# Facet and Trellis Plots in Python

How to make Facet and Trellis Plots in Python with Plotly.





## Facet and Trellis Plots

Facet plots, also known as trellis plots or small multiples, are figures made up of multiple subplots which have the same set of axes, where each subplot shows a subset of the data. While it is straightforward to use plotly's subplot capabilities (/python/subplots/) to make such figures, it's far easier to use the built-in facet_row and facet_col arguments in the various Plotly Express functions.

Plotly Express (/python/plotly-express/) is the easy-to-use, high-level interface to Plotly, which operates on a variety of types of data (/python/px-arguments/) and produces easy-to-style figures (/python/styling-plotly-express/).

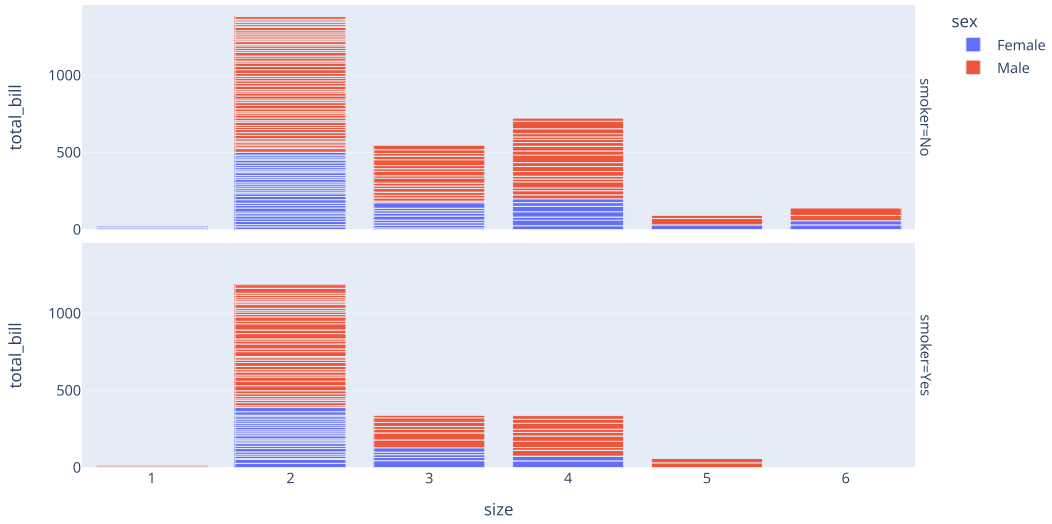
## Scatter Plot Column Facets

```
import plotly.express as px
df = px.data.tips()
fig = px.scatter(df, x="total_bill", y="tip", color="smoker", facet_col="sex")
fig.show()
```


## Bar Chart Row Facets

there's a more presentation-ready horizontal, faceted bar chart in the horizontal bar documentation (/python/horizontal-bar-charts/#Small-multiple-horizontal-bar-onent's-size-more-clearly-than-a-stacked-bar)

```
import plotly.express as px
df = px.data.tips()
fig = px.bar(df, x="size", y="total_bill", color="sex", facet_row="smoker")
fig.show()
```



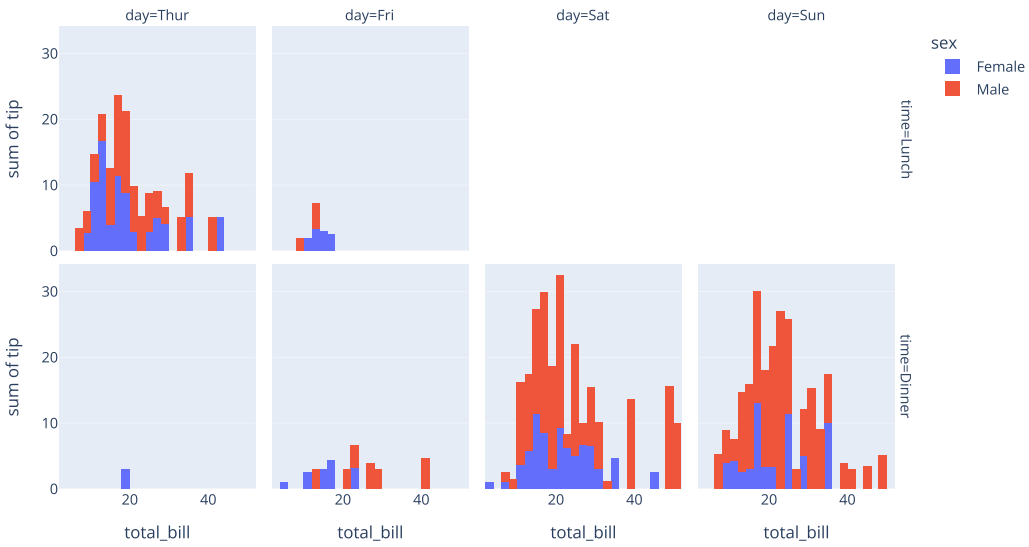

## Wrapping Column Facets

When the facet dimension has a large number of unique values, it is possible to wrap columns using the `facet_col_wrap` argument.

```
import plotly.express as px
df = px.data.gapminder()
fig = px.scatter(df, x='gdpPercap', y='lifeExp', color='continent', size='pop',
                 facet_col='year', facet_col_wrap=4)
fig.show()
```

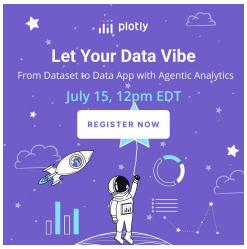
## Histogram Facet Grids

```
import plotly.express as px
df = px.data.tips()
fig = px.histogram(df, x="total_bill", y="tip", color="sex", facet_row="time", facet_col="day",
       category_orders={"day": ["Thur", "Fri", "Sat", "Sun"], "time": ["Lunch", "Dinner"]})
fig.show()
```

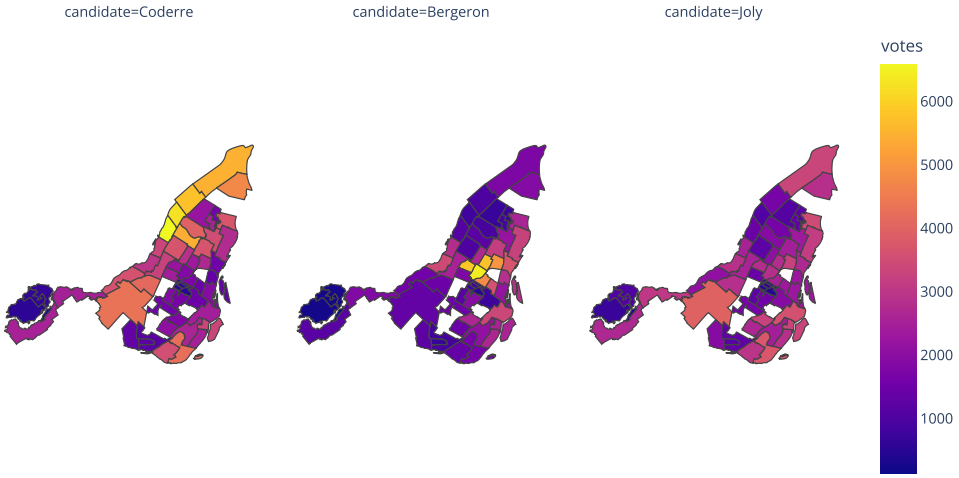



## Choropleth Column Facets

*new in version 4.13*



```
import plotly.express as px

df = px.data.election()
df = df.melt(id_vars="district", value_vars=["Coderre", "Bergeron", "Joly"],
             var_name="candidate", value_name="votes")
geojson = px.data.election_geojson()

fig = px.choropleth(df, geojson=geojson, color="votes", facet_col="candidate",
                    locations="district", featureidkey="properties.district",
                    projection="mercator"
                   )
fig.update_geos(fitbounds="locations", visible=False)
fig.show()
```

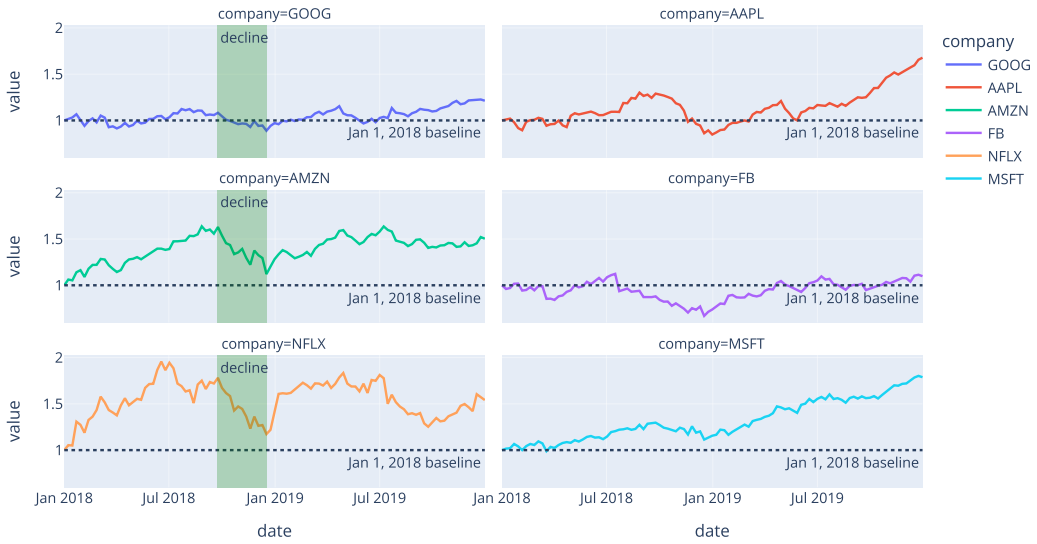



## Adding Lines and Rectangles to Facet Plots

*introduced in plotly 4.12*

It is possible to add labelled horizontal and vertical lines and rectangles (/python/horizontal-vertical-shapes/) to facet plots using `.add_hline()`, `.add_vline()`, `.add_hrect()` or `.add_vrect()`. The default `row` and `col` values are `"all"` but this can be overridden, as with the rectangle below, which only appears in the first column.

```
import plotly.express as px

df = px.data.stocks(indexed=True)
fig = px.line(df, facet_col="company", facet_col_wrap=2)
fig.add_hline(y=1, line_dash="dot",
              annotation_text="Jan 1, 2018 baseline",
              annotation_position="bottom right")

fig.add_vrect(x0="2018-09-24", x1="2018-12-18", col=1,
              annotation_text="decline", annotation_position="top left",
              fillcolor="green", opacity=0.25, line_width=0)
fig.show()
```

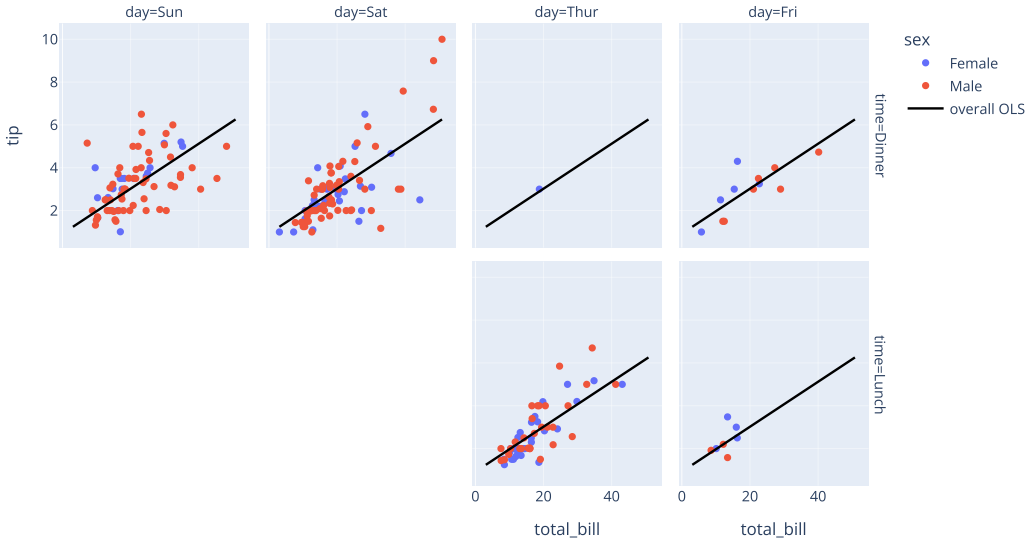



## Adding the Same Trace to All Facets

*introduced in plotly 4.12*

The .add_trace() method can be used to add a copy of the same trace to each facet, for example an overall linear regression line as below. The legendgroup/showlegend pattern below is recommended to avoid having a separate legend item for each copy of the trace. Note that as of v5.2.1, there is a built-in option to add an overall trendline to all facets (https://plotly.com/python/linear-fits/) that uses this technique under the hood.

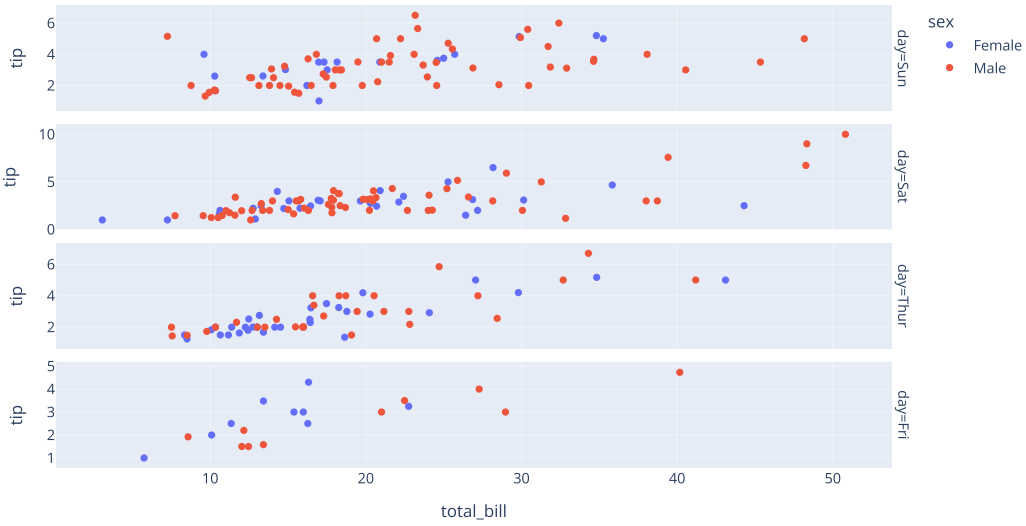
```
import plotly.express as px
df = px.data.tips()
fig = px.scatter(df, x="total_bill", y="tip", color='sex',
                 facet_col="day", facet_row="time")

import statsmodels.api as sm
import plotly.graph_objects as go
df = df.sort_values(by="total_bill")
model = sm.OLS(df["tip"], sm.add_constant(df["total_bill"])).fit()

#create the trace to be added to all facets
trace = go.Scatter(x=df["total_bill"], y=model.predict(),
                   line_color="black", name="overall OLS")

# give it a legend group and hide it from the legend
trace.update(legendgroup="trendline", showlegend=False)

# add it to all rows/cols, but not to empty subplots
fig.add_trace(trace, row="all", col="all", exclude_empty_subplots=True)

# set only the last trace added to appear in the legend
# `selector=-1` introduced in plotly v4.13
fig.update_traces(selector=-1, showlegend=True)
fig.show()
```



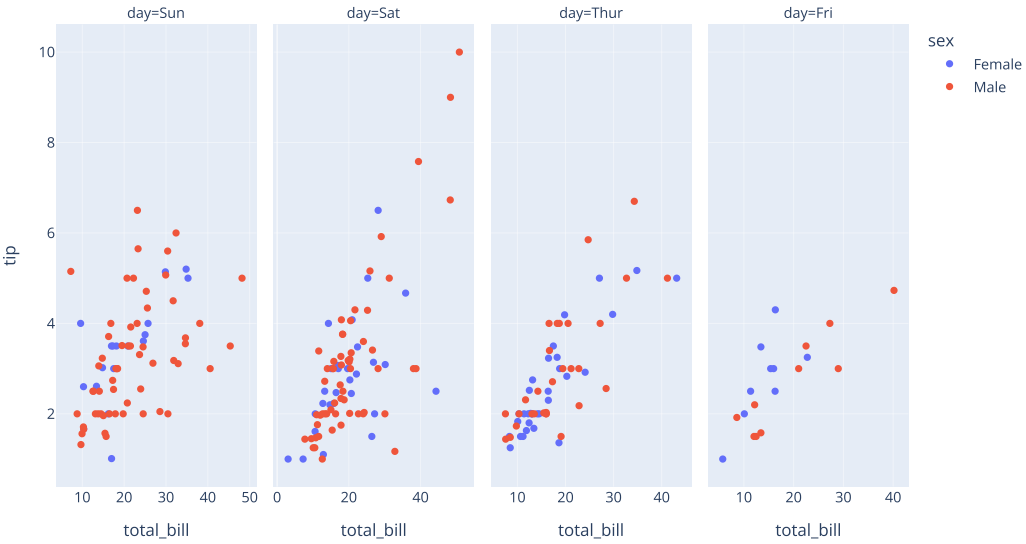

## Facets With Independent Axes

By default, facet axes are linked together: zooming inside one of the facets will also zoom in the other facets. You can disable this behaviour when you use `facet_row` only, by disabling matches on the Y axes, or when using `facet_col` only, by disabling matches on the X axes. It is not recommended to use this approach when using `facet_row` and `facet_col` together, as in this case it becomes very hard to understand the labelling of axes and grid lines.

```
import plotly.express as px
df = px.data.tips()
fig = px.scatter(df, x="total_bill", y="tip", color='sex', facet_row="day")
fig.update_yaxes(matches=None)
fig.show()
```




```
import plotly.express as px
df = px.data.tips()
fig = px.scatter(df, x="total_bill", y="tip", color='sex', facet_col="day")
fig.update_xaxes(matches=None)
fig.show()
```



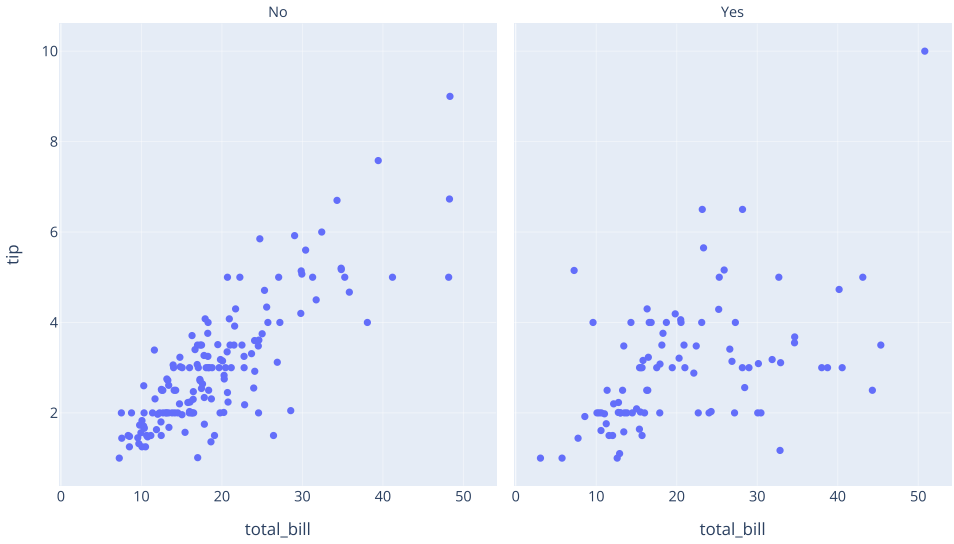

## Customizing Subplot Figure Titles



Since subplot figure titles are annotations (https://plotly.com/python/text-and-annotations/#simple-annotation), you can use the `for_each_annotation` function to modify the figure titles. For example to remove the equal-sign (=).

In the following example, we pass a lambda function to `for_each_annotation` in order to change the figure subplot titles from `smoker=No` and `smoker=Yes` to just `No`

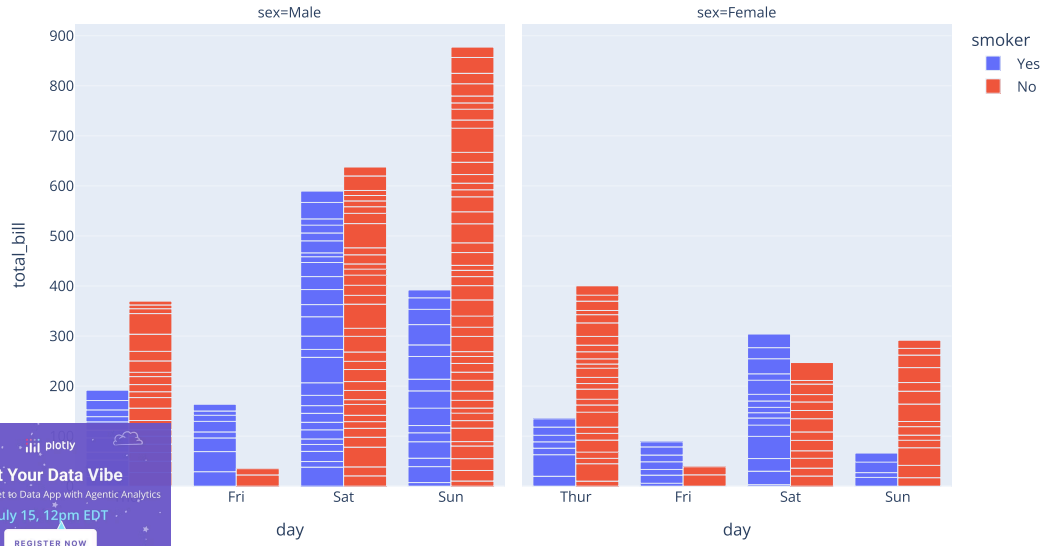
```
import plotly.express as px

fig = px.scatter(px.data.tips(), x="total_bill", y="tip", facet_col="smoker")
fig.for_each_annotation(lambda a: a.update(text=a.text.split("=")[-1]))
fig.show()
```




## Controlling Facet Ordering

By default, Plotly Express lays out categorical data in the order in which it appears in the underlying data. Every 2-d cartesian Plotly Express function also includes a category_orders keyword argument which can be used to control the order in which categorical axes are drawn (/python/categorical-axes/), but beyond that can also control the order in which discrete colors appear in the legend (/python/discrete-color/), and the order in which facets are laid out.

```
import plotly.express as px
df = px.data.tips()
fig = px.bar(df, x="day", y="total_bill", color="smoker", barmode="group", facet_col="sex",
             category_orders={"day": ["Thur", "Fri", "Sat", "Sun"],
                              "smoker": ["Yes", "No"],
                              "sex": ["Male", "Female"]})
fig.show()
```



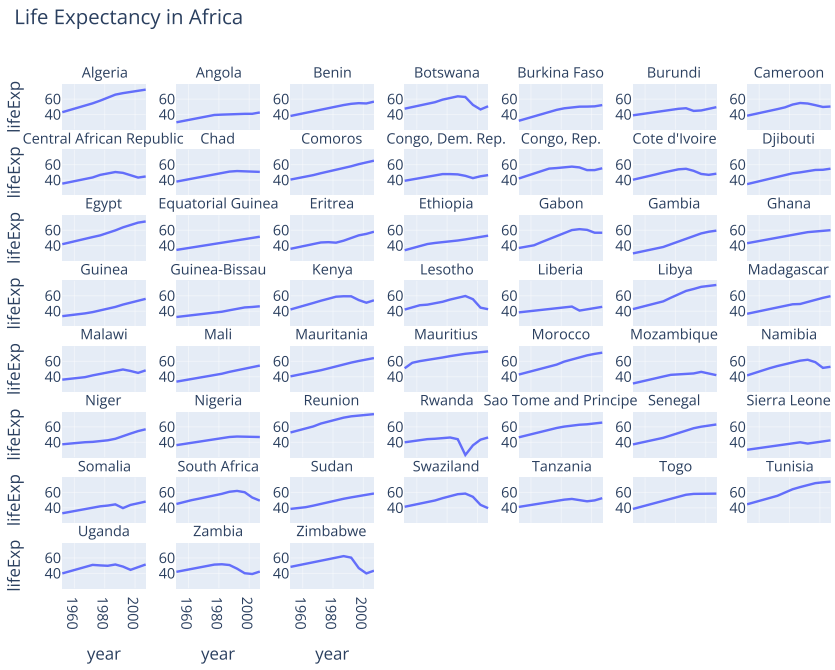
## Controlling Facet Spacing

The facet_row_spacing and facet_col_spacing arguments can be used to control the spacing between rows and columns. These values are specified in fractions of the plotting area in paper coordinates and not in pixels, so they will grow or shrink with the width and height of the figure.

The defaults work well with 1-4 rows or columns at the default figure size with the default font size, but need to be reduced to around 0.01 for very large figures or figures with many rows or columns. Conversely, if activating tick labels on all facets, the spacing will need to be increased.

ɛt Plots

```
import plotly.express as px

df = px.data.gapminder().query("continent == 'Africa'")

fig = px.line(df, x="year", y="lifeExp", facet_col="country", facet_col_wrap=7,
              facet_row_spacing=0.04, # default is 0.07 when facet_col_wrap is used
              facet_col_spacing=0.04, # default is 0.03
              height=600, width=800,
              title="Life Expectancy in Africa")
fig.for_each_annotation(lambda a: a.update(text=a.text.split("=")[-1]))
fig.update_yaxes(showticklabels=True)
fig.show()
```



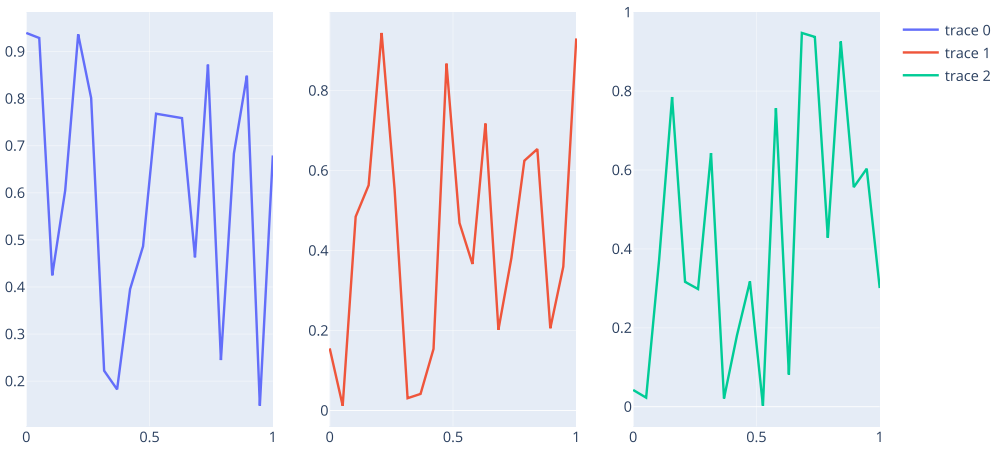

## Synchronizing axes in subplots with matches

Using facet_col from plotly.express let zoom (https://help.plotly.com/zoom-pan-hover-controls/#step-3-zoom-in-and-zoom-out-autoscale-the-plot) and pan (https://help.plotly.com/zoom-pan-hover-controls/#step-6-pan-along-axes) each facet to the same range implicitly. However, if the subplots are created with make_subplots, the axis needs to be updated with matches parameter to update all the subplots accordingly.

Zoom in one trace below, to see the other subplots zoomed to the same x-axis range. To pan all the subplots, click and drag from the center of x-axis to the side:

```
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np

N = 20
x = np.linspace(0, 1, N)

fig = make_subplots(1, 3)
for i in range(1, 4):
    fig.add_trace(go.Scatter(x=x, y=np.random.random(N)), 1, i)
fig.update_xaxes(matches='x')
fig.show()
```

et Plots




## 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
```