Projet NF26 –TD1 7/8

Lorys Hamadache — Romain Creuzenet

21 Juin 2019

1 Introduction

Le but de ce projet est de manipuler diverses données climatiques issues de plusieurs stations réparties sur le globe. C'est l'Espagne qui nous a été affecté et nous avons pris la liberté d'utiliser la période de temps de 2010 à 2013 plutôt que celle de 2001 à 2010, car cette dernière ne comportait pas suffisamment de données (2 stations seulement avec moins de 1000 entrées).

L'enjeu principal est le stockage pertinent des données dans nos bases de données NoSQL Cassandra et d'exploiter ces résultats avec le langage Python pour répondre à plusieurs objectifs. Notre code se doit d'être adapté à une forte quantité de données.

2 Stockage des données

Le stockage des données est la partie la plus importante lors de la réalisation d'un tel outil d'analyse. C'est pourquoi on s'intéresse ici aux données et objectifs de notre outil afin de réaliser un stockage adapté.

2.1 Les données

Nous nous intéressons ici au cas de l'Espagne. L'Espagne consiste en 62 stations réparties sur tout le territoire espagnol continental en passant par les Îles Canaries et les Îles Baléares. Nos données contiennent 30 variables comme la température ou l'humidité. Les données sont téléchargeables en un seul fichier où chaque ligne représente une mesure avec sa station, les cordonnées de celle-ci (lat, lon), la date de la mesure (Précision : Minute) et les mesures des différentes variables. Nous avons pour la période choisie (2011 - 2013) environ 1.5 M d'entrées. On remarque aussi que nous avons des variables toujours nulles, ainsi que de nombreuses variables souvent nulles.

Nous utilisons un stockage en colonne à l'aide de Cassandra mis à notre disposition sur 3 clusters. Notre système se doit d'être toujours opérationnel même lors de l'absence des certaines parties et toujours nous retourner une valeur. C'est aussi un modèle qui nous permet de manipuler plus facilement toutes nos variables incomplètes et les colonnes presque nulles ne prennent pas trop d'espace.

Nous utilisons seulement ces données-ci pour des questions de temps d'importation et de test. Il est plus simple de tester sur une quantité raisonnable de données pour ensuite porter le code en version finale avec toutes les données (2001 - 2019).

2.2 Les Tables et nos objectifs

Objectif 1 :Pour un point donné de l'espace, je veux pouvoir avoir un historique du passé, avec des courbes adaptés. Je vous pouvoir mettre en évidence la saisonnalité et les écarts à la saisonnalité.

Le premier objectif est, pour un point donné, autrement dit, pour l'une des stations espagnoles, d'obtenir un historique des données d'un attribut (par exemple l'humidité). On souhaite afficher et stocker des courbes expliquant l'évolution d'un attribut et sa saisonnalité. Comment stocker nos données afin de pouvoir les exploiter au mieux ?

Pour notre table, appelée TABLE_SPACE, nous avons donc utilisé comme clef de partition la station, permettant ainsi d'obtenir l'ensemble de ses mesures et de pouvoir en faire un historique. Pour différencier chacune de ces mesures, nous nous sommes servi de la date de celles-ci en clef de tri (année, mois, jour, minute, heure).

Dans notre exploitation, nous choisissons une station puis un seul attribut (pour plus de clarté) et que, dans ce cas, la possibilité d'utiliser l'attribut comme clef de partition, la date comme clef de tri et y associer l'ensemble des mesures de chaque station est possible. Or, si suite à une évolution, nous voulons faire, en une fois, l'historique de tous les attributs d'une station cela n'est plus possible, d'où notre choix. De plus, nous stockons des informations

non-essentielles comme la longitude ou la latitude. Cela est fait dans le cas d'une autre exploitation de la base.

Objectif 2: À un instant donné je veux pouvoir obtenir une carte me représentant n'importe quel indicateur.

Le second objectif fut d'obtenir, à un instant donné, une carte affichant les mesures d'un indicateur. La table de l'objectif 1 n'était plus appropriée. Nous avons donc créé une seconde table (TABLE_TIME) avec en clef de partition : la date (année, mois, jour, heur, minute), affin d'obtenir facilement l'ensemble des valeurs à un instant donnée. Comme clef de tri, nous aurions pu utiliser la station, mais nous avons préféré utiliser la longitude et la latitude car leur association correspond à une seule et unique station et ces données sont nécessaires pour ensuite pouvoir placer les mesures sur la carte.

Objectif 3: À un instant donné je veux pouvoir obtenir une carte me représentant n'importe quel indicateur.

L'objectif 3 consiste à effectuer une clusterisation des différentes stations par rapport aux données météorologiques sur une période de temps demandée. Nous faisons une moyenne de chaque attribut de chaque station sur la période étudiée, en récoltant les données en streaming.

Pour effectuer cet objectif, la table de l'objectif 2 a été reprise, car sa clef de partition est le temps. Une itération de requête est effectuée avec une date précise, en étant incrémentée de la plus petite unité de mesure : la minute. Nous aurions pu créer une nouvelle table partitionnée à l'heur ou au jour, mais avons choisi de garder la même table pour un souci de volume de données et en préconisant des périodes courtes. Un nouveau partitionnement aurait suivi les mêmes principes.

3 Notre outil et le traitement des données

Dans cette partie nous allons voir en détail comment nous avons réalisé un outil permettant de répondre aux 3 objectifs et les traitements associés afin de gérer la quantité de données potentielle.

3.1 Interface

Nous avons choisi de réaliser une interface simple textuel en Python. Lorsque le programme est executé (avec `>> python3 main.py`), du texte est affiché à l'utilisateur et ce dernier peut alors intéragir. Il est nécessaire de charger l'environnement Spark au préalable (`source /pyspark.env`). On se retrouve sur l'interface suivante où l'utilisateur peut faire un choix:

```
lhamadac@nf26-1:~/Projet$ python3 main.py
Choisissez ce que vous voulez faire
        1 - Pour un point donné de l'espace, je veux p
        2 - À un instant donné je veux pouvoir obtenir
        3 - Pour une période de temps donnée, je veux
>>>
```

Figure 1: Interface Générale.

3.2 Objectif 1

Pour réaliser l'objectif 1 nous avons créé la fonction `historic`. Comment gérons-nous la quantité de donnée dont nous avons besoin ? Dans un premier temps, la fonction est un prolongement de la partie interface. On demande à l'utilisateur de choisir la station sur laquelle il veut obtenir les courbes. Pour cela, on affiche toutes les stations ayant des données à l'aide d'une requête à la table TABLE_SPACE de notre base Cassandra où nous sélectionnons les stations distinctes. Il y a peu de stockage dans ce cas présent donc aucun problème concernant ce niveau. Puis, on demande à l'utilisateur quel indicateur il souhaite visionner (température, humidité ...).

Comment gérer le nombre important de données de la station ? Dans un premier temps on ne réalise la requête que sur ce que nous avons besoin, c'est-à-dire, la date et la valeur de l'attribut. Notre requête nous retourne un générateur. Mais celui-ci peut contenir trop de données. En effet, nous avons quelques fois plusieurs mesures par heure, pendant plusieurs années. Il est donc impossible de stocker tout cela. Nous décidons d'utiliser Spark afin de paralléliser nos traitements et réaliser des mappings et des reductions successifs. Nous mappons et réduisons nos données en ne gardant que le maximum, le minimum et la moyenne de l'attribut sur la journée. Ces traitements sont réalisés à la

suite (max puis min puis avg). Lorsque la quantité de données sera trop importante on pourra passer à une réduction par semaine ou par mois. Ces données, fortement réduites sont maintenant stockées dans une Time Series grâce à la bibliothèque `pandas`. Nous affichons ensuite sur un même graphique le maximum, le minimum et la moyenne de l'indicateur au cours du temps de cette station. On réalise une interpolation sur les données pour les jours manquants.

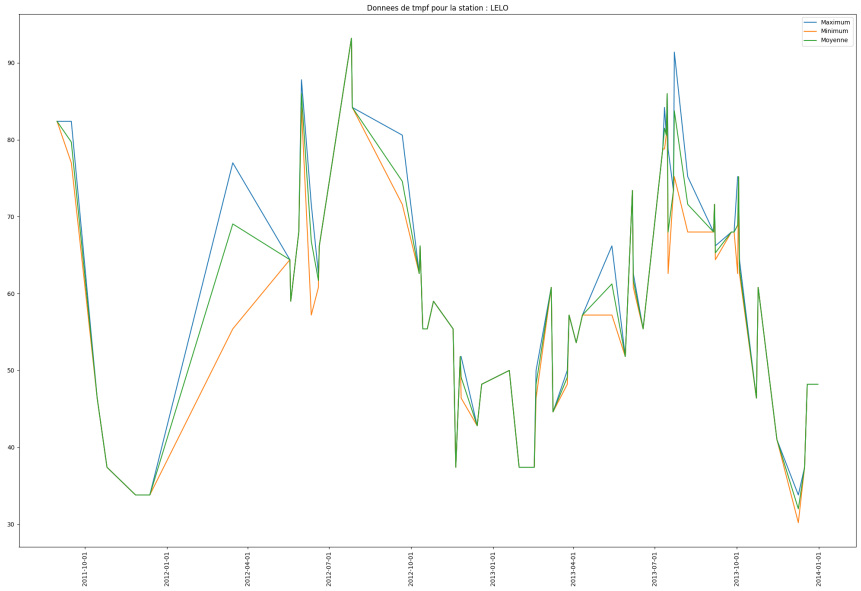

Figure 2: Exemple de tracé pour la station LELO et l'indicateur de température.

Avec la bibliothèque `statsmodels`, on en profite pour réaliser un graphique d'autocorrelation ainsi qu'une décomposition de notre time series en une composante de tendance, une composante saisonnale et les résidus. Cela nous permet de répondre entièrement à la question en étudiant la saisonnalité du signal ainsi que les résidus hors tendance et saisons. On peut voir sur la décomposition une tendance à la baisse de la température et clairement un rythme saisonnier par année.

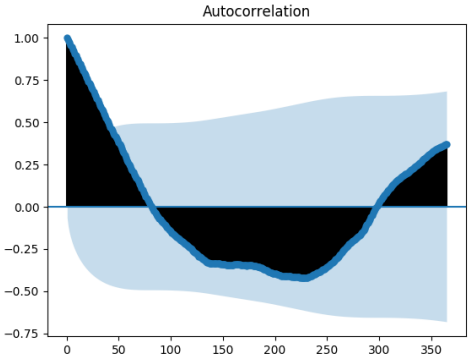

Figure 3: Exemple d'autocorellation pour la station LELO et l'indicateur de température.

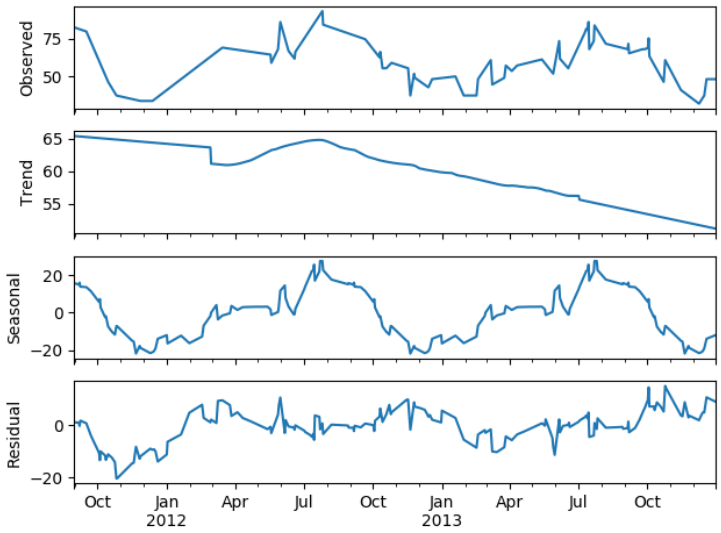

Figure 4: Exemple de décomposition pour la station LELO et l'indicateur de température.

3.3 Objectif 2

Dans un premier temps on demande une date et une heure et un attribut à l'utilisateur de la même façon que précédemment. Si la date n'existe pas, on lui affiche les heures disponibles avec des mesures dans le jour qu'il a demandé. Pour cet objectif, nous n'avons pas de problème de données. En effet, grâce à notre deuxième table (TABLE_TIME) nous pouvons faire une requête, à un temps et à un attribut donnés. La quantité de données est égale au nombre de stations multiplié par 4 (Nom de la station, longitude et latitude et valeur de l'attribut). Ensuite nous utilisons la bibliothèque `mpl_toolkits.basemap` pour afficher une carte de l'Espagne sur laquelle on place les stations (des points) et auquelmes la valeur de l'attribut est annotée. Cela permet de visualiser rapidement un indicateur à travers l'espace.

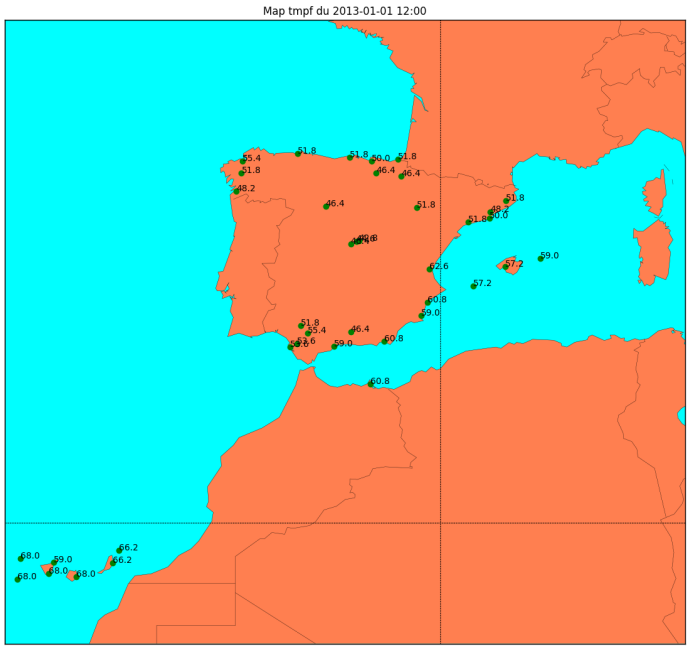

Figure 5: Température en Espagne le 1er Janvier 2013 à 12h00.

3.4 Objectif 3

En premier lieu, nous demandons à l'utilisateur la période sur laquelle il veut réaliser sa clusturisation. Après avoir obtenu le générateur de toutes les données sur cette période, on ne peut pas les stocker de cette façon pour raison évidente de coût mémoire, en particulier si la période est longue et que le nombre de données augmente. Pour cela, on calcule la moyenne par station de tous les attributs sur la période donnée, tout cela est réalisé en "streaming" pour ne pas avoir de problème de mémoire.

Le nombre de cluster souhaité est ensuite demandé à l'utilisateur. On réalise un algorithme des kmeans à la main. On pourrait paralléliser les calculs si nécessaire. Pour clusteriser, dans un premier temps, nous générons les centroïdes avec des valeurs aléatoires comprises entre le minimum et le maximum des moyennes recueillies pour chaque attribut. Nous clusterisons donc sur N dimensions, N étant le nombre d'attributs. La norme des vecteurs obtenus est comparée à celle de chaque station. De nouveaux centroïdes sont calculés par rapport à la moyenne des stations dont la norme de leur vecteur est la plus proche. Cette opération est effectuée jusqu'à une stabilisation des centroïdes. Si les centroïdes n'ont aucune station associée, les centroïdes sont renouvelées aléatoirement.

Une fois clusterisé, les stations sont affichées sur la carte comme dans l'objectif précédent. Leur appartenance à un cluster est visible par la couleur affichée. Sur l'exemple, on remarque qu'avec 3 clusters, quelques groupes logiques se dessinent. Nous avons en vert des stations côtières et en rouge des stations dans les terres.

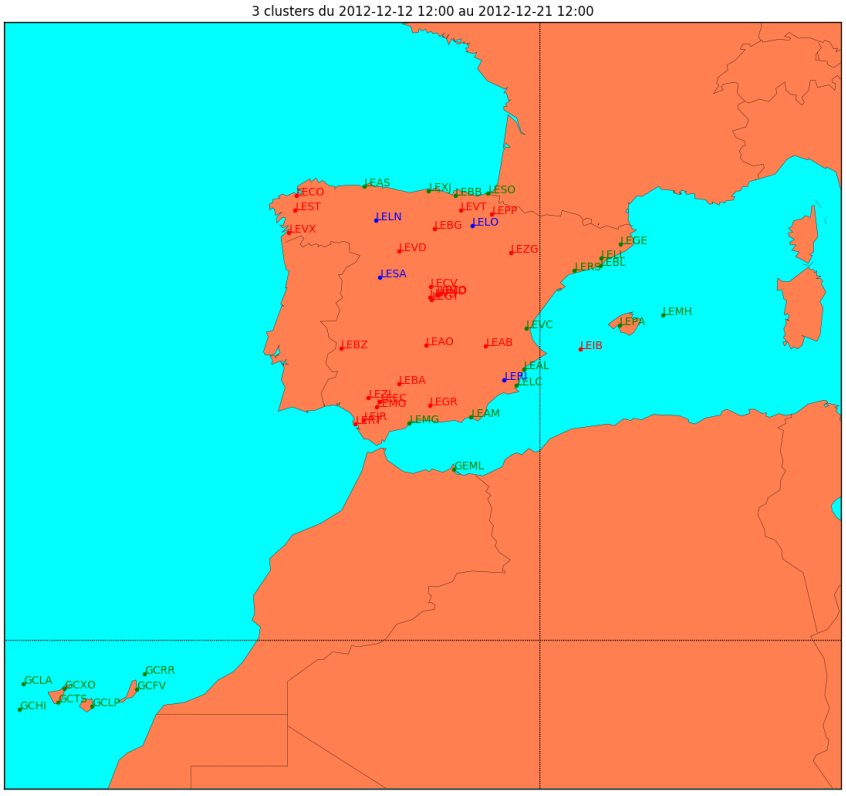

Figure 6: Clusturisation de l'Espagne, avec 3 clusters, calculé sur la période du 2012-12-12 à 12h au 2012-12-21 à 12h.

4 Architecture du projet

Le fichier `parameters.py`*:*
Il permet de stoker la configuration du projet (pays concerné, période de temps étudiée...). Il possède également les informations utiles à tout le projet.
Le fichier `requierments.txt`*:*
Ce fichier permet d'obtenir toutes les librairies à installer pour faire fonctionner le projet.
Le fichier `download_data.py`*:*
Ce fichier permet de télécharger les données du pays concerné sur la période temps étudiée depuis le site internet où elles sont stockées. Ces informations seront ensuite placées dans des fichiers csv par station dans le dossier `out`. Le code est inspiré de celui trouvé sur le site internet gardant les données. Ce fichier n'est utilisé que très rarement, à une initialisation. Il suffit de l'exécuter pour appeler les bonnes fonctions.
Le fichier `create_table.py`*:*
Ce fichier crée les différentes tables utilisées durant le projet et les remplie des informations contenues dans tous les fichiers du dossier `data`. Ce fichier n'est utilisé que très rarement, à la première utilisation. Il suffit de l'exécuter pour appeler les bonnes fonctions.
Le fichier `main.py`*:*
Ce fichier gère l'exploitation des données. Il est utilisé fréquemment. Son exécution permet d'obtenir une interface dans le terminal pour choisir son objectif, les éléments nécessaires et obtenir les résultats.
Le dossier `out`*:*
Ce dossier stocke les graphiques issues de l'exploitation des données. Ils peuvent être consulté ultérieurement.
Le dossier `data`*:*
Ce dossier contient tous les fichiers contenant nos données qui devront être mises dans Cassandra. Ce dossier n'est pas censé être exploité manuellement par l'utilisateur.

5 Conclusion

Ce projet est un projet intéressant pour se confronter à des données réelles. Cela nous permet de comprendre et de mettre en pratique nos connaissances sur les bases de données orientées colonnes. C'est en se posant des questions sur les données et en ne perdant pas de vu l'objectif final (pouvoir traiter énormément de données) que nous pensons à réaliser un traitement de données en "streaming" et à réaliser du mapping et de la réduction. Pour aller plus loin, on pourrait penser à paralléliser l'algorithme des kmeans ou utiliser la fonction existante dans Spark. De plus il serait intéressant d'étudier la performance en fonction du nombre de clusters et de répliquas utilisés.

Vous trouverez le code complet à cette adresse : https://gitlab.utc.fr/rcreuzen/nf26_projet/

Merci pour votre lecture