RAPPORT TECHNIQUE NF26
Analyse des trajets faits en taxi à Porto sur 1 an

TABLE DES MATIERES

1. Présentation des données

Les données utilisées pour ce projet sont disponibles à l'URL suivante : https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015

Le jeu de données contient des informations sur le trajet des 442 taxis de la ville de Porto, au Portugal. Il a été conçu en 2015 et contient plus de 1.7 millions d'instances avec 9 attributs que nous présentons ci-dessous.

La première étape du travail est d'explorer ces données a notre disposition afin de définir un modèle et de préparer un travail de nettoyage et de remodélisation de données.

Description des attributs

Attribut	Type	Informations
TRIP_ID	Chaine de caractères composée de 19 chiffres	Id unique permettant de caractériser un trajet. Pas de données manquantes
CALL_TYPE	1 caractère : {A ; B ; C}	Informations sur la façon dont le taxi a été demandé par le client. Pas de données manquantes
ORIGIN_CALL	Entier	Valeur si CALL_TYPE = « A », NULL sinon
ORIGIN_STAND	Entier	Valeur si CALL-TYPE = « B », NULL sinon
TAXI_ID	Entier	Id permettant d'identifier un taxi. Pas de données manquantes
TIMESTAMP	Entier	Timestamp UNIX
DAY_TYPE	1 caractère : {A ; B ; C}	Pas de données manquantes.
MISSING_DATA	Booléen	TRUE s'il manque des données géographiques pour l'attribut POLYLINE
POLYLINE	Chaine de caractères	Ensemble de coordonnées géographiques (latitude et longitude) caractérisant la position du taxi toutes les 15 secondes. Données manquantes.

Concernant les données géographiques, nous aurions pu utiliser toutes les données GPS disponibles, avec donc un tableau de coordonnées de taille variable. Néanmoins, dans le cadre de ce projet, nous nous limiterons aux coordonnées de la position de départ et de la position d'arrivée. A l'aide d'expressions régulières, nous tronquerons l'attribut POLYLINE afin de ne garder que les informations nécessaires.

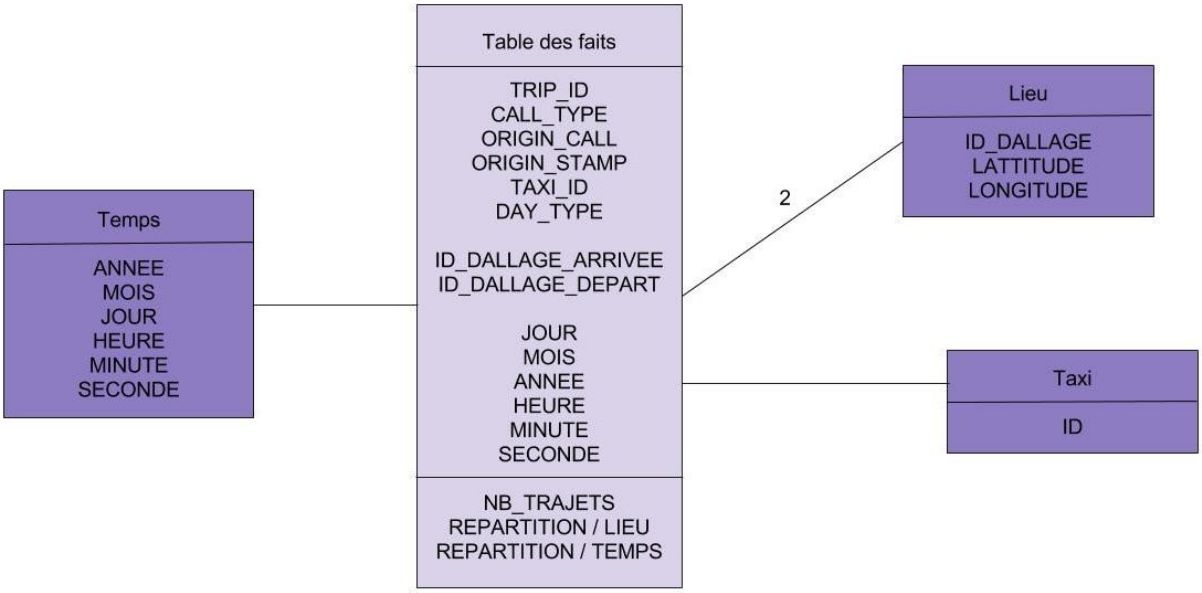
2. Modélisation des données et problématique

Dans notre modélisation, un fait correspondra à un **trajet**.

La modélisation ci-dessous contient trois dimensions :

- la dimension *lieu* qui s'appuye sur un « dallage » de la ville de Porto en fonction de la lattitude et de la longitude. Dans la table de faits, il y aura deux attributs correspondant à cette dimension : un pour le départ et l'autre pour l'arrivée.
- la dimension *temps* qui contient **différents niveaux de granularité**. Le temps pourra s'exprimer par année comme par minute. Ces données seront récupérées grâce à l'attribut TIMESTAMP du jeu de données à disposition.
- la dimensions *taxi* qui contient simplement l'ID du taxi, nous n'avons pas d'informations supplémentaires à disposition.

Ces dimensions existent dans notre modélisation, néanmoins, elles ne seront pas matérialisées dans le sens physique de notre base de données. Elles seront directement contenues dans les différentes tables de faits créées.



Cette modélisation nous permettra de faire différents types de requêtes :

- des requêtes temporelles : quelle est la typologie des trajets pour un jour donné, pour une heure donnée ?
- des requêtes géographiques : où vont les taxis qui partent d'un lieu donné ? d'où viennent les taxis qui arrivent à un lieu donné ?
- des requêtes par rapport à un taxi : statistiques d'un taxi en particulier (trajets moyens par jour, lieu de départ ou d'arrivée le plus fréquent…)

3. Nettoyage, vérification et prétraitement des données

Nous avons vu dans la partie « Description des attributs » qu'il y avait des données manquantes. Cela ne posera pas de problème pour les attributs ORIGIN_CALL et ORIGIN_STAND. Il faudra néanmoins trouver une solution dans le cas de l'attribut POLYLINE.

De plus, nous ferons des vérifications pour les attributs TIMESTAMP et POLYLINE afin de vérifier si des erreurs de saisies ou des données aberrantes ne se sont pas glissées dans le jeu de données.

Enfin, à l'aide de code *Python*, nous allons reformater les données afin d'avoir quelque chose de plus lisible dans notre BDD (par exemple, pour la dimension date).

Attribut TIMESTAMP

Dans cette partie, nous nous appuierons sur le module *Python* « *datetime* » qui permet de manipuler des dates de différents formats. Dans notre cas, on l'utilise pour transformer une date au format timestamp en un objet contenant l'année, le mois, le jour, l'heure, les minutes et les secondes correspondant avec le code suivant :

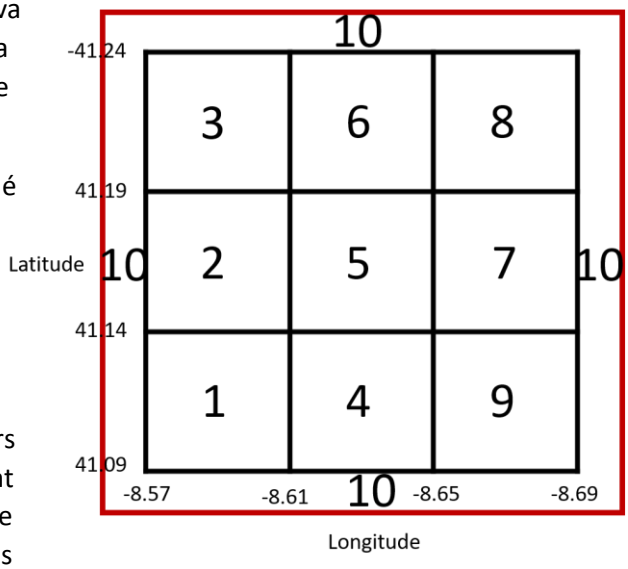
> datetime.datetime.fromtimestamp(int(timestamp)).strftime('%Y-%m-%d-%H-%M-%S')

Pour vérifier qu'il n'y a pas de valeurs aberrantes, j'ai fait un script récupérant la valeur de TIMESTAMP la plus haute et la plus basse.

Le jeu de données contient donc des dates entre le 1er juillet 2013 et le 1er juillet 2014 inclus.

Attribut POLYLINE

Afin de résumer nos données géographiques, on va créer un pavé. En fonction de sa latitude et de sa longitude, chaque couple de coordonnées GPS va être affecté à une des « dalles » de ce pavé.



Par manque de temps, nous avons simplement regardé les coordonnées de Porto et séparé arbitrairement la ville en 9 sous-carré. En temps normal, il aurait fallu faire une analyse plus poussée des données afin de créer un dallage homogène.

En regardant les données, nous avons trouvé des valeurs extra-porto (périphérie proche) et des valeurs aberrantes (les coordonnées correspondent à un point dans la mer par exemple). Pour ne pas perdre d'information, nous avons choisi de conserver ces lignes en leur affectant un id de dallage particulier : 10 pour les valeurs extra-Porto et 11 pour les valeurs aberrantes.

Dans notre table des faits, l'attribut POLYLINE sera donc remplacé par deux id de dallage : celui du départ et de l'arrivée.

4. Architecture Cassandra et mécanisme d'importation

On fait déjà un premier choix général de garder tous les attributs présents dans notre jeu de données. En effet, pour des questions de longévité, il vaut mieux garder trop d'attributs que pas assez. On ne sait pas encore à ce jour quelles seront les requêtes exactes qui nous intéresseront pour le reporting et l'analyse.

Les bases de données noSQL sont basées sur un modèle dénormalisé, avec de la redondance. Nous allons donc construire plusieurs tables de faits avec des *primary key* différentes, chacune adaptée à des requêtes en particulières. La liste présentée ci-dessous n'est pas exhaustive mais elle m'a semblé couvrir les principales requêtes pour répondre à la problématique.

Table 1 : Partitionnement temporel

PRIMARY KEY ((année, mois, jour), heure, minute, seconde, trip_id)

- clé de partitionnement : *(année, mois, jour)*

On a une clé de partitionnement composite (avec 3 attributs) qui va nous permettre de partitionner nos données selon un jour précis. J'ai choisi cette granularité car elle me semblait la plus adaptée. Choisir *(année, mois)* aurait eu pour conséquence de créer des partitions trop importantes par rapport à la taille des nœuds alors que *(année, mois, jour, heure)* n'aurait pas été intéressant pour faire des requêtes pour un jour donné.

Dans toute clé primaire, l'attribut *trip_id* est nécessaire car c'est lui qui rend la clé unique.

La clé de clustering permet d'avoir les données triées temporellement à la seconde près.

Table 2 : Partitionnement géographique selon le pavé de départ

PRIMARY KEY ((id_départ, année), mois, jour, heure, minute, seconde, trip_id))

- clé de partitionnement : *(id_départ, année)*

On partitionne nos données selon l'id du dallage correspond au départ du trajet. Pour des raisons de taille de nœuds, on rajoute l'attribut année à la clé de partitionnement pour que les partitions ne grandissent pas indéfiniment. A la fin de chaque année, la partition est « refermée », plus aucune nouvelle données ne sera rajoutée. Cela permet donc de limiter la taille de la partition car il faut se mettre en situation d'une base de données viable sur plusieurs années.

On a ensuite les données triées temporellement à nouveau avec la clé de clustering.

Table 3 : Partitionnement géographique selon le pavé d'arrivée

PRIMARY KEY ((id_arrivée, année), mois, jour, heure, minute, seconde, trip_id))

- clé de partitionnement : *(id_arrivée, année)*

Cette table est basée sur le même principe que la précédente mais s'intéresse cette fois-ci au pavé d'arrivée d'un trajet.

Table 4 : Partitionnement par rapport aux taxis

PRIMARY KEY (taxi_id, année, mois, jour, heure, minute, seconde, trip_id)

- clé de partitionnement : *taxi_id*

Chaque partition correspondra aux trajets d'un taxi donné. Le jeu de données contenant les informations relatives à 442 taxis, il y aura donc 442 partitions. De la même façon que pour les tables géographiques, on aurait pu ajouter l'année dans la clé de partitionnement mais j'ai trouvé cela moins justifié car, premièrement, les partitions sont plus petites de base et, deuxièmement, si les données sont sur 20 ans, il y aura eu beaucoup de changements parmi les chauffeurs de taxi.

Ces quatre premières tables étaient pour moi les plus « indispensables » et « évidentes » afin de mettre en place des requêtes simples par rapport aux dimensions de notre modèle. Afin d'étendre le champ de notre analyse, j'ai souhaité créer en plus les deux tables suivantes.

Table 5 : Partitionnement par rapport à l'heure

PRIMARY KEY ((heure, année), mois, jour, minute, seconde, trip_id)

- clé de partitionnement : *(heure, année)*

Cette table permettra de faire des requêtes pour mettre en évidence la typologie des trajets par rapport à une heure donnée de la journée. Cela n'aurait pas été une requête intéressante à faire sur notre première table temporelle d'un point de vue base de données. En effet, il aurait fallu parcourir tous les nœuds et récupérer les résultats d'une heure donnée. Cela va à l'encontre de l'avantage fourni par Cassandra et les base de données noSQL.

L'ajout de l'attribut année dans la clé de partitionnement se justifie de la même façon que pour les tables 2 et 3.

Table 6 : Partitionnement par rapport au type de jour (attribut DAY_TYPE)

PRIMARY KEY ((day_type, année, mois), heure, minute, seconde, trip_id)

- clé de partitionnement : *(day_type, année, mois)*

On crée cette table afin de voir les tendances de trajets en fonction du type de jour : un jour spécial (jour férié, vacances…), la veille d'un jour spécial ou un jour normal. Si on partitionne simplement en fonction de l'attribut *day_type*, les nœuds vont être très gros et croitre indéfiniment. On rajoute donc l'année et le mois pour régler ces problèmes de dimensionnement de nœuds.

Ces tables sont créées à l'aide de scripts Python (*create.*py et import.*py*) joints à ce rapport. On utilise la librairie *cassandra.cluster* afin d'envoyer des requêtes à Cassandra depuis du code Python. Pour créer une table on utilise *CREATE* et pour insérer nos données *INSERT*. Ces opérations prennent du temps du au volume des données, il ne fallait mieux pas à avoir à recommencer trop de fois…

5. Reporting

Dans cette partie, je vais présenter différents outils utilisés afin d'analyser et exploiter les données.

a) Statistique descriptive

On peut dans un premier temps calculer des quantités ou des moyennes sur nos données à partir de requêtes simples avec CQL et notamment la commande *COUNT(*)* qui compte les lignes du résultats d'une requête.

Nombre de trajets pat id de dallage d'arrivée en 2013 :

ID_dallage	Nombre de trajets
1	16 833
2	260 732
3	6 571
4	25 572
5	334 705
6	10 474
7	2 122
8	105 046
9	37 268
10	56 268
11	4

Ce résultat montre les limites de mon choix de dallage. En effet, les pavés ne sont pas homogènes en termes de taille. A partir de cette analyse, j'aurais pu adapter mon dallage en redivisant les pavés 2,5 et 8. Le pavé 11 correspondant aux valeurs aberrantes de trajet, il est donc normal d'avoir un si faible résultat.

Le pavé 5 étant celui qui contient le plus d'arrivées, on se propose de faire une analyse plus précise par mois. Nombre de trajets arrivant dans le pavé 5 par mois :

Mois/Année	Nombre de trajets
07/13	56 406
08/13	48 868
09/13	58 198
10/13	60 917
11/13	54 664
12/13	55 652
01/14	50 362
02/14	50 653
03/14	54 097
04/14	52 944
05/14	61 772
06/14	58 512
07/14	102

On voit que le nombre de trajets de taxi par mois arrivant dans ce pavé est relativement constant, entre 48 000 et 62 000. Le mois de l'année n'a donc à priori pas d'impact sur les trajets de taxi.

Le faible résultat pour le mois de juillet 2014 vient du fait que la récupération des données s'est fini le 1er juillet 2014 (inclus), nous avons donc les données d'un seul jour pour ce mois.

On peut pousser l'analyse encore plus loin en regardant pour un mois particulier le nombre de trajets par jour et pour un jour particulier le nombre de trajet par heure. On remarque qu'il y a un creux de trajets entre 1h et 8h du matin. J'ai fait la même requête pour le 1er janvier 2014 et on ne retrouve pas cet écart. Cela s'explique par le fait que beaucoup de gens font la fête durant la nuit pour le nouvel an et donc ont besoin d'un taxi à des heures matinales.

mois	jour	count
10	1	1984
10	2	1917
10	3	1729
10	4	2383
10	5	2296
10	6	1939
10	7	1784
10	8	1788
10	9	1812
10	10	2135
10	11	2350
10	12	2325
10	13	2127
10	14	1969
10	15	1894
10	16	1929
10	17	1798
10	18	2245
10	19	1740
10	20	1690
10	21	1810
10	22	2044
10	23	1758
10	24	1984
10	25	2067
10	26	2115
10	27	2035
10	28	1742
10	29	1754
10	30	1850
10	31	1924

Nombre de trajets pour les jours du mois d'octobre 2013

annee	mois	jour	heure	count
2013	8	28	0	113
2013	8	28	1	96
2013	8	28	2	95
2013	8	28	3	72
2013	8	28	4	60
2013	8	28	5	77
2013	8	28	6	86
2013	8	28	7	65
2013	8	28	8	99
2013	8	28	9	185
2013	8	28	10	219
2013	8	28	11	231
2013	8	28	12	219
2013	8	28	13	201
2013	8	28	14	199
2013	8	28	15	230
2013	8	28	16	261
2013	8	28	17	230
2013	8	28	18	219
2013	8	28	19	194
2013	8	28	20	175
2013	8	28	21	156
2013	8	28	22	128
2013	8	28	23	135

Nombre de trajets pour les heures du 28/08/2013

On veut à présent les statistiques pour un chauffeur de taxi donné (*taxi_id = 20000454*). On récupère ci-dessous le nombre de trajets de ce chauffeur pour chaque mois.

taxi_id	annee	mois	count
20000454	2013	7	532
20000454	2013	8	416
20000454	2013	9	411
20000454	2013	10	426
20000454	2013	11	321
20000454	2013	12	256
20000454	2014	1	330
20000454	2014	2	415
20000454	2014	3	419
20000454	2014	4	209
20000454	2014	5	477
20000454	2014	6	469
20000454	2014	7	1

On remarque que ce taxi fait plus de trajets de mai à octobre par rapport aux autres mois.

b) k-means

Les k-means vont nous permettre de voir la typologie géographique des trajets en fonction de requêtes par rapport au temps ou aux taxis. La méthode des k-means est une méthode statistique de partitionnement de données.

L'idée est de résumer l'ensemble des trajets (représentés par des vecteurs dans $\mathbb{R}^4$) par un nombre fixé de centroïdes. Cela permettra donc de voir une tendance générale de nos données. On ne gardera pas de traces de la « classe » de chaque trajet.

On choisit de représenter un trajet de cette façon :

[latitude_départ, longitude_départ, latitude_arrivée, longitude_arrivée]

Si on pose **k = 3** le nombre de centroïdes, l'algorithme simplifié est le suivant :

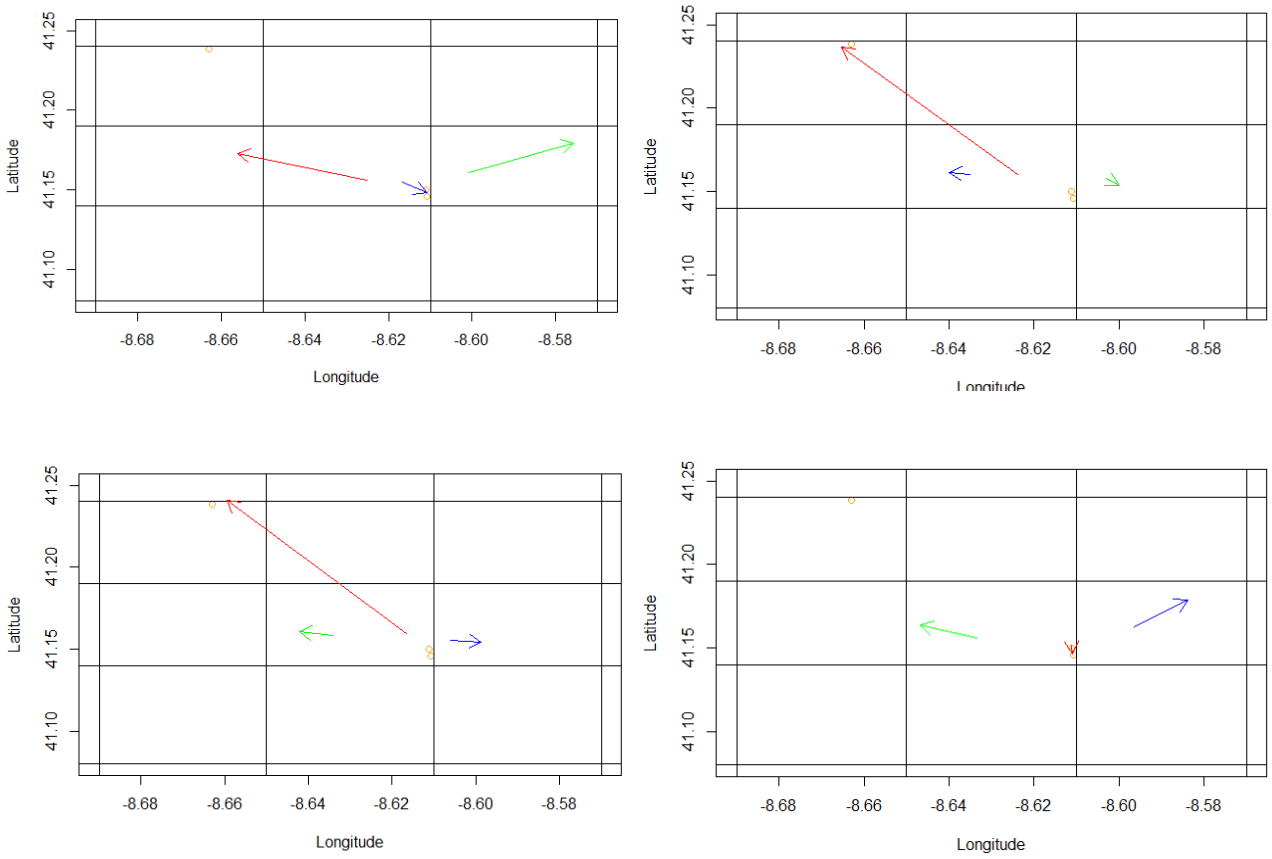
```
Tirage au sort de 3 centroïdes initiaux parmi les trajets existant
Tant que les nouveaux centroïdes ne sont pas égaux aux précédents faire
      Somme = [[0,[0,0,0,0], [0,[0,0,0,0], [0,[0,0,0,0]]
      Pour tous les trajets t de la requêtes faire
            Calcul de la distance entre t et chaque centroïde
            Recherche du centroïde k le plus proche
            Somme[k][0] ++
            Somme[k][1] += t
      nouveaux centroïdes = Somme[i][1]/Somme[i][0]
```

On récupère donc à la fin les centroïdes et on les représente par des flèches sur un graphique longitude/latitude. Le graphique est fait avec le logiciel R par choix. En effet, je maitrise mieux cet outil plutôt qu'une librairie de Python.

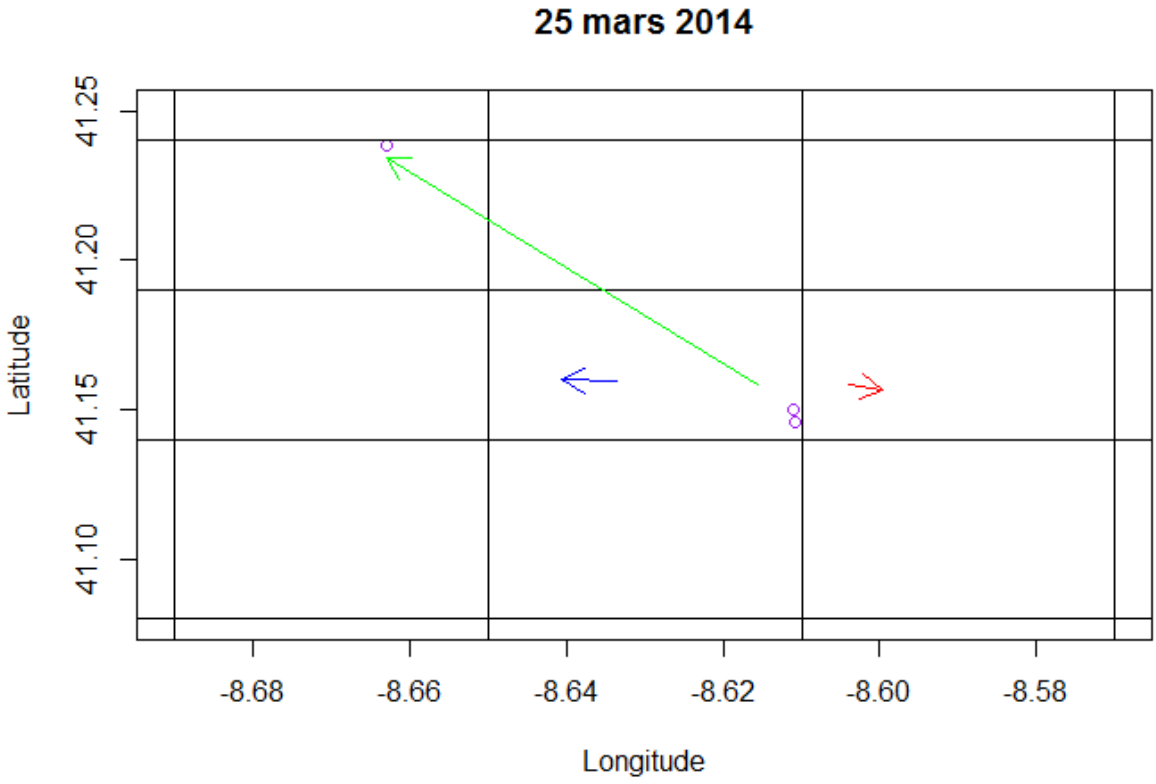
J'ai choisi de faire deux analyses en particulier à partir de cet outil. La première est d'identifier la typologie des trajets en fonction de l'heure de la journée. En effet, il serait intéressant de savoir à quel endroit un taxi doit se placer idéalement à une heure donnée pour avoir plus de chance d'être interpellé. On choisira pour cette étude les heures suivantes : 2h, 8h, 17h et 23h.

La seconde problématique sera de voir si il y a des différences de typologie de trajet en fonction du jour. On choisira alors un jour de l'été (Porto est une destination très touristique durant l'été), un jour normal et un jour spécial (le 31 décembre par exemple).

En plus des centroïdes, on affichera la position de l'aéroport de Porto, de la gare ferroviaire et de l'office de tourisme. L'aéroport correspond au point en haut à gauche de la carte alors que la gare et l'office du tourisme sont très proches, situés au niveau du centre-ville de Porto.



k-means pour les trajets effectués à 2h (haut gauche), 8h (haut droit), 17h (bas gauche) et 23h (bas droit)

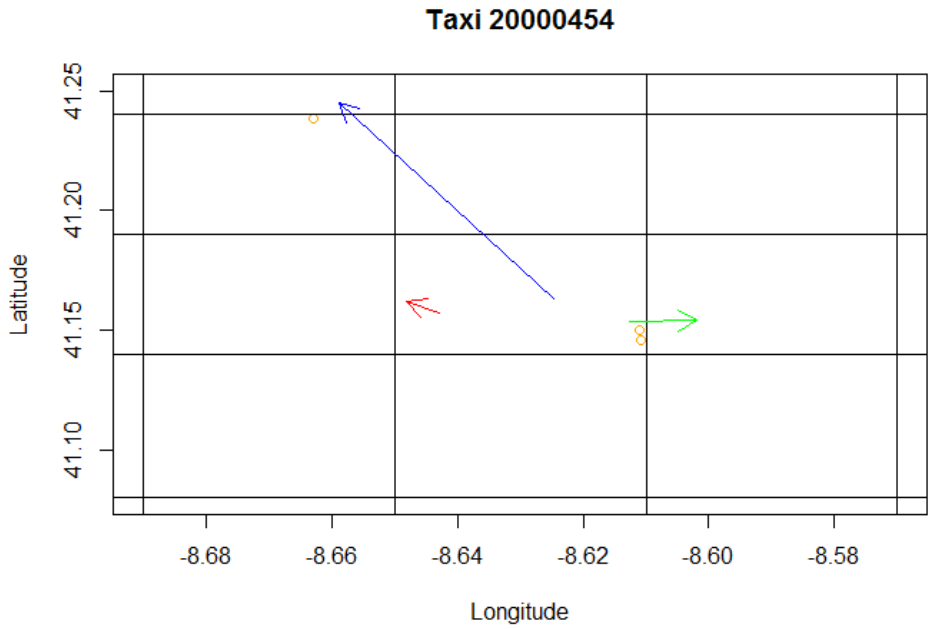


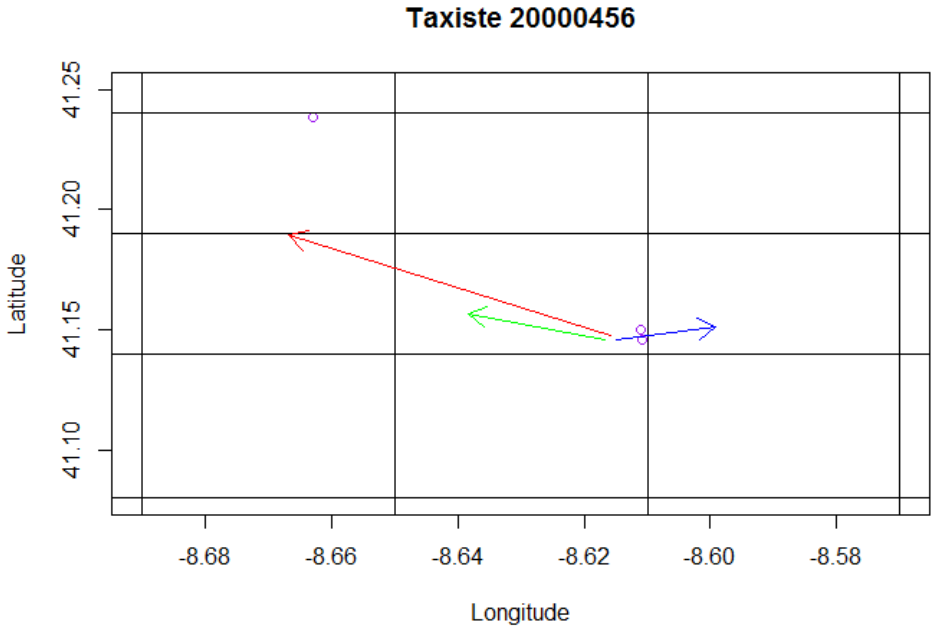
Typologie des trajets du 25 mars 2014

On retrouve pour les trajets ayant lieu à 8h et 17h des tendances similaires. Le trajet du centre jusqu'à l'aéroport de Porto correspond à la flèche la plus longue alors que les deux autres centroïdes partent du centre pour s'excentrer. Cela semble être la typologie des trajets de taxi à Porto « normale » en pleine journée.

Si on s'intéresse à des heures de nuits (23h et 2h), on ne retrouve pas le trajet vers l'aéroport. Les trajets sont plus courts et tous du centre vers la périphérie.

Pour finir, on peut comparer les trajets de deux chauffeurs de taxi différents.





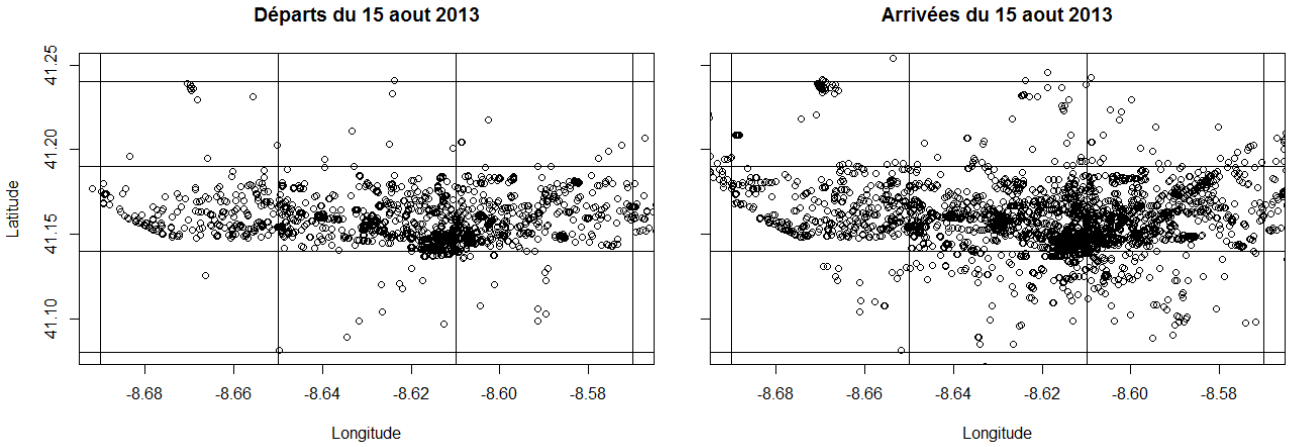
On voit bien que leur typologie de trajet est différente. Le chauffeur correspondant au taxi_id 20000456 ne semble pas faire beaucoup de trajets vers l'aéroport.

c) nuage de points

Le dernier outil que je vais utiliser avant de visualiser les données est un graphique de nuages de points où l'ordonnée du point correspond à la latitude et l'abscisse à la longitude. A partir d'une requête CQL et de code Python, un fichier .csv est obtenu en sorti avec les longitudes et latitudes de chaque trajet correspondant à la requête. J'ai choisi à nouveau d'utiliser le logiciel R pour tracer le graphique représentant mes données.

On peut afficher toutes les positions de départ et d'arrivée des trajets de taxi qui ont eu lieu le 15 août 2013. On retrouve bien un nuage de points au niveau de l'aéroport. On a l'impression qu'il y a plus de points à l'arrivée qu'au départ car les points sont plus dispersés dans la figure de droite. Pour les départs, tous les points sont très condensés.

De plus, tous les trajets sont condensés sur le pavé horizontale du milieu, cela confirme bien l'observation faite plus haut comme quoi les pavé ne sont pas homogènes et devraient être redécoupés.



En analyse générale des données, on voit bien que les points stratégiques pour les chauffeurs de taxi sont le centre-ville et l'aéroport qui est un peu excentré. Néanmoins, peu de trajets sont effectués depuis ou vers l'aéroport la nuit, il est donc plus judicieux dans ces heure-là de se focaliser sur le centre-ville. De plus, le nombre de trajets entre 1h et 8h du matin est beaucoup plus faible que la journée, ce n'est donc pas des heures très propices de travail. La répartition sur l'année des trajets est relativement constante et n'augmente pas particulièrement pendant la saison touristique. Un piste d'amélioration serait donc d'essayer de viser plus les touristes.

Pour plus d'informations, nous aurions pu faire des analyses par rapport au type de jour (férié, vacances, normal…) et par rapport au type d'appel. De plus en faisant une table avec comme clé primaire ((année, mois, jour), taxi_id, trip_id), nous aurions pu faire des « classements » de chauffeur de taxi par jour par exemple.

Conclusion

Ce projet a été une découverte à plusieurs niveaux : base de données non relationnelle, volume de données, langage Python… Dans un premier temps, il a été compliqué de s'adapter à « l'état d'esprit » du NoSQL, mais on se rend rapidement compte des intérêts apportés.

Le travail fait n'est pas exhaustif, d'autres tables des faits pourraient être créées et une analyse plus poussée pourrait être faite. Néanmoins, l'objectif était de découvrir des outils de reporting que les bases de données relationnelles ne permettent pas nécessairement de faire, se familiariser avec la notion de clé de partitionnement et clé de clustering et les requêtes en CQL. En effet, si on essaye de faire les mêmes requêtes qu'on fait habituellement dans des bases de données relationnelles, cela ne marche pas forcément.

De plus, je me suis rendue compte en voulant faire des requêtes particulières que ma table n'était pas forcément adaptée et il n'était pas toujours possible de le modifier. Je pense qu'il faut donc s'appuyer sur des requêtes afin de créer la table et donc réfléchir à l'analyse avant la conception de la base de données.

Ces TD ont été un premier pas vers le *big data* et j'ai trouvé cela vraiment intéressant de se confronter à cette problématique d'actualité. On réfléchit plus avant de lancer un script qui insère 1.7 millions de lignes plutôt que 3 000.