Topic for the class-Customizing plot legends

Unit _2 : Title-Digital data – an Imprint

Date & Time : 19.8.24 11.00 AM – 11.50 AM

Dr. Bhramaramba Ravi

Professor

Department of Computer Science and Engineering

GITAM School of Technology (GST)

Visakhapatnam – 530045

Email: **bravi@gitam.edu**

Unit2-syllabus

- **UNIT 2 Digital Data-An Imprint 9 hours, P - 2 hours** Type of data analytics (Descriptive, diagnostic, perspective, predictive, Prescriptive.) Exploratory Data Analysis (EDA), EDA-Quantitative Technique, EDA - Graphical Technique. Data Types for Plotting, Data Types and Plotting, Simple Line Plots, Simple Scatter Plots, Visualizing Errors, Density and Contour Plots, Histograms, Binnings, and Density, Customizing Plot Legends, Customizing Color bars, Multiple Subplots, Text and Annotation, Customizing Ticks.
- https://www.coursera.org/learn/data-visualization-r

Customizing plot legends

- Plot legends give meaning to a visualization, assigning labels to the various plot elements.
- We previously saw how to create a simple legend; here we'll take a look at customizing the placement and aesthetics of the legend in Matplotlib.
- The simplest legend can be created with the plt.legend() command, which automatically creates a legend for any labeled plot elements (Figure 4-41):
- In[1]: **import matplotlib.pyplot as plt**
- plt.style.use('classic')
- In[2]: %matplotlib inline
- **import numpy as np**
- In[3]: x = np.linspace(0, 10, 1000)
- fig, ax = plt.subplots()
- ax.plot(x, np.sin(x), '-b', label='Sine')
- ax.plot(x, np.cos(x), '--r', label='Cosine')
- ax.axis('equal')
- leg = ax.legend();

Customizing plot legends

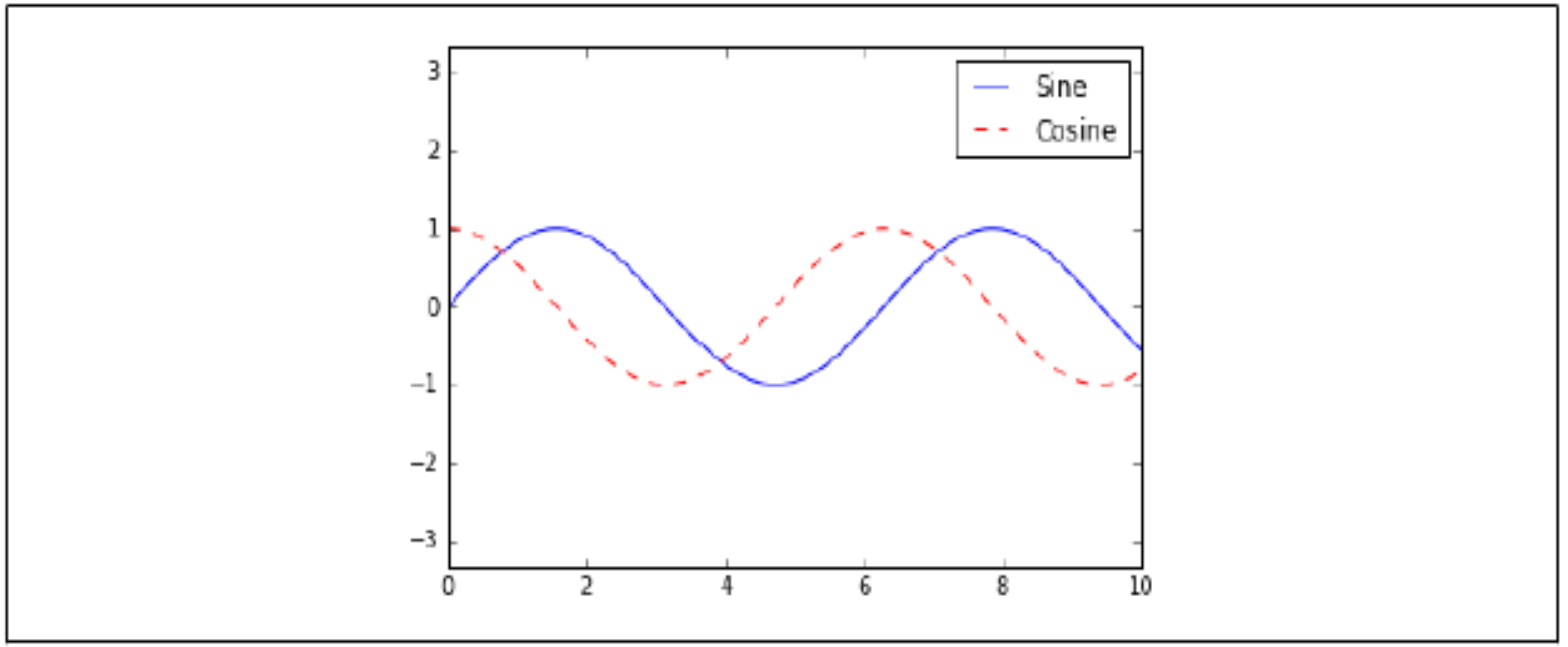

Figure 4-41. A default plot legend

Customizing plot legends

- But there are many ways we might want to customize such a legend. For example, we can specify the location and turn off the frame (Figure 4-42):
- In[4]: ax.legend(loc='upper left', frameon=False)

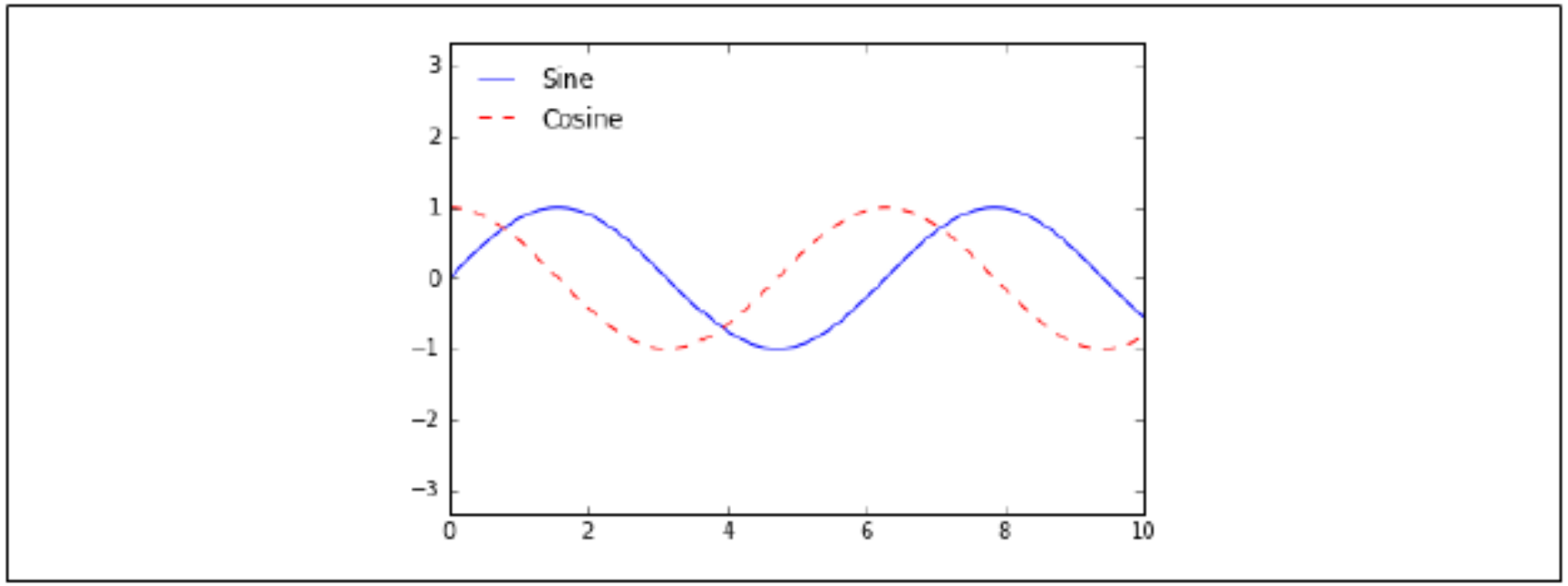

Figure 4-42. A customized plot legend

Customizing plot legends

- We can use the ncol command to specify the number of columns in the legend
- (Figure 4-43):
- In[5]: ax.legend(frameon=False, loc='lower center', ncol=2)

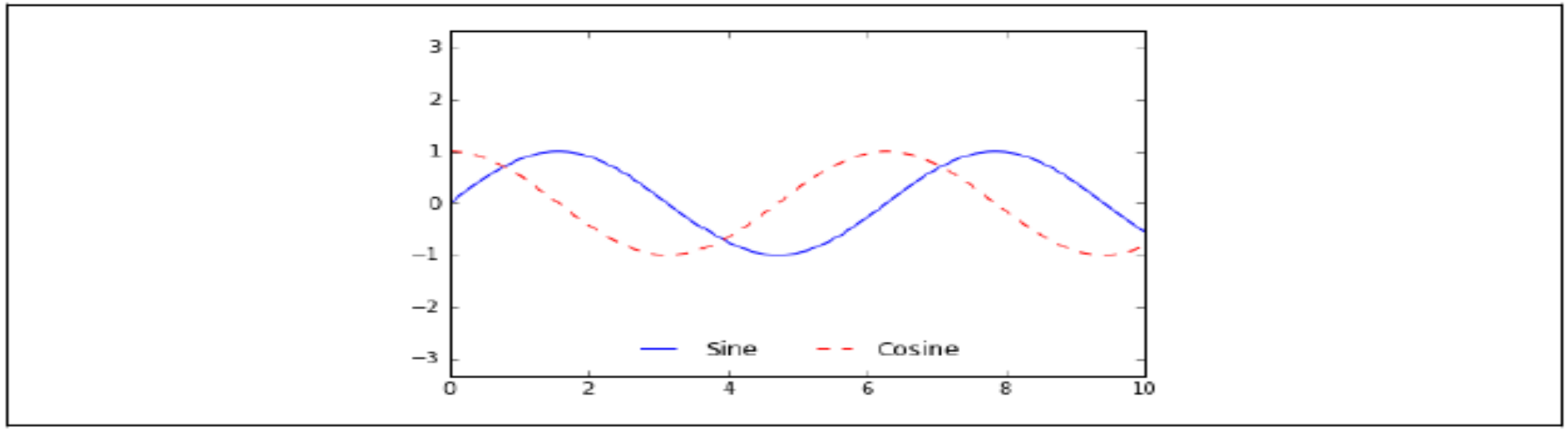

Figure 4-43. A two-column plot legend

Customizing plot legends

- We can use a rounded box (fancybox) or add a shadow, change the transparency (alpha value) of the frame, or change the padding around the text (Figure 4-44):
- In[6]: ax.legend(fancybox=True, framealpha=1, shadow=True, borderpad=1)

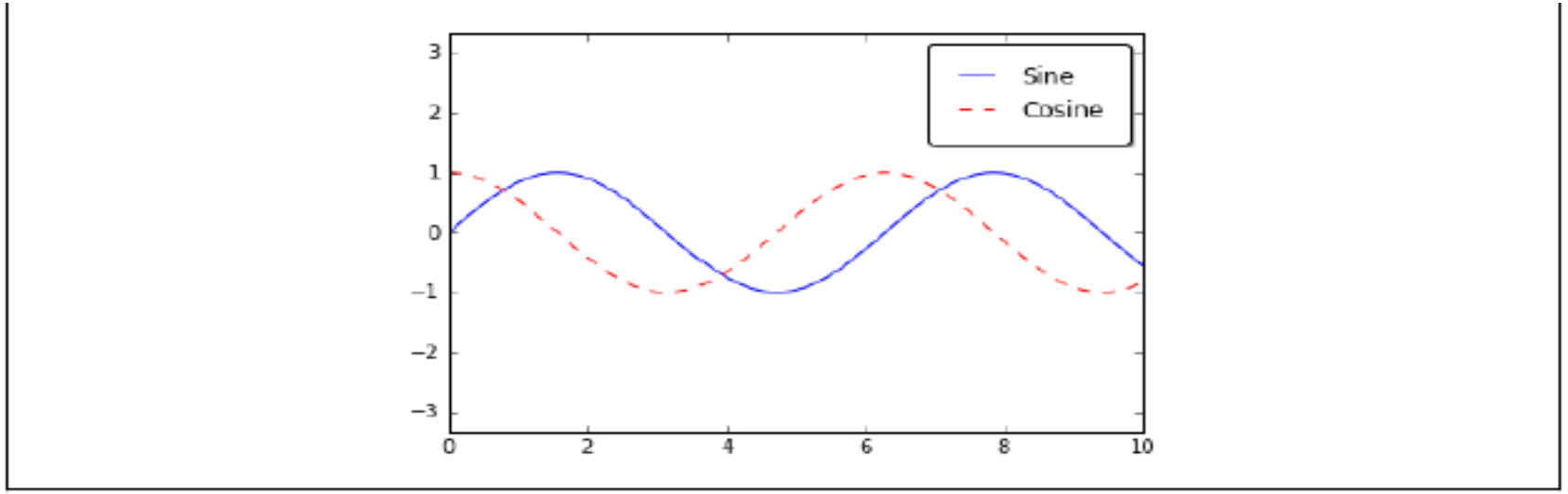

Figure 4-44. A fancybox plot legend

Customizing plot legends

- Choosing elements for the legend

- As we've already seen, the legend includes all labeled elements by default. If this is not what is desired, we can fine-tune which elements and labels appear in the legend by using the objects returned by plot commands.
- The plt.plot() command is able to create multiple lines at once, and returns a list of created line instances.
- Passing any of these to plt.legend() will tell it which to identify, along with the labels we'd like to
- specify (Figure 4-45):

```
In[7]: y = np.sin(x[:, np.newaxis] + np.pi * np.arange(0, 2, 0.5))
lines = plt.plot(x, y)
# lines is a list of plt.Line2D instances
plt.legend(lines[:2], ['first', 'second']);
```

Customizing plot legends

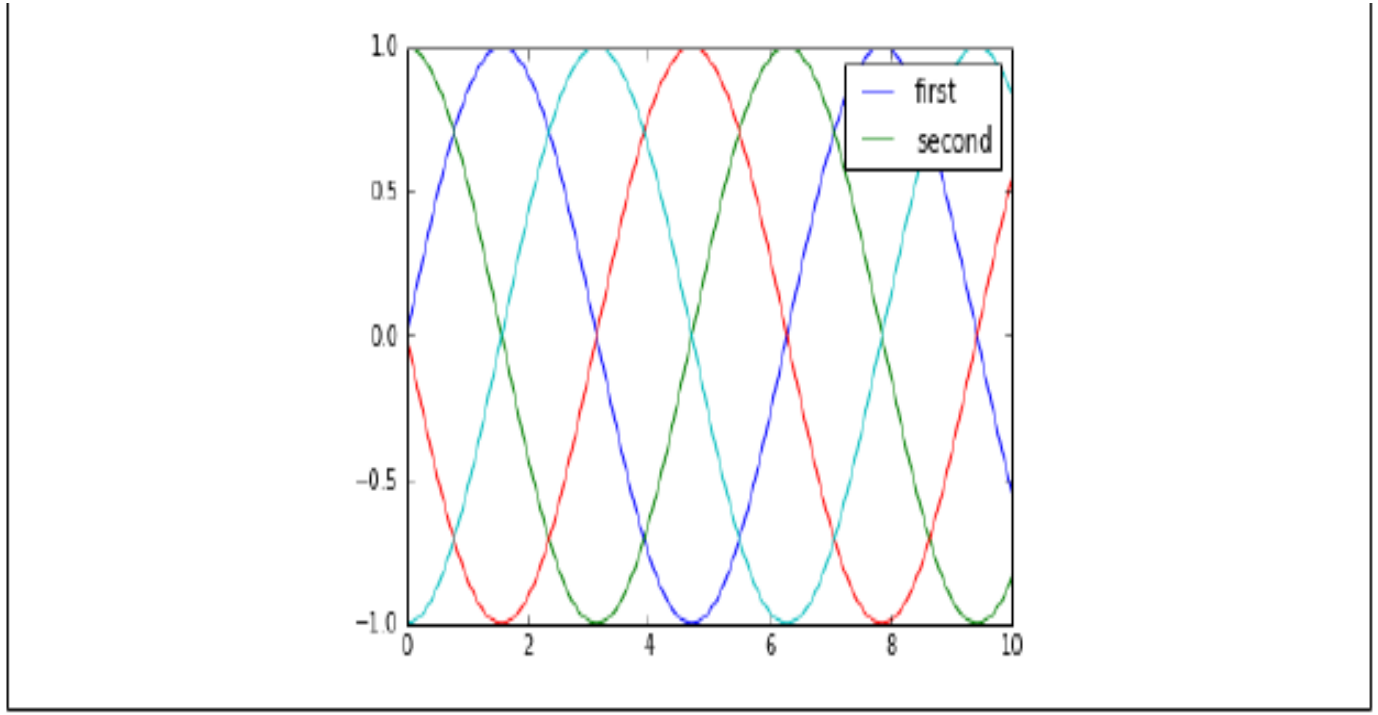

Figure 4-45. Customization of legend elements

Customizing plot legends

- it is clearer to use the first method, applying labels to the plot elements you'd like to show on the legend (Figure 4-46):
- In[8]: plt.plot(x, y[:, 0], label='first')
- plt.plot(x, y[:, 1], label='second')
- plt.plot(x, y[:, 2:])
- plt.legend(framealpha=1, frameon=True);
- By default, the legend ignores all elements without a label attribute set.

Customizing plot legends

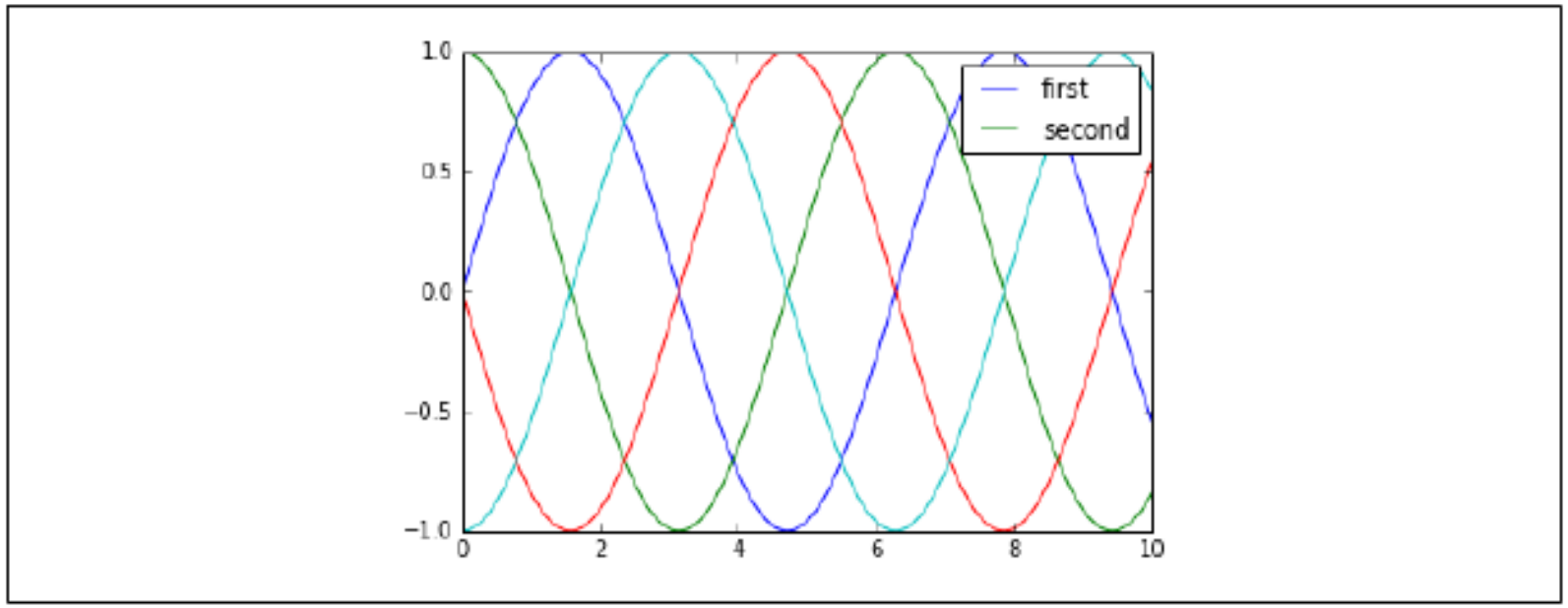

Figure 4-46. Alternative method of customizing legend elements

Customizing plot legends

- **Legend for Size of Points**
- Sometimes the legend defaults are not sufficient for the given visualization.
- For example, perhaps you're using the size of points to mark certain features of the data, and want to create a legend reflecting this.
- Here is an example where we'll use the size of points to indicate populations of California cities.
- We'd like a legend that specifies the scale of the sizes of the points, and we'll accomplish this by plotting some labeled data
- with no entries (Figure 4-47):
- In[9]: **import pandas as pd**
- cities = pd.read_csv('data/california_cities.csv')
- *# Extract the data we're interested in*
- lat, lon = cities['latd'], cities['longd']
- population, area = cities['population_total'], cities['area_total_km2']
- *# Scatter the points, using size and color but no label*
- plt.scatter(lon, lat, label=None,
- c=np.log10(population), cmap='viridis',
- s=area, linewidth=0, alpha=0.5)

Customizing plot legends

```
plt.axis(aspect='equal')
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.colorbar(label='log$_{10}$(population)')
plt.clim(3, 7)
# Here we create a legend:
# we'll plot empty lists with the desired size and label
for area in [100, 300, 500]:
plt.scatter([], [], c='k', alpha=0.3, s=area,
label=str(area) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False,
labelspacing=1, title='City Area')
plt.title('California Cities: Area and Population');
```

Customizing plot legends

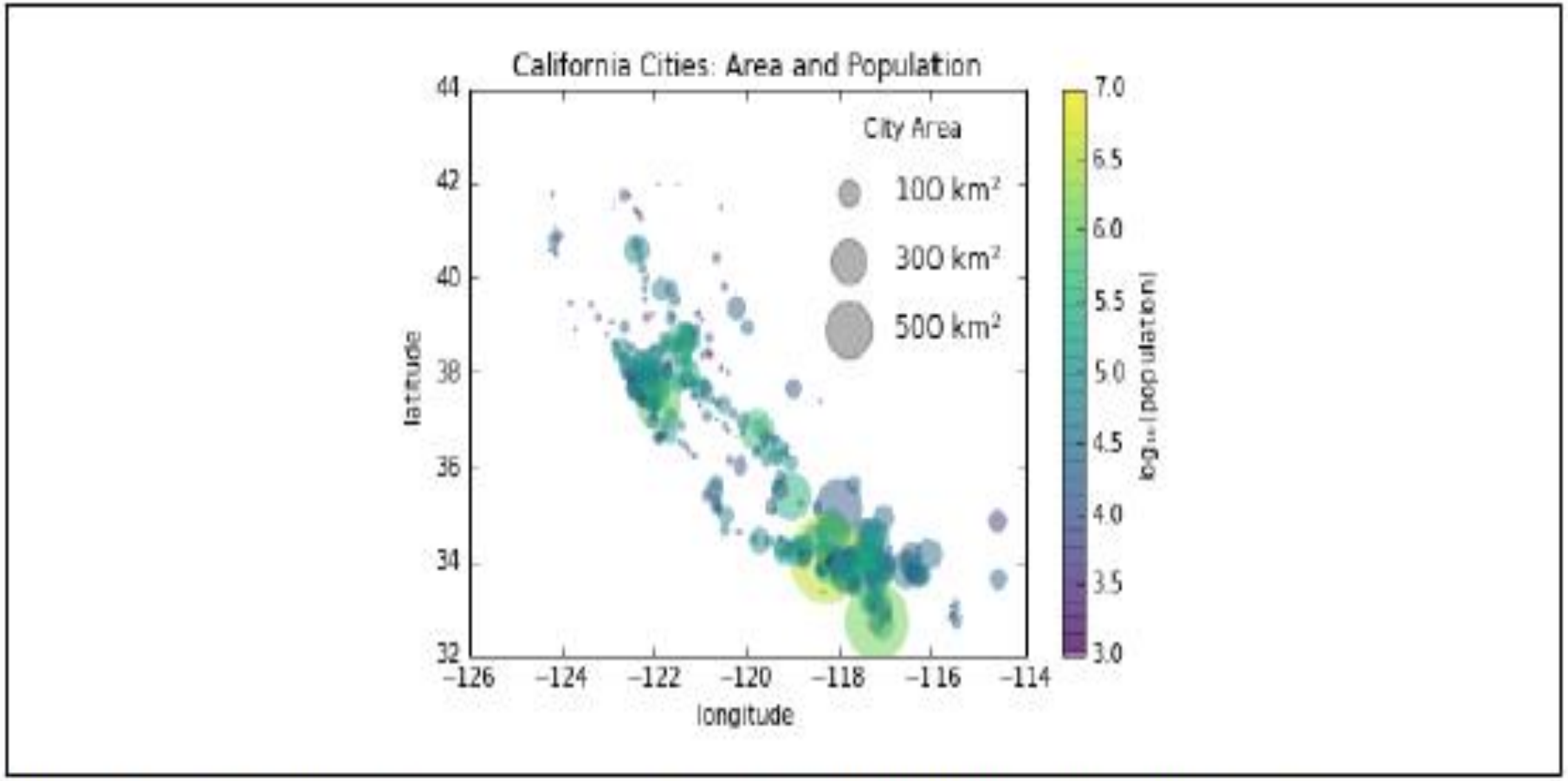

Figure 4-47. Location, geographic size, and population of California cities

Customizing plot legends

- The legend will always reference some object that is on the plot, so if we'd like to display a particular shape we need to plot it.
- In this case, the objects we want (gray circles) are not on the plot, so we fake them by plotting empty lists.
- Notice too that the legend only lists plot elements that have a label specified.
- By plotting empty lists, we create labeled plot objects that are picked up by the legend, and now our legend tells us some useful information.
- This strategy can be useful for creating more sophisticated visualizations.
- Finally, note that for geographic data like this, it would be clearer if we could show state boundaries or other map-specific elements.
- For this, an excellent choice of tool is Matplotlib's Basemap add-on toolkit.

Customizing plot legends

- **Multiple Legends**
- Sometimes when designing a plot you'd like to add multiple legends to the same axes.
- Unfortunately, Matplotlib does not make this easy: via the standard legend interface,
- it is only possible to create a single legend for the entire plot. If you try to create a second legend using plt.legend() or ax.legend(),
- it will simply override the first one.
- We can work around this by creating a new legend artist from scratch, and then using the lower-level ax.add_artist() method to manually add the second artist to the plot (Figure 4-48):

```
In[10]: fig, ax = plt.subplots()
lines = []
styles = ['-', '--', '-.', ':']
x = np.linspace(0, 10, 1000)
for i in range(4):
lines += ax.plot(x, np.sin(x - i * np.pi / 2),
styles[i], color='black')
ax.axis('equal')
```

Customizing plot legends

```
# specify the lines and labels of the first legend
ax.legend(lines[:2], ['line A', 'line B'],
loc='upper right', frameon=False)
# Create the second legend and add the artist manually.
from matplotlib.legend import Legend
leg = Legend(ax, lines[2:], ['line C', 'line D'],
loc='lower right', frameon=False)
ax.add_artist(leg);
```

Customizing plot legends

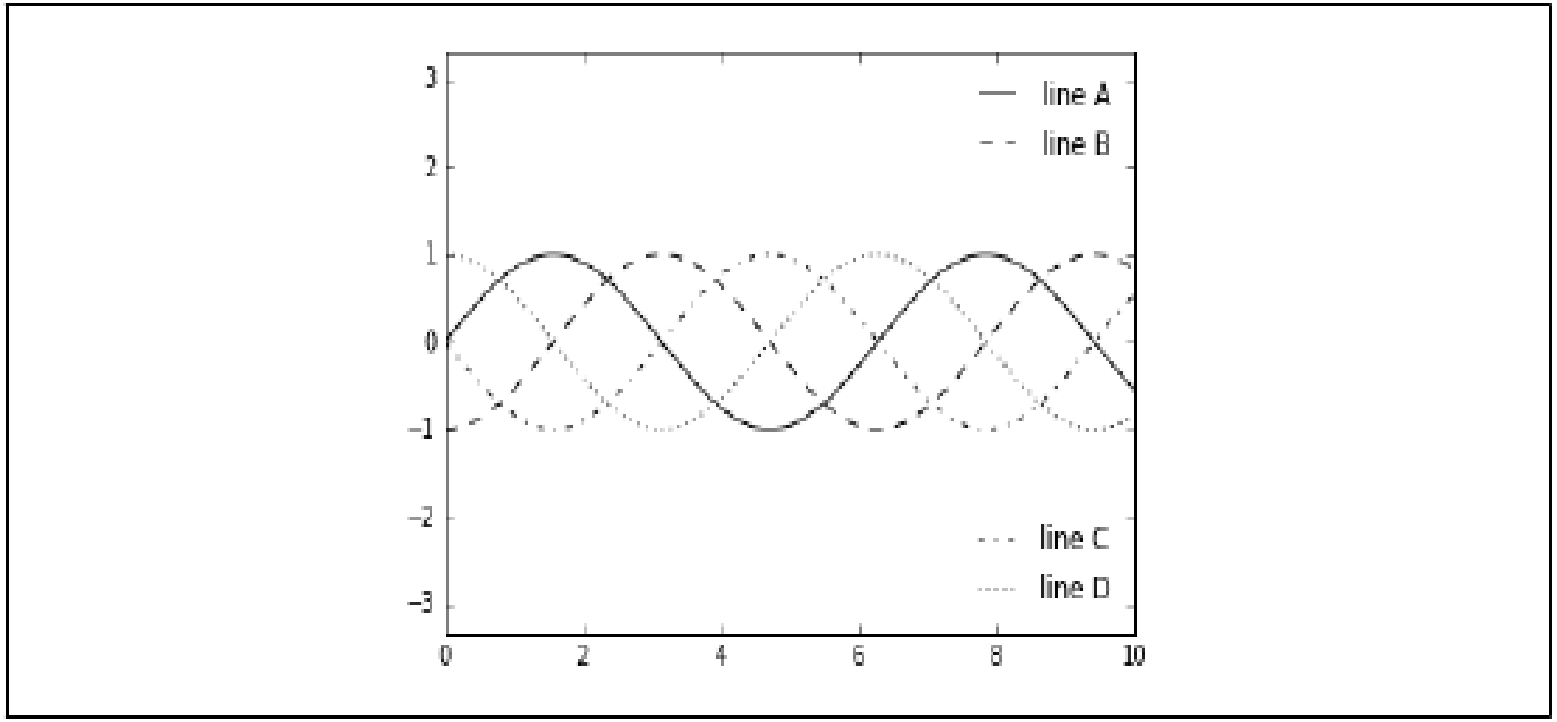

Figure 4-48. A split plot legend

- This is a peek into the low-level artist objects that compose any Matplotlib plot.
- If you examine the source code of ax.legend() (recall that you can do this within the IPython
- notebook using ax.legend??) you'll see that the function simply consists of which is then saved in the legend_attribute and added to the figure when the plot is drawn.

THANK YOU