



Python pour l'apprentissage automatique avec Scikit-Learn

Laurent Risser

Institut de Mathématiques de Toulouse (IMT)

lrissier@math.univ-toulouse.fr

Plan:

1. Généralités sur Python
2. Écosystème Python pour l'analyse de données (non exhaustif !!!)
 - Scipy
 - NetworkX
 - Pandas
 - TensorFlow, Caffe, Pytorch, ...
 - Scikit-learn
3. Sous-Modules de Scikit-learn

1) Généralités sur Python

En bref :


- Libre
- Langage généraliste mais très compact
- Nombreuses extensions et bibliothèques disponibles
- Orienté objet (mais on peut en faire abstraction)

Les grandes forces de Python sont la taille et le dynamisme de sa communauté

Découvrir Python :

- www.docs.python.org
- <http://apprendre-python.com>
- ... mon cours...
- « Ask google »

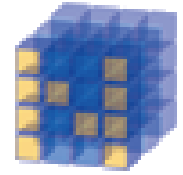
Python pour l'apprentissage :

- Moins spécialisé que R...
- ... mais il existe un **écosystème** de modules Python pour l'analyse (franchement) avancée ...
- ... tout en profitant d'autres modules de lecture et traitement de données variées 

1) Généralités sur Python

Numpy :

- LE module Python pour la programmation numérique
- Contient le *data type* array et des fonctions pour le manipuler
- Tous les modules Python d'analyse de données utilisent des `numpy.array`



```
import numpy as np

my_1D_array = np.array([4,3,2],dtype=np.float128)
print(my_1D_array)

→ [4.0 3.0 2.0]

toto=np.random.randn(2,3)
print(toto)

→ [[-0.37716423 -0.53787731 -0.73762654]
    [-2.90642102 -1.0586924  0.20380006]]

toto.mean()

→ -0.902330239148

...
```

2) Écosystème – Scipy

Scipy est une bibliothèque pour les mathématiques, la science, et l'ingénierie



On peut voir ce que Scipy peut faire à la page <http://docs.scipy.org/doc/scipy/reference/> :

- Integration (scipy.integrate)
- Optimization and root finding (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Sparse Eigenvalue Problems with ARPACK
- Compressed Sparse Graph Routines scipy.sparse.csgraph
- Spatial data structures and algorithms (scipy.spatial)
- **Statistics (scipy.stats)**
- Multi-dimensional image processing (scipy.ndimage)
- Clustering package (scipy.cluster)
- Orthogonal distance regression (scipy.odr)
- Sparse matrices (scipy.sparse)
- Sparse linear algebra (scipy.sparse.linalg)
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- **File inputs/outputs (scipy.io)**
- ... et bien d'autres encore

```
import scipy.stats

rvs1 = scipy.stats.norm.rvs(loc=5, scale=10, size=500)
rvs2 = scipy.stats.norm.rvs(loc=5, scale=10, size=500)
rvs3 = scipy.stats.norm.rvs(loc=8, scale=10, size=500)

#t-test (returns: calculated t-statistic / two-tailed p-value)
scipy.stats.ttest_ind(rvs1, rvs2)
→ (-0.5489, 0.5831)

scipy.stats.ttest_ind(rvs1, rvs3)
→ (-4.533, 6.507e-6)

#Kolmogorov-Smirnov test (returns: KS statistic / two-tailed p-value)
scipy.stats.ks_2samp(rvs1, rvs2)
→ (0.0259, 0.9954)

scipy.stats.ks_2samp(rvs1, rvs3)
→ (0.1139, 0.0027)
```

2) Écosystème – NetworkX


The screenshot shows the NetworkX website homepage in a browser. The browser's address bar displays "https://networkx.github.io". The page features a navigation bar with links for "NetworkX Home", "Documentation", "Download", and "Developer (Github)". The main content area is titled "High-productivity software for complex networks" and includes a descriptive paragraph, a network diagram, and links for "Documentation", "Examples", and "Reference". A "Features" section lists various capabilities of the software. On the right side, there are sections for "Versions", "Latest Release" (networkx-1.11, 30 January 2016), "Development" (2.0dev), and "Contact".

NetworkX

[NetworkX Home](#) | [Documentation](#) | [Download](#) | [Developer \(Github\)](#)

High-productivity software for complex networks

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.



[Documentation](#)
all documentation

[Examples](#)
using the library

[Reference](#)
all functions and methods

Features

- Python language data structures for graphs, digraphs, and multigraphs.
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g. text, images, XML records)
- Edges can hold arbitrary data (e.g. weights, time-series)
- Open source [BSD license](#)
- Well tested: more than 1800 unit tests, >90% code coverage
- Additional benefits from Python: fast prototyping, easy to teach, multi-

Versions

Latest Release


networkx-1.11
30 January 2016
[downloads](#) | [docs](#) | [pdf](#)

Development

2.0dev
[github](#) | [docs](#) | [pdf](#)
build passing
coverage 94%

Contact

[Mailing list](#)
[Issue tracker](#)



2) Écosystème – NetworkX – Un petit exemple

```
import networkx as nx
import csv
```

```
RefGraph=nx.Graph()
```

```
with open(EdgesCsvFile_or_List,'r') as csvfile:
```

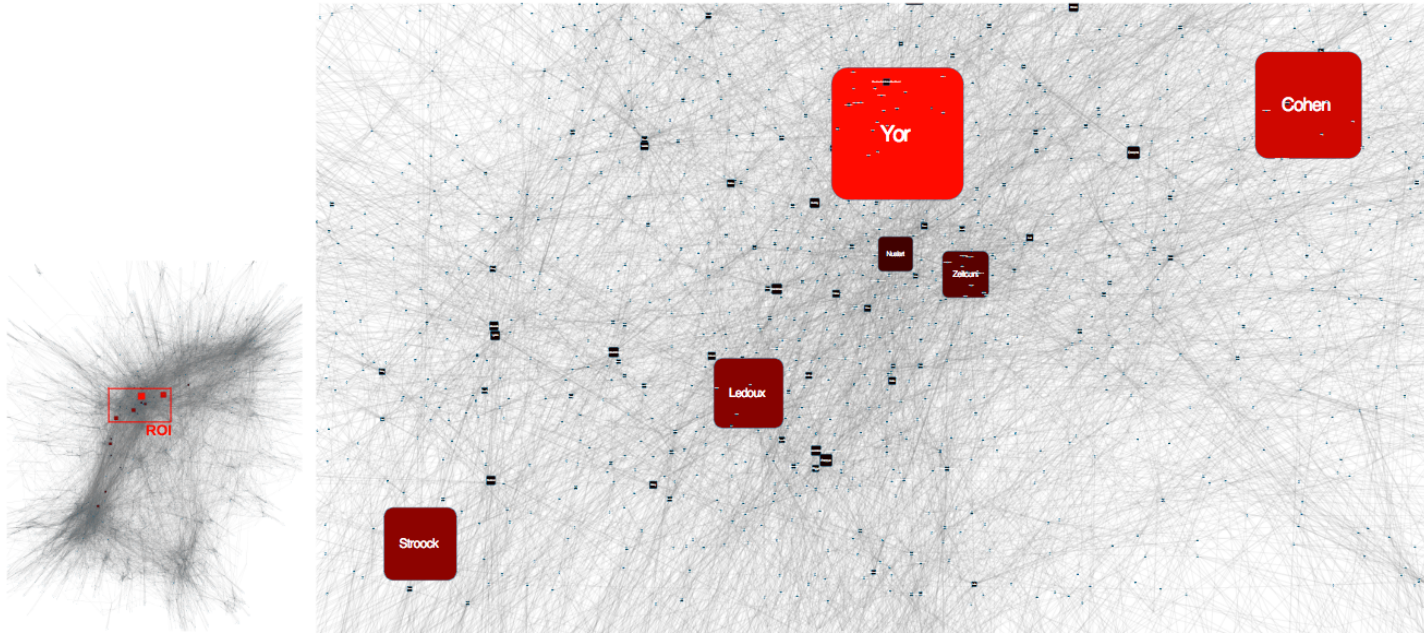
```
    data = csv.reader(csvfile, delimiter=' ', quotechar='%')
```

```
    for row in data:
```

```
        RefGraph.add_weighted_edges_from([(row[0],row[1],float(row[2]))])
```

```
T=nx.minimum_spanning_tree(RefGraph)
```

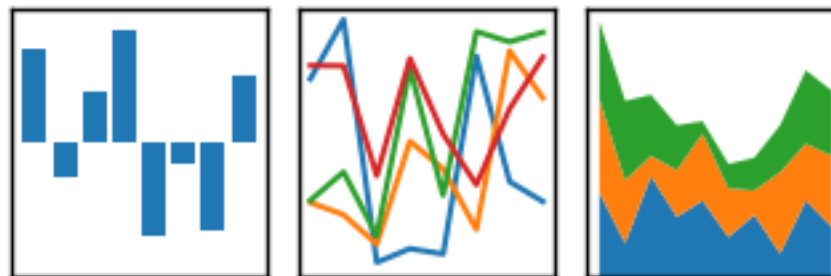
Gestion efficace de graphes relativement gros (10^5 nœuds) :





pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Pandas : Librairie Python pour **extraire**, **préparer** et éventuellement analyser, des données

- Contient les classes Series et DataFrame (tables de données)
- Lecture des fichiers .csv, xls, hdf5, HTML, XML, JSON, MongoDB, SQL, ...
- Sélection/Suppression/Ajout de lignes et de colonnes, fusion de DataFrames
- Gestion de données manquantes et aberrantes
- Génération de nombres aléatoires
- Tests statistiques élémentaires
- Fonctions graphiques
- Gestion des dates
- Gestion de très grosses données (via HDF5)
- ...

2) Écosystème – Pandas – Un petit exemple

```
import pandas as pd

data = {"state": ["Ohio", "Ohio", "Ohio", "Nevada"],
        "year": [2000, 2001, 2002, 2001],
        "pop": [1.5, 1.7, 3.6, 2.4]}

frame = pd.DataFrame(data, columns=["year", "state", "pop"])
```

print frame

```
→      year  state  pop
→  0  2000  Ohio   1.5
→  1  2001  Ohio   1.7
→  2  2002  Ohio   3.6
→  3  2001  Nevada  2.4
```

```
frame2=pd.DataFrame(data,
                    columns=["year", "state", "pop", "debt"],
                    index=["one", "two", "three", "four"])
```

print frame2

```
→      year  state  pop  debt
→  one  2000  Ohio   1.5  NaN
→  two  2001  Ohio   1.7  NaN
→  three 2002  Ohio   3.6  NaN
→  four  2001  Nevada  2.4  NaN
```

...

...

```
frame["state"]
```

```
→ 0  Ohio
→ 1  Ohio
→ 2  Ohio
→ 3  Nevada
→ 4  Nevada
```

```
frame2["debt"] = 16.5
```

```
frame2.set_value('four', 'debt', 10)
```

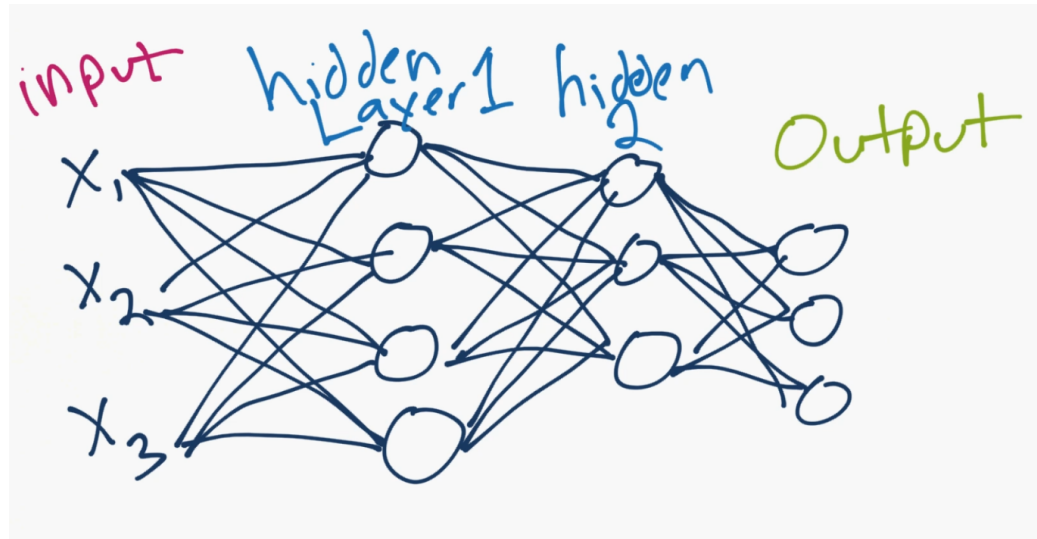
print frame2

```
→      year  state  pop  debt
→  one  2000  Ohio   1.5  16.5
→  two  2001  Ohio   1.7  16.5
→  three 2002  Ohio   3.6  16.5
→  four  2001  Nevada  2.4  10.0
```

→ Utilisation massive des dataframes comme en R !



2) Écosystème – TensorFlow, Caffe, PyTorch, ...



Deep-learning sous Python

<https://pythonprogramming.net/neural-networks-machine-learning-tutorial/>

PYTORCH

Caffe



theano

+ Apprentissage supervisé et prédiction avec XGBoost : **XGBoost**

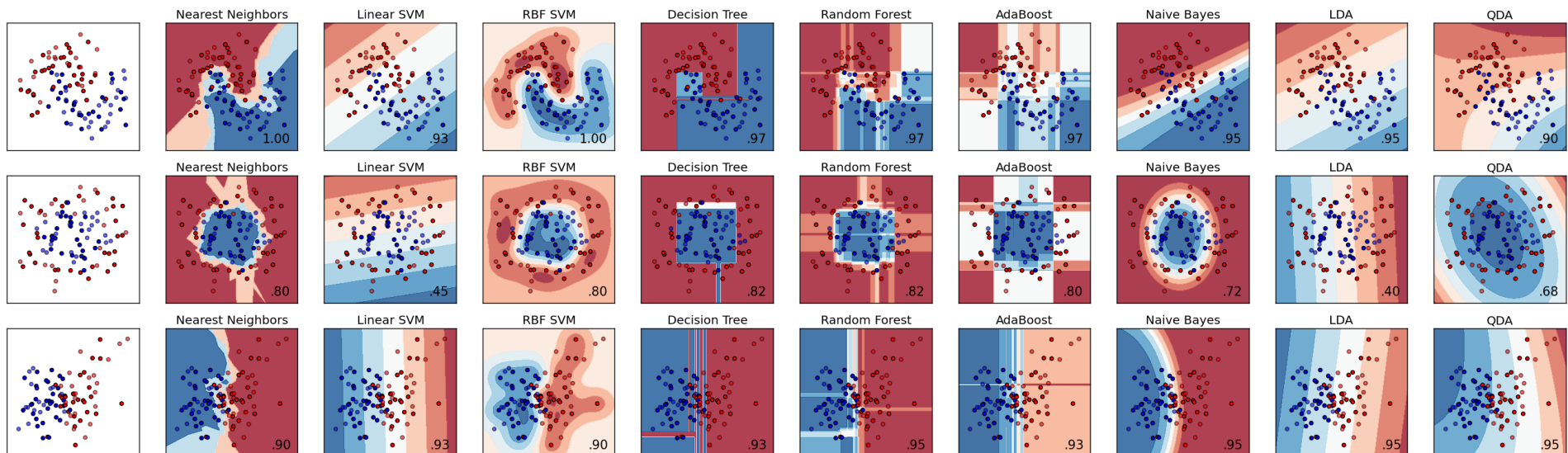
2) Écosystème – Scikit-learn

Scikit-learn est la bibliothèque de référence pour l'analyse de données sous Python.

→ Utilise massivement NumPy, Matplotlib et SciPy.

On peut se faire une idée de ce que Scikit-learn peut faire à <http://scikit-learn.org/stable/> :

- Classification supervisée → SVM, random forest, ...
- Regression → Ridge regression, Lasso, ...
- Clustering → k-Means, spectral clustering, mean-shift, ...
- Dimensionality reduction → PCA, non-negative matrix factorization, ...
- Model selection → cross validation, parameter search, classification metrics, ...



3) Scikit-learn – Exemple en classification supervisée

SVM (http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```

```
# import some data to play with
iris = datasets.load_iris()
```

```
print(iris.data)
```

```
→ [[ 5.1  3.5  1.4  0.2]
    [ 4.9  3.   1.4  0.2]
    ...
    [ 6.2  3.4  5.4  2.3]
    [ 5.9  3.   5.1  1.8]]
```

**150 observations
en dimension 4**

```
print(iris.target)
```

```
→ [0 0 0 0 0 ... 1 1 1 ... 2 2 2]
```

150 labels dans {0,1,2}

3) Scikit-learn – Exemple en classification supervisée

SVM (http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# import some data to play with
iris = datasets.load_iris()

print(iris.data)
→ [[ 5.1  3.5  1.4  0.2]
    [ 4.9  3.   1.4  0.2]
    ...
    [ 6.2  3.4  5.4  2.3]
    [ 5.9  3.   5.1  1.8]]

print(iris.target)
→ [0 0 0 0 0 ... 1 1 1 ... 2 2 2]

# we create an instance of SVM and fit out data.
X = iris.data[:, :2]
y = iris.target

svm_inst=svm.SVC(kernel='linear', C=1.0)
svc = svm_inst.fit(X, y)
```

```
# create a mesh of values to test
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                    np.arange(y_min, y_max, 0.02))

#predict the class of the points on the mesh
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

#plot the result
...
```

3) Scikit-learn – Exemple en classification supervisée

SVM (http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```

```
# import some data to play with
iris = datasets.load_iris()
```

```
print(iris.data)
→ [[ 5.1  3.5  1.4  0.2]
    [ 4.9  3.  1.4  0.2]
    ...
    [ 6.2  3.4  5.4  2.3]
    [ 5.9  3.  5.1  1.8]]
```

```
print(iris.target)
→ [0 0 0 0 ... 1 1 1 ... 2 2 2]
```

```
# we create an instance of SVM and fit out data.
```

```
X = iris.data[:, :2]
y = iris.target
```

```
svm_inst=svm.SVC(kernel='linear', C=1.0)
svc = svm_inst.fit(X, y)
```

```
# create a mesh of values to test
```

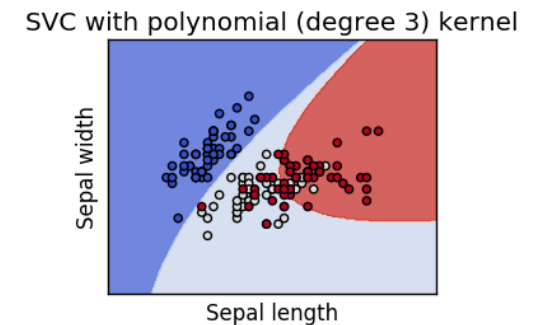
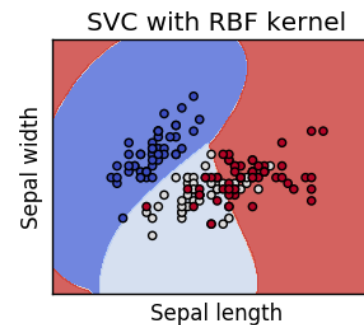
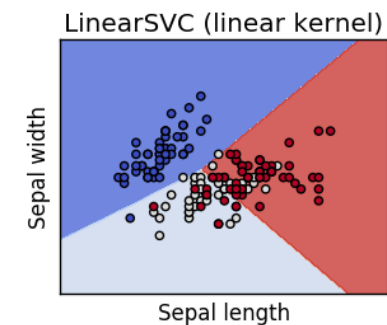
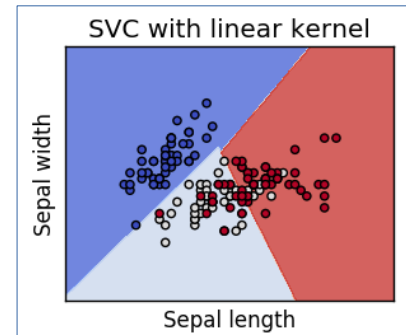
```
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                    np.arange(y_min, y_max, 0.02))
```

```
#predict the class of the points on the mesh
```

```
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
#plot the result
```

```
...
```



3) Scikit-learn – Exemple en classification non-supervisée (clustering)

Reduction de dimension et K-means (http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

np.random.seed(42)

digits = load_digits()
print(digits.data)
→ [[ 0.  0.  5. ...,  0.  0.  0.]
    [ 0.  0.  0. ..., 10.  0.  0.]
    ...,
    [ 0.  0.  1. ...,  6.  0.  0.]
    [ 0.  0. 10. ..., 12.  1.  0.]]
```

**1797 observations
en dimension 64**

3) Scikit-learn – Exemple en classification non-supervisée (clustering)

Reduction de dimension et K-means (http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
```

```
np.random.seed(42)
```

```
digits = load_digits()
```

```
print(digits.data)
```

```
→ [[ 0.  0.  5. ...,  0.  0.  0.]
    [ 0.  0.  0. ..., 10.  0.  0.]
    ...,
    [ 0.  0.  1. ...,  6.  0.  0.]
    [ 0.  0. 10. ..., 12.  1.  0.]]
```

**1797 observations
en dimension 64**

```
data = scale(digits.data) #centre/réduit les donnees
```

```
red_data=PCA(n_components=2).fit_transform(data)
```

```
print(red_data)
```

```
→ [[ 1.91422151 -0.95454005]
    [ 0.58898326  0.92462171]
    ...,
    [ 1.07605978 -0.38092876]
    [-1.25771163 -2.22756272]]
```

**1797 observations
en dimension 2**

3) Scikit-learn – Exemple en classification non-supervisée (clustering)

Reduction de dimension et K-means (http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

np.random.seed(42)

digits = load_digits()
print(digits.data)
→ [[ 0.  0.  5. ...,  0.  0.  0.]
    [ 0.  0.  0. ..., 10.  0.  0.]
    ...,
    [ 0.  0.  1. ...,  6.  0.  0.]
    [ 0.  0. 10. ..., 12.  1.  0.]]

data = scale(digits.data) #centre/réduit les donnees
red_data=PCA(n_components=2).fit_transform(data)
print(red_data)
→ [[ 1.91422151 -0.95454005]
    [ 0.58898326  0.92462171]
    ...,
    [ 1.07605978 -0.38092876]
    [-1.25771163 -2.22756272]]
```

```
#lance le clustering
kmeans = KMeans(init='k-means++', n_clusters=9, n_init=9)
kmeans.fit(reduced_data)

#représente le résultat
Z = kmeans.predict(reduced_data)

plt.scatter(reduced_data[:, 0],reduced_data[:, 1],c=Z)
plt.show()
```

3) Scikit-learn – Exemple en classification non-supervisée (clustering)

Reduction de dimension et K-means (http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale

np.random.seed(42)

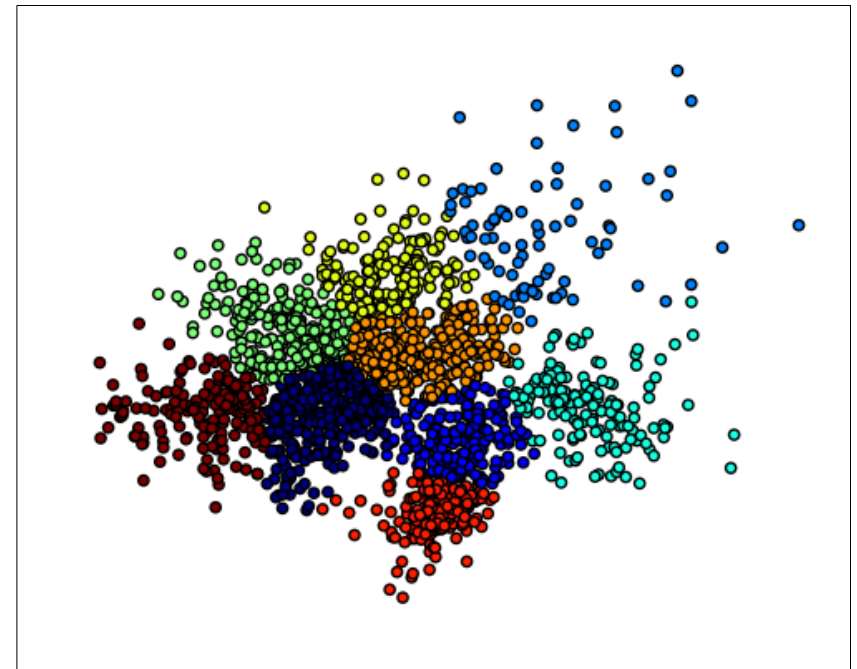
digits = load_digits()
print(digits.data)
→ [[ 0.  0.  5. ...,  0.  0.  0.]
    [ 0.  0.  0. ..., 10.  0.  0.]
    ...,
    [ 0.  0.  1. ...,  6.  0.  0.]
    [ 0.  0. 10. ..., 12.  1.  0.]]

data = scale(digits.data) #centre/réduit les données
red_data=PCA(n_components=2).fit_transform(data)
print(red_data)
→ [[ 1.91422151 -0.95454005]
    [ 0.58898326  0.92462171]
    ...,
    [ 1.07605978 -0.38092876]
    [-1.25771163 -2.22756272]]
```

```
#lance le clustering
kmeans = KMeans(init='k-means++', n_clusters=9, n_init=9)
kmeans.fit(reduced_data)

#représente le résultat
Z = kmeans.predict(reduced_data)

plt.scatter(reduced_data[:, 0],reduced_data[:, 1],c=Z)
plt.show()
```



3) Scikit-learn – Exemple en régression

Régression (http://scikit-learn.org/stable/_downloads/plot_isotonic_regression.py)

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.collections import LineCollection
from sklearn.linear_model import LinearRegression
from sklearn.isotonic import IsotonicRegression
from sklearn.utils import check_random_state
```

```
n = 100
x = np.arange(n)
rs = check_random_state(0)
y = rs.randint(-50, 50, size=(n,)) + 50. * np.log(1 +
np.arange(n))
```

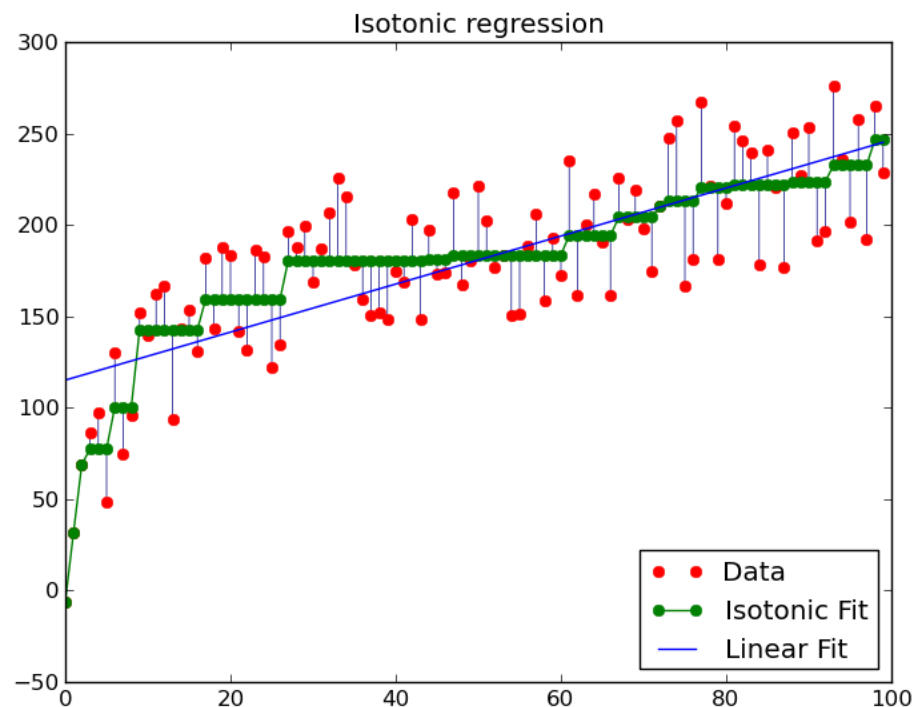
```
# Fit IsotonicRegression and LinearRegression models
```

```
ir = IsotonicRegression() ←
y_ = ir.fit_transform(x, y)
lr = LinearRegression() ←
lr.fit(x[:, np.newaxis], y)
```

```
# plot result
```

```
segments = [[[i, y[i]], [i, y_[i]]] for i in range(n)]
lc = LineCollection(segments, zorder=0)
lc.set_array(np.ones(len(y)))
lc.set_linewidths(0.5 * np.ones(n))
```

```
...
```



JOUEZ LES EXERCICES PROPOSES
ET REPONDEZ AUX QUESTIONS

Ensuite vous pouvez jouer les
notebooks wikistat : www.wikistat.fr