# AnimalMaintenance README

## Animal Maintenance

Animal Maintenance is a Python class responsible for creating, reading, updating, and deleting records within a mongo database.

## Motivation

Animal Maintenance hopes to provide an easy way to integrate data within a database backed by Mongo DB. It acts an interface to connect and perform functions that a user would perform when maintaining records for an animal shelter.

## Getting Started

In order to work with AnimalMaintenance.py, you’ll need to have a username and password provided. Contact your administrator to have these provided for you, or to have a user created for your needs.

**Importing Data**

An initial dataset can be imported into an existing MongoDB connection. In order to use

AnimalMaintenace.py, you should create the *AAC*database and the *animals* collection from the

CSV file *aac\_shelter\_outcomes.csv*. An example of a successful import of this dataset can be

seen here:

Graphical user interface, text

Description automatically generated

**User Accounts**

It is recommended to have an Administrator account and user account ready when working with

the datasets provided here. Examples of the Admin and *aacuser* account can be seen here. (It is

important to note that the scope of the non-admin user is **strongly** recommend being limited to

the databases they will be interacting with)

Admin:

Text

Description automatically generated

User account named ‘*aacuser*’:

Text

Description automatically generated

## Installation

AnimalMaintenance.py was created and developed using PyCharm Community Edition version 2020.3. This can also be used to work with the script since it can be imported as a module. PyCharm can be downloaded here <https://www.jetbrains.com/pycharm/download/>

AnimalMaintenance.py can also be used alongside Jupyter notebook. More information on Jupyter can be found here <https://jupyter.org/>

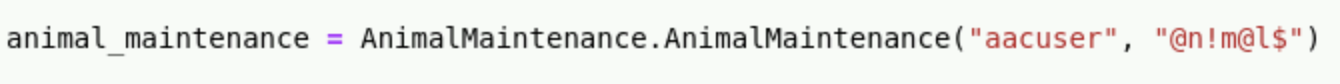
## Usage

I’ll go over how to work with the Animal Maintenance class by importing as a module using Jupyter notebook.

Create a new .ipynb file and import the script AnimalMaintenance.py in a directory of your choosing. Use the import command shown to import the script into your .ipynb file  
*A picture containing text

Description automatically generated*

### Tests

To instantiate the object, create a variable and provide the username and password provided by your administrator   


**Create**

In order to use the *create* function, you must create a data set with the respective values. This must be a Key/Value dictionary data set. An example of a data set can be seen here:

Text

Description automatically generated

**A picture containing text

Description automatically generated**

Upon creating the data set, call the .create() method passing in the data set created. You can also choose to display a result to determine if the addition was successful.

A picture containing text

Description automatically generated

**Read**

To use the .read() function, provide a key/value dataset to base your query on and pass it to the function accordingly. If multiple results are expected, iterate over the result sets using a loop as seen here:

Graphical user interface, text, application

Description automatically generated

**Update**

To use the .update() function, you can re-use the existing query in the find\_data to update the same document, or you can provide a new key/value pair to base your query on. You must also create a second key/value data set to provide details to update the desired document. In this example, we will change the name of the adopted pet to it’s new name ‘Toad’:

A picture containing diagram

Description automatically generated

The print statement here is used to show that the result set is a formatted JSON object to show details on the result of executing PyMongo’s update\_one function.

Text

Description automatically generated

**Delete**

To use the .delete() function, you can re-use the same key/value dataset used in the read and update functions, or you can create a new key/value dataset to specify the document to delete. This function uses PyMongo’s delete\_one function to remove the specified document. It’s returns a formatted JSON message to display information on if the deletion was successful or not.

Chart

Description automatically generated

**Using AnimalMaintenance with a Dashboard**

In this section, we will cover how to implement *Animal Maintenance* within a Dashboard for data visualization. The dashboard created will be writing in Jupyter Notebook and will use the following imports for it’s functionality.



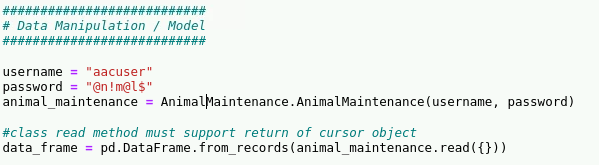
**Initialize AnimalMaintenance**

The first route will be to instantiate a new instance of the *AnimalMaintenance* module. This can

be done in a similar fashion to the CRUD application described above. In this step, we will also

create a new object using the *pandas* module in order to dynamically populate our data

elements.



**Prepare an embedded image.**

Using the *base64*module, we are able to provide a .png image and embed it within our html using the *dash\_html\_components­* module. This action prepares the image file for its insertion to the data table which we will create in the next step.

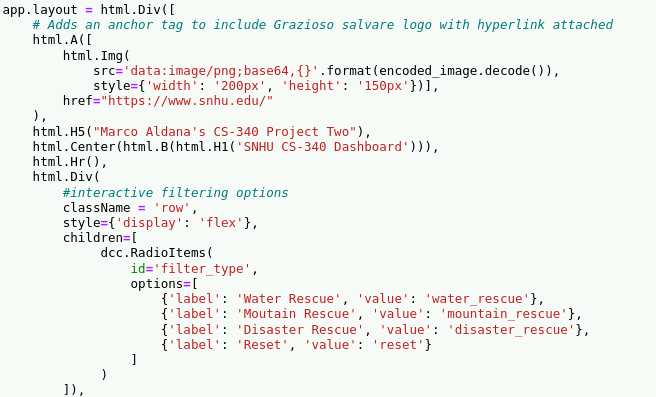


**Create a Dynamic Data Table.**

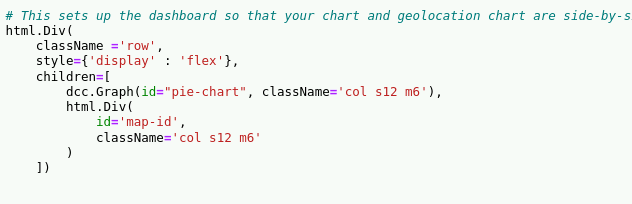
In this section, we will create a data table based on the *data\_frame* object we created earlier.

This data table will allow for dynamic data filtering based on a user selected radio-button. The

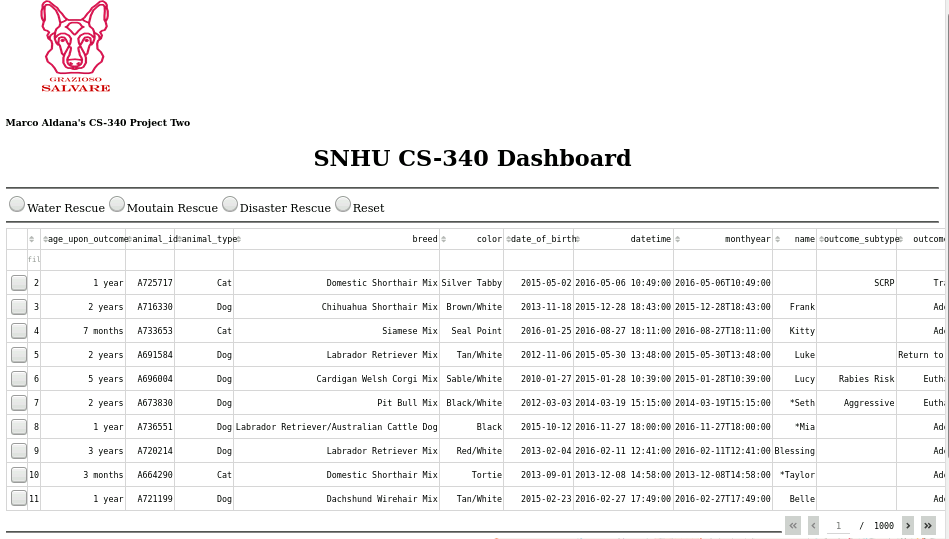
selection of the radio button will drive the contents of the table, and will also allow a reset function to set the table back to its original state.







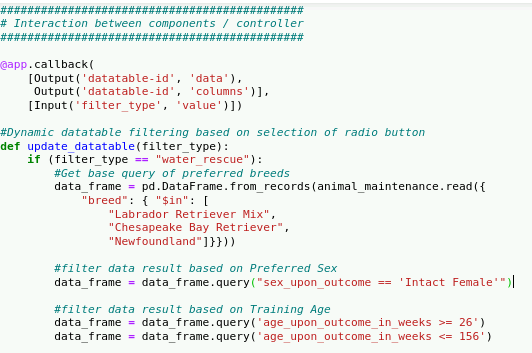
If implemented correctly, the data-table should look like the following in its initial creation.



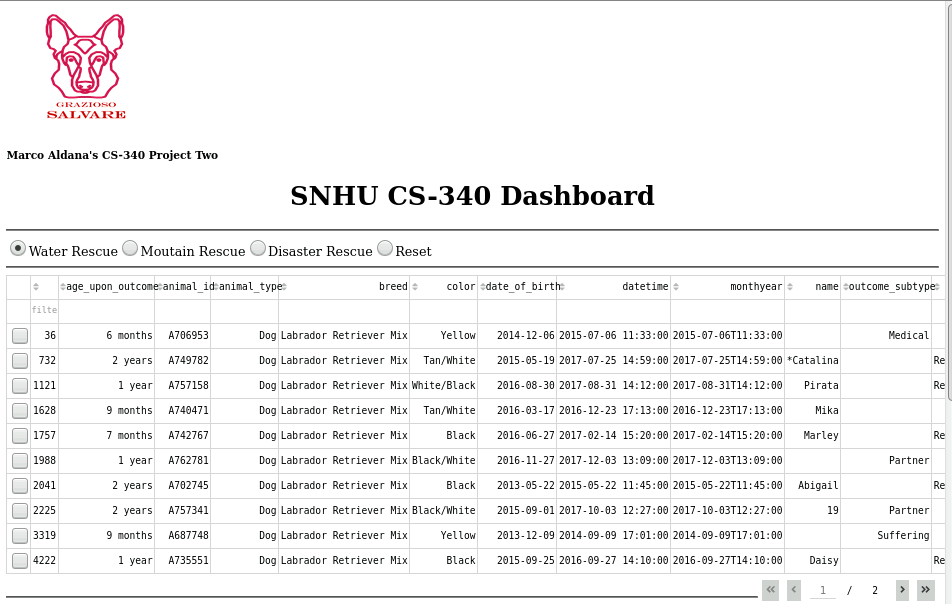
**Dynamically update the contents of our data-table**

If a user wants to quickly access a certain result set, the user can select one of the provided radio buttons to filter the data to a much more limited response. Each radio button here executes a query that calls out to our MongoDB collection and updates the data set based on the response. An example of how to implement the query filtering can be seen here. **Note:** It is important to realize that these query parameters are entirely based on the user requirements. When implementing, you can feel free to adjust the query parameters to whatever you see fit.

**Example: Adding filtering on the Water Rescue Radio button**



**Example: Implementation of filtering data**



**Adding Charts and Geographic locations to your Dash**

In the examples above which covered how to create a data table, some code was presented which creates elements to which we can dynamically generate charts and geographic location maps based on the data provided!

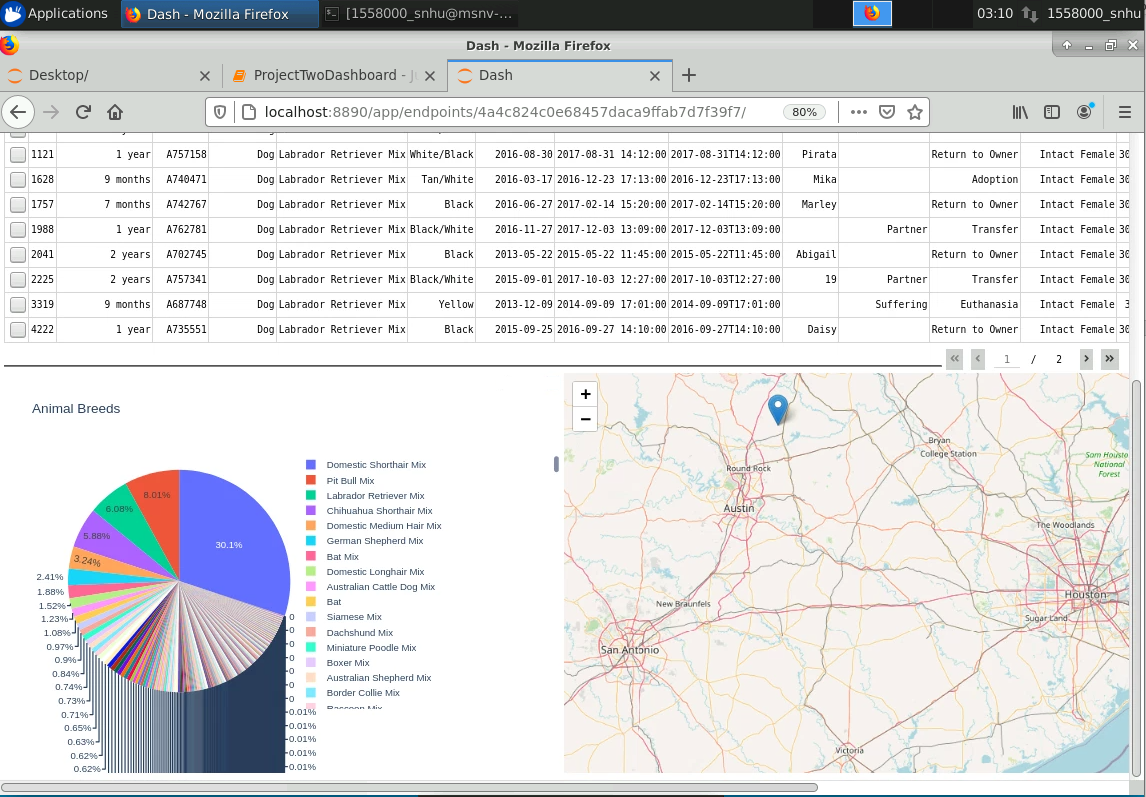
The below snippet creates callback functions to dynamically generate a Pie-Chart and Geolocation map. The following snippet will show what the implementation is in practice.

**Note:** The following examples can be tailored however you see fit. The below snippets are simply an example of what can be achieved. There are many different graphs that can be used within this dashboard.

**Example: Creating callbacks to generate Pie-Chart and Geolocation**



**Example: Implementation of Pie-Chart and Geolocation map**



## Contact

Questions? Contact Marco Aldana at marco.aldana@snhu.edu