In [ ]:

```
!pip install plotly --upgrade
```

In [ ]:

```
import os

if not os.getenv("IS_TESTING"):
    # Automatically restart kernel after installs
    import IPython

    app = IPython.Application.instance()
    app.kernel.do_shutdown(True)
```

In [ ]:

```
!pip install pyTigerGraph
```

In [ ]:

```
import pyTigerGraph as tg

# Connect to the solution
conn = tg.TigerGraphConnection(host="https://covid19graph.i.tgcloud.io/", p
conn.apiToken = conn.getToken(conn.createSecret())
```

In [ ]:

```
print(conn.gsql('''
USE GRAPH MyGraph

DROP QUERY grabInfectionLocationDetails

CREATE QUERY grabInfectionLocationDetails() FOR GRAPH MyGraph SYNTAX v2 {

  TYPEDEF TUPLE <FLOAT lat, FLOAT lon, STRING infcase, STRING province, UIN

  HeapAccum<INFO> (1000, num_confirmed_cases DESC, population DESC) @@infor

  Seed = {City.*};
  Res = SELECT tgt FROM Seed:c - (CASE_IN_CITY:e)- InfectionCase:i -(CASE_I
        ACCUM @@information+=INFO(e.latitude, e.longitude, i.infection_case
  PRINT @@information;

}

INSTALL QUERY grabInfectionLocationDetails
'''))
```

In [ ]:

```
print(conn.gsql('''
USE GRAPH MyGraph

DROP QUERY getSearchStats

CREATE QUERY getSearchStats() {
    Seed = {SearchStat.*};

    PRINT Seed;
}
INSTALL QUERY getSearchStats

'''))
```

In [ ]:

```
print(conn.gsql('''
USE GRAPH MyGraph

DROP QUERY patientTravelLocations

CREATE QUERY patientTravelLocations(Vertex<Patient> p) FOR GRAPH MyGraph {
  TYPEDEF TUPLE <FLOAT lat, FLOAT lon, DATETIME visited_day, STRING travel_

  HeapAccum<INFORMATION> (1000, visited_day DESC) @@information;

  Seed = {p};

  Res = SELECT tgt FROM Seed:s - (PATIENT_TRAVELED:e) - TravelEvent:tgt
        ACCUM @@information+=INFORMATION(tgt.latitude, tgt.longitude, tgt.v

  PRINT @@information;

}

INSTALL QUERY patientTravelLocations
'''))
```

In [ ]:

```
#creating your dashboard
 #Install the necessary libraries

!pip install -q jupyter-dash
!pip install dash_bootstrap_components

# Import Pandas, Datetime, and Plotly Express

import pandas as pd
from datetime import datetime
import plotly.express as px

# Import Dash Libraries

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

In [ ]:

```
TG_YELLOW = "#FFCF9E"
TG_ORANGE = "#f5bd1f"
WHITE = "#fffff8"
BAR_LIGHTRED = "#ed8d84"
```

```
MAINSTYLE = {
  "background-color": "#FFFF99",
  'font': {'family': 'Roboto'},
}

# Links to logos and font styles

TG_LOGO = "https://i.ibb.co/gMMXcQB/Untitled-design-9.png"
DASH_LOGO = "https://rapids.ai/assets/images/Plotly_Dash_logo.png"
FONT_AWESOME = "https://use.fontawesome.com/releases/v5.8.1/css/all.css"
PLOTLY_LOGO = "https://dash.plotly.com/docs/assets/images/dark_plotly_dash_
```

In [ ]:

```
TG_YELLOW = "#FFCF9E"
TG_ORANGE = "#f5bd1f"
WHITE = "#fffff8"
BAR_LIGHTRED = "#ed8d84"

MAINSTYLE = {
  "background-color": "#FFFF99",
  'font': {'family': 'Roboto'},
}

# Links to logos and font styles

TG_LOGO = "https://i.ibb.co/gMMXcQB/Untitled-design-9.png"
DASH_LOGO = "https://rapids.ai/assets/images/Plotly_Dash_logo.png"
FONT_AWESOME = "https://use.fontawesome.com/releases/v5.8.1/css/all.css"
PLOTLY_LOGO = "https://dash.plotly.com/docs/assets/images/dark_plotly_dash_
sidebar = html.Div(
    [
        # A brief description

        html.Center(html.P(
            "A Plotly dashboard for TigerGraph's COVID-19 Starter Kit", cla
        )),
        html.Br(),

        html.Hr(style = {'borderColor':WHITE}),

        # The navbar itself, with three separate sections
        dbc.Nav(
            [
                dbc.NavLink("General Overview", href="/", active="exact", s
                dbc.NavLink("Patient Search", href="/page-1", active="exact
                dbc.NavLink("Infection Maps", href="/page-2", active="exact
            ],
            vertical=True,
            pills=True,
        ),

        html.Hr(style = {'borderColor':WHITE}),
        html.Br(),

        html.Br(),
        # The TigerGraph logo as well as a link to TG Cloud

        html.Center(dbc.Row(dbc.Col(html.Img(src=TG_LOGO, height="150px", s
        html.Center(html.B(html.A("TigerGraph Cloud", href="https://www.tig

    ],
    style=SIDEBAR_STYLE,
)
```

In [ ]:

```
#creating a general page
val = conn.runInstalledQuery("ageDistribution")[0]["@@ageMap"]
del val["2021"]
age_data = pd.DataFrame.from_dict(val.items())
age_data["age"] = [int(i) for i in age_data[0]]
age_data["frequency"] = age_data[1]
age_data = age_data.sort_values(by="age")

# Grab the search stats
search = conn.runInstalledQuery("getSearchStats")[0]["Seed"]
search_list = [[datetime.strptime(i["v_id"], "%Y-%m-%d"), i["attributes"]["

search_df = pd.DataFrame(search_list, columns=["date", "cold", "coronavirus
search_df = search_df.sort_values(by="date")

def createGeneralPage():
    row = html.Div([
          dbc.Row([
              html.H1("South Korean COVID-19 Statistics"),
          ], justify='center'),
          dbc.Col([
              dbc.Row([
                  dbc.Col([
                      dbc.Card(
                          [
                              dbc.CardBody(
                                  [
                                      html.H2(conn.getVertexCount("Patient"
                                      html.P("Total Patients")
                                  ]
                              ),
                          ],
                          style={"width": "15rem"},
                      )
                  ]),
                  dbc.Col([
                      dbc.Card(
                          [
                              dbc.CardBody(
                                  [
                                      html.H2(conn.getVertexCount("Infectio
                                      html.P("Total Infection Cases")
                                  ]
                              ),
                          ],
                          style={"width": "15rem", "margin-left": "-5rem"},
                      )
                  ], style = {}),
                  dbc.Col([
                      dbc.Card(
                    [
                        dbc.CardBody(
                            [
                                html.H2(conn.getVertexCount("Province
                                html.P("Total Provinces")
                            ]
                        ),
                    ],
                    style={"width": "15rem", "margin-left": "-10rem"}
                )
            ]),
            dbc.Col([
                dbc.Card(
                    [
                        dbc.CardBody(
                            [
                                html.H2(conn.getVertexCount("TravelEv
                                html.P("Total Travel Events")
                            ]
                        ),
                    ],
                    style={"width": "15rem", "margin-left": "-15rem"}
                )
            ]),
        ]),
        html.Br(),
        dbc.Row([
            dbc.Col([
                # Line Graph with Search Stats
                dbc.Card([
                    dbc.CardBody([
                        dcc.Graph(figure= px.line(search_df, x="date"
                    ], className="mb-3", style={"width": "30rem", "he
                ], style={"margin-left": "1rem", "width": "30rem"}),
            ]),
            dbc.Col([
                # Age Distribution
                dbc.Card([
                    dbc.CardBody([
                        dcc.Graph(figure=px.bar(age_data, "age", "fre
                    ], className="mb-3", style={"width": "30rem", "he
                ], style={"margin-left": "-7rem", "width": "30rem"}),
            ]),
        ]),
    ], style={"margin-left":"20rem"})
])
return row
```

In [ ]:

```
#creating patient search
  row = html.Div([
          dbc.Row([
              html.H1("Patient Search and Dashboard"),
          ], justify='center'),
          dbc.Col([
              dbc.Row([
                  dbc.Col([
                      dbc.Card(
                                [
                                    dbc.CardBody( # Patient Card
                                        [
                                            html.H4(f"Patient Number { patient['p
                                            dbc.ListGroup([
                                                dbc.ListGroupItem(f"Global Number
                                                dbc.ListGroupItem(f"Birth Year: {
                                                dbc.ListGroupItem(f"Sex: { patien
                                                dbc.ListGroupItem(f"Contact Numbe
                                                dbc.ListGroupItem(f"Disease: { pa
                                                dbc.ListGroupItem(f"State: { pati
                                            ])
                                        ]
                                    ),
                                ],
                                style={"width": "18rem"}, id="patient_card",
                            ),
                            html.Br(),
                            dbc.Card([
                                dbc.CardBody([ # Text Search
                                    dcc.Input(
                                        id = "search",
                                        placeholder='Enter a patient ID like 2000
                                        type='number',
                                        value='2000000205'
                                    ),
                                    dash_table.DataTable( # Interactive DataTable
                                        id='table',
                                        columns=[{"name": i, "id": i} for i in in
                                        data=inf_patients.to_dict('records'),
                                    )
                                ], className="mb-3", style={"width": "20rem"})
                            ], style={"margin-left": "-1rem", "width": "20rem"}),
                        ]),
                        dbc.Col([
                            dbc.Card([
                                dbc.CardBody([ # Patient Map
                                    html.H3("Locations Patient Visited"),
                                    dcc.Graph("travel_map", figure={"layout": {"h
                                ])
                            ], style={"width": "40rem"}),
                            html.Br(),
                            dbc.Card([
                                dbc.CardBody([ # Patient Timeline
                                    html.H3("Patient Timeline"),
                                    dcc.Graph("patient_timeline", figure={"layout
                                ])
                            ], style={"width": "40rem", "height": "22rem"})
                        ], style={"margin-left":"-40rem"})
                    ])
                ], style={"margin-left":"20rem"})
        ])
        return row
```

In [ ]:

```
#create the map section
res = conn.runInstalledQuery("grabInfectionLocationDetails")[0]["@@informat
df = pd.DataFrame(res)
```

```
def createMapSection():
    row = html.Div([
        dbc.Row([
            html.H1("COVID-19 Confirmed Cases Mapped"),
        ], justify='center'),
        dbc.Col([
            dbc.Row([
                dbc.Col([
                    dbc.Row([
                        dbc.Card([
                            dbc.CardBody([
                                dcc.Dropdown( # Pick Map Type
                                    id = "map_type",
                                    options=[
                                            {"label": "Street Map", "value"
                                            {"label": "Geometric Map", "val
                                            {"label": "Density Map", "value
                                    ],
                                    value="light",
                                    clearable=False
                                ),
                            ], className="mb-3", style={"width": "33rem"})
                        ], style={"margin-left": "1rem"}),
                    ]),
                    html.Br(),
                    dbc.Row([
                        dbc.Col([
                            dbc.Card([ # Map
                                dbc.CardBody([
                                    dcc.Graph(id="map_graph", figure={"layo
                                ])
                            ], style={"width": "33rem", "height": "39.5rem"
                        ]),
                    ])
                ]),
                dbc.Col([
                    dbc.Col([
                        dbc.Card([
                            dbc.CardBody([
                                dcc.Dropdown( # Pick Chart Type
                                    id = "graph_options",
                                    options=[
                                            {"label": "County vs. Number of
                                            {"label": "County vs. Number of
                                    ],
                                    value="county_cases",
                                    clearable=False
                                ),
                                dcc.Graph(id="bar_graph", figure={"layout":
                            ])
                        ], style={"width": "33rem", "height": "47rem", "mar
                    ])
                ])
            ], justify='center'),
        ], style={"margin-left":"20rem"})
    ])
    return row
```

In [ ]:
```python
app = JupyterDash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP], su

# Grab all of the content we created
mapContent = html.Div(createMapSection(), id="page-content")
patientContent = html.Div(createPatientSearch(), id='page2-content')
generalContent = html.Div(createGeneralPage(), id='page3-content')

# Create the map layout
app.layout = html.Div([dcc.Location(id="url"), sidebar, mapContent])
# app.scripts.config.serve_locally = False # Uncomment this for removing th
```

In [ ]:
```python
#callbacks
# The first callback: loads each page based on the selections in the sideba
@app.callback(Output("page-content", "children"),
              [Input("url", "pathname")],
             )

def render_page_content(pathname):

    # Direct to the appropriate content for each url
    if pathname == "/":
        return generalContent
    elif pathname == "/page-1":
        return patientContent
    elif pathname == "/page-2":
        return mapContent

    # If the user tries to reach a different page, return an error!
    return dbc.Jumbotron(
        [
            html.H1("404: Not found", className="text-danger"),
            html.Hr(),
            html.P(f"Uh oh! Unfortunately, the pathname {pathname} was unab
        ]
    )
```

In [ ]:
```python
# Second callback: changes the map based on the dropdown
@app.callback(
    Output('map_graph', 'figure'),
    Input('map_type', 'value'),
)
def update_graph(typ):
    res = conn.runInstalledQuery("grabInfectionLocationDetails")[0]["@@info
    df = pd.DataFrame(res)

    if typ == "light":
        # Send the street map
        fig = px.scatter_mapbox(df, lat="lat", lon="lon", size="num_confirm
        fig.update_layout(mapbox_style="open-street-map")
        fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
    elif typ == "dark":
        # Send the geo map
        fig = px.scatter_mapbox(df, lat="lat", lon="lon", size="num_confirm
        fig.update_layout(
            mapbox_style="white-bg",
            mapbox_layers=[
                {
                    "below": 'traces',
                    "sourcetype": "raster",
                    "sourceattribution": "United States Geological Survey",
                    "source": [
                        "https://basemap.nationalmap.gov/arcgis/rest/servic
                    ]
                }
              ])
        fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
    else:
        fig = px.density_mapbox(df, lat='lat', lon='lon', z='num_confirmed_
                        mapbox_style="stamen-terrain")
        fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
    return fig

# Third callback: Updates the bar graph based on the selected values from t
@app.callback(
    Output('bar_graph', 'figure'),
    Input('map_graph', 'selectedData'),
    Input('graph_options', 'value'),
)
def update_graph(data, graph_type):
    if data:
        if graph_type == "county_cases":
            df = pd.DataFrame({"num_confirmed_cases": [i["customdata"][0] f
            bar = px.bar(df, x="place", y="num_confirmed_cases", color="pop
            return bar
        elif graph_type == "3d_graph":
            df = pd.DataFrame({"num_confirmed_cases": [i["customdata"][0] f
            bar = px.scatter_3d(df, "num_confirmed_cases", "population", "a
            return bar
    else:
        res = conn.runInstalledQuery("grabInfectionLocationDetails")[0]["@@
        df = pd.DataFrame(res)
        if graph_type == "county_cases":
            bar = px.bar(df, x="infcase", y="num_confirmed_cases", color="p
            return bar
        elif graph_type == "3d_graph":
            bar = px.scatter_3d(df, "num_confirmed_cases", "population", "a
            return bar
```

In [ ]:

```
# Fourth callback: Updates the entire Patient Search to be responsive (mult
@app.callback(
    Output('table', 'data'),
    Output('patient_card', 'children'),
    Output('travel_map', 'figure'),
    Output('search', 'value'),
    Output('patient_timeline', 'figure'),
    Input('search', 'value'),
    Input('table', 'active_cell'),
    Input('table', 'data')
)
def update_graph(data, clickData, table_data):
    ctx = dash.callback_context
    if not ctx.triggered:
        triggered_widget = 'No clicks yet'
    else:
        triggered_widget = ctx.triggered[0]['prop_id'].split('.')[0]

    if triggered_widget == "search":
        inf_patients = pd.DataFrame(conn.runInstalledQuery("listPatients_In
        patient = str(data)

    elif triggered_widget == "table":
        col = clickData['column_id']
        row = clickData['row']
        inf_patients = pd.DataFrame(conn.runInstalledQuery("listPatients_In
        patient = str(table_data[row][col])
    else:
        res = "6100000035"
        inf_patients = pd.DataFrame(conn.runInstalledQuery("listPatients_In
        patient = res

    travel_locations = conn.runInstalledQuery("patientTravelLocations", par
    if travel_locations:
        travel_locations = pd.DataFrame(travel_locations)
        fig = px.scatter_mapbox(travel_locations, lat="lat", lon="lon")
        fig.update_layout(mapbox_style="open-street-map")
        fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
    else:
        travel_locations = pd.DataFrame(travel_locations)
        travel_locations["lat"] = []
        travel_locations["lon"] = []
        fig = px.scatter_mapbox(travel_locations, lat="lat", lon="lon")
        fig.update_layout(mapbox_style="open-street-map")
        fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

    patient_vertex = conn.getVerticesById("Patient", patient)[0]["attribute
    card = dbc.CardBody(
              [
                  html.H4(f"Patient Number { patient_vertex['patient_id'] }
                  dbc.ListGroup([
                      dbc.ListGroupItem(f"Global Number: { patient_vertex['
                      dbc.ListGroupItem(f"Birth Year: { patient_vertex['bir
                      dbc.ListGroupItem(f"Sex: { patient_vertex['sex'] }"),
                      dbc.ListGroupItem(f"Contact Number: { patient_vertex[
                      dbc.ListGroupItem(f"Disease: { patient_vertex['diseas
                      dbc.ListGroupItem(f"State: { patient_vertex['state']
                  ])
              ]
          )

    if patient_vertex["confirmed_date"] <= patient_vertex["released_date"]:
        timeline_df = pd.DataFrame([
            dict(State="Symptoms to Confirmation", Start=patient_vertex["sy
            dict(State="Confirmed to Release Date", Start=patient_vertex['c
            dict(State="Released to Now", Start=patient_vertex['released_da
        ])
    else: # If they're not released yet
        timeline_df = pd.DataFrame([
            dict(State="Symptoms to Confirmation", Start=patient_vertex["sy
            dict(State="Confirmed to Release Date", Start=patient_vertex['c
            dict(State="Released to Now", Start="2022", Finish="2022", Heal
        ])

    timeline_fig = px.timeline(timeline_df, x_start="Start", x_end="Finish"
    timeline_fig.update_yaxes(autorange="reversed") # otherwise tasks are l

    return inf_patients.to_dict('records'), card, fig, patient, timeline_fi
```

In [ ]:

```
RUN THE APP:
    app.run_server(mode='external')
```

In [ ]:

```
dash app running on:..
```