## About the Project/Project Title

This project is called Shelter Sorter. This project is designed to help clients parse a database for specific animal conditions, in order to find good animals for specific jobs. This was designed for the company

Grazioso Salvare, who wanted help parsing a shelter database for animals that can learn how to be rescue dogs. MonogDB is used as the database server for this project because it can easily interface with Python using the PyMongo libraries. The dash framework was used because it can create easy to use web interfaces with minimal to no HTML knowledge needed. It also provides easy ways to work with database structures, like cursor objects which are returned by queries.

## Motivation

This project is being created to help Grazioso Salvare easily find animals that can be trained to take part in certain rescue missions. They identified specific conditions that make ideal search animals for different environments but were not able to easily search shelter databases before.

## Getting Started

Make sure you have a running instance of a MongoDB server. This can be started with or without authentication but running with is recommended. To run with authentication there will need to be user accounts already created. User accounts can be created by following the instructions on the MongoDB website, [MongoDB Manual Enable Access Control](https://docs.mongodb.com/manual/tutorial/enable-authentication/). Once that is running change the port number in the init method (after the localhost area) to the port your Mongo instance is using, and change the database to whatever your database is. My examples say AAC so change any instance of AAC to your database name. From there you can copy the test code from the example below and enter your desired data and query. This is all done within the AnimalShelter.py file. From there you should be able to run the Project2 file, which will display within a web application

## Installation

To use this project a running MongoDB server will be needed with the desired databases installed. It is recommended to have user accounts setup for this project as well. Python will also need to be installed to run this project as well. The Jupyter notebook will also need to be installed in order to run the ipynb file.

## Usage

Once the project 2 file is run, there should be no need to interface with any code. This is a fully functional webapp. Below are some examples of the different screens you would see while using this project.

### Screenshots

This is the initial starting screen when loading this project. Notice the different radio buttons at the top of this dashboard. Those control the filters we have set based on specifications from GraziosoGraphical user interface, text

Description automatically generated

This is what shows up when running the disaster rescue filter.

Graphical user interface

Description automatically generated

This is what shows up when running the water rescue filter.

Graphical user interface, application

Description automatically generated

This is what shows up when running the mountain rescue filter.

Graphical user interface, text

Description automatically generated

When the reset button is checked, which is what the default starting place is, this happens.

Graphical user interface, text, application

Description automatically generated

## Challenges Faced

One of the major challenges faced in this project was trying to get the map to display all the markers that are shown in the dashboard. This was achieved by creating a helper method that takes the dataframe and the index of which row we are on. I then ran a loop that added markers to a list which was passed into the map object as children.

def update\_map(viewData,filter\_type):

dff = pd.DataFrame.from\_dict(viewData)

# Austin TX is at [30.75,-97.48]

markers = [dl.TileLayer(id="base-layer-id")] # start with this and append markers

if filter\_type == 'diaster' or filter\_type == 'mountain':

for i in range(0,4): #in a dynamic database this wouldnt be hard coded but I couldnt figure out how to get the row count easily

markers.append(getMarker(dff,i)) #to do multiple markers

else:

for i in range(0,10):

markers.append(getMarker(dff,i))

return [

dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.5,-97.48], zoom=10, children=markers)

]

Another challenge I faced was getting the pie chart to work based on the filters. One part of that was when running the unfiltered view of the database, there was so many species that the pie chart looked very bad. In order to fix this I ran a loop looking for any breeds with less than 50 animals as part of that. They were added to the “other breeds” category, and this cleaned up the chart a lot. You can see the code used for this within project two, the source code is to cluttered to add into this readme.

## Roadmap/Features (Optional)

This project is considered complete at this time. Further releases might happen but are not planned at this time.

**External Resources used**

[Dash Core Components](https://dash.plotly.com/dash-core-components)

[Plotly Pie Charts](https://plotly.com/python/pie-charts/)

[PyMongo](https://api.mongodb.com/python/2.3/faq.html)

[Dash Leaflet (for Map)](https://dash-leaflet.herokuapp.com/)

## Contact

Your name: Brian Golden