# CS 340 README Template

## About the Grazioso Salvare Dashboard

*The purpose of the Grazioso Salvare Dashboard is to provide the company with a tool for identifying suitable candidates for search and rescue training by querying the database information of local animal shelters.*

## Motivation

*The purpose of the Dashboard is to provide a user interface for the Grazioso Salvare company to interact with information stored in a MongoDB database.*

## Getting Started

*Please refer to document CS-340ProjectOneREADME.docs for information on creating the Python CRUD module used by the GS Dashboard.*

*MongoDB was chosen for this project to capitalize on the scalability and flexibility the of SQL databases. The CRUD controller was built using Python for its powerful libraries, data visualization, and Python’s open-source Dash Framework.*

## Installation

*Jupyter Notebook:* [*https://jupyter.org/install*](https://jupyter.org/install)

*Pymongo (library)* [*https://pypi.org/project/pymongo/*](https://pypi.org/project/pymongo/)

*Byson.objectid (npm)* [*https://www.npmjs.com/package/bson-objectid*](https://www.npmjs.com/package/bson-objectid)

*Please ensure the following imports are included in for proper Dashboard setup.*

*\*Note: the animalShelter import is the CRUD Python controller file referenced in the Getting started section.*

from jupyter\_plotly\_dash import JupyterDash

import dash

import dash\_leaflet as dl

import dash\_core\_components as dcc

import dash\_html\_components as html

import plotly.express as px

import dash\_table

from dash.dependencies import Input, Output

import base64

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pymongo import MongoClient

from dash import Dash, callback\_context #This did not work AT ALL

import animalShelter

## Usage

*The GS Dashboard features an interactive data table, geolocation chart, and pie chart. The table and chart information can be filtered according to four separate queries: Reset, Water, Wilderness, and Disaster. The Water, Wilderness, and Disaster buttons filter the dataset so that only potential training candidates are displayed based on the specifications set by Grazioso Salvare.*

### Code Example

#This code portion is used to call Python CRUD module

# search() is a CRUD method

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

# Data Manipulation / Model

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

username = "aacuser"

password = "BLB019640!"

shelter = animalShelter.AnimalShelter()

# class read method must support return of cursor object and accept projection json input

df = pd.DataFrame.from\_records(shelter.search({})) # search() is a CRUD method

image\_filename = 'Grazioso Salvare Logo.png' # Graziso Salvare logo

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

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

# Dashboard Layout / View

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

app = JupyterDash('SimpleExample')

# The first section formats the company logo, URL anchor, and recognition text

app.layout = html.Div([

    html.Div(id='hidden-div', style={'display':'none'}),

    html.A([html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()),

                    style={'height':'20%','width':'20%'}),], href='https://www.snhu.edu'),

    html.Center(html.B(html.H1('Trice Bonomo SNHU CS-340 Dashboard'))),

    html.Hr(),

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

# The query buttons

        html.Button(id='submit-button-one',n\_clicks=0, children='Water'),

        html.Button(id='submit-button-two', n\_clicks=0, children='Wilderness'),

        html.Button(id='submit-button-three', n\_clicks=0, children='Disaster'),

        html.Button(id='submit-button-four', n\_clicks=0, children='Reset'),

        html.Div(id='container\_button')

    ]),

# The Dash Table layout

    dash\_table.DataTable(

        style\_data={      # wrap cell data

            'whiteSpace':'normal',

            'height':'auto',

        },

        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",

        column\_selectable=False,

        row\_selectable=False,

        row\_deletable=False,

        fixed\_rows=True,

        selected\_columns=[],

        selected\_rows=[],

        page\_action="native",

        page\_current=0,

        page\_size=10,

    ),

# Another image, URL anchor, and id tag insert

 html.Br(),

    html.Hr(),

        html.A([html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()),

                    style={'height':'20%','width':'20%'}),], href='https://www.snhu.edu'),

    html.H1('Trice Bonomo SNHU CS-340 Dashboard', style={'width':'600px'}),

# This code was sets up the layout of the geolocation map and the pie chart.

# Unfortunately, the functionality did not execute as intended. Despite adding

# the extra Div, the pie chart and geolocation map would not sit side by side.

    html.Div(className='row', children=[

    html.Div(

            id='map-id',

            className='col s6',

            ),

    html.Div([ # a drop bar was added to change the pie chart displays

        html.Label(['Analytics']),

        dcc.Dropdown(

        id='my\_dropdown',style={'width': '600px'},

# chart options are interactive and change as you tab through the data pages

        options=[

            {'label': 'Breed', 'value': 'breed'},

            {'label': 'Age (in Weeks)', 'value': 'age\_upon\_outcome\_in\_weeks'},

        ])

    ]),

    html.Div([

        dcc.Graph(id='the\_graph',style={'width': '600px', 'height': '500px'}, className='col s6',)

            ]),

                html.A([html.Img(src='data:image/png;base64,{}'.format(encoded\_image.decode()),

                    style={'height':'20%','width':'20%'}),], href='https://www.snhu.edu'),

    html.H1('Trice Bonomo SNHU CS-340 Dashboard', style={'width':'600px'}),

    ])

])

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

# Interaction Between Components / Controller

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

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

@app.callback(

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

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

#This callback handles button clicks for submit-button's one and two

#The Dash callbacks would not work properly, and I could not

#The examples on the plotly website didn’t work either, so this is an imperfect

#solution for managing the button sorting.

@app.callback(Output('datatable-id',"data"),

             [Input('submit-button-one', 'n\_clicks'),

              Input('submit-button-two','n\_clicks'),

              Input('submit-button-three','n\_clicks'),

              Input('submit-button-four','n\_clicks')

             ])

def on\_click(btn1,btn2,btn3,btn4):

    if (int(btn1)== 0 and int(btn2) == 0 and int(btn3)==0): #default display

        df = pd.DataFrame.from\_records(shelter.search({}))

    elif (int(btn1) == 1 and int(btn2) == 0 and int(btn3) == 0 and int(btn4) == 0 ):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Female","age\_upon\_outcome\_in\_weeks":{"$gte":26},

             "age\_upon\_outcome\_in\_weeks":{"$lte":156}}))) #display water

    elif (int(btn1) == 1 and int(btn2) == 1 and int(btn3) == 0 and int(btn4) == 0 ):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":26},

            "age\_upon\_outcome\_in\_weeks":{"$lte":156}}))) #display wilderness

    elif (int(btn1) == 1 and int(btn2) == 1 and int(btn3) == 1 and int(btn4) == 0 ):

        df = pd.DataFrame(list(shelter.search(

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

            "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":20},

            "age\_upon\_outcome\_in\_weeks":{"$lte":300}}))) #display disaster

    elif (int(btn1) == 1 and int(btn2) == 1 and int(btn3) == 1 and int(btn4) == 1 ):

        df = pd.DataFrame(list(shelter.search({}))) #display all

    elif (int(btn1) == 1 and int(btn2) == 1 and int(btn3) == 2 and int(btn4) == 1 ):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":20},

            "age\_upon\_outcome\_in\_weeks":{"$lte":300}}))) #display disaster

    elif (int(btn1) == 1 and int(btn2) == 2 and int(btn3) == 2 and int(btn4) == 1 ):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":26},

            "age\_upon\_outcome\_in\_weeks":{"$lte":156}}))) #display wilderness

    elif (int(btn1) == 2 and int(btn2) == 2 and int(btn3) == 2 and int(btn4) == 1 ):

        df = pd.DataFrame(list(shelter.search(

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

            "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Female","age\_upon\_outcome\_in\_weeks":{"$gte":26},

             "age\_upon\_outcome\_in\_weeks":{"$lte":156}}))) #display water

    elif (int(btn3) > int(btn1)):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Female","age\_upon\_outcome\_in\_weeks":{"$gte":26},

             "age\_upon\_outcome\_in\_weeks":{"$lte":156}}))) #display water

    elif (int(btn1) >= int(btn2)):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":26},

            "age\_upon\_outcome\_in\_weeks":{"$lte":156}}))) #display wilderness

    elif (int(btn1) >= int(btn2)):

        df = pd.DataFrame(list(shelter.search(

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

             "outcome\_type":{"$in":["Adoption","Transfer"]},"sex\_upon\_outcome":"Intact Male","age\_upon\_outcome\_in\_weeks":{"$gte":20},

            "age\_upon\_outcome\_in\_weeks":{"$lte":300}}))) #display disaster

    elif (int(btn1) >= int(btn2)):

        df = pd.DataFrame(list(shelter.search({}))) #display all

    return df.to\_dict('records')

def update\_styles(selected\_columns):

    return [{

        'if': { 'column\_id': i },

        'background\_color': '#D2F3FF'

    } for i in selected\_columns]

#callback for displaying map

@app.callback(

    Output('map-id', "children"),

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

def update\_map(viewData):

#Code for geolocation chart

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

    # Austin TX is at [30.75,-97.48]

    return [

        dl.Map(style={'width': '600px', 'height': '500px'}, center=[30.75,-97.48], zoom=10, children=[

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

            # Marker with tool tip and popup

            dl.Marker(position=[30.75,-97.48], children=[

                dl.Tooltip(dff.iloc[0,4]),

                dl.Popup([

                    html.H1("Animal Name"),

                    html.P(dff.iloc[1,15])

                ])

            ])

        ])

    ]

#callback for displaying piechart

@app.callback(

    Output(component\_id='the\_graph', component\_property='figure'),

    [Input(component\_id='my\_dropdown', component\_property='value'),

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

)

def update\_graph(my\_dropdown,viewData):

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

    piechart=px.pie(

            data\_frame=dff,

            names=my\_dropdown,

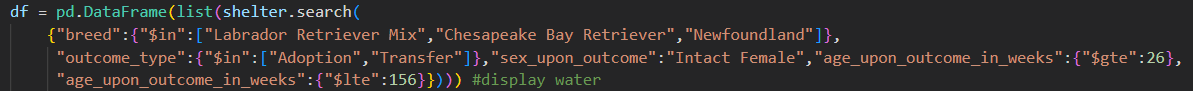
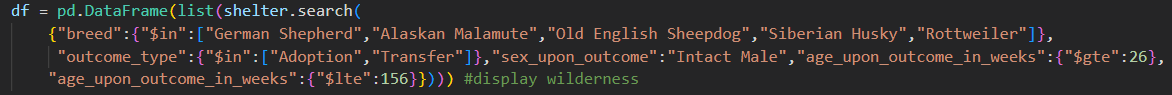
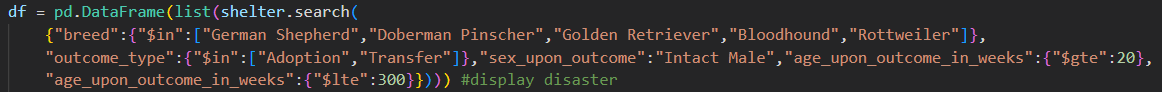
            hole=.3,

            )

    return (piechart)

app

### Screenshots

 *A closer look at the individual queries. The first line lists the different breeds specified for different rescue missions. The second and third lines specify the age range, and sexual outcome, and remove unavailable animals from the search results.*

*This query reverts the data table back to its unfiltered state.*

Map

Description automatically generated

Figure 1:Unfiltered Pie Chart using "Breed" sort function

A picture containing chart

Description automatically generated

Figure 2: Geolocation chart with Grazioso logo & credit tag.

Table

Description automatically generatedTable

Description automatically generated

Figure 4:(2/3)

Figure 3: Interactive Data table with no filter (1/3)

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generatedFigure 4: Closer view of unsorted data (3/3)

Figure 3: Wilderness filter sort. \*\*Note: This table is supposed to be empty. All potential candidates remaining after the correct breeds, genders, and ages were eliminated were pets.

Graphical user interface, text, application, email

Description automatically generatedTable

Description automatically generated

Figure 5: Disaster filter sort.

Figure 6: Water sort dataset

## Table Description automatically generated

Figure 7:Reset to unfiltered dataset

## Challenges

*This project was a nightmare to make. After weeks of working with MongoDB in a Linux shell, I was no longer stalled by issues with those tools, but Jupyter Notebook never got any better. Adding Dash framework and Plotly to the mix only made matters worse. Plotly’s button documentation is flawed. The callback\_context button example on the website did not work at all, and I was unable to figure out how to get my charts aligned at the bottom of the data table. I tried several options, but the charts didn’t follow the typical rules of HTML formatting.*

*Despite all the time and effort, I put into this project, I am not proud of it, and I find that rather demoralizing.*

## Contact

Your name: Trice Bonomo