# Time Series and Date Axes in Python

How to plot date and time in python.



## Time Series using Axes of type `date`

Time series can be represented using either `plotly.express` functions (`px.line`, `px.scatter`, `px.bar` etc) or `plotly.graph_objects` charts objects (`go.Scatter`, `go.Bar` etc). For more examples of such charts, see the documentation of line and scatter plots (https://plotly.com/python/line-and-scatter/) or bar charts (/python/bar-charts/).

For financial applications, Plotly can also be used to create Candlestick charts (/python/candlestick-charts/) and OHLC charts (/python/ohlc-charts/), which default to date axes.

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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html) or datetime NumPy array (https://docs.scipy.org/doc/numpy/reference/arrays.datetime.html).

```
# Using plotly.express
import plotly.express as px

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

```
# Using graph_objects
import plotly.graph_objects as go

import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = go.Figure([go.Scatter(x=df['Date'], y=df['AAPL.High'])])
fig.show()
```

the

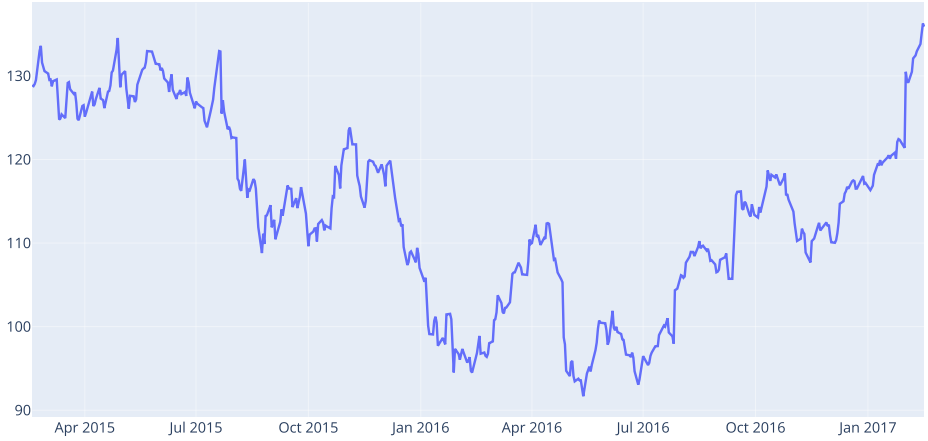
Minor

eriod

Range

:tons

ɔy Zoom




## Time Series 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/)**.**

the

Minor

eriod

Range

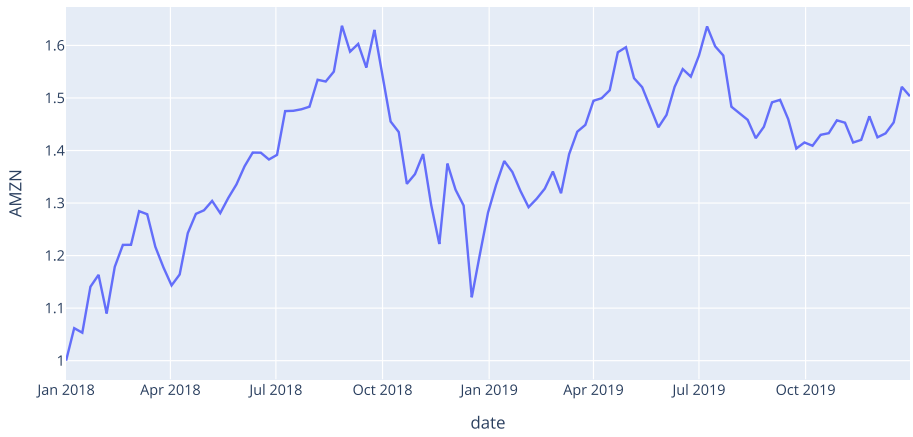
:tons

ɔy Zoom

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

app = Dash(__name__)


app.layout = html.Div([
    html.H4('Stock price analysis'),
    dcc.Graph(id="time-series-chart"),
    html.P("Select stock:"),
    dcc.Dropdown(
        id="ticker",
        options=["AMZN", "FB", "NFLX"],
        value="AMZN",
        clearable=False,
    ),
])


@app.callback(
    Output("time-series-chart", "figure"),
    Input("ticker", "value"))
def display_time_series(ticker):
    df = px.data.stocks() # replace with your own data source
    fig = px.line(df, x='date', y=ticker)
```

DOWNLOAD

## Stock price analysis




Select stock:

AMZN



## Different Chart Types on Date Axes

Any kind of cartesian chart can be placed on date axes, for example this bar chart of relative stock ticker values.

```
import plotly.express as px

df = px.data.stocks(indexed=True)-1
fig = px.bar(df, x=df.index, y="GOOG")
fig.show()
```

the

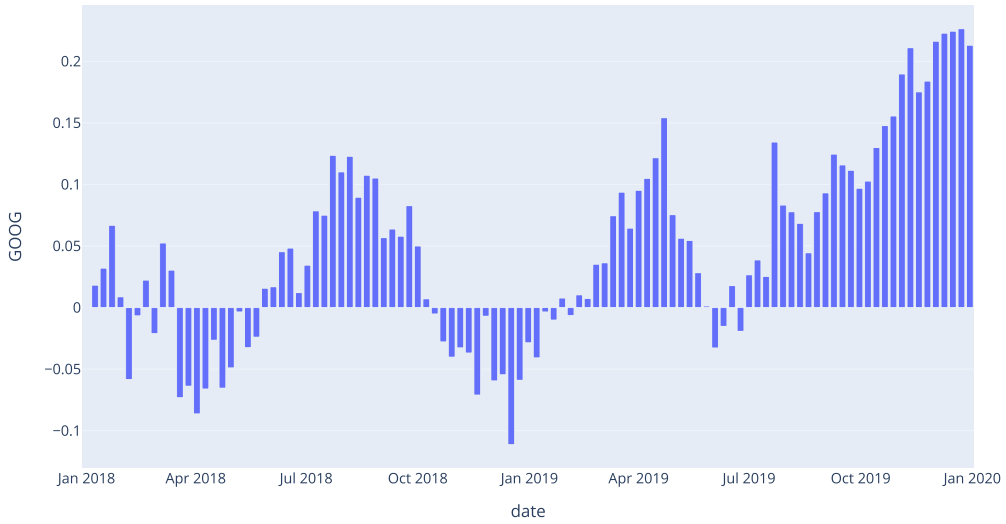
Minor

eriod

Range

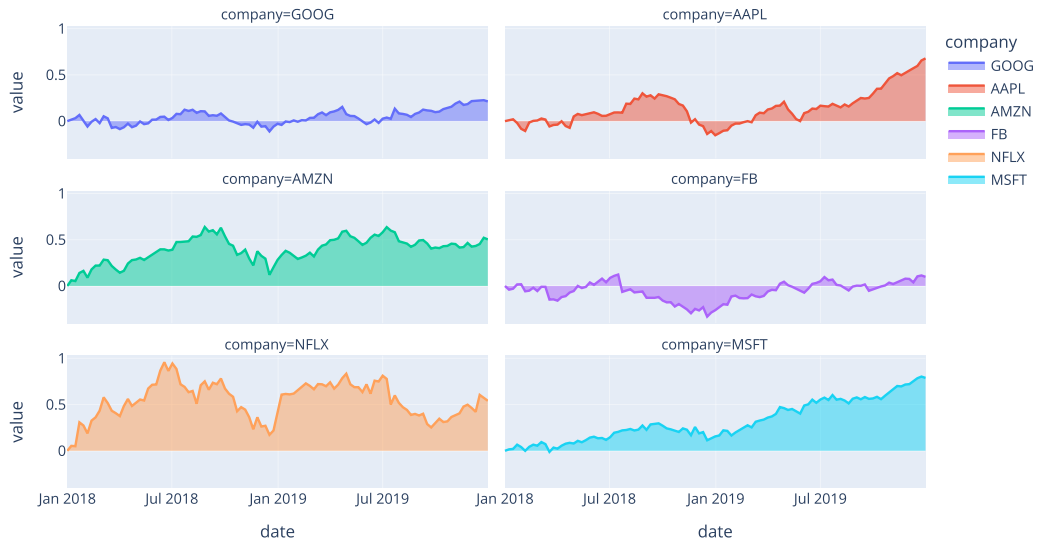
tons

y Zoom




Or this facetted (/python/facet-plots/) area plot:

```
import plotly.express as px

df = px.data.stocks(indexed=True)-1
fig = px.area(df, facet_col="company", facet_col_wrap=2)
fig.show()
```



## Configuring Tick Labels

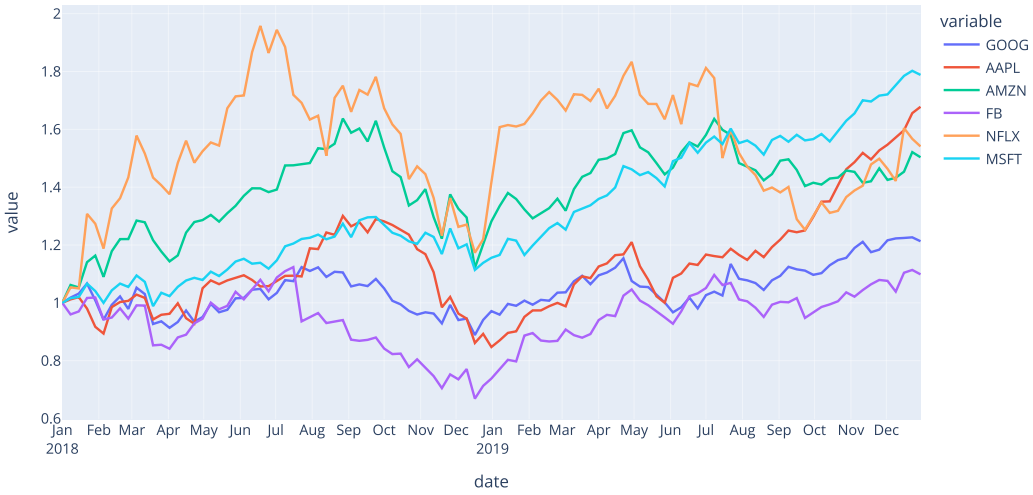
By default, the tick labels (and optional ticks) are associated with a specific grid-line, and represent an *instant* in time, for example, "00:00 on February 1, 2018". Tick labels can be formatted using the `tickformat` attribute (which accepts the d3 time-format formatting strings (https://github.com/d3/d3-time-format)) to display only the month and year, but they still represent an instant by default, so in the figure below, the text of the label "Feb 2018" spans part of the month of January and part of the month of February. The `dtick` attribute controls the spacing between gridlines, and the "M1" setting means "1 month". This attribute also accepts a number of milliseconds, which can be scaled up to days by multiplying by `24*60*60*1000`.

Date axis tick labels have the special property that any portion after the first instance of '\n' in `tickformat` will appear on a second line only once per unique value, as with the year numbers in the example below. To have the year number appear on every tick label, '<br>' should be used instead of '\n'.

Note that by default, the formatting of values of X and Y values in the hover label matches that of the tick labels of the corresponding axes, so when customizing the tick labels to something broad like "month", it's usually necessary to customize the hover label (/python/hover-text-and-formatting/) to something narrower like the actual date, as below.

```
import plotly.express as px
df = px.data.stocks()
fig = px.line(df, x="date", y=df.columns,
              hover_data={"date": "|%B %d, %Y"},
              title='custom tick labels')
fig.update_xaxes(
    dtick="M1",
    tickformat="%b\n%Y")
fig.show()
```

custom tick labels




## Moving Tick Labels to the Middle of the Period

*new in 4.10*

By setting the `ticklabelmode` attribute to "period" (the default is "instant") we can move the tick labels to the middle of the period they represent. The gridlines remain at the beginning of each month (thanks to `dtick="M1"`) but the labels now span the month they refer to.

```
import plotly.express as px
df = px.data.stocks()
fig = px.line(df, x="date", y=df.columns,
              hover_data={"date": "|%B %d, %Y"},
              title='custom tick labels with ticklabelmode="period"')
fig.update_xaxes(
    dtick="M1",
    tickformat="%b\n%Y",
    ticklabelmode="period")
fig.show()
```

the

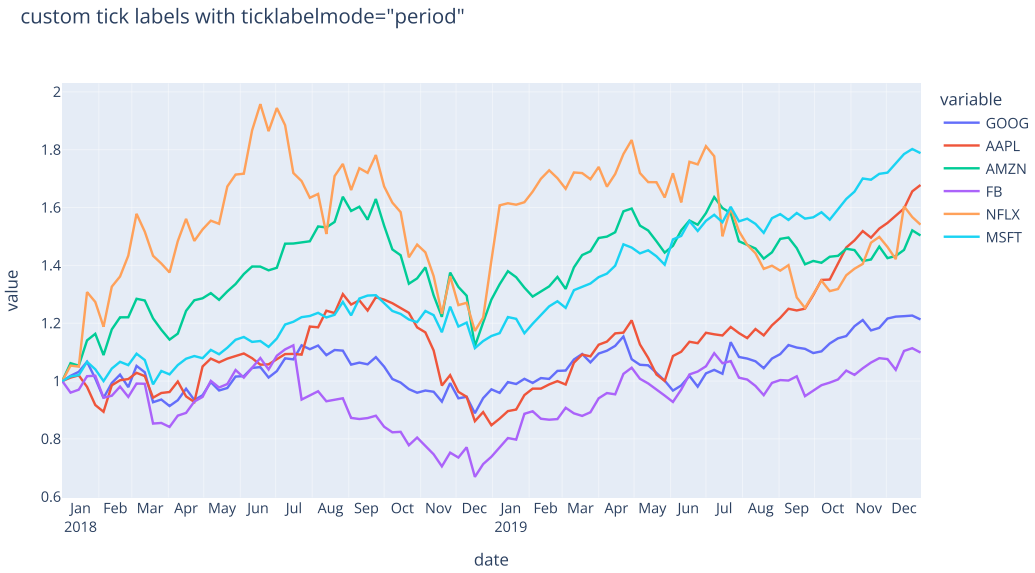
Minor

eriod

Range

:tons

ɔy Zoom




## Adding Minor Ticks

*new in 5.8*

You can add minor ticks to an axis with `minor`. This takes a `dict` of properties to apply to minor ticks. See the figure reference (https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-minor) for full details on the accepted keys in this dict.

In this example, we've added minor ticks to the inside of the x-axis and turned on minor grid lines.

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

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

fig.update_xaxes(minor=dict(ticks="inside", showgrid=True))

fig.show()
```

the

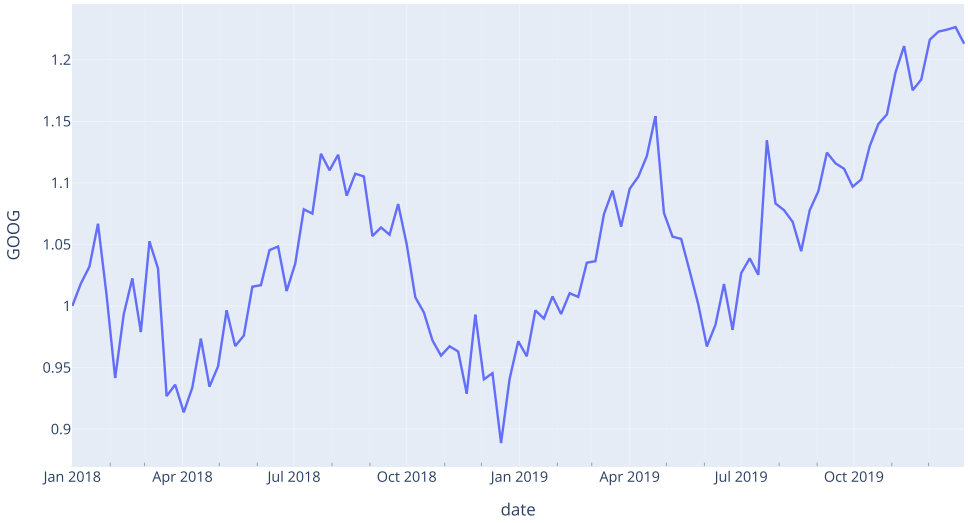
Minor

eriod

Range

:tons

ɔy Zoom




## Monthly Period Labels With Weekly Minor Ticks

*new in 5.8*

You can set `dtick` on `minor` to control the spacing for minor ticks and grid lines. In the following example, by setting `dtick=7*24*60*60*1000` (the number of milliseconds in a week) and setting `tick0="2016-07-03"` (the first Sunday in our data), a minor tick and grid line is displayed for the start of each week. When zoomed out, we can see where each month and week begins and ends.

the

Minor

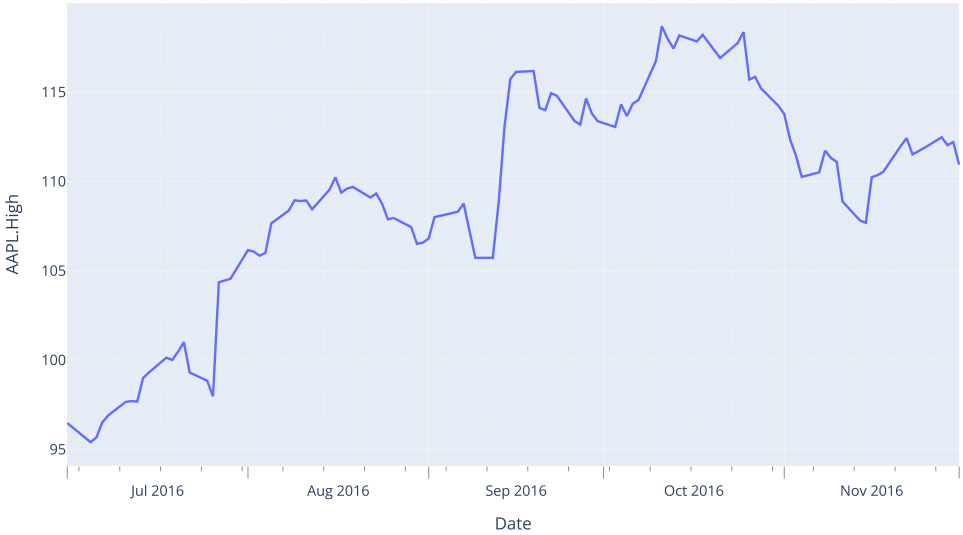
eriod

Range

:tons

ɔy Zoom

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

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')
df = df.loc[(df["Date"] >= "2016-07-01") & (df["Date"] <= "2016-12-01")]

fig = px.line(df, x='Date', y='AAPL.High')
fig.update_xaxes(ticks= "outside",
                 ticklabelmode= "period",
                 tickcolor= "black",
                 ticklen=10,
                 minor=dict(
                     ticklen=4,
                     dtick=7*24*60*60*1000,
                     tick0="2016-07-03",
                     griddash='dot',
                     gridcolor='white')
                )

fig.show()
```




## Summarizing Time-series Data with Histograms

Plotly histograms (/python/histograms/) are powerful data-aggregation tools which even work on date axes. In the figure below, we pass in daily data and display it as monthly averages by setting histfunc="avg and xbins_size="M1".

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

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.histogram(df, x="Date", y="AAPL.Close", histfunc="avg", title="Histogram on Date Axes")
fig.update_traces(xbins_size="M1")
fig.update_xaxes(showgrid=True, ticklabelmode="period", dtick="M1", tickformat="%b\n%Y")
fig.update_layout(bargap=0.1)
fig.add_trace(go.Scatter(mode="markers", x=df["Date"], y=df["AAPL.Close"], name="daily"))
fig.show()
```

the

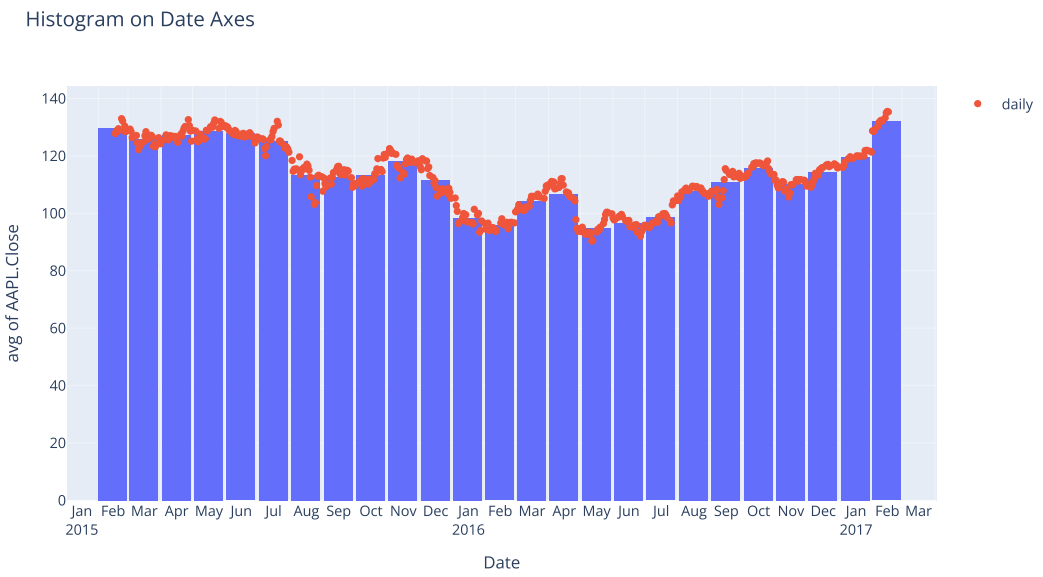
Minor

eriod

Range

:tons

ɔy Zoom




## Displaying Period Data

*new in 4.11*

If your data coded "January 1" or "January 31" in fact refers to data collected throughout the month of January, for example, you can configure your traces to display their marks at the start end, or middle of the month with the `xperiod` and `xperiodalignment` attributes. In the example below, the raw data is all coded with an X value of the 10th of the month, but is binned into monthly periods with `xperiod="M1"` and then displayed at the start, middle and end of the period.

the

Minor

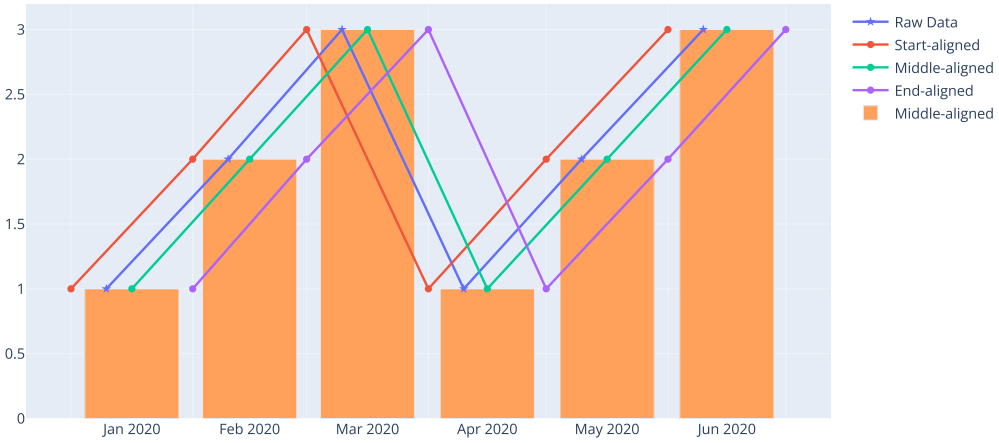
eriod

Range

tons

y Zoom

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

df = pd.DataFrame(dict(
    date=["2020-01-10", "2020-02-10", "2020-03-10", "2020-04-10", "2020-05-10", "2020-06-10"],
    value=[1,2,3,1,2,3]
))

fig = go.Figure()
fig.add_trace(go.Scatter(
    name="Raw Data",
    mode="markers+lines", x=df["date"], y=df["value"],
    marker_symbol="star"
))
fig.add_trace(go.Scatter(
    name="Start-aligned",
    mode="markers+lines", x=df["date"], y=df["value"],
    xperiod="M1",
    xperiodalignment="start"
))
fig.add_trace(go.Scatter(
    name="Middle-aligned",
    mode="markers+lines", x=df["date"], y=df["value"],
    xperiod="M1",
    xperiodalignment="middle"
))
fig.add_trace(go.Scatter(
    name="End-aligned",
    mode="markers+lines", x=df["date"], y=df["value"],
    xperiod="M1",
    xperiodalignment="end"
))
fig.add_trace(go.Bar(
    name="Middle-aligned",
    x=df["date"], y=df["value"],
    xperiod="M1",
    xperiodalignment="middle"
))
fig.update_xaxes(showgrid=True, ticklabelmode="period")
fig.show()
```



## Hover Templates with Mixtures of Period data

*New in v5.0*

When displaying periodic data with mixed-sized periods (i.e. quarterly and monthly) in conjunction with x or x unified hovermodes and using hovertemplate (https://plotly.com/python/hover-text-and-formatting/), the xhoverformat attribute can be used to control how each period's X value is displayed, and the special %{xother} hover-template directive can be used to control how the X value is displayed for points that do not share the exact X coordinate with the point that is being hovered on. %{xother} will return an empty string when the X value is the one being hovered on, otherwise it will return (%{x}). The special %{_xother}, %{xother_} and %{_xother_} variations will display with spaces before, after or around the parentheses, respectively.

the

Minor

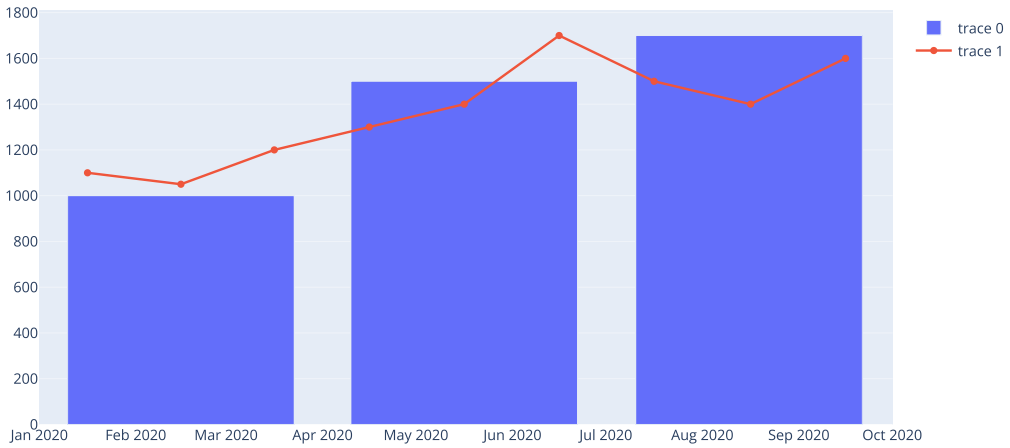
eriod

Range

:tons

ɔy Zoom

```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Bar(
    x=["2020-01-01", "2020-04-01", "2020-07-01"],
    y=[1000, 1500, 1700],
    xperiod="M3",
    xperiodalignment="middle",
    xhoverformat="Q%q",
    hovertemplate="%{y}%{_xother}"
))

fig.add_trace(go.Scatter(
    x=["2020-01-01", "2020-02-01", "2020-03-01",
      "2020-04-01", "2020-05-01", "2020-06-01",
      "2020-07-01", "2020-08-01", "2020-09-01"],
    y=[1100,1050,1200,1300,1400,1700,1500,1400,1600],
    xperiod="M1",
    xperiodalignment="middle",
    hovertemplate="%{y}%{_xother}"
))

fig.update_layout(hovermode="x unified")
fig.show()
```




## Time Series Plot with Custom Date Range

The range can be set manually using either datetime.datetime objects, or date strings.

```
# Using plotly.express
import plotly.express as px

import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.line(df, x='Date', y='AAPL.High', range_x=['2016-07-01','2016-12-31'])
fig.show()
```

the

Minor

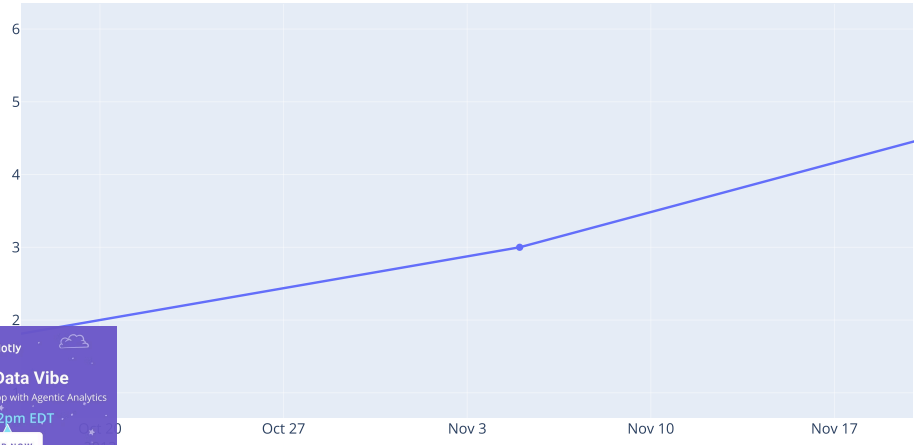
eriod

Range

:tons

ɔy Zoom


```
# Using graph_objects

import plotly.graph_objects as go
import datetime

x = [datetime.datetime(year=2013, month=10, day=4),
     datetime.datetime(year=2013, month=11, day=5),
     datetime.datetime(year=2013, month=12, day=6)]

fig = go.Figure(data=[go.Scatter(x=x, y=[1, 3, 6])])
# Use datetime objects to set xaxis range
fig.update_layout(xaxis_range=[datetime.datetime(2013, 10, 17),
                               datetime.datetime(2013, 11, 20)])
fig.show()
```



```
import plotly.graph_objects as go
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.line(df, x='Date', y='AAPL.High')

# Use date string to set xaxis range
fig.update_layout(xaxis_range=['2016-07-01','2016-12-31'],
                  title_text="Manually Set Date Range")
fig.show()
```

the

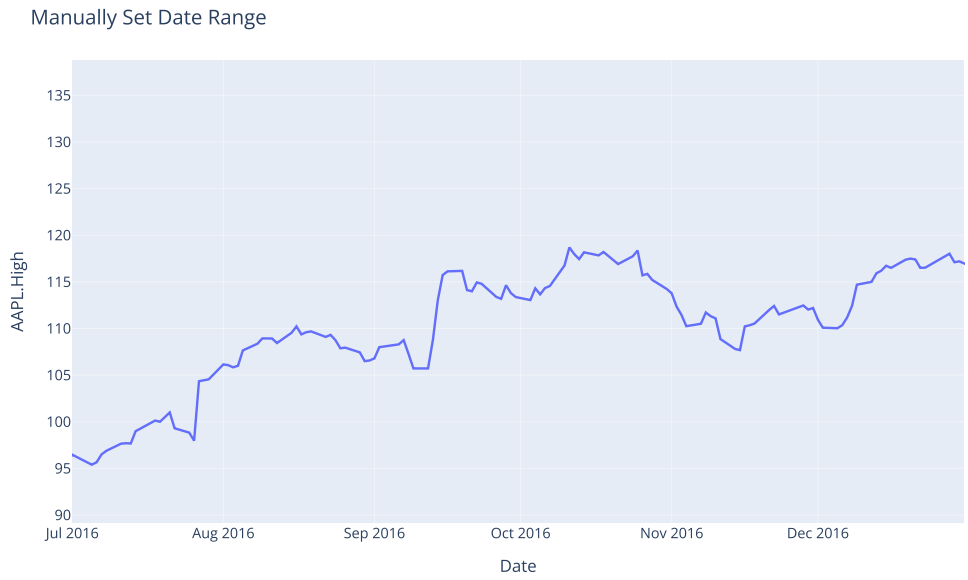
Minor

eriod

Range

:tons

ɔy Zoom




## Time Series With Range Slider

A range slider is a small subplot-like area below a plot which allows users to pan and zoom the X-axis while maintaining an overview of the chart. Check out the reference for more options: https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-rangeslider (https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-rangeslider)

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

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.line(df, x='Date', y='AAPL.High', title='Time Series with Rangeslider')

fig.update_xaxes(rangeslider_visible=True)
fig.show()
```

the

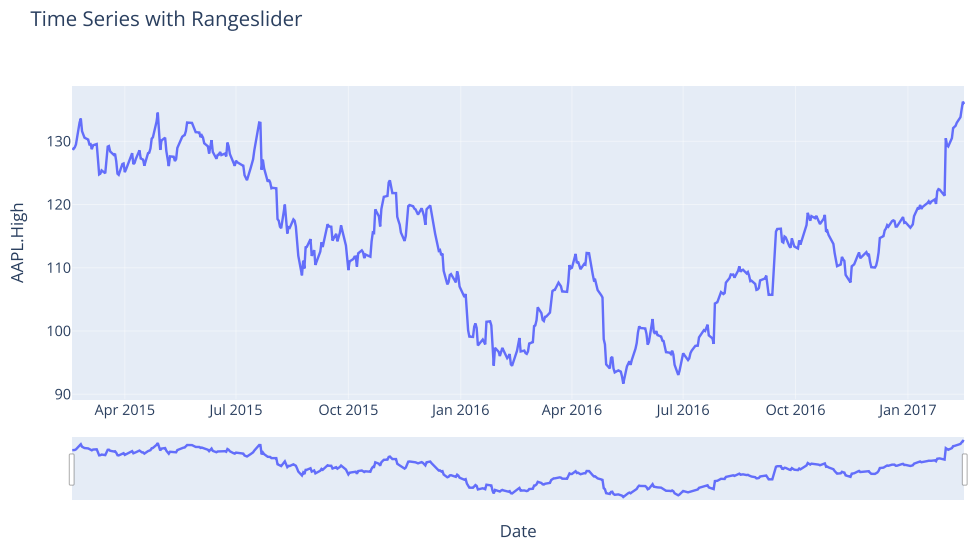
Minor

eriod

Range

:tons

ɔy Zoom




## Time Series with Range Selector Buttons

Range selector buttons are special controls that work well with time series and range sliders, and allow users to easily set the range of the x-axis. Check out the reference for more options: https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-rangeselector (https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-rangeselector)

the

Minor

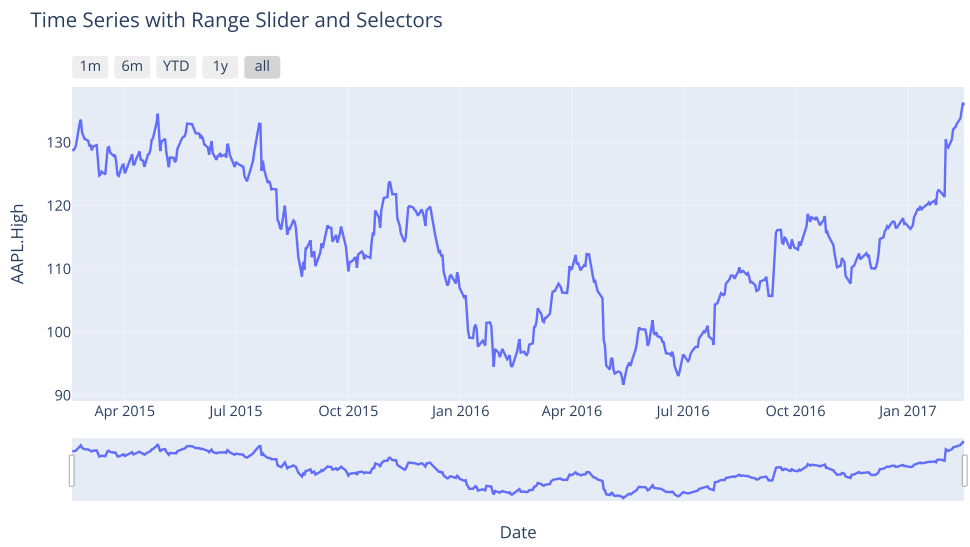
```
import plotly.express as px
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.line(df, x='Date', y='AAPL.High', title='Time Series with Range Slider and Selectors')

fig.update_xaxes(
    rangeslider_visible=True,
    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")
        ])
    )
)
fig.show()
```

eriod

Range

:tons

ɔy Zoom




## Customizing Tick Label Formatting by Zoom Level

The `tickformatstops` attribute can be used to customize the formatting of tick labels depending on the zoom level. Try zooming in to the chart below and see how the tick label formatting changes. Check out the reference for more options: https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-tickformatstops (https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-tickformatstops)

the

Minor

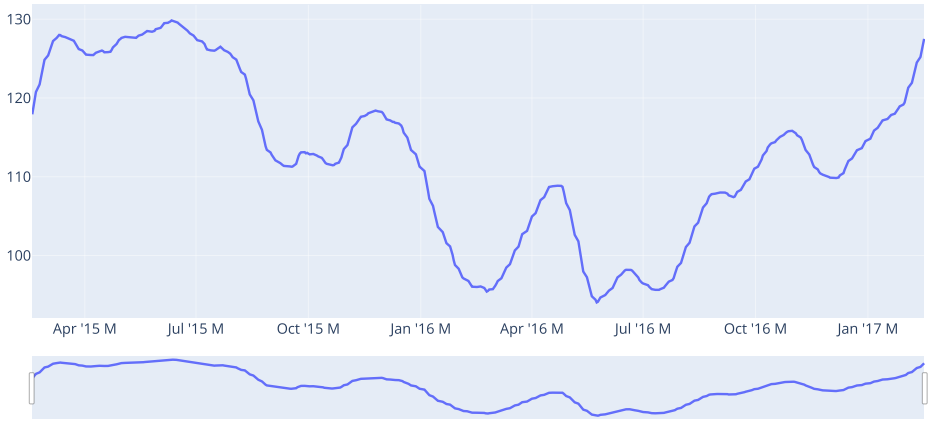
eriod

Range

:tons

ɔy Zoom

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

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = go.Figure(go.Scatter(
    x = df['Date'],
    y = df['mavg']
))

fig.update_xaxes(
    rangeslider_visible=True,
    tickformatstops = [
        dict(dtickrange=[None, 1000], value="%H:%M:%S.%L ms"),
        dict(dtickrange=[1000, 60000], value="%H:%M:%S s"),
        dict(dtickrange=[60000, 3600000], value="%H:%M m"),
        dict(dtickrange=[3600000, 86400000], value="%H:%M h"),
        dict(dtickrange=[86400000, 604800000], value="%e. %b d"),
        dict(dtickrange=[604800000, "M1"], value="%e. %b w"),
        dict(dtickrange=["M1", "M12"], value="%b '%y M"),
        dict(dtickrange=["M12", None], value="%Y Y")
    ]
)

fig.show()
```

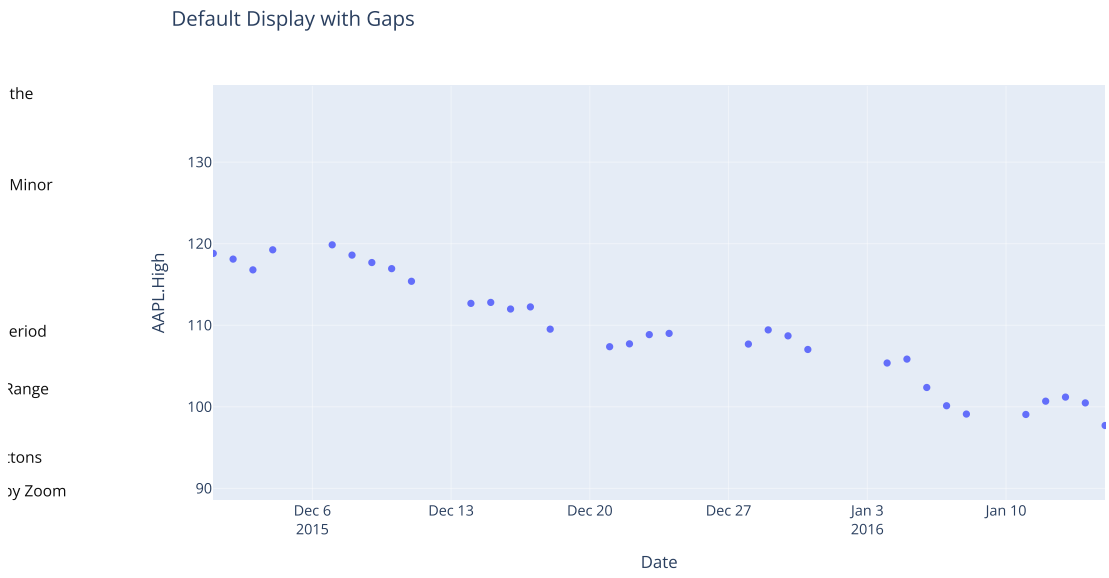



## Hiding Weekends and Holidays

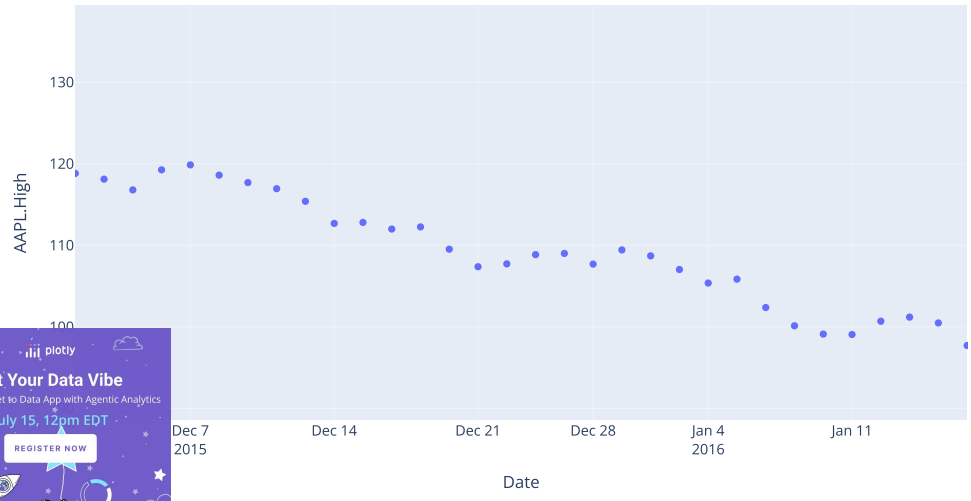
The rangebreaks attribute available on x- and y-axes of type date can be used to hide certain time-periods. In the example below, we show two plots: one in default mode to show gaps in the data, and one where we hide weekends and holidays to show an uninterrupted trading history. Note the smaller gaps between the grid lines for December 21 and January 4, where holidays were removed. Check out the reference for more options: https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-rangebreaks (https://plotly.com/python/reference/layout/xaxis/#layout-xaxis-rangebreaks)

> Note: a known limitation of this feature is that it does not support `scattergl` traces. When using this feature on plots with more than a few hundred data points with `px.scatter` or `px.line` or `px.area`, you may need to pass in `render_mode="svg"` to ensure that the underlying trace type is `scatter` and not `scattergl`.

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

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.scatter(df, x='Date', y='AAPL.High', range_x=['2015-12-01', '2016-01-15'],
                 title="Default Display with Gaps")
fig.show()
```




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

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/finance-charts-apple.csv')

fig = px.scatter(df, x='Date', y='AAPL.High', range_x=['2015-12-01', '2016-01-15'],
                 title="Hide Weekend and Holiday Gaps with rangebreaks")
fig.update_xaxes(
    rangebreaks=[
        dict(bounds=["sat", "mon"]), #hide weekends
        dict(values=["2015-12-25", "2016-01-01"])  # hide Christmas and New Year's
    ]
)
fig.show()
```



## Hiding Non-Business Hours

The `rangebreaks` feature described above works for hiding hourly periods as well.

```
import plotly.express as px
import pandas as pd
import numpy as np
np.random.seed(1)

work_week_40h = pd.date_range(start='2020-03-01', end='2020-03-07', freq="BH")

df = pd.DataFrame(dict(
    date = work_week_40h,
    value = np.cumsum(np.random.rand(40)-0.5)
))

fig = px.scatter(df, x="date", y="value",
                 title="Default Display with Gaps")
fig.show()
```

the

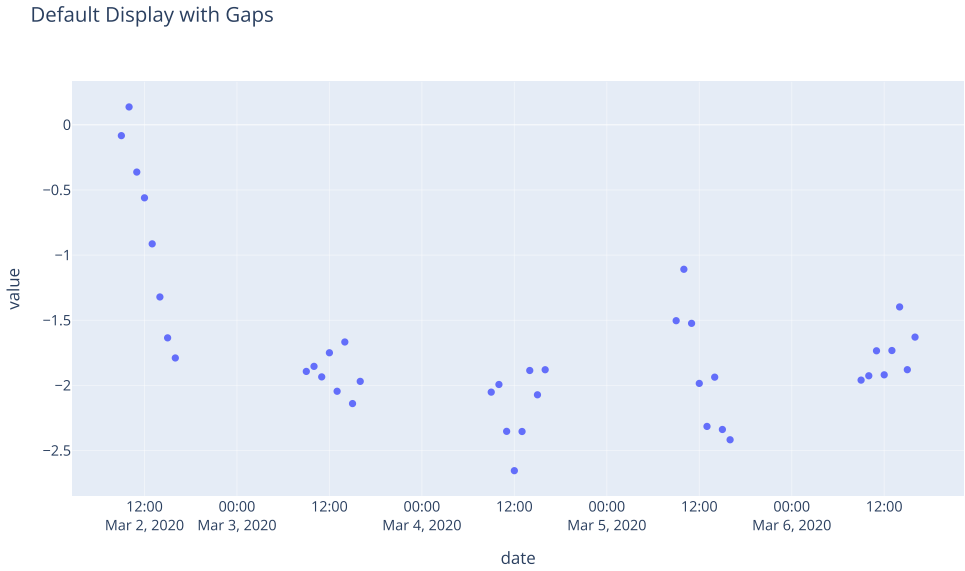
Minor

eriod

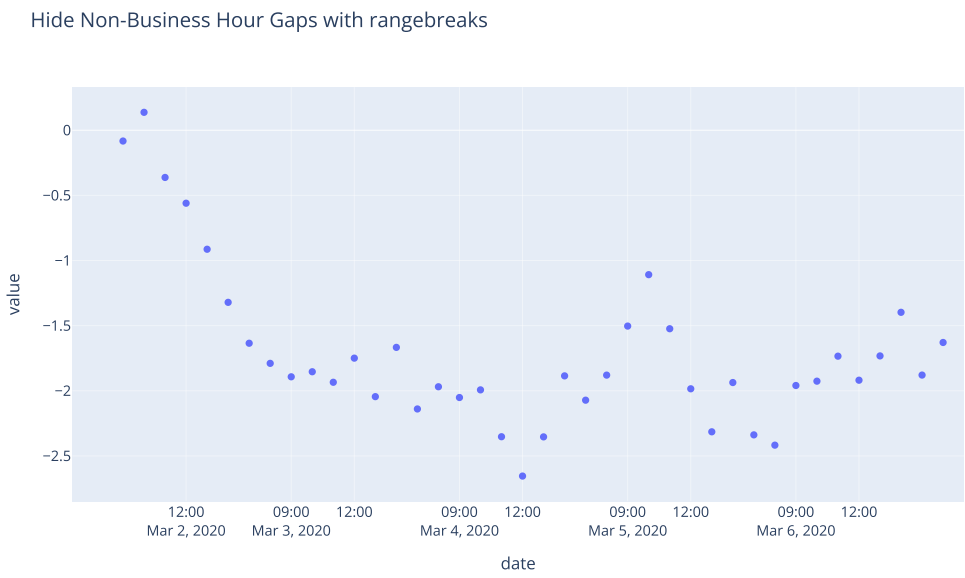
Range

:tons

ɔy Zoom



```
import plotly.express as px
import pandas as pd
import numpy as np
np.random.seed(1)

work_week_40h = pd.date_range(start='2020-03-01', end='2020-03-07', freq="BH")

df = pd.DataFrame(dict(
    date = work_week_40h,
    value = np.cumsum(np.random.rand(40)-0.5)
))

fig = px.scatter(df, x="date", y="value",
                 title="Hide Non-Business Hour Gaps with rangebreaks")
fig.update_xaxes(
    rangebreaks=[
        dict(bounds=[17, 9], pattern="hour"), #hide hours outside of 9am-5pm
    ]
)
fig.show()
```




## 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)
])

if __name__ == '__main__':
    app.run(debug=True, use_reloader=False)  # Turn off reloader if inside Jupyter
```

the

**JOIN OUR MAILING LIST**

Minor

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/)

eriod

**About Us**

Range

Careers (https://plotly.com/careers)

Resources (https://plotly.com/resources/)

:tons

Blog (https://medium.com/@plotlygraphs)

by Zoom

**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/)