Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Jan. 6 – 17, 2020

Python for Data Analytics

MatplotLib

Outline

- Introduction to Data Visualization
- What is MatplotLib?
- Pyplot submodule functions
- What is Seaborn?

Introduction to Data Visualization

Visualization

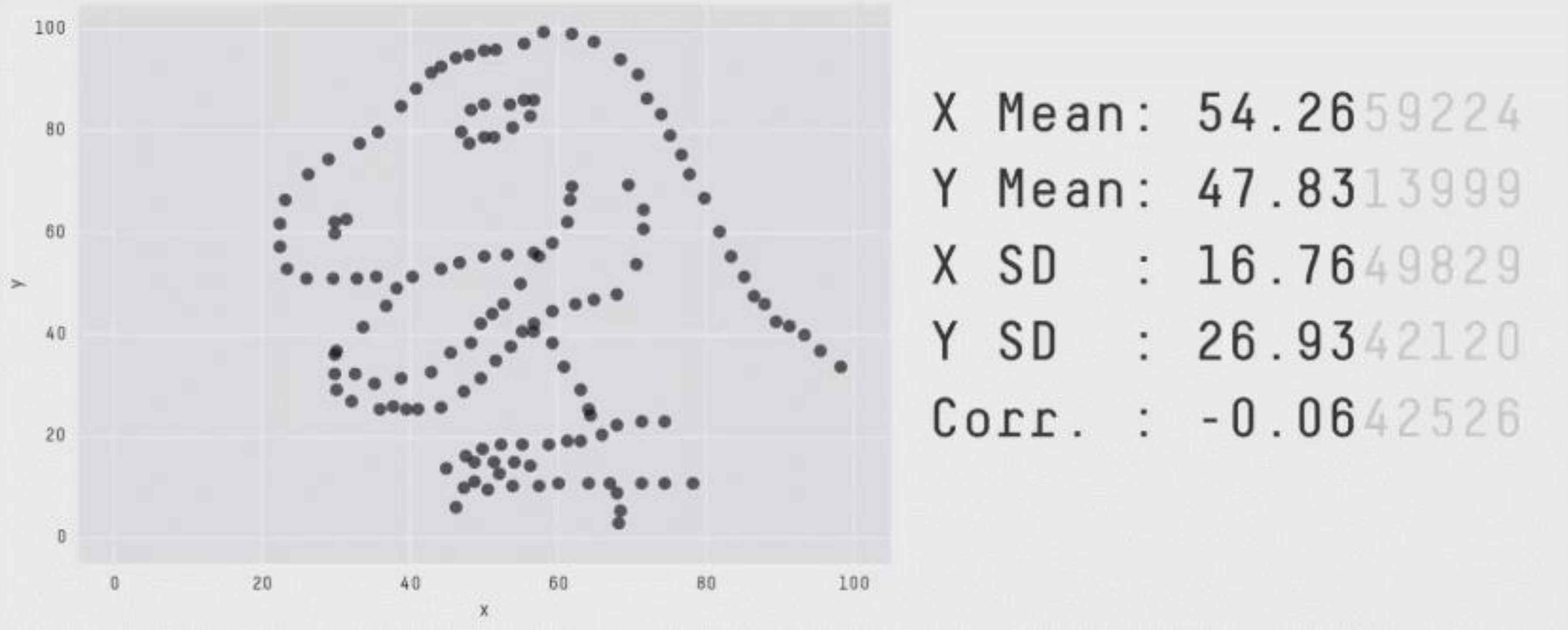
- Data exploration visualization: figuring out what is true
- Data presentation visualization: convincing other people it is true

- Before you run any analysis, build any machine learning system, etc., always visualize your data

- If you can't identify a trend or make a prediction for your dataset, neither will an automated algorithm

Visualization vs. Statistics

- Visualization almost always presents a more informative (though less quantitative) view of your data than statistics
  - This is a mathematical property: *n* data points and *m* equations to satisfy, with *n* > *m*



Source: https://www.autodeskresearch.com/publications/samestats

Data Types

- **Nominal**
  - Categorical data
  - No quantitative value, no ordering
  - Example – Pet: {dog, cat, rabbit, …}
  - Operations: $=, \neq$
- **Ordinal**
  - Categorical data, with ordering
  - Example – Rating: {1, 2, 3, 4, 5}
  - Operations: $=, \neq, \geq, \leq, >, <$
- **Interval**
  - Numerical data
  - Zero has no fixed meaning
  - Example – Temperature Celsius
  - Operations: $=, \neq, \geq, \leq, >, <, +, -$
- **Ratio**
  - Numerical data
  - Zero has special meaning
  - Example – Height, Weight, …
  - Operations: $=, \neq, \geq, \leq, >, <, +, -, \times, \div$

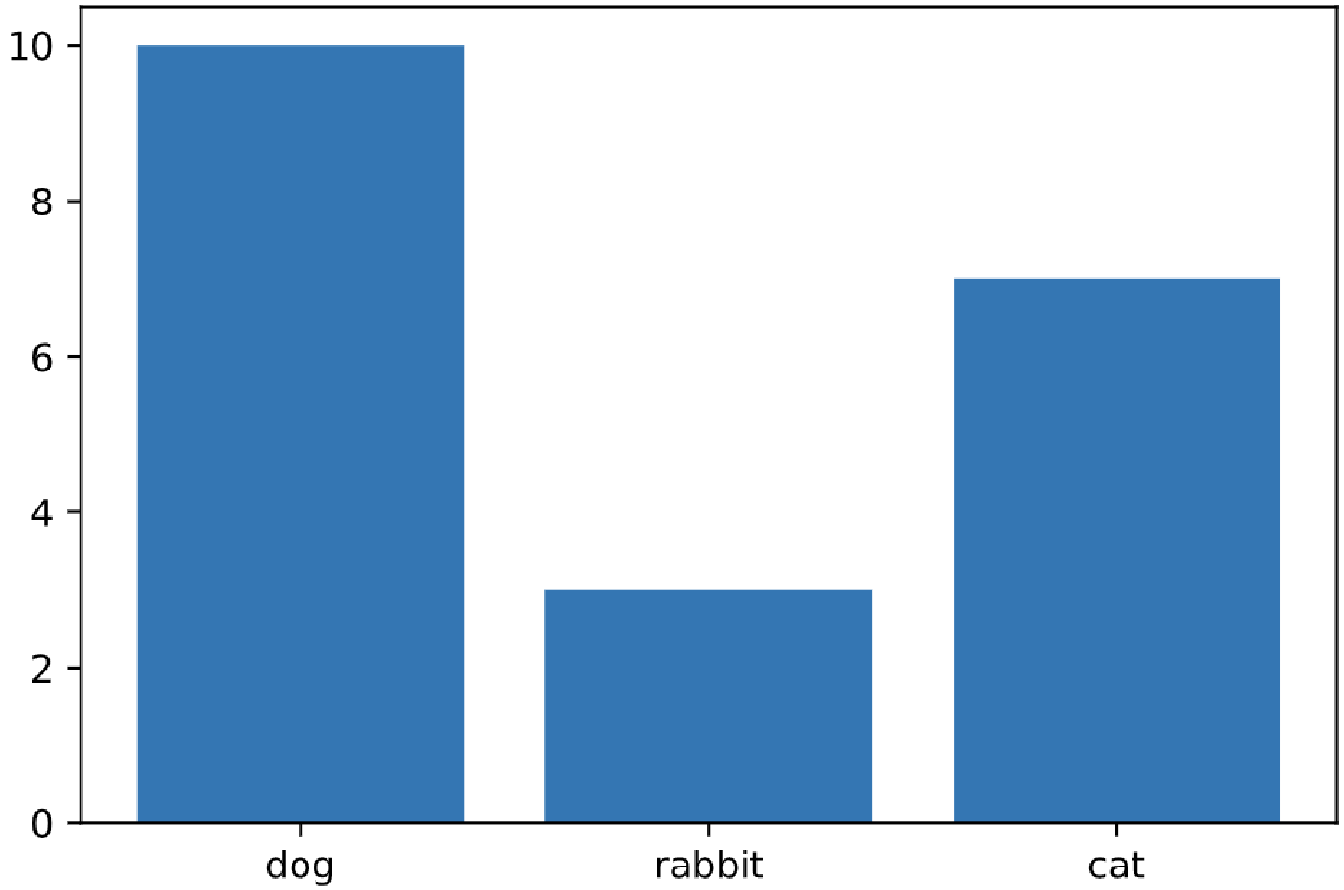
Visualization Types

- 1D
  - Bar chart, pie chart, histogram
- 2D
  - Scatter plot, line plot, box and whisker plot, heatmap
- 3D
  - Scatter matrix, bubble chart

1D: Bar Chart

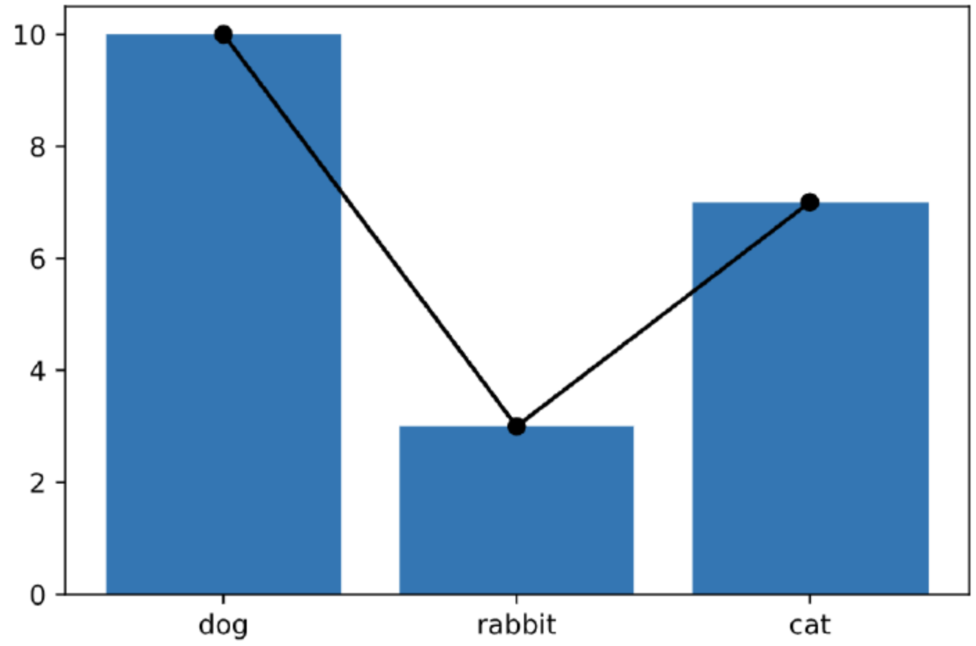
	Data
Nominal	✓
Ordinal	✓
Interval	✗
Ratio	✗

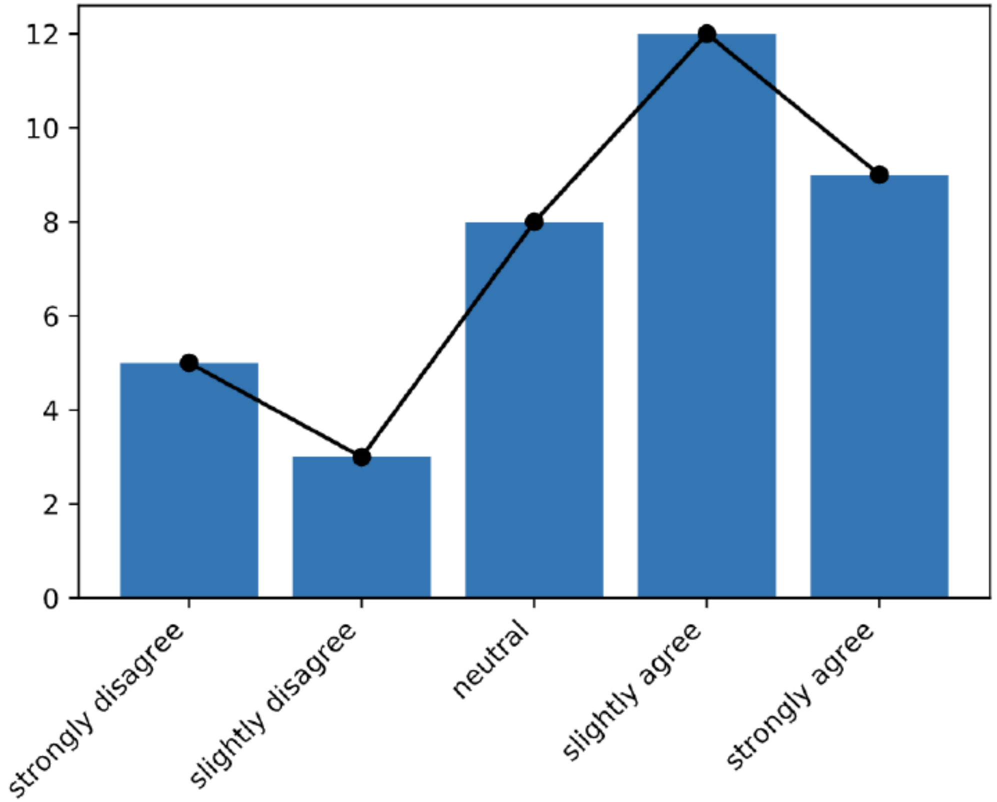
Suggestions, not rules



1D: Bar Chart (bad)

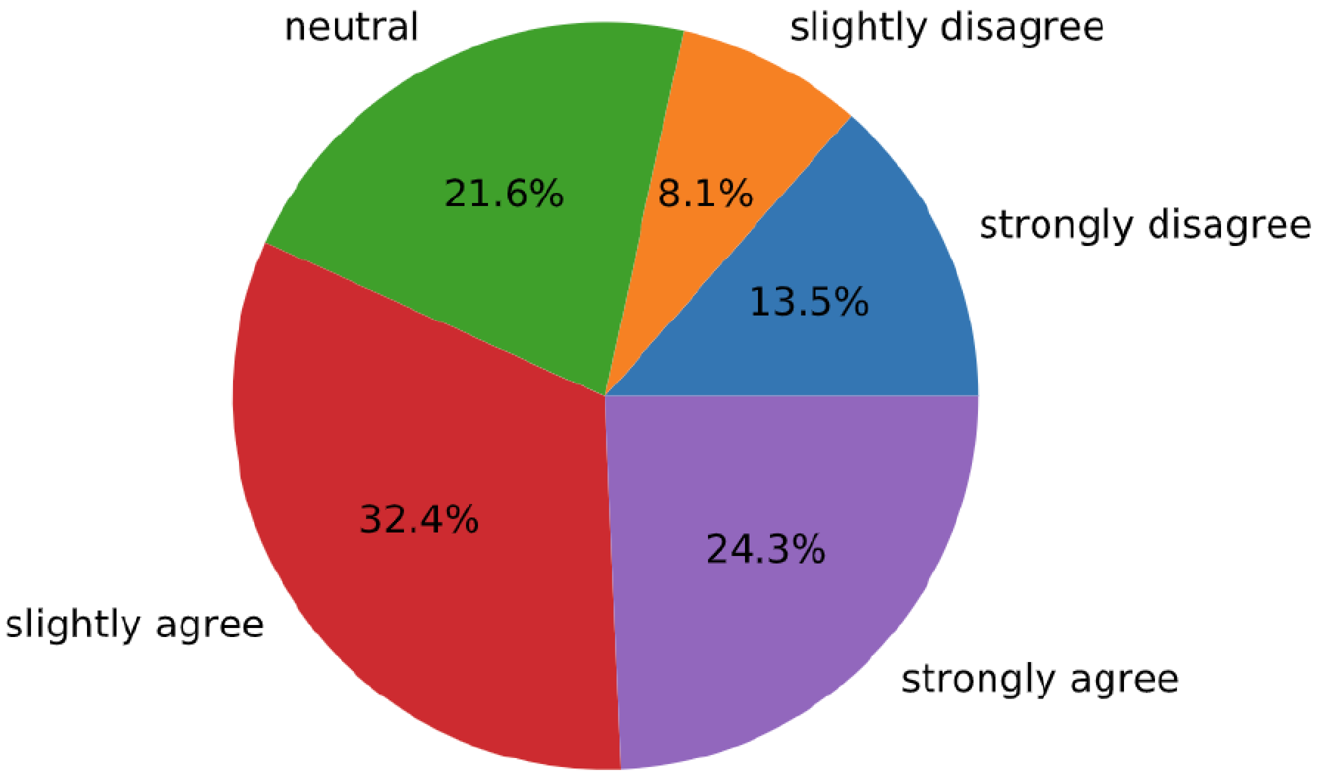
- Don't use lines within a bar chart categorical or ordinal features





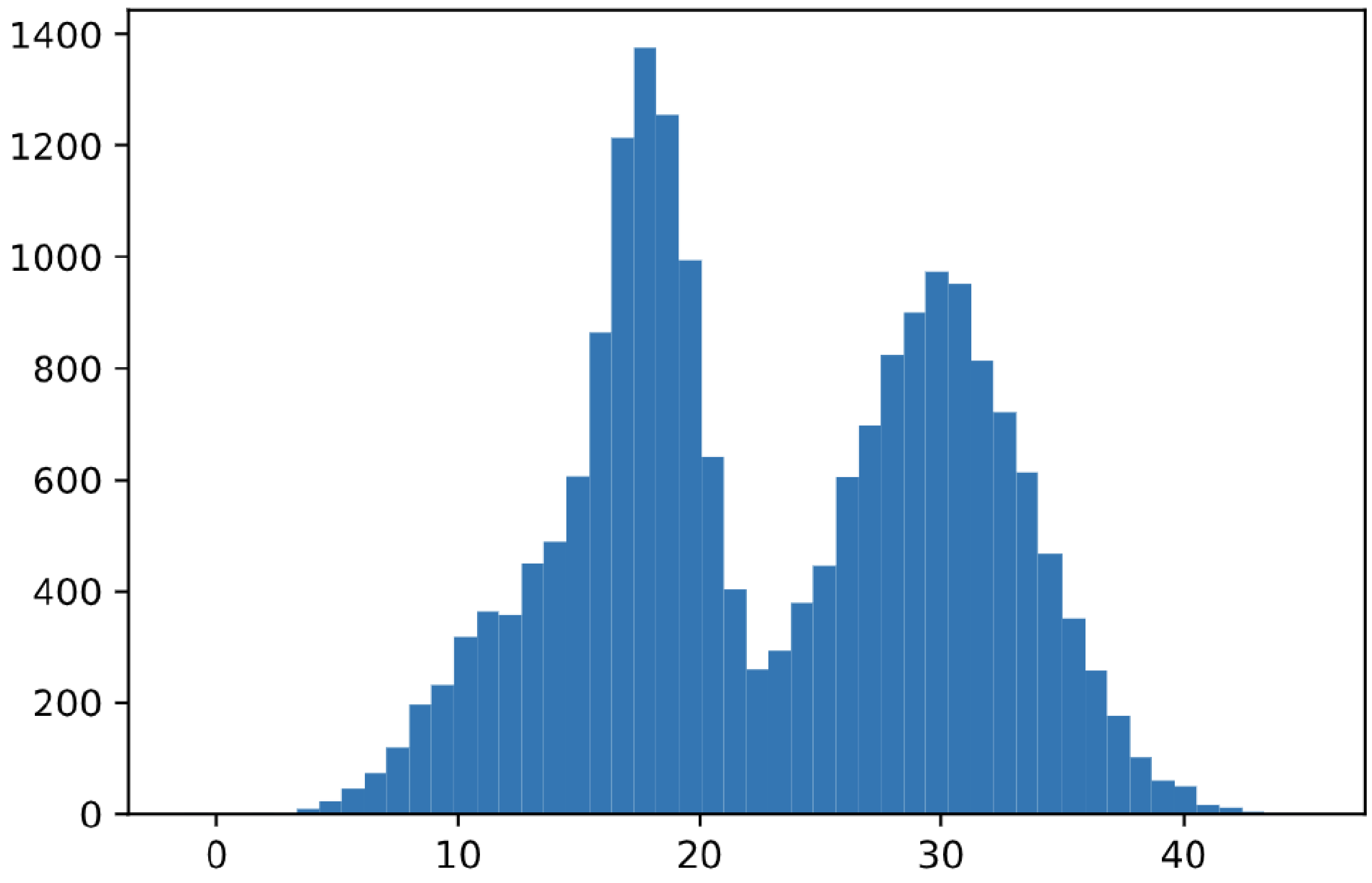
1D: Pie Chart

	Data
Nominal	✓
Ordinal	✓
Interval	✗
Ratio	✗



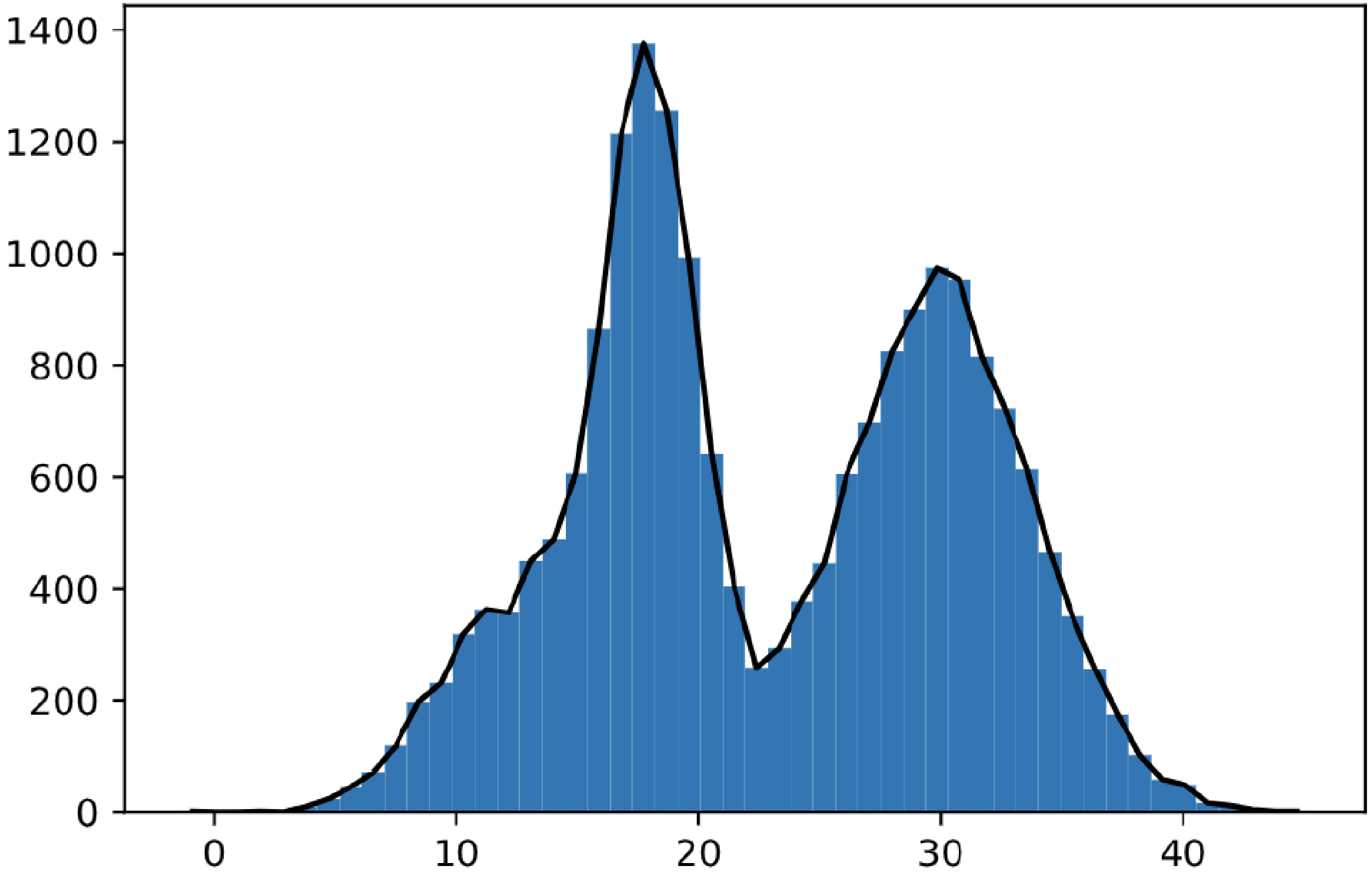
1D: Histogram

	Data
Nominal	✗
Ordinal	✗
Interval	✓
Ratio	✓



1D: Histogram

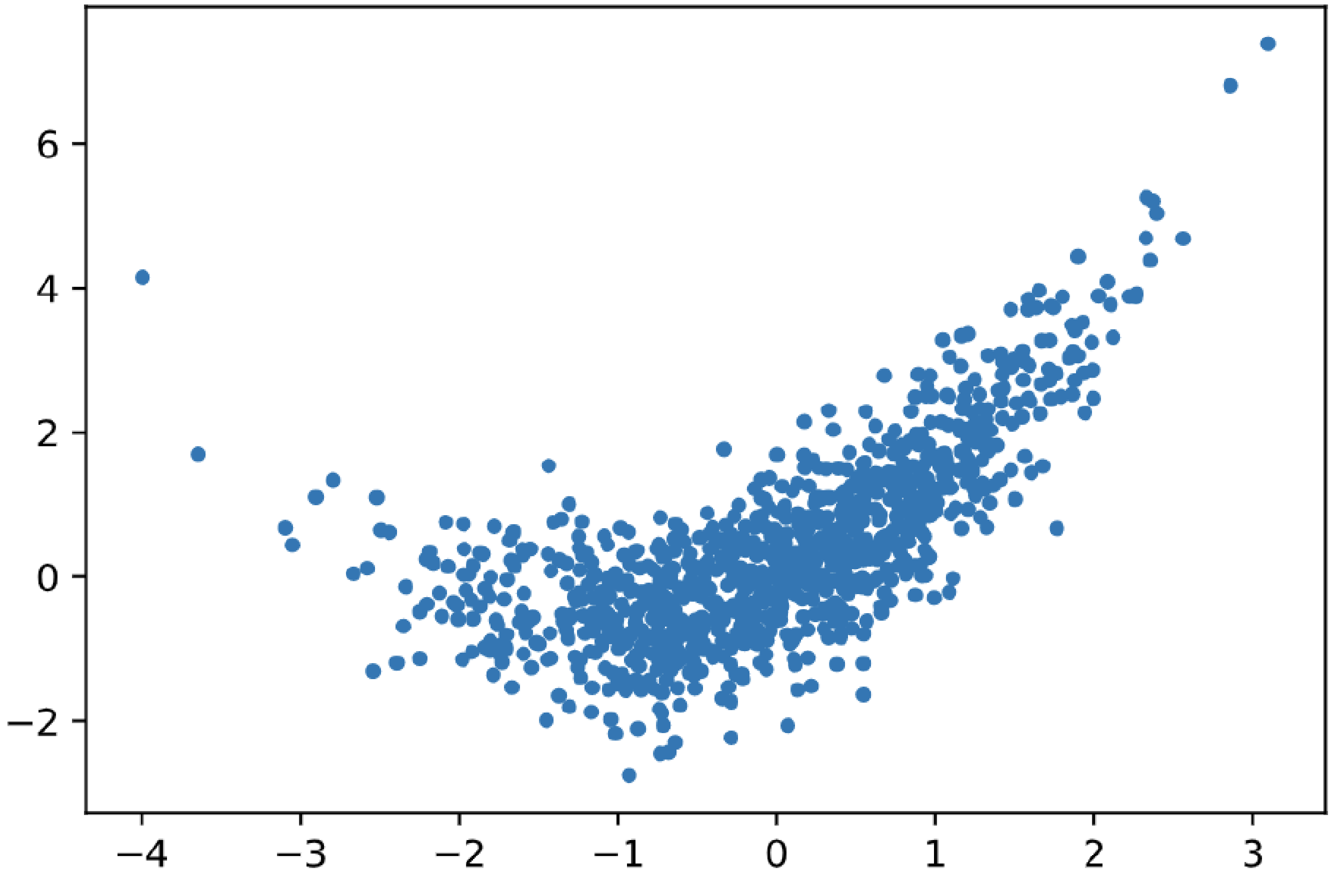
- OK to use lines within a histogram (but not very informative)



2D: Scatter Plot

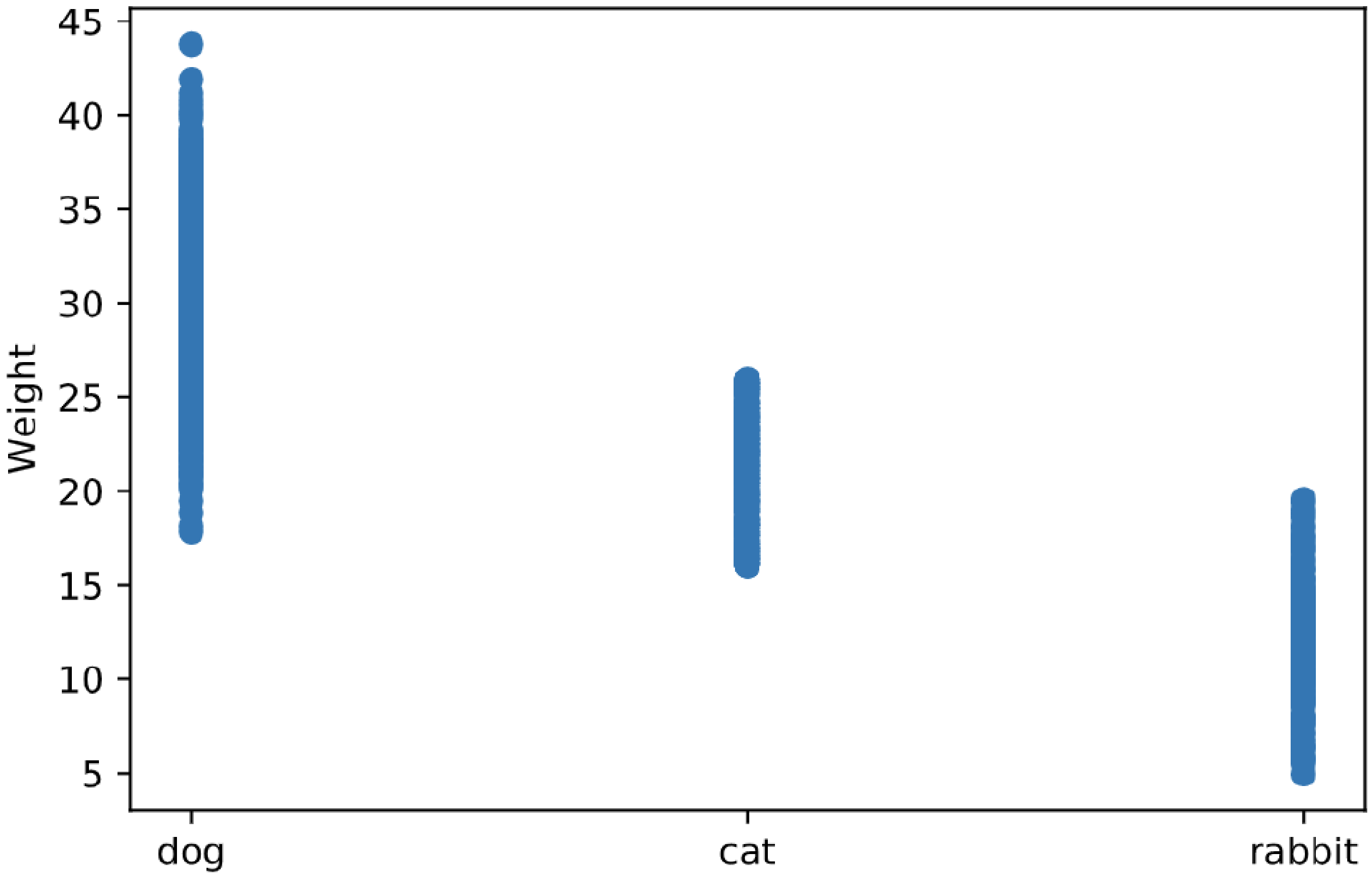
	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval	✓	✓
Ratio	✓	✓

Why not ordinal data in first dimension?



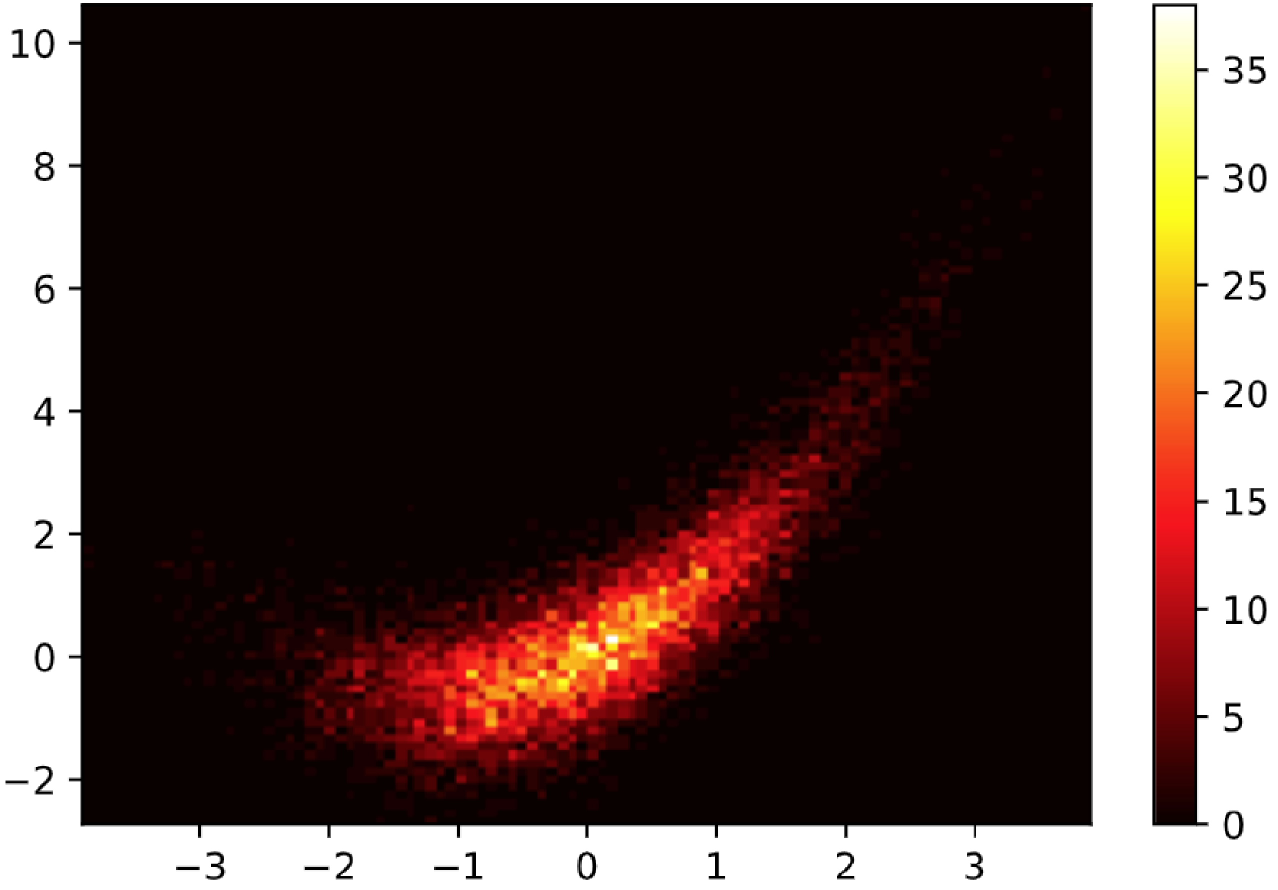
2D: Scatter Plot (bad)

	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval	✓	✓
Ratio	✓	✓



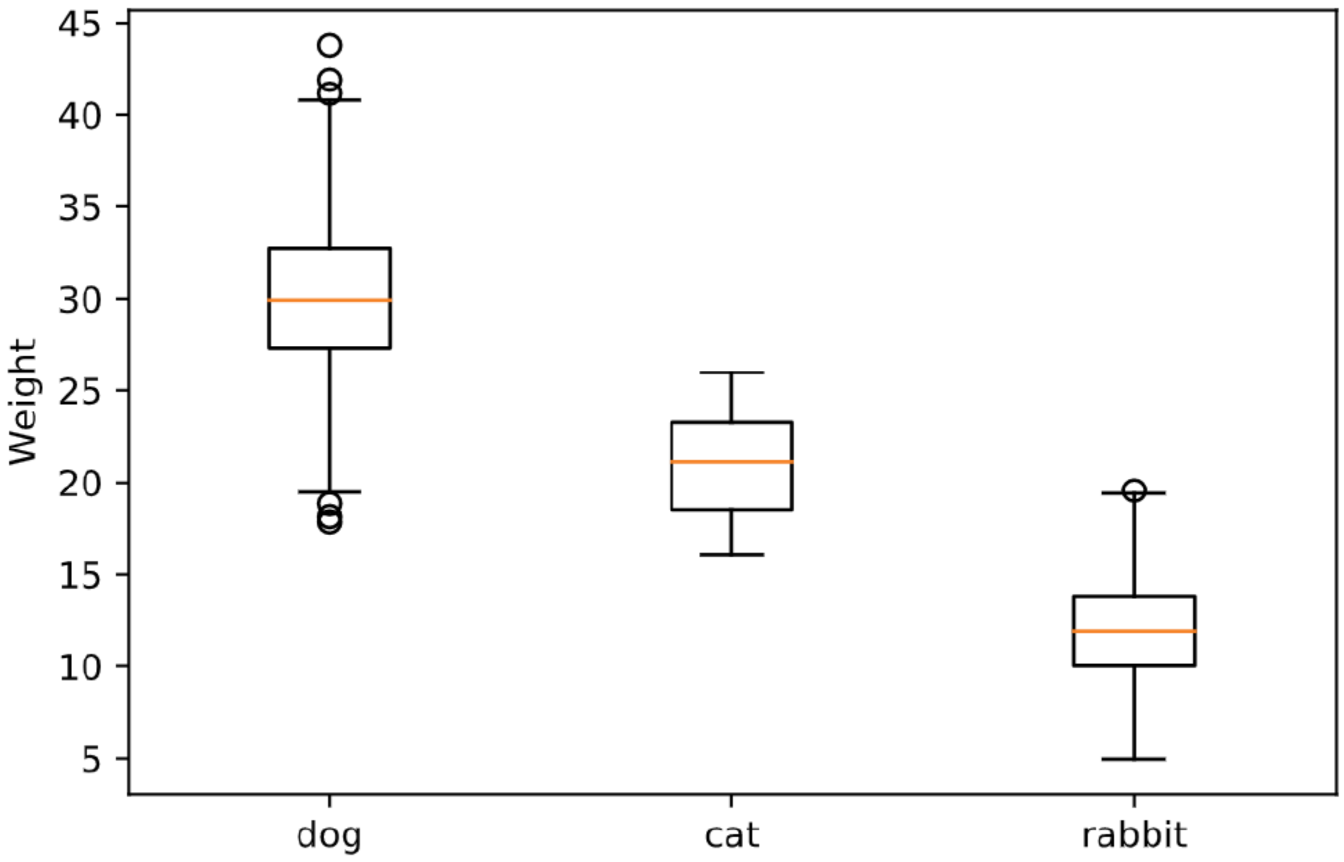
2D: Heatmap (density or 2D histogram)

	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval	✓	✓
Ratio	✓	✓



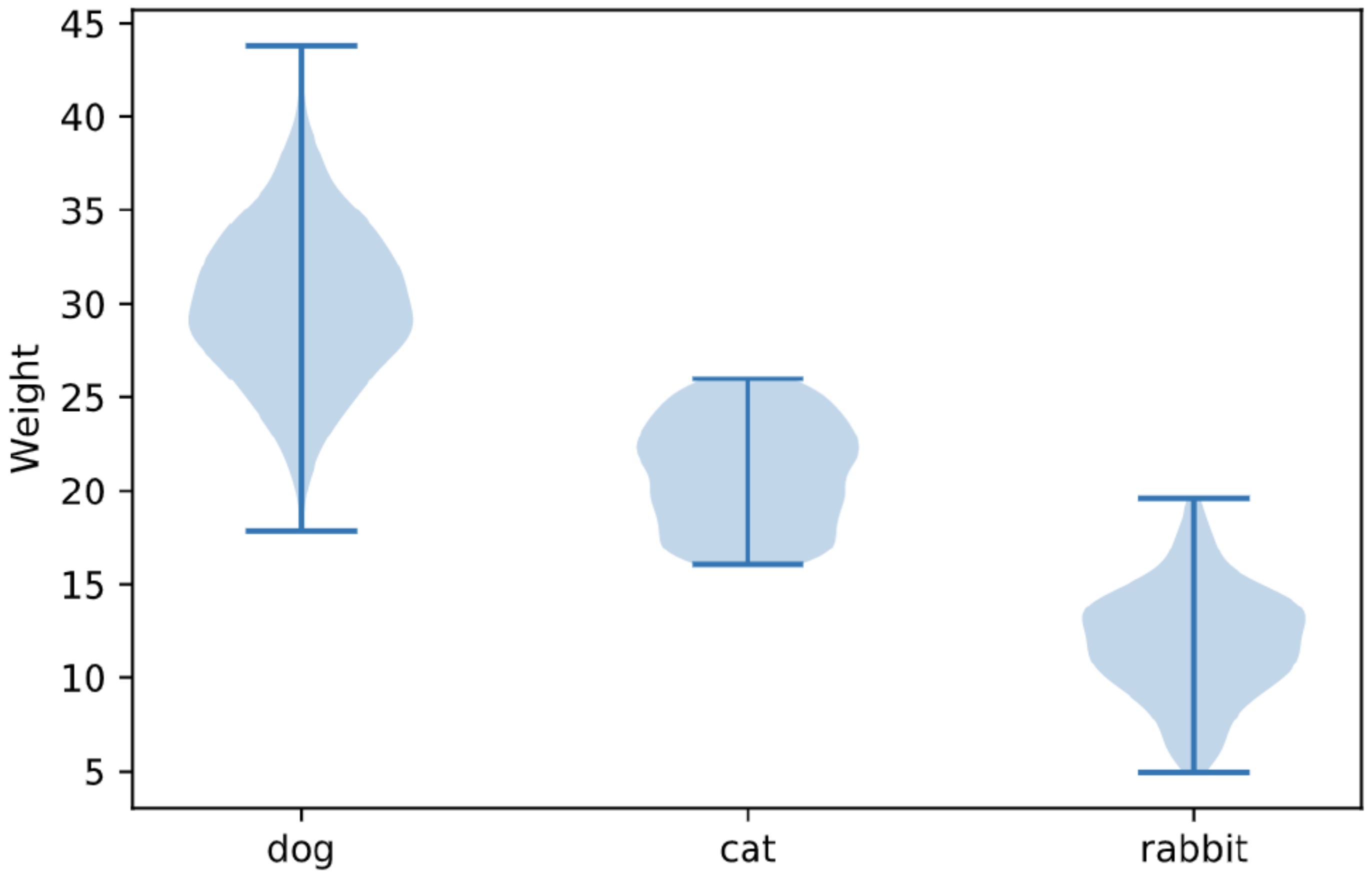
2D: Box and Whiskers

	Dim 1	Dim 2
Nominal	✓	✗
Ordinal	✓	✗
Interval	✗	✓
Ratio	✗	✓



2D: Violin Plot

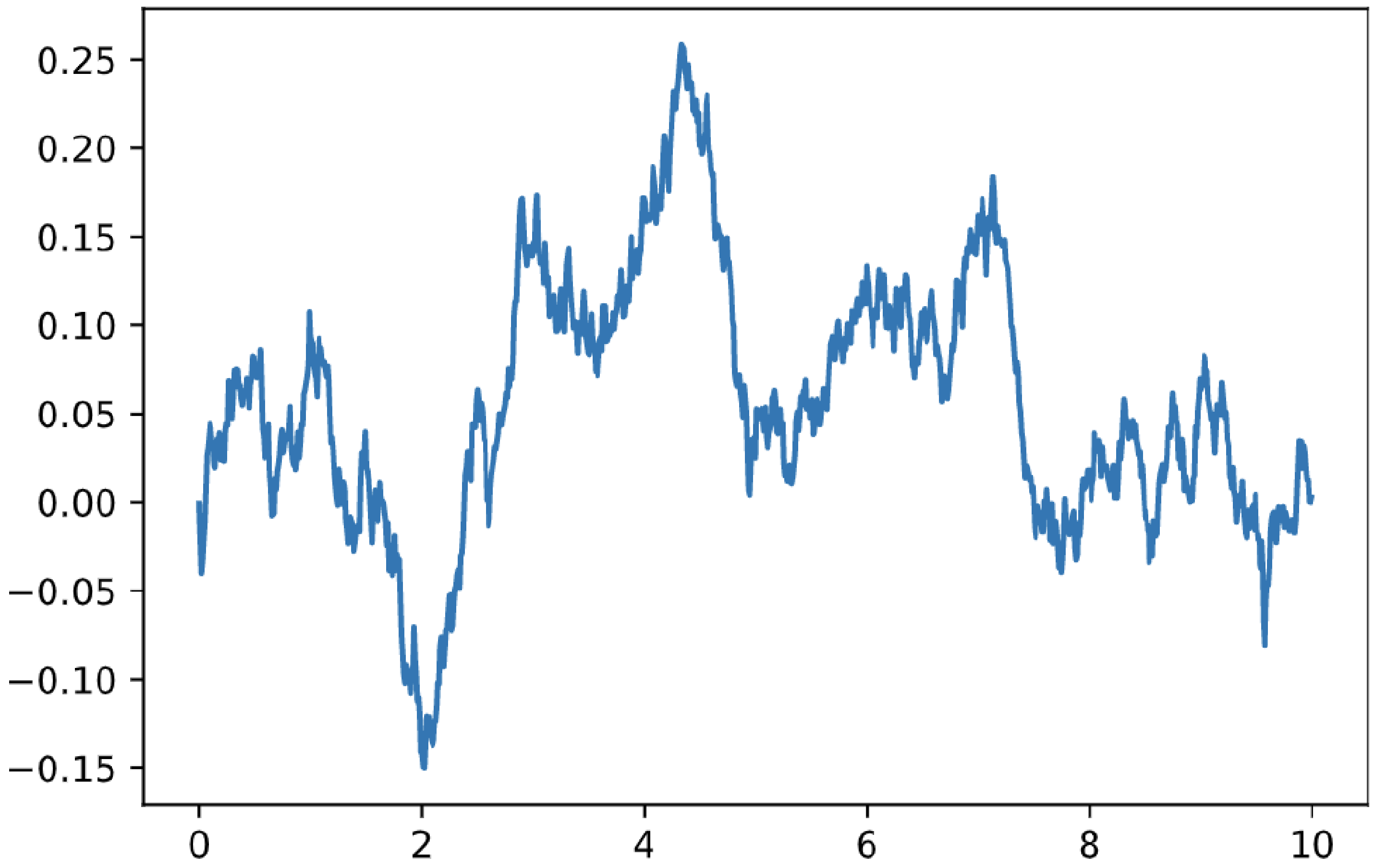
	Dim 1	Dim 2
Nominal	✓	✗
Ordinal	✓	✗
Interval	✗	✓
Ratio	✗	✓



2D: Line Plot

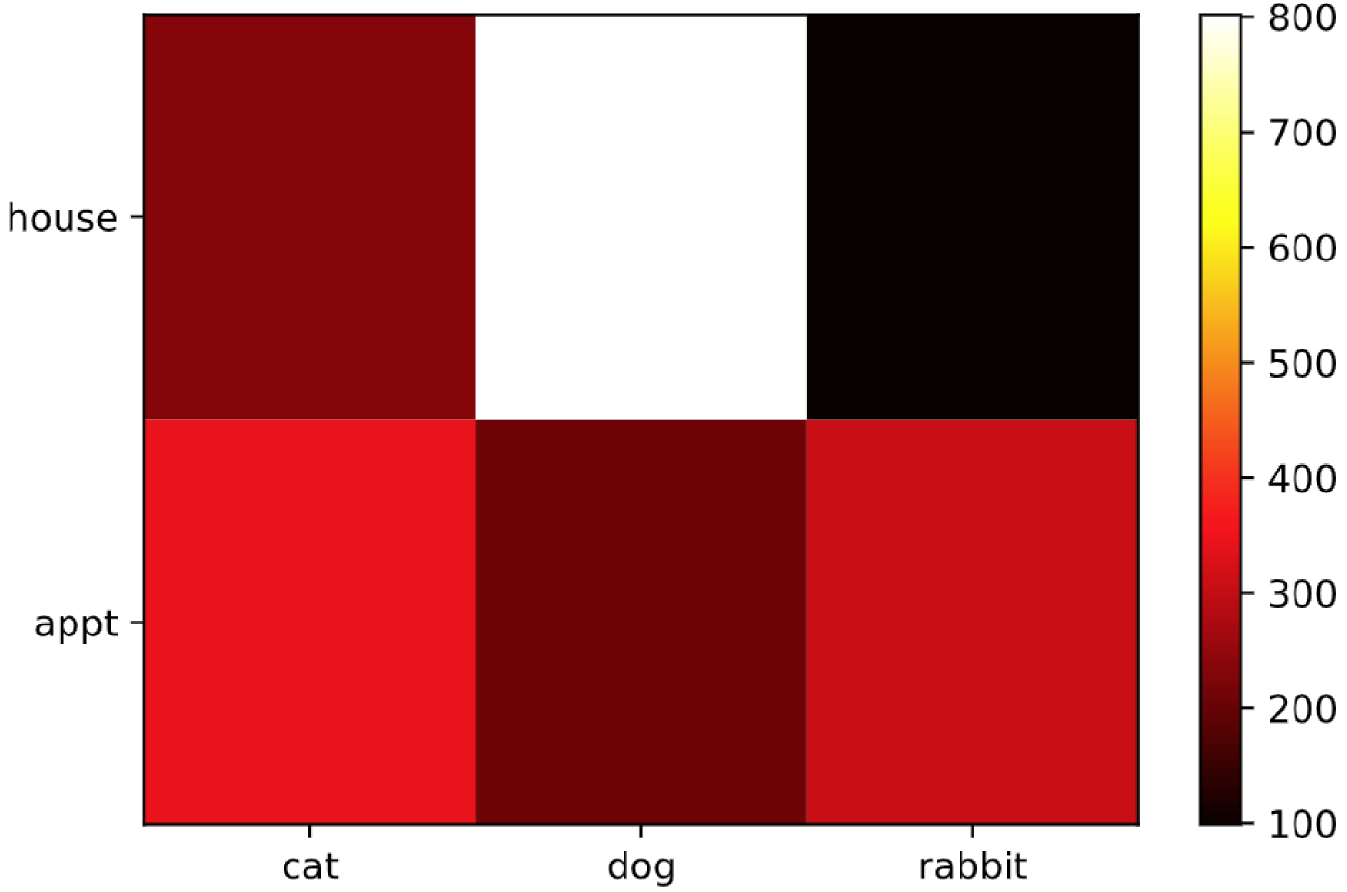
	Dim 1	Dim 2
Nominal	✗	✗
Ordinal	✗	✗
Interval	✓	✓
Ratio	✓	✓

Why not ordinal data in first dimension?



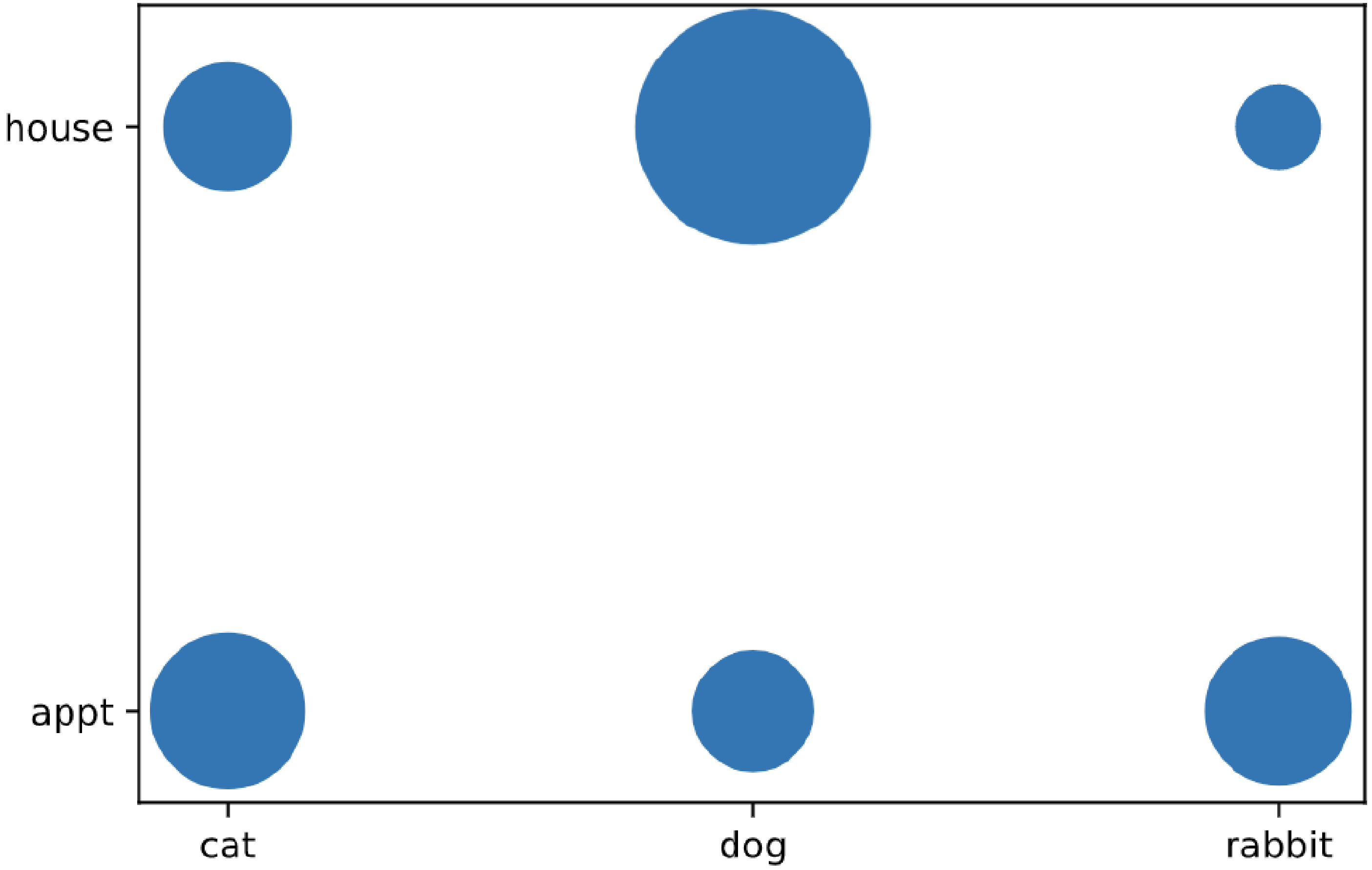
2D: Heatmap (matrix)

	Dim 1	Dim 2
Nominal	✓	✓
Ordinal	✓	✓
Interval	✗	✗
Ratio	✗	✗



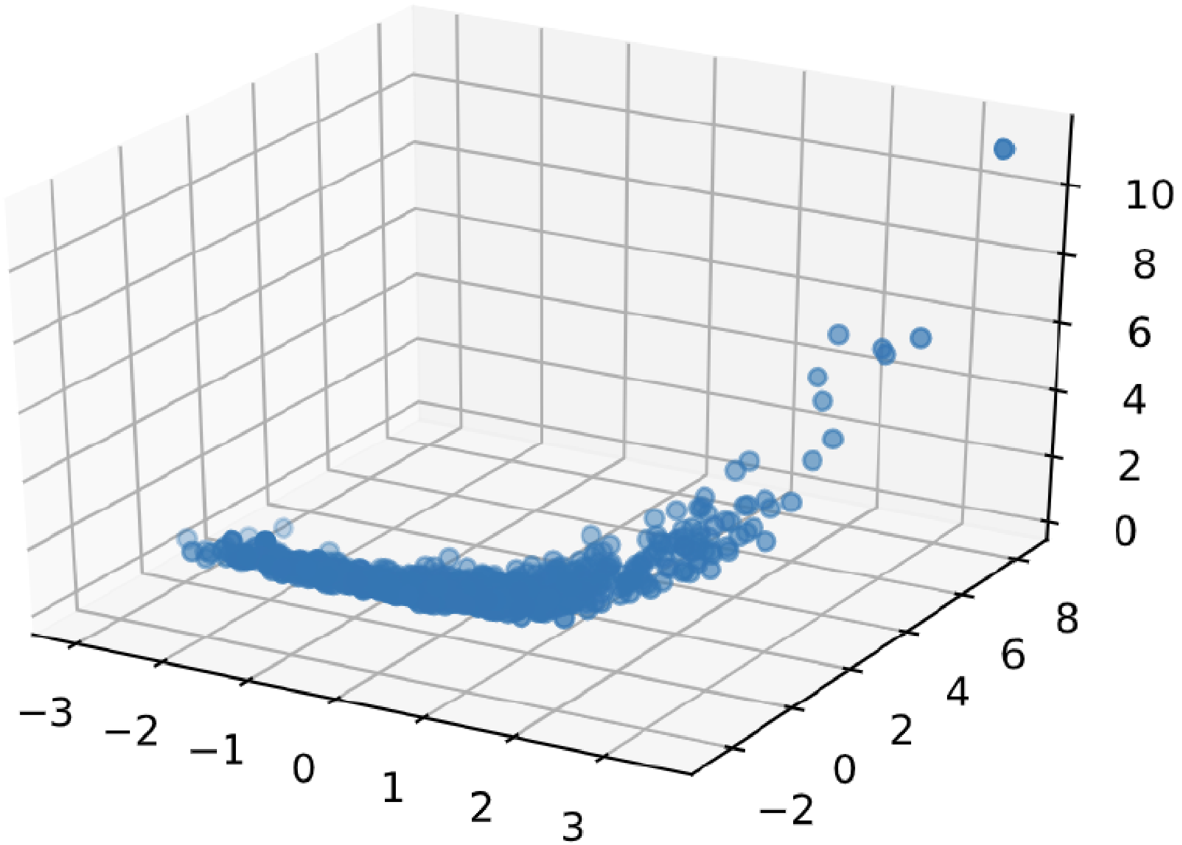
2D: Bubble Plot

	Dim 1	Dim 2
Nominal	✓	✓
Ordinal	✓	✓
Interval	✗	✗
Ratio	✗	✗



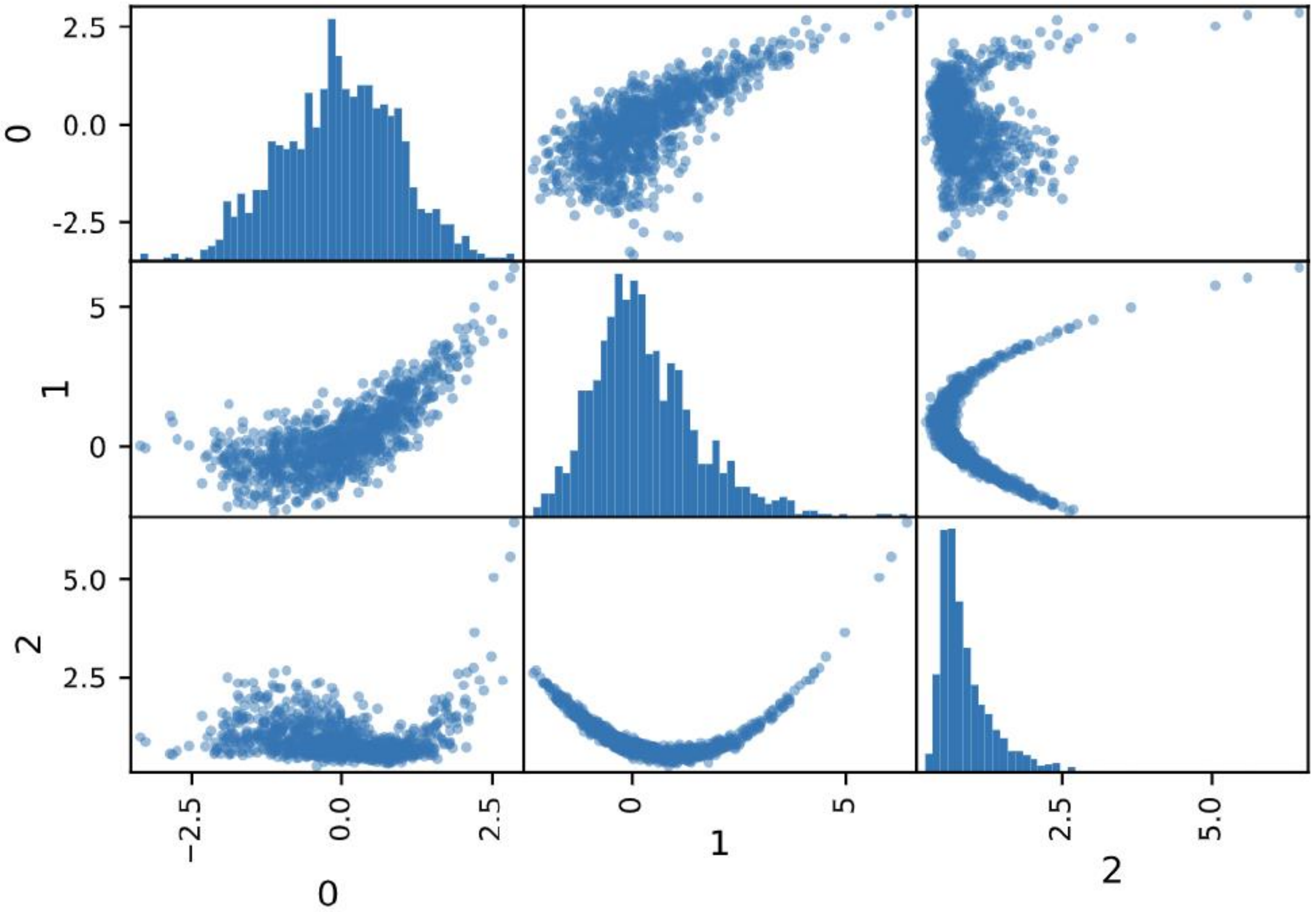
3D+: 3D Scatter Plot

	Dim 1	Dim 2	Dim 3
Nominal	✗	✗	✗
Ordinal	✗	✗	✗
Interval	✓	✓	✓
Ratio	✓	✓	✓



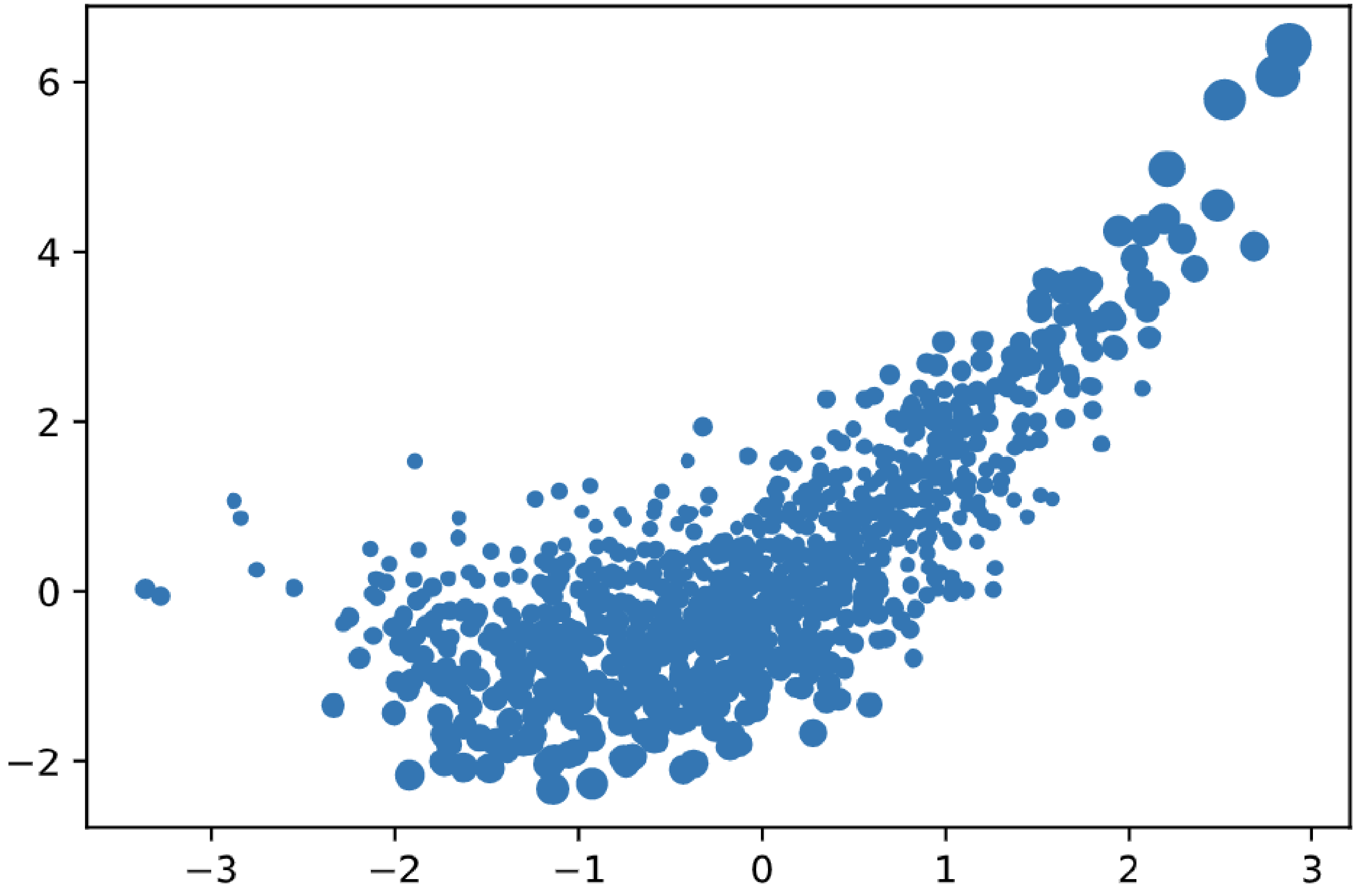
3D+: Scatter Plot Matrix

	Dim 1	Dim 2	Dim 3
Nominal	✗	✗	✗
Ordinal	✗	✗	✗
Interval	✓	✓	✓
Ratio	✓	✓	✓



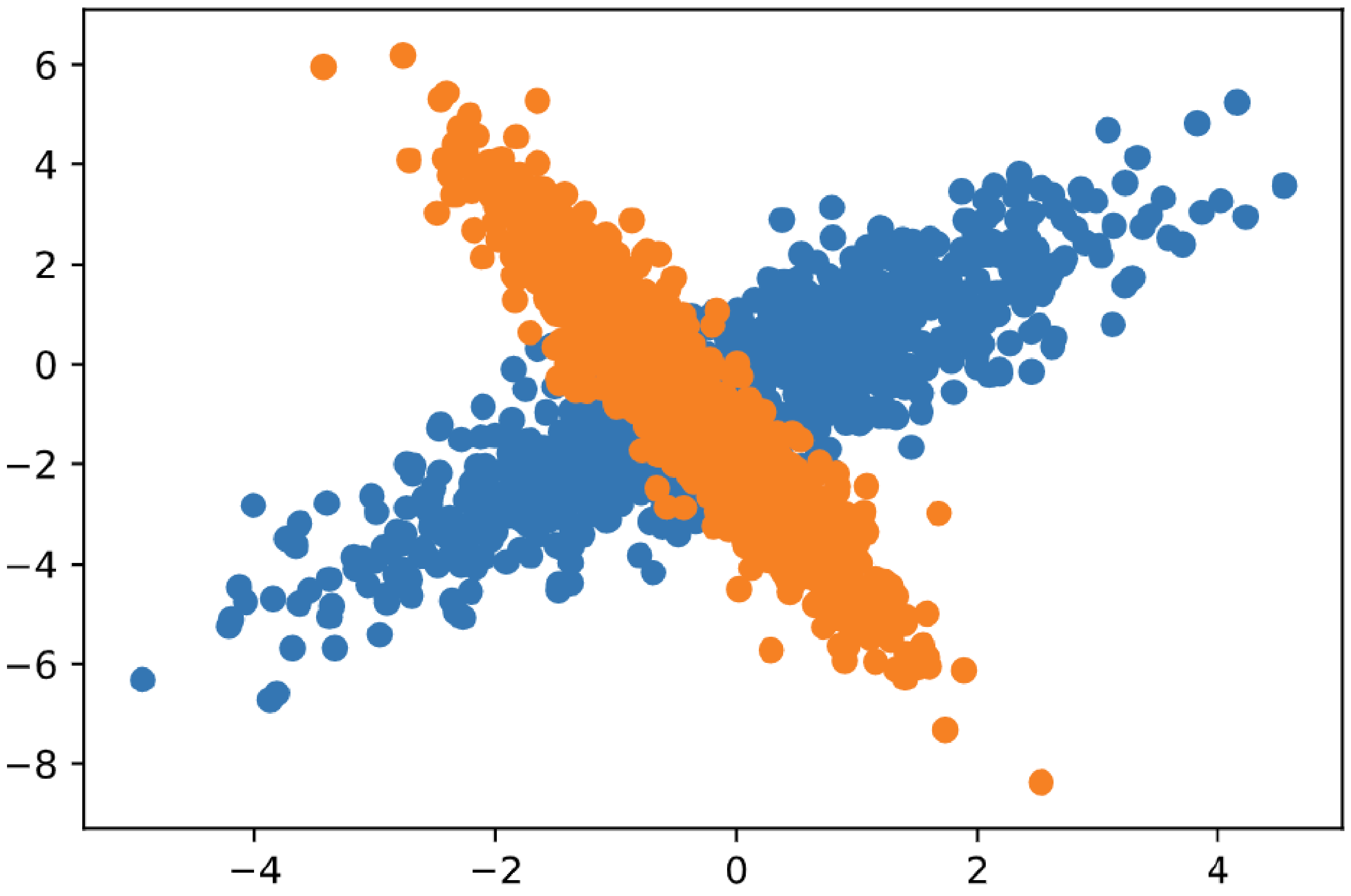
3D+: Bubble Plot

	Dim 1	Dim 2	Dim 3
Nominal	✗	✗	✗
Ordinal	✗	✗	✗
Interval	✓	✓	✓
Ratio	✓	✓	✓



3D+: Color Scatter Plot

	Dim 1	Dim 2	Dim 3
Nominal	✗	✗	✓
Ordinal	✗	✗	✓
Interval	✓	✓	✗
Ratio	✓	✓	✗

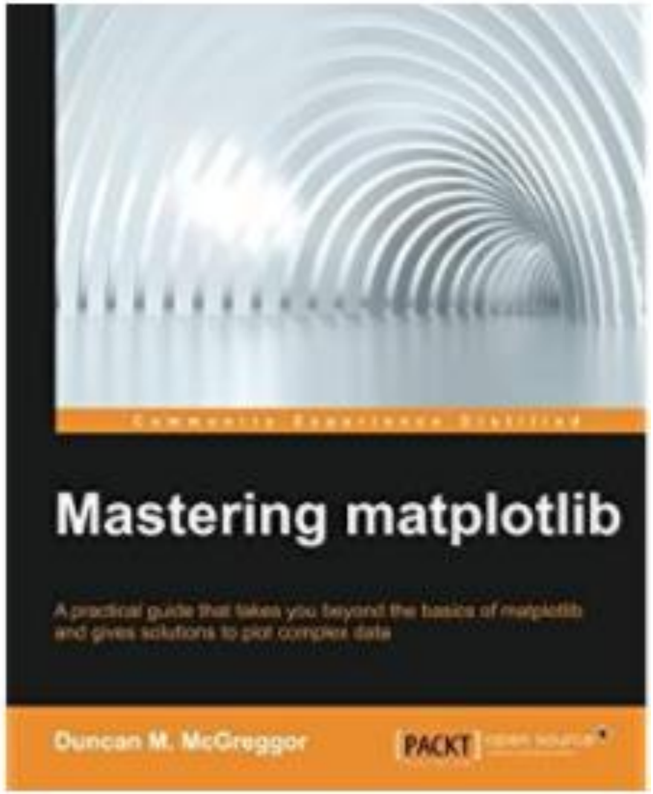


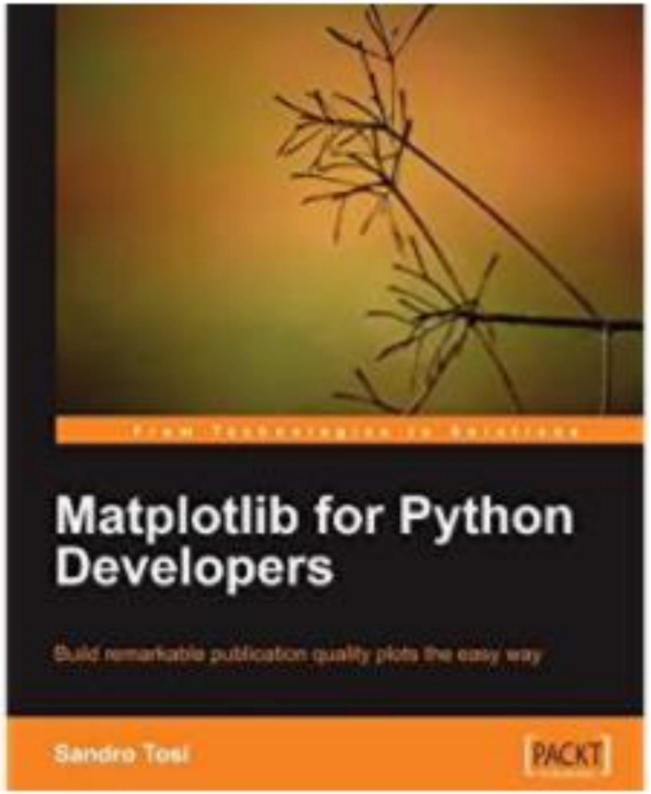
Matplotlib

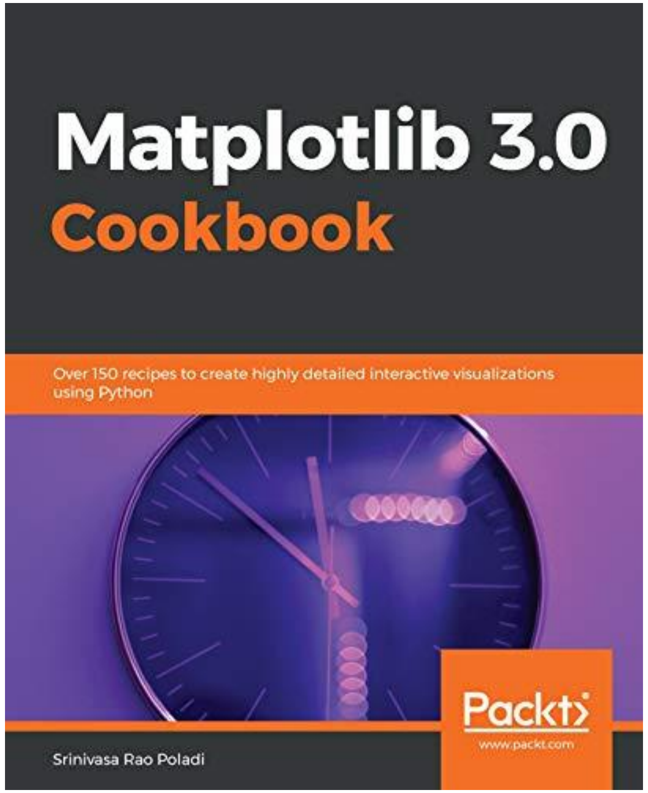
- Matplotlib is the standard for plotting in Python / Jupyter Notebook
- Matplotlib used to generate fairly ugly plots by default, but in recent versions this is no longer the case, so minimal need for additional libraries
- It is aimed at generating static plots, not very good for interacting with data (with a few exceptions)
- A number of additional libraries provide some level of interactive plot (and static plots), but matplotlib is enough of a standard

What is MatplotLib?

Many Matplotlib Books







What is "Matplotlib" Module?

- Plotting library for Python
  - Reliant on NumPy

- MATLAB-style plotting interface with hierarchical organized elements
  - pyplot module: provides state-machine environment at the top of hierarchy

- Open source (http://matplotlib.org)
  - Original author: John D. Hunter (2003) → Michael Droettboom (2012)

- pyplot tutorials
  - https://matplotlib.org/users/pyplot_tutorial.html

Overview

- matplotlib submodules
  - pyplot: provides MATLAB-like plotting framework
    - Closely related with actual plotting functionality
  - image: image loading, rescaling and display operations
  - colors: converting numbers or color arguments to RGB or RGBA
  - cm(colormap): provides function for color mapping functionality
  - collections: for efficient drawing of large collections

- We'll focus on pyplot module because it is in charge of creating plotting

- Other submodules support better plotting environment

Matplotlib Submodule List

- matplotlib.afm
- matplotlib.animation
- matplotlib.artist
- matplotlib.axes
- matplotlib.axis
- matplotlib.backend_bases
- matplotlib.backend_managers
- matplotlib.backend_tools
- matplotlib.backends.backend_agg
- matplotlib.backends.backend_cairo
- matplotlib.backends.backend_mixed
- matplotlib.backends.backend_nbagg
- matplotlib.backends.backend_pdf
- matplotlib.backends.backend_pgf
- matplotlib.backends.backend_ps
- matplotlib.backends.backend_svg
- matplotlib.backends.backend_tkagg

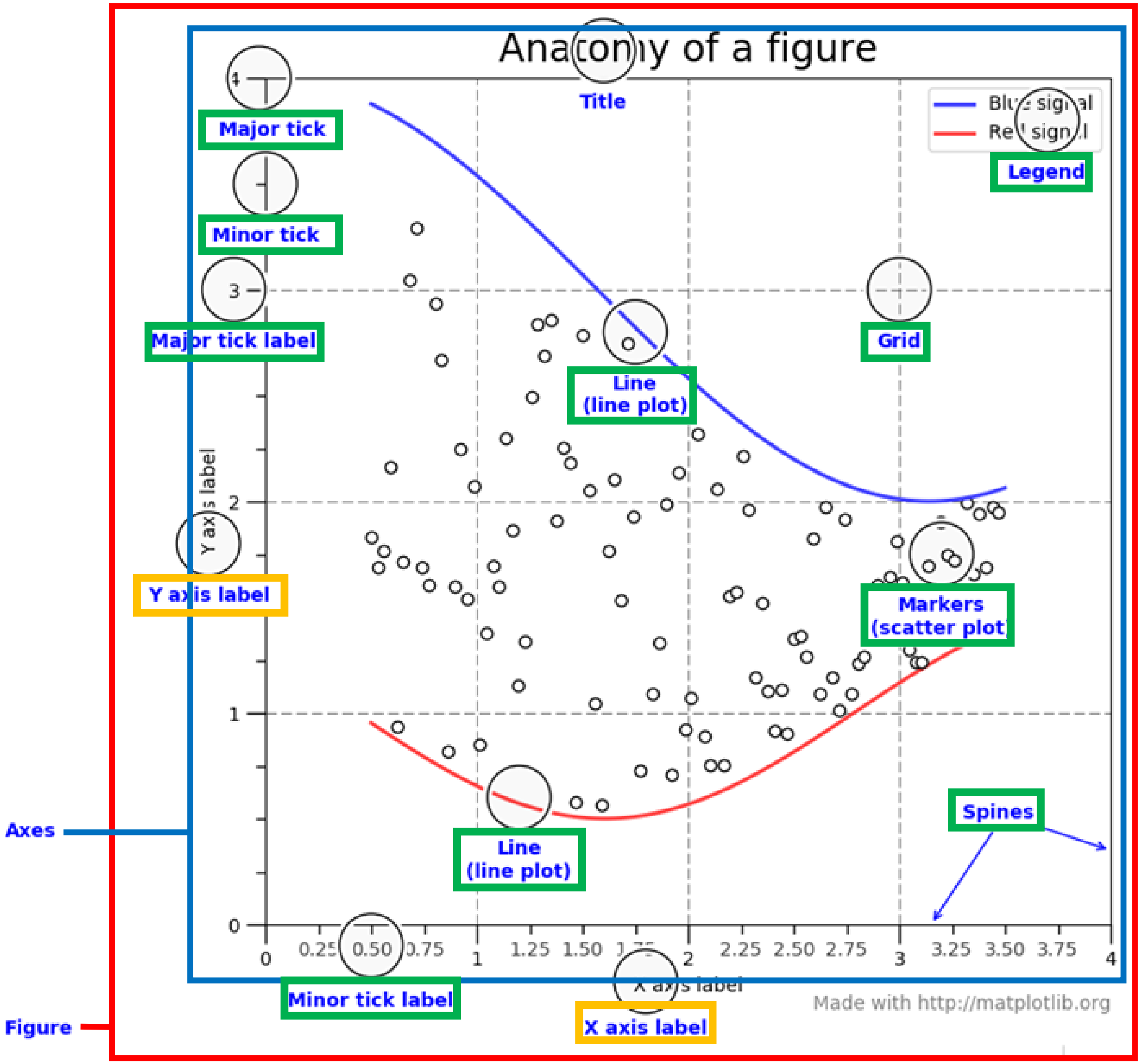
- matplotlib.blocking_input
- matplotlib.category
- matplotlib.cbook
- matplotlib.cm
- matplotlib.collections
- matplotlib.colorbar
- matplotlib.colors
- matplotlib.container
- matplotlib.contour
- matplotlib.dates
- matplotlib.dviread
- matplotlib.figure
- matplotlib.font_manager
- matplotlib.fontconfig_pattern
- matplotlib.gridspec
- matplotlib.image
- matplotlib.legend

- matplotlib.legend_handler
- matplotlib.lines
- matplotlib.markers
- matplotlib.mathtext
- matplotlib.mlab
- matplotlib.offsetbox
- matplotlib.patches
- matplotlib.path
- matplotlib.patheffects
- matplotlib.projections
- matplotlib.projections.polar
- matplotlib.pyplot
- matplotlib.rcsetup
- matplotlib.sankey
- matplotlib.scale
- matplotlib.sphinxext.plot_di
- matplotlib.spines

- matplotlib.style
- matplotlib.table
- matplotlib.testing
- matplotlib.testing.compare
- matplotlib.testing.decorators
- matplotlib.testing.disable_internet
- ma exceptions
- ma
- ma
- ma
- ma yout
- matplotlib.transforms
- matplotlib.tri
- matplotlib.type1font
- matplotlib.units
- matplotlib.widgets

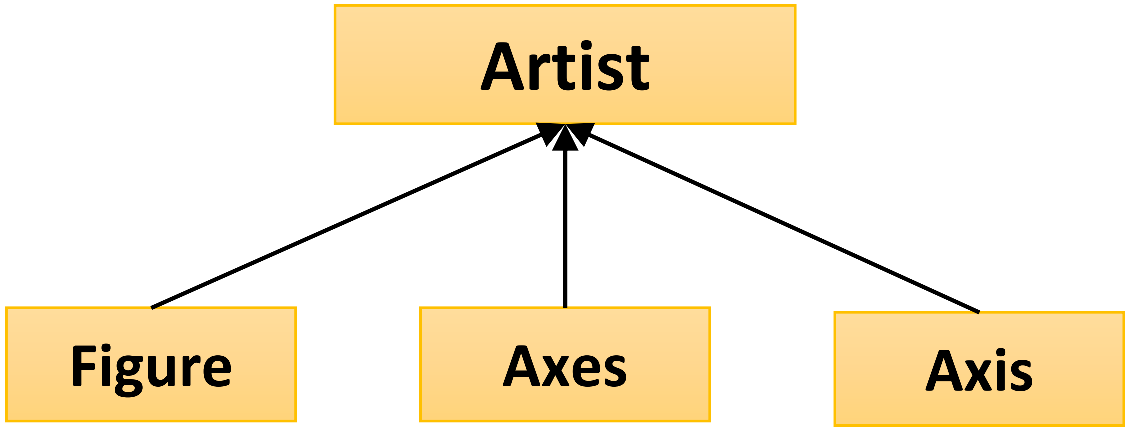
plot()
bar()
hist()
boxplot()
subplot()
...

Pyplot Plotting Example



Matplotlib Classes

- 4 important classes in matplotlib



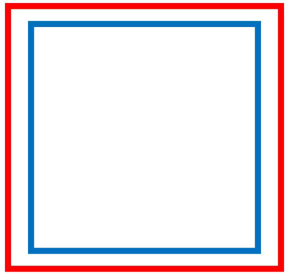
- Artist
  - Basically everything you can see on the figure
- Figure
  - Consider as a window
  - A figure may contain several child Axes and keep track of them
  - Several figures can be created
- Axes: region of an image or a plot
- Axis: strings labeling ticks

Using Pyplot

For jupyter notebook:
plot graphs without show()

- Import library

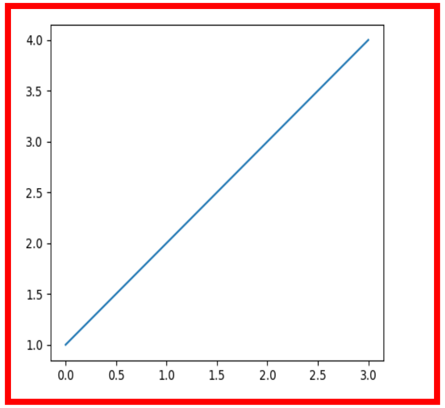
```
%matplotlib inline
import matplotlib.pyplot as plt
```



  - Pyplot uses 1 Figure with 1 Axes by default and use it as current Axes
  - Each objects can be added as needed

- Plot on current Axes

```
plt.plot([1, 2, 3, 4])
```



  - Automatically set the x, y ranges and tick labels

- Display figure

```
plt.show()
```

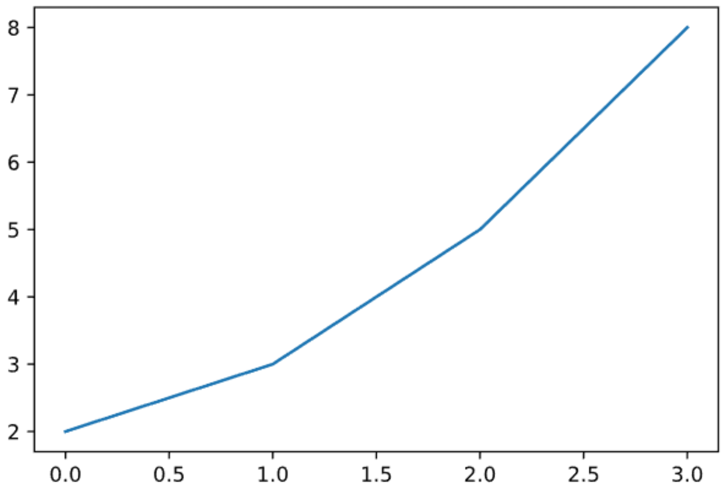
  - For actual display of figures in pyplot, additional statement is necessary

Pyplot Basics

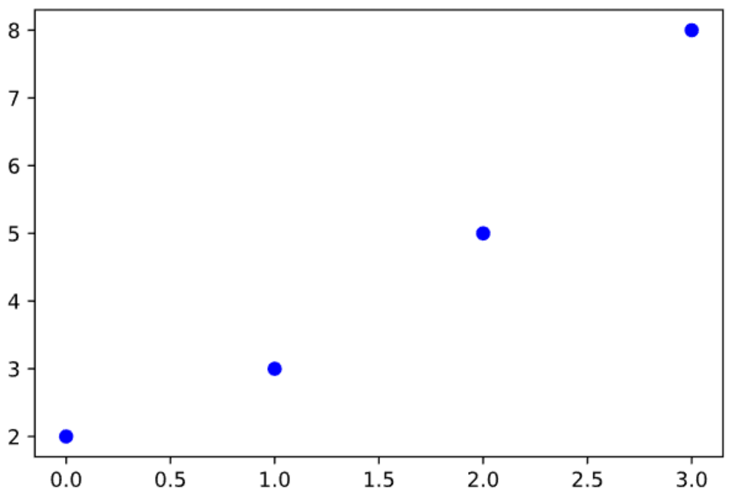
Line and Scatter Plot

- *plt*.`plot`([*x*], *y*, [*fmt*], [*x2*], *y2*, [*fmt2*], ...)
  - The coordinates of the points or line nodes given by *x*, *y*
  - If *x* is omitted, index array [0, 1, ..., *N*-1] is used as *x*
  - *fmt*: defines basic formatting like color, marker and linestyle (default: blue line graph)

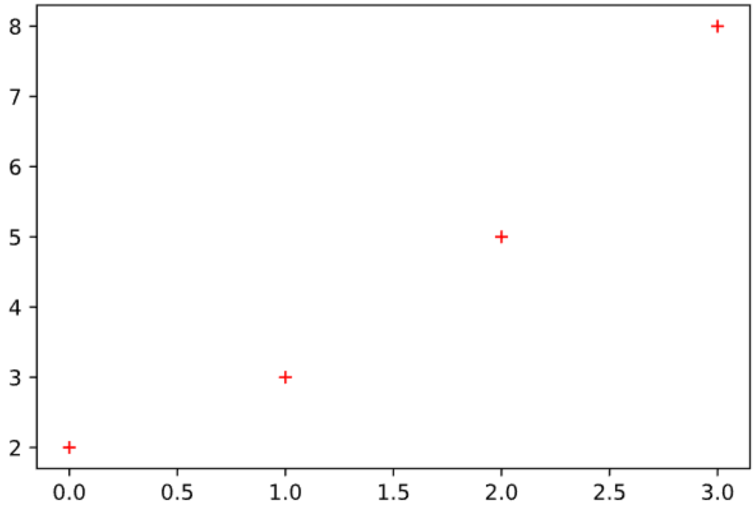
```
plt.plot([2,3,5,8])
```




```
plt.plot([2,3,5,8], 'bo')
```




```
plt.plot([2,3,5,8], 'r+')
```



Markers

'.'	point marker	's'	square marker
','	pixel marker	'p'	pentagon marker
'o'	circle marker	'*'	star marker
'v'	triangle_down marker	'h'	hexagon1 marker
'^'	triangle_up marker	'H'	hexagon2 marker
'<'	triangle_left marker	'+'	plus marker
'>'	triangle_right marker	'x'	x marker
'1'	tri_down marker	'D'	diamond marker
'2'	tri_up marker	'd'	thin_diamond marker
'3'	tri_left marker	'|'	vline marker
'4'	tri_right marker	'_'	hline marker

Line Styles and Colors

`'-'`	solid line style
`'--'`	dashed line style
`'-.'`	dash-dot line style
`':'`	dotted line style

`'b'`	blue
`'g'`	green
`'r'`	red
`'c'`	cyan
`'m'`	magenta
`'y'`	yellow
`'k'`	black
`'w'`	white

```
fmt = '[marker][line][color]'
```

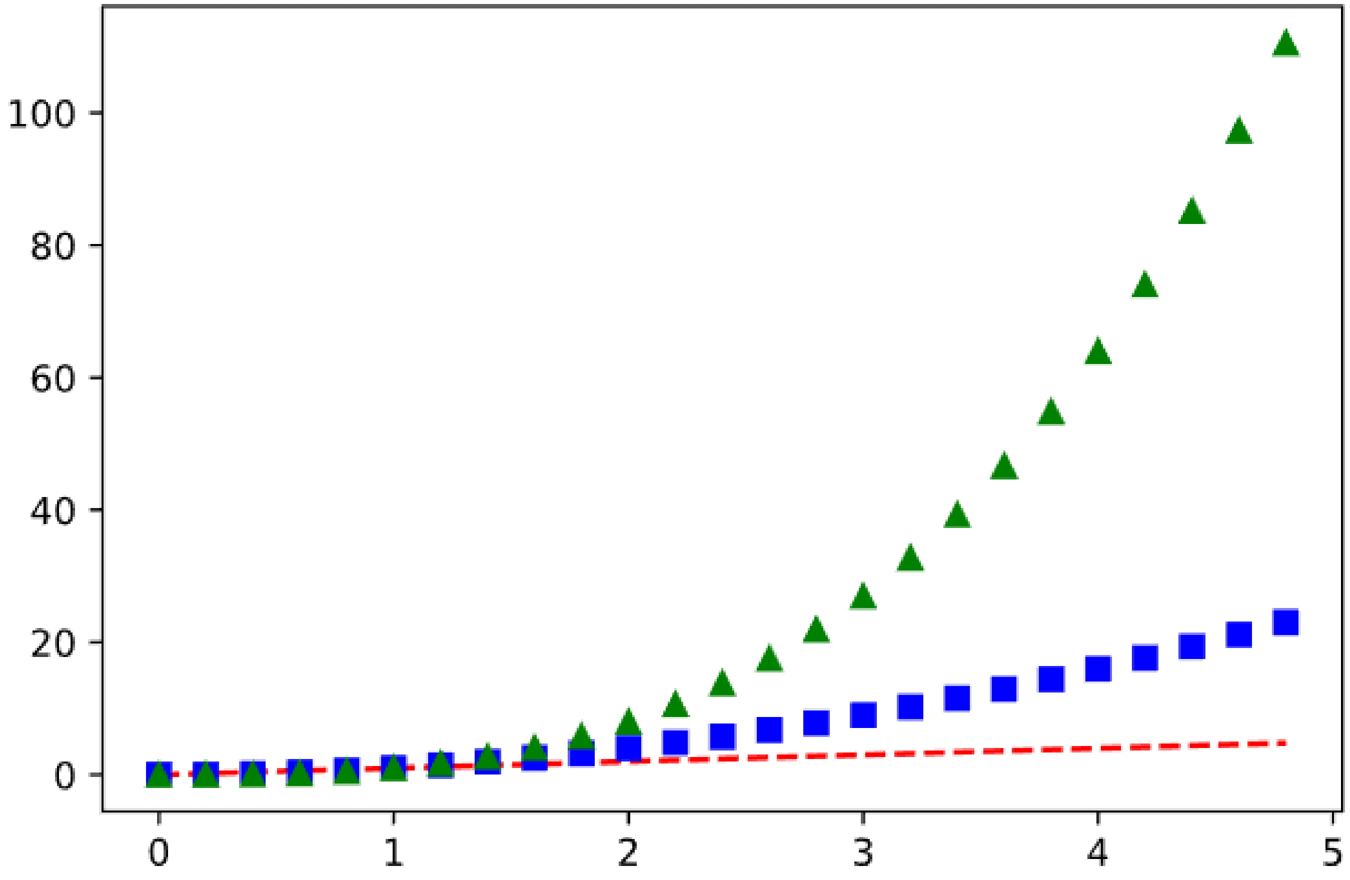
(Same color for line and marker)

Plotting Multiple Graphs

- Plotting 3 different data at once

```
t = np.arange(0.0, 5.0, 0.2)
plt.plot(t, t, 'r--',
         t, t**2, 'bs',
         t, t**3, 'g^')
```

- `'r--'`: red dashed
- `'bs'`: blue square
- `'g^'`: green triangle



More Examples

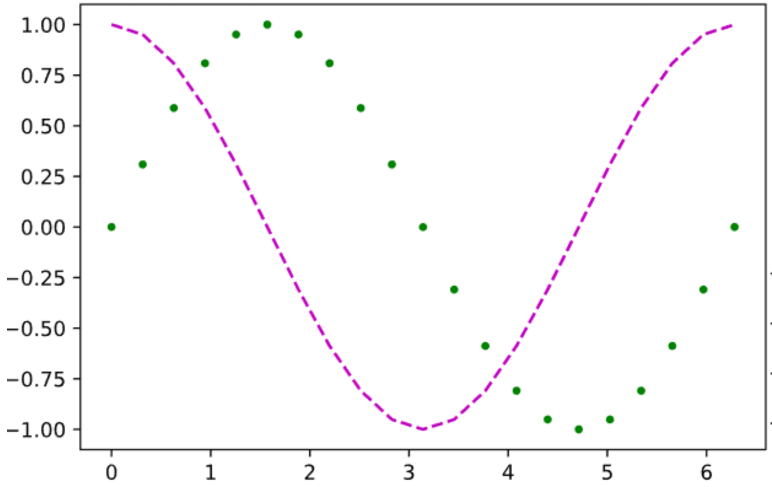
```
x = np.linspace(0, 2*np.pi, 21)
y = np.sin(x)
y2 = np.cos(x)
```

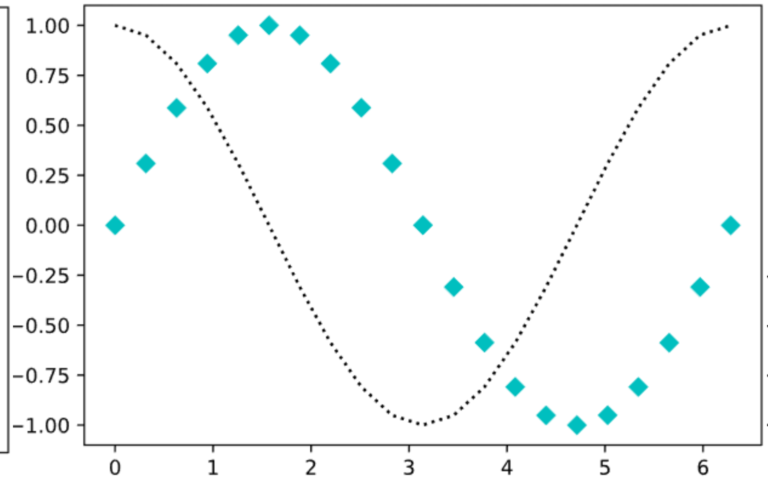
Choose colors automatically

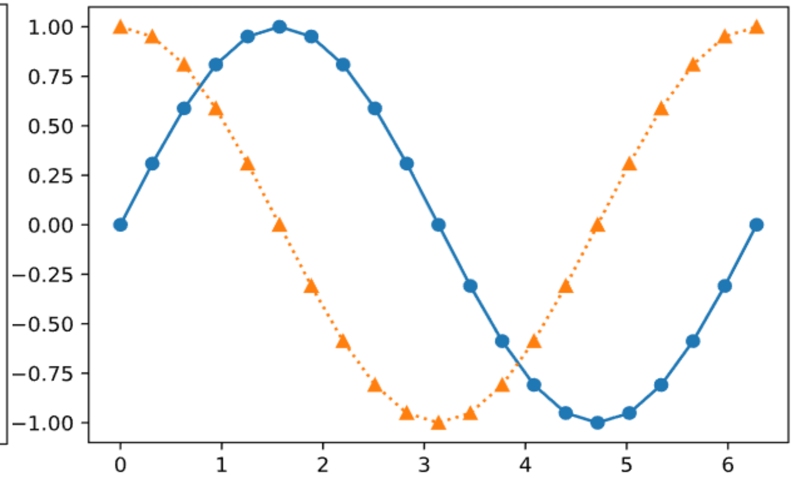
```
plt.plot
(x, y, 'g.', x, y2, 'm--')
```

```
plt.plot
(x, y, 'cD', x, y2, 'k-.')
```

```
plt.plot
(x, y, '-o', x, y2, '^:')
```



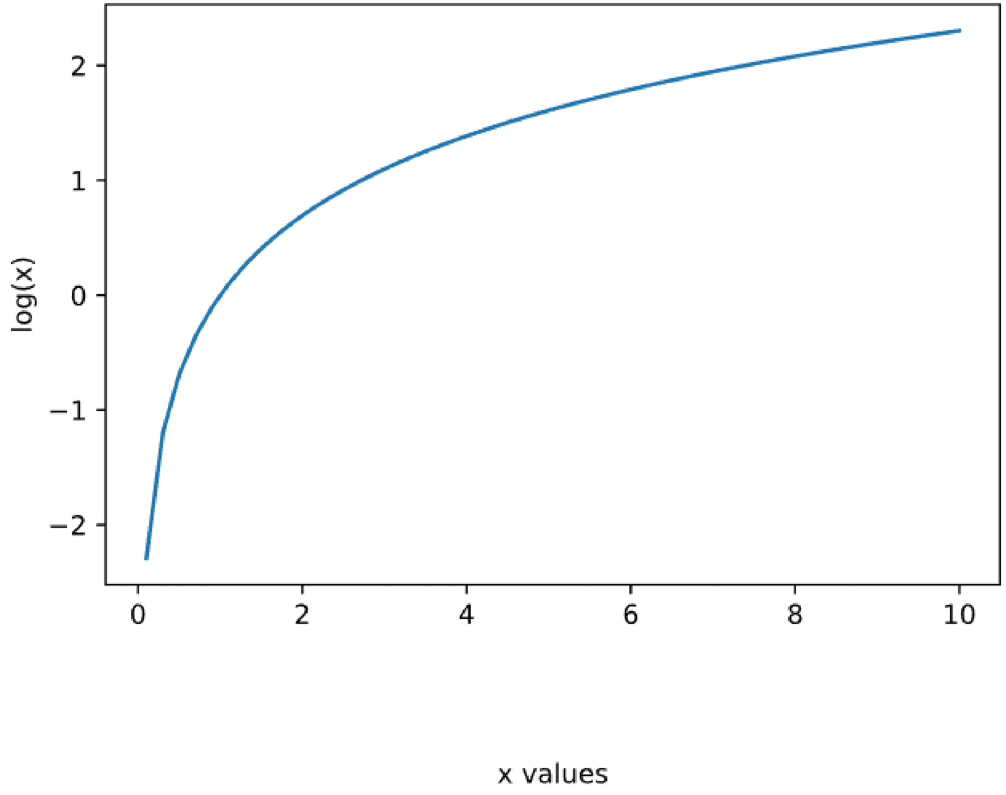




Axis Labels

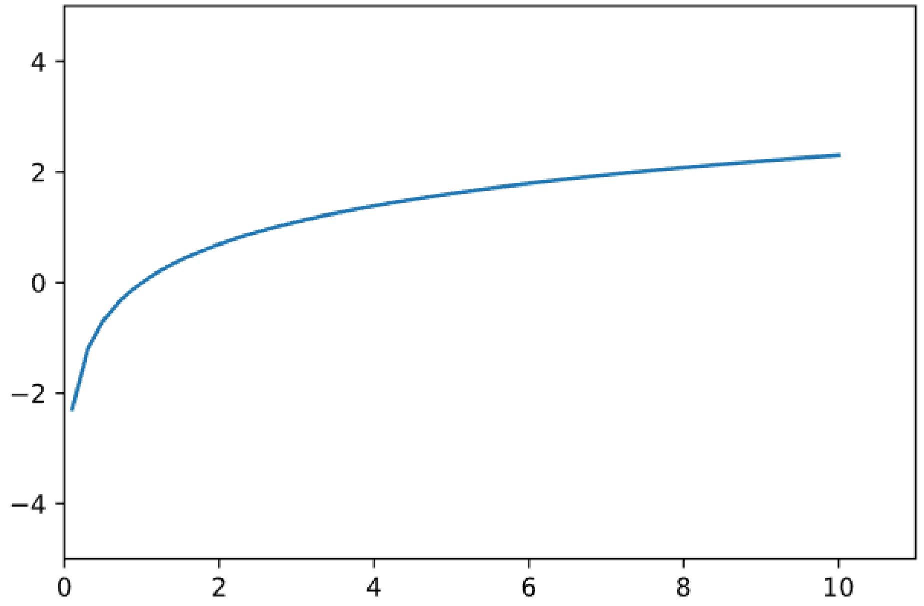
- *plt.*`xlabel`(*xlabel*, [*labelpad*], ...)
- *plt.*`ylabel`(*ylabel*, [*labelpad*], ...)
  - Set the label for the x-axis (or y-axis)
  - *xlabel*, *ylabel*: string for the label
  - *labelpad*: spacing in points from the axes bounding box including ticks and tick labels

```
x = np.linspace(0.1, 10, 100)
y = np.log(x)
plt.xlabel('x values', labelpad=50)
plt.ylabel('log(x)')
plt.plot(x, y)
```

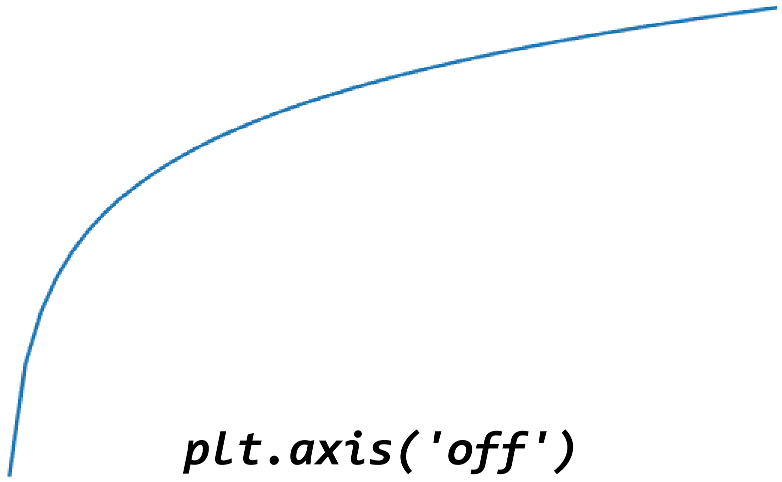


Axis

- *plt*.`axis`(*limits, ..*)
  - Set (or get) some axis properties
  - *limits*: a list with the axis limits to be set.
    `[xmin, xmax, ymin, ymax]`
  - Use `'off'` to hide all the axes




```
x = np.linspace(0.1, 10, 100)
y = np.log(x)
plt.axis([0, 11, -5, 5])
# plt.axis('off')
plt.plot(x, y)
```

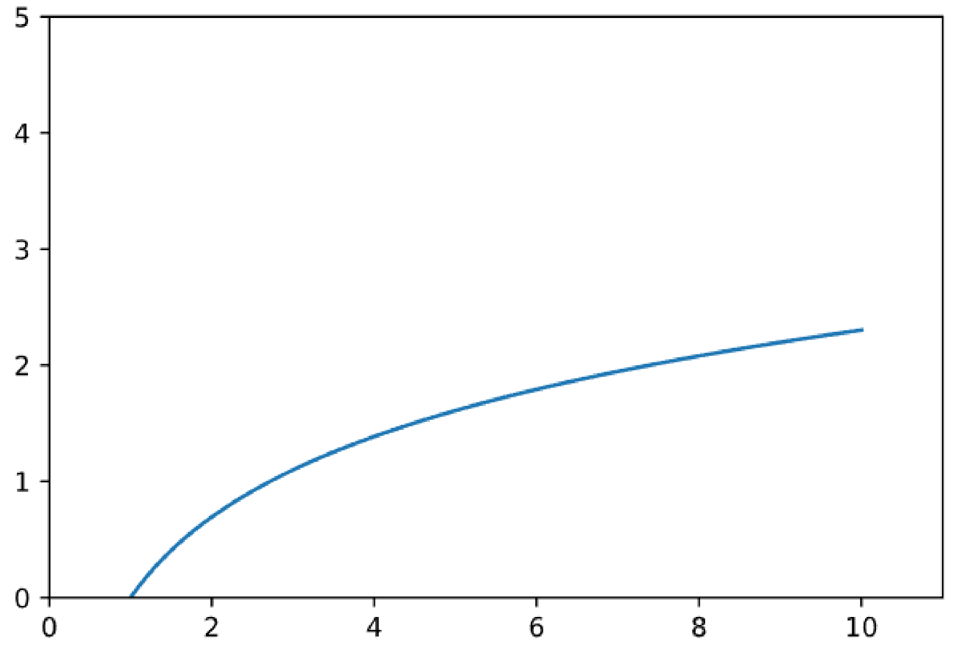


`plt.axis('off')`

Axis Limit

- *plt.*`xlim`(*left, right*)
- *plt.*`ylim`(*bottom, top*)
  - Set (or get) the x-limits (or y-limits) of the current axes
  - *left, right*: x-limits
  - *bottom, top*: y-limits

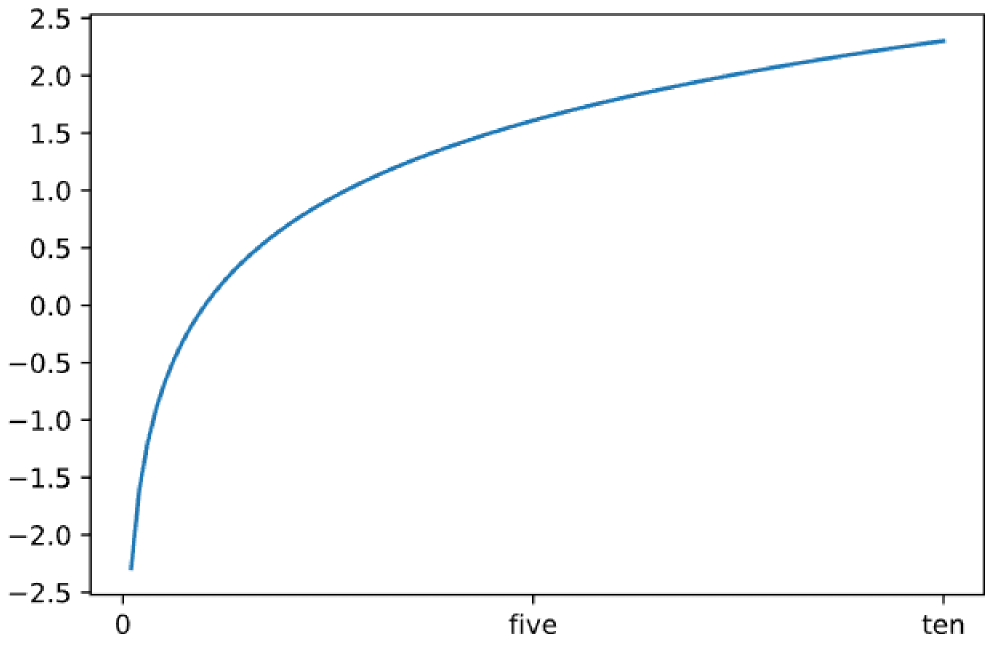
```
x = np.linspace(0.1, 10, 100)
plt.xlim(0, 11)
plt.ylim(top=5)
plt.plot(x, np.log(x))
```



Ticks

- *plt*.`xticks`(*ticks*, [*labels*], ...)
- *plt*.`yticks`(*ticks*, [*labels*], ...)
  - Set the current tick locations and labels of the x-axis (or y-axis)
  - *ticks*: A list of positions ticks should be placed
  - *labels*: A list of explicit labels to place at the given locations

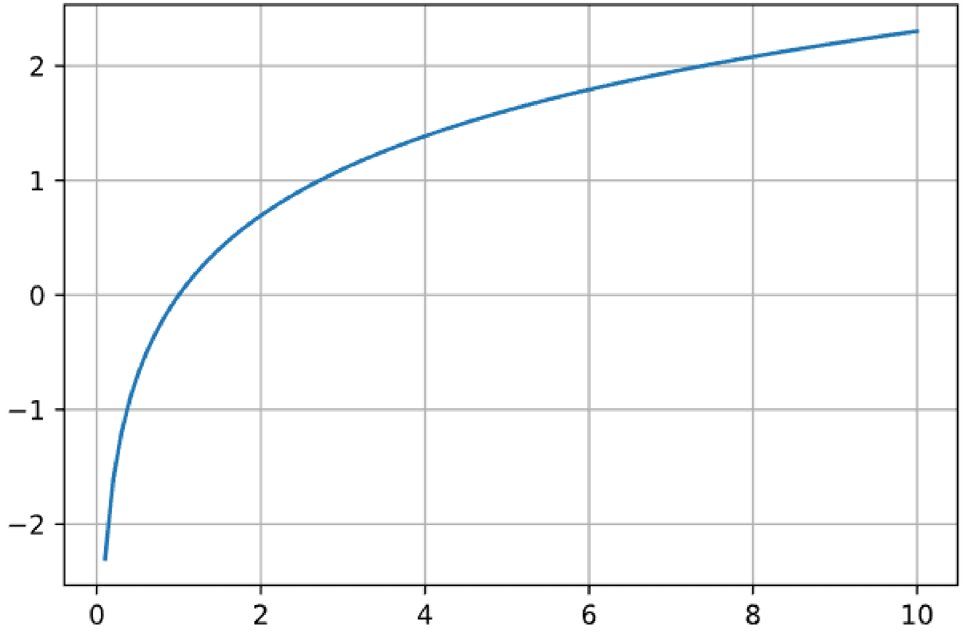
```
x = np.linspace(0.1, 10, 100)
plt.xticks(np.arange(0, 11, 5),
    ('0', 'five', 'ten'))
plt.yticks(np.arange(-3, 3, 0.5))
plt.plot(x, np.log(x))
```



Grids

- *plt*.`grid`([*b*], [*which*], [*axis*], ...)
  - Configure the grid lines
  - *b*: if True, show the grid lines (toggle if no argument given)
  - *which*: the grid lines to apply the changes on ('major', 'minor', or 'both)
  - *axis*: the axis to apply ('x', 'y', or 'both)

```
x = np.linspace(0.1, 10, 100)
plt.grid()
plt.plot(x, np.log(x))
```

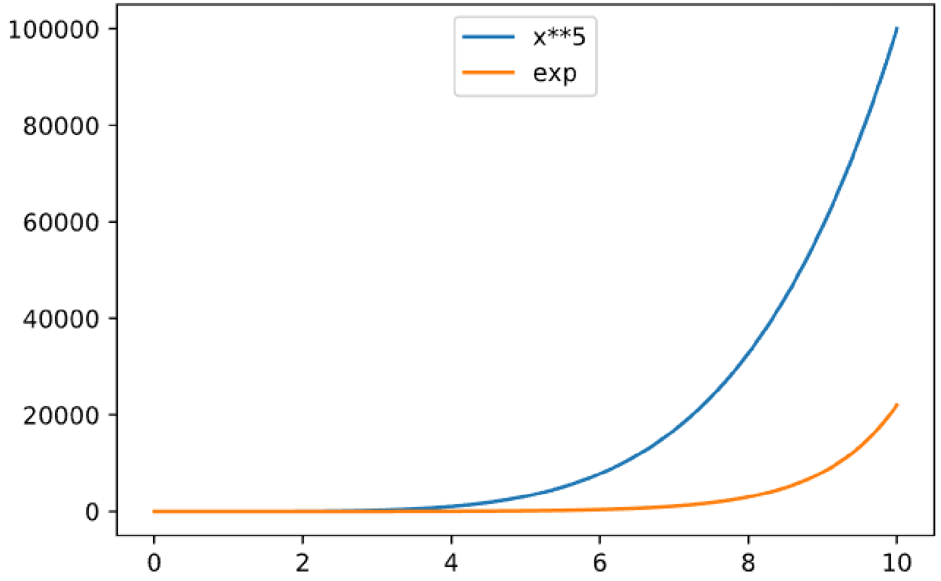


Legend

- *plt.*`legend`*([loc], [ncol], ...)*
  - Place a legend on the axes
  - If no argument is given, automatically determine and show the legend
  - *loc*: the location of the legend (default: 'best')
  - *ncol*: the number of columns (default: 1)

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

```
x = np.linspace(0, 10, 100)
plt.plot(x, x**5, label='x**5')
plt.plot(x, np.exp(x), label='exp')
plt.legend(loc='upper center')
```

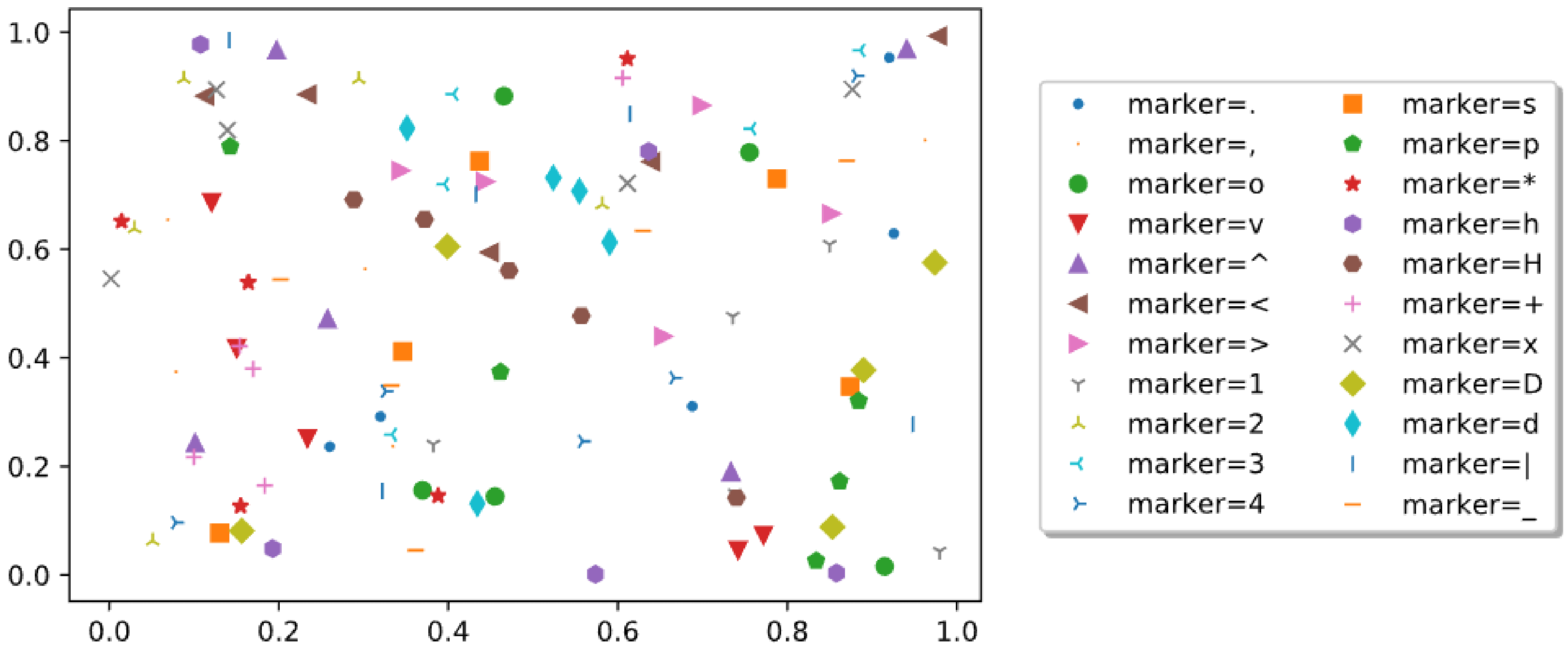


Legend: More Options

- *plt*.`legend`(...)
  - *bbox_to_anchor*: box that is used to position the legend in the arbitrary location with the form (x, y) or (x, y, width, height)
  - *frameon*: control whether the legend should be drawn on a frame (default: True)
  - *fancybox*: control whether round edges should be enabled (default: True)
  - *shadow*: control whether to draw a shadow behind the legend (default: False)
  - *fontsize*: control the font size of the legend
  - *title*: the legend's title
  - *markerscale*: the relative size of legend markers compared
  - *markerfirst*: if True, legend marker is placed to the left (default: True)
  - ...

Legend: Example

```
markers = '.,ov^<>1234sp*hH+xDd|_'
for m in markers:
    plt.plot(np.random.rand(5), np.random.rand(5), m, label='marker=%c' % m)
plt.legend(ncol=2, bbox_to_anchor=(1.05, 0.9), shadow=True)
```



Texts

- *plt.*`title`(*label*, [*loc*], [*pad*], ...)
  - label: text to use for the title
  - loc: which title to set ('center', 'left', or 'right')
  - pad: the offset from the top of the axes
- *plt.*`text`(*x*, *y*, *s*, [*loc*], [*pad*], ...)
  - x, y: the position to place the text
  - s: the text to display
- *plt.*`annotate`(*s*, [*xy*], [*xytext*], ...)
  - s: the text to display
  - xy: the point to annotate
  - xytext: the position to place the text at

```
x = np.linspace(0.1, 10, 100)
plt.plot(x, np.log(x))
plt.title("The log(x) graph")
plt.text(6, -1, 'matplotlib rules!')
plt.annotate('log(1) = 0', xy=(1,0),
    xytext=(2,-0.1),
    arrowprops=dict(facecolor='pink',
    shrink=0.05))
```

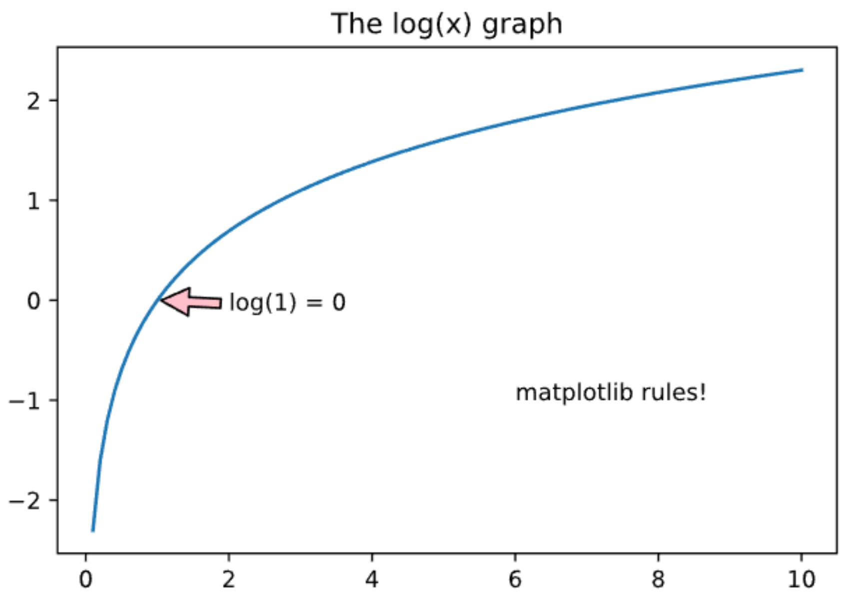
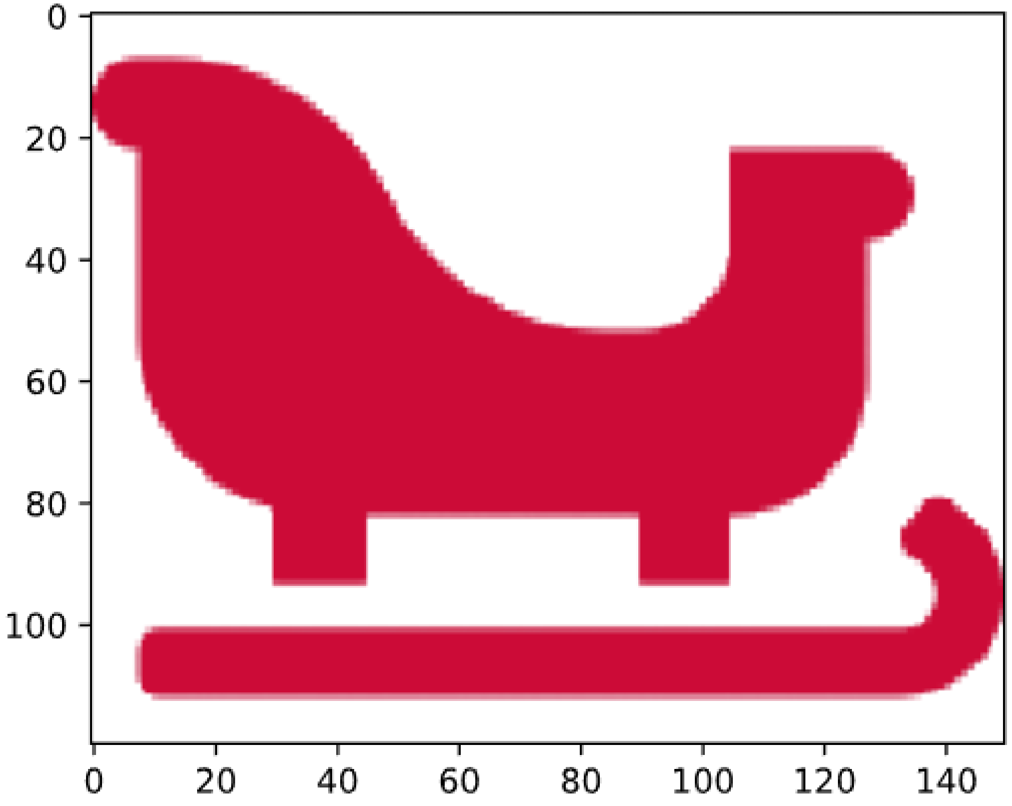


Image Submodule

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

image = mpimg.imread('sleds.png')
plt.imshow(img)
```



Saving Plots

- *plt.*`savefig`(*fname*, [*format*], [*dpi*], [*transparent*], ...)
  - Save the current figure
  - *fname*: file name to save. If format is not given, the output format is inferred from the extension of the fname
  - *format*: the file format (e.g., 'png', 'pdf', 'svg', 'jpg', ...)
  - dpi: the resolution in dots per inch
  - transparent: if True, make the plot transparent (default: False)

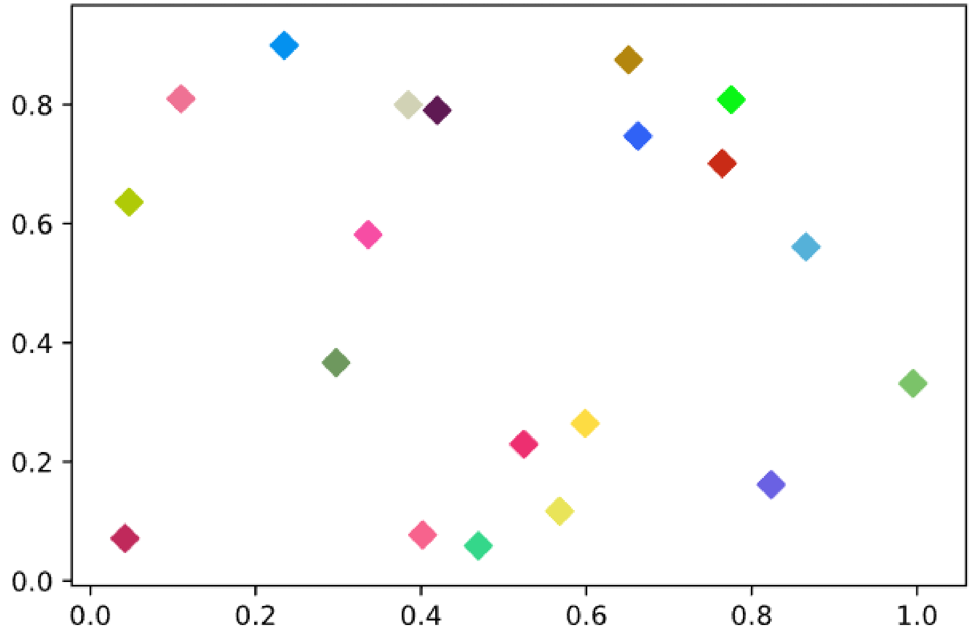
```
x = np.linspace(0.1, 10, 100)
plt.plot(x, np.log(x))
plt.savefig('log.png', dpi=300, transparent=True)
```

Scatter Plot

scatter()

- *plt*.`scatter`(*x*, *y*, [*s*], [*c*], [*marker*], [*alpha*], ...)
  - A scatter plot of y vs. x with varying marker size and/or color
  - *x*, *y*: data positions
  - *s*, *c*: the marker size (in points**2) and its color
  - *marker*: the marker style
  - *alpha*: the alpha blending value
    (0: transparent ~ 1: opaque)

```
x = np.random.rand(20)
y = np.random.rand(20)
colors=['#'+'%06x'%np.random.randint(256**3)
        for _ in range(20)]
plt.scatter(x, y, s=50, c=colors, marker='D')
```



scatter() vs. plot()

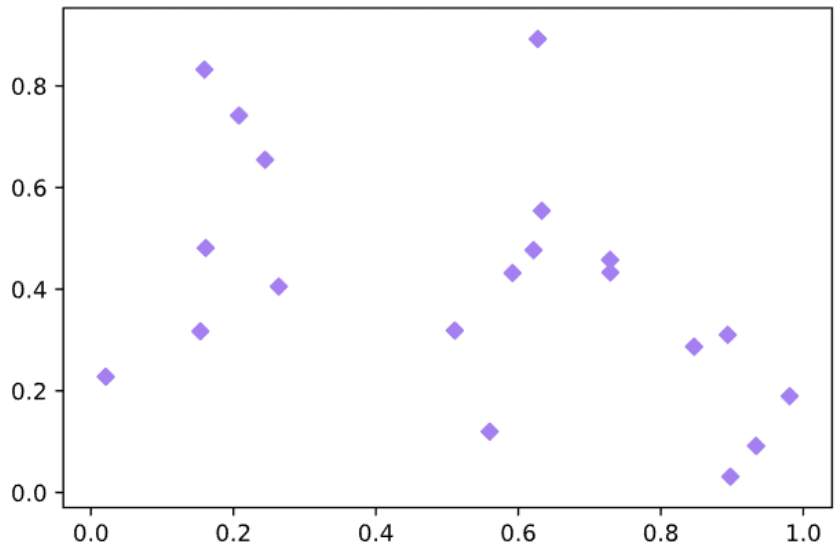
- `pyplot.plot()`

```
np.random.seed(20200101)
N=20
x = np.random.rand(N)
y = np.random.rand(N)
plt.plot(x,y,'D',c='#a37ff1',markersize=5)
```

- `pyplot.scatter()`

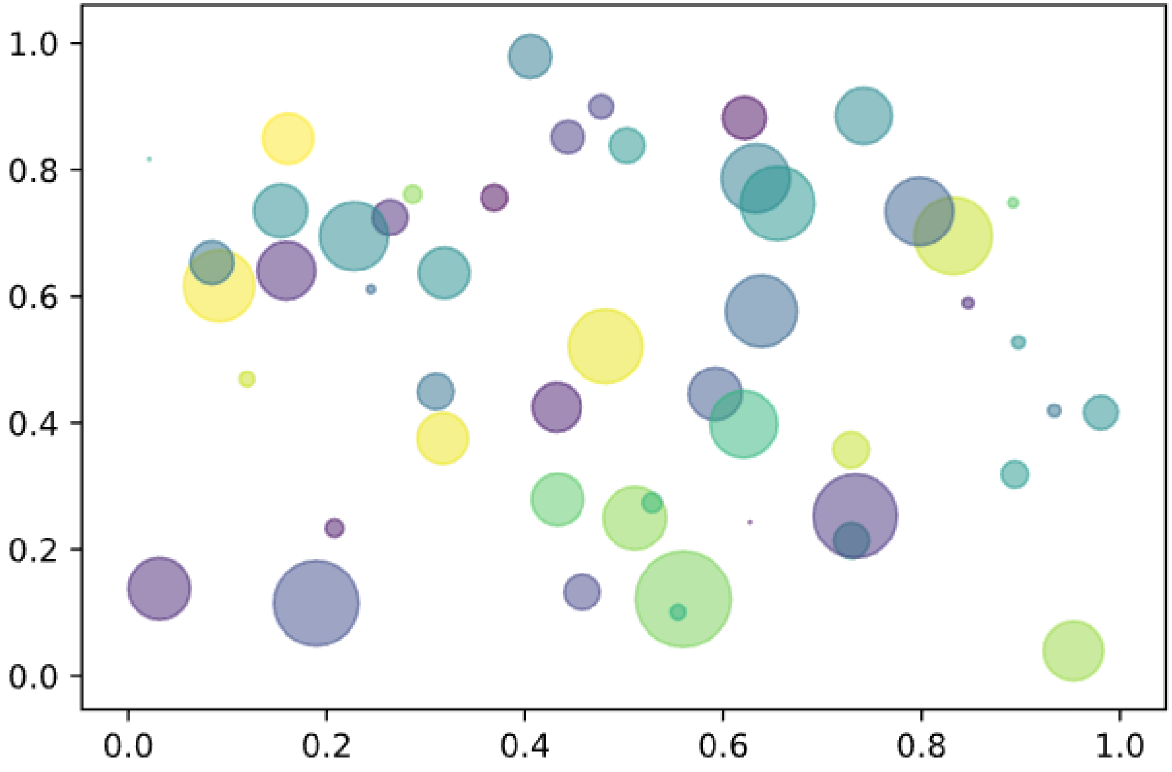
```
np.random.seed(20200101)
N=20
x = np.random.rand(N)
y = np.random.rand(N)
plt.scatter(x,y,s=25,c='#a37ff1',marker='D')
```

- Both can be used for simple cases ➔ same result
- `scatter()` is more comfortable to configure each data point
- `plot()` allows to plot the lines connecting data points



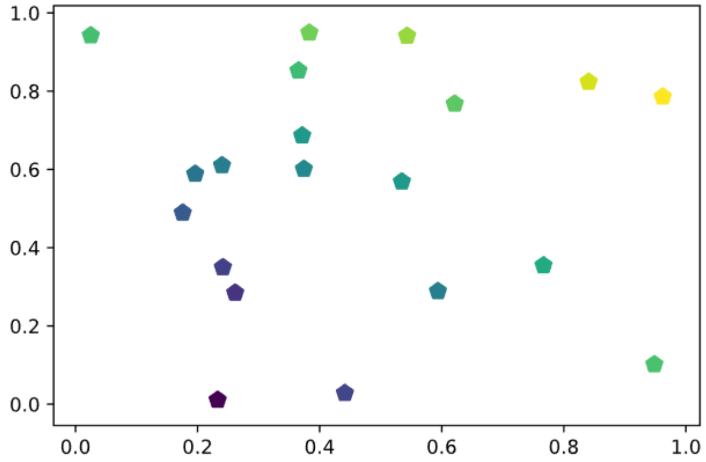
Scatter Plot with Different Marker Sizes

```
np.random.seed(20200101)

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area=(30 * np.random.rand(N))**2

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
```

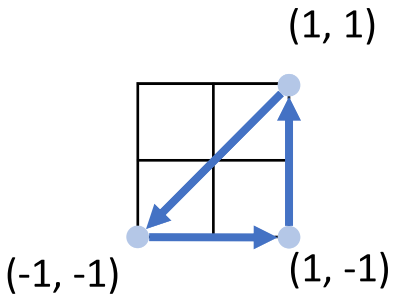


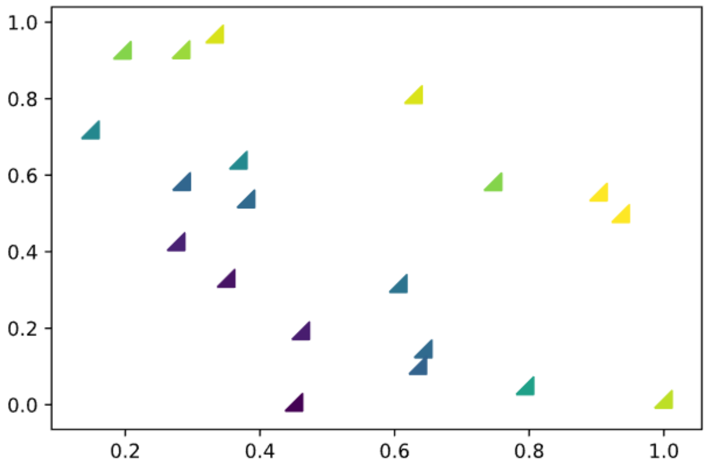
Scatter Plot with Polygon Symbols

- Marker = (numsides, style, angle)
  - *numsides*: number of sides
  - *style*: regular polygon(0), star-like symbol(1), asterisk(2), circle(3)
  - *angle*: angle of rotation
- User-defined marker:
  - A list of (x, y) describing a symbol with center = (0, 0)



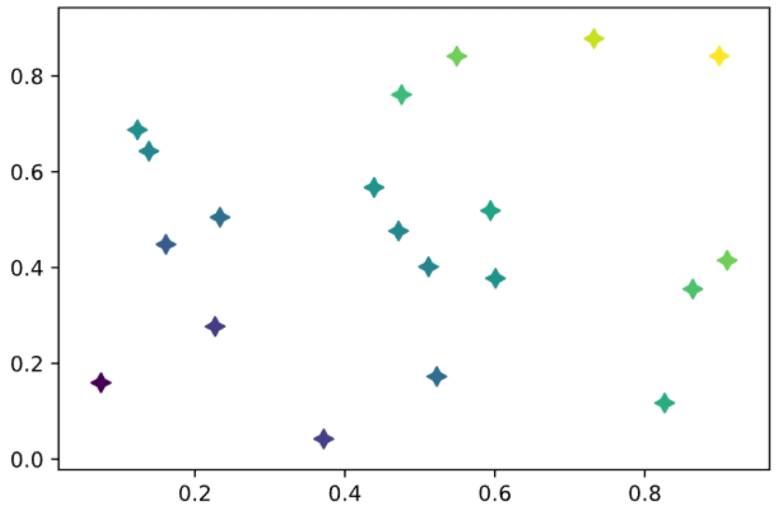

```
x = np.random.rand(20)
y = np.random.rand(20)
z = np.sqrt(x**2 + y**2)
plt.scatter(x, y, s=80, c=z, marker=(5,0))
v = np.array([[-1,-1],[1,-1],[1,1],[-1,-1]])
plt.scatter(x, y, s=80, c=z, marker=v)
```

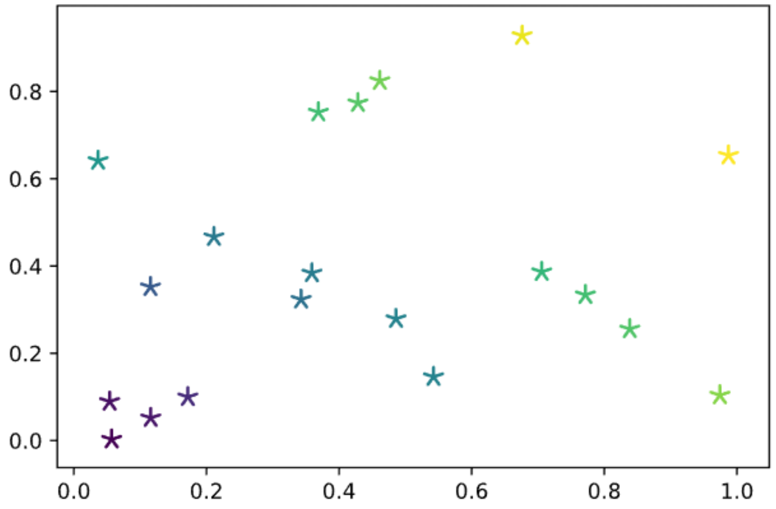




Scatter Plot with Stars and Asterisks

```
x = np.random.rand(20)
y = np.random.rand(20)
z = np.sqrt(x**2 + y**2)

# with 4-side star
plt.scatter(x, y, s=80, c=z, marker=(4,1))

# with 5-side asterisk
plt.scatter(x, y, s=80, c=z, marker=(5,2))
```



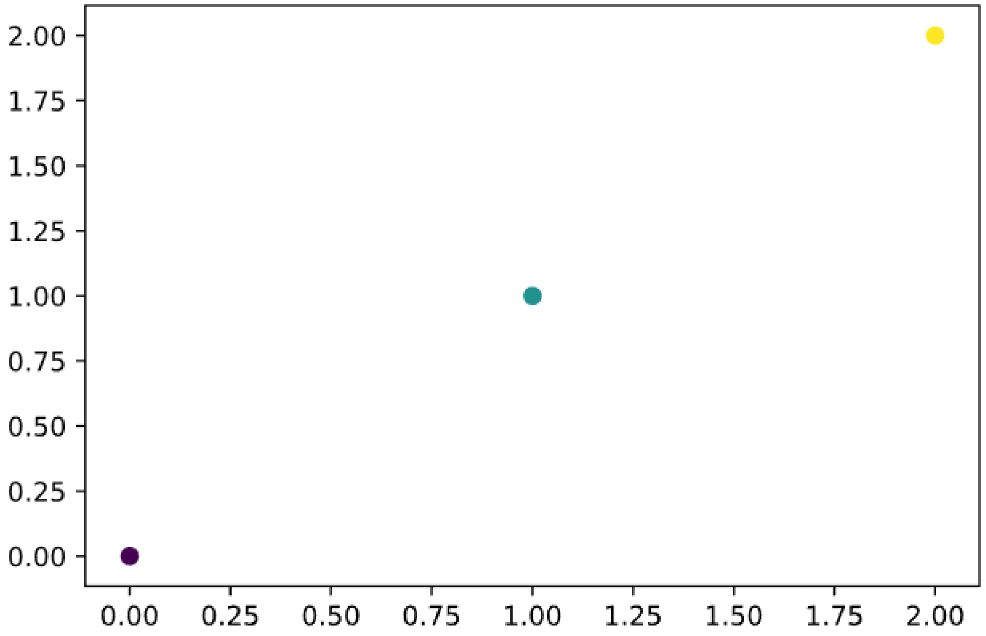


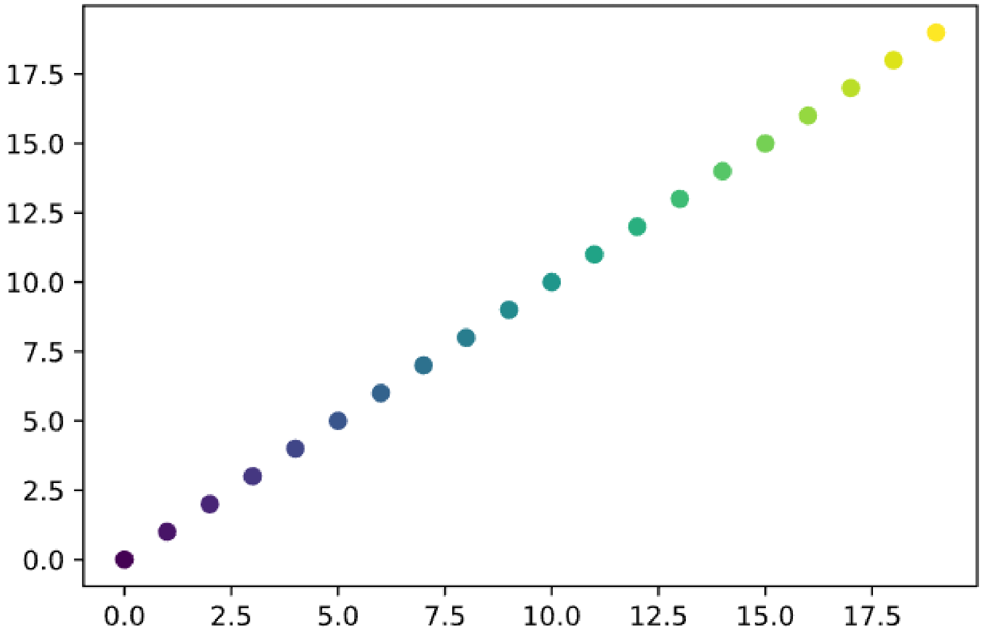
Default Color Map

- 'viridis':


```
x = np.arange(3)
plt.scatter(x, x, c=x)
```

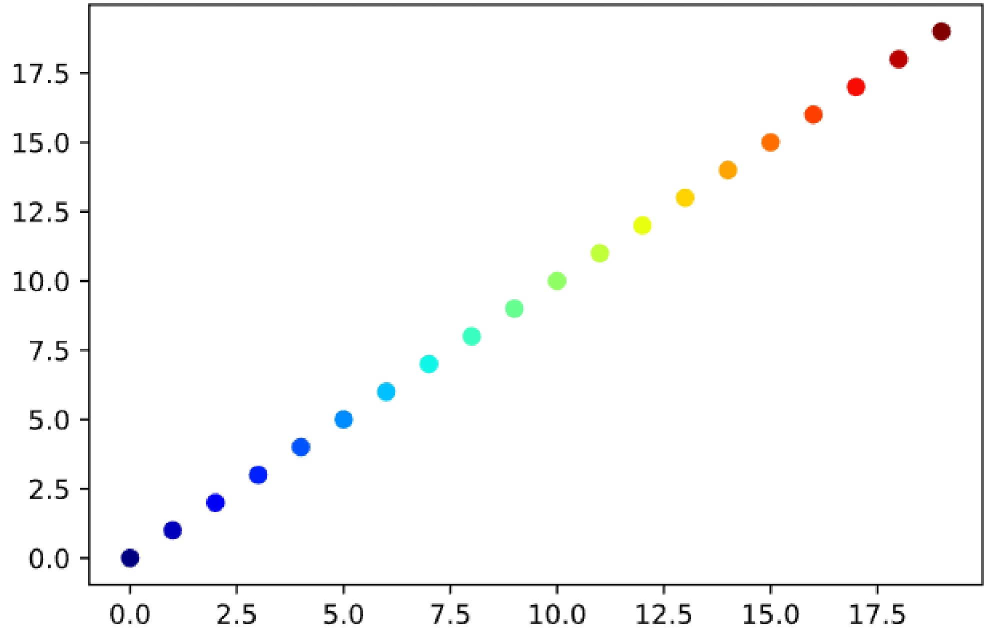
```
x = np.arange(20)
plt.scatter(x, x, c=x)
```

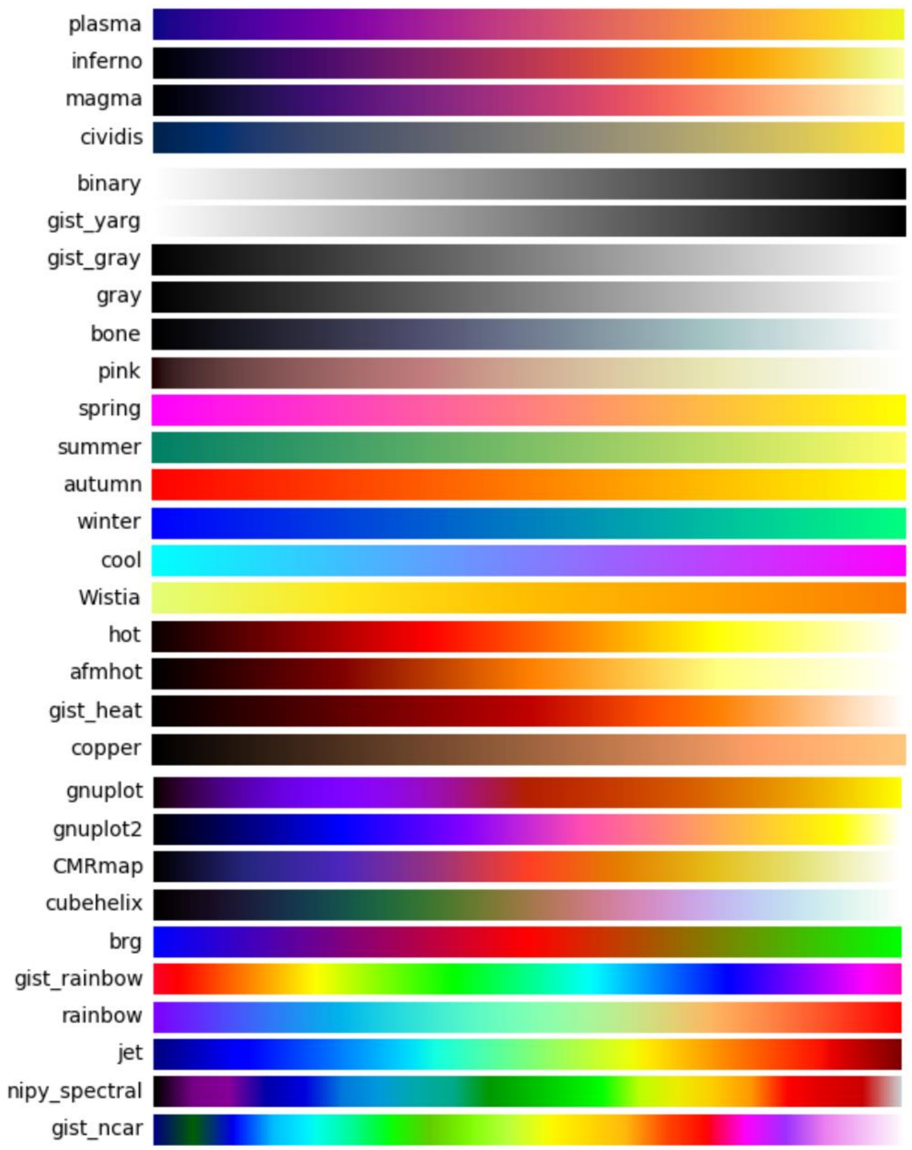




Changing a Color Map

```
plt.rc('image', cmap='jet')
x = np.arange(20)
plt.scatter(x, x, c=x)
```





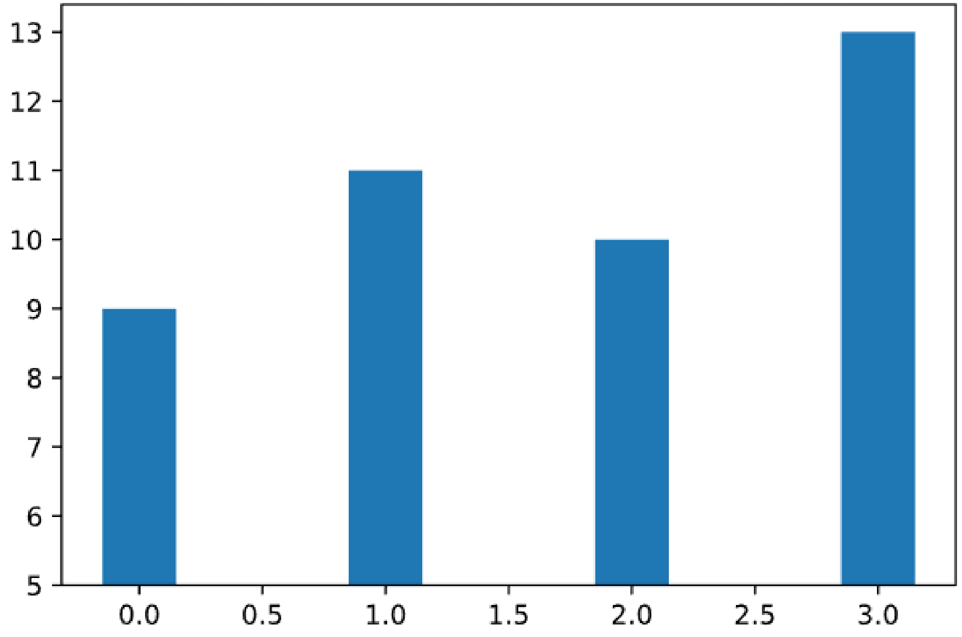
For other colormaps, visit
https://matplotlib.org/tutorials/colors/colormaps.html

Bar Chart

bar()

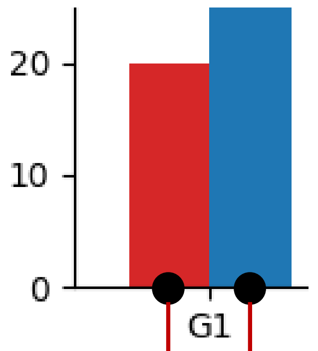
- *plt.*bar(*x*, *height*, [*width*], [*bottom*], [*align*], ...)
  - Make a bar plot
  - *x*, *height*: data points
  - *width*: the width(s) of the bars (default: 0.8)
  - *bottom*: the y coordinate(s) of the bars bases
  - *align*: alignment of the bars to the x coordiantes -- 'center' (default) or 'edge'

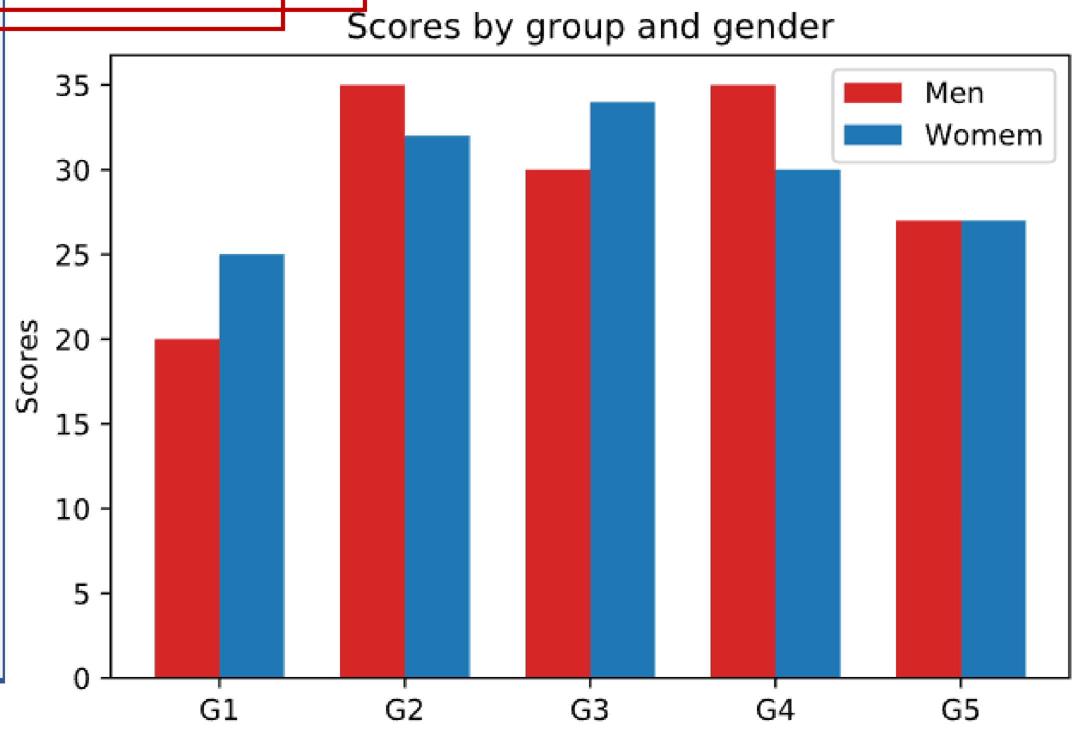
```
x = np.arange(4)
y = [4, 6, 5, 8]
plt.bar(x, y, width=0.3, bottom=5)
```



Side-by-Side Bar Chart

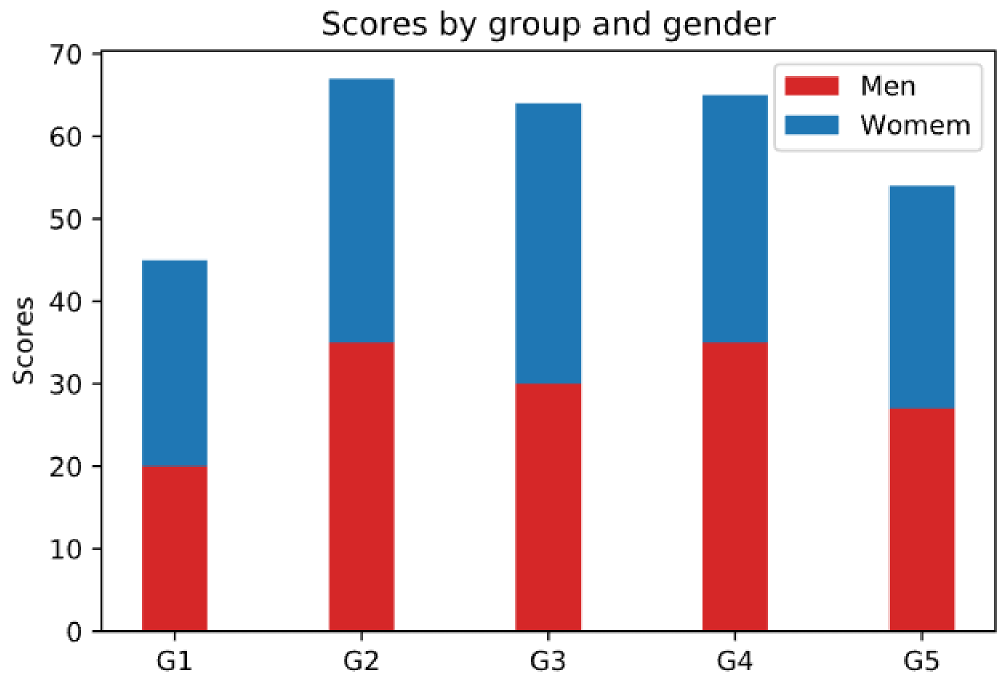
```
N = 5

m = (20, 35, 30, 35, 27)
w = (25, 32, 34, 30, 27)
x = np.arange(N)
width=0.35

plt.bar(x-width/2, m, width, color='#d62728')
plt.bar(x+width/2, w, width)

plt.xticks(x, ('G1','G2','G3','G4','G5'))
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.legend(('Men', 'Womem'))
```





Stacked Bar Chart

```
N = 5

m = (20, 35, 30, 35, 27)
w = (25, 32, 34, 30, 27)
x = np.arange(N)
width=0.35

plt.bar(x, m, width, color='#d62728')
plt.bar(x, w, width, bottom=m)

plt.xticks(x, ('G1','G2','G3','G4','G5'))
plt.ylabel('Scores')
plt.title('Scores by group and gender')
plt.legend(('Men', 'Womem'))
```

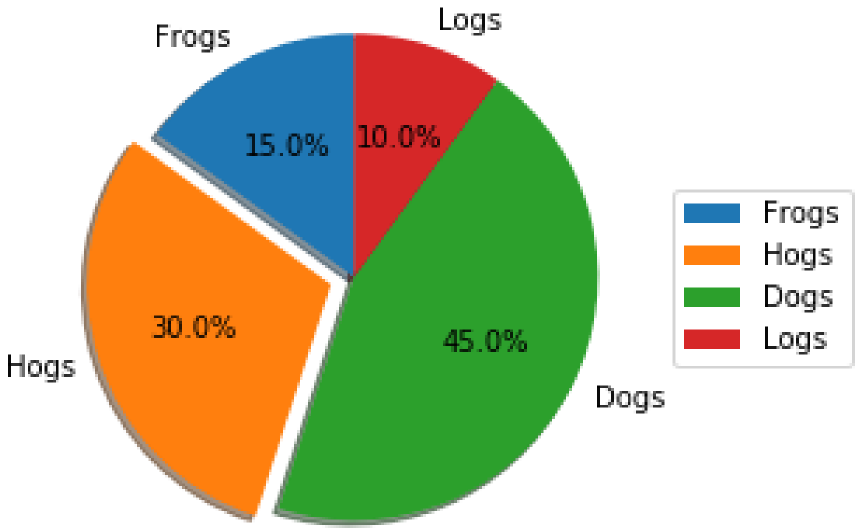


Pie Chart

bar()

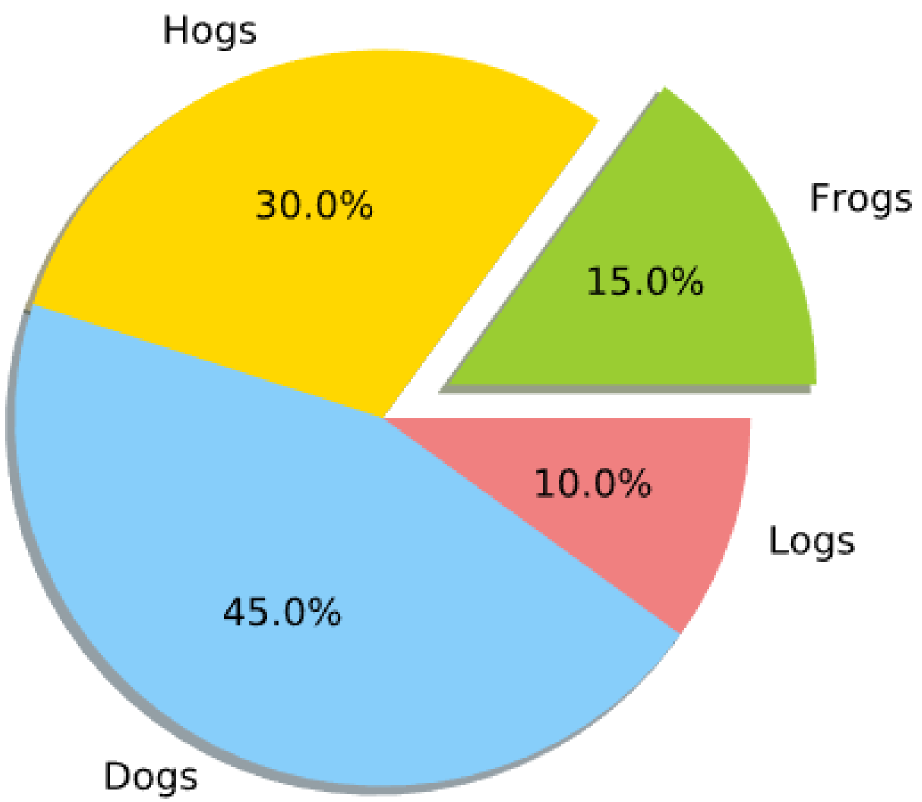
- *plt*.pie(*x*, [*labels*], [*colors*], [*explode*], [*startangle*], [*autopct*], …)
  - Plot a pie chart
  - *x*: the wedge sizes
  - *labels*: a sequence of labels
  - *colors*: a sequence of colors
  - *explode*: the fraction of the radius with which to offset each wedge
  - *startangle*: start pie chart with this degree
  - *autopct*: label wedges with numeric value

```
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
ex = (0, 0.1, 0, 0)
plt.pie(sizes, explode=ex, labels=labels,
   autopct='%.1f%%', shadow=True, startangle=90)
plt.legend(loc='center right',
   bbox_to_anchor=(1.4, 0.5))
```



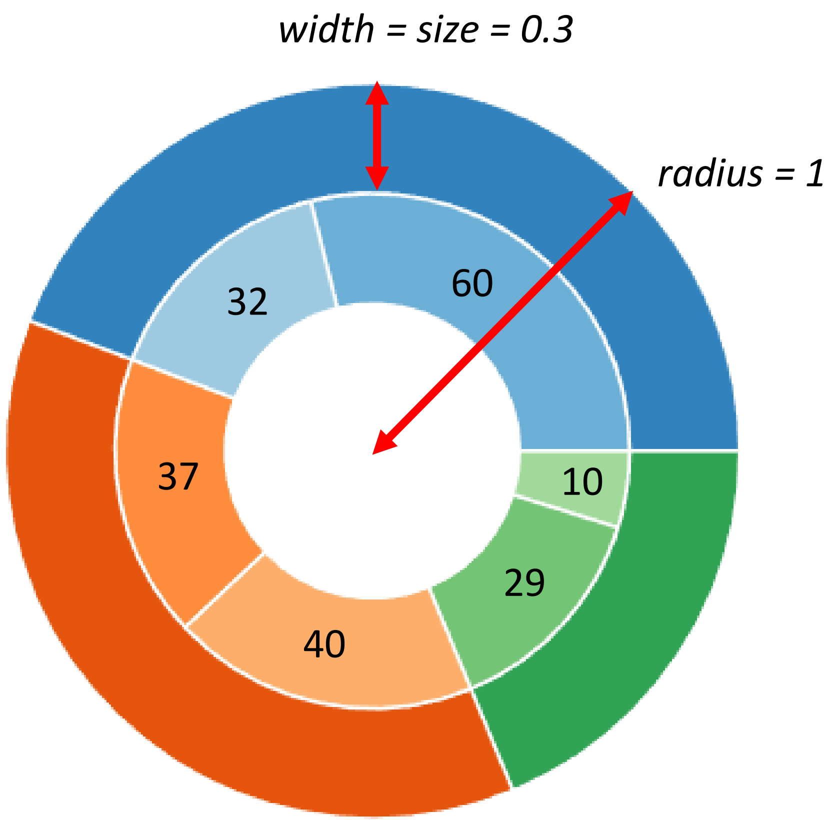
Using Specific Colors

```
labels = ['Frogs','Hogs','Dogs','Logs']
sizes = [15, 30, 45, 10]
ex = (0.2, 0, 0, 0)
c = ['yellowgreen', 'gold',
     'lightskyblue', 'lightcoral']

plt.pie(sizes, explode=ex, labels=labels,
    colors=c, autopct='%.1f%%',
    shadow=True, startangle=0)

plt.axis('equal')
```



Nested Pie Chart

```
size = 0.3
vals = np.array([[60.,32.],[37.,40.],[29.,10.]])

cmap = plt.get_cmap('tab20c')
outer_colors = cmap(np.arange(3)*4)
inner_colors = cmap(np.array([1,2,5,6,9,10]))

plt.pie(vals.sum(axis=1), radius=1,
    colors=outer_colors,
    wedgeprops=dict(width=size, edgecolor='w'))
plt.pie(vals.flatten(), radius=1-size,
    colors=inner_colors,
    wedgeprops=dict(width=size, edgecolor='w'))
plt.axis('equal')
```

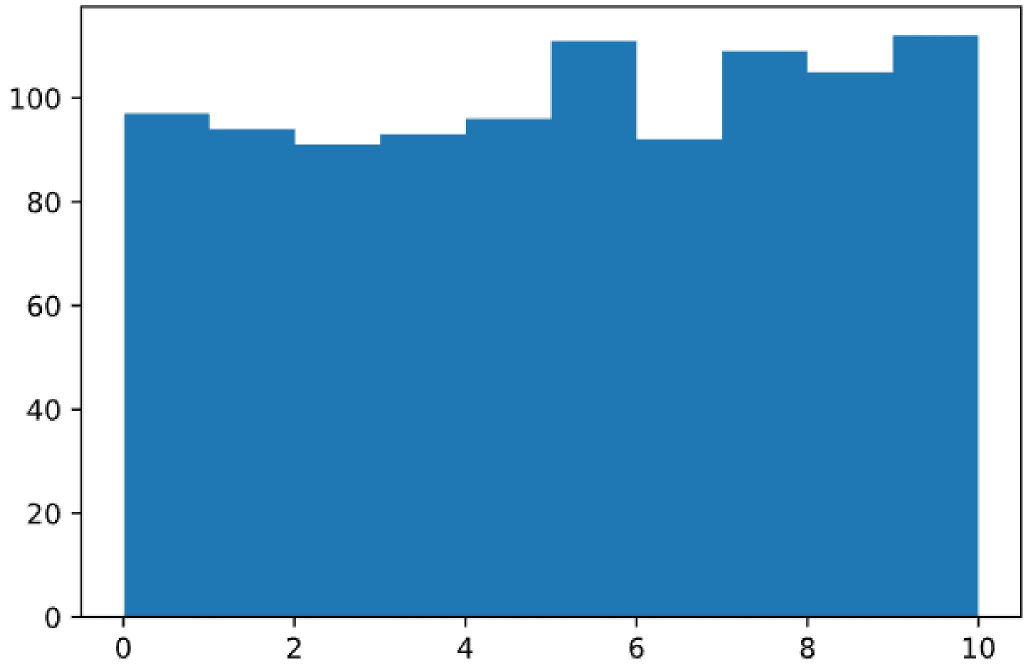


Histogram

hist()

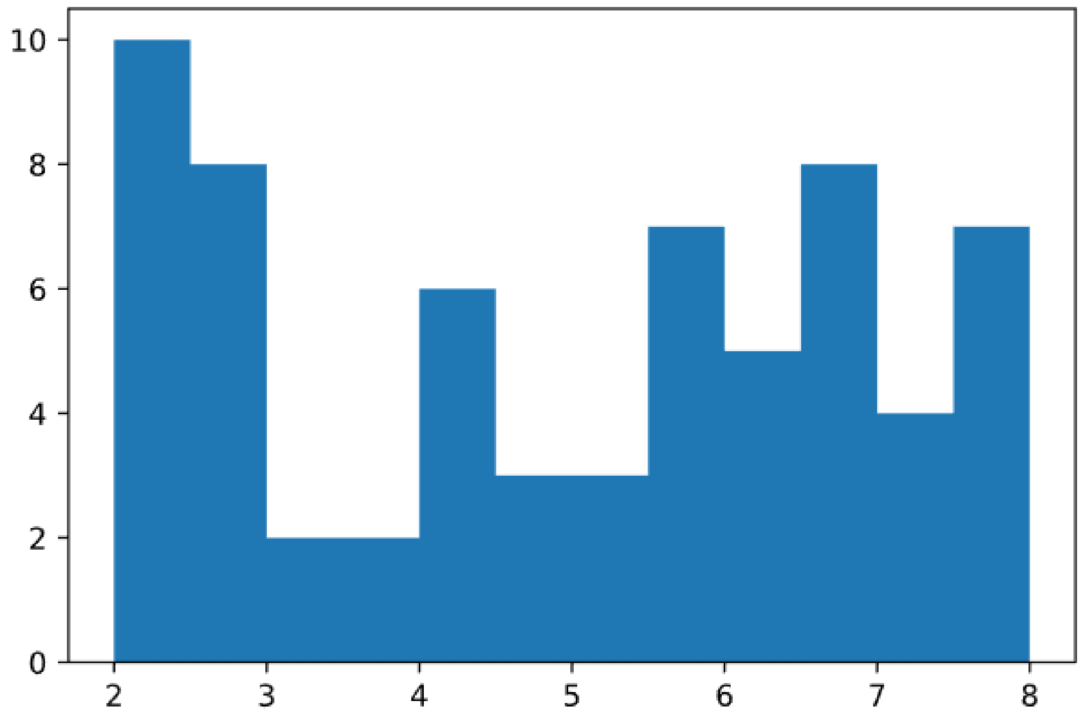
- *plt*.`hist`(*x*, [*bins*], [*range*], [*cumulative*], [*bottom*], ...)
  - Plot a histogram
  - *x*: input values
  - *bins*: number of bins (default: 10)
  - *range*: the lower and upper range of the bins
  - *cumulative*: if True, shows the cumulative sum
  - *bottom*: the baseline of each bin

```
x = np.random.uniform(0., 10., 1000)
plt.hist(x)
```



Range and Bins

```
x = np.random.uniform(0., 10., 100)
plt.hist(x, range=[2., 8.], bins=12)
```



Split [2.0, 8.0] into 12 bins:

bin *0* has the count of $2. \le x < 2.5$
bin *1* has the count of $2.5 \le x < 3.0$

bin *i* has the count of $2.+0.5i \le x < 2.5+0.5i$

bin *11* has the count of $7.5 \le x < 8.0$

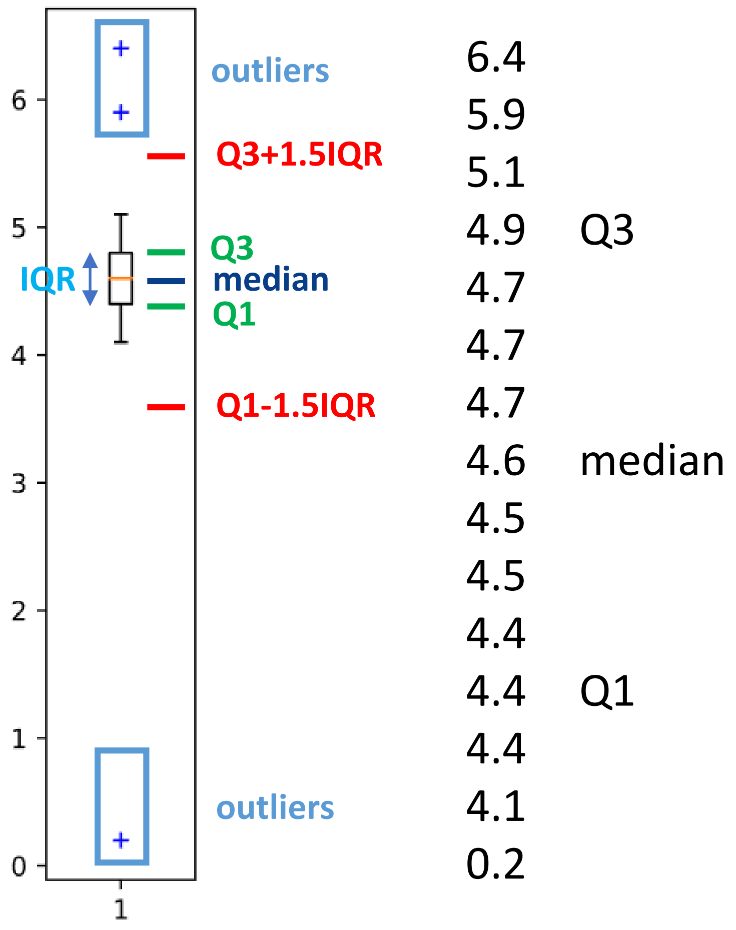
Box Plot

boxplot()

- *plt*.boxplot(*x*, [*notch*], [*sym*], [*vert*], [*whis*], ...)
  - Make a box and whisker plot
  - *x*: input data
  - *notch*: If True, produce a notched box plot (default: False)
  - *sym*: the symbol for outliers
  - *vert*: If True, make the boxes vertical (default: True)
  - *whis*: the reach of the whiskers (default: 1.5)

  (cf.) IQR = InterQuartile Range

```
x = [0.2, 4.1, 4.4, 4.4, 4.4, 4.5, 4.5, 4.6,
     4.7, 4.7, 4.7, 4.9, 5.1, 5.9, 6.4]
plt.boxplot(x, sym='b+')
```

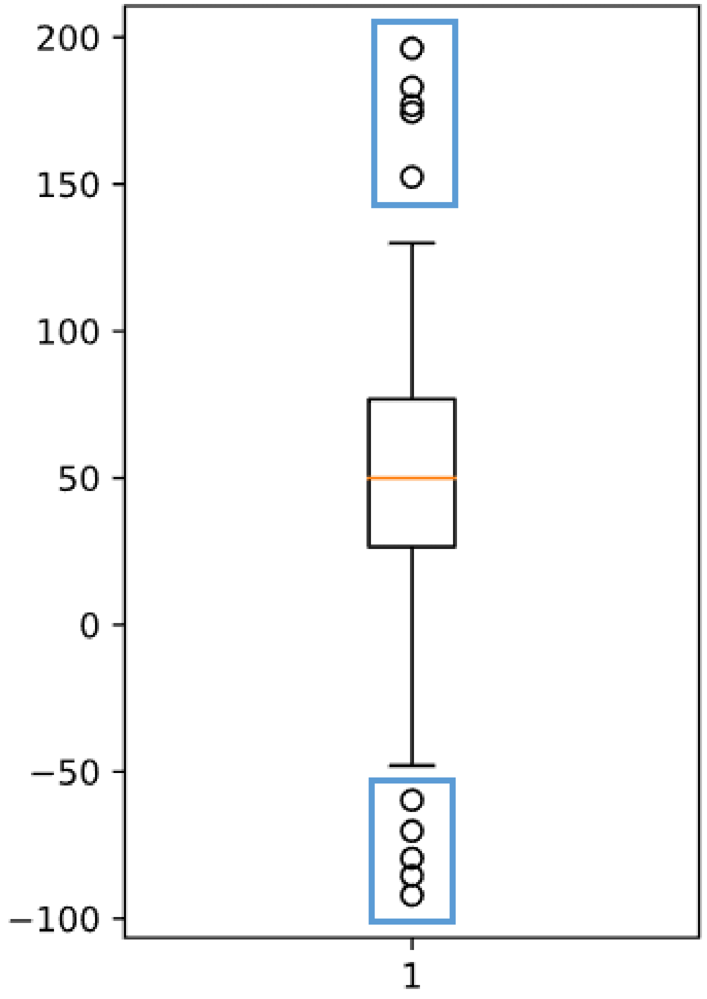


IQR = Q3 – Q1 = 4.9 - 4.4 = 0.5
Q1 - 1.5IQR = 4.4 – 1.5*0.5 = 3.65
Q3 + 1.5IQR = 4.9 + 1.5*0.5 = 5.65

Box Plot Example

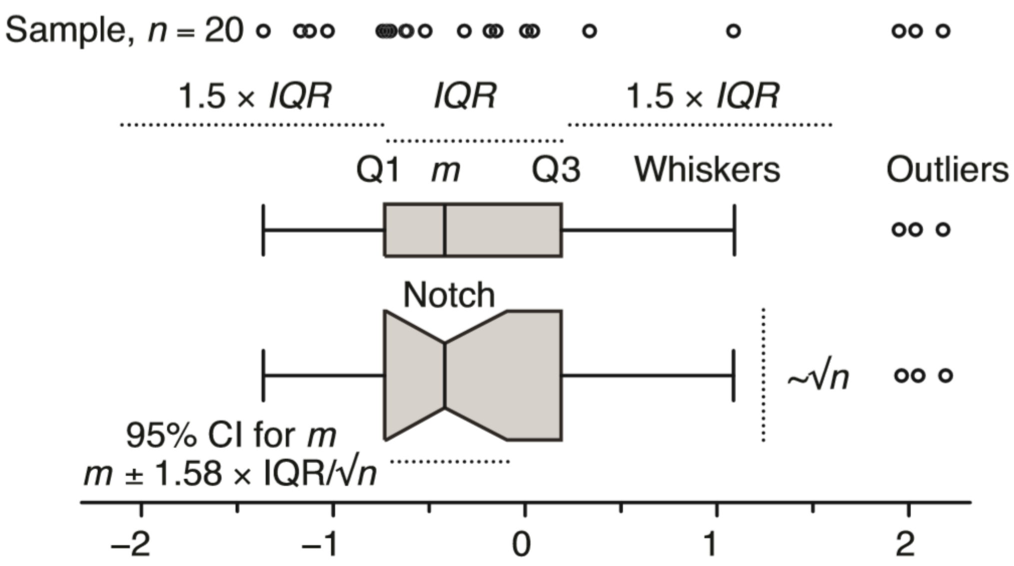
```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)

data = np.concatenate((spread, center, high, low), 0)
plt.figure(figsize=(3, 5))  # fig size in inches
plt.boxplot(data)
```

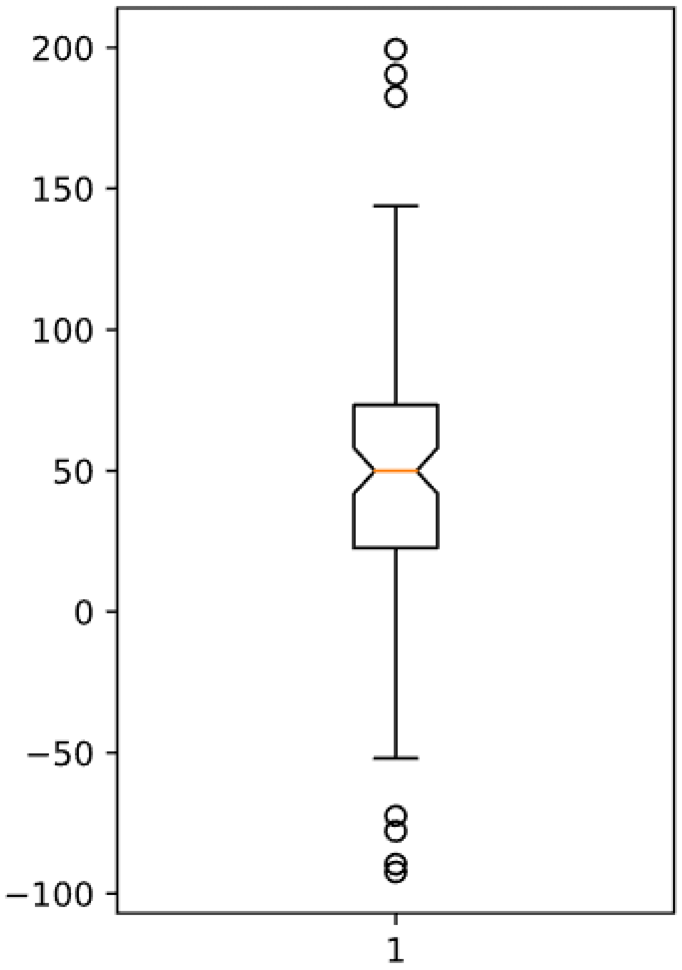
- spread: 50 numbers in [0, 100)
- center: 25 numbers with the value 50.0
- high: 10 numbers in [100, 200)
- low: 10 numbers in (-100, 0]



Notched Box Plot

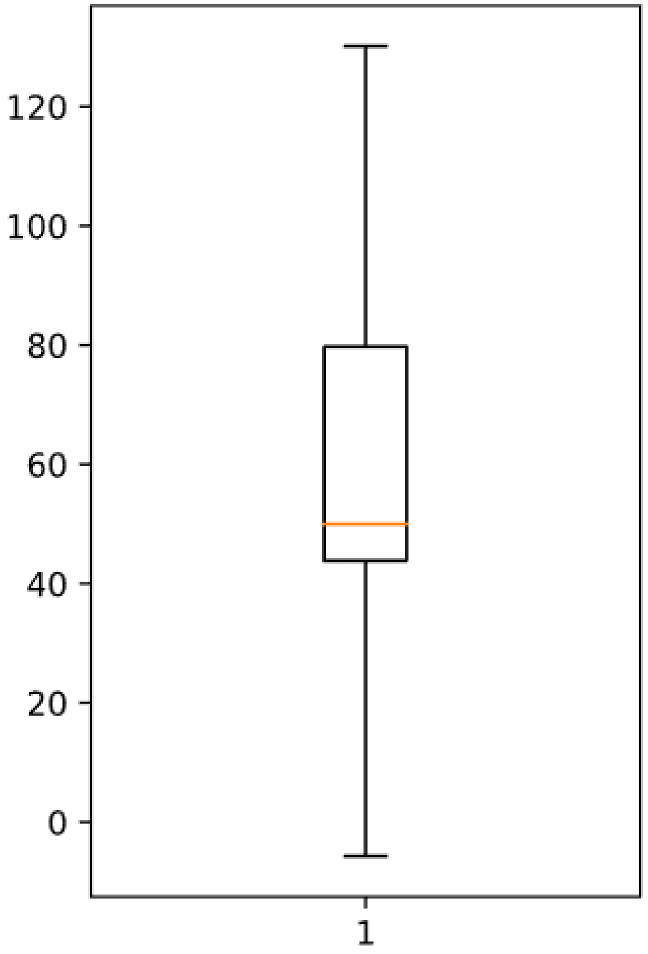
```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)
data = np.concatenate((spread, center, high, low), 0)
plt.figure(figsize=(3, 5))  # fig size in inches
plt.boxplot(data, notch=True)
```





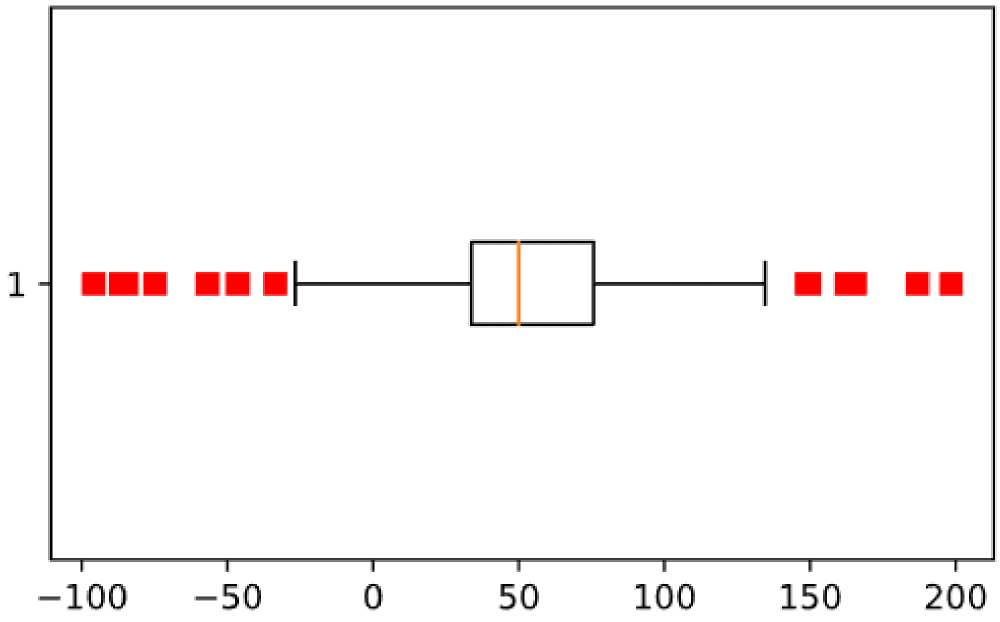
Removing Outliers

```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)
data = np.concatenate((spread, center, high, low), 0)

plt.figure(figsize=(3, 5))  # fig size in inches
plt.boxplot(data, sym=' ')
```



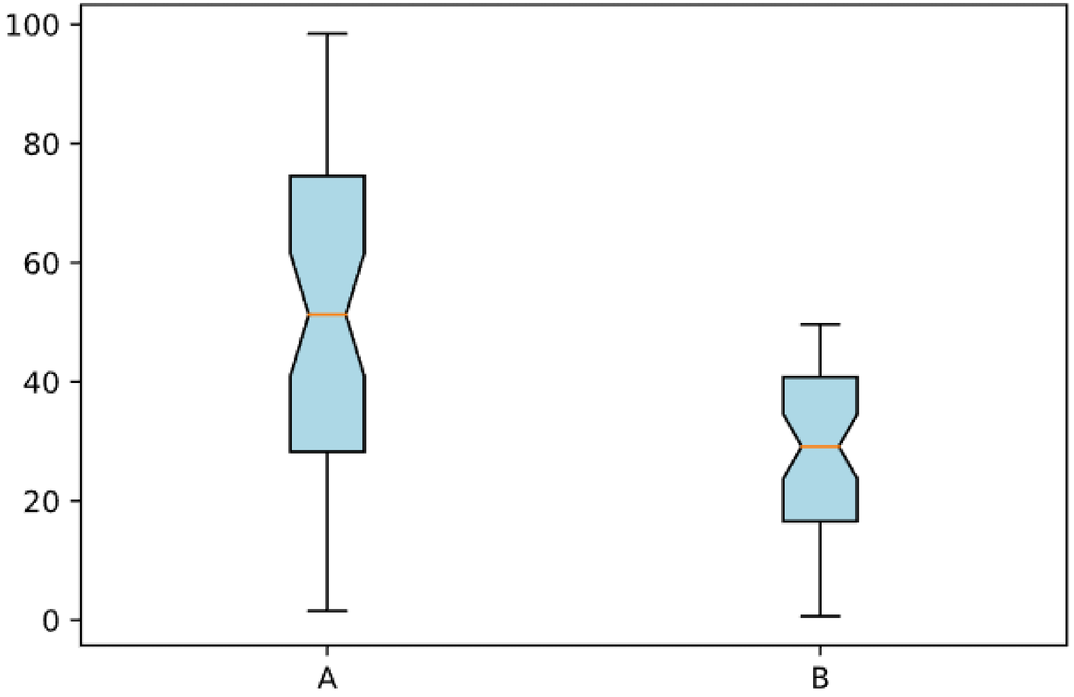
Horizontal Box Plot

```
spread = np.random.rand(50)*100
center = np.ones(25)*50
high = np.random.rand(10)*100+100
low = np.random.rand(10)*(-100)
data = np.concatenate((spread, center,
    high, low), 0)

plt.figure(figsize=(5, 3))
plt.boxplot(data, vert=False, sym='rs')
```



Multiple Box Plots

```
p1 = np.random.rand(50)*100
p2 = np.random.rand(50)*50
data = [p1, p2]

plt.boxplot(data, notch=True,
    patch_artist=True,
    boxprops=dict(facecolor='lightblue'),
    labels=['A', 'B'])
plt.boxplot(data)
```



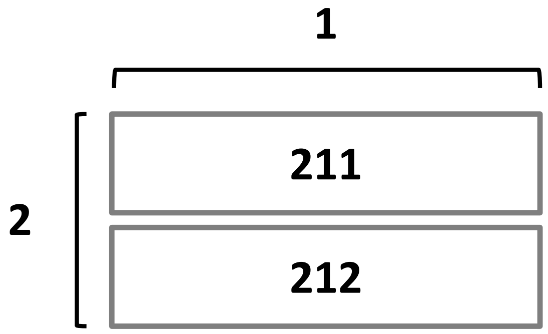
Subplots

figure()

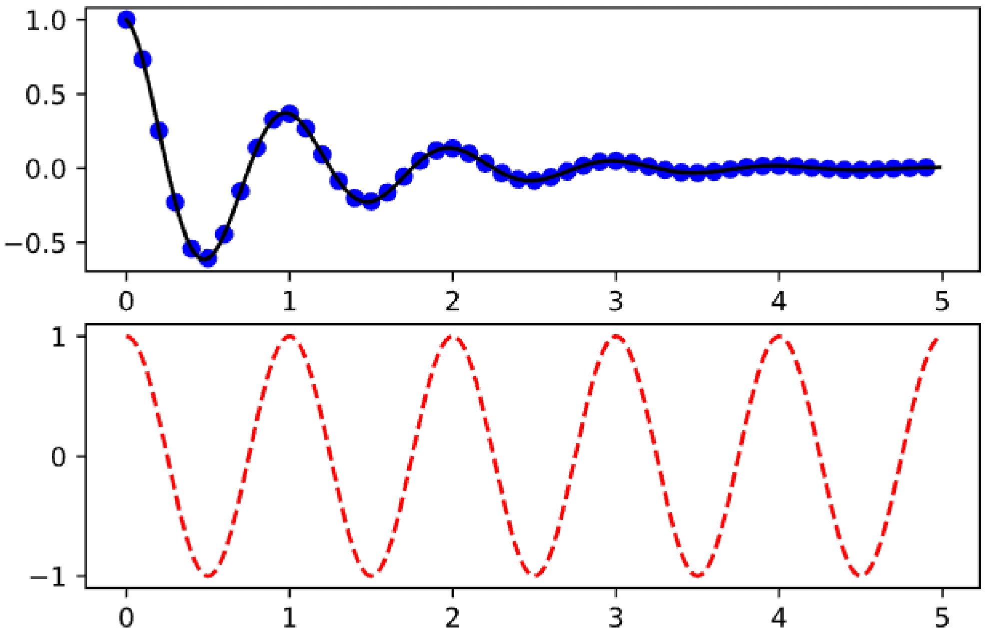
- *plt*.`figure`([*num*], [*figsize*], [*facecolor*], [*edgecolor*], [*frameon*], ...)
  - Create a new figure
  - *num*: If not provided, a new figure will be created with the figure number incremented. If provided, a figure with this id is created
  - *figsize*: (width, height) in inches
  - *facecolor*: the background color
  - *edgecolor*: the border color
  - *frameon*: If True, draw the figure frame (default: True)

subplot()

- *plt*.subplot([*pos*], ...)
  - Add a subplot to the current figure
  - *pos*: a three digit integer denoting the number of rows, the number of columns, and the index of the subplot
  - Returns the axes of the subplot



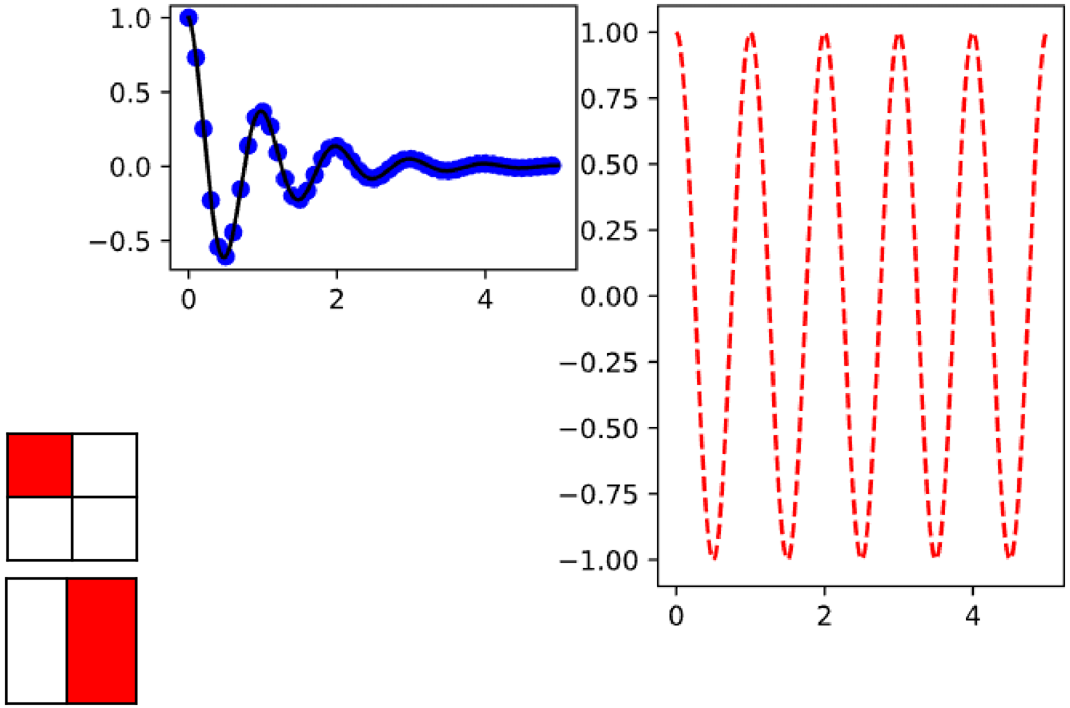

```
def f(t):
    return np.exp(-t)*np.cos(2*np.pi*t)
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

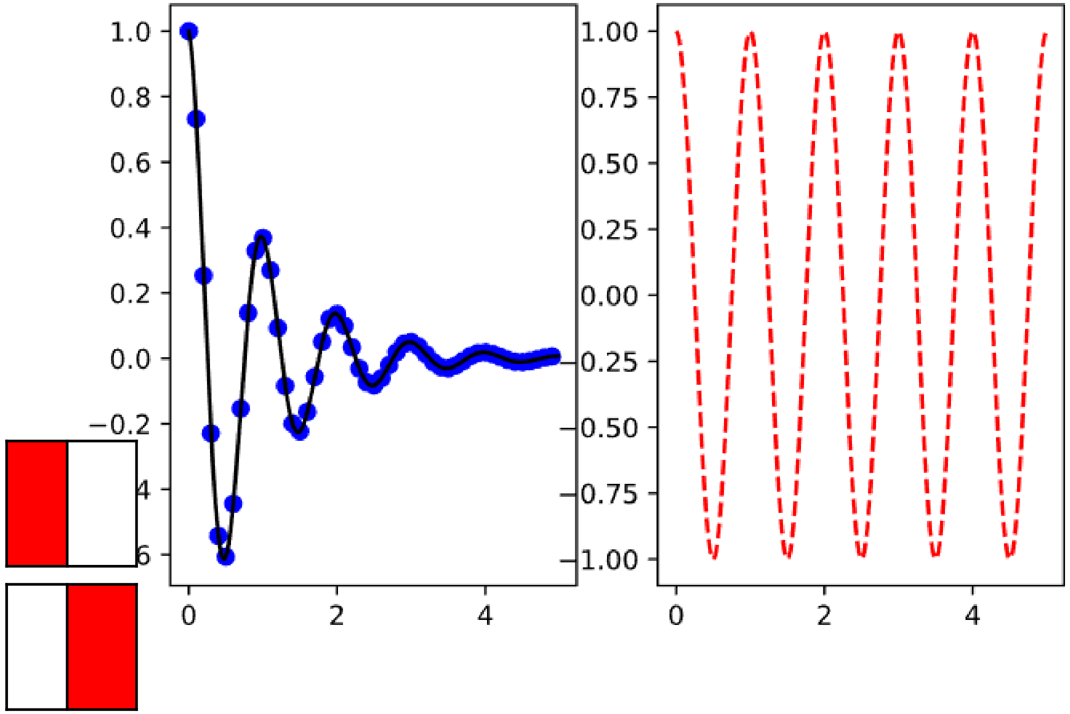


Subplot Examples

```python
plt.subplot(221)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(122)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```

```python
plt.subplot(121)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(122)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
```





Seaborn

What is "Seaborn" Module?

- Python visualization library based on matplotlib
  - More attractive and informative statistical graphics
  - Wrapper of matplotlib which improved the default styles
  - Even if plotting using only matplotlib.pyplot, importing seaborn will make nicer plots
- Dependency
  - Data structures: NumPy, Pandas
  - Statistical routines: SciPy
- Open source
  - https://seaborn.pydata.org

```
>>> import seaborn as sns
```

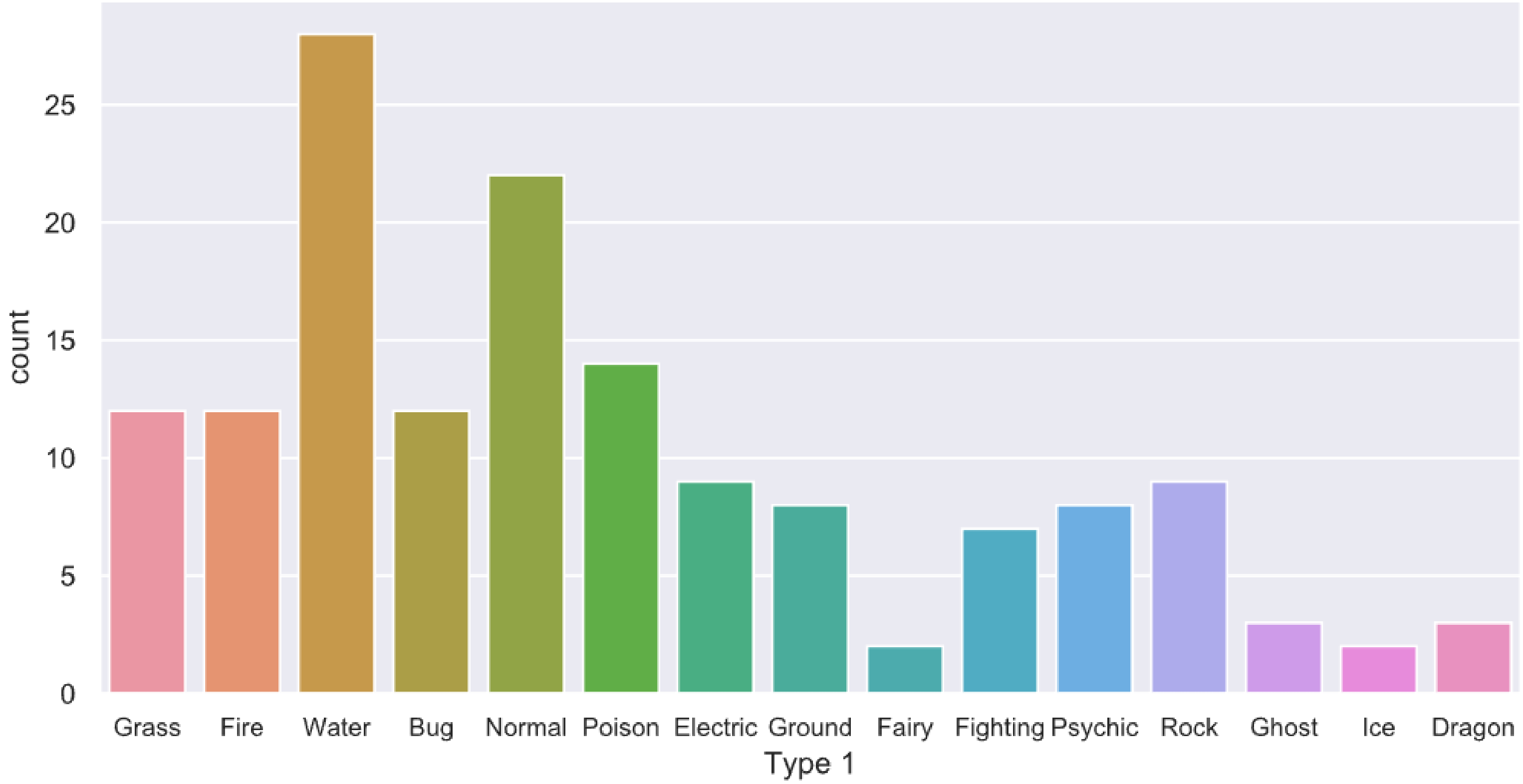
Pokemon Dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv('pokemon.csv')
df.head()
```

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Stage	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	2	False
3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	3	False
4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	2	False

Count Plot

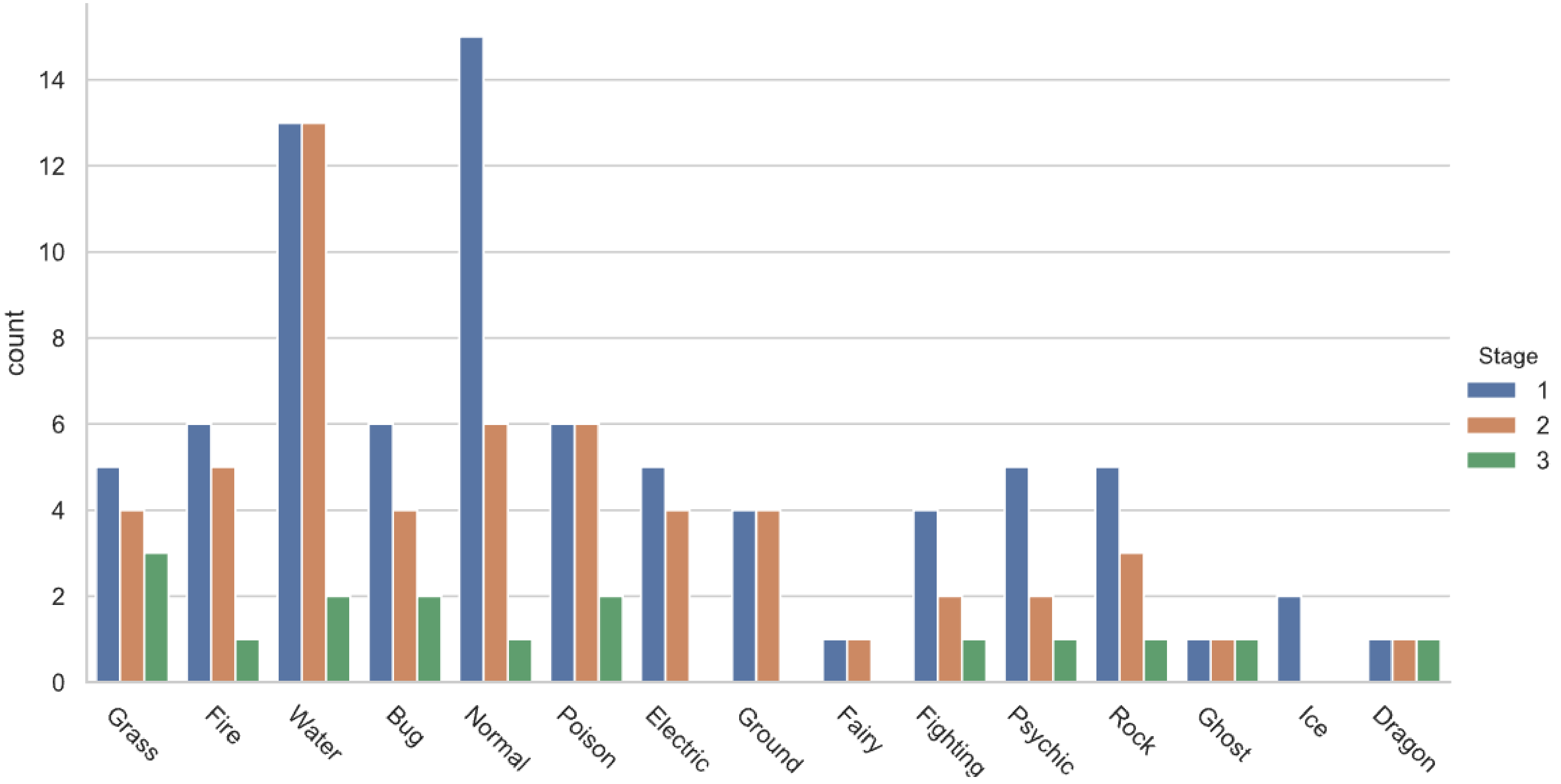
```
sns.set(style='darkgrid')
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.countplot(x='Type 1', data=df)
```



Categorical Plot

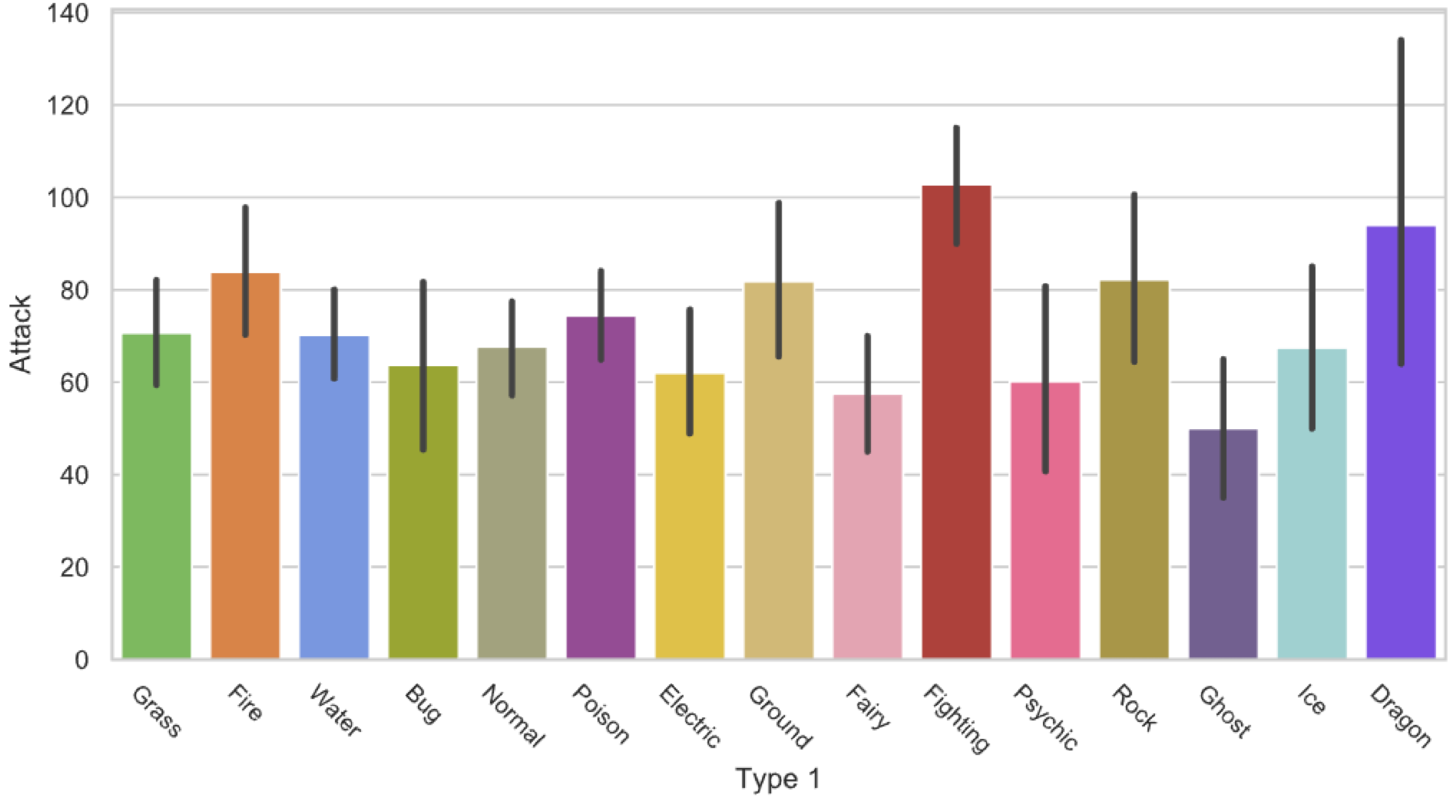
Group data

```
sns.catplot(x='Type 1', kind='count', hue='Stage', data=df, aspect=1.8)
plt.xticks(rotation=-45)
```



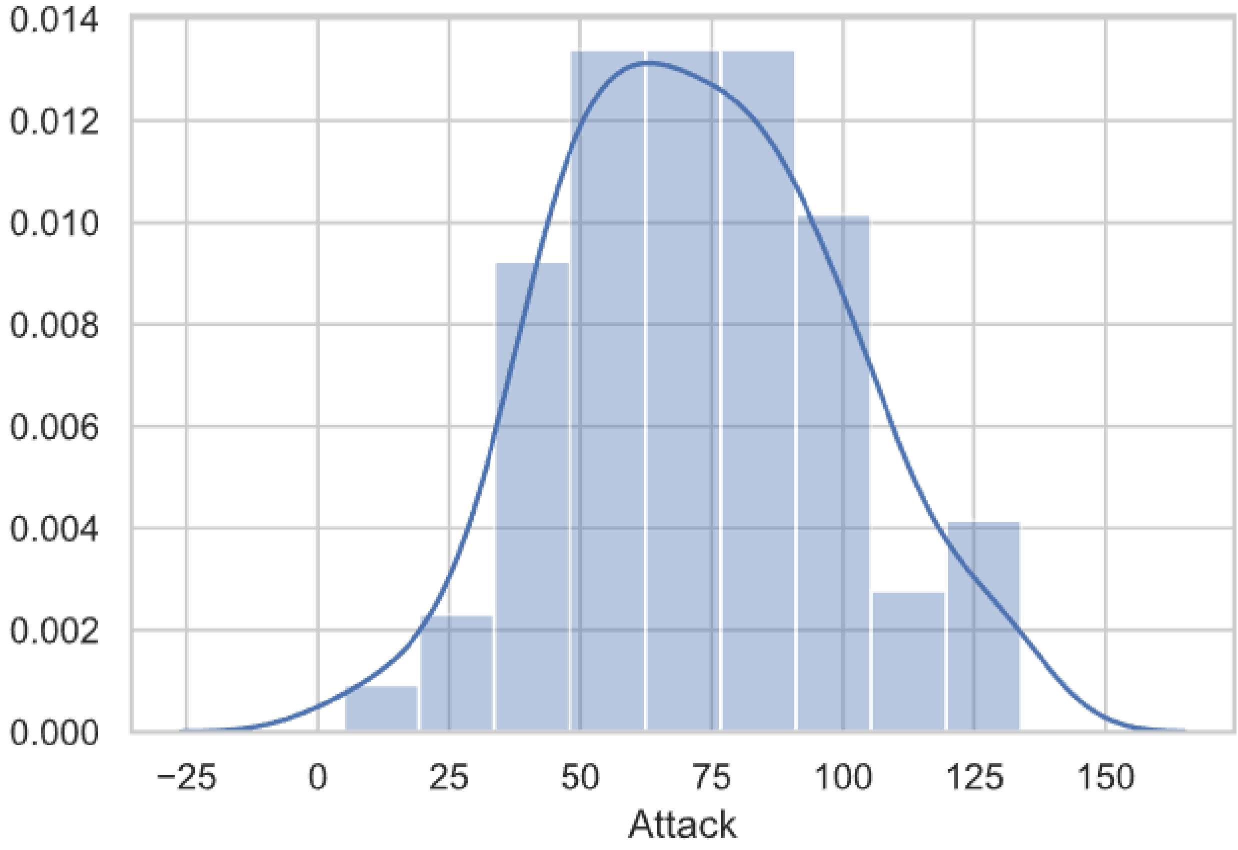
Bar Plot

```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10, rotation=-45)
sns.barplot(x='Type 1', y='Attack', data=df, palette=poke_colors)
```



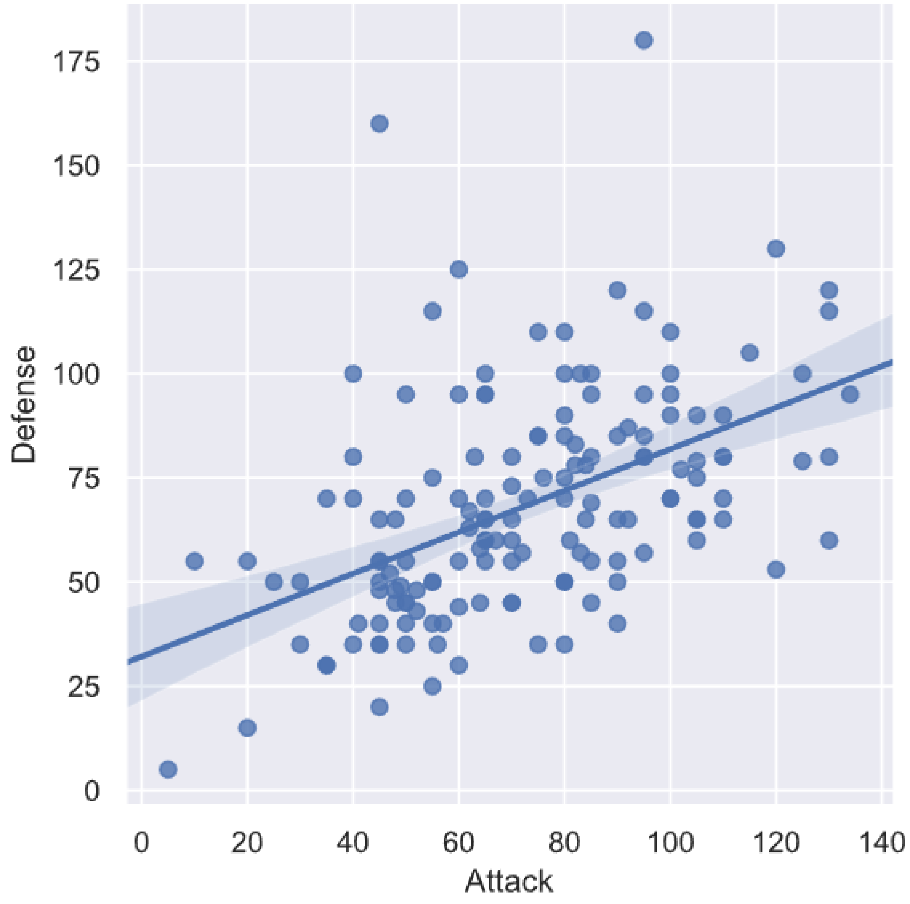
Histogram

```
sns.distplot(df.Attack)
```

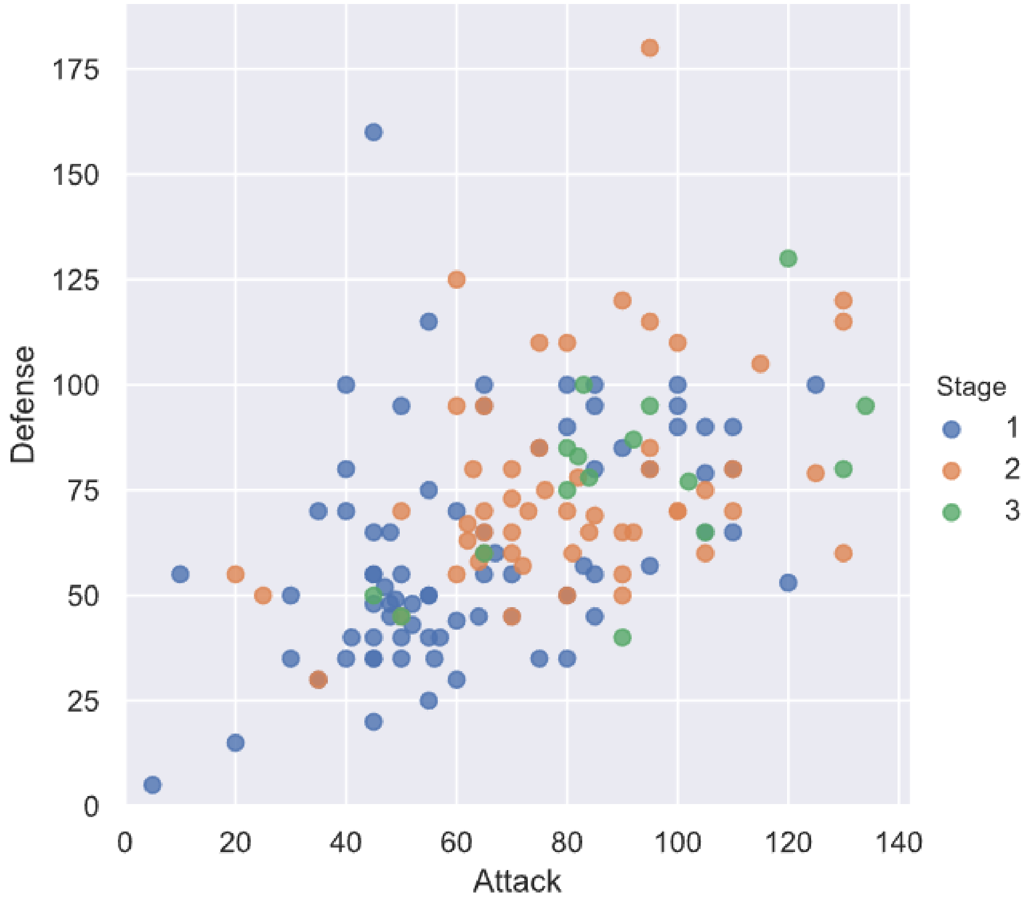


Scatter Plot

```
sns.lmplot(x='Attack', y='Defense', data=df)
```

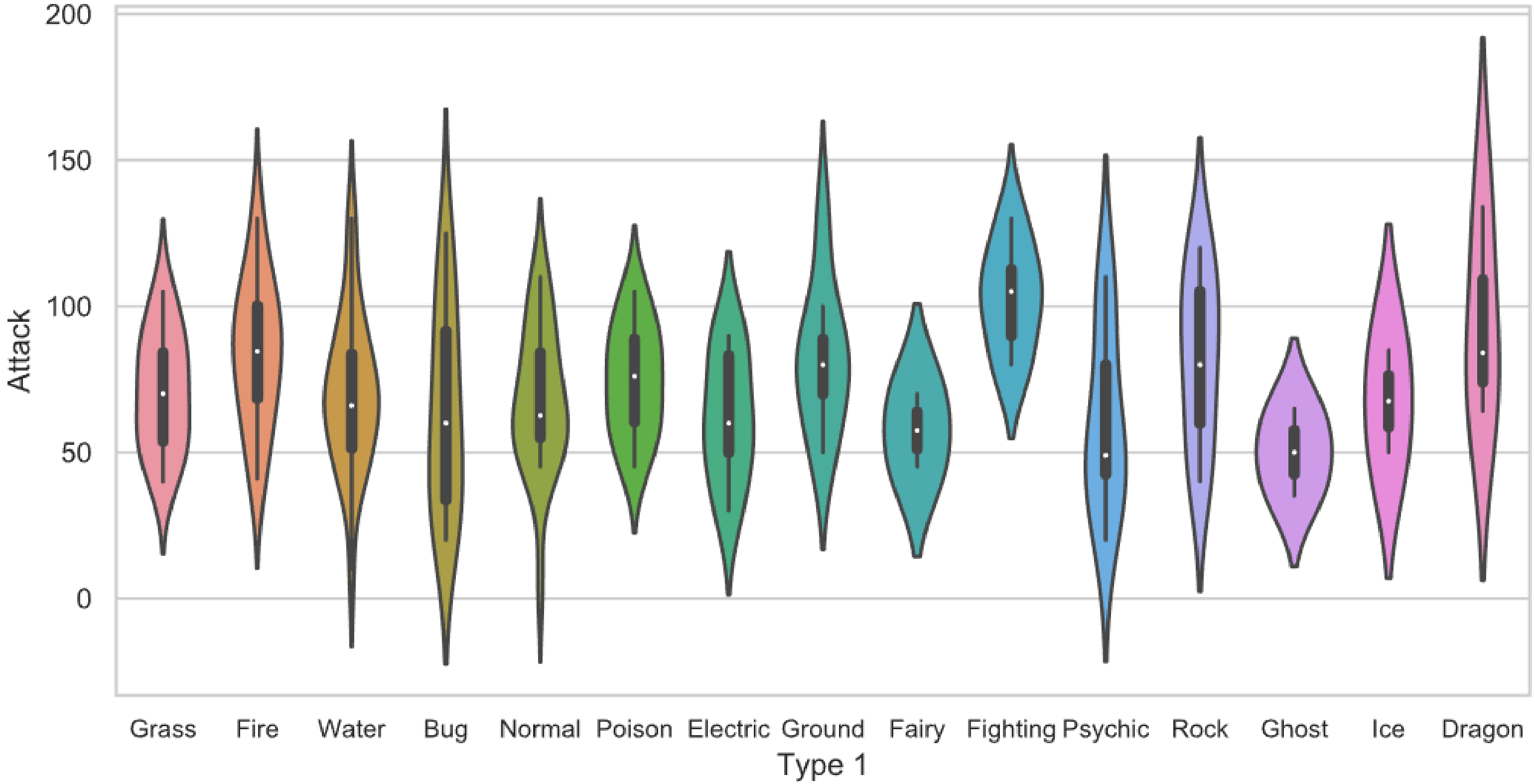



```
sns.lmplot(x='Attack', y='Defense',
    fit_reg=False, hue='Stage', data=df)
```



Violin Plot

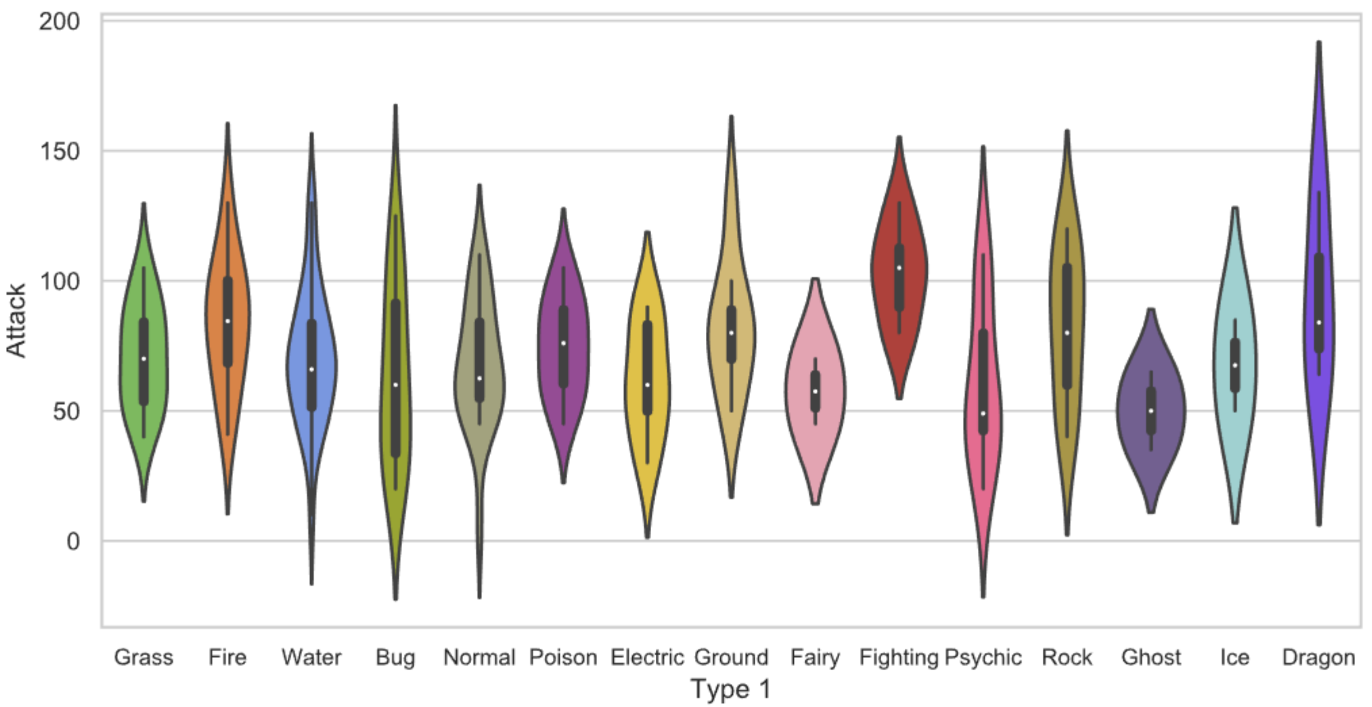
```
sns.set_style('whitegrid')
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.violinplot(x='Type 1', y='Attack', data=df)
```



Violin Plot (with Customized Colors)

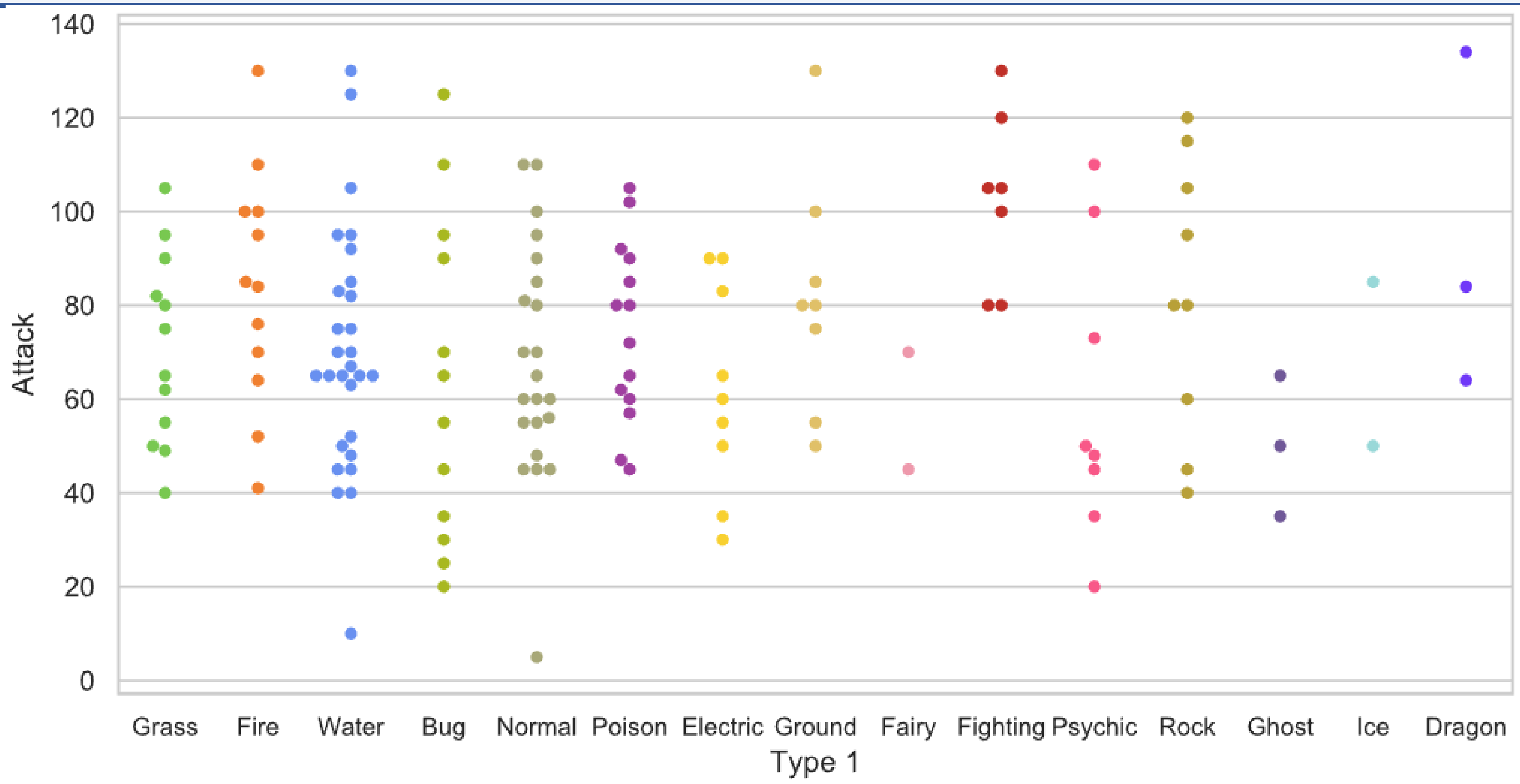
```
poke_colors = ['#78C850',  # Grass
               '#F08030',  # Fire
               '#6890F0',  # Water
               '#A8B820',  # Bug
               '#A8A878',  # Normal
               '#A040A0',  # Poison
               '#F8D030',  # Electric
               '#E0C068',  # Ground
               '#EE99AC',  # Fairy
               '#C03028',  # Fighting
               '#F85888',  # Psychic
               '#B8A038',  # Rock
               '#705898',  # Ghost
               '#98D8D8',  # Ice
               '#7038F8',  # Dragon
               ]
```

```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.violinplot(x='Type 1', y='Attack', data=df,
   palette=poke_colors)
```



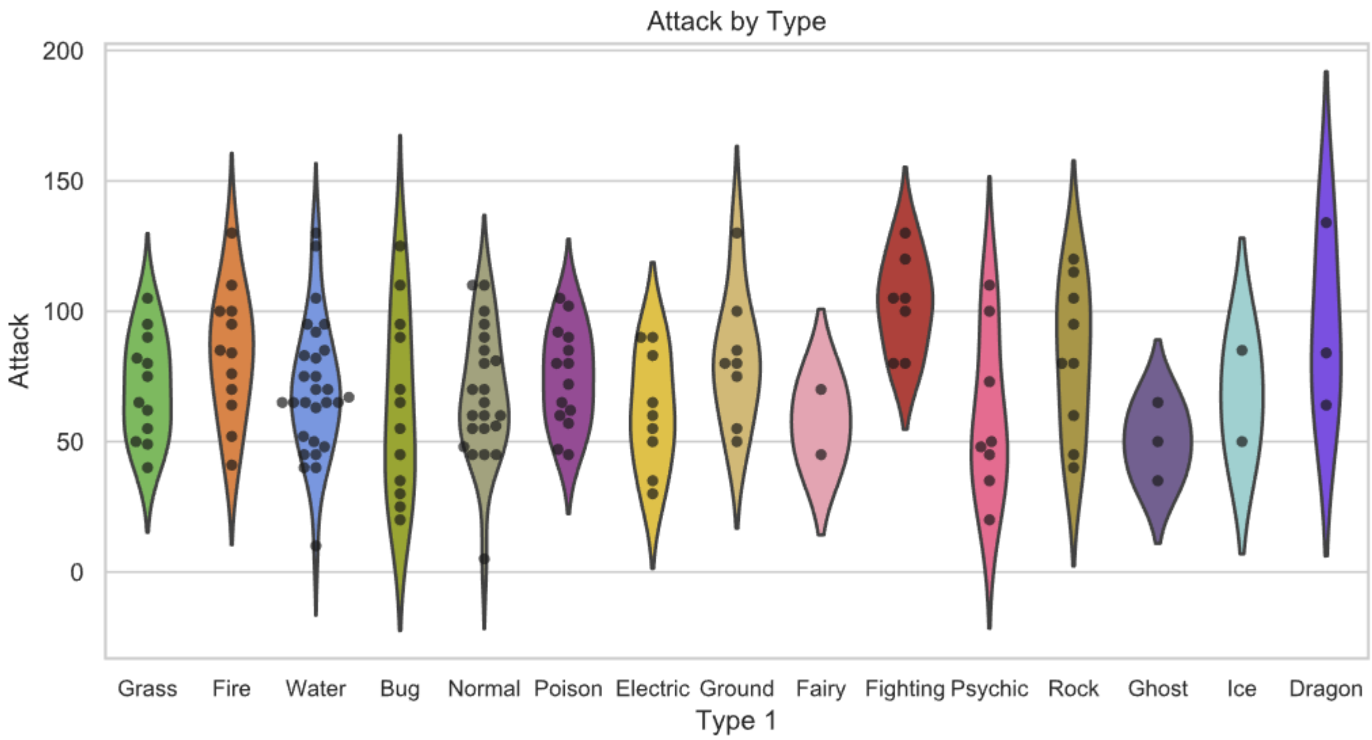
Swarm Plot

```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.swarmplot(x='Type 1', y='Attack', data=df, palette=poke_colors)
```



Violin + Swarm Plot

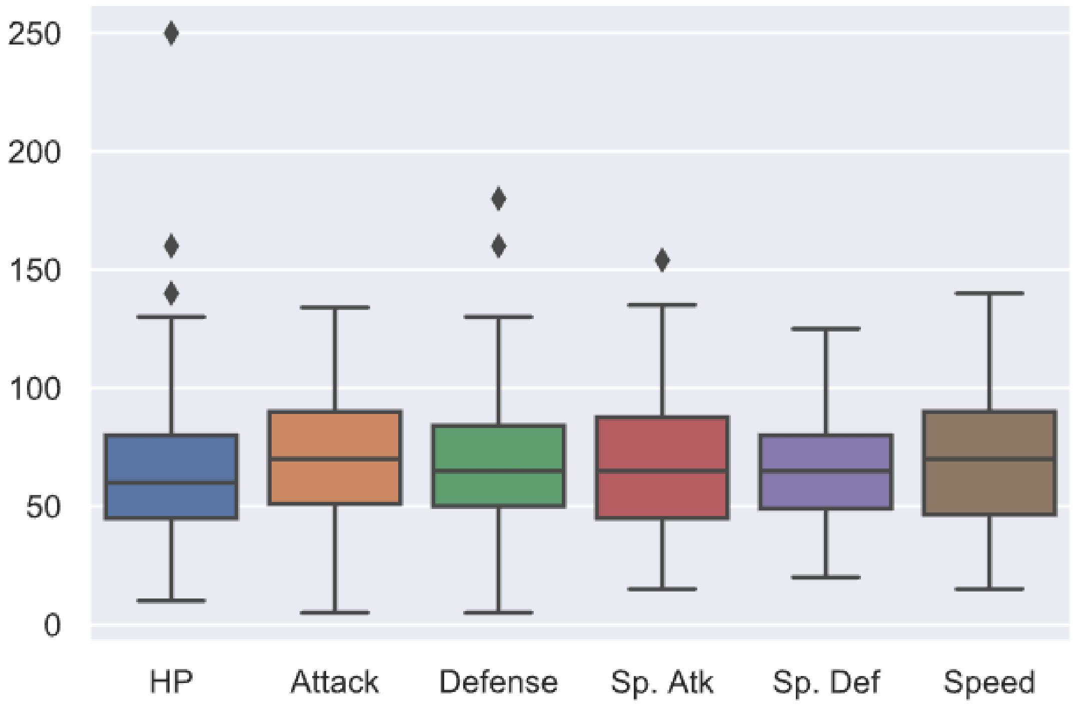
```
plt.figure(figsize=(10, 5))
plt.xticks(fontsize=10)
sns.violinplot(x='Type 1', y='Attack', data=df, palette=poke_colors, inner=None)
sns.swarmplot(x='Type 1', y='Attack', data=df, color='k', alpha=0.7)
plt.title('Attack by Type')
```

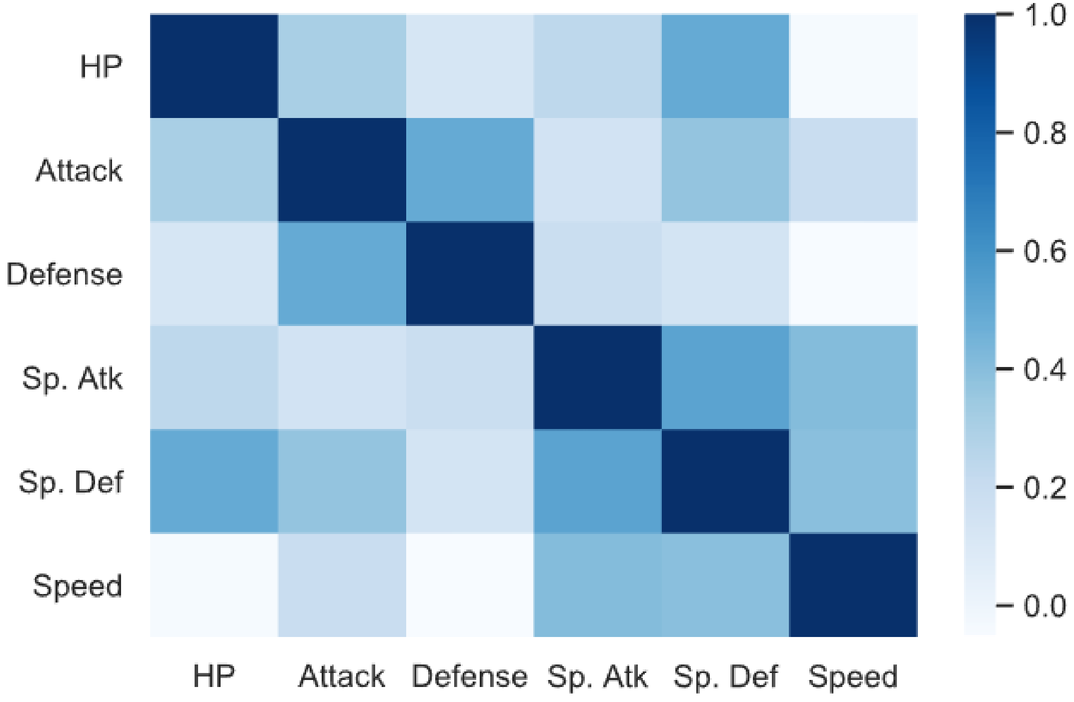


Box Plot and Heatmap

```
stats_df = df.drop(['Total', 'Stage',
                    'Legendary'], axis=1)
sns.boxplot(data=stats_df)
```

```
corr = stats_df.corr()
sns.heatmap(corr, cmap='Blues')
```

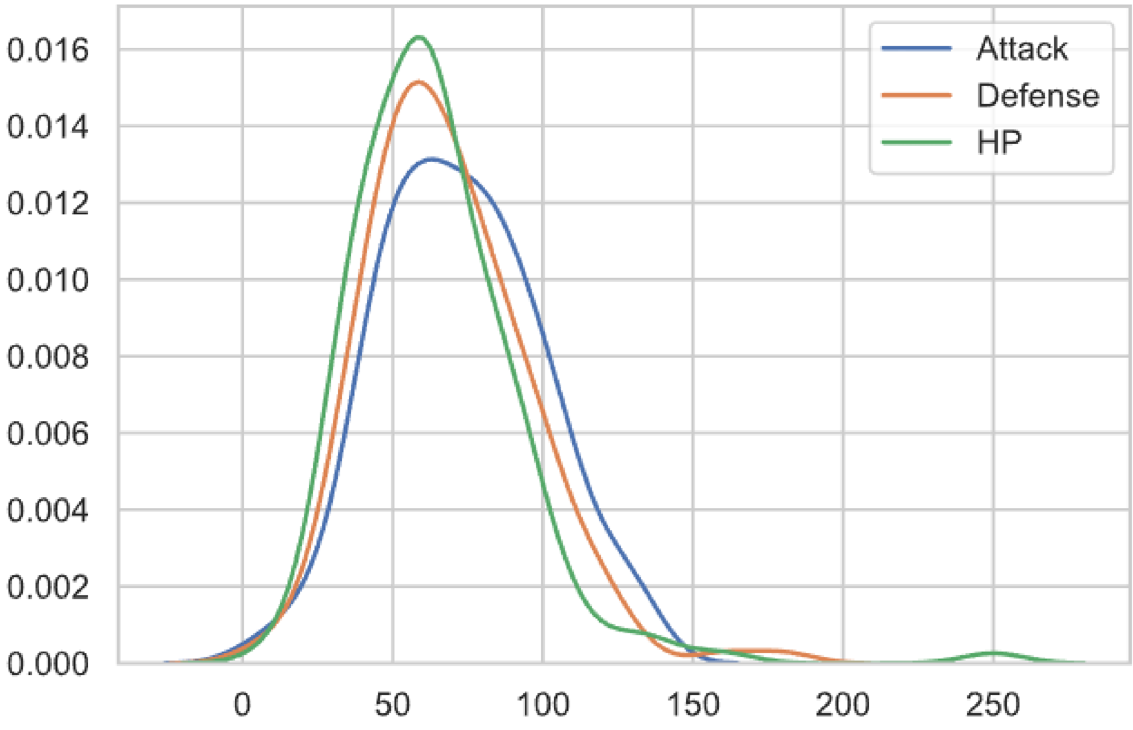


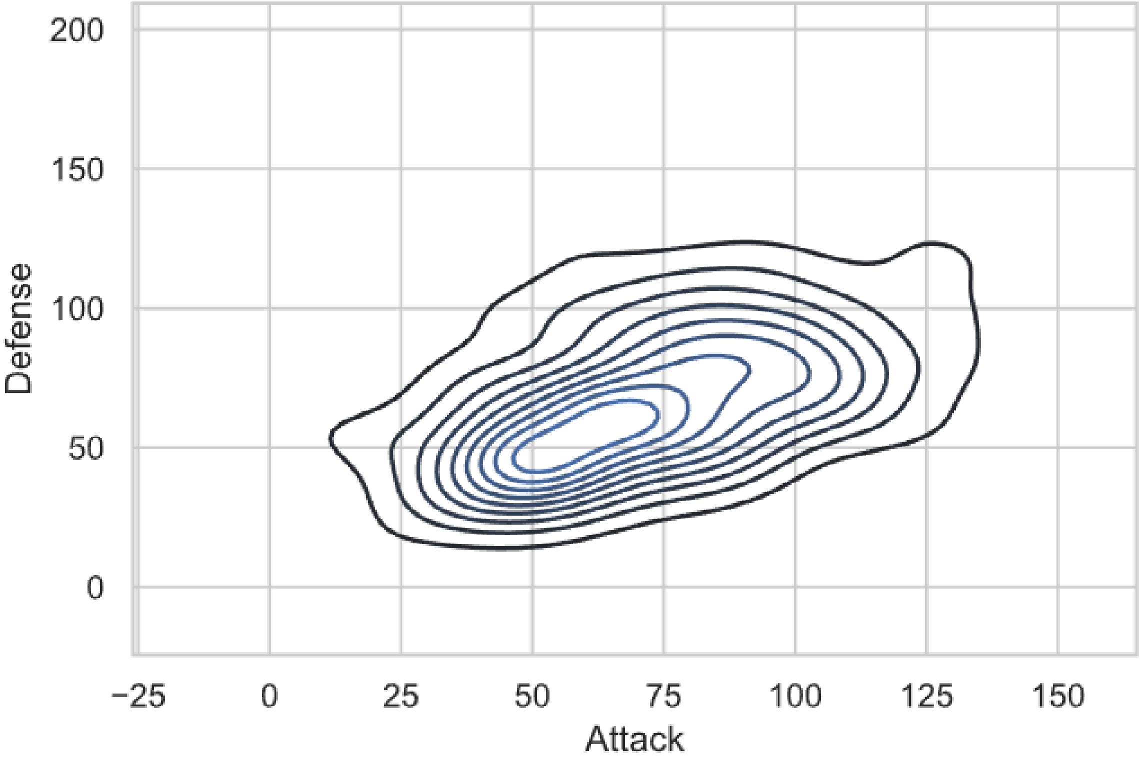


KDE Plot

```
sns.kdeplot(df.Attack)
sns.kdeplot(df.Defense)
sns.kdeplot(df.HP)
```

```
sns.kdeplot(df.Attack, df.Defense)
```





Pairplot

```
s.pairplot(df[['Type 1',
               'Attack',
               'Defense',
               'HP',
               'Speed']],
           hue='Type 1')
```