Student name: Tien Phat Nguyen

Student ID: 223171213

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

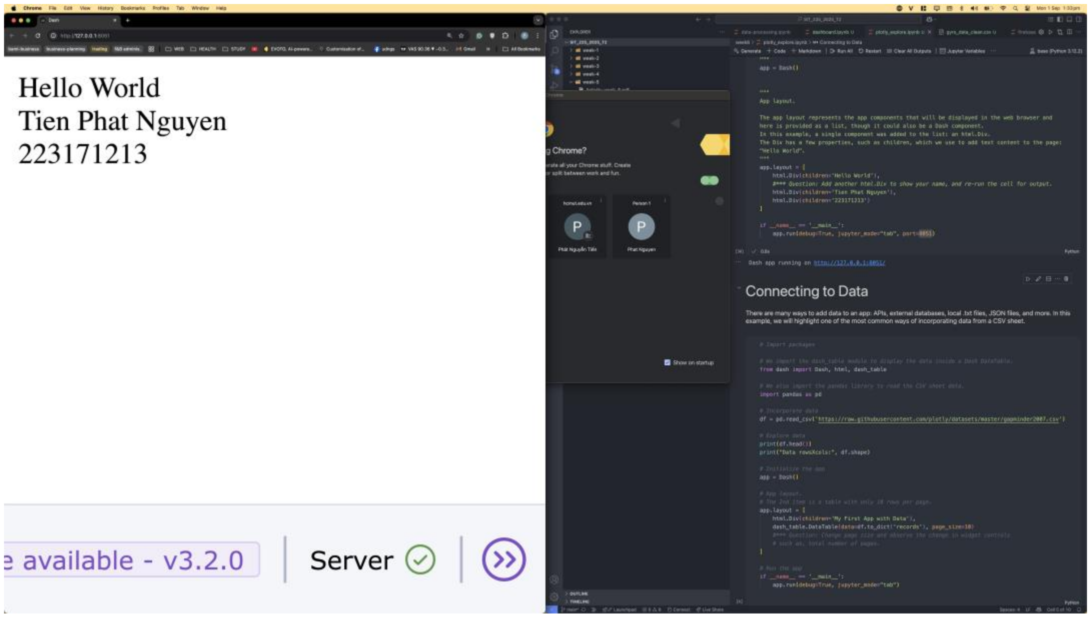
No hardware is required.

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. Answer: <Your 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: <Your answer>

Hello World
Tien Phat Nguyen
223171213

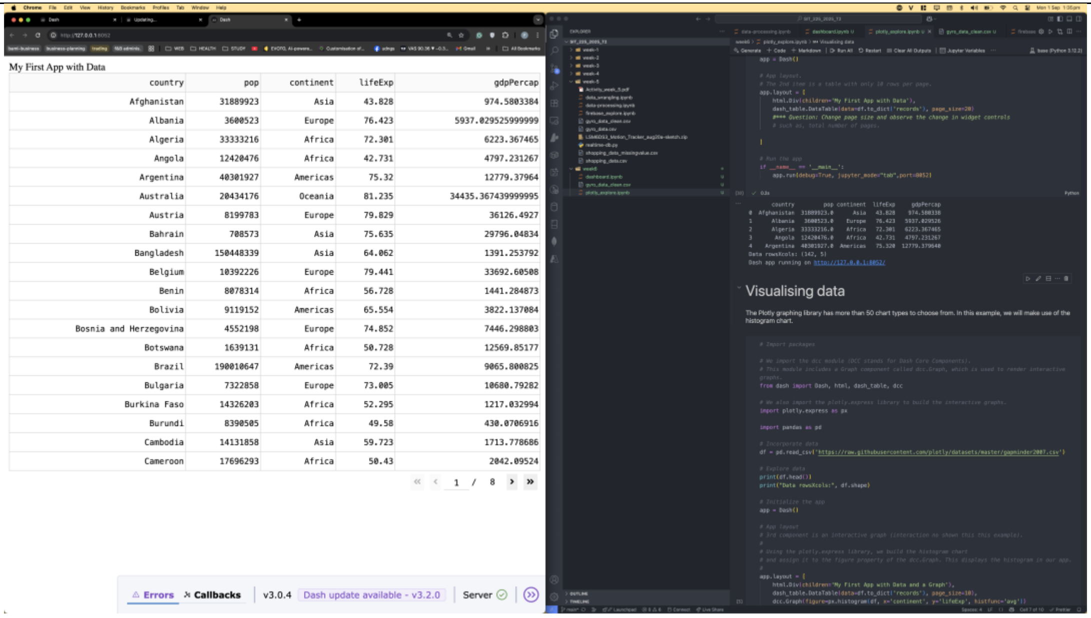
```
app = Dash()

"""
App layout.

The app layout represents the app components that will be displayed in the web browser and
here is provided as a list, though it could also be a Dash component.
In this example, a single component was added to the list: an html.Div.
The Div has a few properties, such as children, which we use to add text content to the page:
"Hello World".
"""
app.layout = [
    html.Div(children='Hello World'),
    #### Question: Add another html.Div to show your name, and re-run the cell for output.
    html.Div(children='Tien Phat Nguyen'),
    html.Div(children='223171213')
]

if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab", port=8051)
```

```
Dash app running on http://127.0.0.1:8051/
```

Connecting to Data

There are many ways to add data to an app: APIs, external databases, local .txt files, JSON files, and more. In this example, we will highlight one of the most common ways of incorporating data from a CSV sheet.

```
# Import packages

# We import the dash_table module to display the data inside a Dash DataTable.
from dash import Dash, html, dash_table

# We also import the pandas library to read the CSV sheet data.
import pandas as pd

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout.
# The 2nd item is a table with only 10 rows per page.
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10)
    #### Question: Change page size and observe the change in widget controls
    # such as, total number of pages.
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab")
```



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
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	708573	Asia	75.635	29796.04834
Bangladesh	150448339	Asia	64.062	1391.253792
Belgium	10392226	Europe	79.441	33692.60508
Benin	8078314	Africa	56.728	1441.284873
Bolivia	9119152	Americas	65.554	3822.137084
Bosnia and Herzegovina	4552198	Europe	74.852	7446.298803
Botswana	1639131	Africa	50.728	12569.85177
Brazil	190010647	Americas	72.39	9065.800825
Bulgaria	7322858	Europe	73.005	10680.79282
Burkina Faso	14326203	Africa	52.295	1217.032994
Burundi	8390505	Africa	49.58	430.0706916
Cambodia	14131858	Asia	59.723	1713.778686
Cameroon	17696293	Africa	50.43	2042.09524

« ‹ 1 / 8 › »

Errors Callbacks v3.0.4 Dash update available - v3.2.0 Server

```
app = Dash()

# App layout.
# The 2nd item is a table with only 10 rows per page.
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=20)
    #*** Question: Change page size and observe the change in widget controls
    # such as, total number of pages.

]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab",port=8052)
```

```
       country          pop continent  lifeExp     gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1      Albania   3600523.0    Europe   76.423   5937.029526
2      Algeria  33333216.0    Africa   72.301   6223.367465
3       Angola  12420476.0    Africa   42.731   4797.231267
4    Argentina  40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8052/
```

Visualising data

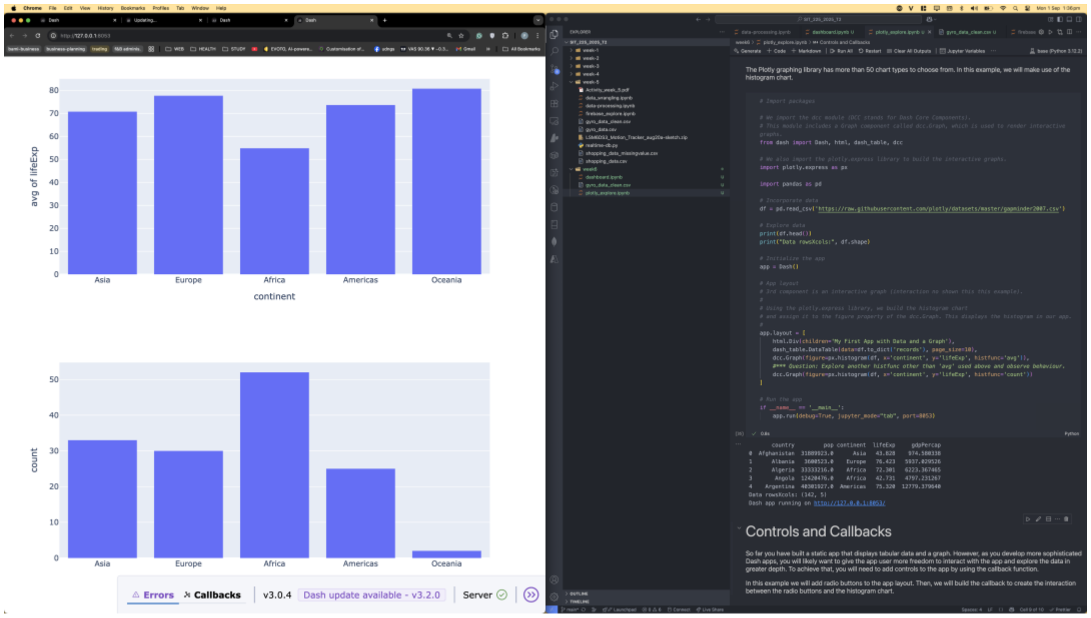
The Plotly graphing library has more than 50 chart types to choose from. In this example, we will make use of the histogram chart.

```
# Import packages

# We import the dcc module (DCC stands for Dash Core Components).
# This module includes a Graph component called dcc.Graph, which is used to render interactive
graphs.
from dash import Dash, html, dash_table, dcc

# We also import the plotly.express library to build the interactive graphs.
import plotly.express as px

import pandas as pd

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout
# 3rd component is an interactive graph (interaction no shown this this example).
#
# Using the plotly.express library, we build the histogram chart
# and assign it to the figure property of the dcc.Graph. This displays the histogram in our app.
#
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='avg'))
```

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>



The Plotly graphing library has more than 50 chart types to choose from. In this example, we will make use of the histogram chart.

```
# Import packages

# We import the dcc module (DCC stands for Dash Core Components).
# This module includes a Graph component called dcc.Graph, which is used to render interactive
graphs.
from dash import Dash, html, dash_table, dcc

# We also import the plotly.express library to build the interactive graphs.
import plotly.express as px

import pandas as pd

# Incorporate data
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminder2007.csv')

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout
# 3rd component is an interactive graph (interaction no shown this this example).
#
# Using the plotly.express library, we build the histogram chart
# and assign it to the figure property of the dcc.Graph. This displays the histogram in our app.
#
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='avg')),
    #*** Question: Explore another histfunc other than 'avg' used above and observe behaviour.
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp', histfunc='count'))
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab", port=8053)
```

```
       country          pop continent  lifeExp     gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1      Albania   3600523.0    Europe   76.423   5937.029526
2      Algeria  33333216.0    Africa   72.301   6223.367465
3       Angola  12420476.0    Africa   42.731   4797.231267
4    Argentina  40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8053/
```

Controls and Callbacks

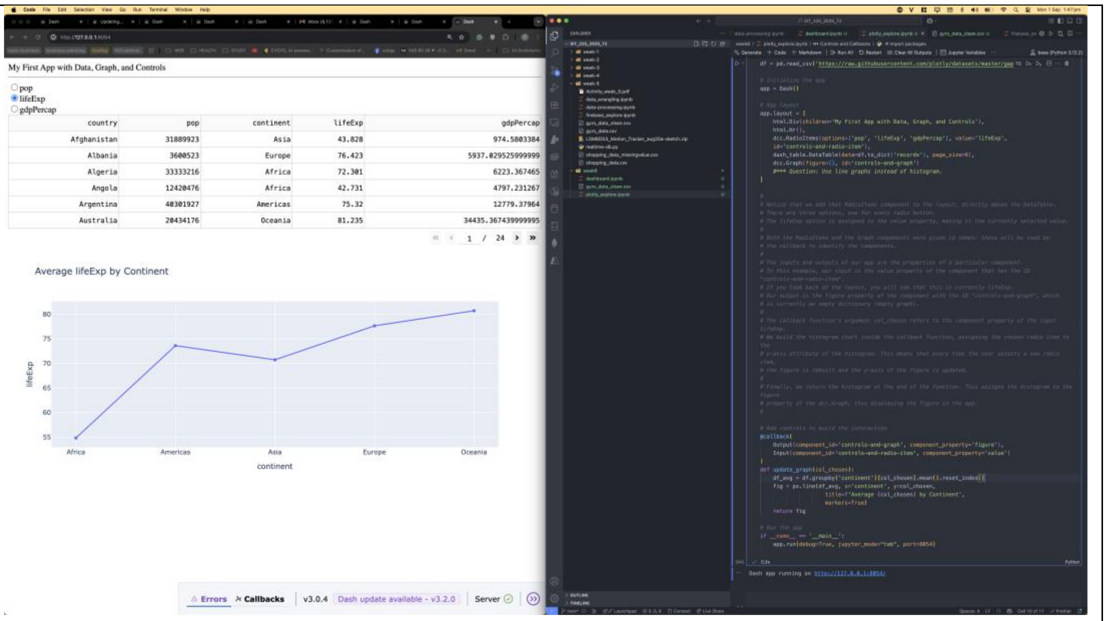
So far you have built a static app that displays tabular data and a graph. However, as you develop more sophisticated Dash apps, you will likely want to give the app user more freedom to interact with the app and explore the data in greater depth. To achieve that, you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the callback to create the interaction between the radio buttons and the histogram chart.

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: <Your answer>



My First App with Data, Graph, and Controls

pop [✓ lifeExp] gdpPercap

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
Australia	20434176	Oceania	81.235	34435.367439999995

```python
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gap

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'], value='lifeExp',
    id='controls-and-radio-item'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
    #*** Question: Use line graphs instead of histogram.
]

#
# Notice that we add that RadioItems component to the layout, directly above the DataTable.
# There are three options, one for every radio button.
# The lifeExp option is assigned to the value property, making it the currently selected value.
#
# Both the RadioItems and the Graph components were given id names: these will be used by
# the callback to identify the components.
#
# The inputs and outputs of our app are the properties of a particular component.
# In this example, our input is the value property of the component that has the ID
"controls-and-radio-item".
# If you look back at the layout, you will see that this is currently lifeExp.
# Our output is the figure property of the component with the ID "controls-and-graph", which
# is currently an empty dictionary (empty graph).
#
# The callback function's argument col_chosen refers to the component property of the input
lifeExp.
# We build the histogram chart inside the callback function, assigning the chosen radio item to
the
# y-axis attribute of the histogram. This means that every time the user selects a new radio
item,
# the figure is rebuilt and the y-axis of the figure is updated.
#
# Finally, we return the histogram at the end of the function. This assigns the histogram to the
figure
# property of the dcc.Graph, thus displaying the figure in the app.
#

# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph', component_property='figure'),
    Input(component_id='controls-and-radio-item', component_property='value')
)
def update_graph(col_chosen):
    df_avg = df.groupby('continent')[col_chosen].mean().reset_index()
    fig = px.line(df_avg, x='continent', y=col_chosen,
                  title=f'Average {col_chosen} by Continent',
                  markers=True)
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab", port=8054)
```

```
Dash app running on http://127.0.0.1:8054/
```

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.

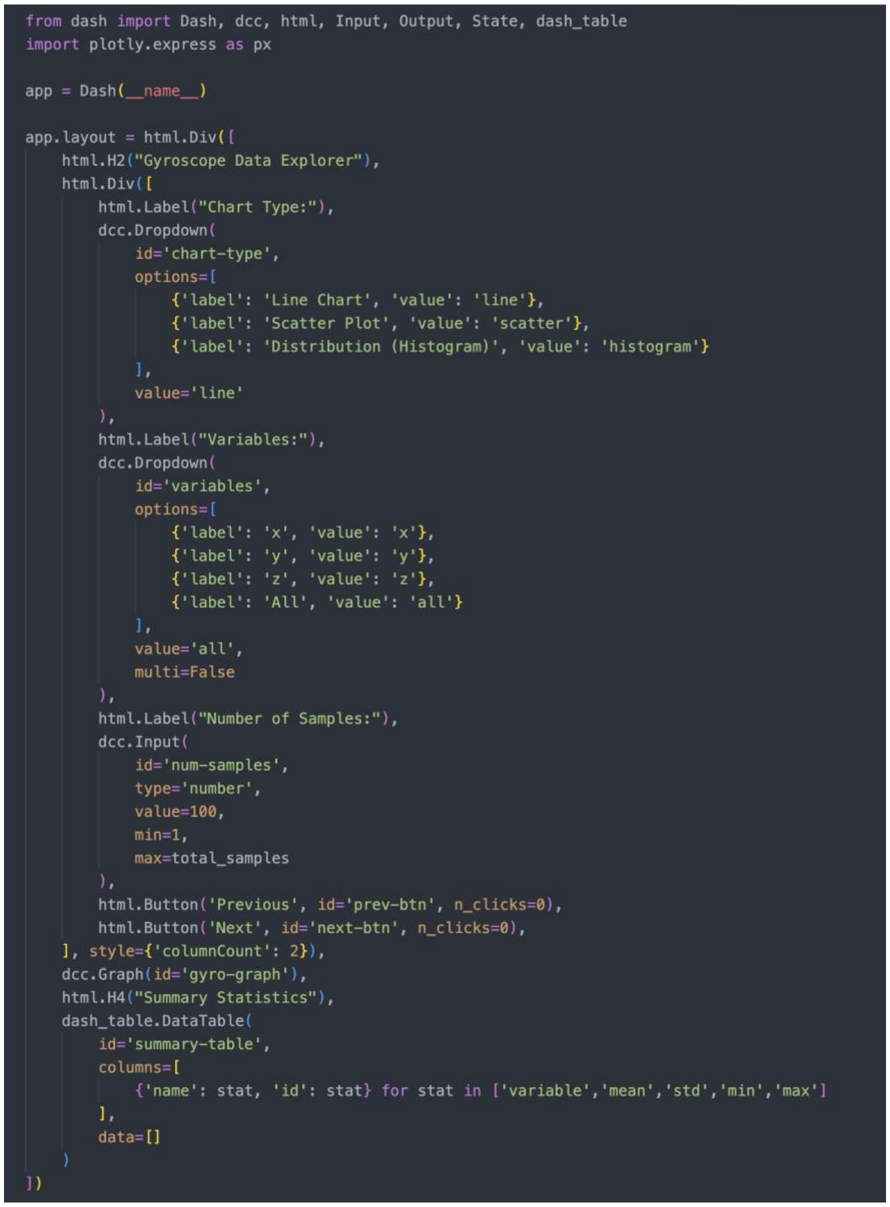
Answer: <Your answer>
I will use Plotly Dash to build a live dashboard showing each sensor stream with line graphs updating in near real time and simple tables for recent values. It help me combine interactive controls so I can choose which metric to view, set time range filters, and compare average vs current readings, but sometimes the layout may not be very optimize. I also need be careful because sometimes data refresh too fast, charts becomes messy,

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.

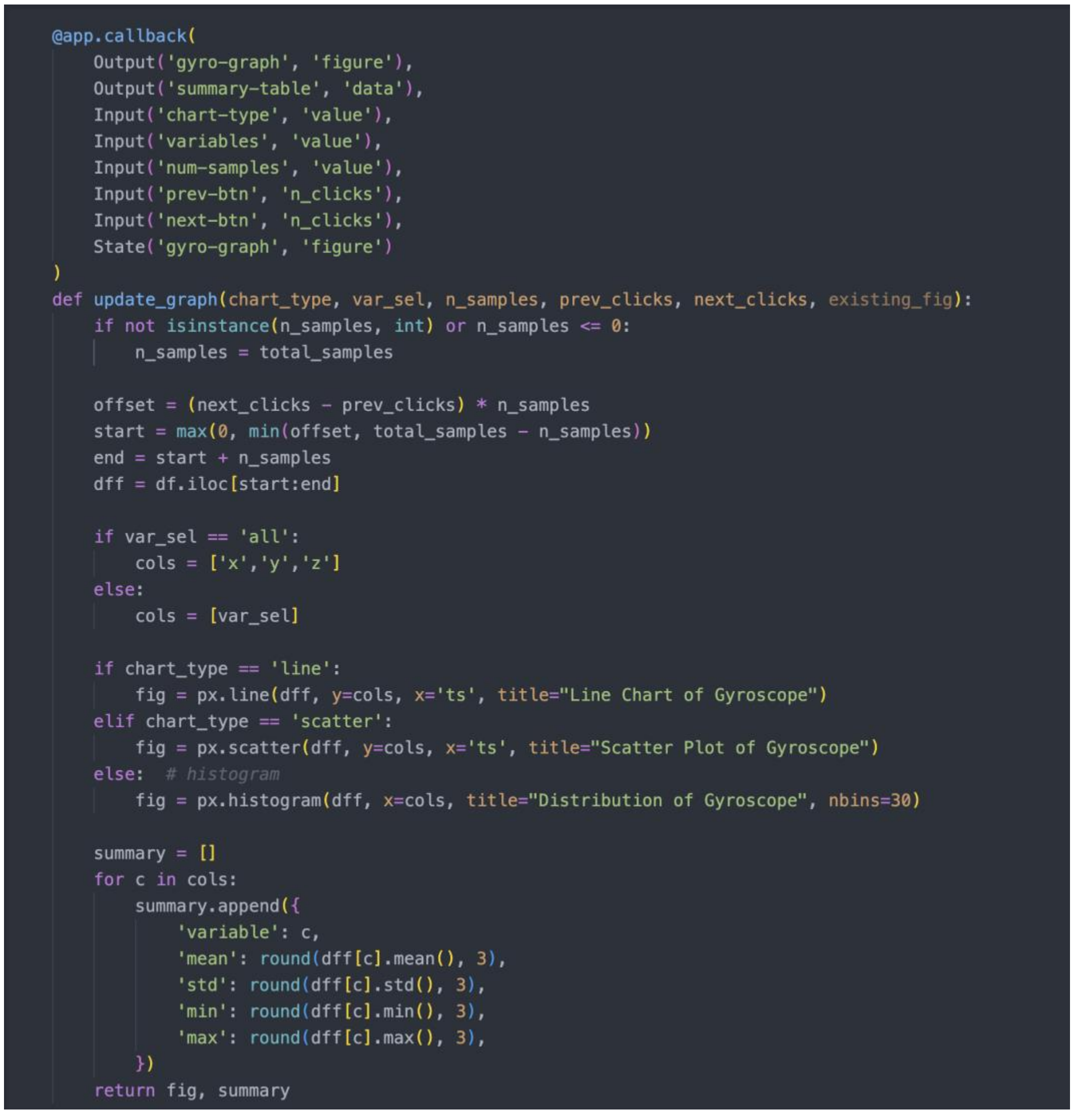
Answer: <Your answer>

Pass Task:



```python
from dash import Dash, dcc, html, Input, Output, State, dash_table
import plotly.express as px

app = Dash(__name__)

app.layout = html.Div([
    html.H2("Gyroscope Data Explorer"),
    html.Div([
        html.Label("Chart Type:"),
        dcc.Dropdown(
            id='chart-type',
            options=[
                {'label': 'Line Chart', 'value': 'line'},
                {'label': 'Scatter Plot', 'value': 'scatter'},
                {'label': 'Distribution (Histogram)', 'value': 'histogram'}
            ],
            value='line'
        ),
        html.Label("Variables:"),
        dcc.Dropdown(
            id='variables',
            options=[
                {'label': 'x', 'value': 'x'},
                {'label': 'y', 'value': 'y'},
                {'label': 'z', 'value': 'z'},
                {'label': 'All', 'value': 'all'}
            ],
            value='all',
            multi=False
        ),
        html.Label("Number of Samples:"),
        dcc.Input(
            id='num-samples',
            type='number',
            value=100,
            min=1,
            max=total_samples
        ),
        html.Button('Previous', id='prev-btn', n_clicks=0),
        html.Button('Next', id='next-btn', n_clicks=0),
    ], style={'columnCount': 2}),
    dcc.Graph(id='gyro-graph'),
    html.H4("Summary Statistics"),
    dash_table.DataTable(
        id='summary-table',
        columns=[
            {'name': stat, 'id': stat} for stat in ['variable','mean','std','min','max']
        ],
        data=[]
    )
])
```

This code sets up a Dash app that shows gyroscope data by creating a web layout with dropdowns for choosing the chart type and variables, an input box for the number of samples, and “Previous/Next” buttons for navigation. It then displays the selected chart in a Graph component and a data table below to show summary statistics like mean, standard deviation, minimum, and maximum.



```python
@app.callback(
    Output('gyro-graph', 'figure'),
    Output('summary-table', 'data'),
    Input('chart-type', 'value'),
    Input('variables', 'value'),
    Input('num-samples', 'value'),
    Input('prev-btn', 'n_clicks'),
    Input('next-btn', 'n_clicks'),
    State('gyro-graph', 'figure')
)
def update_graph(chart_type, var_sel, n_samples, prev_clicks, next_clicks, existing_fig):
    if not isinstance(n_samples, int) or n_samples <= 0:
        n_samples = total_samples

    offset = (next_clicks - prev_clicks) * n_samples
    start = max(0, min(offset, total_samples - n_samples))
    end = start + n_samples
    dff = df.iloc[start:end]

    if var_sel == 'all':
        cols = ['x','y','z']
    else:
        cols = [var_sel]

    if chart_type == 'line':
        fig = px.line(dff, y=cols, x='ts', title="Line Chart of Gyroscope")
    elif chart_type == 'scatter':
        fig = px.scatter(dff, y=cols, x='ts', title="Scatter Plot of Gyroscope")
    else:  # histogram
        fig = px.histogram(dff, x=cols, title="Distribution of Gyroscope", nbins=30)

    summary = []
    for c in cols:
        summary.append({
            'variable': c,
            'mean': round(dff[c].mean(), 3),
            'std': round(dff[c].std(), 3),
            'min': round(dff[c].min(), 3),
            'max': round(dff[c].max(), 3),
        })
    return fig, summary
```

This callback function first checks if the user's sample number is valid and then calculates which rows of the DataFrame to show by using the difference between next and previous clicks to set an offset and slice the data. After that, it decides which gyroscope variables (x, y, z, or a single axis) to plot and chooses the right Plotly Express chart type—line, scatter, or histogram—based on the user's dropdown selection. Finally, it computes simple statistics (mean, standard deviation, min, max) for each chosen variable and returns both the updated figure and a summary table.

Github: https://github.com/tienphatnguyendev/SIT_225_2025_T2/tree/main/week-6

Video: https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=20ab2f11-0fa3-452e-b5b8-b34b005f11b0

```python
# Fill in student ID and name
#
student_id = "223171213"
student_first_last_name = "Tien Phat Nguyen"
print(student_id, student_first_last_name)
```

```
223171213 Tien Phat Nguyen
```

```python
# install plotly and dash, if not yet already
! pip install plotly dash

import plotly, dash
print(plotly.__version__)
print(dash.__version__)
```

```
Requirement already satisfied: plotly in
/opt/anaconda3/lib/python3.12/site-packages (5.22.0)
Requirement already satisfied: dash in
/opt/anaconda3/lib/python3.12/site-packages (3.0.4)
Requirement already satisfied: tenacity>=6.2.0 in
/opt/anaconda3/lib/python3.12/site-packages (from plotly) (8.2.2)
Requirement already satisfied: packaging in
/opt/anaconda3/lib/python3.12/site-packages (from plotly) (23.2)
Requirement already satisfied: Flask<3.1,>=1.0.4 in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (3.0.3)
Requirement already satisfied: Werkzeug<3.1 in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (3.0.3)
Requirement already satisfied: importlib-metadata in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (7.0.1)
Requirement already satisfied: typing-extensions>=4.1.1 in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (4.11.0)
Requirement already satisfied: requests in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (2.31.0)
Requirement already satisfied: retrying in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (1.3.4)
Requirement already satisfied: nest-asyncio in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in
/opt/anaconda3/lib/python3.12/site-packages (from dash) (69.5.1)
Requirement already satisfied: Jinja2>=3.1.2 in
/opt/anaconda3/lib/python3.12/site-packages (from Flask<3.1,>=1.0.4-
>dash) (3.1.4)
Requirement already satisfied: itsdangerous>=2.1.2 in
/opt/anaconda3/lib/python3.12/site-packages (from Flask<3.1,>=1.0.4-
>dash) (2.2.0)
Requirement already satisfied: click>=8.1.3 in
/opt/anaconda3/lib/python3.12/site-packages (from Flask<3.1,>=1.0.4-
>dash) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in
/opt/anaconda3/lib/python3.12/site-packages (from Flask<3.1,>=1.0.4-
>dash) (1.6.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/opt/anaconda3/lib/python3.12/site-packages (from Werkzeug<3.1->dash)
(2.1.3)
Requirement already satisfied: zipp>=0.5 in
/opt/anaconda3/lib/python3.12/site-packages (from importlib-metadata-
>dash) (3.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/anaconda3/lib/python3.12/site-packages (from requests->dash)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
/opt/anaconda3/lib/python3.12/site-packages (from requests->dash)
(3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/anaconda3/lib/python3.12/site-packages (from requests->dash)
(2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/anaconda3/lib/python3.12/site-packages (from requests->dash)
(2024.7.4)
Requirement already satisfied: six>=1.7.0 in
/opt/anaconda3/lib/python3.12/site-packages (from retrying->dash)
(1.16.0)
5.22.0
3.0.4
```

Hello world

Building and launching an app with Dash can be done with just 5 lines of code. Follow the tutorial (https://dash.plotly.com/tutorial) for more detail.

```python
from dash import Dash, html

"""
Initialize the app.

This line is known as the Dash constructor and is responsible for
initializing your app.
It is almost always the same for any Dash app you create.
"""
app = Dash()


"""
App layout.

The app layout represents the app components that will be displayed in
the web browser and
```

```
here is provided as a list, though it could also be a Dash component.
In this example, a single component was added to the list: an
html.Div.
The Div has a few properties, such as children, which we use to add
text content to the page: "Hello World".
"""
app.layout = [
    html.Div(children='Hello World'),
    #*** Question: Add another html.Div to show your name, and re-run
the cell for output.
    html.Div(children='Tien Phat Nguyen'),
    html.Div(children='223171213')
]

if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab", port=8051)
```

```
Dash app running on http://127.0.0.1:8051/

<IPython.core.display.Javascript object>
```

Connecting to Data

There are many ways to add data to an app: APIs, external databases, local .txt files, JSON files, and more. In this example, we will highlight one of the most common ways of incorporating data from a CSV sheet.

```
# Import packages

# We import the dash_table module to display the data inside a Dash
DataTable.
from dash import Dash, html, dash_table

# We also import the pandas library to read the CSV sheet data.
import pandas as pd

# Incorporate data
df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/
gapminder2007.csv')

# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()
```

```
# App layout.
# The 2nd item is a table with only 10 rows per page.
app.layout = [
    html.Div(children='My First App with Data'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=20)
    #*** Question: Change page size and observe the change in widget
controls
    # such as, total number of pages.

]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab",port=8052)
```

```
       country         pop continent  lifeExp     gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1      Albania   3600523.0    Europe   76.423   5937.029526
2      Algeria  33333216.0    Africa   72.301   6223.367465
3       Angola  12420476.0    Africa   42.731   4797.231267
4    Argentina  40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8052/

<IPython.core.display.Javascript object>
```

Visualising data

The Plotly graphing library has more than 50 chart types to choose from. In this example, we will make use of the histogram chart.

```
# Import packages

# We import the dcc module (DCC stands for Dash Core Components).
# This module includes a Graph component called dcc.Graph, which is
used to render interactive graphs.
from dash import Dash, html, dash_table, dcc

# We also import the plotly.express library to build the interactive
graphs.
import plotly.express as px

import pandas as pd

# Incorporate data
df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/
gapminder2007.csv')
```

```
# Explore data
print(df.head())
print("Data rowsXcols:", df.shape)

# Initialize the app
app = Dash()

# App layout
# 3rd component is an interactive graph (interaction no shown this
this example).
#
# Using the plotly.express library, we build the histogram chart
# and assign it to the figure property of the dcc.Graph. This displays
the histogram in our app.
#
app.layout = [
    html.Div(children='My First App with Data and a Graph'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=10),
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp',
histfunc='avg')),
    #*** Question: Explore another histfunc other than 'avg' used
above and observe behaviour.
    dcc.Graph(figure=px.histogram(df, x='continent', y='lifeExp',
histfunc='count'))
]

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab", port=8053)

       country         pop continent  lifeExp     gdpPercap
0  Afghanistan  31889923.0      Asia   43.828    974.580338
1      Albania   3600523.0    Europe   76.423   5937.029526
2      Algeria  33333216.0    Africa   72.301   6223.367465
3       Angola  12420476.0    Africa   42.731   4797.231267
4    Argentina  40301927.0  Americas   75.320  12779.379640
Data rowsXcols: (142, 5)
Dash app running on http://127.0.0.1:8053/

<IPython.core.display.Javascript object>
```

Controls and Callbacks

So far you have built a static app that displays tabular data and a graph. However, as you develop more sophisticated Dash apps, you will likely want to give the app user more freedom to interact with the app and explore the data in greater depth. To achieve that, you will need to add controls to the app by using the callback function.

In this example we will add radio buttons to the app layout. Then, we will build the callback to create the interaction between the radio buttons and the histogram chart.

```
# Import packages

# We import dcc like we did in the previous section to use dcc.Graph.
# In this example, we need dcc for dcc.Graph as well as the radio
buttons component, dcc.RadioItems.
#
# To work with the callback in a Dash app, we import the callback
module and the two arguments
# commonly used within the callback: Output and Input.
#
from dash import Dash, html, dash_table, dcc, callback, Output, Input
import pandas as pd
import plotly.express as px

# Incorporate data
df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/
gapminder2007.csv')

# Initialize the app
app = Dash()

# App layout
app.layout = [
    html.Div(children='My First App with Data, Graph, and Controls'),
    html.Hr(),
    dcc.RadioItems(options=['pop', 'lifeExp', 'gdpPercap'],
value='lifeExp', id='controls-and-radio-item'),
    dash_table.DataTable(data=df.to_dict('records'), page_size=6),
    dcc.Graph(figure={}, id='controls-and-graph')
    #*** Question: Use line graphs instead of histogram.
]

#
# Notice that we add that RadioItems component to the layout, directly
above the DataTable.
# There are three options, one for every radio button.
# The lifeExp option is assigned to the value property, making it the
currently selected value.
#
# Both the RadioItems and the Graph components were given id names:
these will be used by
# the callback to identify the components.
#
# The inputs and outputs of our app are the properties of a particular
component.
# In this example, our input is the value property of the component
```

```
that has the ID "controls-and-radio-item".
# If you look back at the layout, you will see that this is currently
lifeExp.
# Our output is the figure property of the component with the ID
"controls-and-graph", which
# is currently an empty dictionary (empty graph).
#
# The callback function's argument col_chosen refers to the component
property of the input lifeExp.
# We build the histogram chart inside the callback function, assigning
the chosen radio item to the
# y-axis attribute of the histogram. This means that every time the
user selects a new radio item,
# the figure is rebuilt and the y-axis of the figure is updated.
#
# Finally, we return the histogram at the end of the function. This
assigns the histogram to the figure
# property of the dcc.Graph, thus displaying the figure in the app.
#

# Add controls to build the interaction
@callback(
    Output(component_id='controls-and-graph',
component_property='figure'),
    Input(component_id='controls-and-radio-item',
component_property='value')
)
def update_graph(col_chosen):
    df_avg = df.groupby('continent')[col_chosen].mean().reset_index()
    fig = px.line(df_avg, x='continent', y=col_chosen,
                    title=f'Average {col_chosen} by Continent',
                    markers=True)
    return fig

# Run the app
if __name__ == '__main__':
    app.run(debug=True, jupyter_mode="tab", port=8054)
```

```
Dash app running on http://127.0.0.1:8054/
```

```
<IPython.core.display.Javascript object>
```