ENSAI - SID

Technologies NoSQL

Projet final de NoSQL

Auteur :
Lydia VIJAYARAJAH

11 février 2018

1 - Définition de l'endroit idéal

Le projet consiste à trouver le meilleur endroit où s'installer à New York à partir des jeux de données disponibles sur le site Open Data NYC. Pour cela je vais définir ce que je considère comme étant l'endroit idéal où habiter. Une fois les critères de l'endroit idéal choisit, je vais chercher les jeux de données correspondants sur le site puis les mettre dans une des bases NoSQL vues en cours.

Les aspects qui me semblent les plus déterminants sont :

— **La sécurité :** Il est important d'être dans un lieu sûr, tout particulièrement lorsqu'on arrive dans une nouvelle ville, dans un nouveau pays. Le jeu de données que je vais utiliser pour juger de la sûreté sera le jeu de données sur les crimes commis de 2016 à maintenant.
— **Les espaces verts :** J'aimerai de plus habiter dans un endroit avec de nombreux arbres. Il existe un jeu de données répertoriant les arbres de New York ainsi que leur état que j'utiliserai.
— **L'hygiène :** Ayant une peur bleue des rongeurs, je préfererai qu'il n'y en ai pas aux alentours de mon lieu de résidence. Pour cela j'utilise le jeu de données listant les inspections liées aux rongeurs
— **Les loisirs :** J'ai l'habitude d'aller à la bibliothèque afin d'emprunter des livres dans mon temps libre. J'apprécierai donc avoir une bibliothèque à côté de chez moi. J'utilise pour vérifier cela le jeu de données des bibliothèques de la ville.
— J'ai décidé de récupérer le jeu de données donnant le nombre d'habitants par Borough pour pouvoir comparer les nombres de crimes par Borough.

jeu de données sur les crimes
jeu de données sur les arbres
jeu de données sur les rongeurs
jeu de données sur les bibliothèques

2 - Téléchargements et imports

Il n'est pas nécessaire de lancer chaque fichier décrit par la suite, il suffira de lancer **config.sh** pour que chaque fichier se lancent automatiquement. Ce fichier permet aussi le téléchargement de python et de tous les modules dont nous aurons besoins par la suite.

```
################################################################################
#             Installation de pythons et des modules nécessaires               #
################################################################################

sudo apt-get install python
sudo apt-get install python-pip
sudo pip install --upgrade pymongo requests
```

FIGURE 1 – Téléchargement de python et de ses modules
config.sh

2.1 - Téléchargement des données

Le téléchargement des différents jeux de données se fait par le biais du fichier **downloadData.py**. Ce fichier python permet de télécharger toutes les données au format csv, chaque fichiers csv sera placés dans un répertoire nommé "downloads".

```
def download_file(url, output_file):

    headers = {}
    r = requests.get(url, headers=headers, stream=True)

    with open(output_file, 'wb') as f:
        for chunk in r.iter_content(chunk_size=4096):
            if chunk:
                f.write(chunk)
        f.flush()

    return output_file
```

FIGURE 2 – Méthode utilisée pour télécharger les données
downloadData.py

2.3 - La base de données

J'ai décidé de travailler avec la base de données MongoDB. En effet, en plus d'être celle que je maîtrise le plus, la base de données MongoDB est orientée documents. Il sera donc plus facile de traiter nos données avec MongoDB. J'ai pu constater après quelques recherches que MongoDB est de plus une des bases les plus utilisées (base NoSQL la plus populaire selon Improgrammer) et il serait donc intéressant de l'utiliser dans le cadre du projet.

Pour utiliser MongoDB je passerai par le biais du module pymongo disponible sur python. L'installation de MongoDB et son démarrage se font dans le script **config.sh**.

```
#############################################################################
#                          Installation de MongoDB                          #
#############################################################################

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
0C49F3730359A14518585931BC711F9BA15703C6

echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-
org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

sudo apt-get update

sudo apt-get install -y mongodb-org

sudo service mongod start
```

FIGURE 3 – Téléchargement de MongoDB et démarrage
config.sh

Les données téléchargées doivent être importées sur MongoDB. Cela est fait par le biais du fichier python **loadData.py** qui utilise aussi le fichier **allDAO.py** pour faire le lien entre la base et python.

```
cnx_string = "mongodb://localhost:27017"
cnx = pymongo.MongoClient(cnx_string)
database = cnx.NewYorkCity

print("Chargement des données dans MongoDB (peut prendre quelques minutes)")

print("loading Population ...")
populations = allDAO.PopulationDAO(database)
with open('./downloads/Population.csv','r') as csvfile:
    populationsReader = csv.reader(csvfile, delimiter=',', quotechar='"')
    next(populationsReader)
    for row in populationsReader:
        #print ', '.join(row)
        data = json.loads("{}")
        data["Age Group"] = row[0]
        data["Borough"] = row[1]
        data["2020"] = row[16]
        populations.insert(data)

print("")
```

FIGURE 4 – Chargement de Populations.csv dans MongoDB
loadData.py

3 - Trouver le meilleur endroit

Maintenant que les données sont importées dans MongoDB, on peut les analyser afin de trouver l'endroit idéal à New York.

3.1 - La sécurité

Je vais chercher à restreindre le champs des recherches en retirant certains Boroughs. Pour cela je commence par compter le nombre de crimes par Borough. Cependant ce chiffre n'a pas beaucoup de sens si on ne le compare pas au nombre d'habitants par Borough.

Les résultats renvoyés par les requêtes sont sous la forme de liste de dictionnaire. On peut donc récupérer le nombre d'habitants (estmation pour 2020) comme ci-dessous :

```
popsQuery = list(population.find({},{'Borough':1,'2020':1,'_id':0} ))

result = dict()

for pq in popsQuery :
    result[pq['Borough'][3:].upper()] = float(pq['2020'])
```

FIGURE 5 – Habitants par Borough
Requests.py

On peut maintenant comparer le nombre de crimes par Boroughs en divisant le résulat par le nombre d'habitants.

```
felsQuery = list(felonies.aggregate([
        {'$group': {'_id': "$BORO_NM", 'tot':{ "$sum": 1}}}
]))

for fq in felsQuery :
    result[fq['_id']] = float(fq['tot'])/result[fq['_id']]
```

FIGURE 6 – Crimes par habitant
Requests.py

On garde les 3 Boroughs ayant le plus faible nombre de crimes par habitant : Queens, Staten Island et Brooklyn.

3.2 - Les espaces verts

On restreint donc la recherche à ces trois Boroughs (que l'on a mis dans une liste nommée "meilleurs"). Mon deuxième critère de sélection repose sur le taux d'arbres en bon état dans les Boroughs, cela me permettra de retirer un Borough du champs de recherche.

```
totTrees = trees.find({'borough':{'$in':meilleurs},'status':"Alive"}).count()

treesQuery = list(trees.aggregate([
    {'$match':{'borough':{'$in':meilleurs},'status':"Alive"}},
    {'$group':{'_id' : "$borough",'count':{'$sum':1}}},
    {'$project':{'percent':{'$divide':["$count",totTrees]}
    }
  },
    {'$sort':{'percent':-1}}
]))

for tq in treesQuery :
    result[tq['_id']] = float(tq['percent'])
```

FIGURE 7 – Taux d'arbres en bon état
Requests.py

On ne garde plus que le Queens et Brooklyn (on change donc la valeur de "meilleurs").

3.3 - L'hygiène

Je vais chercher maintenant à cibler un peu plus l'endroit idéal en ne cherchant plus par Borough mais par ZipCode.

On compte le nombre de rats par ZipCode pour les ZipCodes du Queens et de Brooklyn.

```
ratsQuery = list(rodents.aggregate([
    {'$match':{'BOROUGH':{'$in':meilleurs}}},
    {'$group':{'_id' : "$ZIP_CODE",'count':{'$sum':1}}},
    {'$sort':{'count':-1}}
]))
```

FIGURE 8 – Nombre de rats par ZipCode
Requests.py

3.4 - Les loisirs

On enlève les ZipCodes pour lesquels il n'y a pas de bibliothèques.Je veux maintenant attribuer un score aux ZipCodes en fonction du nombre de rats dans le ZipCode et du nombre de bibliothèques. Pour cela je récupère parmi les nombres de rats par ZipCode, le nombre maximum. Je considère que celui-ci vaut -5. On calcule le score comme suit :

$$maxRats = \max_{zip \in zipCodes} (\text{nombre de rats}_{zip}) \tag{1}$$

$$Score(Zip) = \text{nombre de bibliothèques}_{zip} - \frac{\text{nombre de rats}_{zip} * 5}{maxRats} \tag{2}$$

Cela va nous permettre de classer les ZipCodes.

```
for zipcode in libsQuery:
    if zipcode['_id'] in result.keys():
        resultlib[zipcode['_id']]=-(float(result[zipcode['_id']]*5)/float(37147))+zipcode['count']

resultlib = OrderedDict(sorted(resultlib.items(), key=itemgetter(1)))
```

FIGURE 9 – Attribution du score
Requests.py

On garde les 3 meilleurs ZipCodes : 11234, 11229 et 11223. Après vérification, ces trois ZipCodes sont dans Brooklyn et plutôt proche les uns des autres. Il serait donc conseillé pour moi de chercher dans ces quartiers.

Il serait intéressant de regarder les données au format GeoJSON pour introduire une notion de distance. On pourrait aussi vérifier qu'il y ait effectivement des stations de métro aux alentours de ces quartiers.