## *Written and designed by: Adam Vosburg*

## About the Project:

*Animal Sanctuary CRUD Module*

This project is a Python module that provides Create and Read functionality for an Animal Sanctuary database. It allows users to add new animal records to the database and retrieve existing records based on various criteria.

## Motivation:

The Animal Sanctuary CRUD Module was created to make accessing and managing animal records in a rescue shelter setting easier. By providing an easily accessible interface for adding and retrieving animal data, this module helps sanctuary staff maintain accurate and up-to-date information about the animals in their care. This project exists to improve the animal management operations at Austin Animal Center and to ensure that critical information about each animal is readily accessible.

## Getting Started:

To get a local copy up and running, follow these simple steps:

* 1. Ensure you have Python 3.6 or later installed on your system.
  2. Clone this repository to your local machine.
  3. Install the required dependencies (see Installation section).
  4. Set up your MongoDB database and user authentication.
  5. Run the module to start interacting with your Animal Sanctuary database.

## Installation:

To use this software, you'll need the following tools:

1. Python 3.6 or later: Chosen for its simplicity, readability, and extensive library support for database operations.
2. MongoDB: Selected for its flexibility with document-based storage, ideal for animal records with varying attributes.
3. PyMongo library: The official MongoDB driver for Python provides a seamless interface between Python and MongoDB.

To install the required library, run the following command:

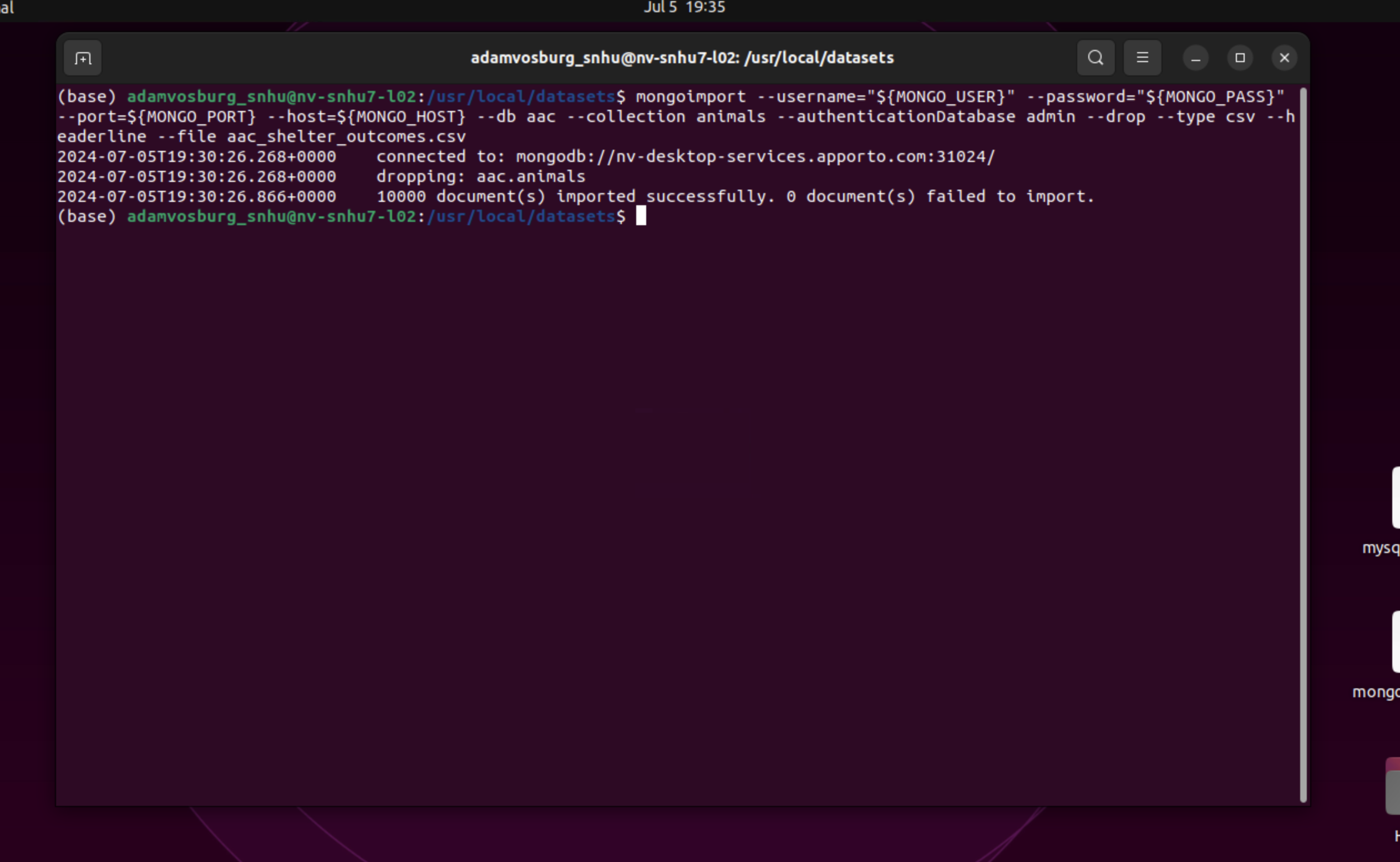
*pip install pymongo*

## Usage:

The Animal Sanctuary CRUD Module provides a simple interface for creating, reading, updating, and deleting animal records in the sanctuary database. Here are some examples of how to import the csv file using MongoDB, create a new user and use the module:

### Code Example:

### Importing a CSV and creating a user:

****

This step imports the animal data into our MongoDB database. The output shows the number of documents successfully imported, which should match the number of records in your CSV file. This creates the initial dataset that our CRUD module will interact with.

****

Creating a dedicated user for our database is a crucial security practice. This user has specific permissions to interact with the animal sanctuary data, limiting potential unauthorized access. Always use these credentials when connecting to the database through the CRUD module.

**Setting up the testing class:**

Before we start using our CRUD module, it's important to test our connection to the database. The following code creates a test instance of our AnimalSanctuary class and attempts to connect to the MongoDB server. This ensures that our setup is correct before we proceed with any database operations."

### 

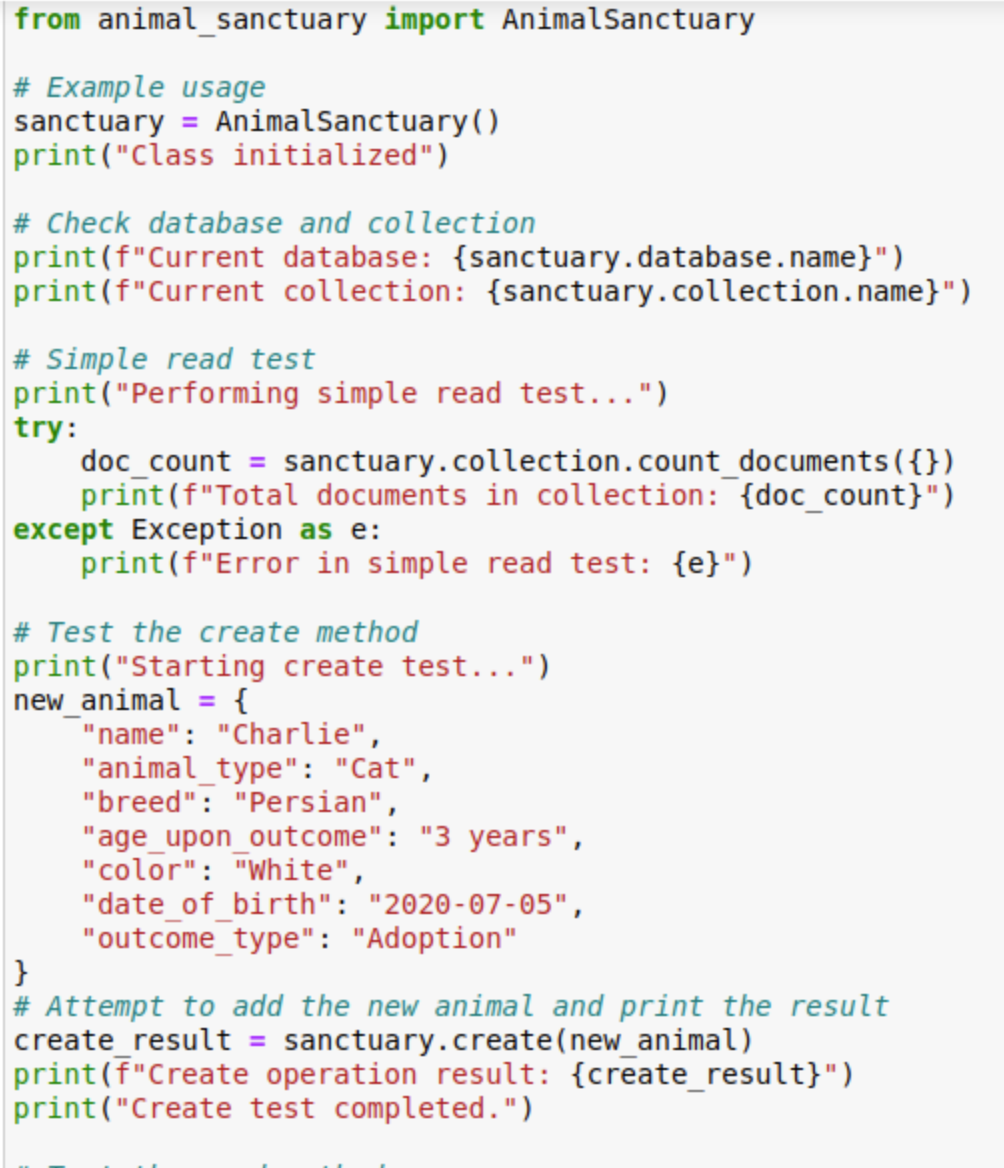
Now that we've successfully connected to our database and created a new record, let's test the other CRUD operations. The following code demonstrates how to read existing records, update them with new information, and delete records from our database. These operations are essential for maintaining an up-to-date and accurate animal sanctuary database.

### Tests:

To run tests for the Animal Sanctuary CRUD Module, follow these steps:

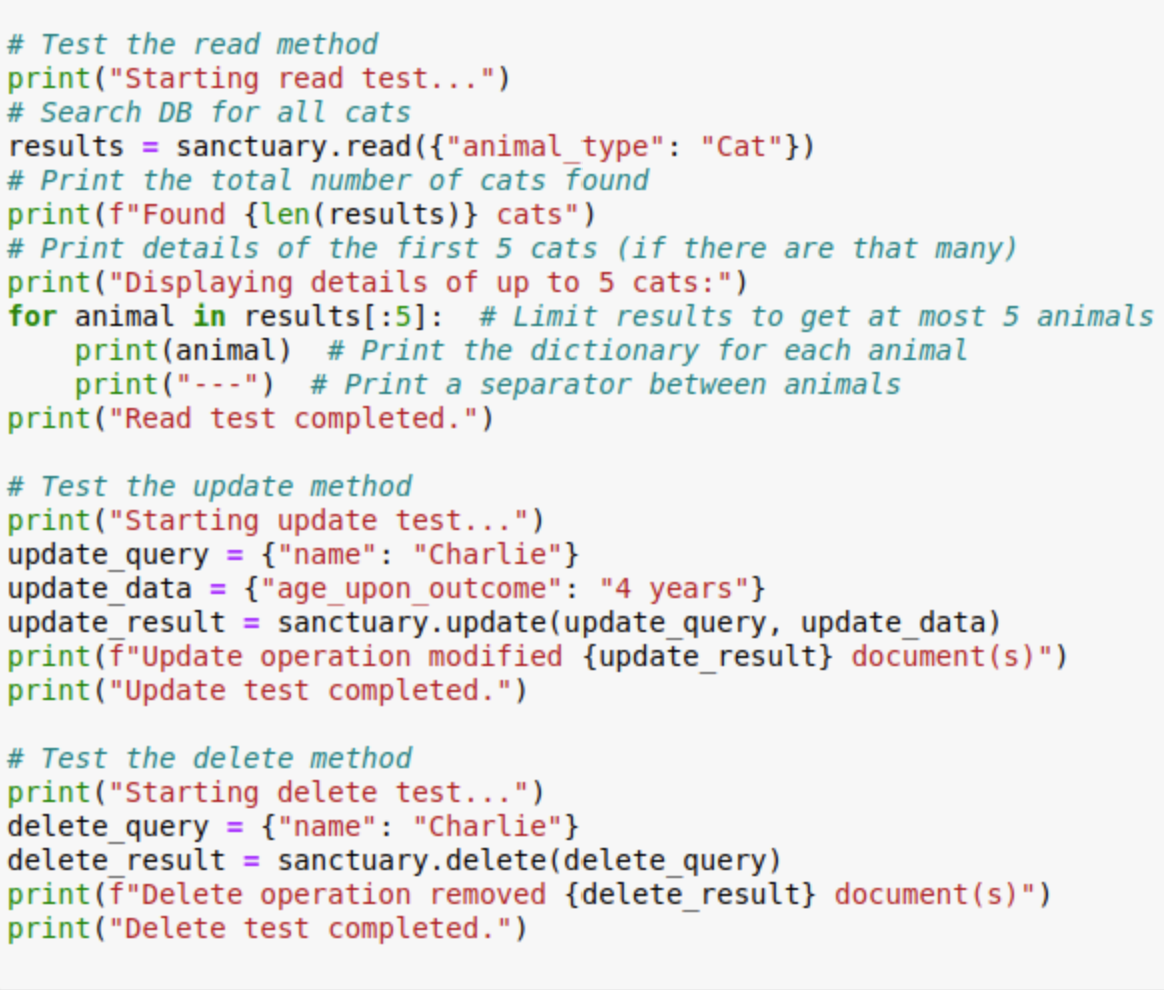
1. Ensure you have pytest installed (*pip install pytest* if you don't).
2. Create a file named *test\_animal\_sanctuary.ipynb* in the same directory as your module.
3. Add the following test code:

**Create Test:**

**

This test adds a new animal record to our database. If successful, it will return True, indicating that the record was created.

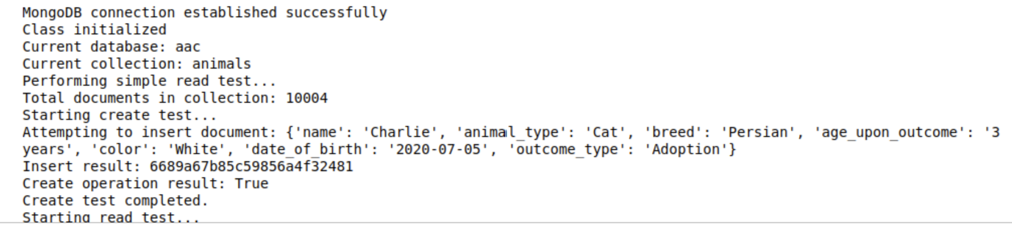
**Read/Update/Delete Test:**

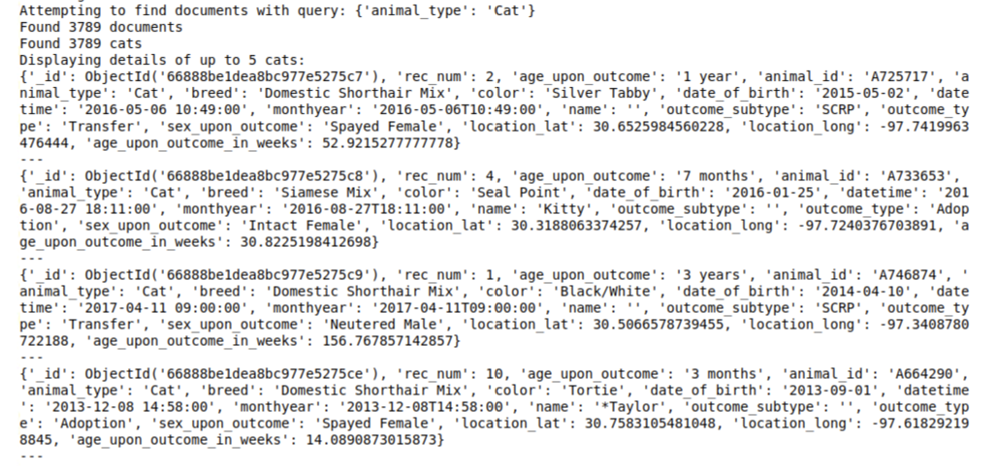
****

### These tests demonstrate the remaining CRUD operations. We first read all cat records, then update a specific record, and finally delete it. The output will show the number of records affected by each operation.

### Screenshots of successful test:

The following screenshots demonstrate what successful test outputs should look like. When running your own tests, compare your output to these examples. Look for similar success messages and record counts to ensure your CRUD module is functioning correctly.







**Grazioso Salvare Dashboard – Extending off Animal Sanctuary CRUD:**

The Animal Sanctuary CRUD Module has been extended with a dynamic web dashboard for Grazioso Salvare. This dashboard allows users to filter and visualize animal data for rescue operations interactively.

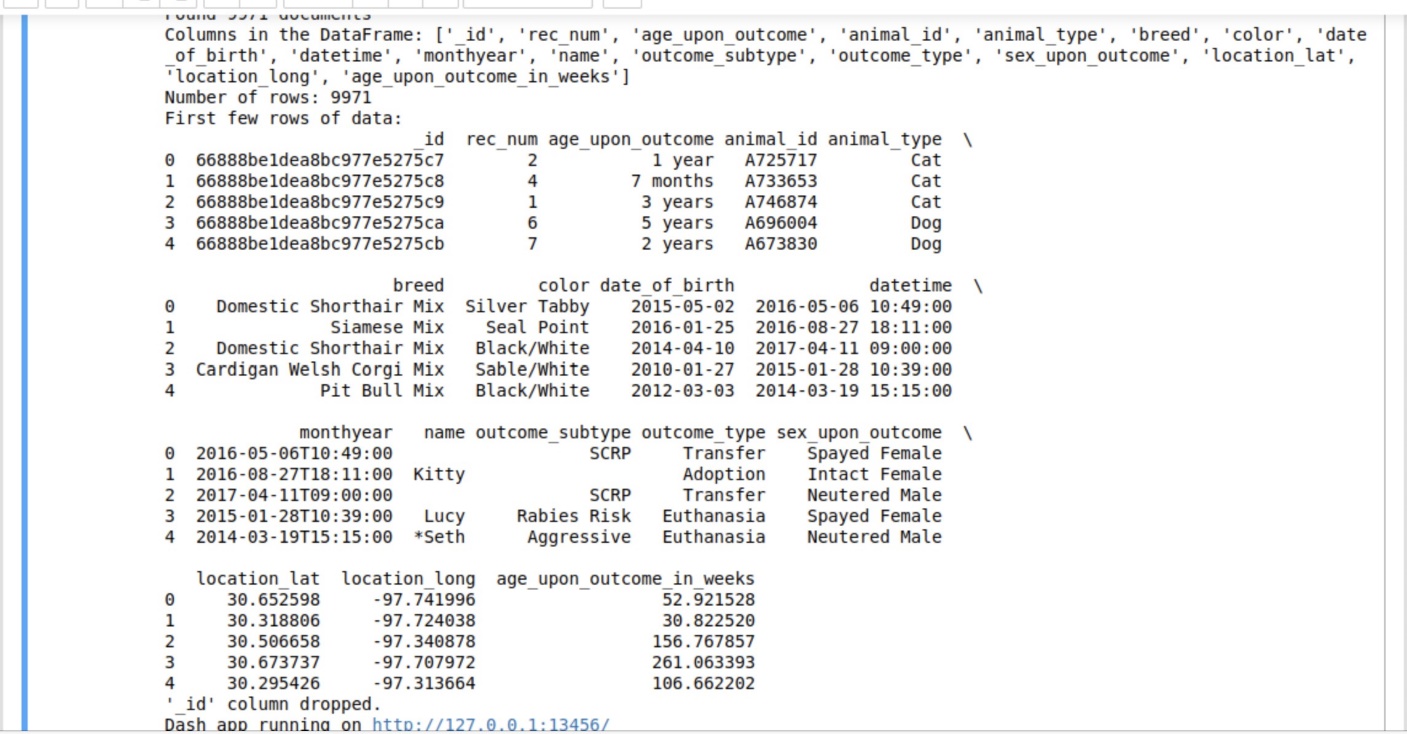
**Functionality**

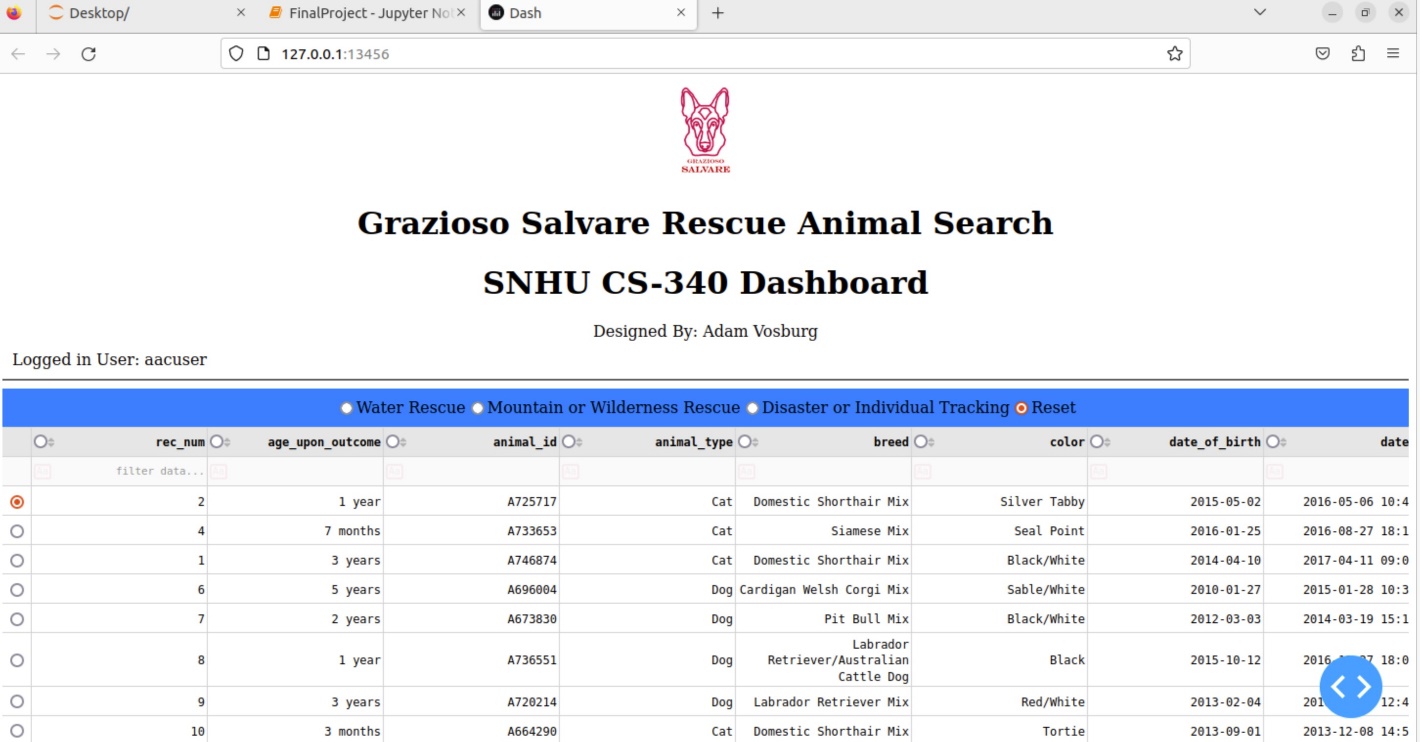
The Grazioso Salvare Dashboard includes the following key features:

* Interactive data filtering based on rescue types:
* **Water Rescue:** Filters for breeds like Labrador Retriever Mix, focusing on intact females aged 26 to 156 weeks.
* **Mountain/Wilderness Rescue:** Selects breeds such as German Shepherd, targeting intact males aged 26 to 156 weeks.
* **Disaster/Individual Tracking**: Includes breeds like Doberman Pinscher, looking for intact males aged 20 to 300 weeks
* Dynamic data table displaying filtered animal records, showing animal ID, breed, sex upon outcome, color, age, and rescue type.
* Geolocation map pinpointing the location of selected animals, providing a visual representation of where each animal was found or is located.
* Pie chart visualizing the breed distribution for each selected rescue types, offering insights into the prevalence of different breeds within each rescue category.

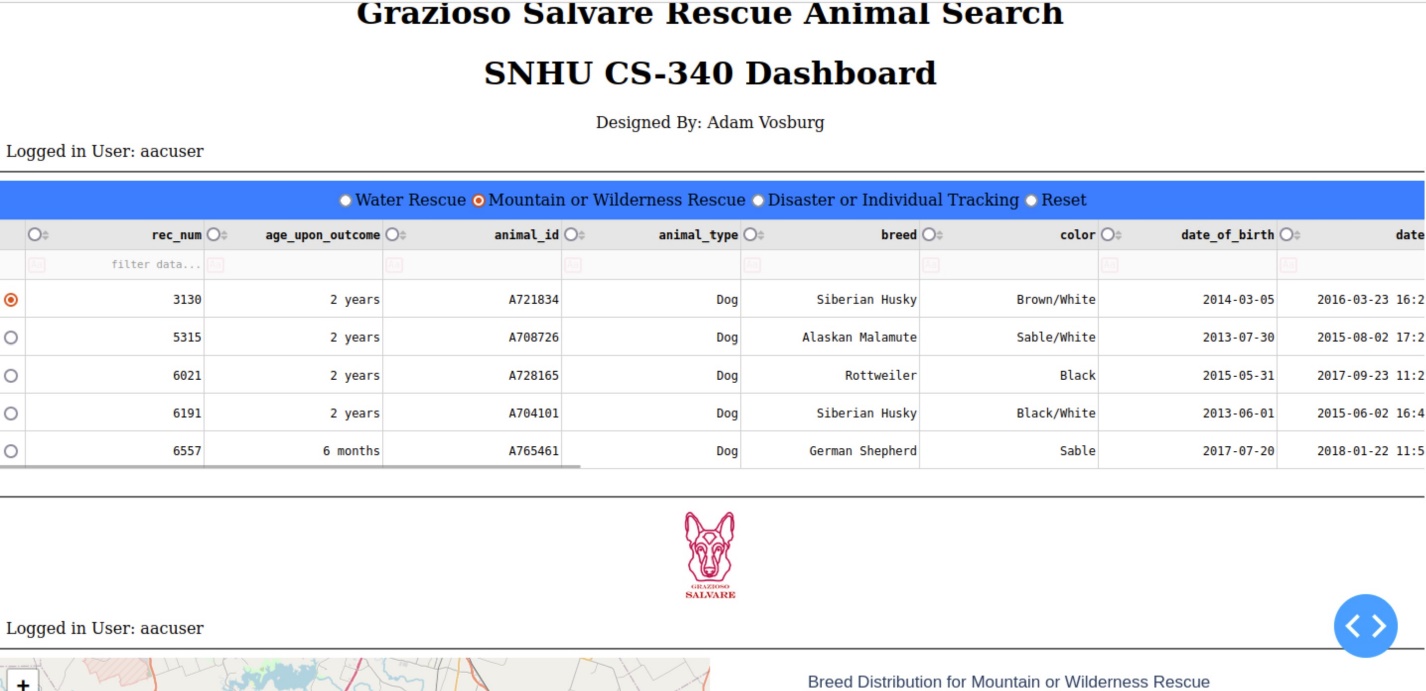
**Screenshots demonstrating the dashboard functionality:**

***Code output***

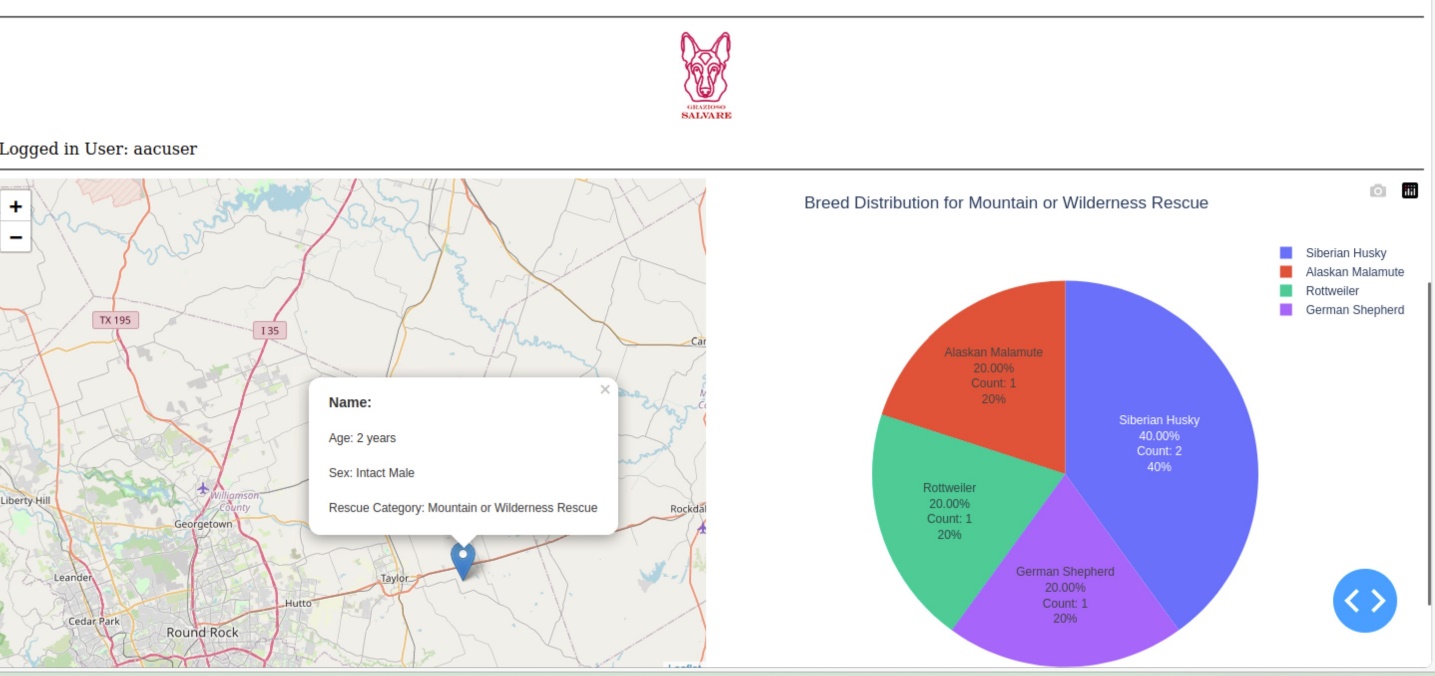
****

***Main login and user information***

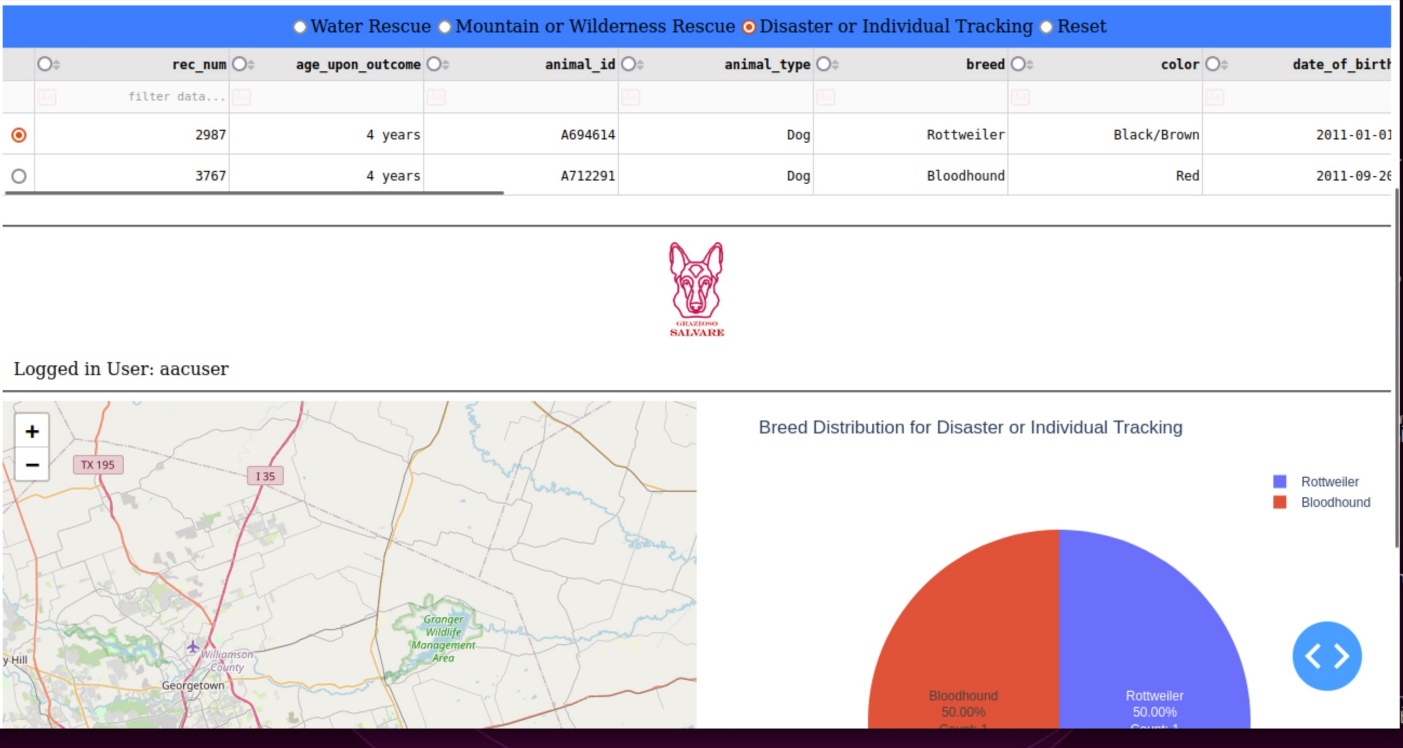
***Data table with rescue type “Mountains or Wilderness” radio button selected***

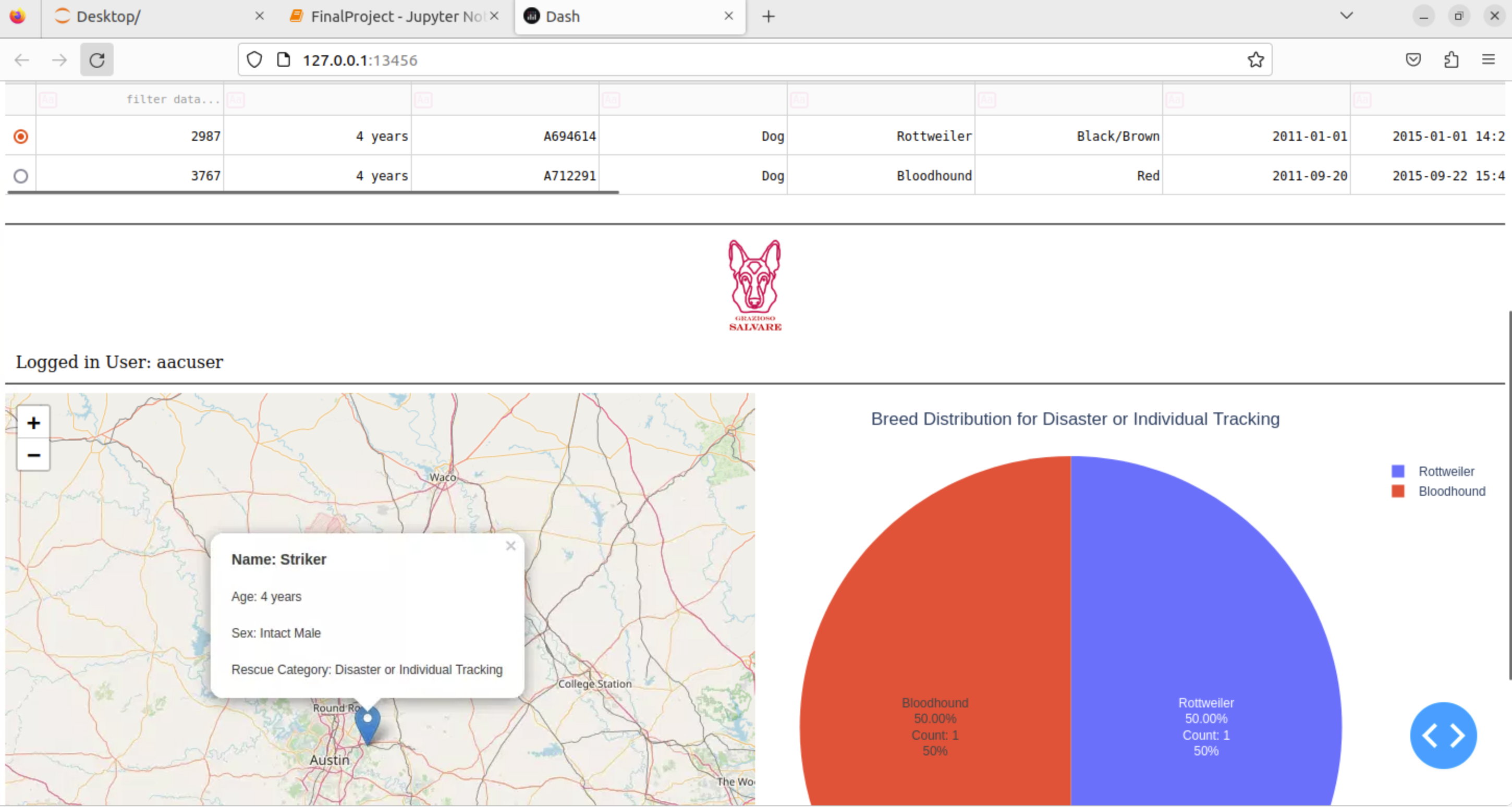
****

***Interactive map label pins show selected animal as well as pie graph results***

****

***Pie graph and map change with new animal selection***

****

****

**Tools and Technologies:**

In addition to the tools used for the CRUD module, the dashboard incorporates:

* + Dash: A Python framework for building web applications. Dash allows for creating interactive, web-based analytic applications entirely in Python
  + Plotly: Used in tandem with Dash to create interactive visualizations, particularly the pie chart in this project.
  + Dash Leaflet: An extension for Dash that provides mapping capabilities, used to create the geolocation chart.
  + MongoDB continues to serve as the database, chosen for its flexibility with document-based storage and excellent integration with Python through the PyMongo driver.

**Resources:**

* [Dash Documentation] found at (https://dash.plotly.com/dash-core-components)
* [Dash Datatable] found at (https://plotly.com/datatable)
* [Dash Leaflet Documentation] found at (https:// dash-leaflet-docs.onrender.com)

**Implementation Steps:**

After setting up the MongoDB database, importing a new user, and developing the CRUD Python module for database operations, I began creating a new Jupyter Notebook for the dashboard implementation. The steps that follow outline the process:

1. Set up the MongoDB database and imported the animal data, ensuring all records were properly structured and indexed for efficient querying.
2. I developed the CRUD Python module for database operations, including functions to filter animals based on specific rescue criteria.
3. I created a new Jupyter Notebook (FinalProjectDashboard.ipynb) for the dashboard implementation and set up the necessary imports and configurations.
4. I Implemented data loading and preprocessing using Pandas, including the creation of a 'rescue\_type' column based on breed and age criteria.
5. I designed the dashboard layout using Dash components, structuring the UI with filters, data tables, map, and chart areas.
6. I developed interactive filtering functionality using Dash callbacks, allowing real-time data updates based on user selections.
7. I created the dynamic data table using Dash DataTable, configuring it to display relevant animal information and respond to filters.
8. I implemented the geolocation map using Dash Leaflet, plotting animal locations and adding pop-ups with animal details.
9. Developed the pie chart for breed distribution using Plotly, including logic to group less common breeds for clearer visualization.
10. I integrated all components and implemented callback functions for interactivity, ensuring smooth data flow between different parts of the dashboard.
11. Finally, I tested the dashboard thoroughly and made necessary adjustments, including performance optimizations and UI/UX improvements such as better aligning text and changing color schemes.

**Challenges and Solutions:**

* **Data Preprocessing:** Ensuring data consistency and handling missing values was challenging. I managed this by coding in data cleaning and using Pandas for efficient data manipulation.
* **Performance Optimization:** With a large dataset, ensuring quick response times for filtering and visualization was imperative. This was achieved by optimizing database queries and using efficient data structures to effectively pull data succinctly.
* **Map Integration:** Integrating the geolocation map with dynamic data was initially confusing. However, after further research into Dash Leaflet documentation, the provided examples helped me overcome this challenge.
* **Interactive Filtering:** Implementing real-time filtering that affected multiple components simultaneously required careful planning of callback structures in Dash. This was resolved by designing an efficient callback chain.
* **Visualization Design:** Creating meaningful and easily interpretable visualizations, especially for the breed distribution, was a bit challenging. I addressed this by implementing grouping for less common breeds and adding detailed hover information.

## Contact:

Adam Vosburg