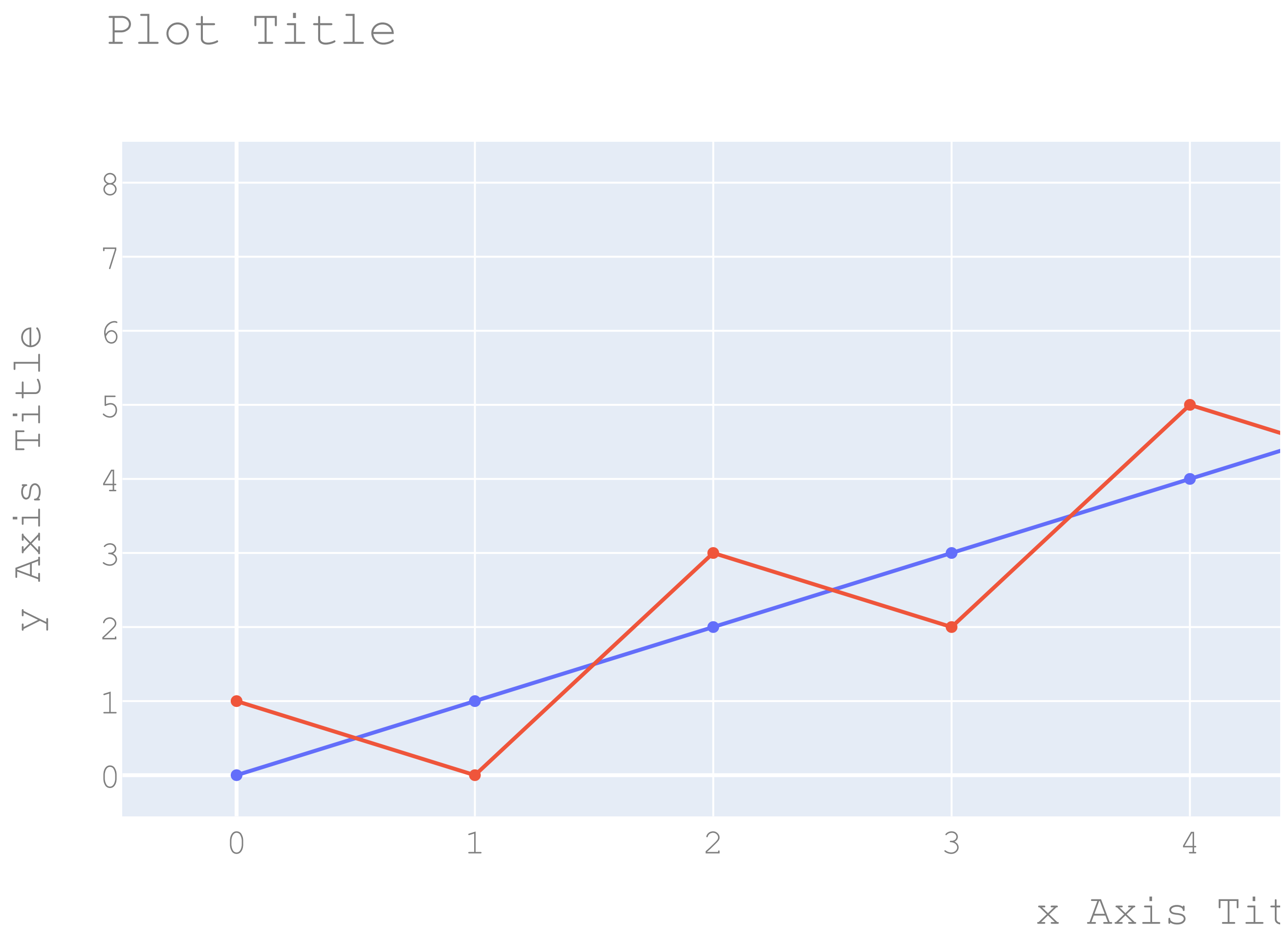
Plotly Graphs with Khom

```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=[0, 1, 2, 3, 4, 5, 6, 7, 8],
    y=[0, 1, 2, 3, 4, 5, 6, 7, 8],
    name='Name of Trace 1' # this sets its legend entry
    ))

fig.add_trace(go.Scatter(
    x=[0, 1, 2, 3, 4, 5, 6, 7, 8],
    y=[1, 0, 3, 2, 5, 4, 7, 6, 8],
    name='Name of Trace 2'
    ))

fig.update_layout(
    title='Plot Title',
    xaxis_title='x Axis Title',
    yaxis_title='y Axis Title',
    font=dict(
    family='Courier New, monospace',
    size=18,
    color="#7f7f7f"
)
)

fig.show()

# import plotly.graph_objects as go

# fig = go.Figure()

# fig.add_trace(go.Scatter(
# x=[0, 1, 2, 3, 4, 5, 6, 7, 8],
# y=[0, 1, 2, 3, 4, 5, 6, 7, 8],
# name="Name of Trace 1" # this sets its legend entry
# ))

# fig.add_trace(go.Scatter(
# x=[0, 1, 2, 3, 4, 5, 6, 7, 8],
# y=[1, 0, 3, 2, 5, 4, 7, 6, 8],
# name="Name of Trace 2"
# ))
# fig.update_layout(
# title="Plot Title",
# xaxis_title="x Axis Title",
# yaxis_title="y Axis Title",
# font=dict(
# family="Courier New, monospace",
# size=18,
# color="#7f7f7f"
# )
# )

# fig.show()
```



python: plotly attribute error

```
!pip install plotly
```

```
    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
    Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (5.5.0)
    Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from plot
    Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-package
```

```
pip install chart-studio
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting chart-studio
  Downloading chart_studio-1.1.0-py3-none-any.whl (64 kB)
     |████████████████████████████████| 64 kB 2.5 MB/s
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from chart
Collecting retrying>=1.3.3
  Downloading retrying-1.3.3.tar.gz (10 kB)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist-packages (from ch
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py) ... done
  Created wheel for retrying: filename=retrying-1.3.3-py3-none-any.whl size=11447 sha25
  Stored in directory: /root/.cache/pip/wheels/f9/8d/8d/f6af3f7f9eea3553bc2fe6d53e4b287
Successfully built retrying
Installing collected packages: retrying, chart-studio
Successfully installed chart-studio-1.1.0 retrying-1.3.3
```

```
from _plotly_future_ import v4_subplots
from plotly.subplots import make_subplots
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
# import plotly.plotly as py

from chart_studio import plotly

fig1 = make_subplots(
    rows=2, cols=2,
    specs=[[{"type": "pie"}, {"type": "pie"}],
           [{"type": "table"}, {"type": "table"}]],
)

fig1.add_trace(
    go.Pie(
        labels=pie_alarms_total['alarm_type'],
        values=pie_alarms_total['alarm_timestamp'],
        name="Total Alarms",
        title="test"
    ),
    row=1, col=1

)

fig1.add_trace(
    go.Pie(
        labels=pie_alarms_notbd['alarm_type'],
        values=pie_alarms_notbd['alarm_timestamp'],
        name="No TBDs"
    ),
    row=1, col=2
)

fig1.add_trace(
    go.Table(
        header=dict(
            values=pie_alarms_total['alarm_type'],
            line_color='darkslategray',
            fill_color='lightskyblue'
        ),
        cells=dict(
            values=pie_alarms_total['alarm_timestamp'],
            line_color='darkslategray',
            fill_color='lightcyan'
        )
    ),
    row=2, col=1
)

fig1.add_trace(
    go.Table(
        header=dict(
            values=pie_alarms_notbd['alarm_type'],
            line_color='darkslategray',
            fill_color='lightskyblue'
        ),
        cells=dict(
            values=pie_alarms_notbd['alarm_timestamp'],
            line_color='darkslategray',
            fill_color='lightcyan'
        )
    ),
    row=2, col=2
)

# fig1.update_layout(title_text="Test")
fig1.layout.update(title_text="Test")

plot(fig1, filename="test.html")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-e6b99b7e4e6f> in <module>()
     15 fig1.add_trace(
     16     go.Pie(
---> 17         labels=pie_alarms_total['alarm_type'],
     18         values=pie_alarms_total['alarm_timestamp'],
```

Imshow in Python

How to display image data in Python with Plotly.

Displaying RBG image data with px.imshow px.imshow displays multichannel (RGB) or single-channel ("grayscale") image data.

```
import plotly.express as px
import numpy as np
img_rgb = np.array([[[255, 0, 0], [0, 255, 0], [0, 0, 255]],
                    [[0, 255, 0], [0, 0, 255], [255, 0, 0]]
                   ], dtype=np.uint8)
fig = px.imshow(img_rgb)
fig.show()
```

Next

```
import matplotlib.pyplot as plt
# Your plotting code here
plt.tight_layout()
```

```
<Figure size 432x288 with 0 Axes>
```

```
plt.tight_layout(img_rgb)
```

```
<Figure size 432x288 with 0 Axes>
```

Alternatively,

```
pip install fuzzywuzzy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting fuzzywuzzy
  Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)
Installing collected packages: fuzzywuzzy
Successfully installed fuzzywuzzy-0.18.0
```

```
import matplotlib.pyplot as plt
from fuzzywuzzy import fuzz

fig, ax = plt.subplots()

My_means = ratios.pivot_table('Data', 'Sample ID', 'User ID', aggfunc= np.mean)
My_stds = ratios.pivot_table('Data', 'Sample ID', 'User ID', aggfunc= np.std)

# My_means = ratios.pivot_table('Data', 'Sample ID', 'User ID', aggfunc= np.mean)
# My_stds = ratios.pivot_table('Data', 'Sample ID', 'User ID', aggfunc= np.std)

# plot_it = My_means.plot(kind='bar', ax=ax, …)
plot_it = My_means.plot(kind='bar', color=stack_cols, stacked=True, yerr=My_stds, capsize=3,


fig.tight_layout()
```

```
      File "<ipython-input-24-0d31d2bb17be>", line 6
        My means = fuzz.ratio.('Data', 'Sample ID', 'User ID', aggfunc= np.mean)
```

Learn More

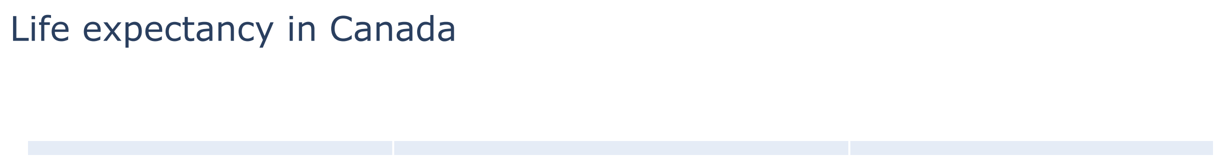
Line Charts in Python

How to make line charts in Python with Plotly. Examples on creating and styling line charts in Python with Plotly.

Line Plots with plotly.express

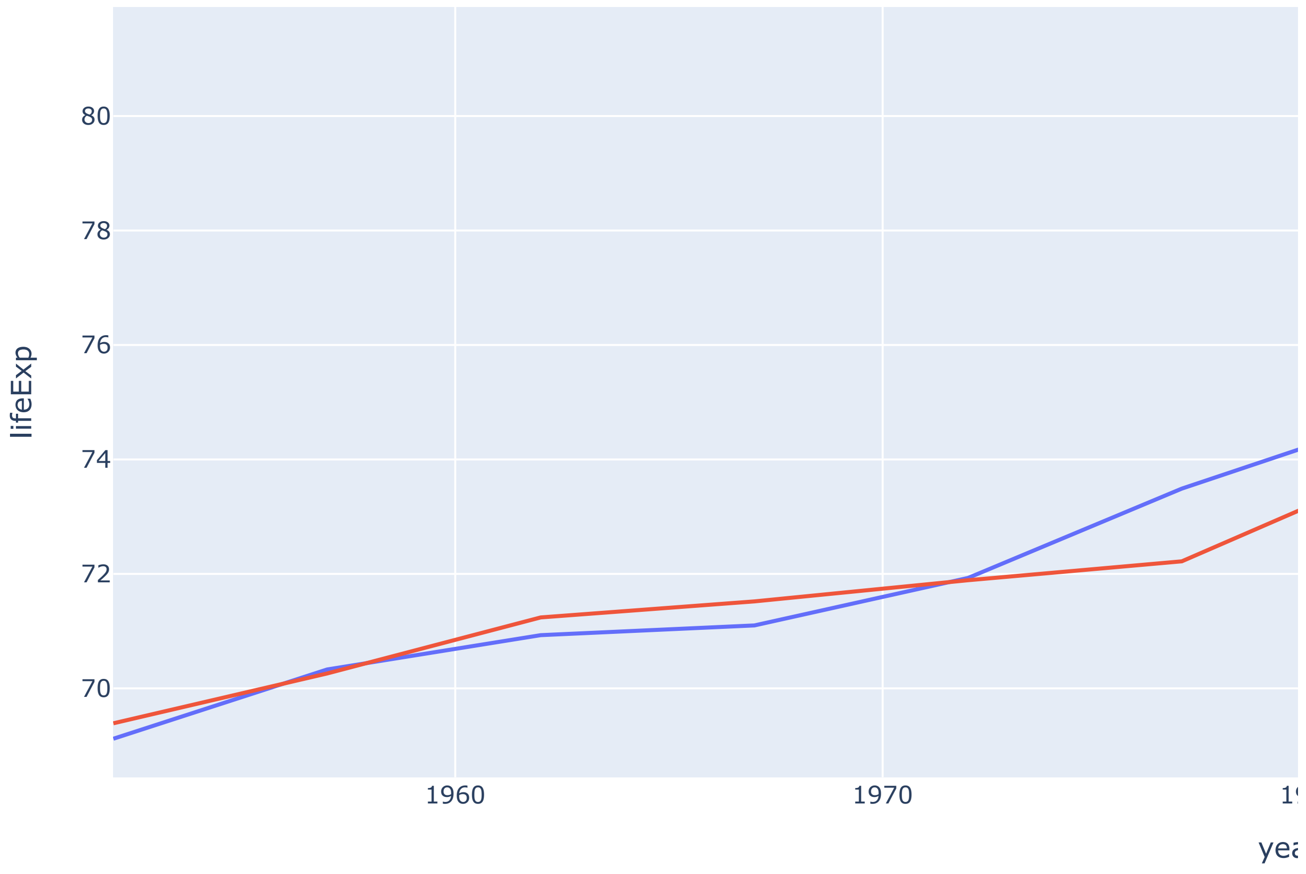
Plotly Express is the easy-to-use, high-level interface to Plotly, which operates on a variety of types of data and produces easy-to-style figures. With px.line, each data point is represented as a vertex (which location is given by the x and y columns) of a polyline mark in 2D space.

```
import plotly.express as px

df = px.data.gapminder().query("country=='Canada'")
fig = px.line(df, x="year", y="lifeExp", title='Life expectancy in Canada')
fig.show()
```



Line Plots with column encoding color

```
import plotly.express as px

df = px.data.gapminder().query("continent=='Oceania'")
fig = px.line(df, x="year", y="lifeExp", color='country')
fig.show()
```



Line charts in Dash

Dash is the best way to build analytical apps in Python using Plotly figures. To run the app below,

```
pip install dash
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Collecting dash
  Downloading dash-2.5.1-py3-none-any.whl (9.8 MB)
     |████████████████████████████████| 9.8 MB 5.6 MB/s
Collecting dash-html-components==2.0.0
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Requirement already satisfied: Flask>=1.0.4 in /usr/local/lib/python3.7/dist-packages (
Collecting dash-table==5.0.0
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.7/dist-packages 
Collecting flask-compress
  Downloading Flask_Compress-1.12-py3-none-any.whl (7.9 kB)
Requirement already satisfied: Jinja2<3.0,>=2.10.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: Werkzeug<2.0,>=0.15 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: itsdangerous<2.0,>=0.24 in /usr/local/lib/python3.7/dist
Requirement already satisfied: click<8.0,>=5.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from plot
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.7/dist-package
Collecting brotli
  Downloading Brotli-1.0.9-cp37-cp37m-manylinux1_x86_64.whl (357 kB)
     |████████████████████████████████| 357 kB 45.9 MB/s
Installing collected packages: brotli, flask-compress, dash-table, dash-html-components
Successfully installed brotli-1.0.9 dash-2.5.1 dash-core-components-2.0.0 dash-html-com
```

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

app = Dash(__name__)


app.layout = html.Div([
    html.H4('Life expentancy progression of countries per continents'),
    dcc.Graph(id="graph"),
    dcc.Checklist(
        id="checklist",
        options=["Asia", "Europe", "Africa","Americas","Oceania"],
        value=["Americas", "Oceania"],
        inline=True
    ),
])


@app.callback(
    Output("graph", "figure"),
    Input("checklist", "value"))
```

```
def update_line_chart(continents):
    df = px.data.gapminder() # replace with your own data source
    mask = df.continent.isin(continents)
    fig = px.line(df[mask],
        x="year", y="lifeExp", color='country')
    return fig


app.run_server(debug=True)
```

```
Dash is running on http://127.0.0.1:8050/

Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-6-b37935700b91> in <module>()
     28 
     29 
---> 30 app.run_server(debug=True)

3 frames
/usr/local/lib/python3.7/dist-packages/werkzeug/serving.py in run_simple(hostname, port, application, use_reloader, use_debugger, use_evalex, extra_files, reloader_interval, reloader_type, threaded, processes, request_handler, static_files, passthrough_errors, ssl_context)
   1028             s = socket.socket(address_family, socket.SOCK_STREAM)
   1029             s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
-> 1030             s.bind(server_address)
   1031             if hasattr(s, "set_inheritable"):
   1032                 s.set_inheritable(True)

OSError: [Errno 98] Address already in use
```

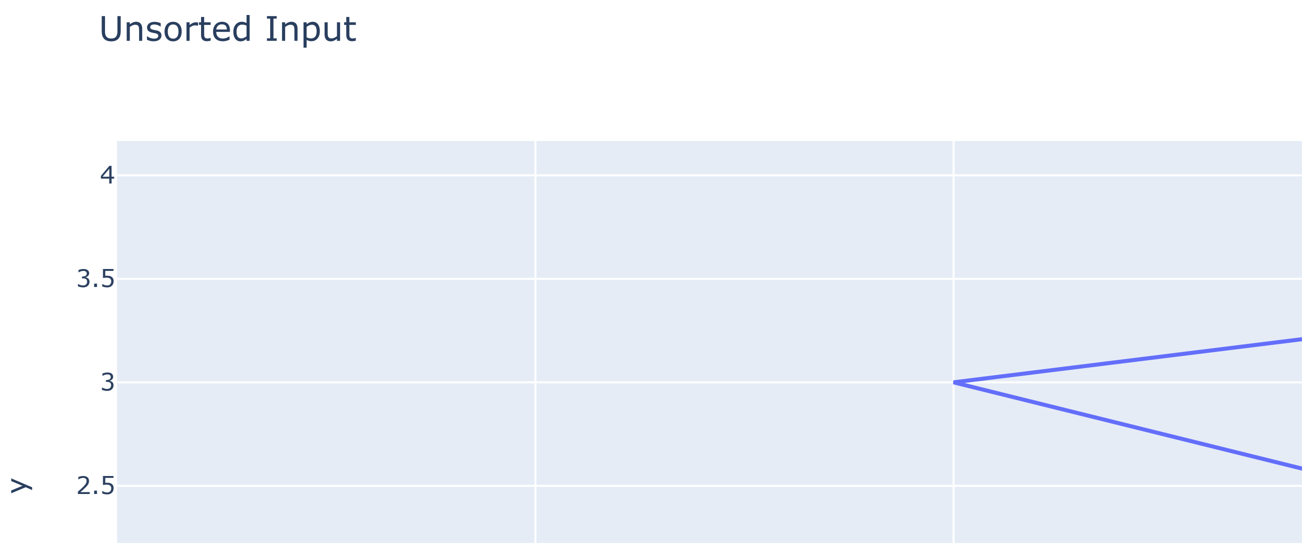
Data Order in Line Charts

Plotly line charts are implemented as connected scatterplots (see below), meaning that the points are plotted and connected with lines in the order they are provided, with no automatic reordering.

This makes it possible to make charts like the one below, but also means that it may be required to explicitly sort data before passing it to Plotly to avoid lines moving "backwards" across the chart.

```
import plotly.express as px
import pandas as pd
```

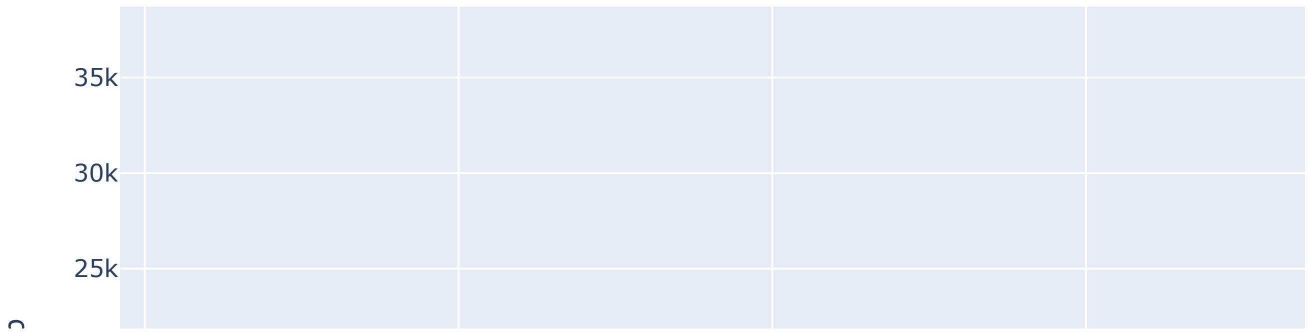
```
df = pd.DataFrame(dict(
    x = [1, 3, 2, 4],
    y = [1, 2, 3, 4]
))
fig = px.line(df, x="x", y="y", title="Unsorted Input")
fig.show()

df = df.sort_values(by="x")
fig = px.line(df, x="x", y="y", title="Sorted Input")
fig.show()
```



Connected Scatterplots

In a connected scatterplot, two continuous variables are plotted against each other, with a line connecting them in some meaningful order, usually a time variable. In the plot below, we show the "trajectory" of a pair of countries through a space defined by GDP per Capita and Life Expectancy. Botswana's life expectancy

```
import plotly.express as px

df = px.data.gapminder().query("country in ['Canada', 'Botswana']")

fig = px.line(df, x="lifeExp", y="gdpPercap", color="country", text="year")
fig.update_traces(textposition="bottom right")
fig.show()
```



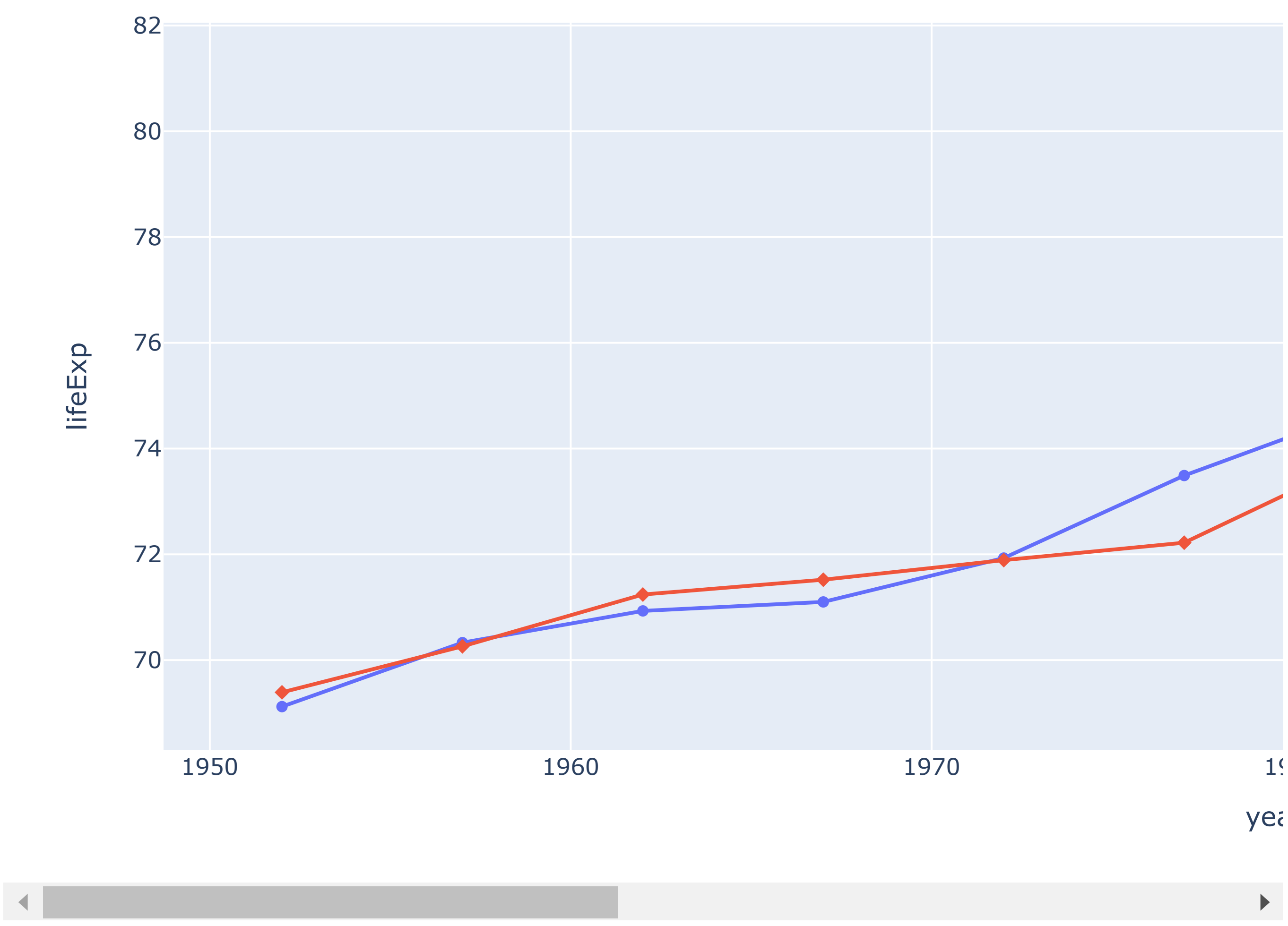
Line charts with markers

The markers argument can be set to True to show markers on lines.

```
import plotly.express as px
df = px.data.gapminder().query("continent == 'Oceania'")
fig = px.line(df, x='year', y='lifeExp', color='country', markers=True)
fig.show()
```

The symbol argument can be used to map a data field to the marker symbol. A wide variety of symbols are available.

```
import plotly.express as px
df = px.data.gapminder().query("continent == 'Oceania'")
fig = px.line(df, x='year', y='lifeExp', color='country', symbol="country")
fig.show()
```



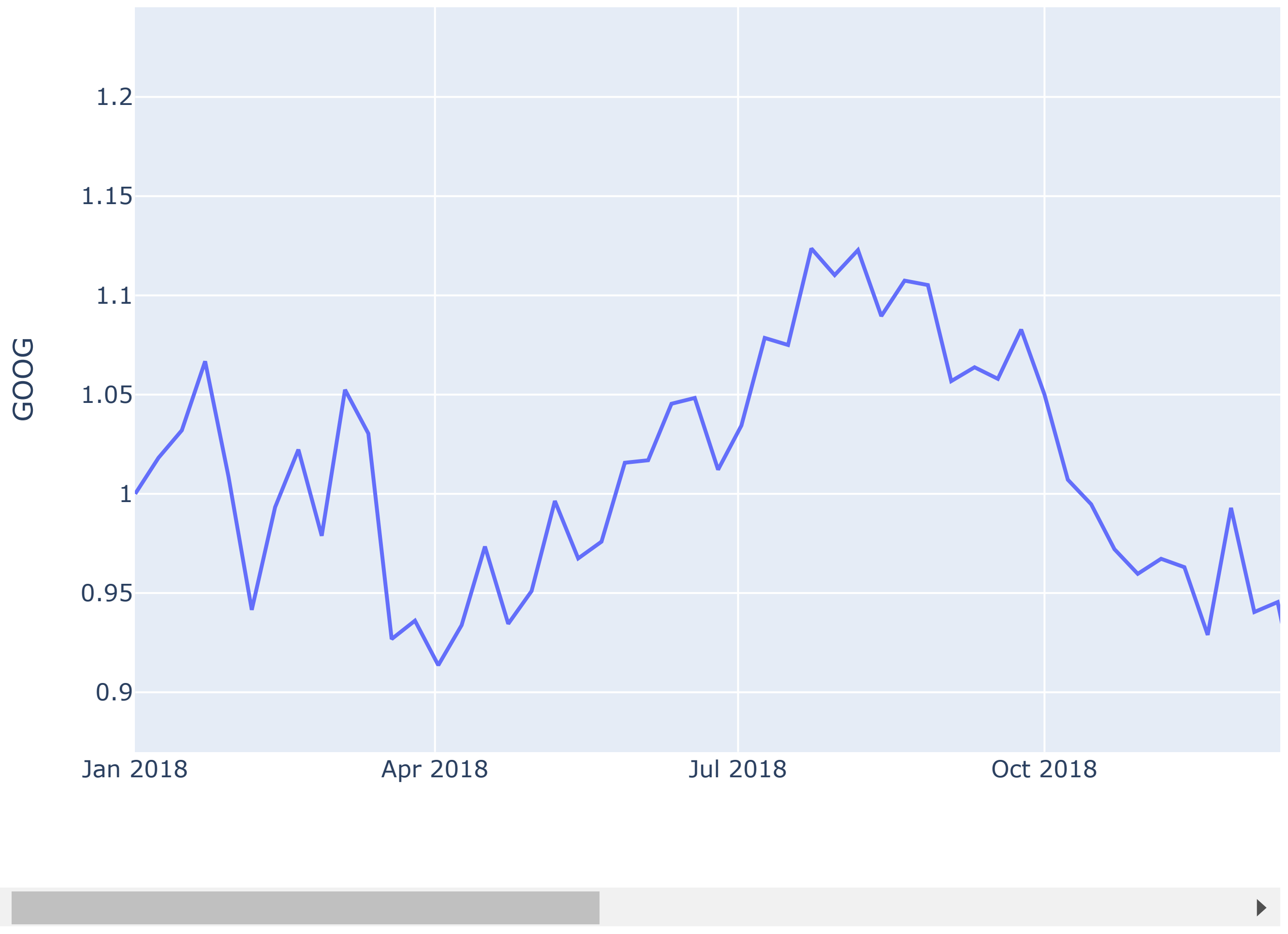
Line plots on Date axes

Line plots can be made on using any type of cartesian axis, including linear, logarithmic, categorical or date axes. Line plots on date axes are often called time-series charts.

Plotly auto-sets the axis type to a date format when the corresponding data are either ISO-formatted date strings or if they're a date pandas column or datetime NumPy array.

```
import plotly.express as px
```

```
df = px.data.stocks()
fig = px.line(df, x='date', y="GOOG")
fig.show()
```

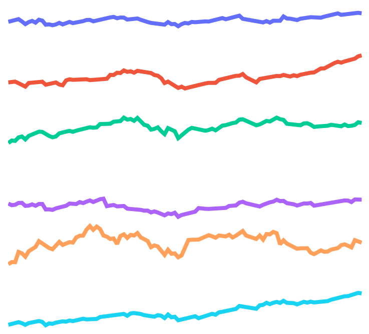


Sparklines with Plotly Express

Sparklines are scatter plots inside subplots, with gridlines, axis lines, and ticks removed.

```
import plotly.express as px
df = px.data.stocks(indexed=True)
fig = px.line(df, facet_row="company", facet_row_spacing=0.01, height=200, width=200)

# hide and lock down axes
fig.update_xaxes(visible=False, fixedrange=True)
fig.update_yaxes(visible=False, fixedrange=True)

# remove facet/subplot labels
fig.update_layout(annotations=[], overwrite=True)
```

```
# strip down the rest of the plot
fig.update_layout(
    showlegend=False,
    plot_bgcolor="white",
    margin=dict(t=10,l=10,b=10,r=10)
)

# disable the modebar for such a small plot
fig.show(config=dict(displayModeBar=False))
```

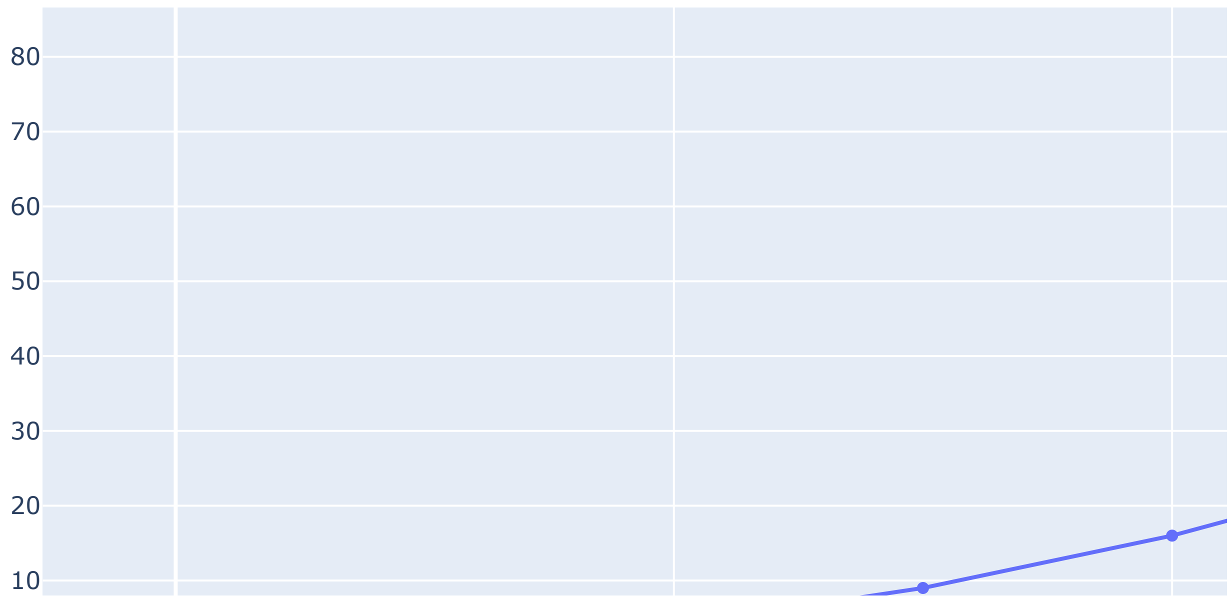


Line Plot with go.Scatter

If Plotly Express does not provide a good starting point, it is possible to use the more generic go.Scatter class from plotly.graph_objects. Whereas plotly.express has two functions scatter and line, go.Scatter can be used both for plotting points (makers) or lines, depending on the value of mode. The different options of go.Scatter are documented in its reference page.

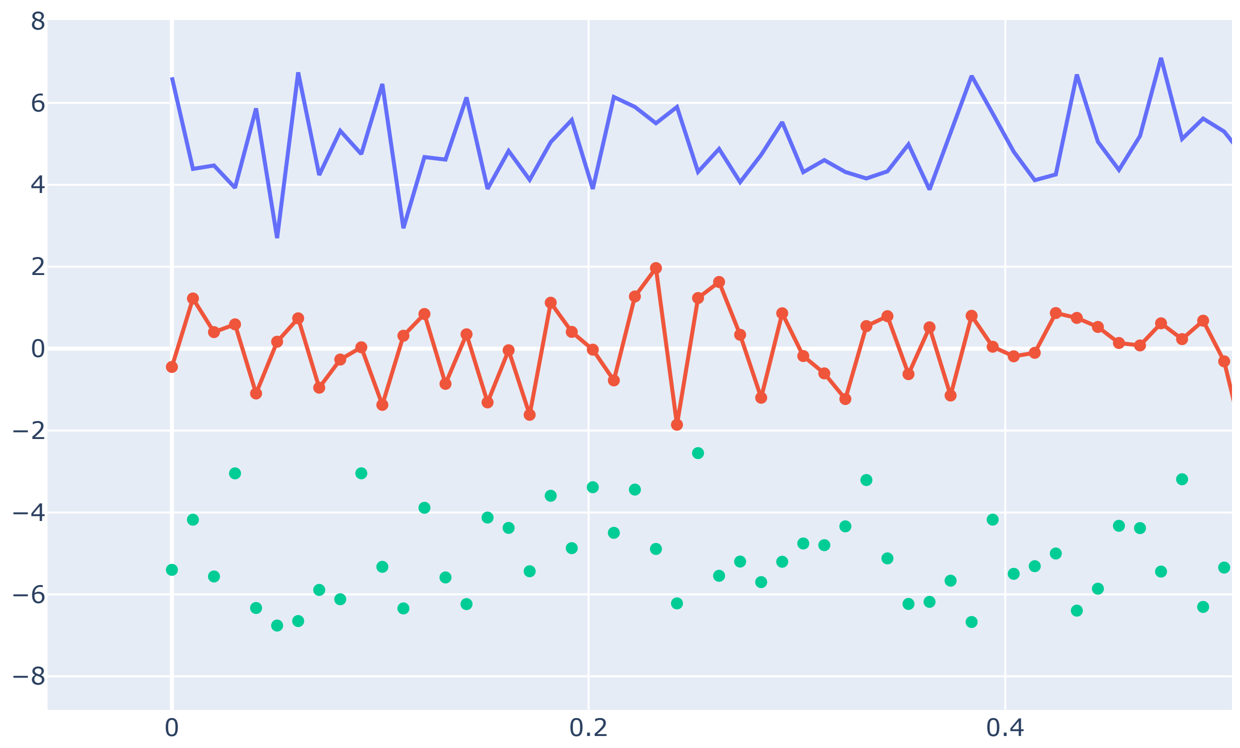
Simple Line Plot

```
import plotly.graph_objects as go
import numpy as np

x = np.arange(10)

fig = go.Figure(data=go.Scatter(x=x, y=x**2))
fig.show()
```



Line Plot Modes

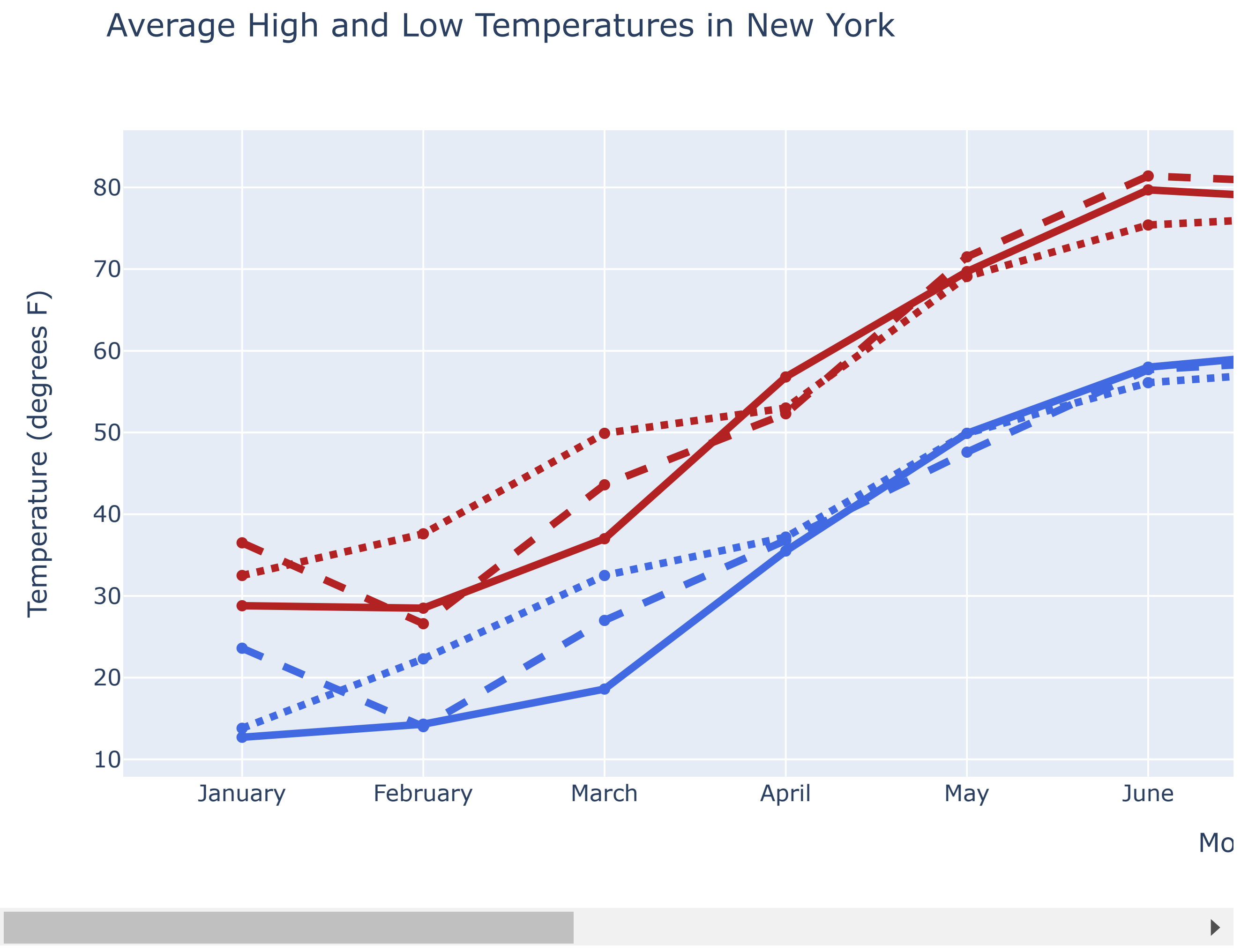
```
import plotly.graph_objects as go

# Create random data with numpy
import numpy as np
np.random.seed(1)

N = 100
random_x = np.linspace(0, 1, N)
random_y0 = np.random.randn(N) + 5
random_y1 = np.random.randn(N)
random_y2 = np.random.randn(N) - 5

# Create traces
fig = go.Figure()
fig.add_trace(go.Scatter(x=random_x, y=random_y0,
                    mode='lines',
                    name='lines'))
fig.add_trace(go.Scatter(x=random_x, y=random_y1,
                    mode='lines+markers',
                    name='lines+markers'))
fig.add_trace(go.Scatter(x=random_x, y=random_y2,
                    mode='markers', name='markers'))

fig.show()
```



Style Line Plots

This example styles the color and dash of the traces, adds trace names, modifies line width, and adds plot and axes titles.

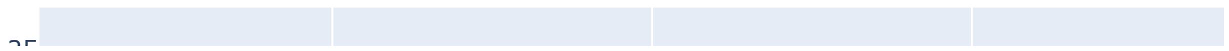
```
import plotly.graph_objects as go

# Add data
month = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
         'August', 'September', 'October', 'November', 'December']
high_2000 = [32.5, 37.6, 49.9, 53.0, 69.1, 75.4, 76.5, 76.6, 70.7, 60.6, 45.1, 29.3]
low_2000 = [13.8, 22.3, 32.5, 37.2, 49.9, 56.1, 57.7, 58.3, 51.2, 42.8, 31.6, 15.9]
high_2007 = [36.5, 26.6, 43.6, 52.3, 71.5, 81.4, 80.5, 82.2, 76.0, 67.3, 46.1, 35.0]
low_2007 = [23.6, 14.0, 27.0, 36.8, 47.6, 57.7, 58.9, 61.2, 53.3, 48.5, 31.0, 23.6]
high_2014 = [28.8, 28.5, 37.0, 56.8, 69.7, 79.7, 78.5, 77.8, 74.1, 62.6, 45.3, 39.9]
low_2014 = [12.7, 14.3, 18.6, 35.5, 49.9, 58.0, 60.0, 58.6, 51.7, 45.2, 32.2, 29.1]

fig = go.Figure()
# Create and style traces
fig.add_trace(go.Scatter(x=month, y=high_2014, name='High 2014',
                         line=dict(color='firebrick', width=4)))
fig.add_trace(go.Scatter(x=month, y=low_2014, name = 'Low 2014',
                         line=dict(color='royalblue', width=4)))
fig.add_trace(go.Scatter(x=month, y=high_2007, name='High 2007',
                         line=dict(color='firebrick', width=4,
                              dash='dash') # dash options include 'dash', 'dot', and 'dashdot
))
fig.add_trace(go.Scatter(x=month, y=low_2007, name='Low 2007',
                         line = dict(color='royalblue', width=4, dash='dash')))
fig.add_trace(go.Scatter(x=month, y=high_2000, name='High 2000',
                         line = dict(color='firebrick', width=4, dash='dot')))
fig.add_trace(go.Scatter(x=month, y=low_2000, name='Low 2000',
                         line=dict(color='royalblue', width=4, dash='dot')))

# Edit the layout
fig.update_layout(title='Average High and Low Temperatures in New York',
                  xaxis_title='Month',
                  yaxis_title='Temperature (degrees F)')


fig.show()
```



Connect Data Gaps

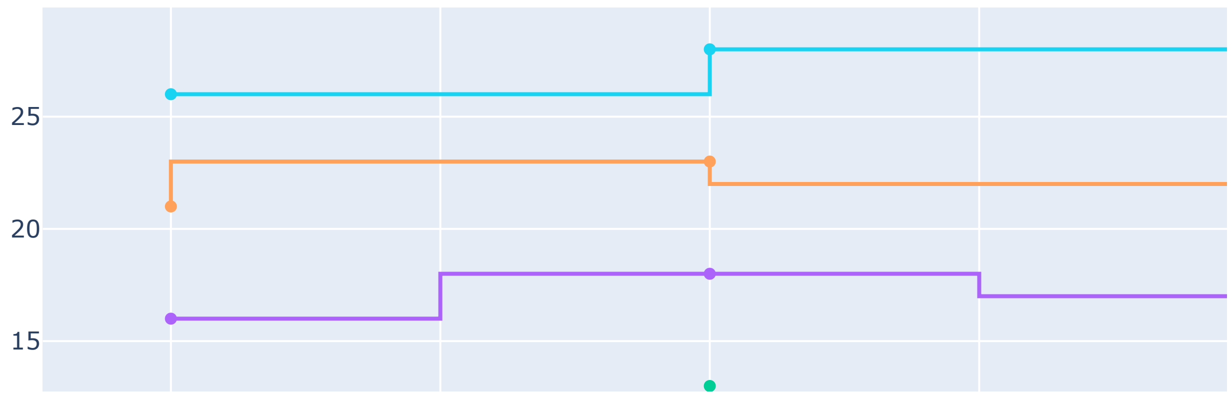
connectgaps determines if missing values in the provided data are shown as a gap in the graph or not. In this tutorial, we showed how to take benefit of this feature and illustrate multiple areas in mapbox.

```
import plotly.graph_objects as go

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

fig = go.Figure()

fig.add_trace(go.Scatter(
    x=x,
    y=[10, 20, None, 15, 10, 5, 15, None, 20, 10, 10, 15, 25, 20, 10],
    name = '<b>No</b> Gaps', # Style name/legend entry with html tags
    connectgaps=True # override default to connect the gaps
))
fig.add_trace(go.Scatter(
    x=x,
    y=[5, 15, None, 10, 5, 0, 10, None, 15, 5, 5, 10, 20, 15, 5],
    name='Gaps',
))

fig.show()
```

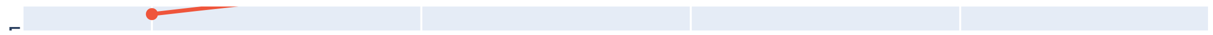


Interpolation with Line Plots

```
import plotly.graph_objects as go
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 3, 2, 3, 1])

fig = go.Figure()
fig.add_trace(go.Scatter(x=x, y=y, name="linear",
                    line_shape='linear'))
fig.add_trace(go.Scatter(x=x, y=y + 5, name="spline",
                    text=["tweak line smoothness<br>with 'smoothing' in line object"],
                    hoverinfo='text+name',
                    line_shape='spline'))
fig.add_trace(go.Scatter(x=x, y=y + 10, name="vhv",
                    line_shape='vhv'))
fig.add_trace(go.Scatter(x=x, y=y + 15, name="hvh",
                    line_shape='hvh'))
fig.add_trace(go.Scatter(x=x, y=y + 20, name="vh",
                    line_shape='vh'))
fig.add_trace(go.Scatter(x=x, y=y + 25, name="hv",
                    line_shape='hv'))

fig.update_traces(hoverinfo='text+name', mode='lines+markers')
fig.update_layout(legend=dict(y=0.5, traceorder='reversed', font_size=16))

fig.show()
```



Label Lines with Annotations

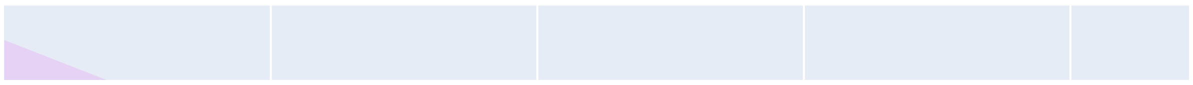


```
import plotly.graph_objects as go
import numpy as np

title = 'Main Source for News'
labels = ['Television', 'Newspaper', 'Internet', 'Radio']
colors = ['rgb(67,67,67)', 'rgb(115,115,115)', 'rgb(49,130,189)', 'rgb(189,189,189)']

mode_size = [8, 8, 12, 8]
line_size = [2, 2, 4, 2]

x_data = np.vstack((np.arange(2001, 2014),)*4)

y_data = np.array([
    [74, 82, 80, 74, 73, 72, 74, 70, 70, 66, 66, 69],
    [45, 42, 50, 46, 36, 36, 34, 35, 32, 31, 31, 28],
    [13, 14, 20, 24, 20, 24, 24, 40, 35, 41, 43, 50],
    [18, 21, 18, 21, 16, 14, 13, 18, 17, 16, 19, 23],
])

fig = go.Figure()

for i in range(0, 4):
    fig.add_trace(go.Scatter(x=x_data[i], y=y_data[i], mode='lines',
        name=labels[i],
        line=dict(color=colors[i], width=line_size[i]),
        connectgaps=True,
    ))

    # endpoints
    fig.add_trace(go.Scatter(
        x=[x_data[i][0], x_data[i][-1]],
        y=[y_data[i][0], y_data[i][-1]],
        mode='markers',
        marker=dict(color=colors[i], size=mode_size[i])
    ))

fig.update_layout(
    xaxis=dict(
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=False,
    ),
    autosize=False,
    margin=dict(
        autoexpand=False,
        l=100,
        r=20,
        t=110,
    ),
    showlegend=False,
    plot_bgcolor='white'
)

annotations = []

# Adding labels
for y_trace, label, color in zip(y_data, labels, colors):
    # labeling the left_side of the plot
    annotations.append(dict(xref='paper', x=0.05, y=y_trace[0],
                              xanchor='right', yanchor='middle',
                              text=label + ' {}%'.format(y_trace[0]),
                              font=dict(family='Arial',
                                        size=16),
                              showarrow=False))
    # labeling the right_side of the plot
    annotations.append(dict(xref='paper', x=0.95, y=y_trace[11],
                              xanchor='left', yanchor='middle',
                              text='{}%'.format(y_trace[11]),
                                  font=dict(family='Arial',
                                            size=16),
                                  showarrow=False))
# Title
annotations.append(dict(xref='paper', yref='paper', x=0.0, y=1.05,
                              xanchor='left', yanchor='bottom',
                              text='Main Source for News',
                              font=dict(family='Arial',
                                        size=30,
                                        color='rgb(37,37,37)'),
                              showarrow=False))
# Source
annotations.append(dict(xref='paper', yref='paper', x=0.5, y=-0.1,
                              xanchor='center', yanchor='top',
                              text='Source: PewResearch Center & ' +
                                   'Storytelling with data',
                              font=dict(family='Arial',
                                        size=12,
                                        color='rgb(150,150,150)'),
                              showarrow=False))

fig.update_layout(annotations=annotations)

fig.show()
```

Filled Lines

```
import plotly.graph_objects as go
import numpy as np
```

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x_rev = x[::-1]

# Line 1
y1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y1_upper = [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
y1_lower = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
y1_lower = y1_lower[::-1]

# Line 2
y2 = [5, 2.5, 5, 7.5, 5, 2.5, 7.5, 4.5, 5.5, 5]
y2_upper = [5.5, 3, 5.5, 8, 6, 3, 8, 5, 6, 5.5]
y2_lower = [4.5, 2, 4.4, 7, 4, 2, 7, 4, 5, 4.75]
y2_lower = y2_lower[::-1]

# Line 3
y3 = [10, 8, 6, 4, 2, 0, 2, 4, 2, 0]
y3_upper = [11, 9, 7, 5, 3, 1, 3, 5, 3, 1]
y3_lower = [9, 7, 5, 3, 1, -.5, 1, 3, 1, -1]
y3_lower = y3_lower[::-1]


fig = go.Figure()

fig.add_trace(go.Scatter(
    x=x+x_rev,
    y=y1_upper+y1_lower,
    fill='toself',
    fillcolor='rgba(0,100,80,0.2)',
    line_color='rgba(255,255,255,0)',
    showlegend=False,
    name='Fair',
))
fig.add_trace(go.Scatter(
    x=x+x_rev,
    y=y2_upper+y2_lower,
    fill='toself',
    fillcolor='rgba(0,176,246,0.2)',
    line_color='rgba(255,255,255,0)',
    name='Premium',
    showlegend=False,
))
fig.add_trace(go.Scatter(
    x=x+x_rev,
    y=y3_upper+y3_lower,
    fill='toself',
    fillcolor='rgba(231,107,243,0.2)',
    line_color='rgba(255,255,255,0)',
    showlegend=False,
    name='Ideal',
))
fig.add_trace(go.Scatter(
    x=x, y=y1,
    line_color='rgb(0,100,80)',
    name='Fair',
))
fig.add_trace(go.Scatter(
    x=x, y=y2,
    line_color='rgb(0,176,246)',
    name='Premium',
))
fig.add_trace(go.Scatter(
    x=x, y=y3,
    line_color='rgb(231,107,243)',
    name='Ideal',
))

fig.update_traces(mode='lines')
fig.show()
```



Reference

See function reference for px.line() or https://plotly.com/python/reference/scatter/ for more information and chart attribute options!

What About Dash?

Dash 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.

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 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( ... )

import dash
import dash_core_components as dcc
# import dash_html_components as html
from dash import html

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

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

```
     Dash is running on http://127.0.0.1:8050/

     Dash is running on http://127.0.0.1:8050/

     Dash is running on http://127.0.0.1:8050/
```

```
Dash is running on http://127.0.0.1:8050/

 * Serving Flask app "__main__" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
```

✓ 16s completed at 3:45 PM