Python初級數據分析員證書

（五）進階Python數據分析及可視化技巧

12.Plotly套件

12.Plotly套件

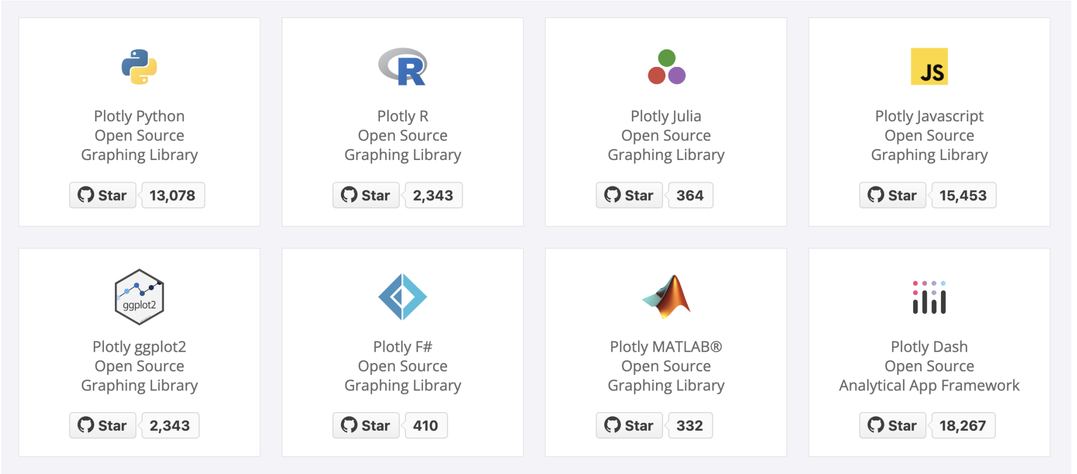
Chapter Summary

- Introduction
- Installation
- Getting start with bubble chart
- Interactive graphing
- Discrete color and continuous color
- Facet plots
- Plotly Express
- Matplotlib vs Plotly

Introduction

Plotly : Open Source Graphing Libraries

Plotly's Python graphing library makes interactive, publication-quality graphs. In this chapter we are focusing this module for data analysis. plotly | Graphing Libraries

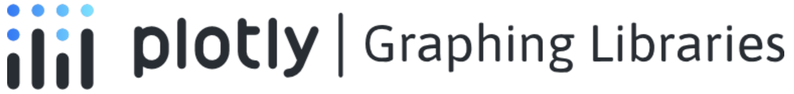


Plotly : Dash Open Source Dash Enterprise

Dash is the original low-code framework for rapidly building data apps in Python, R, Julia, and F# (experimental).

Written on top of Plotly.js and React.js, Dash is ideal for building and deploying data apps with customized user interfaces. It's particularly suited for anyone who works with data.

Installation



- **Current version 5.13.1**
- **Install using PIP**

```
pip install plotly
```

- **Import plotly.express**

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

Getting start

We are going to use some of the plotly DataFrame, such as iris for plot demo.

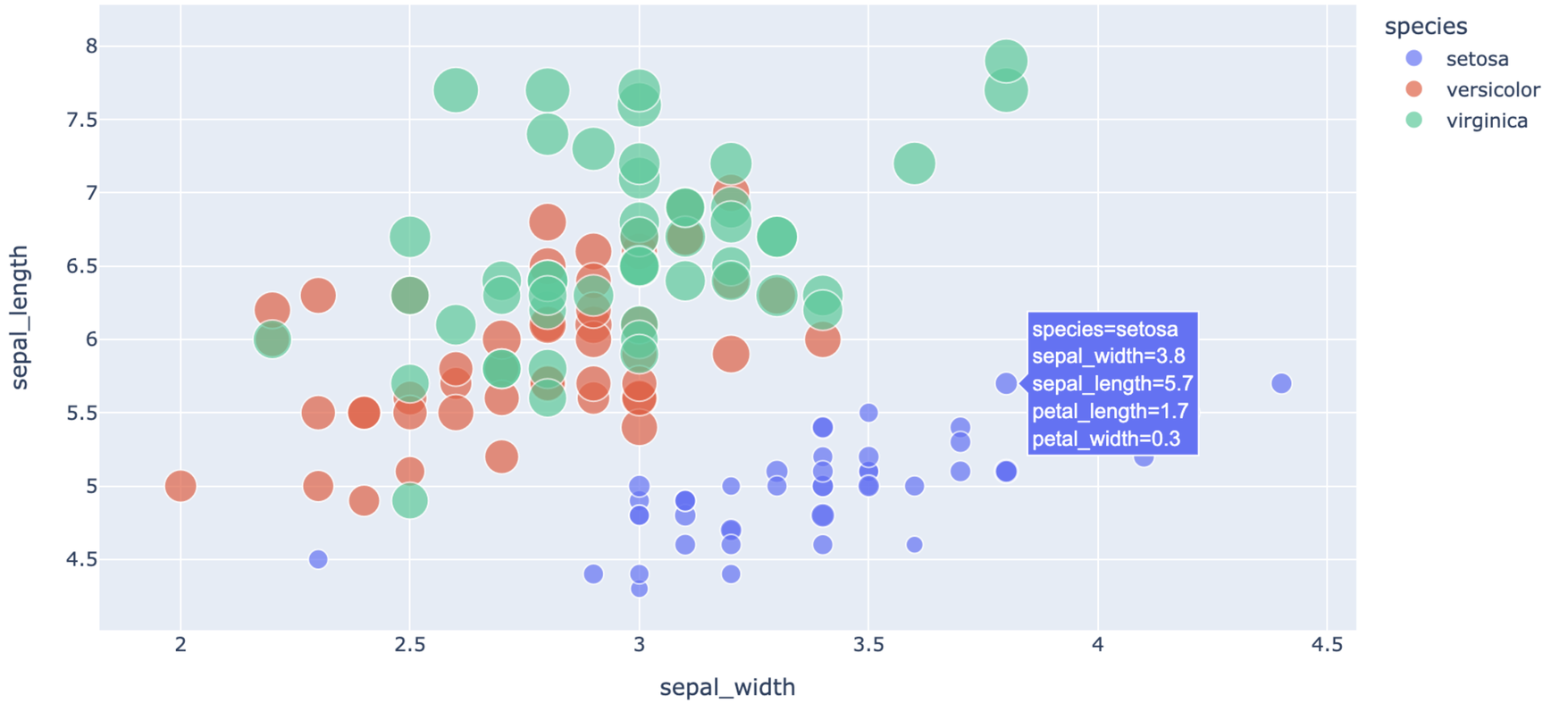
```
1 import plotly.express as px
2 df = px.data.iris()
3 df.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
47	4.6	3.2	1.4	0.2	setosa	1
26	5.0	3.4	1.6	0.4	setosa	1
143	6.8	3.2	5.9	2.3	virginica	3
49	5.0	3.3	1.4	0.2	setosa	1
67	5.8	2.7	4.1	1.0	versicolor	2

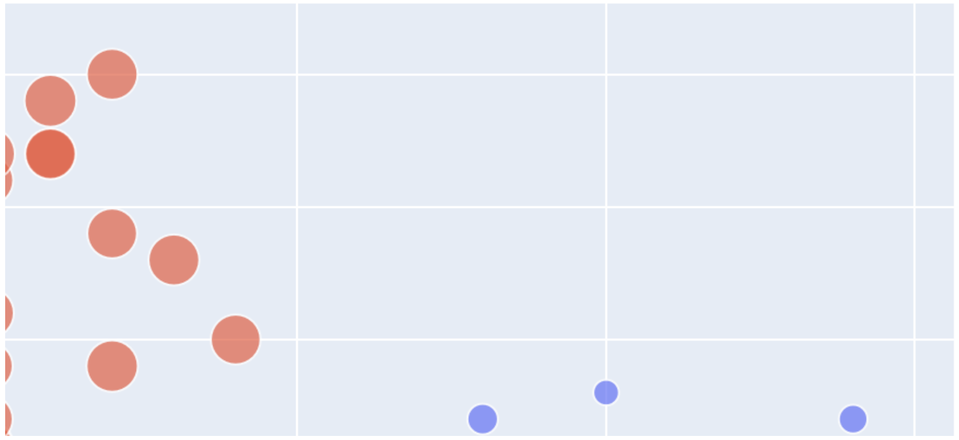
px.scatter – bubble chart

```
1  fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species",
2                   size='petal_length', hover_data=['petal_width'])
3  fig.show()
```

Move your pointer along the plot to see what interactive means in plotly.

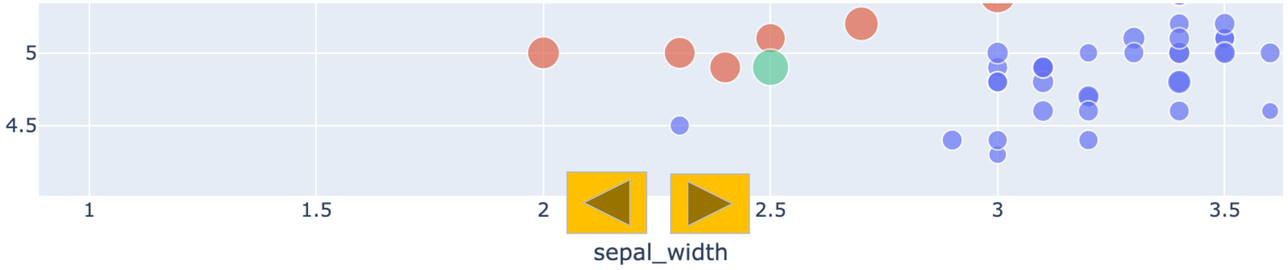


Interactive graphing



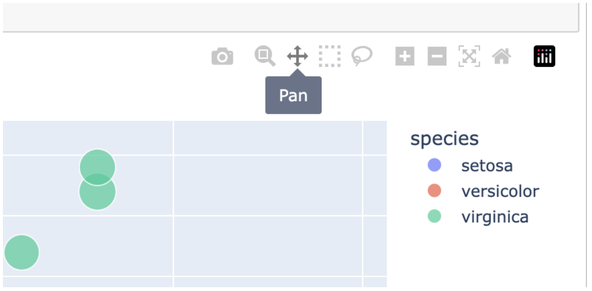
You can toggle the species to isolate one or more kind.

Discover the interactive graph



Drag the x-axis left or right

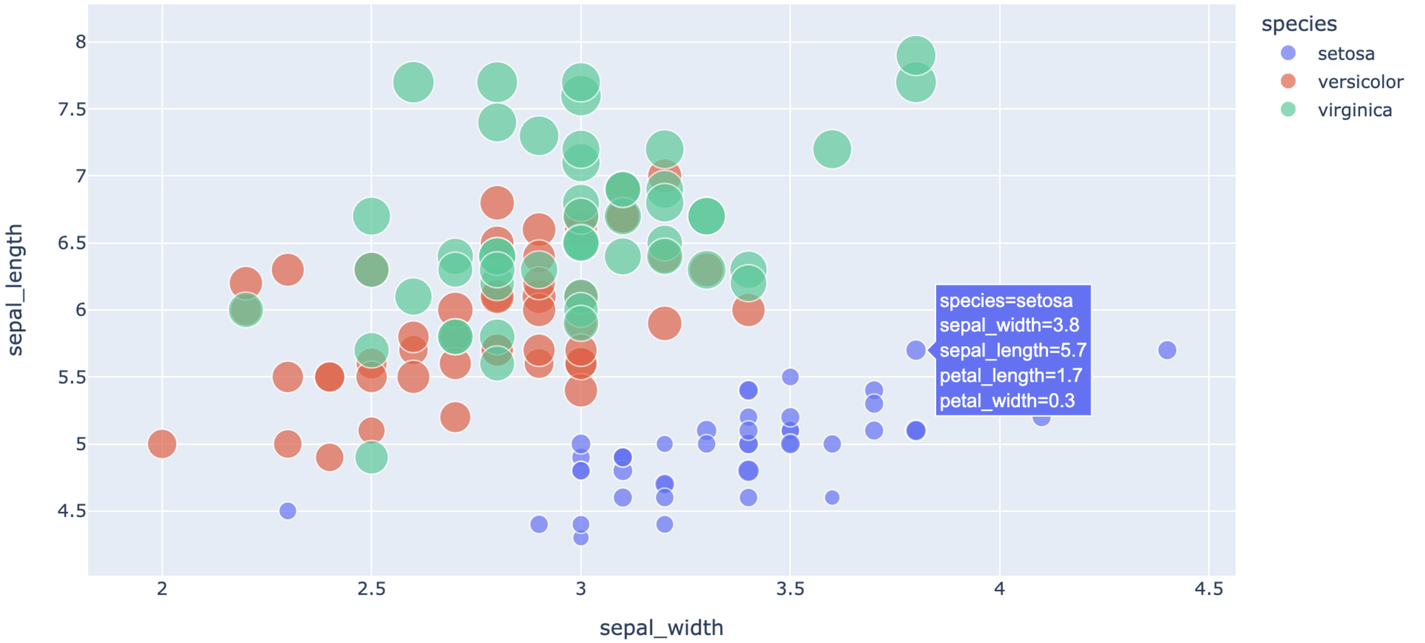
Use the Pan to drag the graph



Setting size and colour with column names

```
fig = px.scatter(df, x="sepal_width", y="sepal_length", color="species",
                 size='petal_length', hover_data=['petal_width'])
fig.show()
```

Scatter plots with variable-sized circular markers are often known as bubble charts. Note that colour and size data are added to hover information. You can add other columns to hover data with the hover_data argument of px.scatter.



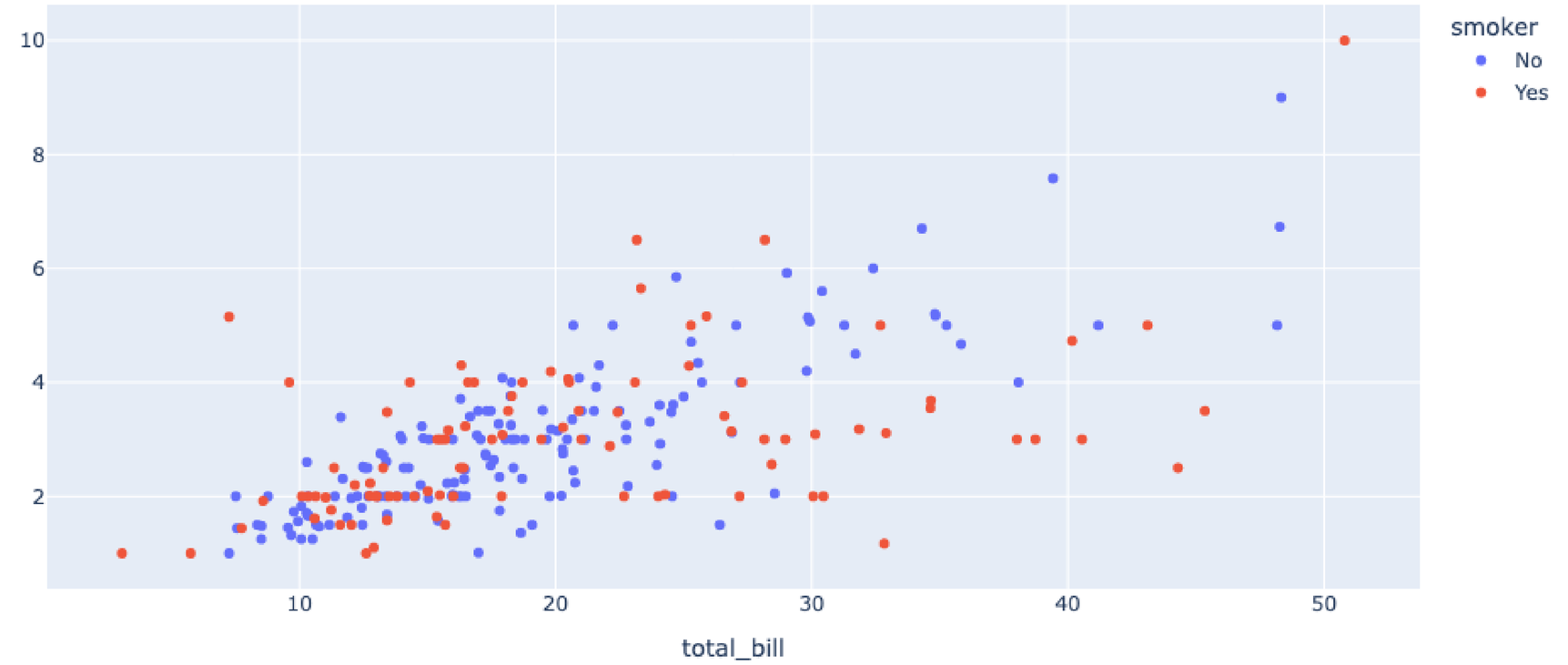
Discrete Color and Continuous Color

```
1  tips = px.data.tips()
2  tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    object
 3   smoker      244 non-null    object
 4   day         244 non-null    object
 5   time        244 non-null    object
 6   size        244 non-null    int64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.5+ KB
```

```
1  df = px.data.tips()
2  fig = px.scatter(df, x="total_bill", y="tip", color="smoker",
3                   title="String 'smoker' values mean discrete colors")
4  fig.show()
```

String 'smoker' values mean discrete colors



Plotly assigns data values to discrete colors if the data is non-numeric.

Discrete Color and Continuous Color

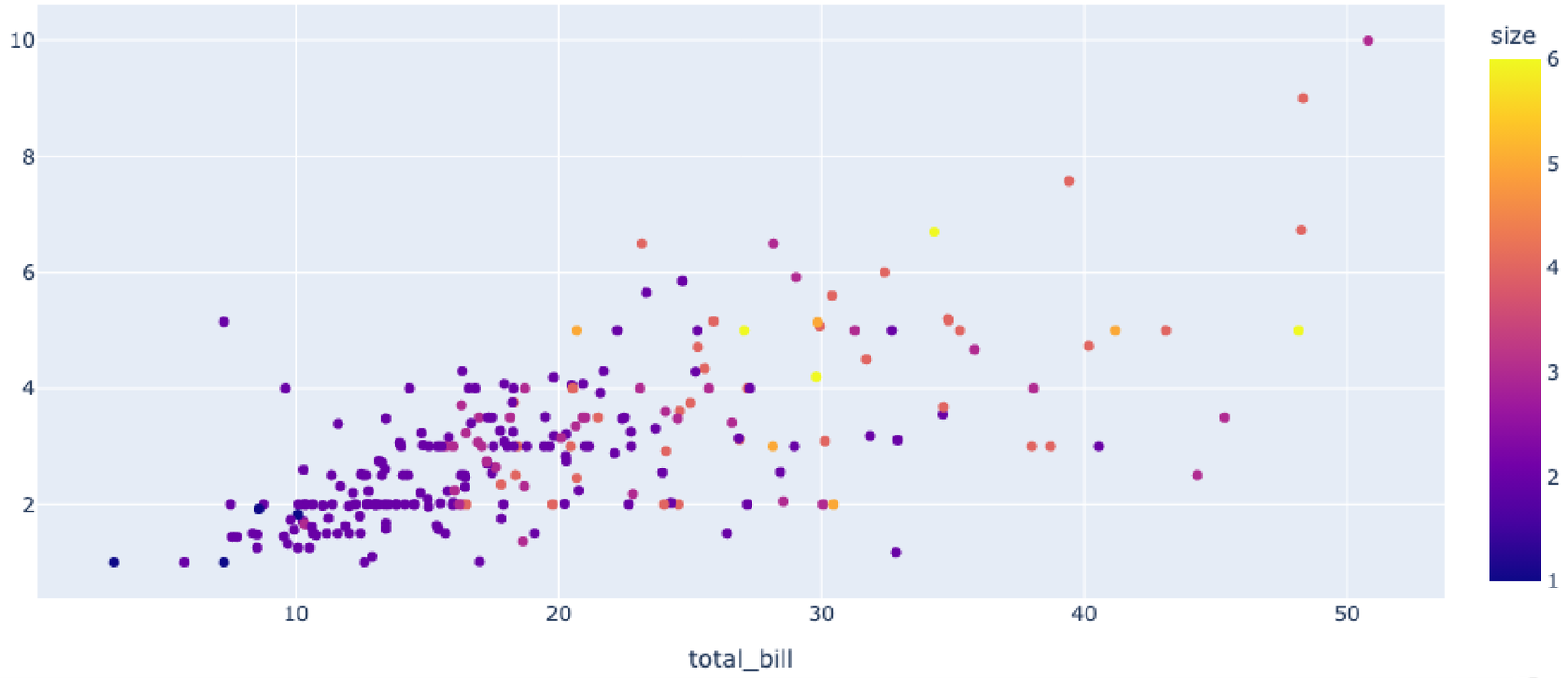
```
1 tips = px.data.tips()
2 tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    object
 3   smoker      244 non-null    object
 4   day         244 non-null    object
 5   time        244 non-null    object
 6   size        244 non-null    int64
dtypes: float64(2), int64(1), object(4)
memory usage: 13.5+ KB
```

If the data is numeric, the color will automatically be considered continuous.

```
1 fig = px.scatter(df, x="total_bill", y="tip", color="size",
2                   title="Numeric 'size' values mean continuous color")
3 fig.show()
```

Numeric 'size' values mean continuous color

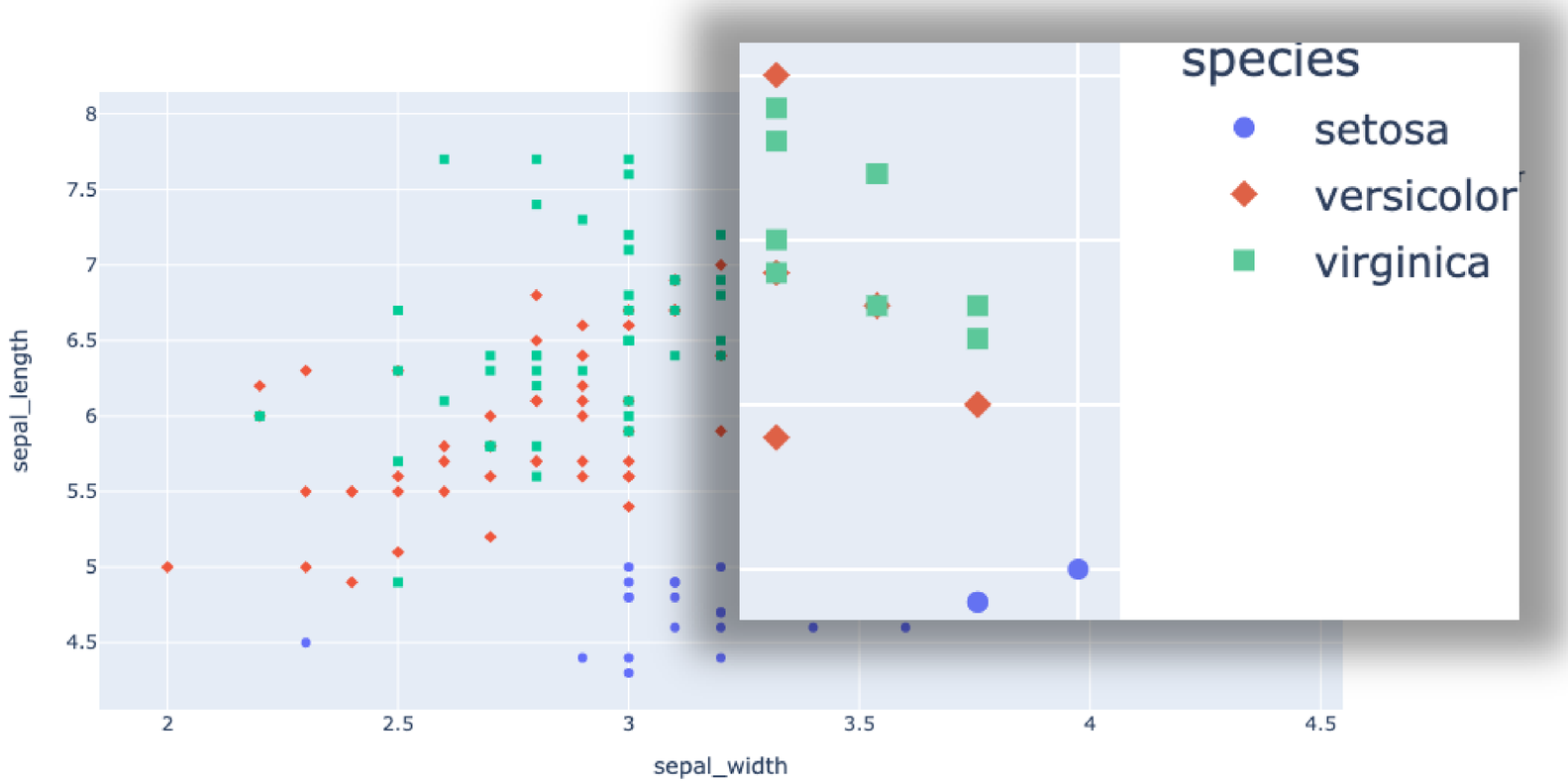


Symbol argument

The **symbol** argument can be mapped to a column and to change the shape.

The symbol can be customized as well.

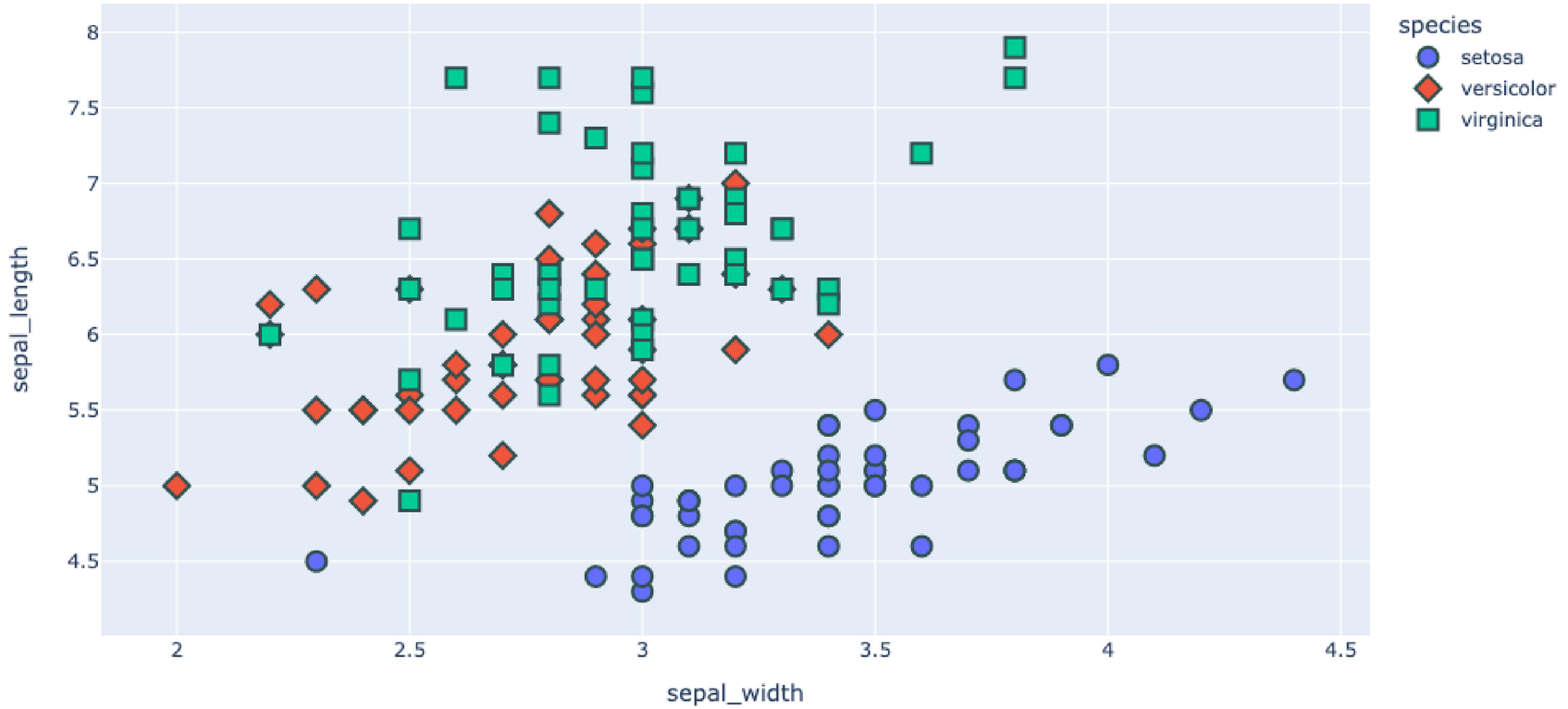
```
1  df = px.data.iris()
2  fig = px.scatter(df, x="sepal_width", y="sepal_length",
3                   color="species", symbol="species")
4  fig.show()
```



Customized symbol

```
1  fig = px.scatter(df, x="sepal_width", y="sepal_length",
2                   color="species", symbol="species").update_traces(
3                  marker=dict(size=12, line=dict(width=2,
4                  color='DarkSlateGrey')), selector=dict(mode='markers'))
5  fig.show()
```

Customized the symbol with .update_trace( )

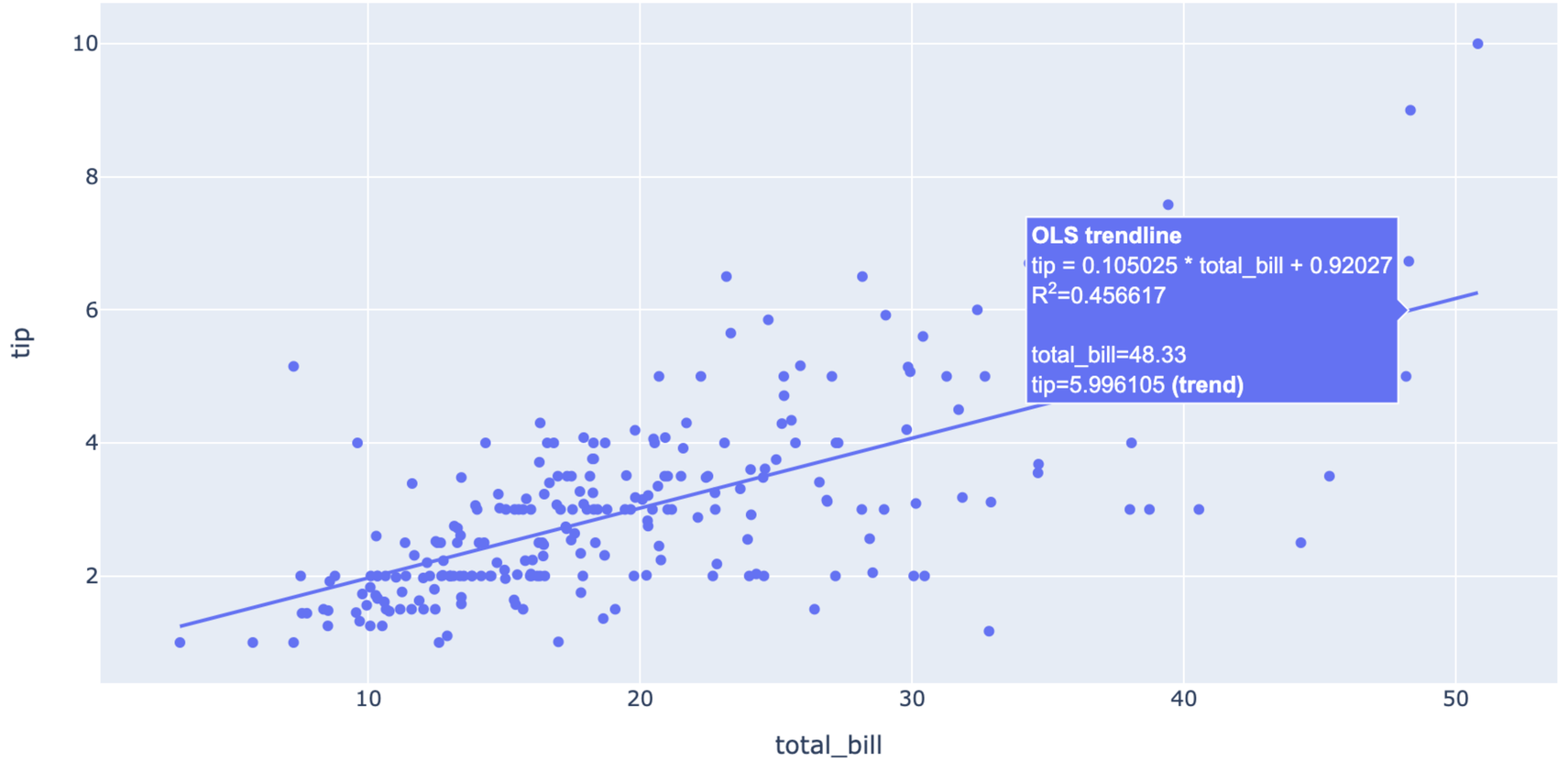


Linear Regression Line

```
1  fig = px.scatter(tips, x="total_bill", y="tip", trendline="ols")
2  fig.show()
```

Hover on the line,

you can see the line function and R^2



Linear Regression Line

```
1 df_gold.sample(3)
```

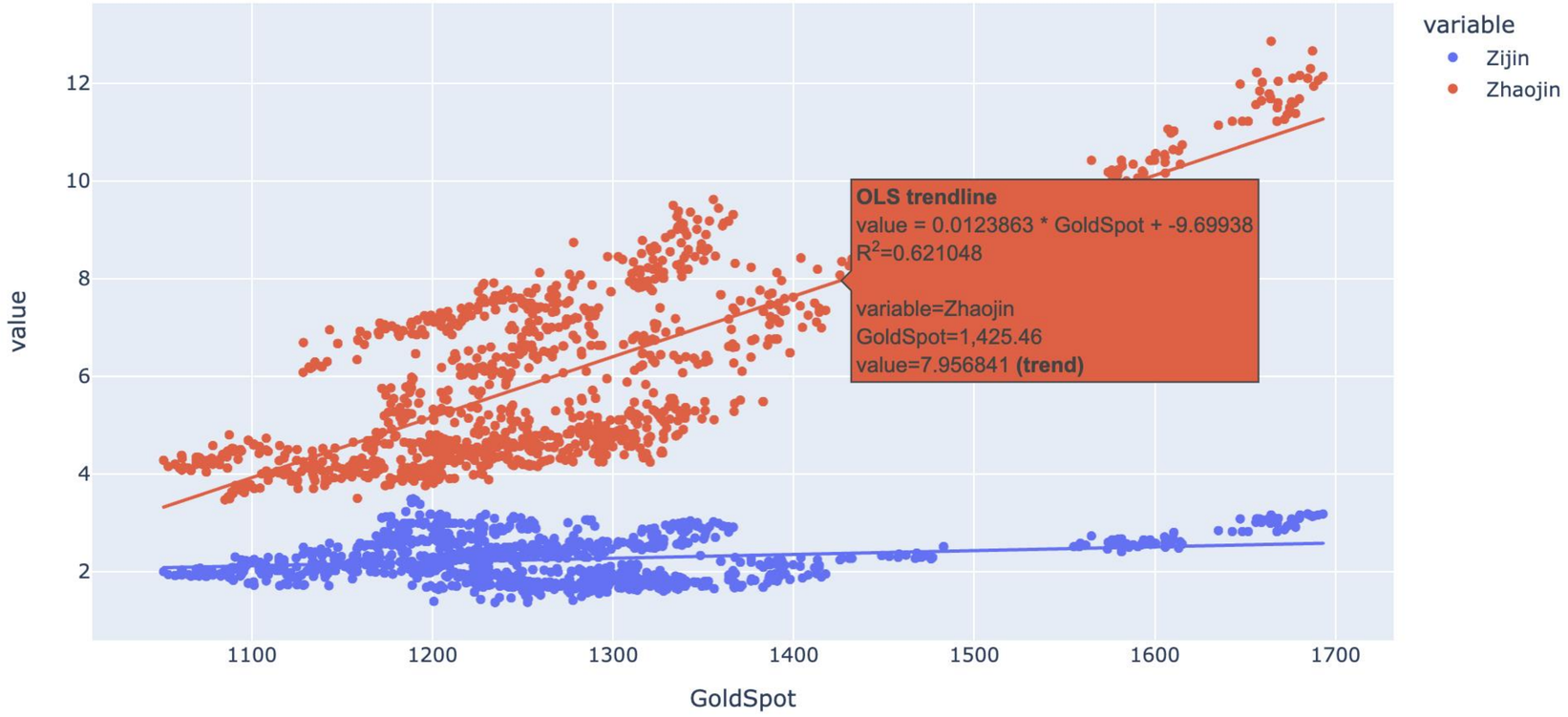
Date	GoldSpot	Zijin	Zhaojin
2013-09-05	1367.48	1.96	7.34
2014-01-20	1254.35	1.70	4.66
2014-07-09	1327.83	1.77	4.72

Plot two regression line for Zijin-vs-Gold and Zhaojin-vs-Gold.

You may see the R^2 of each line.

For better presentation, you may plot the log return of the assets instead of prices.

```
1 fig = px.scatter(df_gold, x="GoldSpot",
2         y=["Zijin","Zhaojin"],  trendline="ols" )
3 fig.show()
```



Facet plots – facet_col

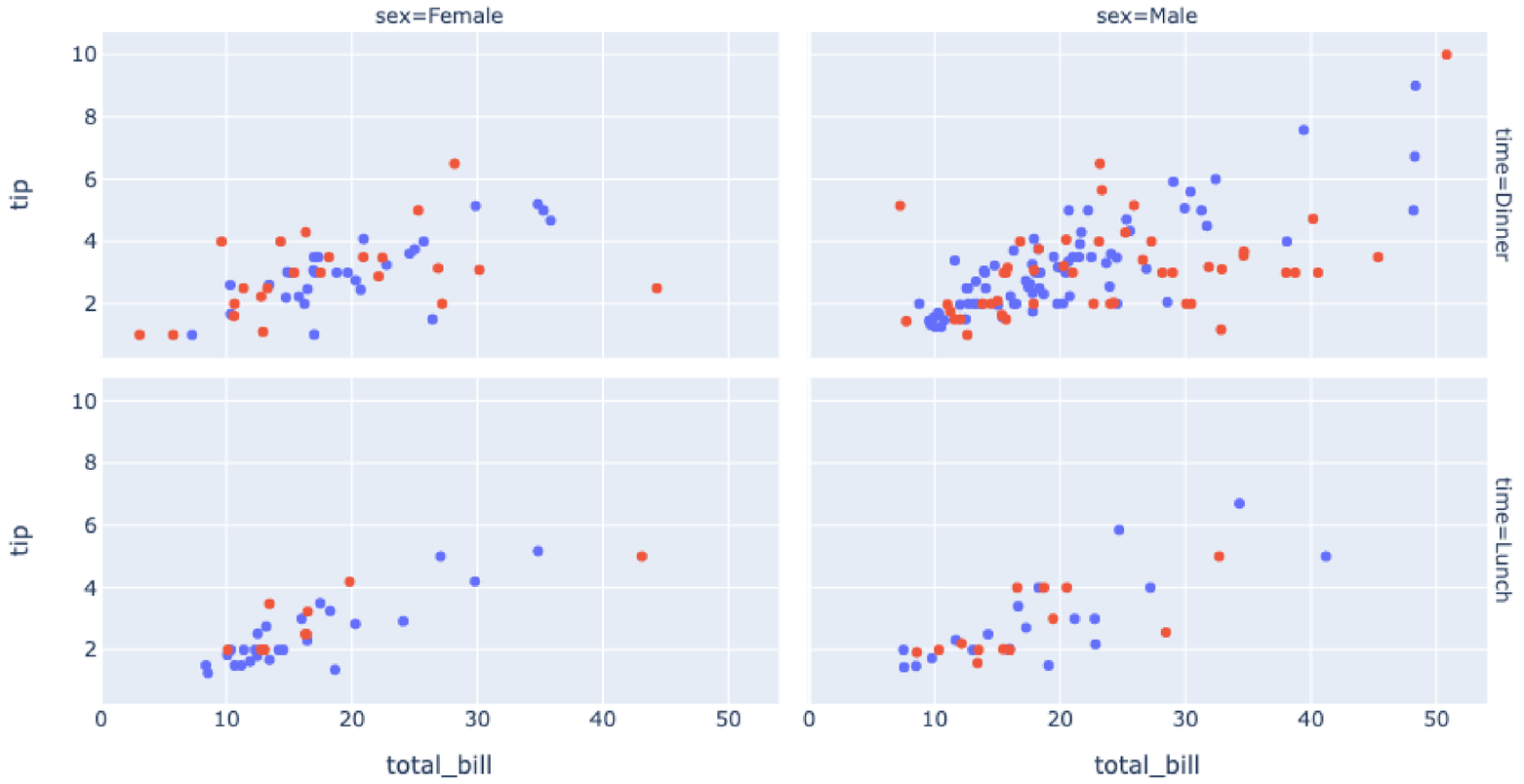
Facet plots are figures made up of multiple subplots which have the same set of axes, where each subplot shows a subset of the data.

facet_col is for distinguish the value among the other ones.

```
1  fig = px.scatter(tips, x="total_bill", y="tip",
2                   color="smoker", facet_col="sex")
3  fig.show()
```

Facet plots – facet_row

```
1  fig = px.scatter(tips, x="total_bill", y="tip",
2      color="smoker", facet_col="sex", facet_row="time")
3  fig.show()
```



facet_row is for the purpose of showing the difference on rows bases.

facet_row here shows the difference of lunch and dinner.

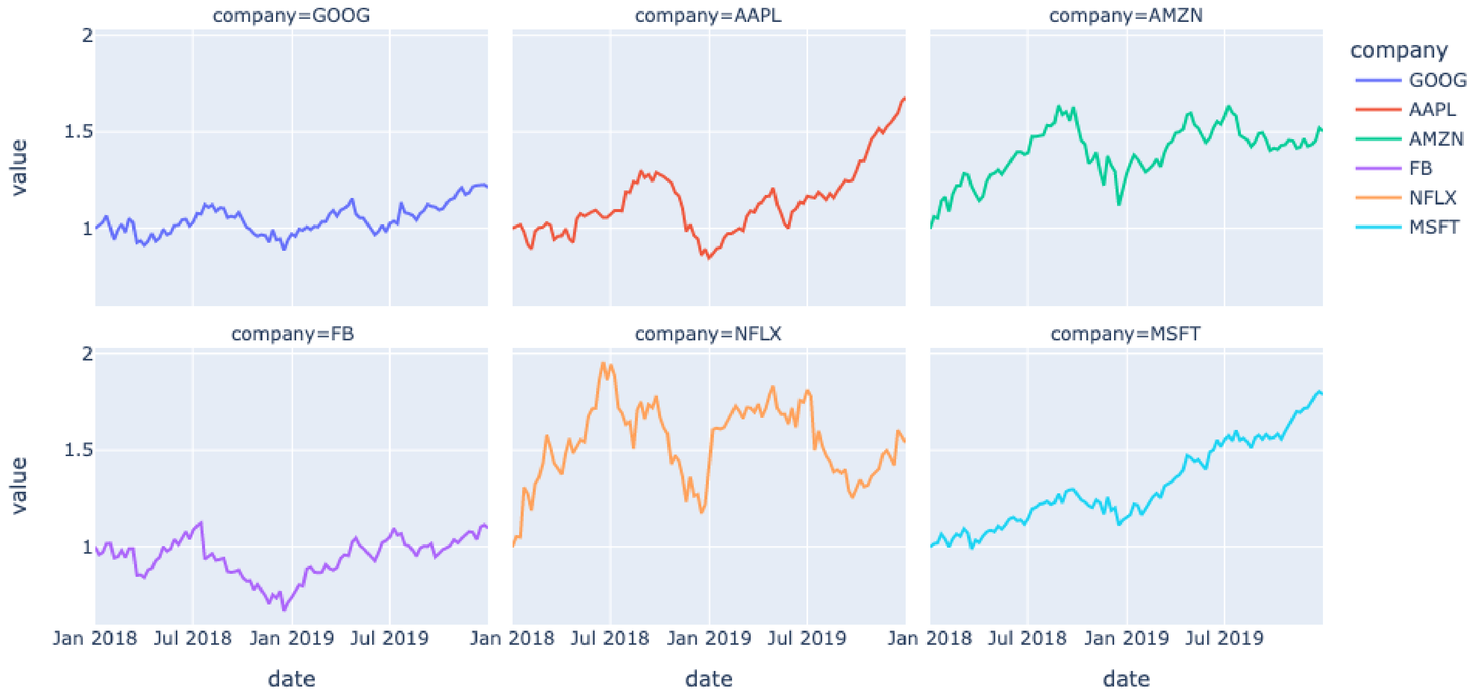
Facet plot – facet_col_wrap

```
df_stock = px.data.stocks(indexed=True)
df_stock.head(3)
```

company	GOOG	AAPL	AMZN	FB	NFLX	MSFT
date						
2018-01-01	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
2018-01-08	1.018172	1.011943	1.061881	0.959968	1.053526	1.015988
2018-01-15	1.032008	1.019771	1.053240	0.970243	1.049860	1.020524

facet_col_wrap is the column per row argument. This can be applied in other type of plot.

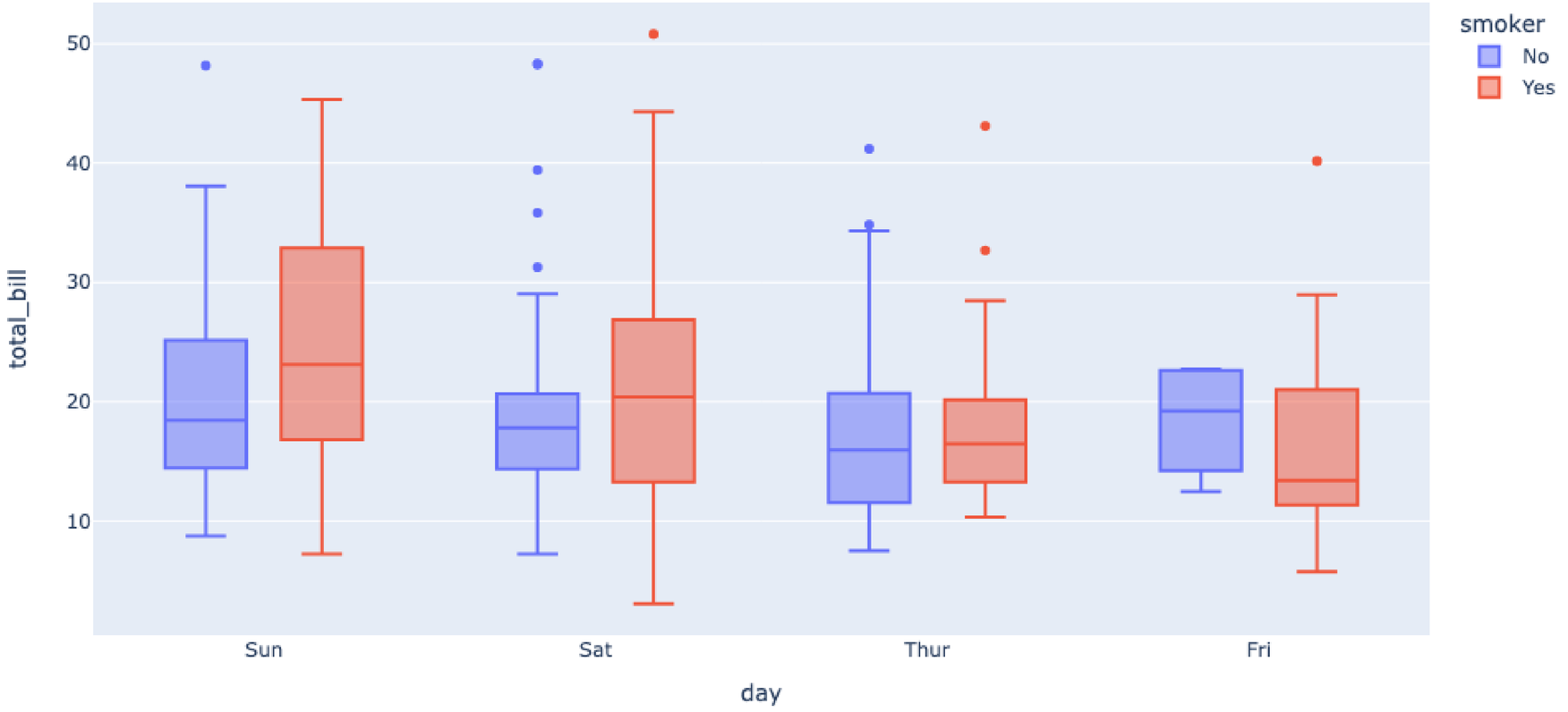
```
fig = px.line(df_stock, facet_col="company", facet_col_wrap=3)
fig.show()
```



px.box – Box plot

```
1  fig = px.box(tips, x="day", y="total_bill", color="smoker")
2  fig.update_traces(quartilemethod="exclusive") # or "inclusive", or "linear" by default
3  fig.show()
```

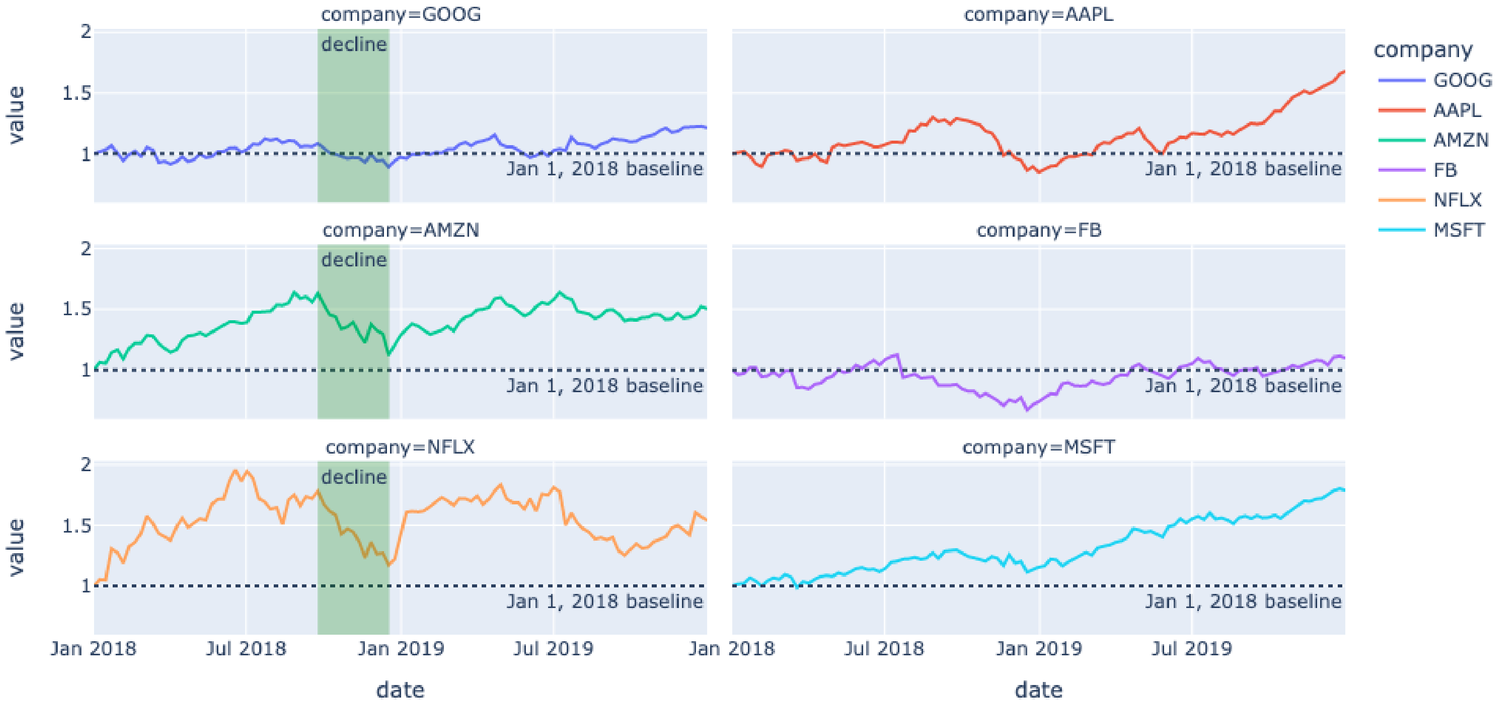
The exclusive algorithm uses the median to divide the ordered dataset into two halves. If the sample is odd, it does not include the median in either half. Q1 is then the median of the lower half and Q3 is the median of the upper half.



Adding lines and rectangles to facet plots

```
1  fig = px.line(df_stock, facet_col="company", facet_col_wrap=2)
2  fig.add_hline(y=1, line_dash="dot",
3                annotation_text="Jan 1, 2018 baseline",
4                annotation_position="bottom right")
5  fig.add_vrect(x0="2018-09-24", x1="2018-12-18", col=1,
6                annotation_text="decline", annotation_position="top left",
7                fillcolor="green", opacity=0.25, line_width=0)
8  fig.show()
```

It is possible to add labelled horizontal and vertical lines and rectangles 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.

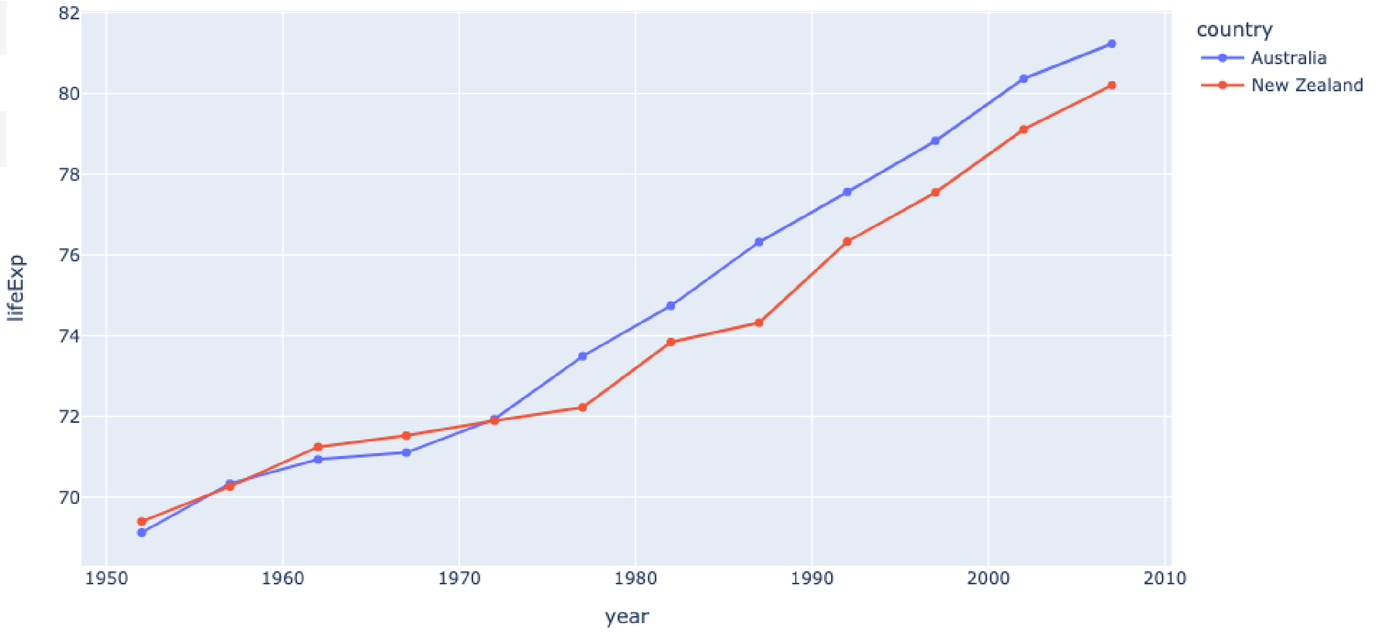


Line plots

```
1  df_life = px.data.gapminder().query("continent == 'Oceania'")
2  fig = px.line(df_life, x='year', y='lifeExp', color='country', markers=True)
3  fig.show()
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
65	Australia	Oceania	1977	73.49	14074100	18334.19751	AUS	36
64	Australia	Oceania	1972	71.93	13177000	16788.62948	AUS	36
60	Australia	Oceania	1952	69.12	8691212	10039.59564	AUS	36

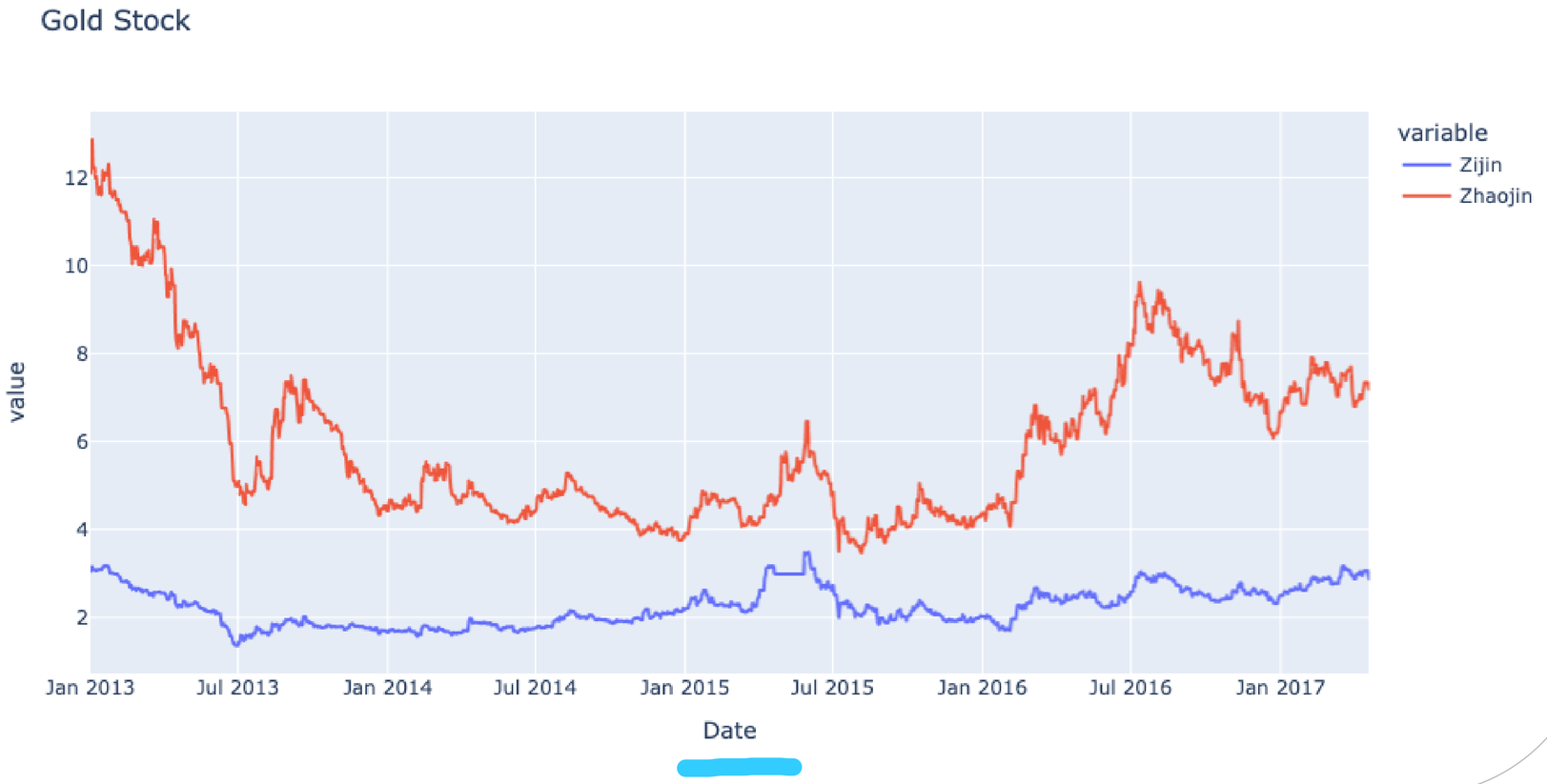
Line plot is comparably simple and straight forward.



Line plots with Datetime index

```
1  df_gold = pd.read_csv("gold_goldstock.csv")
2  df_gold["Date"] = pd.to_datetime(df_gold['Date'], dayfirst=True)
3  df_gold = df_gold.set_index('Date')
4
5  fig = px.line(df_gold, x=df_gold.index, y=["Zijin","Zhaojin"], title="Gold Stock")
6  fig.show()
```

Since plotly recognize DF in Pandas format, the index would not be Date if been set as index. Same idea in Numpy or other library.



Large range variables

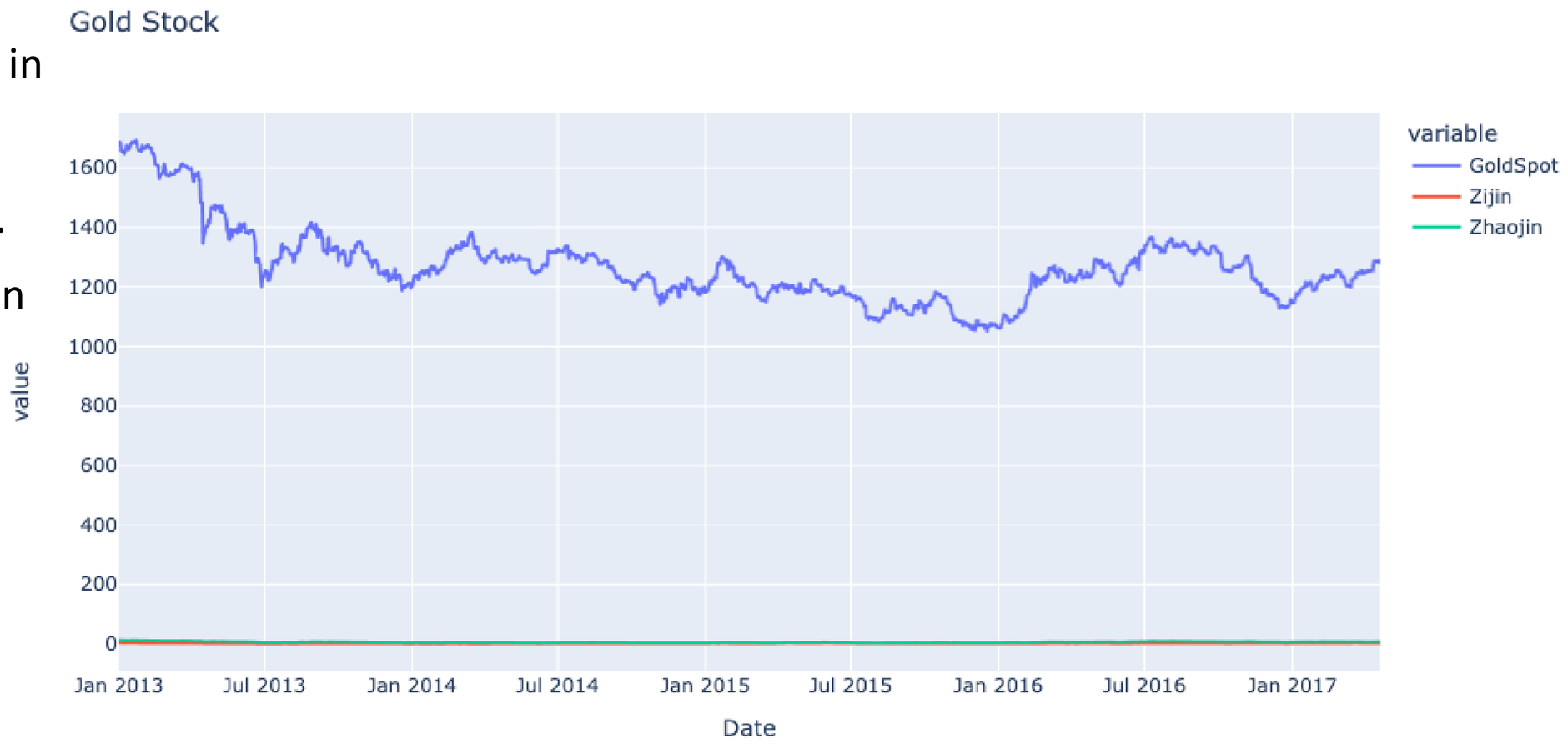
In some cases, the value in some variable might be much larger than others.

It is difficult to visualize in scale.

GoldSpot: $16xx.xx

Zijin: $3.xx

Zhaojin: $13.xx

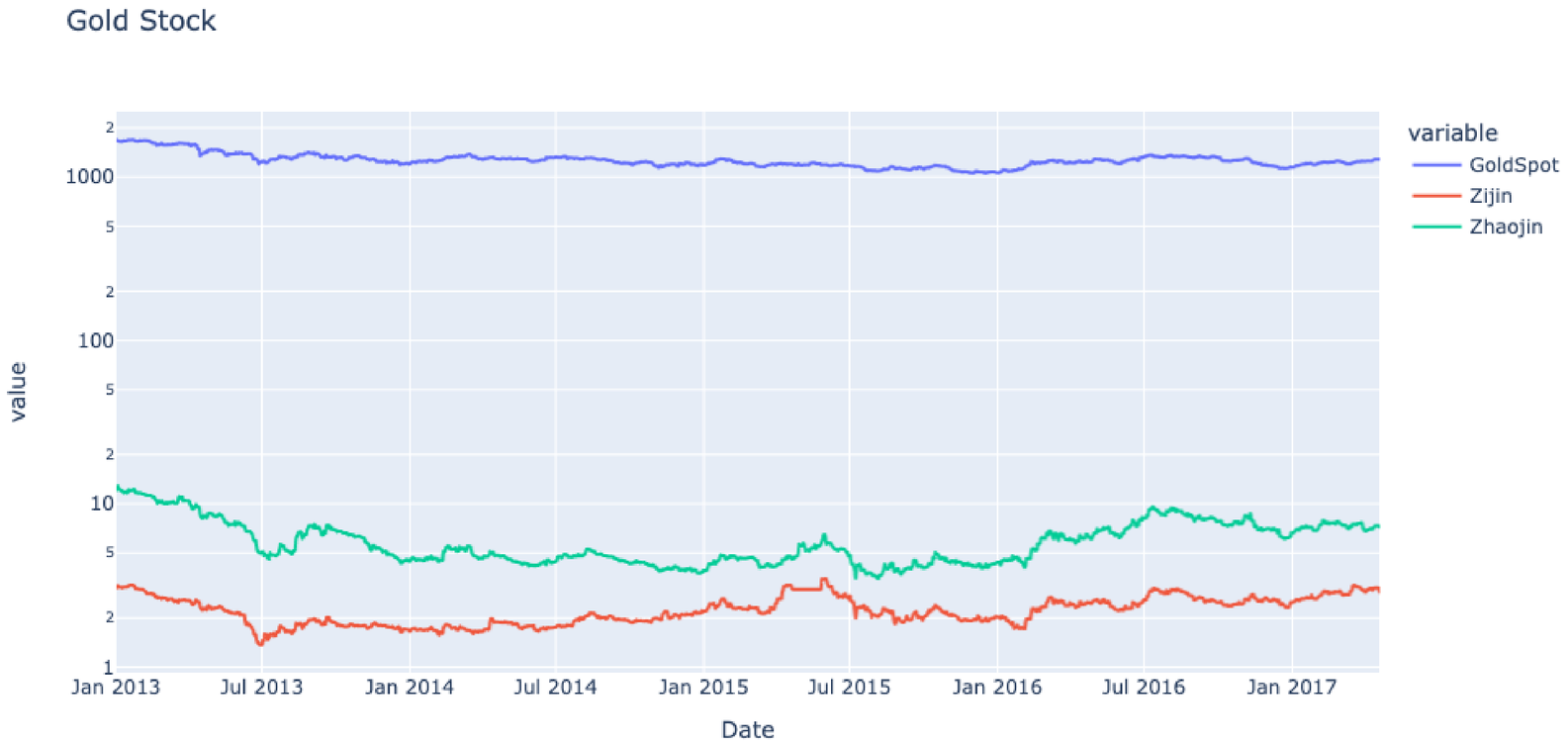


Enable log scale

```
fig = px.line(df_gold, x=df_gold.index, y=["GoldSpot","Zijin","Zhaojin"],
              log_y=True, title="Gold Stock")
fig.show()
```

Set log_y=True to adjust the value in $\log_{10}$ scale.

Same idea could apply on x-axis (then use log_x=True).

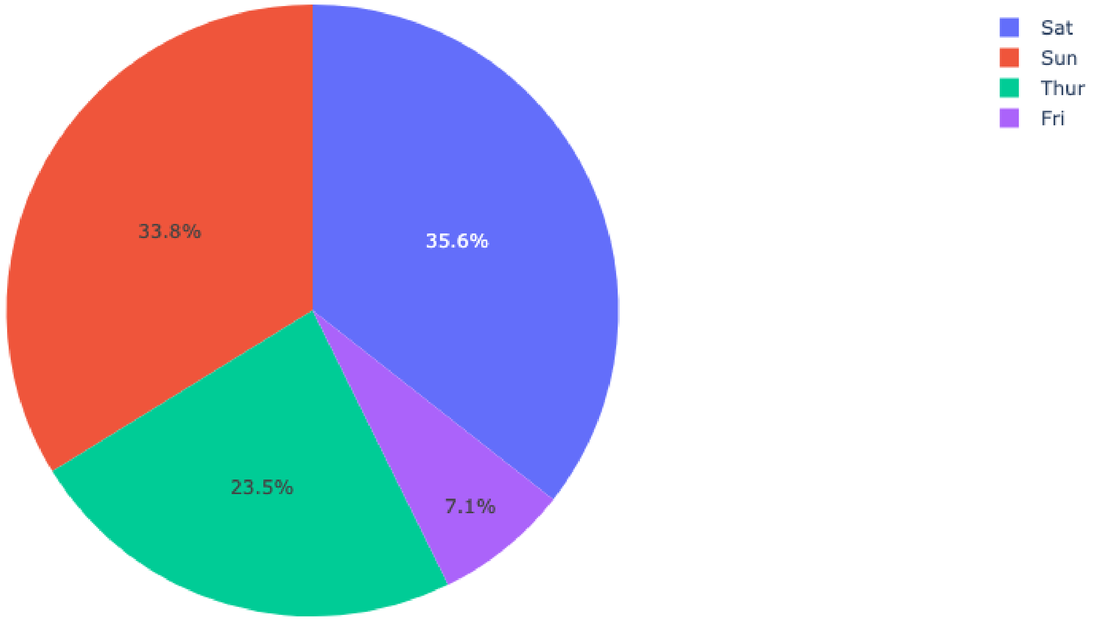


px.pie – Pie Chart

```
1  fig = px.pie(tips, values='tip', names='day')
2  fig.show()
```

Values: the numeric data

Names: the category data

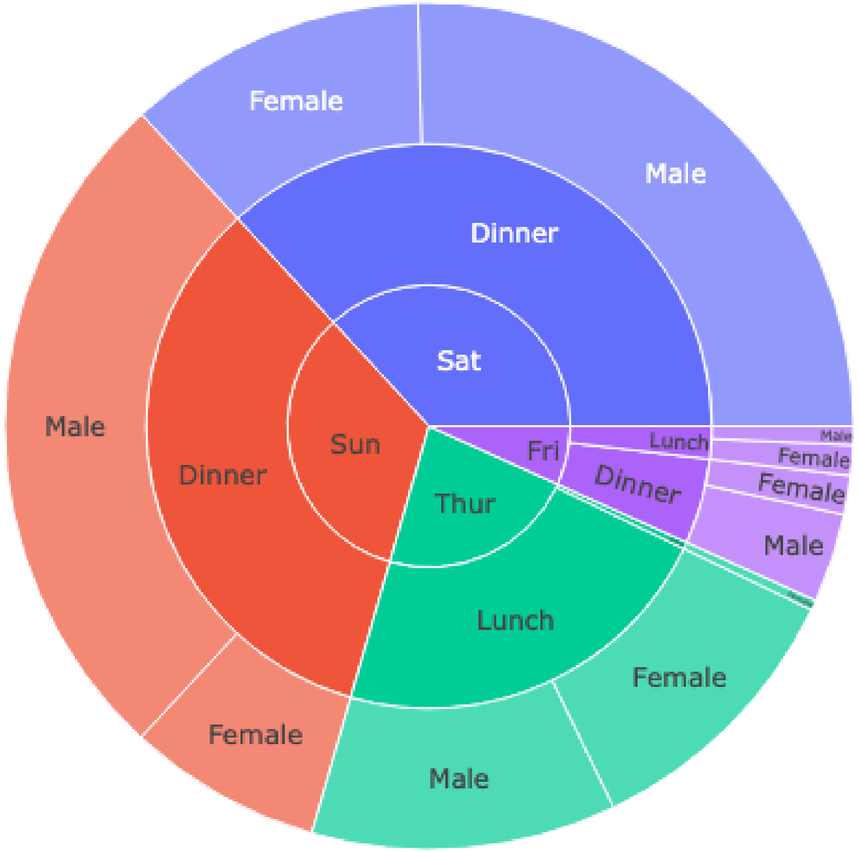


px.sunburst – Sunburst of a rectangular DF

```
1 fig = px.sunburst(tips, path=['day', 'time', 'sex'], values='total_bill')
2 fig.show()
```

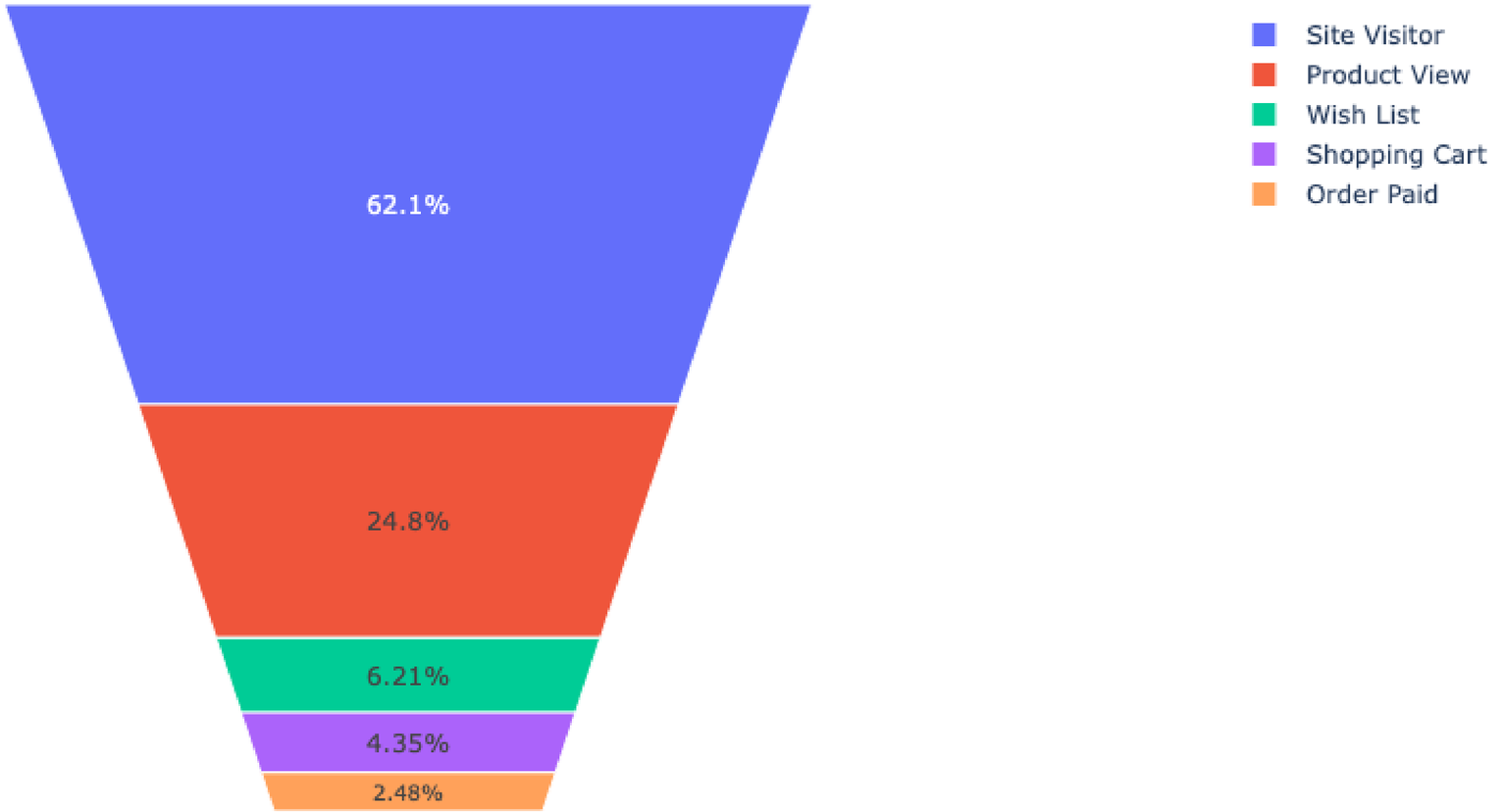
Sunburst is advance pie chart for hierarchy sub-categories.

Path defines the level of each hierarchy.



px.funnel_area – Funnel plot

```
1  fig = px.funnel_area(names=["Site Visitor","Product View",
2                              "Wish List", "Shopping Cart", "Order Paid"],
3                              values=[500, 200, 50, 35, 20])
4  fig.show()
```



With px.funnel_area, each row of the DataFrame is represented as a stage of the funnel.

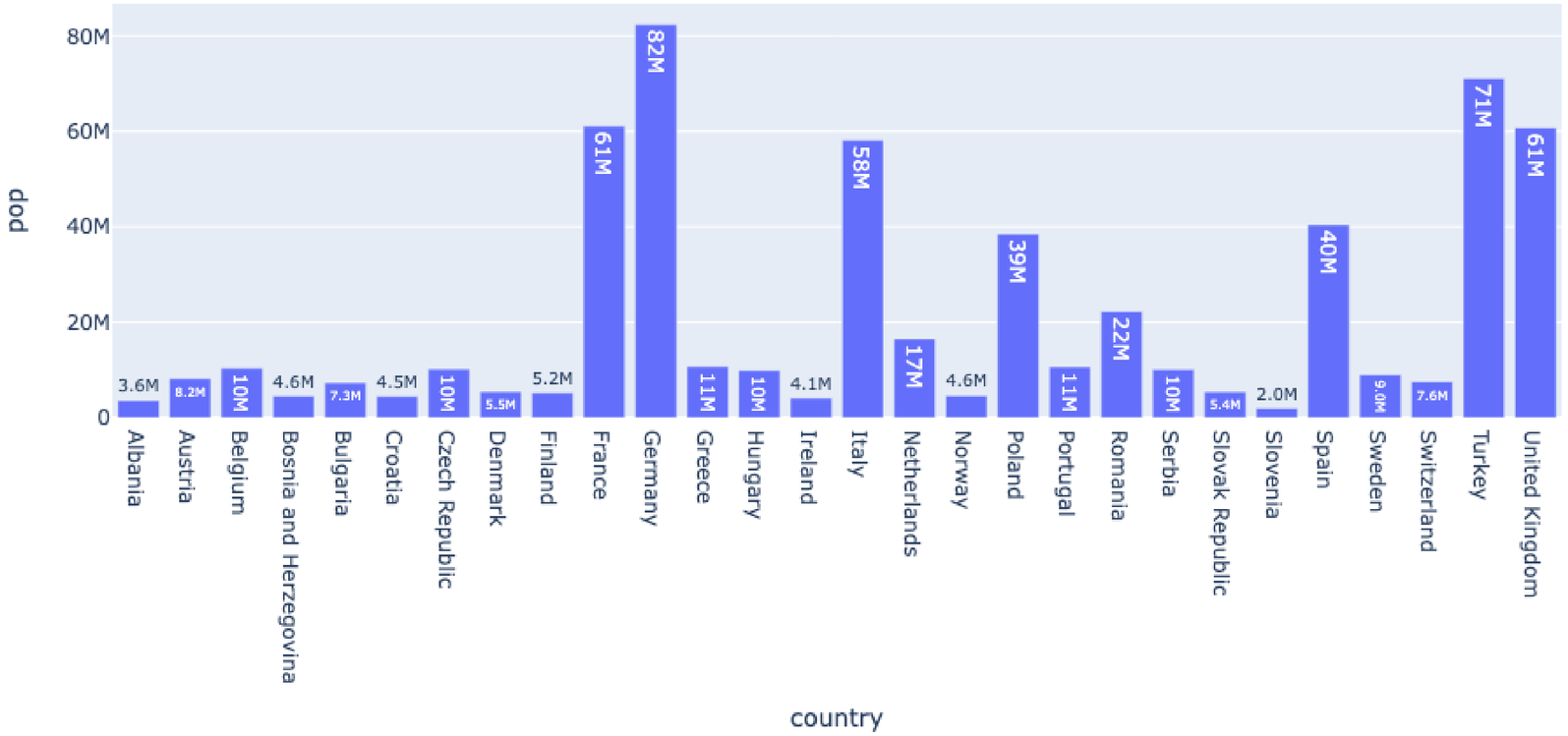
px.bar – Bar Chart

```
1  df_pop = px.data.gapminder().query("continent == 'Europe' and year == 2007 and pop > 2.e6")
2  fig = px.bar(df_pop, y='pop', x='country', text_auto='.2s',
3               title="Default: various text sizes, positions and angles")
4  fig.show()
```

Default: various text sizes, positions and angles

By default, Plotly will scale and rotate text labels to maximize the number of visible labels, which can result in a variety of text angles and sizes and positions in the same figure.

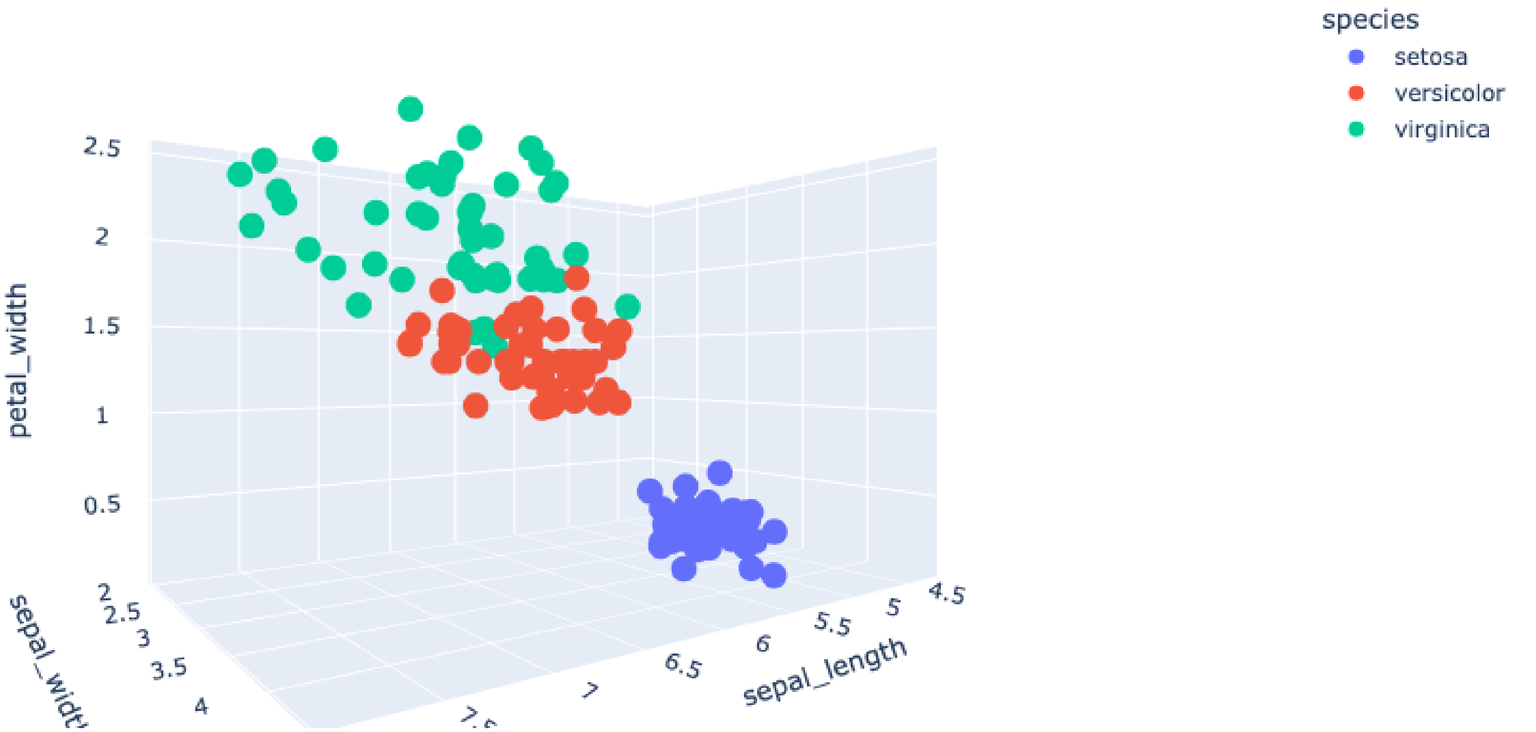
The textfont, textposition and tex tangle trace attributes can be used to control these.



3D scatter plot

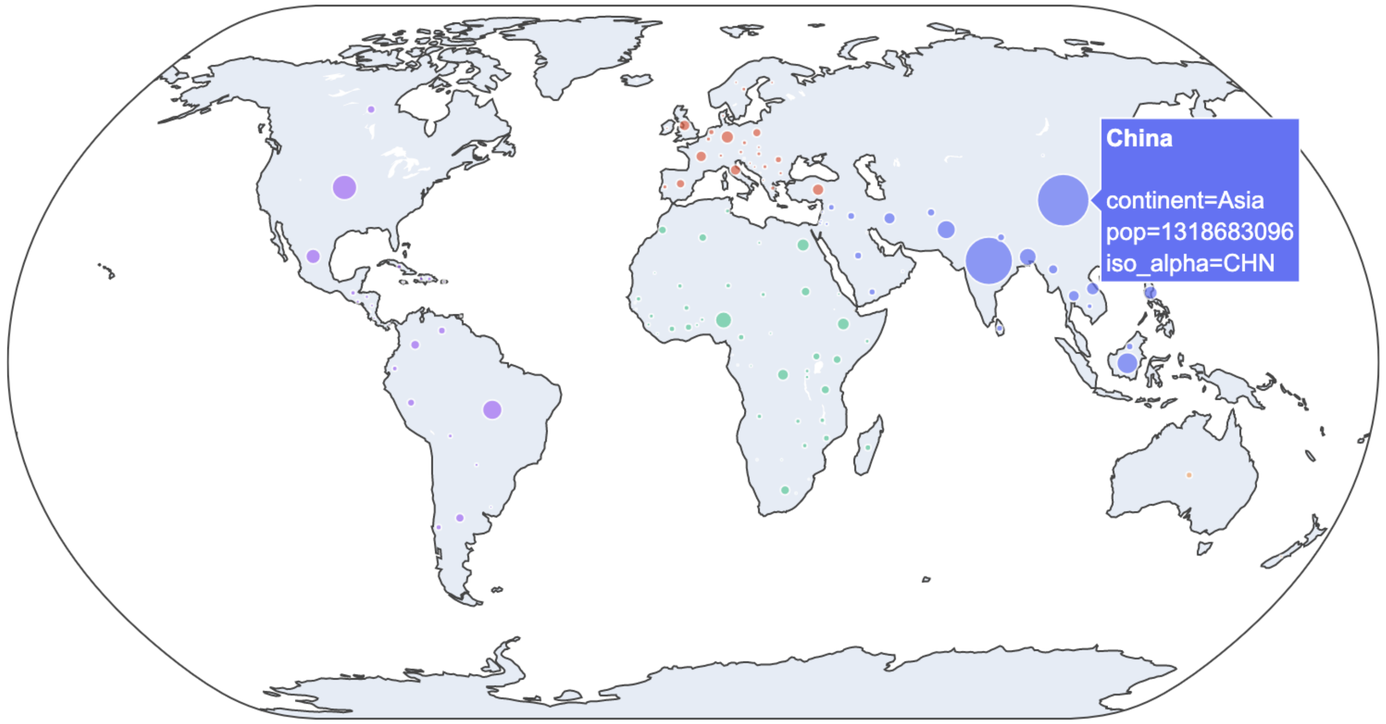
4 lines of code to create a 3D interactive graph with .px

```
df_iris = px.data.iris()
fig = px.scatter_3d(df_iris, x='sepal_length',
        y='sepal_width', z='petal_width', color='species')
fig.show()
```



px.scatter_geo – Geographical Scatter Plot

```
df_geo = px.data.gapminder().query("year == 2007")
fig = px.scatter_geo(df_geo, locations="iso_alpha",
                     color="continent", # which column to use to set the color of markers
                     hover_name="country", # column added to hover information
                     size="pop", # size of markers
                     projection="natural earth")
fig.show()
```

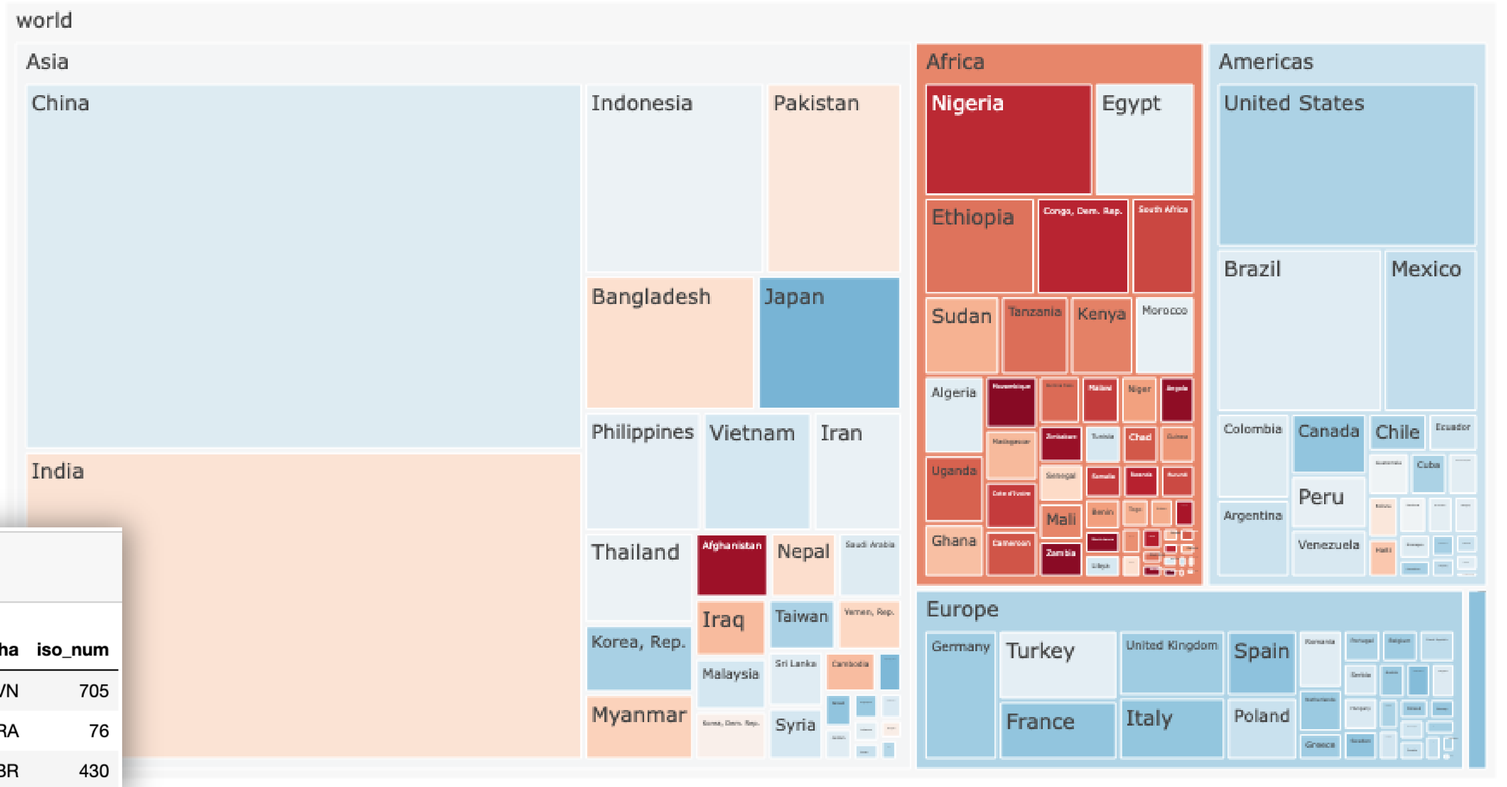


Customize geographical scatter plot

px.treemap

```python
import numpy as np
fig = px.treemap(df_geo, path=[px.Constant("world"), 'continent', 'country'], values='pop',
                 color='lifeExp', hover_data=['iso_alpha'],
                 color_continuous_scale='RdBu',
                 color_continuous_midpoint=np.average(df_geo['lifeExp'], weights=df_geo['pop']))
fig.update_layout(margin = dict(t=50, l=25, r=25, b=25))
fig.show()
```

If a color argument is passed, the color of a node is computed as the average of the color values of its children, weighted by their values.



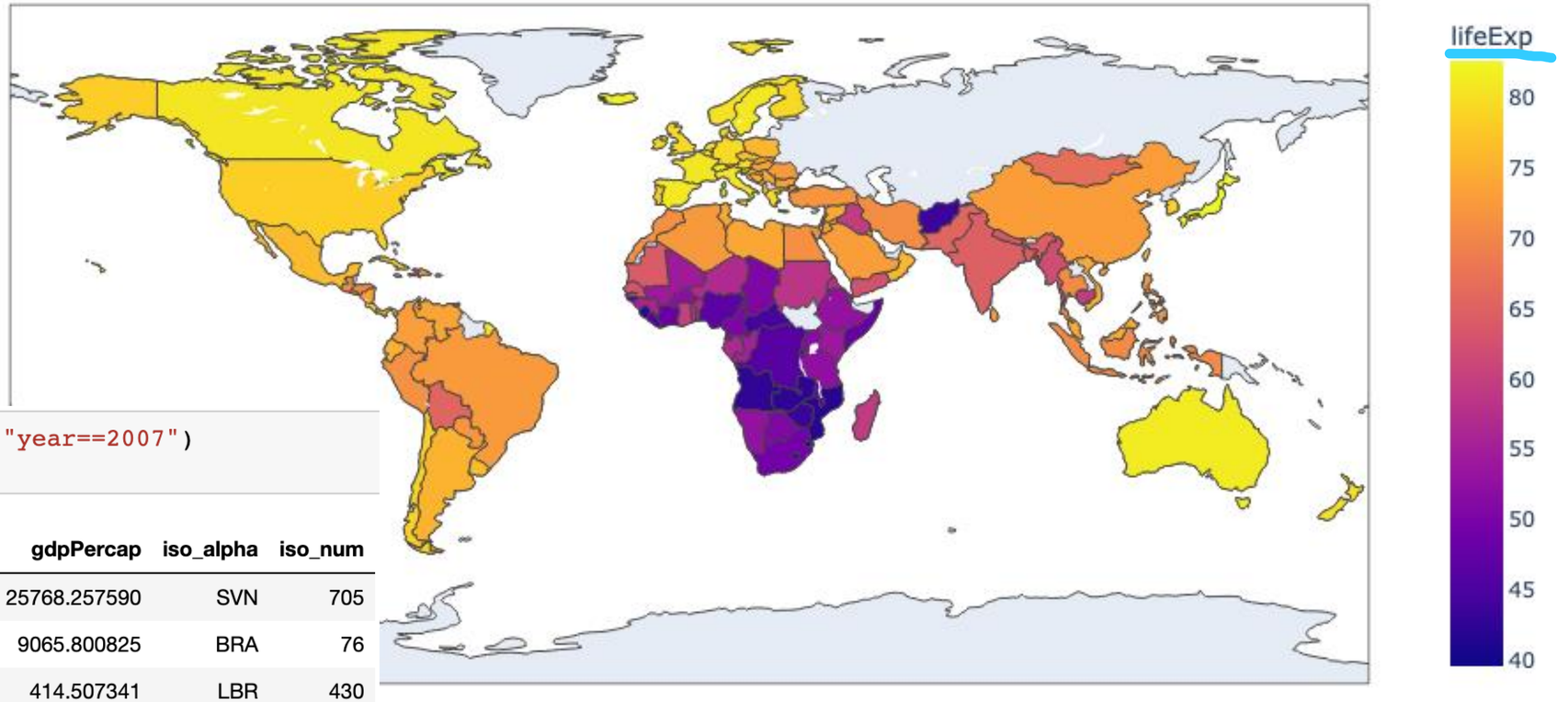

```python
df_exp = px.data.gapminder().query("year==2007")
df_exp.sample(5)
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
1391	Slovenia	Europe	2007	77.926	2009245	25768.257590	SVN	705
179	Brazil	Americas	2007	72.390	190010647	9065.800825	BRA	76
899	Liberia	Africa	2007	45.678	3193942	414.507341	LBR	430

px.choropleth – Choropleth Map

```
1  fig = px.choropleth(df_exp, locations="iso_alpha",
2                      color="lifeExp", # lifeExp is a column of gapminder
3                      hover_name="country", # column to add to hover information
4                      color_continuous_scale=px.colors.sequential.Plasma)
5  fig.show()
```

In a choropleth map, each row of data_frame is represented by a colored region mark on a map.



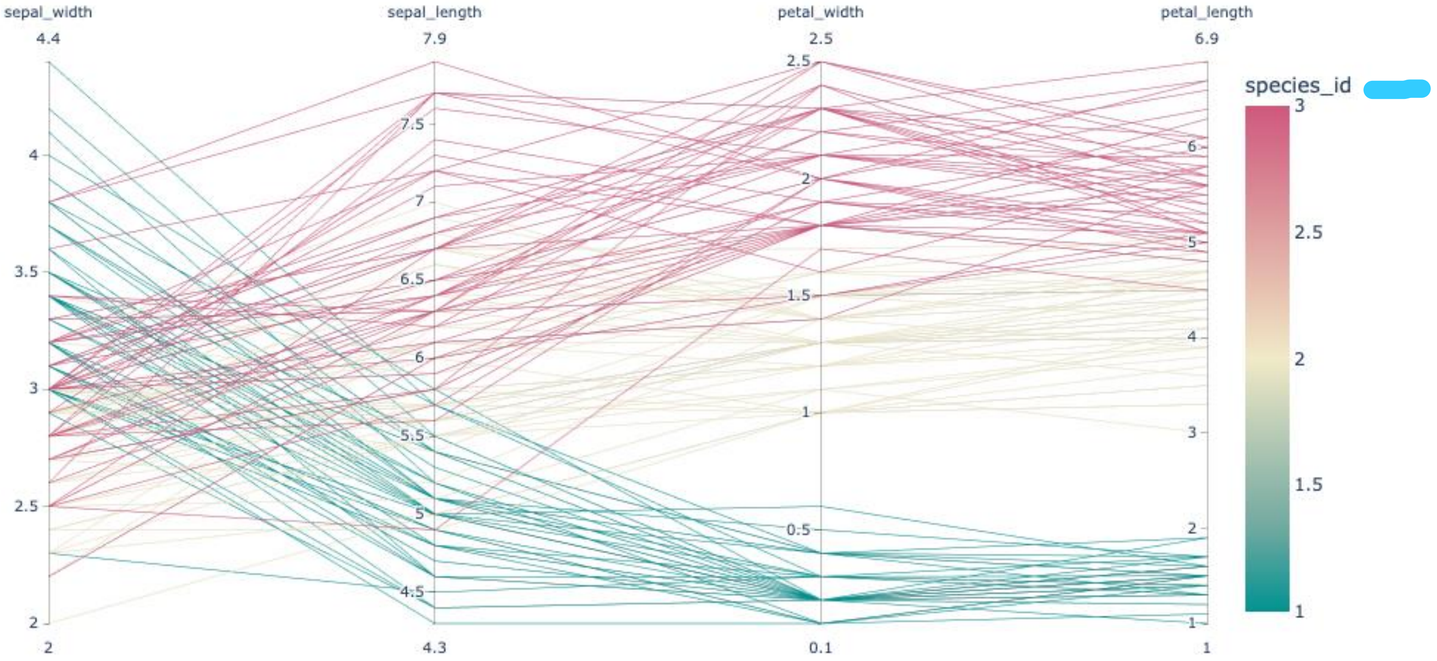

```
1  df_exp = px.data.gapminder().query("year==2007")
2  df_exp.sample(5)
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
1391	Slovenia	Europe	2007	77.926	2009245	25768.257590	SVN	705
179	Brazil	Americas	2007	72.390	190010647	9065.800825	BRA	76
899	Liberia	Africa	2007	45.678	3193942	414.507341	LBR	430

px.parallel_coordinates

In a parallel coordinates plot with px.parallel_coordinates, each row of the DataFrame is represented by a polyline mark which traverses a set of parallel axes, one for each of the dimensions.

```
fig = px.parallel_coordinates(df_iris, color="species_id",
                              dimensions=['sepal_width', 'sepal_length', 'petal_width',
                                          'petal_length'],
                              color_continuous_scale=px.colors.diverging.Tealrose,
                              color_continuous_midpoint=2)
fig.show()
```




```
df_iris.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
23	5.1	3.3	1.7	0.5	setosa	1
28	5.2	3.4	1.4	0.2	setosa	1
138	6.0	3.0	4.8	1.8	virginica	3
32	5.2	4.1	1.5	0.1	setosa	1
66	5.6	3.0	4.5	1.5	versicolor	2

Can you figure out the idea of machine learning in Object Recognition?

Graph Objects

The plotly.graph_objects module (typically imported as go) contains an automatically-generated hierarchy of Python classes which represent non-leaf nodes in this figure schema.

The term "graph objects" refers to instances of these classes.

Functions in Plotly Express, which is the recommended entry-point into the plotly library, are all built on top of graph objects, and all return instances of plotly.graph_objects.Figure.

You might see code one public resources like “import plotly.graph_objects as go”

This should be either earlier version plotly code or tend to customize some of the attributes, parameter, or setting.

Matplotlib vs Plotly

Features	matplotlib	plotly | Graphing Libraries
Syntax	Complex and lengthy	Shorter code and simple structure in px
Visual Design	Standard output and full range of plots	Publication style output with interactive functions
Popularity	Popular in data science	User number is growing fast
Advance Function	Various plots with standard function	Additional advance function available
Coding Requirement	Comparatively simple to use	Skilful coding on advance function such as live data on Dash

Reference

Official Website:

https://plotly.com/python/

Plotly Express:

https://plotly.com/python/plotly-express/

GitHub Open Source Code:

https://github.com/plotly/plotly.py