Overview

Day	2
Name	Practice #2 : Exploration for others : dashboard with dash
Skills	Reflexivity, create
Time	1h *(finishing in this time line is not required nor graded)*
Submission	No submission

Practice

Preliminary

tools info

Plotly

The plotly Python library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases.

Built on top of the Plotly JavaScript library (plotly.js), plotly enables Python users to create beautiful interactive web-based visualizations that can be displayed in Jupyter notebooks, saved to standalone HTML files, or served as part of pure Python-built web applications using Dash. The plotly Python library is sometimes referred to as "plotly.py" to differentiate it from the JavaScript library.

→ Plotly Python Open Source Graphing Library

Dash

Dash is a productive Python framework for building web applications. Written on top of Flask, Plotly.js, and React.js, Dash is ideal for building data visualization apps with highly custom user interfaces in pure Python. It's particularly suited for anyone who works with data in Python.

Through a couple of simple patterns, Dash abstracts away all of the technologies and protocols that are required to build an interactive web-based application. Dash is simple enough that you can bind a user interface around your Python code in an afternoon.

Dash apps are rendered in the web browser. You can deploy your apps to servers and then share them through URLs. Since Dash apps are viewed in the web browser, Dash is inherently cross-platform and mobile ready.

→ Starting with Dash

environment setup

It would be easier here to use your local environment :

- pip install dash==1.16.0

Dashboard making

Step 1 : Dash discovery

Use the file "dash_app_demo.py"

Let's test your Dash installation. Download all files from the practice folder if you did not yet. In the corresponding folder run : *python dash_app_demo.py*

Try the hot reload feature : try to change the Layout : text, add elements in the code, save, and see if the page reload by itself (you can see examples in the documentation) :

- Add a style option to the title (= the html.H1 component)

Step 2 : Setting up a simple real example from our data

Use the file "dash_starter.py"

- Choose a graph that you made in the Data Exploration part, and that is based on a specific year's data

- Add this graph in the layout part :
  - → check how the previous example was made : what parts do you need to make a graph ?
  - → Create a figure object as the previous exemple, and add its value : copy paste from the notebook the chart you choose

    fig = px.x...
  - → Add it in the layout, as the previous example

    dcc.Graph ...

- Create a function that allow to plot this graph taking the year as an argument, following those steps :
  1. Create a function that returns the same figure of the graph you added *(see how the generate_dashtable function is made and reuse the same structure. Here you want to return a chart created with plotly)*
  2. In the layout, instead of using the graph you created with **fig=xxx**, use this function to build the chart. Test if it works.

3. In the function, add an argument (the year) and update the function to take it into account

   `df_merged.loc[df_merged['year'].isin([year])]`

4. In the layout, update the function adding a year as an argument. Test if it works.

- Add a third graph in the layout using this function that will display the same graph for another year

Step 3 : Getting to know callbacks

Use the file "dash_basic_callback.py"

- In the dashboard, in the first part “1/ Callback demo”, try to change the value and see how it responds

- Have a look at the code, and make a schema with a pen to understand the differents parts, elements and ids for the callback to work

- You will now create a new callback : add a text below the slider that is printing the selected years :
  1. Add a div that will contain our new output (see the example of **my-output**)
  2. Create a new callback with the same template as the example. See all the elements you need to update for it to work

- Now we want to add the previous chart *(and its function we defined before)*, to take as an argument the selected year with the slider. Let’s follow the following steps :
  1. Copy and paste the function from the last example in the “# Functions definitions” part
  2. In the layout, below the slider, add a chart built using this function (as in last example)
  3. Now, let’s update the callback to take into account that we have 2 outputs of the slider (the text and now the chart)
     a. Add an Output line with the chart id

        `Output(`***id***`, 'figure')`

     b. Add another result in the callback function

        `def function(input_value):`

*return **value1**, **value2***

To go further, you can see many more examples in the Dash documentation and dashboard examples.