

## About the Project

Grazioso Salvare is a rescue-animal training company and part of their work is to find dogs that are good candidates for search and rescue training. Grazioso Salvare has asked for software that can interact with existing data from animal shelters in the aim of identifying suitable candidates for training. The end state of this project will be a full stack software program capable of identifying and filtering training candidates. The first step of this project is to create Python middleware that can interact with MongoDB, specifically CRUD (Create, Read, Update, Delete) functionality.

## Motivation

The goal of this project is to create an open-source dashboard that will allow Grazioso Salvare and other companies to utilize this system and adjust them to their own purposes. This program could be adjusted to filter for service dogs, therapy dogs, or simply be used by an adoption agency to find animals that customers request. This makes the software much more modular and able to expand into different areas of the industry.

**Tools used**

MongoDB was selected as the database utility because it is an extremely powerful and fast database manager. There is also a library that interfaces it with python extremely well in the form of PyMongo. Dash is used as the framework to display the information derived from MongoDB to the website dashboard. To interface between MongoDB and Dash, Pandas was used to create data frames from the MongoDB data that Dash can read and display in html format.

**Challenges**

The biggest challenge of this project was getting the map to recenter itself when a different row in the data table is selected. The issue was that I didn’t read the Dash documentation closely enough to realize that the callback to derived\_viewport\_selected\_row\_ids returns a list of indices, and not the index of the row that is selected. After much trial and error, I eventually figured out that using the following lines of code to get the coordinates of the animal was all I needed to get it working perfectly:

|  |
| --- |
| lat = dff.get('location\_lat')[selRow[0]]  long = dff.get('location\_long')[selRow[0]] |

Everything else in this project was reasonably simple. I struggled with Dash’s callback format at first because it was new to me, but once I understood that all it is is preset variables to be passed into a function and returned it made perfect sense to me.

## Getting Started

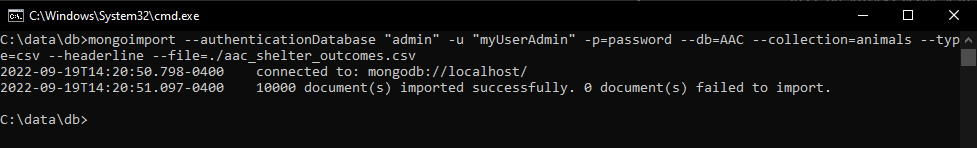
To get started, import a csv database file into MongoDB using the following command in the terminal or command prompt window:

|  |
| --- |
| mongoImport *--*db=DatabaseName *--*collection=CollectionName *--type*=csv *--*headerline *--*file=./fileName.csv |

If the database requires authentication, add the csv file as an administrator using the following command:

|  |
| --- |
| mongoImport *--*authenticationDatabase "admin" -u "AdminUserName" -p=password *--*db=DatabaseName *--*collection=CollectionName *--type*=csv *--*headerline *--*file=./fileName.csv |

Here is a screenshot of the command I used with the successful output:



Add the AnimalShelter.py file to the directory where your project files are and import it using the following command:

|  |
| --- |
| from AnimalShelter import AnimalShelter |

## Installation

Pymongo, numpy, pandas, dash, jupyter-dash, and dash-leaflet are required for this project. To install them enter the following command into your terminal:

|  |
| --- |
| pip install pymongo numpy pandas dash jupyter-dash dash-leaflet |

If Python is not linked to your path and the above command does not work, you can try:

|  |
| --- |
| python3 -m pip install pymongo numpy pandas dash jupyter-dash dash-leaflet |

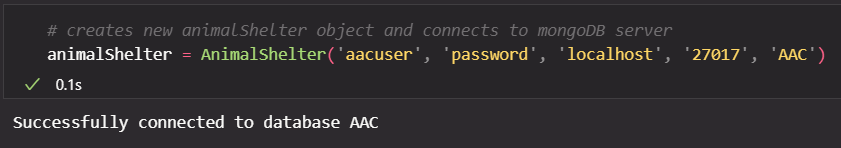
## Usage

**Connecting to MongoDB**

To connect to a database, instantiate a new AnimalShelter object with the following command:

|  |
| --- |
| animalShelter = AnimalShelter('userName', 'password', 'ipAddress', 'port', 'databaseName') |

### Where each data member is a string relevant to the database you are trying to access. Upon successful connection to the database, a message will be printed as shown in this example:



If the server is unavailable, or if the login info is incorrect an exception will be raised, and the program will terminate.

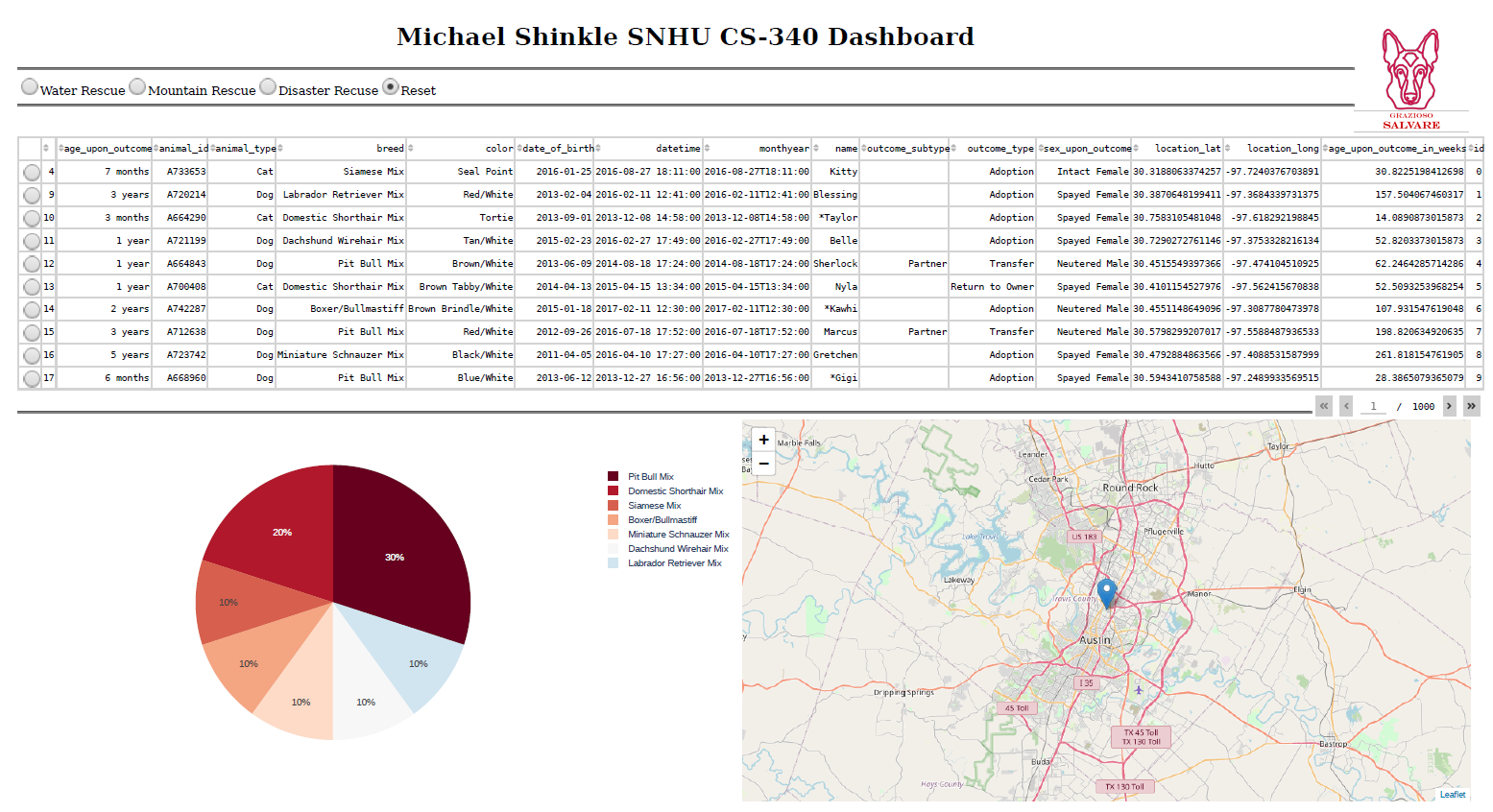




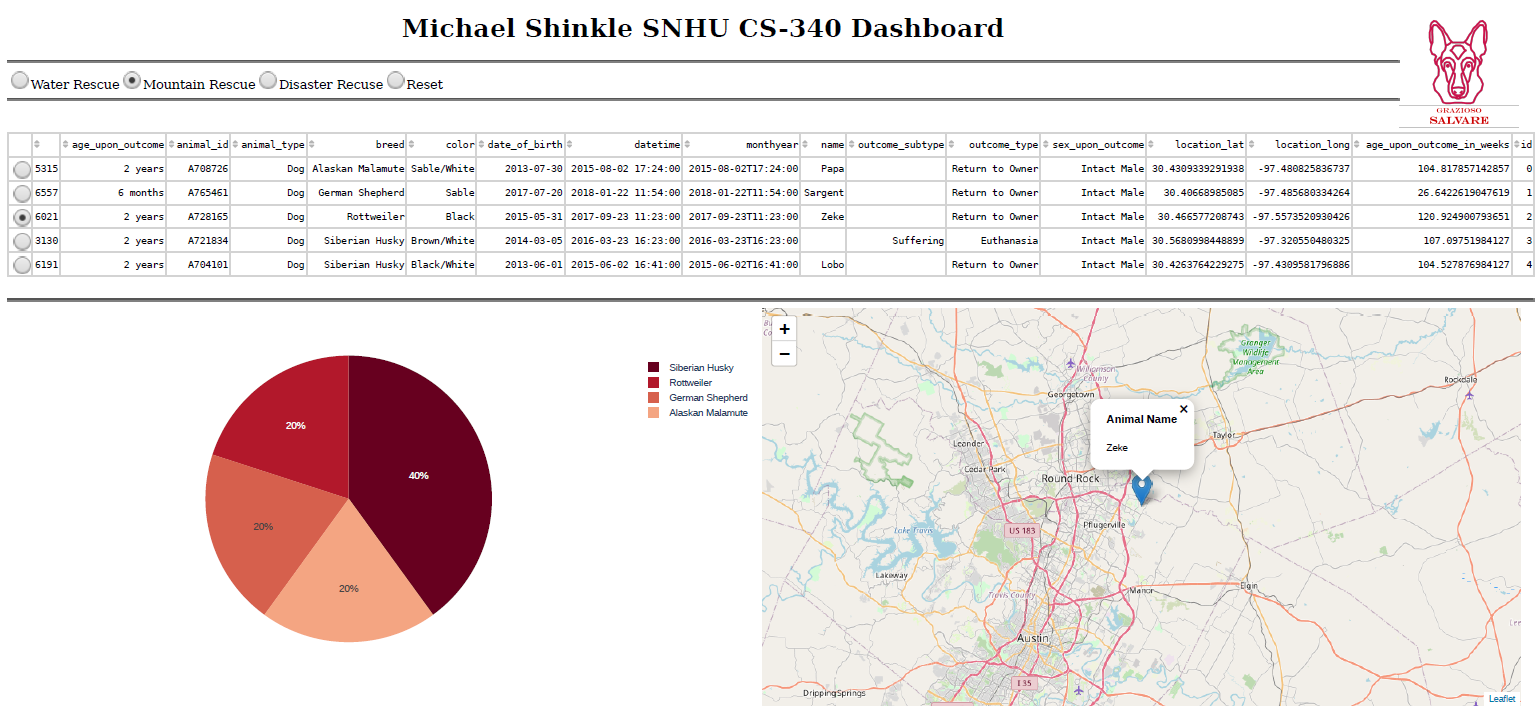
**Access the Grazioso Salvare Dashboard**

To access the Grazioso Salvare Dashboard, add the ProjectTwoDashboard.ipynb file to the same directory as AnimalShelter.py. Update the login credentials to match your MongoDB and run the code. When the information is displayed you will see a table with 10 rows corresponding to each document in your database. The table is paginated to avoid an overflow of data on the screen. Below the data table is a pie chart showing the breed information of the data currently being shown in the data table, as well as a map showing the location of the animal. If you select a row in the data table, the map will automatically recenter on the location of that animal. You can also filter for animals that would be adept at the type of training that is displayed in the radio buttons at the top. When you make a selection from the radio buttons, the data table and pie chart will automatically update to match the search query. The reset button returns the original data frame that was generated when the dashboard was first loaded.

Dashboard when first loaded:



Dashboard filtered for Mountain Rescue Dogs:



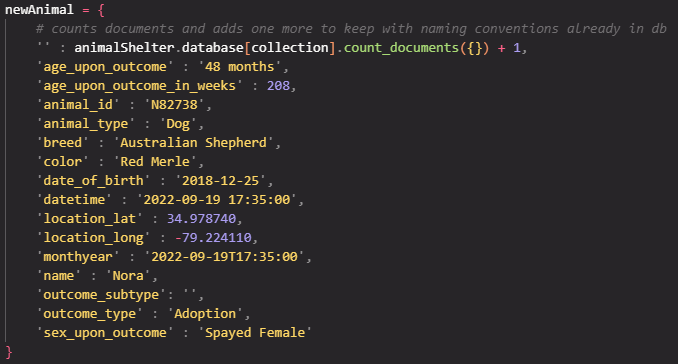
**Create New Document in MongoDB**

To create a new document in the database, use the following command:

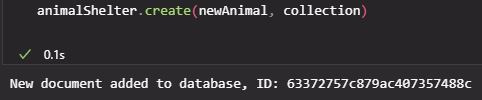
|  |
| --- |
| animalShelter.create(data, 'collectionName') |

### Where the data is a dictionary with all the document data, and collectionName is a string of the collection you want to insert the document into. The function will automatically generate a list of each field name to make sure that the database remains consistent. In my testing file I declared the variable collection set to a string with the collection name in the database I am using, so screenshots will show that variable. For your uses, make sure you either set a variable to the collection you want to use or explicitly pass the name of it in a string in function calls.

### For example, here is the sample dictionary used to add a document to the database associated with creating this file:



When the function is run with the dictionary and the name of the collection, it returns the ID generated by MongoDB upon successful insertion of the document.



If a field is missing from the dictionary, an exception is raised with the missing field. In this case, I tried to add a document with the color field removed from the dictionary.



**Read Documents in MongoDB**

There are multiple ways to read a document in the database. The readOne() function will return a dictionary of the first document’s data that matches the query. The readAll() function will return a cursor that points to the first document in the search results. You can iterate over the cursor to print out the document data, but that will consume the cursor and you will have to invoke the readAll() function again before you can look up the data.

To read one document in the database, use the following command:

|  |
| --- |
| data = animalShelter.readOne(query, 'collection') |

### Where query is the search criteria in a dictionary format, and collection is the name of the collection to search through.

### An example readOne Call is here:

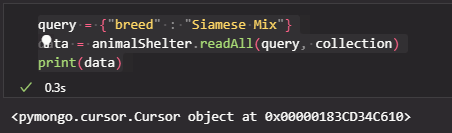
### 

### 

### To read all documents that match the query, use the readAll() function and the following command:

|  |
| --- |
| data = animalShelter.readAll(query, 'collection') |

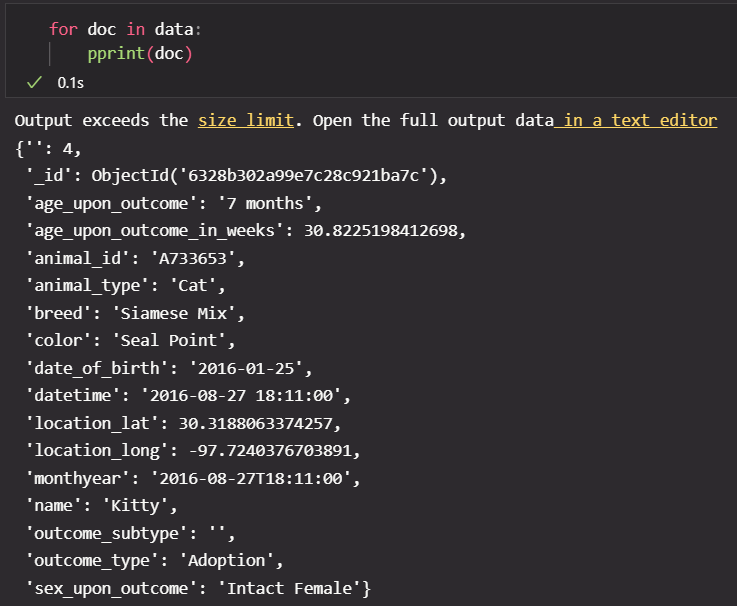
### Where query is the search criteria in a dictionary format, and collectionName is the name of the collection to search through. When a search is made it returns a cursor that points to the first document in the search results. An example readAll call is here:



To print the documents that the cursor is pointing to use the following command:

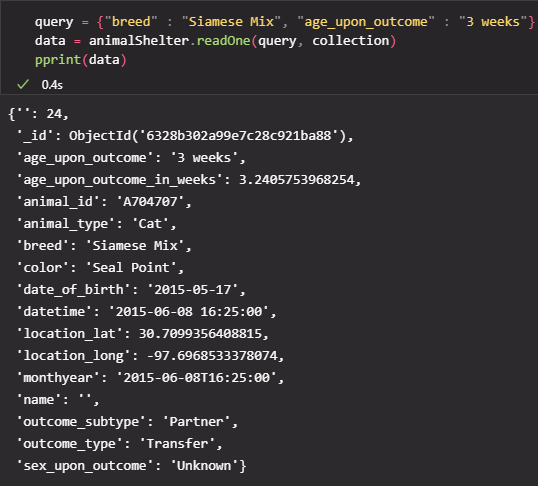
|  |
| --- |
| for doc in data:  pprint(doc) |

This will print each document as a dictionary with each key/value pair on its own line as shown here:



You can also use a regular print function, but that will not separate the fields by line, so it is harder to read.

Queries for multiple filters can also be made by adding multiple items to the query dictionary. Here is an example:



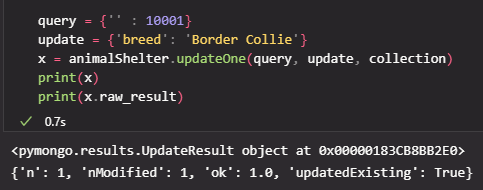
**Update Documents in MongoDB**

You can update documents in the database as well. The updateOne() function updates the first document that matches the query. The updateMany() function updates all the documents that match the query.

To update one document in the database, use the following command:

|  |
| --- |
| data = animalShelter.updateOne(query, update, 'collection') |

Where query is the search criteria in dictionary format, update is a dictionary with the keys and values to update, and collection is a string with the collection name. An example updateOne call is here:

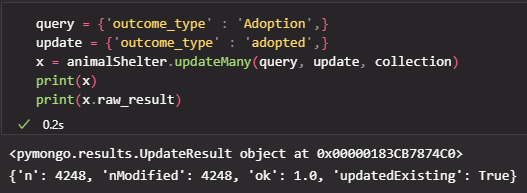


This function returns an updateResult object. You can access the data of that object using the raw\_result attribute as shown in the screenshot above. You can also access the matched\_count and modified\_count attributes to get the values in the dictionary that raw\_result returns.

To update all documents that match the query, use the updateMany function with the following command:

|  |
| --- |
| data = animalShelter.updateMany(query, update, 'collection') |

Where query is the search criteria in dictionary format, update is a dictionary with the keys and values to update, and collection is a string with the collection name. An example updateMany call is here:



This function returns an updateResult object. You can access the data of that object using the raw\_result attribute as shown in the screenshot above. You can also access the matched\_count and modified\_count attributes to get the values in the dictionary that raw\_result returns.

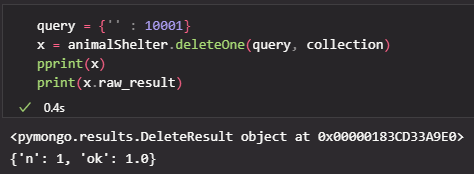
**Delete Documents in MongoDB**

To delete documents from the database, use the deleteOne() or deleteMany() functions, respectively. The deleteOne() function will delete the first document that matches the query, while the deleteMany() function will delete all the documents that match the query.

To delete one document from the database, use the following command:

|  |
| --- |
| x = animalShelter.deleteOne(query, 'collection') |

Where query is the search criteria in a dictionary format, and collectionName is the name of the collection to search through. An example deleteOne call is here:

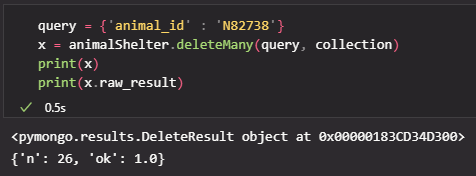


This function returns a deleteResult object. You can access the data of that object using the raw\_result attribute as shown in the screenshot above. You can also access the deleted\_count attribute to get the value in the dictionary that raw\_result returns.

To delete all documents that match the query from the database, use the following command:

|  |
| --- |
| x = animalShelter.deleteMany(query, 'collection') |

Where query is the search criteria in a dictionary format, and collectionName is the name of the collection to search through. An example deleteMany call is here:



This function returns a deleteResult object. You can access the data of that object using the raw\_result attribute as shown in the screenshot above. You can also access the deleted\_count attribute to get the value in the dictionary that raw\_result returns.

## Roadmap/Features (Optional)

*Provide an open issues list of proposed features (and known issues). If you have ideas for releases in the future, it is* *a good idea to list them in the README. What makes your project stand out?*  
  
*Note: This section is optional for the purposes of this assignment. If you choose not to fill out this section, remove it from your final README file.*

## Contact

Michael Shinkle

[Michael.shinkle@snhu.edu](mailto:Michael.shinkle@snhu.edu)