# Animal Shelter Database Dashboard

This project was created for finding rescue animals quickly and easily. Grazioso Salvare has determined that rescue dogs typically have selection criteria and this project will enable them to find dogs that match this criteria via a web browser.

# Necessary Installs

* [Python 3.6+](https://www.python.org/downloads/)
* Core language being used in the project.
* [Jupyter Notebook Interface](https://docs.jupyter.org/en/stable/install/notebook-classic.html)
* Used for running Python code reliably.
* [MongoDB](https://www.mongodb.com/docs/manual/installation/)
* MongoDB provides the M within the MVC pattern. The `AnimalShelter.py` Python module accesses the database and performs CRUD operations in response to controllers. More on MVC down below.
* PyMongo
* `pip install pymongo`
* MongoDB interfaces particularly well with Python through the use of the PyMongo Python Module. Interfacing using PyMongo is strikingly similar to interfacing using Mongosh, the terminal program alternative. This consistency makes it incredibly simple to test queries before putting them in the application layer by using the terminal to test them instead, bypassing the browser and annoying load times.
* Dash
* `pip install dash`
* Dash provides a framework that enables developers to provide the VC within the MVC pattern. Put simply, views are implemented through Dash components and controllers are implemented through callback functions that respond to changes within the view asynchronously. More on MVC down below.
* Pandas
* `pip install pandas`
* Pandas is used for creating a `DataFrame` from the data retrieved from the 'AnimalShelter.py' Python module.

## MVC

* This pattern works well with RESTful APIs which is applied within the `AnimalShelter.py` Python module.
* A short description of MVC is as follows:
* Model: defines the data & updates the view.
* View: defines the user interface & sends user input to the controller.
* Controller: defines control logic & manipulates the model according to the user input recieved from 'View'.

# Getting Started

* Clone this repository and then import the data for MongoDB:
* This data comes from The Austin Animal Center located in Austin, Texas: [DOI](https://doi.org/10.26000/025.000001)
* Note that the environment variables: `MONGO\_USER`, `MONGO\_PASS`, `MONGO\_PORT`, and `MONGO\_HOST` must be set accordingly before running this command as-is.

git clone https://github.com/nlwhite/CS340.git

cd CS340

mongoimport --username="${MONGO\_USER}" --password="${MONGO\_PASS}" \

--port=${MONGO\_PORT} --host=${MONGO\_HOST} \

--db=AAC --collection=animals --authenticationDatabase=admin \

--file=./aac\_shelter\_outcomes.csv --type=csv --headerline

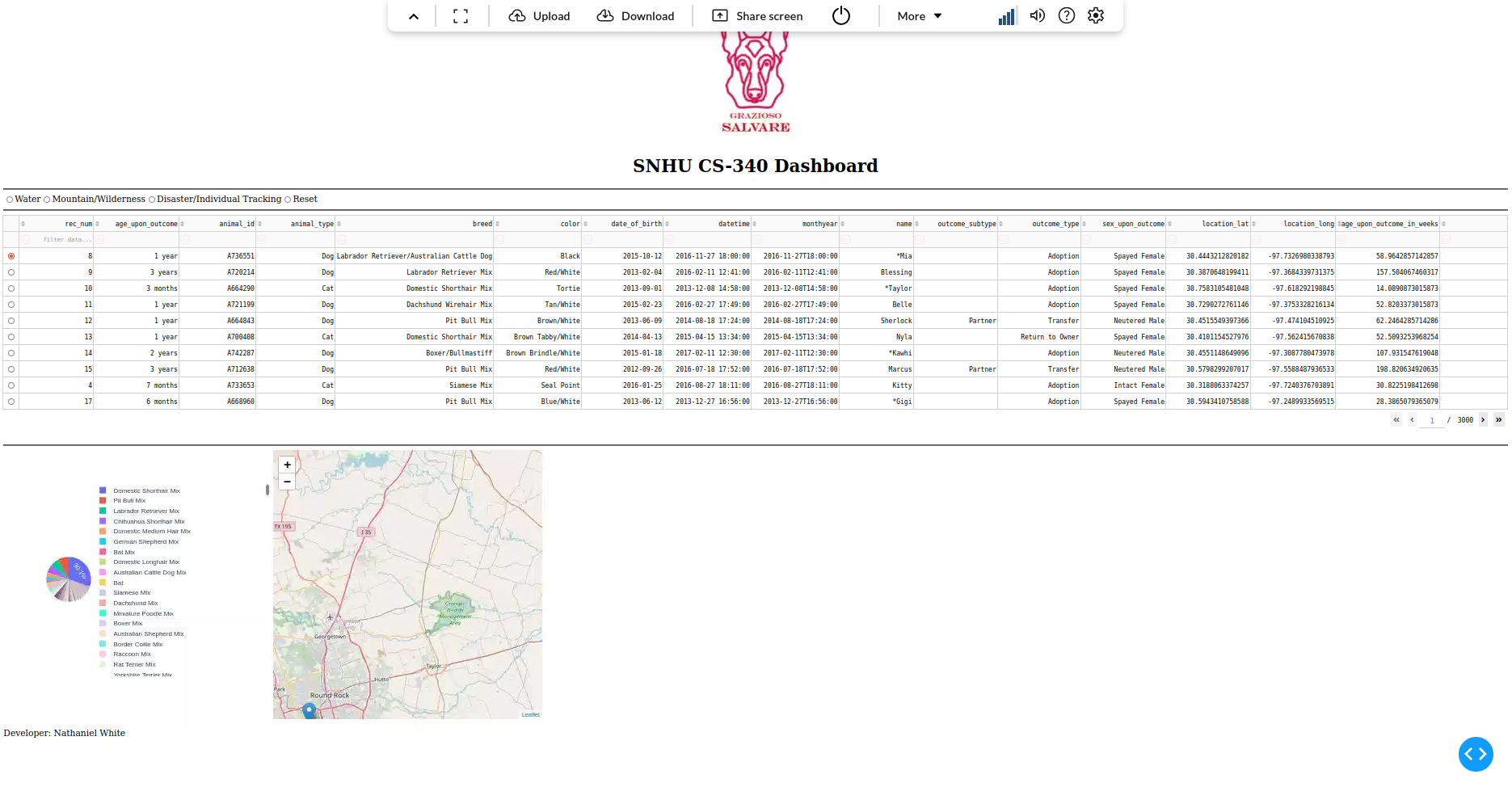
* To start the application, open the Jupyter Notebook and click 'Run'.
* If the app started successfully, the exact http address is printed near the bottom.
* This url may be accessed by any computer logged into the same wifi as the computer that ran the application. Please ensure the wifi is secure and cannot be accessed by unknown users.
* By default, the application is launched on port 33229. If this port is already in use, either change the port (line 25 of the Jupyter Notebook) or end the process currently running on that port.

# Usage

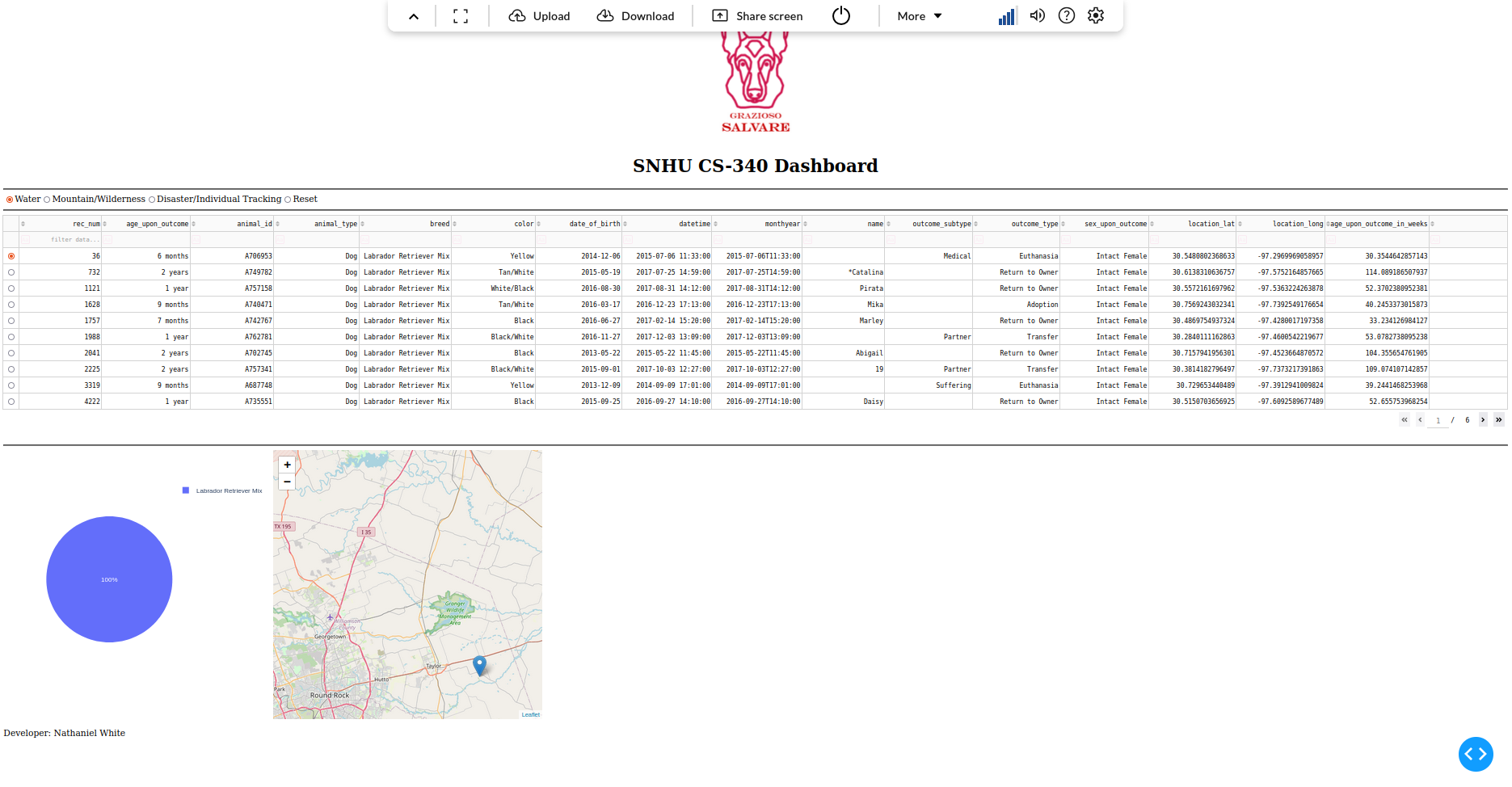
* When first loading the page, the user is greeted with:
* a datatable that holds 1000 records,
* a pie-chart with sectors of animal breeds,
* and an interactive map that indicates where the first animal in the database is located.
* The user has three radio buttons to their disposal for filtering the animal data. These are the types of rescue animals as profiled by Grazioso Salvare. The exact filtering criterion is outlined as follows:
* Water
* Breeds: Labrador Retriever Mix, Chesapeake Bay Retriever, Newfoundland
* Sex: Intact Female
* Age: 26-156 weeks
* Mountain/Wilderness
* Breeds: German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, Rottweiler
* Sex: Intact Male
* Age: 26-156 weeks
* Disaster/Individual Tracking
* Breeds: Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler
* Sex: Intact Male
* Age: 20-300 weeks
* Lastly, there is one last button for removing the filters named, 'Reset'

## Screenshots

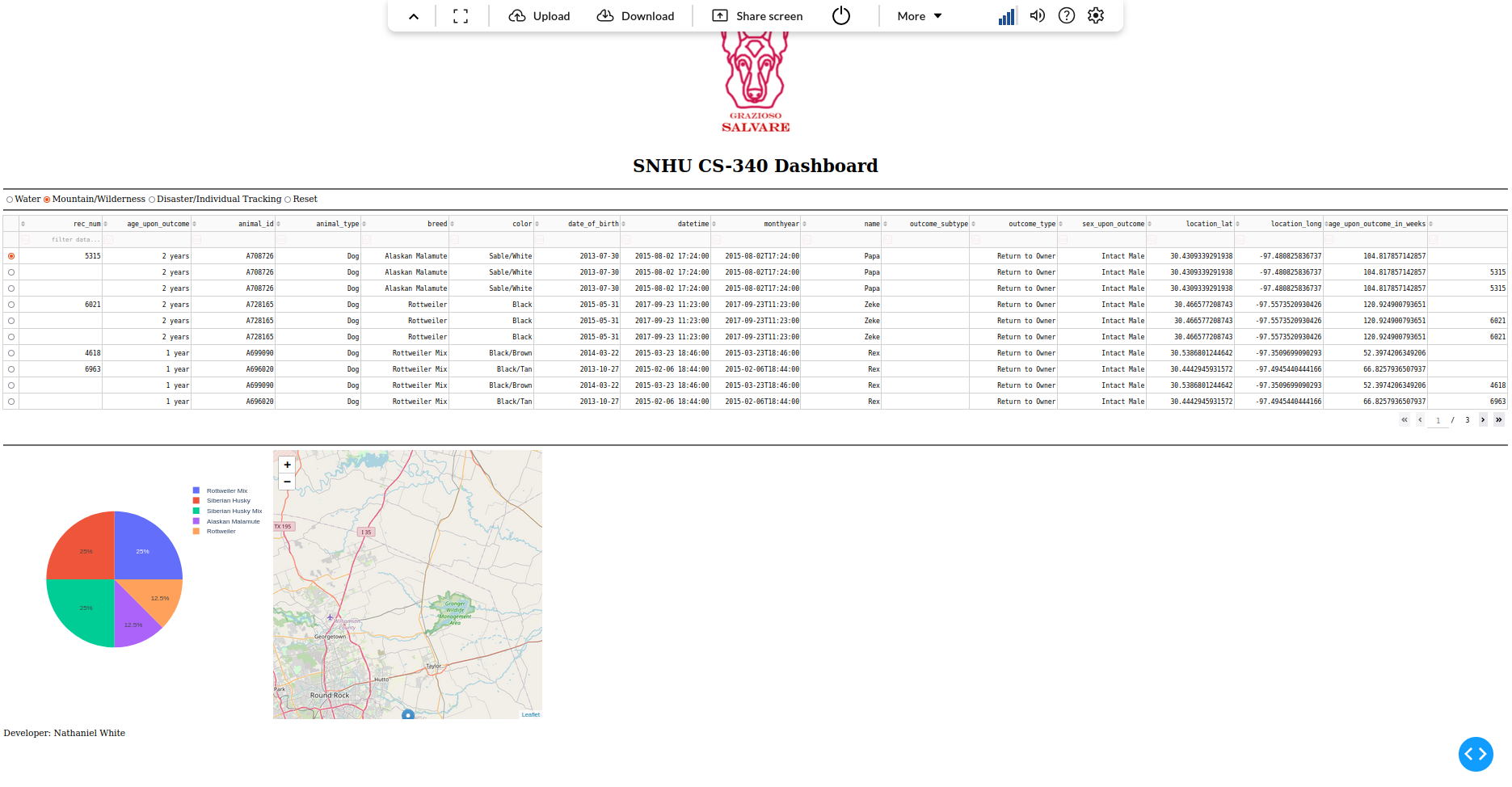
Starting state:



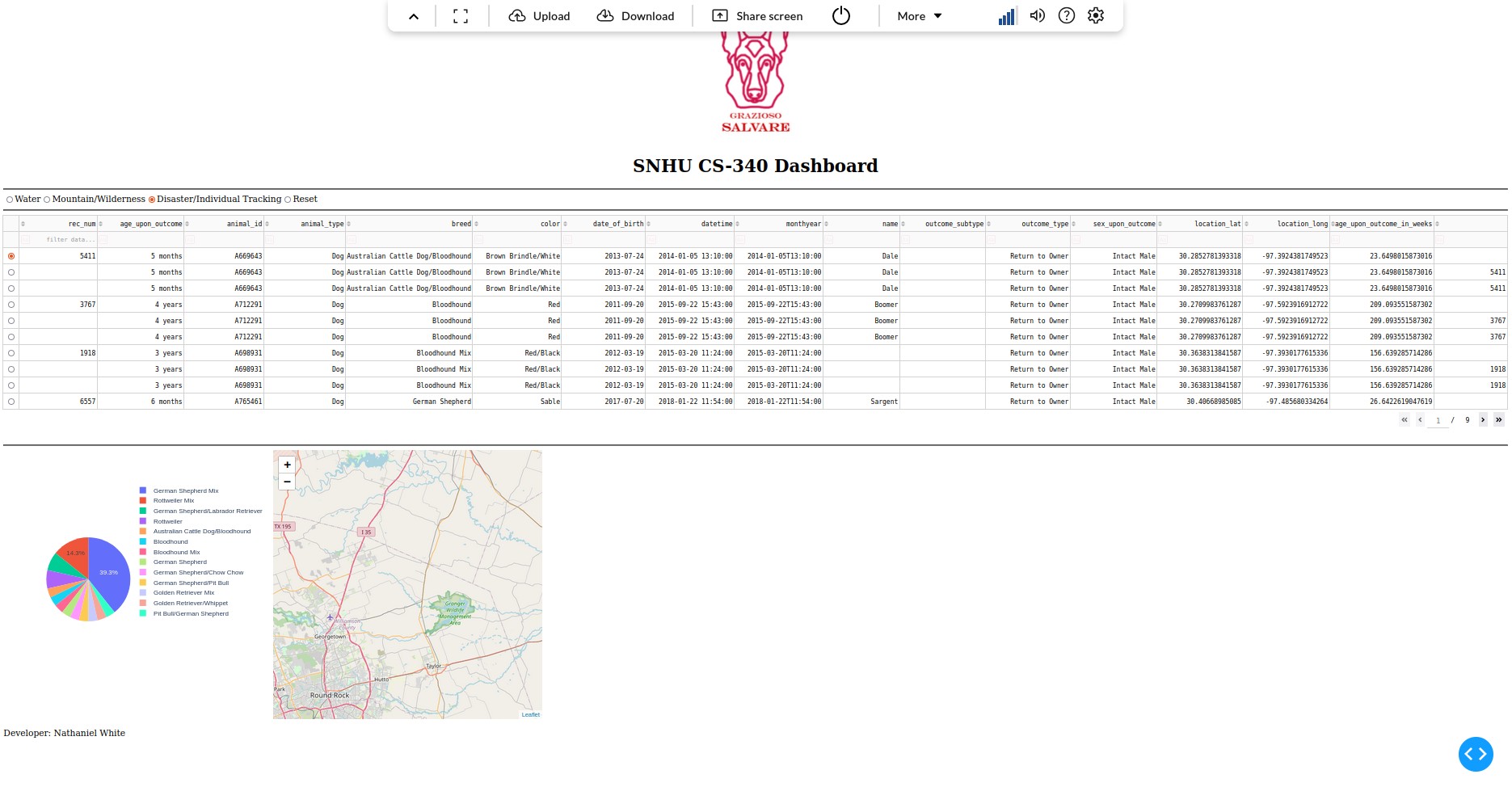
After clicking the 'Water' radio button:



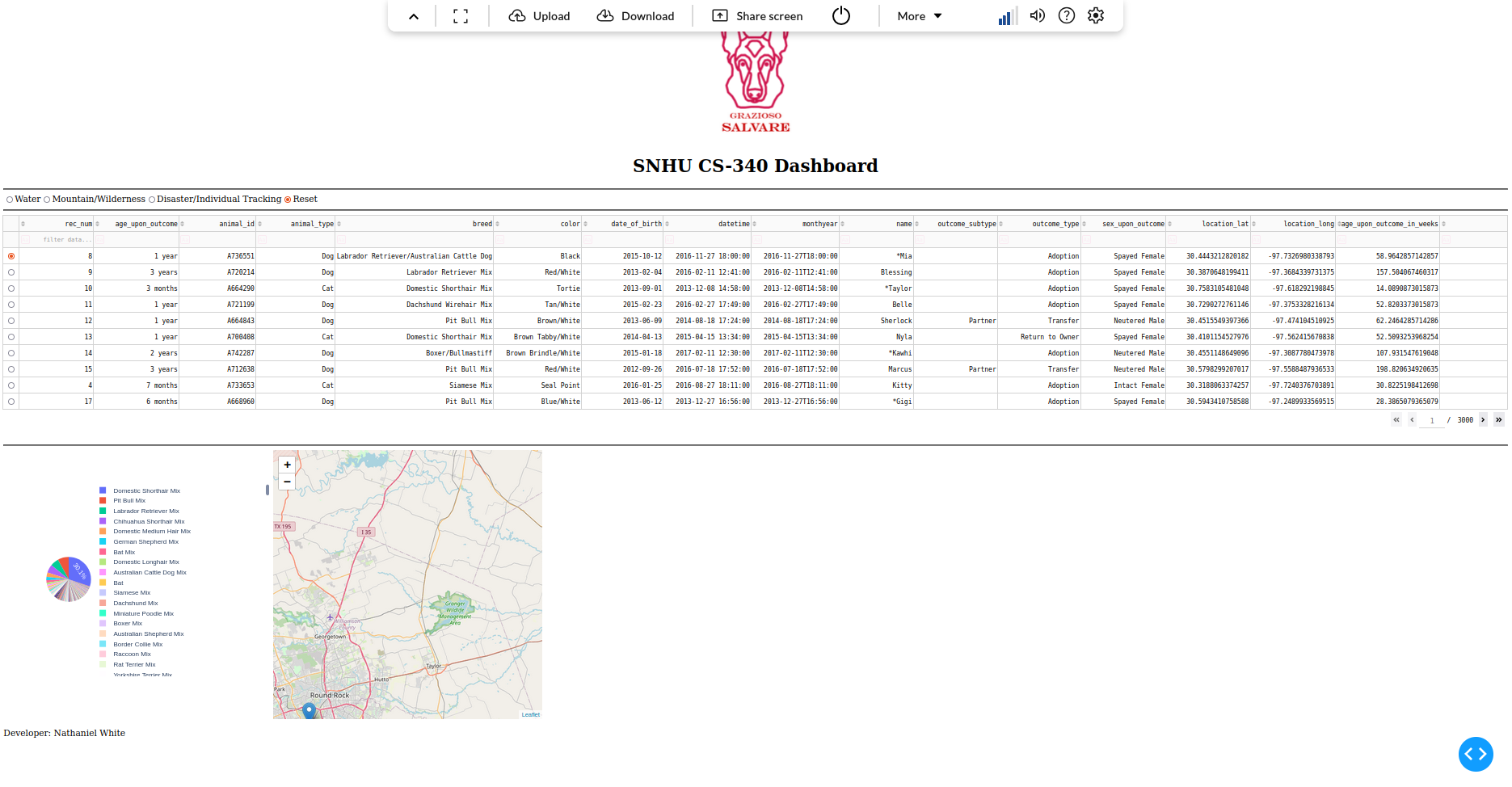
After clicking the 'Mountain/Wilderness' radio button:



After clicking the 'Disaster/Individual Tracking' radio button:



After clicking the 'Reset' radio button:



# Steps Taken

1. Imported the data from the `aac\_shelter\_outcomes.csv` file into MongoDB.
2. Created a user that can authenticate with the database.
3. Created a CRUD Python module that accesses the database as this user.

* Username, password, host, port, database name, and collection name are all decided by the user of the module to ensure portability.
* `AnimalShelter.create(data: dict) -> bool` inserts an animal document into the animals collection. Returns `True` if successful insertion, `False` otherwise.
* `AnimalShelter.read(query: dict) -> list` returns a list of animals that fit the query. Note that PyMongo specific syntax should be passed as strings: i.e. `{"$gte": 10}` instead of `{$gte: 10}`.
* `AnimalShelter.update(query: dict, update: dict) -> int` updates all animal documents specified by `query` with the data specified by `update`. Returns the number of updated animals.
* `AnimalShelter.delete(query: dict) -> int` deletes all animals specified by `query`. Returns the number of deleted animals.

1. Created a dashboard that visualizes the data and provides filter functionality for the user by calling methods from the previously mentioned CRUD Python module.

* Radio buttons for filtering data
* Datatable
* Responds to radio button filters.
* May have a single row selected at a time, but columns are not selectable.
* Pie Chart
* Responds to radio button filters.
* Interactive Map
* Responds to Datatable row selection.

# Challenges

* For the radio buttons, it was unclear which path was best for filtering the data. I had to decide between querying the database using the CRUD Python module I wrote, or filtering the Pandas dataframe. The latter option would be filtering 'old' data while the former would fetch the data again. I decided to go with the former since it wouldn't be ideal to reboot the entire application when looking for a new entry in the database. Although, since the data is stored in a local CSV file rather than an external server, it's unlikely that the data will have changed without the application administrator's knowledge.