



# Dropdown Menus in Python

How to add dropdowns to update Plotly chart attributes in Python.

Plotly Studio: Transform any dataset into an interactive data application in minutes with AI. [Sign up for early access now.](https://plotly.com/studio/?utm_medium=graphing-libraries&utm_campaign=studio_early_access&utm_content=sidebar) ([https://plotly.com/studio/?utm\\_medium=graphing-libraries&utm\\_campaign=studio\\_early\\_access&utm\\_content=sidebar](https://plotly.com/studio/?utm_medium=graphing-libraries&utm_campaign=studio_early_access&utm_content=sidebar))

## Methods

The `updatemenu` method (<https://plotly.com/python/reference/layout/updatemenus/#layout-updatemenus-buttons-method>) determines which `plotly.js` function (<https://plotly.com/javascript/plotlyjs-function-reference/>) will be used to modify the chart. There are 4 possible methods:

- "restyle": modify data or data attributes
- "relayout": modify layout attributes
- "update": modify data **and** layout attributes
- "animate": start or pause an [animation](https://plotly.com/python/#animations) (<https://plotly.com/python/#animations>).

## Restyle Dropdown

The "restyle" method should be used when modifying the data and data attributes of the graph.

## Update One Data Attribute

This example demonstrates how to update a single data attribute: chart type with the "restyle" method.

```

import plotly.graph_objects as go

import pandas as pd

# Load dataset
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/volcano.csv")

# create figure
fig = go.Figure()

# Add surface trace
fig.add_trace(go.Surface(z=df.values.tolist(), colorscale="Viridis"))

# Update plot sizing
fig.update_layout(
    width=800,
    height=900,
    autosize=False,
    margin=dict(t=0, b=0, l=0, r=0),
    template="plotly_white",
)

# Update 3D scene options
fig.update_scenes(
    aspectratio=dict(x=1, y=1, z=0.7),
    aspectmode="manual"
)

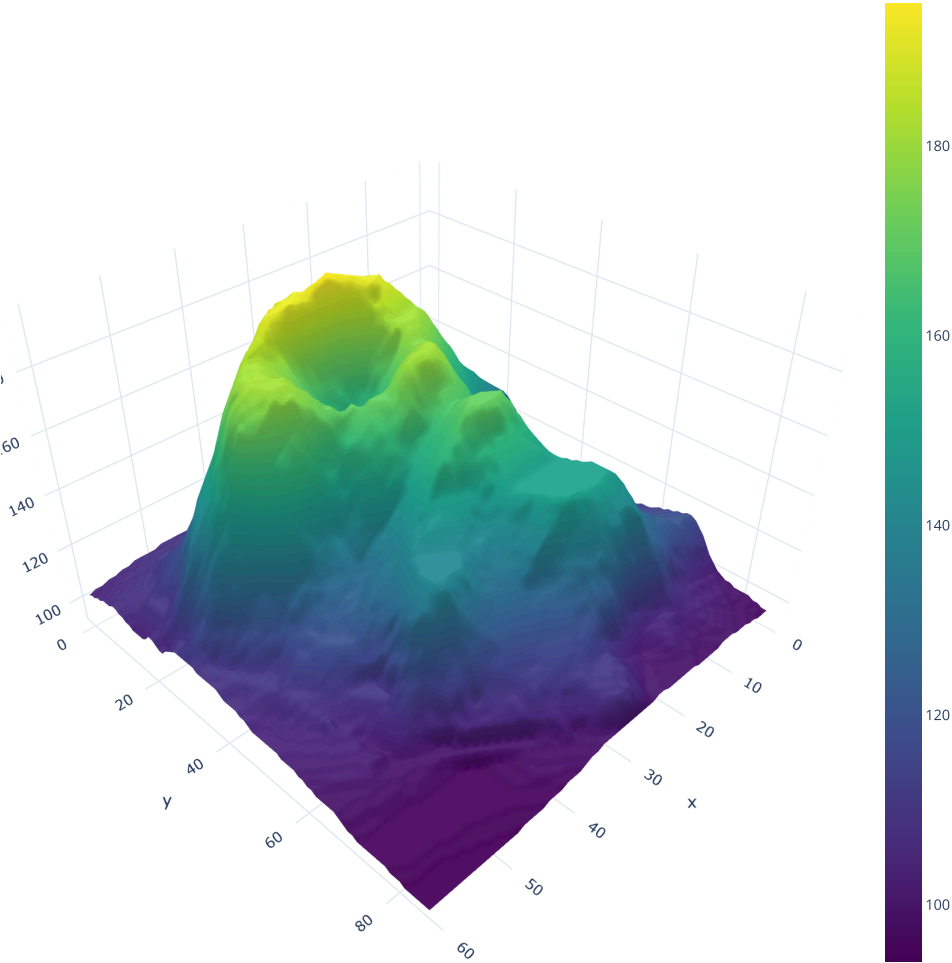
# Add dropdown
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=["type", "surface"],
                    label="3D Surface",
                    method="restyle"
                ),
                dict(
                    args=["type", "heatmap"],
                    label="Heatmap",
                    method="restyle"
                )
            ]),
            direction="down",
            pad={"r": 10, "t": 10},
            showactive=True,
            x=0.1,
            xanchor="left",
            y=1.1,
            yanchor="top"
        ),
    ]
)

# Add annotation
fig.update_layout(
    annotations=[
        dict(text="Trace type:", showarrow=False,
            x=0, y=1.085, yref="paper", align="left")
    ]
)

fig.show()

```

Trace type: 3D Surface ▼



## Update Several Data Attributes

This example demonstrates how to update several data attributes: colorscale, colorscale direction, and line display with the "restyle" method.

```

import plotly.graph_objects as go

import pandas as pd

# Load dataset
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/volcano.csv")

# Create figure
fig = go.Figure()

# Add surface trace
fig.add_trace(go.Heatmap(z=df.values.tolist(), colorscale="Viridis"))

# Update plot sizing
fig.update_layout(
    width=800,
    height=900,
    autosize=False,
    margin=dict(t=100, b=0, l=0, r=0),
)

# Update 3D scene options
fig.update_scenes(
    aspectratio=dict(x=1, y=1, z=0.7),
    aspectmode="manual"
)

# Add dropdowns
button_layer_1_height = 1.08
fig.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=["colorscale", "Viridis"],
                    label="Viridis",
                    method="restyle"
                ),
                dict(
                    args=["colorscale", "Cividis"],
                    label="Cividis",
                    method="restyle"
                ),
                dict(
                    args=["colorscale", "Blues"],
                    label="Blues",
                    method="restyle"
                ),
                dict(
                    args=["colorscale", "Greens"],
                    label="Greens",
                    method="restyle"
                ),
            ]),
            direction="down",
            pad={"r": 10, "t": 10},
            showactive=True,
            x=0.1,
            xanchor="left",
            y=button_layer_1_height,
            yanchor="top"
        ),
        dict(
            buttons=list([
                dict(
                    args=["reversescale", False],
                    label="False",
                    method="restyle"
                ),
                dict(
                    args=["reversescale", True],
                    label="True",
                    method="restyle"
                ),
            ]),
            direction="down",
            pad={"r": 10, "t": 10},
            showactive=True,

```

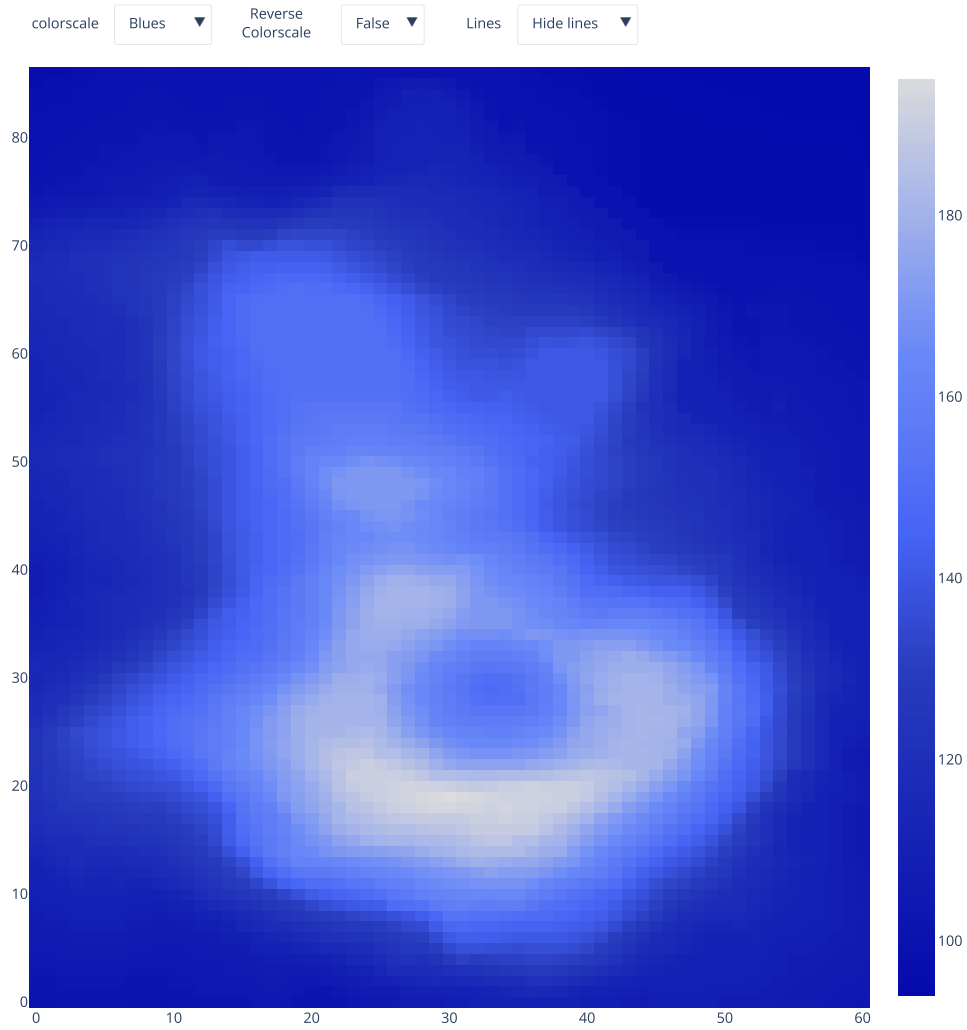
```

        x=0.37,
        xanchor="left",
        y=button_layer_1_height,
        yanchor="top"
    ),
    dict(
        buttons=list([
            dict(
                args=[{"contours.showlines": False, "type": "contour"}],
                label="Hide lines",
                method="restyle"
            ),
            dict(
                args=[{"contours.showlines": True, "type": "contour"}],
                label="Show lines",
                method="restyle"
            ),
        ]),
        direction="down",
        pad={"r": 10, "t": 10},
        showactive=True,
        x=0.58,
        xanchor="left",
        y=button_layer_1_height,
        yanchor="top"
    ),
]
)

fig.update_layout(
    annotations=[
        dict(text="colormap", x=0, xref="paper", y=1.06, yref="paper",
              align="left", showarrow=False),
        dict(text="Reverse<br>Colormap", x=0.25, xref="paper", y=1.07,
              yref="paper", showarrow=False),
        dict(text="Lines", x=0.54, xref="paper", y=1.06, yref="paper",
              showarrow=False)
    ])

fig.show()

```



## Relayout Dropdown

The "relayout" method should be used when modifying the layout attributes of the graph.

## Update One Layout Attribute

This example demonstrates how to update a layout attribute: chart type with the "relayout" method.

```

import plotly.graph_objects as go

# Generate dataset
import numpy as np
np.random.seed(1)

x0 = np.random.normal(2, 0.4, 400)
y0 = np.random.normal(2, 0.4, 400)
x1 = np.random.normal(3, 0.6, 600)
y1 = np.random.normal(6, 0.4, 400)
x2 = np.random.normal(4, 0.2, 200)
y2 = np.random.normal(4, 0.4, 200)

# Create figure
fig = go.Figure()

# Add traces
fig.add_trace(
    go.Scatter(
        x=x0,
        y=y0,
        mode="markers",
        marker=dict(color="DarkOrange")
    )
)

fig.add_trace(
    go.Scatter(
        x=x1,
        y=y1,
        mode="markers",
        marker=dict(color="Crimson")
    )
)

fig.add_trace(
    go.Scatter(
        x=x2,
        y=y2,
        mode="markers",
        marker=dict(color="RebeccaPurple")
    )
)

# Add buttons that add shapes
cluster0 = [dict(type="circle",
                  xref="x", yref="y",
                  x0=min(x0), y0=min(y0),
                  x1=max(x0), y1=max(y0),
                  line=dict(color="DarkOrange"))]

cluster1 = [dict(type="circle",
                  xref="x", yref="y",
                  x0=min(x1), y0=min(y1),
                  x1=max(x1), y1=max(y1),
                  line=dict(color="Crimson"))]

cluster2 = [dict(type="circle",
                  xref="x", yref="y",
                  x0=min(x2), y0=min(y2),
                  x1=max(x2), y1=max(y2),
                  line=dict(color="RebeccaPurple"))]

fig.update_layout(
    updatemenus=[
        dict(buttons=list([
            dict(label="None",
                  method="relayout",
                  args=["shapes", []]),
            dict(label="Cluster 0",
                  method="relayout",
                  args=["shapes", cluster0]),
            dict(label="Cluster 1",
                  method="relayout",
                  args=["shapes", cluster1]),
            dict(label="Cluster 2",
                  method="relayout",
                  args=["shapes", cluster2]),
            dict(label="All",
                  method="relayout",

```

```

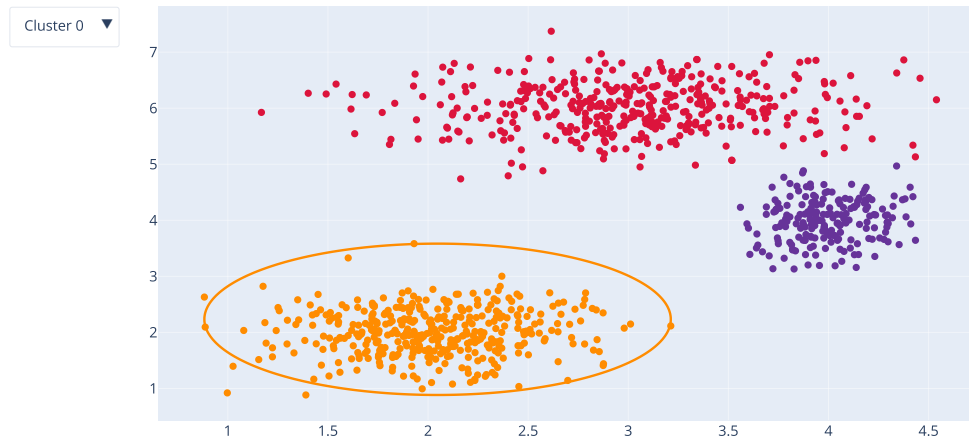
        args=["shapes", cluster0 + cluster1 + cluster2])
    ),
)
]
)

# Update remaining layout properties
fig.update_layout(
    title_text="Highlight Clusters",
    showlegend=False,
)

fig.show()

```

### Highlight Clusters



### Update Dropdown

The "update" method should be used when modifying the data and layout sections of the graph.

This example demonstrates how to update which traces are displayed while simultaneously updating layout attributes such as the chart title and annotations.



```

import plotly.graph_objects as go

import pandas as pd

# Load dataset
df = pd.read_csv(
    "https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv")
df.columns = [col.replace("AAPL.", "") for col in df.columns]

# Initialize figure
fig = go.Figure()

# Add Traces

fig.add_trace(
    go.Scatter(x=list(df.Date),
               y=list(df.High),
               name="High",
               line=dict(color="DarkBlue")))

fig.add_trace(
    go.Scatter(x=list(df.Date),
               y=[df.High.mean()] * len(df.index),
               name="High Average",
               visible=False,
               line=dict(color="DarkBlue", dash="dash"))))

fig.add_trace(
    go.Scatter(x=list(df.Date),
               y=list(df.Low),
               name="Low",
               line=dict(color="Crimson")))

fig.add_trace(
    go.Scatter(x=list(df.Date),
               y=[df.Low.mean()] * len(df.index),
               name="Low Average",
               visible=False,
               line=dict(color="Crimson", dash="dash"))))

# Add Annotations and Buttons
high_annotations = [dict(x="2016-03-01",
                          y=df.High.mean(),
                          xref="x", yref="y",
                          text="High Average:<br> %.3f" % df.High.mean(),
                          ax=0, ay=-40),
                     dict(x=df.Date[df.High.idxmax()],
                          y=df.High.max(),
                          xref="x", yref="y",
                          text="High Max:<br> %.3f" % df.High.max(),
                          ax=-40, ay=-40)]

low_annotations = [dict(x="2015-05-01",
                         y=df.Low.mean(),
                         xref="x", yref="y",
                         text="Low Average:<br> %.3f" % df.Low.mean(),
                         ax=0, ay=40),
                    dict(x=df.Date[df.Low.idxmin()],
                         y=df.Low.min(),
                         xref="x", yref="y",
                         text="Low Min:<br> %.3f" % df.Low.min(),
                         ax=0, ay=40)]

fig.update_layout(
    updatemenus=[
        dict(
            active=0,
            buttons=list([
                dict(label="None",
                     method="update",
                     args=[{"visible": [True, False, True, False]},
                           {"title": "Yahoo",
                            "annotations": []}]),
                dict(label="High",
                     method="update",
                     args=[{"visible": [True, True, False, False]},
                           {"title": "Yahoo High",
                            "annotations": high_annotations}]),
                dict(label="Low",

```

```

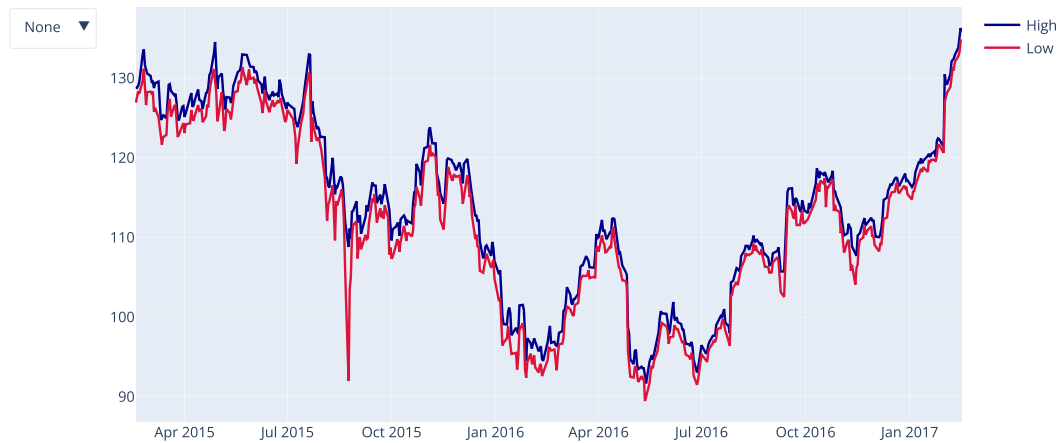
        method="update",
        args=[{"visible": [False, False, True, True]},
              {"title": "Yahoo Low",
               "annotations": low_annotations}],
        dict(label="Both",
             method="update",
             args=[{"visible": [True, True, True, True]},
                   {"title": "Yahoo",
                    "annotations": high_annotations + low_annotations}]),
    ),
)
])

# Set title
fig.update_layout(title_text="Yahoo")

fig.show()

```

Yahoo



## Graph Selection Dropdowns in Jinja

It is straight forward to create each potential view as a separate graph and then use Jinja to insert each potential view into a div on a JavaScript enabled webpage with a dropdown that chooses which div to display. This approach produces code that requires little customization or updating as you e.g. add, drop, or reorder views or traces, so it is particularly compelling for prototyping and rapid iteration. It produces web pages that are larger than the webpages produced through the built in method which is a consideration for very large figures with hundreds or thousands of data points in traces that appear in multiple selections. This approach requires both a Python program and a Jinja template file. The documentation on [using Jinja templates with Plotly](https://plotly.com/python/interactive-html-export/#inserting-plotly-output-into-html-using-a-jinja2-template) (<https://plotly.com/python/interactive-html-export/#inserting-plotly-output-into-html-using-a-jinja2-template>) is relevant background.

## Python Code File

```

import plotly.express as px
from jinja2 import Template
import collections
# Load the gapminder dataset
df = px.data.gapminder()

# Create a dictionary with Plotly figures as values
fig_dict = {}

# we need to fill that dictionary with figures. this example assumes that each figure has a title and that
# we want to use the titles as descriptions in the drop down
# This example happens to fill the dictionary by creating a scatter plot for each continent using the 2007 Gapminder data
for continent in df['continent'].unique():
    # Filter data for the current continent
    continent_data = df[(df['continent'] == continent) & (df['year'] == 2007)]

    fig_dict[continent] = px.scatter(continent_data, x='gdpPercap', y='lifeExp',
                                    title=f'GDP vs Life Expectancy for {continent}',
                                    labels={'gdpPercap': 'GDP per Capita (USD)', 'lifeExp': 'Life Expectancy (Years)'},
                                    hover_name='country', size="pop", size_max=55
                                    )

    #Standardizing the axes makes the graphs easier to compare
    fig_dict[continent].update_xaxes(range=[0,50000])
    fig_dict[continent].update_yaxes(range=[25,90])

# Create a dictionary, data_for_jinja with two entries:
# the value for the "dropdown_entries" key is a string containing a series of <option> tags, one tag for each item in the drop down
# the value for the "divs" key is a string with a series of <div> tags, each containing the content that appears only when the user selects the cor
responding item from the dropdown
# in this example, the content of each div is a figure and descriptive text.
data_for_jinja= collections.defaultdict(str)
text_dict = {}
for n, figname in enumerate(fig_dict.keys()):
    text_dict[figname]=f"Here is some custom text about the {figname} figure" #This is a succinct way to populate text_dict; in practice you'd prob
ably populate it manually elsewhere
    data_for_jinja["dropdown_entries"]+=f"<option value='{figname}'>{fig_dict[figname].layout.title.text}</option>"
    #YOU MAY NEED TO UPDATE THE LINK TO THE LATEST PLOTLY.JS
    fig_html = fig_dict[figname].to_html(full_html=False, config=dict(responsive=False, scrollZoom=False, doubleClick=False), include_plotlyjs = "c
dn")
    initially_hide_divs_other_than_the_first = "style=\"display:none;\"""*(n>0)
    data_for_jinja["divs"]+=f'<div id="{figname}" class="content-div" {initially_hide_divs_other_than_the_first}>{fig_html}{text_dict[figname]}</di
v>'

# Insert data into the template and write the file to disk
# You'll need to add the path to your template and to your preferred output location
input_template_path=r"<path-to-jinja-template.html>"
output_html_path=r"<path-to-output-file.html>"

with open(output_html_path, "w", encoding='utf-8') as output_file:
    with open(input_template_path) as template_file:
        j2_template = Template(template_file.read())
        output_file.write(j2_template.render(data_for_jinja))

```

## Jinja HTML Template

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5
6  </head>
7  <body>
8      <div class="container">
9          <h1>Select an analysis</h1>
10         <select id="dropdown" class="form-control">
11             {{ dropdown_entries }}
12         </select>
13
14
15         {{ divs }}
16
17     </div>
18
19     <script>
20         document.getElementById('dropdown').addEventListener('change', function() {
21             const divs = document.querySelectorAll('.content-div');
22             divs.forEach(div => div.style.display = 'none');
23
24             const selectedDiv = document.getElementById(this.value);
25             if (selectedDiv) {
26                 selectedDiv.style.display = 'block';
27             }
28         });
29     </script>
30 </body>
31 </html>

```

## Reference

See <https://plotly.com/python/reference/layout/updatemenus/> (<https://plotly.com/python/reference/layout/updatemenus/>) for more information about updatemenu dropdowns.

## What About Dash?

[Dash](https://dash.plot.ly/) (<https://dash.plot.ly/>) is an open-source framework for building analytical applications, with no Javascript required, and it is tightly integrated with the Plotly graphing library.

Learn about how to install Dash at <https://dash.plot.ly/installation> (<https://dash.plot.ly/installation>).

Everywhere in this page that you see `fig.show()`, you can display the same figure in a Dash application by passing it to the figure argument of the [Graph component](https://dash.plot.ly/dash-core-components/graph) (<https://dash.plot.ly/dash-core-components/graph>) from the built-in `dash_core_components` package like this:

```


import plotly.graph_objects as go # or plotly.express as px
fig = go.Figure() # or any Plotly Express function e.g. px.bar(...)
# fig.add_trace( ... )
# fig.update_layout( ... )

from dash import Dash, dcc, html

app = Dash()
app.layout = html.Div([
    dcc.Graph(figure=fig)
])

app.run(debug=True, use_reloader=False) # Turn off reloader if inside Jupyter

```



# Dash your way to interactive web apps.

No JavaScript required!

GET STARTED NOW

### My First App with Data, Graph, and Controls

pop

lifeExp

gdpPerCap

country	pop	continent	lifeExp	gdpPerCap
Afghanistan	31889923	Asia	43.828	974.5883384
Albania	2600522	Europe	76.422	5937.625525999999
Algeria	33333216	Africa	72.361	6223.367665
Angola	12428676	Africa	42.731	4707.231267
Argentina	40301927	Americas	75.32	12779.37964
Australia	20434176	Oceania	81.235	34435.367439999995
Austria	8199783	Europe	79.829	36126.4927
Bahrain	708573	Asia	75.635	29796.04834
Bangladesh	158448339	Asia	64.062	1501.253792
Belgium	10592226	Europe	79.441	33692.08508
Benin	8078314	Africa	56.728	1441.284873
Bolivia	9119152	Americas	65.554	3822.137884

continent


Asia

Europe

Africa

Americas

Oceania



([https://dash.plotly.com/tutorial?utm\\_medium=graphing\\_libraries&utm\\_content=python\\_footer](https://dash.plotly.com/tutorial?utm_medium=graphing_libraries&utm_content=python_footer))

JOIN OUR MAILING LIST

Sign up to stay in the loop with all things Plotly — from Dash Club to product updates, webinars, and more!

SUBSCRIBE  
([HTTPS://GO.PLOT.LY/SUBSCRIPTION](https://go.plot.ly/subscription))

About Us

Careers (<https://plotly.com/careers>)  
Resources (<https://plotly.com/resources/>)  
Blog (<https://medium.com/@plotlygraphs>)

Products

Dash (<https://plotly.com/dash/>)  
Consulting and Training  
(<https://plotly.com/consulting-and-oem/>)

Support

Community Support (<https://community.plot.ly/>)  
Documentation (<https://plotly.com/graphing-libraries>)

Pricing

Enterprise Pricing (<https://plotly.com/get-pricing/>)