TPA GROUPE 2

MASTER 2 MBDS – MIAGE
UNIVERSITE CÔTE D'AZUR

2022/2023

Table des matières

Table des illustrations

Description du projet

Ce projet a pour objectif de mettre en œuvre tous les acquis des modules de big data, data visualisation, Machine Learning et Hadoop Map Reduce.

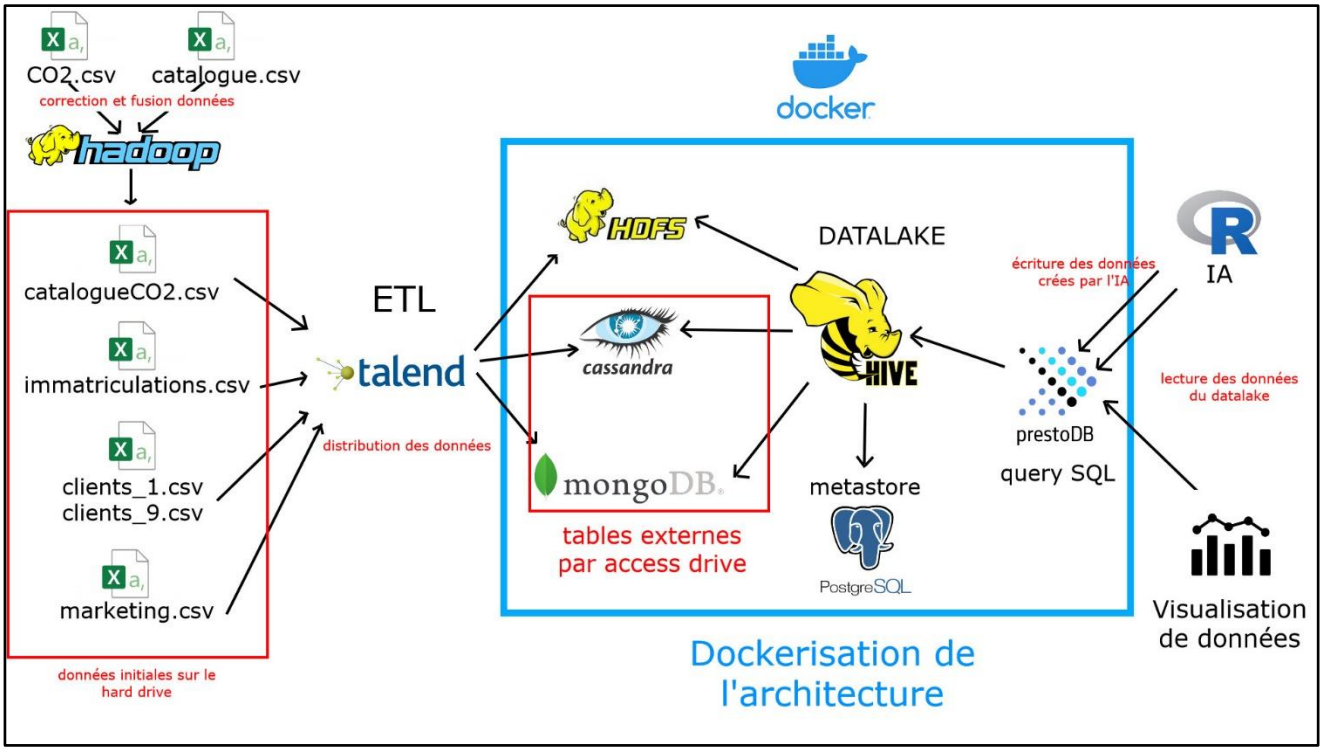
L'objectif du projet est d'aider notre "client", un concessionnaire automobile, à partir des données qu'il nous a fournies, à évaluer les catégories de véhicules susceptibles d'intéresser des clients et visualiser les données.

Pour ce faire et pour arriver à nos objectifs, nous devons effectuer plusieurs étapes qui correspondent à différents modules :

1) **Hadoop Map Reduce** : Le client nous a fourni un fichier CO2 qui contient des données supplémentaires à affecter au catalogue. Il faudra donc passer par Map Reduce pour nettoyer le fichier CO2 et adapter les données à Catalogue.
2) **Big Data** : A partir des données brutes, nous devons mettre en place des bases de données de différentes natures qui vont stocker les données, un lac de données (data Lake) central qui va pointer vers les bases par des tables externes, et un outil de requêtage SQL qui permettra ensuite d'effectuer des requêtes SQL sur les données centralisées. Pour envoyer les données sur les bases, il faudra utiliser un ELT (Extract Load and Transform) ou ETL (Extract Transform and Load).
3) **Machine Learning** : Avec les données que l'on peut récupérer grâce au moteur SQL, nous devrons ensuite créer un modèle de prédiction qui permet de prédire la catégorie de voiture la plus adaptée à un client à partir des informations données.
4) **Data Visualisation** : Avec toute l'architecture mise en place et les données au complet, il faudra effectuer des visualisations pertinentes pour le client et également pour le vendeur.

Architecture BDA/DL

Description de l'architecture



1 - Architecture BDA/DL

Le schéma ci-dessus représente une visualisation de toute notre pipeline que nous allons décrire en détail :

Nous avons au départ le fichier catalogueCO2 issu des manipulations Map Reduce (cf. Hadoop Map Reduce) et le reste des données brutes en csv se trouvant dans notre disque dur.

Ces données seront lues par notre ETL, qui va extraire les données, les uniformiser, régler les erreurs ou inconsistances éventuelles, puis distribuer ces données vers des bases différentes (cf. ETL pour plus d'informations).

Les bases de données que nous avons utilisées sont MongoDB et Cassandra.
Cassandra étant la base que nous avons choisie nous-mêmes car nous avions déjà l'habitude de l'utiliser grâce à un autre projet que notre groupe avait effectué l'année dernière. Les données sont aussi envoyées vers HDFS (Hadoop Data File System).

Les données sont distribuées de la façon suivante :

Cassandra : catalogueCO2, moitié de clients (clients_1), marketing

MongoDB : immatriculations et moitié de clients (clients_9)

HDFS : toutes les données

Une fois que les données ont été distribuées, il faut créer dans Hive les tables externes vers ces données en question.

L'ETL se charge de la création des tables dans les bases de données et aussi de la création des tables externes, issues de HDFS, dans Hive.
Pour la création des tables externes vers les données sur MongoDB et Cassandra, nous avons utilisé beeline et créé les tables à la main en passant par des handlers.

```
CREATE external TABLE marketingCassandra (
    age int,
    sexe string,
    taux int,
    situationFamiliale string,
    nbEnfantsAcharge int,
    deuxiemeVoiture boolean
)
STORED BY 'org.apache.hadoop.hive.cassandra.CassandraStorageHandler'
WITH SERDEPROPERTIES("cassandra.cf.name" = "marketing2","cassandra.host"="cassandra","cassandra.port" = "9160")
TBLPROPERTIES ( "cassandra.ks.name" = "tpa");

CREATE EXTERNAL TABLE clientsMongo
(
  id STRUCT<oid:STRING, bsontype:INT>,
  age string,
  sexe string,
  taux string,
  situationfamiliale string,
  nbenfantsacharge string,
  estdeuxiemevoiture string,
  immatriculation string
)
STORED BY 'com.mongodb.hadoop.hive.MongoStorageHandler'
WITH
SERDEPROPERTIES('mongo.columns.mapping'='{"id":"_id","age":"age","sexe":"sexe","taux":"taux","situationfamiliale":"situatio
nFamiliale","nbenfantsacharge":"nbEnfantsAcharge","estdeuxiemevoiture":"estDeuxiemeVoiture","immatriculation":"immatriculat
ion"}')
TBLPROPERTIES('mongo.uri'='mongodb://mongodb:27017/tpa-groupe2.clients');
```

2 - table externe marketing (Cassandra) et clients (MongoDB)

Après avoir mis en place la data Lake, il ne restera plus qu'à utiliser prestoDB qui va s'occuper d'effectuer des requêtes SQL vers Hive de la part de l'IA ou de la visualisation de données.
PrestoDB va également créer les nouvelles tables sur Hive qui correspondent aux nouvelles données issues des manipulations de l'IA, à savoir immatriculations et marketing avec une nouvelle colonne appelée "catégorie" prédite par l'IA.

Pour déployer toute cette architecture, nous avons décidé de la containeriser avec Docker.

Nous sommes partis d'un projet open source qui lançait un environnement Hive sur docker, et nous l'avons enrichi pour lancer aussi tous les autres éléments de notre propre architecture.

Cela nous permet à travers un docker-compose de pouvoir lancer toute l'architecture quel que soit l'environnement et sans se soucier de problèmes d'installation préalable.

Le docker lance Cassandra sur le port 9042, MongoDB sur le port 27017, Hadoop, Hive sur le port 10000, un Metastore pour Hive en PostgreSQL et prestodb sur le port 8080.

Le fichier docker-compose.yml nous permet donc de lancer facilement toute l'architecture BDA/DL sur n'importe quelle machine tant que docker est installé.

ETL

Pour rendre les données disponibles dans notre lac de données, nous avons utilisé l'ETL « Talend Open Studio For Big Data » version 7.3 de Talend.

Pourquoi Talend et pas un autre ETL du marché ?

- Logiciel Open Source gratuit existant depuis 2012
- Outil basé sur Eclipse
- Facile à prendre en main, interface graphique
- Multiplateforme (Linux, Windows & MacOS)
- Création de « Job[1] »
- Gestion de fichiers
- Contrôle et gestion des flux de données
- Exportation du job en JAR exécutable
- Système de contexte

Le système de contexte et le JAR exécutable permettent une exécution du job de l'ETL où que vous soyez. En effet, lorsque l'on exporte le Job, un JAR exécutable et un dossier « contexts » où se trouve un fichier « xx.properties » sont créés. Ce fichier contient les informations nécessaires pour se connecter aux bases de données et pour connaître la racine des répertoires contenant les fichiers CSV.

En d'autres termes, les contextes permettent de basculer aisément d'un environnement à un autre. Par exemple, changer de keyspace dans cassandra…

Comme expliqué précédemment, notre ETL fonctionne ainsi :

3 - Fonctionnement de l'ETL

L'ETL a été très utile pour « uniformiser » les données.
En effet, les CSV possèdent un grand nombre de données simulant « diverses sources ».

Prenons l'exemple du sexe d'un client qui peut avoir 10 valeurs différentes : Vide, « F », « Feminin », « Femme », « Féminin », « Homme », « H », « Masculin », « N/D ».

[1] Composants connectés permettant de définir, mettre en place et d'exécuter des processus de gestion de flux de données.

La situation familiale d'un client peut, quant à elle, avoir 12 valeurs :
Vide, « ? », « Celibataire », « Célibataire », « Divorcé », « Divorcée », « En couple », « Marie(e) », « Marié(e) », « N/D », « Seul » et « Seule ».

L'âge d'un client peut être non défini ou négatif…
Le nombre d'enfants à charge peut être vide, « ? » ou négatif …

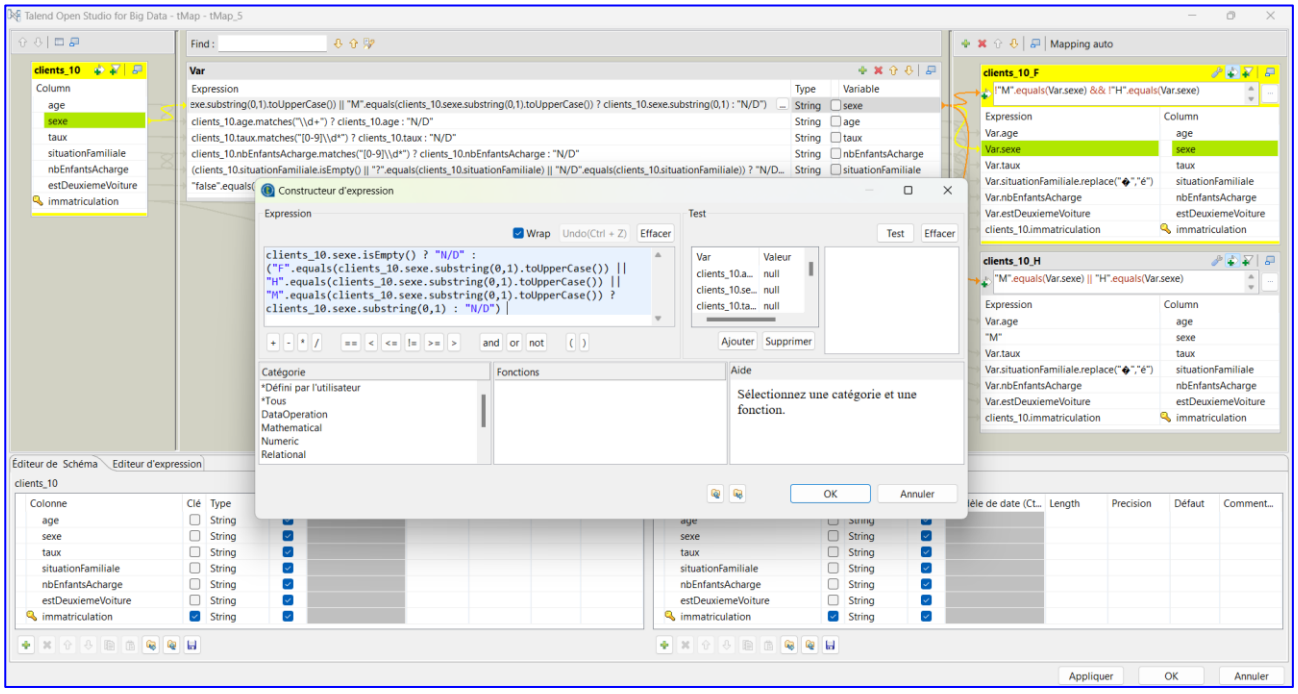
situationfamiliale
?
Celibataire
C?libataire
Divorcee
Divorc?e
En Couple
Marie(e)
Mari?(e)
N/D
Seul
Seule

sexe
?
F
Feminin
Femme
F?minin
Homme
M
Masculin
N/D

nbenfantsacharge
-1
0
1
2
3
4
?

4 - Diversité des valeurs pour un client

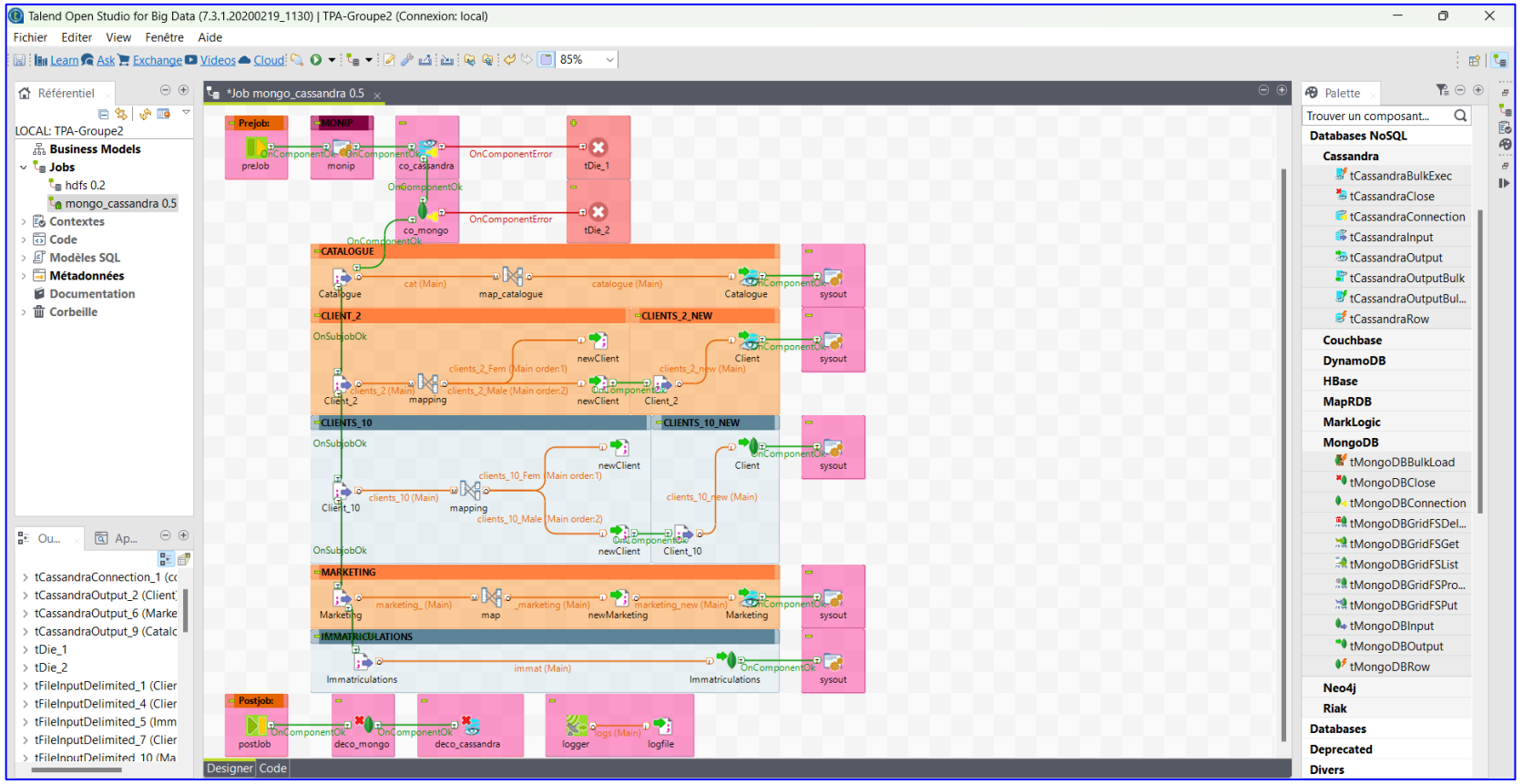
L'ETL va prendre les données du CSV en entrée puis va effectuer les modifications adéquates selon diverses conditions à travers un « tMap ». Grâce à des opérations ternaires, puis à un filtrage (voir image ci-dessous), nos anciens sexes de client deviennent soit « H », soit « F » soit « N/D ». Les âges et nombres d'enfants à charge négatifs, null (« ? ») ou encore vides deviennent « N/D ». Il en va de même pour la situation familiale qui est uniformisée. Les accents sont également réglés.



5 - Uniformisation des données via Talend

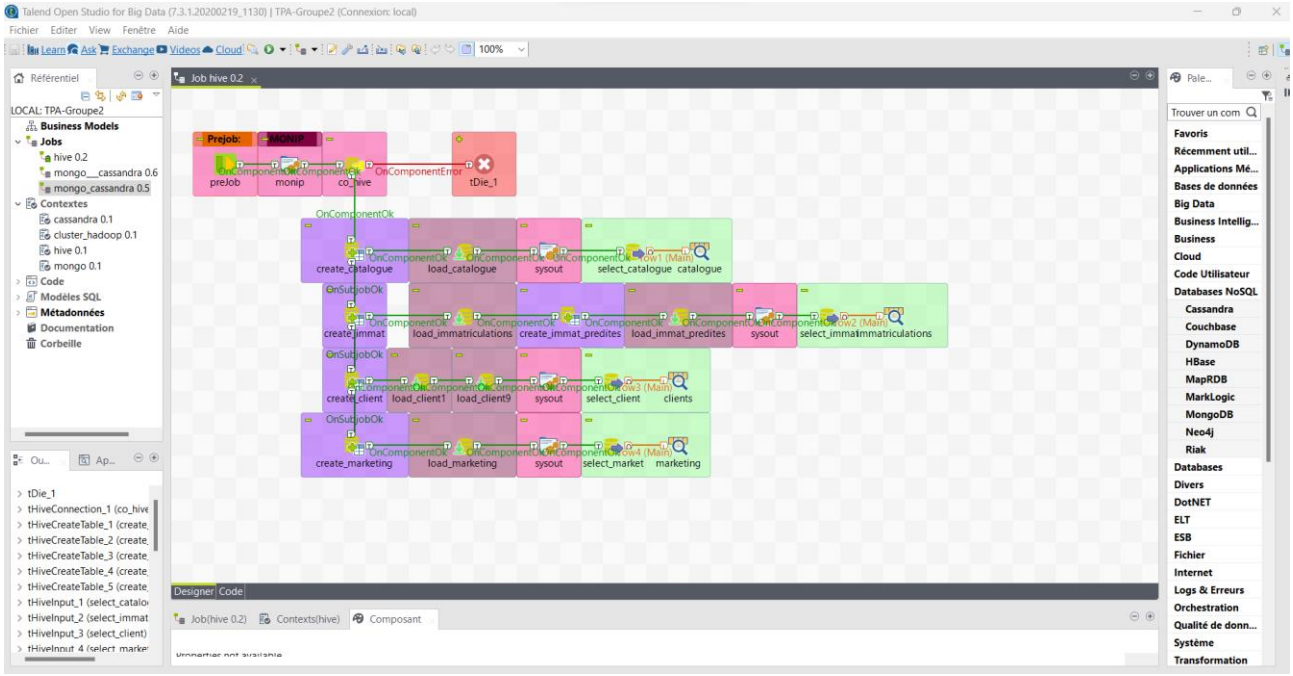
Le job Talend permettant l'insertion dans les bases MongoDB et Cassandra est montré illustration 6.

Tout d'abord, nous nous connectons aux bases de données puis nous lisons les CSV, nous faisons le nécessaire dans le tMap et nous écrivons dans la base.
Avec les variables de contextes définies actuellement, sont créés dans mongoDB, dans la base de données « tpa », les tables « clients » et « immatriculations » et sont créés dans Cassandra, dans le keyspace « tpa », les tables « catalogue », « clients », « marketing ».
Les éléments « sysout » affiche la progression du Job pour faire connaître l'avancement à l'utilisateur. Enfin, nous fermons les connexions aux bases de données.



6 – ETL – mongoDB/Cassandra

Le job Talend permettant l'écriture dans Hive ressemble donc à ceci :



7 – ETL - Hive

Nous nous connectons à Hive, nous créons les tables puis nous chargeons (load) les csv préalablement placés dans HDFS.

Docker

Tout notre data Lake est containerisé via docker-compose.yml. Nous retrouvons entre autres le Hive, le MongoDB, le Cassandra, le PrestoDB, le PostgreSQL…

Un Dockerfile est également présent pour toute la configuration Hadoop/hive.

Avoir "dockerisé" notre data Lake est un grand plus pour nous car cela facilite l'utilisation de notre solution. Cela signifie qu'il n'est pas nécessaire d'avoir un environnement déjà prêt (avec les bases de données préalablement installées). Seuls java, Docker et un interpréteur R sont nécessaires.

La structure de notre docker-compose.yml est donnée en Annexe.

Création du data Lake

Au préalable, il faut avoir défini deux variables d'environnements :

```
export MONIP=`hostname -I | awk '{print $1}'`;
export MONPATH=/mnt/d/TPA/2Data/Groupe_TPT_2/
```

Avec MONPATH[2] = la racine de l'emplacement des fichiers CSV.

[2] Il ne faut pas oublier le « / » à la fin de la valeur de MONPATH

Voici ce que donne la commande « tree » quand nous sommes dans le répertoire « MONPATH ». Il faut que vous ayez la même :

```
MONPATH
├── M2_DMA_Catalogue
│   └── Catalogue.csv
├── M2_DMA_Clients_10
│   └── Clients_9.csv
├── M2_DMA_Clients_2
│   └── Clients_1.csv
├── M2_DMA_Immatriculations
│   └── Immatriculations.csv
└── M2_DMA_Marketing
    └── Marketing.csv
```

Ensuite, nous pouvons exécuter le docker.sh qui créera les containeurs.

Un « docker container ls » permettra de s'assurer que tous les containeurs sont bien lancés.

Quand cela est correctement fait, nous obtenons cet affichage :

```
root@hugo: /mnt/d/TPA/tpa-  root@hugo: /mnt/d/TPA/tpa-ç
[+] Running 8/8
 ⠿ Container docker-hive-metastore-postgresql-1  Created                                    0.2s
 ⠿ Container docker-namenode-1                   Created                                    0.2s
 ⠿ Container cassandra                           Created                                    0.2s
 ⠿ Container docker-presto-coordinator-1         Created                                    0.2s
 ⠿ Container docker-hive-metastore-1             Created                                    0.2s
 ⠿ Container docker-hive-server-1                Created                                    0.2s
 ⠿ Container docker-datanode-1                   Created                                    0.2s
 ⠿ Container mongodb                             Created                                    0.2s
Attaching to cassandra, docker-datanode-1, docker-hive-metastore-1, docker-hive-metastore-postgresql-1, docker-hive-
server-1, docker-namenode-1, docker-presto-coordinator-1, mongodb
docker-hive-metastore-1             | Configuring core
docker-hive-metastore-1             |  - Setting hadoop.proxyuser.hue.hosts=*
docker-hive-metastore-1             |  - Setting fs.defaultFS=hdfs://namenode:8020
docker-hive-metastore-1             |  - Setting hadoop.proxyuser.hue.groups=*
docker-hive-metastore-1             |  - Setting hadoop.http.staticuser.user=root
docker-hive-metastore-1             | Configuring hdfs
docker-hive-metastore-1             |  - Setting dfs.namenode.datanode.registration.ip-hostname-check=false
docker-hive-metastore-1             |  - Setting dfs.permissions.enabled=false
docker-hive-metastore-1             |  - Setting dfs.webhdfs.enabled=true
docker-hive-metastore-1             | Configuring yarn
docker-hive-metastore-1             |  - Setting yarn.resourcemanager.fs.state-store.uri=/rmstate
docker-hive-metastore-1             |  - Setting yarn.timeline-service.generic-application-history.enabled=true
docker-hive-metastore-1             |  - Setting yarn.resourcemanager.recovery.enabled=true
```

8 - Lancement des containers docker

```
root@hugo: /mnt/d/TPA/tpa-ç  root@hugo: /mnt/d/TPA/tpa-
root@hugo:/mnt/d/TPA/tpa-groupe2# docker container ls
CONTAINER ID   IMAGE                                          COMMAND                  CREATED          STATUS
          PORTS                                                                                NAMES
8dde3b542a2d   mongo:latest                                   "docker-entrypoint.s…"   58 seconds ago   Up 56 sec
onds           0.0.0.0:27017->27017/tcp, :::27017->27017/tcp                                        mongodb
1ceacc3a060b   shawnzhu/prestodb:0.181                        "./bin/launcher run"     58 seconds ago   Up 55 sec
onds           0.0.0.0:8080->8080/tcp, :::8080->8080/tcp                                            docker-presto-coordi
nator-1
0f541e98143c   bde2020/hive:2.3.2-postgresql-metastore        "entrypoint.sh /bin/…"   58 seconds ago   Up 55 sec
onds           0.0.0.0:10000->10000/tcp, :::10000->10000/tcp, 10002/tcp                             docker-hive-server-1
3913a062f7ec   bde2020/hive:2.3.2-postgresql-metastore        "entrypoint.sh /opt/…"   58 seconds ago   Up 56 sec
onds           10000/tcp, 0.0.0.0:9083->9083/tcp, :::9083->9083/tcp, 10002/tcp                      docker-hive-metastor
e-1
fd309ed4af25   bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8   "/entrypoint.sh /run…"   58 seconds ago   Up 55 sec
onds (healthy)   0.0.0.0:50070->50070/tcp, :::50070->50070/tcp                                      docker-namenode-1
4176c98e9cdb   bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8   "/entrypoint.sh /run…"   58 seconds ago   Up 53 sec
onds (healthy)   0.0.0.0:50075->50075/tcp, :::50075->50075/tcp                                      docker-datanode-1
615731e06eed   bde2020/hive-metastore-postgresql:2.3.0        "/docker-entrypoint.…"   58 seconds ago   Up 56 sec
onds           5432/tcp                                                                             docker-hive-metastor
e-postgresql-1
a2a4bd60ca98   cassandra:3.0.8                                "/docker-entrypoint.…"   58 seconds ago   Up 55 sec
onds           7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp, :::9042->9042/tcp   cassandra
root@hugo:/mnt/d/TPA/tpa-groupe2#
```

9 - liste des containers lancés

A présent, pour créer les tables dans MongoDB et Cassandra, puis y écrire les données transformées via Talend, il faut exécuter le script shell « mongo_cassandra_run.sh » présent dans le répertoire « etl ».

Pour placer les fichiers dans HDFS, on utilise les commandes suivantes :

```
docker cp ${MONPATH}/M2_DMA_Immatriculations/Immatriculations.csv $(docker ps -aqf "name=hive-server"):/Immatriculations.csv;
docker cp ${MONPATH}/M2_DMA_Immatriculations/Immatriculations_predites.csv $(docker ps -aqf "name=hive-server"):/Immatriculations_predites.csv;
docker cp ${MONPATH}/M2_DMA_Catalogue/Catalogue_new.csv $(docker ps -aqf "name=hive-server"):/Catalogue_new.csv;
docker cp ${MONPATH}/M2_DMA_Clients_2/Clients_1_new.csv $(docker ps -aqf "name=hive-server"):/Clients_1_new.csv;
docker cp ${MONPATH}/M2_DMA_Clients_10/Clients_9_new.csv $(docker ps -aqf "name=hive-server"):/Clients_9_new.csv;
docker cp ${MONPATH}/M2_DMA_Marketing/Marketing_new.csv $(docker ps -aqf "name=hive-server"):/Marketing_new.csv;
```

Enfin, pour écrire dans Hive, il faut exécuter le script shell « hive_run.sh » présent dans le répertoire « etl » après avoir exécuté les commandes suscitées.

Vous obtiendrez le résultat visible sur l'image 11 « Création du data Lake » page suivante.

Vous accéderez à Hive en faisant :

```
docker compose exec hive-server bash
```

Puis une fois dans le terminal du container « hive-server », faites :

```
/opt/hive/bin/beeline -u jdbc:hive2://localhost:10000
```

Là, vous apercevrez vos tables et vous pourrez les requêter.
Un exemple de requêtes pour obtenir les différentes catégories de véhicules et les différentes situations familiales de clients est donné ci-dessous :

```
root@hugo:/mnt/d/TPA/tpa-groupe2/docker# docker compose exec hive-server bash
root@8a9eb9cd63de:/opt# /opt/hive/bin/beeline -u jdbc:hive2://localhost:10000
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop-2.7.4/share/hadoop/common/lib/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Connecting to jdbc:hive2://localhost:10000
Connected to: Apache Hive (version 2.3.2)
Driver: Hive JDBC (version 2.3.2)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 2.3.2 by Apache Hive
0: jdbc:hive2://localhost:10000> select distinct categorie from immatriculations_predites;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1
.X releases.
+------------+
| categorie  |
+------------+
| berline    |
| citadine   |
| familiale  |
| monospace  |
| routiere   |
+------------+
5 rows selected (3.717 seconds)
0: jdbc:hive2://localhost:10000> select distinct situationFamiliale from clients;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1
.X releases.
+---------------------+
| situationfamiliale  |
+---------------------+
| Celibataire         |
| Divorcee            |
| En Couple           |
| Marie(e)            |
| N/D                 |
+---------------------+
5 rows selected (1.616 seconds)
0: jdbc:hive2://localhost:10000>
```

10 - Requêtes dans Hive

```
root@hugo:/mnt/d/TPA/tpa-groupe2# export MONPATH=/mnt/d/TPA/2Data/Groupe_TPT_2/
root@hugo:/mnt/d/TPA/tpa-groupe2# export MONIP=`hostname -I | awk '{print $1}'`;
root@hugo:/mnt/d/TPA/tpa-groupe2# sh etl/mongo_cassandra_run.sh
ip où tournent les databases: 172.21.167.213
chemin où sont les csv: /mnt/d/TPA/2Data/Groupe_TPT_2/
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.datastax.shaded.netty.util.internal.PlatformDependent0 (file:/mn
t/d/TPA/tpa-groupe2/etl/lib/cassandra-driver-core-3.0.0-shaded.jar) to field java.nio.Buffer.address
WARNING: Please consider reporting this to the maintainers of com.datastax.shaded.netty.util.internal.Plat
formDependent0
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
catalogue référencé dans cassandra
client_2 référencé dans cassandra
client_10 référencé dans mongo
marketing référencé dans cassandra
immatriculations référencées dans mongo
root@hugo:/mnt/d/TPA/tpa-groupe2# docker cp ${MONPATH}/M2_DMA_Marketing/Marketing_new.csv $(docker ps -aqf
 "name=hive-server"):/Marketing_new.csv;
Successfully copied 4.096kB to 0f541e98143c:/Marketing_new.csv
root@hugo:/mnt/d/TPA/tpa-groupe2# docker cp ${MONPATH}/M2_DMA_Clients_10/Clients_9_new.csv $(docker ps -aq
f "name=hive-server"):/Clients_9_new.csv;
Successfully copied 7.724MB to 0f541e98143c:/Clients_9_new.csv
root@hugo:/mnt/d/TPA/tpa-groupe2# docker cp ${MONPATH}/M2_DMA_Clients_2/Clients_1_new.csv $(docker ps -aqf
 "name=hive-server"):/Clients_1_new.csv;
Successfully copied 3.863MB to 0f541e98143c:/Clients_1_new.csv
root@hugo:/mnt/d/TPA/tpa-groupe2# docker cp ${MONPATH}/M2_DMA_Catalogue/Catalogue_new.csv $(docker ps -aqf
 "name=hive-server"):/Catalogue_new.csv;
Successfully copied 25.09kB to 0f541e98143c:/Catalogue_new.csv
root@hugo:/mnt/d/TPA/tpa-groupe2# docker cp ${MONPATH}/M2_DMA_Immatriculations/Immatriculations_predites.c
sv $(docker ps -aqf "name=hive-server"):/Immatriculations_predites.csv;
Successfully copied 140.5MB to 0f541e98143c:/Immatriculations_predites.csv
root@hugo:/mnt/d/TPA/tpa-groupe2# docker cp ${MONPATH}/M2_DMA_Immatriculations/Immatriculations.csv $(dock
er ps -aqf "name=hive-server"):/Immatriculations.csv;
Successfully copied 121MB to 0f541e98143c:/Immatriculations.csv
root@hugo:/mnt/d/TPA/tpa-groupe2# sh etl/hive_run.sh
catalogue référencé dans hive
.-----.
|catalogue|
|=---=|
|count|
|=---=|
|270  |
'-----'

immatriculations référencées dans hive
immatriculations_predites référencées dans hive
.-------.
|immatriculations|
|=-----=|
|count  |
|=-----=|
|2000000|
'-------'

clients référencés dans hive
.------.
|clients|
|=----=|
|count |
|=----=|
|300000|
'------'

Marketing référencé dans hive
.-----.
|marketing|
|=---=|
|count|
|=---=|
|80   |
'-----'

root@hugo:/mnt/d/TPA/tpa-groupe2#
```

11 - Création du data Lake

Hadoop Map Reduce

L'objectif de cette partie du projet est de corriger les erreurs et adapter le fichier CO2.csv pour l'intégrer dans la table Catalogue.

Pour ce faire, Il faut analyser les fichiers Catalogue.csv et CO2.csv.

Analyse des données

La première réflexion que nous devons faire, c'est comment intégrer CO2.csv dans la table Catalogue ?
Pour répondre à cette question, nous allons analyser les deux fichiers.

Fichier Catalogue :

```
marque,nom,puissance,longueur,nbPlaces,nbPortes,couleur,occasion,prix
Volvo,S80 T6,272,très longue,5,5,blanc,false,50500
Volvo,S80 T6,272,très longue,5,5,noir,false,50500
Volvo,S80 T6,272,très longue,5,5,rouge,false,50500
Volvo,S80 T6,272,très longue,5,5,gris,true,35350
Volvo,S80 T6,272,très longue,5,5,bleu,true,35350
Volvo,S80 T6,272,très longue,5,5,gris,false,50500
Volvo,S80 T6,272,très longue,5,5,bleu,false,50500
Volvo,S80 T6,272,très longue,5,5,rouge,true,35350
Volvo,S80 T6,272,très longue,5,5,blanc,true,35350
Volvo,S80 T6,272,très longue,5,5,noir,true,35350
Volkswagen,Touran 2.0 FSI,150,longue,7,5,rouge,false,27340
Volkswagen,Touran 2.0 FSI,150,longue,7,5,gris,true,19138
Volkswagen,Touran 2.0 FSI,150,longue,7,5,bleu,true,19138
Volkswagen,Touran 2.0 FSI,150,longue,7,5,gris,false,27340
Volkswagen,Touran 2.0 FSI,150,longue,7,5,bleu,false,27340
Volkswagen,Touran 2.0 FSI,150,longue,7,5,blanc,true,19130
Volkswagen,Touran 2.0 FSI,150,longue,7,5,noir,true,19138
Volkswagen,Touran 2.0 FSI,150,longue,7,5,rouge,true,19130
Volkswagen,Touran 2.0 FSI,150,longue,7,5,blanc,false,27340
Volkswagen,Touran 2.0 FSI,150,longue,7,5,noir,false,27340
Volkswagen,Polo 1.2 6V,55,courte,5,3,blanc,true,8540
Volkswagen,Polo 1.2 6V,55,courte,5,3,blanc,false,12200
Volkswagen,Polo 1.2 6V,55,courte,5,3,noir,false,12200
Volkswagen,Polo 1.2 6V,55,courte,5,3,noir,true,8540
Volkswagen,Polo 1.2 6V,55,courte,5,3,bleu,true,8540
Volkswagen,Polo 1.2 6V,55,courte,5,3,bleu,false,12200
Volkswagen,Polo 1.2 6V,55,courte,5,3,rouge,true,8540
Volkswagen,Polo 1.2 6V,55,courte,5,3,rouge,false,12200
Volkswagen,Polo 1.2 6V,55,courte,5,3,gris,false,12200
Volkswagen,Polo 1.2 6V,55,courte,5,3,gris,true,8540
```

12 - fichier catalogue.csv

Dans le fichier Catalogue, nous avons les attributs marque, nom, puissance, longueur, nbPlaces, nbPortes, couleur, occasion et prix.

Le fichier ne présente aucune erreur.
Le lien entre le fichier CO2 et Catalogue a été fait via l'attribut « marque ».

Fichier CO2 :

```
,Marque / Modele,Bonus / Malus,Rejets CO2 g/km,Cout enerie
2,AUDI E-TRON SPORTBACK 55 (408ch) quattro,-6 000€ 1,0,319 €
3,AUDI E-TRON SPORTBACK 50 (313ch) quattro,-6 000€ 1,0,356 €
4,AUDI E-TRON 55 (408ch) quattro,-6 000€ 1,0,357 €
5,AUDI E-TRON 50 (313ch) quattro,-6 000€ 1,0,356 €
6,BMW i3 120 Ah,-6 000€ 1,0,204 €
7,BMW i3s 120 Ah,-6 000€ 1,0,204 €
8,CITROEN BERLINGO,-6 000€ 1,0,203 €
9,CITROEN C-ZERO,-6 000€ 1,0,491 €
10,DS DS3 CROSSBACK E-Tense (136ch),-6 000€ 1,0,251 €
11,HYUNDAI KONA electric 64 kWh,-6 000€ 1,0,205 €
12,HYUNDAI KONA electric 39 kWh,-6 000€ 1,0,205 €
13,JAGUAR I-PACE EV400 (400 CEE) - Automatique - 4 roues motrices,-6 000€ 1,0,271 €
14,"KIA e-NIRO Moteur Ã©lectrique synchrone Ã  aimant permanent, 150kW (204ch)",-6 000€ 1,0,212 €
15,"KIA e-NIRO Moteur Ã©lectrique synchrone Ã  aimant permanent, 100kW (136ch)",-6 000€ 1,0,203 €
16,KIA SOUL Moteur Ã©lectrique 150 kW (204ch),-6 000€ 1,0,214 €
17,KIA SOUL Moteur Ã©lectrique 100kW (136ch),-6 000€ 1,0,214 €
18,MERCEDES EQC 400 4MATIC,-6 000€ 1,0,291 €
19,MERCEDES VITO Tourer long eVITO (85kW) Traction 4x2 PTAC 3200kg,-6 000€ 1,0,411 €
20,MERCEDES VITO Tourer extra-long eVITO (85kW) Traction 4x2 PTAC 3200kg,-6 000€ 1,0,411 €
21,MINI MINI Cooper SE Hatch 3P (F56),-6 000€ 1,0,199 €
23,NISSAN Nouvelle NISSAN LEAF,-6 000€ 1,0,155 €
24,NISSAN Nouvelle NISSAN LEAF,-6 000€ 1,0,177 €
25,PEUGEOT 208 E- Tense (136 ch),-6 000€ 1,0,221 €
26,PEUGEOT ION electrique (67ch),-6 000€ 1,0,241 €
27,PEUGEOT PARTNER,-6 000€ 1,0,203 €
28,RENAULT ZOE (43Kw),-6 000€ 1,0,206 €
29,RENAULT ZOE (51Kw),-6 000€ 1,0,206 €
30,RENAULT ZOE (53Kw),-6 000€ 1,0,206 €
31,SKODA CITIGOe iV,-6 000€ 1,0,210 €
32,SMART EQ FORFOUR 22 kW Mode 3,-6 000€ 1,0,175 €
33,SMART EQ FORFOUR,-6 000€ 1,0,175 €
34,SMART EQ FORFOUR 22 kW Mode 2,-6 000€ 1,0,213 €
35,SMART EQ FORFOUR 7 kW Mode 2,-6 000€ 1,0,213 €
36,SMART EQ FORTWO COUPE,-6 000€ 1,0,175 €
37,SMART EQ FORTWO COUPE 22 kW Mode 2,-6 000€ 1,0,201 €
```

13 - Fichier C02.csv

Nous pouvons remarquer qu'il y a plusieurs erreurs dans ce fichier notamment :

- L'ID qui a été enlevé de l'entête.
- Le fichier qui possède des caractères mal encodés car il n'est pas en UTF-8.
- Le calcul de la moyenne du Bonus/Malus, du rejet en CO2 et du coût d'énergie de chaque marque qui est compliqué à faire étant donné que les valeurs sont sous formes de chaine de caractères. Il y a également des valeurs telles que « -6 000€ » et « - »
- Les marques sont en majuscules alors qu'elles étaient en minuscules dans le fichier Catalogue.
- La marque et le modèle sont dans deux colonnes séparées sur le fichier catalogue contrairement au fichier CO2 où ils sont fusionnés dans une même colonne.
- CO2 contient plus d'informations que catalogue.

Après avoir analysé les fichiers donnés, nous allons adapter le fichier CO2 pour intégrer ses informations complémentaires dans catalogue.

Correction des erreurs du fichier CO2

Il faut tout d'abord corriger les erreurs dans CO2.csv avant de fusionner ce dernier avec catalogue.

Pour ce faire, nous avons appliqué la fonction replaceAll sur les trois colonnes bonus/malus, rejets co2 et coût énergie afin de les transformer en entiers pour faire les calculs.

Les replaceAll utilisés vont permettre d'enlever les espaces des milliers, ainsi que le signe d'euro.
Malgré la suppression des espaces et de l'euro, nous avons toujours des valeurs erronées. En effet, la colonne Bonus/Malus contient parfois des « –600€ 1 » à la place de –6000. Nous avons finalement remplacé « - 6 000€ 1 » par « -6000 ».
Il faut également noter que les colonnes de ce fichier peuvent contenir des valeurs « - » que nous avons remplacées par 0.

Comme nous l'avons mentionné ci-dessus, la marque et le modèle étaient fusionnés dans une même colonne. L'objectif était donc de récupérer la marque.

Prenons « MINI MINI Cooper SE Hatch 3P (F56) » comme exemple. La marque se trouve à la première position, mais comment pouvoir la récupérer étant donné que dans certains cas il y a des espaces dans le nom de la marque ?

La seule solution était d'utiliser un fichier (marques.csv) contenant toutes les marques de voitures que nous allons stocker dans une liste. Grâce à cette dernière, nous avons pu finalement récupérer la marque (en comparant la liste générée et le premier mot de la colonne « marque, modèle »).

Après avoir fait ces corrections, il est maintenant possible de calculer les moyennes de chaque marque.

Map/Reduce

La clé choisie est donc la marque, et la valeur est une chaîne de caractères contenant « Bonus/Malus », « rejet CO2 g/km » et « coût en énergie », séparés par des virgules.

Dans le reduce nous allons donc boucler sur les valeurs tout en faisant un split ayant pour délimiteur une virgule pour pouvoir récupérer les trois valeurs des attributs qui nous intéressent.

Nous avons donc calculé les moyennes de chaque attribut pour chacune des marques présentes dans CO2.csv.

L'objectif maintenant est de mettre ces valeurs dans Catalogue. Ce qui n'était pas difficile à faire, nous avons simplement à comparer la marque et écrire à la suite de la ligne les données que nous avons calculées.

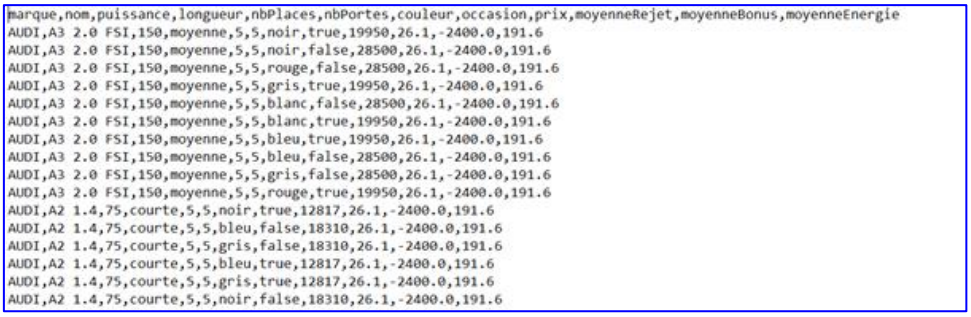
Pour les marques de voitures qui ne sont pas dans le fichier CO2.csv, nous avons inséré la moyenne de coût d'énergie (de même pour les autres colonnes) de toutes les marques de véhicules qui sont présents des deux côtés. Pour ce faire, nous devons parcourir tout le fichier pour calculer la moyenne de toutes les moyennes. Pour atteindre cet objectif, nous avons donc stocké toutes les moyennes dans un fichier temporaire.

Récupération du nouveau fichier Catalogue

Le nouveau fichier catalogue se trouve sur HDFS. Pour l'avoir directement en local, nous avons exécuté la commande « hdfs dfs -copyToLocal /finals . » via le programme Java.

Nous allons ensuite supprimer le dossier qui est sur HDFS puisqu'il ne sera plus utilisé étant donné qu'il est présent en local.

Nous allons exécuter avec java, la commande « hdfs dfs -rm -r /finals »



```
marque,nom,puissance,longueur,nbPlaces,nbPortes,couleur,occasion,prix,moyenneRejet,moyenneBonus,moyenneEnergie
AUDI,A3 2.0 FSI,150,moyenne,5,5,noir,true,19950,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,noir,false,28500,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,rouge,false,28500,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,gris,true,19950,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,blanc,false,28500,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,blanc,true,19950,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,bleu,true,19950,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,bleu,false,28500,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,gris,false,28500,26.1,-2400.0,191.6
AUDI,A3 2.0 FSI,150,moyenne,5,5,rouge,true,19950,26.1,-2400.0,191.6
AUDI,A2 1.4,75,courte,5,5,noir,true,12817,26.1,-2400.0,191.6
AUDI,A2 1.4,75,courte,5,5,bleu,false,18310,26.1,-2400.0,191.6
AUDI,A2 1.4,75,courte,5,5,gris,false,18310,26.1,-2400.0,191.6
AUDI,A2 1.4,75,courte,5,5,bleu,true,12817,26.1,-2400.0,191.6
AUDI,A2 1.4,75,courte,5,5,gris,true,12817,26.1,-2400.0,191.6
AUDI,A2 1.4,75,courte,5,5,noir,false,18310,26.1,-2400.0,191.6
```

14 - Résultats du nouveau fichier Catalogue

L'image ci-dessus représente le résultat final du fichier catalogue après avoir intégré les informations complémentaires.

Exécution du programme

Il faut préalablement télécharger le fichier marques.csv en plus du fichier CO2 et Catalogue.

Pour exécuter le programme il faut faire la commande :

```
hadoop jar tpa_hadoop.jar org.mbds.hadoop.tp.WCatalogue
VOTRE_PATH]/marques.csv [VOTRE_PATH]/Catalogue.csv [VOTRE_PATH]/CO2.csv
/finals
```

Il faut noter qu'un message d'erreur sera affiché si le nombre d'arguments est invalide.

Visualisation des données

Les utilisateurs visés

- Vendeur : C'est un concessionnaire qui propose des voitures à vendre.
- Prospect/ Client : Un prospect est un client potentiel ayant des besoins qui correspondent à des voitures que le vendeur propose.

Tâches utilisateurs

Utilisateur	Tâches utilisateur
Vendeur	Faire acheter des voitures à des prospects ou des clients et évaluer le type de véhicule le plus susceptible d'intéresser ses clients.
Client	Cet utilisateur est susceptible d'acheter une voiture, il veut identifier le type de véhicule le plus adapté à ses besoins.

Objectifs de visualisation

- Vendeur :
  - ✓ Il va visualiser les catégories des voitures les plus et les moins vendues.
  - ✓ Il va visualiser les catégories de voitures vendues par rapport aux profils des clients (selon sa situation familiale).
  - ✓ Il va présenter la répartition des émissions CO2 des voitures de son catalogue pour attirer les clients les plus sensibles à l'écologie.
- Prospect/Client :
  - ✓ Il va visualiser les voitures achetées par des profils similaires aux siens.

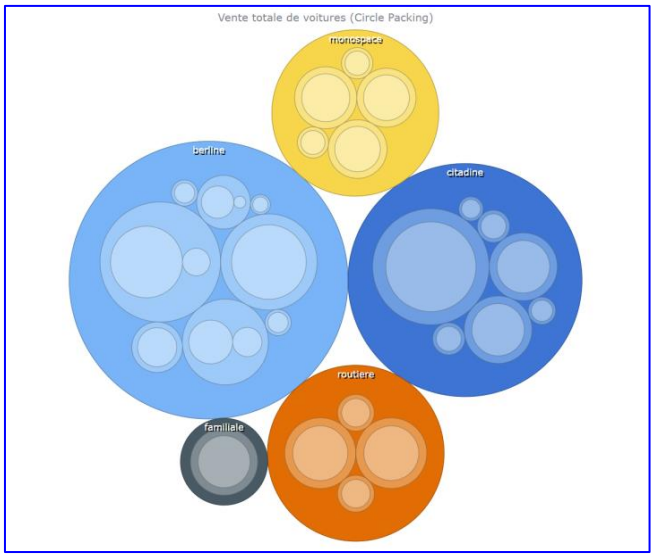
Visualisation pipeline et techniques de visualisation

1er graphe :

Pour répondre au premier objectif du vendeur (visualiser les catégories de voitures les plus/moins vendues), nous avons décidé d'utiliser la technique « Circle Packing » : https://observablehq.com/@d3/pack

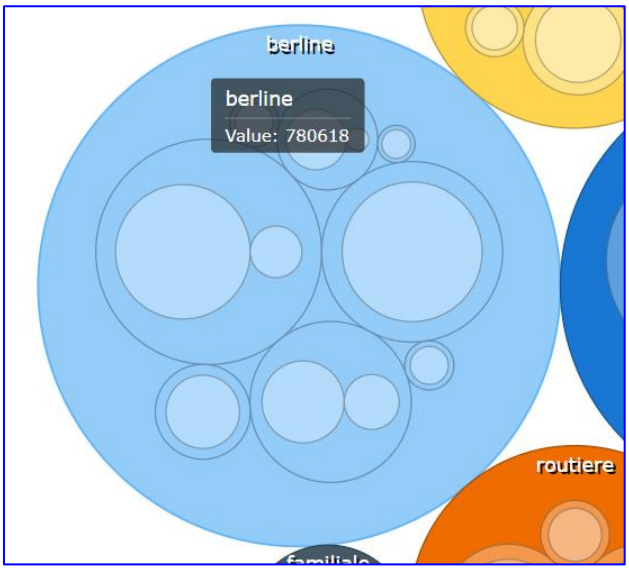
Pour ce faire, nous avons utilisé la table immatriculation qui contient des informations sur les voitures vendues cette année pour faire la transformation des données.

En tenant compte des attributs (Marque, Puissance, Longueur, Nb places, Nb portes, prix), nous avons identifié des catégories de voitures (citadine, routière, familiale…). Ces dernières seront définies grâce au Machine Learning.
Puis, nous nous sommes servis de cette table pour compter le nombre de ventes de chaque catégorie de voitures.
De ce fait, nous avons une nouvelle table, contenant deux colonnes (la catégorie et le nombre de ventes), que nous avons utilisé pour faire le graphe.

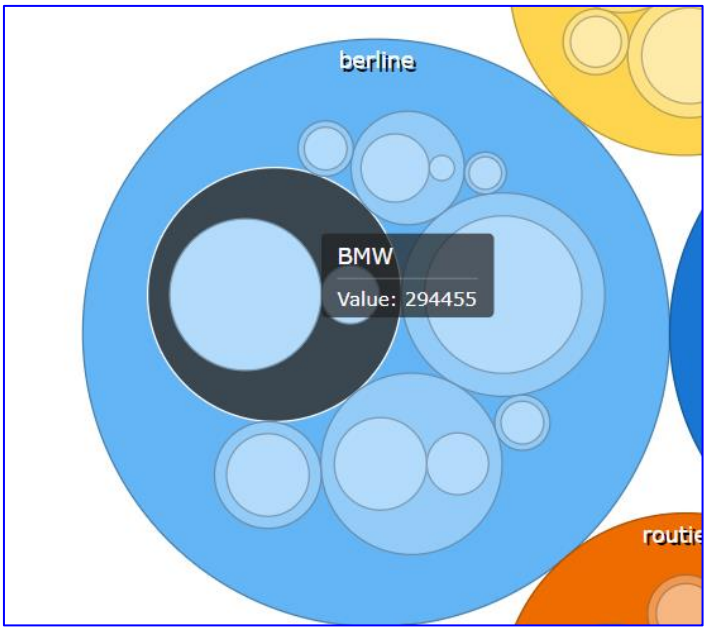


15 - Vente totale de voiture

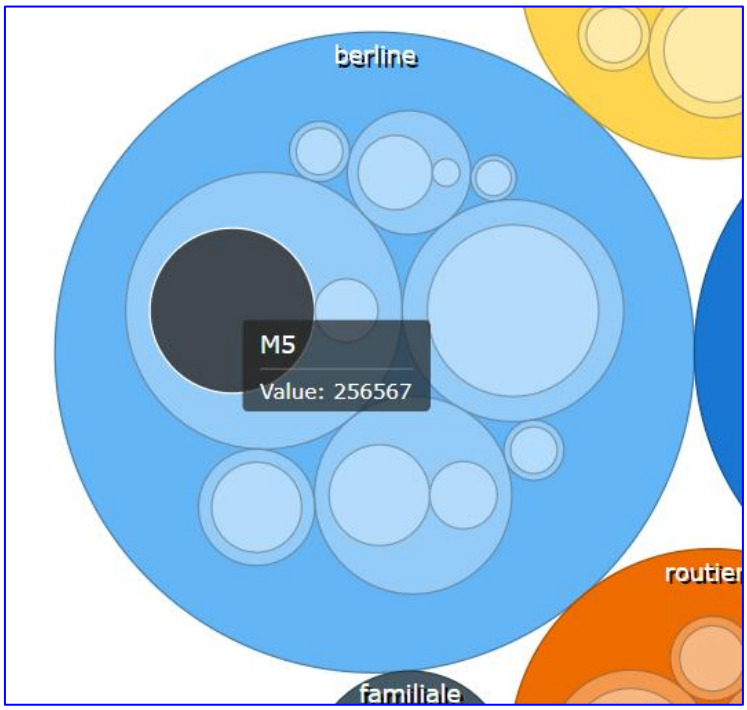
La figure ci-dessus montre le nombre total de voitures vendues cette année. Il faut noter que les 3 niveaux de cercles représentent respectivement les catégories (berline, monospace, citadine, routière, familiale), les marques et les modèles des véhicules.
Les tailles des cercles représentent le nombre de ventes en fonction de la catégorie, la marque ou le modèle.
Prenons l'exemple de la catégorie berline, nous pouvons remarquer qu'il y a 780 618 voitures, appartenant à plusieurs marques, qui étaient vendues (comme le montre la figure 16).
Si nous prenons la marque BMW comme exemple (étant donné que c'est la marque la plus vendue dans cette catégorie), nous pouvons constater que le nombre total des ventes est de 294 455 voitures (Figure 17).
Au niveau de cette marque, nous avons 2 modèles de voitures (2 cercles). Par exemple, le premier cercle représente le modèle M5 et comme le montre la figure 18, 256 567 voitures de ce type ont été vendues cette année.



16 - Vente de berlines



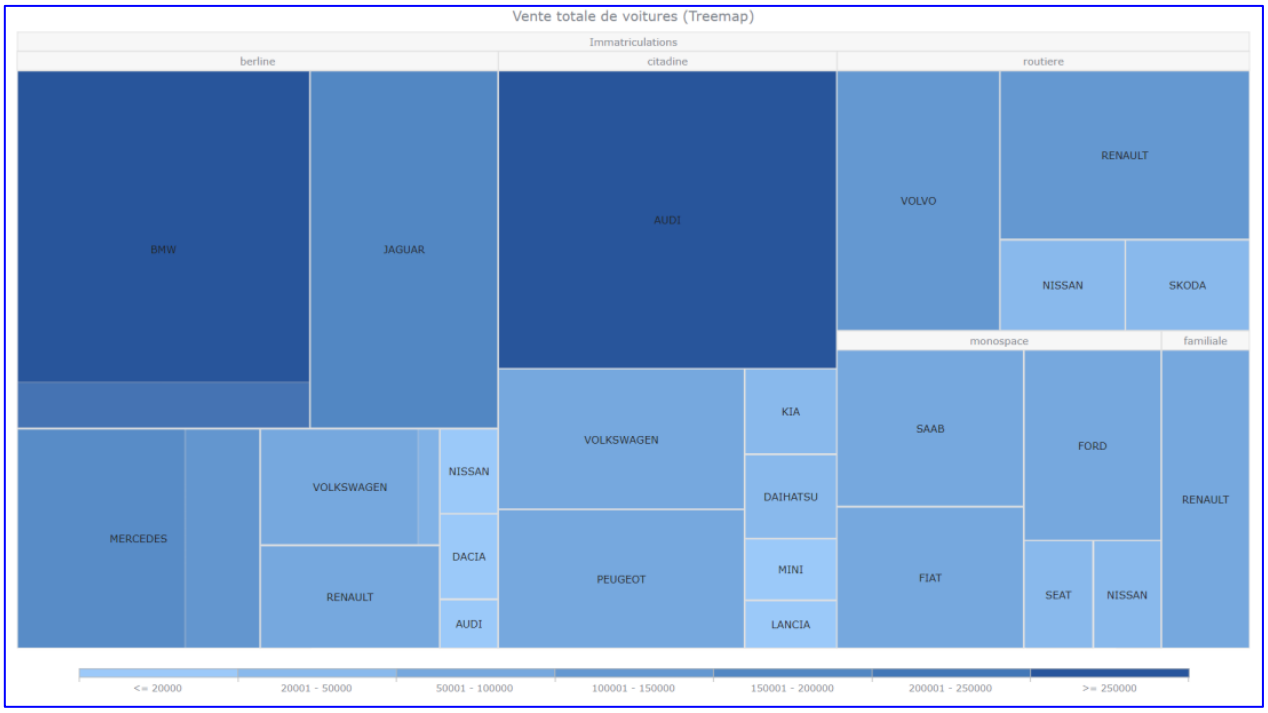
17 - Vente de la marque BMW



18 - Vente du modèle BMW M5

2ème graphe :

Nous avons décidé de répondre au premier objectif du vendeur mais d'une façon différente par rapport au premier graphe.
Etant donné que tous les traitements étaient faits dans l'étape précédente, nous nous sommes intéressés directement à la table finale, générée précédemment (contenant la catégorie et le nombre de ventes) et nous avons utilisé la technique « treemap » : https://datavizcatalogue.com/methods/treemap.html

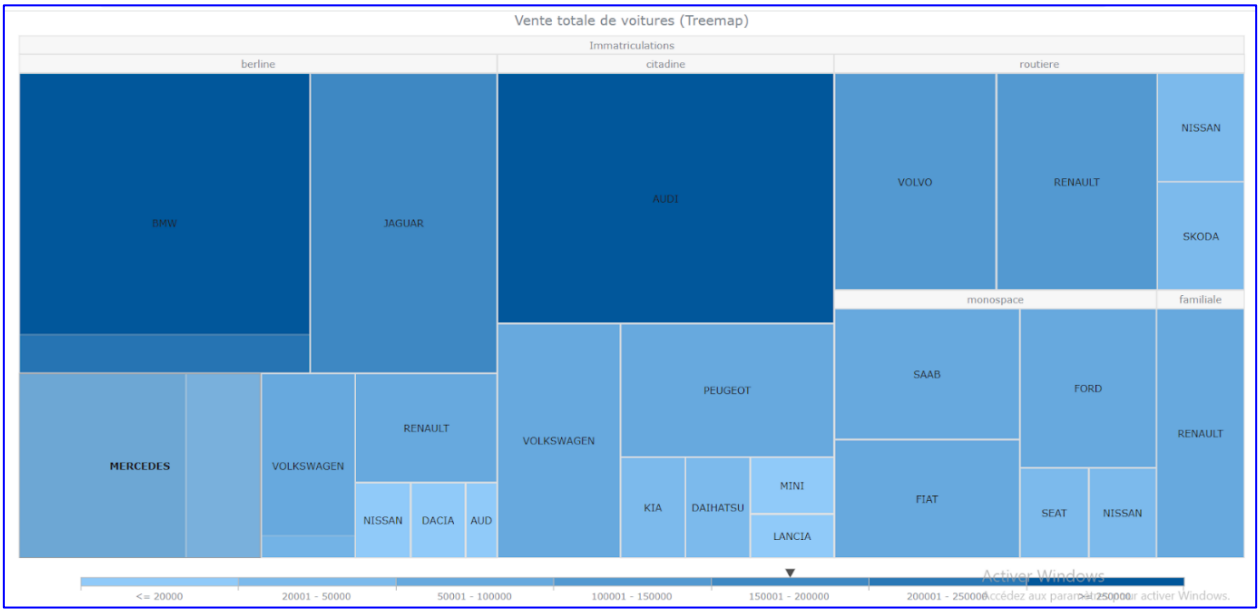


19 - Vente totale des voitures (treemap)

La figure ci-dessus montre la vente totale des voitures de cette année. Dans cette dernière, il y a deux niveaux de visualisation.
Au premier plan, nous avons une vision globale où nous trouverons les catégories des voitures et au niveau de chacune, nous avons les marques correspondantes.

Nous avons également une échelle qui représente des intervalles contenant les nombres de voitures vendues. Il faut noter que les tailles et les couleurs des rectangles sont relatives au nombre de ventes.

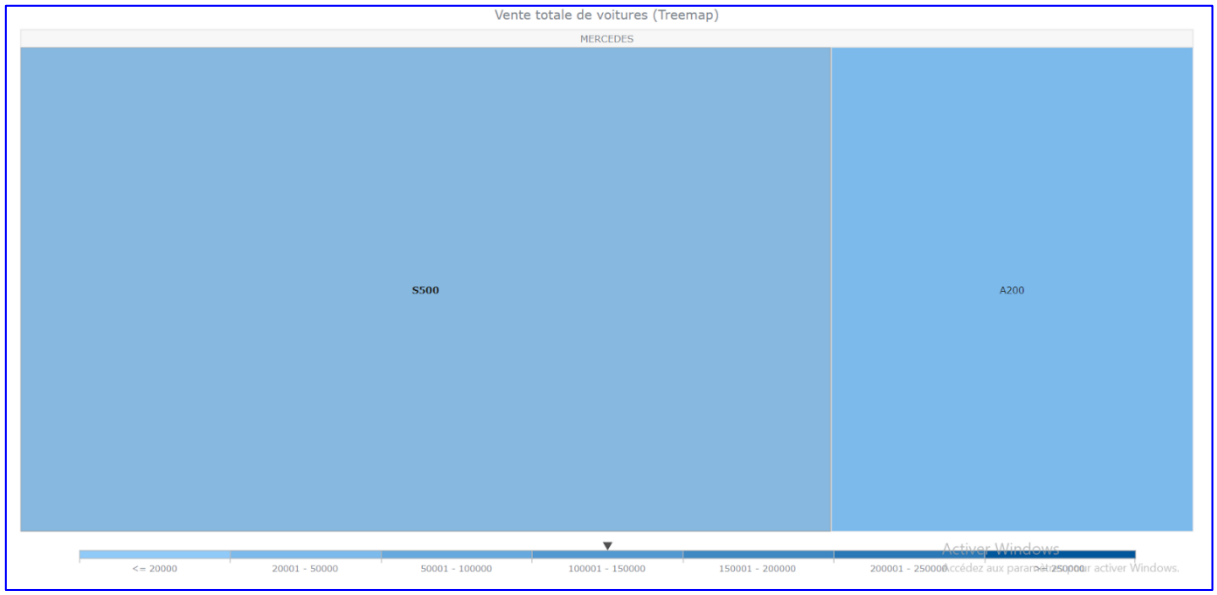
En prenant l'exemple de Mercedes, nous remarquons que le nombre de voitures vendues était entre 150001 et 200000 (voir Figure 20).



20 - Vente totale des Mercedes (berline)

Au deuxième plan (en choisissant une catégorie et une marque), nous avons une vision plus détaillée où nous pouvons visualiser les différents modèles d'une marque spécifique qui sont eux aussi représentés par des rectangles de différentes tailles et couleurs.

En reprenant notre exemple avec la marque Mercedes, en cliquant sur la marque, nous obtenons les ventes totales des modèles S500 et A200 (Figure 21).



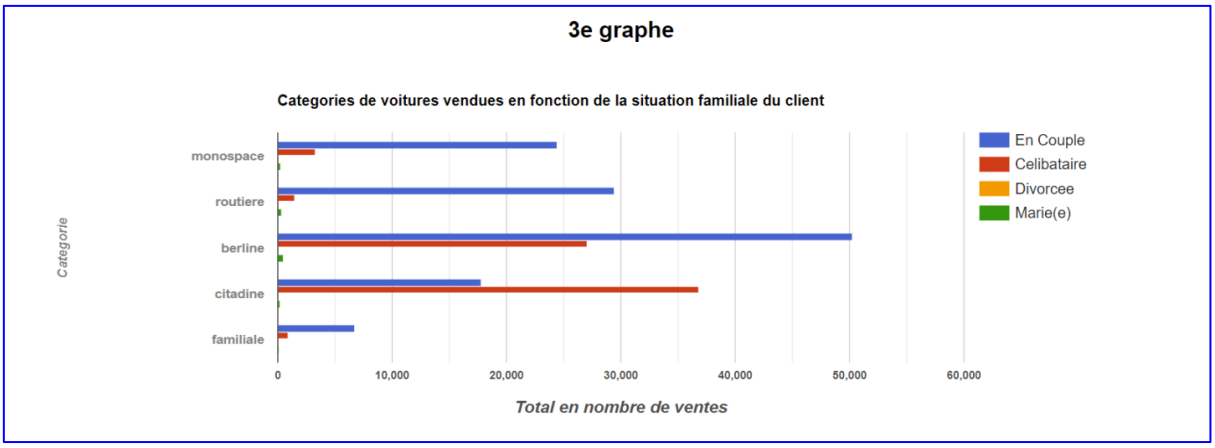
21 - Vente totale des Mercedes S500 et A200

3ème graphe :

Pour répondre au 2ème objectif du vendeur *(visualiser les catégories de voitures vendues par rapport aux profils des clients),* la technique « Multi-set Bar Chart » était la plus appropriée.
Ce graphe va être également utilisé pour répondre à l'objectif du client qui est de visualiser les voitures achetées par des profils semblables au sien.

Pour ce faire, nous avons utilisé la table immatriculation afin d'identifier les catégories de véhicules.
Puisque les différentes voitures de notre catalogue répondent à des besoins différents, nous avons utilisé la table client pour identifier les différents profils.
Cette dernière contient les clients qui ont réalisé des achats au cours de cette année.

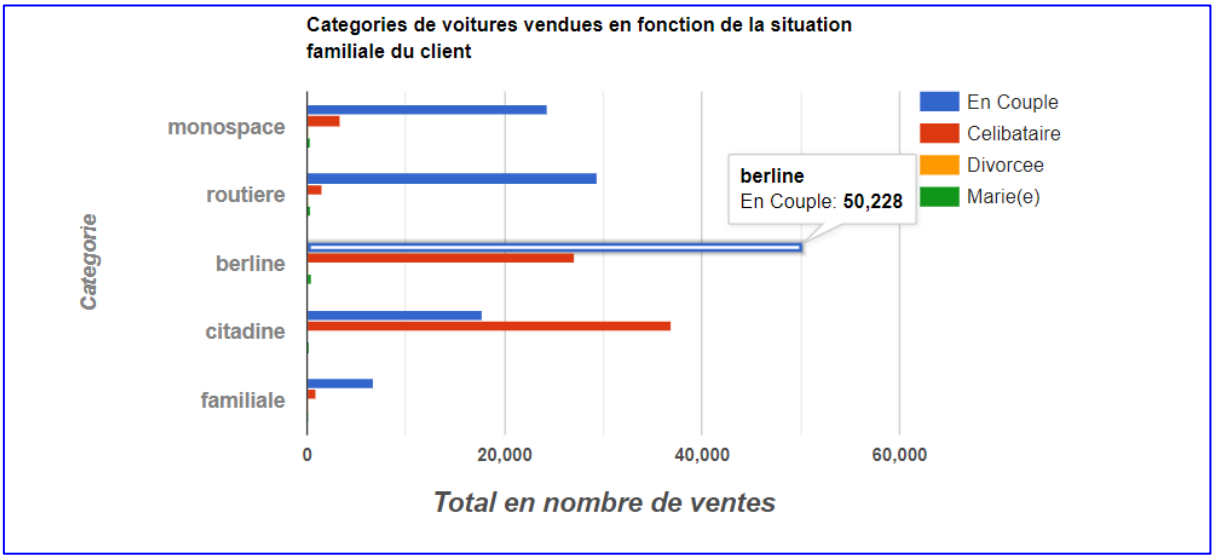
Pour réaliser ce graphique, nous nous sommes intéressés à l'attribut « situation familiale ». Par exemple, les grandes voitures ont de l'espace pour transporter toute une famille (clients mariés par exemple). De ce fait, nous avons généré une nouvelle table de deux colonnes (la catégorie et la situation familiale du client) que nous avons utilisée pour réaliser notre graphe « catégorie de voitures vendues en fonction du profil client ».



22 - Catégorie de voitures vendues en fonction du profil client

La figure ci-dessus montre les catégories de véhicules vendus en fonction de la situation familiale du client.
Les axes des x et des y représentent respectivement les ventes totales et les catégories des voitures.
Les couleurs des barres sont relatives aux différentes situations familiales (en couple, célibataire, divorcé, marié).

Prenons l'exemple des berlines, environ 50 000 clients qui sont en couple ont acheté des voitures appartenant à cette catégorie.



23 - Vente des berlines par les clients "en couple"

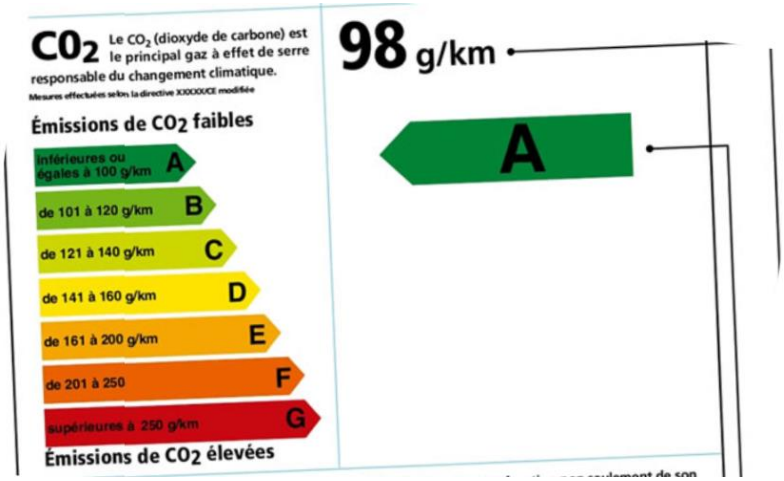
4ème graphe :

Pour répondre au 3ème objectif du vendeur (*présenter la répartition des émissions CO2 des voitures de son catalogue pour attirer les clients les plus sensibles à l'écologie*), nous avons utilisé la technique « pie-chart » :
https://observablehq.com/@d3/pie-chart.

Pour réaliser ce graphique, nous avons eu recours à la table catalogue_new. Cette dernière a été générée au niveau de la partie Hadoop de ce projet.

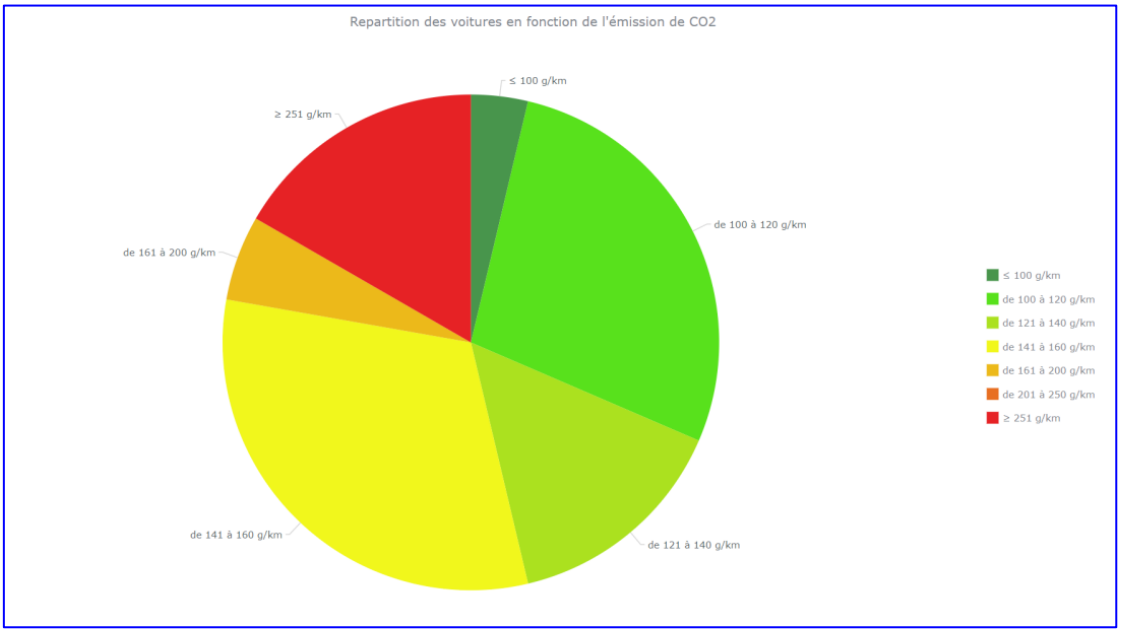
Cette table est le résultat de la fusion entre la table catalogue et la table CO2 qui contient les détails sur l'émission CO2 (colonne rejet CO2 g/km).

Nous avons ensuite utilisé les étiquettes énergie/CO2 des véhicules permettant d'identifier les émissions de carbone pour 100 kilomètres parcourus.
Ces étiquettes comportent un classement de la lettre A à la lettre G, A caractérisant les véhicules qui émettent le moins de carbone et G ceux qui en émettent le plus (Figure 24).
Ces étiquettes nous ont permis de classer les valeurs des rejets CO2 (de la table catalogue_new) par intervalles.



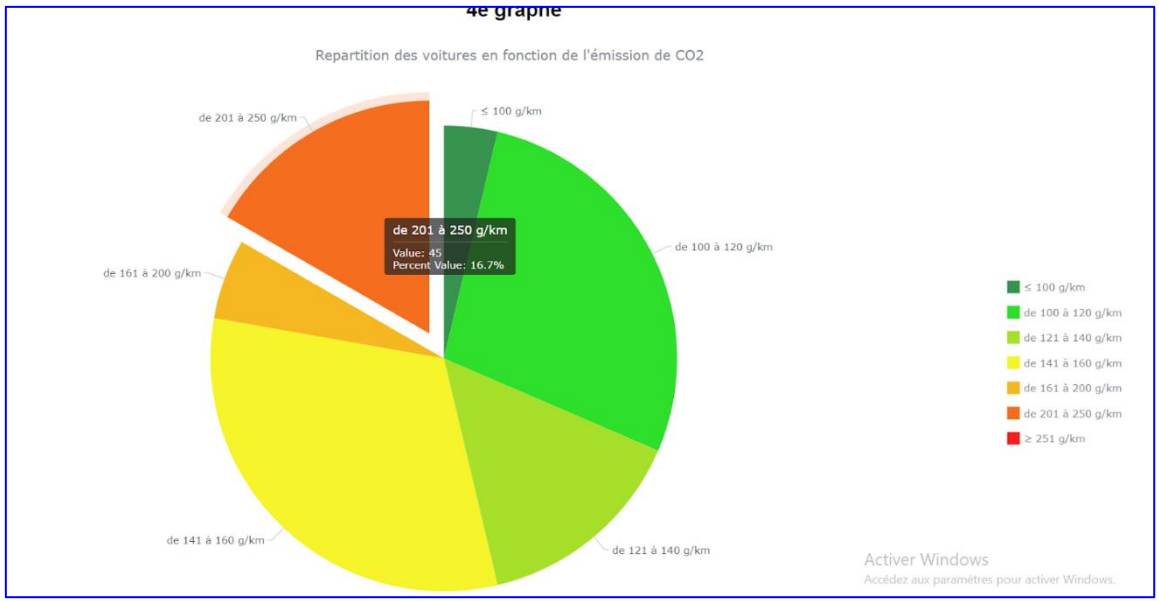
24 - Etiquette émissions de CO2 des véhicules

Nous nous sommes servis de ces fichiers pour créer une nouvelle table contenant le nombre de voitures et le rejet CO2 g/km (intervalles des étiquettes) correspondant à ces voitures. Cela nous a permis de réaliser le graphe « pie-chart ».



25 - Répartition des voitures en fonction de l'émission de CO2

La figure ci-dessus montre la répartition des voitures en fonction de l'émission de CO2. Le graphique est divisé en secteurs, où chaque secteur montre la taille relative de chaque valeur.
Les couleurs des secteurs étaient inspirées des couleurs des étiquettes énergétiques. La couleur rouge représente par exemple les voitures qui émettent le plus de CO2. En s'appuyant sur ce graphe, le vendeur ne va pas proposer de telles voitures aux clients les plus sensibles à l'écologie et il va se rendre compte que 16.7 % des véhicules de son catalogue appartiennent à cette classe (Figure 26).



26 - Nombre de voitures appartenant à la classe G (rouge)

Exécution du script

Il y a deux scripts, un qui permet d'utiliser les fichiers en local (tpa_local.R) et l'autre qui permet de récupérer les bases de données dans Hive grâce à prestoDB (tpa_distant.R).

Récupération et gestion des données

Les données que nous allons exploiter sont directement récupérées du data Lake accessible par des requêtes prestoDB. Ces requêtes sont effectuées directement en R à partir de la bibliothèque RPresto.

Les résultats des requêtes sont stockés dans des tableaux sur lesquels nous avons effectué des visualisations pour vérifier l'intégralité des données.

Visualisation des données du Catalogue

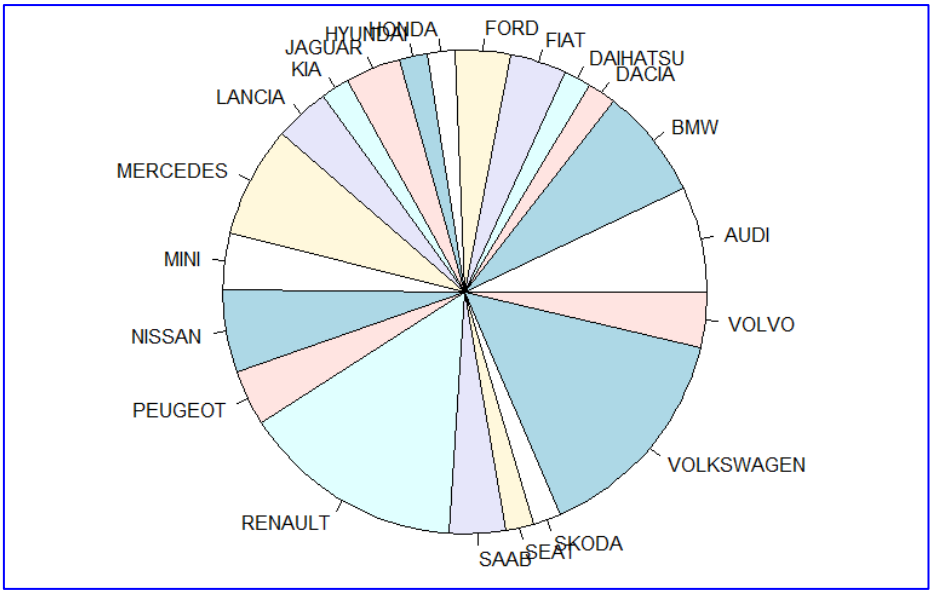
Les données dont nous disposions au début présentaient des problèmes (valeurs incohérentes, encodage …) qui ont été corrigés à l'aide de l'ETL (Section 1).

En première partie, nous avons récupéré les données de catalogue et nous les avons visualisées avec des graphes de natures différentes. Nous avons visualisé des diagrammes pour chaque colonne du catalogue mais les illustrations qui suivent n'affichent, par simplicité, que les graphes relatifs à la colonne « marque ».

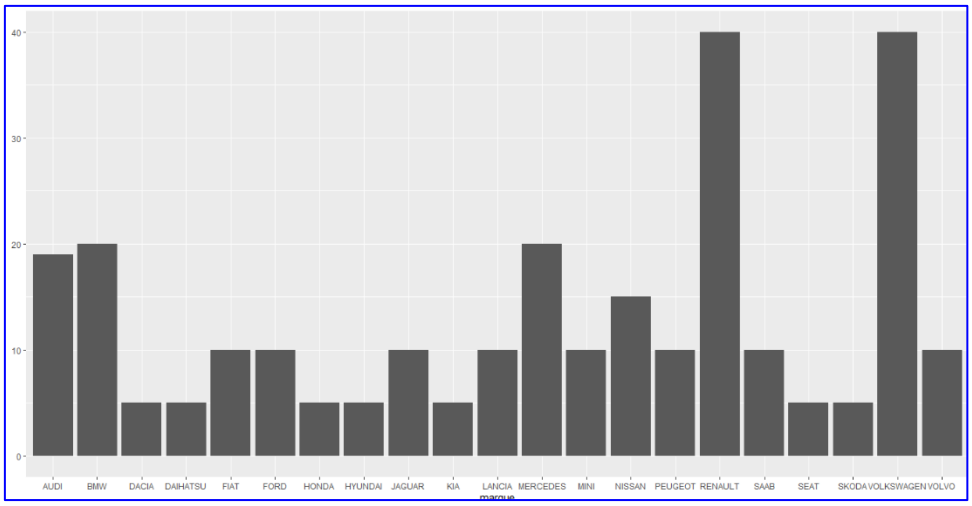
```
table(vehicules$marque)

    AUDI      BMW    DACIA DAIHATSU     FIAT     FORD    HONDA  HYUNDAI
      19       20        5        5       10       10        5        5
  JAGUAR      KIA  LANCIA MERCEDES     MINI   NISSAN  PEUGEOT  RENAULT
      10        5       10       20       10       15       10       40
    SAAB     SEAT    SKODA VOLKSWAGEN    VOLVO
      10        5        5       40       10
```

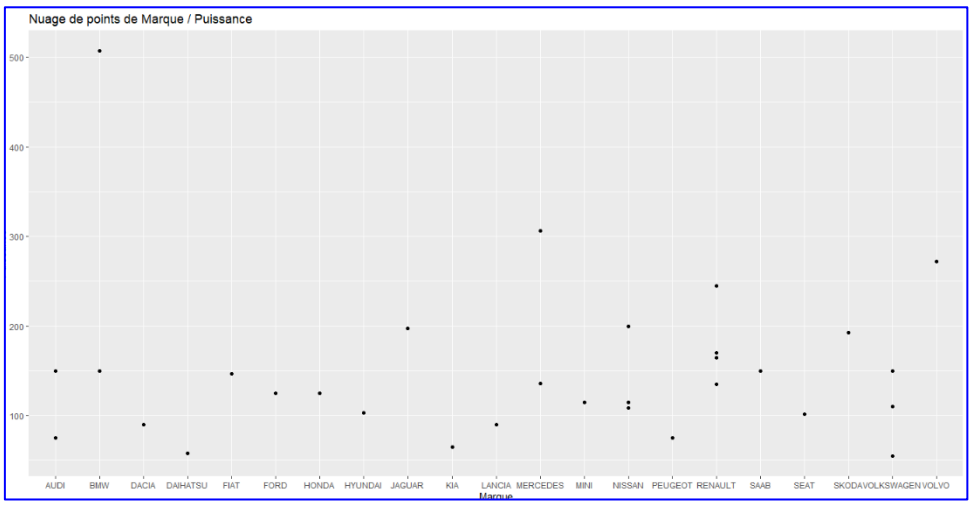
27 - Table d'effectif des marques des véhicules



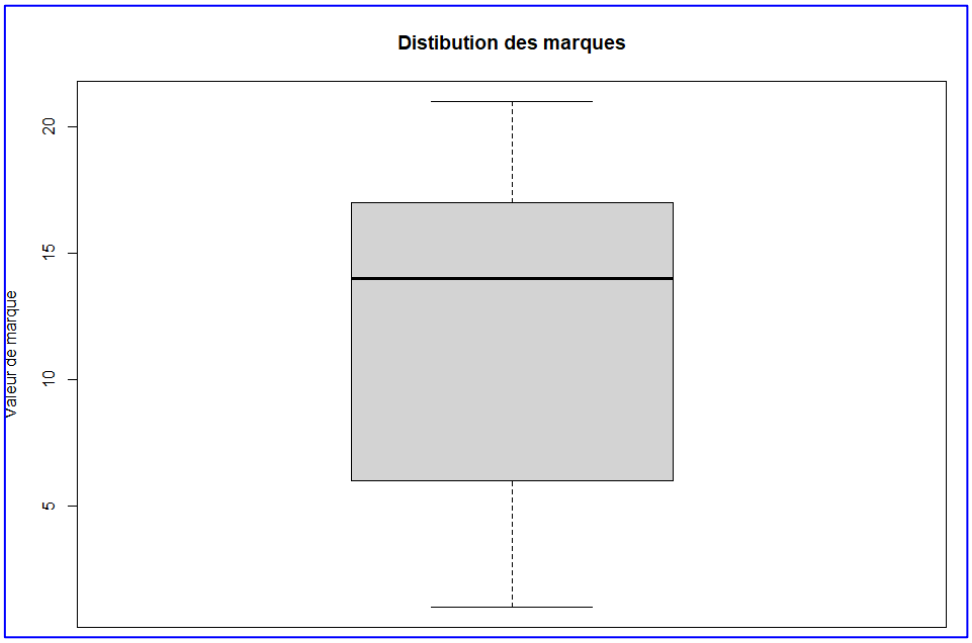
28 - Graphique sectoriel



29 - Histogramme



30 - Nuage de points



31 - Boîte à moustache

Catégories de véhicules dans Catalogue

Nous devions ensuite entrainer un modèle pour la prédiction des catégories des véhicules. Pour ce faire, nous avions besoin de données initiales (définies subjectivement par nous-même).

Nous avons donc affecté à chaque véhicule du catalogue une catégorie en nous basant sur des critères trouvés en recherchant sur internet.

Une fois cette manipulation effectuée, nous avons pu passer à l'étape de l'identification des catégories de véhicules.

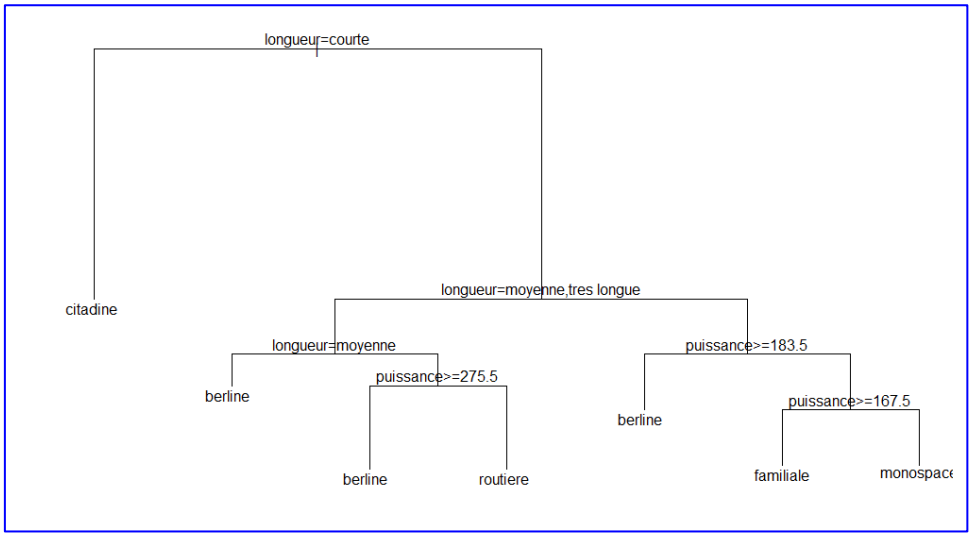
Identification des catégories de véhicules

Une fois les catégories affectées aux données du catalogue, nous avons créé un modèle dont le but est de prédire les catégories des véhicules provenant de la table immatriculations.

Pour arriver à cette fin, nous avons suivi le processus suivant :

1) Nous avons divisé les données du catalogue en données d'entrainement (2/3 du dataset) et données de validation (le 1/3 restant).

2) Ensuite, nous avons construit un arbre de décision à partir des données d'entrainement qui détermine la catégorie à partir de « longueur, puissance et nombre de portes ». Le tri des données a été effectué au fur et à mesure jusqu'à l'obtention de données prédites qui nous semblaient les plus correctes possible.



32 - Identification des catégories de véhicules

Après validation de notre arbre, nous l'avons appliqué aux données de la table immatriculations dans le but de prédire la colonne catégorie de chaque élément, le résultat sera appelé immatriculations_predites.

Nous avons ensuite écrit cette table immatriculations_predites en base pour pouvoir nous en servir ultérieurement.

Identification des catégories de véhicules pour chaque client

La prochaine étape est de créer un modèle qui pourra prédire une catégorie de voiture associée à un client.
Cette démarche a pour but de prédire les types de voitures susceptibles d'intéresser un client potentiel en connaissant uniquement son profil.

Pour cela nous allons dans un premier temps fusionner la table clients avec la table immatriculations_predites en utilisant la colonne "immatriculation" comme clé de jointure.
De plus, nous allons uniquement garder la colonne "catégorie" des données de la table immatriculations_predites car c'est la seule qui nous intéresse.

Nous avons ainsi une table clients avec une catégorie associée à chaque client. Cette table pourra être utilisée pour entrainer un modèle.

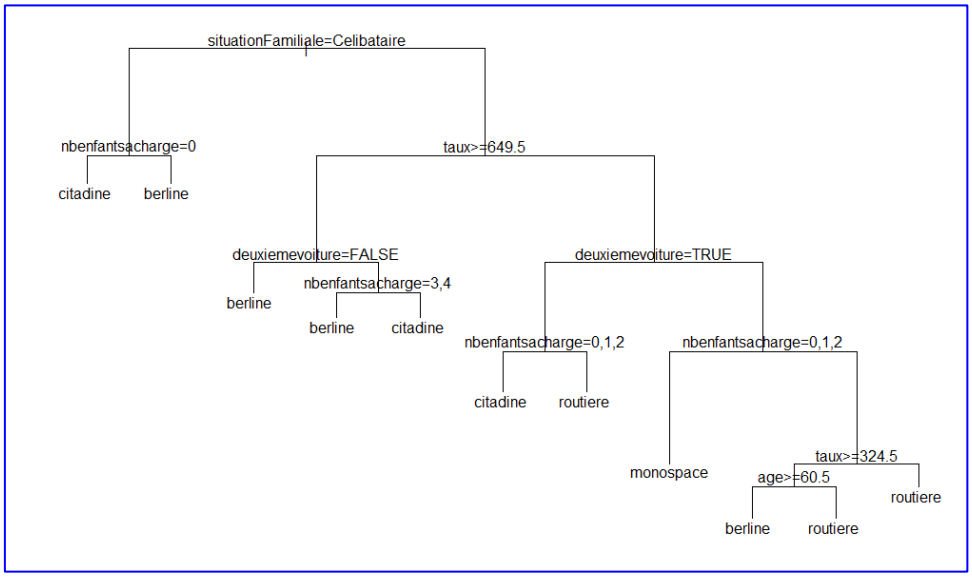
```
clientsV <- merge(x = clients, y = vehicules_imm[ , c("immatriculation","categorie")], by = "immatriculation")
```

Afin d'avoir un modèle assez précis, nous sommes passés par trois méthodes différentes et nous les avons comparées.

Arbre RPart

La première méthode que nous avons testée est l'arbre de décision.
Nous avons construit un arbre de décision à partir de toutes les données de clients sauf immatriculations.
Nous avons aussi fait attention à transformer la colonne "nbenfantsacharge" en factor (entier) pour éviter que l'arbre nous renvoie une décision en décimal sur la variable en question, ce qui n'a aucun sens dans la vie réelle.

L'arbre obtenu est le suivant :



33 - arbre RPart

Et son taux de réussite est de 67% avec 65 323 réussites.

```
> nbr_succes
[1] 65323
> taux_succes
[1] 0.667099
>
```

34 - taux de réussite arbre RPart

Random forest

Nous avons ensuite utilisé random forest pour entrainer le modèle.
Généralement random forest a été le plus rapide à s'exécuter et nous l'avons exécuté plusieurs fois avec des indices différents à chaque tentative pour obtenir le meilleur résultat.

Voici les résultats avec 100 ntree et 1 mtry :

```
> rf <- randomForest(categorie~., clientsV_EA, ntree = 100, mtry = 1)
> rf_class <- predict(rf,clientsV_ET, type="class")
> print(table(clientsV_ET$categorie, rf_class))
           rf_class
            berline citadine familiale monospace routiere
  berline     34216      704         0      2509     1008
  citadine    10674    16159         0        12        5
  familiale    2299        6         0      1395       22
  monospace    5701        5         0      7899       71
  routiere     6577       15         0      3033     5611
>
```

35 - résultats avec 100 ntree et 1 mtry

Et voici les résultats avec 300 ntree et 5 mtry :

```
> rf <- randomForest(categorie~., clientsV_EA, ntree = 300, mtry = 5)
> rf_class <- predict(rf,clientsV_ET, type="class")
> print(table(clientsV_ET$categorie, rf_class))
           rf_class
            berline citadine familiale monospace routiere
  berline     24175     5677       991      6053     1541
  citadine     5666    21141         2        30       11
  familiale     697        9       542      1735      739
  monospace    2325       14      1338      7985     2014
  routiere     2544       18      1423      6127     5124
```

36 - résultats avec 300 ntree et 5 mtry

Nous remarquons que la 2nde tentative est plus performante que la première, donc c'est celle que nous allons garder pour notre évaluation finale.

KNN

Pour KNN, nous avons suivi la même démarche que pour random forest. Dans le cas de KNN nous avons effectué 5 tests différents, chacune avec un k et distance différents.

Résultats pour 10 k et 2 distances :

```
> test_knn(10, 2)

           berline citadine familiale monospace routiere
  berline    38150       43        19       165       60
  citadine   26611       37        17       135       50
  familiale   3680        2         2        17       21
  monospace  13492       19         4       119       42
  routiere   15039       12         8        85       92
```

37 - résultats pour 10 k et 2 distances

Résultats pour 10 k et 4 distances :

```
> test_knn(10, 4)

           berline citadine familiale monospace routiere
berline      38197       42         19       121       58
citadine     26634       37         17       112       50
familiale     3684        2          2        13       21
monospace    13511       19          4        99       43
routiere     15063       13          8        61       91
>
```

38 - résultats pour 10 k et 4 distances

Résultats pour 10 k et 5 distances :

```
> test_knn(10, 5)

           berline citadine familiale monospace routiere
berline      38202       43         18       116       58
citadine     26639       37         18       106       50
familiale     3685        2          2        12       21
monospace    13519       19          4        91       43
routiere     15067       12          8        58       91
>
```

39 - résultats pour 10 k et 5 distances

Résultats pour 50 k et 5 distances :

```
> test_knn(50, 5)

           berline citadine familiale monospace routiere
berline      37715       28          4       626       64
citadine     26285       23          7       498       37
familiale     3638        0          4        62       18
monospace    13341        9          1       278       47
routiere     14867       16          2       256       95
>
```

40 - résultats pour 50 k et 5 distances

Résultats pour 100 k et 5 distances :

```
> test_knn(100, 5)

           berline citadine familiale monospace routiere
berline      37759       15          0       605       58
citadine     26311       12          2       484       41
familiale     3643        1          1        60       17
monospace    13369        6          0       261       40
routiere     14897       11          1       225      102
>
```

41 - résultats pour 100 k et 5 distances

Application du modèle de prédiction aux données Marketing

Le modèle qui nous semble le plus performant est random forest.
C'est donc celui que nous avons choisi et que nous avons appliqué aux clients de la table marketing.
Une nouvelle table, que nous appelâmes marketing_predit, fut ainsi générée.
Elle est donnée dans l'illustration qui suit :

	age	sexe	taux	situationFamiliale	nbenfantsacharge	deuxiemevoiture	predictionvoiture
1	21	F	1396	Celibataire	0	FALSE	citadine
2	35	M	223	Celibataire	0	FALSE	citadine
3	48	M	401	Celibataire	0	FALSE	citadine
4	26	F	420	En Couple	3	TRUE	monospace
5	80	M	530	En Couple	3	FALSE	berline
6	27	F	153	En Couple	2	FALSE	monospace
7	59	F	572	En Couple	2	FALSE	monospace
8	43	F	431	Celibataire	0	FALSE	berline
9	64	M	559	Celibataire	0	FALSE	monospace
10	22	M	154	En Couple	1	FALSE	monospace
11	79	F	981	En Couple	2	FALSE	berline
12	55	M	588	Celibataire	0	FALSE	monospace
13	19	F	212	Celibataire	0	FALSE	berline
14	34	F	1112	En Couple	0	FALSE	citadine
15	60	M	524	En Couple	0	TRUE	citadine
16	22	M	411	En Couple	3	TRUE	routiere
17	58	M	1192	En Couple	0	FALSE	citadine
18	54	F	452	En Couple	3	TRUE	berline
19	35	M	589	Celibataire	0	FALSE	monospace
20	59	M	748	En Couple	0	TRUE	citadine
21	21	F	1396	Celibataire	0	FALSE	citadine
22	35	M	223	Celibataire	0	FALSE	citadine
23	48	M	401	Celibataire	0	FALSE	citadine
24	26	F	420	En Couple	3	TRUE	monospace
25	80	M	530	En Couple	3	FALSE	berline
26	27	F	153	En Couple	2	FALSE	monospace
27	59	F	572	En Couple	2	FALSE	monospace
28	43	F	431	Celibataire	0	FALSE	berline

42 - application du modèle de prédiction aux données Marketing

Sources

Cassandra Handler : https://github.com/dvasilen/Hive-Cassandra/tree/HIVE-1.2.0-HADOOP-2.6.0-CASSANDRA-3.1.1

Mongo Handler : https://github.com/mongodb/mongo-hadoop

Docker-Hive : https://github.com/big-data-europe/docker-Hive

RPresto : https://github.com/prestodb/RPresto

Docker : https://www.docker.com/

Talend : https://help.talend.com/r/fr-FR/7.3/studio-getting-started-guide-open-studio-for-big-data/introduction

Annexes

```yaml
version: "3"

services:
  cassandra:
    container_name: cassandra
    ports:
      - "9042:9042"
    hostname: cassandra
    network_mode: cassandra
    environment:
      - "CASSANDRA_BROADCAST_ADDRESS=${CIP}"
    image: 'cassandra:3.0.8'
  mongo:
    container_name: mongodb
    ports:
      - '27017:27017'
    image: mongo:latest
  namenode:
    image: bde2020/hadoop-namenode:2.0.0-hadoop2.7.4-java8
    volumes:
      - namenode:/hadoop/dfs/name
    environment:
      - CLUSTER_NAME=test
    env_file:
      - ./hadoop-hive.env
    ports:
      - "50070:50070"
  datanode:
    image: bde2020/hadoop-datanode:2.0.0-hadoop2.7.4-java8
    volumes:
      - datanode:/hadoop/dfs/data
    env_file:
      - ./hadoop-hive.env
    environment:
      SERVICE_PRECONDITION: "namenode:50070"
    ports:
      - "50075:50075"
  hive-server:
    image: bde2020/hive:2.3.2-postgresql-metastore
    env_file:
      - ./hadoop-hive.env
    environment:
      HIVE_CORE_CONF_javax_jdo_option_ConnectionURL: "jdbc:postgresql://hive-metastore/metastore"
      SERVICE_PRECONDITION: "hive-metastore:9083"
    ports:
      - "10000:10000"
  hive-metastore:
    image: bde2020/hive:2.3.2-postgresql-metastore
    env_file:
      - ./hadoop-hive.env
    command: /opt/hive/bin/hive --service metastore
    environment:
      SERVICE_PRECONDITION: "namenode:50070 datanode:50075 hive-metastore-postgresql:5432"
    ports:
      - "9083:9083"
  hive-metastore-postgresql:
    image: bde2020/hive-metastore-postgresql:2.3.0
  presto-coordinator:
    image: shawnzhu/prestodb:0.181
    ports:
      - "8080:8080"

volumes:
  namenode:
  datanode:
```

43 - docker-compose.yml