**Grazioso Salvare – Search-and-Rescue Dashboard**

**About the Project / Project Title**

This project implements an interactive dashboard for Grazioso Salvare that enables users to query, filter, and visualize Austin Animal Center (AAC) Outcomes data to identify dogs suitable for search-and-rescue training. It follows the MVC architecture: MongoDB as the model (data store and queries), Dash UI components as the view (DataTable, geolocation map, and a secondary chart), and Python callbacks + the Project One CRUD module as the controller to connect user inputs to MongoDB queries.

The application provides rescue-type filters (All, Water Rescue, Mountain/Wilderness, and Disaster/Tracking), along with optional breed multi-select, sex-upon-outcome multi-select, and an age-in-weeks range slider. The DataTable initially presents an unfiltered view of all records and supports pagination, sorting, and native column filtering; it also enforces single-row selection with the first row selected by default to keep the geolocation chart in sync. The Leaflet map centers on the selected animal’s coordinates (using location\_lat and location\_long) and shows the breed in a tooltip with the animal’s name in a popup; the second chart (a pie chart) updates dynamically to reflect the current table view (for instance, showing the top breeds by count).

**Motivation**

Grazioso Salvare requires a client-facing tool that reduces manual review, supports intuitive filtering by Rescue Type, Breed, Sex, and Age in Weeks, and visualizes the results. This dashboard centralizes those tasks into a single, easy-to-use interface with location mapping and dynamic summaries.

Branding requirements are met by displaying the Grazioso Salvare logo and a unique identifier with the author’s name on the dashboard header. To demonstrate functionality (Step 6), screenshots should be included (See Below) showing the initial unfiltered table and branding, applying a filter and seeing the DataTable update, the map centering on the first selected animal with tooltip and popup, and the pie chart reflecting the filtered data.

**Getting Started**

Prerequisites:

* Python 3.9+
* JupyterLab or Jupyter Notebook
* MongoDB with the aacuser account and AAC dataset loaded (from Project One)
* Project One CRUD module: AnimalShelter
* Packages: dash, jupyter-dash, dash-leaflet, plotly, pandas, numpy, pymongo

**Installation**

1) Clone or download this repository.

2) Ensure MongoDB is running and accessible using the aacuser credentials.

3) (Optional) Create and activate a virtual environment.

pip install dash jupyter-dash dash-leaflet plotly pandas numpy pymongo

4) Open ProjectTwoDashboard.ipynb in JupyterLab and run all cells to launch the app.

**Usage**

Use the Rescue Type radio buttons (All, Water Rescue, Mountain/Wilderness, Disaster/Tracking), then optionally refine by Breed, Sex, and Age in Weeks. The DataTable will update automatically. Selecting a row re-centers the Leaflet map on that animal and updates the pie chart to reflect the visible data. The dashboard header displays the Grazioso Salvare logo and the unique identifier.

**Code Example**

Example rescue-type query builder:

def build\_rescue\_query(rescue\_type):  
 if rescue\_type == 'WATER':  
 return {  
 'animal\_type': 'Dog',  
 'breed': {'$in': ['Labrador Retriever Mix', 'Chesapeake Bay Retriever', 'Newfoundland']},  
 'sex\_upon\_outcome': {'$in': ['Neutered Male', 'Spayed Female']},  
 'age\_upon\_outcome\_in\_weeks': {'$gte': 26, '$lte': 156}  
 }  
 return {}

Example DataTable data update callback:

@app.callback(Output('datatable-id','data'), Input('filter-type','value'))  
def update\_dashboard(filter\_type):  
 query = build\_rescue\_query(filter\_type)  
 records = db.read(query)  
 return pd.DataFrame(records).drop(columns=['\_id'], errors='ignore').to\_dict('records')

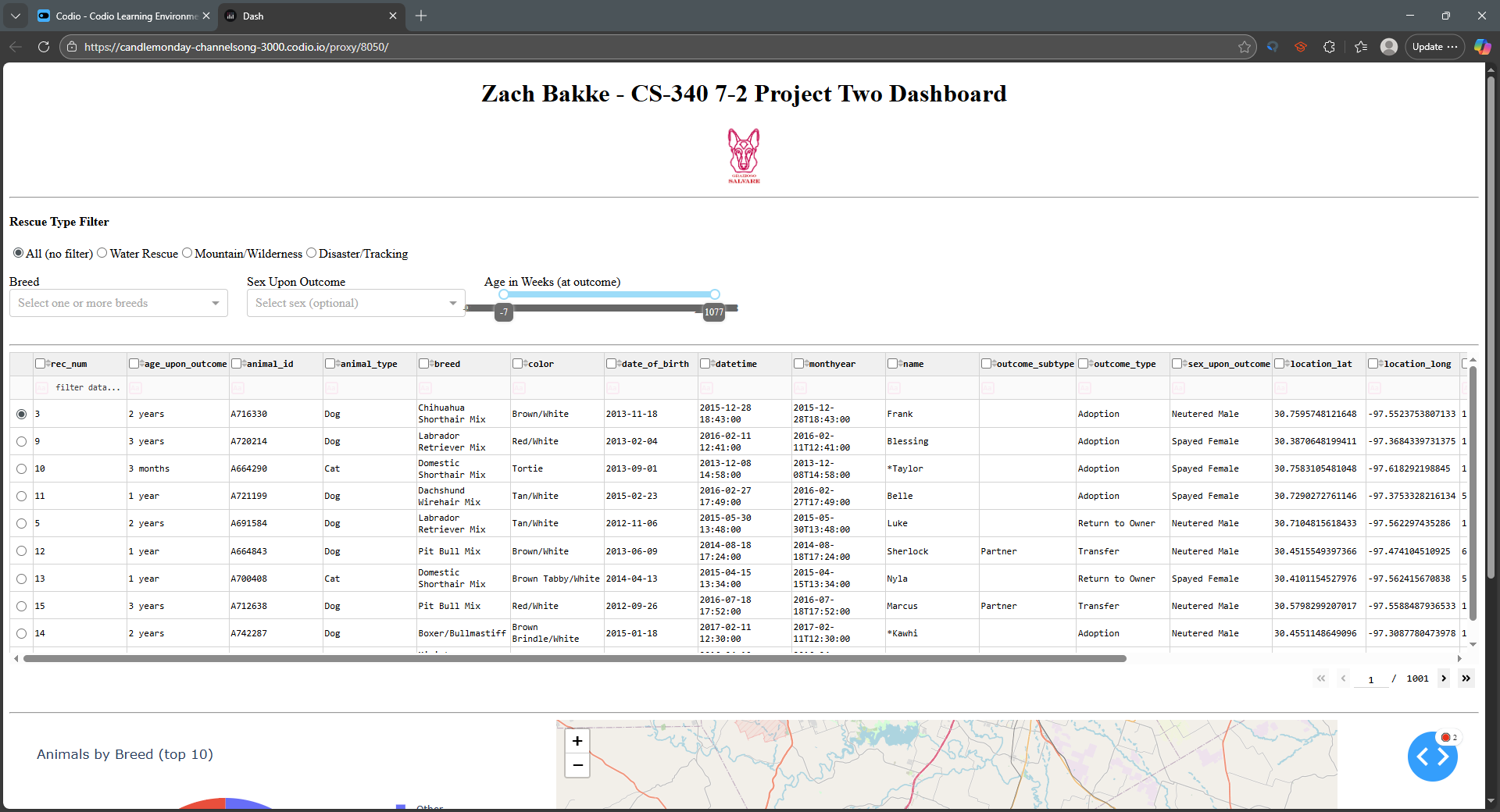
**Tests**

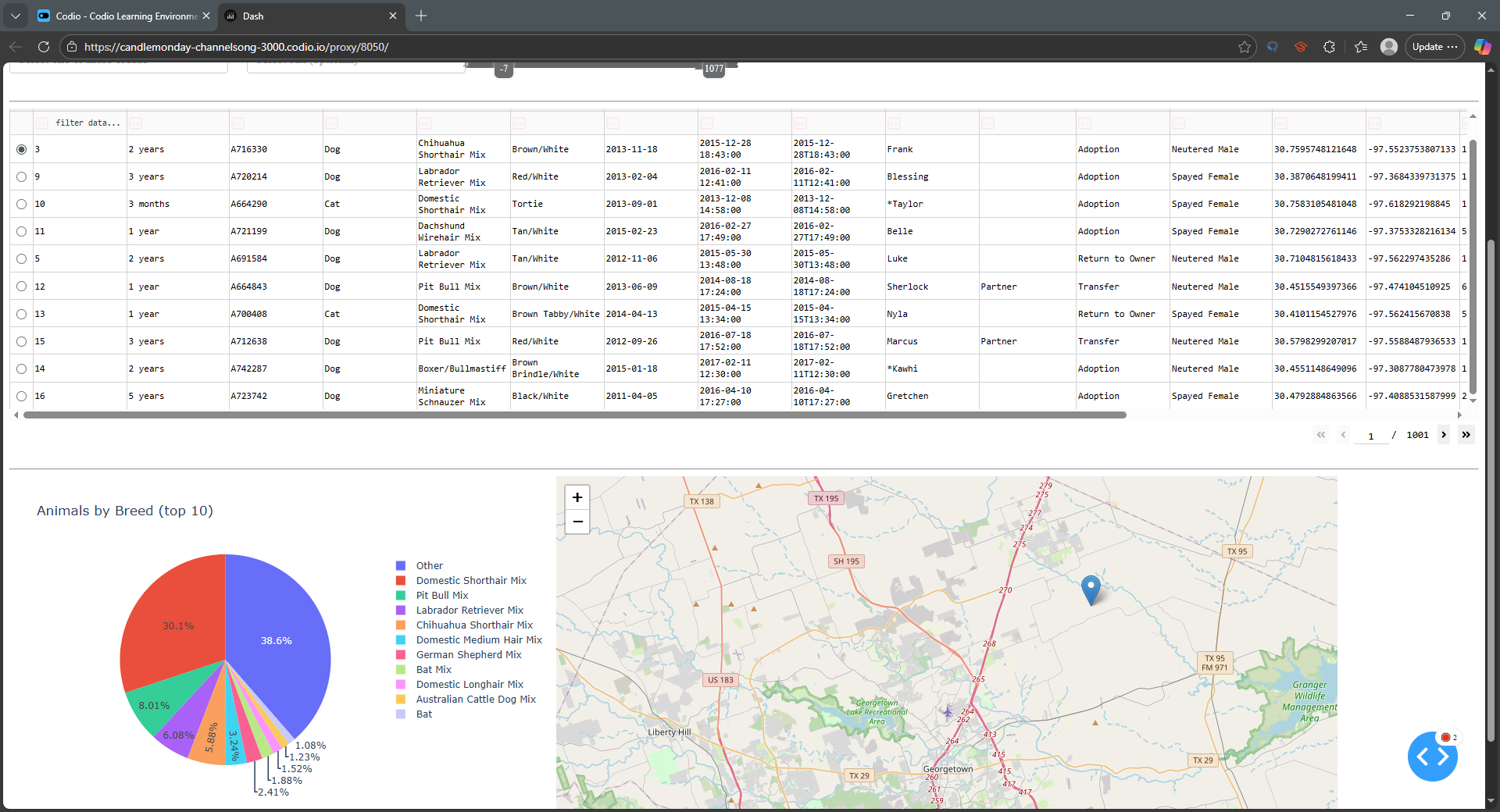
Verified initial MongoDB data load; server-side filtering by rescue type, breed, sex, and age; DataTable updates on control changes; map centers on selected row and shows tooltip/popup; pie chart reflects the DataTable’s derived view; and no console or callback errors during interactions.

**Screenshots**

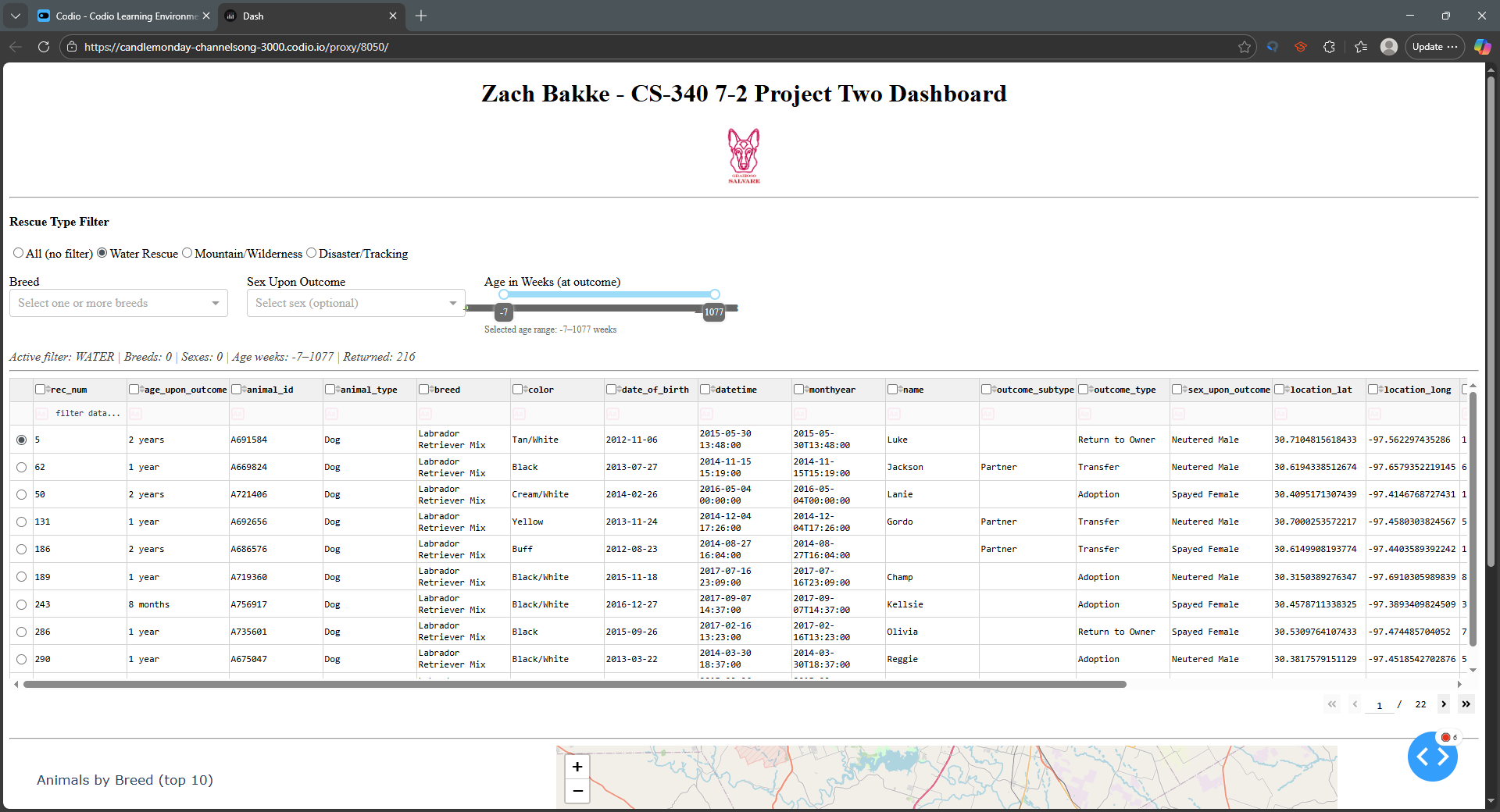
Each of your screenshots or your screencast should contain the **Grazioso Salvare logo** and your **unique identifier**. Your screenshots or screencast must show the following:

* The starting state of your dashboard, which should include your widgets for the **interactive options to filter data (such as radio items or drop-downs), the interactive data table, and the charts**

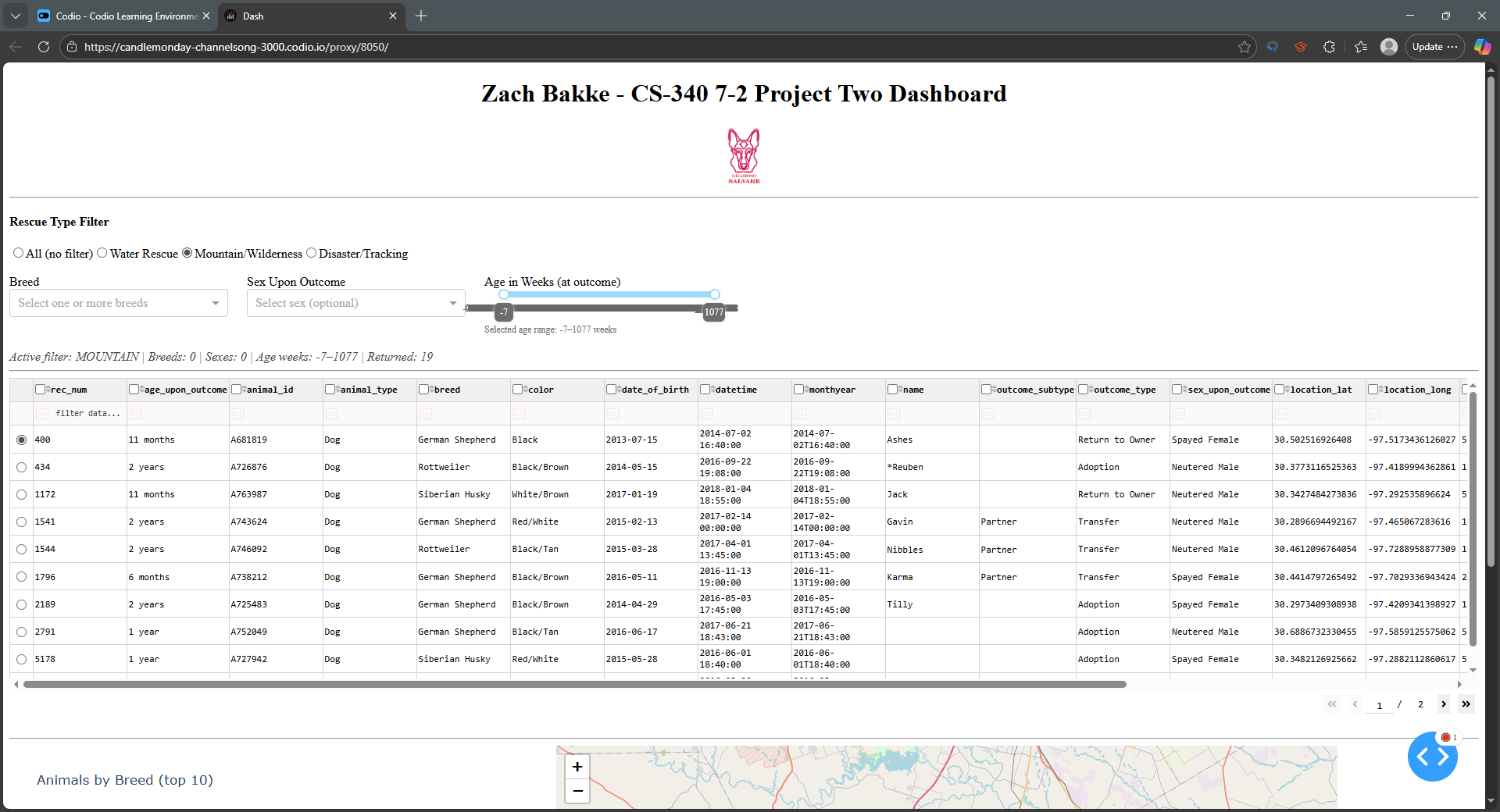


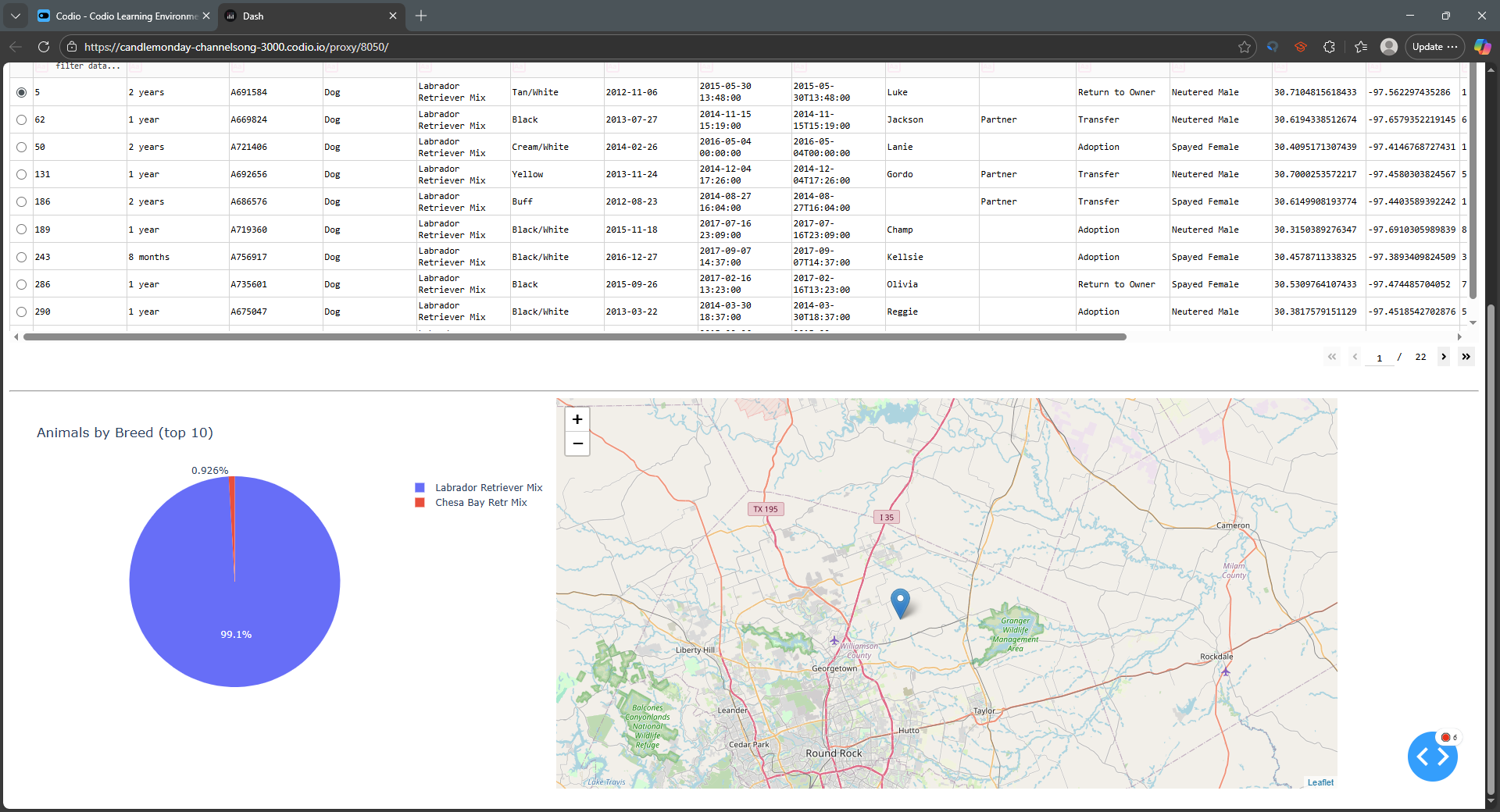


* Executions of your dashboard, showing the widgets after **each** of the following data filters has been applied (four screenshots total):
  + Water Rescue

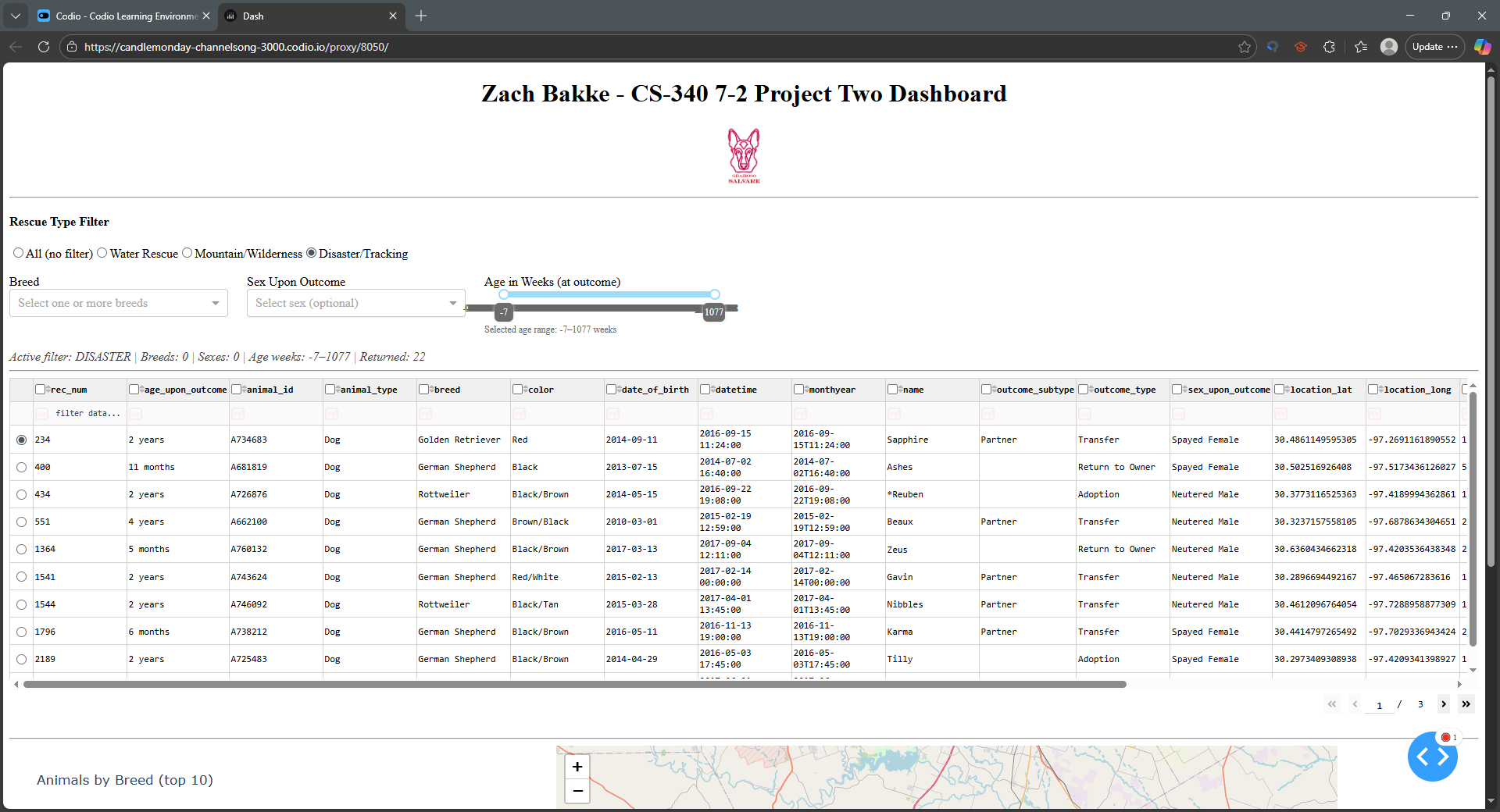


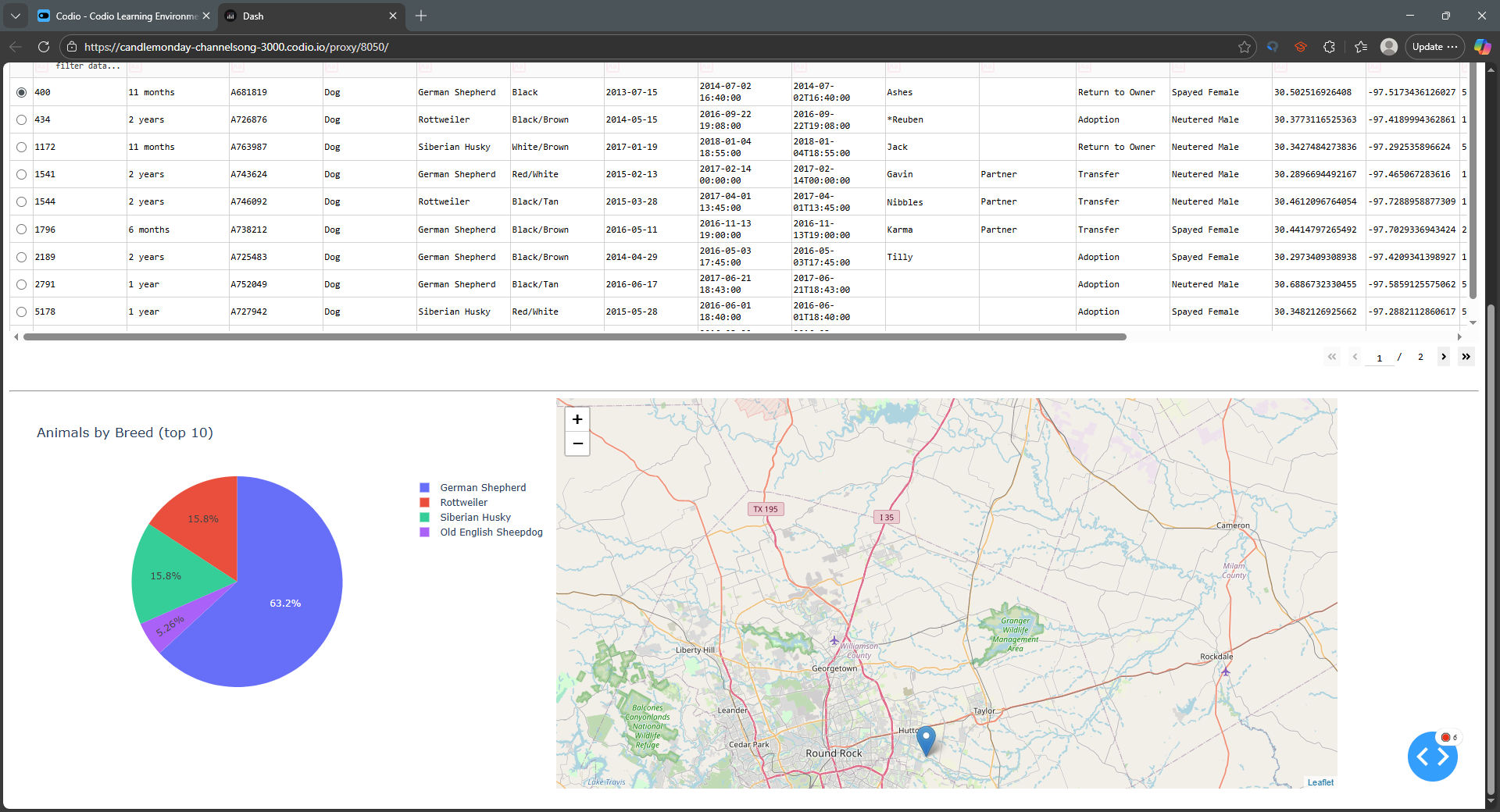
* + Mountain or Wilderness Rescue

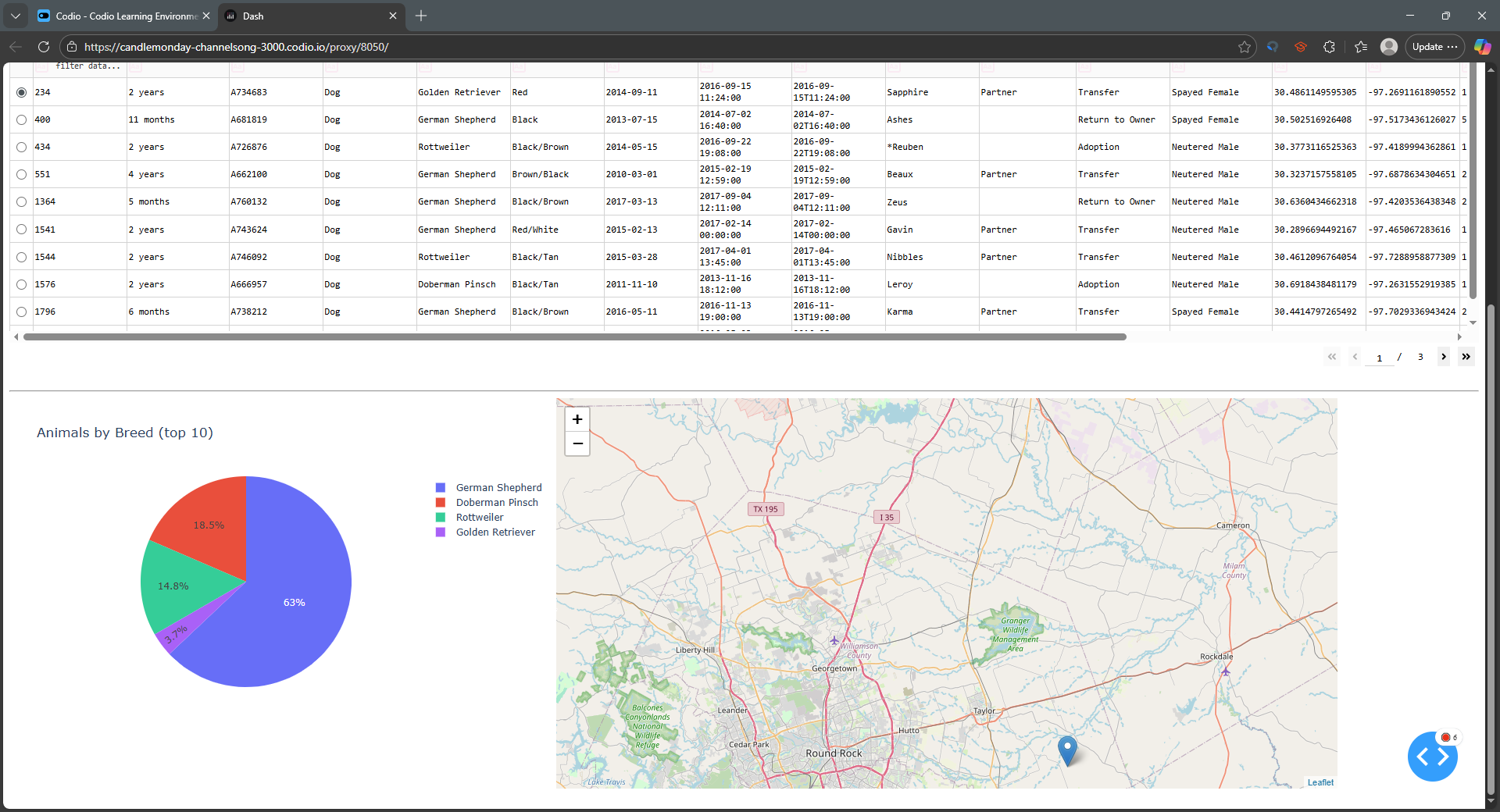




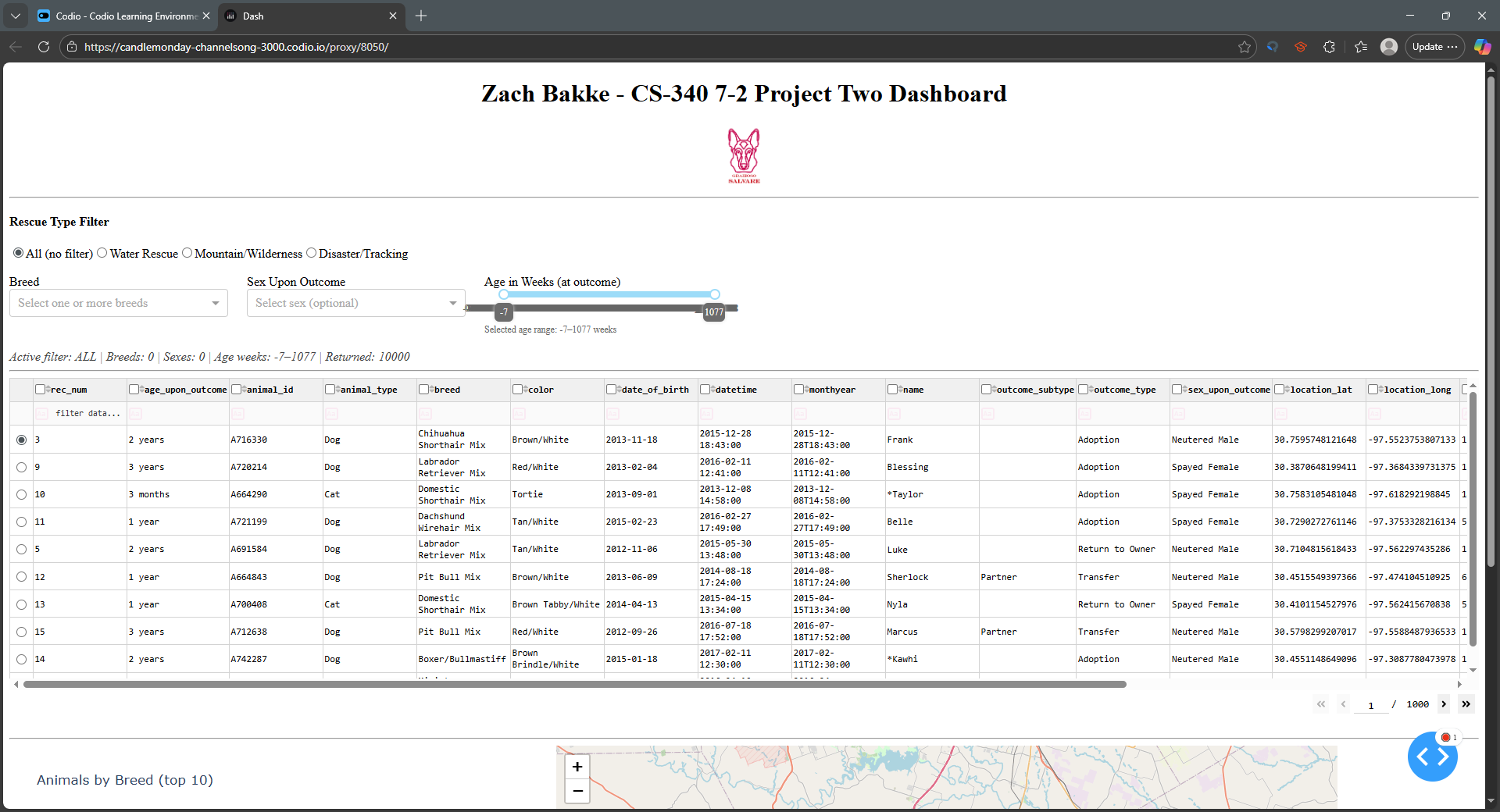
* + Disaster or Individual Tracking

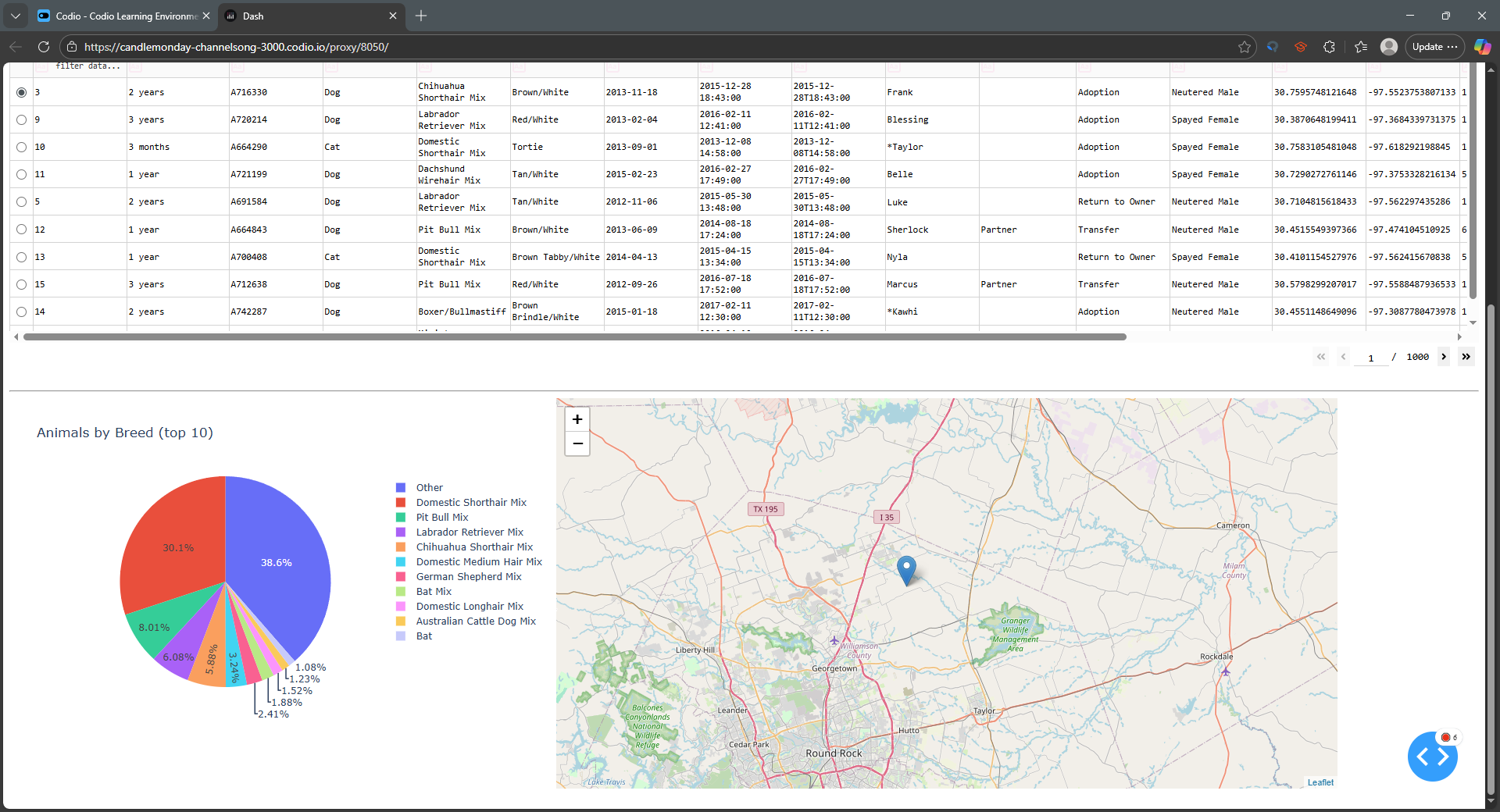






* + Reset (returns all widgets to their original, unfiltered state)





**Contact**

Your name: Zach Bakke

Additional Information and Previous ReadMe Files are below.

7-2 Additional Information

The solution uses Python and JupyterLab for iterative, reproducible development; MongoDB is chosen as the model because its document-oriented, JSON-like structure maps directly to Python dictionaries and Pandas DataFrames, enabling flexible schemas for fields like breed, name, location\_lat, location\_long, and derived metrics such as age\_upon\_outcome\_in\_weeks. The PyMongo-backed CRUD module (AnimalShelter) encapsulates connection and queries and returns lists of dictionaries that are easily rendered in Dash. Dash is used for both view and controller components since it enables building full-featured, interactive web applications in pure Python without bespoke JavaScript.

Core components include dash\_table.DataTable for tabular exploration (with pagination, sorting, and native filters), dcc.RadioItems, dcc.Dropdown, and dcc.RangeSlider for intuitive control of filters, dcc.Graph (via Plotly Express) for the dynamic chart, and dash\_leaflet for the geolocation map (tile layer, marker, tooltip, and popup).

The controller is realized through Dash callbacks: the main filter callback constructs a MongoDB query from user inputs (rescue type baseline intersected with optional breed, sex, and age constraints), updates the DataTable’s data property, and posts a concise filter summary and age readout; a second callback renders the pie chart from the DataTable’s derived view so it reflects sorting, pagination, and client-side filters; a third callback renders the Leaflet map and always returns at least a base map (centered on Austin) to ensure the chart area is never blank, then adds a marker with tooltip and popup when valid coordinates are present.

To reproduce and run the project, ensure Python 3.9+ and MongoDB are available with the aacuser account and the AAC Outcomes dataset loaded into the expected database and collection from Project One. Open ProjectTwoDashboard.ipynb in JupyterLab, confirm the AnimalShelter CRUD module points to your MongoDB (either via a no‑argument constructor that hardcodes credentials, per assignment, or a constructor accepting username and password), and run all cells.

The app can run inline using app.run\_server(mode='inline', debug=True) or externally in a browser tab if preferred. Use the rescue-type radio buttons and optional breed, sex, and age filters; verify the DataTable updates, then select a row to see the Leaflet map recenter and the pie chart update based on the visible table data.

During development, several challenges were addressed: Dash DataTable raised a prop-type error when attempting to persist unsupported properties such as page\_size, which was resolved by persisting only allowed properties (page\_current, sort\_by, filter\_query, and selected\_rows); AnimalShelter constructor mismatches (no-argument vs. (username, password)) caused TypeError, which was resolved by aligning instantiation with the actual signature or by providing a top-level try/fallback; initial map rendering issues were mitigated by always returning a base map and switching from fragile positional indexing to robust, named fields for latitude, longitude, breed, and name; and slider tooltip readability problems in some Jupyter themes were handled by using integer bounds, enabling always-visible tooltips, and adding a separate readout below the slider.

The repository should include a brief set of screenshots demonstrating initial load, applying the use of each filter, the map selection behavior, and the pie chart update; optional files include requirements (CS-340 Dashboard Specifications Document) for dependency management adaptation as requested by the client.

For reference and deeper understanding of the tools used, see Dash’s documentation and component guide at <https://dash.plotly.com/>, Plotly Express at <https://plotly.com/python/plotly-express/>, dash-leaflet examples at <https://dash-leaflet.herokuapp.com/>, and MongoDB/PyMongo documentation at <https://www.mongodb.com/docs/> and <https://pymongo.readthedocs.io/>.

**5-1 ReadMe (Version 2)**

* An **explanation of the purpose of the CRUD Python module**

The AnimalShelter CRUD module is a reusable Python component that provides database access to MongoDB for managing animal records in support of Grazioso Salvare’s mission to identify and categorize dogs for search‑and‑rescue training. The module encapsulates Create, Read, Update, and Delete operations against the aac.animals collection, exposing a simple class interface that client-side scripts and notebooks can import. Its purpose is to make it straightforward for developers and analysts to insert new animal documents, query existing records with flexible filters, modify selected documents, and remove unneeded entries, all while following clean coding practices and robust error handling so the code is easy to understand, maintain, and extend.

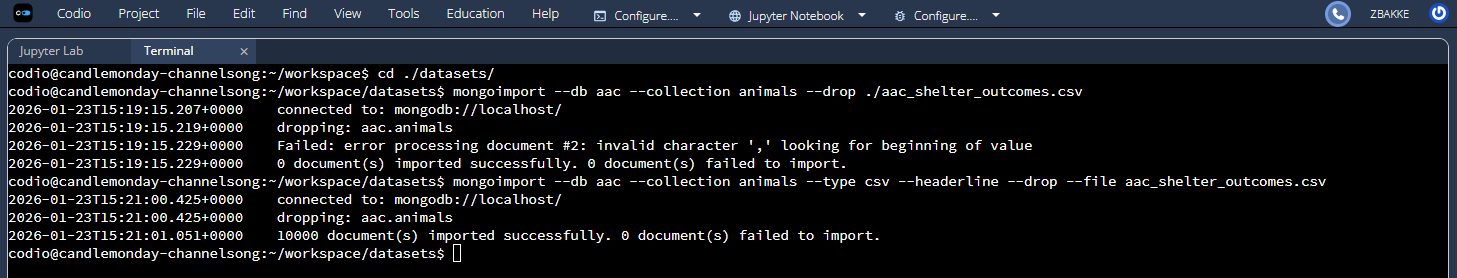
* An **explanation of how the module should be used**, including:
  + A description of the Python driver for Mongo that was used and why it was chosen

This module uses pymongo, the official MongoDB driver for Python, because it is stable, widely adopted, and well-documented, which ensures compatibility with MongoDB features such as insert\_one, find, update\_one/update\_many, and delete\_one/delete\_many. These APIs make it possible to implement the required CRUD behaviors reliably, including cursor handling for reads and clear return contracts. Authentication is configured for a local MongoDB instance using the course-provided credentials (aacuser / CS340zb!) on localhost:27017, with the default database set to aac and the target collection set to animals. If your deployment authenticates against a specific database, the connection string can include an authSource parameter to match your environment. To run the module, ensure MongoDB is installed and running, install pymongo with pip install pymongo, place AnimalShelter.py alongside your scripts or notebooks, and import the class to begin interacting with the database.

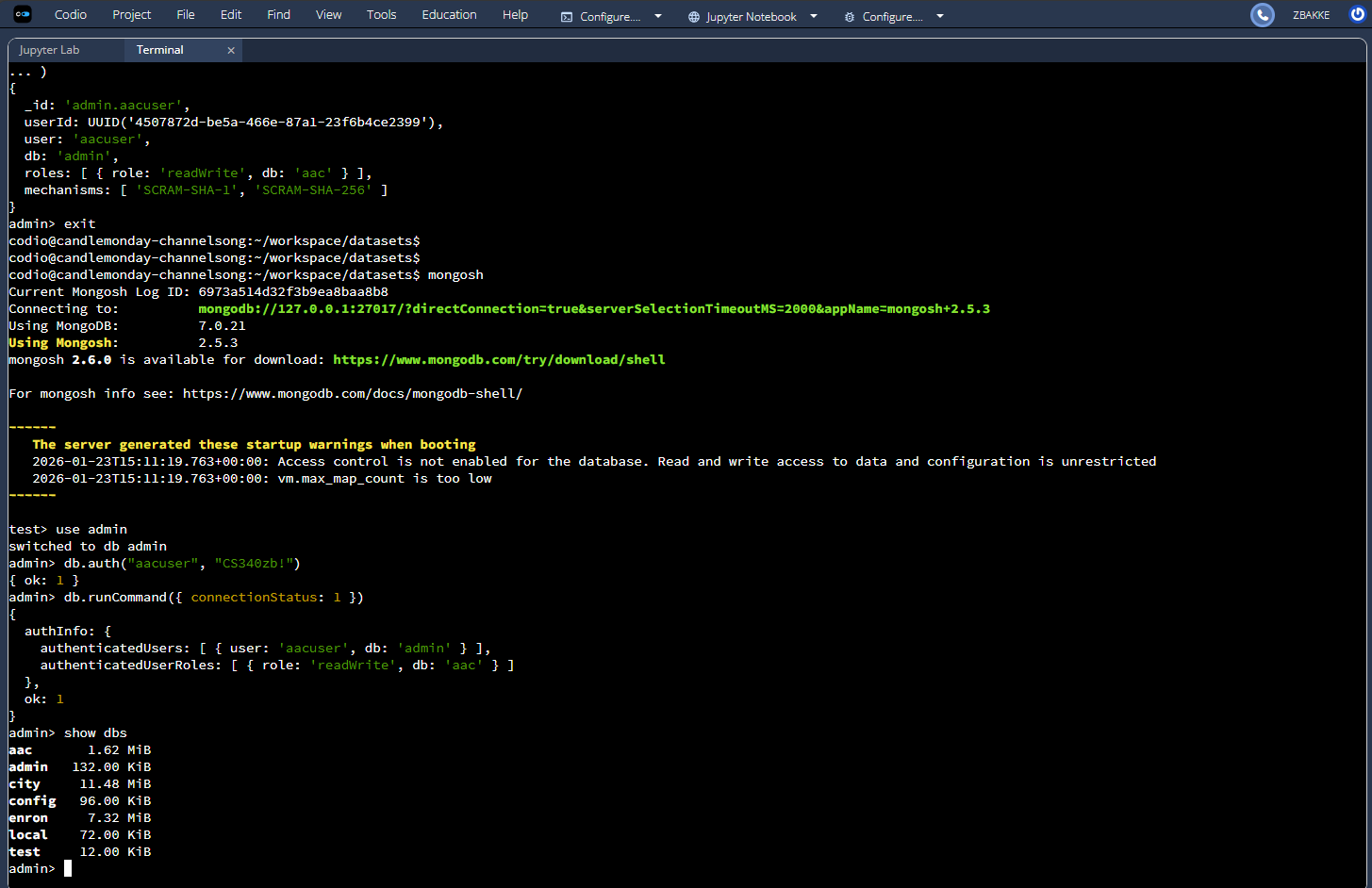
* + An explanation of the attributes and working functionality of the CRUD operations

The core functionality is organized into four methods with clear, predictable behavior. The create method accepts a non-empty dictionary and inserts it into the collection, returning True if the operation is acknowledged and an \_id is generated, or False if the input is invalid or the driver reports an error. The read method fulfills the requirement to use find() rather than find\_one() by issuing a query with a dictionary filter, defaulting to an empty filter when none is provided, and converting the resulting cursor to a list before returning it; it yields an empty list if input is invalid or an error occurs. The update method takes a filter and a set of update values and can modify one or many documents depending on a flag; it automatically wraps plain dictionaries into a $set update document when no update operator is provided and returns the number of documents actually modified. Finally, the delete method accepts a filter and removes one or many documents based on a flag, returning the number of documents deleted. Across all methods, basic validation and exception handling keep callers safe from driver exceptions while providing clear outcomes that higher-level code can test and log.

* A **demonstration of the module’s functional operations**, including:
  + Screenshots of the MongoDB import execution. You took these screenshots in Step 1.



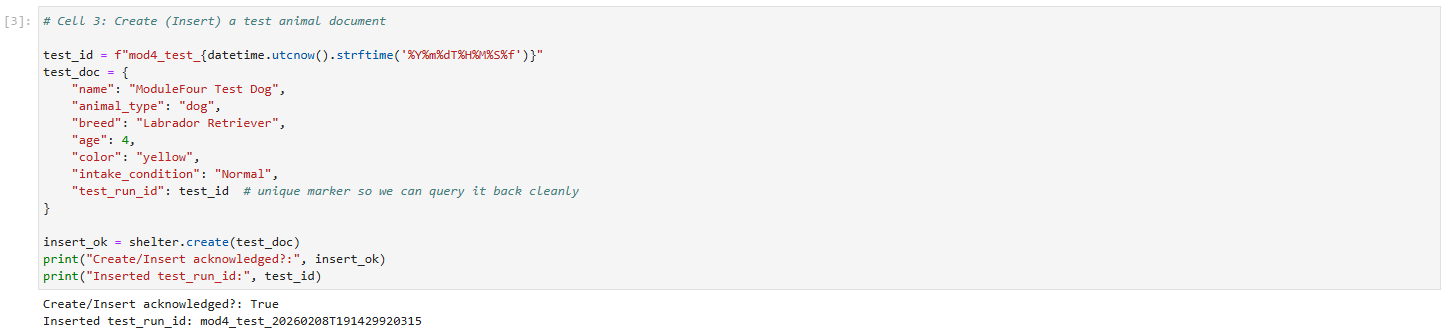
* + Screenshots of the user authentication execution. You took these screenshots in Step 2.

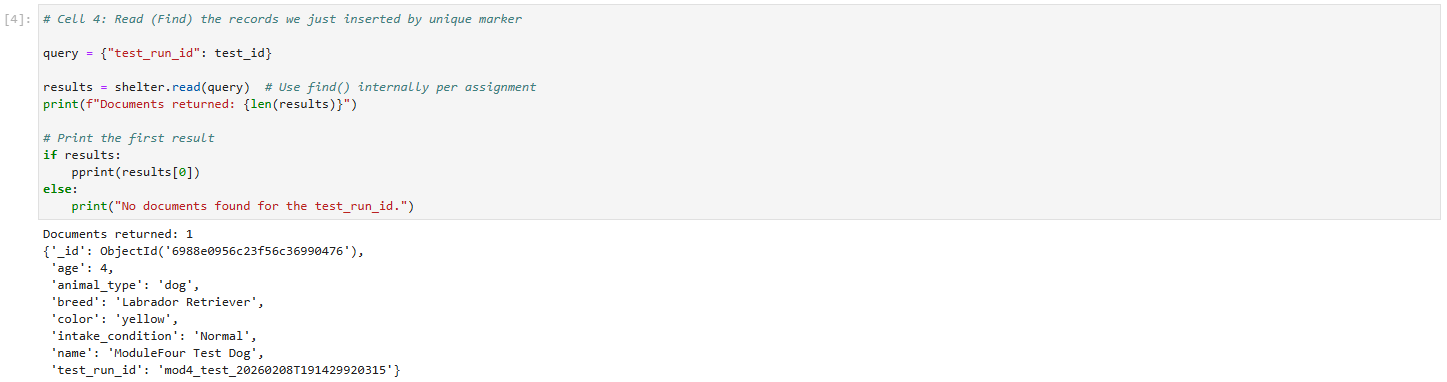


* + Screenshots of the CRUD functionality test execution. You took these screenshots in Step 4.

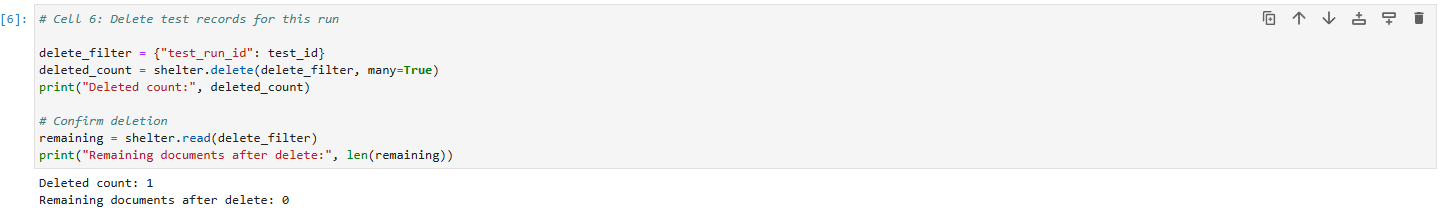












**4-2 ReadMe (Version 1)**

## About the Project/Project Title

The AnimalShelter CRUD Module is a Python-based project designed to implement the Create and Read portions of CRUD functionality using a MongoDB database. The module provides a structured way to insert new animal records into a database and retrieve them through flexible query operations. It serves as a foundation for further development of full CRUD capabilities which will be added in later stages of the larger assignment. This project focuses on clean coding practices, strong documentation, error handling, and designing code that is reusable in database-driven applications.

## Motivation

The motivation behind the creation of this project is to gain practical experience building Python modules that interact with real-world databases in a reliable and professional manner. MongoDB is widely used due to its flexibility and ability to store structured documents without requiring a rigid schema. Implementing the Create and Read functionality allows developers to understand essential data-handling workflows, including document insertion, retrieval with cursors, and secure authentication.

## Getting Started

To get a local copy of the project running, the user must first configure their environment to match the MongoDB setup created in Module Three. This setup includes the creation of the aac database, the animals collection, and the authorized MongoDB user “aacuser” with the password “CS340zb!”. Once MongoDB is running locally, the user can place the AnimalShelter.py file into their working directory and import it into a Python interpreter or Jupyter Notebook. From there, the user can instantiate the AnimalShelter class, which will automatically connect to the database using the stored credentials. The Create and Read operations can then be tested by inserting documents and querying them back through the provided methods.

## Installation

This project requires Python 3, MongoDB Community Server, and the pymongo library. Python serves as the foundation for writing the CRUD module, while pymongo enables direct communication between Python and MongoDB. Jupyter Notebook is used to test the module in an interactive environment, making it easier to execute the code step-by-step and capture screenshots of the results. To install the required tools, the user must first ensure that MongoDB is installed and running on their machine. Next, the user can install pymongo using the terminal command:

*pip install pymongo*

With MongoDB, pymongo, and Python properly configured, the user is ready to run the CRUD module and test its features within Jupyter Notebook.

## Usage

The module is used by importing the AnimalShelter class, creating an instance of it, and then calling the create and read methods to interact with the database. The module connects automatically to the aac database and animals collection using the correct authentication credentials. Once instantiated, the user can insert a document by passing a dictionary to the create method. If the insertion is successful, the method returns True. To retrieve records, the user supplies a dictionary query to the read method, which returns a list of matching documents. In Jupyter Notebook, the output of these operations can easily be displayed for verification and included in the required screenshots for this assignment.

### Code Example

from AnimalShelter import AnimalShelter

*# Instantiate the CRUD module*

shelter = AnimalShelter()

*# Create example*

new\_animal = {

    "name": "Biscuit",

    "species": "dog",

    "age": 4,

    "color": "yellow"

}

insert\_success = shelter.create(new\_animal)

print("Insert successful:", insert\_success)

*# Read example*

results = shelter.read({"species": "dog"})

for record in results:

    print(record)

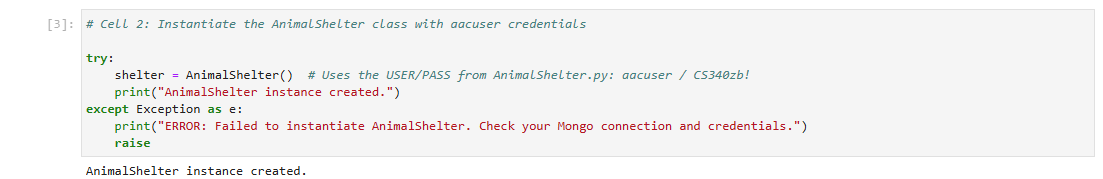
This example demonstrates how the module is used to insert a new animal record and retrieve documents that match a given query. Developers can use this code to quickly verify that their database connection and CRUD module are functioning correctly.

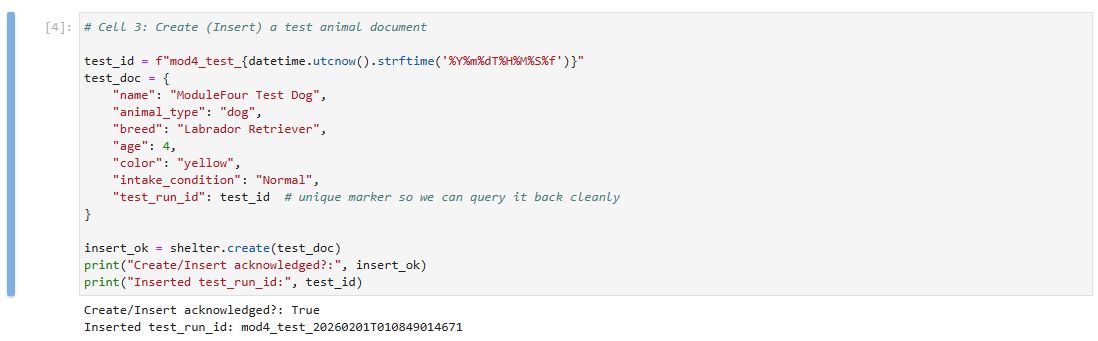
### Tests

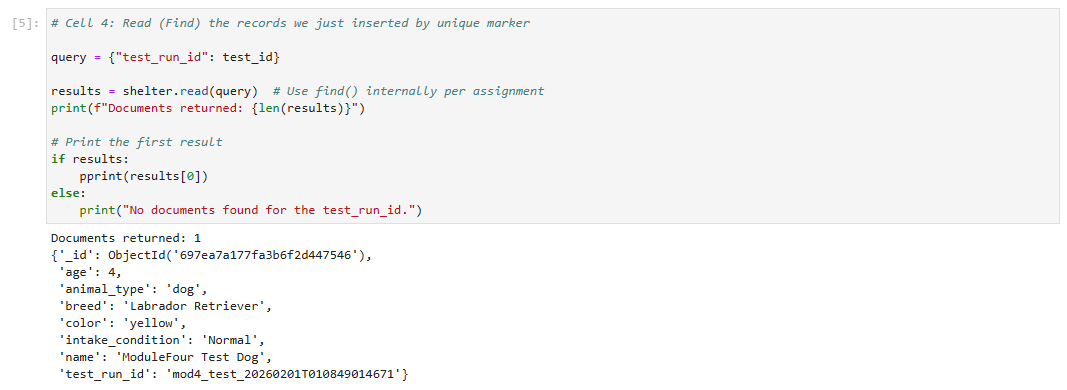
Testing is done using a Jupyter Notebook file such as ModuleFourTestScript.ipynb. In the notebook, the user imports the CRUD module, instantiates the class, performs the create operation, and then retrieves records using the read method. Each test displays the output directly in the notebook, allowing the user to confirm correct functionality.

### Screenshots









## Contact

Your name: Zach Bakke