import dash

from dash import dcc, html, callback, Input, Output

import dash\_bootstrap\_components as dbc

import pandas as pd

import socketio

import plotly.express as px

from dash.dependencies import Input, Output

import plotly.express as px

import socketio

import threading

import time

import json

import boto3

def list\_all\_objects(session, bucket\_name, prefix=''):

"""List all objects in an S3 bucket with a given prefix."""

s3 = session.client('s3')

paginator = s3.get\_paginator('list\_objects\_v2')

all\_files = []

try:

for page in paginator.paginate(Bucket=bucket\_name, Prefix=prefix):

for obj in page.get('Contents', []):

all\_files.append(obj['Key'])

except Exception as e:

print(f"Error listing objects in bucket {bucket\_name}: {e}")

return all\_files

def process\_json\_file(session, bucket\_name, file\_key):

"""Process a single JSON file from S3."""

s3 = session.client('s3')

try:

file\_response = s3.get\_object(Bucket=bucket\_name, Key=file\_key)

content = file\_response['Body'].read().decode('utf-8')

return json.loads(content)

except json.JSONDecodeError as e:

print(f"Error decoding JSON in file {file\_key}: {e}")

except Exception as e:

print(f"Error fetching file {file\_key} from bucket {bucket\_name}: {e}")

return []

def load\_initial\_s3\_data():

"""Load and process all JSON files from the S3 bucket."""

start\_time = time.time()

session = boto3.Session()

bucket\_name = 'trv-kinesis-fnol-data-sink-claim-dc3-dev-us-east-1-umtotr'

folder\_key = '2024/cat\_sample\_data.json'

# List all files

all\_files = list\_all\_objects(session, bucket\_name, prefix=folder\_key)

all\_json\_list = []

# Process each file

for file\_key in all\_files:

if file\_key:

json\_list = process\_json\_file(session, bucket\_name, file\_key)

all\_json\_list.extend(json\_list)

if all\_json\_list:

# Convert to DataFrame

df = pd.DataFrame(all\_json\_list)

# Preprocess data

records = []

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

try:

value\_data = json.loads(row['value'])

value\_data.update(row.to\_dict())

records.append(value\_data)

except json.JSONDecodeError as e:

print(f"Error decoding nested JSON in row: {e}")

processed\_df = pd.DataFrame(records)

print(f"Data fetched and processed in {time.time() - start\_time:.2f} seconds")

return processed\_df

else:

print("No valid JSON data found.")

return pd.DataFrame()

# Global variable to cache data for initial load

cached\_data = load\_initial\_s3\_data()

# Create a Socket.IO client

sio = socketio.Client()

@sio.event

def connect():

print('Connected to WebSocket server')

@sio.event

def disconnect():

print('Disconnected from WebSocket server')

@sio.event

def connect\_error(data):

print('Connection error:', data)

@sio.on('new\_data')

def on\_new\_data(data):

global cached\_data

print('Received new data:')

# Write data to DataFrame

try:

df = pd.DataFrame(data['data'])

if not cached\_data.empty:

cached\_data = pd.concat([cached\_data, df], ignore\_index=True)

else:

cached\_data = df

except Exception as e:

print(f"Error processing received data: {e}")

def start\_websocket\_client():

sio.connect('http://localhost:8060') # Replace with your WebSocket server address

try:

while True:

time.sleep(1) # Keep the thread alive

except KeyboardInterrupt:

print('WebSocket client stopped')

sio.disconnect()

# Start WebSocket client in a separate thread

threading.Thread(target=start\_websocket\_client, daemon=True).start()

# Initialize the Dash app

app = dash.Dash(\_\_name\_\_, external\_stylesheets=[dbc.themes.SOLAR])

# App Layout with Tabs

app.layout = dbc.Container([

html.H1("Cat Claims Data Dashboard", style={'text-align': 'center', 'margin-bottom': '50px'}),

# Interval component for automatic refresh (every 10 seconds)

dcc.Interval(id='interval-component', interval=10\*1000, n\_intervals=0),

dbc.Row([

dbc.Col(html.Div([

html.H4("Total Distinct Claimants", style={'color': '#FF6347'}),

html.H2(id="distinct-claimants", style={'font-size': '36px', 'color': '#2E8B57'})

]), width=3),

dbc.Col(html.Div([

html.H4("Total High Risk Claims", style={'color': '#FFD700'}),

html.H2(id="high-risk-claims", style={'font-size': '36px', 'color': '#2E8B57'})

]), width=3),

dbc.Col(html.Div([

html.H4("Total Amount Claimed", style={'color': '#1E90FF'}),

html.H2(id="total-claim-amount", style={'font-size': '36px', 'color': '#2E8B57'})

]), width=3),

dbc.Col(html.Div([

html.H4("Appraisal Completion %", style={'color': '#32CD32'}),

html.H2(id="appraisal-completion", style={'font-size': '36px', 'color': '#2E8B57'})

]), width=3)

]),

html.Hr(),

# Tabs for different charts

dbc.Tabs([

dbc.Tab(label='High Risk Claims by State', tab\_id='tab-1', children=[

dcc.Graph(id='high-risk-claims-by-state', style={'height': '400px'})

]),

dbc.Tab(label='Appraisal Requests', tab\_id='tab-2', children=[

dcc.Graph(id='appraisal-request-completion', style={'height': '400px'})

]),

dbc.Tab(label='Claim Count Over Time', tab\_id='tab-3', children=[

dcc.Graph(id='claim-count-over-time', style={'height': '400px'})

]),

dbc.Tab(label='Total Claim Amount Paid', tab\_id='tab-4', children=[

dcc.Graph(id='total-claim-amount-paid', style={'height': '400px'})

]),

dbc.Tab(label='Claims Per Accident State', tab\_id='tab-5', children=[

dcc.Graph(id='claims-per-accident-state', style={'height': '400px'})

]),

dbc.Tab(label='Inspection Types', tab\_id='tab-6', children=[

dcc.Graph(id='inspection-types', style={'height': '400px'})

]),

dbc.Tab(label='Vehicle Total Loss Percentage', tab\_id='tab-7', children=[

dcc.Graph(id='vehicle-total-loss-percentage', style={'height': '400px'})

])

], id='tabs', active\_tab='tab-1'),

], fluid=True)

# Callbacks

@app.callback(

Output('distinct-claimants', 'children'),

Output('high-risk-claims', 'children'),

Output('total-claim-amount', 'children'),

Output('appraisal-completion', 'children'),

Output('claim-count-over-time', 'figure'),

Output('high-risk-claims-by-state', 'figure'),

Output('appraisal-request-completion', 'figure'),

Output('total-claim-amount-paid', 'figure'),

Output('claims-per-accident-state', 'figure'),

Output('inspection-types', 'figure'),

Output('vehicle-total-loss-percentage', 'figure'),

[Input('tabs', 'active\_tab'),

Input('interval-component', 'n\_intervals')]

)

def update\_dashboard(active\_tab,n\_intervals):

global cached\_data

df = cached\_data.copy()

# KPI Calculations

distinct\_claimants = df[df['KPI'] == 'TUMBLE\_COUNT\_DISTINCT\_CLAIMANT']['cnt\_distinct\_clmnt'].sum(skipna=True)

high\_risk\_claims = df[df['KPI'] == 'HIGH\_RISK\_CLAIM\_COUNT']['risk\_clm\_cnt'].sum(skipna=True)

total\_claim\_amount = df[df['KPI'] == 'SUM\_TOTAL\_CLM\_AMT\_PAID']['sum\_tot\_clm\_amt\_paid'].sum(skipna=True)

appraisal\_completion = df[df['KPI'] == 'APPRAISAL\_COMPLETE\_PERCENT']['appraisal\_completion\_percentage'].mean(skipna=True)

# \*\*Claim Count Over Time (Enhanced Visualization)\*\*

df\_claims = df[df['KPI'] == 'TUMBLE\_COUNT\_DISTINCT\_CLAIM']

claim\_count\_fig = px.density\_heatmap(

df\_claims,

x="window\_start",

y="cnt\_distinct\_clm",

title="Claim Count Over Time",

nbinsx=30, nbinsy=20,

color\_continuous\_scale='Viridis', # Adding a vibrant color scale

labels={'cnt\_distinct\_clm': 'Distinct Claims', 'window\_start': 'Time'},

)

# Update layout for better presentation

claim\_count\_fig.update\_layout(

title\_font=dict(size=24, color='#FF6347', family="Arial"), # Bold title

xaxis\_title="Time Window",

yaxis\_title="Claim Count",

font=dict(family="Arial", size=14),

paper\_bgcolor="#FFFFFF",

plot\_bgcolor="#E5ECF6",

coloraxis\_colorbar=dict(

title="Claim Count",

tickvals=[0, 50, 100, 150, 200],

ticks="outside",

tickfont=dict(size=12),

)

)

# High Risk Claims by State

df\_risk\_state = df[df['KPI'] == 'HIGH\_RISK\_CLAIM\_COUNT'].groupby('state').sum(numeric\_only=True).reset\_index()

geo\_fig = px.choropleth(df\_risk\_state,

locations='state',

locationmode="USA-states",

color='risk\_clm\_cnt',

scope="usa",

title="High Risk Claims by State")

# Appraisal Requests Over Time

df\_appraisal = df[df['KPI'] == 'APPRAISAL\_REQUEST\_COUNT']

# Aggregate appraisal requests by time window

df\_aggregated\_appraisal = df\_appraisal.groupby('window\_start', as\_index=False).agg({

'appraisal\_request\_count': 'sum' # Aggregate count per time window

})

# Create an enhanced area chart for better impact

appraisal\_fig = px.area(

df\_aggregated\_appraisal,

x="window\_start",

y="appraisal\_request\_count",

title="Appraisal Requests Over Time (Enhanced Visualization)",

labels={

'appraisal\_request\_count': 'Appraisal Requests',

'window\_start': 'Time'

},

color\_discrete\_sequence=['#3498DB'] # Vibrant blue color for area fill

)

# Update layout for a polished presentation

appraisal\_fig.update\_layout(

title\_font=dict(size=26, color='#1ABC9C', family="Arial Black"), # Bold title

xaxis\_title="Time Window",

yaxis\_title="Appraisal Requests",

font=dict(family="Calibri", size=14),

paper\_bgcolor="#FFFFFF", # Clean white background

plot\_bgcolor="#F4F6F7", # Subtle light gray plot background

margin=dict(l=50, r=50, t=70, b=50), # Balanced margins

)

# Total Claim Amount Paid Over Time

df\_total\_paid = df[df['KPI'] == 'SUM\_TOTAL\_CLM\_AMT\_PAID']

## Create an area chart for total claim amount paid

total\_claim\_amount\_fig = px.area(

df\_total\_paid,

x="window\_start",

y="sum\_tot\_clm\_amt\_paid",

title="Total Claim Amount Paid Over Time (Area Chart)",

labels={'sum\_tot\_clm\_amt\_paid': 'Total Claim Amount Paid', 'window\_start': 'Time'},

color\_discrete\_sequence=['#FF7F50'] # Coral color for area fill

)

# Update layout for better presentation

total\_claim\_amount\_fig.update\_layout(

title\_font=dict(size=28, color='#1E90FF', family="Arial Black"), # Vibrant blue title with bold font

xaxis\_title="Time Window",

yaxis\_title="Total Claim Amount Paid ($)",

font=dict(family="Calibri", size=16, color='#2F4F4F'), # Elegant font and color for text

paper\_bgcolor="#FFFFFF", # Clean white background for professional look

plot\_bgcolor="#E8F6F3", # Soft mint-green plot area for subtle contrast

showlegend=False, # Legend not necessary for an area chart

margin=dict(l=50, r=50, t=70, b=50), # Balanced margins

)

# Claims Per Accident State

df\_per\_accdnt\_st = df[df['KPI'] == 'CNT\_PER\_ACCDNT\_ST']

df\_aggregated = df\_per\_accdnt\_st.groupby('r\_accident\_state', as\_index=False).agg({

'cnt\_per\_accdnt\_st': 'sum' # Sum claims per state

})

# Create an enhanced bar chart

claims\_per\_accdnt\_st\_fig = px.bar(

df\_aggregated,

x="r\_accident\_state",

y="cnt\_per\_accdnt\_st",

title="Claims Per Accident State (Bar Chart)",

labels={

'r\_accident\_state': 'Accident State',

'cnt\_per\_accdnt\_st': 'Claim Count'

},

color\_discrete\_sequence=px.colors.sequential.Blues\_r # Gradient blue for a clean, professional look

)

# Update layout for a polished look

claims\_per\_accdnt\_st\_fig.update\_layout(

title\_font=dict(size=26, color='#2E86C1', family="Arial Black"), # Vibrant blue bold title

xaxis\_title="Accident State",

yaxis\_title="Number of Claims",

font=dict(family="Calibri", size=14),

paper\_bgcolor="#FFFFFF", # Clean white background

plot\_bgcolor="#F4F6F7", # Subtle light gray plot background

margin=dict(l=50, r=50, t=70, b=50), # Balanced margins

)

# Inspection Types

df\_inspct\_typ = df[df['KPI'] == 'CNT\_INSPCT\_TYP']

inspection\_types\_fig = px.sunburst(

df\_inspct\_typ,

path=["Inspect\_Type"],

values="cnt\_inspct\_typ",

title="Distribution of Inspection Types",

color="cnt\_inspct\_typ",

color\_continuous\_scale=px.colors.sequential.Viridis,

)

# Enhance layout for clarity and appeal

inspection\_types\_fig.update\_layout(

title\_font=dict(size=22, color="#FF4500", family="Arial Black"),

font=dict(family="Courier New", size=14, color="#000000"),

paper\_bgcolor="#F9F9F9",

margin=dict(l=50, r=50, t=50, b=50),

sunburstcolorway=["#FF5733", "#33FFBD", "#FFC300", "#DAF7A6", "#FF33EC", "#339FFF"], # Custom vibrant colors

)

# Optional: Adding hover details

inspection\_types\_fig.update\_traces(

hovertemplate="<b>Inspection Type:</b> %{label}<br><b>Count:</b> %{value}<extra></extra>"

)

# Vehicle Total Loss Percentage

df\_loss = df[df['KPI'] == 'VEHICLE\_TOTAL\_LOSS\_PERCENT']

# Ensure no NaN or zero values for 'vehicle\_total\_loss\_percentage'

df\_loss = df\_loss[df\_loss['vehicle\_total\_loss\_percentage'].notna()] # Remove rows with NaN

df\_loss = df\_loss[df\_loss['vehicle\_total\_loss\_percentage'] > 0] # Remove rows with zero or negative values

# Here I'm assuming you want to aggregate by 'window\_start'

df\_loss\_aggregated = df\_loss.groupby("window\_start")["vehicle\_total\_loss\_percentage"].sum().reset\_index()

vehicle\_loss\_fig = px.sunburst(

df\_loss\_aggregated,

path=["window\_start"],

values="vehicle\_total\_loss\_percentage",

title="Vehicle Total Loss Percentage Over Time (Sunburst Chart)",

color="vehicle\_total\_loss\_percentage",

color\_continuous\_scale="Viridis", # Vibrant color scale for better contrast

hover\_data=["window\_start", "vehicle\_total\_loss\_percentage"], # Display more info on hover

)

vehicle\_loss\_fig.update\_layout(

title\_font=dict(size=26, color='#1D3557', family="Arial, sans-serif" ), # Larger, bold title

paper\_bgcolor="#F1FAEE",

plot\_bgcolor="#E5E5E5",

font=dict(family="Arial", size=14),

margin=dict(t=50, b=50, l=50, r=50), # Add margins for better layout

hoverlabel=dict(bgcolor="white", font\_size=12, font\_family="Arial"), # Better hover label style

coloraxis\_colorbar=dict(

title="Loss Percentage",

tickvals=[0, 5, 10, 15, 20], # Custom color scale ticks

ticks="outside",

ticklen=5,

tickcolor="#1D3557"

),

)

return (

f"{distinct\_claimants:,}",

f"{high\_risk\_claims:,}",

f"${total\_claim\_amount:,.2f}",

f"{appraisal\_completion:.2f}%",

claim\_count\_fig,

geo\_fig,

appraisal\_fig,

total\_claim\_amount\_fig,

claims\_per\_accdnt\_st\_fig,

inspection\_types\_fig,

#indemnity\_payment\_fig,

vehicle\_loss\_fig

)

if \_\_name\_\_ == '\_\_main\_\_':

app.run\_server(host='0.0.0.0', port=8050)