

# DATA VISUALIZATION AND DASHBOARD FOR MONITORING COVID-19

*Mobile Internet Course Project*

**Yang Jinhai, 517030910300**

June 15, 2020

## 1 Introduction

In 2020, COVID-19 has turned the world upside down. Everything has been impacted. How we live and interact with each other, how we work and communicate, how we move around and travel. Every aspect of our lives has been affected. Although the world is in lockdown, governments, epidemiologists, school principals, entrepreneurs and families around the world are already planning the next steps: how to safely reopen schools and businesses, how to commute and travel without transmitting or contracting infection. Therefore, they need a clear understanding of COVID-19. A visualization of COVID-19 and a dashboard for monitoring will make their decision more accurately and reasonably.

In this course project, my purpose is to visualize the data and make a dashboard for monitoring by using Python only.

All the work is done by myself because there is only one person in the group.

## 2 Environment

- **IDE**

VS Code, Pycharm or any your personal choice of IDE that can write Python code

- **Python: belowed are some package used in this project**

1. **Requests**

Python built-in module *urllib* is used to access network resources. However, it's troublesome to use it. Therefore, I use *requests*, a more convenient library based on *urllib*, to handle the url resources.

2. **Pandas**

*Pandas* is a data analysis library based on *numpy* in Python. It provides good support for time series analysis which helps us to process data to make a dashboard.

3. **Jsonpath**

*Jsonpath* is an information extraction library in Python used to parse multiple layer of nested json data (extract specified information form json document).

4. **Pyecharts**

Echarts is a data visualization JS library open sourced by Baidu. *Pyecharts* is a library for generating Echarts charts which interface with Python to conveniently generate graphs with data.

5. **Plotly**

The *plotly* Python library (*plotly.py*) 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.py* enables Python users to create interactive web-based visualizations served as part of pure Python-built web applications using *Dash*.

6. **Dash**

*Dash* is a productive Python framework written on top of *Flask*, *Plotly.js* and *React.js* for building web applications. It is ideal for building data visualization apps with highly custom user interfaces in pure Python.

## 3 Data Visualization

### 3.1 Data Acquisition

There are two data sources I use to acquire the data, one is [Tencent's COVID-19 Real Time Tracking](#), another is [Novel Coronavirus \(COVID-19\) Cases, provided by JHU CSSE](#).

From Tencent's COVID-19 Real Time Tracking, I can get a real time data of COVID-19. The code of getting the data of different countries from this website are followed:

```
1 def catch_foreign_data():
2     url_foreign = "https://api.inews.qq.com/newsqa/v1/automation/foreign/country/ranklist"
3     response = requests.get(url_foreign).json()
4     return response
5
6 data = catch_foreign_data()
```

However, Tencent's Real Time Tracking provides no convenient interface for public to get COVID-19 data with time series. Therefore, it's a better choice to get data with time series provided by JHU CSSE. These data will be updated on Github each day which can be downloaded to local to increase the speed of our program if we don't pursue the real-time nature of our website. The code of requesting data from JHU CSSE is followed:

```
1 # crawl the data directly from github
2 url_confirmed = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/'\
3 'csse_covid_19_time_series/time_series_covid19_confirmed_global.csv'
4 url_deaths = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/'\
5 'csse_covid_19_time_series/time_series_covid19_deaths_global.csv'
6 url_recovered = 'https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/'\
7 'csse_covid_19_time_series/time_series_covid19_recovered_global.csv'
8
9 df_confirmed = pd.read_csv(url_confirmed)
10 df_deaths = pd.read_csv(url_deaths)
11 df_recovered = pd.read_csv(url_recovered)
```

Then, the data has been acquired. I use the data from Tencent's COVID-19 Real Time Tracking for data visualization and the data from JHU CSSE for dashboard.

### 3.2 Visualization

In this part, I use the *pyecharts* library generate world distribution map of COVID-19.

At first, I use *jsonpath* library to process the crawled json file by getting the name of countries and the number of each country's confirm cases. Then I combine them to be `data_pair` which is the input data form of *pyecharts*' map component.

```
1 data = catch_foreign_data()
2 name = jsonpath.jsonpath(data, "$..name")
3 confirm = jsonpath.jsonpath(data, "$..confirm")
4 data_list = list(zip(name, confirm))
```

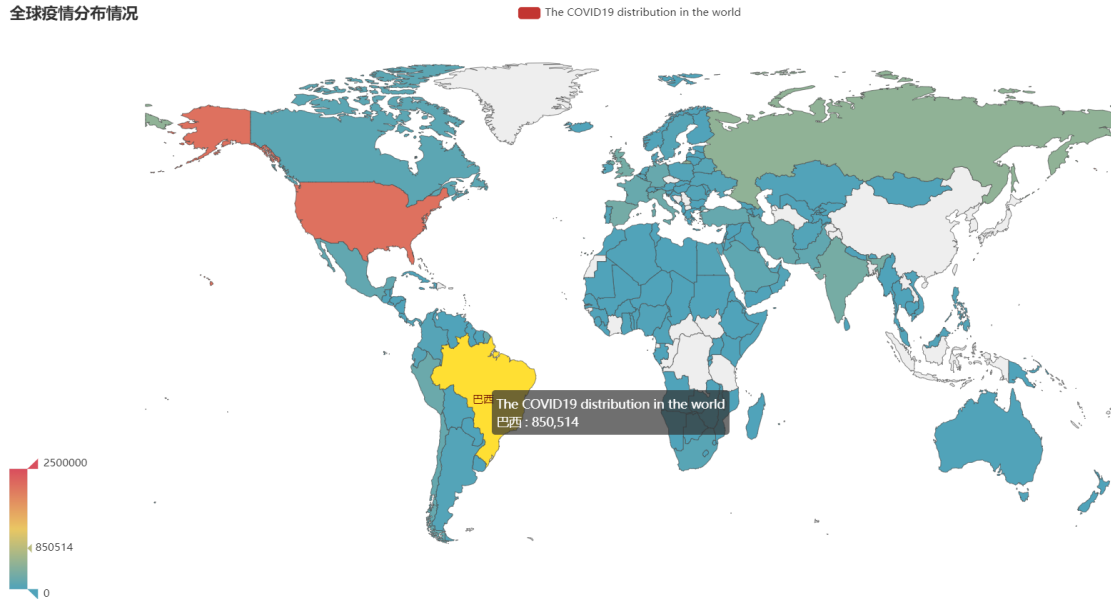
In addition, the map component in the *pyecharts* requires the English name of each country. However, the data crawled from Tencent's COVID-19 Real Time Tracking only contains Chinese name of each country. Therefore, a Chinese-English dictionary of countries is need. Below is a part of the dictionary:

```
1 country_dict = {
2     'Afghanistan': '阿富汗',
3     'Singapore': '新加坡',
4     'Argentina': '阿根廷',
5     'Armenia': '亚美尼亚',
6     'Australia': '澳大利亚',
7     'Greece': '希腊',
8     'Greenland': '格陵兰',
9     'India': '印度',
10    'Ireland': '爱尔兰',
11    'Iran': '伊朗',
12    'Iraq': '伊拉克',
13    'Iceland': '冰岛',
14    'Israel': '以色列',
15    'Italy': '意大利',
16    'Jamaica': '牙买加',
17    'Japan': '日本',
18    'Kazakhstan': '哈萨克斯坦',
19    'Kenya': '肯尼亚',
20    'Cambodia': '柬埔寨',
21    'South Korea': '韩国'
22 }
```

After processing the data, I can use the data to generate the world distribution map of COVID-19. The code of generating map is followed:

```
1 world_map = Map(init_opts=opts.InitOpts(width='1400px', height='700px'))
2 world_map.add(series_name="The COVID19 distribution in the world",
3               data_pair=data_list,
4               maptype="world",
5               name_map=country_dict,
6               is_map_symbol_show=False)
7 world_map.set_series_opts(label_opts=opts.LabelOpts(is_show=False))
8 world_map.set_global_opts(
9     title_opts=opts.TitleOpts(title="全球疫情分布情况"),
10    visualmap_opts=opts.VisualMapOpts(max_=2500000))
11 world_map.render("World.html")
```

Then the world distribution map has been generated:



However, as we can see on the map, some countries are left blank. There may be two reasons for this phenomenon. One is because there is no corresponding value in the *country-dict* dictionary. Another reason may be that the country left blank has no affected cases. I will do some further exploring to figure out the exact reason and fix the map.

## 4 Design of Dashboard for Monitoring

The dashboard is the most important and difficult part of this project. The data I used is from JHU CSSE because it associates with time series. I mainly use a Python built-in library **Dash** to finish this part. In addition, **Pandas** is an important tool to process the data and **Plotly** is also an important tool to draw the graph. The dashboard will be display as a website at last.

### 4.1 Total confirmed, death and recovered cases

On the top of this website, I display the total global confirmed, death and recovered cases. I use the total global confirmed cases as example. First, I use label H3 to make the title of this part:

```
1 html.H3(
2     children='Total Confirmed',
3     style={
4         'textAlign': 'center',
5         'color': colors['confirmed_text'],
6     }
7 ),
```

Then I process the data to get the sum of confirmed, death and recovered cases:

```
1 df_confirmed_total = df_confirmed.iloc[:, 4:].sum(axis=0)
2 df_deaths_total = df_deaths.iloc[:, 4:].sum(axis=0)
3 df_recovered_total = df_recovered.iloc[:, 4:].sum(axis=0)
```

After processing the data, I use label P in this part which automatically creates some blank before and after the element making the text a paragraph:

```

1  html.P(
2      f"{df_confirmed_total[-1]:,d}",
3      style={
4          'textAlign': 'center',
5          'color': colors['confirmed_text'],
6          'fontSize': 30
7      }
8  ),
9
10 html.P(
11     'Past 24h increase: +' + f"{df_confirmed_total[-1] - df_confirmed_total[-2]:,d}" + '(' +
12     str(round((df_confirmed_total[-1] - df_confirmed_total[-2])/df_confirmed_total[-1]*100, 2)) + '%)',
13     style={
14         'textAlign': 'center',
15         'color': colors['confirmed_text'],
16     }
17 )

```

Therefore, the preview of the total global confirmed, death and recovered cases has been generated:



## 4.2 Graph of global COVID-19 cases (total and daily)

The basic components of the graph are radio buttons (button of total cases and daily cases) and graph:

```

1  html.Div([
2      dcc.RadioItems(
3          id='graph-type',
4          options=[{'label': i, 'value': i} for i in ['Total Cases', 'Daily Cases']],
5          value='Total Cases',
6          labelStyle={'display': 'inline-block'},
7          style={'fontSize': 20, },
8      )
9  ]),
10
11 html.Div([
12     dcc.Graph(
13         id='global-graph',
14     )
15 ]),

```

Here, I will introduce an important component in *Dash*: **callback**. The callback component consists of two components: input and output. When the value of the input component changes, it will automatically call the function packaged by the callback decorator, using the updated content as the input parameter, returning the input content of the function, and updating the value of the output component. This method using callback function is a type of **Reactive Programming**.

In this part, the input value is the value of the graph-type which is determined by clicking the RadioItems component. The output value is the graph generated according to the input value.

```

1  # output: id='global_graph', 'figure' [output: the figure of a graph]
2  # input: 'graph-type', 'value' [input: the value of the radio_items]
3  @app.callback(
4      Output('global-graph', 'figure'),
5      [Input('graph-type', 'value')])
6  def update_graph(graph_type):
7      fig_global = draw_global_graph(df_confirmed_total, df_deaths_total, df_recovered_total, graph_type)
8      return fig_global

```

In the `draw_global_graph()` function, I use the datetime in the pandas array as the index of this graph to draw it.

When the input component's value is 'Daily', the daily cases' data equals to the today's total data minus yesterday's total data.

In addition, I use a function named `add_trace()` to add trace to a plotly interactive visualization, to see the data accurately on each day.

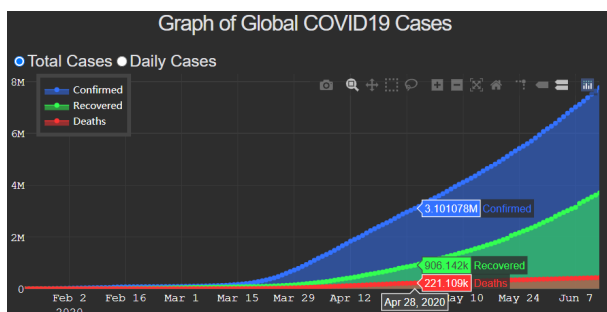
In summary, I can draw the figure combining the function above. The code is followed:

```

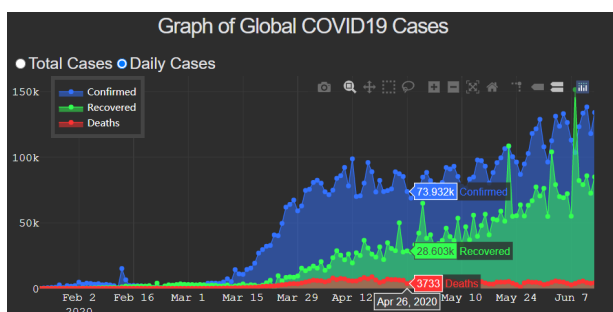
1  # draw the global figure: total/daily to attached to dcc component
2  def draw_global_graph(confirmed_total, deaths_total, recovered_total, graph_type='Total Cases'):
3      confirmed_total.index = pd.to_datetime(confirmed_total.index)
4
5      if graph_type == 'Daily Cases':
6          confirmed_total = (confirmed_total - confirmed_total.shift(1)).drop(confirmed_total.index[0])
7          deaths_total = (deaths_total - deaths_total.shift(1)).drop(deaths_total.index[0])
8          recovered_total = (recovered_total - recovered_total.shift(1)).drop(recovered_total.index[0])
9
10     fig = go.Figure()
11
12     fig.add_trace(go.Scatter(x=confirmed_total.index, y=confirmed_total,
13                             mode='lines+markers',
14                             name='Confirmed',
15                             line=dict(color='#3372FF', width=1),
16                             fill='tozeroy',))
17     fig.add_trace(go.Scatter(x=confirmed_total.index, y=recovered_total,
18                             mode='lines+markers',
19                             name='Recovered',
20                             line=dict(color='#33FF51', width=1),
21                             fill='tozeroy',))
22     fig.add_trace(go.Scatter(x=confirmed_total.index, y=deaths_total,
23                             mode='lines+markers',
24                             name='Deaths',
25                             line=dict(color='#FF3333', width=1),
26                             fill='tozeroy',))
27
28     fig.update_layout(
29         hovermode='x',
30         font=dict(
31             family="Courier New, monospace",
32             size=14,
33             color=colors['figure_text'],
34         ),
35         legend=dict(
36             x=0.02,
37             y=1,
38             traceorder="normal",
39             font=dict(
40                 family="sans-serif",
41                 size=12,
42                 color=colors['figure_text']
43             ),
44             bgcolor=colors['background'],
45             borderwidth=5
46         ),
47         paper_bgcolor=colors['background'],
48         plot_bgcolor=colors['background'],
49         margin=dict(
50             l=0, r=0, t=0, b=0
51         ),
52         height=300,
53     )
54
55     fig.update_xaxes(showgrid=True, gridwidth=1, gridcolor='#3A3A3A')
56     fig.update_yaxes(showgrid=True, gridwidth=1, gridcolor='#3A3A3A')
57
58     fig.update_yaxes(zeroline=True, zerolinewidth=2, zerolinecolor='#3A3A3A')
59     return fig

```

The figure I draw can be viewed below:



(a) Global COVID-19 total cases



(b) Global COVID-19 daily cases

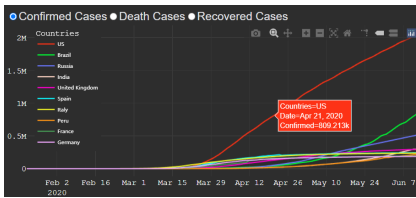
### 4.3 Graph of top ten highest cases countries (confirmed, death and recovered)

The graph can be drawn similarly using the procedure of drawing graph of global COVID-19 cases (total and daily) above. The most important part's (callback function) code is followed:

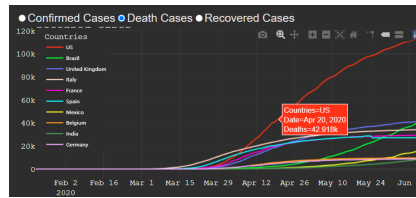
```
1 # output: id='high10-graph', 'figure' [output: the figure of a graph]
2 # input: 'graph-high10-type', 'value' [input: the value of the radio_items]
3 @app.callback(
4     Output('high10-graph', 'figure'),
5     [Input('graph-high10-type', 'value')])
6 def update_graph_high10(graph_high10_type):
7     fig_high10 = draw_highest_10(df_confirmed_t_stack, df_deaths_t_stack, df_recovered_t_stack, graph_high10_type)
8     return fig_high10
```

The function *draw\_highest\_10()* can be written similarly with the function *draw\_global\_graph()* by sorting the data and selecting the highest 10 countries for confirmed, death and recovered independently.

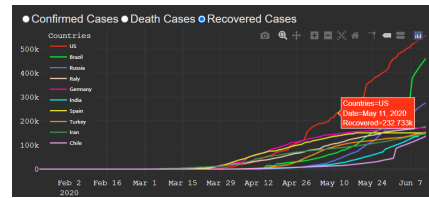
The figure I draw can be viewed below:



(c) Top 10 confirmed cases countrise



(d) Top 10 death cases countrise



(e) Top 10 recovered cases countrise

### 4.4 Top ten total and single day highest cases countries (confirmed and death)

The preview of the top 10 highest cases countries, including total and single day highest on confirmed and death cases will be handled separately.

At first, the data need to be transformed into a form in html label P which I have used in the preview of the total confirmed, death and recovered cases. I write a function to process data and transform it to a label P form (country name | total cases of the country +increasing cases' number(percent if death)). The code is followed:

```
1 def high_cases(country_name, total, single, color_word='#63b6ff', confirmed_total=1, deaths=False):
2
3     if deaths:
4         percent = (total/confirmed_total)*100
5         return html.P([
6             html.Span(
7                 country_name + ' | ' + f"{int(total):,d}",
8                 style={'backgroundColor': colors['highest_case_bg'], 'borderRadius': '6px', }
9             ),
10            html.Span(
11                ' + ' + f"{int(single):,d}",
12                style={'color': color_word, 'margin': 2, 'fontWeight': 'bold', 'fontSize': 14, }
13            ),
14            html.Span(
15                f' ({percent:.2f}%)', style={'color': color_word, 'margin': 2, 'fontWeight': 'bold', 'fontSize': 14, }
16            ),
17        ],
18        style={'textAlign': 'center', 'color': 'rgb(200,200,200)', 'fontSize': 12, }
19    )
20
21    return html.P([
22        html.Span(
23            country_name + ' | ' + f"{int(total):,d}",
24            style={'backgroundColor': colors['highest_case_bg'], 'borderRadius': '6px', }
25        ),
26        html.Span(
27            ' + ' + f"{int(single):,d}", style={'color': color_word, 'margin': 2, 'fontWeight': 'bold', 'fontSize': 14, }
28        ),
29    ],
30    style={'textAlign': 'center', 'color': 'rgb(200,200,200)', 'fontSize': 12, }
31    )
```

Then, the data in a new form must be appended to a list to fit the data form which will be displayed. I take the top 10 highest countries on confirmed cases as example. The code is followed:

```

1 noToDisplay =10
2
3 confirm_cases =[]
4 for i in range(noToDisplay):
5     confirm_cases.append(high_cases(df_confirmed_sorted_total.iloc[i, 0], df_confirmed_sorted_total.iloc[i, 1],
6                                     df_confirmed_sorted_total.iloc[i, 2]))

```

After data processing, I use the label P to display the data processed before. The code is followed:

```

1 html.P([
2     html.Span('Countries with highest cases: ',),
3     html.Br(),
4     html.Span(' + past 24hrs',
5               style={'color': colors['confirmed_text'], 'fontWeight': 'bold', 'fontSize': 14, })
6 ],
7   style={
8       'textAlign': 'center',
9       'color': 'rgb(200,200,200)',
10      'fontSize': 12,
11      'backgroundColor': '#3B5998',
12      'borderRadius': '12px',
13      'fontSize': 17,
14  }
15 ),
16 html.P(confirm_cases)

```

The preview of top 10 highest cases countries (total and single day) on confirmed and death has been generated and displayed below:

Countries with highest cases: + past 24hrs	Single day highest cases: + past 24hrs	Countries with highest deaths: + past 24hrs (Death Rate)	Single day highest Death: + past 24hrs (Death Rate)
US   2,074,526 +25,540	US   2,074,526 +25,540	US   115,436 +767 (5.56%)	Brazil   42,720 +892 (5.02%)
Brazil   850,514 +21,704	Brazil   850,514 +21,704	Brazil   42,720 +892 (5.02%)	US   115,436 +767 (5.56%)
Russia   519,458 +8,697	India   308,993 +11,458	United Kingdom   41,747 +181 (14.11%)	Mexico   16,872 +424 (11.82%)
India   308,993 +11,458	Russia   519,458 +8,697	Italy   34,301 +78 (14.49%)	India   8,884 +386 (2.88%)
United Kingdom   295,828 +1,426	Chile   167,355 +6,509	France   29,401 +24 (15.18%)	Chile   3,101 +231 (1.85%)
Spain   243,605 +396	Pakistan   132,405 +6,472	Spain   27,136 +0 (11.14%)	Peru   6,308 +220 (2.86%)
Italy   236,651 +346	Peru   220,749 +5,961	Mexico   16,872 +424 (11.82%)	United Kingdom   41,747 +181 (14.11%)
Peru   220,749 +5,961	South Africa   65,736 +3,809	Belgium   9,650 +4 (16.11%)	Russia   6,819 +114 (1.31%)
France   193,746 +526	Mexico   142,690 +3,494	India   8,884 +386 (2.88%)	Pakistan   2,551 +88 (1.93%)
Germany   187,267 +41	Saudi Arabia   123,308 +3,366	Germany   8,793 +10 (4.70%)	Italy   34,301 +78 (14.49%)

## 4.5 Table of all countries' COVID-19 cases and their graph

In this part, I will design a table which contains all countries' COVID-19 cases on confirmed, death and recovered. There is a button on the left side of each country. If the button is clicked, two graphs which visualize the country's COVID-19 cases will be displayed on the right side of the table.

### 4.5.1 Table of all countries' COVID-19 cases

To make a table containing all countries' COVID-19 cases data, I use a component in *Dash* named *DataTable* to complete this task.

At first, I should finish the task of data processing. In this part, I create a new array `map_data` to sort the data with the column needed such as Province/State, Country/Region, Confirmed, Deaths and Recovered. Then, I sort the array descending by the index of Confirmed.

```

1 # Recreate required columns for data_table
2 map_data =df_confirmed[["Province/State", "Country/Region", "Lat", "Long"]]
3 map_data['Confirmed'] =df_confirmed.loc[:, df_confirmed.columns[-1]]
4 map_data['Deaths'] =df_deaths.loc[:, df_deaths.columns[-1]]
5 map_data['Recovered'] =df_recovered_fill.loc[:, df_recovered_fill.columns[-1]]
6 map_data['Recovered'] =map_data['Recovered'].fillna(0).astype(int) # too covert value back to int and fillna with zero
7
8 # last 24 hours increase
9 # map_data['Deaths_24hr'] = df_deaths.iloc[:, -1] - df_deaths.iloc[:, -2]
10 # map_data['Recovered_24hr'] = df_recovered_fill.iloc[:, -1] - df_recovered_fill.iloc[:, -2]
11 # map_data['Confirmed_24hr'] = df_confirmed.iloc[:, -1] - df_confirmed.iloc[:, -2]
12 map_data.sort_values(by='Confirmed', ascending=False, inplace=True)

```

After data-processing, I use the `map_data` to make the *DataTable*. The style of the *DataTable* is easy to set by assigning value to *DataTable*'s parameters. We can set the data to be sortable by setting the `sort_action` equals to "native". The code of making a *DataTable* is now followed:

```

1 dt.DataTable(
2     data=map_data.to_dict('records'),
3     columns=[
4         {"name": i, "id": i, "deletable": False, "selectable": True} for i
5         in ['Province/State', 'Country/Region', 'Confirmed', 'Deaths', 'Recovered']
6     ],
7     fixed_rows={'headers': True, 'data': 0},
8     style_header={'backgroundColor': 'rgb(30, 30, 30)', 'fontWeight': 'bold'},
9     style_cell={
10         'backgroundColor': 'rgb(100, 100, 100)',
11         'color': colors['text'],
12         'maxWidth': 0,
13         'fontSize': 14,
14     },
15     style_table={'maxHeight': '510px', 'overflowY': 'auto'},
16     style_data={'whiteSpace': 'normal', 'height': 'auto', },
17     style_data_conditional=[
18         {'if': {'row_index': 'even'}, 'backgroundColor': 'rgb(60, 60, 60)', },
19         {'if': {'column_id': 'Confirmed'}, 'color': colors['confirmed_text'], 'fontWeight': 'bold'},
20         {'if': {'column_id': 'Deaths'}, 'color': colors['deaths_text'], 'fontWeight': 'bold'},
21         {'if': {'column_id': 'Recovered'}, 'color': colors['recovered_text'], 'fontWeight': 'bold'},
22     ],
23     style_cell_conditional=[
24         {'if': {'column_id': 'Province/State'}, 'width': '26%'},
25         {'if': {'column_id': 'Country/Region'}, 'width': '26%'},
26         {'if': {'column_id': 'Confirmed'}, 'width': '16%'},
27         {'if': {'column_id': 'Deaths'}, 'width': '11%'},
28         {'if': {'column_id': 'Recovered'}, 'width': '16%'},
29     ],
30     editable=False,
31     filter_action="native",
32     sort_action="native",
33     sort_mode="single",
34     row_selectable="single",
35     row_deletable=False,
36     selected_columns=[],
37     selected_rows=[],
38     page_current=0,
39     page_size=1000,
40     id='datatable',
41 ),

```

The generated DataTable contains all the countries' data we get including Confirmed, Deaths and Recovered cases. By clicking on the left side, we can get a clearer picture of the COVID-19 situation in each country by two graphs I design in the next part. The figure of the DataTable is below:

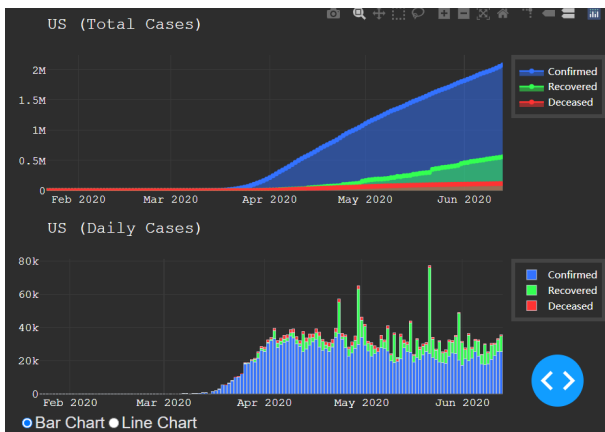
	Province/State	Country/Region	Confirmed	Deaths	Recovered
filter data...					
●		Spain	243605	27136	150376
●		Italy	236651	34301	174865
●		Peru	220749	6308	111724
●		France	189311	29339	69783
●		Germany	187267	8793	171970
●		Iran	184955	8730	146748
●		Turkey	176677	4792	150087
●		Chile	167355	3101	137296
●		Mexico	142690	16872	104078
●		Pakistan	132405	2551	50056
●		Saudi Arabia	123308	932	82548
●		Bangladesh	84379	1139	17828
●		Qatar	78416	70	55252
●	Hubei	China	68135	4512	63623
●		South Africa	65736	1423	36850

### 4.5.2 Graph of each specific country's COVID-19 cases

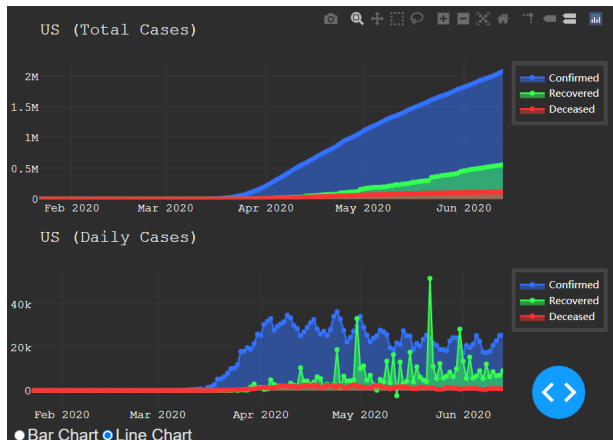
This is the last part of my project. Two graphs of the specific country's COVID-19 cases are made to make a clearer picture of the situation and trend of COVID-19. The most important callback function can be written as belowed code:

```
1 # output: id='line-graph', 'figure'
2 # output: id='bar-graph', 'figure'
3 # input: id='datatable', 'selected_rows'
4 # input: id='graph-line', 'value'
5 @app.callback(
6     [Output('line-graph', 'figure'),
7      Output('bar-graph', 'figure')],
8     [Input('datatable', 'selected_rows'),
9      Input('graph-line', 'value')])
10 def map_selection(selected_rows, graph_line):
11     if len(selected_rows) == 0:
12         fig1 = draw_single_country_scatter(df_confirmed_t, df_deaths_t, df_recovered_t, 0)
13         fig2 = draw_single_country_bar(df_confirmed_t, df_deaths_t, df_recovered_t, 0, graph_line)
14         return fig1, fig2
15     else:
16         fig1 = draw_single_country_scatter(df_confirmed_t, df_deaths_t, df_recovered_t, selected_rows[0])
17         fig2 = draw_single_country_bar(df_confirmed_t, df_deaths_t, df_recovered_t, selected_rows[0], graph_line)
18         return fig1, fig2
```

Besides, I made the daily cases chart into two form: bar chart and line chart. The viewer can choose each type of graph he prefers to see the trend from the COVID-19's outbreak to today. The graphs are shown below:



(f) Bar



(g) Line

## 5 Future Work

There exist two parts I want to improve:

(1) The first part is the data visualization. The map I make in the first part has no clear relevant to the Dashboard I make in the next part. Therefore, I want to make the map to be a part of Dashboard in particular as a part of the DataTable. When clicked on button on the left side of the DataTable, not only will the two graph be shown, but the map will also be updated to the clicked country's map with the COVID-19's data.

(2) The next part is to deploy the website on server which can ensure that everyone with internet access can view the Global COVID-19 Dashboard.