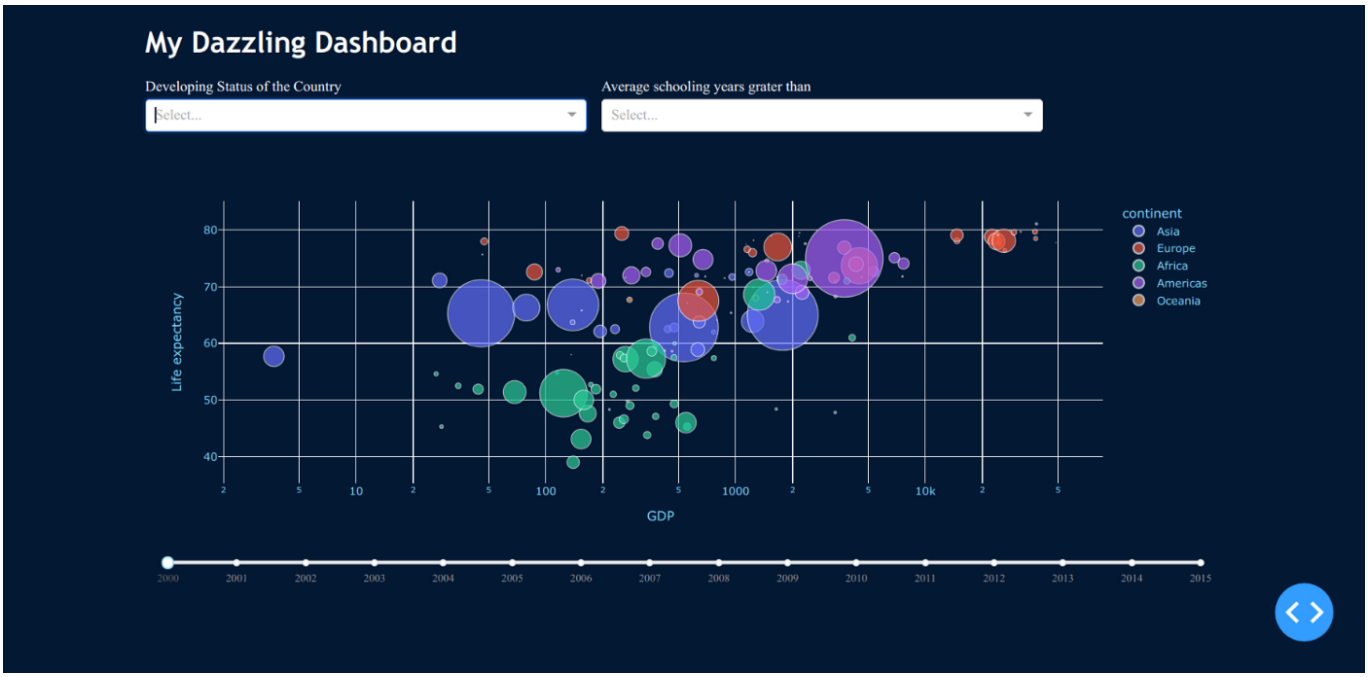
This is How I Create Dazzling Dashboards Purely in Python.

Plotly dash apps are the fastest way to build production-grade dashboards in python.

Thuwarakesh Murallie Sep 6 · 6 min read ★



Screenshot of the dashboard created in Python — by the Author.

I don't have to convince you why we need an interactive dashboard. But what most people don't know is that they don't have to buy expensive licenses of *Tableau* or

PowerBI. You don't have to enroll in a *JavaScript* course either.

Dash apps allow you to build interactive dashboards purely in Python. Interestingly, it could reach heights that popular BI platforms can not. Also, you can host it on your servers and your terms.

Why Dash? Why not Tableau, Power BI, or some JavaScript library?

BI Platforms such as **Tableau and PowerBI** do a fantastic job. It allows even non-technical managers to do data exploration themselves. I don't have complaints about them.

They are excellent tools to perform analysis on read-only datasets. But in large data science project, you'll have to perform complex actions. For instance, you have to trigger a backend function and start the model retraining.

In such cases, my best solution was to build a web app from scratch. **JavaScript data visualization libraries** such as HighCharts are excellent tools for this. They have callbacks for almost every possible user action. I use them to send data back to the server and control it better.

But this wasn't a walk in the park. My data science team is exceptional in Python and R but not in *JavaScript*. *N*ot on web frameworks such as *Django* either. And that's not enough; to build modern web apps, you need *frontend web frameworks such as React*.

As we progressed, we realized the harsh truth. Every new technology in our stack inflates the difficulty exponentially.

And we were fortunate to find Dash.

If you're looking for a lightweight alternative, check out ***Streamlit***. Read along if you need a flexible, complete Python dashboarding solution.

Now, this is how I create dazzling dashboards in Python.

Building your first dashboard in Python (in less than 1 minute.)

Yes, building dashboards in Dash is that simple. Install Pandas and dash with the following command, then start the timer.

```
pip install dash pandas
```

In your project directory, create a file called app.py with the below content.

```python
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px
import pandas as pd

app = dash.Dash()

df = pd.read_csv(
    "https://raw.githubusercontent.com/ThuwarakeshM/geting-started-with-plottly-dash/mai
)

fig = px.scatter(
    df,
    x="GDP",
    y="Life expectancy",
    size="Population",
    color="continent",
    hover_name="Country",
    log_x=True,
    size_max=60,
)

app.layout = html.Div([dcc.Graph(id="life-exp-vs-gdp", figure=fig)])


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

initial.py hosted with ❤️ by GitHub view raw

Simplest Plotly Dash app.

Showtime! let's run the dashboard with the following command:

```
python app.py
```

You'll see it starting a server at port 8050. If you visit http://127.0.0.1:8050 on your browser, you'd see the dashboard that looks like the following:

Dashboard created in Python only. — by the Author

If you've created a similar app using JavaScript libraries, you'd appreciate the difference. Dash saves a ton of time by eliminating an incredible amount of boilerplate code. Even popular BI tools have lots of prework to get to this point.

Awesome. That's quite a thing for inspiring you to build dashboards. But you might have realized that it's not dazzling yet. In the following sections, we'll discuss how

- we can improve the layout;
- add interactions and callbacks to the widgets, and;
- style the app further.

With this, you can create the dazzling dashboard you need. Browse the Plottly Dash gallery for more of such dashboards for inspiration.

Adding more widgets to the layout.

Dash follows an HTML-like element hierarchy. You can attach any of Dash's HTML components or Core components to the layout property of the app. The layout property is the root of a Dash app's element hierarchy.

Core components are a pre-configured set of widgets such as dropdowns and sliders.

Dash's HTML components cover almost every HTML element available. To create a heading, you can use `html.H1` and `html.P` to create a paragraph. The children attribute allows you to nest one HTML component within another.

```
app.layout = html.Div(
    [
        # Dropdown to filter developing/developed country.
        html.Div(
            [
                dcc.Dropdown(
                    id="status-dropdown",
                    options=[{"label": s, "value": s} for s in df.Status.unique()], # Cr
                ),
            ]
        ),
        # Dropdown to filter countries with average schooling years.
        html.Div(
            [
                dcc.Dropdown(
                    id="schooling-dropdown",
                    options=[
                        {"label": y, "value": y}
                        for y in range(
                            int(df.Schooling.min()), int(df.Schooling.max()) + 1
                        )
                    ], # add options from the dataset.
                ),
            ]
```

```
        ),
        # Placeholder to render teh chart.
        html.Div(dcc.Graph(id="life-exp-vs-gdp"), className="chart"),

        # Slider to select year.
        dcc.Slider(
            "year-slider",
            min=df.Year.min(), # dynamically select minimum and maximum years from the d
            max=df.Year.max(),
            step=None,
            marks={year: str(year) for year in range(df.Year.min(), df.Year.max() + 1)},
            value=df.Year.min(),
        ),
    ],
)
```

layout.py hosted with ❤️ by GitHub view raw

Using components in a Plotly Dash app. — by the Author

In the above code, we've included three core components — two dropdowns and a slider. These controller elements allow us to filter the chart data and create interactivity in the next section.

Adding interactivity with component callbacks.

Dash's core components have callbacks to control the response for a user action. This feature is remarkable of why Dash apps outshine popular BI platforms.

You can use this call back to control a chart re-rendering or to trigger a heavy analysis too. Check out my article on performing massive computation to use Dash apps along with Celery.

Here in this post, we use callbacks to filter the chart data.

```
@app.callback(
    Output("life-exp-vs-gdp", "figure"),
    Input("year-slider", "value"),
    Input("status-dropdown", "value"),
```

```
    Input("schooling-dropdown", "value"),
)
def update_figure(selected_year, country_status, schooling):
    filtered_dataset = df[(df.Year == selected_year)]

    if schooling:
        filtered_dataset = filtered_dataset[filtered_dataset.Schooling <= schooling]

    if country_status:
        filtered_dataset = filtered_dataset[filtered_dataset.Status == country_status]

    fig = px.scatter(
        filtered_dataset,
        x="GDP",
        y="Life expectancy",
        size="Population",
        color="continent",
        hover_name="Country",
        log_x=True,
        size_max=60,
    )

    return fig
```

callback.py hosted with ❤️ by GitHub view raw

Using callbacks in a Plotly Dash app. — by the Author

The callback function is annotated with the @app.callback decorator. The first argument of this decorator is the Output component in the element tree. We need to specify the id of that element and the property we need to change. This property will change to the return value of the callback function.

Then the decorated will accept any number of input arguments. Each will be tied to a core component in the same way we attached the output component. We can specify the id of the element and the property that emits the change value. Usually, this would be 'value.'

Each input in the decorator should have a respective argument in the callback function's definition.

Finally, we moved the figure component inside the callback function. Every time we run the callback function, it creates a new figure instance and updates the UI.

Styling to the dashboard.

You can use the inline styling options available in Dash app. But with little CSS, you could have spectacular results.

In-dash, you can style elements in three different ways.

Inline styling

Every Dash component accepts a style argument. You can pass a dictionary and style any element. This is the most convenient way to style a Dash app.

```
1   html.H1("My Dazzling Dashboard", style={"color": "#011833"}),
```

inline_style.py hosted with ❤️ by GitHub view raw

Inline styling of Dash app component. — by the Author

Local stylesheets.

Alternatively, you can pass a class name attribute to any Dash component and use a separate CSS file to style it. You should place this CSS file inside an asset folder in your project directory. Dash will automatically pick it and apply its styles to the components.

```
1   # - app.py
2   # - assets/
3   #     |-- style.css
4
5   # style.css
6   # ----------------------------------------
7   # .title { color: #011833 }
8
9   # app.py
10  html.H1("My Dazzling Dashboard", className='title')
11
12
13
```

local_stylesheet.py hosted with ❤️ by GitHub view raw

Local stylesheets in Plotly Dash app. — by the Author

External stylesheets.

You can also use stylesheets from the internet. For instance, dash has this preconfigured stylesheet that comes with convenient utility classes. You can specify the style sheet and use its class names in elements to make them beautiful.

```
app = dash.Dash(
    __name__, external_stylesheets="https://codepen.io/chriddyp/pen/bWLwgP.css"
)

# Alternative way
app.css.append_css({
    "external_url": "https://codepen.io/chriddyp/pen/bWLwgP.css"
})
```

ext_stylesheet.py hosted with ❤️ by GitHub view raw

External stylesheets in Plotly Dash app. — by the Author

Here is the complete styled source code of the application. We've used a local stylesheet and organized the HTML in a way to support styling. You can find the complete code and the local stylesheet in this GitHub repository.

```
import dash
import dash_core_components as dcc
import dash_html_components as html
from dash_html_components.Label import Label
from pandas.io.formats import style
import plotly.express as px
import pandas as pd
from dash.dependencies import Input, Output

app = dash.Dash(
    __name__,
)

df = pd.read_csv(
    "https://raw.githubusercontent.com/ThuwarakeshM/geting-started-with-plottly-dash/ma
)
```

```
17
18  colors = {"background": "#011833", "text": "#7FDBFF"}
19
20  app.layout = html.Div(
21      [
22          html.H1(
23              "My Dazzling Dashboard",
24          ),
25          html.Div(
26              [
27                  html.Div(
28                      [
29                          html.Label("Developing Status of the Country"),
30                          dcc.Dropdown(
31                              id="status-dropdown",
32                              options=[
33                                  {"label": s, "value": s} for s in df.Status.unique()
34                              ],
35                              className="dropdown",
36                          ),
37                      ]
38                  ),
39                  html.Div(
40                      [
41                          html.Label("Average schooling years grater than"),
42                          dcc.Dropdown(
43                              id="schooling-dropdown",
44                              options=[
45                                  {"label": y, "value": y}
46                                  for y in range(
47                                      int(df.Schooling.min()), int(df.Schooling.max()) +
48                                  )
49                              ],
50                              className="dropdown",
51                          ),
52                      ]
53                  ),
54              ],
55              className="row",
56          ),
57          html.Div(dcc.Graph(id="life-exp-vs-gdp"), className="chart"),
58          dcc.Slider(
59              "year-slider",
60              min=df.Year.min(),
61              max=df.Year.max()
```

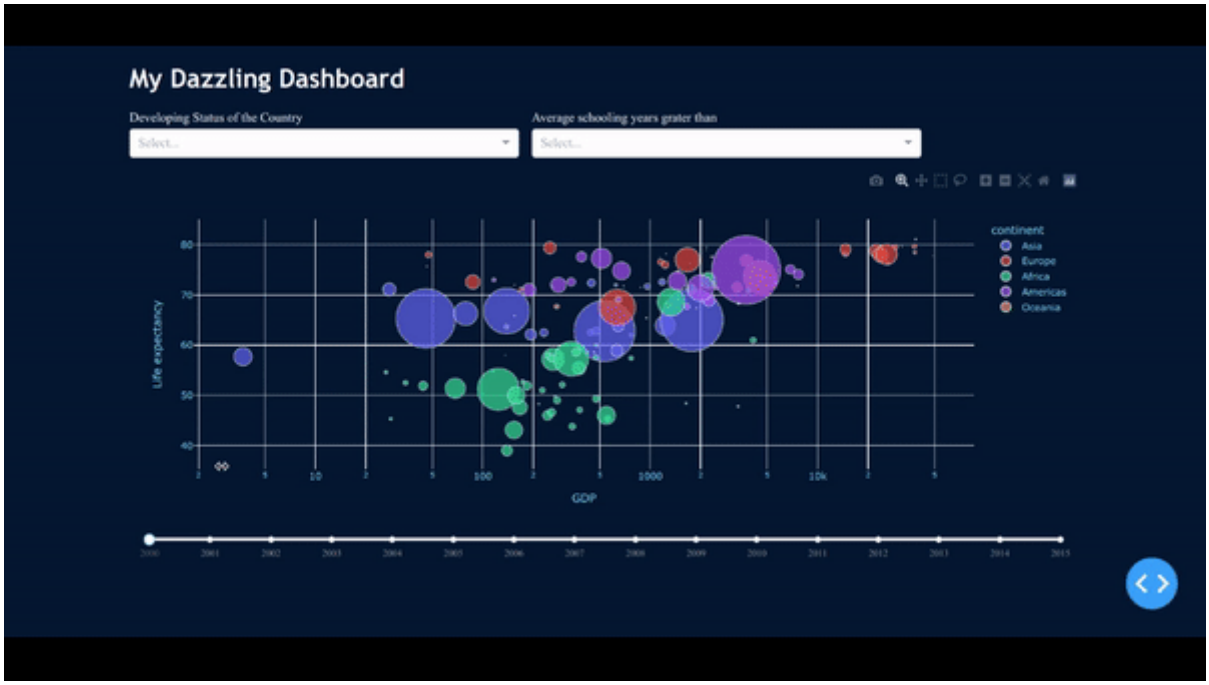
```
            max=df.Year.max(),
            step=None,
            marks={year: str(year) for year in range(df.Year.min(), df.Year.max() + 1)},
            value=df.Year.min(),
        ),
    ],
    className="container",
)


@app.callback(
    Output("life-exp-vs-gdp", "figure"),
    Input("year-slider", "value"),
    Input("status-dropdown", "value"),
    Input("schooling-dropdown", "value"),
)
def update_figure(selected_year, country_status, schooling):
    filtered_dataset = df[(df.Year == selected_year)]

    if schooling:
        filtered_dataset = filtered_dataset[filtered_dataset.Schooling <= schooling]

    if country_status:
        filtered_dataset = filtered_dataset[filtered_dataset.Status == country_status]

    fig = px.scatter(
        filtered_dataset,
        x="GDP",
        y="Life expectancy",
        size="Population",
        color="continent",
        hover_name="Country",
        log_x=True,
        size_max=60,
    )

    fig.update_layout(
        plot_bgcolor=colors["background"],
        paper_bgcolor=colors["background"],
        font_color=colors["text"],
    )

    return fig


```

```
106  if __name__ == "__main__":
107      app.run_server(debug=True)
```

app.py hosted with ❤️ by GitHub view raw

Complete source code of a Dashboard created in Python. by the Author

Your web app refreshes as you update your code with the above. And it may look like the below — your first version of a dazzling dashboard.



Screencast of a basic styled Dashboard created in Python. — By the Author.

Final Thoughts

plotly, Dash apps are an incredible tool for Python developers. Since most data science teams are not specializing in JavaScript, building dashboards with Dash saves a ton of their time.

We can use Tableau, PowerBI, and similar BI platforms for data exploration and visualization. But Dash apps outshine them as they tightly integrate with the backend code.