*# Thomas Martin*  
*# CS-340-11221-M01 Client/Server Development 2024*  
*# 7-2 Project Two Submission*  
*# Southern New Hampshire University*  
*# June 20, 2024*  
  
**from** jupyter\_dash **import** JupyterDash  
  
**import** dash  
**import** dash\_leaflet **as** dl  
**from** dash **import** dash\_table  
**import** pandas **as** pd  
**from** pymongo **import** MongoClient  
**import** base64  
**from** dash **import** html   
**import** dash\_html\_components **as** html  
**from** dash **import** dcc   
**import** dash\_core\_components **as** dcc  
**from** dash.dependencies **import** Input, Output  
**import** plotly.express **as** px  *# Importing px from plotly*  
  
**from** CRUD **import** AnimalShelter  
  
*###########################*  
*# Data Manipulation / Model*  
*###########################*  
  
username = "aacuser"  
password = "SNHU1234"  
shelter = AnimalShelter(username, password)  
  
*# Dataframe*  
df = pd.DataFrame.from\_records(shelter.read({}))  
  
*# MongoDB v5+ is going to return the '\_id' column and that is going to have an*  
*# invlaid object type of 'ObjectID' - which will cause the data\_table to crash - so we remove*  
*# it in the dataframe here. The df.drop command allows us to drop the column. If we do not set*  
*# inplace=True - it will reeturn a new dataframe that does not contain the dropped column(s)*  
  
*## Debug*  
print(len(df.to\_dict(orient='records')))  
print(df.columns)  
  
**if** '\_id' **in** df.columns:  
       df.drop(columns=['\_id'], inplace=True)  
**else**:  
       print("The '\_id' column does not exist in the DataFrame.")  
  
*#########################*  
*# Dashboard Layout / View*  
*#########################*  
app = JupyterDash('\_\_name\_\_')  
  
*#image\_filename = 'Grazioso Salvare Logo.png' # replace with your own image*  
*#encoded\_image = base64.b64encode(open(image\_filename, 'rb').read()).decode*  
  
app.layout = html.Div([  
    html.Div([  
        *#html.A([html.Img(src=f'data:image/png;base64,{encoded\_image}', style={'width': '150px', 'float': 'left', 'margin-right': '10px'})], href='http://www.snhu.edu'),*  
        html.H1('Animal Shelter Dashboard', style={'textAlign': 'center'}),  
        html.H1('Thomas Martin', style={'textAlign': 'center'}),  
    ]),  
    html.Div([  
        html.Button('Water Rescue', id='water-btn', n\_clicks=0),  
        html.Button('Mountain or Wilderness Rescue', id='mountain-btn', n\_clicks=0),  
        html.Button('Disaster or Individual Tracking', id='disaster-btn', n\_clicks=0),  
        html.Button('Reset', id='reset-btn', n\_clicks=0),  
    ], style={'margin-left': '0px', 'clear': 'both'}),  
    dash\_table.DataTable(  
        id='datatable-id',  
        columns=[{"name": i, "id": i} **for** i **in** df.columns],  
        data=df.to\_dict('records'),  
        filter\_action="native",  
        sort\_action="native",  
        sort\_mode="multi",  
        page\_action="native",  
        page\_current=0,  
        page\_size=10,  
        row\_selectable='single',  
        selected\_rows=[0]  
    ),  
    html.Div([  
        html.Div(id='map-id', style={'width': '50%', 'height': '300px', 'float': 'left'}),  
        dcc.Graph(id='pie-chart', style={'width': '50%', 'height': '300px', 'float': 'right'})  
    ])  
])  
  
*# Callback for the button bar*  
@app.callback(  
    [Output('datatable-id', 'data'),  
     Output('datatable-id', 'selected\_rows'),  
     Output('pie-chart', 'figure')],  
    [Input('water-btn', 'n\_clicks'),  
     Input('mountain-btn', 'n\_clicks'),  
     Input('disaster-btn', 'n\_clicks'),  
     Input('reset-btn', 'n\_clicks')]  
)  
**def** update\_data(water\_clicks, mountain\_clicks, disaster\_clicks, reset\_clicks):  
    ctx = dash.callback\_context  
    button\_id = ctx.triggered[0]['prop\_id'].split('.')[0]  
  
    *# Filter based on the button clicked*  
    **if** button\_id == 'water-btn':  
        filtered\_df = 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** button\_id == 'mountain-btn':  
        filtered\_df = 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** button\_id == 'disaster-btn':  
        filtered\_df = 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)  
        ]  
    **else**:  
        filtered\_df = df  
  
    *# Create a pie chart*  
    pie\_df = filtered\_df['breed'].value\_counts().reset\_index()  
    pie\_df.columns = ['breed', 'count']  
    total\_count = pie\_df['count'].sum()  
    pie\_df['percentage'] = pie\_df['count'] / total\_count  
  
    *# Grouping breeds with less than 1% as "Other"*  
    other\_percentage = pie\_df[pie\_df['percentage'] < 0.01]['percentage'].sum()  
    pie\_df = pie\_df[pie\_df['percentage'] >= 0.01]  
  
    *# Using pandas.concat instead of DataFrame.append*  
    other\_row = pd.DataFrame({'breed': ['Other'], 'count': [0], 'percentage': [other\_percentage]})  
    pie\_df = pd.concat([pie\_df, other\_row], ignore\_index=True)  
  
    *# Plotting the pie chart based on the 'percentage' column*  
    figure = px.pie(pie\_df, values='percentage', names='breed', title='Distribution by Breed')  
  
    **return** filtered\_df.to\_dict('records'), [0], figure  
  
  
@app.callback(  
    Output('map-id', 'children'),  
    [Input('datatable-id', 'selected\_rows'),  
     Input('datatable-id', 'data')]  
)  
**def** update\_map(selected\_rows, data):  
    selected\_row = data[selected\_rows[0]]  
    **return** dl.Map(style={'width': '100%', 'height': '300px'}, center=[selected\_row['location\_lat'], selected\_row['location\_long']], zoom=14, children=[  
        dl.TileLayer(),  
        dl.Marker(position=[selected\_row['location\_lat'], selected\_row['location\_long']], children=[  
            dl.Tooltip(selected\_row['name']),  
            dl.Popup([  
                html.H1('Animal Name: ' + selected\_row['name']),  
                html.P('Animal Type: ' + selected\_row['animal\_type'])  
            ])  
        ])  
    ])  
  
app.run\_server(debug=True)