KAZAKH-BRITISH TECHNICAL UNIVERSITY
KBTU
School of IT and Engineering

MACHINE LEARNING

LECTURE #2

Number of credits: 3 (2/0/1)
Course code – CSCI3234
MS Teams team code – **y4ntwlz**

Adilet Yerkin, MS in Engineering, Senior Lecturer
a.yerkin@kbtu.kz

INTRODUCTION

"Love is the power to see similarity in the dissimilar."
Theodor Adorno

Similarity

Many data mining applications require the determination of similar or dissimilar objects, patterns, attributes, and events in the data. In other words, a methodical way of quantifying similarity between data objects is required. Virtually all data mining problems, such as clustering, outlier detection, and classification, require the computation of similarity.

A formal statement of the problem of similarity or distance quantification is as follows:

Given two objects **O1** and **O2**, determine a value of the similarity **Sim(O1, O2)**
(or distance **Dist(O1, O2)**) between the two objects.

Similarity is used in tasks like nearest-neighbor search and recommender systems.
Distance is used in clustering and anomaly detection.
In practice, a "similarity" can always be converted to a "distance" and vice versa.

!!! (depending on the domain at hand. But in some domains, cannot be expressed in closed form)

QUANTITATIVE DATA

The most common distance function for quantitative data is the L_p-norm

between two data points $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$

is defined as follows:

$$Dist(\overline{X}, \overline{Y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

p = 1 (Manhattan or L1 distance)

$$D_1(x, y) = \sum |x_i - y_i|$$

p = 2 (Euclidean or L2 distance)

$$D_2(x, y) = \sqrt{\sum (x_i - y_i)^2}$$

Minkowski distance

Two special cases of the Lp-norm are the Euclidean (p = 2) and the Manhattan (p = 1) metrics.

1. The Euclidean distance is the straight-line distance between two data points.
2. The Manhattan distance is the “city block” driving distance in a region in which the streets are arranged as a rectangular grid, such as the Manhattan Island of New York City.

DISTANCES FOR BINARY DATA

Simple Matching Coefficient (SMC) The Sokal-Michener distance

is a similarity measure used to compare two binary (0/1) vectors. It considers the proportion of matching components (both 0s and 1s) between the vectors.

$$\mathrm{SMC} = \frac{M_{11} + M_{00}}{M_{11} + M_{10} + M_{01} + M_{00}}$$

		A	
		0	1
B	0	M_{00}	M_{10}
	1	M_{01}	M_{11}

- M_{00} is the total number of attributes where *A* and *B* both have a value of 0,
- M_{11} is the total number of attributes where *A* and *B* both have a value of 1,
- M_{01} is the total number of attributes where *A* has value 0 and *B* has value 1, and
- M_{10} is the total number of attributes where *A* has value 1 and *B* has value 0.

Single ordinal attribute

{poor, fair, OK, good, wonderful}

This definition of similarity (dissimilarity) for an ordinal attribute should make the reader a bit uneasy since this assumes equal intervals between successive values of the attribute, and this is not necessarily so. Otherwise, we would have an interval or ratio attribute. Is the difference between the values fair and good really the same as that between the values OK and wonderful?

Probably not, but in practice, our options are limited, and in the absence of more information, this is the standard approach for defining proximity between ordinal attributes.

Table 2.7. Similarity and dissimilarity for simple attributes

Attribute Type	Dissimilarity	Similarity
Nominal	d={ 0if x=y1if x≠y	s={ 1if x=y0if x≠y
Ordinal	d=|x−y|/(n−1) (values mapped to integers 0 to n−1, where *n* is the number of values)	s=1−d
Interval or Ratio	d=|x−y|	s=−d, s=11+d, s=e−d,s=1−d−min_dmax_d−min_d

JACCARD COEFFICIENT

Jaccard similarity

Sometimes we care only about where both objects have 1s (presence) and we want to ignore joint 0s (absence).

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

For binary vectors:

$$J = \frac{M_{11}}{M_{11} + M_{10} + M_{01}}$$

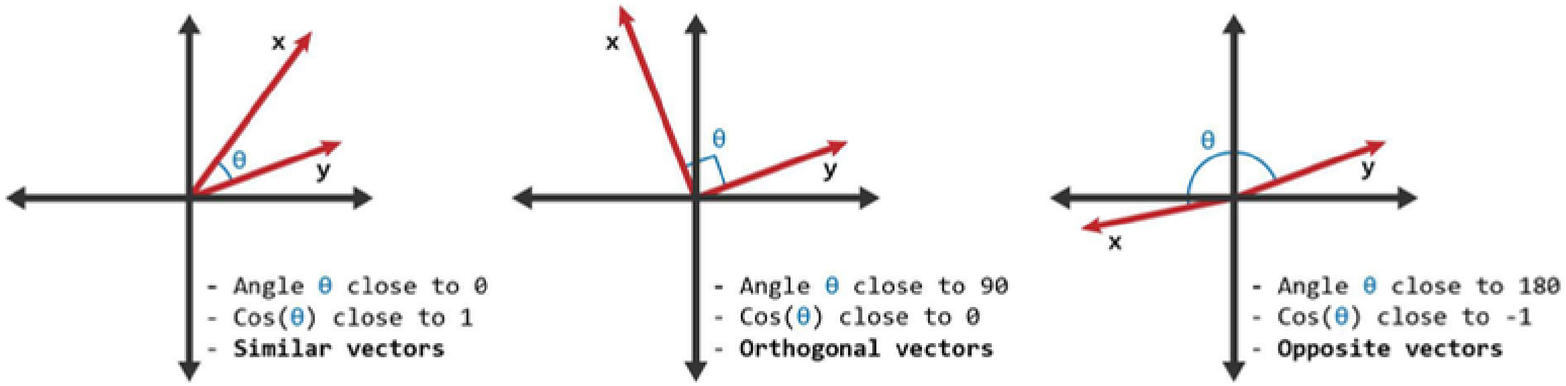
User A likes: {Inception, Interstellar, Dune}
User B likes: {Dune, Blade Runner, Arrival}
Intersection = {Dune} → 1
Union = {Inception, Interstellar, Dune, Blade Runner, Arrival} → 5

Jaccard = 1 / 5 = 0.2

COSINE SIMILARITY

Text data are often very high-dimensional and sparse: most words do not appear in a single document. The absolute length of a document is less important than the angle between its word-frequency vector and another's.

$$\text{cosine}(x, y) = \frac{x \cdot y}{\|x\|\|y\|}$$



If two vectors point in the same direction, the angle between them is small → high similarity.

If they are at 90° (orthogonal), they share no similarity → 0.

If they point in opposite directions, similarity is negative.

CORRELATION

When we explore datasets, a common question is:

Do these two variables move together?

For example:

1. Does increasing advertising budget increase sales?
2. Is there a relationship between exercise time and cholesterol levels?

The statistical concept that answers these questions is correlation.

It tells us:
Direction – do the variables move in the same or opposite direction?
Strength – how tightly do the data points follow a straight-line pattern?

Remember: Correlation is about relationship, not cause-and-effect.

CORRELATION

Types of Correlation

Type	Description
Positive	As X increases, Y also increases.
Negative	As X increases, Y decreases.
Zero (no correlation)	No predictable relationship.

CORRELATION VS. CAUSATION

It is critical to emphasize:
Correlation does NOT imply causation.

Example: The number of people who drown in swimming pools and the number of Nicolas Cage movies released in a year may be positively correlated, but one does not cause the other.

Such relationships can be explained by a **lurking variable** (e.g., season, economic factors).

PEARSON'S CORRELATION COEFFICIENT (R)

The most widely used measure is Pearson's r, which captures linear relationships between two continuous variables.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

SPEARMAN'S RANK CORRELATION

Sometimes the relationship between two variables is monotonic (always increasing or decreasing) but not strictly linear.

Spearman's Rank Correlation Coefficient, often denoted as ρ (rho) or r_s, measures the strength and direction of a monotonic relationship between two variables.

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

n = number of observations

d_i = difference between the ranks of each pair

Detects monotonic but nonlinear relationships

MUTUAL INFORMATION

Mutual Information (MI) captures any dependency between two variables.

Mutual Information measures how much knowing one variable reduces uncertainty about another.

$$I(X;Y) = \sum_{x} \sum_{y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

Feature Selection - Choose features with highest MI with the target (captures nonlinear relationships that Pearson correlation may miss).

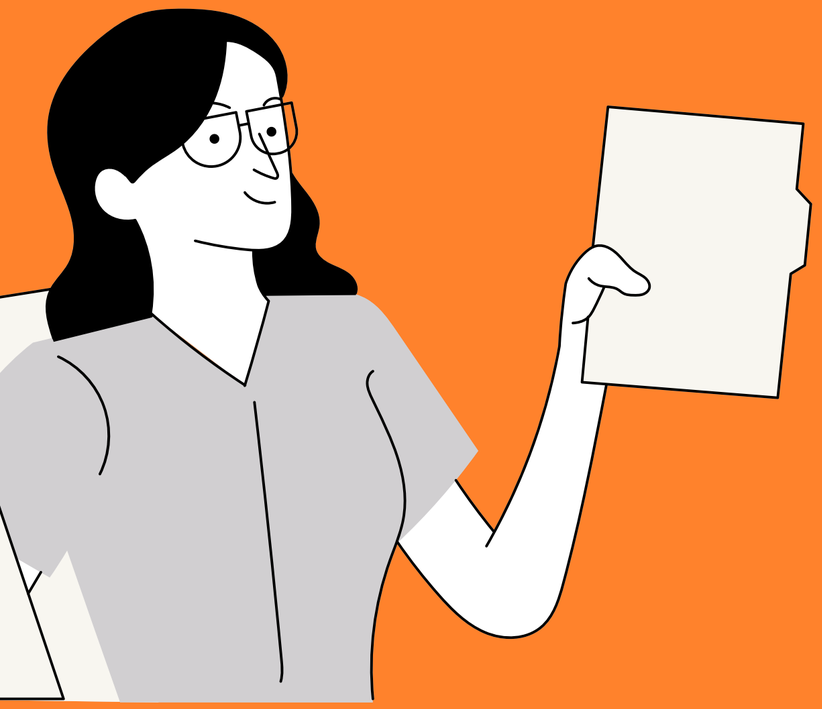
Dimensionality reduction

Reduce the dimensionality of a dataset while retaining as much variance (information) as possible.

Summarize many correlated variables with a smaller set of new variables that still capture most of the original information.

Principal component analysis (PCA)

PCA



Purpose: Reduce the dimensionality of a dataset while retaining as much variance (information) as possible.

Key idea: Find new orthogonal axes (principal components) along directions of maximum variance.

Benefits:

- Simplifies models → less overfitting.
- Speeds up computation.
- Reveals hidden structure.

PCA

Think about taking a photo of a 3-D object:

The camera projects 3-D reality onto a 2-D picture while preserving as much detail as possible.

PCA does something similar: it “photographs” high-dimensional data onto fewer dimensions, keeping the “important” details (the largest variations).

Working with so many variables is hard:

- Models run slowly.
- Patterns are hidden.
- Many columns may carry overlapping information.

Math, Step by Step

Let the original dataset be

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

n = number of observations (rows)

p = number of features (columns)

Math, Step by Step

Compute column means

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

Subtract to get the centered matrix

$$X_c = X - \mathbf{1}\mu^\top$$

The sample covariance matrix is

$$\Sigma = \frac{1}{n-1} X_c^\top X_c$$

$$\Sigma = \begin{bmatrix} \mathrm{Cov}(x_1, x_1) & \mathrm{Cov}(x_1, x_2) & \dots \\ \mathrm{Cov}(x_2, x_1) & \mathrm{Cov}(x_2, x_2) & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

The covariance matrix Σ collects all pairwise covariances

Eigenproblem

$$\Sigma w = \lambda w.$$

λ - eigenvalue, w - eigenvector

PCA

1. We compute the covariance matrix
(which describes how features vary together).

2. We find its eigenvectors → these are the principal components (new axes).

3. We look at the eigenvalues → they tell us how much data variation each principal component captures.

PCA

Eigenvectors - give the principal component directions (new axes of our data space).

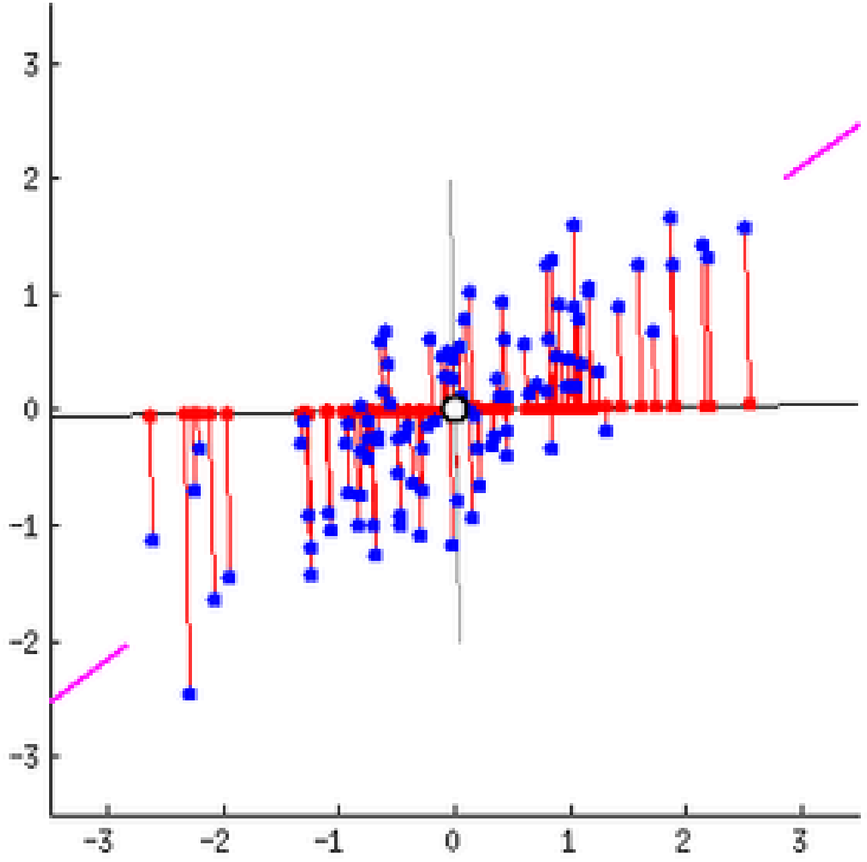
Eigenvalues -tell how much variance lies along each eigenvector

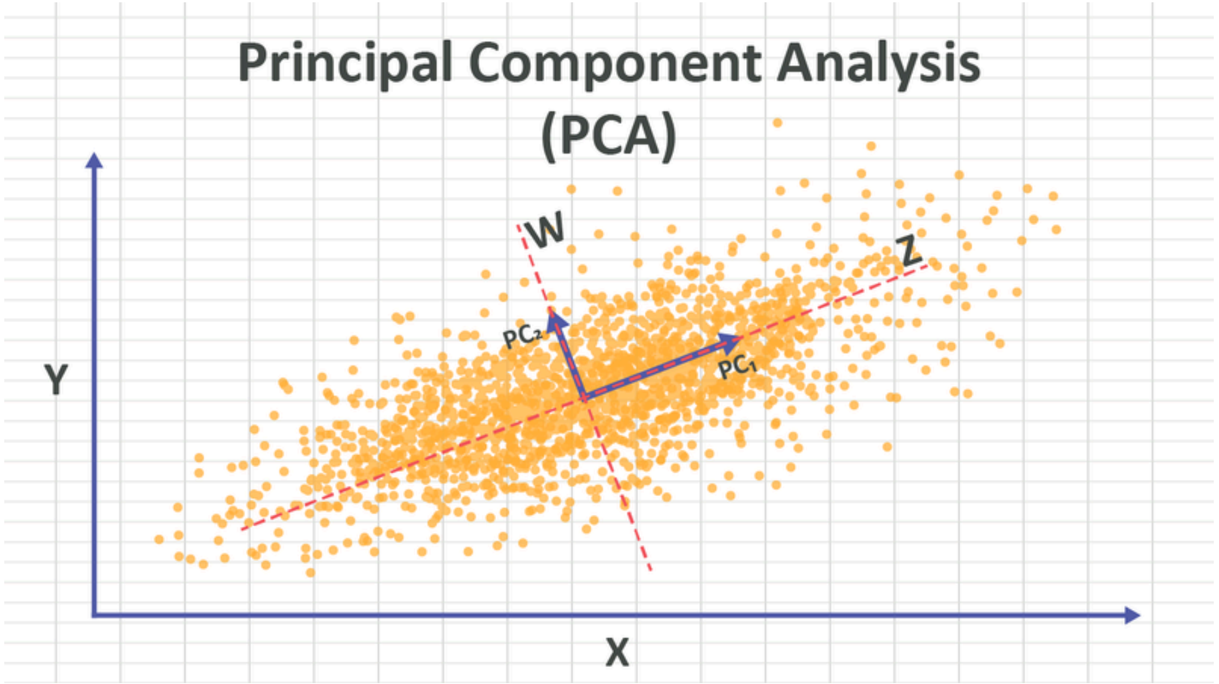
Eigenvalues ≈ stored variance.
Larger eigenvalue → more important component

Components are perpendicular, ensuring no duplicated information

Replace hundreds of correlated features with a few uncorrelated PC

PCA


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Principal Component Analysis
(PCA)
W
Z
PC₂
PC₁
Y
X

Applications

Field	Example	Why PCA Helps
Image processing	Compress handwriting digits (MNIST)	Smaller files, faster recognition
Face recognition	“Eigenfaces” – represent faces as a few PCs	Detect faces efficiently
Finance	Summarize dozens of stock returns	Identify key market factors
Medicine	Gene-expression data with thousands of genes	Visualize patient clusters
Marketing	Customer purchasing behavior	Segment customers without noise

Kernel Methods

Goal of Kernel Methods

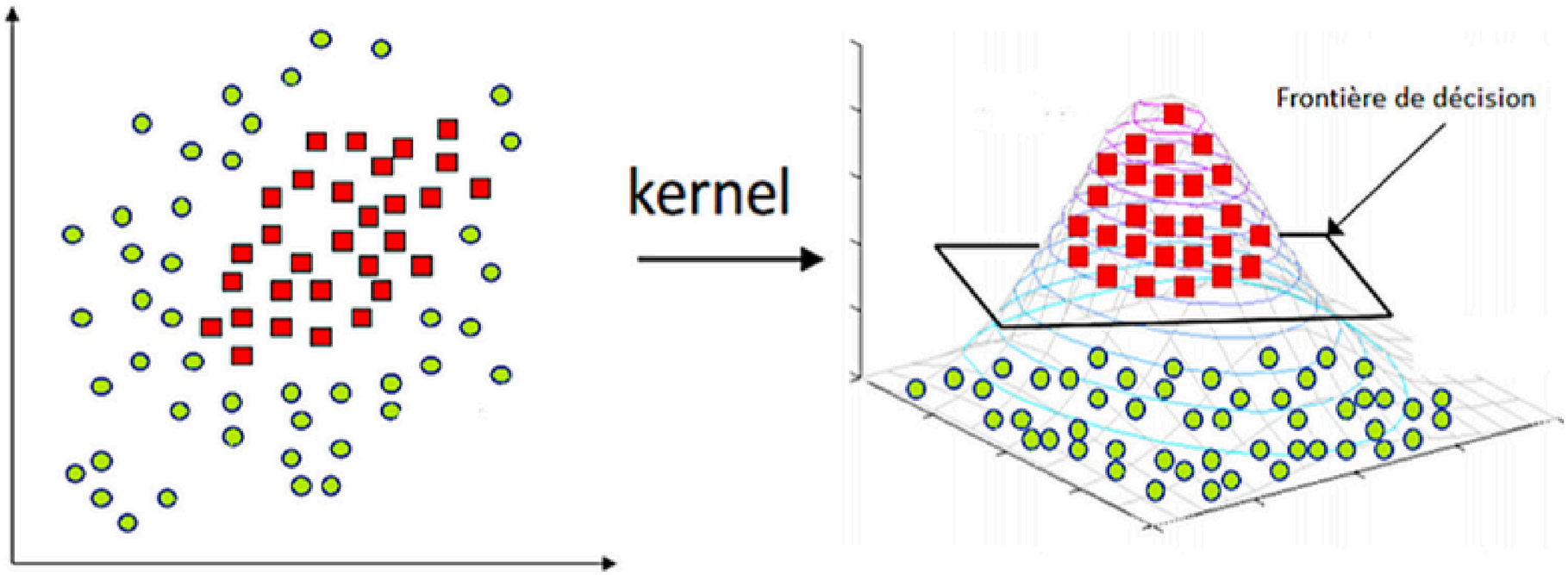
Many ML algorithms (like SVM, PCA, Ridge Regression) work best when the data is linearly separable or well-behaved in its current space. Real data is rarely like that.

Idea: Map data into a higher-dimensional space where linear methods work—without explicitly computing the mapping.

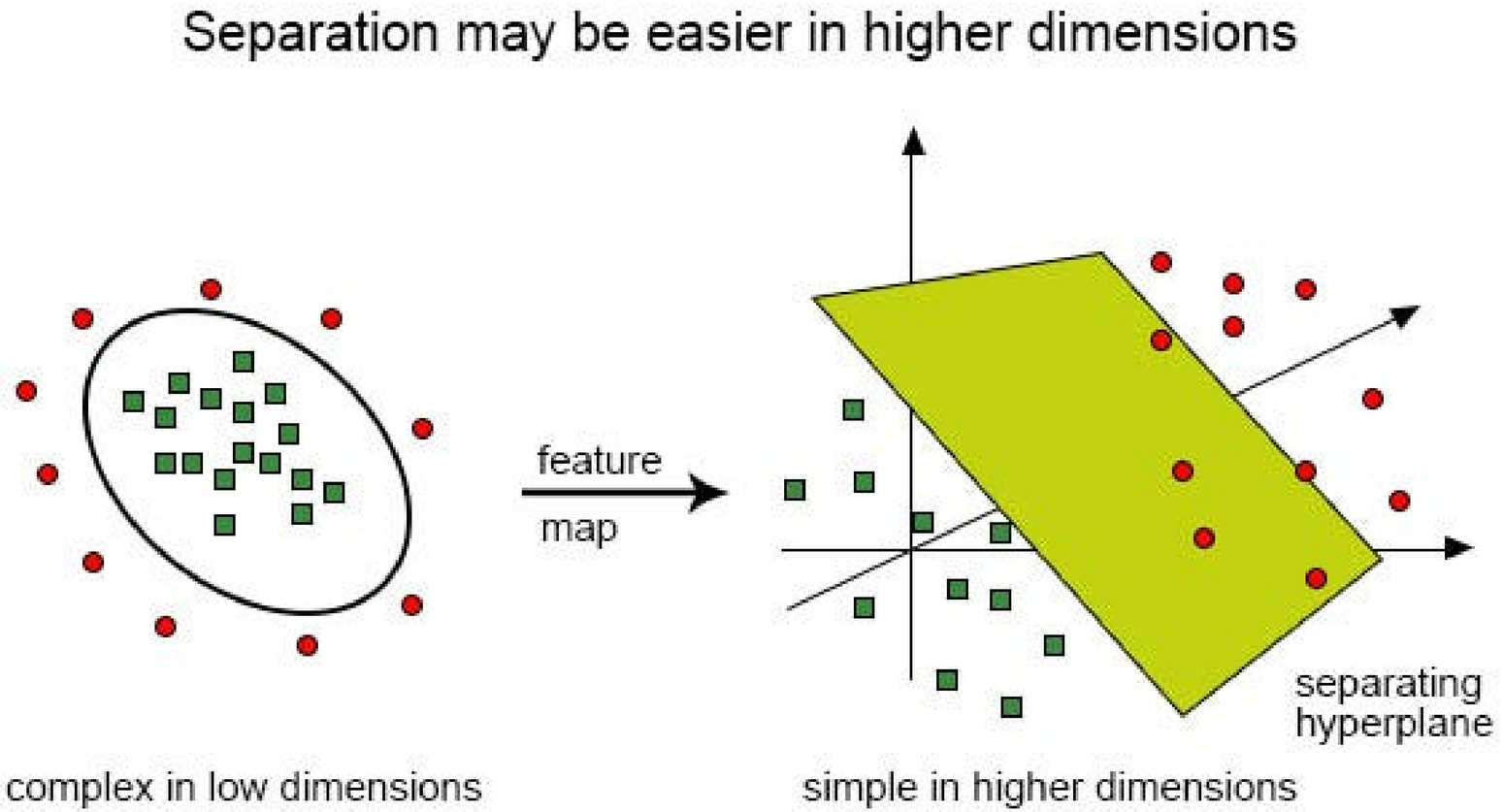
$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$$

where D≥d (and could even be infinite)

Kernel Methods



Kernel Methods



Kernel Methods

kernel trick:
Instead of mapping with a function φ(x) and then computing dot products φ(x)·φ(y),
Find a function K(x,y) that directly computes that dot product:

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

Kernel Methods

ϕ(x) - mapping function—it takes your original data point x and transforms it into a (usually higher-dimensional) feature vector

Input: the original feature vector x (e.g., [x1, x2]).

Output: a new vector ϕ(x) that contains more complex combinations of the original features.

Polynomial mapping

$$\phi(x) = (x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2)$$

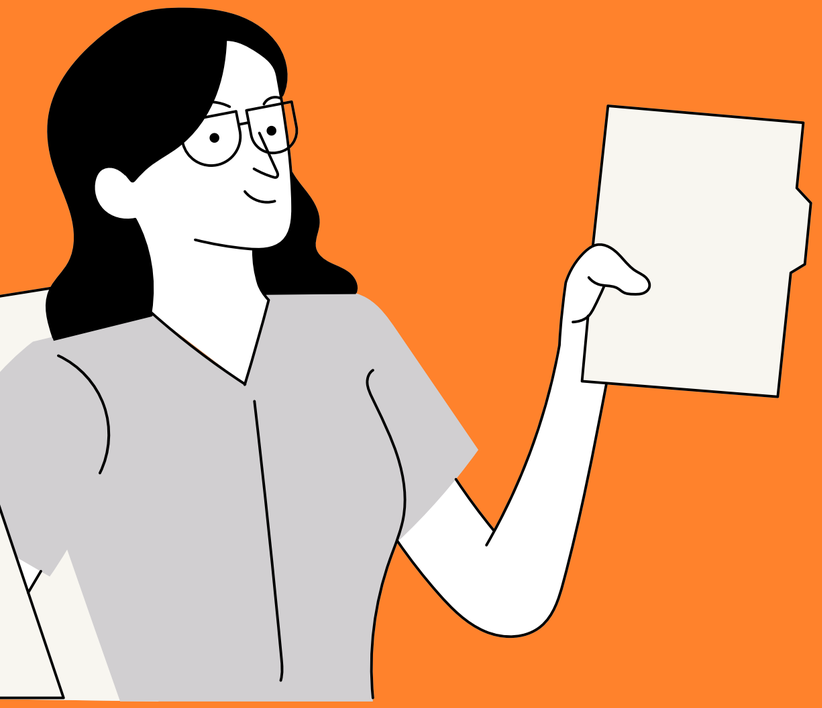
Many algorithms only need **inner products** like

$\langle \phi(x), \phi(y) \rangle$.

Explicitly calculating $\phi(x)$ can be huge or impossible.

The **kernel trick** lets us skip building $\phi(x)$ and directly compute that inner product with a kernel $K(x, y)$.

Kernel Functions



Purpose: To let algorithms like SVM, PCA, or regression learn non-linear patterns by working in a higher-dimensional feature space without explicitly computing that space

Key idea: Use a kernel function K(x, z) to compute the inner product of two data points after they are mapped into that hidden space

Benefits:

- Handle complex, non-linear data with linear algorithms.
- Computational efficiency: avoid expensive high-dimensional mappings.
- Flexibility: many kernels (RBF, polynomial, string, graph) capture different notions of similarity.
- Better performance: often higher accuracy and richer decision boundaries than plain linear models.

kernel matrix

SVM:

$$f(x) = \text{sign}\left(\sum_i \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b\right)$$

Notice: the formula never asks for ϕ(x) by itself—only for inner products between pairs.

If we can compute those inner products directly via K(xi,xj), the algorithm's math is identical to running it in the true high-dimensional feature space.

Kernel Functions

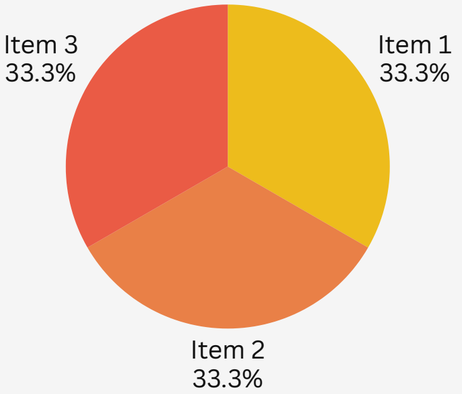
Kernel	Formula	Intuition / Use-case
Linear	$K(x, z) = x^\top z$	No mapping—just the plain dot product.
Polynomial	$K(x, z) = (x^\top z + c)^d$	Adds interactions up to degree d.
Gaussian RBF	$K(x, z) = \exp\left(-\|x - z\|^2 / 2\sigma^2\right)$	Infinite-dimensional, smooth; most common for SVM.
Sigmoid	$K(x, z) = \tanh(\alpha x^\top z + c)$	Related to neural-net activation.

Because the learning algorithms need only the pairwise inner products, and the kernel matrix gives those exact inner products without ever building the huge vectors, using K is mathematically equivalent to explicitly mapping into the high-dimensional feature space.

What is Data Visualization?

Data visualization is the practice of transforming quantitative or qualitative data into graphical form.

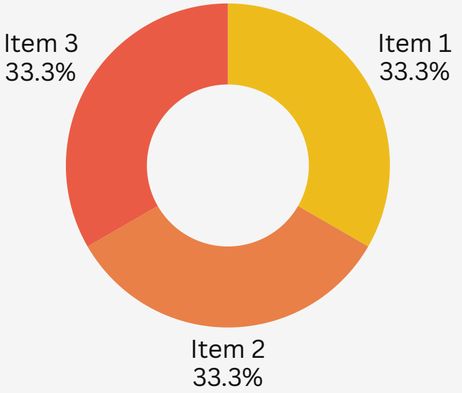
Its purpose is to reveal patterns, trends, correlations, outliers, and insights that are difficult to detect in raw tables.



Ratio: 60:40

Visualization supports:

1. Exploratory Data Analysis (EDA) – understanding the data before modeling.
2. Communication – presenting insights to non-technical stakeholders.
3. Decision-making – enabling faster and more accurate business decisions.
4. Monitoring – tracking KPIs, operational metrics, financial indicators.



Best Practices in Dashboard Visualization

1. One page = one story
2. Place the most important metrics at the top
3. Minimize unnecessary filters
4. Use consistent color coding
5. Allow drill-down and interaction when needed
6. Keep charts simple and interpretable

Humans instantly recognize visual differences in:

- color
- size
- position
- length
- orientation
- shape
- contrast
- texture

Color Theory

Avoid red–green combinations (color blindness)
Maintain high contrast
Limit number of colors: 5–7 maximum

Best Practices

1. Visualization is a fundamental skill in data analysis and business analytics.
2. Choosing the right chart requires understanding the analytical question.
3. Effective visualizations rely on perception, cognitive psychology, and design principles.
4. Visualizations must be clear, honest, and purposeful.

Category	Common Graphs
Comparison	Bar, Column, Line, Slope, Dot
Distribution	Histogram, KDE, Boxplot, Violin
Composition	Pie, Donut, Treemap, Waterfall
Relationship	Scatter, Bubble, Heatmap
Time-series	Line, Area, Step, Candlestick
Geospatial	Choropleth, Density Map
High-dimensional	PCA, t-SNE, Parallel Coordinates
Specialized	Funnel, Gantt, Sankey, Word Cloud

Common Python Libraries for Data Visualization

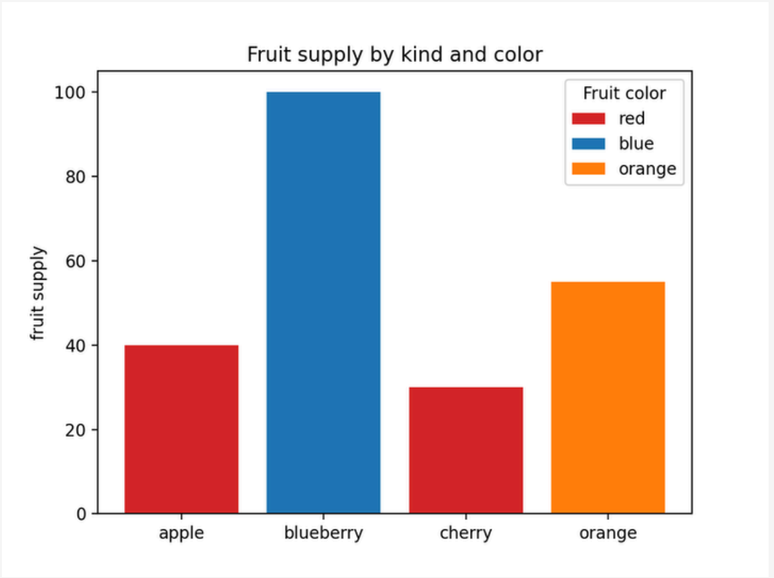
Matplotlib

Type: Low-level, foundational library
Best for: Basic plots, full control over details
Key features:

- Line, bar, scatter, histograms, pie charts, etc.
- Highly customizable
- Forms the foundation of most other visualization libraries

https://matplotlib.org

```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

fruits = ['apple', 'blueberry', 'cherry', 'orange']
counts = [40, 100, 30, 55]
bar_labels = ['red', 'blue', '_red', 'orange']
bar_colors = ['tab:red', 'tab:blue', 'tab:red', 'tab:orange']

ax.bar(fruits, counts, label=bar_labels, color=bar_colors)

ax.set_ylabel('fruit supply')
ax.set_title('Fruit supply by kind and color')
ax.legend(title='Fruit color')

plt.show()
```



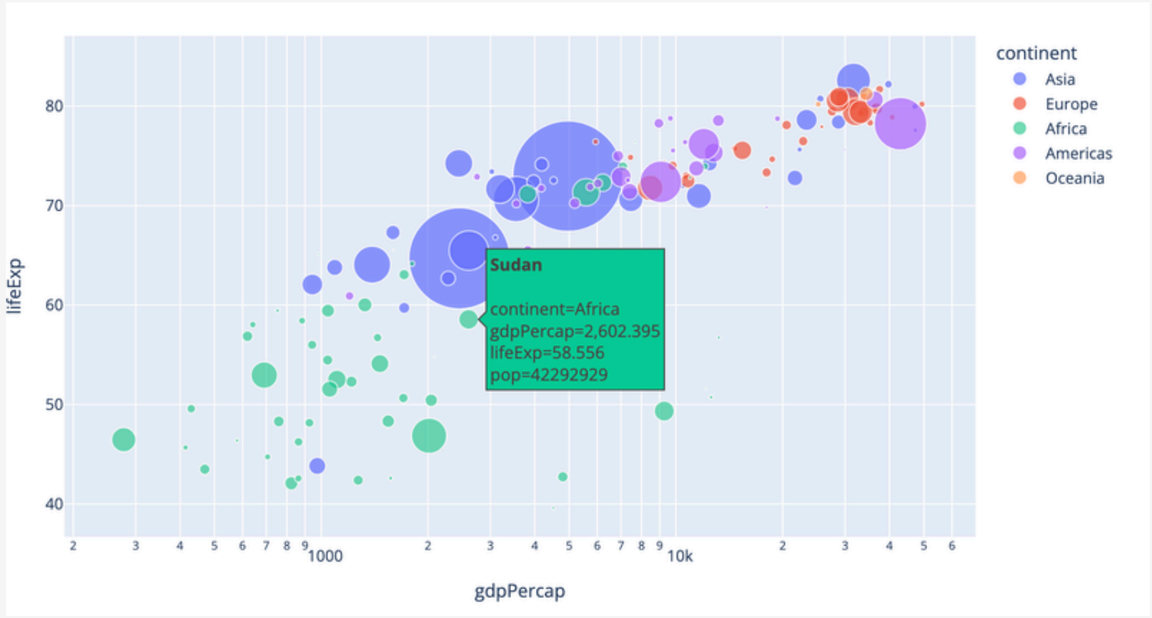
Common Python Libraries for Data Visualization

Plotly

Type: Interactive visualization
Best for: Dashboards, web apps, interactive analysis
Key features:

- Zooming, filtering, tooltips
- Works in Jupyter, Dash, and web
- Excellent for time series and maps

```python
import plotly.express as px
df = px.data.gapminder()

fig = px.scatter(df.query("year==2007"), x="gdpPercap", y="lifeExp",
                 size="pop", color="continent",
                 hover_name="country", log_x=True, size_max=60)
fig.show()
```



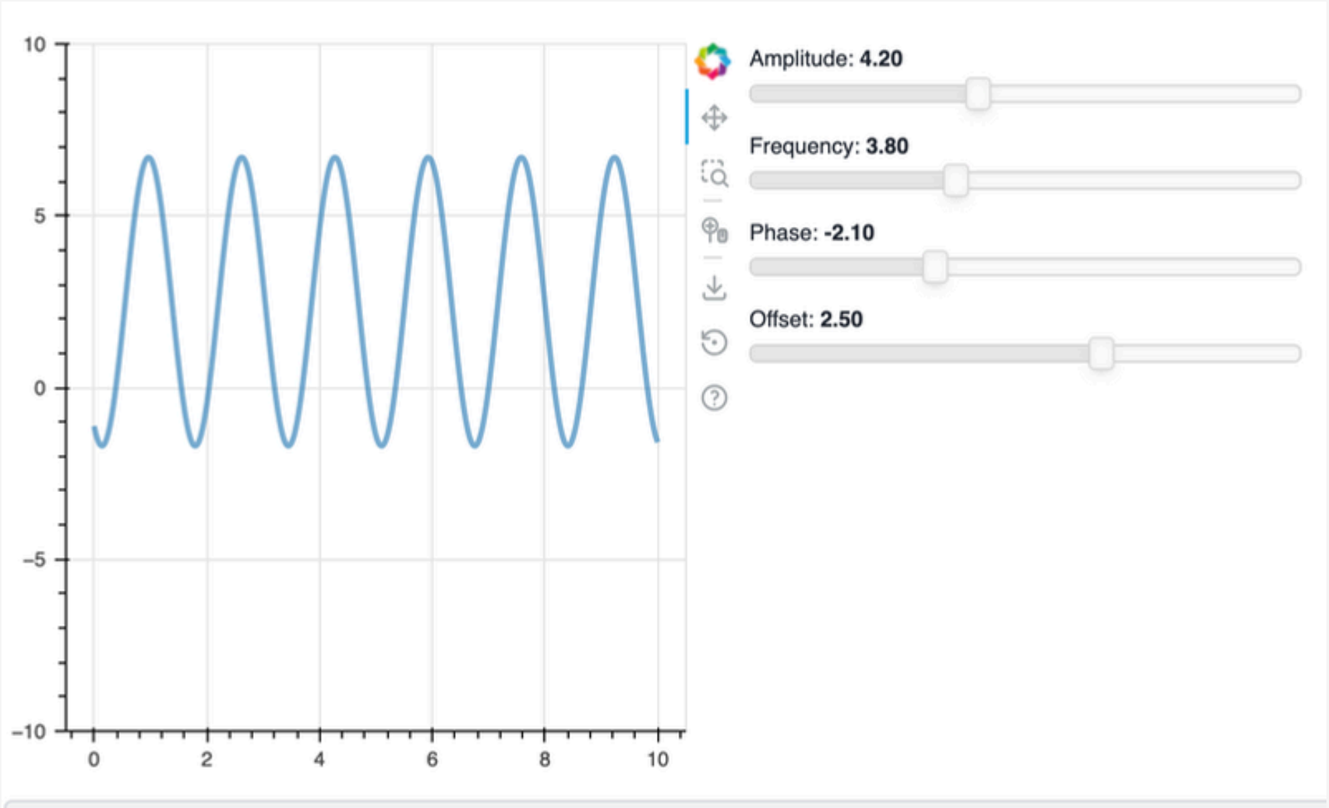
https://plotly.com/python/basic-charts/

Common Python Libraries for Data Visualization

Bokeh

Type: Interactive visualizations for web
Best for: Browser-based interactive dashboards
Key features:

- Server support (Bokeh Server)
- Linked brushing
- Very fast for large datasets



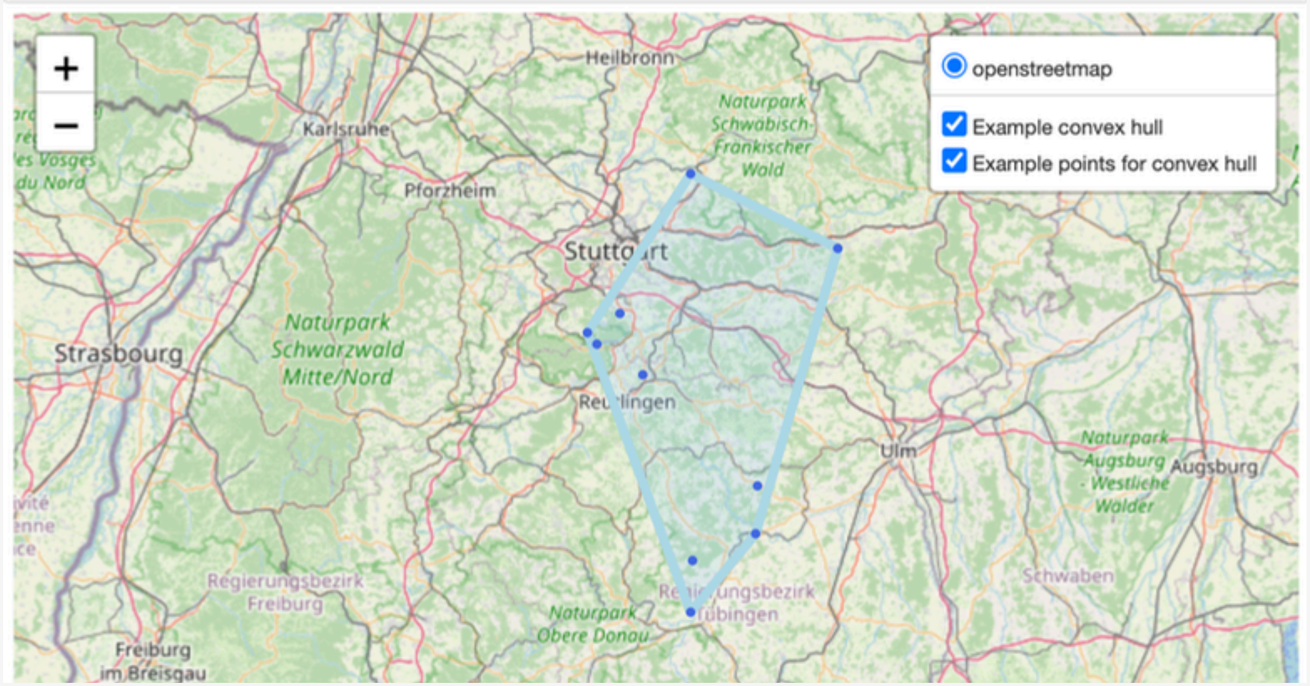
https://docs.bokeh.org

Common Python Libraries for Data Visualization

Folium

Type: Interactive mapping library
Best for: Leaflet-based maps in Jupyter
Key features:

- Perfect for geospatial dashboards
- Jupyter-friendly
- Add markers, layers, heatmaps



https://python-visualization.github.io

Common Python Libraries for Data Visualization

Library	Type	Best For
Matplotlib	2D base	Full control, fundamentals
Seaborn	Statistical	Quick beautiful plots
Plotly	Interactive	Dashboards, web apps
Bokeh	Interactive	Browser & servers
Altair	Declarative	Clean analysis plots
Dash	Framework	BI dashboards
ggplot	Grammar-based	R-style plotting
Holoviews	High-level	Large data + interactivity
Geopandas	Spatial	Maps, GIS
Folium	Interactive maps	Leaflet maps
Mayavi / PyVista	3D	Scientific visualization
MissingNo	Data quality	Missing values

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

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

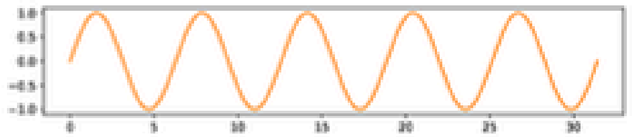
2 Prepare

```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

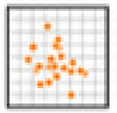
4 Observe



Choose

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```

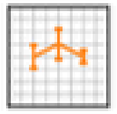
```
Z = np.random.uniform(0, 1, (8, 8))

ax.imshow(Z)
```

```
Z = np.random.uniform(0, 1, (8, 8))

ax.contourf(Z)
```

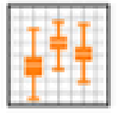
```
Z = np.random.uniform(0, 1, 4)

ax.pie(Z)
```

```
Z = np.random.normal(0, 1, 100)

ax.hist(Z)
```

```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



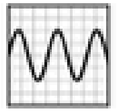
```
Z = np.random.normal(0, 1, (100, 3))

ax.boxplot(Z)
```



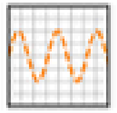
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

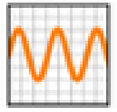
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



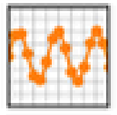
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



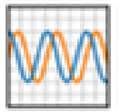
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



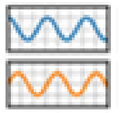
Organize

You can plot several data on the same figure, but you can also split a figure in several subplots (named *Axes*):

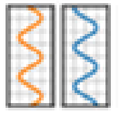
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

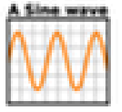


```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

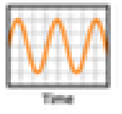


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```




```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```



Explore

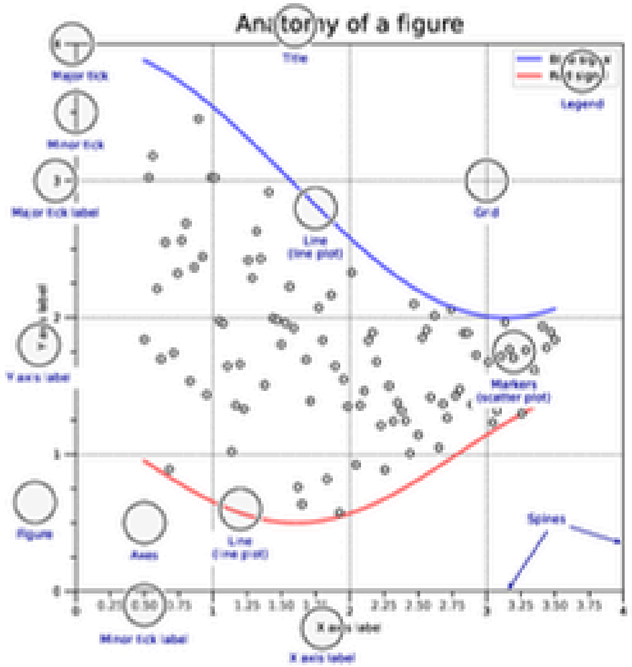
Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib for intermediate users

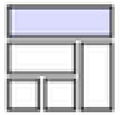
A matplotlib figure is composed of a hierarchy of elements that forms the actual figure. Each element can be modified.



Figure, axes & spines

```
fig, axs = plt.subplots(3, 3)
axs[0, 0].set_facecolor("#ddddff")
axs[2, 2].set_facecolor("#ffffdd")
```

```
gs = fig.add_gridspec(3, 3)
ax = fig.add_subplot(gs[0, :])
ax.set_facecolor("#ddddff")
```



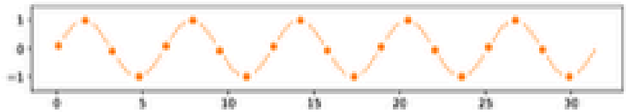
```
fig, ax = plt.subplots()
ax.spines["top"].set_color("None")
ax.spines["right"].set_color("None")
```

Ticks & labels

```
from mpl.ticker import MultipleLocator as ML
from mpl.ticker import ScalarFormatter as SF
ax.xaxis.set_minor_locator(ML(0.2))
ax.xaxis.set_minor_formatter(SF())
ax.tick_params(axis='x',which='minor',rotation=90)
```

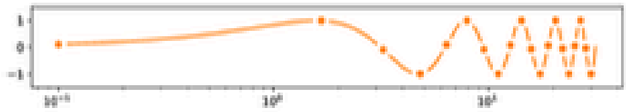
Lines & markers

```
X = np.linspace(0.1, 10*np.pi, 1000)
Y = np.sin(X)
ax.plot(X, Y, "C1o:", markevery=50, mec="1.0")
```



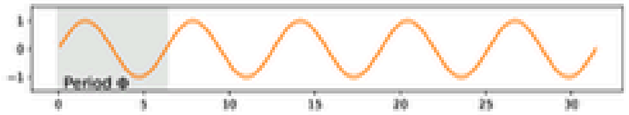
Scales & projections

```
fig, ax = plt.subplots()
ax.set_xscale("log")
ax.plot(X, Y, "C1o-", markevery=50, mec="1.0")
```



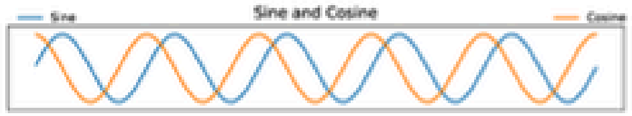
Text & ornaments

```
ax.fill_betweenx([-1, 1], [0], [2*np.pi])
ax.text(0, -1, r" Period $\Phi$")
```



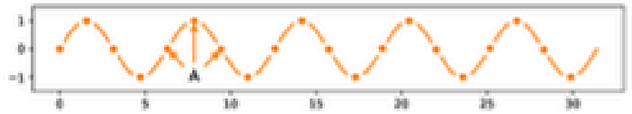
Legend

```
ax.plot(X, np.sin(X), "C0", label="Sine")
ax.plot(X, np.cos(X), "C1", label="Cosine")
ax.legend(bbox_to_anchor=(0,1,1,.1), ncol=2,
          mode="expand", loc="lower left")
```



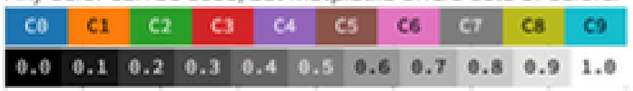
Annotation

```
ax.annotate("A", (X[250],Y[250]), (X[250],-1),
  ha="center", va="center", arrowprops={
    "arrowstyle": "->", "color": "C1"})
```



Colors

Any color can be used, but Matplotlib offers sets of colors:



Size & DPI

Consider a square figure to be included in a two-column A4 paper with 2 cm margins on each side and a column separation of 1 cm. The width of a figure is (21 - 2*2 - 1)/2 = 8 cm. One inch being 2.54 cm, figure size should be 3.15×3.15 in.

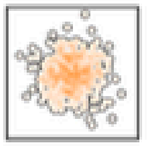
```
fig = plt.figure(figsize=(3.15, 3.15), dpi=50)
plt.savefig("figure.pdf", dpi=600)
```

Matplotlib tips & tricks

Transparency

Scatter plots can be enhanced by using transparency (alpha) in order to show area with higher density. Multiple scatter plots can be used to delineate a frontier.

```
X = np.random.normal(-1, 1, 500)
Y = np.random.normal(-1, 1, 500)
ax.scatter(X, Y, 50, "0.0", lw=2) # optional
ax.scatter(X, Y, 50, "1.0", lw=0) # optional
ax.scatter(X, Y, 40, "C1",  lw=0, alpha=0.1)
```



Rasterization

If your figure has many graphical elements, such as a huge scatter, you can rasterize them to save memory and keep other elements in vector format.

```
X = np.random.normal(-1, 1, 10_000)
Y = np.random.normal(-1, 1, 10_000)
ax.scatter(X, Y, rasterized=True)
fig.savefig("rasterized-figure.pdf", dpi=600)
```

Offline rendering

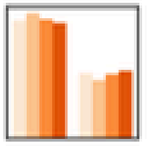
Use the Agg backend to render a figure directly in an array.

```
from matplotlib.backends.backend_agg import FigureCanvas
canvas = FigureCanvas(Figure())
... # draw some stuff
canvas.draw()
Z = np.array(canvas.renderer.buffer_rgba())
```

Range of continuous colors

You can use colormap to pick from a range of continuous colors.

```
X = np.random.randn(1000, 4)
cmap = plt.get_cmap("Oranges")
colors = cmap([0.2, 0.4, 0.6, 0.8])

ax.hist(X, 2, histtype='bar', color=colors)
```



Text outline

Use text outline to make text more visible.

```
import matplotlib.patheffects as fx
text = ax.text(0.5, 0.1, "Label")
text.set_path_effects([
  fx.Stroke(linewidth=3, foreground='1.0'),
  fx.Normal()])
```

Multiline plot

You can plot several lines at once using *None* as separator.

```
X,Y = [], []
for x in np.linspace(0, 10*np.pi, 100):
  X.extend([x, x, None]), Y.extend([0, sin(x), None])
ax.plot(X, Y, "black")
```

Dotted lines

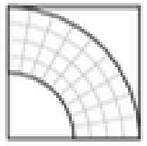
To have rounded dotted lines, use a custom `linestyle` and modify `dash_capstyle`.

```
ax.plot([0, 1], [0, 0], "C1",
        linestyle=(0, (0.01, 1)), dash_capstyle='round')
ax.plot([0, 1], [1, 1], "C1",
        linestyle=(0, (0.01, 2)), dash_capstyle='round')
```

Combining axes

You can use overlaid axes with different projections.

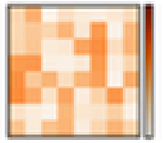
```
ax1 = fig.add_axes([0, 0, 1, 1],
                   label='cartesian')
ax2 = fig.add_axes([0, 0, 1, 1],
                   label='polar',
                   projection='polar')
```



Colorbar adjustment

You can adjust a colorbar's size when adding it.

```
im = ax.imshow(Z)

cb = plt.colorbar(im,
        fraction=0.046, pad=0.04)
cb.set_ticks([])
```



Taking advantage of typography

You can use a condensed font such as Roboto Condensed to save space on tick labels.

```
for tick in ax.get_xticklabels(which='both'):
    tick.set_fontname('Roboto Condensed')
```

Getting rid of margins

Once your figure is finished, you can call `tight_layout()` to remove white margins. If there are remaining margins, you can use the `pdfcrop` utility (comes with TeX live).

Hatching

You can achieve a nice visual effect with thick hatch patterns.

```
cmap = plt.get_cmap("Oranges")
plt.rcParams['hatch.color'] = cmap(0.2)
plt.rcParams['hatch.linewidth'] = 8
ax.bar(X, Y, color=cmap(0.6), hatch="/")
```

Read the documentation

Matplotlib comes with an extensive documentation explaining the details of each command and is generally accompanied by examples. Together with the huge online gallery, this documentation is a gold-mine.

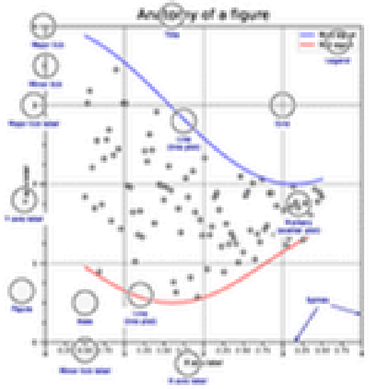
matplotlib

Cheat sheet — Version 3.9.4

Quick start

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


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X, Y, color='green')

fig.savefig("figure.pdf")
plt.show()
```

Anatomy of a figure



Subplots layout

```
subplot[s](rows, cols, ...)
fig, axs = plt.subplots(3, 3)

G = gridspec(rows,cols, ...)
ax = G[0, :]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax = d.new_horizontal("10%")
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/questions/tagged/matplotlib
- https://gitter.im/matplotlib/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

```
plot([X], Y, [fmt], ...)
X, Y, fmt, color, marker, linestyle

scatter(X, Y, ...)
X, Y, [s]izes, [c]olors, marker, cmap

bar[h](x, height, ...)
x, height, width, bottom, align, color

im[show](Z, ...)
Z, cmap, interpolation, extent, origin

contour[f]([X], [Y], Z, ...)
X, Y, Z, levels, colors, extent, origin

pcolormesh([X], [Y], Z, ...)
X, Y, Z, vmin, vmax, cmap

quiver([X], [Y], U, V, ...)
X, Y, U, V, C, units, angles

pie(X, ...)
Z, explode, labels, colors, radius

text(x, y, text, ...)
x, y, text, va, ha, size, weight, transform

fill[_between][x](...)
X, Y1, Y2, color, where
```

Advanced plots

```
step(X, Y, [fmt], ...)
X, Y, fmt, color, marker, where

boxplot(X, ...)
X, notch, sym, bootstrap, widths

errorbar(X,Y,xerr,yerr, ...)
X, Y, xerr, yerr, fmt

hist(X, bins, ...)
X, bins, range, density, weights

violinplot(D, ...)
D, positions, widths, vert

barbs([X], [Y], U, V, ...)
X, Y, U, V, C, length, pivot, sizes

eventplot(positions, ...)
positions, orientation, lineoffsets

hexbin(X, Y, C, ...)
X, Y, C, gridsize, bins
```

Scales

```
ax.set_[xy]scale(scale, ...)
```

linear — any values
log — values > 0
symlog — any values
logit — 0 < values < 1

Projections

```
subplot(..., projection=p)
```

p='polar'
p='3d'

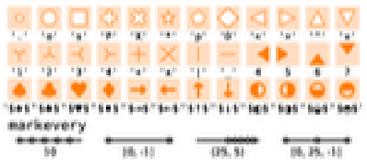
```
p=ccrs.Orthographic()
import cartopy.crs as ccrs
```

Lines

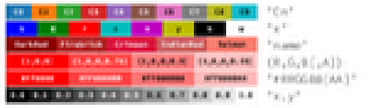
```
linestyle or ls
```


```
capstyle or dash_capstyle
```

Markers

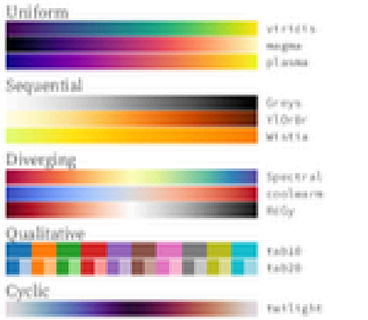


Colors



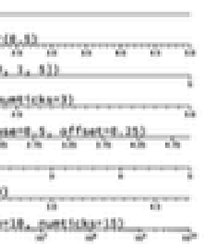
Colormaps

```
plt.get_cmap(name)
```



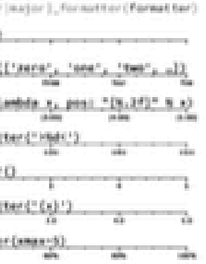
Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```



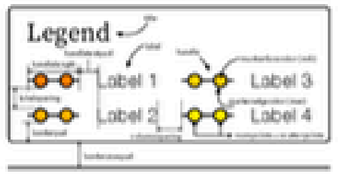
Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(Formatter)
```



Ornaments

```
ax.legend(...)
handles, labels, loc, title, frameon
```




```
ax.colorbar(...)
mappable, ax, cax, orientation
```

```
ax.annotate(...)
text, xy, xytext, xycoords, textcoords, arrowprops
```

Event handling

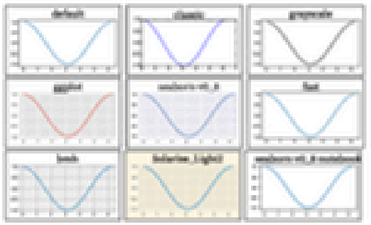
```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0, 2*np.pi, 100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
    line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
    plt.gcf(), animate, interval=5)
plt.show()
```

Styles

```
plt.style.use(style)
```



Quick reminder

```
ax.grid()
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(ticks, [labels])
ax.set_[xy]ticklabels(labels)
ax.set_title(title)
ax.tick_params(width=10, ...)
ax.set_axis_[on|off]()

fig.suptitle(title)
fig.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, ...)
[fig|ax].patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

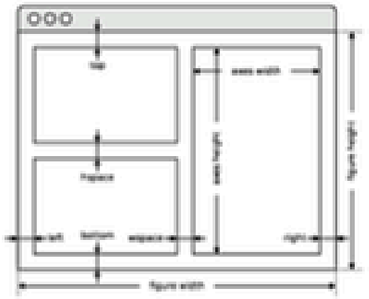
ctrl + s — Save	ctrl + w — Close plot
r — Reset view	f — Fullscreen 0/1
f — View forward	b — View back
p — Pan view	o — Zoom to rect
x — X pan/zoom	y — Y pan/zoom
g — Minor grid 0/1	G — Major grid 0/1
l — X axis log/linear	L — Y axis log/linear

Ten simple rules

1. Know your audience
2. Identify your message
3. Adapt the figure
4. Captions are not optional
5. Do not trust the defaults
6. Use color effectively
7. Do not mislead the reader
8. Avoid "chartjunk"
9. Message trumps beauty
10. Get the right tool

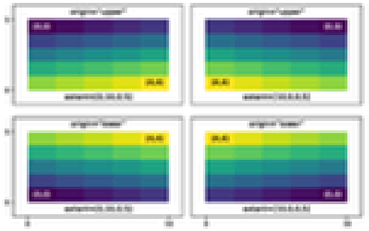
Axes adjustments

plt.subplots_adjust(...)



Extent & origin

ax.imshow(extent=..., origin=...)



Text alignments

ax.text(..., ha=... , va=..., ...)

Text parameters

ax.text(..., family=..., size=..., weight=...)
ax.text(..., fontproperties=...)

The quick brown fox — xx-large (1.73)
The quick brown fox — x-large (1.44)
The quick brown fox — large (1.20)
The quick brown fox — medium (1.00)
The quick brown fox — small (0.83)
The quick brown fox — x-small (0.69)
The quick brown fox — xx-small (0.58)

The quick brown fox jumps over the lazy dog — black (900)
The quick brown fox jumps over the lazy dog — bold (700)
The quick brown fox jumps over the lazy dog — semibold (600)
The quick brown fox jumps over the lazy dog — normal (400)
The quick brown fox jumps over the lazy dog — ultralight (100)

The quick brown fox jumps over the lazy dog — monospace
The quick brown fox jumps over the lazy dog — serif
The quick brown fox jumps over the lazy dog — sans
The quick brown fox jumps over the lazy dog — cursive

The quick brown fox jumps over the lazy dog — italic
The quick brown fox jumps over the lazy dog — normal

THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG — small-caps
The quick brown fox jumps over the lazy dog — normal

Uniform colormaps

viridis
plasma
inferno
magma
cividis

Sequential colormaps

Greys
Purples
Blues
Greens
Oranges
Reds
YlOrBr
YlOrRd
OrRd
PuRd
RdPu
BuPu
GnBu
PuBu
YlGnBu
PuBuGn
BuGn
YlGn

Diverging colormaps

PiYG
PRGn
BrBG
PuOr
RdGy
RdBu
RdYlBu
RdYlGn
Spectral
coolwarm
bwr
seismic

Qualitative colormaps

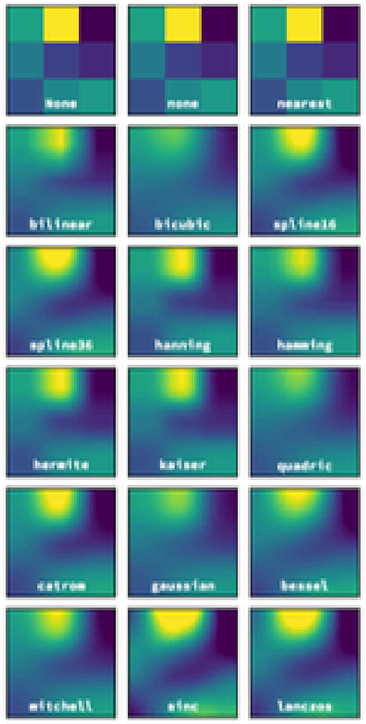
Pastel1
Pastel2
Paired
Accent
Dark2
Set1
Set2
Set3
tab10
tab20
tab20b
tab20c

Miscellaneous colormaps

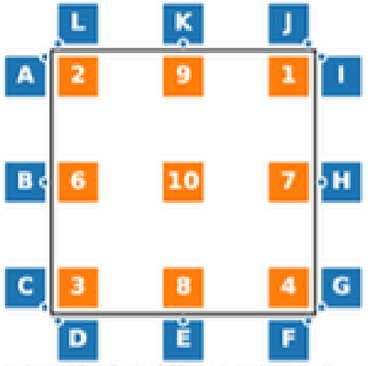
terrain
ocean
cubehelix
rainbow
twilight

Color names

Image interpolation



Legend placement

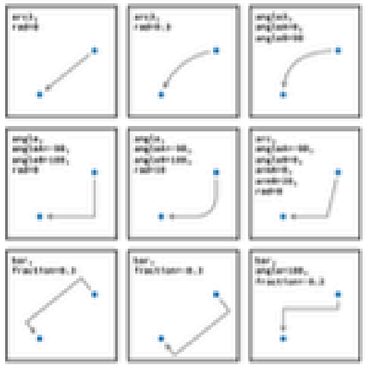


ax.**legend**(loc="string", bbox_to_anchor=(x, y))

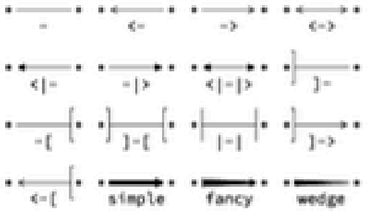
2: upper left — 9: upper center — 1: upper right
6: center left — 10: center — 7: center right
3: lower left — 8: lower center — 4: lower right

A: upper right / (-0.1, 0.9) — B: center right / (-0.1, 0.5)
C: lower right / (-0.1, 0.1) — D: upper left / (0.1, -0.1)
E: upper center / (0.5, -0.1) — F: upper right / (0.9, -0.1)
G: lower left / (1.1, 0.1) — H: center left / (1.1, 0.5)
I: upper left / (1.1, 0.9) — J: lower right / (0.9, 1.1)
K: lower center / (0.5, 1.1) — L: lower left / (0.1, 1.1)

Annotation connection styles



Annotation arrow styles



How do I ...

... **resize a figure?**
→ fig.set_size_inches(w, h)
... **save a figure?**
→ fig.savefig("figure.pdf")
... **save a transparent figure?**
→ fig.savefig("figure.pdf", transparent=True)
... **clear a figure/an axes?**
→ fig.clear() → ax.clear()
... **close all figures?**
→ plt.close("all")
... **remove ticks?**
→ ax.set_[xy]ticks([])
... **remove tick labels ?**
→ ax.set_[xy]ticklabels([])
... **rotate tick labels ?**
→ ax.tick_params(axis="x", rotation=90)
... **hide top spine?**
→ ax.spines["top].set_visible(False)
... **hide legend border?**
→ ax.legend(frameon=False)
... **show error as shaded region?**
→ ax.fill_between(X, Y+error, Y-error)
... **draw a rectangle?**
→ ax.add_patch(plt.Rectangle((0, 0), 1, 1)
... **draw a vertical line?**
→ ax.axvline(x=0.5)
... **draw outside frame?**
→ ax.plot(..., clip_on=False)
... **use transparency?**
→ ax.plot(..., alpha=0.25)
... **convert an RGB image into a gray image?**
→ gray = 0.2989*R + 0.5870*G + 0.1140*B
... **set figure background color?**
→ fig.patch.set_facecolor("grey")
... **get a reversed colormap?**
→ plt.get_cmap("viridis_r")
... **get a discrete colormap?**
→ plt.get_cmap("viridis", 10)
... **show a figure for one second?**
→ fig.show(block=False), time.sleep(1)

Performance tips

```
scatter(X, Y)                                  slow
plot(X, Y, marker="o", ls="")                  fast

for i in range(n): plot(i, x[i], "o")          slow
plot(X, marker="o", ls="")                     fast

cla(); imshow(…); canvas.draw()                slow
im.set_data(…); canvas.draw()                  fast
```

Beyond Matplotlib

Seaborn: Statistical data visualization
Cartopy: Geospatial data processing
yt: Volumetric data visualization
mpld3: Bringing Matplotlib to the browser
Datashader: Large data processing pipeline
plotnine: A grammar of graphics for Python

NUMFOCUS
OPEN CODE = BETTER SCIENCE

THANK YOU!