Atelier Pratique : Random Forest avec Scikit-learn

Ce notebook présente une introduction pratique à l'algorithme Random Forest en utilisant Scikit-learn. Appliqué pour un exemple de Classification (avec la base de données Iris)

0 : Installation des packages nécessaire pour l'atelier

In [ ]:
```
#!pip install pandas numpy scikit-learn matplotlib seaborn joblib
```

1 : Importation de la base de donnée Iris

In [ ]:
```
from sklearn.datasets import load_iris
import pandas as pd

# chargement de la base de données iris
# cette une base de données classique pour la classification
iris = load_iris()

# conversion de la base de données iris en DataFrame pandas
df_iris = pd.DataFrame(iris.data, columns=iris.feature_names)
# ajout de la colonne cible (target) à la DataFrame
# la colonne cible contient les étiquettes de classe pour chaque échantillon
# Les étiquettes de classe sont les espèces d'iris (Iris Setosa, Iris Versicolor, Iris Virginica)
# Les étiquettes de classe sont des entiers (0, 1, 2) représentant les trois espèces d'iris
# 0: Iris Setosa, 1: Iris Versicolor, 2: Iris Virginica
df_iris['target'] = iris.target

df_iris.head(10)
# iris contient 150 échantillons, chacun avec 4 caractéristiques (longueur et largeur des sépales et pétales)
```

Out[ ]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
5	5.4	3.9	1.7	0.4	0
6	4.6	3.4	1.4	0.3	0
7	5.0	3.4	1.5	0.2	0
8	4.4	2.9	1.4	0.2	0
9	4.9	3.1	1.5	0.1	0

2. Entrainement et évaluation de modèle RF

In [3]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Split les données en ensembles d'entraînement et de test
# La fonction train_test_split divise le jeu de données en ensembles d'entraînement et de test
# Le paramètre test_size spécifie la proportion du jeu de données à inclure dans la division de test
# Le paramètre random_state est utilisé pour initialiser le générateur de nombres aléatoires pour la reproducti
# L'ensemble d'entraînement sera utilisé pour entraîner le modèle, et l'ensemble de test sera utilisé pour éval
# Les caractéristiques (X) sont les quatre premières colonnes de la DataFrame, et la variable cible (y) est la

X = df_iris.drop('target', axis=1)
y = df_iris['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Entraînement du modèle
# Le modèle RandomForestClassifier est un classificateur basé sur des arbres de décision
# Le paramètre n_estimators spécifie le nombre d'arbres dans la forêt

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# évaluation du modèle
# Le modèle est évalué sur l'ensemble de test
```

```
y_pred = model.predict(X_test)

# La matrice de confusion est une table qui est souvent utilisée pour décrire la performance d'un modèle
# La matrice de confusion compare les étiquettes prédites par le modèle avec les étiquettes réelles
# classification_report fournit des informations détaillées sur la précision, le recall et le score F1
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

In [5]:
```
# Predict the species of a new iris flower
# The predict method is used to make predictions on new data
# The input data should be in the same format as the training data
# The input data should be a 2D array or DataFrame with the same number of features as the training data
# For example, to predict the species of an iris flower with the following features:
# Sepal length: 5.1 cm      Sepal width: 3.5 cm      Petal length: 1.4 cm        Petal width: 0.2 cm

# The input data should be a DataFrame with one row and four columns
# The predict method will return the predicted species (0, 1, or 2) for the input data
print(model.predict(pd.DataFrame([5.1,3.5,1.4,0.2]).T))
# The predict_proba method returns the probability estimates for all classes
# The predict_proba method returns a 2D array with the same number of rows as the input data and the same numbe
# The predict_proba method returns the probability estimates for each class
# For example, to get the probability estimates for the same iris flower as above:
print(model.predict_proba(pd.DataFrame([5.1,3.5,1.4,0.2]).T))
```

```
[0]
[[1. 0. 0.]]
```

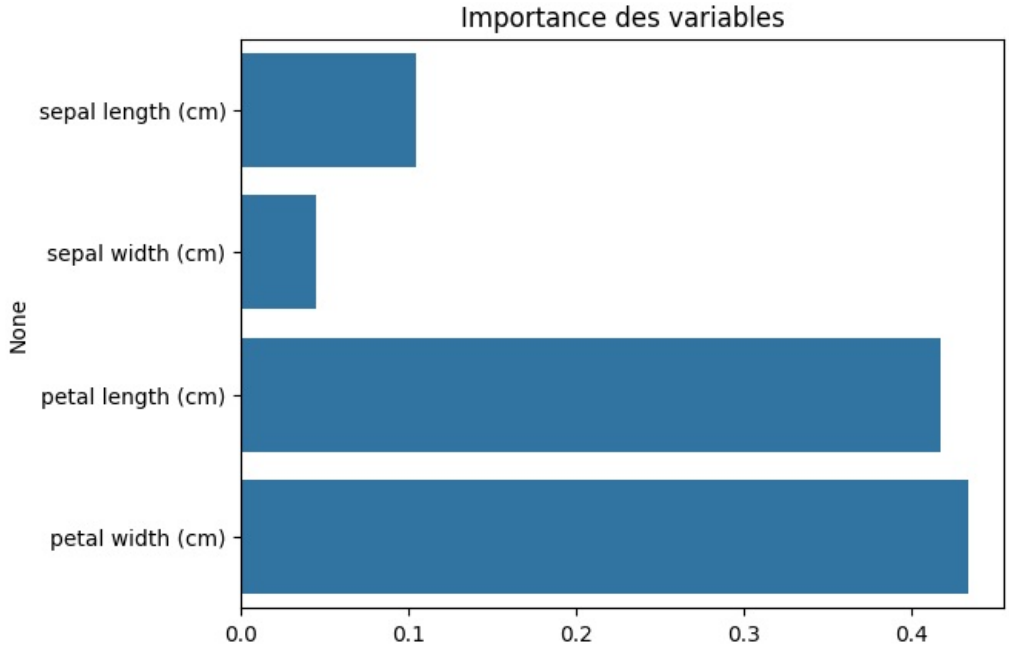
3. Manipulation de l'importance des variables

In [6]:
```
# The feature_importances_ attribute returns the importance of each feature in the model
# The feature importances are a measure of how much each feature contributes to the model's predictions
# The feature importances can be used to identify the most important features in the dataset

importances = model.feature_importances_
# The feature importances are stored in a numpy array
# The feature importances are in the same order as the features in the training data
features = X.columns
for feature, importance in zip(features, importances):
    print(f"{feature}: {importance:.4f}")


import seaborn as sns
import matplotlib.pyplot as plt
# Visualize the feature importances
# The barplot function creates a bar plot of the feature importances
sns.barplot(x=importances, y=features)
plt.title("Importance des variables")
plt.show()
```

```
sepal length (cm): 0.1041
sepal width (cm): 0.0446
petal length (cm): 0.4173
petal width (cm): 0.4340
```



4. Sauvgarde du modèle entrainé pour l'utiliser dans une App

In [7]:
```
import joblib
joblib.dump(model, 'random_forest_model.pkl')
```

Out[7]:
```
['random_forest_model.pkl']
```

In [8]:
```
#pour le chargement du modèle
model = joblib.load('random_forest_model.pkl')
# le modèle peut être utilisé pour faire des prédictions sur de nouvelles données
model.predict(X_test)
```

Out[8]:
```
array([1, 0, 2, 1, 1, 0, 1, 2, 1, 1, 2, 0, 0, 0, 0, 1, 2, 1, 1, 2, 0, 2,
       0, 2, 2, 2, 2, 2, 0, 0, 0, 0, 1, 0, 0, 2, 1, 0, 0, 0, 2, 1, 1, 0,
       0])
```