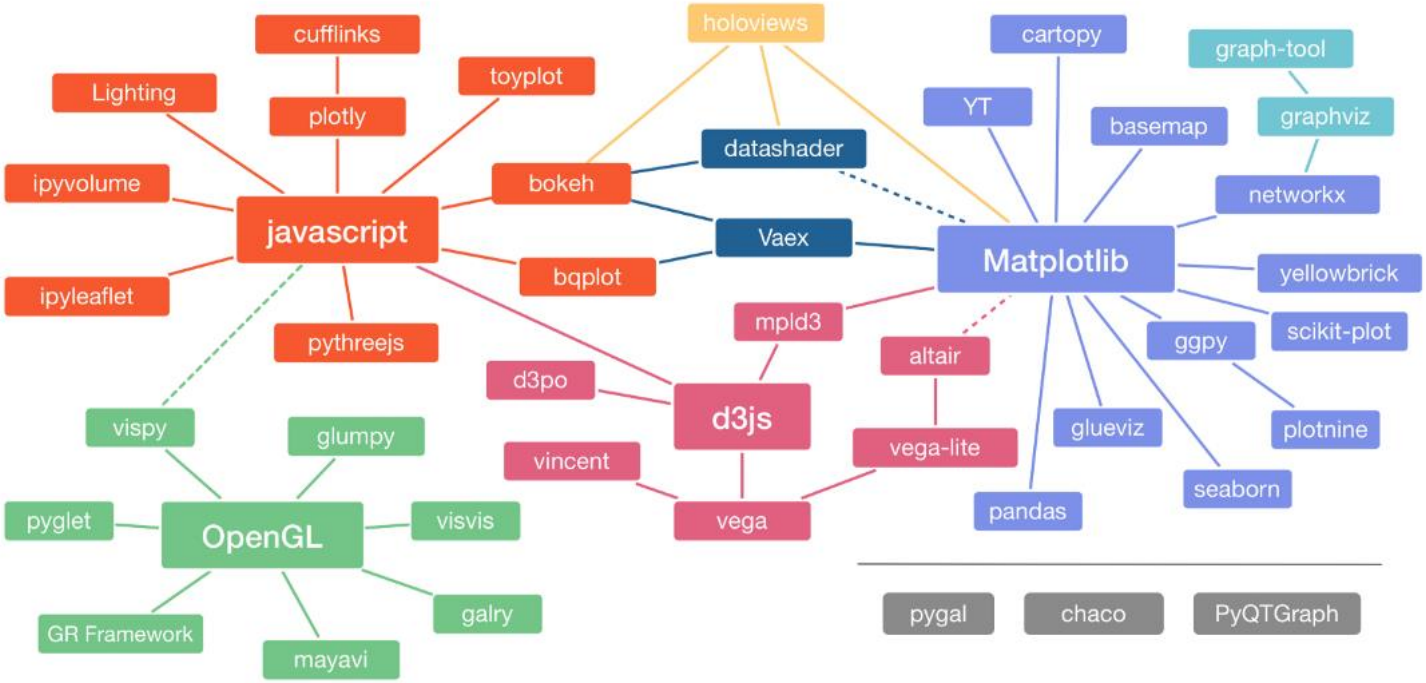
Data Visualization with Python

Veera Muangsin

Python Visualization Landscape



- Scientific visualization (green)
  - visualize 3D (+time) physical processes.
  - OpenGL
- Information visualization (others)

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

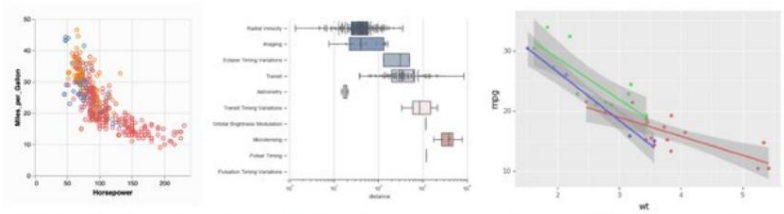
Python Visualization Packages

- **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

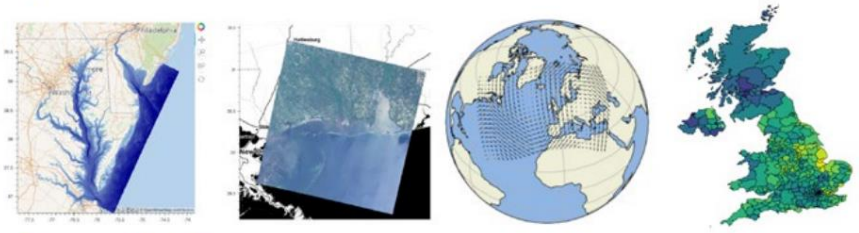
The most basic plot types are shared between multiple libraries, and others are only available in certain libraries.

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.



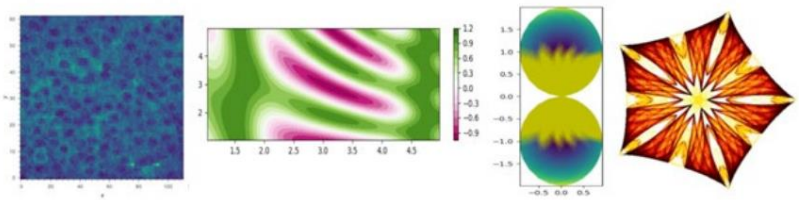
Statistical plots (scatter plots, lines, areas, bars, histograms)

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



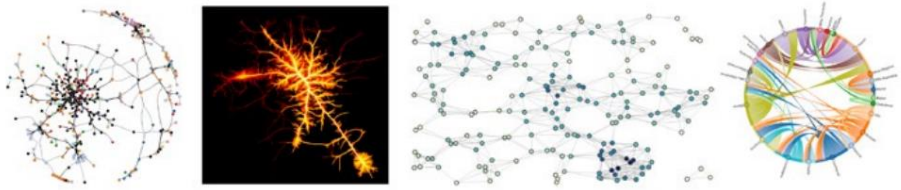
Geographical data

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



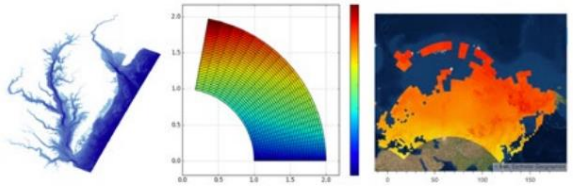
Multidimensional arrays (regular grids, rectangular meshes)

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



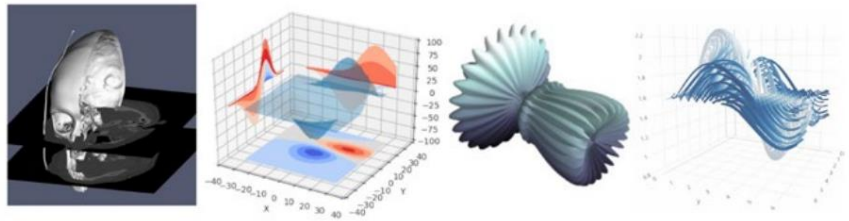
Networks/graphs

NetworkX, Plotly, Bokeh, HoloViews, and Datashader.



Irregular 2D meshes (triangular grids)

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



3D (meshes, scatter, etc.)

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

Matplotlib

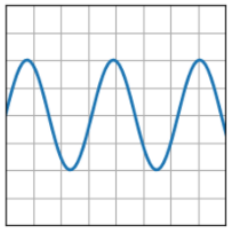
Matplotlib

- 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)

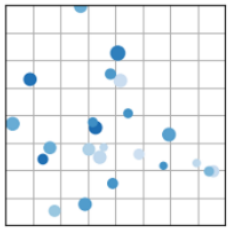
```
import matplotlib.pyplot as plt
```

Plot Types

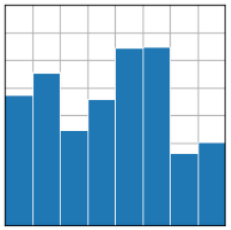
- https://matplotlib.org/stable/plot_types/index.html



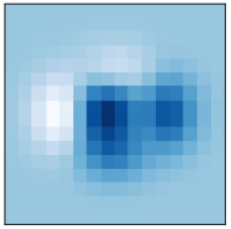
plot(x, y)



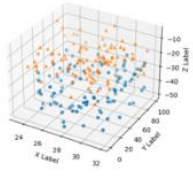
scatter(x, y)



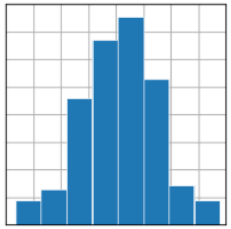
bar(x, height) / barh(y, width)



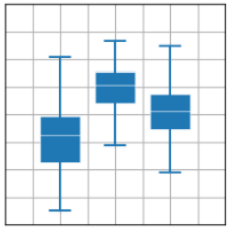
imshow(Z)



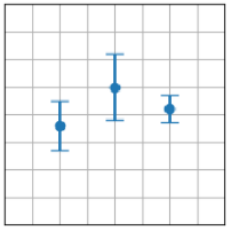
3D scatterplot



hist(x)



boxplot(X)



errorbar(x, y, yerr, xerr)

pie(x)

Matplotlib Gallery

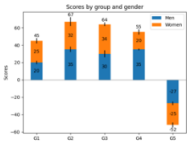
- https://matplotlib.org/stable/gallery/index.html

Examples

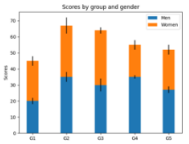
This page contains example plots. Click on any image to see the full image and source code.

For longer tutorials, see our tutorials page. You can also find external resources and a FAQ in our user guide.

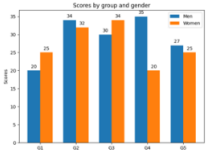
Lines, bars and markers



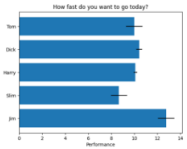
Bar Label Demo



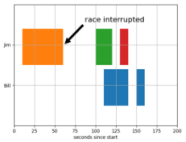
Stacked bar chart



Grouped bar chart with labels

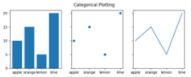


Horizontal bar chart

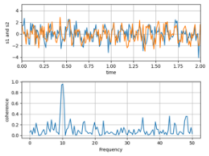


Broken Barh

CapStyle



Plotting categorical variables



Plotting the coherence of two signals

On this page

Matplotlib APIs

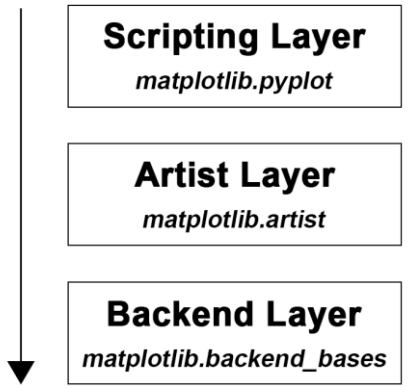
- Pyplot API
  - Command style functions that make matplotlib work like MATLAB.
  - matplotlib.pyplot
  - Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
  - Less-flexible than the object-oriented API

- Object-oriented API
  - Create Figure and Axes objects and call their methods to add content and modify the appearance.
  - matplotlib.figure: axes creation, figure-level content
  - matplotlib.axes: most plotting methods, Axes labels, access to axis styling, etc.
  - More control and customization

https://matplotlib.org/stable/api/index.html

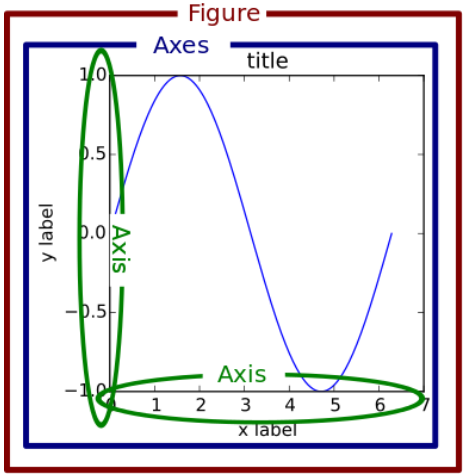
Matplotlib components

- A plot can be built by adding new elements.
- Figure: overall space that contain one or more plots
- Axes: individual plots in the figure



- plt.xxx()
- Plotting: plt.plot(), plt.scatter(), plt.bar(), …
- Format: plt.xlabel(), plt.ylabel(), plt.legend(), …

- ax.xxx()
- Formatting: title, label, axis, tick, etc.

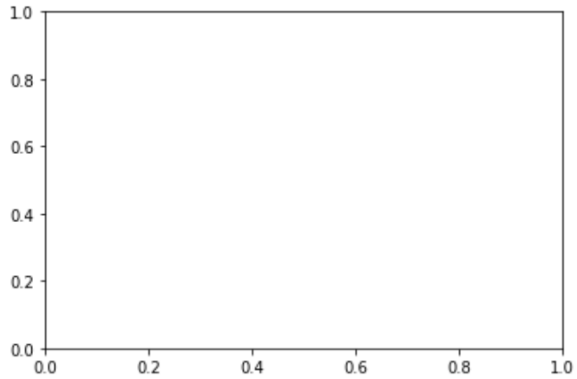


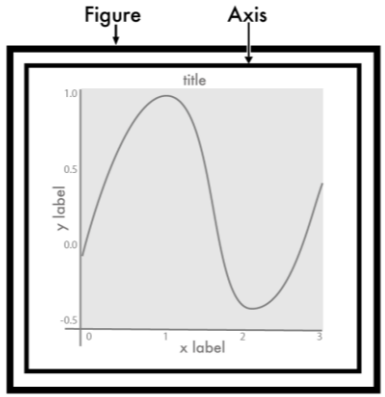
- Creating a Figure and one or more Axes using pyplot.subplots()

```
fig, axes = plt.subplots(2)
axes[0]…
```

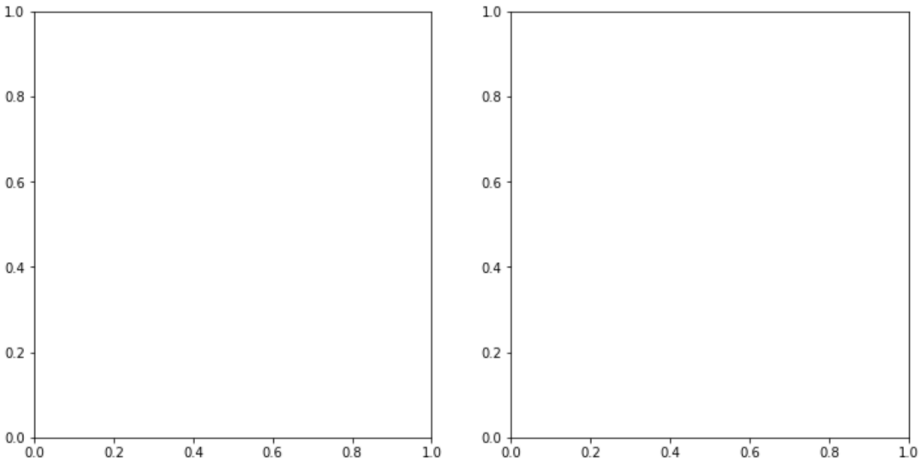
- A plot can be built by adding new elements.
- Figure: overall space that contain one or more plots
- Axes: individual plots in the figure

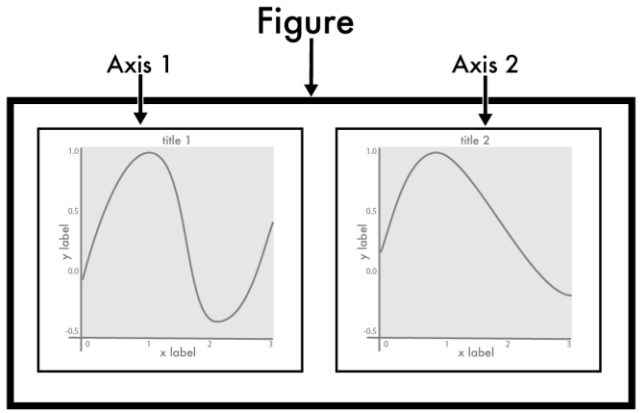
```
# Create a figure containing a single plot (axis)
fig, ax = plt.subplots()
```



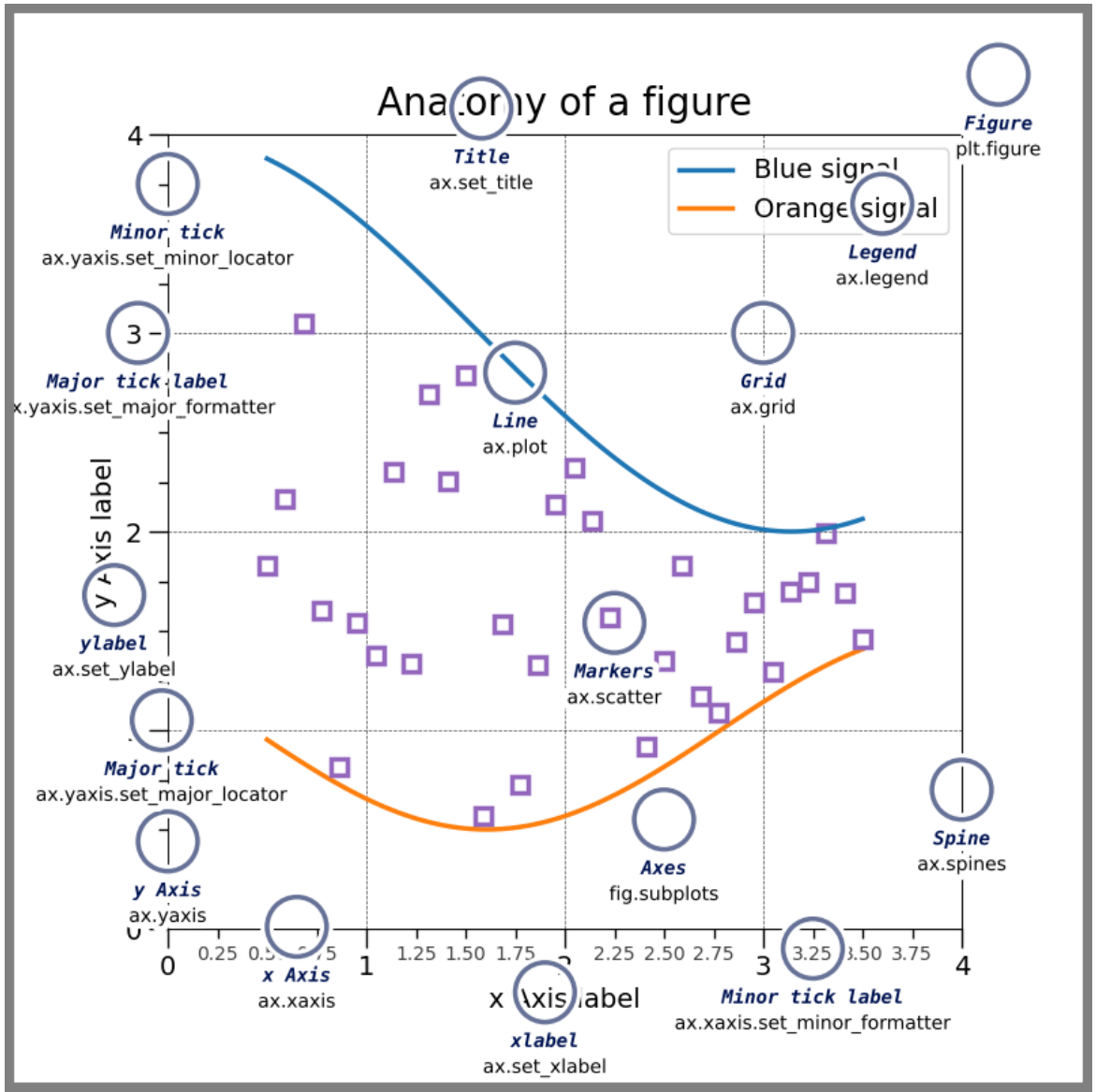



```
# Figure with two plots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 6))
```





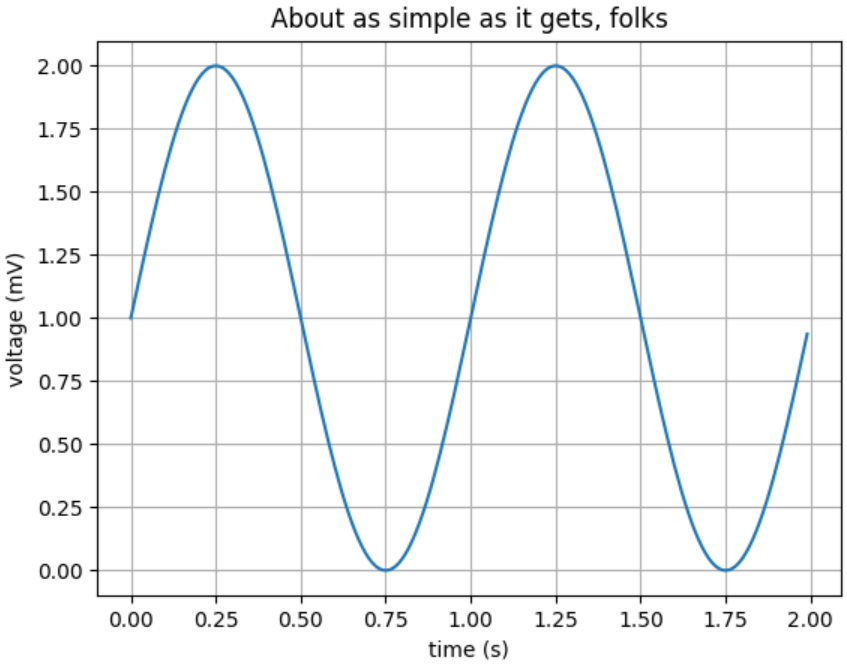
Anatomy of a figure



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()
```



Case study: Iris Flower Dataset

- 50 observations from each of three species of iris flowers
  - Iris Setosa, Iris versicolor, Iris virginica
- 4 features
  - sepal length, sepal width, petal length, petal width

Iris setosa

Iris versicolor

Iris virginica

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

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
iris.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

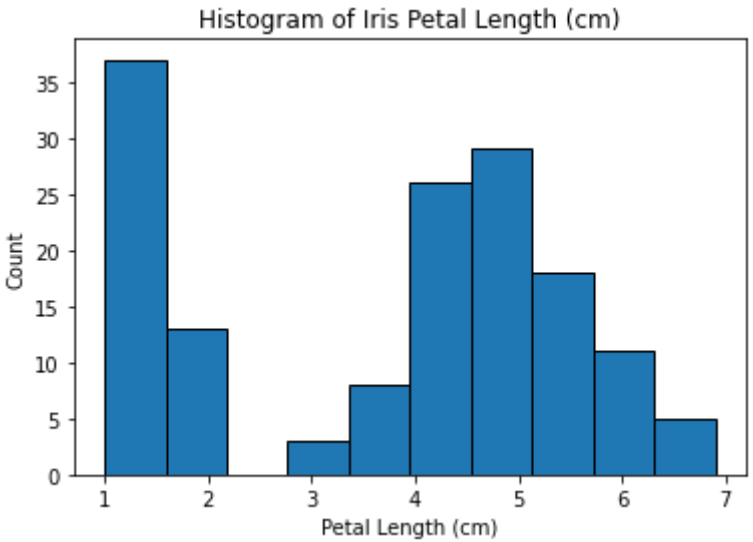
Histogram

Using pyplot

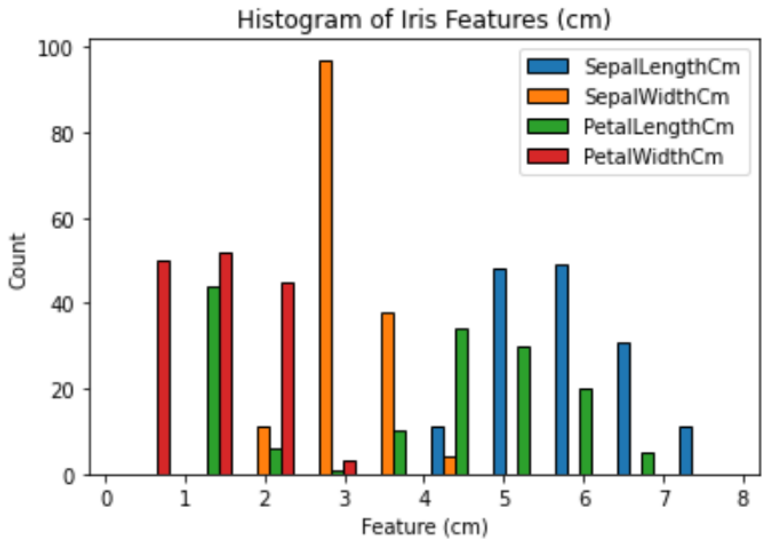
```
plt.hist(iris['PetalLengthCm'], bins=10, edgecolor='black')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Count')
plt.title('Histogram of Iris Petal Length (cm)')
plt.show()
```

Using axes

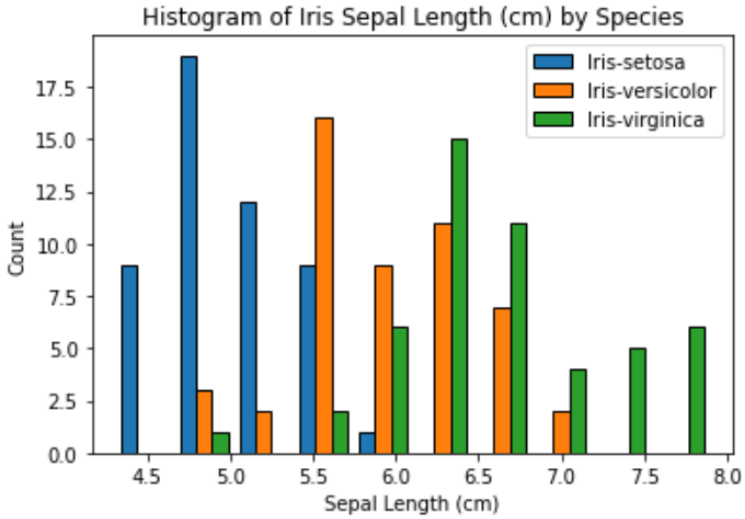
```
fig, axes = plt.subplots()
axes.hist(iris['PetalLengthCm'], bins=10, edgecolor='black')
axes.set_xlabel('Petal Length (cm)')
axes.set_ylabel('Count')
axes.set_title('Histogram of Iris Petal Length (cm)')
plt.show()
```

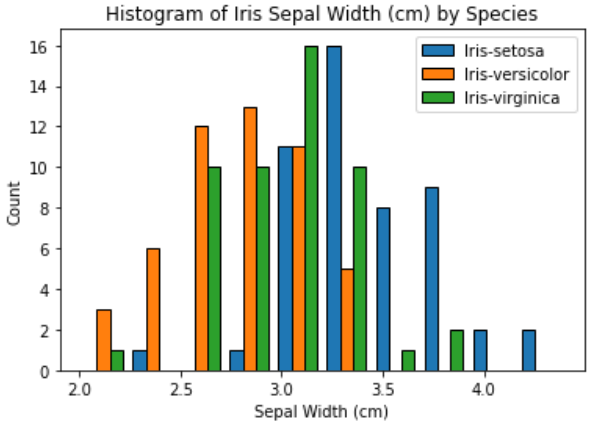


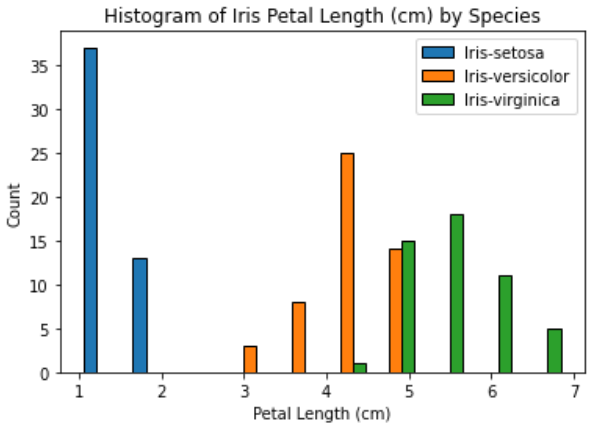
```
plt.hist(iris.loc[:, 'SepalLengthCm':'PetalWidthCm'], bins=10, edgecolor='black', label=iris_features)
plt.legend()
plt.xlabel('Feature (cm)')
plt.ylabel('Count')
plt.title('Histogram of Iris Features (cm)')
plt.show()
```

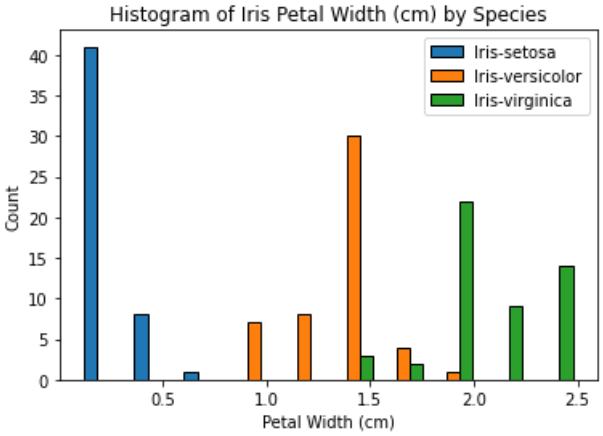


```
SepalLengthCm_groupby_Species = [x['SepalLengthCm'] for _, x in iris[['Species', 'SepalLengthCm']].groupby('Species')]

plt.hist(SepalLengthCm_groupby_Species, bins=10, edgecolor='black', label=iris_species)
plt.legend()
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Count')
plt.title('Histogram of Iris Sepal Length (cm) by Species')
plt.show()
```







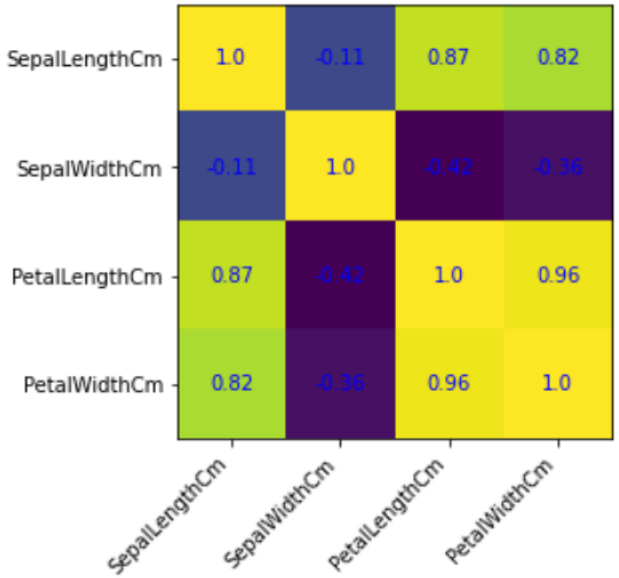


Heatmap (imshow)

```
correlation = iris.loc[:, 'SepalLengthCm':'PetalWidthCm'].corr()
correlation
```

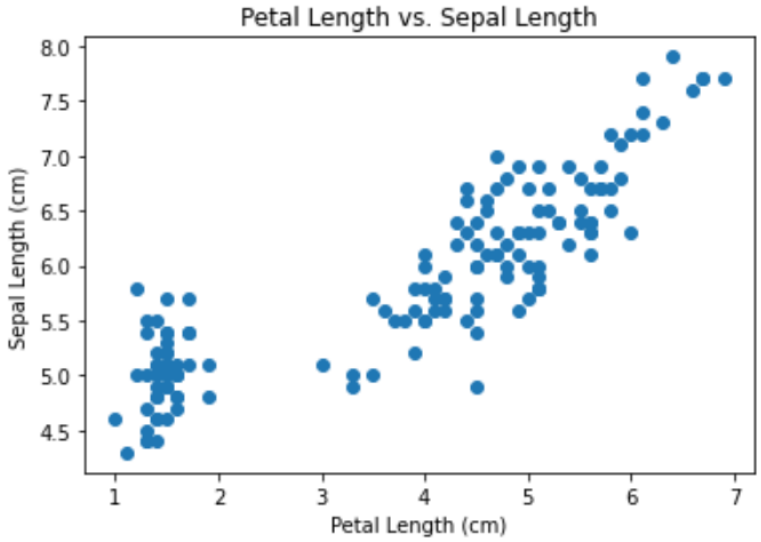
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

```
fig, ax = plt.subplots()
ax.imshow(correlation)
ax.set_xticks(np.arange(len(correlation)), correlation.index)
ax.set_yticks(np.arange(len(correlation)), correlation.index)
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right")
# Loop over data dimensions and create text annotations.
for i in range(len(correlation)):
    for j in range(len(correlation)):
        text = ax.text(j, i, round(correlation.iloc[i, j], 2),
                       ha="center", va="center", color="b")
```



Scatter Plot

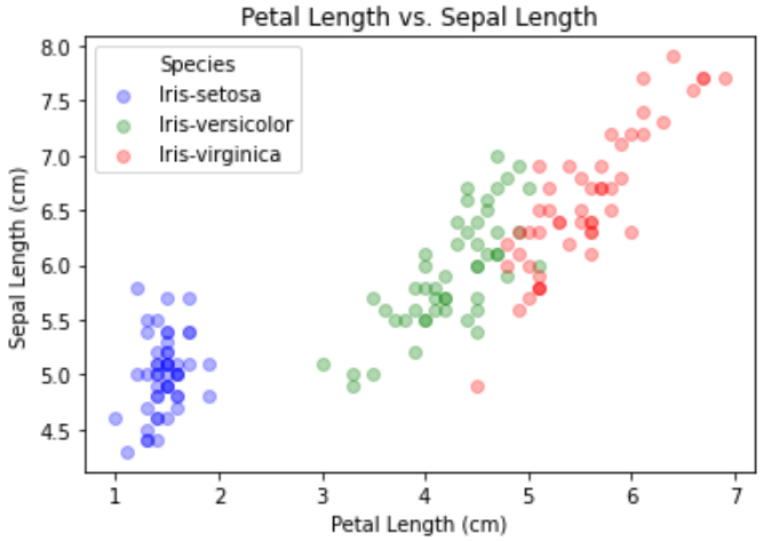
```
plt.scatter(x=iris['PetalLengthCm'], y=iris['SepalLengthCm'])
plt.xlabel('Petal Length (cm)')
plt.ylabel('Sepal Length (cm)')
plt.title('Petal Length vs. Sepal Length')
plt.show()
```



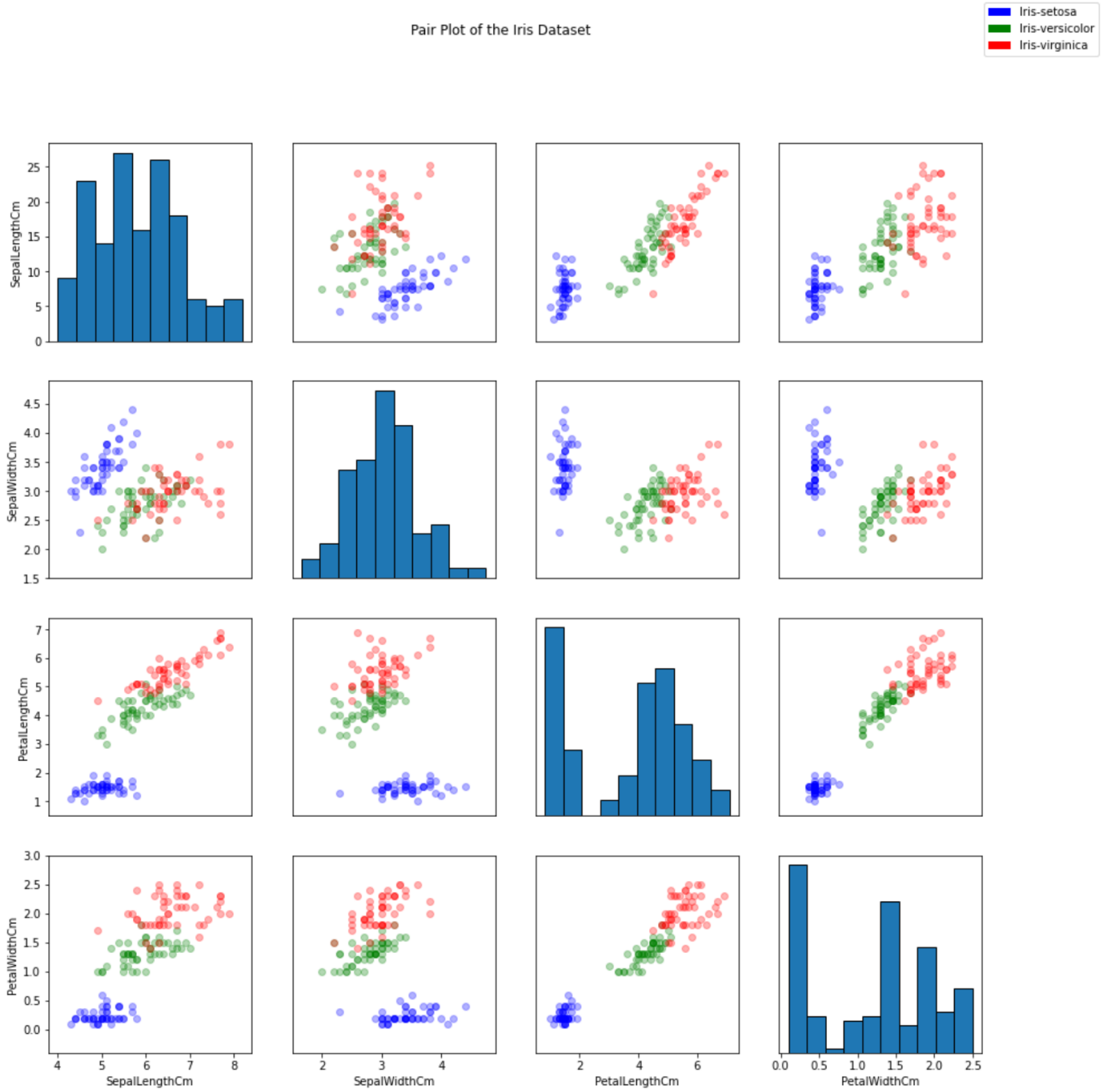
```
color_map = dict(zip(iris_species, ['blue', 'green', 'red']))
color_map
```

```
{'Iris-setosa': 'blue', 'Iris-versicolor': 'green', 'Iris-virginica': 'red'}
```

```
for species, group in iris.groupby('Species'):
    plt.scatter(group['PetalLengthCm'], group['SepalLengthCm'],
                color=color_map[species],
                alpha=0.3, edgecolor=None,
                label=species)

plt.legend(frameon=True, title='Species')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Sepal Length (cm)')
plt.title('Petal Length vs. Sepal Length')
plt.show()
```



```
import matplotlib.patches as mpatches

n = len(iris_features)
figure, axes = plt.subplots(n, n, figsize=(15, 15))

for i in range(n):
    for j in range(n):
        if i == j:
            axes[i, j].hist(iris[iris_features[i]], bins=10, edgecolor='black')
        else:
            for species, group in iris.groupby('Species'):
                axes[i, j].scatter(group[iris_features[j]], group[iris_features[i]],
                                   color=color_map[species],
                                   alpha=0.3, edgecolor=None,
                                   label=species)
            axes[i, j].set_xlim([features_min[iris_features[j]]-0.5, features_max[iris_features[j]]+0.5])
            axes[i, j].set_ylim([features_min[iris_features[i]]-0.5, features_max[iris_features[i]]+0.5])
        axes[i, j].get_xaxis().set_visible(False)
        axes[i, j].get_yaxis().set_visible(False)
        if (i == n-1):
            axes[i, j].get_xaxis().set_visible(True)
            axes[i, j].set_xlabel(iris_features[j])
        if (j == 0):
            axes[i, j].get_yaxis().set_visible(True)
            axes[i, j].set_ylabel(iris_features[i])

rect = []
for key, value in color_map.items():
    rect.append(mpatches.Rectangle((0, 0), 1, 1, fc=value))

figure.legend(rect, iris_species)
figure.suptitle("Pair Plot of the Iris Dataset")

plt.show()
```


Pair Plot of the Iris Dataset
Iris-setosa
Iris-versicolor
Iris-virginica
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm

KMeans Clustering

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)
cluster = kmeans.fit_predict(iris.loc[:, 'SepalLengthCm':'PetalWidthCm'])
```

```
cluster
```

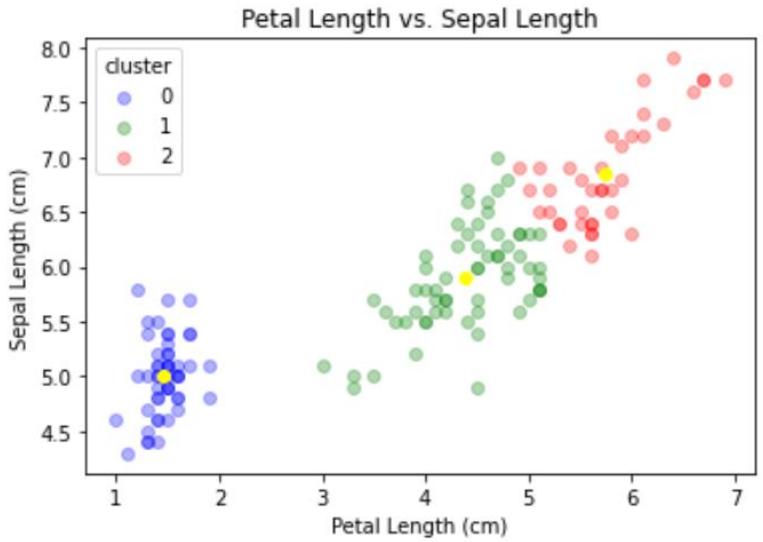
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

```
iris['cluster'] = cluster
```

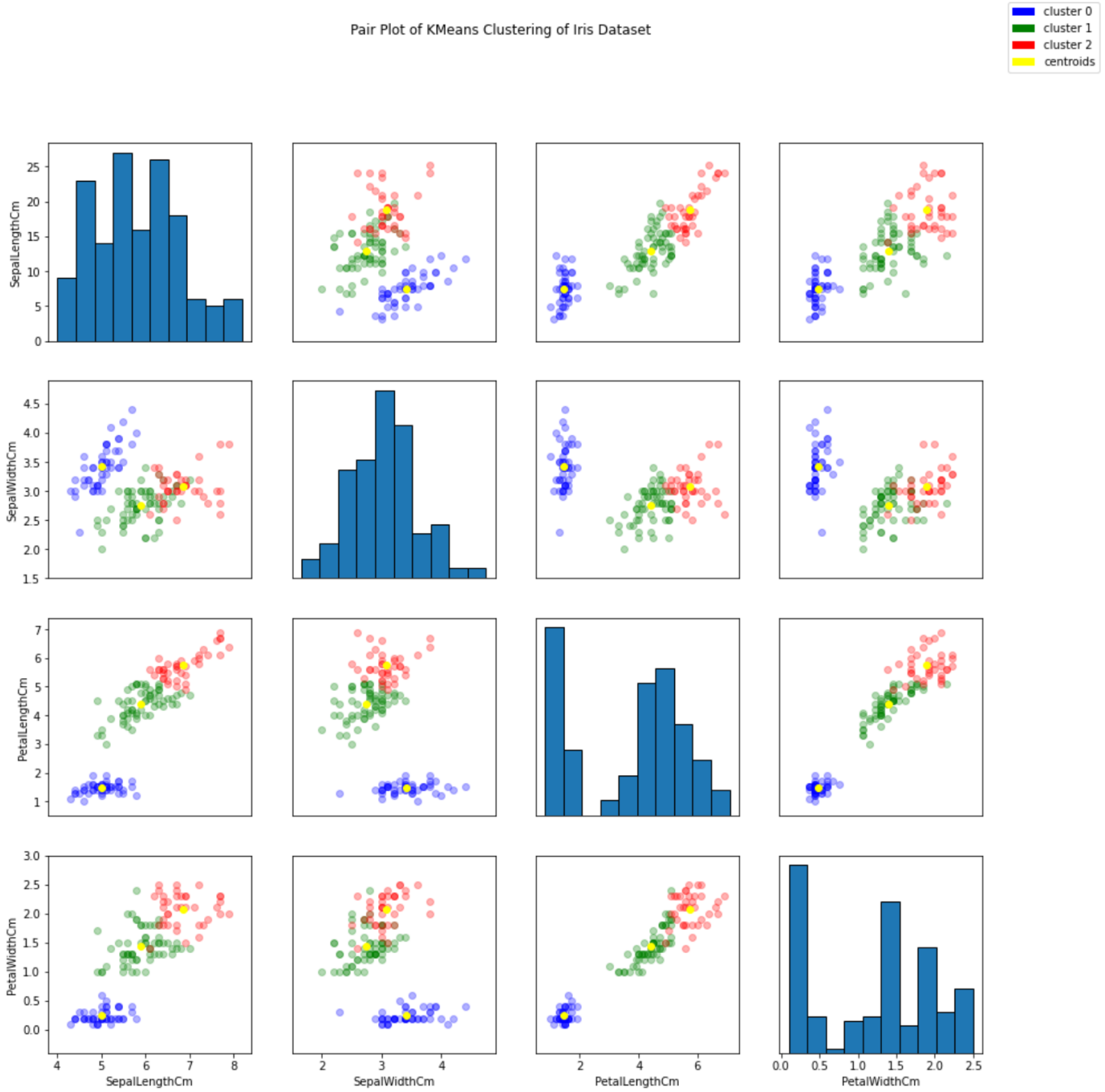
```
print(kmeans.cluster_centers_)
```

```
[[6.85       3.07368421 5.74210526 2.07105263]
 [5.006      3.418      1.464      0.244     ]
 [5.9016129  2.7483871  4.39354839 1.43387097]]
```

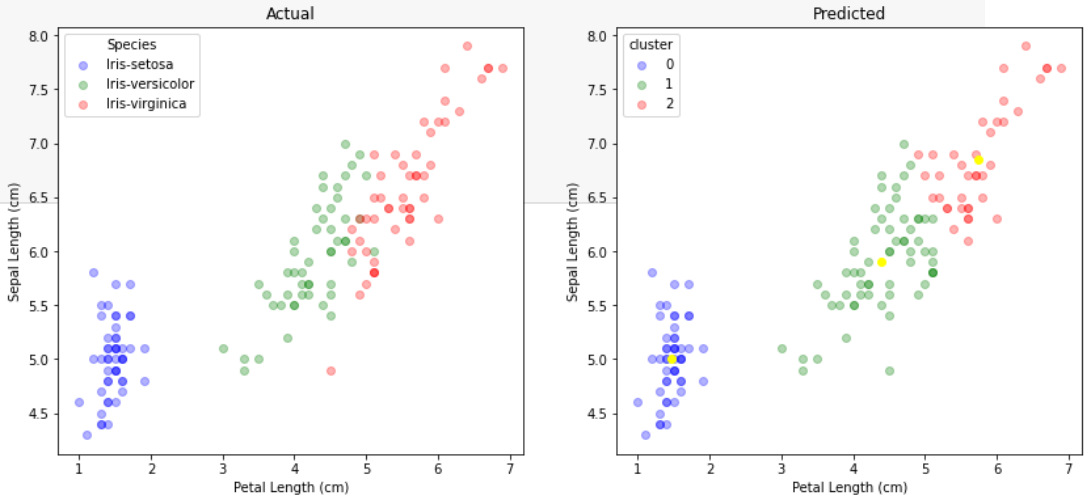
```
color_map = ['blue', 'green', 'red', 'yellow']

for cluster, group in iris.groupby('cluster'):
    plt.scatter(group['PetalLengthCm'], group['SepalLengthCm'],
                color=color_map[cluster],
                alpha=0.3, edgecolor=None,
                label=cluster)
    plt.scatter(kmeans.cluster_centers_[cluster, 2], kmeans.cluster_centers_[cluster, 0],
                                    color=color_map[3])
plt.legend(frameon=True, title='cluster')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Sepal Length (cm)')
plt.title('Petal Length vs. Sepal Length')
plt.show()
```



```
import matplotlib.patches as mpatches

n = len(iris_features)
figure, axes = plt.subplots(n, n, figsize=(15, 15))

color_map = ['blue', 'green', 'red', 'yellow']

for i in range(n):
    for j in range(n):
        if i == j:
            axes[i, j].hist(iris[iris_features[i]], bins=10, edgecolor='black')
        else:
            for cluster, group in iris.groupby('cluster'):
                axes[i, j].scatter(group[iris_features[j]], group[iris_features[i]],
                                   color=color_map[cluster],
                                   alpha=0.3, edgecolor=None,
                                   label=cluster)
                axes[i, j].scatter(kmeans.cluster_centers_[cluster, j], kmeans.cluster_centers_[cluster, i],
                                    color=color_map[3], label='centroids')
            axes[i, j].set_xlim([features_min[iris_features[j]]-0.5, features_max[iris_features[j]]+0.5])
            axes[i, j].set_ylim([features_min[iris_features[i]]-0.5, features_max[iris_features[i]]+0.5])
        axes[i, j].get_xaxis().set_visible(False)
        axes[i, j].get_yaxis().set_visible(False)
        if (i == n-1):
            axes[i, j].get_xaxis().set_visible(True)
            axes[i, j].set_xlabel(iris_features[j])
        if (j == 0):
            axes[i, j].get_yaxis().set_visible(True)
            axes[i, j].set_ylabel(iris_features[i])

rect = []
for color in color_map:
    rect.append(mpatches.Rectangle((0, 0), 1, 1, fc=color))

figure.legend(rect, ['cluster 0', 'cluster 1', 'cluster 2', 'centroids'])
figure.suptitle("Pair Plot of KMeans Clustering of Iris Dataset")

plt.show()
```


Pair Plot of KMeans Clustering of Iris Dataset
cluster 0
cluster 1
cluster 2
centroids
SepalLengthCm
SepalWidthCm
PetalLengthCm
PetalWidthCm

```
# Comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (14, 6))

color_map = dict(zip(iris_species, ['blue', 'green', 'red']))
for species, group in iris.groupby('Species'):
    ax1.scatter(group['PetalLengthCm'], group['SepalLengthCm'],
                color=color_map[species],
                alpha=0.3, edgecolor=None,
                label=species)

ax1.legend(frameon=True, title='Species')
ax1.set_xlabel('Petal Length (cm)')
ax1.set_ylabel('Sepal Length (cm)')
ax1.set_title('Actual')


color_map = ['blue', 'green', 'red', 'yellow']
for cluster, group in iris.groupby('cluster'):
    plt.scatter(group['PetalLengthCm'], group['SepalLengthCm'],
                color=color_map[cluster],
                alpha=0.3, edgecolor=None,
                label=cluster)
    ax2.scatter(kmeans.cluster_centers_[cluster, 2], kmeans.cluster_centers_[cluster, 0],
                                color=color_map[3])
ax2.legend(frameon=True, title='cluster')
ax2.set_xlabel('Petal Length (cm)')
ax2.set_ylabel('Sepal Length (cm)')
ax2.set_title('Predicted')


plt.show()
```



Pandas plot

pandas.DataFrame.plot

pandas.DataFrame.plot.area

pandas.DataFrame.plot.bar

pandas.DataFrame.plot.barh

pandas.DataFrame.plot.box

pandas.DataFrame.plot.density

pandas.DataFrame.plot.hexbin

pandas.DataFrame.plot.hist

pandas.DataFrame.plot.kde

pandas.DataFrame.plot.line

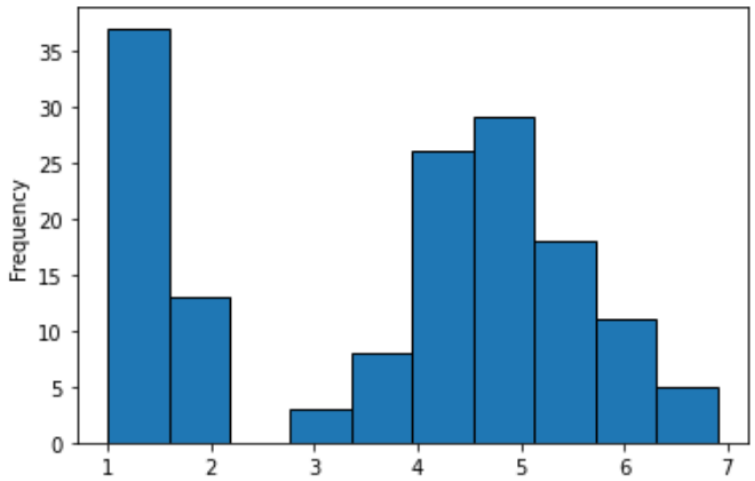
pandas.DataFrame.plot.pie

pandas.DataFrame.plot.scatter

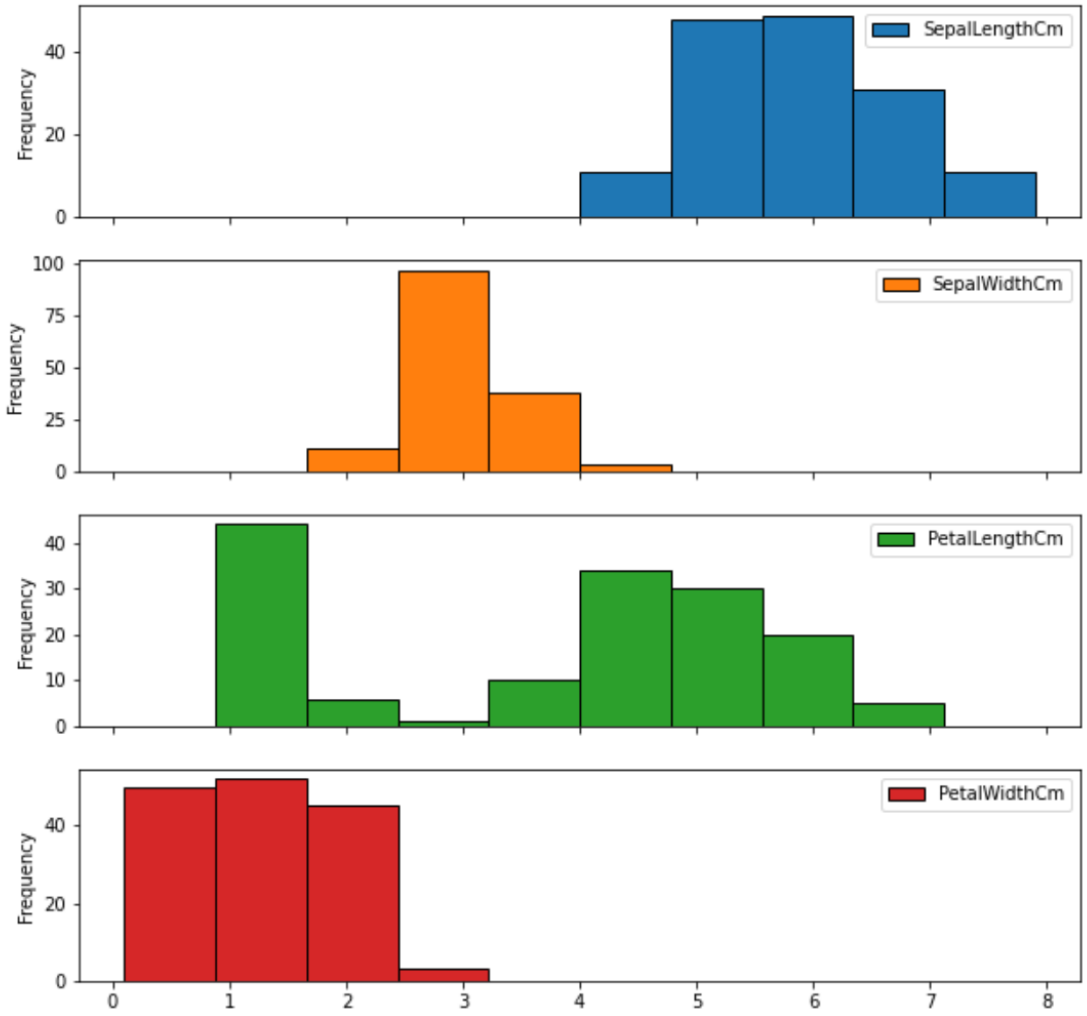
pandas.DataFrame.boxplot

- Pandas provides some basic plot functions.
- Work with Matplotlib
- https://pandas.pydata.org/docs/user_guide/visualization.html

```
iris['PetalLengthCm'].plot.hist(bins=10, edgecolor='black' )
plt.show()
```

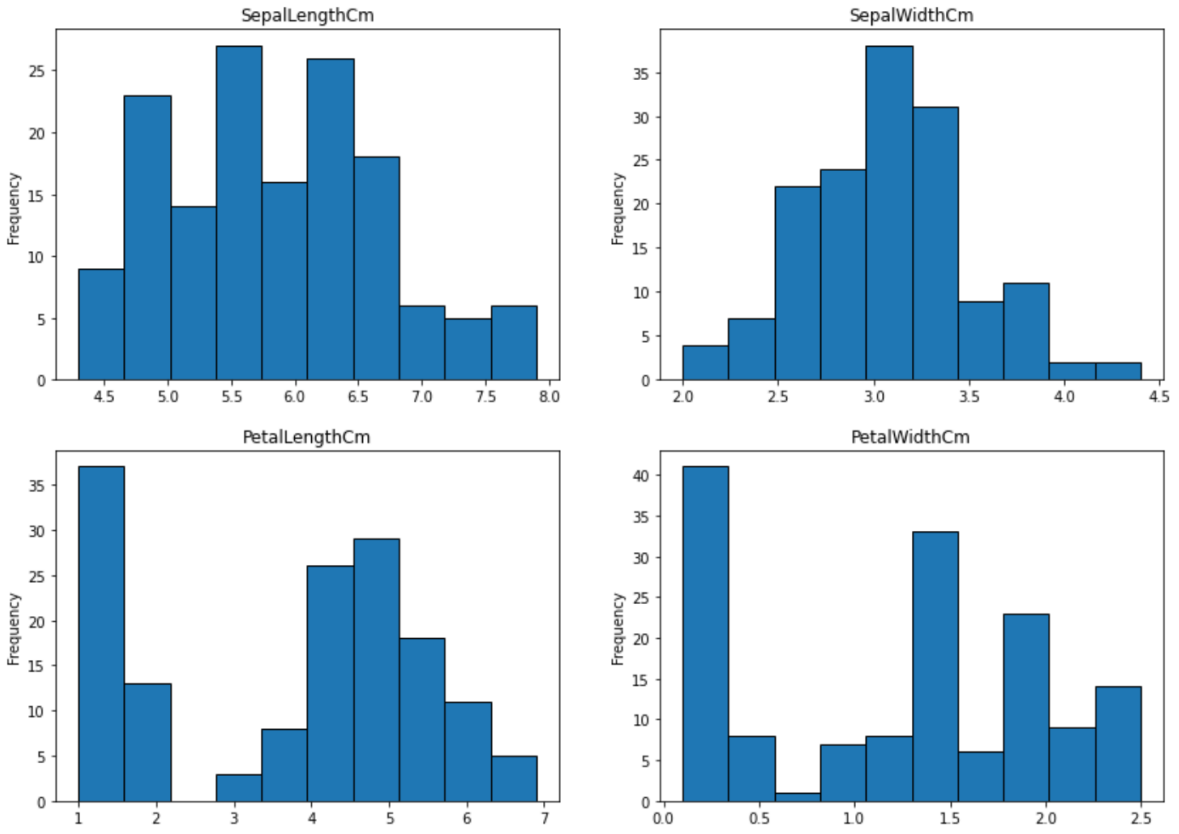


```
iris.loc[:, 'SepalLengthCm':'PetalWidthCm'].plot.hist(bins=10, edgecolor='black', figsize=(10, 10), subplots=True)
plt.show()
```



Pandas plot + 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()
```



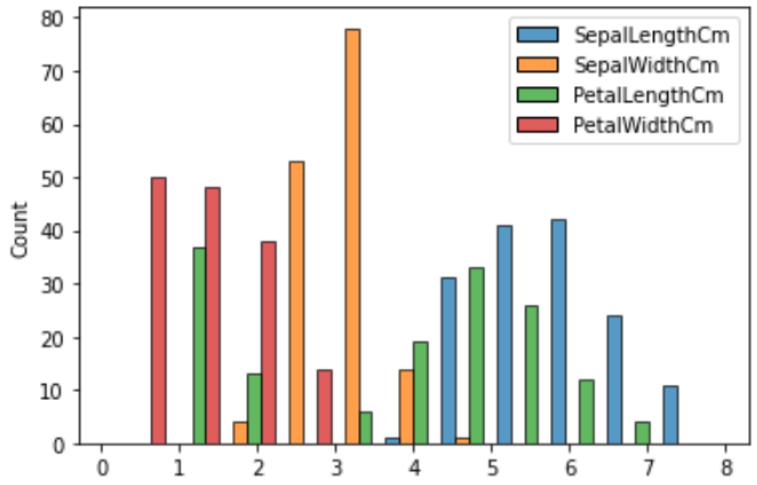
Seaborn

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

```
import seaborn as sns
```

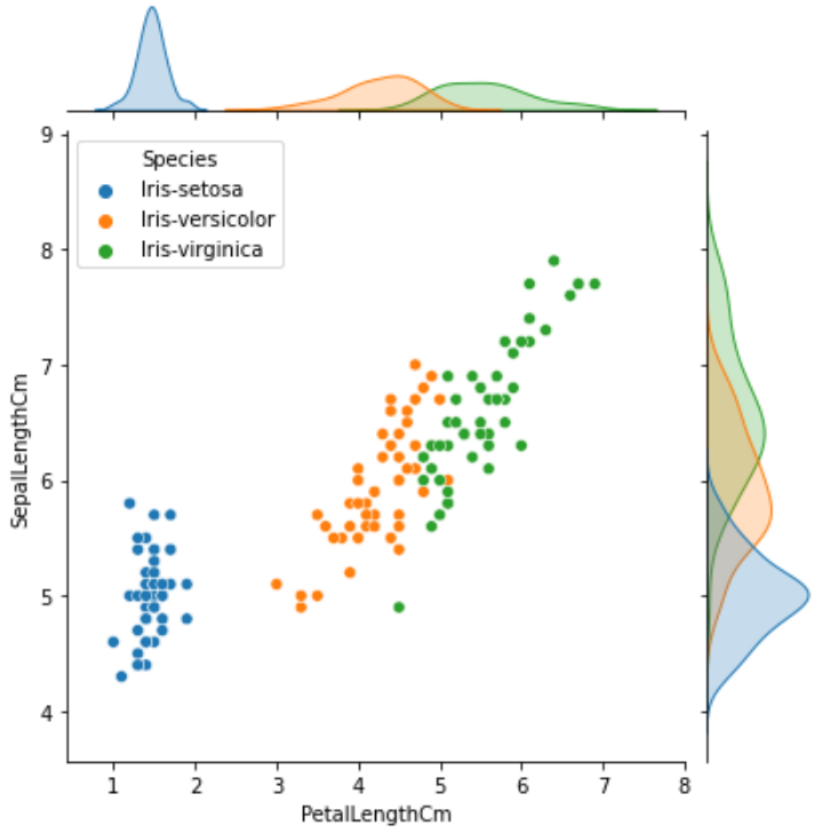
Histogram with Seaborn

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



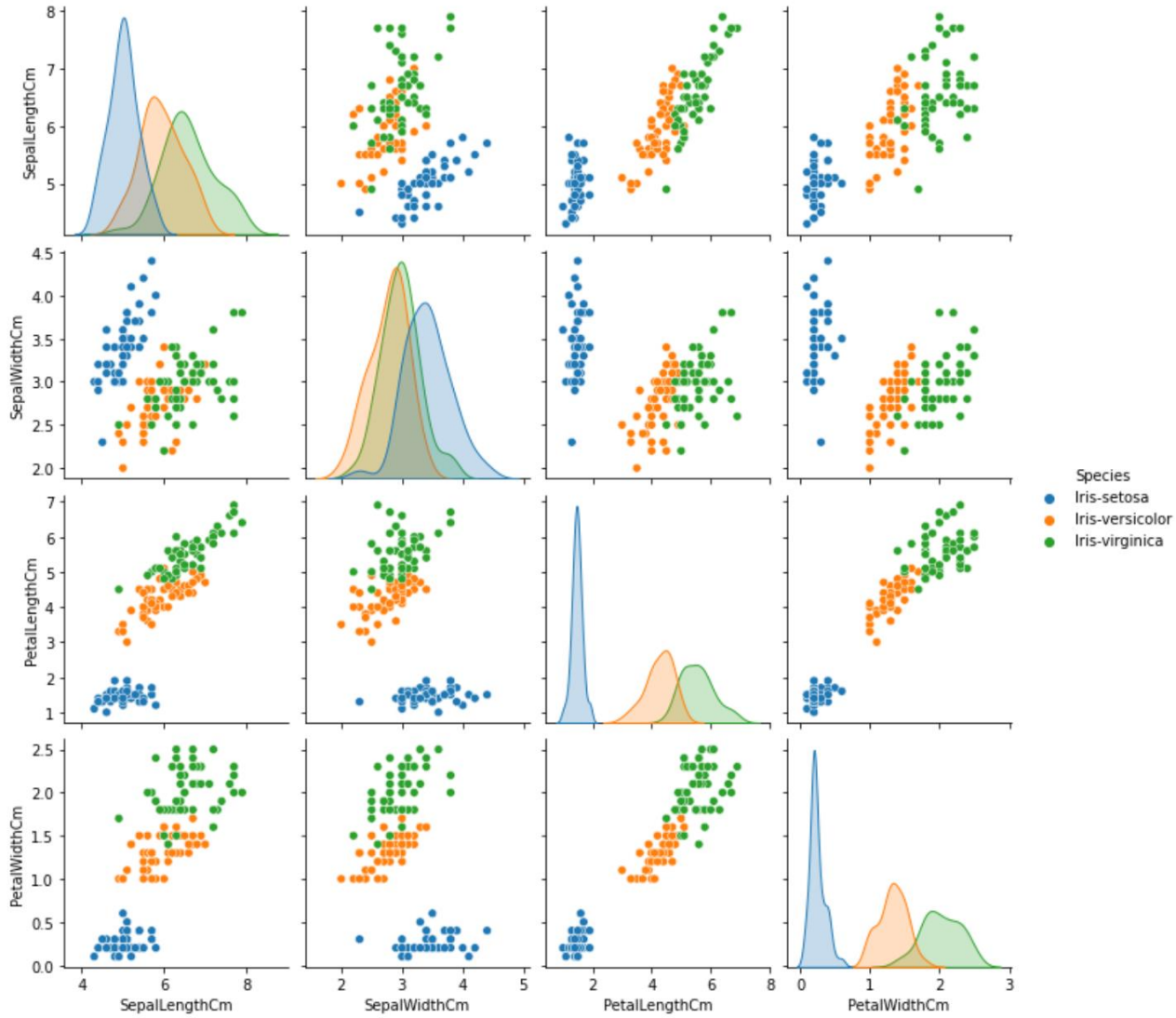
Joint Plot with Seaborn

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



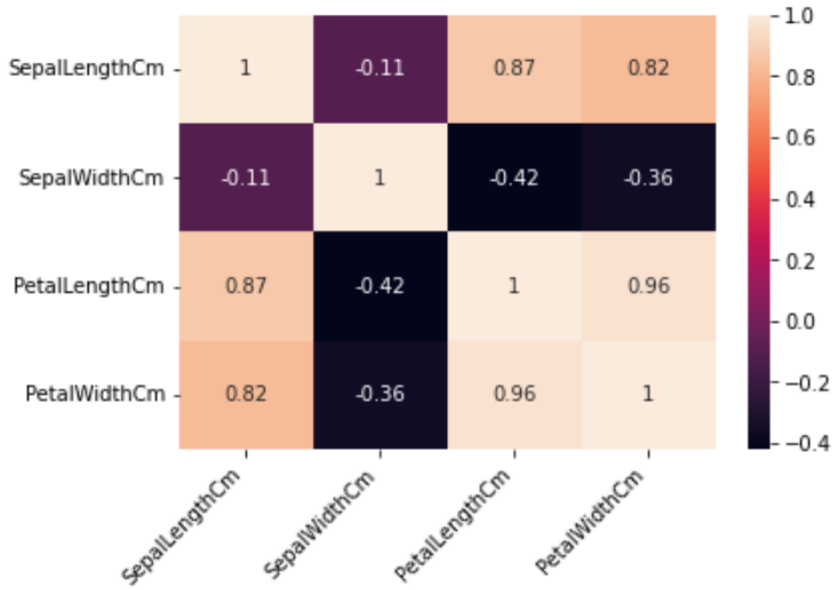
Pair Plot with Seaborn

```
sns.pairplot(iris.loc[:, ['SepalLengthCm','SepalWidthCm','PetalLengthCm', 'PetalWidthCm', 'Species']], hue='Species')
plt.show()
```



Heatmap with Seaborn

```
fig, ax = plt.subplots()
sns.heatmap(correlation, annot=True)
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right")
plt.show()
```

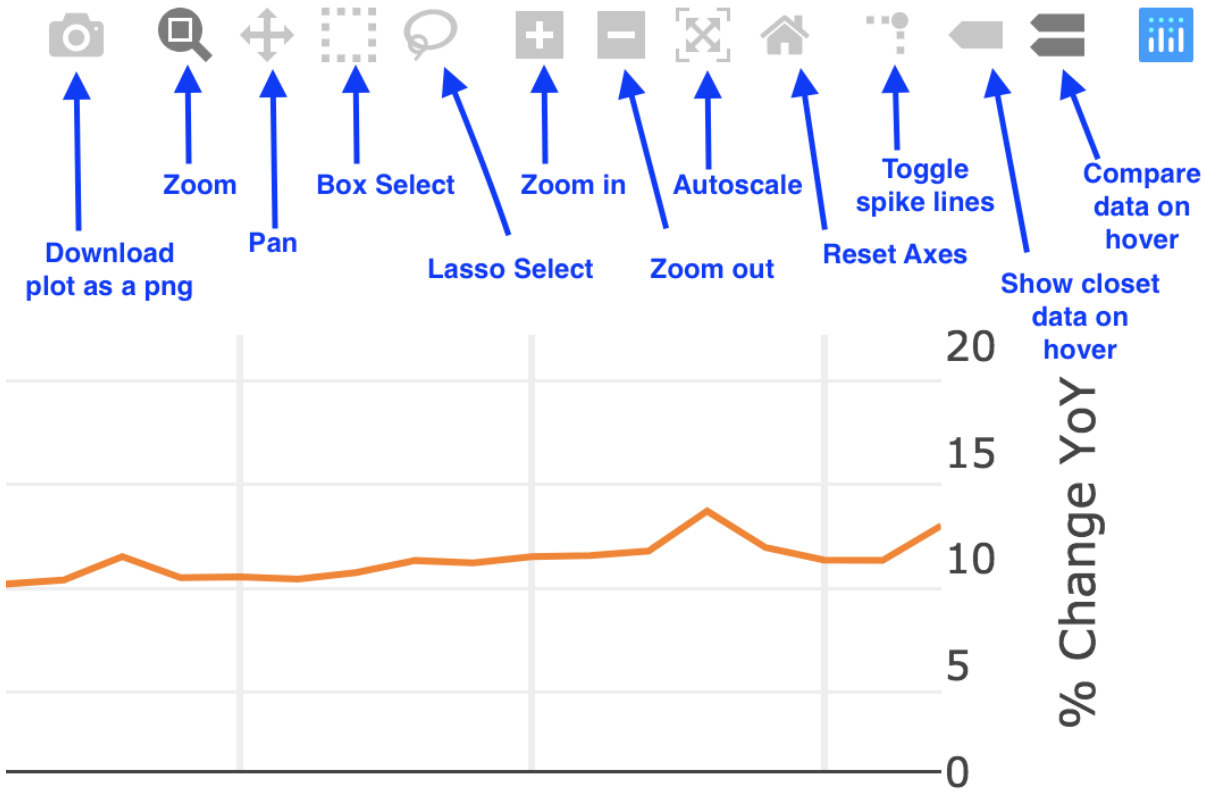


Plotly

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/

Plotly Graph Tools

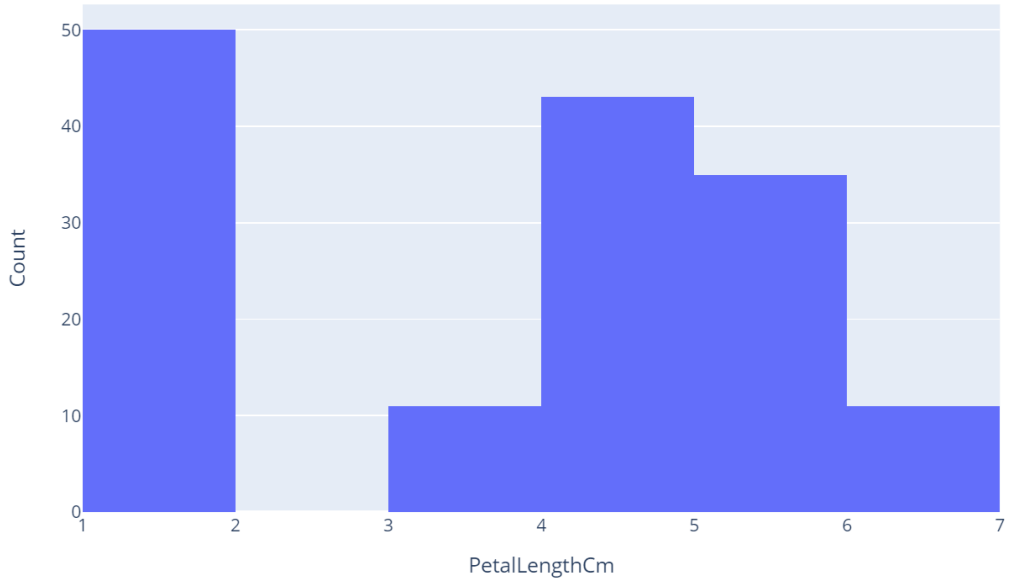


Plotly Components

- Graph Objects (plotly.graph_objects)
  - Figures
  - Traces
  - Layouts
- Figures are represented as instances of plotly.graph_objects.Figure class, and are serialized as text in JSON.

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.
```

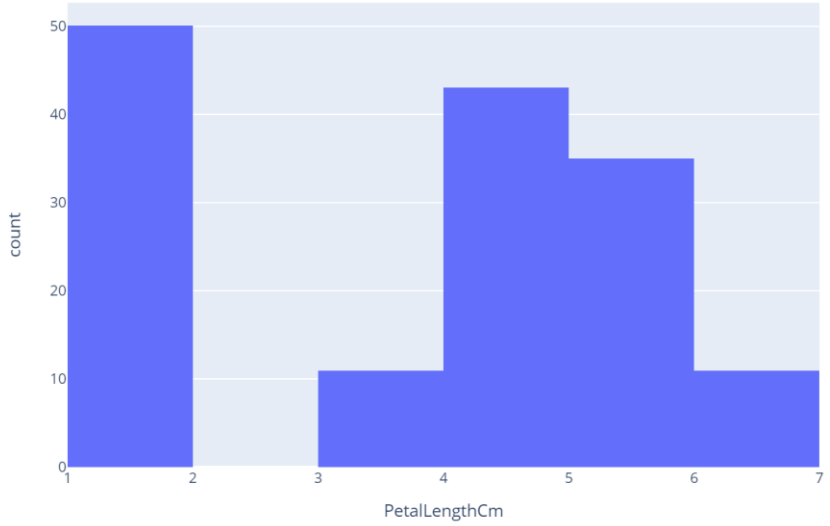


Histogram with Plotly Express

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*)

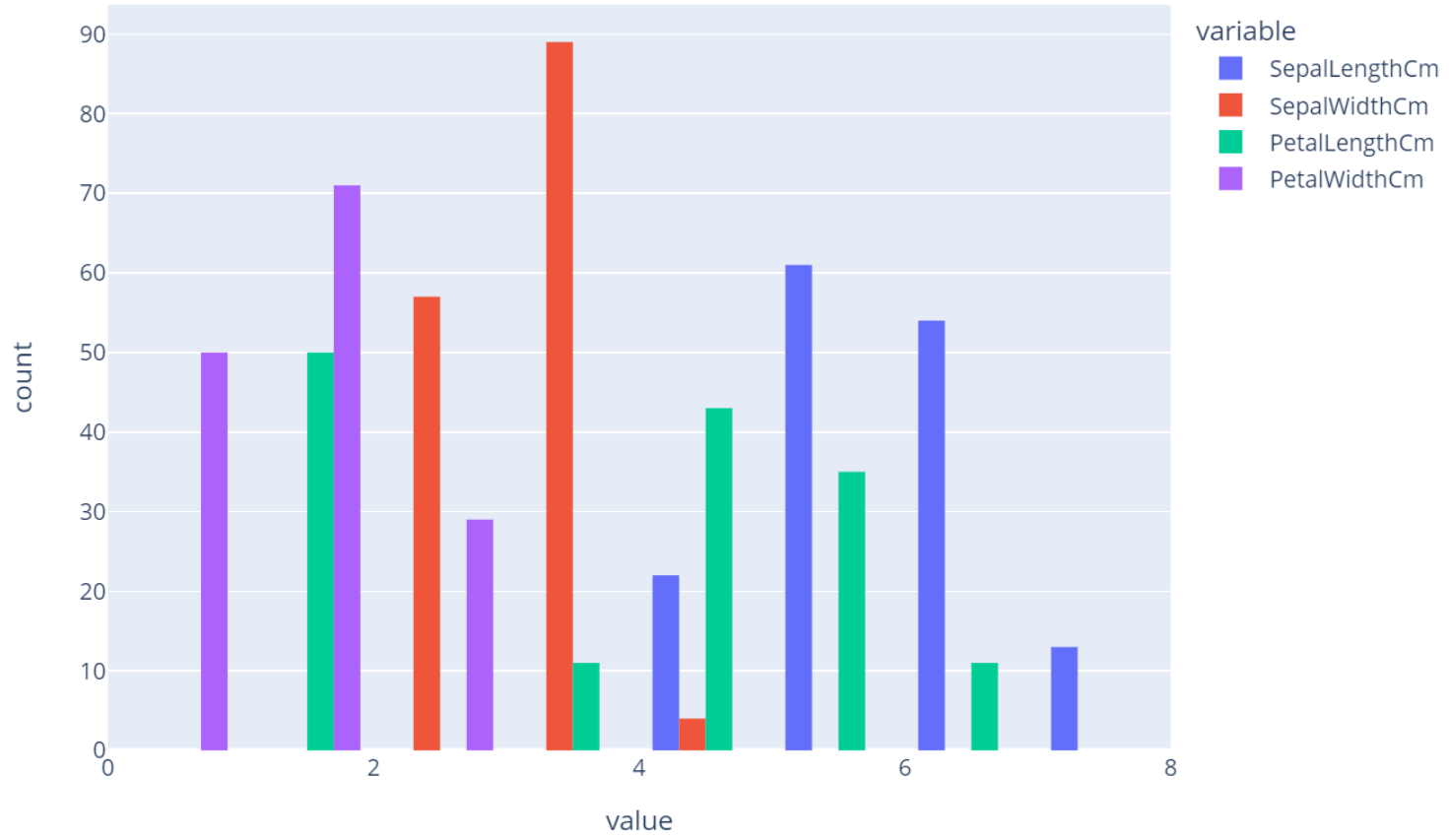
```
import plotly.express as px

px.histogram(iris, x='PetalLengthCm', nbins=10)
```



https://towardsdatascience.com/histograms-with-plotly-express-complete-guide-d483656c5ad7

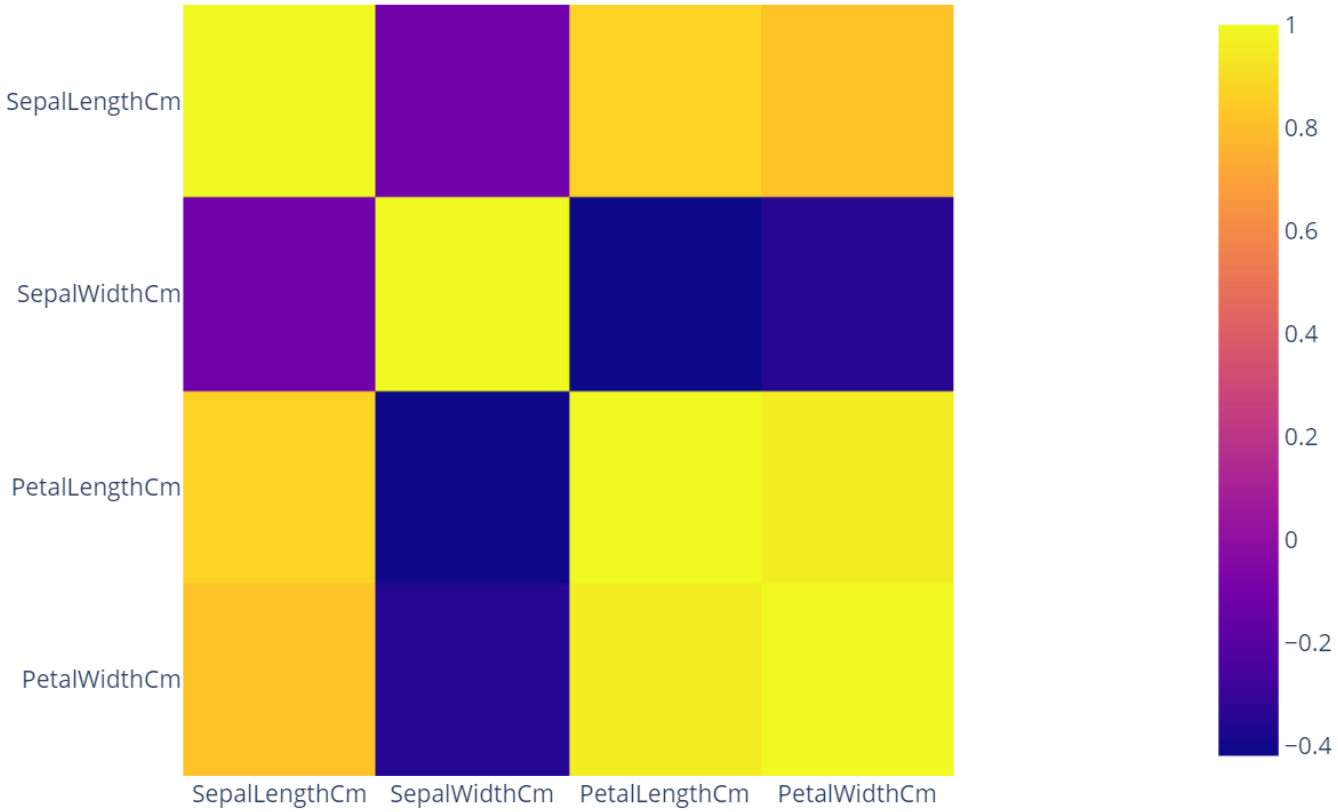
Histogram with Plotly Express

```
px.histogram(iris.loc[:, 'SepalLengthCm':'PetalWidthCm'], nbins=10, barmode="group")
```



Heatmap with Plotly (imshow)

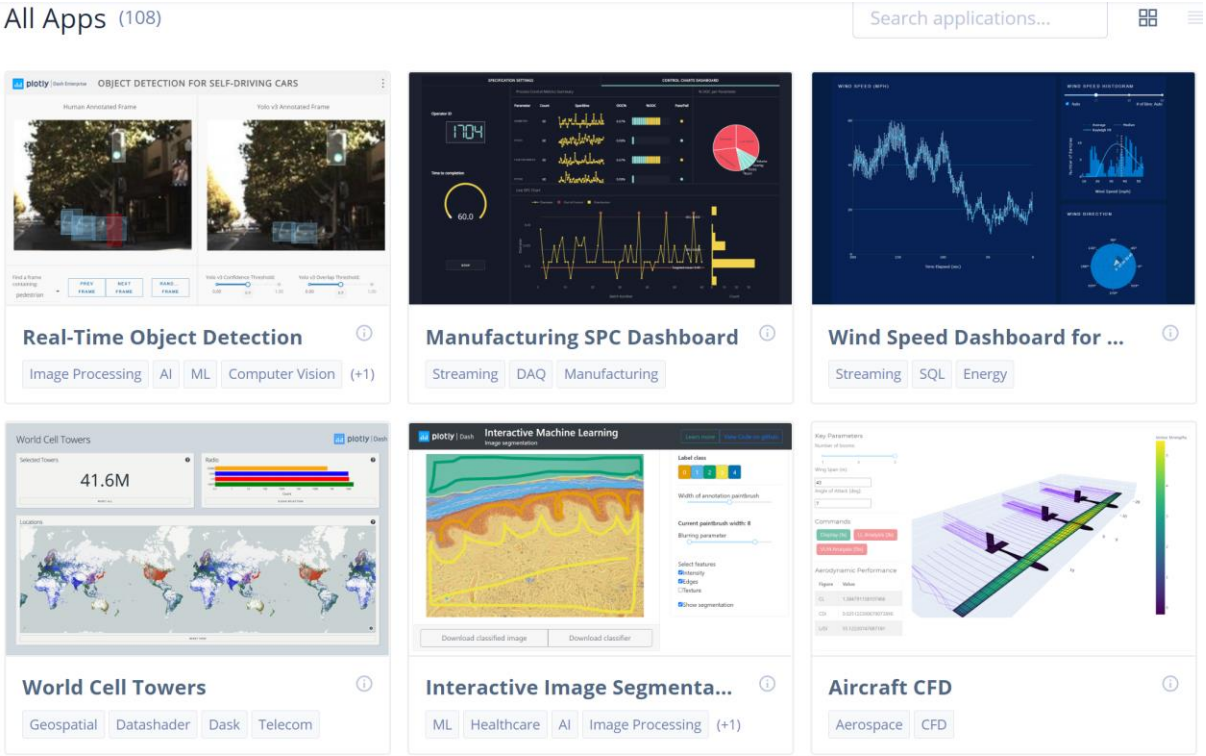
```
fig = px.imshow(correlation)
fig.show()
```



Dash

Dash

- Framework for building analytical applications.
- Based on Plotly
- https://dash.plotly.com/
- https://dash.gallery/Portal/



All Apps (108)

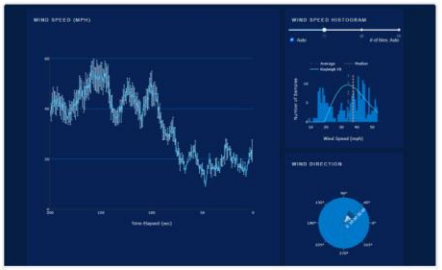
Search applications...

Real-Time Object Detection

Image Processing | AI | ML | Computer Vision | (+1)

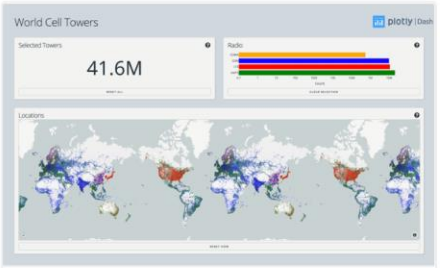
Manufacturing SPC Dashboard

Streaming | DAQ | Manufacturing



Wind Speed Dashboard for ...

Streaming | SQL | Energy

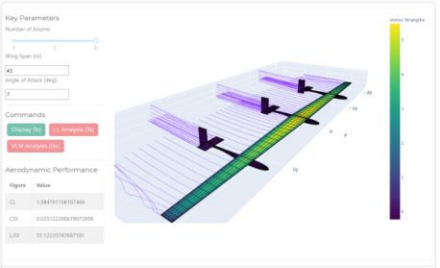


World Cell Towers

Geospatial | Datashader | Dask | Telecom

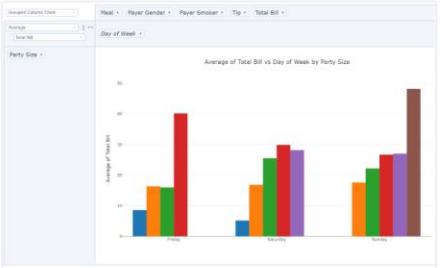
Interactive Image Segmenta...

ML | Healthcare | AI | Image Processing | (+1)



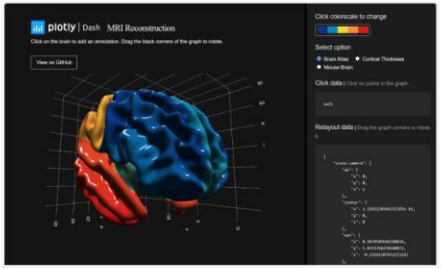
Aircraft CFD

Aerospace | CFD



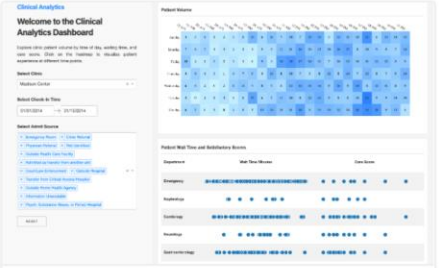
Pivottable Demo

Financial | Insurance



Brain Surface Viewer

Biotechnology | 3d



Clinical Analytics Dashboard

Pharma | Healthcare

Dash Program Structure

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

Simple Dash application

```
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/
```