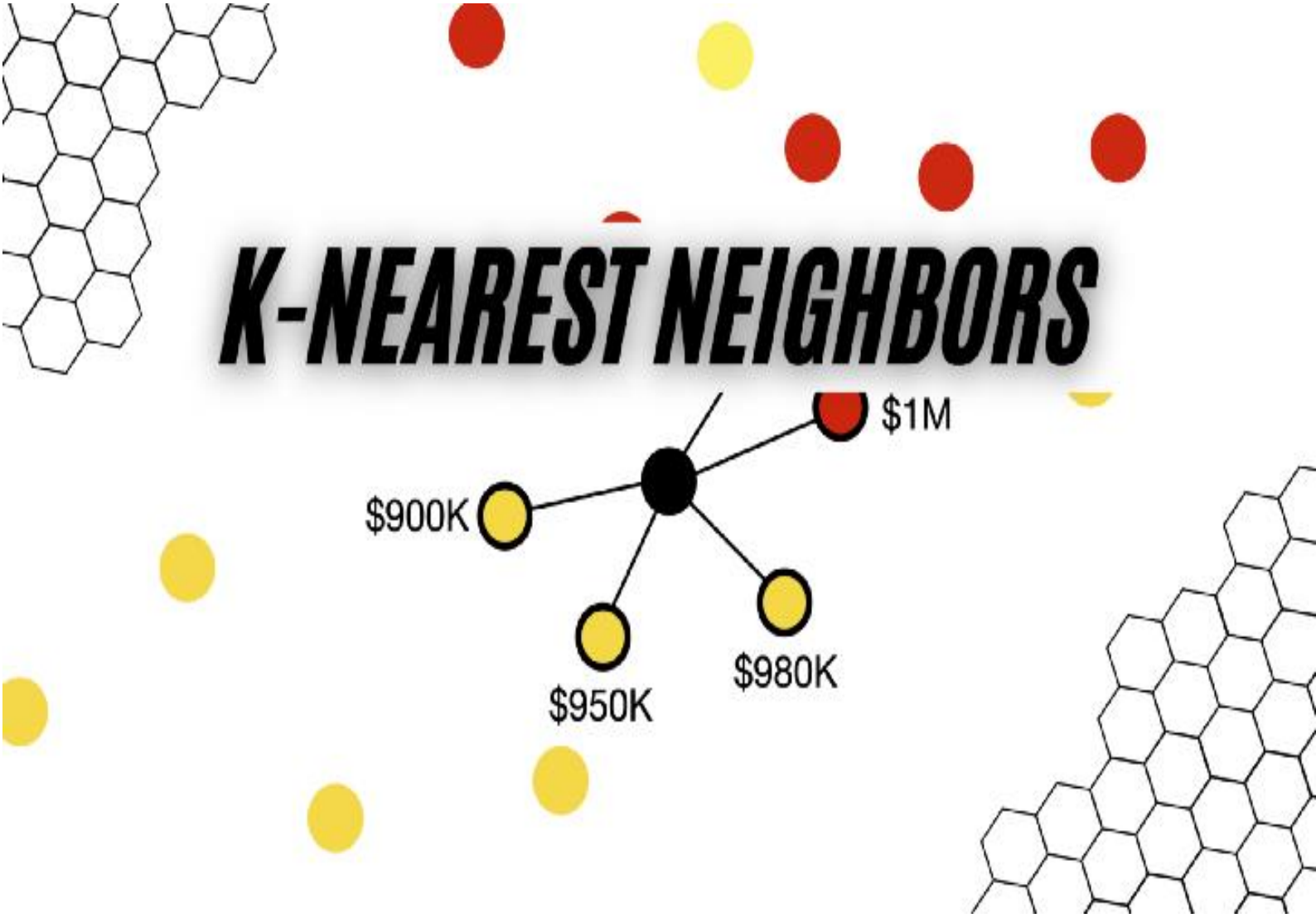


<u>Présenté et soutenu par Groupe III</u> : Souleymane Kodjo

Ouzairou Djiré

Abdrahame Idrissa Doumbia

<u>Responsable pédagogique</u> :Dr A. SIDIBE

UTILISATION DE L'ALGORITHME KNN

SOMMAIRE

1 Logiciel utilisée 1

2 Importation des bibliotheques requis 3

3 Importation des données 3

4 Mapping 3

5 Première visualisation des données 4

6 Echantillon non etiquetée 5

7 Visualisation de Echantillon a prédire dans le graphe 5

8 Séparation du jeu de donnée en training set et testing set 6

9 Algorithmique KNN pour K=2 6

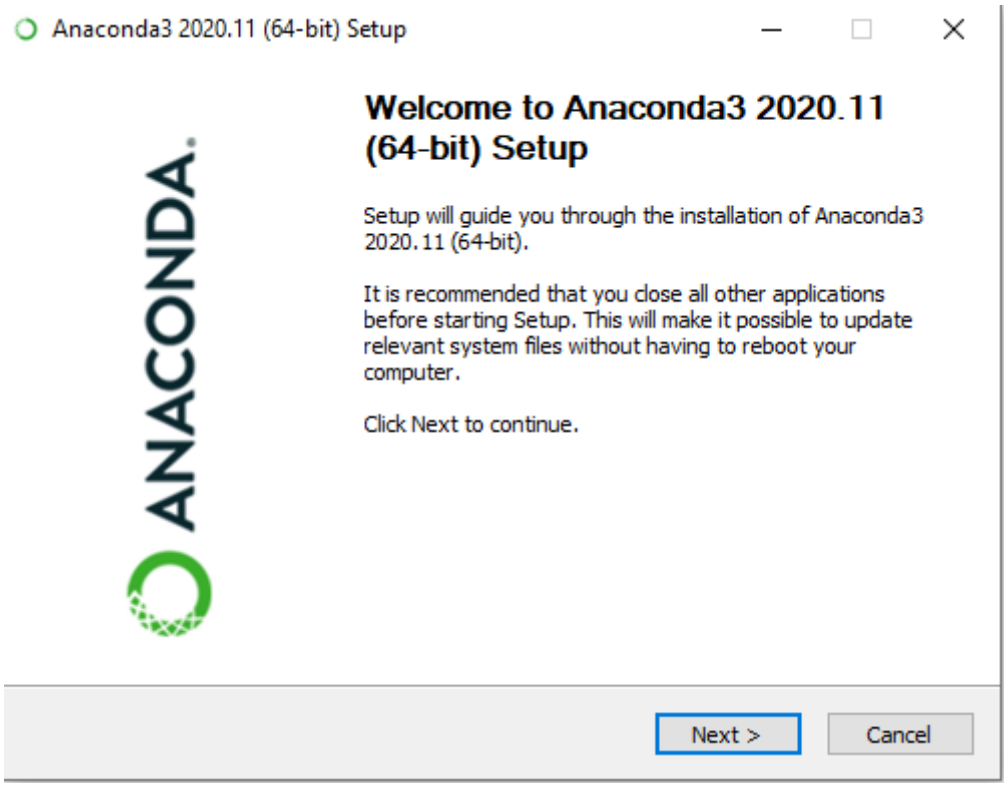
10 Optimisation du taux d'erreurs 7

11 Création d'une fonction python résultat qui retourne la prédiction 7

12 Prédiction finale 8

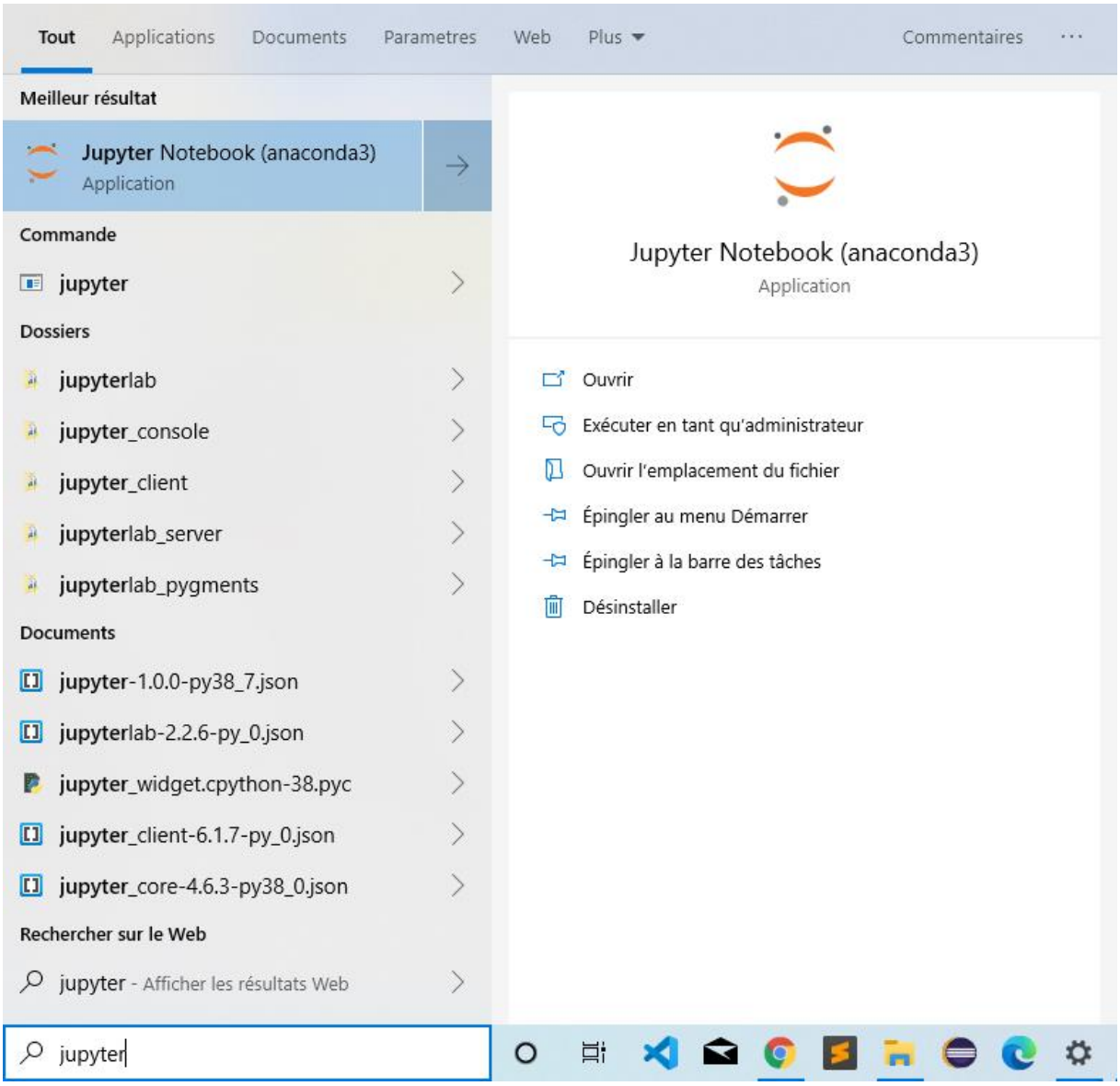
1 Logiciel utilisée

Installer Python Anaconda : le meilleur outil de Machine Learning



Anaconda contient tous les outils et librairies dont vous avez besoin pour faire du Machine Learning : Numpy, Matplotlib, Sklearn, etc.
Une fois Anaconda installé, On a va pouvoir lancer l'application Jupyter Notebook depuis votre barre de recherche Windows

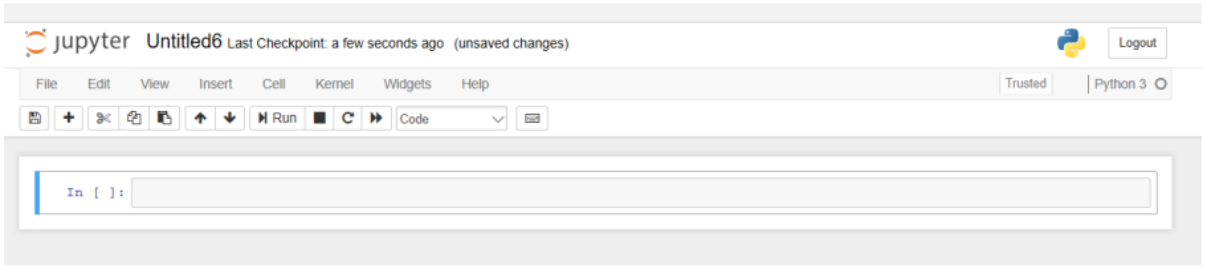
UTILISATION DE L'ALGORITHME KNN



Jupyter Notebook : est une application Web qui permet de créer et de partager des codes Python. Il n'est pas nécessaire d'avoir une connexion Internet pour vous servir de Jupyter, les données sont stockée localement

La fenêtre principale de Jupyter n'est ni plus ni moins qu'un explorateur de fichier relié à votre disque dur, vous pouvez ouvrir tous vos fichiers, Dataset, codes, etc. depuis cette fenêtre. Cliquez sur le bouton 'New' situé en haut à droite de cette fenêtre pour commencer à écrire un nouveau programme (puis cliquez sur Python 3

UTILISATION DE L'ALGORITHME KNN



2 Importation des bibliothèques requis

Ici on va importer les bibliothèques nécessaire grave au mot clé **import**

Importation des bibliothèques requis

Entrée [1]:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.neighbors import KNeighborsClassifier
```

3 Importation des données

La fonction $read\_csv()$ de librairie pandas permet d'importer le dataset « iris_min »

Importation des données

Entrée [3]:

```
1 df = pd.read_csv(r"C:\Users\Kodjo\data_science\iris_min.csv")
```

Entrée [ ]:

```
1
```

4 Mapping

Pour faciliter l'interprétation Ici on va remplacer

Iris-setosa par 0

Iris-virginica par 1

Iris-versicolor par 2

Formatage

Entrée [4]:

```
1 df['class'] = df['class'].replace("Iris-setosa", 0)
2 df['class'] = df['class'].replace("Iris-virginica", 1)
3 df['class'] = df['class'].replace("Iris-versicolor", 2)
```

Entrée [ ]:

```
1
```

5 Première visualisation des données

Une fois le fichier csv modifié, on définit ensuite deux variables x et y représentant respectivement les valeurs de nos *Feature* (les facteurs). Puis les données sont visualisées sous forme de graphique (abscisse : "petal_length", ordonnée : "petal_width")

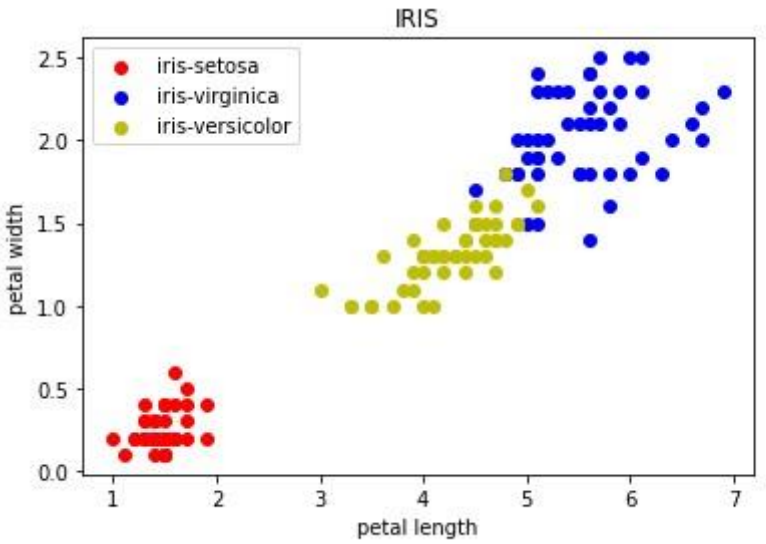
Visualisation en nuage de point

Entrée [ ]:
```
x=df['petal_length']
y=df['petal_width']
label=df['class']
plt.scatter(x[label==0], y[label==0], c='r', label='iris-setosa')
plt.scatter(x[label==1], y[label==1], c='b', label='iris-virginica')
plt.scatter(x[label==2], y[label==2], c='y', label='iris-versicolor')
plt.xlabel("petal length")
plt.ylabel("petal width")
plt.title("IRIS")
plt.legend()
plt.show()

```

Entrée [ ]:
```
1
```

- Le nuage de point



6 Echantillon non étiquetée

Ici on va définir un échantillon a prédire par sa longueur et largeur de pétale

```
# Echantillion predire
```

Entrée [6]:
```
#Les attribut de echantillion
```

Entrée [7]:
```
#Valeur
longueur = 2.5
largeur = 0.75
k=3
```

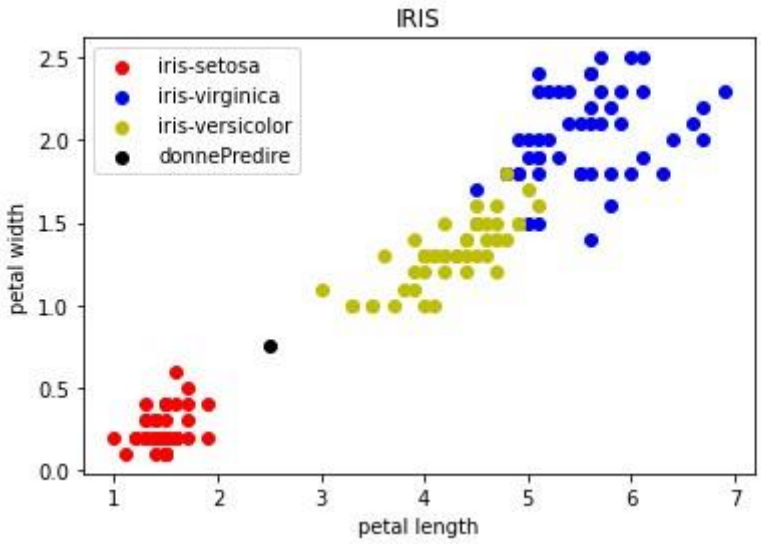
7 Visualisation de Echantillon a prédire dans le graphe

Affichage de l'ensemble de données (dataset+ échantillon)

```
# Visualisation
```

Entrée [8]:
```
plt.scatter(x[label==0], y[label==0], c='r', label='iris-setosa')
plt.scatter(x[label==1], y[label==1], c='b', label='iris-virginica')
plt.scatter(x[label==2], y[label==2], c='y', label='iris-versicolor')
#On a joute au graphe
plt.scatter(longueur, largeur, color='k')
plt.xlabel("petal length")
plt.ylabel("petal width")
plt.title("IRIS")
plt.legend()
plt.show()
```

- Le nuage de point



UTILISATION DE L'ALGORITHME KNN

8 Séparation du jeu de donnée en training set et testing set

On utilise la fonction ***train_test_split***() pour fractionner l'ensemble des données en données de train et de test qui sont 80% et 20%

X Contient les $Feature$ et $label$ nos $Target$

X_train et X_test Représentent respectivement 80% et 20% de X

y_train et y_test Représentent respectivement 80% et 20% de $label$

Separation du jeux de donnée en training set et testing set

Entrée [25]:
```
#feature
feature_columns = ['petal_length', 'petal_width']
X = df[feature_columns].values
```

Entrée [26]:
```
#target
label = df['class']
```

Entrée [27]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,label, train_size=0.8, random_state=0)
```

Entrée [ ]:
```

```

9 Algorithmique KNN pour K=2

Maintenant que les données sont fractionnées en train-set , nous allons créer un classifieur a l'aide de la méthode ***KNeighborsClassifier***() qui prend en paramètre $n\_neighbors = valeur$ et on définit par la suite le paramètre K pour valeur 2 et le taux d'erreur de prédiction.

On entraine le model a l'aide de la méthode ***filt***() qu'on lui passe en paramètres x_train et y_train

La méthode ***score***() calcule la performance du model

Algorithme KNN

Entrée [28]:
```
#algo KNN
k=2
#Creation de model de classification
model = KNeighborsClassifier(n_neighbors=k)
#(data, target)
model.fit(X_train,y_train)
model.score(X_test, y_test)
```

Out[28]: 0.9333333333333333

UTILISATION DE L'ALGORITHME KNN

10 Optimisation du taux d'erreurs

On va faire varier la valeur de k pour une meilleure performance du model

Optimisation du score sur les données test

Entrée [30]:
```
errors = {}
valeur_k = list(range(2,15))
```

Entrée [31]:
```
for k in valeur_k:
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    score = 100*(1-model.score(X_test, y_test))
    error = {k:score}
    errors.update(error)
```

Entrée [32]:
```
errors
```

Out[32]:
```
{2: 6.666666666666665,
 3: 3.3333333333333326,
 4: 3.3333333333333326,
 5: 3.3333333333333326,
 6: 3.3333333333333326,
 7: 3.3333333333333326,
 8: 3.3333333333333326,
 9: 3.3333333333333326,
 10: 3.3333333333333326,
 11: 3.3333333333333326,
 12: 3.3333333333333326,
 13: 3.3333333333333326,
 14: 3.3333333333333326}
```

11 Création d'une fonction python résultat qui retourne la prédiction

C'est la classe dans lequel échantillon a appartient.

Entrée [ ]:
```
#Fonction resltat
def resultat():
    if prediction == 0 :
        return "Resultat : setosa"
    if prediction == 1 :
        return "Resultat : virginica"
    if prediction == 2 :
        return "Resultat : verginicolor"
```

12 Prédiction finale

Entrée [ ]:
```
#Donnée d'entr"
longueur = 2.5
largeur = 0.75
```

Entrée [36]:
```
#L'algorithmique est deja importée
d = list(zip(x,y))
model = KNeighborsClassifier(n_neighbors=3)
model.fit(d, label)
prediction = model.predict([[longueur, largeur]])
```

Entrée [40]:
```
#Fonction resltat
def resultat():
    if prediction == 0 :
        return "Resultat : setosa"
    if prediction == 1 :
        return "Resultat : virginica"
    if prediction == 2 :
        return "Resultat : verginicolor"
resultat()
```

Out[40]: 'Resultat : setosa'