# CS 340 Project One README

## About the Project

**Grazioso Salvare Project**

In this project our company Global Rain is working on behalf of our client Grazioso Salvare. The client Grazioso Salvare identifies dogs that would be good candidates for search and rescue operations and has contacted us to set up a database for five different non profit agencies that house dogs and to create an API and a web facing application that their clients can use to browse these databases for dogs that would be suitable for these operations. This will be a full stack development project and we will be tasked with categorizing the data in the database and also building the API and application through which the clients access the database

## Motivation

The motivation behind the creation of this project is to help the client with their mission to find search and rescue dogs for their customer base. To do this we will have to first gather the data and import it into a database and then build an API that can be used to search the database and finally to create a web facing application that the clients can use to access the database information and to find suitable dogs for their training purposes.

## Getting Started

To use the data in this project you will first have to set up an account inside the MongoDB with the correct privileges. Once that is done and the data you need is imported using the correct commands you can import the data using the mongo import tool to create the database for the shelter you wish to peruse. After this you can use the API to make calls to the database to query certain result and to create, read, update, and delete the items in the database according to your needs. This will require some tools which I will discuss below.

## Installation

The tools needs to use the CRUD python module and the database are as follows (I have included links to access the downloads for each):

* Linux (https://www.linux.org/pages/download/)
* MongoDB(<https://www.mongodb.com/try/download/community>)
* Python Environment for Linux(<https://www.mongodb.com/try/download/community>)
* Jupyter Notebooks for Linux(<https://jupyter.org/install>)
* The Python Module created for CRUD(will be included in the github)
* Dash library for Python

I will be discussing the tools in more detail in the section marked Functionality and Rationale below

## Usage

The python module will be used for basic CRUD operations and will provide users with the ability to alter and read the information in the database by importing the module into their systems or by using the application itself.

### Code Example

Setting up and running the Python Module caused some challenges in the beginning for me. I will document them here and provide some remediation advice.

The first obstacle I ran into was authorization. In order for the code to work properly you need to have both and read and write privileges in the database. So before running the code I first had to go to the admin database in Mongo and give the user listed in the python module read and write privileges. After this was done I was able to proceed into the actual code.

The first part of the Python code is just setting up the user credentials and connecting to the database. You need to use this import at the top of the code to make sure it is able to connect to the database.

**from pymongo import MongoClient**

After doing so you will need to enter in the user credentials. As noted earlier you need to enter in a user that has both read and write privileges to make this part of the code work. After doing so you can create both the read and write methods. The write method is fairly straightforward but I would include a conditional that prints when the code has successfully ran:

**if data is not None:  
 self.database.animals.insert\_one(data)  
 pprint(data)  
 return True # data should be dictionary  
 else:  
 raise Exception("Nothing to save, because data parameter is empty")  
 return False**

Using this it will return True if the code runs and false if the data parameter is empty. The read portion of the code is otherwise fairly straightforward. Next you will need to do the read method. To do this I used find instead of findOne() and this caused some issues for me as I did not include the cursor at first. To remediate this issue I needed to use a cursor:

**cursor = self.database.animals.find(query)  
 result = list(cursor)  
 return result**

Once I included the cursor I was able to successfully use the find method to query my database. The rest of the read function is similar to the write method but using this particular code fragment improves the functionality of the code. Other than this the code is relatively straightforward and you are able to both create and find items in the database using it.

For the update and delete method the code is very similar the only notable difference is it returns **modified\_count**  and **deleted\_count** so you know how many items you’ve updated or deleted.

**Tests**

In order to access the python module used for interacting with the database you will have to import the python module by creating a jupyter notebook and typing these import commands at the top of the page:

**import pymongo  
from pymongo import MongoClient  
from crud import AnimalShelter**

After doing this you can use this command to create a new data entry. I have included an example entry. You can fill in the data with other information and add other attributes as well. After entering in the data the if conditional will check if it has been entered successfully:

**CREATE**

**data = {  
'age\_upon\_outcome': '1 year',**

**'breed' : "Corgi",  
'animal\_type': "Dog"  
}  
  
animal = AnimalShelter()  
  
insert = animal.create(data)  
  
if insert:  
print("done")  
else:  
print("not done")**

**READ**

You can also interact with the module and query results using this example code:

**query = {  
 'breed':'Corgi'  
}  
  
query = animal.read(query)  
  
if query:  
 for info in query:  
 print(info)  
else:  
 print("no info")**

UPDATE

You can also interact with the module to update data based on certain parameters:

**##update method data    
update\_breed = {  
  'breed': 'Corgi'  
}  
  
update\_data = {  
 'age\_upon\_outcome': '3 years'  
}  
##method use for update  
update = animal.update(update\_breed, update\_data)  
  
if update:  
 print("successfully updated")  
else:  
 print("no update")**

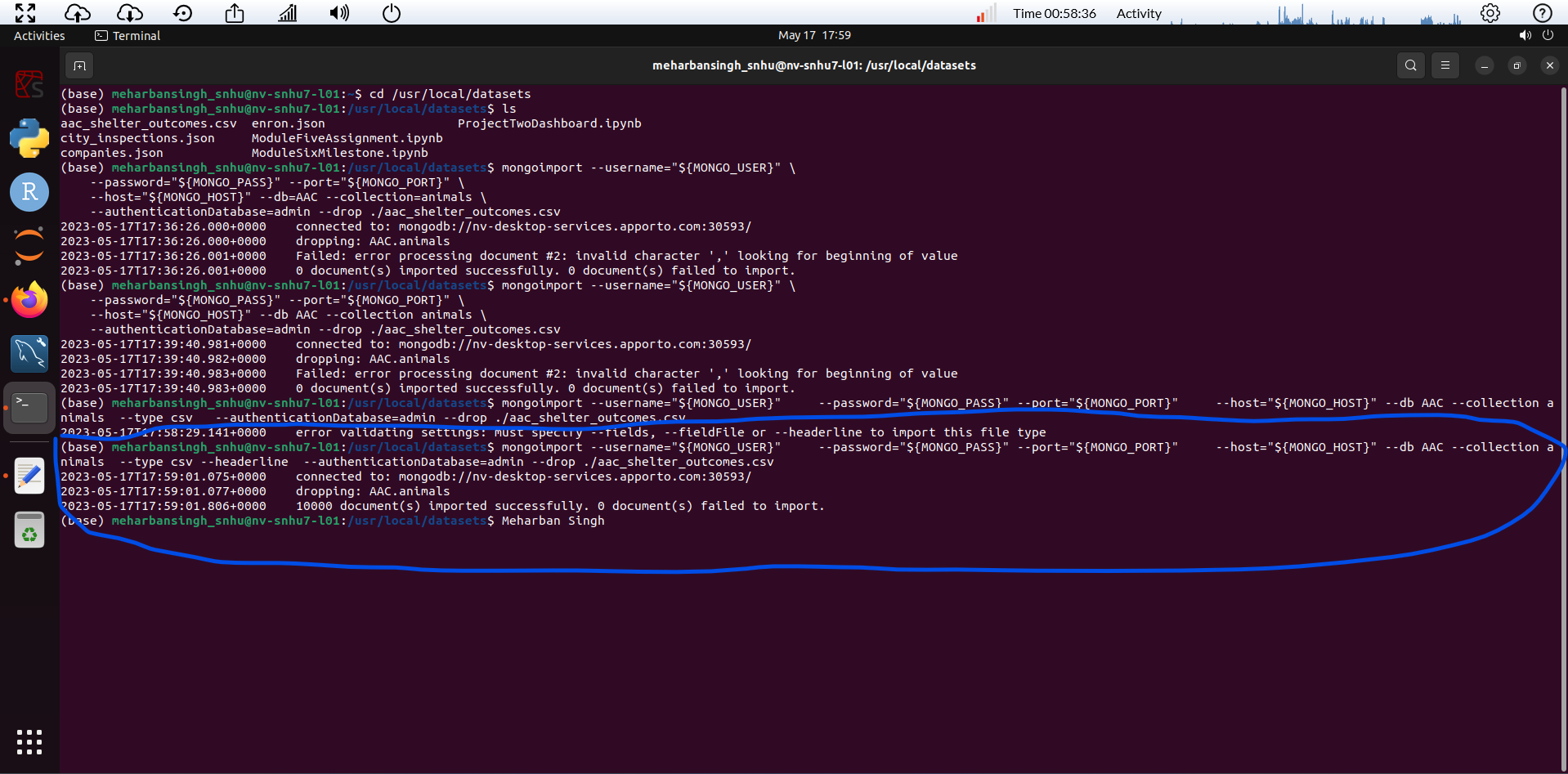
DELETE

##delete datadelete\_data = {

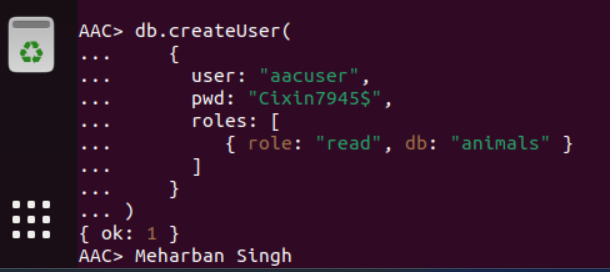
'breed':'Corgi'}##delete methoddelete\_test = animal.delete(delete\_data)if delete\_test:print("successfully deleted")else:print("not deleted")

### Screenshots

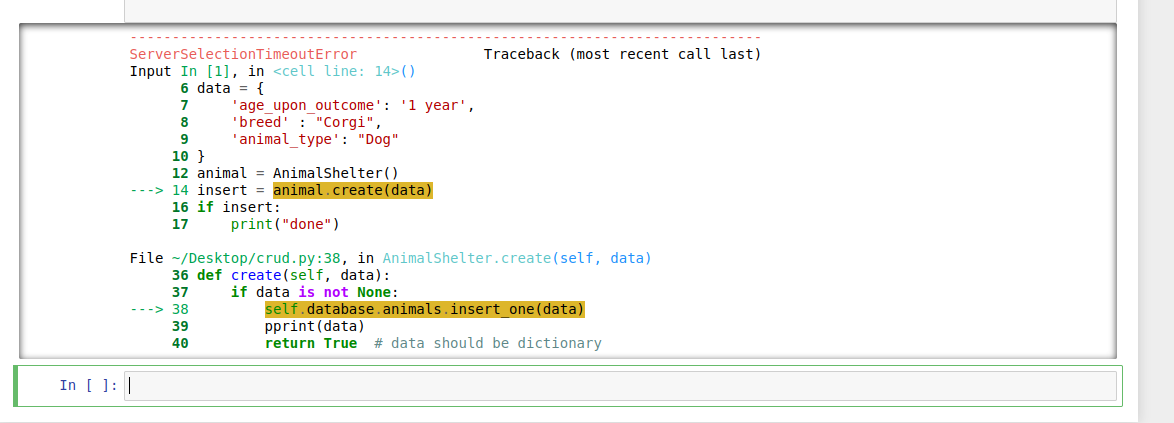
Here is a screenshot that shows how to add a shelter to the database. You will need admin or read and write privileges to do so:



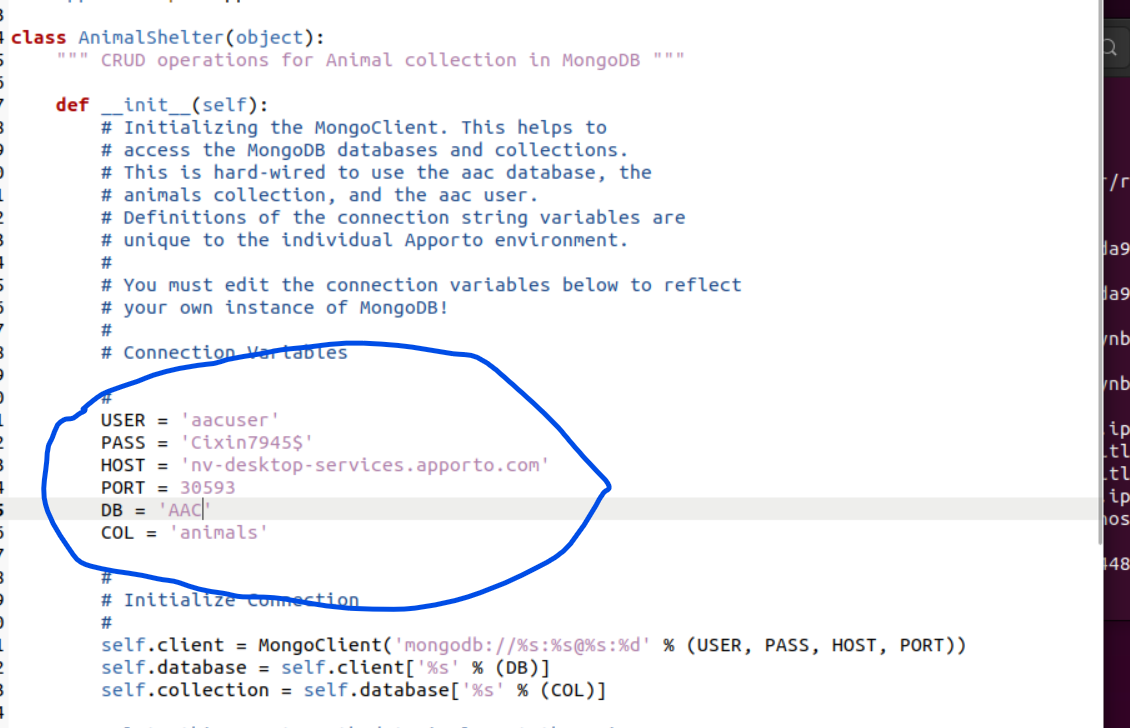
Here is a screenshot that shows how to create a user:



Here is a screenshot of the error you get if you attempt to use a user that does not have read and write privileges:



As you can see the server will time out if you do not have the correct privileges. You will also get a server timeout if you enter in any of the credentials incorrectly in the python file so make sure that the highlighted section including your port number, username, password, hostname, and database all match what you have in your own MongoDb. Otherwise you will not be able to connect as these are all specific to your system and database:

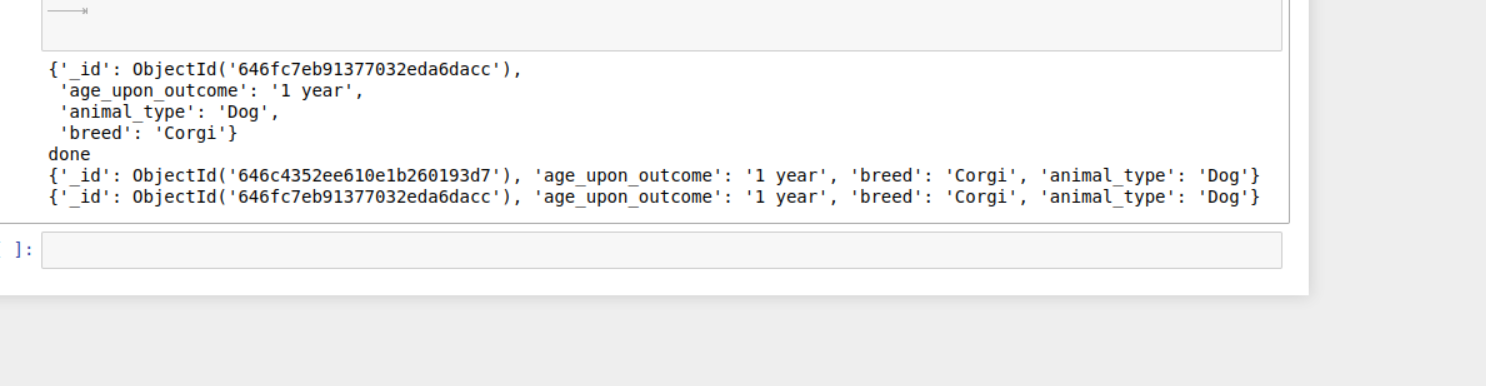


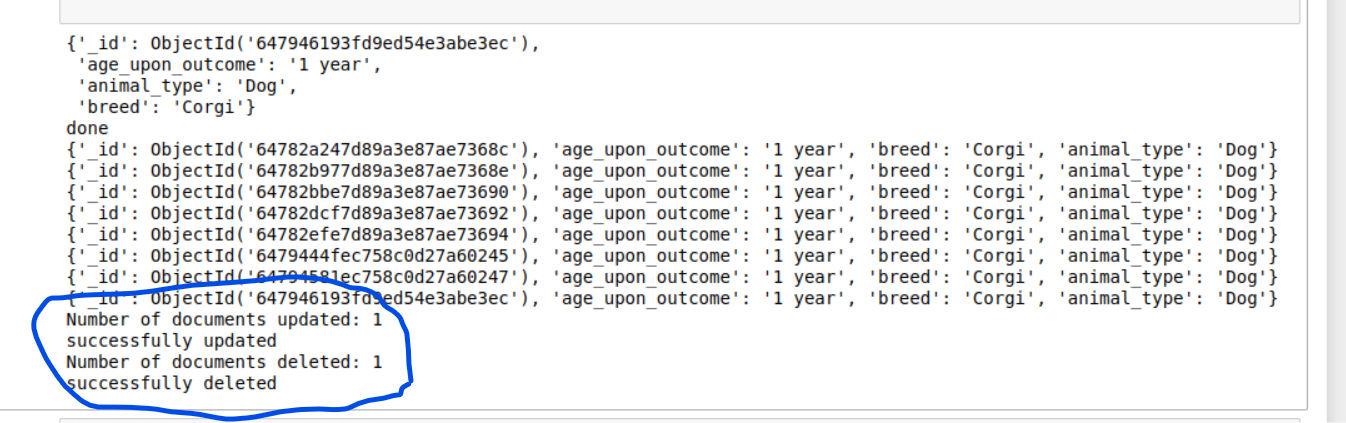
Here is a photo of the commands I ran in my jupyter notebook. You can use this as a reference to interact with the database and CRUD module:

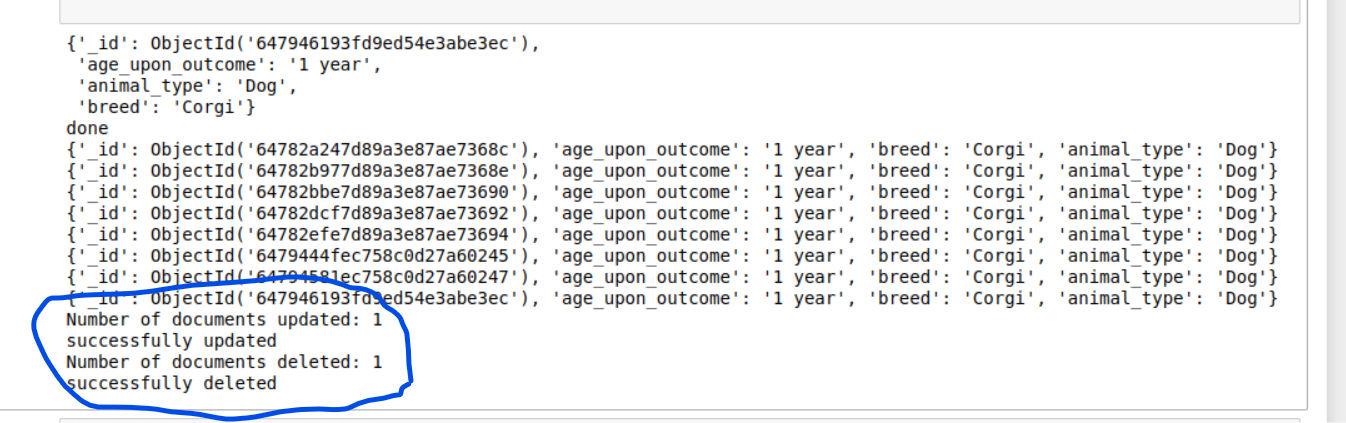




If the code does run successfully you should see this at the bottom of your Jupyter Notebook. This shows that both the read and write were done successfully as it printed the done showing the successful insert operation and the Object ID which is returned after querying:

After executing the update and delete commands you will see this output:





**Functionality:**

The app is linked to the database via the API described above. In order to make it easier for our clients to access the information stored in the database we have created a front end application using the Dash library which calls the CRUD API to query certain results inside the database and formats it into a readable format for the end user. Alongside this there is also a pie graph that visually represents the data based on the query and also a geolocation chart that shows exactly where each animal is located based on what you click. Here is a link to a screencast that shows how the app works in more detail:

<https://drive.google.com/file/d/1HBuLnN2tYkfckeAD-gZavKPYclaUUnjL/view?usp=sharing>

**Tools/Rationality for choice of tools:**

The two tools used for the dashboard section were MongoDB and Dash. MongoDB was used primarily because it offers a large array of querying capabilities and is well suited to being used as the database for full stack development. It is easy to use with Python since it is offers native integration within Python which means that I could easily query and read data from inside the code used to create the dashboard. It also gives me the power to use the MongoDB easily since it is easy to manipulate queries and offers a good deal of flexibility in terms of what I needed to see inside the dashboard. You can use the PyMongo driver to integrate Python with Mongo quite seamlessly and this made it good choice to use for my application.

Dash was used as well since it is again easily integrated into Python via the library and offers a wide array of tools for both viewing the data in a UI and for controlling the data. You can use Dash and HTML to create a UI front end that can be viewed by the client and then link it to controllers and buttons through which the user can manipulate the data by linking the code to the MongoDB. Furthermore geolocation capabilities are built into the dash framework and can be easily accessed and implemented into the UI. Dash also provides a development server which makes it easy to deploy the application and view the data.

I provided the links above for the other tools used (including Mongo) in the section titled “Installation” and over here I will provide the links for Dash documentation:

https://dash.plotly.com/

**Steps taken to complete the Dashboard portion of this app:**

I explained the steps taken to create the API and access the data above and in this section I will explain how I set up the dashboard portion. The dashboard was a little more complicated since it involved setting up the callbacks, querying the data properly, and making sure the buttons displayed and worked properly. First I had to create the HTML section where I put the data for the buttons and created a link for them in each of the requisite callbacks for the proper functions. Then after that I created the callbacks for the buttons that linked the buttons to certain queries depending on what button was pressed. For example if you pressed the Water button it would go into the update\_dashboard function in the code and pass that parameter into the code via a conditional statement and then query it in mongodb then display the data on the screen. After that I created the geolocation and pie graphs which were a little more straightforward. I simply had to link the information being queried into the pie graph and geolocation and dash took care of most of the heavy lifting in that regard. After all this was setup I also created a reset button linked to an empty query that would return all the results.

**Challenges faced for Dash Portion of Project:**

I faced many challenges during this portion of the assignment. First I couldn’t get the image to display. Then I realized that I had to first encode the image and then pass that onto the HTML document in order for it work. Second I could not get the buttons to display properly and couldn’t decided if I wanted a radio or a dropdown. I was having trouble properly implementing the syntax for the buttons and they would not show up in the app. I fixed this by reading the documentation more carefully and following the syntax exactly. After doing all this I had trouble with the update\_dashboard function. Whenever I would click anything I would keep getting multiple errors saying datable\_id.data cannot be updated. The dash would work but none of the data was querying properly. I realized that this was because one of my query’s was searching for a non existent attribute and this was causing it to not display. Once I fixed this it began to display properly. Related to this I also realized that I needed to pass the query data to a DataFrame and then return a dictionary in order for the data to display properly. Otherwise I would get another callback error saying that the data could not be retrieved.

Another issue I was having was that the pie graph was not displaying and neither was the geolocation. I realized that this was related to the dashboard so once I fixed that data it began to display properly. I also had issues with the callbacks because I kept making syntax errors when passing the html button data to the callbacks but once those were fixed the app ran smoothly.

**Contact Name:** Meharban Singh