## About Rescue Animal Finder

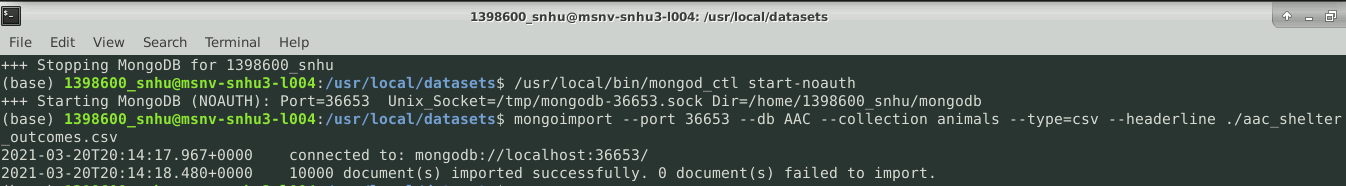
The program is a web application that contains a client dashboard for finding the best rescue animals for various environments. The dashboard allows users to search a database of animals from five different shelters. Users can filter their searches for animals that are ideal for water rescues, mountain rescues, and disaster rescues. The board lists the animals, locates them on a geolocation chart, and displays a pie chart of the different dog breeds. The dashboard incorporates a module that performs CRUD operations on the database.

## Finding Animals to Train

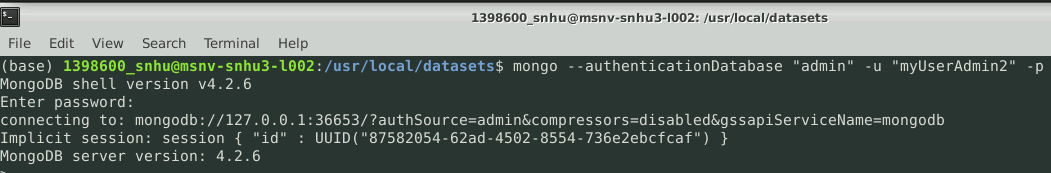
The web application is designed for the company Grazioso Salvare that trains rescue-animals. They needed a better way to find effective rescue animals such as dogs two years old or younger. They also needed to find types of dogs that excel in various environments like mountains and water. The web application allows the company to search a database of animals from five local shelters for the best rescue candidates that suite their needs.

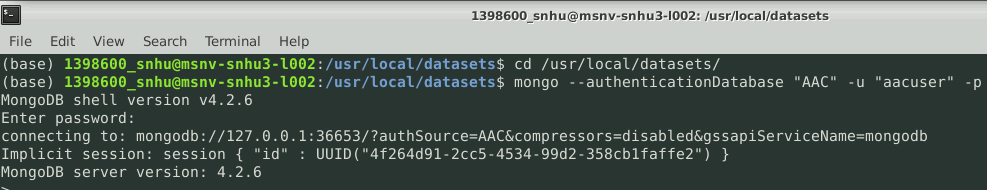
## Getting Started

Start with a pull request using an IDE that supports the Python interpreter. Install the MongoDB database and import the “animals” collection using: mongoimport - -port 36653 - -db AAC - -collection animals - -type=csv - -headerline ./aac\_shelter\_outcomes.csv.



The dashboard and python module use an IPYNB file to connect to the database. The AAC database uses MongoDB which uses collections of documents instead of tables. The database authenticates with the username “aacuser” and is password protected.





It is authorized in the \_init\_() method of the project.

To complete the project, the read({}) method from the Python module is used to get all the documents from the database. The only challenge I ran into with that was that I accidently deleted all documents from the AAC database. It took me a while to realize code worked the whole time, but it had nothing to read. A geolocation map function was added that takes the longitude and latitude of the animal from the first document read. This part was very straight forward and did not raise any challenges. Four radio buttons were added that use if statements corresponding with each radio button value. Every condition uses different attributes in the PyMongo find() method to filter only specific types of dogs for each button. The challenge was researching how to set up radio buttons in the layout. The logic was easy but finding the syntax was the tricky part. The graph uses a callback method that takes in the breed of dogs read from the database and displays a pie chart based on breeds. Similar to the radio buttons, the only challenge was finding the syntax to develop a pie chart in the callback method.

The create() method uses an insert() method to create a document if the appropriate parameters are used. The challenge with the method was authorizing the database when calling create(). This was overcome by specifying which database to authorize in the \_init\_() method. The read() method uses the find() method to search for a document matching read()’s parameters. The challenge was displaying the content of the document. This hurtle was overcome by using a loop to print each line of the document instead of the variable representing the document. The update() and delete() methods were very straight forwards as the only issues that arose were worked out in the first two methods.

## Installation

The project uses a variety of tools, languages and libraries:

* The program is developed in Python 3. Python is a flexible language that is very quick to code in. It uses a powerful interpreter program that requires less specification from the programmer. <https://www.python.org/download/releases/3.0/>
* The database uses the MongoDB platform that organizes data into collections of documents. The structure is easy to use, has great scaling potential, and is very fast with its indexing capabilities. https://www.mongodb.com/2
* The PyMongo library will need to be installed in the project. This driver allows the Python to use commands familiar to MongoDB such as insert(), find(), update(), and remove(). It translates Python dictionaries into MongoDB dictionaries and vice versa. https://pypi.org/project/pymongo/
* The Jupyter Notebook tool is used to run the Python module and connect to the database. This tool is great for running the software in browsers giving users database access from devices with internet access. https://jupyter.org/
* The dashboard uses the Jupyter-dash interface. This allows Jupyter to display widgets for users to interact with. The library uses a layout section written in HTML to specify how the view of the dashboard will look. The library uses callback sections written in Python that corresponds with a method. As users interact with widgets displayed in the layout, the callback method takes in the new inputs and returns an output that updates the dashboard. https://pypi.org/project/jupyter-dash/

## How to Use Rescue Animal Finder

To use the project, select a filter to look through dogs that can best be trained for various environments. This will run the CRUDDatabase modules read() function to pull up animals. To perform CRUD operations on the database run the CRUDDatabase Python module from the AnimalShelter IPYNB file. The new\_animal dictionary can be used to create a document for the AAC database. The create(new\_animal) method uses the variable to add the document in the “animals” collection. The create() method checks for valid parameters then passes them to insert() method. The parameters for the read() method should specify details from a document to look up. The full document will be retrieved and displayed in the AnimalShelter file. The read() method checks for valid parameters, then passes them to the find() method. The Update() method takes two parameters, one identifies the document segment to update, the other takes in the content to update it with. The update() method checks for valid parameters, then passes them to a different update() method. The delete() function can be use to remove any documents that are no longer relevant. The delete() function checks for valid parameters, then passes them to the remove() function.

The methods all use the PyMongo driver to perform their various operations. PyMongo was selected as it contains Python commands that translate to corresponding MongoDB commands. The driver translates PyMongo queries to Python dictionaries and vice versa. This allows for the database abilities of MongoDB and the flexibility of Python that will be needed for the dashboard interface.

### The Dashboard

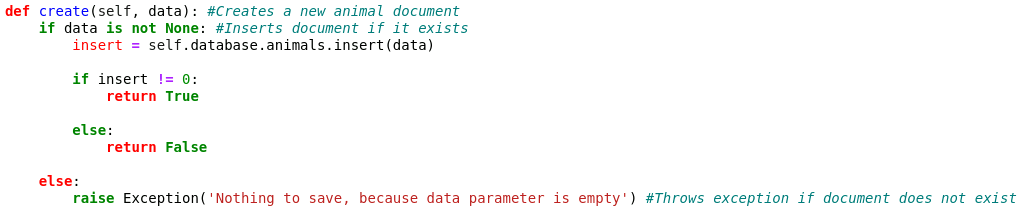
### The dashboard uses radio buttons that allows the user to filter dogs best for water rescues, mountain rescues, disaster rescues, and a reset for all dogs: It desplays a pie chart that breaks down the percentages of animals available by breed:

### The dashboard also desplays a geomap next to the chart that shows the location of the first animal:

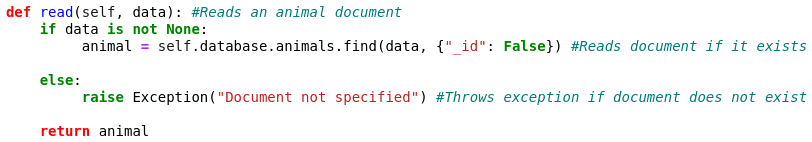
### The CRUDDatabase Library

The library contains an AnimalShelter() class containing CRUD functions for the AAC database. The AnimalShelter() contains a initiate method that connects to a MongoDB database and authorizes an admin account for the ACC database:

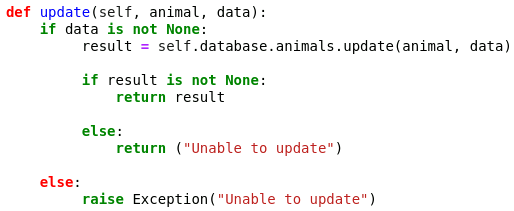
AnimalShelter() contains a create method that adds documents or throws an exception if parameters are empty:



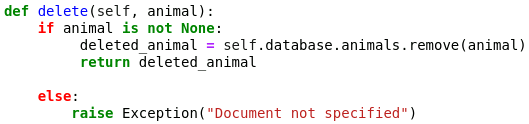
AnimalShelter() contains a read method that retrieves documents or throws an exception if parameters are empty:



AnimalShelter() contains an update method that updates documents or throws an exception if parameters are empty:

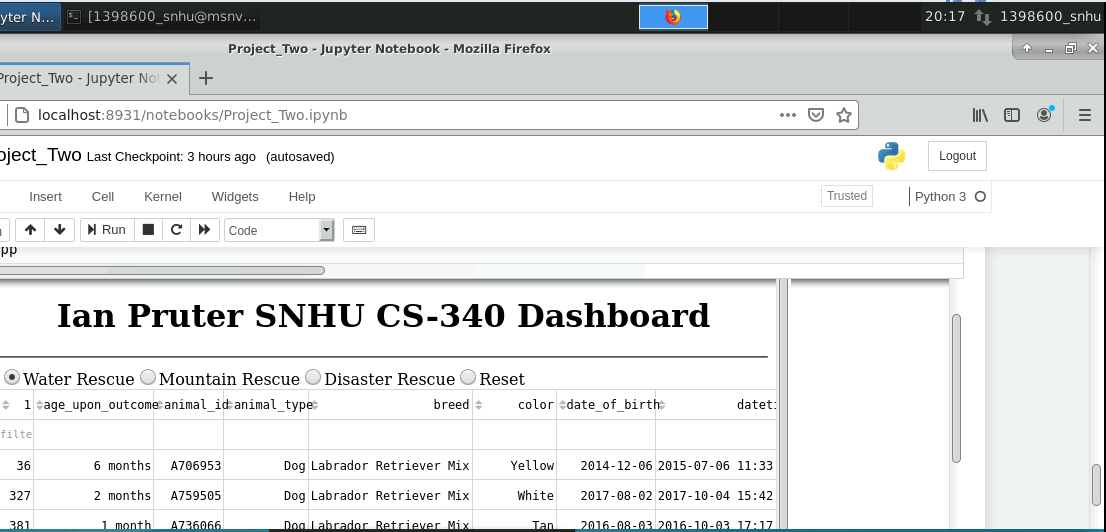


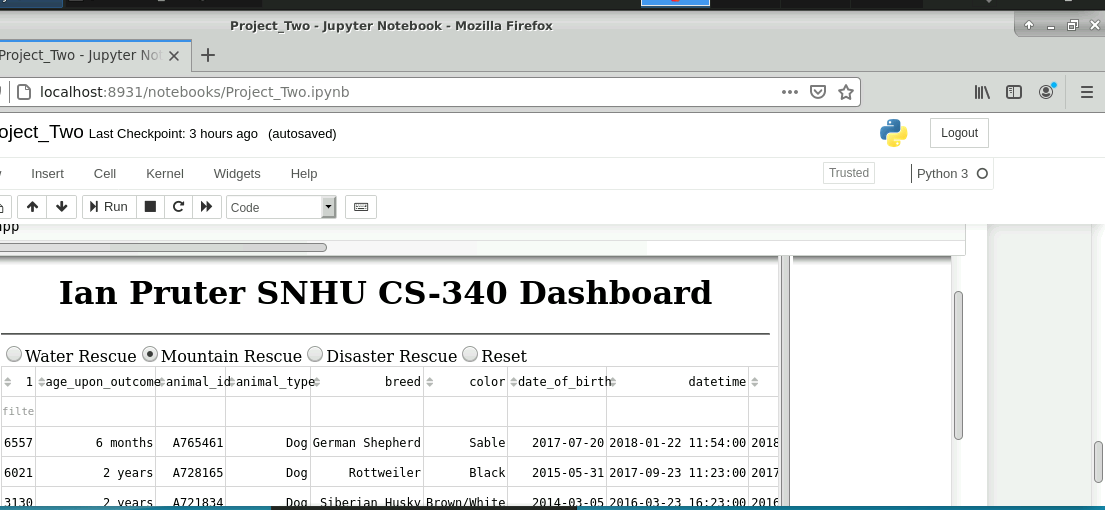
AnimalShelter() contains a delete method that removes documents or throws an exception if parameters are empty:

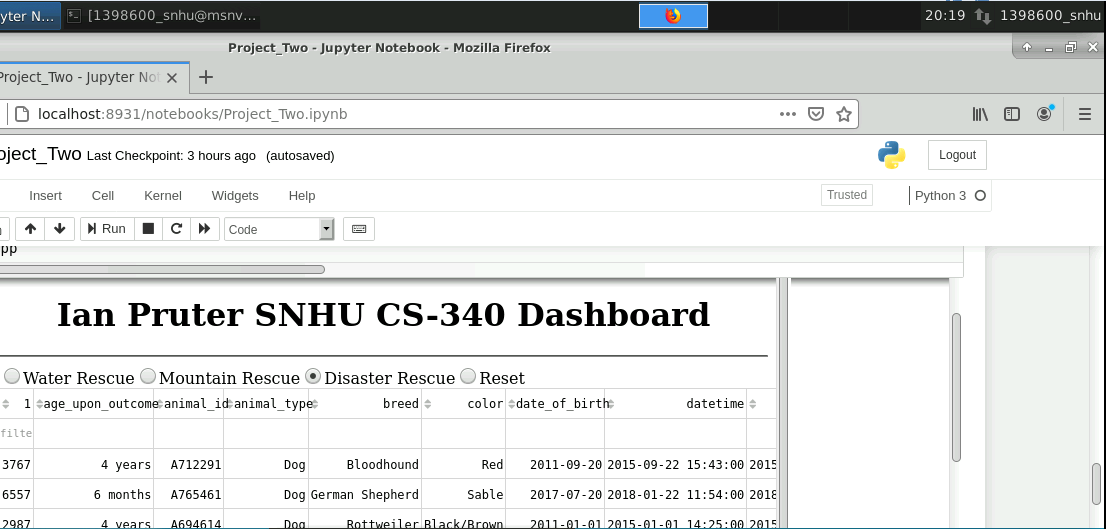


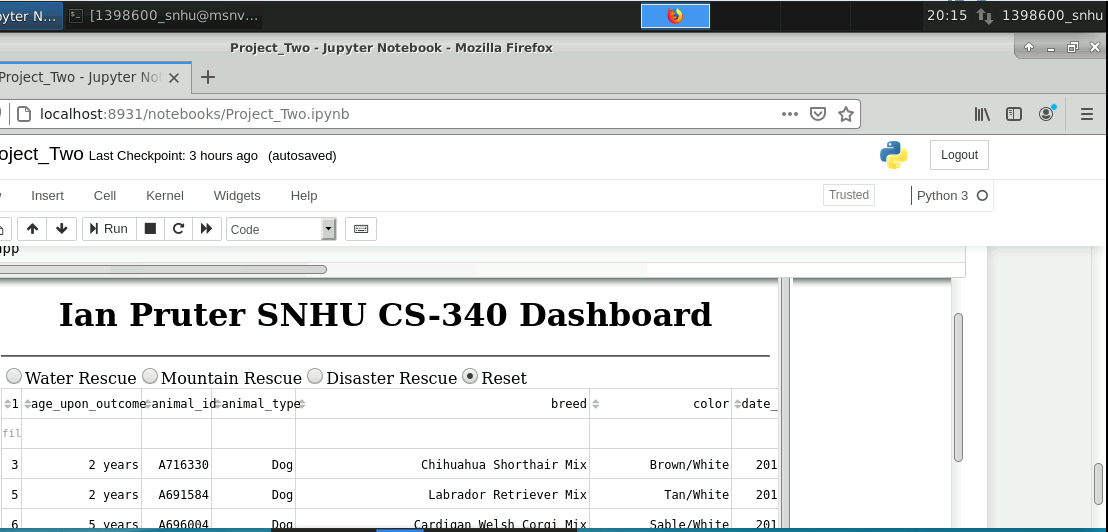
### How To Test It

Check each radio button and make sure the lists of animals change accordingly:

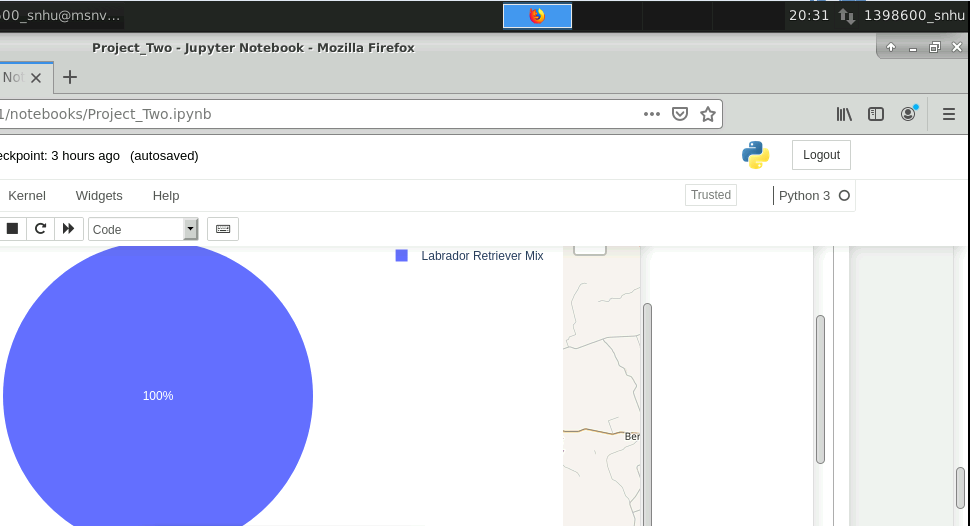


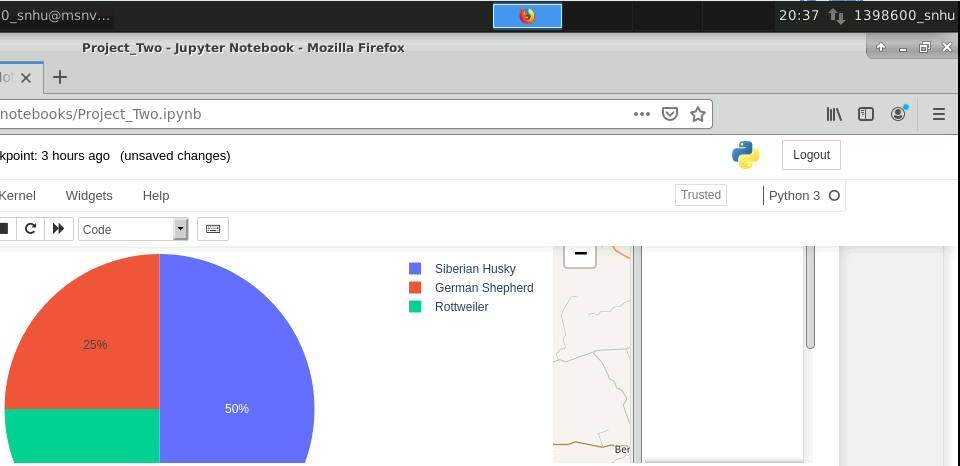


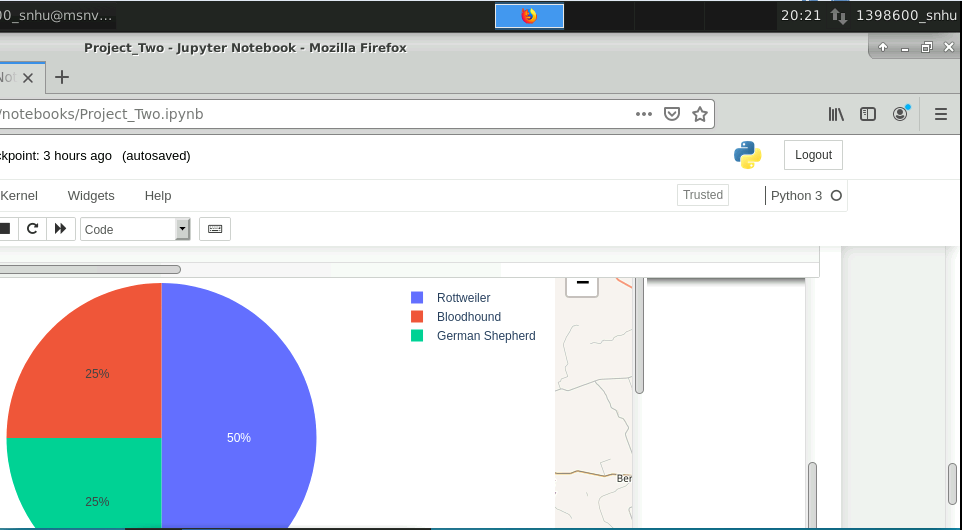


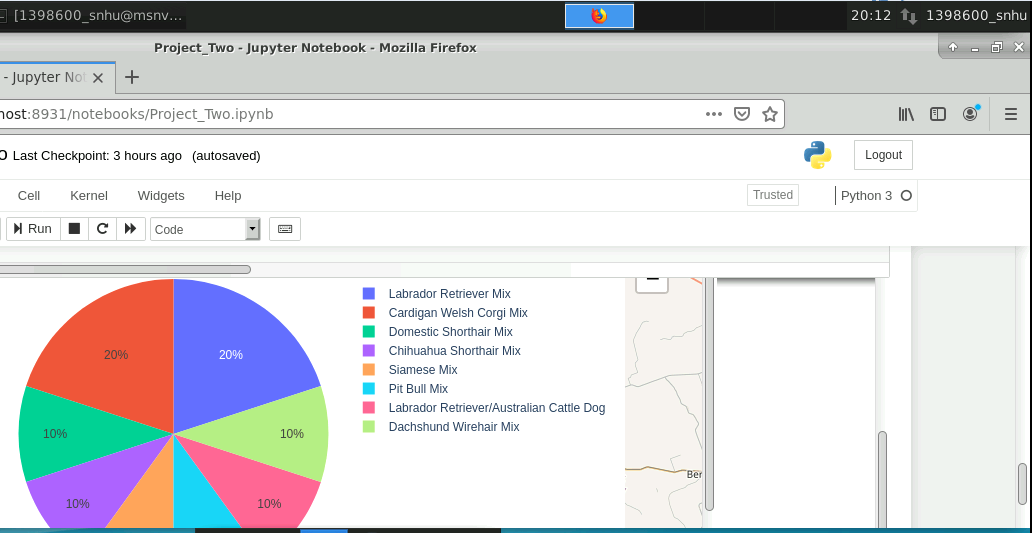


Test each chart to ensure it corresponds with the animal specifications best suited for the varies rescue types:

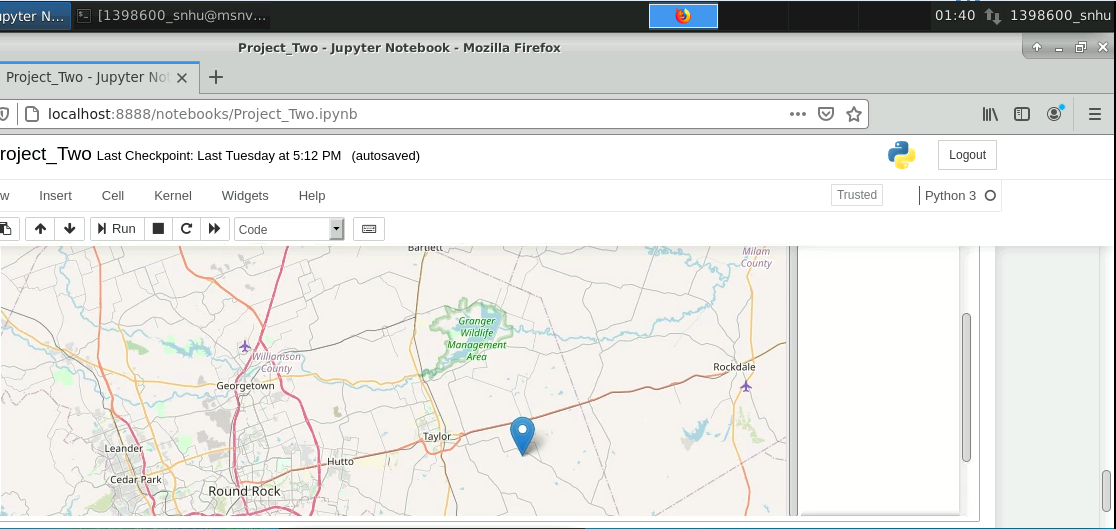


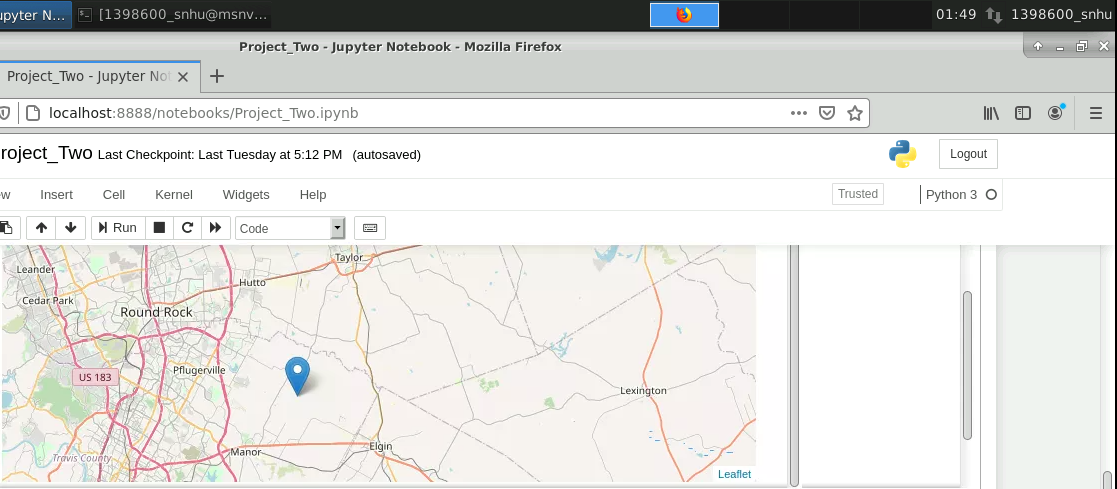


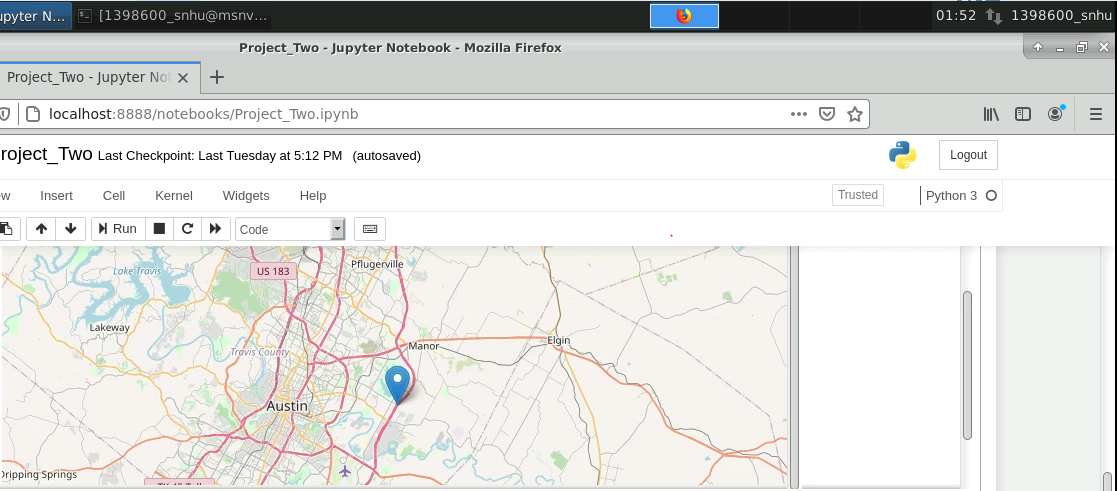


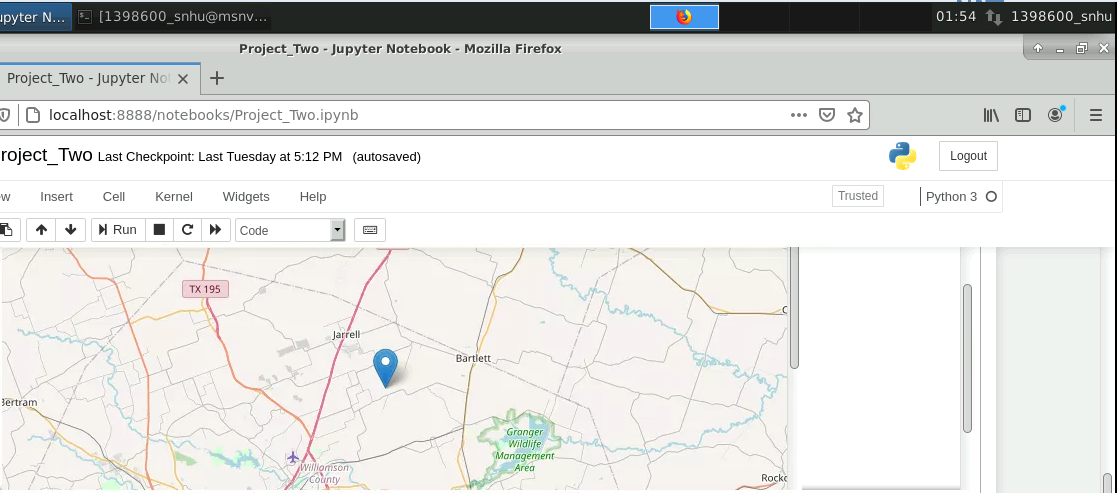


Test each geo map to ensure it corresponds with the animal in the first row:



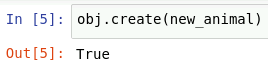




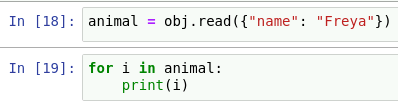


To confirm the AnimalShelter() methods worked, create(), read(), update(), and delete() were tested.

The create() method was first tested by creating a specific document without error:

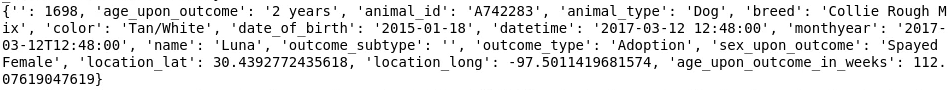


That same document was then read from the database and printed without error:



The document was edited and then printed out without error:

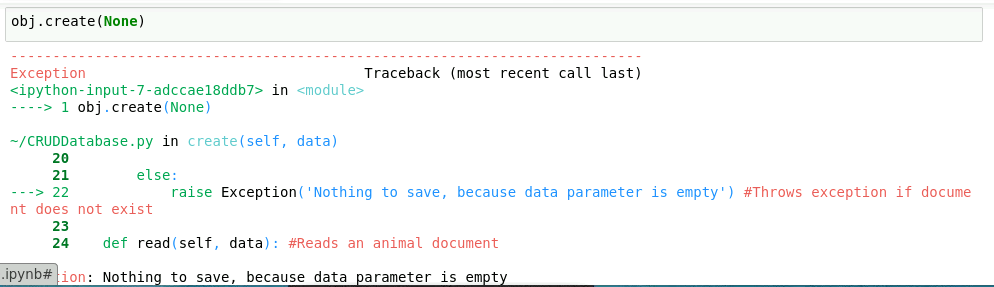


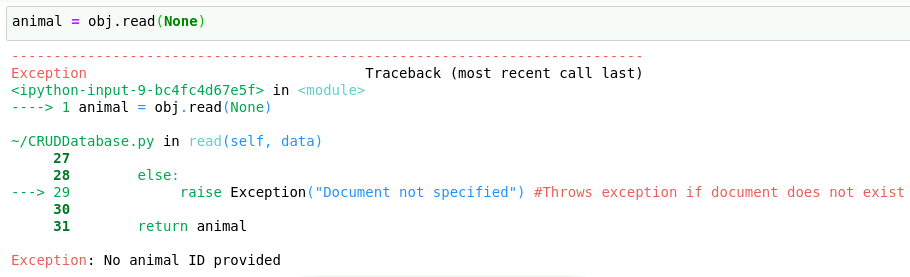


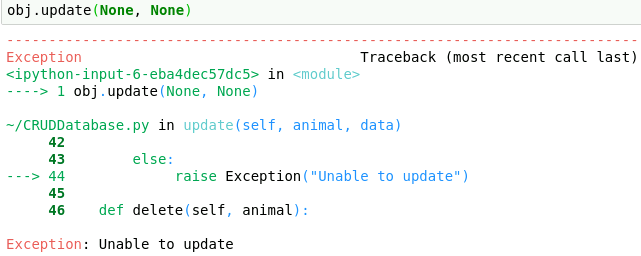
The document was then deleted and printed to demonstrate there was nothing to display:

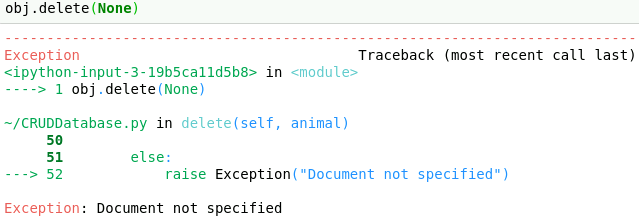


The exceptions were tested as well to ensure the program handled unexpected input by replacing the original test document parameters with “None”:



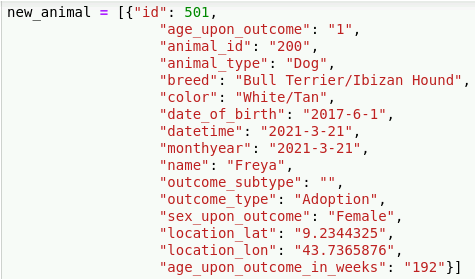




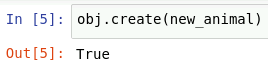


### Example Images

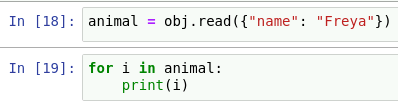
The new\_animal dictionary contains the content of the new animal document:

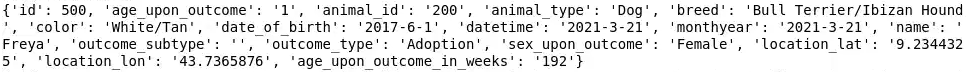


The create() method uses new\_animal to insert the data into a new document:



The read() method takes in details of a document and finds it in the “animals” collections. It then prints out the document line by line:





The update() method takes in a line to edit and the content to replace it with. It then returns the results in JSON format:



The delete() method takes in an identifying parameter and removes any documents matching it:



## Contact

Ian Pruter

ian.pruter@snhu.edu