Cloudisian
TRANSFORMING IDEAS TO IMPACT

Project Summary: Real-Time Sensor Data Dashboard with Dash & Kafka

---

📊 **Overview:** This project presents a real-time dashboard for visualizing live sensor data using the Dash framework (Python) and Plotly graphs. The dashboard is capable of fetching data from a mock generator or a Kafka stream and dynamically updating various charts and alerts in real time.

🛠️ **Tech Stack:**

- Dash (for frontend dashboard interface)
- Plotly (for visualizations)
- Kafka (real-time data stream via kafka-python)
- Pandas (for data handling)
- Deque (for in-memory buffering)
- Random, datetime (for mock data simulation)

📈 **Dashboard Components:**

1. **Live Line Chart:**
    - Tracks temperature over time.
    - Updates every 1 second (1000 ms).
2. **Gauge Chart:**
    - Displays current temperature as a speedometer-style gauge.
3. **Histogram:**
    - Shows distribution of temperature values (bin size: 10).
4. **Scatter Plot:**
    - Plots temperature vs. humidity.
5. **Bar Chart:**
    - Shows pressure values over time.
6. **Heatmap:**
    - Displays a matrix of humidity and temperature over time.
7. **Text Alerts:**
    - Displays warning if temperature exceeds 30°C.

- Shows “System Normal” when values are within safe range.

🔄 Real-Time Update Logic:

- Data updated using `dcc.Interval` every 1000 milliseconds.
- Callback function `update_graph()` handles new data generation and graph refreshes.
- Uses deque for storing the latest 100 sensor readings.

📝 Data Simulation:

- Function `generate_mock_data()` simulates live sensor values:
  - Temperature (18–32 °C)
  - Humidity (30–80%)
  - Pressure (980–1020 hPa)
  - Vibration (0–10 units)

📢 Kafka Integration (Optional):

- Kafka consumer function `get_kafka_data()` included (commented out).
- Ready to plug in for production environments by replacing mock generator.

⚠️ Alert Logic:

- If temperature > 30°C, display alert: **“High Temperature”** (in red).
- Else, display: **“System Normal”** (in green).

📂 Configuration:

- `BUFFER_SIZE = 100`
- `UPDATE_INTERVAL = 1000 ms`
- Multiple visualizations organized in a responsive flex layout.

✅ Bug Fixes & Notes:

- `_name_ == '_main_'` corrected to `__name__ == '__main__'`
- Fixed `returnz` typo in mock function.
- Heatmap z-axis formatted correctly as a list of lists.

🚀 **Run Instructions:**

```
pip install dash pandas plotly kafka-python
python your_script.py
```

Conclusion: This dashboard is suitable for IoT-based sensor monitoring and real-time analytics. It is easily extendable for production Kafka pipelines and supports rich visualization for various sensor metrics.

Source code

```
# Install required libraries

# pip install dash pandas plotly kafka-python


import dash

from dash import dcc, html

from dash.dependencies import Input, Output

import plotly.graph_objs as go

import pandas as pd

import random

from datetime import datetime

from collections import deque

# The import below now works because you installed the correct 'kafka-python' library

from kafka import KafkaConsumer  # For real Kafka integration


# Initialize Dash app

app = dash.Dash(__name__)
```

```
# Configuration
BUFFER_SIZE = 100  # Data points to keep in memory
UPDATE_INTERVAL = 1000  # Dashboard update interval in ms

# Mock data buffer (Replace with Kafka consumer in production)
data_buffer = deque(maxlen=BUFFER_SIZE)

# Initialize dashboard layout
app.layout = html.Div([
  html.H1("Real-Time Sensor Data Dashboard"),

  # Main line graph and update interval
  html.Div([
    dcc.Graph(id='live-graph', animate=True),
    dcc.Interval(id='graph-update', interval=UPDATE_INTERVAL),
  ]),

  # Grid for other visualizations
  html.Div([
    html.Div([
      dcc.Graph(id='gauge-1'),
      dcc.Graph(id='histogram-1'),
    ], style={'display': 'flex'}), # Using flex for basic alignment

    html.Div([
      dcc.Graph(id='scatter-1'),
```

```
        dcc.Graph(id='bar-1'),
    ], style={'display': 'flex'}),

    html.Div([
        dcc.Graph(id='heatmap-1'),
        html.Div(id='text-alerts', style={'padding': '20px', 'fontSize': '24px'}),
    ], style={'display': 'flex'})
  ])
])

# Mock data generator
def generate_mock_data():
  # FIX: Changed 'returnz' to 'return'
  return {
    'timestamp': datetime.now(),
    'temperature': random.uniform(18, 32),
    'humidity': random.uniform(30, 80),
    'pressure': random.uniform(980, 1020),
    'vibration': random.uniform(0, 10)
  }

# Kafka consumer example (uncomment to use)
# def get_kafka_data():
#   consumer = KafkaConsumer('sensor-topic',
#                bootstrap_servers=['localhost:9092'])
#   for message in consumer:
#     yield message.value
```

```
@app.callback(
   [Output('live-graph', 'figure'),
    Output('gauge-1', 'figure'),
    Output('histogram-1', 'figure'),
    Output('scatter-1', 'figure'),
    Output('bar-1', 'figure'),
    Output('heatmap-1', 'figure'),
    Output('text-alerts', 'children')],
   [Input('graph-update', 'n_intervals')]
)
def update_graph(n):
   # Generate/get new data
   new_data = generate_mock_data()
   data_buffer.append(new_data)

   # Create DataFrame from buffer
   df = pd.DataFrame(data_buffer)

   # --- Create Figures ---

   # 1. Main line graph
   live_graph_fig = {
      'data': [go.Scatter(
         x=df['timestamp'],
         y=df['temperature'],
         name='Temperature',
```

```
        mode='lines+markers'
    )],
    'layout': go.Layout(title='Temperature Over Time', uirevision='true')
}

# 2. Gauge chart
gauge_fig = {
    'data': [go.Indicator(
        mode="gauge+number",
        value=df['temperature'].iloc[-1],
        title={'text': "Current Temp (°C)"},
        gauge={'axis': {'range': [18, 32]}}
    )],
    'layout': go.Layout(title='Temperature Gauge')
}

# 3. Histogram
hist_fig = {
    'data': [go.Histogram(x=df['temperature'], nbinsx=10, name='Temp')],
    'layout': go.Layout(title='Temperature Distribution')
}

# 4. Scatter plot
scatter_fig = {
    'data': [go.Scatter(
        x=df['temperature'],
        y=df['humidity'],
```

```
        mode='markers',
        name='Data points'
      )],
      'layout': go.Layout(title='Temp vs Humidity', xaxis_title='Temperature',
yaxis_title='Humidity')
    }

    # 5. Bar chart
    bar_fig = {
      'data': [go.Bar(
        x=[d.strftime('%H:%M:%S') for d in df['timestamp']],
        y=df['pressure'],
        name='Pressure'
      )],
      'layout': go.Layout(title='Pressure Readings')
    }

    # 6. Heatmap
    heatmap_fig = {
      'data': [go.Heatmap(
        x=[d.strftime('%H:%M:%S') for d in df['timestamp']],
        y=['Humidity', 'Temp'],
        z=[df['humidity'].tolist(), df['temperature'].tolist()], # Ensure z is a list of lists
        colorscale='Viridis'
      )],
      'layout': go.Layout(title='Sensor Heatmap')
    }
```

```
    # Alert system
    alerts = []
    if df['temperature'].iloc[-1] > 30:
        alerts.append(html.P("ALERT: High Temperature!", style={'color': 'red', 'fontWeight': 'bold'}))
    else:
        alerts.append(html.P("System Normal", style={'color': 'green'}))

    return live_graph_fig, gauge_fig, hist_fig, scatter_fig, bar_fig, heatmap_fig, alerts


if __name__ == '__main__':
    app.run(debug=True)
```