# CS 340 README

## About the Project/Project Title

The Grazioso Salvare Animal Dashboard was created to facilitate the adoption of reliable search-and-rescue animals. The dashboard queries information provided by local animal shelters and returns the ideal candidates for Grazioso Salvare’s training program. This is done using a local MongoDB server that is accessed using a Python CRUD Module. Plotly’s Dash library utilized this data to create intuitive and interactive data charts that facilitate user understanding.

## Motivation

The motivation behind this program is a desire to learn how to interact with MongoDB and to use the data accessed to create a vibrant, interactive dashboard. Python was chosen as there is robust support for interactions with MongoDB through PyMongo. Dash was picked as it allowed me to create a dashboard that functions using Jupyter Notebook. The convenience and ease-of-use made my choice in tools an easy decision.

## Getting Started

First, you want to start by importing your dataset into MongoDB. After this, set up authentication controls for both an administrator account and a user account. Next, you will need to create a CRUD module of your own in Python. To test your work, start up your MongoDB and use Jupyter Notebook to create a script to test your Python code. Once your CRUD module is created and tested, use Plotly Dash to create a dashboard that communicates with the CRUD module. Be sure to include a data table that contains complex queries and a few graphs.

## Installation

The tools needed for this project are a Linux distribution, MongoDB, Python, Anaconda, Jupyter Notebook, Dash, and a dataset of your choice.

* Linux - <https://distrowatch.com/>
* MongoDB - <https://www.mongodb.com/docs/manual/installation/>
* Python - <https://www.python.org/downloads/>
* Anaconda - https://www.anaconda.com/products/distribution
* Jupyter Notebook - <https://docs.jupyter.org/en/latest/install.html>
* Dash - https://dash.plotly.com/dash-core-components/download
* Datasets - https://catalog.data.gov/dataset?res\_format=JSON

## Usage

### Code Example

@app.callback(

    Output("graph", "figure"),

    [Input('dropdown-id', 'value')]

)

*def* update\_graph(*selection*):

    # initial condition and condition when selection is cleared

    if *selection* is None:

        df = pd.DataFrame.from\_records(shelter.readAll({}))

    # if the user selects Water Rescue from the dropdown

    if *selection* == 'Water Rescue':

        searchQuery = {"breed": { "$in" : ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]}, "sex\_upon\_outcome" : "Intact Female", "age\_upon\_outcome\_in\_weeks" : { "$lt" : 156, "$gt" : 26 }}

        df = pd.DataFrame(list(shelter.readAll(searchQuery)))

    # if the user selects Mountain/Wilderness Rescue from the dropdown

    if *selection* == 'Mountain/Wilderness Rescue':

        searchQuery = {"breed": { "$in" : ["German Shepherd", "Alaskan Malamute", "Old English Sheepdog",  "Siberian Husky", "Rottweiler"]}, "sex\_upon\_outcome" : "Intact Male", "age\_upon\_outcome\_in\_weeks" : { "$lt" : 156, "$gt" : 26 }}

        df = pd.DataFrame(list(shelter.readAll(searchQuery)))

    # if the user selects Disaster/Individual Tracking from the dropdown

    if *selection* == 'Disaster/Individual Tracking':

        searchQuery = {"breed": { "$in" : ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"]}, "sex\_upon\_outcome" : "Intact Male", "age\_upon\_outcome\_in\_weeks" : { "$lt" : 300, "$gt" : 20 }}

        df = pd.DataFrame(list(shelter.readAll(searchQuery)))

    fig = px.pie(df, *names*= 'breed')

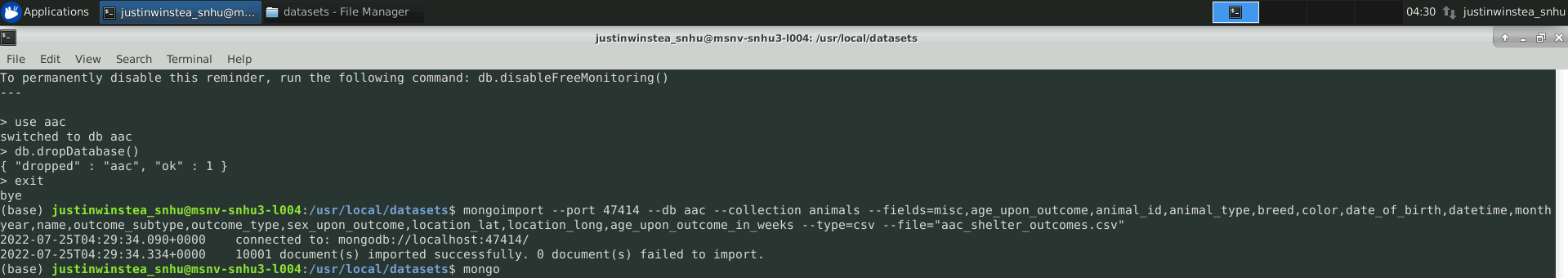
    fig.update\_traces(*textposition*='inside')

    return fig

### Tests

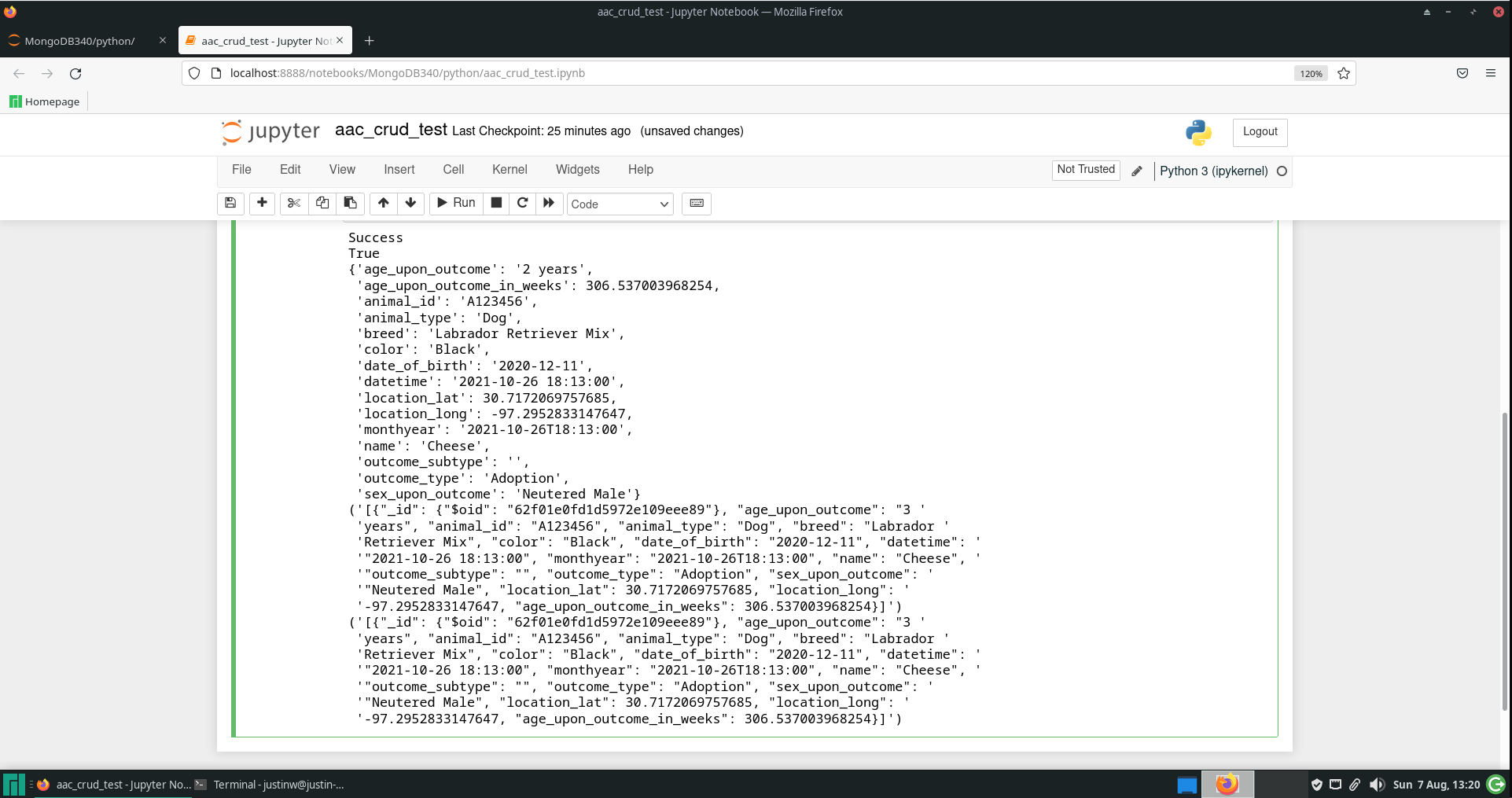
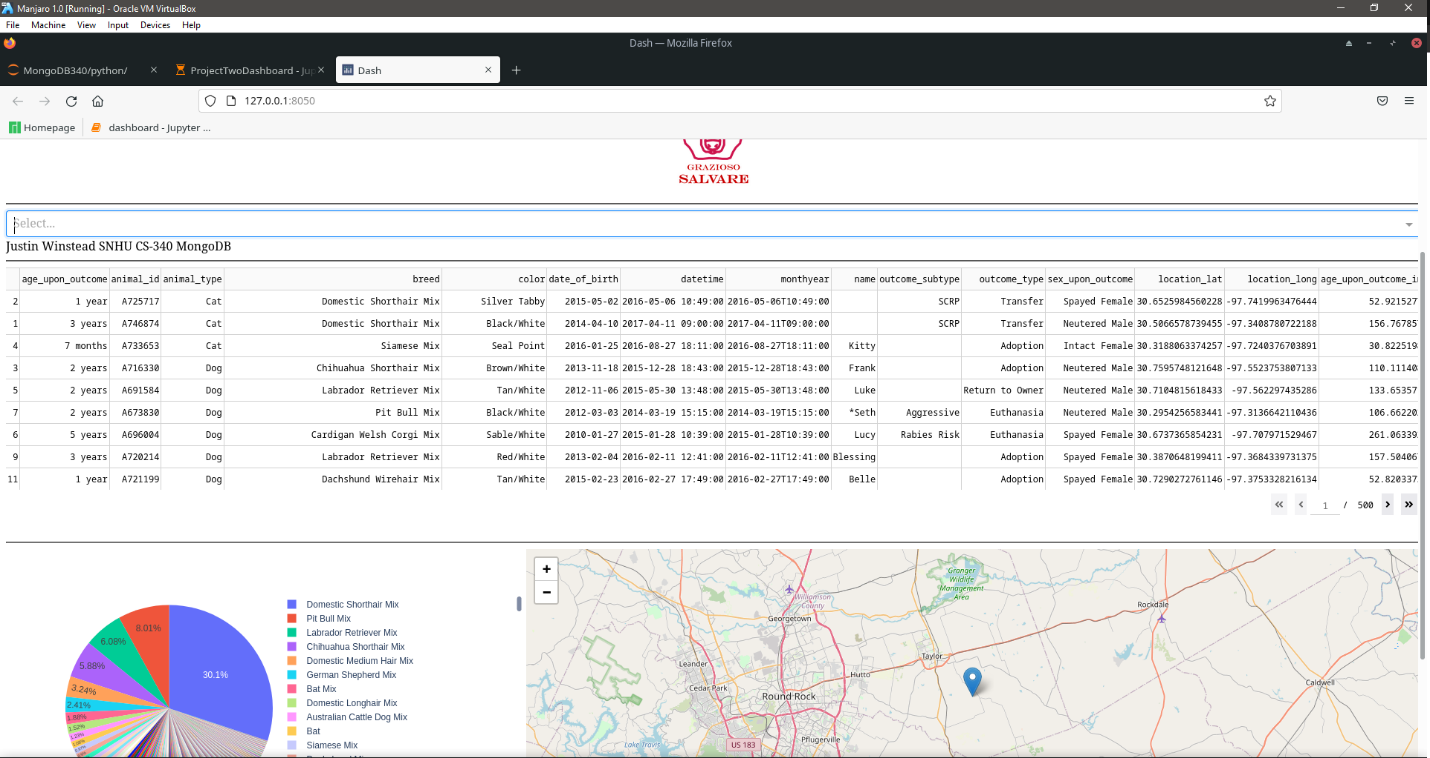
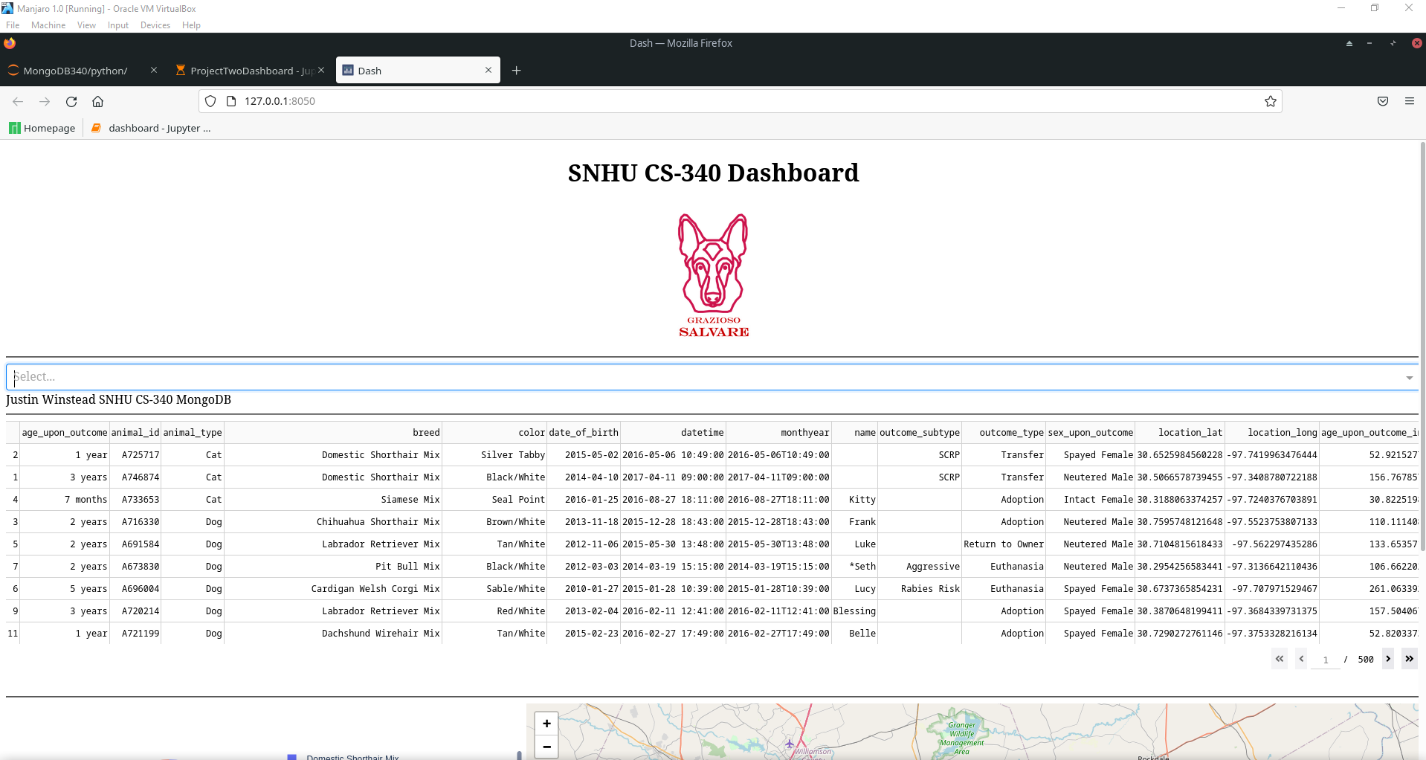
### The tests that I created are a bit simplistic. However, they demonstrate the functionality of my CRUD module effectively. I call the create function to upload the provided JSON value to the database. Then the read function is called to ensure that the create function is working properly. Another function I included, update, searches the database for a document using its first argument and updates some fields in each document found using the second. The delete function deletes all documents that match a given search parameter. In addition to these tests I created, I included values that would indicate that a process was completed successfully like using print to display ‘True’ to the console. In addition to the tests I created, I developed a dashboard that allowed the user to filter through three complex queries. I tested this interaction by comparing the results between my dashboard and the MongoDB shell. Using this method, I determined the code I developed filtered the user’s requests properly and updated the graphical aids to reflect this change.

### Screenshots

****Text

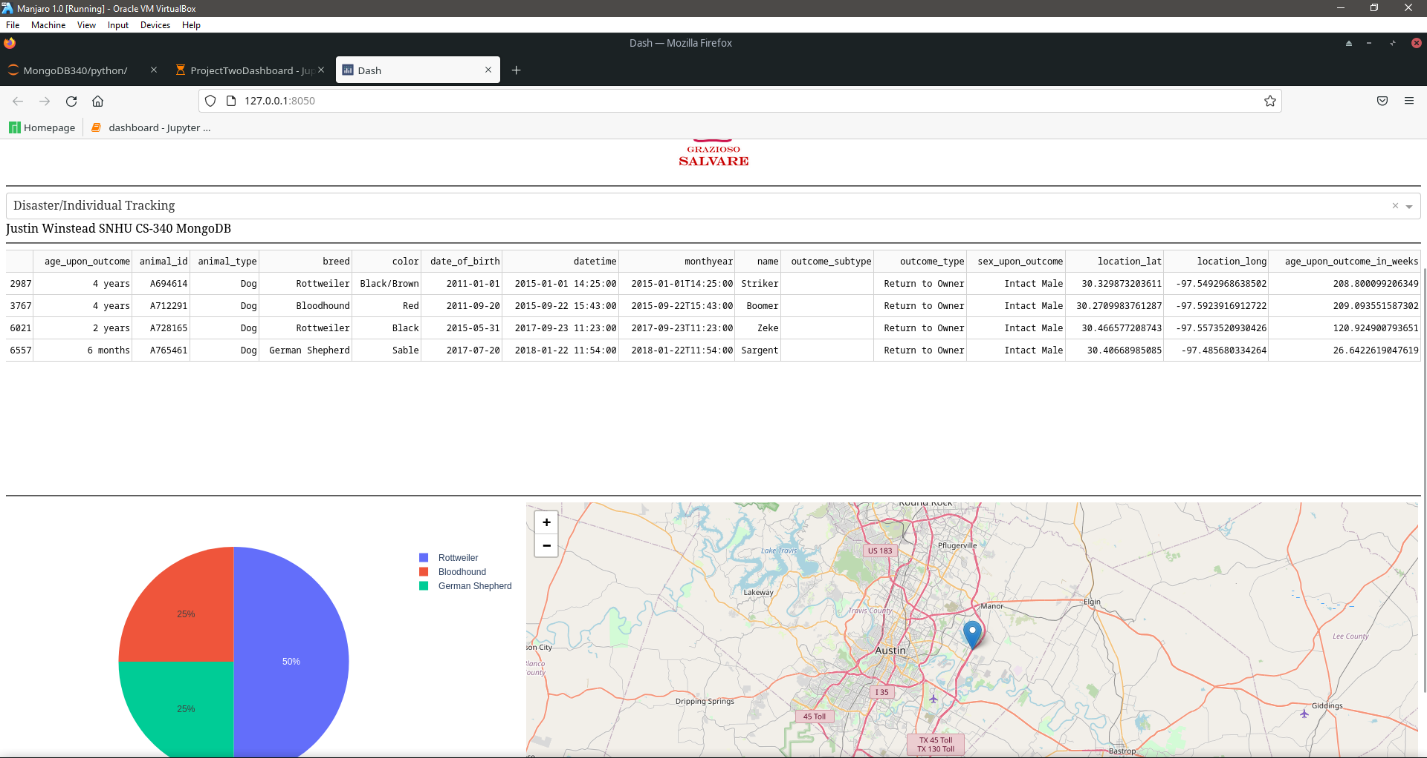
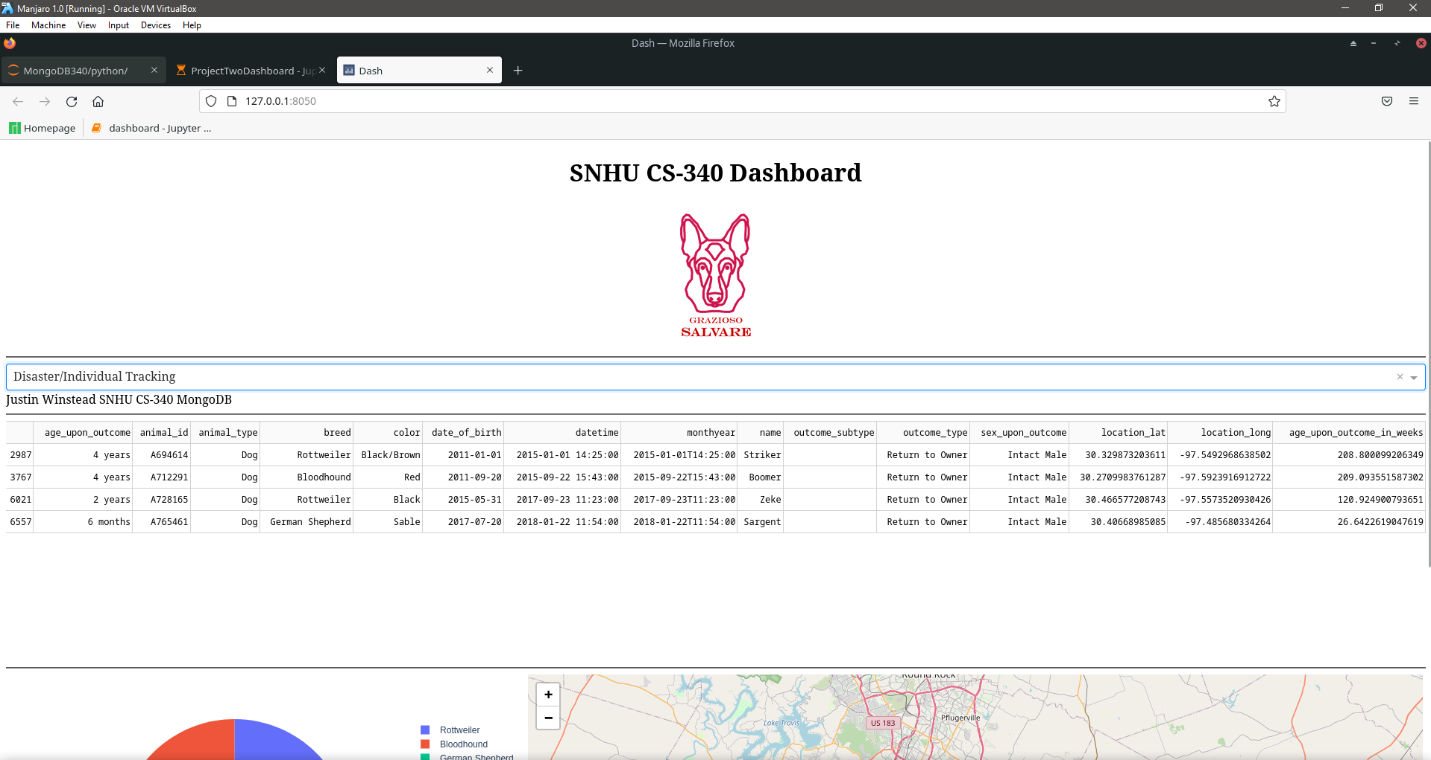
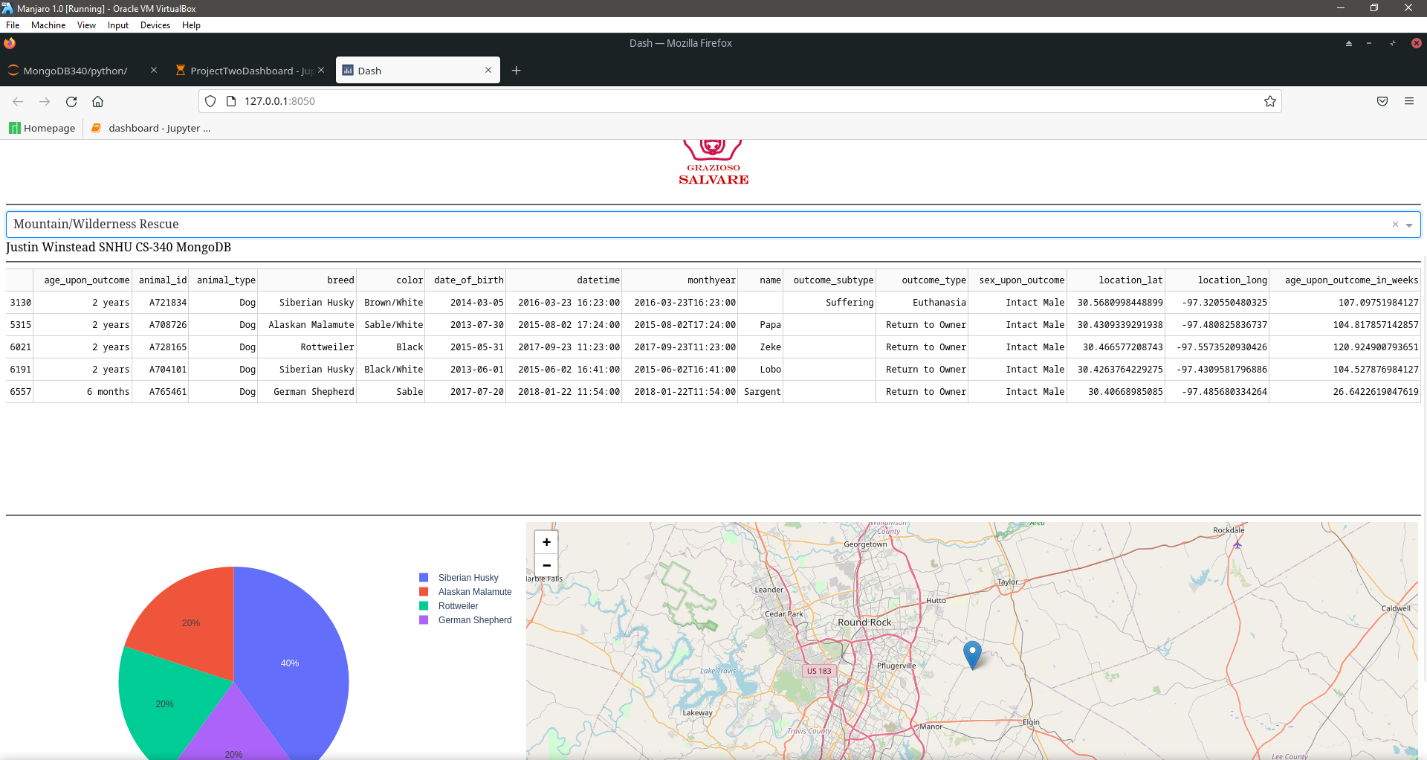
Description automatically generatedText

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated*Text

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, text

Description automatically generated*

**Challenges**

The largest challenge that I faced was using the Plotly Pie Chart. I felt that the documentation was not very thorough, and it left a lot to be desired. Eventually, through a lot of Google use, I was able to figure out how to make it work effectively with the dataset that I had.

## Contact

Your name: Justin Winstead