

# Práctica 3: Mongo

Martínez Ostoia Néstor Iván  
Bases de Datos No Estructuras - 0600  
Ciencia de Datos, IIMAS, UNAM

Mayo 2021

---



## 1. Introducción

MongoDB es una base de datos orientada a documentos. La razón principal de tener este enfoque es para enfocarse en el escalamiento de manera más sencilla. Que sea orientada a documentos significa que el concepto de renglón (*row*) no existe, sino que es reemplazado por un documento. Esto permite tener estructuras de datos más complejas dentro del mismo registro pues dentro de los documentos se pueden tener documentos anidados. A continuación presento algunas de las características principales de MongoDB:

- Escalamiento: dentro de los múltiples esquemas de escalamiento, MongoDB está pensada para escalar horizontalmente, es decir, poder repartir la información dentro de múltiples servidores. MongoDB automáticamente se encarga de distribuir eficientemente la información
- Funcionalidades adicionales: MongoDB es una base de datos de propósito general, es decir, a parte de las operaciones CRUD normales, provee funcionalidades como las siguientes:
  - Indexado: creación de índices secundarios e incluso soporte para índices geoespaciales
  - Agregación: similar a los *pipelines* de procesamiento de datos, estos permiten construir motores de análisis de datos al procesar la información en pasos sencillos del lado del servidor
  - Almacenamiento de archivos: soporte para un protocolo sencillo que permite almacenar grandes archivos y meta datos

## 1.1. Conceptos clave dentro de MongoDB

- Documento: unidad básica de almacenamiento de información y cuenta con una llave única dentro de cada colección ("\_id")
- Colección: se puede pensar como una tabla con un esquema dinámico
- Una instancia de MongoDB puede almacenar múltiples bases de datos independientes

### 1.1.1. Documentos

Un documento es un conjunto ordenado de llaves junto con valores asociados para cada llave. Por ejemplo:

```
1 {"greeting" : "Hello, world", "views" : 3}
```

Una característica importante de los documentos es que no pueden contener llaves duplicadas.

### 1.1.2. Colecciones

Una colección es un conjunto de documentos con esquemas dinámicos. Es decir, los documentos dentro de una colección pueden tener múltiples formas. Por ejemplo, los siguientes dos documentos pueden pertenecer a la misma colección:

```
1 {"greeting" : "Hello, world", "views" : 3}
2 {"signoff" : "Good night"}
```

### 1.1.3. Bases de datos

MongoDB agrupa las colecciones dentro de bases de datos. Una instancia de MongoDB puede contener más de una base de datos y cada una agrupar cero o más colecciones.

### 1.1.4. Índices

Los índices en MongoDB funcionan de manera similar a los índices en las bases de datos relacionales; son creados para incrementar el rendimiento de las consultas sobre campos comunes. Debido a esto, es usualmente una buena práctica buscar los campos sobre los cuales se realizan consultas frecuentes y aquellas que necesiten ser ejecutadas en el menor tiempo posible.

De igual manera, los índices se aprovechan de mejor manera si buscamos obtener con ellos conjuntos pequeños de información. Es decir, la eficiencia de los índices decrementa conforme se incrementa el porcentaje del subconjunto de información que se quiere obtener.

## 2. Desarrollo

### 2.1. Requerimientos

Desarrollar un sistema de paseos que cumpla con lo siguiente:

1. Cada usuario podrá registrarse en el sistema dando de alta información personal junto con ubicación (coordenadas) de al menos un lugar (casa, trabajo, escuela, etc) y con opción a agregar más
2. Cada usuario podrá buscar estaciones de bicicletas más cercanas desde uno de sus lugares registrados
3. Cada usuario podrá planear viajes de un tiempo en específico (10 min, 30 min, etc). El sistema deberá recomendar viajes usando las estaciones del usuario más cercanas (o una elegida por él) y los destinos que mejor se acomoden al tiempo que el usuario indicó
4. Cada usuario podrá buscar una ruta específica (indicando estación de origen y destino). En el caso que el sistema no tenga dicha ruta, el sistema deberá armarla de forma indirecta

## 2.2. Interfaz gráfica

Para la implementación de este proyecto, decidí hacerlo simulando una aplicación móvil desde la terminal. La idea de hacerlo así fue por dos razones:

1. Mejor interfaz y experiencia para usuarios de mi sistema
2. Facilidad técnica para implementar algunos requerimientos mediante el cliente de MongoDB en Python

En la siguiente imagen muestro una captura de pantalla de la interfaz principal <sup>1</sup>

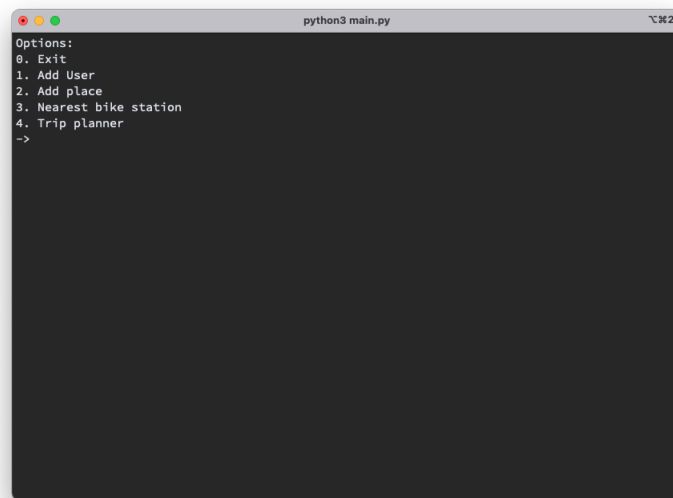


Figura 1: Para entrar a la interfaz hay que correr el script **main.py**

---

<sup>1</sup>Aunado a la ejecución de **main.py**, hay que insertar en una base de datos en MongoDB los archivos JSON de la carpeta: **mongo/citybike-data** y correr las siguientes líneas después de insertar las estaciones: **db.stations.createIndex('loc': '2dsphere')**

## 2.3. Limpieza de datos

Para la limpieza de datos me enfoque principalmente en generar dos datasets <sup>2</sup>:

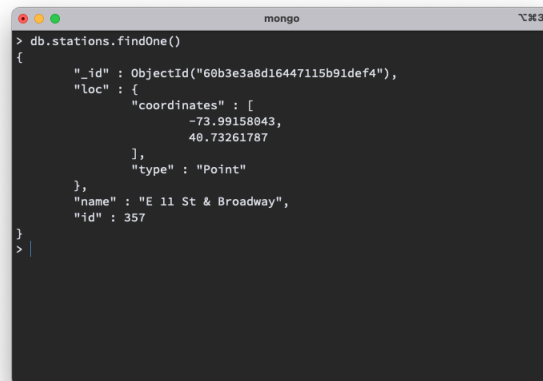
1. **Estaciones de bicicletas:** para construir este conjunto de datos <sup>3</sup> lo que hice fue tomar las estaciones de origen y destino que fueran únicas en los conjuntos de datos [originales](#) y escribirlo a un archivo JSON
2. **Viajes:** para construir el conjunto de datos tomé los datos originales y simplemente seleccioné las columnas de interés: 'trip\_duration', 'start\_time', 'stop\_time', 'start\_station\_id' 'end\_station\_id'

## 2.4. Inserción de datos

Tanto para las estaciones como los viajes, en la limpieza de datos (sección [2.3](#)) generé varios archivos JSON y los inserté directo a una base de datos en MongoDB.

### 2.4.1. Verificación

#### ■ Estaciones



```
mongo
> db.stations.findOne()
{
  "_id" : ObjectId("60b3e3a8d16447115b91def4"),
  "loc" : {
    "coordinates" : [
      -73.99158043,
      40.73261787
    ],
    "type" : "Point"
  },
  "name" : "E 11 St & Broadway",
  "id" : 357
}
```

#### ■ Viajes

---

<sup>2</sup>Revisar [mongo/data-cleaning.ipynb](#)

<sup>3</sup>[mongo/citybike-data/bike\\_stations.json](#)

```
mongo
> db.trips.findOne()
{
  "_id" : ObjectId("60b41b6cac4b58767bfa062"),
  "trip_duration" : 1547,
  "start_time" : "2013-07-01 00:00:02",
  "stop_time" : "2013-07-01 00:25:49",
  "start_station_id" : 388,
  "end_station_id" : 459
}
```

## 2.5. Implementación de requerimientos

### 2.5.1. Inserción de usuarios

1. Registro de nuevos usuarios

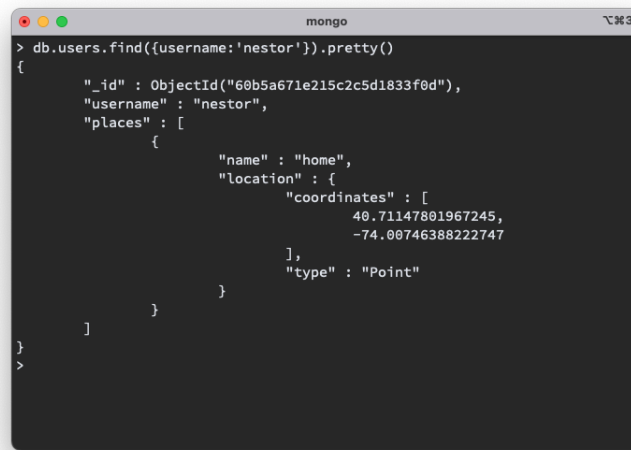
```
python3 main.py
Options:
0. Exit
1. Add User
2. Add place
3. Nearest bike station
4. Trip planner
-> 1
```

```
python3 main.py
Options:
0. Return
1. Add user details
-> 1
```

```
python3 main.py
Username: nestor
Place name: home
Latitude (only numbers): 40.71147801967245
Longitude (only numbers): -74.00746388222747

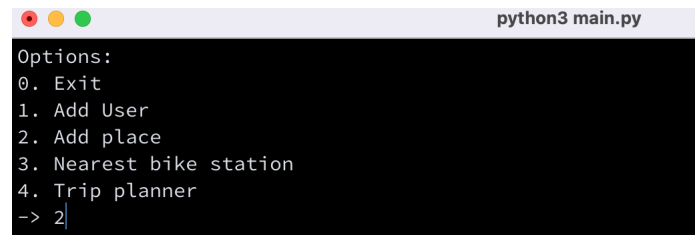
Is data correct?
0. No
1. Yes
-> 1
```

Verificación:

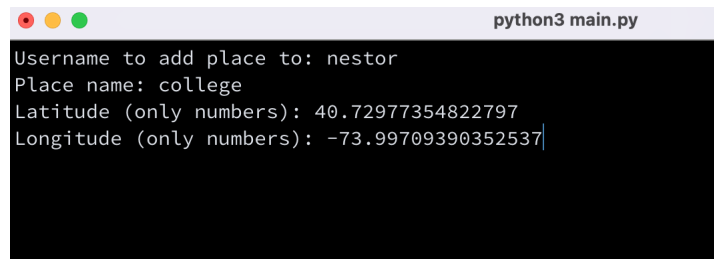


```
mongo
> db.users.find({username:'nestor'}).pretty()
{
  "_id" : ObjectId("60b5a671e215c2c5d1833f0d"),
  "username" : "nestor",
  "places" : [
    {
      "name" : "home",
      "location" : {
        "coordinates" : [
          40.71147801967245,
          -74.00746388222747
        ],
        "type" : "Point"
      }
    }
  ]
}
```

## 2. Adición de más lugares



```
python3 main.py
Options:
0. Exit
1. Add User
2. Add place
3. Nearest bike station
4. Trip planner
-> 2
```



```
python3 main.py
Username to add place to: nestor
Place name: college
Latitude (only numbers): 40.72977354822797
Longitude (only numbers): -73.99709390352537
```

Verificación:

```
mongo
> db.users.find({username:'nestor'}).pretty()
{
  "_id" : ObjectId("60b5a671e215c2c5d1833f0d"),
  "username" : "nestor",
  "places" : [
    {
      "name" : "home",
      "location" : {
        "coordinates" : [
          40.71147801967245,
          -74.00746388222747
        ],
        "type" : "Point"
      }
    },
    {
      "name" : "college",
      "location" : {
        "coordinates" : [
          40.72977354822797,
          -73.99709390352537
        ],
        "type" : "Point"
      }
    }
  ]
}
```

Query:

```
1 self.db['users'].update_one({'username': username}, {
2     '$push': {
3         'places': {
4             'name': place,
5             'location': {
6                 'coordinates': [latitude, longitude],
7                 'type': 'Point'
8             }
9         }
10    }
11 })
```

### 2.5.2. Búsqueda de estaciones más cercanas

```
python3 main.py
Options:
0. Exit
1. Add User
2. Add place
3. Nearest bike station
4. Trip planner
-> 3
```

```
python3 main.py ㄿ#2
Username: nestor

Showing nestor's places (select one):
1. home
2. college
-> |
```

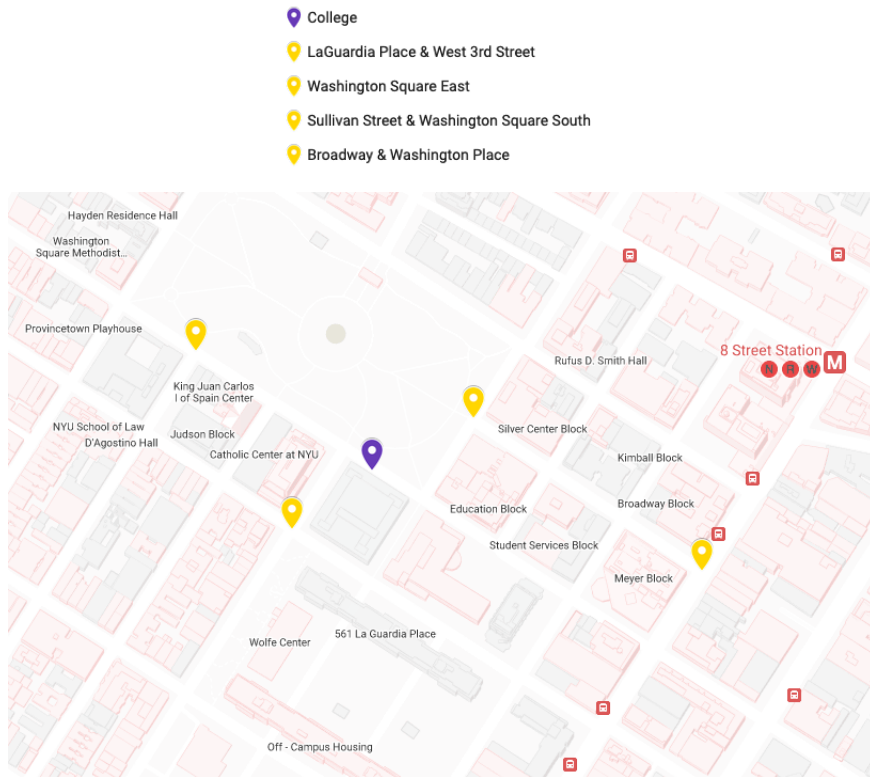
```
python3 main.py ㄿ#2
Username: nestor

Showing nestor's places (select one):
1. home
2. college
-> 2

Showing nearest stations to college:
1. LaGuardia Pl & W 3 St
   108.4 m
2. Washington Square E
   140.8 m
3. Sullivan St & Washington Sq
   183.5 m
4. Washington Pl & Broadway
   269.8 m
5. MacDougal St & Washington Sq
   302.3 m
6. Mercer St & Bleeker St
   331.8 m
7. Great Jones St
   381.4 m
8. Washington Pl & 6 Ave
   383.4 m
9. E 10 St & 5 Ave
   429.0 m
10. Carmine St & 6 Ave
    431.9 m
Press any key to continue |
```

Verificación:





Query:

```
1 stations = self.client.db['stations']
2 result = stations.aggregate([
3     {
4         "$geoNear": {
5             "near": {"type": "Point", "coordinates": [lon, lat]},
6             "key": "loc",
7             "distanceField": "dist",
8             "spherical": True,
9             "maxDistance": 500
10        },
11    },
12    {"$limit": 10}
13 ])
```

### 2.5.3. Planeación de viajes

Para esta sección consideraré la hora en la que el usuario busca planear su viaje para ofrecer resultados más precisos. A continuación las capturas de pantalla de este requerimiento:

```
python3 main.py ㄿ%2
Options:
0. Exit
1. Add User
2. Add place
3. Nearest bike station
4. Trip planner
-> 4|
```

```
python3 main.py ㄿ%2
Select a method to start your trip:
1. Specific bike station
2. Nearest bike stations
3. Nearest bike stations to your places
->
```

■ Búsqueda de una estación en específico

```
python3 main.py ㄿ%2
Select a method to start your trip:
1. Specific bike station
2. Nearest bike stations
3. Nearest bike stations to your places
-> 1

Bike station name: E 11 St & Broadway|
```

```
python3 main.py
Select a method to start your trip:
1. Specific bike station
2. Nearest bike stations
3. Nearest bike stations to your places
-> 1

Bike station name: E 11 St & Broadway

How long do you want your trip to last? (minutes): 30

Fetching trips [Time=23:01:16]...

Trip #1: [23.0 min]
  E 11 St & Broadway -> E 12 St & 3 Ave in 2.5 minutes and [2290.92 m]
  E 12 St & 3 Ave -> Concord St & Bridge St in 23.0 minutes and [6061.14 m]
-----

Trip #2: [26.5 min]
  E 11 St & Broadway -> Lafayette St & E 8 St in 2.3 minutes and [2543.47 m]
  Lafayette St & E 8 St -> W 39 St & 9 Ave in 26.5 minutes and [1130.85 m]
-----

Trip #3: [33.6 min]
  E 11 St & Broadway -> Washington Pl & Broadway in 1.9 minutes and [2768.54 m]
  Washington Pl & Broadway -> W 47 St & 10 Ave in 33.6 minutes and [1524.05 m]
-----
```

Figura 2: El sistema arrojará hasta 5 opciones de viajes distintos enlistando las estaciones dentro de cada viaje, así como la distancia recorrida en cada estación y el tiempo transcurrido. Cuando no hay viajes directos, el sistema arrojará viajes con una tolerancia de  $\pm 10$  minutos

- Búsqueda por estaciones más cercanas

```
python3 main.py
Select a method to start your trip:
1. Specific bike station
2. Nearest bike stations
3. Nearest bike stations to your places
-> 2

Your current latitude: 40.71293477090227
Your current longitude: -74.01338596045515

Showing nearest stations:
1. Murray St & West St
   229.7 m
2. South End Ave & Liberty St
   255.1 m
3. Barclay St & Church St
   268.6 m
4. Greenwich St & Warren St
   331.8 m
5. Park Pl & Church St
   341.6 m
6. Vesey Pl & River Terrace
   379.9 m
7. Warren St & Church St
   413.3 m
8. Liberty St & Broadway
   498.5 m

Bike station: 1
```

Figura 3: Las estaciones mostradas en esta imagen son las 10 estaciones más cercanas al edificio *One World Trade Center* en Nueva York

```
python3 main.py
Bike station: 1

How long do you want your trip to last? (minutes): 60

Fetching trips [Time=23:03:10]...

Trip #1: [67.2 min]
  Murray St & West St -> Barclay St & Church St in 2.2 minutes and [4980.1 m]
  Barclay St & Church St -> Broadway & W 49 St in 38.4 minutes and [984.23 m]
  Broadway & W 49 St -> Warren St & Church St in 67.2 minutes and [4757.32 m]
-----

Trip #2: [70.3 min]
  Murray St & West St -> Vesey Pl & River Terrace in 1.7 minutes and [5040.29 m]
  Vesey Pl & River Terrace -> Johnson St & Gold St in 26.5 minutes and [6382.29 m]
  Johnson St & Gold St -> W 4 St & 7 Ave S in 70.3 minutes and [2695.71 m]
-----

Trip #3: [84.1 min]
  Murray St & West St -> West St & Chambers St in 2.0 minutes and [4675.56 m]
  West St & Chambers St -> Centre St & Chambers St in 31.2 minutes and [4787.89 m]
  Centre St & Chambers St -> Hicks St & Montague St in 50.1 minutes and [6451.9 m]
  Hicks St & Montague St -> Dean St & 4 Ave in 84.1 minutes and [7677.37 m]
-----
```

■ Búsqueda por lugares de usuario

```
python3 main.py
Select a method to start your trip:
1. Specific bike station
2. Nearest bike stations
3. Nearest bike stations to your places
-> 3
username: nestor

Showing nestor's places (select one):
1. home
2. college
-> 2
```

```
python3 main.py

Showing nestor's places (select one):
1. home
2. college
-> 2

Showing nearest stations to college:
1. LaGuardia Pl & W 3 St
   108.4 m
2. Washington Square E
   140.8 m
3. Sullivan St & Washington Sq
   183.5 m
4. Washington Pl & Broadway
   269.8 m
5. MacDougal St & Washington Sq
   302.3 m
6. Mercer St & Bleecker St
   331.8 m
7. Great Jones St
   381.4 m
8. Washington Pl & 6 Ave
   383.4 m
9. E 10 St & 5 Ave
   429.0 m
10. Carmine St & 6 Ave
    431.9 m

Number of bike station: 8
```

```
python3 main.py ㄿ2

Number of bike station: 8

How long do you want your trip to last? (minutes): 45

Fetching trips [Time=23:05:55]...

Trip #1: [44.0 min]
  Washington Pl & 6 Ave -> MacDougal St & Prince St in 2.4 minutes and [3310.47 m]
  MacDougal St & Prince St -> Rivington St & Ridge St in 44.0 minutes and [3738.53 m]
-----

Trip #2: [55.8 min]
  Washington Pl & 6 Ave -> Washington Square E in 1.7 minutes and [2678.94 m]
  Washington Square E -> Emerson Pl & Myrtle Ave in 31.6 minutes and [6707.16 m]
  Emerson Pl & Myrtle Ave -> Metropolitan Ave & Bedford Ave in 55.8 minutes and [4467.
56 m]
-----

Trip #3: [60.6 min]
  Washington Pl & 6 Ave -> Washington Pl & Broadway in 2.7 minutes and [2768.54 m]
  Washington Pl & Broadway -> W 47 St & 10 Ave in 34.4 minutes and [1524.05 m]
  W 47 St & 10 Ave -> Sullivan St & Washington Sq in 60.6 minutes and [2816.76 m]
-----
```

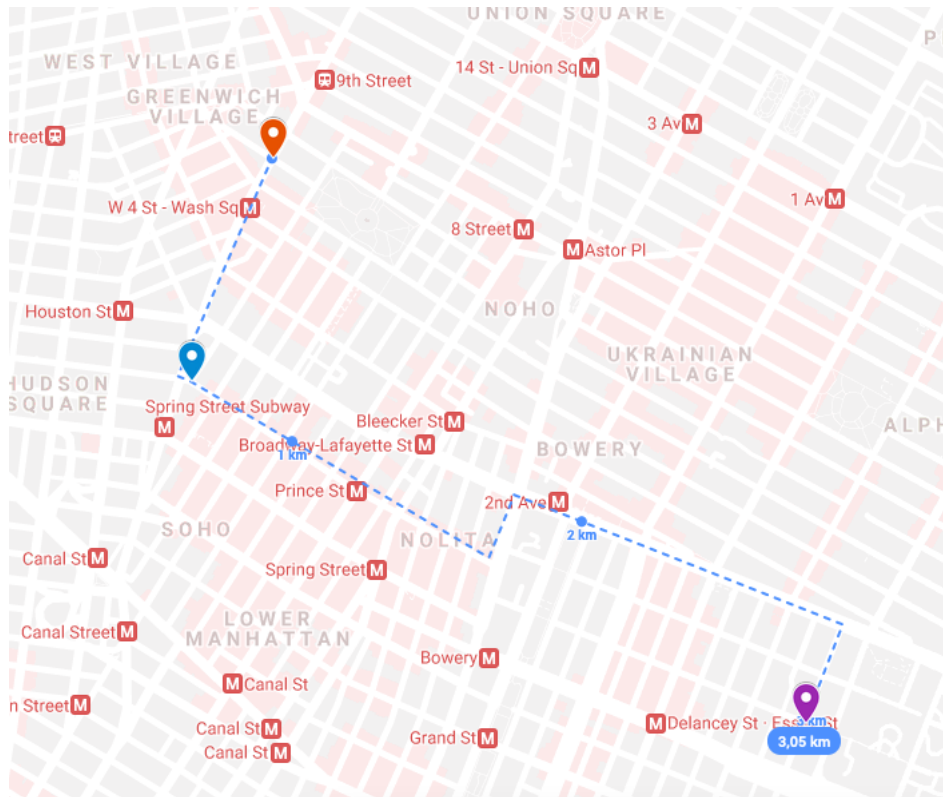


Figura 4: Verificación del primer viaje generado para las lugares de usuario

- 📍 Washington Place & 6th Ave...
- 📍 MacDougal Street & Prince S...
- 📍 Rivington Street & Ridge Stre...

Query:

```

1 result = trips.aggregate([
2     {
3         "$match": {
4             "$and": [
5                 {"start_station_id": bike_station.id},
6                 {"end_station_id": {"$ne": bike_station.id}}
7             ]
8         },
9     },
10    {"$project": {
11        "_id": 0,
12        "hour": {"$hour": {"date": {"$dateFromString": {"dateString": '
13        $start_time'}}}},
14        "year": {"$year": {"date": {"$dateFromString": {"dateString": '

```



```

14     "month": {"$month": {"date": {"$dateFromString": {"dateString": '
15     $start_time'}}}},
16     "trip_duration": 1,
17     "start_station_id":1,
18     "end_station_id":1
19   }},
20   {"$sort": {"hour": 1, "year": -1, "month": 1, "trip_duration": -1}},
21   {
22     "$match": {
23       "$and": [
24         {"hour": {"$gte": user_hour-1, "$lte": user_hour+1}},
25         {"trip_duration": {"$lte": trip_time + 120}}
26       ]
27     }
28  })

```

### 3. Conclusiones

Sin lugar a dudas este proyecto fue muy retador en todos los sentidos, desde la limpieza de los datos hasta la creación de las rutas. Haciendo un análisis de lo que fue esta práctica, sin duda lo que más me quedo es la eficiencia con la que trabaja MongoDB y la versatilidad que tiene para poder trabajar con proyectos de muchas facetas. Como lo mencionaba en la introducción, gran parte de lo que hace a MongoDB una de las mejores bases de datos NoSQL es la adición de diversos frameworks. El trabajar con el framework de agregación se me hizo muy natural pues es algo con lo que estoy acostumbrado a trabajar como científico de datos.

### Referencias

- Pimentel A. *Notas sobre MongoDB*. Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas. UNAM. 2021
- Bradshaw S. et.al. *MongoDB The Definitive Guide*. Third Edition. OReilly. 2020.
- <https://docs.mongodb.com/>