# CS340 Project Two: AAC Interactive Dashboard Documentation README - Michael Lorenz

## Contents

[Contents](#)

[About the Project](#oytswe2yhlqz)

[AAC Database Interactive Dashboard](#_r8hml27mil2i)

[Motivation](#_x66iew6icbgd)

[Steps to Complete This Project / Getting Started](#_89c5hf4koqo4)

[Challenges During Development](#_ubk069j2kvlg)

[Installation](#daaso2s2r58b)

[Usage / Required Functionality](#mouxun4484r6)

[Starting State - Displays an Interactive dashboard](#_ewbul1z8sg33)

[Filter animals by Water Rescue Type](#_6c8pwdpdbtd7)

[Filter animals by Mountain or Wilderness Rescue Type](#_1x34udwtpwq)

[Filter animals by Disaster or Individual Tracking Rescue Type](#_p2syebojzq79)

[Reset Filters to the Initial State](#_av8c2dbt1tdg)

[Grazioso Salvare Logo](#_x9rmr4dqsgqd)

[Explanation of Widgets](#_vxwedrm9dtq)

[Contact](#4c5dsj9abc7r)

## About the Project

### AAC Database Interactive Dashboard

*Author: Michael Lorenz*

This project facilitates a connection to the MongoDB Animal Control Center database using a web dashboard that utilizes a Python Module to interact with the database, allowing the functionality to filter database entries based on preferences and visualize the resulting information. The dashboard includes functionality to customize the number of entries per page, filter by species, filter by outcomes, and filter by preferred dog occupations as a tool for Grazioso Salvare to filter dogs by breed, sex, and age.

### Motivation

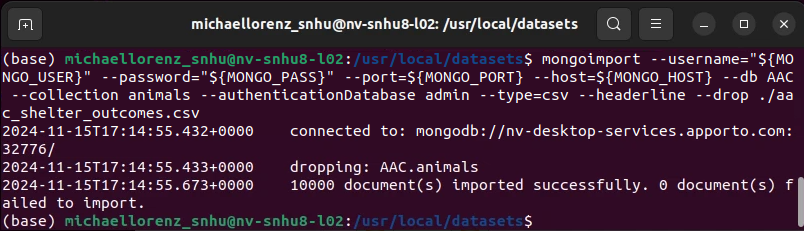
This project was created for CS340 - Client Sever Development and is designed to meet Grazioso Salvare’s requirements to use existing data to identify and categorize available dogs from the AAC animal shelters. This code is being made open-source as requested by the client so that similar organizations may use and adapt it.

## Steps to Complete This Project / Getting Started

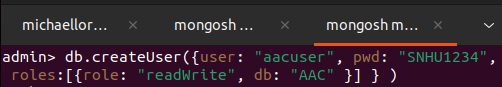
This project was developed with the Model-View-Controller (MVC) architecture. The following process was followed to use this architecture to create this project. Refer to these steps to set up and use the dashboard locally:

1. I completed this project using the tools and project files outlined in the [**Installation**](#daaso2s2r58b) section.
2. Set up the “Model,” representing the data and its interactions.

First, I imported the database and set up a user-authenticated account from the Mongosh shell. This allows the model to have access to the primary source of data. Here is an example import command from the directory containing the dataset. Use an MongoDB admin account and connection to replace username, password, port, and host credentials.



Then, I set up a user account with read/write access to the database.

****

(see [MongoDB user authentication steps](https://www.mongodb.com/docs/manual/tutorial/create-users/#connect-and-authenticate) for more details)

Next, I developed the *AnimalShelter* Python module to serve as the model layer to abstract database interaction using the PyMongo driver. This includes functions for reading, writing, updating, and deleting the data. Include *AnimalShelter.py* in the directory containing the dashboard. The dashboard uses the model by instantiating an object with the proper user and connection credentials.

| shelter = AnimalShelter(username=[USER], password=[PASS], database=[DB NAME], collection=[COLLECTION NAME], host=[HOST NAME], port=[PORT NUMBER]) |
| --- |

([See README for this module for further usage and support](https://docs.google.com/document/u/0/d/12fn147TO1-SVp8pAUu0yGsI52wm34XOO/edit))

1. Set up the “View,” representing the user interface–what the user sees and interacts with

I used the Dash library components of this dashboard to serve as the view, defining the HTML UI components the user sees, such as dropdowns, charts, maps, and tables. This allows me to address requirements to display the data table and required widgets and allows the user to view all the data visually, providing a user-friendly experience. Use AACInteractiveDashboard.ipynb to access the view.

1. Set up the “Controller” to connect the model and view

To update the view and enable interactivity, the controller uses user input from the dash components to process them and update the view based on data retrieved from the model. To do this, I used callbacks activated by user input from the Dash components, such as the data filters and row selection, to call functions from the Python module Model layer with customized arguments from the input to query for specific criteria from the database. This allowed me to address the client's need to filter by dog rescue occupation and allow interactivity. Use AACInteractiveDashboard.ipynb to access the controller.

## Challenges During Development

I overcame several challenges during the development of this dashboard.

* **Handling callback input and outputs:** The controller handles input from the user in Dash components to output changes to the view based on results from the model. Utilizing multiple inputs (each different filter) to modify a single output (the data table) was difficult because I could only have one callback for each output. To overcome this, I developed the callback *update\_entries\_shown(),* which gets input when any filter values are changed and aggregates every filter into one query that retrieves data using the AnimalShelter module and returns a new data table with that data.
* **Querying by Dog Breed Requirement:** I initially faced difficulty with the requirement to filter by specific dog breeds for the rescue type filters. As an example, the client requested to filter by breeds such as “Labrador Retriever Mix, “Chesapeake Bay Retriever,” and “Newfoundland” for the Water Rescue type, but the AAC Database also had breed entries such as “Labrador Retriever/ Newfoundland” or other Labrador Retriever mixed with many other breeds. I overcame this by getting clarification on the requirement to use the exact breeds listed.

There were also problems with how breeds were formatted in the database. I had to use the MongoDB command *db.animals.distinct(‘breed’)* to discover that “Chesapeake Bay Retriever” is actually entered as “Chesa Bay Retr,” and “Doberman Pinscher” is entered as “Doberman Pinsch,” which I had to enter precisely in the dashboard code.

## Installation

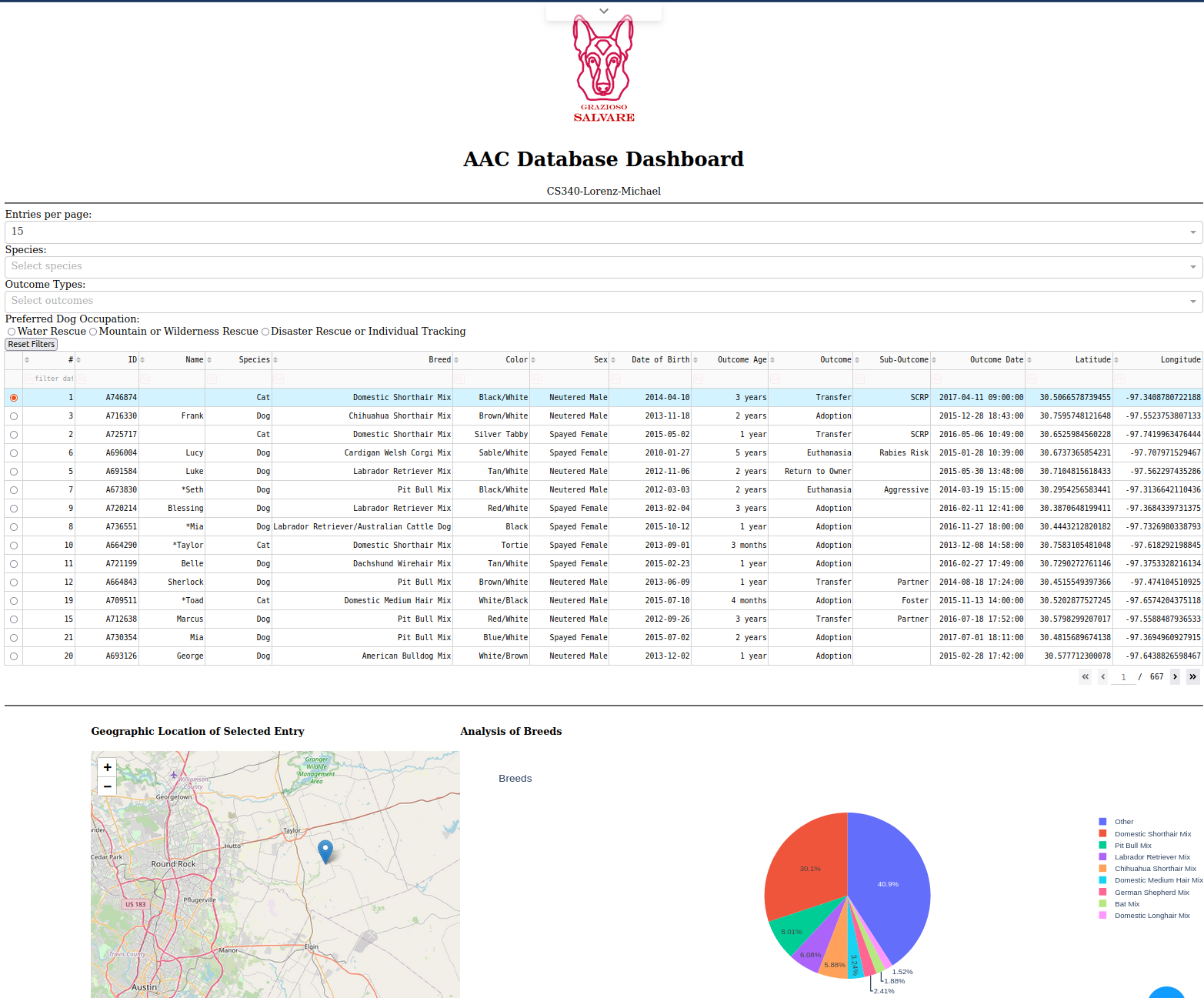
The following software/files/libraries are needed to set up and use the Dashboard:

* AACInteractiveDashboard.ipynb and AnimalShelter.py files - Keep these in the same directory and run AACInteractiveDashboard to operate the Dashboard
* [The Austin Animal Center Outcomes dataset](https://learn.snhu.edu/content/enforced/1748847-CS-340-12877.202486-1/course_documents/aac_shelter_outcomes.csv?isCourseFile=true&_&d2lSessionVal=ScQoRYLN9OTGv4T9RbCyYwie6&ou=1332057&ou=1748847) (or other data imported into a database). Find import instructions in [**Steps to Complete This Project / Getting Started**](#_89c5hf4koqo4)
* [Python 3.9.12](https://www.python.org/downloads/release/python-3912/) ([downloads](https://www.python.org/downloads/)) - Used as the base language for the dashboard and model module.
* [MongoDB 6.0.13](https://www.mongodb.com/docs/manual/release-notes/6.0/#6.0.13) with [Mongosh 1.8.0](https://github.com/mongodb-js/mongosh/releases/tag/v1.8.0) ([installation instructions](https://www.mongodb.com/docs/manual/installation/)) - Used as the model component to provide the source dataset used by the dashboard. MongoDB was selected because the database design of MongoDB (NoSQL) allows flexibility in using JSON document data structures to store and retrieve data from the dataset, making it seamless with the PyMongo driver.
* [PyMongo](https://pymongo.readthedocs.io/en/stable/) driver 3.12.0 (terminal command pip install pymongo==3.12.0 after Python is installed) - Used by the model layer AnimalShelter.py module to interface with MongoDB database and serve the dashboard’s controller, allowing querying and updating the database directly from Python.
* [Dash 2.8.1](https://github.com/plotly/dash/releases/tag/v2.8.1) with Dash Leaflet 0.1.23 (terminal command pip install dash==2.8.1 and (terminal command pip install dash-leaflet==0.1.23 ) - The Dash framework is utilized as both the View and Controller component of this program and abstracts much of the web development process. Dash is used to build the user interface “View,” which comprises HTML components and widgets such as data tables, maps, and charts. Dash is also the “Controller,” allowing a connection between the PyMongo interface module and the view to allow users to interact with the database from the frontend dashboard.
* [Plotly 5.6.0](https://github.com/plotly/plotly.py/releases/tag/v5.6.0) ( pip install plotly==5.6.0 ) - Used to create visual components such as pie charts within the dashboard
* [Numpy 1.21.5](https://numpy.org/devdocs/release/1.21.5-notes.html) ( pip install numpy==1.21.5 ) - Used for numerical computations
* [Pandas 1.4.2](https://pandas.pydata.org/pandas-docs/version/1.4.2/getting_started/install.html) ( pip install pandas==1.4.2 ) - Provides capability to prepare the dataset for the View to control how the dataset is presented.
* [Matplotlib 3.5.1](https://matplotlib.org/3.5.1/) - Used with Plotly for plotting.
* A Python IDE such as [PyCharm](https://www.jetbrains.com/pycharm/download/?section=windows) or [Jupyter Notebook](https://jupyter.org/install) (optional) - Used as a development environment to modify the dashboard or module.

## Usage / Required Functionality

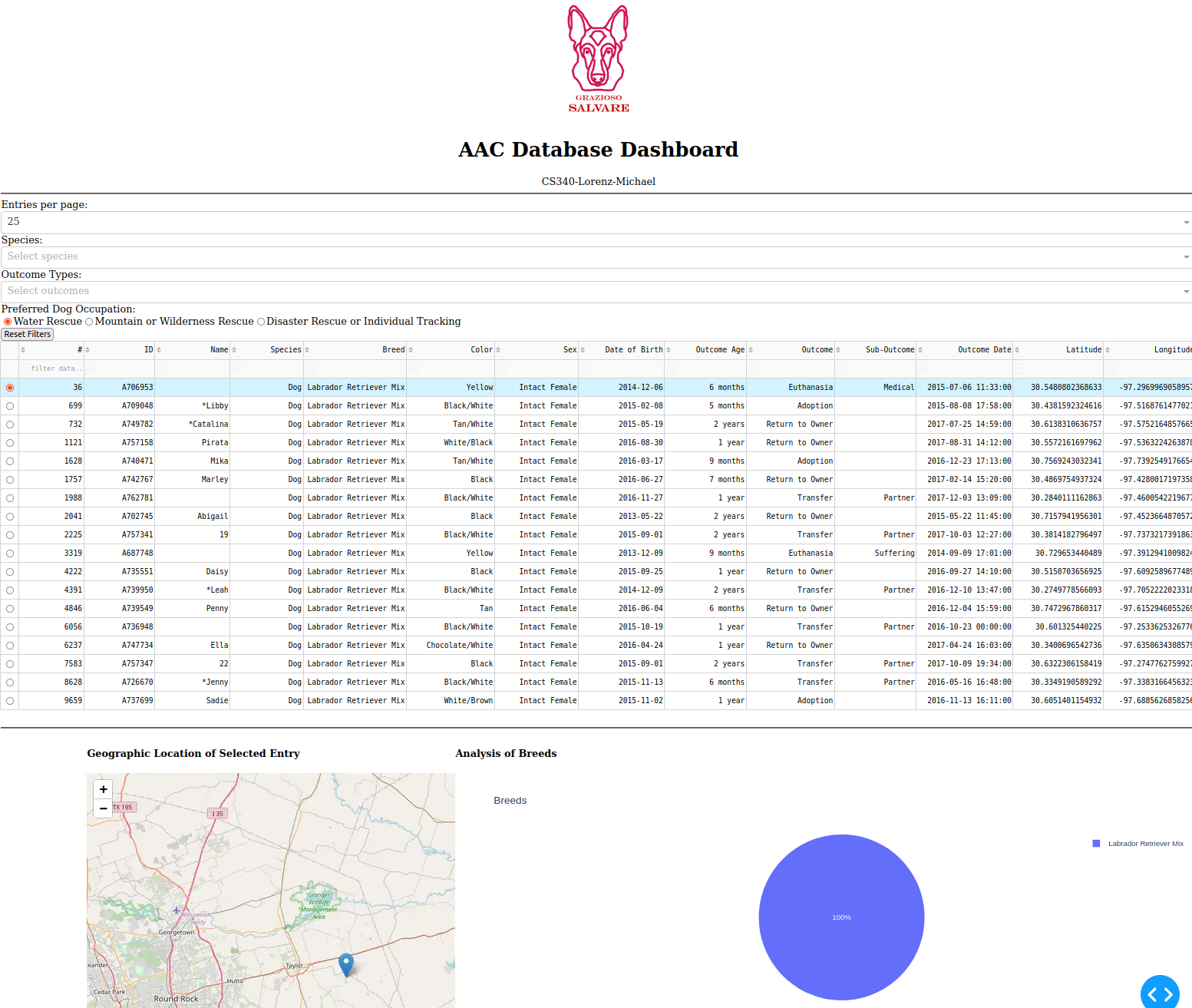
The dashboard was designed to perform the following required functions:

### Starting State - Displays an Interactive dashboard



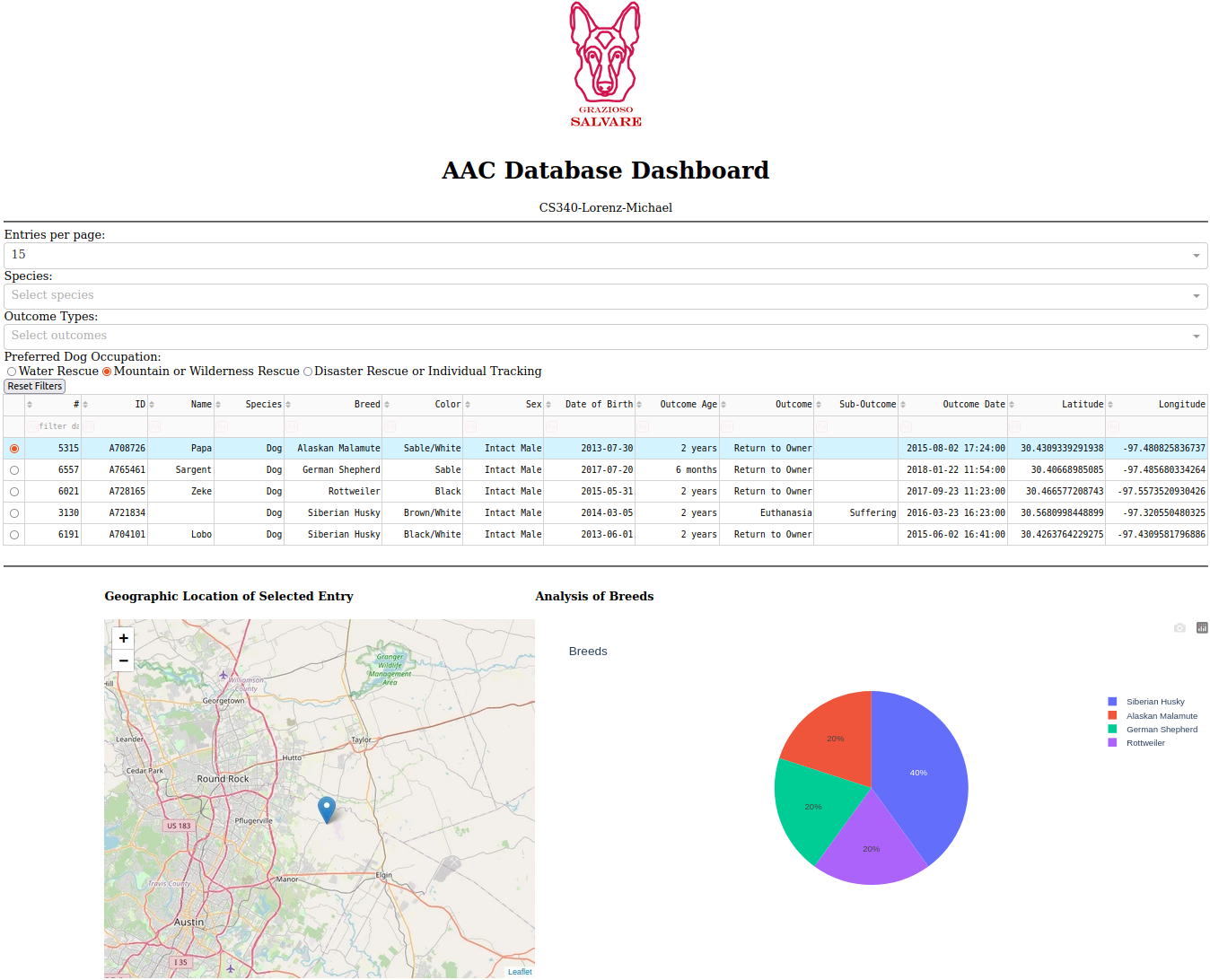
The starting page should show the interactive dashboard in its default state, where filter values are blank by default; the first entry is selected and shown on the geographic map; and the pie chart shows an analysis of breeds based on all data in the data table.

### Filter animals by Water Rescue Type



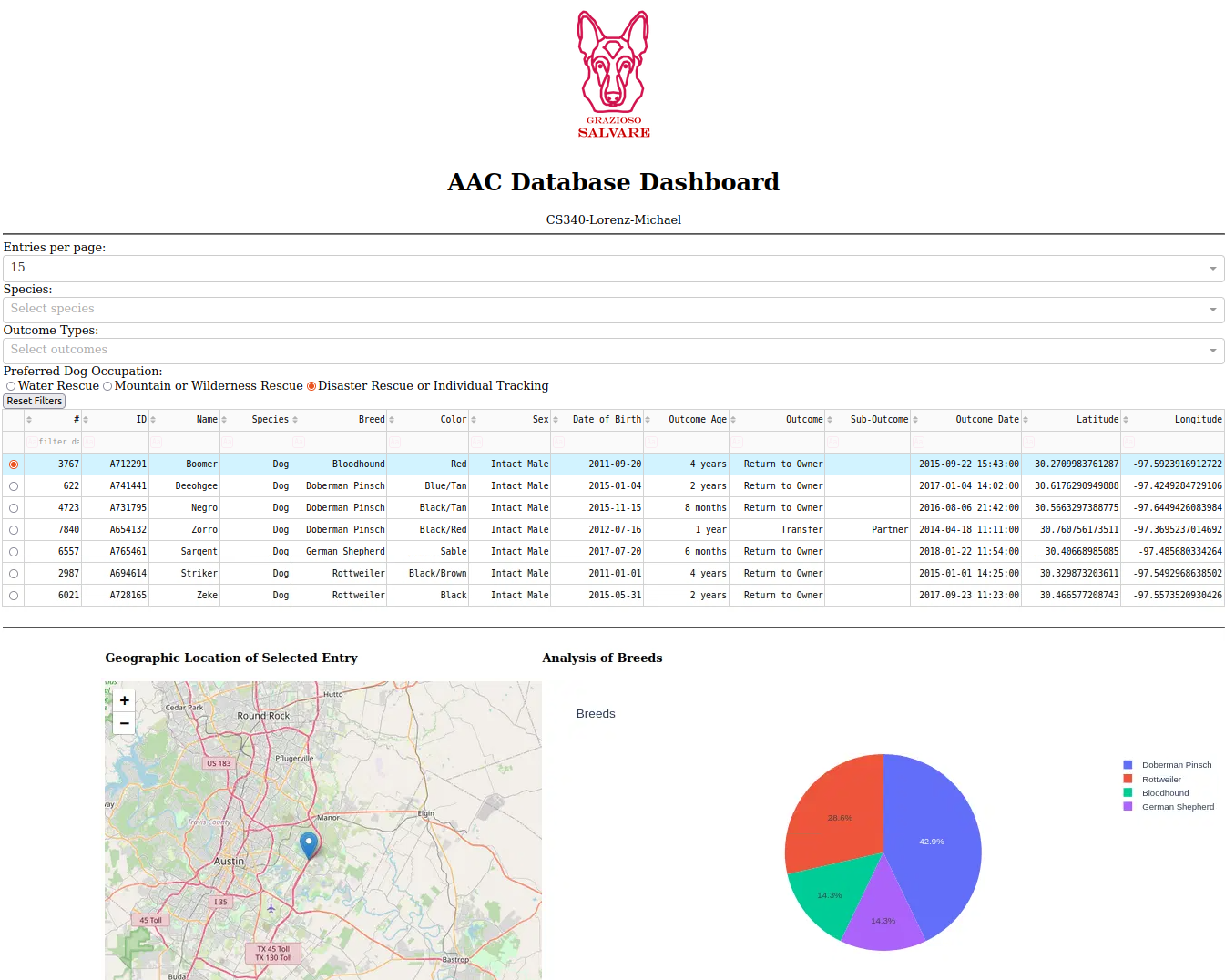
Clicking the “Water Rescue” filter shows dogs with breeds - Lab Retriever Mix, Chesapeake Bay Retriever, and Newfoundland; sex - intact female; age - 25 to 156 weeks. The map and chart are updated to show the selected row location and the proportions of breeds in this query.

### Filter animals by Mountain or Wilderness Rescue Type



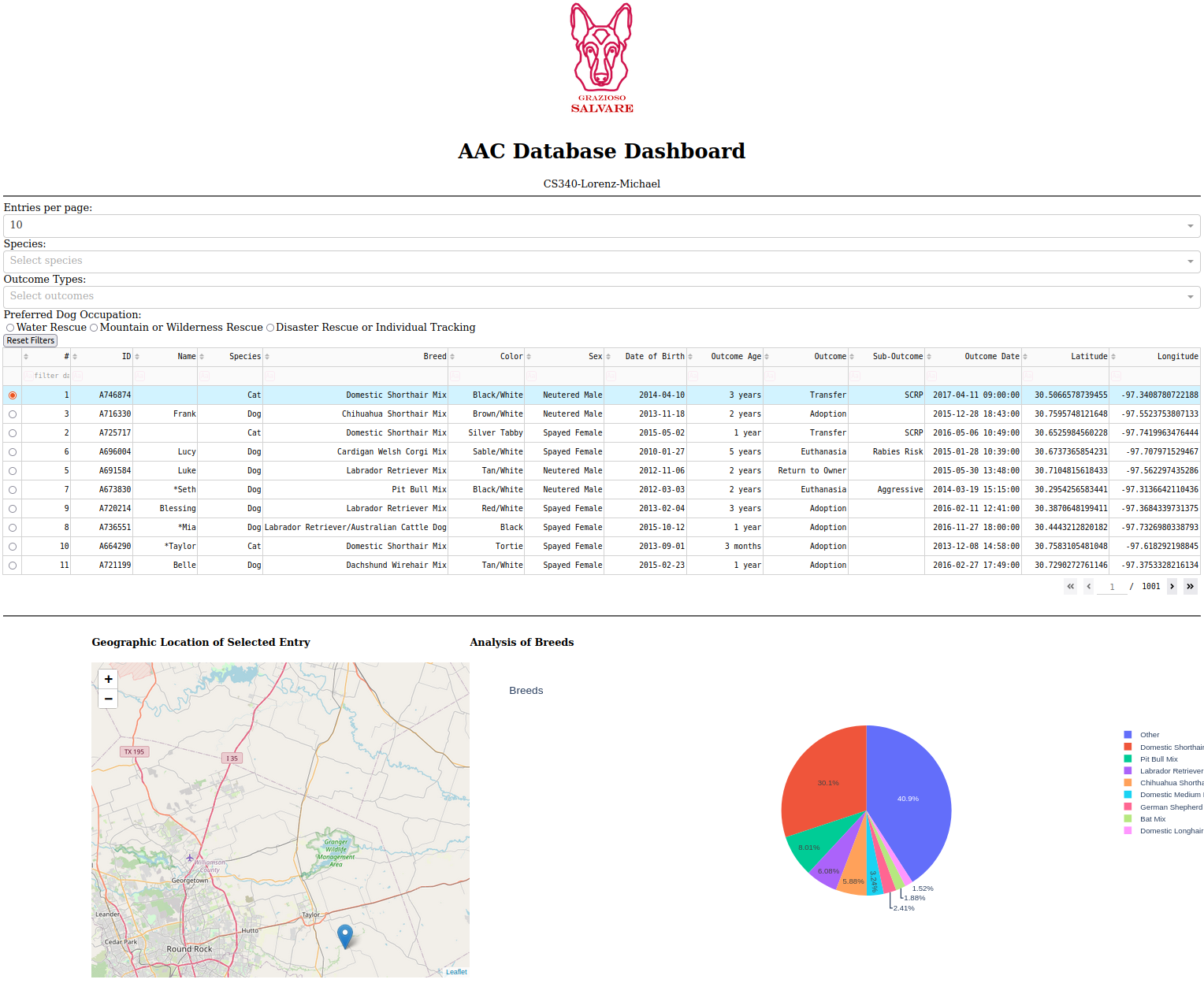
Clicking the “Mountain or Wilderness Rescue” filter shows dogs with breeds - German Shepherd, Alaskan Malamute, Old English Sheepdog; sex - intact male; age - 25 to 156 weeks. The map and chart are updated to show the selected row location and the proportions of breeds in this query.

### Filter animals by Disaster or Individual Tracking Rescue Type



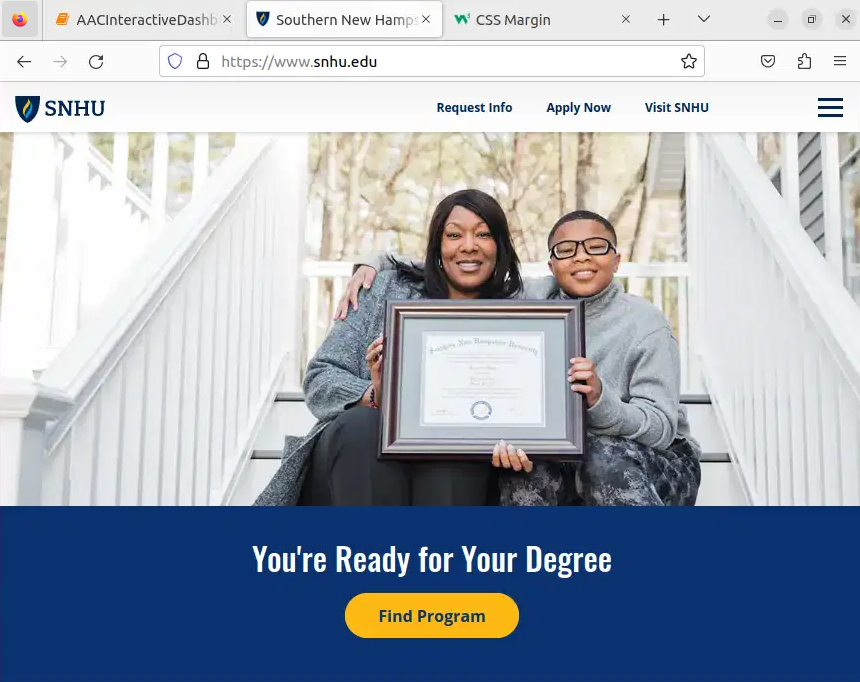
Clicking the “Disaster Rescue or Individual Tracking” filter shows dogs with breeds - Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, Rottweiler; sex - intact male; age - 20 to 300 weeks. The map and chart are updated to show the selected row location and the proportions of breeds in this query.

### Reset Filters to the Initial State



Clicking the “Reset Filters” button resets the value for each filter (Species, Outcome Types, Preferred Dog Occupation) to a blank value, which resets the data table and widgets to show the entire database.

### Grazioso Salvare Logo



Clicking the logo image links to the client's website, as required.

### Explanation of Widgets

| **Geographic Location Selection Map**    Selecting any row in the table updates the map to show the entry location. Hovering on the marker shows the breed, and clicking it shows the animal’s name. | **Breed Analysis Pie Chart**    Updating filters will update the pie chart to reflect the proportion of breeds in the table after filtering. Any breed that makes up less than 1.5% of the proportion is combined into the “other” category. |
| --- | --- |

## Contact

Michael Lorenz:

[michael.lorenz@snhu.edu](mailto:michael.lorenz@snhu.edu)

<https://github.com/Halfwitz>

[**> Return to top**](#oytswe2yhlqz)