# CS 340 README ANIMAL SHELTER

## About the Project/Project Title

This project handles the filtering of shelter animals with the assistance of a pyMongo Crud module associated with a MongoDB Database. The front end uses Dash, Pandas, and Plotly to show data effectively and efficiently to the user with easily filterable options. When selected from the Dash Data table the location of the animal is shown on the interactive map. When filtered animals are sorted by breed onto the accompanying pie chart.

## Motivation

This project was built primarily to help Grazioso Salvare quickly find Dogs within a specified set of criteria to be trained as search and rescue dogs. Given the criteria they have found that best meet specific needs of different rescue missions the Dogs are sorted and filtered.

## Getting Started

To get started you will need Python 3.X and all the appropriate modules/packages installed on your machine (see import statements at top of both packages. All packages can be easily installed with the pip install command. Next ensure your MongoDB is actively running, you have input the port into crud module, and have changed the login credentials for your intended database.

Text

Description automatically generated

The port and the Database name need to be changed in both lines of the Animal Shelter class (port is 37046 and database is AAC in above screenshot).

Graphical user interface, text

Description automatically generated with medium confidence

Here username, password and the class name need to be updated for your work. These can also be added to your project as entered criteria, though they are hardcoded for ease in this example.

From here you are ready to start using the modules and MongoDB with the appropriate CRUD commands from the Animal Shelter class you have instantiated. In the attached program we use these basic steps to setup a Dash application in the browser as shown below.

Table

Description automatically generated with low confidence

How this was done will be shown further on in this read me.

## Installation

If you followed the steps above in terms of downloading all the required modules, running a MongoDB server, and using pyMongo you should be good to go. This project was built in the Jupyter Notebook so minor modifications will need to be made to run this as a stand-alone application outside of Jupyter Dash. Please see the Dash documentation for more information on how to run a dash application.

## Usage

### Code Example

For using the CRUD module follow these steps:

First, import the module and instantiate with a username and password:

From animal\_shelter import AnimalShelter

Shelter = AnimalShelter(“username”, “password”)

To insert a document, use the create function:

Shelter.create({“field\_name”: “desired\_value”, “field\_name2”: “desired\_value”})

To query an individual document and get the first matching document use read\_one:

Shelter.read\_one({“field\_name”:”value to match”}) //Multiple fields can be input

To get a cursor pointing to all matching documents use read\_all:

Shelter.read\_all({“field\_name”:”value to match”})

To update a document, use the update function:

Shelter.update({search\_dictionary}, {field\_name:value})

To delete a document, use the delete command:

Shelter.delete({search dictionary})

{Please note it is important to ensure you use your search dictionary with the read\_all command first to ensure you are only deleting the individual document you want, and not attempting to delete multiple documents or an incorrect document}

For the Dash application we will need to get pretty heavy into the Dash Documentation as while the website is nice, reference material can sometimes not be adequate for even basic work in Dash if you are new to it.

We will start with the DataTable Shown below:

Table

Description automatically generated with low confidence

To build this table with the already setup Mongo Database and CRUD module we will need a little bit of setup code to prepare for the table.

Text

Description automatically generated

The above code imports the Pandas Data Frame of all information in the database. (the df.drop command removes the “unserializable” MongoDB ID object as we do not need it and it can cause issue if kept in the data frame)

Text

Description automatically generated

Above we are building a list of hidden columns, this for loop excludes the listed columns and appends all others to the default\_hidden list. This is an easy method for when you dynamically load data frames to a Dash Data Table and only want to show a handful of the columns initially.

Graphical user interface

Description automatically generated with low confidence

These are our filtering dictionaries. Each dictionary was built and tested to ensure it returned all relevant results for our goals with finding Dogs to train for various rescue tasks. Modify these dictionaries as needed to meet your needs, or build new dictionaries (just make sure to include options for filtering in the RadioItems object.

Now that we have laid the groundwork for our table and radio buttons lets look at how we actually implement them in a dash application. As you can see below, the dcc.RadioItems component is held withing a html Div. The “options” field can take data in various ways, but for ease we found that adding buttons as a list of dictionaries was the most effective way. Here we set the label, the value and the id of each button. Setting the label ensures we have text next to the radio item, the value is what we use to determine which button is selected in the application callback and the id allows you to target a button directly (though we do not use that functionality in this project).

The data table is in effect its own html component. The most important feature of the dash table are the first 4 settings. First, we ensure we have an id, without this interactivity is difficult/impossible in Dash. Next, we set the columns, each column has a name and id, they are not delete able, they are hide able (enable toggle for columns) and they are selectable as cells. From here, we load in the actual data for the table with the data = df.to\_dict(‘records’) command, ensuring each cell is filled with the appropriate data for the appropriate column. Finally, we use the default\_hidden list to set all undesired columns to hidden. This last command allows the data to be toggled if the user desires, while not showing unnecessary information initially. The remaining lines all deal with style and settings for the data table, for now you can use these but I encourage you to view the dash data table references to understand what each of these settings do, especially if its function is unclear to you.

Text

Description automatically generated

So now that we have a basic data table setup it is time to add a callback to make the table interactive. For this we setup a basic dash callback function.

Text

Description automatically generated

The app.callback function takes the value from the radio item we setup earlier. The output portion goes to our data table, sending back the updated data value. As you can see, we use the id of the required input and output and specify what we want to input and output from/to. In this case we are outputting to the data attribute of the data table, and we are taking the value attribute from the radio item as input to our function. (Note: a function defined below the callback is executed when the callback is triggered). In the function, we load the data frame fresh from the database then we filter the data with the predefined dictionaries based off the value selected in the radio item. From here we ensure that we drop the “\_id” column once again and then return the df as a dictionary like we initially did to populate the table’s cells.

Next, we will build the graph and map. The code to add the map and chart are relatively simple as dash does most of the work in the background.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Above, the graph and chart are both held in html divs held in one larger div. This code is relatively straightforward though it is important to note that each feature is contained as a child (children =[]) in the div. With this code, you will see a blank chart and a map of the world centered on Austin, Texas (center = [30.75, -97.48] is the latitude and longitude of Austin). Additionally, in the maps children we include the basic map tile layer, please see the dash-leaflet documentation for more complicated maps and features.

To build and populate the graph, we will need a callback function. Below you can see a basic dash callback like the one for the data table, though the output has been changed to the figure attribute of the graph. The only real change in this callback function as compared to the one used above is the fig line. Here we build a Plotly express pie chart with the data frame. The names field is used to chose the field we would like to display (in this case breed) and finally the hole attribute just creates the hole in the center of the pie chart.

Text

Description automatically generated

Finally, we will discuss the map. Here, we take two inputs, one is the the derived\_viewport\_data, the other is the selected\_rows attribute. The first is used to take the smaller filtered data table (and could be implemented in the chart function as well), the later is used to determine rows that are currently selected. We initialize an empty list called markers then iterate through the derived data population the latitude and longitude for each animal. Each animal also gets a tooltip with their name and the animals shelter id. Once each marker is returned the list is added to the tile layer and then returned to the map’s children. Each time a row is selected the callback function adds or removes the required marker and returns it to the map.

Text

Description automatically generated

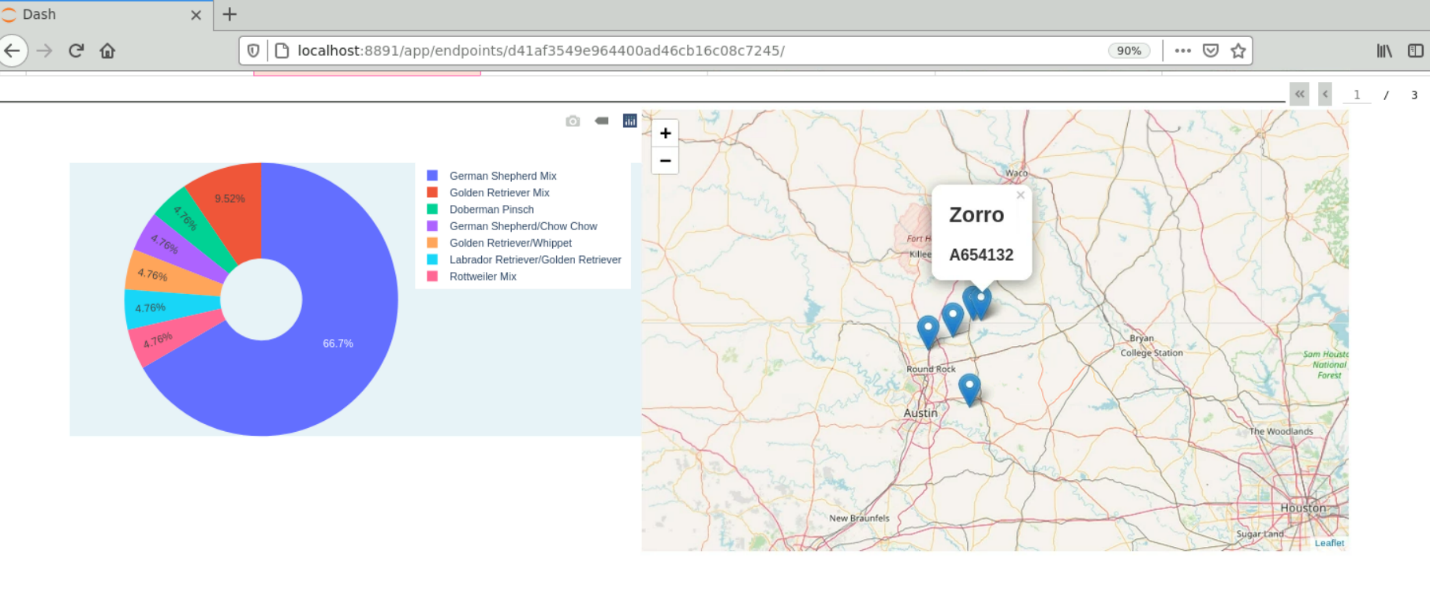
Tests

To test this application, attempt to change rows, toggle columns, change filters and test anything else. The program should work relatively smoothly, though it does load somewhat slow initially due to the initial layout of the pie chart containing hundreds of breeds.

### Screenshots

The above section for sample code contains many of the screenshots related to the project. Below are some additional (and repeat) screenshots of the application and its features.

Table

Description automatically generated with low confidenceA picture containing table

Description automatically generatedGraphical user interface

Description automatically generated with medium confidenceA picture containing graphical user interface

Description automatically generated

Of note is the toggle columns button, which allows a user to determine which columns are visible. This is done with a checklist dropdown menu. This is all handled by dash internally when the columns are set to hide able.

## Roadmap/Features (Optional)

One improvement that should likely be made in this project is the pie chart initial loading time. Due to the numerous different breeds contained in the database this feature takes a while to load initially, and each time the data table is set to all. To alleviate this it may be worth disabling the pie chart when the data is unfiltered.

## Contact

Joshua Kovacevich, Email: Joshua.Kovacevich@snhu.edu