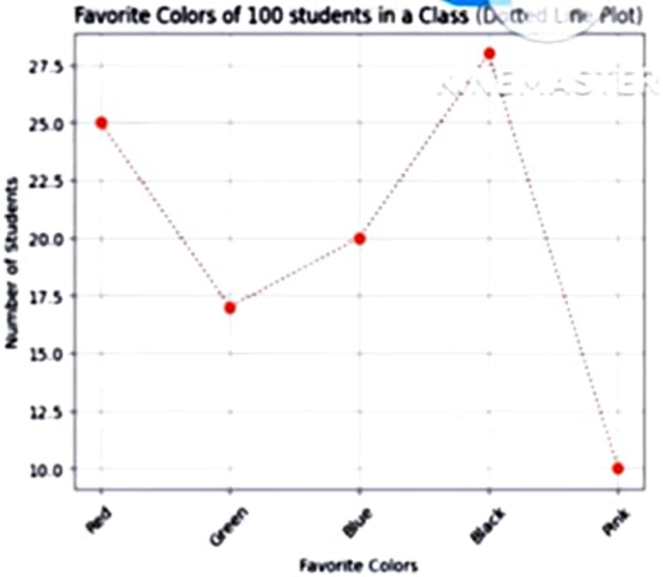
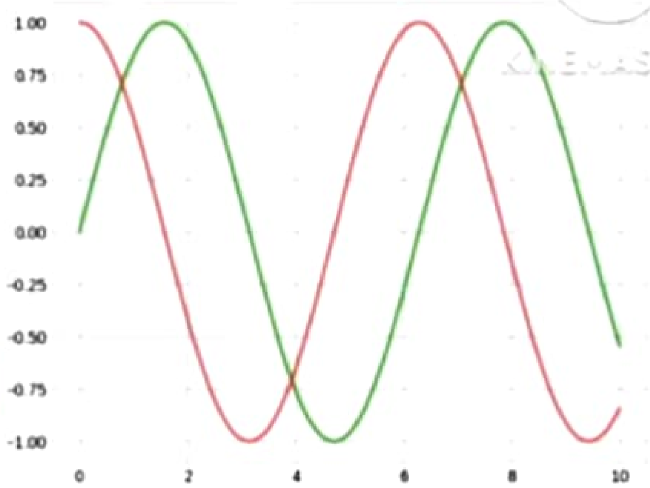
```python
import matplotlib.pyplot as plt
# Data
colors = ['Red', 'Green', 'Blue', 'Black', 'Pink']
students = [25, 17, 20, 28, 10]
# Create a line plot with dotted lines
plt.plot(colors, students, linestyle=':', marker='o', color='Red')
# Add labels and title
plt.xlabel('Favorite Colors')
plt.ylabel('Number of Students')
plt.title('Favorite Colors of 100 students in a Class (Dotted Line Plot)')
# Display the chart
plt.grid(True)  # Add grid lines
plt.xticks(rotation=45)  # Rotate x-axis labels for better visibility
plt.show()
```



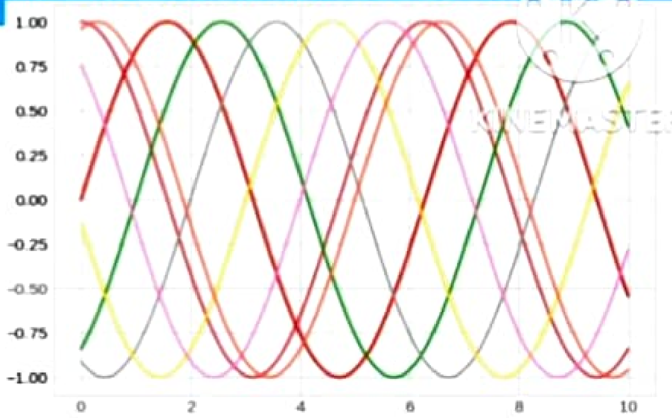
line plot

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
fig = plt.figure()
ax = plt.axes()
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
plt.plot(x, np.sin(x));
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
```



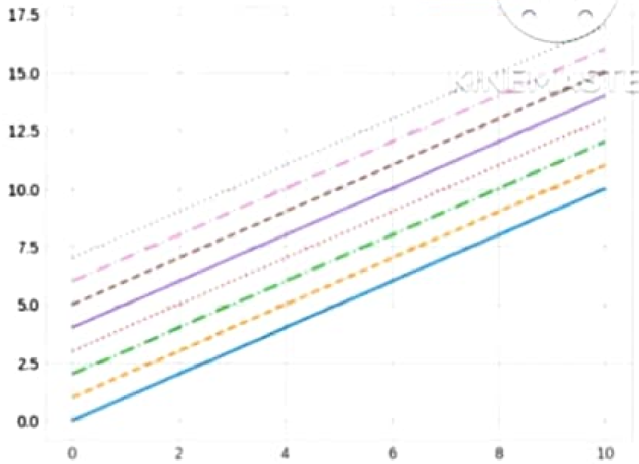
Adjusting the Plot: Line Colors and Styles

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
fig = plt.figure()
ax = plt.axes()
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
plt.plot(x, np.sin(x));
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
plt.plot(x, np.sin(x - 0), color='red') #  color by name
plt.plot(x, np.sin(x - 1), color='g') #  color code (rgb)
plt.plot(x, np.sin(x - 2), color='0.58') # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44') # Hexcode
plt.plot(x, np.sin(x - 4), color=(1.0,0.5,0.8)) # RGB  values 0 and 1
plt.plot(x, np.sin(x - 5), color='tomato'); #  HTML color names
```



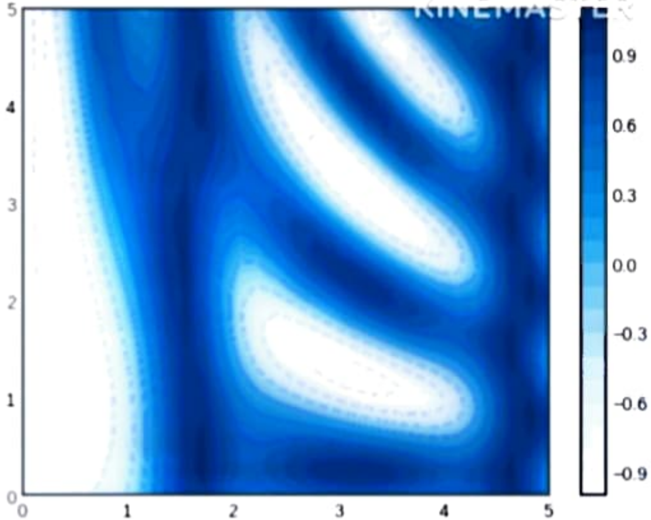
line style using the linestyle keyword

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
fig = plt.figure()
ax = plt.axes()
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
plt.plot(x, x + 4, linestyle='-') # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':'); # dotted
```



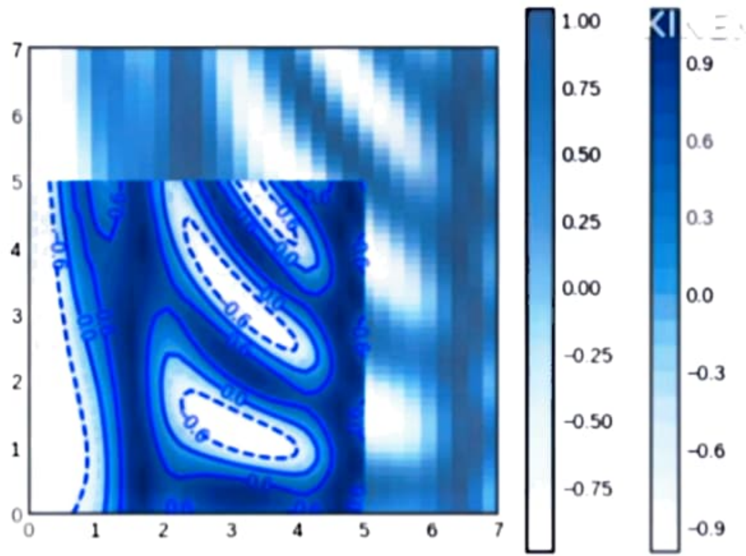
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
fig = plt.figure()
ax = plt.axes()
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
plt.plot(x, x + 0, '-g') # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r'); # dotted red
```

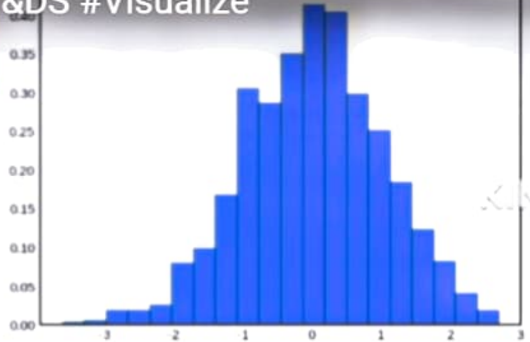
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np
def f(x, y):
  return np.sin(x) ** 12 + np.cos(15 + y * x) * np.cos(x)
x = np.linspace(0, 5, 30)
y = np.linspace(0, 5, 60)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contour(X, Y, Z, colors='blue');
plt.contour(X, Y, Z, 20, cmap='Blues');
plt.contourf(X, Y, Z, 20, cmap='Blues')
plt.colorbar();
```



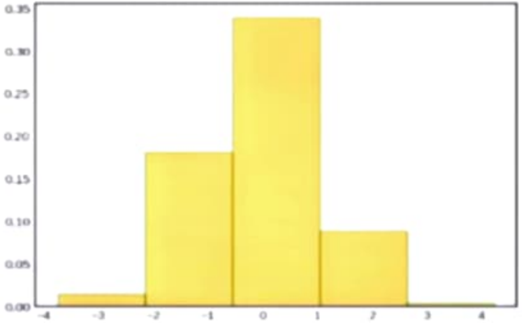
blue regions are "peaks," while the white regions are "valleys."

```python
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np
def f(x, y):
  return np.sin(x) ** 12 + np.cos(15 + y * x) * np.cos(x)
x = np.linspace(0, 5, 30)
y = np.linspace(0, 5, 60)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contour(X, Y, Z, colors='blue');
plt.contour(X, Y, Z, 20, cmap='Blues');
plt.contourf(X, Y, Z, 20, cmap='Blues')
plt.colorbar();
contours = plt.contour(X, Y, Z, 3, colors='blue')
plt.clabel(contours, inline=True, fontsize=10)
plt.imshow(Z, extent=[0, 7, 0, 7], origin='lower', cmap='Blues', alpha=0.7)
plt.colorbar();
```





```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data, bins=20, density=True, alpha=0.5, histtype='bar', color='blue', edgecolor='green')
plt.show()
```



```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data, bins=5, density=True, alpha=0.5, histtype='bar', color='orange', edgecolor='green')
plt.show()
```

subplots

Subplot- groups of smaller axes that can exist together within a single figure.

plt.axes function.

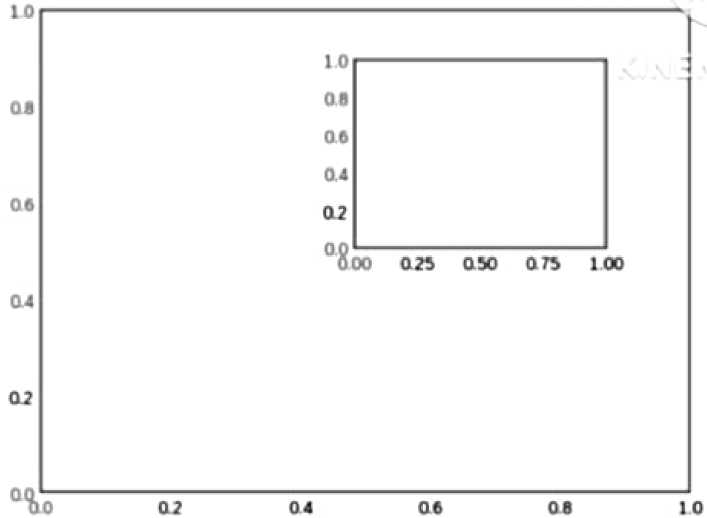
[bottom, left, width,height] in the figure coordinate system,

ranges from 0 at the bottom left of the figure to 1 at the top right

Subplots in Matplotlib allow you to create multiple plots within the same figure. They are useful when you want to display different views of the data or compare multiple datasets side by side

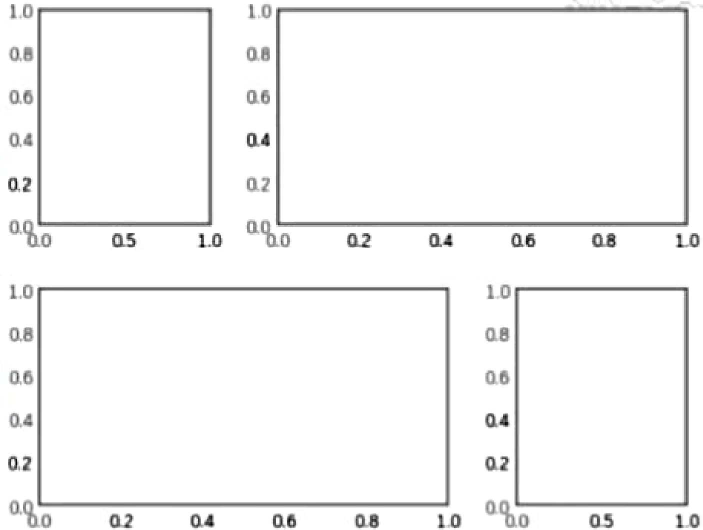
plt.axes

```
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np
ax1 = plt.axes() # standard axes
ax2 = plt.axes([0.50, 0.50, 0.3, 0.3])
```

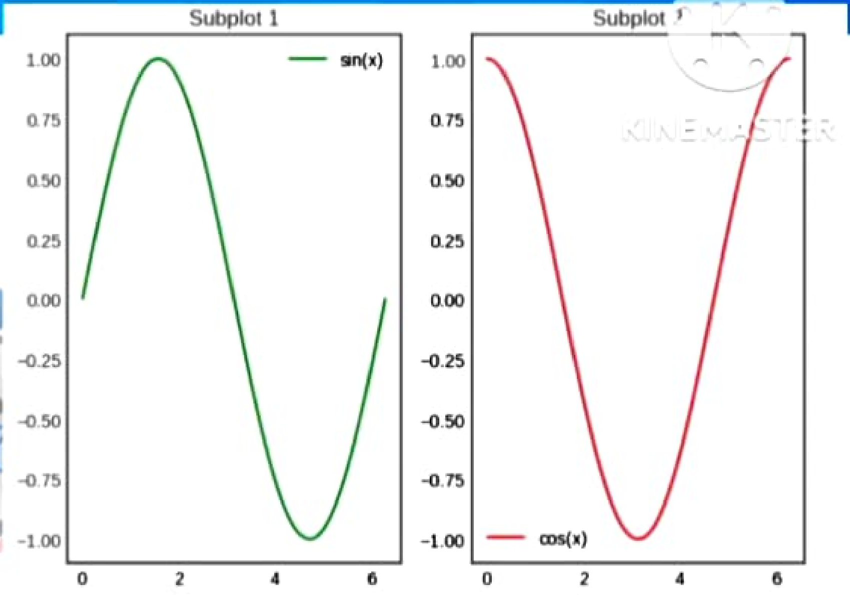


plt.subplot

```
import matplotlib.pyplot as plt
import numpy as np
grid = plt.GridSpec(2, 3, wspace=0.4, hspace=0.3)
plt.subplot(grid[0, 0])
plt.subplot(grid[0, 1:])
plt.subplot(grid[1, :2])
plt.subplot(grid[1, 2]);
# Show the plot
plt.show()
```

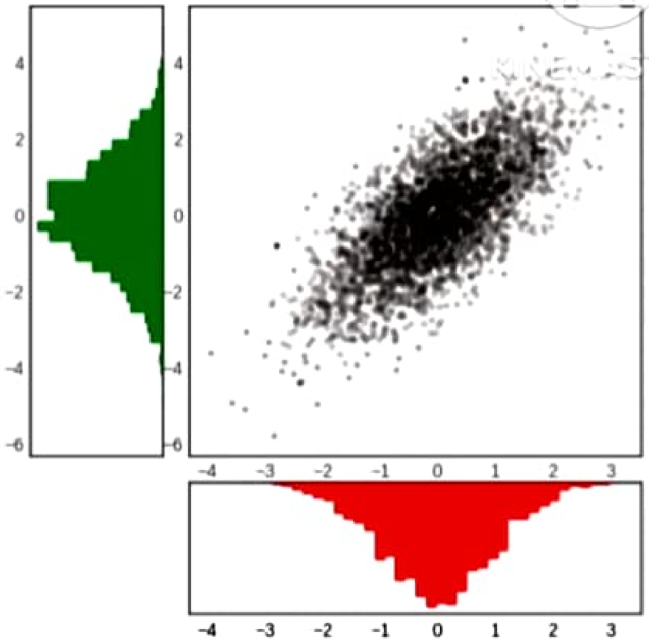


```
import matplotlib.pyplot as plt
import numpy as np
# Sample data
x = np.linspace(0, 2 * np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
# Create a 1x2 grid of subplots
plt.subplot(1, 2, 1)  # 1 row, 2 columns, first subplot
plt.plot(x, y1, label='sin(x)',color='green')
plt.title('Subplot 1')
plt.legend()
plt.subplot(1, 2, 2)  # 1 row, 2 columns, second subplot
plt.plot(x, y2, label='cos(x)', color='red')
plt.title('Subplot 2')
plt.legend()
# Adjust layout to prevent overlapping titles and labels
plt.tight_layout()
# Show the plot
plt.show()
```



Multiple subplots

```
import matplotlib.pyplot as plt
import numpy as np
# Show the plot
plt.show()
mean = [0,0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 3000).T
 # Set up the axes with gridspec
fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)
main_ax = fig.add_subplot(grid[:-1, 1:])
y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)
 # scatter points on the main axes
main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)
 # histogram on the attached axes
x_hist.hist(x, 40, histtype='stepfilled',
orientation='vertical', color='red')
x_hist.invert_yaxis()
y_hist.hist(y, 40, histtype='stepfilled',
orientation='horizontal', color='green')
y_hist.invert_xaxis()
```



Text and Annotation

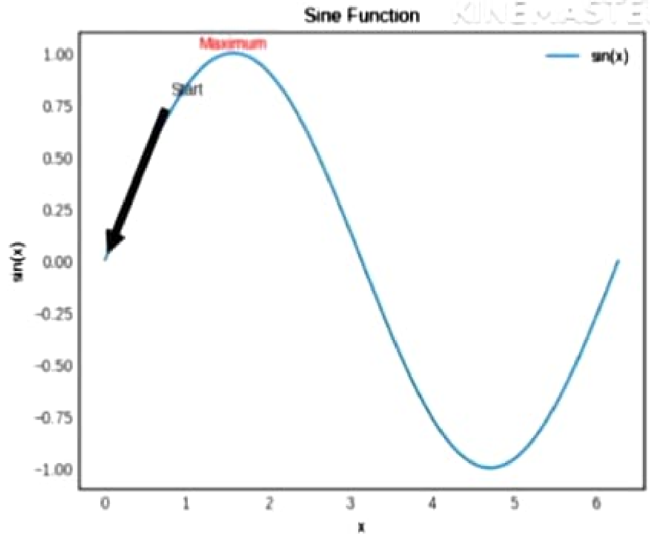
Text and annotations in Matplotlib are used to add additional information, labels, or comments to your plots. textual cues and labels are necessary annotate to help convey interesting information

can customize the position, appearance, and content of text and annotations based on your specific needs

lines, ticks, and labels that make up the axes.

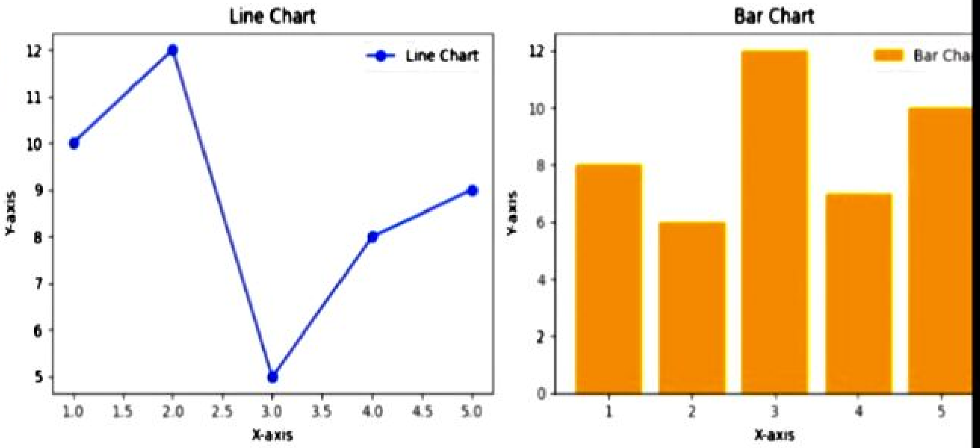
Text and annotation

```
import matplotlib.pyplot as plt
import numpy as np
# Sample data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
# Create a plot
plt.plot(x, y, label='sin(x)')
plt.title('Sine Function')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.legend()
# Add text at a specific point
plt.text(np.pi/2, 1, 'Maximum', ha='center', va='bottom', color='red', fontsize=10)
# Add an annotation with an arrow
plt.annotate('Start', xy=(0, 0), xytext=(1, 0.8),
        arrowprops=dict(facecolor='black', shrink=0.05),
        fontsize=10, ha='center')
# Show the plot
plt.show()
```

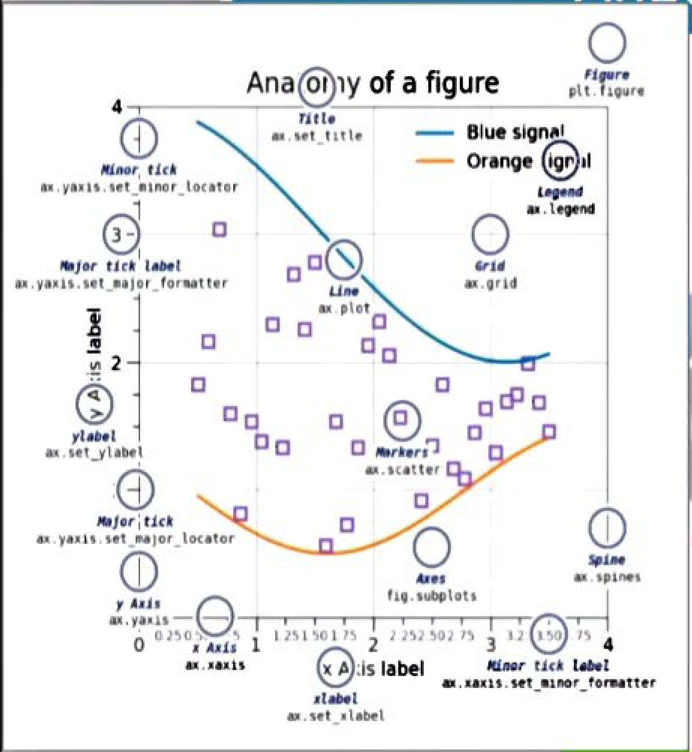


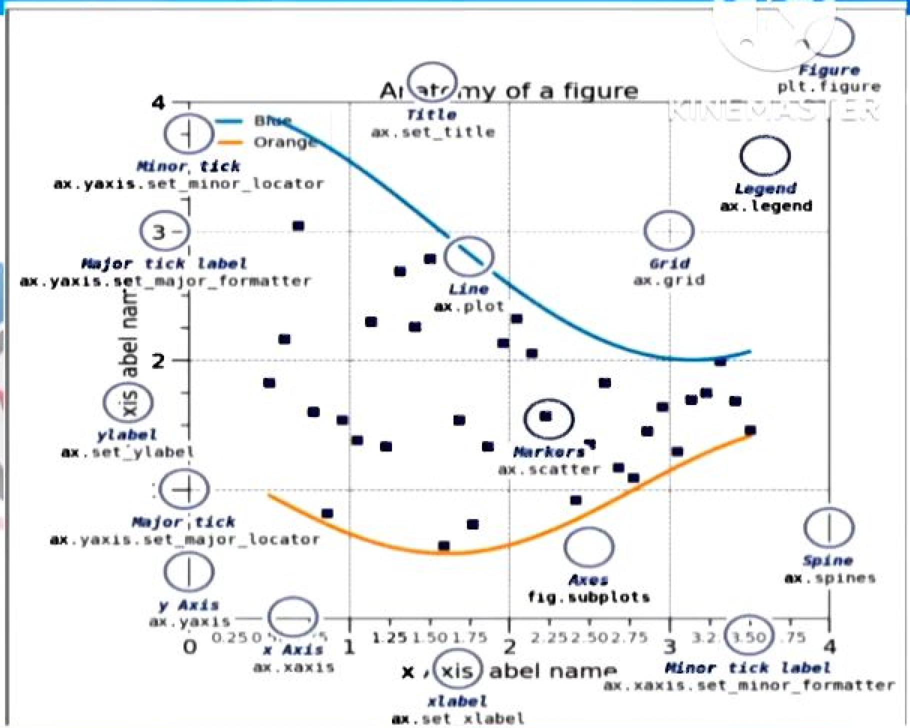
line plot and bar plot as subplot for same data

```
import matplotlib.pyplot as plt
import numpy as np
# Sample data
x = np.arange(1, 6)
y_line = np.array([10, 12, 5, 8, 9])
y_bar = np.array([8, 6, 12, 7, 10])
# Create a figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
# Subplot 1: Line chart
ax1.plot(x, y_line, label='Line Chart', marker='o', color='blue')
ax1.set_title('Line Chart')
ax1.set_xlabel('X-axis')
ax1.set_ylabel('Y-axis')
ax1.legend()
# Subplot 2: Column (Bar) chart
ax2.bar(x, y_bar, label='Bar Chart', color='orange')
ax2.set_title('Bar Chart')
ax2.set_xlabel('X-axis')
ax2.set_ylabel('Y-axis')
ax2.legend()
# Adjust layout to prevent overlapping titles and labels
plt.tight_layout()
# Show the plot
plt.show()
```



Anatomy of figure





Customizing Matplotlib

To control the appearance of plots

Customizing Plot Appearance

- Setting Line Styles and Colors
- Adding Labels and Title
- Changing Fonts and Font Sizes
- Adding Legends

Customizing Axes

- Setting Axis Limits
- Changing Tick Locations and Labels
- Setting Axis Scale
- Adding Grid

Saving and Displaying Plots

- Saving Plots to File
- Displaying Plots

Customizing Plot Appearance

Setting Line Styles and Colors:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [2, 4, 6, 8]
plt.plot(x, y, linestyle='--', color='red', marker='o', label='Line 1')
```

Adding Labels and Title

```
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Title of the Plot')
```

Changing Fonts and Font Sizes

```
plt.rcParams.update({'font.size': 12, 'font.family': 'Arial'})
```

Adding Legends

```
plt.legend()
```

Customizing Axes

Setting Axis Limits

```
plt.xlim(0, 5)
plt.ylim(0, 10)
```

Changing Tick Locations and Labels

```
plt.xticks([1, 2, 3, 4], ['A', 'B', 'C', 'D'])
```

Setting Axis Scale

```
plt.yscale('log')  # Other options: 'linear', 'log', 'symlog', 'logit'
```

Adding Grid

```
plt.grid(True)
```

Saving and Displaying Plots

Saving Plots to File

plt.savefig('plot.png', dpi=300)

Displaying Plots

plt.show()

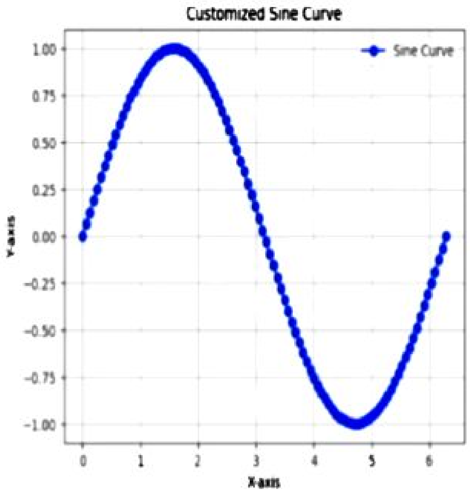
runtime configuration (rc) containing the default styles for every plot element you create.

rc parameters

```
plt.rc('axes', facecolor='#E6E6E6', edgecolor='none',
axisbelow=True, grid=True, prop_cycle=colors)
plt.rc('grid', color='w', linestyle='solid')
plt.rc('xtick', direction='out', color='gray')
plt.rc('ytick', direction='out', color='gray')
plt.rc('patch', edgecolor='#E6E6E6')
plt.rc('lines', linewidth=2)
```

Sample code


```
import matplotlib.pyplot as plt
import numpy as np
# Generate some sample data
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)
# Plot customization
plt.plot(x, y, label='Sine Curve', linestyle='-', color='blue', marker='o')
plt.title('Customized Sine Curve')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.legend()
plt.grid(True)
# Save the plot
plt.savefig('customized_plot.png', dpi=300)
# Display the plot
plt.show()
```



rich features for customization

Creating Subplots
```
plt.subplot(2, 1, 1)  # 2 rows, 1 column, plot 1
plt.plot(x, y)
```

Adding Text
```
plt.text(1, 0.5, 'Text Example', fontsize=12, color='green')
```

Adding Annotations
```
plt.annotate('Maximum', xy=(np.pi / 2, 1), xytext=(np.pi / 2, 1.5),
arrowprops=dict(facecolor='red', shrink=0.05))
```

Using Colormaps
```
plt.imshow(data, cmap='viridis', interpolation='none')
plt.colorbar()
```

Using Styles
```
plt.style.use('seaborn-darkgrid')
```

Setting Figure Size
```
plt.figure(figsize=(8, 4))
```

Adjusting Subplot Spacing
```
plt.subplots_adjust(left=0.1, right=0.9, top=0.9, bottom=0.1)
```

Plotting Histograms
```
plt.hist(data, bins=30, alpha=0.5, color='steelblue', edgecolor='black')
```

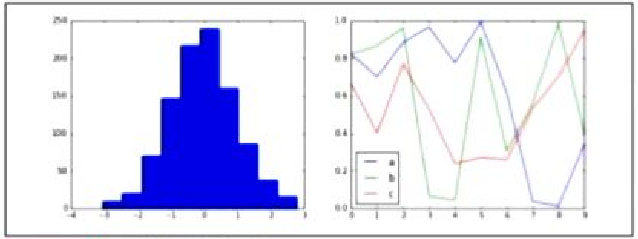
Error Bars
```
plt.errorbar(x, y, yerr=errors, fmt='-o', color='orange', ecolor='red', capsize=5)
```

Creating 3D Plotsfrom
```
mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot3D(x, y, np.sin(x))
```

Matplotlib documentation.

Virtual classroom Lectures

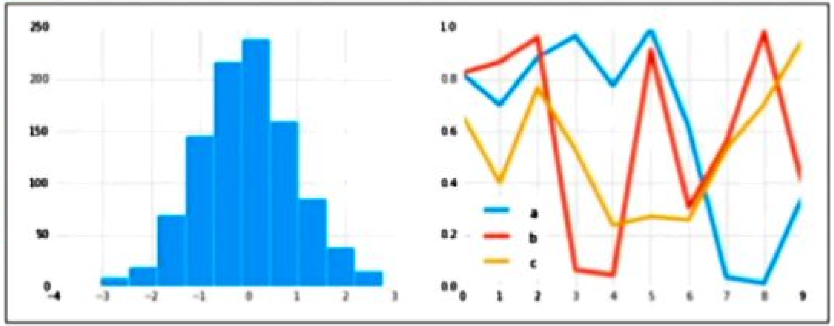
Stylesheets-default &FiveThirtyEight style



[redacted] *Matplotlib's default style*

bold colors, thick lines, and transparent axes

FiveThirtyEight style

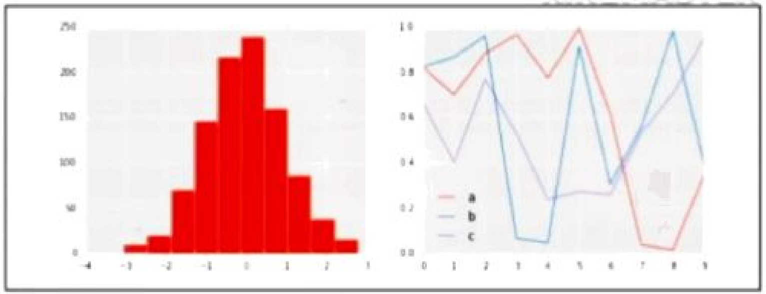



```
with plt.style.context('fivethirtyeight'):
 hist_and_lines()
```

Stylesheets-ggplot & bmh

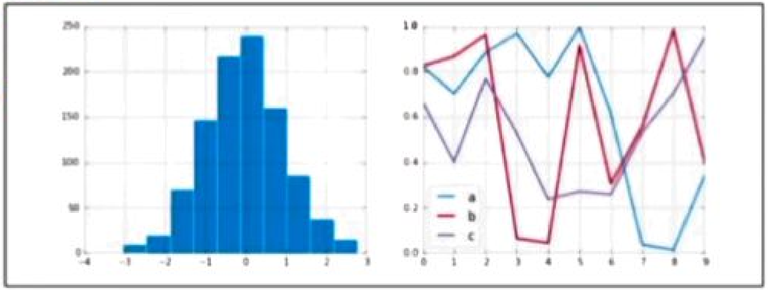
ggplot

```
with plt.style.context('ggplot'):
hist_and_lines()
```



Bayesian Methods for Hackers style

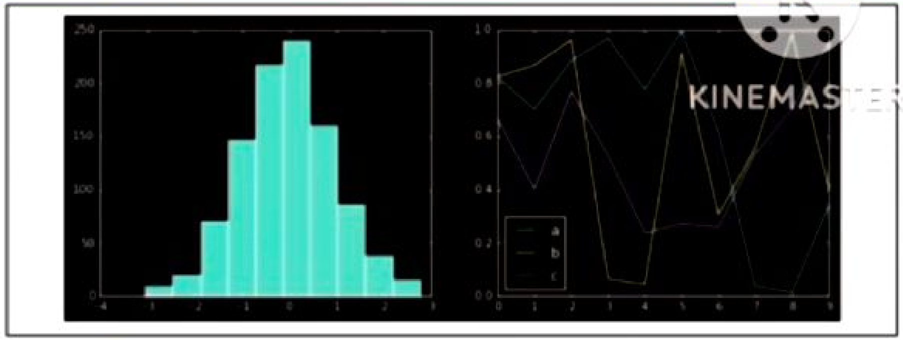
```
with plt.style.context('bmh'):
hist_and_lines()
```



Dark background

with plt.style.context('dark_background'

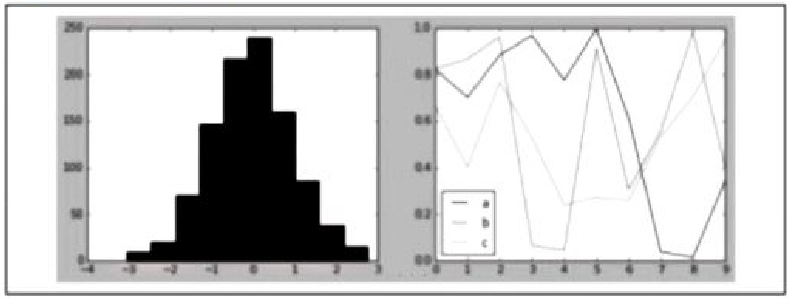
hist_and_lines()



Grayscale

with plt.style.context('grayscale'):

hist_and_lines()

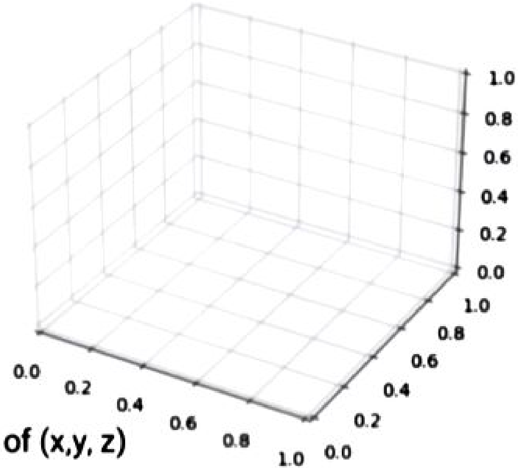


Three-Dimensional Plotting in Matplotlib

Three-dimensional plots by importing the mplot3d toolkit

Three-dimensional plotting is viewing figures interactively

```
from mpl_toolkits import mplot3d
import numpy as np
 import matplotlib.pyplot as plt
fig = plt.figure()
 ax = plt.axes(projection='3d')
```



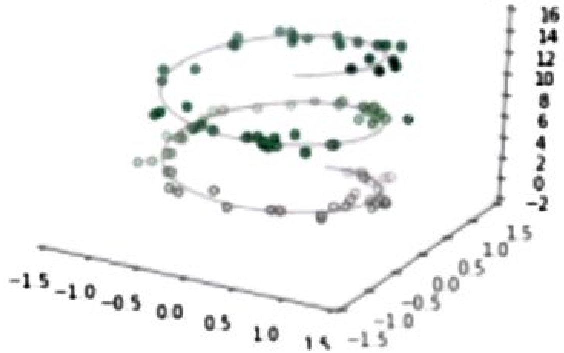
three-dimensional plot is a line or scatter plot created from sets of (x,y, z) triples.

ax.plot3D

ax.scatter3D functions

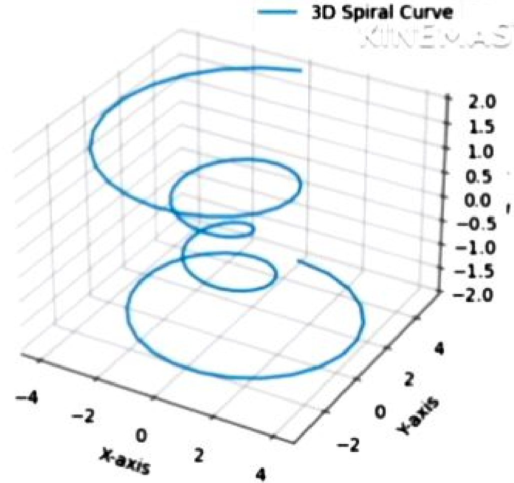
Points and lines in three dimensions

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
ax = plt.axes(projection='3d')
 # Data for a three-dimensional line
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)
ax.plot3D(xline, yline, zline, 'gray')
 # Data for three-dimensional
scattered points
zdata = 15 * np.random.random(100)
xdata = np.sin(zdata) + 0.1 *
np.random.randn(100)
ydata = np.cos(zdata) + 0.1 *
np.random.randn(100)
ax.scatter3D(xdata, ydata, zdata,
c=zdata, cmap='Greens');
```



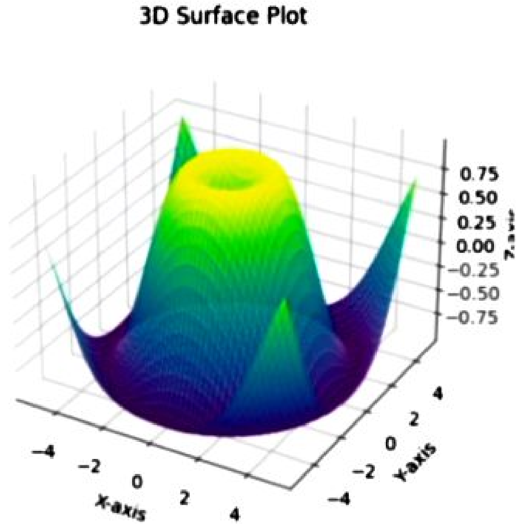
3D Spiral Curve

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
# Create a figure and a 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
# Generate some sample data
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
# Plot the 3D curve
ax.plot(x, y, z, label='3D Spiral Curve')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.legend()
# Show the plot
plt.show()
```



3D Surface Plot


```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
# Create a figure and a 3D axis
fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
# Generate sample data
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
x,y = np.meshgrid(x,y)
z = np.sin(np.sqrt(x**2 + y**2))
# Plot a 3D surface
ax.plot_surface(x, y, z, cmap='viridis')
# Add labels
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
ax.set_title('3D Surface Plot')
# Show the plot
plt.show()
```



figsize : This parameter is the Figure dimension (width, height) in inches.
dpi : This parameter is the dots per inch.
facecolor : This parameter is the figure patch facecolor.
edgecolor : This parameter is the figure patch edge color.
linewidth : This parameter is the linewidth of the frame.
frameon : This parameter is the suppress drawing the figure background patch.
subplotpars : This parameter is the Subplot parameters.
tight_layout : This parameter is used to adjust subplot parameters.
constrained_layout : This parameter is used to adjust positioning of plot elements.

Geographic Data with Basemap

Geographic Data with Basemap

Matplot-lib's main tool for this type of visualization is the Basemap toolkit

Basemap is a useful tool for Python users to have in their virtual toolbelts.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None, lat_0=50, lon_0=-100)
m.bluemarble(scale=0.5);
```

geographic visualizations

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
        width=8E6, height=8E6,
        lat_0=45, lon_0=-100,)
m.etopo(scale=0.5, alpha=0.5)

# Map (long, lat) to (x, y) for plotting
x, y = m(-122.3, 47.6)
plt.plot(x, y, 'ok', markersize=5)
plt.text(x, y, ' Seattle', fontsize=12);
```

Map Projections

(e.g., direction, area, distance, shape, or other considerations)- **map along with the longitude and latitude lines**

```
from itertools import chain
def draw_map(m, scale=0.2):
    # draw a shaded-relief image
    m.shadedrelief(scale=scale)
        # lats and longs are returned as a dictionary
    lats = m.drawparallels(np.linspace(-90, 90, 13))
    lons = m.drawmeridians(np.linspace(-180, 180, 13))
    # keys contain the plt.Line2D instances
    lat_lines = chain(*(tup[1][0] for tup in lats.items()))
    lon_lines = chain(*(tup[1][0] for tup in lons.items()))
    all_lines = chain(lat_lines, lon_lines)
        # cycle through these lines and set the desired style
    for line in all_lines:
        line.set(linestyle='-', alpha=0.3, color='
```

Cylindrical projections

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='cyl', resolution=None,
            llcrnrlat=-90, urcrnrlat=90,
            llcrnrlon=-180, urcrnrlon=180, )
draw_map(m)
```

cylindrical projections are the Mercator (projection='merc') and the cylindrical equal area (projection='cea') projections.
latitude (lat) and longitude (lon) of the lower-left corner (llcrnr) and upper-right corner (urcrnr) for the desired map

Pseudo-cylindrical projections

Pseudo-cylindrical projections

pseudo-cylindrical projections are the sinusoidal (projection='sinu') and Robinson (projection='robin') projections. Mollweide projection (projection='moll') is one common example

```
fig = plt.figure(figsize=(8, 6), edgecolor='w')
m = Basemap(projection='moll', resolution=None,
        lat_0=0, lon_0=0)
draw_map(m)
```

Perspective projections

Perspective projections
 orthographic projection (projection='ortho')
gnomonic projection (projection='gnom') and
stereographic projection (projection='stere').

example of the orthographic projection

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='ortho', resolution=None,
          lat_0=50, lon_0=0)
draw_map(m);
```

Conic projections

Conic projections

Lambert Conformal Conic projection (projection='lcc'), which we saw earlier in the map of North America. It projects the map onto a cone arranged in such a way that two standard parallels (specified in Basemap by lat_1 and lat_2)

conic projections are the equidistant conic projection (projection='eqdc') and the Albers equal-area projection (projection='aea'). Conic projections, like perspective projections

```
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution=None,
        lon_0=0, lat_0=50, lat_1=45, lat_2=55,
        width=1.6E7, height=1.2E7)
draw_map(m)
```

Drawing a Map Background

The Basemap package contains a range of useful functions for drawing borders of physical features like continents, oceans, lakes, and rivers, as well as political boundaries such as countries and US states and counties

Physical boundaries and bodies of water

drawcoastlines(): Draw continental coast lines

drawlsmask(): Draw a mask between the land and sea, for use with projecting images on one or the other

drawmapboundary(): Draw the map boundary, including the fill color for oceans.

drawrivers(): Draw rivers on the map

fillcontinents(): Fill the continents with a given color; optionally fill lakes with another color

Political boundaries

drawcountries(): Draw country boundaries

drawstates(): Draw US state boundaries

drawcounties(): Draw US county boundaries

Map features

drawgreatcircle(): Draw a great circle between two points

drawparallels(): Draw lines of constant latitude

drawmeridians(): Draw lines of constant longitude

drawmapscale(): Draw a linear scale on the map

Whole-globe images

bluemarble(): Project NASA's blue marble image onto the map

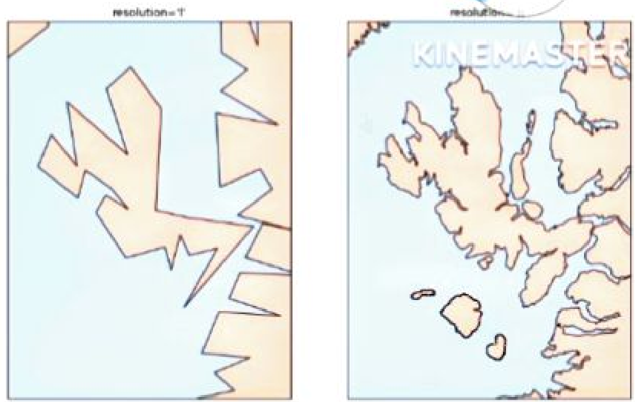
shadedrelief(): Project a shaded relief image onto the map

etopo(): Draw an etopo relief image onto the map

warpimage(): Project a user-provided image onto the map

```
fig, ax = plt.subplots(1, 2, figsize=(12, 8))

for i, res in enumerate(['l', 'h']):
    m = Basemap(projection='gnom', lat_0=57.3, lon_0=-6.2,
                width=90000, height=120000, resolution=res, ax=ax[i])
    m.fillcontinents(color="#FFDDCC", lake_color='#DDEEFF')
    m.drawmapboundary(fill_color="#DDEEFF")
    m.drawcoastlines()
    ax[i].set_title("resolution='{0}'".format(res));
```



Plotting Data on Maps

ome of these map-specific methods are:

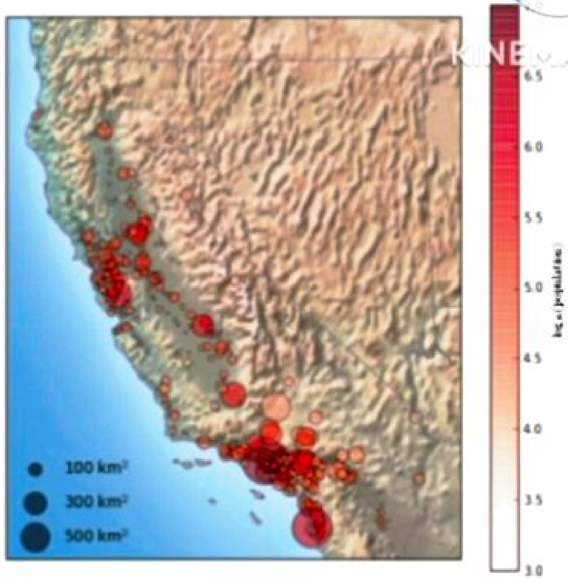
contour()/contourf() : Draw contour lines or filled contours
imshow(): Draw an image
pcolor()/pcolormesh() : Draw a pseudocolor plot for irregular/regular meshes
plot(): Draw lines and/or markers.
scatter(): Draw points with markers.
quiver(): Draw vectors.
barbs(): Draw wind barbs.
drawgreatcircle(): Draw a great circle.

Scatter plot over a map background

demonstrated the use of size and color in a scatter plot to convey information about the location, size, and population of California cities

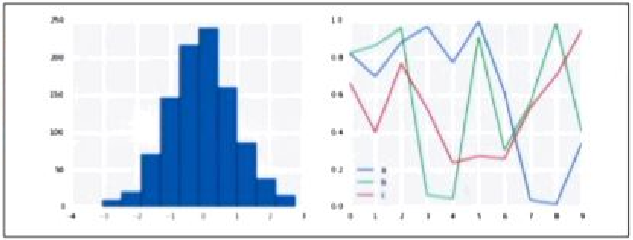
```
import pandas as pd
cities = pd.read_csv('data/california_cities.csv')
# Extract the data we're interested in
lat = cities['latd'].values
lon = cities['longd'].values
population = cities['population_total'].values
area = cities['area_total_km2'].values
# 1. Draw the map background
fig = plt.figure(figsize=(8, 8))
m = Basemap(projection='lcc', resolution='h',
lat_0=37.5, lon_0=-119,
width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color='gray')
m.drawcountries(color='gray')
m.drawstates(color='gray')
```

```
# 2. scatter city data, with color reflecting population
# and size reflecting area
m.scatter(lon, lat, latlon=True,
c=np.log10(population), s=area,
cmap='Reds', alpha=0.5)
# 3. create colorbar and legend
plt.colorbar(label=r'$\log_{10}({\rm population})$')
plt.clim(3, 7)
# make legend with dummy points
for a in [100, 300, 500]:
plt.scatter([], [], c='k', alpha=0.5, s=a,
label=str(a) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False,
labelspacing=1, loc='lower left');
```



Visualization with Seaborn

Seaborn provides an API on top of Matplot-lib that offers sane choices for plot style and color defaults, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas DataFrames.



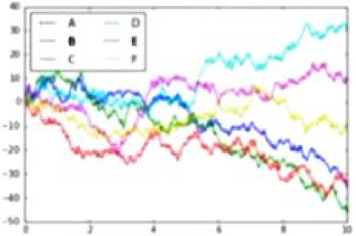
"Visualization with Seaborn

import seaborn

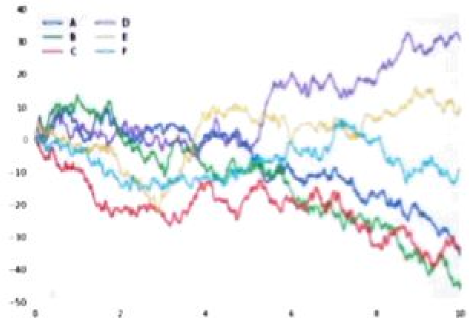
hist_and_lines()

Matplotlib's default style Vs Seaborn's default style

```
import matplotlib.pyplot as plt
plt.style.use('classic')
import numpy as np
import pandas as pd
# Create some data
rng = np.random.RandomState(0)
x = np.linspace(0, 10, 500)
y = np.cumsum(rng.randn(500, 6), 0)
# Plot the data with Matplotlib defaults
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left')
```




```
import seaborn as sns
sns.set()
# same plotting code as above!
plt.plot(x, y)
plt.legend('ABCDEF', ncol=2, loc='upper left');
```



Seaborn is that it provides high-level commands to create a variety of plot types useful for statistical data exploration.Seaborn API is much more convenient.