Optional Lab Dash

2026-01-12

Background

This lab is optional and is not required for the course.

Dash is a popular framework (created by the Plotly team) for building interactive dashboards in Python. It is free and open-source, but it is not required for this course. Although the basics are straightforward, building polished web applications usually requires additional knowledge (e.g., layout, callbacks, state, styling, deployment).

Dash apps run in a web browser using a client–server model:

Server (Python): runs your Dash code and computes updates.

Client (browser): renders the interface using HTML/CSS/JavaScript.

Dash can be run in Google Colab, but it is often frustrating because the server must run in a background thread and is difficult to stop/restart cleanly. For this lab, you should use a local Python environment.

To install Dash:

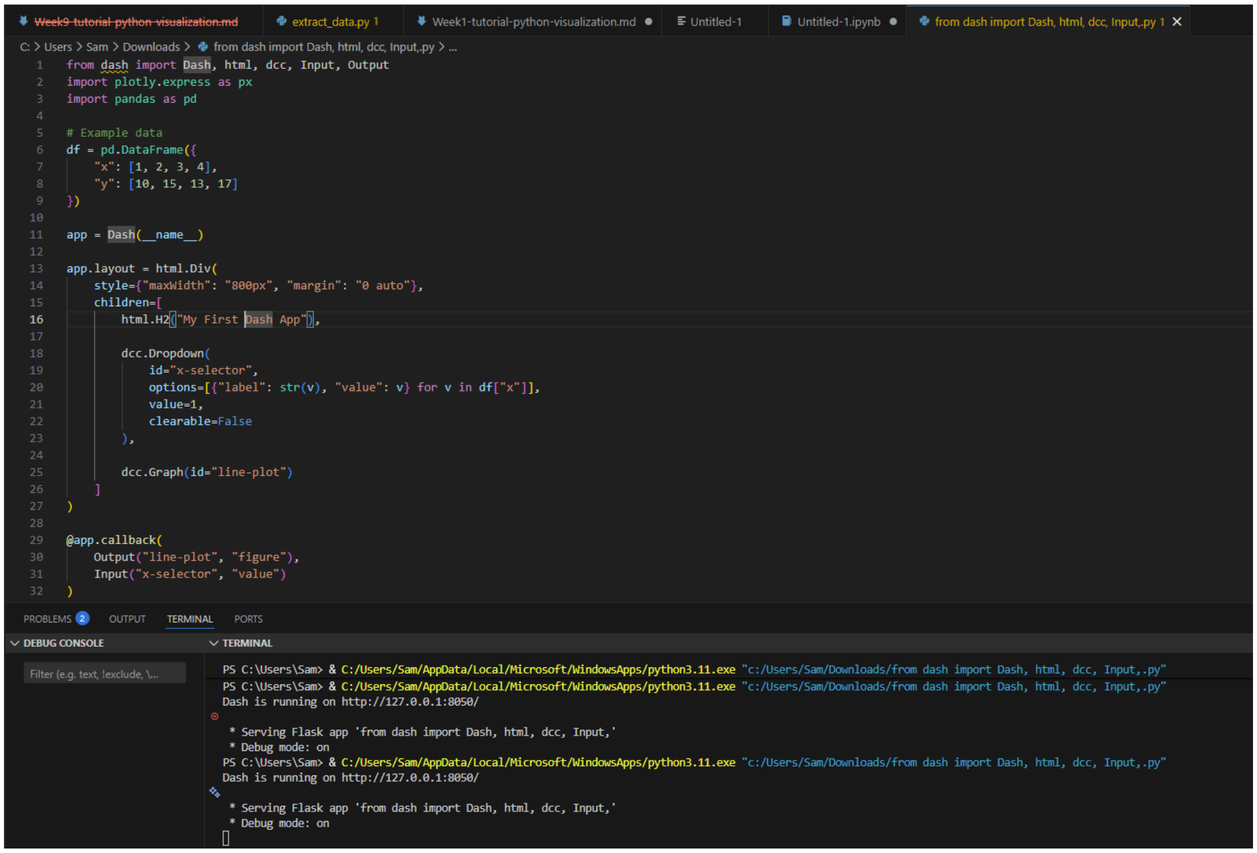
```
pip install dash
```

Getting started with a basic example.

```
from dash import Dash, html, dcc, Input, Output
import plotly.express as px
import pandas as pd

# Example data
df = pd.DataFrame({
    "x": [1, 2, 3, 4],
    "y": [10, 15, 13, 17]
})

app = Dash(__name__)

app.layout = html.Div([
    html.H3("Simple Dash App"),

    dcc.Dropdown(
        id="x",
        options=df["x"].tolist(),
        value=1
    ),
```

```python
    dcc.Graph(id="plot")
])

@app.callback(
    Output("plot", "figure"),
    Input("x", "value")
)
def update_plot(x_min):
    return px.line(df[df["x"] >= x_min], x="x", y="y")

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

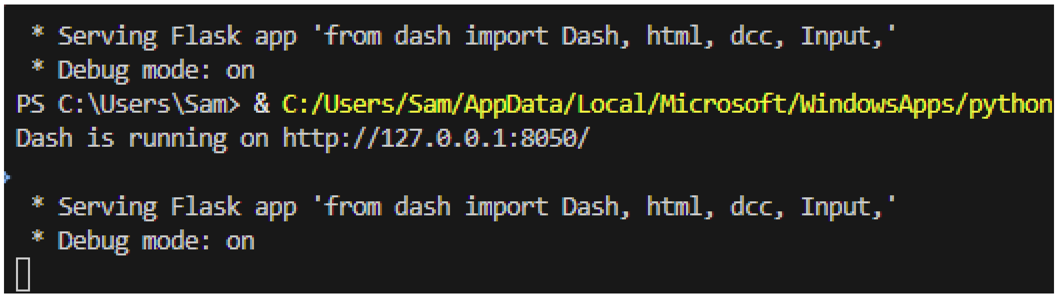
In my case I am using VScode to edit and run the python code.


~~Week9-tutorial-python-visualization.md~~
extract_data.py 1
Week1-tutorial-python-visualization.md
Untitled-1
Untitled-1.ipynb
from dash import Dash, html, dcc, Input,.py 1
C: > Users > Sam > Downloads > from dash import Dash, html, dcc, Input,.py > ...
PROBLEMS 2
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
TERMINAL
Filter (e.g. text, !exclude, \...

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

# Example data
df = pd.DataFrame({
    "x": [1, 2, 3, 4],
    "y": [10, 15, 13, 17]
})

app = Dash(__name__)

app.layout = html.Div(
    style={"maxWidth": "800px", "margin": "0 auto"},
    children=[
        html.H2("My First Dash App"),

        dcc.Dropdown(
            id="x-selector",
            options=[{"label": str(v), "value": v} for v in df["x"]],
            value=1,
            clearable=False
        ),

        dcc.Graph(id="line-plot")
    ]
)

@app.callback(
    Output("line-plot", "figure"),
    Input("x-selector", "value")
)
```

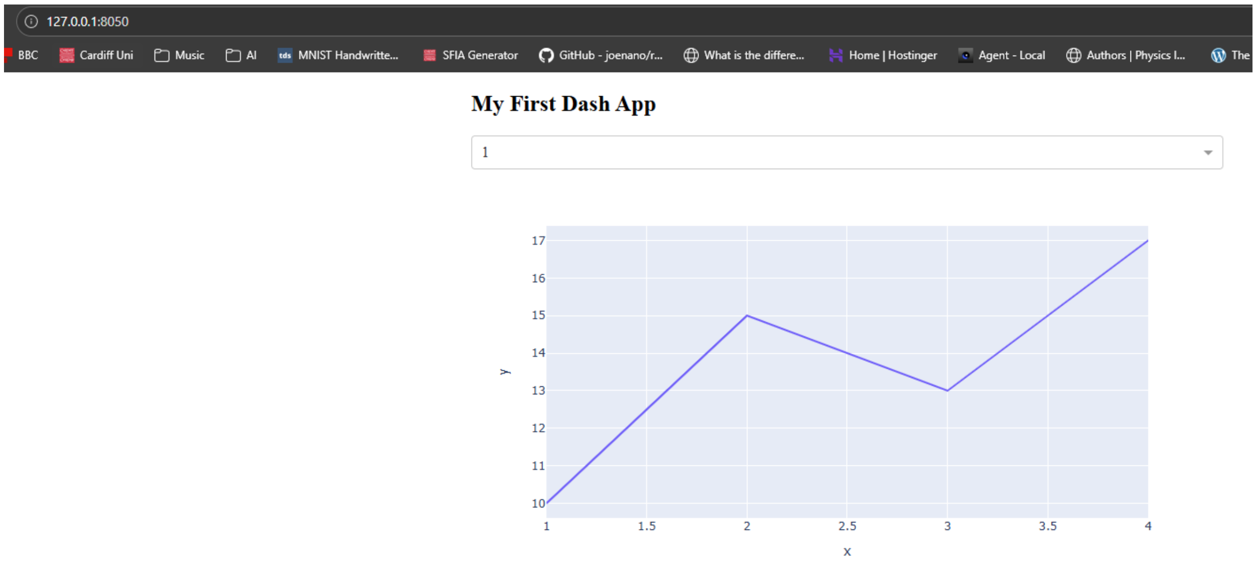
```
PS C:\Users\Sam> & C:/Users/Sam/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/Sam/Downloads/from dash import Dash, html, dcc, Input,.py"
PS C:\Users\Sam> & C:/Users/Sam/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/Sam/Downloads/from dash import Dash, html, dcc, Input,.py"
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'from dash import Dash, html, dcc, Input,'
 * Debug mode: on
PS C:\Users\Sam> & C:/Users/Sam/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/Sam/Downloads/from dash import Dash, html, dcc, Input,.py"
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'from dash import Dash, html, dcc, Input,'
 * Debug mode: on
```

When you run the app, your terminal will show a local URL (typically http://127.0.0.1:8050/).



```
 * Serving Flask app 'from dash import Dash, html, dcc, Input,'
 * Debug mode: on
PS C:\Users\Sam> & C:/Users/Sam/AppData/Local/Microsoft/WindowsApps/python
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app 'from dash import Dash, html, dcc, Input,'
 * Debug mode: on
```

Open the URL in your browser:



Note: While the server is running, changes you make to the code will not automatically update the output. Stop the app with Ctrl + C in the terminal, then run it again.

```
dcc.Dropdown(
        id="x",
        options=df["x"].tolist(),
        value=1
    ),
```

This component creates a dropdown selector. The options argument expects a list, and here we use the unique values from the dataframe. Dash core components are documented here: https://dash.plotly.com/dash-core-components

```
def update_plot(x_min):
    return px.line(df[df["x"] >= x_min], x="x", y="y")
```

This function filters the dataframe using the selected minimum value. For example, if you select 2, the plot uses only rows where x >= 2. We do not call update_plot() ourselves—Dash calls it automatically when the input changes.

```
@app.callback(
    Output("plot", "figure"),
    Input("x", "value")
)
```

The callback connects components together:

Input: the dropdown with id “x” (its “value” changes).

Output: the graph with id “plot” (its “figure” is updated).

Excersise

1. Create a Dash app (static figure)

Use the Trailing 365 data and chart to create a simple Dash app. At this stage the chart is static, so you can build the figure once and pass it directly into dcc.Graph.

```
app.layout = html.Div([
    html.H3("Trailing 365 Days Data"),
    dcc.Graph(id="trailing365-graph", figure=fig_trailing)
])
```

There is no need for the callback and update.

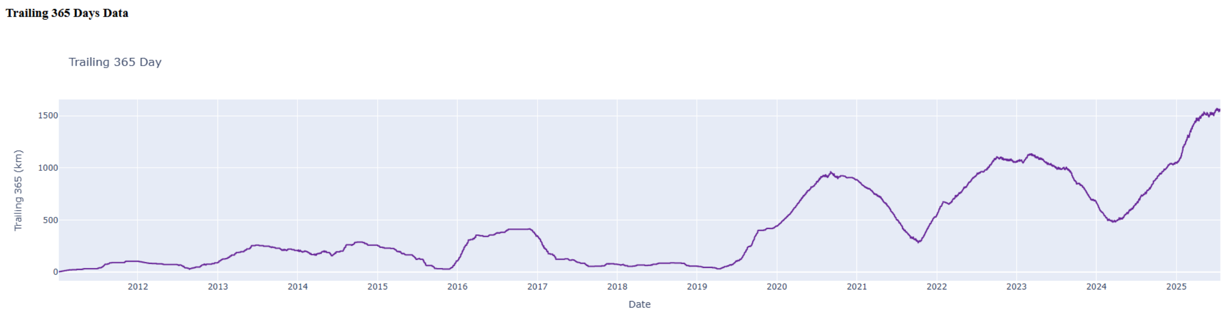


Figure 1: Static

2. Create a year dropdown.

Add a year dropdown so the chart updates when a year is selected. The figure should now be created inside an update_graph() function and returned by a callback.

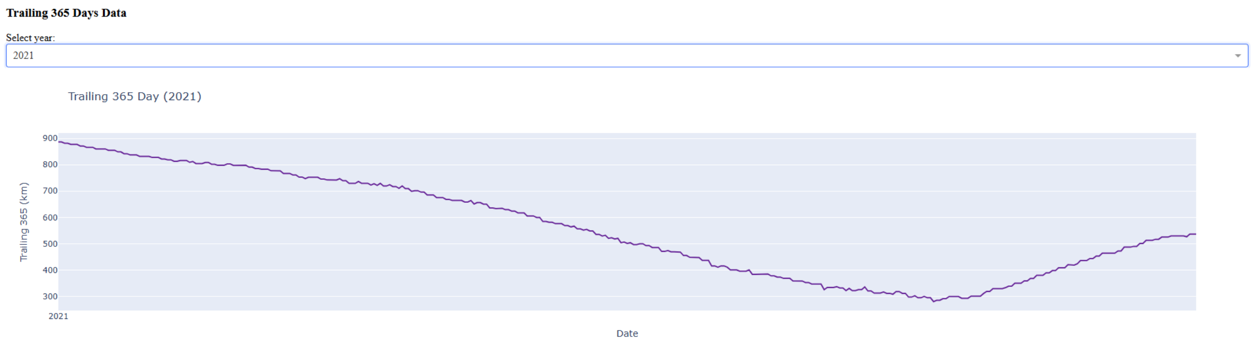


Figure 2: Year

Solutions

```
from dash import Dash, html, dcc, Input, Output
import plotly.express as px
import pandas as pd
# Load data from Google Drive
file_id = "1lgjURcPZaIDCqBa_bjbkmZk9i12M9x9W"
url = f"https://drive.google.com/uc?id={file_id}"
trailing365 = pd.read_csv(url)
```

```python
# Build the line chart
fig_trailing = px.line(
    trailing365,
    x="Date",
    y="Trailing365",
    title="Trailing 365 Day",
    labels={"Date": "Date", "Trailing365": "Trailing 365 (km)"}
)

fig_trailing.update_traces(
    line=dict(color="rebeccapurple", width=2)
)

fig_trailing.update_xaxes(
    dtick="M12",
    tickformat="%Y"
)

app = Dash(__name__)

app.layout = html.Div([
    html.H3("Trailing 365 Days Data"),
    dcc.Graph(id="trailing365-graph", figure=fig_trailing)
])

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

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

# Load data from Google Drive
file_id = "1lgjURcPZaIDCqBa_bjbkmZk9i12M9x9W"
url = f"https://drive.google.com/uc?id={file_id}"
trailing365 = pd.read_csv(url)

# Ensure Date is datetime and extract Year
trailing365["Date"] = pd.to_datetime(trailing365["Date"])
trailing365["Year"] = trailing365["Date"].dt.year

# Available years for dropdown
years = sorted(trailing365["Year"].unique())

app = Dash(__name__)

app.layout = html.Div([
    html.H3("Trailing 365 Days Data"),

    html.Label("Select year:"),
    dcc.Dropdown(
        id="year-selector",
        options=[{"label": str(y), "value": y} for y in years],
        value=years[-1],           # default = most recent year
        clearable=False
    ),

    dcc.Graph(id="trailing365-graph")
])

@app.callback(
    Output("trailing365-graph", "figure"),
    Input("year-selector", "value")
)
def update_graph(selected_year):
    df_year = trailing365[trailing365["Year"] == selected_year]

    fig = px.line(
        df_year,
        x="Date",
        y="Trailing365",
        title=f"Trailing 365 Day ({selected_year})",
        labels={"Date": "Date", "Trailing365": "Trailing 365 (km)"}
    )

    fig.update_traces(
        line=dict(color="rebeccapurple", width=2)
    )

    fig.update_xaxes(
        dtick="M12",
        tickformat="%Y"
    )

    return fig

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