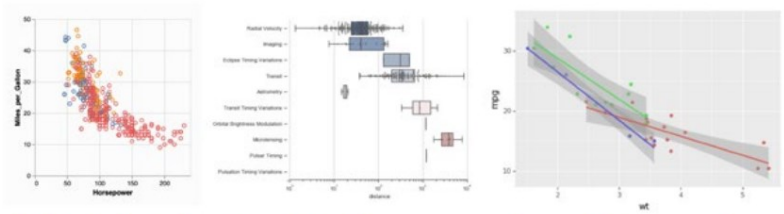
Data Visualization with Python

Veera Muangsin

Plot Types

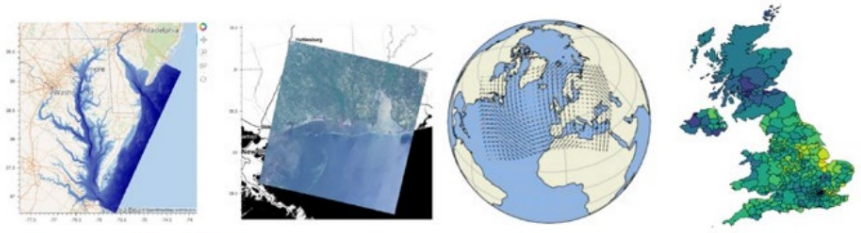
The most basic plot types are shared between multiple libraries, and others are only available in certain libraries.

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.



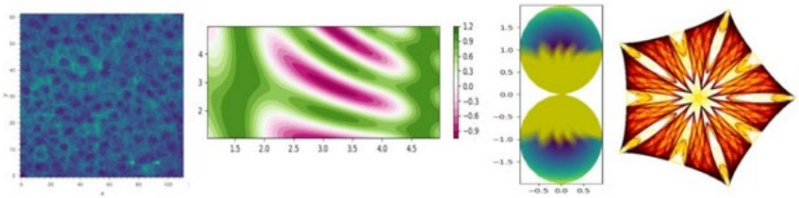
Statistical plots (scatter plots, lines, areas, bars, histograms)

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



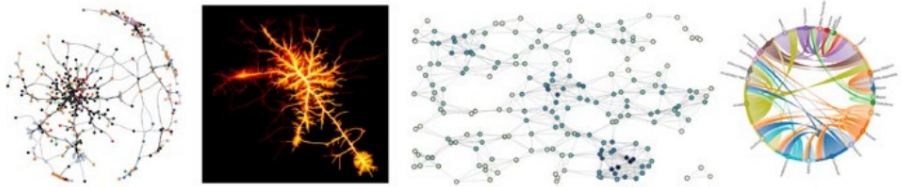
Geographical data

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



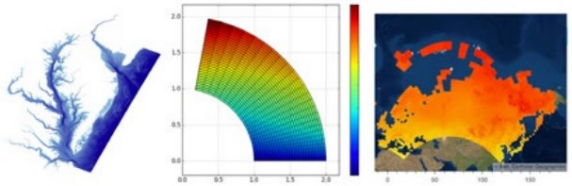
Multidimensional arrays (regular grids, rectangular meshes)

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



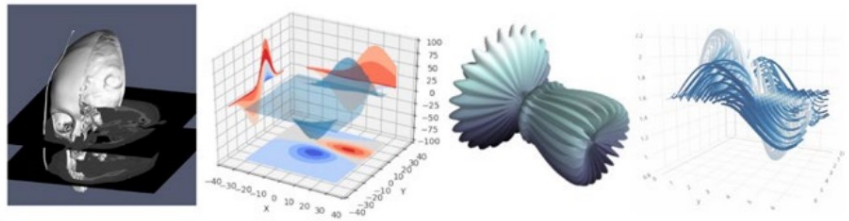
Networks/graphs

NetworkX, Plotly, Bokeh, HoloViews, and Datashader.



Irregular 2D meshes (triangular grids)

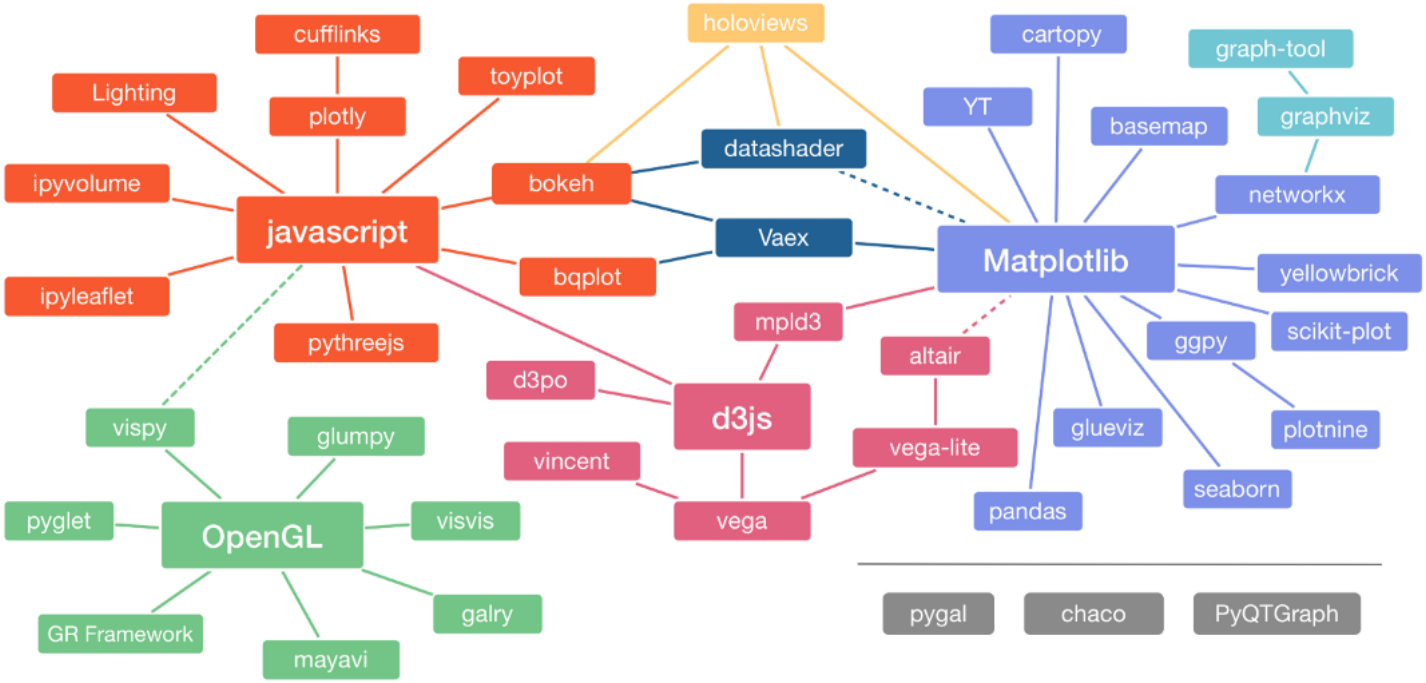
Well supported by the SciVis libraries plus Matplotlib, Bokeh, Datashader, and HoloViews.



3D (meshes, scatter, etc.)

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

Python Visualization Landscape



- Scientific visualization (green)
  - visualize 3D (+time) physical processes.
  - Based on OpenGL
- Information visualization (all other colors)

- Dashboard and app (not in the diagram)
  - Streamlit
  - Dash
  - PyGWalker

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

Python Visualization Packages

- **Matplotlib**
- Matplotlib wrapper
  - **Pandas plot**
  - Seaborn https://seaborn.pydata.org/
  - Scikit-plot https://scikit-plot.readthedocs.io/
  - Plotnine (ggplot): based on R's ggplot2 https://plotnine.org/
- Javascript-based Viz (interactive viz)
  - **Plotly** https://plot.ly/python/
  - Bokeh http://bokeh.org/
- Dashboard web app
  - **Streamlit** https://streamlit.io/
  - Dash (based on Plotly) https://dash.plotly.com/

Matplotlib

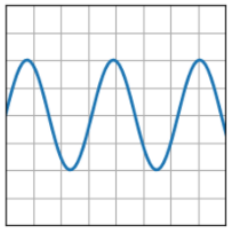
Matplotlib

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

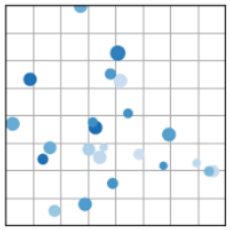
```
import matplotlib.pyplot as plt
```

Plot Types

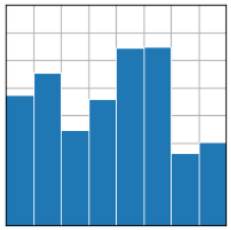
- https://matplotlib.org/stable/plot_types/index.html



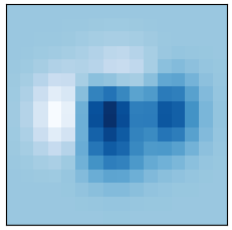
plot(x, y)



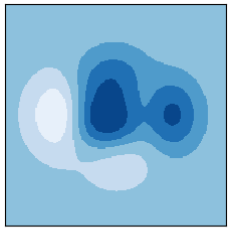
scatter(x, y)



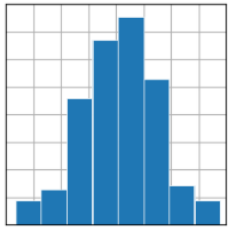
bar(x, height) / barh(y, width)



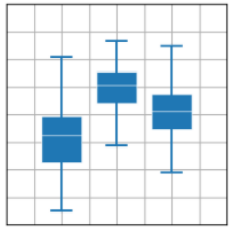
imshow(Z)



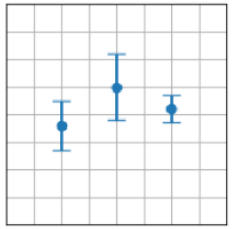
contourf(X, Y, Z)



hist(x)

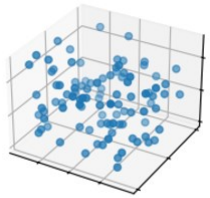


boxplot(X)



errorbar(x, y, yerr, xerr)

pie(x)



scatter(xs, ys, zs)

Matplotlib Gallery

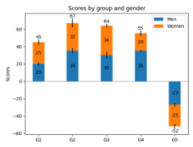
- https://matplotlib.org/stable/gallery/index.html

Examples

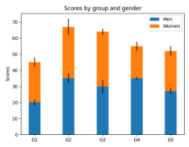
This page contains example plots. Click on any image to see the full image and source code.

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

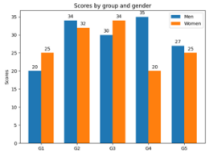
Lines, bars and markers



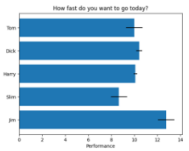
Bar Label Demo



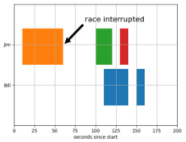
Stacked bar chart



Grouped bar chart with labels

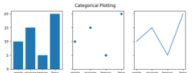


Horizontal bar chart

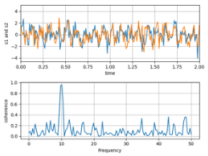


Broken Barh

CapStyle



Plotting categorical variables



Plotting the coherence of two signals

On this page

Lines, bars and markers
Images, contours and fields
Subplots, axes and figures
Statistics
Pie and polar charts
Text, labels and annotations
pyplot
Color
Shapes and collections
Style sheets
axes_grid1
axisartist
Showcase
Animation
Event handling
Front Page
Miscellaneous
3D plotting
Scales
Specialty Plots
Spines
Ticks
Units
Embedding Matplotlib in graphical user interfaces
Userdemo
Widgets

Matplotlib APIs

Two styles:

- 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.
  - More convenient but less flexible than the object-oriented API

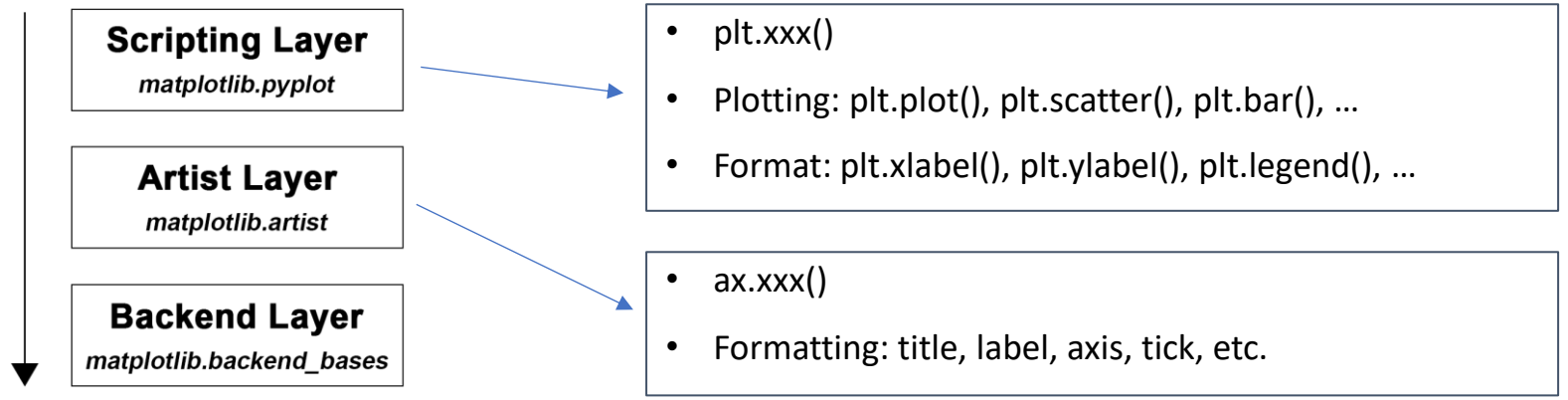
- Object-oriented API
  - Create Figure and Axes objects and call their methods to add content and modify the appearance.
  - matplotlib.figure: axes creation, figure-level content
  - matplotlib.axes: most plotting methods, axis labels, axis styling, etc.
  - More control and customization
  - Recommended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

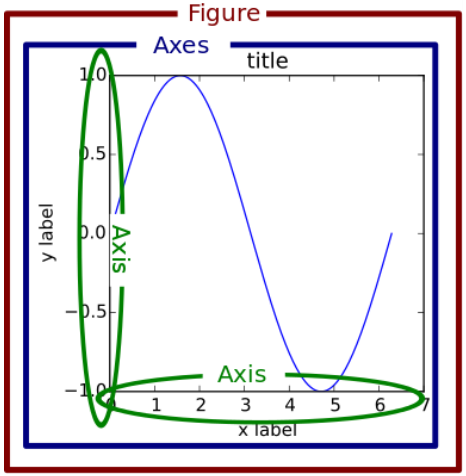
Stick to one style. Mixing pyplot API with OO API can make your code harder to read and maintain.

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

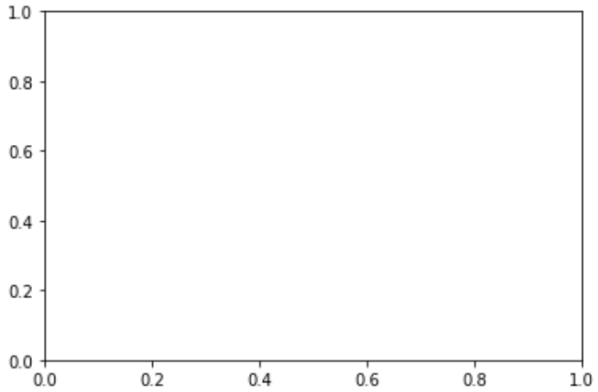


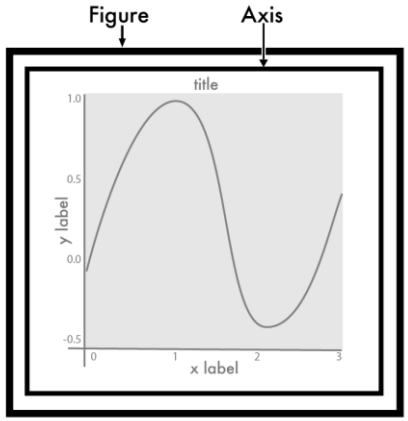


Note: ‘axes’ is plural form of ‘axis’.

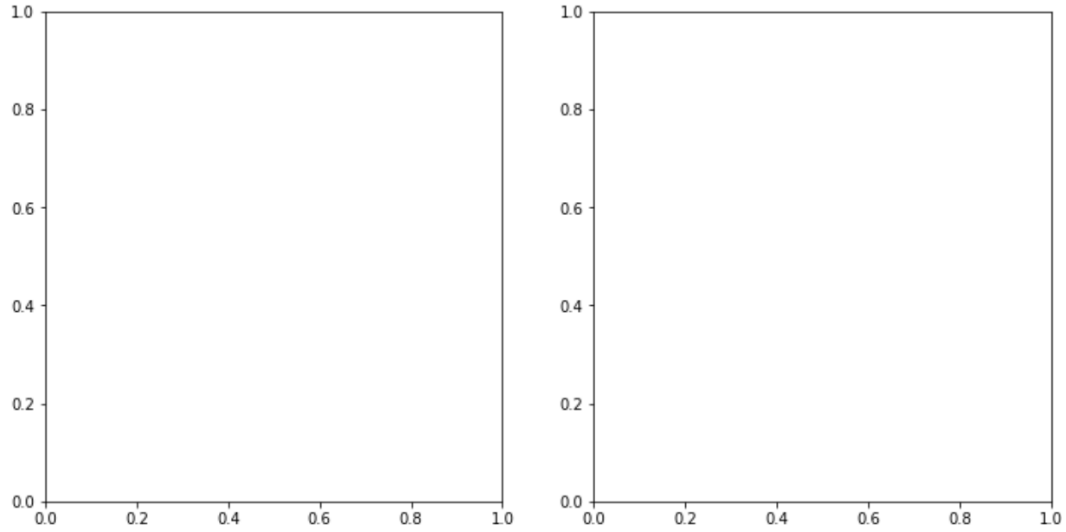
Creating a Figure and one or more Axes using `pyplot.subplots()`

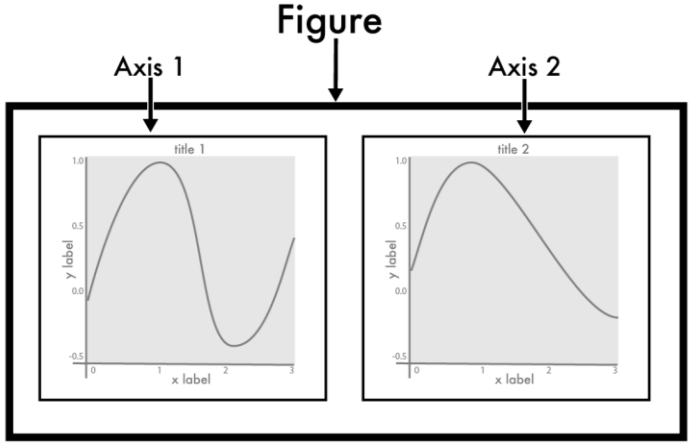
```
# Create a figure containing a single plot (axis)
fig, ax = plt.subplots()
```






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

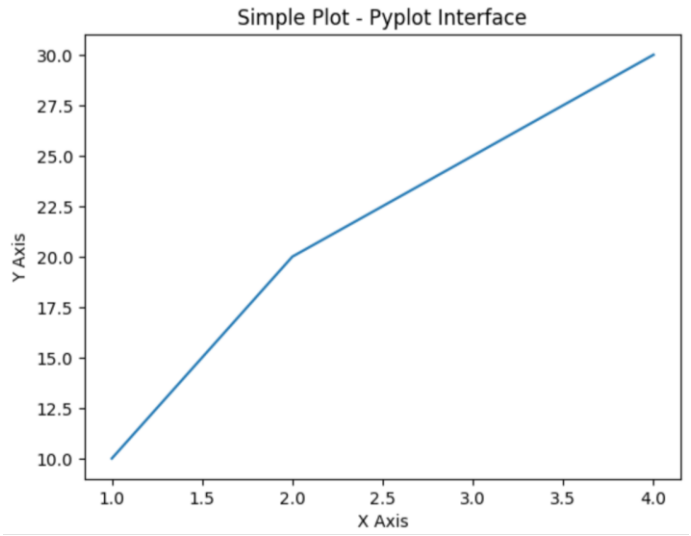


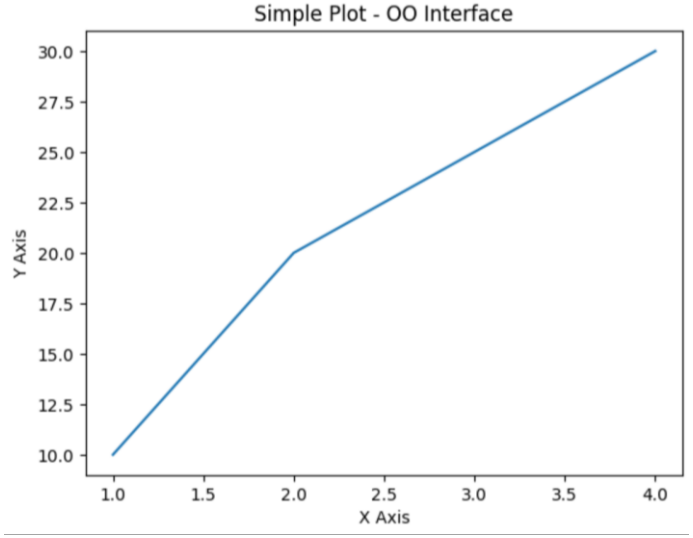


Pyplot vs OO

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

plt.plot(x, y)
plt.title('Simple Plot - Pyplot Interface')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.show()
```

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, y)  # Plot some data on the axes.
ax.set_title('Simple Plot - OO Interface')  # Add a title to the axes.
ax.set_xlabel('X Axis')  # Add an x-label to the axes.
ax.set_ylabel('Y Axis')  # Add a y-label to the axes.
plt.show()
```



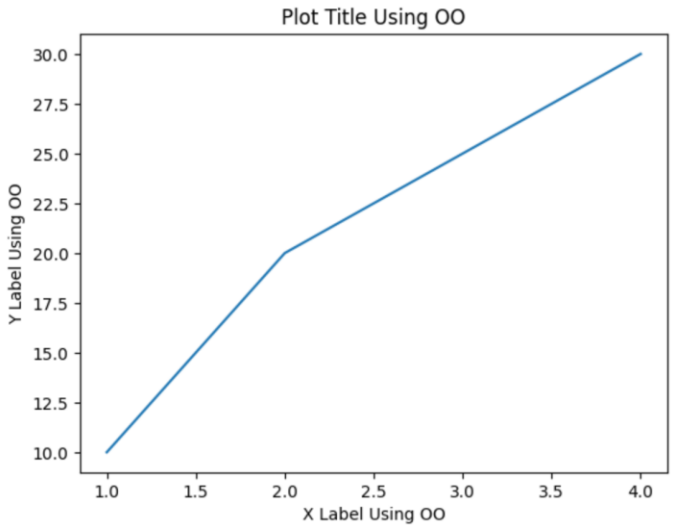


same output

Pyplot + OO (possible but not recommended)

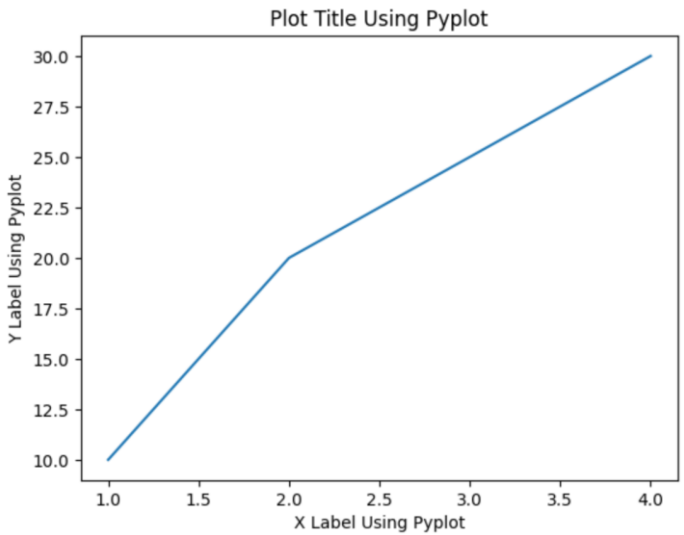
Create with pyplot, modify with OO

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

# Create a plot using Pyplot
plt.plot(x, y)

# Grab the current Axes and Figure using Pyplot, then modify using OO
ax = plt.gca()  # Get current Axes
fig = plt.gcf()  # Get current Figure

# Now, use OO methods
ax.set_title('Plot Title Using OO')
ax.set_xlabel('X Label Using OO')
ax.set_ylabel('Y Label Using OO')

plt.show()
```

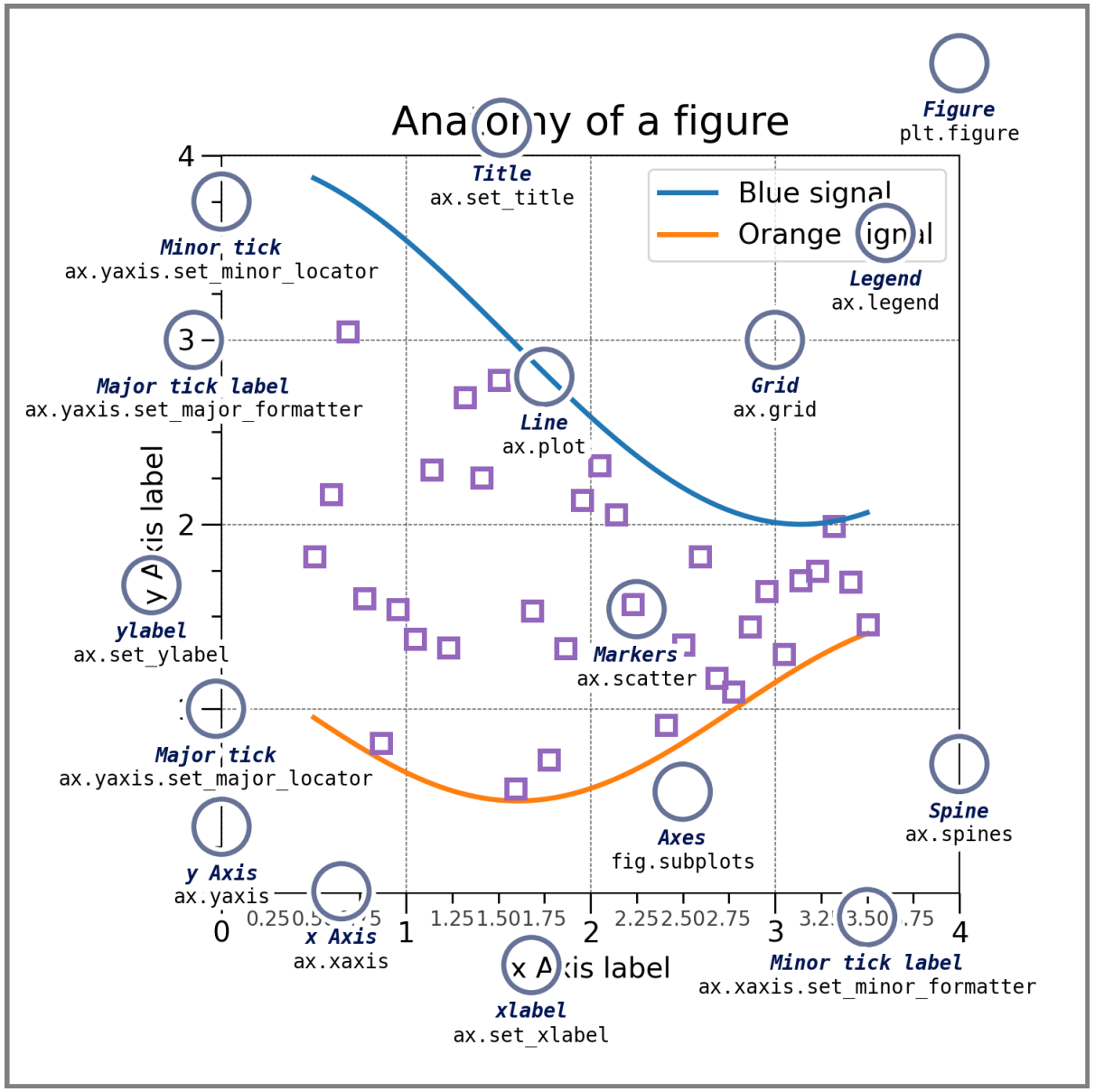


Create with OO, modify with pyplot

```python
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

# Start with OO
fig, ax = plt.subplots()
ax.plot(x, y)

# Use Pyplot to modify, but this affects the current plot
plt.title('Plot Title Using Pyplot')
plt.xlabel('X Label Using Pyplot')
plt.ylabel('Y Label Using Pyplot')

plt.show()
```



Anatomy of a figure



https://matplotlib.org/stable/gallery/showcase/anatomy.html

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

from matplotlib.patches import Circle
from matplotlib.patheffects import withStroke
from matplotlib.ticker import AutoMinorLocator, MultipleLocator

royal_blue = [0, 20/256, 82/256]


# make the figure

np.random.seed(19680801)

X = np.linspace(0.5, 3.5, 100)
Y1 = 3+np.cos(X)
Y2 = 1+np.cos(1+X/0.75)/2
Y3 = np.random.uniform(Y1, Y2, len(X))

fig = plt.figure(figsize=(7.5, 7.5))
ax = fig.add_axes([0.2, 0.17, 0.68, 0.7], aspect=1)

ax.xaxis.set_major_locator(MultipleLocator(1.000))
ax.xaxis.set_minor_locator(AutoMinorLocator(4))
ax.yaxis.set_major_locator(MultipleLocator(1.000))
ax.yaxis.set_minor_locator(AutoMinorLocator(4))
ax.xaxis.set_minor_formatter("{x:.2f}")

ax.set_xlim(0, 4)
ax.set_ylim(0, 4)

ax.tick_params(which='major', width=1.0, length=10, labelsize=14)
ax.tick_params(which='minor', width=1.0, length=5, labelsize=10,
               labelcolor='0.25')

ax.grid(linestyle="--", linewidth=0.5, color='.25', zorder=-10)

ax.plot(X, Y1, c='C0', lw=2.5, label="Blue signal", zorder=10)
ax.plot(X, Y2, c='C1', lw=2.5, label="Orange signal")
ax.plot(X[::3], Y3[::3], linewidth=0, markersize=9,
        marker='s', markerfacecolor='none', markeredgecolor='C4',
        markeredgewidth=2.5)

ax.set_title("Anatomy of a figure", fontsize=20, verticalalignment='bottom')
ax.set_xlabel("x Axis label", fontsize=14)
ax.set_ylabel("y Axis label", fontsize=14)
ax.legend(loc="upper right", fontsize=14)


# Annotate the figure

def annotate(x, y, text, code):
    # Circle marker
    c = Circle((x, y), radius=0.15, clip_on=False, zorder=10, linewidth=2.5,
               edgecolor=royal_blue + [0.6], facecolor='none',
               path_effects=[withStroke(linewidth=7, foreground='white')])
    ax.add_artist(c)

    # use path_effects as a background for the texts
    # draw the path_effects and the colored text separately so that the
    # path_effects cannot clip other texts
    for path_effects in [[withStroke(linewidth=7, foreground='white')], []]:
        color = 'white' if path_effects else royal_blue
        ax.text(x, y-0.2, text, zorder=100,
                ha='center', va='top', weight='bold', color=color,
                style='italic', fontfamily='monospace',
                path_effects=path_effects)

        color = 'white' if path_effects else 'black'
        ax.text(x, y-0.33, code, zorder=100,
                ha='center', va='top', weight='normal', color=color,
                fontfamily='monospace', fontsize='medium',
                path_effects=path_effects)


annotate(3.5, -0.13, "Minor tick label", "ax.xaxis.set_minor_formatter")
annotate(-0.03, 1.0, "Major tick", "ax.yaxis.set_major_locator")
annotate(0.00, 3.75, "Minor tick", "ax.yaxis.set_minor_locator")
annotate(-0.15, 3.00, "Major tick label", "ax.yaxis.set_major_formatter")
annotate(1.68, -0.39, "xlabel", "ax.set_xlabel")
annotate(-0.38, 1.67, "ylabel", "ax.set_ylabel")
annotate(1.52, 4.15, "Title", "ax.set_title")
annotate(1.75, 2.80, "Line", "ax.plot")
annotate(2.25, 1.54, "Markers", "ax.scatter")
annotate(3.00, 3.00, "Grid", "ax.grid")
annotate(3.60, 3.58, "Legend", "ax.legend")
annotate(2.5, 0.55, "Axes", "fig.subplots")
annotate(4, 4.5, "Figure", "plt.figure")
annotate(0.65, 0.01, "x Axis", "ax.xaxis")
annotate(0, 0.36, "y Axis", "ax.yaxis")
annotate(4.0, 0.7, "Spine", "ax.spines")

# frame around figure
fig.patch.set(linewidth=4, edgecolor='0.5')
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
- Task: clustering analysis

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

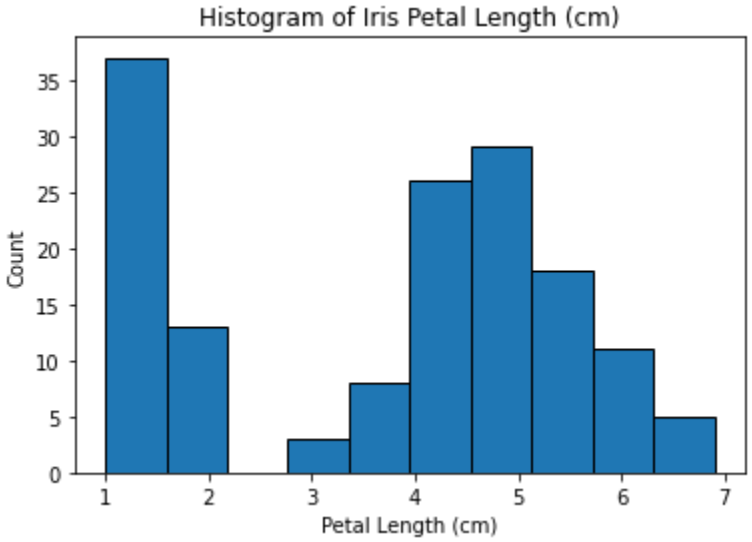
Histogram of one feature

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 OO

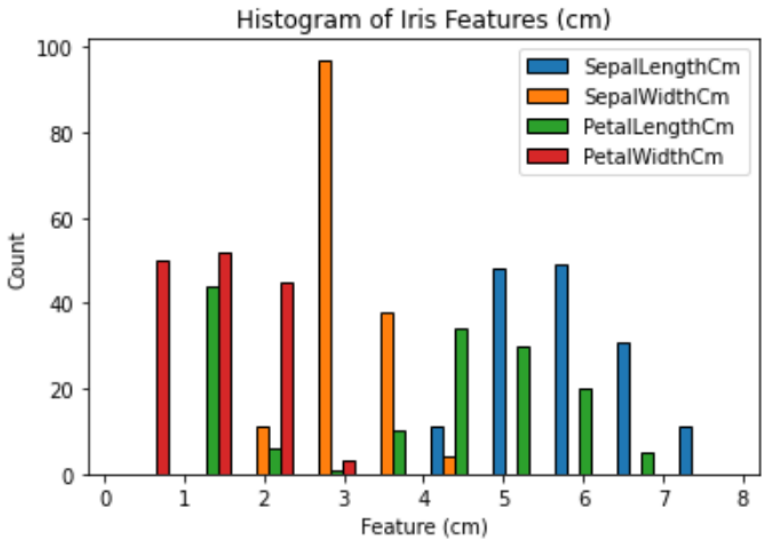
```
fig, axes = plt.subplots()
axes.hist(iris['PetalLengthCm'], bins=10, edgecolor='black')
axes.set_xlabel('Petal Length (cm)')
axes.set_ylabel('Count')
axes.set_title('Histogram of Iris Petal Length (cm)')
plt.show()
```



Note: next examples will show only the pyplot version.

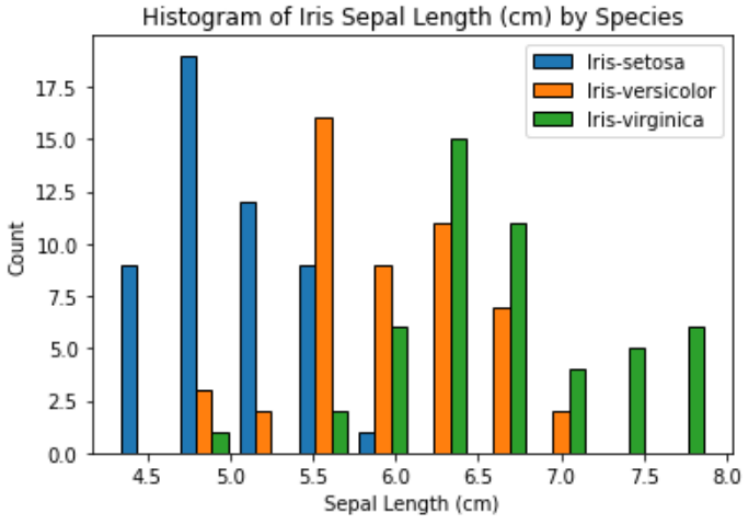
Histogram of all features

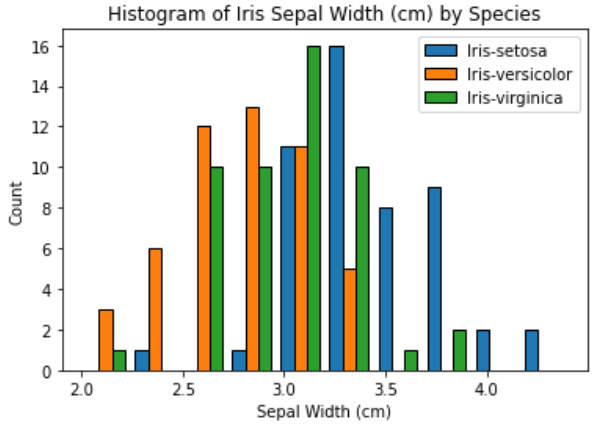
```
plt.hist(iris.loc[:, 'SepalLengthCm':'PetalWidthCm'], bins=10, edgecolor='black', label=iris_features)
plt.legend()
plt.xlabel('Feature (cm)')
plt.ylabel('Count')
plt.title('Histogram of Iris Features (cm)')
plt.show()
```

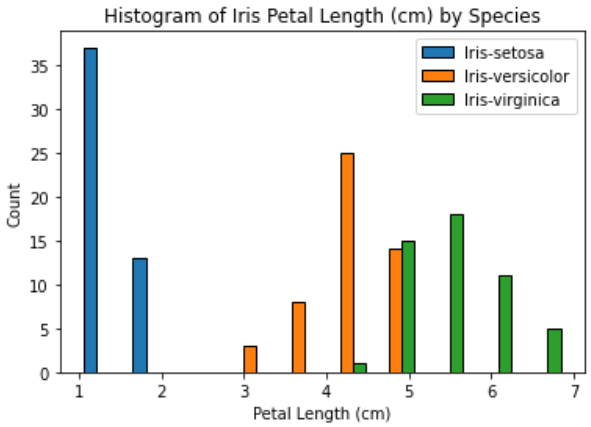


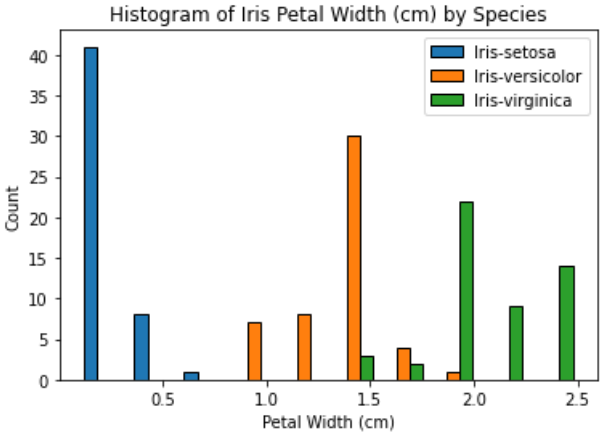
Histogram of each feature by species

```
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()
```







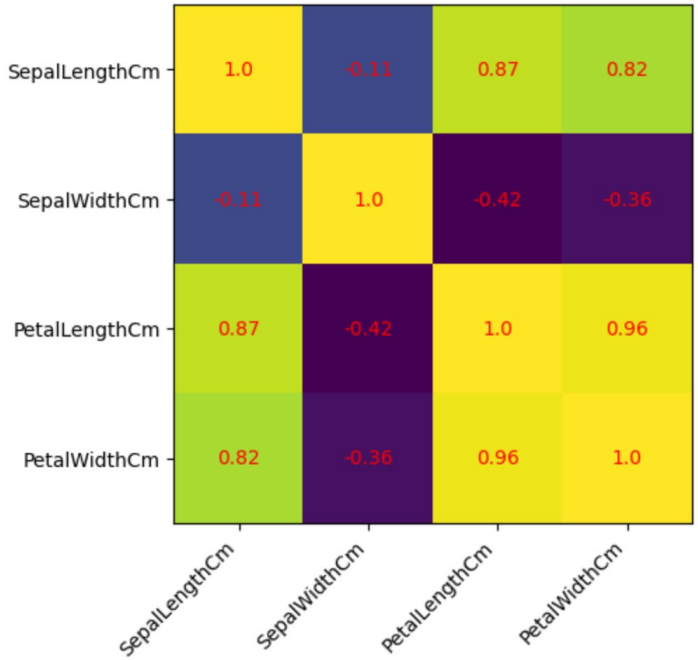


Correlation Heatmap (imshow)

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

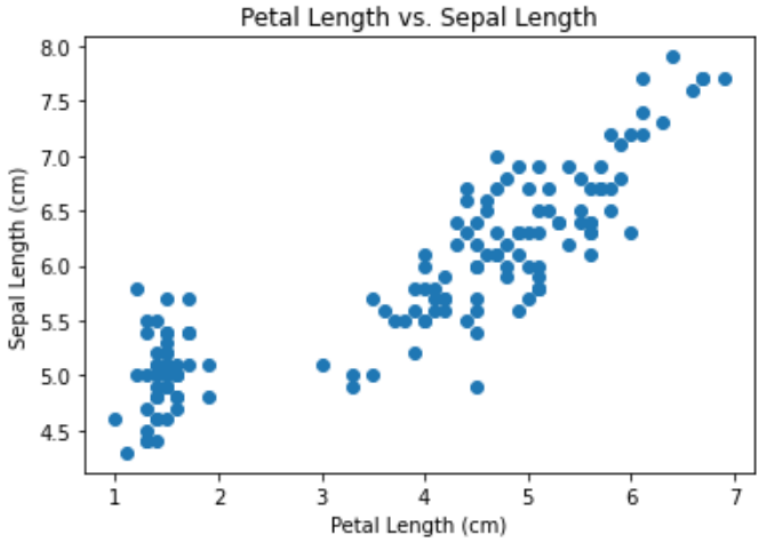
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

```
plt.imshow(correlation)
plt.xticks(np.arange(len(correlation)), correlation.index)
plt.yticks(np.arange(len(correlation)), correlation.index)

# Rotate the tick labels and set their alignment.
plt.xticks(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 = plt.text(j, i, round(correlation.iloc[i, j], 2),
                        ha="center", va="center", color="r")

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()
```

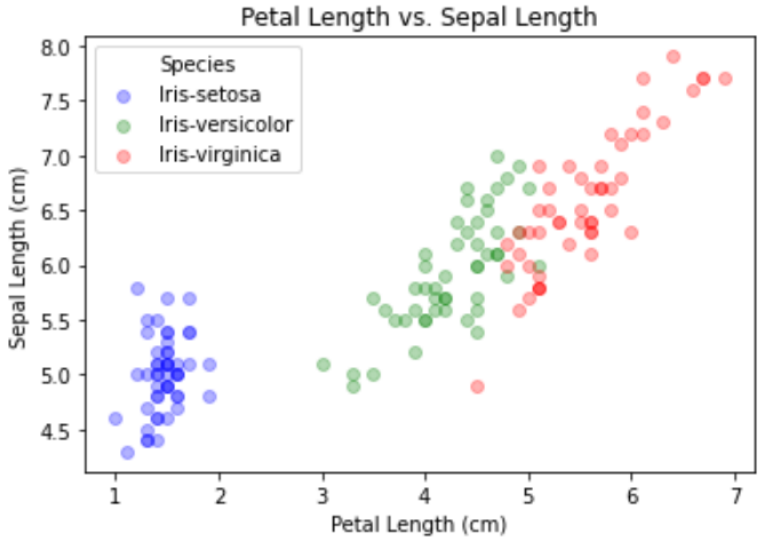


Scatter Plot with colors

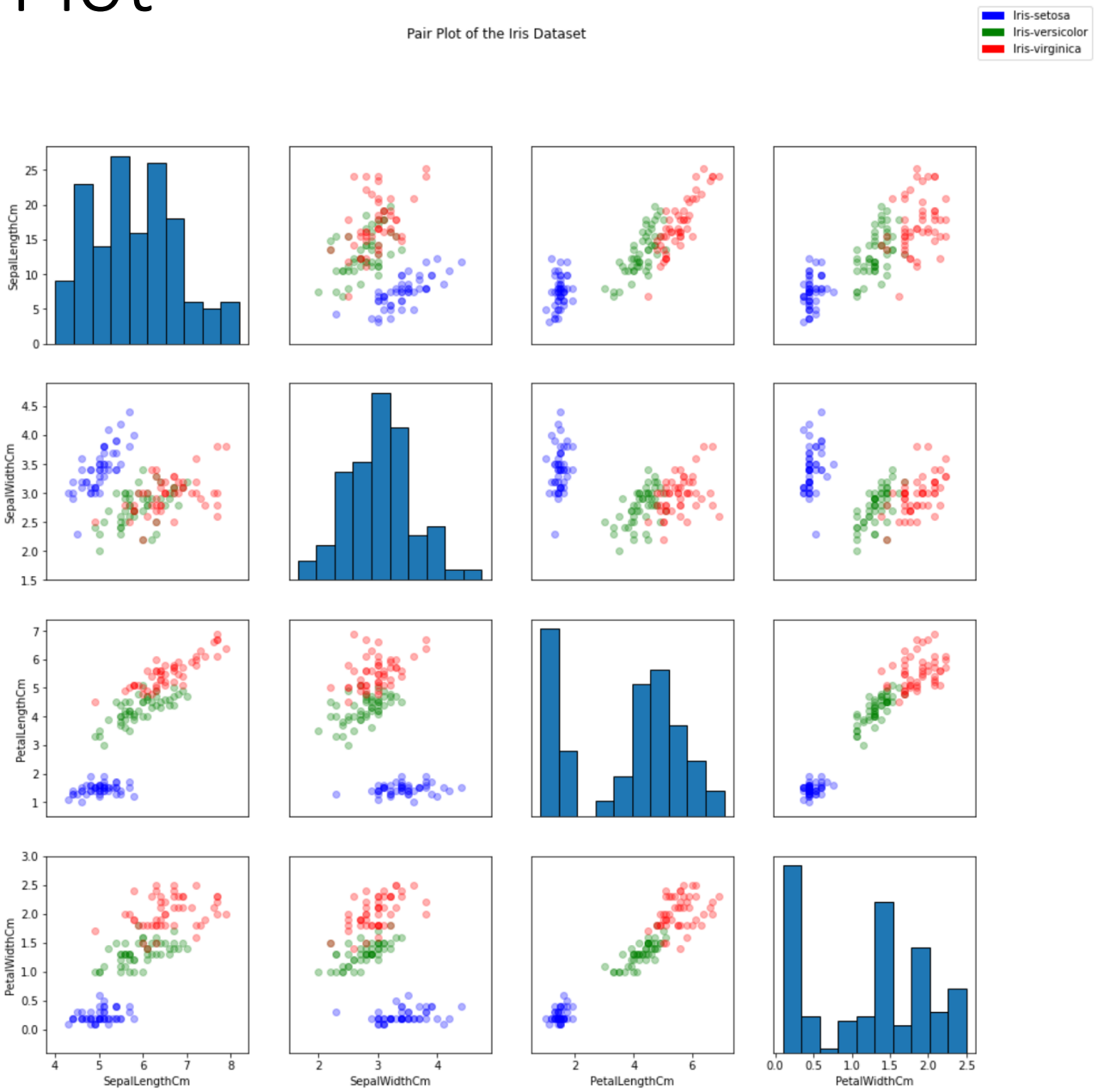
```
color_map = dict(zip(iris_species, ['blue', 'green', 'red']))
color_map
```

```
{'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()
```



Pair Plot


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

Pair Plot

```
import matplotlib.patches as mpatches

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

for i in range(n):
    for j in range(n):
        plt.subplot(n, n, i*n + j + 1)

        if i == j:
            plt.hist(iris[iris_features[i]], bins=10, edgecolor='black')
        else:
            for species, group in iris.groupby('Species'):
                plt.scatter(group[iris_features[j]], group[iris_features[i]],
                            color=color_map[species],
                            alpha=0.3, edgecolor=None,
                            label=species)
            plt.xlim([features_min[iris_features[j]]-0.5, features_max[iris_features[j]]+0.5])
            plt.ylim([features_min[iris_features[i]]-0.5, features_max[iris_features[i]]+0.5])

        if i != n-1:
            plt.xticks([])
        else:
            plt.xlabel(iris_features[j])

        if j != 0:
            plt.yticks([])
        else:
            plt.ylabel(iris_features[i])

# Adding the legend manually
rects = [mpatches.Patch(color=color_map[species], label=species) for species in color_map]
plt.figlegend(rects, color_map.keys(), loc='upper right')

plt.suptitle("Pair Plot of the Iris Dataset")
plt.show()
```

K-Means 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])
```

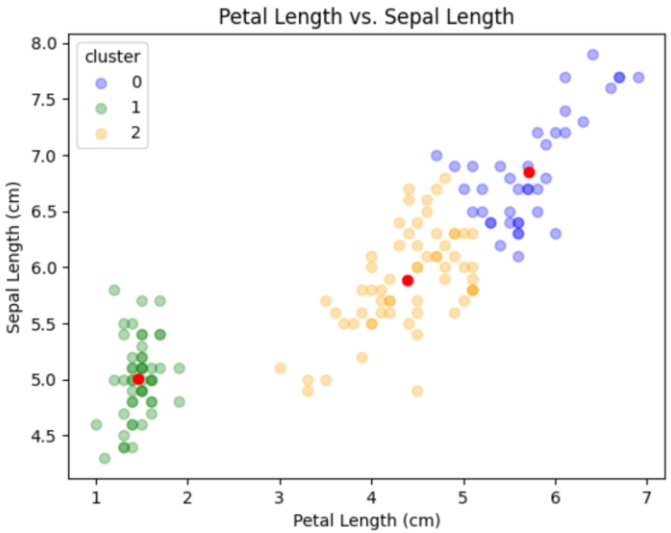
```
iris['cluster'] = cluster
```

```
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]]
```

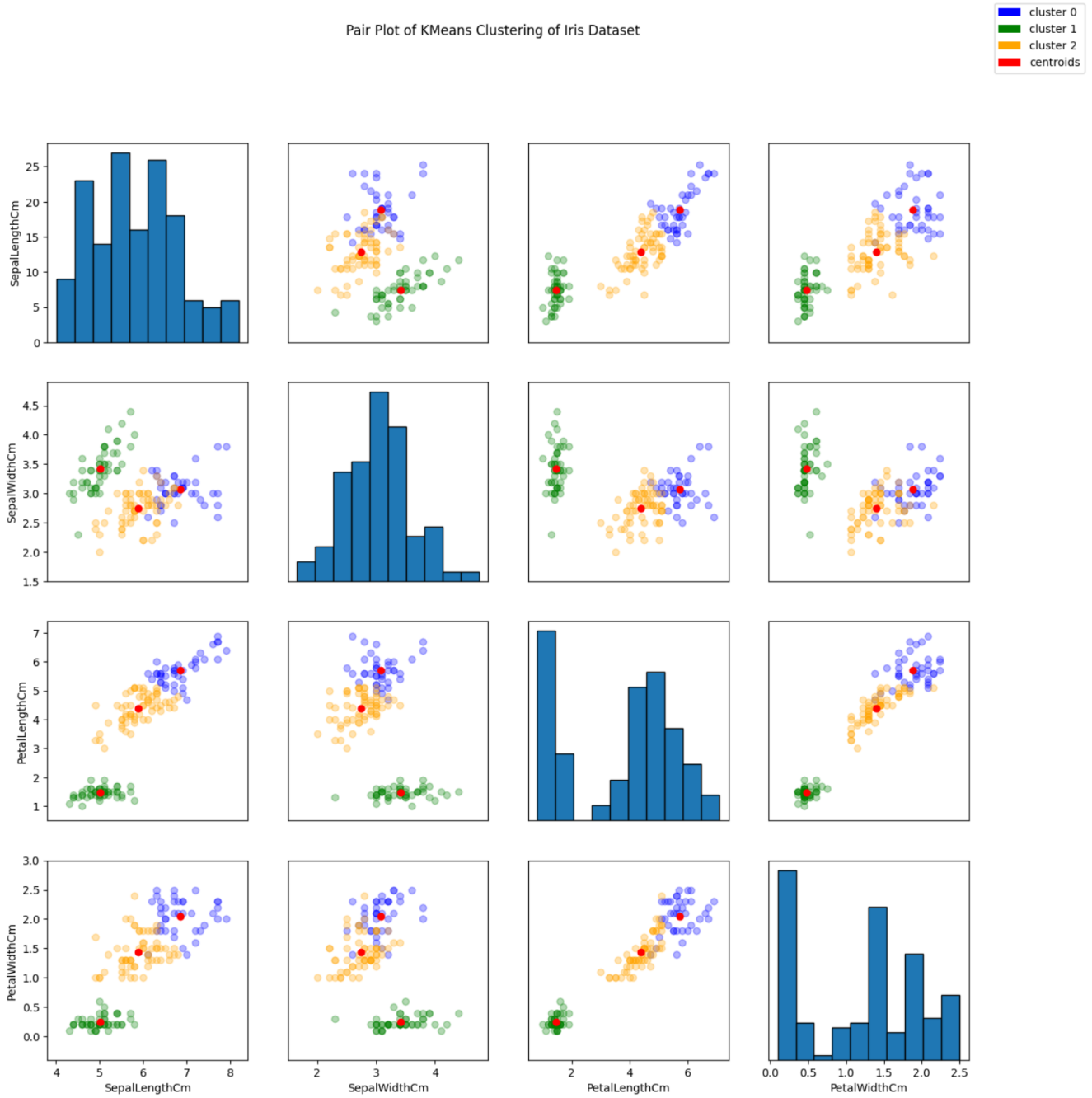
K-Means Clustering

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

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()
```



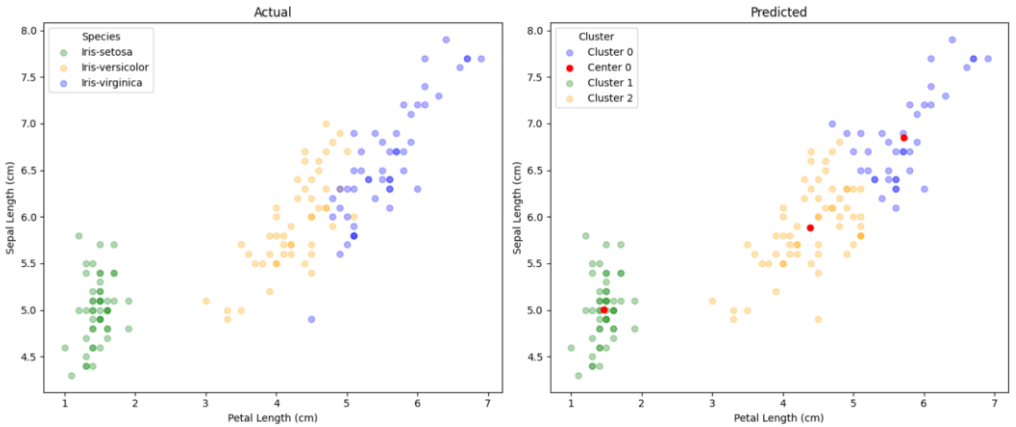
K-Means Clustering: pair plot

```
import matplotlib.patches as mpatches

n = len(iris_features)
plt.figure(figsize=(15, 15))
color_map = ['blue', 'green', 'orange', 'red']

for i in range(n):
    for j in range(n):
        plt.subplot(n, n, i * n + j + 1)

        if i == j:
            plt.hist(iris[iris_features[i]], bins=10, edgecolor='black')
        else:
            for cluster, group in iris.groupby('cluster'):
                plt.scatter(group[iris_features[j]], group[iris_features[i]],
                            color=color_map[cluster],
                            alpha=0.3, edgecolor=None,
                            label=f"Cluster {cluster}" if i == 0 and j == 1 else "")
                # Assuming kmeans.cluster_centers_ is available and correct
                plt.scatter(kmeans.cluster_centers_[cluster, j], kmeans.cluster_centers_[cluster, i],
                            color=color_map[3], label='Centroids' if i == 0 and j == 1 else "")
            plt.xlim([features_min[iris_features[j]]-0.5, features_max[iris_features[j]]+0.5])
            plt.ylim([features_min[iris_features[i]]-0.5, features_max[iris_features[i]]+0.5])

        if i != n-1:
            plt.xticks([])
        if j != 0:
            plt.yticks([])
        if i == n-1:
            plt.xlabel(iris_features[j])
        if j == 0:
            plt.ylabel(iris_features[i])

patches = [mpatches.Patch(color=color_map[i], label=f'Cluster {i}') for i in range(len(color_map) - 1)]
patches.append(mpatches.Patch(color=color_map[-1], label='Centroids'))
plt.figlegend(handles=patches, loc='upper right', title='Legend')

plt.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

Compare result

```python
# Comparison
plt.figure(figsize=(14, 6))

color_map_species = dict(zip(iris_species, ['green', 'orange', 'blue']))

# Plot actual species distribution
plt.subplot(1, 2, 1)
for species, group in iris.groupby('Species'):
    plt.scatter(group['PetalLengthCm'], group['SepalLengthCm'],
                color=color_map_species[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('Actual')

# Move to the next subplot for predicted clusters
plt.subplot(1, 2, 2)
color_map_cluster = ['blue', 'green', 'orange', 'red']
for cluster, group in iris.groupby('cluster'):
    plt.scatter(group['PetalLengthCm'], group['SepalLengthCm'],
                color=color_map_cluster[cluster],
                alpha=0.3, edgecolor=None,
                label=f"Cluster {cluster}")
    # Plot cluster centers
    plt.scatter(kmeans.cluster_centers_[cluster, 2],
                kmeans.cluster_centers_[cluster, 0],
                color=color_map_cluster[3],
                label=f"Center {cluster}" if cluster == 0 else "")
plt.legend(frameon=True, title='Cluster')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Sepal Length (cm)')
plt.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

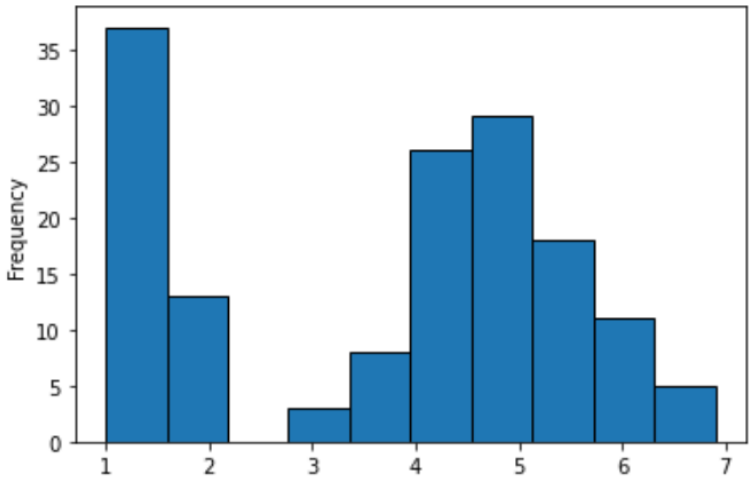
pandas.DataFrame.plot.pie

pandas.DataFrame.plot.scatter

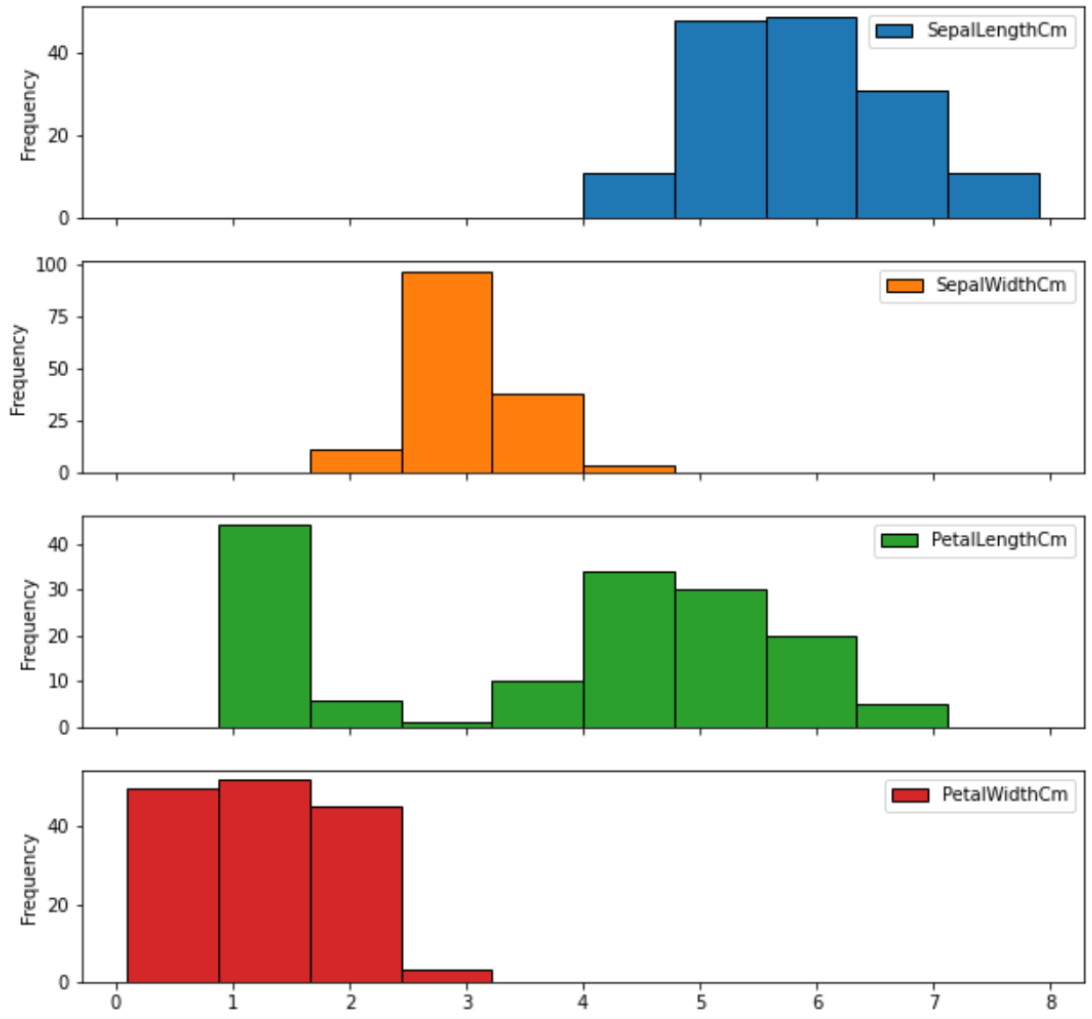
pandas.DataFrame.boxplot

- 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()
```

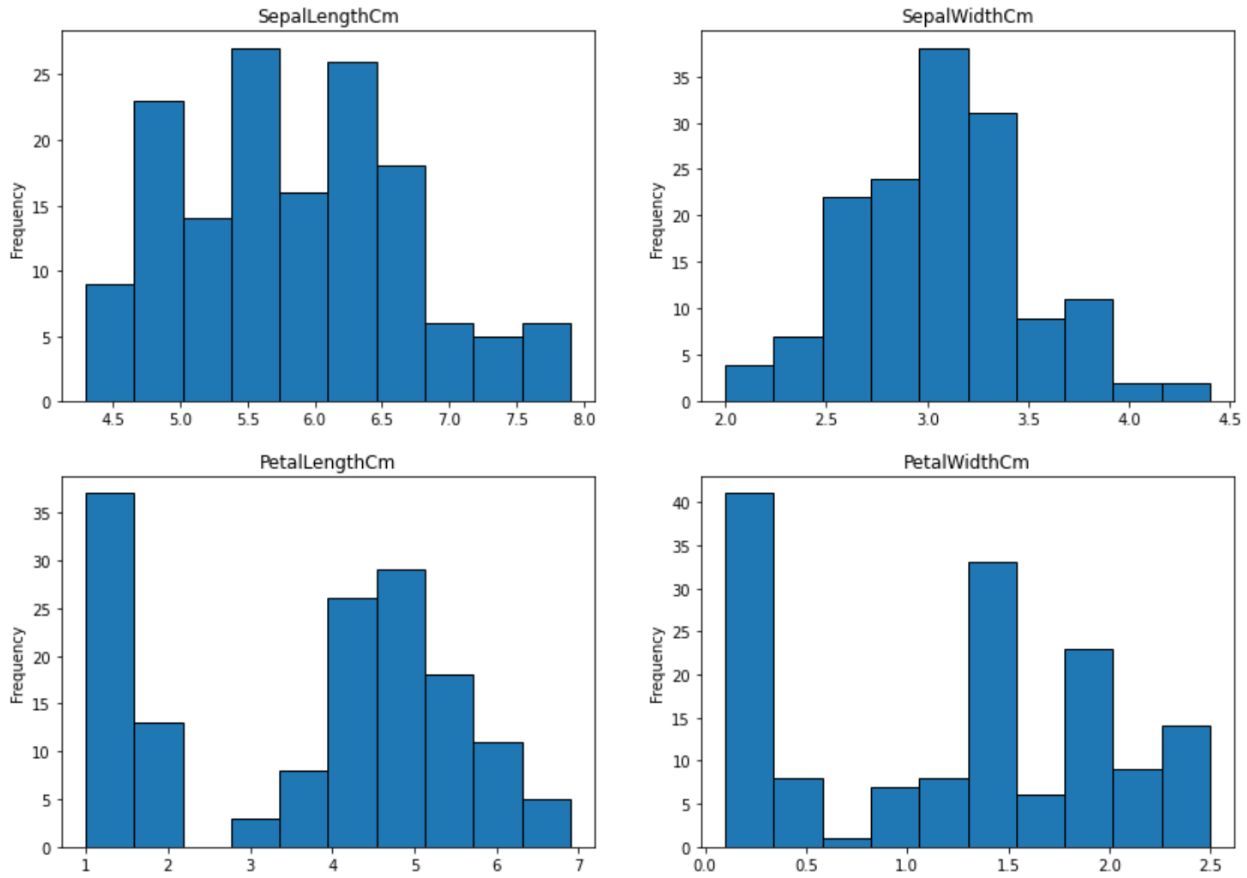


```
iris.loc[:, 'SepalLengthCm':'PetalWidthCm'].plot.hist(bins=10, edgecolor='black', figsize=(10, 10), subplots=True)
plt.show()
```



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()
```



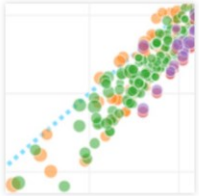
Visualization with Plotly

Plotly

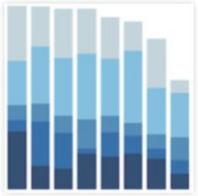
- Based on Javascript
- Interactive viz
- Also support
  - 3D viz
  - Animation
  - Geographical viz
- https://plot.ly/python
- https://chart-studio.plotly.com/

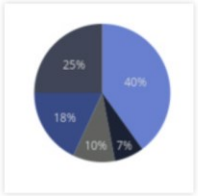
Basic Charts

More Basic Charts »


Scatter Plots


Line Charts


Bar Charts

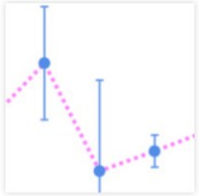


Pie Charts

Bubble Charts

Statistical Charts

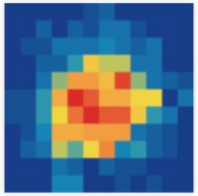
More Statistical Charts »


Error Bars


Box Plots


Histograms


Distplots


2D Histograms

Maps

More Maps »


MapLibre Migration


Tile Choropleth Maps

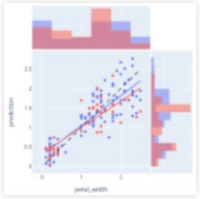
Lines on Tile Maps

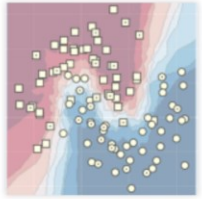
Filled Area on Tile Maps

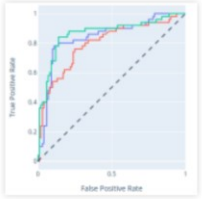

Bubble Maps

Artificial Intelligence and Machine Learning

More AI and ML »


ML Regression


kNN Classification


ROC and PR Curves


PCA Visualization


AI/ML Apps with Dash

https://plot.ly/python

Plotly APIs

Two Styles:

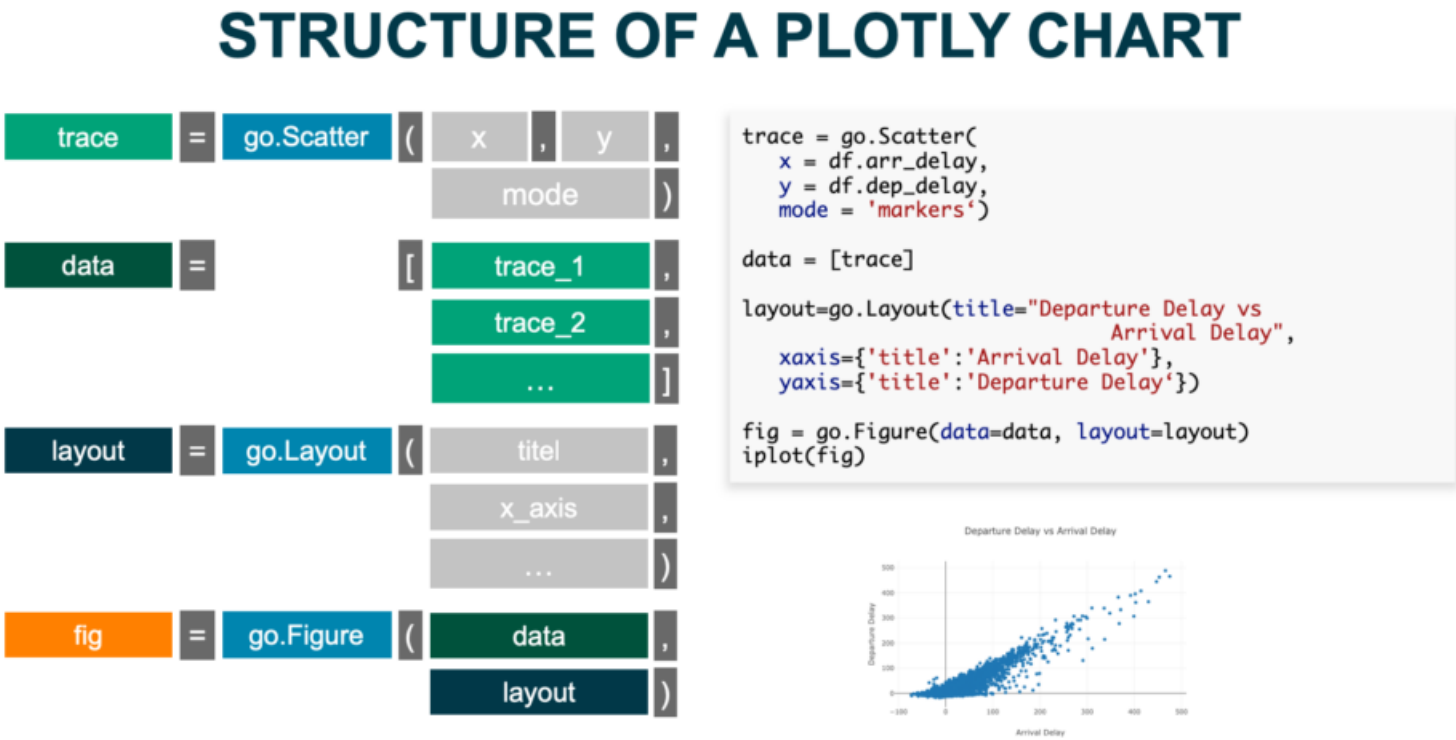
- Graph Objects interface
  - More control and customization
  - Create a plot using multiple functions to add content and modify the appearance
  - Recommended for more complex plots

- Plotly Express
  - High-level wrapper for Plotly
  - Create a plot using one function with a lot of parameters
  - More convenient but less flexible

Stick to one style. Mixing Plotly Express and Graph Objects can make your code harder to read and maintain.

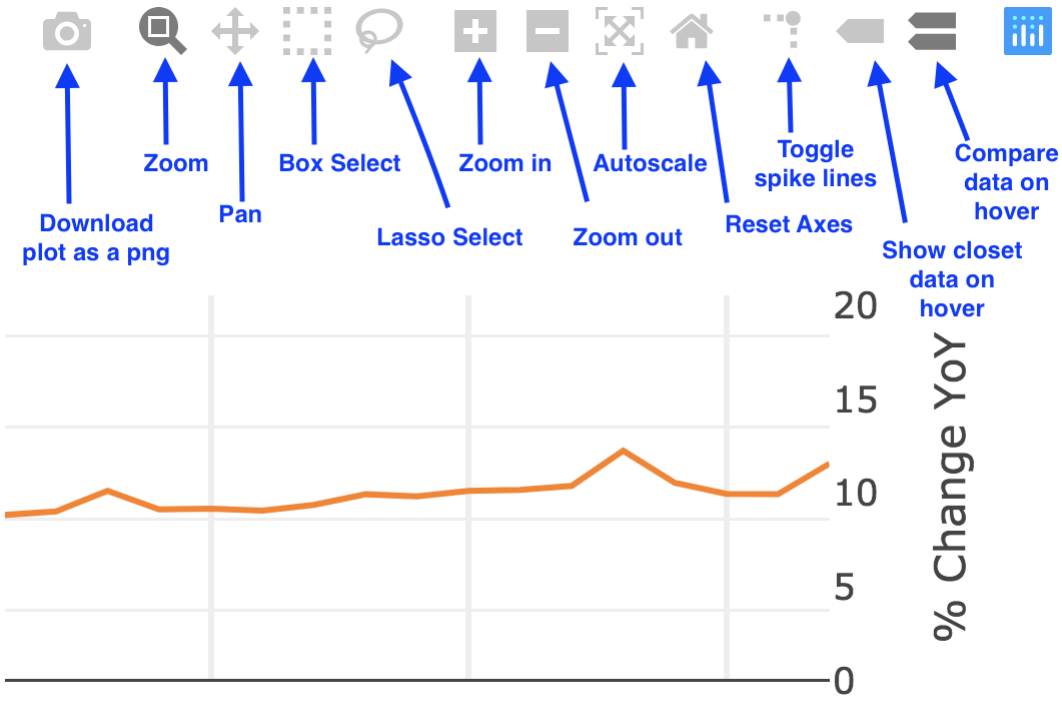
Plotly Graph Objects

- Graph Objects components
  - Figures
  - Traces
  - Layouts



Modebar

- Plotly's modebar is the toolbar that appears when you hover over a Plotly chart.
- It provides interactive features to manipulate the view of the plot.

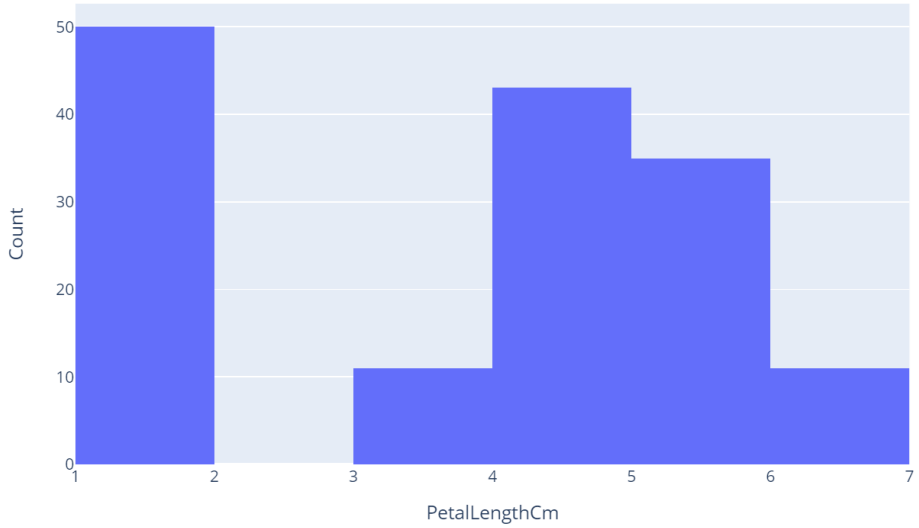


Histogram with Plotly Graph Objects

```python
import plotly.graph_objects as go

trace = go.Histogram(x= iris['PetalLengthCm'], nbinsx=10)
data = [trace]
layout = go.Layout(xaxis_title="PetalLengthCm", yaxis_title="Count")
fig = go.Figure(data=data, layout=layout)
fig.show()
```

```python
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()
```

Two styles
Same result



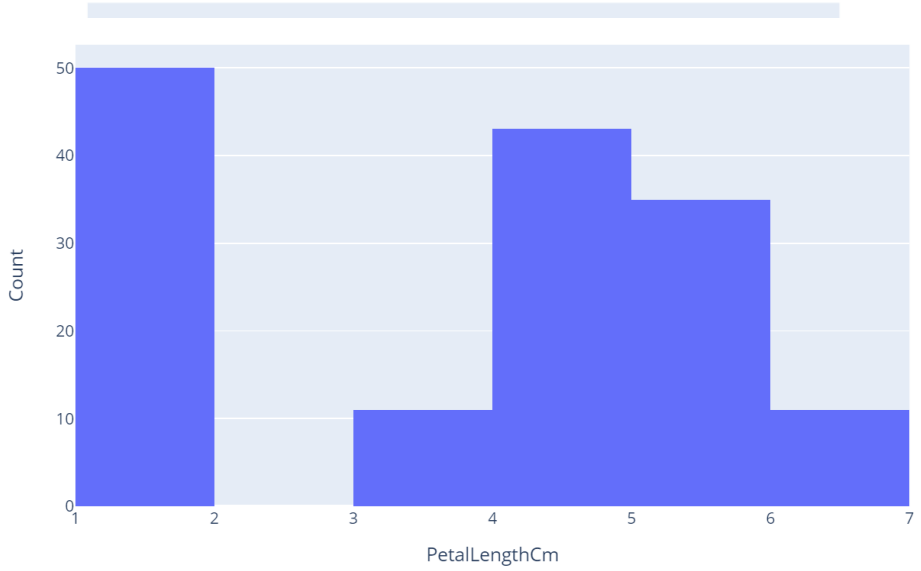
Histogram with Plotly Express

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

plotly.express.histogram

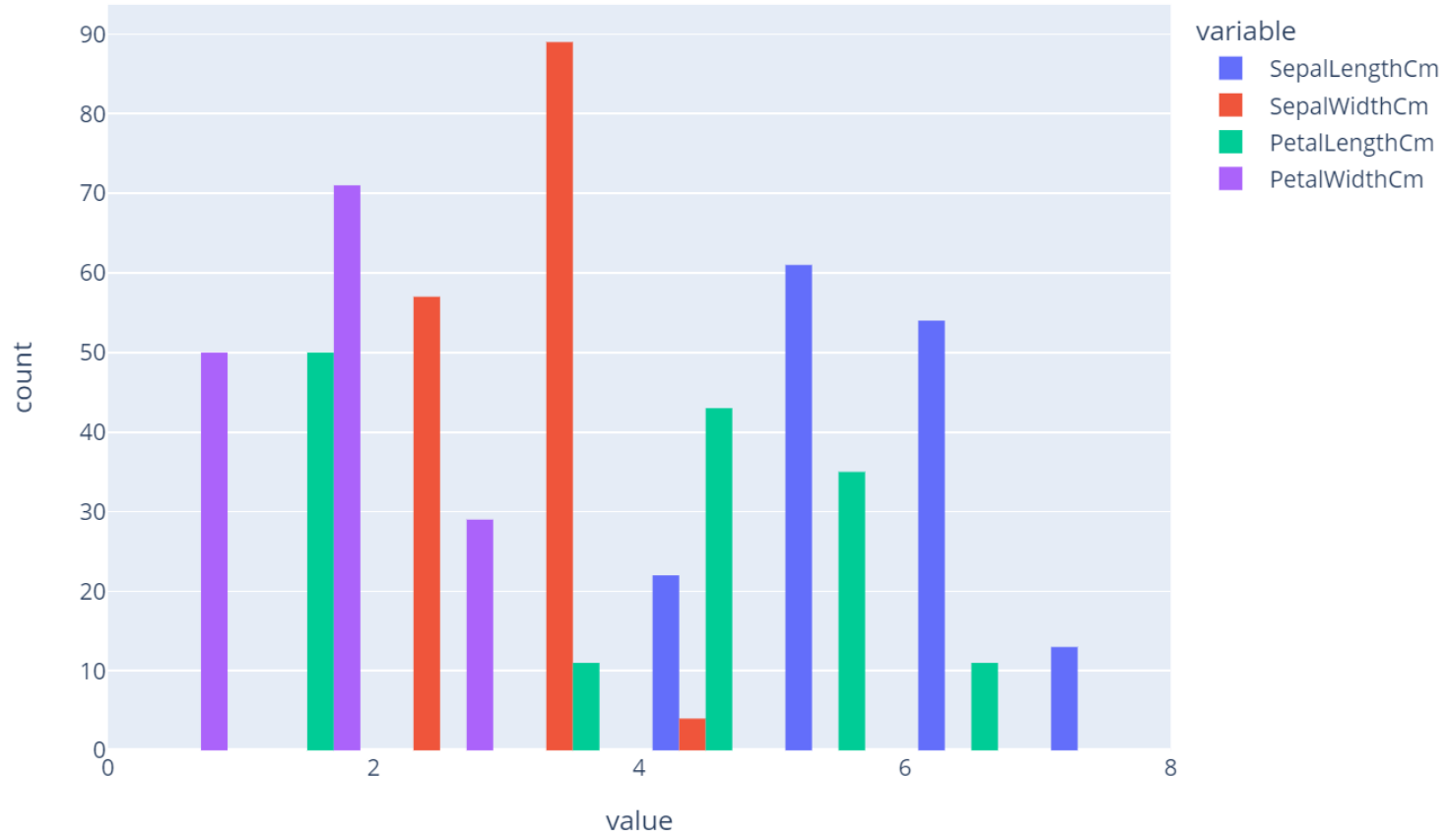
plotly.express. **histogram** (*data_frame=None, x=None, y=None, color=None, pattern_shape=None, facet_row=None, facet_col=None, facet_col_wrap=0, facet_row_spacing=None, facet_col_spacing=None, hover_name=None, hover_data=None, animation_frame=None, animation_group=None, category_orders=None, labels=None, 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)
```



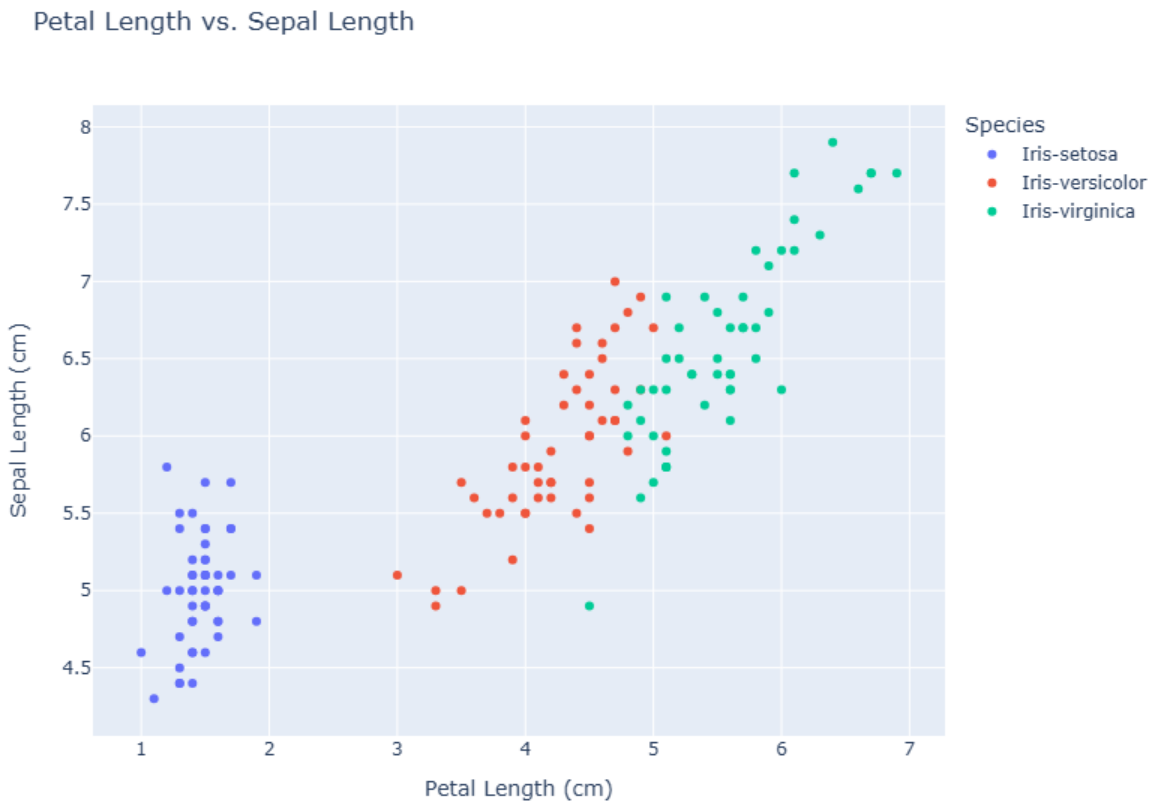
Histogram with Plotly Express

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



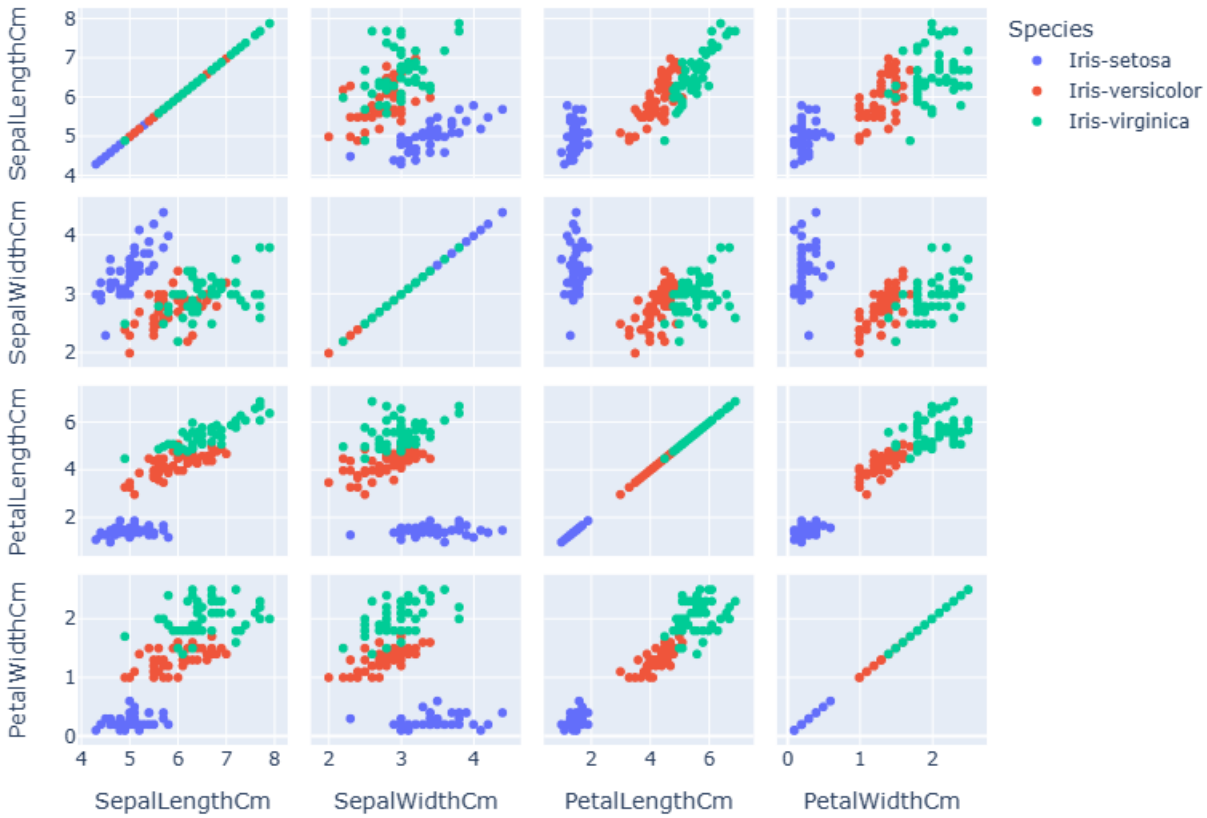
Scatter Plot with Plotly Express

```
fig = px.scatter(iris, x='PetalLengthCm', y='SepalLengthCm',
                 title='Petal Length vs. Sepal Length',
                 color='Species',
                 hover_name='Species',
                 labels=dict(PetalLengthCm='Petal Length (cm)', SepalLengthCm='Sepal Length (cm)'))

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



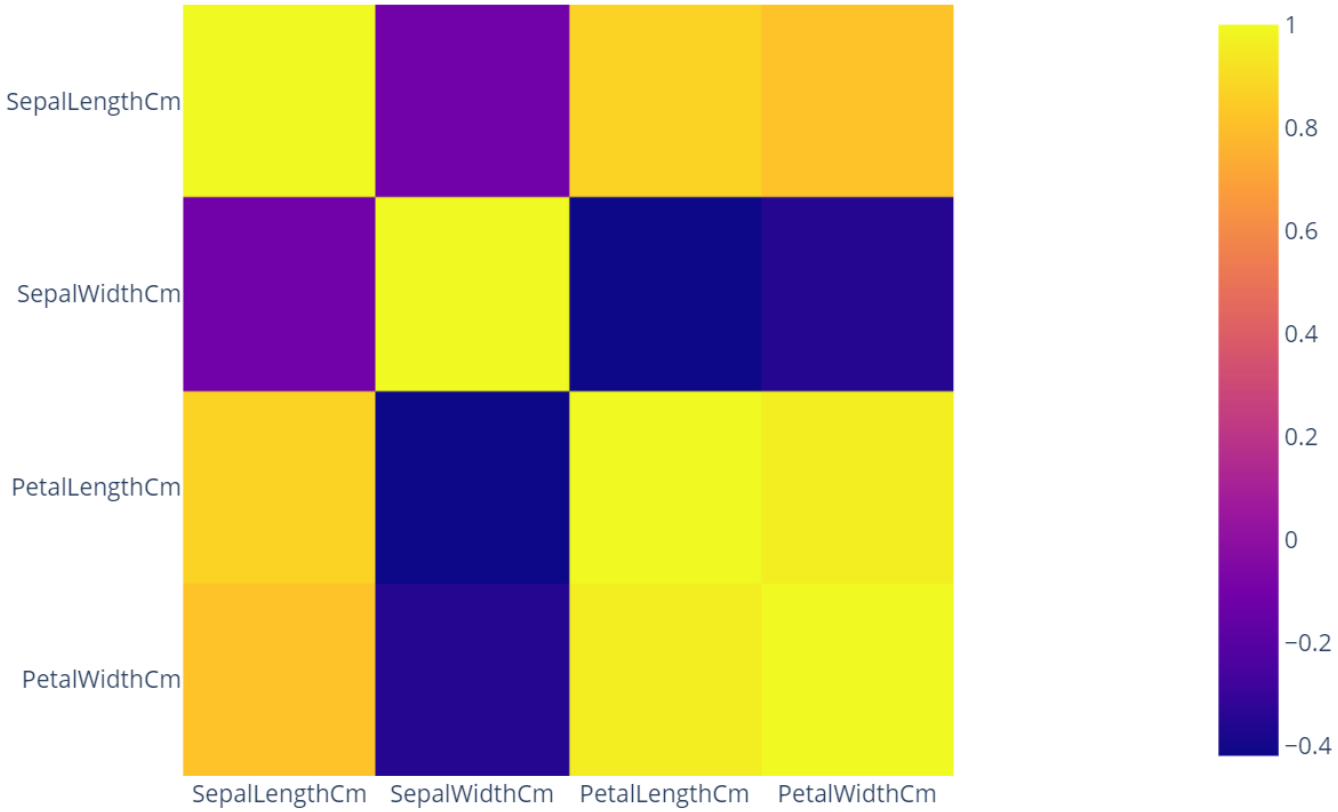
Scatter Matrix (Pair Plot) with Plotly Express

```
fig = px.scatter_matrix(iris,
                        dimensions=['SepalLengthCm','SepalWidthCm','PetalLengthCm', 'PetalWidthCm'],
                        color='Species')

fig.show()
```



Heatmap with Plotly Express (imshow)

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



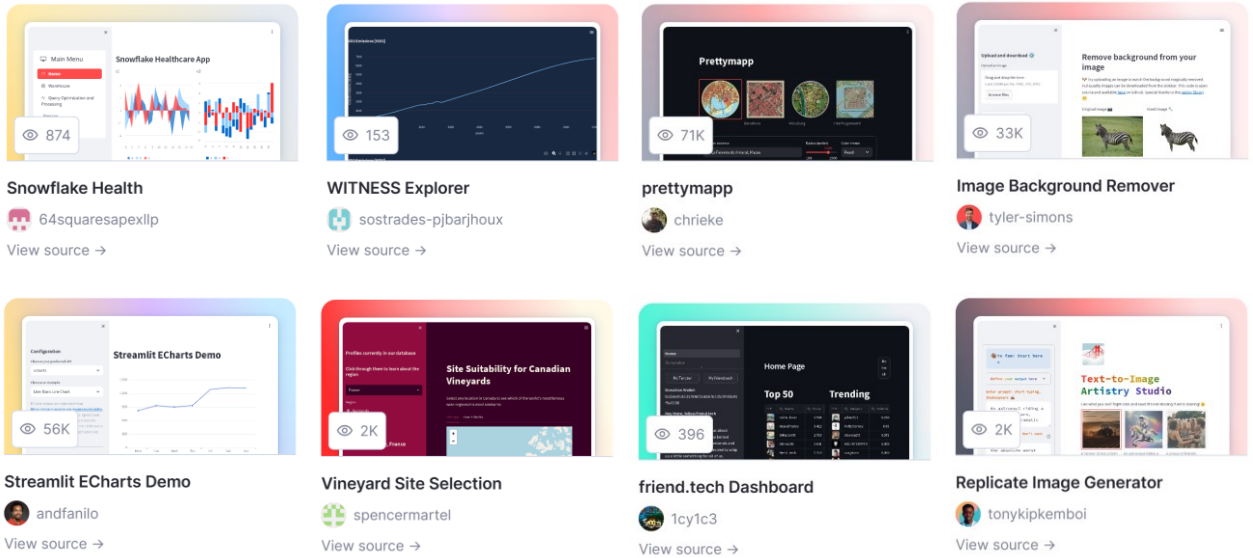
Data analysis and visualization projects with Plotly

- Spotify
  - https://www.kaggle.com/code/varunsaikanuri/spotify-data-visualization
- GapMinder
  - https://www.kaggle.com/code/ssarkar445/starting-with-plotly-express/notebook
- Top 100 Universities
  - https://www.kaggle.com/code/kanncaa1/plotly-tutorial-for-beginners/notebook
- IMDb movie database
  - https://www.kaggle.com/code/andrewedward37/visualization-with-plotly-express-a-starter-guide

Create an interactive app with Streamlit

Streamlit

- Python library for creating web apps for machine learning and data science.
- Built-in support for displaying rich media (charts, maps), data frames, and even machine learning models, making it well-suited for data science and machine learning projects.
- It integrates seamlessly with libraries like Pandas, Matplotlib, Plotly, and others.
- https://streamlit.io
- https://streamlit.io/gallery
- https://docs.streamlit.io/get-started



Streamlit

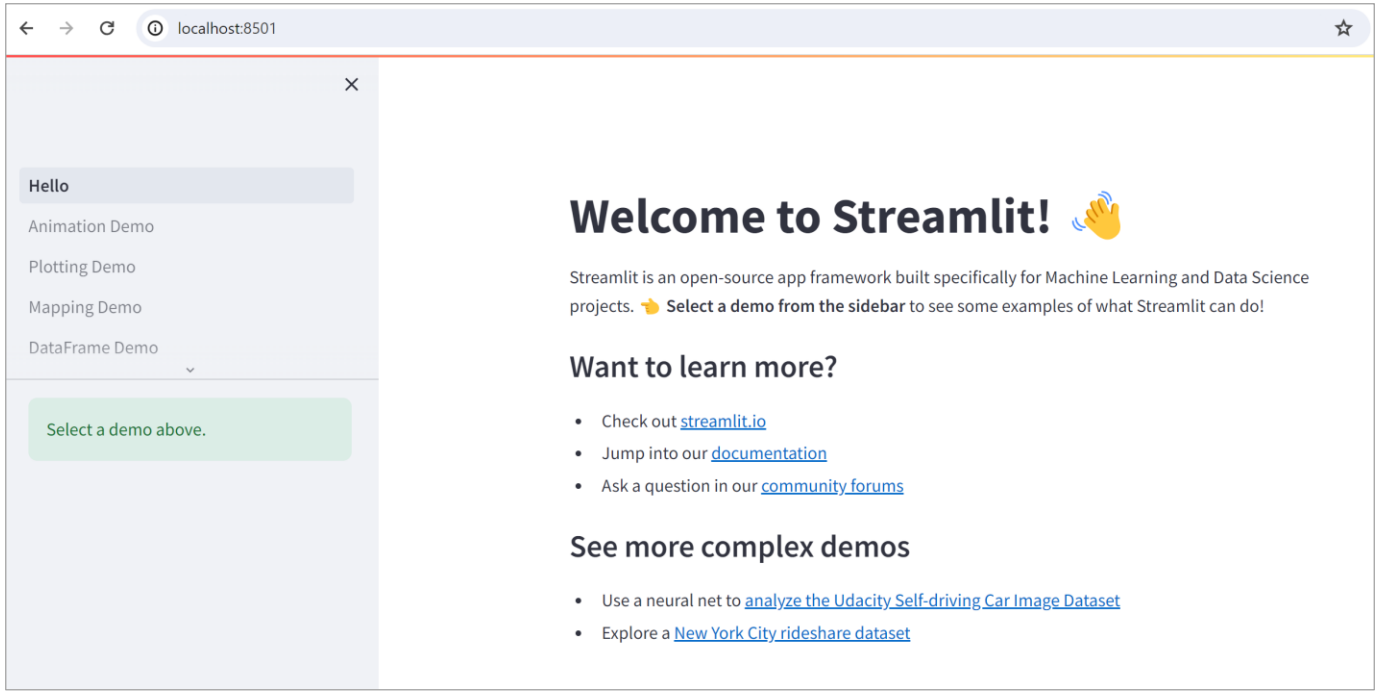
- Install

```
pip install streamlit
```

- Validate the installation by running the built-in ‘hello’ app.

```
streamlit hello
```

The program will run on http://localhost:8501/



Streamlit App: Hello World!

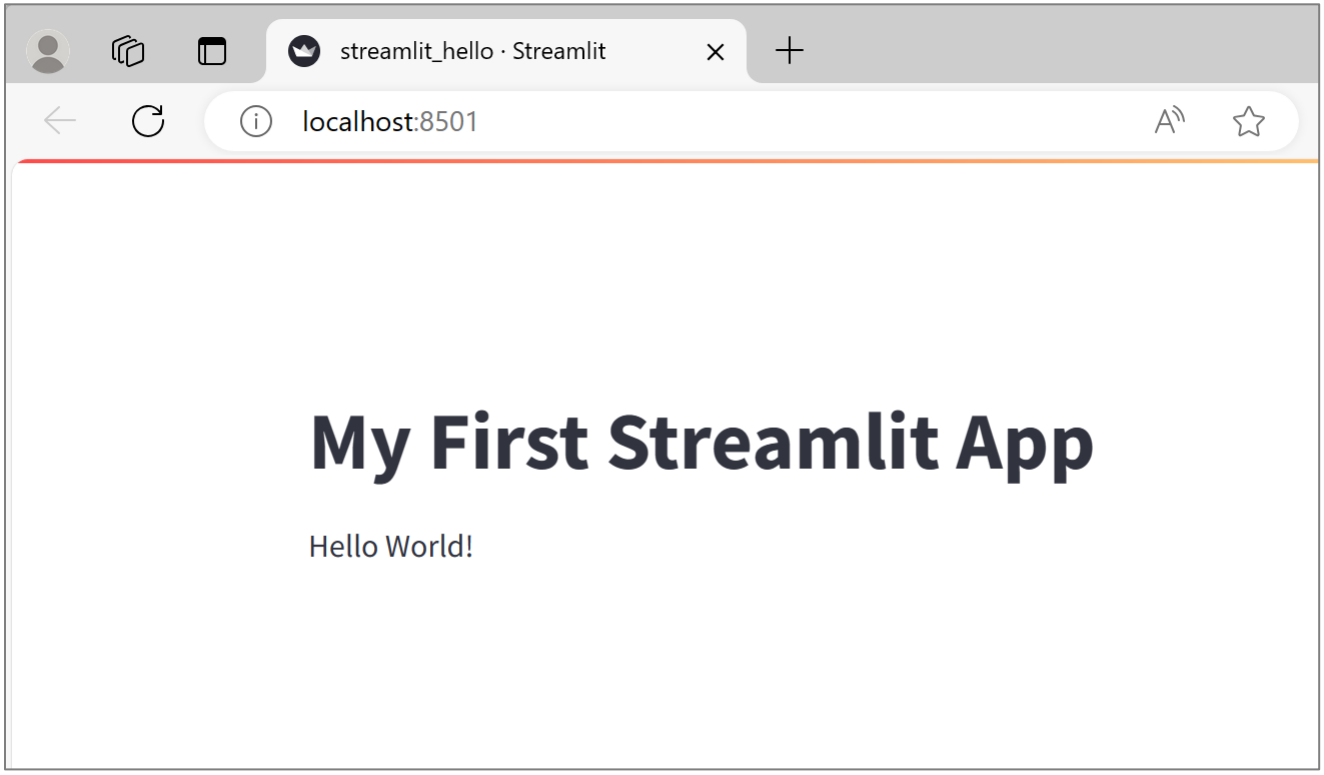
```
import streamlit as st

st.title('My First Streamlit App')
st.write('Hello World!')
```

streamlit_hello.py

How to run:

Run on command line: `streamlit run .\streamlit_hello.py`

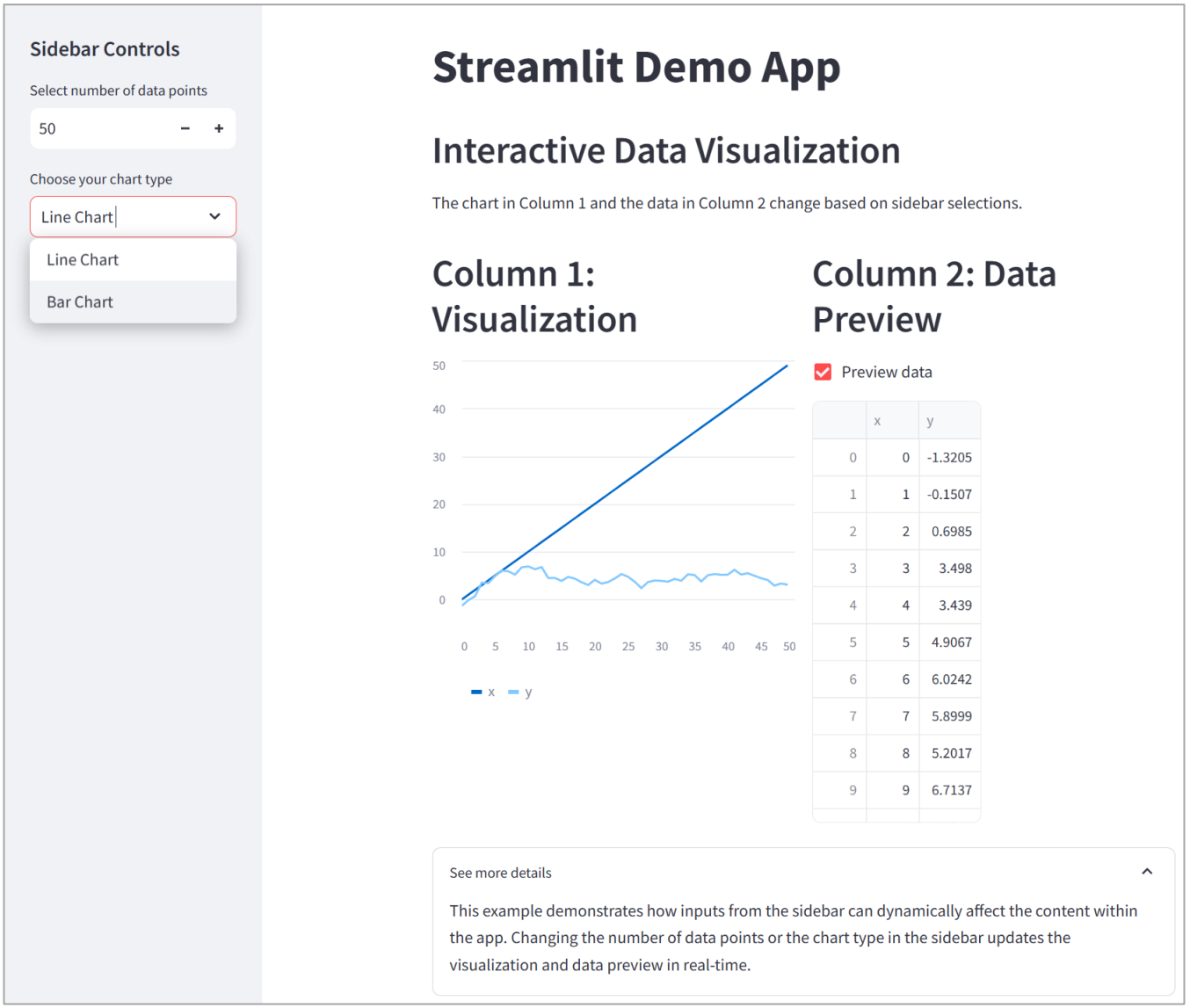
Then open in browser: http://localhost:8501/



Streamlit Structure

- Layout
  - `st.container(), st.columns()`
  - `st.expander()`
- Sidebar
  - `st.sidebar()`
  - `st.sidebar.number_input(), st.sidebar.selectbox()`
- Input
  - `st.number_input(), st.selectbox(), st.slider(), st.button(), st.checkbox()`
- Output
  - `st.write(), st.dataframe()`
  - `st.line_chart(), st.bar_chart(), st.plotly_chart()`
  - `st.map`
- Heading
  - `st.title(), st.header()`

Streamlit Structure Demo



Streamlit Structure Demo (1)

streamlit_demo.py

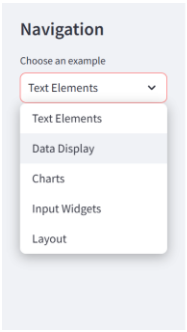
```
import streamlit as st
import numpy as np
import pandas as pd

# Sidebar
st.sidebar.header('Sidebar Controls')
# Sidebar Widgets for controlling what's shown in the columns
number = st.sidebar.number_input('Select number of data points', min_value=10, max_value=100,
value=50)
option = st.sidebar.selectbox(
    'Choose your chart type',
    ('Line Chart', 'Bar Chart')
)
st.sidebar.write('You selected:', option)

# Main content
st.title('Streamlit Demo App')

# Using container
with st.container():
    st.header('Interactive Data Visualization')
    st.write('The chart in Column 1 and the data in Column 2 change based on sidebar
selections.')

# Generating sample data based on the number input
data = pd.DataFrame({
  'x': range(number),
  'y': np.random.randn(number).cumsum()
})
```

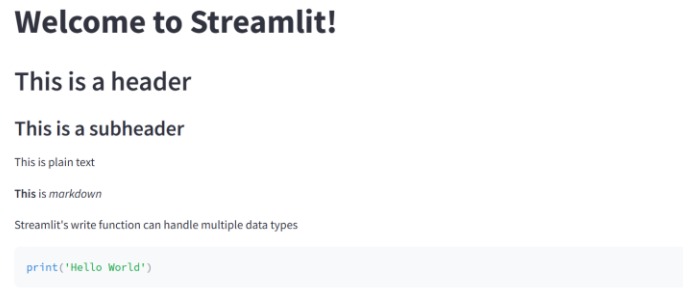
Streamlit Structure Demo (2)

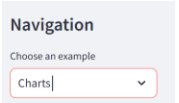
```
# Columns for displaying charts and data based on sidebar inputs
col1, col2 = st.columns(2)

with col1:
    st.header('Column 1: Visualization')
    if option == 'Line Chart':
        st.line_chart(data)
    elif option == 'Bar Chart':
        st.bar_chart(data['y'])

with col2:
    st.header('Column 2: Data Preview')
    # Checkbox to select whether to preview the data
    if st.checkbox('Preview data', key='preview_data'):
        # Display the DataFrame without the index
        st.dataframe(data.reset_index(drop=True))

# Expander for additional details
with st.expander("See more details"):
    st.write("""
        This example demonstrates how inputs from the sidebar can dynamically affect the content
        within the app. Changing the number of data points or the chart type in the sidebar
        updates the visualization and data preview in real-time.
    """)
```

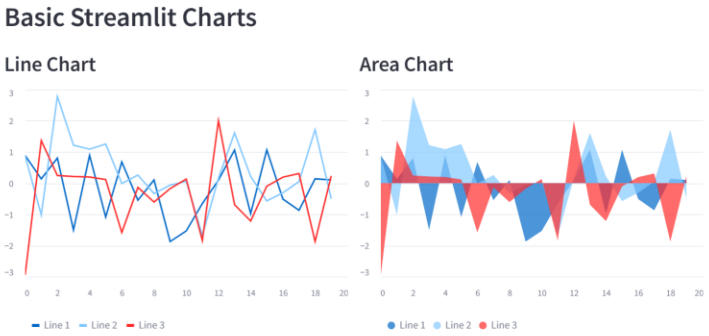
More Examples

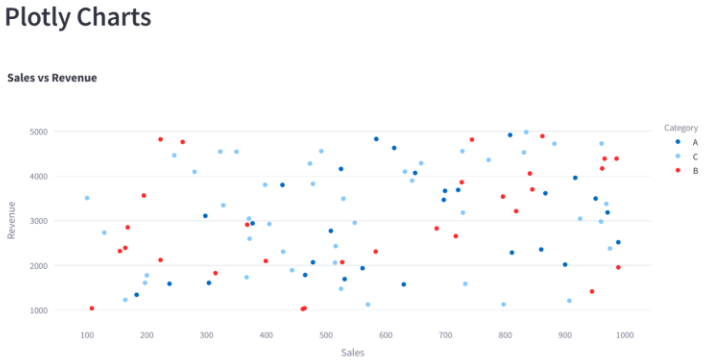
`streamlit_examples.py`

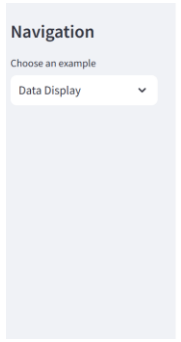




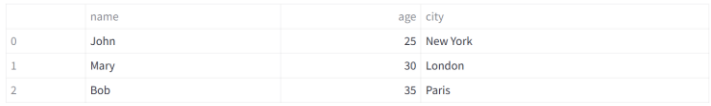




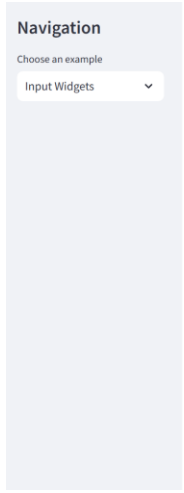


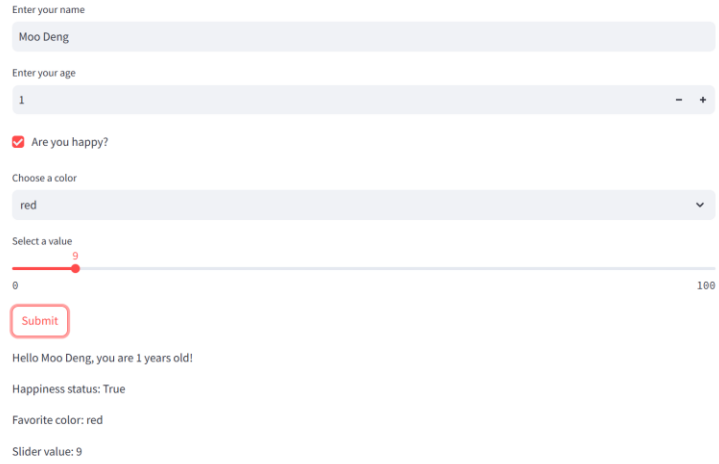


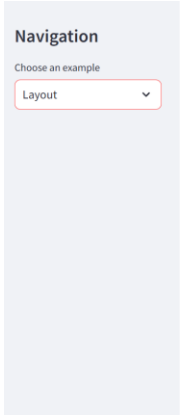
	name	age	city
0	John	25	New York
1	Mary	30	London
2	Bob	35	Paris

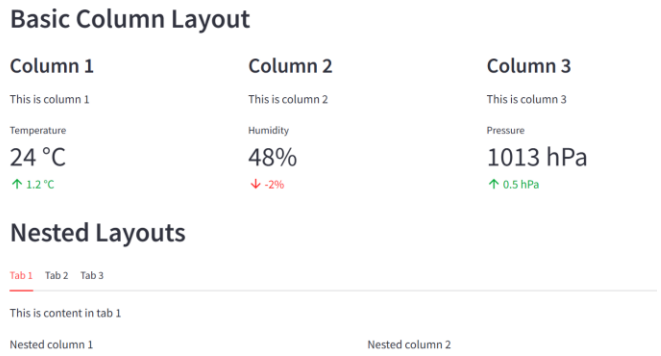


	name	age	city
0	John	25	New York
1	Mary	30	London
2	Bob	35	Paris









Cache

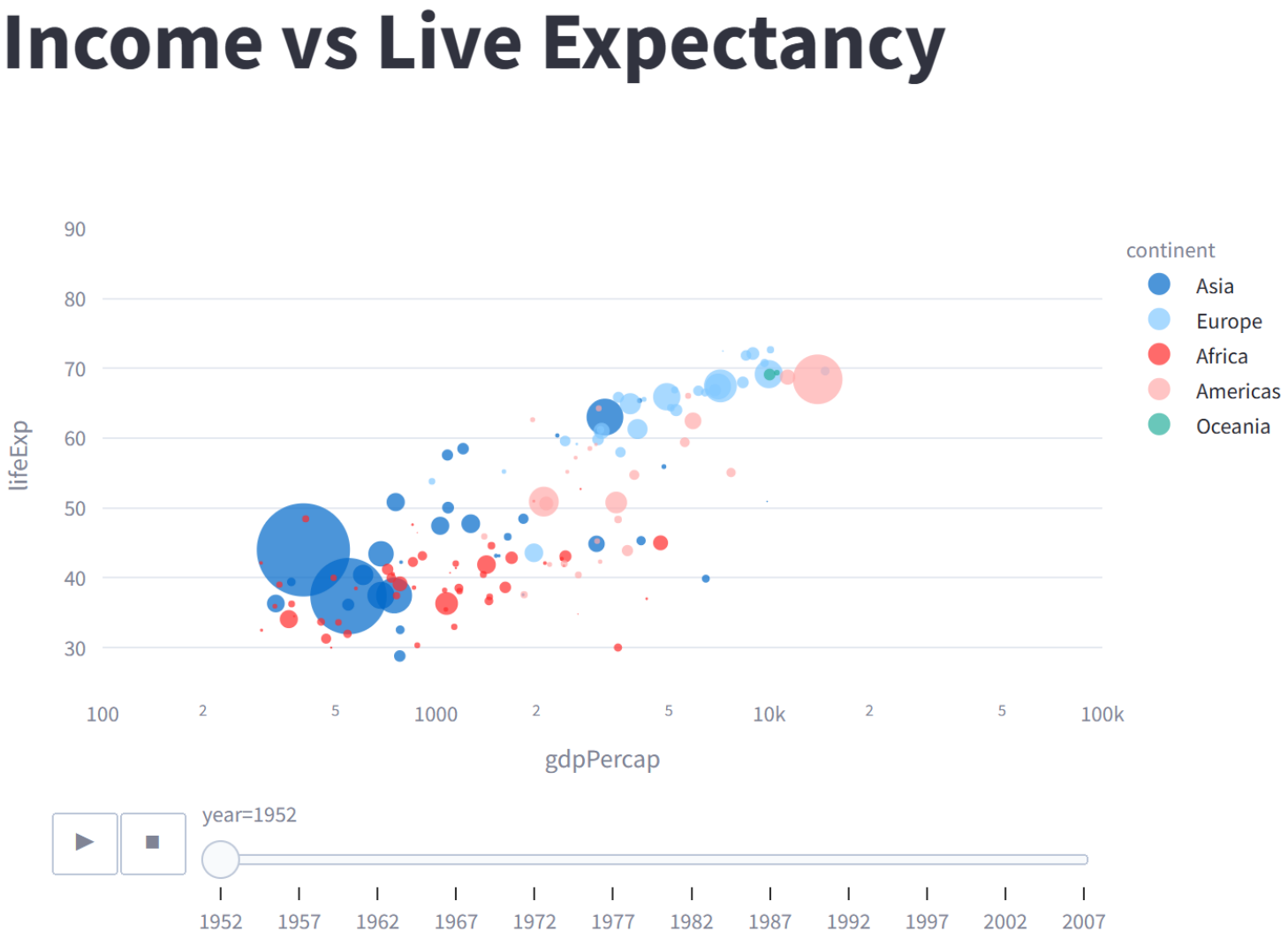
- Each time a user interaction occurs, the entire script is rerun.
- This design choice simplifies the development process but can lead to performance degradation for complex apps.
- To improve performance, Streamlit introduces caching.
- Returned value of a function with the same parameters and the same function code will be cached and reused.
- Use `@st.cache_data` and `@st.cache_resource` decorators.
  - Use @st.cache_data for most data objects
  - Use @st.cache_resource for database connections, ML models, file handles, threads, etc.

```
@st.cache_data
def load_data(url):
    df = pd.read_csv(url)
    return df

df = load_data("https://github.com/plotly/datasets/raw/master/uber-rides-data1.csv")
```

```
@st.cache_resource
def init_connection():
    host = "hh-pgsql-public.ebi.ac.uk"
    database = "pfmegrnargs"
    user = "reader"
    password = "NWDMCE5xdipIjRrp"
    return psycopg2.connect(host=host, database=database, user=user, password=password)

conn = init_connection()
```

https://docs.streamlit.io/develop/concepts/architecture/caching

Streamlit + Plotly Animation: Gapminder



Streamlit + Plotly Animation: Gapminder

streamlit_gapminder.py

```
import pandas as pd
import plotly.express as px
import streamlit as st

# Load the dataset
#df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/gapminderDataFiveYear.csv')
df = pd.read_csv('gapminderData.csv')

def create_animated_figure():
    fig = px.scatter(df,
                     x="gdpPercap",
                     y="lifeExp",
                     animation_frame="year",
                     animation_group="country",
                     size="pop",
                     color="continent",
                     hover_name="country",
                     log_x=True,
                     size_max=55,
                     range_x=[100,100000],
                     range_y=[25,90])

    fig.update_layout(transition={'duration': 100})
    return fig

st.title('Income vs Live Expectancy')

# Display the animated plot
fig = create_animated_figure()
st.plotly_chart(fig)
```

```
country,year,pop,continent,lifeExp,gdpPercap
Afghanistan,1952,8425333,Asia,28.801,779.4453145
Afghanistan,1957,9240934,Asia,30.332,820.8530296
Afghanistan,1962,10267083,Asia,31.997,853.10071
Afghanistan,1967,11537966,Asia,34.02,836.1971382
Afghanistan,1972,13079460,Asia,36.088,739.9811058
Afghanistan,1977,14880372,Asia,38.438,786.11336
Afghanistan,1982,12881816,Asia,39.854,978.0114388
Afghanistan,1987,13867957,Asia,40.822,852.3959448
Afghanistan,1992,16317921,Asia,41.674,649.3413952
Afghanistan,1997,22227415,Asia,41.763,635.341351
Afghanistan,2002,25268405,Asia,42.129,726.7340548
Afghanistan,2007,31889923,Asia,43.828,974.5803384
Albania,1952,1282697,Europe,55.23,1601.056136
Albania,1957,1476505,Europe,59.28,1942.284244
Albania,1962,1728137,Europe,64.82,2312.888958
Albania,1967,1984060,Europe,66.22,2760.196931
Albania,1972,2263554,Europe,67.69,3313.422188
Albania,1977,2509048,Europe,68.93,3533.00391
Albania,1982,2780097,Europe,70.42,3630.880722
Albania,1987,3075321,Europe,72,3738.932735
Albania,1992,3326498,Europe,71.581,2497.437901
Albania,1997,3428038,Europe,72.95,3193.054604
```

streamlit_gapminder_cache.py

```
import pandas as pd
import plotly.express as px
import streamlit as st

# Load the dataset
@st.cache_data
def load_data():
    return pd.read_csv('gapminderData.csv')

# Cache the figure creation, ttl=3600 means the cache will expire after 1 hour
@st.cache_data(ttl=3600)
def create_animated_figure(data):
    fig = px.scatter(data,
                     x="gdpPercap",
                     y="lifeExp",
                     animation_frame="year",
                     animation_group="country",
                     size="pop",
                     color="continent",
                     hover_name="country",
                     log_x=True,
                     size_max=55,
                     range_x=[100,100000],
                     range_y=[25,90])

    fig.update_layout(transition={'duration':100})
    return fig

st.title('Income vs Live Expectancy')

# Load the dataset
df = load_data()

# Display the animated plot
fig = create_animated_figure(df)
st.plotly_chart(fig)
```

Streamlit: Grading App

score
97
95
94
93
83
92
92
91
90
90
89
89
81
88
88
87
87
85
83
82
82
81
81
81
80
80
80
80
80
80
80
81
82
79
79
79
79
79
79
79
79
77
78
78
78

Simple Grading App

Enter Scores

Choose input method:

- ○ Manual Entry
- ◉ File Upload

Upload CSV file with scores:

Drag and drop file here
Limit 200MB per file • CSV
Browse files

scores.csv 0.5KB

Class Statistics

Average	Median	Count
74.9	76.0	139

Detailed Results

	Score	Grade
0	97	A
1	95	A
2	94	A
3	93	A
4	83	B
5	92	A
6	92	A
7	91	A
8	90	A
9	90	A

Download Results

Grade Criteria

Minimum score for A: 90

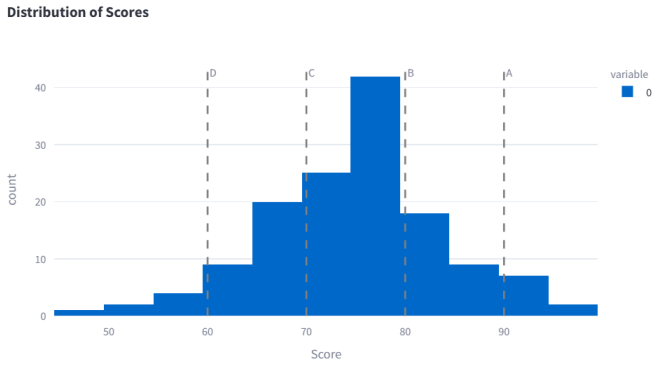
Minimum score for B: 80

Minimum score for C: 70

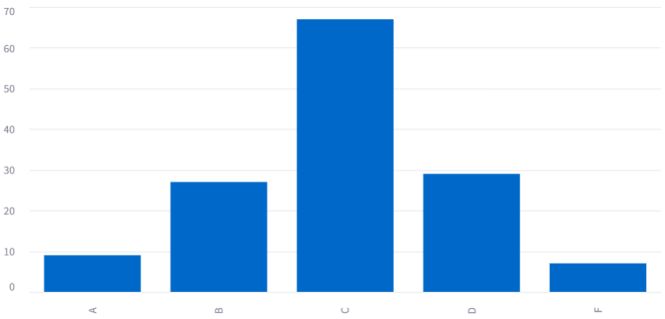
Minimum score for D: 60

Score Distribution

Adjust histogram bin size: 5 (1 – 10)



Grade Distribution



Streamlit: Grading App (1)

streamlit_grading.py

```
import streamlit as st
import pandas as pd
import numpy as np
import plotly.express as px

def get_grade(score, grade_min):
    """Convert score to letter grade based on minimum score criteria"""
    if score >= grade_min['A']: return 'A'
    elif score >= grade_min['B']: return 'B'
    elif score >= grade_min['C']: return 'C'
    elif score >= grade_min['D']: return 'D'
    else: return 'F'

def main():
    # Main page title
    st.title('Simple Grading App')

    # Sidebar with grade criteria
    st.sidebar.header('Grade Criteria')
    grade_min = {
        'A': st.sidebar.number_input('Minimum score for A:', value=90),
        'B': st.sidebar.number_input('Minimum score for B:', value=80),
        'C': st.sidebar.number_input('Minimum score for C:', value=70),
        'D': st.sidebar.number_input('Minimum score for D:', value=60)
    }

    # Input section
    st.header('Enter Scores')
    input_method = st.radio('Choose input method:', ['Manual Entry', 'File Upload'])
```

Streamlit: Grading App (2)

```
    scores = None

    if input_method == 'Manual Entry':
        scores_text = st.text_area(
            "Enter scores (one per line or comma-separated):",
            height=100
        )
        if scores_text:
            if ',' in scores_text:
                scores = [float(x) for x in scores_text.split(',')]
            else:
                scores = [float(x) for x in scores_text.split('\n') if x.strip()]
    else:
        uploaded_file = st.file_uploader("Upload CSV file with scores:", type='csv')
        if uploaded_file:
            df = pd.read_csv(uploaded_file)
            scores = df.iloc[:, 0].tolist()

    if scores:
        # Convert to numpy array for calculations
        scores = np.array(scores)

        # Display basic statistics
        st.header('Class Statistics')
        col1, col2, col3 = st.columns(3)
        with col1:
            st.metric("Average", f"{np.mean(scores):.1f}")
        with col2:
            st.metric("Median", f"{np.median(scores):.1f}")
        with col3:
            st.metric("Count", len(scores))
```

Streamlit: Grading App (3)

```
        # Create histogram with adjustable bins
        st.header('Score Distribution')

        # Add bin size slider
        bin_size = st.slider(
            "Adjust histogram bin size:",
            min_value=1,
            max_value=10,
            value=5,
            help="Smaller values show more detail, larger values show general trends"
        )

        # Calculate number of bins based on bin size
        num_bins = int((np.max(scores) - np.min(scores)) / bin_size) + 1

        fig = px.histogram(
            scores,
            nbins=num_bins,
            title='Distribution of Scores',
            labels={'value': 'Score', 'count': 'Number of Students'}
        )

        # Add grade boundary lines
        for grade, min_score in grade_min.items():
            fig.add_vline(
                x=min_score,
                line_dash="dash",
                line_color="gray",
                annotation_text=grade
            )

        st.plotly_chart(fig)
```

Streamlit: Grading App (4)

```
        # Calculate grades using current criteria
        grades = [get_grade(score, grade_min) for score in scores]
        grade_counts = pd.Series(grades).value_counts().sort_index()

        # Display grade distribution
        st.header('Grade Distribution')
        st.bar_chart(grade_counts)

        # Show detailed results
        st.header('Detailed Results')
        results_df = pd.DataFrame({
            'Score': scores,
            'Grade': grades
        })
        st.dataframe(results_df)

        # Export option
        st.download_button(
            "Download Results",
            results_df.to_csv(index=False),
            "grades.csv",
            "text/csv"
        )

if __name__ == '__main__':
    main()
```

Streamlit: Sentiment Analyzer

`streamlit_sentiment_analyzer.py`

Sentiment Analyzer Based On Text Analysis

adapted from
https://github.com/patidarparas13/Sentiment-Analyzer-Tool

View Dataset

- [ ] Show Raw Dataset
- [ ] Show Preprocessed Dataset

- [ ] Show Data Analysis and Model Performance

Analyze Your Text

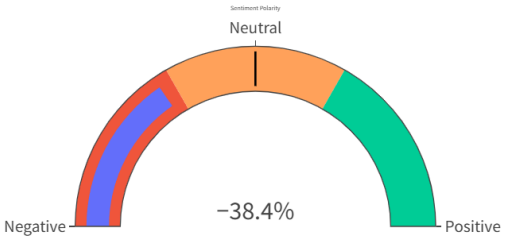
Enter Text for Analysis

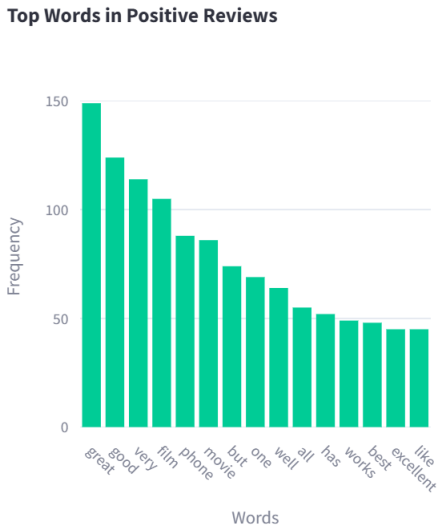
It made me have a liver ache

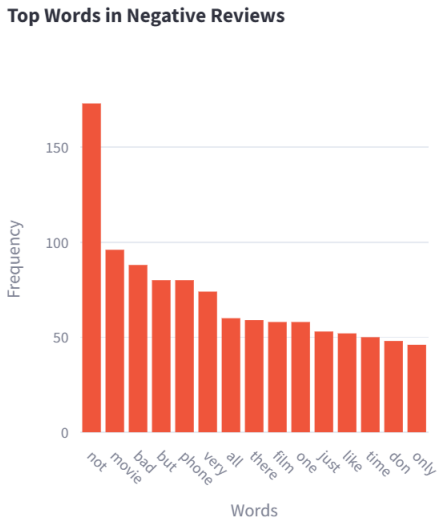
Analyze Sentiment

Sentiment
Negative

Confidence
Moderate Confidence







Model Performance Analysis