

# Statistiques avec SciPy

## Programmation Python

Ricco Rakotomalala

[http://eric.univ-lyon2.fr/~ricco/cours/cours\\_programmation\\_python.html](http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html)

- Scipy est une librairie de calcul scientifique pour Python
- Elle s'appuie sur les structures de données de NumPy (vecteurs, matrices)
- Scipy couvre de nombreux domaines

## Reference

- Clustering package (`scipy.cluster`)
- Constants (`scipy.constants`)
- Discrete Fourier transforms (`scipy.fftpack`)
- Integration and ODEs (`scipy.integrate`)
- Interpolation (`scipy.interpolate`)
- Input and output (`scipy.io`)
- Linear algebra (`scipy.linalg`)
- Miscellaneous routines (`scipy.misc`)
- Multi-dimensional image processing (`scipy.ndimage`)
- Orthogonal distance regression (`scipy.odr`)
- Optimization and root finding (`scipy.optimize`)
- Signal processing (`scipy.signal`)
- Sparse matrices (`scipy.sparse`)
- Sparse linear algebra (`scipy.sparse.linalg`)
- Compressed Sparse Graph Routines (`scipy.sparse.csgraph`)
- Spatial algorithms and data structures (`scipy.spatial`)
- Special functions (`scipy.special`)
- Statistical functions (`scipy.stats`)
- Statistical functions for masked arrays (`scipy.stats.mstats`)
- C/C++ integration (`scipy.weave`)

Nous nous intéresserons en particulier aux modules de calcul statistique.

Il n'est pas possible de tout aborder dans ce support. Pour aller plus loin, voir absolument le manuel de référence (utilisé pour préparer ce diaporama).

<http://docs.scipy.org/doc/scipy/reference/>

Traitement d'un seul vecteur de données

Statistique descriptive, test d'adéquation, test de conformité, etc.

# **STATISTIQUES À 1 ÉCHANTILLON**

**StatSci.org** / [Home](#)

[OzDASL](#)

## Width to Length Ratios in Shoshoni Handicraft

Keywords: t-test

### Description

The data are width-to-length ratios of beaded rectangles used by the Shoshoni Indians of America to decorate their leather goods. One might ask whether the golden rectangle (for which the width-to-length ratio is 0.618) can be considered an aesthetic standard for the Shoshonis just as it was for the Greeks and the Egyptians.

### Download

[Data File](#) (tab-delimited text)

### Source

Dubois, Cora (ed.) (1960). *Lowie's Selected Papers in Anthropology*. University of California Press, Berkeley.

Larsen, R.J., and Marx, M.L., *An Introduction to Mathematical Statistics and Its Applications* 2nd Edition. Prentice-Hall, 1986. Case Study 1.2.2.



Après analyse des points extrêmes (cf. Iglzawicz & Hoaglin [Modified Z-score](#)), 2 observations ont été retirées (solution toujours discutable, mais bon... faisons au plus simple, le sujet est SciPy)



```
d = np.array([0.553,0.57,0.576,0.601,0.606,0.606,0.609,0.611,0.615,0.628,0.654,0.662,0.668,0.67,0.672,0.69,0.693,0.749])
```

# Statistiques descriptives

```
import numpy as np
import scipy.stats as stat #noter l'utilisation de l'alias stat pour accéder au sous module stats de SciPy

#objet statistique descriptives
stat_des = stat.describe(d)
print(stat_des)
DescribeResult(nobs=18, minmax=(0.55300000000000005, 0.749), mean=0.6351666666666666,
variance=0.0025368529411764714, skewness=0.38763289979752136, kurtosis=-0.35873690487519205)

#stat_des est de type "namedtuple" du module collections - voir les différentes manières d'accéder aux champs

#par indice
print(stat_des[0]) # 18, le 1er champ (indice 0) est nobs

#par nom possible aussi
print(stat_des.nobs) # 18

#possibilité d'éclater les résultats avec une affectation multiple
n,mm,m,v,sk,kt = stat.describe(d)
print(n,m) # 18 0.635166, accéder au nombre d'obs. et à la moyenne

#médiane de NumPy
print(np.median(d)) # 0.6215

#fonction de répartition empirique
print(stat.percentileofscore(d,0.6215)) # 50.0, la moitié des obs. ont une valeur inf. à 0.6215
```

Objectif : obtenir les valeurs des quantiles et des fonctions de répartition pour différentes lois statistiques utilisées pour l'inférence.

#loi normale centrée et réduite

```
print(stat.norm.ppf(0.95,loc=0,scale=1)) # quantile d'ordre 0.95 de la loi normale CR = 1.64485  
print(stat.norm.cdf(1.96,loc=0,scale=1)) # 0.975
```

#loi de Student - ddl = 30

```
print(stat.t.ppf(0.95,df=30)) # 1.6972  
print(stat.t.cdf(1.96,df=30)) # 0.9703
```

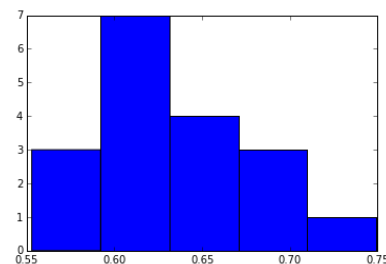
#loi du khi-2 - ddl = 10

```
print(stat.chi.ppf(0.95,df=10)) # 4.2787  
print(stat.chi.cdf(4.84,df=10)) # 0.9907
```

#loi de Fisher, ddl numérateur = 1, ddl dénominateur = 30

```
print(stat.f.ppf(0.95,dfn=1,dfd=30)) # 4.1709  
print(stat.f.cdf(3.48,dfn=1,dfd=30)) # 0.9281
```

Objectif : vérifier que la distribution est compatible avec la loi Normale (Gauss)



Histogramme de fréquence  
(cf. le module matplotlib)

## #test de normalité d'Agostino

```
ag = stat.normaltest(d) # message d'avertissement, n est trop faible pour un test fiable  
print(ag) # (0.714, 0.699), statistique de test et p-value (si p-value <  $\alpha$ , rejet de l'hyp. de normalité)
```

## #test de Normalité Shapiro-Wilks

```
sp = stat.shapiro(d)  
print(sp) # (0.961, 0.628), statistique et p-value
```

## #test d'adéquation d'Anderson-Darling

```
ad = stat.anderson(d,dist="norm") # test possible pour autre loi que « norm »  
print(ad) # (0.3403, array([ 0.503, 0.573, 0.687, 0.802, 0.954]), array([ 15., 10., 5., 2.5, 1. ])) → stat  
de test, seuils critiques pour chaque niveau de risque, on constate ici que la p-value est sup. à 15%
```



Pour plus d'information sur ces techniques, voir : R. Rakotomalala, « [Tests de normalité – Techniques empiriques et tests statistiques](#) », Version 2.0, 2008.

Objectif : disposer d'un générateur de nombres aléatoires permet de réaliser des simulations ou d'accéder à des techniques basées sur le ré-échantillonnage (bootstrap, etc.). Tant SciPy que NumPy proposent des outils à cet effet.

**#génération de valeurs aléatoires - loi normale (0, 1)**

```
alea1 = stat.norm.rvs(loc=0,scale=1,size=30)
```

```
print(stat.normaltest(alea1)) # (2.16, 0.338), compatible avec la loi normale (heureusement !)
```

**#génération - loi exponentielle**

```
alea2 = stat.expon.rvs(size=30)
```

```
print(stat.normaltest(alea2)) # (17.62, 0.00015), non compatible (bien sûr)
```

**#Numpy aussi propose un générateur**

```
alea3 = np.random.normal(loc=0,scale=1,size=30)
```

```
print(stat.normaltest(alea3)) # (2.41, 0.299), compatible
```

**#échantillonnage de m obs. parmi n**

```
d1 = np.random.choice(d,size=5,replace=False) #sans remise
```

```
print(d1) # (0.69 0.628 0.606 0.662 0.668)
```

```
d2 = np.random.choice(d,size=5,replace=True) #avec remise
```

```
print(d2) # (0.654 0.67 0.628 0.654 0.609), j'ai un peu forcé la chance...
```



<u>Objectif :</u>	$H_0 : \mu = 0.618$
	$H_1 : \mu \neq 0.618$

#test de conformité de la moyenne

```
print(stat.ttest_1samp(d,popmean=0.618))
```

# (1.446, 0.166), stat. de test et p-value, p-value <  $\alpha$ , rejet de  $H_0$

\*\*\* si l'on s'amuse à détailler les calculs \*\*\*

#moyenne

```
m = np.mean(d) # 0.6352
```

#écart-type – ddof = 1 pour effectuer le calcul :  $1/(n-1)$

```
sigma = np.std(d,ddof=1) # 0.0504
```

#stat. de test t

```
import math
```

```
t = (m - 0.618)/(sigma/math.sqrt(d.size))
```

```
print(t) # 1.446, on retrouve bien la bonne valeur de la stat de test
```

#p-value – c'est un test bilatéral

#t distribution de Student, cdf() : cumulative distribution function

```
p = 2.0 * (1.0 - stat.t.cdf(math.fabs(t),d.size-1))
```

```
print(p) # 0.166, et la bonne p-value
```

Traitement de deux vecteurs de données

Comparaison de populations, mesures d'association, etc.

## **STATISTIQUES À 2 VECTEURS**

**Story Name:** Improving Reading Ability

**Story Topics:** [Education](#)

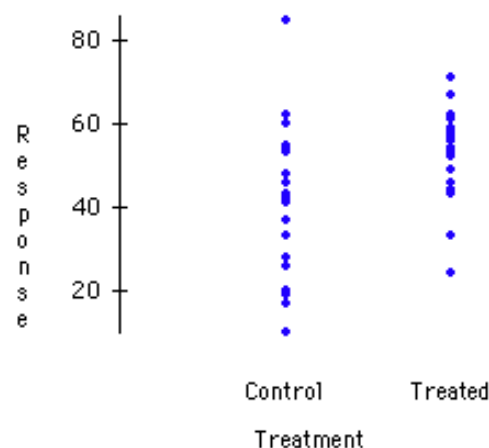
**Datafile Name:** [DRP Scores](#)

**Methods:** [Two sample t-test](#) , [Summary statistics](#)

**Abstract:** An educator conducted an experiment to test whether new directed reading activities in the classroom will help elementary school pupils improve some aspects of their reading ability. She arranged for a third grade class of 21 students to follow these activities for an 8-week period. A control classroom of 23 third graders followed the same curriculum without the activities. At the end of the 8 weeks, all students took a Degree of Reading Power (DRP) test, which measures the aspects of reading ability that the treatment is designed to improve.

Summary statistics on the two groups of children show that the average score of the treatment class was almost ten points higher than the average of the control class. A two-sample t-test is appropriate for testing whether this difference is statistically significant. The t-statistic is 2.31, which is significant at the .05 level.

**Image:** Boxplots of test score for treatment and control classes.



```
import numpy as np
import scipy.stats as stat

#treated – valeurs pour échantillon des individus ayant suivi le traitement
dt = np.array([24,43,58,71,43,49,61,44,67,49,53,56,59,52,62,54,57,33,46,43,57])
#control – échantillon de contrôle
dc = np.array([42,43,55,26,62,37,33,41,19,54,20,85,46,10,17,60,53,42,37,42,55,28,48])

#t-test – comparaison de param. de localisation – hyp. de variances égales
print(stat.ttest_ind(dt,dc)) # (t = 2.2665, p-value = 0.0286)

#t-test de Welch – comparaison de moyennes – hyp. de variances inégales
print(stat.ttest_ind(dt,dc,equal_var=False)) # (2.3109, 0.0264)

#test de Mann-Whitney - non paramétrique - avec correction de continuité
print(stat.mannwhitneyu(dt,dc)) # (stat. U = 135, p-value unilatérale = 0.00634)

#test de Bartlett – comparaison de paramètres d'échelle (variance)
print(stat.bartlett(dt,dc)) # (stat. = 3.8455, p-value = 0.0498)

#test de Ansari Bradley
print(stat.ansari(dt,dc)) # (stat. = 266, p-value = 0.2477)

#test de Levene
print(stat.levene(dt,dc)) # (stat. = 2.342, p-value = 0.1334)

#test de Kolmogorov-Smirnov – écart entre les fonctions de répartition empiriques
print(stat.ks_2samp(dt,dc)) # (stat. = 0.4699, p-value = 0.0099)
```

Voir R. Rakotomalala

[Comparaison de populations – Tests paramétriques](#)

[Comparaison de populations – Test non paramétriques](#)

**Story Name:** Women in the Labor Force

**Story Topics:** [Sociology](#) , [Economics](#)

**Datafile Name:** [Labor Force](#)

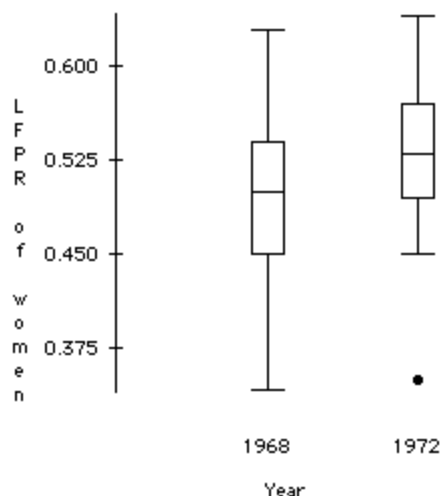
**Methods:** [Boxplot](#) , [Paired t-test](#)

**Abstract:** This dataset contains the labor force participation rate (LFPR) of women in 19 cities in the United States in each of two years (1968 and 1972). The data help to measure the growing presence of women in the labor force over this period.

It may seem reasonable to compare LFPR rates in the two years with a pooled t-test since the United States did not change much from 1968 to 1972. However, the data are naturally paired because the measurements were made in the same cities for each of the two years. It is better to compare each city in 1972 to its own value in 1968.

The data offer a good example of how a paired t-test can be more effective when it is appropriate. Here, a pooled t-test is not significant, but a paired test is significant and the .05 level for testing the null hypothesis of no change in the LFPR between 1968 and 1972.

**Image:** Boxplots of the labor force participation rate of women in 19 U.S. cities in the years 1968 and 1972



<http://lib.stat.cmu.edu/DASL/Stories/WomenintheLaborForce.html>

```
#paired samples test
```

```
d1968 = np.array([0.42,0.5,0.52,0.45,0.43,0.55,0.45,0.34,0.45,0.54,0.42,0.51,0.49,0.54,0.5,0.58,0.49,0.56,0.63])
```

```
d1972 = np.array([0.45,0.5,0.52,0.45,0.46,0.55,0.60,0.49,0.35,0.55,0.52,0.53,0.57,0.53,0.59,0.64,0.5,0.57,0.64])
```

```
#t-test related samples - paramétrique
```

```
print(stat.ttest_rel(d1968,d1972))# (stat.test = -2.45, p-value = 0.024)
```

```
#test des rangs signés – non paramétrique
```

```
print(stat.wilcoxon(d1968,d1972)) # (stat = 16, p-value = 0.0122)
```

➔ La proportion des femmes actives est significativement (à 5%) différente en 1972

Voir R. Rakotomalala

[Comparaison de populations – Tests paramétriques](#)

[Comparaison de populations – Test non paramétriques](#)

**Story Name:** Alcohol and Tobacco

**Story Topics:** [Consumer](#) , [Health](#)

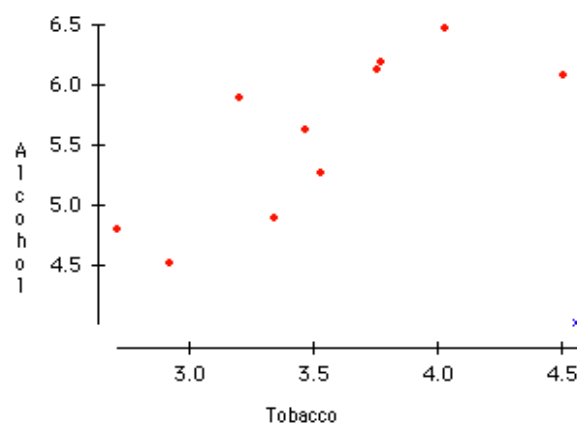
**Datafile Name:** [Alcohol and Tobacco](#)

**Methods:** [Correlation](#) , [Dummy variable](#) , [Outlier](#) , [Regression](#) , [Scatterplot](#)

**Abstract:** Data from a British government survey of household spending may be used to examine the relationship between household spending on tobacco products and alcoholic beverages. A scatterplot of spending on alcohol vs. spending on tobacco in the 11 regions of Great Britain shows an overall positive linear relationship with Northern Ireland as an outlier. Northern Ireland's influence is illustrated by the fact that the correlation between alcohol and tobacco spending jumps from .224 to .784 when Northern Ireland is eliminated from the dataset.

This dataset may be used to illustrate the effect of a single influential observation on regression results. In a simple regression of alcohol spending on tobacco spending, tobacco spending does not appear to be a significant predictor of tobacco spending. However, including a dummy variable that takes the value 1 for Northern Ireland and 0 for all other regions results in significant coefficients for both tobacco spending and the dummy variable, and a high R-squared.

**Image:** Scatterplot of Alcohol vs. Tobacco, with Northern Ireland marked with a blue X.



<http://lib.stat.cmu.edu/DASL/Stories/AlcoholandTobacco.html>

**#données pour corrélation et régression (Irlande du Nord non incluse)**

```
dalc = np.array([6.47,6.13,6.19,4.89,5.63,4.52,5.89,4.79,5.27,6.08])
```

```
dtob = np.array([4.03,3.76,3.77,3.34,3.47,2.92,3.2,2.71,3.53,4.51])
```

**#corrélation de Pearson**

```
print(stat.pearsonr(dalc,dtob)) # (r = 0.7843, p-value pour test t = 0.0072)
```

**#corrélation de Spearman - basé sur les rangs**

```
print(stat.spearmanr(dalc,dtob)) # (rho = 0.8303, p-value = 0.0029)
```

**#tau de Kendall - concordance et discordance**

```
print(stat.kendalltau(dalc,dtob)) # (tau = 0.6444, p-value = 0.0095)
```

**#régression linéaire simple**

```
print(stat.linregress(dalc,dtob)) # (pente = 0.6115, const = 0.1081, r =  
0.7843, p-value test signif. pente = 0.0072, sigma err = 0.1710)
```

Voir R. Rakotomalala

[Analyse de corrélation – Etude des dépendances](#)

[Econométrie – La régression simple et multiple](#)



Traitement de  $K > 2$  vecteurs de données

Comparaison de populations (échantillons indépendants)

## **STATISTIQUES À K VECTEURS**

**Story Name:** Hot dogs

**Story Topics:** [Food](#) , [Nutrition](#) , [Science](#)

**Datafile Name:** [Hot dogs](#)

**Methods:** [ANOVA](#)

**Abstract:** People who are concerned about their health may prefer hot dogs that are low in salt and calories. The "Hot dogs" datafile contains data on the sodium and calories contained in each of 54 major hot dog brands. The hot dogs are classified by type: beef, poultry, and meat (mostly pork and beef, but up to 15% poultry meat).

A simple ANOVA model with type as the independent variable and calories as the dependent variable has an F-ratio of 16.074 , which is highly significant. Post-ANOVA analysis of group means shows that meat and beef hot dogs have approximately the same number of calories and poultry hot dogs generally have fewer calories than either beef or meat hot dogs.

The same model with sodium as the dependent variable has an overall F-ratio of 1.78, which is not significant. None of the individual differences are significant.

<http://lib.stat.cmu.edu/DASL/Stories/Hotdogs.html>

#Sodium – données pour tests K échantillons indép.

```
dbef = np.array([495,477,425,322,482,587,370,322,479,375,330,300,386,401,645,440,317,319,298,253])
```

```
dmeat = np.array([458,506,473,545,496,360,387,386,507,393,405,372,144,511,405,428,339])
```

```
dpoultry = np.array([430,375,396,383,387,542,359,357,528,513,426,513,358,581,588,522,545])
```

#test égalité des dispersions (K variances)

```
print(stat.levene(dbef,dmeat,dpoultry)) # (stat. = 0.2494, p-value = 0.7802)
```

#ANOVA à 1 facteur - comparaison des moyennes - paramétrique

```
print(stat.f_oneway(dbef,dmeat,dpoultry)) # (stat. F = 1.7778, p-value = 0.1793)
```

#Test de Kruskal-Wallis – non-paramétrique

```
print(stat.kruskal(dbef,dmeat,dpoultry)) # (stat. K = 4.1728, p-value = 0.0947)
```

➔ Les teneurs en sel des différents types de hot-dog ne sont pas significativement différentes

Techniques de « clustering »

# **CLASSIFICATION AUTOMATIQUE**

Objectif de la classification automatique : construire une catégorisation (typologie) des individus (observations) en se basant sur leurs similarités. Le nombre de groupes est soit fixé à l'avance, soit déduit des calculs.

**Datafile Name:** Cities

**Datafile Subjects:** [Economics](#)

**Story Names:** [Economics of Cities](#)

**Reference:** *Prices and earnings around the globe* Economic Research Department, Union Bank of Switzerland, Zurich.

**Authorization:** Contact Authors

**Description:** The data were collected by the Economic Research Department of the Union Bank of Switzerland. They represent the economic conditions in 48 cities around in world in 1991.

**Number of cases:** 48

**Variable Names:**

1. City: City name
2. Work: Weighted average of the number of working hours in 12 occupations
3. Price: Index of the cost 112 goods and services excluding rent (Zurich = 100)
4. Salary: Index of hourly earnings in 12 occupations after deductions (Zurich = 100)

**The Data:**

City	Work	Price	Salary	
Amsterdam		1714	65.6	49.0
Athens	1792	53.8	30.4	
Bogota	2152	37.9	11.5	
Bombay	2052	30.3	5.3	

*Pour simplifier...*

Les individus présentant des valeurs manquantes ont été supprimés.  
Seules les variables **Price** et **Salary** sont utilisées.

```
import numpy as np
import scipy.stats as stat
import scipy.cluster as cluster
```

#matrice de données

```
M = np.loadtxt("datacluster.txt",delimiter="\t",dtype=float)
```

#écart-type des obs. en ligne pour chaque col.

```
np.std(M,axis=0) # array([21.15, 24.487])
```

#plot - graphique nuage de points

```
import matplotlib.pyplot as plt
plt.plot(M[:,0],M[:,1],"ko")
```

#centrer et réduire

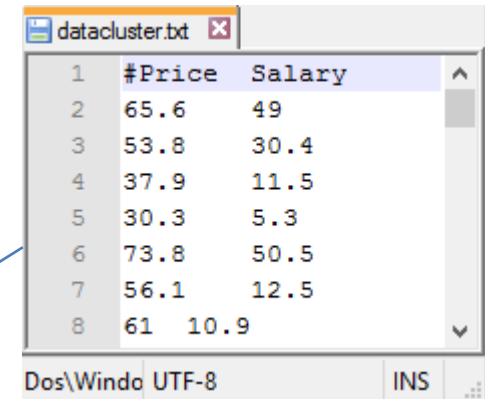
#axis=0, calcul des moyenne/variance des lignes pour chaque col.

#ddof = 0, utiliser  $1/(n-0)$  pour le calcul de la variance

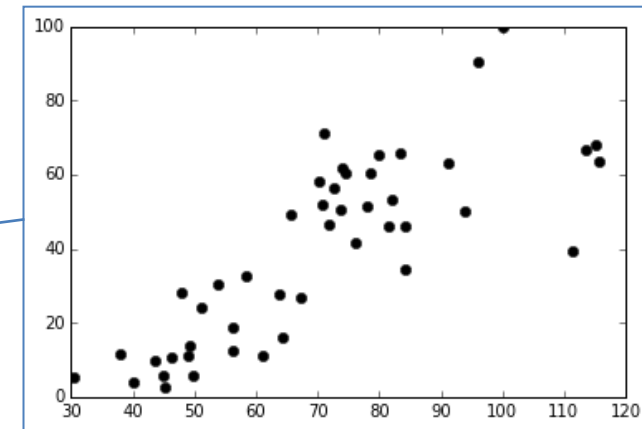
```
Z = stat.zscore(M,axis=0,ddof=0)
```

#écart-type

```
np.std(Z,axis=0) # array([1. , 1.])
```



	#Price	Salary
1	#Price	Salary
2	65.6	49
3	53.8	30.4
4	37.9	11.5
5	30.3	5.3
6	73.8	50.5
7	56.1	12.5
8	61	10.9



L'idée est surtout de ramener les variables sur la même échelle.

# K-means (K-moyennes)

City	Groupe affecté
Copenhagen	2
Geneva	2
Helsinki	2
Oslo	2
Stockholm	2
Tokyo	2
Zurich	2
Amsterdam	1
Brussels	1
Chicago	1
Dublin	1
Dusseldorf	1
Frankfurt	1
Houston	1
London	1
Los Angeles	1
Luxembourg	1
Madrid	1
Milan	1
Montreal	1
New York	1
Paris	1
Sydney	1
Taipei	1
Toronto	1
Vienna	1
Athens	0
Bogota	0
Bombay	0
Buenos Aires	0
Caracas	0
Hong Kong	0
Johannesburg	0
Kuala Lumpur	0
Lagos	0
Lisbon	0
Manila	0
Mexico City	0
Nairobi	0
Nicosia	0
Panama	0
Rio de Janeiro	0
Sao Paulo	0
Seoul	0
Singapore	0
Tel Aviv	0

Utilisation des données  
centrées et réduites.

#k-means en 3 groupes

centroid,label = **cluster.vq.kmeans2**(Z,k=3)

#centres de classes

print(centroid)

#id des groupes pour chaque individu

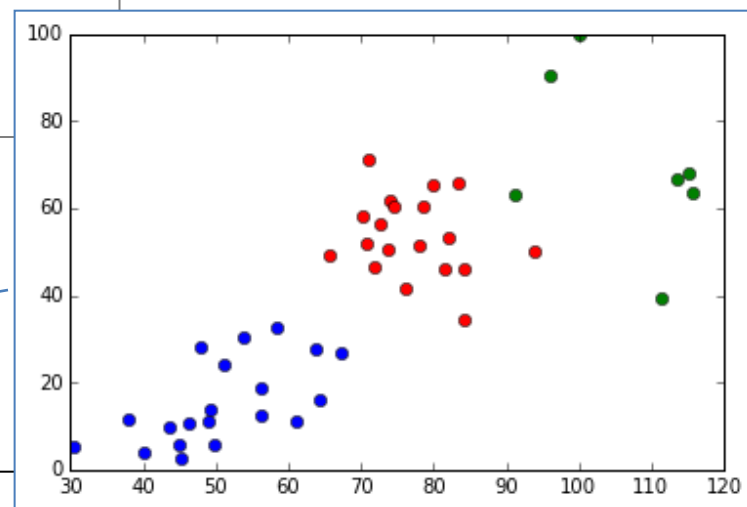
print(label) #[1 0 0 0 1 0 0 1 2 1 ...]

#représentation graphique

```
plt.plot(M[label==0,0],M[label==0,1],"bo",  
         M[label==1,0],M[label==1,1],"ro",  
         M[label==2,0],M[label==2,1],"go")
```

Les centres de classes sont  
calculés sur les données  
centrées et réduites initialement.

```
[[-0.91229626 -0.98442173]  
 [ 0.33362129  0.57652783]  
 [ 1.70101723  1.24777225]]
```



#dendrogramme - méthode de Ward

```
W = cluster.hierarchy.ward(Z)
```

#affichage du dendrogramme

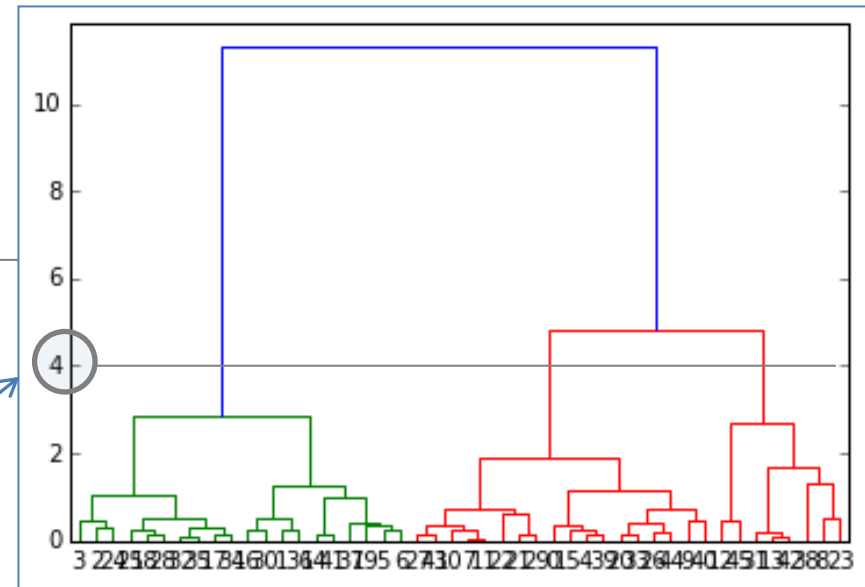
```
cluster.hierarchy.dendrogram(W)
```

#découpage

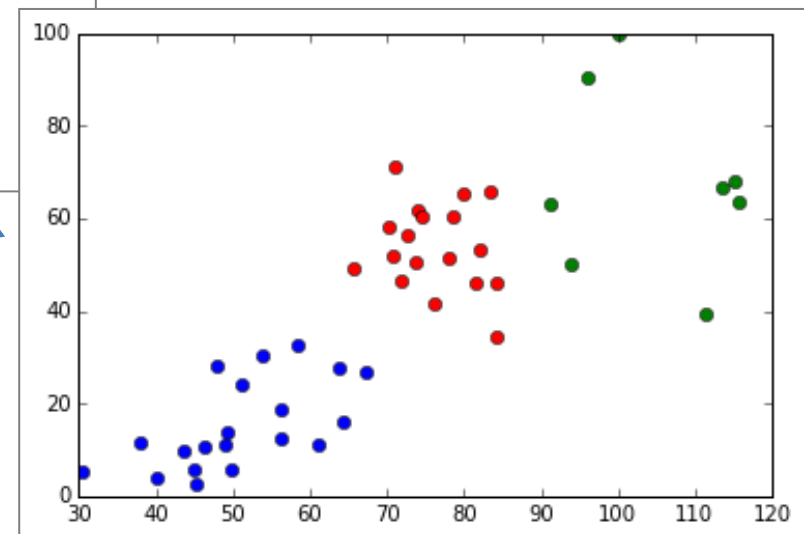
```
idx = cluster.hierarchy.fcluster(W,t=4,criterion="distance")
```

#graphique again

```
plt.plot(M[idx==1,0],M[idx==1,1],"bo",  
         M[idx==2,0],M[idx==2,1],"ro",  
         M[idx==3,0],M[idx==3,1],"go")
```



Ce découpage induit la création de 3 groupes...



... les mêmes presque (à 1 individu près) que ceux du K-means.



De la documentation à profusion (n'achetez pas des livres sur Python)

Site du cours

[http://eric.univ-lyon2.fr/~ricco/cours/cours\\_programmation\\_python.html](http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html)

Site de Python

Welcome to Python - <https://www.python.org/>

Python 3.4.3 documentation - <https://docs.python.org/3/index.html>

Portail Python

Page Python de [Developpez.com](http://developpez.com)

Quelques cours en ligne

P. Fuchs, P. Poulain, « [Cours de Python](#) » sur Developpez.com

G. Swinnen, « [Apprendre à programmer avec Python](#) » sur Developpez.com

« [Python](#) », Cours interactif sur [Codecademy](#)

POLLS (KDNuggets)

**Data Mining / Analytics Tools Used**

Python, 4<sup>ème</sup> en [2015](#)

**What languages you used for data mining / data science?**

Python, 3<sup>ème</sup> en [2014](#) (derrière R et SAS)