**Grazioso Salvare Dashboard README**

## About the Project/Project Title

This README file outlines the Grazioso Salvare Rescue Dashboard project, detailing its purpose, tools, functionalities, and completion steps. The dashboard enables users to filter animals based on specific rescue criteria, access detailed data on selected animals, and view their geolocation on an interactive map.

## Motivation

The Grazioso Salvare Rescue Dashboard is an intuitive web application developed to assist Grazioso Salvare in locating and filtering animals for specialized rescue training purposes. This dashboard allows users to filter animals based on criteria such as Water Rescue, Mountain or Wilderness Rescue, and Disaster and Individual Tracking. Users can view details of animals, including breed, age, and sex, and locate selected animals on an interactive map.

## Tools

**MongoDB** is a flexible and scalable database perfect for handling semi-structured animal data. Its document-oriented structure is an advantage for storing animal records, and it integrates effortlessly with Python via the PyMongo library.

**The Dash Framework** is an excellent foundation for building web applications with interactive components. It allows developers to create dynamic user interfaces easily, making it perfect for real-time data visualization and user engagement. With its user-friendly design and seamless integration with Python libraries like Plotly, Dash enables the rapid creation of stunning data visualizations. This efficiency and flexibility make it a top choice for data-driven projects.

**Dash Leaflet**: Used for the geolocation map within the dashboard, allowing for custom map markers and popups. Set up the MongoDB connection with the necessary animal data from the Austin Animal Center.

**Installation**

**Clone the Repository**: Open your terminal (or Command Prompt) and clone the project repository by running: “git clone <https://github.com/TreRay/CS340>”. Then setup a virtual environment on Windows, MacOs or Linnux.

**Install Project Dependencies**: To install the required Python packages listed in `requirements.txt`, run the following command: “pip install -r requirements.txt”

**Ensure that MongoDB is running** and that the Austin Animal Center data has been populated. Update your script's database connection details: Host: The default is "localhost" for local setups. Port: Use the default unless it has been modified. Credentials: Add the necessary authentication credentials (username and password).

**MongoDB Credentials:**

**cd /usr/local/datasets**

**mongoimport --username="aacuser" \**

**--password="Tre1990" \**

**--port=${MONGO\_PORT} \**

**--host=${MONGO\_HOST} \**

**--type csv \**

**--db AAC \**

**--headerline \**

**--collection animals \**

**--authenticationDatabase admin \**

**--drop ./aac\_shelter\_outcomes.csv**

**To start the application**, run `ProjectTwoDashboard.ipynb`.

**Access the Dashboard**: Once the application starts, open a web browser with:”

<http://127.0.0.1:8050>”

**Resources**

<https://www.mongodb.com/docs/>

<https://www.dash-leaflet.com/>

<https://jupyter.org/>

## Code

*The Code provided from the Crud\_ops.py file contains the ANIMALSHELTER class. This class creates, reads, updates, and deletes documents within the database.*

*from bson.objectid import ObjectId*

*from pymongo import MongoClient, errors*

*import logging*

*import os*

*#TR 10/24/2024*

*# CRUD module with the AnimalShelter class*

*class AnimalShelter:*

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

*def \_\_init\_\_(self, username='aacuser', password='Tre1990', host=None, port=None, db='AAC', collection='animals', auth\_db='admin'):*

*#Initialize the AnimalShelter class with MongoDB connection parameters*

*# Fetch environment variables if host and port are not provided*

*self.USER = username*

*self.PASS = password*

*self.HOST = host if host else os.getenv('MONGO\_HOST')*

*# Check if port is provided and valid, otherwise default to 27017*

*if port and isinstance(port, str) and port.isdigit():*

*self.PORT = int(port)*

*else:*

*self.PORT = int(os.getenv('MONGO\_PORT', 27017))*

*print(f"HOST: {self.HOST}, PORT: {self.PORT}") #TR 10/24/2024*

*self.DB = db*

*self.COLLECTION = collection*

*self.AUTH\_DB = auth\_db*

*# Construct MongoDB URI*

*mongo\_uri = f"mongodb://{self.USER}:{self.PASS}@{self.HOST}:{self.PORT}/{self.AUTH\_DB}"*

*# Initialize connection*

*try:*

*self.client = MongoClient(mongo\_uri)*

*self.database = self.client[self.DB]*

*self.collection = self.database[self.COLLECTION]*

*logging.info(f"Connected to MongoDB at {self.HOST}:{self.PORT}")*

*except errors.PyMongoError as e:*

*logging.error(f"Failed to connect to MongoDB at {self.HOST}:{self.PORT}. Error: {e}")*

*raise*

*def create(self, data):*

*#Creates a new document in the database.*

*if data: #TR 10/24/2024*

*try:*

*result = self.collection.insert\_one(data)*

*return {"status": "success", "inserted\_id": str(result.inserted\_id), "acknowledged": result.acknowledged}*

*except errors.PyMongoError as e:*

*logging.error(f"Error while inserting data: {e}")*

*return {"status": "error", "message": str(e)}*

*else:*

*raise ValueError("Data cannot be empty")*

*def read(self, query=None):*

*#Searches for documents in the database based on a query*

*if query is None:*

*query = {} # Default to an empty query to fetch all records*

*try:*

*results = list(self.collection.find(query))*

*return {"status": "success", "data": results}*

*except errors.PyMongoError as e:*

*logging.error(f"Error while reading data: {e}")*

*return {"status": "error", "message": str(e)}*

*def update(self, query, updated\_data): #TR 10/24/2024*

*#Update documents in the database*

*if query and updated\_data:*

*try:*

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

*return {"status": "success", "modified\_count": result.modified\_count}*

*except errors.PyMongoError as e:*

*logging.error(f"Error while updating data: {e}")*

*return {"status": "error", "message": str(e)}*

*else:*

*raise ValueError("Query and updated data cannot be empty")*

*def delete(self, query):*

*#Delete documents from the database*

*if query:*

*try:*

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

*if result.deleted\_count > 0:*

*return {"status": "success", "deleted\_count": result.deleted\_count}*

*else:*

*return {"status": "warning", "message": "No documents matched the query, no documents deleted"}*

*except errors.PyMongoError as e:*

*logging.error(f"Error while deleting data: {e}")*

*return {"status": "error", "message": str(e)}*

*else:*

*raise ValueError("Query cannot be empty")*

*def close\_connection(self): #TR 10/24/2024*

*"""Close the MongoDB connection."""*

*self.client.close()*

**Code**

This code from ProjectTwoDashboard.ipynb creates a Python dashboard using Dash and MongoDB to display animals by rescue type for Grazioso Salvare. Users can select a rescue type from a dropdown to view a filtered data table and a pie chart of breed distribution. Selecting an animal also shows its location on a map. The app updates data interactively using Dash callbacks.

**from jupyter\_dash import JupyterDash**

**import dash\_leaflet as dl**

**from dash import dcc, html, dash\_table**

**from dash.dependencies import Input, Output**

**import base64**

**import pandas as pd**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import plotly.express as px**

**#TR 10/24/2024**

**# Import ANIMALSHELTER Class**

**from crud\_ops import AnimalShelter**

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

**# MongoDB username and password**

**username = "aacuser"**

**password = "Tre1990"**

**# Initialize AnimalShelter class for database operations**

**db = AnimalShelter(username, password)**

**# Function to fetch data based on rescue type**

**def fetch\_data(rescue\_type): #TR 10/24/2024**

**query = {}**

**if rescue\_type == 'Water Rescue':**

**query = {**

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

**"sex\_upon\_outcome": "Intact Female",**

**"age\_upon\_outcome\_in\_weeks": {"$gte": 26, "$lte": 156}**

**}**

**elif rescue\_type == 'Mountain or Wilderness Rescue':**

**query = {**

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

**"sex\_upon\_outcome": "Intact Male",**

**"age\_upon\_outcome\_in\_weeks": {"$gte": 26, "$lte": 156}**

**}**

**elif rescue\_type == 'Disaster or Individual Tracking':**

**query = {**

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

**"sex\_upon\_outcome": "Intact Male",**

**"age\_upon\_outcome\_in\_weeks": {"$gte": 20, "$lte": 300}**

**}**

**return db.read(query)**

**# Fetch initial data**

**data = fetch\_data('Reset')**

**if data['status'] == 'success' and data['data']: #TR 10/24/2024**

**df = pd.DataFrame.from\_records(data['data'])**

**if '\_id' in df.columns:**

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

**else:**

**print(f"Error fetching data: {data.get('message', 'Unknown error')}")**

**df = pd.DataFrame()**

**########################## Dashboard Layout / View##########################**

**# Initialize the Dash app**

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

**# Add the Grazioso Salvare logo image**

**image\_filename = '/home/tremainerayne\_snhu/Desktop/Grazioso Salvare Logo.png'**

**try:**

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

**except Exception as e:**

**print(f"Error encoding image: {e}")**

**encoded\_image = None**

**# Define the layout for the dashboard**

**app.layout = html.Div([**

**html.Center(html.B(html.H1('Tremaine Rayner'))),**

**html.Hr(),**

**html.Img(src=f'data:image/png;base64,{encoded\_image}') if encoded\_image else html.Div("Image not found"),**

**html.Hr(),**

**dcc.Dropdown(**

**id='rescue-type-dropdown',**

**options=[**

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

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

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

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

**],**

**value='Reset',**

**placeholder="Select a rescue type"**

**),**

**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',**

**row\_selectable='single',**

**selected\_rows=[],**

**page\_size=10,**

**style\_table={'height': '300px', 'overflowY': 'auto'},**

**style\_cell={'textAlign': 'left'},**

**),**

**html.Br(),**

**html.Hr(), #TR 10/24/2024**

**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##############################################**

**# Callback to update the data based on rescue type selection**

**@app.callback(**

**[Output('datatable-id', 'data'),**

**Output('graph-id', 'children'),**

**Output('map-id', 'children')],**

**[Input('rescue-type-dropdown', 'value'),**

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

**def update\_dashboard(rescue\_type, selected\_rows):**

**data = fetch\_data(rescue\_type)**

**if data['status'] == 'success' and data['data']:**

**dff = pd.DataFrame.from\_records(data['data'])**

**if '\_id' in dff.columns:**

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

**else:**

**dff = pd.DataFrame()**

**# Update DataTable**

**table\_data = dff.to\_dict('records') #TR 10/24/2024**

**# Update Pie Chart**

**if not dff.empty:**

**fig = px.pie(dff, names='breed', title='Animal Breeds')**

**graph = dcc.Graph(figure=fig)**

**else:**

**graph = html.Div("No data available.")**

**# Update Map**

**if not dff.empty and selected\_rows is not None and len(selected\_rows) > 0:**

**if len(selected\_rows) > 0 and selected\_rows[0] < len(dff):**

**row = selected\_rows[0]**

**if 'location\_lat' in dff.columns and 'location\_long' in dff.columns:**

**lat = dff.iloc[row]['location\_lat']**

**long = dff.iloc[row]['location\_long']**

**name = dff.iloc[row]['name']**

**map\_component = [**

**dl.Map(style={'width': '1000px', 'height': '500px'}, center=[lat, long], zoom=10, children=[**

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

**dl.Marker(position=[lat, long], children=[**

**dl.Tooltip(dff.iloc[row]['breed']),**

**dl.Popup([html.H1(name), html.P(dff.iloc[row]['breed'])])**

**])**

**])**

**]**

**else:**

**map\_component = html.Div("Location data not available.")**

**else: #TR 10/24/2024**

**map\_component = html.Div("No map data available.")**

**else:**

**map\_component = html.Div("No map data available.")**

**return table\_data, graph, map\_component**

**# Callback to highlight selected column**

**@app.callback(**

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

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

**def update\_styles(selected\_columns):**

**if selected\_columns is None:**

**selected\_columns = []**

**return [{**

**'if': {'column\_id': i},**

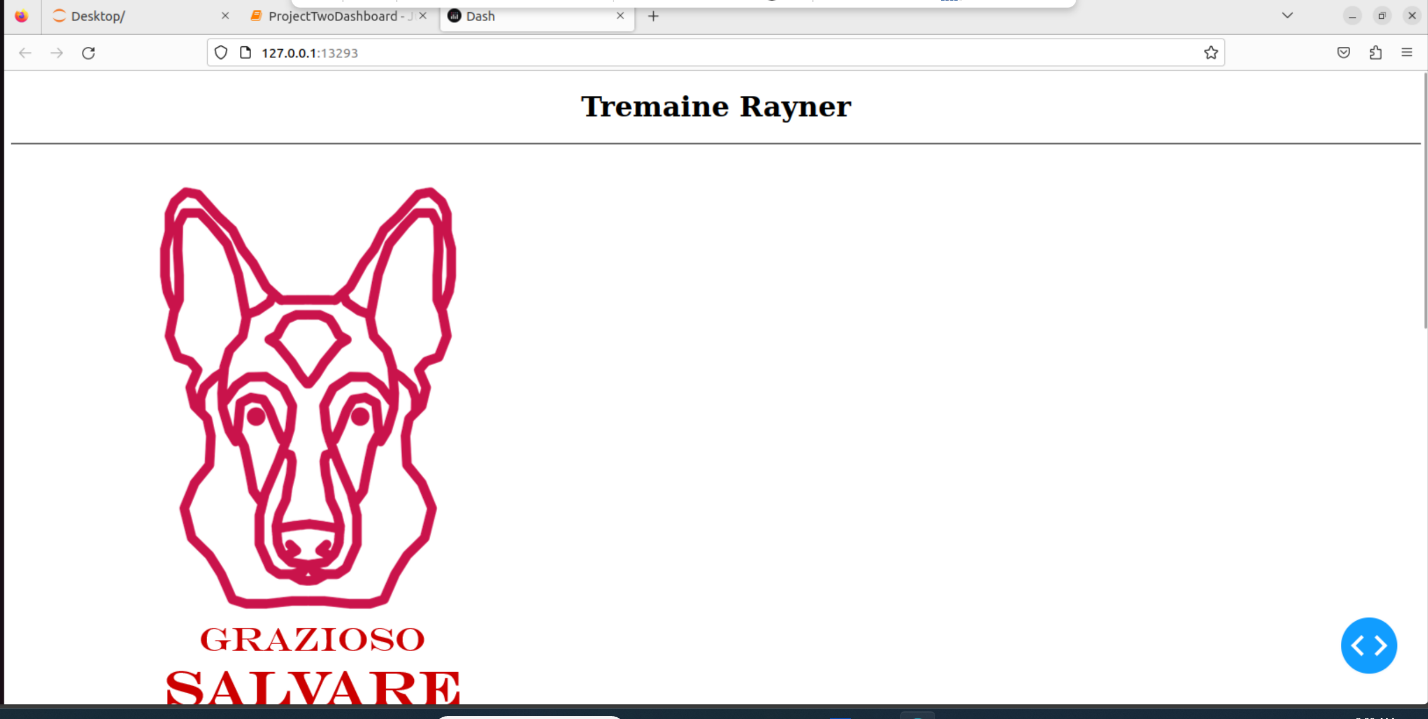
**'background\_color': '#D2F3FF' #TR 10/24/2024**

**} for i in selected\_columns]**

**# Run the app**

**app.run\_server(debug=True)**

**Dashboard Screenshots**

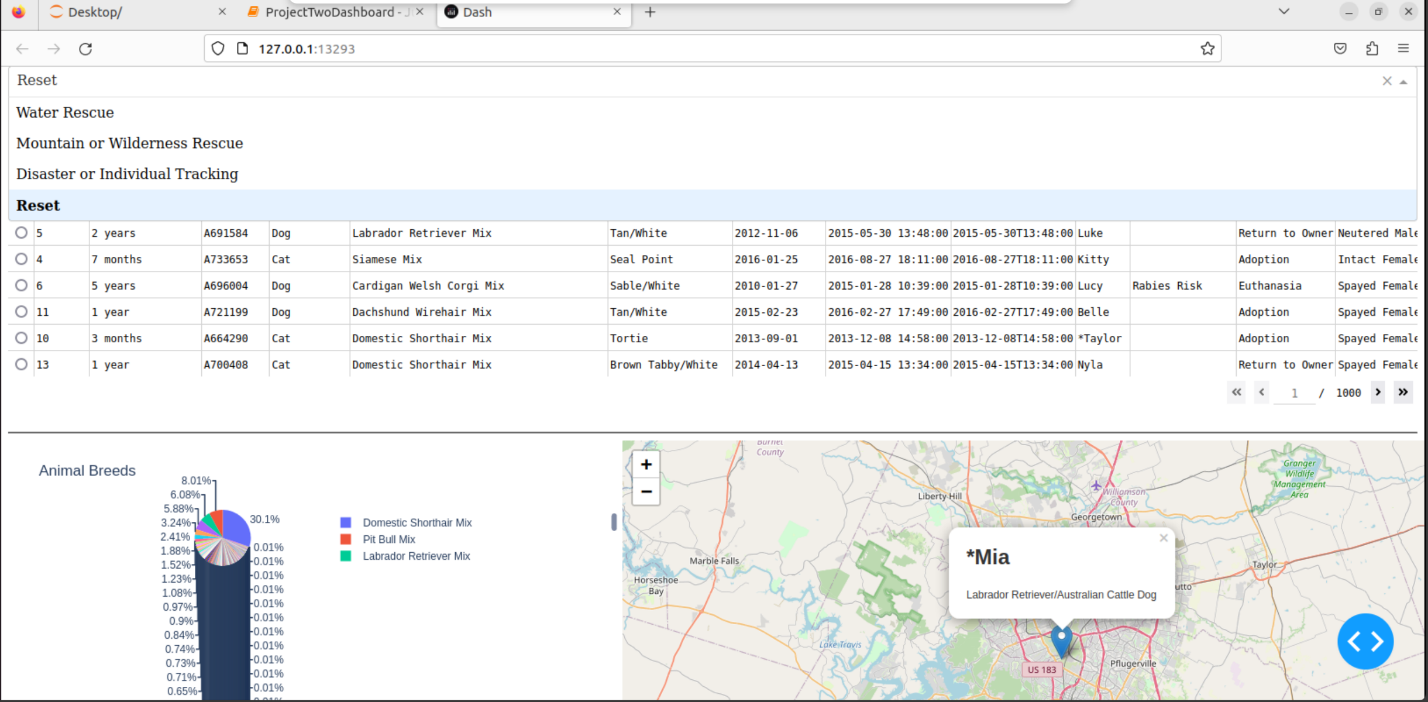
****

**Screenshot of the dashboard showing my unique identifier.**

A screenshot of a computer

Description automatically generated

**Users can click on an animal then the map will show its location.**



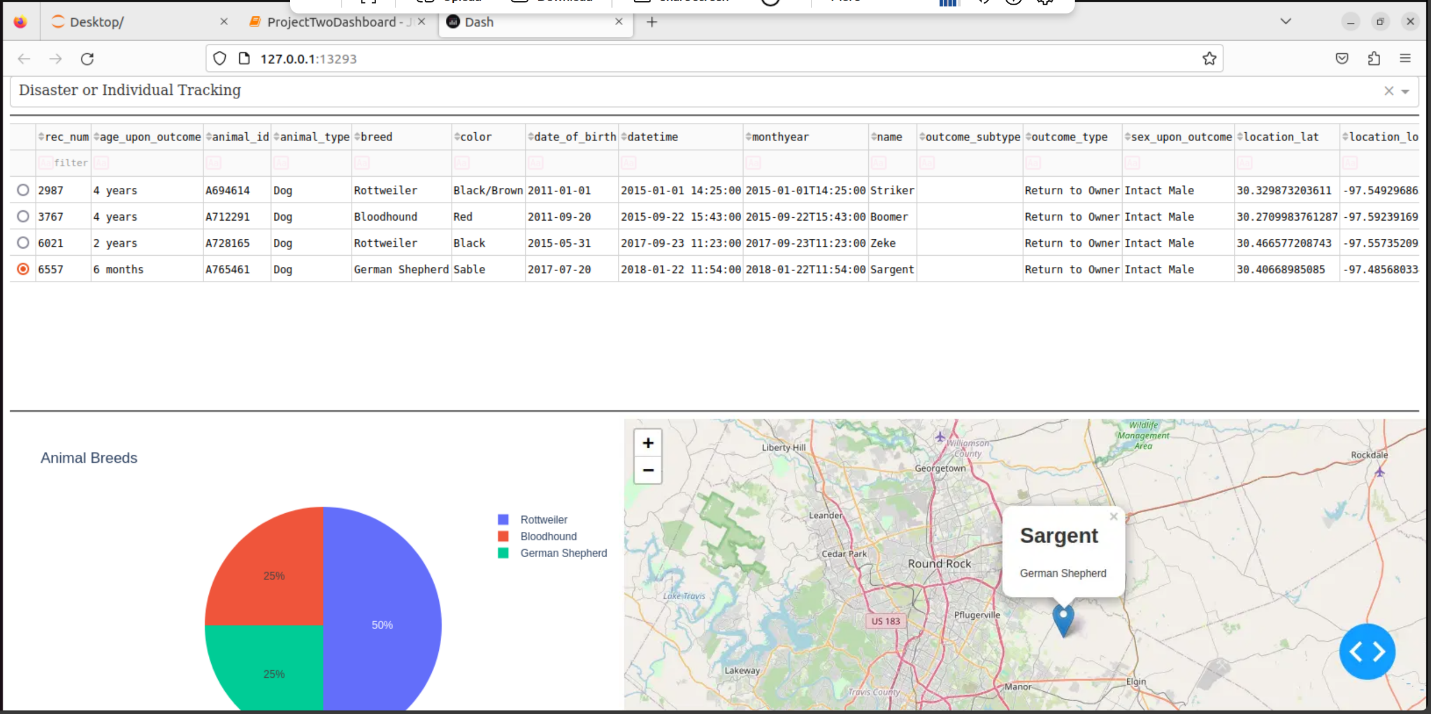
**Users can view the drop-down menu to select from Water rescue, Mountain or wilderness rescue, and Disaster or individual tracking. Clicking reset will return users to the basic search.**

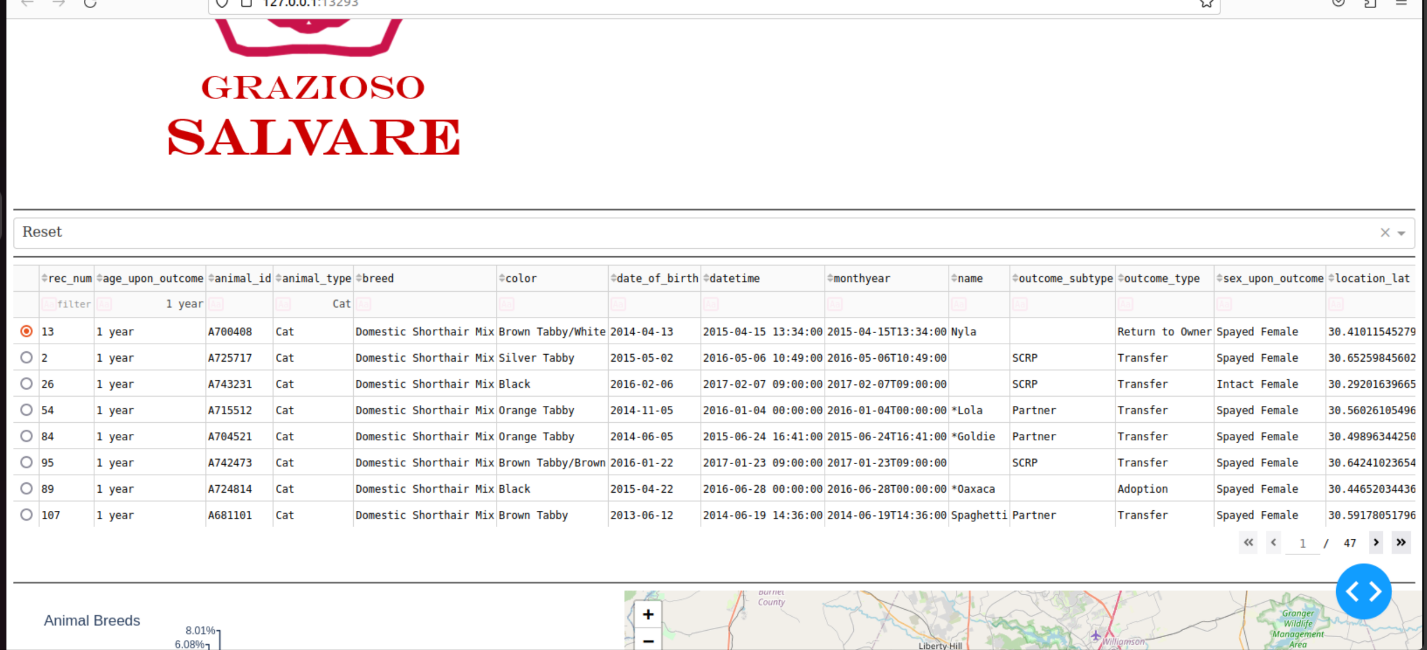
A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated





**Users can also use the filters to search for an animal based on specific criteria like age and breed.**

## Contact

Your name: Tremaine Rayner