



ENVIRONNEMENT ET PROJETS

MATH ET DATASCIENCE

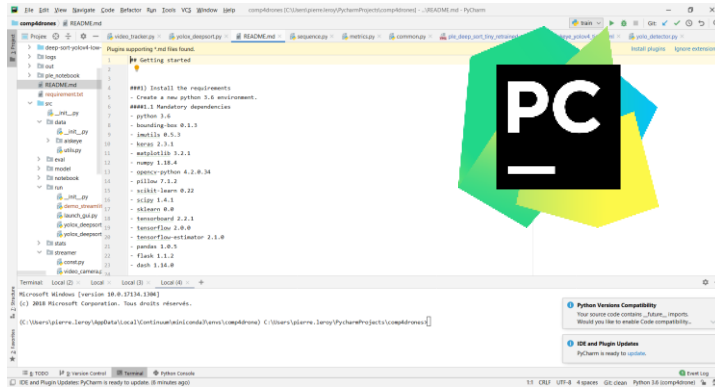




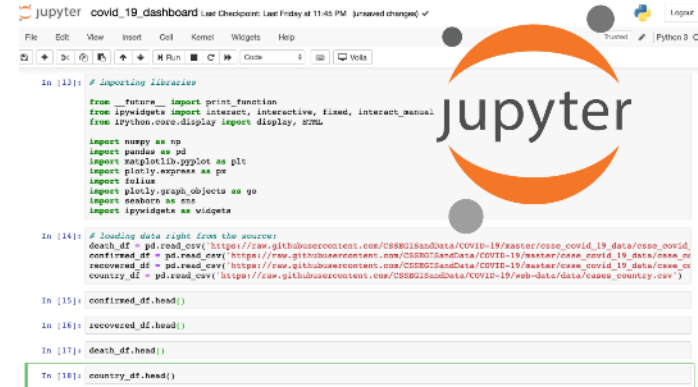
**01**

## **IDE VS NOTEBOOK**

# IDE VS NOTEBOOK



Pycharm community



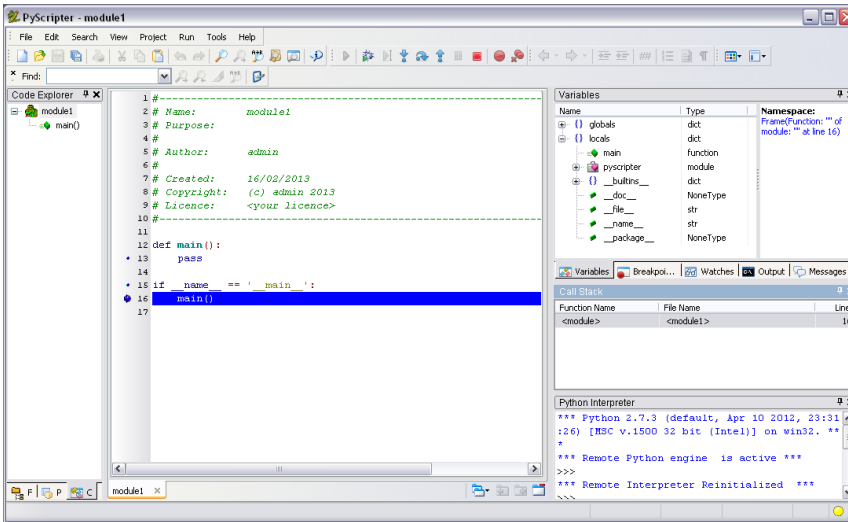
Jupyter notebook

# IDE VS NOTEBOOK

**Deux questions :**

**Qui parmi-vous a déjà manipulé l'un ou l'autre de ces outils ?**

**Lequel des deux pensez-vous le plus adapté à la datascience ?**



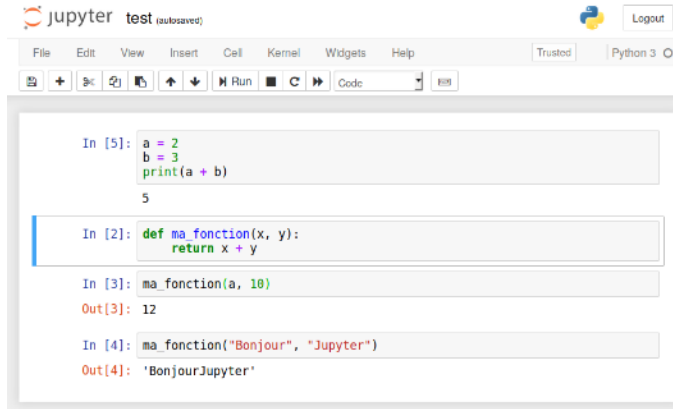
## Avantages:

- Dispose d'un Debugger
- Gain de temps sur des gros projets
- Gere beaucoup de types de fichiers
- Paramètres et configuration enregistrables
- Facilement interfaçable avec Git
- Et pleins d'autres outils

## Inconvénients:

- Parfois lourd à mettre en place
- Moins interactif
- Les scripts python se ré-exécutent depuis le début

# NOTEBOOK



```

In [5]: a = 2
        b = 3
        print(a + b)
        5

In [2]: def ma_fonction(x, y):
        return x + y

In [3]: ma_fonction(a, 10)
Out[3]: 12

In [4]: ma_fonction("Bonjour", "Jupyter")
Out[4]: 'BonjourJupyter'

```

## Avantages:

- Rapide
- Permet des interactions et manipulations rapides
- Inclusion possible de texte, graphes, ....
- Permet facilement de présenter du code et des résultats ( démo avec un client, ...)

## Inconvénients:

- Souvent source d'erreur quand long
- Code dur à entretenir
- Il est dur de diviser le code entre plusieurs notebooks pour éviter que ceux-ci soient trop long
- Peu adapté à du collaboratif

# IDE VS NOTEBOOK

Quel est le meilleur outil pour la datascience, **notebook ou ide** ?

- ils sont **complémentaires**, n'en délaissions pas une au profit de l'autre !

Pour **explorer vos données, tester rapidement** une nouvelle bibliothèque, **présenter des résultats** nécessitant l'exécution de code

- > **LE NOTEBOOK**

Pour traiter efficacement **l'ensemble du code et fichier annexes du projet** :

- > **L'IDE**

A noter que :

- les fonctionnalités présentes des notebooks sont en train d'augmenter, peut-être un jour auront-ils toutes les fonctionnalités des IDEs ?
- les IDEs gèrent (avec plus ou moins de succès) les notebooks



**02**

**CRÉATION D'UN PROJET**



	conda	pip
install python package	✓	✓
create virtual environment	✓, built-in	✗, requires virtualenv or venv
package format	.tar.bz2 , .conda	.whl , .tar.gz
manages	binaries	wheel or source
can require compilers	✗	✓
package types	any	Python-only
dependency checks	✓	✗
package sources	Anaconda repo and Anaconda cloud	PyPI

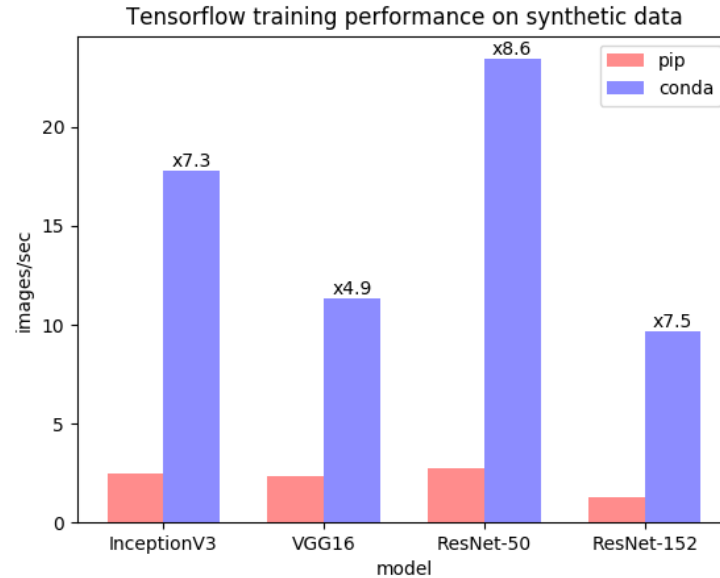
- **Pip et conda sont les deux gestionnaires d'environnement les plus utilisés en Python**
- **Conda peut installer des binaires optimisés - > meilleures performances**
- **Plus de bibliothèques et de versions disponibles avec pip**
- **Le vrai argument : Conda permet d'utiliser pip si besoin !**

**Une approche semble particulièrement intéressante, celle d'utiliser un environnement d'exécution au sein d'un container docker.**

**Pourquoi ?**

**Tout simplement car tous les développeurs du projet sont certains d'utiliser le même environnement d'exécution (le container).**

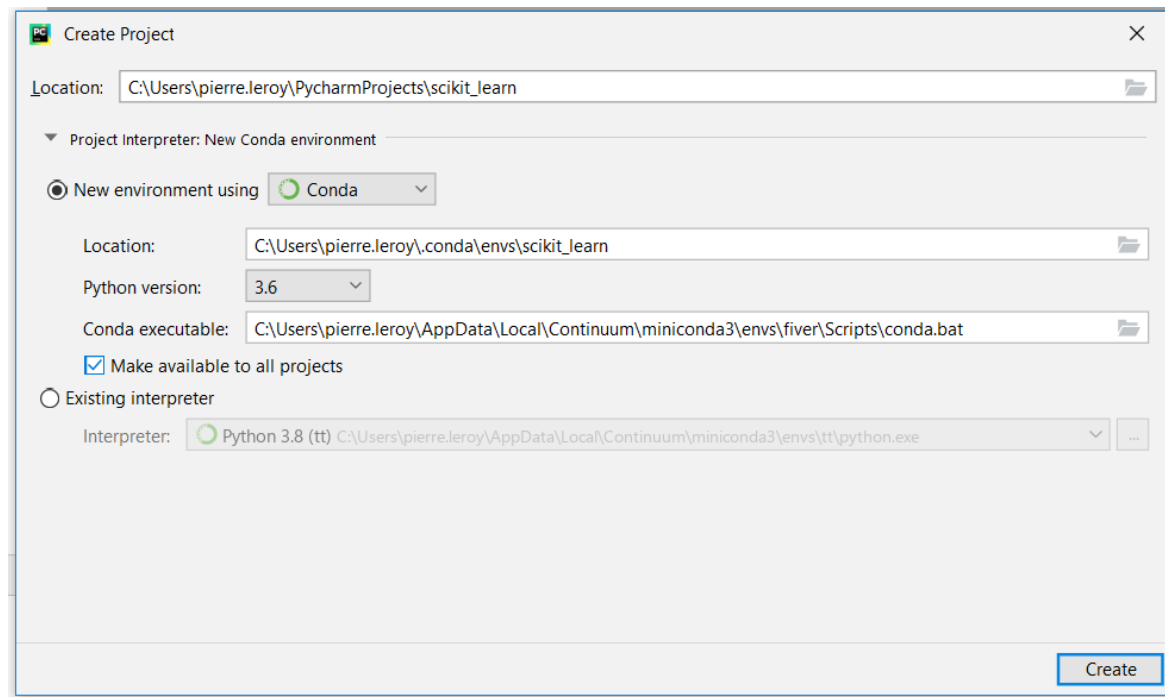
**Dans la version community, Pycharm ne permet pas de le faire. Vscode le fait 😊**



[source](#)

- Toujours se créer un environnement en début de projet (conda create ou virtual env)
- Installer les packages avec conda, sauf si pas disponible, dans ce cas utiliser pip

# CRÉATION D'UN PROJET SOUS PYCHARM



# EXEMPLE D'UN PROJET SCALIAN

```
|— params --> Liste des fichiers d'expérience
| |— *.yaml --> Fichier de configuration d'une XP
| |— archive -->
| |— ckpts --> Checkpoints des modèles
|— README.md --> Fichier de Documentation
|— requirement.txt --> Liste des dépendances python
|— Notebooks --> Liste des notebooks et de suivi client
|— src --> Contient le code source
| |— app --> Classes pour fonctionnement IHM
| |— data --> Script de traitement de données
| |— logging --> logger
| |— model --> Scripts des modèles
| |— preprocessing --> Scripts de Préprocessings
| |— run --> Entrée du projet (contient un fichier de prédiction et d'entraînement)
| |— utils --> Utilitaires
| |— visu --> Fonctions de visualisations
```

# BONNES PRATIQUES : YAML POUR LES PARAMÈTRES

```
split:
  train_test_split:
    test_size: 0.2
    random_state: 42
  preprocessing:
    Interpld:
      points_to_interpolate: 600
      normalisation: Normalize_remove_nan
    PCA:
      n_components: 2
  model:
    RandomForestClassifier:
      n_estimators: 100
```

- Facilement lisible
- Facilement modifiable
- Evite

# EXEMPLE D'UTILISATION DE YAML

```
from absl import app, flags
flags.DEFINE_string('config', 'cfgs/deep_sort/deep_sort_tiny.yaml', "path to config file")
FLAGS = flags.FLAGS

def main(_argv):
    FLAGS(_argv)
    with open(FLAGS.config, 'r') as f:
        cfg = yaml.safe_load(f.read())

if __name__ == "__main__":
    app.run(main)
```

En utilisant la librairie yaml, vous pouvez récupérer la valeur de votre fichier de config au sein **d'un dictionnaire** python

# EXEMPLE D'UTILISATION DE YAML

Name:  ☐ Share through VCS ☐ Allow parallel run

Configuration Logs

Script path:

**Parameters:**

Environment

Environment variables:

Python interpreter:

Interpreter options:

Working directory:

☒ Add content roots to PYTHONPATH

☒ Add source roots to PYTHONPATH

Execution

☐ Emulate terminal in output console

☐ Run with Python Console

☐ Redirect input from:

Before launch:

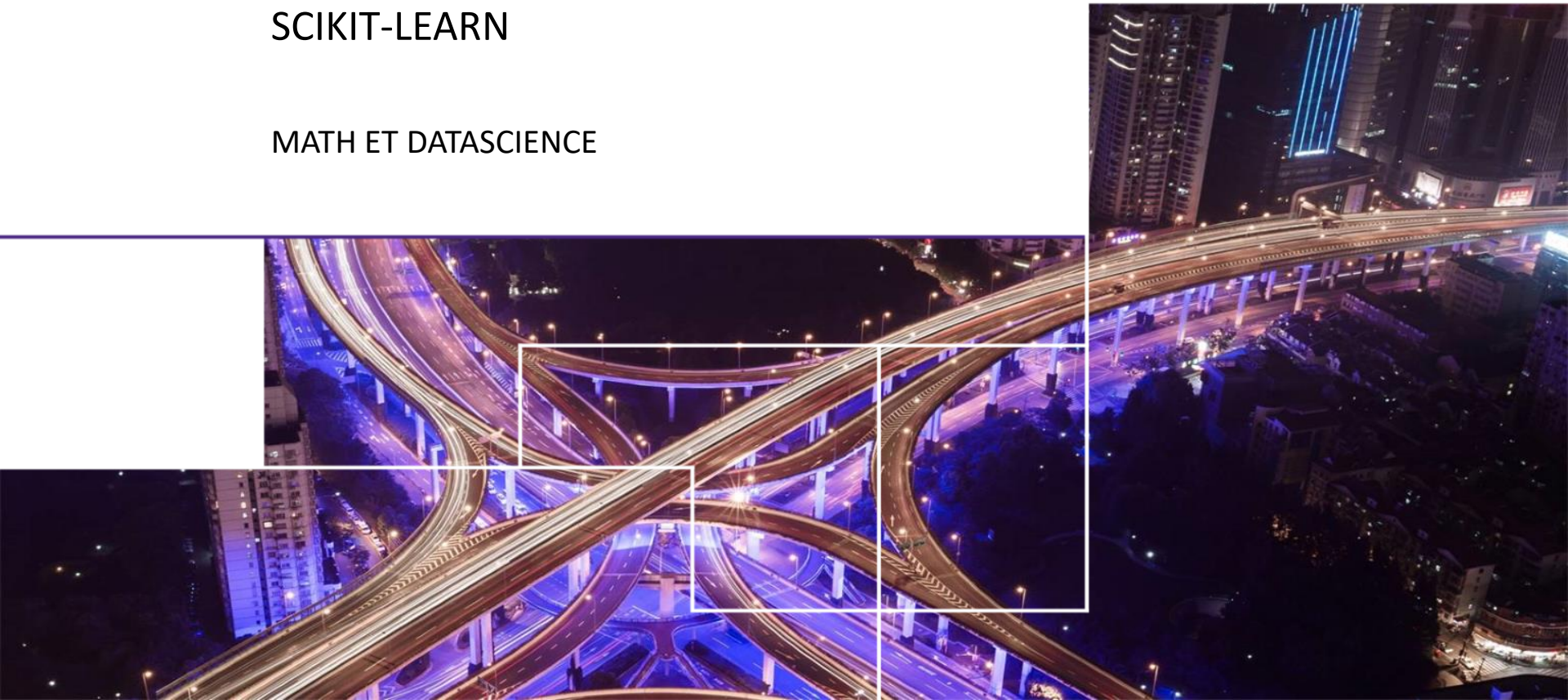
**Vous n'avez plus qu'à lancer votre code en fournissant à l'argument config le lien vers votre fichier d'expérience**





SCIKIT-LEARN

MATH ET DATASCIENCE



- Bibliothèque en Python créée par l'Inria
- Potentiellement la bibliothèque la plus utilisée en Datascience
  - *Elle contient un ensemble de méthodes « génériques » pour traiter les données et y appliquer des modèles de machine learning*
  - *Pour des méthodes complexes, ou propre à un type de données spécifiques, il faut souvent chercher ailleurs*
- Son formalisme est devenu un standard en machine learning
- Première version bêta en 2010.
- Officiellement passée en version 1.0 le 24 September 2021

## 6.3. Preprocessing data

- 6.3.1. Standardization, or mean removal and variance scaling
- 6.3.2. Non-linear transformation
- 6.3.3. Normalization
- 6.3.4. Encoding categorical features
- 6.3.5. Discretization
- 6.3.6. Imputation of missing values
- 6.3.7. Generating polynomial features
- 6.3.8. Custom transformers

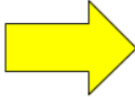
## 6.4. Imputation of missing values

- 6.4.1. Univariate vs. Multivariate Imputation
- 6.4.2. Univariate feature imputation
- 6.4.3. Multivariate feature imputation
- 6.4.4. References
- 6.4.5. Nearest neighbors imputation
- 6.4.6. Marking imputed values

**Allez voir la documentation en ligne !**

[source](#)

# UN EXEMPLE : LE ONE HOT ENCODING

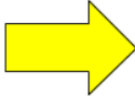


Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

**Avec Pandas rien de très nouveau ..**

```
one_hot_encoded_training_predictors = pd.get_dummies(train_predictors)
```



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

**Avec Pandas rien de très nouveau ..**

```
one_hot_encoded_training_predictors = pd.get_dummies(train_predictors)
one_hot_encoded_test_predictors = pd.get_dummies(test_predictors)
```

# AVEC SCIKIT :

```
from sklearn.preprocessing import OneHotEncoder

# Create the encoder.
encoder = OneHotEncoder(handle_unknown="ignore")
encoder.fit(X_train)    # Assume for simplicity all features are categorical.

# Apply the encoder.
X_train = encoder.transform(X_train)
X_test = encoder.transform(X_test)
```

- Permet d'éviter les erreurs lorsqu'une valeur est présente dans le *test* mais pas dans le *train*. Par exemple si train contient « rouge », »vert » et le *test* « rouge », « jaune », les colonnes vertes et rouges uniquement seront créées pour *test*.

# UN AUTRE EXEMPLE : STANDARDSCALER

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
X_train["Age_scaled"] = scaler.fit_transform(X_train["Age"].values.reshape(-1, 1))
```

```
df_train[["Age", "Age_scaled"]]
```

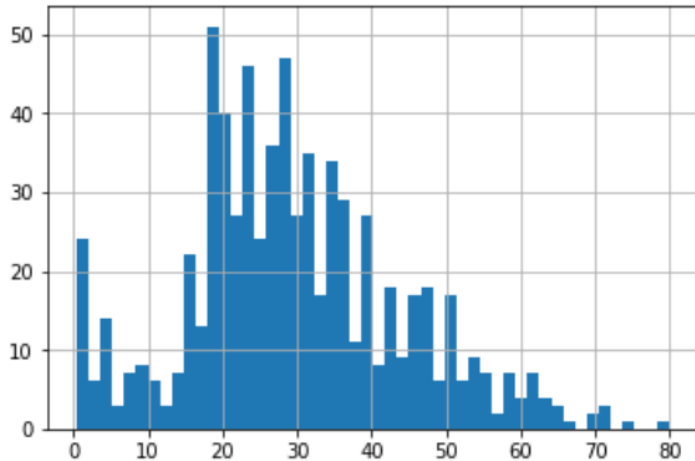
	Age	Age_scaled
0	22.0	-0.530377
1	38.0	0.571831
2	26.0	-0.254825

- Toutes ces classes s'utilisent de la même manière :
  - On importe la classe
  - On instancie l'objet
  - On applique le fit
  - On applique le transform pour utiliser l'objet (fit\_transform enchaîne les deux étapes)

# STANDARD SCALER

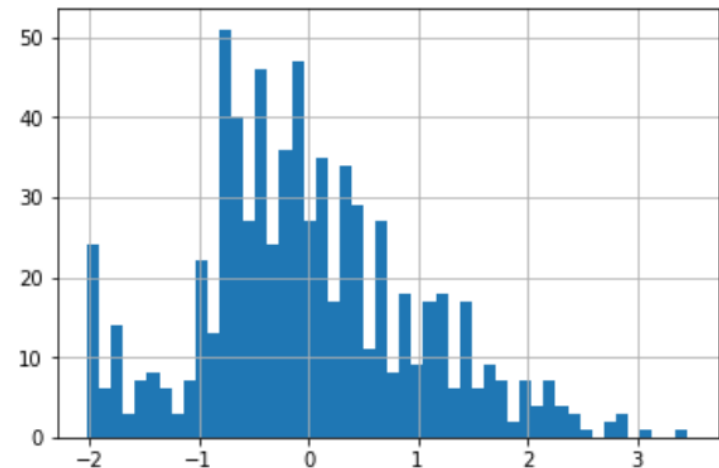
```
X_train["Age"].hist(bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x23972650860>
```



```
X_train["Age_scaled"].hist(bins=50)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2397259acc0>
```

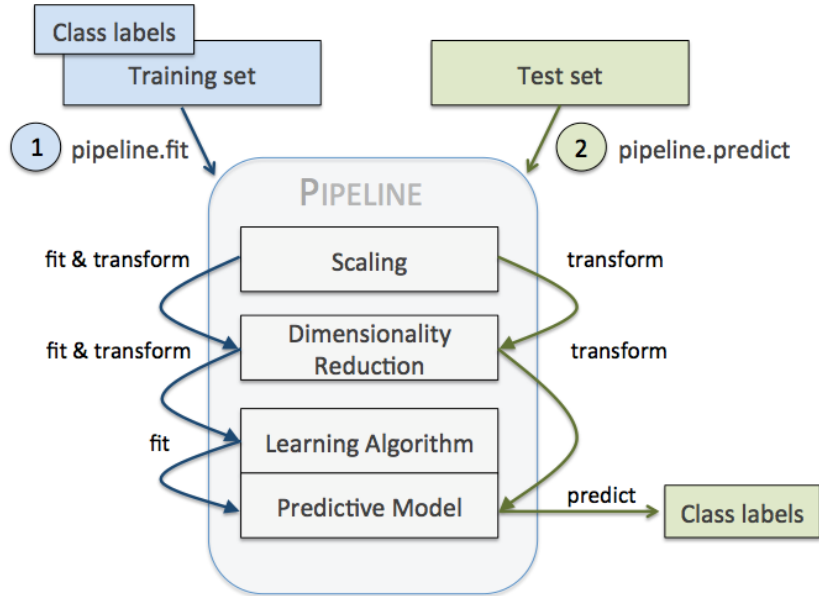


**Standard scaler conserve la même distribution des données mais la transforme de telle sorte que la moyenne soit à 0. Nous verrons dans la suite du cours pourquoi ce genre de transformation est presque indispensable en Machine Learning**



# COMMENT S'ORGANISER DANS UN PROJET DE DATASCIENCE COMPLEXE AVEC SCIKIT ?





**Permet de construire un objet pipeline qui encapsule toutes les méthodes jusqu'à la prédiction.**

**Cette encapsulation permet de rendre le code robuste et rejouable facilement.**

```
>>> from sklearn.svm import SVC
>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.pipeline import Pipeline
>>> X, y = make_classification(random_state=0)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,
...                                                    random_state=0)
>>> pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC())])
>>> # The pipeline can be used as any other estimator
>>> # and avoids leaking the test set into the train set
>>> pipe.fit(X_train, y_train)
Pipeline(steps=[('scaler', StandardScaler()), ('svc', SVC())])
>>> pipe.predict(X_test)
```

Cette méthode est très utilisée et pratique. Elle permet de construire un objet pipeline qui encapsule toutes les méthodes allant de la data préparation jusqu'à la prédiction. Cette encapsulation permet de rendre le code robuste et rejouable facilement.

# BONUS : COMMENT CRÉER SA PROPRE ÉTAPE ?

```
1 import numpy as np
2 from sklearn.base import BaseEstimator, TransformerMixin
3 from scipy.interpolate import interp1d
4 from src.data.meta_array import MetaArray
5
6
7 class Interp1d(BaseEstimator, TransformerMixin):
8     def __init__(self):
9         """
10         Initialisation
11         """
12         None
13
14     def fit(self, X, y=None):
15         """
16         Implements sklearn estimator fit. Do nothing since no training is required for interpolation.
17         :param X: data
18         :param y: optional, labels
19         :return: self
20         """
21         # print("Fitting interp1d")
22         return self
23
24     def transform(self, X, y=None):
25         """
26         Interpolate time series
27         :param X: list of data containing Safe data and metadatatime series
28         """
29         X = interpolate_dataset(X)
30         return X
31
```

# COMMENT ENREGISTRER NOTRE PIPELINE ?

```
import pickle
with open(artifact_path / pickle_name, 'wb+') as f:
    pickle.dump(pipeline, f)
```

L'objet pickle permet de sauvegarder un objet d'une classe avec toutes ses valeurs et ses méthodes.

```
file = open("model/pipeline.pkl", 'rb')
pipeline = pickle.load(file)
pipeline.predict(X_test)
```

En rechargeant le fichier, l'objet peut être utilisé comme avant.

**La popularité de sklearn a fait de cette librairie la norme.**

**Bien l'utiliser, par exemple avec l'objet pipeline présente de nombreux avantages : reproductibilité, qualité de code, robustesse, facilité d'export, ...**

**Pensez à consulter sa documentation pour toute information sur ces capacités, les méthodes disponibles pour votre cas d'usage, etc.**

**Son utilisation peut s'avérer limitant dans certains projets plus complexes, mais dans beaucoup de cas, d'autres librairies complémentaires, basées sur le même formalisme, existent.**

**Scikit-learn encourage une programmation objet (PO) des projets de datascience. La PO se prête en général bien au Machine Learning, et rend son utilisation plus facile.**

**Un exemple de pipeline basé sur scikit, mais offrant plus de fonctionnalités : IMBLEARN.**

**IMBLEARN s'utilise et s'interface exactement de la même manière que Scikit.**

**Elle possède entre autres :**

- **Des méthodes d'échantillonnages plus avancées**
- **Un objet pipeline permettant plus de fonctionnalités, comme des modifications des labels.**

**Si Sklearn ne correspond pas à votre besoin, pensez à chercher ailleurs !**