Southern New Hampshire University

CS-340 AAC Dash Application README

About the Project/Project Title

This project is a Dash dashboard that displays potential animal rescue recruits from a MongoDB database. Users can filter animals by different rescue scenarios (Water, Mountain, Disaster), view key animal data in a table, see breed distribution charts, and get geolocation info on selected animals.

Installation

Required tools:

- *Python 3.x*
- Jupyter Notebook
- Packages:
  - jupyter-dash,
  - dash,
  - dash-leaflet,
  - pandas,
  - plotly,
  - pymongo,
  - crud_with_ctor_args

pip command for package installation:

pip install jupyter-dash dash dash-leaflet pandas plotly pymongo

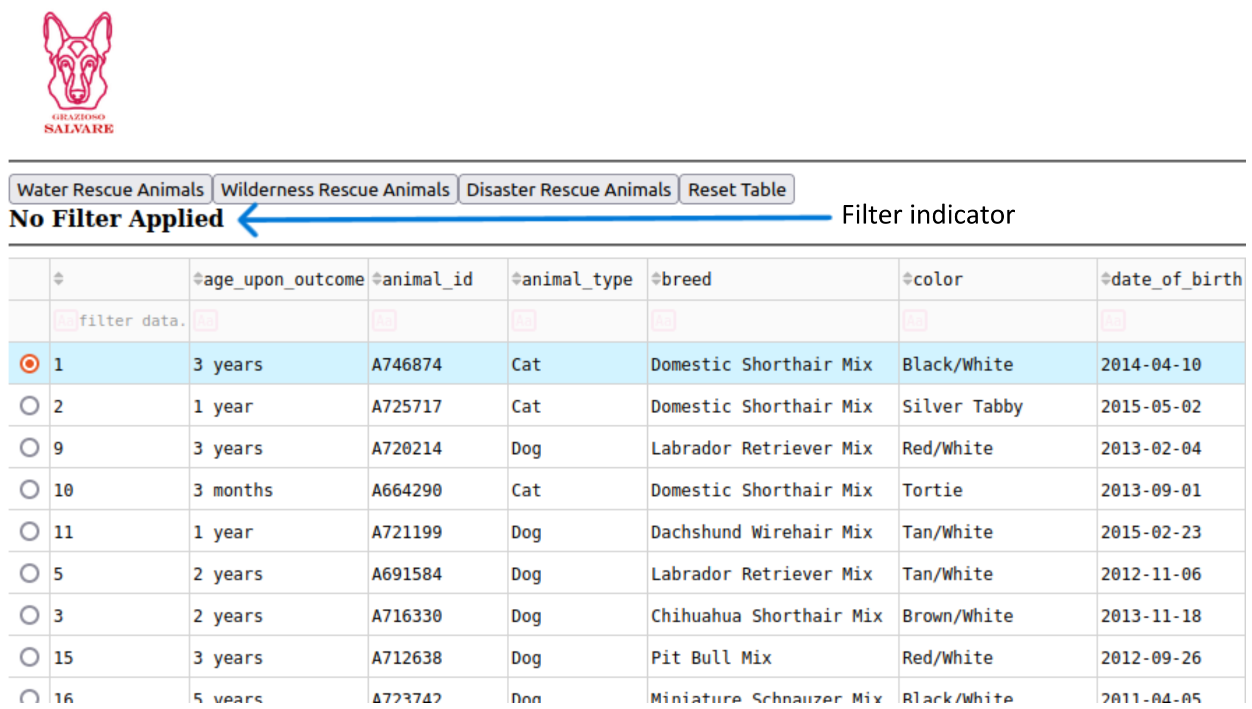
***NOTE:** *The crud_with_ctor_args package cannot be installed via pip. Download/copy the single python file to use it.* (in theory that file would be available in the repo where this readme lives)

Getting Started

Launch the Jupyter notebook, run the lone cell containing code to start the dashboard server. After running, a link to the Dash page will appear in the Jupyter output. Use the interactive buttons inside the notebook's Dash app to filter animals by rescue scenarios. Select a row's radio button to update the map widget to show the location of the animal in the row.

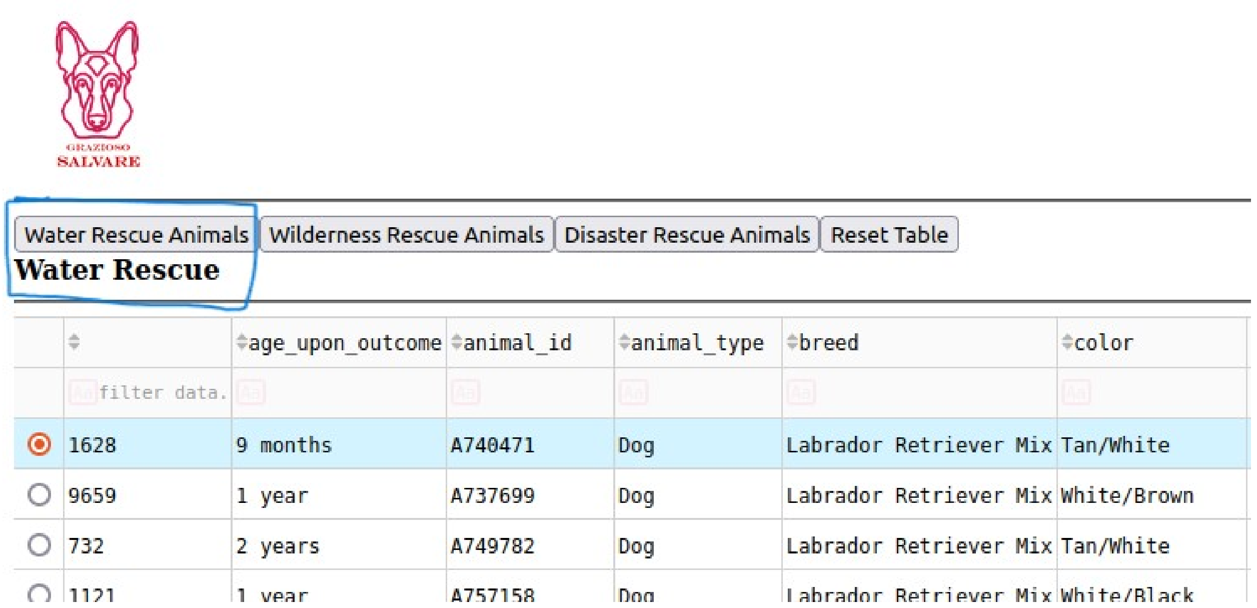
Usage

After following the instructions in "Getting Started" - you should have access to the webpage. The use of the application is fairly straightforward. Initially, the table is shown with no filters applied to the dataset, and each item in the set will appear in a distinct table row. There is a label underneath the filter buttons which will report the active filter on the data set. On start, you'll see that no filters are applied, screenshot below:



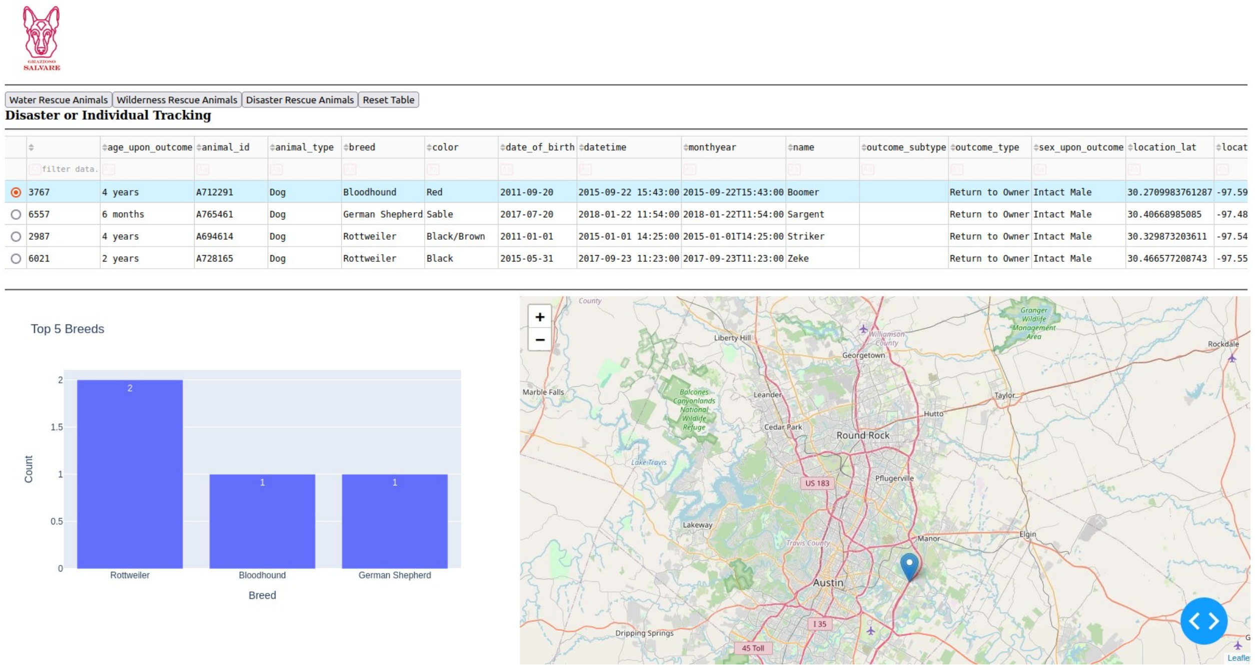
		age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth
●	1	3 years	A746874	Cat	Domestic Shorthair Mix	Black/White	2014-04-10
○	2	1 year	A725717	Cat	Domestic Shorthair Mix	Silver Tabby	2015-05-02
○	9	3 years	A720214	Dog	Labrador Retriever Mix	Red/White	2013-02-04
○	10	3 months	A664290	Cat	Domestic Shorthair Mix	Tortie	2013-09-01
○	11	1 year	A721199	Dog	Dachshund Wirehair Mix	Tan/White	2015-02-23
○	5	2 years	A691584	Dog	Labrador Retriever Mix	Tan/White	2012-11-06
○	3	2 years	A716330	Dog	Chihuahua Shorthair Mix	Brown/White	2013-11-18
○	15	3 years	A712638	Dog	Pit Bull Mix	Red/White	2012-09-26
○	16	5 years	A723742	Dog	Miniature Schnauzer Mix	Black/White	2011-04-05

After selecting a filter button by clicking on it, the data in the table will refresh, and this label will indicate the newly applied filter, I've highlighted the button clicked, note how the text now reflects the same text description:



Water Rescue

		age_upon_outcome	animal_id	animal_type	breed	color
	filter data.					
●	1628	9 months	A740471	Dog	Labrador Retriever Mix	Tan/White
○	9659	1 year	A737699	Dog	Labrador Retriever Mix	White/Brown
○	732	2 years	A749782	Dog	Labrador Retriever Mix	Tan/White
○	1121	1 year	A757158	Dog	Labrador Retriever Mix	White/Black



Disaster or Individual Tracking

		age_upon_outcome	animal_id	animal_type	breed	color	date_of_birth	datetime	monthyear	name	outcome_subtype	outcome_type	sex_upon_outcome	location_lat	locat
	filter data.														
●	3767	4 years	A712291	Dog	Bloodhound	Red	2011-09-20	2015-09-22 15:43:00	2015-09-22T15:43:00	Boomer		Return to Owner	Intact Male	30.2709983761287	-97.59
○	6557	6 months	A765461	Dog	German Shepherd	Sable	2017-07-20	2018-01-22 11:54:00	2018-01-22T11:54:00	Sargent		Return to Owner	Intact Male	30.40668985085	-97.48
○	2987	4 years	A694614	Dog	Rottweiler	Black/Brown	2011-01-01	2015-01-01 14:25:00	2015-01-01T14:25:00	Striker		Return to Owner	Intact Male	30.329873203611	-97.54
○	6021	2 years	A728165	Dog	Rottweiler	Black	2015-05-31	2017-09-23 11:23:00	2017-09-23T11:23:00	Zeke		Return to Owner	Intact Male	30.466577208743	-97.55

At the bottom of the page you'll see a bar chart breaking down the dataset's top 5 most common breeds (which also changes according to filters applied and the resulting data), and by selecting a table row's radio button, the map will display the recorded location of the row's animal.

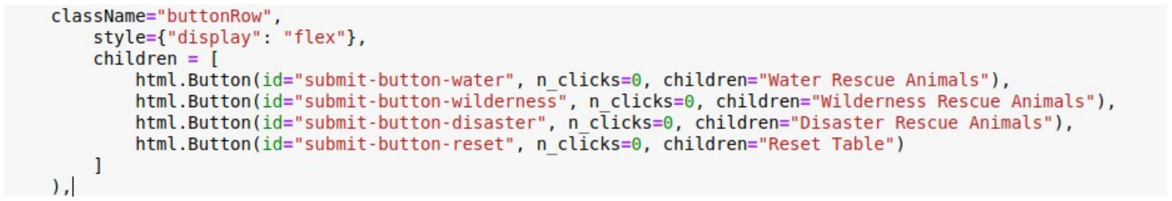
Extending

If you want to create your own filters, either replace one of the existing buttons or add a new one, register the button's ID with the callback as an Input, the value used for this input is irrelevant (currently all set to pass in n_clicks, but it doesn't matter as the program uses Dash.callback_context package to identify the button calling the function).

Code Example

I'll walk through the process of adding a simple button that will update the data displayed to only show cats.

First we'll add an HTML button element, which is done via following the existing buttons syntax, and adding it to the same "children" array in the app.layout definition.

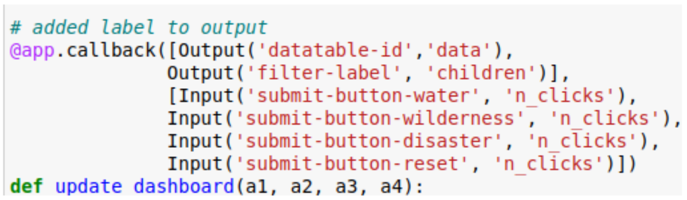


```
className="buttonRow",
    style={"display": "flex"},
    children = [
        html.Button(id="submit-button-water", n_clicks=0, children="Water Rescue Animals"),
        html.Button(id="submit-button-wilderness", n_clicks=0, children="Wilderness Rescue Animals"),
        html.Button(id="submit-button-disaster", n_clicks=0, children="Disaster Rescue Animals"),
        html.Button(id="submit-button-reset", n_clicks=0, children="Reset Table")
    ]
),
```

So our Cat button can be added by including the following line to **children** list:

html.Button(id="submit-button-cats", n_clicks=0, children="Cats")

Next we must update the callback used to update the dashboard to receive this button as an input. The function labeled update_dashboard has the @app.callback annotation above it, this is where we'll register our new Input.

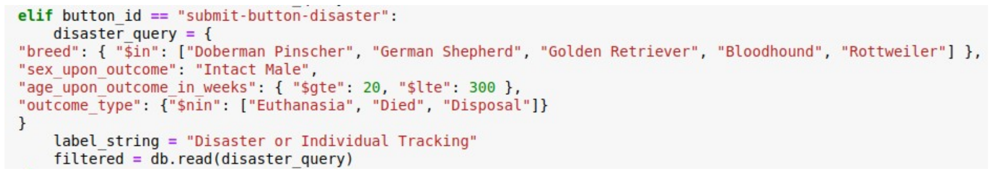


```
# added label to output
@app.callback([Output('datatable-id','data'),
               Output('filter-label', 'children')],
               [Input('submit-button-water', 'n_clicks'),
               Input('submit-button-wilderness', 'n_clicks'),
               Input('submit-button-disaster', 'n_clicks'),
               Input('submit-button-reset', 'n_clicks')])
def update_dashboard(a1, a2, a3, a4):
```

Add the Cat button's identifier to the list of Inputs, and attach an argument to the update_dashboard definition (although the values passed are never used, Dash requires that the function accept the same number of args as there are inputs).

So we'd add the line: Input('submit-button-cats', 'n_clicks') to the list of Inputs (the second list argument in the callback's parentheses). Also adding another argument to update_dashboard, following what I've done here we should call this new argument 'a5', but again to be clear – the name doesn't matter, what matters is that the number of arguments match the length of the Input list in the callback annotation.

Add an elif block to the function, instructing what MongoDB query to use with the database when our cat's button ID is recognized. Following this pattern:



```
elif button_id == "submit-button-disaster":
    disaster_query = {
"breed": { "$in": ["Doberman Pinscher", "German Shepherd", "Golden Retriever", "Bloodhound", "Rottweiler"] },
"sex_upon_outcome": "Intact Male",
"age_upon_outcome_in_weeks": { "$gte": 20, "$lte": 300 },
"outcome_type": {"$nin": ["Euthanasia", "Died", "Disposal"]}
}
    label_string = "Disaster or Individual Tracking"
    filtered = db.read(disaster_query)
```

Which would look like this:

```
elif button_id == "submit-button-cats":
    cat_query = {"animal_type" : "Cat"}
    label_string = "Cat Filter Applied"
    filtered = db.read(cat_query)
```

The return value for update_dashboard is a tuple of:

1. the data used to populate the table.
2. a string to indicate to the user which filter is applied (displayed underneath buttons)

If you need to return more values, add an element to the callback annotations Output list.

And that's it. Your button will now trigger the table to show all the cats in the dataset. Explore the data and its schema in order to create more intricate/interesting queries.

Contact
Your name: Brian Baecher
brian.baecher@snhu.edu