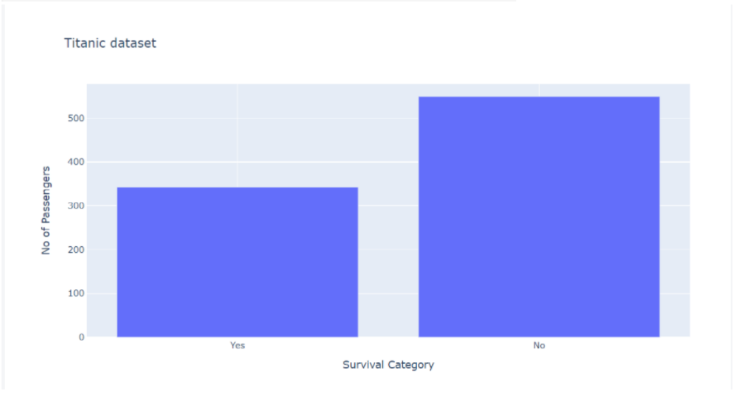
Plotly

```
import plotly.graph_objects as go
import pandas as pd

df = pd.read_csv("Basic_Titanic_Dataset.csv")
```

BAR CHART

```
survival_label = []
survival_label.append(df['Survived'].where(df['Survived']==1).count())
survival_label.append(df['Survived'].where(df['Survived']==0).count())
```

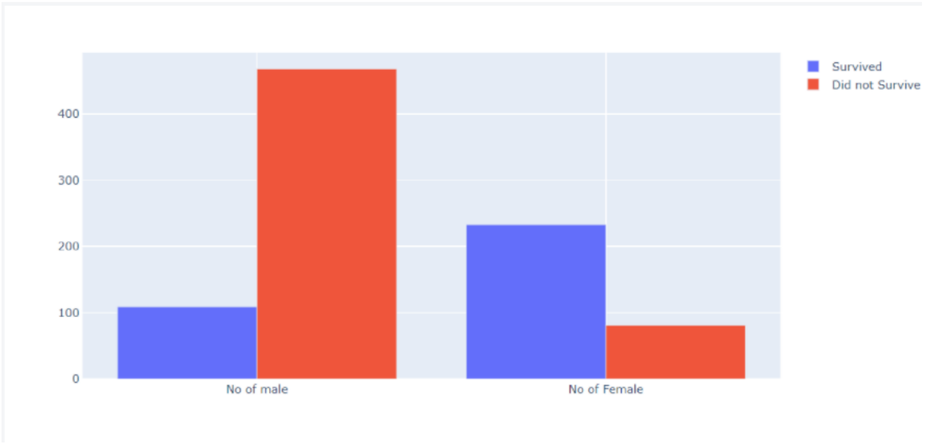
```
fig = go.Figure([go.Bar(
    x = ['Yes','No'],
    y = survival_label,
)])
fig.update_layout(
    title_text = "Titanic dataset",
    xaxis = dict(
        title = "Survival Category"
    ),
    yaxis = dict(
        title = "No of Passengers"
    )
)
fig.show()
```



Grouped Bar Chart

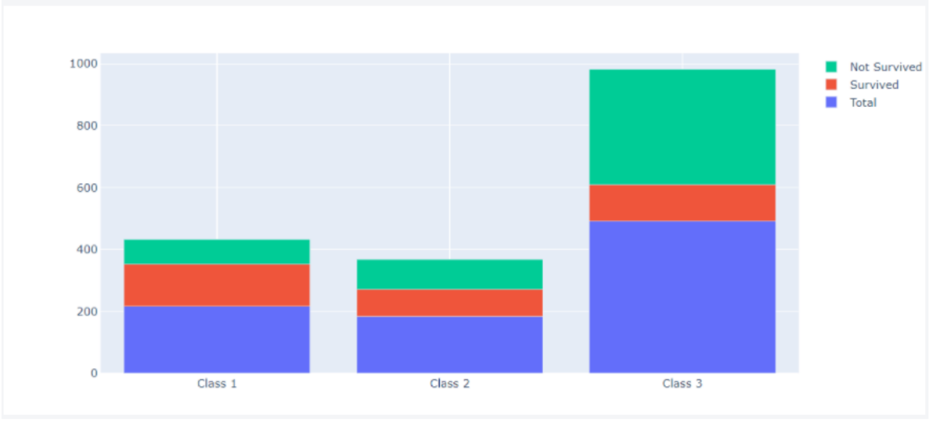
```
survival_yes_cat = []
survival_yes_cat.append(df['Survived'].where((df['Survived']==1) & (df['Sex']=='male')).count())
survival_yes_cat.append(df['Survived'].where((df['Survived']==1) & (df['Sex']=='female')).count())
survival_no_cat = []
survival_no_cat.append(df['Survived'].where((df['Survived']==0) & (df['Sex']=='male')).count())
survival_no_cat.append(df['Survived'].where((df['Survived']==0) & (df['Sex']=='female')).count())
```

```
fig = go.Figure(
    data = [
        go.Bar(name = 'Survived', x = ['No of male','No of Female'], y = survival_yes_cat),
        go.Bar(name = 'Did not Survive', x = ['No of male','No of Female'], y = survival_no_cat),
    ]
)
fig.update_layout(barmode = 'group')
fig.show()
```



STACKED BAR CHART

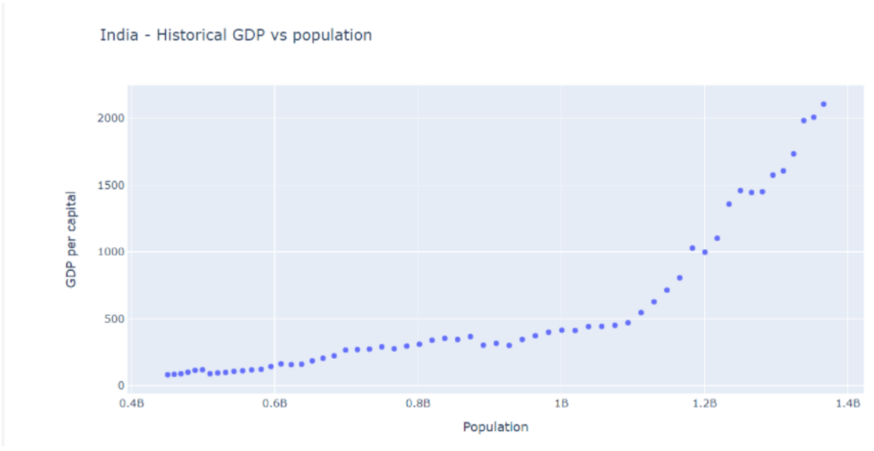
```
fig.update_layout(barmode = 'stack')
```



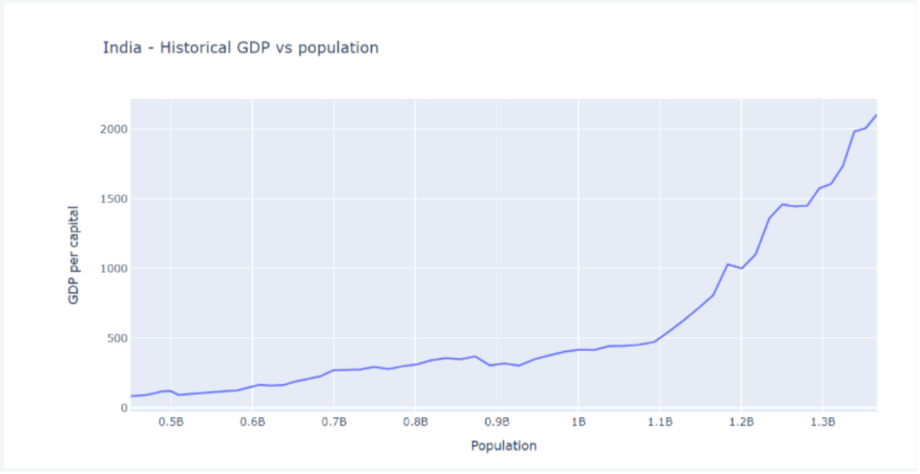
Scatter Plots

```
fig = go.Figure()

# Add traces
fig.add_trace(go.Scatter(x=df['Population'], y=df['GDP_Per_Capital'],
                    mode='markers',
                    name='markers'))

fig.show()
```

```
fig.update_layout(title_text = 'India - Historical GDP vs population',
                  xaxis = dict(
                      title='Population'
                  ),
                  yaxis = dict(
                      title = 'GDP per capital'
                  ))
fig.show()
```

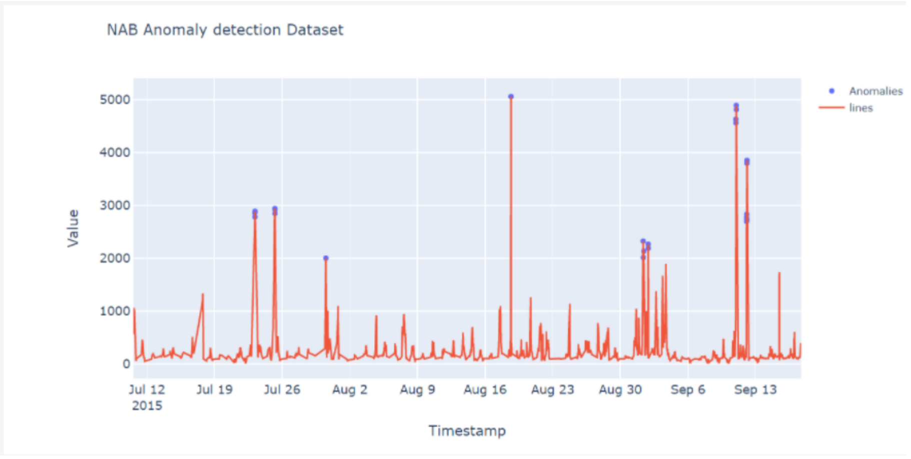


Now, you can change the same figure container to plot a line graph by only changing the value of mode and marker to the value 'line'. The output is as follows:



```
df_anomalous_pts = df.loc[df.value > 2000]
fig = go.Figure()

# Add traces
fig.add_trace(go.Scatter(x=df_anomalous_pts['timestamp'], y=df_anomalous_pts['value'],
                    mode='markers',
                    name='Anomalies'))
fig.add_trace(go.Scatter(x=df['timestamp'], y=df['value'],
                    mode='lines',
                    name='lines'))
```

```
fig.update_layout(title_text='NAB Anomaly detection Dataset',
                  xaxis=dict(
                             title='Timestamp',
                             titlefont_size=16,
                             tickfont_size=14,
                         ),
                   yaxis=dict(
                             title='Value',
                             titlefont_size=16,
                             tickfont_size=14,
                         ),
                 )
```



```
trace2 = go.Scatter(
                    x = df2002.index,
                    y = df2002.views,
                    mode = "lines+markers",
                    name = "TED2002",
                    marker = dict(color = 'rgba(255, 128, 2, 0.8)'),
                    text= df2002.main_speaker)
```

Feedback:
Yes, text represents the hovered text on the graph.

x - data to be plotted on the X Axis

y - data to be plotted on the Y Axis

mode - Style of the trace, whether line or marker

name - Legend for the graph

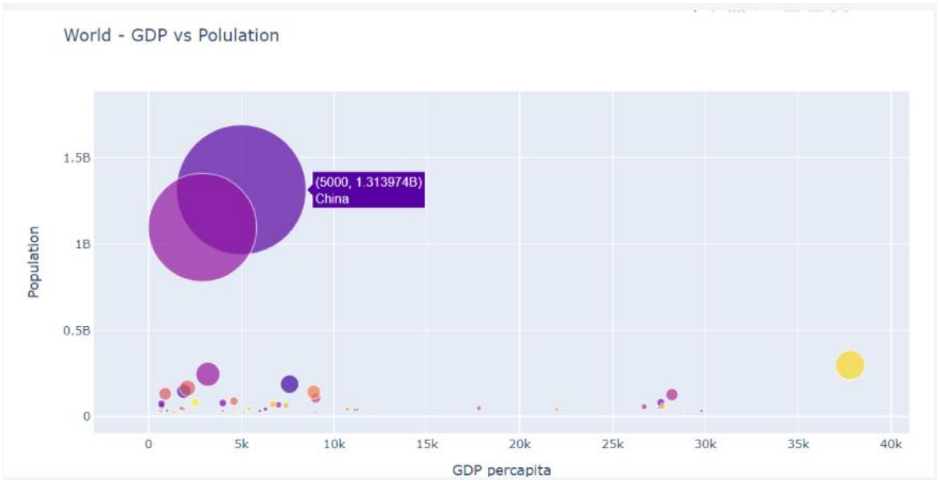
marker - style attributes for the mode

text - hovered text

BUBBLE CHART

```
#Add traces
fig = go.Figure()

# Add traces
fig.add_trace(go.Scatter(x=filter_df['GDP ($ per capita)'], y=filter_df['Population'],
                    mode='markers',
                    name='markers',
                    marker= dict(size= (filter_df['Population']/10000000),
                         color = count)
                        ))
```

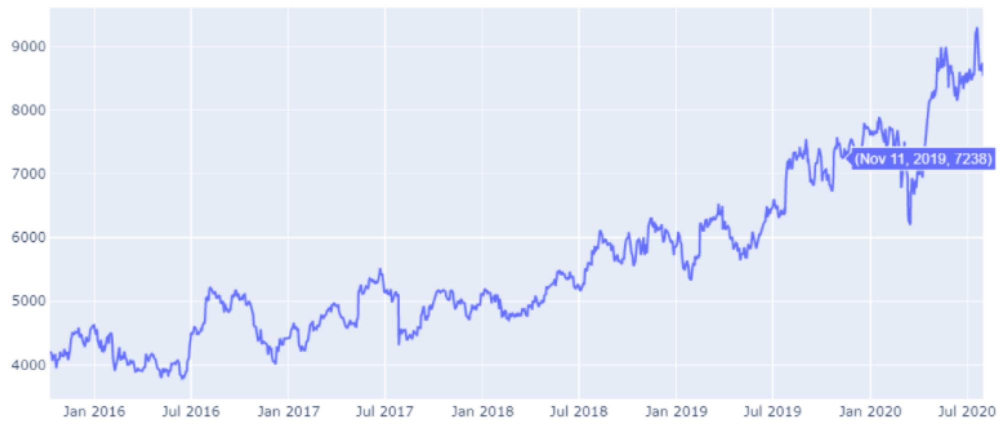
```
fig.update_layout(title_text='World - GDP vs Polulation',
                  xaxis = dict(title = 'GDP percapita'),
                  yaxis = dict(title = 'Population'))
```



TIMESERIES

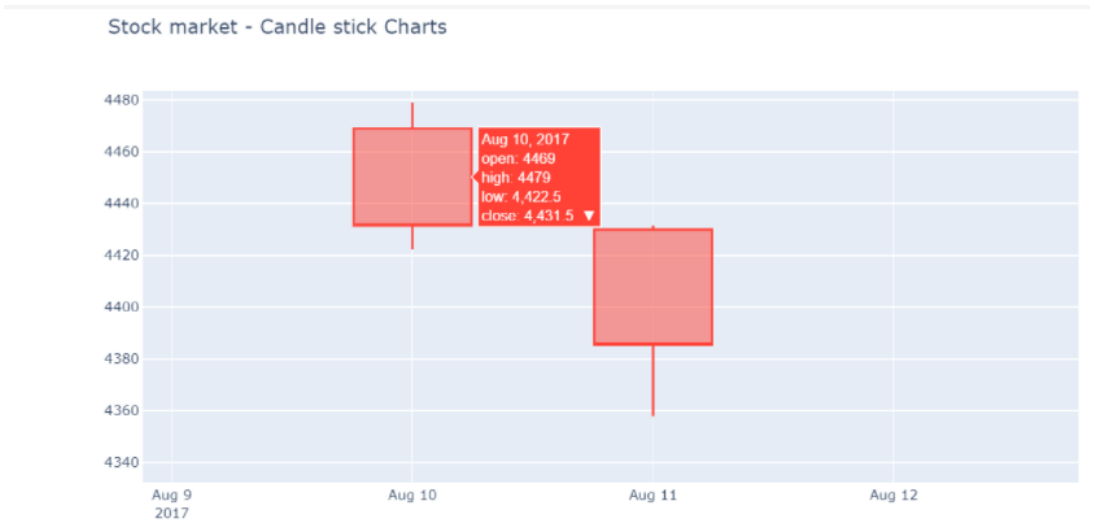
Timeseries

```
In [6]: #create the figure container with data object. The trace is a scatter chart
        fig  = go.Figure(go.Scatter(
            x = list(df['Date']),
            y = list(df['Close'])
        ))

        #plot the figure container
        fig.show()
```



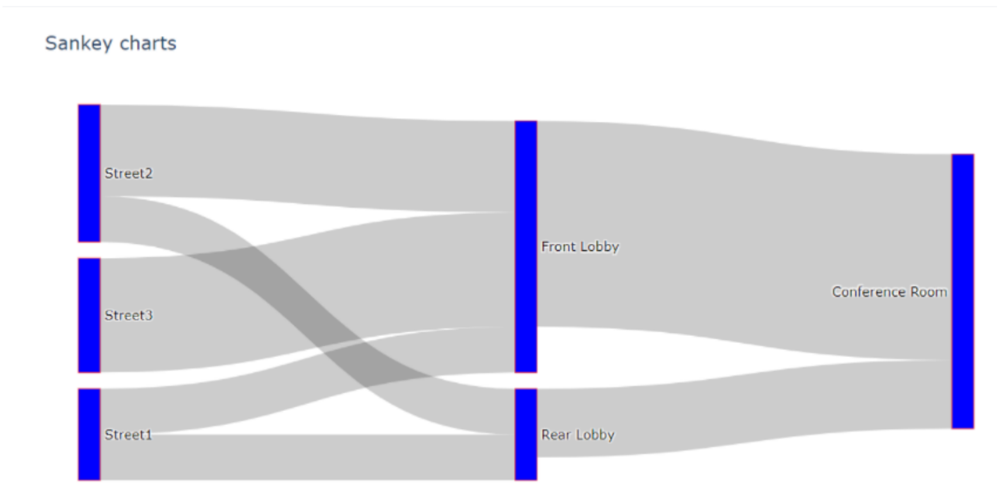
CANDLESTICK

```
#create the figure container with data object. The trace is a candlestick chart
fig = go.Figure(data = [ go.Candlestick(
    x = df['Date'],
    open  = df['Open'],
    high = df['High'],
    low = df['Low'],
    close = df['Close']
)])

# create the layout object with rangeslider value as false
fig.update_layout (title_text = "Stock market - Candle stick Charts",
                  xaxis = dict(
                      rangeslider = dict(
                          visible = False
                      )
                  ))

#plot the figure container
fig.show()
```



SANKY CHARTS

```
locations = ['Street1',
             'Street2',
             'Street3',
             'Front Lobby',
             'Rear Lobby',
             'Conference Room'
            ]
fig = go.Figure(data = [
    go.Sankey(
        node = dict(
            pad = 15,
            thickness = 20,
            line = dict(color = "black",width = 0.5),
            label = locations,
            color = "blue"
        ),
        link = dict(source = [0, 0, 1, 1, 2, 3, 4],
                   target = [3,4,3,4,3,5,5],
                  value = [2,2,4,2,5,9,3])
    )
])
```



BUTTONS AND DROPDOWNS

```
fig.add_trace(
    go.Scatter(
        x = list(df.index),
        y = list(df.High),
        name = "High",
        line = dict(color = "#33CFA5")
    )
)
```

```
fig.add_trace(
    go.Scatter(
        x = list(df.index),
        y = [df.High.mean()] * len(df.index),
        name = "High Average",
        line = dict(color = "#33CFA5", dash = "dash")
    )
```

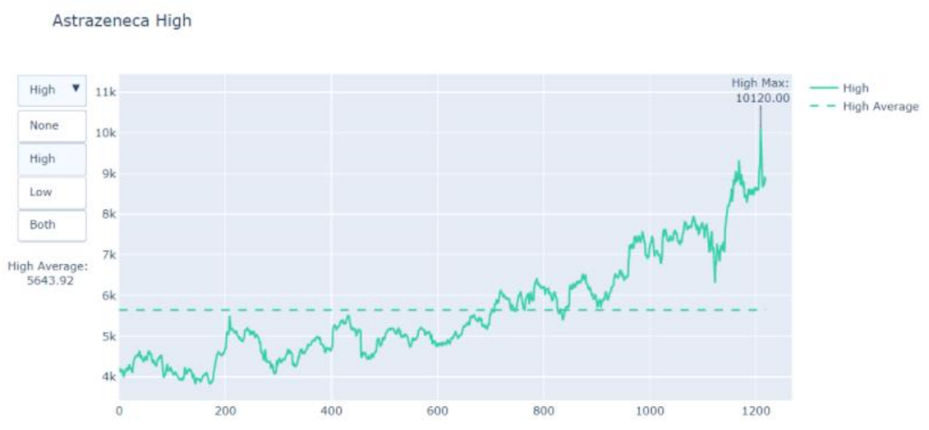
```
fig.add_trace(
    go.Scatter(
        x = list(df.index),
        y = list(df.Low),
        name = "Low",
        line = dict(color = "#F06A6A")
    )
)
```

```
    fig.add_trace(
        go.Scatter(
            x = list(df.index),
            y = [df.Low.mean()] * len(df.index),
            name = "Low Average",
            line = dict(color = "#F06A6A", dash = "dash")
        )
```

```
high_annotations = [dict(x="2015-05-01",
                         y=df.High.mean(),
                         xref="x", yref="y",
                         text="High Average:<br> %.2f" % df.High.mean(),
                         ax=0, ay=-40),
                    dict(x=df.High.idxmax(),
                         y=df.High.max(),
                         xref="x", yref="y",
                         text="High Max:<br> %.2f" % df.High.max(),
                         ax=0, ay=-40)]
```

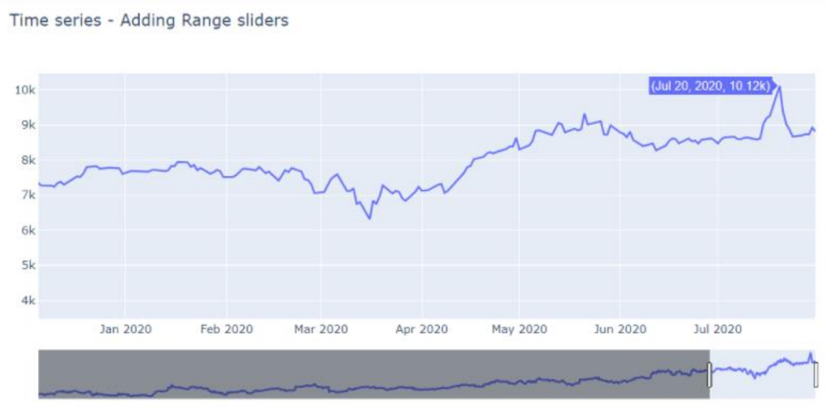
```
low_annotations = [dict(x="2015-05-01",
                        y=df.Low.mean(),
                        xref="x", yref="y",
                        text="Low Average:<br> %.2f" % df.Low.mean(),
                        ax=-40, ay=40),
                   dict(x=df.High.idxmin(),
                        y=df.Low.min(),
                        xref="x", yref="y",
                        text="Low Min:<br> %.2f" % df.Low.min(),
                        ax=0, ay=40)]
```

```
fig.update_layout(
    updatemenus = [
        dict(
            type = "dropdown",
           buttons=list([
                dict(label="None",
                     method="update",
                     args=[{"visible": [True, False, True, False]},
                           {"title": "Astrazeneca",
                            "annotations": []}]),
                dict(label="High",
                     method="update",
                     args=[{"visible": [True, True, False, False]},
                           {"title": "Astrazeneca High",
                            "annotations": high_annotations}]),
                dict(label="Low",
                     method="update",
                     args=[{"visible": [False, False, True, True]},
                           {"title": "Astrazeneca Low",
                            "annotations": low_annotations}]),
                dict(label="Both",
                     method="update",
                     args=[{"visible": [True, True, True, True]},
                           {"title": "Astrazeneca",
                            "annotations": high_annotations + low_annotations}]),
            ]),
        )
    ]
)
```



RANGESLIDERS

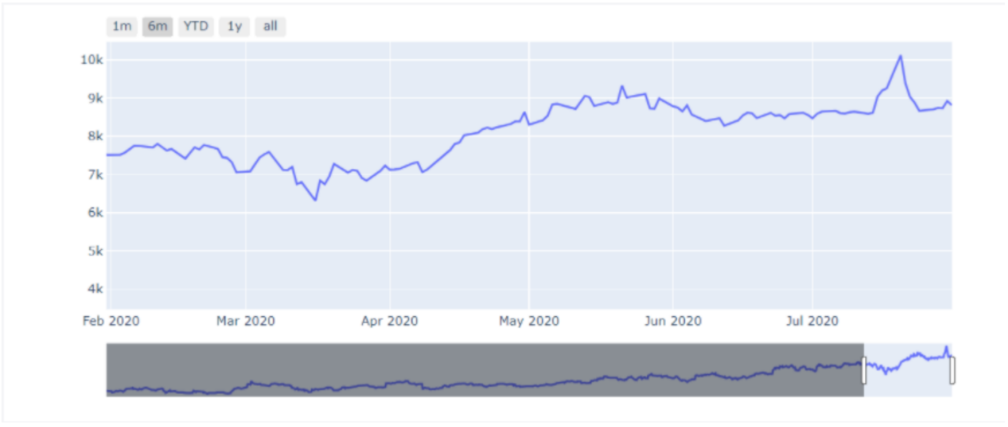
```
fig.update_layout(
    title_text = "Time series - Adding Range sliders",
    xaxis = dict(
        rangeslider = dict(
            visible = True
        )
    )
)
```



BUTTON WITH RANGESLIDER

- **Label** - Name of button
- **Step** - Movement bandwidth
- **Stepmode** - Direction of movement
- **Count** - Shows the amount of data to be displayed based on the step and the stepmode

```
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                     label="1m",
                     step="month",
                     stepmode="backward"),
                dict(count=6,
                     label="6m",
                     step="month",
                     stepmode="backward"),
                dict(count=1,
                     label="YTD",
                     step="year",
                     stepmode="todate"),
                dict(count=1,
                     label="1y",
                     step="year",
                     stepmode="backward"),
                dict(step="all")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)
```



3D CHARTS

```
fig = go.Figure(data=[go.Mesh3d(x=df['Temperature'],y=df['Humidity'],z=df['Wind_Speed'],
                    opacity=0.5,
                    color='rgba(244,22,100,0.6)'
                   )])
```

```
fig.update_layout(
    width=600,
    height=700,
    autosize=False,
    margin=dict(r=20, l=10, b=10, t=10),
    template="plotly_white",
)

# Update 3D scene options
fig.update_scenes(
    aspectratio=dict(x=1, y=1, z=1),
    aspectmode="manual"
)
```

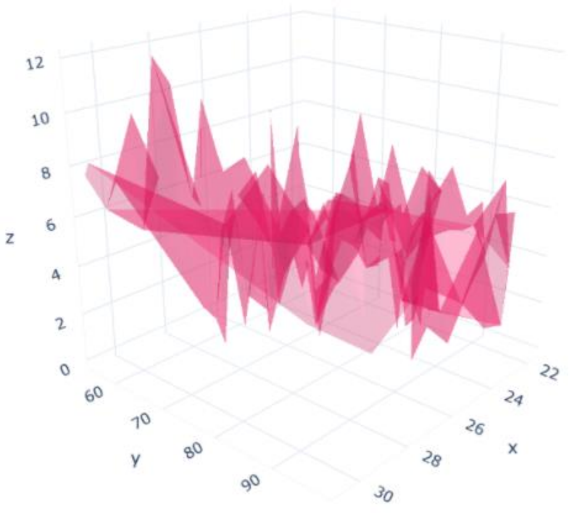


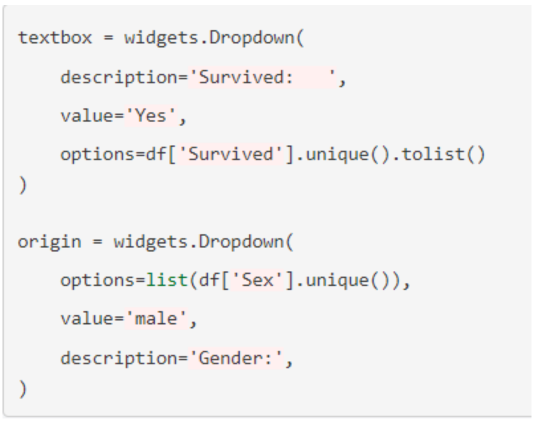
FIGURE WIDGETS

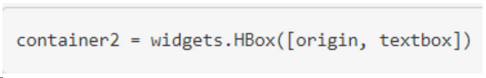
```
from ipywidgets import widgets
```

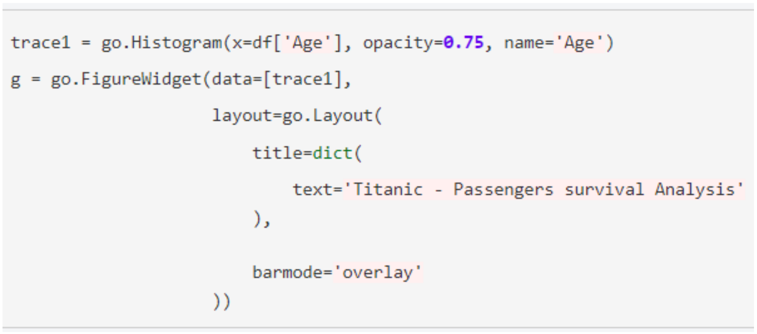
```
pclass = widgets.IntSlider(
    value=1,
    min=1,
    max=3,
    step=1,
    description='Passenger Class:',
    continuous_update=False
)
```

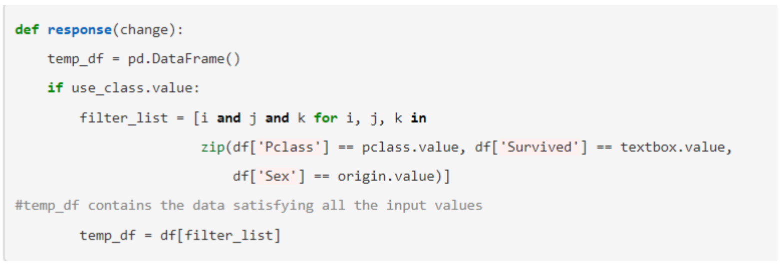
```
use_class = widgets.Checkbox(
    description='Enable Class: ',
    value=True,
)
```

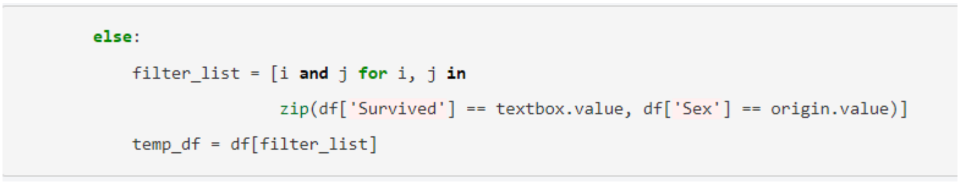
```
container = widgets.HBox(children=[use_class, pclass])
```

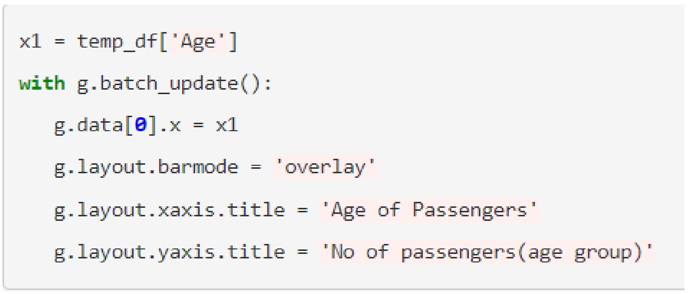

```
textbox = widgets.Dropdown(
    description='Survived:    ',
    value='Yes',
    options=df['Survived'].unique().tolist()
)

origin = widgets.Dropdown(
    options=list(df['Sex'].unique()),
    value='male',
    description='Gender:',
)
```

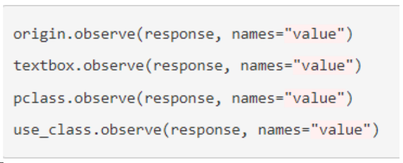

```
container2 = widgets.HBox([origin, textbox])
```

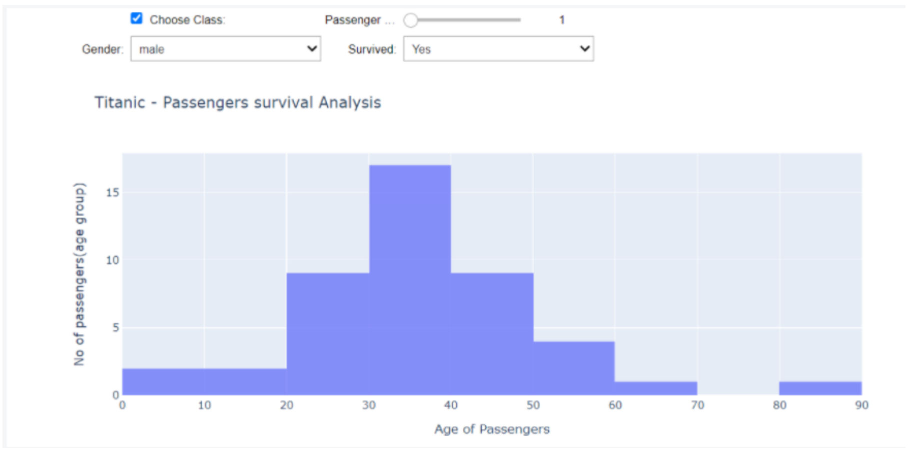

```
trace1 = go.Histogram(x=df['Age'], opacity=0.75, name='Age')
g = go.FigureWidget(data=[trace1],
                    layout=go.Layout(
                        title=dict(
                            text='Titanic - Passengers survival Analysis'
                        ),

                        barmode='overlay'
                    ))
```


```
def response(change):
    temp_df = pd.DataFrame()
    if use_class.value:
        filter_list = [i and j and k for i, j, k in
                       zip(df['Pclass'] == pclass.value, df['Survived'] == textbox.value,
                           df['Sex'] == origin.value)]
#temp_df contains the data satisfying all the input values
        temp_df = df[filter_list]
```


```
    else:
        filter_list = [i and j for i, j in
                       zip(df['Survived'] == textbox.value, df['Sex'] == origin.value)]
        temp_df = df[filter_list]
```


```
x1 = temp_df['Age']
with g.batch_update():
    g.data[0].x = x1
    g.layout.barmode = 'overlay'
    g.layout.xaxis.title = 'Age of Passengers'
    g.layout.yaxis.title = 'No of passengers(age group)'
```


```
origin.observe(response, names="value")
textbox.observe(response, names="value")
pclass.observe(response, names="value")
use_class.observe(response, names="value")
```



☑ Choose Class: Passenger ... 1

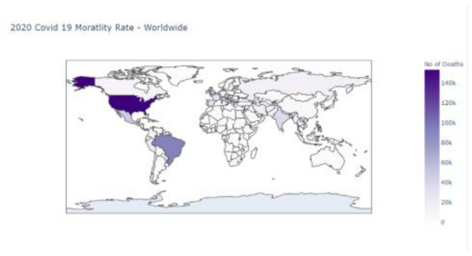
Gender: male Survived: Yes

Titanic - Passengers survival Analysis
No of passengers(age group)
0
5
10
15
0
10
20
30
40
50
60
70
80
90
Age of Passengers

MAPS

- **Location** - The input to this parameter is a list of country names or if the map is confined to a particular country then it should be set to names of cities in the corresponding state.
- **Z** - The input to this parameter is the data that is to be plotted on the map with respect to the scale, in our case it is the mortality rate.
- **Locationmode** - If it is a global map then we give the value as country names; else, if it is confined to a particular state, then we give the state code.
- **Colour Scale** - This corresponds to the colour of the scale, which will be used on the map.
- **Colorbar_title** - The value of this parameter is the title of the graph.

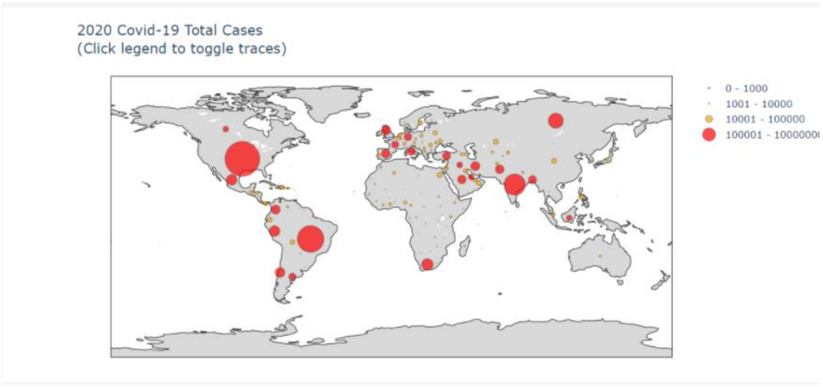
```
fig = go.Figure(data = go.Choropleth(
    locations = df['location'].values.tolist(),
    z = df['total_deaths'].values.astype(float),
    locationmode = 'country names',
    colorscale = 'purples',
    colorbar_title = 'No of Deaths',
))
```



BUBBLE MAPS

```
df['text'] = df['location'] + '<br>Total Cases ' + (df['total_cases']/1e5).astype(str)+' million'
limits = [(0,1000),(1001,10000),(10001,100000),(100001,10000000)]
colors = ["green","yellow","orange","red"]
scale = 5000
```

```python
fig = go.Figure()

for i in range(len(limits)):
    lim = limits[i]
    df_sub = df[(df['total_cases']>=lim[0]) & (df['total_cases']<=lim[1])]
    fig.add_trace(go.Scattergeo(
        locationmode = 'country names',
        lon = df_sub['longitude'],
        lat = df_sub['latitude'],
        text = df_sub['text'],
        marker = dict(
            size = df_sub['total_cases']/scale,
            color = colors[i],
            line_color='rgb(40,40,40)',
            line_width=0.5,
            sizemode = 'area'
        ),
        name = '{0} - {1}'.format(lim[0],lim[1])))
```



ANIMATIONS

- **Label** - Name on the button
- **Method** - Enter the respective plotly.js function. Since it is animate, we will assign animate as the value (discussed in the segment on 'Introduction to Custom Controls')
- **fromcurrent** - will set whether hitting the 'Play' button will resume animation from the current frame or from the beginning. We choose to resume from the current frame (which becomes significant since we're about to add a pause button).
- **transition** - defines the transition between frames
- **frame** - defines the state of the frame itself
- **Arguments** - Consist of a dictionary which takes the parameters, transition and frame.

```
fig = go.Figure(
    data=[go.Scatter(x=[0], y=[0])],
    layout=go.Layout(
        xaxis=dict(range = [dateList[0], dateList[-1]]),
        yaxis=dict(range = [0, 1 + max(Gdp)]),
        title="Animated Chart",
        updatemenus=[
            dict(
            type="buttons",
            buttons=list([
                dict(label="Play",
                          method="animate",
                         args= [None,
                         dict(fromcurrent = True,
                              transition = dict(duration = 1000),
                              frame = dict(duration = 1000)
                             )
                        ]),
                 dict(label="Pause",
                          method="animate",
                         args= [[None], dict(mode = 'immediate')])
                    ])
            )
        ]
    ),
    frames=framesList
)
```