a- plotly | Graphing Libraries (https://plotly.com/)(/graphing-libraries/)

utm_campaign=studio_cloud_launch&utm_content=sidebar)

Python (/python) > Animations (/python/#animations) > Intro to Animations

Suggest an edit to this page (https://github.com/plotly/plotly.py/edit/doc-prod/doc/python/animations.md)

Intro to Animations in Python

s

An introduction to creating animations with Plotly in Python.

ress

Caveats

ts

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)

ns

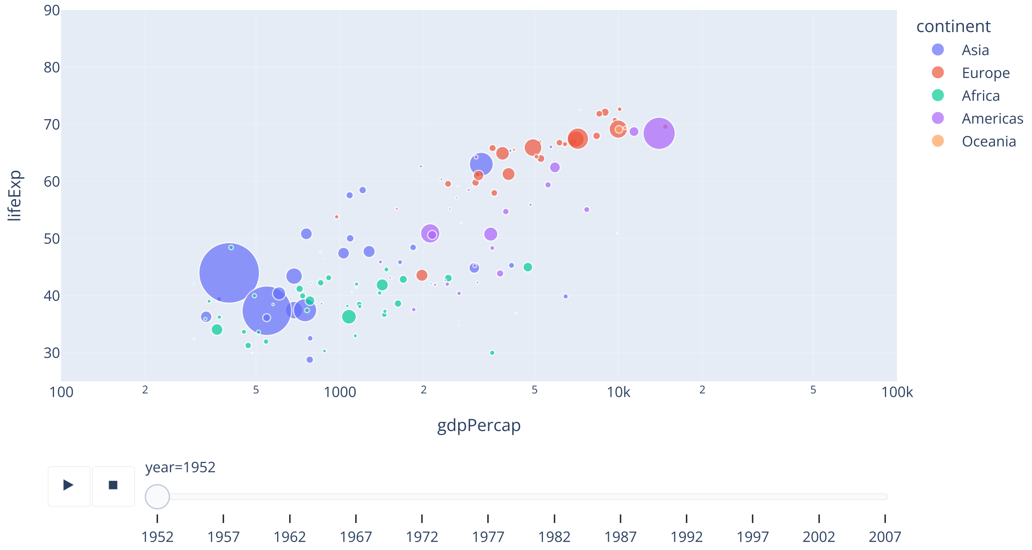
Animated figures with Plotly Express

r Curve

Several Plotly Express (/python/plotly-express/) functions support the creation of animated figures through the animation_frame and animation_group arguments.

Here is an example of an animated scatter plot created with Plotly Express. Note that you should always fix the x_range and y_range to ensure that your data remains visible throughout the animation.

```
import plotly.express as px
df = px.data.gapminder()
px.scatter(df, x="gdpPercap", y="lifeExp", animation_frame="year", animation_group="country",
           size="pop", color="continent", hover_name="country",
           log_x=True, size_max=55, range_x=[100,100000], range_y=[25,90])
```



Animated figures in Dash

Dash (https://plotly.com/dash/) is the best way to build analytical apps in Python using Plotly figures. To run the app below, run pip install dash, click "Download" to get the code and run python app.py.

Get started with the official Dash docs (https://dash.plotly.com/installation) and **learn how to effortlessly** style (https://plotly.com/dash/design-kit/) **&** deploy (https://plotly.com/dash/app-manager/) **apps like this with** Dash Enterprise (https://plotly.com/dash/)**.**

s

ɔress

Caveats

ts

ns

r Curve

DOWNLOAD

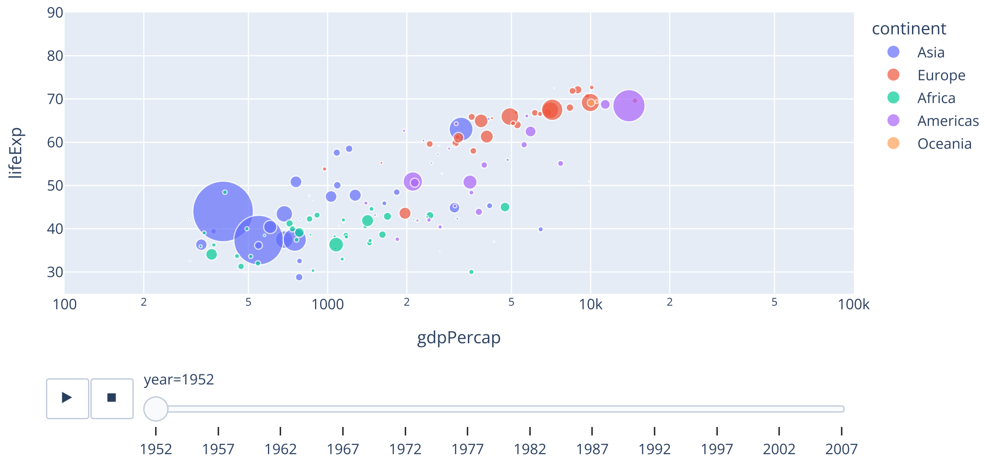
```
from dash import Dash, dcc, html, Input, Output
import plotly.express as px

app = Dash(__name__)


app.layout = html.Div([
    html.H4('Animated GDP and population over decades'),
    html.P("Select an animation:"),
    dcc.RadioItems(
        id='selection',
        options=["GDP - Scatter", "Population - Bar"],
        value='GDP - Scatter',
    ),
    dcc.Loading(dcc.Graph(id="graph"), type="cube")
])


@app.callback(
    Output("graph", "figure"),
    Input("selection", "value"))
def display_animated_graph(selection):
    df = px.data.gapminder() # replace with your own data source
    animations = {
        'GDP - Scatter': px.scatter(
```

Animated GDP and population over decades

Select an animation:

● GDP - Scatter

○ Population - Bar

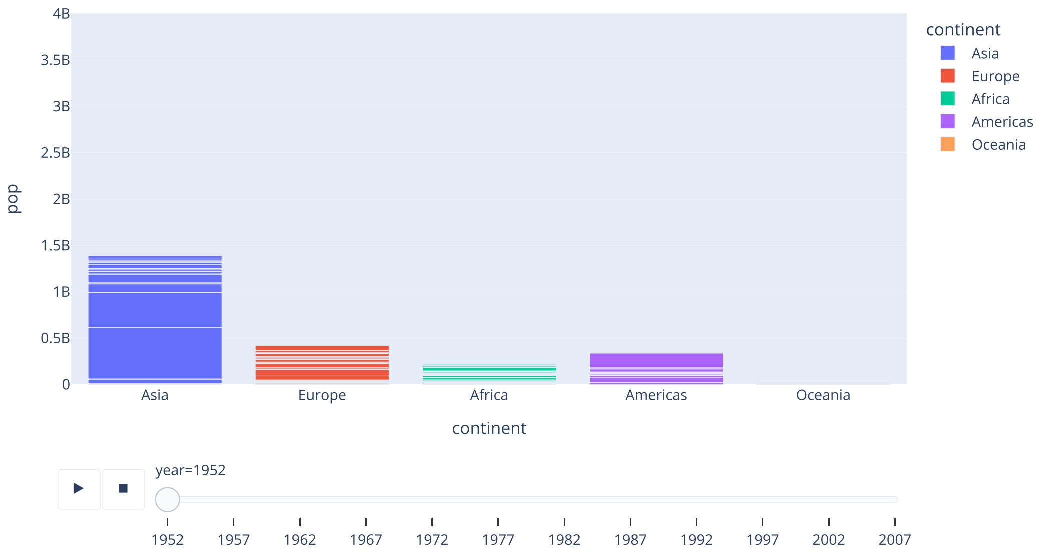


Animated Bar Charts with Plotly Express

Note that you should always fix the y_range to ensure that your data remains visible throughout the animation.

```
import plotly.express as px

df = px.data.gapminder()

fig = px.bar(df, x="continent", y="pop", color="continent",
  animation_frame="year", animation_group="country", range_y=[0,4000000000])
fig.show()
```

s
ress
Caveats
ts
ns
r Curve



Current Animation Limitations and Caveats

- Animations are designed to work well when each row of input is present across all animation frames, and when categorical values mapped to symbol, color and facet are constant across frames. Animations *may be misleading or inconsistent* if these constraints are not met.
- Although Plotly Express supports animation for many chart and map types, smooth inter-frame transitions are today *only* possible for scatter and bar
- Plotly Express will *not* automatically compute the union of all x/y/color ranges, so these must be specified manually to avoid scale jumps across frames

Animated figures with Graph Objects

The remainder of this section describes the low-level graph objects (/python/graph-objects/) API for constructing animated figures manually.

Frames

Along with data and layout, frames can be added as a key in a figure object. The frames key points to a list of figures, each of which will be cycled through when animation is triggered.

Adding Control Buttons to Animations

You can add play and pause buttons to control your animated charts by adding an updatemenus array to the layout of your figure. More information on style and placement of the buttons is available in Plotly's updatemenus reference (https://plotly.com/python/reference/layout/updatemenus/).
The buttons are defined as follows:

```
"updatemenus": [{"type": "buttons",
                 "buttons": [{"label": "Your Label",
                              "method": "animate",
                              "args": [See Below]}]}]
```

Defining Button Arguments

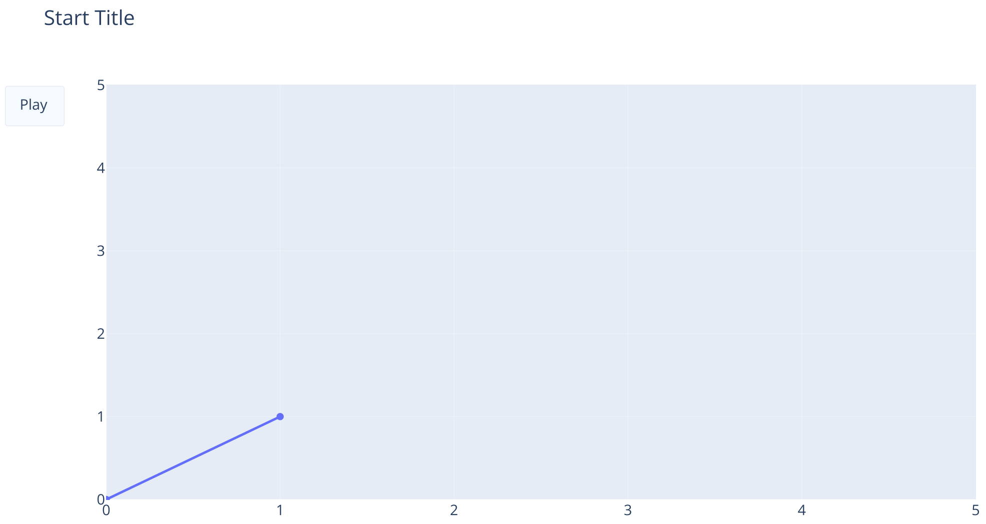
- None: Setting "args" to undefined (i.e. "args": [None]) will create a simple play button that will animate all frames.
- string: Animate all frames with group "<some string>". This is a way of scoping the animations in case you would prefer to animate without explicitly enumerating all frames.
- ["frame1", "frame2", ...]: Animate a sequence of named frames.
- [{data: [], layout: {}, traces: []}, {...}]: Nearly identical to animating named frames; though this variant lets you inline data instead of adding it as named frames. This can be useful for interaction where it's undesirable to add and manage named frames for ephemeral changes.
- [null]: A simple way to create a pause button (requires mode: "immediate"). This argument dumps the currently queued frames (mode: "immediate"), and then animates an empty sequence of frames ([null]).
- **Please Note:** We **do not** recommend using: [ ]. This syntax may cause confusion because it looks indistinguishable from a "pause button", but nested properties have logic that treats empty arrays as entirely removable, so it will function as a play button.

  Refer to the examples below to see the buttons in action!

s
ıress
Caveats
:s
ns
r Curve

Simple Play Button

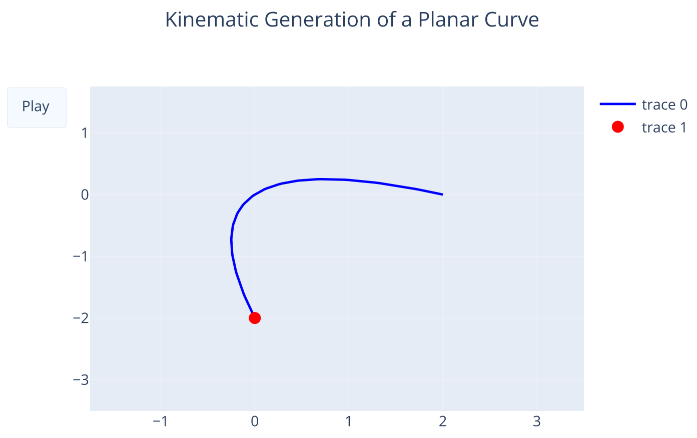
```
import plotly.graph_objects as go

fig = go.Figure(
    data=[go.Scatter(x=[0, 1], y=[0, 1])],
    layout=go.Layout(
        xaxis=dict(range=[0, 5], autorange=False),
        yaxis=dict(range=[0, 5], autorange=False),
        title=dict(text="Start Title"),
        updatemenus=[dict(
            type="buttons",
            buttons=[dict(label="Play",
                          method="animate",
                          args=[None])])]
    ),
    frames=[go.Frame(data=[go.Scatter(x=[1, 2], y=[1, 2])]),
            go.Frame(data=[go.Scatter(x=[1, 4], y=[1, 4])]),
            go.Frame(data=[go.Scatter(x=[3, 4], y=[3, 4])],
                     layout=go.Layout(title_text="End Title"))]
)

fig.show()
```



Moving Point on a Curve

s
ɔress
Caveats
ts
ns
r Curve

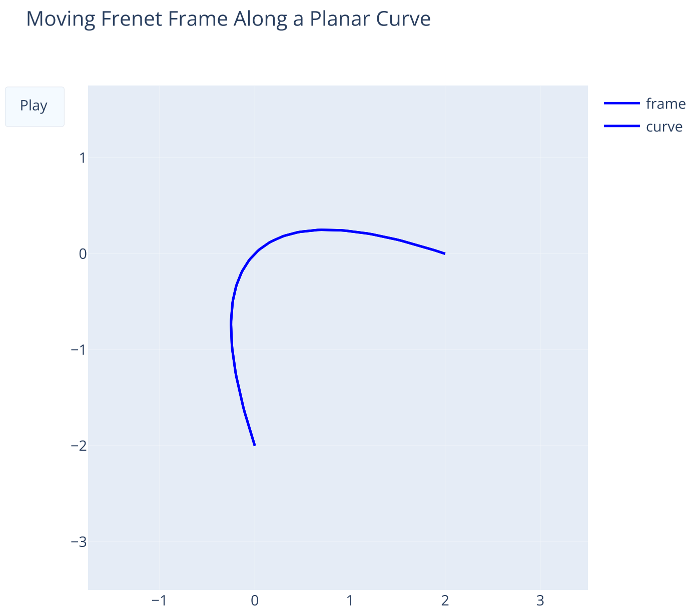
```
import plotly.graph_objects as go
import numpy as np
# Generate curve data
t = np.linspace(-1, 1, 100)
x = t + t ** 2
y = t - t ** 2
xm = np.min(x) - 1.5
xM = np.max(x) + 1.5
ym = np.min(y) - 1.5
yM = np.max(y) + 1.5
N = 25
s = np.linspace(-1, 1, N)
xx = s + s ** 2
yy = s - s ** 2


# Create figure
fig = go.Figure(
    data=[go.Scatter(x=x, y=y,
                     mode="lines",
                     line=dict(width=2, color="blue")),
          go.Scatter(x=[xx[0]], y=[yy[0]],
                     mode="markers",
                     marker=dict(color="red", size=10))])
fig.update_layout(width=600, height=450,
        xaxis=dict(range=[xm, xM], autorange=False, zeroline=False),
        yaxis=dict(range=[ym, yM], autorange=False, zeroline=False),
        title_text="Kinematic Generation of a Planar Curve", title_x=0.5,
        updatemenus = [dict(type = "buttons",
        buttons = [
            dict(
                args = [None, {"frame": {"duration": 10, "redraw": False},
                                "fromcurrent": True, "transition": {"duration": 10}}],
                label = "Play",
                method = "animate",

                )])])

fig.update(frames=[go.Frame(
                        data=[go.Scatter(
                                   x=[xx[k]],
                                   y=[yy[k]])],
                        traces=[1]) # fig.data[1] is updated by each frame
        for k in range(N)])

fig.show()
```



Moving Frenet Frame Along a Planar Curve

s
ɔress
Caveats
:s
ns
r Curve

```
import plotly.graph_objects as go

import numpy as np


# Generate curve data
t = np.linspace(-1, 1, 100)
x = t + t ** 2
y = t - t ** 2
xm = np.min(x) - 1.5
xM = np.max(x) + 1.5
ym = np.min(y) - 1.5
yM = np.max(y) + 1.5
N = 50
s = np.linspace(-1, 1, N)
xx = s + s ** 2
yy = s - s ** 2
vx = 1 + 2 * s
vy = 1 - 2 * s  # v=(vx, vy) is the velocity
speed = np.sqrt(vx ** 2 + vy ** 2)
ux = vx / speed  # (ux, uy) unit tangent vector, (-uy, ux) unit normal vector
uy = vy / speed

xend = xx + ux  # end coordinates for the unit tangent vector at (xx, yy)
yend = yy + uy

xnoe = xx - uy  # end coordinates for the unit normal vector at (xx,yy)
ynoe = yy + ux


# Create figure
fig = go.Figure(
    data=[go.Scatter(x=x, y=y,
                     name="frame",
                     mode="lines",
                     line=dict(width=2, color="blue")),
          go.Scatter(x=x, y=y,
                     name="curve",
                     mode="lines",
                     line=dict(width=2, color="blue"))
          ],
    layout=go.Layout(width=600, height=600,
                     xaxis=dict(range=[xm, xM], autorange=False, zeroline=False),
                     yaxis=dict(range=[ym, yM], autorange=False, zeroline=False),
                     title=dict(text="Moving Frenet Frame Along a Planar Curve"),
                     hovermode="closest",
                     updatemenus=[dict(type="buttons",
                                       buttons=[dict(label="Play",
                                                     method="animate",
                                                     args=[None])])]),

    frames=[go.Frame(
        data=[go.Scatter(
            x=[xx[k], xend[k], None, xx[k], xnoe[k]],
            y=[yy[k], yend[k], None, yy[k], ynoe[k]],
            mode="lines",
            line=dict(color="red", width=2))
        ]) for k in range(N)]
)

fig.show()
```



Using a Slider and Buttons

The following example uses the well known Gapminder dataset (https://www.gapminder.org/tag/gdp-per-capita/) to exemplify animation capabilities. This bubble chart animation shows the change in 'GDP per Capita' against the 'Life Expectancy' of several countries from the year 1952 to 2007, colored by their respective continent and sized by population.

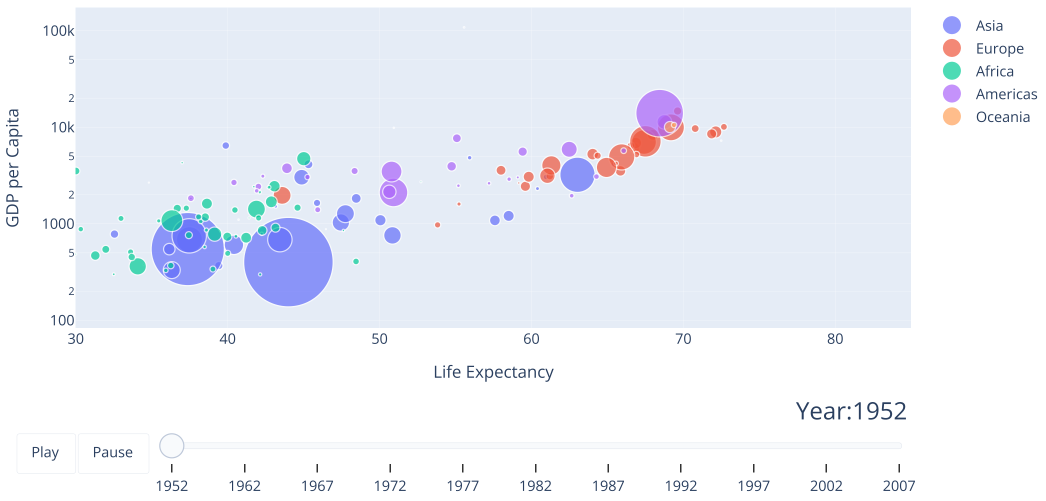
This is also an example of building up the structure of a figure as a Python dictionary, and then constructing a graph object figure from that dictionary.

s
ɔress
Caveats
:s
ns
r Curve

```python
import plotly.graph_objects as go

import pandas as pd

url = "https://raw.githubusercontent.com/plotly/datasets/master/gapminderDataFiveYear.csv"
dataset = pd.read_csv(url)

years = ["1952", "1962", "1967", "1972", "1977", "1982", "1987", "1992", "1997", "2002",
         "2007"]

# make list of continents
continents = []
for continent in dataset["continent"]:
    if continent not in continents:
        continents.append(continent)
# make figure
fig_dict = {
    "data": [],
    "layout": {},
    "frames": []
}

# fill in most of layout
fig_dict["layout"]["xaxis"] = {"range": [30, 85], "title": "Life Expectancy"}
fig_dict["layout"]["yaxis"] = {"title": "GDP per Capita", "type": "log"}
fig_dict["layout"]["hovermode"] = "closest"
fig_dict["layout"]["updatemenus"] = [
    {
        "buttons": [
            {
                "args": [None, {"frame": {"duration": 500, "redraw": False},
                                "fromcurrent": True, "transition": {"duration": 300,
                                                                    "easing": "quadratic-in-out"}}],
                "label": "Play",
                "method": "animate"
            },
            {
                "args": [[None], {"frame": {"duration": 0, "redraw": False},
                                  "mode": "immediate",
                                  "transition": {"duration": 0}}],
                "label": "Pause",
                "method": "animate"
            }
        ],
        "direction": "left",
        "pad": {"r": 10, "t": 87},
        "showactive": False,
        "type": "buttons",
        "x": 0.1,
        "xanchor": "right",
        "y": 0,
        "yanchor": "top"
    }
]

sliders_dict = {
    "active": 0,
    "yanchor": "top",
    "xanchor": "left",
    "currentvalue": {
        "font": {"size": 20},
        "prefix": "Year:",
        "visible": True,
        "xanchor": "right"
    },
    "transition": {"duration": 300, "easing": "cubic-in-out"},
    "pad": {"b": 10, "t": 50},
    "len": 0.9,
    "x": 0.1,
    "y": 0,
    "steps": []
}

# make data
year = 1952
for continent in continents:
    dataset_by_year = dataset[dataset["year"] == year]
    dataset_by_year_and_cont = dataset_by_year[
```

s
ɔress
Caveats
ts
ns
r Curve

```
        dataset_by_year["continent"] == continent]

    data_dict = {
        "x": list(dataset_by_year_and_cont["lifeExp"]),
        "y": list(dataset_by_year_and_cont["gdpPercap"]),
        "mode": "markers",
        "text": list(dataset_by_year_and_cont["country"]),
        "marker": {
            "sizemode": "area",
            "sizeref": 200000,
            "size": list(dataset_by_year_and_cont["pop"])
        },
        "name": continent
    }
    fig_dict["data"].append(data_dict)

# make frames
for year in years:
    frame = {"data": [], "name": str(year)}
    for continent in continents:
        dataset_by_year = dataset[dataset["year"] == int(year)]
        dataset_by_year_and_cont = dataset_by_year[
            dataset_by_year["continent"] == continent]

        data_dict = {
            "x": list(dataset_by_year_and_cont["lifeExp"]),
            "y": list(dataset_by_year_and_cont["gdpPercap"]),
            "mode": "markers",
            "text": list(dataset_by_year_and_cont["country"]),
            "marker": {
                "sizemode": "area",
                "sizeref": 200000,
                "size": list(dataset_by_year_and_cont["pop"])
            },
            "name": continent
        }
        frame["data"].append(data_dict)

    fig_dict["frames"].append(frame)
    slider_step = {"args": [
        [year],
        {"frame": {"duration": 300, "redraw": False},
         "mode": "immediate",
         "transition": {"duration": 300}}
    ],
        "label": year,
        "method": "animate"}
    sliders_dict["steps"].append(slider_step)


fig_dict["layout"]["sliders"] = [sliders_dict]

fig = go.Figure(fig_dict)

fig.show()
```



Important Notes

- Defining redraw: Setting redraw: false is an optimization for scatter plots so that animate just makes changes without redrawing the whole plot. For other plot types, such as contour plots, every frame **must** be a total plot redraw, i.e. redraw: true.

Reference

For additional information and attributes for creating bubble charts in Plotly see: https://plotly.com/python/bubble-charts/ (https://plotly.com/python/bubble-charts/). For more documentation on creating animations with Plotly, see https://plotly.com/python/#animations (https://plotly.com/python/#animations).

What About Dash?

Dash (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) 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
```

s

ɔress

Caveats

:s

ns

r Curve

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)

Products

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

Pricing

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

About Us

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

Support

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

Terms of Service (https://community.plotly.com/tos) Privacy Policy (https://plotly.com/privacy/)