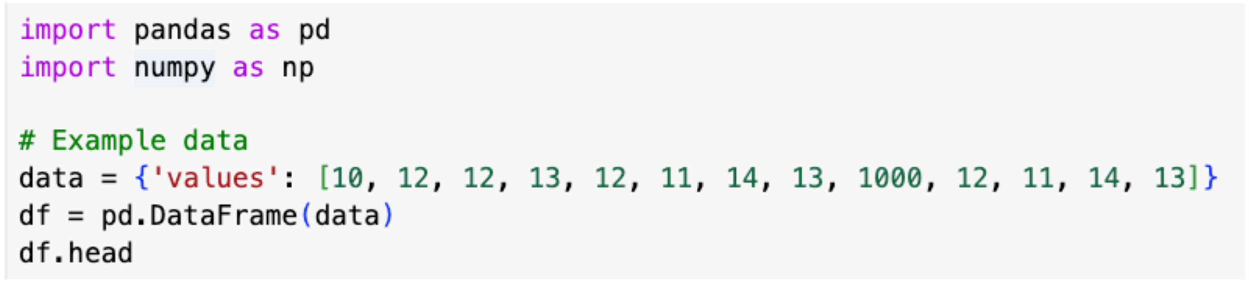
Lecture 12 Visualization

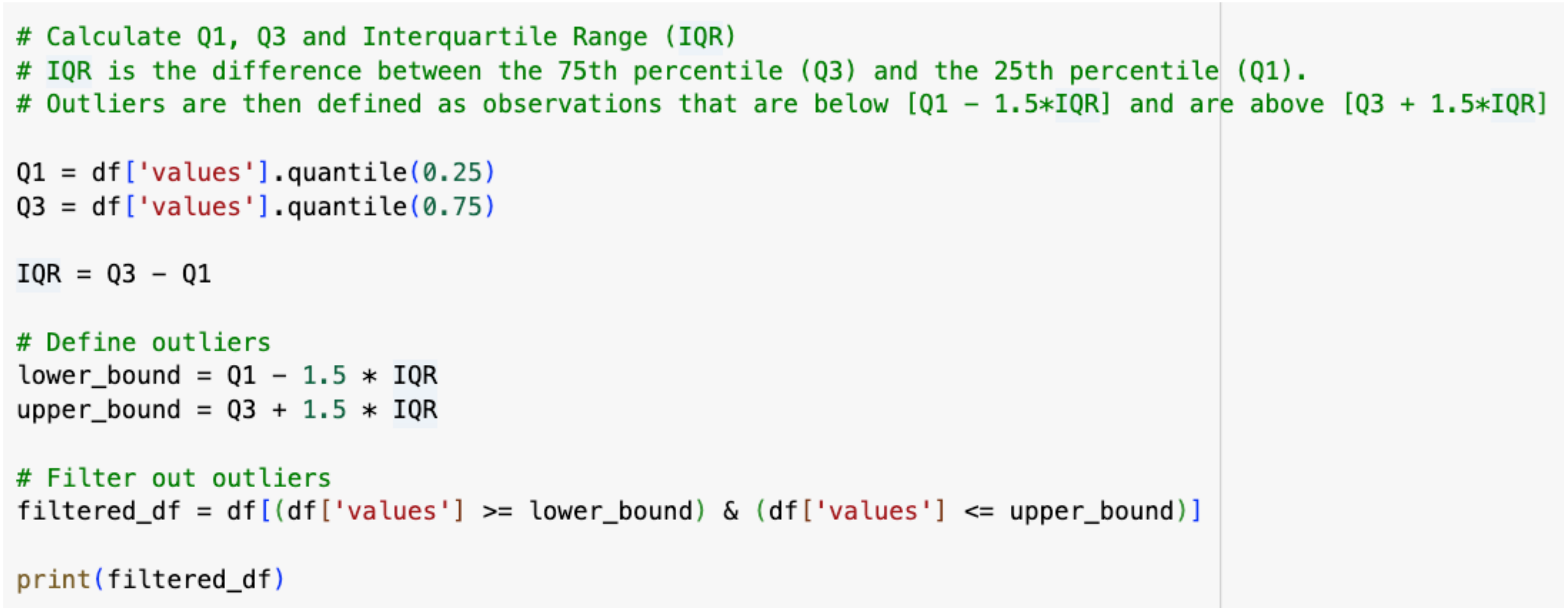
IDS 400

Programming for Data Science in Business

Removing Outliers



```python
import pandas as pd
import numpy as np

# Example data
data = {'values': [10, 12, 12, 13, 12, 11, 14, 13, 1000, 12, 11, 14, 13]}
df = pd.DataFrame(data)
df.head
```



```python
# Calculate Q1, Q3 and Interquartile Range (IQR)
# IQR is the difference between the 75th percentile (Q3) and the 25th percentile (Q1).
# Outliers are then defined as observations that are below [Q1 - 1.5*IQR] and are above [Q3 + 1.5*IQR]

Q1 = df['values'].quantile(0.25)
Q3 = df['values'].quantile(0.75)

IQR = Q3 - Q1

# Define outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out outliers
filtered_df = df[(df['values'] >= lower_bound) & (df['values'] <= upper_bound)]

print(filtered_df)
```

Visualization

Data is only good as it's presented.

Humans more easily grasp information through visualization. In a business context, visualization helps convey a story to decision makers, and help them understand how the business data is being interpreted to determine business decisions. Visualization can help handle large amounts of data in a pictorial format to provide a summary of unseen patterns in the data and reveal insights behind the data to establish a business goal.

Visualization Tools

- Basic charts
  - Histogram
  - Scatter plot
  - Bubble plot
  - Heatmap
  - Barplot
  - WordCloud
  - Pie plot
  - Line plot
  - Stacked area plot
- Maps
- Interactive chart

Visualization Packages

- Basic packages
  - Pandas
  - Matplotlib
  - Seaborn
- Maps
  - Folium
- Interactive visualization
  - Bokeh

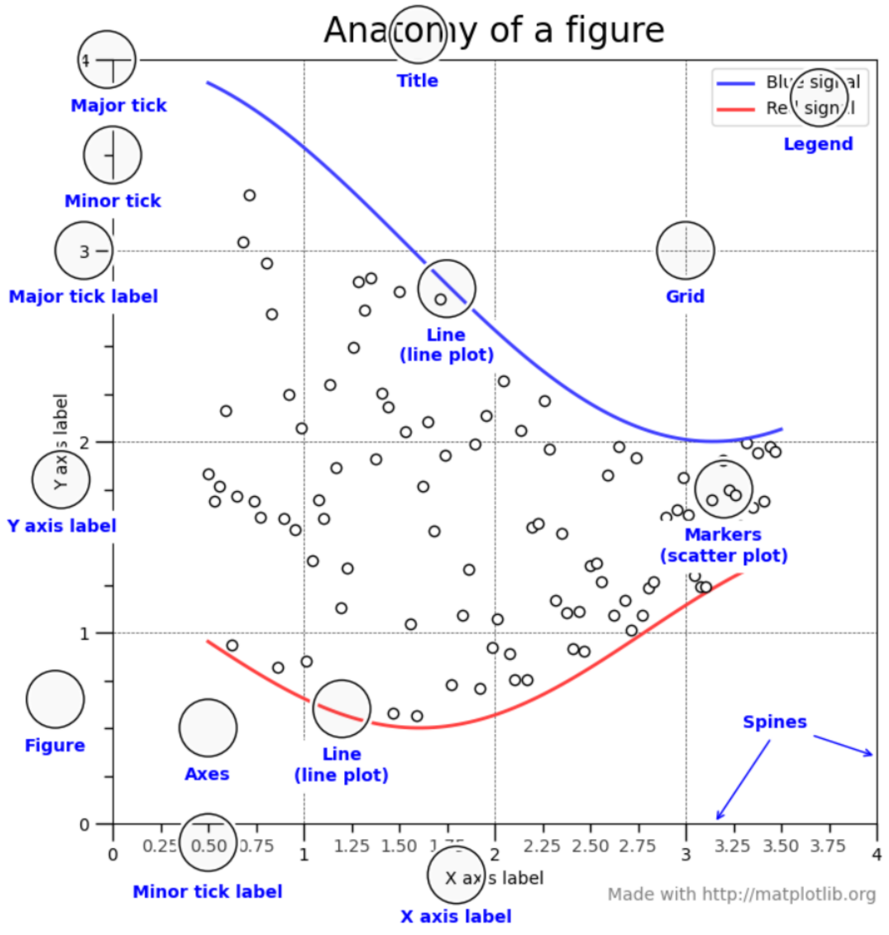
Matplotlib

- *Matplotlib* is one of the first visualization tool in Python. Many other visualization packages such as Seaborn are built on top of Matplotlib.
- It is the python version of visualization in Matlab. It is a powerful yet a bit complicated visualization tool as you need to customize many properties of a figure.
- In many cases, we can also combine matplotlib with other packages.

Anatomy of Matplotlib

- This figure shows the name of several matplotlib elements composing a figure.
- Here is the code of producing the plot. You can look up the corresponding code when you want to include a specific element.

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



Seaborn

- *Seaborn* is a data visualization library built on top of matplotlib and closely integrated with Panda's data structures in Python.
- Seaborn utilizes fascinating themes, while matplotlib used for making basic graphs.

Let us start with something simple

- First of all, you need to think about what aspect of data you want to present:

  - Distribution
  - Correlation
  - Comparison
  - Time trend
  - Maps

Let us start with something simple

- First of all, you need to think about what aspect of data you want to present:

Distribution	Histogram
Correlation	Scatter plot
Comparison	Bar chart
Time trend	Line chart
Maps	…

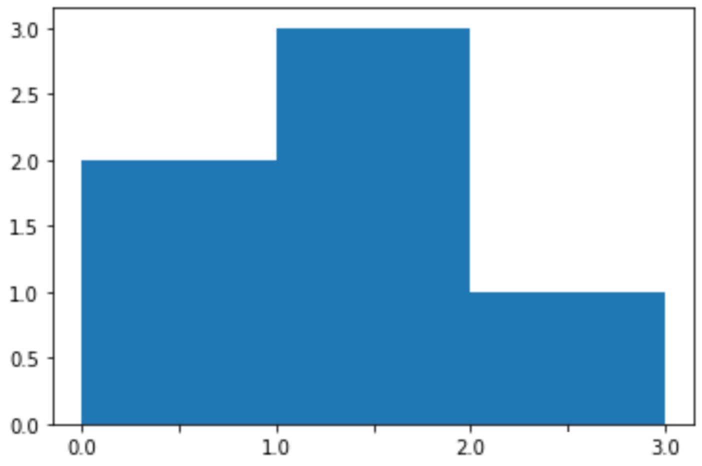
Distribution

- Histogram
- Density plot

Histogram using matplotlib

- Example from Lecture Numpy

```
import matplotlib.pyplot as plt

import numpy as np

x = np.array([0.5, 0.7, 1.0, 1.2, 1.3, 2.1])

bins1 = np.array([0, 1, 2, 3])

print("ans=\n", np.histogram(x, bins1))
```

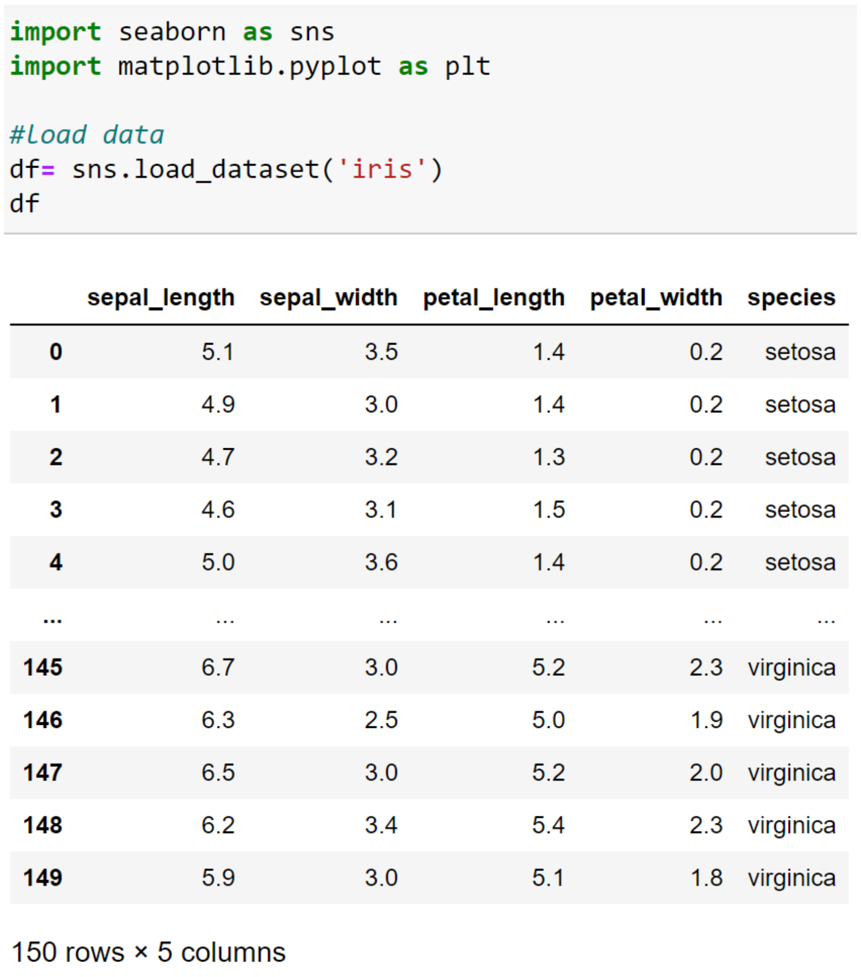


```
ans=
 (array([2, 3, 1], dtype=int64), array([0, 1, 2, 3]))
```

```
plt.hist(x, bins=bins1)

plt.show();
```

In some cases, you need this semicolon to display the figure in jupyter notebook

Data Description



```python
import seaborn as sns
import matplotlib.pyplot as plt

#load data
df= sns.load_dataset('iris')
df
```

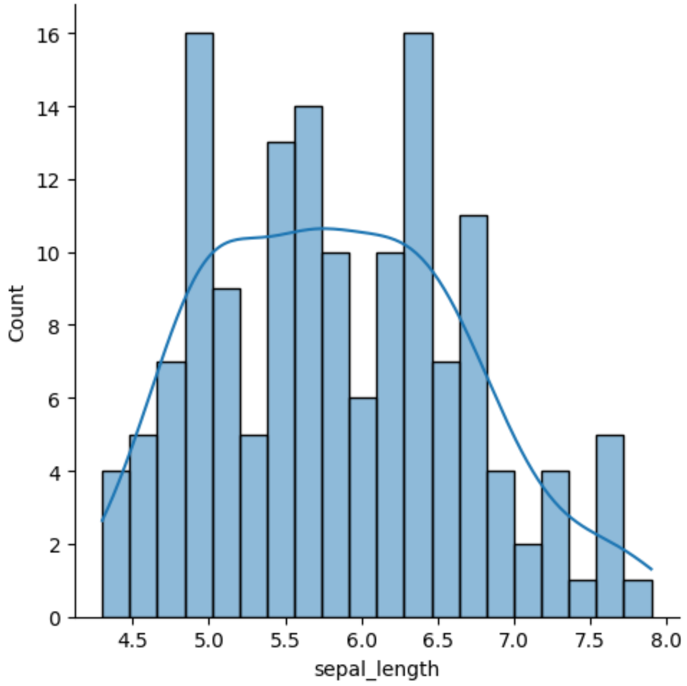
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

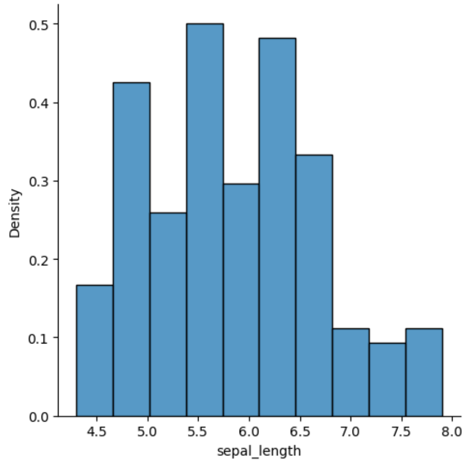
Count/Density plot using Seaborn

- ***seaborn.displot()*** shows a univariate distribution of observations with a line on it.
- The y-axis is the count.

```python
import seaborn as sns
import matplotlib.pyplot as plt

#load data
df= sns.load_dataset('iris')

#displot function create a histogram
sns.displot( df["sepal_length"], bins = 20 ,  kde=True)
plt.show()
```



```python
#displot function create a histogram
sns.displot( df["sepal_length"], bins = 10 ,  stat="density" )
plt.show()
```



Sometimes, if the plot is not showing up, call the function plt.show()

Note: histogram shows the counts of values in each range, while a density plot shows the proportion of values in each range

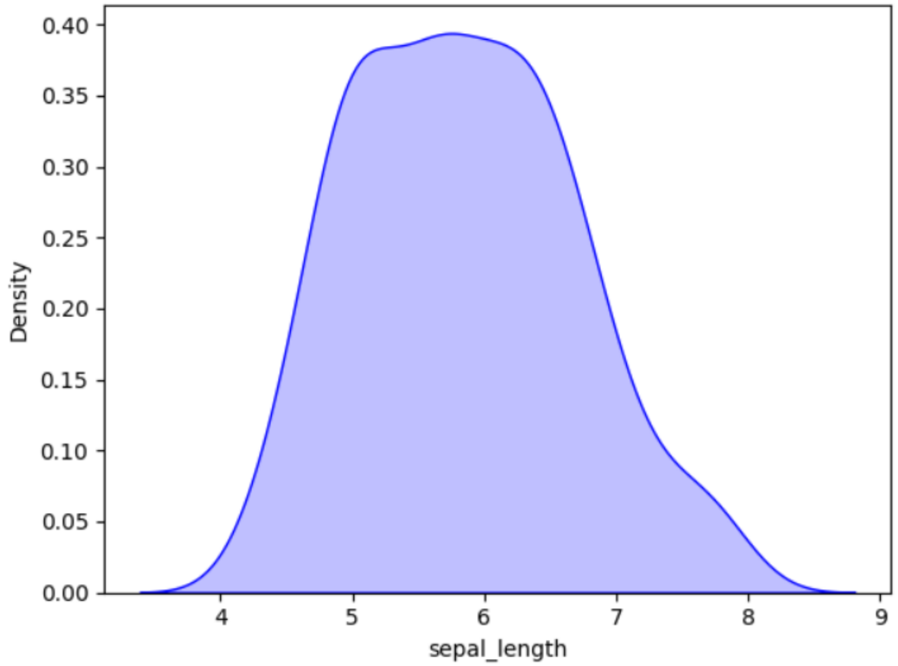
Distribution

- If you *just want the curve not the histogram*, you can use ***kdeplot*** function.

  It is another way of generating distribution plot.

```
sns.kdeplot(df["sepal_length"], fill= True, color = "b")
```

```
<Axes: xlabel='sepal_length', ylabel='Density'>
```

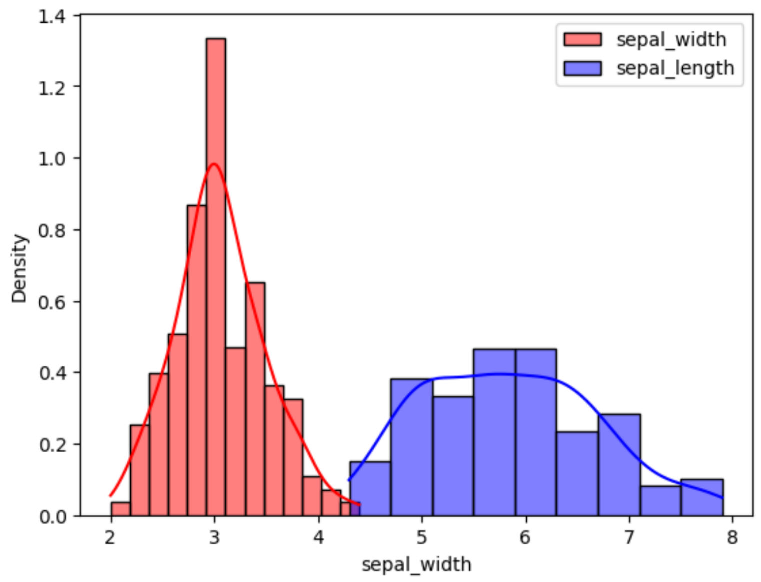


Distribution

- You can use *histplot* or *kdeplot* to draw multiple histograms too.

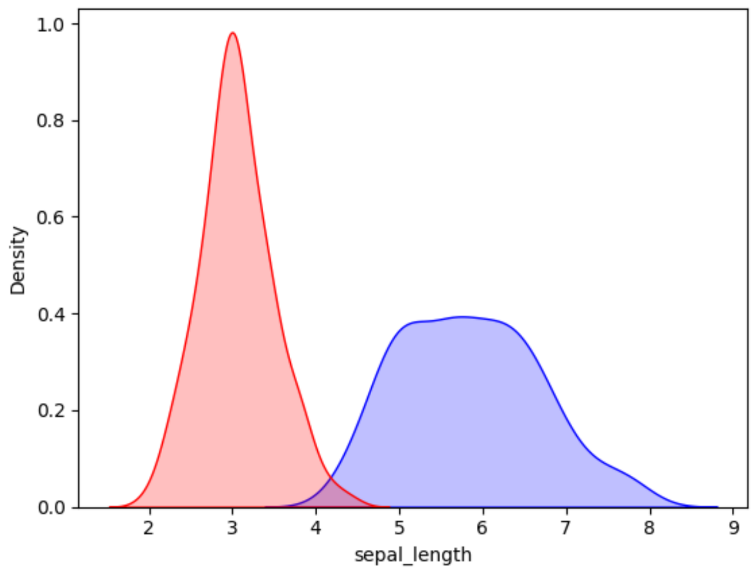
```
sns.histplot( df["sepal_width"], label= 'sepal_width', color = 'r', kde=True, stat="density" )
sns.histplot( df["sepal_length"], label= 'sepal_length', color = 'b', kde=True, stat="density" )
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fde4adb7eb0>
```



```
sns.kdeplot(df['sepal_length'], fill=True, color="b")
sns.kdeplot(df['sepal_width'], fill=True, color="r")
```

```
<Axes: xlabel='sepal_length', ylabel='Density'>
```

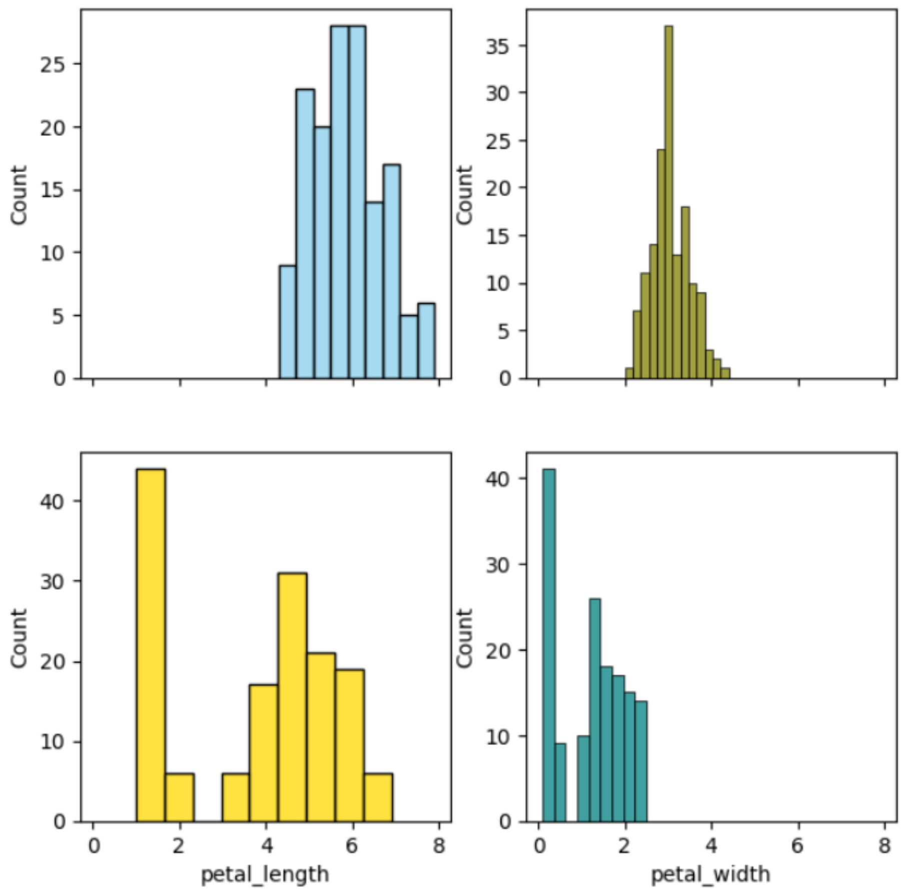


Multiple Plots

```
#using subplot to create multiple histograms together.
f, axes = plt.subplots(2, 2, figsize=(7, 7), sharex=True)
sns.histplot( df["sepal_length"] , color="skyblue", ax=axes[0, 0])
sns.histplot( df["sepal_width"] , color="olive", ax=axes[0, 1])
sns.histplot( df["petal_length"] , color="gold", ax=axes[1, 0])
sns.histplot( df["petal_width"] , color="teal", ax=axes[1, 1])
```

```
<Axes: xlabel='petal_width', ylabel='Count'>
```

- Creating multiple histogram together using ***subplot***.
- This is a function from matplotlib package.
- *sharex* indicates whether subplots will share x-axis range or not.

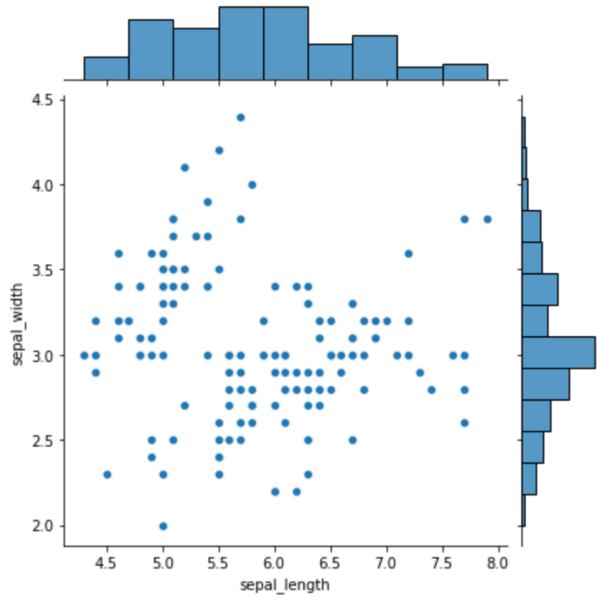


Joint Distribution - make it 2D

- Even better, if we want 2D histogram for multiple variables (i.e. joint distribution)
- There are different types of graph you can choose:
  - scatter, kde, hex, resid, reg, etc…

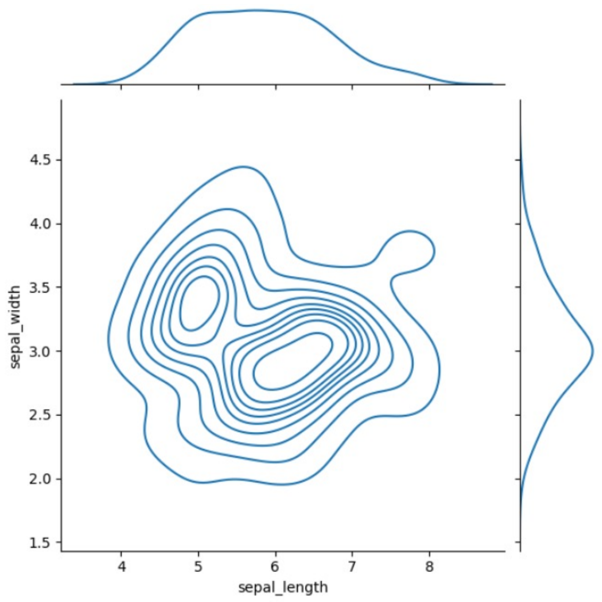
```
sns.jointplot(x=df["sepal_length"], y=df["sepal_width"], kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x233d48a3c40>
```



```
sns.jointplot(x=df["sepal_length"], y=df["sepal_width"], kind='kde')
```

```
<seaborn.axisgrid.JointGrid at 0x7fde43ec4220>
```



Regression

```
sns.kdeplot(x= df['sepal_length'], y = df['sepal_width'], fill=True, color="b")
```

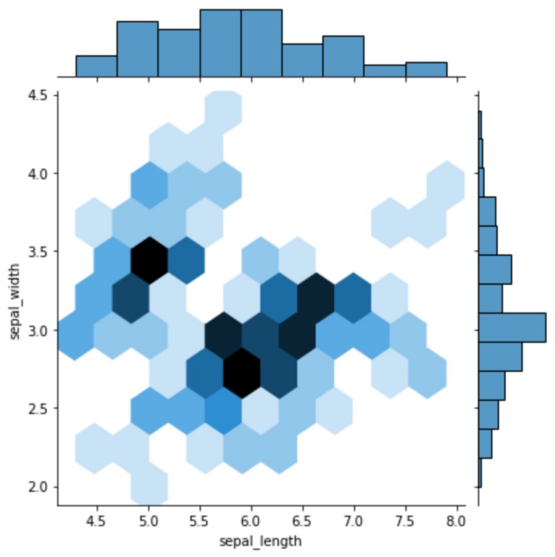
```
import pandas as pd
import statsmodels.api as sm

X = df["sepal_width"]
y = df["sepal_length"]

# Add a constant to the model (the intercept term)
X = sm.add_constant(X)

# Fit the model
model = sm.OLS(y, X).fit()
print(model.summary())
```

Joint Distribution - make it 2D

- Even better, if we want 2D histogram for multiple variables (i.e. joint distribution)
- There are different types of graph you can choose:
  - scatter, kde, hex, resid, reg, etc…

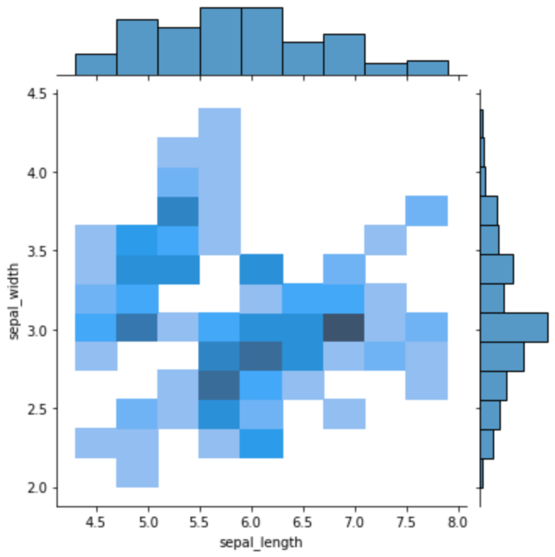
```
sns.jointplot(x=df["sepal_length"], y=df["sepal_width"], kind='hex')
```

```
<seaborn.axisgrid.JointGrid at 0x233d44075e0>
```



```
sns.jointplot(x=df["sepal_length"], y=df["sepal_width"], kind='hist')
```

```
<seaborn.axisgrid.JointGrid at 0x233d4548100>
```

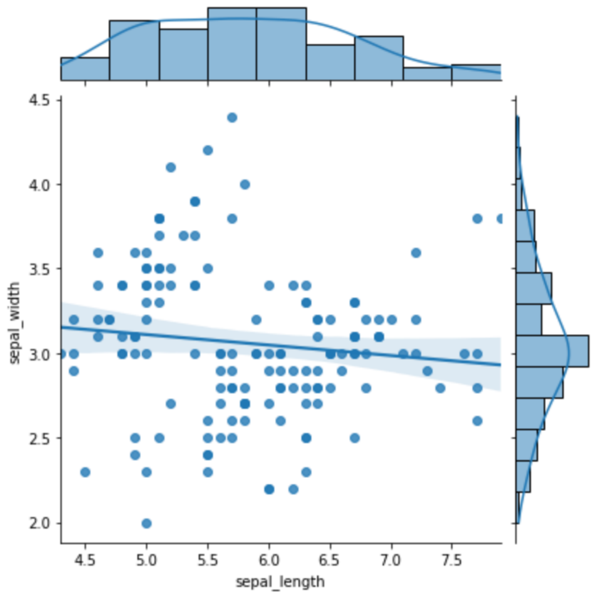


Joint Distribution - make it 2D

- Even better, if we want 2D histogram for multiple variables (i.e. joint distribution)
- There are different types of graph you can choose:
  - scatter, kde, hex, resid, reg, etc…

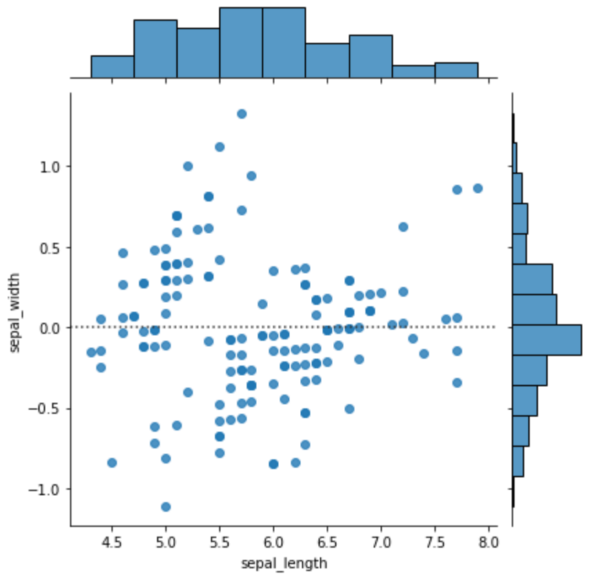
```
sns.jointplot(x=df["sepal_length"], y=df["sepal_width"], kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x233d467ed00>
```



```
sns.jointplot(x=df["sepal_length"], y=df["sepal_width"], kind='resid')
```

```
<seaborn.axisgrid.JointGrid at 0x233d4780f10>
```



Information about Residual Plot

Residual plots are a diagnostic tool and can be used to improve the model by identifying the nature of the residuals and any non-random patterns that may be present.

Here's what a residual plot can tell you about your regression model:

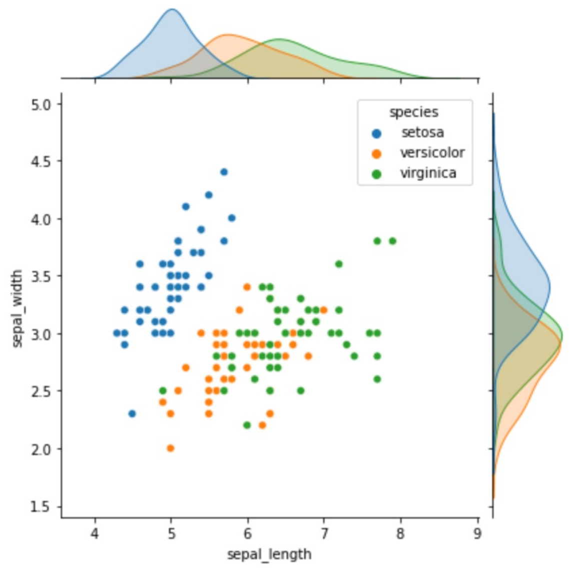
1. **Homoscedasticity**: If the residuals are equally spread throughout the range of predictors, this indicates homoscedasticity, which is an assumption of linear regression.
2. **Heteroscedasticity**: If the residuals fan out or form a pattern as the predictors increase or decrease, this indicates heteroscedasticity, which means that the variance of the errors is not constant.
3. **Linearity**: The residual plot can show if there's a linear relationship between the independent and dependent variables. If the residuals seem to form a particular pattern (like a curve), this suggests that the relationship is not linear.
4. **Outliers**: Outliers can also be identified in a residual plot. They are points with a large distance from the horizontal axis (either high above or below it).
5. **Influential Data Points**: These are points that, if removed, would significantly change the estimate of the regression equation. They can often be identified in a residual plot because they are far from the general cloud of points.
6. **Model Fit**: Overall, the residual plot can be used to check the goodness of fit for a linear regression model. If the model is well-fitted, the residuals would be randomly scattered around zero without forming specific patterns.

Joint Distribution - make it 2D

- Assigning a ***hue*** variable will add conditional colors to the plot and draw separate density curves on the marginal axes.

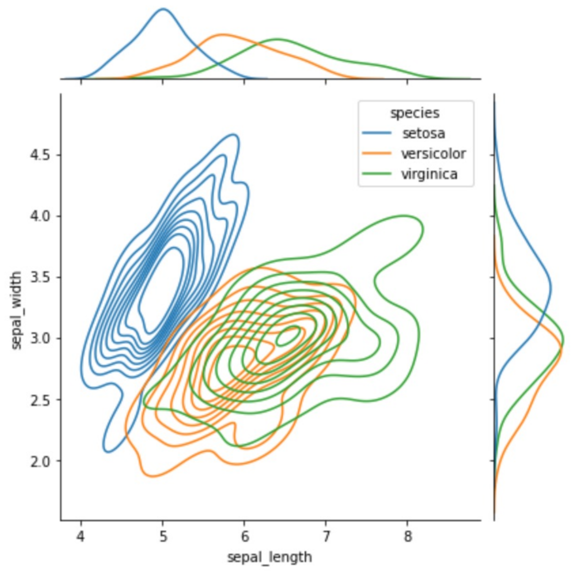
```
sns.jointplot(data=df, x="sepal_length", y="sepal_width", hue="species")
```

```
<seaborn.axisgrid.JointGrid at 0x233d5ed8580>
```



```
sns.jointplot(data=df, x="sepal_length", y="sepal_width", hue="species", kind="kde")
```

```
<seaborn.axisgrid.JointGrid at 0x233d6024430>
```



Correlation

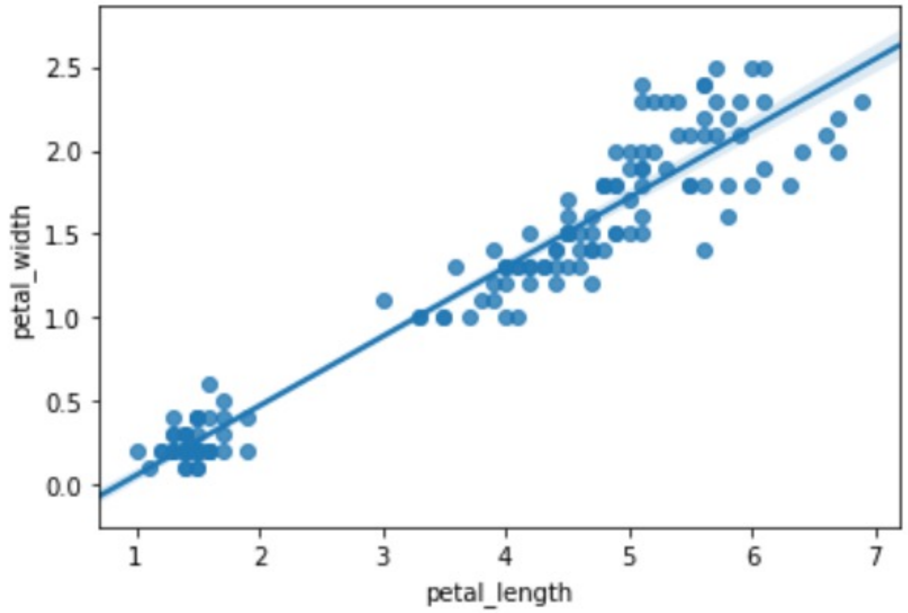
- The joint distribution figure we showed earlier can be used to demonstrate correlation.
- Other figures showing correlation also includes:
    - Scatter plot
    - Contour plot
    - Heatmap
    - Bubble plot

Scatter plot

- We want to find out the relationship between petal length and petal width.
- *Regplot* can generate a scatter plot as well as a regression.

```
sns.regplot(x=df["petal_length"], y=df["petal_width"])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x14799caecc8>
```

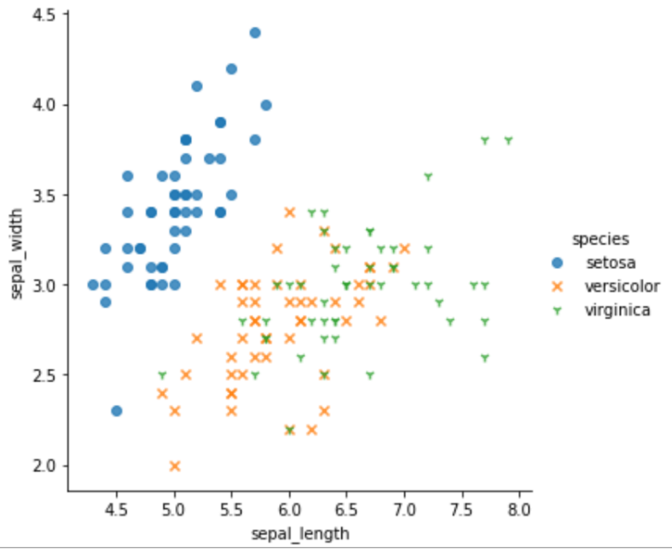


Scatter plot

- How this relationship is different across different iris type?
- We can further use ***lmplot*** as an advanced version of regplot to plot observations with different labels.

```
# fit_reg: whether show regression line
# hue: variable defining the label of data
# legend: whether show legend
# markers: assign different marker for each label

sns.lmplot( x="sepal_length", y="sepal_width", data=df, fit_reg=False,
           hue='species', legend=True, markers=["o", "x", "1"])
```

```
<seaborn.axisgrid.FacetGrid at 0x1deebae0c08>
```

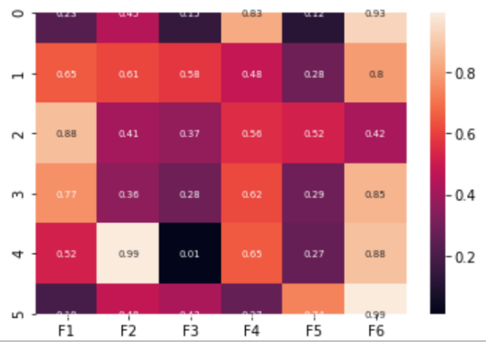


Heatmap

- The heatmap is a way of representing the data in a 2-dimensional form to provide a colored visual summary of information.
- The data values are represented as colors in the graph.
- You can use ***heatmap()*** to plot heatmaps.

```python
import pandas as pd
import numpy as np
df= pd.DataFrame(np.random.random((6,6)),columns=["F1","F2","F3","F4","F5","F6"])
print(df)

# annot: If True, write the data value in each cell.
# annot_kws: Keyword arguments for ax.text when annot is True.
sns.heatmap(df, annot=True, annot_kws={"size": 7})
```

```
         F1        F2        F3        F4        F5        F6
0  0.231058  0.448930  0.147859  0.828728  0.116866  0.934627
1  0.654130  0.613930  0.580342  0.483666  0.283013  0.796370
2  0.875352  0.411784  0.373912  0.556326  0.524377  0.420391
3  0.769515  0.361407  0.283456  0.618964  0.287965  0.853895
4  0.521347  0.992872  0.010315  0.651425  0.271167  0.883886
5  0.191486  0.481789  0.426781  0.265089  0.738887  0.990333

<matplotlib.axes._subplots.AxesSubplot at 0x1dee9c1fd88>
```



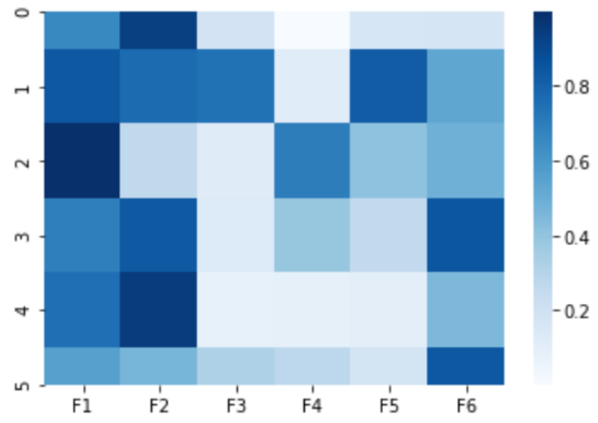
Heatmap

- Change the color of Heatmap using parameter *cmap*
- More details on color palettes in Seaborn:

  https://seaborn.pydata.org/tutorial/color_palettes.html

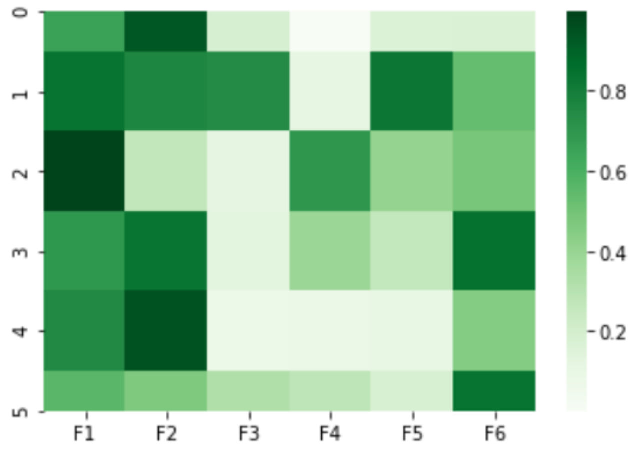
```
#heatmap using different colors:
sns.heatmap(df, cmap="Blues")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1479ae9c708>
```



```
sns.heatmap(df, cmap="Greens")
```

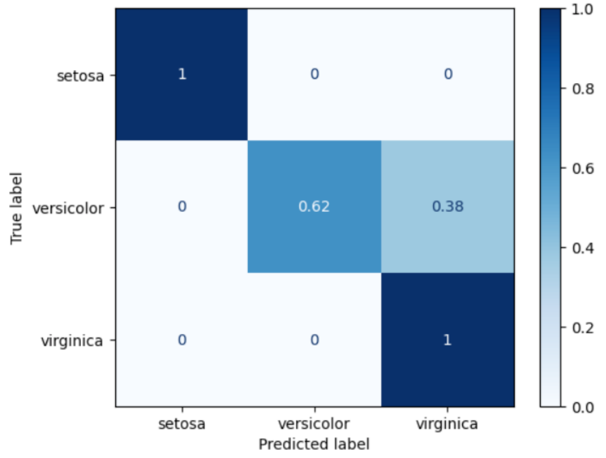
```
<matplotlib.axes._subplots.AxesSubplot at 0x1479af40f08>
```



Display Confusion Matrix

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

from sklearn import datasets, svm
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split

# import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target
class_names = iris.target_names

# Split the data into a training set and a test set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Run classifier, using a model that is too regularized (C too low) to see
# the impact on the results
classifier = svm.SVC(kernel="linear", C=0.01).fit(X_train, y_train)

disp = ConfusionMatrixDisplay.from_estimator(classifier, X_test, y_test,
                                             display_labels=class_names,
                                             cmap=plt.cm.Blues,normalize= normalize)

print(disp.confusion_matrix)

plt.show()
```

```
[[1.   0.   0.  ]
 [0.   0.62 0.38]
 [0.   0.   1.  ]]
```



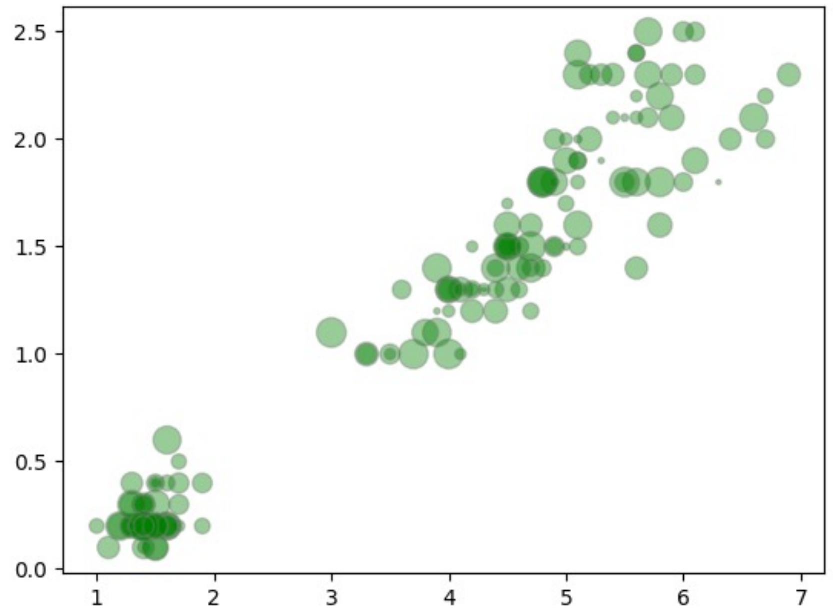
https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

Bubble plot

- Let’s go back to matplotlib, and use the ***scatter()*** function

```
size = np.random.rand(150)
df = sns.load_dataset('iris')

# s: size of the bubbles
# c: color palette
# alpha: adjust the transparency level
plt.scatter(df["petal_length"], df["petal_width"],
            s = size*200, c="g", alpha=0.4, edgecolors="grey")
```

```
<matplotlib.collections.PathCollection at 0x7fde4582de10>
```



Comparison

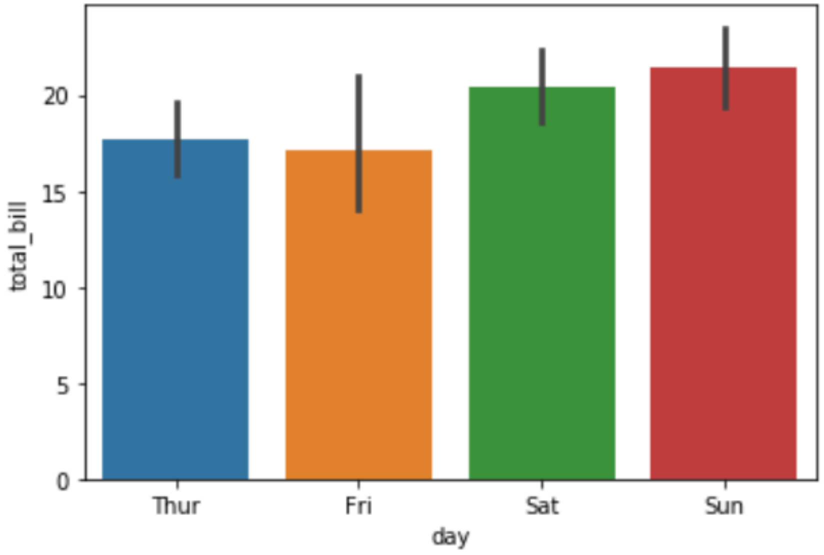
- Bar chart
- Box-whisker plot

Bar chart

- Create a bar chart based on certain categories.

```
tips = sns.load_dataset("tips")
print(tips.head())
ax = sns.barplot(x="day", y="total_bill", data=tips)
```

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

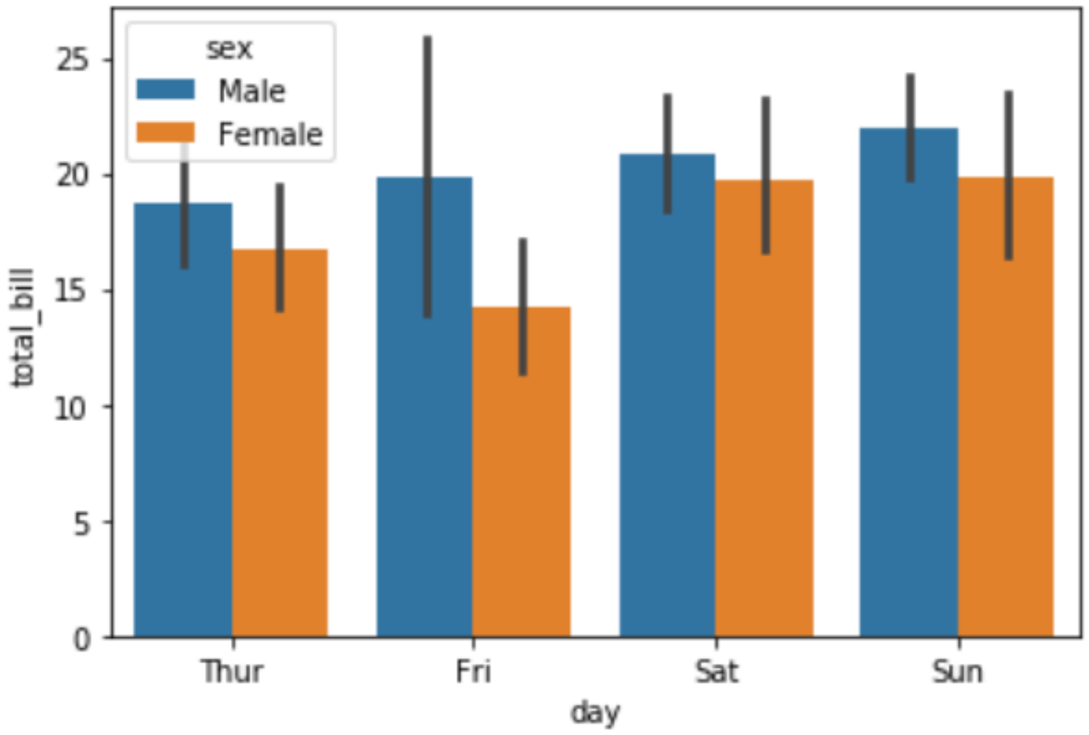


This one shows the average bill. The vertical line on top shows the confidence interval (interpret it as standard deviation)

Bar chart

- If you want to further compare tips from waiter/waitress:

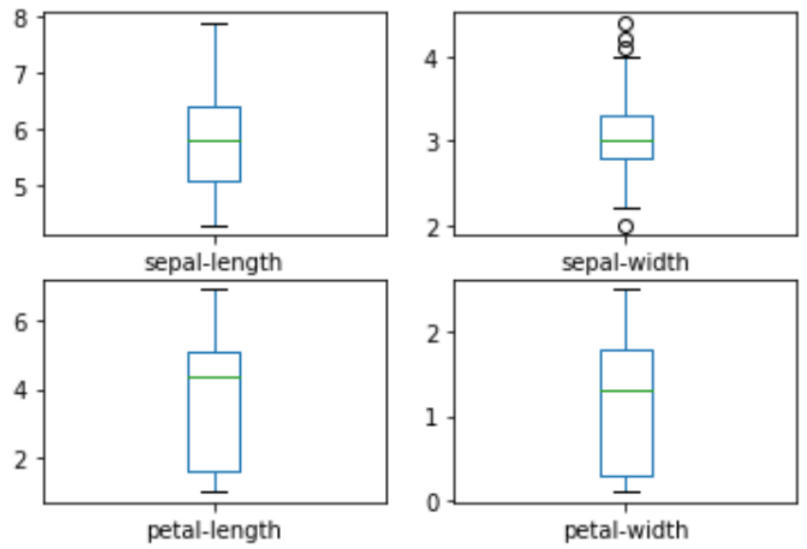
```
ax = sns.barplot(x="day", y="total_bill", hue="sex", data=tips)
```

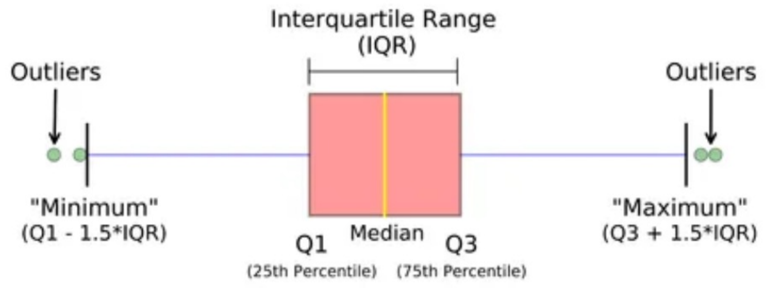


Similar to previous lmplot() functions, parameter ***hue*** is used for different labels

Box plot from last lecture

```
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
```



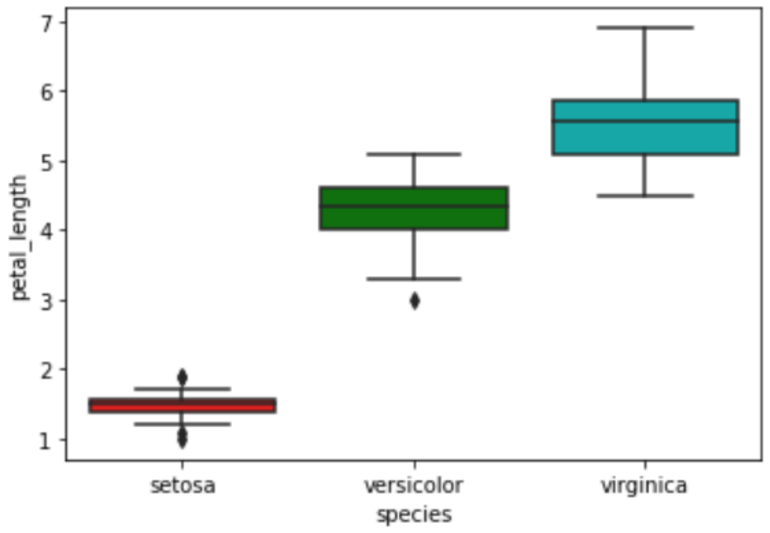


Box-whisker plot

- Going back to the iris dataset.
- We want to compare petal length of different iris.
- If you want to customize the color for each box:

```
custom_pal= {"versicolor": "g", "setosa": "r", "virginica":"c"}
sns.boxplot( x=df["species"], y=df["petal_length"], palette=custom_pal)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1479b2832c8>
```



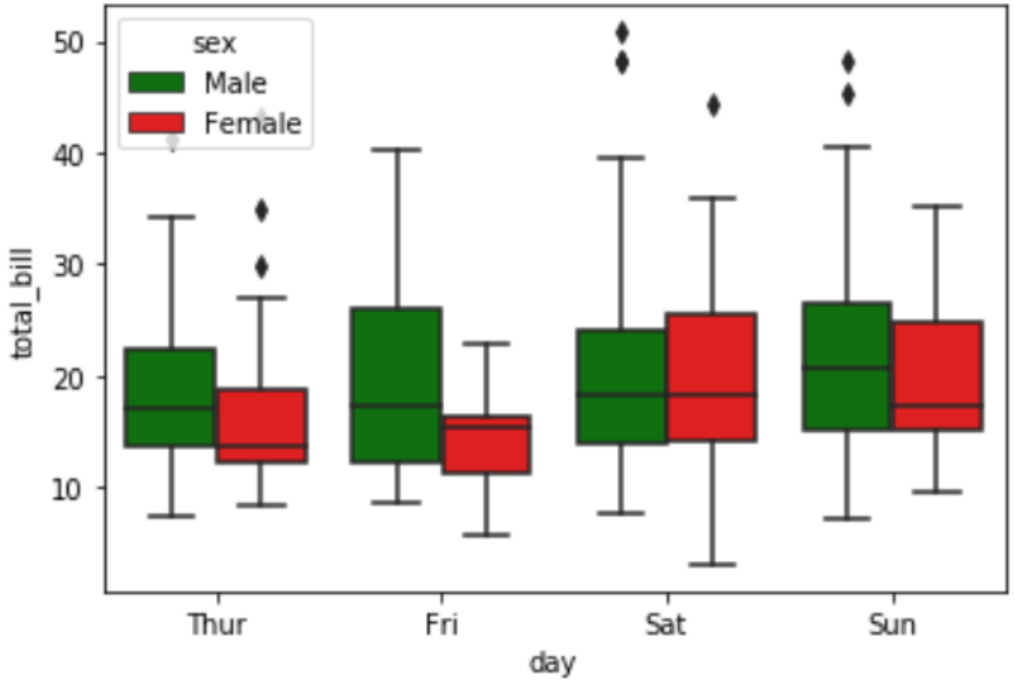
Specify a color in Matplotlib: https://matplotlib.org/stable/tutorials/colors/colors.html

Box-whisker plot

- One more step, if you want to plot box plot for different groups side by side:

```
custom_pal= {"Male": "g", "Female": "r"}
sns.boxplot(x="day", y="total_bill",
            hue="sex",data=tips,palette=custom_pal)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1479b320a88>
```



Time trend

- Line plot
- Stacked area plot

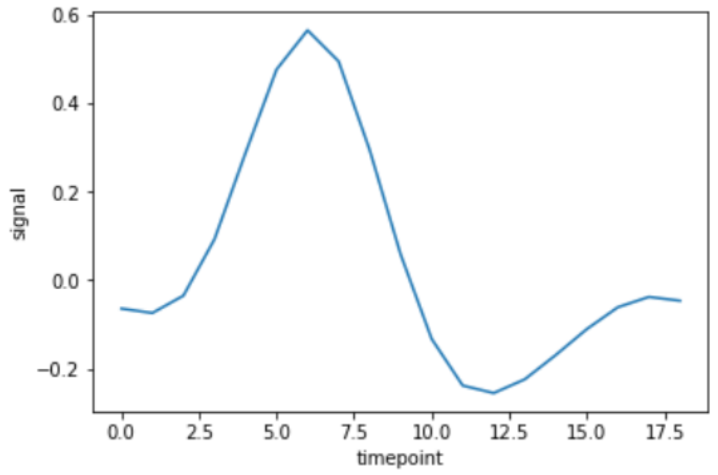
Line plot

- Line plot is often used to capture the time trend.
- The *fmri* dataset is the FMRI results of patients across time.
- We want to plot how FMRI results for patient ***s1*** change over time.

```python
fmri= sns.load_dataset("fmri")
print(fmri.head(20))
dft=fmri[(fmri["subject"]=="s1")&(fmri["region"]=="parietal")&
(fmri["event"]=="stim")]
sns.lineplot(x="timepoint", y="signal", data=dft)
```

```
   subject  timepoint event    region    signal
0      s13         18  stim  parietal -0.017552
1       s5         14  stim  parietal -0.080883
2      s12         18  stim  parietal -0.081033
3      s11         18  stim  parietal -0.046134
4      s10         18  stim  parietal -0.037970
5       s9         18  stim  parietal -0.103513
6       s8         18  stim  parietal -0.064408
7       s7         18  stim  parietal -0.060526
8       s6         18  stim  parietal -0.007029
9       s5         18  stim  parietal -0.040557
10      s4         18  stim  parietal -0.048812
11      s3         18  stim  parietal -0.047148
12      s2         18  stim  parietal -0.086623
13      s1         18  stim  parietal -0.046659
14      s0         18  stim  parietal -0.075570
15     s13         17  stim  parietal -0.008265
16     s12         17  stim  parietal -0.088512
17      s7          9  stim  parietal  0.058897
18     s10         17  stim  parietal -0.016847
19      s9         17  stim  parietal -0.121574
```

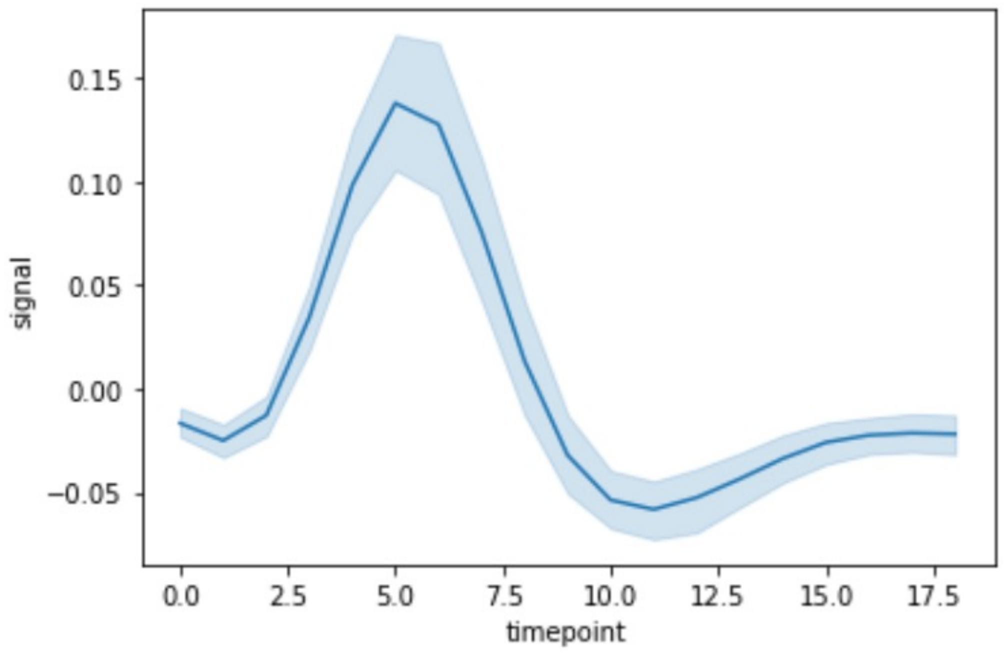
```
<matplotlib.axes._subplots.AxesSubplot at 0x1479b428b08>
```



Line plot

- In some case, there are multiple observations in one period, and you want to know the range of data for each period.
- We want to plot how fmri results change over time across all patients.

```
ax = sns.lineplot(x="timepoint", y="signal", data=fmri)
```

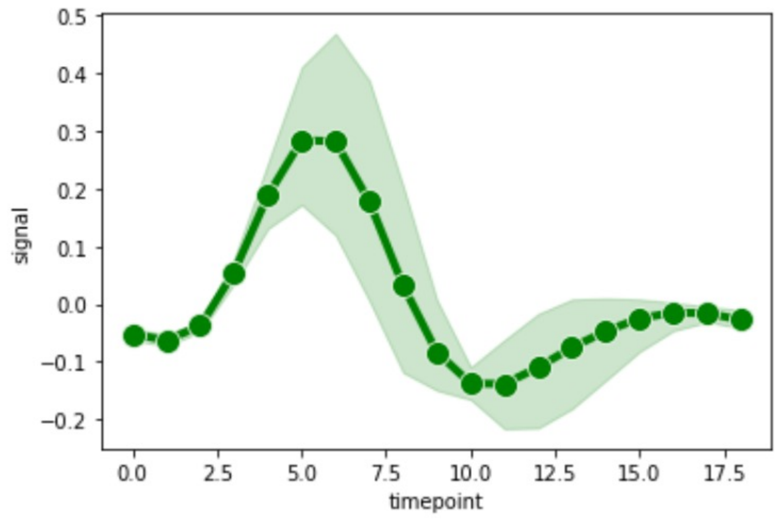


Line plot

- We want FMRI results for patient ***s1*** and ***s2***.
- Multiple lines with different color/marker/linewidth.

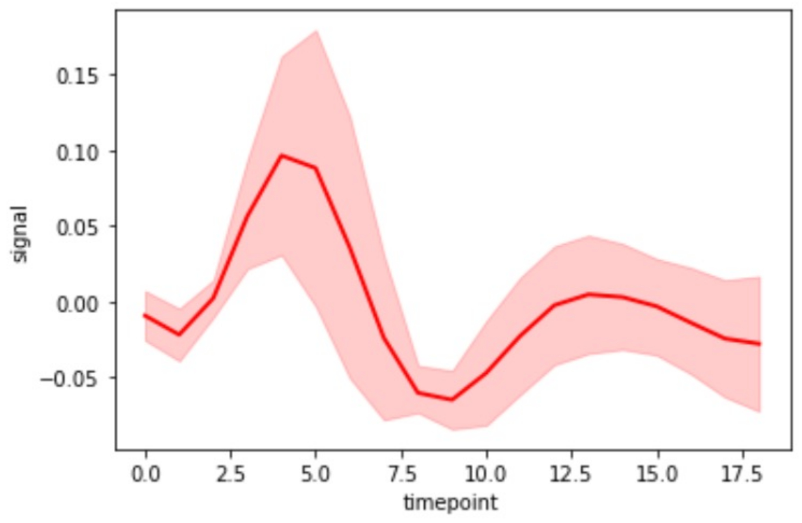
```
dfs1=fmri[(fmri["subject"]=="s1")]
sns.lineplot(x="timepoint", y="signal",
             data=dfs1,marker='o',markersize=12,
             color='g',linewidth=4)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1479b50b648>
```



```
dfs2=fmri[(fmri["subject"]=="s2")]
sns.lineplot(x="timepoint", y="signal",
             data=dfs2,marker='',
             color='r',linewidth=2)
```

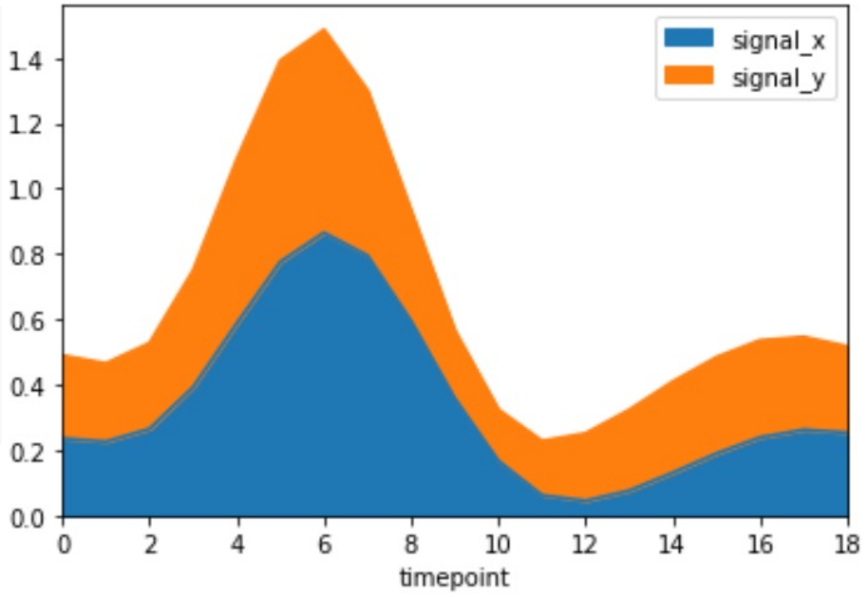
```
<matplotlib.axes._subplots.AxesSubplot at 0x1479b576f48>
```



Stacked area plot

- Now we want to compare parietal and frontal FMRI results over time.
- With pandas, the stacked area charts are made using the plot.area() function.
- We need to processed the data first.

```
dfs1t = fmri[(fmri["subject"]=="s1")&
             (fmri["region"]=="parietal")&
             (fmri["event"]=="stim")]
dfs2t = fmri[(fmri["subject"]=="s1")&
             (fmri["region"]=="frontal")&
             (fmri["event"]=="stim")]
dft = pd.merge(dfs1t,dfs2t,on='timepoint')
dft2c = dft[['timepoint','signal_x','signal_y']]

# When stacked is True, each column must be either all positive or negative
dft2c['signal_x'] = dft2c['signal_x']+0.3
dft2c['signal_y'] = dft2c['signal_y']+0.3
dft2sort = dft2c.sort_values(by=['timepoint'])

ax = dft2sort.plot.area(x='timepoint')
```



https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html

Map

- Some packages:
  - Basemap
  - Folium - a very cool tool based on Java

Folium

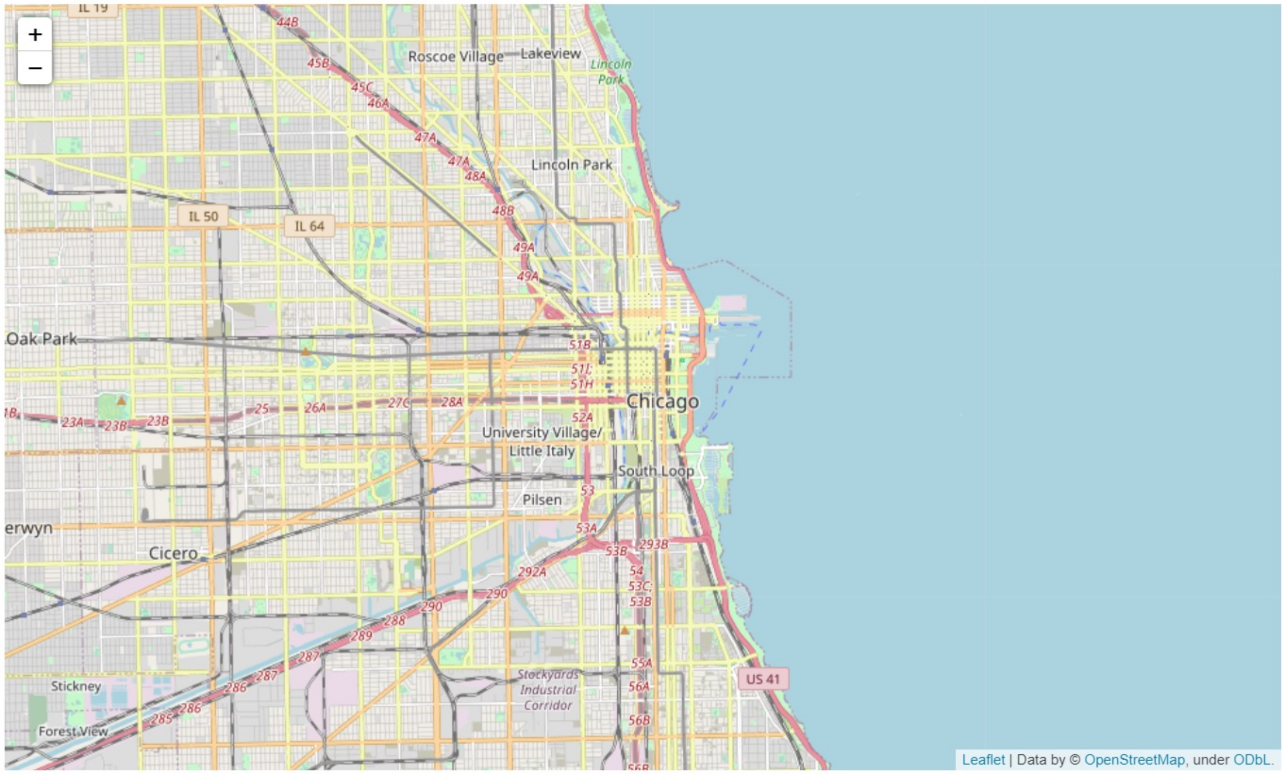
- It is powerful and easy to use.
- It can also be easily embedded in a webpage.
- First, you need to run the application: ***anaconda prompt***
  - *Windows*: Go with the mouse to the Windows Icon (lower left) and start typing "Anaconda". There should show up some matching entries. Select "Anaconda Prompt". A new command window, named "Anaconda Prompt" will open.
  - *macOS*: Cmd+Space to open Spotlight Search and type “Navigator” to open the program.
- When the window pop up, you enter:

conda install -c conda-forge folium

Folium

- Let’s setup the map.
- This create a folium map with initial location and zoom level.

```
import folium
m = folium.Map(location=[41.8781, -87.6298], zoom_start=12)
m
```



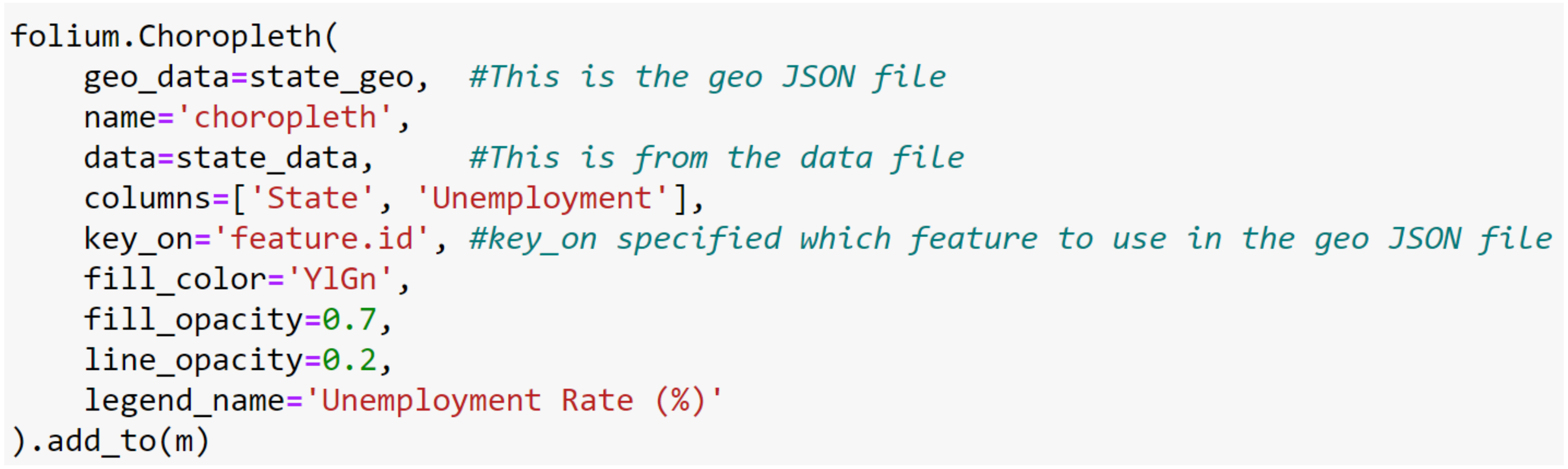
Folium

- Now we want to draw unemployment data of all states on a map.
- Let's read the data on US unemployment data for different states.

```
# read the data
url = 'https://raw.githubusercontent.com/python-visualization/folium/master/examples/data'
state_geo = f'{url}/us-states.json'
state_unemployment = f'{url}/US_Unemployment_Oct2012.csv'
state_data = pd.read_csv(state_unemployment)
```

Folium

- Setup the details of the choropleth map

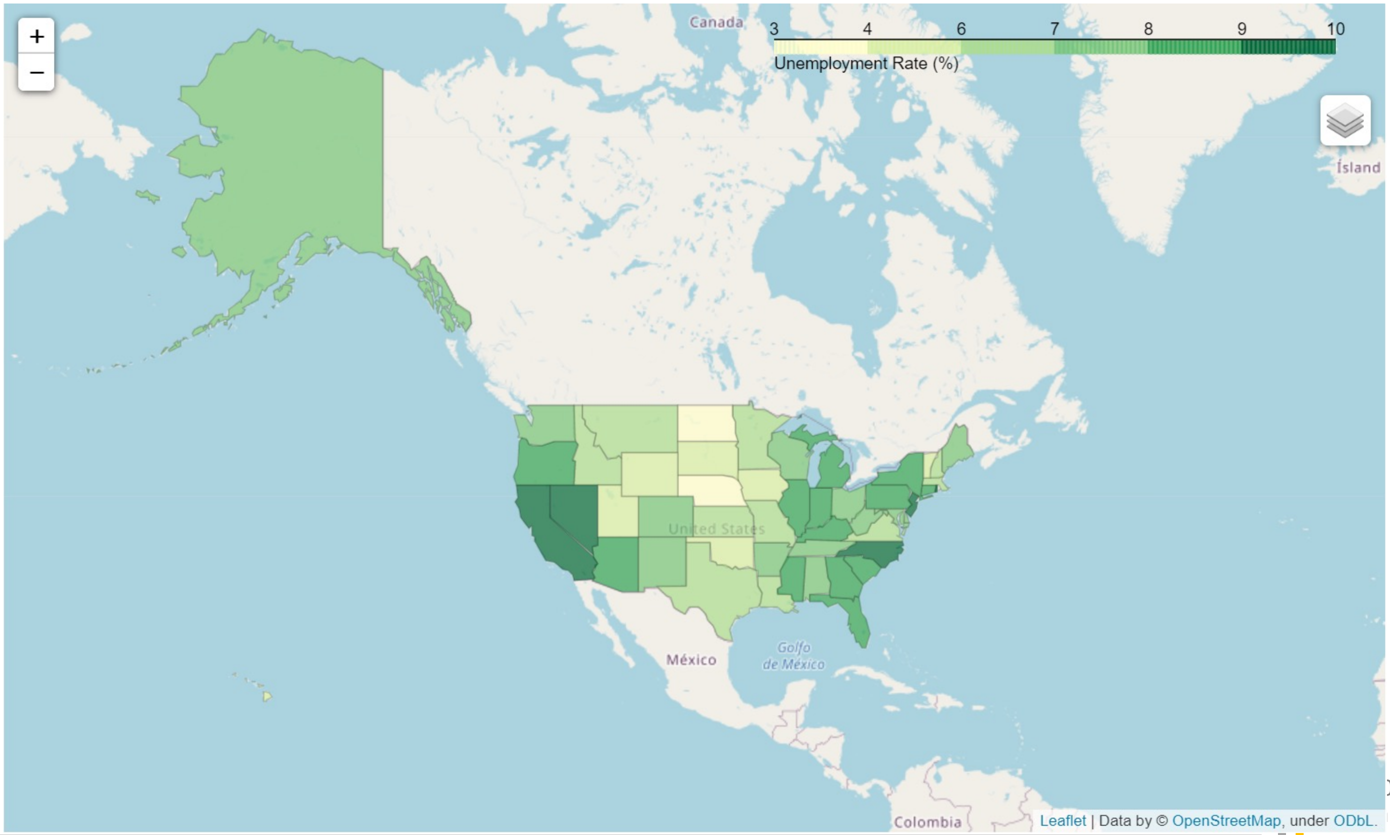


```
folium.Choropleth(
    geo_data=state_geo,  #This is the geo JSON file
    name='choropleth',
    data=state_data,     #This is from the data file
    columns=['State', 'Unemployment'],
    key_on='feature.id', #key_on specified which feature to use in the geo JSON file
    fill_color='YlGn',
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Unemployment Rate (%)'
).add_to(m)
```

Folium

- Continue

```python
folium.LayerControl().add_to(m) # Display the map in Jupyternotebook
m
```



Interactive plot

- Pygal
- Bokeh
- Plotly
- ……

Lab *Visualization*