Module 10: Visualization

- Takes the data and displays it visually (graph, chart).
- We can understand the data better with visual aid. For instance, the trend of data, overview of the data, preferences in classes.
- We will be working with matplotlib module.

%matplotlib inline : renders the visual picture in Jupyter Notebook. Without this line, the matplotlib library will prompt the visual in a new window.

In [1]:

```
import pandas as pd
from pandas_datareader  import data

import matplotlib.pyplot as plt
%matplotlib inline
```

1. plot() method

In [2]:

```
all_companies = pd.read_csv("data/all_stocks_5yr.csv", index_col=["date"], parse_dates=["da
all_companies.head()
```

Out[2]:

date	open	high	low	close	volume	Name
2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

In [3]:

```
mask = all_companies.Name == "FB"
FB = all_companies[mask].copy()
```

In [4]:

```
FB.drop("Name", axis = "columns", inplace=True)
```

In [5]:

```
FB.head(3)
```

Out[5]:

date	open	high	low	close	volume
2013-02-08	28.89	29.17	28.51	28.545	37662614
2013-02-11	28.61	28.68	28.04	28.260	36979533
2013-02-12	27.67	28.16	27.10	27.370	93417215

When applying the plot() method on dataFrame, all columns are plotted on the graph.

index == X-axis, columns == Y-axis.

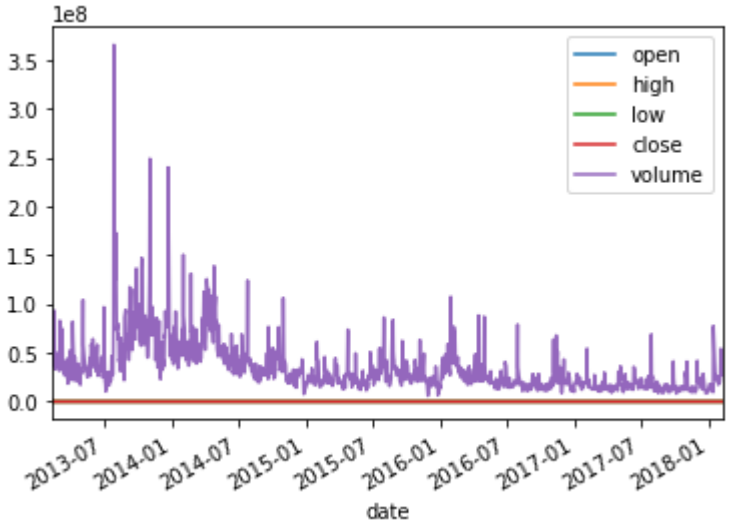
This graph is hard to understand. We can only see the Volume data being plotted. Since Volume values are a lot bigger than the other columns, the line plotted for open, high, low and close values cannot be seen.

In [6]:

```
FB.plot()
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a111930a88>
```



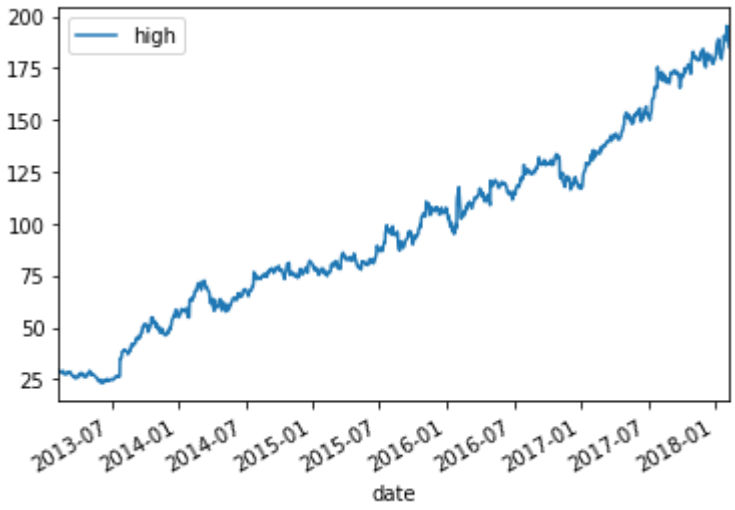
In the plot() method, it has the y parameter. This parameter allows us to choose which columns to be the y-axis.

In [7]:

```
FB.plot(y="high")
```

Out[7]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a114d59288>
```

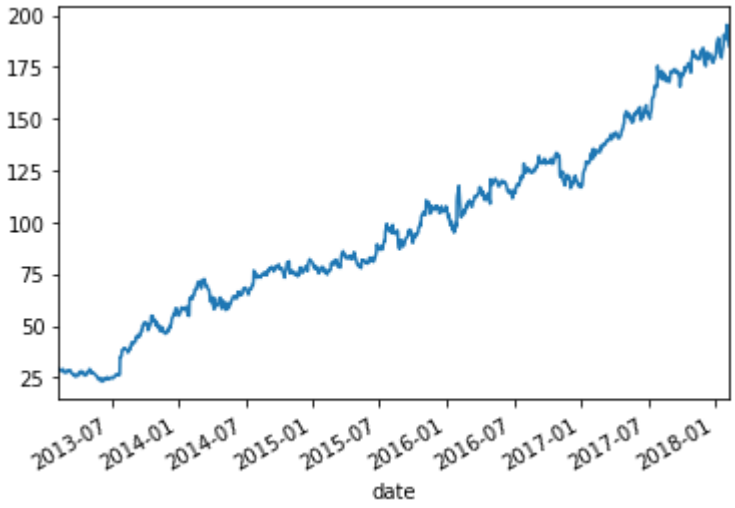


In [8]:

```
FB["high"].plot()
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a114e58288>
```



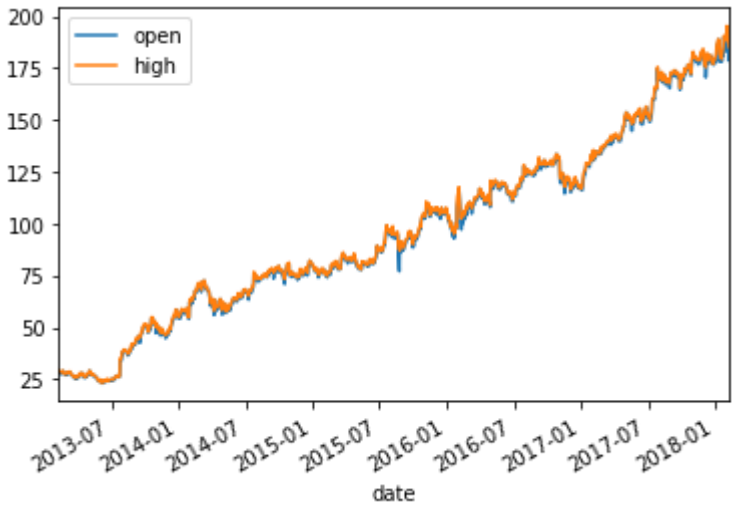
Plotting two columns in one graph

In [9]:

```
FB[["open", "high"]].plot()
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a114ee6a08>
```



2. Modifying Aesthetics

We can stylize the graph to have different looks. Below is the list of styles available for us to choose from.

In [10]:

```
plt.style.available
```

Out[10]:

```
['bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark-palette',
 'seaborn-dark',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'seaborn',
 'Solarize_Light2',
 'tableau-colorblind10',
 '_classic_test']
```

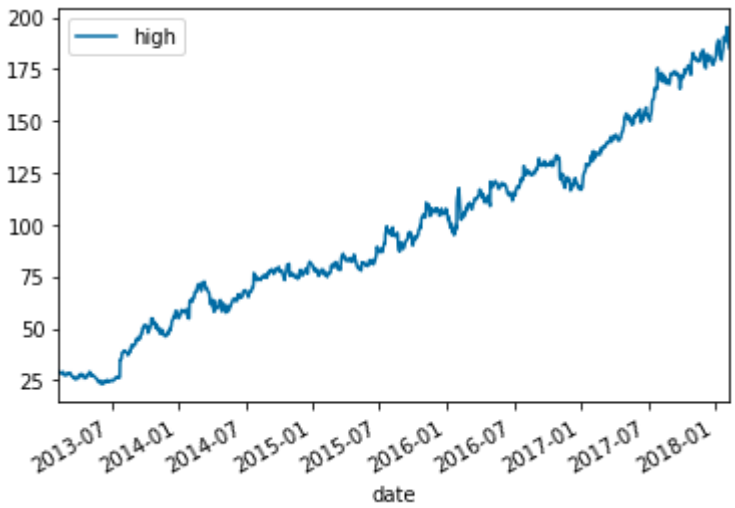
To use one of the style, we simply have to pass the style name in **plt.style.use() method**

In [11]:

```
plt.style.use("tableau-colorblind10")
FB.plot(y="high")
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a114f7e6c8>
```

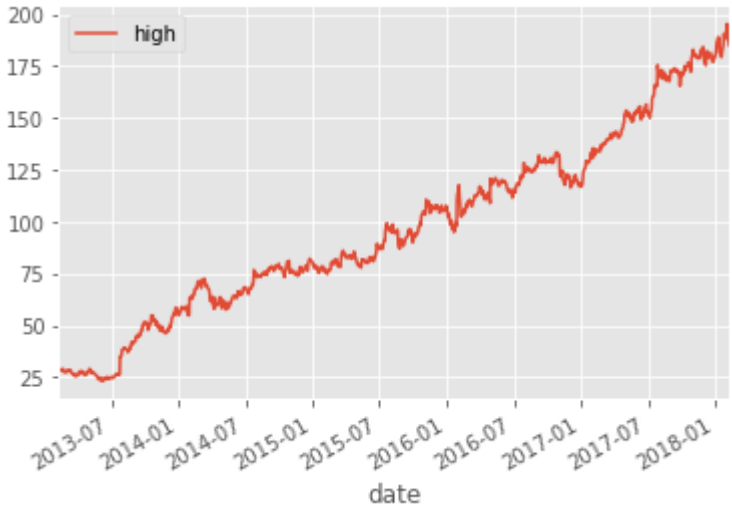


In [12]:

```
plt.style.use("ggplot")
FB.plot(y="high")
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a115009b08>
```



3. Bar Charts

A bar chart is a chart or graph that presents categorical data.

In [13]:

```
nba = pd.read_csv("data/nba.csv")
nba.head()
```

Out[13]:

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0

From Position column, we can visualize each position value in a graph.

In [14]:

```
nba["Position"].value_counts()
```

Out[14]:

```
SG    102
PF    100
PG     92
SF     85
C      78
Name: Position, dtype: int64
```

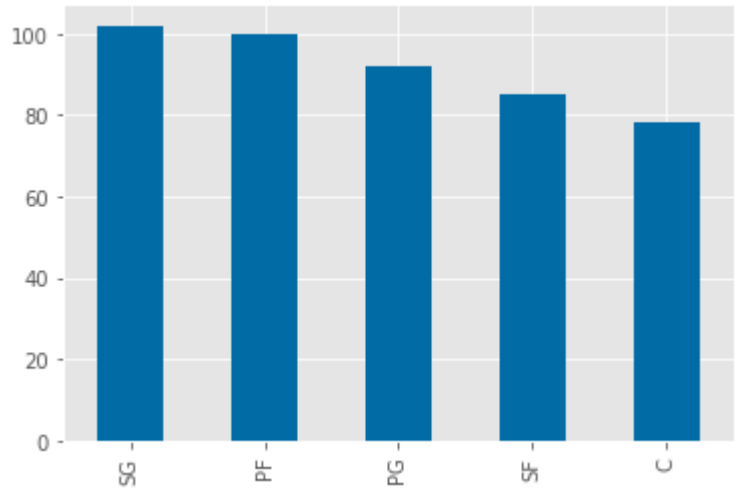
kind indicates the type of graph to display. By default, it is "line" which stands for line graph.

In [15]:

```
plt.style.use("tableau-colorblind10")
nba["Position"].value_counts().plot(kind="bar")
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1150b2988>
```



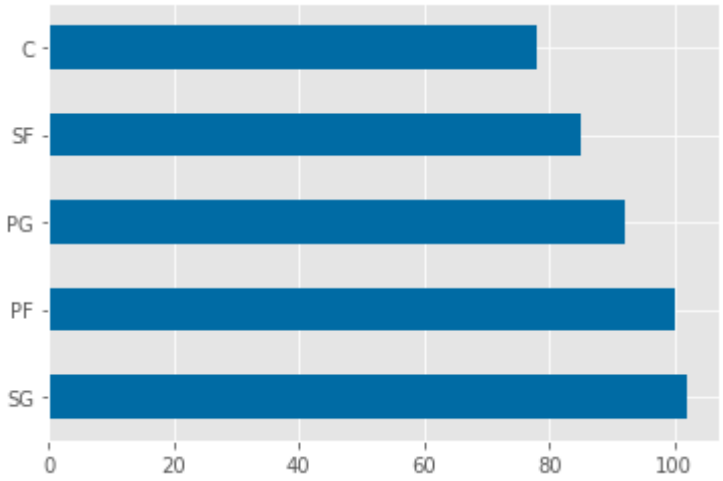
kind = "barh" stands for horizontal bar graph. This will swap the x-axis and y-axis

In [16]:

```
nba["Position"].value_counts().plot(kind="barh")
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a115124d48>
```



4. Pie Chart

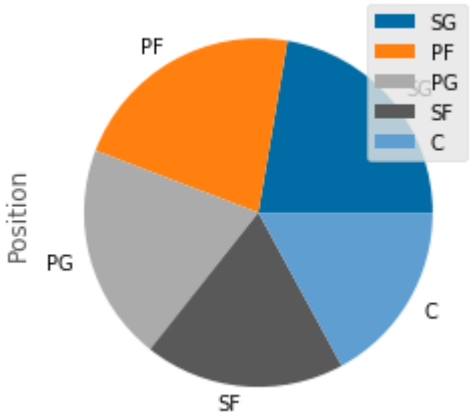
kind = "pie" represents a pie chart graph. Notice the additional parameter, legend. This parameter shows which class the colors represent.

In [17]:

```
nba["Position"].value_counts().plot(kind="pie", legend = True)
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1153e9888>
```



5. Histogram

A histogram represents the frequency distribution of continuous variables.

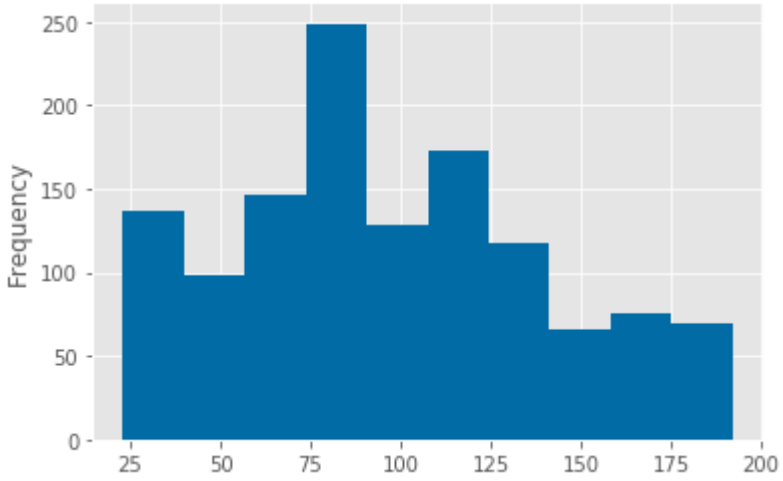
- kind = "hist" represents a Histogram graph.
- bins : Number of histogram bins to be used. The bigger the number, the smaller the bar

In [18]:

```
FB["open"].plot(kind="hist", bins = 10)
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a115457a88>
```

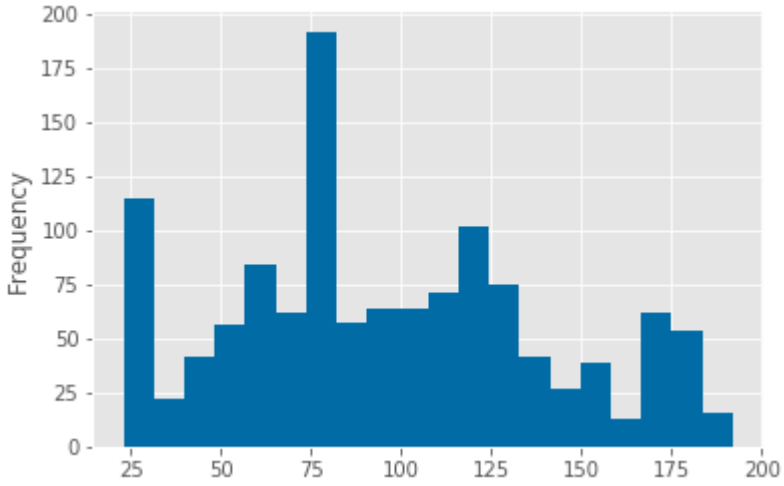


In [19]:

```
FB["open"].plot(kind="hist", bins = 20)
```

Out[19]:

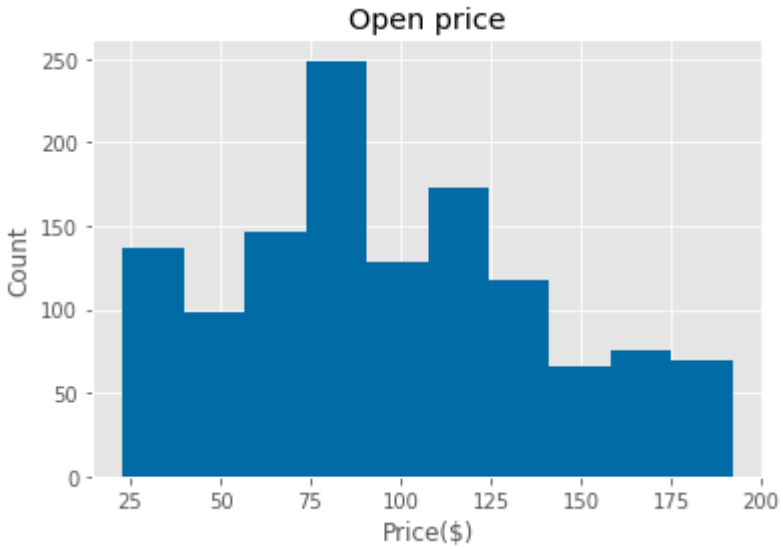
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1154e2c48>
```



Another way is to call Matplotlib method and pass the data that we want to plot.
plt.hist is used to plot a **Histogram graph**. We can add title and labels to the graph as well.

In [20]:

```
plt.hist(FB["open"])
plt.xlabel("Price($)")
plt.title("Open price")
plt.ylabel("Count")
plt.show()
```



6. Scatter plot

For this example, we will be using iris datasets. Iris datasets are well-known data to test with Machine Learning, Analysis, or Deep Learning. It is often used for testing out machine learning algorithms and visualizations.

In [21]:

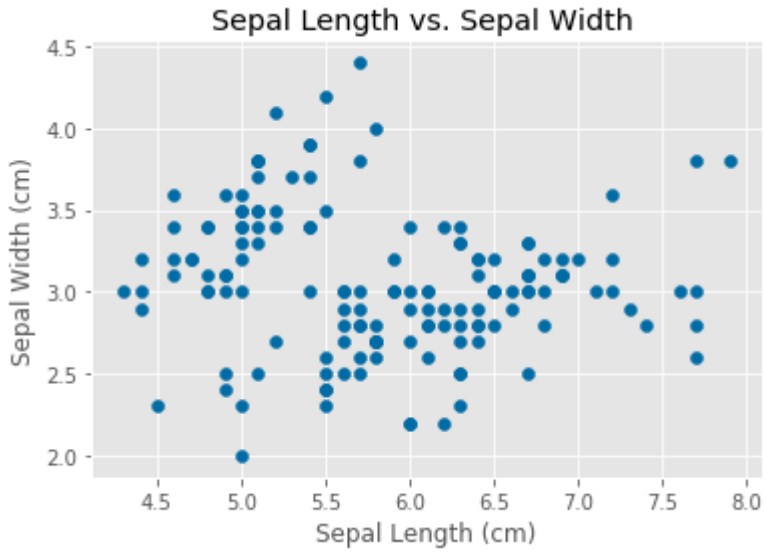
```
from sklearn import datasets

iris_data = datasets.load_iris()
iris_data.data
iris_data.target
iris_data.target_names

iris_df = pd.DataFrame(data=iris_data.data
                       , columns=iris_data.feature_names
                      ).join(
    pd.DataFrame(data=iris_data.target, columns=['Species']))

iris_df.head()
```

Out[21]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [22]:

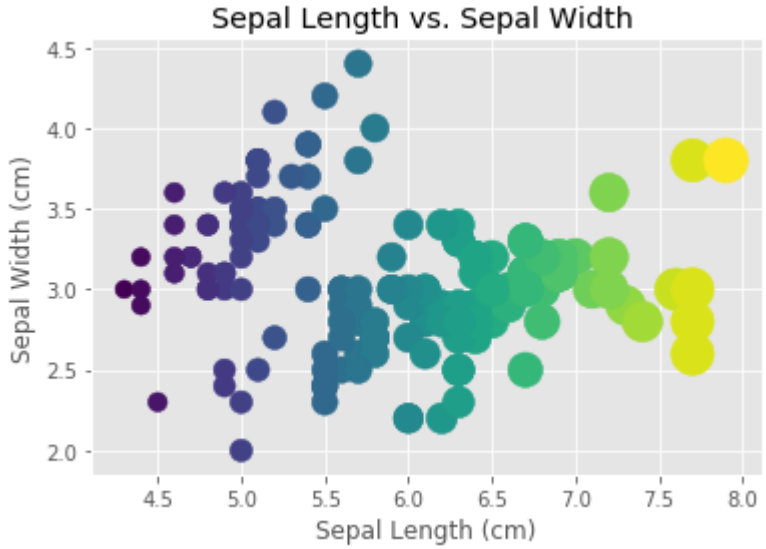
```
plt.scatter(x =iris_df["sepal length (cm)"] , y = iris_df["sepal width (cm)"] )
#add title
plt.title('Sepal Length vs. Sepal Width ')
#Name x label
plt.xlabel('Sepal Length (cm)')
#Name y label
plt.ylabel('Sepal Width (cm)')
plt.show()
```



7. Bubble plot

In [23]:

```
#Emulate ggplot
plt.style.use('ggplot')

#Create buble chart
plt.scatter(x =iris_df["sepal length (cm)"] , y = iris_df["sepal width (cm)"] ,
            s=iris_df["sepal length (cm)"]**3,
#             marker='s',
            c =iris_df["sepal length (cm)"]
           )

#add title
plt.title('Sepal Length vs. Sepal Width ')

#Name x label
plt.xlabel('Sepal Length (cm)')

#Name y label
plt.ylabel('Sepal Width (cm)')
plt.show()
```



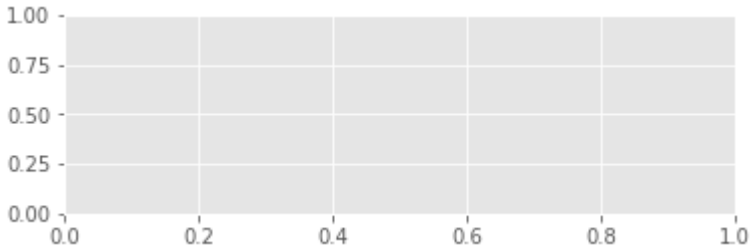
8.Subplots

In [24]:

```
import matplotlib.pyplot as plt
# plot a line, implicitly creating a subplot(111)
plt.plot([1,2,3])
# now create a subplot which represents the top plot of a grid
# with 2 rows and 1 column. Since this subplot will overlap the
# first, the plot (and its axes) previously created, will be removed
plt.subplot(211)
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1180ac088>
```



The subplot() method accepts parameters as shown:

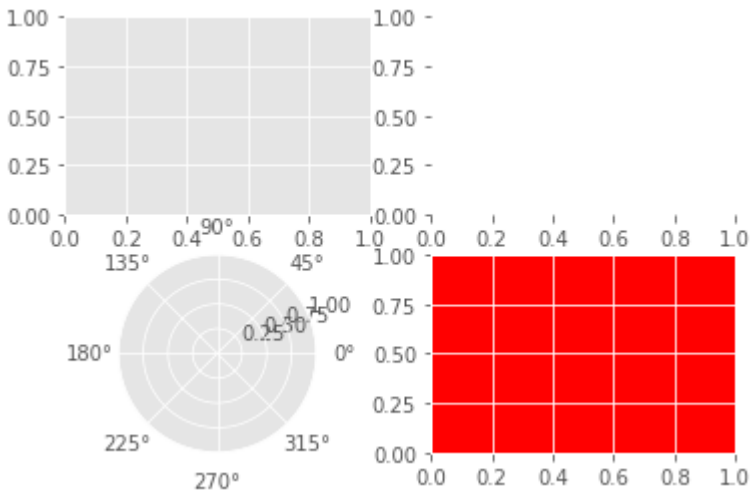
```
subplot(nrows, ncols, index,)
```

So for example, subplot(2,2,1) stands for 2 rows, 2 columns and the graph at position 1.

In [25]:

```
plt.subplot(221)

# equivalent but more general
ax1=plt.subplot(2, 2, 1)

# add a subplot with no frame
ax2=plt.subplot(222, frameon=False)

# add a polar subplot
plt.subplot(223, projection='polar')

# add a red subplot that shares the x-axis with ax1
plt.subplot(224, sharex=ax1, facecolor='red')

# delete ax2 from the figure
plt.delaxes(ax2)

# add ax2 to the figure again
plt.subplot(ax2)

plt.show()
```

```
C:\Users\Ismail\Anaconda3\lib\site-packages\ipykernel_launcher.py:4: Matplot
libDeprecationWarning: Adding an axes using the same arguments as a previous
axes currently reuses the earlier instance.  In a future version, a new inst
ance will always be created and returned.  Meanwhile, this warning can be su
ppressed, and the future behavior ensured, by passing a unique label to each
axes instance.
  after removing the cwd from sys.path.
```



9. Save graph into image file

In [26]:

```
plt.savefig('Figure1.png')
```

```
<Figure size 432x288 with 0 Axes>
```

In [27]:

```
plt.savefig("Figure 2", format="png")
```

```
<Figure size 432x288 with 0 Axes>
```

In [28]:

```
plt.savefig("Figure 3", format="rgba")
```

```
<Figure size 432x288 with 0 Axes>
```

Suppported files type

In [29]:

```
import matplotlib
figure = matplotlib.figure.Figure()
fcb = matplotlib.backends.backend_agg.FigureCanvasBase(figure)
supported_file_types = fcb.get_supported_filetypes()
supported_file_types
```

Out[29]:

```
{'ps': 'Postscript',
 'eps': 'Encapsulated Postscript',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

10. Visualize the data using Plotly

Plotly's Python graphing library makes interactive, publication-quality graphs. Simple graphs that can be created are line plots, scatter plots, area charts, and bar charts.

Plotly visualizations can be exported using cloud and Dash Framework. Dash is a framework to build web applications. Dash is built on top of Flask (Python Framework) and can be used to create a simple dashboard for visualizations.

In [30]:

```
## Firstly we have to install plotly library
!pip install plotly
```

```
Collecting plotly
  Downloading https://files.pythonhosted.org/packages/f7/05/3c32c6bc85acbd30
a18fbc3ba732fed5e48e5f8fd60d2a148877970f4a61/plotly-4.2.1-py2.py3-none-any.w
hl (https://files.pythonhosted.org/packages/f7/05/3c32c6bc85acbd30a18fbc3ba7
32fed5e48e5f8fd60d2a148877970f4a61/plotly-4.2.1-py2.py3-none-any.whl) (7.2M
B)
Collecting retrying>=1.3.3
  Downloading https://files.pythonhosted.org/packages/44/ef/beae4b4ef80902f2
2e3af073397f079c96969c69b2c7d52a57ea9ae61c9d/retrying-1.3.3.tar.gz (https://
files.pythonhosted.org/packages/44/ef/beae4b4ef80902f22e3af073397f079c96969c
69b2c7d52a57ea9ae61c9d/retrying-1.3.3.tar.gz)
Requirement already satisfied: six in c:\users\ismail\anaconda3\lib\site-pac
kages (from plotly) (1.12.0)
Building wheels for collected packages: retrying
  Building wheel for retrying (setup.py): started
  Building wheel for retrying (setup.py): finished with status 'done'
  Created wheel for retrying: filename=retrying-1.3.3-cp37-none-any.whl size
=11435 sha256=6741953024e49eeb6ba621b49d5d63451428dc59deb547a2a3efbf031452fe
dc
  Stored in directory: C:\Users\Ismail\AppData\Local\pip\Cache\wheels\d7\a9
\33\acc7b709e2a35caa7d4cae442f6fe6fbf2c43f80823d46460c
Successfully built retrying
Installing collected packages: retrying, plotly
Successfully installed plotly-4.2.1 retrying-1.3.3
```

In [31]:

```
df = pd.read_csv("data/bigmac.csv")
df.head()
```

Out[31]:

	Date	Country	Price in US Dollars
0	1/2016	Argentina	2.39
1	1/2016	Australia	3.74
2	1/2016	Brazil	3.35
3	1/2016	Britain	4.22
4	1/2016	Canada	4.14

In [32]:

```
df["Date"].unique()
```

Out[32]:

```
array(['1/2016', '7/2015', '1/2015', '7/2014', '1/2014', '7/2013',
       '1/2013', '7/2012', '1/2012', '7/2011', '7/2010', '1/2010'],
      dtype=object)
```

Visualize the top expensive place for McDonalds.

The data is from 2010 to 2016. Hence, we need to get the lowest price for each Country. After that, we group the value by Country. Then, we create variable x and y to visualize the graph. Finally, we use Plotly library and Graph Objects to visualize the graph into Bar Chart.

In [33]:

```
temp = df.groupby("Country").mean()["Price in US Dollars"].sort_values(ascending=False)
x = temp.head(10).index
y = temp.head(10).values

temp.head(10)
```

Out[33]:

```
Country
Norway          7.037500
Switzerland     6.877500
Sweden          5.955833
Venezuela       5.549000
Denmark         5.132500
Brazil          5.110000
Finland         4.978000
France          4.767000
Belgium         4.753000
Italy           4.740000
Name: Price in US Dollars, dtype: float64
```

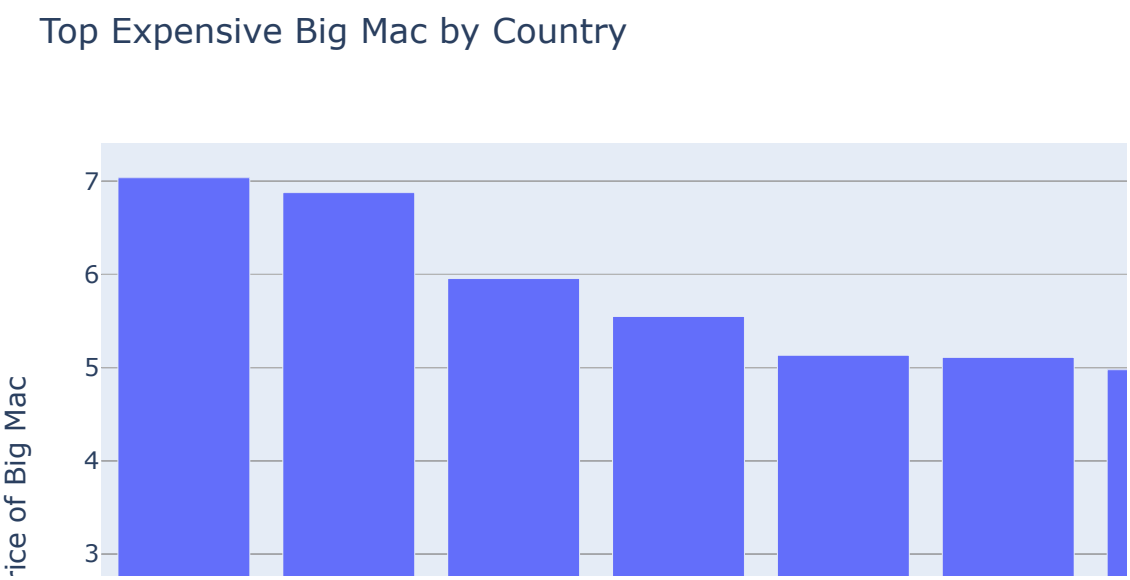
To plot a graph, we need the x and y variables. We use the groupby() method, to return the country as the index and mean price as the values. Hence, we can use .index, and .values attributes to get the x and y variables.

Below is an example of a simple graph. Firstly, we create a graph object in which the x and y values will be passed to.

Plotly has many parameters which can make the graph more beautiful and easier to read. The main components of the graph is the title, x label, and y label.

In [34]:

```
temp = df.groupby("Country").mean()["Price in US Dollars"].sort_values(ascending=False)
x = temp.head(10).index
y = temp.head(10).values

import plotly.graph_objects as go

fig = go.Figure(data=[go.Bar(x=x, y=y)])

layout = go.Layout(title="Top Expensive Big Mac by Country",
                   xaxis = dict(title= "Country"),
                   yaxis = dict(title="Price of Big Mac"),


                  )

fig.update_layout(layout)
fig.show()
```

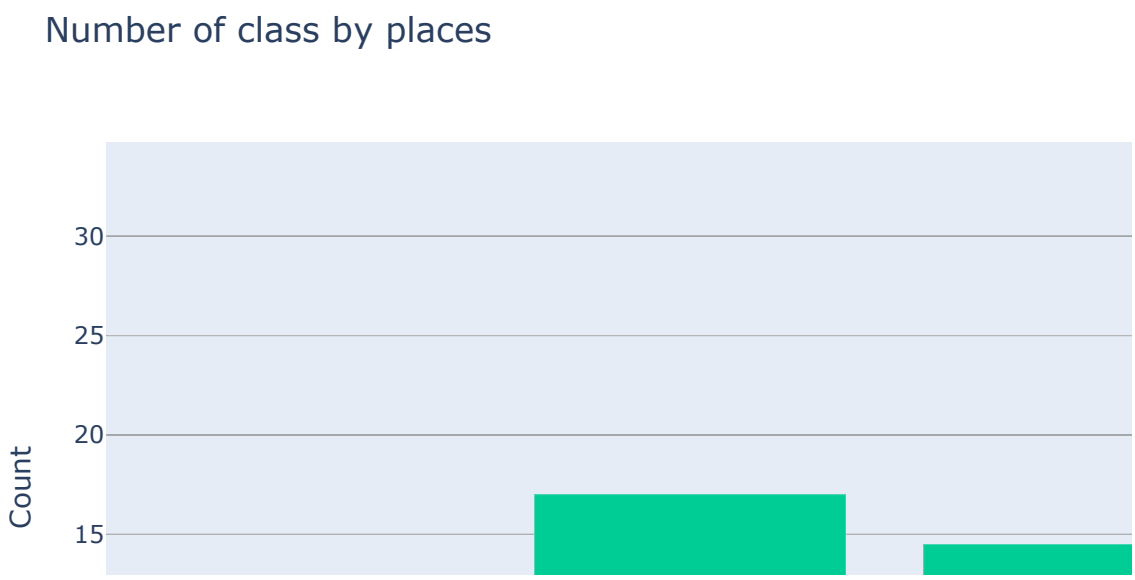


Example of a stacked bar graph.

In this example, we have three classes, which are Toronto, Ottawa and Montreal. This way can help us to compare the number in each class. You can **choose which class to visualise by clicking the legend on the right.**

In [35]:

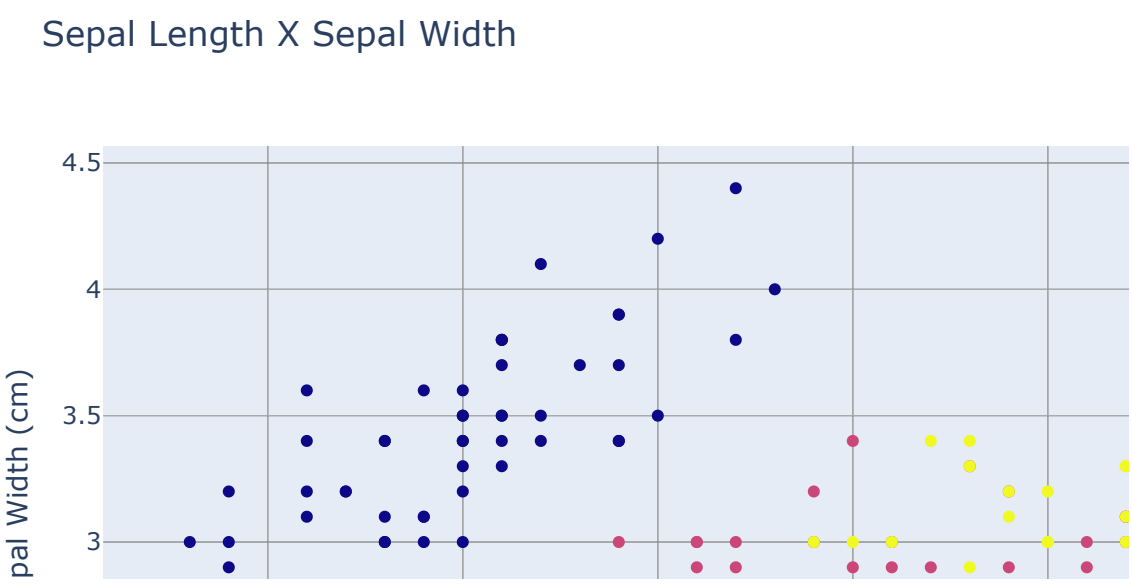
```
import plotly.graph_objects as go

x=['A','B','C','D']
fig = go.Figure()
fig.add_trace(go.Bar(x =x, y=[2,5,1,9], name='Montreal'))
fig.add_trace(go.Bar(x=x, y=[1, 4, 9, 16], name='Ottawa'))
fig.add_trace(go.Bar(x=x, y=[6, 8, 4.5, 8], name='Toronto'))

fig.update_layout(title= "Number of class by places" , barmode='stack', xaxis={'title' : 'C
fig.show()
```



Scatter plot using Plotly.

In [36]:

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

fig = go.Figure()

fig.add_traces(go.Scatter(x =iris_df["sepal length (cm)"] ,
                          y = iris_df["sepal width (cm)"] ,
                          mode='markers',
                          marker = {"color": iris_df["Species"]},
                         )
              )


fig.update_layout(title="Sepal Length X Sepal Width " ,
                  xaxis=dict(title='Sepal Length (cm)'),
                  yaxis=dict(title='Sepal Width (cm)'),
                  showlegend=True
                 )
```



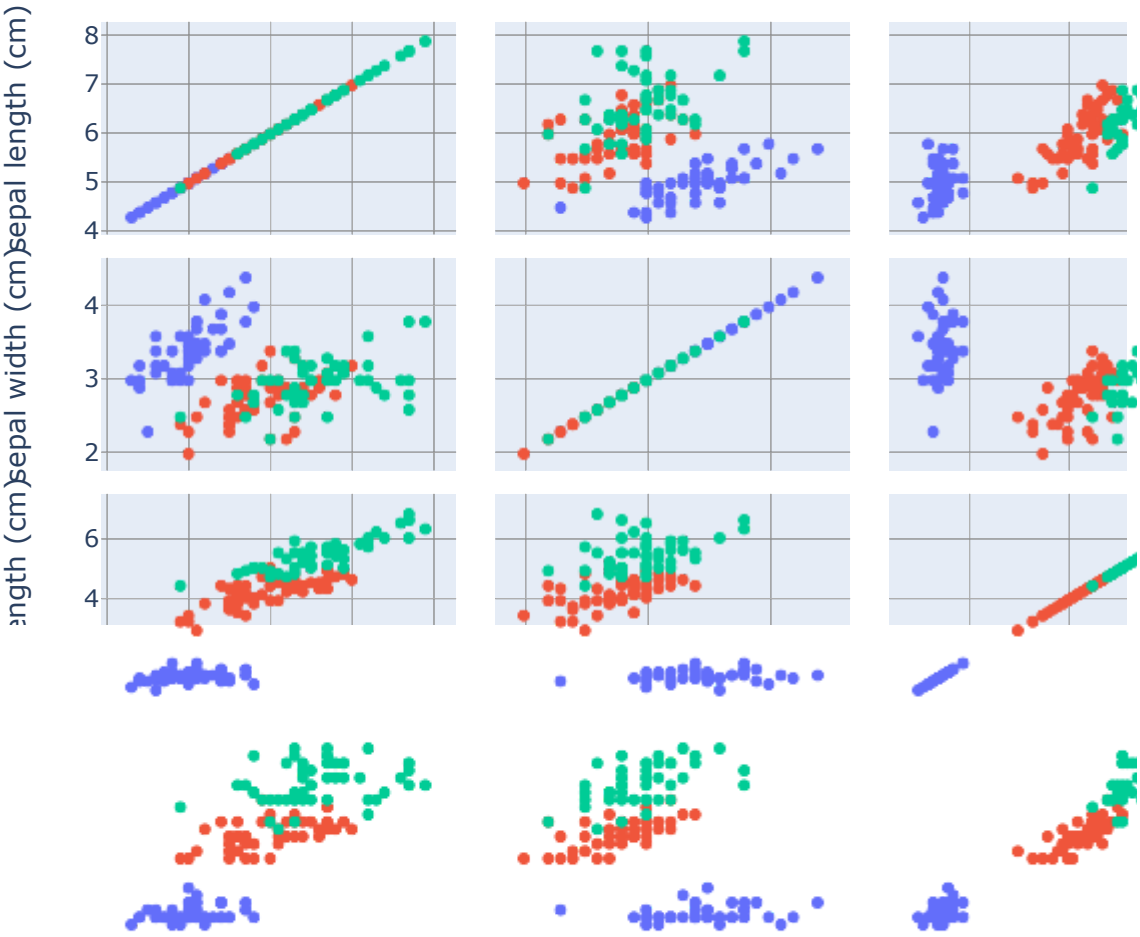
Scatter Plot Matrix using Plotly

In seaborn, this is known as a Pairplot. Although it has a different name, it functions the same. Using 3 to 4 lines of codes, we can visualize all the features. Scatter Matrix plot can only plot numerical columns. We use categorical columns to color the points.

In the example, the "Species" column was used to color the values. The features can be understood much better.

In [37]:

```
import plotly.express as px
# iris = px.data.iris()
iris_df["Species"] = iris_df["Species"].astype("category")

fig = px.scatter_matrix(iris_df,
    dimensions=["sepal length (cm)", "sepal width (cm)" ,"petal length (cm)", "petal width
    color='Species'
                        )
fig.show()
```



Subplots using Plotly

It is difficult to compare the graphs if they are not placed in the same figure. Hence, we can use subplots to plot many graphs at different positions.

Subplots work like a 2D matrix. The graph is position by row and columns.

In [38]:

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

fig = make_subplots(rows=1, cols=2, subplot_titles=["Sepal Length (cm) X Sepal Width (cm)"
                                                    "Petal Length (cm) X Petal Width (cm)"]
```

The **make_subplots()** method will create an object of subplots. We can specify the size of the subplots. We can also add titles for each subplot.

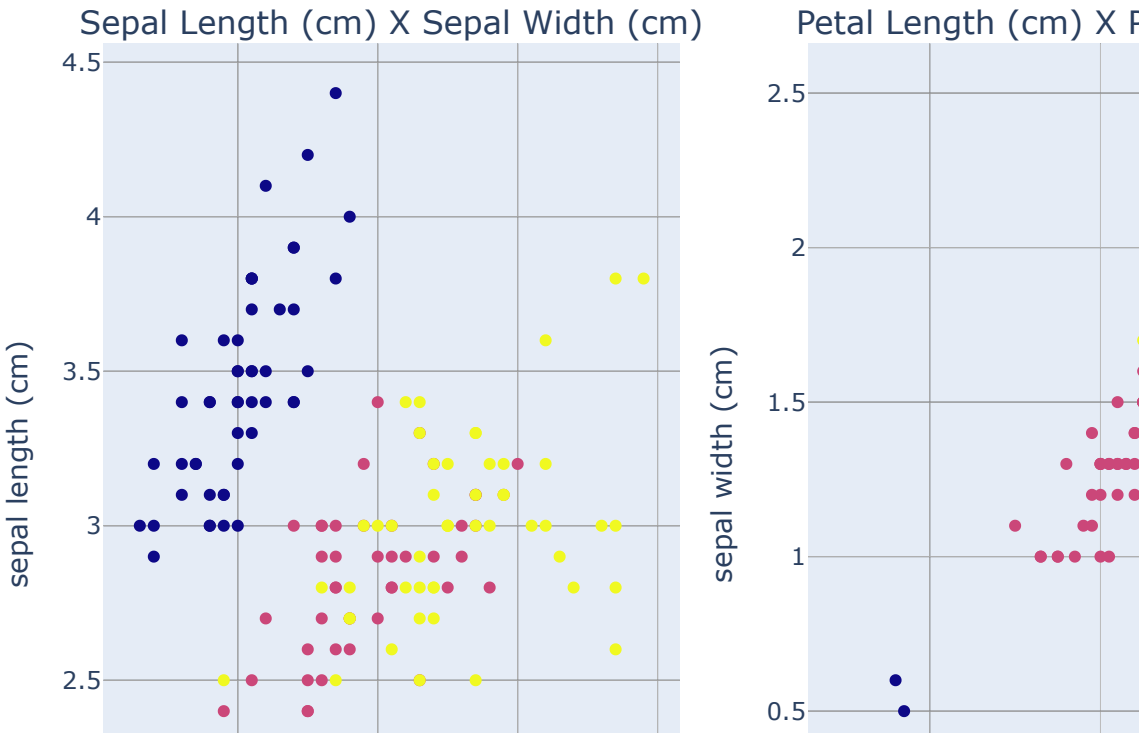
The **add_trace()** method is used to add the graph we want. The graph can be in graph object (we used go.Scatter).

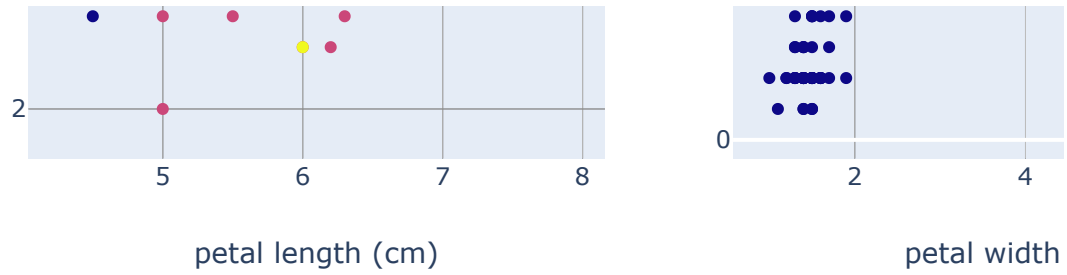
In the same method, we nee to specify the graph position (row=1, col = 1).

The x and y labels should be added to each graph to make it more understandable. In the example, update_xaxes() and update_yaxes() was used to add the x and y labels respectively.

In [39]:

```
fig.add_trace(
    go.Scatter(x=iris_df["sepal length (cm)"],
               y=iris_df["sepal width (cm)"],
               mode='markers',
               marker = dict(color =iris_df['Species']),
               name = "Graph 1",
               showlegend=False,),

    row=1, col=1
)

fig.add_trace(
    go.Scatter(x=iris_df["petal length (cm)"],
               y=iris_df["petal width (cm)"],
               mode='markers',
               marker = dict(color =iris_df['Species']),
               name = "Graph 2",
               showlegend=False,),
    row=1, col=2
)


# Update yaxis properties
fig.update_yaxes(title_text="sepal length (cm)", row=1, col=1)
fig.update_yaxes(title_text="sepal width (cm)", row=1, col=2)

# Update xaxis properties
fig.update_xaxes(title_text="petal length (cm)", row=1, col=1)
fig.update_xaxes(title_text="petal width (cm)", row=1, col=2)

fig.update_layout(height=600, width=800, )
fig.show()
```




2
5
6
7
8
0
2
4
petal length (cm)
petal width