2022/2023

Pandas
SKLearn
MatplotLib

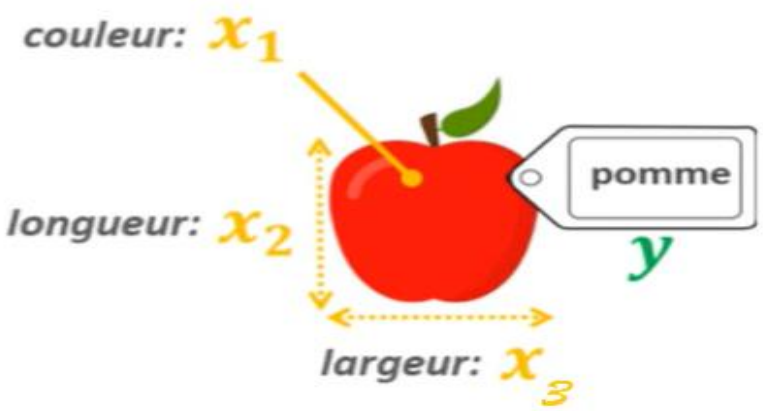
Réalisé par: Naïma Daghfous

l'Apprentissage Supervisé

La machine reçoit des données
caractérisées par des variables x
et annotées d'une variable y

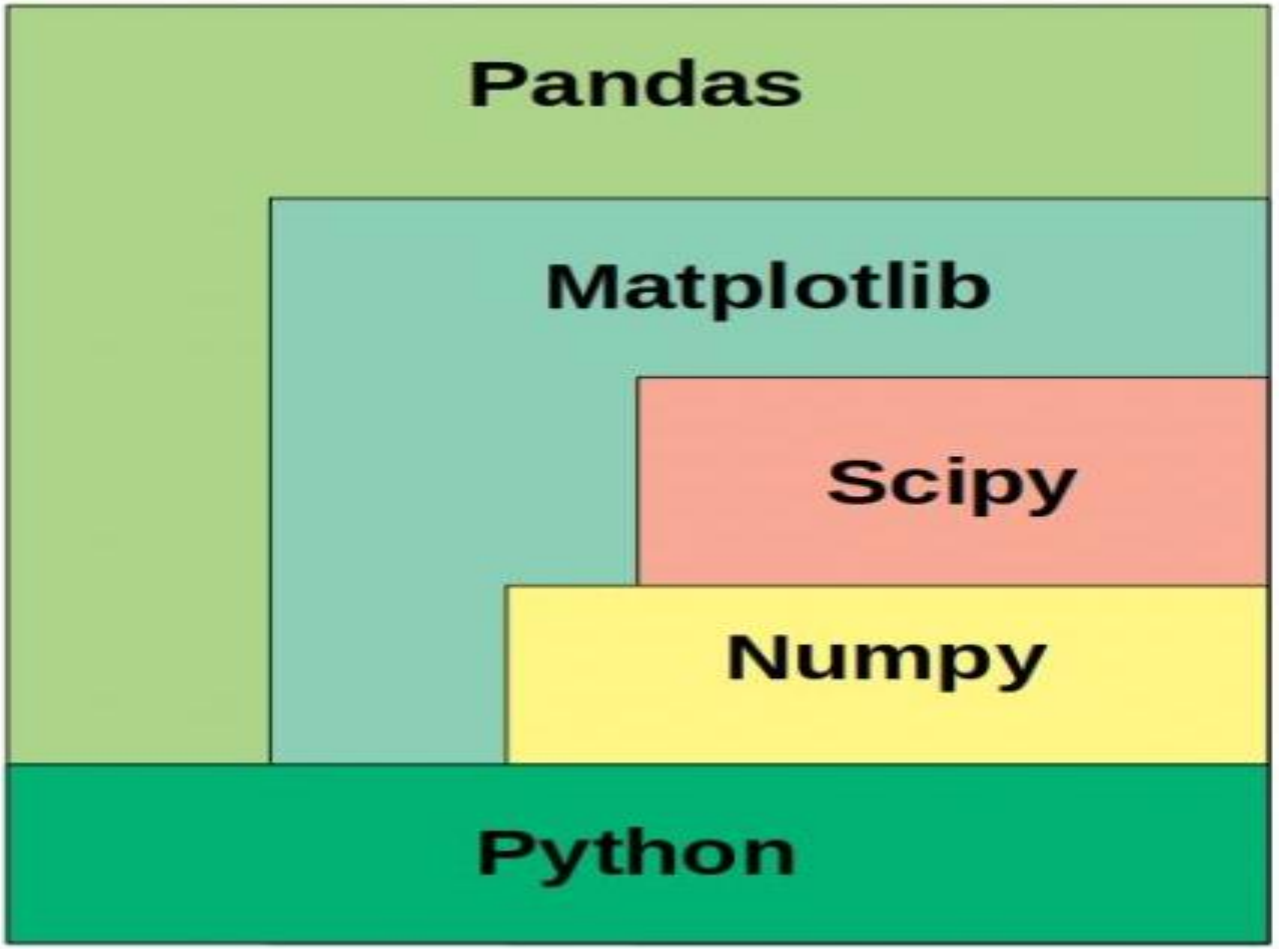
Features

Label / Target



Prédire y
en fonction de x

Pandas



Pandas

- **Pandas est une bibliothèque open-source permettant la manipulation et l'analyse de données de manière simple et intuitive en Python**

- L'une des forces de Pandas est qu'il se base sur la très populaire bibliothèque **NumPy**. En plus de cela, les données produites par Pandas sont souvent utilisées comme données en entrée pour les fonctions de **plotting** de **Matplotlib**, l'analyse statistique en **SciPy**, les algorithmes de **machine learning** en **Scikit-learn**. Les data scientists l'utilisent pour le chargement, le traitement et l'analyse des données tabulaires (données stockées sous format .csv, .tsv ou .xlsx)

Pandas

- Ce qui fait la force de Panda est qu'elle :

- fournit une structure de donnée appelée **Dataframe** rapide et efficace pour la manipulation des données avec indexation intégrée ;
- dispose d'outils pour lire et écrire dans des fichiers de différents formats (.csv, .txt, .xlsx, .sql, .hdf5, etc…) ;
- offre une flexibilité pour traiter les données de type hétérogènes ou manquantes ;
- est open source ;
- fournit une documentation très détaillée et facile à lire

Pandas

pd.**read_excel()**
pd.**read_csv()**
pd.**read_....()**

} *Charger des données (selon le format)*

df.**head()** *Afficher le début du DataFrame*

df.**describe()** *Statistiques rapides*

df.**drop**(['*column*', '*column*', ...]) *Éliminer certaines colonnes*

df.**dropna**(axis=0) *Éliminer les lignes aux données manquantes*

df.['*column*'].**value_counts**() *Compter les répétitions*

df.**groupby**(['*column*'])

Pandas

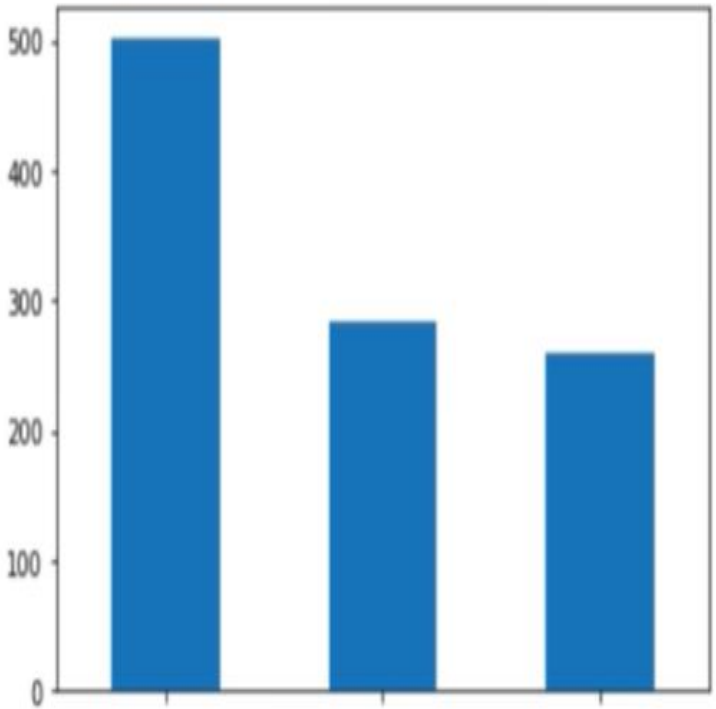
Pandas utilise Matplotlib.pyplot !

```
df.plot()
df.plot.bar()
df.hist(bins=...)
df.plot.scatter(x=..., y=...)
```

pd.plotting.scatter_matrix(df)

```
data['pclass'].value_counts().plot.bar()
data['']
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f6cd8db208>
```



Pandas

Dans Pandas, 2 structures de données:

- **Séries**
- **DataFrames**

Series + **Series** = **DataFrame**

	apples
0	3
1	2
2	0

+

	oranges
0	0
1	3
2	7

=

	apples	oranges
0	3	0
1	2	3
2	0	7

Série : tableau **Numpy 1D + axe d'index**

DataFrame : ~ Dictionnaire de Séries

Rappel: Dict ['clef '] = valeur

Df ['column'] = une Série

	pclass	survived	sex	age
0	1	1	female	29.0000
1	1	1	male	0.9167
2	1	0	female	2.0000
3	1	0	male	30.0000

Pandas

data ['age'] = Série (ndarray)

data ['age'][0:10] (indexing)

data ['age'] < 18 (mask)

data[data ['age'] < 18] (boolean indexing)

data [['age', 'pclass']] = DataFrame

data.iloc [0:2, 0:2] -> localisation par index

data.loc[0:2 , 'age']

```
data['age'][0:10]
0    29.0000
1     0.9167
2     2.0000
3    30.0000
4    25.0000
5    48.0000
6    63.0000
7    39.0000
8    53.0000
9    71.0000
```

```
data['age'] < 18
0    False
1     True
2     True
3    False
4    False
5    False
6    False
7    False
8    False
9    False
```

```
data.iloc[0:2,0:2]
```

	pclass	survived
0	1	1
1	1	1

```
data.loc[0:2, ['age', 'sex']]
```

	age	sex
0	29.0000	female
1	0.9167	male
2	2.0000	female

Pandas

data ['age'] = Série (ndarray)

data ['age'][0:10] (indexing)

data ['age'] < 18 (mask)

data[data ['age'] < 18] (boolean indexing)

data [['age', 'pclass']] = DataFrame

data.iloc [0:2, 0:2] -> localisation par index

data.loc[0:2 , 'age']

```
data['age'][0:10]
```

```
0    29.0000
1     0.9167
2     2.0000
3    30.0000
4    25.0000
5    48.0000
6    63.0000
7    39.0000
8    53.0000
9    71.0000
```

```
data['age'] < 18
```

```
0    False
1     True
2     True
3    False
4    False
5    False
6    False
7    False
8    False
9    False
```

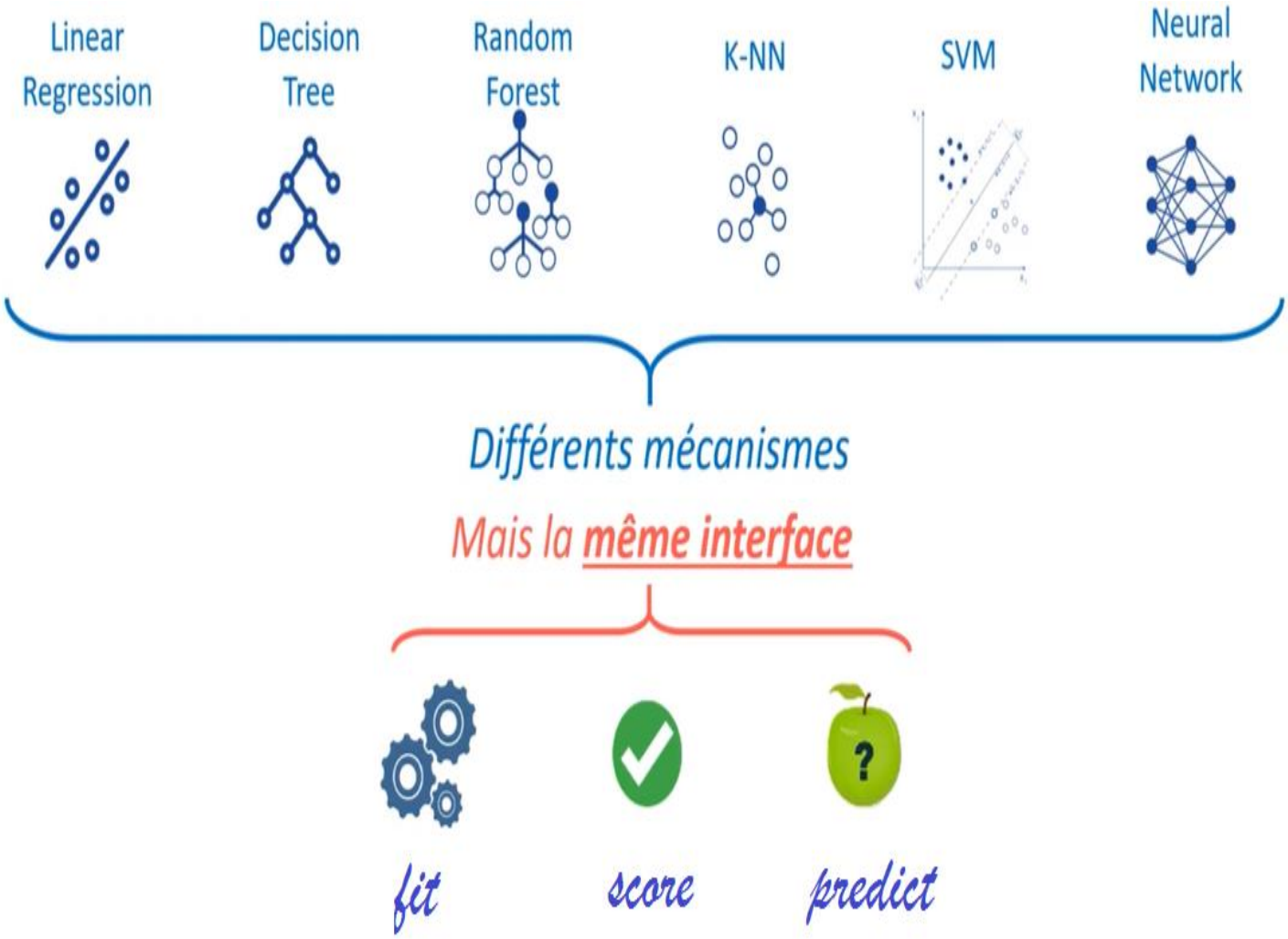
```
data.iloc[0:2,0:2]
```

	pclass	survived
0	1	1
1	1	1

```
data.loc[0:2, ['age', 'sex']]
```

	age	sex
0	29.0000	female
1	0.9167	male
2	2.0000	female

SKLearn :Toujours 3 Methodes



Exemple :RL

1. **Sélectionner** un **estimateur** et préciser ses **hyperparamètres :**

   model = ***LinearRegression(.....)***

2. **Entrainer** le modèle sur les données X, y *(divisées en 2 tableaux **Numpy**)*

   *model.**fit(X, y)***

3. **Évaluer** le modèle

   *model.**score(X, y)***

4. **Utiliser** le modèle

   *model.**predict(X)***

Linear Regression

```
model = LinearRegression()
model.fit(X, y)
model.score(X, y)
model.predict(X)
```

PYTHON FOR DATA SCIENCE CHEAT SHEET

Python Scikit-Learn

Introduction

Scikit-learn:"sklearn" is a machine learning library for the Python programming language. Simple and efficient tool for data mining, Data analysis and Machine Learning.

Importing Convention - import sklearn

Preprocessing

Data Loading

- Using NumPy:

```
>>>import numpy as np
>>>a=np.array([(1,2,3,4),(7,8,9,10)],dtype=int)
>>>data = np.loadtxt('file_name.csv',
   delimiter=',')
```

- Using Pandas:

```
>>>import pandas as pd
>>>df=pd.read_csv('file_name.csv',header=0)
```

Train-Test Data

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

>>> X_train, X_test, y_train, y_test =
train_test_split(X,y,random_state=0)
```

Data Preparation

- Standardization

```
>>>from sklearn.preprocessing import
StandardScaler
>>>get_names = df.columns
>>>scaler =
preprocessing.StandardScaler()
>>>scaled_df = scaler.fit_transform(df)
>>>scaled_df =
pd.DataFrame(scaled_df,
columns=get_names)m
```

- Normalization

```
>>>from sklearn.preprocessing import
   Normalizer
>>>pd.read_csv("File_name.csv")
 >>>x_array = np.array(df['Column1'])
#Normalize  Column1
>>>normalized_X =
preprocessing.normalize([x_array])
```

Working On Model

Model Choosing

Supervised Learning Estimator:

- Linear Regression:

```
>>> from sklearn.linear_model import
LinearRegression
>>> new_lr=
LinearRegression(normalize=True)
```

- Support Vector Machine:

```
>>> from sklearn.svm import SVC
>>> new_svc = SVC(kernel='linear')
```

- Naive Bayes:

```
>>> from sklearn.naive_bayes import
GaussianNB
>>> new_gnb = GaussianNB()
```

- KNN:

```
>>> from sklearn import neighbors
>>>
knn=neighbors.KNeighborsClassifier(n_ne
ighbors=1)
```

Unsupervised Learning Estimator:

- Principal Component Analysis (PCA):

```
>>> from sklearn.decomposition import
PCA
>>> new_pca=PCA(n_components=0.95)
```

- K Means:

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=5,
random_state=0)
```

Train-Test Data

Supervised:

```
>>>new_lr.fit(X,y)
>>> knn.fit(X_train, y_train)
>>>new_svc.fit(X_train, y_train)
```

Unsupervised :

```
>>> k_means.fit(X_train)
>>> pca_model_fit =
new_pca.fit_transform(X_train)
```

Post-Processing

Prediction

Supervised:

```
>>> y_predict =
new_svc.predict(np.random.random((3,5)))
>>> y_predict = new_lr.predict(X_test)
>>> y_predict = knn.predict_proba(X_test)
```

Unsupervised:

```
>>> y_pred = k_means.predict(X_test)
```

Model Tuning

Grid Search:

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3), "metric":
        ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization:

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5), "weights":
["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
param_distributions=params, cv=4, n_iter=8, random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

Evaluate Performance

Classification:

1. Confusion Matrix:

```
>>> from sklearn.metrics import
      confusion_matrix
>>> print(confusion_matrix(y_test,
      y_pred))
```

2. Accuracy Score:

```
>>> knn.score(X_test, y_test)
>>> from sklearn.metrics import
      accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Regression:

1. Mean Absolute Error:

```
>>> from sklearn.metrics import mean_absolute_error

>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_predict)
```

2. Mean Squared Error:

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_predict)
```

3. R^2 Score :

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_predict)
```

Clustering:

1. Homogeneity:

```
>>> from sklearn.metrics import
homogeneity_score
>>> homogeneity_score(y_true,
y_predict)
```

2. V-measure:

```
>>> from sklearn.metrics import
v_measure_score
>>> metrics.v_measure_score(y_true,
y_predict)
```

Cross-validation:

```
>>> from
sklearn.cross_validation
import cross_val_score
>>>
print(cross_val_score(knn,
X_train, y_train, cv=4))
>>>
print(cross_val_score(new_
lr, X, y, cv=2))
```

FURTHERMORE:
Python for Data Science Certification Training Course

MatplotLib

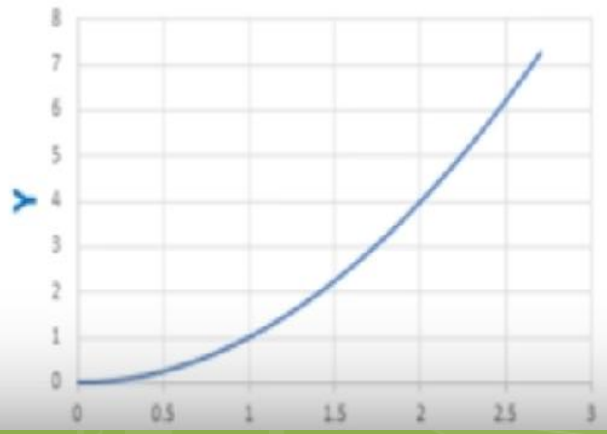
Fonction

```
plt.plot(x, y)
plt.show(x, y)
```

OOP

```
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

Même résultat



MatplotLib

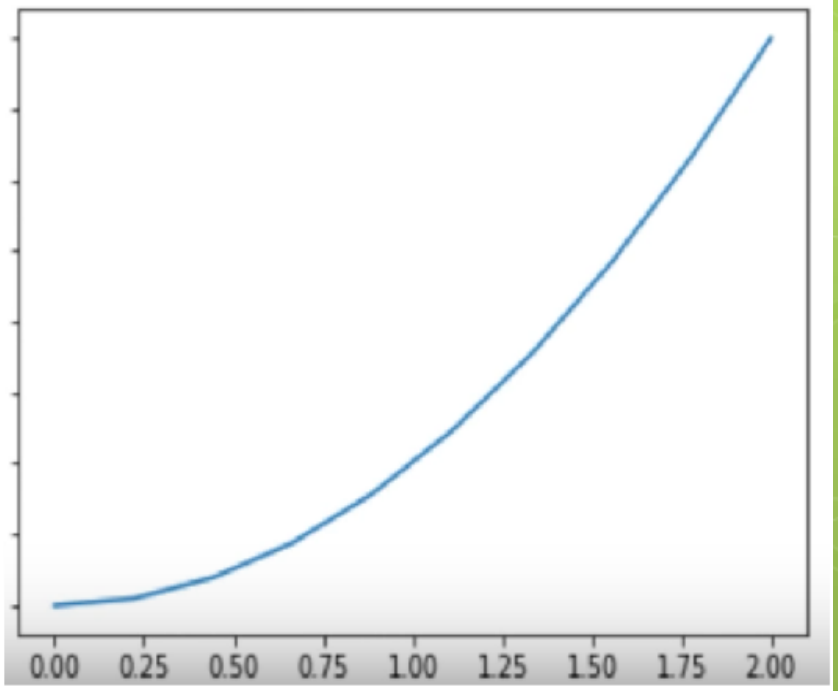
plot(x, y, label=..., lw=..., ls=..., c=...)

- label=... : Nom de la courbe
- lw=... : Épaisseur du trait
- ls=... : Style du trait
- c=... : Couleur du trait

X et Y : dimensions égales !

```
import matplotlib.pyplot as plt
```

```
plt.plot(x, y)
plt.show()
```



MatplotLib

plt.figure() <- Début de la figure

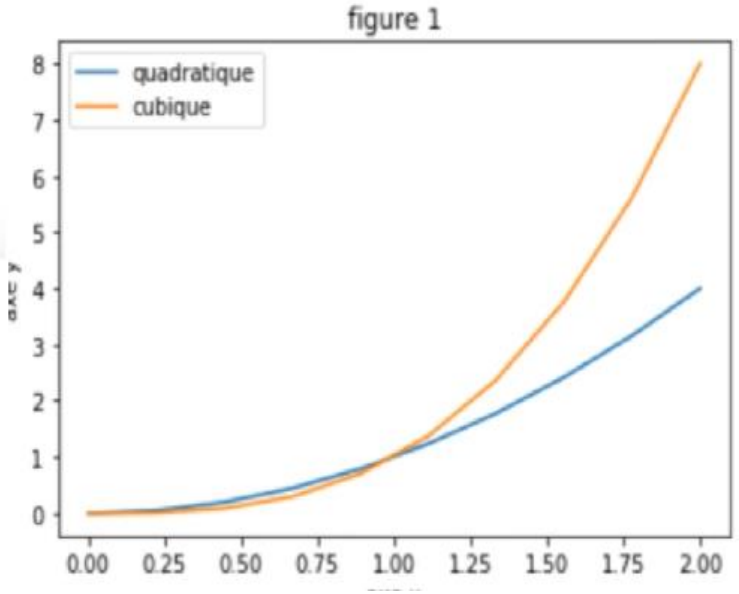
plt.plot(..., ...)
plt.plot(..., ...)
plt.xlabel('texte')
plt.title('texte')
plt.legend()
} Contenu

plt.show() <- Affiche la figure

plt.savefig('text.png')

```
import matplotlib.pyplot as plt
```

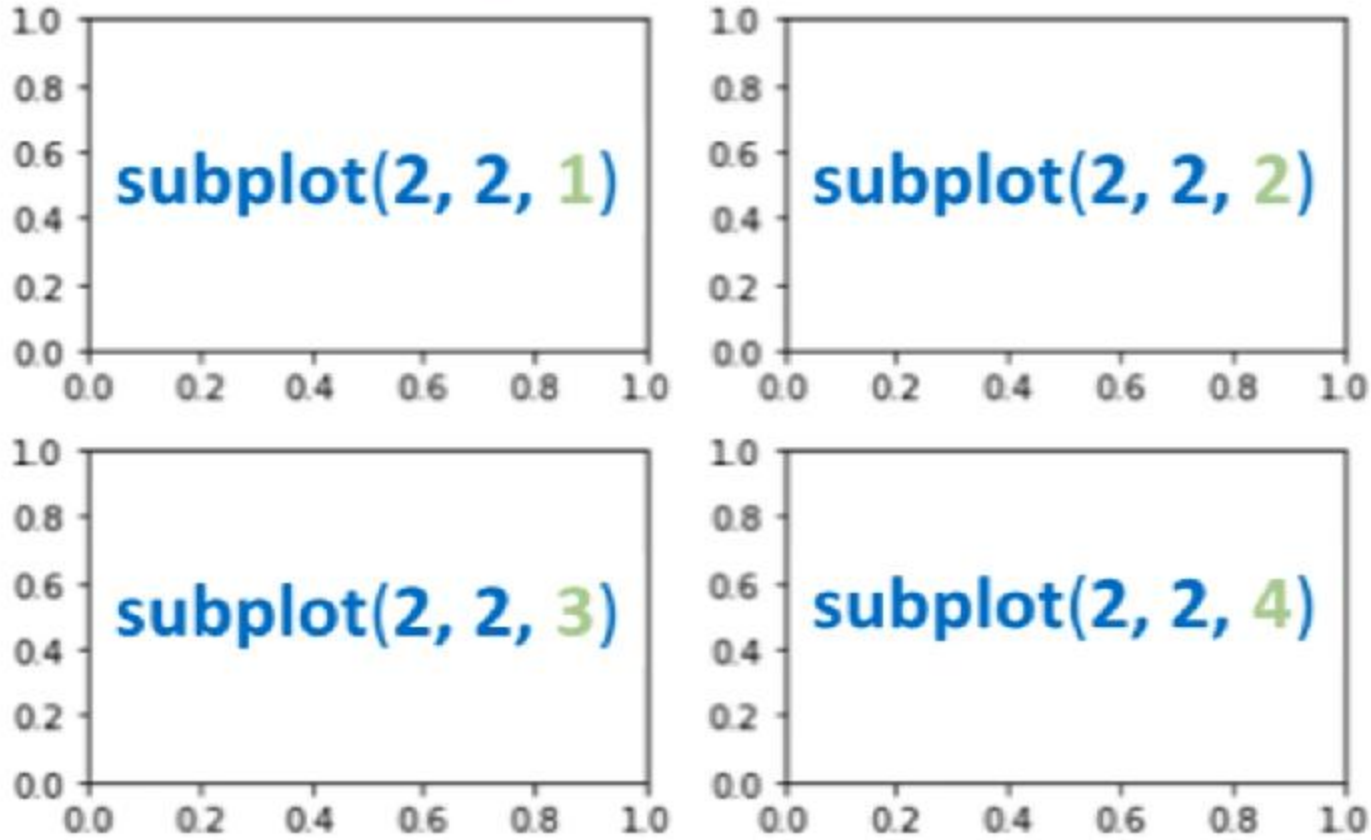
```
plt.figure()
plt.plot(x, y, label='quadratique')
plt.plot(x, x**3, label='cubique')
plt.title('figure 1')
plt.xlabel('axe x')
plt.ylabel('axe y')
plt.legend()
plt.show()
plt.savefig('figure.png')
```



MatplotLib

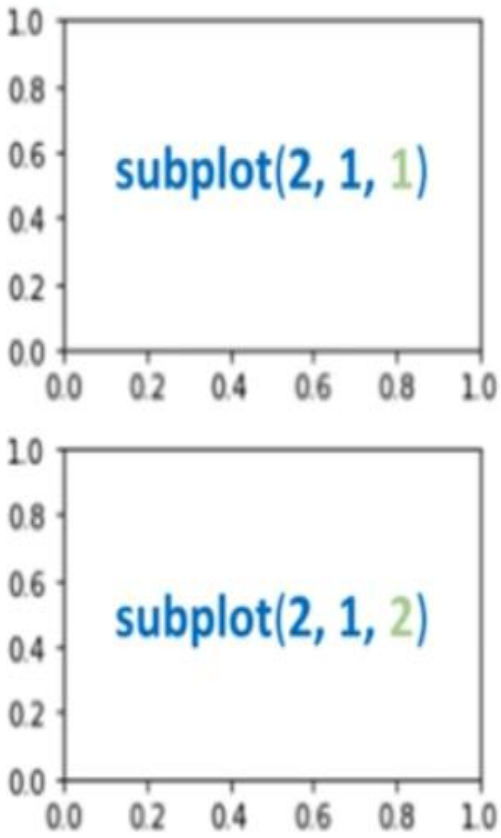
Une **grille** de graphiques :

plt.**subplot**(**lignes**, **colonnes**, position)



MatplotLib

Un Exemple:




```
plt.subplot(2, 1, 1)
plt.plot(x, y, c='red')
plt.subplot(2, 1, 2)
plt.plot(x, y, c='blue')
```

```
[<matplotlib.lines.Line2D at 0x1360865b358>]
```