# ACC Data Dashboard

# 

## About the Project

This project is a Python-based web application dashboard that provides Create, Read, Update, and Delete (CRUD) functionality for interacting with a MongoDB database. It is designed to enable seamless integration between a client-side user interface and a MongoDB backend specifically for managing the Austin Animal Center (AAC) database. The dashboard allows users to filter, view, and analyze data about dogs suitable for different types of rescue training.

## Motivation

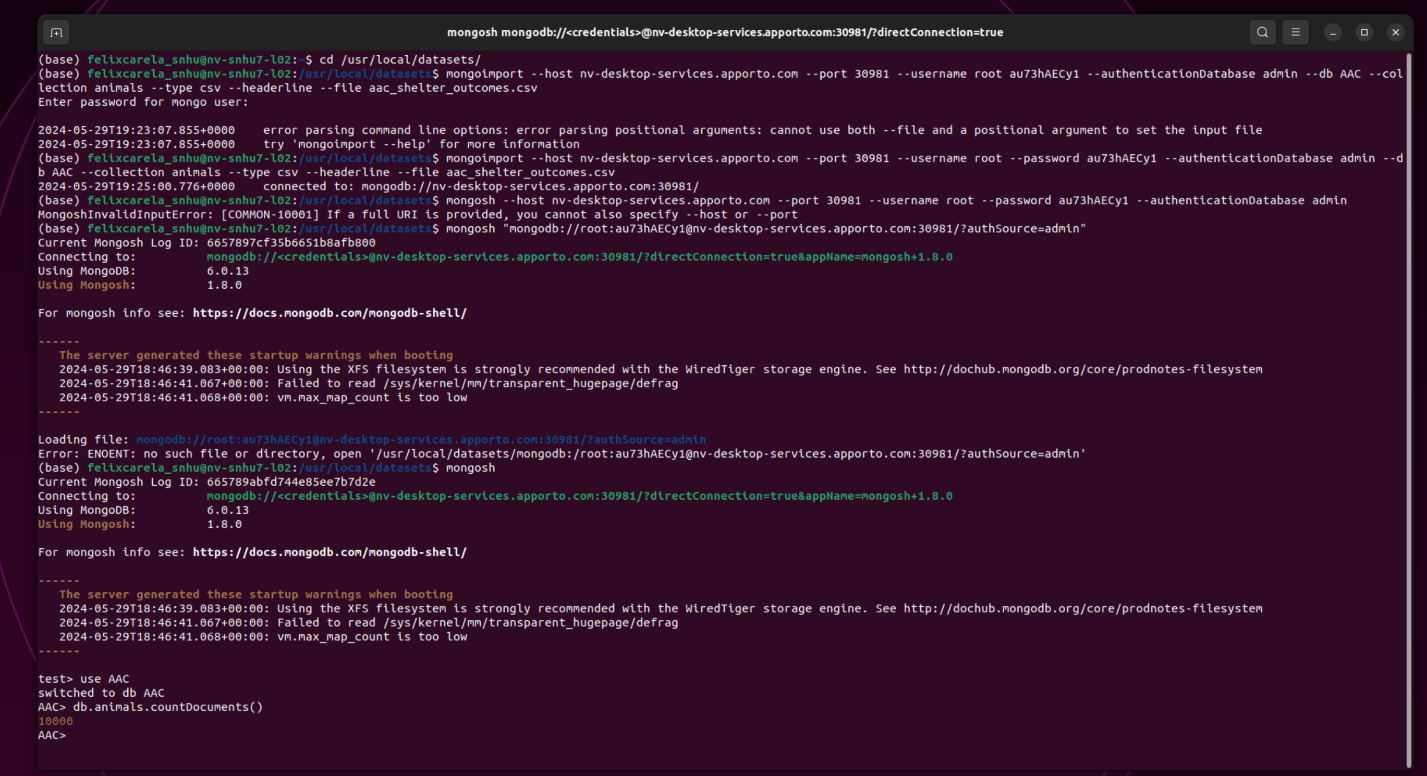
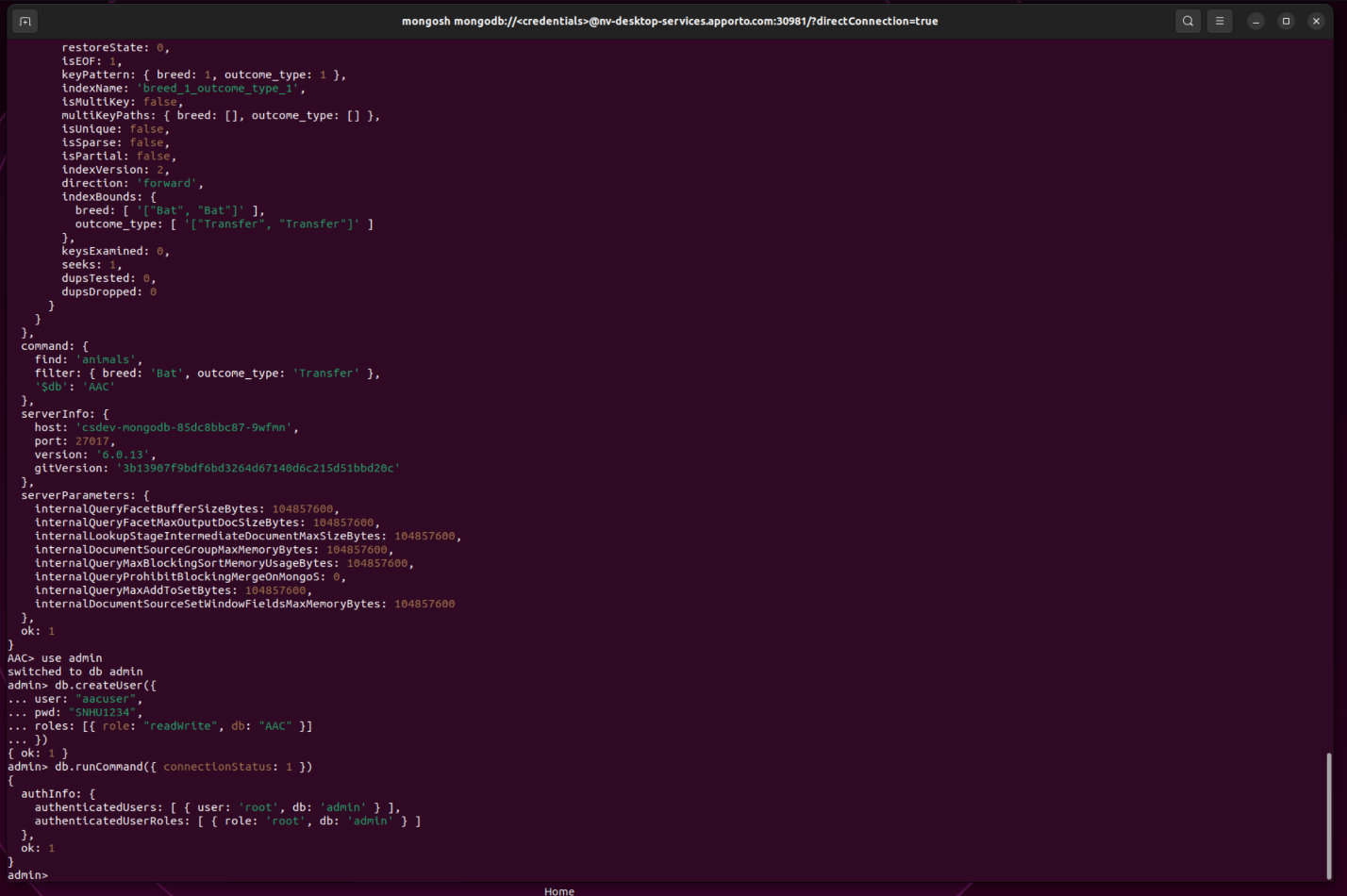
The motivation behind this project is to provide an intuitive and user-friendly interface for Grazioso Salvare to identify and categorize dogs suitable for various types of search-and-rescue training. The project was created as part of a course assignment to demonstrate the practical implementation of a full-stack application using Python, MongoDB, and Dash.

## Getting Started

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

1. **Prerequisites:**
   * Python 3.x
   * MongoDB installed and running
   * The PyMongo library for MongoDB interaction
   * Dash libraries for web application
2. **Installation:**
3. Clone the repository to your local machine: git clone <repository\_url>
4. Navigate to the project directory: cd project\_directory
5. Install the required libraries using pip: pip install pymongo dash jupyter-dash plotly dash-leaflet

### Setup MongoDB

1. **Import the Austin Animal Center Outcomes data set:**Open a terminal and navigate to the correct directory, then use the mongoimport tool:  
     
   mongoimport --host nv-desktop-services.apporto.com --port 30981 --username root --password au73hAECy1 --authenticationDatabase admin --db AAC --collection animals --type csv --headerline --file aac\_shelter\_outcomes.csv  
     
   
2. **Create a user account for authentication:**In the Mongo shell, create the aacuser account:  
     
   use AAC  
   db.createUser({  
   user: "aacuser",  
   pwd: "SNHU1234",  
   roles: [{ role: "readWrite", db: "AAC" }]  
   })  
   db.runCommand({ connectionStatus: 1 })  
     
   

## Usage

### Python Driver for MongoDB

This project uses the PyMongo library as the Python driver for MongoDB. PyMongo is the official MongoDB driver for Python. It was chosen for its simplicity, as it provides an intuitive API for CRUD operations. Additionally, PyMongo is well-documented, which makes it easier to find resources and support. It is optimized for performance, ensuring efficient data operations. Being the official driver, PyMongo has a large user base and community support, which is beneficial for troubleshooting and extending functionality.

### CRUD Operations

The AnimalShelter class provides four primary methods to interact with the MongoDB database: Create, Read, Update, and Delete (CRUD). Below is an explanation of each method, its attributes, and functionality.

* **create**: Inserts a document into a specified MongoDB database and collection. It takes data as an attribute, which is a dictionary representing the document to be inserted. The method checks if data is not None and uses the insert\_one function to insert the document. It returns True if the insertion is successful; otherwise, it raises an exception.
* **read**: Queries for documents from a specified MongoDB database and collection. It takes query as an attribute, which is a dictionary representing the key/value lookup pair for the find API call. The method checks if query is not None and uses the find function to retrieve documents. It returns a list of documents if the query is successful; otherwise, it raises an exception.
* **update**: Queries for and changes documents from a specified MongoDB database and collection. It takes query as an attribute, which is a dictionary representing the key/value lookup pair for the find API call, and update\_data, which is a dictionary representing the key/value pairs to update. The method checks if both query and update\_data are not None and uses the update\_many function to update the documents. It returns the number of documents modified if the update is successful; otherwise, it raises an exception.
* **delete**: Queries for and removes documents from a specified MongoDB database and collection. It takes query as an attribute, which is a dictionary representing the key/value lookup pair for the find API call. The method checks if the query parameter is not None and uses the delete\_many function to remove the documents. It returns the number of documents deleted if the deletion is successful; otherwise, it raises an exception.

### Code Example

from pymongo import MongoClient

class AnimalShelter(object):

""" CRUD operations for Animal collection in MongoDB """

def \_\_init\_\_(self, user, password, host, port, db, collection):

# Initialize the MongoClient

self.client = MongoClient(f'mongodb://{user}:{password}@{host}:{port}/?directConnection=true')

self.database = self.client[db]

self.collection = self.database[collection]

def create(self, data):

if data is not None:

self.collection.insert\_one(data) # data should be dictionary

return True

else:

raise Exception("Nothing to save, because data parameter is empty")

return False

def read(self, query):

if query is not None:

cursor = self.collection.find(query)

result = [document for document in cursor]

return result

else:

raise Exception("Query is empty")

return []

def update(self, query, update\_data):

if query is not None and update\_data is not None:

result = self.collection.update\_many(query, {'$set': update\_data})

return result.modified\_count

else:

raise Exception("Query or update\_data is empty")

return 0

def delete(self, query):

if query is not None:

result = self.collection.delete\_many(query)

return result.deleted\_count

else:

raise Exception("Query is empty")

return 0

### Tests

To test the CRUD functionality, run the following commands:

# Import the AnimalShelter class from the animal\_shelter module

from animal\_shelter import AnimalShelter

# Instantiate the AnimalShelter class

shelter = AnimalShelter('aacuser', 'SNHU1234', 'localhost', 27017, 'AAC', 'animals')

# Test the create method

data = {

"animal\_id": "A123457",

"name": "Max",

"breed": "Labrador Retriever",

"age": 3,

"color": "Black"

}

print("Inserting data: ", shelter.create(data))

# Test the read method

query = {"name": "Max"}

print("Query result: ", shelter.read(query))

# Test the update method

update\_query = {"name": "Max"}

update\_data = {"age": 4, "color": "Brown"}

print("Documents updated: ", shelter.update(update\_query, update\_data))

print("Query result after update: ", shelter.read(query))

# Test the delete method

delete\_query = {"name": "Max"}

print("Documents deleted: ", shelter.delete(delete\_query))

print("Query result after delete: ", shelter.read(query))

## Using the Code with the Dashboard

This code creates an interactive dashboard using Dash. The dashboard includes filtering options, an interactive data table, and visualizations.

# Setup the Jupyter version of Dash

from jupyter\_dash import JupyterDash

# Configure the necessary Python module imports for dashboard components

import dash\_leaflet as dl

from dash import dcc

from dash import html

import plotly.express as px

from dash import dash\_table

from dash.dependencies import Input, Output, State

import base64

import pandas as pd

import urllib.parse

# Import the CRUD module

from animal\_shelter import AnimalShelter

###########################

# Data Manipulation / Model

###########################

# Data variables

username = "aacuser"

password = "SNHU1234"

host = "nv-desktop-services.apporto.com"

port = 30981

db = "AAC"

collection = "animals"

# Connect to database via CRUD Module

shelter = AnimalShelter(username, password, host, port, db, collection)

# Define the initial query for "Disaster or Individual Tracking"

initial\_query = {"breed": {"$in": ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"]}}

# Read data from MongoDB based on the initial query

df = pd.DataFrame.from\_records(shelter.read(initial\_query))

# Drop the '\_id' column for compatibility with Dash DataTable

df.drop(columns=['\_id'], inplace=True)

#########################

# Dashboard Layout / View

#########################

app = JupyterDash(\_\_name\_\_)

# Add Grazioso Salvare’s logo

image\_filename = '/home/felixcarela\_snhu/Desktop/Grazioso Salvare Logo.png' # path to the image file in the virtual environment

encoded\_image = base64.b64encode(open(image\_filename, 'rb').read()).decode()

app.layout = html.Div([

    html.Div([

        html.Img(

            src=f'data:image/png;base64,{encoded\_image}',

            style={'width': '50px', 'height': '50px', 'vertical-align': 'middle', 'margin-right': '10px'}

        ),

        html.B(html.H1('CS-340 Dashboard - Felix Carela', style={'display': 'inline', 'vertical-align': 'middle'}))

    ], style={'text-align': 'center', 'display': 'inline-block', 'width': '100%'}),

    html.Hr(),

    # Add interactive filtering options

    html.Div([

        dcc.RadioItems(

            id='filter-type',

            options=[

                {'label': 'Water Rescue', 'value': 'Water'},

                {'label': 'Mountain or Wilderness Rescue', 'value': 'Mountain'},

                {'label': 'Disaster or Individual Tracking', 'value': 'Disaster'},

                {'label': 'Reset', 'value': 'Reset'}

            ],

            value='Disaster',  # Set the default value to 'Disaster'

            labelStyle={'display': 'inline-block'}

        )

    ]),

    html.Hr(),

    dash\_table.DataTable(

        id='datatable-id',

        columns=[{"name": i, "id": i, "deletable": False, "selectable": True} for i in df.columns],

        data=df.to\_dict('records'),

        editable=False,

        filter\_action="native",

        sort\_action="native",

        sort\_mode="multi",

        column\_selectable="single",

        row\_selectable="single",

        row\_deletable=False,

        selected\_rows=[0],

        page\_action="native",

        page\_current=0,

        page\_size=10,

    ),

    html.Br(),

    html.Hr(),

    # This sets up the dashboard so that the chart and geolocation chart are side-by-side

    html.Div(className='row', style={'display': 'flex'}, children=[

        html.Div(id='graph-id', className='col s12 m6'),

        html.Div(id='map-id', className='col s12 m6')

    ])

])

#############################################

# Interaction Between Components / Controller

#############################################

@app.callback(

    Output('datatable-id', 'data'),

    [Input('filter-type', 'value')]

)

def update\_dashboard(filter\_type):

    if filter\_type == 'Reset':

        # Retrieve all records from the MongoDB collection

        dff = pd.DataFrame.from\_records(shelter.read({}))

    else:

        query = {}

        if filter\_type == 'Water':

            query = {"breed": {"$in": ["Labrador Retriever Mix", "Chesapeake Bay Retriever", "Newfoundland"]}}

        elif filter\_type == 'Mountain':

            query = {"breed": {"$in": ["German Shepherd", "Alaskan Malamute", "Old English Sheepdog", "Siberian Husky", "Rottweiler"]}}

        elif filter\_type == 'Disaster':

            query = {"breed": {"$in": ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"]}}

        dff = pd.DataFrame.from\_records(shelter.read(query))

    # Drop the '\_id' column for compatibility with Dash DataTable

    dff.drop(columns=['\_id'], inplace=True)

    return dff.to\_dict('records')

# Display the breeds of animal based on quantity represented in the data table

@app.callback(

    Output('graph-id', "children"),

    [Input('datatable-id', "derived\_virtual\_data")]

)

def update\_graphs(viewData):

    dff = pd.DataFrame.from\_dict(viewData)

    # Count the occurrences of each breed

    breed\_counts = dff['breed'].value\_counts().reset\_index()

    breed\_counts.columns = ['breed', 'count']

    # Select the top 10 results

    top\_breeds = breed\_counts.nlargest(10, 'count')

    return [

        dcc.Graph(

            figure=px.pie(top\_breeds, names='breed', values='count', title='Top 10 Results')

        )

    ]

# This callback will highlight a cell on the data table when the user selects it

@app.callback(

    Output('datatable-id', 'style\_data\_conditional'),

    [Input('datatable-id', 'selected\_columns')]

)

def update\_styles(selected\_columns):

    return [{

        'if': {'column\_id': i},

        'background\_color': '#D2F3FF'

    } for i in selected\_columns]

# This callback will update the geo-location chart for the selected data entry

@app.callback(

    Output('map-id', "children"),

    [Input('datatable-id', "derived\_virtual\_data"),

     Input('datatable-id', "derived\_virtual\_selected\_rows")]

)

def update\_map(viewData, index):

    if viewData is None:

        return

    dff = pd.DataFrame.from\_dict(viewData)

    if index is None or len(index) == 0:

        row = 0

    else:

        row = index[0]

    return [

        dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.75, -97.48], zoom=10, children=[

            dl.TileLayer(id="base-layer-id"),

            dl.Marker(position=[dff.iloc[row, dff.columns.get\_loc('location\_lat')], dff.iloc[row, dff.columns.get\_loc('location\_long')]], children=[

                dl.Tooltip(dff.iloc[row, dff.columns.get\_loc('breed')]),

                dl.Popup([

                    html.H1("Animal Name"),

                    html.P(dff.iloc[row, dff.columns.get\_loc('name')])

                ])

            ])

        ])

    ]

app.run\_server(debug=True)

## Explanation of the Dash Framework

Dash is a productive Python framework for building web applications. It is suited for applications that require interactive data visualization. Dash abstracts away all the technologies and protocols required to build a full-fledged web application, and it provides the following features:

* **User Interface (UI)**: Dash uses React.js to render the components of the web application. It allows developers to create interactive and highly customizable UIs using simple Python code.
* **Routing**: Dash provides URL routing and multi-page apps, which helps in creating complex applications with multiple pages.
* **Interactivity**: Dash applications are built using Flask, which means they can easily interact with RESTful APIs and other web services. The framework provides decorators to create callback functions that make components in the application interactive.
* **Visualization**: Dash integrates seamlessly with Plotly, a graphing library, to provide rich data visualizations. These visualizations are interactive and can be customized extensively.

By using Dash, developers can focus on writing analytical applications without needing to worry about frontend development intricacies. The combination of Plotly for visualizations and React.js for UI components makes Dash a great tool for creating data-driven web applications.

### Steps Taken to Complete the Project

1. **Setup and Installation**: Installed necessary libraries and set up the MongoDB database.
2. **CRUD Module Implementation**: Developed the AnimalShelter class to perform CRUD operations.
3. **Dashboard Development**: Created the dashboard using Dash, adding interactive components and visualizations.
4. **Testing and Deployment**: Tested the application functionality and deployed the dashboard.

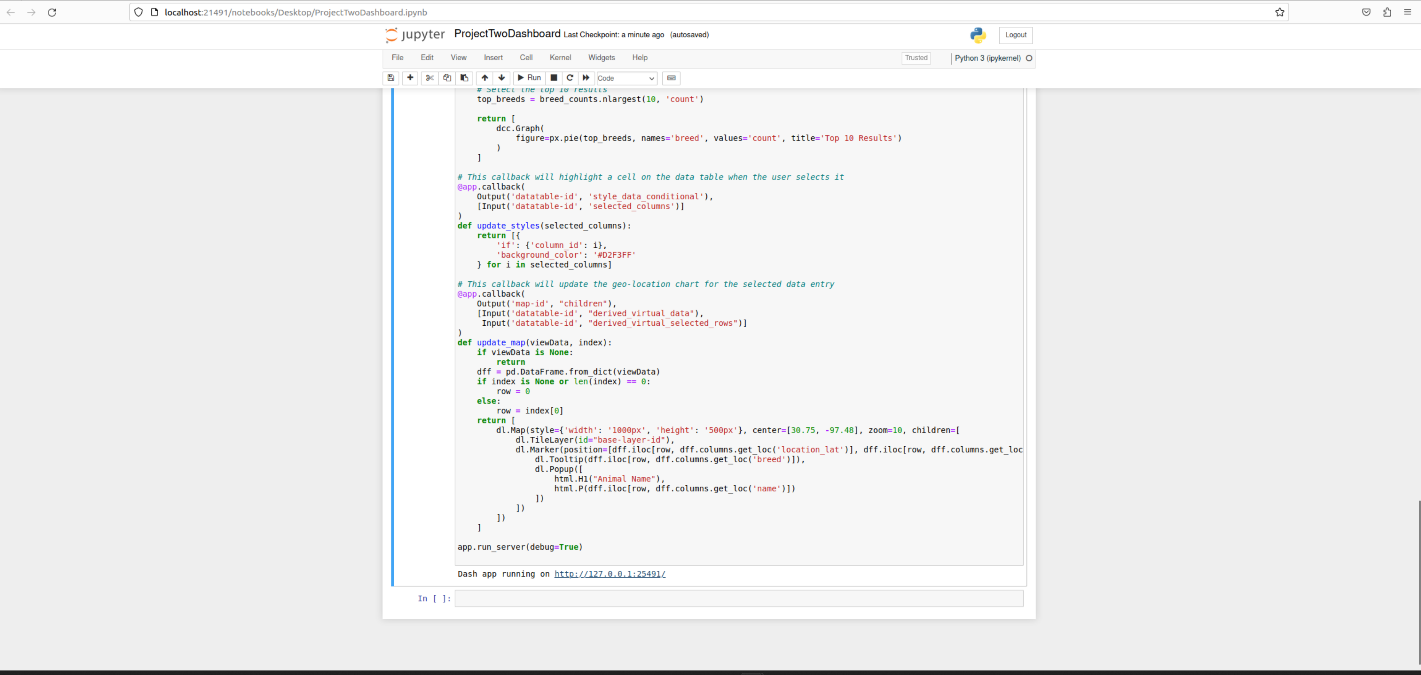
### Challenges and Solutions

* **Data Handling**: Handling large datasets in the dashboard required optimizing queries and managing data efficiently.
  + **Solution**: Used efficient querying and data manipulation techniques to handle large datasets.
* **Interactivity**: Ensuring smooth interactivity and responsiveness in the dashboard.
  + **Solution**: Utilized Dash's callback functions to manage interactions between components effectively.

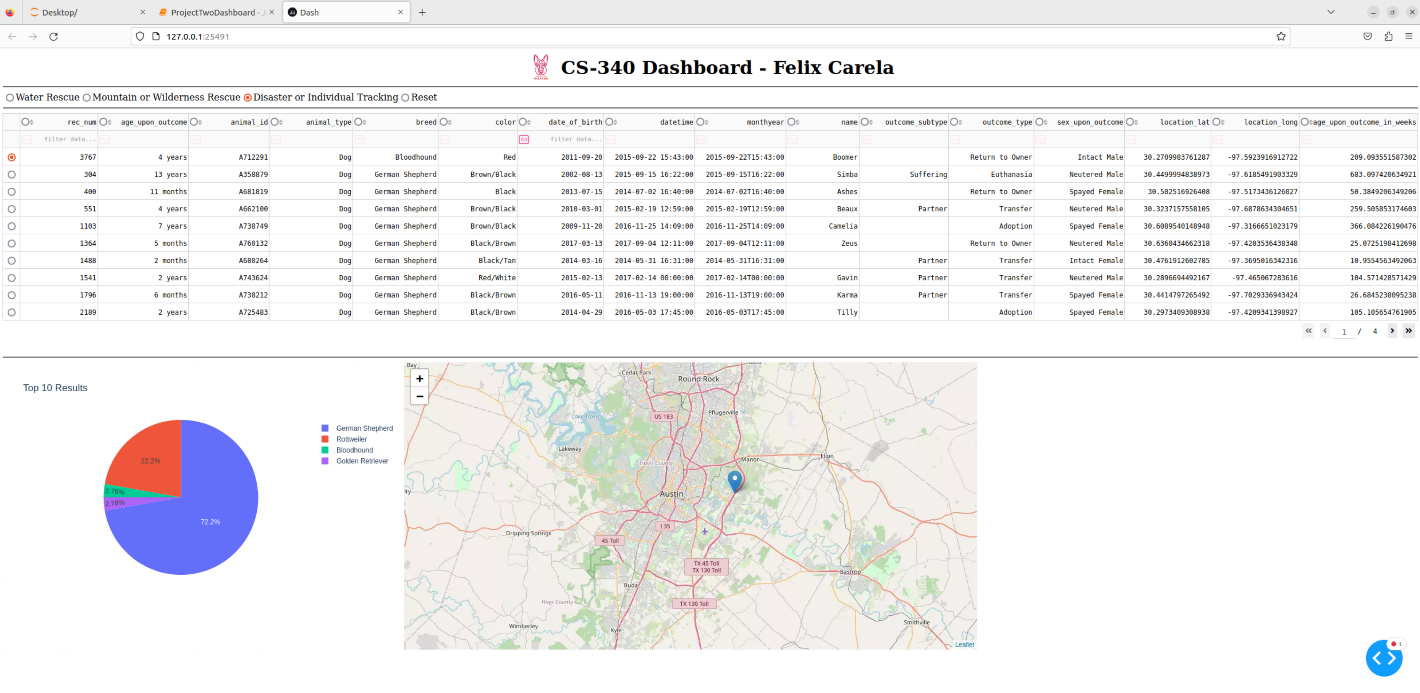
### Screenshots

Below it shows how it will look when running the code.

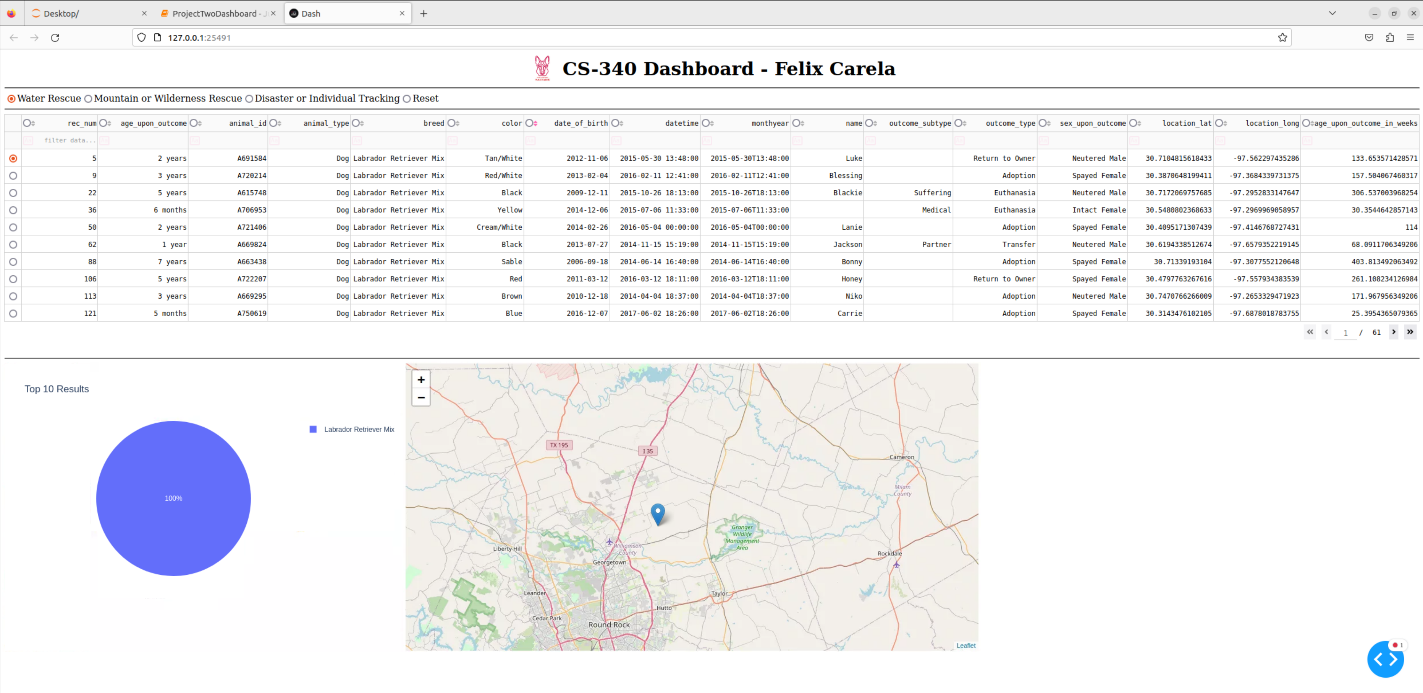
1. Running the code:



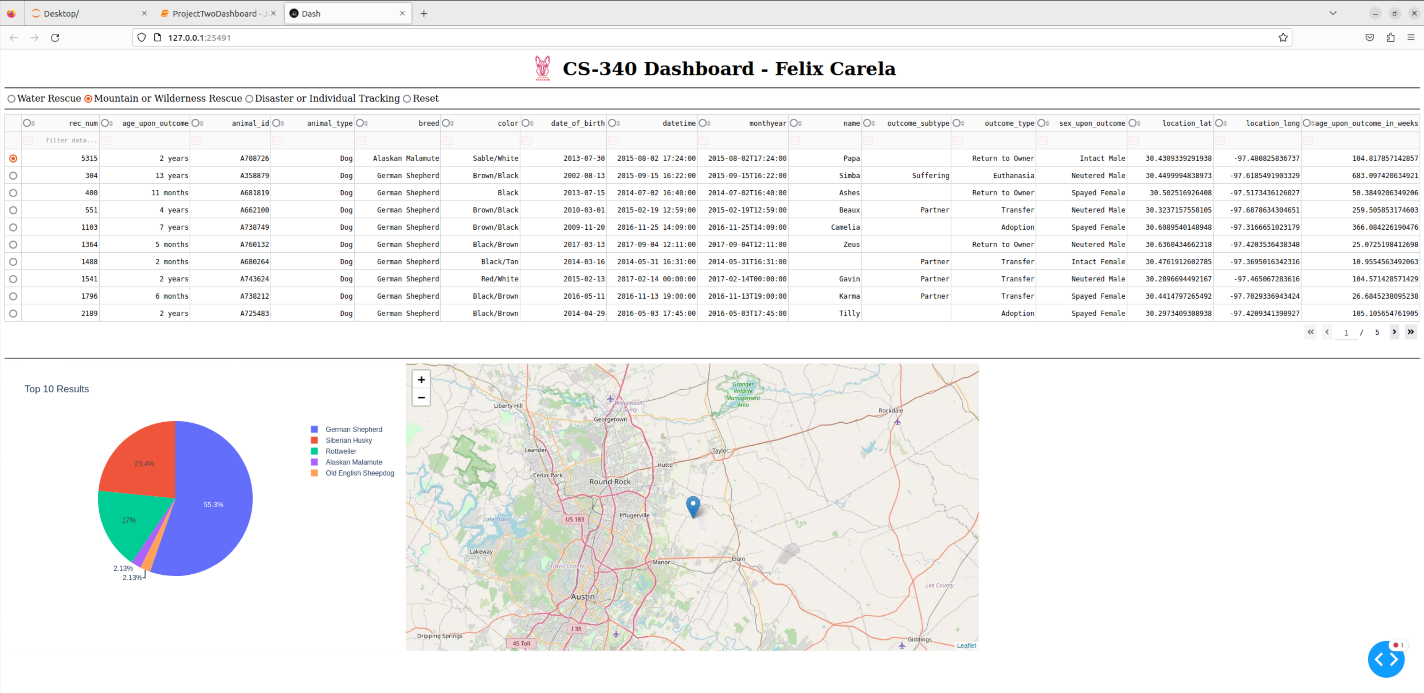
1. Starting state and disaster or individual tracking filter:

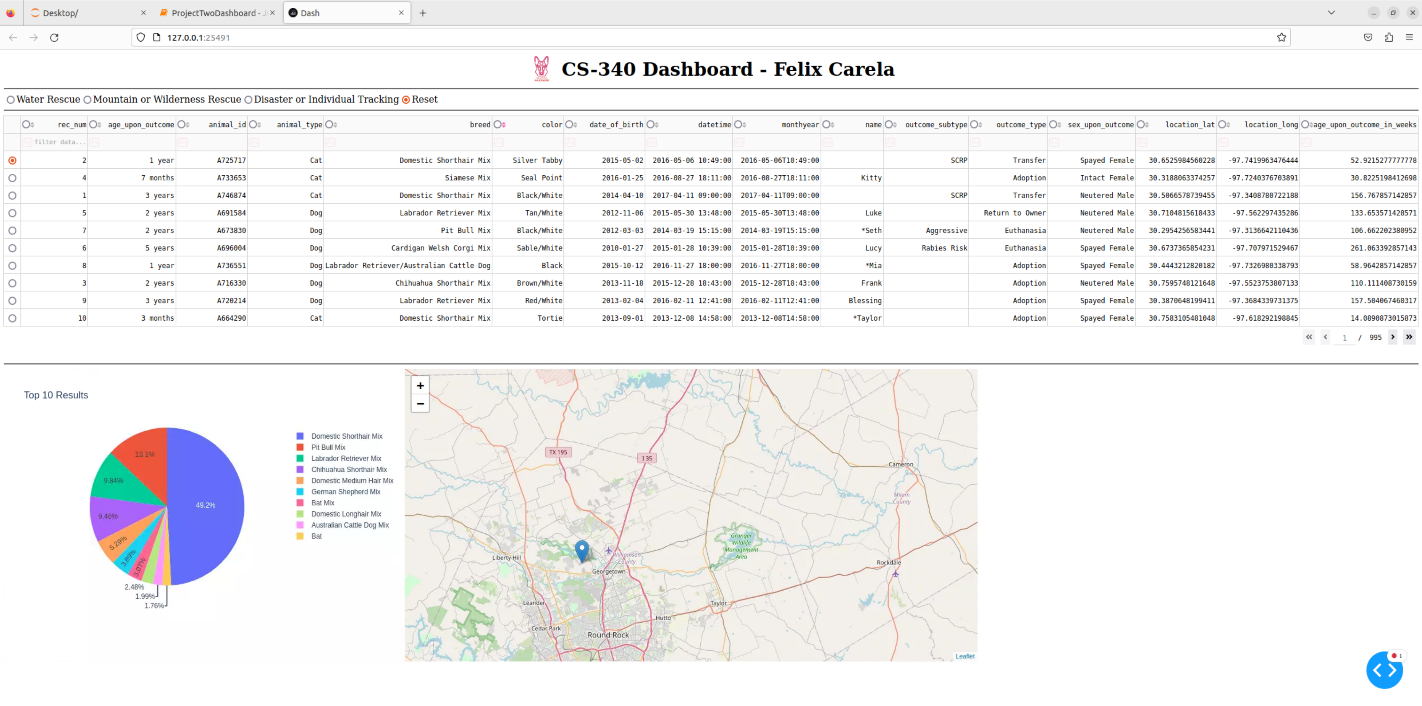


1. Water rescue filter:



1. Mountain or wilderness rescue filter:



1. Reset state:  
   

## Contact

Felix Carela

felixacarela@gmail.com