Student name: Zakarya Guerinat

Student ID:217090531

SIT225: Data Capture Technologies

Activity 6.1: Plotly data dashboard

Plotly Dash apps give a point-&-click interface to models written in Python, vastly expanding the notion of what's possible in a traditional "dashboard". With Dash apps, data scientists and engineers put complex Python analytics in the hands of business decision-makers and operators. In this activity, you will learn basic building blocks of Plotly to create Dash apps.

Hardware Required

No hardware is required.

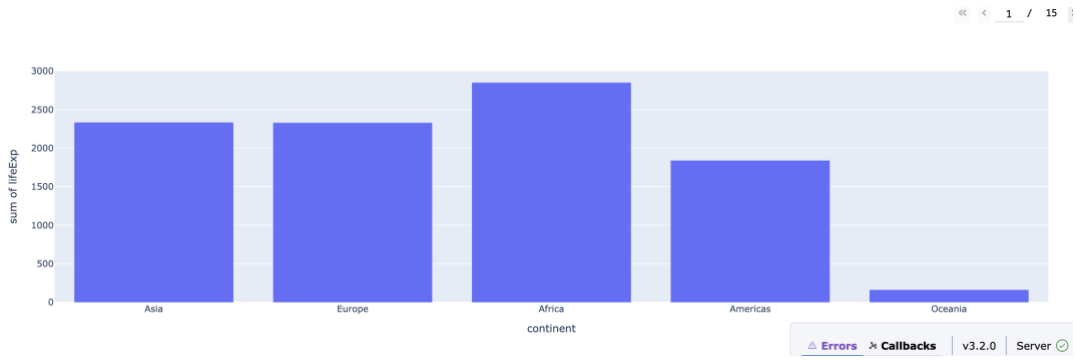
Software Required

Plotly library and Dash module
Python 3

Steps

Step	Action
1	Install Plotly and dash using the command below in the command line. $ pip install plotly dash You can download Jupyter Notebook from here (https://github.com/deakin-deep-dreamer/sit225/blob/main/week_6/plotly_explore.ipynb) and run all the cells. The Notebook contains multiple sections such as Hello World which follows a sample code in a following cell. If you run the Hello world cell it will show Plotly Dash web page. The cell also includes a Question (#*** Question) which you will need to carry out to get a modified output. You will need to capture the output and share the screenshot in the following steps.

2	Question: **Hello World** cell has a question - add another html.Div to show your name, and re-run the cell for output. You will need to update the code, run the cell, capture the screenshot of the output and paste it here. Hello World Zakarya Guerinat 217090531 SIT225 Week 6 Plotly and Dash Answer:
3	Question: **Connecting to Data** cell has a question - change page size and observe the change in widget controls such as, total number of pages. You will need to update the code, run the cell, capture the screenshot of the output and paste it here. Answer: My First App with Data country pop continent lifeExp gdpPercap Afghanistan 31889923 Asia 43.828 974.5803384 Albania 3600523 Europe 76.423 5937.029525999999 Algeria 33333216 Africa 72.301 6223.367465 Angola 12420476 Africa 42.731 4797.231267 Argentina 40301927 Americas 75.32 12779.37964 « ‹ 1 / 29 › »

4	Question: **Visualising data** cell has a question - explore another histfunc other than 'avg' used above and observe behaviour. You will need to update the code, run the cell, capture the screenshot of the output and paste it here. Answer: <Your answer>
5	Question: **Controls and Callbacks** cell has a question - use line graphs instead of histogram. You will need to update the code, run the cell, capture the screenshot of the output and paste it here. Answer: 
6	Question: Now you have learned how to use Plotly Dash for visualising your data, describe how you will be using this tool for your desired sensor monitoring dashboard with a number of sensors including DHT22 or accelerometer data. Answer: I will use plotly dash to build a dashboard that shows data from a dht22 sensor , I will show show the change in temperature and humidity, the dashboard will have charts to see how the readings change over time and tables to quickly check the latest values.
7	Question: Convert the Notebook to PDF and merge with this activity sheet PDF. You will need this merged PDF to combine with this week's OnTrack task for submission. Answer: <Your answer>

Week 6 Ontrack Task

```python
# app.py
# Plotly Dash Gyroscope Explorer
# - Reads ./gyro_samples_clean.csv from your Week 6 project root
# - Chart type dropdown: Line, Scatter, Histogram, Box (basic + distribution)
# - Axis selector: choose X, Y, Z (any or all)
# - Sample window: enter N samples, page with Prev/Next
# - Summary table updates whenever the graph updates

import os
import pandas as pd
import numpy as np
from dash import Dash, dcc, html, dash_table, Input, Output, State, ctx
import plotly.graph_objects as go
import plotly.express as px

CSV_PATH = "./gyro_samples_clean.csv"  # keep the CSV in the same folder as app.py

# ---------- Load & prepare data ----------
if not os.path.exists(CSV_PATH):
    raise FileNotFoundError(f"CSV not found at {CSV_PATH}. Put gyro_samples_clean.csv
in the same folder as app.py.")

df_raw = pd.read_csv(CSV_PATH)

# Try to detect timestamp column
time_col = None
for cand in ["timestamp", "time", "datetime", "date"]:
    if cand in df_raw.columns:
        time_col = cand
        break

if time_col is not None:
    # parse to datetime if possible; if it fails, we keep as string
    try:
        df_raw[time_col] = pd.to_datetime(df_raw[time_col])
    except Exception:
        pass

# Try to detect gyro axis columns (robust to various names)
axis_candidates = {
    "x": ["x", "gx", "gyro_x", "gyroX", "accel_x"],
    "y": ["y", "gy", "gyro_y", "gyroY", "accel_y"],
```

```
    "z": ["z", "gz", "gyro_z", "gyroZ", "accel_z"],
}
axis_map = {}
for axis, names in axis_candidates.items():
    for name in names:
        if name in df_raw.columns:
            axis_map[axis] = name
            break

# If nothing found, fall back to the first three numeric columns
if len(axis_map) == 0:
    numeric_cols = [c for c in df_raw.columns if
pd.api.types.is_numeric_dtype(df_raw[c])]
    if len(numeric_cols) >= 3:
        axis_map = {"x": numeric_cols[0], "y": numeric_cols[1], "z": numeric_cols[2]}
    elif len(numeric_cols) > 0:
        # Partial availability is okay; we'll only show what exists
        for lbl, col in zip(["x", "y", "z"], numeric_cols):
            axis_map[lbl] = col

# Build a working DataFrame with friendly column names
df = df_raw.copy()
for short, real in axis_map.items():
    if short != real and real in df.columns:
        df.rename(columns={real: short}, inplace=True)

# Index column for plotting if no time available
df["sample_idx"] = np.arange(len(df))

# Which axes do we actually have?
available_axes = [a for a in ["x", "y", "z"] if a in df.columns and
pd.api.types.is_numeric_dtype(df[a])]
if len(available_axes) == 0:
    raise ValueError("No numeric X/Y/Z gyro columns found. Ensure your CSV has x,y,z
(or recognizable) numeric columns.")

# ---------- Build the app ----------
app = Dash(__name__)
app.title = "Gyroscope CSV Explorer"

app.layout = html.Div(
    className="container",
    children=[
        html.H1("Gyroscope CSV Explorer", style={"marginBottom": "0.5rem"}),
        html.Div(
            f"Loaded {len(df)} samples from {os.path.basename(CSV_PATH)}",
            style={"color": "#666", "marginBottom": "1rem"},
        ),
```

```
        # Controls row
        html.Div(
            style={"display": "grid", "gridTemplateColumns": "1fr 1fr 1fr 0.5fr
0.5fr", "gap": "12px", "alignItems": "end"},
            children=[
                html.Div([
                    html.Label("Chart type"),
                    dcc.Dropdown(
                        id="chart-type",
                        options=[
                            {"label": "Line", "value": "line"},
                            {"label": "Scatter", "value": "scatter"},
                            {"label": "Histogram", "value": "hist"},
                            {"label": "Box (distribution)", "value": "box"},
                        ],
                        value="line",
                        clearable=False,
                    ),
                ]),
                html.Div([
                    html.Label("Axes (X/Y/Z)"),
                    dcc.Dropdown(
                        id="axes",
                        options=[{"label": ax.upper(), "value": ax} for ax in
available_axes],
                        value=available_axes,  # default: show all that exist
                        multi=True,
                    ),
                ]),
                html.Div([
                    html.Label("Samples per page (N)"),
                    dcc.Input(
                        id="n-samples",
                        type="number",
                        min=10, step=10,
                        value=min(500, len(df)),  # sensible default
                        debounce=True,
                        style={"width": "100%"},
                    ),
                ]),
                html.Div([
                    html.Label(" "),
                    html.Button("Prev", id="prev-btn", n_clicks=0, style={"width":
"100%"}),
                ]),
                html.Div([
                    html.Label(" "),
```

```
                    html.Button("Next", id="next-btn", n_clicks=0, style={"width":
"100%"}),
                ]),
            ]
        ),

        # Store for paging offset
        dcc.Store(id="page-offset", data=0),

        # Window info
        html.Div(id="window-info", style={"margin": "12px 0", "color": "#444"}),

        # Graph
        dcc.Graph(id="gyro-graph", style={"height": "520px"}),

        # Summary table
        html.H3("Summary of Current Window"),
        dash_table.DataTable(
            id="summary-table",
            style_table={"maxWidth": "720px"},
            style_cell={"textAlign": "center", "padding": "6px"},
            style_header={"fontWeight": "bold"},
        ),
    ]
)

# ---------- Helpers ----------
def compute_window(df_in, offset, n):
    """Return a slice of df_in from offset to offset+n, clipped to valid range."""
    total = len(df_in)
    if total == 0:
        return df_in.iloc[0:0], 0, 0, 0
    n = max(1, int(n)) if pd.notna(n) else total
    start = max(0, min(int(offset), max(0, total - 1)))
    end = min(total, start + n)
    return df_in.iloc[start:end], start, end, total

def make_summary(df_win, axes, start, end, total):
    """Build a tidy stats table for the selected axes in the current window."""
    rows = []
    for ax in axes:
        if ax in df_win.columns and pd.api.types.is_numeric_dtype(df_win[ax]):
            s = df_win[ax].dropna()
            if len(s) == 0:
                continue
            stats = {
                "Axis": ax.upper(),
                "Count": int(s.count()),
```

```
                "Mean": float(s.mean()),
                "Std": float(s.std(ddof=1)) if s.count() > 1 else 0.0,
                "Min": float(s.min()),
                "25%": float(s.quantile(0.25)),
                "Median": float(s.median()),
                "75%": float(s.quantile(0.75)),
                "Max": float(s.max()),
            }
            rows.append(stats)
    meta = f"Showing samples {start}–{end-1} of {total} (window size {end-start})"
    return rows, meta

def build_figure(df_win, axes, chart_type):
    """Return a Plotly Figure according to chart type and selected axes."""
    # X-axis: timestamp if available, else sample_idx
    x_axis = "sample_idx"
    x_title = "Sample #"
    if "timestamp" in df_win.columns or "time" in df_win.columns or "datetime" in
df_win.columns or "date" in df_win.columns:
        # choose first available time-like column in the original priority
        for cand in ["timestamp", "time", "datetime", "date"]:
            if cand in df_win.columns:
                x_axis = cand
                x_title = cand.capitalize()
                break

    fig = go.Figure()

    if chart_type in ("line", "scatter"):
        mode = "lines" if chart_type == "line" else "markers"
        for ax in axes:
            if ax in df_win.columns:
                fig.add_trace(go.Scatter(
                    x=df_win[x_axis],
                    y=df_win[ax],
                    mode=mode,
                    name=ax.upper(),
                ))
        fig.update_layout(
            xaxis_title=x_title,
            yaxis_title="Angular rate / units",
            legend_title="Axis",
            margin=dict(l=40, r=20, t=20, b=40),
        )

    elif chart_type == "hist":
        # Overlaid histograms for each axis
        for ax in axes:
```

```
            if ax in df_win.columns:
                fig.add_trace(go.Histogram(
                    x=df_win[ax],
                    name=ax.upper(),
                    opacity=0.65,
                ))
        fig.update_layout(
            barmode="overlay",
            xaxis_title="Value",
            yaxis_title="Count",
            legend_title="Axis",
            margin=dict(l=40, r=20, t=20, b=40),
        )

    elif chart_type == "box":
        # Box plots for distribution comparison
        for ax in axes:
            if ax in df_win.columns:
                fig.add_trace(go.Box(
                    y=df_win[ax],
                    name=ax.upper(),
                    boxmean=True
                ))
        fig.update_layout(
            yaxis_title="Value",
            margin=dict(l=40, r=20, t=20, b=40),
        )

    return fig

# ---------- Callbacks ----------
@app.callback(
    Output("page-offset", "data"),
    Input("prev-btn", "n_clicks"),
    Input("next-btn", "n_clicks"),
    Input("n-samples", "value"),
    Input("axes", "value"),
    Input("chart-type", "value"),
    State("page-offset", "data"),
    prevent_initial_call=True
)
def update_offset(prev_clicks, next_clicks, n_samples, axes, chart_type, offset):
    """
    Paging logic.
    - If Prev/Next clicked: move by N samples.
    - If other controls changed (n, axes, chart type): reset to 0 for a fresh view.
    """
    triggered = ctx.triggered_id
```

```
    total = len(df)
    n = max(1, int(n_samples)) if pd.notna(n_samples) else total
    max_start = max(0, total - n)

    if triggered == "prev-btn":
        new_offset = max(0, int(offset or 0) - n)
    elif triggered == "next-btn":
        new_offset = min(max_start, int(offset or 0) + n)
    else:
        # Controls changed → reset paging
        new_offset = 0
    return int(new_offset)

@app.callback(
    Output("gyro-graph", "figure"),
    Output("summary-table", "data"),
    Output("summary-table", "columns"),
    Output("window-info", "children"),
    Input("chart-type", "value"),
    Input("axes", "value"),
    Input("n-samples", "value"),
    Input("page-offset", "data"),
)
def update_outputs(chart_type, axes, n_samples, offset):
    # normalize axes input (could be None or single str)
    if axes is None:
        axes = []
    if isinstance(axes, str):
        axes = [axes]
    axes = [a for a in axes if a in available_axes]
    if not axes:
        # Fall back to at least one axis
        axes = available_axes[:1]

    df_win, start, end, total = compute_window(df, offset or 0, n_samples)
    fig = build_figure(df_win, axes, chart_type)
    rows, meta = make_summary(df_win, axes, start, end, total)

    columns = [{"name": k, "id": k} for k in (rows[0].keys() if rows else ["Axis",
"Count", "Mean", "Std", "Min", "25%", "Median", "75%", "Max"])]
    return fig, rows, columns, meta

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

The app loads **gyro_samples_clean.csv** from the project root and tries to auto-detect gyro columns for X/Y/Z .It also detects a timestamp column if present; otherwise it uses a running sample index as the x-axis. The layout shows controls for Chart type, Axis selection (X/Y/Z), Window size (N samples), and Prev/Next buttons, a live graph, and a summary table. A hidden dcc. Store keeps the current page offset so the app can page through the dataset in stable windows.

A single callback slices the data into a window and builds a Plotly figure:

Line and Scatter: adds one trace per selected axis vs. time (or sample index).

Histogram: overlays distributions of selected axes within the current window.

Box: adds one box per axis to compare spread/median/outliers.

Axis labels and legends are automatically set; if a timestamp column exists, it's used as the x-axis.

Chart type, axis selection, and N samples recalculate and draw the graph and reset paging to the first window for clarity. The summary table calculates descriptive stats (count, mean, std, min/max, quartiles) for the currently displayed data, so it always matches the graph.

https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=5453c808-9077-426c-9091-b3460064965b