

## Table des matières

<b>1</b>	<b>Description des données</b>	<b>1</b>
<b>2</b>	<b>Méthode de travail</b>	<b>2</b>
2.1	Filtrage des données . . . . .	3
2.1.1	Réduction des dimensions . . . . .	3
2.1.2	Valeurs manquantes . . . . .	4
2.1.3	Outliers . . . . .	4
2.2	Clustering . . . . .	7
2.2.1	Détermination du nombre de <i>clusters</i> . . . . .	7
2.3	Clustering . . . . .	8
2.3.1	Normalisation . . . . .	8
2.4	Conclusion . . . . .	9
<b>3</b>	<b>Secteurs d'activité des différents pays</b>	<b>10</b>
3.1	Sélection des données . . . . .	10
3.2	Clustering . . . . .	10
3.2.1	Clustering hiérarchique . . . . .	10
3.2.2	Clustering <i>K-Means</i> . . . . .	10
3.3	Interprétation . . . . .	10
3.3.1	Arbre de décision . . . . .	10
<b>4</b>	<b>Création d'un module de vérification du <i>K-Mean</i></b>	<b>11</b>
4.1	Principe de l'algorithme . . . . .	11
4.2	Implémentation . . . . .	11
4.3	Axes d'amélioration . . . . .	13

## 1 Description des données

Le jeu de données provient du site <http://worldbank.org>. Il concerne 210 pays et fournit des informations comme :

- Le pourcentage du **Revenu National Brute (RNB)** généré par l'agriculture, l'industrie ou les services.
- La quantité de produits importés et exportés.
- Des informations démographiques et géographiques comme la densité et l'âge de la population ou la surface du pays.
- Des informations sanitaires comme la quantité de personnes infectées par le **Virus d'immunodéficience Humaine (VIH)**.
- Des centaines d'autres indicateurs.

Par ailleurs, ce jeu de donnée semble être un très bon entraînement pour nous exercer au *datamining*. En effet, en plus d'exhiber des données très concrètes pour nous, il est évident que des corrélations sont présentes.

Enfin, il pourra être très intéressant d'observer ces corrélations d'abord à l'échelle mondiale, puis à l'échelle de l'Europe par exemple.

## 2 Méthode de travail

Cette partie va tenter de décrire très précisément la démarche de travail que nous avons établi au début de ce TP, notamment pendant les deux séances préparatoires. En plus de vous indiquer quelles sont les étapes que nous avons suivies pour arriver à nos résultats, elles constitueront une base de travail solide que nous pourrions améliorer tout au long des deux séances de *datamining*. Enfin, elle seront à nos yeux un indicateur important quant à la rigueur de notre travail et pourra, après correction nous être d'une grande aide.

## 2.1 Filtrage des données

Contrairement à ce que nous avons cru avant de commencer, il faut très souvent élaguer son *dataset* afin de donner plus de sens aux valeurs. Étudions les différentes phases et techniques associées à ce pré-traitement.

### 2.1.1 Réduction des dimensions

Une des problématique majeure du *datamining* s'appelle la *malédiction de la dimensionnalité*. Cette « malédiction » provient du fait que plus le nombre de dimensions (*i.e.* le nombre de colonnes) du jeu de donnée croît, moins la notions de distance<sup>1</sup> n'a de sens. En effet, les distances ont tendance à se réduire proportionnellement avec le nombre de dimensions. Or, les algorithmes de *clustering* utilisent cette distance comme critère principal de choix à l'appartenance à tel ou tel *cluster*. Les résultats risquent donc fortement de perdre en sens et en précision.

Voyons alors comment nous pouvons, dans le cadre d'un jeu de donnée contenant beaucoup de colonne, palier à ce défi technique.

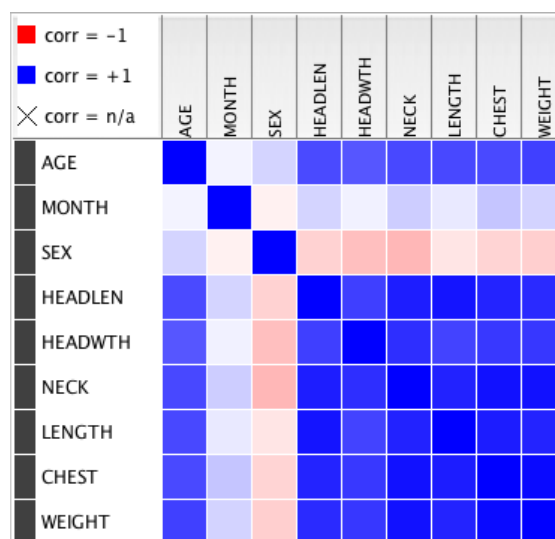
#### Choix des colonnes

Dans un premier temps, il est fondamental de ne sélectionner que les colonnes qui nous intéressent.

Dans le cadre d'une étude de *datamining* réalisée par des professionnels, l'expérience dans le domaine analysé joue beaucoup et permettra de faire un choix cohérent et judicieux des dimensions à conserver. Cependant, il est fréquent que les données ne soient pas dans le domaine d'expérience du chercheur ou pire, très abstraites. Dans ce cas, il faut faire appel à d'autre techniques.

#### Correlation

Il peut être très intéressant d'utiliser une matrice de corrélation afin de mettre en valeurs les colonnes qui pourraient nous être utiles, et surtout celles qui nous sont inutiles.



La lecture de cette matrice est très simple. Plus les couleurs (rouge ou bleu) sont foncées, plus la corrélation (*i.e.* la facilité à déduire les données d'une colonne à partir d'une autre) entre les colonnes est forte.

1. Quelque soit la fonction de distance utilisée (Manhattan, Euclidienne, Sebestyen, *etc.*)

Il est évident que les diagonales soient de la couleur la plus foncée possible car une colonne est toujours corrélée avec elle même.

Dans cet exemple, il est facile de déduire de la matrice que le sexe et le mois de mesure n'ont finalement pas beaucoup d'impacte sur les autres paramètres.

Nous pouvons donc les ignorer dans notre *clustering*.

## PCA

La [Primary Component Analysis \(PCA\)](#) est une méthode de réduction automatique des dimensions. Elle a l'avantage de préserver *mathématiquement* le plus d'information possible. Pour cela, elle essaie de chercher la meilleure combinaison linéaire des différentes colonnes afin d'atteindre, au choix : *un nombre fixé de dimension* ou *une conservation de X% de l'information*.

Bien que pratique, elle est (tout du moins à notre niveau d'expertise en *datamining*) relativement dangereuse car les combinaisons linéaires qui résultent d'une [PCA](#) perdent beaucoup de leur sémantique. Aussi, il est assez compliqué d'obtenir la formule de sortie (*i.e.* l'application linéaire qui a été définie par la [PCA](#)) afin de l'interpréter.

### 2.1.2 Valeurs manquantes

Certains *dataset* contiennent des lignes n'exhibant pas toutes les colonnes. Il y a donc « des cases vides ». Ce problème anodin peut pourtant poser des problèmes et il faut adopter une des stratégies suivantes :

- Supprimer l'ensemble des lignes où au moins une valeur manque : C'est la solution la plus propre mais le jeu de donnée risque de se réduire drastiquement.
- Remplacer la valeur manquante : Soit par une moyenne des autres valeurs, soit par une valeur « label » (e.g. `missing`), soit même par une valeur aléatoire.

La meilleure solution reste toute de même de bien sélectionner ses colonnes, et de supprimer les lignes contenant une valeur manquante. Il faut cependant trouver un équilibre entre l'exactitude des valeurs utilisées et la taille du [dataset](#).

### 2.1.3 Outliers

Un *outlier* est une valeur atypique du jeu de donnée risquant de biaiser l'analyse que nous pourrions en faire.

Cependant, elles peuvent aussi aider à traiter un problème dans un cas plus général. C'est pourquoi leur élimination (ou conservation) est problématique et fait appel aussi bien au bon sens, qu'à une parfaite compréhension des données ainsi qu'à l'expérience.

## Recherche « à la main »

La première méthode de recherche a l'avantage d'être simple. Elle consiste simplement à effectuer une recherche visuelle des *outliers* via un *scatter plot* pour les jeux de données de dimension inférieure à 2 et via un *parallel coordinates* sinon. L'humain étant naturellement doué fort pour détecter les valeurs extrêmes, les *outliers* devraient être assez simple à exclure.

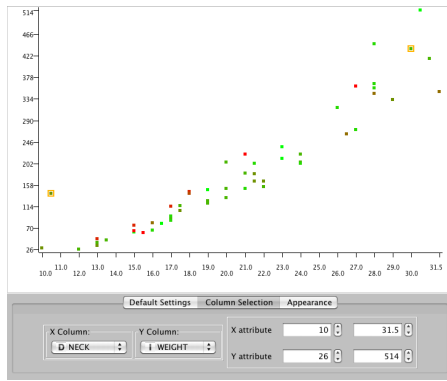


FIGURE 1 – Un exemple de diagramme « Scatter plot »

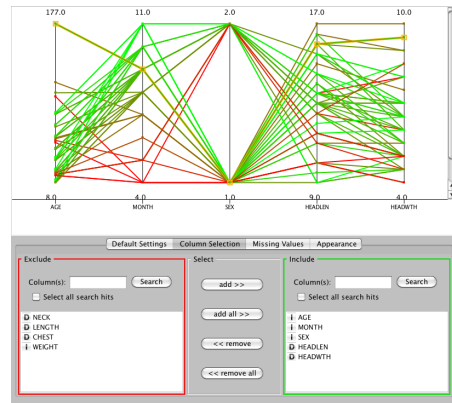


FIGURE 2 – Un exemple de diagramme « ParallelCoordinates »

### Clustering hiérarchique

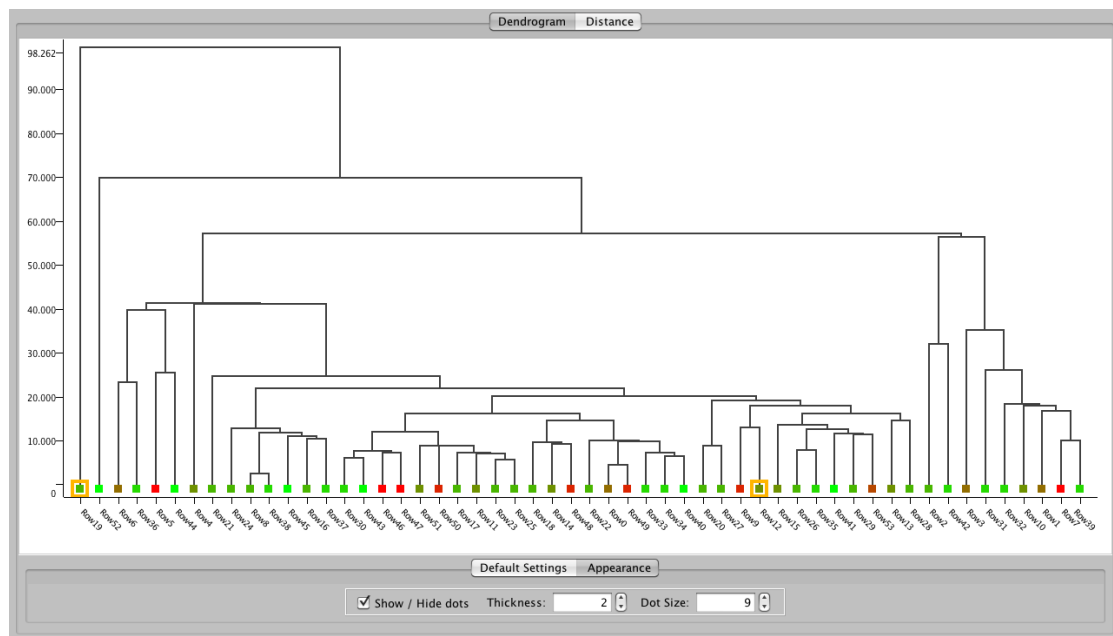
Une méthode efficace et qui me semble un peu plus rigoureuse consiste à utiliser un *clustering* hiérarchique afin d'en extraire les *outliers*.

La méthode est très simple :

*Chaque point ou cluster est progressivement absorbé par le cluster le plus proche.*

En effet, le dernier point ou petit groupe de point qui se fait « coller » en dernier représente un *outlier*.

Sur la figure 4, il est évident que le point le plus à gauche est un *outlier*. En effet, il s'est collé en dernier, et ce, à une distance presque égale au rayon du cluster contenant tous les autres points.

FIGURE 3 – Exemple de dendrogramme extrait d'un *clustering* hiérarchique

### Box plot

C'est encore un moyen très rapide de démasquer les *outliers* car ils font figurer sur le même diagramme : la médian, les quartiles et les décile. Nous avons donc toutes les

données statistiques nécessaire afin d'évaluer l'appartenance d'un point à notre futur jeu de données.

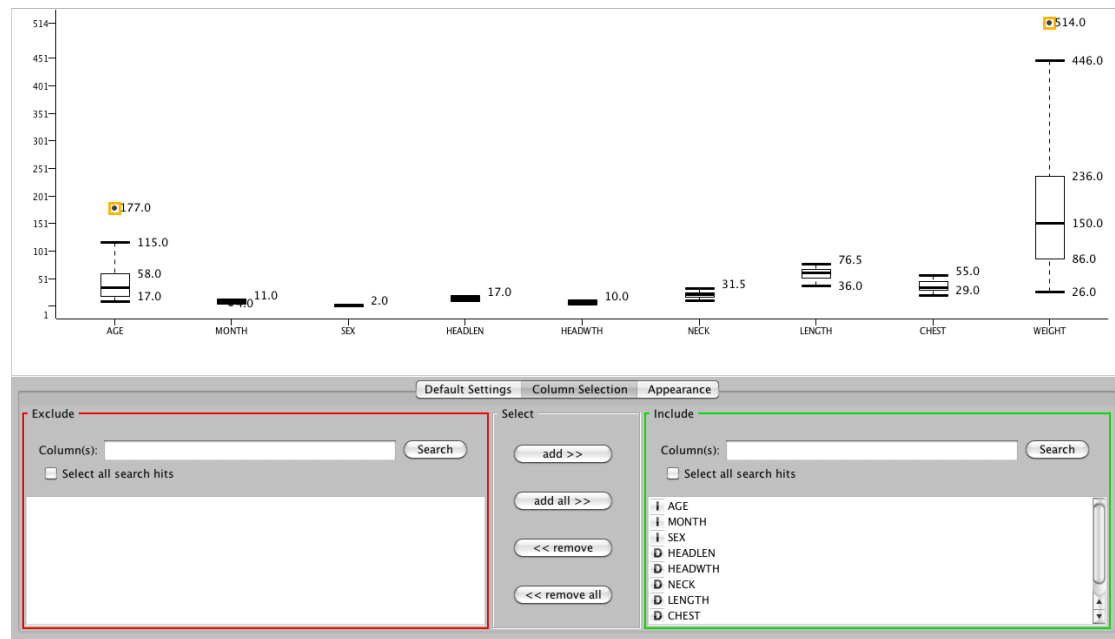


FIGURE 4 – Exemple de boîte à moustache

## Méthodes statistiques

Une des dernières méthodes (que nous n'utilisons pas sur Knime) est une approche statistique des données. Il est prouvé que de nombreux phénomènes suivent une loi normale. De plus, nous connaissons très bien cette loi et des mathématiciens ont pu modéliser l'atypique afin de fournir des méthodes automatiques d'exclusion des *outliers*.

Nous ne rentrerons pas plus dans le détail mais parmi ces méthodes, nous pouvons citer le critère de PEIRCE, celui de CHAUVENET ou encore le test de GRUBB.

## 2.2 Clustering

Une fois la phase de sélection des données, il reste évidemment à effectuer le *clustering*. Cette étape consiste à classer les données en paquet afin de pouvoir en extraire des relations entre eux.

Cette étape se réalise grâce à de nombreux algorithmes. Profitons de cette partie pour étudier les spécificités de 3 grands algorithmes :

- Le *clustering* hiérarchique
- La méthode *K-Mean*
- La méthode *Fuzzy C-Mean*

### 2.2.1 Détermination du nombre de clusters

Le nombre de *clusters* n'est pas laissé libre aux différents algorithmes et il faut prendre une décision cohérente.

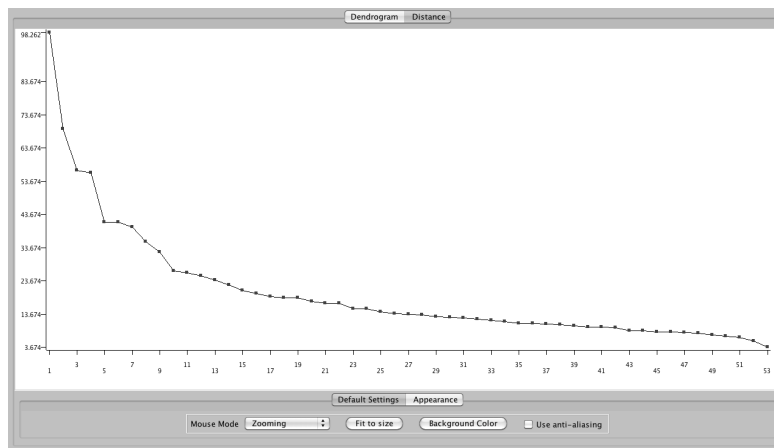
Parfois, ce nombre est fixé : e.g. lorsque le client commande une classification en 3 groupes « maigre », « normal », « obèse ».

Cependant, ce n'est pas toujours (et même rarement) le cas. Il existe cependant une méthode pour déterminer de manière formelle un nombre de cluster adapté au jeu de données.

#### **Clustering hiérarchique**

Encore une fois, l'utilisation du *clustering* hiérarchique est recommandé pour prendre des décisions.

L'idée est que le nombre de cluster correspond au nombre de *gaps* de la courbe de distance :



Dans ce cas, il apparaît que 3 *clusters* serait un bon choix.

Il faut cependant bien garder à l'esprit que le nombre de *gaps* est fortement dépendant du type de fusion utilisé pour le *clustering*. Knime propose 3 types de fusion :

- **SINGLE**<sup>2</sup> : Établit la distance entre deux *clusters* comme étant la plus petite distance entre deux points des *clusters*.
- **AVERAGE** : Établit la distance entre deux *clusters* comme étant la distance entre les centroids des *clusters*.
- **COMPLETE** : Établit la distance entre deux *clusters* comme étant la plus grande distance entre deux points des *clusters*.


Par conséquent, le choix entre ces différentes fonction de distance s'effectue en fonction de la distribution des données. En effet, en **SINGLE** ou **COMPLETE**, les extrêmes

2. Appelé aussi **Nearest Neighbour**, ou encore **Shortest Distance**

auront énormément d'influence sur le *clustering*. En **AVERAGE** par contre, ils auront *a priori* pas d'influence.

## 2.3 Clustering

Une fois les données « nettoyées » et le nombre de *clusters* déterminé, il est enfin temps de procéder au *clustering* (*enfin!*).

Nous pouvons d'ores et déjà distinguer deux types de *clustering* : 

- Le *clustering* hiérarchique :
- Le *clustering* partitionnel :

Nous n'avons pas d'intérêt dans ce document à détailler le fonctionnement interne de ces algorithmes, cependant, il est intéressant de voir dans quel cas nous choisirons l'un ou l'autre.

<b>Nom</b> : K-Mean	<b>Type</b> : Partitionnel
<b>Avantages</b> : <ul style="list-style-type: none"> <li>– Accepte un <i>dataset</i> relativement grand.</li> <li>– Rapide.</li> <li>– Donne une appartenance binaire (<i>i.e.</i> « appartient » ou « n'appartient pas » au cluster).</li> </ul>	
<b>Désavantages</b> : <ul style="list-style-type: none"> <li>– Sensible aux conditions initiales, il nécessite donc une vérification de la stabilité.</li> </ul>	

<b>Nom</b> : Fuzzy C-Mean	<b>Type</b> : Partitionnel
<b>Avantages</b> : <ul style="list-style-type: none"> <li>– Accepte un <i>dataset</i> relativement grand.</li> <li>– Rapide.</li> <li>– Donne une appartenance réelle (<i>i.e.</i> le pourcentage d'appartenance à chaque cluster) au cluster.</li> </ul>	
<b>Désavantages</b> : <ul style="list-style-type: none"> <li>– Sensible aux conditions initiales, il nécessite donc une vérification de la stabilité.</li> </ul>	

### 2.3.1 Normalisation

La question de la normalisation est fondamentale dans le sens qu'elle influence le résultat donné par les algorithmes de *clustering* du même type que *K-Mean*. En effet, *K-Mean* ne normalise pas automatiquement les données et la magnitude de celles-ci peuvent alors influencer ce *clustering*.

Nous disposons de plusieurs types de normalisation :

**Min-Max** C'est la technique de normalisation linéaire classique. Elle soustrait la valeur minimum de l'attribut à toutes les valeurs et les divise ensuite par la plus grande valeur de l'attribut. Nous obtenons ainsi un attribut borné dans  $[0; 1]$ . Ce type de normalisation peut être utilisé partout, est bijective (*sous entendu : facilement bijective*) et ne modifie que linéairement les données.

**Z-Score** La technique Z-Score est déjà beaucoup moins anodine et à utiliser avec précaution. En effet, elle mesure l'écart entre l'écart type et la moyenne. Cette



transformation se base sur une distribution gaussienne des données (*Cf.* Loi normale) et n'est ni bijective ni linéaire, ce qui en fait une normalisation à utiliser que si l'on sait ce que l'on fait.

**Decimal Scaling** C'est une division par la plus petite puissance de dix faisant passer la plus grande valeur de l'attribut en dessous de 1. C'est linéaire, simple et relativement efficace. De plus, cette opération conserve les ordres de grandeurs ce que les deux méthodes précédentes ne font pas.

Elle est cependant inutile dans le cas de valeurs relatives (*e.g.* du type pourcentages).

## 2.4 Conclusion

Bien que cette partie résumant notre démarche expérimentale pourrait ressembler à un résumé du cours et des mes lectures sur le sujet, il nous a permis d'une part d'aborder des questions épineuses comme la normalisation, le manque de valeurs ou encore la malédiction du clustering, et ceux, en détail. De plus, il nous a semblé cohérent que, à la vue d'un sujet aussi libre, nous nous devions de justifier nos résultats par l'application d'une méthode rigoureuse.

### 3 Secteurs d'activité des différents pays

Pour cette première étude de cas, nous avons choisi d'utiliser les données afin de caractériser l'activité d'un pays en fonction de :

- La taille du pays
- La population
- Le **Produit Intérieur Brute (PIB)**
- Le **RNB**
- Les exports industriels

Notre démarche dans cette analyse est simple : Après avoir établi un *clustering* sur le pourcentage du **PIB** dû à l'agriculture, l'industrie et les services, nous avons créé un arbre de décision prenant pour critère de sélection les données citées ci-dessus.

#### 3.1 Sélection des données

Après avoir sélectionné les colonnes qui nous intéressent ( *i.e.* **Agriculture, value added (% of GDP), Industry, value added (% of GDP)** et **Service, value added (% of GDP)**), nous obtenons la matrice suivante :

Le diagramme de est :

Enfin le diagramme *parallel coordinates* :

#### 3.2 Clustering

##### 3.2.1 Clustering hiérarchique

Le choix du type de *clustering* hiérarchique est, *a priori* délicat. Voici les résultats obtenus avec un *clustering* hiérarchique :

**SINGLE**

**COMPLETE**

##### 3.2.2 Clustering K-Means

**Vérification de la stabilité du *clustering***

#### 3.3 Interprétation

##### 3.3.1 Arbre de décision

Ajouter le  
nom de la  
colonne cor-  
respondante

Ajouter  
l'environ-  
nement  
subimages  
pour une  
meilleur  
intégration

boite à  
chaussure ??

Guinée :  
95% de son  
PIB est  
industriel

Inclure le  
tableau de  
vérification  
de l'entropie

## 4 Création d'un module de vérification du K-Mean

En plus des études de fouille de donnée réalisé, il nous a semblé important d'avoir une compréhension plus approfondi des enjeux de la vérification des résultats ainsi que de l'utilisation du logiciel utilisé pour le TP : *Knime*.

C'est pourquoi nous avons réalisé une boîte<sup>3</sup> permettant de vérifier très simplement le convergence de l'algorithme *K-Mean*.

### 4.1 Principe de l'algorithme

Le principe est relativement simple. En effet, mon algorithme se « contente » de compter les différentes configurations de clusters observées au travers des itérations (et donc des configurations initiales).

### 4.2 Implémentation

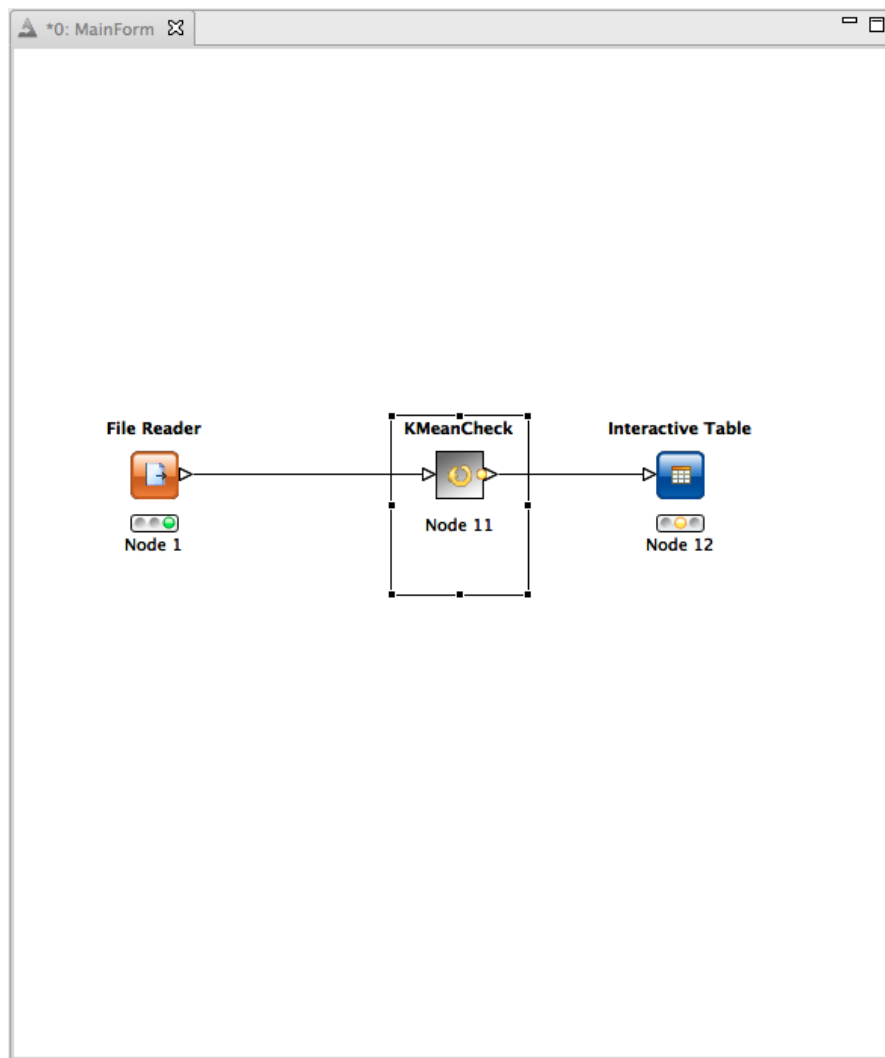


FIGURE 5 – Vue de l'utilisateur final

---

3. Le terme *Knime* est *meta node*

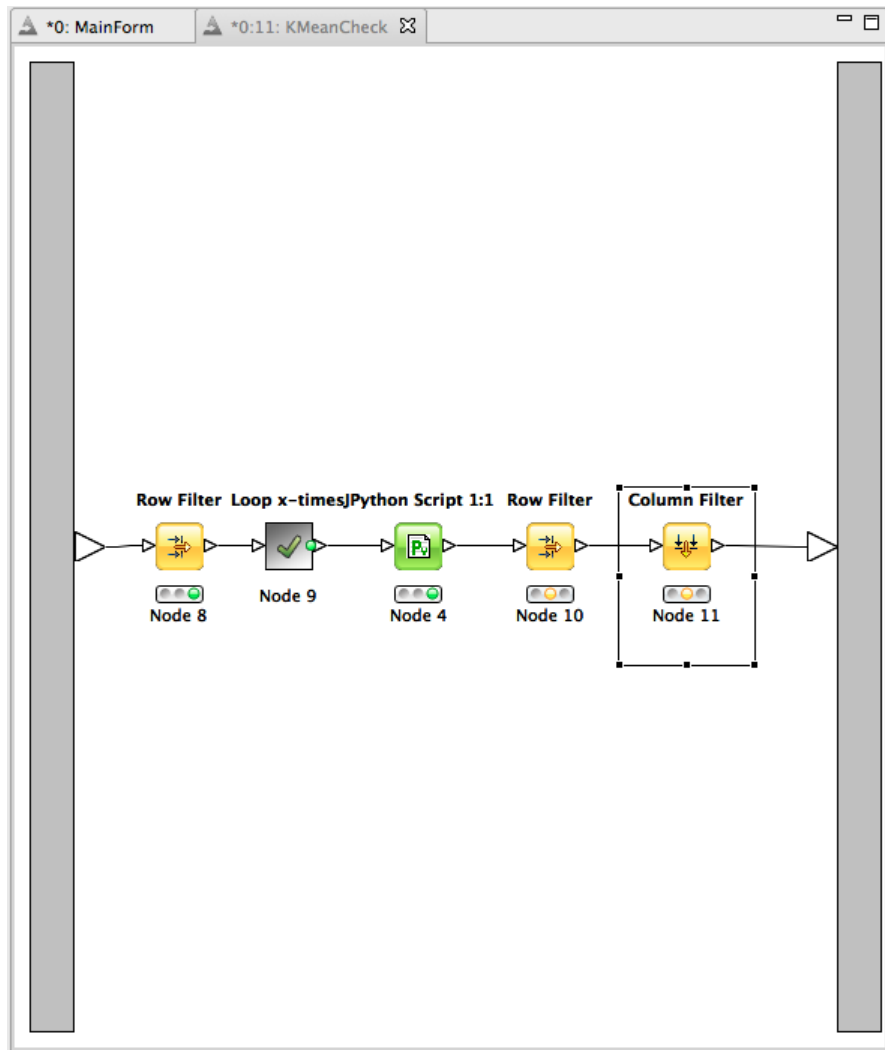


FIGURE 6 – Vue intérieure de la boîte

Row ID	S Nombre de configurations
Row0#0	5

FIGURE 7 – Résultat

### Code python

```

1  # This program is free software: you can redistribute it and/or modify
2  # it under the terms of the GNU General Public License as published by
3  # the Free Software Foundation, either version 3 of the License, or
4  # (at your option) any later version.
5  #
6  # This program is distributed in the hope that it will be useful,
7  # but WITHOUT ANY WARRANTY; without even the implied warranty of
8  # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
9  # GNU General Public License for more details.
10 #
11 # You should have received a copy of the GNU General Public License
12 # along with this program. If not, see <http://www.gnu.org/licenses/>.
13 #
14 # Author : Maxime Gaudin

```

```

15 # Year : 2011

17 iterator = inData0.iterator()
   idts = inData0.getDataTableSpec()

19
   clustersColIndex = idts.findColumnIndex("Cluster")
21 nbClusterIndex = idts.findColumnIndex("nbCluster")

23 MaxDimensionIndex = nbClusterIndex - 1

25 # Lecture des données
   rows = []
27 while iterator.hasNext():
   row = iterator.next()
29   rows.append(row)

31 iterationColIndex = idts.findColumnIndex("Iteration")
   iterationList = [] # ( [cluster1, cluster2, ... ] )
33
   for currentRow in rows:
35     clustersCoordinates = []
       nbClusters = str(currentRow.getCell(nbClusterIndex))
37
       # Récupération des données
39       for i in range(0, int(nbClusters)):
           currentCluster = []
41
           for j in range(1, MaxDimensionIndex):
43             currentCluster.append(str(currentRow.getCell(j)))

45     clustersCoordinates.append(currentCluster)
       clustersCoordinates.sort()
47     iterationList.append(clustersCoordinates)

49 # Analyse
   # List -> Set
51 uniques = []
   for x in iterationList:
53     if x not in uniques: uniques.append(x)

55 print(uniques)

57 # Sortie du résultat
   for currentRow in rows:
59     resultCell = StringCell(str(len(uniques)))
       newRow = AppendedColumnRow(currentRow, [resultCell])
61     outContainer.addRowToTable(newRow)

```

### 4.3 Axes d'amélioration

Pour des raisons de temps, nous avons choisi de ne pas implémenter l'ensemble des fonctions que nous aurions voulues. Par exemple, il serait très utile de connaître la fréquence d'apparition des configurations de clusters afin de déterminer sur une configuration est « minoritaire » par rapport à une autre.

*Peut-être que les 4IF de l'année prochaine seront tentés de reprendre le code pour l'améliorer !*

## Glossaire

**DATASET**

Jeu de donnée [Page(s) [4](#)]

## Liste des accronymes

**PCA**

Primary Component Analysis [Page(s) [4](#)]

**PIB**

Produit Intérieur Brute [Page(s) [10](#)]

**RNB**

Revenue National Brute [Page(s) [1](#), [10](#)]

**VIH**

Virus d'immunodéficience Humaine [Page(s) [1](#)]

TP réalisé sur



**Fin**

