Introduction to Data Science

Dash & Plotly

TA: 李姵徵

Introduction

- **plotly**
  - 開源套件，主要用來畫圖及做出視覺化互動報表
  - 支援多種程式語言，如：Python、R、JavaScript等
  - 支援線上和離線兩種模式，提供穩定的API可與現有的應用整合

- **Dash**
  - 輕量型的Python網頁應用程式框架
  - 不需要 JavaScript 也可以建立出具備高度互動性的圖表與儀表板

Install & Import

```
!pip install plotly
!pip install cufflinks
!pip install dash
!pip install jupyter-dash
```

- **cufflinks**
  - 在plotly的基礎上做進一步的包裝
  - 使用方法統一且參數的配置簡單

```
import pandas as pd
import numpy as np
import plotly as py
import cufflinks as cf
import seaborn as sns
import plotly.express as px
%matplotlib inline

# Allows us to create graph objects for making more customized plots
import plotly.graph_objects as go

# Allows us to grab data from a supplied URL
from urllib.request import urlopen

# Used to decode JSON data
import json

# Make Plotly work in your Jupyter Notebook
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)

# Use Plotly locally
cf.go_offline()
```

plotly express & graph object

- **plotly express (px)**
  - 語法簡潔、功能強大
  - 可繪製大部分的圖表類型，如：散布圖、長條圖、柱狀圖等

- **graph object (go)**
  - 具有更加詳細的報錯信息
  - 屬性修改簡單，便於更新圖像，如：添加新內容、更改布局等

Line Plot

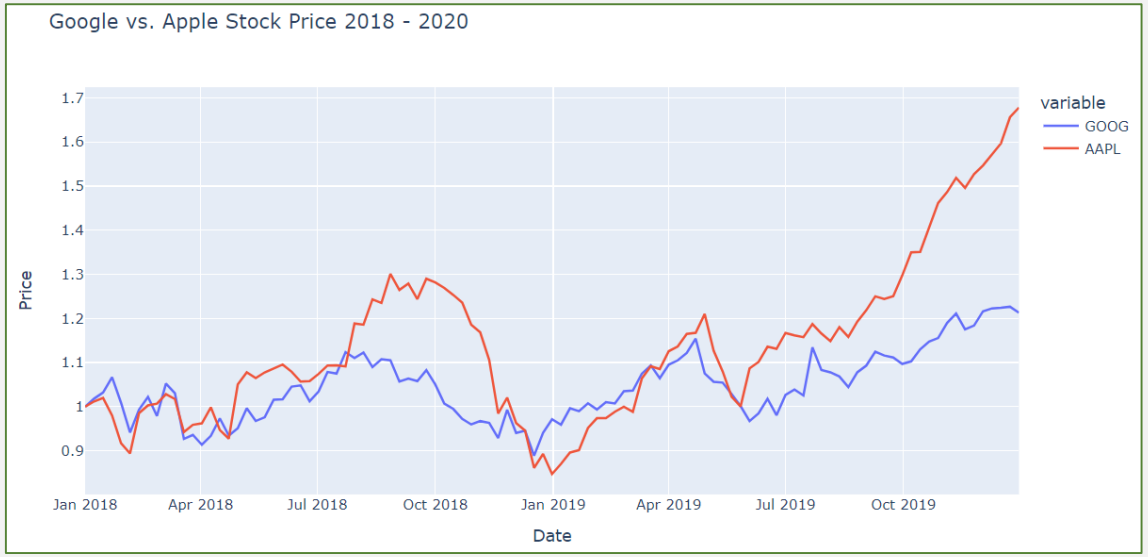
```
# Use included stocks data
df_stocks = px.data.stocks()
df_stocks.head()
```

	date	GOOG	AAPL	AMZN	FB	NFLX	MSFT
0	2018-01-01	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
1	2018-01-08	1.018172	1.011943	1.061881	0.959968	1.053526	1.015988
2	2018-01-15	1.032008	1.019771	1.053240	0.970243	1.049860	1.020524
3	2018-01-22	1.066783	0.980057	1.140676	1.016858	1.307681	1.066561
4	2018-01-29	1.008773	0.917143	1.163374	1.018357	1.273537	1.040708

```
# Make a one line plot
px.line(df_stocks,
        x = 'date',
        y = 'GOOG',
        labels = {'date': 'Date',
                  'GOOG': 'Price'},
        title = 'Google Stock Price 2018 - 2020')
```

Line Plot

```
# Make a multiple line plot
px.line(df_stocks,
        x = 'date',
        y = ['GOOG', 'AAPL'],
        labels = {'date': 'Date',
                  'value': 'Price'},
        title = 'Google vs. Apple Stock Price 2018 - 2020')
```



- 使用list將想選的變數都包起來
- 改變軸的名稱
  - 一個變數 ⇒ 原本的名稱: 要更改的名稱
  - 多個變數 ⇒ value: 要更改的名稱

Line Plot

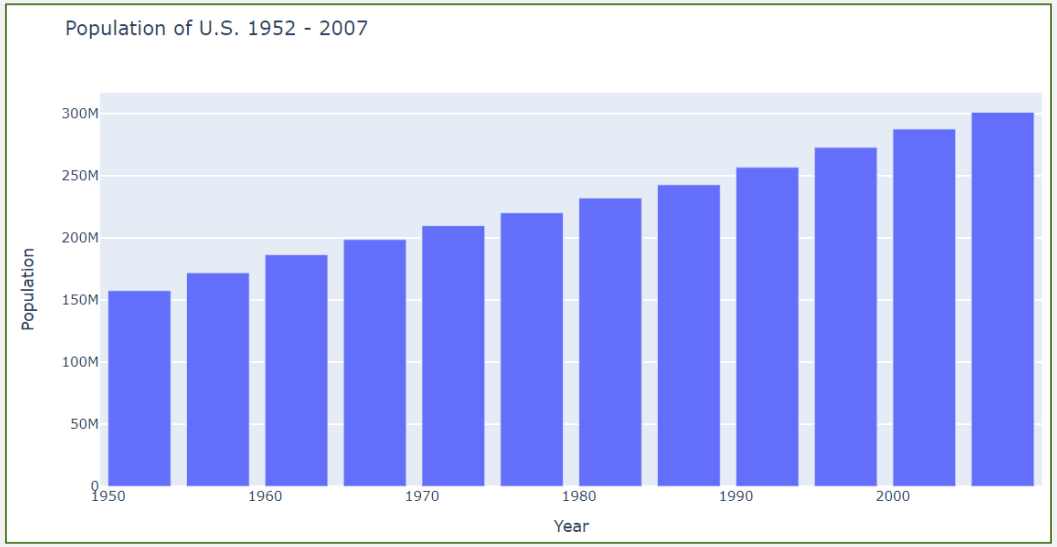
```
# Create a figure to add plots
fig = go.Figure()

fig.add_trace(go.Scatter(x = df_stocks.date,
                         y = df_stocks.GOOG,
                         mode = 'lines',
                         name = 'Google'))
fig.add_trace(go.Scatter(x = df_stocks.date,
                         y = df_stocks.AAPL,
                         mode = 'lines+markers',
                         name = 'Apple'))

# # Further style the figure
fig.update_layout(title = 'Stock Price Data 2018 - 2020',
                  xaxis_title = 'Date',
                  yaxis_title = 'Price')
```

Bar Chart

```
# Load data
df_gap = px.data.gapminder()
df_gap.head()
```

	country	continent	year	lifeExp	pop	gdpPercap	iso_alpha	iso_num
0	Afghanistan	Asia	1952	28.801	8425333	779.445314	AFG	4
1	Afghanistan	Asia	1957	30.332	9240934	820.853030	AFG	4
2	Afghanistan	Asia	1962	31.997	10267083	853.100710	AFG	4
3	Afghanistan	Asia	1967	34.020	11537966	836.197138	AFG	4
4	Afghanistan	Asia	1972	36.088	13079460	739.981106	AFG	4

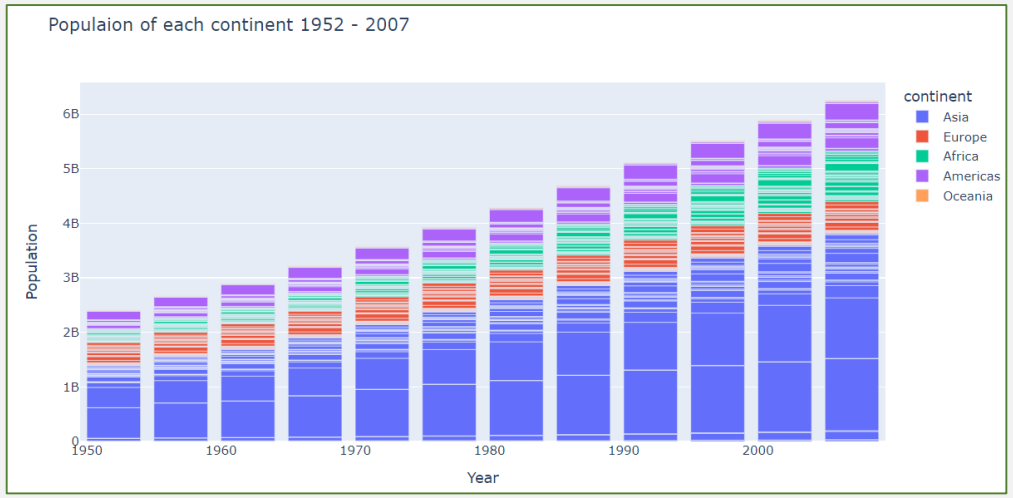
```
px.bar(df_gap.groupby('country').get_group('United States'),
       x = 'year',
       y = 'pop',
       labels = {'year': 'Year',
                 'pop': 'Population'},
       title = 'Population of U.S. 1952 - 2007'
      )
```



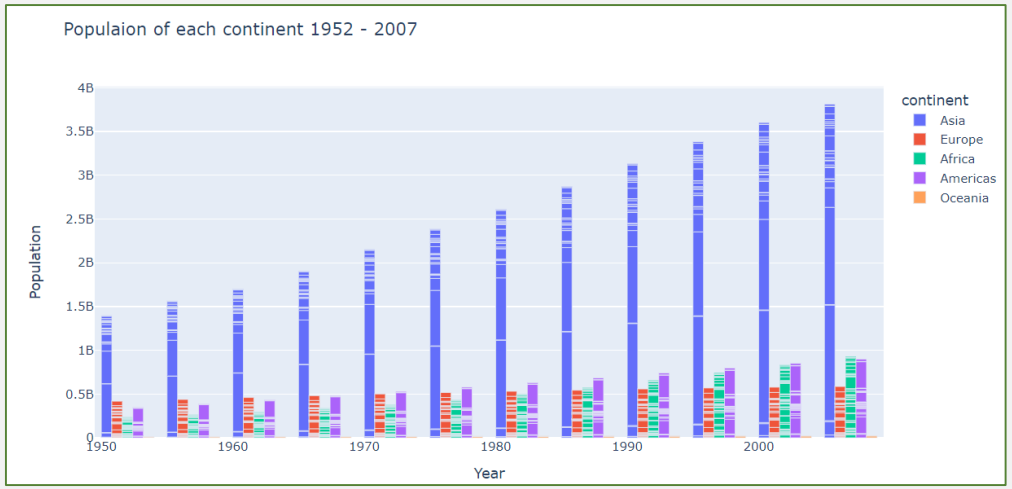
- **Horizontal Bar Chart**
  - orientation = 'h‘

Bar Chart

```
# Create a stacked bar
px.bar(df_gap,
       x = 'year',
       y = 'pop',
       color = 'continent',
       labels = {'year': 'Year',
                 'pop': 'Population'},
       title = 'Populaion of each continent 1952 - 2007')
```

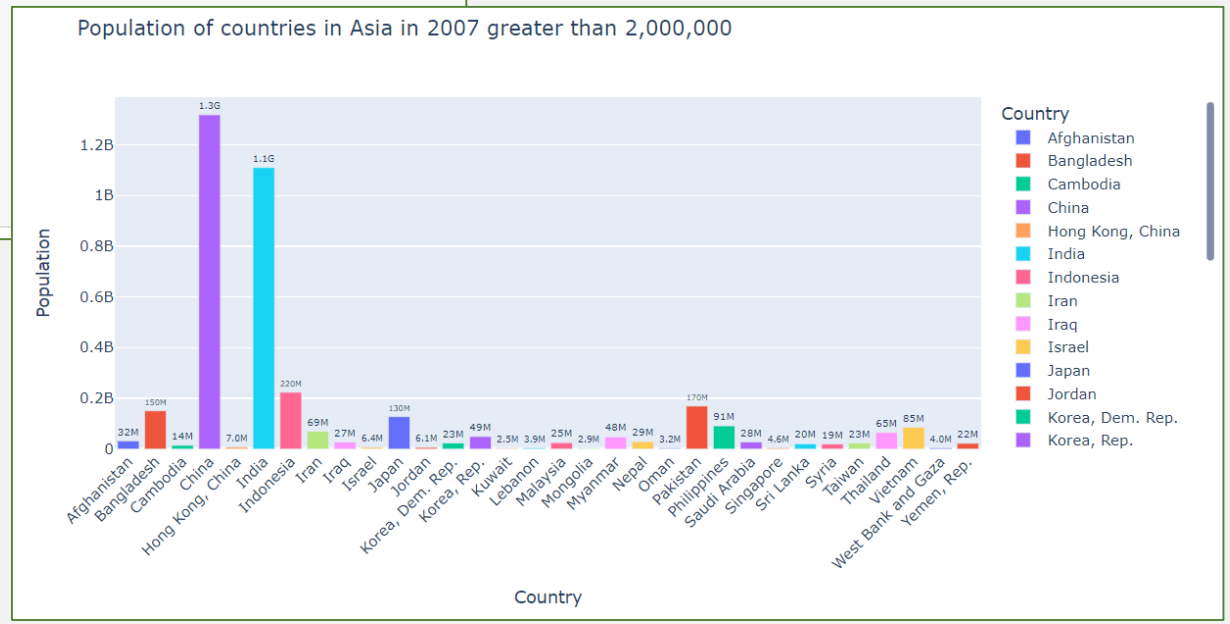



```
# Create a stacked bar
px.bar(df_gap,
       x = 'year',
       y = 'pop',
       color = 'continent',
       barmode = 'group',
       labels = {'year': 'Year',
                 'pop': 'Population'},
       title = 'Populaion of each continent 1952 - 2007')
```



Bar Chart

```
# Display pop data for countries in Asia in 2007 greater than 2000000
fig = px.bar(df_gap[(df_gap['continent'] == 'Asia') & (df_gap['pop'] > 2000000) & (df_gap['year'] == 2007)],
             x = 'country',
             y = 'pop',
             text = 'pop',
             color = 'country',
             labels = {'country': 'Country',
                       'pop': 'Population'},
             title = 'Population of countries in Asia in 2007 greater than 2,000,000')

# Put total Population above bars with 2 values of precision
fig.update_traces(texttemplate = '%{text:.2s}',
                  textposition = 'outside')

# Set fontsize and Rotate labels 45 degrees
# uniformtext_mode='hide': hide the text if it won't fit
fig.update_layout(uniformtext_minsize = 8,
                  xaxis_tickangle = -45)
```



- **trace**
  - 僅能相對定位，無法絕對定位
- **uniformtext**
  - 強制所有文本標籤具有相同大小

Scatter Plot

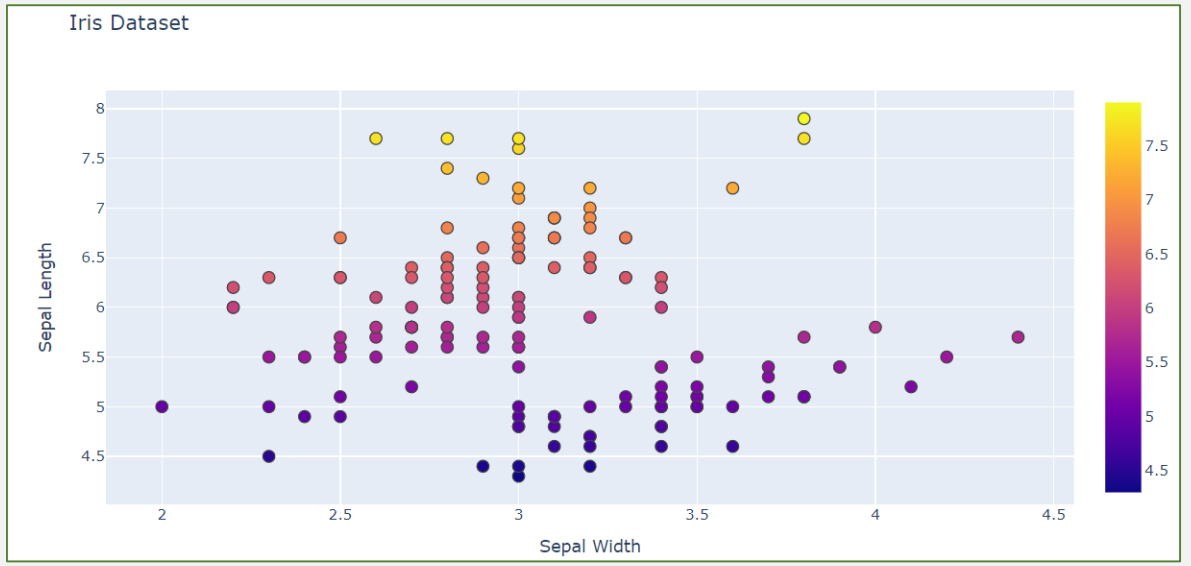
```
# Iris data set
df_iris = px.data.iris()
df_iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species	species_id
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1

```
# Create a scatter plot
px.scatter(df_iris,
           x = "sepal_width",
           y = "sepal_length",
           color = "species",
           size = 'petal_length',
           hover_data = ['petal_width'],
           labels = {'sepal_width': 'Sepal Width',
                     'sepal_length': 'Sepal Length'},
           title = 'Iris dataset')
```

Scatter Plot

```
# Create a customized scatter plot
fig = go.Figure()
fig.add_trace(go.Scatter(x = df_iris.sepal_width,
                         y = df_iris.sepal_length,
                         mode = 'markers',
                         marker_color = df_iris.sepal_length,
                         text = df_iris.species,
                         marker = dict(showscale = True)))

# Customize marker edges with line width 1 and black
fig.update_traces(marker_line_width = 1,
                  marker_size = 10)

fig.update_layout(title = 'Iris Dataset',
                  xaxis_title = 'Sepal Width',
                  yaxis_title = 'Sepal Length')
```

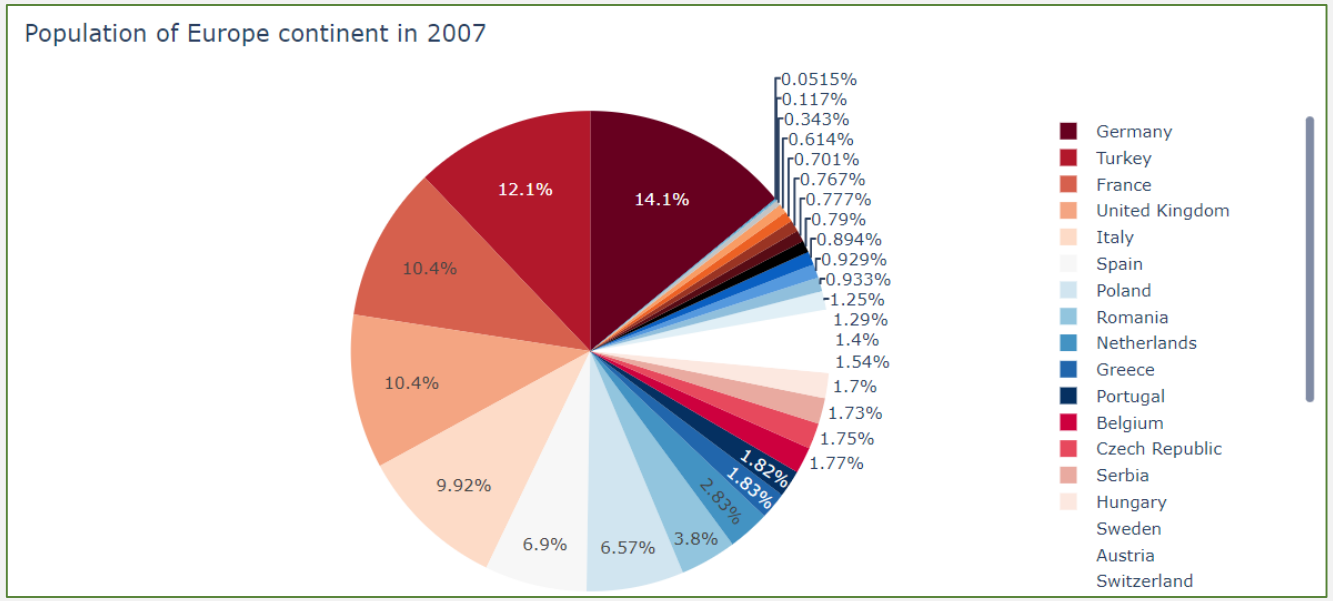


- **from plotly.validators.scatter.marker import SymbolValidator**
  - 有許多可用的symbol
  - 基本款：circle, square, diamond, cross, x, triangle, pentagon, hexagram, star, diamond, hourglass, bowtie, asterisk, hash, y, and line

Pie Chart

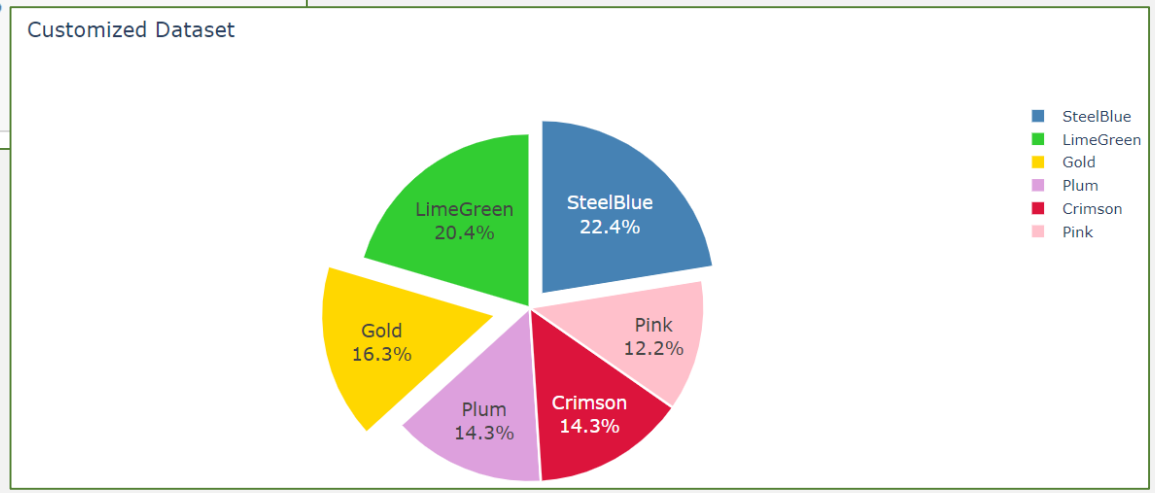
```
# Create a pie chart
px.pie(df_gap[(df_gap.year == 2007) & (df_gap.continent == 'Europe')],
       values = 'pop',
       names = 'country',
       labels = {'pop': 'Population'},
       title = 'Population of Europe continent in 2007',
       color_discrete_sequence = px.colors.sequential.RdBu)
```

- **px.colors.sequential**
  - https://plotly.com/python/builtin-colorscales/



Pie Chart

```
# Customize a pie chart
colors = ['SteelBlue', 'LimeGreen', 'Gold', 'Plum', 'Crimson', 'Pink']
fig = go.Figure(data=[go.Pie(labels = colors,
                             values = [110, 100, 80, 70, 70, 60])])

# Define hover information, text size, pull amount for each pie slice, and stroke
fig.update_traces(hoverinfo = 'label+percent',
                  textfont_size = 15,
                  textinfo = 'label+percent',
                  pull = [0.1, 0, 0.2, 0, 0, 0],
                  marker = dict(colors = colors,
                                line = dict(color = '#FFFFFF',
                                            width = 2)))

fig.update_layout(title = 'Customized Dataset')
```

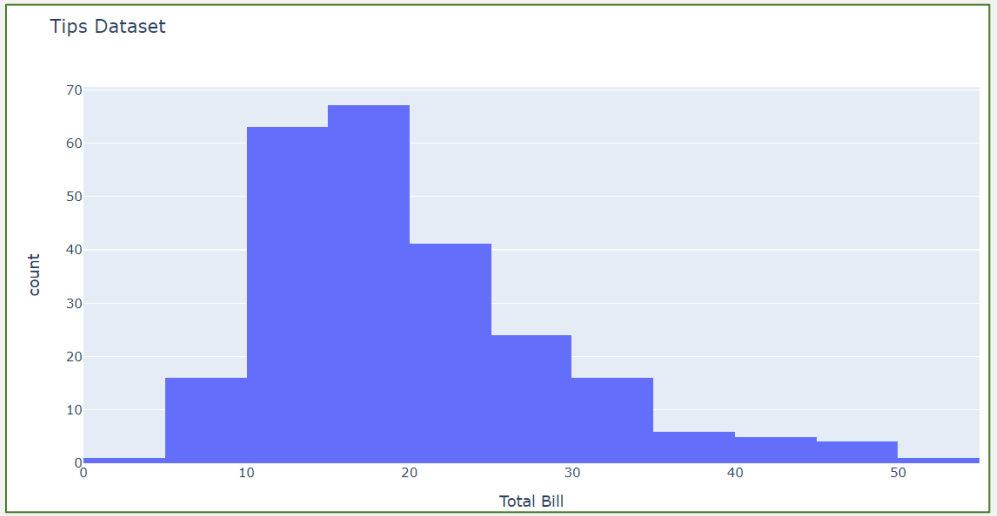


Histogram

```
# Load Dataset
df_tips = px.data.tips()
df_tips.head()
```

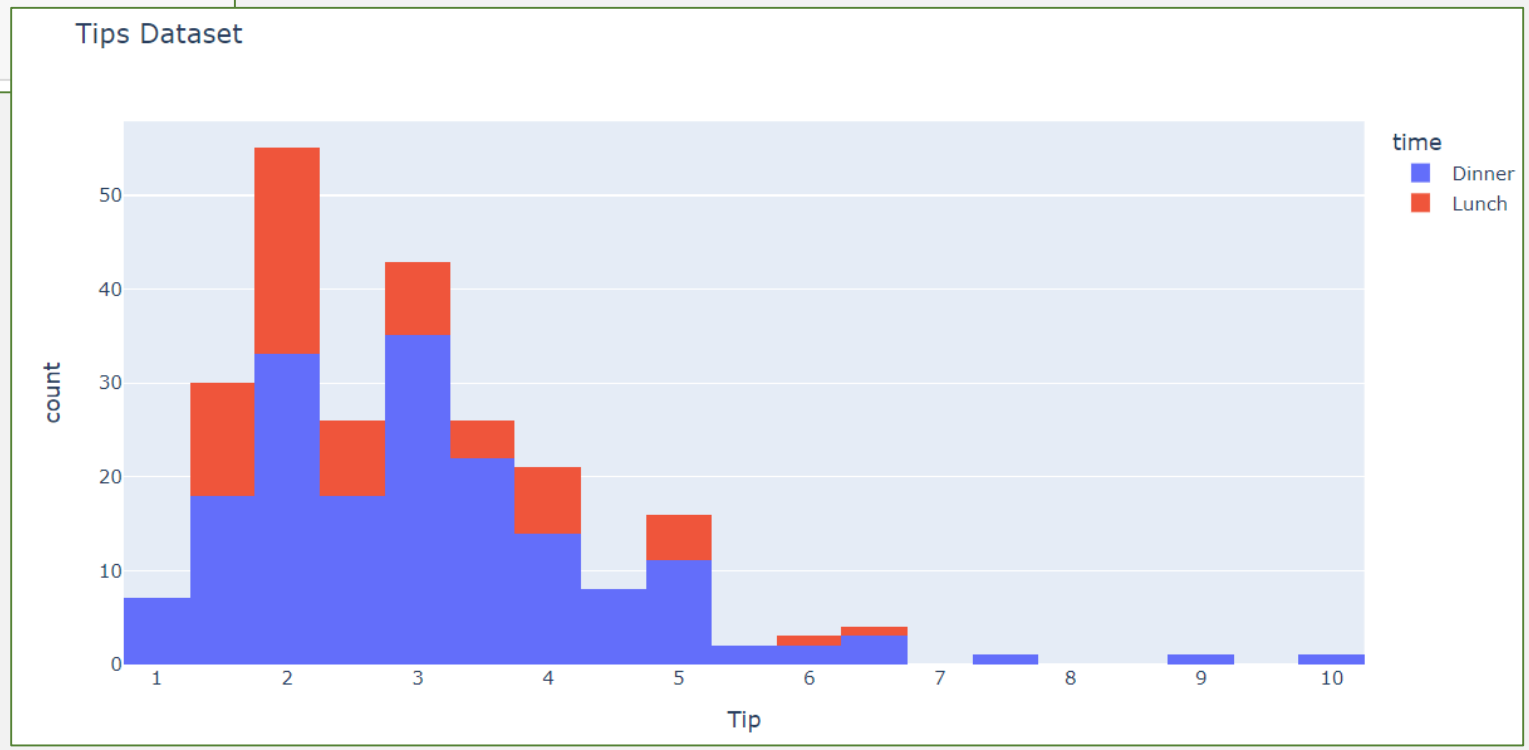
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
# Create a histogram
px.histogram(df_tips,
             x = 'total_bill',
             nbins = 10,
             labels = {'total_bill': 'Total Bill'},
             title = 'Tips Dataset')
```



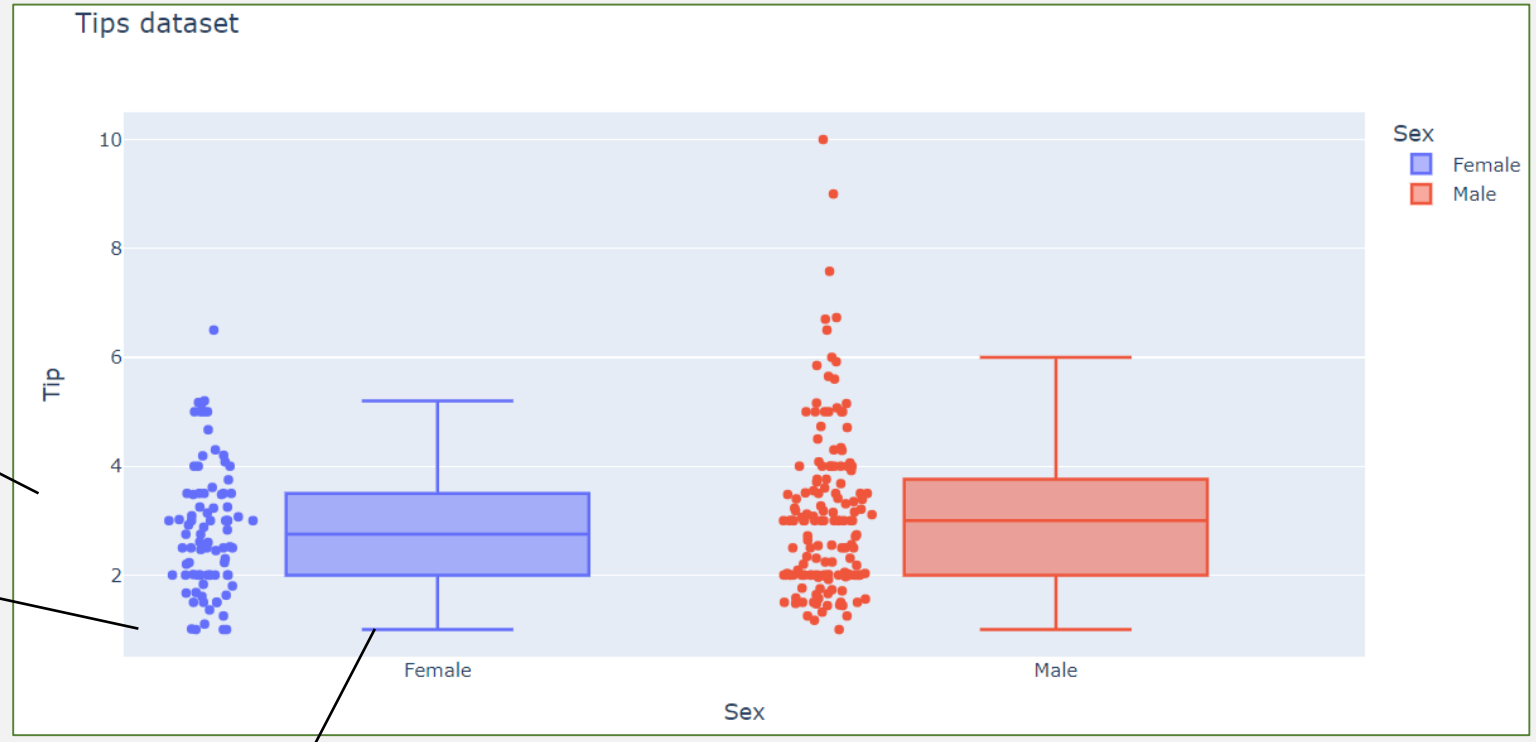
Histogram

```
# Stack histograms based on different column data
px.histogram(df_tips,
             x = 'tip',
             color = 'time',
             labels = {'tip': 'Tip'},
             title = 'Tips Dataset')
```



Box Plot

```
# Create a box plot
px.box(df_tips,
       x = 'sex',
       y = 'tip',
       points = 'all',
       color = 'sex',
       labels = {'sex': 'Sex',
                 'tip': 'Tip'},
       title = 'Tips dataset')
```



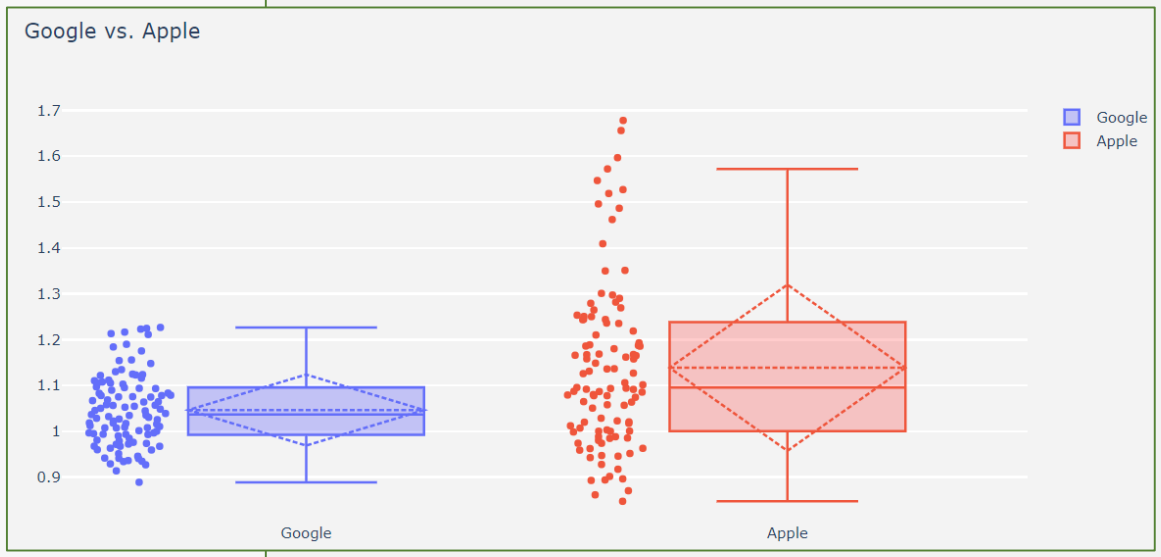
paper_bgcolor

plot_bgcolor

whisker

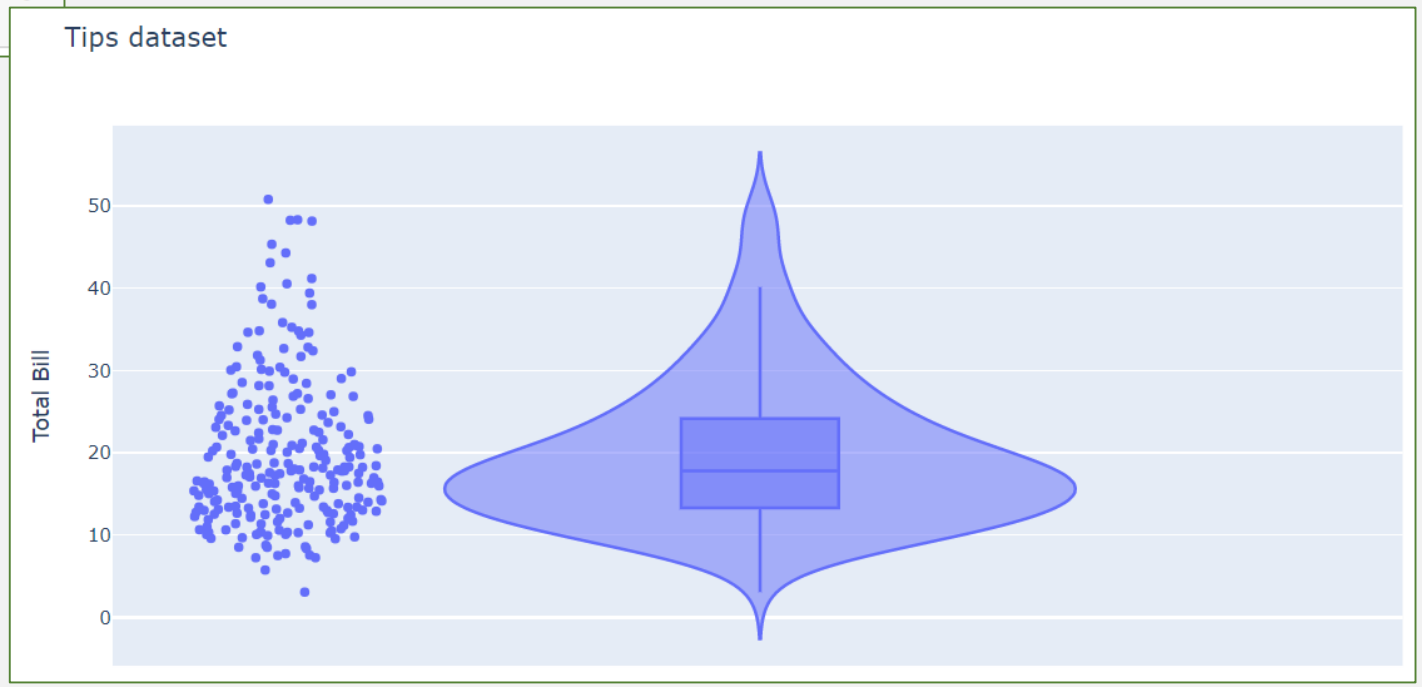
Box Plot

```
# Customerize a box plot
fig = go.Figure()
# Show all points, spread them so they don't overlap and change whisker width
fig.add_trace(go.Box(y = df_stocks.GOOG,
                     boxpoints = 'all',
                     name = 'Google',
                     fillcolor = 'rgba(0, 0, 255, 0.2)',
                     boxmean = 'sd',
                     jitter = 0.4,
                     whiskerwidth = 0.6))

fig.add_trace(go.Box(y = df_stocks.AAPL,
                     boxpoints = 'all',
                     name = 'Apple',
                     fillcolor = 'rgba(255, 0, 0, 0.2)',
                     boxmean = 'sd',
                     jitter = 0.4,
                     whiskerwidth = 0.6))

# Change background / grid colors
fig.update_layout(title = 'Google vs. Apple',
                  yaxis = dict(gridcolor = 'rgb(255, 255, 255)',
                               gridwidth = 2),
                  paper_bgcolor = 'rgb(243, 243, 243)',
                  plot_bgcolor = 'rgb(243, 243, 243)')
```



Violin Plot

```
# Create a violin plot
px.violin(df_tips,
          y = 'total_bill',
          box = True,
          points = 'all',
          labels = {'total_bill': 'Total Bill'},
          title = 'Tips dataset')
```



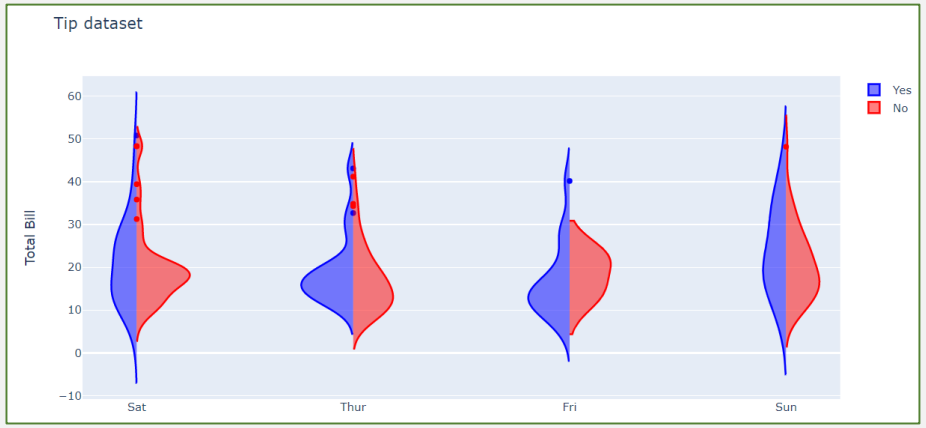
- **violin plot**
  - box plot + 機率密度函數
  - 可展示資料更多資訊

Violin Plot

```
# Create multiple violin plots
px.violin(df_tips,
          y = 'tip',
          x = 'smoker',
          color = 'sex',
          box = True,
          points = 'all',
          labels = {'tip': 'Tip',
                    'smoker': 'Smoker'},
          hover_data = df_tips.columns,
          title = 'Tip data')
```

Violin Plot

```
# Morph left and right sides based on whether the customer smokes
fig = go.Figure()
fig.add_trace(go.Violin(x = df_tips.day[df_tips.smoker == 'Yes'],
                        y = df_tips.total_bill[df_tips.smoker == 'Yes'],
                        legendgroup = 'Yes',
                        scalegroup = 'Yes',
                        name = 'Yes',
                        side = 'negative',
                        line_color = 'blue'))

fig.add_trace(go.Violin(x = df_tips.day[df_tips.smoker == 'No'],
                        y = df_tips.total_bill[df_tips.smoker == 'No'],
                        legendgroup = 'Yes',
                        scalegroup = 'Yes',
                        name = 'No',
                        side = 'positive',
                        line_color = 'red'))

fig.update_layout(title = 'Tip dataset',
                  yaxis_title = 'Tip')
```



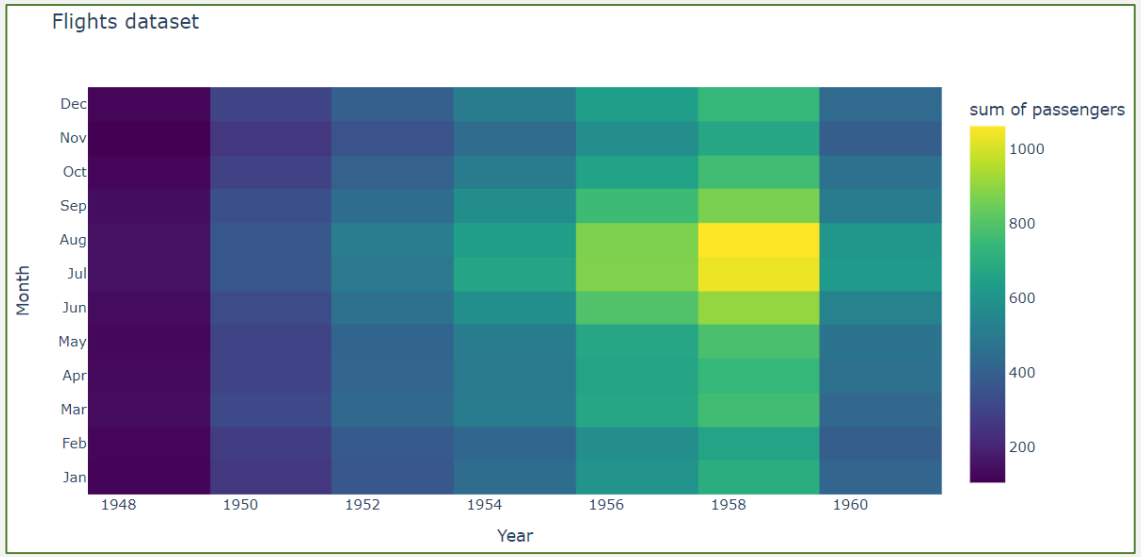
- **legendgroup**
  - 將同一群的legend合在一起
  - 可藉由點選圖例的名稱將整群一起顯示或隱藏

Density Heatmap

```
# Using Seaborn data
flights = sns.load_dataset("flights")
flights.head()
```

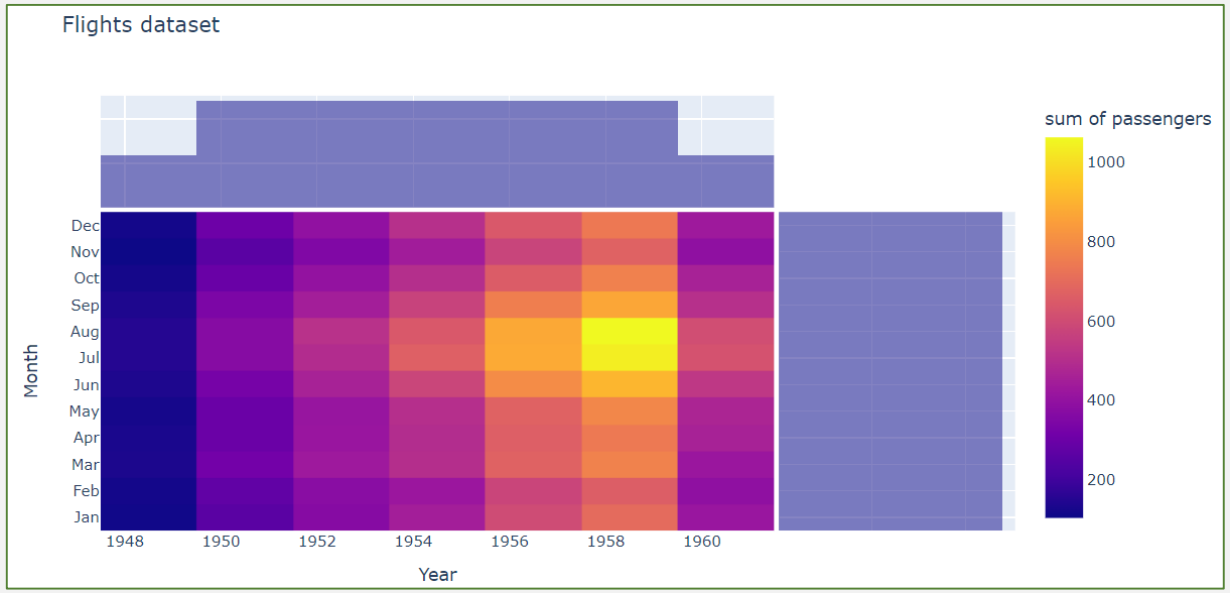
	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121

```
# Create a density heatmap
px.density_heatmap(flights,
                   x = 'year',
                   y = 'month',
                   z = 'passengers',
                   color_continuous_scale="Viridis",
                   labels = {'year': 'Year',
                             'month': 'Month'},
                   title = 'Flights dataset')
```



Density Heatmap

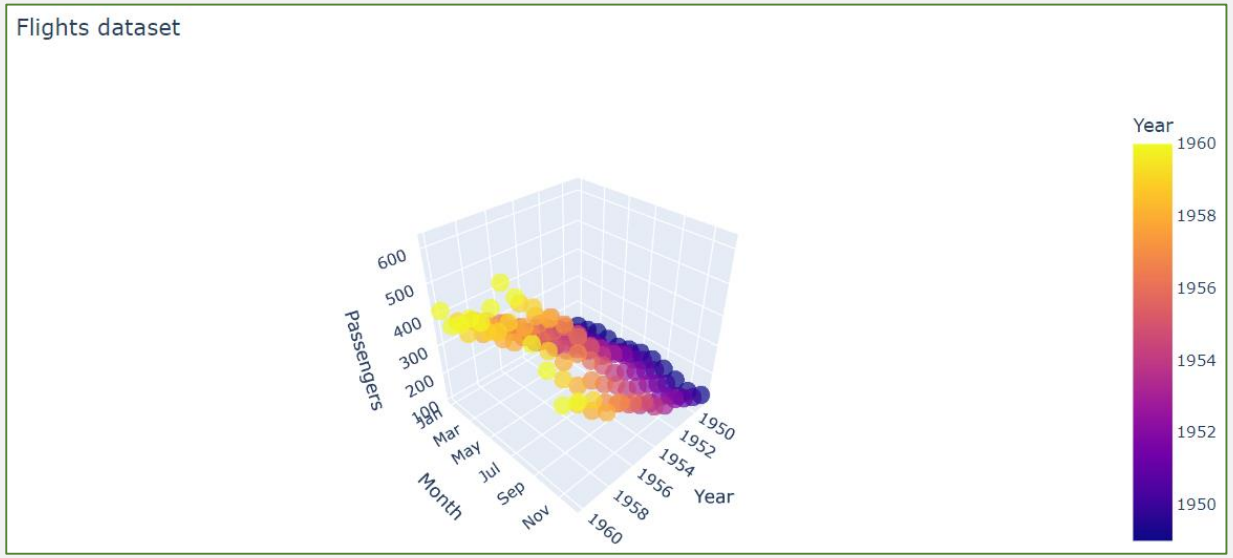
```
# Add histograms
px.density_heatmap(flights,
                   x = 'year',
                   y = 'month',
                   z = 'passengers',
                   marginal_x = "histogram",
                   marginal_y = "histogram",
                   labels = {'year': 'Year',
                             'month': 'Month'},
                   title = 'Flights dataset')
```



- **color_continuous_scales**
  - 顏色變化可參考P. 13的連結

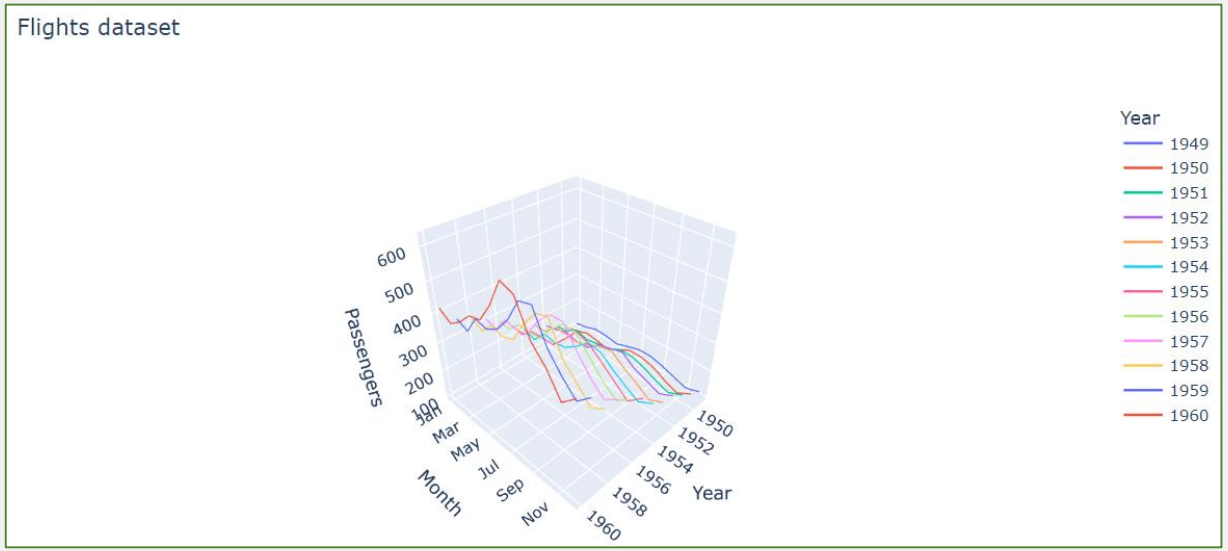
3D Scatter Plot

```
# Create a 3D scatter plot
px.scatter_3d(flights,
              x = 'year',
              y = 'month',
              z = 'passengers',
              color = 'year',
              opacity = 0.7,
              labels = {'year': 'Year',
                        'month': 'Month',
                        'passengers': 'Passengers'},
              title = 'Flights dataset')
```



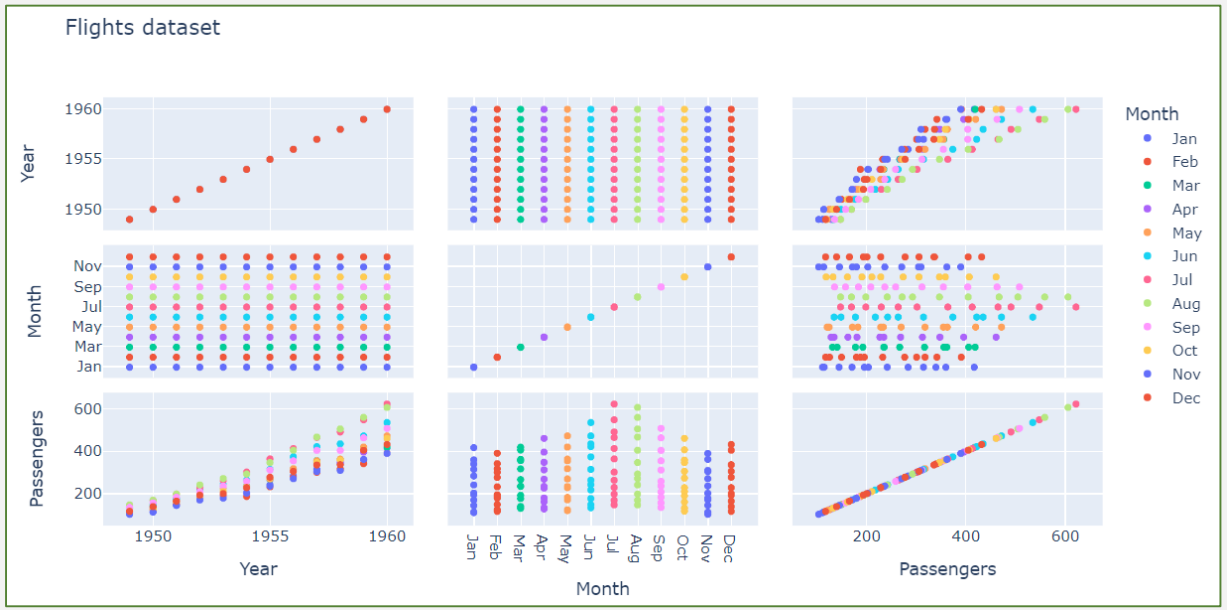
3D Line Plot

```
# Create a 3D Line Plot
px.line_3d(flights,
           x = 'year',
           y = 'month',
           z = 'passengers',
           color = 'year',
           labels = {'year': 'Year',
                     'month': 'Month',
                     'passengers': 'Passengers'},
           title = 'Flights dataset')
```



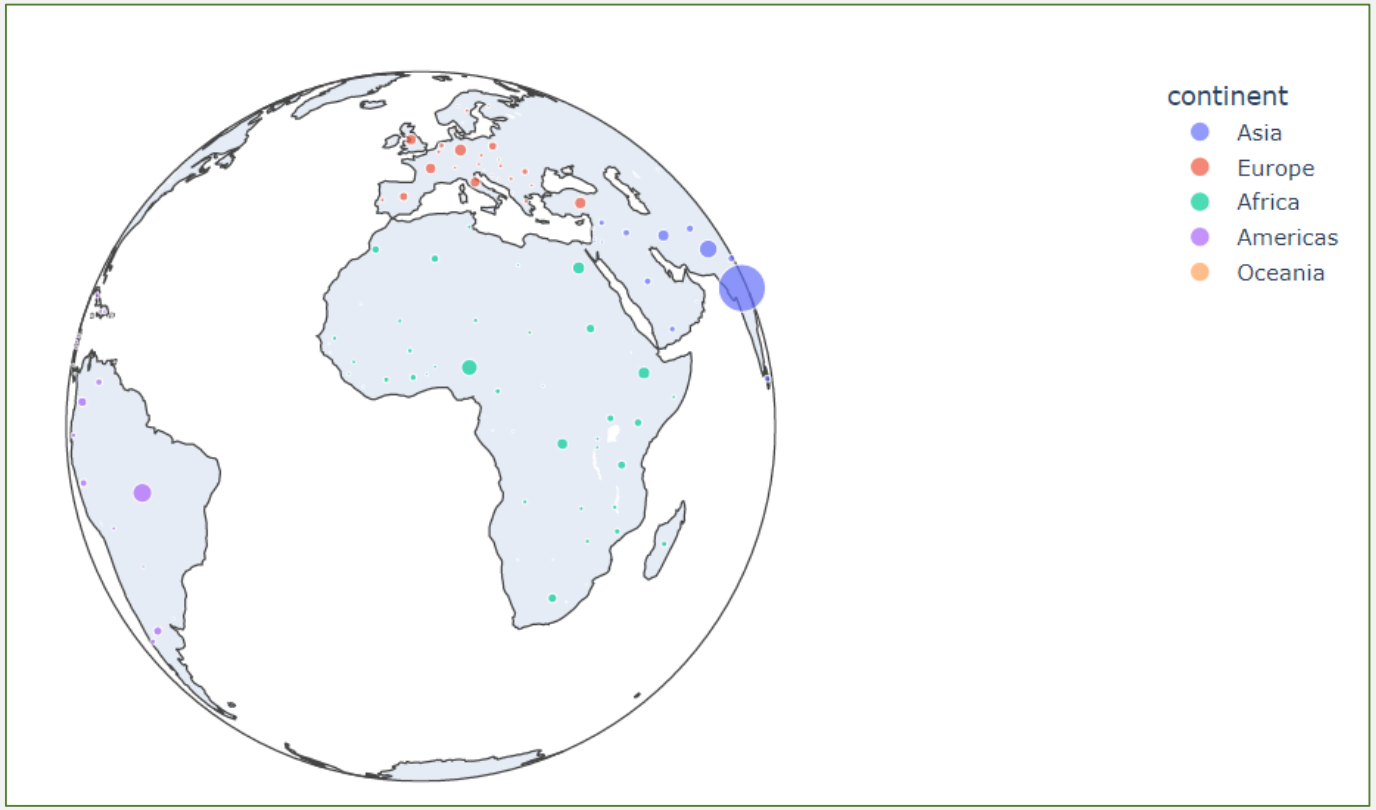
Scatter Matrix

```
# Create a scatter matrix
px.scatter_matrix(flights,
                  color = 'month',
                  labels = {'year': 'Year',
                            'month': 'Month',
                            'passengers': 'Passengers'},
                  title = 'Flights dataset')
```



Map Scatter Plot

```
# Create a map scatter plot
px.scatter_geo(df_gap[df_gap.year == 2007],
               locations = "iso_alpha",
               color = "continent",
               hover_name = "country",
               size = "pop",
               projection = "orthographic")
```



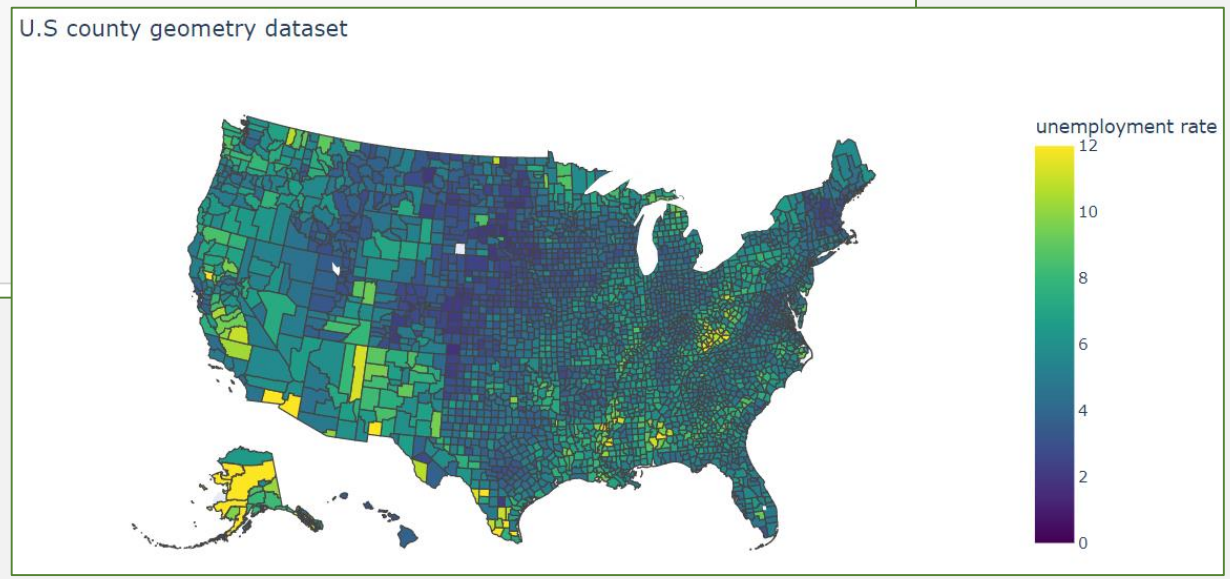
- **projection**
  - equirectangular

    (等距長方投影)
  - mercator

    (麥卡托 / 等角圓柱投影)
  - natural earth

Choropleth Map

```
# Grab US county geometry data
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.json') as response:
    counties = json.load(response)

# Grab unemployment data based on each counties Federal Information Processing number
df = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/fips-unemp-16.csv", dtype={"fips": str})

# Create a choropleth map using the county JSON data, color using unemployment values on a range of 12
px.choropleth(df,
              geojson = counties,
              locations = 'fips',
              color = 'unemp',
              color_continuous_scale = "Viridis",
              range_color = (0, 12),
              scope = "usa",
              labels = {'unemp':'unemployment rate'},
              title = 'U.S county geometry dataset')
```

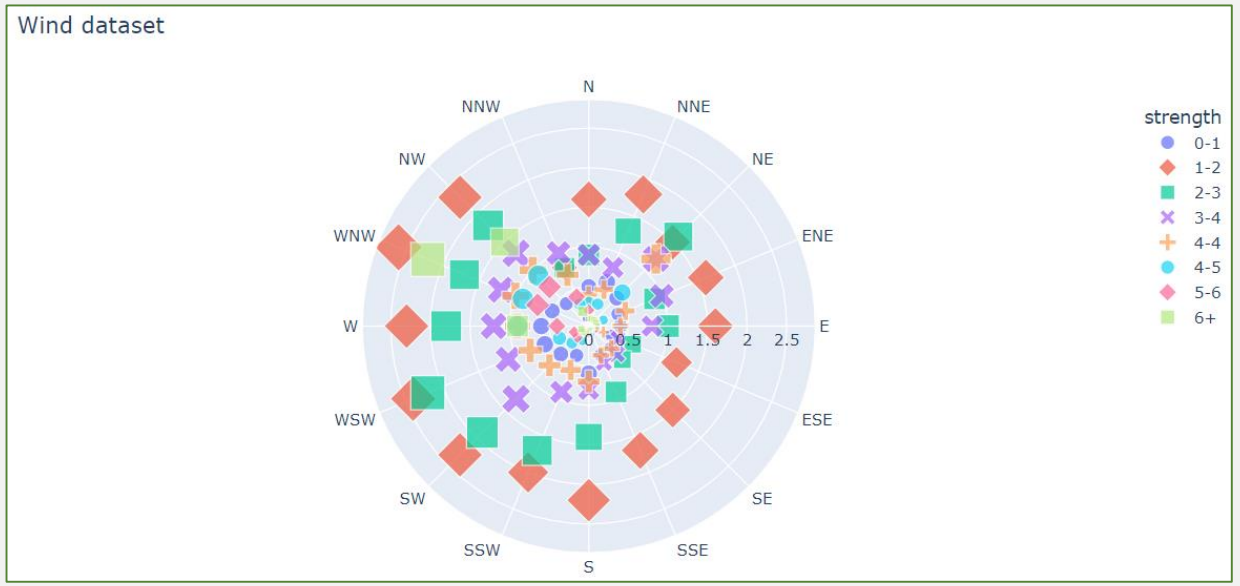


Polar Chart

```
# Load dataset
df_wind = px.data.wind()
df_wind.head()
```

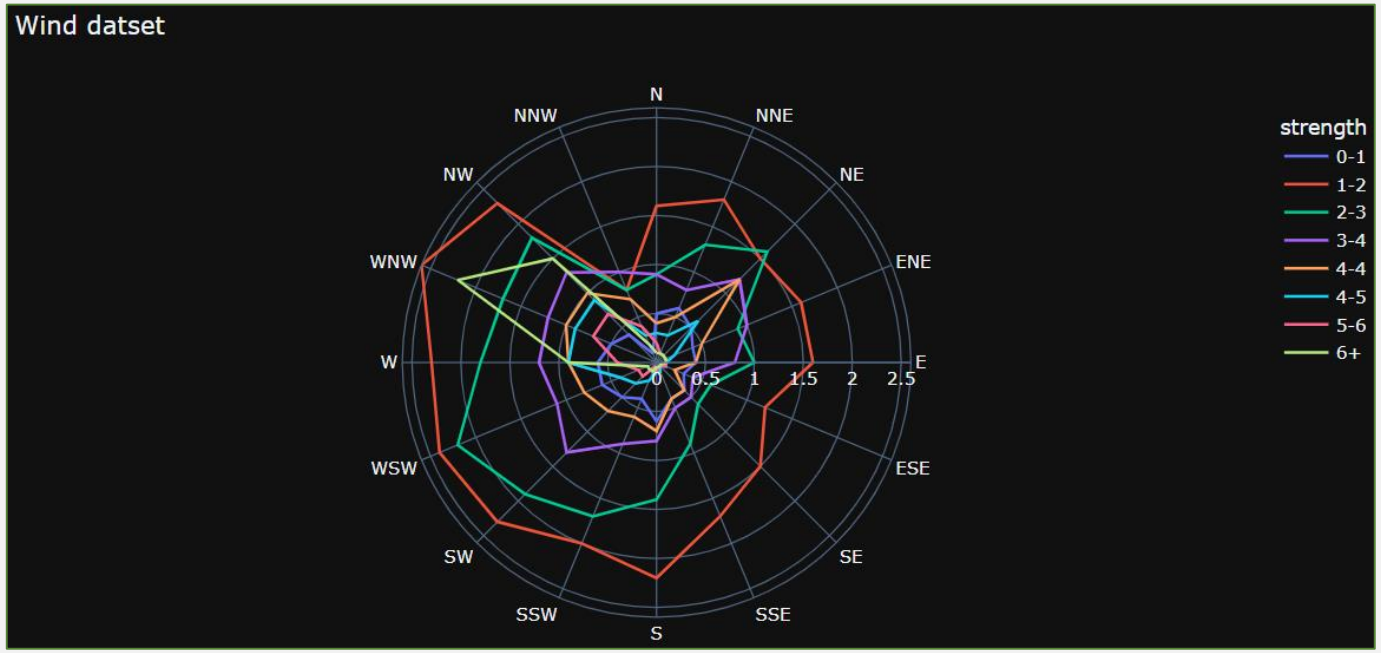
	direction	strength	frequency
0	N	0-1	0.5
1	NNE	0-1	0.6
2	NE	0-1	0.5
3	ENE	0-1	0.4
4	E	0-1	0.4

```
# Create a polar chart
px.scatter_polar(df_wind,
                 r = "frequency",
                 theta = "direction",
                 color = "strength",
                 size = "frequency",
                 symbol = "strength",
                 title = 'Wind dataset')
```



Polar Chart

```
# Create a polar chart with line style
px.line_polar(df_wind,
              r = "frequency",
              theta = "direction",
              color = "strength",
              line_close = True,
              template = "plotly_dark",
              title = 'Wind datset')
```

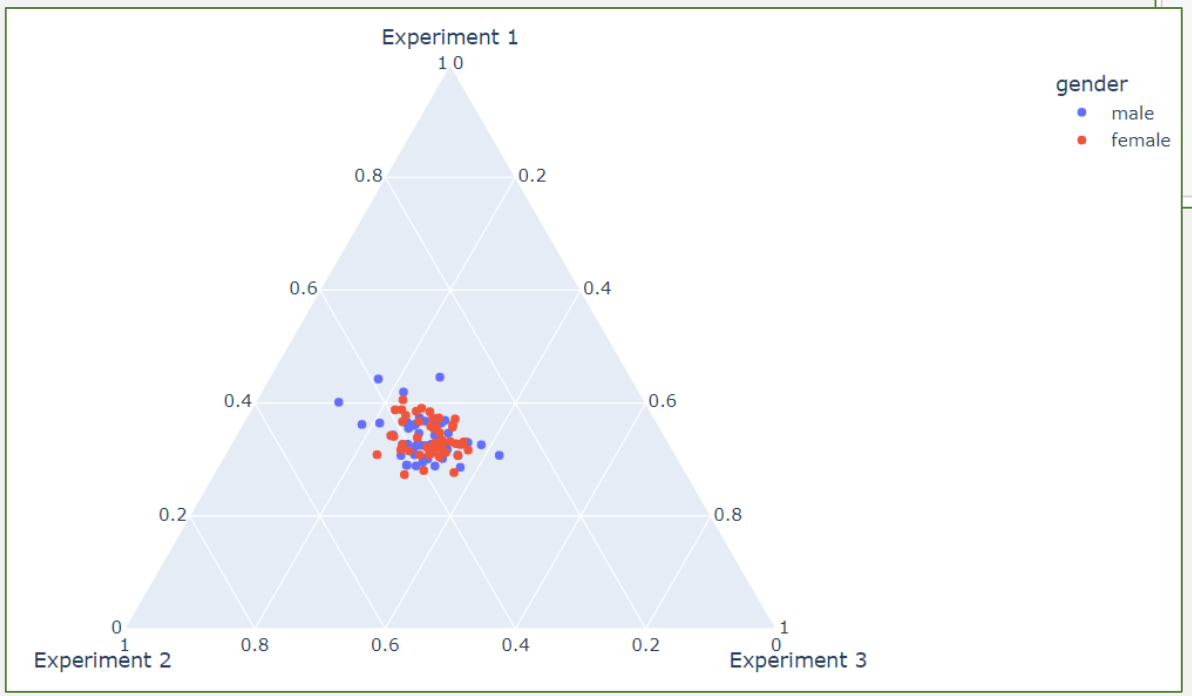


- **line_close**
  - 將線連起來
  - 不會因沒有資料而斷掉

Ternary Plot

```
# Load dataset
df_exp = px.data.experiment()
df_exp.head()
```

```
# Create a ternary plot
px.scatter_ternary(df_exp,
                   a = "experiment_1",
                   b = "experiment_2",
                   c = 'experiment_3',
                   hover_name = "group",
                   color = "gender",
                   labels = {'experiment_1': 'Experiment 1',
                             'experiment_2': 'Experiment 2',
                             'experiment_3': 'Experiment 3'})
```



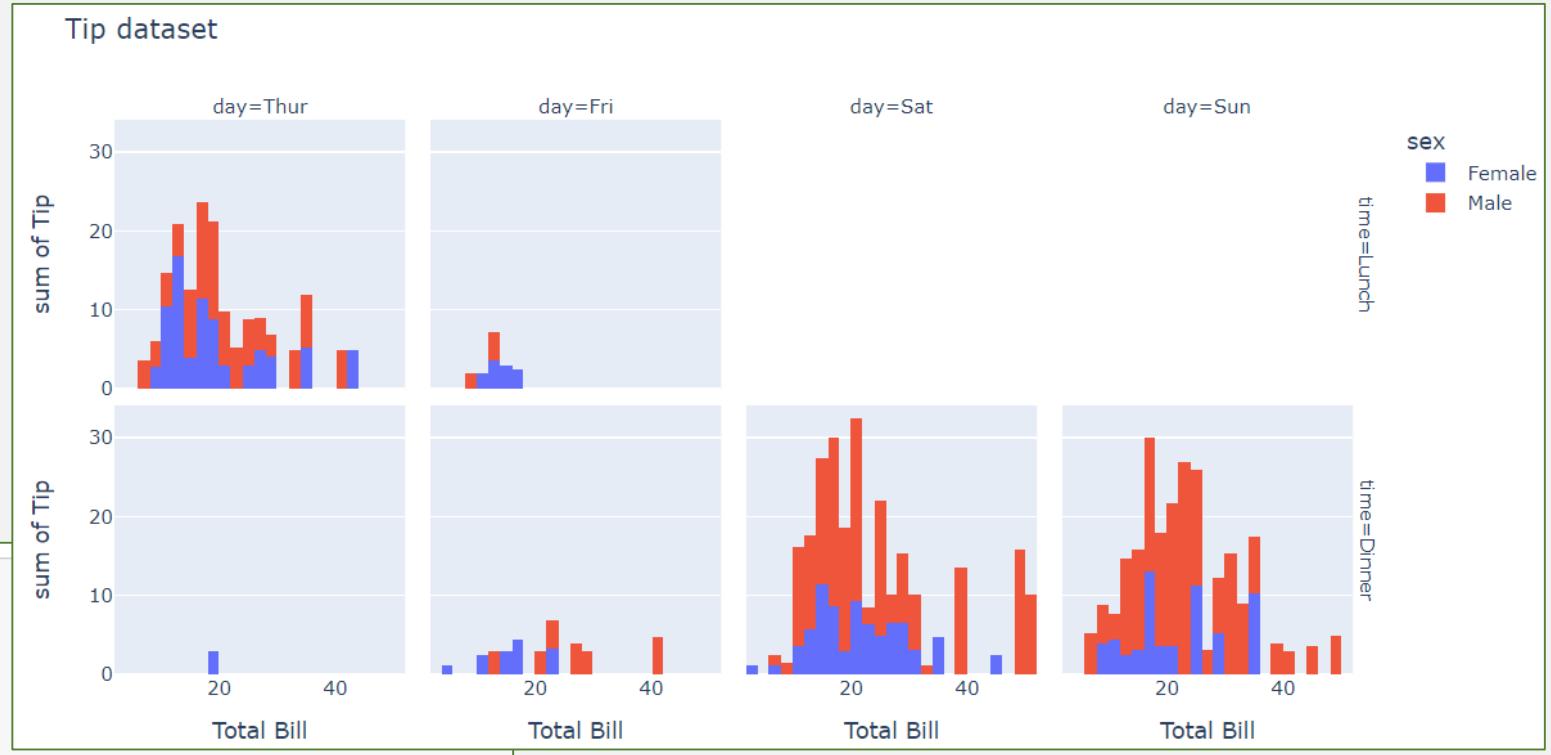
	experiment_1	experiment_2	experiment_3	gender	group
0	96.876065	93.417942	73.033193	male	control
1	87.301336	129.603395	66.056554	female	control
2	97.691312	106.187916	103.422709	male	treatment
3	102.978152	93.814682	56.995870	female	treatment
4	87.106993	107.019985	72.140292	male	control

Facets

```
px.scatter(df_tips,
           x = "total_bill",
           y = "tip",
           color = "smoker",
           facet_col = "sex",
           labels = {'total_bill': 'Total Bill',
                     'tip': 'Tip'},
           title = 'Tip dataset')
```

- **facets**
  - 具有相同軸集的多個子圖組成
  - 每個子圖是數據的一個子集

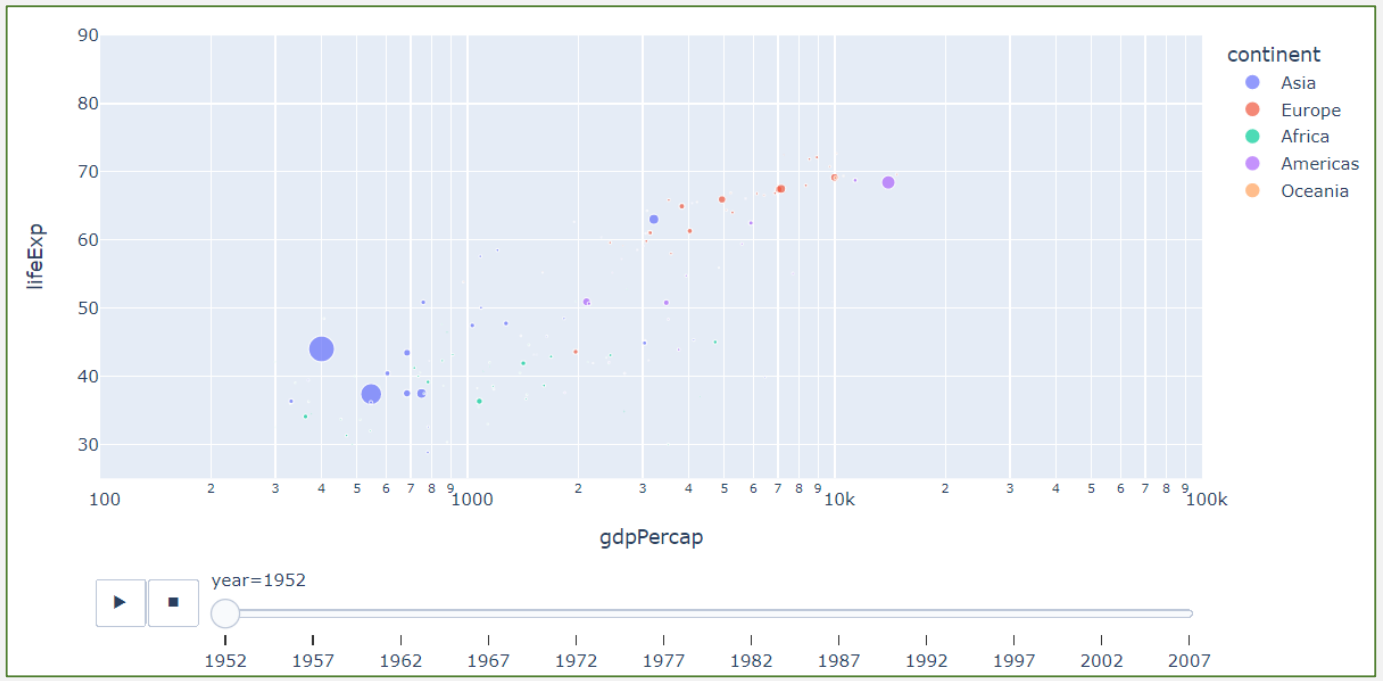
Facets




```
# Line up data in rows and columns
px.histogram(df_tips,
             x = "total_bill",
             y = "tip",
             color = "sex",
             facet_row = "time",
             facet_col = "day",
             category_orders = {"day": ["Thur", "Fri", "Sat", "Sun"],
                                "time": ["Lunch", "Dinner"]},
             labels = {'total_bill': 'Total Bill',
                       'tip': 'Tip'},
             title = 'Tip dataset')
```

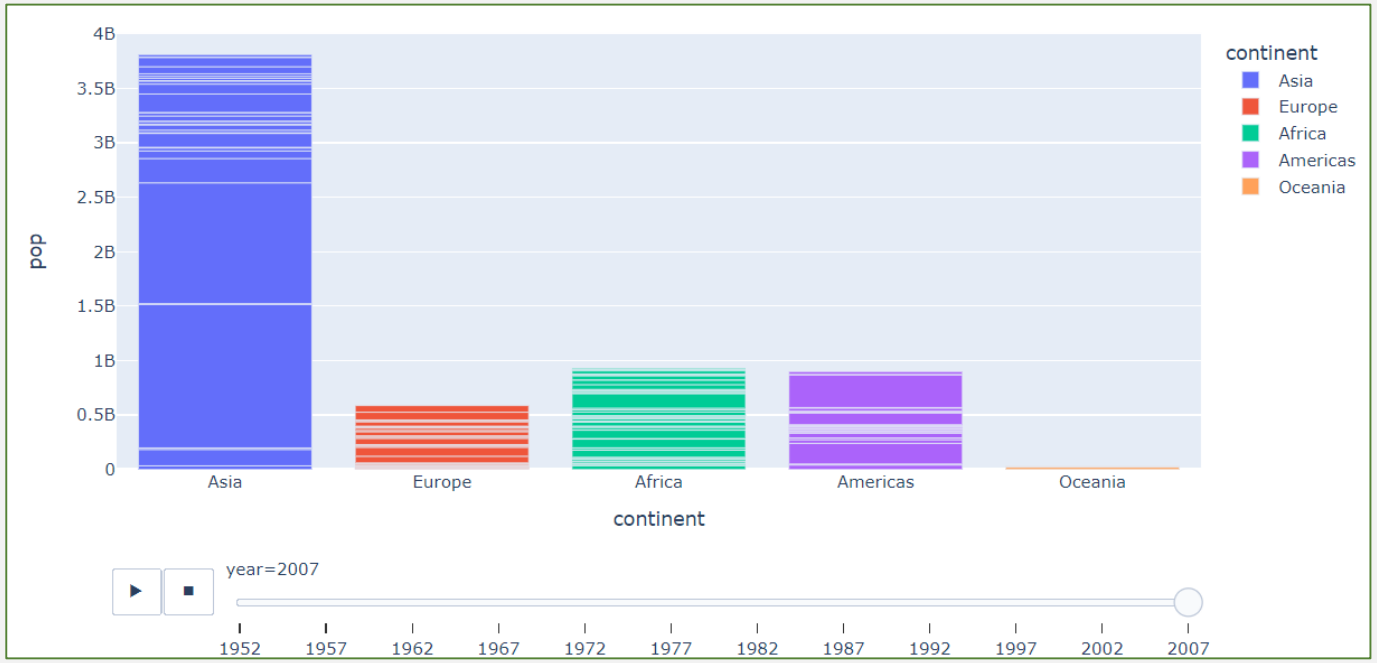
Animated Plot

```
# Create an animated scatter plot
px.scatter(df_gap,
           x = "gdpPercap",
           y = "lifeExp",
           animation_frame = "year",
           animation_group = "country",
           size = "pop",
           color = "continent",
           hover_name = "country",
           log_x = True,
           range_x = [100,100000],
           range_y = [25,90])
```



Animated Plot

```
# Create an animated bar chart
px.bar(df_gap,
       x = "continent",
       y = "pop",
       color = "continent",
       animation_frame = "year",
       animation_group = "country",
       range_y = [0,4000000000])
```



ps. 以上plotly範例參考自https://github.com/derekbanas/plotly-tutorial

Dash

- **Layout**
  - 網頁應用程式的主題、外觀及使用者介面元件
  - 與R的Shiny套件對照就是在ui.R中撰寫的程式碼

- **Callbacks**
  - 負責產生網頁應用程式的互動
  - 與R的Shiny套件對照就是在server.R中撰寫的程式碼

ps. 以下dash範例參考自官網：https://dash.plotly.com

Layout - 1

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

# Dash ships with supercharged components for interactive user interfaces
# Write HTML or use an HTML templating engine
# dcc: dash core components
from dash import dcc, html

app = dash.Dash(__name__)

# Color settings
colors = {
    'background': '#2c3e50',
    'text': '#3498db'
}
```

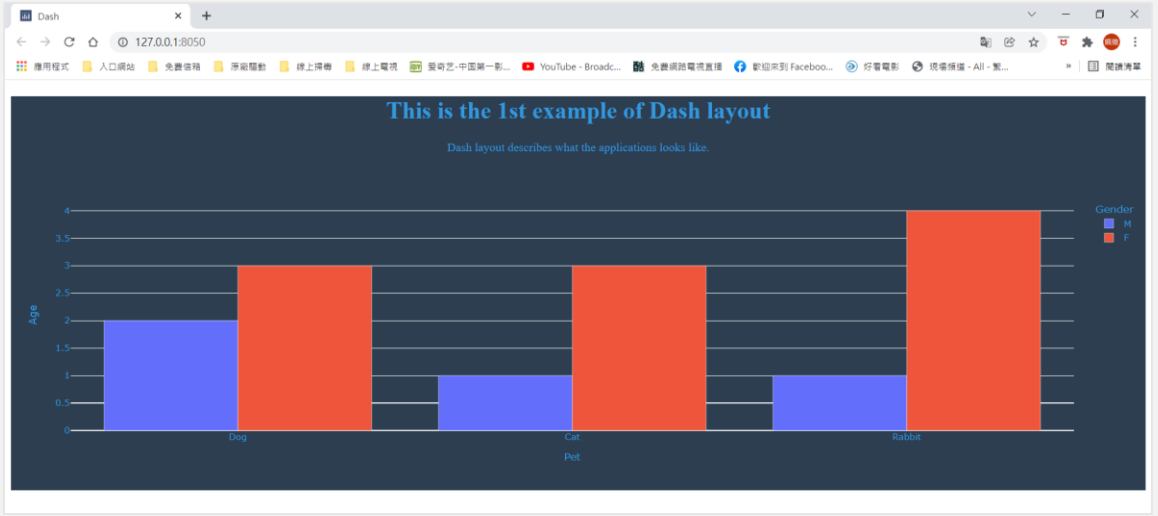
```
# Create an example data
df = pd.DataFrame({
    'Pet': ['Dog', 'Cat', 'Rabbit', 'Dog', 'Cat', 'Rabbit'],
    'Age': [2, 1, 1, 3, 3, 4],
    'Gender': ['M', 'M', 'M', 'F', 'F', 'F']
})

# Create a bar chart
fig = px.bar(df,
             x='Pet',
             y='Age',
             color='Gender',
             barmode='group'
             )

fig.update_layout(
    plot_bgcolor=colors['background'],
    paper_bgcolor=colors['background'],
    font_color=colors['text']
)
```

- 若為**Dash v1.x**，請使用以下方法載入套件
  - import dash_core_components as dcc
  - import dash_html_components as html

Layout - 1

```
# Layout settings
app.layout = html.Div(
    style={'backgroundColor': colors['background']},
    children=[
        # <h1 style='...'>This is the 1st example of Dash layout</h1>
        html.H1(
            children='This is the 1st example of Dash layout',
            style={
                'textAlign': 'center',  # HTML: text-align
                'color': colors['text']
            }
        ),

        html.Div(
            children='Dash layout describes what the applications looks like.',
            style={
                'textAlign': 'center',
                'color': colors['text']
            }
        ),

        dcc.Graph(
            id='layout-example-graph-1',
            figure=fig
        )
    ]
)
```

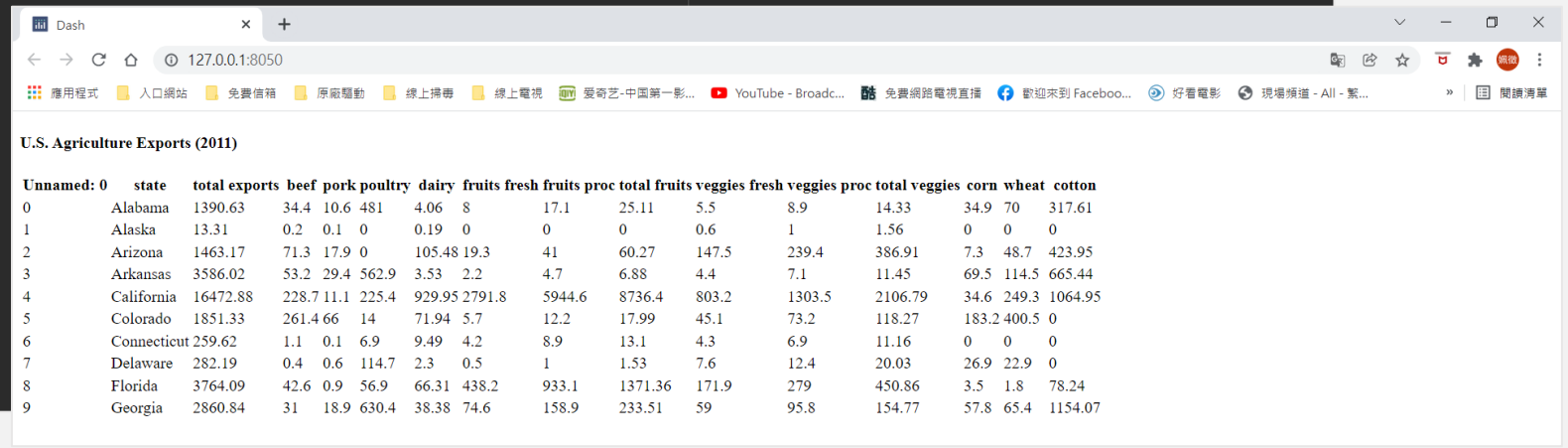
```
if __name__ == '__main__':
    app.run_server(debug=False)
```



- children可以省略
- HTML class = Dash className

Layout - 2

```python
import pandas as pd
import dash
from dash import html

df = pd.read_csv('https://gist.githubusercontent.com/chriddyp/c78bf172206ce24f77d6363a2d754b59/raw/c353e8ef842413cae56ae3920b8fd78468aa4cb2/usa-agricultural-exports-2011.csv')


def generate_table(dataframe, max_rows=10):
    return html.Table([
        html.Thead(
            html.Tr([html.Th(col) for col in dataframe.columns])
        ),
        html.Tbody([
            html.Tr([
                html.Td(dataframe.iloc[i][col]) for col in dataframe.columns
            ]) for i in range(min(len(dataframe), max_rows))
        ])
    ])


app = dash.Dash(__name__)

app.layout = html.Div([
    html.H4(children='U.S. Agriculture Exports (2011)'),
    generate_table(df)
])

if __name__ == '__main__':
    app.run_server(debug=False)
```

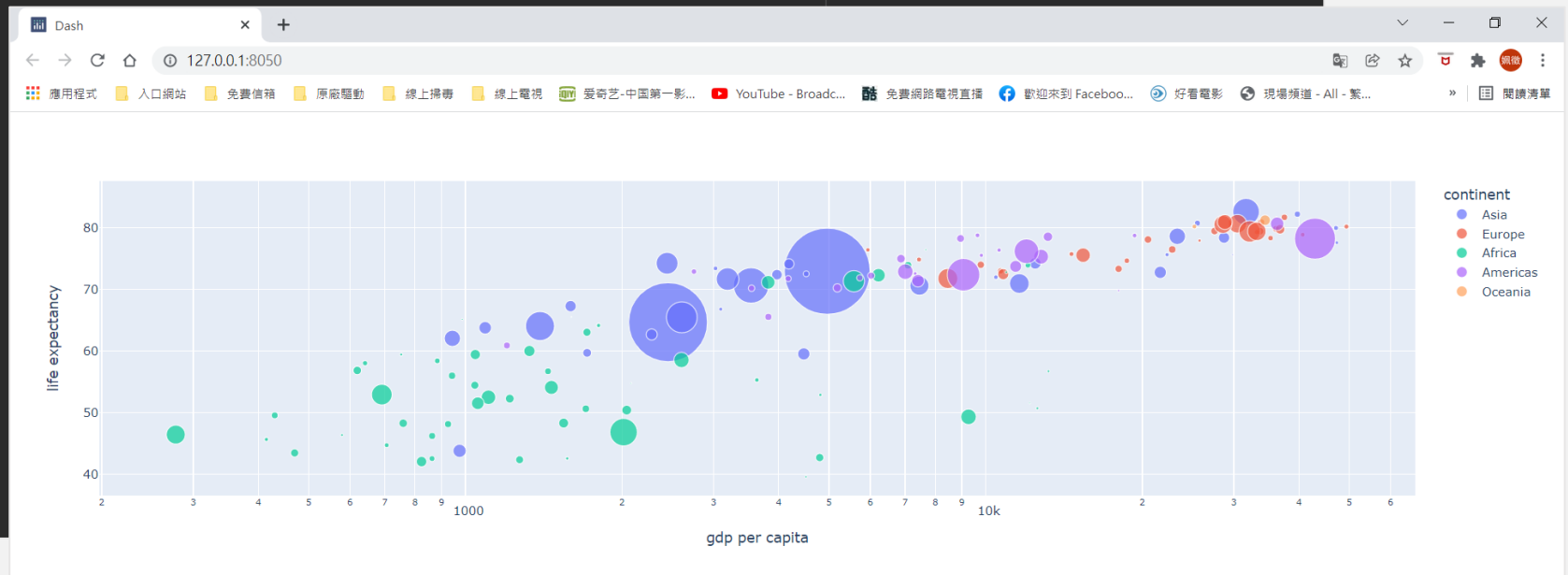


U.S. Agriculture Exports (2011)

Unnamed: 0	state	total exports	beef	pork	poultry	dairy	fruits fresh	fruits proc	total fruits	veggies fresh	veggies proc	total veggies	corn	wheat	cotton
0	Alabama	1390.63	34.4	10.6	481	4.06	8	17.1	25.11	5.5	8.9	14.33	34.9	70	317.61
1	Alaska	13.31	0.2	0.1	0	0.19	0	0	0	0.6	1	1.56	0	0	0
2	Arizona	1463.17	71.3	17.9	0	105.48	19.3	41	60.27	147.5	239.4	386.91	7.3	48.7	423.95
3	Arkansas	3586.02	53.2	29.4	562.9	3.53	2.2	4.7	6.88	4.4	7.1	11.45	69.5	114.5	665.44
4	California	16472.88	228.7	11.1	225.4	929.95	2791.8	5944.6	8736.4	803.2	1303.5	2106.79	34.6	249.3	1064.95
5	Colorado	1851.33	261.4	66	14	71.94	5.7	12.2	17.99	45.1	73.2	118.27	183.2	400.5	0
6	Connecticut	259.62	1.1	0.1	6.9	9.49	4.2	8.9	13.1	4.3	6.9	11.16	0	0	0
7	Delaware	282.19	0.4	0.6	114.7	2.3	0.5	1	1.53	7.6	12.4	20.03	26.9	22.9	0
8	Florida	3764.09	42.6	0.9	56.9	66.31	438.2	933.1	1371.36	171.9	279	450.86	3.5	1.8	78.24
9	Georgia	2860.84	31	18.9	630.4	38.38	74.6	158.9	233.51	59	95.8	154.77	57.8	65.4	1154.07

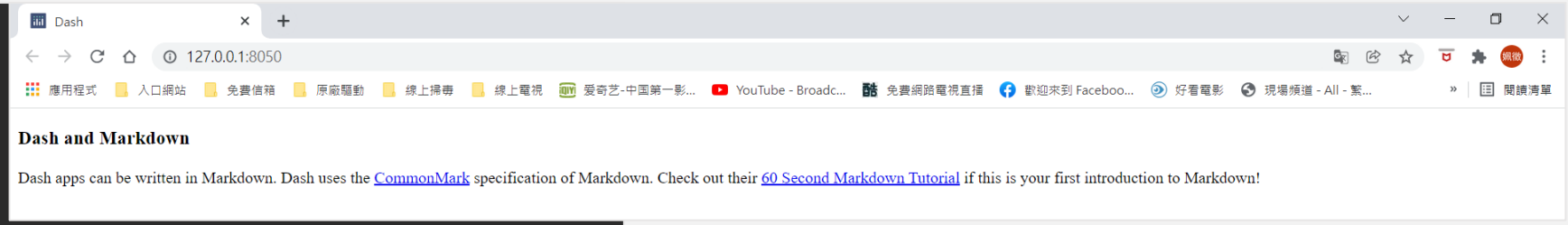
Layout - 3

```python
import dash
from dash import dcc, html
import plotly.express as px
import pandas as pd


app = dash.Dash(__name__)

df = pd.read_csv('https://gist.githubusercontent.com/chriddyp/5d1ea79569ed194d432e56108a04d188/raw/a9f9e8076b837d541398e999dcbac2b2826a81f8/gdp-life-exp-2007.csv')

fig = px.scatter(df,
                 x='gdp per capita',
                 y='life expectancy',
                 size='population',
                 color='continent',
                 hover_name='country',
                 log_x=True,
                 size_max=60)

app.layout = html.Div([
    dcc.Graph(
        id='life-exp-vs-gdp',
        figure=fig
    )
])

if __name__ == '__main__':
    app.run_server(debug=False)
```



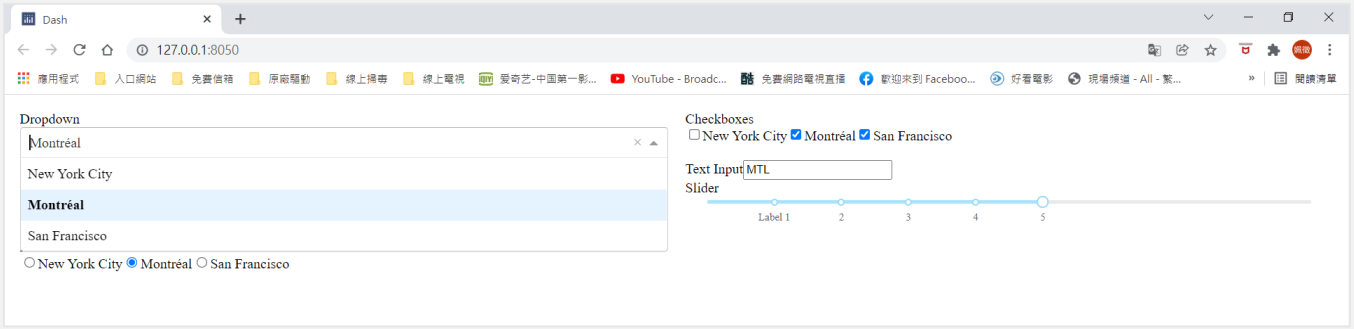
Layout - 4

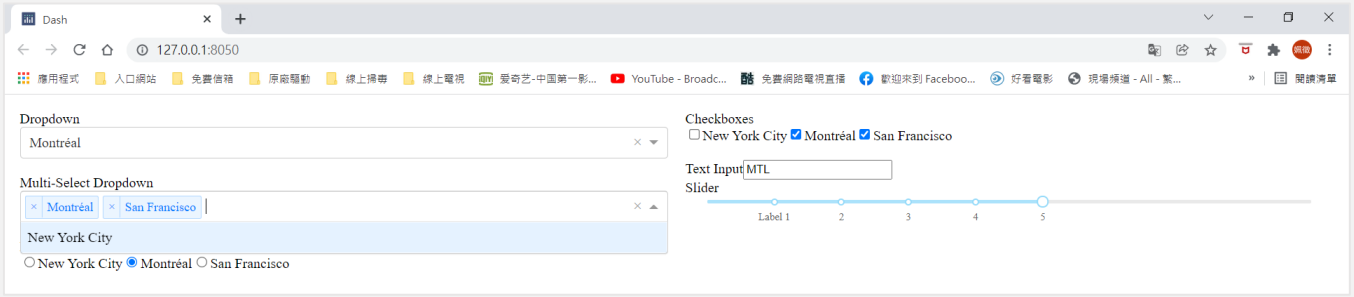
```python
import dash
from dash import dcc, html

app = dash.Dash(__name__)

markdown_text = '''
### Dash and Markdown

Dash apps can be written in Markdown.
Dash uses the [CommonMark](http://commonmark.org/)
specification of Markdown.
Check out their [60 Second Markdown Tutorial](http://commonmark.org/help/)
if this is your first introduction to Markdown!
'''

app.layout = html.Div([
    dcc.Markdown(children=markdown_text)
])

if __name__ == '__main__':
    app.run_server(debug=False)
```



- **dash core components (dcc)**
  - Graph
  - Markdown
  - Slider、RangeSlider
  - Input、Textarea
  - Checklist、RadioItems
  - DatePickerSingle、DatePickerRange
  - Upload、Download

Layout - 5



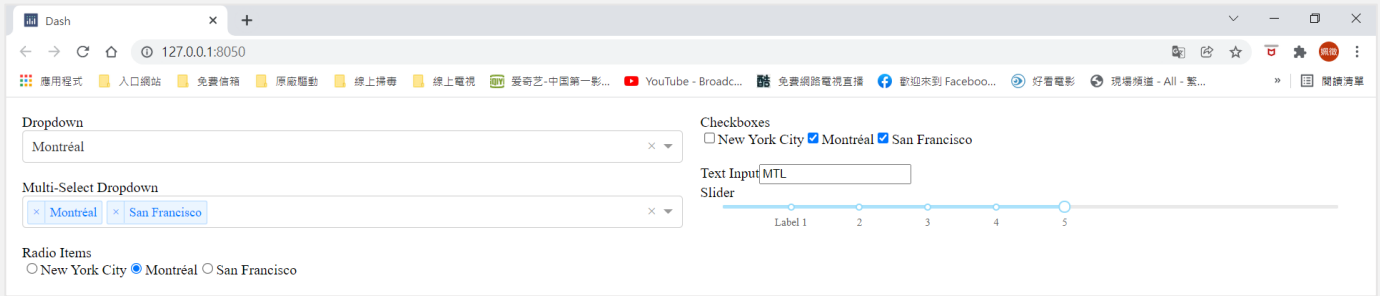



```python
import dash
from dash import dcc, html

app = dash.Dash(__name__)

app.layout = html.Div([
    html.Div(children=[
        html.Label('Dropdown'),
        dcc.Dropdown(
            options=[
                {'label': 'New York City', 'value': 'NYC'},
                {'label': u'Montréal', 'value': 'MTL'},
                {'label': 'San Francisco', 'value': 'SF'}
            ],
            value='MTL'
        ),

        html.Br(),
        html.Label('Multi-Select Dropdown'),
        dcc.Dropdown(
            options=[
                {'label': 'New York City', 'value': 'NYC'},
                {'label': u'Montréal', 'value': 'MTL'},
                {'label': 'San Francisco', 'value': 'SF'}
            ],
            value=['MTL', 'SF'],
            multi=True
        ),
```

Layout - 5

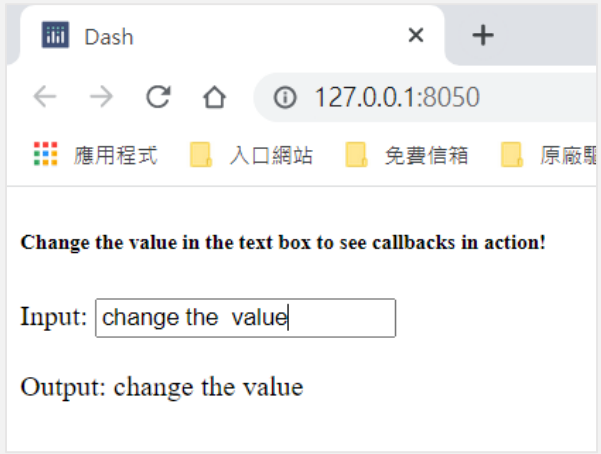
```
        html.Br(),
        html.Label('Radio Items'),
        dcc.RadioItems(
            options=[
                {'label': 'New York City', 'value': 'NYC'},
                {'label': u'Montréal', 'value': 'MTL'},
                {'label': 'San Francisco', 'value': 'SF'}
            ],
            value='MTL'
        ),
    ], style={'padding': 10, 'flex': 1}),

    html.Div(children=[
        html.Label('Checkboxes'),
        dcc.Checklist(
            options=[
                {'label': 'New York City', 'value': 'NYC'},
                {'label': u'Montréal', 'value': 'MTL'},
                {'label': 'San Francisco', 'value': 'SF'}
            ],
            value=['MTL', 'SF']
        ),

        html.Br(),
        html.Label('Text Input'),
        dcc.Input(value='MTL', type='text'),
```

```
        html.Br(),
        html.Label('Slider'),
        dcc.Slider(
            min=0,
            max=9,
            marks={i: 'Label {}'.format(i) if i == 1 else str(i) for i in range(1, 6)},
            value=5,
        ),
    ], style={'padding': 10, 'flex': 1})
], style={'display': 'flex', 'flex-direction': 'row'})

if __name__ == '__main__':
    app.run_server(debug=False)
```



Callbacks - 1

```python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H6("Change the value in the text box to see callbacks in action!"),
    html.Div([
        "Input: ",
        dcc.Input(id='my-input', value='initial value', type='text')
    ]),
    html.Br(),
    html.Div(id='my-output'),

])


@app.callback(
    Output(component_id='my-output', component_property='children'),
    Input(component_id='my-input', component_property='value')
)
def update_output_div(input_value):
    return 'Output: {}'.format(input_value)


if __name__ == '__main__':
    app.run_server(debug=False)
```



- **dash.dependencies.Input**
  - Be used in the callback definitions
- **dcc.Input**
  - An actual component

Callbacks - 2

```
import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminderDataFiveYear.csv')

app = dash.Dash(__name__)

app.layout = html.Div([
    dcc.Graph(id='graph-with-slider'),
    dcc.Slider(
        id='year-slider',
        min=df['year'].min(),
        max=df['year'].max(),
        value=df['year'].min(),
        marks={str(year): str(year) for year in df['year'].unique()},
        step=None
    )
])
```

Callbacks - 2

```python
@app.callback(
    Output('graph-with-slider', 'figure'),
    Input('year-slider', 'value'))
def update_figure(selected_year):
    filtered_df = df[df.year == selected_year]

    fig = px.scatter(filtered_df,
                     x='gdpPercap',
                     y='lifeExp',
                     size='pop',
                     color='continent',
                     hover_name='country',
                     log_x=True,
                     size_max=55)

    fig.update_layout(transition_duration=500)

    return fig


if __name__ == '__main__':
    app.run_server(debug=False)
```

- **原始數據**
  - 在初始化時便載入 (全域變數)
  - 不會因為callbacks而被修改

Callbacks - 3

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

import pandas as pd

app = dash.Dash(__name__)

df = pd.read_csv('https://plotly.github.io/datasets/country_indicators.csv')

available_indicators = df['Indicator Name'].unique()

app.layout = html.Div([
    html.Div([

        html.Div([
            dcc.Dropdown(
                id='xaxis-column',
                options=[{'label': i, 'value': i} for i in available_indicators],
                value='Fertility rate, total (births per woman)'
            ),
            dcc.RadioItems(
                id='xaxis-type',
                options=[{'label': i, 'value': i} for i in ['Linear', 'Log']],
                value='Linear',
                labelStyle={'display': 'inline-block'}
            )
        ], style={'width': '48%', 'display': 'inline-block'}),
```

```
        html.Div([
            dcc.Dropdown(
                id='yaxis-column',
                options=[{'label': i, 'value': i} for i in available_indicators],
                value='Life expectancy at birth, total (years)'
            ),
            dcc.RadioItems(
                id='yaxis-type',
                options=[{'label': i, 'value': i} for i in ['Linear', 'Log']],
                value='Linear',
                labelStyle={'display': 'inline-block'}
            )
        ], style={'width': '48%', 'float': 'right', 'display': 'inline-block'})
    ]),

    dcc.Graph(id='indicator-graphic'),

    dcc.Slider(
        id='year--slider',
        min=df['Year'].min(),
        max=df['Year'].max(),
        value=df['Year'].max(),
        marks={str(year): str(year) for year in df['Year'].unique()},
        step=None
    )
])
```

Callbacks - 3

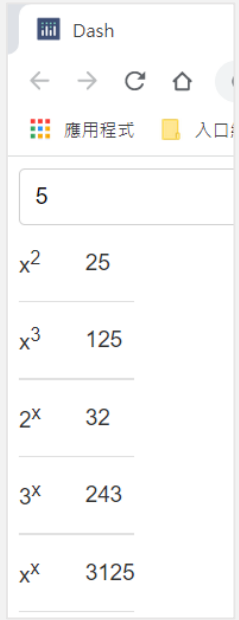
```python
@app.callback(
    Output('indicator-graphic', 'figure'),
    Input('xaxis-column', 'value'),
    Input('yaxis-column', 'value'),
    Input('xaxis-type', 'value'),
    Input('yaxis-type', 'value'),
    Input('year--slider', 'value'))
def update_graph(xaxis_column_name, yaxis_column_name,
                 xaxis_type, yaxis_type,
                 year_value):
    dff = df[df['Year'] == year_value]

    fig = px.scatter(x=dff[dff['Indicator Name'] == xaxis_column_name]['Value'],
                     y=dff[dff['Indicator Name'] == yaxis_column_name]['Value'],
                     hover_name=dff[dff['Indicator Name'] == yaxis_column_name]['Country Name'])

    fig.update_layout(margin={'l': 40, 'b': 40, 't': 10, 'r': 0}, hovermode='closest')

    fig.update_xaxes(title=xaxis_column_name,
                     type='linear' if xaxis_type == 'Linear' else 'log')

    fig.update_yaxes(title=yaxis_column_name,
                     type='linear' if yaxis_type == 'Linear' else 'log')

    return fig


if __name__ == '__main__':
    app.run_server(debug=False)
```

- **多個input、一個output**
  - 任何一個input發生改變都會執行callbacks
  - 即使只有更改一個input，Dash也會蒐集所有input的狀態傳遞給callbacks

Callbacks - 4

```python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    dcc.Input(
        id='num-multi',
        type='number',
        value=5
    ),
    html.Table([
        html.Tr([html.Td(['x', html.Sup(2)]), html.Td(id='square')]),
        html.Tr([html.Td(['x', html.Sup(3)]), html.Td(id='cube')]),
        html.Tr([html.Td([2, html.Sup('x')]), html.Td(id='twos')]),
        html.Tr([html.Td([3, html.Sup('x')]), html.Td(id='threes')]),
        html.Tr([html.Td(['x', html.Sup('x')]), html.Td(id='x^x')]),
    ]),
])
```

```python
@app.callback(
    Output('square', 'children'),
    Output('cube', 'children'),
    Output('twos', 'children'),
    Output('threes', 'children'),
    Output('x^x', 'children'),
    Input('num-multi', 'value'))
def callback_a(x):
    return x**2, x**3, 2**x, 3**x, x**x


if __name__ == '__main__':
    app.run_server(debug=False)
```

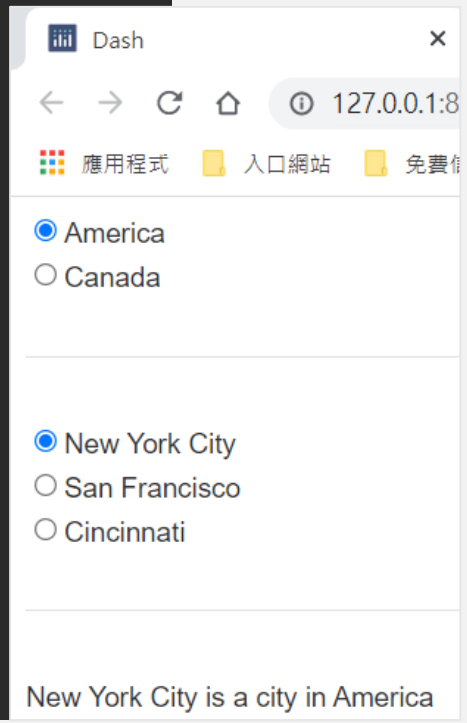


- **小提醒 (合併輸出並不總是一個好主意)**
  - 若輸出有部分(非全部)一樣輸入，將它們分開可避免不必要的更新
  - 具有相同輸入但它們對輸入執行不同計算，分開可以使它們並行執行

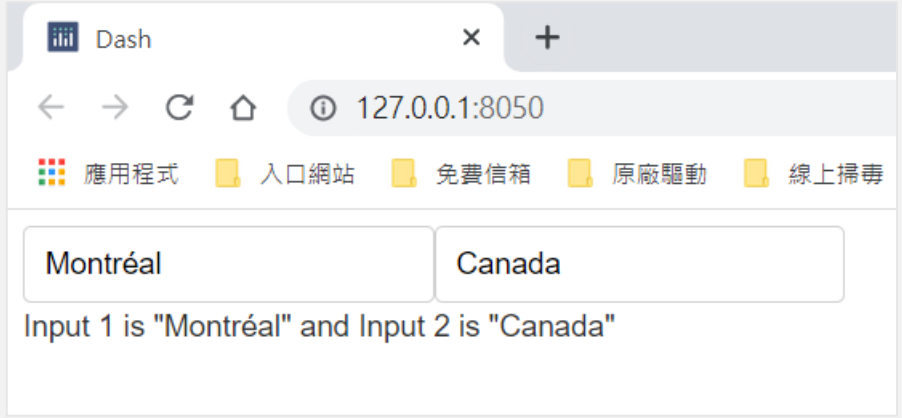
Callbacks - 5

```python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwqP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

all_options = {
    'America': ['New York City', 'San Francisco', 'Cincinnati'],
    'Canada': [u'Montréal', 'Toronto', 'Ottawa']
}
app.layout = html.Div([
    dcc.RadioItems(
        id='countries-radio',
        options=[{'label': k, 'value': k} for k in all_options.keys()],
        value='America'
    ),

    html.Hr(),

    dcc.RadioItems(id='cities-radio'),

    html.Hr(),

    html.Div(id='display-selected-values')
])
```

```python
@app.callback(
    Output('cities-radio', 'options'),
    Input('countries-radio', 'value'))
def set_cities_options(selected_country):
    return [{'label': i, 'value': i} for i in all_options[selected_country]]


@app.callback(
    Output('cities-radio', 'value'),
    Input('cities-radio', 'options'))
def set_cities_value(available_options):
    return available_options[0]['value']


@app.callback(
    Output('display-selected-values', 'children'),
    Input('countries-radio', 'value'),
    Input('cities-radio', 'value'))
def set_display_children(selected_country, selected_city):
    return u'{} is a city in {}'.format(
        selected_city, selected_country,
    )


if __name__ == '__main__':
    app.run_server(debug=False)
```

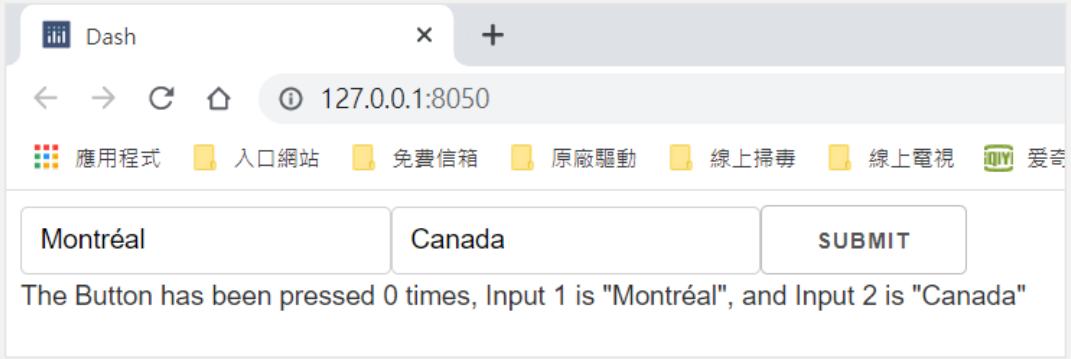


Callbacks - 6




```python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output

external_stylesheets = ["https://codepen.io/chriddyp/pen/bWLwgP.css"]

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    dcc.Input(id="input-1", type="text", value="Montréal"),
    dcc.Input(id="input-2", type="text", value="Canada"),
    html.Div(id="number-output"),
])


@app.callback(
    Output("number-output", "children"),
    Input("input-1", "value"),
    Input("input-2", "value"),
)
def update_output(input1, input2):
    return u'Input 1 is "{}" and Input 2 is "{}"'.format(input1, input2)


if __name__ == "__main__":
    app.run_server(debug=False)
```

Callbacks - 7



- **Input**
  - 更改描述的任何屬性都會觸發callbacks
- **State**
  - 可在不觸發callbacks的情況傳遞額外的值

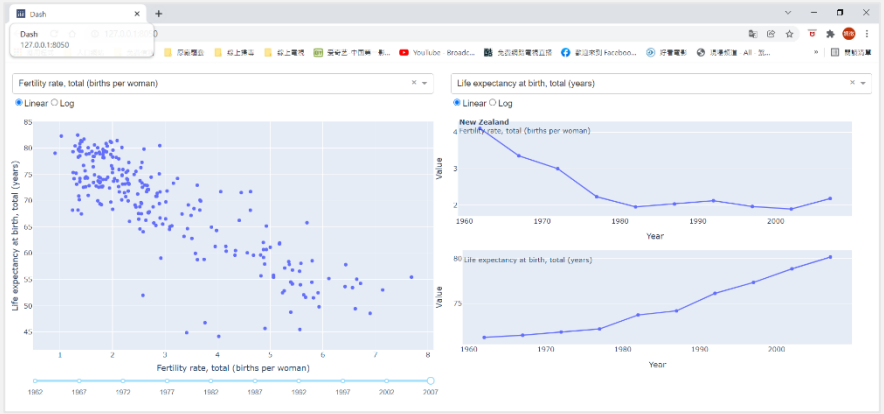
```python
import dash
from dash import dcc, html
from dash.dependencies import Input, Output, State

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

app.layout = html.Div([
    dcc.Input(id='input-1-state', type='text', value='Montréal'),
    dcc.Input(id='input-2-state', type='text', value='Canada'),
    html.Button(id='submit-button-state', n_clicks=0, children='Submit'),
    html.Div(id='output-state')
])


@app.callback(Output('output-state', 'children'),
              Input('submit-button-state', 'n_clicks'),
              State('input-1-state', 'value'),
              State('input-2-state', 'value'))
def update_output(n_clicks, input1, input2):
    return u'''
        The Button has been pressed {} times,
        Input 1 is "{}",
        and Input 2 is "{}"
    '''.format(n_clicks, input1, input2)


if __name__ == '__main__':
    app.run_server(debug=False)
```

Interactive Graph - 1

```
import dash
from dash import dcc, html
import pandas as pd
import plotly.express as px

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

df = pd.read_csv('https://plotly.github.io/datasets/country_indicators.csv')

available_indicators = df['Indicator Name'].unique()

app.layout = html.Div([
    html.Div([

        html.Div([
            dcc.Dropdown(
                id='crossfilter-xaxis-column',
                options=[{'label': i, 'value': i} for i in available_indicators],
                value='Fertility rate, total (births per woman)'
            ),
            dcc.RadioItems(
                id='crossfilter-xaxis-type',
                options=[{'label': i, 'value': i} for i in ['Linear', 'Log']],
                value='Linear',
                labelStyle={'display': 'inline-block', 'marginTop': '5px'}
            )
        ],
        style={'width': '49%', 'display': 'inline-block'}),

        html.Div([
            dcc.Dropdown(
                id='crossfilter-yaxis-column',
                options=[{'label': i, 'value': i} for i in available_indicators],
                value='Life expectancy at birth, total (years)'
            ),
            dcc.RadioItems(
                id='crossfilter-yaxis-type',
                options=[{'label': i, 'value': i} for i in ['Linear', 'Log']],
                value='Linear',
                labelStyle={'display': 'inline-block', 'marginTop': '5px'}
            )
        ], style={'width': '49%', 'float': 'right', 'display': 'inline-block'})
    ], style={
        'padding': '10px 5px'
    }),

    html.Div([
        dcc.Graph(
            id='crossfilter-indicator-scatter',
            hoverData={'points': [{'customdata': 'Japan'}]}
        )
    ], style={'width': '49%', 'display': 'inline-block', 'padding': '0 20'}),
    html.Div([
        dcc.Graph(id='x-time-series'),
        dcc.Graph(id='y-time-series'),
    ], style={'display': 'inline-block', 'width': '49%'}),
```

Interactive Graph - 1

```
    html.Div(dcc.Slider(
        id='crossfilter-year--slider',
        min=df['Year'].min(),
        max=df['Year'].max(),
        value=df['Year'].max(),
        marks={str(year): str(year) for year in df['Year'].unique()},
        step=None
    ), style={'width': '49%', 'padding': '0px 20px 20px 20px'})
])
```




```
@app.callback(
    dash.dependencies.Output('crossfilter-indicator-scatter', 'figure'),
    [dash.dependencies.Input('crossfilter-xaxis-column', 'value'),
     dash.dependencies.Input('crossfilter-yaxis-column', 'value'),
     dash.dependencies.Input('crossfilter-xaxis-type', 'value'),
     dash.dependencies.Input('crossfilter-yaxis-type', 'value'),
     dash.dependencies.Input('crossfilter-year--slider', 'value')])
def update_graph(xaxis_column_name, yaxis_column_name,
                 xaxis_type, yaxis_type,
                 year_value):
    dff = df[df['Year'] == year_value]

    fig = px.scatter(x=dff[dff['Indicator Name'] == xaxis_column_name]['Value'],
            y=dff[dff['Indicator Name'] == yaxis_column_name]['Value'],
            hover_name=dff[dff['Indicator Name'] == yaxis_column_name]['Country Name']
            )

    fig.update_traces(customdata=dff[dff['Indicator Name'] == yaxis_column_name]['Country Name'])

    fig.update_xaxes(title=xaxis_column_name, type='linear' if xaxis_type == 'Linear' else 'log')

    fig.update_yaxes(title=yaxis_column_name, type='linear' if yaxis_type == 'Linear' else 'log')

    fig.update_layout(margin={'l': 40, 'b': 40, 't': 10, 'r': 0}, hovermode='closest')

    return fig
```

Interactive Graph - 1

```
def create_time_series(dff, axis_type, title):

    fig = px.scatter(dff, x='Year', y='Value')

    fig.update_traces(mode='lines+markers')

    fig.update_xaxes(showgrid=False)

    fig.update_yaxes(type='linear' if axis_type == 'Linear' else 'log')

    fig.add_annotation(x=0, y=0.85, xanchor='left', yanchor='bottom',
                       xref='paper', yref='paper', showarrow=False, align='left',
                       text=title)

    fig.update_layout(height=225, margin={'l': 20, 'b': 30, 'r': 10, 't': 10})

    return fig
```

```
@app.callback(
    dash.dependencies.Output('x-time-series', 'figure'),
    [dash.dependencies.Input('crossfilter-indicator-scatter', 'hoverData'),
     dash.dependencies.Input('crossfilter-xaxis-column', 'value'),
     dash.dependencies.Input('crossfilter-xaxis-type', 'value')])
def update_y_timeseries(hoverData, xaxis_column_name, axis_type):
    country_name = hoverData['points'][0]['customdata']
    dff = df[df['Country Name'] == country_name]
    dff = dff[dff['Indicator Name'] == xaxis_column_name]
    title = '<b>{}</b><br>{}'.format(country_name, xaxis_column_name)
    return create_time_series(dff, axis_type, title)


@app.callback(
    dash.dependencies.Output('y-time-series', 'figure'),
    [dash.dependencies.Input('crossfilter-indicator-scatter', 'hoverData'),
     dash.dependencies.Input('crossfilter-yaxis-column', 'value'),
     dash.dependencies.Input('crossfilter-yaxis-type', 'value')])
def update_x_timeseries(hoverData, yaxis_column_name, axis_type):
    dff = df[df['Country Name'] == hoverData['points'][0]['customdata']]
    dff = dff[dff['Indicator Name'] == yaxis_column_name]
    return create_time_series(dff, axis_type, yaxis_column_name)


if __name__ == '__main__':
    app.run_server(debug=False)
```

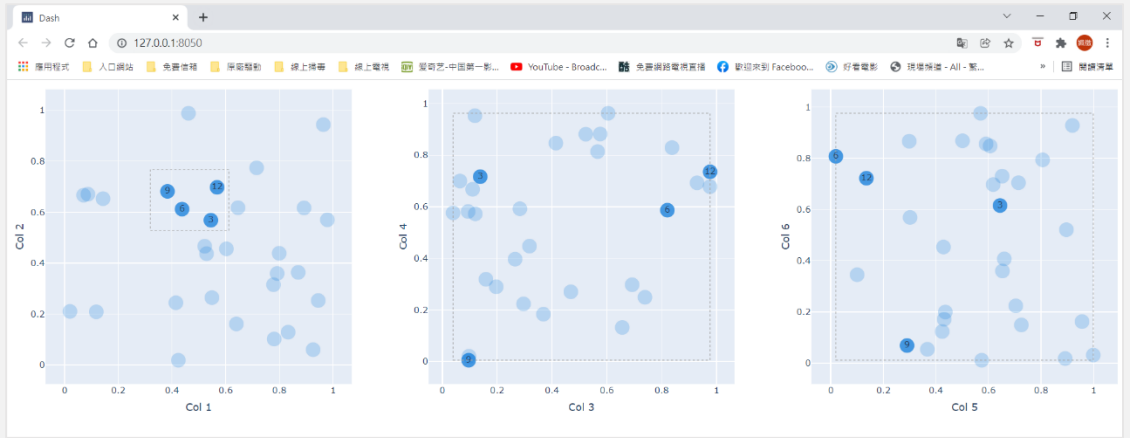
Interactive Graph - 2

```python
import dash
from dash import dcc, html
import numpy as np
import pandas as pd
from dash.dependencies import Input, Output
import plotly.express as px

external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']

app = dash.Dash(__name__, external_stylesheets=external_stylesheets)

# make a sample data frame with 6 columns
np.random.seed(0)  # no-display
df = pd.DataFrame({"Col " + str(i+1): np.random.rand(30) for i in range(6)})

app.layout = html.Div([
    html.Div(
        dcc.Graph(id='g1', config={'displayModeBar': False}),
        className='four columns'
    ),
    html.Div(
        dcc.Graph(id='g2', config={'displayModeBar': False}),
        className='four columns'
        ),
    html.Div(
        dcc.Graph(id='g3', config={'displayModeBar': False}),
        className='four columns'
    )
], className='row')
```

```python
def get_figure(df, x_col, y_col, selectedpoints, selectedpoints_local):

    if selectedpoints_local and selectedpoints_local['range']:
        ranges = selectedpoints_local['range']
        selection_bounds = {'x0': ranges['x'][0], 'x1': ranges['x'][1],
                            'y0': ranges['y'][0], 'y1': ranges['y'][1]}
    else:
        selection_bounds = {'x0': np.min(df[x_col]), 'x1': np.max(df[x_col]),
                            'y0': np.min(df[y_col]), 'y1': np.max(df[y_col])}

    fig = px.scatter(df, x=df[x_col], y=df[y_col], text=df.index)

    fig.update_traces(selectedpoints=selectedpoints,
                      customdata=df.index,
                      mode='markers+text',
                      marker={'color': 'rgba(0, 116, 217, 0.7)', 'size': 20},
                      unselected={'marker': {'opacity': 0.3}, 'textfont': {'color': 'rgba(0, 0, 0, 0)'}})

    fig.update_layout(margin={'l': 20, 'r': 0, 'b': 15, 't': 5}, dragmode='select', hovermode=False)

    fig.add_shape(dict({'type': 'rect',
                        'line': {'width': 1, 'dash': 'dot', 'color': 'darkgrey'}},
                       **selection_bounds))
    return fig
```

Interactive Graph - 2

```python
# this callback defines 3 figures
# as a function of the intersection of their 3 selections
@app.callback(
    Output('g1', 'figure'),
    Output('g2', 'figure'),
    Output('g3', 'figure'),
    Input('g1', 'selectedData'),
    Input('g2', 'selectedData'),
    Input('g3', 'selectedData')
)
def callback(selection1, selection2, selection3):
    selectedpoints = df.index
    for selected_data in [selection1, selection2, selection3]:
        if selected_data and selected_data['points']:
            selectedpoints = np.intersect1d(selectedpoints,
                [p['customdata'] for p in selected_data['points']])

    return [get_figure(df, "Col 1", "Col 2", selectedpoints, selection1),
            get_figure(df, "Col 3", "Col 4", selectedpoints, selection2),
            get_figure(df, "Col 5", "Col 6", selectedpoints, selection3)]


if __name__ == '__main__':
    app.run_server(debug=False)
```



End

最後一次的助教課囉~ 努力到現在的大家辛苦了~

恭喜還留下的人已經來到這門課的尾聲，希望大家都能收穫滿滿！