# CS 340 README – Jon Frodin

## About the Project/Project Title

The purpose of this web application is to provide a user-friendly interface to sift through data stored in a database, powered by Mongo DB, to return desirable combinations of breed, sex, and age to training rescue dogs. The project consists of a database, an API, and a dynamic dashboard.

**Motivation for Using Mongo DB**

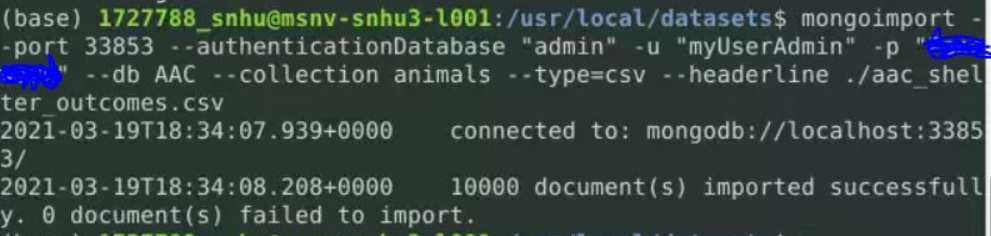
Mongo was used, specifically with this program, because it provides a quick setup of the database from a csv document as well as a Python-friendly interface. Python can use database tools like SQL, but the syntax is so different that it can be cumbersome to switch back and forth. The selection tools when running the basic CRUD functions of a database are significantly more complex in a SQL database than they are with a Mongo DB, using Python.[[1]](#footnote-1)

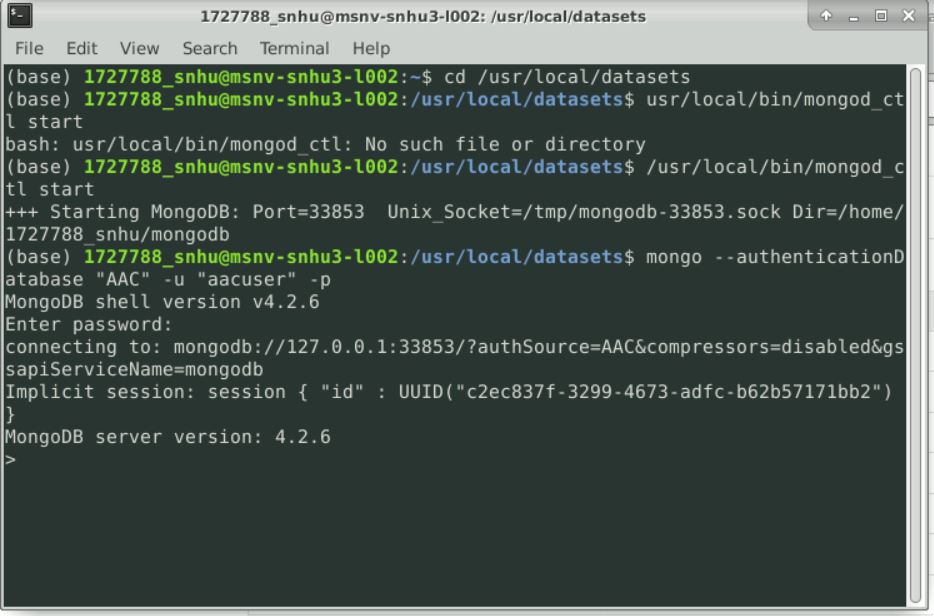
**Motivation for Using Dash**

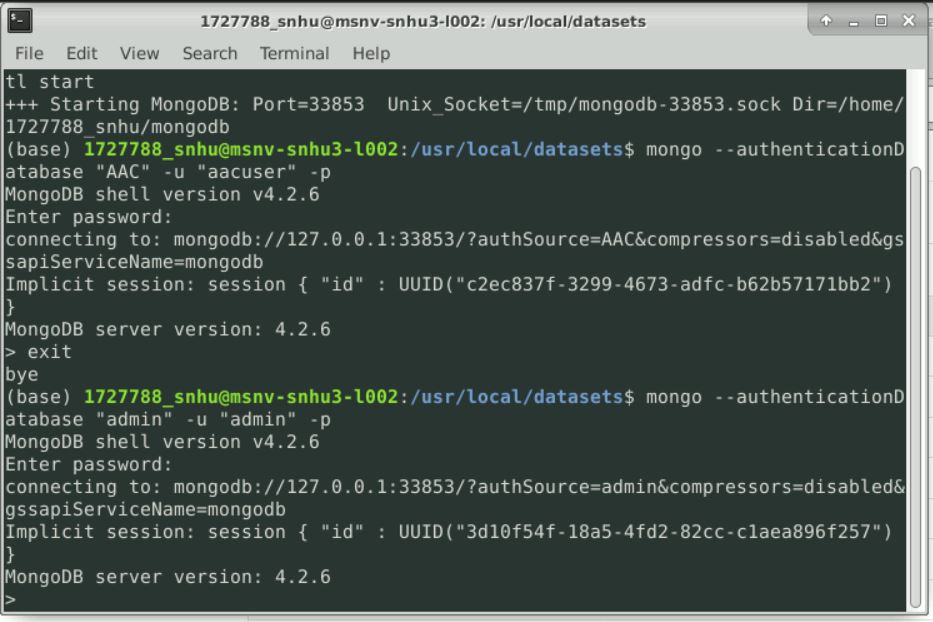
Dash, as the tool used to build the dashboard, was desirable due to its dynamic nature. Dash is a react JavaScript based tool that provides an incredibly responsive framework. Dash involves html Dash tags that control outputs to segments. Then, updates to any of the target inputs specified in the app callbacks process based on instructions programmed in the Python module.

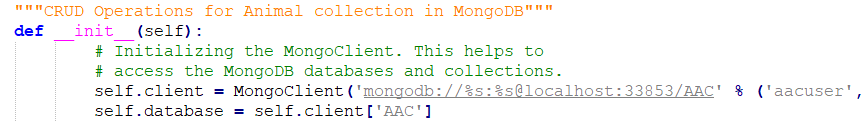
**Getting Started**

To create a local copy of this program, you will need several steps. The first is to create a Mongo Database. The second is to program a Python CRUD module to access that database. The final step is to create a Dash web application that implements the Python CRUD module. This is a multi-layered application that reacts to changes in the table in real time.

1. Create a Mongo Database and create a database called AAC.
2. Create a user with read/write privileges to that AAC database.
3. Import the data from aac\_shelter\_outcomes.csv file
   1. The csv file has some issues. I overcame the import issue by passing my user and password for the AAC database in the import command in the Linux terminal.
   2. Some people were able to import the AAC csv file by running Mongo via -noauth but I was unable to. If you have the same issue – use step a.
   3. Update the port number (Mongo gives you the port number when you start the service) on the file import.
   4. Example import: 
   5. Login as “aacuser” or as “admin”





1. Update the port number on localhost of the Animal\_App.py Python code.
   1. This will be essential when importing the
2. Update the “aacuser” and “Password” to the username and password you created.
   1. Sample Initialization: 
3. To run the tests, add the test code to a Jupyter notebook and ensure that the test data for the create function is different each time or delete the added record between tests.
4. Create a new Dash web application dashboard and configure with the desired HTML/CSS layout and appropriate ids for the data frame, map, and chart.
5. Create an app callback to populate the initial data frame with all of the data.
6. Create radial options and program the database queries based on the desired breed specifications for the client.
   1. During this stage, complex queries are needed to combine many searches into a parent search and return the results. This creates many syntax problems if you are not careful.
7. Create an application callback to update the map with the first item of a given category until the user selects an item. Then, create functionality that determines the user selection and displays that on the map instead.
   1. This stage was quite complicated. I know I didn’t end up doing it the absolute best way possible because mine only allows up to five selections simultaneously.
   2. Care must be taken when isolating the selected row. For example, the provided starting point used a variable included with Dash called “supplied\_virtual\_row\_ids” which was nearly impossible to use. Skip that and go straight to “selected\_rows” as you will have a much easier integration time.
8. Create a pie chart from the displayed data on the screen at any given moment. With the pie chart in hand, create an application callback that updates the pie chart with the data that is filtered from the display filters.
   1. This had several problems. The plotly documentation I could find said that the parameter “values” could be int or string. Neither of these actually work. Technically int does, but it doesn’t generate the results you want. The documentation also says you can use a row or column from the data frame.[[2]](#footnote-2) That appears to be completely inaccurate because I tried many ways to implement that and none of them worked. It actually needs an index of names and an array of numbers. The index of names should be under “names” and the array of occurrences should be an array under the variable “values”.
   2. Once you have the pie chart populating, develop methods that implement the functionality that you are looking for and the correct target data (I used ‘breed’).

## Installation

The tools you will need to run this include Jupyter Notebooks, Python for command line, and MongoDB. The installation of each is detailed in labeled sections right below this line.

**Jupyter Notebooks**: Jupyter can be installed from the command line in any major operating system using the simple instructions here: <https://jupyter.org/install>. For detailed instructions like Proxy servers for Windows, Mac, and Linux, follow the instructions here: <https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html>

**Python**: Detailed installation instructions for Python are available here: <https://realpython.com/installing-python/>. Once you have Python installed, you should be able to use this program from the Terminal on Mac or Linux or from the Command Prompt for Windows.

**MongoDB**: MongoDB comes in Community or Enterprise editions. Detailed instructions for the installation and downloading of MongoDB are available here: <https://docs.mongodb.com/manual/installation/>.

**Plotly**

Plotly must be imported in order to generate the proper charts. Plotly is a charting tool for Python applications and can be imported directly into your Python module from your Jupyter notebook. If you need a local copy of Plotly installed, see the documentation here: <https://www.journaldev.com/19692/python-plotly-tutorial#:~:text=Installation.%20To%20install%20plotly%2C%20open%20a%20terminal%20window,to%20install%20to%20collect%20dependencies%20and%20download%20them%3A>

**Dash**

Dash is a framework used to build web applications. You can import the Dash Core Components into your Jupyter notebook and you can install Dash using the following information: <https://pypi.org/project/dash/>

**Pandas**

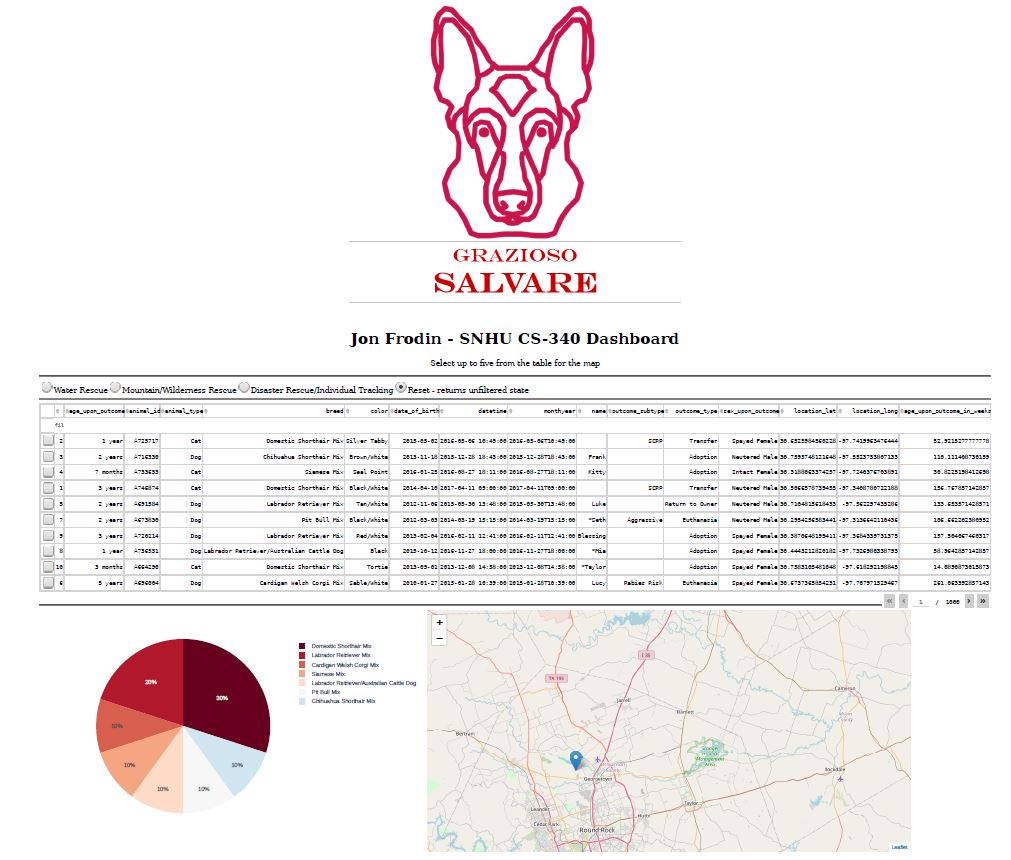
Pandas is used in this web application as well. Pandas is a tool for Python that creates the data frames. Pandas has other dependencies and information that should be reviewed before use here: <https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html>

## Usage

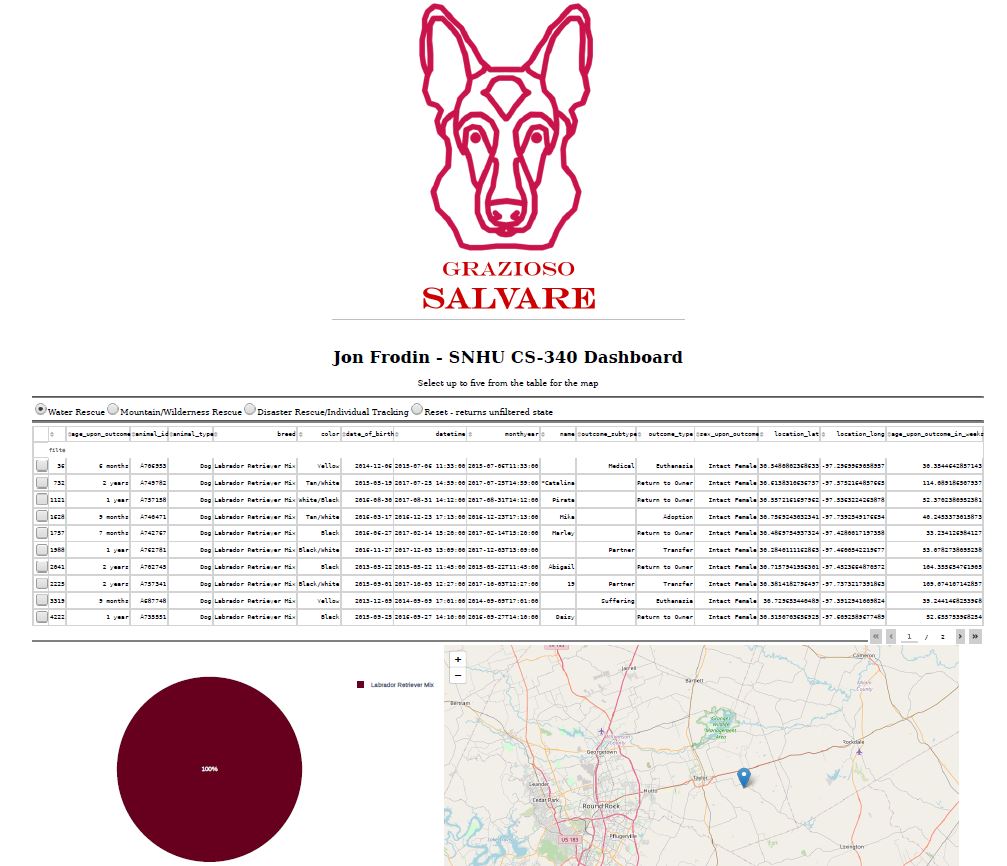
This application currently has 3 main functions. The first uses radial buttons to sort data based on breed, age, and sex characteristics for the different types of rescue dogs desired by Grazioso Salvare. Clicking any of the radial buttons runs the database queries and returns an updated data frame with the desired results. The final radial button titled “Reset” reverts the table back to the un-filtered state. The second function is the map and dynamic map updates. This starts the map with a marker at the position of the first item listed in the data frame. Once the user selects a row (or, up to five at a time), the map changes the marker (or adds markers for multiple selections) to the selected row(s). The third function is the dynamic pie chart. This simply sorts the data by breed and outputs a pie chart of the current data frame (not the full database) based on breed of animals shown.

This application currently has four functions. The first two obtain data from the user on what to enter into the database (create) or what to search for in the database (read). The second two functions actually run the creation and the search. The obtain functions ask the user for specific input (for the read data a key is requested and a value to run the search, while in the create function the keys are provided and only the values are requested). The create function works by passing a Python dictionary into the MongoDB as a new item and returning Boolean True if it is successful or False if it is not. The read function returns False if nothing is available or it returns the data if matching data is found.

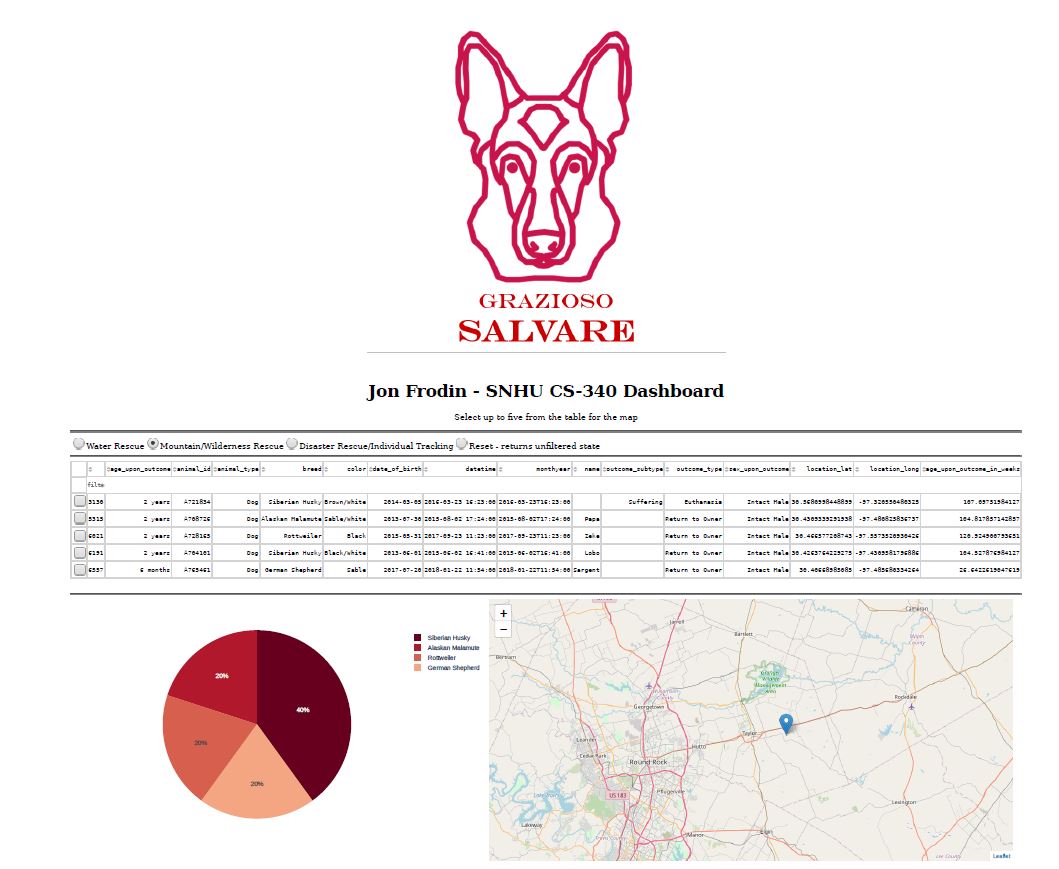
**Reset (Show all Data)**

****

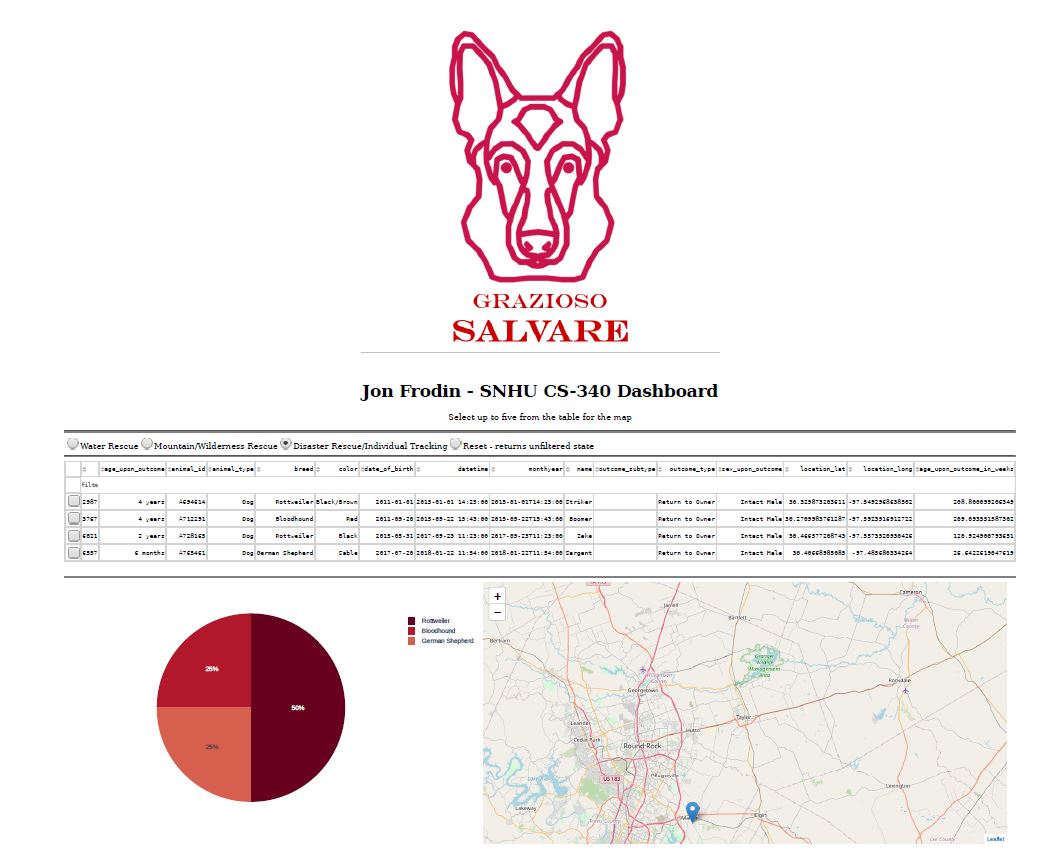
**Water Rescue**

****

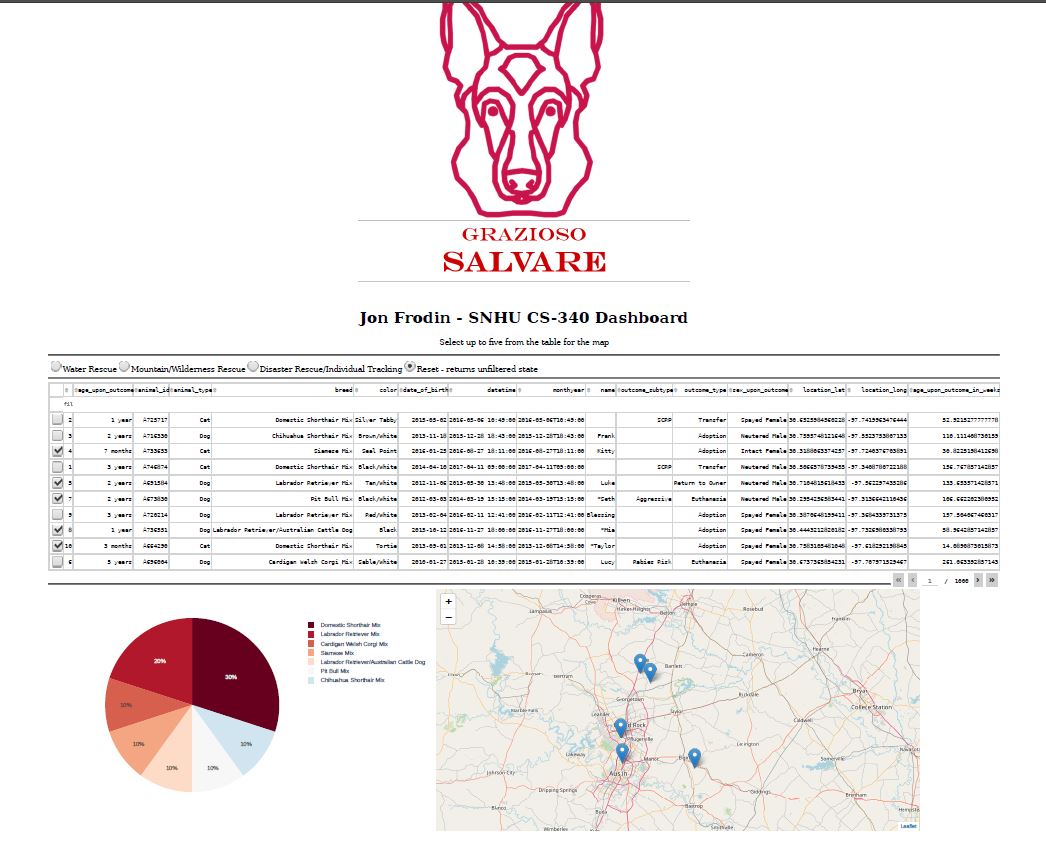
**Mountain Rescue**

****

**Disaster Rescue**

****

**Multi Select**

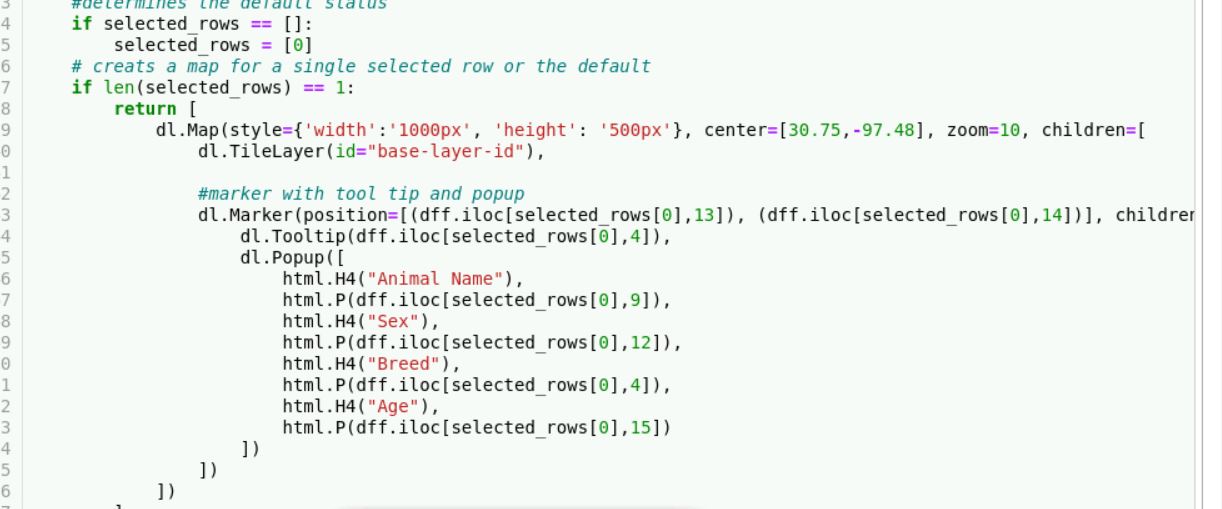
****

**Code Samples from Dashboard**

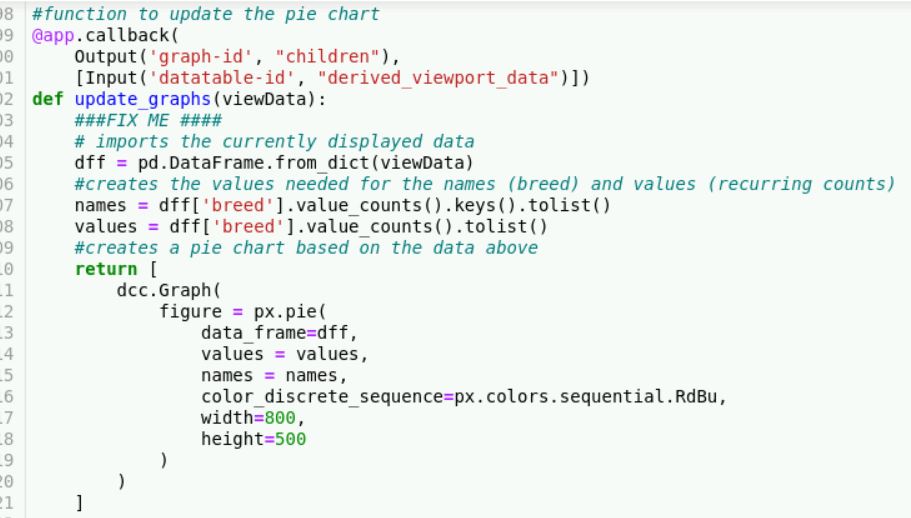
*Complex Query*

**

*Map Markers*

**

*Pie Chart*

**

### Code Example (CRUD Operations)

*Create*:

create():

try:

if data != null

insert\_result = self.database.animals.insert\_one(data)

pprint(insert\_result)

return True

else:

raise Exception(“exception content”)

except:

return Fale

*Read*:

Read(target):

Try:

If target != null:

read\_result = list(self.database.animals.find(target))

Pprint(read\_result)

Return True

Else:

Raise Exception(“exception content”)

Return False

Except Exception as e:

Print(message + e)

*Update:*

Update(source, destination, count)

Try:

If source != null:

If count == 1:

Update\_result = update(self.database.animals.updateOne(source, destination))

Pprint(update\_result)

Return True

Elif count == 2:

Update\_result = update(self.database.animals.updateMany(source, destination))

Pprint(update\_result)

Return True

Except:

Raise Exception(“exception content”)

Return False

*Delete:*

Delete(target, count)

Try:

If target != null:

If count == 1:

Delete\_result = delete(self.database.animals.delete\_one(target)

Pprint(delete\_result)

Return True

Elif count == 2:

Delete\_result = delete(self.database.animals.delete\_many(target)

Pprint(delete\_result)

Return True

Except:

Raise Exception(“exception content”)

Return False

**Code Examples (Dashboard)**

*Radial Buttons*

Import view data

If radial\_one is selected:

Run complex query

Update view data

Return view data

*Map*

Import view data

If selected\_rows is None:

Selected\_rows = 0

If length(selected\_rows) == 1:

Return map with 1 map marker and tooltip

Else if length(selected\_rows) == 2:

Return map with 2 map markers and tooltips

Etc.

*Pie Chart*

Import view data

Names = data(breed) – obtain and transfer to list

Values = data(breed) – obtain occurrence counts and transfer to list

Return graph using the view data, names as the search key, and values as the pie slice values

### Tests

The create function has two unit tests and the read function has two unit tests. One test in each tests the failure case and the other tests for a successful case. The create functions test whether the return value is true or false. There are several more tests to test the various functionality of each CRUD method. A code example is provided here:

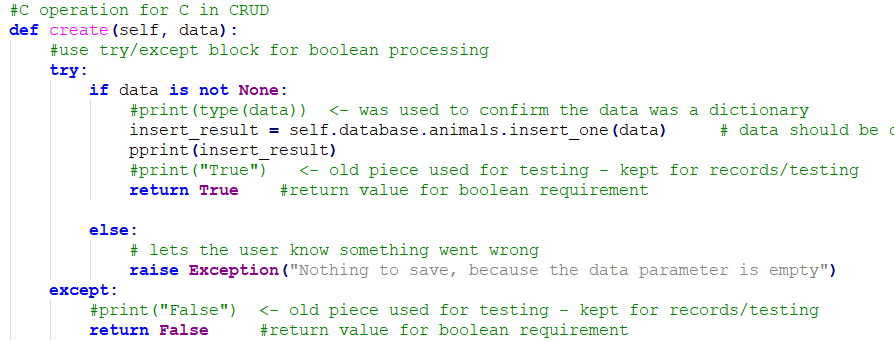
testData():

testShelter = animalShelter()

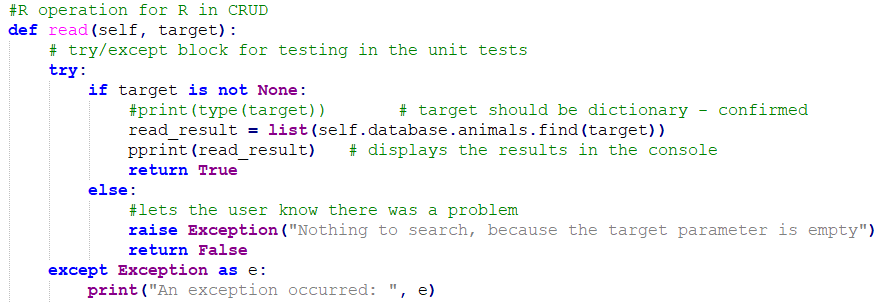
assertTrue(testShelter.create(sampleData))

### Screenshots

Sample Create:

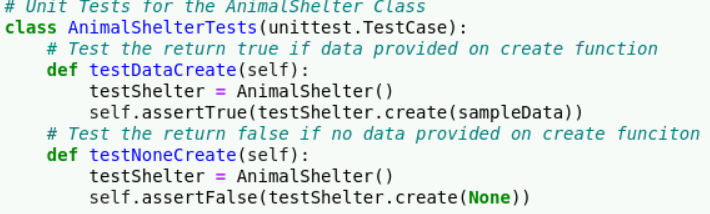


Sample Read:

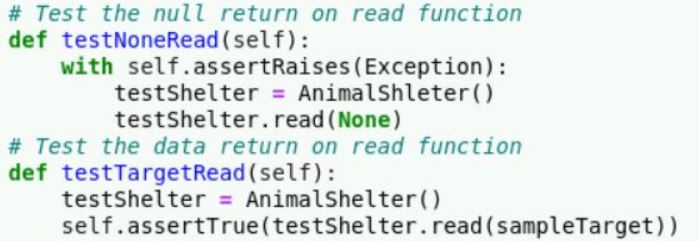


Sample Tests:

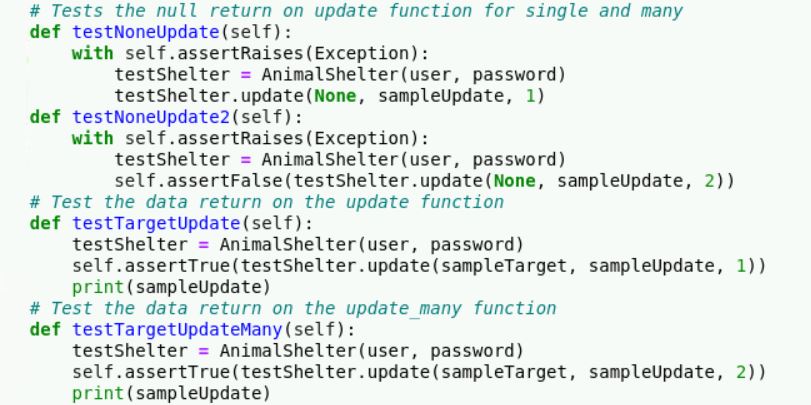
*Create*



*Read*

**

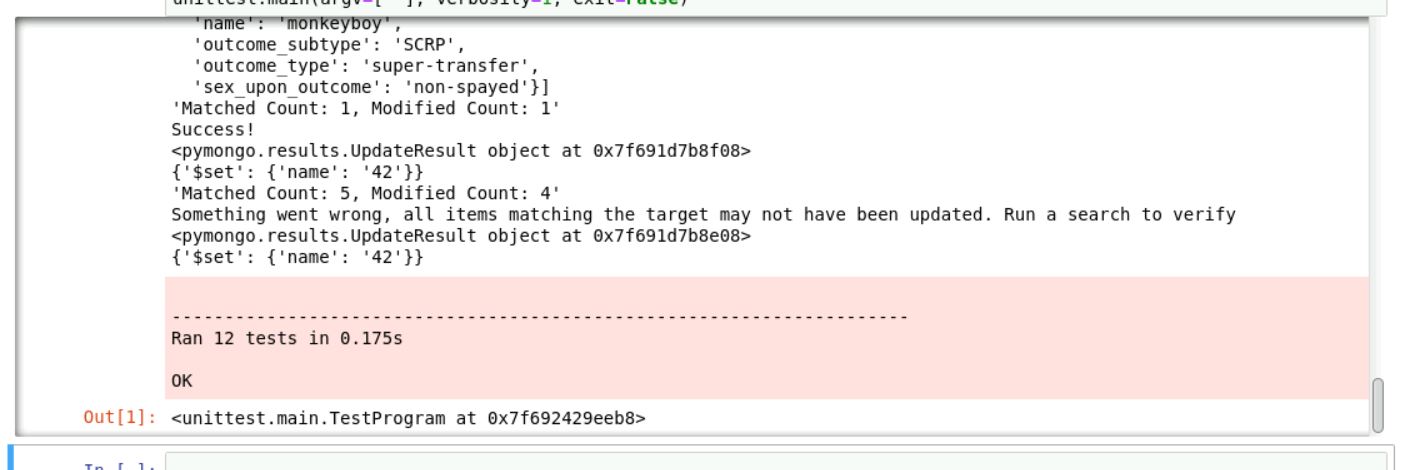
*Update*

**

*Delete*

**

*Test Results:*

**

## Roadmap/Features (Optional)

Features to come:

My project allows up to 5 items to be selected for the map. I would like to update that to allow any number of items to be selected and populate markers on the map. I would also like to update the map feature to allow an automatic clearing of the data when the filter (and therefore the data frame) is changed. Lastly, I would really like to update this with a more dynamic pie chart where the user could update the pie chart based on some sort of dropdown. So, if they wanted age to be a determining factor in the chart instead of breed, for them to be able to select that and still have the chart working.

Known issues (CRUD and Tests):

The big known issue at this time is that when you run the unit tests, they can only be run once without getting an error on the create function. I have no idea why that is at this time because I can manually run the create function all day with the exact same data, but only once via the test. If I run the test a second time *after* accessing the database via the terminal and deleting the item I have added, then it works again. I think it is trying to verify the new addition, but I don’t see how that matters to the test since the create function itself is capable of adding the same item again and again and I have tested this. The update tests, as a note, will also only work one time because once names are updated, if you do not provide unique data, then they will not be updated again.

Known issues (Dashboard):

I had a hard time making the Dashboard able to show multiple selected rows at once. I rectified this in a clunky way by having several, manually-coded, output statements for options of 1-5 for brevity. I did not want to keep going to 10 or higher because of the huge number of lines that would make in the code. I would like to find a cleaner and more concise way of coding this. The app also seems to slow down if you do too much. Lastly, if you do not de-select an item when you switch filters, then the selection of items does not work again to update the map. I looked for ways to fix this, but it is a bug persistent in the final version.

## Contact

Jon Frodin at jon.frodin@snhu.edu

1. https://www.slideshare.net/jnunemaker/why-mongodb-is-awesome/61-var\_result\_dbitemsmapReducemap\_reduce [↑](#footnote-ref-1)
2. <https://plotly.com/python-api-reference/generated/plotly.express.pie.html> [↑](#footnote-ref-2)