INSA Lyon
20, avenue Albert Einstein
69621 Villeurbanne Cedex

Datamining

« Project Flickr »

du 24 Février 2014 au 10 Mars 2014

Hexanôme ***H4211*** *:*
Rémi DOMINGUES
Adrien DUFFY-COISSARD

Enseignants :
Mehdi KAYTOUE
Jean-François BOULICAUT

Année scolaire 2013-2014

Sommaire

1 Introduction

Cette étude doit permettre à terme au Grand Lyon d'améliorer la gestion de ses transports en commun et la vie de ses touristes.

L'objectif de celle-ci réside donc premièrement dans l'identification des points d'intérêts situés dans la ville de Lyon. Pour ce faire, l'utilisation d'algorithmes de clustering dans un contexte de fouille de données doit permettre l'identification des zones visées.

À cet effet, ledit clustering s'effectuera sur la base d'un fichier CSV obtenu grâce à la collecte de photos géolocalisées via l'API Flickr.

2 Analyse et nettoyage des données

Les données reçues au format CSV correspondent à la structure suivante : <id,user, longitude, latitude, hashtags, legend, minutes_taken, hour_taken, day_taken, month_taken, year_taken, hour_uploaded, day_uploaded, month_uploaded, year_uploaded, url>

Afin d'obtenir un jeu de données propre sur lequel effectuer nos analyses, il est en premier lieu nécessaire de filtrer, corriger ou valider la cohérence des 83 155 lignes reçues. Pour ce faire, le logiciel de fouille de données Knime est utilisé.

Au sein du flux de traitement utilisé se trouve tout d'abord des blocs préliminaires dédiés au filtrage des données incohérentes ou ne concernant pas l'étude présente :

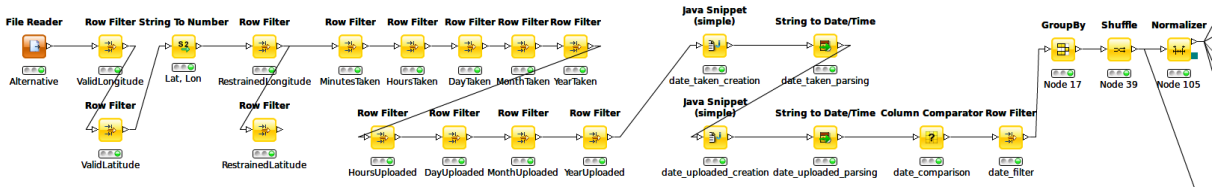

FIGURE 1 – Flux de traitement dédié au filtre des données incohérentes

2.1 Parsing

Le parsing des données s'effectue ici à l'aide du composant **File Reader**, ainsi paramétré :
- Column delimiter : ,
- Read column headers : true
- Ignore spaces and tabs : true
- Advanced
  - Quote support :
    - Quoted strings can extend over multiple lines : true
    - Quotes characters : ët '
  - Ignore spaces : Ignore extra delimiters at end of rows : true
  - Short lines : Allow short lines : true
  - Character decoding : ISO-8859-1

2.2 Latitude et longitude

Un filtre est tout d'abord appliqué sur les colonnes latitude et longitude de type string, à l'aide de deux composants **Row filter**. Seules les lignes respectant l'expression régulière suivante décrivant un nombre flottant sont alors conservées : $[-]?[0-9]+\backslash.[0-9]+$

Les deux colonnes sont ensuite parsées en nombre grâce au composant **String to Number**.

Afin de limiter notre étude à la ville de Lyon, et non à ses alentours, ce dans l'optique de rester dans la portée des transports en commun lyonnais, un filtre des composantes latitude et longitude est ensuite appliqué. On obtient donc grâce à deux composants **Row Filter** les contraintes suivantes :
- $4.802508 \leq longitude \leq 4.908976$
- $45.715569 \leq latitude \leq 45.793688$

22 782 lignes sont supprimées lors de l'application des filtres sur la latitude et la longitude (60 373 lignes restantes).

2.3 Champs date et heure

Afin d'obtenir des données cohérentes vis-à-vis de la date et de l'heure de prise de la photo et de son upload, on applique les contraintes suivantes à l'aide de blocs **Row Filter** :

— $0 \leq minutes\_taken \leq 59$
— $0 \leq hour\_taken \leq 23$
— $1 \leq day\_taken \leq 31$
— $1 \leq month\_taken \leq 12$
— $2000 \leq year\_taken \leq 2014$

— $0 \leq hour\_uploaded \leq 23$
— $1 \leq day\_uploaded \leq 31$
— $1 \leq month\_uploaded \leq 12$
— $2000 \leq year\_uploaded \leq 2014$

Notons ici que seules les prises effectuées depuis l'an 2000 sont acceptées. On observe en effet que la densité de prise des photos par année augmente fortement dans les années 2000. Afin de viser les flux touristiques les plus récents, il serait par ailleurs peu intéressant de porter notre étude sur les prises antérieures, exception faite dans un but d'étude de variation des flux touristiques au fil des ans.

225 lignes sont supprimées lors de l'application des filtres sur les champs date et heure des prises et upload de photos (60 148 lignes restantes).

Dans un souci de cohérence des données finales, on supprime ensuite toutes les lignes dont la date de prise est postérieure à la date d'upload (notons ici que les champs heure et minute ne sont pas comparés, des conflits dus aux heures locales des appareils photos des touristes et/ou du serveur distant pouvant être observés).

Pour cela deux composants **Java Snippet** sont utilisés, dans le but de créer deux colonnes (date_taken et date_uploaded) contenant un champ date sous forme de String :

```
return String.format("%d/%d/%d", $day_taken$, $month_taken$, $year_taken$);
```

```
return String.format("%d/%d/%d", $day_uploaded$, $month_uploaded$,
    $year_uploaded$);
```

Deux blocs **String to Date/Time** permettent alors de transformer au format date les chaînes de caractère initialement de format <dd/MM/yyyy>

Un bloc **Column Comparator** effectue ensuite la comparaison suivante afin d'initialiser une nouvelle colonne date_compare :

```
if(date_uploaded $\heq$ date_taken) {
    date_compare = 'true'
} else {
    date_compare = 'false'
}
```

Un dernier **Row Filter** permet enfin de n'accepter que les lignes dont la valeur du champ date_compare est à "true" .

563 lignes sont supprimées lors du filtre de comparaison des dates de prise et d'upload (59 585 lignes restantes).

2.4 Gestion des doublons

Afin de supprimer les doublons, un composant **GroupBy** est enfin appliqué sur la colonne <id>. En ne retenant que les premières valeurs de chaque groupe pour chaque colonne, ce filtre supprime 31 ligne, amenant le nombre de lignes total à 59 554.

2.5 Mélange

De par leur complexité, les clustering ultérieurs pourront ne s'appliquer que sur un nombre limité de lignes. Afin d'éviter des résultats faussés par des données non représentatives, le composant **Shuffle** nous permet ici de mélanger les lignes des données.

2.6 Normalisation

Les clustering à venir s'effectuant sur la latitude et la longitude, on normalise ces deux données entre 0 et 1.

3 Clustering et évaluation

Les données étant désormais filtrées, la seconde partie de notre travail réside dans le clustering des données à l'aide d'algorithmes implémentés dans Knime ou dans la librairie SciKit-Learn en Python.

Pour ce faire, différents flux de traitement ont été mis en place, prenant en entrée les données sortant du **GroupBy** précédemment décrit.

3.1 K-Means

3.1.1 K-Means appliqué à l'ensemble des données

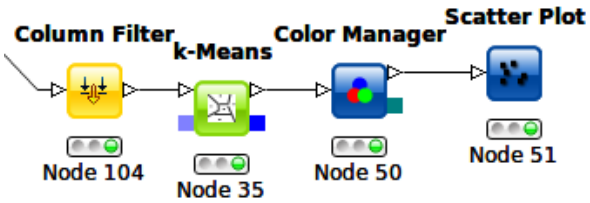

FIGURE 2 – Flux de traitement dédié au clustering par K-Means

Le composant **Column Filter** ne conserve que les colonnes latitude et longitude, sur lesquelles sera appliqué le clustering ensuite affiché en fonction de ces deux composantes.

Le composant **k-Means** est utilisé pour réaliser 60 clusters en un maximum de 99 itérations à partir des colonnes latitude et longitude des données en entrée.

Le nombre de cluster ici utilisé est probablement insuffisant en comparaison du nombre de points d'intérêts potentiel sur Lyon, mais se trouve limité par le bloc **Color Manager**, lequel est utilisé afin d'assigner une couleur à chaque chaîne de caractère unique de la colonne <cluster> crée par k-means, et ne peut générer automatiquement plus de 60 couleurs pour nos valeurs.

Le **Scatter Plot** affiche ensuite les résultats suivants :

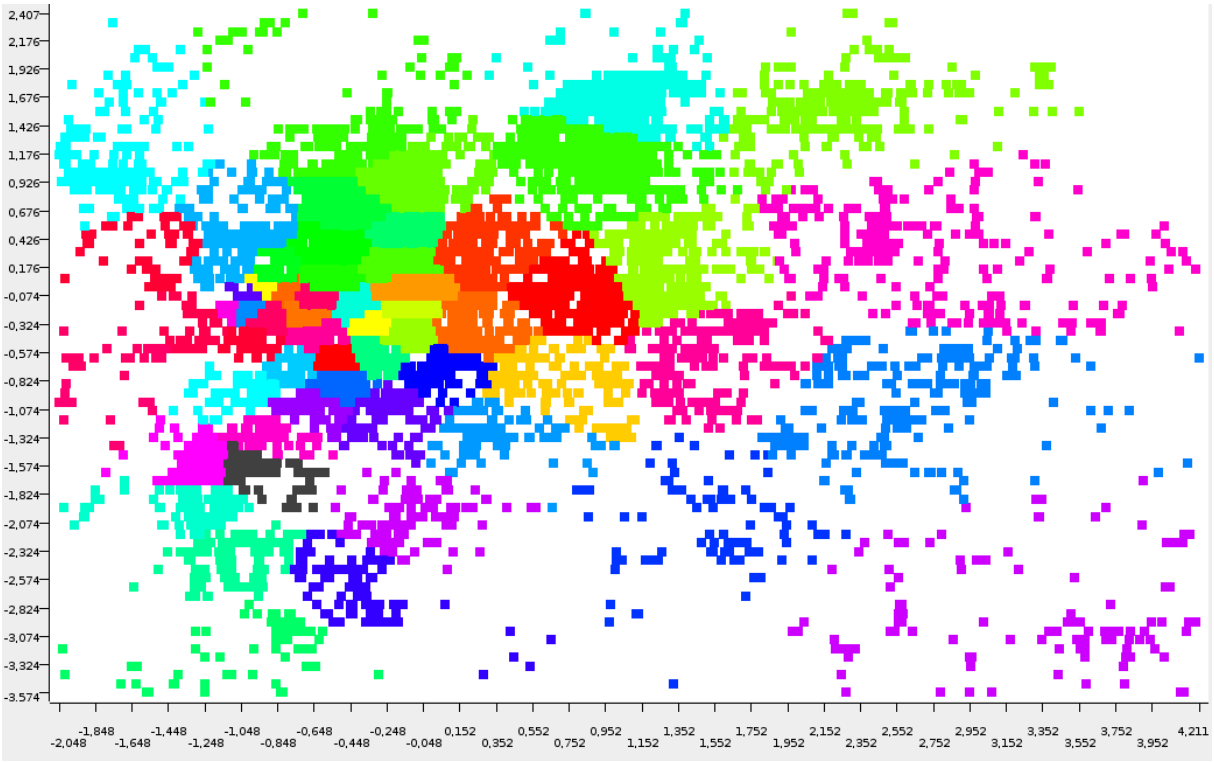

FIGURE 3 – Résultats de K-Means, 60 clusters, 59 554 lignes

Comme attendu, ces résultats sont peu probants, notamment de par la présence de cluster de très faible densité.

Notons que ce clustering, appliqué sur des données ne restreignant pas la latitude et longitude à la ville de Lyon, donne les résultats suivants :

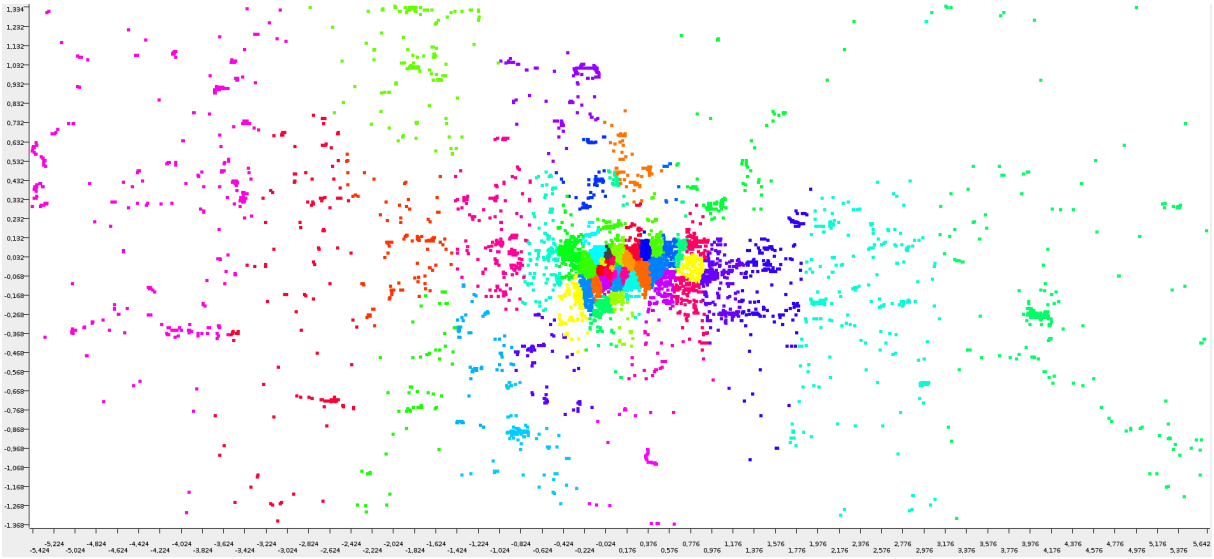

FIGURE 4 – Résultats de K-Means, 60 clusters, 82 199 lignes

Le nombre de cluster est ici hautement insuffisant au vu du clustering appliqué au centre ville. Les clusters éloignés ne semblent pas non plus pertinents.

3.1.2 K-Means appliqué aux données de jour

Une approche de clustering orientée par plages horaire trouve son intérêt vis-à-vis de la gestion des transports en commun. Celle-ci pourrait indiquer les zones à desservir dans un cadre touristique.

En premier lieu, on s'intéressera aux zones d'intérêt de jour, à savoir entre 6h et 21h.

Pour ce faire, le flux de traitement suivant est utilisé :

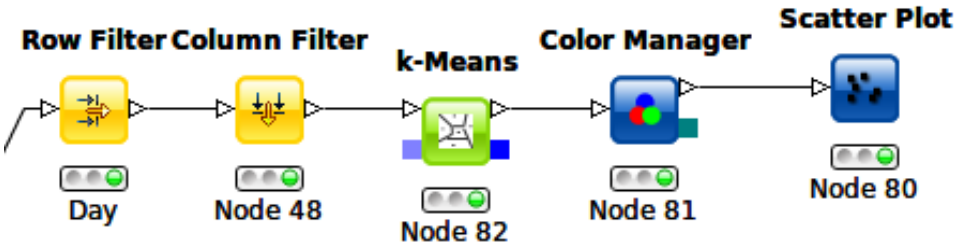

FIGURE 5 – Flux de traitement : K-Means appliqué sur une plage horaire de 6h à 21h

Un **Row Filter** a été ajouté au flux de traitement précédent, lequel n'accepte que les lignes dont l'heure de prise se situe entre 6h et 21h.

Les résultats obtenus sont les suivants :

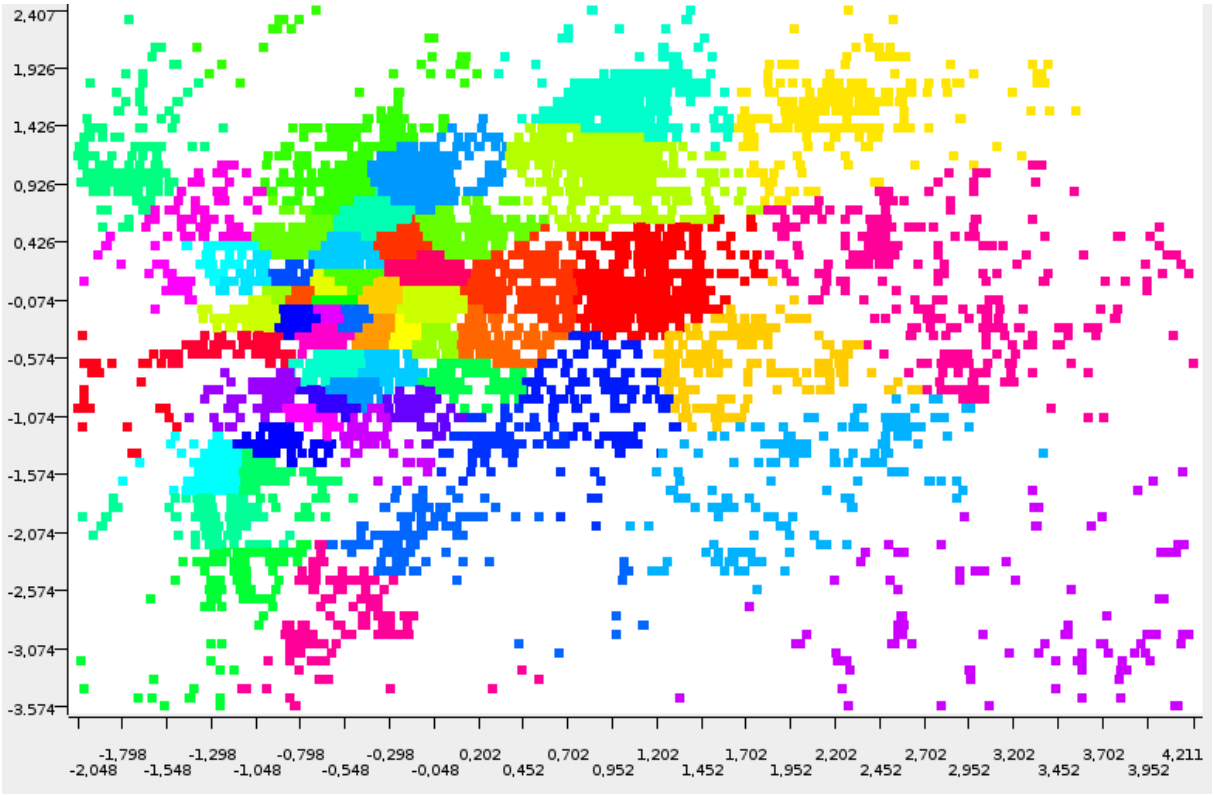

FIGURE 6 – Résultats de K-Means, 60 clusters, 47 820 lignes

De par le nombre de photos traité extrêmement proche des résultats précédents, peu de différences sont observables. Un tel clustering est peu intéressant.

3.1.3 K-Means appliqué aux données de nuit

Les zones d'intérêt de nuit (photos prises de 21h à 6h) peuvent en revanche être davantage exploitées. Pour un flux de traitement identique, le nombre de lignes est bien inférieur (11 734 lignes contre 59 554 initialement).

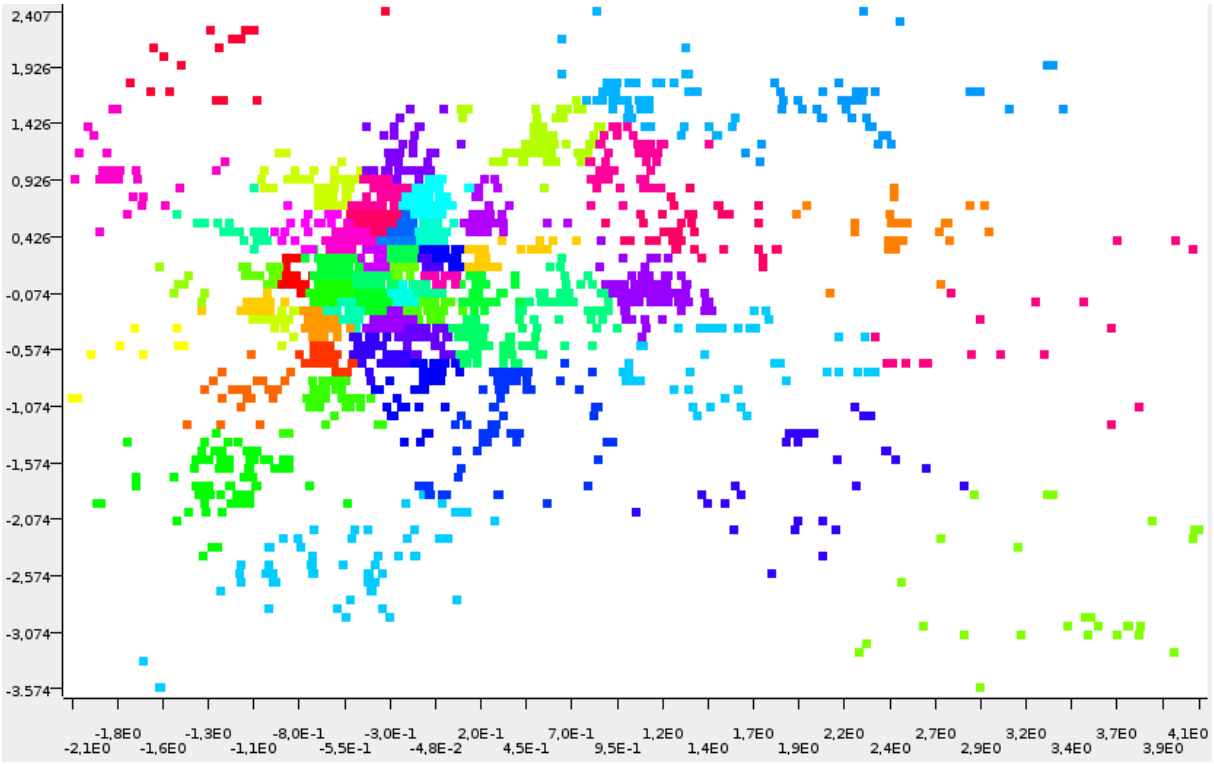

FIGURE 7 – Résultats de K-Means, 60 clusters, 11 734 lignes

Des différences de cluster peuvent être observées. En effet, les touristes semblent se concentrer sur le Vieux Lyon en ces heures noctures, lequel contient de nombreux bars et restaurants.

Néanmoins, des clusters à faible densité subsistent, et le nombre de cluster présent est toujours probablement insuffisant en comparaison du nombre de points d'intérêt sur Lyon.

3.2 Étude de K-Means

K-Means est un algorithme convergeant rapidement et efficace sur des données géolocalisées, notamment dans le cas présent où la répartition des photos est de forme globulaire, celles-ci étant regroupée autour de points d'intérêts.

Néanmoins, celui-ci contient néanmoins un certain nombre d'inconvénients.

— Tout d'abord, le nombre de cluster devant être fixé, cette contrainte s'oppose ici à l'évaluation du nombre de points d'intérêts sur Lyon, ce nombre faisant justement parti des données recherchées.
— Par ailleurs, K-means inclut le bruit dans son clustering, d'où la présence de cluster de faible densité, ne comportant parfois aucun point d'intérêt manifeste.
— Enfin, les limites de cet algorithme sont ici illustrées dans le cas de clusters de taille et de densité différente. Dans le cas du centre de Lyon par exemple, il s'agît d'un cluster de grande taille, et de densité variable. K-means est peu efficace et regroupe ici de nombreux points d'intérêt en un seul.
—

Une approche par densité semble donc d'intérêt, et sera effectuée ultérieurement.

3.3 Clustering hiérarchique

Une approche par clustering hiérarchique est également envisageable. Néanmoins, de par le temps requis par ce clustering, un échantillonnage à hauteur de 1 000 photos est ici utilisé.

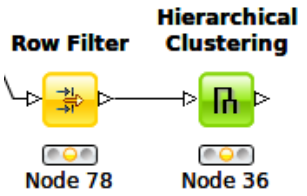

FIGURE 8 – Flux de traitement : Clustering hiérarchique sur 1 000 photos

Le clustering hiérarchique s'effectue sur la latitude et longitude, pour 90 clusters en sortie, la distance Euclidienne étant utilisée.

Les résultats obtenus sont les suivants :

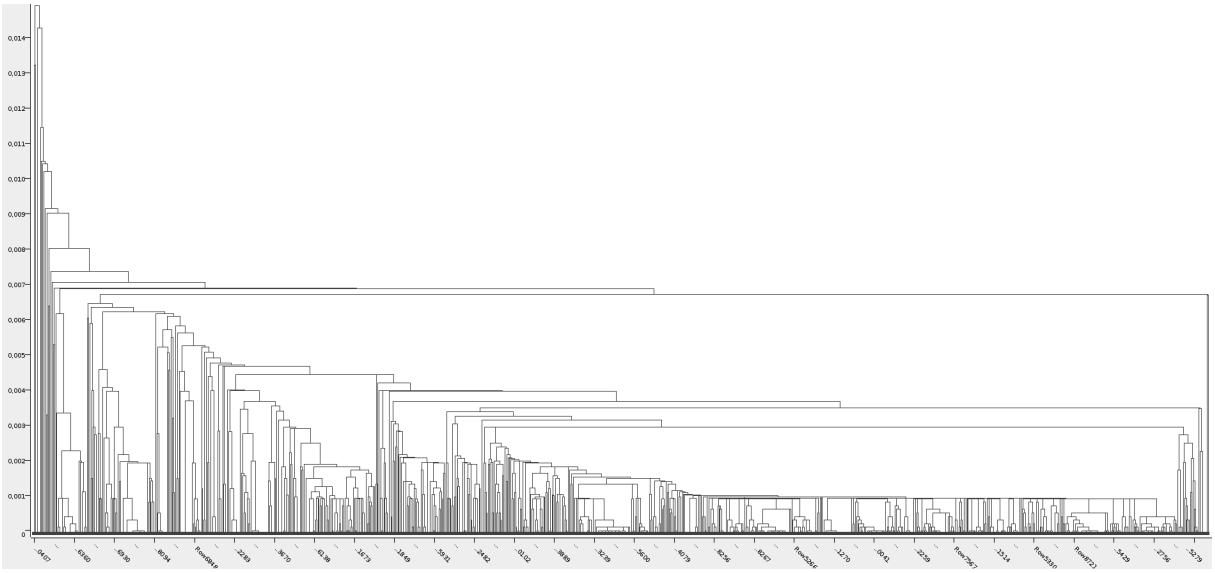

FIGURE 9 – Dendrogramme pour 1 000 lignes

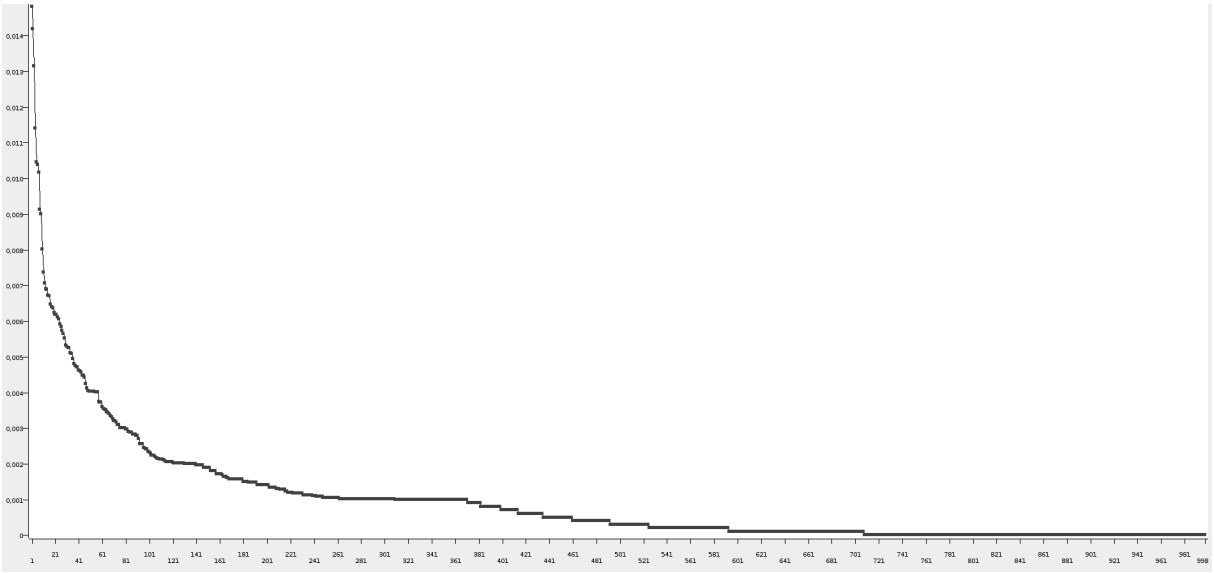

FIGURE 10 – Distance entre les clusters pour 1 000 lignes

hierarchical : permet de choisir nbclusters, mais echantillonnage choisir centroides puis donnner à K-means

Si l'on observe des regroupements certains, avec des clusters d'une intéressante densité, le choix du seuil de coupe de l'arbre du dendrogramme n'est pas aisé, la hauteur des clustering de densité variant fortement. Le choix du nombre de cluster est donc ici encore nécessaire, inconvénient dans le cas où l'un des buts du clustering est de déterminer le nombre et l'identité des points d'intérêt. Malgré un *shuffle*, l'échantillonnage n'est par ailleurs pas forcément représentatif.

Toutefois, une approche intéressante pourrait être de déterminer un ensemble de centroïdes basé sur une étude dudit dendrogramme, puis d'utiliser ceux-ci comme centroïdes initiaux pour l'algorithmes K-means. Les résultats dégagés par K-means pourraient alors en être améliorés.

3.4 DBSCAN

Une approche par densité est à présent effectuée, utilisant l'algorithme DBSCAN. Rappelons que DBScan utilise deux paramètres afin de calculer des clusters, que sont une distance epsilon, et un nombre minimum de point MinPoints. Le fonctionnement de cet algorithme repose sur ce principe : on recherche les points dont les voisins à une distance inférieur à espilon sont au moins au nombre de minPoint. On part d'un point, et on cherche ainsi tous les points qui sont ses voisins, auquel on applique l'algorithme jusqu'à ne plus avoir assez de voisins. Les points parcourus et vérifiants le nombre de voisins minimum feront alors partie du même cluster.

Il est intéressant d'amener une première reflexion quant à l'efficacité de notre DBScan dès lors que nous avons les informations sur son algorithme de calcul. Si on fixe une valeur d'epsilon très faible, nous pourrons discerner des clusters dans un amas de point à forte densité. Si un autre amas possède une densité plus faible, le cluster risque d'être effacé du fait qu'il n'y aura pas minpoints au voisinage d'un point. On pourrait alors augmenter la valeur d'epsilon, mais de nouveau, nous nous retrouvons face à un poblème. A l'inverse, cette fois-ci, les zones à très forte densité ne formeront qu'un cluster car tous les points seront assez proches les uns des autres pour être en dessous de la barre de epsilon. Nous allons donc voir si il est possible de concillier ces deux points afin d'obtenir un clustering correct.

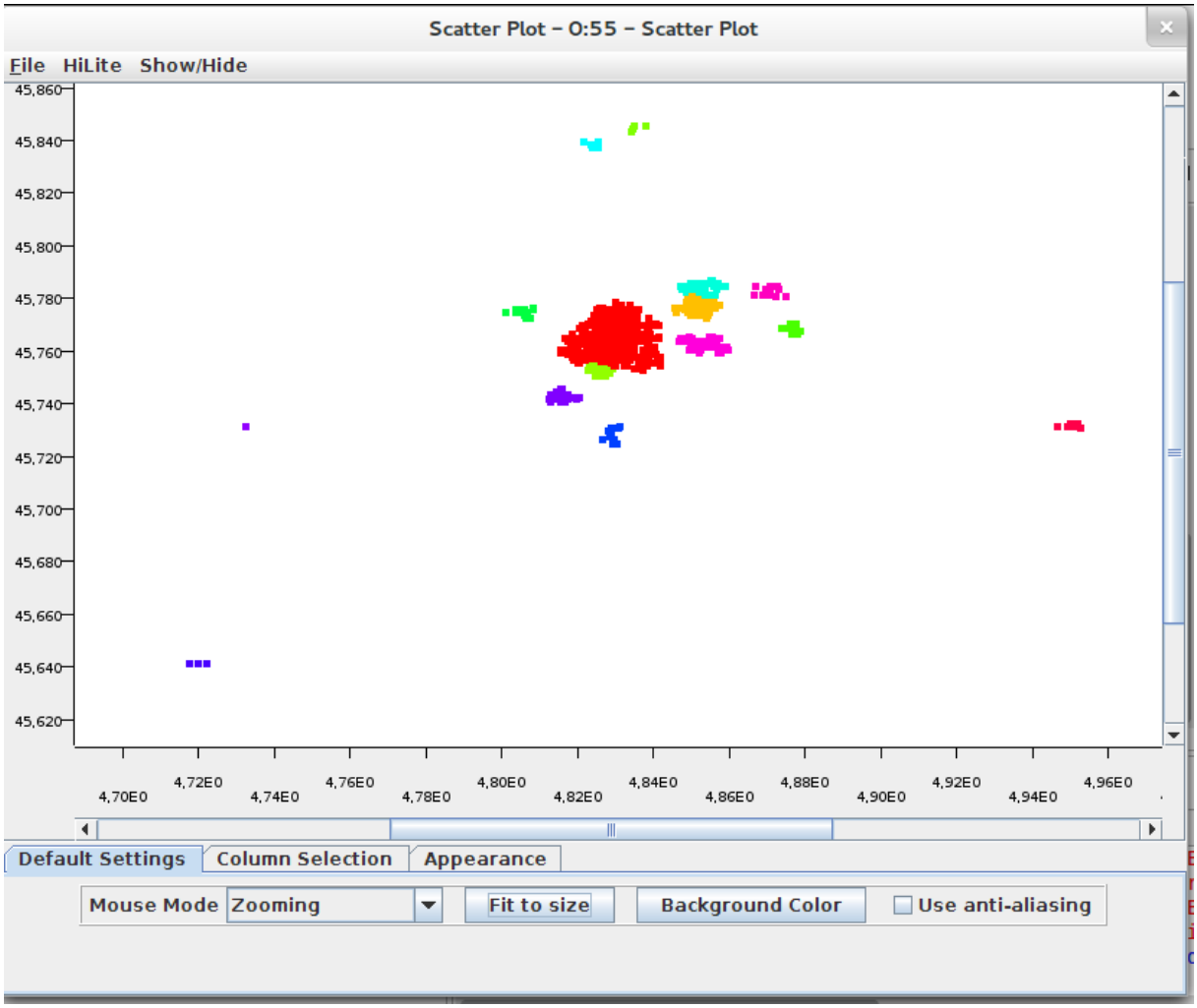

FIGURE 11 – Visualisation des clusters de bordure de Lyon

Nous obtenons tout d'abord un premier résultat satisfaisant, mais seulement pour les clusters externes à ce qu'on pourrait identifier comme le centre de lyon au vu de la densité de photos. En effet, on observe dans le périphérique du grand lyon des clusters bien identifiés, mais un seul cluster est identifié pour l'ensemble des points apparaissants au centre de l'image. Ce n'est pas ce que nous cherchons. Il faut alors affiner la distance, afin de pouvoir les séparer et obtenir des clusters plus fins .

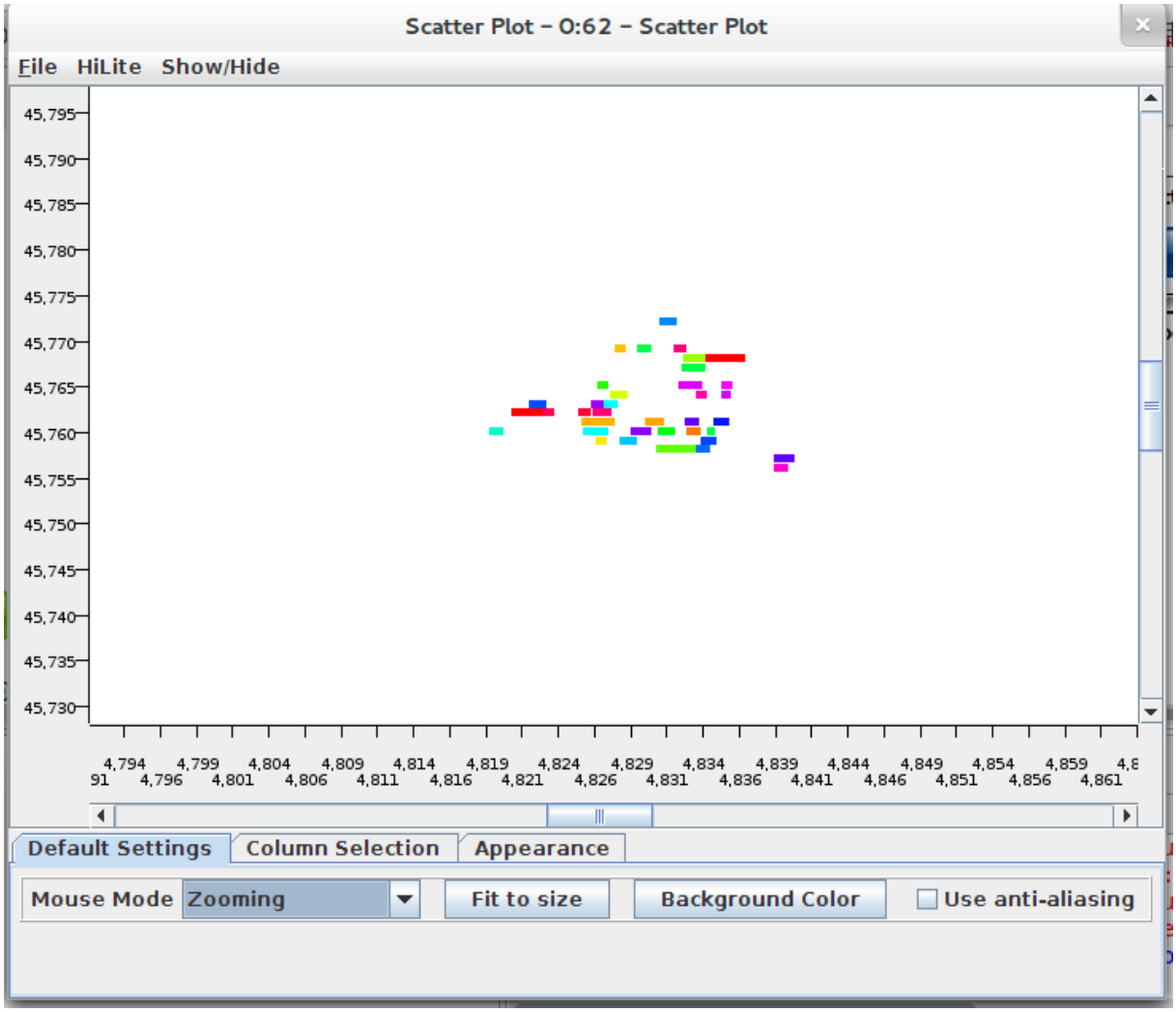

FIGURE 12 – Visualisation des clusters du centre de Lyon

Afin d'éviter de perdre les clusters de l'extérieur, il nous suffit de récupérer le cluster central, et de lui ré-appliquer DBScan, avec cette fois ci des valeurs plus fines. Nous obtenons alors le résultat précédent, sur lequel on observe des clusters intéressants ! Cela donne sous Knime un schéma de cette forme :

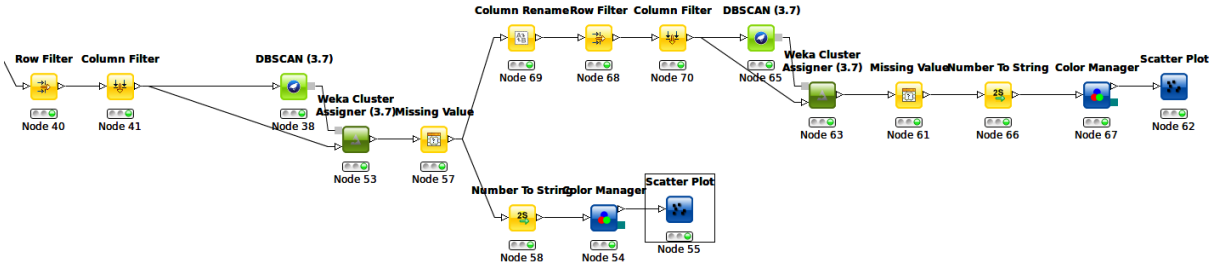

FIGURE 13 – Flux de traitement : Clustering par DBSCAN

Il faudrait dans l'idéal rediriger les deux sorties vers la même table, mais après quelques essaies non-fructueux, nous avons préféré en rester là, et se centrer sur K-Means pour la partie de visualisation. Nous pouvons conclure, comme vu dans le cours, que DBScan n'est pas performant pour identifier des clusters sur des données à densité variable, comme c'est le cas ici entre le centre de lyon et le périphérique.

3.5 Mean Shift

Souhaitant expérimenter l'algorithme Mean Shift, utilisant une approche du clustering par densité, nous exportons à présent l'ensemble des données nettoyées et filtrées au format CSV.

3.5.1 Export des données

Cet export se situe entre le bloc **Shuffle** et **Normalizer** (Voir 1).

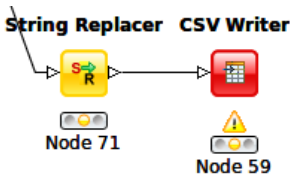

FIGURE 14 – Flux de traitement : Export des données au format CSV sous Knime

Le composant **String Replacer** permet de transformer les caractères $'\backslash n'$ en la suite de caractères $'\backslash\backslash n'$ dans le champ <legend> (utilisation d'une expression régulière).

L'ensemble des données est ensuite exporté par le composant **CSV Writer**.

3.5.2 Mean Shift en Python

La librairie open source d'analyse de données **Pandas** a tout d'abord été utilisée afin de réaliser le parsing du document CSV et la manipulation de la matrice de données obtenue.

L'algorithme **Mean Shift** de la librairie Python **SciKit-Learn** est ensuite appliqué sur la latitude et longitude, les données étant filtrées au préalable par Knime.pour la construction d'une estimation nommée *bandwidth*. Paramètres :

— quantile : 0.005
— n_samples : None

Une instance de la classe MeanShift est ensuite crée à l'aide de l'estimation bandwidth précédente et des paramètres suivants :

— bin_seeding : True
— min_bin_freq : 30
— cluster_all : False

On supprime alors tous les clusters dont le nombre de points est inférieur à 30. La suppression du bruit et le filtre de nombre de points par cluster restreint le nombre de photos qui était initialement de 59 554 à 47 800. Les résultats suivants sont obtenusy :

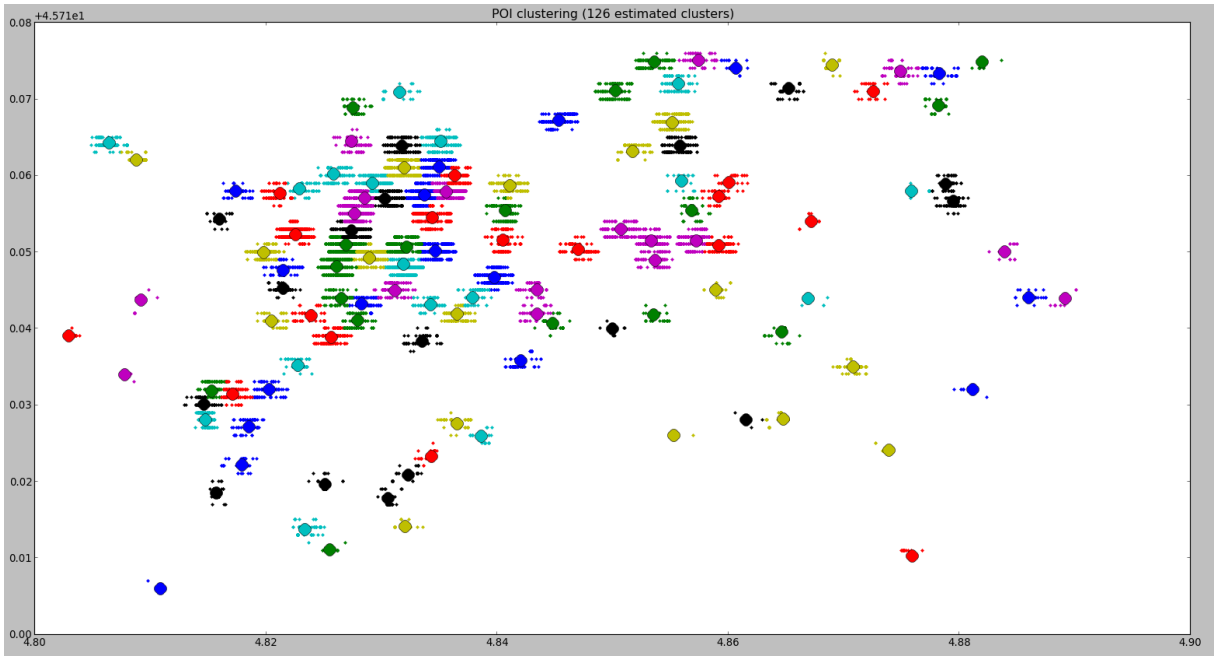

FIGURE 15 – Résultats du clustering par Mean Shift, 126 clusters, 47 800 photos

Comme on peut l'observer, l'approche par densité donne de bien meilleurs résultats. Le bruit n'est pas ici clusterisé et a été éliminé des résultats. Seuls les clusters de densité suffisante et ayant un certain nombre de points sont alors conservés. Toutefois, la validité de ce clustering devrait encore être vérifiée à l'aide d'outils tels Google Places.

Au vu des résultats probants précédents, des clustering plus affinés peuvent également être envisagés. Ci-dessous, les points d'intérêts correspondant aux prises de photos réalisées entre le 6 et 9 Décembre inclus, ces dates correspondant à la Fête des Lumières sur Lyon, et de ce fait à un pic massif d'utilisation des transports en commun.

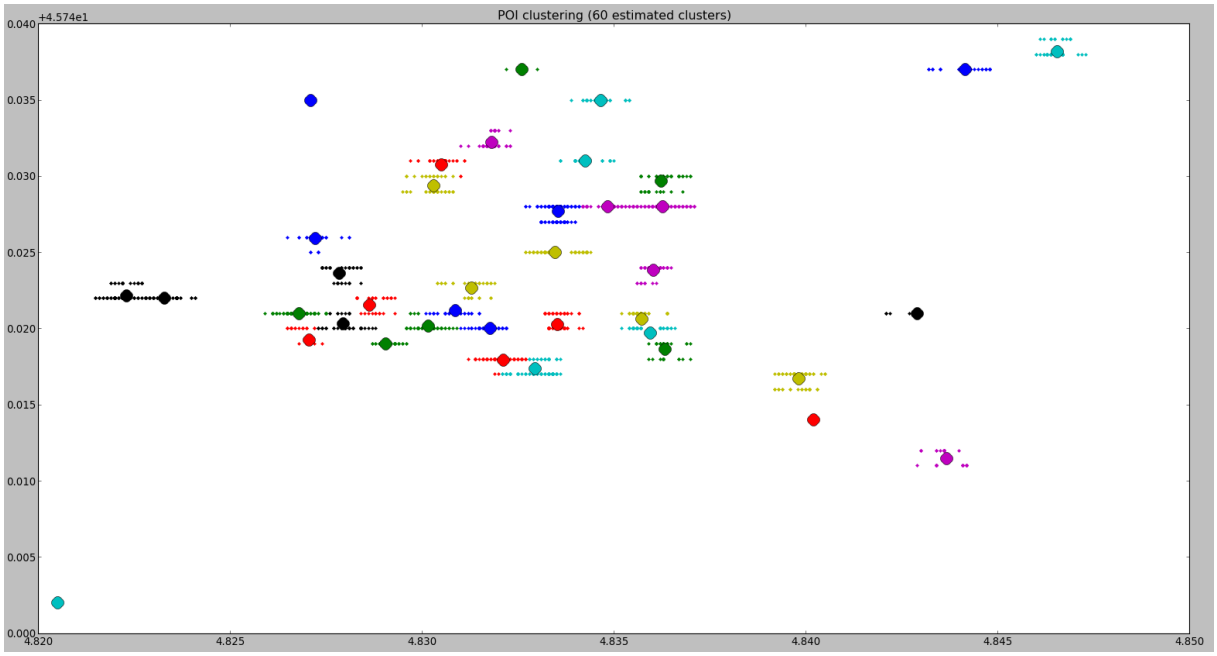

FIGURE 16 – Résultats du clustering par Mean Shift, 60 clusters, 4 100 photos

Un rapprochement entre ces zones et les arrêts TCL les plus proches permettrait d'optimiser le désservissement des zones les plus visitées à cette époque de l'année.

Le résultat de ces clustering est enfin exporté au format CSV afin d'être affiché grâce à un serveur Web.

4 Visualisation des résultats

Afin de visualiser nos clusters, nous utilisons l'API Google Map, qui nous permet de placer des points sur une carte et d'intérargir avec ces derniers afin d'obtenir des informations. Nous procédons comme cela : les clusters sont représentés par des cercles de couleurs, lors d'un clic sur ces cercles, nous affichons les points représentants le lieu de prise d'une photo. Lors du clic sur le marqueur de lieu, la photo s'affiche.

Nous pouvons observer la concentration des clusters au centre de Lyon et sur les bords des fleuves.

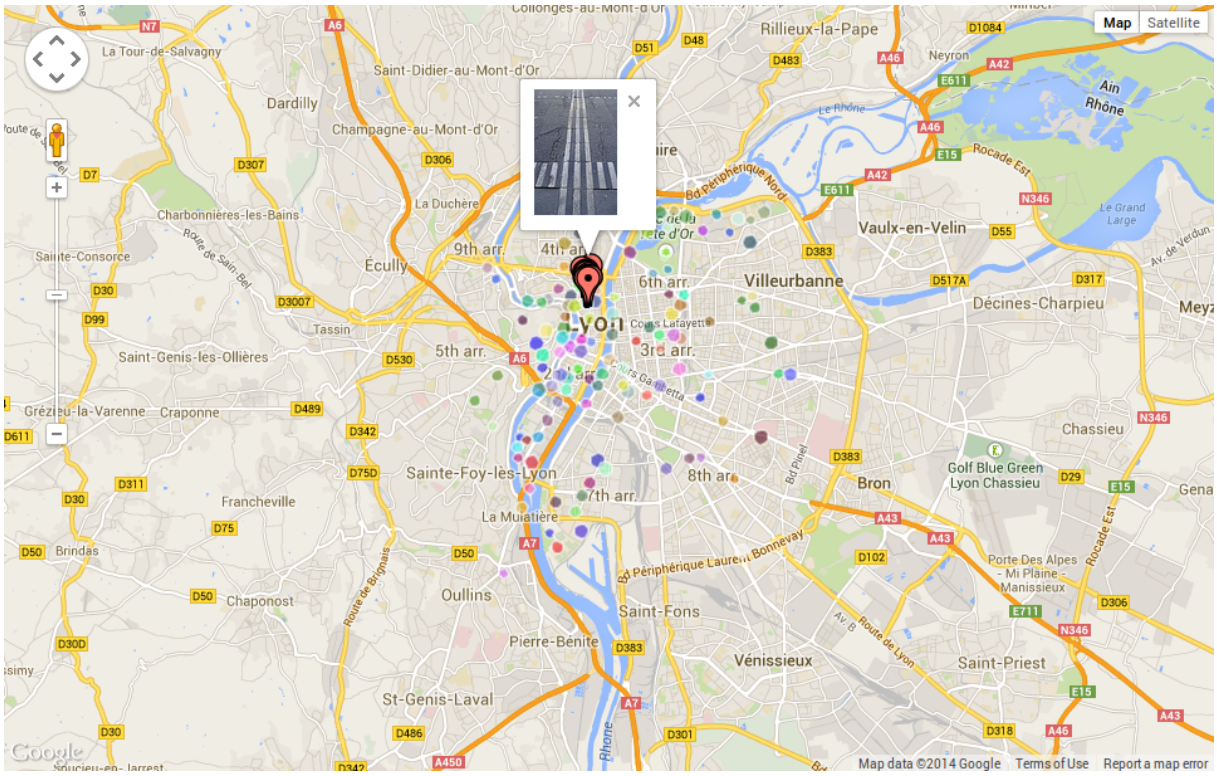

FIGURE 17 – Résultats du clustering par Mean Shift, sur googleMap

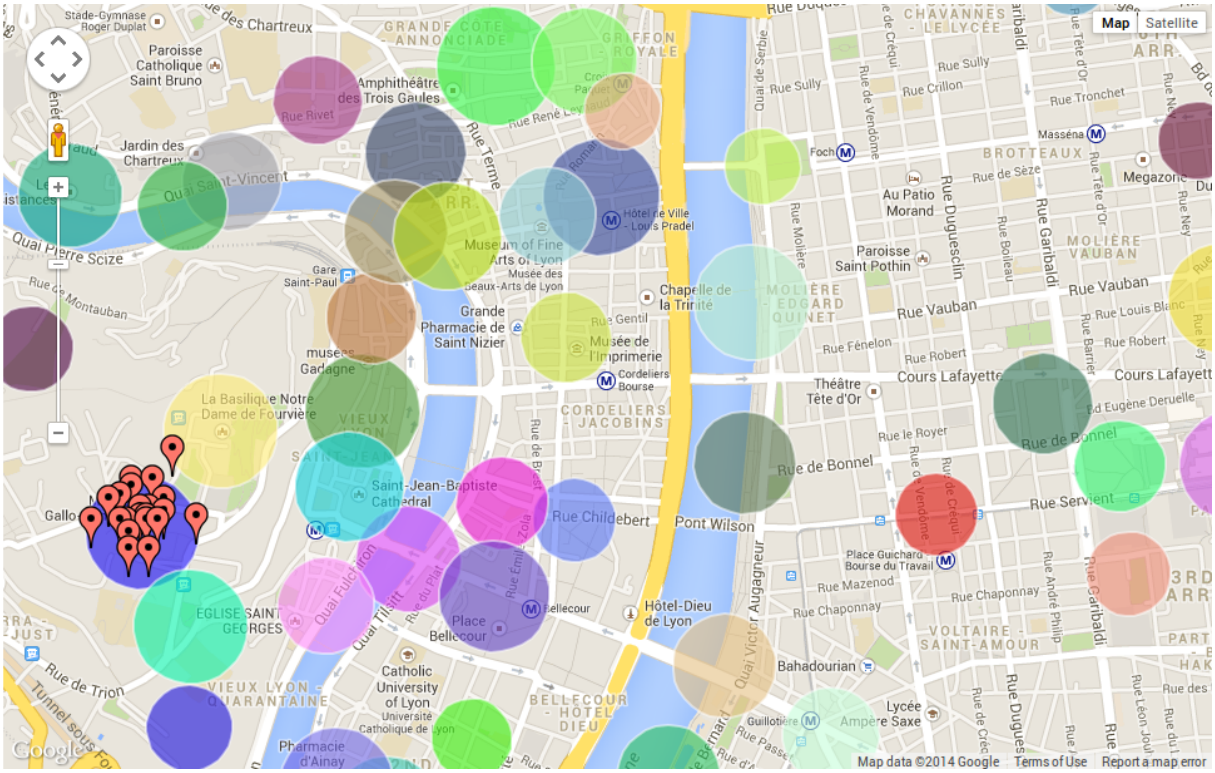

FIGURE 18 – Résultats du clustering par Mean Shift, sur googleMap, plan resserré

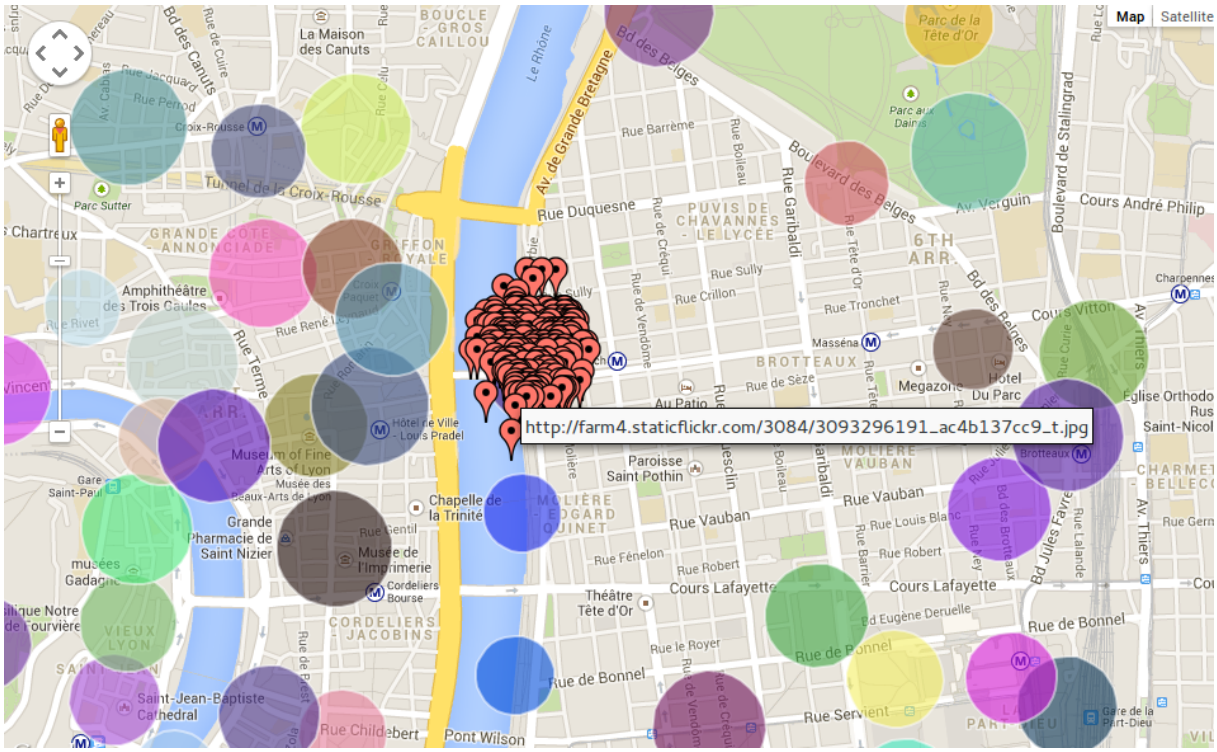

FIGURE 19 – Résultats du clustering par Mean Shift, sur googleMap

Il est appréciable de pouvoir visualiser concrêtement ces données sur une carte. Nous avons essayé de fournir un service permettant à une personne se trouvant dans un lieu de pouvoir avoir une indication sur les lieux qu'elle pourrait ensuite visiter, grâce à une analyse des mouvements des utilisateurs de flickr, mais cela n'a pas abouti par manque de temps. Il aurait été intéressant de reconnaître des parcours ẗypesäu coeur de Lyon.

5 Conclusion

Pour conclure, nous avons pu observer la bonne application de K-means sur notre problème, qui a pu mener avec succès à une visualisation de points d'intérêts au sein de Lyon, représentés par des clusters de points de prise de photos. L'application de DB-Scan s'est avérée directement moins efficace, il a fallut contourner son principal défaut par une séparation des données afin de les traiter sur deux plans différents. L'intégration à Google Map a ensuite été possible après une étude de l'API, et une familiarisation qui n'était pas encore évolué avec le javascript. L'utilisation du langage Python n'a pas posé de problème puisque nous étions tous deux compétents dans le domaine.