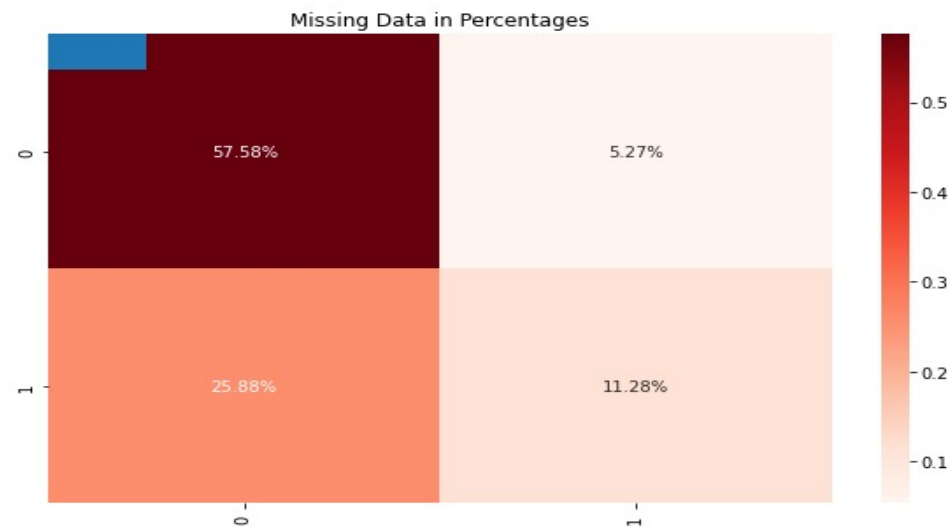
	precision	recall	f1-score	support
0	0.69	0.93	0.79	617
1	0.71	0.30	0.42	366
accuracy			0.69	983
macro avg	0.70	0.61	0.60	983
weighted avg	0.70	0.69	0.65	983

Cette méthode nous donne un nombre d'exactitude de 65% (basé sur le score F1) et 69% du pourcentage de rappel, tandis que la moyenne pondérée de précision a le nombre le plus élevé parmi les autres avec 70%.

f. Confusion matrix

La fonction **confusion_matrix ()**, c'est un tableau qui sert à décrire les performances d'un modèle de classification sur un ensemble de données de test dont les vraies valeurs sont connues.

TP : 57.58 % ; TN : 25.88 % ; FP : 5.27 % ; FN : 11.28 %





g. Application du Cross validation

Elle permet de mesurer la précision de notre modèle

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model_svm, X, y, cv=50)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

Cette méthode consiste à savoir si l'utilisation des premiers 75 % des données pour la formation et des derniers 25 % des données pour les tests est la meilleure façon de diviser les données. Plutôt que de trop se préoccuper de savoir quel bloc serait le meilleur pour les tests, la validation croisée les utilise tous, un par un, et résume les résultats à la fin. Par exemple, la validation croisée commencerait par utiliser les 3 premiers blocs pour entraîner la méthode. Ensuite, il utilise le dernier bloc pour tester la méthode, etc. A la fin, chaque bloc de données est utilisé pour les tests. **(pour mieux comprendre le principe du Cross validation voir annexe 2)**

Et 0,67 présente la moyenne des 50 valeurs

```
[0.75757576 0.72727273 0.66666667 0.71212121 0.68181818 0.60606061
0.63636364 0.63636364 0.66666667 0.62121212 0.59090909 0.65151515
0.71212121 0.75757576 0.74242424 0.69696967 0.71212121 0.69696967
0.68181818 0.63636364 0.71212121 0.68181818 0.65151515 0.71212121
0.77272727 0.60606061 0.63076923 0.63076923 0.55384615 0.73846154
0.64615385 0.66153846 0.61538462 0.64615385 0.66153846 0.66153846
0.55384615 0.61538462 0.73846154 0.69230769 0.67692308 0.72307692
0.70769231 0.63076923 0.63076923 0.72307692 0.75384615 0.72307692
0.70769231 0.66153846]
0.67 accuracy with a standard deviation of 0.05
```



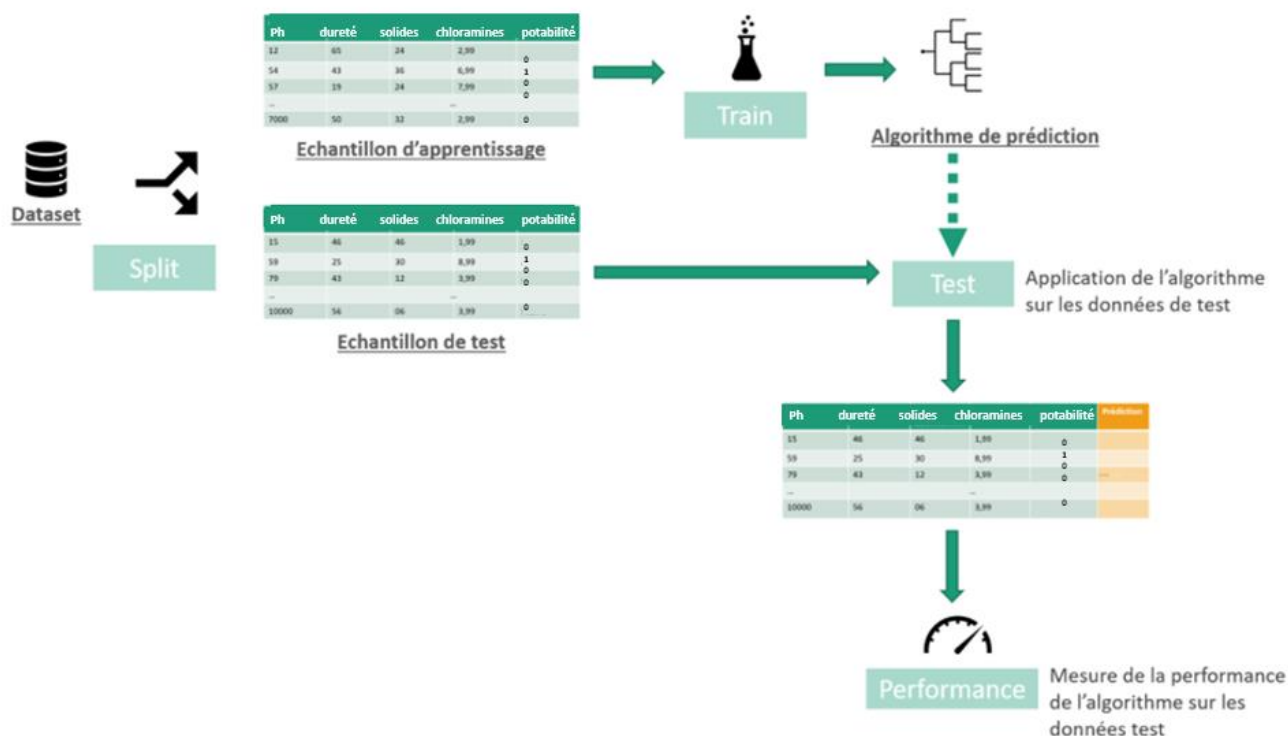

IV. Procédure commune de mesure de la performance d'un algorithme :

1. Méthodes de mesure de performance

Au démarrage de notre projet, on dispose d'un dataset avec tous les échantillons s'ils ont été potable ou non. Ce dataset, vous allez devoir le séparer en 2 parties :

- Un dataset d'apprentissage : données utilisées pour notre modèle
- Un dataset de test : une fois votre modèle entraîné sur l'échantillon d'apprentissage, on peut l'appliquer sur le dataset de test. On aura donc le statut réel de ces échantillons ainsi que la prédiction faite par votre modèle.

La mesure de la performance se fait toujours sur l'échantillon de test. Pourquoi ? Tout simplement parce que von doit tester la performance de notre modèle sur des données qui n'ont pas été utilisées pour construire le modèle. Le schéma ci-dessous illustre toutes ces étapes.





Plusieurs indicateurs permettent **de mesurer la performance des modèles**. Chacun a ses spécificités et il faut bien souvent en utiliser plusieurs pour avoir une vision complète de la performance de votre modèle.

1.1 La matrice de confusion

(Matrice de confusion) ou tableau de contingence est un outil permettant de mesurer les performances d'un modèle de Machine Learning en vérifiant notamment à quelle fréquence ses prédictions sont exactes par rapport à la réalité dans des problèmes de classification.

Pour bien comprendre le fonctionnement d'une matrice de confusion, il convient de bien comprendre les quatre terminologies principales : TP, TN, FN et FN. Voici la définition précise de chacun de ces termes :

Les bonnes prédictions :

- TP (True Positives) : les cas où la prédiction est positive, et où la valeur réelle est effectivement positive.

L'algorithme juge que l'échantillon est une eau potable, et cet échantillon est bel et bien potable en réalité.

- TN (True Negatives) : les cas où la prédiction est négative, et où la valeur réelle est effectivement négative.

L'algorithme juge que l'échantillon est une eau non potable, et cet échantillon n'est effectivement pas potable en réalité.

Les prédictions fausses :



- FP (False Positive) : les cas où la prédiction est positive, mais où la valeur réelle est négative.

L'algorithme juge que l'échantillon est une eau potable, et cet échantillon est une eau non potable en réalité.

- FN (False Negative) : les cas où la prédiction est négative, mais où la valeur réelle est positive.

L'algorithme juge que l'échantillon est une eau non potable, mais cet échantillon est une eau potable en réalité.



		Actual 	
		0	1
Predicted 	0	High TN	Low FN
	1	Low FP	High TP

Trois indicateurs sont adaptés pour évaluer la performance d'un modèle de classification et qui sont calculés à partir de la matrice de confusion. Ils sont assez simples à comprendre et sont très complémentaires : l'accuracy, le recall et la precision

L'indicateur le plus simple est **l'accuracy** : il indique le pourcentage de bonnes prédictions. C'est un très bon indicateur parce qu'il est très simple à comprendre.

$$\text{Accuracy} = \frac{\text{Vrai positif} + \text{Vrai négatif}}{\text{Total}}$$

1.2 Principe Cross validation

La validation croisée comporte deux étapes principales : la division des données en sous-ensembles (appelés plis) et la rotation de la formation et de la validation entre ces sous-ensembles. La technique de division a généralement les propriétés suivantes:

- Chaque pli a approximativement la même taille.
- Les données peuvent être sélectionnées aléatoirement dans chaque pli ou stratifiées.



- Tous les plis sont utilisés pour former le modèle sauf un, qui est utilisé pour la validation. Ce pli de validation doit être tourné jusqu'à ce que tous les plis soient devenus un pli de validation une fois et une seule.

Il est recommandé que chaque exemple soit contenu dans un et un seul pli.

L'un des avantages de la CV est d'observer les prédictions du modèle par rapport à toutes les instances de l'ensemble de données. Cela garantit que le modèle a été testé sur l'ensemble des données sans les tester simultanément. Des variations sont attendues à chaque étape de la validation ; par conséquent, le calcul de la moyenne et de l'écart-type peut réduire les informations à quelques valeurs de comparaison.

1.3 Accuracy

La précision nous renseigne sur le nombre de points de données correctement classés par rapport au nombre total de points de données. Comme son nom l'indique, la précision indique dans quelle mesure les valeurs prédites sont proches des valeurs cibles.

Précision = # de points correctement classés / # de points totaux

Mais la précision n'est pas toujours une bonne mesure pour l'évaluation des modèles en raison de la simplicité de son hypothèse.

1.4 Représentation graphique de données

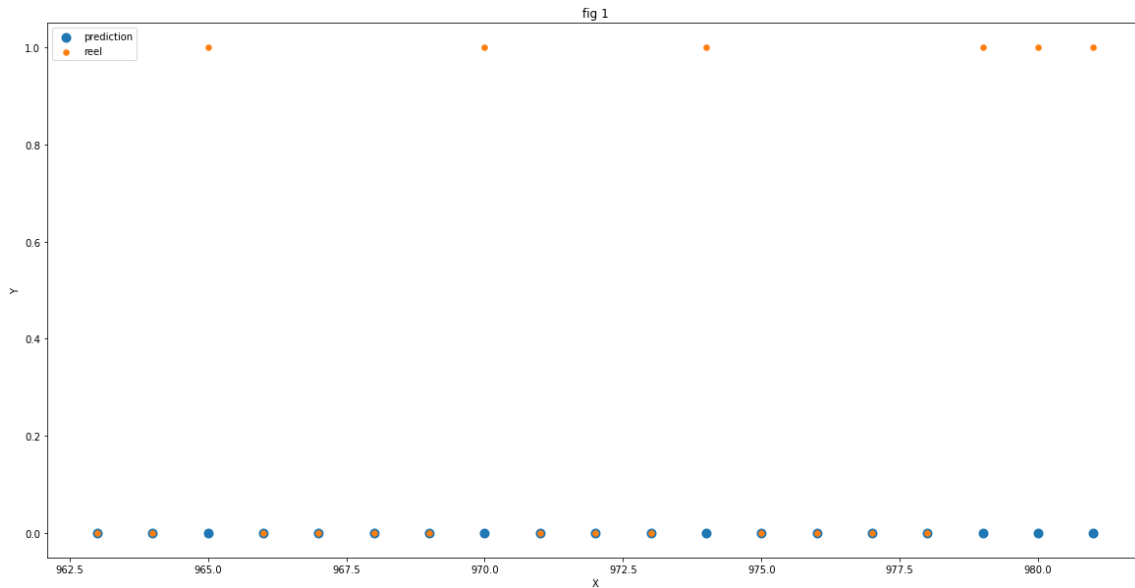
Pour mieux visualiser la performance du modèle on peut tracer une courbe à l'aide du code ci-dessous :

On importe d'abord `matplotlib.pyplot` qui est une interface basée sur l'état à `matplotlib`. Elle fournit une manière implicite, semblable à MATLAB, de tracer. Elle ouvre également les figures sur votre écran, et agit comme le gestionnaire de l'interface graphique des figures.

```
import matplotlib.pyplot as plt
plt.figure()
# cree des courbes
x = list(range(len(pred_lg)))
print(len(x))
plt.scatter(x[-20:-1], pred_lg[-20:-1], lw=4, label = 'prediction') # la 1er courbe
plt.scatter(x[-20:-1], y_test[-20:-1], lw=0.5, label = 'reel') # 2 eme courbe # 3 eme courbe
plt.title('fig 1') # nomer la figure
plt.xlabel('X') # nomer l'axe des X
plt.ylabel('Y') # nomer l'axe des y
plt.legend() # sert a cree une legend
plt.show()
```



Comme exemple on a testé le model de la régression logistique afin de visualiser sa performance et on obtenue le graphe suivant :



2. Comparaison des accuacy-score (obtenus à partir de la matrice de confusion) résultants des trois algorithmes de classification utilisés à l'aide de Python

L'accuracy_score calculé pour évaluer la performance d'un modèle de classification est tiré à partir de la matrice de confusion :

→ Code :

```
# Comparaison des methodes

from sklearn.metrics import accuracy_score
lg = accuracy_score(y_test, pred_lg)
kn = accuracy_score(y_test, pred_kn)
sv = accuracy_score(y_test, pred_svm)

models = pd.DataFrame({'Model': ['Logistic Regression', 'KNeighbours', 'SVM'], 'Accuracy_score': [lg, kn, sv]})
models
sns.barplot(x='Accuracy_score', y='Model', data=models)

models.sort_values(by='Accuracy_score', ascending=False)
```



→ **Exécution :**

Le code en dessus permet de regrouper les accuracy—score obtenus (calculé à partir de la matrice de confusion) par chaque algorithme étudié en utilisant la fonction DataFrame (can be created using a single list or a list of lists):

- Régression logistique
- K-voisins les plus proches (KNN)
- Machine à vecteurs de soutien (SVM)

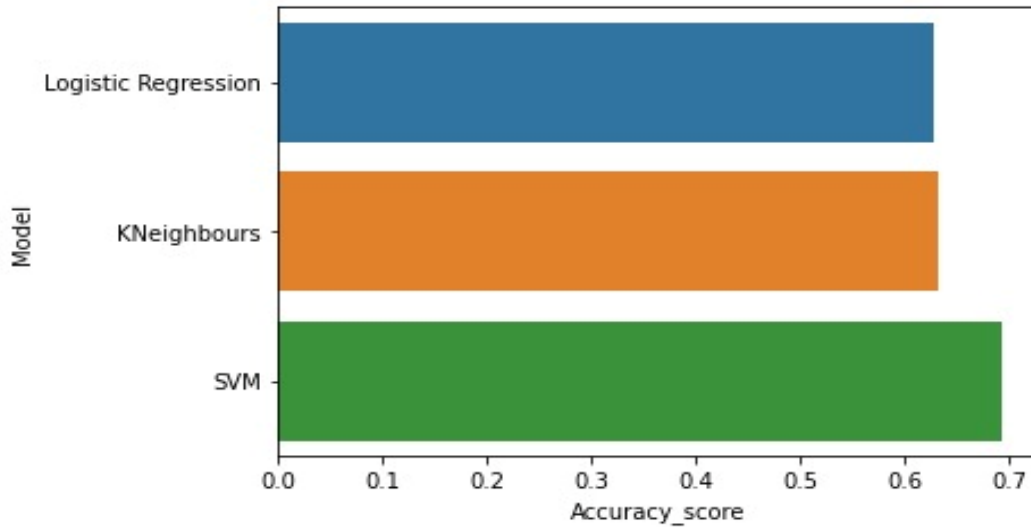
```
Out[2]:
```

	Model	Accuracy_score
2	SVM	0.692777
1	KNeighbours	0.632757
0	Logistic Regression	0.628688

On peut déduire que : Accuracy-score : SVM : 69 % - K-NN : 63 % - L-R : 62 %

N.B : notons que ces accuracy-score sont les meilleurs obtenus pour chaque algorithmes après avoir testé plusieurs valeurs des paramètres déterminants cet accuracy score, c'est le rôle de l'ingénieur en Machine learning de trouver le meilleur accuracy pour faire une meilleure interprétation de la performance d'un algorithme par rapport à l'autre en cherchant une solution à notre problème de classification binaire.

Pour visualiser ce résultat sous forme de graphique, on fait appel à Seaborn :une bibliothèque de visualisation de données Python basée sur Matplotlib. Il fournit une interface de haut niveau pour dessiner des graphiques statistiques attrayants et informatifs. Ceci grâce à la fonction sns.barplot. Le résultat d'exécution est affiché comme suit :



La comparaison des résultats des méthodes de classification

→ **Interprétation :**

Sur la base de la pratique que nous avons menée, résumée dans le tableau et le graphique des accuracy-score, la méthode qui obtient l'accuracy-score le plus élevé est SVM Support Vector Machines de (69 %).

Cela peut être expliqué par le fait que l'algorithme est basé sur 3 astuces pour obtenir de très bonnes performances tant en qualité de prédiction qu'en complexité de calcul :

- On cherche l'hyperplan comme solution d'un problème d'optimisation sous-conainte. La fonction à optimiser intègre un terme de qualité de prédiction et un terme de complexité du modèle.
- Le passage à la recherche de surfaces séparatrices non linéaires est introduit en utilisant un noyau kernel qui code une transformation non linéaire des données.
- Numériquement, toutes les équations s'obtiennent en fonction de certains produits scalaires utilisant le noyau et certains points de la base de données (ce sont les Support Vectors).



3. Comparaison des accuracy-score obtenus à partir de la cross-validation appliquée aux trois algorithmes de classification utilisés :

→ Interprétation :

On utilise couramment la Cross-Validation en Machine Learning pour **comparer différents modèles** et sélectionner le plus approprié pour un problème spécifique. Elle est à la fois simple à comprendre, simple à implémenter et moins biaisée que les autres méthodes. Découvrons à présent les principales techniques de validation croisée.

Méthodes	Accuracy_score de la Cross validation (La moyenne obtenue)
SVM	0.67
K-NN	0.63
Régression Logistique	0.61

On peut donc calculer l'accuracy selon la cross validation, et non pas en passant par la matrice de confusion.

On remarque que :

D'une façon générale l'accuracy_score calculé rejoint les résultats obtenus, la méthode de SVM Support Vector Machines est considérée comme la meilleure car elle donne l'accuracy_score le plus élevé 67% qui dépasse celui de la méthode K-NN 63% et de la régression logistique 61%.



V. Les annexes

Description des données

```
#Importation des librairies
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#importation de la base de donnee
data = pd.read_csv("C:\\Users\\hp\\Desktop\\water_potability.csv")

data.head() #afficher les 5 premiers echantillons
print(data.shape) #definir taille de la matrice
print(data.columns) #afficher les titres des colonnes

data.describe() #decrire la base de donnee

data.info() #nombre de valeur non nulle par variable, type de la valeur et taille(entier ou autres) des donnees

plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot= True, cmap='coolwarm') #visualiser la correlation entre les features

ax = sns.countplot(x = "Potability",data= data, saturation=0.8) # probabilite d'eau potable et de l'eau non potable
plt.xticks(ticks=[0, 1], labels = ["Not Potable", "Potable"])
plt.show()

#visualiser la distribution des parametres
plt.rcParams['figure.figsize'] = [20,10]
data.hist()
plt.show()

sns.violinplot(x='Potability', y='ph', data=data, palette='rocket') #visualiser la relation entre le ph et la potabilite
sns.violinplot(x='Potability', y='Solids', data=data, palette='rocket') #visualiser la relation entre solides et la potabilite

print(data.nunique()) #compter les valeurs uniques de chaque variable
print(data.isnull().sum())# afficher le nombre des valeurs manquantes par feature

data.isnull().mean().plot.bar(figsize=(10,6)) #pourcentage des donnees manquantes
plt.ylabel('Percentage of missing values')
plt.xlabel('Features')
plt.title('Missing Data in Percentages')

#remplissage des donnees manquantes
data [ 'ph' ] = data [ 'ph' ] . fillna ( data [ 'ph' ] . mean())
data [ 'Sulfate' ] = data [ 'Sulfate' ] . fillna ( data [ 'Sulfate' ] . mean ())
data [ 'Trihalomethanes' ] = data [ 'Trihalomethanes' ] . fillna ( data [ 'Trihalomethanes' ].mean ())

# afficher le nombre des valeurs manquantes par feature
data .isnull().sum()
```



Séparation des données

```
#separation des donnees

X = data . drop ( 'Potability' , axis= 1 )
y = data ['Potability']
X .shape , y . shape

# importer StandardScaler to perform scaling

from sklearn.preprocessing import StandardScaler

#definir un echelle
scaler = StandardScaler()
X = scaler.fit_transform(X)
X

# importer train-test split
from sklearn.model_selection import train_test_split

# Séparation du Dataset entre les données d'apprentissage (70%) et les données de test (30%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

Methode de la regression logistique



```
#Methode de la regression logistique

#importation des librairies
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# creation du modele
model_lg = LogisticRegression(max_iter=120,random_state=0, n_jobs=20)

# entrainement du modele
model_lg.fit(X_train, y_train)

# prediction sur l'ensemble du teste
pred_lg = model_lg.predict(X_test)

# Calcul de l'Accuracy Score
lg = accuracy_score(y_test, pred_lg)

#afficher la valeur de la precision
print(lg)

# mesurer la qualite de prediction
print(classification_report(y_test,pred_lg))

# afficher la matrice de confusion
cm1 = confusion_matrix(y_test, pred_lg)
sns.heatmap(cm1/np.sum(cm1), annot = True, fmt= '0.2%', cmap = 'Reds')

#Application de cross validation

#Importer les librairies
from sklearn.model_selection import cross_val_score

#appliquer le cross validation
scores = cross_val_score(model_lg, X, y, cv=50)
#afficher les differentes score
print(scores)
#calculer la moyenne des valeurs
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))

#representation graphique
import matplotlib.pyplot as plt
plt.figure()
# cree des courbes
x = list(range(len(pred_lg)))
print(len(x))
plt.scatter(x[:-20:-1],pred_lg[:-20:-1],lw=4,label = 'prediction') # la 1er courbe
plt.scatter(x[:-20:-1],y_test[:-20:-1],lw=0.5,label = 'reel') # 2 eme courbe # 3 eme courbe
plt.title('fig 1') # nomer la figure
plt.xlabel('X') # nomer l'axe des X
plt.ylabel('Y') # nomer l'axe des y
plt.legend() # sert a cree une legend
plt.show()
```



Méthode de K-nearest neighbors (kNN)

```
#Methode de K-nearest neighbors

#import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
#from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
#from sklearn import preprocessing
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report

# Creation du model
model_kn = KNeighborsClassifier(n_neighbors=9, leaf_size=20)
# entraînement du model
model_kn.fit(X_train, y_train)

# prediction sur l'ensemble du teste
pred_kn = model_kn.predict(X_test)
# Calcul de l'Accuracy Score
kn = accuracy_score(y_test, pred_kn)

#afficher la valeur de la precision
print(kn)

# mesurer la qualite de prediction
print(classification_report(y_test,pred_kn))

from sklearn.metrics import confusion_matrix
import seaborn as sns
# confusion Maxtrix
cm1 = confusion_matrix(y_test, pred_kn)
sns.heatmap(cm1/np.sum(cm1), annot = True, fmt= '0.2%', cmap = 'Reds')

#Application de cross validation

scores = cross_val_score(model_kn, X, y, cv=50)
print(scores)
print("%.2f accuracy with a standard deviation of %.2f" % (scores.mean(), scores.std()))
```



Methode de Support Vector Machine (SVM)

```
#Methode de Support Vector Machine (SVM)

#from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import seaborn as sns
from sklearn.svm import SVC

#creation du model
model_svm = SVC(kernel='rbf', random_state = 42)

#entrainement du model
model_svm.fit(X_train, y_train)

# Making Prediction
pred_svm = model_svm.predict(X_test)

# calcul de l'Accuracy Score
sv = accuracy_score(y_test, pred_svm)
# afficher la valeur de la precision
print(sv)
# mesurer la qualite de prediction
print(classification_report(y_test,pred_svm))

# confusion Maxtrix
cm6 = confusion_matrix(y_test, pred_svm)
sns.heatmap(cm6/np.sum(cm6), annot = True, fmt= '0.2%', cmap = 'Reds')

#Application de cross validation

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_score

scores = cross_val_score(model_svm, X, y, cv=50)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```