Section 3 Premiers pas : 1ère base

```
C:\MongoDB\bin>mongosh
Current Mongosh Log ID: 6818706ddadd1a6fb6b5f898
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.5.0
Using MongoDB:          8.0.9
Using Mongosh:          2.5.0

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/


To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

------
   The server generated these startup warnings when booting
   2025-05-05T09:57:14.527+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
------

test>
```

Créer une base

```
test> use mesclients
switched to db mesclients
mesclients> db
mesclients
mesclients>
```

Créer un utilisateur

```
mesclients> db.createUser(
... {
... user: "valentin",
... pwd: "0000",
... roles: [ { role: "readWrite", db: "config" }]
... }
... );
{ ok: 1 }
mesclients>
```

Créer une collection

```
mesclients> db.createCollection("clients")
{ ok: 1 }
```

Insérer les documents – équivalent de lignes en SQL

```
mesclients> db.clients.insertOne([
...     {
...         "nom": "Mac Cain",
...         "prénom": "John",
...         "tel": 12345678
...       }
... ])
{
  acknowledged: true,
  insertedId: ObjectId('68187429dadd1a6fb6b5f89a')
}
mesclients>
```

```
mesclients> db.clients.insertMany([
...     {
...     "nom": "Massonniere",
...     "prénom": "Valentin",
...     "tel": 123456789
...   },
...   {
...     "nom": "Dufour",
...     "prénom": "Alain",
...     "tel": 123456789,
...     "société": "Altran",
...     "niveau": 10
...   }
...   ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('681874c4dadd1a6fb6b5f89b'),
    '1': ObjectId('681874c4dadd1a6fb6b5f89c')
  }
}
mesclients>
```

Pour regarder le contenu actuel :

```
mesclients> db.clients.find()
[
  {
    _id: ObjectId('68187403dadd1a6fb6b5f899'),
    nom: 'Mac Cain',
    'prénom': 'John',
    tel: 12345678
  },
  {
    '0': { nom: 'Mac Cain', 'prénom': 'John', tel: 12345678 },
    _id: ObjectId('68187429dadd1a6fb6b5f89a')
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89b'),
    nom: 'Massonniere',
    'prénom': 'Valentin',
    tel: 123456789
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89c'),
    nom: 'Dufour',
    'prénom': 'Alain',
    tel: 123456789,
    'société': 'Altran',
    niveau: 10
  }
]
mesclients>
```

Modifier un document

1. Ajouter à Mac Cain la société Microsof

```
mesclients> db.clients.updateOne(
... { _id: ObjectId('68187429dadd1a6fb6b5f89a')},
... {
...     $set: {
...     société: "Microsoft",
...     }
... }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mesclients> db.clients.find()
[
  {
    _id: ObjectId('68187403dadd1a6fb6b5f899'),
    nom: 'Mac Cain',
    'prénom': 'John',
    tel: 12345678
  },
  {
    '0': { nom: 'Mac Cain', 'prénom': 'John', tel: 12345678 },
    _id: ObjectId('68187429dadd1a6fb6b5f89a'),
    'société': 'Microsoft'
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89b'),
    nom: 'Massonniere',
    'prénom': 'Valentin',
    tel: 123456789
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89c'),
    nom: 'Dufour',
    'prénom': 'Alain',
    tel: 123456789,
    'société': 'Altran',
    niveau: 10
  }
]
mesclients>
```

2. Enlever la donnée niveau à Dufour

```
mesclients> db.clients.updateOne(
...     { _id: ObjectId('681874c4dadd1a6fb6b5f89c')},
...     {
...         $unset: {
...         niveau: 10,
...         }
...     }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mesclients> db.clients.find()
[
  {
    _id: ObjectId('68187403dadd1a6fb6b5f899'),
    nom: 'Mac Cain',
    'prénom': 'John',
    tel: 12345678
  },
  {
    '0': { nom: 'Mac Cain', 'prénom': 'John', tel: 12345678 },
    _id: ObjectId('68187429dadd1a6fb6b5f89a'),
    'société': 'Microsoft'
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89b'),
    nom: 'Massonniere',
    'prénom': 'Valentin',
    tel: 123456789
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89c'),
    nom: 'Dufour',
    'prénom': 'Alain',
    tel: 123456789,
    'société': 'Altran'
  }
]
mesclients>
```

3. Pour Dufour, renommer le champ societe en company

```
mesclients> db.clients.update(
...     { _id: ObjectId('681874c4dadd1a6fb6b5f89c') },
...     { $rename: { 'société': 'company' } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mesclients> db.clients.find()
[
  {
    _id: ObjectId('68187403dadd1a6fb6b5f899'),
    nom: 'Mac Cain',
    'prénom': 'John',
    tel: 12345678
  },
  {
    '0': { nom: 'Mac Cain', 'prénom': 'John', tel: 12345678 },
    _id: ObjectId('68187429dadd1a6fb6b5f89a'),
    'société': 'Microsoft'
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89b'),
    nom: 'Massonniere',
    'prénom': 'Valentin',
    tel: 123456789
  },
  {
    _id: ObjectId('681874c4dadd1a6fb6b5f89c'),
    nom: 'Dufour',
    'prénom': 'Alain',
    tel: 123456789,
    company: 'Altran'
  }
]
mesclients>
```

1. lister les documents de clients, uniquement pour les champs nom et tel

```
mesclients> db.clients.find({}, { nom: 1, tel: 1, _id: 0 })
[
  { nom: 'Mac Cain', tel: 12345678 },
  {},
  { nom: 'Massonniere', tel: 123456789 },
  { nom: 'Dufour', tel: 123456789 }
]
mesclients> db.clients.find(
...     {
...       $or: [
...         { nom: 'Mac Cain' },
...         { société: 'Altran' }
...       ]
...     },
...     { nom: 1, prénom: 1, _id: 0 }
...   )
[ { nom: 'Mac Cain', 'prénom': 'John' } ]
mesclients> db.clients.find(
...     {
...       $or: [
...         { nom: 'Mac Cain' },
...         { company: 'Altran' }
...       ]
...     },
...     { nom: 1, prénom: 1, _id: 0 }
...   )
[
  { nom: 'Mac Cain', 'prénom': 'John' },
  { nom: 'Dufour', 'prénom': 'Alain' }
]
mesclients>
```

2. lister les nom et prenom des personnes dont le nom est Mac Cain OU la société est Altran

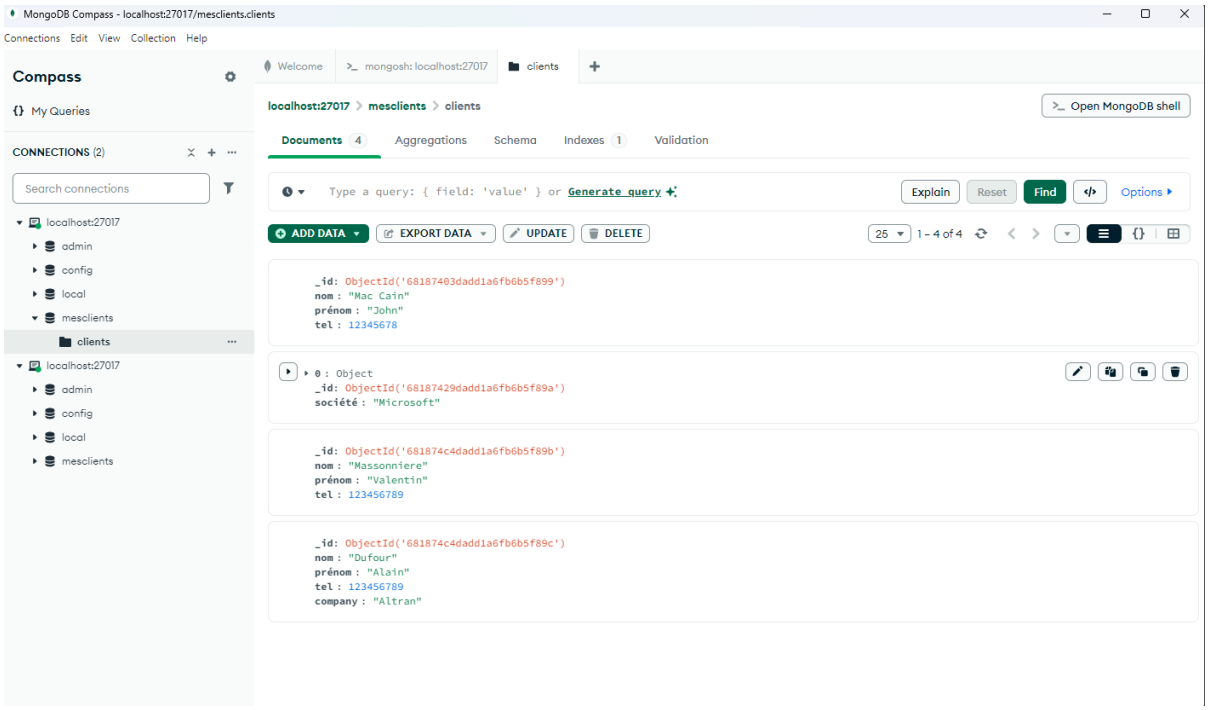
```
mesclients> db.clients.find({}, { nom: 1, tel: 1, _id: 0 })
[
  { nom: 'Mac Cain', tel: 12345678 },
  {},
  { nom: 'Massonniere', tel: 123456789 },
  { nom: 'Dufour', tel: 123456789 }
]
mesclients> db.clients.find(
...     {
...       $or: [
...         { nom: 'Mac Cain' },
...         { société: 'Altran' }
...       ]
...     },
...     { nom: 1, prénom: 1, _id: 0 }
...   )
[ { nom: 'Mac Cain', 'prénom': 'John' } ]
mesclients> db.clients.find(
...     {
...       $or: [
...         { nom: 'Mac Cain' },
...         { company: 'Altran' }
...       ]
...     },
...     { nom: 1, prénom: 1, _id: 0 }
...   )
[
  { nom: 'Mac Cain', 'prénom': 'John' },
  { nom: 'Dufour', 'prénom': 'Alain' }
]
mesclients>
```

Trier les documents par ordre croissant des noms

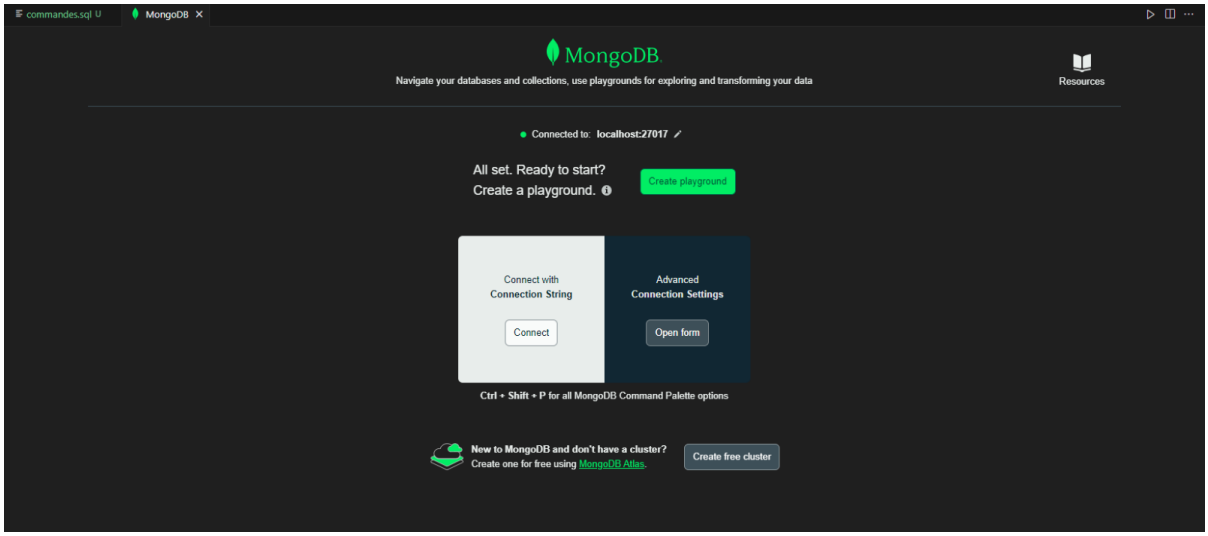
```
mesclients> db.clients.find({}, { nom: 1, tel: 1, _id: 0 }).sort({ nom: 1 })
[
  {},
  { nom: 'Dufour', tel: 123456789 },
  { nom: 'Mac Cain', tel: 12345678 },
  { nom: 'Massonniere', tel: 123456789 }
]
mesclients>
```

Section 4 Utiliser MongoDB Compass

Se logger par :



Section 5 MongoDB et VS Code

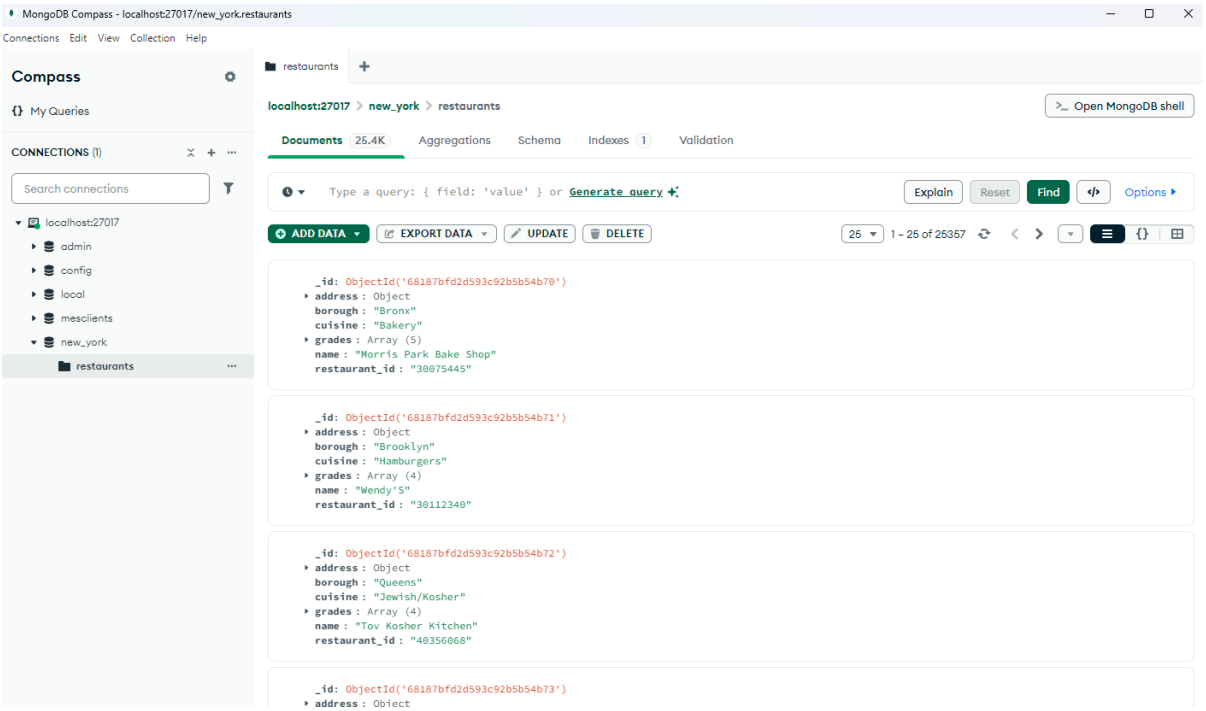


Section 6 Travail sur un jeu de données

Dé zipper le fichier.
Puis charger cette BD dans
mongoDB .
Dans la fenêtre Cmde, sortir du
client mongoDB par un exit

```
C:\MongoDB\bin>mongoimport --db new_york --collection restaurants --file "C:\MongoDB\restaurants.json"
2025-05-05T10:51:09.596+0200    connected to: mongodb://localhost/
2025-05-05T10:51:10.121+0200    25357 document(s) imported successfully. 0 document(s) failed to import.
```

Relancer dans la fenêtre Cmd le client mongoDB par mongos



1. chercher les restaurants de Manhattan dont la cuisine est American dont le code postal est 10025. Se limiter aux 5 premiers restaurants

```
> db.restaurants.find(
    {
        $and: [
            { cuisine: "American " },
            { "address.zipcode": "10025" }
        ]
    }
).limit(5)
< {
  _id: ObjectId('68187bfd2d593c92b5b54b82'),
  address: {
    building: '730',
    coord: {
      type: 'Point',
      coordinates: [
        -73.96805719999999,
        40.7925587
      ]
    },
    street: 'Columbus Avenue',
    zipcode: '10025'
  },
  borough: 'Manhattan',
  cuisine: 'American ',
  grades: [
    {
      date: 2014-09-12T00:00:00.000Z,
      grade: 'B',
      score: 26
    },
    {
      date: 2013-08-28T00:00:00.000Z,
      grade: 'A',
```

2. requête pour connaître leur nombre

```
> db.restaurants.countDocuments({
          $and: [
              { cuisine: "American " },
              { "address.zipcode": "10025" }
          ]
  });
< 46
```

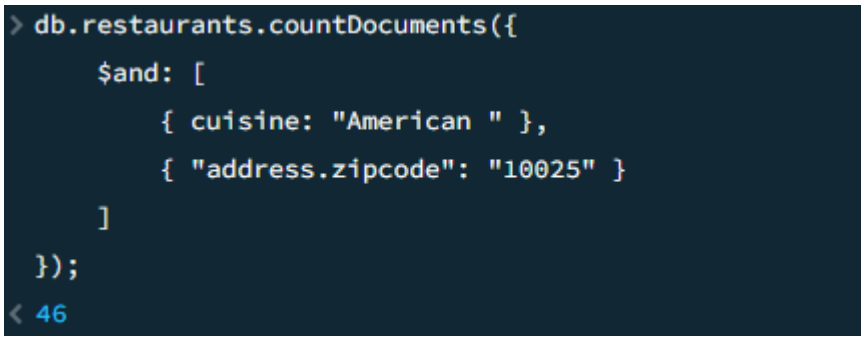
3. reprendre le résultat du 1 pour n'afficher que le nom, le voisinage (borough) et le type de cuisine

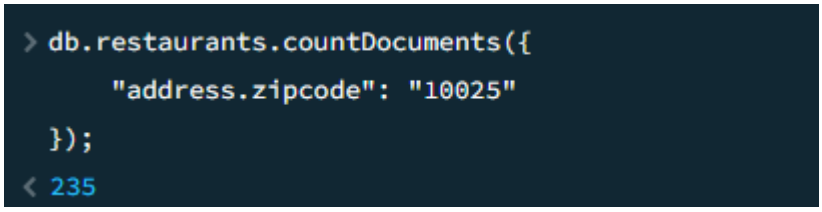
```
db.restaurants.find(
    {
        $and: [
            { cuisine: "American " },
            { "address.zipcode": "10025" }
        ]
    },
    { name: 1, borough: 1, cuisine: 1, _id: 0 }
).limit(5)
{
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'P & S Deli Grocery'
}
{
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'Spoon Bread Catering'
}
{
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'Broadway Restaurant'
}
{
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'Dive Bar'
}
{
  borough: 'Manhattan',
  cuisine: 'American ',
  name: "Tom'S Restaurant"
}
```

4. quels sont les différents type de cuisine que l'on peut trouver à New york ?

```
> use new_york
< switched to db new_york
> db.restaurants.distinct("cuisine");
< [
    'Afghan',
    'African',
    'American ',
    'Armenian',
    'Asian',
    'Australian',
    'Bagels/Pretzels',
    'Bakery',
    'Bangladeshi',
    'Barbecue',
    'Bottled beverages, including water, sodas, juices, etc.',
    'Brazilian',
    'CafÃ©/Coffee/Tea',
    'Café/Coffee/Tea',
    'Cajun',
    'Californian',
    'Caribbean',
    'Chicken',
    'Chilean',
    'Chinese',
    'Chinese/Cuban',
    'Chinese/Japanese',
    'Continental',
    'Creole',
    'Creole/Cajun',
    'Czech',
    'Delicatessen',
```

5. Quel est le pourcentage de type de cuisine « American » vis à vis des autres types ?





46*100/235 = 19,6%

Le pourcentage de type de cuisine « American » vis à vis des autres types est de 19,6%

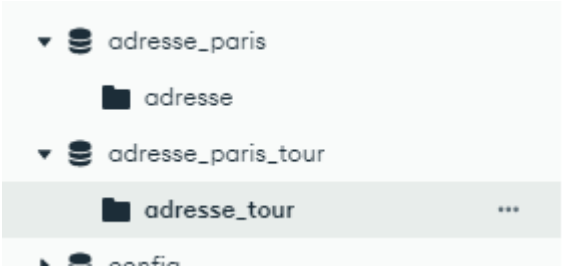
Section 8 Utilisation de Python

Pour cette solution j'ai eu 2 idées :

1. Utiliser les coordonnées gps des adresses dans les 2 base de données avec une marge d'erreur de 300m et les comparés et ajouter le zipcode.
2. Utiliser des abréviations et formatter les adresses dans un format spécifique et comparé la chaine de caractère.

J'ai choisi la première solution dans le dossier exo8.

Dans un premier temps j’ai importé les 2 bases de données



J’ai ensuite créé une classe pour établir une connexion à mongodb

```python
import pymongo
from pymongo.errors import ConnectionFailure, ConfigurationError, PyMongoError

class MongoDB:

    @staticmethod
    def get_client():
        try:
            myclient = pymongo.MongoClient("mongodb://localhost:27017/", serverSelectionTimeoutMS=5000)
            myclient.admin.command('ping')
            return myclient
        except (ConnectionFailure, ConfigurationError) as e:
            print(f"Erreur de connexion à MongoDB : {e}")
            return None

    @staticmethod
    def get_db(db_name):
        try:
            myclient = MongoDB.get_client()
            if myclient is None:
                return None
            mydb = myclient[db_name]
            return mydb
        except PyMongoError as e:
            print(f"Erreur lors de la récupération de la base de données '{db_name}' : {e}")
            return None

    @staticmethod
    def get_col(db_name, col_name):
        try:
            mydb = MongoDB.get_db(db_name)
            if mydb is None:
                return None
            mycol = mydb[col_name]
            return mycol
        except PyMongoError as e:
            print(f"Erreur lors de la récupération de la collection '{col_name}' dans la base '{db_name}' : {e}")
            return None
```

Puis dans un fichier séparé j'ai créé me permettant de récupérer les données de ses 2 tables.

ql > exo8 > mongo_requests.py > ...

```python
from bdd_connect import MongoDB
from pymongo.errors import PyMongoError

def get_adresse_paris():
    try:
        col = MongoDB.get_col("adresse_paris", "adresse")
        if col is None:
            print("Impossible de récupérer la collection 'adresse' de 'adresse_paris'")
            return []
        return list(col.find())
    except PyMongoError as e:
        print(f"Erreur lors de la récupération des adresses de Paris : {e}")
        return []

def get_adresse_paris_tour():
    try:
        col = MongoDB.get_col("adresse_paris_tour", "adresse_tour")
        if col is None:
            print("Impossible de récupérer la collection 'adresse' de 'adresse_paris_tour'")
            return []
        return list(col.find())
    except PyMongoError as e:
        print(f"Erreur lors de la récupération des adresses de Paris (tour) : {e}")
        return []
```

J’ai ensuite codé les fonctions pour obtenir la distance entre 2 coordonnées, comparer 2 coordonnées avec une marge d’erreur puis la fonction qui détecte un zip code dans un format d’adresse spécifique à notre bdd. Ensuite je dissocie les adresses en dehors du rayon de 2km et les adresses ayant déjà un code postal dans l’adresse. Pour les adresses ayant déjà un code postal, je reformatte cette adresse en enlevant ce code et en le remettant dans une key valeur à part, pour les adresses sans zipcode je compare les coordonnées de celle-ci avec toutes les coordonnées de la deuxième base de données afin de voir une correspondance.