Introduction to Data Visualization with Python

Customizing plots

|  |
| --- |
| Graphs on common axes |
| In [1]: import matplotlib.pyplot as plt  In [2]: plt.plot(t, temperature, 'red')  In [3]: plt.plot(t, dewpoint, 'blue') # Appears on same axes  In [4]: plt.xlabel('Date')  In [5]: plt.title('Temperature & Dew Point')  In [6]: plt.show() # Renders plot objects to screen |
| Using axes() |
| In [1]: plt.axes([0.05,0.05,0.425,0.9])  In [2]: plt.plot(t, temperature, 'red')  In [3]: plt.xlabel('Date')  In [4]: plt.title('Temperature')  In [5]: plt.axes([0.525,0.05,0.425,0.9])  In [6]: plt.plot(t, dewpoint, 'blue')  In [7]: plt.xlabel('Date')  In [8]: plt.title('Dew Point')  In [9]: plt.show() |
| Using subplot() |
| In [1]: plt.subplot(2, 1, 1)  In [2]: plt.plot(t, temperature, 'red')  In [3]: plt.xlabel('Date')  In [4]: plt.title('Temperature')  In [5]: plt.subplot(2, 1, 2)  In [6]: plt.plot(t, dewpoint, 'blue')  In [7]: plt.xlabel('Date')  In [8]: plt.title('Dew Point')  In [9]: plt.tight\_layout()  In [10]: plt.show() |
| Controlling axis extents |
| ● axis([xmin, xmax, ymin, ymax]) sets axis extents  ● Control over individual axis extents  ● xlim([xmin, xmax])  ● ylim([ymin, ymax])  ● Can use tuples, lists for extents  ● e.g., xlim((-2, 3)) works  ● e.g., xlim([-2, 3]) works also |
| Other axis() options  Invocation ----> Result  axis(‘off’) ----> turns off axis lines, labels  axis(‘equal’) ----> equal scaling on x, y axes  axis(‘square’) ----> forces square plot  axis(‘tight’) ----> sets xlim(), ylim() to show all data |
| # Save the image as 'xlim\_and\_ylim.png'  plt.savefig('xlim\_and\_ylim.png') |
| Using legend() |
| In [1]: import matplotlib.pyplot as plt  In [2]: plt.scatter(setosa\_len, setosa\_wid,  ...: marker='o', color='red', label='setosa')  In [3]: plt.scatter(versicolor\_len, versicolor\_wid,  ...: marker='o', color='green', label='versicolor')  In [4]: plt.scatter(virginica\_len, virginica\_wid,  ...: marker='o', color='blue', label='virginica')  In [5]: plt.legend(loc='upper right')  …. |
| Legend locations |
| string code string code string code  'upper left' 2  'upper center' 9  'upper right' 1  'center left' 6  'center' 10  'center right' 7  'lower left' 3  'lower center' 8  'lower right' 4  'best' 0  'right' 5 |
| Using annotate() for text |
| In [1]: plt.annotate('setosa', xy=(5.0, 3.5))  In [2]: plt.annotate('virginica', xy=(7.25, 3.5))  In [3]: plt.annotate('versicolor', xy=(5.0, 2.0))  In [4]: plt.show() |
| Options for annotate()  option --🡪 description  s --🡪 text of label  xy --🡪 coordinates to annotate  xytext --🡪 coordinates of label  arrowprops --🡪 controls drawing of arrow |
| Using annotate() for arrows |
| In [1]: plt.annotate('setosa', xy=(5.0, 3.5),  ...: xytext=(4.25, 4.0), arrowprops={'color':'red'})  In [2]: plt.annotate('virginica', xy=(7.2, 3.6),  ...: xytext=(6.5, 4.0), arrowprops={'color':'blue'})  In [3]: plt.annotate('versicolor', xy=(5.05, 1.95),  ...: xytext=(5.5, 1.75),  ...: arrowprops={'color':'green'})  In [4]: plt.show() |
| Working with plot styles |
| ggplot style  In [1]: import matplotlib.pyplot as plt  In [2]: plt.style.use('ggplot') |
| fivethirtyeight style  In [1]: import matplotlib.pyplot as plt  In [2]: plt.style.use('fivethirtyeight') |
| ######### |
| # Plot with legend as before  plt.plot(year, computer\_science, color='red', label='Computer Science')  plt.plot(year, physical\_sciences, color='blue', label='Physical Sciences')  plt.legend(loc='lower right')  # Compute the maximum enrollment of women in Computer Science: cs\_max  cs\_max = computer\_science.max()  # Calculate the year in which there was maximum enrollment of women in Computer Science: yr\_max  yr\_max = year[computer\_science.argmax()]  # Add a black arrow annotation  plt.annotate('Maximum', xy=(yr\_max, cs\_max), xytext=(yr\_max+5, cs\_max+5), arrowprops=dict(facecolor='black'))  # Add axis labels and title  plt.xlabel('Year')  plt.ylabel('Enrollment (%)')  plt.title('Undergraduate enrollment of women')  plt.show() |
|  |

|  |
| --- |
| Working with 2D arrays |
| Using meshgrid() |
| import numpy as np  u = np.linspace(-2, 2, 3)  v = np.linspace(-1, 1, 5)  X,Y = np.meshgrid(u, v)  Z = X\*\*2/25 + Y\*\*2/4  Sampling on a grid  print('Z:\n', Z)  plt.set\_cmap('gray')  plt.pcolor(Z)  plt.show() |
| numpy.linspace() |
| numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)  >>> np.linspace(2.0, 3.0, num=5)  array([ 2. , 2.25, 2.5 , 2.75, 3. ])  >>> np.linspace(2.0, 3.0, num=5, endpoint=False)  array([ 2. , 2.2, 2.4, 2.6, 2.8])  >>> np.linspace(2.0, 3.0, num=5, retstep=True)  (array([ 2. , 2.25, 2.5 , 2.75, 3. ]), 0.25) |
| Orientations of 2D arrays & images |
| Z = np.array([[1, 2, 3], [4, 5, 6]])  print('Z:\n', z)  plt.pcolor(Z)  plt.show() |

|  |  |
| --- | --- |
| Visualizing bivariate functions | |
| Pseudocolor plot | In [7]: plt.pcolor(Z)  In [8]: plt.show() |
| Color bar | In [9]: plt.pcolor(Z)  In [10]: plt.colorbar()  In [11]: plt.show() |
| Color map | In [12]: plt.pcolor(Z, cmap= 'gray')  In [13]: plt.colorbar()  In [14]: plt.show() |
| In [15]: plt.pcolor(Z, cmap= 'autumn') |
| Axis tight | In [18]: plt.pcolor(Z)  In [19]: plt.colorbar()  In [20]: plt.axis('tight')  In [21]: plt.show() |
| Plot using mesh grid | In [22]: plt.pcolor(X, Y, Z) # X, Y are 2D meshgrid  In [23]: plt.colorbar()  In [24]: plt.show() |
| Contour plots | In [25]: plt.contour(Z)  In [26]: plt.show() |
| More contours | In [27]: plt.contour(Z, 30) |
| Contour plot using meshgrid | In [29]: plt.contour(X, Y, Z, 30) |
| Filled contour plots | In [31]: plt.contourf(X, Y, Z, 30)  In [32]: plt.colorbar() |

|  |  |
| --- | --- |
| Visualizing bivariate distributions | |
| Histograms in 1D | In [1]: counts, bins, patches = plt.hist(x, bins=25)  In [2]: plt.show() |
| hist2d(): Rectangular binning | In [1]: plt.hist2d(x, y, bins=(10, 20) range=((30, 55), (8, 48)) # x & y are 1D arrays of same length  In [2]: plt.colorbar()  In [5]: plt.show() |
| hexbin(): Hexagonal binning | In [1]: plt.hexbin(x, y, gridsize=(15,10), extent=(xmin, xmax, ymin, ymax)) |

|  |  |
| --- | --- |
| Working with images | |
| Images | ● Grayscale images: rectangular 2D arrays  ● Color images: typically three 2D arrays (channels)  ● RGB (Red-Green-Blue)  ● Channel values:  ● 0 to 1 (floating-point numbers)  ● 0 to 255 (8 bit integers) |
| Loading images | In [1]: img = plt.imread('sunflower.jpg')  In [2]: print(img.shape)  (480, 640, 3)  In [3]: plt.imshow(img)  In [4]: plt.axis('off')  In [5]: plt.show() |
| Reduction to gray-scale image | In [6]: collapsed = img.mean(axis=2)  In [7]: print(collapsed.shape)  (480, 640)  In [8]: plt.set\_cmap('gray')  In [9]: plt.imshow(collapsed, cmap='gray')  In [10]: plt.axis('off')  In [11]: plt.show() |
| Uneven samples | In [12]: uneven = collapsed[::4,::2] # nonuniform subsampling  In [13]: print(uneven.shape)  (120, 320)  In [14]: plt.imshow(uneven)  In [15]: plt.axis('off')  In [16]: plt.show() |
| Adjusting aspect ratio | In [17]: plt.imshow(uneven, aspect=2.0)  In [18]: plt.axis('off')  In [19]: plt.show() |
| Adjusting extent | In [20]: plt.imshow(uneven, cmap='gray', extent=(0,640,0,480))  In [21]: plt.axis('off')  In [22]: plt.show() |

|  |
| --- |
| ################## |
| # Load the image into an array: img  img = plt.imread('480px-Astronaut-EVA.jpg')  # Specify the extent and aspect ratio of the top left subplot  plt.subplot(2,2,1)  plt.title('extent=(-1,1,-1,1),\naspect=0.5')  plt.xticks([-1,0,1])  plt.yticks([-1,0,1])  plt.imshow(img, extent=(-1,1,-1,1), aspect=0.5)  # Specify the extent and aspect ratio of the top right subplot  plt.subplot(2,2,2)  plt.title('extent=(-1,1,-1,1),\naspect=1')  plt.xticks([-1,0,1])  plt.yticks([-1,0,1])  plt.imshow(img, extent=(-1,1,-1,1), aspect=1)  # Specify the extent and aspect ratio of the bottom left subplot  plt.subplot(2,2,3)  plt.title('extent=(-1,1,-1,1),\naspect=2')  plt.xticks([-1,0,1])  plt.yticks([-1,0,1])  plt.imshow(img, extent=(-1,1,-1,1), aspect=2)  # Specify the extent and aspect ratio of the bottom right subplot  plt.subplot(2,2,4)  plt.title('extent=(-2,2,-1,1),\naspect=2')  plt.xticks([-2,-1,0,1,2])  plt.yticks([-1,0,1])  plt.imshow(img, extent=(-2,2,-1,1), aspect=2)  # Improve spacing and display the figure  plt.tight\_layout()  plt.show() |
|  |
| # Load the image into an array: image  image = plt.imread('640px-Unequalized\_Hawkes\_Bay\_NZ.jpg')  # Extract minimum and maximum values from the image: pmin, pmax  pmin, pmax = image.min(), image.max()  print("The smallest & largest pixel intensities are %d & %d." % (pmin, pmax))  # Rescale the pixels: rescaled\_image  rescaled\_image = 256\*(image - pmin) / (pmax - pmin)  print("The rescaled smallest & largest pixel intensities are %.1f & %.1f." %  (rescaled\_image.min(), rescaled\_image.max()))  # Display the original image in the top subplot  plt.subplot(2,1,1)  plt.title('original image')  plt.axis('off')  plt.imshow(image)  # Display the rescaled image in the bottom subplot  plt.subplot(2,1,2)  plt.title('rescaled image')  plt.axis('off')  plt.imshow(rescaled\_image)  plt.show() |
|  |