Plotly Dashboard

참고 레퍼런스

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1cfe69ba-48b0-4869-a97a-026b6411dd02/pdfcoffee.com_python-dashboards-pdf-free.pdf

Interactive Dashboard and Data Apps with Plotly and Dash

Jupyter 대시보드 생성 참고

JupyterDash 소개

Jupyter 환경 (예 : 클래식 Notebook , JupyterLab , Visual Studio Code 노트북 , nteract , PyCharm 노트북 등)에서 Dash 앱을 쉽게 빌드 할 수있는 새로운 라이브러리 인 JupyterDash 출시를 발표하게되어 기쁩니다 . Dash는 순수 Python (JavaScript 필요 없음)을 사용하여 전체 스택 분석 웹 애플리케이션을 빌드하기위한 Plotly의 오픈

https://ichi.pro/ko/jupyterdash-sogae-141970669813806

Dash_Installation

```
pip install dash==0.21.0                ## core dash backend
pip install dash-renderer==0.11.3       ##dash front-end
pip install dash-html-components==0.9.0   ##HTML components
pip install dash-core-components==0.21.2  ## supercharged components
pip install plotly --upgrade            ## plotly graphing library
```

- **Dash**는 앱의 backbone을 제공하는 메인 패키지로, interactivity와 exception을 관리하는 몇가지 툴을 제공
- **Dash-html-components**는 html tag를 Python class로 쉽게 사용할 수 있도록 만들어줌. Dashoboard의 페이지 layout을 담당

  ex) html.H1(children='Hello Dash')는 <h1>Hello Dash></h1>과 동일함
- **dash-core-components**는 JavaScript, HTML, React.js의 CSS에서 제공하는 것과 같은 높은 수준의 우저 인터랙티브 요소 (ex. Dropdown, Date Picker, Slider ...)를 제공. Dashboard의 개별 그래프를 담당
- dash-bootstrap-components는 레이아웃과 시각적 요소와 연관된 다양한 기능을 제공

Dash의 구성

Dash app의 일반적인 구조

- imports (boilerplate)
- app instantiation : 직관적으로 app을 만드는 과정에 해당, '__name__' 파라미터는 현재 만들어진 app에서 static asset(정적 자산?)을 위치시킴 - static asset은 java에서 static resource와 비슷한건가? static resource는 이미 만들어져있는 자원을 유저가 조

회할 때마다 새로 불러오지 않고 이미 만들어진 걸 바로 보여준다는 의미

```
app = dash.Dash(__name__)
```

- app layout : html.Div 나 children 파라미터를 이용해 유저가 보는 요소들을 만들어 냄

```
app.layout = html([
    dcc.Dropdown()
    dcc.Graph()
     ...
])
```

- callback functions: 코드 안의 다양한 기능들이 순서 상관없이 상호작용될 수 있도록 도와줌

```
@app.callback()
...
@app.callback()
...
```

- running the app

```
if __name__ == '__main__':
  app.run_server()
```

HTML Component

자주 쓰이는 Dash HTML Component

- children: component의 내용을 담는 첫번째 메인 컨테이너
- className: class와 동일한 기능 수행
- id: component마다 여러개의 id를 설정하여 관리하기 쉽게 만들어주는, 유저 인터랙션을 활용하는 데 중요한 요소
- style: html의 Style과 비슷하지만 Dictionary로 속성을 관리하는 것과 CamelCase로 쓰인다는 점에서 차이

html 의 기본적인 레이아웃에 대해 학습

HTML Tutorial

With our "Try it Yourself" editor, you can edit the HTML code and view the result: Click on the "Try it Yourself" button to see how it works. In this HTML tutorial, you will find more than 200 examples. With our online "Try it Yourself" editor, you can edit and test each example yourself!

https://www.w3schools.com/html/

Bootstrap을 이용해 Layout과 Theme 바꾸기

- Theme: Bootstrap으로 바꿀 수 있는 전체적인 컴포넌트 모양
- Grid system: 사용자 관점의 픽셀, 해상도 등에 따라 보이는 모양 변경
- Responsiveness: 사용자의 스크린 사이즈에 맞는 화면 제시
- Prebuilt components: Bootstrap에서 drop-down, tab 메뉴 등이 이미 만들어져있음
- Encoded colors: 컴퓨터 색상표 말고 속성 목적(danger, warning)을 색상으로 활용가능

Theme 지정

```
import dash_bootstrap_components as dbc
```

```
app.dash.Dash(__name__, external_stylesheets = [dbc.themes.BOOTSTRAP])
```

Grid System and responsiveness

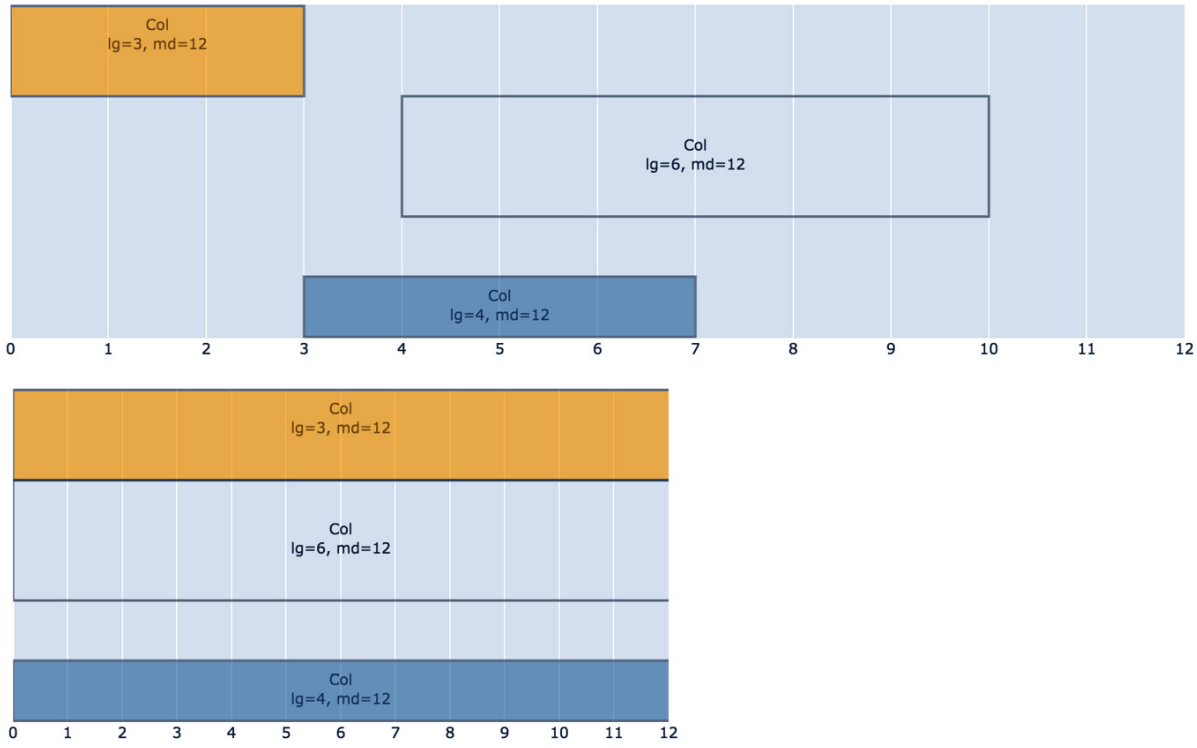
html.Div에 children parameter를 추가해서 내용(item)을 덧붙이면 모든 item이 화면에 꽉 차게 full size로 보여짐. 또 element의 순서 그대로 스크린에 보여짐

→ Bootstrap을 사용하면 스크린 화면이 줄어들면 item도 알아서 화면에서 차지하는 비율이 줄어듦

→ screen 크기에 따라 proportion이 자동으로 줄어들면 비효율적인 공간(원래 차지하는 공간이 작았던 화면은 불필요하게 작게 줄어들어 가독성이 떨어질 수 있음)이 생기는데, Bootstrap에서는 5가지 화면 너비를 구별해서(각각 xs, sm, md, lg, xl) 원하는 상황의 아이템 너비를 각각의 파라미터에 지정할 수 있음

→ 아이템 내용을 왼쪽여백에 붙여서 시작하지 않고 가운데나 특정 길이부터 시작할 수 있음

```
import dash_bootstrap_components as dbc

dbc.Col(children = [child1, child2, ...], lg=6, md=12)
## large 크기일 때 width, medium일 때 width 12

dbc.Col(children = [child1, child2, ...], lg = ['size': 6, 'offset':4}, md = 12)
## 아래의 빈 컬럼2의 예시에 해당. offset=4는 4번 눈금부터 배치하라는 의미
```



여러개의 아이템을 가로로 나란히 배열하고 싶다면 배열할 column item들을 하나의 리스트로 두고 Row element의 children 속성으로 보내면 된다.

```
import dash_bootstrap_components as dbc

dbc.Row([
  dbc.Col('Column1', width = 2),
  dbc.Col('Column2', width = 5),
  dbc.Col('Column3', width = 4),
])
```

Prebuilt components

https://dash-bootstrap-components.opensource.faculty.ai/ 페이지를 참고하여 이미 만들어져있는 드롭박스, 메뉴, toast 양식 등을 사용할 수 있음

Bootstrap 이용 예시

Encoded colors

color = "danger", color = "warning" 등으로 목적에 맡게 매칭된 컬러를 활용 가능. 가능한 컬러 이름: primary, secondary, success, warning, danger, info, light, dark

Bootstrap 컴포넌트를 사용할 때는 refactoring에 신경을 쓰고 코드를 작성해야 함. refactoring(새로운 버전이 나왔을 때 코드를 이전 버전에서 새로운 버전으로 바꾸는 것을 의미하는 듯) 따라서 리팩토링을 생각하여 copy and paster를 하지 말고 매뉴얼적으로 리팩토링 디테일을 챙길 수 있게 신경쓰기

Dash App 구조 탐색

아래 내용을 기반으로 Jupyter notebook을 이용한 interactive app을 만드는 방법을 공부

- ID parameter 이해하기
- Dash - input, output 이용하기
- Callback function

Jupyter notebook(or Jupyter lab)에서 Dash 이용하기

- Jupyter_dash 설치하기
- Jupyter_dash는 코드를 서버에 돌릴 때 다음 세 모드를 추가로 제공
  - **external**: pycharm에서 하던 것처럼 새 웹페이지로 코드 결과를 보여줌
  - **inline**: jupyter의 코드 셀 바로 아래에 코드 결과페이지를 보여줌
  - **jupyterlab**: Jupyter Lab에서 실행시 새로운 탭으로 결과를 보여줌
- 결과의 가로세로 길이를 지정할 수 있음

```
from jupyter_dash import JupyterDash
app = JupyterDash(__name__)

##결과창의 가로세로 길이 지정
app.run_server(mode = 'inline', height = 600, width = '80%')
```

디버깅을 위한 기능 function 나누기

디버깅을 쉽게 하려면 코드 기능의 feature들을 따로따로 두고, 구동할 때만 합치는 것이 좋다.

```
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html

## Input을 위한 Dropdown 메뉴
app = JupyterDash(__name__)

app.layout = html.Div([
    dcc.Dropdown(options=[{'label':color, 'value':color}
                          for color in ['blue','green', 'yellow']]),
    html.Div() ## 이곳에 Output 함수를 넣음
])
if __name__=='__main__':
    app.run_server(mode = 'inline')
```

- 위 코드는 dropdown 컴포넌트에서 input item을 받고, 두번째 html.Div()에는 Output을 위한 함수를 넣을 수 있다

- 이렇게 Standalone한 코드로 분할하여 잘게 쪼개놓으면 디버그나 추가수정에 용이

```
## Output을 위한 Standalone code
def display_selected_color(color):
    if color is None:
        color = 'nothing'
    return 'You selected' + color
```

ID Parameter

- id parameter는 모든 dash component에 붙일 수 있는 일종의 이름으로 unique identifier임
- descriptive name을 붙여 쉽게 관리할 수 있도록 하기 위함
- 앱이 가진 interactivity 속성이 증가해 복잡도가 올라가면 id parameter를 반드시 사용하는 것이 좋음
- dash_core_components와 dash_html_components의 속성에 id = '' 를 추가하여 해당 속성에 이름 지정

```
app.layout = html.Div([
    dcc.Dropdown(id = 'color_dropdown',
        options=[{'label':color, 'value':color}
                        for color in ['blue','green', 'yellow']]),
    html.Div(id = 'color_output')
])
```

Dash inputs and outputs

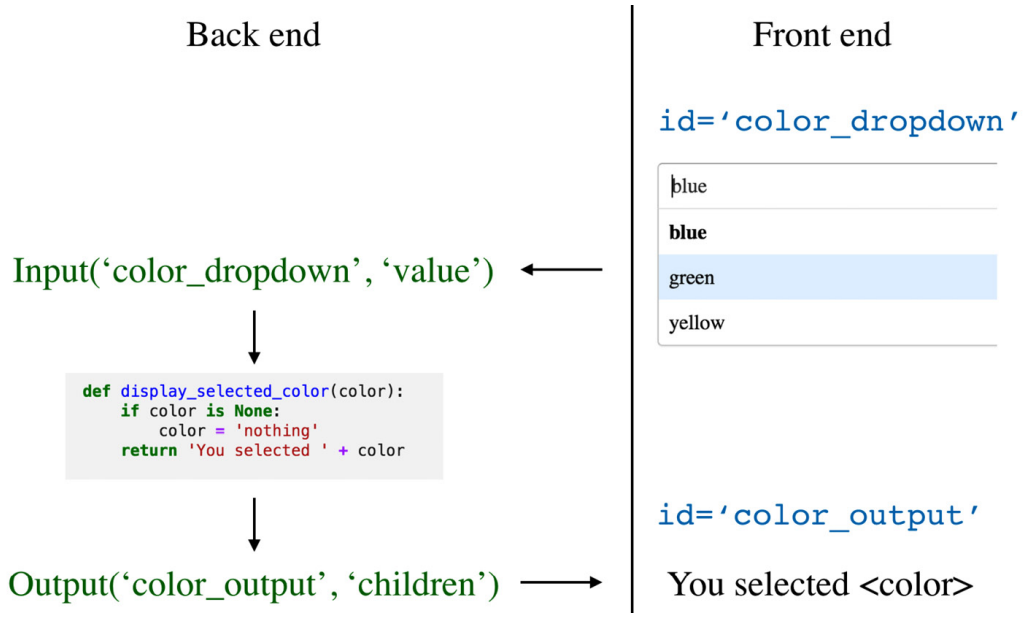
- Input과 Output은 dash.dependencies 패키지를 통해 불러올 수 있음

  (1) Output: 함수 계산 결과로 나오는 요소. Dash는 Property를 이용해 Output에 변화를 줄 수 있음. 현재 예시 코드에는 children에 해당함

  ```
  Output(component_id = 'color_ouput', component_property = 'children')
  ```

  (2) Input: callback function의 input이 될 component를 지정하는 작업이 필요. 현재 예시코드에는 value에 해당함

  ```
  Input(component_id = 'color_dropdown', component_property = 'value')
  ```



Callback Funcion

- 하위 계산 function 순서가 메인함수보다 뒤에 와도 callback을 사용해서 메인함수 결과가 제대로 출력되게 도와주는 듯?

- Callback function을 실행할 때 Output이 Input보다 먼저 나와야 함
- Output에 Div의 children을 지정했다고 하면, Ouput의 children에 다른 value를 지정했을 시 해당 값이 overwritten된다.

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

app = JupyterDash(__name__)

app.layout = html.Div([
    dcc.Dropdown(id = 'color_dropdown',
        options=[{'label':color, 'value':color}
                        for color in ['blue','green', 'yellow']]),
    html.Br(),
    html.Div(id = 'color_output')
])

@app.callback(Output('color_output', 'children'),
              Input('color_dropdown', 'value'))

def display_selected_color(color):
    if color is None:
        color = 'nothing'
    return 'You selected ' + color

if __name__=='__main__':
    app.run_server(mode = 'inline', host='10.200.172.190')
```

- 이 코드 해석이 고비인데.. callback함수에서 Output은 id = 'color_output'인 html.Div 컴포넌트의 children(Div의 내용에 해당)에 해당함. Input은 id = 'color_dropdown'인 dcc.Dropdown 컴포넌트의 value에 해당함
- 따라서 이 코드의 callback 함수를 해석하면 color_dropdown의 value인 '사용자가 선택한' color 변수가 계산된 결과값이 color_ouput의 children으로 들어가라는 소리임
- 내가 궁금한 건 def로 함수를 정의만 했는데 도대체 어떻게..? displayed_selected_color가 실행됐느냐 이다. 아무리봐도.. Output으로 이 결과값이 쏴질 연결고리가 없는데... (→ 내 생각엔 callback함수 자체가 callback + 다음에 오는 함수 연산 → 그 다음에 앞에서 등장했던 Ouput 코드에 쏴줌)
- Callback 함수를 이용해 인터랙티브한 그래프를 생성하는 방법은 아래 페이지를 참조

https://dash.plotly.com/basic-callbacks

- callback 뒤에 여러개의 함수가 있어도 바로 다음에 있는 함수만 연산을 적용해서 결과값을 쏴주는 듯
- 여러개의 Output id + 여러개의 callback을 정의해서 여러개의 인터랙티브 결과값을 한 번에 보여줄 수 있음

```
## 예시코드
@app.callback(Output('output_1', 'children'),
              Output('output_2', 'style'),
              Input('dropdown', 'value'),
              Input('date_picker', 'value'),
              Input('text_area', 'value'))
def my_function(dropdown_value, date_value, text_value):
    print('Hello')
    send_email()
    output_1 = process(dropdown_value, date_value)
    output_2 = process(text_value)

    return output_1, output_2
```

Callback 함수의 속성

- 여러개의 Input이 가능
- 리스트가 Input이 될 수 있음
- 여러개의 Output이 가능함 (한 input으로 테이블과 그래프가 동시에 등장하게 만들 수 있음)

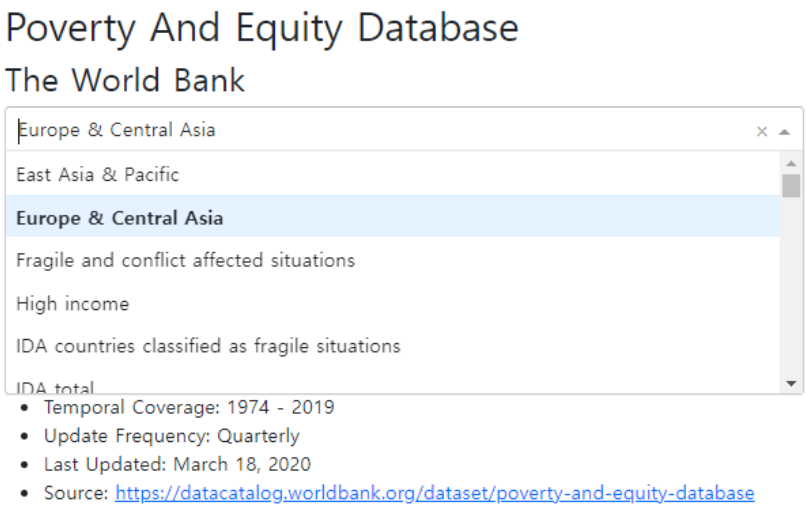
- return 함수 이전의 function 작동을 활용가능 (callback 함수 안에서 log를 따로 저장시켜 사람들이 input으로 어떤 것들을 눌렀는지 데이터를 저장 가능)
- Output 정의 파라미터가 Input 파라미터보다 먼저 나와야 함
- Input의 순서는 callback 함수의 순서를 그대로 따라감
- 'State' 파라미터를 이용해 Input이 바뀔때마다 Output이 바로 바뀌어서 보여지는 불편함을 해결 (Input으로 텍스트가 들어갈 때 텍스트 한 글자가 쓰여질 때마다 Output이 바뀌면 사용성 하락)

```
## 여러개의 Input Output 코드 예시
@app.callback(Output('output_1', 'children'),
              Output('output_2', 'style'),
              Input('dropdown', 'value'),
              Input('date_picker', 'value'),
              Input('text_area', 'value'))
def my_function(dropdown_value, date_value, text_value):
    print('Hello')
    send_email()
    output_1 = process(dropdown_value, date_value)
    output_2 = process(text_value)

    return output_1, output_2
```

callback을 이용해 dataframe과 연동되는 인터랙티브 앱 만들기

```
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output, State

import pandas as pd

app = JupyterDash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])

poverty_data = pd.read_csv('data/PovStatsData.csv')

app.layout = html.Div([
    html.H1('Poverty And Equity Database'),
    html.H2('The World Bank'),
    dcc.Dropdown(id='country',
                 options=[{'label': country, 'value': country}  ## Input 받음
                          for country in data['Country Name'].unique()]),
    html.Br(),
    html.Div(id='report'),  ##Output 출력
    html.Br(),
    dbc.Tabs([
       dbc.Tab([
           html.Ul([
               html.Br(),
               html.Li('Number of Economies: 170'),
               html.Li('Temporal Coverage: 1974 - 2019'),
               html.Li('Update Frequency: Quarterly'),
               html.Li('Last Updated: March 18, 2020'),
               html.Li([
                   'Source: ',
                   html.A('https://datacatalog.worldbank.org/dataset/poverty-and-equity-database',
                          href='https://datacatalog.worldbank.org/dataset/poverty-and-equity-database')
               ])
           ])
        ], label='Project Info')  ## dbc.Tab의 Label 이름
    ]),
])


@app.callback(Output('report', 'children'),
              Input('country', 'value'))
def display_country_report(country):
    if country is None:
        return ''

    filtered_df = data[(data['Country Name']==country) & ## Input으로 들어온 Country 값과 동일하면서
                               (data['Indicator Name']=='Population, total')] ##'Population, total'이 Indicator Name값에 해당하는 data
    population = filtered_df.loc[:, '2010'].values[0]  ## '2010' column의 값을 가져오기

    return [html.H3(country),
            f'The population of {country} in 2010 was {population:,.0f}.']
```

```
if __name__=='__main__':
    app.run_server(mode = 'inline', host='10.200.172.190')
```



Plotly Figure Object

- 사용자가 쉽게 이해할 수 있는 그래프 만들기
- Plotly Figure 오브젝트의 레이아웃과 데이터 조정하는 법

Plotly Figure의 main Attribute

- **데이터**: 데이터의 속성으로 인해 정해지는 그래프의 모양(bar, line, pie 등등)
- **레이아웃**: 그래프의 텍스트 정보(title, axis-title 등), 디자인
- 인터랙티브 요소를 관리하는 **프레임** Attribute도 있지만 main Attribute로 보진 않음

cf) Figure 오브젝트는 전역변수이기 때문에 다른 쉘에서 수정해도 변경사항이 반영됨

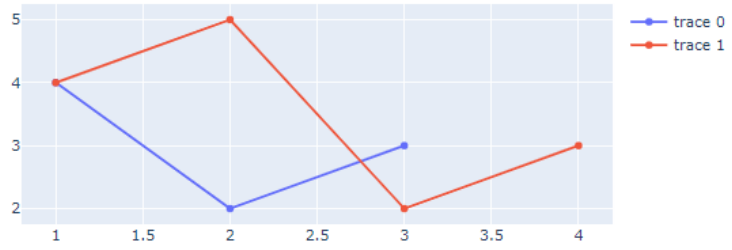
Data Attribute

- 데이터 그래프는 add_** (ex. add_scatter, add_bar) 를 이용해 추가할 수 있음
- x와 y의 값은 list, tuple, numpy array, pandas series 형태로 넣을 수 있음

Layout Attribute

- figure.attribute.sub_attribute = value 형식으로 그래프 조정

```
fig = go.Figure()
fig.add_scatter(x = [1,2,3], y = [4,2,3])  ## adding data attribute
fig.add_scatter(x = [1,2,3,4], y = [4,5,2,3])
fig.show()
```




```
fig.layout.title = 'The Figure Title'  ## adding layout attribute
fig.layout.xaxis.title = 'The X-axis title'
fig.layout.yaxis.title = 'The Y-axis title'
```

```
fig.show()
```

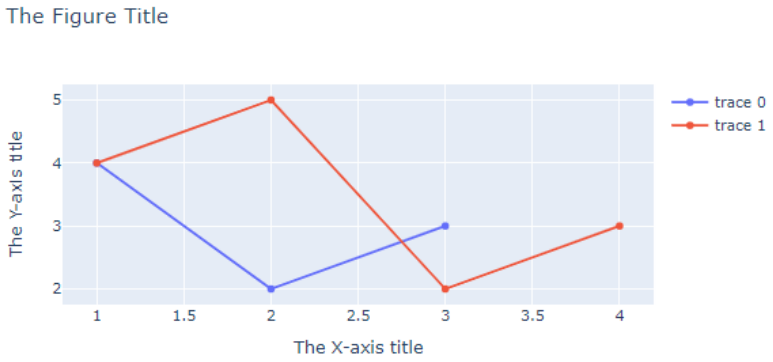


Figure Object의 요소 확인하기

- Figure에 현재 부여된 속성을 확인하기 위해 fig.show('json')을 이용해 tree 형식으로 속성 확인 가능

```
fig.show('json')
```

Configuration options

- dictionary로 정의하며 변경하려는 속성에 해당하는 key 값에 따라 list나 string 값을 받음

- displayModeBar: mode bar를 전부 다 보여줄건지 여부를 결정 (default : True)
- responsive: 브라우저 창 사이즈에 맞춰 figure dimension을 바꿀지 여부를 결정 (True)
- toImageButtonOptions: 이미지 다운로드 시 파일 포맷을 지정 (SVG, PNG, JPG 등), 이미지 파일의 이름, 너비와 높이도 조정 가능
- modeBarButtonsToRemove: mode bar에서 필요하지 않은 버튼들을 제거

```
fig.show(config={'displaylogo': False,

                 'modeBarButtonsToAdd': ['drawrect', 'drawcircle','eraseshape']})
```

만들어진 그림을 다른 포맷으로 변환하기

- to_***나 write_*** 를 이용해서 다른 포맷으로 변환
- html 파일로 만들기 (새 창으로 전송)

```
fig.write_html('html_plot.html',
               config = {'toImageButtonOptions':{'format':'svg'}})
```

- 이미지 파일로 만들기

```
fig.write_image('path/image_file.svg', height = 600, width = 800)
```

dataframe을 불러와서 그래프 만들기

- PovStatsData의 인구수 지표 '2010' 컬럼 값이 상위 20개 안에 드는 나라의 bar graph

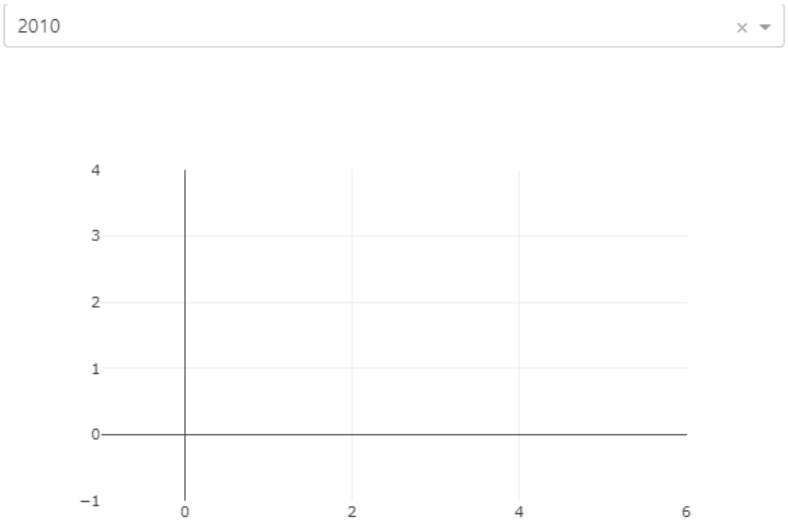
```
data = pd.read_csv('data/PovStatsData.csv')

regions = ['East Asia & Pacific', 'Europe & Central Asia',
           'Fragile and conflict affected situations', 'High income',
           'IDA countries classified as fragile situations', 'IDA total',
           'Latin America & Caribbean', 'Low & middle income', 'Low income',
           'Lower middle income', 'Middle East & North Africa',
           'Middle income', 'South Asia', 'Sub-Saharan Africa',
           'Upper middle income', 'World']

population_df = data[data['Country Name'].isin(regions) & (data['Indicator Name']=='Population, total')]

year = '2010'
year_df = population_df[['Country Name', year]].sort_values(year, ascending = False)[:20]

fig = go.Figure()
fig.add_bar(x = year_df['Country Name'], y = year_df[year])
fig.layout.title = f'Top twenty countries by population - {year}'
fig.show()
```

callback 함수와 그래프를 결합하기

- app.layout의 Graph 컴포넌트는 다른 dcc와 마찬가지로 인터랙티브한 그래프를 그릴 수 있게 한다
- dcc_Graph() 컴포넌트는 빈 그래프를 생성하고, component_id와 component_property를 정의하여 callback 함수로 그래프 변형 가능

```
app = JupyterDash(__name__)

app.layout = html.Div([
    ### value는 사용자에게 처음에 보여지는 default 값에 해당, option의 label-value는 label를 선택하면 value로 입력됨
    dcc.Dropdown(id = 'year_dropdown', value = '2010', options = [{'label':year, 'value':str(year)}
                                                                  for year in range(1974, 2019)]),
    dcc.Graph(id = 'population_chart'),
])
```

```
if __name__=='__main__':
    app.run_server(mode = 'inline')
```



- options는 다음과 같은 수식으로도 다양하게 정의 가능 (value 리스트 기반으로 label 리스트를 만들어내도 됨)

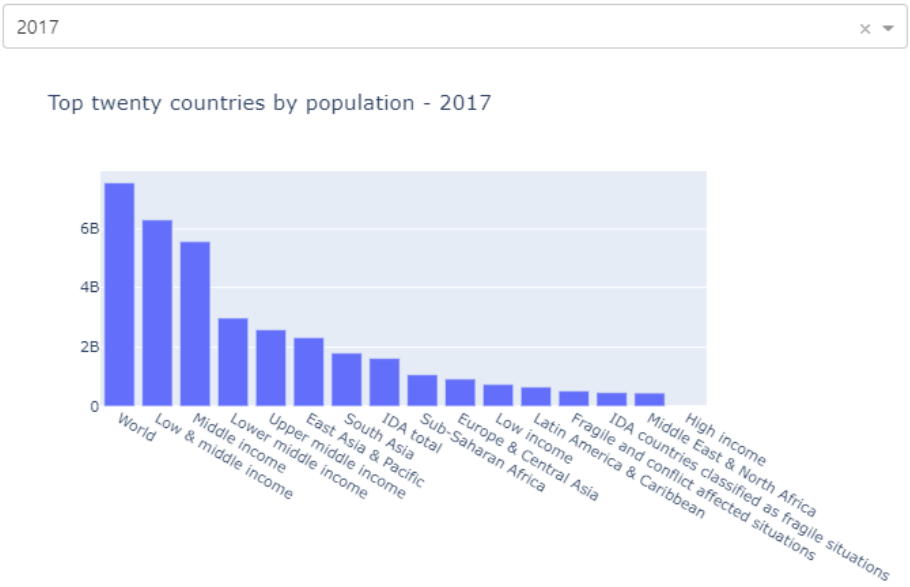
```
[{'label':color.title(), 'value':color} for color in ['blue', 'green', 'yellow']]
```

```
[{'label': 'Blue', 'value': 'blue'},
 {'label': 'Green', 'value': 'green'},
 {'label': 'Yellow', 'value': 'yellow'}]
```

- 위 기능들을 종합하여 최종적으로는 사용자가 선택한 카테고리에 해당하는 값을 확인하는 아래와 같은 그래프를 만들 수 있다.

```
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output, State
import plotly.graph_objects as go
import pandas as pd

app = JupyterDash(__name__)

##사용할 기본 데이터 만들기
data = pd.read_csv('data/PovStatsData.csv')

regions = ['East Asia & Pacific', 'Europe & Central Asia',
           'Fragile and conflict affected situations', 'High income',
           'IDA countries classified as fragile situations', 'IDA total',
           'Latin America & Caribbean', 'Low & middle income', 'Low income',
           'Lower middle income', 'Middle East & North Africa',
           'Middle income', 'South Asia', 'Sub-Saharan Africa',
           'Upper middle income', 'World']

population_df = data[data['Country Name'].isin(regions) & (data['Indicator Name']=='Population, total')]

## 앱 구성하기
app.layout = html.Div([
    ### value는 사용자에게 처음에 보여지는 default 값에 해당, option의 key-value는 key를 선택하면 value로 입력됨
    dcc.Dropdown(id = 'year_dropdown', value = '2010', options = [{'label':year, 'value':str(year)}
                                                                  for year in range(1974, 2019)]),
    dcc.Graph(id = 'population_chart'),
])

@app.callback(Output('population_chart', 'figure'),
              Input('year_dropdown', 'value'))
def plot_countries_by_population(year):
    year_df = population_df[['Country Name', year]].sort_values(year, ascending = False)[:20]
    ## fig.show() 가 아니라 return fig로 해줘야 함. fig.show()로 하면 무한 렉걸림
    fig = go.Figure()
    fig.add_bar(x = year_df['Country Name'], y = year_df[year])
    fig.layout.title = f'Top twenty countries by population - {year}'
```

```
    return fig

if __name__=='__main__':
    app.run_server(mode = 'inline')
```



Input 2개 - Output 2개인 인터랙티브 대시보드

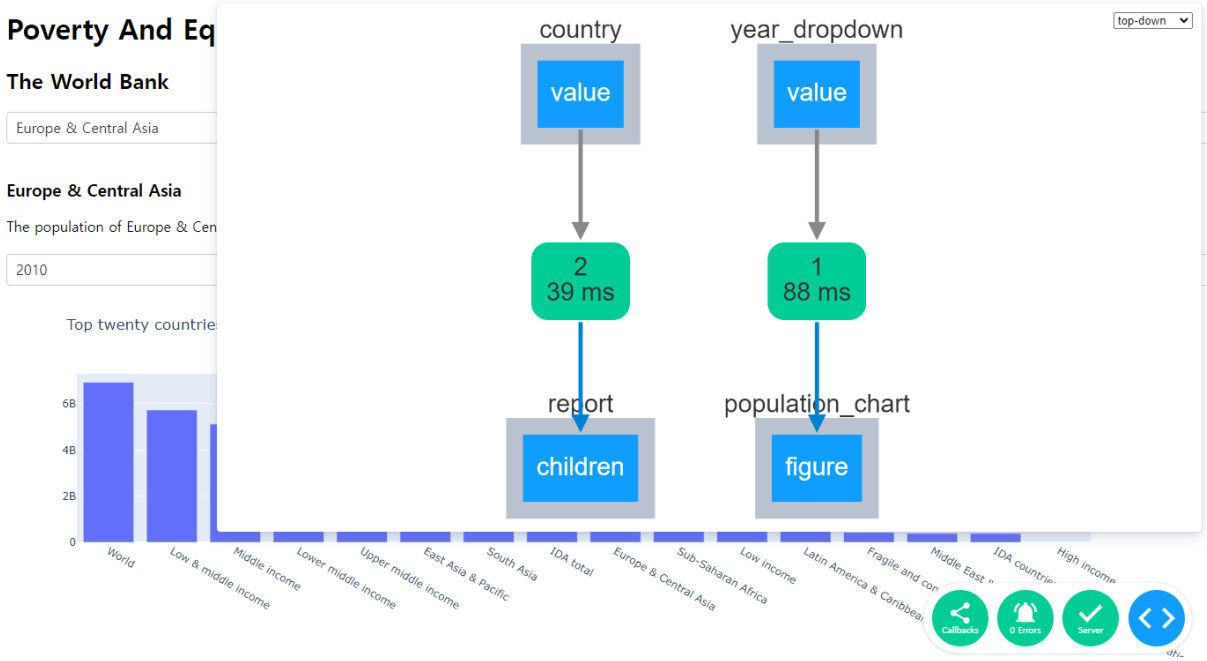
```
from jupyter_dash import JupyterDash
import dash_core_components as dcc
import dash_html_components as html
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output, State
import plotly.graph_objects as go
import pandas as pd

app = JupyterDash(__name__)

##사용할 기본 데이터 만들기
data = pd.read_csv('data/PovStatsData.csv')

regions = ['East Asia & Pacific', 'Europe & Central Asia',
           'Fragile and conflict affected situations', 'High income',
           'IDA countries classified as fragile situations', 'IDA total',
           'Latin America & Caribbean', 'Low & middle income', 'Low income',
           'Lower middle income', 'Middle East & North Africa',
           'Middle income', 'South Asia', 'Sub-Saharan Africa',
           'Upper middle income', 'World']

population_df = data[data['Country Name'].isin(regions) & (data['Indicator Name'] == 'Population, total')]

## 앱 구성하기
app.layout = html.Div([
    ## 보고서 상단 제목
    html.H1('Poverty And Equity Database'),
    html.H2('The World Bank'),
    ## Input 1 - country
    dcc.Dropdown(id = 'country',
                 options = [{'label':country, 'value':country} for country in data['Country Name'].unique()]),
    html.Br(),
    ## Output 1 - report
    html.Div(id = 'report'),
    html.Br(),
    ### Input 2 - year
    dcc.Dropdown(id='year_dropdown', value='2010', options=[{'label': year, 'value': str(year)}
                                                            for year in range(1974, 2019)]),
    ### Output 2 그래프 (population_chart)
    dcc.Graph(id='population_chart'),

    dbc.Tabs([
        dbc.Tab([
            html.Ul([
                html.Br(),
                html.Li('Numbers of Economies: 170'),
```

```
                html.Li('Temporal Coverage: 1974 - 2019'),
                html.Li('Update Frequency: Quarterly'),
                html.Li('Last Updated: March 18, 2020'),
                html.Li(['Source: ',
                         html.A('https://datacatalog.worldbank.org/dataset/poverty-and-equity-database',
                                href='https://datacatalog.worldbank.org/dataset/poverty-and-equity-database')])
            ])
        ], label='Key Facts'),  ## Tabs 안의 Tab 이름
        dbc.Tab([
            html.Ul([
                html.Br(),
                html.Li('Book title: Interactive Dashboards and Data Apps with Plotly and Dash'),
                html.Li(['GitHub repo: ',
                         html.A(
                             'https://github.com/PacktPublishing/Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash',
                             href='https://github.com/PacktPublishing/Interactive-Dashboards-and-Data-Apps-with-Plotly-and-Dash')])
            ])
        ], label='Project Info')  ## 두번째 탭 이름
    ]),  ## 여기까지가 dbc.Tabs
])


## 두번째 Output인 id = report에 들어갈 결과물
@app.callback(Output('report', 'children'),
              Input('country', 'value'))
def display_country_report(country):
    if country is None:
        return ''

    filtered_df = data[(data['Country Name'] == country) &
                               (data['Indicator Name'] == 'Population, total')]
    population = filtered_df.loc[:, '2010'].values[0]

    return [html.H3(country),
            f'The population of {country} in 2010 was {population:,.0f}.']


## 첫번째 Output id = population_chart에 들어갈 결과물
@app.callback(Output('population_chart', 'figure'),
              Input('year_dropdown', 'value'))
def plot_countries_by_population(year):
    year_df = population_df[['Country Name', year]].sort_values(year, ascending=False)[:20]
    ## fig.show() 가 아니라 return fig로 해줘야 함. fig.show()로 하면 무한 렉걸림
    fig = go.Figure()
    fig.add_bar(x=year_df['Country Name'], y=year_df[year])
    fig.layout.title = f'Top twenty countries by population - {year}'
    return fig

if __name__=='__main__':
    app.run_server(mode = 'inline', host='10.200.172.190')
```

- 아니 근데 만들어놓고 보니까 dbc.Tabs 이쪽에서 Tab이 제대로 구성이 안 되었는데 뭐가 문제인지 못 찾겠음...

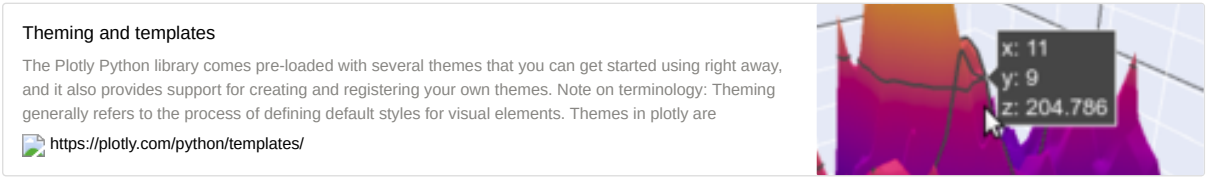
Callback에 걸린 시간 확인

- Plotly dash 결과를 외부 창으로 새롭게 띄우면 오른쪽 아래에 <> 파란색 버튼이 존재. 여기서 세번째 버튼을 누르면 input과 output에 걸린 시간을 확인할 수 있음



figure의 layout 바꾸기

- fig.layout.template = template이름 으로 Plotly 표의 테마를 정할 수 있음
- 선택할 수 있는 템플릿은 다음과 같음 : 'ggplot2', 'seaborn', 'simple_white', 'plotly',
  'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',
  'ygridoff', 'gridon', 'none'
- 아래 사이트에서 템플릿 적용 예시 확인



Data Manipulation and Preparation for Plotly

Long format(tidy) data의 형식

- 한 row가 중복되지 않은 하나의 observation에 대한 설명을 가지고 있는 것
- Tidy data의 효용성과 다양한 타입의 데이터를 Tidy 데이터로 정리하는 법은 Hadley Wickham의 논문 ['Tidy Data'](https://www.jstatsoft.org/article/view/v059i10)를 참조

Wide format

	country	indicator	2015	2020
0	country_A	indicator 1	100	120
1	country_B	indicator 1	10	15

Long (tidy) format

	country	indicator	year	value
0	country_A	indicator 1	2015	100
1	country_B	indicator 1	2015	10
2	country_A	indicator 1	2020	120
3	country_B	indicator 1	2020	15

Melting Dataframes

wide format 데이터는 long format 데이터로 변환하는 것이 데이터의 직관성 면에서 좋은 선택임. 변환은 melt 함수를 써서 이루어짐

```
wide_df.melt(id_vars = ['country', 'inicator'],
             value_vars = ['2015', '2020'],
             var_name = 'year')
```

- id_vars: 데이터를 mapping하기 위한 key에 해당(변하지 않고 row가 추가되면서 중복되는 부분)
- value_vars: 어떤 컬럼을 변수들로 바꾸어 넣을지
- var_name: 변수들 넣을 컬럼의 이름

Pivoting Dataframes

```
data_pivot = \
data_melt.pivot(index = ['Country Name', 'Country Code', 'year'], columns = 'Indicator Name',
                values = 'value').reset_index()

## pivot이 중복 없이 잘 생성되었는지 확인
data_pivot[['Country Code', 'year']].duplicated().any()
```

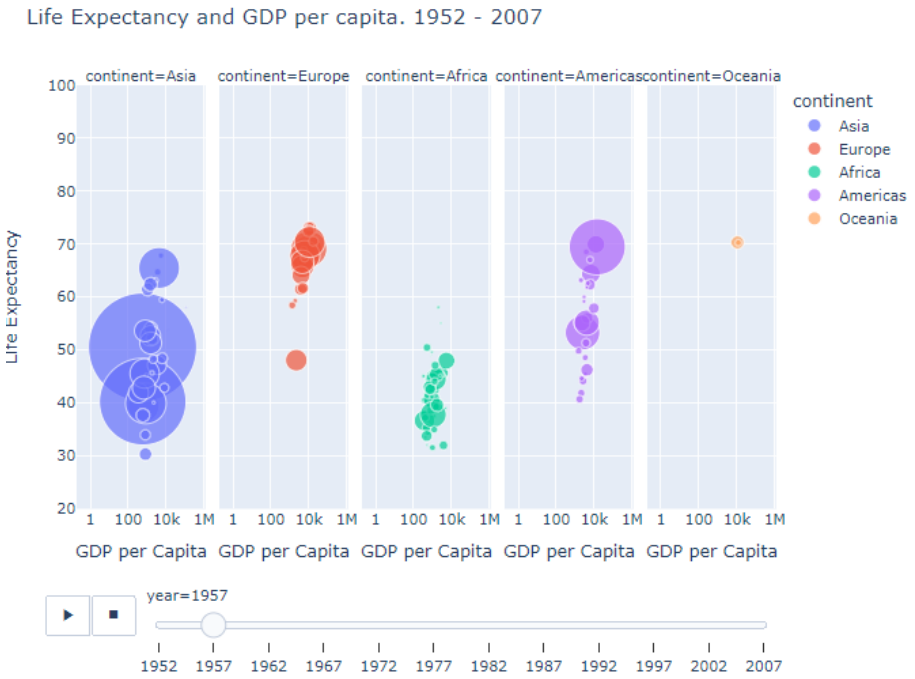
Learning Plotly Express

이번 챕터는 plotly express에 들어있는 gapminder 데이터를 활용

Plotly express의 기본 파라미터를 이용해 인터랙티브한 그래프를 그려보면 아래와 같다.

```
import plotly.express as px
gapminder = px.data.gapminder()

px.scatter(data_frame=gapminder, x='gdpPercap',

           y='lifeExp',

           size='pop',

           facet_col='continent',

           color='continent',

           title='Life Expectancy and GDP per capita. 1952 - 2007',

           labels={'gdpPercap': 'GDP per Capita','lifeExp': 'Life Expectancy'},

           log_x=True,

           range_y=[20, 100],

           hover_name='country',

           animation_frame='year',

           height=600,
```

```
        size_max=90)
```



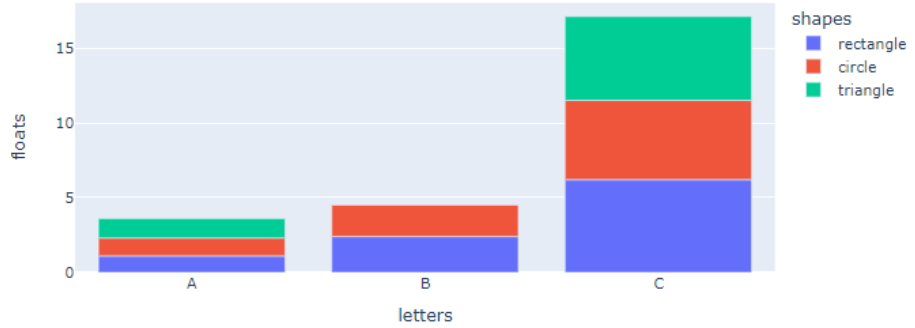
- hue와 동일한 기능을 하는 것: color
- facet_col: 해당 카테고리에 따라 병렬로 그래프를 그려줌
- size: 산점도 marker의 사이즈를 결정, 변수가 될 수도 있고 (size = '컬럼명') 특정 크기로 지정(size = 8) 혹은 list 형태로 한 row에 따라 전부 다르게 지정할 수 있음 (size = [35]*8 은 총 8개의 row를 가진 데이터에 35 사이즈 마커를 적용)
- animation_frame: 해당 변수가 달라짐에 따라 변화하는 그래프의 모습을 볼 수 있음. 애니메이션 재생은 왼쪽의 ▶️버튼을 누르면 됨

- bar graph에서 이렇게도 쌓기 가능

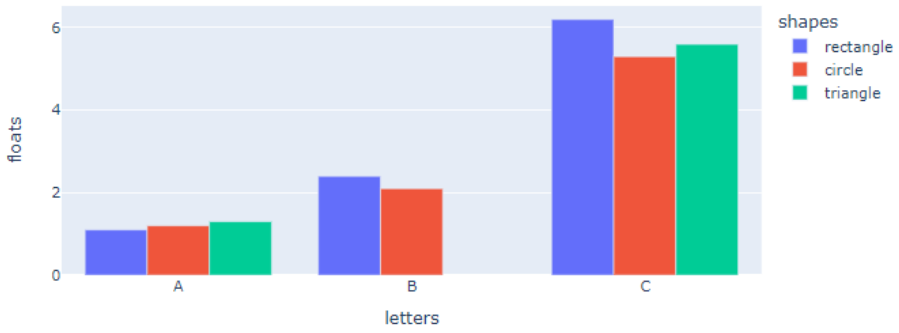
```
df = pd.DataFrame({

    'numbers': [1, 2, 3, 4, 5, 6, 7, 8],

    'colors': ['blue', 'green', 'orange', 'yellow', 'black', 'gray', 'pink', 'white'],

    'floats': [1.1, 1.2, 1.3, 2.4, 2.1, 5.6, 6.2, 5.3],

    'shapes': ['rectangle', 'circle', 'triangle', 'rectangle', 'circle', 'triangle', 'rectangle', 'circle'],

    'letters': list('AAABBCCC')

})
```

	numbers	colors	floats	shapes	letters
0	1	blue	1.1	rectangle	A
1	2	green	1.2	circle	A
2	3	orange	1.3	triangle	A
3	4	yellow	2.4	rectangle	B
4	5	black	2.1	circle	B
5	6	gray	5.6	triangle	C
6	7	pink	6.2	rectangle	C
7	8	white	5.3	circle	C

```
px.bar(df, x = 'letters', y = 'floats', color = 'shapes')
```



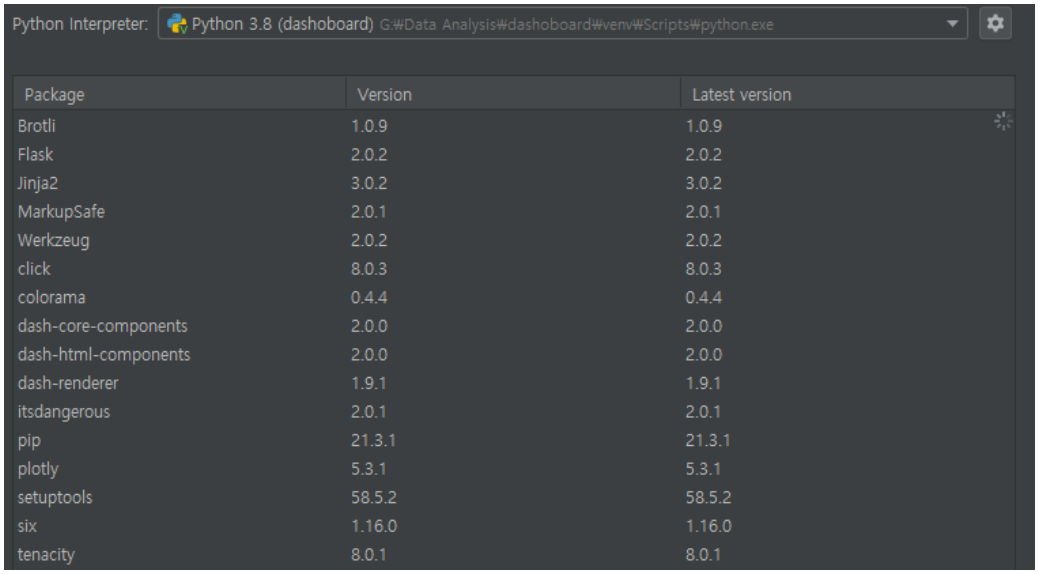

```
px.bar(df, x = 'letters', y = 'floats', color = 'shapes', barmode = 'group')
```



Interactively Comparing Values with Bar charts and Dropdown Menus

Dashboard 실습

[Dash설치](#Dash_Installation)의 5개 패키지를 설치했고 패키지 리스트는 다음과 같음

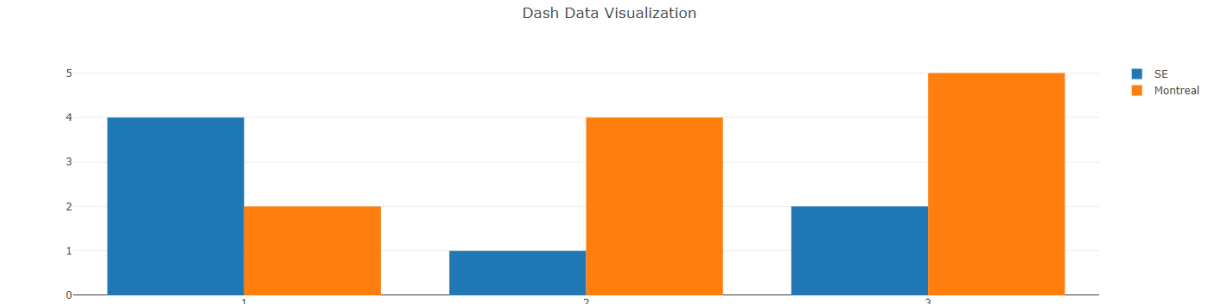


Package	Version	Latest version
Brotli	1.0.9	1.0.9
Flask	2.0.2	2.0.2
Jinja2	3.0.2	3.0.2
MarkupSafe	2.0.1	2.0.1
Werkzeug	2.0.2	2.0.2
click	8.0.3	8.0.3
colorama	0.4.4	0.4.4
dash-core-components	2.0.0	2.0.0
dash-html-components	2.0.0	2.0.0
dash-renderer	1.9.1	1.9.1
itsdangerous	2.0.1	2.0.1
pip	21.3.1	21.3.1
plotly	5.3.1	5.3.1
setuptools	58.5.2	58.5.2
six	1.16.0	1.16.0
tenacity	8.0.1	8.0.1

1. 간단한 대시보드 만들기

Hello Dash

Dash: A web application framework for Python.




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

app = dash.Dash()         ## 코드설명1

##코드설명2
app.layout = html.Div(children = [
    html.H1(children = 'Hello Dash'),
    html.Div(children='Dash: A web application framework for Python.'),

##코드설명3
    dcc.Graph(
        id = 'example-graph',
        figure={
            'data': [
                {'x': [1,2,3], 'y':[4,1,2], 'type': 'bar', 'name':'SE'},
```

```
                {'x': [1,2,3], 'y':[2,4,5], 'type':'bar', 'name':u'Montreal'},
            ],
            'layout': {
                'title': 'Dash Data Visualization'
            }
        }
    )
])

##코드설명4
if __name__ =='__main__':
    app.run_server()
```

- 코드설명1

  Dash()는 dash app을 시작. Dash instance 생성을 더 편리하게 하기 위해 dash.Dash를 app으로 명명

- 코드설명2

  app의 layout을 설정한다.

  - H1, Div는 html 태그에 대응하는 컴포넌트 속성을 만들어냄. (H1→ <h1>~</h1>, Div→ ~)
  - children은 html 컴포넌트의 속성을 넣는 데 사용함. childeren은 method의 첫번째 속성이므로 건너뛰어도 'children='을 굳이 쓸 필요는 없음

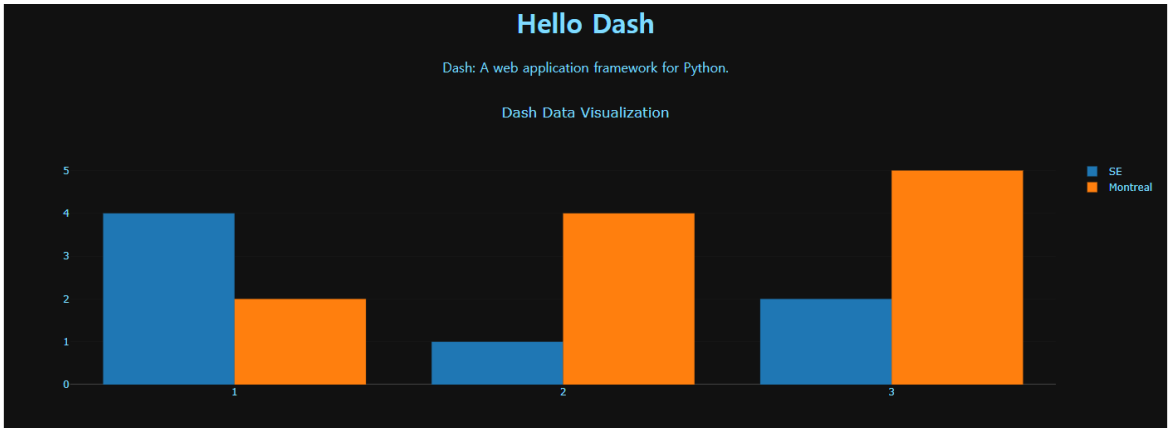
- 코드설명3

  Dash 그래프를 figure를 통해 정의

- 코드설명4

  - 로컬 Falsk 서버에 코드를 띄우는 역할
  - run_server에 debug=True를 추가하면 서버 에러시 문제를 진단할 수 있음

html 디자인 변경




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

app = dash.Dash()

## 뒤에서 사용할 style color을 Dictionary에 저장해 관리
colors = {
 'background': '#111111',
 'text': '#7FDBFF'
}

app.layout = html.Div(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':'SE'},
                {'x': [1,2,3], 'y':[2,4,5], 'type':'bar', 'name':u'Montreal'},
            ],
            'layout': {
                'plot_bgcolor': colors['background'],
                'paper_bgcolor': colors['background'],
                'font': {
                    'color': colors['text']
                },
                'title': 'Dash Data Visualization'
            }
        }
    )],
    style = {'backgroundColor': colors['background']}
)

if __name__ =='__main__':
    app.run_server()
```

Dash HTML Components

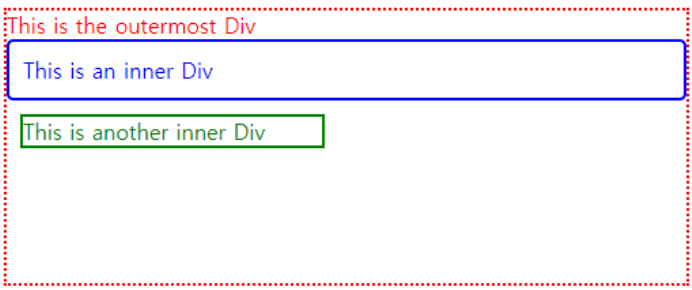
Dash에서 제공중인 html Component는 다음 페이지에서 확인

> Dash HTML Components
>
> Dash provides all of the available HTML tags as user-friendly Python classes. This chapter explains how this works and the few important key differences between Dash HTML components and standard html.
>
> https://dash.plot.ly/dash-html-components

Dash HTML Component와 HTML은 아래와 같은 특징이 다름

- sytle 속성이 dictionary로 정의됨
- style 속성은 CamelCase로 기재됨
- class key를 ClassName으로 부름
- Style properties in pixel units can be supplied as just numbers without the px unit (머선소리고?)

실습예제




```
import dash
import dash_html_components as html
```

```
app = dash.Dash()

app.layout = html.Div([
    'This is the outermost Div',
    html.Div(
        'This is an inner Div',
        style = {'color':'blue', 'border':'2px blue solid', 'borderRadius':5,'padding':10, 'Width':220}
    ),
    html.Div(
        'This is another inner Div',
        style = {'color':'green', 'border':'2px green solid', 'margin':10, 'width':220}
    ),
],
style = {'width':500, 'height':200, 'color':'red', 'border':'2px red dotted'}
)

if __name__ == '__main__':
    app.run_server()
```

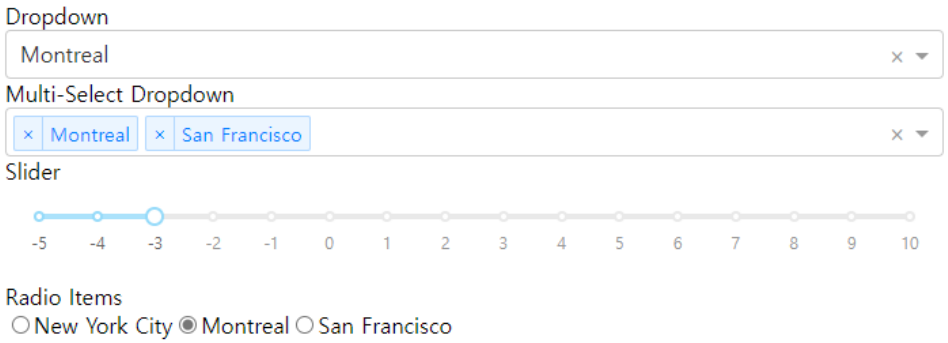
Dash Core Components

Dash Core Components

Dash ships with supercharged components for interactive user interfaces. A core set of components, written and maintained by the Dash team, is available in the dash-core-components library. For production Dash apps, the Dash Core Components styling & layout should be managed with Dash Enterprise Design Kit.

https://dash.plotly.com/dash-core-components

실습예제




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

app = dash.Dash()

app.layout = html.Div([
    # Dropdown
    html.Label("Dropdown"),
    dcc.Dropdown(
        options = [
            {'label': 'New York City', 'value':'NYC'},
            {'label':'Montreal', 'value':'MTL'},
            {'label':'San Fransisco', 'value':'SF'}
        ],
        value = 'MTL'
    ),

    # Multi-Select Dropdown
    html.Label('Multi-Select Dropdown'),
    dcc.Dropdown(
        options = [
            {'label':'New York City', 'value':'NYC'},
```

```
            {'label':u'Montreal', 'value':'MTL'},
            {'label':'San Francisco', 'value':'SF'}
        ],
        value = ['MTL', 'SF'],
        multi = True
    ),

    # Slider
    html.Label('Slider'),
    html.P(
        dcc.Slider(
            min = -5,
            max = 10,
            step = 0.5,
            marks = {i: i for i in range(-5,11)},
            value = -3
        )
    ),

    ## Radio Items
    html.Label('Radio Items'),
    dcc.RadioItems(
        options = [
            {'label': 'New York City', 'value':'NYC'},
            {'label':'Montreal', 'value':'MTL'},
            {'label':'San Francisco', 'value':'SF'}
        ],
        value = 'MTL'
    )
], style = {'width':'50%'})

if __name__ == '__main__':
    app.run_server()
```

Callback 데이터를 저장하는 법

https://dash.plotly.com/sharing-data-between-callbacks

Ouput의 공백 제거하기

https://community.plotly.com/t/remove-some-space-above-graph-within-div/14980