Rapport Projet Bigdata ISEP 2015

CHANTHAVONG Delphine
RASOLOMALALA Narisely
PHILIPPE Romain
DIVAD Loïc

Novembre 2015

Table des matières

Table des figures

1. Introduction

2. Environnement de travail, outils et organisation.

Pour pouvoir collaborer plus facilement, nous avons déployé un “cluster” Hadoop que nous installons et administrons nous-même. Cela nous a permis d'éviter des problèmes de ressources liés à l'utilisation des nombreux logiciels.

Le cluster est composé de 3 serveurs **PSV1, PSV2, PSV3** (ces abréviations sont utilisées dans la suite de la description de l'environnement) sur lesquels nous avons installé la distribution Hortonworks de Hadoop (HDP 2.2) et nous permet de travailler ensemble à distance.

Dans un premier temps, il nous faut connecter les serveurs entre eux. Pour cela, on génère avec le premier serveur (**PSV1**) une clé SSH qui sera copiée sur les deux autres serveurs (dans le fichier knowhost du user root).

Ensuite, on débute l'installation de l'écosystème. On commence donc par l'installation d'Ambari-serveur que l'on a placée sur **PSV1**.

Par la suite, lorsqu'Ambari serveur est lancé, il nous est demandé de fournir les adresses IP des deux autres serveurs du cluster. Ambari installe alors Ambari-agent sur chaque serveur qui lui se chargera d'installer tous les logiciels Hadoop (il nous propose une répartition des logiciels que l'on a ensuite adaptée à nos ressources)

Note sur les agents Ambari : Il consiste en une série de scripts python dédiée à de nombreuses tâches comme l'installation des paquets, la mise en place et l'autorisation des répertoires ou encore le redémarrage des services Hadoop.

Notre cluster est finalement installé. Nous présenterons donc une répartition des différents logiciels. Chaque machine possède 4GO RAM et possède un Datanode sur un file system de 1 TO.

Note	SV1	SV2	SV3
	Ambari serveur		
		namenode	
			sec namenode
		Yarn	Yarn
			Oozie
			Hbase
			Hive
	Zookeeper	Zookeeper	Zookeeper
Éteint		Storm	
Éteint		Kafka	
	Hue		
	Livy		
(client)	Tez	Tez	Tez
(client)	Pig	Pig	Pig
(client)	Sqoop	Sqoop	Sqoop

FIGURE 2.1 – Répartition des logiciels

3. Intégration de données

3.1 Description des fichiers

— *Data integration : Explain how the data has been recovered from the EnerNOC website and how it has been integrated into HDFS and a Hive database.*

Le dataset est hébergé sur une plateforme amazon[1]. Il s'agit d'un open data, ces informations sont accessibles et utilisables par tous. Elles sont mises à disposition par EnerNoc[2], un éditeur de logiciels orienté dans la consommation d'énergie.

Divers formats sont proposés par l'entreprise, mais nous avons uniquement téléchargé les sources de données au format CSV. L'archive contenant les fichiers est alors téléchargée sur une des machines disposant d'un client Hadoop.

```
$ wget https://open-enernoc-data.s3.amazonaws.com/anon/csv-only.tar.gz
```

Puis décompressée :

```
$ tar -zxvf all-data.tar.gz
```

On obtient alors un sous dossier de données sources et un sous dossier de "meta data" servant à expliquer les fichiers. On rédige ici une description des fichiers.

Tous les fichiers portent l'extension CSV et ont pour unique titre un identifiant[3]. Il y en a 100. Tous les csv ont un poids proche de 4.5 Mo (± 0.2 Mo) pour 105409 lignes chacun. Le dossier total pèse 460 Mo. Ils présentent tous une première ligne d'en-tête et 105408 autres lignes correspondant chacune à une mesure pour un site à un instant donné. Les mesures sont prises toutes les 5min pendant un an. Ce qui signifie qu'il y a 5min entre chaque ligne du csv. Les colonnes sont les suivantes : timestamp, dttm_utc, value, estimated, anomaly.

timestamp	Date de la mesure au format POSIX[4]
dttm_utc	Conversion de la date au format UTC (yyyy-MM-dd HH :mm :ss)
value	Consommation en kWh
estimated	Boolean pour savoir si la valeur est estimée
anomaly	Indicateur pour savoir si la valeur est erronée

FIGURE 3.1 – Description des fichier xxx.csv

Ces fichiers sont à croiser avec le fichier de meta data. Il présente une ligne par site étudié.

[1] Lien de téléchargement du jeu de données : *Les archives*

[2] EnerNoc : www.enernoc.com

[3] La correspondance entre les sites et les identifiants des fichiers est établie dans le fichier de méta data qui sera vu plus bas.

[4] Nombre de secondes écoulées depuis 1er janvier 1970 00 :00 :00 UTC

SITE_ID	Identifiant du site. C'est le nom du fichier contenant ces mesures.
INDUSTRY	Secteur du site
SUB_INDUSTRY	Sous catégorie du secteur du site
SQ_FT	Surface du site en pied carré
LAT	Latitude du site
LNG	Longitude du site
TIME_ZONE	Nom du fuseau horaire du site
TZ_OFFSET	Heure de décalage par rapport au méridien de Greenwich

FIGURE 3.2 – Description du fichier all_sites.csv

Note : Dans une démarche pragmatique, avant l'insertion des données dans HDFS nous avons visualisé quelques CSV au hasard à l'aide d'outils mieux maîtrisés. Nous utilisons donc le langage python pour tirer quelques conclusions qui nous permettrons de mieux appréhender les questions. On indique dans la remarque suivante les points qui ont attiré notre attention.

Remarques :

- Il n'y a que 4 industries prises en compte
- Les industries sont bien équilibrées en nombre de sites (25sites / industrie)
- 5 fuseaux horaires différents sont répertoriés
- On trace brièvement les 2000 premiers points (7j) du site 100 (Education)

De toutes ces remarques découlent les intuitions suivantes. Il peut être intéressant de créer une table partitionnée par industrie. Les Load Curves peuvent varier plus ou moins entre 25 et 5 kWh en une demie journée.

Nous insérons maintenant le dataset dans le filesystem d'hadoop. Pour une meilleure exploitation des fichiers nous retirons les entêtes pour chaque fichier avant l'insertion. Une fois placé dans le dossier contenant les mesures, on applique la commande suivante :

```
$ ls | xargs -n 1 sed -i 1d
```

Remarque : Même si les fichiers contenant les mesures ont un encodage correct et un délimiteur de champ/ligne adapté, le fichier de meta données lui est problématique. En effet, les lignes sont séparées par le caractère ^M. Hue propose différents délimiteurs pour les champs mais pas pour les lignes. Le problème est résolu avec la commande suivante :

```
$ sed -i -e "s/\r/\n/g"
```

Les fichiers sont alors finalement tous prêts à être insérés dans le filesystème hadoop. La partie suivante explique comment nous les avons transférés après le nettoyage.

3.2 Script d'intégration

Pourquoi un script d'intégration ? Dans un premier temps, l'interface hue ne propose pas l'intégration d'un dossier entier. Il peut donc vite être fastidieux de sélectionner les 100 fichiers. Les fichiers ne sont pas sur nos postes personnels mais sont déjà présents sur le serveur. Et enfin, l'opération sera a priori répétée un grand nombre de fois, il peut être intéressant de l'en-capsuler avec quelques fonctionnalités utiles.

La commande principale est la suivante :

```
$ hdfs dfs -put <fichier/dossier>
```

Description de la commande :

- hdfs fait appel au client hadoop
- dfs indique qu'il s'agit d'une commande du filesystem (à différencier des commandes de lancement de logiciels, de formatage ou encore de configuration)
- -put[5] la commande à exécuter

Nous plaçons donc cette commande dans un script simple qui prendra en compte les choses suivantes :

- L'utilisateur du script est bien le user commun
- La cible du script est bien un dossier
- Compter le nombre de fichiers et demande une confirmation
- Retour d'un code erreur

Le script est alors exporté sous le nom "push" par l'utilisateur commun (panda) et s'utilise de la façon suivante :

```
$ push -t <dossier>
```

Finalement, dans un système d'information dans lequel un jeu de données (comme celui proposé par Enernoc) arriverait au fur et à mesure, il peut être intéressant d'automatiser l'importation dans le file system hadoop avec ce genre de script.

```
1 Script start with "panda" privileges.
2 100 file(s) from csv will be put in the HDFS
3 Please, confirm this operation (y/n) y
4 Start acces to the hadoop-client.
5 Script terminate with code: 0
```

FIGURE 3.3 – Prompt du script d'intégration des fichiers

La création du script était une démarche pertinante si on considére un vrai Système d'information. Dans ce cas les fichiers porraient arriver de manière périodique. L'import des data dans le file système peut alors être automatisé et retourner des codes erreurs.

[5] Pour le projet nous nous appuyons sur la documentation : *Commandes hdfs*
[6] Retrouvez la totalité du script dans note *répository*

4. Modélisation des données

4.1 Stratégie

— *Data & Modeling : What data model and representation model should you use in your Hive database ? Why ? What issues will you have to deal with if you have to manage the same data type for 10 million sites ?*

Pour mener à bien le projet, notre équipe propose une stratégie en trois étapes. Toutes les mesures seront accumulées dans une table externe all_records. C'est une table en ligne et les fichiers y sont stockés au format TEXTFILE (cela revient à concaténer tous nos csv). Cette table représente un niveau immuable des données, les lignes n'y sont ni modifiées ni supprimées. On s'appuie sur elles pour effectuer les requêtes de remplissage des tables de travail : work_table<description>. Cette fois, la table est interne et la donnée y est compressée. Le modèle de données et les requêtes de remplissage dépendent alors de la portée de l'étude[7] (ceux que nous avons exploités seront décrits dans cette partie). Enfin, le résultat des différentes agrégations est stocké dans des vues HIVE afin d'y simplifier l'accès.

```
1 LOAD DATA INPATH '/user/panda/enernoc/' OVERWRITE INTO TABLE tibet.all_records;
```

FIGURE 4.1 – Requête de remplissage de la table all_records

4.2 Modélisation

4.2.1 Point de départ : les tables Enernoc

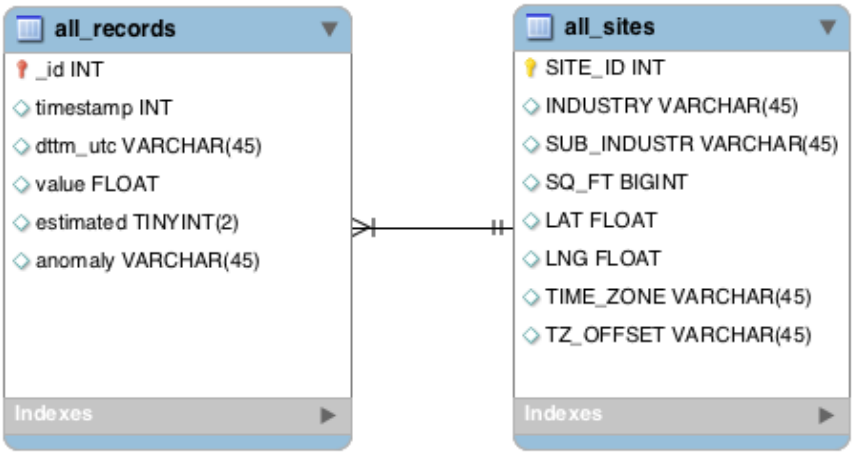

FIGURE 4.2 – Entité-Relation relative aux tables de départ fournies pas Enernoc

Note : Lien entre `all_site` et `all_records`. Dès les premières injections de données et créations de table il semble évidant que le lien entre ces deux tables nécessite une jointure. Celle-ci

[7] Dans notre cas nous étudions les 100 sur toute l'année. La table construite avec "select * ..."

se réalise sur les champs : all_sites.SITE_ID et all_records. Comme expliqué en III.1 dans la description des fichiers, les mesures portent en nom de fichier l'id du site mais il ne se retrouve pas dans les colonnes. Nous effectuons donc la jointure sur la méta donnée INPUT__FILE__NAME qui retourne l'adresse complète de la ressource. (ex : hdfs ://ns3099426....eu :8020/apps/hive/warehouse/project.db/enernoc/474.csv) Notre première UDF[8] consistera alors à parser ce chemin pour obtenir l'id, ici **474**. Cette infomation est accessible pour chaque ligne de la table.

```
1 public Text evaluate(Text input){
2     if (input == null) return new Text("");
3     final String path = input.toString();
4     final int index = path.lastIndexOf("/");
5     final int offset = path.lastIndexOf(".");
6     return new Text(path.substring(index + 1, offset));
7 }
```

FIGURE 4.3 – Implémentation de l'UDF : filename

Une fois l'UDF crée et le projet compilé le fichier *.jar* est insérer dans Hue en temps que Jar temporaire Hive.

```
1 SELECT all_sites.site_id, all_sites.industry,
2 all_records.dttm_utc, all_records.value
3 FROM all_sites JOIN all_records
4 ON filename(all_records.INPUT__FILE__NAME) == all_sites.site_id;
```

FIGURE 4.4 – Exemple d'utilisation de l'UDF filename

On peut observer à la ligne 4 l'utilisation de la fonction filename. A l'avenir pour ne plus avoir à répéter cette jointure on contruira une vue s'appuyant sur cette reqête.

4.2.2 Difficultés rencontrées liées à la modélisation : La table colonne

En partant de la définition même de la LD (dont le calcul est le principal objectif du projet), on comprend que de nombreuses agrégations vont devoir être exécutées. Nous envisageons donc dans un premier temps de ranger les données en colonne avec une table proche de ce format.

	site1	site2	site3	site4	site5	site100
timestamp5						
timestamp10						
etc						

FIGURE 4.5 – Première approche du sujet. Ce format de table à été abandonné

Quels sont les inconvénients et pourquoi nous ne l'avons pas mise en place ? Dans un premier temps, il devient difficile de faire des agrégations sur les dates, il faut sommer les différentes colonnes entre elles. Les colonnes étant variables, il devient très difficile d'imaginer la gestion d'un plus grand nombre de sites comme les 10 millions proposés dans le sujet. De plus, cette représentation semble impliquer une forte perte d'information.

Finalement, les difficultés liées à l'augmentation du nombre de sites sont les suivantes :

- Temps de calcul des loadcurves
- Scalabilité des programmes (l'augmentation du temps de calcul suit l'augmentation du nombre de données)
- Profondeur des jointures (plus le nombre de sites augmente plus la jointure évoquée en III.2 ralentit la remontée des résultats)
- Agrégations de plus en plus coûteuses (avec une durée d'étude qui s'élargit aussi dans le temps, il sera de plus en plus difficile de calculer la LD)

4.2.3 Les solutions privilégiées

Après plusieurs essais pour obtenir les résultats demandés (LD) dans la suite du projet, notre équipe s'est tournée vers différentes solutions de dénormalisation. Elle reposent principalement sur les types introduits par Hive comme Array<T1> ou Map<T1, T2>[9]. Une table de travail peut répondre à plusieurs requêtes.

Note : les requêtes que nous avons regroupées ensemble ne sont pas forcément ordonnées. Ainsi, la première table nous permet de traiter les LD 1 et 3, puis on traite ensemble 2 et 4, enfin 5, 6 et 7 avec la troisième table de travail. Les LD seront donc abordées dans cet ordre dans le reste du rapport et lors de la soutenance.

`datetime` [STRING]	`Values` [Array<FLOAT>]
"2012-01-20 00:05:00"	[,]
"2012-01-20 00:10:00"	[,]

FIGURE 4.6 – Modélisation de la table : work_table_site_axis

Cette table est utilisée pour le calcul des LD 1 et 3. On y retrouve toutes les dates de mesure une seule fois et pour chaque date de mesure correspond un tableau de valeurs qui regroupe les mesures de tous les sites observés. On a donc 305409 lignes, et sur chacune un tableau de 100 éléments. Cette table est particulièrement efficace pour l'agrégation SUM qui est demandée.

`datetime` [STRING]	`industry` [STRING]	`values` [Array<FLOAT>]
"2012-03-15 07:20:00"	Commercial Property	[,]
	Light Industrial	[,]
	Education	[,]
	Food Sales &Storage	[,]
"2012-03-15 07:25:00"	Commercial Property	[,]
	... etc	[,]

FIGURE 4.7 – Modélisation de la table : work_table_industry_axis

Cette table est similaire à la précédente mais présente un découpage supplémentaire avec une colonne `industry`. Elle est d'ailleurs partitionnée par industrie. On a donc plus de lignes (4*305409) et seulement des tableaux de 25 valeurs par lignes. La table permet alors de réaliser les moyennes par industrie demandée dans les questions 2 et 4.

Note : Nous nous sommes demandés s'il était préférable de n'utiliser que cette dernière table puisqu'elle peut répondre aux deux types de questions (1 à 4) moyennant quelques agrégations supplémentaires. Mais le gain de temps qui va avec la première table n'est pas négligeable.

date	season	value	id	industry	sub_indus	sq_ft	lat	lng

Cette table est utilisée pour le reste des LD. Ici, on répète énormément d'informations, le but est de supprimer les jointures nécessaires pour répondre aux questions, et obtenir les résultats plus vite. L'exemple suivant présente le temps d'exécution de la requête pour la première LD avec et sans remise enforme des tables.

[8] Les UDFs sont réunies dans un *projet java* dans lequel toute l'équipe a contribué.
[9] Ou T1 & T2 sont des types de base, comme ceux vu en cours : STRING, FLOAT BIGINT etc.

```sql
-- Requete d'origine -> 16:41 min --
SELECT dttm_utc, SUM(value) FROM all_records GROUP BY all_records.dttm_utc ORDER BY dttm_utc;
-- Requete sur la tables de travail -> 2:08 min --
SELECT datetime AS datetime, arraysum(values) AS total
FROM work_table_site_axis
ORDER BY datetime;
```

FIGURE 4.8 – Comparaison du temps d'execution des requêtes sur les tables de travail.

Conclusion sur la modélisation :
Pour mener à bien les analyses demandées nous accepterons donc une perte d'information sur certaines tables mais aussi la duplication de certaines informations. De sorte que les requêtes s'exécutent le plus vite possible. La dénormalisation nous permet de rehausser certaines agrégations de manière à ne plus avoir à les refaire. On remarque aussi que la compression au format ORC accélère également les requêtes. Le contre coût est le temps de remplissage des tables (voir partie IV). Mais une fois mise en place, elle permet un gain de temps. Ce temps doit toujours être limité si l'on veut placer des applications (de production) en bout de système. Quitte à laisser les tables se remplir lors de batch nocturnes.

5. Calculs des Load Curves

Comme indiqué dans la partie III. 3, nous regroupons ici le calcul des LD qui se ressemblent. Le découpage se fera donc en trois groupes :

- Somme pour l'ensemble des sites
- Moyenne par industrie
- Recherche de maximum par jour

5.1 Somme pour l'ensemble des sites

- Calculate the sum LD for the 100 sites (timestamp interval : 5 minutes)
- Calculate the total LD for the 100 sites (timestamp interval : a week)

Hive propose plusieurs types complexes, dont Array<T>. C'est une structure de données indexées qui peut contenir plusieurs éléments de type T. On s'en servira comme les types : INT, FLOAT ou encore STRING en l'assignant à une colonnes values. Le code suivant est la requête Hive pour créer la table `work_tabl_axis`, vue à la figure (...)

```
1  CREATE TABLE tibet.work_table_site_axis(
2    datetime STRING,
3    values ARRAY<FLOAT>
4  )
5  COMMENT 'Work table, values containe all sites measures for the same datetime.'
6  PARTITIONED BY (season STRING)
7  CLUSTERED BY(datetime) INTO 4 BUCKETS
8  ROW FORMAT DELIMITED
9  FIELDS TERMINATED BY ','
10 STORED AS ORC;
```

FIGURE 5.1 – Création de la première table de travail pour les LD 1 & 3

Cette table va donc nous permettre d'itérer sur moins de lignes (nombre divisé par 100 ce qui nous donne uniquement "305409 lignes"). Également elle nous permet de supprimer une agrégation coûteuse (GROUP BY datetime).

Note important : Nous aurions pu remplir cette table directement avec la somme demandé dans le sujet. Mais conservant la structure d'Array d'autres opérations, comme la Moyenne, le min, le max peuvent être réalisée grâce à cette table.

Une fois construite, elle est alors remplie à l'aide de la commande suivante. Lors de la dernière exécution de cette requête le jobbrowser indique un temps de remplissage de 24:51 min :

```
1  FROM view_work_table
2  INSERT OVERWRITE TABLE work_table_site_axis PARTITION (season)
3  SELECT dttm_utc AS datetime,
4  COLLECT_LIST(value) AS values
5  season(dttm_utc) AS season
6  GROUP BY dttm_utc ORDER BY dttm_utc;
```

FIGURE 5.2 – Remplissage de la table work_table_site_axis

[10] Resources : Documentation Hive, *Language manual UDF*

La table que nous avons construite un Array de type <FLOAT>. Nous nous appuyons donc sur la documentation Hive[10] pour trouver une UDF prédéfinie retournant un type Array. `Collect_list` est une UDAF qui va regrouper les éléments de la colonnes value conformément à l'agrégation GROUP BY `dttm_utc`.

Note :la table `view_work_table` est une vue réalisant la jointure entre les tables `all_records` et `all_sites`. Cette jointure est nécessaire pour le remplissage de toutes les tables que nous allons créer. Nous l'utilisons donc pour simplifier les requêtes de remplissage.

Remarque (sur l'UDF collect_list) : la fonction `COLLECT_LIST` est à bien différentier de `COLLECT_SET`, qui elle, ne conserve pas les dupliquas. Une valeur ne peut alors paraître qu'une fois dans l'array. Après avoir rencontré cette nuance nous évaluons la construction de la table avec la commande suivante :
`SELECT datetime, size(values) FROM work_table_site_axis ORDER BY datetime;`
La requête indique que la plupart[11] des colonnes values propose bien 100 valeurs. Une fois la table remplie, on exécute les deux requêtes suivantes qui vont respectivement répondre aux questions 1 et 3.

```
1 SELECT datetime AS datetime, arraysum(values) AS total
2 FROM work_table_site_axis
3 ORDER BY datetime;
```

FIGURE 5.3 – Requête de calcul de la loadcurve 1

le tableau suivant présente la forme du résultat de la requête précédente. 105407 linges sont retournées. Le JobBrowser indique une durée de `2:08min`.

datetime	total
2015-01-01 00:05:00	***
2015-01-01 00:10:00	***
...	...

FIGURE 5.4 – Forme de la réponse pour la LD1

```
1 SELECT getweek(datetime) AS week, SUM(arraysum(values)) AS total
2 FROM work_table_site_axis
3 GROUP BY getweek(datetime)
4 ORDER BY week;
```

FIGURE 5.5 – Requête de calcul de la LD3

le tableau suivant présente la forme du résultat de la requête précédente. 52 linges sont retournées. Le JobBrowser indique une durée de `2:12min`.

datetime	total
week n°1	***
week n°2	***
...	...

FIGURE 5.6 – Forme de la réponse pour la LD3

A la ligne 1 de la figure 5.5 la fonction `getweek` est une UDF crée par notre groupe. Elle prend un en entré un text au format date ou datetime (`yyyy-MM-dd` ou `yyyy-MM-dd HH:mm:ss`) et retourne un numéro de semaine. En cherchant dans la documentation Hive on trouve aussi

[11] On remarque que certaines mesures peuvent sauter. Il y a des cellules de 99 ou 98 valeures
[12] 54 lignes avec la fonction getweek, 52 lignes avec la fonction weekoftheyear

`weekoftheyear` qui retourne un nombre de 1 à 53. On conserve alors notre fonction qui se différencie en indiquant l'année et le numéro de semaine. Elle peut donc potentiellement être utilisée sur plusieurs années. C'est aussi l'occasion de présenté la dépendance Joda-Time[13]. Le code suivant présente la foncion pour récupérer les numéros de semaine.

```
1   /**
2    * Take a datetime format like: 2011-12-31 01:05:00 <br/>
3    * Return {YEAR} / week {week number} <br/>
4    * Usage: SELECT getweek(dttm_utc) as week, SUM(value) FROM ...
5    * GROUP BY getweek(dttm_utc);
6    * @param dateTime
7    * @return
8    */
9   public Text evaluate(Text dateTime){
10    DateTime dtt = Bamboo.dateParse(dateTime.toString());
11    if (dtt == null ) return new Text("NULL");
12    String w = dtt.getWeekyear() + " / week:" + dtt.getWeekOfWeekyear();
13    return new Text(w);
14  }
```

FIGURE 5.7 – Code de la fonnction getweek.

Enfin la fonction `arraysum` est aussi une UDF crée par le groupe Pandas. Elle réalise simplement la somme des éléments de type Array<FLOAT>. Contrairement à `weekoftheyear` nous ne trouvons pas d'UDF prédéfinie permétant de d'exécuter cette tâche, bien qu'il existe d'autres fonctions comme `size`, `explode` etc... Nous **supposons** donc que cette fonction n'exite pas à cause du type paramétrique. En effet le type de la collone est Array<T>. Si l'oppération "+" n'est pas définie pour le type T la somme n'est pas possible. Par exemple faire la somme d'un array de char ne veut rien dire.

```
1   /**
2    * Take a column name with a Hive Array(float) type and <br/>
3    * return the sum of the column for the current row.
4    * @param col
5    * @return
6    */
7   public FloatWritable evaluate(ArrayList<FloatWritable> col){
8     float res = 0;
9     for (FloatWritable c : col){ res += c.get(); }
10    return new FloatWritable(res);
11  }
```

FIGURE 5.8 – Code de la fonnction arraysum.

[13] *Joda-Time* Library java de gestion des dates. L'intégralité du code (documenté et testé) pour le projet Bigdata peut être récupéré sur le *Répository du Groupe Pandas*

5.2 Moyenne par industrie

5.3 Recherche de maximum par jour

Les tables précédentes ne peuvent pas répondre à ces trois dernières requêtes. On en crée donc une troisième qui correspondra à un autre type d'étude. Dans cette partie, on observe les mesures par jour, contrairement aux tables précédentes où les lignes correspondaient à des mesures prises toutes les 5 minutes. Elle contient donc 366 lignes et ce remplie en prêt de 16:38min

date	season	value	site_id	sub_industry	sq_ft	lat	lng	industry	industry

FIGURE 5.9 – Format de la table work_table_day_axis

La requête de remplissage est un peut plus complexe.

```
1  FROM all_sites JOIN ( SELECT
2    to_date(dttm_utc) AS date,
3    SUM(value) AS value, filename(INPUT__FILE__NAME) AS site_id FROM all_records
4    GROUP BY filename(INPUT__FILE__NAME), to_date(dttm_utc) ORDER BY date
5  ) AS tab ON tab.site_id  = all_sites.site_id
6  INSERT OVERWRITE TABLE work_table_day_axis PARTITION (industry)
7  SELECT tab.date AS date, season(tab.date) AS season,
8  tab.value AS value, tab.site_id AS site_id,
9  all_sites.sub_industry AS sub_industry,
10 all_sites.sq_ft AS sq_ft,
11 all_sites.lat AS lat,
12 all_sites.lng AS lng,
13 all_sites.industry AS industry
14 ORDER BY date, site_id;
```

FIGURE 5.10 – Requête de remplissage de la table work_table_day_axis

Une fois remplie on réalise la requête pour obtenir le ratio entre la consomation par jour et la surface de chaque site. La somme pour chaque sites retourne donc 4 ligne en 3:32min.

```
1  SELECT industry, SUM(value/sq_ft) AS ratio FROM work_table_day_axis
2  GROUP BY industry ORDER BY ratio;
```

FIGURE 5.11 – Requête de calcul de la LD5

La requête suivante est juste un variante. On y modifie uniquement la clause GROUP BY. Ce nouveau découpage renvoie donc 16 lignes en un temps sensiblement identique 3:48min.

```
1  SELECT industry, SUM(value/sq_ft) AS ratio FROM work_table_day_axis
2  GROUP BY industry ORDER BY ratio;
```

FIGURE 5.12 – Requête de calcul de la LD5BIS

industry	season	energy intensity
Education	WINTER	49.4558152795
Education	AUTUMN	50.2533932526
Education	SPRING	52.8670349976
Education	SUMMER	59.7216345381
Commercial Property	AUTUMN	109.140407874
Commercial Property	WINTER	115.552683121
Commercial Property	SPRING	121.860600182
Commercial Property	SUMMER	134.317169258
Food Sales & Storage	WINTER	278.465446131
Food Sales & Storage	AUTUMN	292.341505588
Food Sales & Storage	SPRING	312.841578989
Food Sales & Storage	SUMMER	358.074345503
Light Industrial	WINTER	2051.96315678
Light Industrial	SPRING	2102.53981548
Light Industrial	AUTUMN	2315.32341343
Light Industrial	SUMMER	2441.62428432

FIGURE 5.13 – Power consuption par indutry et par saison

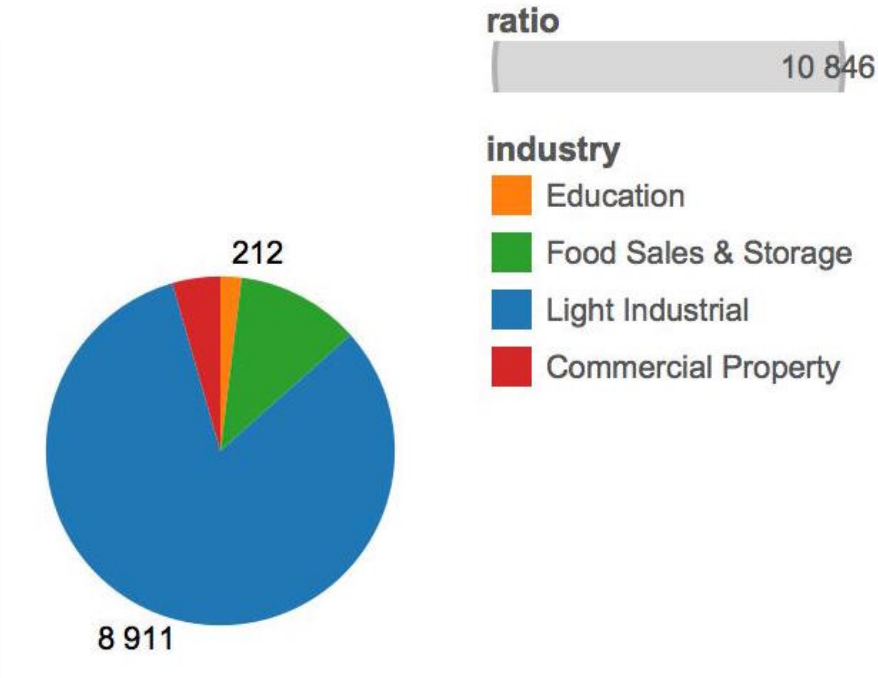

FIGURE 5.14 – Pie chart du ration consomation/suface par industry

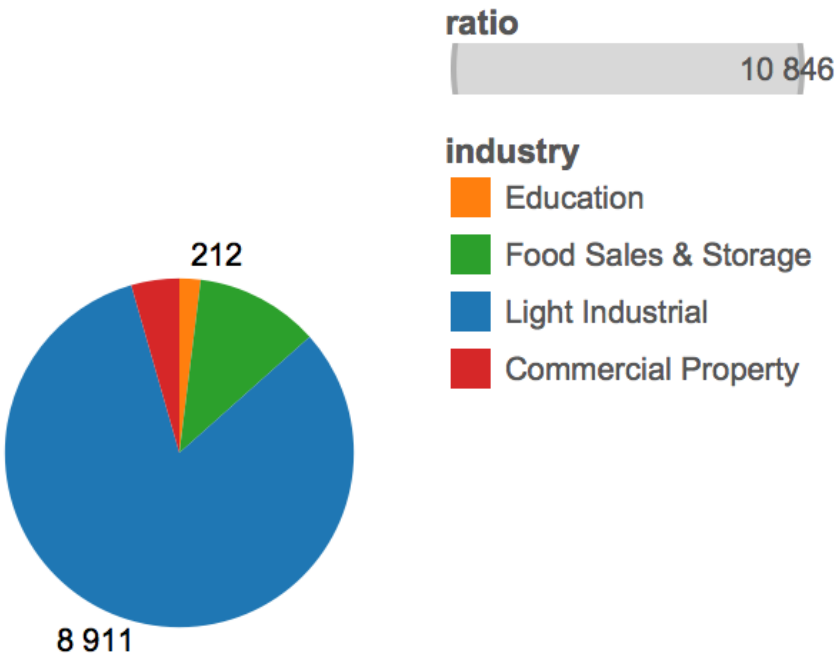

FIGURE 5.15 – Pie chart du ration consomation/suface par industry

6. Data visualisation

6.1 Tableau Software

Pour visualiser toutes les analyses effectuées dans les précédentes questions, nous avons utilisé Tableau Software qui est l'outil de visualisations mais aussi d'analyses par excellence.

Dans le but d'optimiser les performances de Tableau Software, nous créons dans Hive une table qui agrège les données par jour et qui contient également l'identifiant du site car on ne peut pas avoir accès au metadata à partir de Tableau (en l'occurence d'id du site).

Tableau Software aurait énormément de mal à gérer plusieurs millions de lignes. C'est pour cela que nous avons créé cette table qui compte environ 37000 lignes.

Ensuite, nous nous servons du connecteur natif Hive de Tableau. Comme nous avons aussi besoin du détail de chaque site, nous faisons une jointure sur le champ `id_site` comme le montre la figure ci-contre :

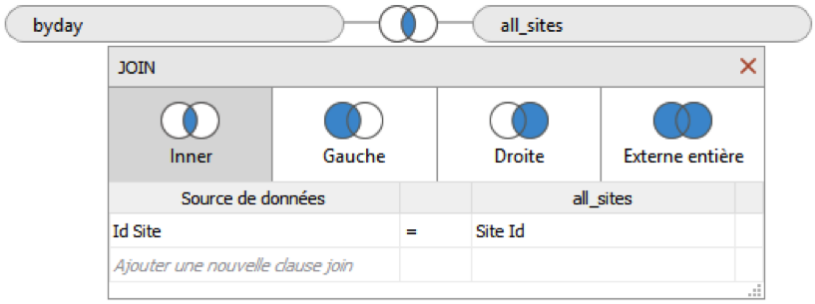

FIGURE 6.1 – Sources de données et connection aux tables hive

Enfin nous avons créé un extrait de cette jointure en local pour pouvoir travailler plus rapidement.

Nous pouvons alors mettre en valeur les données collectées, les analyser et prendre des décisions en fonction des constatations. On peut imaginer que ces analyses peuvent permettre de mieux répartir la consommation des différents sites et ainsi réduire les coûts. Vous trouverez ci-dessous des exemples de visualisations :

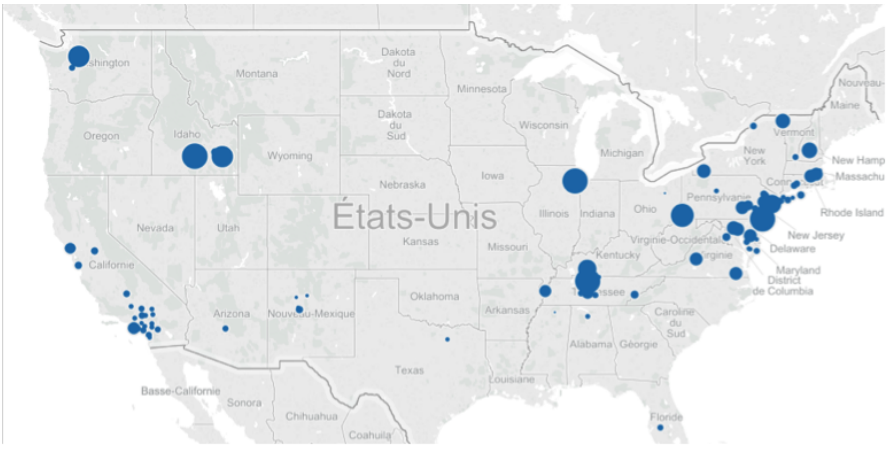

FIGURE 6.2 – Consommation moyenne annuelle par site

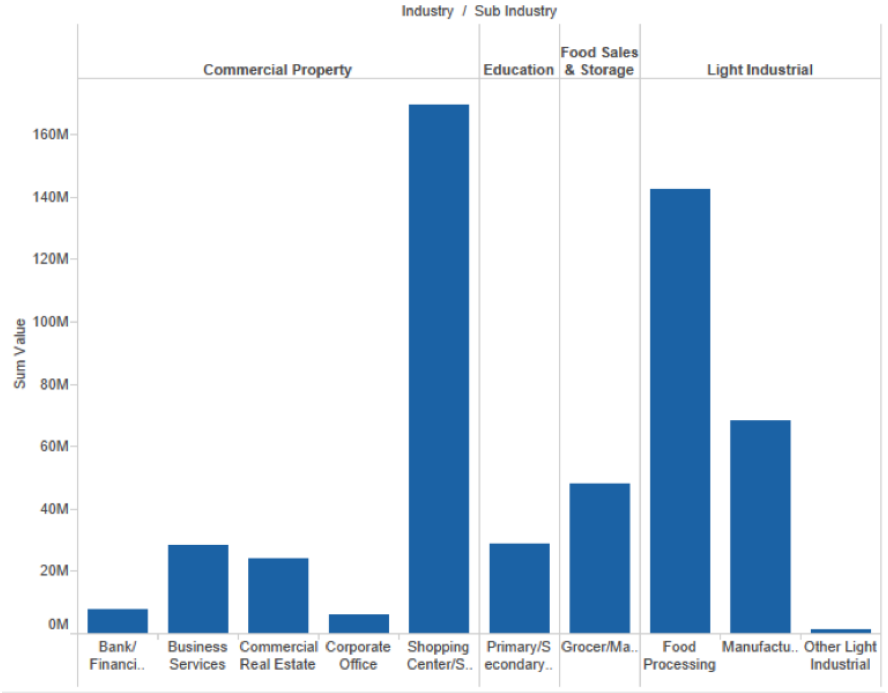

FIGURE 6.3 – Consommation moyenne annuelle par industrie et sous industrie

6.2 Bonus : corrélation entre la température et la consommation électrique

La mise en corrélation de la température et la consommation énergétique est composée de plusieurs étapes :

- Recherche de la source de données
- Associer chaque site à une station météorologique
- Extraire les données températures
- Corréler les données de température avec la consommation

6.2.1 Recherche de la source de données

Tout d'abord, nous avons recherché les différentes sources de données disponibles en matière de données climatiques, à savoir les APIs (*http://openweathermap.org/history*) ou les serveur

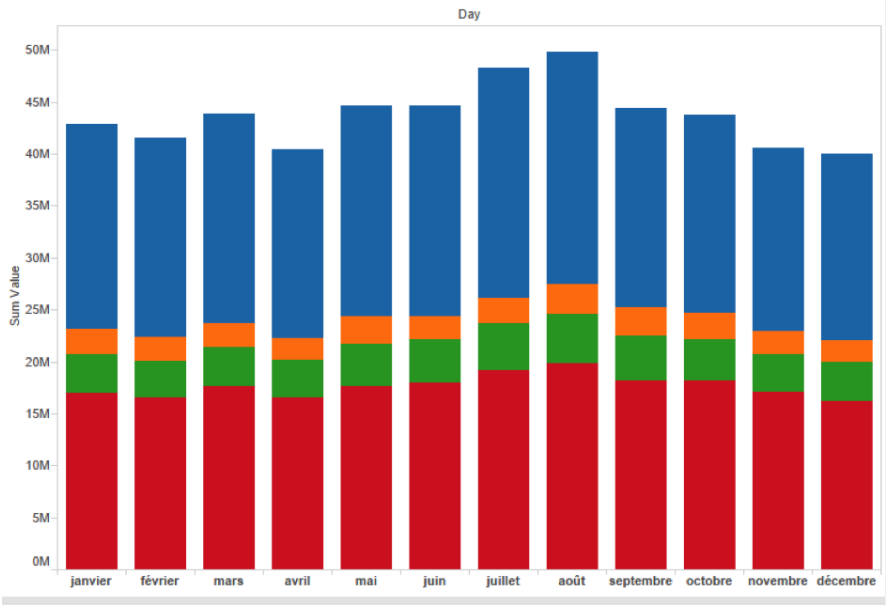

FIGURE 6.4 – Répartition de la consommation moyenne mensuelle par industrie

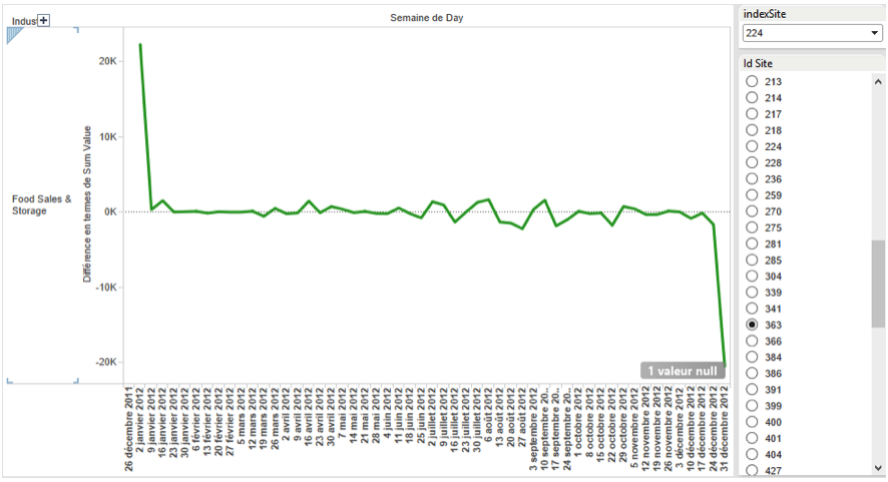

FIGURE 6.5 – Possibilité de choisir une industrie et faire la différence par rapport à la moyenne de son industrie

FTP.

Avec les APIs, on peut directement envoyer les coordonnées GPS du site ainsi que la date de début et la date de fin et on obtient la température. Nous avons très vite écartés cette car nous avons trouvé que la plupart des services web étaient payants pour les données historiques de températures (environ 150$ par mois *http://openweathermap.org/price*).

Nous nous sommes alors tourné vers les données collectées par l'état américain : NOAA (National Centers For Environmental Information). Cet organisme fournit des données de qualité sur différents indicateurs, force du vent, humidité et température moyenne. Il met à disposition des serveurs FTP. Voici plusieurs liens utiles :

-
- *ftp.ncdc.noaa.gov/pub/data/gsod/readme.txt* permet de comprendre les données, les unités, les différentes colonnes, le lien entre les fichiers etc.
- *ftp.ncdc.noaa.gov/pub/data/noaa/isd-history.csv* liste les stations météorologiques disponibles.
- *ftp.ncdc.noaa.gov/pub/data/gsod/2012/* contient l'ensemble des données de l'année 2012 pour les différentes stations.

Nous avons téléchargé ces fichiers en local pour avoir un aperçu des données dans le but de connaître la véracité de ces dernières.

6.2.2 Associer chaque site à une station météorologique

Une fois la source de données téléchargée, nous devons associé chaque site à la station météo la plus proche avec un script R. Nous avons codé un script R qui permet d'associer chaque site à une station. Nous nous sommes aperçu que l'on ne retrouvait pas toutes les stations du référentiel dans le détail des fichiers. Il a donc fallu calculer, pour chaque site, la distance avec les stations par ordre croissant jusqu'à trouver une correspondance (entre le référentiel et détail des données). Nous avons rajouté trois colonnes au fichier original all_sites.csv : deux colonnes pour identifier la station : `id_station` et WBAN ainsi que distance (distance entre la station et le site). Pour information, nous avons trouvé une distance moyenne entre le site et la station de 20 km, avec une distance maximale de 90 km. Nous pensons que 20km n'est pas une distance très importante par rapport à la surface des Etats-Unis et que les températures obtenues sont des bons indicateurs.

6.2.3 Extraire les données de températures

Cette étape consiste à parcourir les sites, rechercher dans les différents fichiers la station correspondante et à extraire uniquement l'information qui nous intéresse, c'est-à-dire la température moyenne par jour.

A la fin, nous obtenons un fichier d'environ 35000 lignes au lieu de 36000. Nous pouvons donc constater qu'il manque 1500 lignes (365 jours * 100 sites). En effet toutes les stations ne fournissent pas l'année 2012 entière.

Ensuite nous avons crée une table (`byday_temperature`) sur Hive avec les données obtenues : date , température, identifiant du site.

6.2.4 Corréler les données de température avec la consommation

Pour corréler ces données nous avons joint trois tables `byday`(consommation électrique totale par jour), `byday_temperature` (temperature par jour par site) et `all_sites` (détails des sites) comme le montre la figure ci-dessous.

De la même manière que pour le IV, nous avons extrait les données pour une question de performance. Nous pouvons alors montrer la corrélation entre les deux variables (en bleu la consommation et orange la température) sur le graphe ci-dessous :

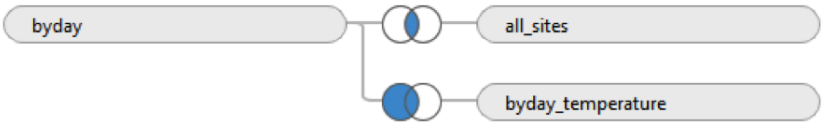

FIGURE 6.6 – Sources de données

De la même manière que pour le IV, nous avons extrait les données pour une question de performance. Nous pouvons alors montrer la corrélation entre les deux variables (en bleu la consommation et orange la température) sur le graphe ci-dessous :

Nous pouvons d'ores et déjà affirmer qu'il y a une relation entre température et consommation. On observe que lorsque la température diminue, la consommation augmente. Lorsqu'il fait froid, on a besoin de plus d'énergie pour chauffer les bâtiments. Pour mettre en évidence cette corrélation, nous pouvons représenter la consommation électrique en fonction de la température par industrie (voir ci-dessous).

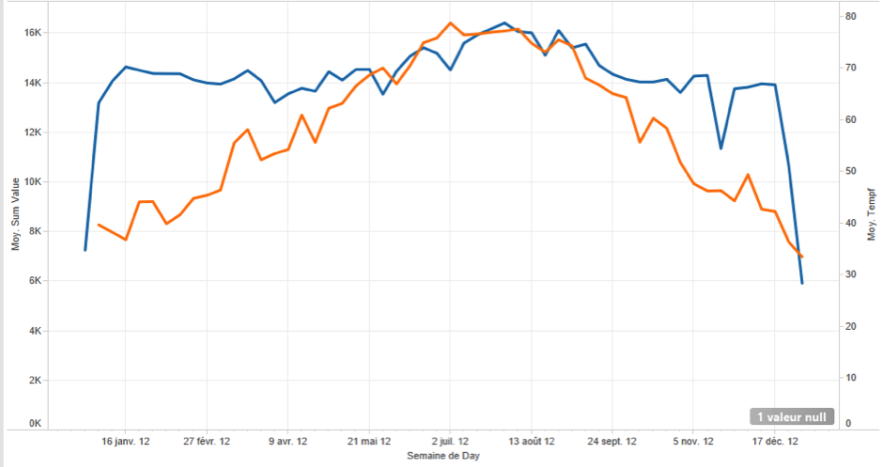

FIGURE 6.7 – Comparaison entre température et consomation éléctrique

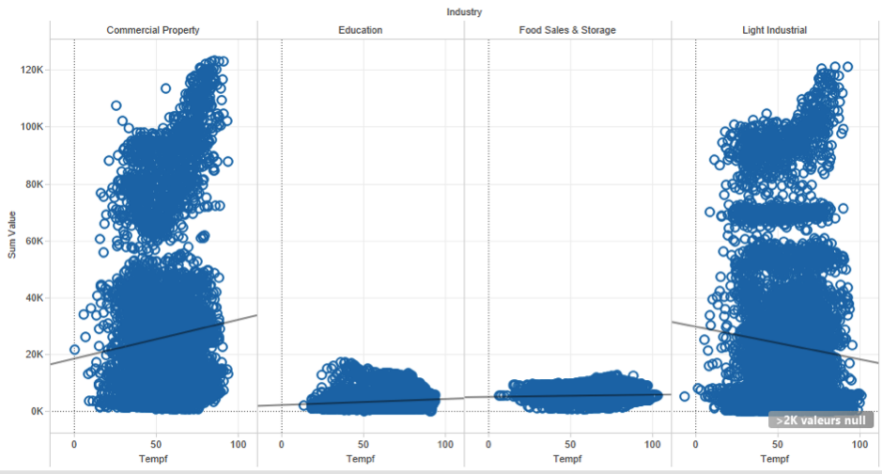

FIGURE 6.8 – Regression Consomation / Température

Tableau permet de décrire cette corrélation, nous nous intéresserons plus particulièrement à la valeur “R au carré”. Pour obtenir le facteur de corrélation, il suffit de mettre au carré cette valeur, c’est-à-dire : 0.465. On apprend également que le facteur Industry est significatif lorsque la p-value est inférieure à 0.0001.

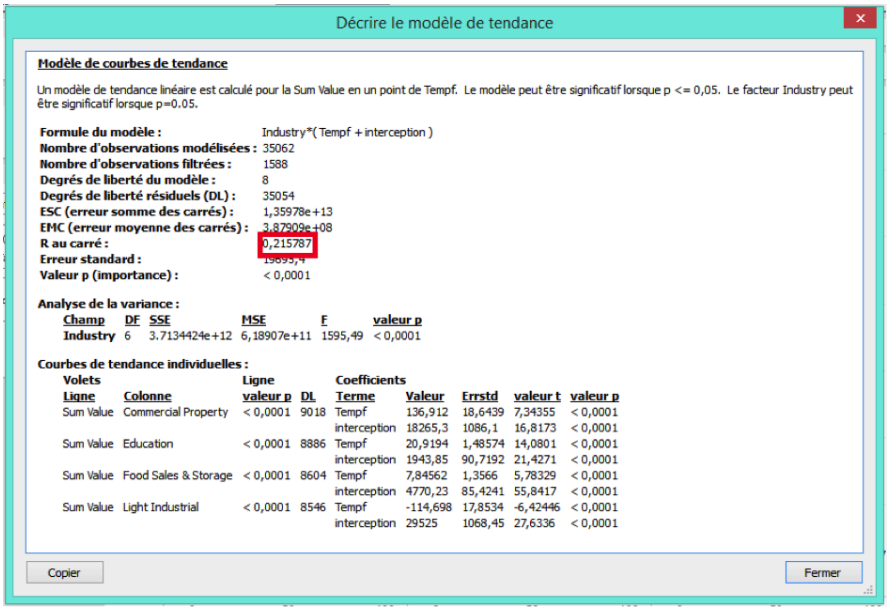

FIGURE 6.9 – Description du model réalisé par tableau

7. Conclusion

A. Team work & code sources du projets

A.1 Workflow de l'équipe Panda

A.2 Exploration et plot sous python

A.3 Recherche des températures avec R

B. Matériel

C. Ressources