BIG DATA ANALYTICS EXAM BY NAVTACC

AAMNA NAZ AWAN

PRACTICAL TASK: CREATE A DASH APP WITH CALBACK FUNCTIONS TO HANDLE REAL TIME DATA UPDATES

OVERVIEW OF THIS APP:-

This Dash app provides an interactive platform for real-time data visualization, allowing users to explore three types of dynamically generated data: random, sinusoidal, and exponential. Key features include:

- **Real-Time Updates:** Graph refreshes every second with live data.
- **Dynamic Y-Axis Limits:** Adjusts based on selected data type.
- **Download Functionality:** Users can download the graph as a PNG image.
- **Responsive Design:** Utilizes Dash Bootstrap components for a polished layout.
- **User Interaction:** Dropdown menu for selecting different data types.

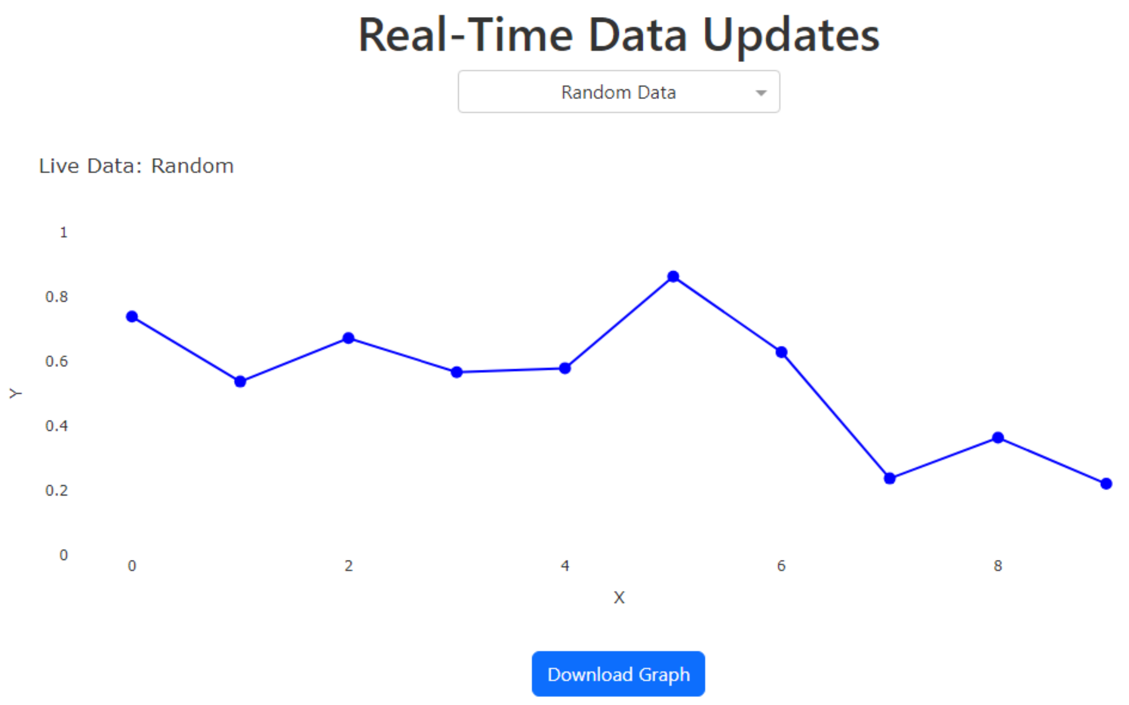
HERE IS MY CODE:-

```python
import dash
from dash import dcc, html, Input, Output, State
import plotly.graph_objs as go
import numpy as np
import base64
from dash.dependencies import Input, Output
import dash_bootstrap_components as dbc

# Create a Dash app with Bootstrap
app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])

# Layout of the app with Bootstrap components
app.layout = dbc.Container(
    style={'textAlign': 'center', 'padding': '30px'},
    children=[
        html.H1("Real-Time Data Updates", style={'color': '#333'}),
        dcc.Dropdown(
            id='data-type-dropdown',
            options=[
                {'label': 'Random Data', 'value': 'random'},
                {'label': 'Sinusoidal', 'value': 'sinusoidal'},
                {'label': 'Exponential', 'value': 'exponential'}
            ],
            value='random',
            clearable=False,
            style={'width': '50%', 'margin': 'auto'}
        ),
        dcc.Graph(id='live-update-graph'),
        dbc.Button("Download Graph", id="download-button", color="primary",
className="mb-3"),
        dcc.Interval(
            id='interval-component',
            interval=1 * 1000,  # in milliseconds
            n_intervals=0
        ),
        dcc.Download(id="download")
    ]
)

# Callback to update the graph based on the selected data type
@app.callback(
    Output('live-update-graph', 'figure'),
    Input('interval-component', 'n_intervals'),
    Input('data-type-dropdown', 'value')
)
def update_graph(n, data_type):
    x = np.arange(10)

    # Generate data based on the selected type
    if data_type == 'random':
        y = np.random.rand(10)
        y_range = [0, 1]
    elif data_type == 'sinusoidal':
        y = np.sin(x + n / 10)
        y_range = [-1, 1]
    else:  # exponential
        y = np.exp(x / 5) + np.random.rand(10) * 0.5  # Adding some noise
        y_range = [0, np.max(y) * 1.1]

    # Create the figure
    figure = go.Figure()
    figure.add_trace(go.Scatter(x=x, y=y, mode='lines+markers',
name=data_type.capitalize(),
                                 marker=dict(size=10, color='blue' if
data_type == 'random' else ('orange' if data_type == 'sinusoidal' else
'green'))))
    figure.update_layout(
        title=f'Live Data: {data_type.capitalize()}',
        xaxis_title='X',
        yaxis_title='Y',
        yaxis=dict(range=y_range),
        plot_bgcolor='#ffffff',
        font=dict(color='#333'),
        template='plotly'
    )

    return figure

# Callback to download the graph as an image
@app.callback(
    Output("download", "data"),
    Input("download-button", "n_clicks"),
    State('live-update-graph', 'figure'),
    prevent_initial_call=True,
)
def download_graph(n_clicks, figure):
    # Encode the figure as a PNG image
    img_bytes = figure.to_image(format="png")
    encoded_img = base64.b64encode(img_bytes).decode()
    return dict(content=encoded_img, filename="graph.png")

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

MY APP URL:-

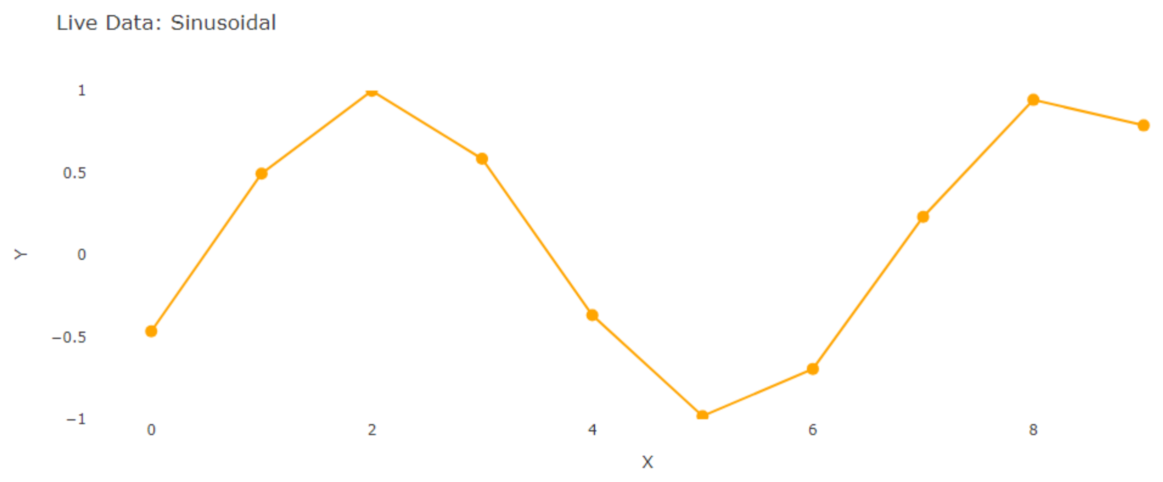
http://127.0.0.1:8050/

DEMO OF APP:-



Real-Time Data Updates

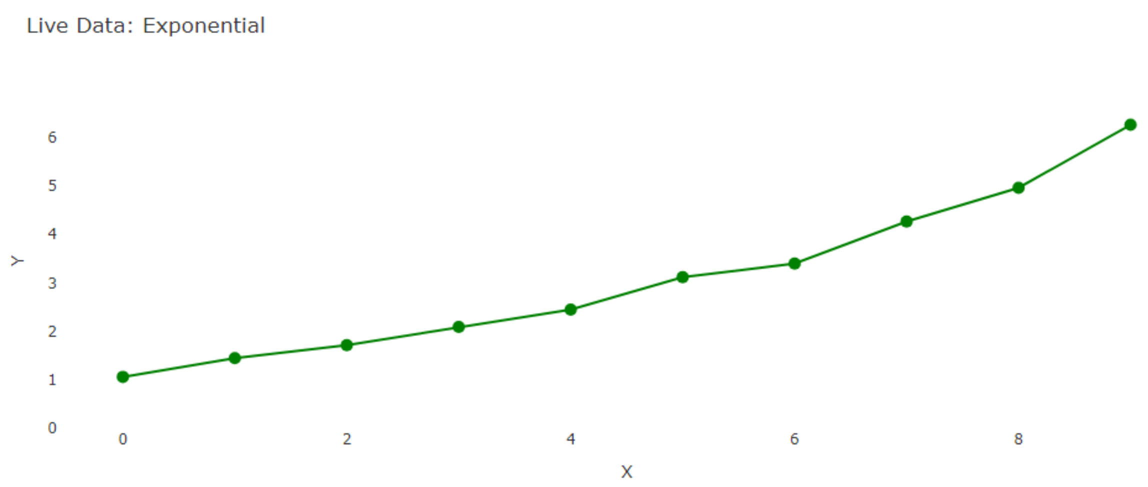
Sinusoidal



Download Graph

Real-Time Data Updates

Exponential



Download Graph