## Motivation

This project will benefit the efficiency of information sharing at Grazioso Salvare. They have five animal shelters they work out of, so having a single location where data can be sourced from can help each location easily access potential candidates from other shelters. It can also help them quickly identify their current demographics from anywhere they have access to the database. Additionally, they can easily use the database to categorize the inhabitants of the animal shelters based on qualifiers for search and rescue training to help identify and prioritize the best candidates for the training.

This project was developed to be able to integrate Mongo DB with a Python script and create a dashboard that allows end users to interact with the database without writing or reading any code. Python is a programming language that is known for handling large data sets effectively, so it is important to have an efficient way of creating, reading, updating, and destroying data documents from within a python project (as opposed to directly modifying databases from the Mongo shell).

## Project Function

This project allows end users to view Grazioso Salvare animal shelter data in table, pie chart, and geolocation form. It provides an interactive table that can be filtered to show only results that fulfil three categories for characteristics of dogs that are preferred for specialized rescue training: water rescue, mountain/wilderness rescue, and disaster or individual tracking. Each of these filters can be applied by selecting their corresponding radio button above the data table. The default setting (no applied filters) can be selected by selecting the ‘reset’ radio button. Examples of dashboard features are shown below:

A screenshot of a computer

Description automatically generated

Figure 1: Default dashboard view for Grazioso Salvare; reset is auto-selected to display all data, unfiltered

The table can be manipulated manually as well (beyond the pre-set filter buttons); columns can be filtered by typing a filter query in the first row post-header on the table, or columns can be sorted by selecting the up/down arrows in the corresponding header. Rows can be selected by selecting the radio button in the second column of the table, or they can be deleted by selecting the ‘x’ in the leftmost column. The following screenshot shows a filter query for all records with animal\_id = “A733653”. There appear to be six duplicate records; the third is selected, and its location is shown in the geolocation chart on the bottom right widget. The pie chart (bottom left widget) shows the breakdown of results by breed.

A screenshot of a computer

Description automatically generated

Figure 2: Filtered results, showing the pie chart and geolocation chart

Another way of interacting with the table is by sorting. In the following screenshot, the tables has been sorted by low-to-high for the age\_upon\_outcome column:

A screenshot of a computer

Description automatically generated

Figure 3: Sorted results, showing age-upon-outcome from low to high values

Another way of interacting with the data includes deletion of rows. The ‘x’ can be selected in the leftmost column of any row to delete it from the table. Compare Figure 3 on page 2 with Figure 4 below. The cat with rec\_num = 3637 was deleted in this way.

A screenshot of a computer

Description automatically generated

Figure 4: Table showing deleted entry removed from results

The search results and their corresponding charts for each preset filter are shown through screenshots below. The number of results per page were limited to five to allow better visibility of the charts while including the unique identifier:

A screenshot of a map

Description automatically generated

Figure 5: Results for Water Rescue filtering

A screenshot of a computer

Description automatically generated

Figure 6: Results for Mountain or Wilderness Rescue filtering

A screenshot of a map

Description automatically generatedFigure 7: Results for Disaster or Individual Tracking filtering

A screenshot of a map

Description automatically generated

Figure 8: Results for 'reset' selection, no filters applied

## Tools and Installations

Several tools were used in the creation of this dashboard. Each served a specific purpose, detailed below along with information on installation

**Tool Name:** Python

**Installation**: https://realpython.com/installing-python/  
**Reason For Use:** Python is a widely used and user-friendly programming language that use simplify access to MongoDB. It allows for easy interaction with MongoDB through libraries like PyMongo. Reading and writing Python code can be much easier than interacting with the Mongo Shell, especially for new programmers. Additionally, access of MongoDB data through this python script can allow for easy integration of MongoDB with other user applications and python scripts.

**Tool Name:** Jupyter Notebooks  
**Installation:** <https://jupyter.org/install>

**Reason For Use:** Jupyter Notebooks is a popular IDE that allows for interactive computing and data visualization. It provides a flexible and user-friendly interface that eliminates the need for users to interact directly with the Python terminal. Jupiter Notebook’s modular format for executing code in cells can reduce the computing time on large datasets by minimizing the number of lines of code that must be run throughout development.

**Tool Name:** PyMongo

**Installation:** <https://pypi.org/project/pymongo/>

**Reason For Use:** PyMongo is a Python driver for MongoDB. This gives a convenient and efficient way of interacting with MongoDB databases from Python applications. It allows developers to perform various database operations such as inserting, searching, updating, and deleting documents using Python syntax. PyMongo integrates MongoDB with Python applications, which gives the programmer a straightforward way to communicate between the two tools.

**Tool Name:** MongoDB

**Installation:** <https://www.mongodb.com/docs/manual/installation/>

**Reason For Use:** MongoDB is a no-SQL database that can store and manage large unstructured datasets. It gives a flexible and scalable schema that is easily accessible and easy to integrate with data-driven applications. MongoDB uses a query language akin to JavaScript, which makes it easy to perform complex queries. Furthermore, it has features such as upserts, which allow the user to insert new documents or update existing ones based on specified criteria depending on the existence of a queried document. It also has options like writing acknowledgements that are used to help the developer understand the behavior of database operations.

**Tool Name:** Dash (Jupyter-Dash)  
**Installation (Dash):** https://dash.plotly.com/installation   
**Installation (Jupyter-Dash):** https://pypi.org/project/jupyter-dash/  
**Installation (Dash-Leaflet):** https://www.dash-leaflet.com/  
**Reasons For Use:** Jupyter-dash was used in this case because the project was developed in a Jupyter Notebook. I will refer to it as Dash moving forward as Jupyter-Dash is simply a library that allows Dash applications to be developed within a Jupyter Notebook environment. Dash is a web application framework built on Flask, React, and Plotly that is used for creating interactive dashboards with Python syntax (Castillo, n.d.). In the case of this project, it was used because this project requires user interactions with the data table and filter selections to produce changes in the pic chart and geolocation chart. Dash handles many of the backend functions to update the dashboard without requiring the end user to reload the entire page. Additionally, using Dash instead of another popular framework, such as React, prevents the need for JavaScript code. Using Python only greatly simplifies the application. Dash also integrates with third-party libraries, such as Dash-Leaflet, which can be used to add interactive maps to dashboards.

**Tool Name:** Pandas  
**Installation:** <https://pandas.pydata.org/docs/getting_started/install.html>

**Reason For Use:** Pandas is a common Python library used for structuring and manipulating tabular data. It can easily convert a dictionary into a structure called DataFrame, which can handle filtering, sorting, and transformation functions. In some cases, the data was filtered using MongoDB queries, but columns were updated after filtering actions using pandas functionality.

## Getting Started

To get a local copy up and running, follow the following steps:

1. Download the file aacConnect.py to your local computer
2. With Mongo installed, create or access a Mongo database on your computer called “AAC”. If this naming convention does not make sense for your purposes, you may choose a different name, but then also update the DB connection variables as described in step 5.
3. To upload existing data to your database, navigate to the location of the dataset (using the cd command in your command terminal). For example:

(base) $cd /usr/local/datasets

1. Import your dataset using mongoimport. Depending on your operating system and the file type of your dataset, the way you use mongoimport may differ. Follow the instructions found here: <https://www.mongodb.com/docs/database-tools/mongoimport/>. A screenshot of this process is shown in the Usage section called “screenshots”.
2. If you don’t already have one, add a user for your database that has read and write privileges.

You can do this by using the following command within your admin database:

db.createUser({user:**”yourUsername”,** pwd: **“yourPassword”**, roles:{role:”readWrite”, db: “**AAC**”})

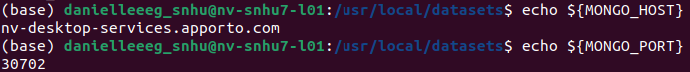
In the statement above, replace the bold text with values specific to your database name and your preferred username and password.

1. You can then login using the following command:

db.auth(‘yourUserName’, ‘yourPassword’)

A screenshot of this process is shown in the Usage section called “screenshots”.

1. To identify your host and port, run the following commands in your command terminal, at the location of your database. If these values are not returned, you may need to declare them in your environment variables.



1. Next, download the dashboard file, GraziosoSalvareDashboard.ipynb, and open in a Jupyter Notebook. Some installations may be needed in order to use all packages (see Installations section)
2. To specify the user account and database you are using, declare your username, password, host, port, database name, and the collection name when creating an AnimalShelter object. An example of this is as follows *A white box with red text

   Description automatically generated*
3. Run the cell and the path for viewing the dashboard should be returned in the output. Follow that path to view the dashboard.

## Creating the Project

Several steps needed to be taken in a particular order to complete this project; many components were built off of each other. The following steps are in order of which the project was built:

1. Import the animal shelter data into the database (AAC) in MongoDB. This supplied the database with data to work with, and is useful for testing CRUD functions later on in the project. This is done in a Linux terminal.

A computer screen with white text

Description automatically generated

Figure 9: Importing a csv file to the AAC MongoDB database

1. Create a new user that has access to read and write data in the database specified in step 1. This user information will be used later on, so save the username and password. This is done in a MongoDB terminal (accessed by calling mongosh from the OS terminal).

A screenshot of a computer

Description automatically generated

Figure 10: Creating a user for accessing the database with read/write permissions

1. Identify your host and port number by calling the following commands; if your environment variables are defined differently, update the commands to reflect the variables used for host and port. This is done in the Linux terminal (can be done before step 2)

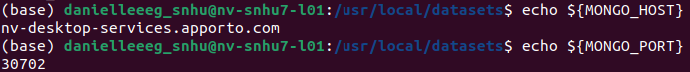


Figure 11: Accessing host and port number

1. Now its time to create the CRUD python script. To recreate this project or a similar application, modify the following code to suit your needs. The following class definition handles access to the MongoDB when user, host, port, and database information is given in the class instance declaration:

A computer screen shot of a program

Description automatically generated

Figure 12: CRUD MongoDB access

1. The next steps are to add the CRUD functionality (create, read, update, and delete functionality to the CRUD python script, started in step #4. This can be done several different ways, and be very simple in nature (using bare minimal function) to more complex (with advanced error handling). To get started with developing CRUD functionality, the following article shares some tips for using MongoDB operations in Python: <https://www.mongodb.com/developer/languages/python/python-quickstart-crud/>

To see the finished CRUD application for the Grazioso Salvare project, see the aacConnect.py file.

1. Once the CRUD file is fully developed, development of the dashboard can begin. This is where you will want to recall the information about user login, port number, and host name for creating an instance of animal shelter using the CRUD methods developed in step 5. This is shown in the following image.

A screen shot of a computer

Description automatically generated

Figure 13: Creating an instance of Animal shelter

1. Next you will want to get your initial data set by calling the read function from the CRUD script. This may look something like the following code:

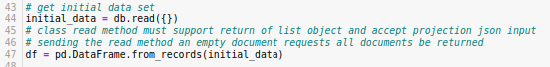


Figure 14: Getting data from database for initializing the dashboard

1. To initialize the application using Dash, you will use the following app declaration, (at the beginning of your dashboard script:

app = JupyterDash(\_\_name\_\_)

After the app declaration, you will declare the layout, and define callbacks and callback methods in later steps. In your code, after all app components have been defined, the very last line of code should be the following, which runs the application.

app.run\_server(debug=True)

1. Now in between the two lines of code described in step 8, the app layout can be defined using HTML. Callbacks and methods should be declared to handle events that happen when users interact with the dashboard. In the case of the Grazioso Salvare dashboard, an interactive datatable (with pre-built filters), a geolocation chart, and a pie chart were added to the layout, each with callbacks that allowed the user to interact with the data and produce updated charts and table results. To view these callbacks, methods, and the app layout, go to the GraziosoSalvareDashboard.ipynb script.
   1. For information on defining a layout for a dashboard, go to the following site:  
      https://dash.plotly.com/layout
   2. For information on creating callbacks and their corresponding functions, got o the following site:   
      <https://dash.plotly.com/basic-callbacks>
   3. For information on creating a geolocation chart, use the following resource:   
      https://www.dash-leaflet.com/

## Challenges:

Most of the challenges I faced were the result of not fully understanding CRUD functions of MongoDB. For example, determining the query strings for each of the specialized filters took some trial and error with creating the query dictionary. In the end, I figured out how to use $and conjunctions for requiring multiple query filters to be true in the return data. I largely depended on the documentation for PyMongo. Any questions I had about how to create or read a document in my database was answered by this article: <https://pymongo.readthedocs.io/en/stable/tutorial.html>

When it came to creating callbacks, the syntax was quite straightforward, but I struggled with it a bit because I mistakenly included an extra set of parentheses around the arguments of the callback. This created issues with updating the table in the dashboard because the callback was not expecting a tuple as an input. While troubleshooting, the following article helped me better understand writing callbacks: <https://dash.plotly.com/basic-callbacks>. I visually was not able to identify the error caused by extra parentheses, so I resolved the error by deleting the existing callback and rewriting it from scratch. On the second time writing it, I did not add the extra parentheses and the data table then filtered correctly.

# References

Castillo, D. (n.d.). *Develop Data Visualization Interfaces in Python With Dash*. Retrieved from Real Python: https://realpython.com/python-dash/#what-is-dash