Topic for the class:– Scatter Plots
Unit _2 : Title-Digital data – an Imprint
Date & Time : 12.8.24 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

Simple scatter plots

- Another commonly used plot type is the simple scatter plot, a close cousin of the line plot.
- Instead of points being joined by line segments, here the points are represented individually with a dot, circle, or other shape.
- We'll start by setting up the notebook for plotting and importing the functions we will use:
- In[1]: %matplotlib inline
- **import matplotlib.pyplot as plt**
- plt.style.use('seaborn-whitegrid')
- **import numpy as np**

Scatter plots with plt.plot

- Earlier, we looked at plt.plot/ax.plot to produce line plots.
- It turns out that this same function can produce scatter plots as well (Figure 4-20):
- In[2]: x = np.linspace(0, 10, 30)
- y = np.sin(x)
- plt.plot(x, y, 'o', color='black');

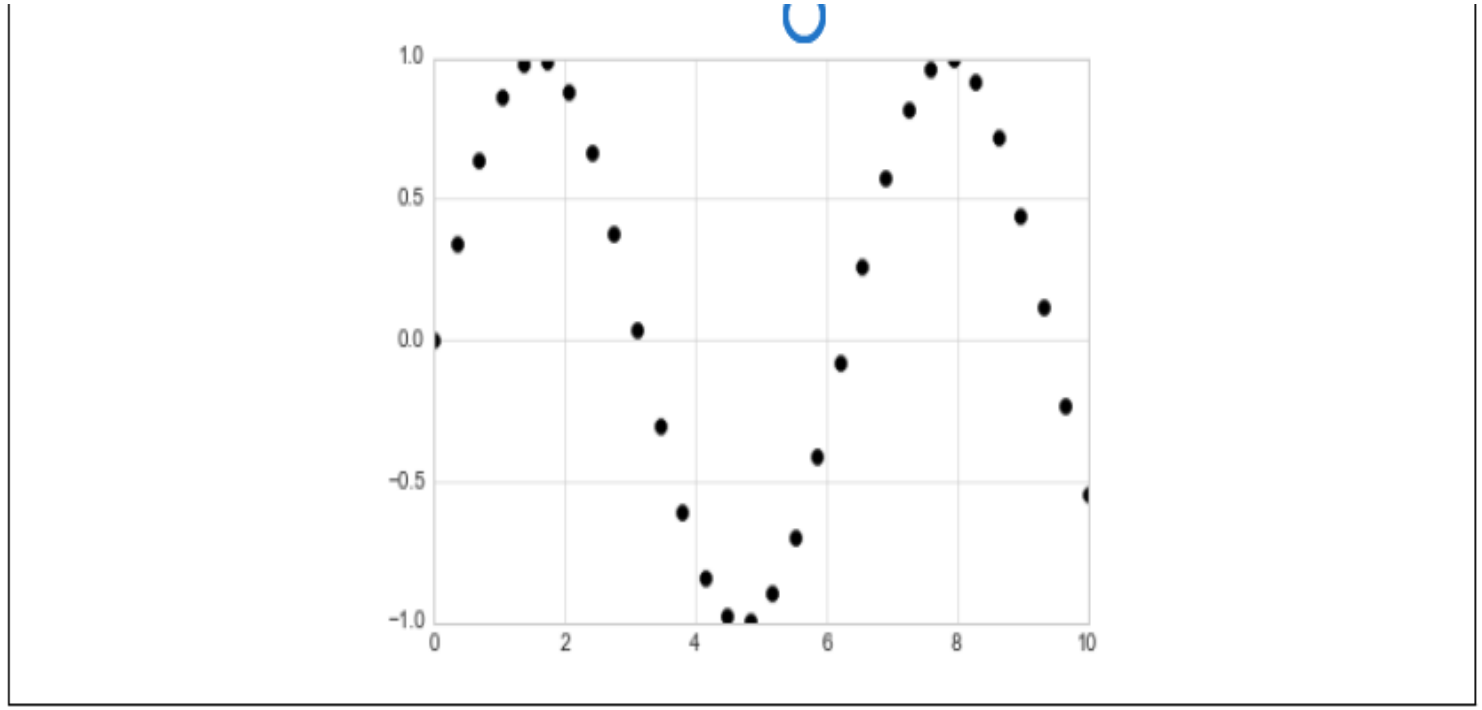

Figure 4-20. Scatter plot example

Scatter plots with plt.plot

- The third argument in the function call is a character that represents the type of symbol used for the plotting.
- Just as you can specify options such as '-' and '--' to control the line style, the marker style has its own set of short string codes.
- The full list of available symbols can be seen in the documentation of plt.plot, or in Matplotlib's
- online documentation.
- Most of the possibilities are fairly intuitive, and you can see a number of the more common ones here (Figure 4-21):
- In[3]: rng = np.random.RandomState(0)
- for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
- plt.plot(rng.rand(5), rng.rand(5), marker,
- label="marker='{0}'".format(marker))
- plt.legend(numpoints=1)
- plt.xlim(0, 1.8);

Scatter plots with plt.plot contd.

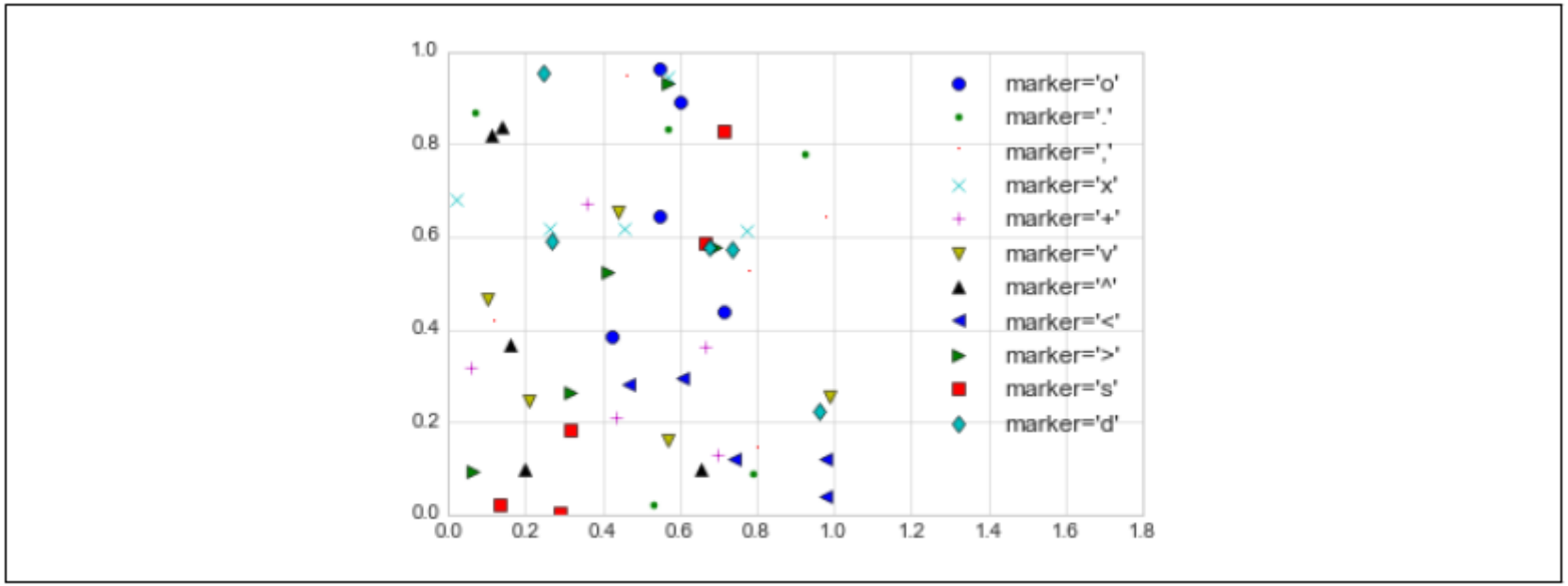

Figure 4-21. Demonstration of point numbers

Scatter plots with plt.plot contd.

- For even more possibilities, these character codes can be used together with line and color codes to plot points along with a line connecting them (Figure 4-22):
- In[4]: plt.plot(x, y, '-ok'); *# line (-), circle marker (o), black (k)*

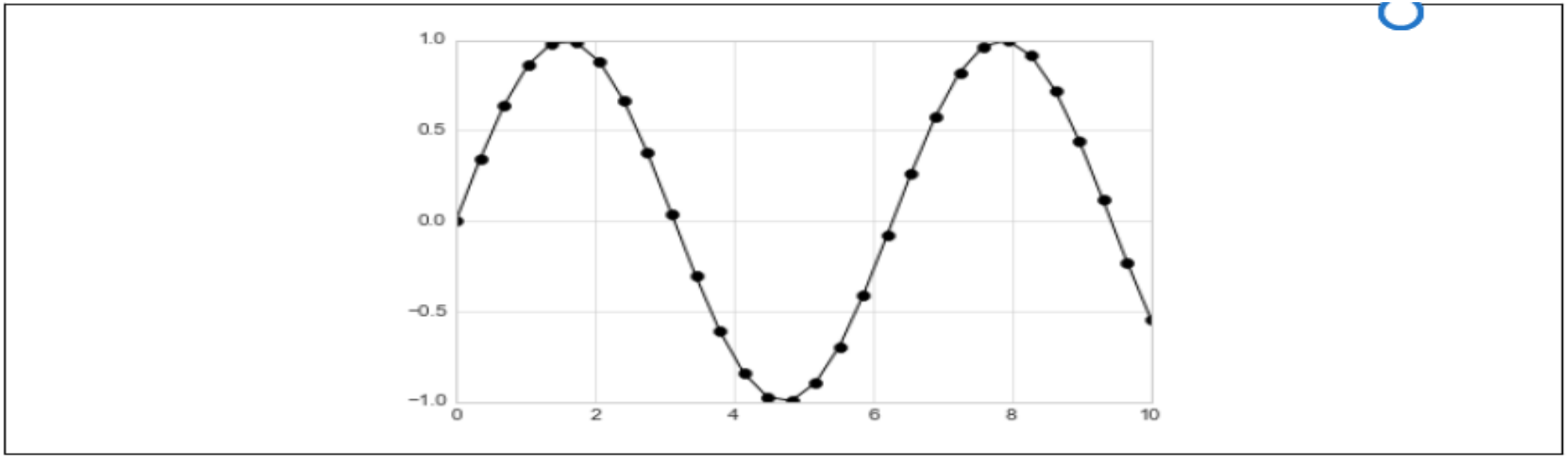

Figure 4-22. Combining line and point markers

Scatter plots with plt.plot

- Additional keyword arguments to plt.plot specify a wide range of properties of the lines and markers (Figure 4-23):
- In[5]: plt.plot(x, y, '-p', color='gray',
- markersize=15, linewidth=4,
- markerfacecolor='white',
- markeredgecolor='gray',
- markeredgewidth=2)
- plt.ylim(-1.2, 1.2);
- This type of flexibility in the

plt.plot function allows for a wide variety of possible visualization options.

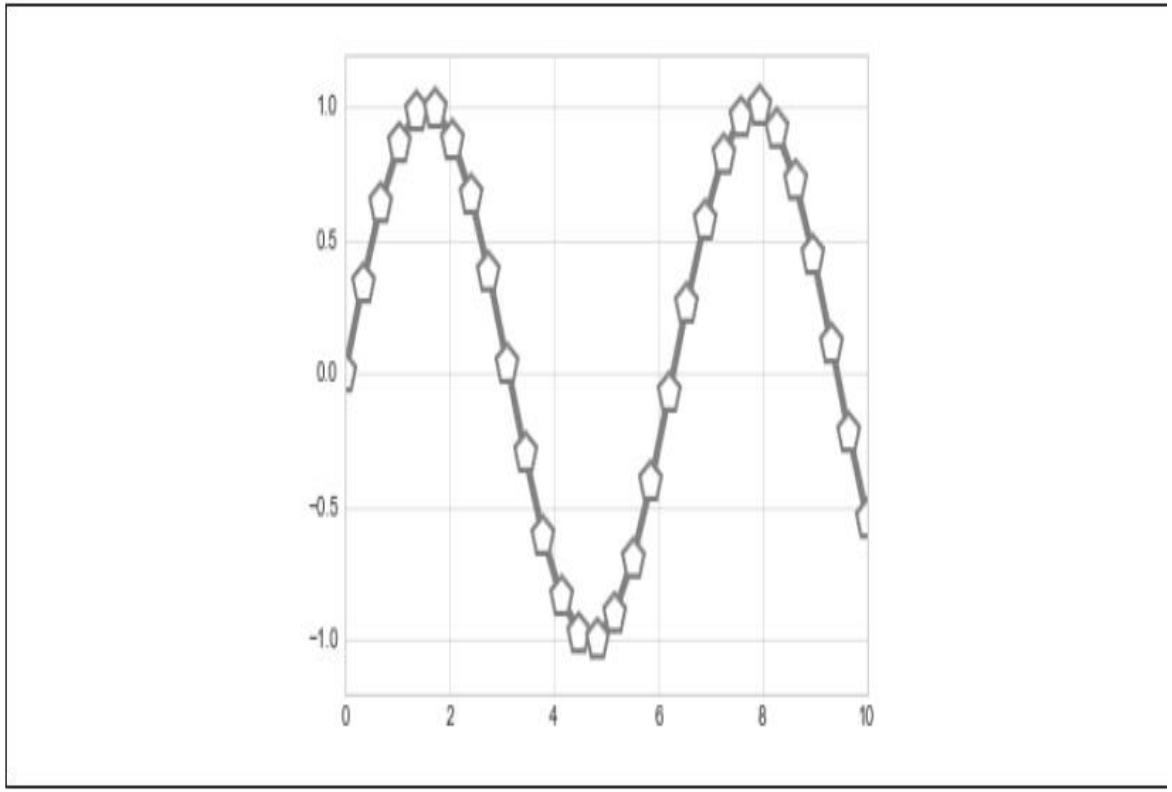

Figure 4-23. Customizing line and point numbers

Scatter plots with plt.scatter

- A second, more powerful method of creating scatter plots is the plt.scatter function, which can be used very similarly to the plt.plot function (Figure 4-24):
- In[6]: plt.scatter(x, y, marker='o');

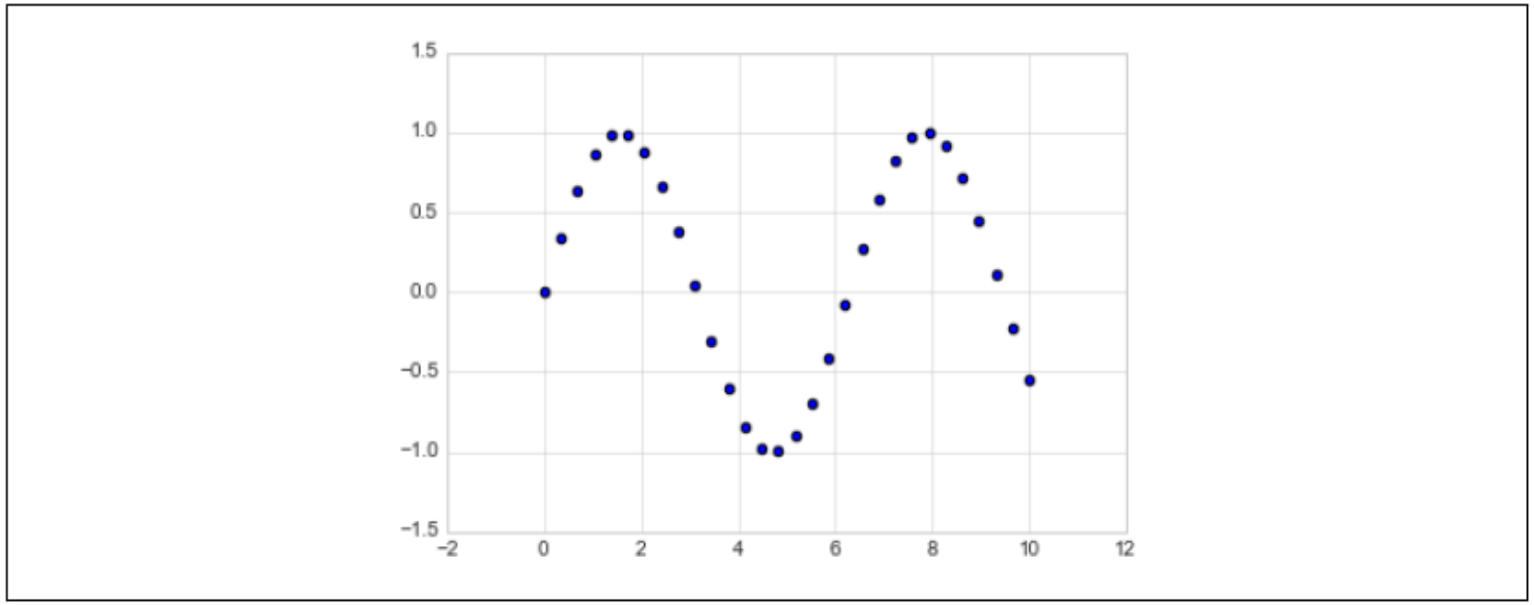

Figure 4-24. A simple scatter plot

Scatter plots with plt.scatter

- The primary difference of plt.scatter from plt.plot is that it can be used to create scatter plots where the properties of each individual point (size, face color, edge color, etc.) can be individually controlled or mapped to data.
- Let's show this by creating a random scatter plot with points of many colors and sizes.
- In order to better see the overlapping results, we'll also use the alpha keyword to adjust the transparency level (Figure 4-25):
- In[7]: rng = np.random.RandomState(0)
- x = rng.randn(100)
- y = rng.randn(100)
- colors = rng.rand(100)
- sizes = 1000 * rng.rand(100)
- plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
- cmap='viridis')
- plt.colorbar(); *# show color scale*

Scatter plots with plt.scatter

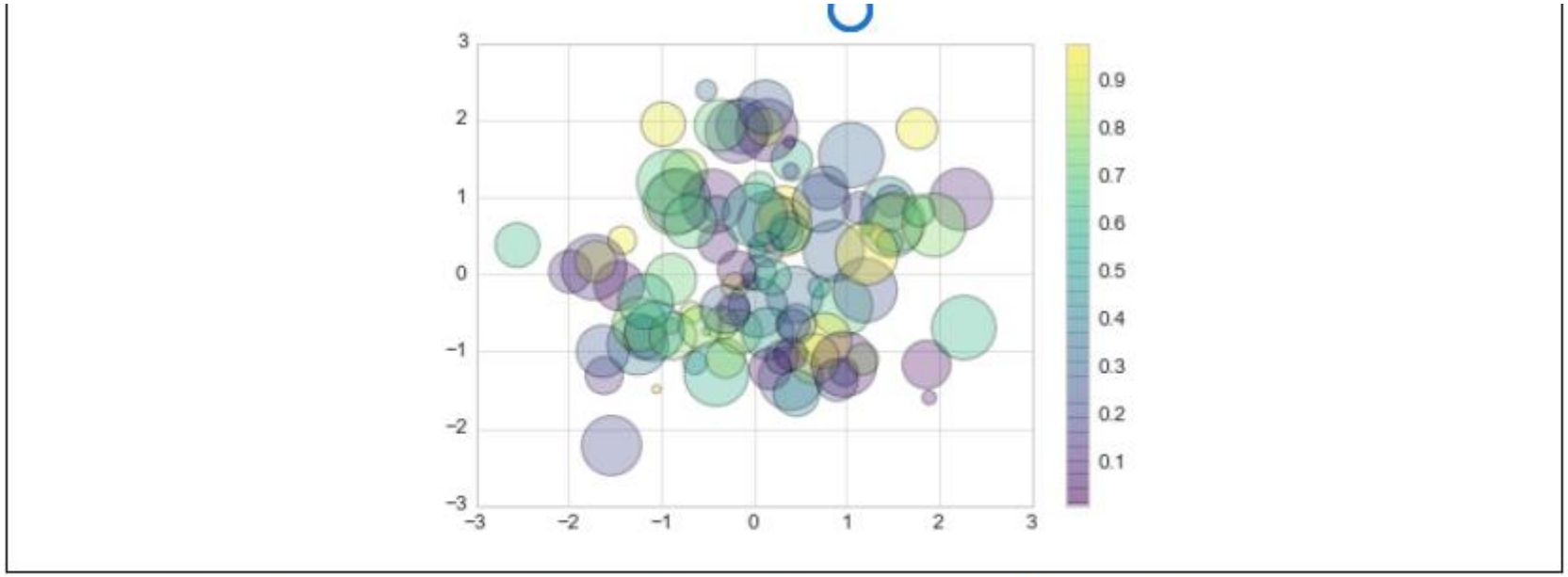

Figure 4-25. Changing size, color, and transparency in scatter points

Scatter plots with plt.scatter

- the color argument is automatically mapped to a color scale (shown here
- by the colorbar() command), and the size argument is given in pixels. In this way,
- the color and size of points can be used to convey information in the visualization, in
- order to illustrate multidimensional data.
- For example, we might use the Iris data from Scikit-Learn, where each sample is one
- of three types of flowers that has had the size of its petals and sepals carefully measured
- (Figure 4-26):
- In[8]: **from** sklearn.datasets **import** load_iris
- iris = load_iris()
- features = iris.data.T
- plt.scatter(features[0], features[1], alpha=0.2,
- s=100*features[3], c=iris.target, cmap='viridis')
- plt.xlabel(iris.feature_names[0])
- plt.ylabel(iris.feature_names[1]);

Scatter plots with plt.scatter

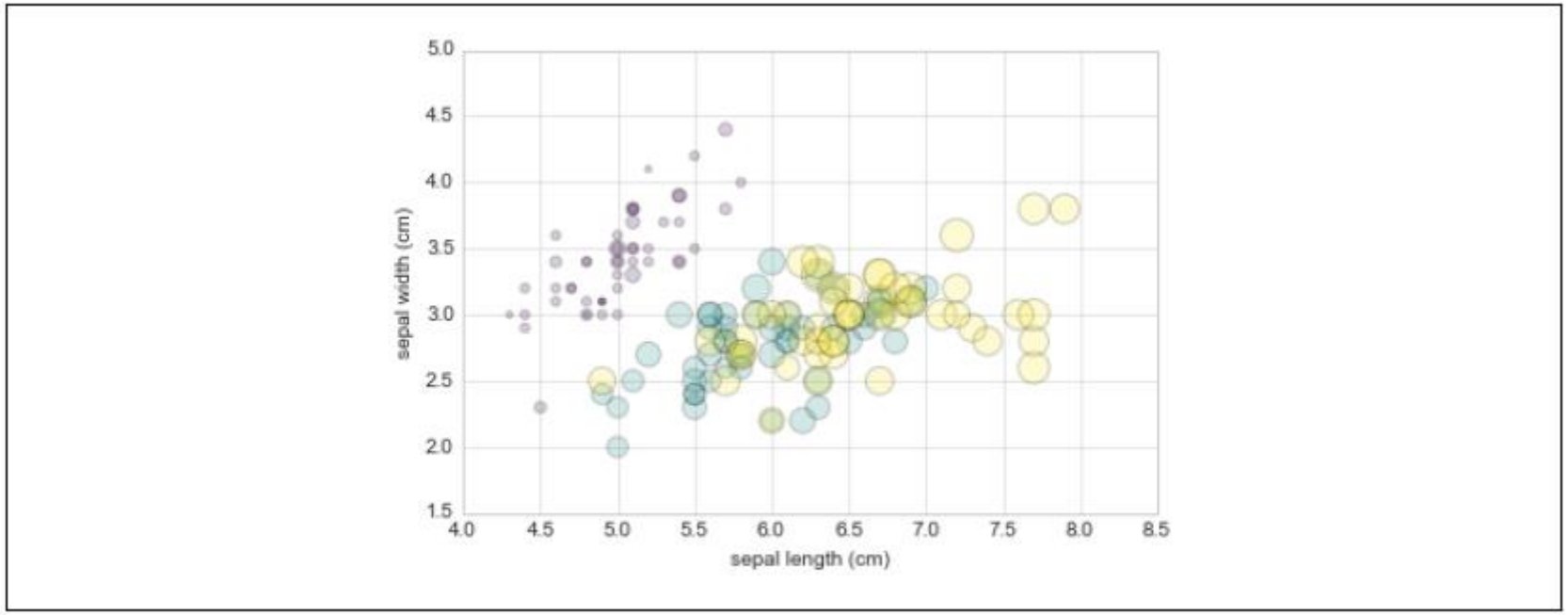

Figure 4-26. Using point properties to encode features of the Iris data

Scatter plots with plt.scatter

- We can see that this scatter plot has given us the ability to simultaneously explore four different dimensions of the data: the (x, y) location of each point corresponds to the sepal length and width, the size of the point is related to the petal width, and the color is related to the particular species of flower.
- Multicolor and multifeature scatter plots like this can be useful for both exploration and presentation of data.

Plot versus Scatter: A note on efficiency

- Aside from the different features available in plt.plot and plt.scatter, why might you choose to use one over the other?
- While it doesn't matter as much for small amounts of data, as datasets get larger than a few thousand points, plt.plot can be noticeably more efficient than plt.scatter.
- The reason is that plt.scatter has the capability to render a different size and/or color for each point, so the renderer must do the extra work of constructing each point individually.
- In plt.plot, on the other hand, the points are always essentially clones of each other, so the work of determining the appearance of the points is done only once for the entire set of data.
- For large datasets, the difference between these two can lead to vastly different performance, and for this reason, plt.plot should be preferred over plt.scatter for large datasets.

THANK YOU