# CS 340 Project 1 README

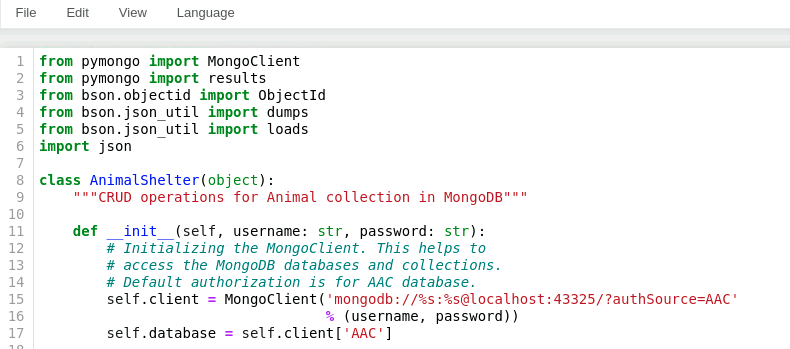
## Description and Required Functionality

The purpose of this module is to provide reusable CRUD functionality to serve the middle layer operations within a NoSQL (MongoDB) database and the user interface, as well as provide the top layer of user interactive functionality. The middle layer will contain the object instance of the database connection as well as those functions needed to create, read, update, and delete documents within the database, executed in response to the front-end user interface commands. The top layer contains the Dash component to organize and display the filtered data, the filter buttons, and the corresponding visual information in pie chart and map forms.

## Tool Usage

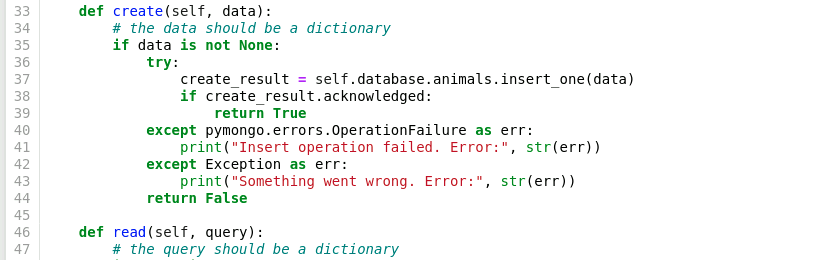
The PyMongo driver was used to establish the server using its MongoClient, and to access the database through the server. The combination of Python, PyMongo, and MongoDB is advantageous because they have the Python language in common. PyMongo is even written in Python, and MongoDB uses JSON formatting for its document data structure.

The AnimalShelter class creates a database connection object to add functionality and CRUD operations to our module. The \_\_init\_\_ method initializes the connection information when the class is imported and called from the front end module(s):

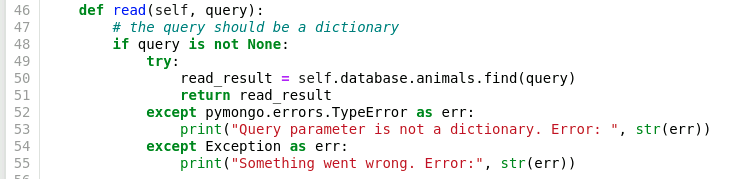


In the above screenshot, the \_\_init\_\_ function initializes the MongoClient instance with the corresponding mongodb address prefix, followed by the %s percent-escaping of the username and password, contained in the arguments within the parentheses at the end of the line. Replace “43325” with your port number from the database “status” command from earlier. “?authSource=AAC” sets the authorization to the AAC database, so that our AAC user credentials (“userSmith”, “expiredtwinkies20”) provide authorized access when the class instance is created later.

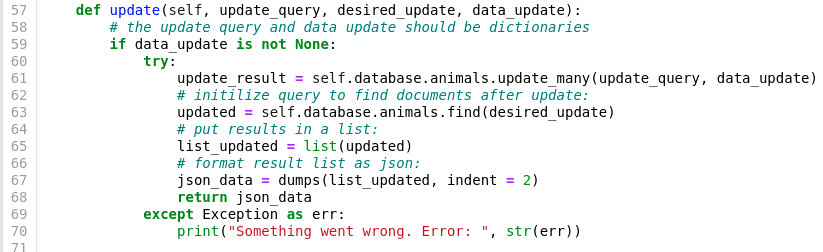
The create() method below adds new animals to the database. It contains a “data” argument which will use a dictionary variable from the front end user prompts when adding a new dog to the database. “Self” is a required first argument for Python methods. It returns “True” to show that the new animal document was added. Exceptions are handled for both OperationFailures and general errors:



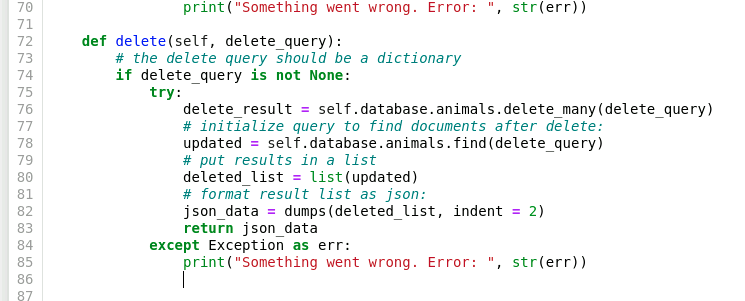
The read() method below performs a search of the database for matching results. It contains a “query” argument which will use a dictionary variable from the front end user prompts to find one or more dogs in the database. It returns a cursor to allow for incremented results pages. Exceptions are handled for both TypeErrors and general errors:



The update() method finds and changes document data according to a search query and the document fields specified. It contains an “update query” argument which captures the query of the documents to be updated, a “desired update” argument to locate the updated information after the method is run, and a “data update” argument which marks the desired update information as the target of the set operation. It returns a list in JSON format of the updated documents. General errors are handled via Exception:



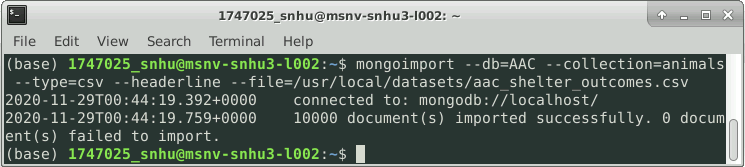
The delete() method below finds and deletes documents matching search criteria. It contains a “delete query” argument which captures the query of the documents to be deleted. The results are formatted in JSON and printed to the screen. If all documents matching the query were deleted, an empty list is returned. General errors are handled via Exception:



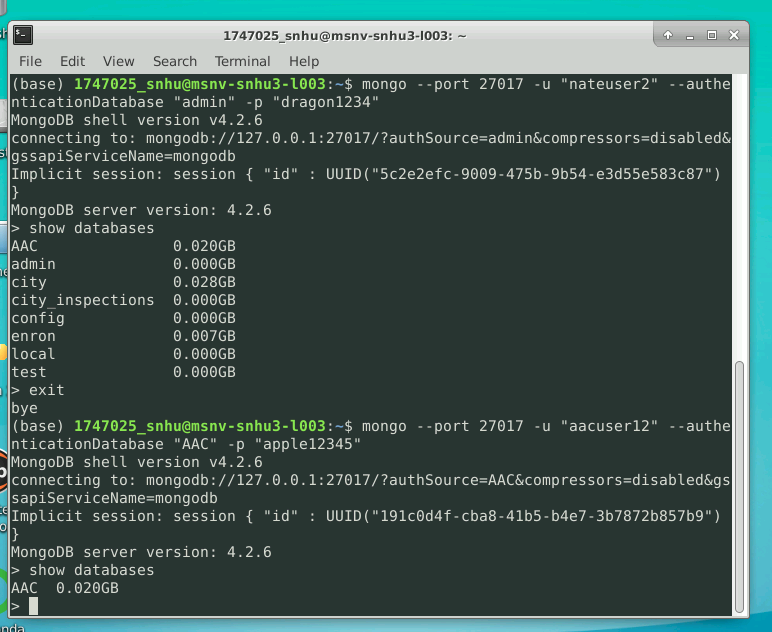
The Dash framework with Python was used to create the front-end user interface. Dash was chosen because of it’s advantage of avoiding the need to learn css, html, or javascript, and as such to permit the full-featured creation of a web application knowing only Python. Dash provides the view structure for the application by integrating html into the dash layout, and labels the interactive elements to tie them to controller functions. The controllers of the application are written in Python, and integrate the functionality of the CRUD module with dash callbacks that respond to user button clicks. A button click uses the CRUD module to send a query which has been tailored to the button’s intended filter. The data is then formatted and output to the datatable, as well as being displayed in the pie chart to show the number of dogs of each respective breed within the filter category.,

**Demonstrations**

Open a new terminal window and import your dataset using the mongoimport tool. Specify the name of your database, collection, and target file location.

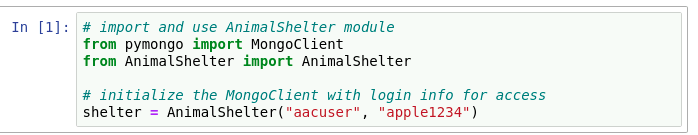


Demonstration user authentication, logging in with both administrative and AAC database user accounts:

**

Test execution of module CRUD functions:

First, the MongoClient is imported, as well as the AnimalShelter module. The AnimalShelter object is instantiated with the AAC database user’s username and password as arguments. This creates the database connection and authorizes the user to perform read/write actions.



Database and collection variables are instantiated for later use by object methods.

 A new animal variable holds the document information in dictionary format, then is used in the method argument to create a new dog document. The result “True” indicates successful document creation.

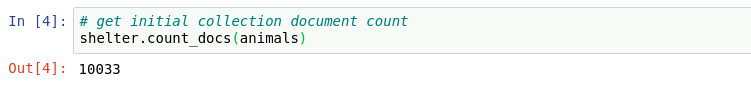
A new animal variable holds the document information in dictionary format, then is used in the method argument to create a new dog document. The result “True” indicates successful document creation.



### Tests

*Describe and show how to run the tests with code examples.*

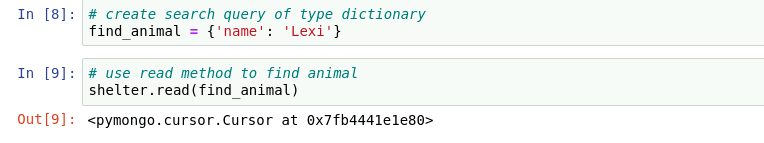
Using a count() method for testing purposes, the initial document count in the collection is found before creating a new animal:



The create() method requires a dictionary variable, which is then the argument used in the method call. The result of creating a new document is shown in the updated count, increased by 1:



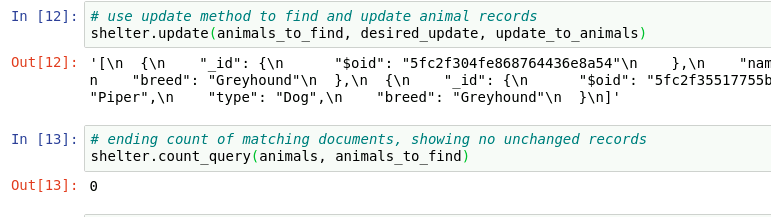
The document is found after creation using a query variable “findAnimal” as the argument for the read() method. The result is the cursor.



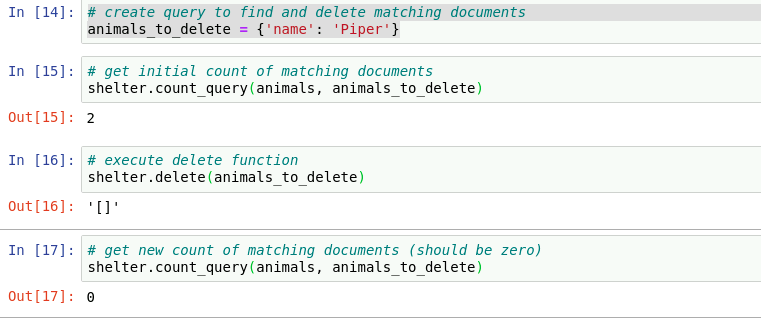
The update() method requires the query variable of “animals to find”, the information replacing the old information in “desired update”, and the setting variable for the update method “update to animals”. Before running the update, we can find the current number of documents requiring an update:



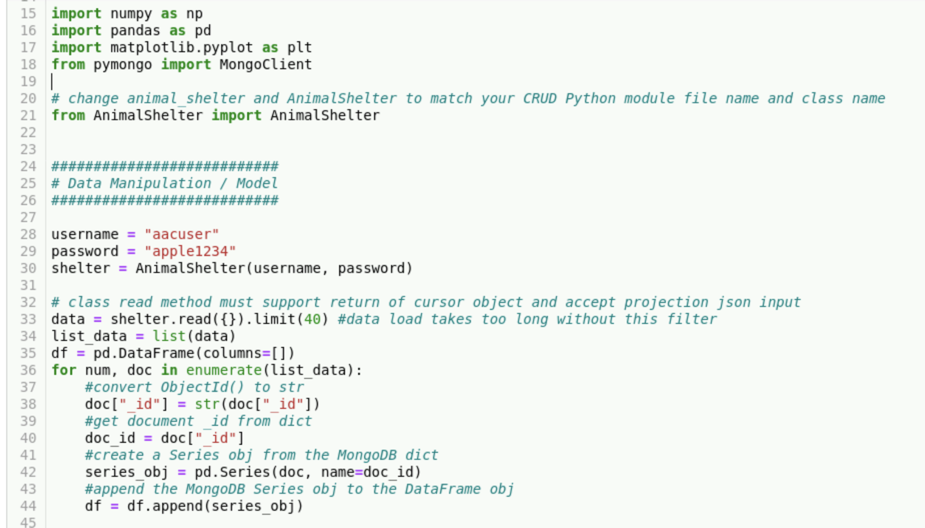
Using the update() method with the corresponding dictionary arguments, two documents are found and updated, then the results are converted to JSON format and displayed. The count is run after the update to find any remaining documents not updated, using the same argument “animals to find”:



The delete() method requires a dictionary argument to find and delete the desired documents. Here, we’ll find and delete the updated documents from the previous step, and then check that they no longer exist:



After importing the necessary components, the login parameters are hardcoded, then an instance of the CRUD module uses the login username and password. Next, a starting set of data is queried from the database using the CRUD module’s read method, limiting results to 40 for the sake of speed. A Pandas dataframe contains the query results after they’re converted to a list. The for loop on line 36 stringifies the \_id column of the JSON data in order to make it iterable, and the dataframe is then rebuilt as df on line 44.



The Dash app instance is instantiated on line 49. The Grazioso Salvare logo file location and the encoding of it with base64 allows it to be read, and later placed in the layout on line 60. The app layout itself begins with the Dash method to add an html divider, with the formatting parameters, my unique identifier (Nate Holcombe’s Dashboard), and the image placement at the top left corner of the page. The next divider on line 64 sets up the filter buttons with their titles, identifiers, and click counters.



The datatable is held in the dash\_table component, with its parameters, followed by the pie chart on line 95, and a final divider for the map.



In the Controllers section, the first callback responds to the click of a button filter, returning the updateTable function that sends the CRUD query and builds a new set of data to be displayed in the Output table. The query sent depends on which button is clicked, using the conditional structure of four elif statements.



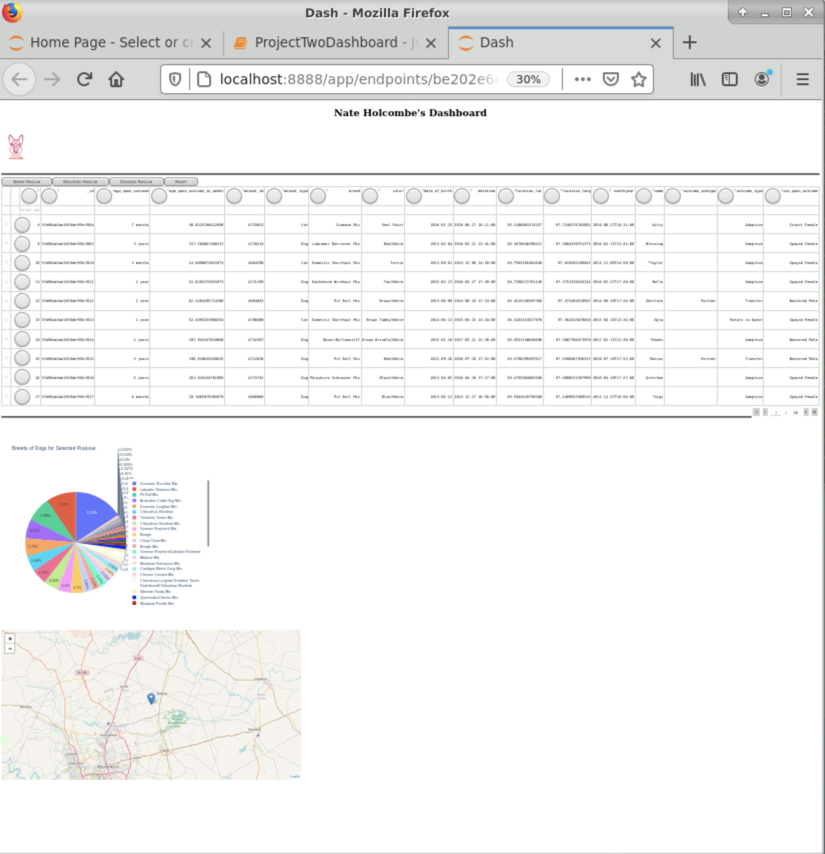
The last conditional statement is shown, followed by the plotly express pie chart. Plotly express provided a highly compatible means by which to integrate such a graph, using the filtered dataframe created in response to the clicked filter button.



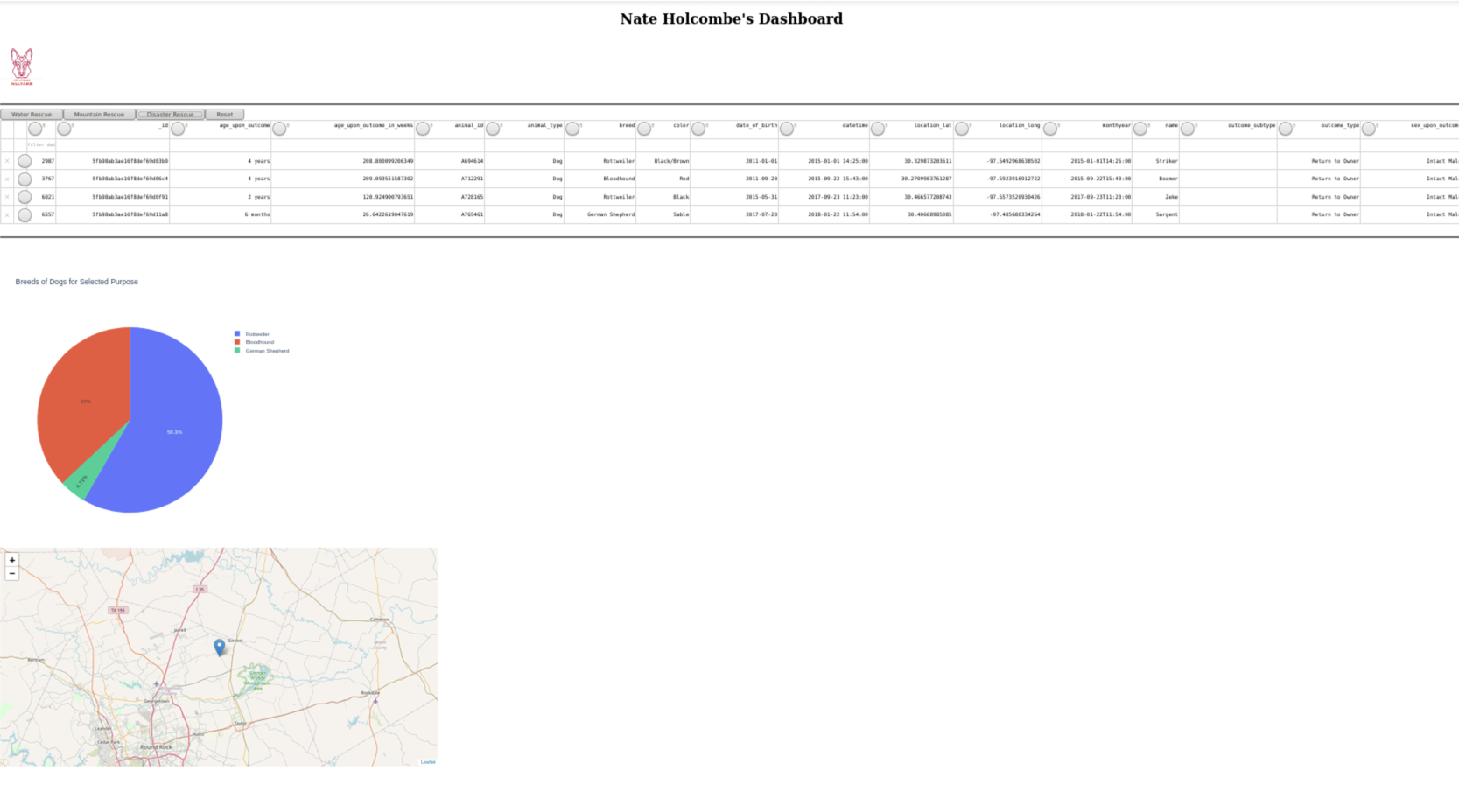
The map callback sends the map in response to the filter button click, with a marker set to the default location just outside of Austin, TX.



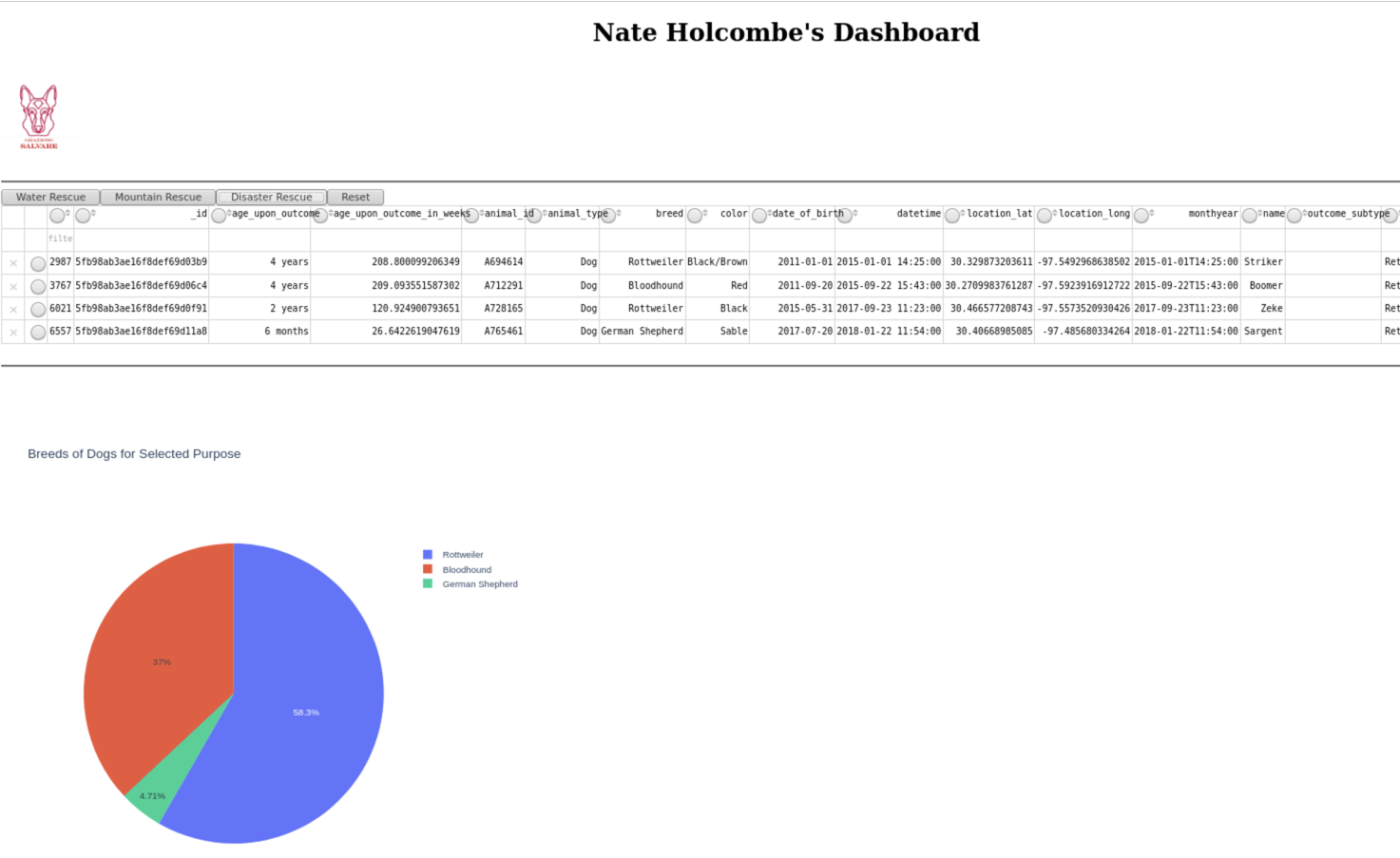
Screenshot of starting state of dashboard:



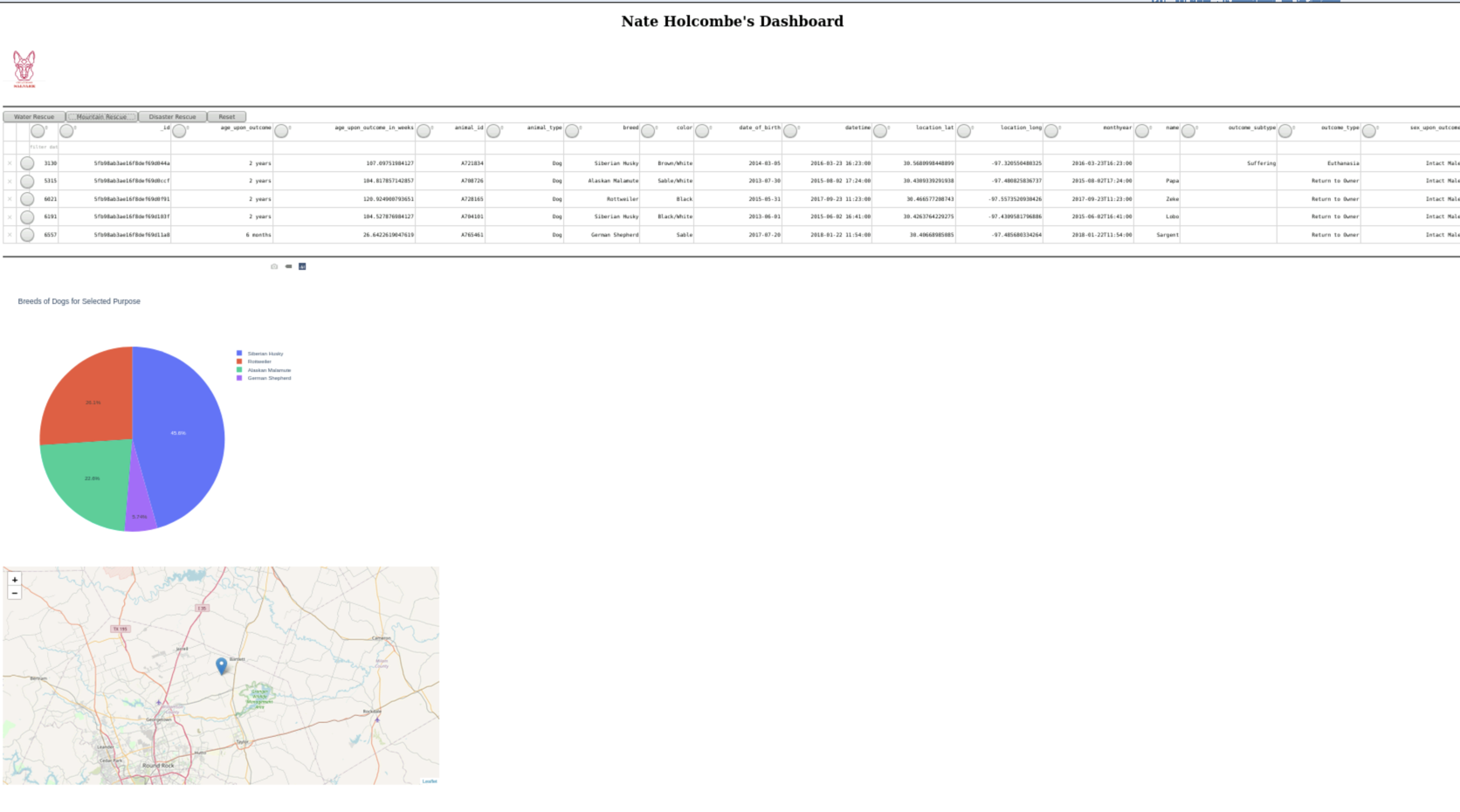
Screenshot of Disaster Rescue button selection:



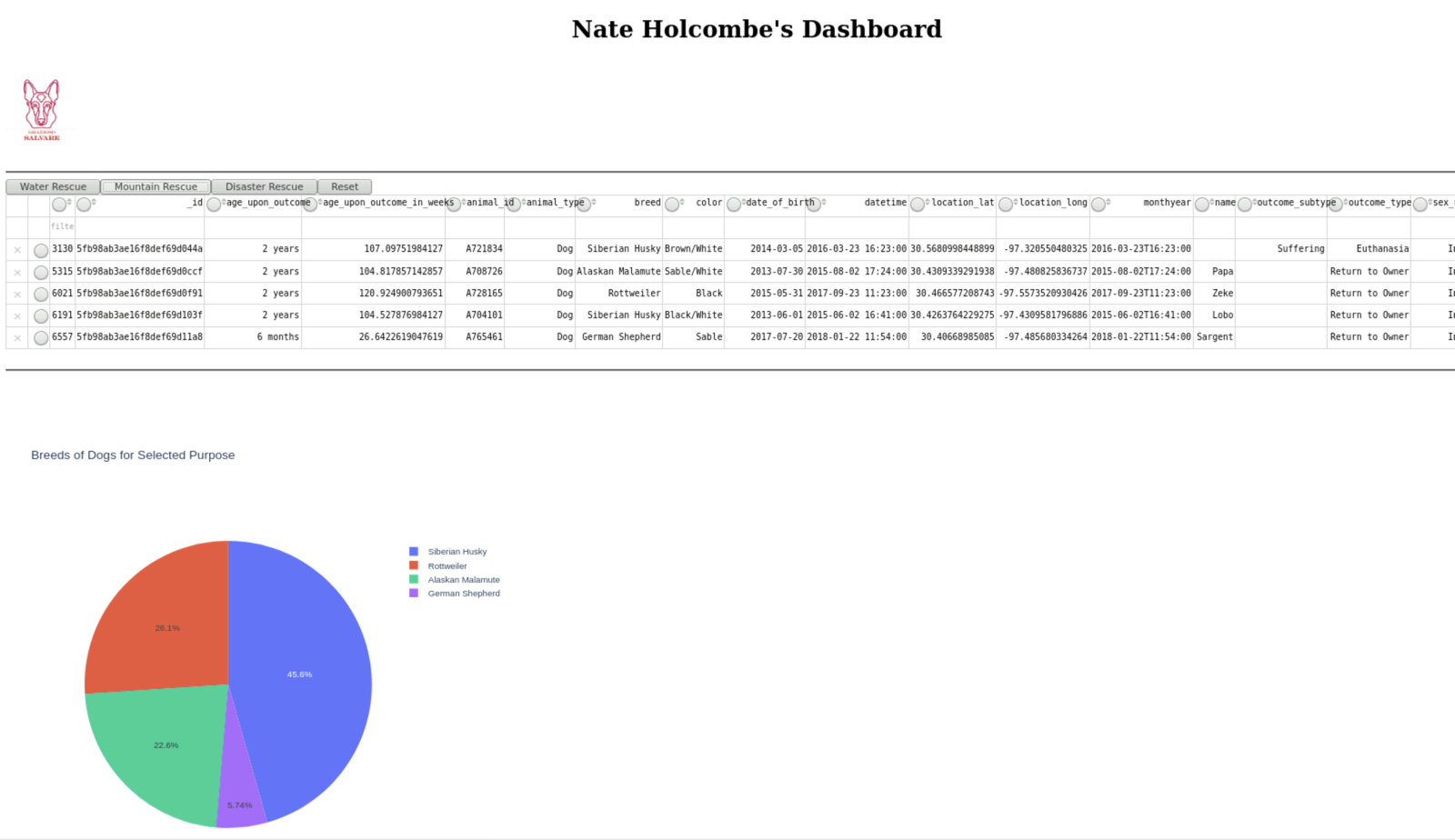
Close up of data and chart:



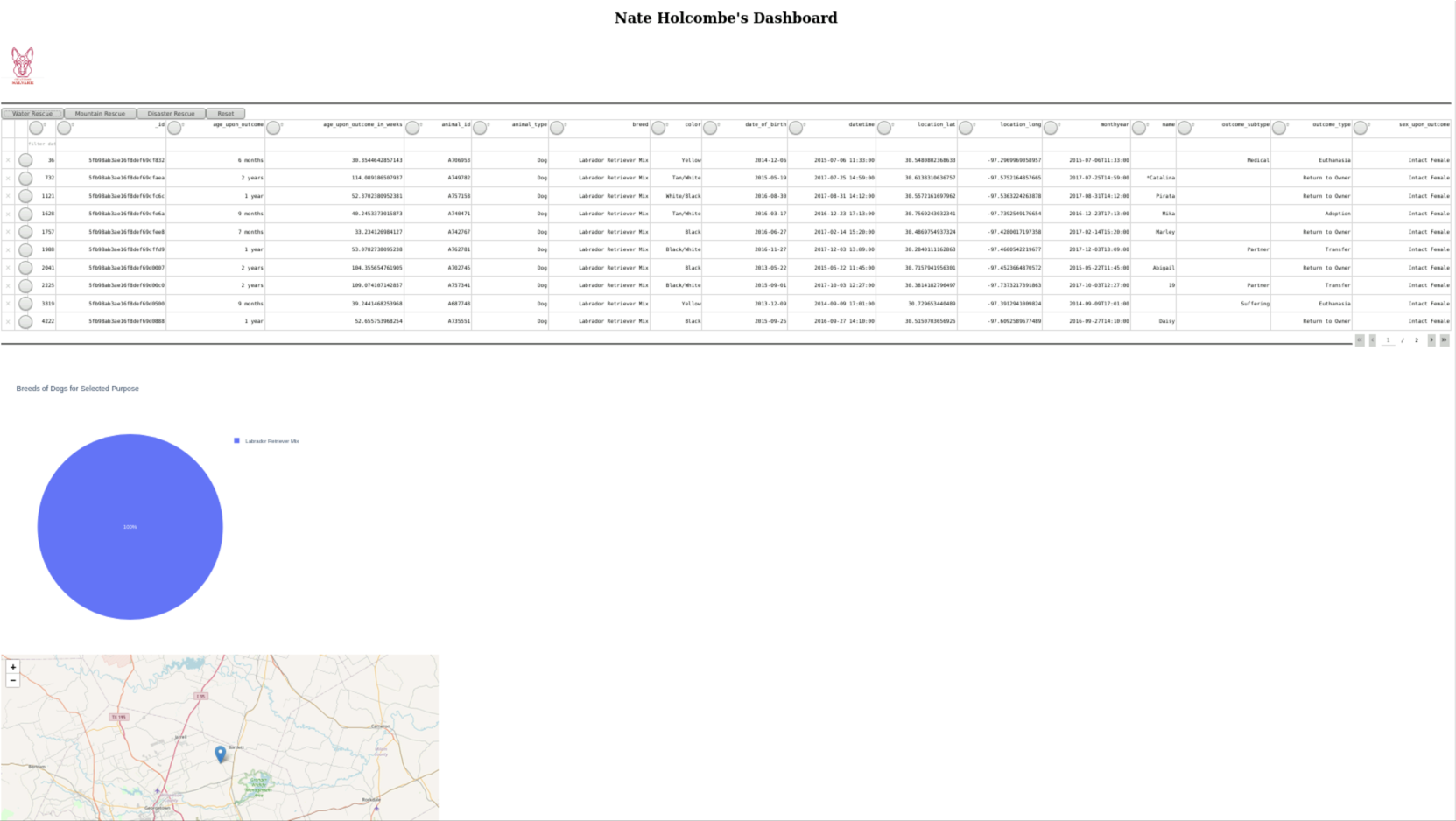
Screenshot of Mountain Rescue button selection:



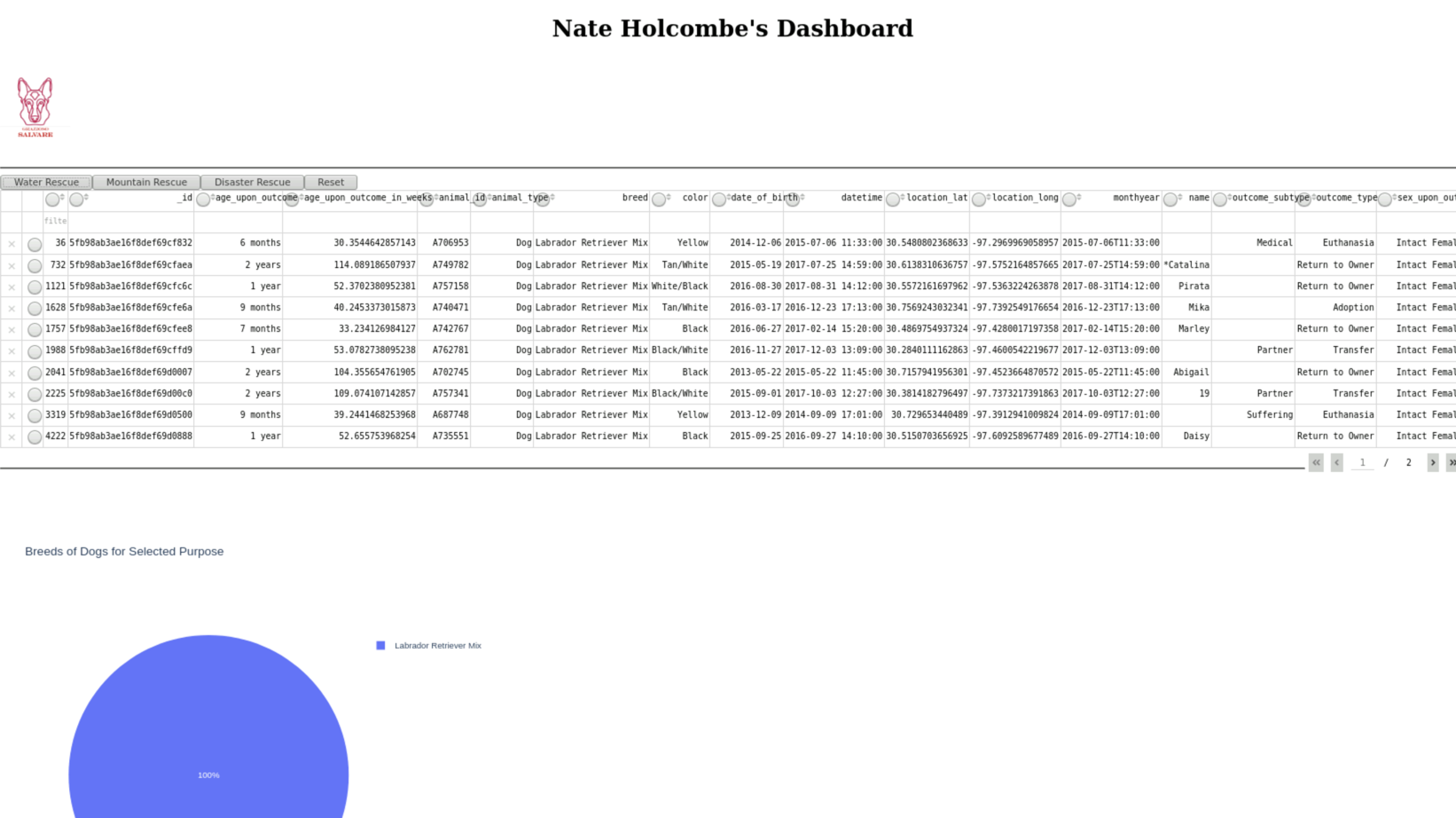
Closeup of data and chart:



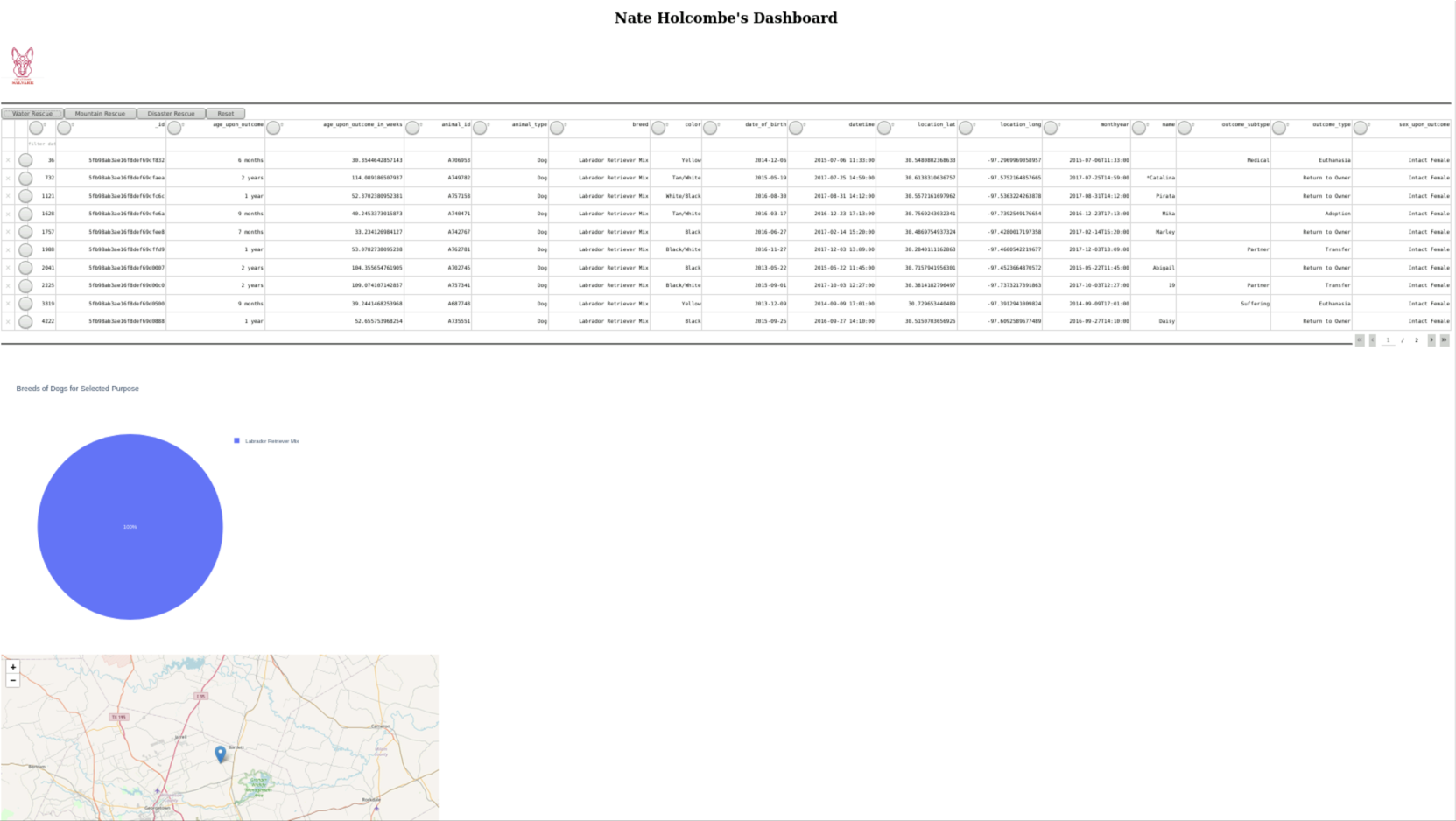
Screenshot of Water Rescue button selection:



Closeup of data and chart:



Screenshot of Reset button selection (does not function):



## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

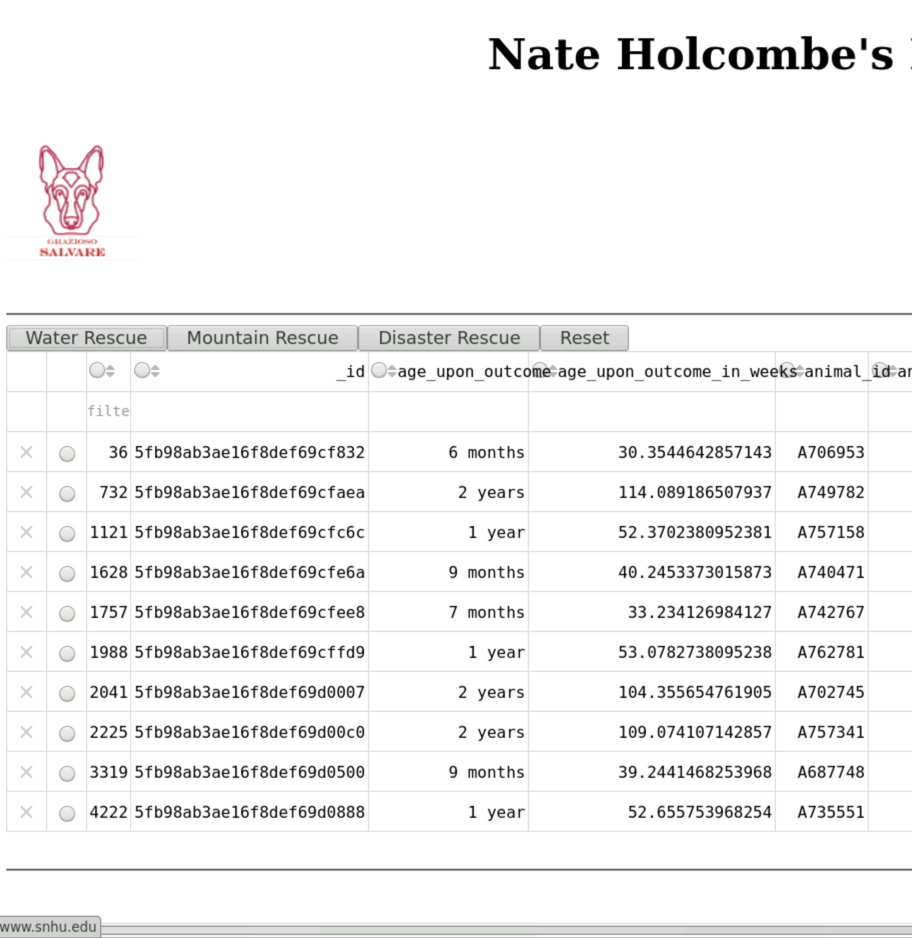
## 

## 

## 

## 

## Screenshot of Logo as link to [www.snhu.edu](http://www.snhu.edu):



**Challenges**

JSON object not iteratable in default dataframe format:

This was a challenge that was overcome by casting the \_id column of the default data to a string.

Map Markers:

The intention was also to display each dogs’ location on the map, but I was not able to produce this functionality after many attempts. I thought that creating a list of dog markers using the iloc() function on the filtered data set would allow iteration of this marker list, each time adding the marker to the map, but the python version of leaflet seems to lack the addTo(map) capabiilties that the Javascript version of leaflet possesses.

Reset Button:

I made an attempt to update the n-clicks count with each button click, so that the buttons not being clicked would be set to n-clicks equal to zero, but this was not permitted since the n-clicks data type was immutable. My attempt to modify the data type to a mutable dict was not successful. Otherwise, the reset button would have simply set all of the n-clicks to zero to display the starting data set.

## Contact

Your name: Nathaniel Holcombe