

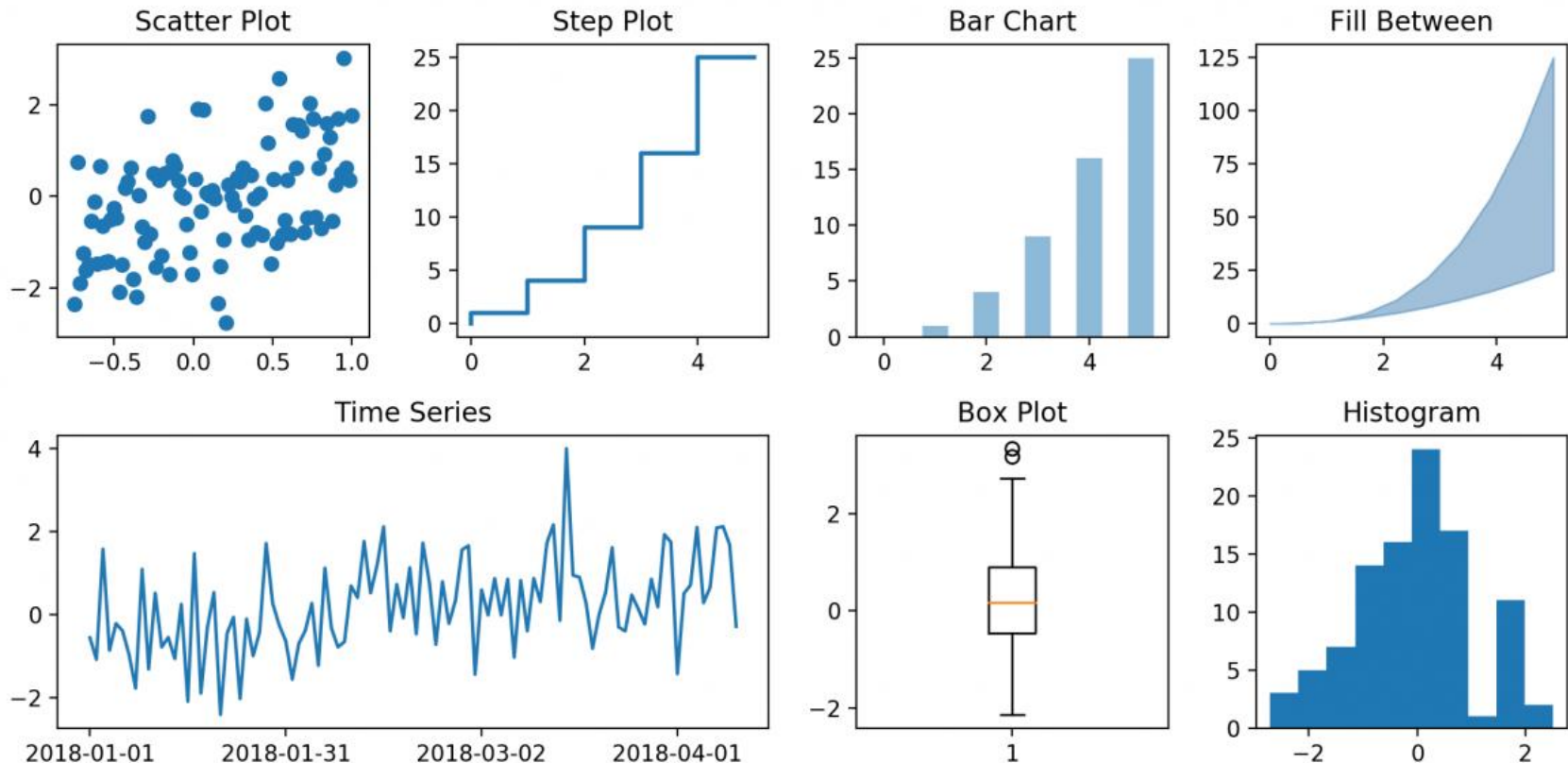
(U4284) Python程式設計 Package Intro Visualization Tool

Speaker: 吳淳硯



What is Data Visualization?

- It is the process of presenting data in a visual format, such as a chart, graph, or map. It helps users identify patterns and trends in a data set, making it easier to understand complex information.
- Visualizations can be used to analyze data, make predictions, and even communicate ideas more effectively.



Six Principle

- Three Types of Communication Problems.
 - ▶ *The technical problem*: How accurately can the symbols of communication be transmitted?
 - ▶ *The semantic problem*: How precisely do the transmitted symbols convey the desired meaning?
 - ▶ *The effectiveness problem*: How effectively does the received meaning affect conduct in the desired way?
- Six principle to consider when communicating data.
 1. Know your goal
 2. Use the right data
 3. Select suitable visualizations
 4. Design for aesthetics
 5. Choose an effective medium and channel
 6. Check at results

Principle 1 - Know Your Goal

- Information and Message are not synonymous.
 - Information: is the set of all possible messages that can be selected by the information source.
 - Message: is what was selected from this set to be communicated.
- Elements of the goal
 - **Who** are you trying to communicate with?
 - **What** do you want them to know?
 - **Why**? What do you want them to do about it?

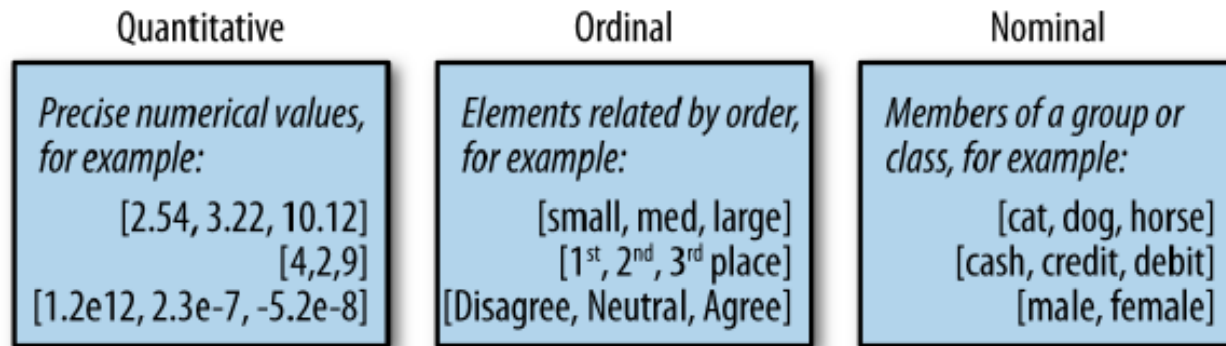


Principle 2 – Use the Right Data

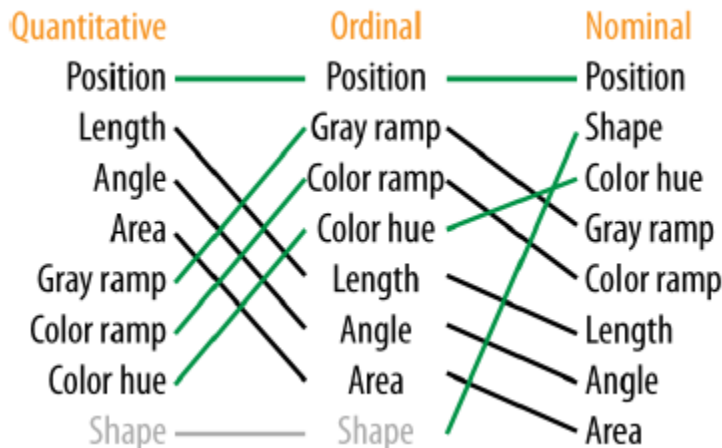
- Why it matters ?
 - If you overcrowd the capacity of the audience, you force a general and inescapable error and confusion.
- Message must be both ethical and based on sound epistemology. Don't lie with statistics.
- Don't fall prey to the many and various forms of statistical and logical fallacies.
 - Causation \neq Correlation
 - Unreasonable inductive leaps
 - Misused Statistics.
 - Overstated conclusions from week samples.

Principle 3 – Select Suitable Visualizations

- How to encode the message?
 - converting the data values themselves into abstract graphical representations, like size or color or shape
- What type of data do you have?



- What are the most effective type of visualization for your data type?

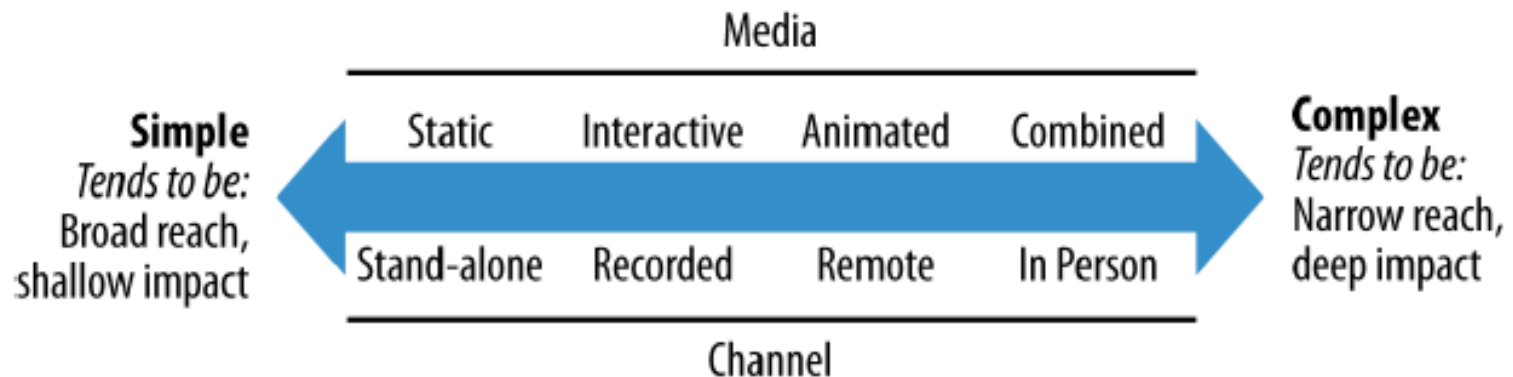


Principle 4 - Design for Aesthetics

- There is a clear value to elegant design and what Willard Cope Brinton called Judicious embellishment of charts.
- There are a number of aesthetic elements, and a handful of common mistakes people make when creating them:
 - Poor color schemes
 - Distracting fonts
 - Many different fonts
 - Sloppy alignment
 - Vertical or angled labels
 - Dark background colors
 - Thick borders or grid lines
 - Useless images and clip art
 - Lazily accepting most software defaults

Principle 5 - Choose an Effective Medium and Channel

- What form the message takes (**medium**) and how it gets delivered to the audience (**channel**) are critical elements of data communication effort.
- When you communicate data, there are a few choices to make about how you will do it:
 - Standalone graphics or narrated?
 - Static, interactive, animated, or combined graphics?
 - If narrated: recorded, live, or both?
 - If live: remote, in person, or both?
 - In all cases: broadcast, directed, or both?



Principle 6 - Check the Results

- It is a good habit in general to incorporate into your efforts feedback loops and checkpoints that help you gauge whether you've achieved your intended results or not.
- There are a few questions to ask when you check the results. We'll call this the **RUI**
 - *Reach*:

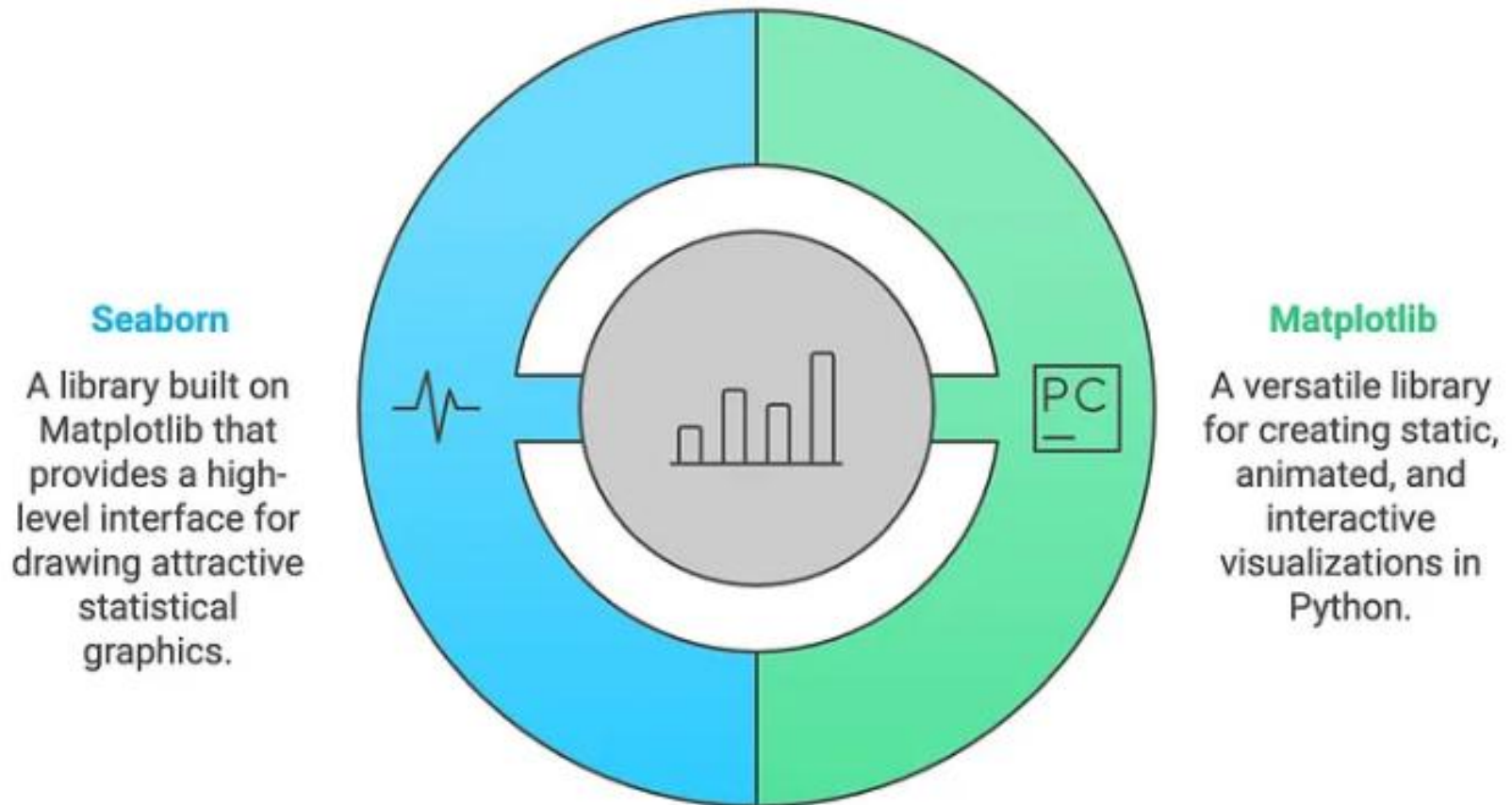
Did the audience even receive your message at all? Who did and who didn't?
 - *Understanding*:

Did the audience interpret the data message in the way you intended?
 - *Impact*:

Did the audience react in the way you wanted them to react?

Graphical Tools in Python

- Python offers several plotting libraries, namely **Matplotlib**, **Seaborn** and many other such data visualization packages with different features for creating informative, customized, and appealing plots to present data in the most simple and effective way.





Parts of a Figure

- The Figure keeps track of all the child **Axes**, a group of special Artists (titles, figure legends, colorbars, etc.), and even nested subfigures.

```
fig = plt.figure()           # an empty figure with no Axes
fig, ax = plt.subplots()     # a figure with a single Axes
fig, axs = plt.subplots(2, 2) # a figure with a 2x2 grid of Axes
# a figure with one Axes on the left, and two on the right:
fig, axs = plt.subplot_mosaic([['left', 'right_top'],
                                ['left', 'right_bottom']])
```

- An Axes is an Artist attached to a Figure that contains a region for plotting data, and usually includes Axis objects (be aware of the difference between **Axes** and **Axis**) that provide ticks and tick labels to provide scales for the data in the Axes.
- Plotting functions expect **numpy.array** or **numpy.ma.masked_array** as input, or objects that can be passed to **numpy.asarray**. Classes that are similar to arrays (“array-like”) such as **pandas** data objects and **numpy.matrix** may not work as intended.

Explicit and Implicit Interfaces

- MATLAB-style Interface (PyPlot-style)

```
plt.figure() # create a plot figure

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```

- Object-oriented Interface (OO-style)

```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```

Matplotlib Gotchas

- While most **plt** functions translate directly to **ax** methods
 `plt.plot` → `ax.plot`
 `plt.legend` → `ax.legend`
 this is not the case for all commands.
- In particular, functions to set limits, labels, and titles are slightly modified.
 `plt.xlabel` → `ax.set_xlabel`
 `plt.ylabel` → `ax.set_ylabel`
 `plt.xlim` → `ax.set_xlim`
 `plt.ylim` → `ax.set_ylim`
 `plt.title` → `ax.set_title`
- In the on Object-oriented interface to plotting, rather than calling these individually, it is often more convenient to use the **ax.set** method to set all these properties at once.

Color Platte

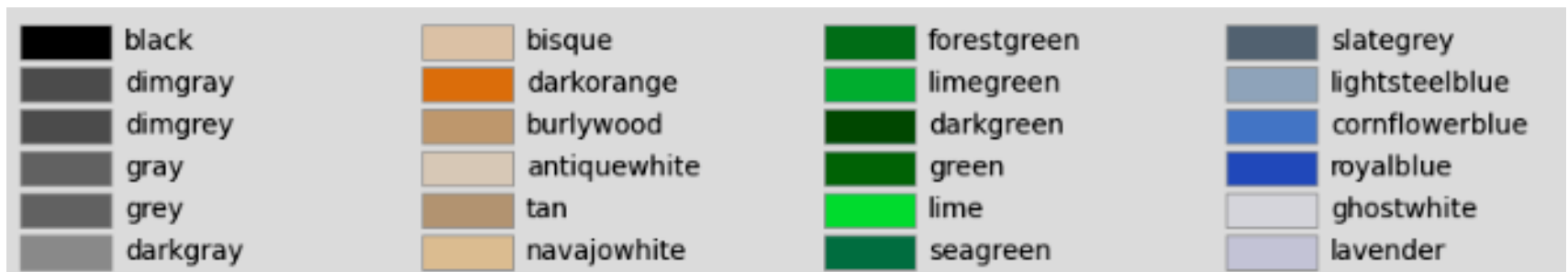
- Base Colors



- Tableau Palette







- CSS Colors (Not list all)












- XKCD Colors

Line Styles

Named linestyles

solid 'solid'	
dotted 'dotted'	
dashed 'dashed'	
dashdot 'dashdot'	

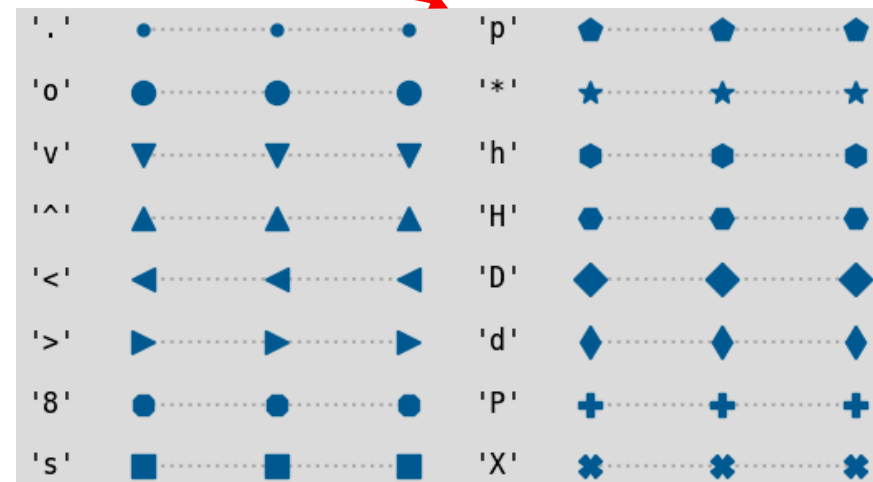
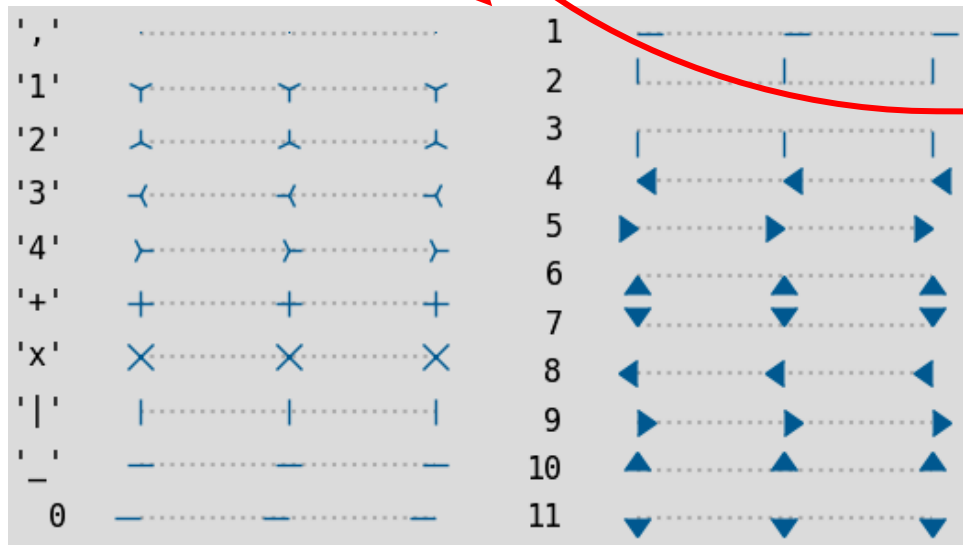
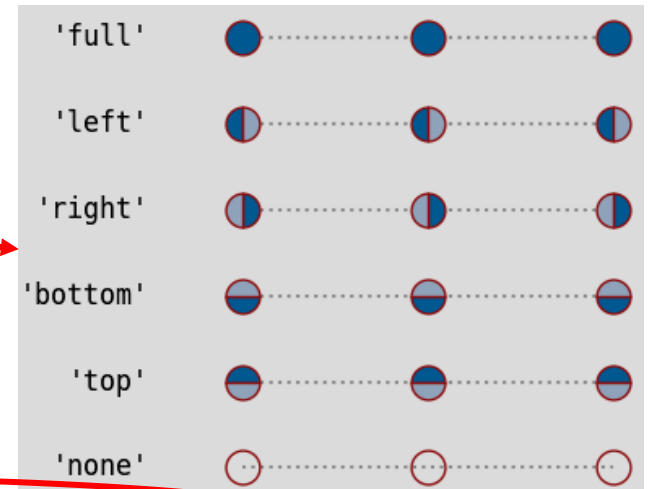
Parametrized linestyles

loosely dotted (0, (1, 10))	
dotted (0, (1, 5))	
densely dotted (0, (1, 1))	
long dash with offset (5, (10, 3))	
loosely dashed (0, (5, 10))	
dashed (0, (5, 5))	
densely dashed (0, (5, 1))	
loosely dashdotted (0, (3, 10, 1, 10))	
dashdotted (0, (3, 5, 1, 5))	
densely dashdotted (0, (3, 1, 1, 1))	
dashdotdotted (0, (3, 5, 1, 5, 1, 5))	
loosely dashdotdotted (0, (3, 10, 1, 10, 1, 10))	
densely dashdotdotted (0, (3, 1, 1, 1, 1, 1))	

Marker reference

- Matplotlib supports multiple categories of marker which are selected using the marker parameter of plot commands:

- Unfilled markers (Single-colored)
- Filled markers (Marker fill styles)
- Markers create from TeX symbols
- Markers created from Paths.



Marker Reference

Mathtext markers

'\$1\$'	1	1	1
'\$\frac{1}{2}\$'	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
'\$f\$'	<i>f</i>	<i>f</i>	<i>f</i>
'\$♪\$'	♪	♪	♪
'\$\mathcal{A}\$'	\mathcal{A}	\mathcal{A}	\mathcal{A}

Path markers

star	★	★	★
circle	●	●	●
cut_star	☆	☆	☆

- Markers can be modified by passing a transform to the MarkerStyle constructor. Ex. Rotation.

Equation marker	$\frac{1}{x}$	$\frac{1}{x}$	$\frac{1}{x}$	$\frac{1}{x}$	$\frac{1}{x}$	$\frac{1}{x}$	$\frac{1}{x}$
	0°	10°	20°	30°	45°	60°	90°

plt.scatter vs plt.plot with o

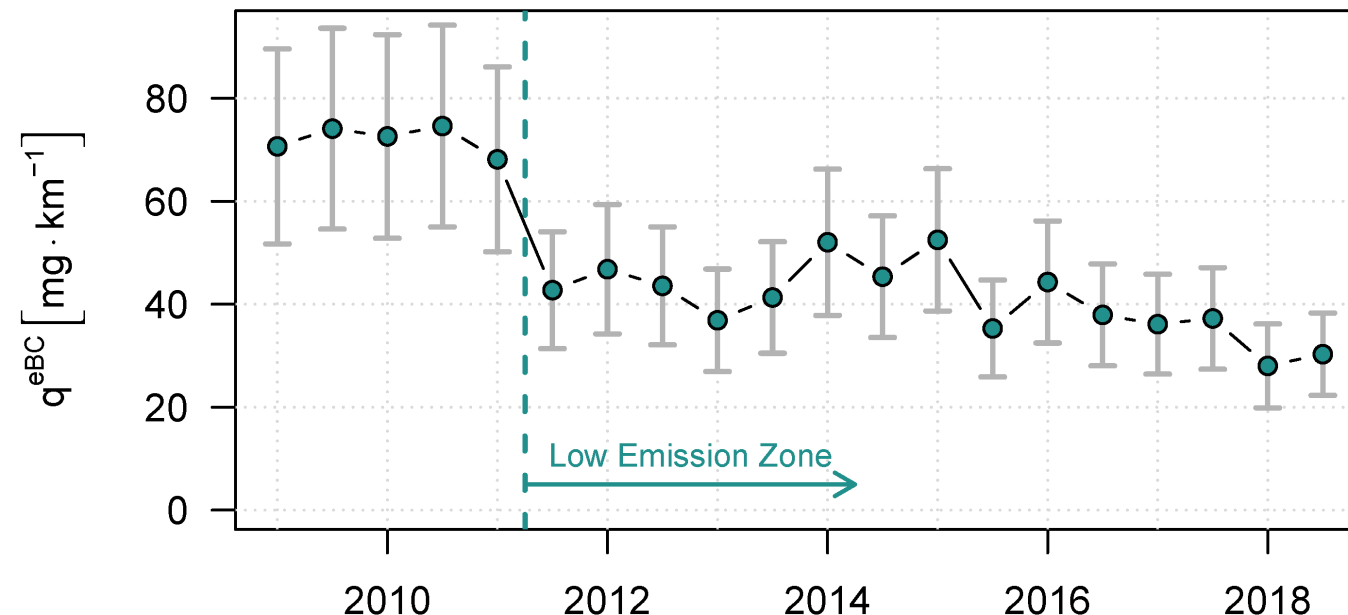
- In the previous chapter we looked at using plt.plot/ax.plot to produce line plots. It turns out that this same function can produce scatter plots as well.

```
rng = np.random.default_rng(0)
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.random(2), rng.random(2), marker, color='black',
             label="marker='{0}'".format(marker))
plt.legend(numpoints=1, fontsize=13)
plt.xlim(0, 1.8);
```

- A Note on Efficiency
 - While it doesn't matter as much for small amounts of data, as datasets get larger than a few thousand points, **plt.plot can be noticeably more efficient than plt.scatter.**
 - The reason is that plt.scatter has the capability to render a different size and/or color for each point, so the renderer must do the extra work of constructing each point individually.

Visualizing Uncertainties

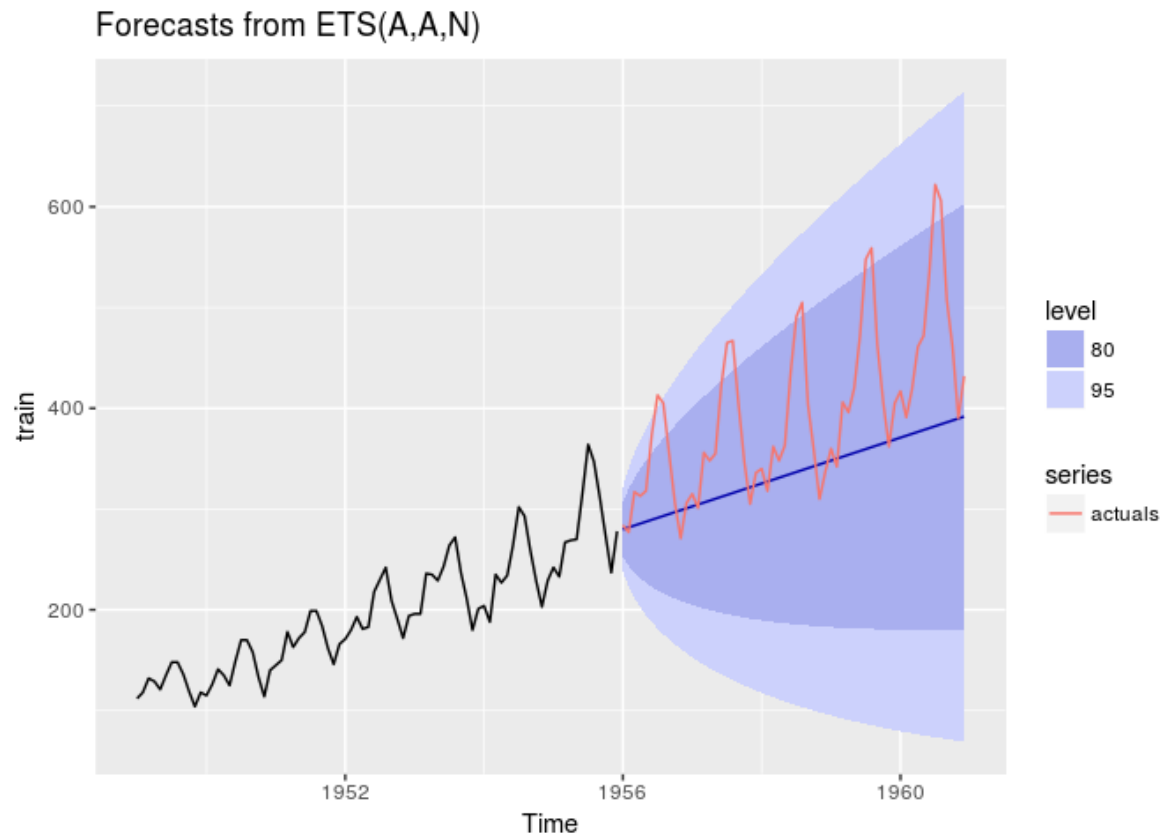
- In visualization of data and results, showing their errors effectively can make a plot convey much more complete information.
- eBC emission factor for the station EIBA as summer/winter averages between 2009 and 2018 from 6-month scatter plots. Error bars indicate 95% confidence interval of the fit plus 25% measurement uncertainty.



Year	q^{eBC} [$\text{mg} \cdot \text{km}^{-1}$]
2009	73 ± 20
2010	73 ± 20
2011	51 ± 14
2012	42 ± 12
2013	41 ± 11
2014	48 ± 13
2015	43 ± 12
2016	38 ± 10
2017	37 ± 10
2018	31 ± 9

Continuous Error

- Continuous error bands are a graphical representation of error or uncertainty as a shaded region around a main trace, rather than as discrete whisker-like error bars.
 - Prediction interval from a model
 - Confidence interval around a mean line



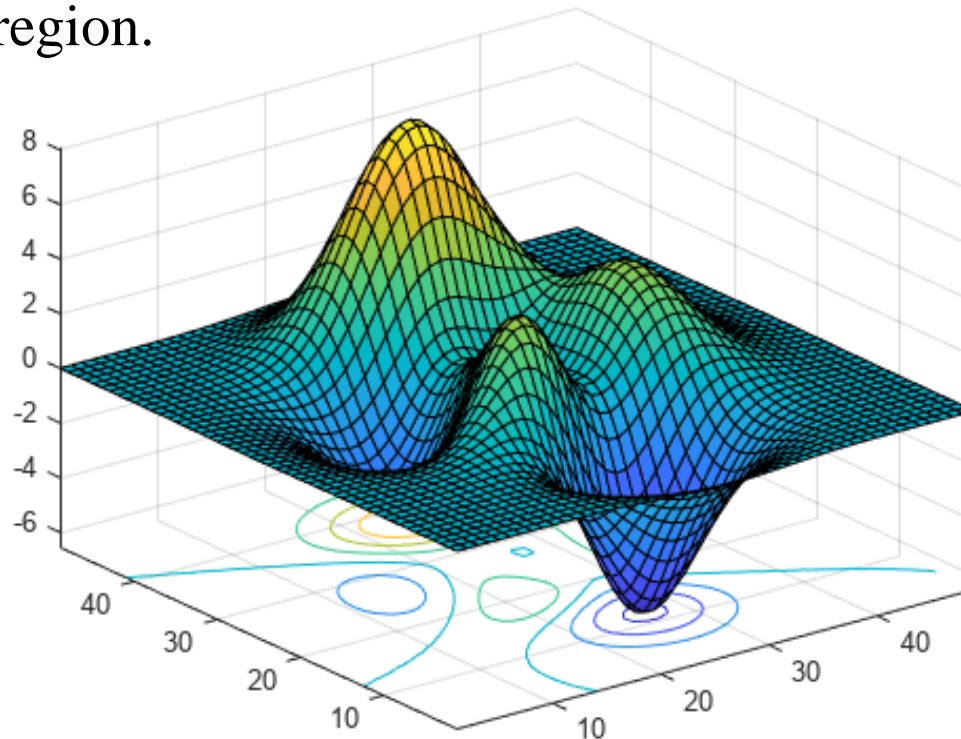
Visual 3D

- A level set of a real valued function f on n real variables is a set where the function takes on a given constant value c .

$$\mathcal{L}_c(f) = \{(x_1, \dots, x_n) | f(x_1, \dots, x_n) = c\}$$

When $n = 2$, is called level curve, also known as contour line.

- Sometimes it is useful to display 3D data in 2D using contours or color-coded region.

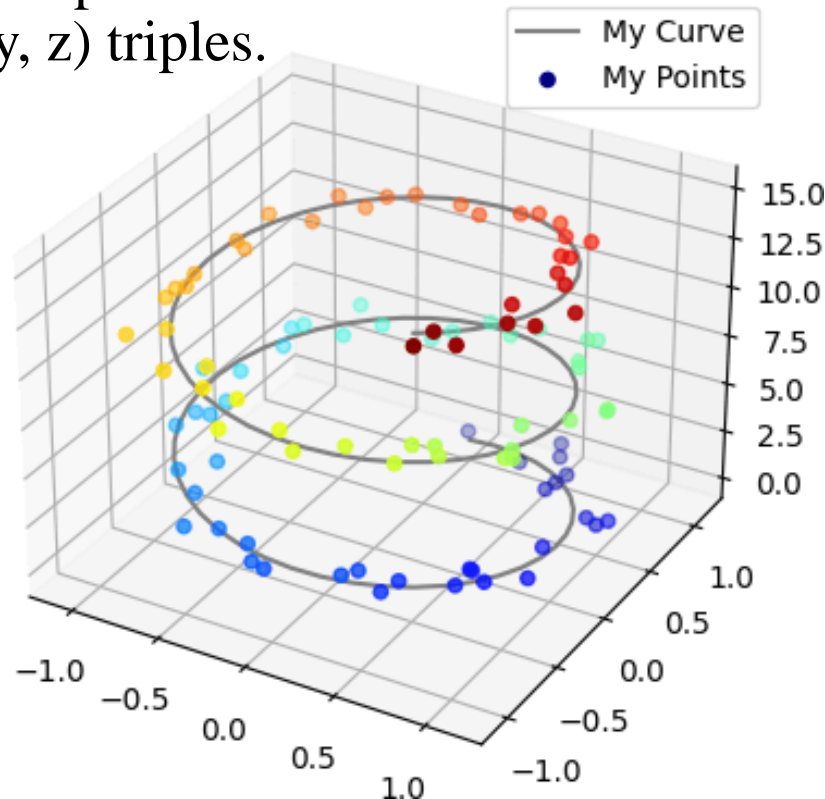


Colormaps

- Colormaps are often split into several categories based on their function.
 - Sequential (data has **ordering**)
Change in lightness and often saturation of color incrementally.
 - Diverging (data **deviates around zero**)
Change in lightness and possibly saturation of two different colors that meet in the middle at an unsaturated color.
 - Cyclic (data **wrap around at the endpoints**)
Change in lightness of two different colors that meet in the middle and beginning/end at an unsaturated color
 - Qualitative (data has no ordering or relationship)
often are miscellaneous colors
- Some of the miscellaneous colormaps have particular uses for which they have been created. Ex. gist_earth, ocean, and terrain all seem to be created for plotting topography.

3D Plot

- Matplotlib was initially designed with only 2D plotting in mind. 3D plots are enabled by importing the `mplot3d` toolkit, included with the main Matplotlib installation. Once this module is imported, a 3D axes can be created by passing the keyword `projection='3d'` to any of the normal axes creation routines.
- The most basic 3D plot is a line or collection of scatter plots created from sets of (x, y, z) triples.



Seaborn

- The main idea of Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration, and even some statistical model fitting.

```
In [11]: sns.jointplot(x="total_bill", y="tip", data=tips, kind='reg');
```

