Final Project  
Project Two

Zane Russell Brown

Southern New Hampshire University

CS-340 Client/Server Development

# CS 340 Project Two: README

## About the Project/Project Title

This project is designed to develop a web application that connects a client-side user interface to a database and outputs the data through a series of visuals using the Dash Plotly dashboard. This project will be able to insert documents, read said documents as well as have delete functionality and update functionality into the MongoDB database and collection. The project will then use the read functionality to query for data and display it onto the dashboard. This document will be broken into two parts. First, the CRUD section of the document will explain how to create a user in the mongo shell and create the CRUD program that will implement the insert, read, delete and update functionality. The second section of the document will explain how to create the dashboard program that will take the data from the CRUD program and output it using visuals.

## Motivation

Web sites and applications are more than just HTML, CSS and Javascript. Users also need to be able to interact with the application in a variety of ways. Such as creating documents, updating them, reading them and deleting them. Additionally, users need to be able to easily understand the data provided to them through visuals. This project aims to create a web application that connects to a user interface and allows users to insert a number of documents with key/value pair and update, read and delete them in order to suit their needs and display the data using a series of visuals. These types of applications can be used with businesses keeping track of a large amount of data such as an inventory or used by individuals for personal uses.

**Section One: CRUD**

## Getting Started

Before we can get started on the Python code, we first need to create an admin user for the MongoDB shell. This will give us access to all commands and functions within the shell and will allow us to have less trouble in the future.

To create an admin user follow these steps.

1. Enter the following commands and supplement your own user name in place of “MyUserName”
   1. **db.createUser(  
       {  
       user: “MyUserName”,  
       pwd: passwordPrompt(),  
       roles: [{ role : “userAdminAnyDatabase”, db : “admin}, “readWriteAnyDatabase”]  
       }  
       )**

After entering this command, you should see a prompt asking for a password. Enter your password and press enter. You will then see an output letting you know that the user was created.

1. Next, enter the following commands to enter the Mongo shell with authentication.  
   1. **/usr/local/bin/mongo\_ctl stop**
   2. **/usr/local/bin/mongod\_ctl start**
   3. **Mongo –authenticationDatabase “admin” -u “MyUserName” -p**
   4. **Enter your password once prompt comes up.**

Congratulations! Now you have created an authorized admin user for your Mongo shell!

Next, we will be working on our project! We will be using jupyter to create the Python script for our program. For information on how to install Jupyter, see the *installation* section.

In order to get our web application to have the create, read, update and delete functions, follow the steps below.

1. **Import the necessary packages.**
   * 

We will be importing both the MongoClient and ObjectId from bson.objectid.

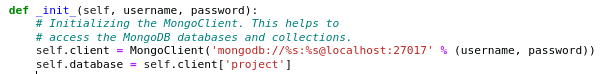
1. **Create a class.**

\*\***NOTE**: We will be using ‘**AnimalShelter**’ as our class name. Change this to a class name that fits the needs of your project.

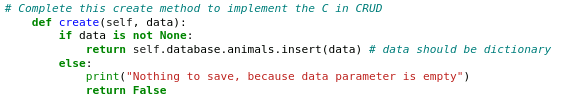
* + 

This will be the class that holds both our Create and Read functions for our CRUD program.

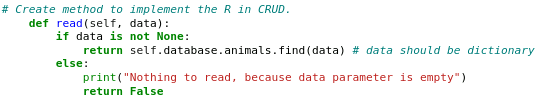
**\*\* NOTE: the ‘animal’ collection may be changed to any collection name your are using. \*\***

1. **Initialize the MongoClient.**
   * 

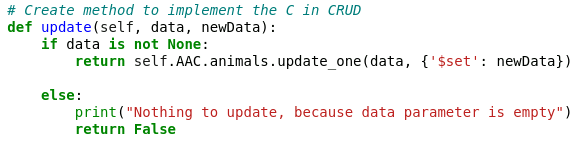
This will help initialize the Mongo Client and allow us to access the databases and collections within MongoDB. The username and password arguments will use the admin user previously created or for a user with specific privileges.

1. **Create the ‘create’ function.**
   * 

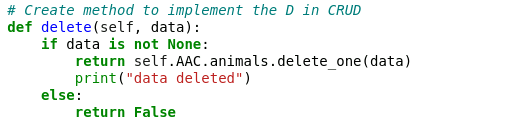
The code above provides the create method. This method takes the ‘data’ argument. This argument is used to take in the inputted data from the user and insert it into the database. Else, it will return a string that lets us know that the insert call was unsuccessful.

1. **Create the ‘read’ function.**
   * 

The code above provides the read method. This method, just like create, takes the ‘data’ argument. This argument is used to pull the inputted data, find it, and display it for the user. If there is no data to be found, a string will be outputted letting us know that the read call has failed.

1. **Create the ‘update’ function**
   * 

The code above provides the update method. This method takes the ‘data’ argument like the previous two methods. However, it also takes a new argument, newData. The first argument, data, takes the users input in order to query for the required database. newData then takes one more user’s input to update the database. If there is no data, an error message is produced.

1. **Create the ‘delete’ function**
   * 

The code above provides the delete method. This method only takes ‘data’ as an argument. Like the create and read method, delete takes the ‘data’ argument and uses that data to query for the necessary database and delete it. A message will then be printed letting us know that it was successful. Else, the program will return a False output.

## Installation

The tools that we will be using to create this program are:

* Jupyter
* Python (Anaconda distribution)
* MongoDB

**Jupyter**

Jupyter is an web tool that provides an online interface that allows users to create python code and will help us create our web application. In order to install Jupyter, follow these steps (provided by the official Jupyter website).

1. **Download Anaconda.** 
   1. Anaconda is a Python distribution that is used in conjunction with the Jupyter notebook. This distribution installs Python and the Jupyter notebook. The Jupyter site recommends the latest Python 3 version.
2. **Install Anaconda following the instructions on the webpage.**
3. **Run the following command to run Jupyter.**
   1. jupyter notebook

\*\*NOTE: a full list of installing instructions can be found [here](https://test-jupyter.readthedocs.io/en/latest/install.html) as well as other alternative download instructions.\*\*

**MongoDB**

MongoDB is a database that stores JSON documents and collections. This tool is used to high volume storage (GURU99, 2020). MongoDB can be used to create efficient and high functioning web applications. For more information on [MongoDB check their website](https://www.mongodb.com/what-is-mongodb).

MongoDB can be installed on Linux, macOS and Windows operating systems. In order to download this tool, please check [here for the MongoDB documentation on installations](https://docs.mongodb.com/manual/installation/) and choose your preferred operating system.

## Usage

### Code Example

Let’s briefly discuss what the code does step by step. We will be using the same example program as the one in the Getting Started section.

Import

* 

The import command allows us to import useful and necessary packages to help our program work. In this case, we imported the **MongoClient** package from **pymongo** directory as well as **ObjectId** package from the **bson.objectid** directory.

Class

* 

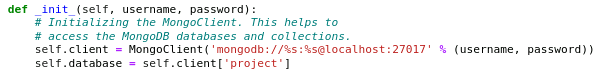
The class function allows users to create a class that will hold the created functions. A program can hold a number of classes that do various things in order to make the entire program work. In this case, we created the ‘AnimalShelter’ class to hold the **create** and **read** functions to hold the animals data.

Define

* 
* 
* 
* 
* 

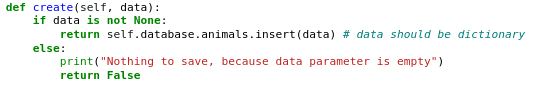
The following **define** (written as **def**) methods allow users to create the functions that are used in the class. The define methods allow users to add custom names (to help us understand what they will do) as well as arguments that the functions will take and use. The \_init\_ function takes a username and password argument, the create, read, update and delete methods take the data argument. The arguments help these methods perform a specific job.

\_init\_ Method

* 

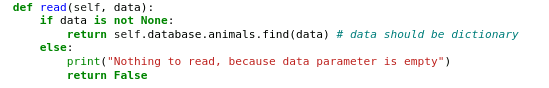
The \_init\_ function that was created for the program allows us to initialize the MongoClient and access the databases and collections from MongoDB. The function does this by taking the users username and password from the arguments and uses them to connect to the MongoClient using 

Create Method

* 

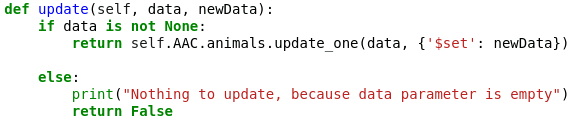
The create function that was created for the program allows for the ability to take the data inputted by the user as an argument and uses the **insert()** function to pass that data into the desired database within MongoDB. This function also uses the additional statements ***if*** and ***else****.* This provides two options for the program to execute. If the data is not empty, it will insert the data into a database. Else, if it is empty, it will return a message letting the user know that the call has failed.

Read Method

* 

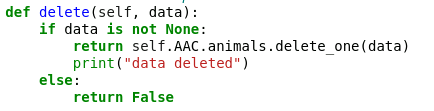
The read function that was created for this program works very similar to the create function. It takes the data that was inserted by the user as an argument and uses the **find()** function to display the data in the database. Again, this function uses the ***if***and ***else*** statements to give the program two options. If the data is found in the database, it will be displayed. If the database is empty, a message will be displayed letting the user know this.

Update Method



The update function is the method in this program that differs a little from the other methods. This method takes two arguments. Both arguments are user inputs. The first argument, ‘data’ is used to query for the necessary database. Then, the next argument, newData is used to update the queried database. The method uses the **update\_one()** function to perform this update. The results will be returned. Else, the program will output a message letting us know that the process failed due to the parameters being empty.

Delete Method

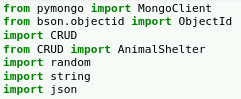


The delete method that was created for this program works a little similar to the create and read methods. It takes the data that was inserted by the user as an argument and uses it to query for the database and then uses the **delete\_one()** function to delete the database. After this is done, a message will be outputted letting the user know that the database has been deleted.

### Tests

Unit tests are very important when creating a program. These tests will let us know if the functions we create work as intended. Here, we will discuss an example unit test to help us check if our functions work correctly. Follow these steps to create a unit test.

**\*\*NOTE: we will be using a unit test for the ‘AnimalShelter’ class we created. These steps will let you know what you can change to fit your program.\*\***

1. **Import necessary packages.** 
   * 

This test requires more imports than our main project. So, we will be importing the **MongoClient** from **pymongo** and **ObjectId** from **bson.objectid** as we did in the main project.

In addition, we need to import the program we just created. Here our program is named ‘**CRUD**’ \*\*This may be different for you\*\*.   
We will also import our class that we created by using ***from CRUD import AnimalShelter***. \*\*Change ‘AnimalShelter’ to your class name\*\*.

Next, we will import **random**, **string** and **json** into our unit test.

1. **Call the class in a variable**
   * 

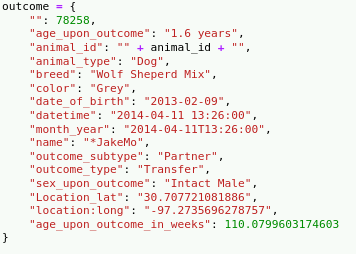
We will need to call our class later, so we will add the call into a variable. This variable can be named anything. It is recommended that the variable name allows users to understand what it is. Additionally, this call takes two arguments, the “username” and “password”. Here, they are added respectively.

1. **Create an ID generator**

* 

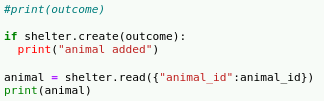
In order to give the users data a unique ID for the purpose of querying, we can use the random() function. The above example creates a random ID using both letters and numbers. The ID is then printed to the console. (printing is optional. It just lets us know that it is working). \*\*Variable name may be changed.\*\*

1. **Create an example dictionary**

* 

Because we are testing the create and read functions we created, example data is necessary. Here we created an example dictionary that will be inserted and read in the unit test. \*\*The information in the dict may be changed to suit your needs\*\*

1. **Output Results**

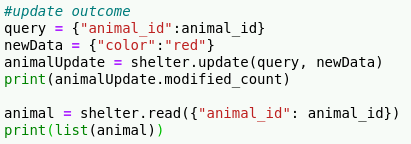
* 

Next, we need to have the unit test produce an output to ensure that our program works as intended. To do this, we can first use an ***if*** statement to print a confirmation message if the data was successfully inserted.

We can then use the **read()** function to find and display the dictionary that was created. If it is displayed, we can confirm that our program works properly.

\*\*Variables and messages may be changed to suit the needs of the user\*\*

1. **Update Outcome**

* 

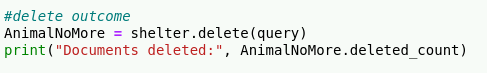
Next, we will test our update method. To do this we will first create three variables. The first variable ‘query’ will hold the query search for animal id. The second variable, ‘newData’ will hold the new data that will be inserted into the database. The last variable will hold our class call and update function using dot notation. The update function will also take our query and newData variables as arguments.

After doing this, we will make sure that the document was modified by using this code:

* 

This will produce and output that lets us know that the change was successful. Finally, we will use the **read()** function to query for the database and then we will use **print()** to output the results.

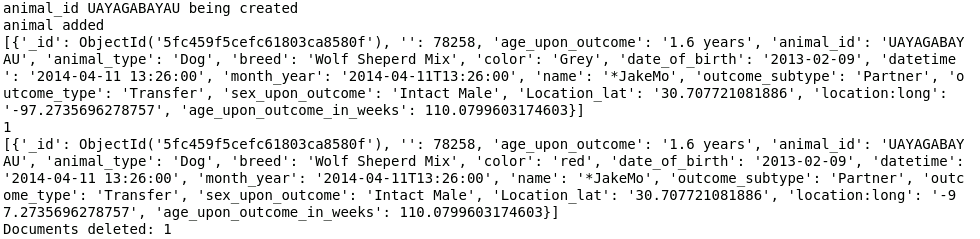
1. **Delete Outcome**

* 

Finally, we will test our delete method. To do this we will first create a variable that will hold our call and use dot notation with the **delete()** function to both query for the database and then delete it. The **delete()** function will also hold our previously created ‘query’ variable to query for the database using animal id. We will then use . This .**deleted\_count** statement gives an output that lets us know that our delete method was successful.

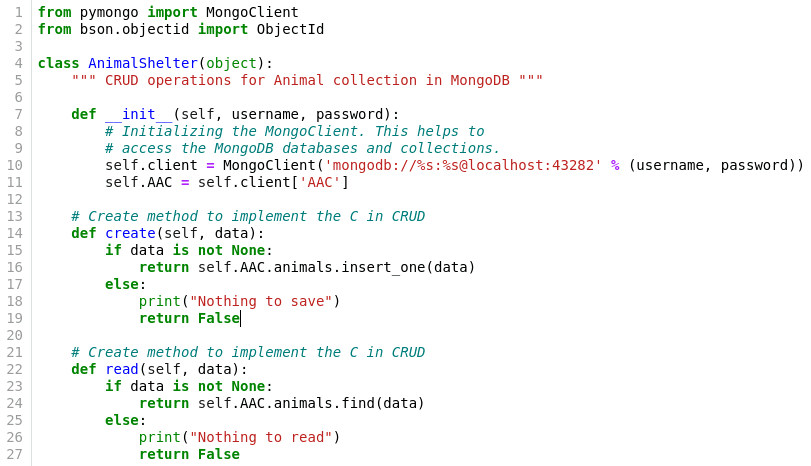
1. **Results**

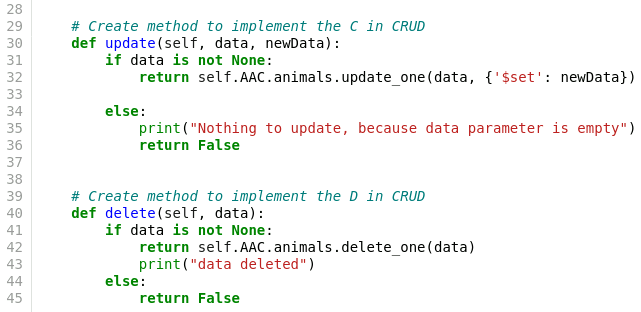
If you have followed this README document and there are no errors, you should get an output similar to the following:

* 

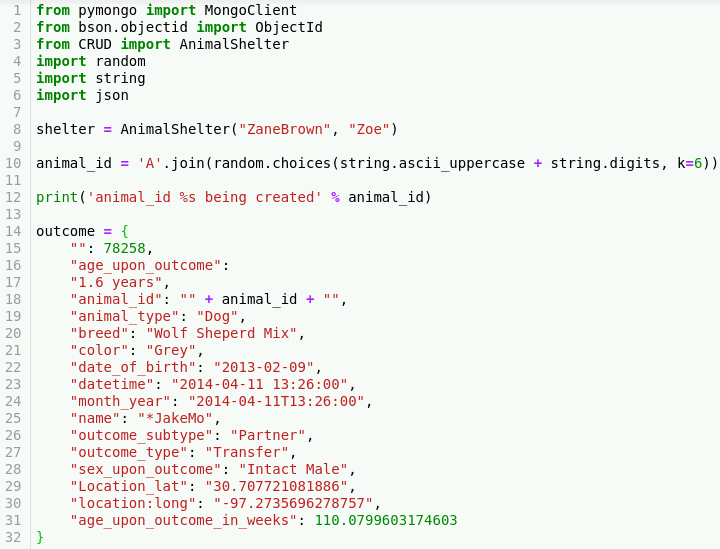
### Screenshots

Below are the full example of the create and read functions as well as the unit test used to test these features.



* 

This is the example program we used in the Getting Started portion of this document.



* 

This is the example unit test that was created to test out the create, read, update and delete methods of the main example program.

**Section One: Dashboard Program**

**Dashboard Functionality**

The dashboard program is designed to display a variety of interactive visuals that helps the user understand the Animal Shelter data better. This program takes the functions used in the CRUD module and uses that to take the Animal Shelter data and inserts it into the dashboard program. It then uses this data and displays it into a data table with interactive buttons and a pie chart. Additional visuals such as a bar chart and geolocation/choropleth chart were in the original design, but due to virtual lab issues, they were not in the final program.

**Steps for Creating Dashboard Program**

Lets discuss the steps that were made in order to complete this dashboard program.

## Getting Started

Before creating the dashboard program, it is important to first authorize yourself within the mongo shell. If you do not do this, it could cause issues when trying to run your program. Every time you log into the computer, you will need to follow these steps to perform authentication.

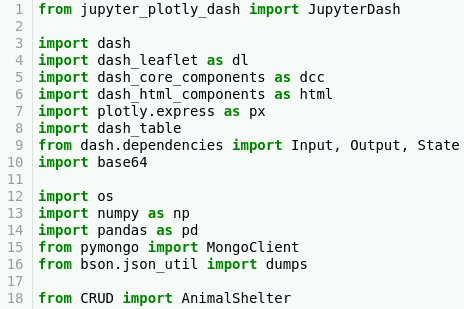
To authorize the user:

1. Enter the following commands to enter the Mongo shell with authentication.  
   * **/usr/local/bin/mongod\_ctl start**
   * **Mongo –authenticationDatabase “admin” -u “MyUserName” -p**
   * **Enter your password once prompt comes up.**

Congratulations! Now you have authorized yourself for your Mongo shell and now can start the dashboard program!

Next, we will be working on our dashboard project! We will again, be using jupyter to create the Python script for our program.

In order to create the dashboard program that outputs visuals from the data found in the Mongo database, follow the steps below.

1. **Import the necessary packages.**
   * 

You will need to import all of these packages in order to create the visuals for the data. These packages support visuals such as the das table, html components, dash core components, etc.

Don’t forget to import your CRUD project that we created earlier. To do this follow the import statement on line 18. Change “CRUD” and “AnimalShelter” to fit your project’s requirements.

1. **Hardcode in Username, Password & Datafile.**

\*\***NOTE**: We will be using ‘**ZaneBrown’ and ‘Zoe’** as our username and password respectively. Change this to the username and password that fits the user we created earlier.

* + 

This creates the username and password variables that will hold our hardcoded values. Then we create the shelter variable to hold our CRUD call that holds the username and password arguments.

* + 

Here you can import the data two different ways. The first way (as shown in the picture above as the commented statement) would be to create a variable that holds the **pd.DataFrame.from\_records(shelter.read({}))** statement. This will use the read() function and read the data from your CRUD module. The second way would be to hardcode in your csv file by placing the file in the same folder as your program and then create a variable that holds **pd.read\_csv(“filename.csv”)**. This will allow the program to read the actual file.

Both of these ways are viable solutions to reading your data.

**\*\*NOTE: Due to virtual lab difficulties, hardcoded csv file was necessary. \*\***

1. **Initialize App.**
   * 

In order to get your app to work, you will have to Initialize it using the code above. This will allow the dashboard application to run in the Jupyter program.

1. **Add in an Image (Optional).**
   * 

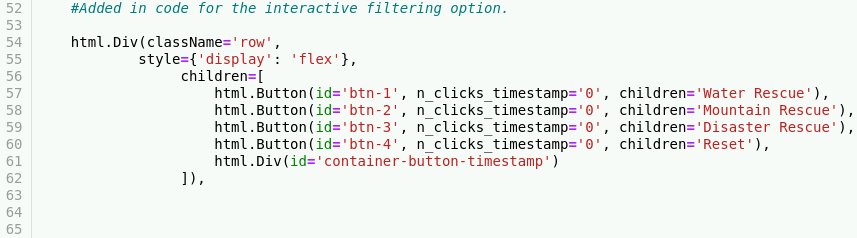
This step is entirely optional. If you wish to include an image into your dashboard program, such as a logo seen above, then follow the above code. The first variable holds the image file name (make sure that the image is in the same folder as your program). The second variable holds the base 64 package that helps encode the image.

1. **Create the app layout.**

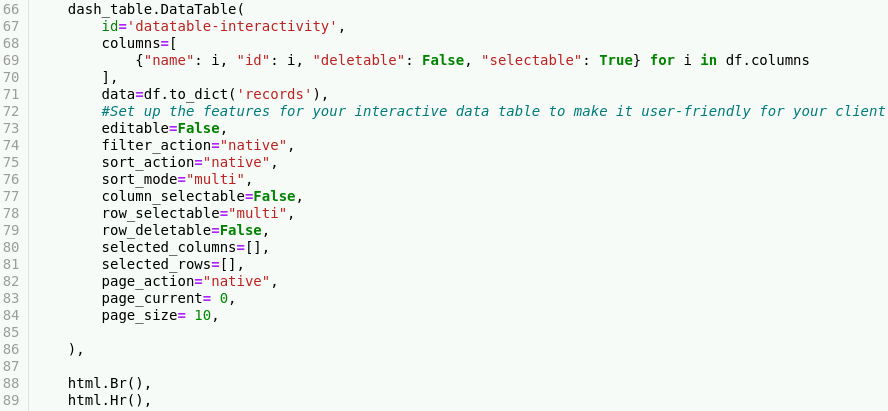
The app layout is where you will hold all of your html Divs that will contain the skeletons of the visuals you wish to display on your dashboard program. There are a wide variety of dash core components and dash charts that you can use with your data. We will be using buttons, dash table, and pie charts for this program. We will also be including extra code for a bar chart, choropleth map and geolocation charts that won’t show up in this tutorial but will serve as a guide if you wish to implement them. For dash core components [see here](https://dash.plotly.com/dash-core-components). For dash charts [see here](https://plotly.com/python/basic-charts/). The next series of screenshots will be the app layout and dash components.

* + 

This is the start of our app layout. First, the **app.layout** and a **html.**Div is used to create the container that will hold all the subsequent dash components and html divs. Next, the **html.H1** is used to give our program a title. This title can be changed to anything. Then we use the **html.Img**() to implement the optional logo. And finally, we use **html.Hr()** to create a break.

* + 

Next, we will create another **html.Div** that will house our buttons for our data table. To create these buttons we are using **html.Button()** and entering the respective information. We want to make these buttons interactive later, so we are also writing in **n\_clicks\_timestamp=’0’**.

* + 

The next thing we need to do is create our dash table. To do this we will use the **dash\_table.DataTable()** statement. Within this statement, we will first create a loop that will run through our data and create the columns and input the data.

We will then input a series of interactive features for the user to use when using the data table. For this project we will be using the above features such as **editable=False, filter\_action=”native”,** etc. Again, there are a variety of these features available and we recommend you look through the documentation and experiment with other features as well.

Finally, we will had a **html.Br()** and **html.Hr()** for a break and a line.

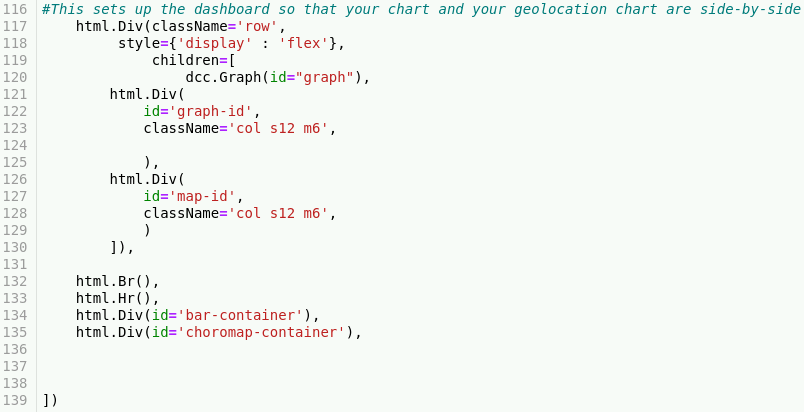
* + 

Now, we will create our pie chart. The above code is our skeleton for the pie chart. We first start with our div and then use the **html.Label([])** to give our pie chart a name. We also want this pie chart to have some interactivity, so we will be adding a dropdown list using **dcc.Dropdown()** within this, we will create an id (we will use this id later, so name it something easy to understand) and we will use the options list to add in our labels (the user will see) and the values (that correspond to our database).

Next we will add the value variable which will default our dropdown with that value. And add in a couple of features to stylize our dropdown.   
Finally, we will add one more **html.div()** which contains the **dcc.Graph** that will help us create the pie chart.

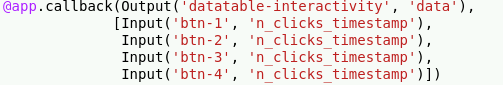
**\*\*NOTE: Don’t forget to close your app.layout = html.Div([]) after inserting the visuals you want to use. If you do forget, it will create a SyntaxError that is difficult to debug. \*\***

As an extra bonus, if you wish to also insert an interactive bar chart and geolocation chart, follow the below code. **(Due to technical issues with the virtual lab, these were unable to be used in the final program).**

* + 

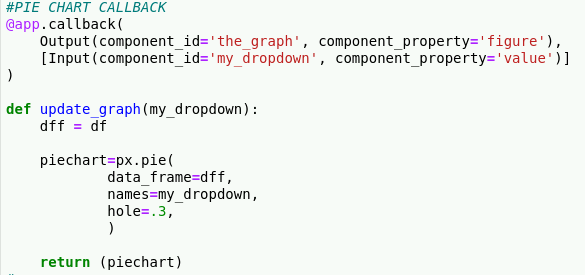
1. **Create the ‘Data Table Button’ Callback**
   * 

Callbacks are what makes our visuals that we implemented in the app layout work. We will first be creating a callback for the data table buttons. These buttons will provide functionality by helping users query for specific data. The callback will first take in the output and inputs. The output will be our data table and the input will be our buttons that we created earlier. See here:

* 

Next, we will create a function that will take our buttons as arguments and use them to create our query functionality. We will do this by using a series of **if** and **elif** statements that will let the program know what button we are pressing and once the button is pressed, it should query for the specific data. It will then return the query to the data table. See here:

* 

1. **Create the ‘Pie Chart’ Callback**
   * 

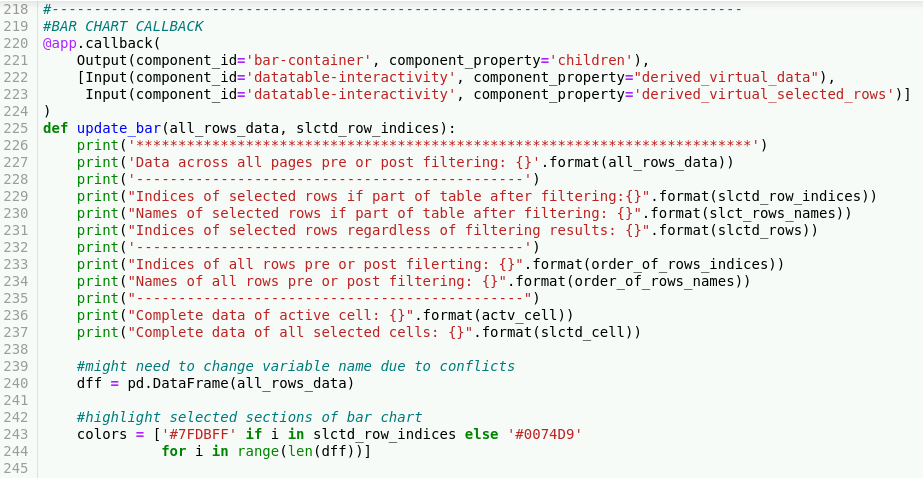
Finally, we will be creating our pie chart callback. This callback is pretty simple. Just like the last callback, it starts with us inputting our output and input. The output will include the id ‘the\_graph’ that we created previously and uses a component\_property of ‘figure’.

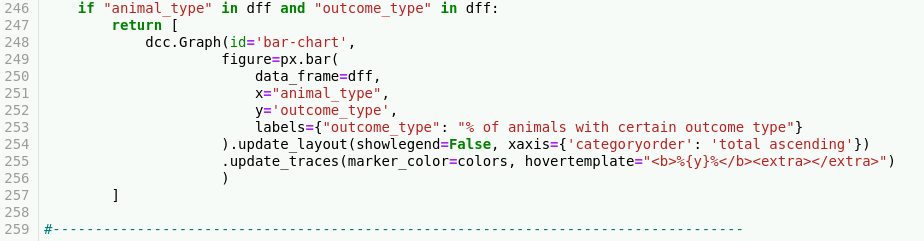
Then we will create a function named ‘update\_graph’ that takes the ‘my\_dropdown’ argument. Insides this function we will call a variable ‘dff’ that will hold our previously created variable ‘df’ and also the variable piechart that holds the **px.pie()** statement and contains the features inside that statement. Finally, we will return the piechart variable to create the pie chart.

1. **Extra Visuals (Optional)**

The next two visuals are not included in the final program screenshots that will be shown in the next section. However, if you want to include the bar chart and choropleth map, we will provide the code to do so below:

**Bar Chart**



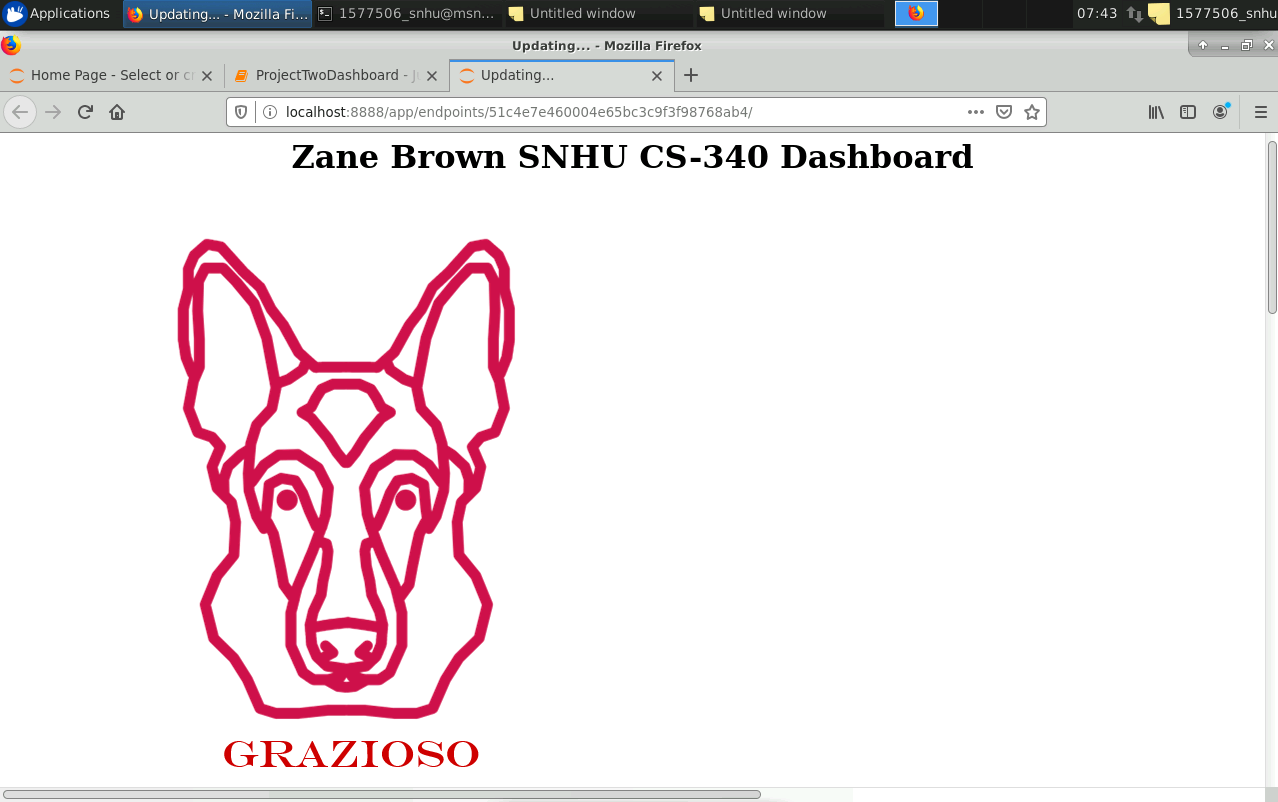


Choropleth Map

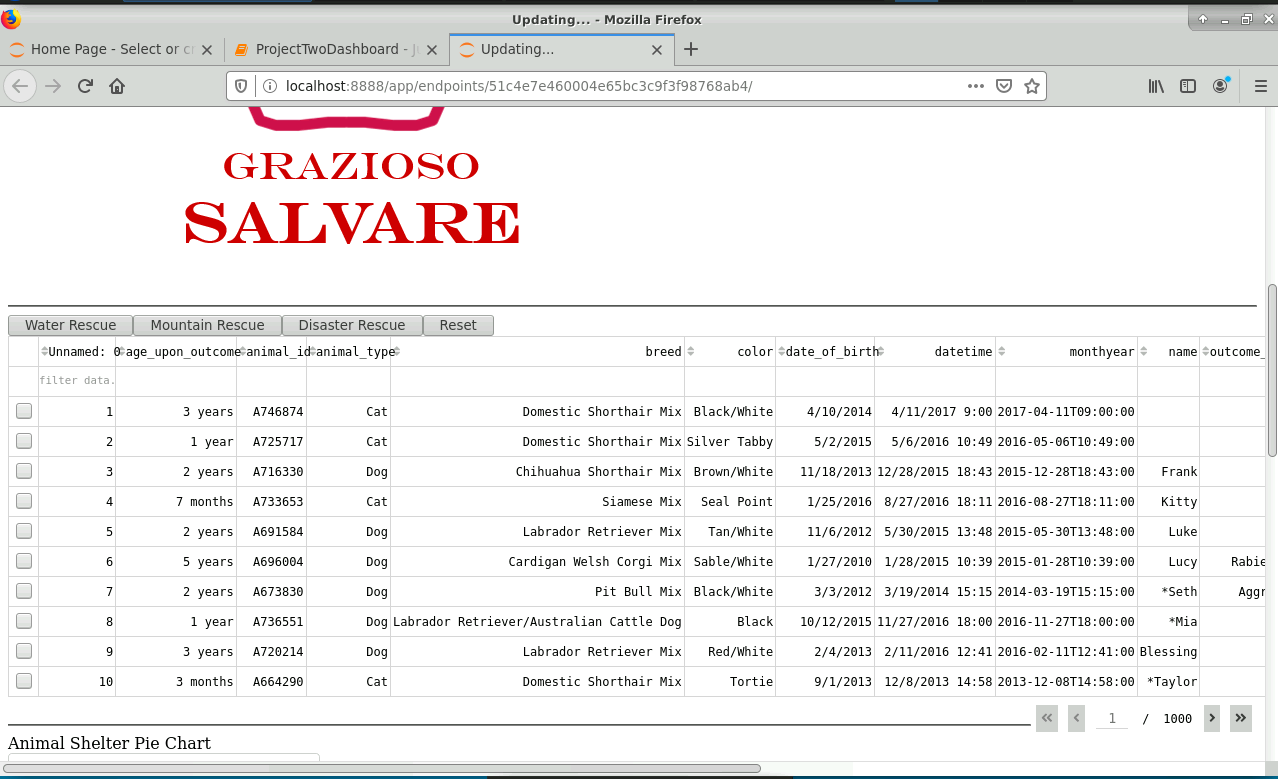
* 

Dashboard Program Output Screenshots

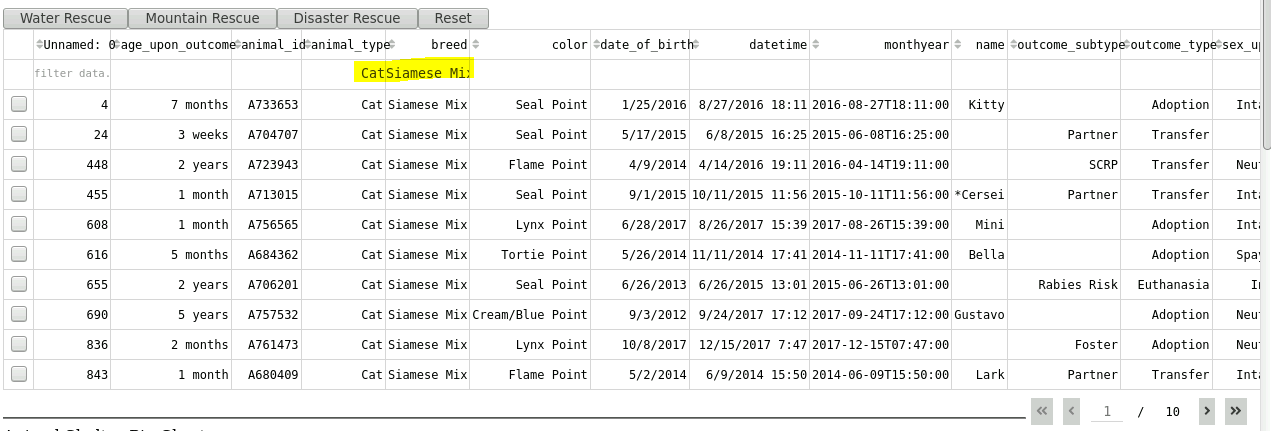
The following screenshots show the output of the program. If you followed the code provided above, you should have similar output.



The program starts with the Title and Logo.

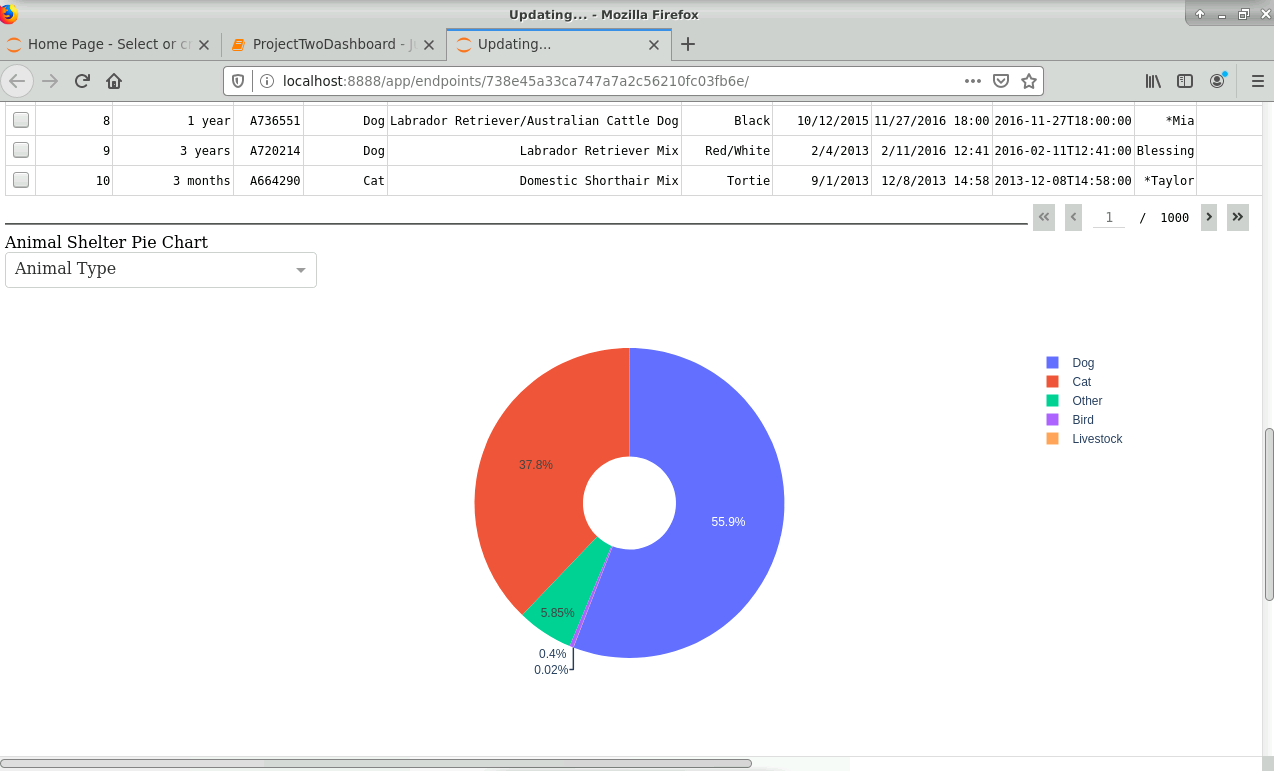


The program contains a data table with our csv data. There are several ways to filter data and query for data. For example:

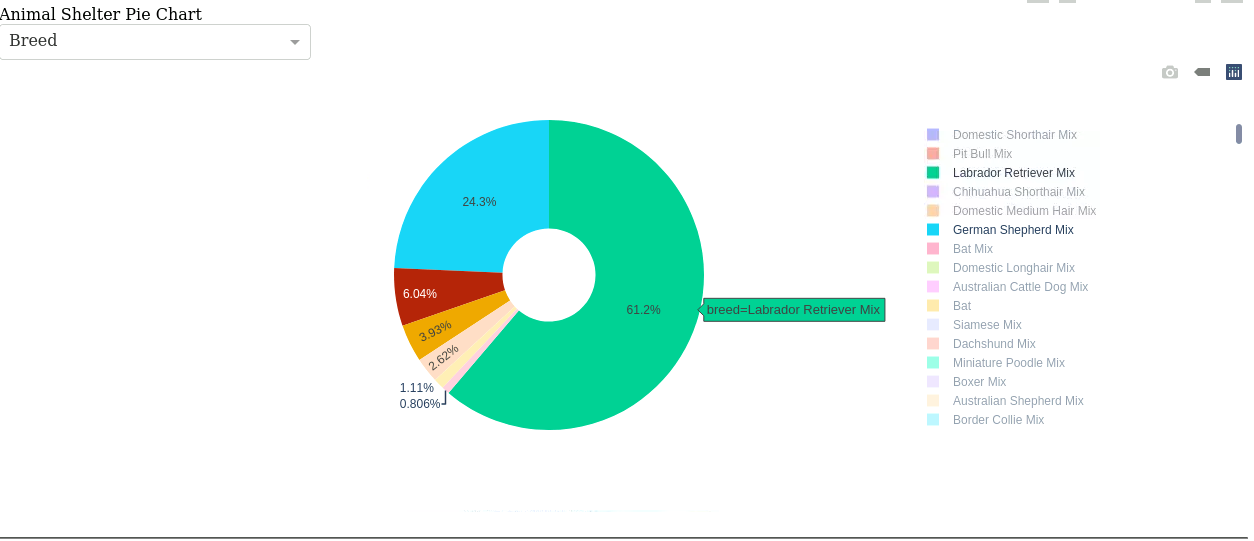


We can query our data by using the filters we implemented. In the above picture, we queried for the ‘animal\_type’ Cat and ‘breed’ Siamese Mix. Additional filters may be used to query for more specific results. You can also reset your data back to unfiltered by erasing any search entries.

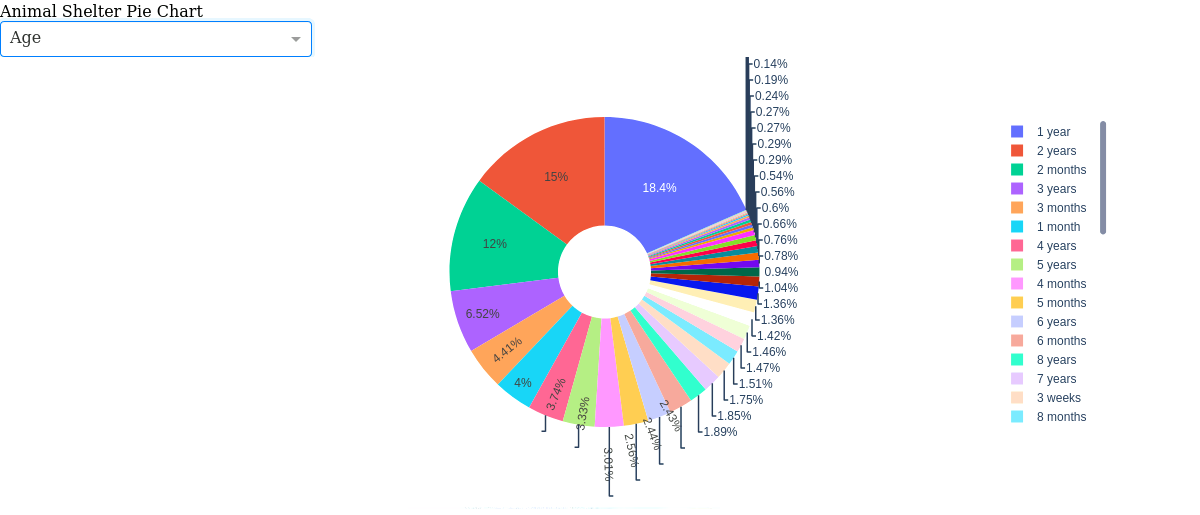
The buttons ‘Water Rescue’, ‘Mountain Rescue’, ‘Disaster Rescue’ and ‘Reset’ are also available to provide query functions.



The program also outputs a pie chart that has a dropdown menu and a interactive key to the right. Both the dropdown menu and key can help query. For example:



Using the pie chart, we are query for the percentage of Labrador retriever mix, German shepherd mix, Siberian Husky mix, Rottweiler mix, German shepherd, Siberian husky and rottweiler.



Additional Example: Using the pie chart we can also query by age. Using the pie chart, we can query through a number of data in the AAC database.

**Tools**

The tools that we will be using to create this program are:

* Jupyter
* Python (Anaconda distribution)
* MongoDB
* Plotly Dash
* Additional External Research Sources

**Jupyter**

Jupyter is a web tool that provides an online interface that allows users to create python code and will help us create our web application. This is our main IDE for both the CRUD module and the dashboard program. Any IDE, such as, PyCharm, may be used to create these programs. We have already discussed the installation of Jupyter. If you need assistance with installation, please refer to the first section of this README.

**MongoDB**

MongoDB is a database that stores JSON documents and collections. This tool is used to high volume storage (GURU99, 2020). MongoDB can be used to create efficient and high functioning web applications. For more information on [MongoDB check their website](https://www.mongodb.com/what-is-mongodb).   
 We chose to use MongoDB with Python to create our web dashboard program due to the many benefits that MongoDB provides. Some of these benefits include MongoDB providing the ability to have a single document model, flexible schema, and saving RAM. Because MongoDB supports a single document model, MongoDB allows for documents to be used as a basic storage unit. This lets developers save data in a single document without treating it as a whole (Web Development, n.a.). Additionally, thanks to MongoDB having flexible schema, developers are able to create any number of fields to an already existing document or collection. MongoDB can then take these and rearrange them accordingly. (Web Development, n.a).

If you need help with the MongoDB installation, see the first section of this README.

**Additional Resources**

In addition to the physical tools that were used to create the functionality of this program, other research tools were used in order to help us understand how to create the program. The first resource that was used was the [Plotly](https://plotly.com/python/basic-charts/) website. This website provides all of the documentation about the dash components and charts. This website was used to research and understand the code that was used in the program.

The other additional resource that was used was the [Charming Data](https://www.youtube.com/channel/UCqBFsuAz41sqWcFjZkqmJqQ/featured) YouTube channel. This channel is focused on Plotly dash tutorials and gives wonderful explanations about how to write code and how the code functions.

**Challenges**

Throughout the course of this program, many challenges were encountered. Many of them were overcome and some were not. In the CRUD application, most of the challenges that were met were due to Virtual Lab issues and authentication issues. Research, I.T. department and the class professor were vital to overcoming these challenges. Once these problems were alleviated, the CRUD program worked with no issues.

The next challenges involved the dashboard program. Here, I met many challenges that were very difficult to solve, and some were not solved due to technical issues. The first challenge that was faced was the program hanging. Due to a series of mistakes in the code, the program would hang when I implemented a specific feature. The program would hang when I implemented the pie chart and geolocation map. However, I was able to fix the pie chart and provide functionality for said pie chart. I was not so fortunate with the geolocation map. For two weeks, I researched the issue, discussed the issue with the professor and other students and could not find a solution. The code looked almost identical to other’s work, but the map would not display and would cause the program to hang. This feature was unfortunately skipped due to these unfixable issues as well as the time constraint.

Additional challenges were met when I implemented the interactive bar chart and choropleth map. The program did not hang and did not produce and error. Instead, it did not output these two charts at all.

Despite the many challenges that were met, I was able to overcome a significant amount of them and in the end, I am very happy with what I was able to create despite the number of problems. I was able to learn a lot from creating this program. I was able to understand how to create these functions and modules and know what to do when code breaks.

**Contact**

Your name: Zane Russell Brown

Contact: [Zane.brown@snhu.edu](mailto:Zane.brown@snhu.edu)

Reference:

Advantages of Using MongoDB with Python - Web Development. (n.d.). Retrieved December 14, 2020, from <https://sites.google.com/site/webdevelopmentnews/advantages-of-using-mongodb-with-python>

Charming Data. (2020, December 12.). Retrieved December 14, 2020, [video] from <https://www.youtube.com/channel/UCqBFsuAz41sqWcFjZkqmJqQ/featured>

What is MongoDB? Introduction, Architecture, Features & Example. (n.d.). Retrieved November 23, 2020, from <https://www.guru99.com/what-is-mongodb.html>