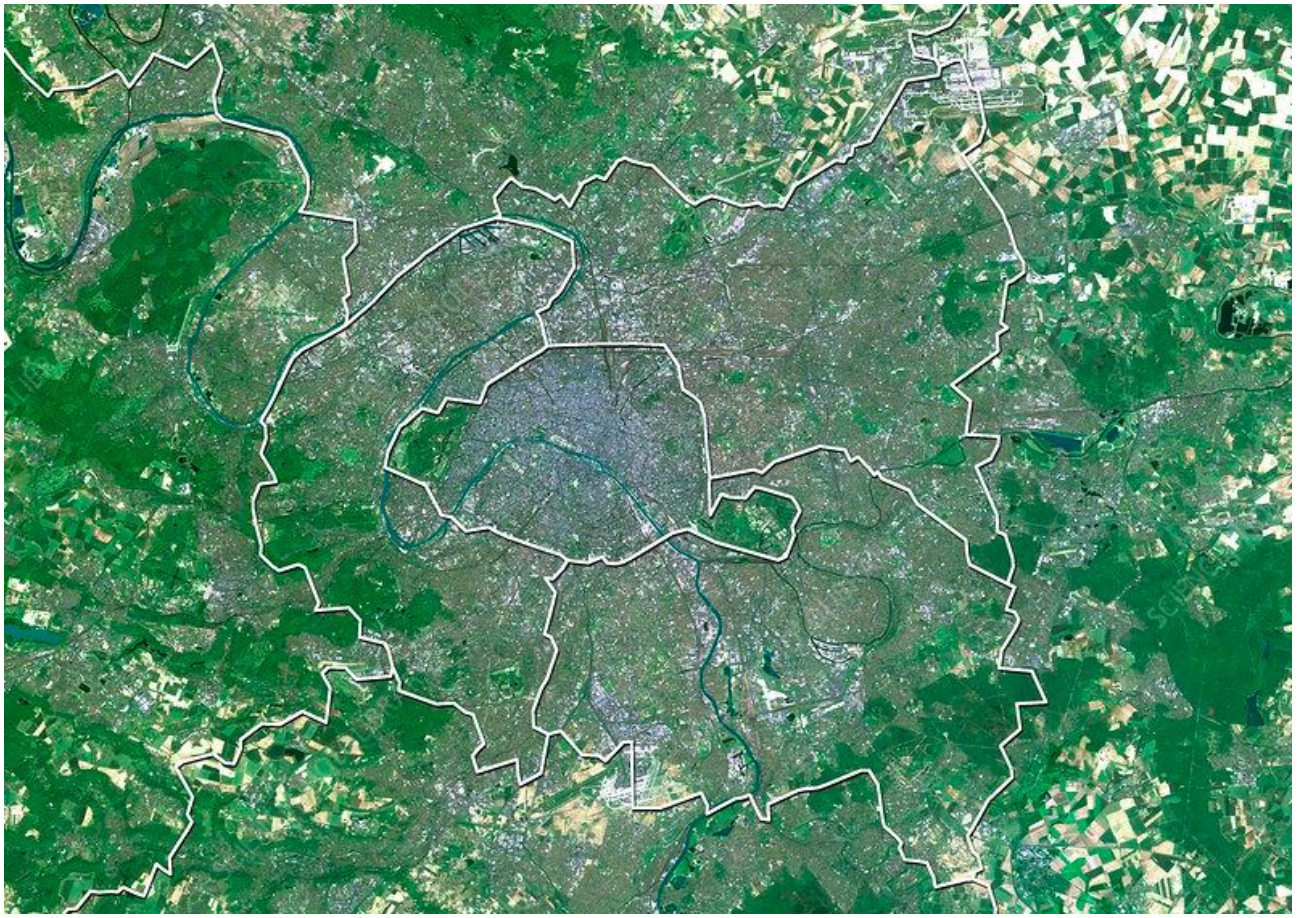
FORMATION DEVELOPPEUR DATA – SIMPLON.CO

Analyse des transports en Île de France



Présenté par : PZ
Encadré par : Manel Boumaiza

TABLE DES MATIÈRES

Résumé

Mon projet a pour but de comparer les temps de transport, sur la totalité des villes et des départements d'Île-de-France, afin de faire émerger des problématiques liées aux transports.

Il consiste en un programme conçu pour s'exécuter sur un serveur de manière autonome. Ce programme envoie automatiquement un mail lorsque celui-ci se termine avec les temps d'exécution, l'heure de fin et, si une erreur intervient, il précise dans le message le type d'erreur rencontré.

Le but de ce projet est de fournir des indicateurs clés, régulièrement mis à jour, aux décideurs afin de faciliter le travail d'analyse et la prise de décision pour l'amélioration des transports en Île-de-France. Cela permet de mettre en évidence les atouts et les faiblesses de l'organisation des transports.

Pour ce faire, j'ai scrapé sur Wikipedia la liste de l'ensemble des villes d'Île-de-France, par département, à l'aide de la librairie Python Beautifull Soup.

Par la suite, grâce à l'API GoogleMaps, j'ai extrait les temps de parcours et les distances entre toutes ces villes et Châtelet-les-Halles, en transport en commun, en vélo, en voiture en trafic 'normal' et aux heures de pointe.

Toutes ces données ont été converties en CSV et en Json afin de pouvoir rapidement les consulter pour l'élaboration du projet et pour conserver un historique.

Les anciens CSV sont automatiquement déplacés vers un nouveau dossier nommé "anciens_csv" lorsque de nouvelles données apparaissent afin de ne pas surcharger le programme lors du traitement des données et pour conserver un historique des collectes. Ainsi le programme ne s'altère pas avec le temps.

Par la suite, les CSV sont intégrés dans une base de données SQL, relationnelle et structurée, afin de permettre la sauvegarde et l'exploitation des données.

Pour préparer la DataViz et la création des KPI, j'ai importé les données de la base de données, puis j'ai optimisé et nettoyé ces données pour créer des dataframes qui me permettront de réaliser le rendu visuel.

L'ensemble de ces dataframes a été rassemblé en un seul. Il regroupe tous les KPI pour chaque villes. Ceci m'a permis de créer une nouvelle base de données NoSQL, regroupant tous les KPI avec une collection par ville, afin de pouvoir effectuer des requêtes sur chaque ville avec ses différents KPI.

Le rendu final est un site Internet créé avec Flask qui regroupe tous les graphes, diagrammes et camemberts et qui permet un requêtage directe de la base de données.

I. Introduction du projet

1. Le contexte

Il existe en Île de France de grandes disparités en termes de transport.

Par exemple, certaines villes ont une politique volontariste sur la « mobilité douce » tandis que d'autres villes ne mettent en œuvre aucune politique.
Certaines villes disposent de nombreux moyens de transport en commun tandis que d'autres n'en ont aucun.

Au cours de cet exposé, je vais extraire des KPI pertinents, afin d'identifier si des villes se distinguent d'autres, et donner les moyens d'en expliquer les causes.

Pour cela, je vais comparer les temps de transport entre toutes les villes d'Ile de France et Châtelet-les-Halles en voiture aux heures de pointe et en trafic normal, en vélo et en transport en communs.

2. Problématique

Définir des KPI concernant les transports en Île de France afin de faire émerger des tendances et des politiques en termes de transport.

Quelles sont les villes ayant les meilleurs ratios temps/distance vers Paris en transport en commun, en vélo ou en voiture ?

Recenser les villes ayant des problèmes de transport et permettre d'en expliquer les causes.

3. Enjeux réglementaires

Concernant les enjeux réglementaires pour le traitement des données de l'application ; les données proviennent de Wikipédia qui met à disposition librement ces données, et de l'API de Google pour lesquelles j'ai dû créer un compte et souscrire un contrat pour utiliser ces données.

4. Organisation du projet

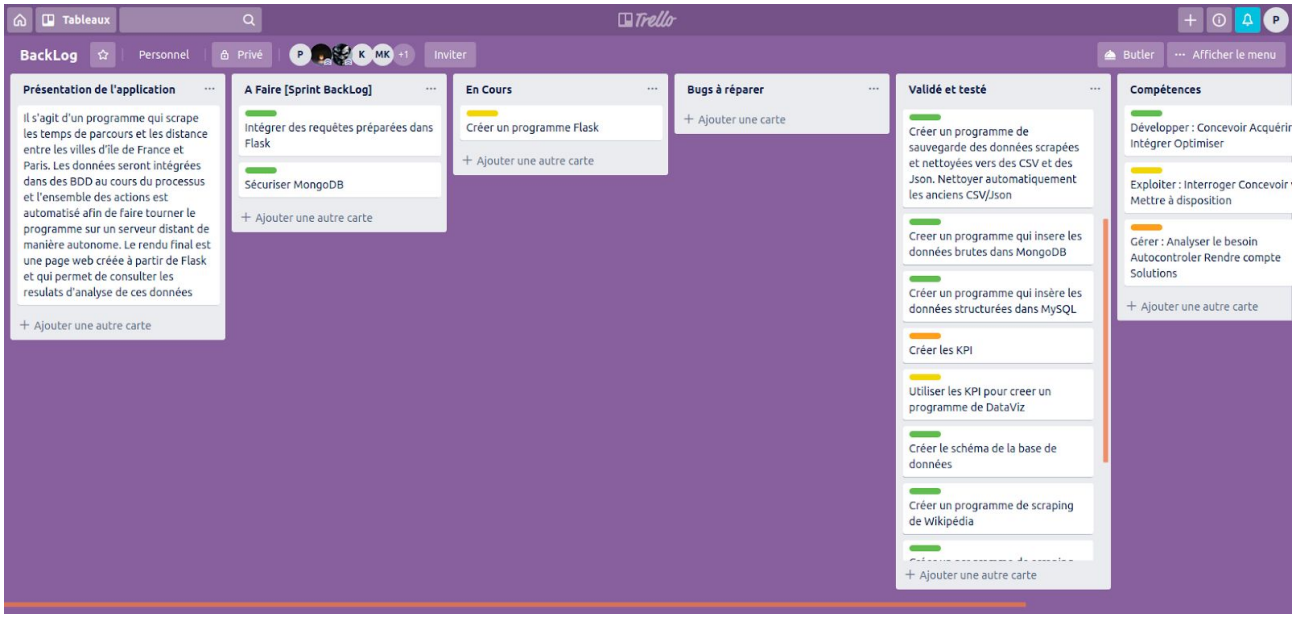
- **Trello**

J’ai opté pour un projet qui automatise tous les processus; de la collecte, au rendu final, en passant par l'intégration dans des bases de données et l’optimisation des données.

Ce projet est conçu pour fonctionner de manière autonome sur un serveur.

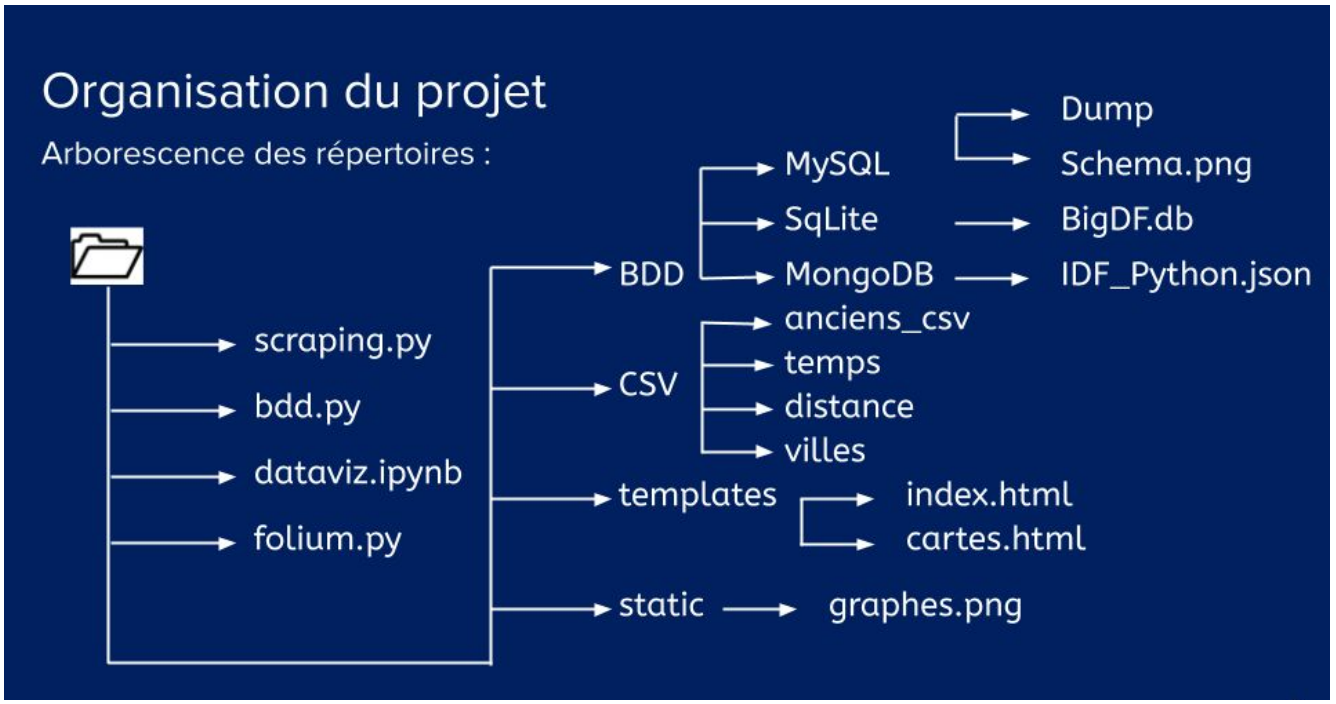
Pour m’organiser dans ce projet, j’ai utilisé l’outil en ligne Trello afin de définir un Backlog que j’ai divisé en tâches précises (des Sprint Backlog).

J’ai pu ainsi définir un planning et visualiser mon avancement sur le projet :

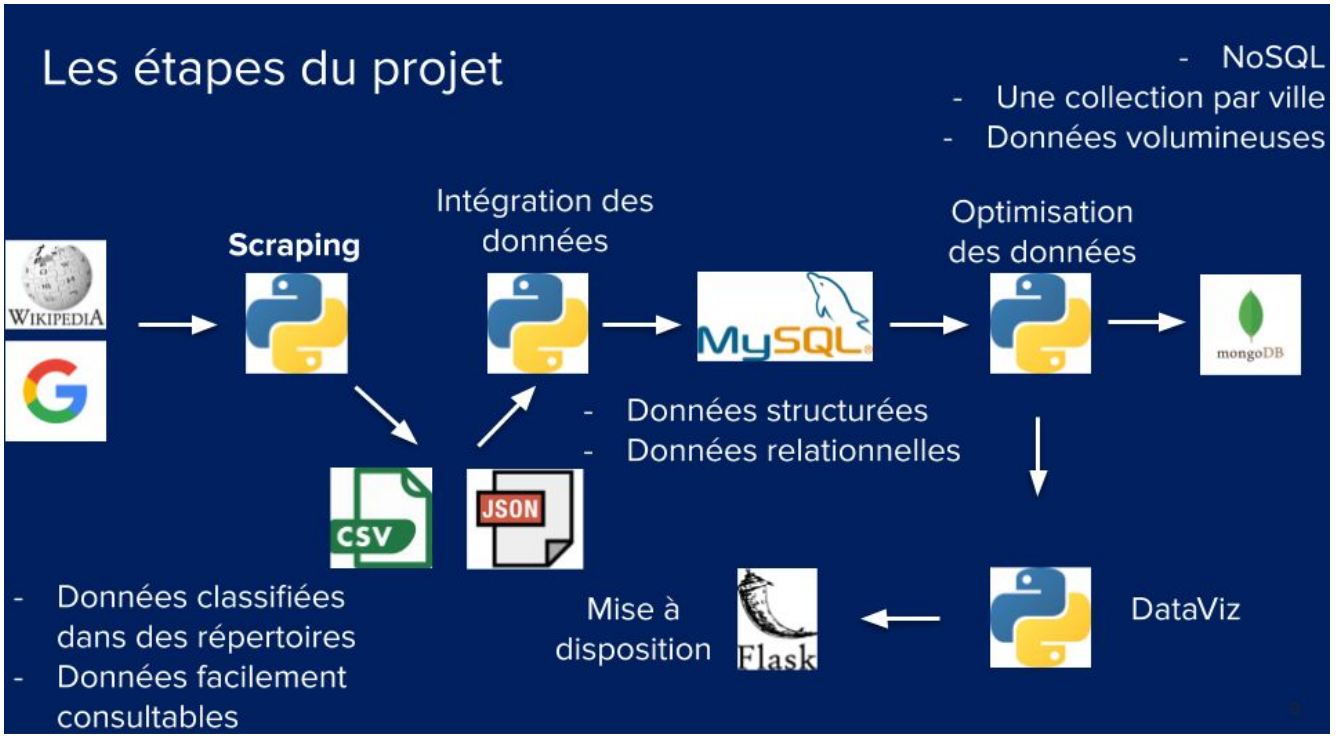


https://trello.com/b/2thMdwot/backlog

- **Arborescence des répertoires**



5. Conception technique



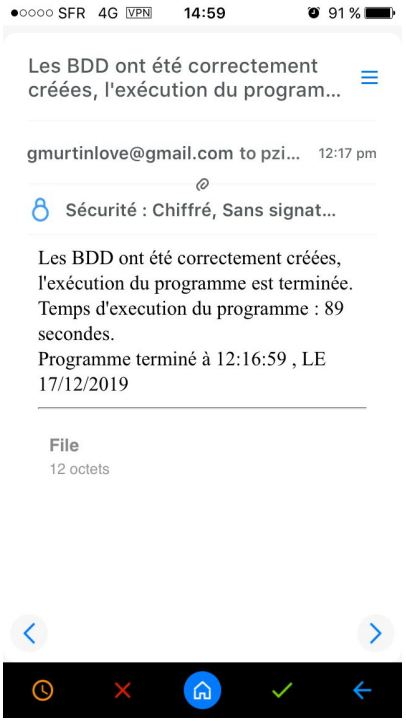
6. Autocontrôle

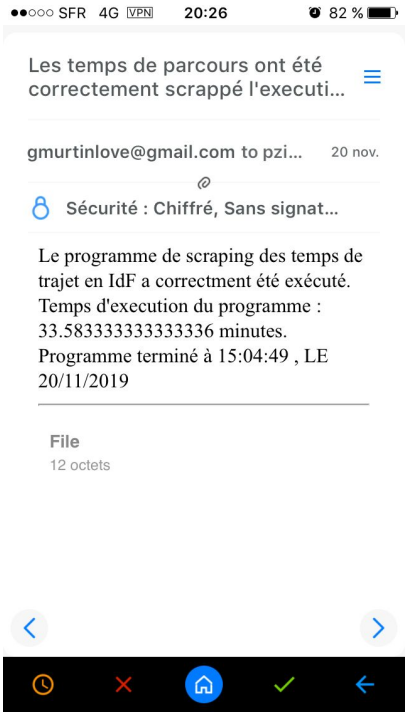
Pour ce qui est de l'autocontrôle, chaque programme envoi un mail lorsque celui-ci s'est correctement exécuté avec un récapitulatif des actions entreprises, les heures de mise en route et le temps d'exécution.

Le programme envoi immédiatement une alerte par mail si une erreur dans le processus est intervenue avec la référence de l'erreur :

```
except Exception as e:
    send_mail("Erreur lors du scraping des temps de parcours. Veuillez vérifier le programme\nVoici l'erreur :\n"+str(e))
```

- **Capture d'écran des messages reçus**





II. L'acquisition des données

1. Collecte de l'ensemble des villes d'Ile de France avec Wikipédia

Pour l'acquisition des données, j'ai utilisé Python et la librairie **Beautifull Soup** pour scraper l'ensemble des villes d'Ile de France sur Wikipédia :

```
import requests
from bs4 import BeautifulSoup


requete =
requests.get("https://fr.wikipedia.org/wiki/Liste_des_communes_de_l%27Essonne")
page = requete.content
soup = BeautifulSoup(page, features="lxml")

essonne=[]
for a in tqdm(soup.find_all('td',{'style':'text-align:left;'})):
    essonne.append(str(a.text))
```

2. Collecte des temps de parcours avec l'API Google Maps

Pour l'acquisition des temps de parcours et des distances en Transport en commun, en vélo et en voiture en trafic "normal" et aux heures de pointes, il m'était impossible d'utiliser Beautifull Soup car les pages web à scraper utilisent JavaScript.

Il m'était aussi impossible de scraper avec Sélénium car au-delà d'un certain nombre de requêtes, j'étais confronté aux Captchas afin de vérifier que je ne suis pas un robot.

Par ailleurs, les données de Google ne sont pas libres de droit.

J'ai donc opté pour l'utilisation de l'API Google Maps :

```
k=-1
for i in tqdm(yvelines):
     k=k+1
     try:
      gmaps = googlemaps.Client(key=clé_api_google)
      direction=yvelines_code_postal[k]+' '+str(i)+' france'
      directions_result = gmaps.directions(
                direction,
```

```
                "Place du Châtelet,Paris",
                mode="driving",
                traffic_model='optimistic',
                departure_time=datetime(2020, 1, 19, 6, 32, 19, 831260))
    temp2_list.append(directions_result)
```

3. Sauvegarde des données

Les données récoltées ont été converties en CSV (données structurées) et en fichier Json (données brutes) afin de pouvoir m'assurer de leur sauvegarde immédiate et pour pouvoir consulter rapidement ces données lors de l'élaboration du projet sans devoir interroger une BDD, ce qui est plus long et fastidieux.

Exemple de création d'un Json :

```
with
open(emplacement_donnees_non_traitees+'/tps_yvelines_voiture'+today1+'.
json', "w") as f:
    for x in temp2_list:
        f.write(str(x))
```

4. Nettoyage automatique des dossiers

Lors d'un nouveau processus de collecte, les anciens fichiers CSV et Json sont automatiquement déplacés vers un autre dossier afin de faciliter la consultation des CSV. Ils ne sont pas supprimés afin de conserver un historique des collectes et pour pouvoir, à l'instar de logs, repérer des erreurs lors de la collecte ou des altérations du programme.

Voici un exemple de nettoyage d'un dossier :

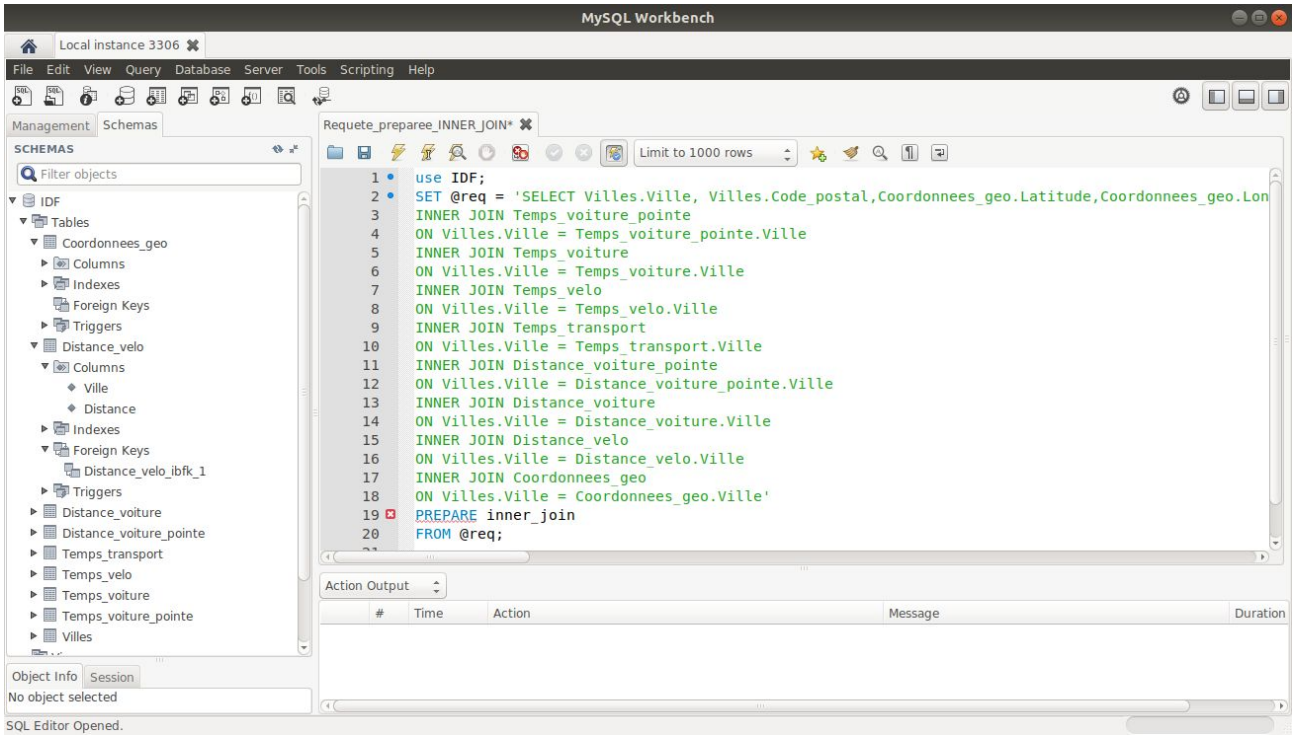
```
for csv in dossier_affaires:
     if csv[-19:-4]!= today1:
     shutil.move(emplacement_csv_temps_voiture+"/"+str(csv),
emplacement_ancien_csv_voiture+"/"+str(csv))
     print("    Anciens CSV déplacés vers le dossier :
"+emplacement_ancien_csv_voiture)
```

III. Intégration des données dans une base de données

1. Choix des bases de données

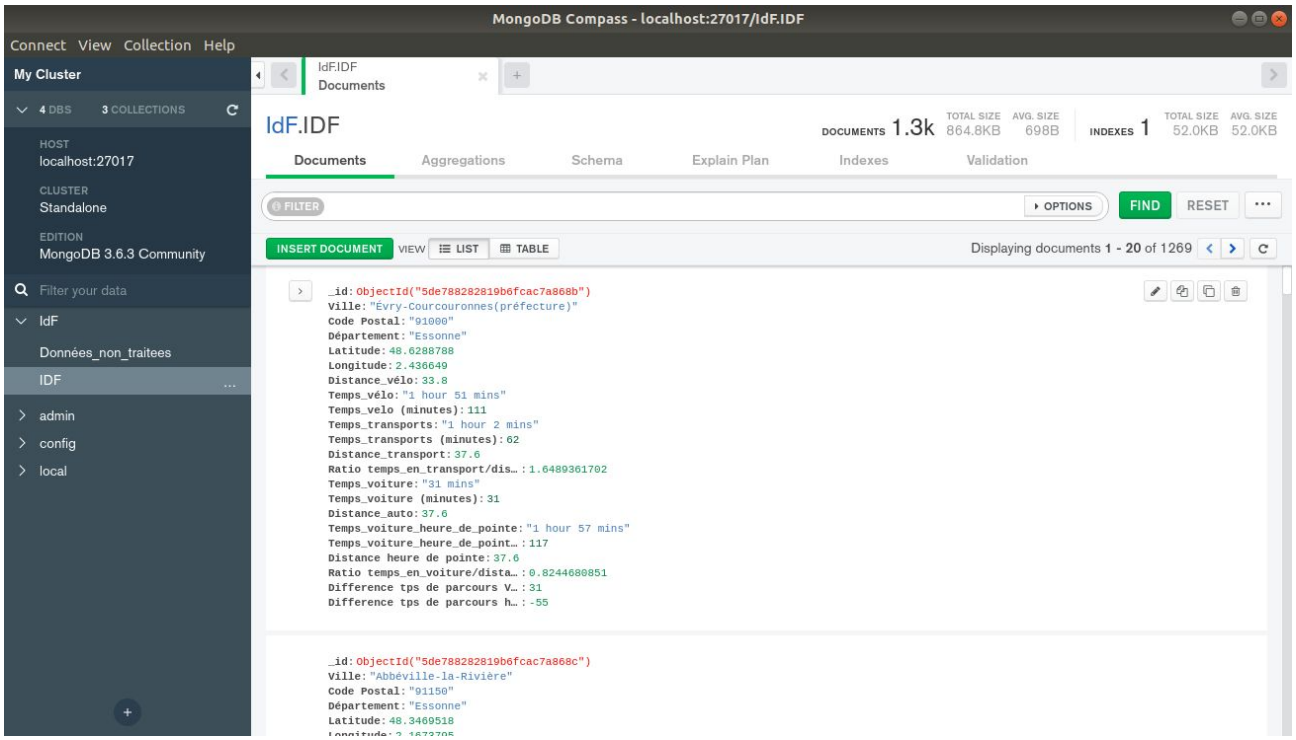
Pour l'intégration des données dans des bases de données, j'ai opté pour du SQL et du NoSQL suivant les types de données à sauvegarder :

- Pour ce qui est des données structurées et relationnelles issues du scraping, je les ai intégrées à MySQL :



- Pour ce qui est de la sauvegarde finale, j'ai regroupé tous les dataframes en un seul avec tous les KPI réunis et j'ai opté pour une base de données NoSQL, avec une collection pour chaque ville.

Voici un exemple de collection dans MongoDB :



2. Intégration des données

J'ai utilisé Python avec les librairies PyMysql pour l'intégration des données dans MySQL et PyMongo pour l'intégration des données dans MongoDB.

- **Avec PyMysql**

Paramètres de connexion à la base de données :

```
 con = pymysql.connect(host=host,
                       user=user,
                       password=password2,
                       autocommit=True,
                       charset='utf8mb4',
                       local_infile=1)
print('Connecté à la BDD: {}'.format(host))

cursor = con.cursor()
```

Exemple de création de la base de données, des tables et des clés étrangères :

```
load_sql = """create table Villes ( Ville varchar(45) primary key,
Code_postal int(6),foreign key (Ville) references
Coordonnees_geo(Ville) on delete cascade)""";
cursor.execute(load_sql)
```

Intégration des données récoltées :

```
print("Intégration des données dans la table")

for csv in os.listdir(emplacement_csv_villes):
   phrase=str(emplacement_csv_villes)+'/'+str(csv)
   load_sql = "LOAD DATA LOCAL INFILE '"+str(phrase)+"' INTO TABLE
Villes FIELDS TERMINATED BY ',' IGNORE 1 ROWS";
   cursor.execute(load_sql)
```

- **Avec PyMongo**

Paramètres de connexion à la base de données :

```
print('\nII. Importation des données non traitées dans MongoDb :\n\n')

client = MongoClient('localhost', 27017)
print("Connecté à MongoDb !\n")
```

Exemple de création de la base de données, et d'une collection :

```
db = client.IdF
print("La BDD '"+str(db.name)+"' a été créée !\n")


collection = db.Données_non_traitees
print("La collection '"+str(collection.name)+"' a été créée !\n")


print("Suppression des anciennes bases de données\n")
collection.remove()
```

Intégration des données récoltées :

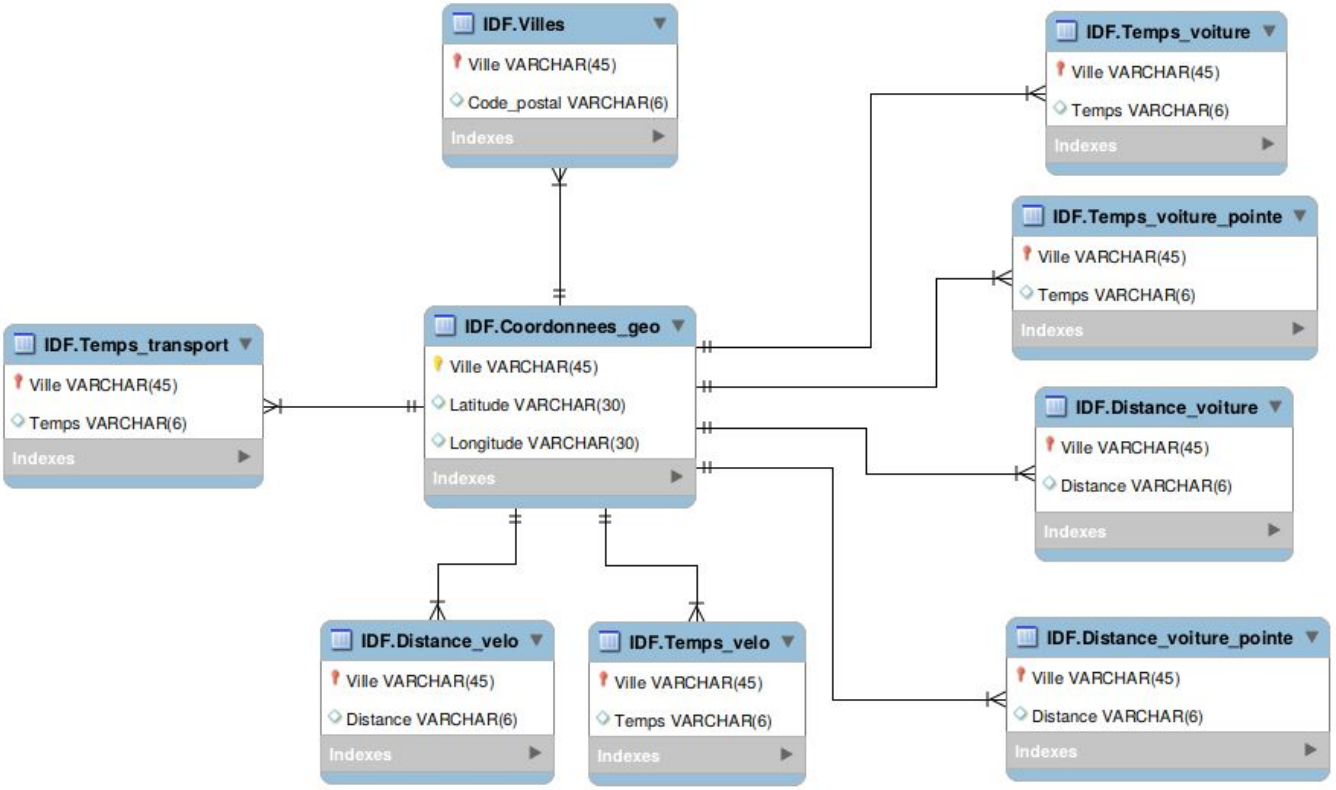
```
for json in tqdm(os.listdir(emplacement_donnees_non_traitees)):
     if os.path.getsize(emplacement_donnees_non_traitees+'/'+str(json)) <= 999999:
          data = pd.read_csv(emplacement_donnees_non_traitees+'/'+str(json))
          payload = JSON.loads(data.to_json(orient='split'))
          collection.insert_one(payload)
          print("Le document Json : "+str(json)+" a été intégré \ndans la base de
données MongoDb '"+str(db.name)+"' dans la collection
'"+str(collection.name)+"'\n")
```

3. Conception de la base de données MySQL

- **Modèle Conceptuel de Données (MCD)**

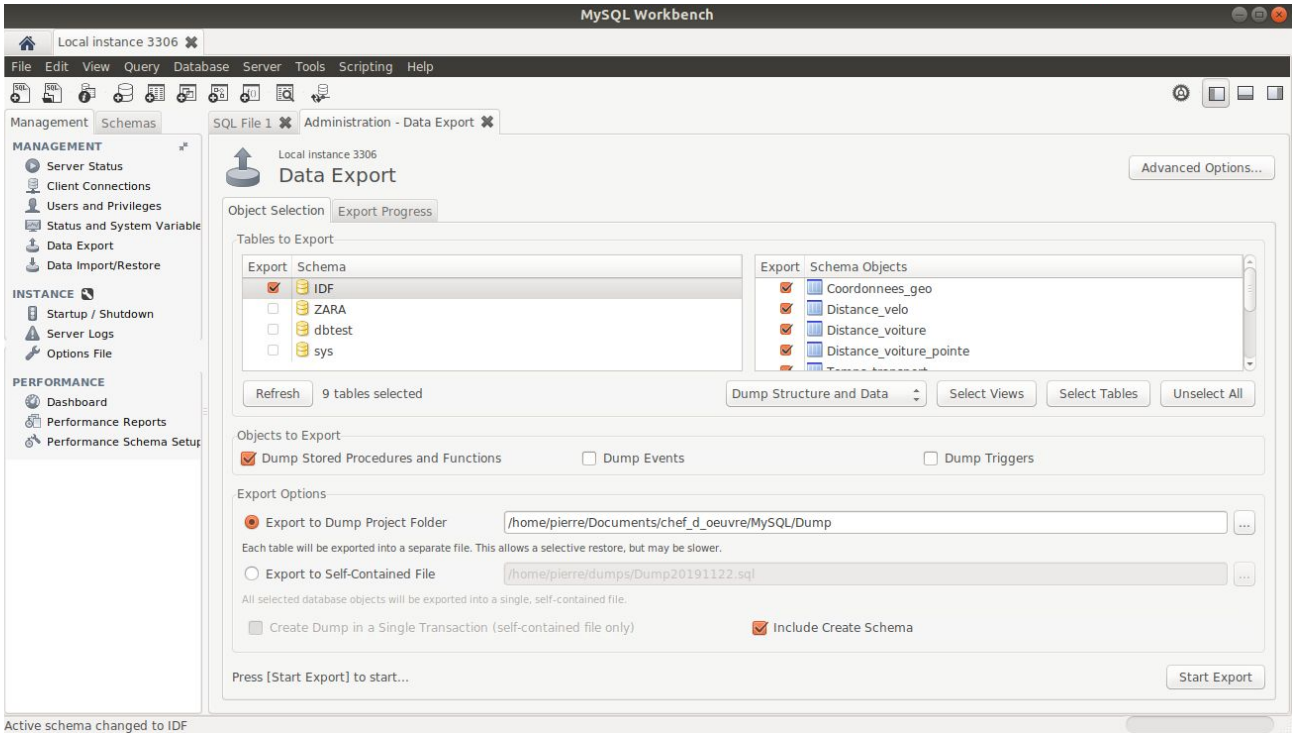
Il s'agit d'une relation "One to many" avec comme contrainte une clé étrangère qui est la colonne 'Ville'

Schéma fonctionnel utilisé pour le projet :



- **Création d’un dossier “Dump” pour la récupération des données**

Manuellement sous Workbench:



Manuellement en ligne de commande :

```
(base) pierre@simplon-thinkpad-t430:~/Documents/chef_d_oeuvre$ mysqldump -u
simplon -p IDF > /home/pierre/Documents/chef_d_oeuvre/BDD/MySQL/Dump/backup.sql
```

Automatiquement au cours du processus :

J’ai créé un programme qui automatise cette sauvegarde avec python “mysqldump”.

```
DB_USER_PASSWORD = 'motdepassemysql'

DB_NAME = 'nomdelabdd'
BACKUP_PATH = 'BDD/MySQL/Dump'

DATETIME = time.strftime('%Y%m%d-%H%M%S')
TODAYBACKUPPATH = BACKUP_PATH + '/' + DATETIME

try:
    os.stat(TODAYBACKUPPATH)
```

```
except:
    os.mkdir(TODAYBACKUPPATH)

db = DB_NAME
dumpcmd = "mysqldump -h " + DB_HOST + " -u " + DB_USER + " -p" +
DB_USER_PASSWORD + " " + db + " > " + pipes.quote(TODAYBACKUPPATH) + "/" + db
+ ".sql"
os.system(dumpcmd)
gzipcmd = "gzip " + pipes.quote(TODAYBACKUPPATH) + "/" + db + ".sql"
os.system(gzipcmd)

print ("")
print ("Base de données sauvegardée !")
print ("La sauvegarde a été créée dans le dossier : '" + TODAYBACKUPPATH)
```

IV. Optimisation des données

1. Optimisation de la structure

J'ai optimisé la structure des données. Pour cela, j'ai utilisé le moteur de données InnoDB plutôt que MyISAM, ce qui permet l'utilisation des clés étrangères.

Par ailleurs, j'ai utilisé l'encodage UTF-8 ce qui permet de gérer tous les caractères internationaux sans problème d'encodage.

2. Sécurisation de la base de donnée

- **Avec MySQL :**

Pour sécuriser ma base de données MySQL, j'ai supprimé l'utilisateur root, qui est utilisé par les pirates, et créé un profil administrateur avec un autre nom :

```
(base) pierre@simplon-thinkpad-t430:~$ sudo mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 5.7.28-0ubuntu0.18.04.4 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights
reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql> GRANT ALL PRIVILEGES ON *.* To 'nouvel_admin'@'hostname'
IDENTIFIED BY 'password';
Query OK, 0 rows affected, 1 warning (0.02 sec)

mysql> FLUSH PRIVILEGES;
```

J'ai aussi créé un compte Invité qui ne peut exécuter que la commande SELECT et SHOW DATABASES dans ma base de données IDF.

```
mysql> CREATE USER Invite@'localhost' IDENTIFIED BY 'password';
mysql> GRANT SELECT, SHOW DATABASES ON IDF.* TO Invite@'localhost';
mysql> FLUSH PRIVILEGES;
```

Pour consulter les droits des utilisateurs :

```
SELECT user,host FROM mysql.user;
```

Pour voir les droits d'un utilisateur sur les bases de données :

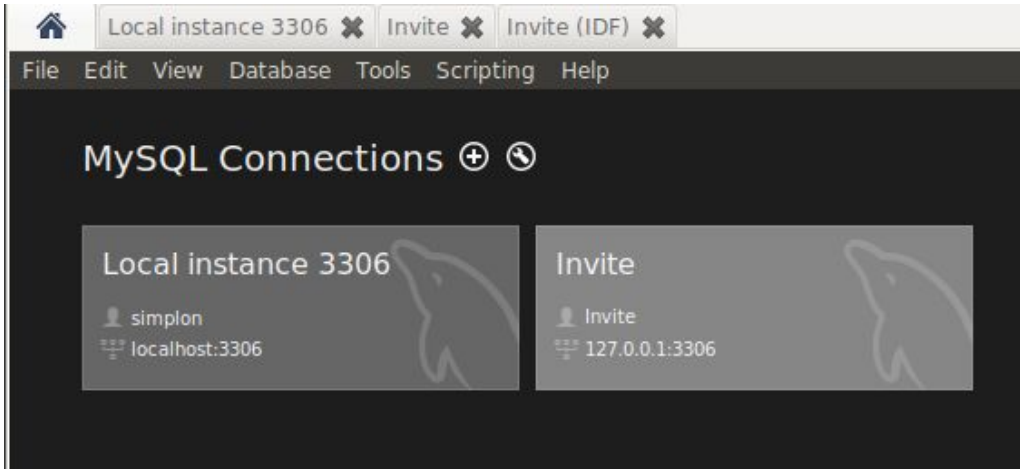
```
SHOW GRANTS FOR 'user'@'localhost';
```

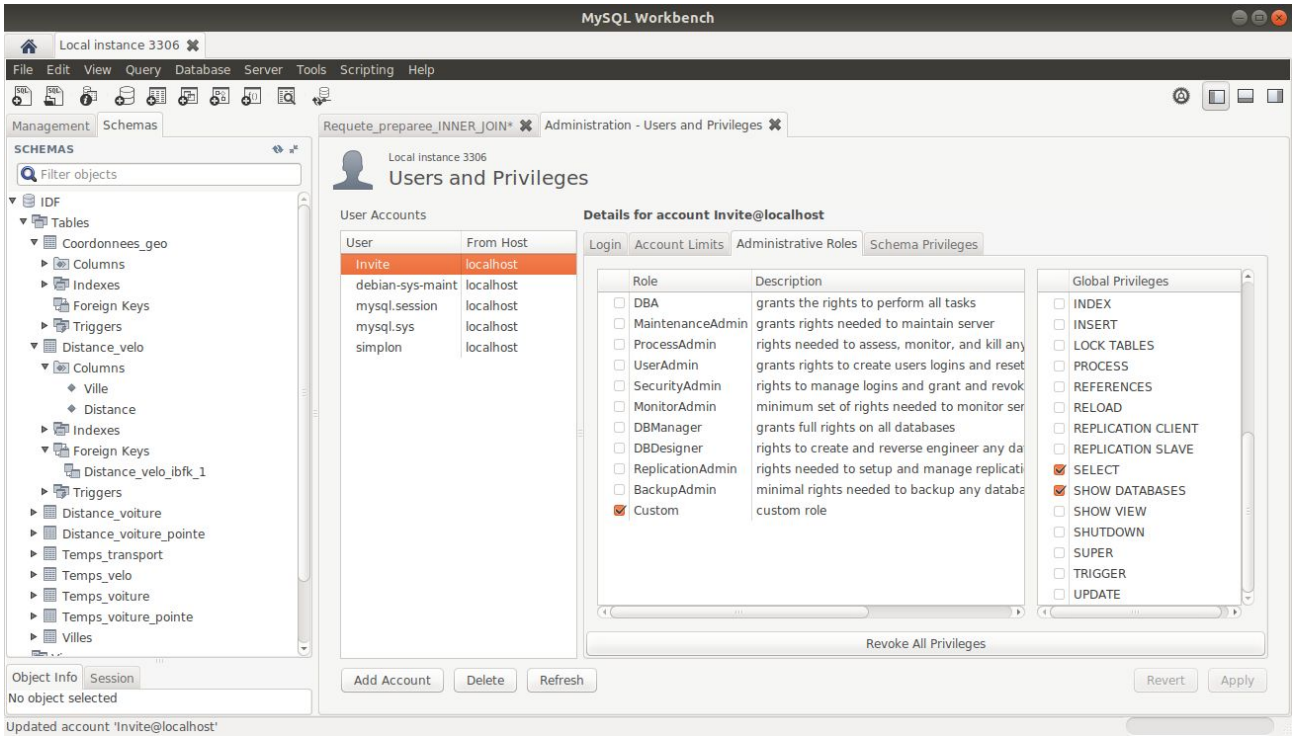
Enfin, j'ai créé un mot de passe fort pour la connexion à MySQL.

Pour restaurer la base de données :

```
mysql -u simplon -p IDF <
/home/pierre/Documents/chef_d_oeuvre/BDD/MySQL/Dump/20191217-121542/IDF.SQL
```

Voici ce que ça donne sur Workbench :





3. Les dataframes

Afin de pouvoir exploiter les données pour de la DataViz, j'ai dû importer les données présentes sur la base de données MySQL avec Python, afin de créer des dataframes :

```
userSQL="xxxxx"
passwordSQL="xxxxx "
databasesql="IDF"

# créer un moteur et connexion à la base de donnée IDF :
engine = create_engine("mysql+pymysql://{user}:{pw}@localhost/{db}"
                       .format(user=userSQL,
                               pw=passwordSQL,
                               db=databasesql,
```

```
                                charset='utf8'))

#lecture de la table Villes en dataframe :
df_villes_sql = pd.read_sql_table("Villes", con=engine)
```

Par la suite, j'ai dû nettoyer ces données afin qu'elles soient exploitables.

Par exemple, j'ai retiré la partie 'km' dans les colonnes de distances pour ne garder qu'un chiffre ou convertir '1 heure 30 minutes' en minutes -> '90' :

```
def convertir_en_minutes (x):
    i=-1
    for cases in x.iloc[:,1]:
        i=i+1
        if str(x.iloc[i,1]).find('day')!=-1:
            x.iloc[i,2]=str(x.iloc[i,1]).replace(' days ','+').replace('
day ','+').replace(' hours', '').replace(' hour','')
            liste_jour=str(x.iloc[i,2]).split('+')
            if len(liste_jour)==1 and liste_jour[0]!='nan':
                x.iloc[i,2]=int(liste_jour[0])*1440
            if len(liste_jour)==2:
                x.iloc[i,2]=int(liste_jour[0])*1440+int(liste_jour[1])*60
            if liste_jour[0]=='nan':
                x.iloc[i,2]=float('NaN')
        else:
            x.iloc[i,2]=str(x.iloc[i,1]).replace(' hour ','+').replace('
hours ','+').replace(' mins', '').replace(' min','')
            liste_heure=str(x.iloc[i,2]).split('+')
            if len(liste_heure)==1 and liste_heure[0]!='nan':
                x.iloc[i,2]=int(liste_heure[0])
            if len(liste_heure)==2:
                x.iloc[i,2]=int(liste_heure[0])*60+int(liste_heure[1])
            if liste_heure[0]=='nan':
                x.iloc[i,2]=float('NaN')
```

J'ai aussi créé de nouvelles colonnes avec tous les KPI pertinents afin de créer la DataViz.

Une fois tous les dataframes créés, j'ai jugé utile de les réunir en un seul et de le sauvegarder sous MongoDB avec une collection pour chaque ville.

V. Rendu visuel

Pour la DataViz, j'ai utilisé les librairies Pandas, Matplotlib, Wordcloud et Folium.

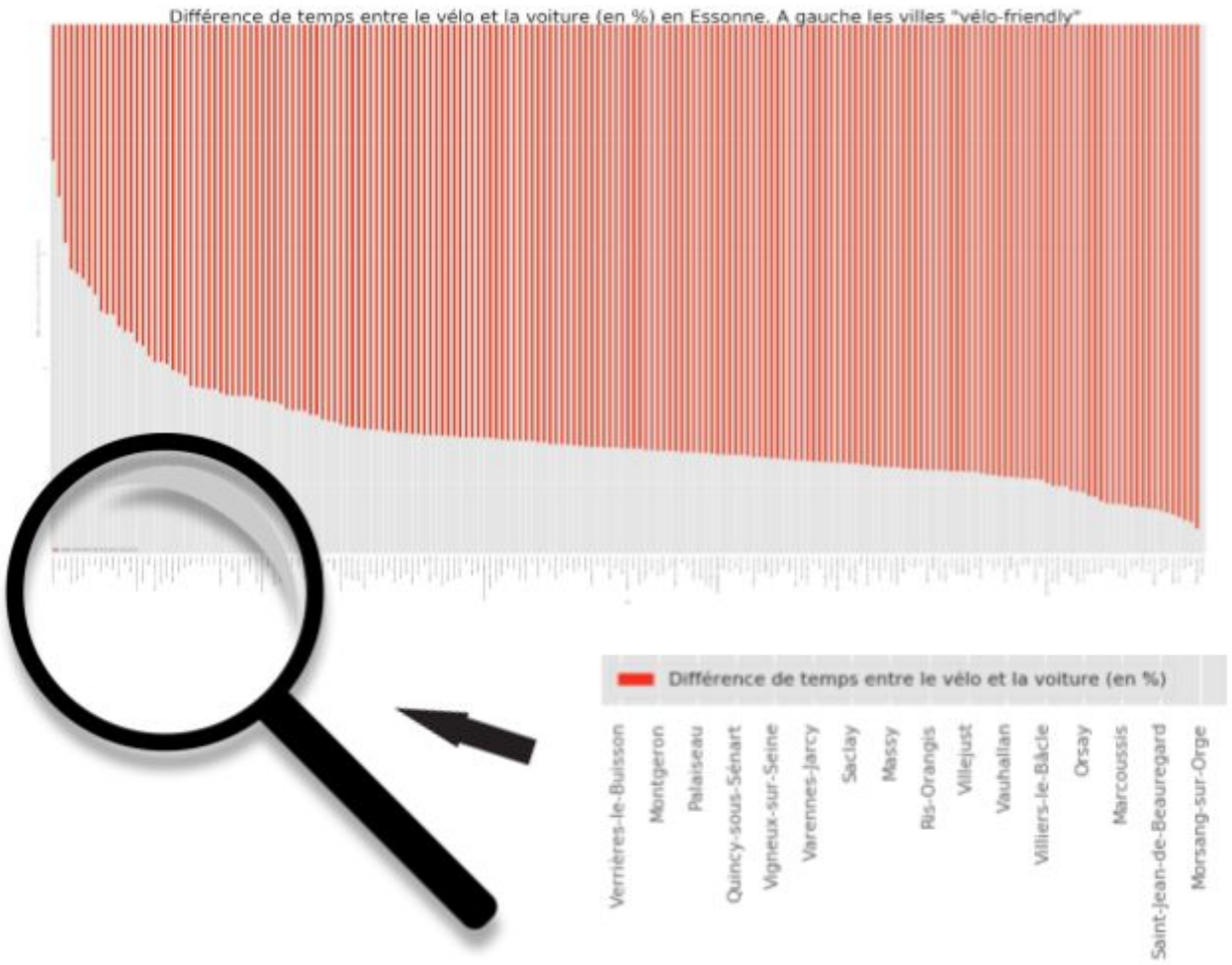
J'ai utilisé les dataframes que j'avais créés.

Tous les graphes, diagrammes et camemberts ont été exportés au format jpeg ou HTML dans le dossier 'static' ou 'templates' pour une utilisation future dans Flask.

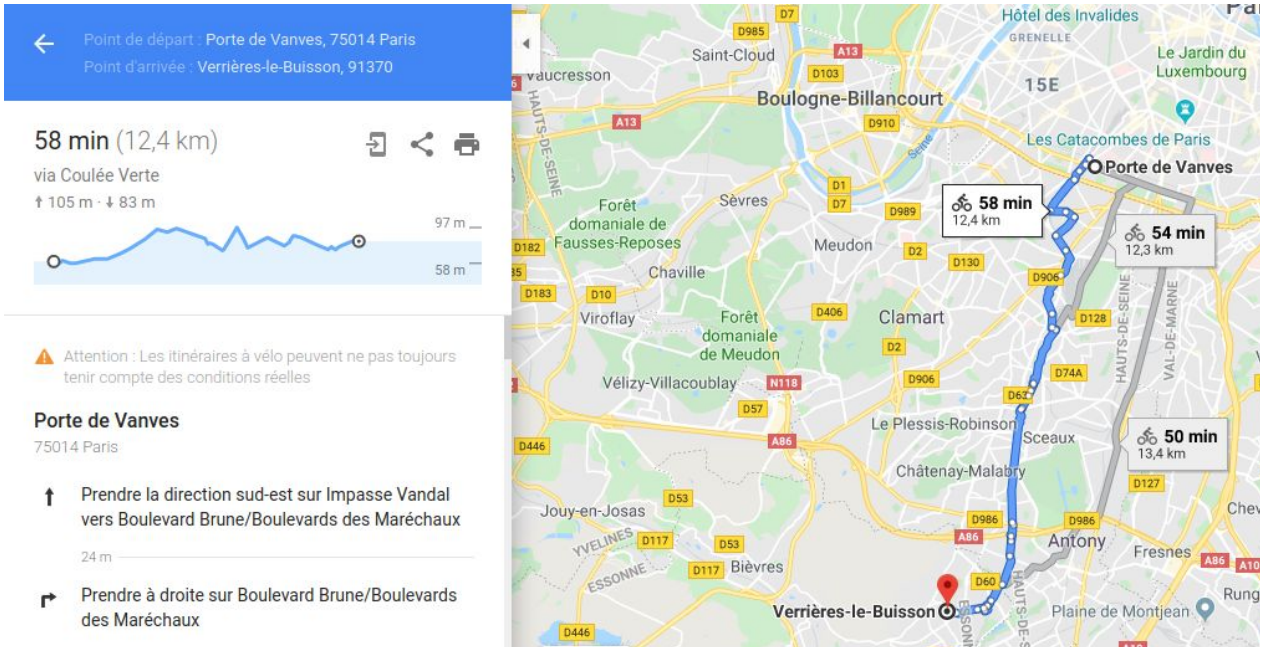
1. Vérification des KPI :

Pour ce qui est de l'analyse de la véracité des KPI, j'ai repris les graphes créés pour l'analyse et j'ai vérifié que les résultats de ces graphes étaient concordants avec la réalité, en vérifiant manuellement les données sur Google Maps et en en expliquant les causes.

Par exemple, concernant le graphe « Différence de temps de parcours entre le vélo et la voiture en pourcentage» pour l'Essonne, j'ai remarqué que la ville de "Verrières-le-Buisson" se distinguait particulièrement :



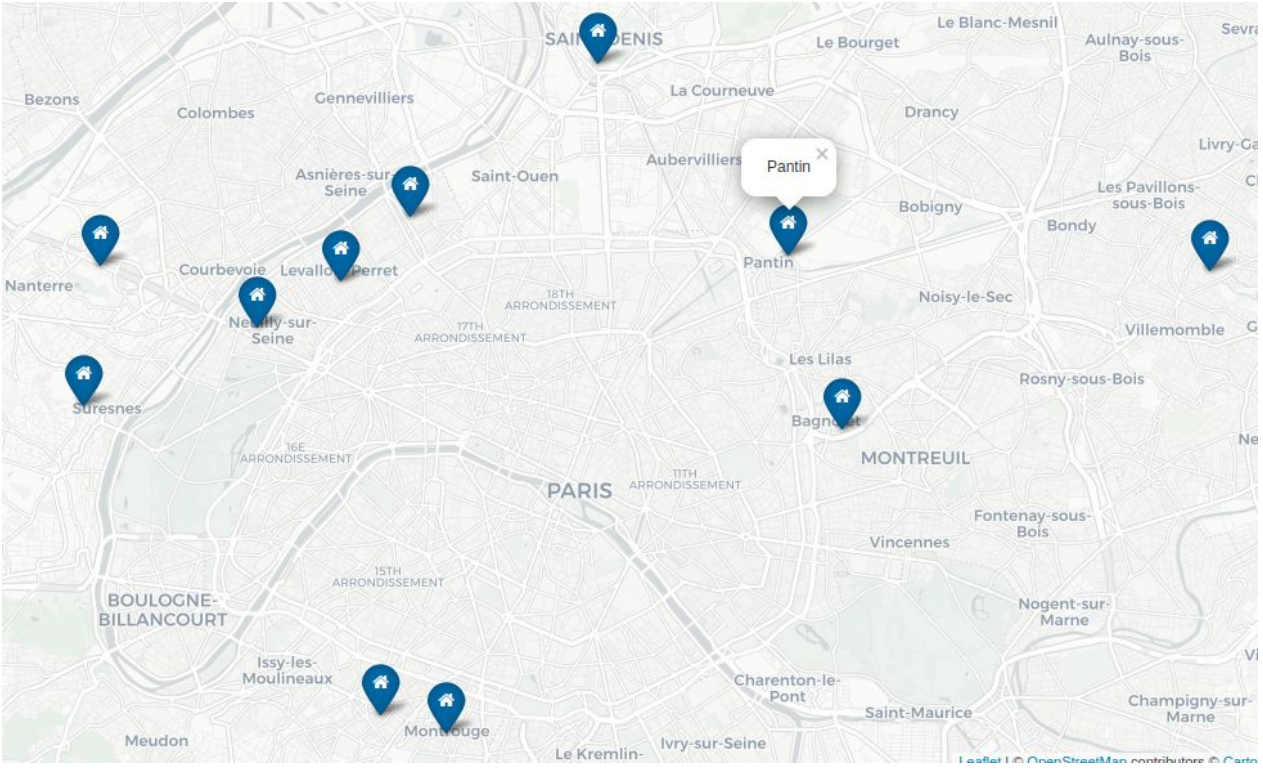
La raison est simple : Il y a une coulée verte entre Paris et Verrières-le-Buisson :



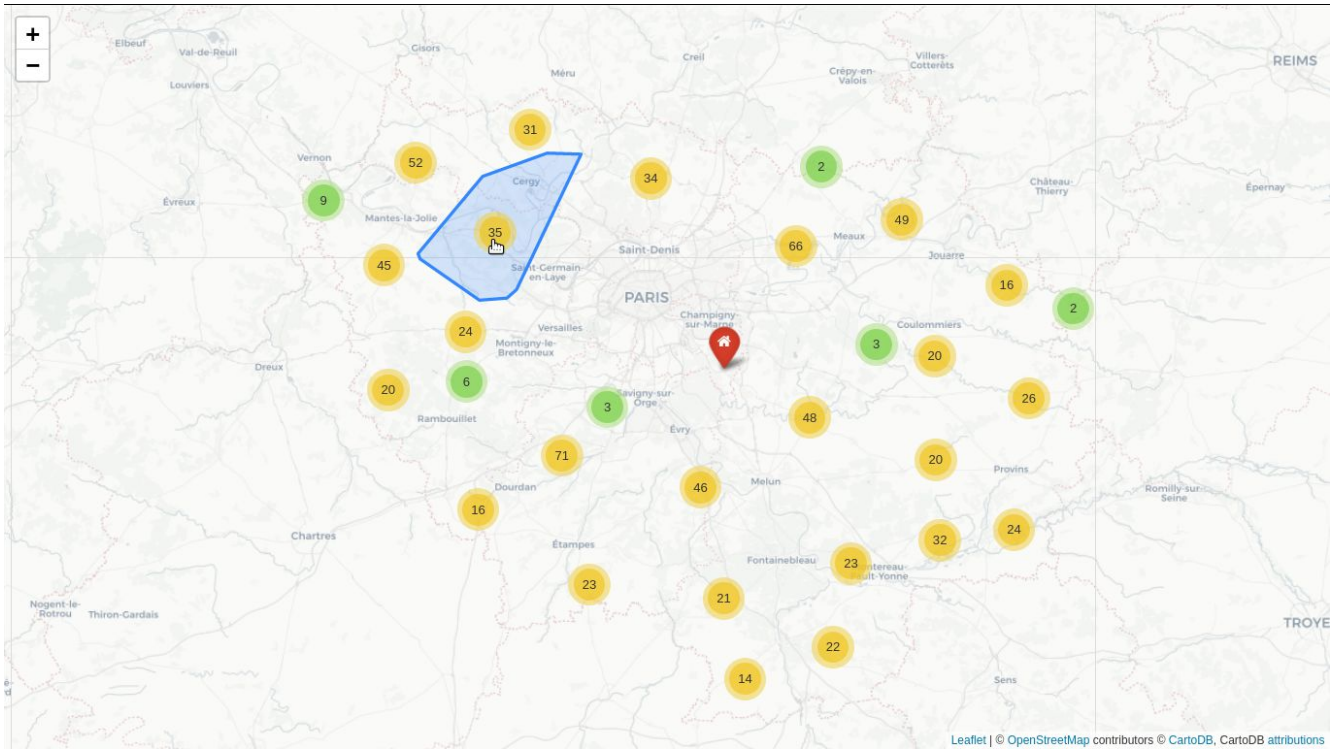
VI. Conclusions de l'analyse

Pour présenter les conclusions de l'analyse, j'ai opté pour des cartes faites avec Folium sur Flask qui montrent les villes ayant les meilleurs ratios en général, les villes sans moyen de transport et les villes ayant les moins bons KPI :

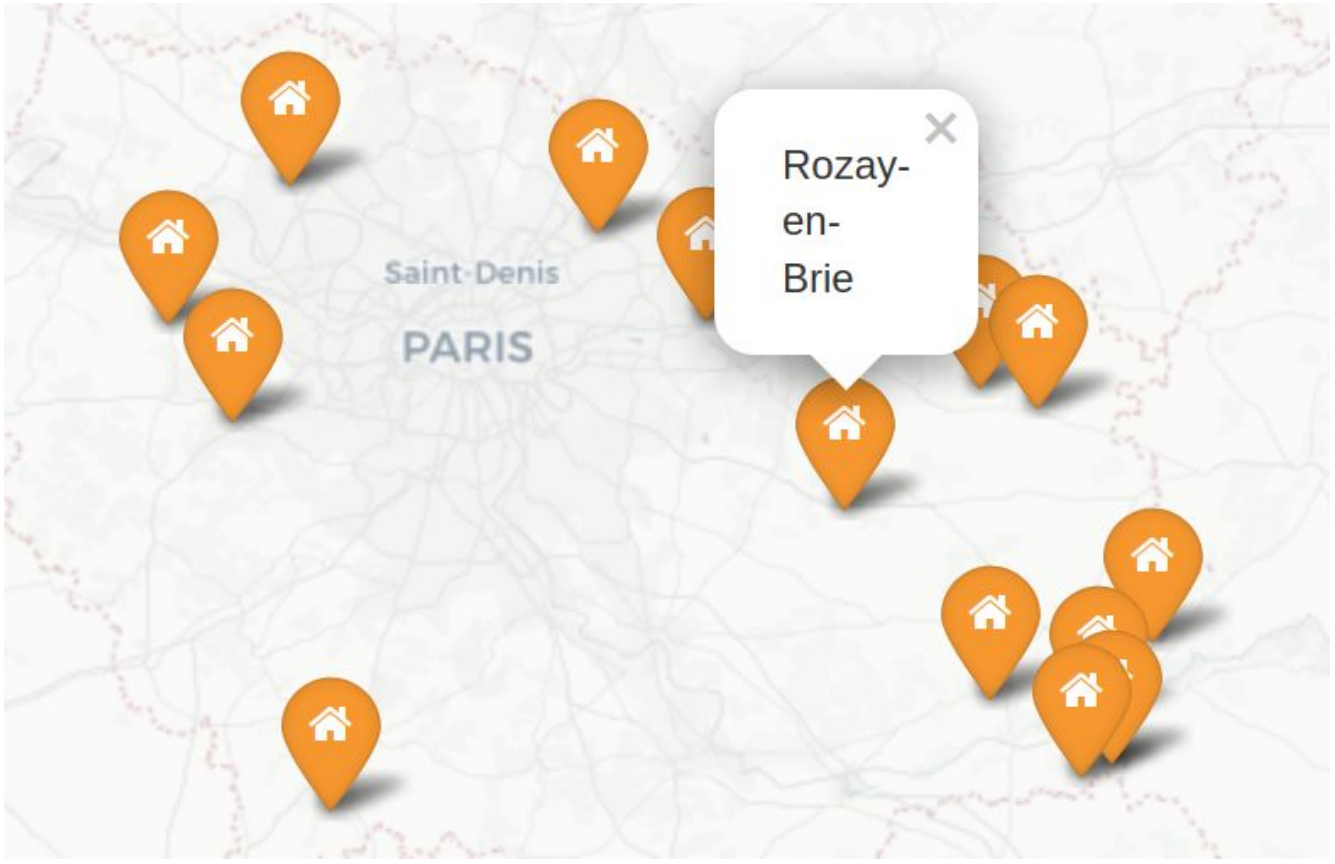
1. Les villes idéales



2. Les villes sans moyen de transport



3. Les villes ayant les moins bons KPI



VII. Pour aller plus loin...

A la suite de ce travail, il faudra analyser ces résultats afin de définir des politiques à mettre en place en termes de transport en Ile-de-France.

Il faudra expliquer pour quelles raisons il existe de telles disparités entre les villes et les départements, et définir des solutions afin d'y remédier. Elles peuvent être réglementaires (comme imposer la construction de pistes cyclables) ou économiques (comme allouer un budget aux transports).

Il serait intéressant de comparer ces résultats avec les budgets alloués aux transports des villes et des départements (par habitant) et vérifier s'il existe une corrélation.

Il pourrait aussi être pertinent de comparer ces résultats avec les taux de chômage par ville pour vérifier l'impact des transports sur l'économie.

Concernant la mise à disposition des résultats, il serait judicieux de créer une page web avec Folium qui interroge la base de données MongoDB ou SQLite et sur laquelle on peut effectuer des requêtes prédéfinies pour consulter les différents KPI.

Il est à noter que les données proviennent uniquement de GoogleMaps et qu'elles ne reflètent pas exactement la réalité des transports en commun en Ile de France. Il s'agit ici de se faire une idée générale. En investiguant plus précisément, on remarque que certaines villes apparaissent comme n'ayant aucun moyen de transport en commun alors qu'elles ont des services de transports municipaux qui ne sont pas pris en compte par GoolgleMaps.

ANNEXES

Voici les différents KPI obtenus :

Données générales :

- Nombre de villes par département
- Distance de Paris
- Temps de parcours en transport, en voiture aux heures de pointe et en trafic “normal”, en vélo

En Transports en commun :

- Ratio distance/temps en transport en commun
- Analyse des temps en transport en commun selon les départements
- Part des villes sans moyen de transport-en-commun par département
- Ratio distance/présence de moyens de transport
- Calcul de la corrélation entre la distance vers Paris et la présence de transports en commun
- Part des villes ayant un temps en transport-en-commun vers Paris supérieur à une heure par département

En Voiture :

- Villes ayant un temps Voiture>Transport
- Ratio distance/temps en voiture par villes
- Différence de temps de parcours entre heure de pointe et trafic normal
- Différence de distance entre heure de pointe et trafic normal
- Part des villes ayant un temps en voiture vers Paris supérieur à une heure

En Vélo :

- Différence de distance entre le vélo et la voiture : Présence de pistes cyclables
- Différence de temps de parcours entre le vélo et la voiture

Outils utilisés

Nom de l'outil	Utilisation	Raisons
MySQL	Langage utilisé pour la création de la base de données à la suite du scraping	Données structurées et relationnelles
JavaScrypt	Langage utilisé pour la création de la base de données finale dans MongoDB	Données volumineuses et organisées par Ville
Python	Langage utilisé pour la création des programmes	Langage performant et facile à utiliser Nombreuses librairies
MongoDB	Consultation des données	Ergonomique Facilité d'utilisation
MySQL Workbench	Diagramme conceptuel Consultation des données	Ergonomique Facilité d'utilisation
Jupyter Notebook	Dataviz Création des programmes Python	Permet de visualiser le rendu final Programmation facilitée par l'utilisation de cases
Editeur de texte **Notepad**	Ouverture de fichiers CSV, exploration des données	Préinstallé sur mon ordinateur Rapidité au lancement
Google Docs, Slide **LibreOffice** **Office Word, Powerpoint**	Création du rapport final	
Google Colab	Permet de travailler sur les programmes Python à distance	Pas besoin d'emmener avec soi son ordinateur
Github **Google Drive**	Sauvegarde et mise à disposition du projet	Gratuit et performant
Trello	Organisation du travail	Ergonomique Facilité d'utilisation

Librairies Python utilisées

Pour la création des différents programmes Python, j'ai utilisé de nombreuses librairies :

- Pour le scraping :

BeautifulSoup et requests pour scraper Wikipédia
googlemaps pour interroger l'API GoogleMaps
pandas et csv pour la création des CSV
shutil pour déplacer les anciens CSV

- Pour l'affichage des barres de progression qui permettent de visualiser l'avancement du programme et d'estimer sa durée d'exécution :

tqdm

- Pour l'envoi des mails :

Smtplib

- Pour la Dataviz :

Matplotlib pour la création des graphes
pandas, math et numpy pour les dataframes
folium pour la création de la carte interactive
seaborn pour le calcul des corrélations
wordcloud pour l'analyse sémantique

- Pour l'intégration dans les bases de données :

pymongo et json
pymysql

- Pour l'affichage du site web:

flask

Exemple d'exécution du programme de scraping

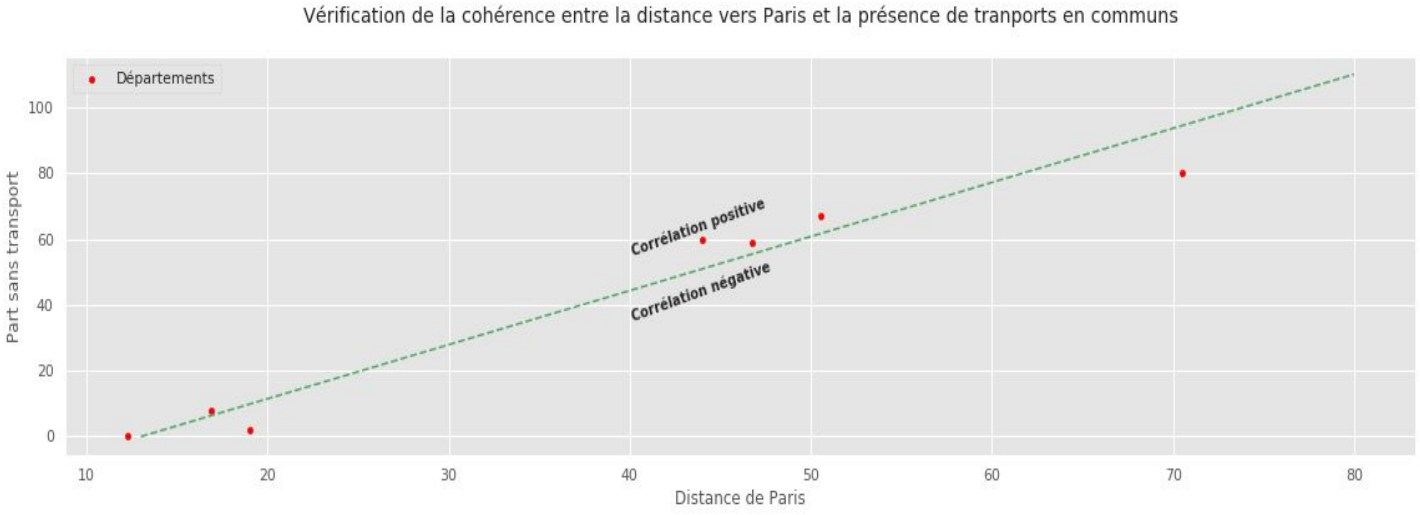
```
PROGRAMME DEMARRE A : 13:46:54 , LE 24/10/2019

I. Scraping des villes d'Île de France sur Wikipédia

    Scraping des villes d'Essonne
100%|████████████████████████████████| 194/194 [00:00<00:00, 147461.94it/s]
    Nettoyage de la liste 'essonne'
100%|████████████████████████████████| 194/194 [00:00<00:00, 670259.45it/s]
    Scraping des villes des Hauts-de-Seine
100%|████████████████████████████████| 36/36 [00:00<00:00, 75535.24it/s]
    Nettoyage de la liste 'hauts_de_seine'
100%|████████████████████████████████| 36/36 [00:00<00:00, 178270.30it/s]
    Scraping des villes de Seine-et-Marne
100%|████████████████████████████████| 507/507 [00:00<00:00, 172145.40it/s]
    Nettoyage de la liste 'seine_et_marne'
100%|████████████████████████████████| 507/507 [00:00<00:00, 1004493.21it/s]
    Scraping des villes de Seine-Saint-Denis
100%|████████████████████████████████| 40/40 [00:00<00:00, 95651.17it/s]
    Nettoyage de la liste 'seine_saint_denis'
100%|████████████████████████████████| 40/40 [00:00<00:00, 201165.66it/s]
    Scraping des villes du Val-de-Marne
100%|████████████████████████████████| 47/47 [00:00<00:00, 91391.88it/s]
    Nettoyage de la liste 'val_de_marne'
100%|████████████████████████████████| 47/47 [00:00<00:00, 179374.24it/s]
    Scraping des villes du Val-d'Oise
100%|████████████████████████████████| 184/184 [00:00<00:00, 143129.07it/s]
    Nettoyage de la liste 'val_d_oise'
100%|████████████████████████████████| 47/47 [00:00<00:00, 208826.58it/s]
    Scraping des villes des Yvelines
100%|████████████████████████████████| 262/262 [00:00<00:00, 162009.09it/s]
    Nettoyage de la liste 'yvelines'
100%|████████████████████████████████| 262/262 [00:00<00:00, 740503.81it/s]

Création des CSVs de la liste des villes pour chaque régions:

    CSV pour les Yvelines créé
    CSV pour le Val d'Oise créé
    CSV pour le Val de Marne créé
    CSV pour la Seine-Saint-Denis créé
    CSV pour la Seine et Marne
    CSV pour les Hauts-de-Seine créé
    CSV pour l'Essonne créé

Suppression des anciens CSV :

    Nettoyage du dossier : csv_villes
    Anciens CSV déplacés vers le dossier : anciens_csv/anciens_csv_villes
```

```
II. Scraping des temps de trajet et des distances vers Place du Châtelet à Paris
pour chacune des villes d'Île de France, suivant différents modes de transport :

- Transports en commun
- Voiture : Trafic normal et heure de pointe
- Vélo

Scraping des temps de trajet en transport en Essonne :
 48%|████████████████████                      | 94/194 [00:36<00:37,
2.70it/s]NOT_FOUND pour la ville : Le Mérévillois
100%|██████████████████████████████████████████| 194/194 [01:15<00:00,  2.56it/s]
    CSV temps en transports pour l'Essonne créé
Scraping des temps de trajet en transport dans les Hauts de Seine :
100%|██████████████████████████████████████████| 36/36 [00:17<00:00,  1.95it/s]
    CSV temps en transports pour les Hauts de Seine créé
Scraping des temps de trajet en transport dans la Seine et Marne :
 69%|█████████████████████████████             | 350/507 [02:10<01:02,
2.53it/s]NOT_FOUND pour la ville : Penchard
100%|██████████████████████████████████████████| 507/507 [03:08<00:00,  2.98it/s]
    CSV temps en transports pour la Seine et Marne créé
Scraping des temps de trajet en transport dans les Yvelines :
100%|██████████████████████████████████████████| 262/262 [01:37<00:00,  3.01it/s]
    CSV temps en transports pour les Yvelines créé

Suppression des anciens CSV transport:

    Nettoyage du dossier : temps/temps_transports
    Anciens CSV déplacés vers le dossier : anciens_csv/anciens_csv_transport

Mail de confirmation de la bonne exécution du programme envoyé !

PROGRAMME TERMINE A : 14:22:49, LE 24/10/2019

Temps d'exécution du programme :  35 minutes
```

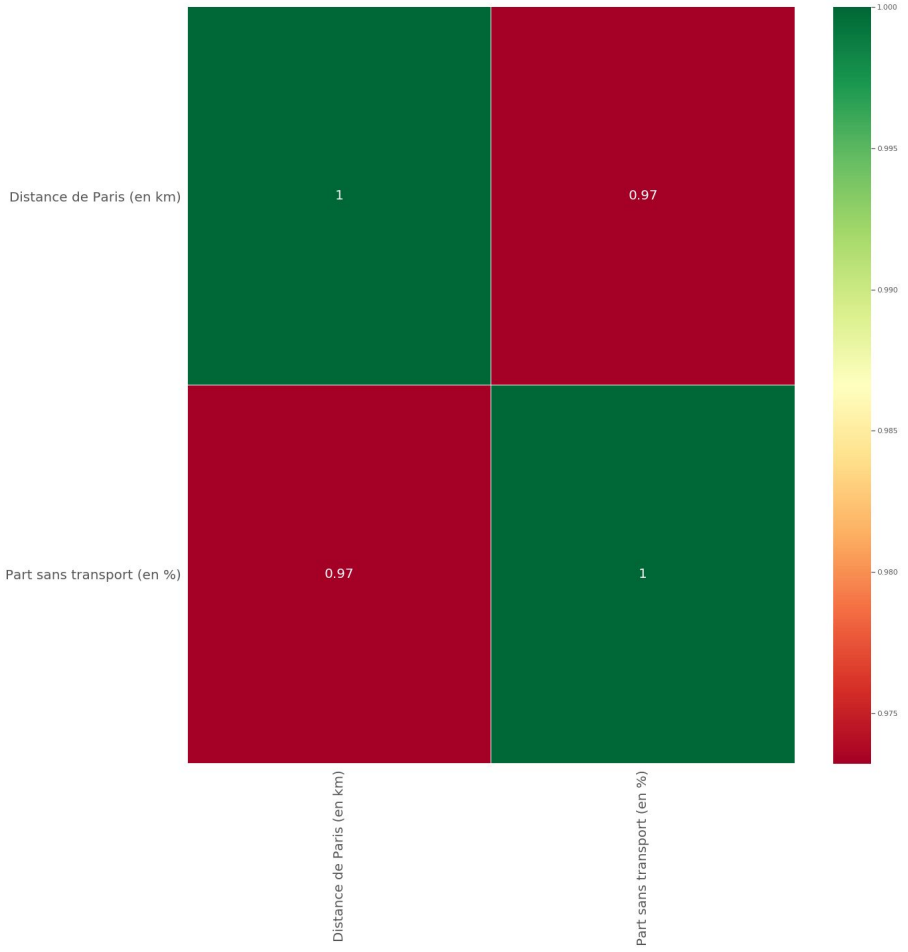
Exemple d'exécution du programme d'intégration des données dans les bases de données

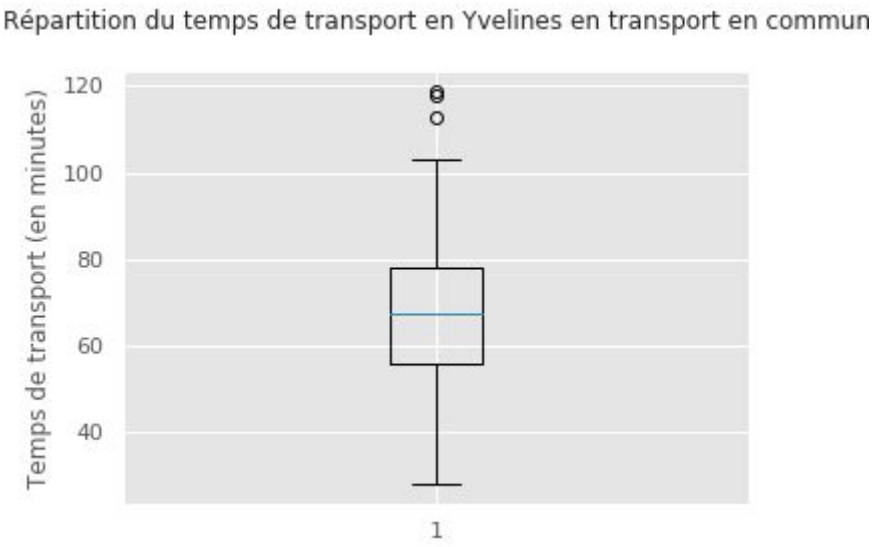
```
(base) pierre@simplon-thinkpad-t430:~/Documents/chef_d_oeuvre$ python
"/home/pierre/Documents/chef_d_oeuvre/2 Integrer dans une BDD et mettre a
disposition en format Python.py"
Veuiller entrer le mot de passe SMTP :
PROGRAMME DEMARRE A : 12:54:45, LE 13/12/2019
I. Exportation des CSVs vers une base de donnée mySQL :
Connected to DB: localhost
Attribution des droits pour MySQL pour charger les CSVs
Création de la base de données IDF
Création de la table 'Coordonnees_geo'
Intégration des données dans la table
Création de la table 'Villes'
Intégration des données dans la table
Création de la table 'Distance_voiture'
Intégration des données dans la table
Création de la table 'Distance_voiture_pointe'

Toutes les données ont été intégrées dans la base de donnée MySQL 'IDF' !
Appuyer sur Enter

II. Importation des données non traitées dans MongoDb :

Connecté à MongoDb !

La BDD 'IdF' a été créée !

La collection 'Données_non_traitees' a été créée !

Suppression des anciennes bases de données

Création des documents

Le document Json : tps_val_de_marne_transport20191120_132105.json a été intégré
dans la base de données MongoDb 'IdF' dans la collection 'Données_non_traitees'
100%|████████████████████████████████████| 28/28 [01:09<00:00,  2.98s/it]


Description de la DB :
- 14 documents ont été intégrés dans la base de données MongoDb 'IdF'
dans la collection 'Données_non_traitees'

- Affichage des stats de la BDD 'IdF' :

{'db': 'IdF', 'collections': 2, 'views': 0, 'objects': 1283, 'avgObjSize':
8437.219017926735, 'dataSize': 10824952.0, 'storageSize': 6656000.0, 'numExtents':
0, 'indexes': 2, 'indexSize': 90112.0, 'fsUsedSize': 109620690944.0,
'fsTotalSize': 313445539840.0, 'ok': 1.0}
```

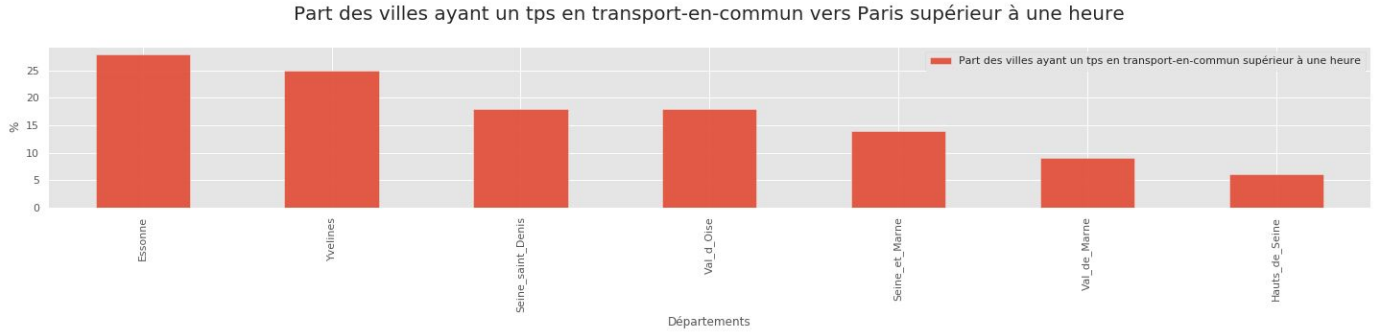
Quelques graphes parlants :

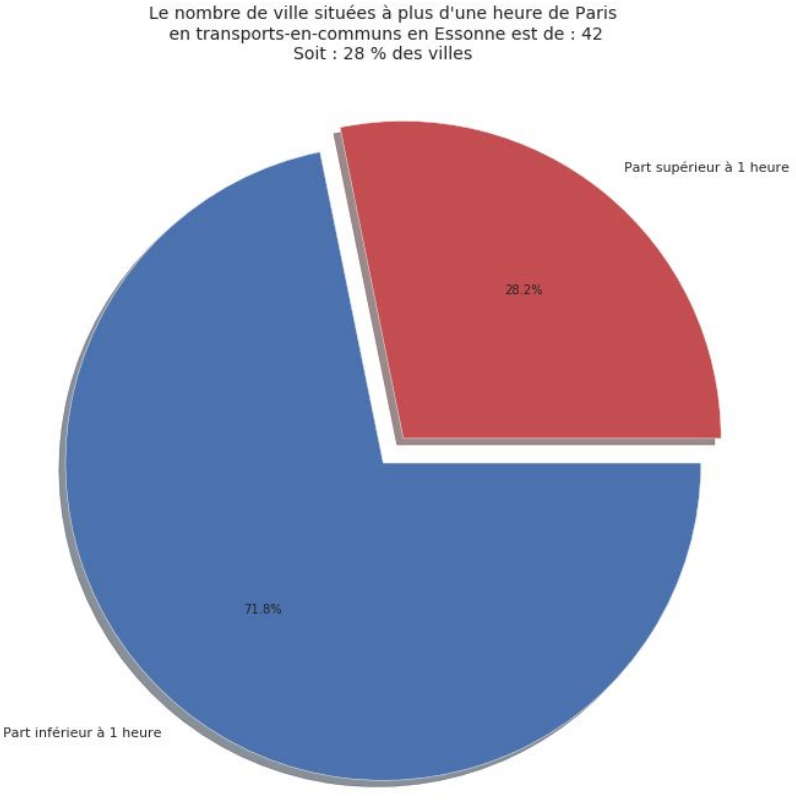


Corrélation entre distance vers Paris et la présence de transports en commun :




Répartition du temps de transport en Yvelines en transport en commun
Temps de transport (en minutes)
120
100
80
60
40
1


Part des villes ayant un tps en transport-en-commun vers Paris supérieur à une heure
Part des villes ayant un tps en transport-en-commun supérieur à une heure
%
25
20
15
10
5
0
Essonne
Yvelines
Seine_saint_Denis
Val_d_Oise
Seine_et_Marne
Val_de_Marne
Hauts_de_Seine
Départements


Le nombre de ville situées à plus d'une heure de Paris
en transports-en-communs en Essonne est de : 42
Soit : 28 % des villes
Part supérieur à 1 heure
28.2%
71.8%
Part inférieur à 1 heure

Villes ayants les moins bons KPI en terme de transport vers Paris

Villes ayants les meilleurs caractéristiques en terme de transport vers Paris

REMERCIEMENTS

Je tiens à remercier nos nombreux formateurs qui nous ont accompagnés durant ces sept mois : **Manel Boumaiza, Sayf Bejaoui, David Azria et Yacine Aslimi.**

Notre responsable de la fabrique : **Kalidou Niang** et tous les intervenants de chez Simplon

Je remercie également mon maître de stage, **Valentin Brondel** CEO de Jus Mundi qui a su m'accueillir dans son entreprise et m'a donné des tâches intéressantes à produire.

Enfin, je remercie tous les apprenants de la promotion 2019 de Développeur Data de Nanterre avec qui j'ai passé une excellente formation et qui m'ont permis de progresser.