# CS 340 README – Brian Modzelewski

## About the Project/Project Title

The goal of this web application is to give a user-friendly interface for sifting through data contained in a database driven by Mongo DB in order to yield suitable breed, sex, and age combinations for training rescue dogs. A database, an API, and a dynamic dashboard comprise the project.

**Motivation for Using Mongo DB**

Mongo was utilized expressly for this software since it allows for speedy database setup from a csv file as well as a Python-friendly interface. Python can utilize database technologies such as SQL, but the syntax is so different that switching back and forth may be difficult. When conducting the fundamental CRUD tasks of a database, the selection tools are substantially more difficult in a SQL database than they are in a Mongo DB using Python.

**Motivation for Using Dash**

Dash, the technology used to create the dashboard, was appealing owing to its dynamic character. Dash is a JavaScript-based react utility that provides an extremely responsive foundation. Dash uses html Dash tags to control segment outputs. Then, depending on Python module instructions, changes are made to any of the target inputs defined in the app callbacks process.

**Getting Started**

*To get a local copy up and running, follow these simple example steps:*

*Make sure all dependencies are installed.*

*Then you can create a new Python file and import the AnimalShelter module.*

*(from crud import AnimalShelter)*

*From this you can create a new AnimalShelter Object and connect to the database*

## Installation

The tools you will need to run this include Jupyter Notebooks, Python for command line, and MongoDB. The installation of each is detailed in labeled sections right below this line.

**Jupyter Notebooks**: Jupyter can be installed from the command line in any major operating system using the simple instructions here: <https://jupyter.org/install>. For detailed instructions like Proxy servers for Windows, Mac, and Linux, follow the instructions here: <https://jupyterlab.readthedocs.io/en/stable/getting_started/installation.html>

**Python**: Detailed installation instructions for Python are available here: <https://realpython.com/installing-python/>. Once you have Python installed, you should be able to use this program from the Terminal on Mac or Linux or from the Command Prompt for Windows.

**MongoDB**: MongoDB comes in Community or Enterprise editions. Detailed instructions for the installation and downloading of MongoDB are available here: <https://docs.mongodb.com/manual/installation/>.

**Plotly**

Plotly must be imported in order to generate the proper charts. Plotly is a charting tool for Python applications and can be imported directly into your Python module from your Jupyter notebook. If you need a local copy of Plotly installed, see the documentation here: <https://www.journaldev.com/19692/python-plotly-tutorial#:~:text=Installation.%20To%20install%20plotly%2C%20open%20a%20terminal%20window,to%20install%20to%20collect%20dependencies%20and%20download%20them%3A>

**Dash**

Dash is a framework used to build web applications. You can import the Dash Core Components into your Jupyter notebook and you can install Dash using the following information: <https://pypi.org/project/dash/>

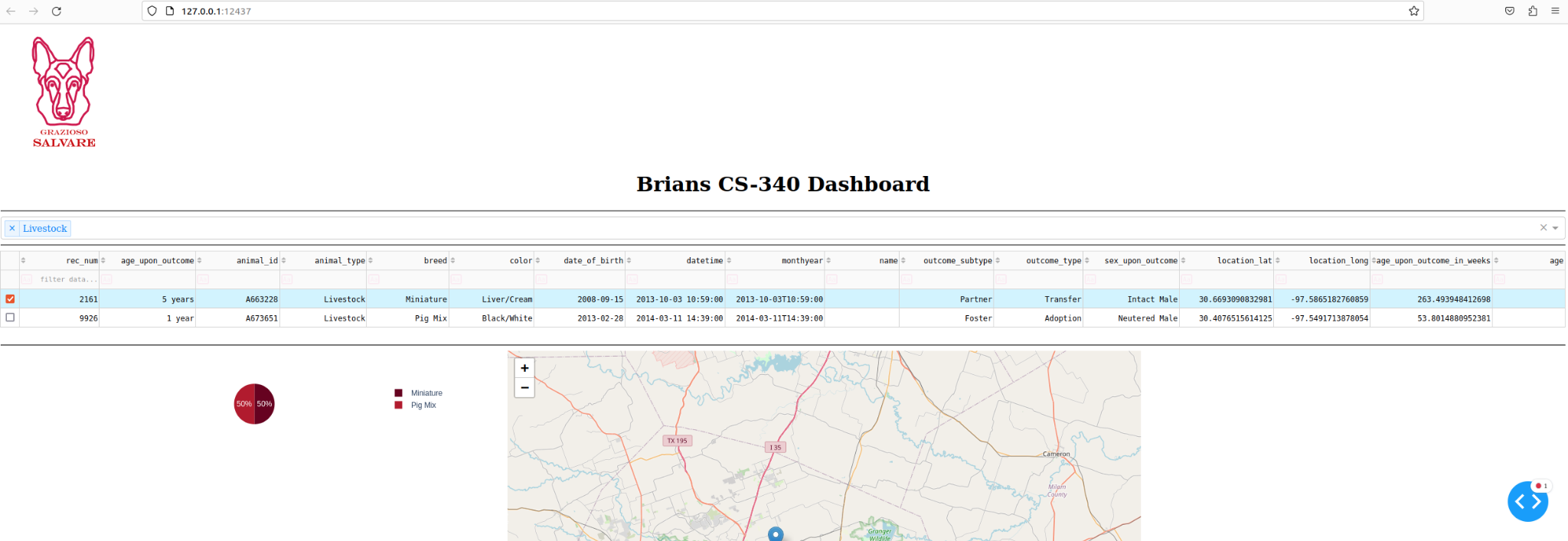
**Pandas**

Pandas is used in this web application as well. Pandas is a tool for Python that creates the data frames. Pandas has other dependencies and information that should be reviewed before use here: <https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html>

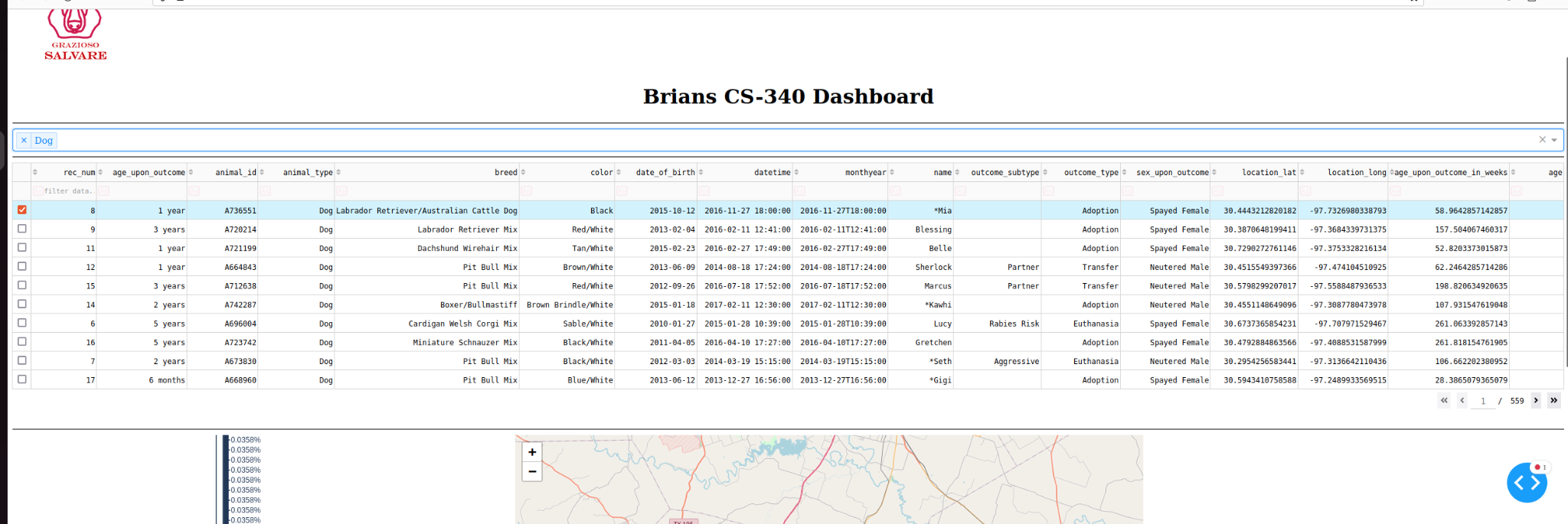
## Usage

This application presently serves three purposes. The first employs radial buttons to sift data regarding the many sorts of rescue canines requested by Grazioso Salvare based on breed, age, and sex characteristics. By clicking any of the drop down, the database queries are executed and an updated data frame with the appropriate results is returned. The map and dynamic map updates are the second function. This places a marker on the map at the location of the first item specified in the data frame. The map shifts the marker (or adds markers for multiple selections) to the selected row(s) once the user picks a row (or up to five at a time). The dynamic pie chart is the third function. This just sorts the data by breed and generates a pie chart based on the breed of animals exhibited from the current data frame (not the entire database). This application presently serves four purposes. The first two collect information from the user about what to add into the database (create) or what to look for in the database (read). The second and third functions are responsible for the creation and search. The acquire functions request particular input from the user (for read data, a key and a value are requested to execute the search, but in the create function, the keys are supplied but just the values are needed).

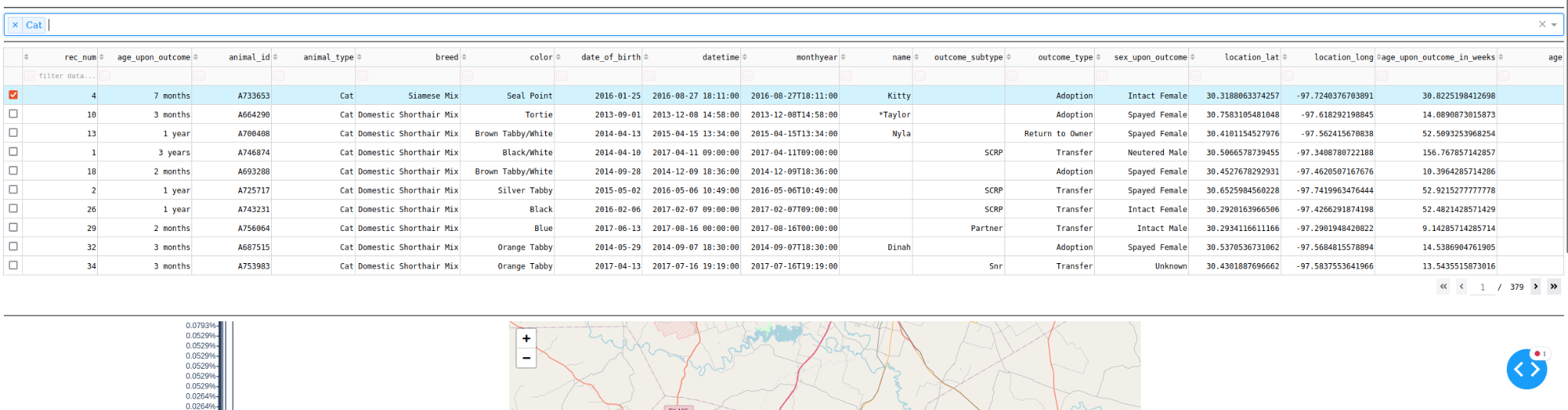
**Livestock**

****

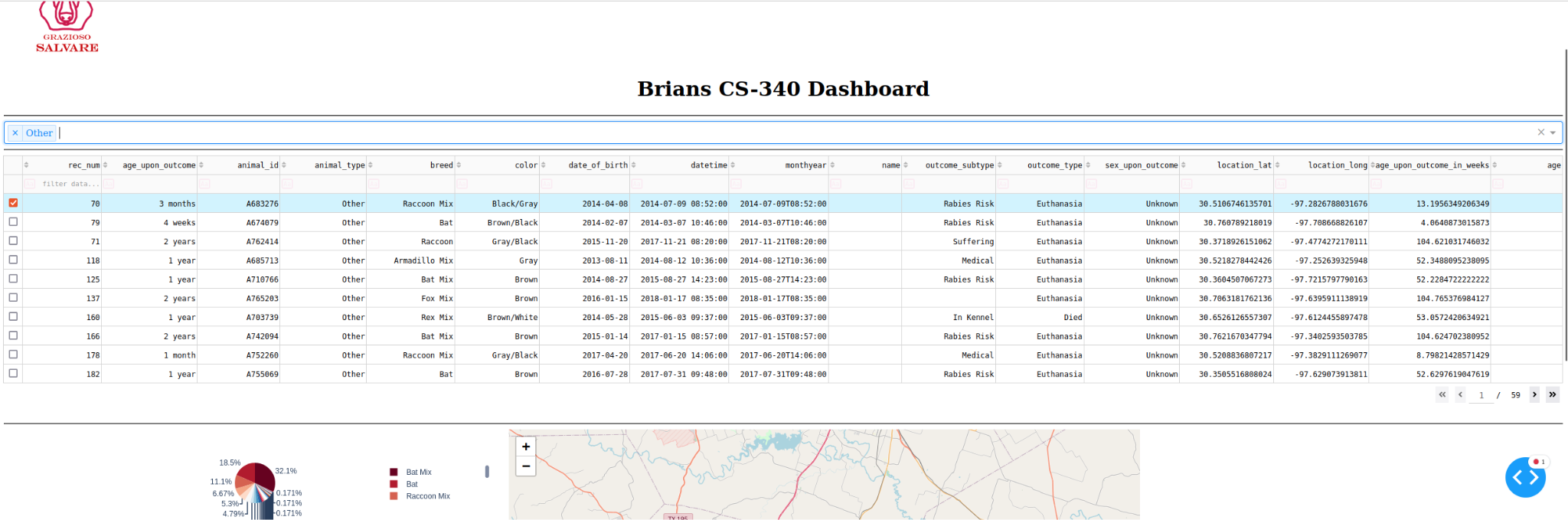
**Dog**

****

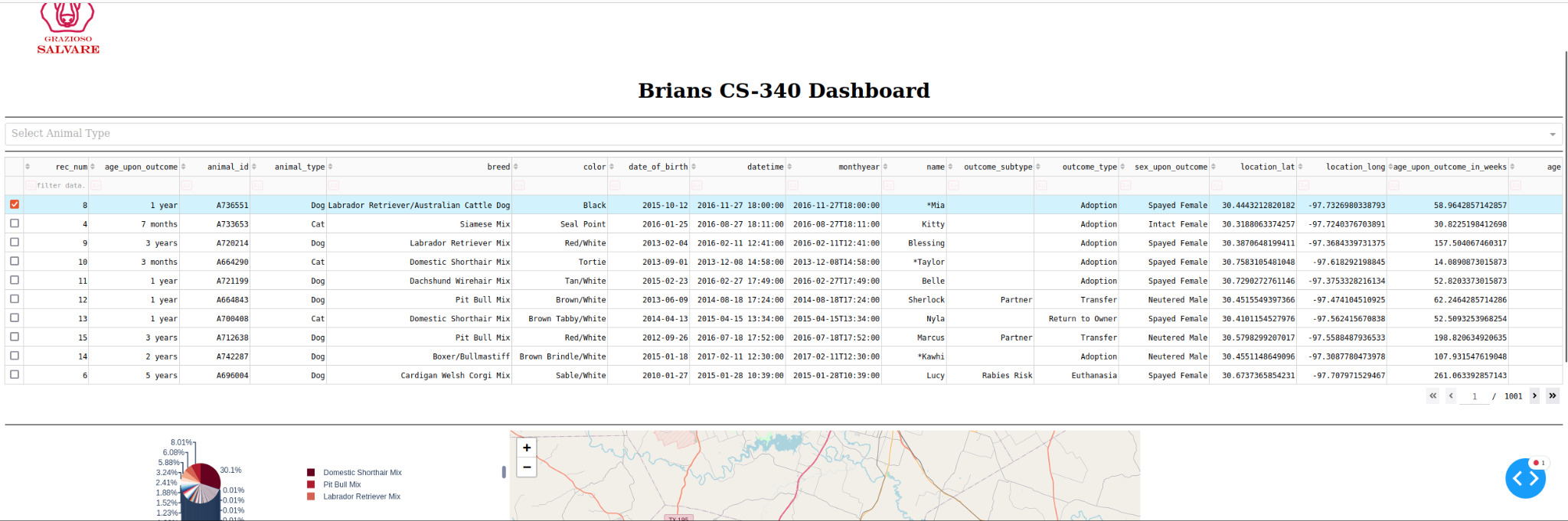
**Cat**

****

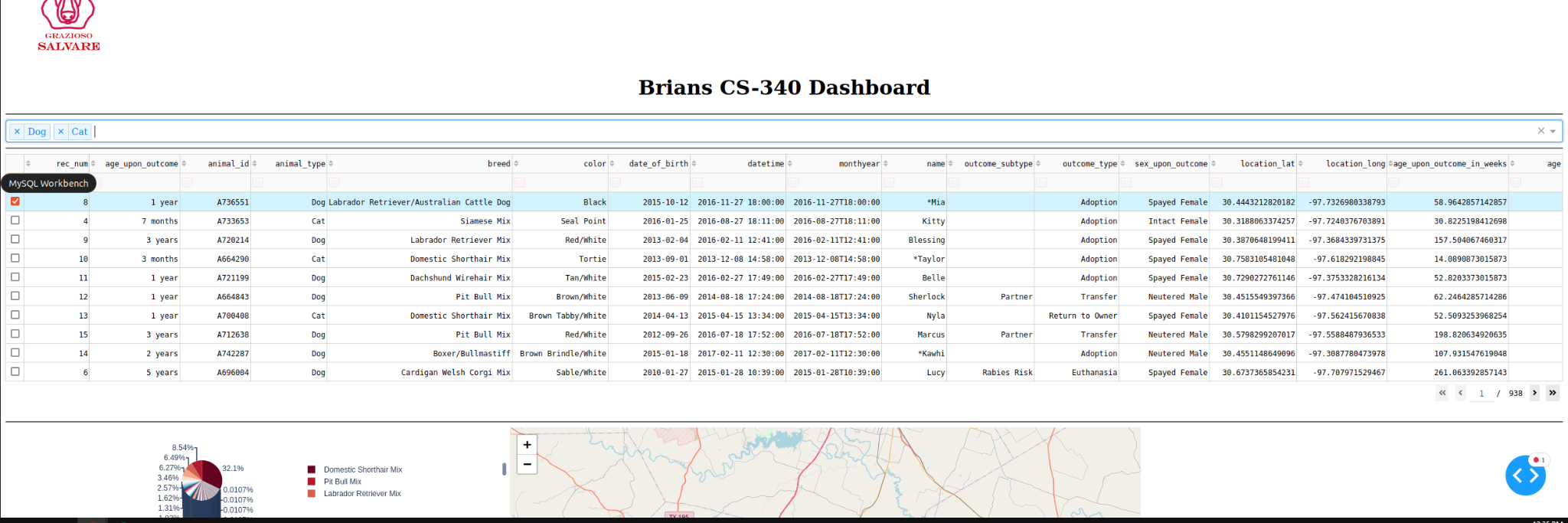
**Other**

****

**Reset (All options shown)**

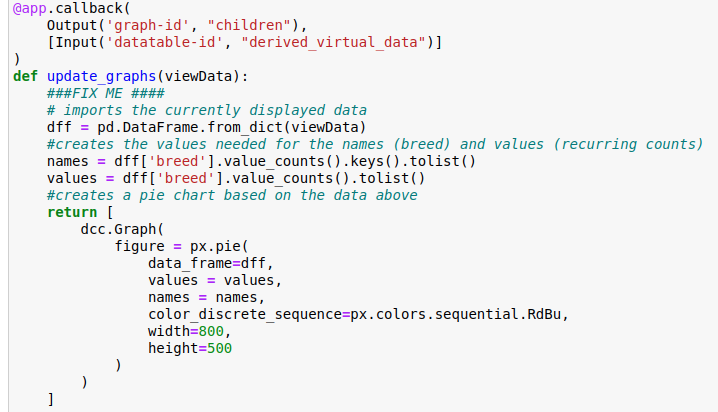
****

**Multi selection**

****

**Code Samples from Dashboard**

**Pie Chart**



### Code Example (CRUD Operations)

*def create(self, data):*

*#Insert a document into the specified MongoDB database and collection.*

*#Return True if successful insert, else False.*

*if data is not None:*

*result = self.collection.insert(data)*

*return True if result.inserted\_id else False*

*else:*

*raise Exception("Nothing to save because data parameter is empty")*

*def read(self, query):*

*#Query for documents from the specified MongoDB database and collection.*

*#Return the result in a list if the command is successful, else an empty list.*

*cursor = self.collection.find(query)*

*result = [document for document in cursor]*

*return result*

*def update(self, query, update\_data):*

*try:*

*updated\_count = self.collection.update\_many(query, update\_data)*

*return updated\_count.modified\_count*

*except Exception as e:*

*print(f"Error updating documents: {e}")*

*return 0*

*def delete(self, query):*

*try:*

*deleted\_count = self.collection.delete\_many(query)*

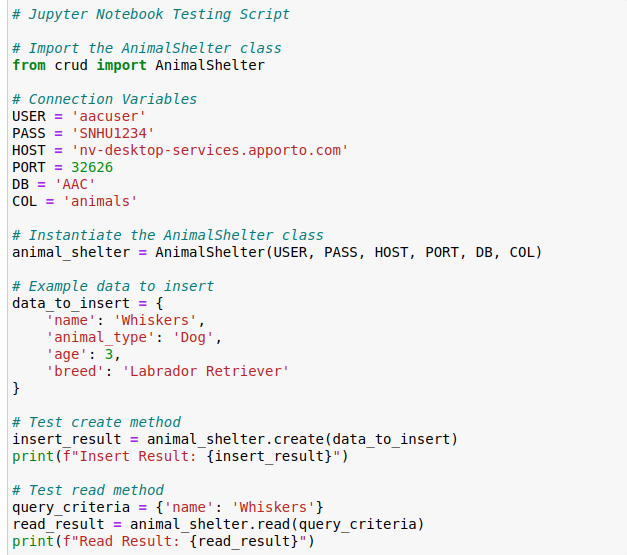
*return deleted\_count.deleted\_count*

*except Exception as e:*

*print(f"Error deleting documents: {e}")*

*return 0*

### Screenshots

**

## Contact

Brian Modzelewski