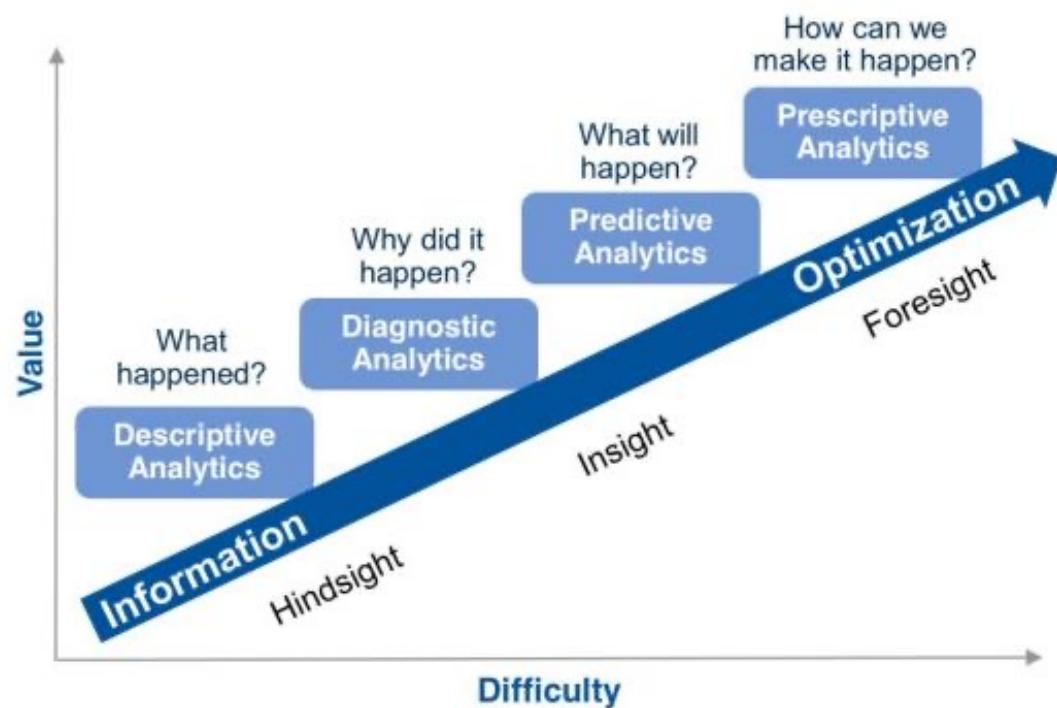


Data Visualization With Python

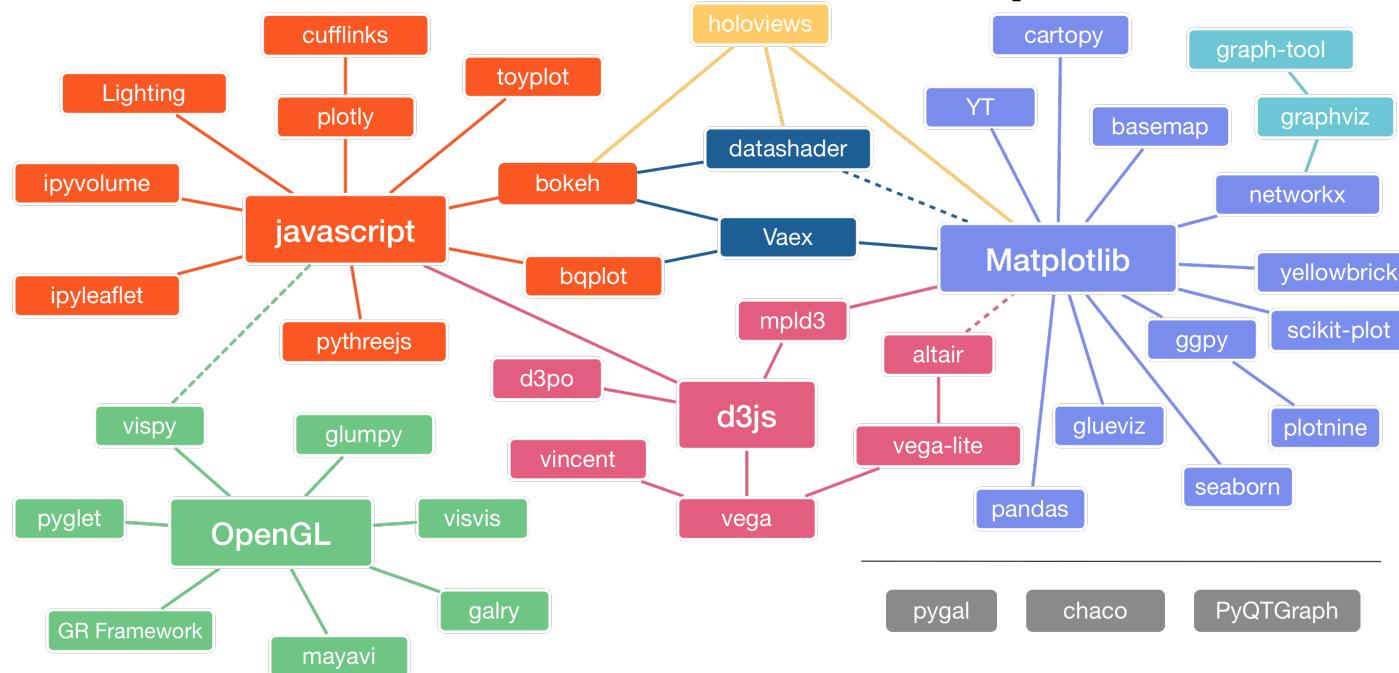
Based on Prof.Veera's slide

Outline

1. Matplotlib
2. Pandas plot
3. Seaborn
4. Plotly & Plotly Express
5. Streamlit & Dash



Python visualization landscape



- Scientific visualization (green)
 - Visualize 3D (+time) physical processes.
 - OpenGL

<https://pyviz.org/overviews/index.html>

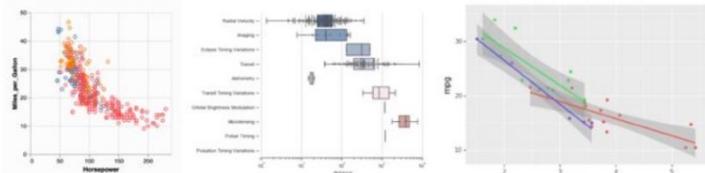
Python visualization package

- **Matplotlib**
 - Matplotlib wrapper
 - Pandas plot
 - Seaborn
 - ggplot (plotnine): based on R's ggplot2, uses Grammar of Graphics
 - <https://plotnine.readthedocs.io>
- **Javascript-based Viz (interactive viz)**
 - Plotly
 - Bokeh <http://bokeh.org/>
- **Dashboard web app**
 - Dash (based on Plotly)
 - Streamlit <https://streamlit.io/>

Plot Types

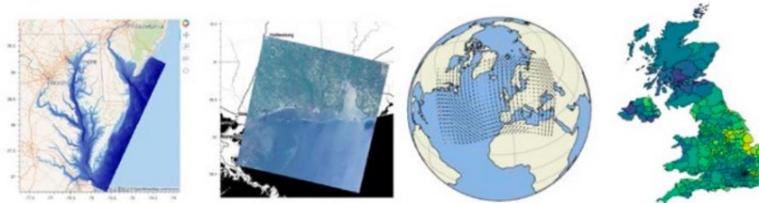
The most basic plot types are shared between multiple libraries, and others are only available in certain libraries.

Given the number of libraries, plot types, and their changes over time, it is very difficult to precisely characterize what's supported in each library. It is usually clear what the focus is if you look at the example galleries for each library.



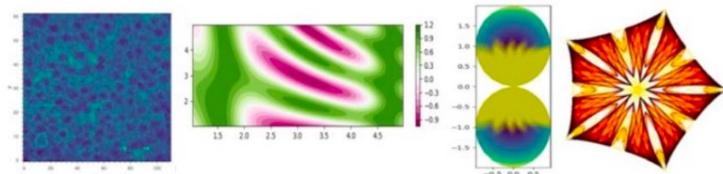
Statistical plots (scatter plots, lines, areas, bars, histograms)

Covered well by nearly all InfoVis libraries, but are the main focus for Seaborn, bqplot, Altair, ggplot2, and plotnine.



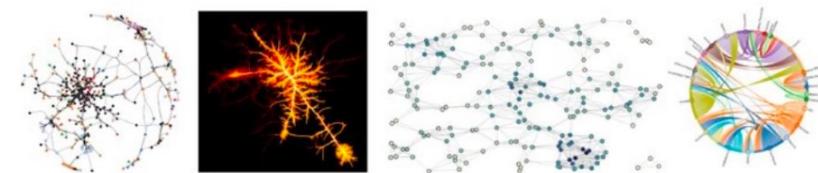
Geographical data

Matplotlib (with Cartopy), GeoViews, ipyleaflet, and Plotly.



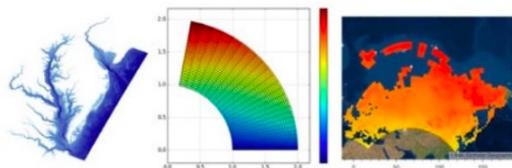
Multidimensional arrays (regular grids, rectangular meshes)

Well supported by Bokeh, Datasader, HoloViews, Matplotlib, Plotly, plus most of the SciVis libraries.



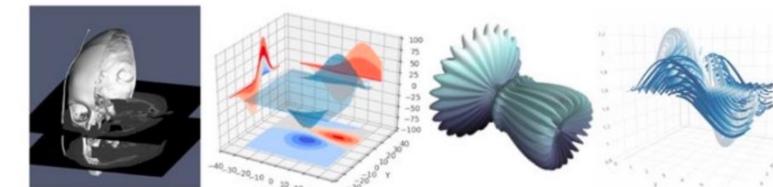
Networks/graphs

NetworkX, Plotly, Bokeh, HoloViews, and Datasader.



Irregular 2D meshes (triangular grids)

Well supported by the SciVis libraries plus Matplotlib, Bokeh, Datasader, and HoloViews.



3D (meshes, scatter, etc.)

Fully supported by the SciVis libraries, plus some support in Plotly, Matplotlib, HoloViews, and ipyvolume.

1) Matplotlib

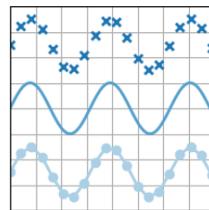
Matplotlib

- <https://matplotlib.org/>
- Mature, widely used
- Low level
 - **Require many steps to adjust the details of a plot**
 - Many ways to achieve the same goal
- Wide range of 2D plot types
- Focus on **static images**
- Export to many file formats (savefig function)
- Limited interactive viz (not enabled by default)

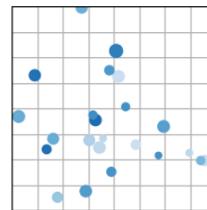
```
import matplotlib.pyplot as plt
```

Plot types

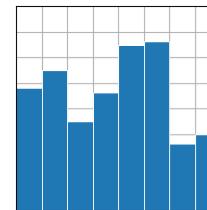
- https://matplotlib.org/stable/plot_types/index.html



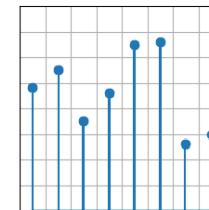
`plot(x, y)`



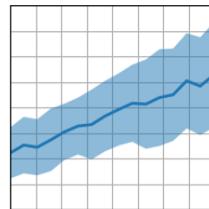
`scatter(x, y)`



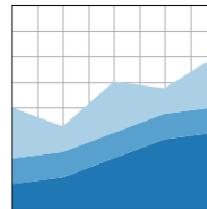
`bar(x, height)`



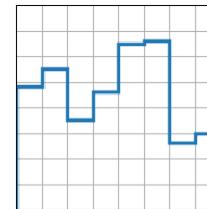
`stem(x, y)`



`fill_between(x, y1, y2)`



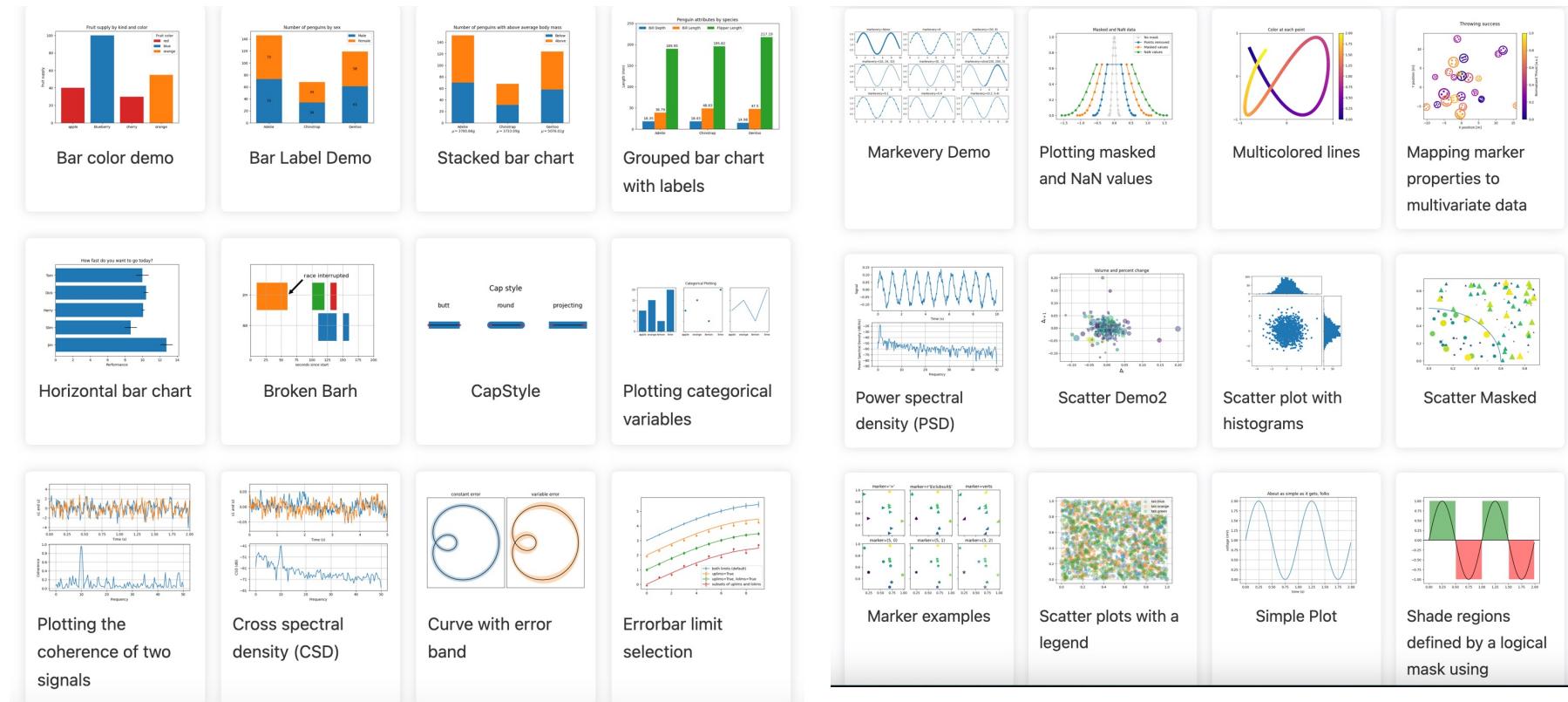
`stackplot(x, y)`



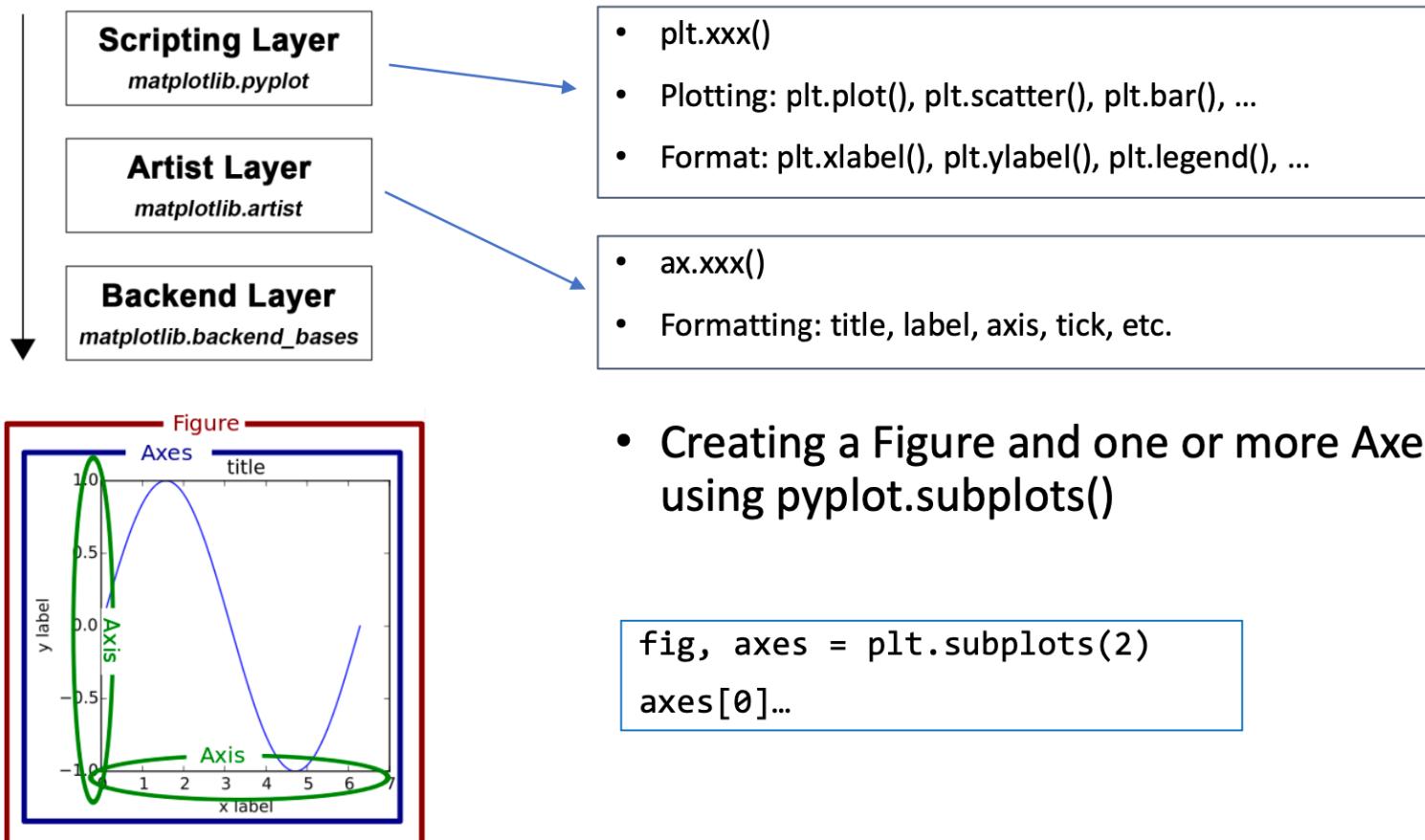
`stairs(values)`

Matplotlib gallery

- <https://matplotlib.org/stable/gallery/index.html>

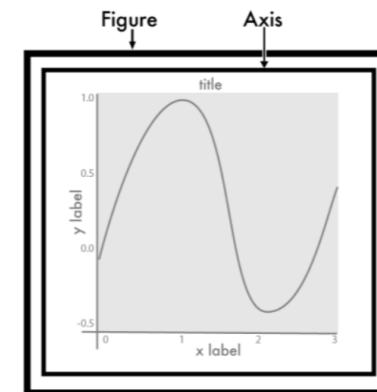
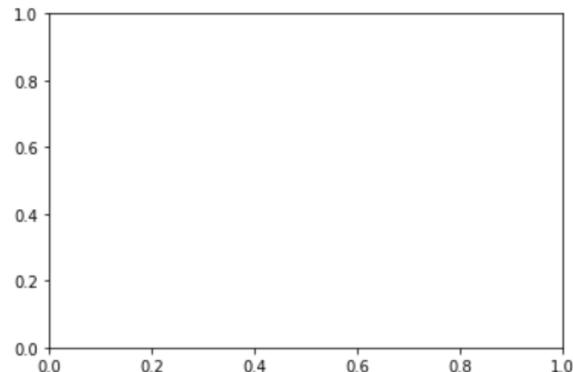


Matplotlib components

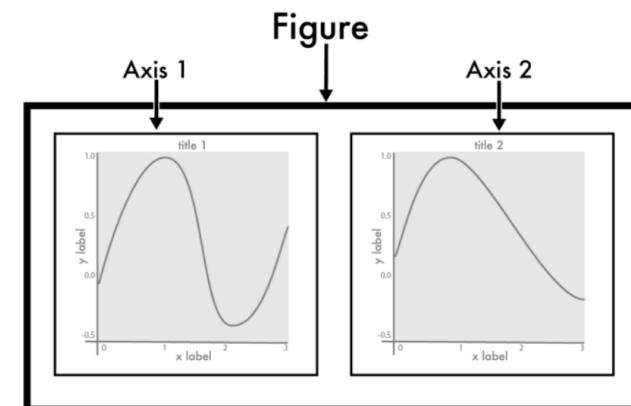
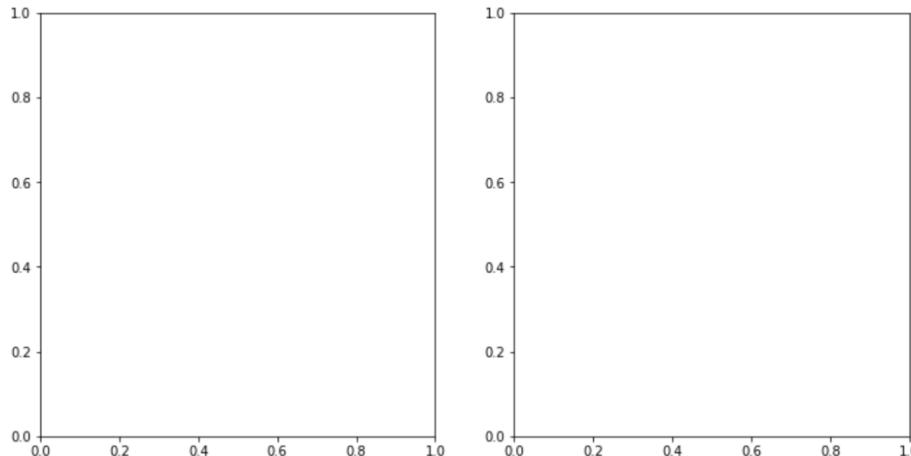


Create figures

```
# Create a figure containing a single plot (axis)
fig, ax = plt.subplots()
```



```
# Figure with two plots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 6))
```



Simple plot

Create a simple plot.

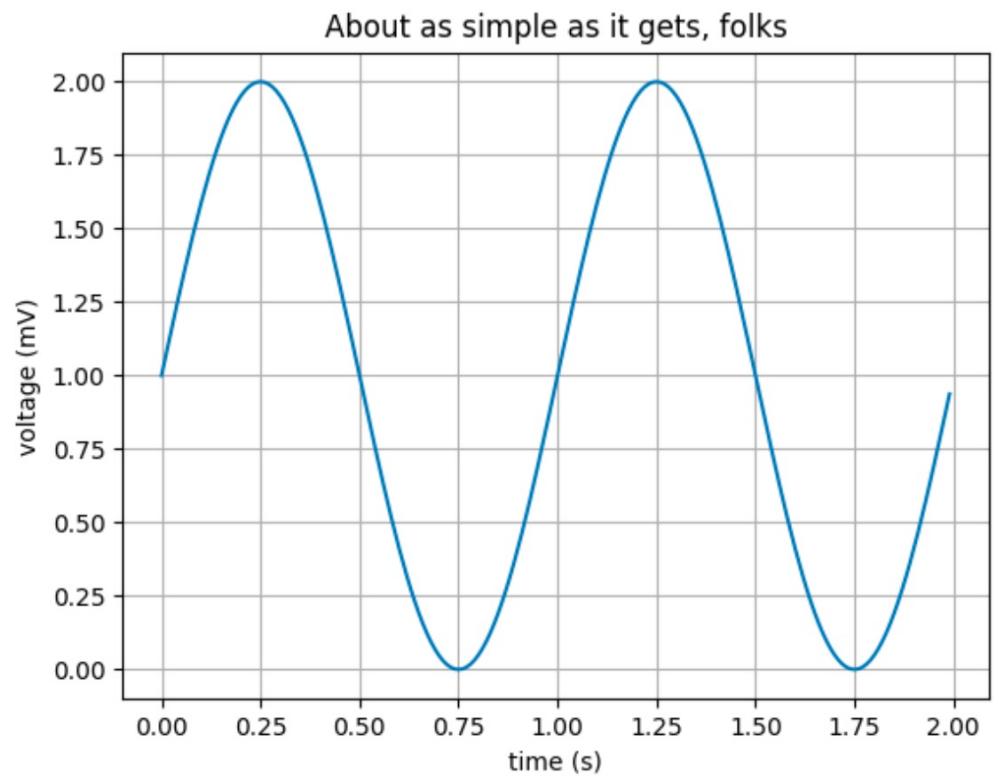
```
import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()

fig.savefig("test.png")
plt.show()
```



Two implementation options

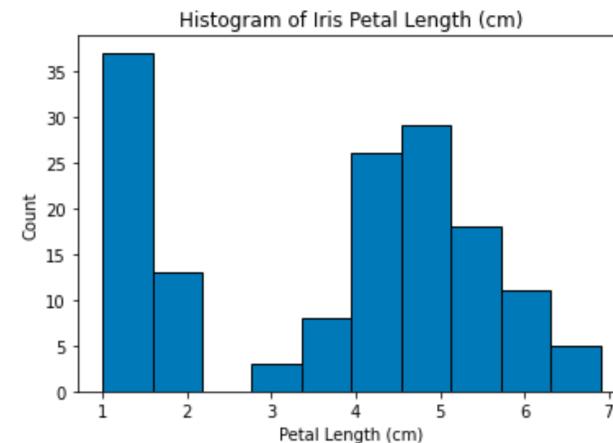
- Pyplot API
 - `matplotlib.pyplot` → `plt`
- Object-oriented API
 - `matplotlib.figure`
 - `matplotlib.axes`

Using pyplot

```
plt.hist(iris['PetalLengthCm'], bins=10, edgecolor='black')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Count')
plt.title('Histogram of Iris Petal Length (cm)')
plt.show()
```

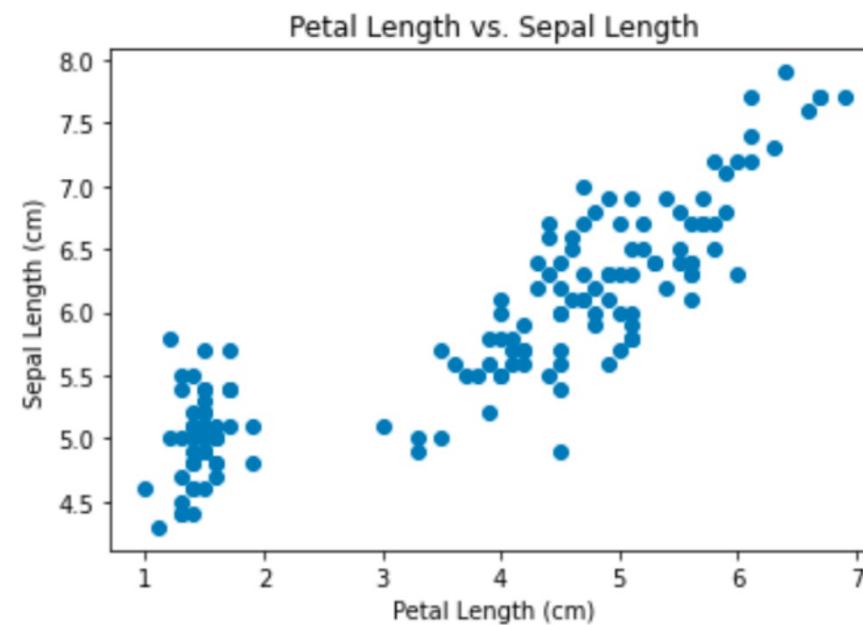
Using axes

```
fig, axes = plt.subplots()
axes.hist(iris['PetalLengthCm'], bins=10, edgecolor='black')
axes.set_xlabel('Petal Length (cm)')
axes.set_ylabel('Count')
axes.set_title('Histogram of Iris Petal Length (cm)')
plt.show()
```



Scatter plot

```
plt.scatter(x=iris['PetalLengthCm'], y=iris['SepalLengthCm'])
plt.xlabel('Petal Length (cm)')
plt.ylabel('Sepal Length (cm)')
plt.title('Petal Length vs. Sepal Length')
plt.show()
```



2) Pandas plot

- Work with matplotlib
- Work directly with Pandas DataFrame



Input/output

General functions

Series

DataFrame

pandas.DataFrame

pandas.DataFrame.index

pandas.DataFrame.columns

pandas.DataFrame.dtypes

pandas.DataFrame.info

pandas.DataFrame.select_dtypes

pandas.DataFrame.values

pandas.DataFrame.axes

pandas.DataFrame.ndim

pandas.DataFrame.size

pandas.DataFrame.shape

pandas.DataFrame.memory_usage

pandas.DataFrame.empty

pandas.DataFrame.set_flags

pandas.DataFrame.astype

pandas.DataFrame.convert_dtypes

[Home](#) > API reference > DataFrame > pandas.DataFrame.plot

pandas.DataFrame.plot

DataFrame.plot(*args, **kwargs)[\[source\]](#)

Make plots of Series or DataFrame.

Uses the backend specified by the option `plotting.backend`. By default, matplotlib is used.**Parameters:****data : Series or DataFrame**

The object for which the method is called.

x : label or position, default None

Only used if data is a DataFrame.

y : label, position or list of label, positions, default None

Allows plotting of one column versus another. Only used if data is a DataFrame.

kind : str

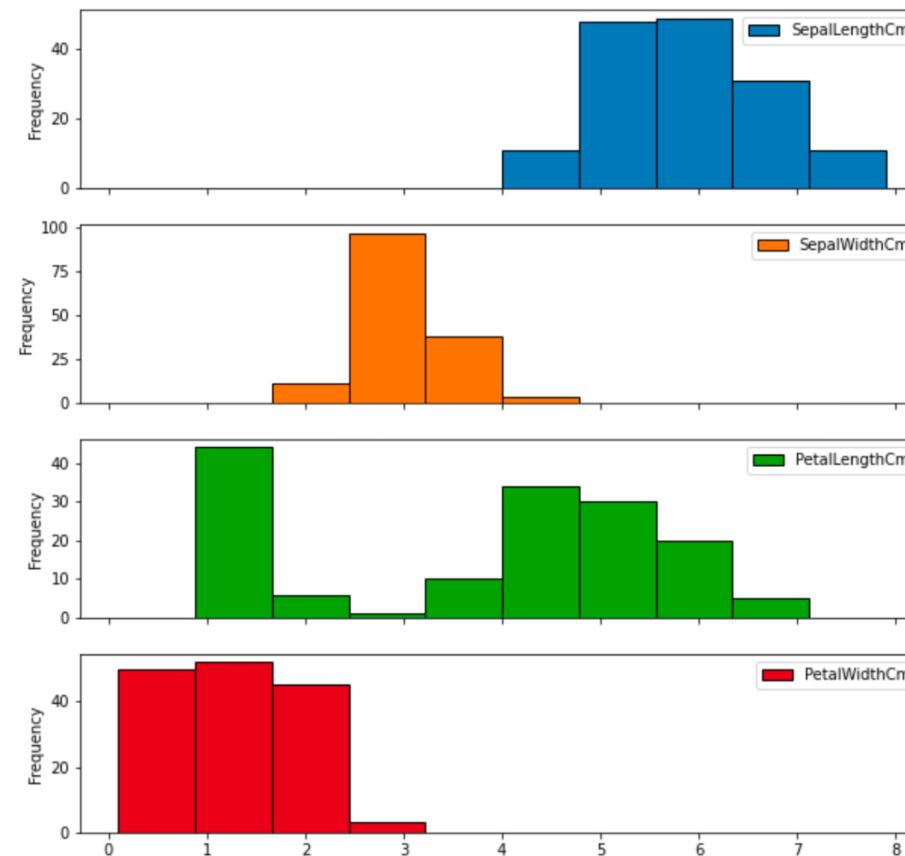
The kind of plot to produce:

- 'line' : line plot (default)
- 'bar' : vertical bar plot
- 'barh' : horizontal bar plot
- 'hist' : histogram

[On this page](#)[DataFrame.plot\(\)](#)[Show Source](#)

Plot directly from Pandas DataFrame

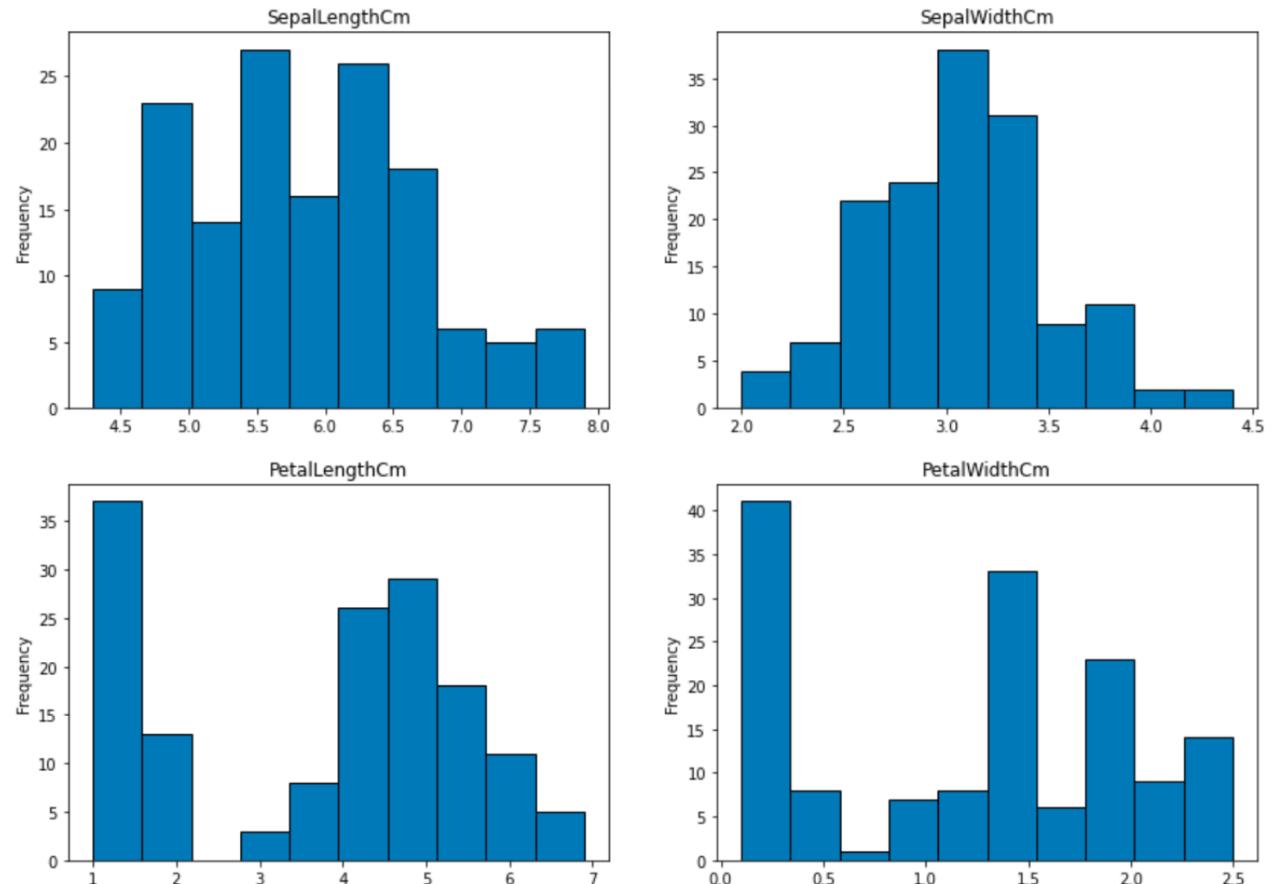
```
iris.loc[:, 'SepalLengthCm':'PetalWidthCm'].plot.hist(bins=10, edgecolor='black', figsize=(10, 10), subplots=True)  
plt.show()
```



Can work with Matplotlib

```
fig, ax = plt.subplots(2, 2, figsize=(14,10))

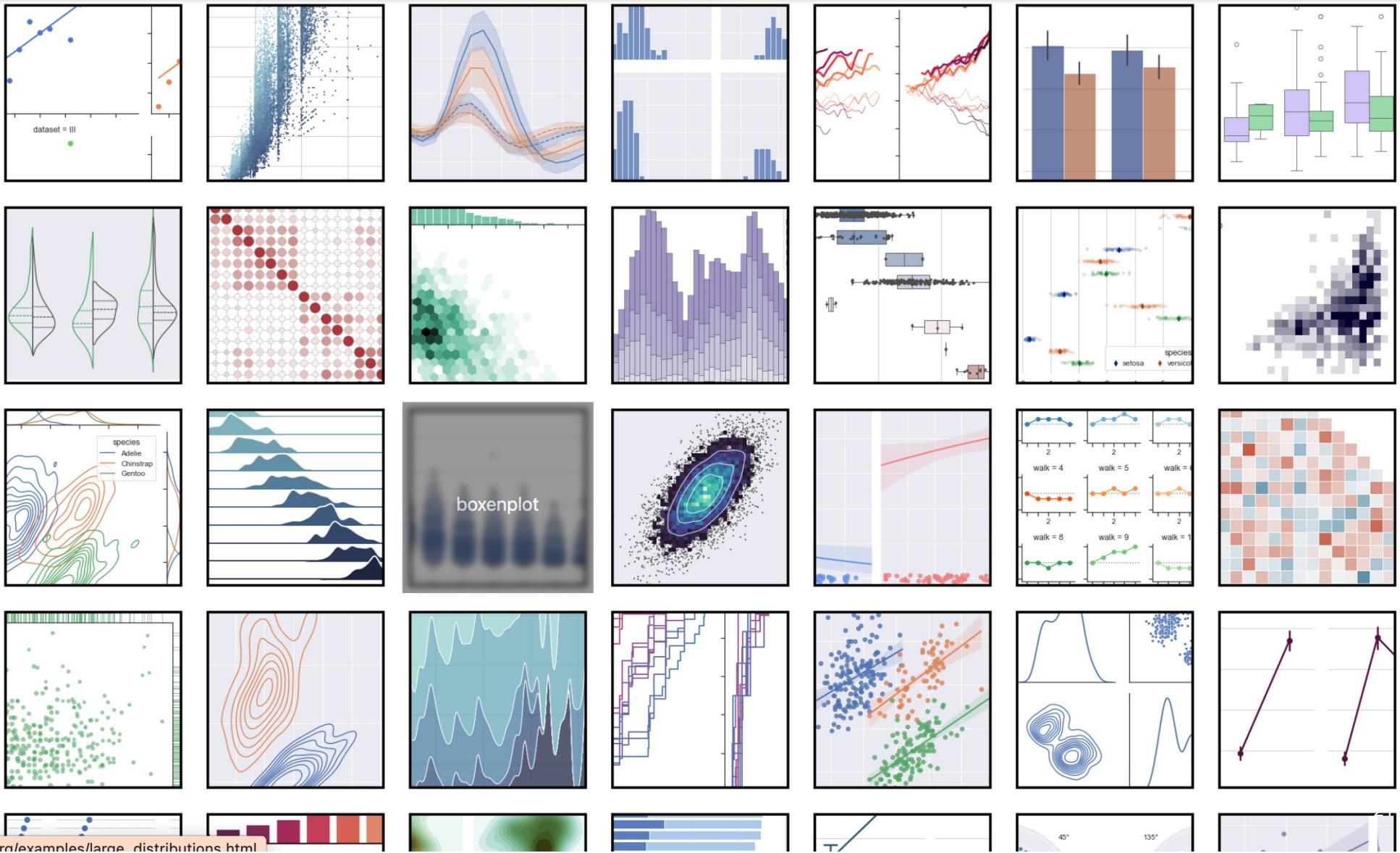
iris['SepalLengthCm'].plot.hist(ax=ax[0,0], bins=10, edgecolor='black')
iris['SepalWidthCm'].plot.hist(ax=ax[0,1], bins=10, edgecolor='black')
iris['PetalLengthCm'].plot.hist(ax=ax[1,0], bins=10, edgecolor='black')
iris['PetalWidthCm'].plot.hist(ax=ax[1,1], bins=10, edgecolor='black')
ax[0,0].set_title('SepalLengthCm')
ax[0,1].set_title('SepalWidthCm')
ax[1,0].set_title('PetalLengthCm')
ax[1,1].set_title('PetalWidthCm')
plt.show()
```



3) Seaborn

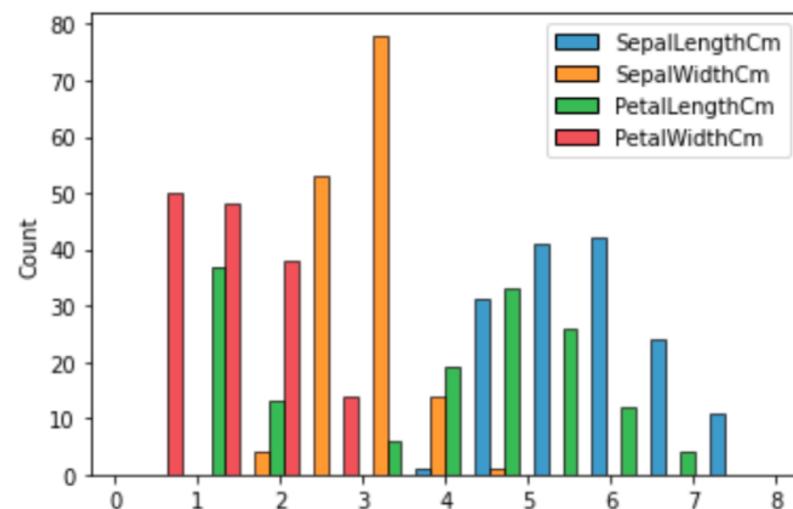
Seaborn

- Based on **Matplotlib** and integrates with **pandas**
- Better than Matplotlib
- dataset-oriented, declarative API
 - Plot types
 - Relational plots
 - Distribution plots
 - Categorical plots
 - Regression plots
 - Matrix plots
 - Multi-plot grids
- <https://seaborn.pydata.org/>
- <https://seaborn.pydata.org/examples/index.html>



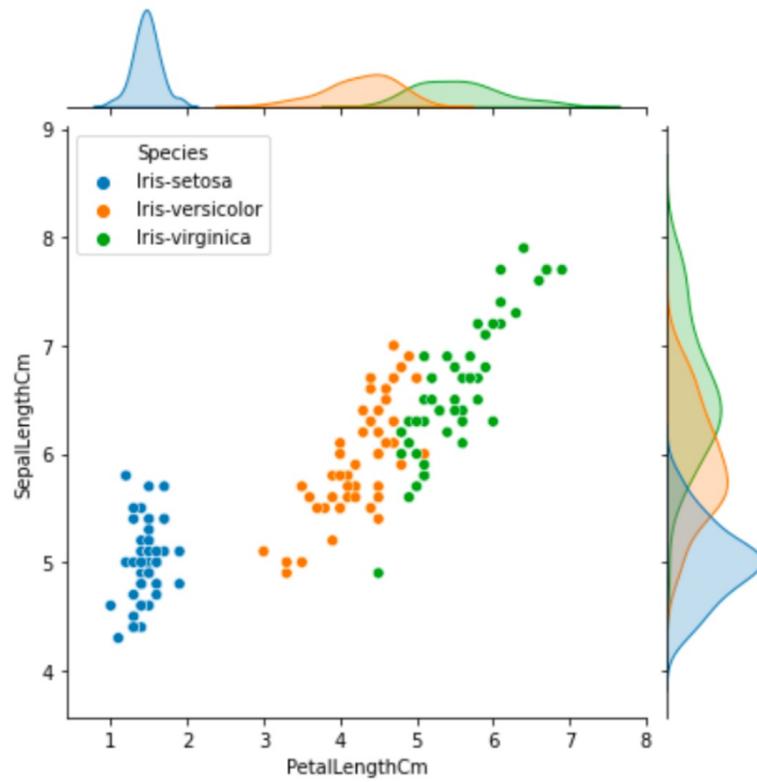
Histogram with Seaborn

```
sns.histplot(data=iris.loc[:, 'SepalLengthCm':'PetalWidthCm'], multiple='dodge')
plt.show()
```



Joint plot with Seaborn

```
sns.jointplot(data=iris, x='PetalLengthCm', y='SepalLengthCm', hue='Species')  
plt.show()
```



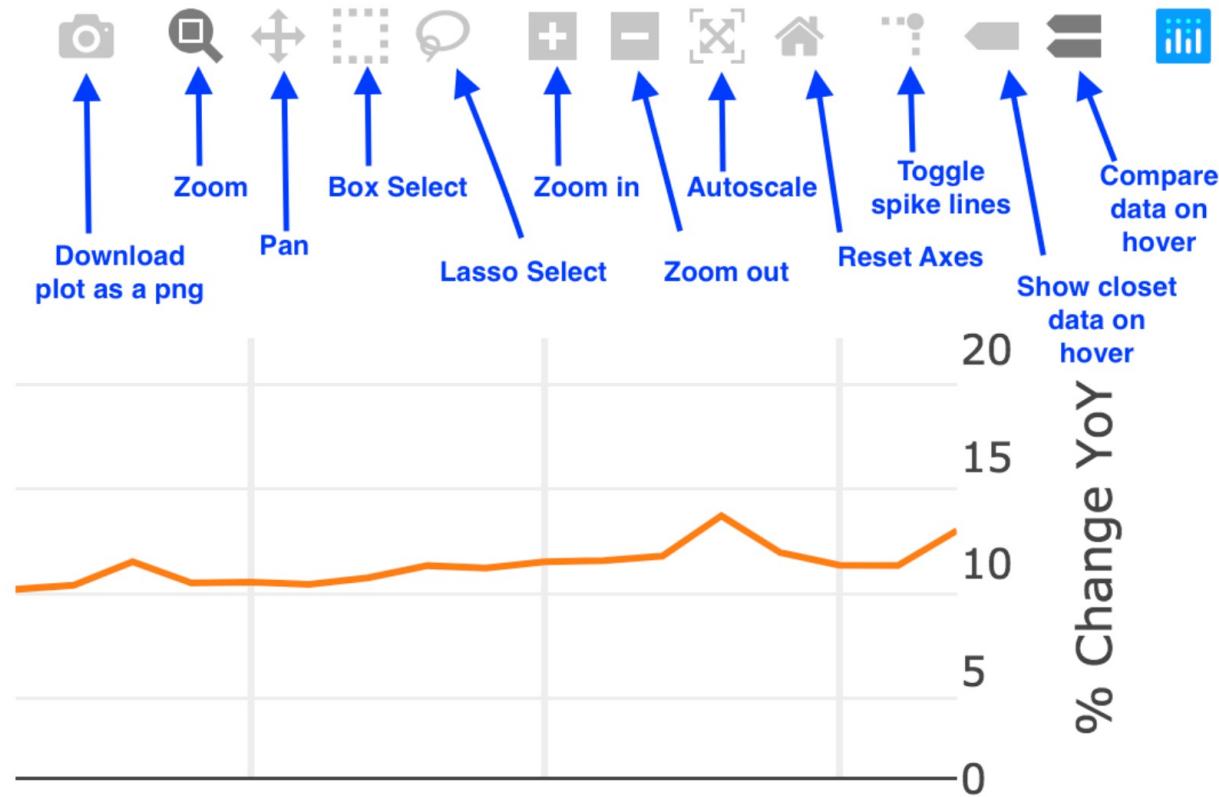
4) Plotly & Plotly Express

Interactive visualization

Plotly & Plotly Express

- Based on Javascript
- Plotly Express is a high-level wrapper for Plotly.
 - Each plot function has a lot of parameters.
- Interactive viz
- Also support
 - 3D viz
 - Animation
 - Geographical viz
- <https://plot.ly/python/>
- <https://chart-studio.plotly.com/feed/#/>

Plotly Graph Tools

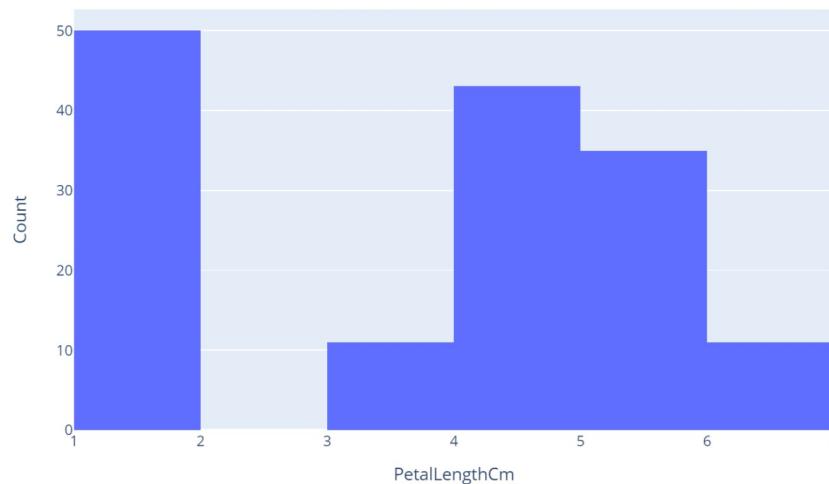


Histogram with Plotly Graph Objects

```
import plotly.graph_objects as go

fig = go.Figure()
fig.add_traces(go.Histogram(x= iris['PetalLengthCm'], nbinsx=10))
fig.update_layout(xaxis_title="PetalLengthCm", yaxis_title="Count")
fig.show()

## Note: nbinsx specifies the maximum number of bins.
## Plotly histogram algorihtm will decide the optimal in size such that the histogram
## best visualize the distribution of the data.
```

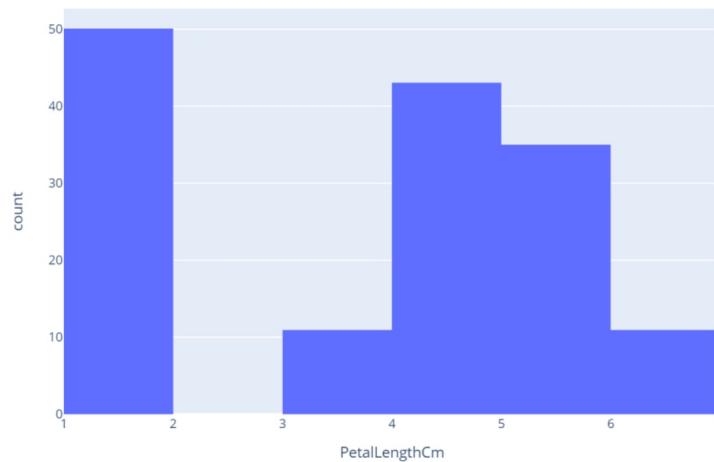


Histogram with Plotly Express

`plotly.express.histogram`

```
plotly.express. histogram(data_frame=None, x=None, y=None, color=None, pattern_shape=None,  
facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None,  
hover_name=None, hover_data=None, animation_frame=None, animation_group=None,  
category_orders=None, labels=None, color_discrete_sequence=None, color_discrete_map=None,  
pattern_shape_sequence=None, pattern_shape_map=None, marginal=None, opacity=None,  
orientation=None, barmode='relative', barnorm=None, histnorm=None, log_x=False, log_y=False,  
range_x=None, range_y=None, histfunc=None, cumulative=None, nbins=None, text_auto=False, title=None,  
template=None, width=None, height=None)
```

```
import plotly.express as px  
px.histogram(iris, x='PetalLengthCm', nbins=10)
```



5) Dashboard with Streamlit and Dash

Interactive web application

Streamlit

- <https://streamlit.io/>
- A lightweight, open-source framework for quickly building and deploying interactive apps.
- Script-based, turning a Python script into an app with minimal effort.
- Focuses on ease of use and **rapid prototyping**.
- Common Use Cases:
 - Data exploration
 - Machine learning model visualization
 - Rapid prototyping of data applications

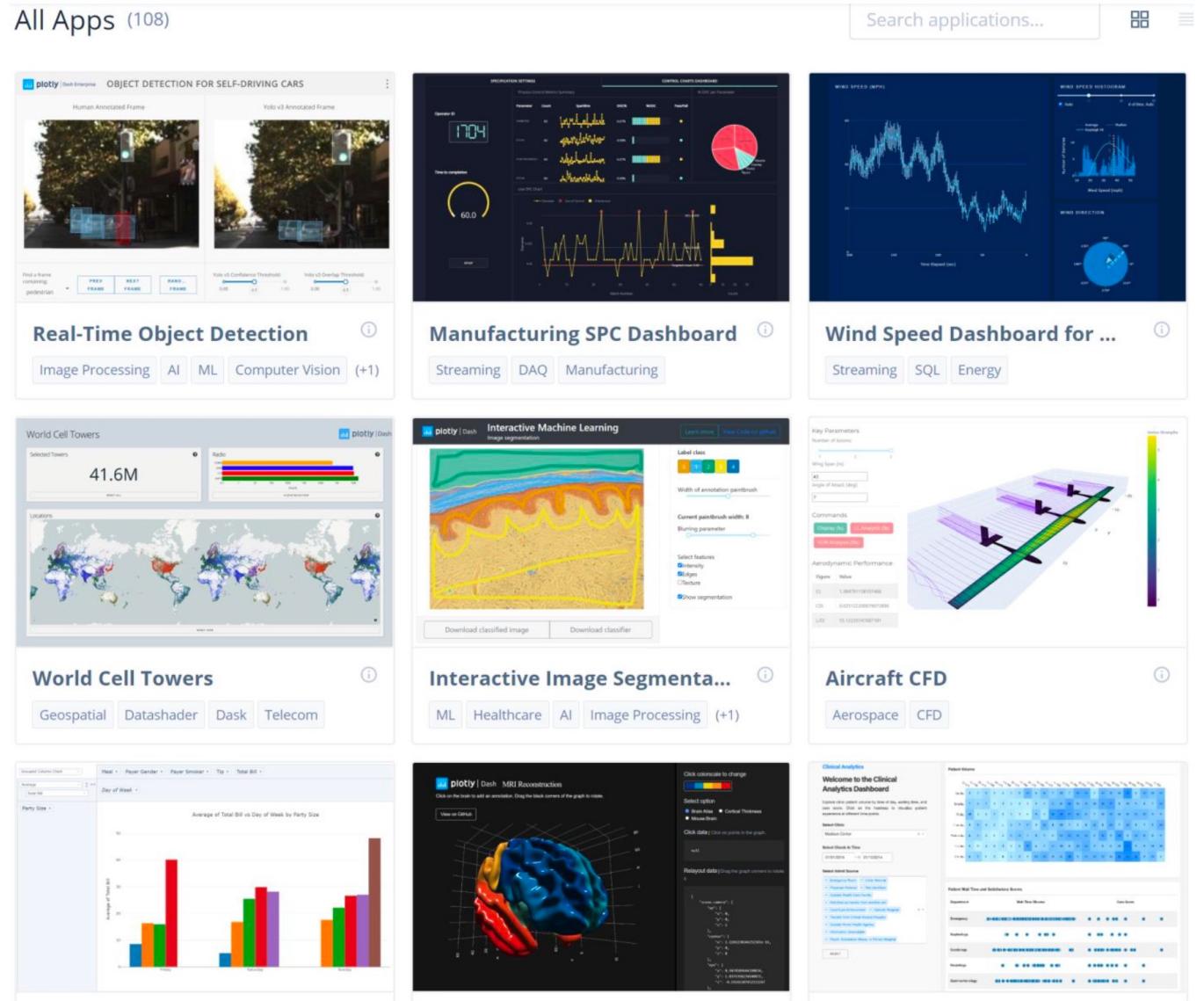
Plotly Dash

- <https://dash.plotly.com/>
- A framework developed by Plotly for creating analytical web applications with Python.
- Component-based, declarative structure using callbacks to manage interactivity.
- Designed for complex dashboards and multi-page applications.
- Common Use Cases:
 - Complex data dashboards
 - Real-time monitoring applications
 - Business and operational reporting tools

Streamlit vs. Dash

Aspect	Streamlit	Dash
Complexity	Simple script-based	More complex component-based
Best For	Rapid prototyping and data exploration	Complex production-ready dashboards
Interactivity	Basic interactive widgets	Advanced customizable callbacks
Learning Curve	Easy with minimal setup	Steeper more declarative structure
UI Customization	Limited with auto-layout	High supports HTML and CSS styling
Multi-Page Support	No (single-page apps only)	Yes (supports multi-page apps)
Component Library	Growing but limited	Extensive includes HTML and core components
Deployment Options	Simple supports cloud deployment	Enterprise-grade deployment options

Dash sample apps



Dash structure

- Layouts
 - Describe the layout of the dashboard.
- Callbacks
 - Interactive control
 - Get input and execute changes.

Simple Dash application (histogram)

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

iris = pd.read_csv('iris.csv')

app = Dash(__name__)

app.layout = html.Div([
    dcc.Graph(id='graph'),
    dcc.Slider(id="bins", min=1, max=10, value=2, step=1,
               marks={str(x):str(x) for x in range(1,11)})
])
```

```
@app.callback(
    Output('graph', 'figure'),
    Input(component_id='bins',
          component_property='value'))

def update_figure(bins):
    fig = px.histogram(iris, x='PetalLengthCm',
                      nbins=bins)
    return fig

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

Dash application (Gapminder)

```
from dash import Dash, dcc, html, 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(__name__)

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

```
@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=True)
```

Dash application (Gapminder)

```
> python .\dash_gapminder.py  
Dash is running on http://127.0.0.1:8050/
```

