

P4: Working with the Database

Boston Blue Bikes Analysis

Project Group 15

Anjali Kabra

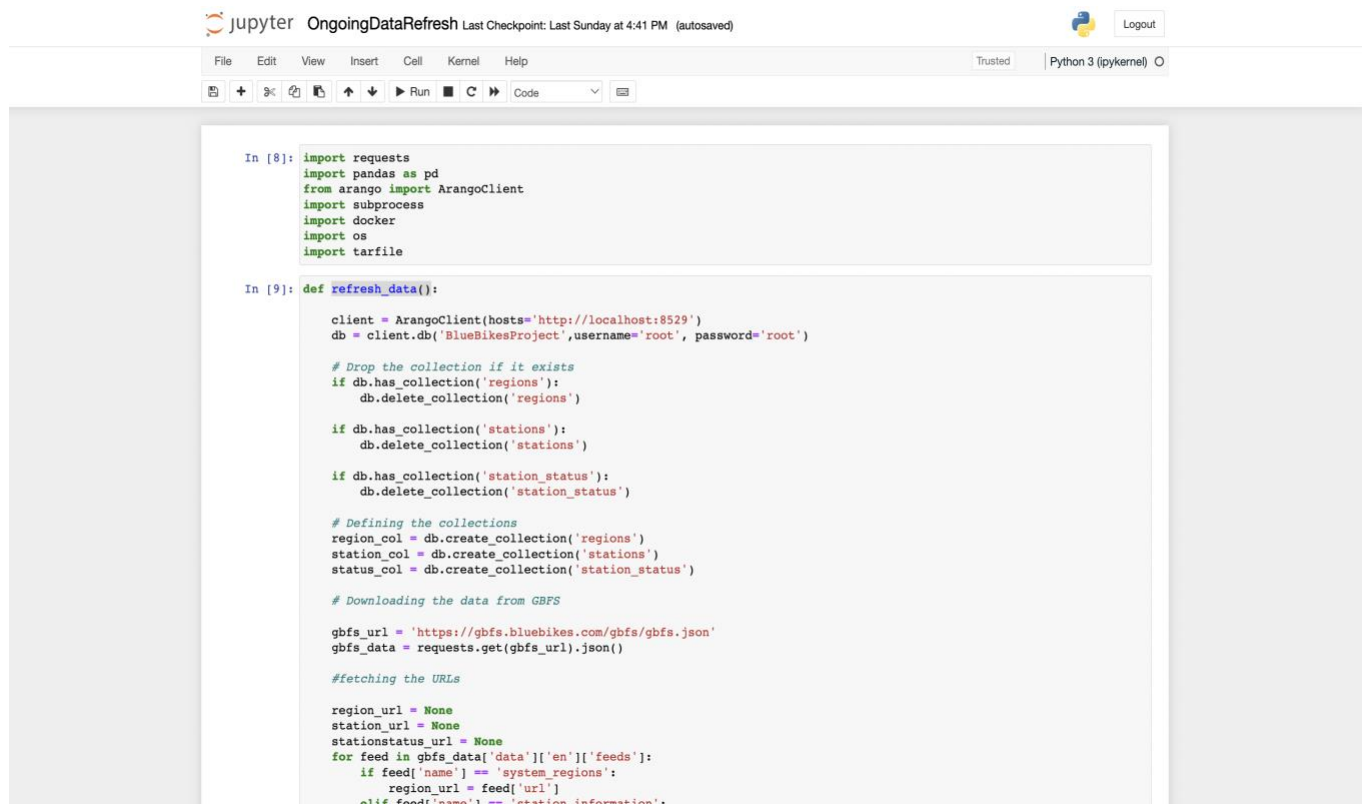
Jhalak Surve

Krishna Kapadia

Shubham Shah

Brief description of the data maintenance process:

- Since we have used GBFS live data for our project (as mentioned in P3) and this data keeps on getting updated from time to time, we have created a python script to fetch the latest data from the API and load it into our database.
- For this purpose, we have created a function in python **refresh_data()** which fetches the latest data from the GBFS website, cleans it and loads into our arango db collections.



The screenshot shows a Jupyter Notebook window titled "OngoingDataRefresh" with a last checkpoint from Sunday at 4:41 PM. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help) and a toolbar with icons for file operations, running cells, and code execution. The code is written in Python 3 (ipykernel). The notebook contains two input cells. The first cell (In [8]) imports necessary libraries: requests, pandas as pd, ArangoClient from arango, subprocess, docker, os, and tarfile. The second cell (In [9]) defines a function named refresh_data(). This function initializes an ArangoClient, connects to a database named 'BlueBikesProject', and checks for the existence of collections 'regions', 'stations', and 'station_status'. If they exist, they are deleted. Then, new collections are created. The function then fetches data from the GBFS API, processes it, and updates the database collections based on the feed type (system_regions or station_information).

```
In [8]: import requests
import pandas as pd
from arango import ArangoClient
import subprocess
import docker
import os
import tarfile

In [9]: def refresh_data():

    client = ArangoClient(hosts="http://localhost:8529")
    db = client.db('BlueBikesProject', username='root', password='root')

    # Drop the collection if it exists
    if db.has_collection('regions'):
        db.delete_collection('regions')

    if db.has_collection('stations'):
        db.delete_collection('stations')

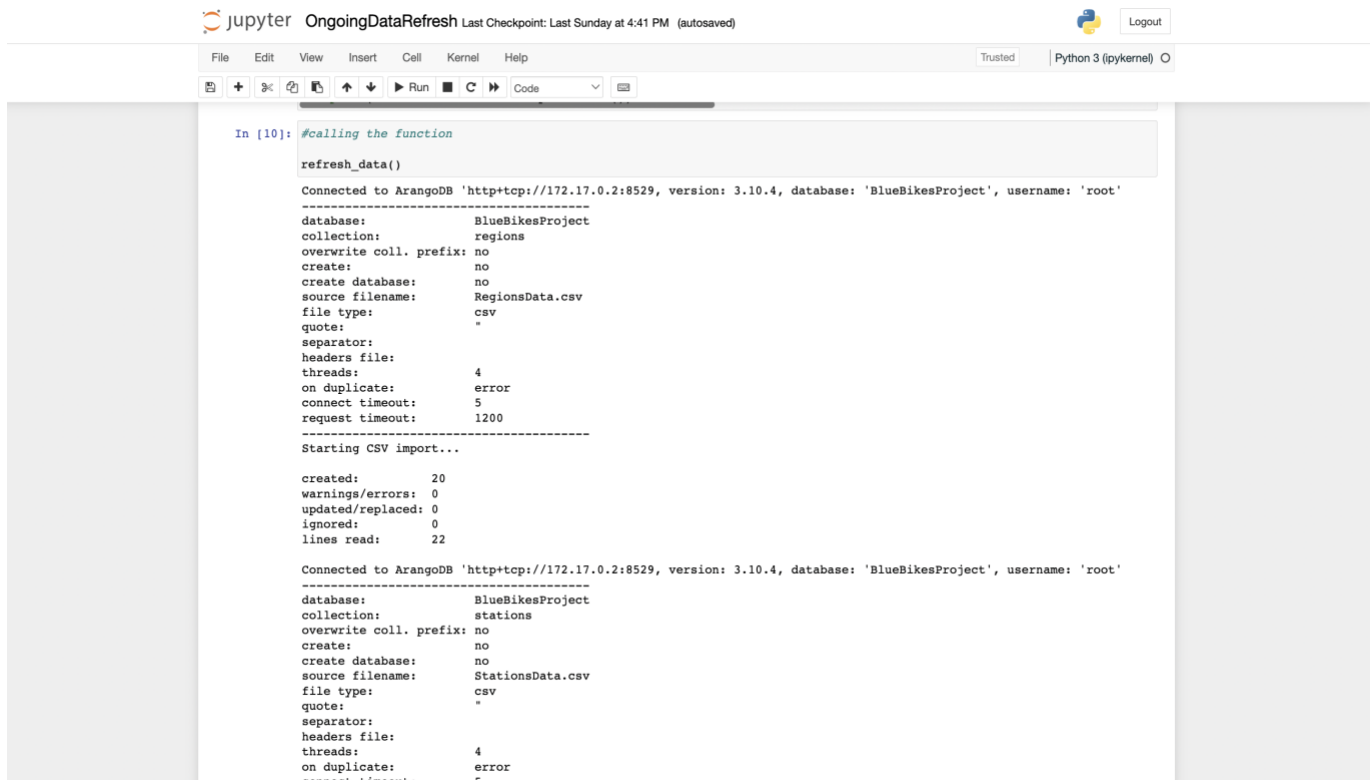
    if db.has_collection('station_status'):
        db.delete_collection('station_status')

    # Defining the collections
    region_col = db.create_collection('regions')
    station_col = db.create_collection('stations')
    status_col = db.create_collection('station_status')

    # Downloading the data from GBFS
    gbfs_url = 'https://gbfs.bluebikes.com/gbfs/gbfs.json'
    gbfs_data = requests.get(gbfs_url).json()

    # Fetching the URLs
    region_url = None
    station_url = None
    stationstatus_url = None
    for feed in gbfs_data['data']['en']['feeds']:
        if feed['name'] == 'system_regions':
            region_url = feed['url']
        elif feed['name'] == 'station_information':
```

- Then, we're calling this function to execute the data refreshment process.



```
In [10]: #calling the function
refresh_data()

Connected to ArangoDB 'http+tcp://172.17.0.2:8529, version: 3.10.4, database: 'BlueBikesProject', username: 'root'
-----
database:      BlueBikesProject
collection:    regions
overwrite coll. prefix: no
create:        no
create database: no
source filename: RegionsData.csv
file type:     csv
quote:         "
separator:
headers file:
threads:       4
on duplicate:  error
connect timeout: 5
request timeout: 1200
-----
Starting CSV import...

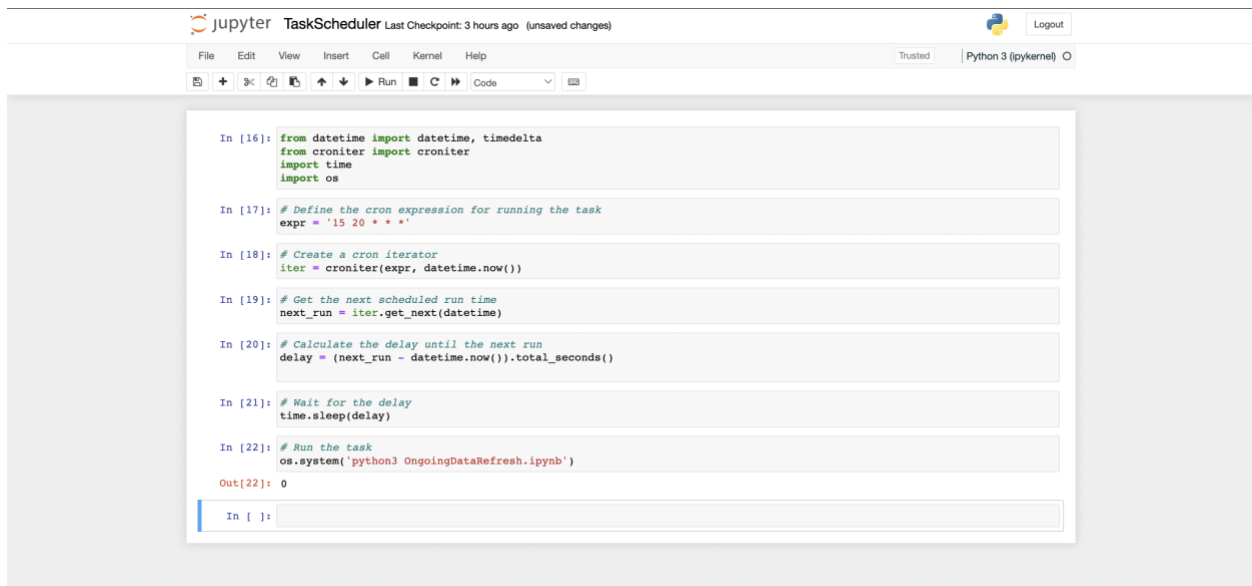
created:      20
warnings/errors: 0
updated/replaced: 0
ignored:      0
lines read:   22

Connected to ArangoDB 'http+tcp://172.17.0.2:8529, version: 3.10.4, database: 'BlueBikesProject', username: 'root'
-----
database:      BlueBikesProject
collection:    stations
overwrite coll. prefix: no
create:        no
create database: no
source filename: StationsData.csv
file type:     csv
quote:         "
separator:
headers file:
threads:       4
on duplicate:  error
connect timeout: 5
```

- We have also attached the jupyter notebook file **OngoingDataRefresh.ipynb** with our submission.

Scheduling the task to run the python script:

- Now, to schedule this python script to run daily, we have used **croniter**, which is a python library for working with cron expressions. Cron expressions are used in MacOS for task scheduling.
- We have created another python script TaskScheduler.ipynb, to schedule this script to run daily at 8:15 PM. (Also attached TaskScheduler.ipynb with the submission)



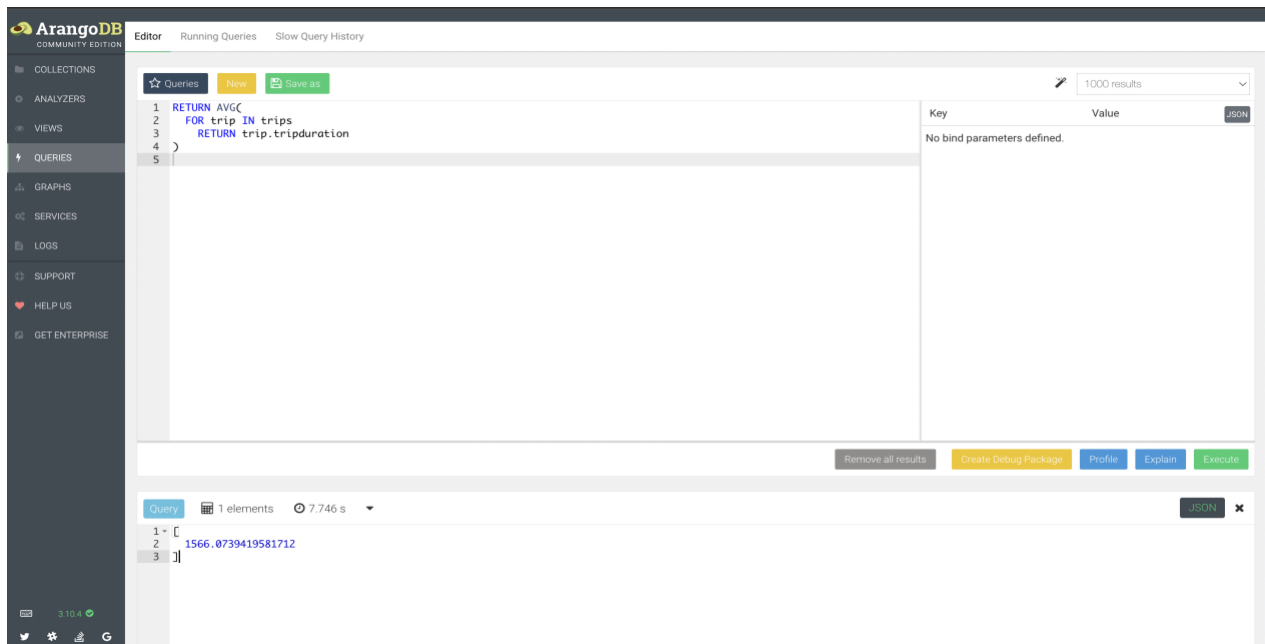
The image shows a Jupyter Notebook interface with the title 'TaskScheduler'. The notebook contains a series of code cells. The first cell imports necessary modules: `from datetime import datetime, timedelta`, `from croniter import croniter`, `import time`, and `import os`. Subsequent cells define a cron expression `expr = '15 20 * * *`, create a cron iterator `iter = croniter(expr, datetime.now())`, get the next scheduled run time `next_run = iter.get_next(datetime)`, calculate the delay until the next run `delay = (next_run - datetime.now()).total_seconds()`, wait for the delay `time.sleep(delay)`, and finally run the task `os.system('python3 OngoingDataRefresh.ipynb')`. The output of the last cell is `Out[22]: 0`.

AQL Queries to work with the database

We have also written some AQL queries to work with our database (attached a .sql file with the submission for this):

1. AQL query to find out the average duration for which a rider rents a bike

```
RETURN AVG(  
  FOR trip IN trips  
  RETURN trip.tripduration  
)
```



The image shows the ArangoDB interface. On the left is a sidebar with navigation options: COLLECTIONS, ANALYZERS, VIEWS, QUERIES (selected), GRAPHS, SERVICES, LOGS, SUPPORT, HELP US, and GET ENTERPRISE. The main area is titled 'Editor' and contains an AQL query editor with the following code:

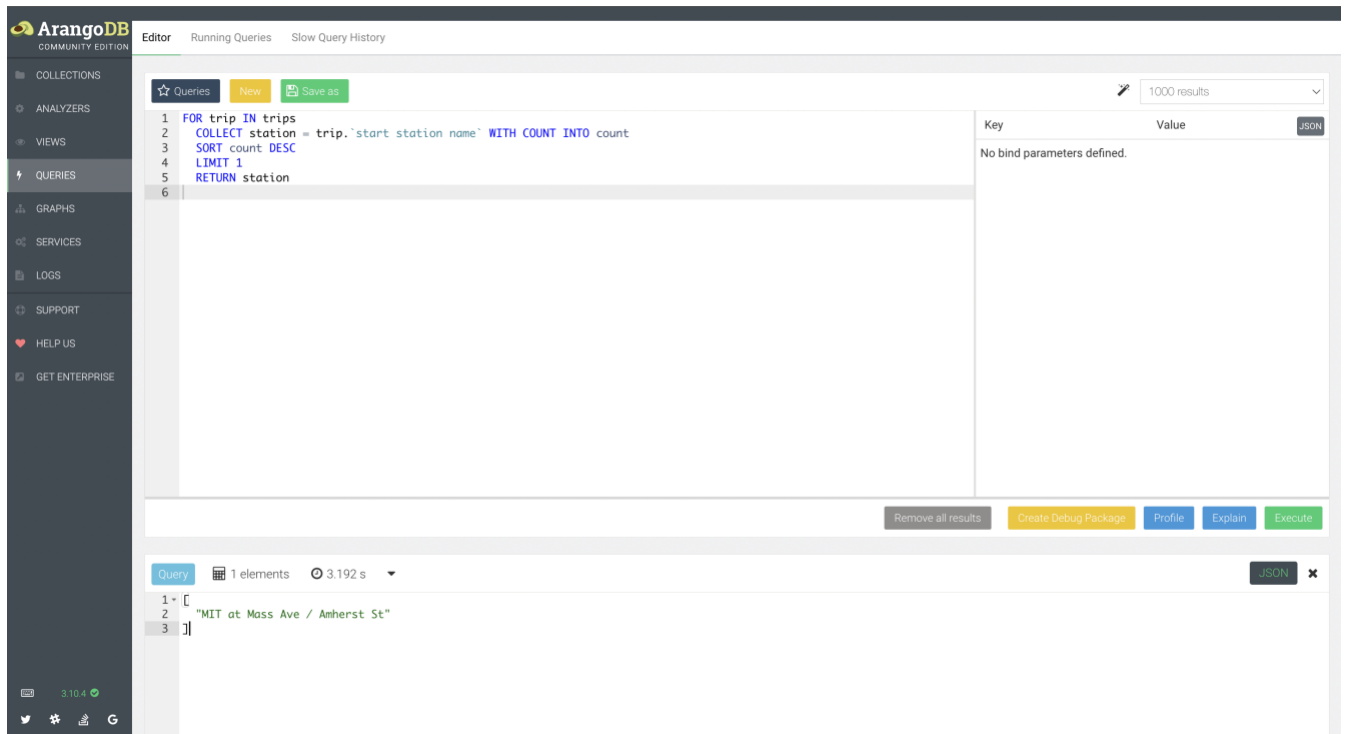
```
1 RETURN AVG(  
2   FOR trip IN trips  
3   RETURN trip.tripduration  
4 )  
5
```

Below the editor, there are buttons: 'Remove all results', 'Create Debug Package', 'Profile', 'Explain', and 'Execute'. The 'Execute' button is highlighted. To the right of the editor, there is a 'Key' and 'Value' section with a 'JSON' button. Below this, it says 'No bind parameters defined.' At the bottom, there is a 'Query' section showing '1 elements' and a timer '7.746 s'. The results are displayed in a table with columns '1' and '2'.

1	2
1	1566.0739419581712

2. AQL query for to find out the most frequently accessed station

```
FOR trip IN trips
  COLLECT station = trip.`start station name` WITH COUNT INTO count
  SORT count DESC
  LIMIT 1
  RETURN station
```



3. AQL query for to find out the rush hours

```
FOR trip IN trips
  LET startHour = DATE_HOUR(DATE_TIMESTAMP(trip.starttime))
  FILTER startHour >= 7 AND startHour <= 9 OR startHour >= 16 AND startHour <= 18
  COLLECT hour = startHour WITH COUNT INTO count
  SORT count DESC
  RETURN {hour: hour, count: count}
```

The screenshot shows the ArangoDB Community Edition interface. On the left is a sidebar with navigation options: COLLECTIONS, ANALYZERS, VIEWS, QUERIES (selected), GRAPHS, SERVICES, LOGS, SUPPORT, HELP US, and GET ENTERPRISE. The main editor area contains an AQL query:

```

1 FOR trip IN trips
2   LET startHour = DATE_HOUR(TIMESTAMP(trip.starttime))
3   FILTER startHour >= 7 AND startHour <= 9 OR startHour >= 16 AND startHour <= 18
4   COLLECT hour = startHour WITH COUNT INTO count
5   SORT count DESC
6   RETURN {hour: hour, count: count}
7

```

Below the query editor, the results are displayed in a table format. The table has two columns: 'hour' and 'count'. The results are sorted by count in descending order.

hour	count
17	294696
16	283100
18	267293
8	112226
9	95958

At the bottom of the interface, there are buttons for 'Remove all results', 'Create Debug Package', 'Profile', 'Explain', and 'Execute'. The status bar at the bottom indicates 'Query', '6 elements', and '3.083 s'.

4. AQL query for Bike usage by user type

```

FOR trip IN trips
  COLLECT userType = trip.usertype WITH COUNT INTO count
  RETURN { userType: userType, tripCount: count }

```

The screenshot shows the ArangoDB Community Edition interface. On the left is a sidebar with navigation options: COLLECTIONS, ANALYZERS, VIEWS, QUERIES (selected), GRAPHS, SERVICES, LOGS, SUPPORT, HELP US, and GET ENTERPRISE. The main editor area contains an AQL query:

```

1 FOR trip IN trips
2   COLLECT userType = trip.usertype WITH COUNT INTO count
3   RETURN { userType: userType, tripCount: count }
4

```

Below the query editor, the results are displayed in a table format. The table has two columns: 'userType' and 'tripCount'. The results are sorted by tripCount in descending order.

userType	tripCount
Customer	964150
Subscriber	3086446

At the bottom of the interface, there are buttons for 'Remove all results', 'Create Debug Package', 'Profile', 'Explain', and 'Execute'. The status bar at the bottom indicates 'Query', '2 elements', and '2.578 s'. There are also buttons for 'Download', 'CSV', and 'Copy To Editor'.