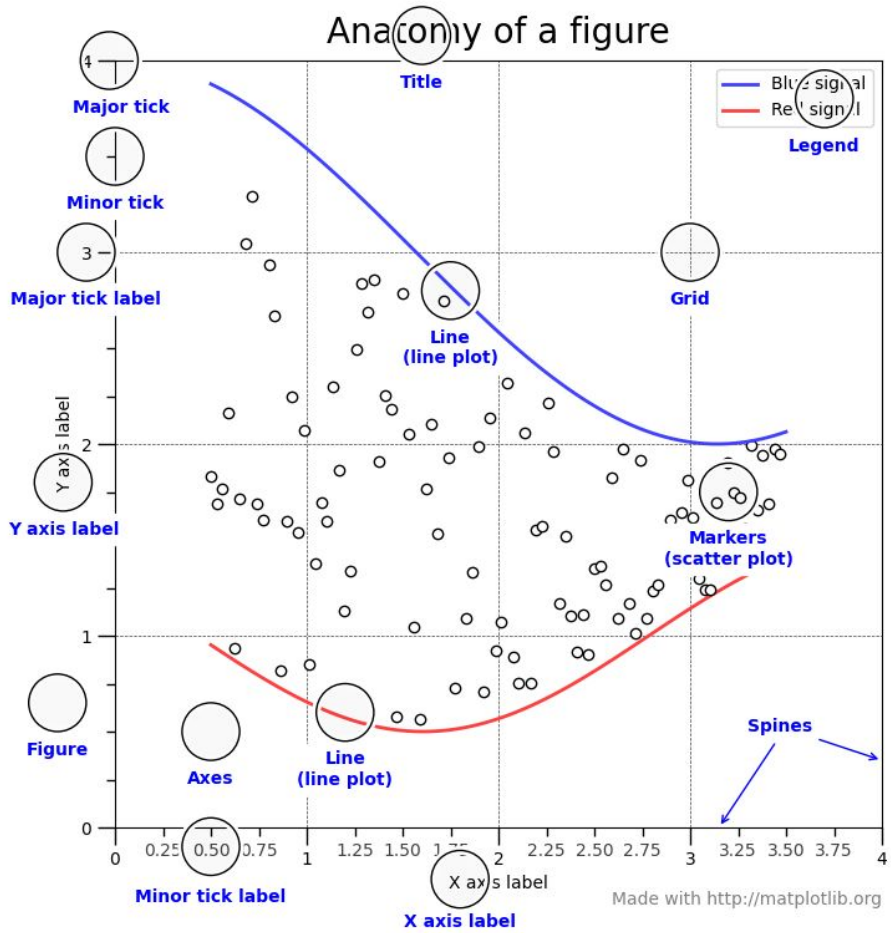
Matplotlib and Seaborn

Become a Graph Viz-ard!



- Matplotlib graphs data onto an object called a Figure
- A Figure can have multiple Axes objects
- Markings on the figure are called artists

Figure 1: Anatomy of a figure - https://matplotlib.org/gallery/showcase/anatomy.html

Two ways to use matplotlib

Object oriented api

- Must create figures and axes manually using `fig, ax = plt.subplots()`
- Plot onto axes using `ax.plot`

PyPlot API

- don't have to manually create and manage plots using subplots()
- use the call `plt.plot(x=...,y=...)`

Two ways to use matplotlib

```
fig, ax = plt.subplots()  # Create a figure and an axes.

ax.plot(x, x, label='linear')  # Plot some data on the axes.

ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...

ax.plot(x, x**3, label='cubic')  # ... and some more.
```

```
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.

plt.plot(x, x**2, label='quadratic')  # etc.

  plt.plot(x, x**3, label='cubic')
```

Two ways to use matplotlib

```
fig, ax = plt.subplots()  # Create a figure and an axes.

ax.plot(x, x, label='linear')  # Plot some data on the axes.

ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...

ax.plot(x, x**3, label='cubic')  # ... and some more.
```

```
plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.

plt.plot(x, x**2, label='quadratic')  # etc.

  plt.plot(x, x**3, label='cubic')
```

Both Create

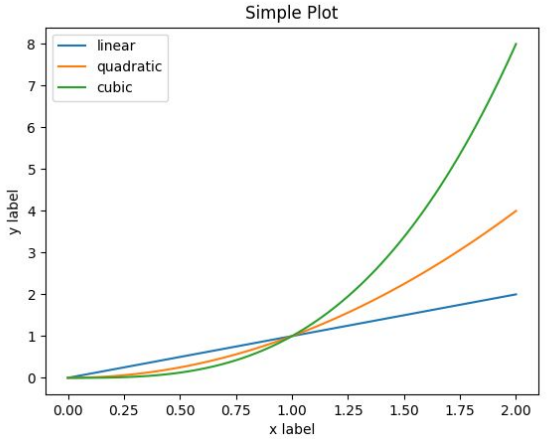


Figure 2:
https://matplotlib.org/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usag

Using seaborn

- extension of Matplotlib that allows the user to easily create well styled graphs
- create graphs by calling seaborn functions such as
  - `sns.barplot(data=...,x=...,y=...)`
- super convenient

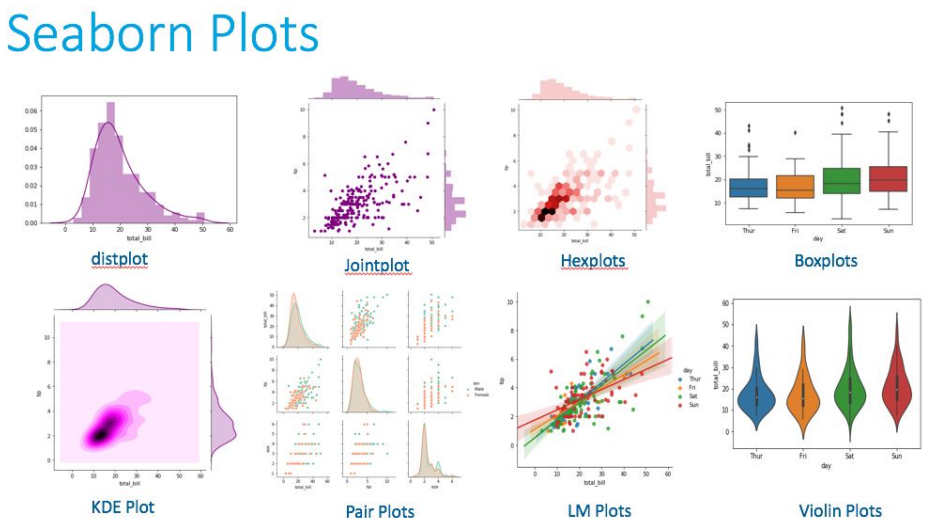


Figure 3: examples of seaborn plots https://medium.com/@mukul.mschauhan/data-visualisation-using-seaborn-464b7c0e5122

Figure vs axes level plots

- relplot, displot, and catplot all create figure level plots, and the rest of the functions underneath them create axis level plots
- For instance, instead of using `sns.scatterplot`, we can use `sns.relplot(data=df, x="x_var", y="y_var", kind="scatter")`
- For line plots, we can use `sns.relplot(data=df, x="x_var", y="y_var", kind="line")`

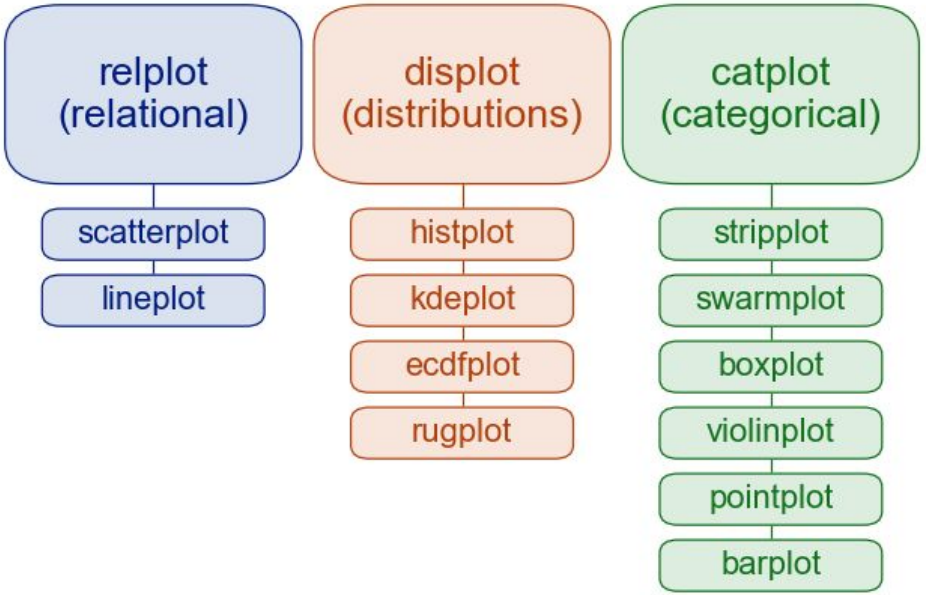


Figure 4: hierarchy of plotting functions
https://seaborn.pydata.org/tutorial/function_overview.html

Facet Grid

- Figure functions return a FacetGrid object, which maps a dataset onto multiple axes arrayed in a grid of rows and columns
  - the resulting object is like a grid of plots, with row*columns total plot
  - useful when you want to visualize the distribution of a variable or the relationship between multiple variables separately within subsets of your dataset
- Having rows and columns allows us to make multiple plots that change which part of the dataset its looking at

```
import seaborn as sns
tips = sns.load_dataset("tips")
tips.head()
sns.relplot(data=tips, x="total_bill", y="tip",
col="time")
```

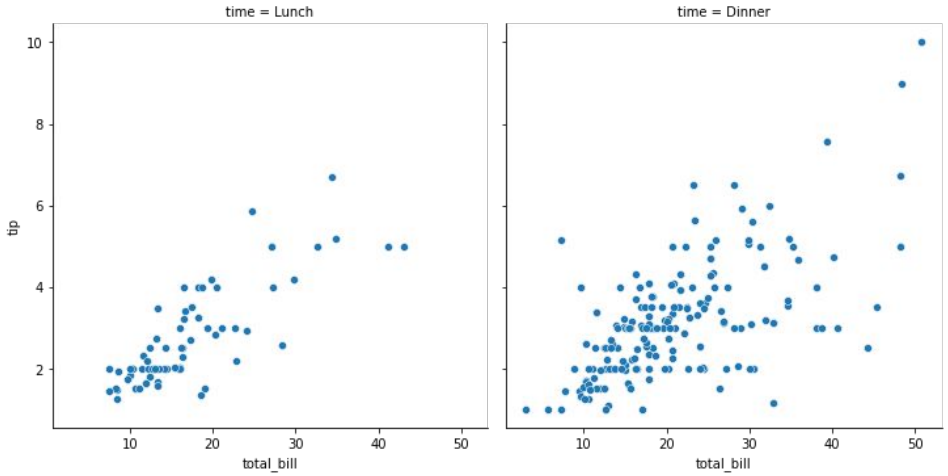


FIgure 6: changing the column parameter

Facet Grid

- Figure functions return a FacetGrid object, which maps a dataset onto multiple axes arrayed in a grid of rows and columns
  - the resulting object is like a grid of plots, with row*columns total plot
  - useful when you want to visualize the distribution of a variable or the relationship between multiple variables separately within subsets of your dataset
- Having rows and columns allows us to make multiple plots that change which part of the dataset its looking at

```
import seaborn as sns
tips = sns.load_dataset("tips")
tips.head()
sns.relplot(data=tips, x="total_bill", y="tip",
hue="day")
```

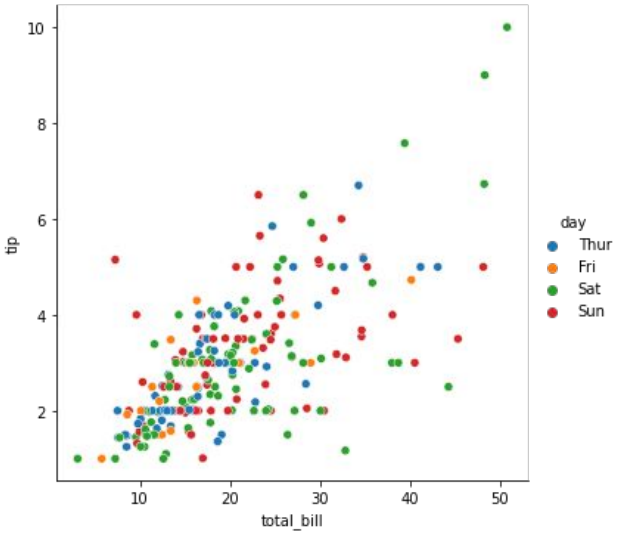


Figure 7: changing the hue parameter

Barplots

```
import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> ax = sns.barplot(x="day", y="total_bill",
data=tips)
```

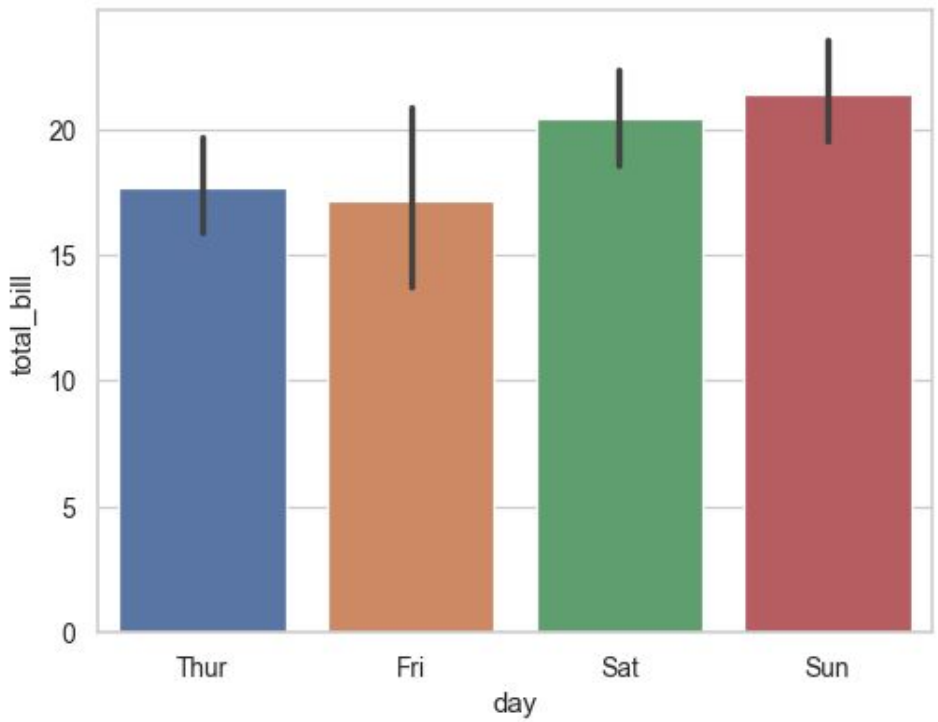


Figure 8: barplots in seaborn
https://seaborn.pydata.org/generated/seaborn.barplot.html

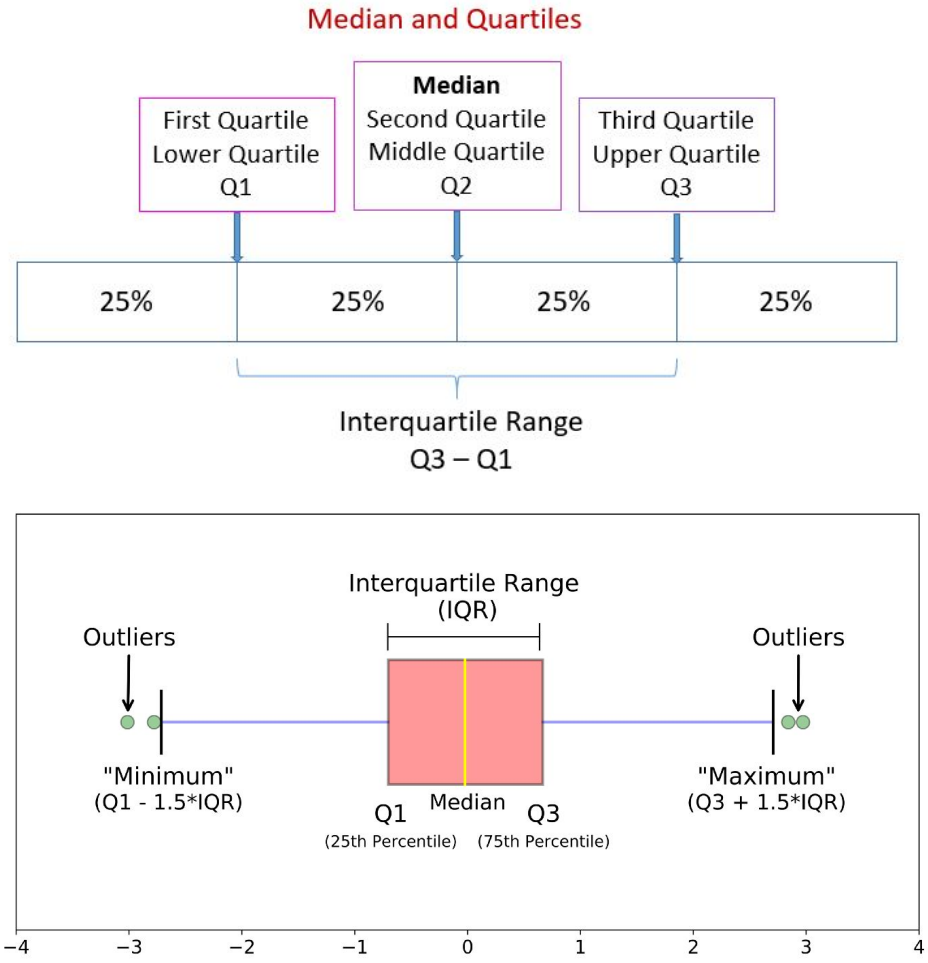
Box Plots

Quartiles

- Quartiles, as the name suggests, split our data up into four equally sized groups (each group contains the same number of datapoints).

Boxplots

- capture the essence of the data's spread, by transforming it in a "box"
  - top line of the box represents the upper quartile,
  - bottom line of the box represents the lower quartile
  - the line between them is the median



Box Plots in Seaborn

```
import seaborn as sns
>>> tips = sns.load_dataset("tips")
>>> ax = sns.boxplot(x=tips["total_bill"])
```

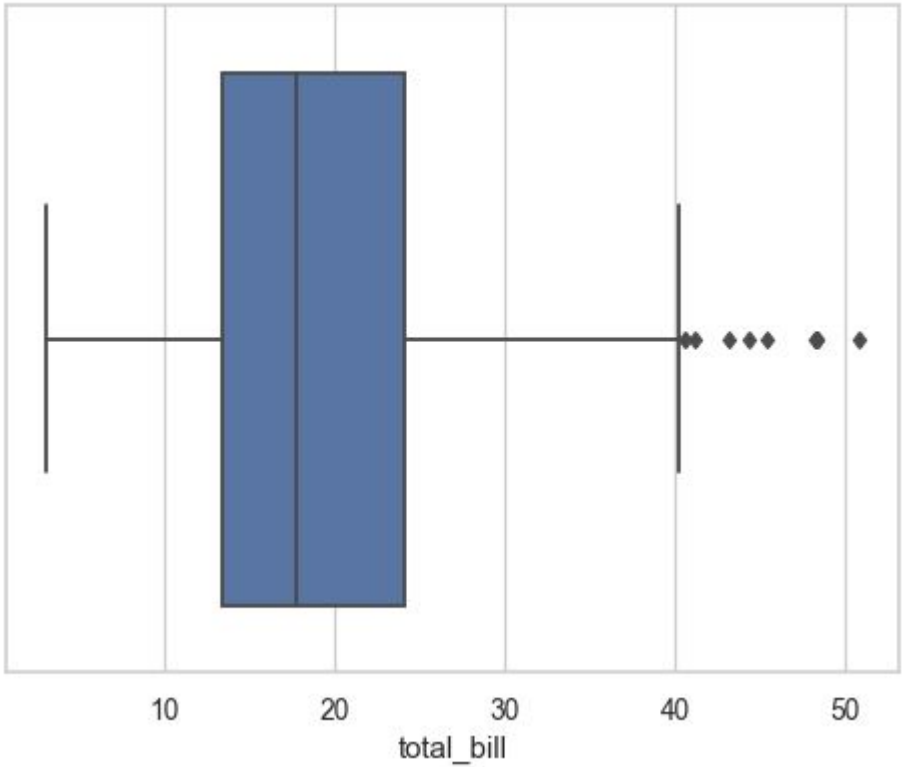


Figure 11: boxplot exmample
https://seaborn.pydata.org/generated/seaborn.boxplot.html

Side by side boxplots in Seaborn

Side by side boxplots

- visual display comparing the levels (the possible values) of one categorical variable by means of a quantitative variable.

```
ax = sns.boxplot(x="day", y="total_bill",
hue="smoker", data=tips, palette="Set3")
```

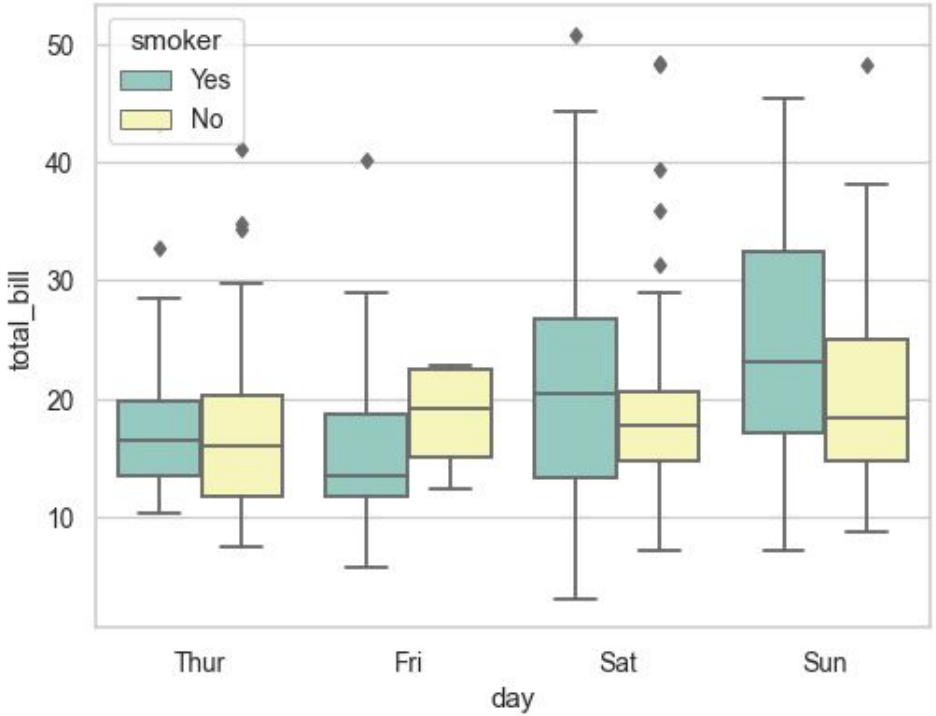


Figure 12: side by side boxplot example
https://seaborn.pydata.org/generated/seaborn.boxplot.html

Scatterplots

- visualize the relationship between two quantitative variables, with each variable being represented along one axis

```
tips = sns.load_dataset("tips")

sns.scatterplot(data=tips, x="total_bill", y="tip")
```

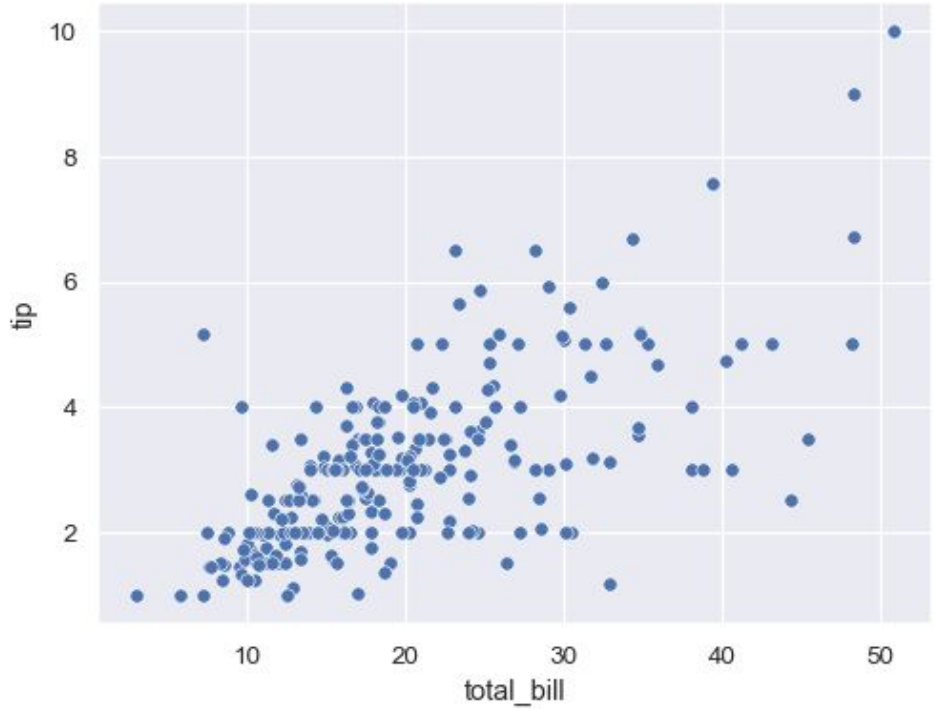


Figure 13: scatter plot example

Histograms

- allows us to visualize the distribution of numerical data
- first take the range of values of our data and divide it into discrete "bins" or intervals
- Then count the number of times our data falls into each interval
    - bins should be consecutive and non overlapping, and the same size

```
penguins = sns.load_dataset("penguins")

sns.displot(penguins, x="flipper_length_mm")
```

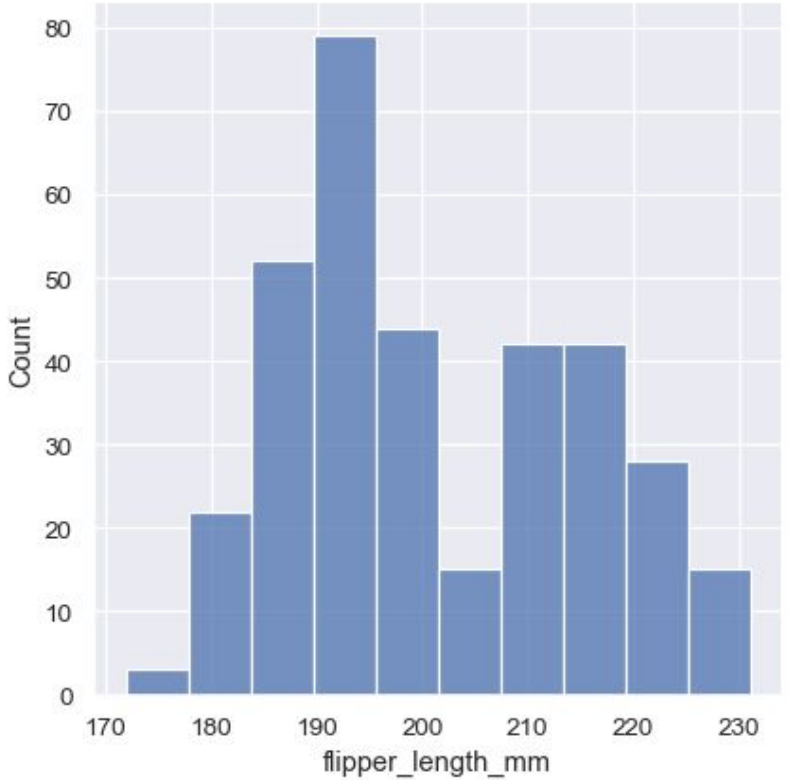


Figure 14: histogram example

Density curves

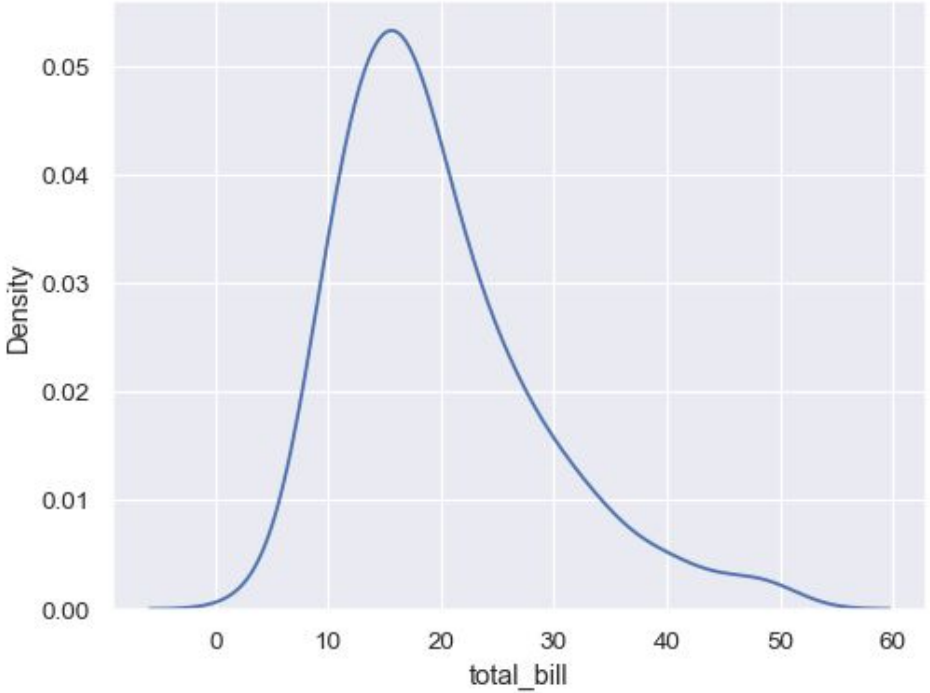


Figure 15: KDE curve example
https://seaborn.pydata.org/generated/seaborn.kdeplot.html#seaborn.kdeplot

- Visualizes distribution just like histograms
- Rather than using discrete bins, a KDE plot smooths the observations with a Gaussian kernel, producing a continuous density estimate

```
tips = sns.load_dataset("tips")

sns.kdeplot(data=tips, x="total_bill")
```

Joint plots

- Visualizes both a bivariate and univariate graph at the same time

```
sns.jointplot(data=df, x='', y='', kind = '',
hue='')
```

- The default behavior (i.e. without supplying the kind variable) will return a scatterplot for the relationship and a histogram for the univariate distributions of x and y.

```
penguins = sns.load_dataset("penguins")
sns.jointplot(data=penguins,
x="bill_length_mm", y="bill_depth_mm")
```

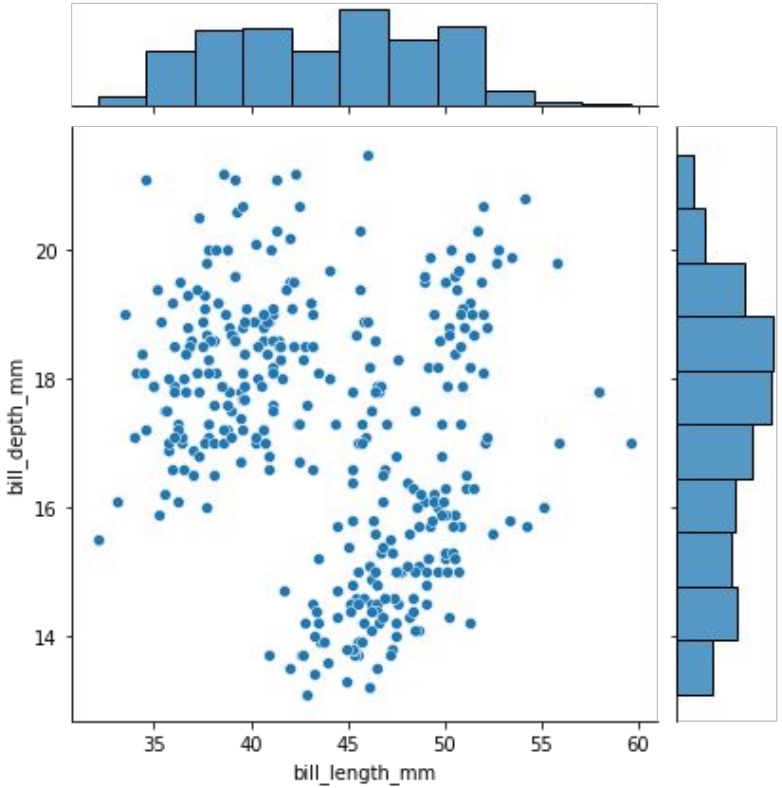
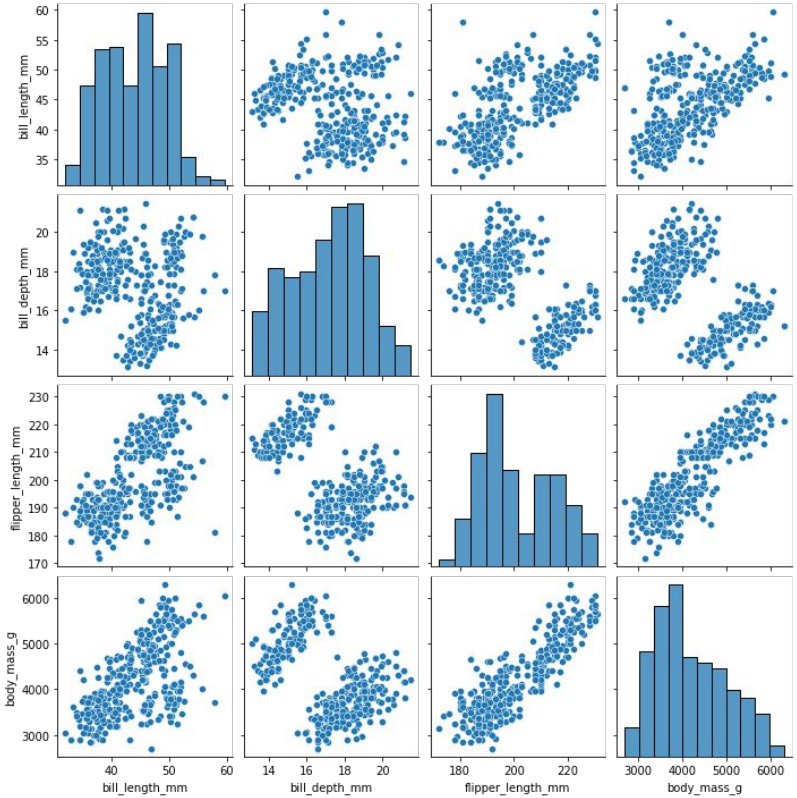


Figure 16: joinplot example

Pair plots



- We might want to visualize many variables and their relationships in an effort to get an understanding of many different relationships in our data
- The pair plot visualizes all pairwise combination of variables for a dataframe

```
penguins = sns.load_dataset("penguins")
sns.pairplot(penguins)
```

Figure 17: pairplot example

Text and Annotations

- Matplotlib provides several text functions that help us do this. When given an Axes ax and a Figure fig, we can perform the following functions.

Add string text_to_add at an arbitrary position (x,y) on the Axes using the Axes coordinate system.

```
ax.text(x, y, text_to_add)
```

Draw an arrow from xytext to xy on an Axes object that has the string text drawn at xytest.

```
ax.annotate(text=text, xy=(x, y), xytext=(text_x, text_y),


arrowprops=dict(arrowstyle="->",connectionstyle="arc3"))
```

Set the title as label for an Axes object.

```
ax.set_title(label)
```

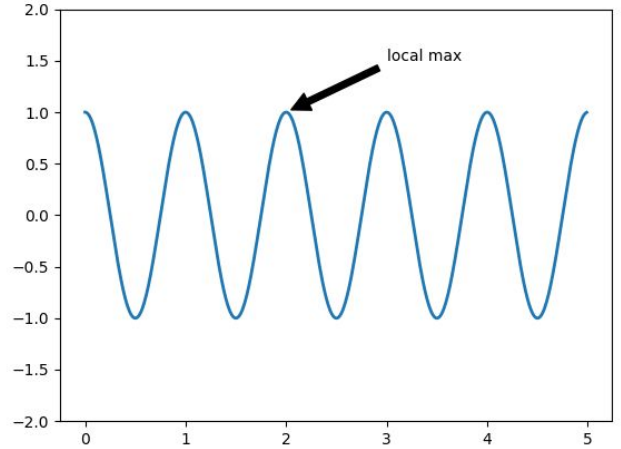


Figure 18: example of an annotation https://matplotlib.org/tutorials/text/annotations.html

Text and Annotations

Set the label for the y axis of a given Axes object as ylabel

```
ax.set_ylabel(ylabel)
```

Set the label for the x axis of a given Axes object as xlabel

```
ax.set_xlabel(xlabel)
```

Set the title for the entire figure as fig_title.

```
fig.suptitle(fig_title)
```

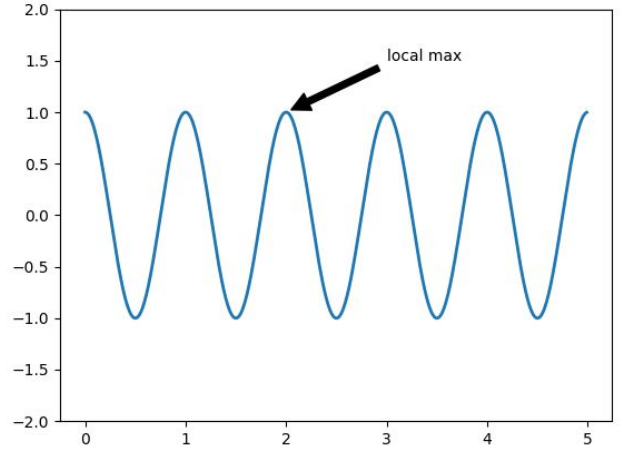


Figure 18: example of an annotation https://matplotlib.org/tutorials/text/annotations.html

Working with images

- Images are an important tool for visualization, we can bring images (in png format) into our code using the matplotlib image library
- The library provides tools for loading, rescaling, and displaying the image within our code environments

```
  ...
  [0.         0.29803923 0.10980392 1.        ]
  [0.         0.3647059  0.18039216 1.        ]
  [0.         0.34117648 0.16078432 1.        ]]

 [[0.         0.40784314 0.1764706  1.        ]
  [0.         0.4        0.17254902 1.        ]
  [0.         0.39607844 0.18039216 1.        ]
  ...
  [0.         0.32941177 0.14117648 1.        ]
  [0.         0.3647059  0.18039216 1.        ]
  [0.         0.3137255  0.12941177 1.        ]]]
<matplotlib.image.AxesImage at 0x7f8e44c3c400>
```

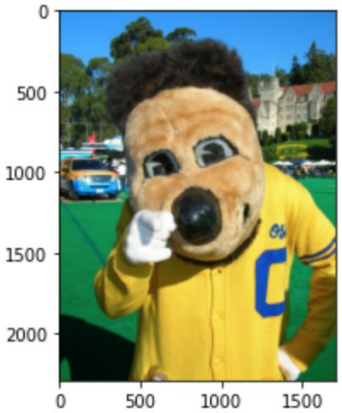


Figure 19: matplotlib image example

Working with images

```
import matplotlib.image as mpimg

# Importing a local image

img = mpimg.imread('oski_game.png')

'''

matplot allows us to see the image file in its raw form by printing:

'''

print(img)

'''

matplot also allows us to view the image by calling imshow from pyplot.

'''

plt.imshow(img)
```