# CS 340 Rescue Animal Database Project README

## About the Project/Project Title

CS-340, Rescue Animal Database Project

This project takes an existing database of animal data and provides access to that database in the form of a website dashboard that displays the data in a grid. The goal is an interface that allows easy access to search for animals with certain criteria that make them eligible for training as rescue animals.

Initial data was provided by Austin Animal Center (AAC) Outcomes.

The project primarily displays database entries with various filtering options to narrow the data down to useful lists that meet rescue dog requirements. While the database interface does support adding new entries and updating values, the dashboard currently does not include implementations to allow this.

## Motivation

This software has been developed for use by Grazioso Salvare, an international rescue-animal training company.

This database and interface are designed to allow users to find suitable animals for training, based on certain criteria. For example, for dogs, breed and age are two important factors that play a part in determining if an animal is suitable and the kind of rescue operation the animal might be best suited for.

Since these values are recorded in the provided data for each animal, using a database and interface is an effective way of both organizing the data, by allowing future additions and deletions, as well as providing quick access to finding qualifying animals using predefined filters and search criteria.

## Getting Started

To run this project locally, you will need to install Python, MongoDB and the JupyterLab environment.

## Installation

This project runs on a MongoDB, using Python code to create the database connection, which is then displayed to the user through the use of Dash via a JupyterLab environment.

This project uses the following Software and libraries.

Software:

* Python 3.11 – Programming language used for all scripts and modules.
  + Rational: Python is a modern programming langue that is simple to create and read; all scripts are written in Python, from the CRUD operations to the dashboard logic (the dashboard is executed using a Python notebook in JupyterLab).
* MongoDB – NoSQL database which stores the animal records.
  + Rational: MongoDB is a NoSQL document-oriented database that works well for JSON-like records such as animal shelter data. Records can have new fields added as needed without requiring a database revision to handle. This expandability could be useful as the company works with the animal records and may need to add indictors in if an animal was selected for trained and what the training outcome was.
* JupyterLab – Environment for running and testing Python scripts and notebooks.
  + Rational: Provides an interactive environment to develop, test, and run Python notebooks and scripts. It supports inline dashboards which are used to display the data in a user-friendly fashion.

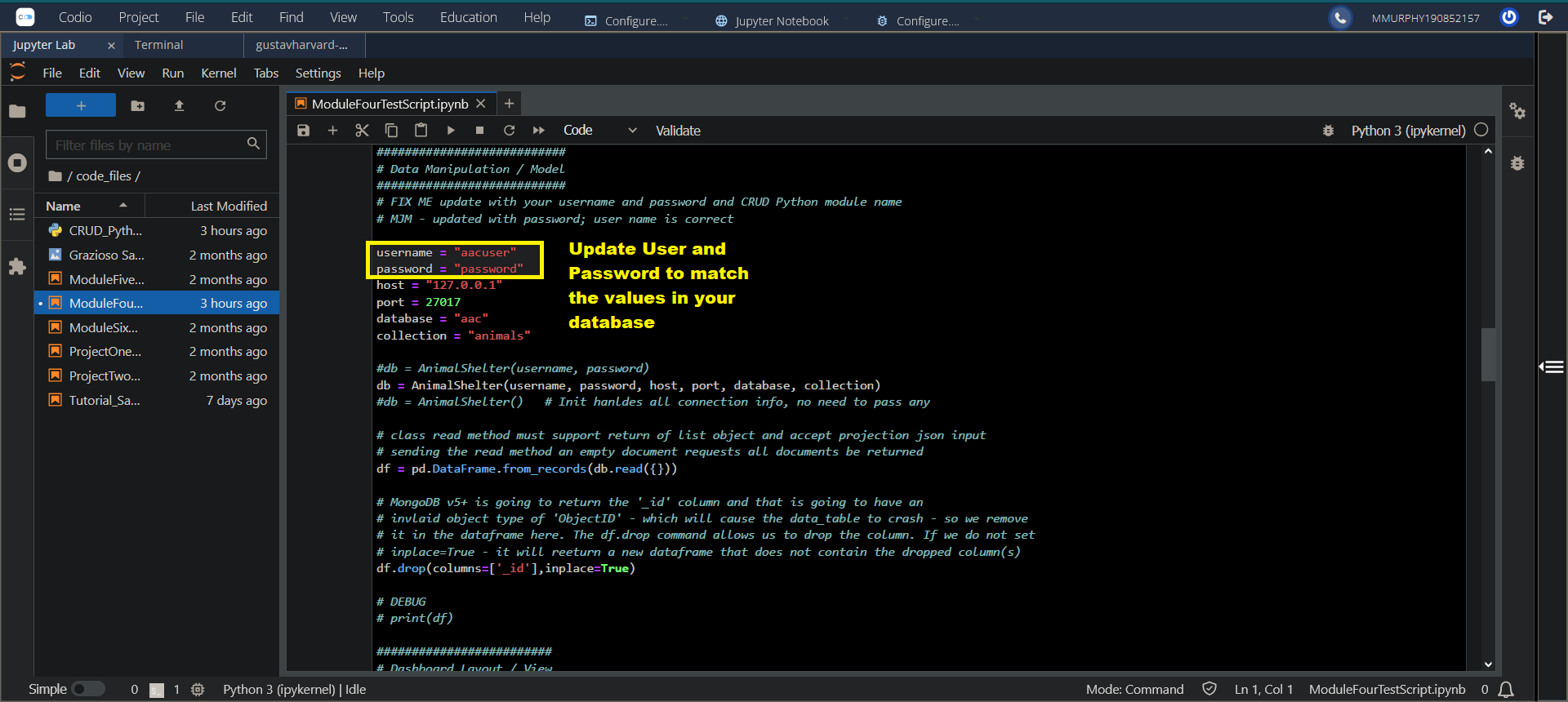
Python Libraries

* pymongo – MongoDB driver for Python to connect and interact with the database.
* pandas – Data manipulation library for working with query results and converting them into DataFrames.
* plotly – Library for charts and graphs within Dash dashboards.
* matplotlib / numpy – Libraries for additional plotting and data analysis.
* jupyter\_dash – Library for running Dash applications inside Jupyter notebooks.
* dash – Library for building the interactive dashboard.
  + dcc – Core Components for dashboard widgets,
  + dash\_leaflet – Library for mapping components in Dash.
  + html – HTML for layout
  + Input, Output – Used to define callback inputs and outputs
  + ctx – Context object to identify which input triggered a callback
  + no\_update – Used to prevent updating a component when no change is needed

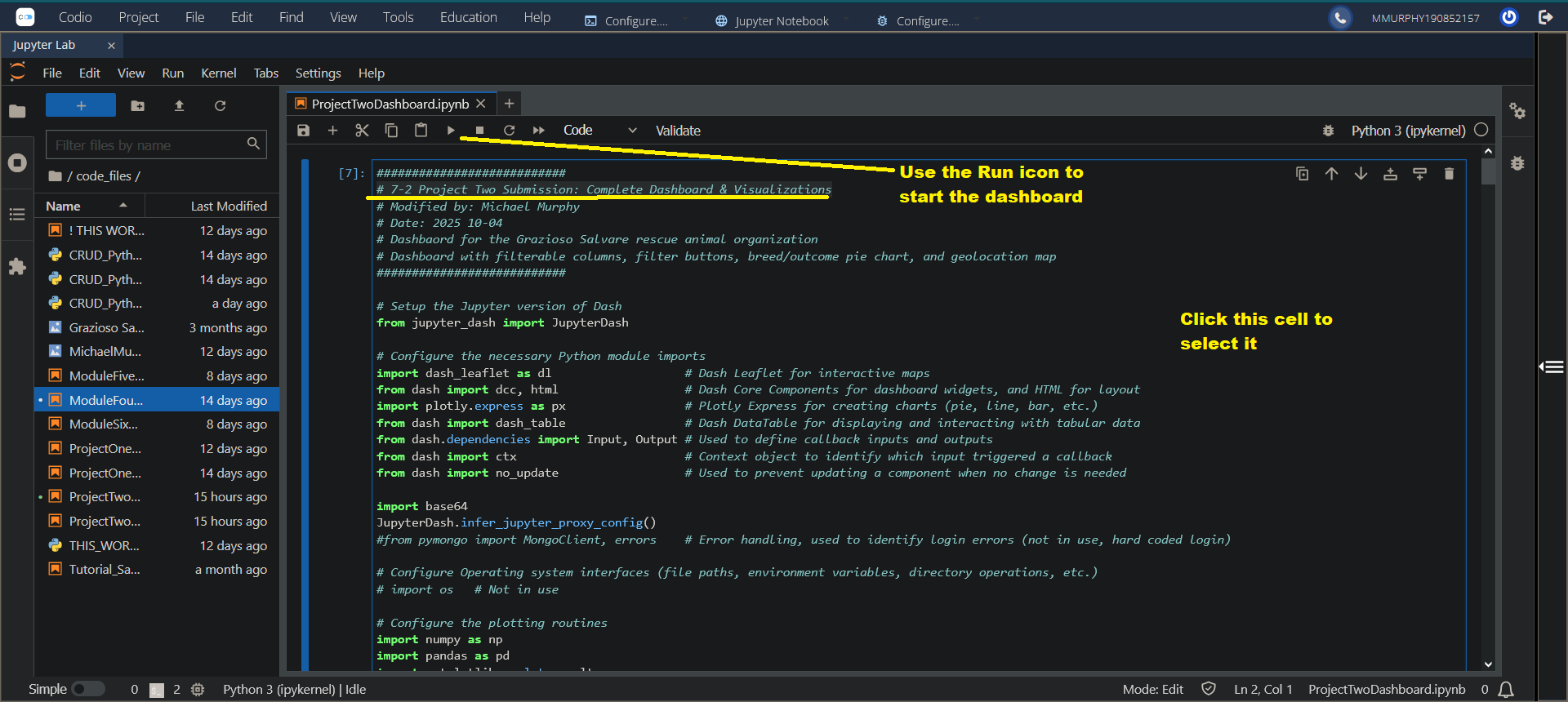
#### Installation Checklist

* Install Python 3.11
  + Available from: Python.org
* Install MongoDB (Community version)
  + Available from: https://www.mongodb.com/try/download/community
* Install the listed Python libraries
  + Install using the pip package manager that comes with Python
* Install JupyterLab
  + Available from: https://jupyter.org/

In MongoDB, create a user that has access to the 'aac' database. Update the code in the '.ipynb' file for the username and password, to pass these values to the database.



Run the project by launching JupyterLab, and run the '.ipynb' file, using the cell that contains the title: 7-2 Project Two Submission: Complete Dashboard & Visualizations. This will kick off the dashboard.



## Completion and Project Challenges

This section covers the steps taken to complete the project and some of the project challenges that were overcome.

#### Completion Steps

This project was completed through the use of iterative feature additions and testing. As with any project, reviewing the requirements is exceptionally important. At each development stage, I took any new requirements and created a document with notes on what needed to be implemented to fulfill those requirements, which aided me in verifying that what I was building was correct. After each portion I not only reviewed my own notes, but I also referred back to the initial requirements list to ensure that I had met them.

The first development steps were to build out the CRUD interface and perform testing on each of those functions. The Jupyter notebook for the project contains the verification unit tests that were used to verify the functionality of the database interface. When testing showed the interface was functional, the next step was to build out the dashboard.

Development began with a simple set of buttons, to show all animals, only dogs, and to filter by the Example entry that I added as part of testing the CRUD interface. Using these I was able to develop an understanding of the interface and how it behaved, and to try out some basic filtering that a user might find useful. Testing these showed that I had work to do in resetting, but they formed the foundation of the filtering work, and using them and the text filters allowed testing of the geolocation map, to show that it was functional and was able to accept dynamic filtering.

The next step was to add the required filters for the animal rescue type. I chose a dropdown as it provided the most intuitive interface for this type of filter, and saved space over a set of radio buttons. I integrated the Rescue type dropdown into the single filtering callback routine, and tested it for functionality, correcting any issues until it functioned properly.

The final stage was to add a pie chart, and a unique control for it. I implemented a second dropdown, and ensured spacing between the two dropdowns, as well as ensuring that the dropdowns themselves were wide enough to easily show the text of the selection. While adding this new dropdown was ultimately successful, it produced some challenges related to unwanted control activation and determining how to properly reset all the widgets.

#### Challenges

There were two major challenges I encountered when building the project: determining how to reset all the widgets back to their initial state, and handling out of bounds index callback errors.

As part of the build process, I created a callback function that handled the button clicks and filter dropdown; the idea was to create a single callback function whose domain was the filtering of the table data. In building it out, I gradually added more inputs and outputs to the function as I added and modified the controls.

However, when I added inputs for the table data and the pie chart dropdown to the callback function so that I could pass through their current values without changing them, I found that uses of those controls were incorrectly captured by the callback and triggered the if..else statement causing undesired resets and setting changes.

With research and testing I found that it was possible to set a value as an output without the need for that control as an input, but I still needed a way to avoid changing their value, which is why I implemented Dash's no\_update functionality. With this, I was able to omit all inputs by controls I did not wish to trigger the callback, and still perform resets to those controls when required, while avoiding any change to them when no change was desired by setting their output value to no\_update.

The next major challenge was the out of bounds index callback errors. For the geolocation map, the index is passed in as an input, however the index is not correlated by Dash with the table's current row display, meaning that it was possible to get invalid values for index. For example, if row 15 was selected, then the filter changed to only have 10 entries, row 15 is no longer a valid index for the currently filtered list.

I spent time determining how and why the index value was incorrect, and my various iterations of testing resulted in a revised and better structured geolocation callback, with additional checks on the viewData and early exits (returns) from the callback when invalid data was found. Ultimately the answer was to stop testing the index for only "None" and instead to check if the index was valid at all (None or empty):

|  |
| --- |
| **if** **not** index**:** **# None or empty list**  row **=** 0 **# Set row directly to 0, which is the initial row** |

## Usage

Here is an example of the dashboard:



Once the '.ipynb' file cell with the Dashboard Layout / View section is run, a new tab will open displaying the dashboard. There may be a short delay in the tab opening and the dashboard displaying.

The **Reset** and **All Dogs** filter buttons are at the top left. Reset will clear and reset the dashboard back to its initial display, while the All Dogs button will filter the list to just Dogs. See the [Functionality Walk Through](#_Functionality_Walk_Through) section below for complete dashboard overview.

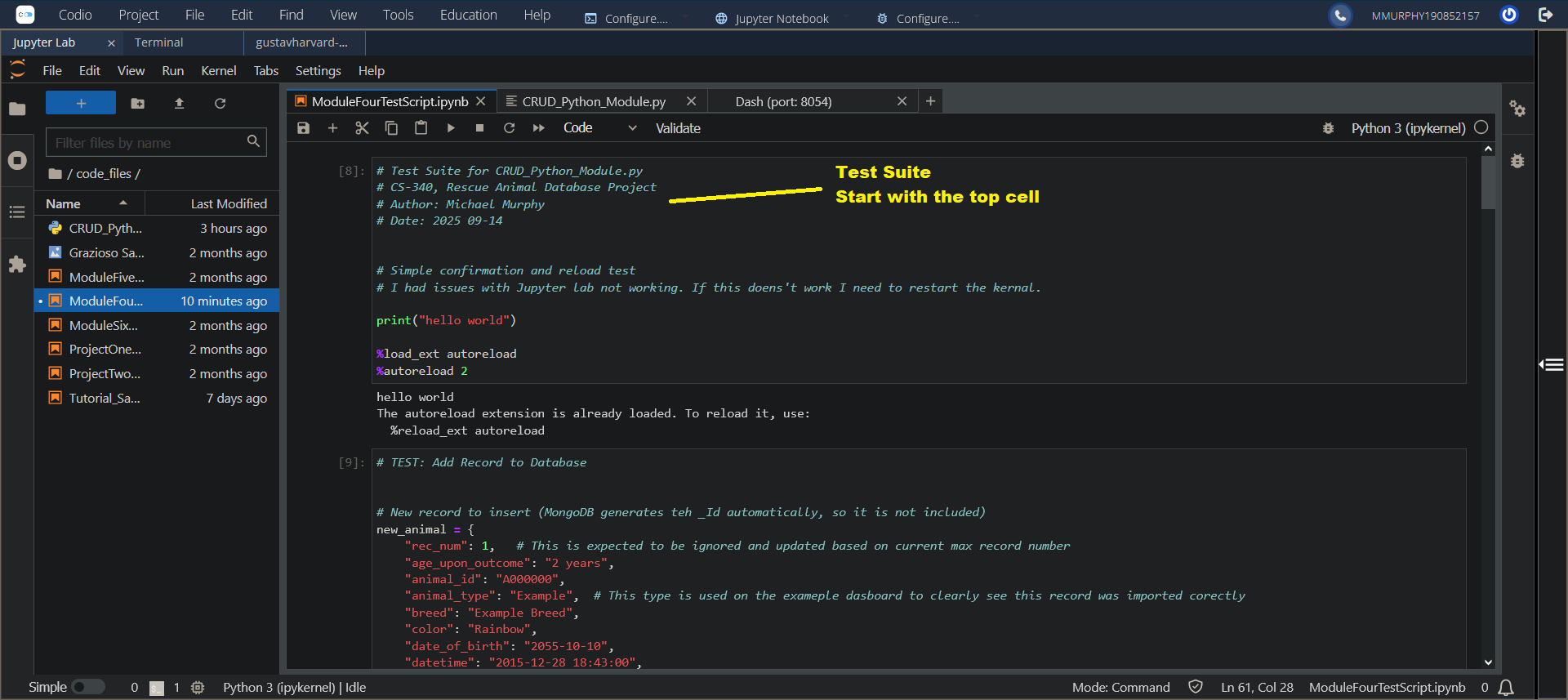
### Code Example

Current Python code is built on an AnimalShelter class and includes a constructor, create and read functions.

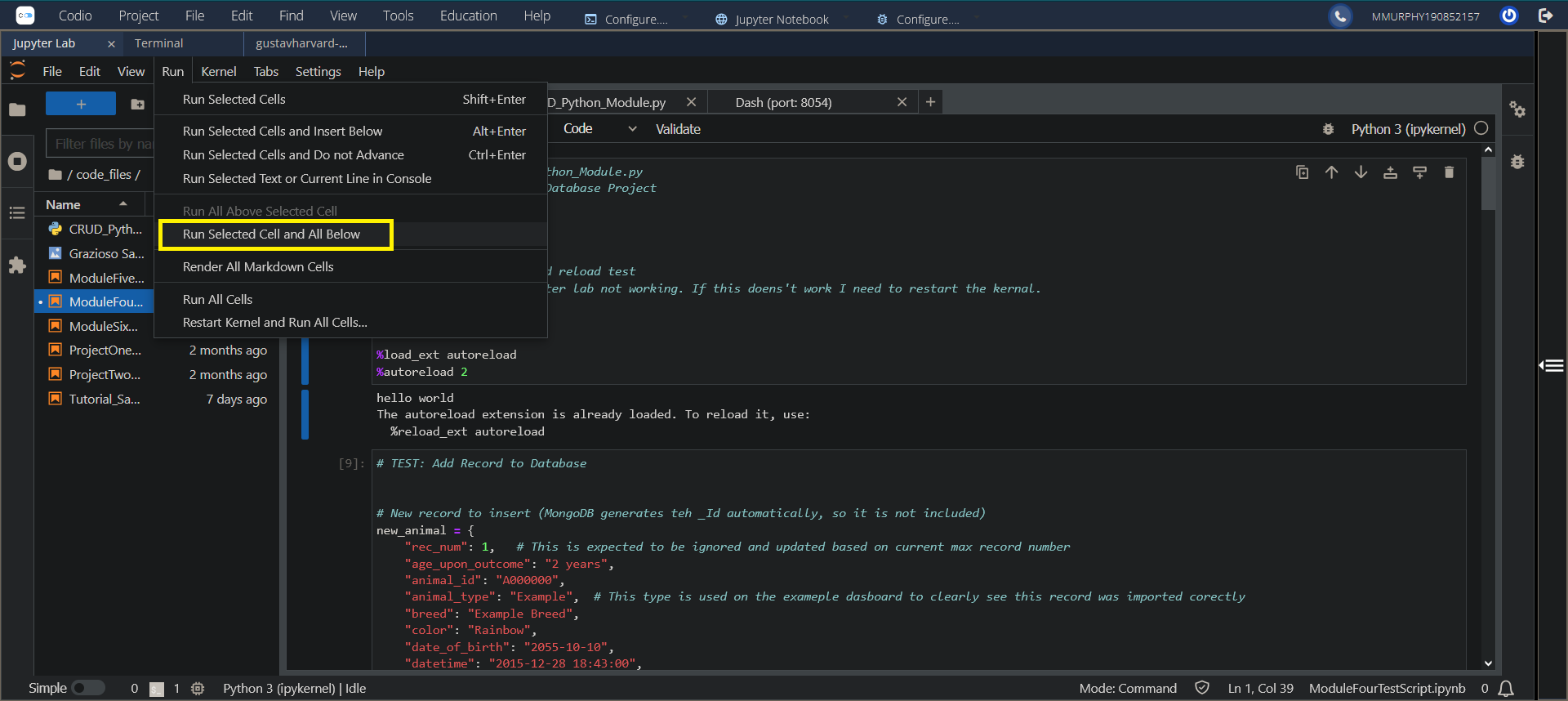
|  |
| --- |
| **class** AnimalShelter**(**object**):**  **# Constructor**  **def** \_\_init\_\_**(**self**,** username**,** password**,** host**,** port**,** database**,** collection**):**  **# Create method**  **def** create**(**self**,** data**):**  **# read method**  **def** read**(**self**,** query**):** |

### Tests

To run and test the database, start with the first test suite cell; this should be the second cell in the workbook (the first cell being the dashboard).



From the Run Menu, select the "Run Selected Call and All Below" option.



This will kick off with a verification "hello world", which will display right below the cell; if that text is not displayed the environment may be having issues, and could require that the kernel be restarted.

The test will proceed to insert an Example record into the database, update that record, then delete the record. If the tests function correctly, you can run just run the add or add + update tests, then confirm the Example data was loaded successfully by running the dashboard cell and searching the database for "Example" in the Breed column.

### Functionality Walk Through

#### Initial Dashboard View (Top)

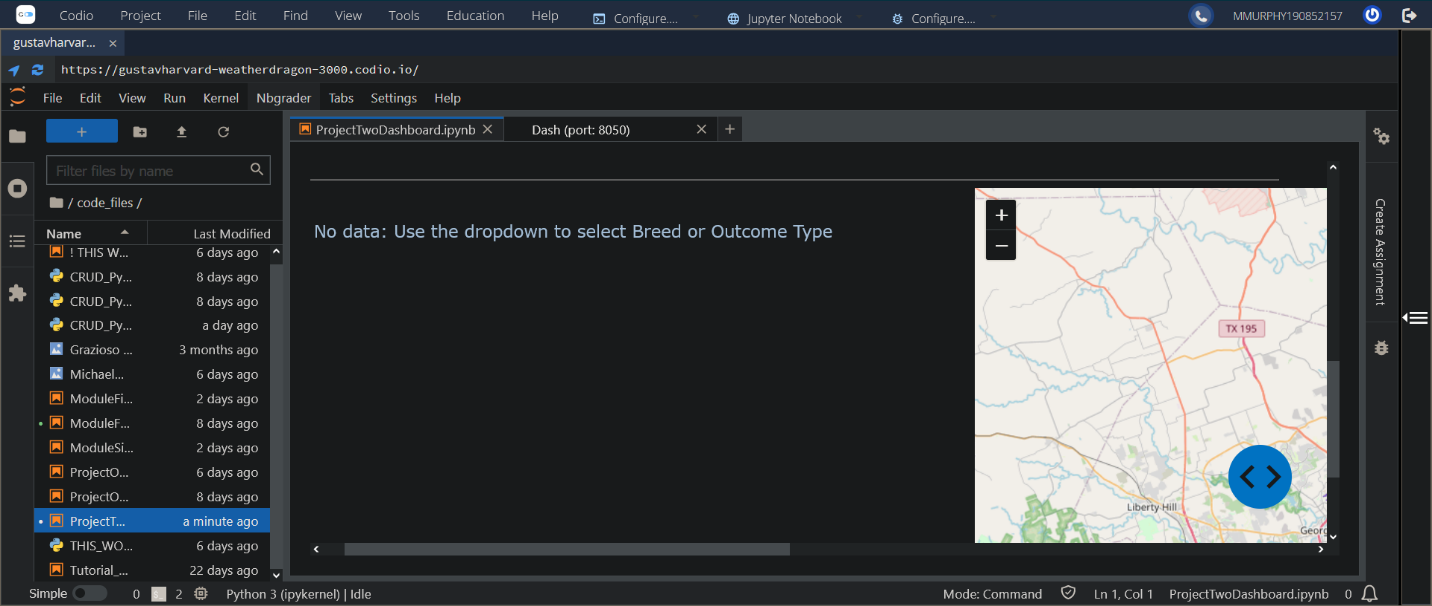


The top section of the dashboard contains the buttons, dropdowns and column text filters that allows a user to filter and view the animal shelter data.

The following is a list of dashboard controls:

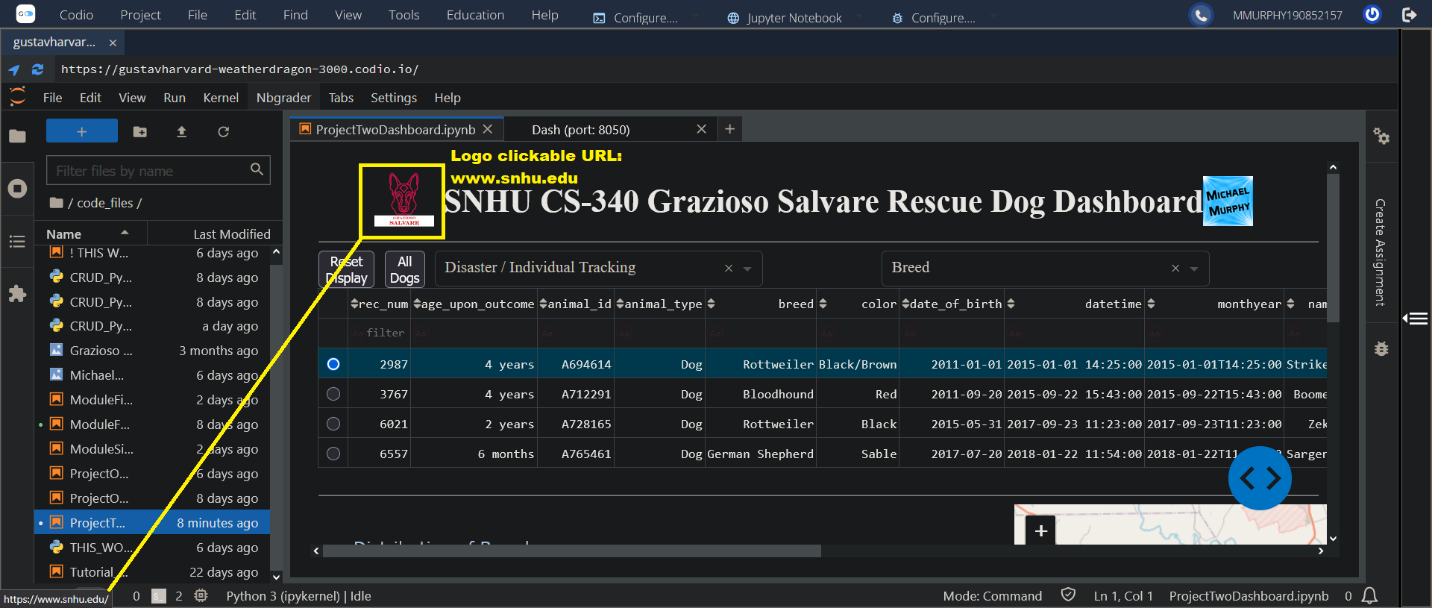
| **Control** | **Description** |
| --- | --- |
| **Reset Display Button** | Resets the data displayed to the initial view; clears the two dropdowns and the text filters. |
| **All Dogs Button** | Shows the full data set, filtered to show only dogs.  Rescue animals are only dogs, so filtering by dogs will be a typical user action. |
| **Rescue Type Dropdown** | Filters the list by the type of rescue operation the dog is suited for:  • Water Rescue  • Mountain or Wilderness Rescue  • Disaster Rescue or Individual Tracking |
| **Pie chart: Breed / Outcome Dropdown** | Filters the pie chart by either the Breed types or the Outcomes of the animals selected in the list.  This allows easy determination of the distribution of breeds and outcomes, as that can impact animal suitability. |
| **Column Filter Text Box** | Each column has a text box above it for filtering. Text entered will need to be an exact match for the value; however you can toggle case sensitivity by the small button inside each text box. |
| **Column Sort Arrows** | Sorts the list based on the column selected by clicking an arrow, and you can toggle the direction by clicking the other arrow. |

#### Initial Dashboard View (Bottom)



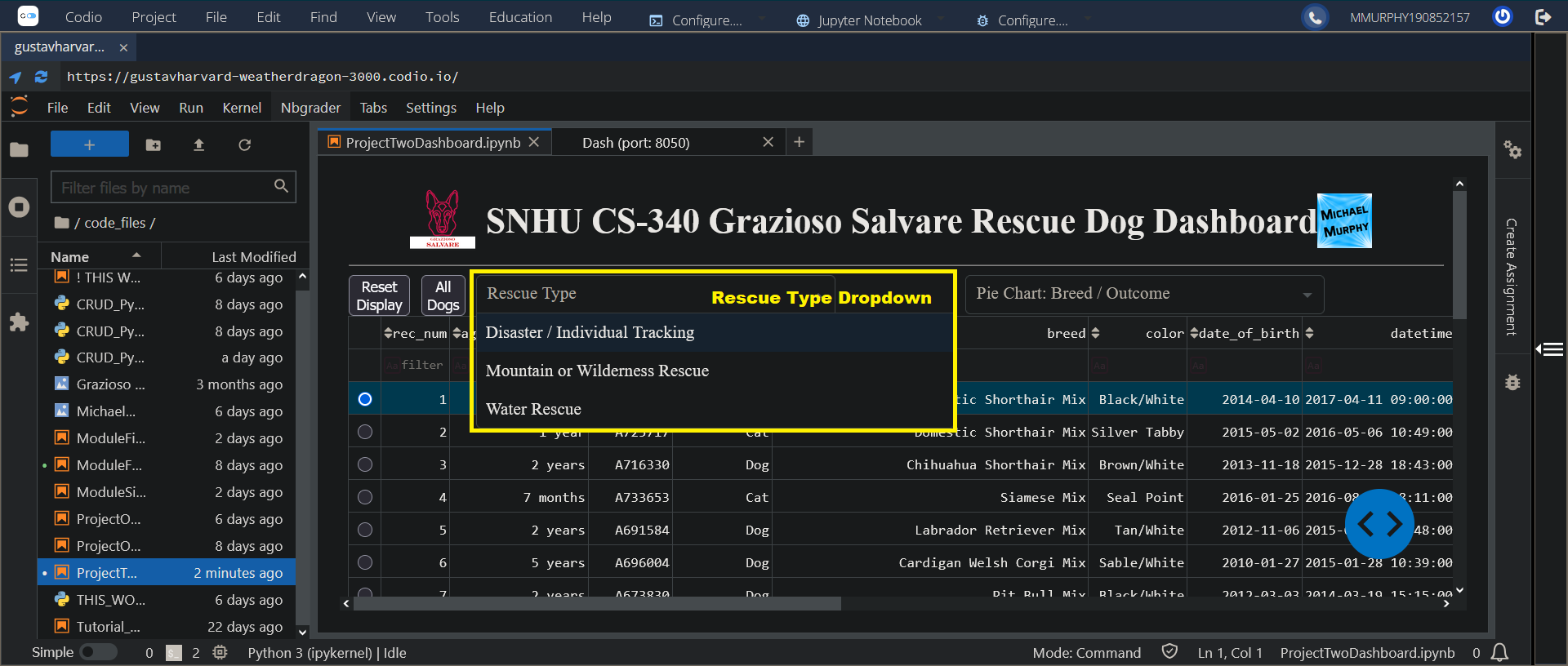
By scrolling down you can view the bottom section where the pie chart and geolocation map are located. The pie chart is intentionally not initially displayed since it would reflect either the breeds or outcomes of the entire dataset. Instead it is recommended to filter the list in some way before setting the pie chart dropdown due to the delay in reading the database for a large set of data. However, it does work for the full set if activated, it is just slow to load.

#### Logo and Header



The dashboard header contains the Grazioso Salvare logo with a URL anchor tag to www.snhu.edu, and a unique identifier containing my name, as per the business requirements.

#### Rescue Type Dropdown

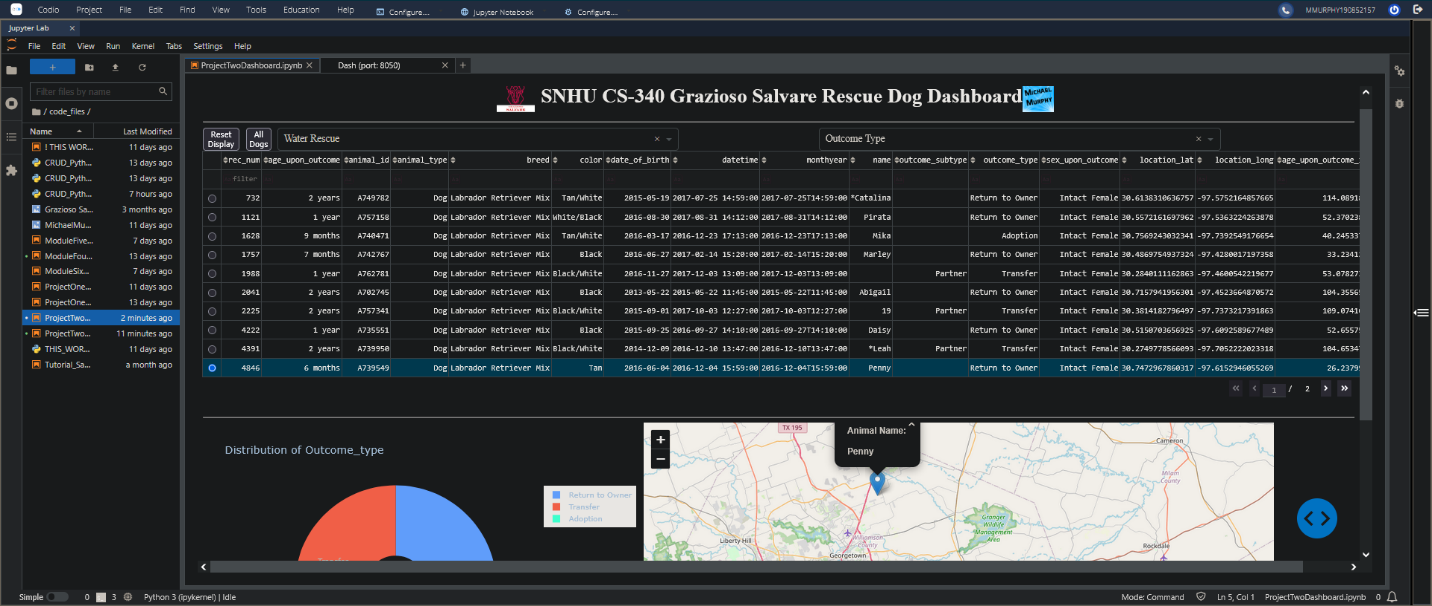


Rescue dropdown filters data by the relevant column values:

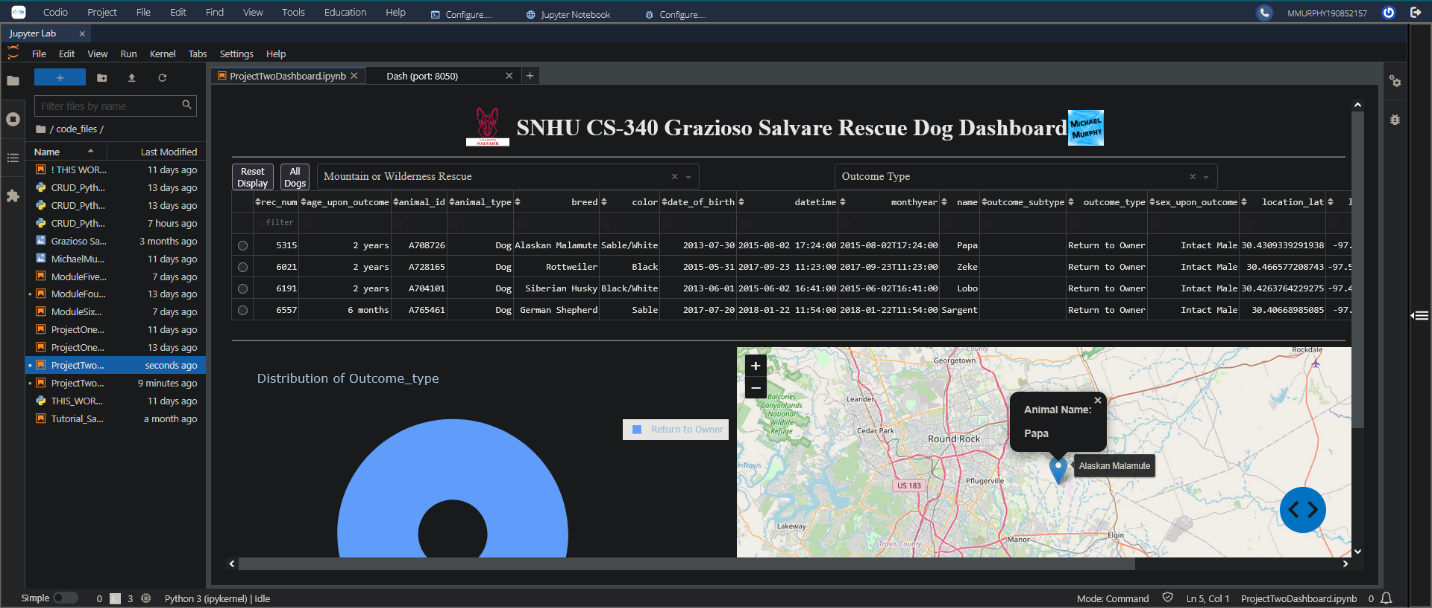
| **Rescue Type** | **Preferred Breeds**  breed | **Preferred Sex**  sex\_upon\_outcome | **Training Age\***  age\_upon\_outcome\_in\_weeks | **Disqualifiers**  outcome\_type |
| --- | --- | --- | --- | --- |
| **Disaster or Individual Tracking** | Doberman Pinscher  German Shepherd  Golden Retriever  Bloodhound  Rottweiler | Intact Male | 20 weeks to 300 weeks | NOT:  Euthanasia  Died |
| **Mountain or Wilderness** | German Shepherd  Alaskan Malamute  Old English Sheepdog  Siberian Husky  Rottweiler | Intact Male | 26 weeks to 156 weeks | NOT:  Euthanasia  Died |
| **Water** | Labrador Retriever Mix  Chesapeake Bay Retriever  Newfoundland | Intact Female | 26 weeks to 156 weeks | NOT:  Euthanasia  Died |

Below are examples of the Rescue Type dropdown in use, and the expected results.

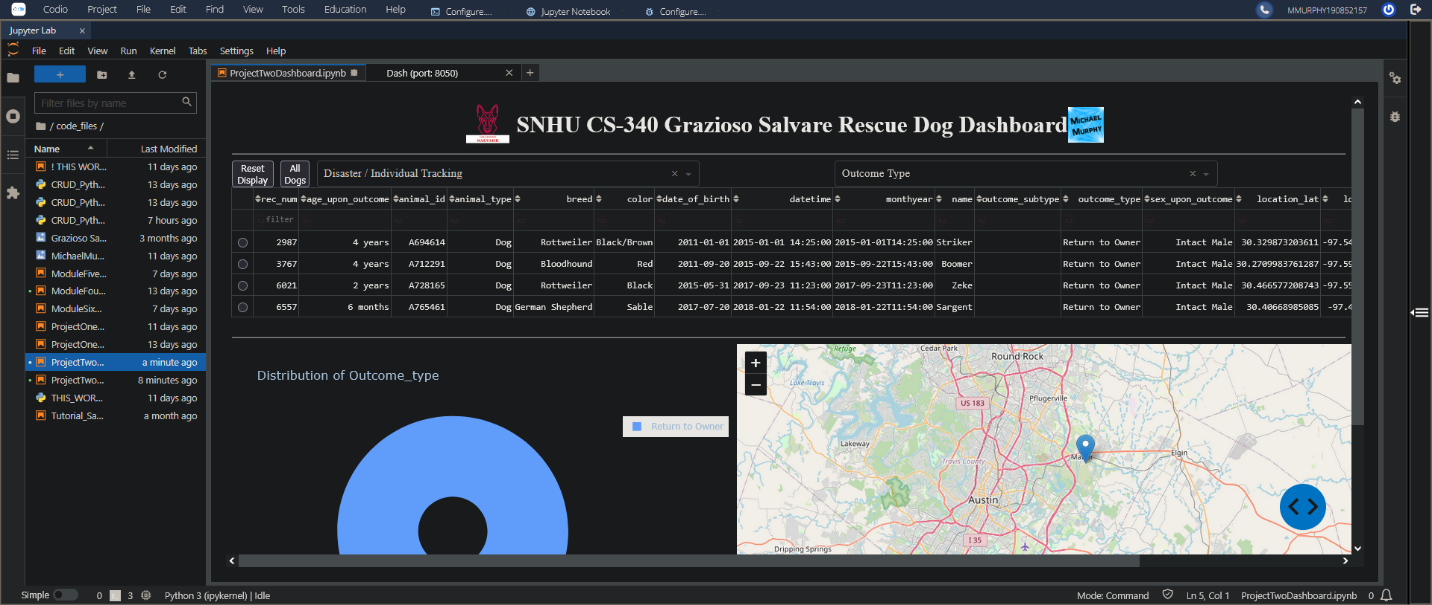
##### Example: Water Rescue



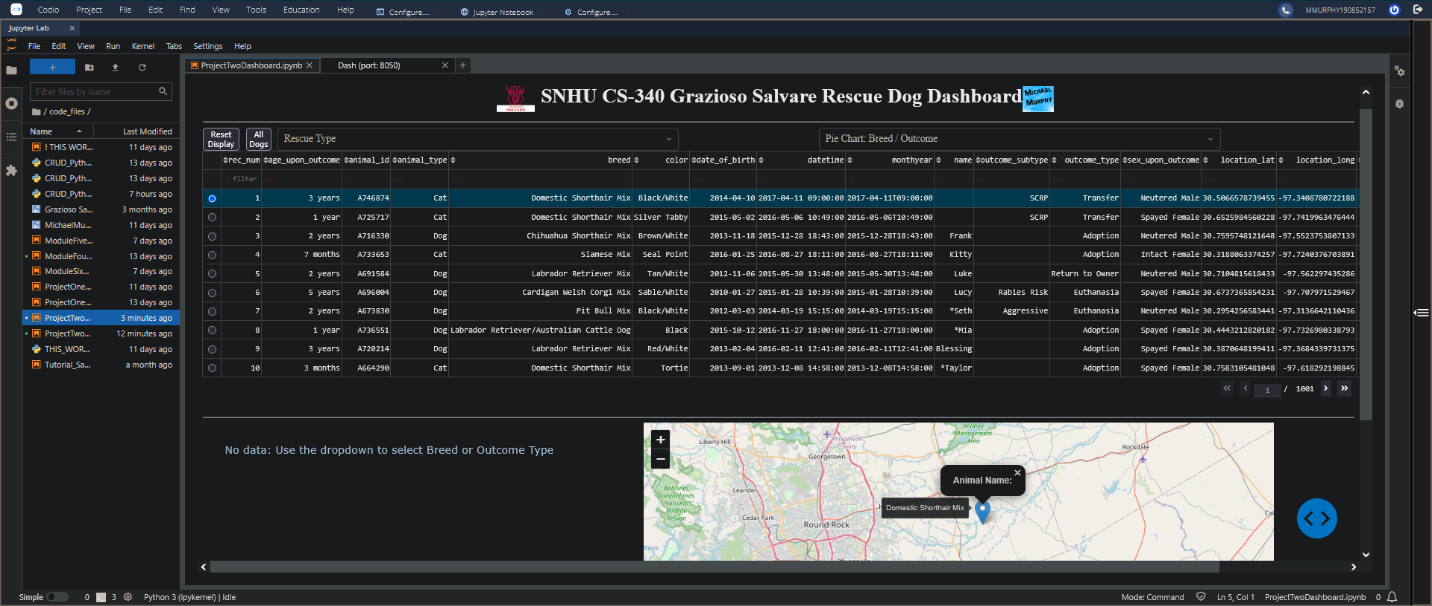
##### Example: Mountain or Wilderness Rescue



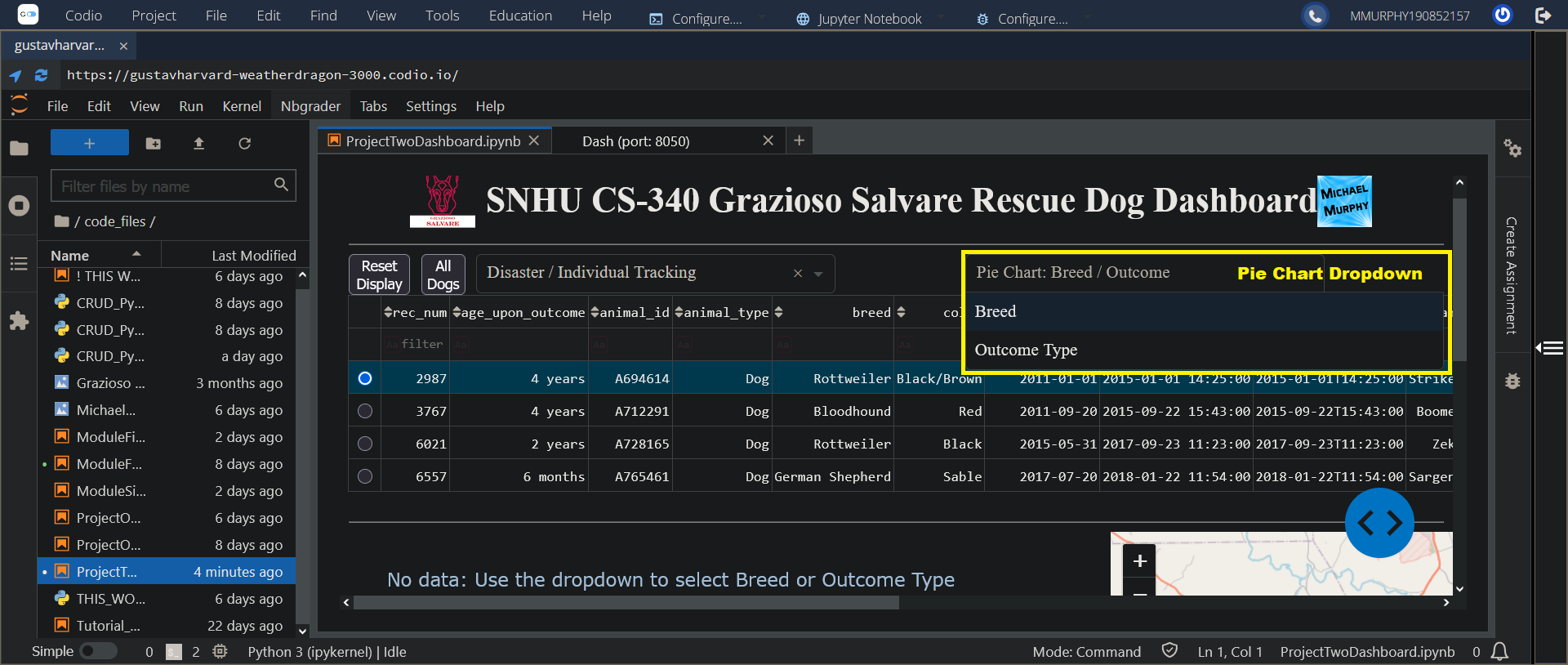
##### Example: Disaster or Individual Tracking



##### Example: Reset (returns all widgets to their original, unfiltered state)



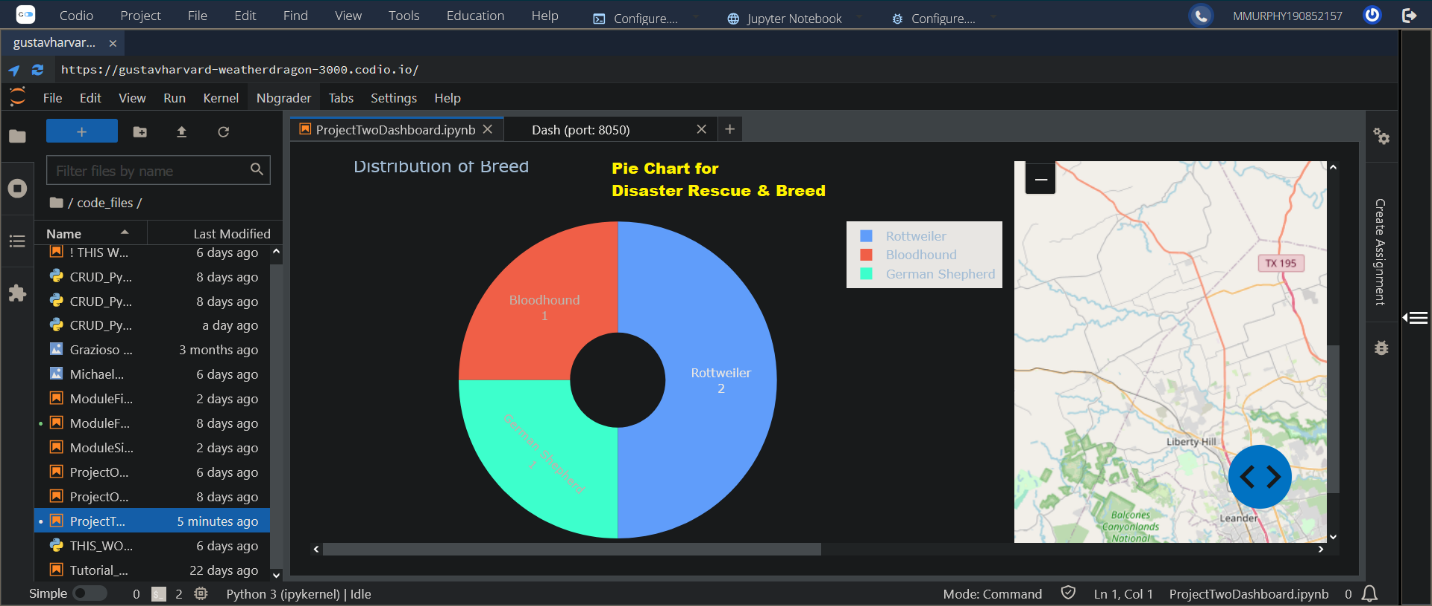
#### Pie Chart: Breed / Outcome



The pie chart dropdown allows the selection of either Breed or Outcome Type. Once selected, it will generate a pie chart and dynamically display the relevant data based on the currently filtered selection.

If no data is available, either because the pie chart dropdown has not been used to select a value or because the filtered data provides no results, then it was display the text as shown above.

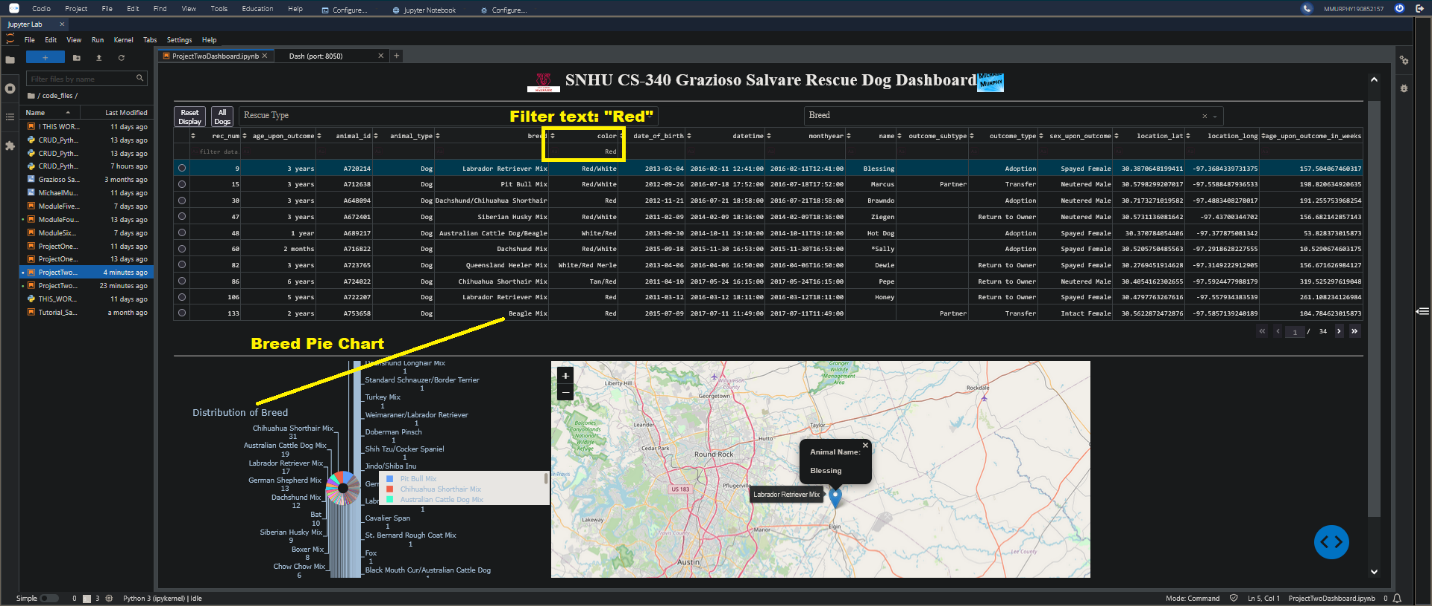
##### Examples:



This is an example of the pie chart and geolocation map that dynamically respond to the filtering options.

* The pie chart starts off inactive by design, since activating it would cause it to read and display the entire database's results.
  + It can be activated at any time via the dropdown selection.
  + If activated before making any other selections to limit the data, it will display a chart based on the full database, which can take a while to render.
* Once the pie chart is activated by selecting Breed or Outcome from the pie chart dropdown, any additional filtering automatically updates the pie chart to show relevant values for the filtered dataset.
* The Geolocation map defaults to the first entry of the filtered data, and will dynamically update to display the location data from the row selected; to select a row, click the radio button by that row. The selected row will highlight in blue.

In this example, the pie chart dropdown was set to "Breed" and the Color column was filtered by the text "Red". This demonstrates the responsiveness of the data table and charts to multiple controls at the same time.



## Roadmap/Features

At this time the project has met all initial requirements for the client. Future development will depend on client needs and any reported defects or corrections.

#### Proposed Additional Features:

* Optimize queries by using database indexes for common queries
  + An index focused on animal\_type, breed, sex\_upon\_outcome, and age\_upon\_outcome\_in\_weeks will speed up making searchs for rescue animals.
* Implement user access verification
* Add code to protect against injection attacks/errors
* Support custom filters
* Support updating records
* Support adding records
* Support the addition of new fields to the records.
* Add new charts or displays

## Contact

Michael Murphy