



Clustering

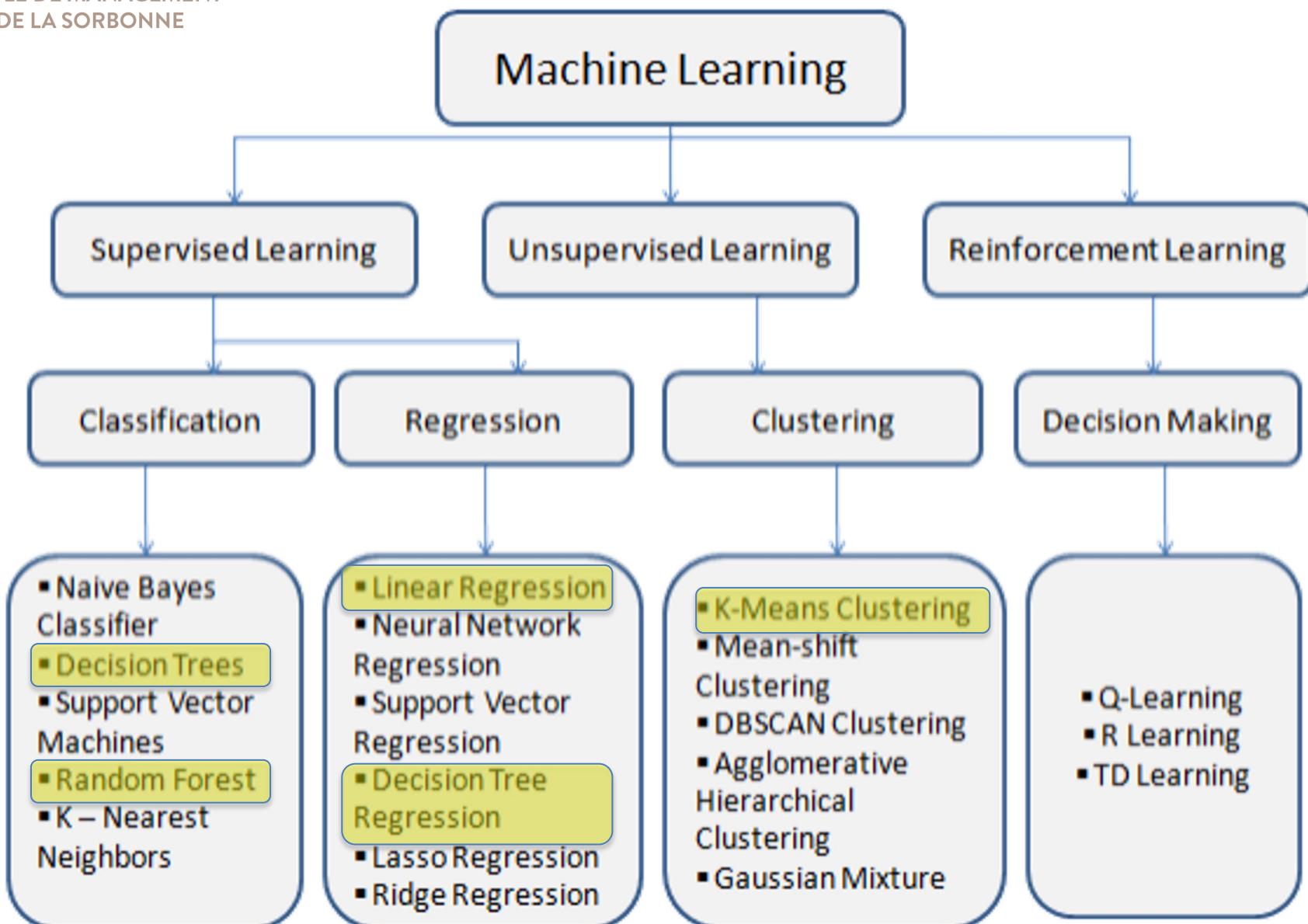
(Classification non-supervisée)

Fichiers sur

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

<http://kirschpm.fr/cours/PythonDataScience/>

Types of Machine Learning





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Apprentissage supervisé / non supervisé

supervised learning

Input data



Annotations

These are
apples



Model



Prediction

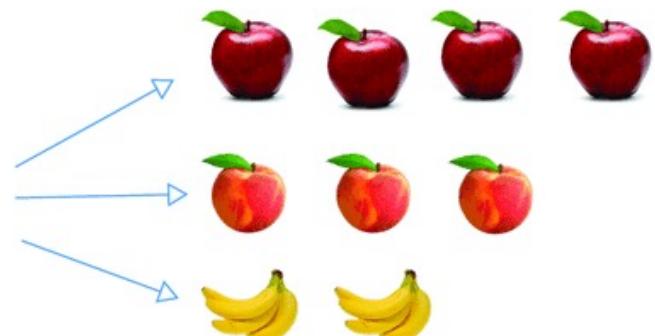
Its an
apple!

unsupervised learning

Input data

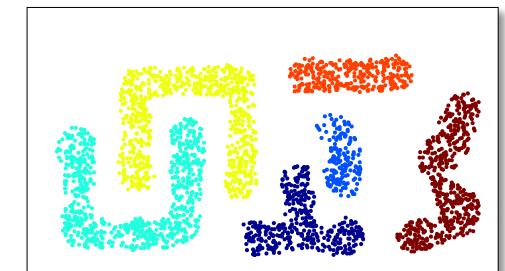
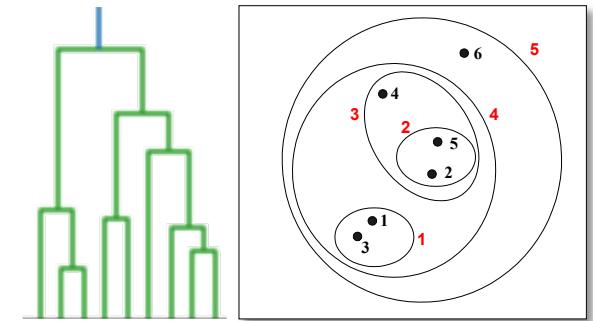
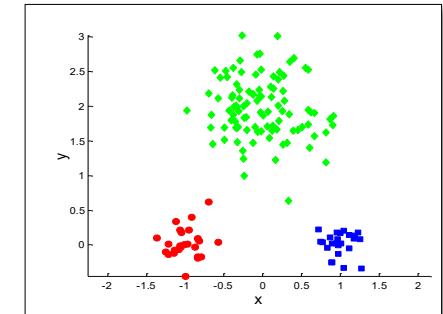


Model



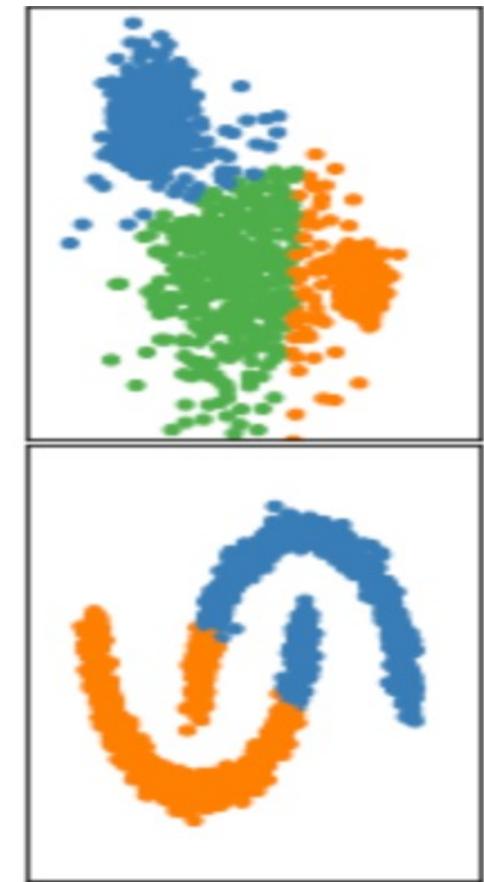
Trois grands types de clustering

- Séparer les éléments en groupes (*clusters*) **disjoints**
- Les clusters ne sont **pas connus à l'avance**
- **Clustering par partition**
 - Une division des données en **sous-ensembles (clusters)** « patatoides »
 - Ex.: K-Means, K-Medians, Fuzzy C-Means
- **Clustering hiérarchique**
 - Un ensemble de *clusters* emboités les uns dans les autres, avec une **structure hiérarchique (arbre)**
 - Ex.: Agglomerative (linkage), Divisive
- **Clustering par densité**
 - Une division en *clusters* s'appuie sur la **densité estimée** des clusters
 - Ex.: DBScan



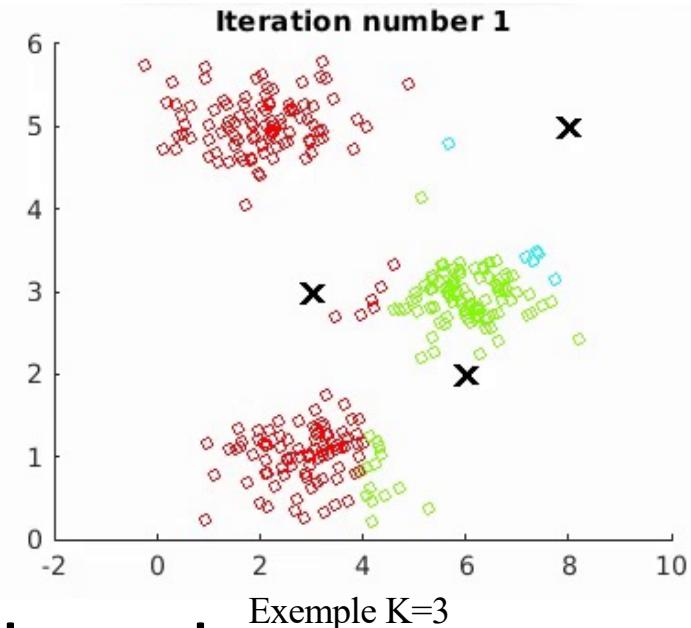
Clustering par Partition (le plus utilisé)

- Une bonne méthode de **regroupement**
 - Exemple d'application : customer segmentation
- Permet de garantir :
 - Une **grande similarité intra-groupe**
 - Une **faible similarité inter-groupe**
- **Quelques limitations :**
 - N'est pas adapté à des « formes » complexes
 - Peut produire des résultats différents à chaque exécution



Fonctionnement K-means

- Prendre **K** points (au hasard) comme centroïdes initiaux
- Répéter
 - Former **K** clusters en assignant les points à leur centroïde le **plus proche**
 - Mettre à jour le centroïde de chaque cluster jusqu'à ce que qu'aucun centroïde ne bouge
- **Notion de distance**
 - Chaque cluster contient **les points les plus proches**
 - Distance Euclidienne \approx **métrique MSE**
 - **Attention à la Normalisation**
- **Aucune étiquette n'est nécessaire**



Clusters doivent être interprétés





UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

K-means

```
from sklearn.cluster import KMeans
```

```
means = KMeans(n_clusters=3)
```

Création objet modèle

On doit indiquer le nombre de clusters K

On construit le modèle (`fit`)

```
means.fit(X)
```

Ou

```
labels = means.fit_predict(X)
```

*Dataframe avec les données (**features**) en entrée
(sans colonne ‘target’ car apprentissage **non supervisé**)*

```
means.labels_
```

On récupère les labels (clusters) attribués à chaque valeur

```
means.cluster_centers_
```

On récupère le centre (centroïde) de chaque clusters

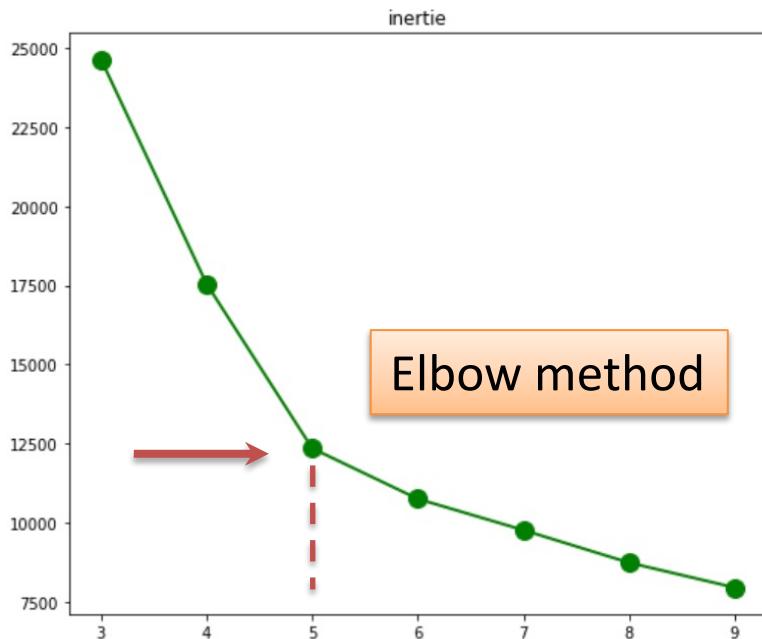
Fonctionnement K-means (hyperparamètres)

- Choix des **centroïdes de départ** est déterminant
 - Mauvais choix → difficultés à converger → résultat + médiocre
- **Hyperparamètres : *init* et *n_init***
 - **init** : méthode d'initialisation
 - Valeurs : k-means++, random ou n-array (*k clusters, n features*)
 - **n_init** : nb de fois l'algo sera exécuté avec \neq centroid seeds
 - Valeur par défaut : 10

```
means = KMeans(init='k-means++',  
                n_init=12,  
                n_clusters=3)  
  
means = KMeans(init='random',  
                n_clusters=3,  
                n_init=12)  
  
import numpy as np  
  
centroids = np.array( [ [-15, 10],  
                      [5, 8],  
                      [-10, 10],  
                      [5,-5] ] )  
  
means = KMeans(init=centroids,  
                n_clusters=4,  
                n_init=1)
```

Fonctionnement K-means (choix du k)

- Comment choisir son K ?
 - Tester des multiples valeurs !
- Comment évaluer la qualité de son cluster ?
 - *Pas de labels* disponibles (« ground truth »)



Inertie (attribut *inertia_*)

- Somme des distances au carré entre un point et son centroïde
- Une faible inertie correspond à une **faible distance intra-cluster**
- Inertie 0 théorique : autant des clusters que des points

means.inertia_

Fonctionnement K-means (choix du k)

- **Silhouette Score**

- À quel point un élément est similaire à son propre cluster comparé aux éléments sur les autres clusters

$$silhouette = \frac{B - A}{\max(A, B)}$$

A → distance moyenne entre un point et les autres points du *cluster plus proche (inter-cluster)*

B → distance moyenne entre un point et les autres points de *son cluster (intra-cluster)*

Score entre [-1, 1]

1 → clusters + dense et mieux séparés

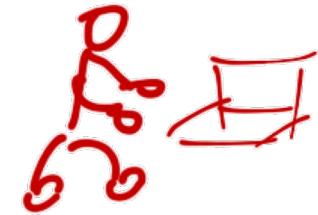
```
from sklearn.metrics import silhouette_score
```

```
scoef = silhouette_score(X, means.labels_)
```

On lui indique les données (X) et les labels obtenus



Hands On !



- Etude du dataset
 - Créer le DataFrame avec les données aléatoires

```
import pandas as pd
from sklearn.datasets import make_blobs
```

On n'oublie pas les imports

```
x,y = make_blobs(n_samples=150, centers=4, n_features=2,
                   cluster_std=[1, 1.5, 2, 2.3], random_state=42)
```

```
df_blobs = pd.DataFrame ({'x1' : x[:,0] ,
                           'x2' : x[:,1],
                           'y' : y })
```

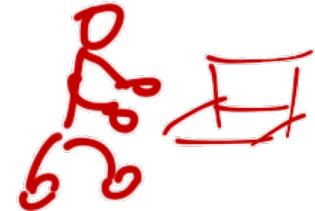
Création d'un dataset de 150 éléments plus ou moins dispersés autour de 4 centres

```
df_blobs.sample(25)
```

Deux features seulement pour rendre la visualisation plus facile

	x1	x2	y
53	-2.972615	8.548556	0
27	-2.165579	7.251246	0
121	-10.193867	9.277608	3
141	-1.696672	10.370526	0

Hands On !

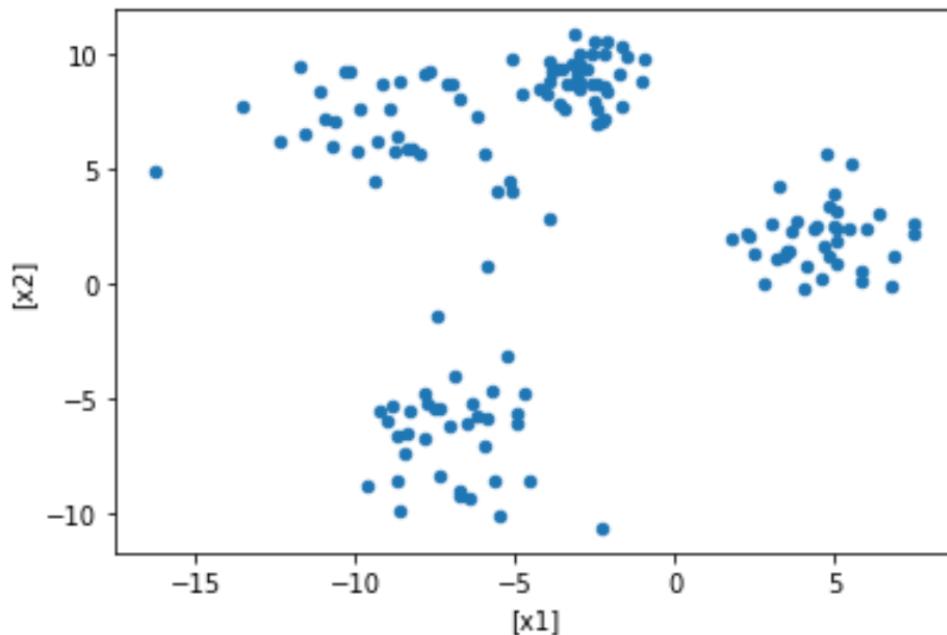


- Visualisation des données
 - On peut afficher les données sur un plan 2D

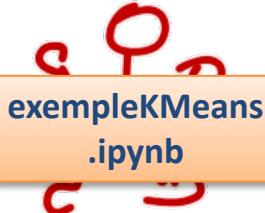
```
import matplotlib.pyplot as plt
%matplotlib inline

df_blobs.plot(kind='scatter',
               x=['x1'], y=['x2'])
```

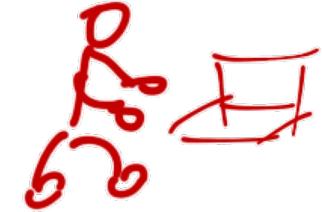
*Création d'un graphique de type **scatter**
à partir de notre **dataframe***



L'observation des différentes combinaisons de *features 2 par 2* peut nous montrer les *features* les plus propices à la séparation en clusters



Hands On !



• Exemple clustering

- Créer le modèle K-Means
- Entraîner le modèle

On va d'abord éliminer la colonne **y**, et ne garder que les features.

```
X = df_blobs.drop(columns=['y'])
```

```
from sklearn.cluster import KMeans
```

On n'oublie pas l'import

```
km = KMeans(init='k-means++', n_clusters=3)
```

*Méthode d'initialisation : **k-means++***

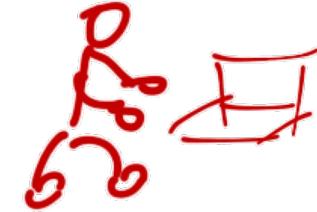
```
km.fit( X )
```

*Entrainement du modèle
(seulement avec les features)*

*Création du modèle
avec 3 classes (**K = 3**)*

Important : modèle non supervisé,
donc on ne fournit pas de target avec le **fit**

Hands On !



- Exemple clustering

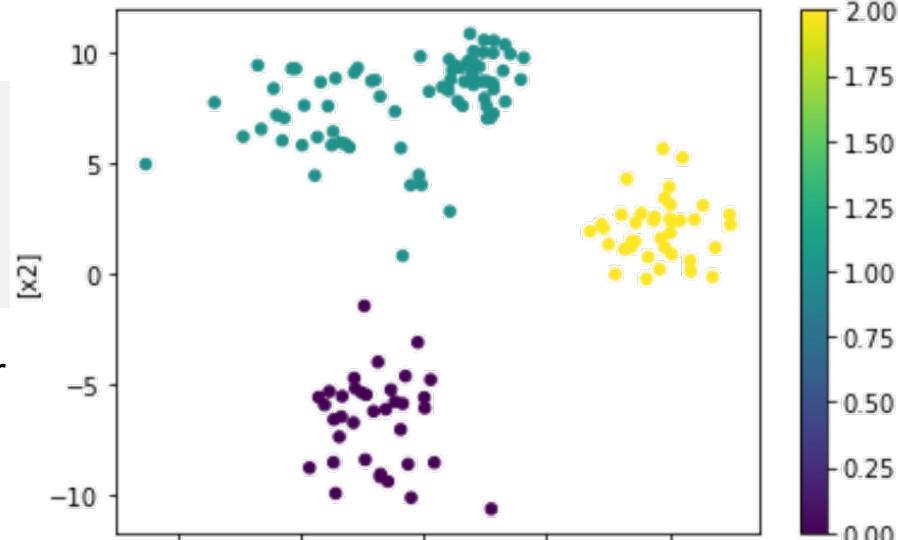
- Afficher les labels créés lors du clustering :

```
print(km.labels_)
```

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

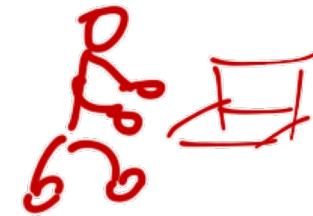
- Visualisation

```
df_blobs.plot(kind='scatter',  
              x=['x1'], y=['x2'],  
              c=km.labels_,  
              colormap='viridis')
```



On utilise les labels comme indicateur de couleur dans l'échelle « viridis ».

Hands On !



- **Exemple clustering : comment trouver le bon K ?**
 - On va tester plusieurs valeurs de K
 - Pour plus de fun : on va augmenter le nombre de points dans le dataset

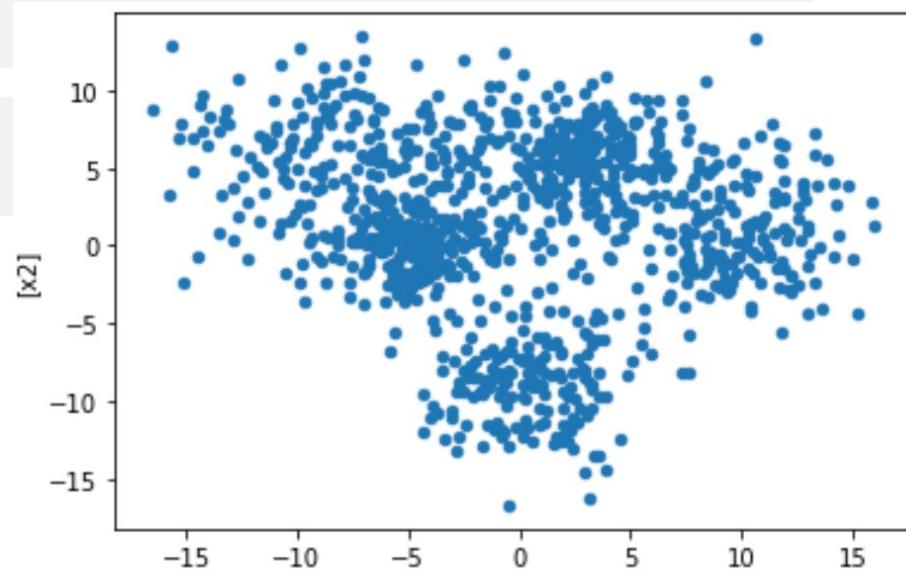
```
x,y = make_blobs(n_samples=1000, centers=6, n_features=2,  
                  cluster_std=[3.5, 4.5, 3.0, 2.5, 1.5, 2.0],  
                  random_state=7)
```

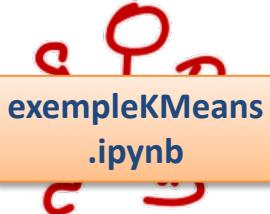
```
X = pd.DataFrame ({'x1' : x[:,0] , 'x2' : x[:,1] })
```

```
X.describe()
```

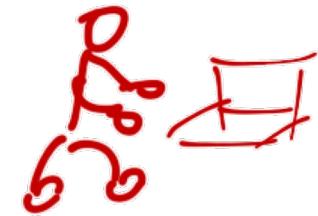
```
X.plot(kind='scatter', x=['x1'],  
       y=['x2'])
```

	x1	x2
count	1000.000000	1000.000000
mean	-0.160541	1.148996
std	6.515354	5.811310
min	-16.541976	-16.697765





Hands On !



- Exemple clustering : comment trouver le bon K ?
 - Définir les valeurs de K à tester
 - Enregistrer les valeurs d'inertie et le Silhouette score pour les comparer

```
from sklearn.metrics import silhouette_score
On n'oublie pas l'import

n_clusters = [3, 4, 5, 6, 7, 8, 9]
On va tester le K de 3 à 9

inerties = []
silhouettes = []
Deux tableaux pour garder l'inertie et le silhouette score
```

```
for k in n_clusters :
    km = KMeans(n_clusters=k, init='k-means++', n_init=12, random_state=7)
    km.fit(X)

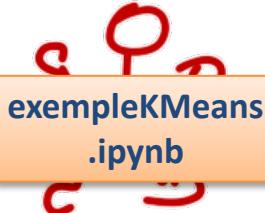
    scoef = silhouette_score(X, km.labels_)
    inerties.append(km.inertia_)

    silhouettes.append(scoef)

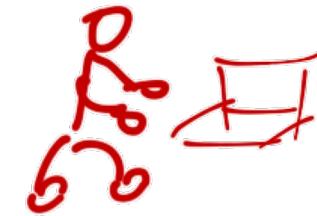
print ('inertie =', km.inertia_, 'silhouette score=', scoef)
```

Pour chaque valeur de k, on va créer un modèle

Puis, on récupère le score silhouette et l'inertie et on les garde



Hands On !



- Exemple clustering : comment trouver le bon K ?
 - On peut aussi afficher les clusters et leurs centroïde

Pour chaque valeur de k, on va créer un modèle

```
for k in n_clusters :
    km = KMeans(n_clusters=k, init='k-means++', n_init=12, random_state=7)
    km.fit(X)

    scoef = silhouette_score(X, km.labels_)
    inerties.append(km.inertia_)
    silhouettes.append(scoef)

    fig, ax = plt.subplots(1,1, figsize=(10,5) )
    X.plot(kind='scatter', x=['x1'], y=['x2'], c=km.labels_,
            colormap='viridis', ax=ax, title=f'{k} clusters')

    ax.plot(km.cluster_centers_[:,0], km.cluster_centers_[:,1], 'k^', ms=12,
            alpha=0.35)
```

Plot des centroides

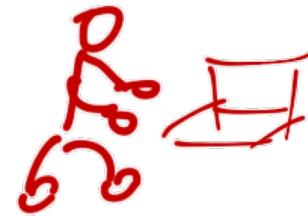
L'attribut cluster_centers_ est une liste contenant le centroïde de chaque cluster.

km.cluster_centers_[:,0] donne les 'x1' des centroides, km.cluster_centers_[:,1] le 'x2'

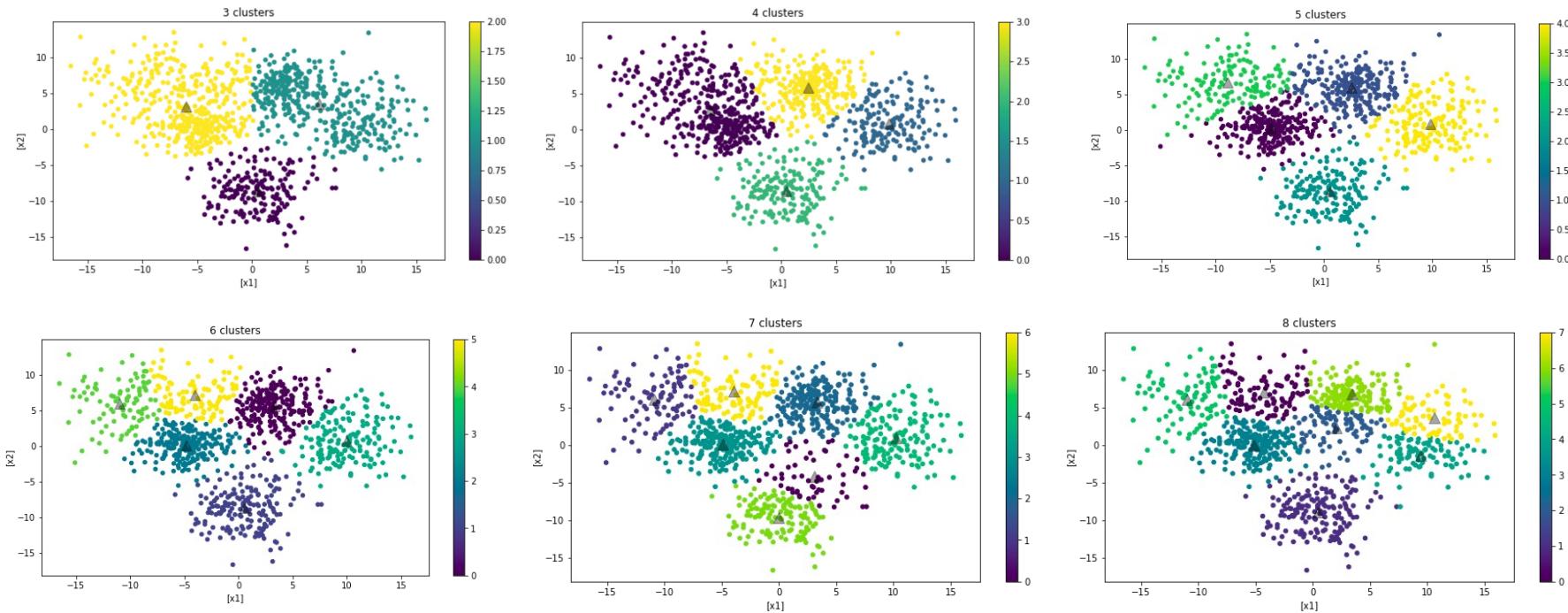
Puis, on récupère le score silhouette et l'inertie et on les garde

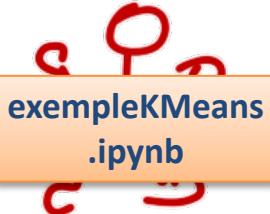
Création de la figure et plot des points colorés avec leur label.

Hands On !

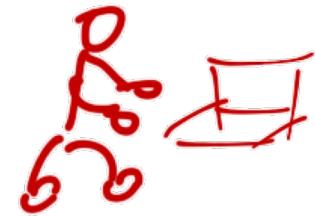


- Exemple clustering : comment trouver le bon K ?
 - On peut aussi afficher les clusters et leurs centroïde





Hands On !



- **Exemple clustering : comment trouver le bon K ?**

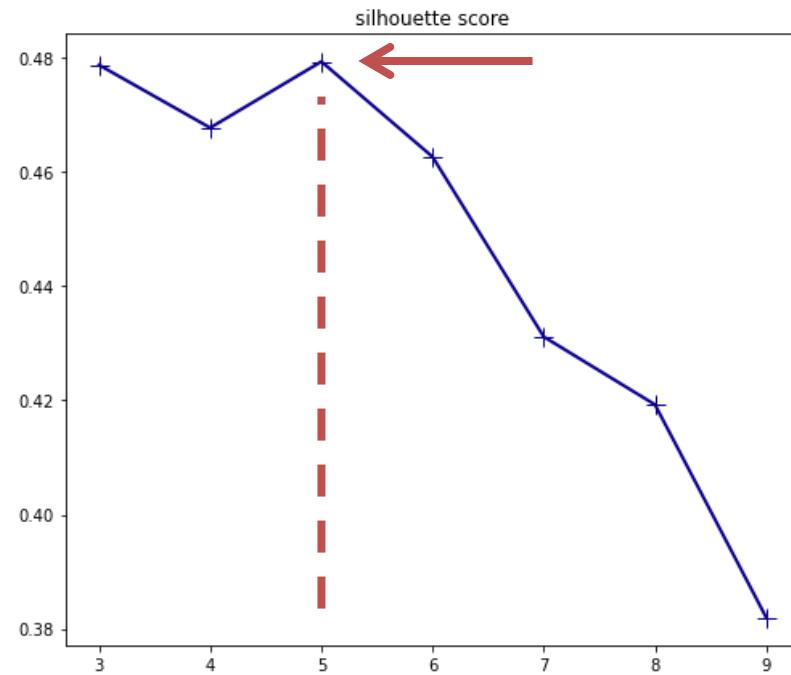
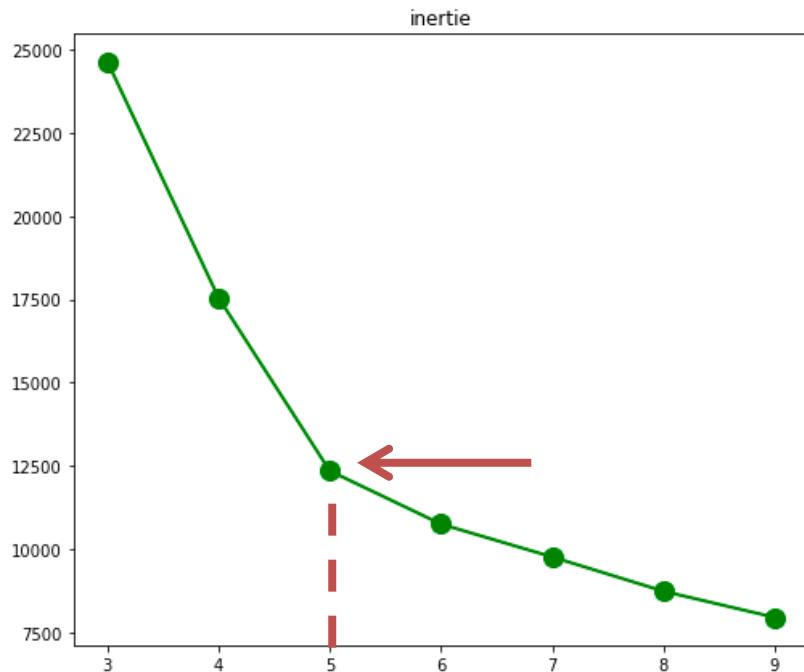
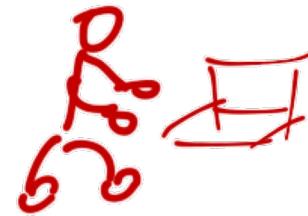
- Comparer les valeurs d'inertie et le score silhouette

```
fig, axs = plt.subplots(1,2, figsize=(18,7)) ← Création de l'espace pour 2 figures côté à côté.
axs[0].set_title('inertie')
axs[0].plot(n_clusters,inerties, marker='o', color='green',
            linewidth=2, markersize=12)

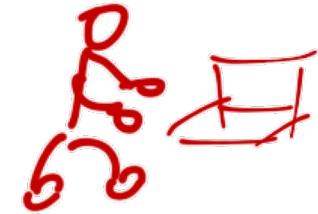
axs[1].set_title('silhouette score') ← Dans le 1er espace (axs[0]), on plotte les valeurs de k (n_clusters) vs les inerties.
axs[1].plot(n_clusters,silhouettes, marker='+',
            color='darkblue',
            linewidth=2, markersize=12)

Au 2ème espace (axs[1]), on plotte les valeurs de k (n_clusters) vs les valeurs du score silhouette.
```

Hands On !



Hands On !



- **Exemple : Analyse Titanic**

- « Segmentation » des passager du Titanic
 - Mieux comprendre la population, pas que en fonction de la survie
- Dataset avec plus de features : **titanic.csv**
 - Survival, Sex, Age
 - pclass: Ticket class (1 = 1st, 2 = 2nd, 3 = 3rd)
 - sibsp: nb de personne de la famille (enfant, épouses...)
 - parch: nb de parents / enfants

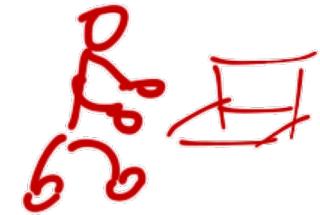
```
import pandas as pd

df_titanic = pd.read_csv('titanic.csv',
                        delimiter=',',
                        header=[0])

df_titanic.info()
```

#	Column	Non-Null Count	Dtype
0	PassengerId	891 non-null	int64
1	Survived	891 non-null	int64
2	Pclass	891 non-null	int64
3	Name	891 non-null	object
4	Sex	891 non-null	object
5	Age	714 non-null	float64
6	SibSp	891 non-null	int64
7	Parch	891 non-null	int64
8	Ticket	891 non-null	object

Hands On !



- **Exemple : Analyse Titanic**

- Nettoyage des données

- Supprimer les colonnes qui ne nous intéressent pas
 - Remplir les données manquantes (âge notamment)

c o o
 exempleKMeans
 Titanic.ipynb
 c o

```
df_titanic.drop(columns=['PassengerId', 'Name', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
                 inplace=True)
```

```
df_titanic['Age'].fillna(df_titanic['Age'].mean(), inplace=True)
```

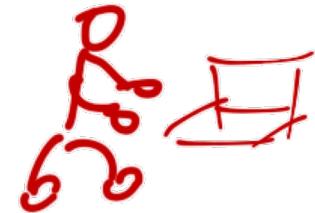
```
df_titanic.info()
```

#	Column	Non-Null Count	Dtype
0	Survived	891 non-null	int64
1	Pclass	891 non-null	int64
2	Sex	891 non-null	object
3	Age	891 non-null	float64
4	SibSp	891 non-null	int64
5	Parch	891 non-null	int64

RangeIndex: 891 entries, 0 to 890
Data columns (total 6 columns):
df_titanic.sample(15)

Survived	Pclass	Sex	Age	SibSp	Parch
437	1	2	female	24.000000	2
343	0	2	male	25.000000	0
226	1	2	male	19.000000	0
383	1	1	female	35.000000	1
309	1	1	female	30.000000	0
119	0	3	female	2.000000	4

Hands On !



- **Exemple : Analyse Titanic**

- Nettoyage des données

- Il faut « encoder » la colonne « Sex »

```
df_titanic = pd.get_dummies(df_titanic, columns=['Sex'], drop_first=True)

df_titanic.info()
```

- Créer des nouvelles colonnes : **NbFamily** et **Alone**

```
df_titanic['FamilyNb'] = df_titanic['SibSp'] + df_titanic['Parch']
df_titanic['Alone'] = ( df_titanic['FamilyNb'] == 0)
```

```
df_titanic.drop(columns=['SibSp', 'Parch'],
                 inplace=True)
```

```
df_titanic.info()
```

RangeIndex:	891 entries, 0 to 890		
Data columns (total	6 columns):		
#	Column	Non-Null Count	Dtype
0	Survived	891 non-null	int64
1	Pclass	891 non-null	int64
2	Age	891 non-null	float64
3	Sex_male	891 non-null	uint8
4	FamilyNb	891 non-null	int64
5	Alone	891 non-null	bool

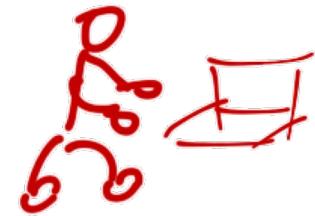


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Hands On !



- Exemple : Analyse Titanic

Survived	Pclass	Age	Sex_male	FamilyNb	Alone
699	0	3	42.000000	1	0 True
206	0	3	32.000000	1	1 False
592	0	3	47.000000	1	0 True
817	0	2	31.000000	1	2 False
430	1	1	28.000000	1	0 True

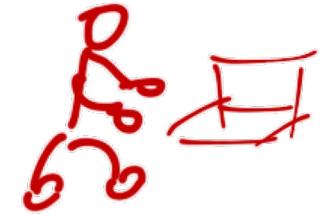
df_titanic.sample(15)

df_titanic.describe()

	Survived	Pclass	Age	Sex_male	FamilyNb
count	891.000000	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.647587	0.904602
std	0.486592	0.836071	13.002015	0.477990	1.613459
min	0.000000	1.000000	0.420000	0.000000	0.000000
25%	0.000000	2.000000	22.000000	0.000000	0.000000
50%	0.000000	3.000000	29.699118	1.000000	0.000000
75%	1.000000	3.000000	35.000000	1.000000	1.000000
max	1.000000	3.000000	80.000000	1.000000	10.000000

exempleKMeans
Titanic.ipynb

Hands On !



- Exemple : Analyse Titanic
 - Construction du modèle

```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=4, random_state=42)

km.fit(df_titanic)

print(km.inertia_)
```

21081.28798483158

```
print (df_titanic.columns)
print (km.cluster_centers_)
```

```
Index(['Survived', 'Pclass', 'Age', 'Sex_male', 'FamilyNb', 'Alone'], dtype='object')
[[3.6000000e-01 2.4800000e+00 2.0840000e+01 6.2000000e-01
 7.4400000e-01 6.3200000e-01]
[3.73239437e-01 1.71126761e+00 5.16408451e+01 6.90140845e-01
 6.47887324e-01 6.05633803e-01]
[5.79710145e-01 2.63768116e+00 4.77057971e+00 5.36231884e-01
 3.27536232e+00 2.89855072e-02]
[3.69767442e-01 2.35348837e+00 3.16040554e+01 6.67441860e-01
 7.02325581e-01 6.76744186e-01]]
```

exempleKMeans
Titanic.ipynb

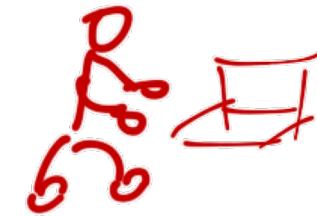


UNIVERSITÉ PARIS 1

PANTHÉON SORBONNE

ÉCOLE DE MANAGEMENT
DE LA SORBONNE

Hands On !



- Exemple : Analyse Titanic
 - Interprétation des labels (clusters) obtenus

```
df_titanic['labels'] = km.labels_
```

```
df_titanic.groupby('labels').describe().transpose()
```

exemplKMeans
Titanic.ipynb

	labels	0	1	2	3
Survived	count	250.000000	142.000000	69.000000	430.000000
	mean	0.360000	0.373239	0.579710	0.369767
	std	0.480963	0.485377	0.497222	0.483304
	min	0.000000	0.000000	0.000000	0.000000
	25%	0.000000	0.000000	0.000000	0.000000
	50%	0.000000	0.000000	1.000000	0.000000
	75%	1.000000	1.000000	1.000000	1.000000
	max	1.000000	1.000000	1.000000	1.000000
Pclass	count	250.000000	142.000000	69.000000	430.000000
	mean	2.480000	1.711268	2.637681	2.353488
	std	0.756232	0.830116	0.593371	0.825398
	min	1.000000	1.000000	1.000000	1.000000
	25%	2.000000	1.000000	2.000000	2.000000
	50%	3.000000	1.000000	3.000000	3.000000
	75%	3.000000	2.000000	3.000000	3.000000
	max	3.000000	3.000000	3.000000	3.000000
Age	count	250.000000	142.000000	69.000000	430.000000
	mean	30.000000	30.000000	30.000000	30.000000
	std	15.413000	15.413000	15.413000	15.413000
	min	0.000000	0.000000	0.000000	0.000000
	25%	14.500000	14.500000	14.500000	14.500000
	50%	24.000000	24.000000	24.000000	24.000000
	75%	34.500000	34.500000	34.500000	34.500000
	max	80.000000	80.000000	80.000000	80.000000

Age	count	250.000000	142.000000	69.000000	430.000000
	mean	20.840000	51.640845	4.770580	31.604055
	std	3.361488	8.025086	3.390390	3.533436
	min	13.000000	42.000000	0.420000	27.000000
	25%	18.000000	45.000000	2.000000	29.699118
	50%	21.000000	50.000000	4.000000	29.699118
	75%	24.000000	56.000000	8.000000	34.000000
	max	26.000000	80.000000	12.000000	41.000000
Sex_male	count	250.000000	142.000000	69.000000	430.000000
	mean	0.620000	0.690141	0.536232	0.667442
	std	0.486360	0.464072	0.502339	0.471679
	min	0.000000	0.000000	0.000000	0.000000
	25%	0.000000	0.000000	0.000000	0.000000
	50%	1.000000	1.000000	1.000000	1.000000
	75%	1.000000	1.000000	1.000000	1.000000
	max	1.000000	1.000000	1.000000	1.000000
FamilyNb	count	250.000000	142.000000	69.000000	430.000000
	mean	0.744000	0.647887	3.275362	0.702326
	std	1.313532	1.105737	1.840173	1.573304