Samantha Durr
October 2, 2024
CS-340
Project 1

Animal Shelter Database

Grazioso Salvare has asked for a database of the animals that they train and use for different types of rescue. As a part of that, they wish for the ability to filter their search for specific animals, types of rescues, and other statuses through the use of an online dashboard that displays their database. To that end, they wish to have geolocation and other charts for being able to interact with the data.

Motivation

The main motivation for this project is to provide Grazioso Salvare with a base framework. This requires using certain components. There are two main components to this program. These components are are MongoDB and the Dash Framework.

MongoDB is the first of these two and is the base backbone for the entire system. This is because it is the database that holds all of the documents on the different animals within the system. MongoDB is a database that is easy to use and can be used across many systems. Another part of this is that it is easy to create code files that make using the database with other programs, such as the Dash Framework, easy and can use the Python language to accomplish the coding for this system.

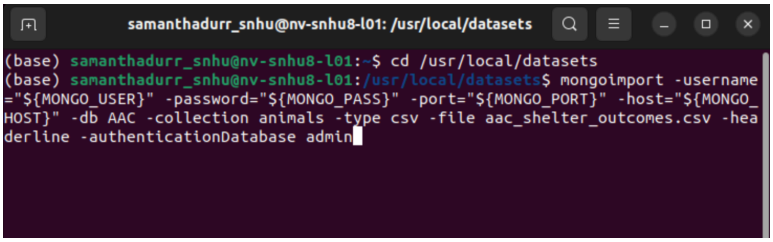
In terms of the Dash Framework, this has the advantage to be dynamic. This is the main controller for interacting with the database and is a web application. The use of this framework allows users to use one application to do all of the searching that they need. This is important for users as more people are used to a graphical interface over the terminal base you see in MongoDB. Additionally, this allows those who are working in the field or with remote offices to access the data without having to be connected to the actual network. With the use of other programs, such as this project's use of Jupyter Notebook, it can use Python code as well. This allows for more a consistent programming approach and less conflicts through that method.

Getting Started

Samantha Durr
October 2, 2024
CS-340
Project 1

Before getting started, you need to make sure that you have MongoDB installed. If you're using a system like the one that can be found through SNHU's Apporto, this should already be installed in the appropriate virtual machine, otherwise, please go to MongoDB's installation guide site and follow the instructions there for your device. The other components that you will need to have are the aac_shelter_outcomes file, the Dash Framework, and a code editor for Python. Please refer to the Installation section for more information.

- After you have MongoDB, the first thing you will need to do is create a database using the data from the aac_shelter_outcomes csv file. Make this file has been supplied to you.
- Once you have the file, change your directory to where it is located.
    - Use: CD (filepath)
        - In the example, the command with the filepath looks like this: **cd /usr/local/datasets**.
- Import and create the database. Use the command below:
        - Command: mongoimport -username="${MONGO_USER}" -password="${MONGO_PASS}" -port="${MONGO_PORT}" -host="${MONGO_HOST}" -db AAC -collection animals -type csv -file aac_shelter_outcomes.csv -headerline -authenticationDatabase admin.
- Below example is a screenshot to show the commands.



- The red boxed example shows the command once an import has successfully completed.

Samantha Durr
October 2, 2024
CS-340
Project 1

```
(base) samanthadurr_snhu@nv-snhu8-l01:/usr/local/datasets$ mongoimport --username="${MONGO_USER}" --password="${MONGO_PASS}" --port=${MONGO_PORT} --host=${MONGO_HOST} --db AAC --collection animals --type csv --file aac_shelter_outcomes.csv --headerline --authenticationDatabase admin
2024-09-17T21:37:31.815+0000    connected to: mongodb://nv-desktop-services.apporto.com:32165/
2024-09-17T21:37:32.036+0000    10000 document(s) imported successfully. 0 document(s) failed to import.
```

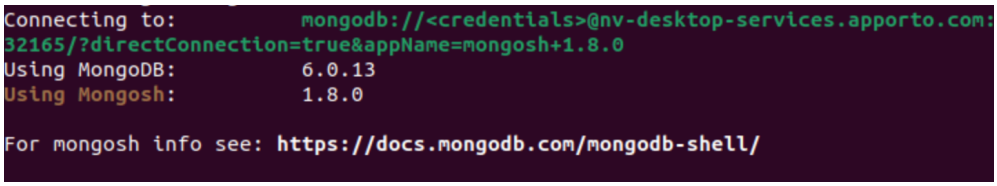
- After importing the documents, you will need to create a user that has access to this database.

  The user is aacuser and you will select the password. The following steps are below:

  - Access MongoDB with the command **mongosh**.
  - Navigate to the administrator database using the command **use admin**.
  - Create user with the following command:
    - **db.createUser({user: “aacuser”, pwd: “yourPassword”, roles: [{“readWrite”, db: “AAC}]})**
      - Replace the “yourPassword” with what you want the password to be and make sure to keep it on you. You will need this.
  - The below screenshot shows the switch to the admin database and the user creation.

```
mongosh mongodb://<credentials>@nv-desktop-services.ap...
Current Mongosh Log ID: 66eb359db3c796d1ccd1f813
Connecting to:          mongodb://<credentials>@nv-desktop-services.apporto.com:32165/?directConnection=true&appName=mongosh+1.8.0
Using MongoDB:          6.0.13
Using Mongosh:          1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-09-18T20:13:17.689+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
   2024-09-18T20:13:19.732+00:00: Failed to read /sys/kernel/mm/transparent_hugepage/defrag
   2024-09-18T20:13:19.732+00:00: vm.max_map_count is too low
------

test> use admin
switched to db admin
admin> db.createUser({user: "aacuser", pwd: "X5q69KmT3", roles: [{role: "readWrite", db: "AAC"}]})
{ ok: 1 }
admin>
```

Samantha Durr
October 2, 2024
CS-340
Project 1

- Once you have created your user, you need to access your CRUD in the CRUD.py file. You need to replace the port number in the line: self.client = *MongoClient(f'mongodb://{username}:{password}@nv-desktop-services.apporto.com:32165/?directConnection=true&appName=mongosh+1.8.0/AAC')* with your own port number.
    - Note: the highlighted red part is the port number you need to change. This can be found by looking at the line, connecting to and the number after the apporto.com.
    - 

Line to look for the port number from.
- Next you need to update your username and password in the SamanthaDurrProject2Dashboard.ipynb file to your information. This can be found in the Data Model area and these are the lines:

```
username = "aacuser"
password = "X5q69KmT3"
```

    - Note: if you are using aacuser, then you'll only need to change the password to your own password.
    - If you are running your own CRUD file, make sure it is in the same place as your Jypiter Notebook file, also, you will need to update the following line:
        - from CRUD import Module4CRUD, DatabaseError
            - Update the red sections with the name of your file and the name of your class.

- Once you have completed those changes, you are fine to move forward with your program as the rest is already handled by the ProjectTwoDashboard file. If you wish to create your own file, that can be found in the usage section.

Installation

The tools you are going to need are a terminal, Jupyter Notebook, a Python code editor (example is Pycharm), MongoDB, and the Dash Framework. Additional libraries will be needed. The library is the pymongo library and the MongoClient specifically.

- The terminal, or command line window, is found already built into your operating system.
- Jupyter Notebook: this is a tool that can be used for creating notebook-based python scripts. To install this, you can do so through the website: https://jupyter.org/install.
- Python Code Editor: Pycharm is the most common and can be downloaded to install here: https://www.jetbrains.com/pycharm/download/other.html.
- MongoDB: This comes in multiple versions, such as community or enterprise. If you want the detailed installation instructions go to: https://docs.mongodb/com/manual/installation/.
- Dash Framework: The dash framework is used in the form of plotly to work with MongoDB, Jupyter Notebook, and Python. The instructions can be found here: https://dash.plotly.com/installation.
- Pymongo and MongoClient: added into the nodule using the from pymongo import MongoClient line at the top of the code.
- CRUD.py: this is the module that was created to run the CRUD for the database. You will need this file to run as is. If you are using your own CRUD file, replace mine with yours.
- ProjectTwoDashboard.ipynb: this file will run the web application to make sure that the CRUD module works. If you have created your own test file, then replace mine with that one.

Samantha Durr
October 2, 2024
CS-340
Project 1

- Grazioso Salvare Logo.png: This is the image for the Grazioso Salvare company that has requested this database being made.

Usage

With the use of the ProjectTwoDashboard file, you do not have as much that you need to create. The main lines you will need to change are based on the data that you have created. The red boxed lines are the ones you need to make the changes to. This is only applicable if your information is not the exact same as what I have used:

```
#### FIX ME #####
# change animal_shelter and AnimalShelter to match your CRUD Python module file name and class name
from CRUD import Module4CRUD, DatabaseError #My CRUD file and the classes that contain the code.

###########################
# Data Manipulation / Model
###########################
# FIX ME update with your username and password and CRUD Python module name

username = "aacuser"
password = "X5q69KmT3" #Password to the version of the code that I use, to be replaced if someone else's user is
                       #different.

# Connect to database via CRUD Module
db = Module4CRUD(username, password)
```

If you are creating your own CRUD file, you will need to ensure that your file, besides containing the Create, Read, Update, and Delete code, has this line at the top of it:

```
from pymongo import MongoClient
```

This allows for creating a MongoDB session with the code. That will save you time and ensure that you are able to log in.

The code that the CRUD module uses falls under 4 methods: create, read, update, and delete. Each of these functions perform certain actions. Below are the explanations and the code examples.

- create(self, data): this method allows for creating and inserting new documents into the AAC's animals collection database. The information passed in is sent to the database to create different documents of these animals.

Code Example

Samantha Durr
October 2, 2024
CS-340
Project 1

```
#Creates the document with the information inserted for the data.

def create(self, data):

    try:

        if data is not None:

            result = self.database.animals.insert_one(data)

            return result.inserted_id is not None

        else:

            raise ValueError("The parameter is empty and there is nothing to save.")

    except Exception as e:

        raise DatabaseError(f"Error in creating the document: {e}")
```

- read(self, criteria = None): this method allows for searching within the database for documents that match certain criteria. As a later example shows in the test, this can be as simple as looking for the type of animal that the user wishes to find.

  Code Example

```
def read(self, criteria=None):
    try:
        if criteria:
            data = self.database.animals.find(criteria, {"_id": False})
        else:
            data = self.database.animals.find({}, {"_id": False})
        return data
    except Exception as e:
        raise DatabaseError(f"Error reading data: {e}")
```

- update(self, criteria, update_data): this is used to change a document's information. If the age, type, or name changes for one of the animals already in the database, this allows for that update to take place.

Samantha Durr
October 2, 2024
CS-340
Project 1

Code Example

```
def update(self, criteria, update_data):
    try:
        result = self.database.animals.update_many(criteria, {"$set": update_data})
        return result.modified_count > 0
    except Exception as e:
        raise DatabaseError(f"There was an error updating the documents:  {e}")
```

- delete(self, criteria): this method is used for removing documents from the database. When the criteria fed into the method matches, it will removed the document from the database and this updates that database to not contain the information any longer.

Code Example

```
def delete(self, criteria):
    try:
        result = self.database.animals.delete_many(criteria)
        return result.deleted_count > 0
    except Exception as e:
        raise DatabaseError(f"There was an error in deleting the documents: {e}")
```

In terms of what you need for the Dashboard, there are several areas that you need to fill in. To begin with, you will need your Jupyter Notebook program to run this. The code that you will need comes in specific sections. You first need to begin with all of the libraries you are going to need, followed by the data necessary to access your CRUD file and log into the database. The example below shows the necessary code.

Code Example

```
# 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
```

Samantha Durr
October 2, 2024
CS-340
Project 1

```
from dash import dash_table
from dash.dependencies import Input, Output, State
import base64

# Configure OS routines
import os

# Configure the plotting routines
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from CRUD import Module4CRUD, DatabaseError #My CRUD file and the classes that contain the code.

#############################
# Data Manipulation / Model
#############################

username = "aacuser"
password = "X5q69KmT3" #Password to the version of the code that I use, to be replaced if someone else's user is
                    #different.

# Connect to database via CRUD Module
db = Module4CRUD(username, password)

# class read method must support return of list object and accept projection json input
# sending the read method an empty document requests all documents be returned
df = pd.DataFrame.from_records(db.read({})).drop('rec_num', axis = 1) #Using this allows for removing the drop
                                                    #column code.
#Sets the different types of rescue that can be searched for.
rescueTypes = ['Water Rescue', 'Mountain or Wilderness Rescue', 'Disaster or Individual Tracking', 'Reset']
```

Once you have this code, you can then go into the layout. This layout will ensure you have the image in place, your title, and the general look of the dashboard. The code for this is below. You can make your own changes to certain areas, such as where you place the image, or of the database being displayed. I have chosen to only display 20 documents on one page for ease of loading. You can remove this or even

shrink or increase that number. Any particular components you want for the appearance of your web

application are placed here.

Code Example

```
app = JupyterDash(__name__)

#FIX ME Add in Grazioso Salvare's logo
image_filename = 'GraziosoSalvareLogo.png' # The image requested by the company.
encoded_image = base64.b64encode(open(image_filename, 'rb').read())

#FIX ME Place the HTML image tag in the line below into the app.layout code according to your design
#FIX ME Also remember to include a unique identifier such as your name or date
#html.Img(src='{}'.format(encoded_image.decode()))

app.layout = html.Div([
   html.Center(html.Img(src = ' {}'.format(encoded_image.decode()), style =
{'width': '150px'},),),
   html.Center(html.B(html.H1('Samantha Durr, CS-340 Dashboard'))),
   html.Hr(),
   html.Div(
      dcc.RadioItems(id = 'rescue-type', options=[{'label': i, 'value': i} for i in rescueTypes],
                  value = 'Reset',
                  labelStyle = {'display': 'inline-block'}), #Sets the radio items in a line.

   ),
   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'),
                     #All of the following responses to the fix me are from my milestone for week 6.
                     row_selectable="single",
                     selected_rows=[0],
                     page_action='native',
                     page_current=0,
                     page_size=20,
                     style_table={'height': '300px', 'overflowY': 'auto'}
                    ),
   html.Br(),
   html.Hr(),
#This sets up the dashboard so that your chart and your geolocation chart are side-by-side
   html.Div(className='row',
       style={'display' : 'flex'},
          children=[
      html.Div(
```

Samantha Durr
October 2, 2024
CS-340
Project 1

```
        id='graph-id',
        className='col s12 m6',

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

The main sections of code that you will need to complete are methods for displaying the graph, the map, the map's style, and the dashboard. These methods update the system to show the changes you make as you interact with the dashboard. These methods should be labeled: update_dashboard, update_graphs, update_styles, and update_map. The following code example shows how this looks in the version used by the original creator.

Code Example

```
@app.callback(Output('datatable-id','data'),
          [Input('rescue-type', 'value')])
def update_dashboard(filter_type):
    #Checks the different radio buttons and the database for specific values being searched for in terms of
    #the different types of rescues. Displays only the those that match the criteria of the appropriate
    #if statement.
    if filter_type == 'Reset':
        dff = df
    elif filter_type == 'Water Rescue':
        dff = df[df.breed.isin(['Labrador Retriever Mix', 'Chesapeake Bay Retriever', 'Newfoundland'])
                & (df.sex_upon_outcome == 'Intact Female')
                & ((df.age_upon_outcome_in_weeks >= 26) & (df.age_upon_outcome_in_weeks <= 156))]
    elif filter_type == 'Mountain or Wilderness Rescue':
        dff = df[df.breed.isin(['German Shepherd', 'Alaskan Malamute', 'Old English Sheepdog',
                            'Siberian Husky', 'Rottweiler'])
                & (df.sex_upon_outcome == 'Intact Male')
                & ((df.age_upon_outcome_in_weeks >= 26) & (df.age_upon_outcome_in_weeks <= 156))]
    elif filter_type == 'Disaster or Individual Tracking':
        dff = df[df.breed.isin(['Doberman Pinscher', 'German Shepherd', 'Golden Retriever',
                            'Bloodhound', 'Rottweiler'])
                & (df.sex_upon_outcome == 'Intact Male')
                & ((df.age_upon_outcome_in_weeks >= 20) & (df.age_upon_outcome_in_weeks <= 300))]
```

Samantha Durr
October 2, 2024
CS-340
Project 1

```
    return dff.to_dict('records')

@app.callback(
    Output('graph-id', "children"),
    [Input('datatable-id', "derived_virtual_data")])
def update_graphs(viewData):
    #Creates a pie chart that displays all of the different breeds. When a specific search is called for,
    #it updates the graph to show the appropriate breeds and what portion they make up of that group.
    if not viewData:
        return dcc.Graph(figure = px.pie())
    else:
        dff = pd.DataFrame.from_dict(viewData)
        fig = px.pie(dff, names = 'breed', title = 'Breed Distribution')
        return dcc.Graph(figure = fig)

@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
# derived_virtual_data will be the set of data available from the datatable in the form of
# a dictionary.
# derived_virtual_selected_rows will be the selected row(s) in the table in the form of
# a list. For this application, we are only permitting single row selection so there is only
# one value in the list.
# The iloc method allows for a row, column notation to pull data from the datatable
@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
    elif index is None:
        return

    dff = pd.DataFrame.from_dict(viewData)
    # Because we only allow single row selection, the list can be converted to a row index here
    if index is None:
```

Samantha Durr
October 2, 2024
CS-340
Project 1

```
        row = 0
    else:
        row = index[0]

    # Austin TX is at [30.75,-97.48]
    return [
        dl.Map(style={'width': '1000px', 'height': '500px'}, center=[30.75,-97.48], zoom=10, children=[
            dl.TileLayer(id="base-layer-id"),
            # Marker with tool tip and popup
            # Column 13 and 14 define the grid-coordinates for the map
            # Column 4 defines the breed for the animal
            # Column 9 defines the name of the animal
            dl.Marker(position=[dff.iloc[row,13],dff.iloc[row,14]], children=[
                dl.Tooltip(dff.iloc[row,8]),#changed this to ensure that the appropriate name would appear.
                dl.Popup([
                    html.H1("Animal Name"),
                    html.P(dff.iloc[row,9])
                ])
            ])
        ])
    ]
```

The last component you will need is to run the dashboard. In this instance, you will see many versions that end with debug=True, but it is best to include the port as well. This helps to mitigate possible starting issues, but also causes the load to be smoother and faster. 8050 is the standard port.

Code Example

```
app.run_server(debug=True, port = 8050)
```

Tests

As long as the dashboard is running, your tests mainly consist of testing that the buttons and maps work as they should. In this regard, you can create smaller test files to ensure that everything is working. For these tests, the best option is to do smaller tests that show the dashboard is working. An example of these are found in the below screenshots for the different tests. If you are following the exact code provided, then you will not need to do much, but the screenshots below show the results.

Screenshots

Samantha Durr
October 2, 2024
CS-340
Project 1

If you are looking to see the code for testing that your CRUD code works, here are screenshots of the testing code. You can use this code to test your own CRUD, if you wrote it and did not use what was provided:

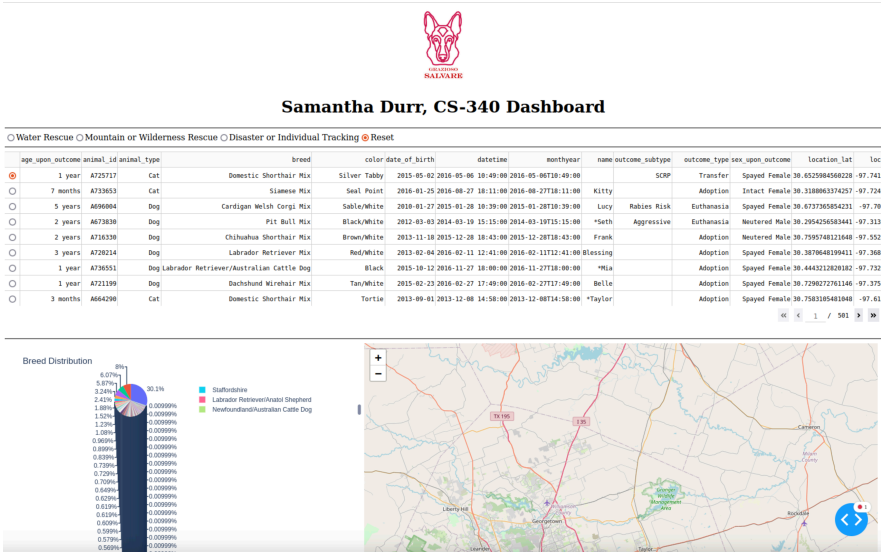
```python
from CRUD import Module4CRUD, DatabaseError

username = "aacuser"
password = "X5q69KmT3"

# Creates an instance of the Module4CRUD class.
animal = Module4CRUD(username, password)

# This is the sample data using one of my old fish and my cat.
animal_data = [
    {
        "name": "Jessie",
        "type": "cat"
    },
    {
        "name": "Kirishima",
        "type": "fish"
    }
]

# Adds the data to the database.
for i in animal_data:
    try:
        success = animal.create(i)
        if success:
            print(f"Created document: {i}")
        else:
            print(f"Failed to create the document: {i}")
    except DatabaseError as e:
        print(f"Database error: {e}")

# Read data
print("\nReading data:")
cats = animal.read({"type": "fish"})
for cat in cats:
    print(cat)
```

Your test results should look something like this:

```
Created document: {'name': 'Lucy', 'type': 'cat', '_id': ObjectId('66fc66df13ec91367e0f732c')}
Created document: {'name': 'Polaris', 'type': 'fish', '_id': ObjectId('66fc66df13ec91367e0f732d')}
Created document: {'name': 'Penny', 'type': 'dog', '_id': ObjectId('66fc66df13ec91367e0f732e')}

Reading data:
{'name': 'Kirishima', 'type': 'fish'}
{'name': 'Kirishima', 'type': 'fish'}
{'name': 'Kirishima', 'type': 'fish'}
{'name': 'Kirishima', 'type': 'fish'}
{'name': 'Orion', 'type': 'fish'}
{'name': 'Polaris', 'type': 'fish'}

Updating data:
Update was successful.

Deleting data:
Delete was successful.
```

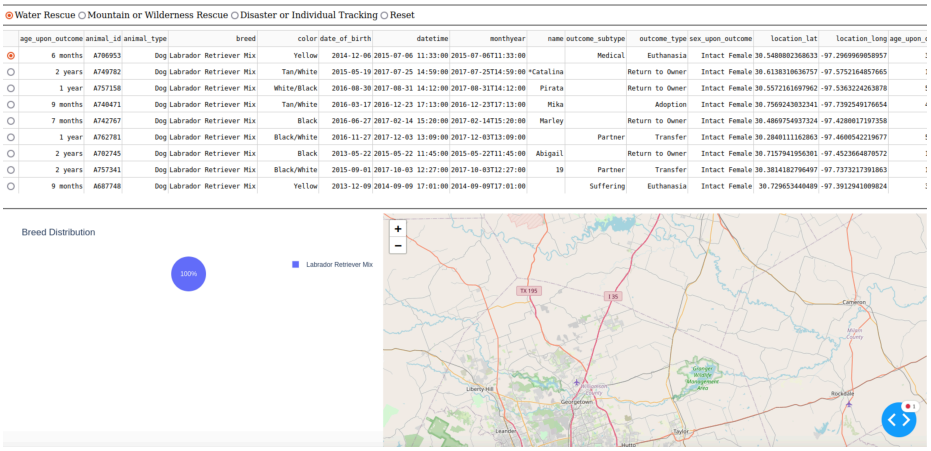
Samantha Durr
October 2, 2024
CS-340
Project 1

For the final run of the code with the dashboard, you should have this as your main page, before doing any filtering:



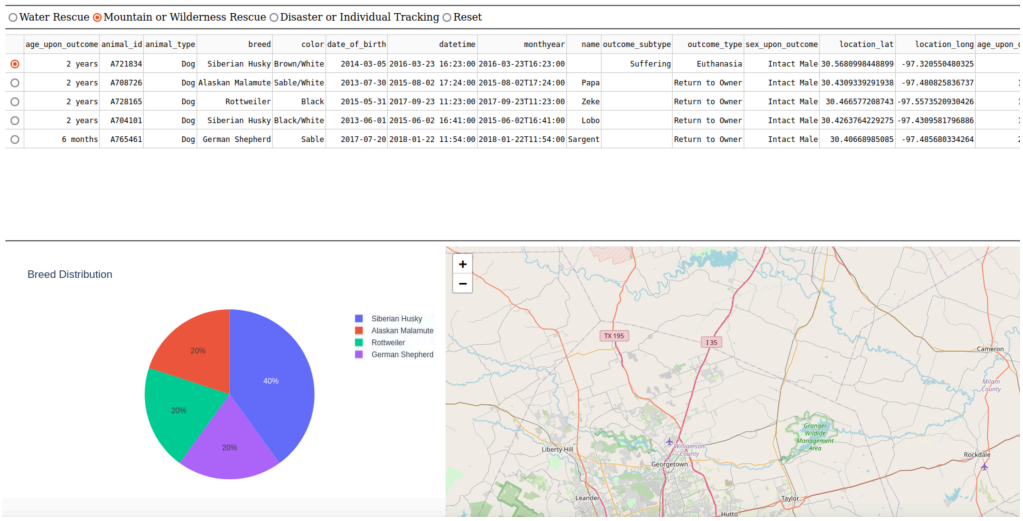
The following screenshots are what you see in the filtered category:

Water Rescue

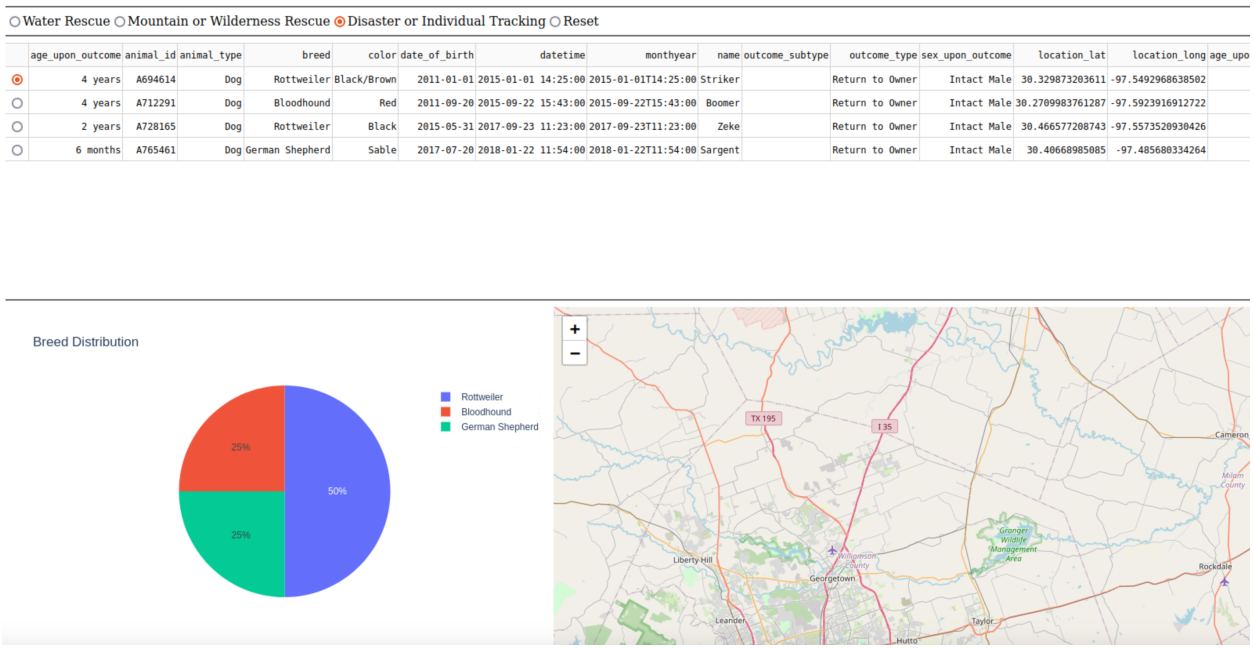


Mountain or Wilderness Rescue:

Samantha Durr
October 2, 2024
CS-340
Project 1



Disaster or Individual Tracking:



Errors

In terms of errors that occurred as I worked on this, the biggest was watching for the placement of your code. Some of the biggest issues came in the framework's appearance. If something was placed in the wrong spot, or an extra }]), any of them, occurred, the error would be difficult to locate. Also, make sure your connection is strong when running this, as it is easy for the system to suddenly disconnect. The

Samantha Durr
October 2, 2024
CS-340
Project 1
final error that caused issues was indentation. Python is a very indent focused language, so always make

sure that your code has the right indents and pay close attention to that.

Contact

Your name: Samantha Durr