IMG_257

**Projet Big Data Analytics : Analyse de la Clientèle d'un Concessionnaire Automobile pour la Recommandation de Modèles de Véhicules**

**Groupe TPA 4**

RAFAMANTANANTSOA Rotsy Vonimanitra

RAMAROSON Sandy Princia

RAMIANDRISOA Rantonantenaina Steve

RATSIRARSON Joharisoa

**Avril 2024 – 9 Juin 2024**

**Enseignants Encadreurs :**

Gabriel MOPOLO

Sergio SIMONIAN

Nicolas PASQUIER

**Résumé du projet**

Ce projet vise à mettre en place un service pour un concessionnaire automobile, permettant ainsi une meilleure compréhension des besoins des clients et la proposition des véhicules les plus adaptés à leurs attentes. En outre, cela favorisera leur essor dans l'industrie du commerce automobile.

C’est un projet de grande envergure incluant le traitement de grandes quantités de données avec la mise en place d’un DataLake et l’analyse des données y figurant.

À travers des méthodes, telles que l'accès aux sources de données via Access Driver et Data Extractor, la mise en place d'un DataLake sur HiveQL, l'utilisation de Hadoop Map Reduce pour le nettoyage et l'analyse des données avec Python, nous garantirons des résultats précis et fiables.

Les principaux résultats de ce travail incluront une architecture du projet optimisée, des données nettoyées et cohérentes, ainsi que des analyses prédictives pour identifier les véhicules les plus appropriés pour chaque client

En conclusion, ce projet permettra aux concessionnaires automobiles de proposer rapidement des véhicules adaptés aux clients qui se présentent en concession, renforçant ainsi leur compétitivité sur le marché.  
  
**Mots clés :** Big Data, analyse de données, Data Lake, ELT, Hadoop Map Reduce, Python, Access Driver, HiveQL, Apache NiFi.

**Abstract**

The aim of this project is to set up a service for a car dealership, enabling it to better understand customers' needs and suggest the most suitable vehicles for them. It will also help them to grow in the automotive trade.

This is a large-scale project involving the processing of large quantities of data, with the setting up of a DataLake and the analysis of the data contained therein.

Through methods such as accessing data sources via Access Driver and Data Extractor, setting up a DataLake on HiveQL, using Hadoop Map Reduce for data cleansing and analyzing the data with Python, we will guarantee accurate and reliable results.

The main results of this work will include an optimized project architecture, cleansed and consistent data, and predictive analytics to identify the most appropriate vehicles for each customer.

In conclusion, this project will enable car dealers to rapidly offer suitable vehicles to customers who come to the dealership, thus strengthening their competitiveness on the market.

**Keywords:** Big Data, Data analysis, Data Lake, ELT, Hadoop Map Reduce, Python, Access Driver, HiveQL, Apache NiFi.

**Liste des figures**

[Figure 1 : Architecture DBA/DL 11](#_Toc168831591)

[Figure 2 : Résultat du processor GetMongo 19](#_Toc168831592)

[Figure 3 : Résultat des processors ReplaceText 19](#_Toc168831593)

[Figure 4 : Liste des fichiers temporaires HDFS 19](#_Toc168831594)

[Figure 5 : Résultat du processor ReplaceText 19](#_Toc168831595)

[Figure 6 : Log d'insertion des données dans Hive 20](#_Toc168831596)

[Figure 7 : Architecture du processus d'alimentation avec NiFi 20](#_Toc168831597)

[Figure 8 : Chargement des données dans data sources 27](#_Toc168831598)

[Figure 9 : Map reduce et chargement des données virtuelles dans hiveQL. 28](#_Toc168831599)

[Figure 10 : Courbe d'identification du nombre de Cluster Optimal (Elbow Method) 30](#_Toc168831600)

[Figure 11 : Aperçu de la table de résultats dans Mongo DB 35](#_Toc168831601)

**Liste des tableaux**

[Tableau 1 : Liste des valeurs transformées 31](#_Toc168831635)

[Tableau 2 : Liste des modèles et des paramètres correspondants 36](#_Toc168831636)

**Liste des acronymes**

BDA/DL: Big Data Analytics/Data Lake

ELT: Extract Load Transform

HDFS: Hadoop Distributed File System

KNN: K-Nearest Neighbors

Mongo DB: Mongo Database

SSE: Sum of Squared Errors

**Table des matières**

Table des matières

[1. Introduction générale 7](#_Toc168836957)

[2. Présentation du projet 8](#_Toc168836958)

[3. Répartition du travail en membre du groupe 9](#_Toc168836959)

[4. Architecture du Data Lake 9](#_Toc168836960)

[5. Construction du data lake par étape 13](#_Toc168836961)

[1. Démarrage de la machine virtuelle et configuration 13](#_Toc168836962)

[2. Mise en place des sources de données 14](#_Toc168836963)

[a. Source Hadoop Distributed File System (HDFS) 14](#_Toc168836964)

[b. Source Oracle NoSQL 14](#_Toc168836965)

[c. Source MongoDB 15](#_Toc168836966)

[3. Mise en place du frontal de données sur HiveQL Data Lake 16](#_Toc168836967)

[a. Création des données virtuelles Hive 16](#_Toc168836968)

[b. Création des données physiques Hive 17](#_Toc168836969)

[c. Création des données de modèle d’analyse 21](#_Toc168836970)

[4. Exploitation des données sur Python 22](#_Toc168836971)

[a. Lancement de Jupyter Notebook 22](#_Toc168836972)

[6. Hadoop Map Reduce 22](#_Toc168836973)

[7. Automatisation des scripts 26](#_Toc168836974)

[8. Analyse de données avec Python 29](#_Toc168836975)

[9. Conclusion générale 37](#_Toc168836976)

[10. Références et Bibliographie 38](#_Toc168836977)

[11. Annexes 38](#_Toc168836978)

[11.1 Vidéo de présentation 38](#_Toc168836979)

[11.2 Dossier contenant les scripts et programmes de construction du lac de données 38](#_Toc168836980)

[11.3 Dossier contenant les scripts et programmes Hadoop Map Reduce 39](#_Toc168836981)

[11.4 Dossier contenant les scripts et programmes d’analyse de données 39](#_Toc168836982)

# Introduction générale

"Vous cherchez à optimiser les ventes de votre concession automobile et à identifier de nouvelles opportunités de croissance ? Alors notre rapport analytique est votre meilleur allié !"

Ce projet vise à appliquer notre maîtrise des techniques du Big Data, du concept Map Reduce, de l’analyse des données et machine learning pour répondre au mieux les besoins des clients des concessionnaires automobile.

Nous aurons alors pour mission de gérer les données offertes par un concessionnaire avec l’architecture Big Data Analytics/Data Lake en les centralisant et en utilisant le concept de Map Reduce avec Hadoop pour effectuer les tâches complexes, et d’effectuer l’analyse des données en préparant les données en entrées des outils d’analyses sur Python avec SQL.

Le plan de notre rapport est structuré comme suit :

* Présentation du projet : en quoi est-il innovant et quelles sont les défis
* Répartition des tâches : présentation de la distribution des tâches entre les membres du groupe
* Architecture du Data Lake : construction et description du Data Lake
* Construction du Data Lake par étape : détails depuis la mise en place des sources de données jusqu’à l’exploitation des données dans les outils d’analyses
* Hadoop Map Reduce : explication de l’utilité et utilisation du processus
* Automatisation des scripts : vous n’aurez qu’à lancer un seul script et toute votre architecture sera prête
* Analyse de données avec Python : clustering des données, entrainement et prédiction
* Conclusion : bilan du projet en général et évaluation par rapport aux objectif fixés

# Présentation du projet

Dans un secteur automobile en constante évolution, où les attentes des clients sont de plus en plus exigeantes, l'analyse de données émerge comme un outil indispensable pour les concessionnaires. Notre projet se distingue par son approche novatrice qui combine les dernières avancées en Big Data et en Data Analytics & Machine Learning pour répondre aux besoins spécifiques de nos clients concessionnaires. En développant des solutions sur mesure, nous ouvrons la voie à une transformation significative de leur approche commerciale, les positionnant à la pointe de l'industrie.

Dans un marché automobile compétitif, répondre aux besoins des clients rapidement et de manière personnalisée est essentiel pour les concessionnaires. Notre projet se situe à un moment clé où l'utilisation intelligente des données devient essentielle pour prendre des décisions stratégiques. Cela offre aux concessionnaires une chance de se démarquer et de réussir dans un environnement en constante évolution.

La réussite de ce projet est d’une importance capitale pour nos clients concessionnaires, car elle détermine leur capacité à rester pertinents et compétitifs dans le marché. Les principaux enjeux résident dans la capacité à traiter efficacement de grandes quantités de données tout en assurant leur qualité et leur pertinence.

De plus, la mise en œuvre comporte des risques, notamment en termes de complexité technique.

Tout d'abord, l'accès aux données à partir de différentes sources est un défi majeur, nécessitant l'utilisation de Access Driver et Data Extractor pour le chargement et la transformation des données (ELT).  
Ensuite, la centralisation de ces données dans le Data Lake, avec HiveQL comme frontal de données, est essentielle pour assurer leur accessibilité et leur intégrité.  
Une fois les données centralisées, le traitement massif avec Hadoop MapReduce garantit leur nettoyage et leur préparation pour l'analyse.  
Enfin, l'utilisation de Python pour l'analyse avancée et la prédiction marque la dernière étape de ce processus complexe.

Malgré ces défis, une approche méthodique et une répartition efficace des tâches sont essentielles pour garantir le succès de ce projet et la satisfaction de nos clients concessionnaires.

# Répartition du travail en membre du groupe

Voici les membres du groupe :

**13** - RAFAMANTANANTSOA Rotsy Vonimanitra

**29** – RAMAROSON Sandy Princia

**31** – RAMIANDRISOA Rantonantenaina Steve

**50** – RATSIRARSON Joharisoa

Nous allons voir ci-dessous la répartition du projet entre les membres du groupe :

* Architecture du data lake : Tous les membres
* Mise en place du data lake :
  + Utilitaires et sources de données :
    - HDFS (Catalogues, CO2) : **29**
    - Programme Python - Oracle NoSQL (Marketing) : **31**
    - Mongo import - MongoDB (Clients, Immatriculations) : **13**
  + Data Access Drivers :
    - HDFS Access Driver (Catalogues) : **29**
    - Oracle NoSQL Access Driver (Marketing): **31**
  + Data Extractor - ELT :
    - NiFi (Clients, Immatriculations) : **13** - **29**
  + Frontal de données HiveQL Data Lake :
    - Données virtuelles (Catalogues, Marketing) : **29** - **31**
    - Données physiques (Clients, Immatriculations) : **13**
    - Données de modèles d’analyses (vue clients\_immatriculations) : **13**
  + Hadoop Map Reduce (Catalogues, CO2): **31** - **50**
  + Utilitaire et outils d’analyse :
    - Clustering : **50** -**31**
    - Classifieur : **50**
    - Modèle de prédiction : **50**
  + Result database : **50**
* Automatisation des scripts (chargement de données dans data source, map reduce, chargement données virtuelles dans HiveQL) : **29**
* Rédaction du TPA : Tous les membres
* Mise en forme du rapport et intégration des différentes sections rédigées : **13**
* Vidéo de présentation du projet : Tous les membres

# Architecture du Data Lake

Notre architecture de Data Lake intègre plusieurs composants essentiels et utilise Hive Data Lake comme frontal de données.

Voici une description détaillée de chaque composant :

1. **Fichiers de Données**

* CO2.csv : Ce fichier continent les données sur les émissions de CO2
* Immatriculations.csv : Ce fichier contient les informations sur les immatriculations effectuées cette année
* Catalogue.csv : Ce fichier contient une liste des catalogues de véhicules disponibles chez le concessionnaire
* Clients\_3.csv et Clients\_16.csv : Ces fichiers contiennent les achats effectués par les clients numéros 3 et 16 cette année
* Marketing.csv : Ce fichier contient des groupes de clients sélectionnés par le service marketing

1. **Utilitaires**

* Utilitaire HDFS : Les fichiers CO2.csv et Catalogues.csv sont importés dans le système de fichiers distribué Hadoop (HDFS) à l'aide d'un utilitaire HDFS.
* Mongo import : Les fichiers Clients\_3.csv, Clients\_16.csv et Immatriculations.csv sont importés dans MongoDB via l’utilitaire mongo import
* Python Program Extractor : Le fichier marketing.csv est importé dans Oracle NoSQL via un programme d’extraction en Python

1. **Sources de données**

* HDFS : Stocke les données de CO2.csv, Catalogue.csv et Catalogue\_CO2.csv (explication ci-dessous dans la partie Hadoop Map Reduce).
* Oracle NoSQL : Stocke les données de Marketing.csv
* MongoDB : Stocke les données des fichiers Clients\_3.csv, Clients\_16.csv et Immatriculations.csv

1. **Hadoop Map Reduce**

* HDFS : Ecriture d’un programme map reduce avec Hadoop pour adapter le fichier CO2.csv et ajouter des informations aux données de la table Catalogue. On obtient une nouvelle source combinant Catalogue et CO2 qu’on a appelé Catalogue\_CO2 et qu’on a remis dans HDFS

1. **Alimentation du Data Lake**

* HDFS Acces driver : Pour l’accès direct aux données depuis la source Catalogue\_CO2
* Oracle NoSQL Access driver : Pour l’accès direct aux données depuis la source Marketing
* NiFi (Extract Load and Transform) : Pour l’alimentation des données physiques du Data Lake, ces données sont les données sur les Clients et les Immatriculations

1. **Les données du Data Lake HiveQL**

* Données virtuelles : Contient la table externe HDFS dont Catalogue\_CO2 et la table externe Oracle NoSQL Marketing. Les données sont accessibles via access drivers.
* Données physiques : Contient les managed tables (tables internes) comme Clients et Immatriculations. Les données sont chargées avec un extracteur de données (NiFi)
* Données des modèles d’analyses : Données qui font une jointure entre les clients et immatriculations sous forme de vue : clients\_immatriculations

1. **Outils d’analyse**

* PyHive : utilisé pour connecter les modèles d’analyses du Data Lake HiveQL à des outils d’analyse Python
* Python : Utilisé pour l’analyse des données.

1. **Base des Résultats**

* MongoDB : Les résultats de l’analyse des données sont visualisés à l’aide de Python et les données résultats des prédictions sont stockés dans une nouvelle table MongoDB

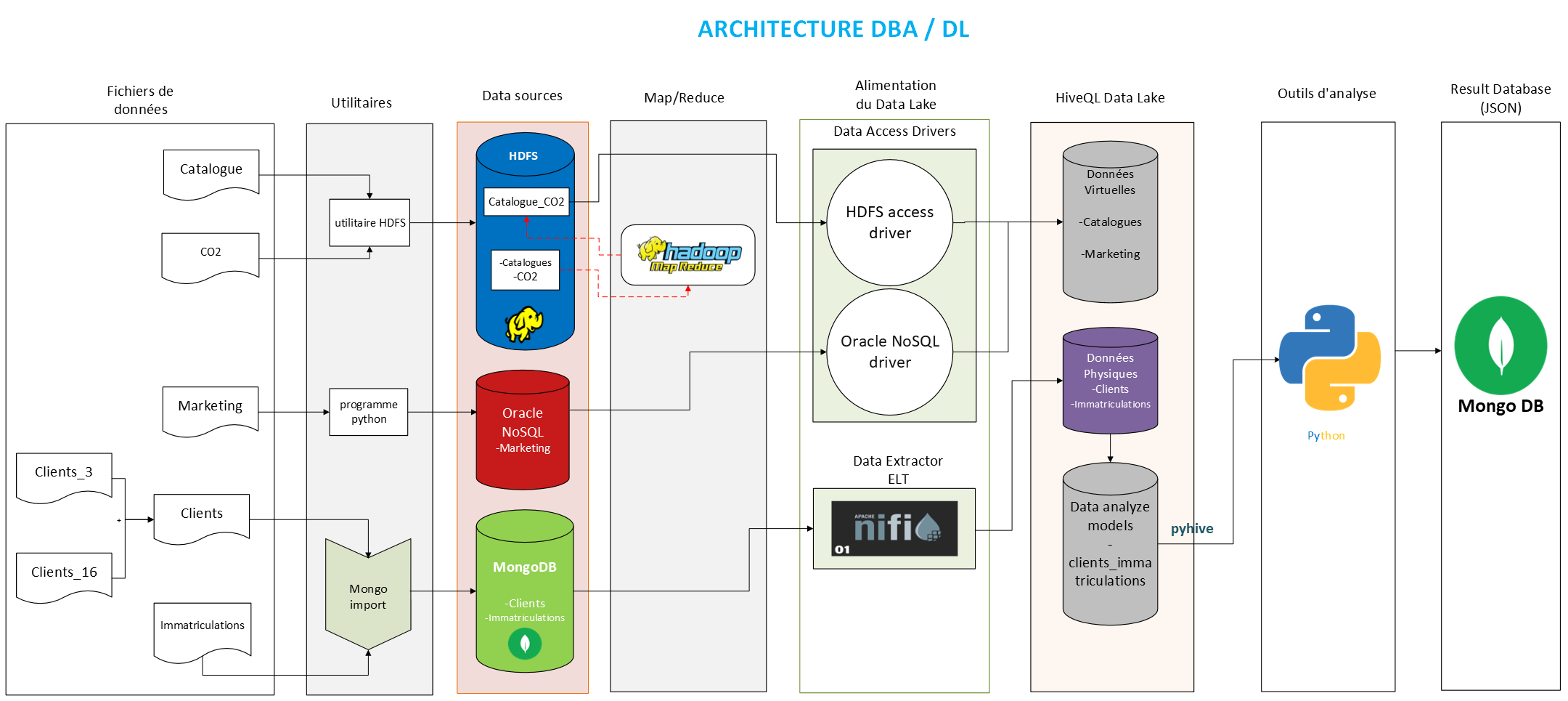


Figure 1 : Architecture DBA/DL

# Construction du data lake par étape

Nous présenterons ici les étapes de la mise en œuvre de l’architecture du data lake de la collecte des données jusqu’à leur exploitation dans les outils d’analyse avec Python.

L’ensemble des scripts et étapes montrés ci-dessous sont exécutables par une simple exécution du fichier **script-tpa.ksh.**

## Démarrage de la machine virtuelle et configuration

* Lancer l’invite de commande

cmd

* Naviguer vers le dossier où se trouve votre machine Vagrant

cd lienVersVotreMachineVirtuelle\INSTALL\_MV\_BIGDATA\_BOX>

* Arrêter si nécessaire puis démarrer la machine

vagrant halt

vagrant up

* Connectez-vous à la machine Vagrant

vagrant ssh

* Démarrer Hadoop HDFS

if ! jps | grep -q 'NameNode'; then

    start-dfs.sh

else

    echo "HDFS is already running."

fi

* Démarrer Hadoop YARN

if ! jps | grep -q 'ResourceManager'; then

    start-yarn.sh

else

    echo "YARN is already running."

fi

* Installer la librairie ‘pandas’, utile pour les programmes Python

if ! pip show pandas > /dev/null; then

    pip install pandas

else

    echo "pandas is already installed."

fi

* Démarrer le serveur Hadoop Hive

nohup hive --service metastore > /dev/null &

nohup hiveserver2 > /dev/null &

* *Copier* le dossier *Groupe\_TPA\_4* a la racine de votre machine vagrant

## Mise en place des sources de données

### Source Hadoop Distributed File System (HDFS)

* Fichiers : Catalogue.csv, CO2.csv
* Procédure :
  + Définition du répertoire de travail

SCRIPT\_DIR="/vagrant/Groupe\_TPA\_4/mapreduceCo2"

cd ${SCRIPT\_DIR}

* + Nettoyage des données de CO2 avec un programme Python

python3 cleanCO2.py

* + Importation des données de CO2 et Catalogue dans HDFS

hadoop fs -put modified\_CO2.csv

hadoop fs -put Catalogue.csv

* + Importation des données de Catalogue\_CO2, résultat du map reduce sur CO2, dans HDFS (Détails dans l’étape [6.Hadoop Map Reduce](#_Hadoop_Map_Reduce))

hadoop fs -mkdir -p /catalogueCO2

hadoop fs -put Catalogue\_CO2.csv /catalogueCO2

* + Créer des dossiers pour stocker les fichiers temporaires sur les clients et immatriculations. Ces fichiers seront utiles pour l’alimentation des tables internes de Hive depuis NiFi.

hdfs dfs -mkdir /clients

hdfs dfs -mkdir /immatriculations

### Source Oracle NoSQL

* Fichier : Marketing.csv
* Procédure :
  + Démarrage du serveur Oracle NoSQL s’il n’est pas encore démarré

if ! jps | grep -q 'kvstore'; then

    nohup java -Xmx64m -Xms64m -jar $KVHOME/lib/kvstore.jar kvlite -secure-config disable -root $KVROOT &

else

    echo "Oracle NoSQL server is already running."

fi

* + Définition du répertoire de travail

SCRIPT\_DIR="/vagrant/Groupe\_TPA\_4"

cd ${SCRIPT\_DIR}

* + Création d’un fichier sql pour l’insertion des données avec un programme Python > génération d’un fichier scriptInsert.sql

python3 ${SCRIPT\_DIR}/scriptInsertMarketing/scriptInsertMarketing.py

* + Lancement du client SQL de Oracle NoSQL

java -jar $KVHOME/lib/sql.jar -helper-hosts localhost:5000 -store kvstore <<EOF

* + Création de la table marketing et insertion des données dans Oracle NoSQL

load -file ${SCRIPT\_DIR}/scriptInsertMarketing/scriptInsert.sql

exit

EOF

### Source MongoDB

* Fichiers : Clients\_3.csv, Clients\_16.csv, Immatriculations.csv
* Procédure :
  + Définition du répertoire de travail

SCRIPT\_DIR="/vagrant/Groupe\_TPA\_4"

cd ${SCRIPT\_DIR}

* + Création des dossiers clients et immatriculations ou on traitera les fichiers csv

mkdir clients

mkdir immatriculations

* + Fusion des fichiers Clients\_3.csv et Clients\_16.csv (traitement avec awk)

cat M2\_DMA\_Clients\_4/Clients\_3.csv <(tail -n +2 M2\_DMA\_Clients\_17/Clients\_16.csv) > clients/ClientsWithoutId.csv

* + Nettoyage des données clients.csv (traitement avec awk)

cd clients

# modifier le fichier csv pour y mettre une colonne id

awk -F, 'BEGIN {OFS=FS; i="id"} {print i, $0; i++}' ClientsWithoutId.csv > ClientsToManage.csv

# renommer la colonne 2eme voiture en deuxiemeVoiture

awk 'BEGIN {FS=OFS=","} FNR==1 {$7="deuxiemeVoiture"} {print}'  ClientsToManage.csv > temp.csv && mv temp.csv ClientsToManage.csv

# mettre les colonnes numeriques avec valeurs non numeriques en 0 par defaut

awk -F',' 'BEGIN {OFS=FS} NR>1 {for (i=1; i<=NF; i++) {if (i==2 || i==4 || i==6) {if ($i=="" || $i=="?" || $i !~ /^[0-9]+$/) $i="0"}}}1' ClientsToManage.csv > temp.csv && mv temp.csv ClientsToManage.csv

#uniformiser la colonne sexe, M pour les valeurs commençant par M et les valeurs non définis (N/D, ? , "",...) et F pour les autres

awk -F',' 'BEGIN {OFS=FS} NR>1 {if ($3 ~ /^F/) $3="F"; else if ($3 ~ /^M/ || $3 == "Homme") $3="M"; else $3="NULL"} 1' ClientsToManage.csv > temp.csv && mv temp.csv ClientsToManage.csv

# mettre la colonne deuxiemeVoiture en false par defaut

awk -F',' 'BEGIN {OFS=FS} NR>1 {if ($7 != "true" && $7 != "false") $7="false"} 1' ClientsToManage.csv > Clients.csv

* + Nettoyage des données immatriculations.csv (traitement avec awk)

cd ${SCRIPT\_DIR}

# modifier le fichier csv pour y mettre une colonne id

awk -F, 'BEGIN {OFS=FS; i="id"} {print i, $0; i++}' M2\_DMA\_Immatriculations/Immatriculations.csv > immatriculations/ImmatriculationsToManage.csv

cd immatriculations

# mettre les colonnes numeriques avec valeurs non numeriques en 0 par defaut

awk -F',' 'BEGIN {OFS=FS} NR>1 {for (i=1; i<=NF; i++) {if (i==5 || i==7 || i==8 || i==11) {if ($i=="" || $i=="?" || $i !~ /^[0-9]+$/) $i="0"}}}1' ImmatriculationsToManage.csv > temp.csv && mv temp.csv ImmatriculationsToManage.csv

# mettre la colonne occasion en false par defaut

awk -F',' 'BEGIN {OFS=FS} NR>1 {if ($10 != "true" && $10 != "false") $10="false"} 1' ImmatriculationsToManage.csv > Immatriculations.csv

* + Création de la table clients et importation des données dans MongoDB

cd ${SCRIPT\_DIR}/clients

mongoimport --db Concessionaire --collection clients --type csv --file Clients.csv --headerline

* + Création de la table immatriculations et importation des données dans MongoDB

cd ${SCRIPT\_DIR}/immatriculations

mongoimport --db Concessionaire --collection immatriculations --type csv --file Immatriculations.csv --headerline

## Mise en place du frontal de données sur HiveQL Data Lake

### Création des données virtuelles Hive

* + - Table externe CATALOGUE\_CO2\_HDFS\_EXT

beeline -u jdbc:hive2://localhost:10000 -n oracle -p welcome1 -e "

DROP TABLE IF EXISTS CATALOGUE\_CO2\_HDFS\_EXT;

CREATE EXTERNAL TABLE CATALOGUE\_CO2\_HDFS\_EXT (

marque STRING, nom STRING, puissance INT, longueur STRING, nbPlaces INT,

nbPortes INT,

couleur STRING,

occasion boolean,

prix INT,

BonusMalus INT,

RejetsCO2 INT,

CoutEnergie INT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ','

STORED AS TEXTFILE LOCATION '/catalogueCO2';

SELECT COUNT(\*) FROM CATALOGUE\_CO2\_HDFS\_EXT;

"

* + - Table externe MARKETING\_ONS\_EXT

beeline -u jdbc:hive2://localhost:10000 -n oracle -p welcome1 -e "

DROP TABLE IF EXISTS Marketing\_ONS\_EXT;

CREATE EXTERNAL TABLE Marketing\_ONS\_EXT  (

id int,

age int,

sexe string,

situationFamiliale string,

nbEnfantsAcharge int,

deuxieme\_voiture boolean

)

STORED BY 'oracle.kv.hadoop.hive.table.TableStorageHandler'

TBLPROPERTIES (

'oracle.kv.kvstore' = 'kvstore',

'oracle.kv.hosts' = 'localhost:5000',

'oracle.kv.hadoop.hosts' = 'localhost/127.0.0.1',

'oracle.kv.tableName' = 'marketing');

"

### Création des données physiques Hive

* + - Table interne CLIENTS dans la database CONCESSIONAIRE

beeline -u jdbc:hive2://localhost:10000 -n oracle -p welcome1 -e "

DROP TABLE IF EXISTS CONCESSIONAIRE.CLIENTS;

CREATE TABLE CONCESSIONAIRE.CLIENTS (

ID int,

AGE int,

SEXE string,

TAUX float,

SITUATIONFAMILIALE string,

NBENFANTSACHARGE int,

DEUXIEMEVOITURE boolean,

IMMATRICULATION string

)

ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe';

"

* + - Table interne IMMATRICULATIONS dans la database CONCESSIONAIRE

beeline -u jdbc:hive2://localhost:10000 -n oracle -p welcome1 -e "

DROP TABLE IF EXISTS CONCESSIONAIRE.IMMATRICULATIONS;

CREATE TABLE CONCESSIONAIRE.IMMATRICULATIONS (

ID int,

IMMATRICULATION string,

MARQUE string,

NOM string,

PUISSANCE int,

LONGUEUR string,

NBPLACES int,

NBPORTES int,

COULEUR string,

OCCASION boolean,

PRIX int

)

ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe';

"

* + - Alimentation des tables internes via Apache NiFi (ELT)

***Apache NiFi*** est une plateforme open-source permettant la gestion et l'automatisation des flux de données en temps réel. Son rôle en tant qu'ELT (Extract, Load, Transform) consiste à extraire des données depuis différentes sources, les charger dans un système de stockage, puis les transformer selon les besoins.

Dans notre architecture, il est utilisé pour charger les données des clients et immatriculations depuis MongoDB et les insérer dans les tables internes de Hive.

Il y a eu des contraintes sur les accès aux serveurs MongoDB et Hadoop Hive depuis la plateforme NiFi. Nous avons alors décidé de l’installer directement dans la machine virtuelle.  
NiFi utilise une interface graphique accessible via navigateur web pour gérer les processus et il fallait faire un port forwarding[[1]](#footnote-1)pour pouvoir y accéder. La plateforme est alors accessible via l’url <http://127.0.0.1:8444/nifi> (8443 vers 8444).

L’installation de NiFi est assez longue. Nous avons alors exporté nos données des tables internes de Hive pour que vous pouviez directement les importer dans vos tables Hive locales et y accéder sans passer par l’installation. Cela est également écrit dans le script d’automatisation script-tpa.ksh cité auparavant.

Voici les étapes suivis pour **l’installation du logiciel Apache NiFi sur votre machine virtuelle**, il n’y aura ensuite qu’à lancer le processus avec le bouton « Start » et l’alimentation des données physiques sur Hive se fera toute seule.

* Copier le fichier d’installation de NiFi depuis le lien drive suivant et collez-le a la racine de votre machine Vagrant : <https://drive.google.com/file/d/1OGgrQQ9FJ7FVWp1Out6PPdOO5hszGOi9/view?usp=drive_link>
* Dézipper le fichier, vous devez avoir une architecture de fichier comme ceci : /vagrant/nifi-1.19.1
* Activer l’accessibilité du port 8443, utilisé par défaut par NiFi. Dans vagrant ssh, exécuter :

sudo su

iptables -A INPUT -p tcp --dport 8443 -j ACCEPT

* Vous pouvez vérifier l’état du port en le voyant dans les résultats de la commande suivante :

iptables -L

* Créer une configuration d'alias personnalisée pour votre profil de machine virtuelle Vagrant.

Sous Windows, rendez-vous dans le dossier Your-username/.ssh et recherchez le fichier *config*, sinon créez le (sans extension).

Ajoutez les lignes suivantes pour créer l'alias oracle-21c-vagrant

Host oracle-21c-vagrant

    HostName 127.0.0.1

    User vagrant

    Port 2222

    UserKnownHostsFile /dev/null

    StrictHostKeyChecking no

    PasswordAuthentication no

    IdentityFile Your-username/.vagrant.d/boxes/package.box/0/virtualbox/vagrant\_private\_key

    IdentitiesOnly yes

    LogLevel FATAL

    PubkeyAcceptedKeyTypes +ssh-rsa

    HostKeyAlgorithms +ssh-rsa

* Utilisez le port forwarding pour accéder à NiFi à partir de l'extérieur

ssh -L 8444:127.0.0.1:8443 oracle-21c-vagrant

* Vous pouvez maintenant démarrer NiFi

cd /vagrant/nifi-1.19.1

./bin/nifi.sh start

* Attendez le démarrage de NiFi, vous pouvez vérifier les logs en attendant qu’une ligne affiche <http://127.0.0.1:8444/nifi>. Le démarrage peut être assez long selon votre machine (5 à 10 minutes).

tail -f -n 100 ./logs/nifi-app.log

* Une fois le logiciel démarré et l’interface s’affichant sur le navigateur, faite un clic droit sur votre souris et cliquez sur le bouton « Start ».
* Vérifier que les données ont bien été importés dans Hive

beeline -u jdbc:hive2://localhost:10000 -n oracle -p welcome1 -e "

select \* from CONCESSIONAIRE.CLIENTS;

select \* from CONCESSIONAIRE.IMMATRICULATIONS;

"

**Architecture du processus sur NiFi :**

* GetMongo : Accède aux données des collections MongoDB et les charges dans le Process Group

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Figure 2 : Résultat du processor GetMongo

* Les 2 premiers ReplaceText : Formattent les json reçues depuis GetMongo en une ligne.



Figure 3 : Résultat des processors ReplaceText

* PutHDFS : Met chaque groupe de données dans un fichier HDFS temporaire. Un fichier HDFS correspond à une liste de json qui va être passé à un script d’insertion dans Hive.

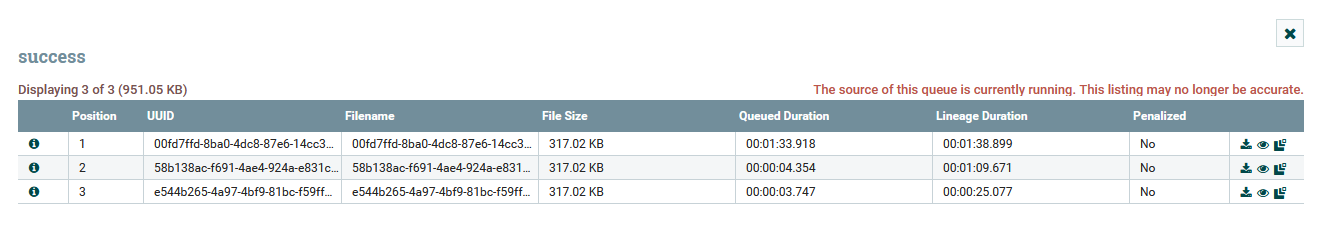


Figure 4 : Liste des fichiers temporaires HDFS

* ReplaceText : écrit le script d'insertion des fichiers temporaires HDFS dans la table Hive



Figure 5 : Résultat du processor ReplaceText

* PutHiveQL : exécute les scripts d’alimentation des tables internes clients et immatriculations dans Hive.

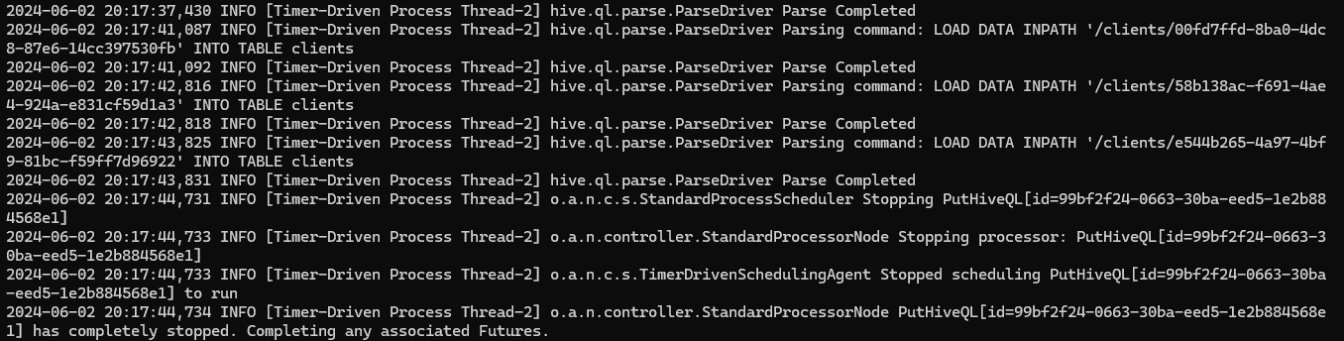


Figure 6 : Log d'insertion des données dans Hive

* LogMessage : affiche les messages d'erreurs et de succès dans le fichier log de NiFi pour le suivi des process.

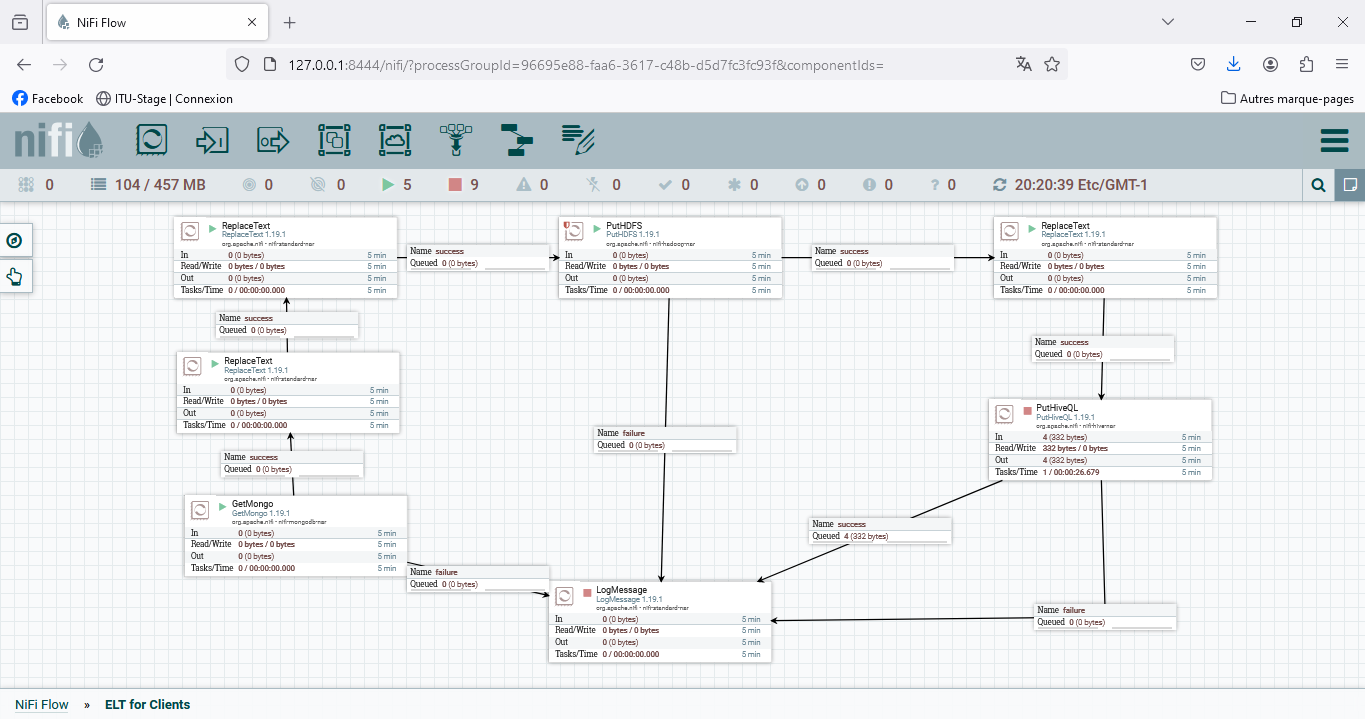


Figure 7 : Architecture du processus d'alimentation avec NiFi

### Création des données de modèle d’analyse

* + - Table de représentation de vue SQL CLIENTS\_IMMATRICULATIONS dans la database CONCESSIONAIRE

beeline -u jdbc:hive2://localhost:10000 -n oracle -p welcome1 -e "

CREATE VIEW CONCESSIONAIRE.CLIENTS\_IMMATRICULATIONS AS

select c.id,c.age, c.sexe, c.taux, c.situationfamiliale, c.nbenfantsacharge,c.deuxiemevoiture, c.immatriculation,i.marque, i.nom,i.puissance,  i.longueur, i.nbplaces, i.nbportes, i.couleur,i.occasion, i.prix

from concessionaire.clients c join concessionaire.immatriculations i on i.immatriculation = c.immatriculation;

"

## Exploitation des données sur Python

Pour cette partie, on a lancé Jupyter Notebook sur notre machine et fait les traitements d’analyse de données, qui seront détaillés plus tard dans la section « [Analyse de données avec Python](#_Analyse_de_données)».

### Lancement de Jupyter Notebook

cd /vagrant/Groupe\_TPA\_4

jupyter notebook --ip=0.0.0.0

On ouvre ensuite un navigateur et va vers le lien de la forme http://127.0.0.1:8888/?token=a9193773ff2cc64bbacfc6df94cd1005c965d09cff17be793 obtenu via la commande précédente.

# Hadoop Map Reduce

On décrit ci-dessous les étapes de la mise en œuvre de l’activité autour de Hadoop Map Reduce :

1. **Mapper Script**

Le script Mapper (map.py) se charge de lire les fichiers CSV et de produire des paires clé-valeur pour chaque ligne du Catalogue.csv en y associant les informations du modified\_CO2.csv correspondantes ou des moyennes calculées si aucune correspondance n'est trouvée. Voici les étapes détaillées :

* + **Lecture des données modifiées de CO2 :**

Le fichier modified\_CO2.csv est lu et ses données sont stockées dans la liste *modified\_co2\_data.*

total\_bonus\_malus\_match = 0

total\_rejets\_co2\_match = 0

total\_cout\_energie\_match = 0

count\_match = 0

output\_lines = []

* + **Premier passage : collecte des statistiques de correspondance :**
    - Le fichier Catalogue.csv est lu ligne par ligne.
    - Si une ligne contient le mot "marque" (en-tête), elle est imprimée avec de nouvelles colonnes ajoutées ("Bonus / Malus", "Rejets CO2 g/km", "Cout energie").
    - Pour chaque ligne de Catalogue.csv, le script recherche les correspondances dans modified\_co2\_data en comparant la marque.
    - Si une correspondance est trouvée, les valeurs de bonus/malus, rejets de CO2, et coût d'énergie sont ajoutées aux accumulateurs.

modified\_co2\_data = []

with open('modified\_CO2.csv', 'r', encoding='utf-8') as modified\_file:

for line in modified\_file:

data = line.strip().split(',')

modified\_co2\_data.append(data)

* + **Calcul des moyennes :**
    - Si des correspondances ont été trouvées (count\_match > 0), les moyennes pour bonus/malus, rejets de CO2, et coût d'énergie sont calculées.
    - Sinon, les moyennes sont mises à zéro.

with open('Catalogue.csv', 'r', encoding='utf-8') as catalogue\_file:

for lineCatalogue in catalogue\_file:

data = lineCatalogue.strip().split(',')

marque = data[0]

if marque.lower() == "marque":

print(f"{lineCatalogue.strip()},\tBonus / Malus,Rejets CO2 g/km,Coutenergie")

continue

for data1 in modified\_co2\_data:

marque\_modele = data1[1]

bonus\_malus = data1[2]

rejets\_co2 = data1[3]

cout\_energie = data1[4]

if marque.lower() in marque\_modele.lower():

total\_bonus\_malus\_match += int(bonus\_malus)

total\_rejets\_co2\_match += int(rejets\_co2)

total\_cout\_energie\_match += int(cout\_energie)

count\_match += 1

# Calculate averages

if count\_match > 0:

avg\_bonus\_malus = total\_bonus\_malus\_match / count\_match

avg\_rejets\_co2 = total\_rejets\_co2\_match / count\_match

avg\_cout\_energie = total\_cout\_energie\_match / count\_match

else:

avg\_bonus\_malus = avg\_rejets\_co2 = avg\_cout\_energie = 0

* + **Deuxième passage : génération des lignes de sortie :**
    - Le fichier *Catalogue.csv* est relu ligne par ligne.
    - Si une ligne contient le mot "marque" (en-tête) et que l'en-tête n'a pas encore été ajouté, il est ajouté à *output\_lines*.
    - Pour chaque ligne de *Catalogue.csv*, le script recherche les correspondances dans *modified\_co2\_data*.
    - Si une correspondance est trouvée, une nouvelle ligne avec les données correspondantes est ajoutée à *output\_lines*.
    - Si aucune correspondance n'est trouvée et que la ligne n'est pas un en-tête, une ligne avec les moyennes calculées est ajoutée à *output\_lines.*

header\_added = False

with open('Catalogue.csv', 'r', encoding='utf-8') as catalogue\_file:

for lineCatalogue in catalogue\_file:

data = lineCatalogue.strip().split(',')

marque = data[0]

if marque.lower() == "marque" and not header\_added:

output\_lines.append(f"{lineCatalogue.strip()},Bonus / Malus,Rejets CO2 g/km,Cout energie")

header\_added = True

continue

match\_found = False

for data1 in modified\_co2\_data:

marque\_modele = data1[1]

bonus\_malus = data1[2]

rejets\_co2 = data1[3]

cout\_energie = data1[4]

if marque.lower() in marque\_modele.lower():

output\_lines.append(f"{lineCatalogue.strip()},{bonus\_malus},{rejets\_co2},{cout\_energie}")

print(f"{lineCatalogue.strip()},\t{bonus\_malus},{rejets\_co2},{cout\_energie}")

match\_found = True

if not match\_found and marque.lower() != "marque":

output\_lines.append(f"{lineCatalogue.strip()},{avg\_bonus\_malus:.2f},{avg\_rejets\_co2:.2f},{avg\_cout\_energie:.2f}")

print(f"{lineCatalogue.strip()},\t{avg\_bonus\_malus:.2f},{avg\_rejets\_co2:.2f},{avg\_cout\_energie:.2f}")

* + **Écriture de la sortie :**

Les lignes générées sont écrites dans le fichier *test.csv.*

writer = csv.writer(csvfile)

for line in output\_lines:

writer.writerow(line.split(","))

1. **Reducer script**

Le script Reducer (reduce.py) se charge de lire l'entrée standard, de calculer les moyennes pour chaque clé unique, et d'écrire les résultats dans un fichier CSV. Voici les étapes détaillées :

* + **Initialisation des accumulateurs et variables d'état :**
* total\_bonus\_malus, total\_rejets\_co2, total\_cout\_energie, number pour accumuler les valeurs.
* lastKey pour suivre la clé précédente.
* output\_lines pour stocker les lignes de sortie.
* i pour vérifier si l'en-tête a été ajouté.

import sys

import csv

# Initialize accumulators and state variables

total\_bonus\_malus = 0

total\_rejets\_co2 = 0

total\_cout\_energie = 0

number = 0

lastKey = None

output\_lines = []

i = 0

* + **Lecture de l'entrée standard et Calcul des moyennes :**
* Chaque ligne est lue et décomposée en clé (*catalogue*) et valeurs (*bonus\_malus, rejets\_co2, cout\_energie*).
* Si la ligne contient "Bonus / Malus" (en-tête) et que c'est la première occurrence, elle est ajoutée au début de *output*\_*lines*.
* Si la clé courante est différente de *lastKey* et que *lastKey* n'est pas None, les moyennes pour *lastKey* sont calculées et ajoutées à *output\_lines*.
* Les accumulateurs sont réinitialisés pour la nouvelle clé.
* Si la clé courante est la même que *lastKey*, les valeurs sont ajoutées aux accumulateurs.

for line in sys.stdin:

line = line.strip()

if not line:

continue

catalogue, co2 = line.split('\t')

bonus\_malus, rejets\_co2, cout\_energie = co2.split(',')

if bonus\_malus.lower() == "bonus / malus":

if i==0:

# add this to the debut of the line

output\_lines.insert(0,f"{catalogue}{bonus\_malus},{rejets\_co2},{cout\_energie}")

i+=1

continue

if lastKey and catalogue != lastKey:

avg\_bonus\_malus = total\_bonus\_malus / number

avg\_rejets\_co2 = total\_rejets\_co2 / number

avg\_cout\_energie = total\_cout\_energie / number

output\_lines.append(f"{lastKey}{avg\_bonus\_malus:.2f},{avg\_rejets\_co2:.2f},{avg\_cout\_energie:.2f}")

total\_bonus\_malus = float(bonus\_malus)

total\_rejets\_co2 = float(rejets\_co2)

total\_cout\_energie = float(cout\_energie)

number = 1

else:

total\_bonus\_malus += float(bonus\_malus)

total\_rejets\_co2 += float(rejets\_co2)

total\_cout\_energie += float(cout\_energie)

number += 1

lastKey = catalogue

* + **Traitement de la dernière clé :**

Une fois toutes les lignes traitées, les moyennes pour *lastKey* sont calculées et ajoutées *à output\_lines*.

if lastKey:

avg\_bonus\_malus = total\_bonus\_malus / number

avg\_rejets\_co2 = total\_rejets\_co2 / number

avg\_cout\_energie = total\_cout\_energie / number

output\_lines.append(f"{lastKey}{avg\_bonus\_malus:.2f},{avg\_rejets\_co2:.2f},{avg\_cout\_energie:.2f}")

* + **Écriture de la sortie :**

Les lignes générées sont écrites dans le fichier *Catalogue\_CO2.csv*.

with open('Catalogue\_CO2.csv', 'w', newline='', encoding='utf-8') as csvfile:

writer = csv.writer(csvfile)

writer.writerows([line.split(",") for line in output\_lines])

1. **Pour lancer le traitement map reduce**

hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \

-input modified\_CO2.csv \

-output updated\_catalogue \

-mapper /vagrant/Groupe\_TPA\_4/mapreduceCo2/example/map.py \

-reducer /vagrant/Groupe\_TPA\_4/mapreduceCo2/example/red.py

# Automatisation des scripts

Afin de minimiser les risques d'erreurs et d'optimiser l'efficacité opérationnelle, il était préférable et avantageux d'automatiser une grande partie des scripts. Cette démarche permet non seulement de réduire les interventions manuelles, sources potentielles de fautes, mais également de gagner plus de temps en n’exécutant qu’un seul script au lieu de plusieurs lignes de scripts une par une.

* On mentionne que tous les logs de ce script sont visibles et vous pourriez les suivre en temps réels. Vous verrez alors au tout début du script une date de début.

* La partie démarrage et installation se feront automatiquement et dans l’ordre. Vous n’auriez donc plus à vous souciez si vous avez oublié de démarrer ou installer quoique ce soit. Voir ci-dessous un résumé des étapes réalisées lors de ce démarrage (cela a été détaillé en haut).

echo -e "--------------- DEMARRAGE ET INSTALLATION ----------------"

# démarrage hadoop HDFS si ce n'est pas déjà fait

# démarrage hadoop YARN si ce n'est pas déjà fait

# installation pandas si ce n'est pas déjà fait

# démarrage du serveur oracle nosql au premier lancement de la VM en mode non secure

# démarrage du serveur hadoop HIV

* Ensuite on a la partie alimentation clients & immatriculations dans mongoDB et création des tables dans hive. Pour cela on a la date de début de ce processus et la date de fin. Cela nous permet donc d’obtenir la durée totale de ce processus en minutes.

echo -e "-------------------------- ALIMENTER CLIENTS & IMMATRICULATIONS DANS MONGODB || CREER TABLE DANS HIVE ---------------------------------------"

echo "Beginning Date Mongo Processing"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

datemongostart=$(date +%s)

./script-mongodb.ksh

echo "Ending Date Mongo Processing"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

datemongoend=$(date +%s)

# calculing duration of process in minutes

durationmongo=$(echo "scale=2; ($datemongoend - $datemongostart) / 60" | bc)

* La partie chargement des données marketing dans oracle NOSQL et l’insertion de ces données dans la table externe de hive a été également automatisée. On a tout de même la date de début, la date de fin et la durée de ce processus.

echo -e "---------------------------- ALIMENTER MARKETING DANS ORACLE NOSQL | INSERER DANS TABLE EXTERNE HIVE -----------------------------------"

echo "Beginning Date Oracle Processing"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

dateoraclestart=$(date +%s)

./script-oracle.ksh

echo "Ending Date Oracle Processing"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

dateoracleend=$(date +%s)

# calculing duration of process in minutes

durationoracle=$(echo "scale=2; ($dateoracleend - $dateoraclestart) / 60" | bc)

* Dernièrement on a la partie alimentation des données catalogues et CO2 dans HDFS, la partie Map Reduce et le chargement de ces données dans la table externe de hive. Tout comme les autres étapes en haut, la date de début, la date de fin ainsi que la durée de ce processus est visible. Voici un résumé du code :

echo -e "-------------------------- ALIMENTER CATALOGUE ET CO2 DANS HDFS | MAP REDUCE | INSERER DANS TABLE EXTERNE HIVE ---------------------------"

echo "Beginning Date HDFS MapReduce Processing"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

datehdfsstart=$(date +%s)

./script-hdfs-mapreduce.ksh

echo "Ending Date HDFS MapReduce Processing"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

datehdfsend=$(date +%s)

# calculing duration of process in minutes

durationhdfs=$(echo "scale=2; ($datehdfsend - $datehdfsstart) / 60" | bc)

* A la fin du script, on a la date de fin, le temps écoulé pour chaque étape et la durée totale de tout le script. Cela va vous permettre de réaliser laquelle des étapes a pris le plus de temps.

echo -e "---------------------------FIN DU SCRIPT -------------------------------------"

echo -e "Ending Date"

date --date="+2 hours" +"%Y-%m-%d %H:%M:%S"

end=$(date +%s)

duration=$(echo "scale=2; ($end - $start) / 60" | bc)

echo -e "\nDURATION MONGODB & CREATE TABLE IN HIVE: $durationmongo minutes"

echo -e "DURATION ORACLE & HIVE: $durationoracle minutes"

echo -e "DURATION HDFS & MAP REDUCE & HIVE: $durationhdfs minutes\n"

echo -e "TOTAL DURATION: $duration minutes\n\n"

* En se référant au schéma de l’architecture du projet, voici donc les parties ayant été automatisées :

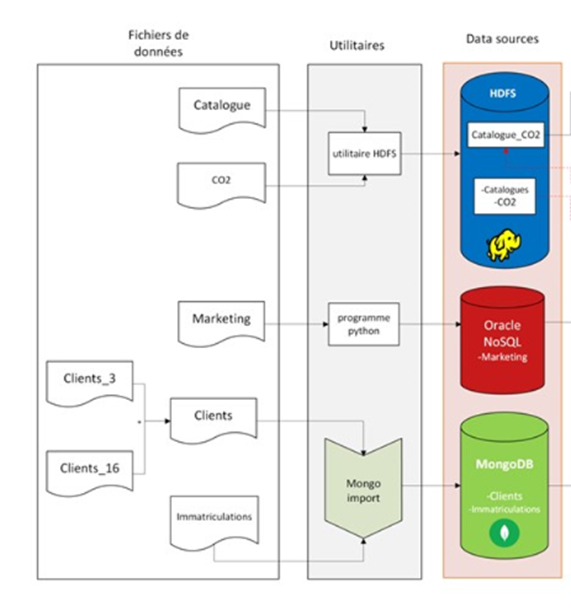


Figure 8 : Chargement des données dans data sources

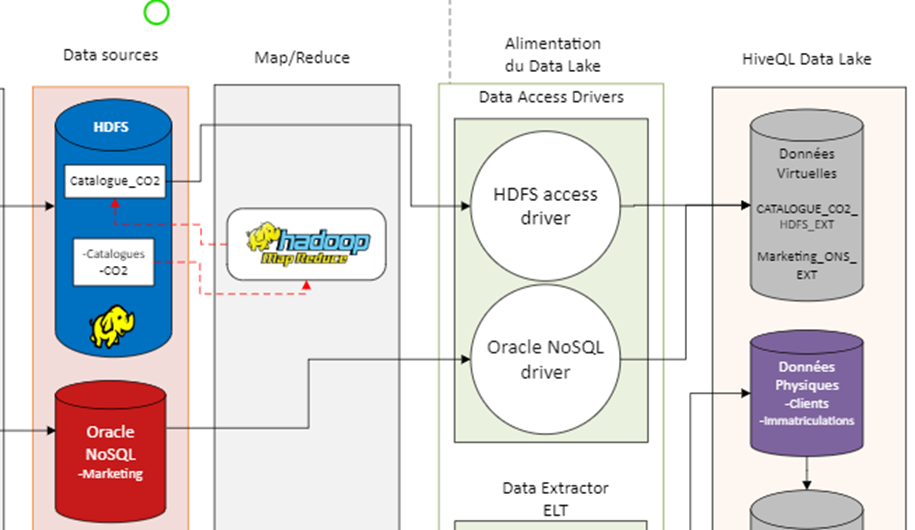


Figure 9 : Map reduce et chargement des données virtuelles dans hiveQL.

# Analyse de données avec Python

Lors de cette phase, nous avons chargé les données de modèle d’analyse depuis le frontal HIVE.

On y a créé une vue fusionnant les données Clients et Immatriculations, nommée ‘clients immatriculations’ afin d’obtenir une matrice dans laquelle chaque ligne correspondra à un client avec les caractéristiques du véhicule qu’il a acquis.

La partie Analyse de données du projet Concessionnaire d’Automobile sera présentée dans la partie ci-présente.

Tout d’abord, voici la liste des tables et des vues utilisées pour effectuer toutes les analyses prévues :

* + **Catalogue** : table externe pour identifier les catégories de véhicules existantes et sera utilisées plus tard dans la partie prédiction,
  + **Clients et Immatriculations** : vue créée dans la base de données HIVE qui sera principalement utilisée pour l’entrainement de notre modèle de prédiction,
  + **Marketing** : table externe utilisée pour prédire les catégories de voiture adaptée pour chaque client présent dans celle-ci

1. **Analyse exploratoire des données**

En premier lieu, nous avons commencé par l’analyse de la table « *catalogue* » afin de pouvoir déterminer les catégories existantes et possible à utiliser ensuite pour essayer d prédire les catégories de véhicules des clients ciblés.

A partir des données de la table catalogue et aussi de comparaison de catalogues de véhicules disponibles en ligne, nous en avons déduit alors **5 catégories** de véhicules auxquelles appartiennent ceux dans catalogue à savoir les catégories : **citadine, familiale, sportive, monospace et luxe**.

Afin d’utiliser à bon efficient les données de la vue ‘clients immatriculations’ et aussi plus tard après entrainement de notre modèle, la table « marketing », nous avons d’abord effectuer quelques traitement et nettoyage :

Suppression des colonnes **« id, immatriculations, marque, nom, couleur** »

Modification des valeurs en Chaîne de Caractères en valeurs numériques.

Ci-dessous le tableau listant les colonnes et les valeurs correspondantes changées.

|  |  |  |  |
| --- | --- | --- | --- |
| **Colonne** | **Valeurs en Chaine de Caractères** | **Valeurs numériques** | **Table concernée** |
| **longueur** | tr�s longue | 3 | **Clients Immatriculations et Marketing** |
| longue | 2 |
| moyenne | 1 |
| courte | 0 |
| **occasion** | True | 1 |
| False | 0 |
| **sexe** | M | 1 |
| F | 0 |
| NULL | -1 |
| **situationfamiliale** | Divorc�e | -1 |
| ?  ‘’  ? | None |
| Seule  Seul  C�libataire | 0 |
| En Couple | 1 |
| Mari�(e) | 2 |
| **age** | ‘ ‘  -1  ? | N/D, plus tard transformée en la valeur **médiane** de la colonne |
| **nbenfantsacharge** | ‘’  ‘ ‘  -1  ? | N/D, plus tard transformée en la valeur **médiane** de la colonne |

Tableau 1 : Liste des valeurs transformées

1. **Clustering**

La partie clustering consistera à regrouper les données obtenues et nettoyées dans ‘clients immatriculations » dans différentes catégories de véhicule, obtenues dans l’analyse des données de la table « catalogue ».

Pour ce faire voici les sous-étapes effectuées dans cette partie :

Utilisation de la méthode K-Means pour la catégorisation des données. K-Means est une méthode qui permet de répartir les points de données en clusters en fonction de leur proximité avec des points centraux (centroïdes) et en ajustant ces centroïdes jusqu'à ce que la meilleure répartition soit trouvée.

Recherche de K-Optimal : pour savoir quel est le nombre de catégories optimales pour répartir les données, nous avons utilisé la méthode du coude ou « *Elbow Method* ». Elle se base sur l'analyse de la somme des distances au carré des points à leur centroïde le plus proche, appelée somme des erreurs au carré (SSE - Sum of Squared Errors).

On applique l’algorithme K-Means pour différentes valeurs de k (dans notre cas ; de 2 à 11).

Pour chaque valeur de 𝑘, on calcule la SSE, qui mesure la compacité des clusters. Plus la SSE est petite, plus les points sont proches de leurs centroïdes respectifs.

On trace un graphique avec 𝑘 sur l'axe horizontal et la SSE correspondante sur l'axe vertical.

Le point où la diminution de la SSE commence à ralentir de manière significative (et forme une sorte de coude ou d'angle) est considéré comme le nombre optimal de clusters. Avant ce point, ajouter des clusters améliore significativement la compacité (réduction de la SSE), mais après ce point, les améliorations deviennent marginales.

On peut observer cette courbe dans l’image ci-dessous.

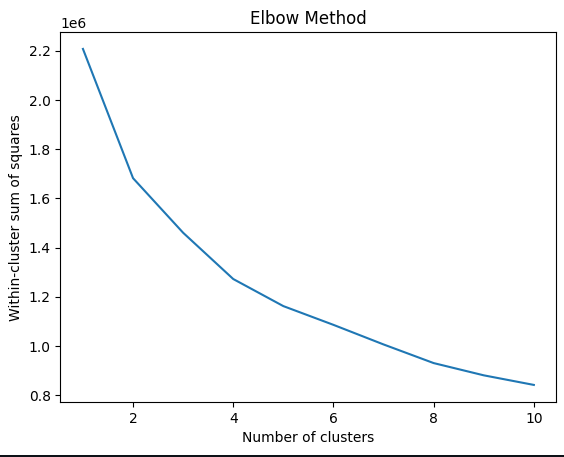


Figure 10 : Courbe d'identification du nombre de Cluster Optimal (Elbow Method)

Dans notre cas, le nombre de cluster optimal est donc aux environs de 3 à 4 clusters, d’où l’utilisation de 4 comme nombre de catégories totaux.

Après avoir subdivisé nos données en 4 clusters, nous devons attribuer à chaque cluster la catégorie de véhicule qui lui correspond. Pour cela, nous avons choisi en fonction des caractéristiques moyennes de chaque cluster :

* Moyenne de prix supérieure à 80000 : Catégorie Voiture de **Luxe**
* Moyenne de la puissance supérieure à 400 et Moyenne de prix supérieure à 50000 : Catégorie Voiture **Sportive**
* Moyenne du nombre d’enfants à charge inférieure à 1 : Catégorie Voiture **Citadine**
* Moyenne de la puissance supérieure à 200 et Moyenne de prix inférieure à 50000 : Catégorie Voiture **Familiale**

1. **Construction du modèle d’Analyse**

Nous avons d’abord commencé par introduire la nouvelle colonne *catégorie* dans notre grande matrice de « clients immatriculations ».

Vu que notre matrice contient un volume assez conséquent de données, nous devons d’abord la sous échantillonner pour alléger la phase d’entraînement de nos modèles

Ensuite, nous avons subdiviser les données d’entrainement et de test : **20 %** de données de test.

Pour la construction de notre modèle, nous avons opté pour obtenir le meilleur modèle de prédiction possible en testant plusieurs types de modèles mais aussi en utilisant les paramètres possibles. Voici la liste des modèles utilisées et les meilleurs paramètres choisis pour chacun mais aussi le score pour identifier le niveau de cohérence du modèle :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nom du modèle** | **Description du modèle** | **Paramètre choisi** | **Description du paramètre** | **Valeurs testées possibles** |
| **Decision Tree**  **(Arbre de decision)** | Modèle de Machine Learning qui utilise une structure en arborescence pour prendre des décisions en divisant les données en sous-ensembles basés sur des tests de caractéristiques, permettant ainsi de prédire la valeur cible à chaque feuille. | **Criteria (Critères)** | Mesures de l'homogénéité des nœuds, utilisées pour déterminer la qualité des divisions en maximisant la pureté des sous-ensembles à chaque bifurcation. | Gini  Entropy (Entropie) |
| **min\_samples\_leaf** | Nombre minimum d'échantillons requis pour qu'un nœud soit une feuille, permettant ainsi de contrôler la taille minimale des sous-ensembles afin d'éviter le surapprentissage. | 1.5  10 |
| **Max\_depth** | Profondeur maximale de l'arbre, limitant le nombre de niveaux pour éviter le surapprentissage et contrôler la complexité du modèle. | None  10  20  30 |
| **Min\_samples\_split** | Nombre minimum d'échantillons requis pour qu'un nœud soit divisé, aidant ainsi à prévenir le surapprentissage en assurant qu'il y ait suffisamment de données pour justifier une nouvelle division. | 2  5  10 |
| **Ccp\_alpha** | Coefficient de complexité pour l'élagage minimal (Minimal Cost-Complexity Pruning), qui élaguera les sous-arbres dont la complexité ajoutée est inférieure à ce seuil, aidant ainsi à simplifier le modèle et à éviter le surapprentissage. | 0  0.01  0.1 |
| **Random Forest (Forêt Aléatoire)** | Modèle de machine learning qui combine plusieurs arbres de décision, chacun construit sur un sous-échantillon différent des données, pour améliorer la précision et la robustesse des prédictions en réduisant le risque de surapprentissage. | **N\_estimators** | Nombre d'arbres de décision à construire, influençant la précision du modèle et sa stabilité, avec un plus grand nombre d'arbres généralement conduisant à de meilleures performances mais à un coût de calcul accru. | 50  100  200 |
| **Criteria** | (Paramètre déjà décrit précédemment) | Gini  Entropy |
| **KNN (k-plus proches voisins)** | Algorithme de machine learning basé sur la similitude des instances, où la classe ou la valeur d'une observation est déterminée par la majorité ou la moyenne des k instances les plus proches dans l'espace des caractéristiques. | **N\_neighbors** | Nombre de voisins à considérer pour déterminer la classe ou la valeur d'une nouvelle observation, influençant la précision et la complexité du modèle, avec un petit nombre de voisins rendant le modèle plus sensible aux variations locales et un plus grand nombre le rendant plus stable mais moins flexible. | 3  5  7 |
| **Weight** | Méthode de pondération des voisins lors de la prédiction, où les poids peuvent être uniformes (chaque voisin contribue de manière égale) ou inversement proportionnels à leur distance (voisins plus proches ont plus d'influence), permettant de tenir compte de la proximité des voisins lors de la prédiction. | Uniform  Distance |
| **Gradient Boosting** | Construit un modèle prédictif en combinant plusieurs modèles de prédiction plus faibles (souvent des arbres de décision peu profonds) de manière séquentielle, où chaque modèle corrigera les erreurs du modèle précédent en se concentrant sur les exemples mal prédits, permettant ainsi d'obtenir des performances prédictives élevées. | **N\_estimators** | (Paramètre déjà décrit précédemment) | 50  100  200 |
| **Learning Rate** | Contrôle la contribution de chaque arbre à l'ensemble en ajustant la valeur de prédiction de chaque arbre ajouté | 0.01  0.1  0.2 |
| **Max\_depth** | (Paramètre déjà décrit précédemment) | 3  5  7 |
| **Ada\_Boost** | Construit un modèle prédictif en combinant plusieurs modèles de prédiction faibles (appelés des "apprenants faibles") de manière séquentielle. Il accorde une attention accrue aux exemples mal classés à chaque itération, permettant ainsi d'obtenir un modèle plus fort en corrigeant progressivement les erreurs des modèles précédents. | **N\_estimators** | (Paramètre déjà décrit précédemment) | 50  100  200 |
| **Learning\_rate** | (Paramètre déjà décrit précédemment) | 0.01  0.1  1.0 |

Tableau 2 : Liste des modèles et des paramètres correspondants

Apres entrainement des modèles voici les résultats obtenus avec les modèles et paramètres les plus performants possibles :

Pour le Decision tree avec ccp\_alpha de 0.01, criterion : entropy, max\_depth : aucune, min\_samples\_leaf : 1, min\_samples\_split : 1 avec un score de **0.82**

Pour le Random Forest avec criterion : ‘entropy’ et n\_estimators : 50 avec un score de **0.79**

Pour le KNN, avec n\_neighbors : 5 et weights : distance avec un score de **0.67**

Pour le Gradient Boosting : learning rate de 0.01, max\_depth de 3, n\_estimators de 200 avec un score de **0.83**

Pour l’Adaboost, learning rate de 0.1, n\_estimators : 100 avec un score **0.73**

Pour permettre la réutilisation des modèles sans devoir re-entraîner à chaque fois tous nos modèles avec tous les paramètres correspondants et à l’avenir charger simplement le fichier pour utiliser un modèle, nous avions utilisé la librairie *joblib* qui nous a permis de sauver les modèles à l’intérieur de fichiers binaires sous l’extension « .pkl ». Cette technique est aussi utile lors d’utilisation de machines ne permettant pas l’entraînement des modèles faute de ressources ou lorsque l’on a recours à des données très volumineuses.

Après comparaison des scores et recherche du modèle ayant le score maximal, le modèle *Gradient Boosting* est le plus performant par son score de **0.83** donc sera utilisé par la suite pour la partie prédiction.

1. **Application du modèle de prédiction aux données Marketing**

Cette dernière sous-partie consiste à appliquer notre modèle de prédiction le plus performant avec les paramètres les plus adaptés (**Gradient Boosting** avec un **learning rate** de 0.01, **n\_estimators** : 200, **max\_depth** : 3 pour notre cas) avec un score de 0.83 sur les données de la table Marketing. Mais avant l’introduction de la nouvelle variable « catégorie », nous avons procédé au nettoyage et au traitement de le table marketing (même traitement que pour la vue « clients immatriculations » pour le changement des colonnes non numériques en colonnes numériques à savoir les colonnes : **sexe, situation familiale, deuxième voiture**).

Nous obtenons alors une nouvelle matrice avec la colonne catégorie prédite par notre modèle d’Analyse de données.

1. **Envoi des nouvelles données de Marketing /Prospect vers la base de résultat**

Pour finaliser la partie Analyse de Données du projet, nous allons enfin renvoyer les données Marketing contenant la nouvelle colonne catégorie vers une collection nommée « result\_datas\_prediction\_marketing » (résultats des données de prédiction de la table Marketing) dans la base de données Concessionnaire de Mongo DB (Base de Résultat).

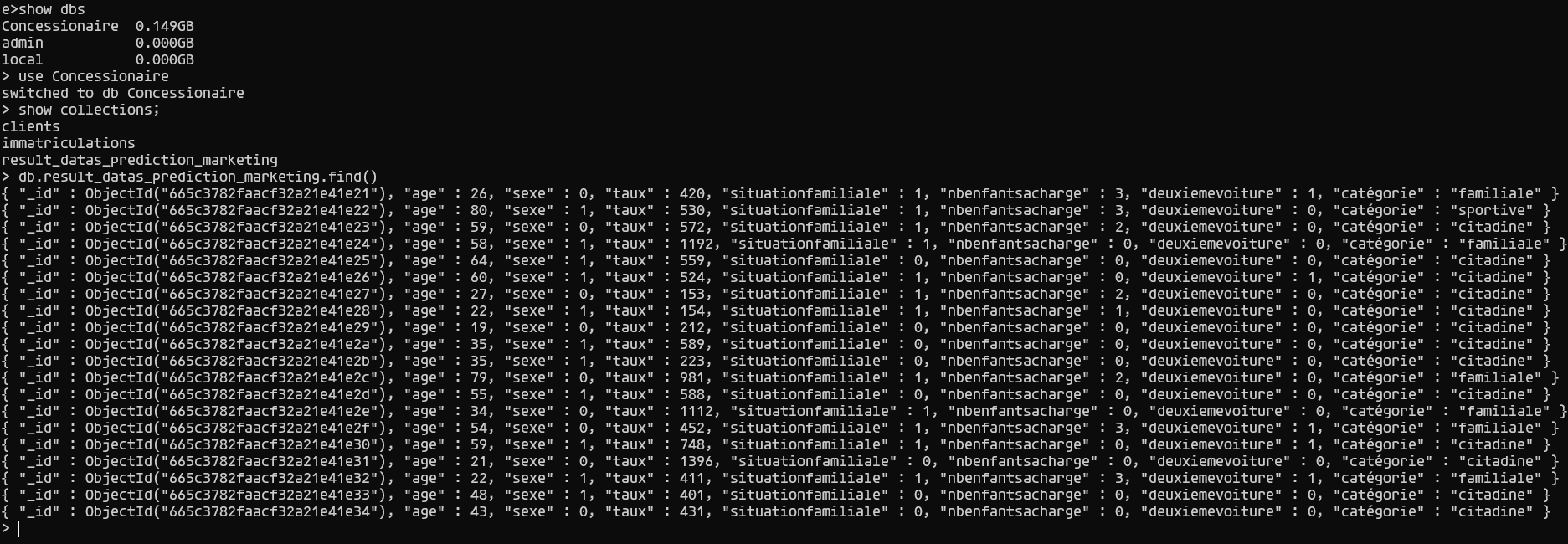
La figure ci-présente est un aperçu des données dans la base de données Non Relationnelle Mongo DB.

Figure 11 : Aperçu de la table de résultats dans Mongo DB

# Conclusion générale

Au terme de ce projet d'analyse de données pour les concessionnaires automobiles, nous dressons un bilan positif des résultats obtenus pour l'entreprise. Grâce à notre solution, les concessionnaires disposent désormais d'un outil pour mieux cibler leurs clients et proposer des véhicules adaptés à leurs besoins, renforçant ainsi leur compétitivité sur le marché.

Cependant, nous avons rencontré des défis majeurs tout au long du projet, notamment en ce qui concerne l’utilisation de l’extracteur de données, NiFi. Nous avons dû apprendre à le manipuler depuis les bases et avons rencontrés divers obstacles qui n’ont heureusement pas été insurmontables.   
Pour la partie Analyse des données, le volume de données pour l’entraînement de nos modèles ont aussi été un défi que nous dû surmonter car sinon, la lenteur et le temps nécessaire à l’entraînement de notre modèle d’analyse sera très vite ingérable. En guise de solutions, nous avons alors procédés à ces 2 étapes à savoir :

* La réduction du nombre de données pour l’entraînement du modèle ce qui a bien évidemment légèrement impacté le score de précision de nos modèles (et surtout de notre meilleur « *Gradient Boosting* » avec un score de précision d’environ 0.83) mais ce score nous a quand même permis de prédire avec une marge d’erreurs les catégories des véhicules dans Marketing
* L’utilisation de la librairie « *joblib* » qui nous a permis de stocker les modèles sous formes de fichiers binaires et donc de facilement les charger lorsque cela est nécessaire.

Pour l'avenir du projet, nous envisageons d'explorer de nouvelles possibilités d'amélioration, telles que l'optimisation des processus d'analyse de données et l'intégration de nouvelles sources de données pour enrichir notre modèle et avoir de meilleures prédictions.

En ce qui concerne le bilan personnel, ce projet a été une expérience enrichissante qui nous a permis de mettre en œuvre les théories et les cours sur le Big Data, le MapReduce et l'analyse de données et machine learning dans un contexte concret. Nous sommes fiers d'avoir contribué au succès de ce projet et nous sommes reconnaissants pour les enseignements et les compétences que nous avons acquis tout au long de ce parcours.

# Références et Bibliographie

* Bibliographie

[1] Nicolas Pasquier, Apprentissage d'Arbres de Décision, [UCApod - Cm03dmml010.Mp4 (univ-cotedazur.fr)](https://pod.univ-cotedazur.fr/video/13068-cm03dmml010mp4/)

[2] Nicolas Pasquier, Cours Magistral, [UCApod - Cm06dmml010.Mp4 (univ-cotedazur.fr)](https://pod.univ-cotedazur.fr/video/13076-cm06dmml010mp4/)

[3] Gabriel MOPOLO-MOKE, MOOCEBFR4 : Ingénierie des Données du Big Data : SGBD NoSql et Lacs de Données avec Big Data SQL par la pratique

[4] Benjamin Renaut, mis à jour par Sergio Simonian, Hadoop, Spark & Map/Reduce

[5] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984), Classification and regression trees. CRC press.

[6] Liaw, A., & Wiener, M. (2002). Classification and regression by random Forest. R news, 2(3), 18-22.

* Sites internet :

[1] Sergio Simonian, A Debian vagrant box with Hadop, <https://github.com/SergioSim/debian-hadoop>

[2] Apache Nifi, Documentation, <https://nifi.apache.org/documentation/v1/>

[3] Medium, How to install NiFi on OCI, <https://learnoci.cloud/how-to-install-nifi-on-oci-af0d9b90d61b>

[4] Github, How to connect To your mongo database inside a Vagrant Virtual Machine from The Windows Host, <https://github.com/Junx2001/Tutorials/blob/main/Mongo%20DB%20Vagrant.md>

[5] My Cloudera, Cloudera Community, [Missing Hive Processors in Nifi 1.19.1](https://community.cloudera.com/t5/Support-Questions/Missing-Hive-Processors-In-Nifi-1-19-1/m-p/359666)

[6] stackoverflow, [PutHiveQL NiFi Processor extremely slow - misconfiguration?](https://stackoverflow.com/questions/57460770/puthiveql-nifi-processor-extremely-slow-misconfiguration)

[7] Trello, Gestion et suivi des tâches pour le projet, <https://trello.com/b/lZpk0nId/mbds-projet-big-data>

# Annexes

## 11.1 Vidéo de présentation

Vidéo sur Microsoft Stream : [Video\_TPA\_G4](https://unice-my.sharepoint.com/personal/sandy-princia_ramaroson_etu_unice_fr/_layouts/15/stream.aspx?id=%2Fpersonal%2Fsandy%2Dprincia%5Framaroson%5Fetu%5Funice%5Ffr%2FDocuments%2FVideo%5FTPA%5FG4%2Emp4&nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&ga=1&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Eb5c2123e%2D2ff1%2D46c3%2Db8f2%2Df0d4aaf650cd)

## 11.2 Dossier contenant les scripts et programmes de construction du lac de données

Même dossier que ce rapport : /Groupe\_TPA\_4/scripts-TPA

Fichier d’automatisation du script à lancer : /Groupe\_TPA\_4/scripts-TPA/script-tpa.ksh

**Vous devez lancer ce script** en

- se plaçant dans le dossier /Groupe\_TPA\_4/scripts-TPA de votre machine vagrant

- exécutez ensuite ./script-tpa.ksh

cd /vagrant/Groupe\_TPA\_4/scripts-TPA

./script-tpa.ksh

## 11.3 Dossier contenant les scripts et programmes Hadoop Map Reduce

Même dossier que ce rapport : /Groupe\_TPA\_4/scripts-TPA

Script que scripts-tpa.ksh appel : /Groupe\_TPA\_4/scripts-TPA/script-hdfs-mapreduce.ksh **(ne lancez plus ce script)**

## 11.4 Dossier contenant les scripts et programmes d’analyse de données

Même dossier que ce rapport : /Groupe\_TPA\_4

Programme Python pour l’analyse de données : /Groupe\_TPA\_4/Analyse\_Donnees – Final.ipynb

Ce fichier est indépendant du script d’automatisation, **vous devez l’ouvrir dans Jupyter Notebook et l’exécuter**, comme montré dans la section « [Exploitation des données sur Python](#_Exploitation_des_données)».

1. Le port forwarding est une technique qui permet de rediriger le trafic réseau entrant d'un port spécifique vers une machine ou un service différent. Ici, on a change le redirigé le trafic du port 8443 vers 8444. [↑](#footnote-ref-1)