PyData
Delhi 2018

Dash : Interactive Web Visualization Apps natively in Python

Madhur Tandon

Github: @madhur-tandon

Dash what?

- A Python Framework for building web applications
- Created by Plotly
- Built on top of React.js, Plotly.js (of course!) and Flask
- Dashboard Implemented in Pure Python (unless you need customized components)
- Open Source <3 (MIT License)

HTML & Core Components

- Dash Apps composed of 2 parts - layout (how it looks) + interactivity of the app

- *dash_html_components* library
  - provides classes for HTML tags and their arguments such as style, className, id

- *dash_core_components* library
  - describes higher-level components that are interactive and are generated with JavaScript, HTML, and CSS through the React.js library
  - Also has Plotly python API available through the Graph class

Installation

```
pip install dash  # The core dash backend
pip install dash-renderer  # The dash front-end
pip install dash-html-components  # HTML components
pip install dash-core-components  # Supercharged components
pip install plotly --upgrade  # Plotly graphing library
```

Basic Imports

```
import dash
import dash_core_components as dcc
import dash_html_components as html

app = dash.Dash()
```

```
app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for Python.
    '''),

    dcc.Graph(
        id='example-graph',
        figure={
            'data': [
                {'x': [1, 2, 3], 'y': [4, 1, 2], 'type': 'bar', 'name': 'SF'},
                {'x': [1, 2, 3], 'y': [2, 4, 5], 'type': 'bar', 'name': u'Montréal'},
            ],
            'layout': {
                'title': 'Dash Data Visualization'
            }
        }
    )
])
```

The layout of the app contains a main *Div*

This *Div* contains 3 children which are

- a H1 tag
- Another *Div*
- An interactive Graph

The first 2 are HTML components and generate HTML tags behind the back.

Thus, html.H1(children='Hello Dash') generates <h1>Hello Dash</h1> in the application.

The graph component is used to make an interactive graph automatically based on the underlying Plotly library. Supports options of zooming, panning, comparing data-points, etc [More on this later]

Diggin’ Deeper with HTML Components

```
colors = {
    'background': '#111111', //(BLACK)
    'text': '#7FDBFF'           //(BLUE)
}
```

```
app.layout =
html.Div(children=[
    html.H1(children='Hello Dash'),



html.Div(children='''
        Dash: A web application framework
        for Python.
    '''),
```

```
app.layout =
html.Div(style={'backgroundColor':
colors['background']}, children=[
    html.H1(
        children='Hello Dash',
        style={
            'textAlign': 'center',
            'color': colors['text']
        }
    ),


html.Div(children='Dash: A web application
framework for Python.', style={
        'textAlign': 'center',
        'color': colors['text']
    }),
dcc.Graph(
        id='example-graph',
        figure={
            'data': [
                {'x': [1, 2, 3], 'y': [4, 1, 2], 'type': 'bar',
'name': 'SF'},
                {'x': [1, 2, 3], 'y': [2, 4, 5], 'type': 'bar',
'name': u'Montréal'},
            ],
            'layout': {
                'title': 'Dash Data Visualization'
            }
        }
    )
```

```
dcc.Graph(
        id='example-graph-2',
        figure={
            'data': [
                {'x': [1, 2, 3], 'y': [4, 1, 2], 'type': 'bar',
'name': 'SF'},
                {'x': [1, 2, 3], 'y': [2, 4, 5], 'type': 'bar',
'name': u'Montréal'},
            ],
            'layout': {
                'title': 'Dash Data Visualization',
                'plot_bgcolor': colors['background'],
                'paper_bgcolor': colors['background'],
                'font': {
                    'color': colors['text']
                }
            }
        }
    )
```

Key Takeaways for HTML Components

```
html.H1('Hello Dash', style={'textAlign': 'center', 'color': '#7FDFF'})
```

generates

```
<h1 style="text-align: center; color: #7FDFF">Hello Dash</h1>
```

- The style property in HTML is a semicolon-separated string. In Dash, you can just supply a dictionary.
- The keys in the style dictionary are camelCased. So, instead of text-align, it's textAlign
- The HTML class attribute is className in Dash.
- The children of the HTML tag is specified through the children keyword argument. By convention, this is always the *first* argument and so it is often omitted.

Core Components (DCC)

dash_core_components includes a set of higher-level components like dropdowns, graphs, markdown blocks, and more.

- Dropdowns
- Multi-Select Dropdowns
- Radio Items
- Checkboxes
- Text Input
- Slider
- GRAPH

```python
html.Label('Dropdown'),
    dcc.Dropdown(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': u'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value='MTL'
    ),

html.Label('Radio Items'),
    dcc.RadioItems(
        options=[
            {'label': 'New York City', 'value': 'NYC'},
            {'label': u'Montréal', 'value': 'MTL'},
            {'label': 'San Francisco', 'value': 'SF'}
        ],
        value='MTL'
    ),
```

```python
html.Label('Text Input'),
    dcc.Input(value='MTL', type='text'),

html.Label('Slider'),
    dcc.Slider(
        min=0,
        max=9,
        marks={i: 'Label {}'.format(i) if i == 1 else str(i)
for i in range(1, 6)},
        value=5,
    ),
```

Visualization with DCC - Graph

Graph renders interactive data visualizations using the open source plotly.js JavaScript graphing library. Plotly.js supports over 35 chart types and renders charts in both vector-quality SVG and high-performance WebGL.

figure argument in Graph component is same as the figure argument in plotly.py

Let’s make a Scatter Plot

```
import dash
import dash_core_components  as dcc
import dash_html_components  as html
import pandas as pd
import plotly.graph_objs as go

app = dash.Dash()
```

```
df = pd.read_csv(
    'https://gist.githubusercontent.com/chriddyp/' +
    '5d1ea79569ed194d432e56108a04d188/raw/' +
    'a9f9e8076b837d541398e999dcbac2b2826a81f8/'+
    'gdp-life-exp-2007.csv')
```

```
app.layout = html.Div([
    dcc.Graph(
        id='life-exp-vs-gdp',
        figure={
            'data': [
                go.Scatter(
                    x=df[df['continent'] == i]['gdp per capita'],
                    y=df[df['continent'] == i]['life expectancy'],
                    text=df[df['continent'] == i]['country'],
                    mode='markers',
                    opacity=0.7,
                    marker={
                        'size': 15,
                        'line': {'width': 0.5, 'color': 'white'}
                    },
                    name=i
                ) for i in df.continent.unique()
            ],
            'layout': go.Layout(
                xaxis={'type': 'log', 'title': 'GDP Per Capita'},
                yaxis={'title': 'Life Expectancy'},
                margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
                legend={'x': 0, 'y': 1},
                hovermode='closest'
            )
        }
    )
])
```

Callbacks - Adding Interactivity

```
import dash
import dash_core_components  as dcc
import dash_html_components  as html
from dash.dependencies  import Input, Output

app = dash.Dash()

app.layout = html.Div([
    dcc.Input(id='my-id', value='initial value', type='text'),
    html.Div(id='my-div')
])
```

```
@app.callback(
    Output(component_id='my-div', component_property='children'),
    [Input(component_id='my-id', component_property='value')]
)
def update_output_div(input_value):
    return 'You\'ve entered "{}"'.format(input_value)
```

- our input is the "value" property of the component that has the ID "my-id". Our output is the "children" property of the component with the ID "my-div”

States

- Just like **dash.dependencies.Inputs**
- Previously, callback function was fired whenever any of the attributes described by the **dash.dependencies.Input** changed
- **dash.dependencies.State** allows you to pass along extra values without firing the callbacks
- Useful for UIs having form or buttons

```
app.layout = html.Div([
    dcc.Input(id='input-1-state', type='text', value='Montréal'),
    dcc.Input(id='input-2-state', type='text', value='Canada'),
    html.Button(id='submit-button', n_clicks=0, children='Submit'),
    html.Div(id='output-state')
])


@app.callback(Output('output-state', 'children'),
             [Input('submit-button', 'n_clicks')],
             [State('input-1-state', 'value'),
              State('input-2-state', 'value')])
def update_output(n_clicks, input1, input2):
    return u'''
        The Button has been pressed {} times,
        Input 1 is "{}",
        and Input 2 is "{}"
    '''.format(n_clicks, input1, input2)
```

- changing text in the **dcc.Input** boxes won't fire the callback but clicking on the button will. The current values of the **dcc.Input** values are still passed into the callback even though they don't trigger the callback function itself.
- we're triggering the callback by listening to the **n_clicks** property of the **html.Button** component.

Variations

- Any "Output" can have multiple "Input" components.
- Each Dash callback function can only update a single Output property. To update multiple Outputs, just write multiple functions.
- Can also chain outputs and inputs together.
    - The output of one callback function could be the input of another callback function.
    - This pattern can be used to create dynamic UIs where one input component updates the available options of the next input component.

Example: Slider with a Graph

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import pandas as pd
import plotly.graph_objs as go

df = pd.read_csv(
    'https://raw.githubusercontent.com/plotly/'
    'datasets/master/gapminderDataFiveYear.csv')

app = dash.Dash()
```

- Loading data into memory can be expensive.
- By loading querying data at the start of the app instead of inside the callback functions, we ensure that this operation is only done when the app server starts.
- When a user visits the app or interacts with the app, that data (the df) is already in memory.

```
app.layout = html.Div([
    dcc.Graph(id='graph-with-slider'),
    dcc.Slider(
        id='year-slider',
        min=df['year'].min(),
        max=df['year'].max(),
        value=df['year'].min(),
        step=None,
        marks={str(year): str(year) for year in df['year'].unique()}
    )
])
```

- A Slider is defined with years ranging from minimum to maximum in the dataset.

```
@app.callback(
    dash.dependencies.Output('graph-with-slider', 'figure'),
    [dash.dependencies.Input('year-slider', 'value')])
def update_figure(selected_year):
    filtered_df = df[df.year == selected_year]
    traces = []
    for i in filtered_df.continent.unique():
        df_by_continent = filtered_df[filtered_df['continent'] == i]
        traces.append(go.Scatter(
            x=df_by_continent['gdpPercap'],
            y=df_by_continent['lifeExp'],
            text=df_by_continent['country'],
            mode='markers',
            opacity=0.7,
            marker={
                'size': 15,
                'line': {'width': 0.5, 'color': 'white'}
            },
            name=i
        ))
```

- We load our dataframe at the start of the app. This data-frame df is in the global state of the app and can be read inside the callback functions.
- the "value" property of the Slider is the input of the app and the output of the app is the "figure" property of the Graph.
- Whenever the value of the Slider changes, Dash calls the callback function update_figure with the new value.

```
return {
        'data': traces,
        'layout': go.Layout(
            xaxis={'type': 'log', 'title': 'GDP Per Capita'},
            yaxis={'title': 'Life Expectancy', 'range': [20, 90]},
            margin={'l': 40, 'b': 40, 't': 10, 'r': 10},
            legend={'x': 0, 'y': 1},
            hovermode='closest'
        )
    }
```

- The function filters the data-frame with this new value, constructs a figure object, and returns it to the Dash application.

Multiple Inputs

```
@app.callback(
    dash.dependencies.Output('indicator-graphic', 'figure'),
    [dash.dependencies.Input('xaxis-column', 'value'),
     dash.dependencies.Input('yaxis-column', 'value'),
     dash.dependencies.Input('xaxis-type', 'value'),
     dash.dependencies.Input('yaxis-type', 'value'),
     dash.dependencies.Input('year--slider', 'value')])
def update_graph(xaxis_column_name, yaxis_column_name,
                 xaxis_type, yaxis_type,
                 year_value):
```

- update_graph function gets called whenever the value property of the **Dropdown**, **Slider**, or **RadioItems** components change.

- The input arguments of the **update_graph** function are the new or current value of the each of the Input properties, in the order that they were specified.

- callback functions are always guaranteed to be passed the representative state of the app.

Fin.

Questions

Please email at

madhurtandon23@gmail.com