INTRODUCTION

Contexte et objectif du projet

Nous avons été contactés par un concessionnaire automobile afin de l'aider à mieux cibler les véhicules susceptibles d'intéresser ses clients. Pour cela il met a mis disposition :
• Son catalogue de véhicules
• Son fichier clients concernant les achats de l'année en cours
• Un accès à toutes les informations sur les immatriculations effectuées cette année
• Une brève documentation des données
• Un vendeur (voir son interview ci-dessous).

A la fin de ce projet nous devons proposer un outil permettant :

- d'évaluer en temps réel le type de véhicule le plus susceptible d'intéresser des clients qui se présentent dans la concession ;
- d'envoyer une documentation précise sur le véhicule le plus adéquat pour des clients sélectionnés par son service marketing (voir ci-dessous).

Données

Le concessionnaire a mis à notre disposions plusieurs fichiers contenant des données sur les véhicules.

- immatriculation.csv :
- *Catalogue.csv*
- *Marketing.csv*
- *Client1.csv*
- *Client2.csv*
- *Co2.csv*

Pour la réalisation de ce projet, nous allons d'abord construire un DataLake grâce aux connaissances sur le BigData. Ensuite, nous allons nous basé sur cet DataLake pour construire un modèle de machine Learning et effectuer les recommandations de véhicules pour le concessionnaire.

Chapitre 1 : Construction de DATALAKE

I- ARCHITECTURE BIG DATA

1- Définition

Une architecture Big Data est un ensemble de composants matériels et logiciels conçus pour traiter des quantités massives de données. Les architectures Big Data sont conçues pour être hautement évolutives et flexibles, et elles sont souvent mises en œuvre dans des environnements distribués pour permettre le traitement parallèle des données. Ces architectures peuvent stocker et traiter des données structurées, semi-structurées et non structurées, et sont conçues pour relever les défis de l'analyse de données à grande échelle.

Source : https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/big-data

2- Présentation de notre architecture

D'abord, nous allons définir l'architecture de notre DataLake. Il sera MIXTE (Physique et Virtuelle) construire autour de HIVE. Dans notre architecture, nous utilisons HIVE comme frontale. Elle est composée d'une base de données :

- de type fichier HDFS qui contient les fichiers Immatriculations, CO2 et clients1
- MongoDB contenant les fichier Marketing
- Oracle NoSQLDB dans le lequel nous avons mis le fichier catalogue
- HIVE contenant le fichier client2.

Les données en dehors de HIVE sont considérées comme virtuelle donc accessibles grâce à des tables externes et ceux dans HIVE seront considérées comme physique accessible à travers des tables internes.

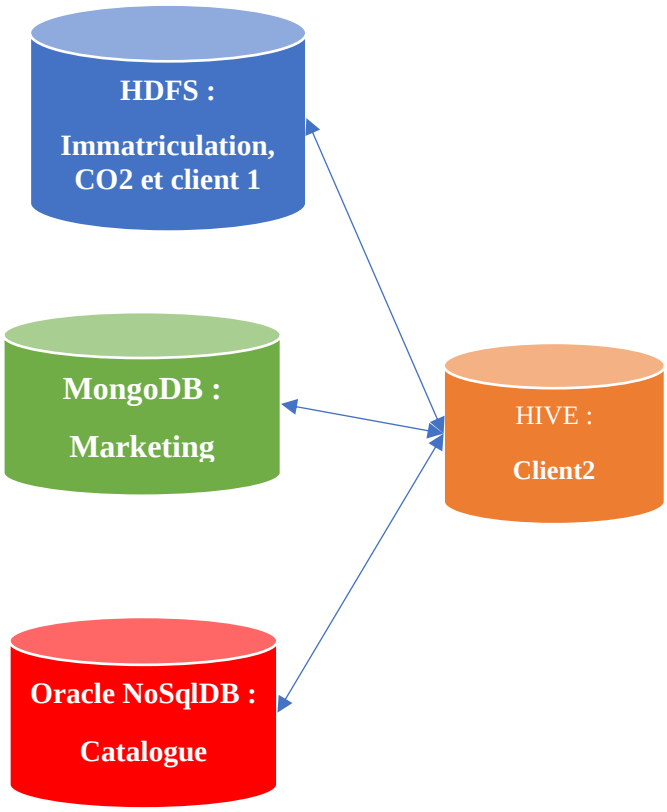


Figure 1 Architecture DataLake

3- Mise en place des environnements de l'architecture

Nous allons présenter les scripts qui ont permis de créer les différentes bases de données et quelques résultats des requêtes de celles-ci.
Notre travaillons avec une machine virtuelle basé sur CentOS. Pour toutes les opérations, nous démarrons la machine virtuelle avec la commande **vagrant up** et effectuons la connexion à celle-ci avec **vagrant ssh.**

3-1 – HDFS

On démarre d'abord **hdfs** en faisant start **start-dfs.sh**

Nous utilisons les commandes suivantes pour mettre les données dans **HDFS**

```
hadoop fs -put /vagrant/VM_DATA/CO2 CO2
hadoop fs -put /vagrant/VM_DATA/Immatriculations Immatriculations
hadoop fs -put /vagrant/VM_DATA/client client1
```

On vérifie que les fichiers sont bien présents

```
[vagrant@oracle-21c-vagrant ~]$ hadoop fs -ls
Found 3 items
drwxr-xr-x   - vagrant supergroup          0 2023-02-15 13:26 CO2
drwxr-xr-x   - vagrant supergroup          0 2023-02-15 13:28 Client1
drwxr-xr-x   - vagrant supergroup          0 2023-02-15 15:22 Immatriculations
[vagrant@oracle-21c-vagrant ~]$
```

3-2 – MongoDB

Pour envoyer les données dans mongo, on utilise :

```
mongoimport --db tpt --collection marketing --type csv --headerline --ignoreBlanks –file
/vagrant/VM_DATA/Marketing/Marketing.csvc
```

On se connecte à la base de données **tpt** et on vérifie si la collection **marketing** est bien présente

```
MongoDB shell version v3.4.24
connecting to: mongodb://127.0.0.1:27017
MongoDB server version: 3.4.24
Server has startup warnings:
2023-03-10T13:31:32.548+0100 I CONTROL  [initandlisten]
2023-03-10T13:31:32.548+0100 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2023-03-10T13:31:32.548+0100 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2023-03-10T13:31:32.548+0100 I CONTROL  [initandlisten]
> show dbs
admin  0.000GB
local  0.000GB
tpt    0.000GB
> use tpt
switched to db tpt
> show tables
marketing
> db.marketing.find()
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb69"), "age" : 21, "sexe" : "F", "taux" : 1396, "situationFamiliale" : "C◆libataire", "nbEnfantsAcharge" : 0, "2eme
voiture" : "false" }
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb6a"), "age" : 35, "sexe" : "M", "taux" : 223, "situationFamiliale" : "C◆libataire", "nbEnfantsAcharge" : 0, "2eme
voiture" : "false" }
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb6b"), "age" : 48, "sexe" : "M", "taux" : 401, "situationFamiliale" : "C◆libataire", "nbEnfantsAcharge" : 0, "2eme
voiture" : "false" }
{ "_id" : ObjectId("63ecd19e8890cdc5b1b8eb6c"), "age" : 26, "sexe" : "F", "taux" : 420, "situationFamiliale" : "En Couple", "nbEnfantsAcharge" : 3, "2eme vo
iture" : "true" }
```

3-3 – OracleNosql

Pour les données dans oracleNoSQL, nous utilisons un programme java pour le faire.

3-4 – HIVE

Après avoir mis les données dans les différentes bases de données, on peut maintenant crée des tables externes à partir de HIVE pour y avoir accès. Avant tout, nous démarrons HIVE et effectuons une connexion.

```
[vagrant@oracle-21c-vagrant ~]$ nohup hive --service metastore > /dev/null & nohup hiveserver2 > /dev/null &
[5] 10468
[6] 10469
[vagrant@oracle-21c-vagrant ~]$ nohup: ignoring input and redirecting stderr to stdout
nohup: ignoring input and redirecting stderr to stdout

[vagrant@oracle-21c-vagrant ~]$ beeline
Beeline version 2.3.9 by Apache Hive
beeline> !connect jdbc:hive2://localhost:10000
Connecting to jdbc:hive2://localhost:10000
Enter username for jdbc:hive2://localhost:10000:
Enter password for jdbc:hive2://localhost:10000:
Connected to: Apache Hive (version 3.1.3)
Driver: Hive JDBC (version 2.3.9)
Transaction isolation: TRANSACTION_REPEATABLE_READ
0: jdbc:hive2://localhost:10000>
```

Ensuite nous mettons le fichier client2.csv dans **'/vagrant/VM_DATA/Client2/Client2.csv'** en local.

3-4-1- **Table interne client2**

Nous créons une table **interne** pour pourvoir accéder et manipuler les données interne

```
CREATE TABLE client2 (
age int,
sexe string,
taux int,
situationFamiliale string,
nbEnfantsAcharge int,
2emeVoiture boolean,
immatriculation string
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED S TEXTFILE;
```

Vérifions que nous avons accès aux données :

```
select * from client2 limit 5;
```

Résultat

```
+---------------+---------------+---------------+-----------------------------+-------------------------+-----------------------+---------------------------+
| client2.age   | client2.sexe  | client2.taux  | client2.situationfamiliale  | client2.nbenfantsacharge | client2.2emevoiture   | client2.immatriculation   |
+---------------+---------------+---------------+-----------------------------+-------------------------+-----------------------+---------------------------+
| NULL          | sexe          | NULL          | situationFamiliale          | NULL                    | NULL                  | immatriculation           |
| 36            | M             | 1168          | C◆libataire                 | 0                       | false                 | 3442 PG 87                |
| 77            | M             | 971           | En Couple                   | 2                       | false                 | 3533 DC 75                |
| 35            | F             | 458           | En Couple                   | 4                       | false                 | 1313 JJ 80                |
| 35            | F             | 404           | En Couple                   | 2                       | false                 | 9948 XU 44                |
+---------------+---------------+---------------+-----------------------------+-------------------------+-----------------------+---------------------------+
5 rows selected (4.064 seconds)
0: jdbc:hive2://localhost:10000>
```


Maintenant que nos trois bases de données distantes sont créées nous pouvons créer les tables externes.

3-4-2 Table externe et requêtes

- ✓ Table externe dans MongoDB

```
drop table marketing_hive_mongo_ext ;
CREATE EXTERNAL TABLE marketing_hive_mongo_ext (
id int,
age int,
sexe string,
taux int,
situationFamiliale string,
nbEnfantsAcharge int,
2emeVoiture string
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
WITH SERDEPROPERTIES('mongo.columns.mapping'='{"id":"_id", "age":"age",
"sexe":"sexe", "taux":"taux", "situationFamiliale":"situationFamiliale",
 "nbEnfantsAcharge": "nbEnfantsAcharge", "2emeVoiture" : "2eme voiture"}')
TBLPROPERTIES ('mongo.uri'='mongodb://localhost:27017/tpt.marketing');
```

Requête :

```
select * from marketing_hive_mongo_ext limit 5;
```

Résultat

```
0: jdbc:hive2://localhost:10000>
0: jdbc:hive2://localhost:10000> select * from marketing_hive_mongo_ext limit 5;
+-----------------------------+-----------------------------+-----------------------------+-----------------------------+-----------------------------+
| marketing_hive_mongo_ext.id | marketing_hive_mongo_ext.age | marketing_hive_mongo_ext.sexe | marketing_hive_mongo_ext.taux | marketing_hive_mongo_ext.situationfamiliale | marketing_hive_mongo_ext.nbenfantsacharge | marketing_hive_mongo_ext.2emevoiture |
+-----------------------------+-----------------------------+-----------------------------+-----------------------------+-----------------------------+-------------------------------------------+--------------------------------------+
| NULL                        | 21                          | F                           | 1396                        | Célibataire                 | 0                                         | false                                |
| NULL                        | 35                          | M                           | 223                         | Célibataire                 | 0                                         | false                                |
| NULL                        | 48                          | M                           | 401                         | Célibataire                 | 0                                         | false                                |
| NULL                        | 26                          | F                           | 420                         | En Couple                   | 3                                         | true                                 |
| NULL                        | 27                          | F                           | 153                         | En Couple                   | 2                                         | false                                |
+-----------------------------+-----------------------------+-----------------------------+-----------------------------+-----------------------------+-------------------------------------------+--------------------------------------+
5 rows selected (8.546 seconds)
0: jdbc:hive2://localhost:10000>
```


- ✓ Table externe dans HDFS
  - ➢ Client1

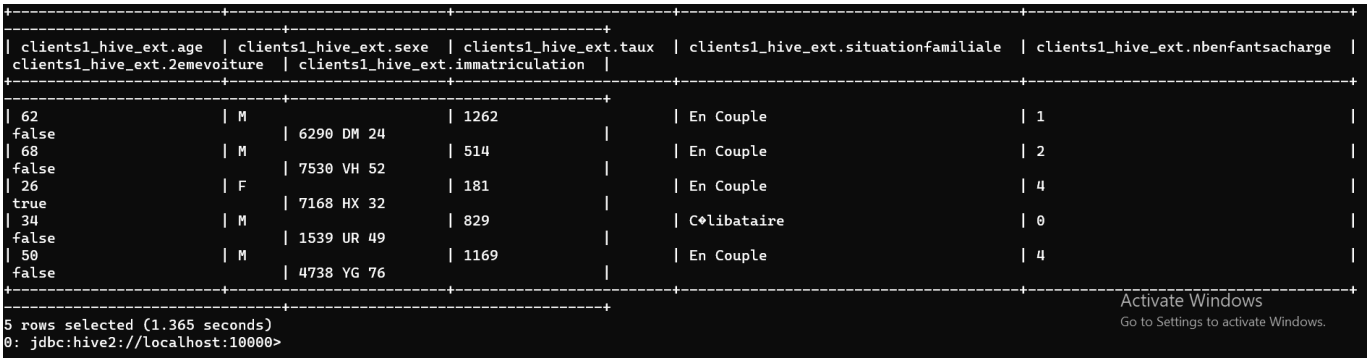
```
drop table clients1_hive_ext;

CREATE EXTERNAL TABLE clients1_hive_ext(
age  int,
sexe string,
taux  int,
situationFamiliale string,
nbEnfantsAcharge int,
2emeVoiture boolean,
immatriculation string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs:/Client1'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Requête

```
select * from clients1_hive_ext limit 5;
```

Résultat :



```
+-------------------------+-------------------------+-------------------------+-----------------------------------+-----------------------------------+
-------------------------------+-----------------------------------+
| clients1_hive_ext.age  | clients1_hive_ext.sexe  | clients1_hive_ext.taux  | clients1_hive_ext.situationfamiliale  | clients1_hive_ext.nbenfantsacharge  |
 clients1_hive_ext.2emevoiture  | clients1_hive_ext.immatriculation  |
+-------------------------+-------------------------+-------------------------+-----------------------------------+-----------------------------------+
-------------------------------+-----------------------------------+
| 62                      | M                       | 1262                    | En Couple                         | 1                                 |
 false                          | 6290 DM 24                        |
| 68                      | M                       | 514                     | En Couple                         | 2                                 |
 false                          | 7530 VH 52                        |
| 26                      | F                       | 181                     | En Couple                         | 4                                 |
 true                           | 7168 HX 32                        |
| 34                      | M                       | 829                     | C◆libataire                       | 0                                 |
 false                          | 1539 UR 49                        |
| 50                      | M                       | 1169                    | En Couple                         | 4                                 |
 false                          | 4738 YG 76                        |
+-------------------------+-------------------------+-------------------------+-----------------------------------+-----------------------------------+
-------------------------------+-----------------------------------+
5 rows selected (1.365 seconds)
0: jdbc:hive2://localhost:10000>
```


- Immatriculation externe

```
drop table immatriculations_hive_ext;
CREATE EXTERNAL TABLE immatriculations_hive_ext(
immatriculation string,
marque string,
nom string,
puissance int,
longueur string,
nbPlaces int,
nbPortes int,
couleur string,
occasion boolean,
prix int
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs:/Immatriculations'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Requête :

```
select * from  immatriculations_hive_ext limit 5;
```

Résultat :

immatriculations_hive_ext.immatriculation	immatriculations_hive_ext.marque	immatriculations_hive_ext.nom	immatriculations_hive_ext.puissance	immatriculations_hive_ext.longueur	immatriculations_hive_ext.nbplaces	immatriculations_hive_ext.nbportes	immatriculations_hive_ext.couleur	immatriculations_hive_ext.occasion	immatriculations_hive_ext.prix
3176 TS 67	Renault	Laguna 2.0T	170	longue	5	5	blanc	false	27300
3721 QS 49	Volvo	S80 T6	272	r♦s longue	5	5	noir	false	50500
9099 UV 26	Volkswagen	Golf 2.0 FSI	150	oyenne	5	5	gris	true	16029
3563 LA 55	Peugeot	1007 1.4	75	ourte	5	5	blanc	true	9625
6963 AX 34	Audi	A2 1.4	75	ourte	5	5	gris	false	18310

5 rows selected (0.513 seconds)
0: jdbc:hive2://localhost:10000>

- Table externe CO2

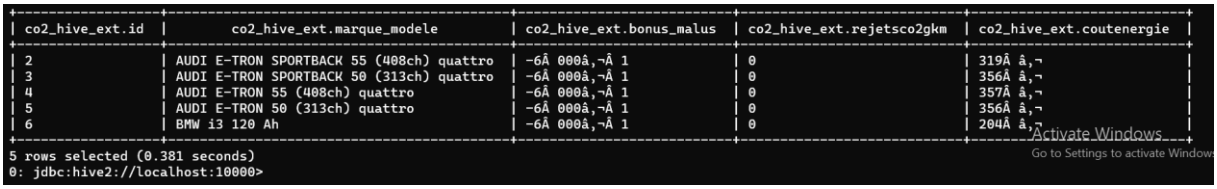
```
drop table CO2_hive_ext;

CREATE EXTERNAL TABLE CO2_hive_ext(
id int,
marque_modele string,
bonus_malus string,
rejetsCO2gkm string,
coutenergie string)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION 'hdfs:/CO2'
TBLPROPERTIES ("skip.header.line.count"="1");
```

Requête :

```
select * from  CO2_hive_ext limit 5;
```

Résultat



co2_hive_ext.id	co2_hive_ext.marque_modele	co2_hive_ext.bonus_malus	co2_hive_ext.rejetsco2gkm	co2_hive_ext.coutenergie
2	AUDI E-TRON SPORTBACK 55 (408ch) quattro	-6Â 000â,¬Â 1	0	319Â â,¬
3	AUDI E-TRON SPORTBACK 50 (313ch) quattro	-6Â 000â,¬Â 1	0	356Â â,¬
4	AUDI E-TRON 55 (408ch) quattro	-6Â 000â,¬Â 1	0	357Â â,¬
5	AUDI E-TRON 50 (313ch) quattro	-6Â 000â,¬Â 1	0	356Â â,¬
6	BMW i3 120 Ah	-6Â 000â,¬Â 1	0	204Â â,¬

5 rows selected (0.381 seconds)
0: jdbc:hive2://localhost:10000>

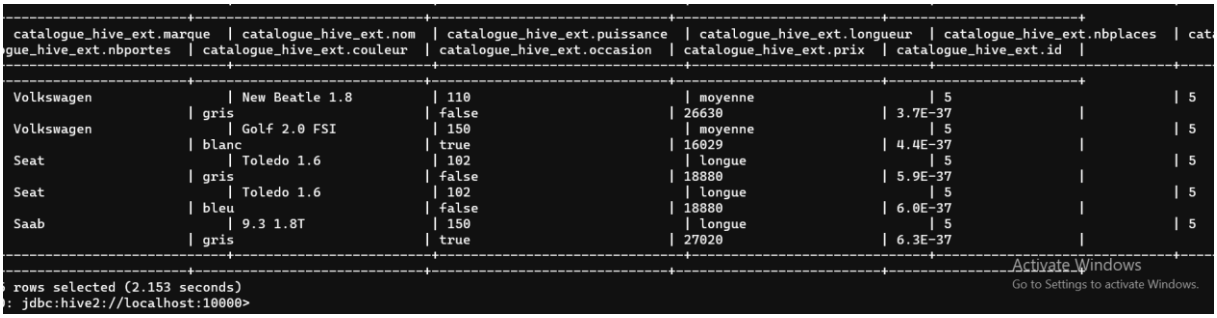
- ✓ **Table externe dans oracle NOSQL**

```
drop table catalogue_hive_ext;
CREATE EXTERNAL TABLE catalogue_hive_ext (
    marque string,
    nom string,
    puissance int,
    longueur string,
    nbPlaces int,
    nbPortes int,
    couleur string,
    occasion boolean,
    prix int,
    id int)
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" = "catalogue");
```

Requête :

```
select * from  catalogue_hive_ext limit 5;
```

Résultat



catalogue_hive_ext.marque	catalogue_hive_ext.nom	catalogue_hive_ext.puissance	catalogue_hive_ext.longueur	catalogue_hive_ext.nbplaces	catalogue_hive_ext.nbportes	catalogue_hive_ext.couleur	catalogue_hive_ext.occasion	catalogue_hive_ext.prix	catalogue_hive_ext.id
Volkswagen	New Beatle 1.8	110	moyenne	5	5	gris	false	26630	3.7E-37
Volkswagen	Golf 2.0 FSI	150	moyenne	5	5	blanc	true	16029	4.4E-37
Seat	Toledo 1.6	102	longue	5	5	gris	false	18880	5.9E-37
Seat	Toledo 1.6	102	longue	5	5	bleu	false	18880	6.0E-37
Saab	9.3 1.8T	150	longue	5	5	gris	true	27020	6.3E-37

rows selected (2.153 seconds)
: jdbc:hive2://localhost:10000>

Conclusion partielle

Dans cette partie, nous avons défini l'architecture de notre dataLake et effectué sa mise en place. Nous avons commencé à construire les bases de données distance et ensuite nous avons créé les tables externes pour y avoir accès. De plus, nous avons effectué des requêtes de consultation pour nous s'assurer que les données étaient bien présentes Dans la suite, nous allons utiliser un Jupiter notebook pour manipuler les données.

Chapitre 2 : TRAITEMENT DES DONNEES

Dans cette section nous allons nous connecter à notre dataLake grâce à python et effectuer les manipulations nécessaires pour le traitement des données.

I- Environnement de travaille

1- Installation de jupyter notebook

```
-- installation des dépendances pour pip

curl https://bootstrap.pypa.io/pip/3.6/get-pip.py -o get-pip.py

python3 get-pip.py --user

--pour installer Jupyter Notebook

~/.local/bin/pip install jupyter
```

Après l'installation, on lance jupyter Notebook avec la commande

```
jupyter notebook --ip=0.0.0.0
```

```
D:\TPMONGO\TP_VM\vagrant-projects\OracleDatabase\21.3.0>vagrant ssh

Welcome to Oracle Linux Server release 8.7 (GNU/Linux 5.15.0-5.76.5.1.el8uek.x86_64)

The Oracle Linux End-User License Agreement can be viewed here:

  * /usr/share/eula/eula.en_US

For additional packages, updates, documentation and community help, see:

  * https://yum.oracle.com/

Last login: Fri Mar 10 15:49:11 2023 from 10.0.2.2
[vagrant@oracle-21c-vagrant ~]$ jupyter notebook --ip=0.0.0.0
[I 16:18:03.453 NotebookApp] Serving notebooks from local directory: /home/vagrant
[I 16:18:03.453 NotebookApp] Jupyter Notebook 6.4.10 is running at:
[I 16:18:03.453 NotebookApp] http://oracle-21c-vagrant:8888/?token=d2481266a82a1118acae6c5a8b4669c7
[I 16:18:03.453 NotebookApp]  or http://127.0.0.1:8888/?token=d2481266a82a1118acae6c5a8b4669c70c739
[I 16:18:03.453 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to
[W 16:18:03.464 NotebookApp] No web browser found: could not locate runnable browser.
[C 16:18:03.465 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/vagrant/.local/share/jupyter/runtime/nbserver-13208-open.html
    Or copy and paste one of these URLs:
        http://oracle-21c-vagrant:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee
     or http://127.0.0.1:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee
```

Nous obtenons une url locale qui nous permet d'ouvrir jupyter notebook :
http://127.0.0.1:8888/?token=d2481266a82a1118acae6c5a8b4669c70c7390d94c2118ee

2- Connexion à HIVE et récupération des données

Notre objectif dans ce notebook est de traiter les données CO2 et effectuer leur intégration dans la table catalogue. Pour ce faire, nous effectuons une connexion à partir de HIVE pour charger les données de CO2 et les manipuler.

✓ Pyspark

PySpark est une bibliothèque Spark écrite en Python pour exécuter des applications Python en utilisant les capacités d'Apache Spark. PySpark permet d'exécuter des applications en parallèle sur un cluster distribué (plusieurs nœuds). En d'autres termes, PySpark est une API Python pour Apache Spark. Apache Spark est un moteur de traitement analytique pour le traitement de données distribuées puissantes à grande échelle et les applications d'apprentissage automatique.

Source : https://sparkbyexamples.com/pyspark-tutorial/

✓ PyHive

Pour la connexion à HIVE à partir de python, nous utilisons **pyhive.** C'est est une collection d'interfaces Python DB-API et SQLAlchemy pour Presto et Hive. Il permet d'interroger des bases de données à partir de Python.

```
from pyhive import hive
conn = hive.connect(host='localhost', port=10000)
cursor = conn.cursor()
```

Nous importons le module hive de pyhive et effectuons une connexion vers notre machine locale.

```
from pyhive import hive
conn = hive.connect(host='localhost', port=10000)
cursor = conn.cursor()

def import_data(table_name):
    cursor.execute(f"SELECT * FROM {table_name}")
    result = cursor.fetchall()
    data = pd.DataFrame(result, columns=[desc[0] for desc in cursor.description])
    # Column transformation
    col = data.columns
    col = col.str.replace(f'{table_name}.', '')
    data.columns = col
    data = spark.createDataFrame(data)

    return data
```

Nous définissons ensuite une fonction pour récupérer les données distances à partir de HIVE. La fonction prend le nom d'une table comme argument, effectue une sélection pour récupérer les données et transforme le résultat en dataframe de Spark.

On utilise cette fonction pour récupérer les données de **CO2** en passant en paramètre les le nom de la table externe**.**

```
co2 = import_data('co2_hive_ext')
```

```
co2.show()
```

Ensuite on affiche le résultat :

```
+---+--------------------+--------------+------------+-----------+
| id|       marque_modele|    bonus_malus| rejetsco2gkm|coutenergie|
+---+--------------------+--------------+------------+-----------+
|  2|AUDI E-TRON SPORT...|  -6Â 000â,¬Â 1|           0|   319Â â,¬|
|  3|AUDI E-TRON SPORT...|  -6Â 000â,¬Â 1|           0|   356Â â,¬|
|  4|AUDI E-TRON 55 (4...|  -6Â 000â,¬Â 1|           0|   357Â â,¬|
|  5|AUDI E-TRON 50 (3...|  -6Â 000â,¬Â 1|           0|   356Â â,¬|
|  6|        BMW i3 120 Ah|  -6Â 000â,¬Â 1|           0|   204Â â,¬|
|  7|       BMW i3s 120 Ah|  -6Â 000â,¬Â 1|           0|   204Â â,¬|
|  8|    CITROEN BERLINGO|  -6Â 000â,¬Â 1|           0|   203Â â,¬|
|  9|      CITROEN C-ZERO|  -6Â 000â,¬Â 1|           0|   491Â â,¬|
| 10|DS DS3 CROSSBACK ...|  -6Â 000â,¬Â 1|           0|   251Â â,¬|
| 11|HYUNDAI KONA elec...|  -6Â 000â,¬Â 1|           0|   205Â â,¬|
| 12|HYUNDAI KONA elec...|  -6Â 000â,¬Â 1|           0|   205Â â,¬|
| 13|JAGUAR I-PACE EV4...|  -6Â 000â,¬Â 1|           0|   271Â â,¬|
| 14|"KIA e-NIRO Moteu...| 150kW (204ch)"|-6Â 000â,¬Â 1|          0|
| 15|"KIA e-NIRO Moteu...| 100kW (136ch)"|-6Â 000â,¬Â 1|          0|
| 16|KIA SOUL Moteur Ã...|  -6Â 000â,¬Â 1|           0|   214Â â,¬|
| 17|KIA SOUL Moteur Ã...|  -6Â 000â,¬Â 1|           0|   214Â â,¬|
| 18|MERCEDES EQC 400 ...|  -6Â 000â,¬Â 1|           0|   291Â â,¬|
| 19|MERCEDES VITO Tou...|  -6Â 000â,¬Â 1|           0|   411Â â,¬|
| 20|MERCEDES VITO Tou...|  -6Â 000â,¬Â 1|           0|   411Â â,¬|
| 21|MINI MINI Cooper ...|  -6Â 000â,¬Â 1|           0|   199Â â,¬|
+---+--------------------+--------------+------------+-----------+
```

On remarque que la *marque et le modèle* sont dans la meme colonne, les valeurs de *bonus-malus* et de *cout énergie* ont des caractères qui ne sont pas interprétable.

II- Traitement des données

Pour le traitement des données nous avons envisager deux approches. La première consiste à utiliser pyspark pour manipuler les données directement et la deuxième consiste à utiliser un programme MapReduce.

1- Traitement directe des données

1-1 **Marque model**

Nous commençons par diviser la colonne marque_modele en deux.

```
split_col = split(co2['marque_modele'], ' ')
```

```
marque = split_col.getItem(0)
```

```
co2 = co2.withColumn('marque', marque)
```

Les marque étant les premières valeurs dans la colonne marque_modele, on effectue un split suivant l'espace et on récupère la première valeur d'indice 0. Le reste constitue le modèle. Nous obtenons ainsi deux colonnes *marque et model.*

On observe les valeurs distinctes de marques ceux de modèle

```
co2.select("marque").distinct().show()
```

```
+----------+
|    marque|
+----------+
|  MERCEDES|
|   PORSCHE|
|   HYUNDAI|
|    TOYOTA|
|     SKODA|
|    NISSAN|
|      LAND|
|   CITROEN|
|   BENTLEY|
|      AUDI|
|      MINI|
|   PEUGEOT|
|    JAGUAR|
|     VOLVO|
|     TESLA|
|       BMW|
|VOLKSWAGEN|
|       KIA|
|     SMART|
|   RENAULT|
+----------+
only showing top 20 rows
```

```
co2.select("model").show()
```

```
+--------------------+
|               model|
+--------------------+
|E-TRON SPORTBACK ...|
|E-TRON SPORTBACK ...|
|E-TRON 55 (408ch)...|
|E-TRON 50 (313ch)...|
|           i3 120 Ah|
|          i3s 120 Ah|
|            BERLINGO|
|              C-ZERO|
|DS3 CROSSBACK E-T...|
|KONA electric 64 kWh|
|KONA electric 39 kWh|
|I-PACE EV400 (400...|
|SOUL Moteur ÃƒÂ©l...|
|SOUL Moteur ÃƒÂ©l...|
|      EQC 400 4MATIC|
|VITO Tourer long ...|
|VITO Tourer extra...|
|MINI Cooper SE Ha...|
|Nouvelle NISSAN LEAF|
|Nouvelle NISSAN LEAF|
+--------------------+
only showing top 20 rows
```

Dans les deux cas, c'est juste les 20 premières lignes qui sont affichées.

1-2 **bonus-malus**

```
co2.select('bonus_malus').distinct().show()
```

```
+-------------+
|  bonus_malus|
+-------------+
|   +7Â 613â,¬|
|-6Â 000â,¬Â 1|
|   +7Â 890â,¬|
|   +7Â 340â,¬|
|   +6Â 810â,¬|
|            -|
|   +7Â 073â,¬|
|   +8Â 460â,¬|
|   +8Â 173â,¬|
|   +8Â 753â,¬|
+-------------+
```

Nous remarquons que les données de bonus-malus ont des caractères spéciaux qui ne sont pas interprétable. Au lieu de ça -6Â 000â,¬Â 1 nous devons plutôt avoir -6000£, de meme pour les autres. Cependant comme nous travaillons avec des données numériques, nous allons avoir **-6000.** On fera pareil pour les autres valeurs. De plus, nous avons des valeurs manquantes représenter par le symbole -, nous allons remplacer les valeurs manquantes par la valeur de bonus-malus la plus fréquente (**le mode**).

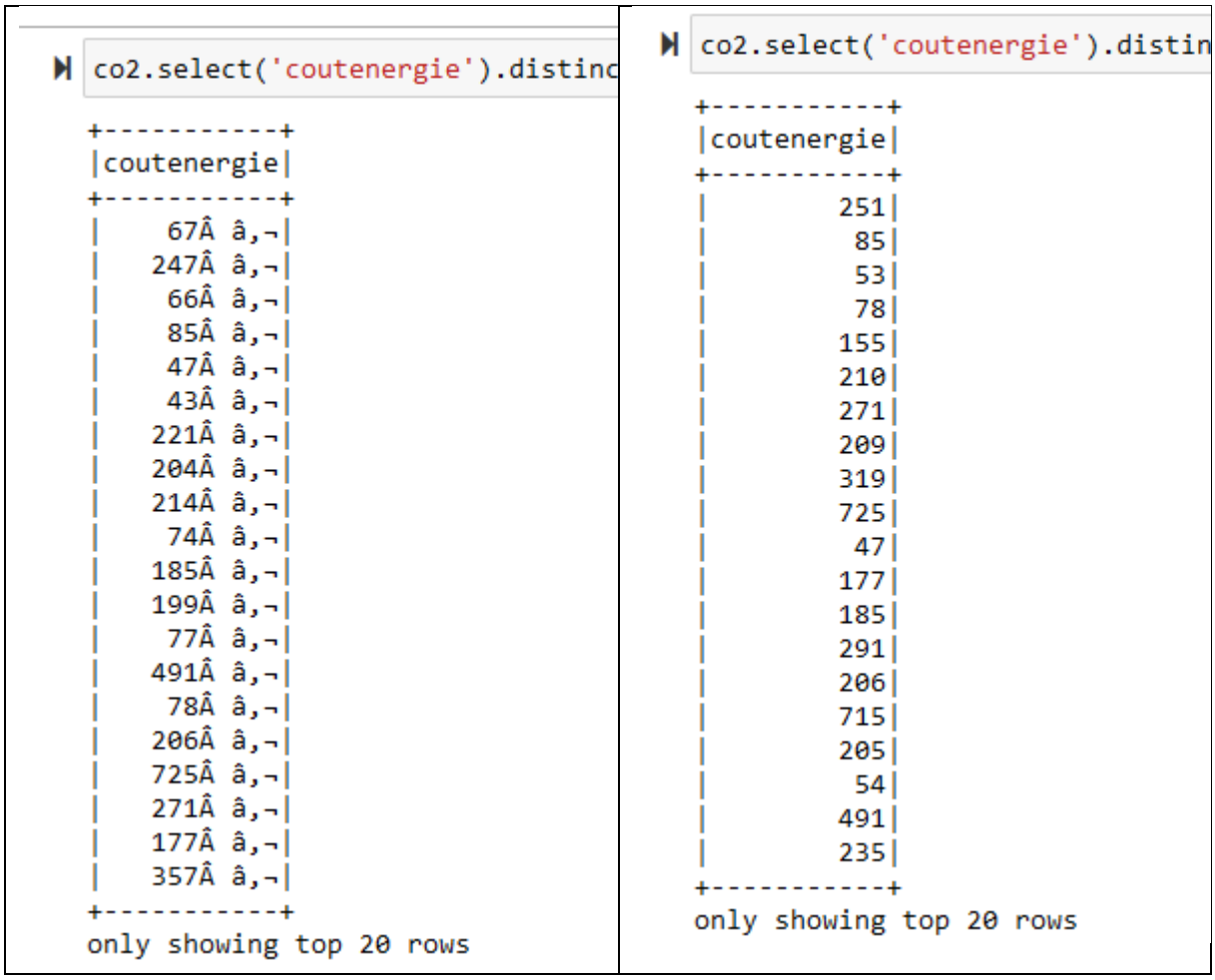
Ainsi après traitement des données erronées nous avons obtenus :

```
co2.select('bonus_malus').distinct().show()
```

```
+-----------+
|bonus_malus|
+-----------+
|       7340|
|       7613|
|      -6000|
|       6810|
|       7890|
|       8753|
|       7073|
|       8460|
|       8173|
+-----------+
```

Les valeurs de bonus-malus ont été traitées maintenant propres pour l'usage. ;
Nous faisons le traitement sur cout et énergie

Avant traitement	Après traitement



```
co2.select('coutenergie').distinc

+-----------+
|coutenergie|
+-----------+
|    67Â â,¬|
|   247Â â,¬|
|    66Â â,¬|
|    85Â â,¬|
|    47Â â,¬|
|    43Â â,¬|
|   221Â â,¬|
|   204Â â,¬|
|   214Â â,¬|
|    74Â â,¬|
|   185Â â,¬|
|   199Â â,¬|
|    77Â â,¬|
|   491Â â,¬|
|    78Â â,¬|
|   206Â â,¬|
|   725Â â,¬|
|   271Â â,¬|
|   177Â â,¬|
|   357Â â,¬|
+-----------+
only showing top 20 rows
```

```
co2.select('coutenergie').distin

+-----------+
|coutenergie|
+-----------+
|        251|
|         85|
|         53|
|         78|
|        155|
|        210|
|        271|
|        209|
|        319|
|        725|
|         47|
|        177|
|        185|
|        291|
|        206|
|        715|
|        205|
|         54|
|        491|
|        235|
+-----------+
only showing top 20 rows
```

2- Traitement des données avec un programme MapReduce

MapReduce est un modèle de programmation utilisé pour traiter de grandes quantités de données parallèlement sur des clusters de machines. Il divise le travail en deux étapes principales : mapper une fonction pour chaque élément de données, produisant une liste de paires clé-valeur, puis réduire les paires clé-valeur en une liste de résultats. Les opérations sont effectuées en parallèle sur différents nœuds du cluster, permettant de traiter rapidement de grandes quantités de données. MapReduce est utilisé dans des systèmes de traitement de données distribués tels que Hadoop et Apache Spark pour l'analyse de données, le traitement de logs, la recherche web, etc.

Source : https://www.ibm.com/topics/mapreduce

Dans notre cas, nous effectuons une fonction mapper.

```python
def mapper(line):
    line = tuple(line)

    # colonne marque
    marque =  line[1].split(" ")[0].replace("\"", "").capitalize() #replace quote near Volswagen
    # colonne Malus/Bonus
    if line[2] in [" 150kW (204ch)\"", " 100kW (136ch)\""]:
        return (None, None)
#     malus_bonus = line[2].strip().replace("¬Â\xa01", "").replace(r"[\d-]+", "").replace("€", "").replace("\"", "")
    malus_bonus = line[2].strip().replace("¬Â\xa01", "").replace(r"[\d-]+", "").replace("€", "").replace("\"", "")

    malus_bonus = ''.join(re.findall(r"[\d-]+", malus_bonus))

#     missing values bonus_malus
    malus_bonus = "0" if len(malus_bonus) == 1 else malus_bonus
    # colonne cout energie
    cout = line[-1]
    cout_splitted = cout.replace('Â', '').split()
    cout = cout_splitted[0] + cout_splitted[1] if len(cout_splitted) == 3 else cout_splitted[0] if len(cout_splitted) == 2 else
    cout = cout.encode('ascii', 'ignore').decode('ascii').replace(',', '')

    # colonne Rejet CO2
    rejet = line[3]

    malus_bonusInt = int(malus_bonus)
    rejetInt = (rejet)
    coutInt = int(cout)

    # on crée le couple key value avec la marque comme key
    new_value = f"{malus_bonusInt}|{rejetInt}|{coutInt}"
    return (marque, new_value)
```

Cette fonction ***mapper*** prend en entrée une ligne de données et applique un ensemble d'opérations pour extraire les informations nécessaires. Elle renvoie un couple (key, value) où la clé (key) est la marque de la voiture extraite à partir de la deuxième colonne, et la valeur (value) est une chaîne de caractères contenant le coût de l'énergie, le rejet de CO2, et le bonus/malus pour la voiture extraits des colonnes correspondantes. Si la colonne Malus/Bonus contient des valeurs spécifiques, la fonction renvoie None, None pour ignorer ces lignes MapReduce.

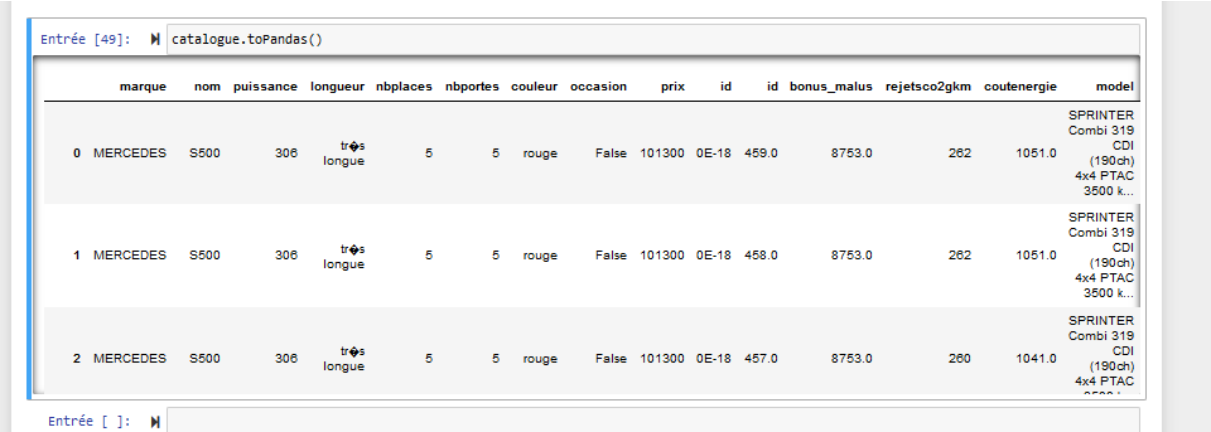
Après la phase de map nous devons effectuer une fonction reducer qui va mettre les pairs de clé ensemble pour reconstituer notre dataset de CO2. Cependant, dans notre approche, nous nous retrouvons avec une fonction map qui nous retourne le résultat que l'on cherche. Du coup, de façon exceptionnelle, nous n'allons pas effectuer de fonction reducer.

3- Jointure entre catalogue et CO2

Pour effectuer la jointure, nous importons le fichier catalogue à l'aide de la fonction *import_data* créer précédemment.

```python
catalogue = catalogue.join(co2, "marque", "left")
```

Nous avons fait un left join en gardant catalogue comme base. La jointure s'est fait sur la marque de véhicule.



```
Entrée [49]: catalogue.toPandas()
```

	marque	nom	puissance	longueur	nbplaces	nbportes	couleur	occasion	prix	id	id	bonus_malus	rejetsco2gkm	coutenergie	model
0	MERCEDES	S500	306	tr\u00e9s longue	5	5	rouge	False	101300	0E-18	459.0	8753.0	262	1051.0	SPRINTER Combi 319 CDI (190ch) 4x4 PTAC 3500 k...
1	MERCEDES	S500	306	tr\u00e9s longue	5	5	rouge	False	101300	0E-18	458.0	8753.0	262	1051.0	SPRINTER Combi 319 CDI (190ch) 4x4 PTAC 3500 k...
2	MERCEDES	S500	306	tr\u00e9s longue	5	5	rouge	False	101300	0E-18	457.0	8753.0	260	1041.0	SPRINTER Combi 319 CDI (190ch) 4x4 PTAC

```
Entrée [ ]:
```

Pour bien visualiser les données, nous avons utiliser le dataframe de pandas.

Conclusion partielle

Nous avons dans cette partie effectuer une connexion à HIVE à partir un notebook jupyter. Ensuite, nous avons importer les données et effectuer leur traitement suivant deux approches qui mènent aux meme résultat. Enfin, nous avons effectuer une jointure entre les données de CO2 provenant de HDFS et ceux de catalogue provenant de OracleNosql.

Conclusion générale

Nous avons dans un premier temps mis en place une architecture BigData pour la création de notre DataLake. Pour ce faire, nous avons créé des bases de données distances dont HDFS pour les fichiers, Mongodb et OracleNosql. Nous avons ensuite utilisé HIVE comme frontale pour créer des tables internes et externes pour manipuler les données physiques et virtuelles. Dans un second nous sommes connecté à notre DataLake à partir un notebook python, dans lequel, nous utilisé pyhive pour la connexion à hive et pyspark pour la manipulation et le traitement des données. De plus, nous avons join les fichiers CO2 et catalogue. Cette architecture, nous a permis de manipuler des données provenant de diverses sources à partir d'un seul frontal HIVE. Ce dataLake, sera utilisé après par les algorithmes de machines learning pour faire des analyses et des prédictions