from jupyter\_dash import JupyterDash

from dash import dcc, html, dash\_table

from dash.dependencies import Input, Output

import dash\_leaflet as dl

import plotly.express as px

import pandas as pd

import base64

# Function to convert age string to weeks

def convert\_age\_to\_weeks(age\_str):

"""Convert age string to weeks"""

if pd.isna(age\_str):

return None

try:

num, unit = age\_str.split()

num = float(num)

unit = unit.lower()

if 'year' in unit:

return num \* 52

elif 'month' in unit:

return num \* 4.34524 # Average weeks in month

elif 'week' in unit:

return num

elif 'day' in unit:

return num / 7

return None

except:

return None

# Initialize the app.

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

# Load the data from CSV file with error handling.

try:

print("Loading data from CSV file...")

df = pd.read\_csv('aac\_shelter\_outcomes.csv')

# Ensure required columns exist.

required\_columns = [

'name', 'animal\_type', 'breed', 'age\_upon\_outcome',

'sex\_upon\_outcome', 'location\_lat', 'location\_long'

]

for col in required\_columns:

if col not in df.columns:

df[col] = None # Add missing columns with null values.

print(f"Successfully loaded {len(df)} records from CSV")

# Convert age to weeks for filtering.

if 'age\_upon\_outcome' in df.columns:

df['age\_upon\_outcome\_in\_weeks'] = df['age\_upon\_outcome'].apply(

lambda x: convert\_age\_to\_weeks(x) if pd.notna(x) else None

)

except Exception as e:

print(f"CSV loading error: {str(e)}")

print("Using empty DataFrame as fallback")

df = pd.DataFrame(columns=required\_columns)

df['age\_upon\_outcome\_in\_weeks'] = None

# Define rescue filters.

RESCUE\_FILTERS = {

'water': {

'animal\_type': 'Dog',

'breed': ['Labrador Retriever Mix', 'Chesapeake Bay Retriever', 'Newfoundland'],

'sex\_upon\_outcome': 'Intact Female',

'age\_upon\_outcome\_in\_weeks': lambda x: (x >= 26) & (x <= 156)

},

'mountain': {

'animal\_type': 'Dog',

'breed': ['German Shepherd', 'Alaskan Malamute', 'Old English Sheepdog', 'Siberian Husky', 'Rottweiler'],

'sex\_upon\_outcome': 'Intact Male',

'age\_upon\_outcome\_in\_weeks': lambda x: (x >= 26) & (x <= 156)

},

'disaster': {

'animal\_type': 'Dog',

'breed': ['Doberman Pinscher', 'German Shepherd', 'Golden Retriever', 'Bloodhound', 'Rottweiler'],

'sex\_upon\_outcome': 'Intact Male',

'age\_upon\_outcome\_in\_weeks': lambda x: (x >= 20) & (x <= 300)

}

}

# Encode logo image.

try:

logo\_path = 'Grazioso Salvare Logo.png'

logo\_encoded = base64.b64encode(open(logo\_path, 'rb').read()).decode()

except Exception as e:

print(f"Logo loading error: {str(e)}")

logo\_encoded = ""

# App layout.

app.layout = html.Div([

# Header with logo and identifier.

html.Div([

html.Img(src=f'data:image/png;base64,{logo\_encoded}',

style={'height': '100px', 'float': 'left'}),

html.H1("Animal Rescue Dashboard", style={'text-align': 'center'}),

html.Div("Created by Jamar Sampson",

style={'text-align': 'right', 'font-style': 'italic'})

], style={'margin-bottom': '20px'}),

# Filter controls.

dcc.RadioItems(

id='rescue-filter',

options=[

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

{'label': 'Mountain Rescue', 'value': 'mountain'},

{'label': 'Disaster Rescue', 'value': 'disaster'},

{'label': 'All Animals', 'value': 'all'}

],

value='all',

labelStyle={'display': 'inline-block', 'margin-right': '10px'}

),

# Data table with enhanced features.

dash\_table.DataTable(

id='data-table',

columns=[{"name": col, "id": col} for col in df.columns],

data=df.to\_dict('records'),

page\_size=10,

page\_action='native',

filter\_action="native",

sort\_action="native",

sort\_mode="multi",

style\_table={'overflowX': 'auto'},

style\_header={

'backgroundColor': 'rgb(230, 230, 230)',

'fontWeight': 'bold'

},

style\_cell={

'textAlign': 'left',

'minWidth': '100px', 'width': '150px', 'maxWidth': '180px',

'whiteSpace': 'normal'

}

),

# Charts and map.

html.Div([

dcc.Graph(id='breed-chart'),

dl.Map(

id='map',

style={'height': '400px', 'margin-top': '20px'},

center=[30.2672, -97.7431],

zoom=10,

children=[dl.TileLayer()]

)

])

])

# Callbacks.

@app.callback(

Output('data-table', 'data'),

Input('rescue-filter', 'value')

)

def update\_table(filter\_value):

if filter\_value == 'all':

return df.to\_dict('records')

# Apply filters for CSV data.

filtered\_df = df.copy()

filters = RESCUE\_FILTERS[filter\_value]

# Apply each filter condition.

for col, condition in filters.items():

if col == 'breed':

filtered\_df = filtered\_df[filtered\_df[col].isin(condition)]

elif col == 'age\_upon\_outcome\_in\_weeks':

filtered\_df = filtered\_df[condition(filtered\_df[col])]

else:

filtered\_df = filtered\_df[filtered\_df[col] == condition]

return filtered\_df.to\_dict('records')

@app.callback(

Output('breed-chart', 'figure'),

Input('data-table', 'derived\_virtual\_data')

)

def update\_chart(data):

if not data or len(data) == 0:

return px.pie(title="No data available")

breed\_counts = pd.DataFrame(data)['breed'].value\_counts().head(10).reset\_index()

return px.pie(breed\_counts, names='index', values='breed', title='Top 10 Breeds')

@app.callback(

Output('map', 'children'),

Input('data-table', 'derived\_virtual\_data')

)

def update\_map(data):

if not data:

return [dl.TileLayer()]

df\_map = pd.DataFrame(data)

markers = [

dl.Marker(

position=[row['location\_lat'], row['location\_long']],

children=[dl.Tooltip(row['name'])]

)

for \_, row in df\_map.iterrows()

if pd.notna(row.get('location\_lat')) and pd.notna(row.get('location\_long'))

]

return [dl.TileLayer()] + markers

app.run\_server(mode='inline')