# CS 340 README

## About the Project/Project Title

*This application is an attempt to allow a user to easily engage with data stored within a MongoDB database entry. The use includes the interactivity of a data list combined with two visuals that provide additional context to the data stored and accessed in the database.*

*MongoDB was selected for its robust use of data models that provide users the ability to scale out the design as it continues to grow. The structure also provides a natural organization for data that doesn’t require the use of multiple data tables that are used with other database options (such as SQL). These benefits allow us to quickly build an experience for our users that can grow and change as our needs develop over time.*

*Beyond our storage of data, we will also focus on our interaction with that data, which will be accomplished through the use of the Plotly Dash framework that allows us to build with python into a web application. It is a tremendous tool for building data visualizations with customizations dependent on users and their desired level of interaction.*

## Motivation

*This project was created to provide a more user-friendly experience when interacting with a mongoDB instance. Instead of knowing mongoDB or the structure of the database, a user should be able to call on the native methods of CRUD.py to perform certain actions for them. This includes an easy to use “user interface” that provides an interactive experience to engage with data stored on the data table.*

## Getting Started

*Be sure to follow the recommended steps to get classes up and running*

## Installation

*- Start a mongoDB instance using the following command-*

*“/usr/local/bin/mongod\_ctl start”*

*-Open Jupyter notebook and access both the CRUD.py and ProjectTwoDashboard.ipynb files.*

*-Be sure to enter the appropriate username and password via within the ProjectTwoDashboard.ipynb file.*

*-To run the test be sure to open the ProjectTwoDashboard.ipynb file within JuptyerLab. Right click on the page and select “create console for editor”, then select “python 3”. Follow this by selecting the lines you wish to run and hitting “Shift+Enter” to run the code.*

## Usage

It is important to remember that this application functions by interacting with only one database*, which stores animal data that contains more than Dogs. Fortunately, the application data table and corresponding tables and filters will only provide data on dogs.*

*A user must select the appropriate filter to identify potential rescue dogs that are listed within the database. Not every dog will be eligible or available for use, but the data provided will serve as a great jumping off point for research and engagement.*

### Code Example

*This first sample shows the read method that accepts a key/value pair in the form of a dict, and passes that argument to the find() method native to mongoDB. If successful the response is returned, if not then we will see an exception. The printing of the return statement was initially difficult, but eventually was successful with corrections to the DB connection.*

*def read(self, data = None) -> dict:*

*"""process to read an entry, will need to pass a dict object that matches entries in the database"""*

*if data is not None:*

*run = self.database.animals.find(data,{"\_id": False})*

*if run != 0:*

*response = run*

*else:*

*response = "unable to find result."*

*else:*

*"raise Exception('Nothing to show, because data parameter is empty')"*

*return response*

This second sample shows the data table code, and should provide additional context to it’s set up. The table is assembled using the Plotly Dash framework. Dash provides a Python framework that allows us to easily build interactive web applications.

dash\_table.DataTable(

id='datatable-id',

columns=[

{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns

],

data=df.to\_dict('records'),

editable=False,

filter\_action="native",

sort\_action="native",

sort\_mode="multi",

column\_selectable=False,

row\_selectable="single",

row\_deletable=False,

selected\_columns=[],

selected\_rows=[],

page\_action="native",

page\_current=0,

page\_size=5,

),

This next sample shows the callback method, which is a native function within Dash that enables us to add additional logic and functionality to basic HTML visuals. This call back exists for the dropdown option, which provides the user with an ability to filter the datatable based on the options selected. You will notice that the variables provided to the read function are in JSON format, which speaks directly to the mongoDB database format.

@app.callback(

Output('datatable-id', 'data'),

[Input('dropdown','value')]

)

def update\_dataTable(value):

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

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

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

dog = {'animal\_type': 'Dog'}

if value is None:

df = pd.DataFrame.from\_records(shelter.read(dog))

elif 'wtr' in value:

df = pd.DataFrame.from\_records(shelter.read(wtr))

elif 'mtn' in value:

df = pd.DataFrame.from\_records(shelter.read(mtn))

elif 'dstr' in value:

df = pd.DataFrame.from\_records(shelter.read(dstr))

elif 'rst' in value:

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

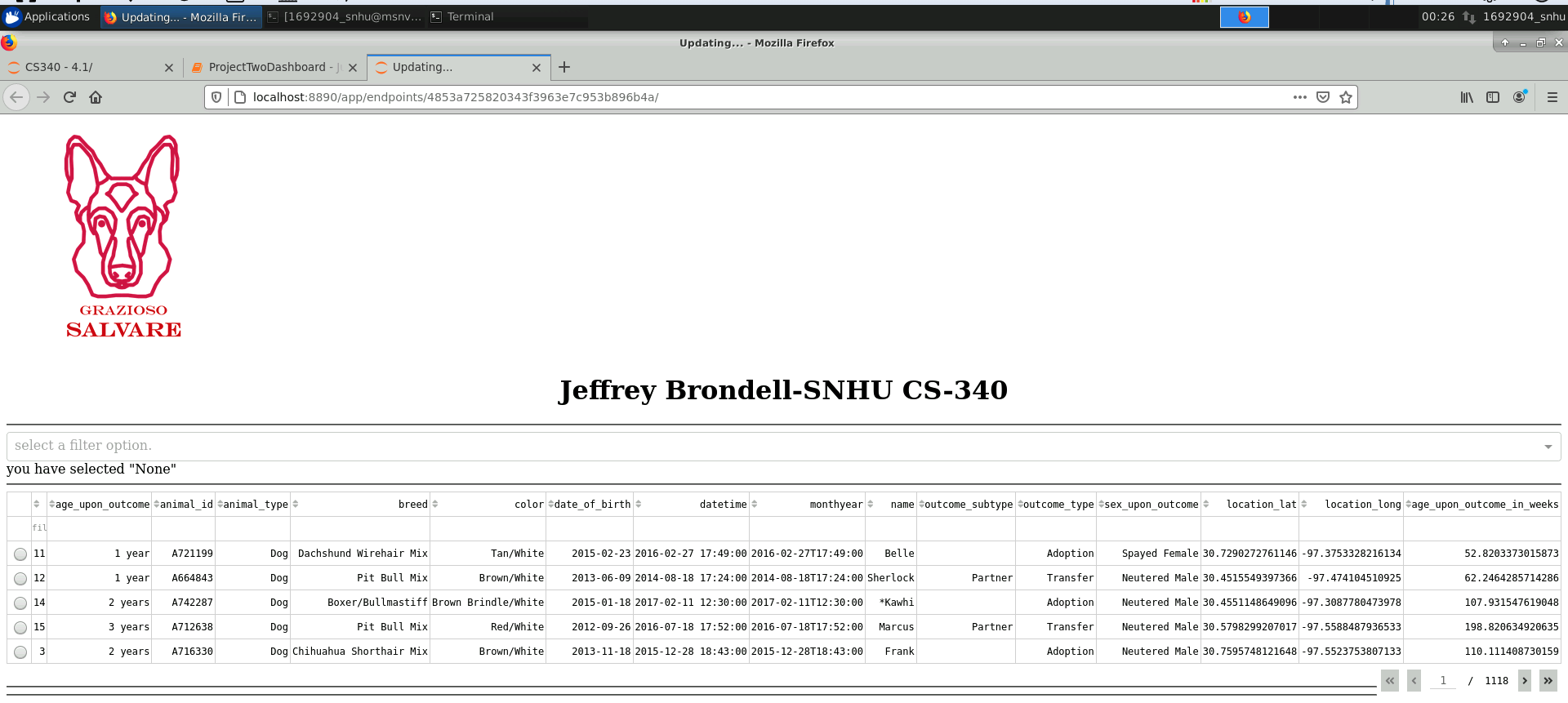
else:

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

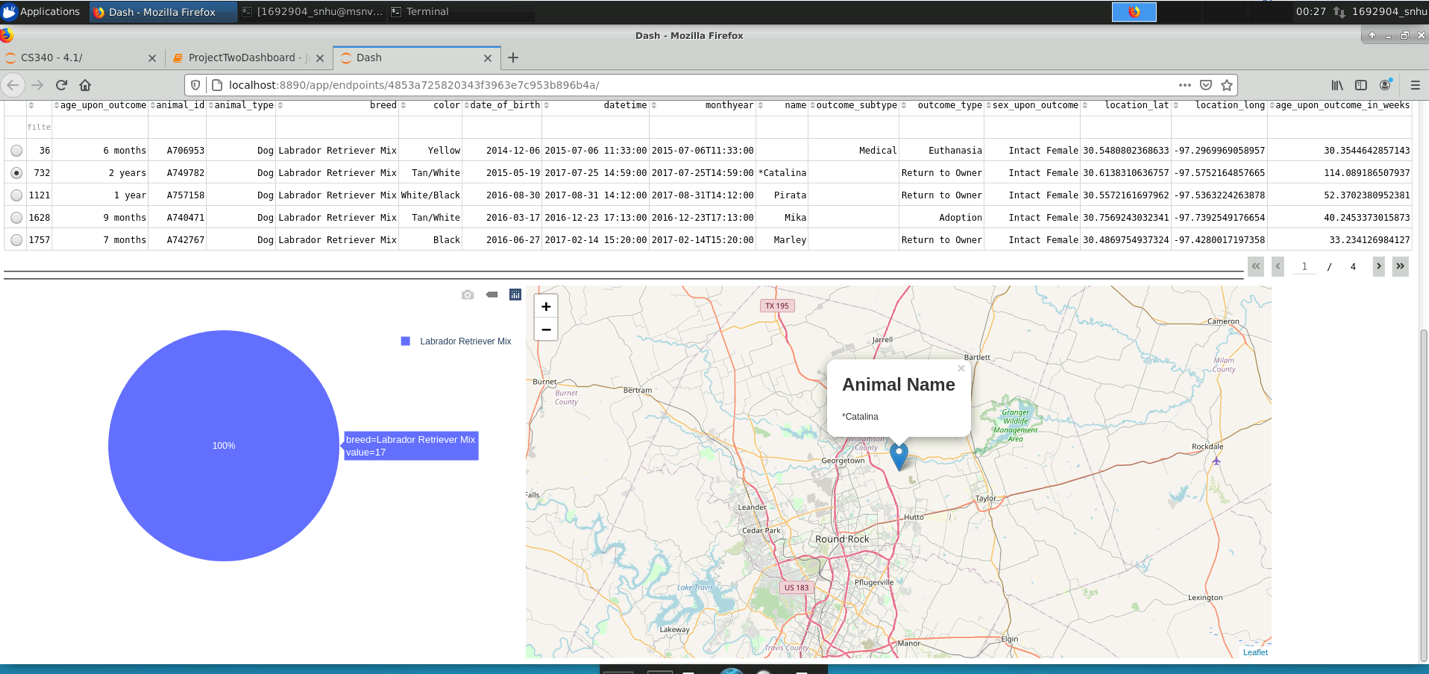
return df.to\_dict('records')

### Screenshots

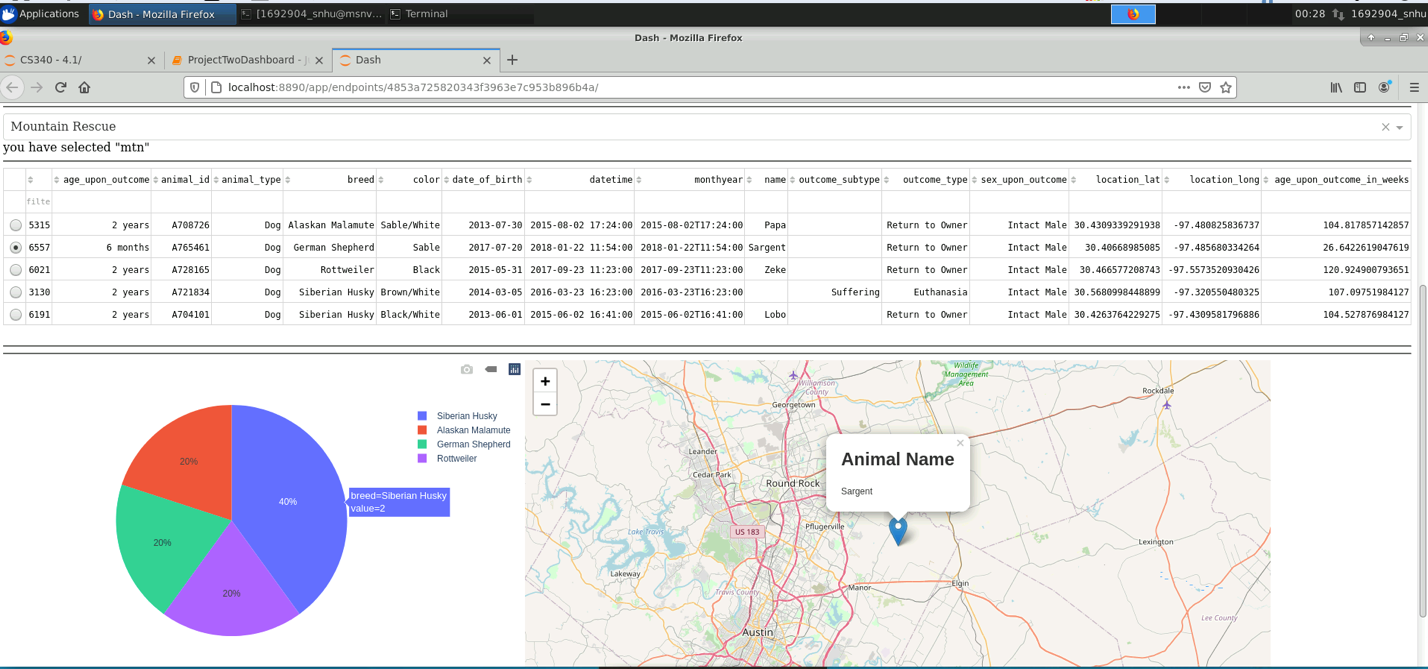
## -This first screen shot will show the data table without an filter options selected. The table will only display dogs, and will not provide any additional visuals unless a filter is selected or an entry selected on the table (for the geolocation). You will notice that the table provides native interactive options for filtering, selecting animals, and sorting. You will also notice the unique ID and the company logo in use.



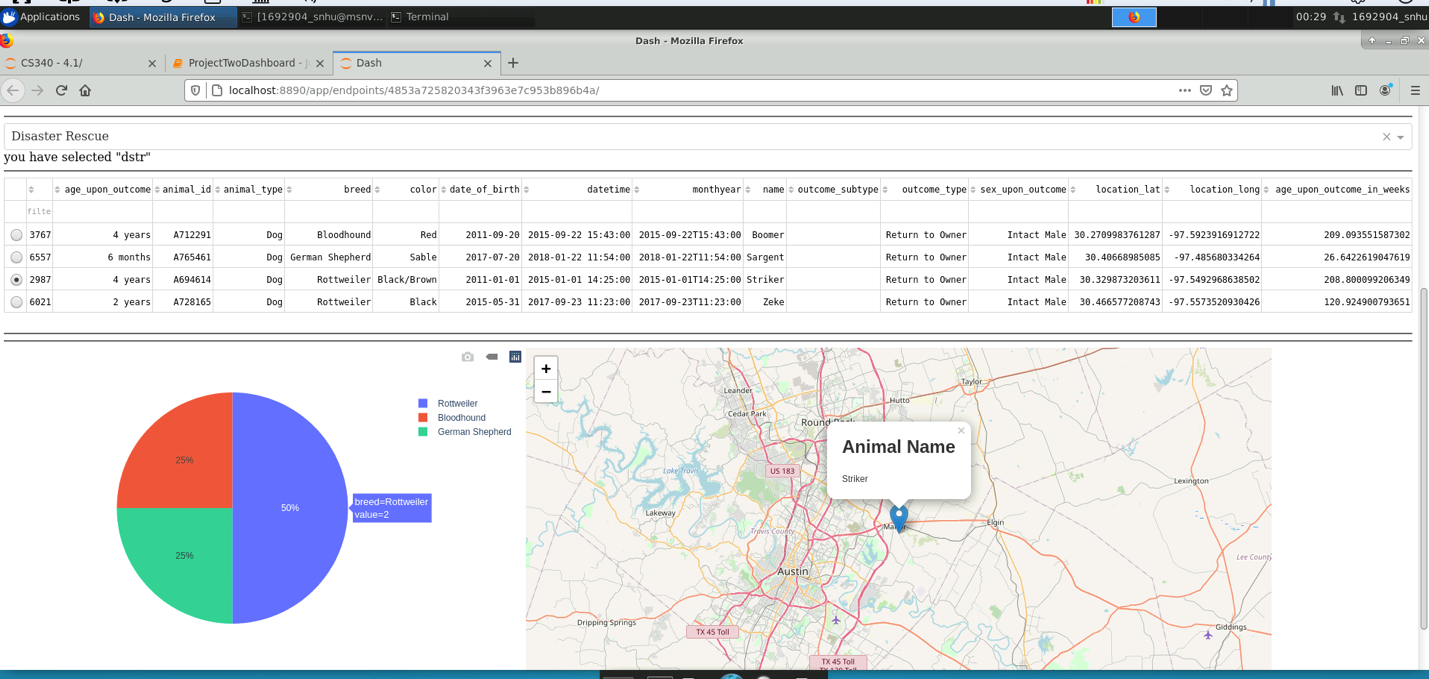
## -The second screen shot will show what happens if a user selects the Water Rescue option as well as selecting any one of the listed dogs. The graph will show the pie chart of data surrounding the Water rescue option and the geo location will show the location of the selected dog. (Notice that only Labrador Retrievers are available. This is due to the other limiters being gender and age and not just breed. It will be important for users to understand this.)

**

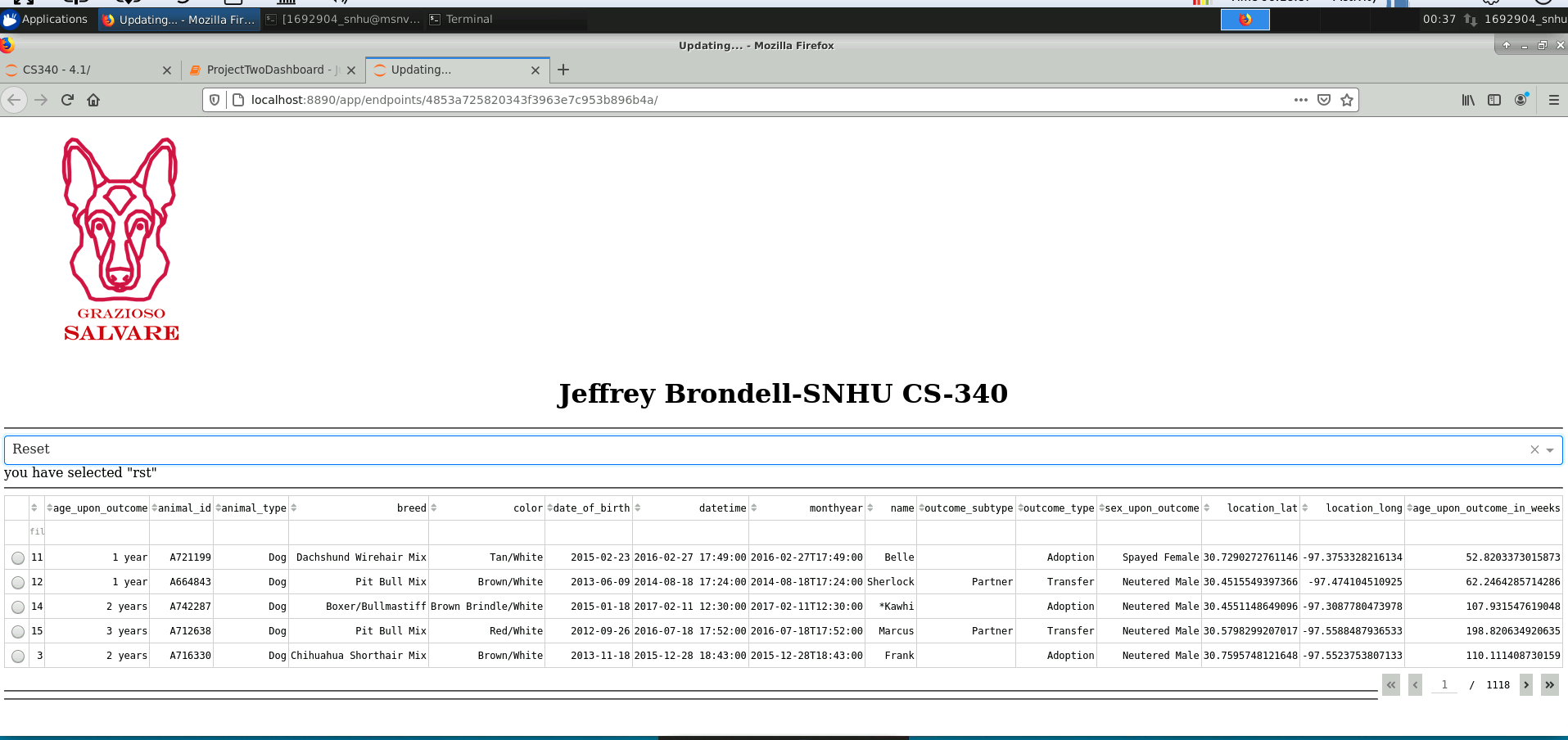
## -The third screen shot will show what happens if a user selects the Mountain Rescue option as well as selecting any one of the listed dogs. The graph will show the pie chart of data surrounding the Mountain rescue option and the geo location will show the location of the selected dog.



## -The fourth screen shot will show what happens if a user selects the Disaster Rescue option as well as selecting any one of the listed dogs. The graph will show the pie chart of data surrounding the Disaster rescue option and the geo location will show the location of the selected dog.



-The final screen shot will show what happens if the user selects the “Reset” option. Here we will see the removal of visuals as well as the resetting of the data table to show all dogs and not just those affiliated with the filters.



## Contact

Jeffrey Brondell (Jeff.Brondell@snhu.edu)