# CS 340 Client/Server Development

***7-2 Project Two : Animal Shelter Dashboard screenshots.***

**Required functionality of this Project.**

This project is to develop a software application that can work with existing data from the animal shelters to identify and categorize available dogs. Existing data refers to a database I created with established successful CRUD routines in Python for MongoDB.

The goal per the Dashboard Specification Documents (which is the requirement(s) for this Project Two), is to complete the development of this project by coding the dashboard and the database interface logic. This will include dashboard attributes. It will also include a database and a client-facing web application dashboard through which users will access the database. The dashboard will have three main features:

1. A data table including all the pets’ information.
2. A pie chart that shows the different breeds of pets inside the table.
3. A map indicating the pet's location.

**Purpose**

The purpose of the project is to simplify the process of searching a database and giving relevant information to customers directly.

Below is an illustration of the Animal App MongoDB interface. Instructions asked that it just offer the functionality of reading and writing, which was quite challenging to test given the multiple pieces of data. I wrote a quick UI (This will be replaced upon the front end being built) to supply data for searches and for writing. Functionality was confirmed with both the sample data provided for each as well as the user input data.

**Rationale for Using Mongo DB**

MongoDB was specifically used for this application because it makes the database setup process from a csv document as well as a Python-friendly interface fairly quick and easy. Mongo also provides great scalability and querying system which when you consider how Python uses database tools like SQL, with of course different syntax, it can be cumbersome to switch back and forth. It is clear that the selection tools when running the basic CRUD functions of a database are significantly more complex in a SQL database than they are with a Mongo DB, using Python.

**Rationale for Using Dash**

Dash, is the tool used to build the dashboard. It was best for this project because it eliminated the need to learn HTML, CSS, and JavaScript in order to create an interactive dashboard. It is dynamic nature; hence, the application build using this framework can be viewed on a webbrowser. Dash is a react JavaScript based tool that provides an incredibly responsive framework.

**Project Completion Process**

To create a local copy of this program:

* First, you have to create a Mongo Database if one does not already exist.
* Secondly, you have to create or ensure accesses/privileges are assigned correctly and are active.
* The third is to program a Python CRUD module that will allow access to the database.
* Finally, create a MongoDB Dashboard or Dash web application that implements the Python CRUD functionality module by doing the following:
* Create a Mongo Database and create a database called AAC.
* Create a user with read/write privileges to that AAC database.
* Import the data from aac\_shelter\_outcomes.csv file
* In the event the csv file fails to import using assigned user and password for the AAC database. You may temporarily have to pass your username and password in the import command in the Linux terminal.
* Update the port number (Mongo provides you the port number when you start the service) on the file import.

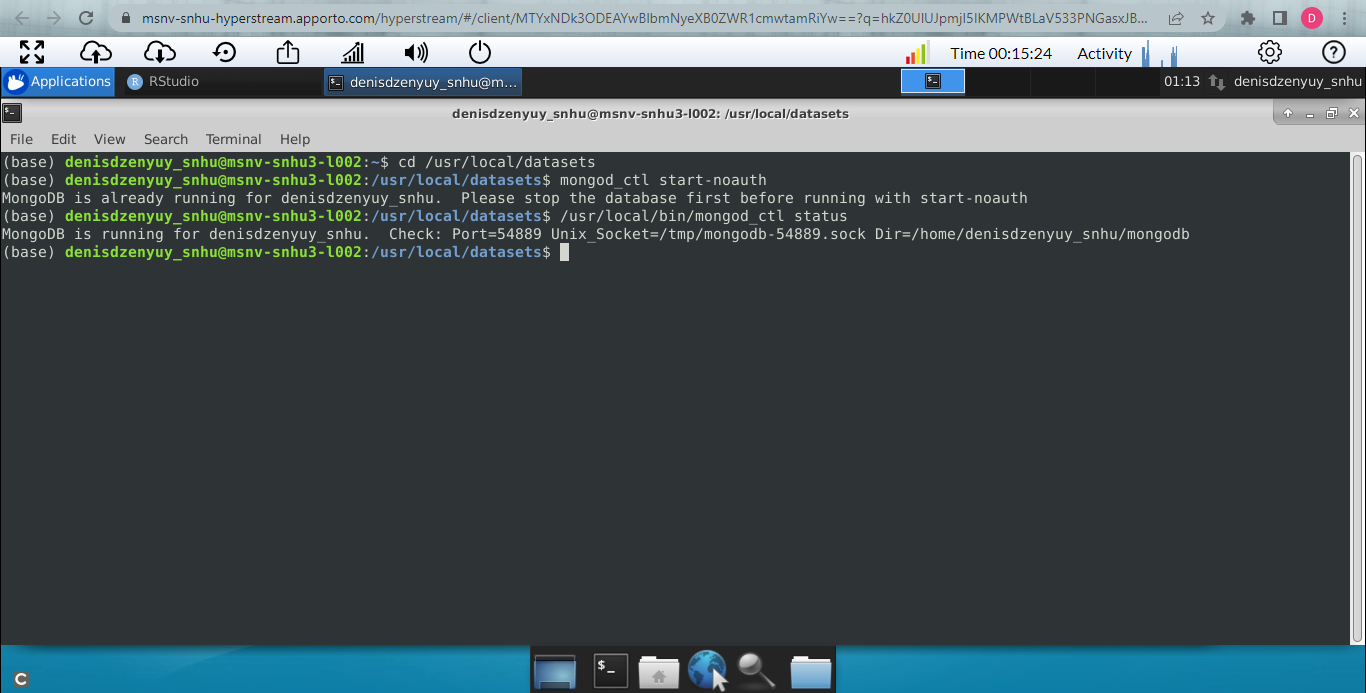
Example steps to complete the above import prompt:

* Using #start mongo without authentication

**/usr/local/bin/mongod\_ctl start-noauth**

Verifying and copying the port number given in the output:

**/usr/local/bin/mongod\_ctl status**

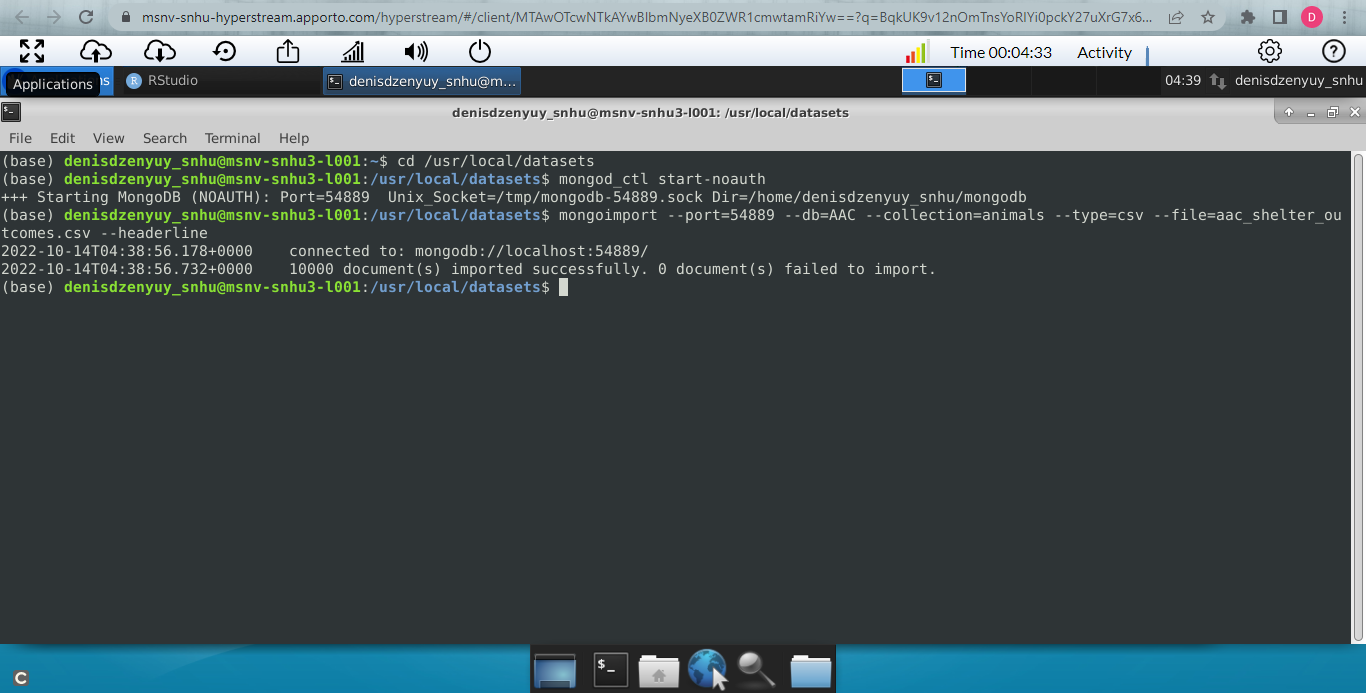


1. Importing and Indexing a Data Set:
2. Changing directory to datasets directory:

**cd /usr/local/datasets**

1. Uploading the Austin Animal Center Outcomes data set into MongoDB by inserting a CSV file using the appropriate MongoDB import tool:

**mongoimport --port=54889 --db=AAC --collection=animals --type=csv --file=aac\_shelter\_outcomes.csv –headerline**



10,000 documents imported.

1. Create an administrator account and a user account in the mongo shell to ensure user authentication to the database and collection that was created.
2. Enabled/created (admin) user authentication for the database

**use admin**

**db.createUser(**

**{**

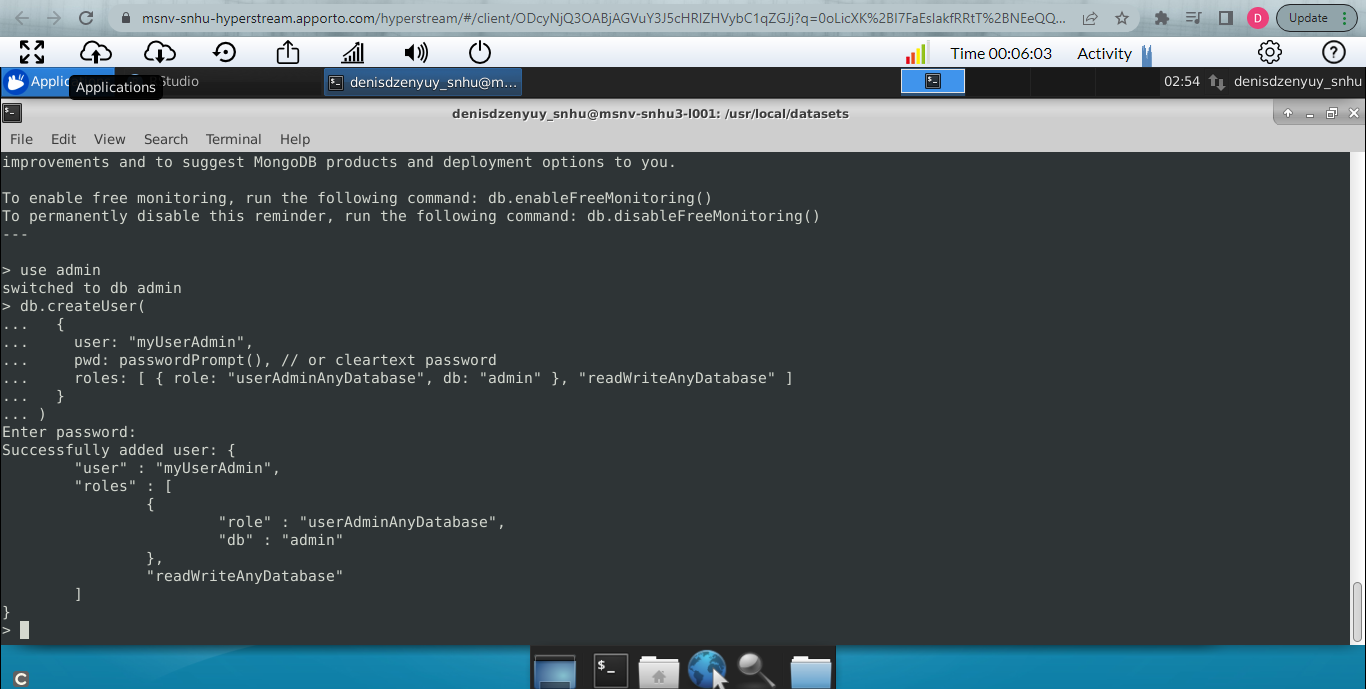
**user: "myUserAdmin",**

**pwd: passwordPrompt(), // or cleartext password**

**roles: [ { role: "userAdminAnyDatabase", db: "admin" }, "readWriteAnyDatabase" ]**

**}**

**)**



1. Create a new user account called “aacuser” for the database AAC.

**use aac**

**db.createUser(**

**{**

**user: "aacuser",**

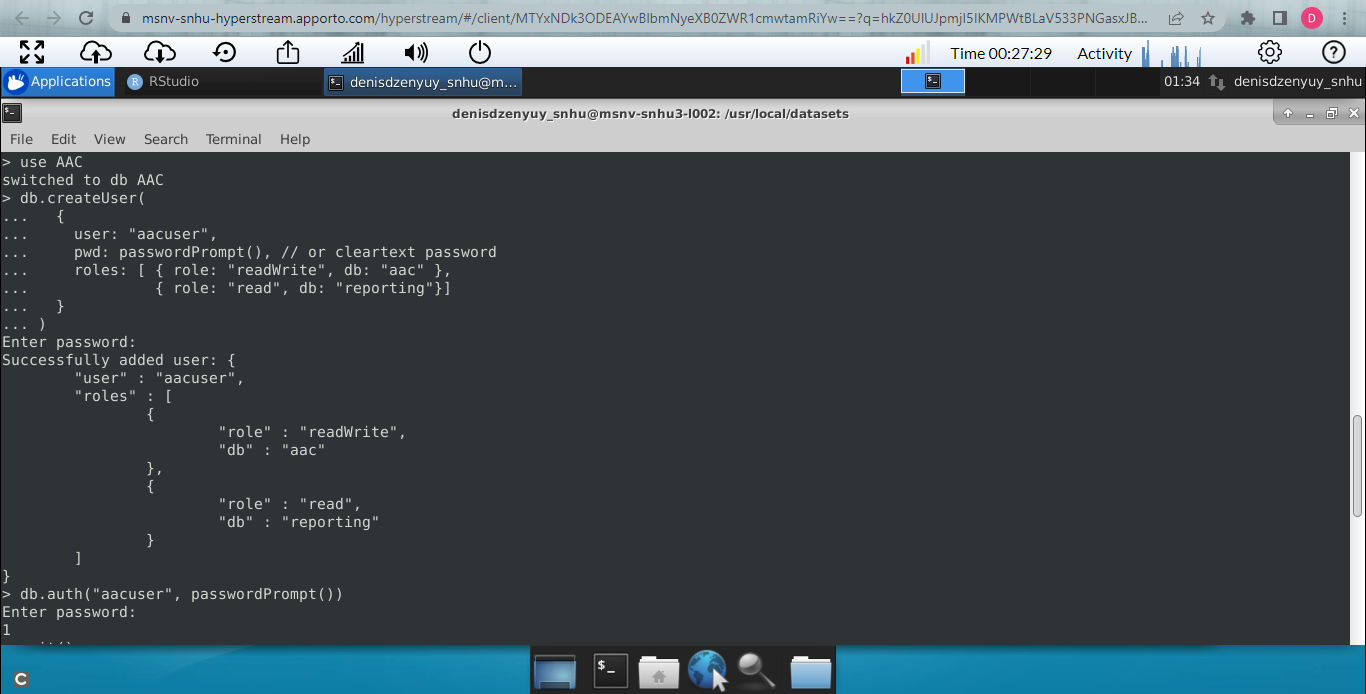
**pwd: passwordPrompt(), // or cleartext password**

**roles: [ { role: "readWrite", db: "test" },**

**{ role: "read", db: "reporting" } ]**

**}**

**)**



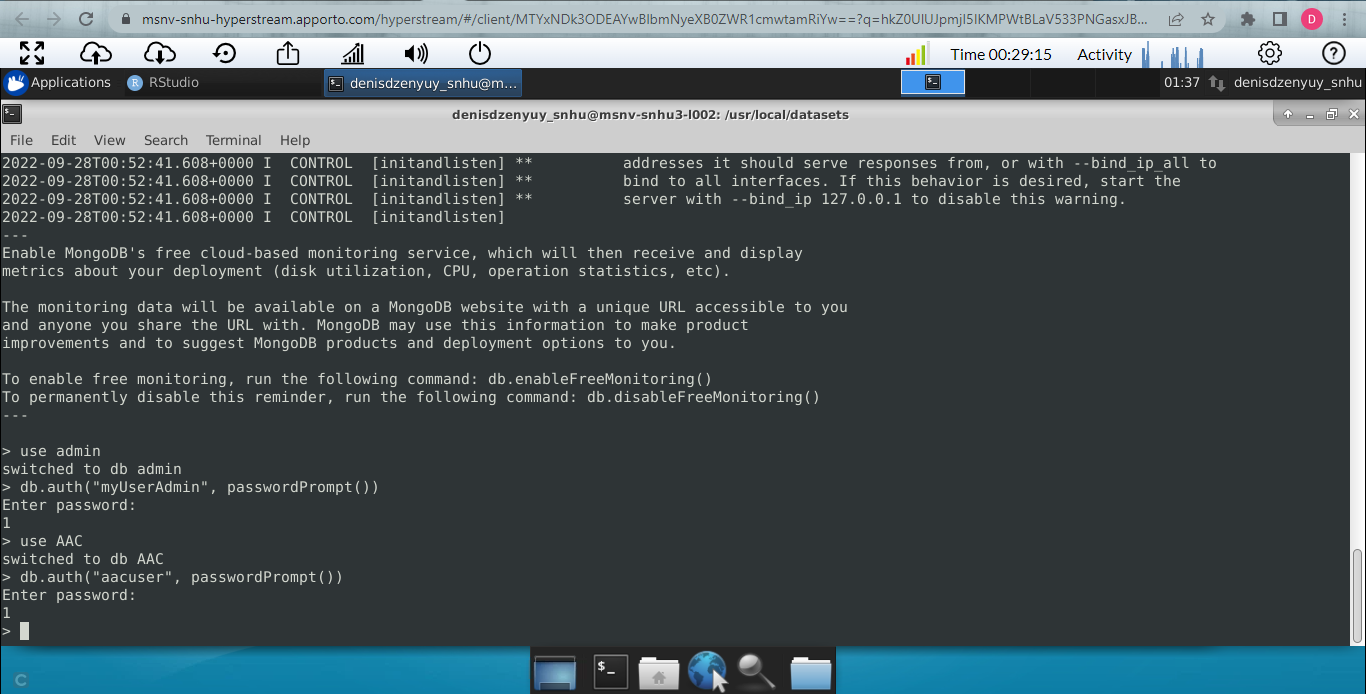
1. Screenshot of login process for newly created (admin/aacuser) users to MongoDB using the mongo shell.

Admin account: **use admin**

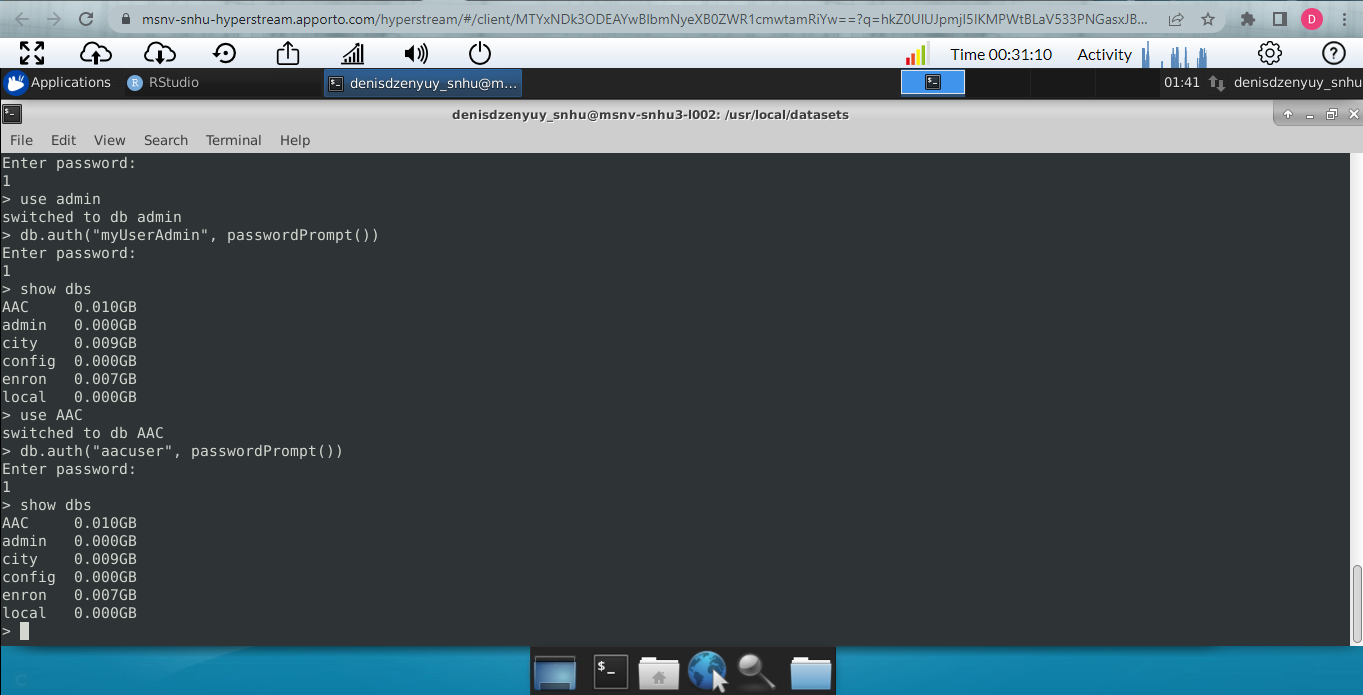
**db.auth("myUserAdmin", passwordPrompt())**

AAC account: **use aac**

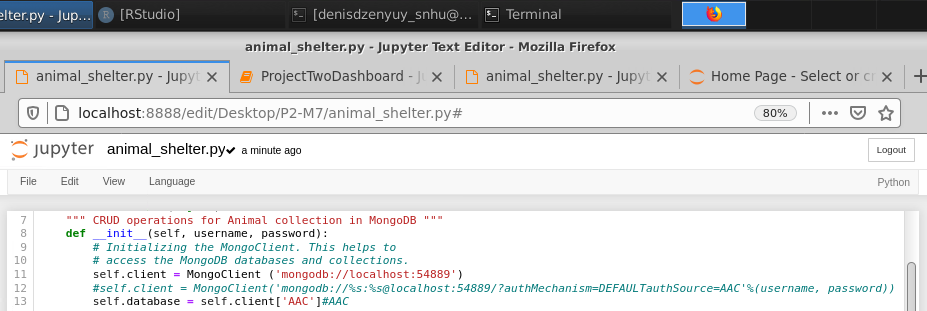
**db.auth("aacuser", passwordPrompt())**



Listing databases using both Admin & aacuser logins/accounts.



* Ensure the port number on localhost of the Animal\_App.py Python code is updated.
* This would be essential when importing the data set into MongoDB by inserting a CSV file
* Update the “aacuser” and “Password” to the username and password you created.
* Sample Initialization:



Test by adding 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.

**Coding the dashboard and the database interface logic:**

* Code a new Dash web application dashboard and configure with the desired HTML/CSS layout and appropriate ids for the data frame, map, and chart.
* Create radial options and program the database queries based on the desired breed specifications for the client.
* Develop an app callback to populate the initial data frame with all of the data.
* Develop an app callback to update the map with the first item of a given category until the user selects an item.
* Build functionality that determines the user selection and displays that on the map instead.
  + There is a possibility of not being able to make more than a set number of simultaneous selections as I came to find out. This was one though part for me to figure out.
* Develop an application callback that updates the pie chart with the data that filters from the display filters.
* Create a pie chart from the displayed data on the screen at any given moment.
* Finally, Develop methods that implement the functionality that you are looking for and the correct target data.

**Installation tools:**

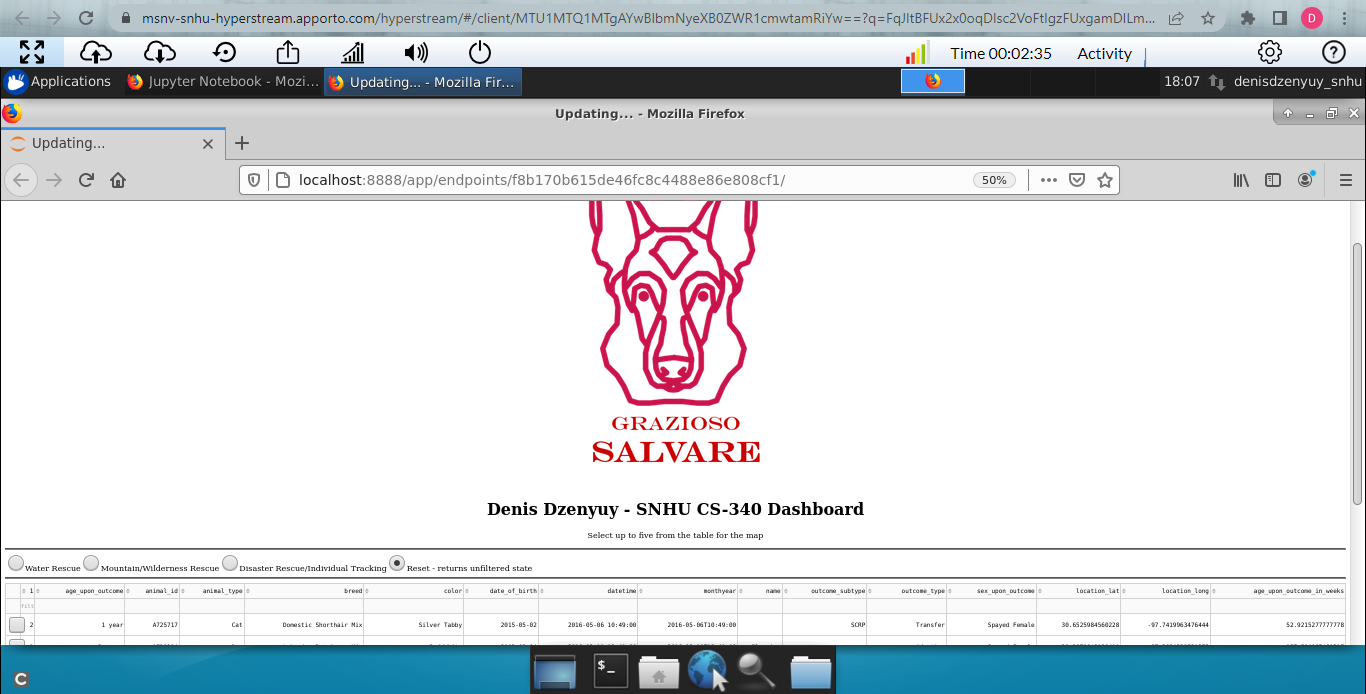
* Python
* MongoDB
* Jupyter Notebook

**Resources/Links:**

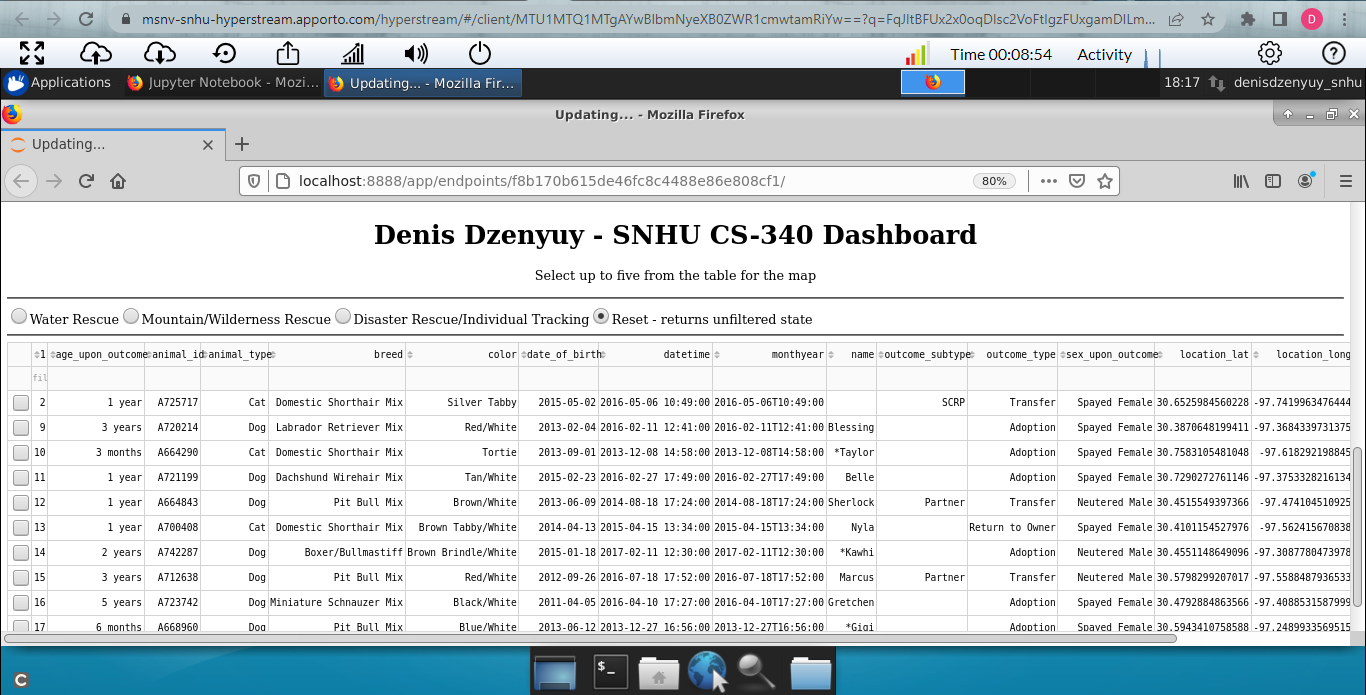
* **Python**: Installation instructions for Python download are available here: <https://realpython.com/installing-python/>. Once you have Python installed, you should be able to use this program from the Terminal on Mac or Linux or from the Command Prompt for Windows.
* **MongoDB**: Detailed instructions for the installation and download of MongoDB (Community or Enterprise editions) 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. To install a local version of Plotly find 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>
* **Image:**The Garzioso Salvare logo was contained in the supporting materials section for this project. It can be obtained here:[**Grazioso Salvare Logo**](https://learn.snhu.edu/content/enforced/1160545-CS-340-T1192-OL-TRAD-UG.22EW1/course_documents/Grazioso%20Salvare%20Logo.png?_&d2lSessionVal=kM43y1biB4r6E4GBRNUz8VHSk&ou=1160545)This high-resolution PNG file contains the Grazioso Salvare logo. The logo is included as part of your dashboard to ensure that the application is properly branded.
* **Data Set:**[Austin Animal Center Outcomes Spreadsheet](https://learn.snhu.edu/content/enforced/1160545-CS-340-T1192-OL-TRAD-UG.22EW1/course_documents/aac_shelter_outcomes.csv?_&d2lSessionVal=kM43y1biB4r6E4GBRNUz8VHSk&ou=1160545)  
  Grazioso Salvare provided this sample data set (CSV file) of animal center outcomes. This was used as the basis of the database and can be used to test the functionality of the code.

**Note:** Per the instructions for this project “This data set has been modified for the purposes of this project. Specifically, the following columns have been added: location\_lat (latitude), location\_long (longitude), and age\_upon\_outcome\_in\_weeks (the age of the animal, given in weeks).”

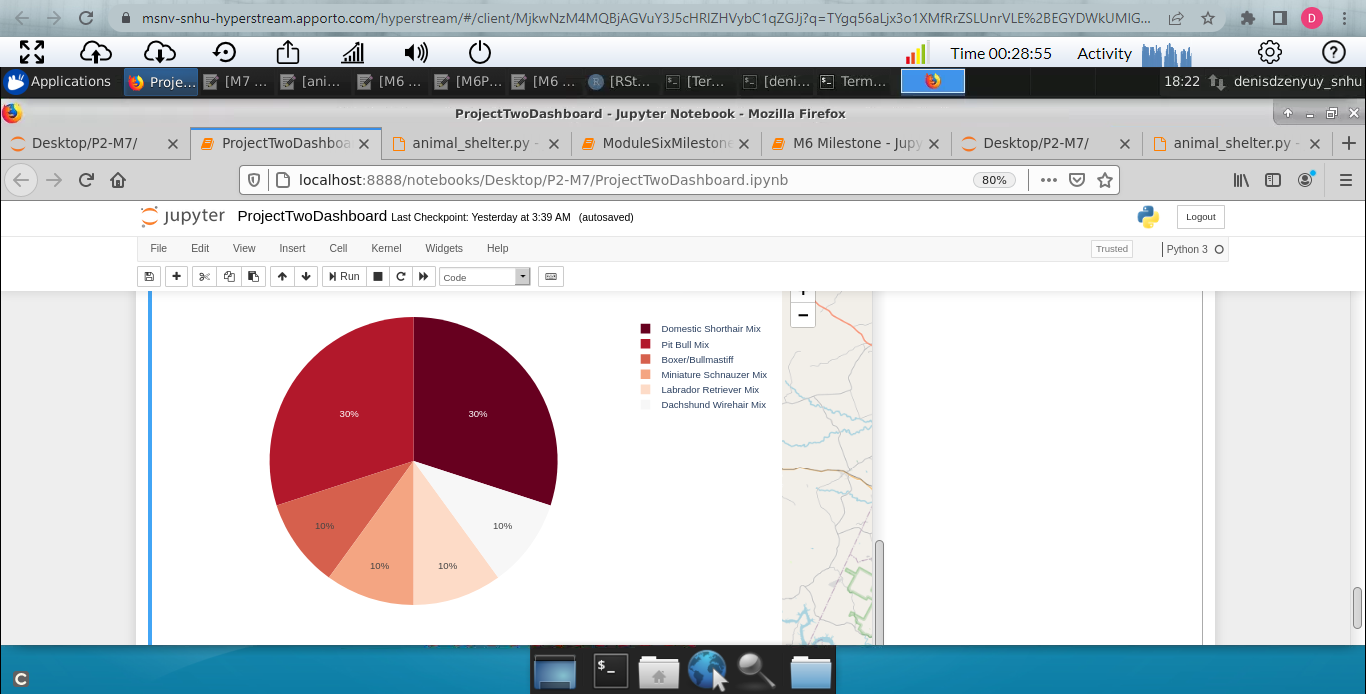
**Grazioso Salvare logo to ensure the application is properly branded**



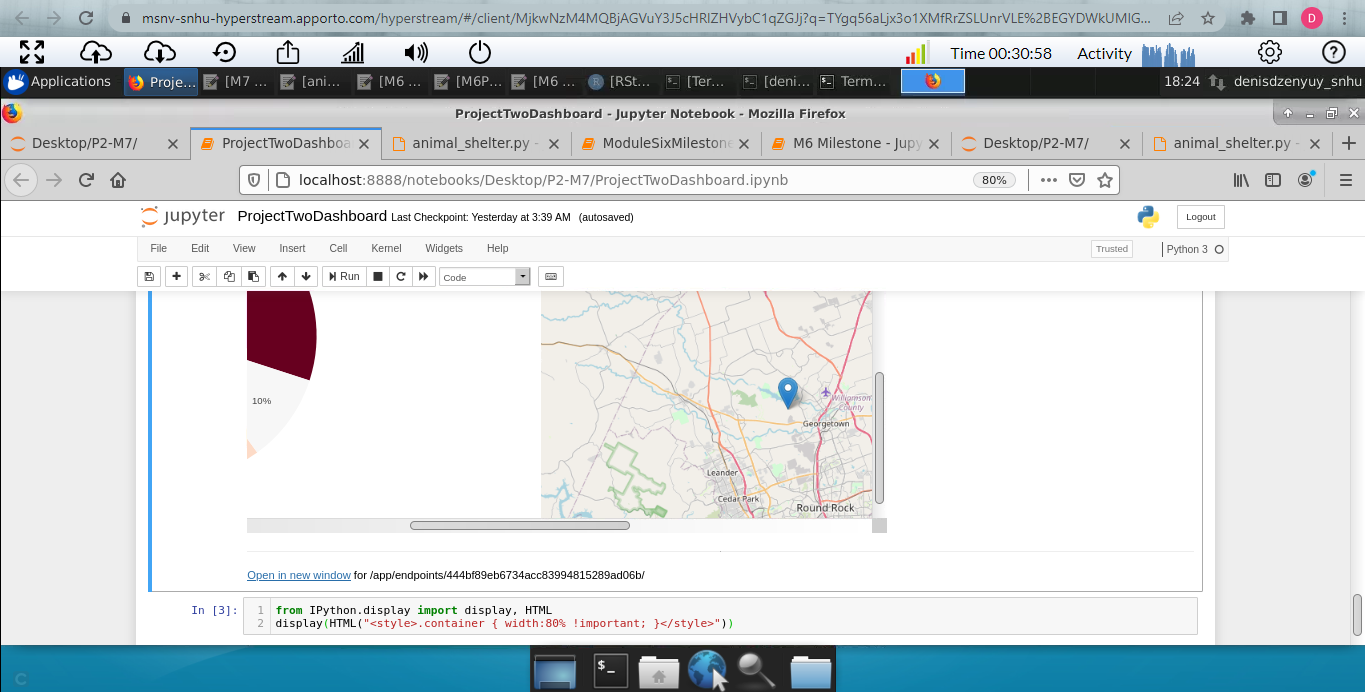
**Dashboard widgets that receive input from the interactive options and present the dynamic updates to the client.**

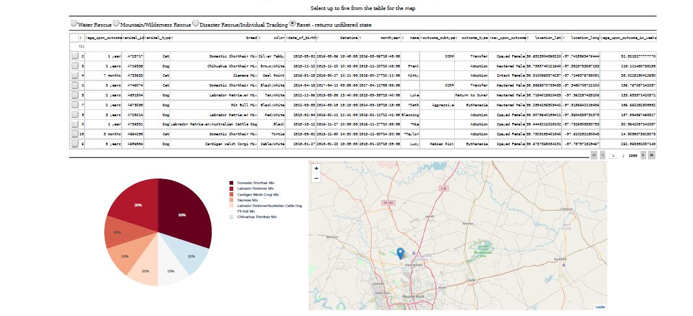


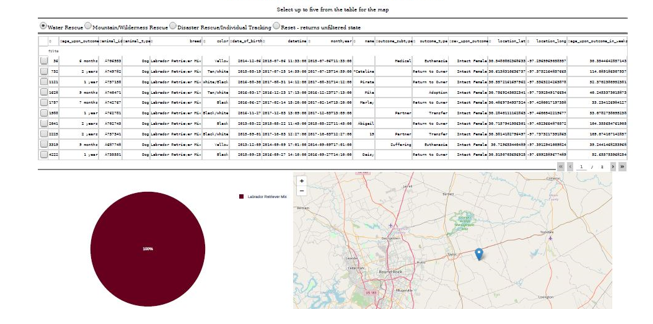
**A pie chart that dynamically respond to the filtering options**

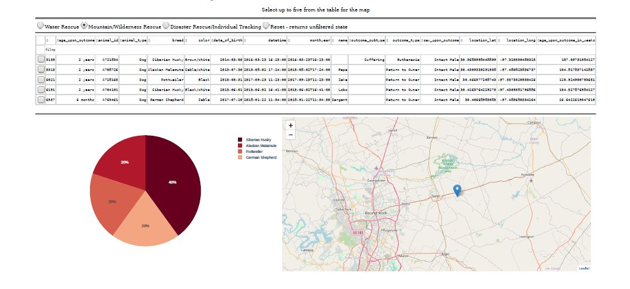


**Geolocation chart that dynamically respond to the filtering options**

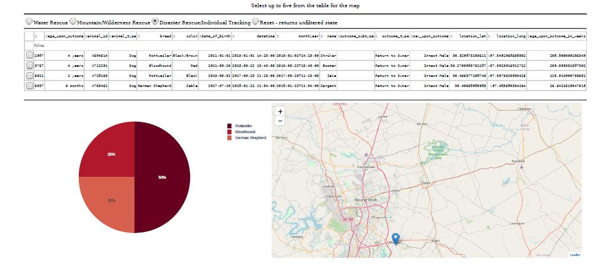


Reset (Show unfiltered Data)

Water Rescue 

Mountain/Wilderness Rescue 

Disaster/Individual Rescue



**Resources:**

**Installation**

The tools needed 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**

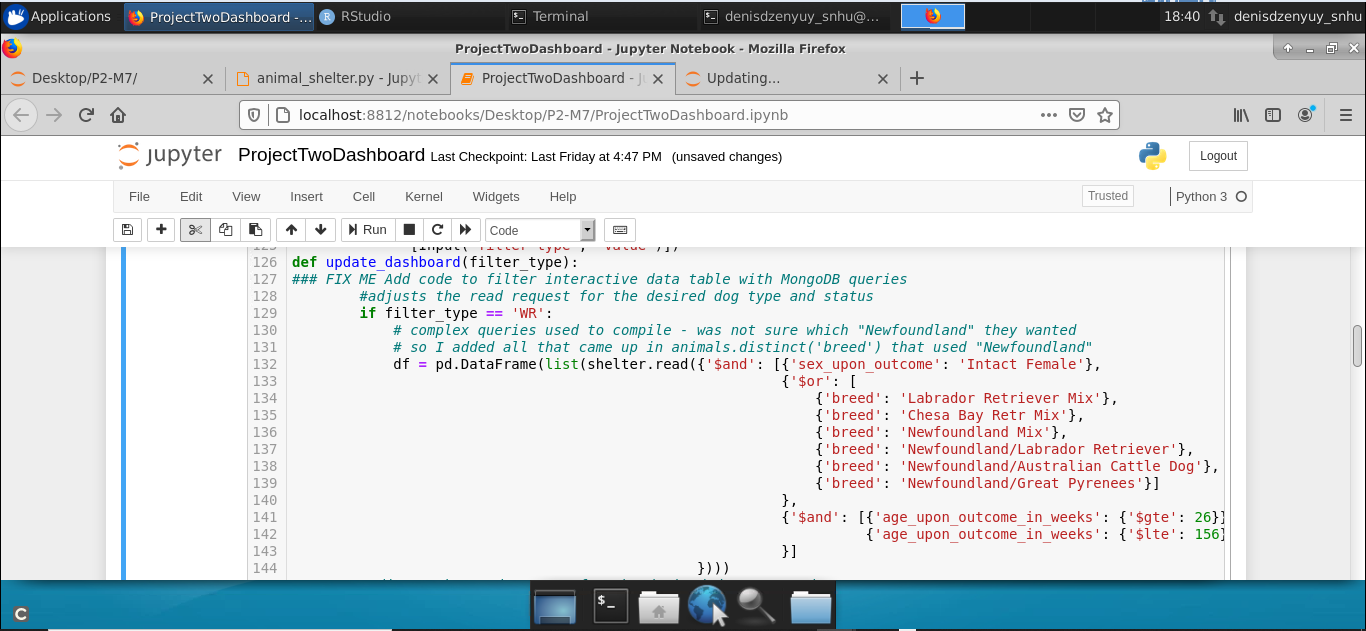
Dash is a framework used to build web applications. You can import the Dash Core Components into your Jupyter notebook and you can install Dash using the following information: <https://pypi.org/project/dash/>

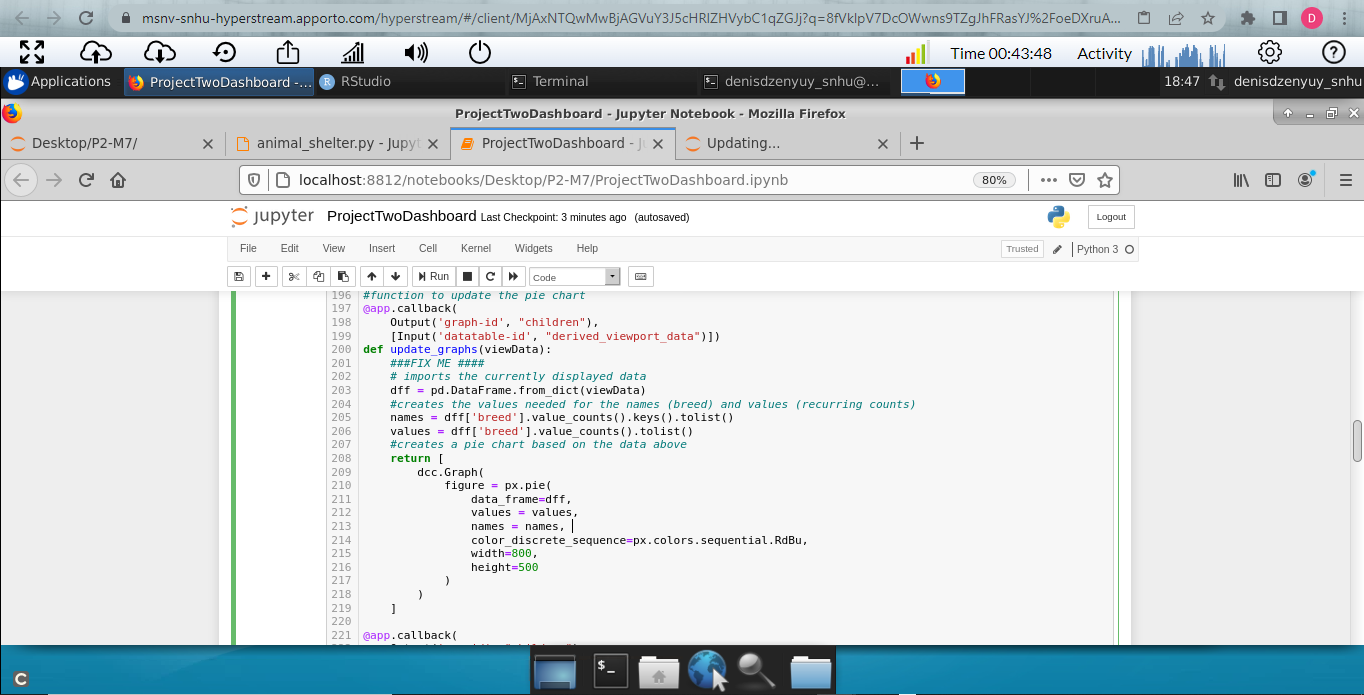
**Pandas**

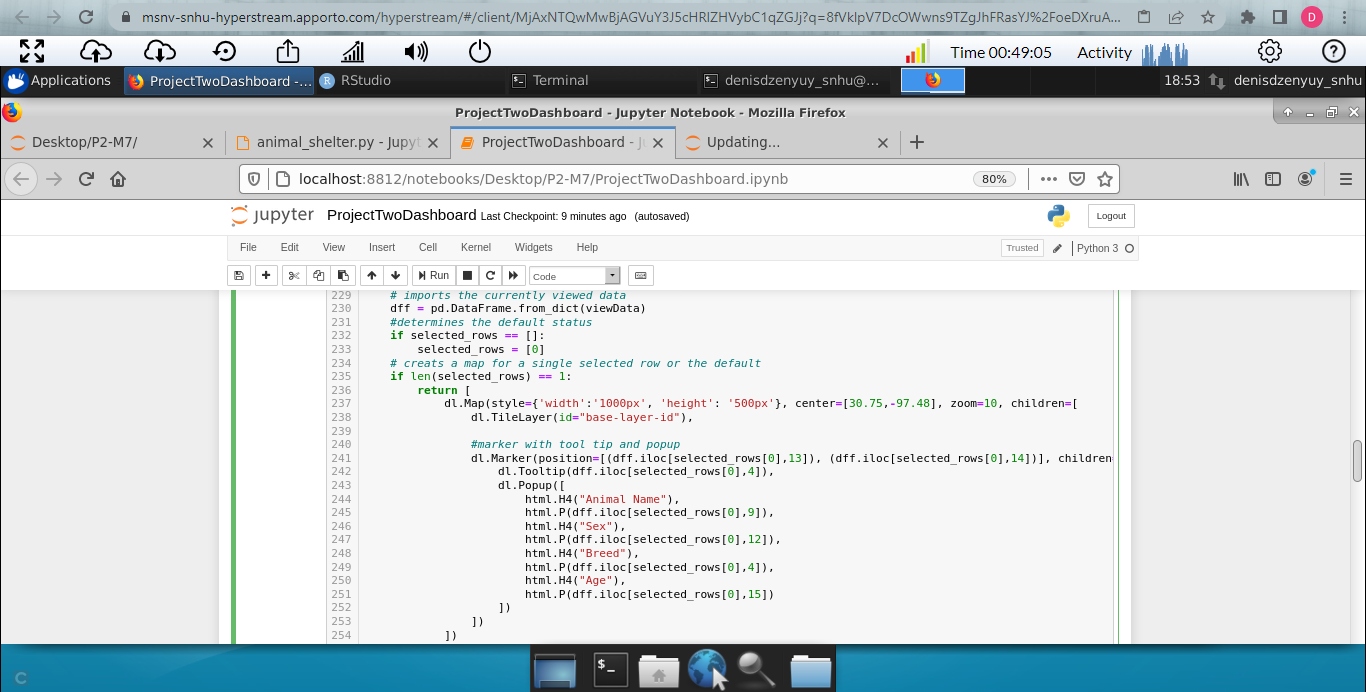
Pandas is used in this web application as well. Pandas is a tool for Python that creates the data frames. Pandas has other dependencies and information that should be reviewed before use here: <https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html>

**Dashboard: Code Samples:**

Filtering interactive data table (Query to compile)



Pie Chart

Map Markers

**Code Example (CRUD Operations)**

***Create*:**

create():

try:

if data != null

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

pprint(insert\_result)

return True

else:

raise Exception(“exception content”)

except:

return Fale

***Read*:**

Read(target):

Try:

If target != null:

read\_result = list(self.database.animals.find(target))

Pprint(read\_result)

Return True

Else:

Raise Exception(“exception content”)

Return False

Except Exception as e:

Print(message + e)

***Update:***

Update(source, destination, count)

Try:

If source != null:

If count == 1:

Update\_result = update(self.database.animals.updateOne(source, destination))

Pprint(update\_result)

Return True

Elif count == 2:

Update\_result = update(self.database.animals.updateMany(source, destination))

Pprint(update\_result)

Return True

Except:

Raise Exception(“exception content”)

Return False

***Delete:***

Delete(target, count)

Try:

If target != null:

If count == 1:

Delete\_result = delete(self.database.animals.delete\_one(target)

Pprint(delete\_result)

Return True

Elif count == 2:

Delete\_result = delete(self.database.animals.delete\_many(target)

Pprint(delete\_result)

Return True

Except:

Raise Exception(“exception content”)

Return False

**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

**Tests**

The create function has two unit tests and the read function has two unit tests. One test in each tests the failure case and the other tests for a successful case. The create functions test whether the return value is true or false. There are several more tests to test the various functionality of each CRUD method. A code example is provided here:

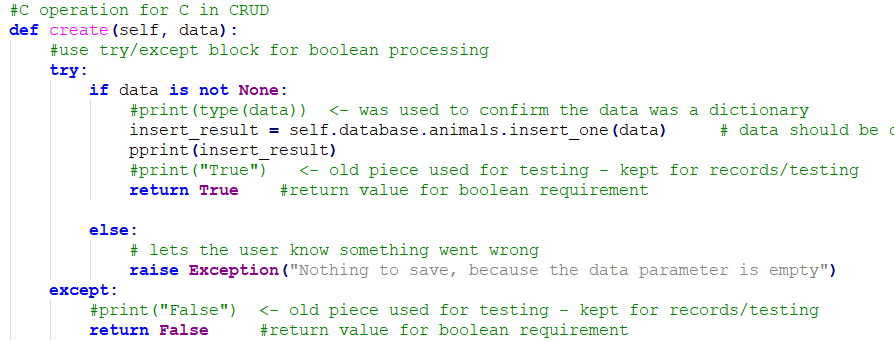
testData():

testShelter = animalShelter()

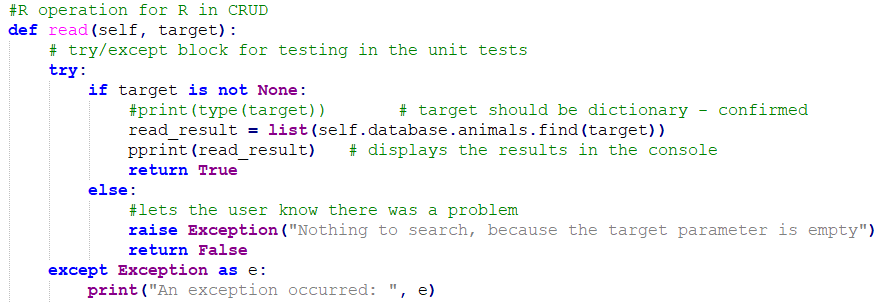
assertTrue(testShelter.create(sampleData))

**Screenshots**

Sample Create:

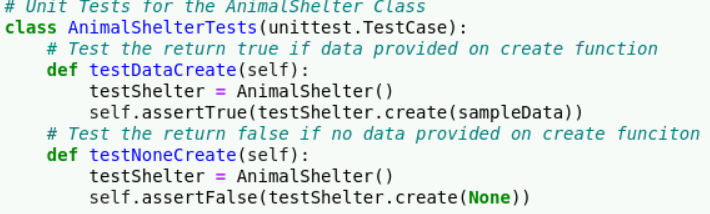


Sample Read:

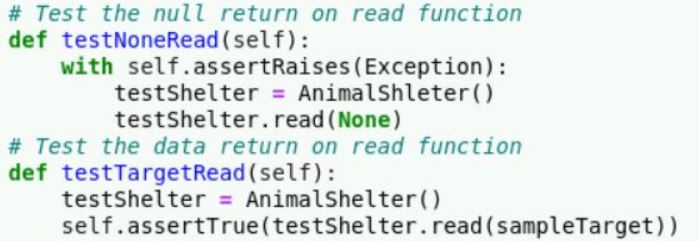


Sample Tests:

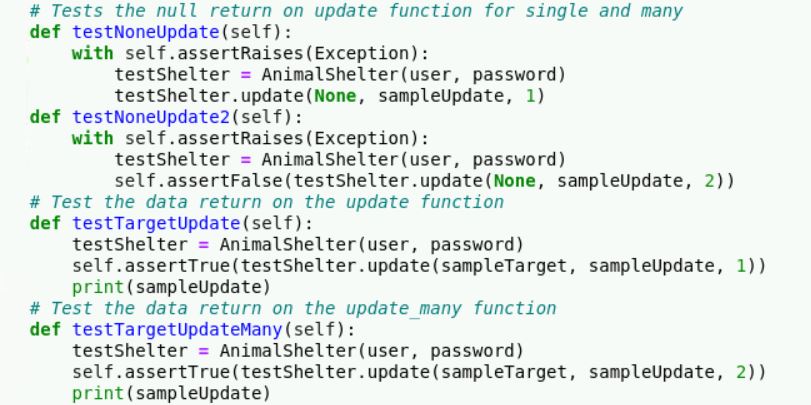
*Create*



*Read*

**

*Update*

**

*Delete*

**

Kind regards,

Denis.