Line Plots

A line chart or line plot or line graph or curve chart is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments. Line Chart on Wikipedia (https://en.wikipedia.org/wiki/Line_chart)

- Line graphs are used to track changes over short and long periods of time. When smaller changes exist, line graphs are better to use than bar graphs. Line graphs can also be used to compare changes over the same period of time for more than one group. Source (https://nces.ed.gov/nceskids/help/user_guide/graph/whentouse.asp)
- Line graphs are usually used to show time series data - that is how one or more variables vary over a continuous period of time. Typical examples of the types of data that can be presented using line graphs are monthly rainfall and annual unemployment rates.
- Line graphs are particularly useful for identifying patterns and trends in the data such as seasonal effects, large changes and turning points.
- As well as time series data, line graphs can also be appropriate for displaying data that are measured over other continuous variables such as distance.
- For example, a line graph could be used to show how pollution levels vary with increasing distance from a source, or how the level of a chemical varies with depth of soil.
- However, it is important to consider whether the data have been collected at sufficiently regular intervals so that estimates made for a point lying half-way along the line between two successive measurements would be reasonable.
- In a line graph the x-axis represents the continuous variable (for example year or distance from the initial measurement) whilst the y-axis has a scale and indicates the measurement.
- Several data series can be plotted on the same line chart and this is particularly useful for analysing and comparing the trends in different datasets. Source (https://www.le.ac.uk/oerresources/ssds/numeracyskills/page_34.htm)

In [1]:
```
!pip install --upgrade plotly
import plotly
plotly.__version__
```

```
Collecting plotly
  Downloading https://files.pythonhosted.org/packages/bf/5f/47ab0d9d843
c5be0f5c5bd891736a4c84fa45c3b0a0ddb6b6df7c098c66f/plotly-4.9.0-py2.py3-
none-any.whl (12.9MB)
    100% |████████████████████████████████| 12.9MB 879kB/s eta 0:00:01
Requirement already satisfied, skipping upgrade: six in /Users/samah/an
aconda3/lib/python3.7/site-packages (from plotly) (1.12.0)
Collecting retrying>=1.3.3 (from plotly)
  Downloading https://files.pythonhosted.org/packages/44/ef/beae4b4ef80
902f22e3af073397f079c96969c69b2c7d52a57ea9ae61c9d/retrying-1.3.3.tar.gz
Building wheels for collected packages: retrying
  Running setup.py bdist_wheel for retrying ... done
  Stored in directory: /Users/samah/Library/Caches/pip/wheels/d7/a9/33/
acc7b709e2a35caa7d4cae442f6fe6fbf2c43f80823d46460c
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.9.0 retrying-1.3.3
```

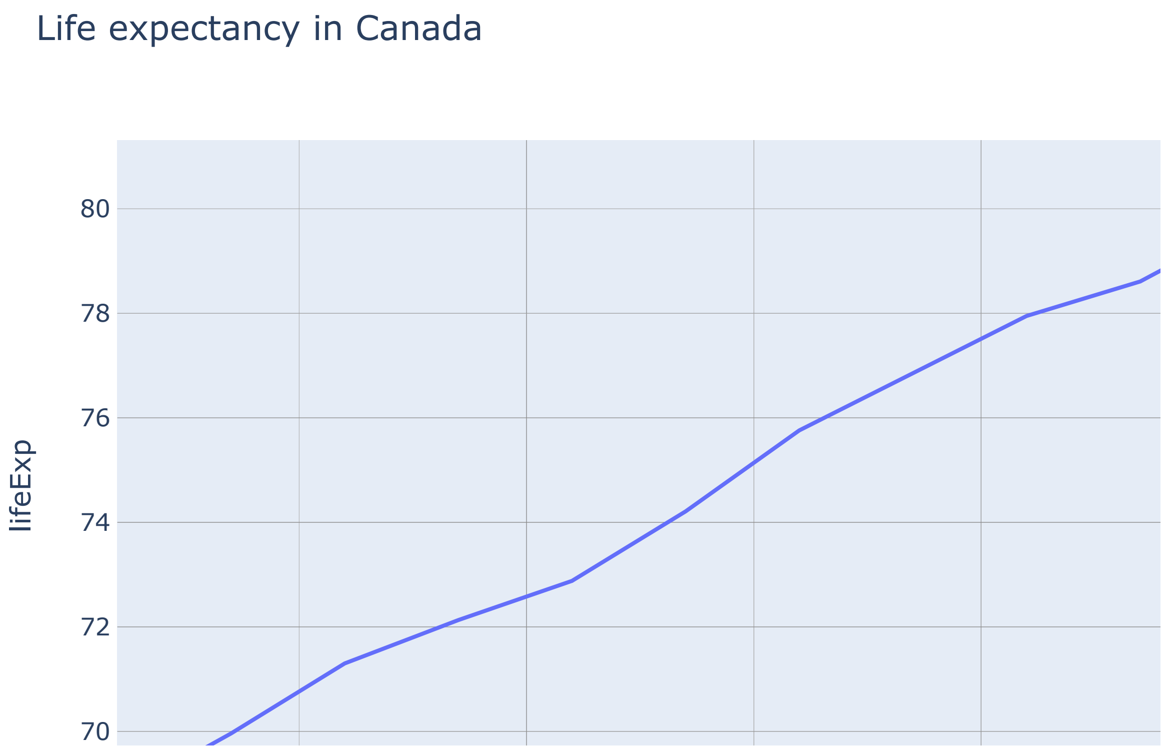
Out[1]:
```
'4.9.0'
```

Simple Line Plot with plotly.express

plotly.express is a high level interface to Plotly .. 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.

In [2]:

```
import plotly.express as px

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



In [3]:

```
df.head()
```

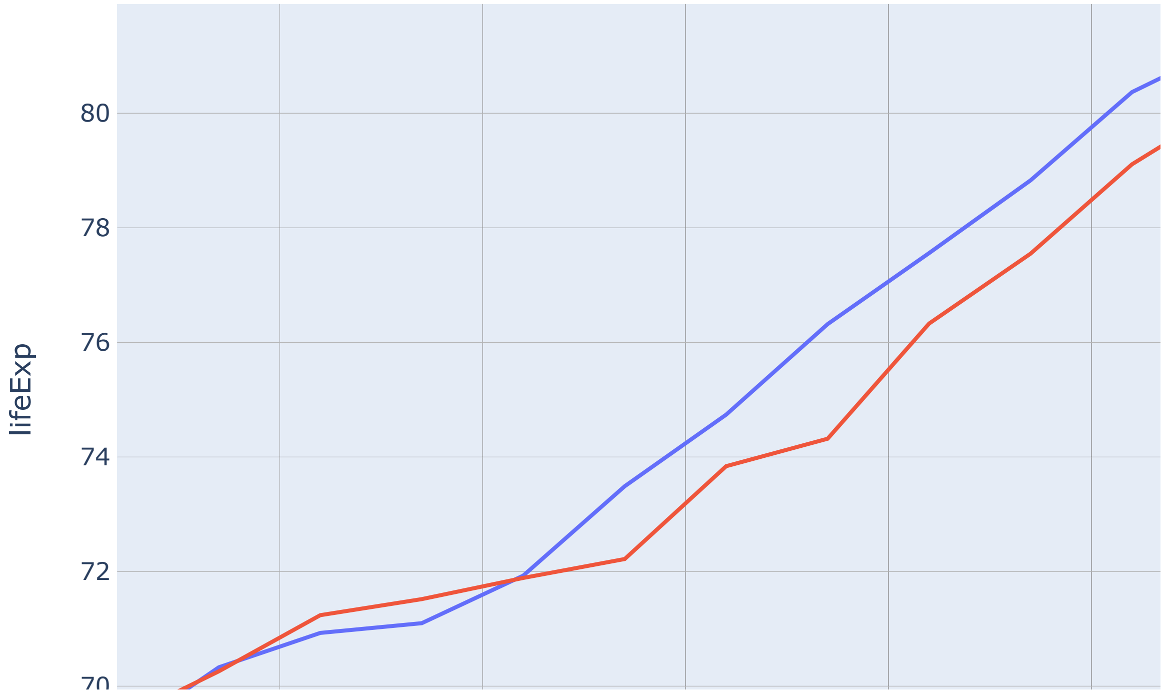
Out[3]:

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
240	Canada	Americas	1952	68.75	14785584	11367.16112	CAN	124
241	Canada	Americas	1957	69.96	17010154	12489.95006	CAN	124
242	Canada	Americas	1962	71.30	18985849	13462.48555	CAN	124
243	Canada	Americas	1967	72.13	20819767	16076.58803	CAN	124
244	Canada	Americas	1972	72.88	22284500	18970.57086	CAN	124

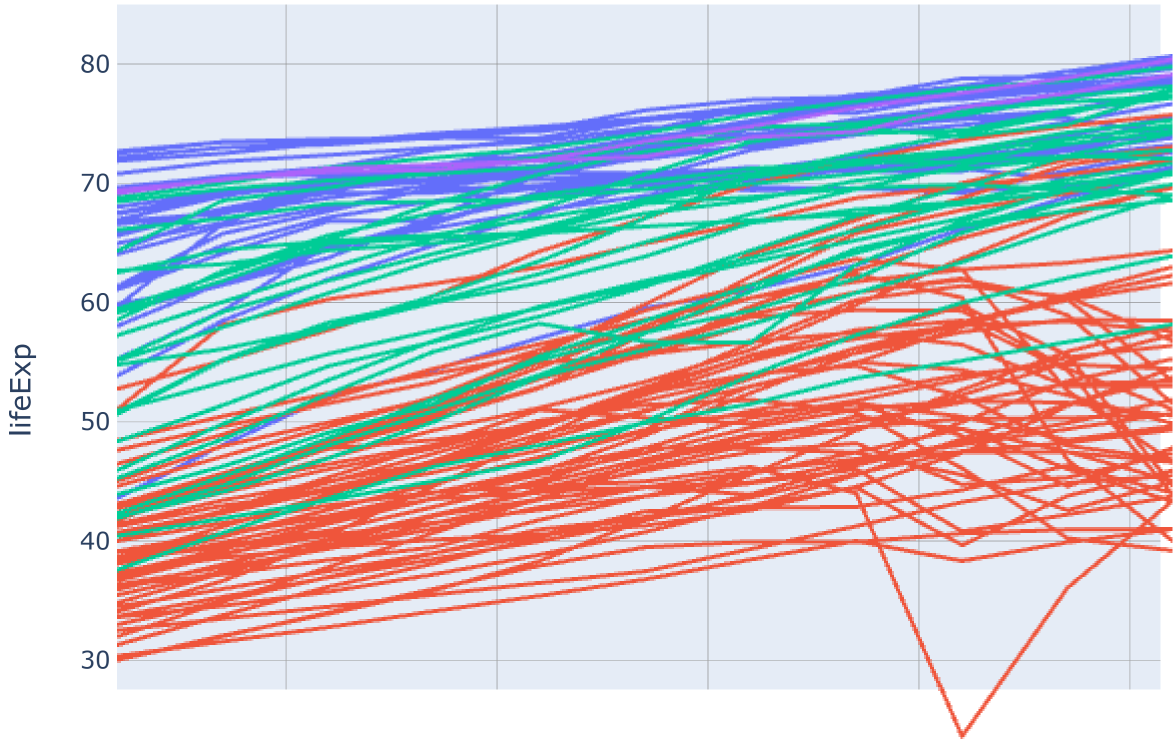
Line Plot with column encoding color

In [4]:

```
import plotly.express as px

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



In [5]:
```
import plotly.express as px

df = px.data.gapminder().query("continent != 'Asia'") # remove Asia for
 visibility
fig = px.line(df, x="year", y="lifeExp", color="continent",
              line_group="country", hover_name="country")
fig.show()
```



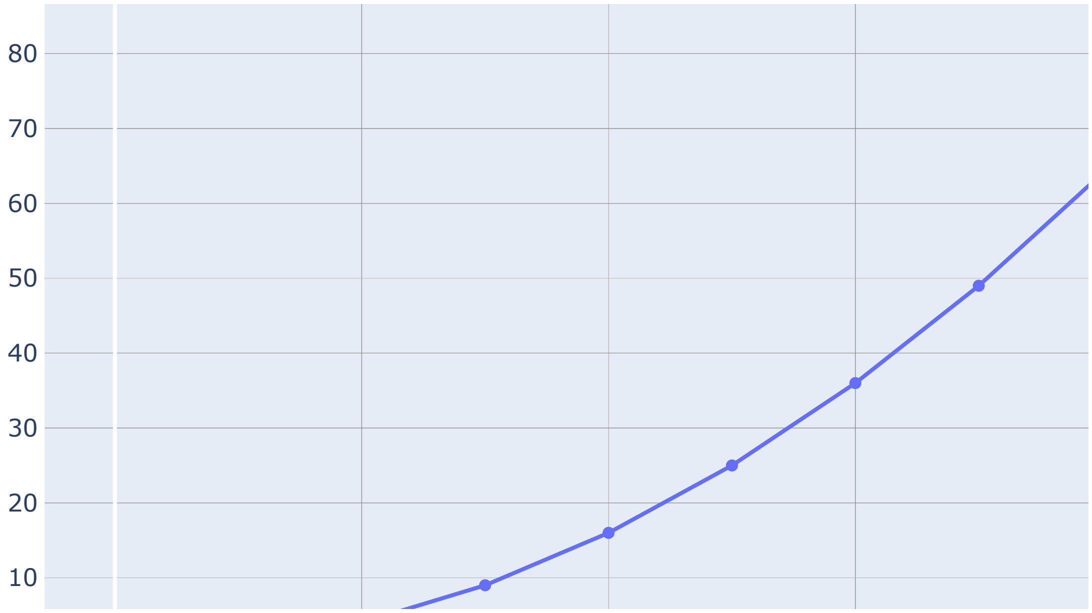
Using Plotly's Graph Objects

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

Line Plot with go.Scatter

In [6]:

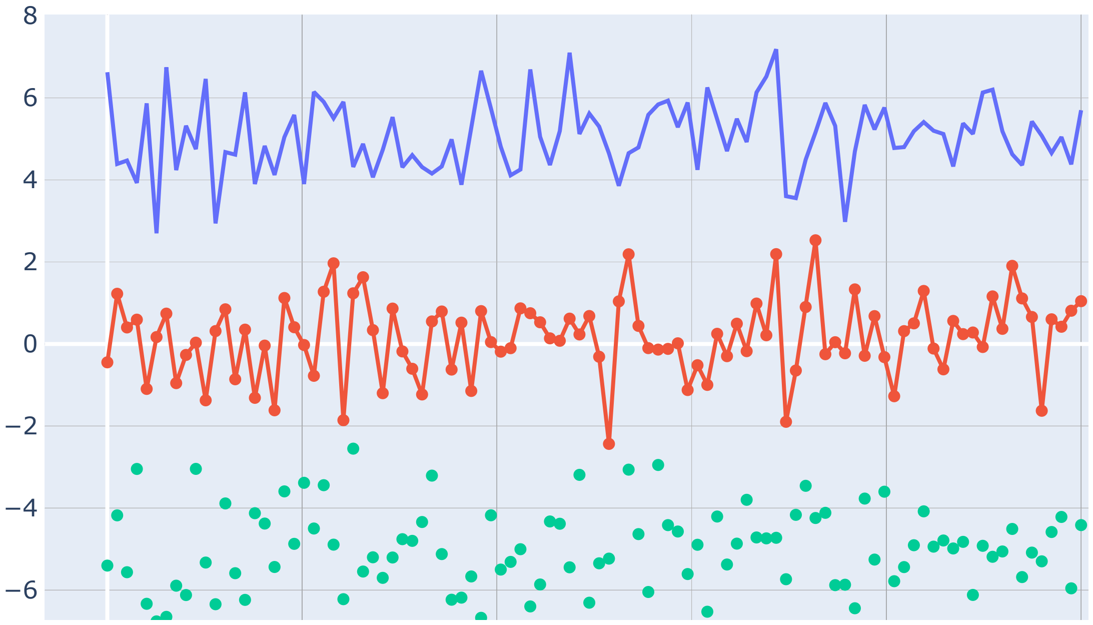
```
import plotly.graph_objects as go
import numpy as np

x = np.arange(10)

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



Line Plot Modes

In [7]:

```
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()
```

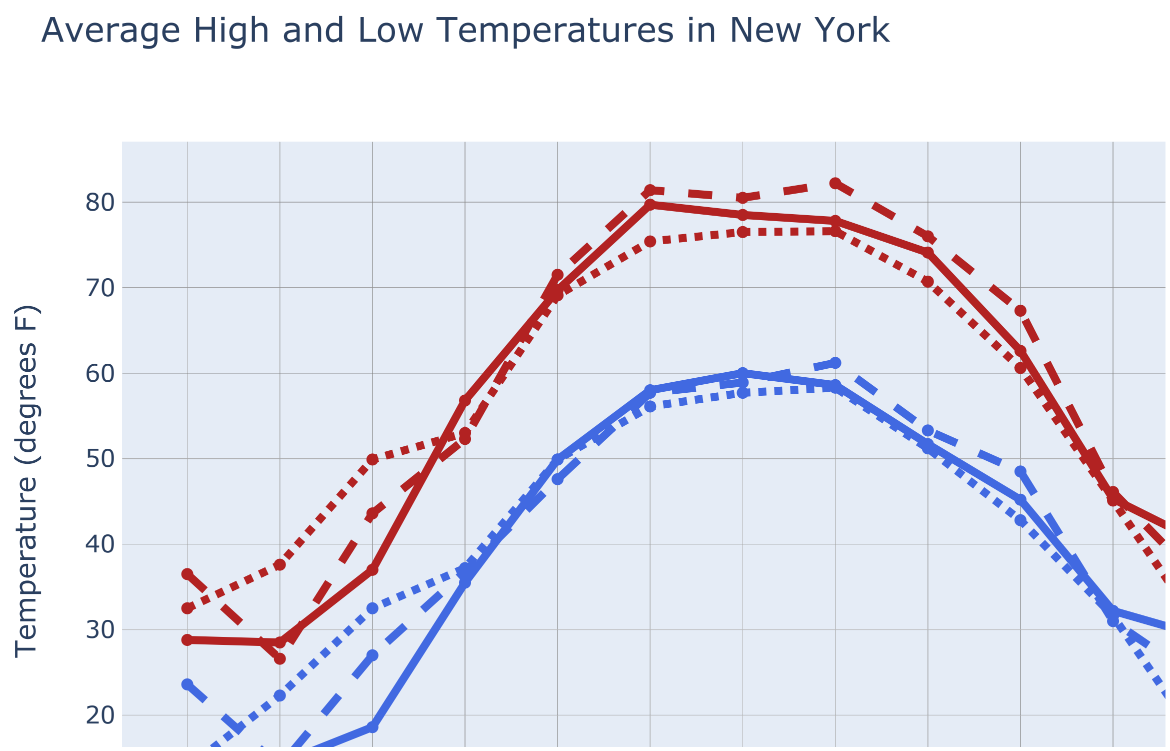


Style Line Plots

Here we style the color and dash of the traces, adds trace names, modify line width, and adds plot and axes titles.

In [8]:

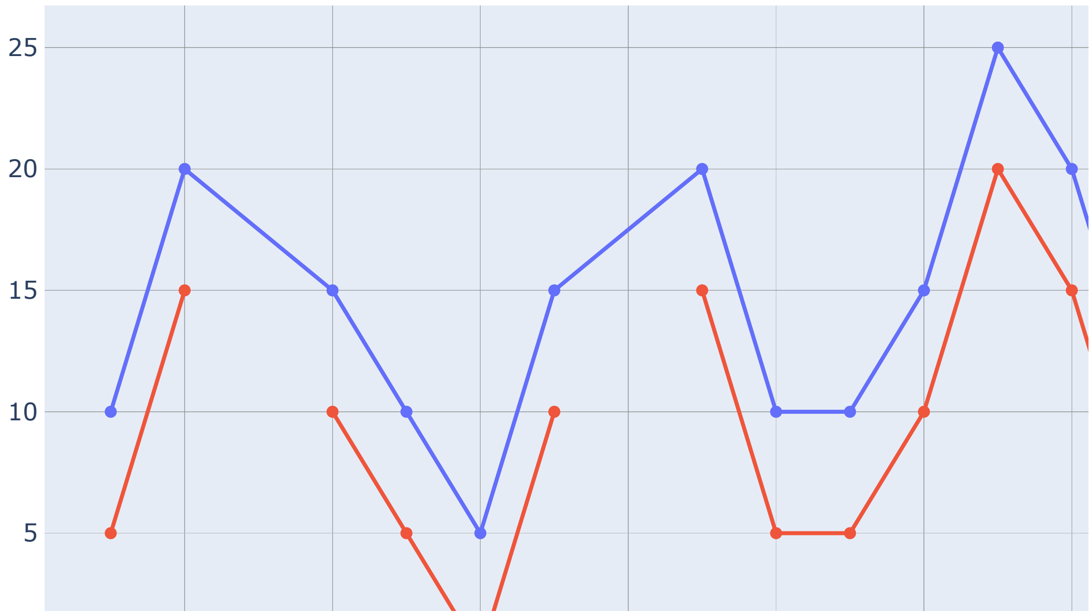
```
import plotly.graph_objects as go

# 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 'das
h', 'dot', and 'dashdot'
))
fig.add_trace(go.Scatter(x=month, y=low_2007, name='Low 2007',
                         line = dict(color='royalblue', width=4, dash='d
ash')))
fig.add_trace(go.Scatter(x=month, y=high_2000, name='High 2000',
                         line = dict(color='firebrick', width=4, dash='d
ot')))
fig.add_trace(go.Scatter(x=month, y=low_2000, name='Low 2000',
                         line=dict(color='royalblue', width=4, dash='do
t')))

# 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

- Data is not always clean and full as we wish
- The *connectgaps* option determines if missing values in the provided data are shown as a gap in the graph or not.

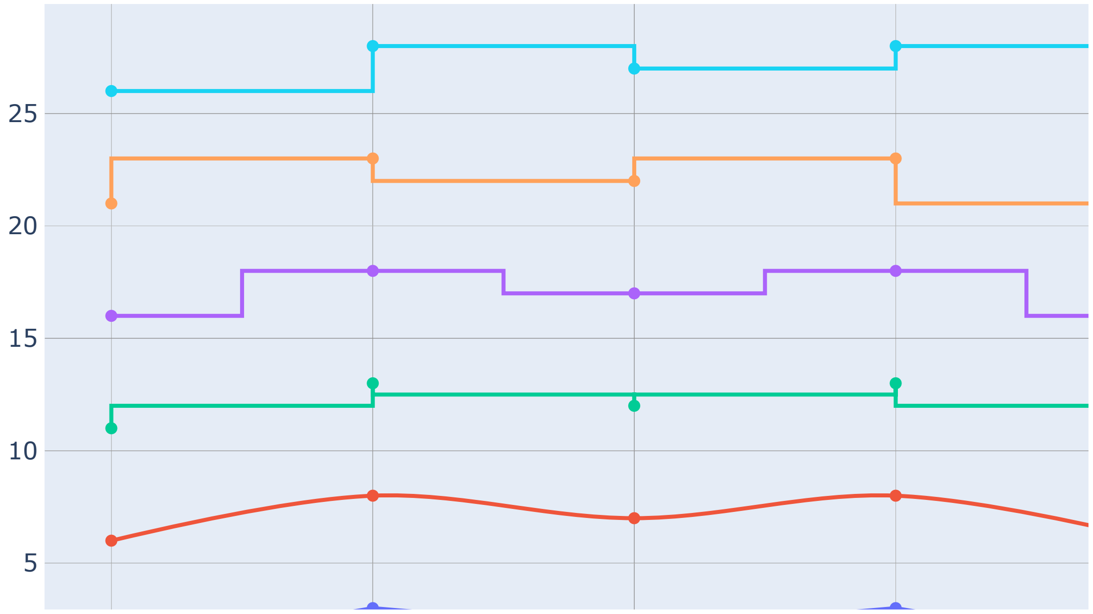
```
In [9]: import plotly.graph_objects as go

        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

In [10]:

```
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", 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()
```



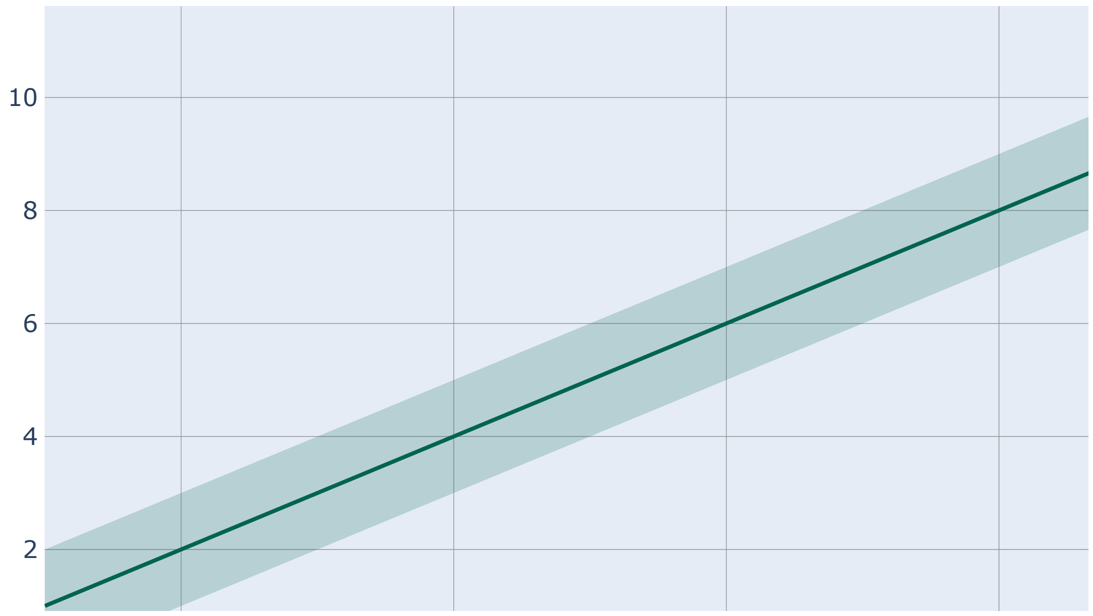
Filled Lines

- This is useful when drawing filled areas on both sides (or a single side) of a line.
- Examples are when one wants to plot a moving average or moving standard deviation

In [11]:

```
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='Fair',
))"""
```

```
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()
```



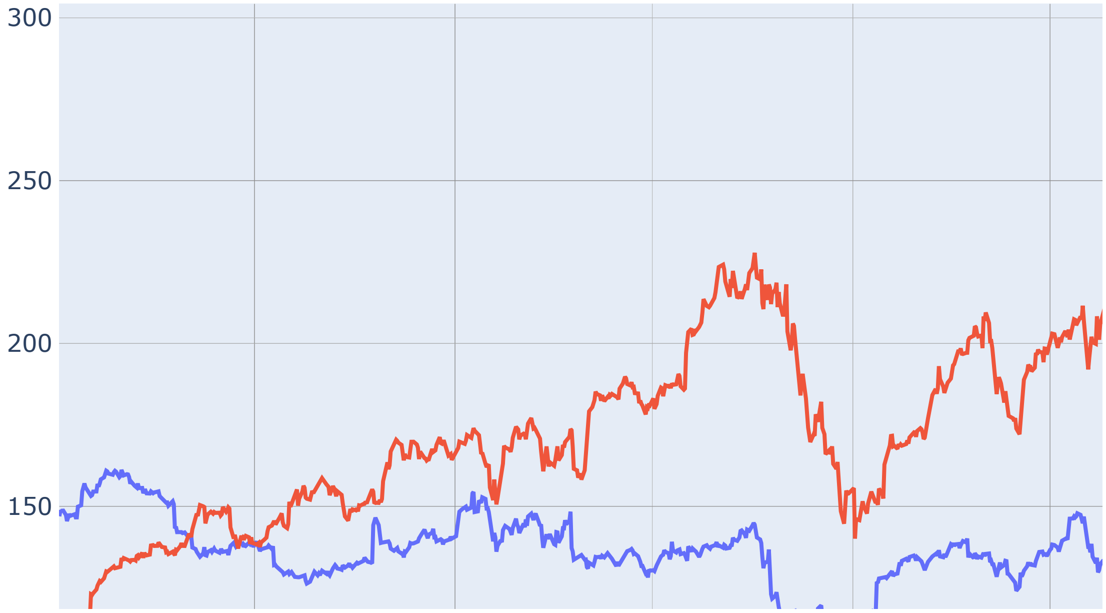
Plotting Moving Average and Moving Standard Deviation

In [12]:

```python
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/nsadawi/SampleCSVDatasets/master/CSVs/stock-prices-2017-2019.csv')
#df.columns
```

In [13]:

```python
fig = go.Figure()

fig.add_trace(go.Scatter(x = df['Date'], y=df['IBM'], name="IBM", line_shape='linear'))
fig.add_trace(go.Scatter(x = df['Date'], y=df['AAPL'], name="AAPL", line_shape='linear'))

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

fig.show()
```



In [14]:

```python
def moving_avg(values,window):
    """A function to compute incremental (not centred) moving avg (not efficiently!)
    It loops through an input list and computes avg of the previous elements (according to window)
    Also computes moving avg of first window - 1 elements!
    Args:
        param1 (list): A list of numerical values.
        param2 (int): The window size.

    Returns:
        a list of moving avg values
    """
    mavg = list()
    for i in range(len(values)):
        if i >= window:
            mavg.append((np.mean(values[(i-window+1):i+1])))
        else:
            mavg.append((np.mean(values[0:i+1])))

    return mavg
```

In [15]:

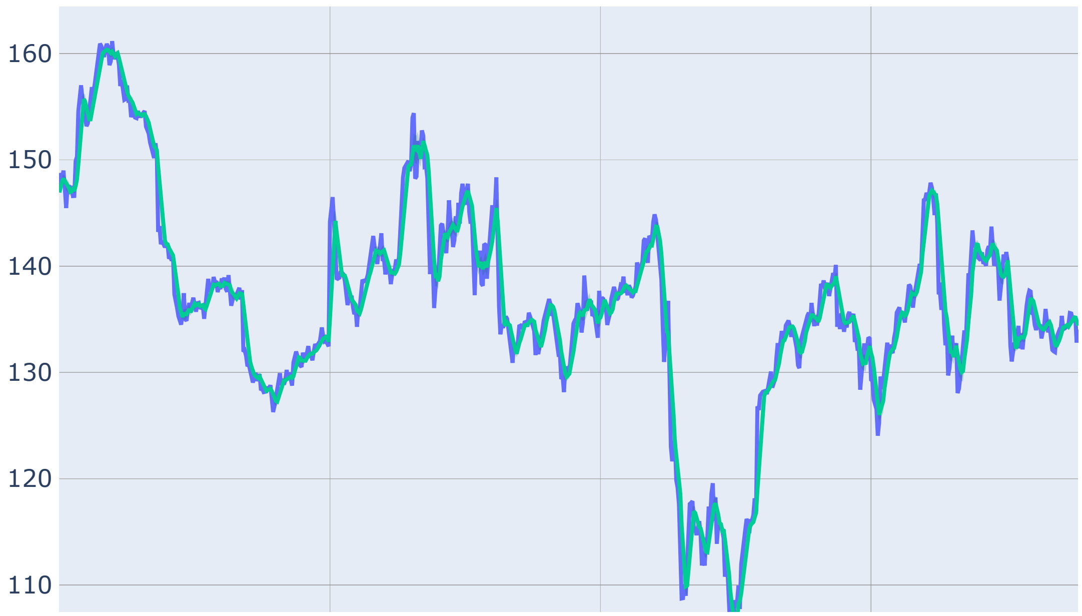
```python
def moving_std(values,window):
    """A function to compute incremental (not centred) moving STD (not efficiently!)
    It loops through an input list and computes STD of the previous elements (according to window)
    Also computes moving STD of first window - 1 elements!
    Args:
        param1 (list): A list of numerical values.
        param2 (int): The window size.

    Returns:
        a list of moving STD values
    """
    mstd = list()
    for i in range(0,len(values)):
        if i >= window:
            mstd.append(np.std(values[(i-window+1):i+1]))
        else:
            mstd.append(np.std(values[0:i+1]))

    return mstd
```

In [16]:
```
def get_lower_upper_bounds(ma,mstd):
    """This function to computes the lower an upper bounds for moving STD around moving Avg
    It loops through the MA list and computes the lower and upper bound values at each point
    For the current ma point, upper bound = value + (mstd at that point/2)
    and lower bound = value - (mstd at that point/2)
    Args:
        param1 (list): A list of Moving Avg values.
        param2 (list): A list of Moving STD values.

    Returns:
        two lists of lower and upper bound values
    """
    upper_bound = list()
    lower_bound = list()
    for idx, val in enumerate(ma):
        ub = val + (mstd[idx]/2)
        lb = val - (mstd[idx]/2)
        upper_bound.append(ub)
        lower_bound.append(lb)
    return lower_bound,upper_bound
```

In [17]:
```
window = 5
data = df['IBM']
x = list(range(len(data)))
#compute moving avg and moving std
ma = moving_avg(data,window)
mstd = moving_std(data,window)
#get lower and upper values of curves surrounding the MA curve
#to represent the moving STD band around MA curve
y_lower,y_upper = get_lower_upper_bounds(ma,mstd)

#values of MA to be plotted
ma_trace = go.Scatter(x=x, y=ma, mode='lines', name="Moving Average")
```

In [18]:

```
fig = go.Figure()

fig.add_trace(go.Scatter(x = x, y=data, name="IBM", line_shape='linear'
))
#fig.add_trace(ma_trace)
fig.add_trace(go.Scatter(
    x=x+x[::-1],# notice how we append x and x reversed to make the full area
    y=y_upper+y_lower[::-1],# notice how we append y and y reversed to make the full area
    fill='toself',
    fillcolor='rgba(0,100,80,0.2)',
    line_color='rgba(255,255,255,0)',
    showlegend=False,
    name='Moving STD',
))
fig.add_trace(ma_trace)


fig.update_layout(legend=dict(y=0.5,  font_size=16))

fig.show()
```



In [ ]: