**Documentation technique – Dataset Flowers**



Louis ADAM / Yusuf FIDAN

Table des matières

[1) Généralités 3](#_Toc57551515)

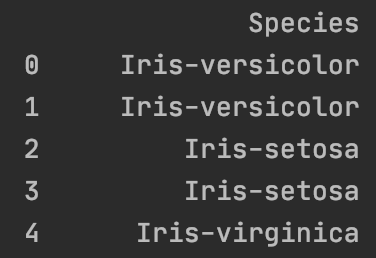
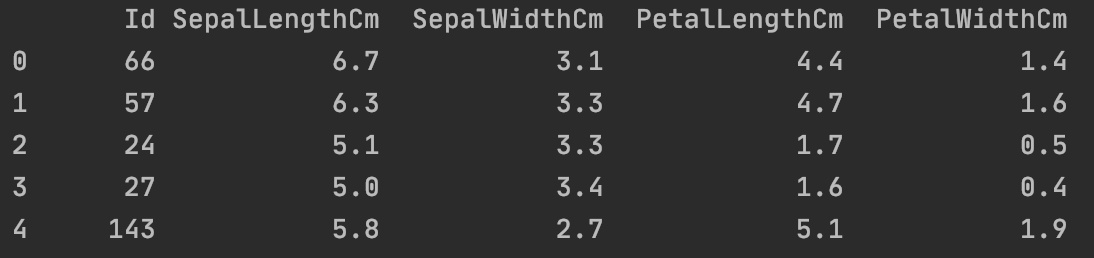
[2) Nettoyage des données 4](#_Toc57551516)

[3) Algorithme de prédiction 6](#_Toc57551517)

[4) Neo4j 7](#_Toc57551518)

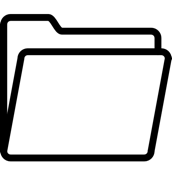
[5) Docker 8](#_Toc57551519)

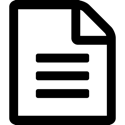
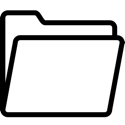
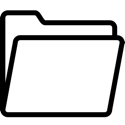
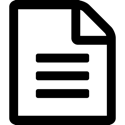
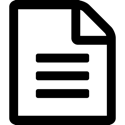
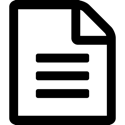
Aperçu du dataset :



# Généralités

Structure du code :





fct.py

algopred.py

Dataset\_2\_flower.csv

functions

main.py

Pycharm

Repo\_G1\_Dataset\_2\_flower

# Nettoyage des données

Une image contenant texte

Description générée automatiquementAvant de nettoyer les données, il faut exporter le csv en dataframe, à l’aide de la fonction dfcsv :

Dans ce dataset, les colonnes à exploiter sont **Id, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm et Species**.

Il faut donc retirer les colonnes non exploitables : **index**, **Unnamed: 0**, **Unnamed: 0.1**, **Unnamed: 0.1.1, level\_0,** **Unnamed: 0.1.1.1**

Ces colonnes sont retirées à l’aide de la fonction dropColumns :

Une image contenant texte

Description générée automatiquement

Une fois ces colonnes retirées, il faut analyser les colonnes non propres. On peut les analyser avec la fonction counts. Elle permet de voir les valeurs de chaque colonne afin de repérer les données non exploitables :

Une image contenant texte

Description générée automatiquement

Une fois les colonnes analysées, on peut traites les colonnes non propres. Dans ce dataset, seule la colonne SepalLengthCm est inexploitable. Pour y remédier, on va exécuter la fonction cleanCol :

Une image contenant texte

Description générée automatiquement

Cette fonction permet de remplacer les valeurs erronées par la moyenne des valeurs exploitables de la colonne.

Pour la colonne Id, la fonction cleanId permet de faire en sorte que chaque id de fleur soit unique :

Une image contenant texte

Description générée automatiquement

Le dataset est donc bien nettoyé. Pour faire une dernière vérification la fonctions isNan permet de générer un dataset contenant les lignes où un nan est présent :

Une image contenant texte

Description générée automatiquement

# Algorithme de prédiction

Les algorithmes de prédiction utilisés dans ce projet permettent d’avoir un indice de fiabilité. 2 algorithmes de prédiction sont utilisés dans le code source : le Random Forest et la Régression Logistique. Ces 2 algorithmes sont définis dans la fonction algodf :

Une image contenant texte

Description générée automatiquement

Le choix de l’algorithme se fait dans l’appel de la fonction (« rf » pour le Random Forest et « lg » pour la Régression Logistique).

Pour le Random Forest, le nombre d’arbre est défini via un GridSearch, qui permet de détecter le paramètre le plus efficace pour avoir le meilleur résultat possible :



Le résultat affiché correspond à l’indice de fiabilité de l’algorithme : plus il se rapproche de 1, plus l’algorithme est fiable.

# Neo4j

Neo4j est un système de gestion de base de données NOSQL basé sur les graphes.

Neo4j est utilisé dans ce projet afin d’avoir une représentation relationnelle entre les différents champs du dataset.

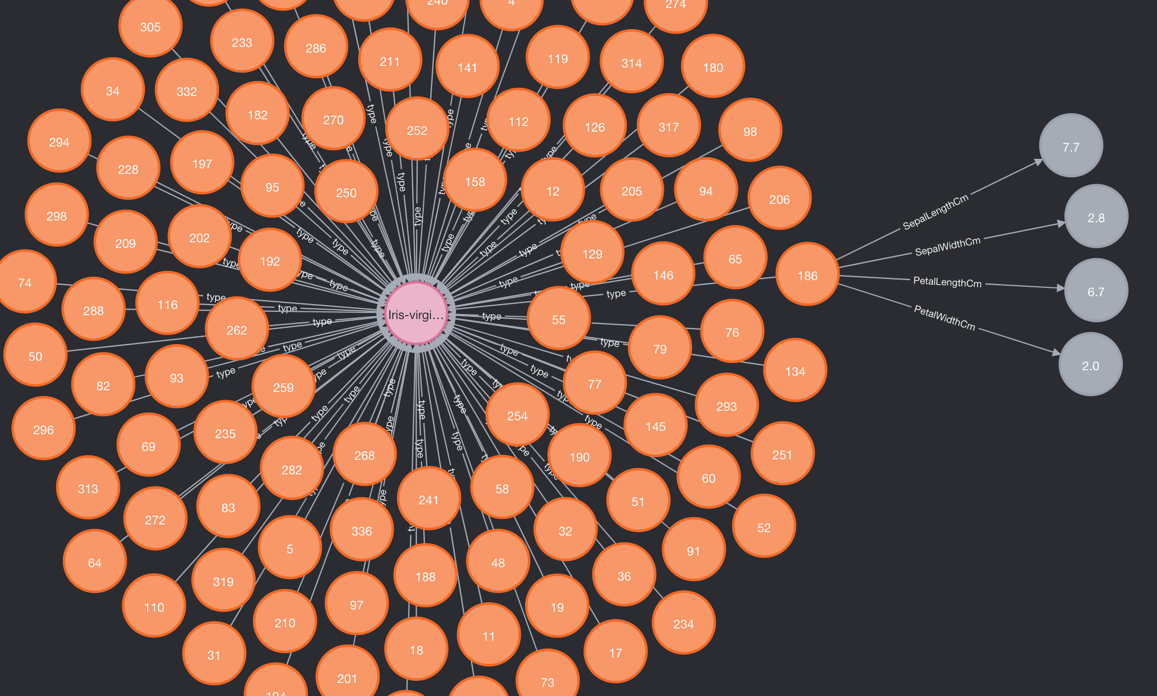
Pour ce faire, l’utilisation du module py2neo est requis.

Voici un aperçu du code permettant de faire les relations entre les Id des fleurs, leur type et leur taille :

Une image contenant texte

Description générée automatiquement

Voici le résultat :



# Docker

Une fois le code terminé, il faut mettre tout cela sur un container Docker. Le but est donc de récupérer l’image Neo4j officielle et d’exécuter les commandes nécessaires à l’exécution du script Python via un Dockerfile.

Voici le contenu du Dockerfile :

FROM neo4j

RUN bash

RUN apt update -y

RUN apt install python3 -y

RUN apt install python3-pip -y

RUN apt install vim -y

RUN pip3 install pandas

RUN pip3 install sklearn

RUN pip3 install py2neo

RUN pip3 install --upgrade py2neo

RUN mkdir -p /usr/src/app

COPY . /usr/src/app

Pour résumer, on utilise l’image officielle neo4j, et on exécute les commandes permettant d’installer python, pip, ainsi que les modules pandas, sklearn, py2neo via pip3.

On crée ensuite un dossier pour y accueillir le code source via la commande COPY.

Une fois le Dockerfile créé, on exécute la commande dans le dossier ou se trouve le Dockerfile :

docker build . -t « neo4jmage »

L’image se créée, ensuite il faut créer le conteneur depuis l’image précédemment créée :

docker run \

--name containerneo4j \

-p7474:7474 -p7687:7687 \

-d \

-v $HOME/neo4j/data:/data \

-v $HOME/neo4j/logs:/logs \

-v $HOME/neo4j/import:/var/lib/neo4j/import \

-v $HOME/neo4j/plugins:/plugins \

--env NEO4J\_AUTH=none \

neo4jmage:latest

Pour exécuter le code source, on va utiliser la commande docker exec :

docker exec -it containerneo4j python3 /usr/src/app/pycharm/main.py

Le conteneur est donc prêt et le script est exécuté.