# CS 340 README – Jeffrey Sparks

## About the Project/Project Title

The purpose of this application is to easily sift through the AAC database entries with ease and fluidity of filtering the information needed.

**Motivation for Using Mongo DB**

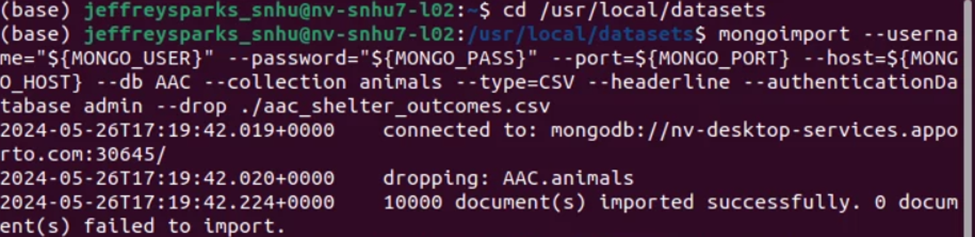
Mongo was used, specifically with this program, because it provides a quick setup of the database from a csv document as well as a Python-friendly interface. While Python can be used to some form of compatibility with tools like SQL, there is a higher level of complexity within communication between the two, the ease of queries using CRUD methodologies with python in tandem with MongoDB made it the ideal pairing for this application 1.

**Motivation for Using Dash**

Dash, as the tool used to build the dashboard, was desirable due to its dynamic nature. Dash is a react JavaScript based tool that provides an incredibly responsive framework. Dash involves html Dash tags that control outputs to segments. Then, updates to any of the target inputs specified in the app callbacks process based on instructions programmed in the Python module.

**Getting Started**

To create a local copy of this program, you will need several steps. The first is to create a Mongo Database. The second is to program a Python CRUD module to access that database. The final step is to create a Dash web application that implements the Python CRUD module. This is a multi-layered application that reacts to changes in the table in real time.

1. Create a Mongo Database and create a database called AAC.
2. Create a user with read/write privileges to that AAC database.
3. Import the data from aac\_shelter\_outcomes.csv file
   1. The csv file has some issues. I overcame the import issue by passing my user and password for the AAC database in the import command in the Linux terminal.
   2. Some people were able to import the AAC csv file by running Mongo via -noauth but I was unable to. If you have the same issue – use step a.
   3. Update the port number (Mongo gives you the port number when you start the service) on the file import.
   4. Example import: 
   5. Login as “aacuser” or as “admin”
4. Update the port number on localhost of the AnimalShelter.py Python code.
   1. This will be essential when importing the
5. Update the “aacuser” and “SNHU1234” to the username and password you created.
   1. Sample Initialization: A computer screen shot of a computer code

      Description automatically generated
6. To run the tests, add the test code to a Jupyter notebook and ensure that the test data for the create function is different each time or delete the added record between tests.
7. Create a new Dash web application dashboard and configure with the desired HTML/CSS layout and appropriate ids for the data frame, map, and chart.
8. Create an app callback to populate the initial data frame with all of the data.
9. Create radial options and program the database queries based on the desired breed specifications for the client.
10. Create an application callback to update the map with the first item of a given category until the user selects an item. Then, create functionality that determines the user selection and displays that on the map instead.
11. Create a pie chart from the displayed data on the screen at any given moment. With the pie chart in hand, create an application callback that updates the pie chart with the data that is filtered from the display filters.

## 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 has three main functions, Firstly, using radial buttons to sort the data based on specific characteristics, such as breed, age, and sex characteristics for the different types of rescues desired by the company Grazioso Salvare. Selecting the radio buttons runs a database query and returns an updated data frame with the desired information received as the output. The “Reset” radio button then un-filters and returns the data table to the initial unfiltered state. Secondly, the mapping feature allows you to select up to five animals within the data frame. Once the user selects the first initial row (Or up to 5 rows at one time) it will then use the associated longitudinal and latitudinal coordinates stored within the database information and output them as a point on a generated map within the dashboard. The third function is a dynamic pie chart that sords data by breed, based upon the information of the breeds of animals currently shown.

**Reset (Show all Data)**

A white paper with red text

Description automatically generated

**Water Rescue**

**A screenshot of a dashboard

Description automatically generated**

**Mountain Rescue**

**A screenshot of a dashboard

Description automatically generated**

**Disaster Rescue**

**A close-up of a map

Description automatically generated**

**Multi Select**

**A screenshot of a computer

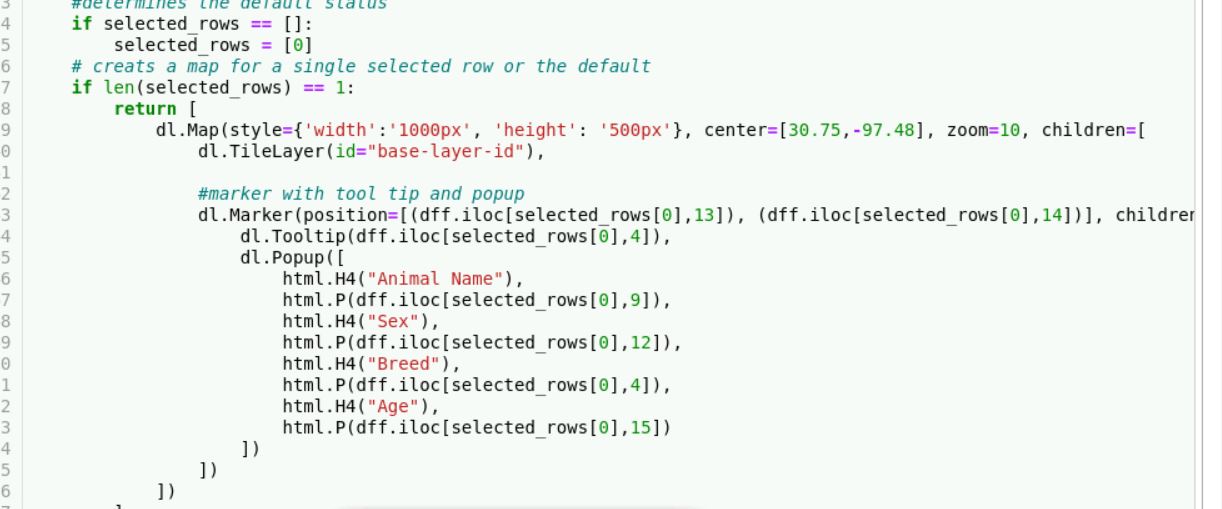
Description automatically generated**

**Code Samples from Dashboard**

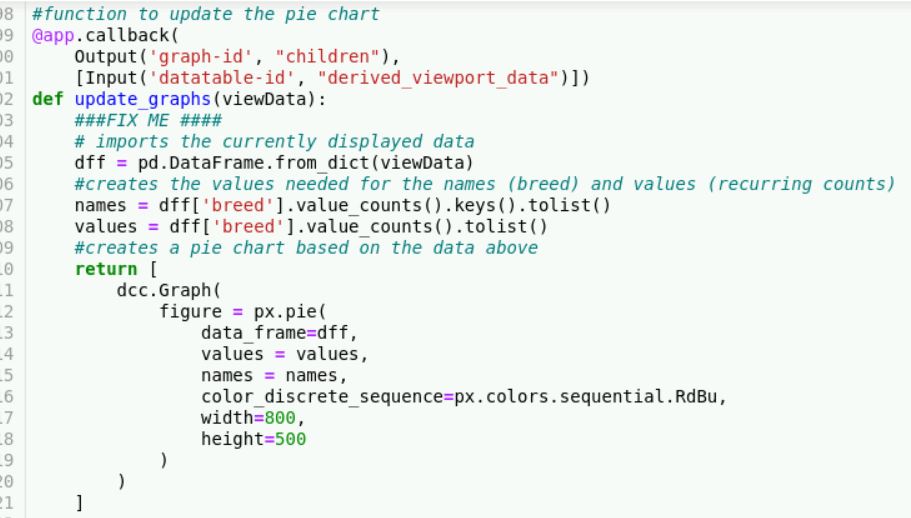
*Complex Query*

**

*Map Markers*

**

*Pie Chart*

**

### Code Example (CRUD Operations)

*def create(self, data):*

*if data is not None:*

*insert = self.database.animals.insert\_one(data)*

*if insert != 0:*

*return True*

*print('True')*

*else:*

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

*#Implement read method*

*def read(self, data):*

*if data is not None:*

*return self.database.animals.find(data)*

*else:*

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

*#Get records with criteria*

*#All records are returned if criteria is None*

*#Default is None*

*#Example: ({""name": "Rex", 'age\_upon\_outcome': '2 months'})*

*#do not return the \_id*

*def getRecordCriteria(self, criteria):*

*if criteria:*

*\_data = self.database.animals.find(criteria, {'\_id' : 0})*

*else:*

*\_data = self.database.animals.find({},{'\_id' : 0})*

*return \_data*

*#Implement update method*

*def update(self, initial, data):*

*if initial is not None:*

*if self.database.animals.count\_documents(initial, limit = 1) != 0:*

*update\_result = self.database.animals.update\_many(initial,{"$set":data})*

*result = update\_result.raw\_result*

*return result*

*else:*

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

*#Implement delete method*

*def delete(self, data):*

*if data is not None:*

*if self.database.animals.count\_documents(data, limit = 1) != 0:*

*delete\_result = self.database.animals.delete\_many(data)*

*result = delete\_result.raw\_result*

*return result*

*else:*

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

**Code Examples (Dashboard)**

*Radial Buttons*

Import view data

If radial\_one is selected:

Run complex query

Update view data

Return view data

*Map*

Import view data

If selected\_rows is None:

Selected\_rows = 0

If length(selected\_rows) == 1:

Return map with 1 map marker and tooltip

Else if length(selected\_rows) == 2:

Return map with 2 map markers and tooltips

Etc.

*Pie Chart*

Import view data

Names = data(breed) – obtain and transfer to list

Values = data(breed) – obtain occurrence counts and transfer to list

Return graph using the view data, names as the search key, and values as the pie slice values

## Contact

Jeffrey Sparks Jeffrey.sparks2@snhu.edu