PennState World Campus

# DAAN862: Analytics Programming in Python

***Lesson 6: Data Visualization with Matplotlib***

## Lesson 6: Data Visualization with Matplotlib

Making informative visualizations is one of the most important tasks in data analysis. It may be a part of th exploratory process. For example, to help identify outliers or needed data transformations or as a way of generating ideas for Python has many add-on libraries for making static or dynamic visualizations. However, we will mainly focus on matplotlib and pandas plot methods that build on top of matplotlip.

At the end of this lesson, you will learn how to:

Plot basic figures with matplotlib

Plot basic figures with pandas

Unfortunately, there is not enough time in this course to give a comprehensive treatment to the breadth and depth of functionality in matplotlib. It should be enough to teach you the ropes to get up and running. The matplotlib gallery and documentation (https://matplotlib.org/index.html) are the best resources for learning advanced features.

By the end of this lesson, please complete all readings and assignments found in the Lesson 6 Course Schedule.

## Lesson 6.1: Plotting with matplotlib

***matplotlib*** is a plotting package designed for creating (mostly two-dimensional) plots. The project was started by John Hunter in 2002 to enable a MATLAB-like plotting interface in Python. *matplotlib* supports various common graphic formats (PDF, JPG, PNG, BMP, GIF, etc.)

### Setup plot preference

In this lesson, we will use an interactive plot environment as shown in Figures 6.1 and 6.2:

In Spyder, click tools, the select preferences (shown in Figure 6.1)

Fig
6.1

A window will pop up, please change your setting as the same with Figure 6.2

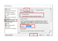
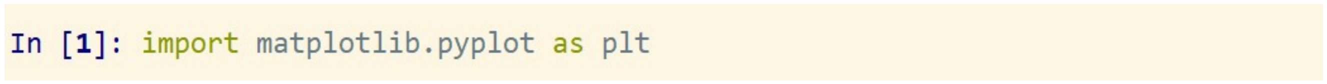


Fig
6.2

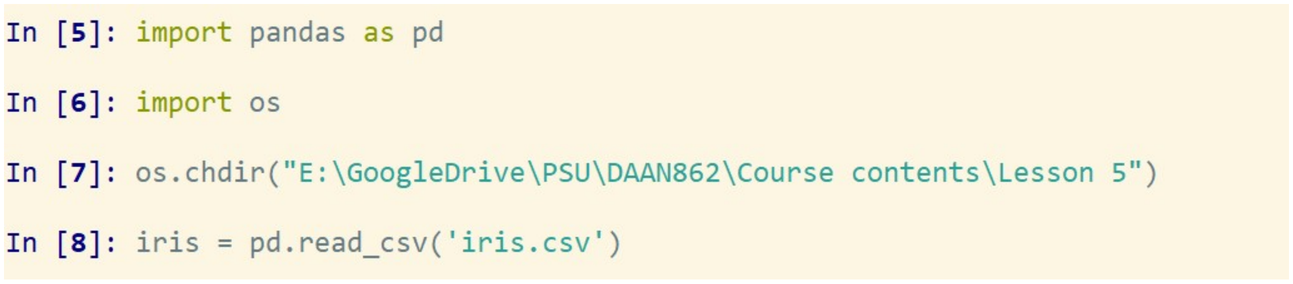
With *matplotlib*, the following import convection is used (shown in Figure 6.3):



```
In [1]: import matplotlib.pyplot as plt
```

Fig 6.3 (click to enlarge)

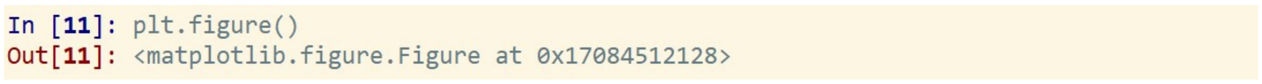
Iris data was used to explore different varieties of plots (shown in Figure 6.4):



```
In [5]: import pandas as pd

In [6]: import os

In [7]: os.chdir("E:\GoogleDrive\PSU\DAAN862\Course contents\Lesson 5")

In [8]: iris = pd.read_csv('iris.csv')
```

Fig 6.4 (click to enlarge)

For all Metaplotlib plots, we start by creating an empty figure (shown in Figure 6.5):



```
In [11]: plt.figure()
Out[11]: <matplotlib.figure.Figure at 0x17084512128>
```

Fig 6.5 (click to enlarge)

A figure will be shown in the separate window as shown in Figure 6.6. You will be able to zoom in, zoom out, move and save the figures by using the tabs on the window. Everytime you change the plot settings, the window in Figure 6.6 will update accordingly:

Fig 6.6

---

*(click the titles to learn more)*

## Line Plots

You can create a simple line plot with *plt.plot* which accepts array-like data x or (x,y). Figure 6.7 shows the line plot for *iris.sepal length*:

Figure 6.7

After creating the *plt.plot* a blue line will show up as shown in Figure 6.8:

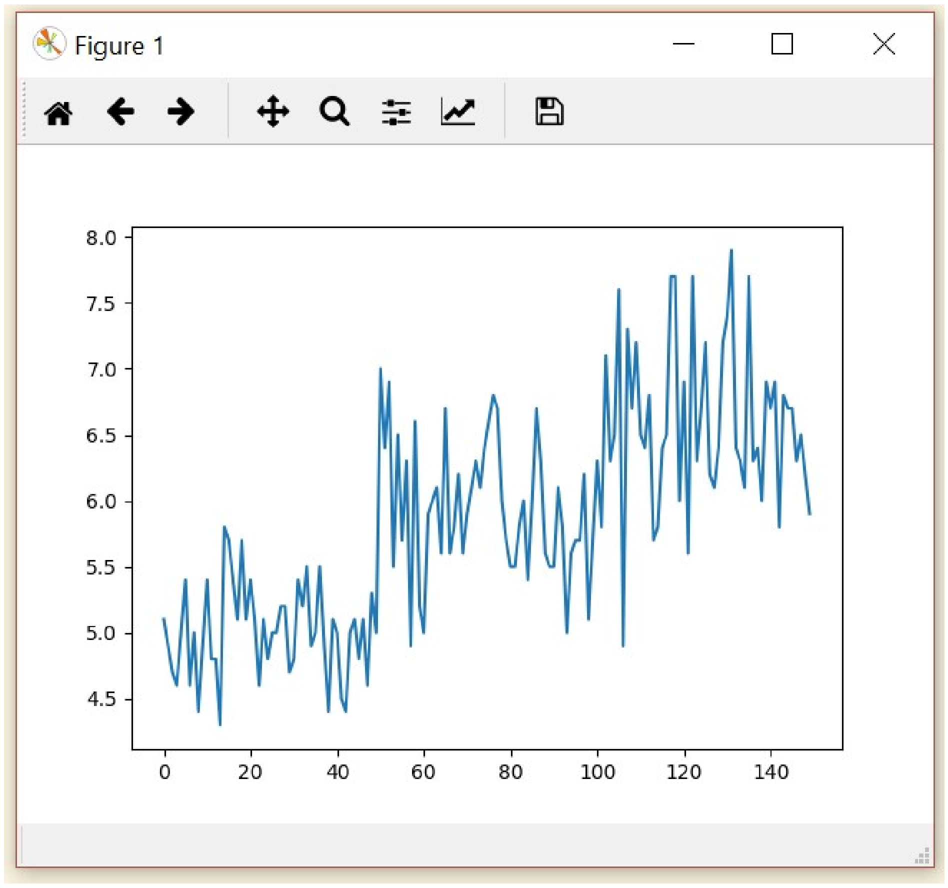
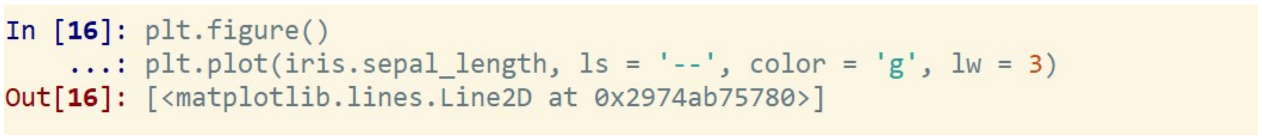


Fig 6.8 (click to enlarge)

You can modify the color and line style with a string indicator as shown in Figure 6.9:



```
In [16]: plt.figure()
    ...: plt.plot(iris.sepal_length, ls = '--', color = 'g', lw = 3)
Out[16]: [<matplotlib.lines.Line2D at 0x2974ab75780>]
```

Fig 6.9 (click to enlarge)

The parameters used for the plot in Figure 6.10 are:

ls: Line style, typically is a string of special character

color or c: color of the line

lw: line width

Color, line, and point style options can be seen here.

The output figure from the code is shown in Figure 6.10:

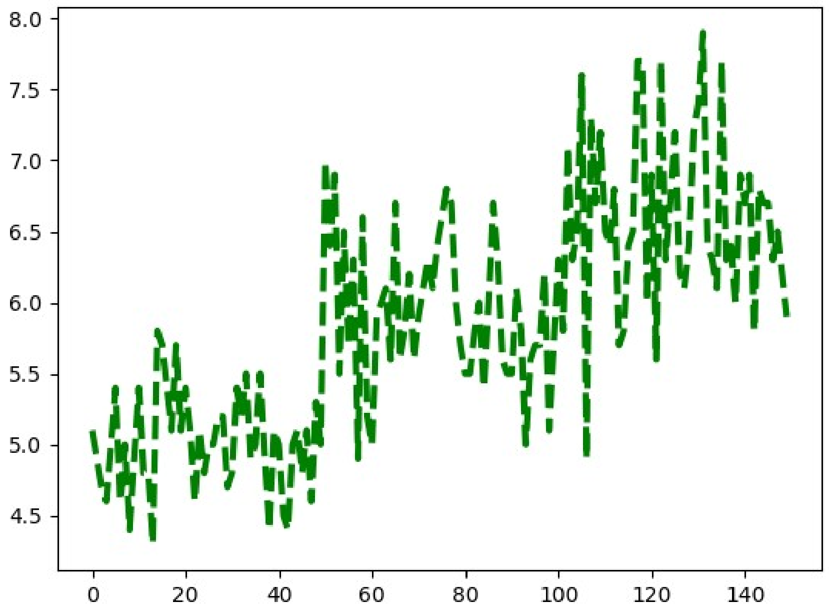
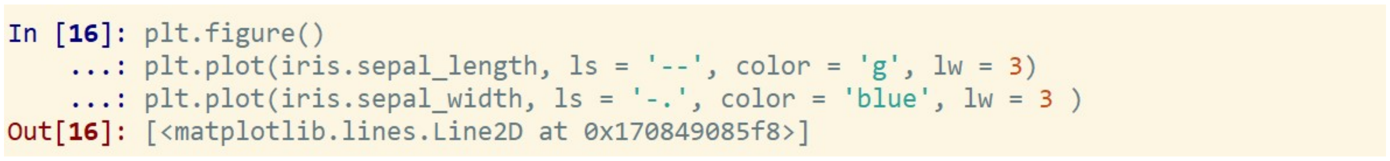


Fig 6.10 (click to enlarge)

If you want to create a single figure with multiple lines, we can simply call the *plot* function multiple times as shown in Figure 6.11 A&B:



```
In [16]: plt.figure()
    ...: plt.plot(iris.sepal_length, ls = '--', color = 'g', lw = 3)
    ...: plt.plot(iris.sepal_width, ls = '-.', color = 'blue', lw = 3 )
Out[16]: [<matplotlib.lines.Line2D at 0x170849085f8>]
```

Fig 6.11 (A) (click to enlarge)

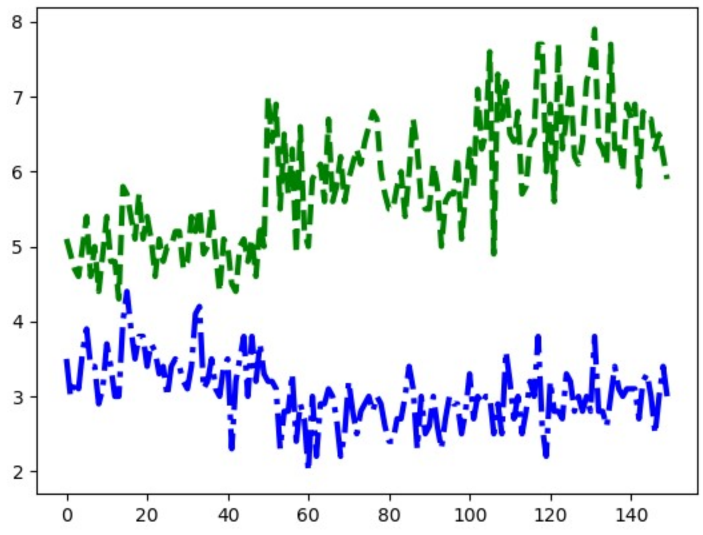


Fig 6.11 (B) (click to enlarge)

## Colors, Markers, and Line Styles

In matplotlib, you can specify the color using a string. Figure 6.12 shows built-in colors and thier visual representation:

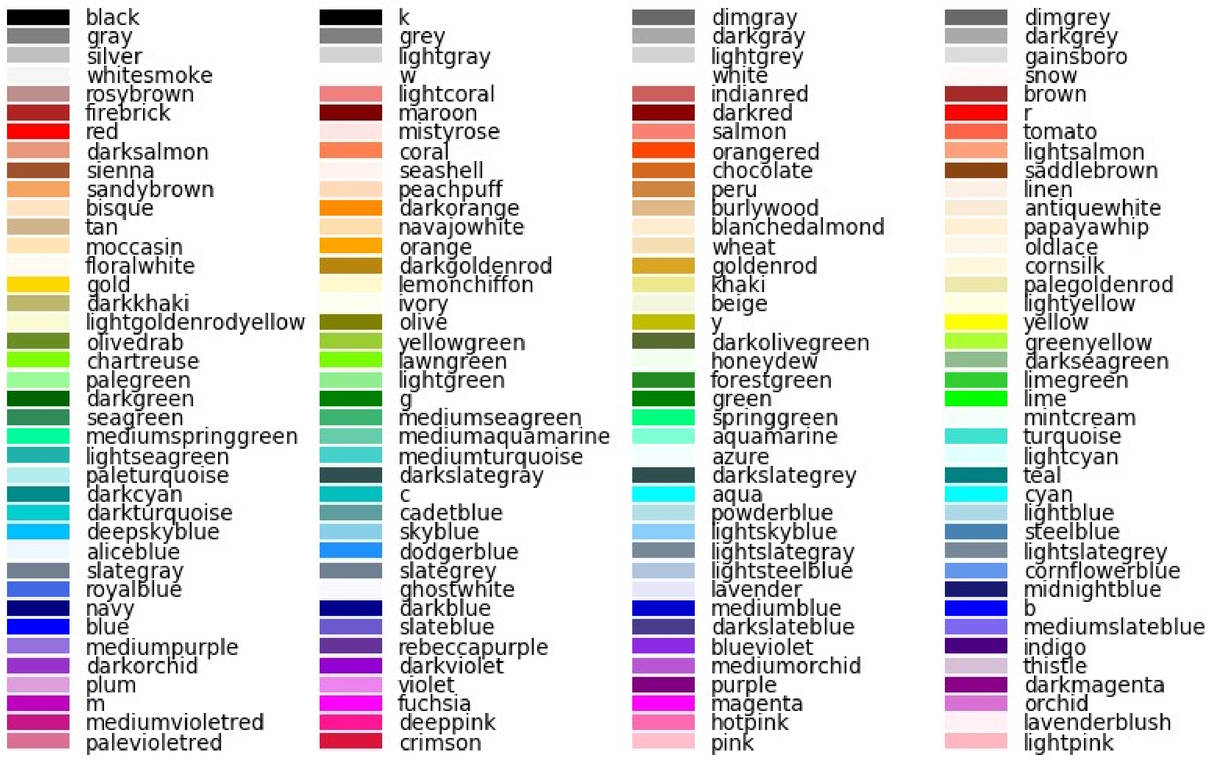


Fig 6.12 (click to enlarge)

The markers that are used in matplotlib are listed in Table 6.1.1:

Table 6.1.1: Common Markers in Matplotlib

| Marker | Description | Marker | Description | Marker | Description |
|---|---|---|---|---|---|
| "." | point | "1" | tri down | "*" | star |
| "," | pixel | "2" | tri up | "h" | hexagon1 |
| "o" | circle | "3" | tri left | "H" | hexagon2 |
| "v" | triangle down | "4" | tri right | "+" | plus |
| "^" | tirangle up | "8" | octagon | "x" | x |
| "<" | triangle left | "s" | square | "X" | x (filled) |
| ">" | triangle right | "p" | pentagon | "D" | diamond |
| "I" | vline | "p" | plus (filled) | "d" | thin diamond |

You can use all marker styles as line styles. In addition, there are another four line styles that are listed in Table 6.1.2:

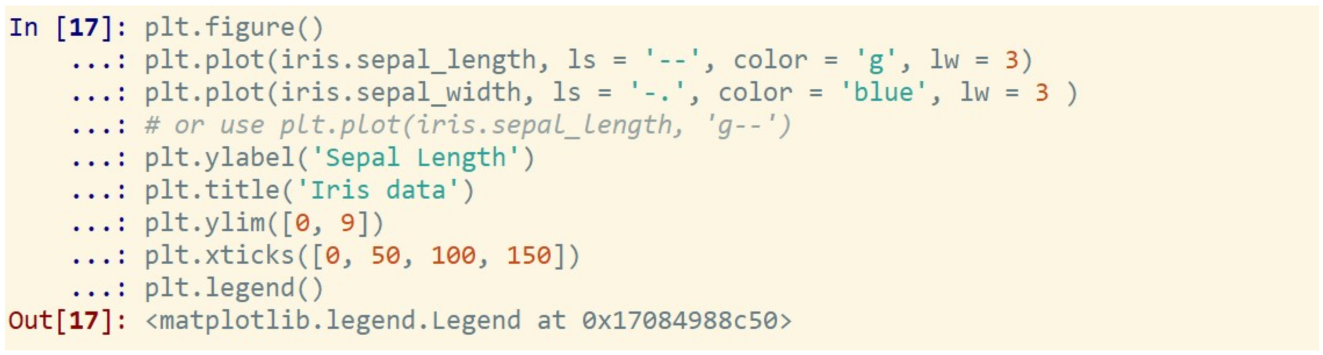
Table 6.1.2: Common Line Style in Matplotlib

| Linestyle | Description |
|---|---|
| '-' | Solid line style |
| '--' | Dashed line style |
| '-.' | Dash-dot line style |
| ':' | Dotted line style |

## Plot Properties

You can customize the plot properties using methods like *xlim*, *xticks*, *xlable*, *legend*, *title*, etc.

Figure 6.12 below shows an example of how to use *plt.ylabel*, *plt.title*, *plt.ylim*,*plt.xticks*, and *plt.legend*:



```
In [17]: plt.figure()
    ...: plt.plot(iris.sepal_length, ls = '--', color = 'g', lw = 3)
    ...: plt.plot(iris.sepal_width, ls = '-.', color = 'blue', lw = 3 )
    ...: # or use plt.plot(iris.sepal_length, 'g--')
    ...: plt.ylabel('Sepal Length')
    ...: plt.title('Iris data')
    ...: plt.ylim([0, 9])
    ...: plt.xticks([0, 50, 100, 150])
    ...: plt.legend()
Out[17]: <matplotlib.legend.Legend at 0x17084988c50>
```

Fig 6.12 (A) (click to enlarge)

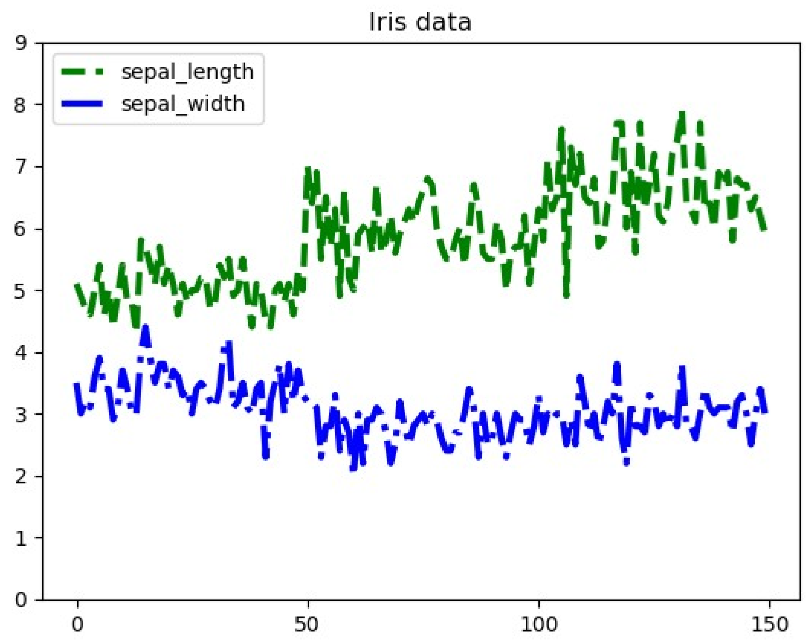


Fig 6.12 (B) (click to enlarge)

## Scatter Plot

Another commonly used plot type is the simple scatter plot. Instead of points being joined by line segments the data points are represented individually by a dot, circle or other marker shapes as shown in Figure 6.13 (B):

```
In [18]: plt.figure()
    ...: plt.scatter(iris.sepal_width, iris.sepal_length, s = 5, c = 'r')
    ...: plt.ylabel('Sepal Length')
    ...: plt.xlabel('Septal width')
    ...: plt.title('Iris Data')
    ...: plt.savefig('scatter.pdf')
```

Fig 6.13 (A) (click to enlarge)

For scatter plot, you need to use "s= a number" to change point size as shown in Figure 6.13 (A).

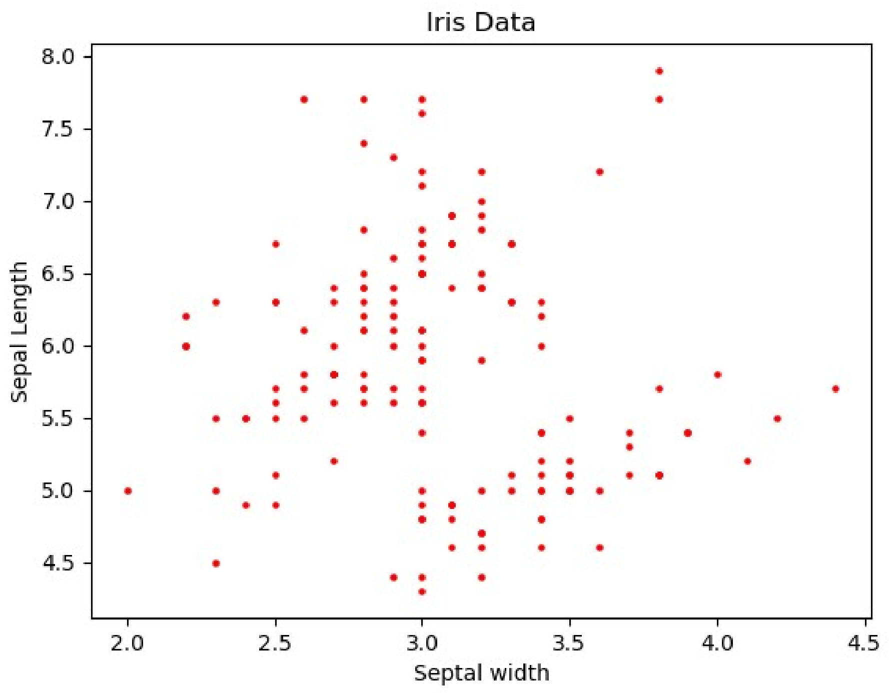


Fig 6.13 (B) (click to enlarge)

You can assign the color to different species by passing factorized 'species' column to color argument (c):

```
In [26]: plt.figure()
    ...: plt.scatter('sepal_width', 'sepal_length', data = iris, s = 7,
    ...:             c = iris.species.factorize()[0], label = iris.species.unique())
    ...: plt.ylabel('Sepal Length')
    ...: plt.xlabel('Septal width')
    ...: plt.title('Iris Data')
    ...: plt.xlim([1, 5])
    ...: plt.ylim([4, 9])
Out[26]: (4, 9)
```

Fig 6.13 (C) (click to enlarge)

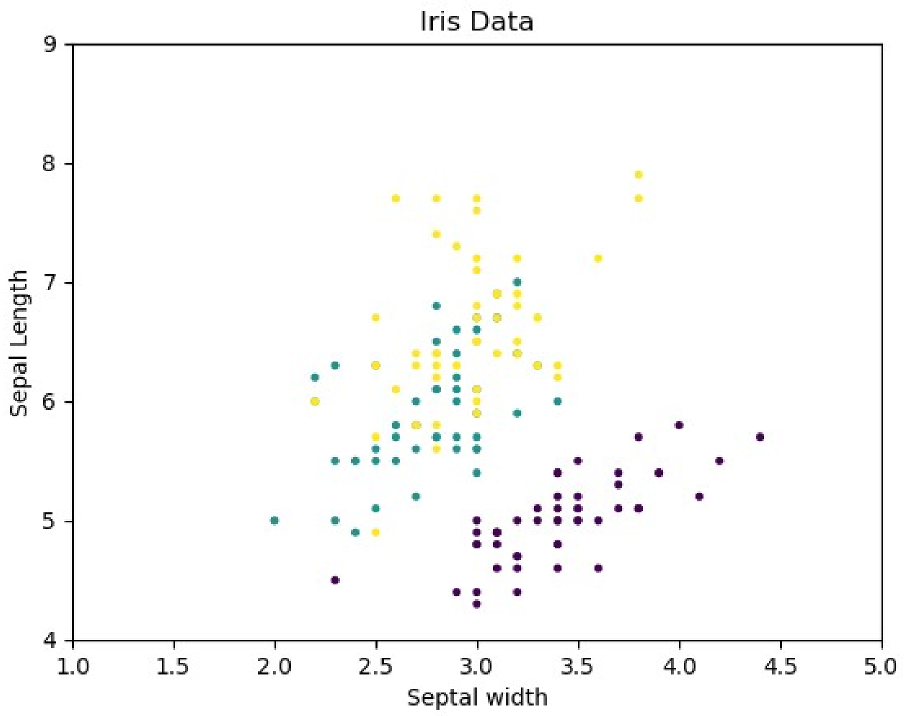
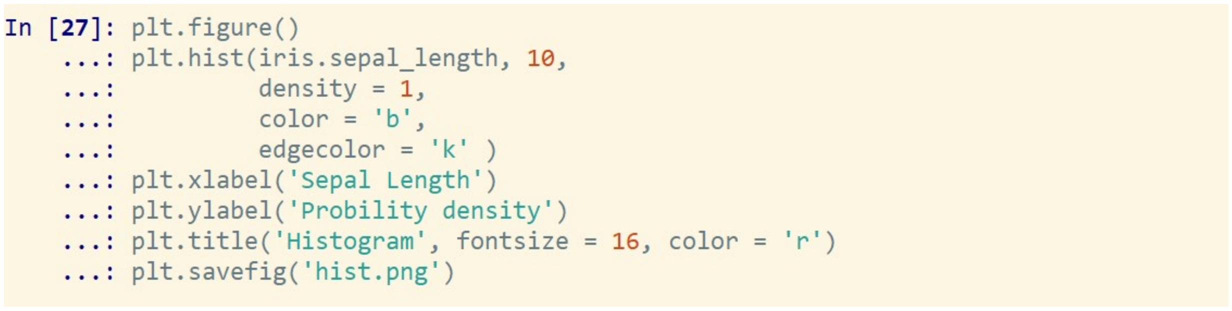


Fig 6.13 (D) (click to enlarge)

## Histograms

Histograms will help you better understand a dataset. You can use *plt.hist* function for this task.

Figure 6.14 A&B shows an example of a histogram of *iris.sepal length* with 10 bins. The density =1 (True) means the frequency is normalized. *Color* and *edgecolor* are used to set up the fill color and outline color, respectively.



```
In [27]: plt.figure()
    ...: plt.hist(iris.sepal_length, 10,
    ...:          density = 1,
    ...:          color = 'b',
    ...:          edgecolor = 'k' )
    ...: plt.xlabel('Sepal Length')
    ...: plt.ylabel('Probility density')
    ...: plt.title('Histogram', fontsize = 16, color = 'r')
    ...: plt.savefig('hist.png')
```

Fig 6.14 (A) (click to enlarge)

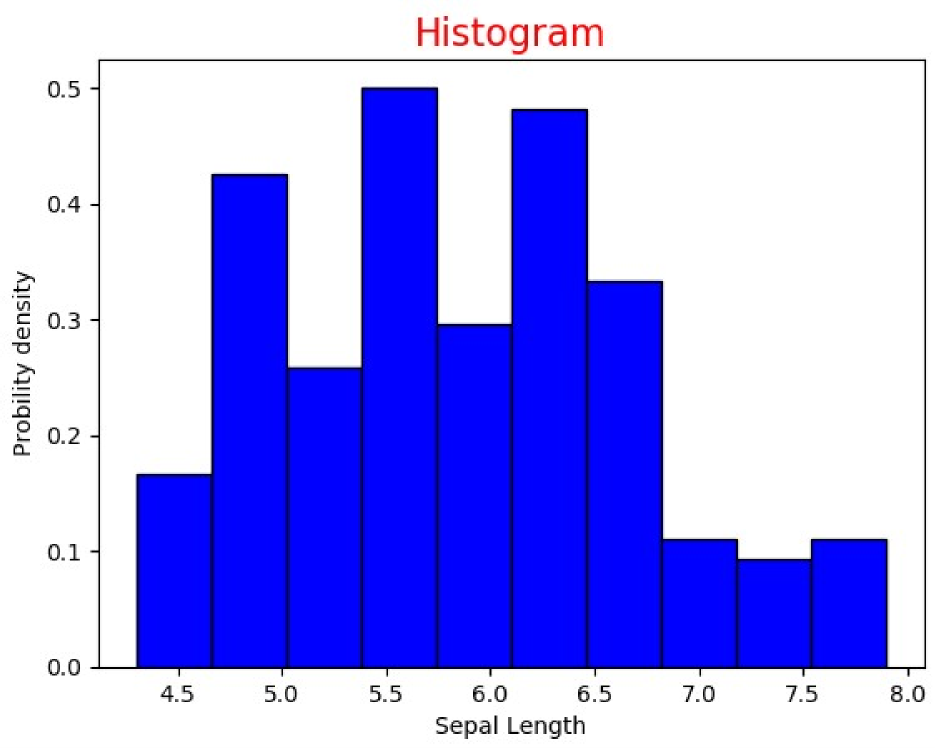


Fig 6.14 (B) (click to enlarge)

The function has many options to tune both the calculation and the display. Table 6.1.2 shows the common parameters that may be used for histograms.

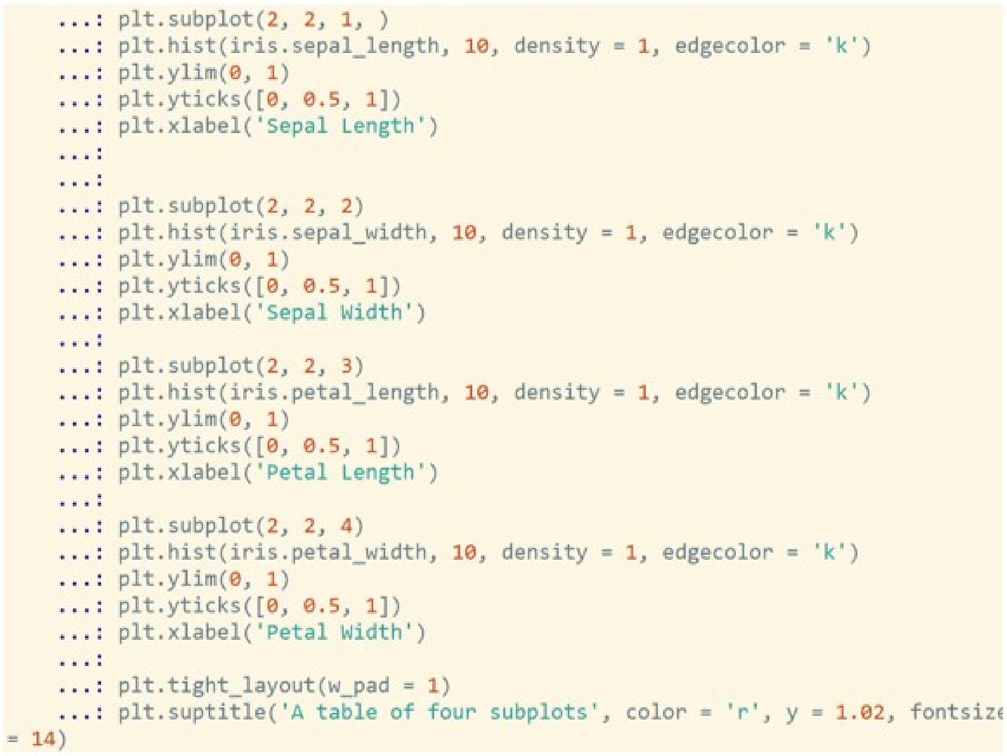
Table 6.1.2: Parameters for Histograms

| Parameters | Description |
|---|---|
| **bins** | Integer or a sequence |
| **density** | Boolean. True or 1 gives the probability density. False or 0 gives the counts |
| **orientation** | 'Horizontal' or 'Vertical' |
| **rwidth** | The relative width of the bars as a fraction of the bin width |

## Subplots

Sometimes it is very useful to put different views of data together. Matplotlib use the subplots function to create several plots in a single figure. Below you can see two different ways to create subplots in Matplotlib.

The first one shown in Figure 6.15 is to initial each subplot by using plt.subplots(n,m,i), where n represents the number of rows, m represents the number of columns, and I represents the $i^{th}$subplot. The range of I is [1, $n_x$m], since it only contains $n_x$m subplots in a figure.



```
    ...: plt.subplot(2, 2, 1, )
    ...: plt.hist(iris.sepal_length, 10, density = 1, edgecolor = 'k')
    ...: plt.ylim(0, 1)
    ...: plt.yticks([0, 0.5, 1])
    ...: plt.xlabel('Sepal Length')
    ...:
    ...:
    ...: plt.subplot(2, 2, 2)
    ...: plt.hist(iris.sepal_width, 10, density = 1, edgecolor = 'k')
    ...: plt.ylim(0, 1)
    ...: plt.yticks([0, 0.5, 1])
    ...: plt.xlabel('Sepal Width')
    ...:
    ...: plt.subplot(2, 2, 3)
    ...: plt.hist(iris.petal_length, 10, density = 1, edgecolor = 'k')
    ...: plt.ylim(0, 1)
    ...: plt.yticks([0, 0.5, 1])
    ...: plt.xlabel('Petal Length')
    ...:
    ...: plt.subplot(2, 2, 4)
    ...: plt.hist(iris.petal_width, 10, density = 1, edgecolor = 'k')
    ...: plt.ylim(0, 1)
    ...: plt.yticks([0, 0.5, 1])
    ...: plt.xlabel('Petal Width')
    ...:
    ...: plt.tight_layout(w_pad = 1)
    ...: plt.suptitle('A table of four subplots', color = 'r', y = 1.02, fontsize
= 14)
```

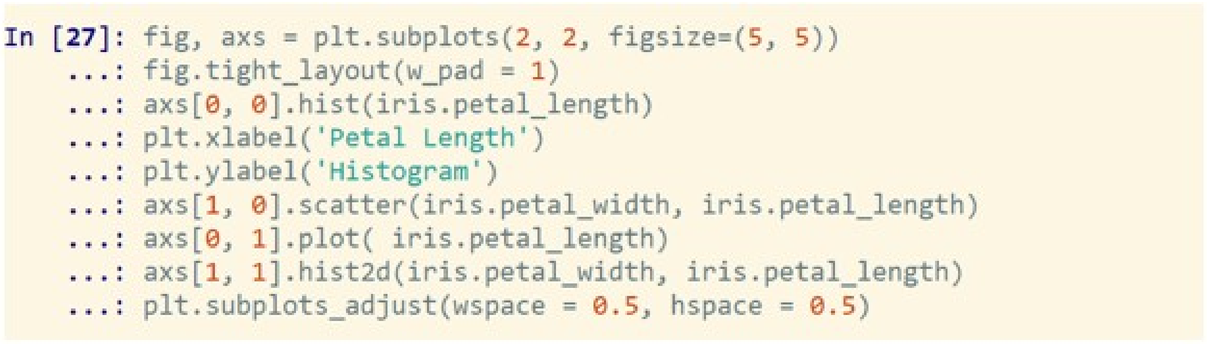
Fig 6.15 (A) (click to enlarge)

Fig

6.15

(A)

The second one is to use plt.subplots(n,m) to create a new figure and a NumPy array containing nxm subplot objects, and assign them to two variables: fig and axs. Each subplots can be indexed like two-dimensional array as shown in Figures 6.16 A&B:



```
In [27]: fig, axs = plt.subplots(2, 2, figsize=(5, 5))
    ...: fig.tight_layout(w_pad = 1)
    ...: axs[0, 0].hist(iris.petal_length)
    ...: plt.xlabel('Petal Length')
    ...: plt.ylabel('Histogram')
    ...: axs[1, 0].scatter(iris.petal_width, iris.petal_length)
    ...: axs[0, 1].plot( iris.petal_length)
    ...: axs[1, 1].hist2d(iris.petal_width, iris.petal_length)
    ...: plt.subplots_adjust(wspace = 0.5, hspace = 0.5)
```

Fig 6.16 (A) (click to enlarge)

Fig

6.16

(B)

The *subplots adjust* function is used to adjust the sepearations between figures.

Transcript

In this video, I will introduce how to plot figures with matplotlib package.

Here we pyplot module of matplotlib, let’s import it as plt first. Also import pandas and os.

Again, we will use iris data as an example. Let’s change the working directory and read iris data into Python.

If you don’t change the preference, the plots will be shown in Console and you need to run entire block if you have multiple line for it.

Here we use an interactive way. You can click tools -> preferences -> Ipython console -> Graphics. And choose automatic here and you need to restart your Spyder.

Here I just rerun line 8-16.

Plot function can create a simple line plot. After run line 20, a new window will pop up and you can play with it in this window. Go to previous or next plots. Move it, Enlarge it, change the border, or you have subplots, you can play with hspace and wspace. Under axes tab, you can Change limit, label, scale and under curves tab, you can change the style. You don't need to write codes to change these properties. Finally, you can save your figures here.

If you use codes to change the properties, the plots will change correspondingly here. Line 23 use green dash to plot iris sepal length. The line width is 3. Line 25 will add ylabel here. Line 26 adds title on the top of the plot. Line 27 changes the limits on y-axis. Line 28 changes the tickers from 0, 20, 40 … to 0, 50, 100, 150.

Scatter function is used to create scatter plots. Line 23 create the scatter plots with spal width as x-axis and sepal width as y-axis.  S means size and we use 5 here. C is color, and we use r red. After we modify x, y labels and title. You can also use savefig function to save the figure.

You can assign different colors to the points according to its group. Here, you assign an array  like object to c.

Next, we introduce how to plot histogram in Python

Hist is the function for this task. This is the histogram for speal length.  10 means 10 bins here. Density =  1 means to normalize the count. If it is 0, it will use the count for y-axis. Color means filling color, edgecolor means the color for the edge of boxes. You can also change the size and color of the title. There are too many arguments to try, you can play with it.

Next we show how to create subplots. There are two ways to do it.

After initial a figure, use subplot function, The first number is how many rows, the second number is how many columns, the third number is to spcify which subplots you want to plot. The range of the third number should be 1 to the production of the first number and the second number. You just to plot what you want to plot under each submplot. For the first one, we plot the histogram for sepal length, the second one is the histogram for sepal width, the third one the histogram for petal length, and the fourth one the histogram for peal width.

You can also assign the figure, and subplots to variables, like fig and axs here. Axs is 2x2 array of plot objects.

Axs[0, 0] means the first figure, and we plot the histogram for petal length.

Axis[1,0] is the subplot on bottom-left, and is a scatter plot of petal width and petal length

Axis[0, 1] means the subplot on top-right. And it is  line plot for petal length

Axis[1, 1] is the 2d histogram.

Finally, you can adjust the spaceing between the subplots.

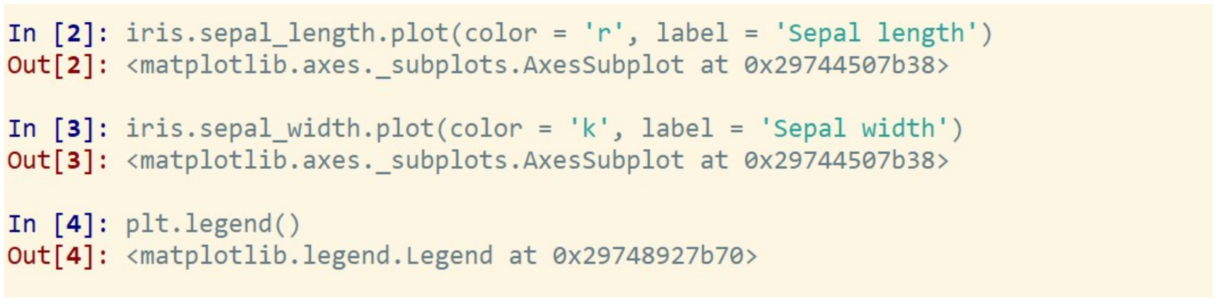
# Lesson 6.2: Plotting with pandas

In pandas, Series and DataFrame have a plot attribute for making some basic plot types. Since the plot methods in pandas are developed on matplotlib you can customize the label, ticker, etc. by using matplotlib functions.

*(click tabs to learn more)*

## Line Plots

By default, *plot* method makes line plots. Figure 6.17 shows an example of the line plot or 'sepal_length' and 'sepal_width' variables. A legend can be added by using *plt.legend*()



```
In [2]: iris.sepal_length.plot(color = 'r', label = 'Sepal length')
Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x29744507b38>

In [3]: iris.sepal_width.plot(color = 'k', label = 'Sepal width')
Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x29744507b38>

In [4]: plt.legend()
Out[4]: <matplotlib.legend.Legend at 0x29748927b70>
```

Fig 6.17 (A) (click to enlarge)

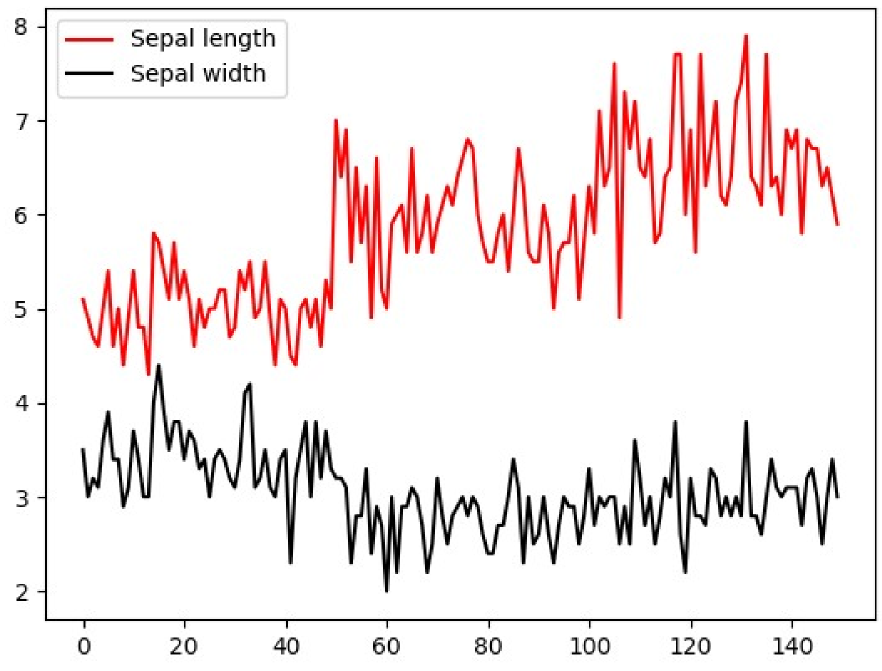
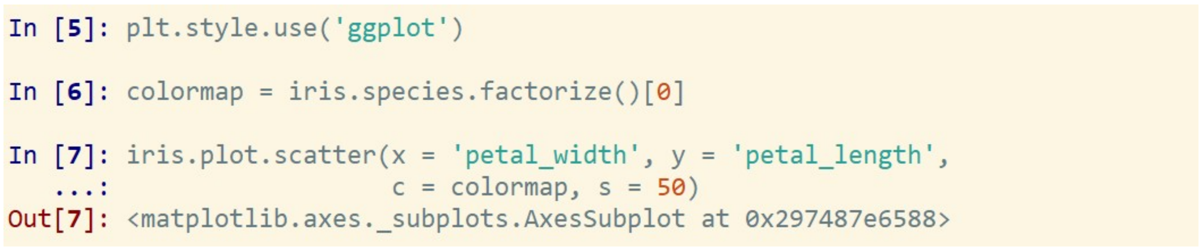


Fig 6.17 (B) (click to enlarge)

## Scatter Plots

Figure 6.18 A&B shows an example of how to make scatter plots. By settin the plot style as 'ggplot', and you can get similar aesthetic to the ggplot2 package in R:



```
In [5]: plt.style.use('ggplot')

In [6]: colormap = iris.species.factorize()[0]

In [7]: iris.plot.scatter(x = 'petal_width', y = 'petal_length',
   ...:                    c = colormap, s = 50)
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x297487e6588>
```

Fig 6.18 (A) (click to enlarge)

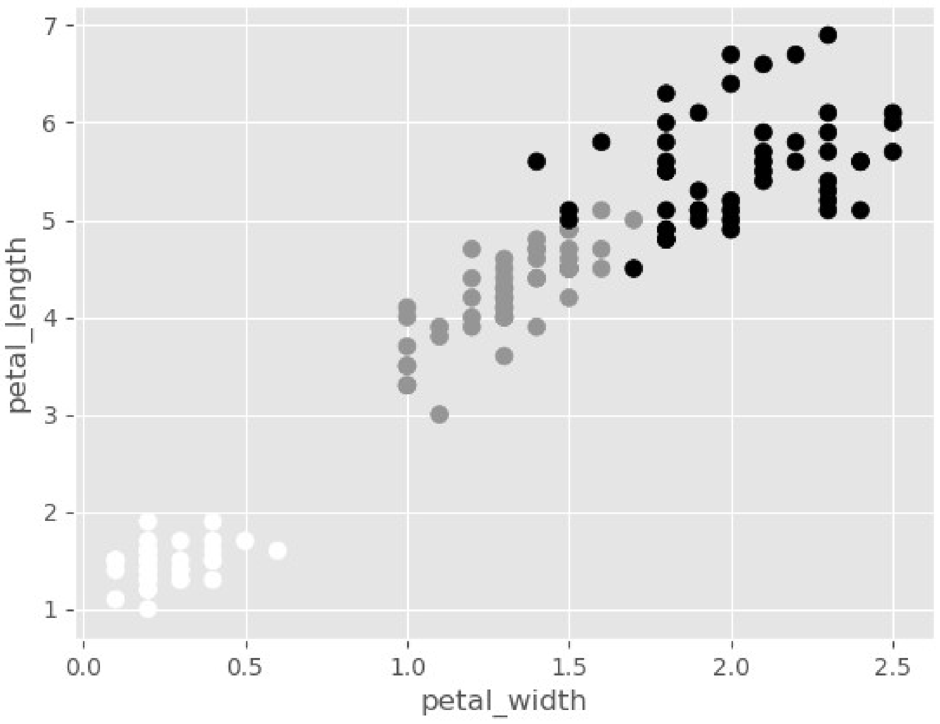
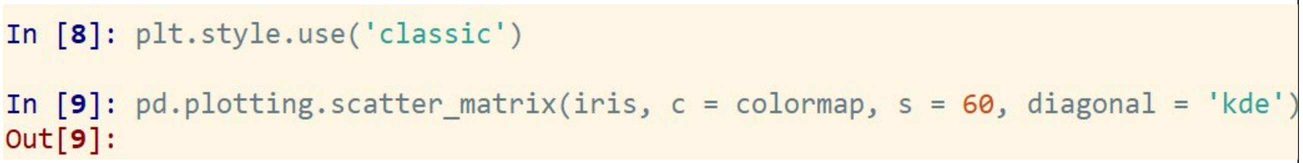


Fig 6.18 (B) (click to enlarge)

## Scatter Plot Matrix

The scatter plot matrix is very to be realized in pandas. Let's try a differnet style for Figure 6.19 A&B:



```
In [8]: plt.style.use('classic')

In [9]: pd.plotting.scatter_matrix(iris, c = colormap, s = 60, diagonal = 'kde')
Out[9]:
```

Fig 6.19 (A) (click to enlarge)

*diagonal*= 'kde' is used to plot the diagonal histogram with kernal density estimation (https://en.wikipedia.org/wiki/Kernel_density_estimation) .

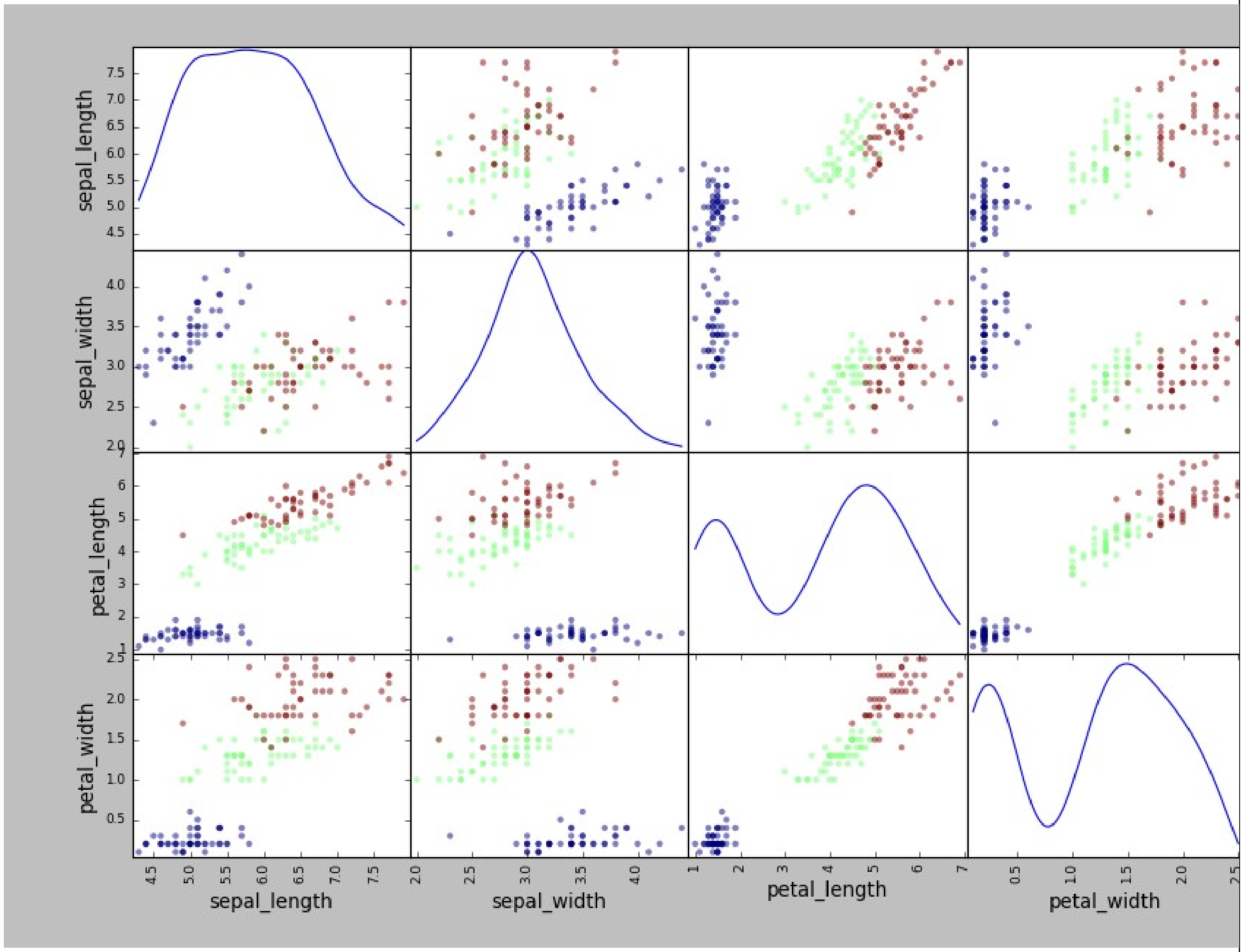
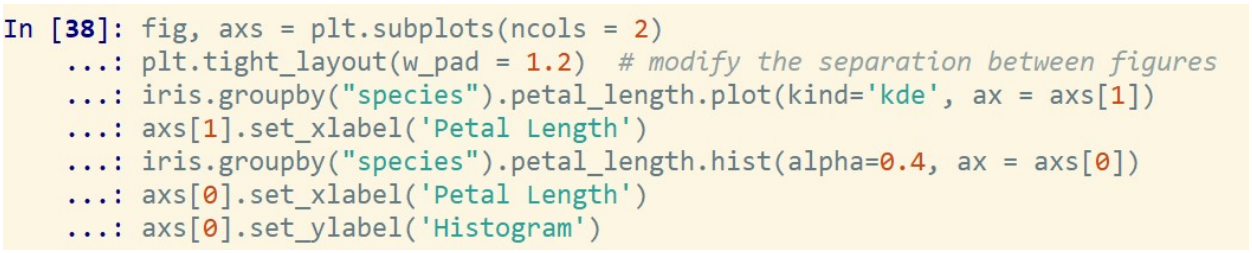


Fig 6.19 (B) (click to enlarge)

## Histogram

Figure 6.20 A&B are subplots with the histogram on the left side and the density plots on the right side (shown in Figure 6.19 B). You have to use the subplots function of matplotlib to create an empty *subplots* first:



```
In [38]: fig, axs = plt.subplots(ncols = 2)
    ...: plt.tight_layout(w_pad = 1.2)  # modify the separation between figures
    ...: iris.groupby("species").petal_length.plot(kind='kde', ax = axs[1])
    ...: axs[1].set_xlabel('Petal Length')
    ...: iris.groupby("species").petal_length.hist(alpha=0.4, ax = axs[0])
    ...: axs[0].set_xlabel('Petal Length')
    ...: axs[0].set_ylabel('Histogram')
```

Fig 6.20 (A) (click to enlarge)

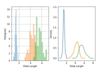
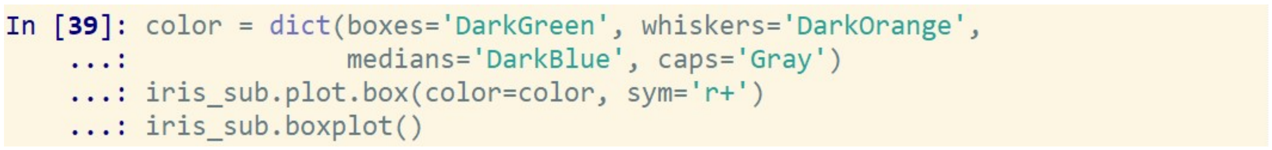


Fig 6.20 (B) (click to enlarge)

## Boxplots

Figure 6.21 A&B shows an example of how to get boxplots for four variables of iris data:



```
In [39]: color = dict(boxes='DarkGreen', whiskers='DarkOrange',
    ...:               medians='DarkBlue', caps='Gray')
    ...: iris_sub.plot.box(color=color, sym='r+')
    ...: iris_sub.boxplot()
```

Fig 6.21 (A) (click to enlarge)

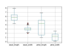


Fig 6.21 (B)

The *dict* color is used to map different colors to boxes, whiskers, medians and caps.

Transcript

In this video, I will introduce how to use pandas for plotting.  Pandas use matplotlib to create various plot methods

First, let create simple line plots, you can just select the column you want to plot and then dot plot function and adding few arguments to change the plot properties if you want. Here we use a red color and add a label.

You can add another line for sepal length.

You can directly use matplotlib functions to modify the plots. Here we use plt.legend() to add legend for it.

Matplotlib has some built-in plot style, here we change to ggplot style, grayish with grid.

For scatter plot, you need to use plot method under dataframe and then choose to scatter. In the scatter method, you can specify the x, y by using the column name. we can create color vector colormap first by passing the category coding.

It is very convenient to plot scatterplot matrix.

First, we try a different style classic.

You can find scatter_matrix under pd.plottting, and parsing entire data to it. Still use different color for different species and size is 60, in the diagonal of the plot, we use kde kernel density estimate for histogram style.

Similarly, you can use the second way which is introduced in 6.1 for creating subplots.

You can create a two-column one-row subplots by using ncols = 2.

If you want to plot histograms for different species, you can group the dataframe by the species column, then select the column you want to plot. Here we group iris by species and then plot the histogram for petal length of different species. You can choose either KDE and the standard histogram by default. Here we plot KDE on the right and standard histogram on the left.

You change the labels by using set_xlabel or set_ylabel.

Last, we show how to create box plots.

Frist, we can create different colors for boxes, whiskers, medians, and caps.

We only select numeric columns since boxplots can only plot numeric columns.

First, you can either choose plot.box or boxplot directly to create the box plots.

Lesson 6 References (4 of 4)

## Lesson 6 References

- https://pandas.pydata.org/pandas-docs/stable/visualization.html (https://pandas.pydata.org/pandas-docs/stable/visualization.html)
- https://matplotlib.org/tutorials/index.html (https://matplotlib.org/tutorials/index.html)