10/01/2022

Compte rendu du premier TP de data mining

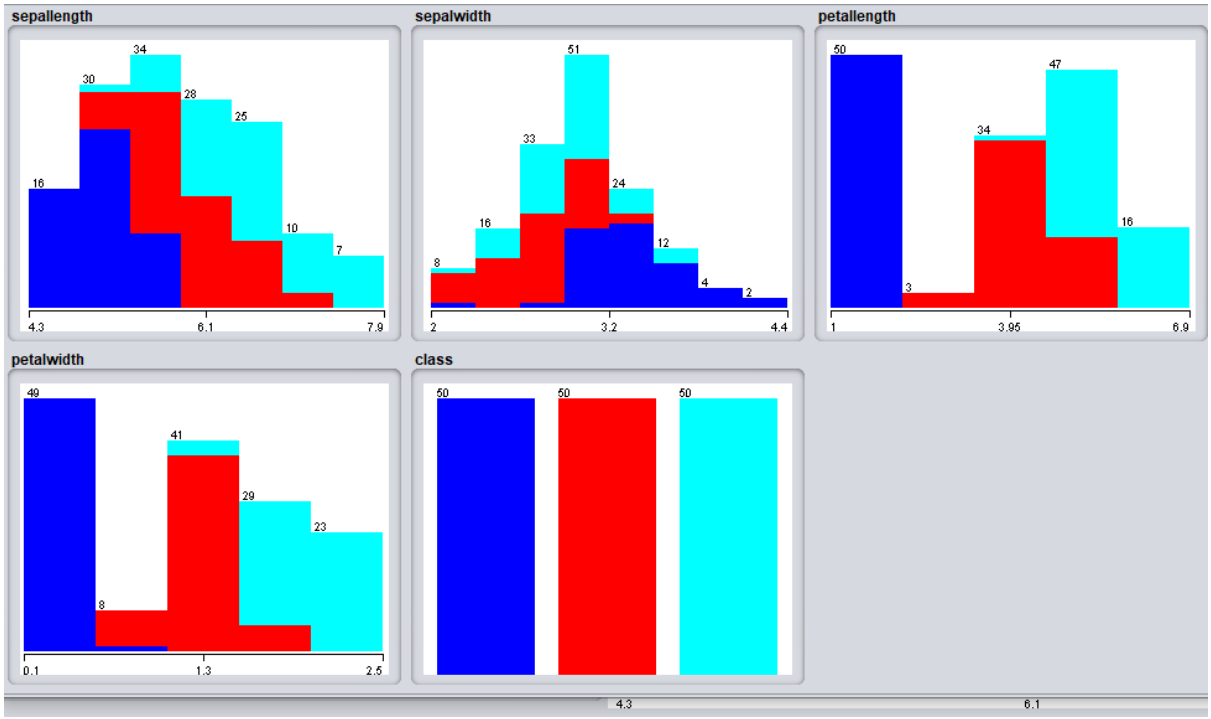
Clustering

FERHATI Amina
M2 DATASCLAE

Partie 1 :

Exploration des données :

- **Dans le menu explorer, j'ai repéré les différents éléments de l'interface :**
    - Taille : 150 instances
    - Nombre d'attributs : 5
    - Classes cibles : Class
    - Valeurs nulles : 0
    - Distribution des classes par attribut : On remarque que l'attribut classe a 3 valeurs : chacune d'entre elles a 50 instances, elles sont bien réparties. On remarque que l'attribut petallength est discriminatif (quand la valeur de cet attribut est entre 1 et 2.18, il y a une seule classe bleue), ainsi que petalwidth qui permet aussi de bien visualiser la répartition des classes.



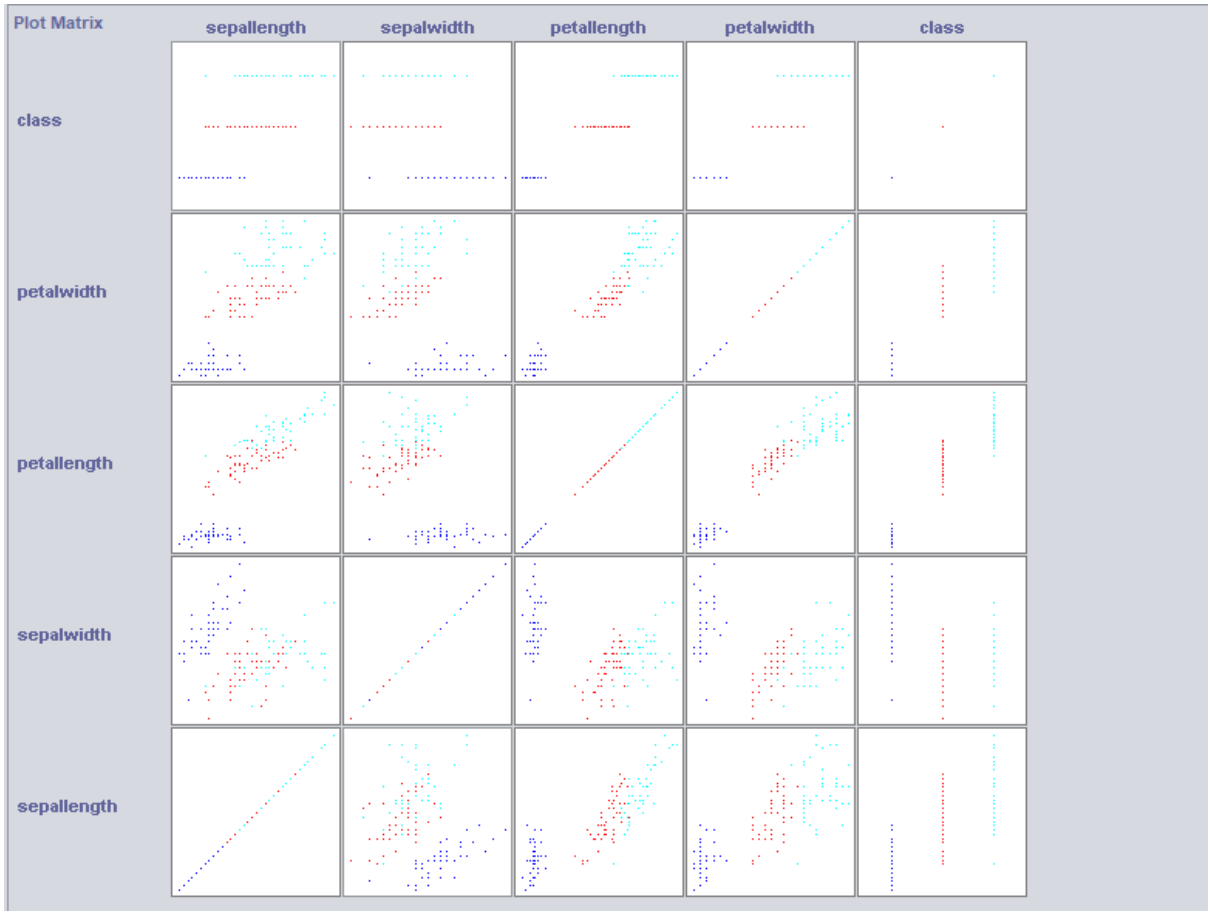
- **Utiliser le bouton « Edit » pour afficher le contenu au format tabulaire (sans l'éditer)**

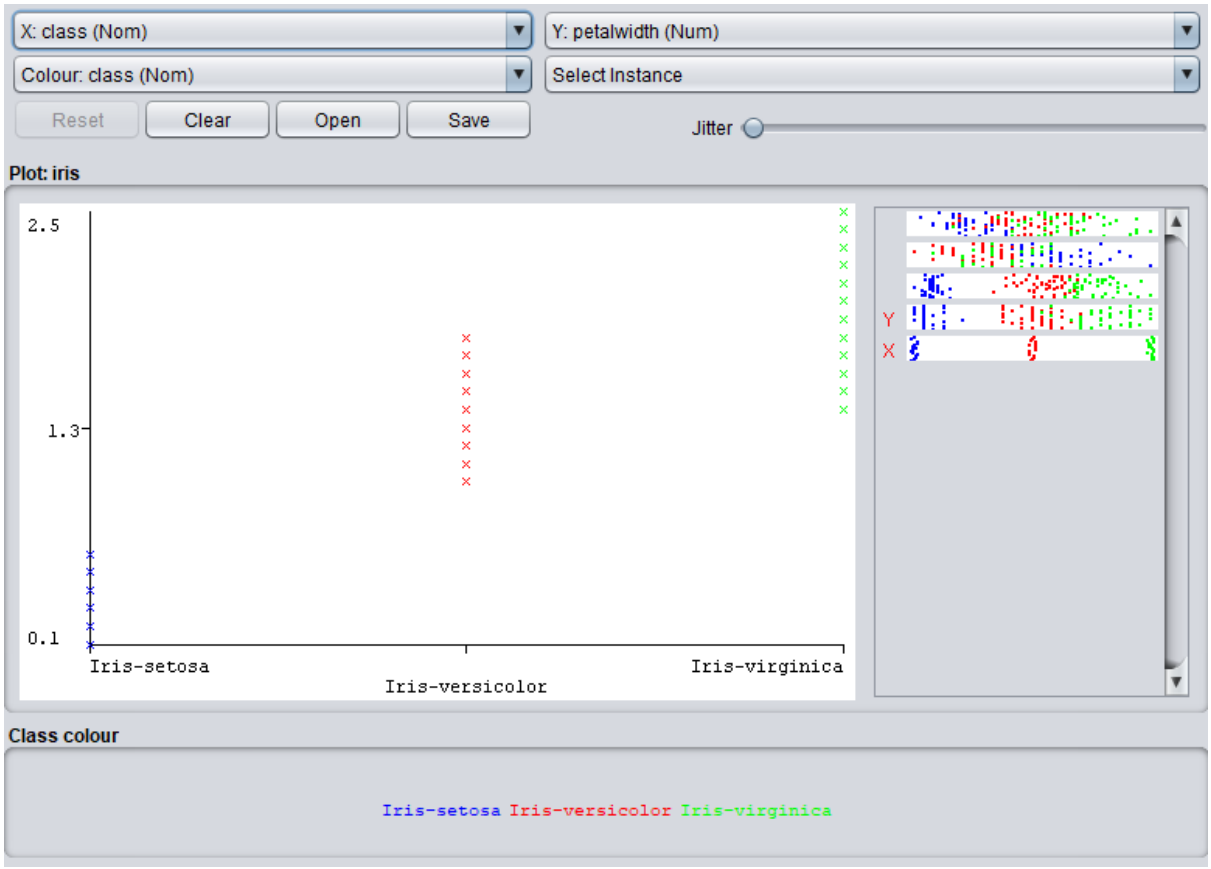
Le type des attributs est numérique à part pour l'attribut classe qui est catégorique (3 valeurs)

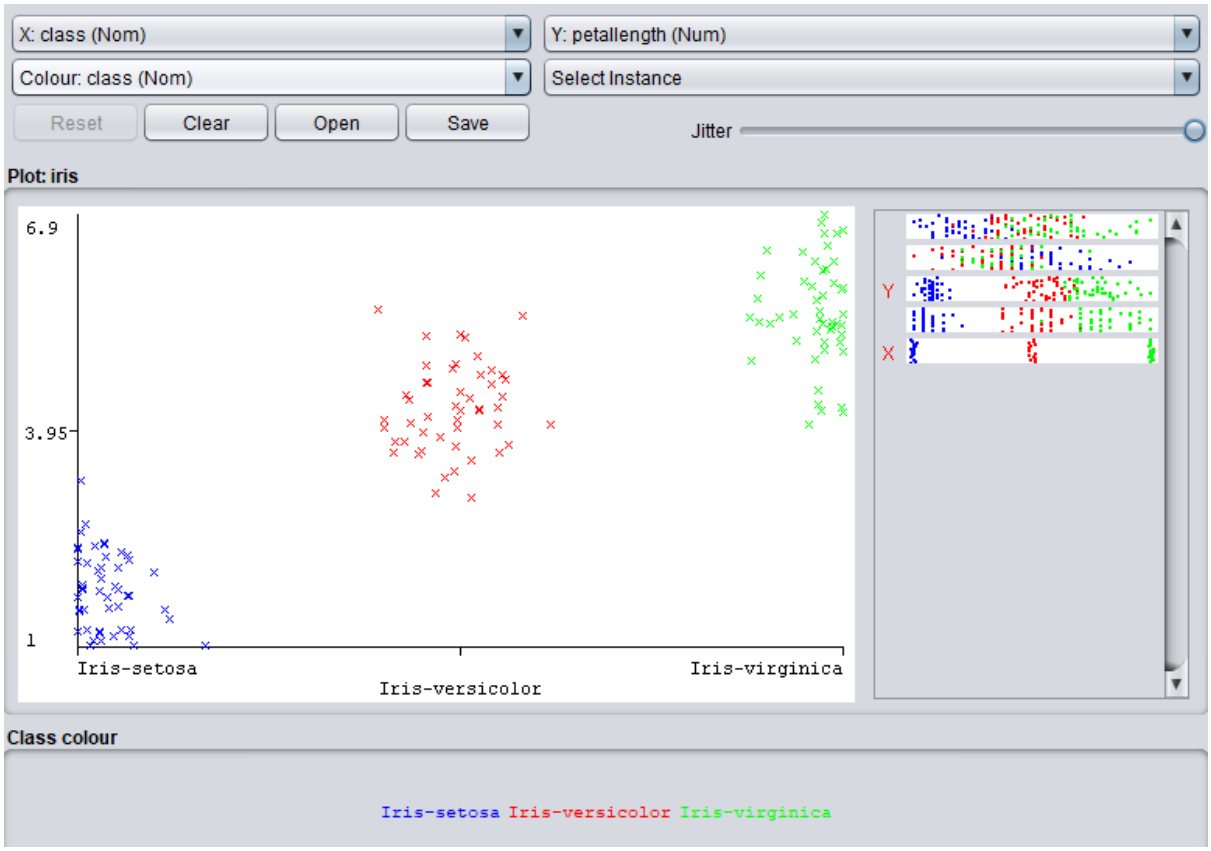
No.	1: sepallength Numeric	2: sepalwidth Numeric	3: petallength Numeric	4: petalwidth Numeric	5: **class** Nominal
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa
11	5.4	3.7	1.5	0.2	Iris-setosa
12	4.8	3.4	1.6	0.2	Iris-setosa
13	4.8	3.0	1.4	0.1	Iris-setosa
14	4.3	3.0	1.1	0.1	Iris-setosa
15	5.8	4.0	1.2	0.2	Iris-setosa
16	5.7	4.4	1.5	0.4	Iris-setosa
17	5.4	3.9	1.3	0.4	Iris-setosa
18	5.1	3.5	1.4	0.3	Iris-setosa
19	5.7	3.8	1.7	0.3	Iris-setosa
20	5.1	3.8	1.5	0.3	Iris-setosa
21	5.4	3.4	1.7	0.2	Iris-setosa
22	5.1	3.7	1.5	0.4	Iris-setosa
23	4.6	3.6	1.0	0.2	Iris-setosa
24	5.1	3.3	1.7	0.5	Iris-setosa
25	4.8	3.4	1.9	0.2	Iris-setosa
26	5.0	3.0	1.6	0.2	Iris-setosa
27	5.0	3.4	1.6	0.4	Iris-setosa
28	5.2	3.5	1.5	0.2	Iris-setosa
29	5.2	3.4	1.4	0.2	Iris-setosa

- **Explorer les données en utilisant l'onglet « Visualize ». Zoomez sur quelques plots de la ScatterMatrice.**

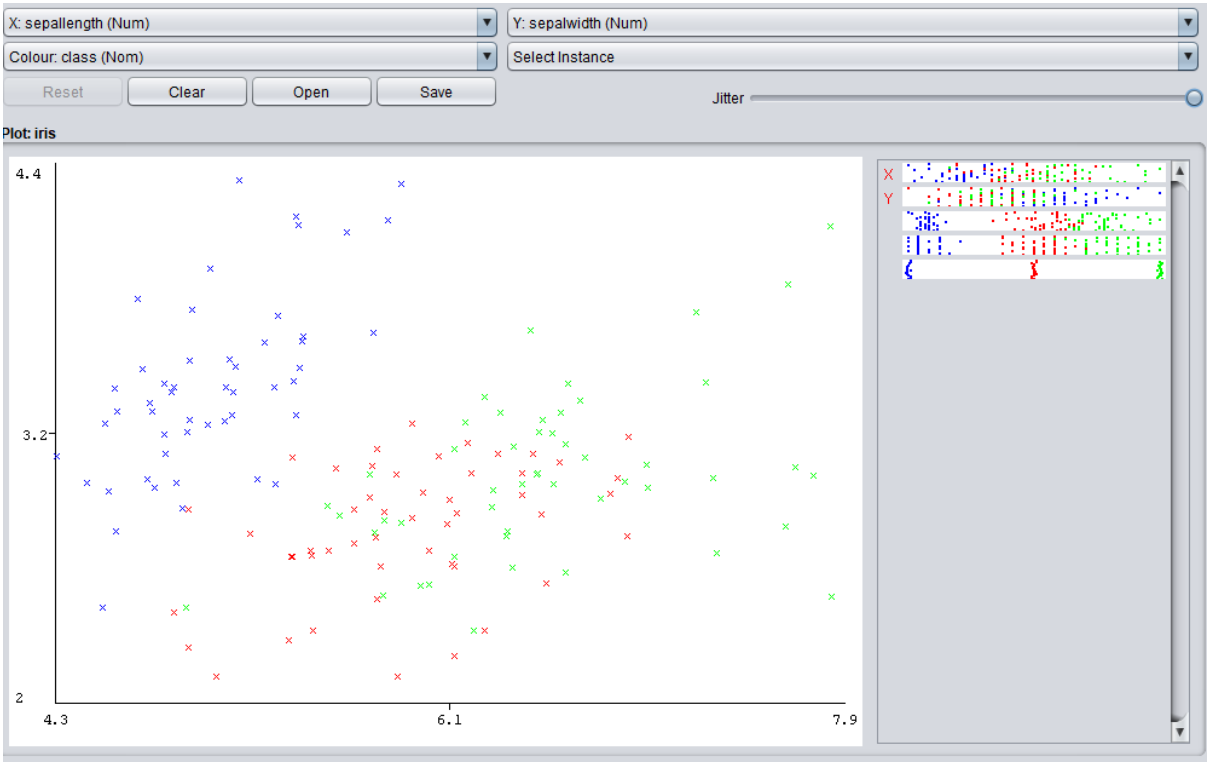
On remarque qu'il n'y a pas de corrélation entre les différentes variables par contre il y a une bonne répartition des classes quand les deux dimensions sont petallength et petalwidth.

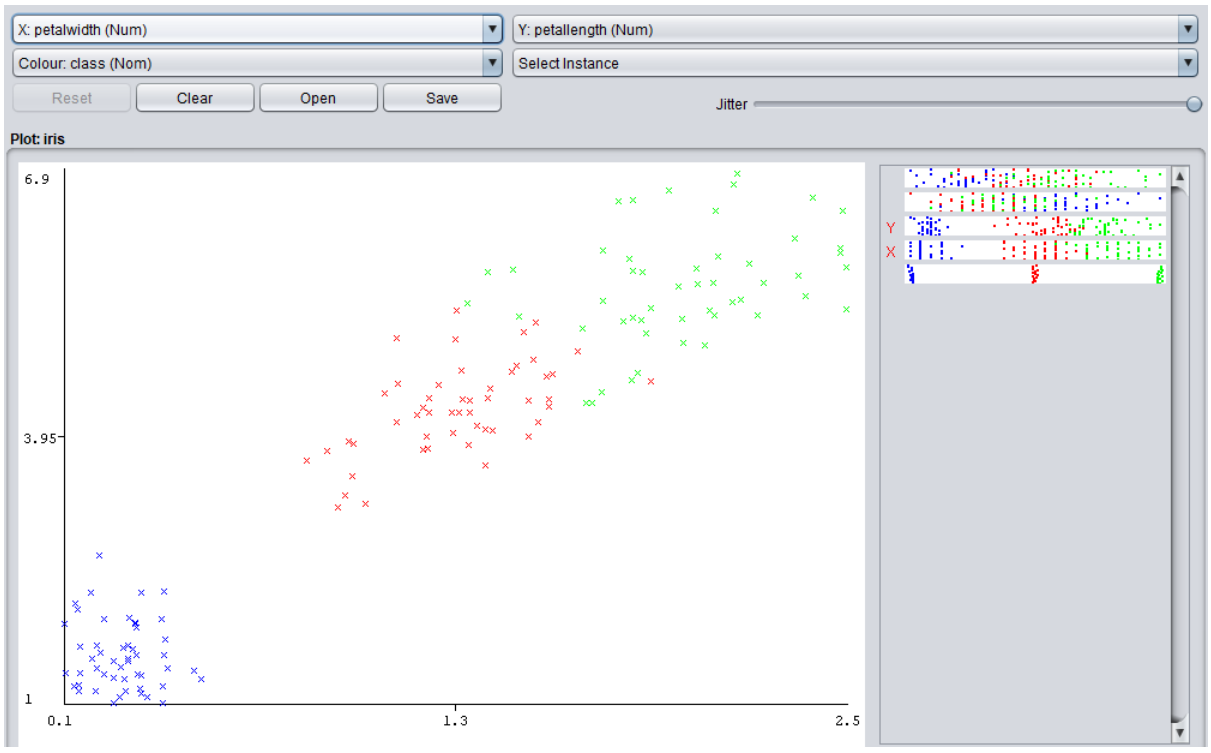

Plot Matrix
sepallength
sepalwidth
petallength
petalwidth
class
class
petalwidth
petallength
sepalwidth
sepallength


X: class (Nom)
Y: petalwidth (Num)
Colour: class (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: iris
2.5
1.3
0.1
Iris-setosa
Iris-versicolor
Iris-virginica
Y
X
Class colour
Iris-setosa Iris-versicolor Iris-virginica



- **<u>On peut aussi visualiser les 4 dimensions sur le même graphique avec Projection plot.</u>**





On peut bien remarquer que quand on a les deux dimensions petalwidth et petallength, on voit qu'il y a une bonne répartition des clusters, contrairement aux autres combinaisons de dimensions.

- **Peut-on déduire des connaissances par simple visualisation ?**

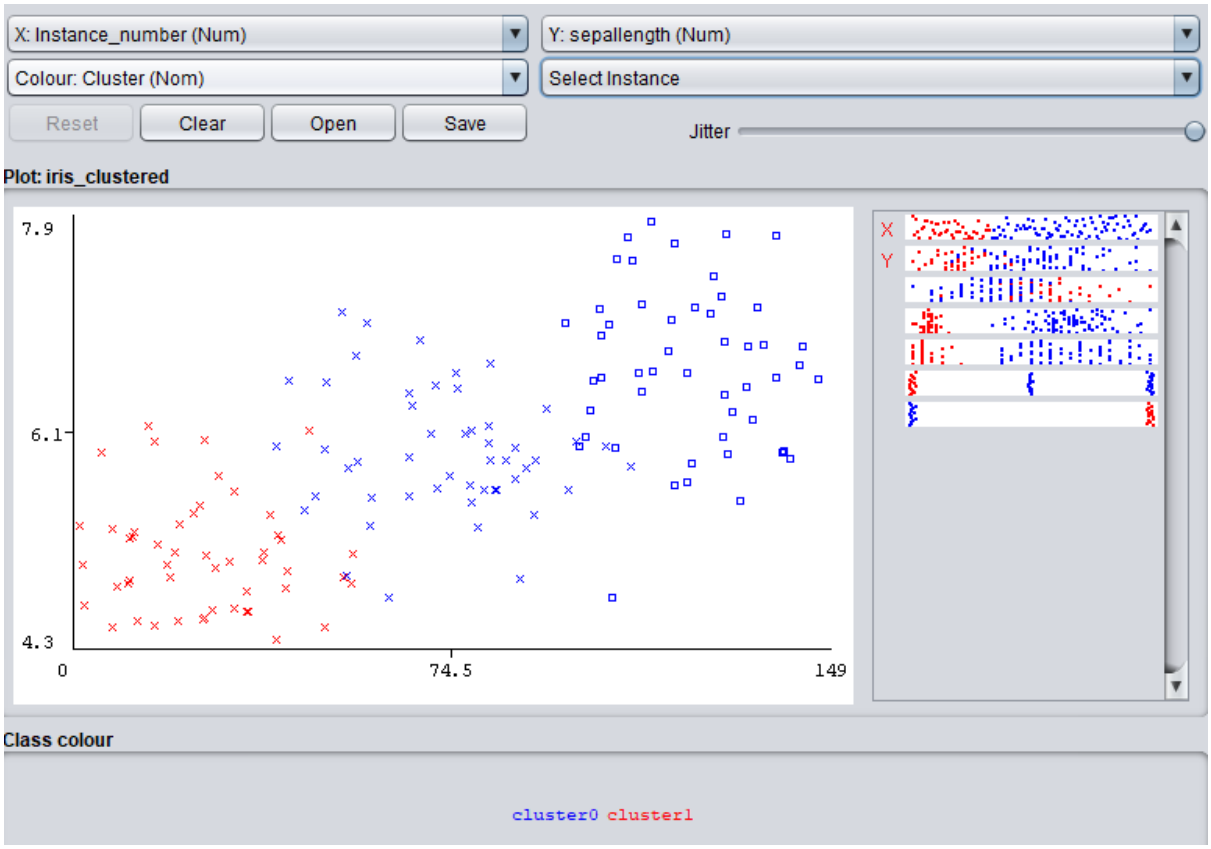
Oui, à partir des graphiques, on peut bien voir la corrélation entre les variables si elle existe ainsi que la pertinence des variables (celles qui aident à bien répartir les données en clusters).

Clustering

- **Appliquer une méthode de clustering (segmentation) K-means d'abord avec les options par défaut avec et la variable classe « Class to cluster evaluation »**

```
kMeans
======

Number of iterations: 7
Within cluster sum of squared errors: 12.143688281579722

Initial starting points (random):

Cluster 0: 6.1,2.9,4.7,1.4
Cluster 1: 6.2,2.9,4.3,1.3

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute      Full Data          0          1
                 (150.0)    (100.0)     (50.0)
================================================
sepallength       5.8433      6.262      5.006
sepalwidth         3.054      2.872      3.418
petallength       3.7587      4.906      1.464
petalwidth        1.1987      1.676      0.244
```

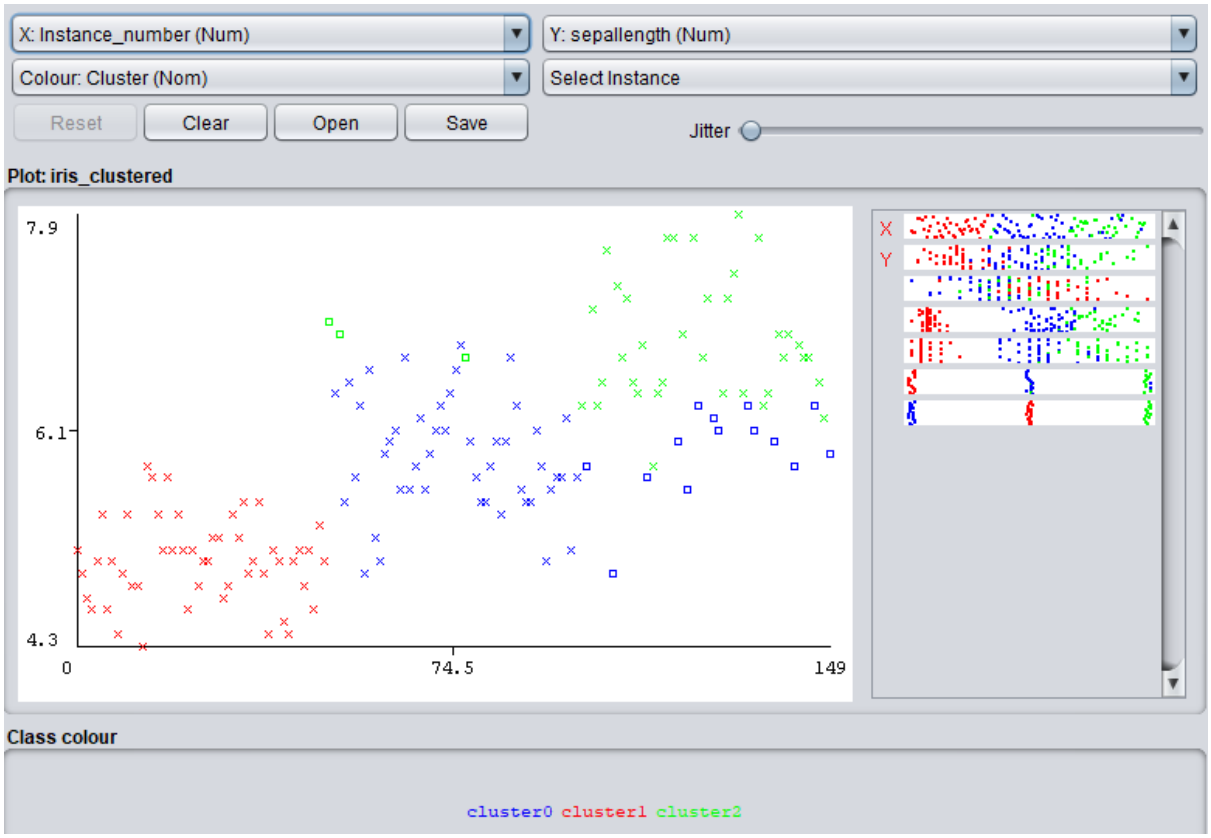
```
=== Model and evaluation on training set ===

Clustered Instances

0      100 ( 67%)
1       50 ( 33%)


Class attribute: class
Classes to Clusters:

  0  1  <-- assigned to cluster
  0 50 | Iris-setosa
 50  0 | Iris-versicolor
 50  0 | Iris-virginica

Cluster 0 <-- Iris-versicolor
Cluster 1 <-- Iris-setosa

Incorrectly clustered instances :       50.0     33.3333 %
```



On voit très bien que le modèle a construit uniquement 2 clusters, c'est pour cela que l'erreur est grande (50 entités mal classées).

- **Appliquer une méthode de clustering (segmentation) K-means d'abord avec l'option k=3avec et la variable classe « Class to cluster evaluation »**

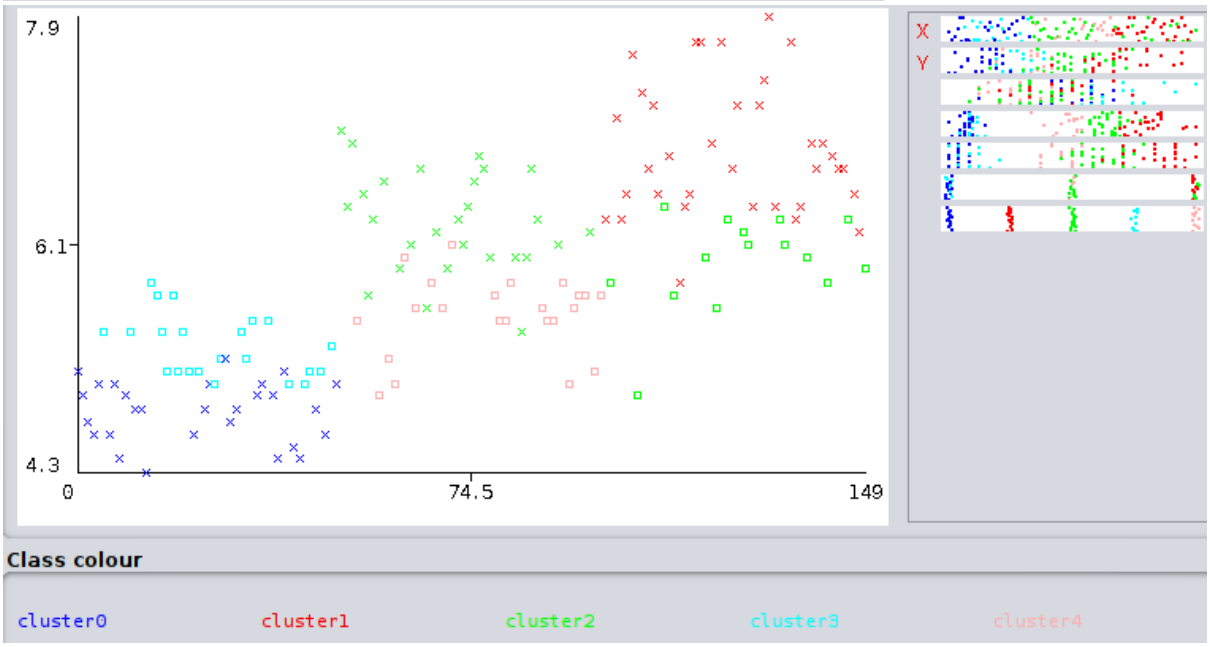
```
=== Model and evaluation on training set ===

Clustered Instances

0       61 ( 41%)
1       50 ( 33%)
2       39 ( 26%)


Class attribute: class
Classes to Clusters:

  0  1  2  <-- assigned to cluster
  0 50  0 | Iris-setosa
 47  0  3 | Iris-versicolor
 14  0 36 | Iris-virginica

Cluster 0 <-- Iris-versicolor
Cluster 1 <-- Iris-setosa
Cluster 2 <-- Iris-virginica

Incorrectly clustered instances :       17.0     11.3333 %
```



Vu qu'on a choisi le nombre de clusters à 3, l'erreur a baissé de 30% à 11%. On peut voir sur le graphique que les carrés bleus sont censés être attribués au cluster 2 mais l'algorithme les a classés dans le cluster 0.

- **Dans un deuxième temps, utiliser la méthode de clustering EM avec les options par défaut.**

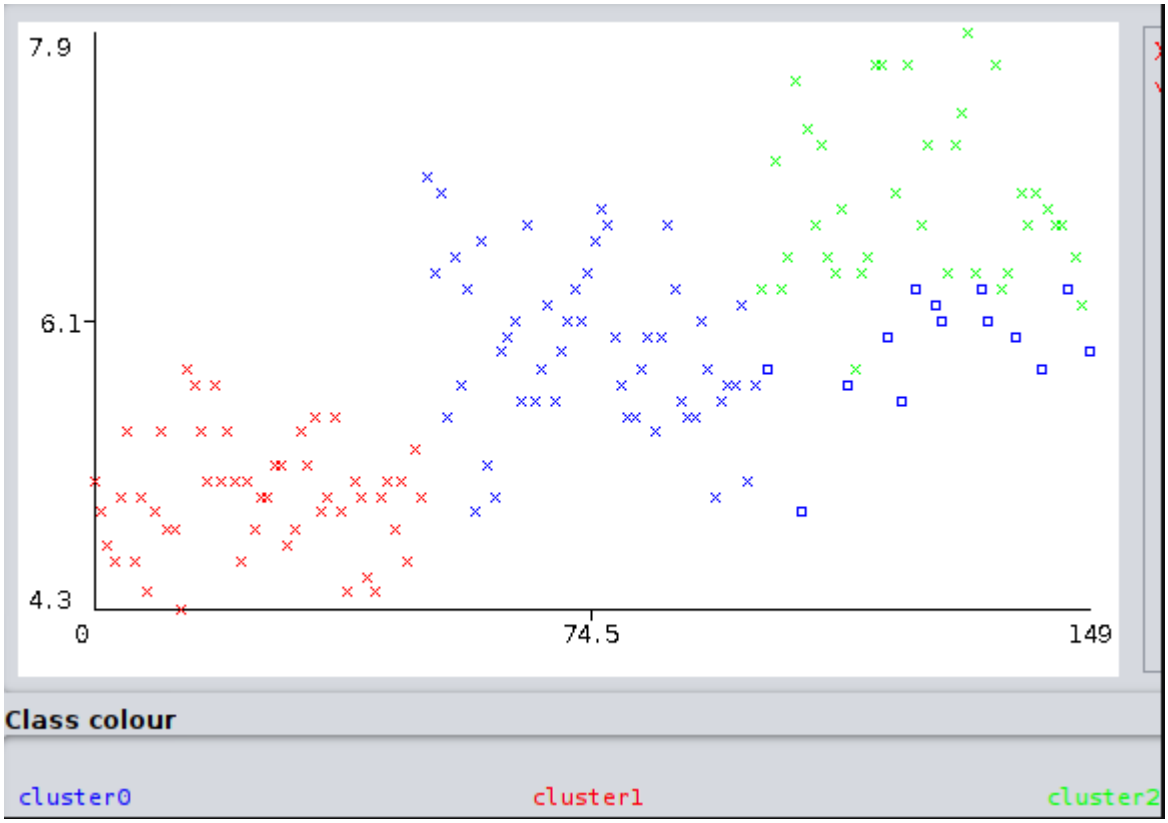
```
=== Model and evaluation on training set ===

Clustered Instances

0       28 ( 19%)
1       35 ( 23%)
2       42 ( 28%)
3       22 ( 15%)
4       23 ( 15%)


Log likelihood: -1.60803


Class attribute: class
Classes to Clusters:

  0  1  2  3  4  <-- assigned to cluster
 28  0  0 22  0 | Iris-setosa
  0  0 27  0 23 | Iris-versicolor
  0 35 15  0  0 | Iris-virginica

Cluster 0 <-- Iris-setosa
Cluster 1 <-- Iris-virginica
Cluster 2 <-- Iris-versicolor
Cluster 3 <-- No class
Cluster 4 <-- No class

Incorrectly clustered instances :       60.0      40       %
```



Avec les options par défaut de l'algorithme, on obtient 5 clusters, c'est pour cela que l'erreur est plus grande (40%) que celle de Kmeans avec les options par défauts.

- **Changer le nombre de clusters dans EM et comparez à nouveau les résultats de EM avec Kmeans.**

```
=== Model and evaluation on training set ===

Clustered Instances

0       64 ( 43%)
1       50 ( 33%)
2       36 ( 24%)


Log likelihood: -2.055


Class attribute: class
Classes to Clusters:

  0  1  2  <-- assigned to cluster
  0 50  0 | Iris-setosa
 50  0  0 | Iris-versicolor
 14  0 36 | Iris-virginica

Cluster 0 <-- Iris-versicolor
Cluster 1 <-- Iris-setosa
Cluster 2 <-- Iris-virginica

Incorrectly clustered instances :       14.0      9.3333 %
```



Après avoir mis le nombre de clusters à 3, on voit que le modèle a fait une bonne classification car l'erreur est uniquement de 14%.

- **Comparaison de EM et Kmeans :**

  - L'algorithme EM est une alternative solide au traditionnel clustering à kmeans car il produit des solutions stables en trouvant des distributions gaussiennes multivariées pour chaque cluster.
  - Le processus de K-Means consiste à assigner chaque observation à un cluster et le processus de EM (Expectation Maximization) consiste à trouver la probabilité d'une observation appartenant à un cluster (probabilité). C'est là que ces deux processus diffèrent
  - EM et K-means sont similaires dans le sens où ils permettent d'affiner le modèle d'un processus itératif pour trouver la meilleure congestion. Cependant, l'algorithme K-means diffère par la méthode utilisée pour calculer la distance euclidienne lors du calcul de la distance entre chacune de deux données ; et EM utilise des méthodes statistiques.

En conclusion, l'algorithme EM offre une alternative puissante au Kmeans avec un meilleur contrôle des caractéristiques du cluster.

- **Xmeans**

```
=== Model and evaluation on training set ===

Clustered Instances

0      100 ( 67%)
1       50 ( 33%)


Class attribute: class
Classes to Clusters:

  0  1  <-- assigned to cluster
  0 50 | Iris-setosa
 50  0 | Iris-versicolor
 50  0 | Iris-virginica

Cluster 0 <-- Iris-versicolor
Cluster 1 <-- Iris-setosa

Incorrectly clustered instances :       50.0     33.3333 %
```

Ce modèle n'a pas pu trouver le nombre de cluster correct, il a classé les données en 2 classes, c'est pour cela qu'on a 50 instances mal classées

Partie 2

Préparation des données

- **Se débarrasser des colonnes qui n'ont pas de valeur dans le dataset :**

Dans notre cas, la colonne id n'aide pas à faire la classification des données.

- **Gérer les valeurs spéciales et nulles :**
  - Pour s'assurer que toutes les colonnes contiennent des valeurs numériques, on exécute cette commande.

```
num_df = (df.drop(columns, axis=1).join(df[columns].apply(pd.to_numeric, errors='coerce')))
```

  - Le résultat de la commande

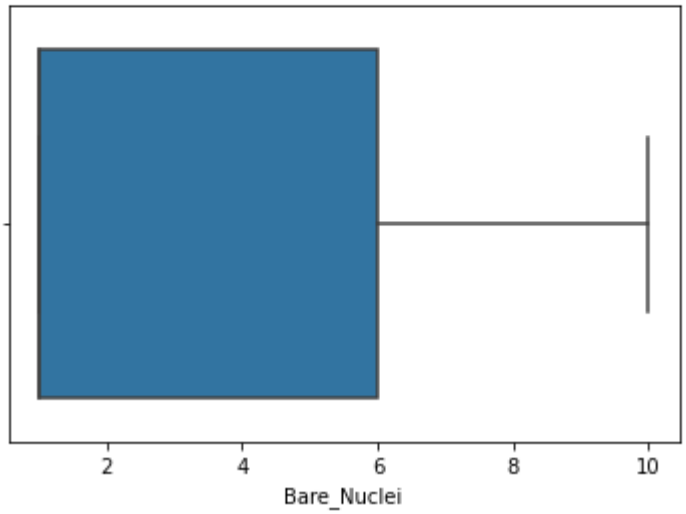
```
num_df.isna().sum()
```

```
Clump_Thickness                0
Uniformity_of_Cell_Size        0
 Uniformity_of_Cell_Shape      0
Marginal_Adhesion              0
Single_Epithelial_Cell_Size    0
Bare_Nuclei                   16
Bland_Chromatin                0
Normal_Nucleoli                0
Mitoses                        0
Class                          0
dtype: int64
```

  - On remarque qu'il y a 16 valeurs nulles, ces valeurs ont été obtenues après avoir transformé '?' en 'nan' dans la colonne Bare_Nuclei.
  - Vu que le nombre de lignes du dataset est petit, on va essayer de remplacer ces valeurs nulles par la moyenne ou bien la médiane, ….
  - Pour décider quelle technique à utiliser on va afficher les graphiques suivants :

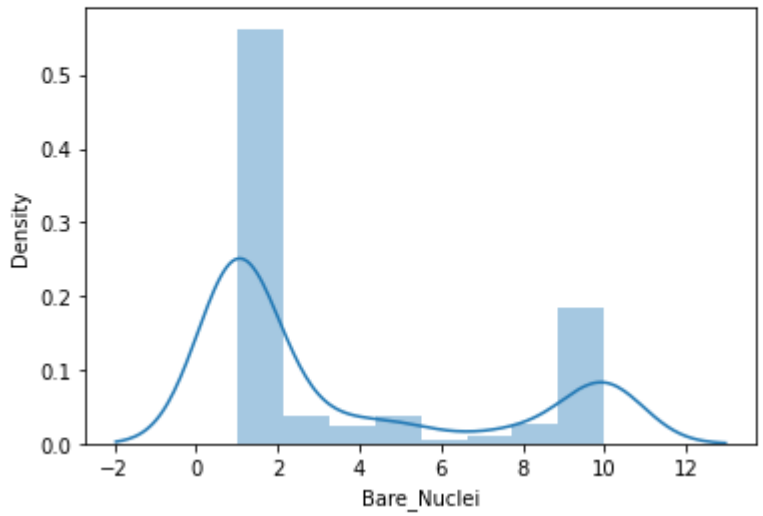
```
import seaborn as sns
# Box plot
sns.boxplot(num_df.Bare_Nuclei)
```

```
<AxesSubplot:xlabel='Bare_Nuclei'>
```




```
# Distribution plot
sns.distplot(num_df.Bare_Nuclei)
```

```
<AxesSubplot:xlabel='Bare_Nuclei', ylabel='Density'>
```



- Lorsque la distribution des données est asymétrique, il est bon d'envisager d'utiliser la valeur médiane pour remplacer les valeurs manquantes.
- Après avoir remplacé ces valeurs par la médiane, on remarque qu'il y a plus de valeurs nulles.

```
: # Remplacer en utilisant la médiane
  median = num_df['Bare_Nuclei'].median()
  num_df['Bare_Nuclei'].fillna(median, inplace=True)
```

```
: num_df.isna().sum()
```

```
Clump_Thickness               0
Uniformity_of_Cell_Size       0
 Uniformity_of_Cell_Shape     0
Marginal_Adhesion             0
Single_Epithelial_Cell_Size   0
Bare_Nuclei                   0
Bland_Chromatin               0
Normal_Nucleoli               0
Mitoses                       0
Class                         0
dtype: int64
```

- **<u>Comprendre les données par des résumés statistiques :</u>**

On affiche la moyenne et la médiane et les quantiles :

```
median = num_df.median()
median
```

```
Clump_Thickness                4.0
Uniformity_of_Cell_Size        1.0
 Uniformity_of_Cell_Shape      1.0
Marginal_Adhesion              1.0
Single_Epithelial_Cell_Size    2.0
Bare_Nuclei                    1.0
Bland_Chromatin                3.0
Normal_Nucleoli                1.0
Mitoses                        1.0
Class                          2.0
dtype: float64
```

```
mean= num_df.mean()
mean
```

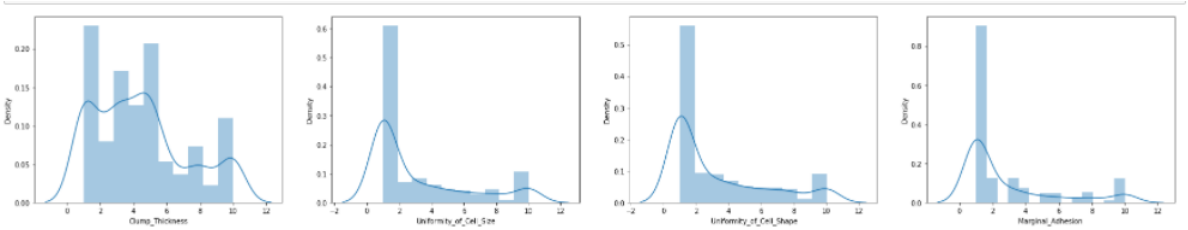
```
Clump_Thickness                4.417740
Uniformity_of_Cell_Size        3.134478
 Uniformity_of_Cell_Shape      3.207439
Marginal_Adhesion              2.806867
Single_Epithelial_Cell_Size    3.216023
Bare_Nuclei                    3.486409
Bland_Chromatin                3.437768
Normal_Nucleoli                2.866953
Mitoses                        1.589413
Class                          2.689557
dtype: float64
```

```
#Summary statistics:

num_df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Clump_Thickness	699.0	4.417740	2.815741	1.0	2.0	4.0	6.0	10.0
Uniformity_of_Cell_Size	699.0	3.134478	3.051459	1.0	1.0	1.0	5.0	10.0
Uniformity_of_Cell_Shape	699.0	3.207439	2.971913	1.0	1.0	1.0	5.0	10.0
Marginal_Adhesion	699.0	2.806867	2.855379	1.0	1.0	1.0	4.0	10.0
Single_Epithelial_Cell_Size	699.0	3.216023	2.214300	1.0	2.0	2.0	4.0	10.0
Bare_Nuclei	699.0	3.486409	3.621929	1.0	1.0	1.0	5.0	10.0
Bland_Chromatin	699.0	3.437768	2.438364	1.0	2.0	3.0	5.0	10.0
Normal_Nucleoli	699.0	2.866953	3.053634	1.0	1.0	1.0	4.0	10.0
Mitoses	699.0	1.589413	1.715078	1.0	1.0	1.0	1.0	10.0
Class	699.0	2.689557	0.951273	2.0	2.0	2.0	4.0	4.0

- On peut voir clairement que les données ne suivent pas une loi normale.

- **<u>Comprendre la distribution des données en utilisant des résumés graphiques :</u>**
  - On peut confirmer aussi cela en utilisant des histogrammes et les courbes de distributions



- On peut aussi vérifier la distribution avec Skewness

```
#skewness in the data
num_df.skew()
```

```
Clump_Thickness                0.592859
Uniformity_of_Cell_Size        1.233137
 Uniformity_of_Cell_Shape      1.161859
Marginal_Adhesion              1.524468
Single_Epithelial_Cell_Size    1.712172
Bare_Nuclei                    1.025347
Bland_Chromatin                1.099969
Normal_Nucleoli                1.422261
Mitoses                        2.961465
Class                          0.654564
dtype: float64
```

- Les variables avec -0,5 < skewness < 0,5 sont symétriques, c'est-à-dire normalement distribuées. C'est pour cela, j'ai testé plusieurs méthodes pour rendre la distribution des variables normale comme le log, la racine carrée et le standard Scaler et la meilleure méthode était le logarithmique.

```
#performing logarithmic transformation on the feature
#num_df.Clump_Thickness=np.log(num_df.Clump_Thickness)
num_df[' Uniformity_of_Cell_Shape']=np.log(num_df[' Uniformity_of_Cell_Shape'])
num_df.Bland_Chromatin=np.log(num_df.Bland_Chromatin)

num_df.Uniformity_of_Cell_Size=np.log(num_df.Uniformity_of_Cell_Size)
num_df.Marginal_Adhesion =np.log(num_df.Marginal_Adhesion )
num_df.Single_Epithelial_Cell_Size=np.log(num_df.Single_Epithelial_Cell_Size)
num_df.Bare_Nuclei =np.log(num_df.Bare_Nuclei )
num_df.Normal_Nucleoli=np.log(num_df.Normal_Nucleoli)
num_df.Mitoses=np.log(num_df.Mitoses)
```

```
#skewness in the data
num_df.skew()
```

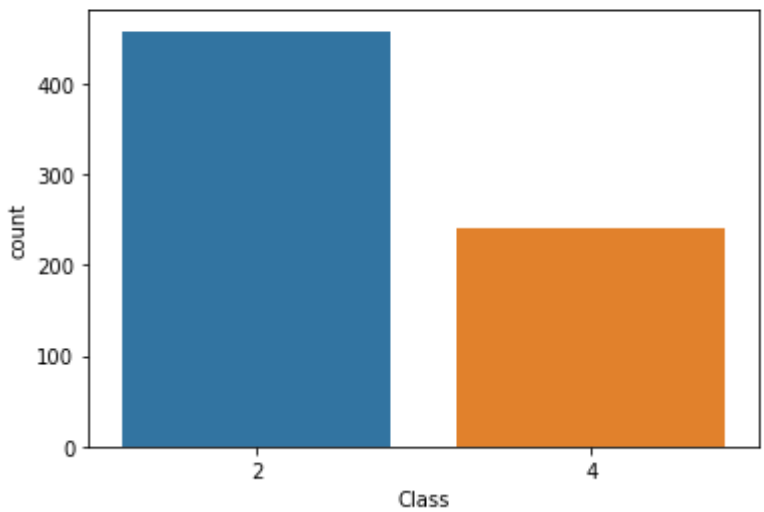
```
Clump_Thickness               -0.449422
Uniformity_of_Cell_Size        0.659280
 Uniformity_of_Cell_Shape      0.537375
Marginal_Adhesion              0.878514
Single_Epithelial_Cell_Size    0.711588
Bare_Nuclei                    0.721701
Bland_Chromatin                0.102300
Normal_Nucleoli                0.963659
Mitoses                        2.462636
Class                          0.654564
dtype: float64
```

- **Voir si les classes sont équilibrées :**

La répartition est quasi-équilibrée.

```
#la densité de notre variable cible
sns.countplot(num_df["Class"])
```

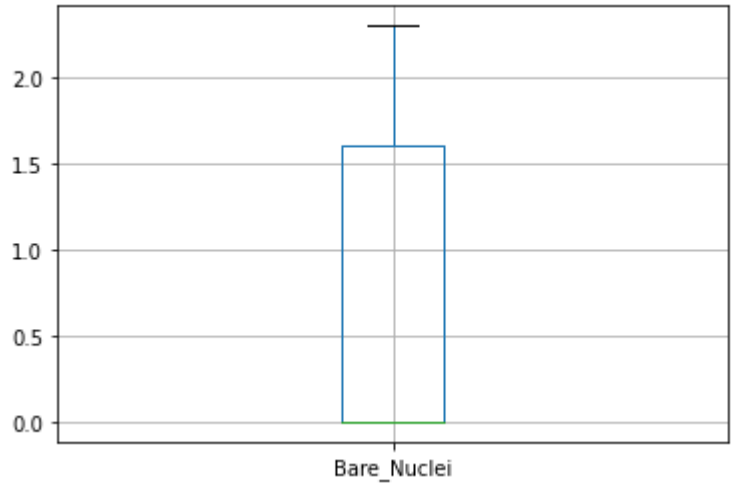
```
<AxesSubplot:xlabel='Class', ylabel='count'>
```



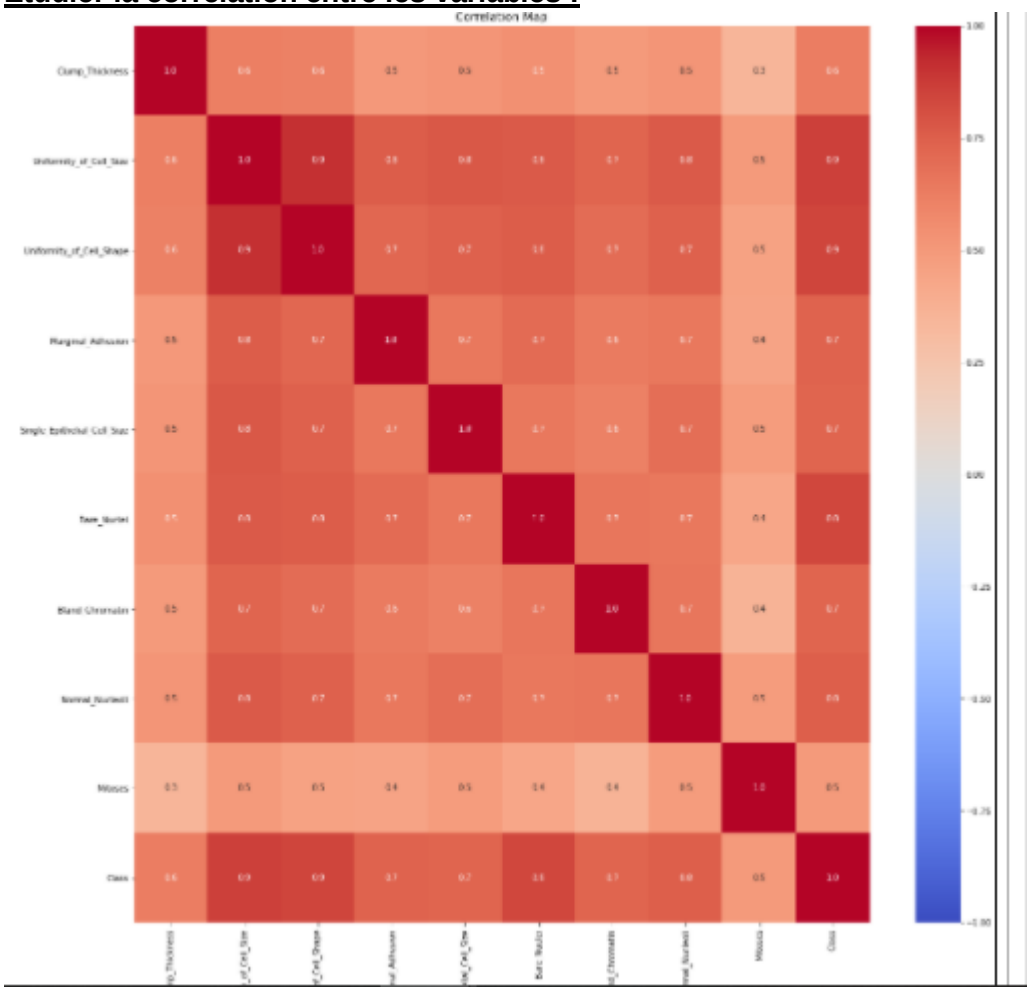
- **Détecter les outliers (éléments aberrants) :**

Pour cela on va afficher les boîtes à moustaches pour les différentes variables, on peut bien remarquer que toutes les variables n'ont pas des outliers après avoir appliqué la fonction

logarithmique à part pour la variable Mitoses qui a la majorité de ses valeurs à 1, donc le reste est détecté comme outlier.



- **Etudier la corrélation entre les variables :**



Mis à part la diagonale on peut voir les variables qui ont un coefficient de corrélation élevé :

- Uniformity of Cell Size, Uniformity of Cell Shape à 0.9 et Uniformity of Cell Size,Bland Chromatin,Single Epithelial Cell Size ... à 0.8. Ainsi, on peut aussi faire des ensembles de variables corrélées (on peut considérer que 2 variables ayant un coeff

supérieur à 0.8 sont corrélées), et décider d'en prendre une seule dans chaque ensemble pour la suite de l'analyse.

- Vu qu'on a peu de dimensions dans ce dataset, on va laisser toutes les variables telles qu'elles sont et on ne va pas réduire la dimension du dataset (par colonne).

- **Déterminer la mesure utilisée pour l'évaluation des modèles :**

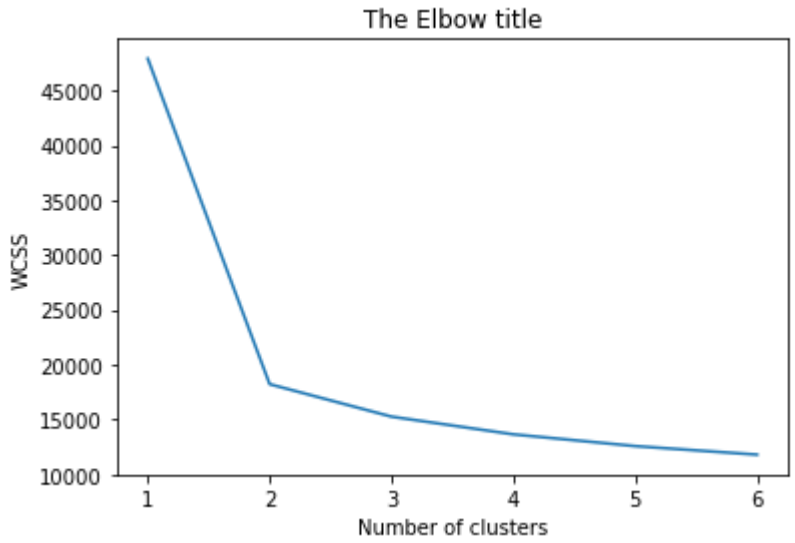
L'indice de Rand calcule une mesure de similarité entre deux clusters en considérant toutes les paires d'échantillons et en comptant les paires qui sont assignées dans les mêmes ou différents clusters dans les clusters prédits et réels.

Dans notre cas, on a utilisé **l'indice de Rand ajusté** est la version corrigée pour le hasard (la chance) de l'indice de Rand. Une telle correction pour le hasard établit une base de référence en utilisant la similarité attendue de toutes les comparaisons par paires entre des regroupements spécifiés par un modèle aléatoire

- **Premier algorithme Kmeans :**

Le k-means est une méthode d'analyse de clusters utilisant un nombre prédéfini de clusters. Elle nécessite une connaissance préalable de "K".

- Pour déterminer le nombre de cluster qu'il faut, j'ai utilisé la méthode elbow et j'ai obtenu deux clusters (la pointe du coude)



- Après avoir fait cela, j'ai réparti les données en train dataset (80%) et test dataset (20%) et j'ai utilisé le paramètre Stratify afin de garder la répartition équilibrée entre les deux classes.

```
#SPLIT
X_train, X_test, y_train, y_test = train_test_split(X_final,y_final, train_size=0.8, random_state = 1,stratify=y_final)
```

- Voici les résultats obtenus

```
kmeans=KMeans(n_clusters=2)
```

```
kmeans.fit(X_train,y_train)
```

```
KMeans(n_clusters=2)
```

```
y_pred=kmeans.predict(X_test)
```

```
from sklearn.metrics.cluster import adjusted_rand_score
adjusted_rand_score(y_test,y_pred)
```

```
0.7557757024833293
```

- **Deuxième algorithme est agglomerative clustering**

Le clustering hiérarchique, est également une méthode d'analyse de cluster qui cherche à construire une hiérarchie de clusters sans avoir un nombre fixe de clusters.

- Les paramètres choisis pour cet algorithme sont : le nombre de clusters qui est à 2, le linkage est ward qui minimise la variance des clusters fusionnés et l'affinity euclidienne qui est la métrique utilisée pour calculer le linkage.
- Voici les résultats de cet algorithme :

7- Agglomerative clustering

```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
```

```
cluster.fit(X_train,y_train)
```

```
AgglomerativeClustering()
```

```
y_pred2=cluster.fit_predict(X_test)
```

```
adjusted_rand_score(y_test,y_pred2)
```

```
0.7304895667992124
```

- **Comparaison des deux algorithmes :**

- Avantages de Kmeans : La convergence est garantie et il est spécialisé pour les clusters de différentes tailles et formes.
- Avantages de Agglomerative clustering : Facilité de traitement de toute forme de similitude ou de distance. Par conséquent, applicabilité à tous les types d'attributs.
- Inconvénient de Kmeans : la valeur K est parfois difficile à être prédite.

- Inconvénient de Agglomerative clustering : Le clustering hiérarchique nécessite le calcul et le stockage d'une matrice de distance n×n. Pour les très grands ensembles de données, cela peut être coûteux et lent.

Dans notre cas, il est facile de détecter le nombre de clusters (2) vu qu'on l'a déjà et le jeu de données est petit donc le Agglomerative clustering n'est pas couteaux. On a aussi les deux indices entre 0,65 =< ARI < 0,80 donc c'est moderated recovery. Cependant, avec les résultats de Rand index, clairement le k-means est le meilleur algorithme pour le dataset.