



# Arbres de Décision avec SciKit Learn

Fichiers sur

<https://github.com/mkirschpin/CoursPython>

# Arbres de Décision

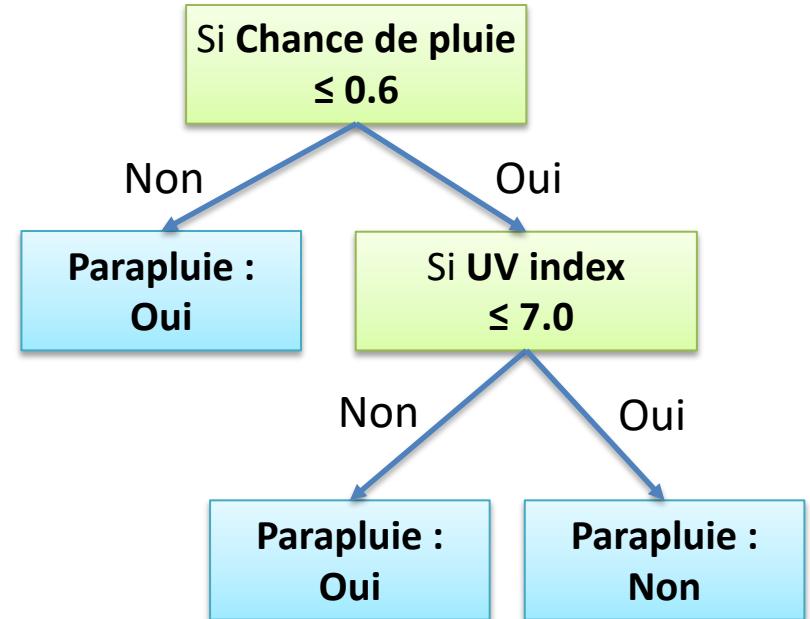
- **Bases**

- Méthode de **classification** simple
- Méthode **supervisée** (données étiquetées)
- Comparable à un enchainement de « **questions** » (**conditions**)  
**if – else**

**Features (variables)**      **Target (classes)**  
 « X set »                          « Y set »

		Parapluie
Chance de pluie	UV Index	Parapluie
0.1	11	True
0.9	1	True
0.3	3	False
0.1	2	False

*Dois-je prendre un parapluie ?*



- La méthode cherche **l'ordre** des « questions » de manière à **minimiser la taille de l'arbre**

# Arbres de Décision en Python

<https://scikit-learn.org>



- **Bibliothèque SciKit Learn**

- Principale bibliothèque de **Machine Learning** en Python
- Nombreuses **méthodes supervisées** et **non-supervisées**
  - Arbres de décision, régression linéaire, K-means...

```
from sklearn.tree import DecisionTreeClassifier
```

- Nombreux **outils** d'aide
  - Exemples de datasets, prétraitement et manipulation des datasets, métriques d'évaluation...

```
from sklearn import datasets
from sklearn.model_selection import ...
from sklearn.metrics import ...
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Arbres de Décision en Python

```
from sklearn.tree import DecisionTreeClassifier
```

```
clf = DecisionTreeClassifier()
```

L'opération « **fit** » réalise l'*entraînement*  
du modèle (« **training** » ou « **fitting** »)

```
clf.fit( x_train[feature_names], y_train[target] )
```

Données d'entraînement  
*(training features)*

Classes (**target**) pour les  
données de training

```
y_pred = clf.predict(x_test[feature_names])
```

Données de test  
*(testing features)*

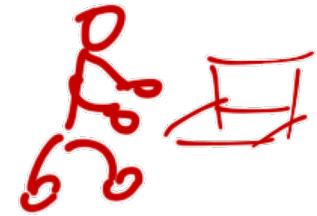
Méthode de classification

Création de l'*objet* qui contiendra  
notre **arbre de décision**.  
C'est lui qu'on va manipuler pour  
créer l'arbre et l'utiliser.

Pour l'entraînement,  
l'opération « **fit** » a  
besoin de deux  
**DataFrames** ...

Une fois entraîné, le modèle  
peut être **testé**, puis utilisé  
pour la **classification**...

# Hands On !



## • Premier exemple

- Créer un nouveau Notebook Jupyter...
- Créer le **DataFrame de *training***

On n'oublie pas  
d'importer la  
bibliothèque Pandas

```
dfUmbrella = pnd.DataFrame (  
    { 'Chance Rain': [0.1, 0.9, 0.3, 0.1, 0.8],  
     'UV Index': [11, 1, 3, 2, 2] ,  
     'Umbrella' : [True, True, False, False, True] } )
```

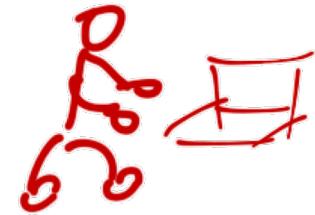
- Identifier dans deux variables les ***features*** et le ***target***

```
feature_names = ['Chance Rain', 'UV Index']  
target = 'Umbrella'
```

- Crée le **DataFrame de test**

```
x_test = pnd.DataFrame (  
    { 'Chance Rain': [0.5, 0.2],  
     'UV Index': [5, 8] } )
```

# Hands On !



- **Premier exemple**

- Crée son objet **DecisionTreeClassifier**

```
clf = DecisionTreeClassifier()
```

On n'oublie pas la bibliothèque `sklearn.tree`

- Entrainer son modèle

```
clf.fit ( dfUmbrella[feature_names] , dfUmbrella[target] )
```

- Tester son modèle

```
y_pred = clf.predict(x_test)  
print (y_pred)
```

[False True]

# Arbres de Décision en Python

- Comment visualiser son modèle ?

- L'arbre de décision créé peut être visualisé

Mode texte

`export_text`

```
from sklearn.tree import export_text

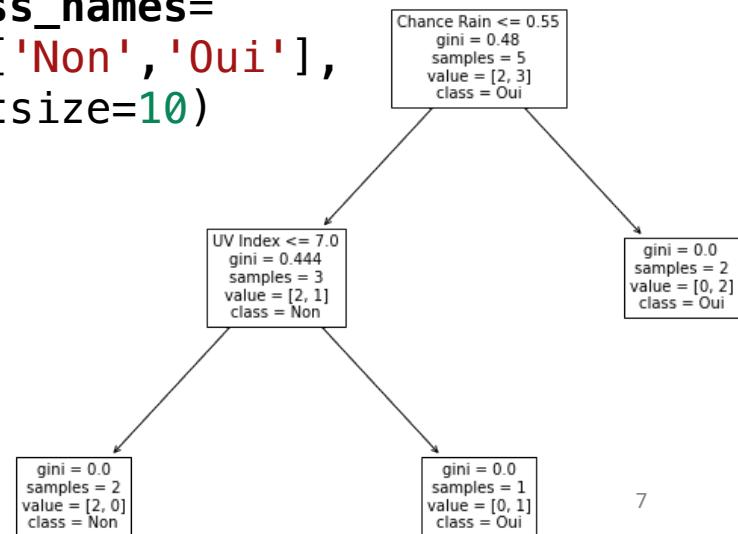
texte = export_text(clf,
                    feature_names=feature_names,
                    spacing=3, decimals=2)

print ( texte )
      |--- Chance Rain <= 0.55
      |   |--- UV Index <= 7.00
      |   |   |--- class: False
      |   |   |--- UV Index >  7.00
      |   |   |   |--- class: True
      |--- Chance Rain >  0.55
      |   |--- class: True
```

Mode graphique (MatPlot)  
`plot_tree`

```
from sklearn.tree import plot_tree

plot_tree(clf,
          feature_names=feature_names,
          class_names=
          ['Non', 'Oui'],
          fontsize=10)
```



# Arbres de Décision en Python

- Quelle est l'importance de chaque *feature* ?
  - Certains *features* peuvent contribuer plus à la décision que d'autres
  - On peut connaître le niveau d'importance des *features* dans un l'arbre entraîné
    - Attribut **feature\_importances\_**

```
pnd.DataFrame (  
    { 'feature_names' : feature_names ,  
      'importance' : clf.feature_importances_  
    })
```

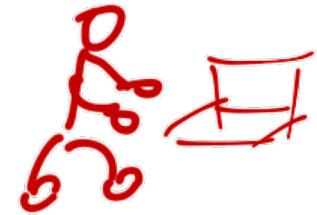
Création d'un DataFrame avec les *features* et leur *importance*  
(juste pour visualiser plus facilement)

Chaque *feature* va avoir un niveau d'importance entre 0 et 1.

	feature_names	importance
0	Chance Rain	0.444444
1	UV Index	0.555556

L'importance indique le niveau d'influence d'une variable dans la décision

# Hands On !



## • Exercice : visualisation d'arbre de décision

- Utiliser **export\_text** pour visualiser l'arbre de décision réalisé
- Utiliser **plot\_tree** pour produire une image de l'arbre
- Afficher l'importance des **features** utilisées

On n'oublie pas la bibliothèque `sklearn.tree`

```
texte = export_text ( clf, feature_names=feature_names,
                      spacing=3, decimals=2)
print (texte)
```

Suggestions :

```
import matplotlib.pyplot as plt

%matplotlib inline
plt.figure(figsize=(12,8))

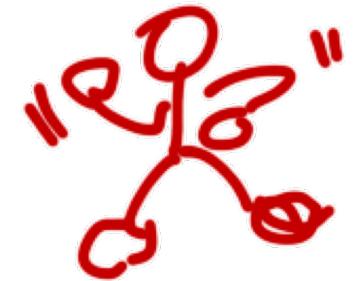
plot_tree(clf, feature_names=feature_names,
          class_names=['Non', 'Oui'], fontsize=10)
```

```
print ( clf.feature_importances_ )
```

# Arbres de Décision en Python

- **Etapes de traitement avec les arbres de décision**

- 1) Préparer les données (**nettoyage**)
  - Uniquement des données **numériques** (*limitation Sklearn*)
- 2) Séparer les données en 2 ensembles : **training** et **test**
  - **Training** :  $\pm 70\%$  / **Test** :  $\pm 30\%$
- 3) Choisir les **features (X set)** et le **target (Y set)**
- 4) Création du modèle
  - Entrainement avec l'opération **fit**
- 5) Tester & évaluer le modèle
  - Tester avec l'opération **predict** et les données de **test**
  - Comparer les **valeurs obtenues** et les **targets réels**
  - Choisir la **métrique** appropriée (*accuracy, precision, recall...*)





UNIVERSITÉ PARIS 1

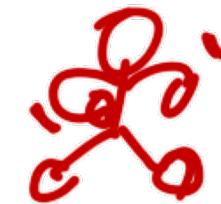
PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Préparer les données

- Préparer les données (**nettoyage**)

- Il faut s'assurer que les données sont propres
    - Éviter les valeurs vides (NA)
    - **Uniquement des données numériques**



- **Encoders** (*LabelEncoder*, *OrdinalEncoder*, *OneHotEncoder*)

- Transformation des catégories en valeurs numériques

[ 'approves' , 'disapproves' ]  [ 0 1 ]

- Différents **encoders** disponibles sur **sklearn.preprocessing**

- **LabelEncoder** : Transformation des **labels** (target) en valeurs **entières** (0 à n-1)
    - **OrdinalEncoder** : Transformation des **données** (features) en valeurs **entières** (de 0 à n-1)
    - **OneHotEncoder** : Transformation des **données** (features) en valeurs **binaires**



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	approves

Target (classes)  
« Y\_set »

Y = labEnc.inverse\_transform(Yenc)

Transformation inverse  
(des valeurs aux labels)

['approves' 'disapproves' 'approves' 'approves' 'disapproves' 'disapproves'

'approves' 'approves'] ← Y\_set : valeurs originales

[0 1 0 0 1 1 0 0] ← Y\_enc : valeurs encodées

# Encoders

## LabelEncoder

Conversion des targets en valeurs numériques

```
from sklearn.preprocessing import LabelEncoder
```

```
labEnc = LabelEncoder()  
labEnc.fit( Y_set )
```

Création et  
entraînement de  
l'encoder

Yenc = labEnc.transform( Y\_set )

Transformation  
des valeurs

valeurs

labEnc.classes\_ ← Classes retrouvées

['approves' 'disapproves']



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
3	male	from Europe	uses Safari	approves
4	female	from US	uses Firefox	disapproves
5	male	from Europe	uses Chrome	disapproves
6	female	from Asia	uses Chrome	approves
7	male	from Asia	uses Chrome	approves

Features (variables)  
« X\_set »

X\_set : valeurs originales

```
['male' 'from Europe' 'uses Safari']
['female' 'from US' 'uses Firefox']
['male' 'from Europe' 'uses Chrome']
['female' 'from Asia' 'uses Chrome']
```

Xord : valeurs encodées

# Encoders

## OrdinalEncoder

Conversion des features en valeurs entiers

```
from sklearn.preprocessing import OrdinalEncoder
```

```
ordEnc = OrdinalEncoder()
ordEnc.fit( X_set )
```

Création et  
entraînement de  
l'encoder

```
Xord = ordEnc.transform( X_set )
```

Transformation  
des valeurs

```
X = ordEnc.inverse_transform(Xord)
```

Transformation inverse  
(des valeurs aux données)

[1.	1.	2.]
[0.	2.	1.]
[1.	1.	0.]
[0.	0.	0.]

```
ordEnc.categories_
```

Liste des  
catégories



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

Données « symboliques »

Catégories

	sex	region	browser	vote
0	male	from US	uses Safari	approves
1	female	from Europe	uses Firefox	disapproves
2	female	from US	uses Safari	approves
-				



Chaque **colonne** sera « éclatée » en **plusieurs**, en fonction du nombre de **catégories** présentes.

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

```
print(Xoh.toarray())
```

# Encoders

## OneHotEncoder

Conversion des **features** en valeurs **binaires**

```
from sklearn.preprocessing import OneHotEncoder
```

```
ohEnc = OneHotEncoder()  
ohEnc.fit( X_set )
```

Création et  
*entraînement* de  
l'encoder

```
Xoh = ohEnc.transform( X_set )
```

Transformation  
des valeurs

valeurs

```
X = ohEnc.inverse_transform(Xoh)
```

Transformation inverse  
(des valeurs aux données)

```
ohEnc.get_feature_names()
```

Liste des  
catégories



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Séparer les données

- Séparer les données en 2 ensembles : **training** et **test**
  - Répartir l'ensemble des données en deux jeux de données
    - **Training** : données qui seront utilisées pour entraîner le modèle
    - **Test** : données qui seront utilisées pour évaluer le modèle
    - Proportion habituelle : 70/80 % vs 30/20 %
  - Opération **train\_test\_split**
    - Sépare de manière « aléatoire » les données en deux ensembles

```
from sklearn.model_selection import train_test_split  
  
df_train , df_test = train_test_split ( df, test_size=0.3 )
```

**Point Python : affectation multiple**  
L'opération retourne **2 valeurs**, chaque **variable** en reçoit une (*dans l'ordre*)

Dataset  
complet

Proportion des données à  
réserver pour les tests



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Séparer les données

```
from sklearn.model_selection import train_test_split
```

On n'oublie pas  
l'import

```
iris_train , iris_test = train_test_split ( df_iris, test_size=0.3 )
```

```
.  
Int64Index: 105 entries, 54 to 148  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype --  
 ---  
 0 sepal length (cm) 105 non-null float64  
 1 sepal width (cm) 105 non-null float64  
 2 petal length (cm) 105 non-null float64  
 3 petal width (cm) 105 non-null float64
```

```
iris_train.info()  
iris_train.head(5)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	target_name
54	6.5	2.8	4.6	1.5	1	versicolor
1	4.9	3.0	1.4	0.2	0	setosa
146	6.3	2.5	5.0	1.9	2	virginica
16	5.4	3.9	1.3	0.4	0	setosa
102	7.1	3.0	5.9	2.1	2	virginica

Chaque ensemble contient une partie des données

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 45 entries, 61 to 95  
Data columns (total 6 columns):  
 # Column Non-Null Count Dtype --  
 ---  
 0 sepal length (cm) 45 non-null float64  
 1 sepal width (cm) 45 non-null float64  
 2 petal length (cm) 45 non-null float64  
 3 petal width (cm) 45 non-null float64
```

```
iris_test.info()  
iris_test.head(5)
```

**train\_test\_split définit deux vues distinctes sur les données**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	target_name
61	5.9	3.0	4.2	1.5	1	versicolor
19	5.1	3.8	1.5	0.3	0	setosa
12	4.8	3.0	1.4	0.1	0	setosa
8	4.4	2.9	1.4	0.2	0	setosa
123	6.3	2.7	4.9	1.8	2	virginica

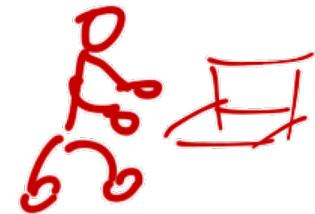


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Hands On !



- Dataset Iris



Iris Versicolor

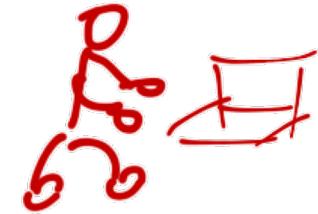


Iris Setosa



Iris Virginica

# Hands On !



## • Exercice : Préparation dataset Iris

*Pour nos prochains exercices, on utilisera le dataset Iris fourni avec SciKit Learn*

- Créer un nouveau Notebook
- Importer les **datasets** de SciKit Learn
  - `from sklearn import datasets`
- Charger le dataset Iris dans une variable
  - `iris = datasets.load_iris()`
- Utiliser la fonction « **dir** » pour voir les attributs de l'objet iris
  - `dir(iris)`
- Afficher à l'écran le description de l'objet iris (**iris.DESCR**)
  - `print (iris.DESCR)`
- Afficher sur l'écran les noms des **features** et des **targets** contenues dans le dataset (**iris.feature\_names** et **iris.target\_names**).

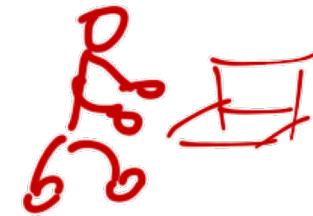


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Hands On !



```
from sklearn import datasets
iris = datasets.load_iris()
dir(iris)
```

Iris plants dataset

\*\*Data Set Characteristics:\*\*

:Number of Instances: 150 (50 in each of three classes)  
 :Number of Attributes: 4 numeric, predictive attributes and the  
 :Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

*Attribut DESCRIPTEUR  
de l'objet  
« iris »*

```
['DESCR',
 'data',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

*Attributs de  
l'objet « iris »*

```
print (iris.DESCR)
print (iris.feature_names)
print (iris.target_names)
```

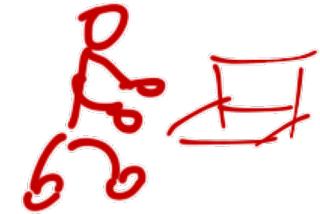
```
['sepal length (cm)', 'sepal width (cm)',
 'petal length (cm)', 'petal width (cm)']
```

*feature\_names*

```
['setosa' 'versicolor' 'virginica']
```

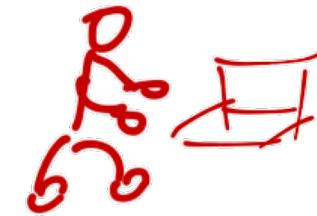
*target\_names*

# Hands On !



- **Exercice : Préparation du dataset Iris**
  - Construire un nouveau **DataFrame df\_iris** avec les données issues du dataset Iris (**iris.data**).
    - `df_iris = pd.DataFrame ( iris.data,  
columns=iris.feature_names )`
  - Ajouter une colonne « **target** » au DataFrame avec les valeurs de targets du dataset (**iris.target**).
    - `df_iris['target'] = iris.target`
  - Afficher les premières lignes du nouveau DataFrame

# Hands On !



```
import pandas as pd
```

```
df_iris = pd.DataFrame ( iris.data,
                           columns=iris.feature_names )
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2

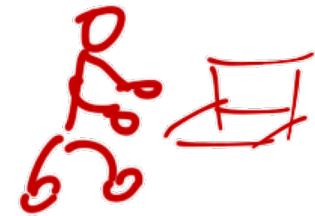
```
df_iris['target'] = iris.target
```

*Si on veut ajouter aussi le target\_name*

```
df_iris['target_name'] = df_iris['target'].apply(
    lambda y: iris.target_names[y] )
df_iris.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	target_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa

# Hands On !



- **Exercice : Séparation données training et test**
  - Séparer le DataFrame `df_iris` en deux ensembles
    - `iris_train` : DF pour le training
    - `iris_test` : DF pour les tests
  - Regarder avec « `info` » les informations de chaque ensemble
  - Afficher les premières lignes de chaque ensemble

```
iris_train, iris_test = train_test_split (df_iris, test_size=0.3)
```

```
iris_train.info()
iris_train.head(5)
```

sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	target_name
54	6.5	2.8	4.6	1	versicolor
1	4.9	3.0	1.4	0	setosa
146	6.3	2.5	5.0	2	virginica
16	5.4	3.9	1.3	0	setosa
102	7.1	3.0	5.9	2	virginica

```

Int64Index: 105 entries, 54 to 148
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   sepal length (cm)  105 non-null   float64
 1   sepal width (cm)  105 non-null   float64
 2   petal length (cm) 105 non-null   float64
 3   petal width (cm)  105 non-null   float64
 4   target            105 non-null   int64  
 5   target_name       105 non-null   object 
dtypes: float64(4), int64(1), object(1)
memory usage: 5.7+ KB

```

# Choisir les *features* et du *target*

- Choisir les colonnes qui seront utilisées en tant que **feature (X set)** et celle qui contient le **target (Y set)**.

`x_train = df_train[ feature_names ]`

*vue partielle sur les données  
(uniquement les features à utiliser)*

*Liste de features (features names)  
qui seront utilisées*

`y_train = df_train['target']`

*vue partielle sur les données  
(uniquement les targets)*

*Colonne avec les valeurs  
de target associées aux  
données*

- On fait la **même chose** pour l'ensemble de **test**



## Choix de features

Particulièrement important  
Dépend des **données** et de  
**l'analyse** à effectuer



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Construire le modèle

- **Construire le modèle → arbre de décision**

- Une fois les données préparées et séparées, on peut entraîner le modèle

```
clf = DecisionTreeClassifier()
```

*Création du modèle*

```
clf.fit ( x_train, y_train )
```

*Entraînement (fitting)  
du modèle*

*Données de training*

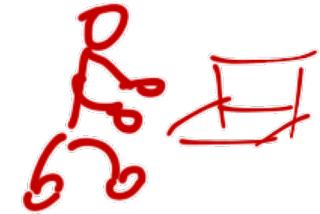
*features de training    targets de training*

```
export_text ( clf,  
              feature_names=feature_names,  
              spacing=3, decimals=2 )
```

*Visualisation de  
l'arbre entraîné  
(optionnel)*

```
plot_tree ( clf,  
            feature_names=feature_names,  
            class_names=target_names,  
            fontsize=10 )
```

# Hands On !

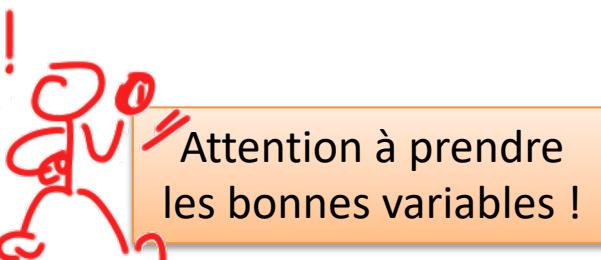


- **Exercice : Choix des features et du target**
  - On va utiliser toutes les **features** du dataset **Iris** pour l'analyse
  - Préparer deux variables **x\_train** et **y\_train** avec les données d'entraînement (respectivement **features** et **target**)
  - Faire la même chose avec les données de test (**x\_test** et **y\_test**)

```
x_train = iris_train[iris.feature_names]  
y_train = iris_train['target']
```

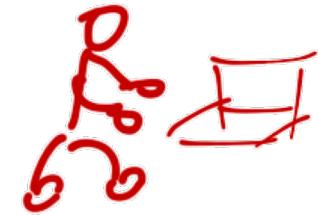
```
x_test = iris_test[iris.feature_names]  
y_test = iris_test['target']
```

```
21      0  
49      0  
35      0  
11      0  
17      0  
Name: target, dtype: int64
```



	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
21	5.1	3.7	1.5	
49	5.0	3.3	1.4	
35	5.0	3.2	1.2	
11	4.8	3.4	1.6	

# Hands On !



## • Exercice : Entraînement du modèle

- Créer un arbre de décision
- Entraîner le modèle avec les variables de training qu'on vient de créer
- Afficher l'arbre entraîné

```
from sklearn.tree import DecisionTreeClassifier  
  
clf = DecisionTreeClassifier()  
clf.fit (x_train, y_train)  
  
texte = export_text(clf,  
                    feature_names=iris.feature_names,  
                    spacing=3, decimals=2)  
print (texte)
```

```
--- petal width (cm) <= 0.80  
    |--- class: 0  
--- petal width (cm) >  0.80  
    |--- petal width (cm) <= 1.65  
        |--- class: 1  
    |--- petal width (cm) >  1.65  
        |--- petal width (cm) <= 1.75  
            |--- sepal width (cm) <= 2.75  
                |--- class: 2  
            |--- sepal width (cm) >  2.75  
                |--- class: 1  
        |--- petal width (cm) >  1.75  
            |--- petal length (cm) <= 4.85  
                |--- sepal length (cm) <= 5.95  
                    |--- class: 1  
                |--- sepal length (cm) >  5.95  
                    |--- class: 2  
            |--- petal length (cm) >  4.85  
                |--- class: 2
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Tester & évaluer le modèle

- **Tout modèle doit être testé**
  - Vérification / validation des résultats
- **Usage des données de test (test set)**
  - Comparaison entre les résultats obtenus avec le modèle et les valeur de **target** identifiés dans les données

Les prédictions ne correspondent pas toujours à la réalité

On peut alors comparer les **labels obtenus** (**y\_pred**) avec les **labels** (**targets**) identifiées (**y\_test**)

```
print (y_pred)  
print (y_test.values)
```

Opération **predict** sur les données de **test**

```
y_pred = clf.predict (x_test)
```

```
[2 2 2 1 1 1 0 0 0 1 1 0 0 0 2 1 2 1 0 0 2 2 1 1 2  
1 2 1 2 0 1 2 2]  
[2 2 2 1 1 1 0 0 0 1 1 0 0 0 2 1 2 1 1 0 0 2 2 1 1 2  
1 1 1 2 0 1 2 2]
```

# Tester & évaluer le modèle

- Pour évaluer le modèle, on utilise des métriques
- Différentes métriques connues
  - **Accuracy** : proportion (ou nb) de prévisions correctes
  - **Precision** : proportion des corrects sur l'ensemble des réponses
  - **Recall** : proportion des corrects sur ce qu'on devrait retrouver
  - **Confusion matrix** : matrice croisant les valeurs observées et les valeurs prédites
  - ...
- Plusieurs métriques disponibles dans la bibliothèque
  - Bibliothèque **sklearn.metrics**
  - **accuracy\_score**
  - **precision\_score**
  - **recall\_score**
  - **Confusion\_matrix**



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Tester & évaluer le modèle

```
from sklearn.metrics import accuracy_score
```

Résultat : proportion entre 0 et 1

```
acc = accuracy_score ( y_test , y_pred )
```

Ce qu'on avait sur les  
données (**Y true**)

Ce qu'on a obtenu avec  
le modèle (**Y predicted**)

```
print(acc) → 0.9555555555555556
```

```
acc = accuracy_score ( y_test, y_pred, normalize=False )
```

Résultat : nombre de prévisions correctes

```
print(acc, "/" , y_test.count()) → 43 / 45
```



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Tester & évaluer le modèle

```
from sklearn.metrics import precision_score, recall_score
```

Ce qu'on avait sur les données (*Y true*)

Ce qu'on a obtenu avec le modèle (*Y predicted*)

```
prec = precision_score ( y_test, y_pred, average='weighted' )
```

Si plusieurs labels, on peut calculer une valeur de précision en tenant compte du nombre d'instances de chaque label

ou calculer la précision pour chaque label

average=None

average='binary'

Si labels binaires (True / False)

Même chose pour le recall...

```
rec = recall_score ( y_test, y_pred, average='weighted' )
```

average='weighted'

average=None

precision_score	0.9611111111111111	[1. 1. 0.875]
-----------------	--------------------	---------------

recall_score	0.9555555555555556	[1. 0.84615385 1.]
--------------	--------------------	--------------------

]



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Tester & évaluer le modèle

```
from sklearn.metrics import confusion_matrix
```

Ce qu'on avait sur les données (*Y true*)

Ce qu'on a obtenu avec le modèle (*Y predicted*)

```
mc = confusion_matrix ( y_test ,y_pred )
```

```
print (mc)
```

```
[[14  0  0]
 [ 0 16  0]
 [ 0  3 12]]
```

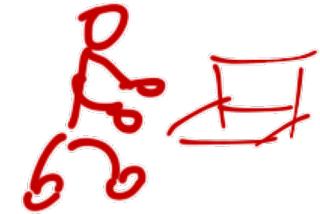


Classe réelle

Classe prédite par le modèle

	0 (setosa)	1 (versicolor)	2 (virginica)
0 (setosa)	14	0	0
1 (versicolor)	0	16	0
2 (virginica)	0	3	12

# Hands On !



- **Exercice : Tester & évaluer le modèle**
  - Utiliser le modèle avec les **données de test**
  - Afficher les valeurs obtenues
  - Utiliser la métrique « **accuracy** » afin d'évaluer les résultats
  - Utiliser la métrique « **precision** » afin d'évaluer les résultats **par label**
  - Afficher la « **confusion matrix** »

```
acc = accuracy_score ( y_test, y_pred )
prec = precision_score ( y_test, y_pred, average=None )
mc = confusion_matrix( y_test ,y_pred)
```

```
0.9111111111111111
[1.          0.83333333 0.93333333]
[[14  0  0]
 [ 0 16  0]
 [ 0  3 12]]
```

```
1 versicolor
2 virginica
2 virginica
1 versicolor
1 versicolor
2 virginica
```

**Suggestion :**  
Si on veut afficher les  
**target names**

```
for val in y_pred :
    label =
        iris.target_names[val]
    print ( val, label )
```

# Validation croisée

- **Important :**
  - Le **modèle** peut **varier** à chaque **exécution**
  - La **qualité** et la **taille** du **dataset** ont une influence importante sur la **qualité** du **modèle**
- **Validation croisée :**
  - **Objectif :**
    - Minimiser les problèmes liés au *dataset* (pas ou peu équilibré, trop petit...)
  - **Principe :**
    - Réaliser **plusieurs itérations** (c.a.d. plusieurs modèles) avec différents ensembles de training et de test
    - Multiples **divisions aléatoires training / test**



UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Validation croisée

- Illustration du principe avec `ShuffleSplit`

- `ShuffleSplit` produit une liste d'index pour les *splits*

```
from sklearn.model_selection import ShuffleSplit
```

```
rs = ShuffleSplit ( n_splits=5 , test_size=0.3 )
```

*Nombre d'itérations à faire*



*Proportion ensemble de test*

```
accuracy_scores = []
```

*On garde les scores obtenus dans une liste*

```
for train_index, test_index in rs.split(df_iris) :
```

*À chaque itération,  
`split` fournit deux  
indexes : un pour le  
**training set** et un  
pour **test set***

```
x_train = df_iris.loc[train_index, iris.feature_names]
```

```
x_test = df_iris.loc[test_index, iris.feature_names]
```

```
...
```

```
clf = DecisionTreeClassifier()
```

```
clf.fit(x_train, y_train)
```

```
y_pred = clf.predict(x_test)
```

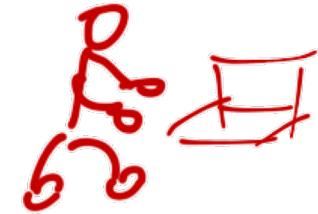
```
accuracy_scores.append( accuracy_score(y_test, y_pred) )
```

*À la fin, les ≠ valeurs d'accuracy sont disponibles*

[0.9333333333333333, 0.9555555555555555]

[0.9777777777777777]

# Hands On !



- Exercice : Tester le principe d'évaluations multiples
  - Construire et évaluer plusieurs modèles
  - Utiliser **ShuffleSplit** pour la division **training/test** du dataset

```
rs = ShuffleSplit(n_splits=5 , test_size=0.3)
accuracy_scores = []
```

*1<sup>er</sup> bloc : préparer le **ShuffleSplit** et la liste pour les évaluations*

```
for train_index, test_index in rs.split(df_iris) :
    x_train = df_iris.loc[train_index, iris.feature_names]
    x_test = df_iris.loc[test_index, iris.feature_names]

    y_train = df_iris.loc[train_index, 'target']
    y_test = df_iris.loc[test_index, 'target']

    clf = DecisionTreeClassifier()
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    accuracy_scores.append(accuracy_score(y_test, y_pred))
```

*2<sup>eme</sup> bloc : préparer X et Y sets de training et de tests*

*3<sup>eme</sup> bloc : entraîner et tester un nouveau modèle*

```
print ( accuracy_scores )
```

*4<sup>eme</sup> bloc : visualiser les évaluations obtenues*



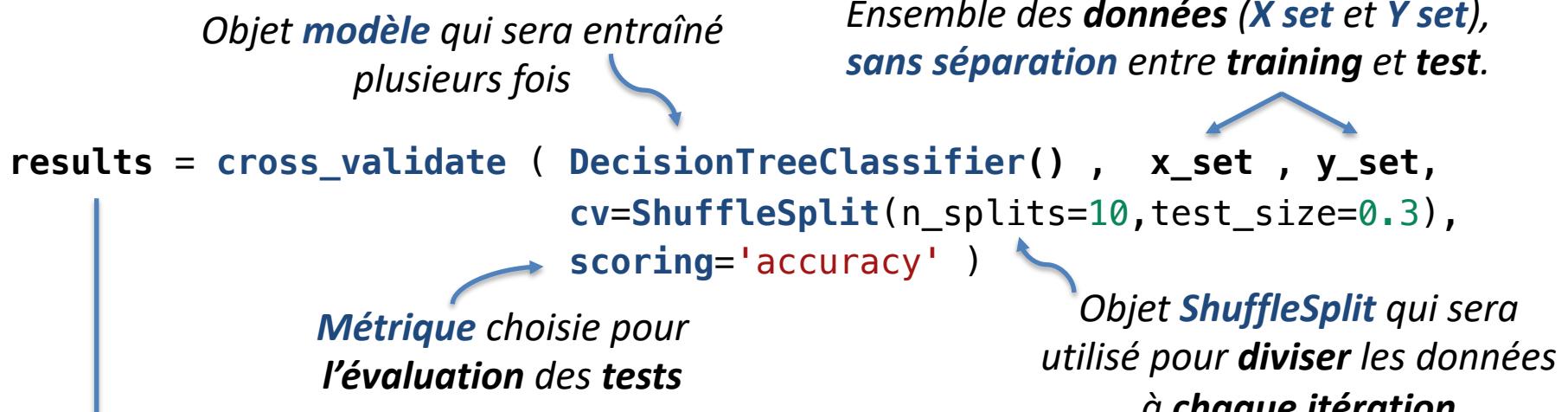
UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT  
DE LA SORBONNE

# Validation croisée

- Opération **cross\_validate** automatise le processus

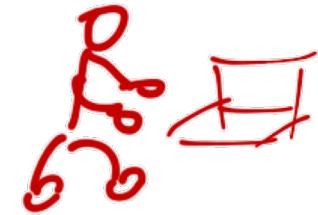


**Résultat** : un dictionnaire avec les **temps d'exécution** et les **métriques**

```
scores = pd.DataFrame ( results )
```

	fit_time	score_time	test_score
0	0.002809	0.003653	0.955556
1	0.005952	0.003189	0.955556
2	0.002753	0.002373	0.955556
3	0.002047	0.001394	0.977778
4	0.002096	0.002007	0.955556
5	0.005273	0.003410	0.955556

# Hands On !



## • Exercice : Réaliser une validation croisée

- Toujours avec le dataset Iris, réaliser une validation croisée
- Construire un DataFrame avec les résultats
- Afficher les résultats obtenus

```
clf = DecisionTreeClassifier()  
rs = ShuffleSplit (n_splits=10, test_size=0.3)
```

*1<sup>er</sup> bloc : préparer les objets  
**DecisionTreeClassifier** et **ShuffleSplit***

```
x_set = df_iris[iris.feature_names]  
y_set = df_iris['target']
```

*2<sup>ème</sup> bloc : préparer le **X set (features)** et  
**Y set (target)** avec l'ensemble des données*

```
results = cross_validate ( clf , x_set , y_set ,  
                           cv=rs,  
                           scoring='accuracy' )
```

*3<sup>ème</sup> bloc : utiliser l'opération  
**cross\_validate** pour réaliser  
la validation croisée*

```
scores = pd.DataFrame ( results)  
scores['test_score'].plot(figsize=(10,4))
```

*4<sup>ème</sup> bloc : **visualiser** les résultats  
(à l'aide d'un **DataFrame**)*