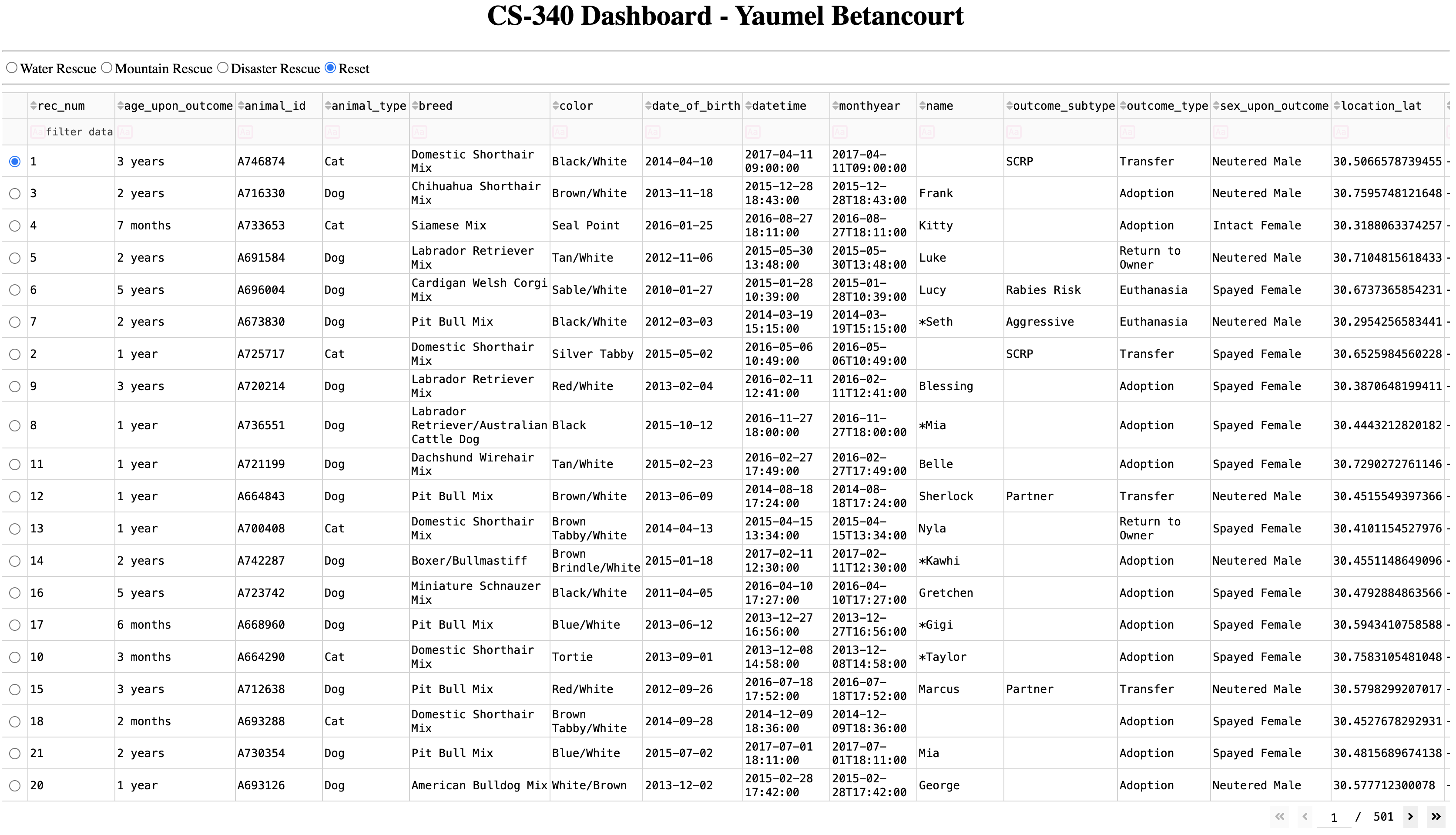
# CS 340 Grazioso Salvare Animal Shelter Project README

## About the Project

This project hosts data from five animal shelters in Austin that can be used to search for animals using different criteria such as species, breed, name, age, etc. An authorized user simply enters their credentials to have access to the database containing the details of all the animals registered in these shelters. This project offers different filters for more optimal searches. The first filtering mechanism uses radio buttons above the table that filter out by specializations, i.e., water rescue, mountain rescue, and disaster rescue. There is also a reset option that just returns all. Each column can be further filtered to limit the results, e.g., breed, color, etc. A pie chart is displayed that gives a breakdown of the five top breeds based on the filters used. A map is also shown to give users a geolocation of where a specific animal can be found.

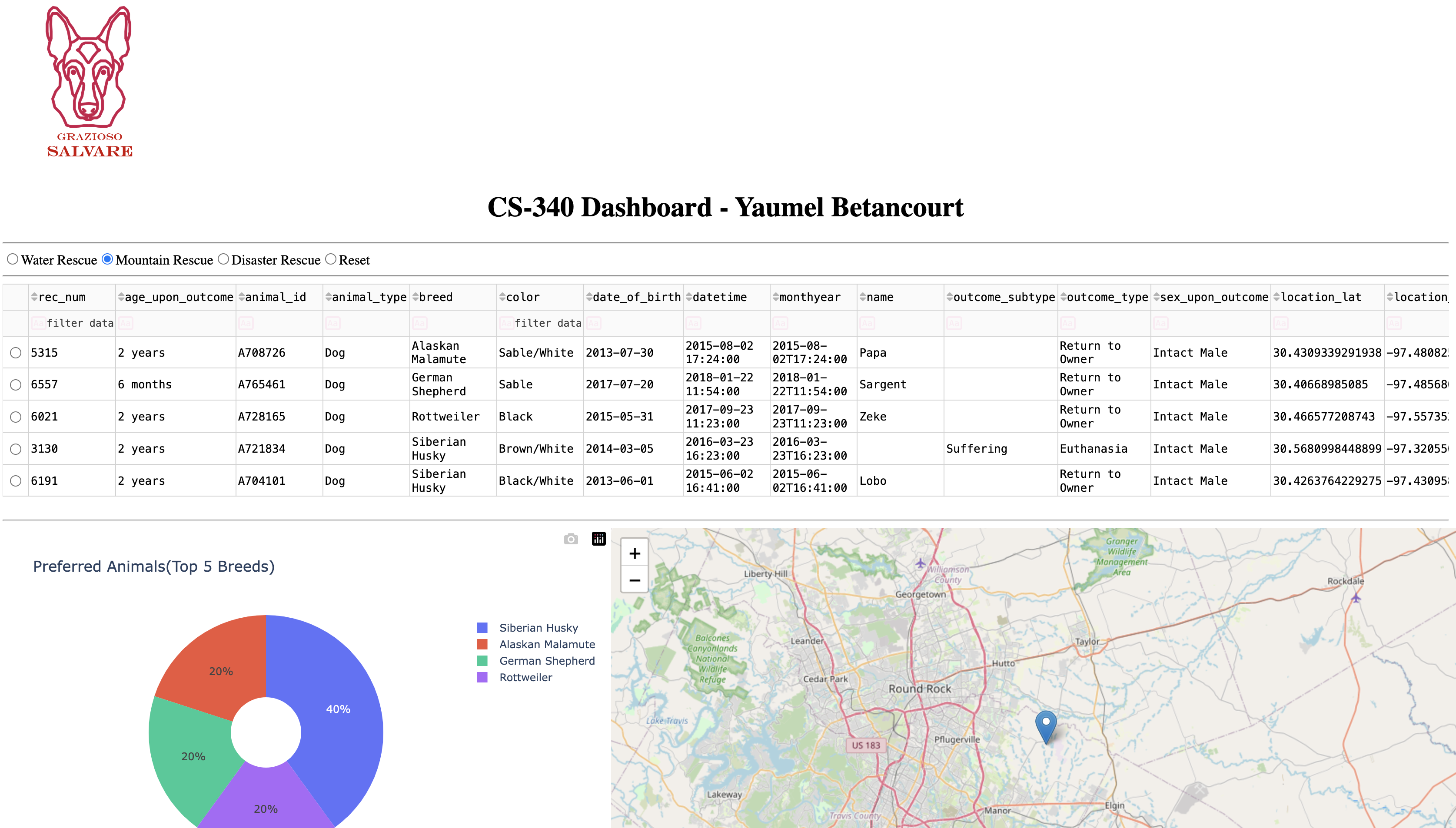
Example of all searches being returned:



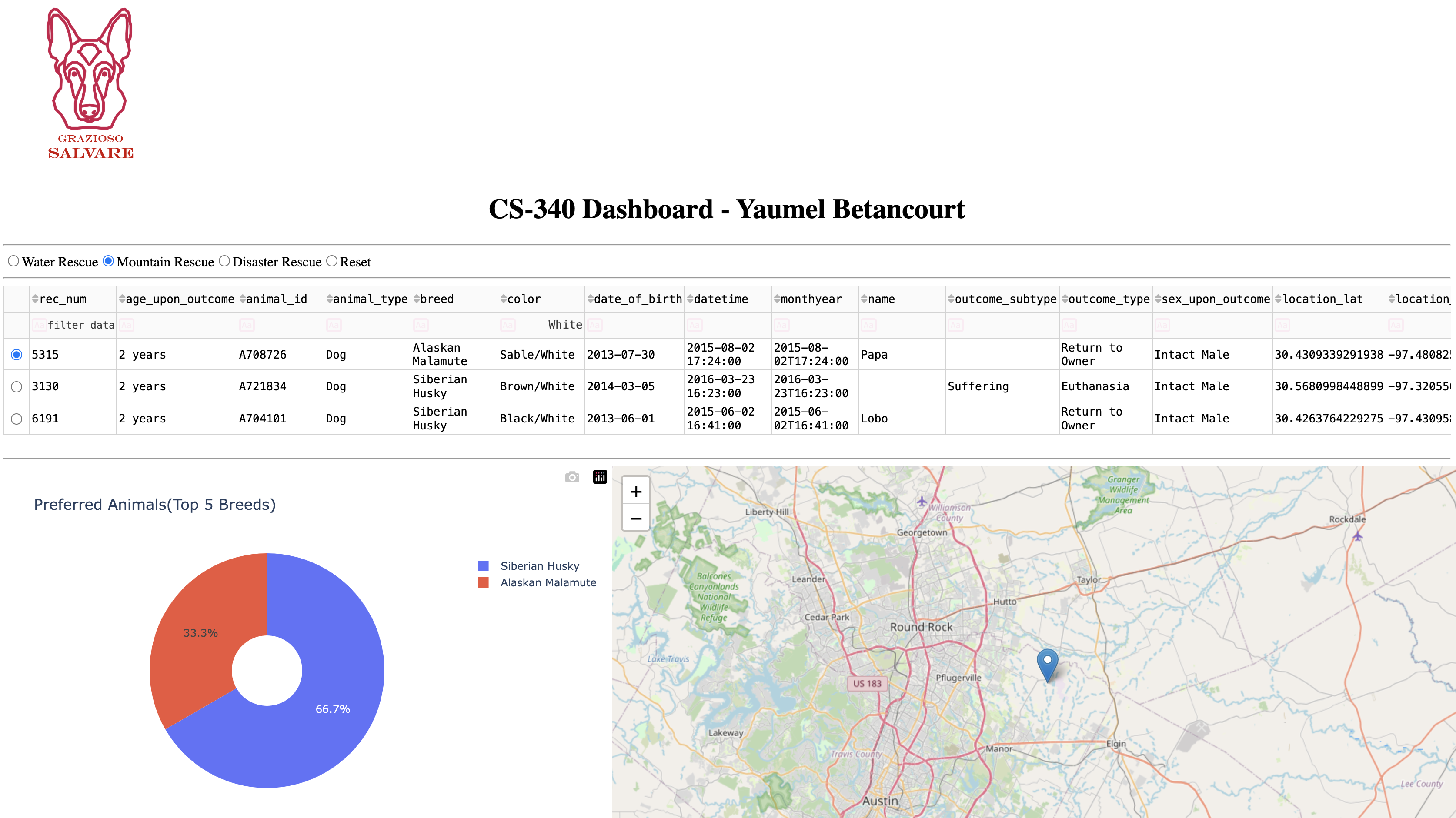
Filtering by color:



Filtering by mountain rescue:



Filtering by mountain rescue and white color:



## Motivation

The goal behind this project is to provide a way of searching these animal shelters for different purposes in a user-friendly fashion. For example, one user may be seeking to adopt a new pet for their family while another user may be seeking a breed that can be trained for special needs, such as search and rescue dogs.

**Tools used**

Several tools were used to create this app. The first and most important tool was MongoDB. Several factors were considered when choosing this NoSQL approach. The first one was data inconsistency. Since the data available was not uniform, a schema-less approach was preferable. MongoDB offers this flexibility as well as providing an efficient option for querying data. Another benefit is that MongoDB scales well horizontally, which is great should the database grow exponentially in the long run.

Python 3 was used as the programming language since it is robust and integrates seamlessly with MongoDB via the Mongo driver.

Lastly, the Dash framework was used to create reactive components that update the view when the state has changed via user input (search filter options). Dash provides an easy way to create these reactive components and feed the updated data returned from the model via a controller

Sources:

MongoDB: <https://www.mongodb.com/docs/manual/>

Python: <https://docs.python.org/3/>

Dash: <https://dash.plotly.com/#dash-fundamentals>

## Creating the application

The first step was to update the view with the model when no filters were applied to make sure the controller was feeding data from the model to the view correctly. Once that was working, the next step was to make sure the filters for the individual columns were working correctly. Once the default search was working correctly (what would become the reset option), radio buttons were created for the specialization filters. Each radio button was attached to a specific search filter (e.g., water rescue). Once all the search filters were working properly, the pie chart was implemented to display the top five breeds for each filter. The pie chart updates to reflect all filters applied to the search. Finally, a map was implemented to display the geolocation of a selected entry, making it easier for a user to locate. Finally, small details were finished, including the addition of the logo and pagination to make the result more user-friendly.

**Challenges**

The main challenge was making sure the data from the model was being displayed correctly on the view. The most difficult part was the map widget. There were many out of bound errors when changing the search results after applying different filters. This was resolved by checking the number of elements after a filter had been applied to make sure that an invalid index was not being used.

## Contact

Yaumel Betancourt (yaumel.betancourt@snhu.edu)