**Grazioso Salvare Dashboard README**

**Project Overview**

The Grazioso Salvare Dashboard is an interactive web application developed to assist Grazioso Salvare, an international rescue-animal training company, in identifying dogs that are suitable for search-and-rescue training. The application connects to a MongoDB database containing animal shelter records and provides dynamic visualizations and filtering options to analyze the data.

This dashboard enables users to filter records based on specific rescue types, visualize preferred dog breeds, and explore the geographical locations of animals. The application is built using the Dash framework and runs within a Jupyter Notebook environment.

**Key Components**

***1. Data Manipulation / Model***

* **Database Connection**: The dashboard connects to a MongoDB database using credentials stored in the AnimalShelter class from a custom CRUD Python module. This module handles all interactions with the database, including reading, creating, updating, and deleting records.
* **Data Retrieval**: The dashboard reads data from the database and processes it for display. If the connection to the database fails, the dashboard gracefully handles the error and displays an appropriate message.

***2. Dashboard Layout / View***

* **Layout:** The dashboard's layout is designed using Dash components, including headers, filtering options, a data table, graphs, and a map. The layout is organized to provide a seamless user experience, with filtering options prominently displayed and visualizations arranged for easy comparison.
* **Interactive Widgets:**
  + **RadioItems**: Used for filtering data by Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking, or resetting the filter.
  + **DataTable:** Displays animal shelter records based on the selected filter, allowing users to select rows to view more details on a map.
  + **Graphs**: Includes a pie chart for visualizing preferred dog breeds and a bar chart for showing outcome types.
  + **Map:** Displays the geographical location of selected animals.

***3. Interaction Between Components / Controller***

* **Callbacks**: The dashboard uses Dash callback functions to manage interactions between components. For example, when a user selects a rescue type, the dashboard queries the database, updates the DataTable, and refreshes the graphs and map based on the new data.
* **Data Updates**: The dashboard dynamically updates all components in response to user inputs, ensuring that the displayed data is always relevant to the selected filters.

**Required Functionality:**

The dashboard meets the following requirements:

* **Interactive Filtering Options:** Users can filter data by rescue type (Water Rescue, Mountain or Wilderness Rescue, Disaster or Individual Tracking) or reset to view all records.
* **Data Table**: The table displays animal records dynamically based on the selected filter, allowing for sorting and selection of specific rows.
* **Geolocation Chart:** Displays the location of selected animals on a map, with interactive markers that provide details such as the animal's name and breed.
* **Second Chart**: A pie chart visualizes the preferred breeds of dogs according to the selected rescue type.
* **Bar Chart**: A bar chart provides a detailed view of outcome types for the selected filter.

**Usage:**

* **Environment**: This dashboard is designed to be run within a Jupyter Notebook environment using JupyterDash.
* **Execution:** Simply run the notebook cells to start the dashboard server. The application will be accessible within the notebook, where you can interact with the filters, DataTable, and visualizations.

**Screenshots:**

This is the starting state of my dashboard, or the unfiltered data with all widgets. The widgets for the interactive options to filter data (such as radio items or drop-downs), are below the interactive data table and shown in the “reset” screenshots in more detail (the reset and the starting state of the widgets are the same):

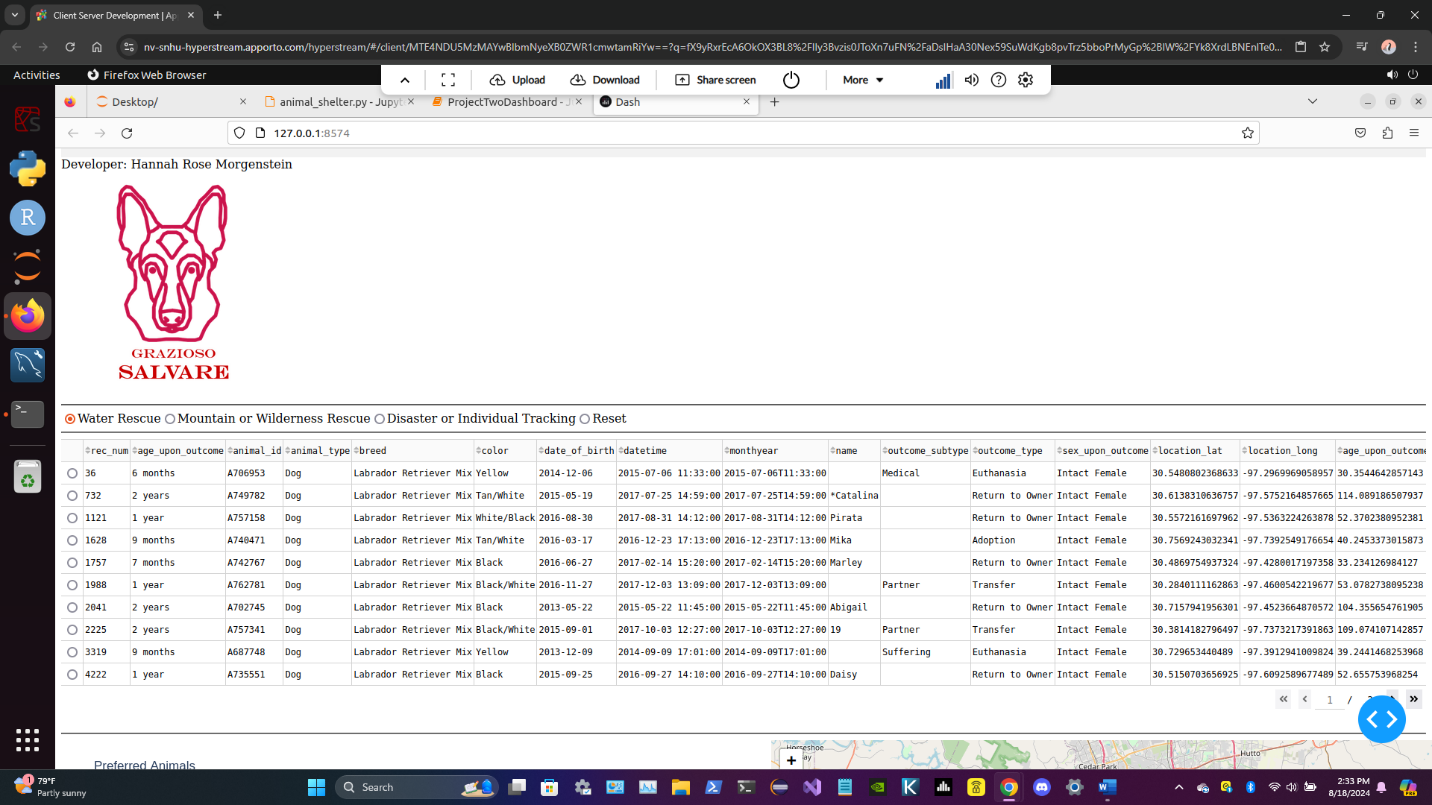
**A screenshot of a computer

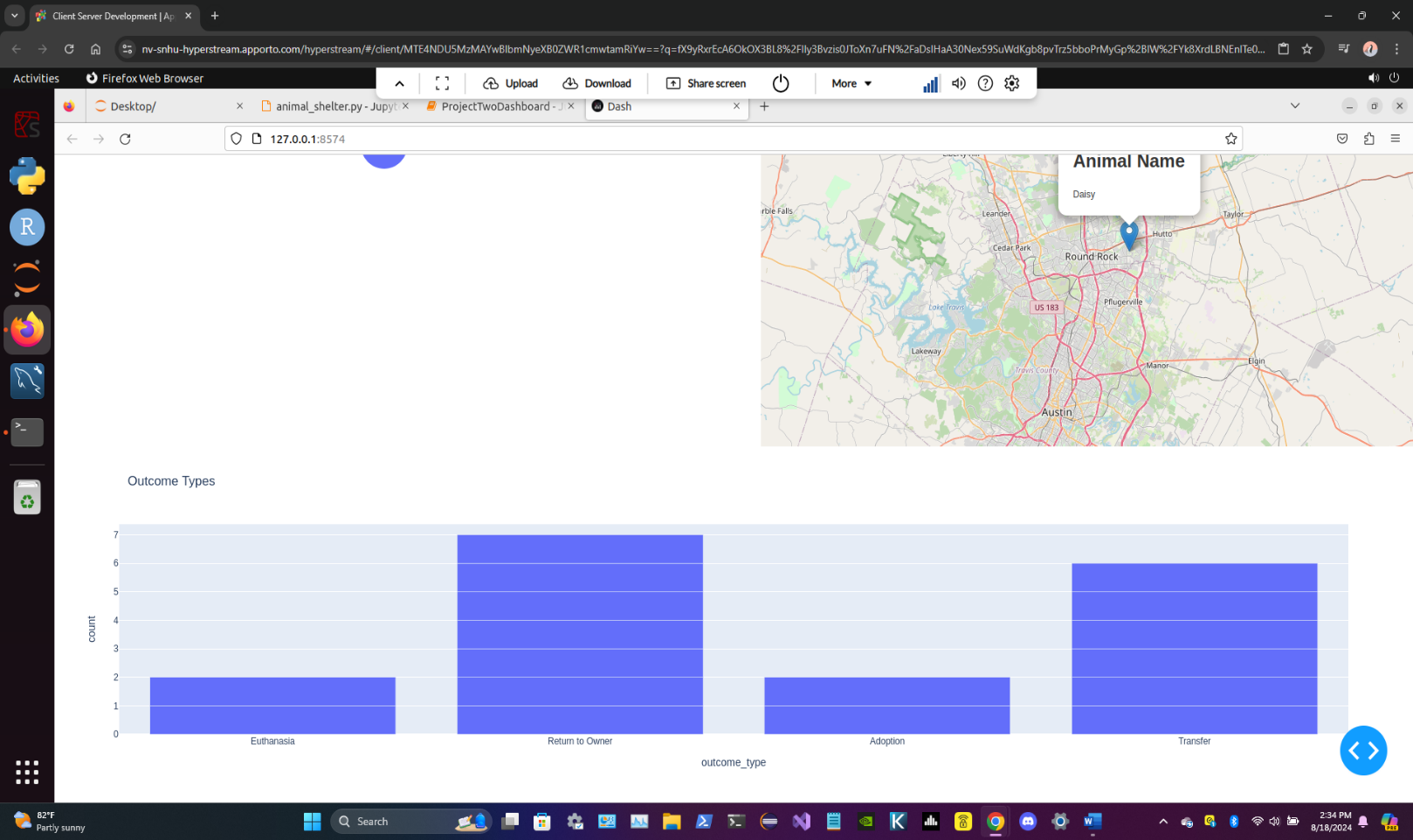
Description automatically generated**

**A screenshot of a computer

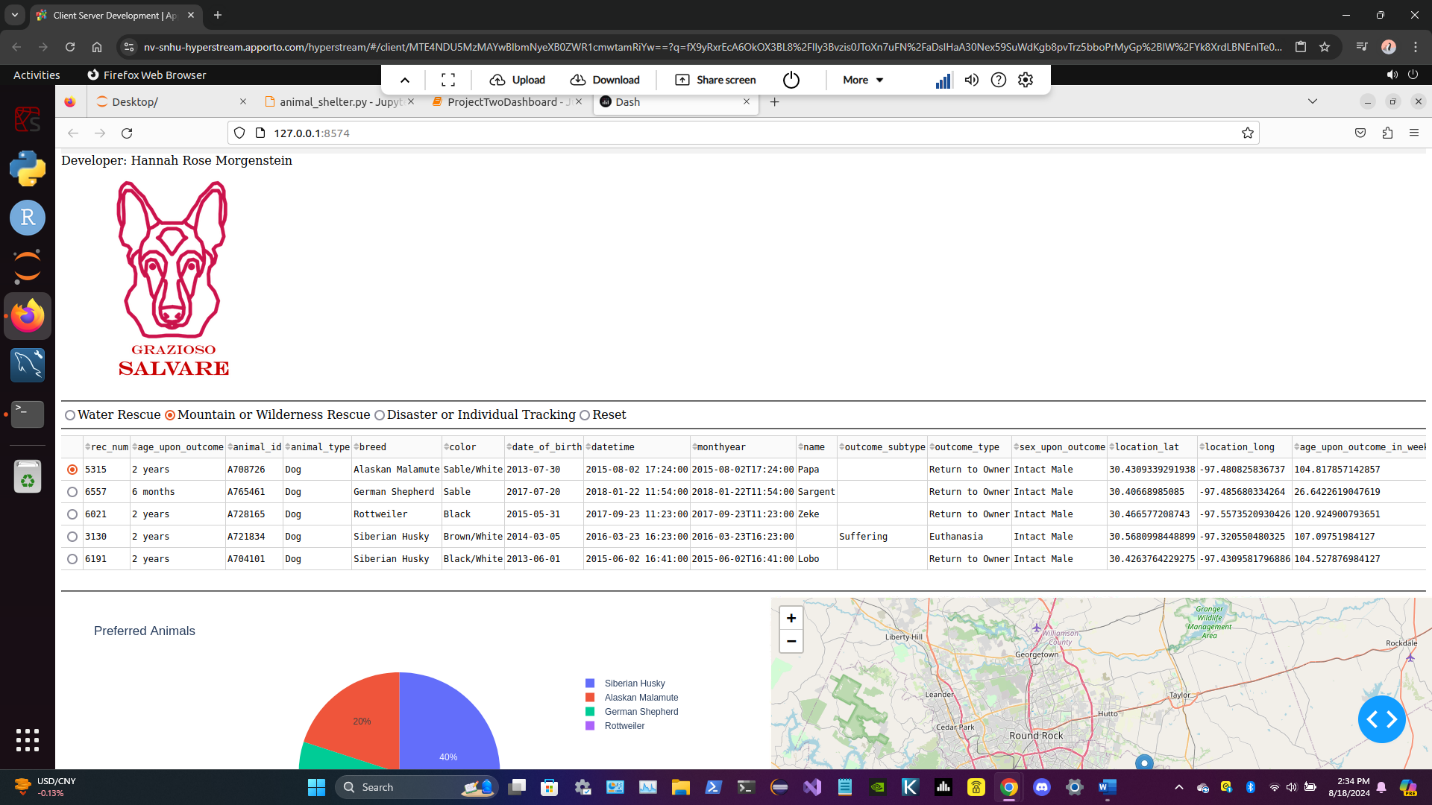
Description automatically generated**

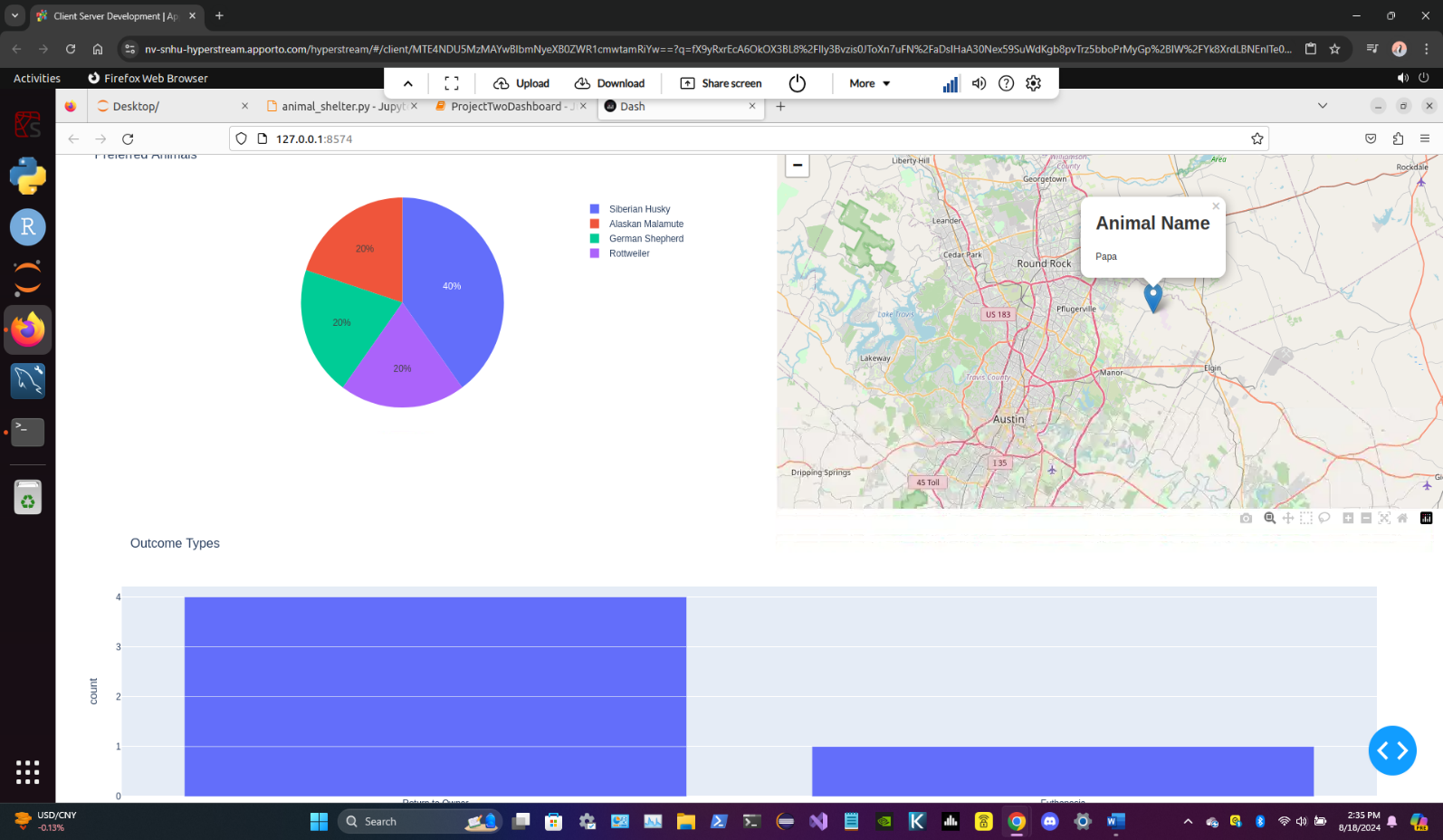
This is the “Water Rescue” data filter. In these screenshots, I am featuring a closer look at the bar graph I created which displays outcome types within the specified data filter. There is also a view of the geolocation chart which displays the selected animal’s name and location:

****

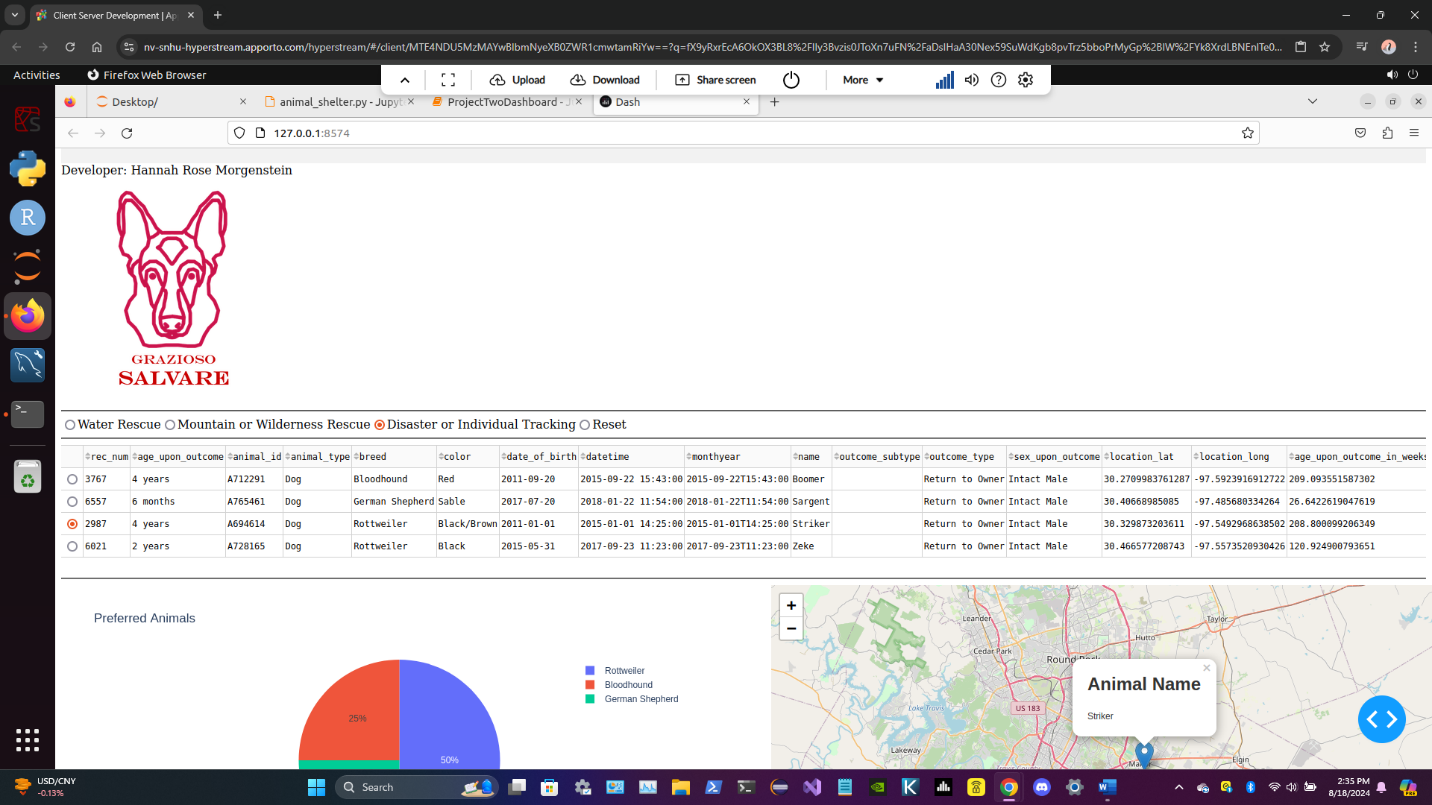
****

Here is the “Mountain or Wilderness Rescue” data filter. These screenshots feature the secondary chart that I have created, which is a pie chart that shows the percentage of preferred animals (according to queries) within each specified data filter:

****

****

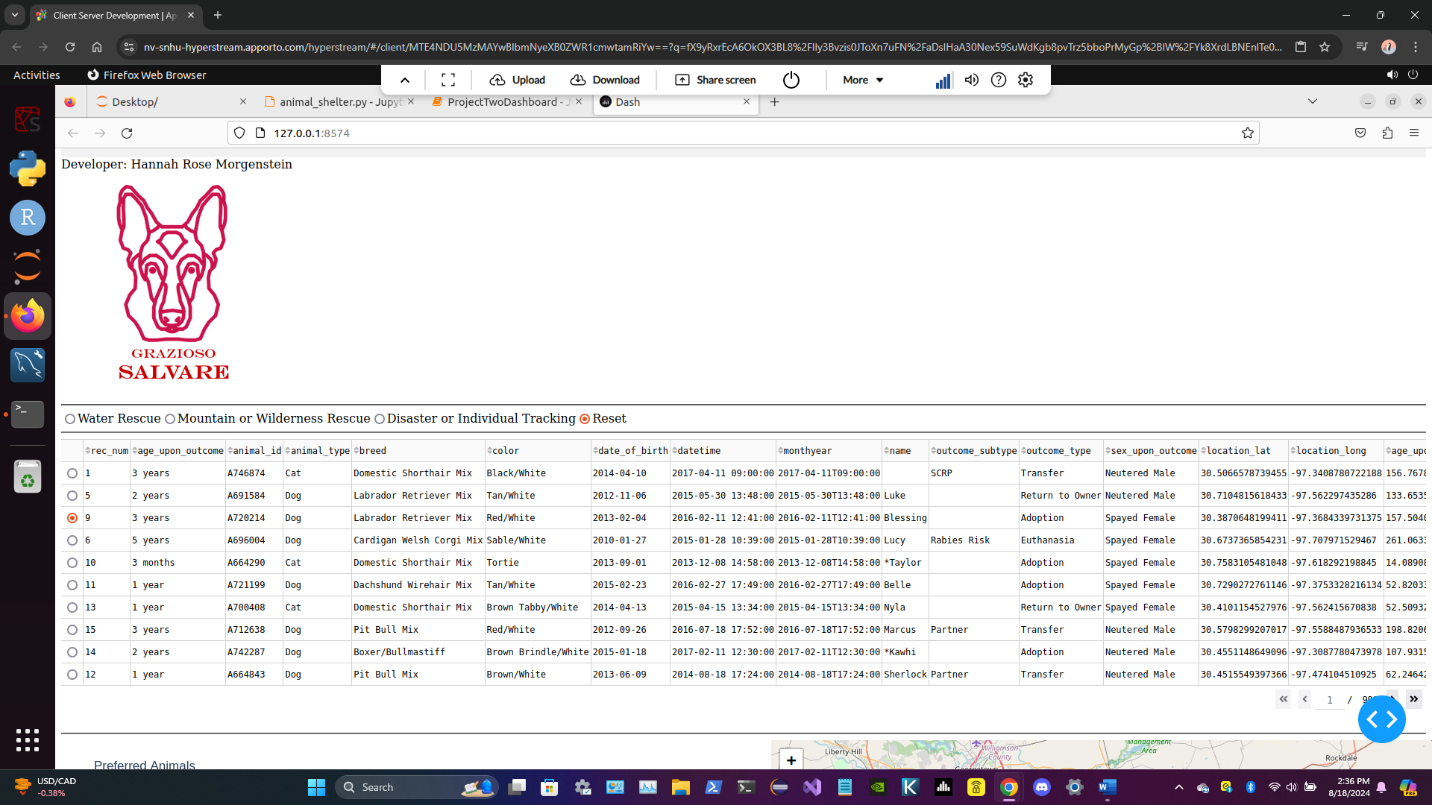
This is the “Disaster or Individual Tracking” data filter:

****

**A computer screen shot of a map

Description automatically generated**

The dashboard back to the “Reset” data filter (or unfiltered data):

****

**A screenshot of a computer

Description automatically generated**

**Tools Used:**

**Dash Framework**

* **Why**: Dash provides a powerful way to build interactive web applications using Python. It simplifies the creation of web-based data visualizations and interactive components, making it ideal for this project.
* **Components Used:**
  + *‘dcc.RadioItems’* for filtering options.
  + *‘dash\_table.DataTable’* for displaying and interacting with data.
  + *‘dcc.Graph’* for rendering the pie chart and bar chart.
  + *‘dash\_leaflet’* for integrating a map with dynamic markers.

**MongoDB**

* **Why**: MongoDB is a flexible NoSQL database that allows for efficient storage and retrieval of unstructured data. It supports complex queries, making it ideal for dynamically filtering animal records.
* **Capabilities**: The database's ability to handle large volumes of data and support for complex queries enables efficient filtering and real-time data updates.

**Python Libraries**

* **Pandas**: Used for data manipulation and cleaning.
* **Plotly Express**: Used for creating interactive charts.
* **NumPy**: Used for numerical operations and data handling.
* **Base64**: Used for encoding images for embedding in the dashboard.

**Explanation of Tools:**

**MongoDB**

* **Purpose:** MongoDB is used as the database model due to its flexibility in handling various data structures. It integrates seamlessly with Python via the pymongo library, allowing for complex queries and efficient data manipulations.

**Dash Framework**

* **Purpose:** Dash provides the view and controller structure for the web application. It enables the creation of interactive web components and dynamic visualizations using Python, aligning with the project's requirements.

**Resources and Software:**

* **Dash Documentation**: Dash by Plotly
* **MongoDB Documentation**: [MongoDB](https://www.mongodb.com/)
* **Jupyter Notebook:** For development, running, and interaction with the dashboard.

**Steps Taken:**

1. **Project Planning:**

* Analyzed client requirements and features for the dashboard based on the provided specifications.
* Designed a project plan and timeline to meet the client's needs and deadlines.

1. **Database Integration:**

* Developed and tested a CRUD Python module for managing records in MongoDB.
* Ensured secure and efficient database connections and queries.

1. **Dashboard Development:**

* Designed and implemented the dashboard layout using Dash, including the integration of all required widgets.
* Configured interactive components for filtering and data display based on user input.

1. **Data Visualization:**

* Created and customized charts and maps to visualize data effectively, ensuring they respond to user filters.
* Applied best practices for data presentation to enhance user experience.

1. **Testing and Debugging:**

* Conducted thorough testing of the dashboard functionality, including filter options and data updates.
* Identified and resolved bugs and issues to ensure smooth operation and accuracy of the dashboard.

1. **Documentation:**

* Compiled a comprehensive README file detailing project setup, usage instructions, and any relevant information.
* Documented code for clarity and maintainability, following best practices and coding standards.

**Challenges and Solutions**

1. **Database Connection Issues**
   * **Challenge:** Establishing a reliable connection to the MongoDB database proved challenging, especially when handling cases where the database might be unreachable or credentials incorrect.
   * **Solution:** Implemented robust error handling in the CRUD Python module to manage connection failures gracefully. This involved:
     + Adding try-except blocks around database connection attempts.
     + Providing informative error messages when connection fails.
     + Ensuring fallback behavior by setting a default state (e.g., empty DataFrame) when data retrieval fails.
2. **Component Interaction**
   * **Challenge:** Ensuring that the dashboard components (DataTable, charts, and map) updated seamlessly based on user interactions was complex due to the interdependencies between them.
   * **Solution:** Managed this through careful design of Dash callbacks, including:
     + **Modular Callback Functions:** Designed callback functions to handle multiple outputs and inputs efficiently, ensuring that changes in one component (e.g., filter selection) propagate correctly to others.
     + **Data Synchronization:** Ensured that data is consistently synchronized across the DataTable, pie chart, bar graph, and map by updating all relevant components within a single callback.
     + **Error Handling in Callbacks:** Added error handling within callback functions to manage cases where data might be missing or incorrectly formatted, ensuring the dashboard remains functional and user-friendly.
3. **Data Integrity and Validation**
   * **Challenge:** Maintaining data integrity and ensuring that the data displayed in the dashboard is accurate and up-to-date.
   * **Solution:**
     + Implemented data validation checks before processing and displaying data. For example, verified that required columns (e.g., location coordinates) are present in the DataFrame before generating the map.
     + Added conditional logic to handle scenarios where data might be incomplete or invalid, providing default values or error messages as needed.
4. **Performance Optimization**
   * **Challenge:** Managing performance issues related to large datasets and ensuring the dashboard remained responsive.
   * **Solution:**
     + Optimized MongoDB queries to retrieve only the necessary data based on user filters, reducing the amount of data processed and displayed.
5. **User Experience and Usability**
   * **Challenge:** Designing an intuitive user interface that provides a clear and engaging user experience.
   * **Solution:**
     + Conducted testing to refine the dashboard layout and interactions.
     + Added clear labels, tooltips, and instructions to guide users in using the dashboard effectively.
     + Ensured that interactive elements (e.g., filters, DataTable selections) are responsive and provide immediate feedback to users.

**Developed by:** Hannah Rose Morgenstein

**Date:** 08-18-2024