



PROJET BIG DATA

---

## Traffic Analyzer

---

**Auteurs :**

Mohammed JAMIL  
Walid MHIDOU

**Encadrant :**

David AUBER

## Table des matières

<b>1</b>	<b>Présentation du sujet</b>	<b>2</b>
1.1	Problématique . . . . .	2
1.2	Objectifs . . . . .	3
<b>2</b>	<b>Solution proposée</b>	<b>4</b>
2.1	MapReduce . . . . .	4
2.2	Architecture du code . . . . .	4
<b>3</b>	<b>Normalisation et nettoyage des données</b>	<b>5</b>
3.1	Format des enregistrement . . . . .	5
3.2	Mapper de normalisation . . . . .	6
<b>4</b>	<b>Analyse des données</b>	<b>7</b>
4.1	Choix des analyses . . . . .	7
4.2	Mapper et Reducer des analyses . . . . .	8
<b>5</b>	<b>Écriture dans HBase</b>	<b>9</b>
<b>6</b>	<b>Réalisation du Dashboard</b>	<b>10</b>

## Introduction

Ce rapport décrit le projet "Traffic Analyzer" qui vise à nettoyer et normaliser les données collectées par les capteurs installés sur le domaine universitaire, et à mettre en place un tableau de bord pour visualiser les données analysées. Ce rapport aborde les différentes étapes de la solution proposée, depuis la normalisation et le nettoyage des données jusqu'à la réalisation du tableau de bord, et se compose de six sections : Présentation du sujet, Solution proposée, Normalisation et nettoyage des données, Analyse des données, Écriture dans HBase, et Réalisation du Dashboard.

## 1 Présentation du sujet

### 1.1 Problématique

Nous disposons d'un ensemble de capteurs installés sur le domaine universitaire. Ces capteurs permettent de détecter le passage d'un type de véhicule et du sens de son sens de circulation sur les différentes entrées de l'université. Le type de capteur utilisé ainsi que leur localisation sur le campus sont donnés sur la carte 1 :

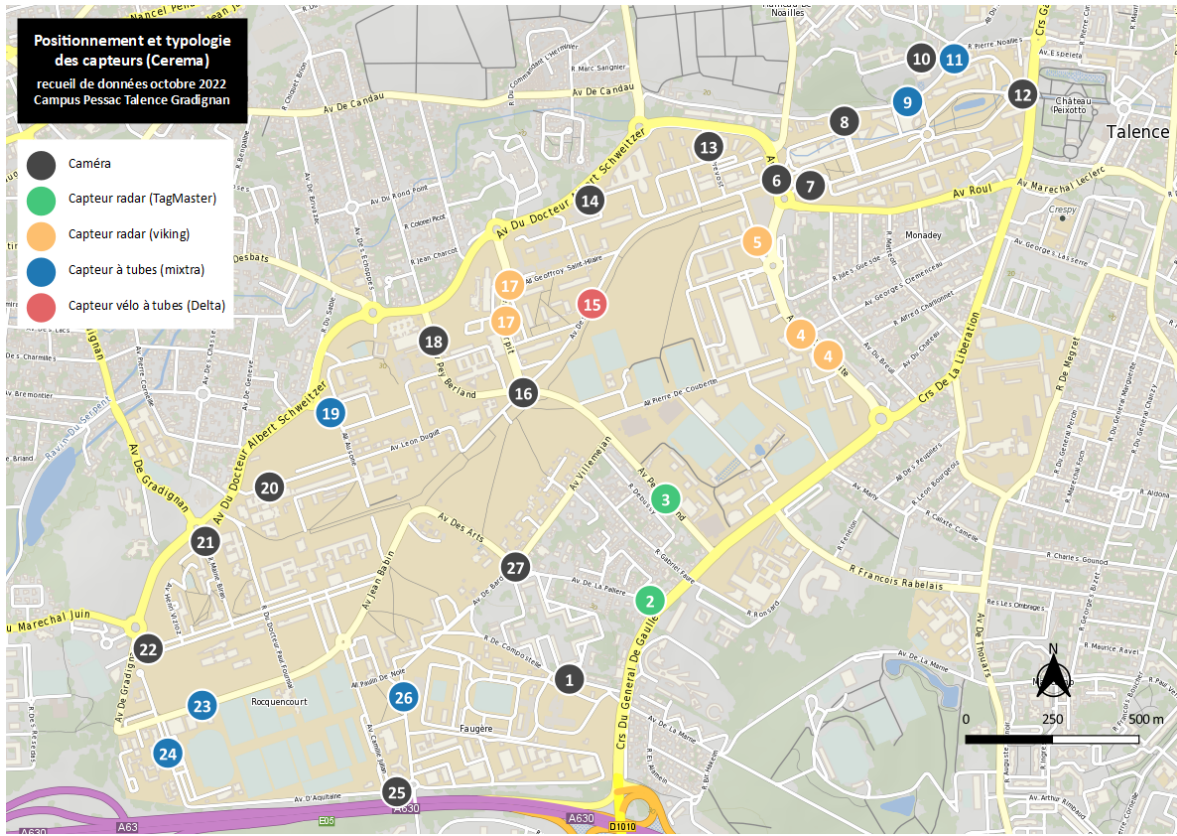


FIGURE 1 – Carte montrant la position des différents capteurs

Ces capteurs nous permettent d’obtenir des informations de différent types et dans différent format en fonction des capteurs.

## 1.2 Objectifs

L’objectif du projet est de nettoyer et normaliser les données collecter par les différents capteurs et mettre en place un Dashboard de visualisation des données analysés. Dans un premier temps, nous avons étudié le contenu des différents fichiers .csv des différents capteurs afin de déterminer un format commun pour tous les enregistrements que nous avons stocké par la suite dans un seul SequenceFile sur HDFS. Ensuite, nous avons mis en place les pré-calculs nécessaires pour la génération de Dashboard et nous avons stocké dans HBase. Enfin, nous avons mis en place le Dashboard permettant de visualiser les informations stockées.

## 2 Solution proposée

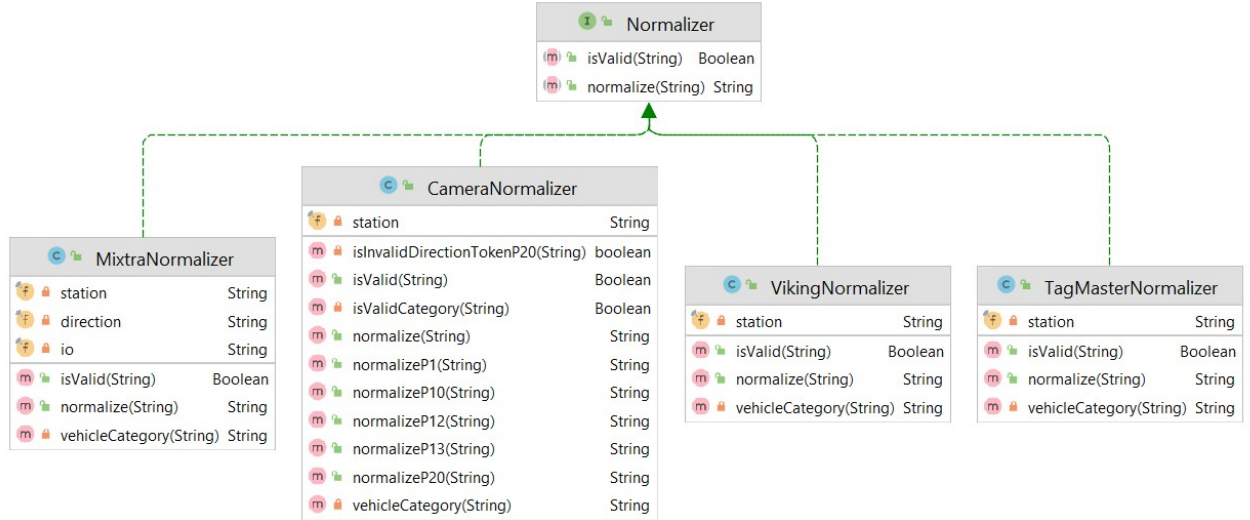
Afin de faire le nettoyage et la normalisation des données et réaliser les pré-calculs nécessaires pour le Dashboard, nous avons fixé le format des enregistrements à la sortie de la phase de normalisation et de nettoyage, ses données sont ensuite utilisées pour entamer la phase d'analyse. la technologie que nous avons décidé d'adopter est Hadoop MapReduce.

### 2.1 MapReduce

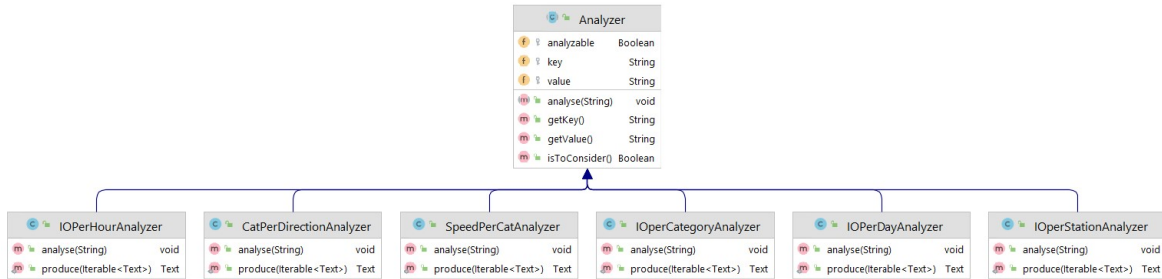
MapReduce est un modèle de programmation pour traiter de grandes quantités de données en parallèle sur un cluster de nœuds. Cela permet de gérer les défis liés au traitement de données volumineuses, tels que la latence, la scalabilité et la tolérance aux pannes. MapReduce se compose de deux phases fondamentales : Map et Reduce. La phase Map transforme les données brutes en une liste intermédiaire, tandis que la phase Reduce combine la liste intermédiaire pour produire les résultats finaux. Hadoop MapReduce est une implémentation open source de MapReduce sur le framework Hadoop pour le stockage et le traitement de données volumineuses.

### 2.2 Architecture du code

Nous avons décidé d'utiliser le patron de conception "Stratégie" pour les deux paquetages de normalisation et d'analyse. C'est un modèle de programmation utilisé pour encapsuler des algorithmes dans des objets séparés. Cela permet de séparer les algorithmes de la logique de l'application, ce qui facilite la maintenance et le changement d'algorithmes. Le patron de conception "Stratégie" permet également de sélectionner à la volée l'algorithme à utiliser en fonction des besoins de l'application. La figure 2 représente l'utilisation du patron de conception **Stratégie** dans le paquetage **Normalizer** qui s'occupe de la normalisation et du nettoyage des données des différents capteurs.

FIGURE 2 – Diagramme de classe pour le package **Normalizer**

Pour le paquetage **Analyzer** qui permet d'effectuer les analyses, nous avons choisi de garder le même schéma de conception. La figure 3 représente l'architecture du paquetage.

FIGURE 3 – Diagramme de classe pour le package **Analyzer**

## 3 Normalisation et nettoyage des données

### 3.1 Format des enregistrement

Le format des enregistrements diffère selon le type du capteur, du coup il a fallu un vrai travail pour étudier le contenu de chaque fichier afin de déterminer un seul format

pour tous ces enregistrements afin de fournir de la donnée propre et bien structurée aux algorithmes qui vont ensuite réaliser les analyses. Nous avons donc choisi de garder le format suivant pour chaque enregistrement.

- **Poste** : il s'agit en effet du numéro du poste du capteur.
- **Horodatage** : Nous avons choisi de fixer de format d'horodatage suivant pour tous les enregistrements de la forme : "YYYY/MM/DD HH:MM:SS"
- **Catégorie** : Nous avons choisi de ne garder que trois catégories de véhicules selon leur volume. 2R, pour les véhicules à deux roues (Vélos, moto, EDPM). VL, pour les véhicules légers. Et PL, pour les véhicules de poids lourds (UT, Bus, ...).
- **Vitesse** : il s'agit de la vitesse en Km/h.
- **IO** : Pour désigner le sens du véhicule enregistré. `in`, pour les véhicules entrants. `out`, pour les véhicules sortants. Et `no_io`, pour les véhicules qui restent à l'intérieur ou à l'extérieur du campus universitaire.

## 3.2 Mapper de normalisation

Pour la phase de nettoyage des données, nous avons défini une classe `DataNormalizer`, qui comporte en plus de la méthode principale `main`, des méthodes de classe pour initialiser un objet `Normalizer` en fonction du type de capteur. On y retrouve également une classe `NormalizationMapper` qui hérite de la classe `Mapper` de Hadoop et surcharge ses méthodes `setup` et `map`. La méthode `setup` initialise l'objet `normalizer` en fonction du type de station et du nom du fichier en cours de traitement. La méthode `map` prend en entrée une ligne de données et utilise l'objet `normalizer` pour vérifier si elle est valide et la normaliser. Les données normalisées sont ensuite écrites en sortie dans un `SequenceFile`.

```
public static class NormalizationMapper extends Mapper<Object, Text, NullWritable, Text> {
    private String fileName;
    private Normalizer normalizer;

    @Override
    protected void setup(Context context){
        InputSplit split = context.getInputSplit();
        if (split instanceof FileSplit) {
            Path filePath = ((FileSplit) split).getPath();
            fileName = filePath.getName();
        }
        Configuration conf = context.getConfiguration();
        String station = conf.get(fileName);
        normalizer = DataNormalizer.initializeNormalizer(station, fileName);
    }

    @Override
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        if (normalizer.isValid(line)) {
            context.write(NullWritable.get(), new Text(normalizer.normalize(line)));
        }
    }
}
```

FIGURE 4 – Code du DataNormalizer

## 4 Analyse des données

### 4.1 Choix des analyses

Afin d'exploiter les données enregistrées par les différents capteurs et réaliser notre Dashboard, nous avons fixé une liste d'analyses à effectuer sur les données après la phase de normalisation. Voici la liste des six analyses que nous avons choisi de présenter sur notre Dashboard.

1. Analyse de la vitesse moyenne des véhicules en fonction de la catégorie de véhicule
2. Analyse de la direction des véhicules en fonction de l'heure de la journée.
3. Analyse de la direction des véhicules en fonction de la journée.
4. Analyse de la distribution des catégories de véhicules en fonction de la direction
5. Analyse de la vitesse moyenne de chaque catégorie
6. Analyse du type de véhicule par direction



## 4.2 Mapper et Reducer des analyses

Pour la phase d'analyse, nous avons une classe **AnalysisMapper** qui étend la classe **Mapper** et implémente les méthodes **setup** et **map**. Dans la méthode **setup**, on initialise un objet **analyzer** en utilisant la configuration de contexte pour obtenir l'identifiant d'analyse. Dans la méthode **map**, on analyse une ligne de données en utilisant l'objet **analyzer**, et si cet objet décide que les données doivent être considérées, alors on les écrit dans le contexte. Nous avons aussi une classe **AnalysisReducer** qui implémente la méthode **reduce** pour agréger les données analysées provenant de la phase de mapping. On utilise la configuration de contexte pour obtenir l'identifiant d'analyse, et on produit une valeur en utilisant la méthode **produce** de la classe **DataAnalyzer**. Enfin, on écrit la paire clé-valeur dans le contexte.

```
public static class AnalysisMapper extends Mapper<Object, Text, Text, Text> {
    private Analyzer analyzer;

    @Override
    protected void setup(Context context) {
        Configuration conf = context.getConfiguration();
        String analysisId = conf.get("analysis_id");
        analyzer = DataAnalyzer.initializeAnalyzer(analysisId);
    }

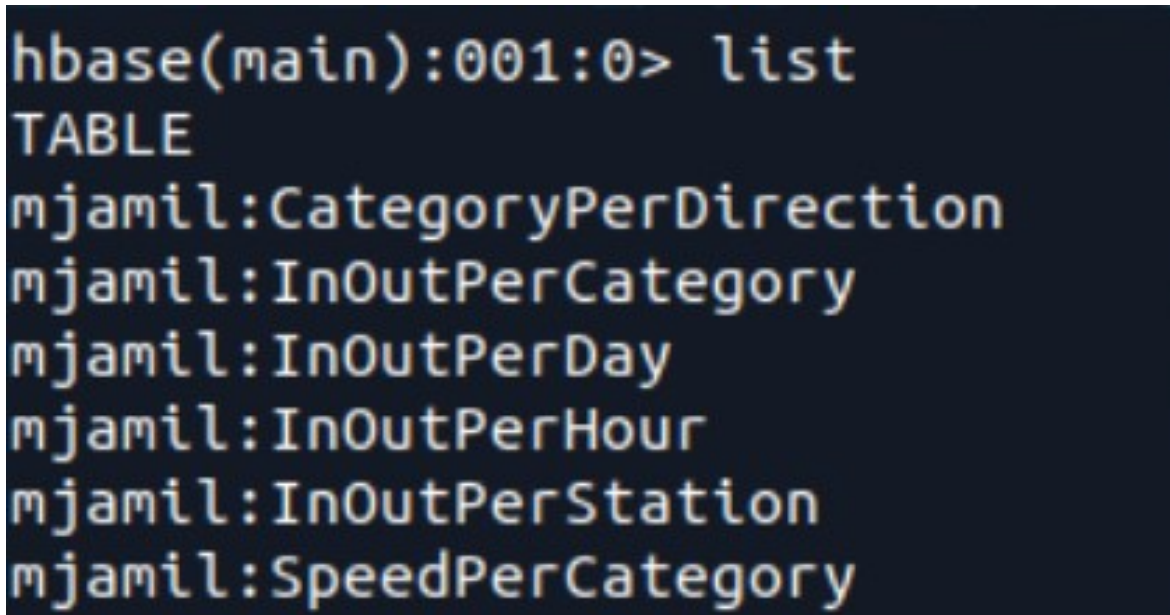
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        analyzer.analyse(line);
        if (analyzer.isToConsider()) {
            context.write(new Text(analyzer.getKey()), new Text(analyzer.getValue()));
        }
    }
}

public static class AnalysisReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        String analysisId = conf.get("analysis_id");
        Text value = DataAnalyzer.produce(analysisId, values);
        context.write(key, value);
    }
}
```

FIGURE 5 – Code du DataAnalyzer

## 5 Écriture dans HBase

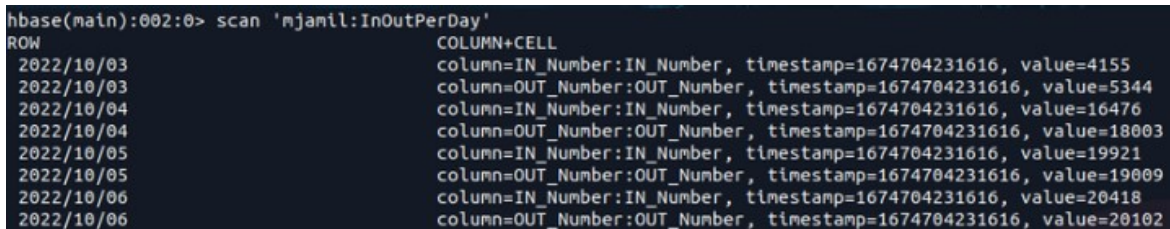
Après avoir normaliser, nettoyer et analyser les données des différent capteurs, sous avons alimenter une base de données HBase avec les résultats des analyses. Pour cela nous avons ajouter une classe `HBaseWriter` qui, pour chaque type d'analyse implémente une classe de réduction qui hérite de `TableReducer` et qui va écrire dans une table HBase dédiée à ce type d'analyse. la figure 6 montre les liste des différents tables des différents analyses.



```
hbase(main):001:0> list
TABLE
mjamil:CategoryPerDirection
mjamil:InOutPerCategory
mjamil:InOutPerDay
mjamil:InOutPerHour
mjamil:InOutPerStation
mjamil:SpeedPerCategory
```

FIGURE 6 – Listes des tables sur HBase

La figure 7 un extrait de la table d'analyse des entrées et des sorties par jour.



```
hbase(main):002:0> scan 'mjamil:InOutPerDay'
ROW                                COLUMN+CELL
2022/10/03                         column=IN_Number:IN_Number, timestamp=1674704231616, value=4155
2022/10/03                         column=OUT_Number:OUT_Number, timestamp=1674704231616, value=5344
2022/10/04                         column=IN_Number:IN_Number, timestamp=1674704231616, value=16476
2022/10/04                         column=OUT_Number:OUT_Number, timestamp=1674704231616, value=18003
2022/10/05                         column=IN_Number:IN_Number, timestamp=1674704231616, value=19921
2022/10/05                         column=OUT_Number:OUT_Number, timestamp=1674704231616, value=19009
2022/10/06                         column=IN_Number:IN_Number, timestamp=1674704231616, value=20418
2022/10/06                         column=OUT_Number:OUT_Number, timestamp=1674704231616, value=20102
```

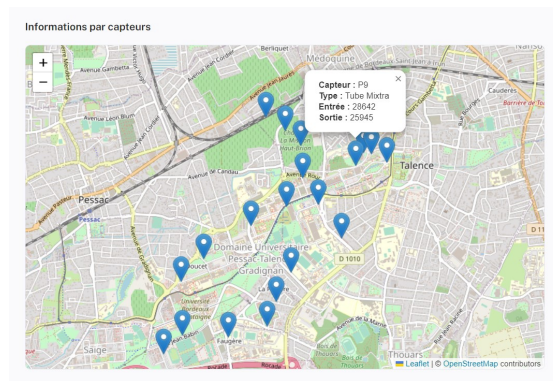
FIGURE 7 – Extrait de la table InOutPerDay

## 6 Réalisation du Dashboard

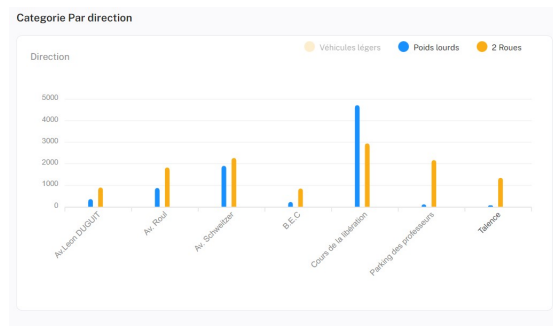
Afin de réaliser notre Dashboard, nous avons utilisé le framework React avec Leaflet pour illustrer la carte. Ainsi nous avons obtenu les résultats suivants :



(a) Capture d'écran du dashboard



(b) Capture d'écran de la carte



(c) Capture d'écran du dashboard

## Conclusion

Pour conclure, ce projet de nettoyage et de normalisation des données collectées par les capteurs installés sur le domaine universitaire a été un succès, malgré quelques difficultés qui nous ont empêché de réaliser un back-end pour notre solution. La solution proposée utilisant Hadoop MapReduce a permis d'effectuer le nettoyage et la normalisation des données de manière efficace et rapide, ainsi que les pré-calculs nécessaires pour le tableau de bord. LDashboard sur lequel nous pouvons visualiser les résultats sur les différents graphes pour une meilleure compréhension de l'état du domaine universitaire.