# Compte rendu KNN

## **Partie 1** : Base de données, Analyse, Prétraitement et Préparation

### 1) Importation des bibliothèques

Entrée [12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

### 2) Visualisation des données

Entrée [13]:
```
dataset = pd.read_csv("combined_csv.csv")
dataset
```

Out[13]:

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 | Score | Interpretation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | a | a | a | a | a | 1 | 1 | 1 | 1 | 1 | 10 | B |
| **1** | b | b | b | b | b | 2 | 2 | 2 | 2 | 2 | 0 | C |
| **2** | c | c | c | c | c | 3 | 3 | 3 | 3 | 3 | 20 | A |
| **3** | a | b | c | a | b | 1 | 2 | 3 | 1 | 2 | 8 | C |
| **4** | b | c | a | c | a | 3 | 2 | 3 | 1 | 2 | 11 | B |

#### 3.1) Traitement des valeurs manquantes NaN

Entrée [14]:
```
dataset.isnull().sum()
```

Out[14]:
```
Q1                8
Q2               12
Q3               13
Q4                9
Q5               14
Q6               12
Q7               10
Q8               12
Q9               10
Q10               8
Score             0
Interpretation    0
dtype: int64
```

On observe plusieurs valeurs manquantes. On peut soit les remplacer par des valeurs nulles (b où 2), où utiliser la fonction "mode" de pandas. Celle-ci va rendre compte des valeurs les plus rendondantes dans chaque colonne du dataset.

Entrée [15]:
```
dataset.mode().iloc[0]
```

Out[15]:
```
Q1                a
Q2                b
Q3                c
Q4                a
Q5                a
Q6                2
Q7                3
Q8                2
Q9                1
Q10               2
Score             8
Interpretation    C
Name: 0, dtype: object
```

On remplace les NaN par les données explicitées ci-dessus.

Entrée [16]:
```
dataset.fillna(dataset.mode().iloc[0], inplace=True)
dataset
```

Out[16]:

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 | Score | Interpretation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | a | a | a | a | a | 1 | 1 | 1 | 1 | 1 | 10 | B |
| **1** | b | b | b | b | b | 2 | 2 | 2 | 2 | 2 | 0 | C |
| **2** | c | c | c | c | c | 3 | 3 | 3 | 3 | 3 | 20 | A |
| **3** | a | b | c | a | b | 1 | 2 | 3 | 1 | 2 | 8 | C |
| **4** | b | c | a | c | a | 3 | 2 | 3 | 1 | 2 | 11 | B |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **210** | c | c | c | c | c | 2 | 3 | 3.0 | 3 | 3 | 14 | B |
| **211** | b | a | c | a | b | 2 | 2.0 | 2 | 3 | 2 | 5 | C |
| **212** | a | c | b | a | a | 1.0 | 3 | 3.0 | 2 | 3 | 8 | C |
| **213** | a | n | e | a | b | v | t | Y | ' | 1 | 3 | C |
| **214** | A | b | c | lk | k | n | 2 | 0 | 5 | 3 | 4 | C |

215 rows × 12 columns

On vérifie qu'il ne nous reste plus de valeurs manquantes NaN.

Entrée [17]:
```
dataset.isnull().sum()
```

Out[17]:
```
Q1                0
Q2                0
Q3                0
Q4                0
Q5                0
Q6                0
Q7                0
Q8                0
Q9                0
Q10               0
Score             0
Interpretation    0
dtype: int64
```

### 3.2) Traitement des valeurs erronées

Notre hypothèse de travail sera la suivante:
1) On va devoir séparer les valeurs de notre jeu de données en deux: True et False.
2) Ensuite, on remplace l'une des deux valeurs (qui représente les valeurs erronnées) par un NaN.
3) Enfin, on applique la solution effectuée à la question précédente, pour remplacer les NaN par les valeurs de la fonction .mode()

#### 3.2.1) Remplacement des valeurs par True et False

On va devoir remplacer toutes les valeurs qui ne correspondent pas à a, b, c ou 1,2 et 3.

Entrée [18]:
```
dataset.iloc[:, 0:10].isin(["a", "b", "c", "1", "2", "3"])
```

Out[18]:

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | True | True | True | True | True | True | True | True | True | True |
| **1** | True | True | True | True | True | True | True | True | True | True |
| **2** | True | True | True | True | True | True | True | True | True | True |
| **3** | True | True | True | True | True | True | True | True | True | True |
| **4** | True | True | True | True | True | True | True | True | True | True |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **210** | True | True | True | True | True | True | True | False | True | True |
| **211** | True | True | True | True | True | True | False | True | True | True |
| **212** | True | True | True | True | True | False | True | False | True | True |
| **213** | True | False | False | True | True | False | False | False | False | True |
| **214** | False | True | True | False | False | False | True | False | False | True |

215 rows × 10 columns

#### 3.2.2) Remplacement des valeurs erronnées par des NaN

On écrase notre dataset originel par le nouveau dataset dans lequel les valeurs erronées sont remplacées par des NaN.

Entrée [19]:
```
dataset.iloc[:, 0:10] = dataset.iloc[:, 0:10].where(dataset.iloc[:, 0:10].isin(["a", "b", "c", "1", "2", "3"]))
dataset
```

Out[19]:

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 | Score | Interpretation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | a | a | a | a | a | 1 | 1 | 1 | 1 | 1 | 10 | B |
| **1** | b | b | b | b | b | 2 | 2 | 2 | 2 | 2 | 0 | C |
| **2** | c | c | c | c | c | 3 | 3 | 3 | 3 | 3 | 20 | A |
| **3** | a | b | c | a | b | 1 | 2 | 3 | 1 | 2 | 8 | C |
| **4** | b | c | a | c | a | 3 | 2 | 3 | 1 | 2 | 11 | B |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **210** | c | c | c | c | c | 2 | 3 | NaN | 3 | 3 | 14 | B |
| **211** | b | a | c | a | b | 2 | NaN | 2 | 3 | 2 | 5 | C |
| **212** | a | c | b | a | a | NaN | 3 | NaN | 2 | 3 | 8 | C |
| **213** | a | NaN | NaN | a | b | NaN | NaN | NaN | NaN | 1 | 3 | C |
| **214** | NaN | b | c | NaN | NaN | NaN | 2 | NaN | NaN | 3 | 4 | C |

215 rows × 12 columns

Entrée [20]:
```
# On remplace les a/A par des 1
dataset = dataset.replace("a", 1)
dataset= dataset.replace("A", 1)

# On remplace les b/B par des 2
dataset = dataset.replace("b", 2)
dataset = dataset.replace("B", 2)

# On remplace les c/C par des 3
dataset = dataset.replace("c", 3)
dataset = dataset.replace("C", 3)

print(dataset)
```

```
       Q1   Q2   Q3   Q4   Q5   Q6   Q7   Q8   Q9 Q10  Score  Interpretation
0       1    1    1    1    1    1    1    1    1    1     10               2
1       2    2    2    2    2    2    2    2    2    2      0               3
2       3    3    3    3    3    3    3    3    3    3     20               1
3       1    2    3    1    2    1    2    3    1    2      8               3
4       2    3    1    3    1    3    2    3    1    2     11               2
..    ...  ...  ...  ...  ...  ...  ...  ...  ...  ..    ...             ...
210     3    3    3    3    3    2    3  NaN    3    3     14               2
211     2    1    3    1    2    2  NaN    2    3    2      5               3
212     1    3    2    1    1  NaN    3  NaN    2    3      8               3
213     1  NaN  NaN    1    2  NaN  NaN  NaN  NaN    1      3               3
214   NaN    2    3  NaN  NaN  NaN    2  NaN  NaN    3      4               3

[215 rows x 12 columns]
```

### 3.2.3) Remplacement des NaN par la fonction .mode()

Entrée [21]:
```
dataset.fillna(dataset.mode().iloc[0], inplace=True)
dataset
```

Out[21]:

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 | Score | Interpretation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 10 | 2 |
| **1** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 3 |
| **2** | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 20 | 1 |
| **3** | 1 | 2 | 3 | 1 | 2 | 1 | 2 | 3 | 1 | 2 | 8 | 3 |
| **4** | 2 | 3 | 1 | 3 | 1 | 3 | 2 | 3 | 1 | 2 | 11 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **210** | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 | 3 | 3 | 14 | 2 |
| **211** | 2 | 1 | 3 | 1 | 2 | 2 | 3 | 2 | 3 | 2 | 5 | 3 |
| **212** | 1 | 3 | 2 | 1 | 1 | 2 | 3 | 2 | 2 | 3 | 8 | 3 |
| **213** | 1 | 2 | 3 | 1 | 2 | 2 | 3 | 2 | 1 | 1 | 3 | 3 |
| **214** | 1 | 2 | 3 | 1 | 1 | 2 | 2 | 2 | 1 | 3 | 4 | 3 |

215 rows × 12 columns

Entrée [22]:
```
# # Conversion du df en int

df = dataset.astype("int")

print(df.dtypes)
```

```
Q1                int32
Q2                int32
Q3                int32
Q4                int32
Q5                int32
Q6                int32
Q7                int32
Q8                int32
Q9                int32
Q10               int32
Score             int32
Interpretation    int32
dtype: object
```

Entrée [24]:
```
# Suppression de la colonne "Score"

df = df.drop(['Score'], axis=1)
```

Entrée [25]:
```
df
```

Out[25]:

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 | Interpretation |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 |
| **1** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 |
| **2** | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 |
| **3** | 1 | 2 | 3 | 1 | 2 | 1 | 2 | 3 | 1 | 2 | 3 |
| **4** | 2 | 3 | 1 | 3 | 1 | 3 | 2 | 3 | 1 | 2 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **210** | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 | 3 | 3 | 2 |
| **211** | 2 | 1 | 3 | 1 | 2 | 2 | 3 | 2 | 3 | 2 | 3 |
| **212** | 1 | 3 | 2 | 1 | 1 | 2 | 3 | 2 | 2 | 3 | 3 |
| **213** | 1 | 2 | 3 | 1 | 2 | 2 | 3 | 2 | 1 | 1 | 3 |
| **214** | 1 | 2 | 3 | 1 | 1 | 2 | 2 | 2 | 1 | 3 | 3 |

215 rows × 11 columns

## **Partie 2** : Développement et entraînement d'un modèle KNN

### KNN From Scratch

- Préparez une fonction permettant de calculer les 3 différentes distances : Euclidean, Manhattan et Minkowski, (def distance(metric=' Euclidean', **kargs)).

```python
# créer une fonction rassemblant les différentes distances
from math import sqrt

# fonction reprenant les trois distances
def distance(metric, a, b, p=0):

    if metric == 'Euclidean':
        return sqrt(sum((e1-e2)**2 for e1, e2 in zip(a,b)))

    elif metric == 'Manhattan' :
        return sum(abs(e1-e2) for e1, e2 in zip(a,b))

    elif metric == 'Minkowski':
        return sum(abs(e1-e2)**p for e1, e2 in zip(a,b))**(1/p)


# data (de test pour voir si la fonction marche)
row1 = [0, 2]
row2 = [2, 0]

# calculer la distance
dist = distance('Manhattan', row1, row2)
print(dist)
```

Autre fonction :

```python
def distance(Data_1, Data_2, metric='euclidean', **kargs):

    for key,value in kargs.items():
        if key == 'p' :
            p = value
        else :
            p = 3

    if metric == 'euclidean' :
        Dis = np.sqrt(np.sum((Data_1-Data_2)**2))
    elif metric == 'manhattan' :
        Dis = np.abs(np.sum(Data_1-Data_2))
    elif metric == 'minkowski':
        Dis = (np.sum(np.abs(Data_1-Data_2))**3)**(1/3)
    return Dis

k = 3
metric = 'manhattan'
```

- Codez l'algorithme de KNN sous forme une fonction (def KNN(Data_Test, Data_Train, Label_Train, k=1, **kargs))

L'algorithme des K plus proches voisins (K-nearest neighbors) ou kNN, est un algorithme appartenant à la famille du Machine Learning. Il fait partie des classes d'algorithmes d'apprentissage supervisé, capable de résoudre des problèmes de classification et de régression.

```
def knn(X_test, X_train, y_train, k, metric) :


    Resultat = []
    for j in range(0,len(X_test)):

        D_T = X_test.iloc[j,:]

        Distance = []
        for i in range(0,len(X_train)):
            D_A = X_train.iloc[i,:]

            #Dis = np.sqrt(np.sum((D_T-D_A)**2))

            Dis = distance(D_T, D_A, metric=metric)
            Distance.append(Dis)

        SS = np.sort(Distance)
        S = np.argsort(Distance)

        index_petite_distance = S[:k]
        Pred = y_train.iloc[index_petite_distance]

        Pred = [np.sum((Pred == 1).astype(int)), np.sum((Pred == 2).astype(int)), np.sum((Pred == 3).astype(int))]

        Pred = np.argmax(Pred)



        Resultat.append(Pred + 1)


    return Resultat
```

Algorithme des K-Folds :

```
# K-fold

from sklearn.model_selection import KFold
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.metrics import accuracy_score

kf = KFold(n_splits=3)
kf.get_n_splits(X)

for train_index, test_index in kf.split(X):
    X_train = X.iloc[train_index,:]
    y_train = y.iloc[train_index]

    X_test = X.iloc[test_index,:]
    y_test = y.iloc[test_index]

    model = KNN(n_neighbors = 3)
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    P = accuracy_score(y_test, y_pred)

    print("Performance:", P)
```

## KNN avec SKLEARN :

## KNN avec SKlearn

La bibliothèque Sklearn propose un panel des techniques de classification, y compris le KNN.

Dans cette étape, vous êtes orientés vers la classe « sklearn.neighbors » pour maitriser les paramètres et les options possibles. Vous êtes censés à préparer un modèle performant pour notre application tout en respectant les consignes de la conception d'un modèle IA (Data préparée, K-fold validation, hyperparamètre, Gridsearch). (N'oubliez pas de présenter une comparaison entre KNN From Scratch et KNN Sklearn dans le compte rendu).

Entrée [81]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Séparer les données en jeu de train et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Normalisation des données
#scaler = StandardScaler()
#X_train = scaler.fit_transform(X_train)
#X_test = scaler.transform(X_test)
```

Entrée [*]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

Entrée [85]:
```
y_pred = classifier.predict(X_test)
y_pred
```

On affiche la précision obtenue

Entrée [57]:
```
from sklearn import metrics
print('Training set accuracy: ', metrics.accuracy_score(y_train, y_pred_train))
print('Test set accuracy: ',metrics.accuracy_score(y_test, y_pred_test))
print('\nMatrice de confusion : \n', confusion_matrix(y_test, y_pred))
```

```
Training set accuracy:  0.8662790697674418
Test set accuracy:  0.6976744186046512

Matrice de confusion :
 [[ 1  2  0]
 [ 0  4 12]
 [ 0  5 19]]
```

## Grid Search:

Phase de calcul

Entrée [*]:
```
knn_best = KNeighborsClassifier(n_neighbors = 4,metric = 'manhattan')
knn_best.fit(X_train, y_train)
```

Entrée [115]:
```
y_pred_train = knn_best.predict(X_train)
y_pred_test = knn_best.predict(X_test)
```

On affiche la précision obtenue

Entrée [116]:
```
from sklearn import metrics
print('Training set accuracy: ', metrics.accuracy_score(y_train, y_pred_train))
print('Test set accuracy: ',metrics.accuracy_score(y_test, y_pred_test))
print('\nMatrice de confusion : \n', confusion_matrix(y_test, y_pred))
```

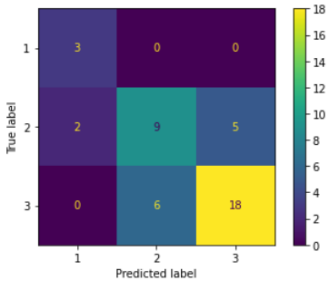
```
Training set accuracy:  0.8662790697674418
Test set accuracy:  0.6976744186046512

Matrice de confusion :
 [[ 3  0  0]
 [ 1  6  9]
 [ 0  6 18]]
```

### Matrice de confusion

Entrée [117]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(knn_best, X_test, y_test)
```

Out[117]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18d8a7c8160>




**Partie 3** : Mettre en place la solution dans l'application de test de personnalité

Exportation du Modèle:

### Joblib

Exporter le modèle

Entrée [96]:
```
import joblib

filename = 'KNN_Final'

joblib.dump(classifier, filename)

nom_du_modele = joblib.load(filename)
```

Modification du script "Test"

```
#  je transforme en DataFrame
Data = pd.DataFrame(d, index=[1])

#  je récupère mes 10 premier itérations
for_test=Data.replace(["a","A","1","1.0"],1).replace(["b","B","2","2.0"],2).replace(["c","C","3","3.0"],3)
#  je découpe ma dataframe
for_test = for_test.iloc[0,0:10]
#  transpose le
for_test = np.expand_dims(for_test, axis = 0)
load_model = joblib.load('KNN_Final')
pred = load_model.predict(for_test)

# je récupère la valeur dans ma 'liste'
print('Prédiction KNN : ', pred[0])
```

Interprétation finale  avec et sans SKlearn:

Votre Score est :  13

---

--------- Interprétation final avec score  ---------

---
Score entre 11 et 14 :

Même s’il vous arrive d’être tendu(e) et stressé(e) en certaines occasions, vous semblez donc capable de prendre soin de vous-même et de dire non aux requêtes déraisonnables.
---

---

--------- Interprétation final avec KNN  ---------

---
Prédiction KNN :  3