Chapter 4 Data visualization

4.1 Introduction to data visualization

Data visualization refers to the representation of data through use of graphics. Making informative visualization is an important task in data analysis, no matter as a part of the exploratory process or as a way of generating ideas for models.

Python has many add-on libraries for making static or dynamic visualizations. In this chapter, we will introduce a library called **matplotlib** which stands for mathematics-plot-library and the techniques of making various graphs for presenting statistical data and also the result of artificial intelligence.

In fact, matplotlib is a desktop plotting package designed for creating publication quality plots. There are many modules under it including **pyplot**. We usually import it by:

```
import matplotlib.pyplot as plt
```

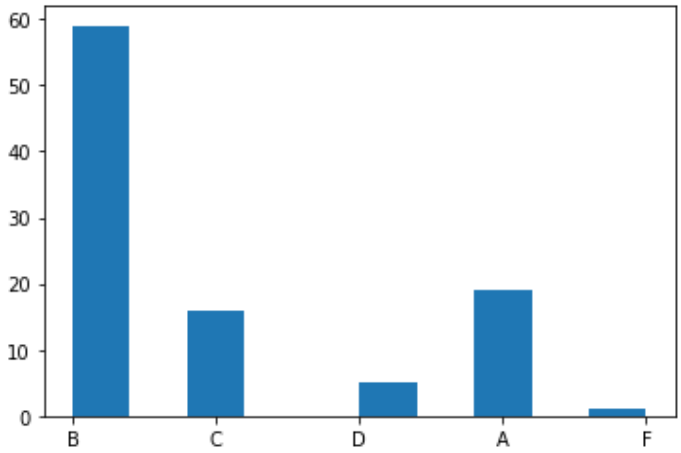
4.2 Frequency plot

Consider a set of one dimensional data. The values can be either numerical values (e.g. marks of students) or non-numerical values (e.g. letter grades of students). **Frequency plot** is based on the number of occurrence of each unique value. Let's use an example to illustrate different plots. The file "ama1234.csv" contains the marks and letter grades of 100 students. We can first read the file as a single DataFrame, then extract the two columns as two Series.

```
import pandas as pd
result = pd.read_csv("ama1234.csv")
marks = result["marks"]
grades = result["grades"]
```

Base on the grades, we can directly plot a **histogram** using `hist()` to show the frequency of students obtaining each grades. The `show()` method displays the plot.

```
plt.hist(grades)
plt.show()
```

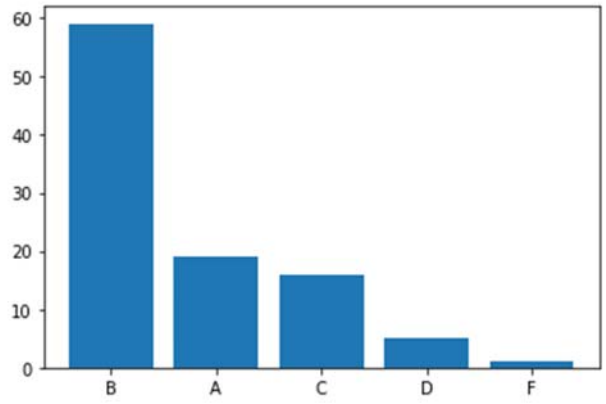


In the previous chapter, we have introduced the `value_counts()` method in Pandas which gives a frequency table of a set of data by counting the frequency of each unique value. The result is a Series with the index being each unique value and the values being the frequency of each unique value. We might first store it as two arrays:

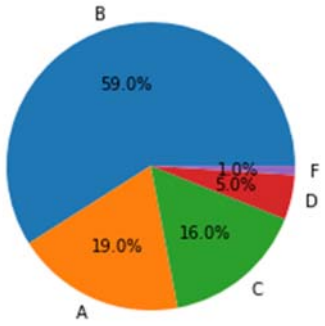
```
x = grades.value_counts().index
y = grades.value_counts().values
```

Using these two arrays x(index) and y(values), we can plot a **bar chart** or a **pie chart** to illustrate frequency of the set of data. For bar chart, two arrays are required for the x-axis and y-axis. For pie chart, only an array of value is necessary. We might also put the labelling and auto-percentage optionally.

```
plt.bar(x,y)
plt.show()
```




```
plt.pie(y,labels=x,autopct='%1.1f%%')
plt.show()
```



4.3 Data binning

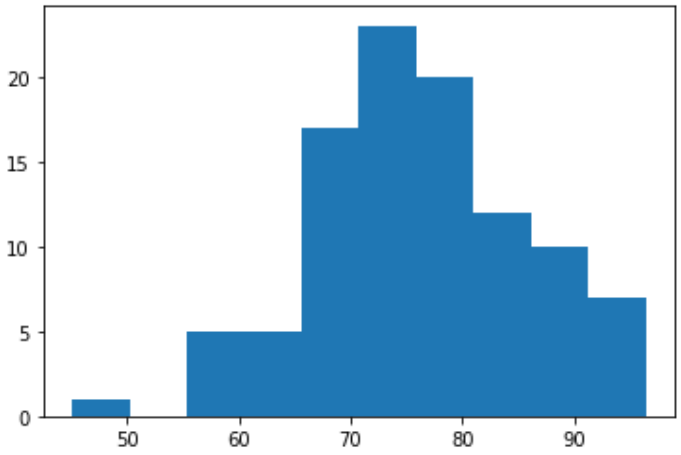
If we look at the Series of marks in the example problem, we can see that the entries are float point numerical values ranged between 0 to 100, corrected to 1 decimal point. It would be unrealistic to make a frequency plot of each unique mark. Instead, we might group similar marks together. In statistics, **data binning** is a way to group numbers of more-or-less continuous values into a smaller number of "bins". After binning, we might plot a histogram or a density plot base on the frequency of binned values.

In pyplot, when we plot a histogram of an array of values from a continuous variable, it will be automatically binned. We can also specify the binning criteria by the number of even width intervals, or by the end points between intervals by the following syntax respectively:

```
plt.hist(data_set, bins=number_of_bins)
plt.hist(data_set, bins=[point_0,point_1,...,point_n])
```

```
plt.hist(marks)
```

```
(array([ 1.,  0.,  5.,  5., 17., 23., 20., 12., 10.,  7.]),
 array([45.1 , 50.23, 55.36, 60.49, 65.62, 70.75, 75.88, 81.01, 86.14,
        91.27, 96.4 ]),
 <a list of 10 Patch objects>)
```

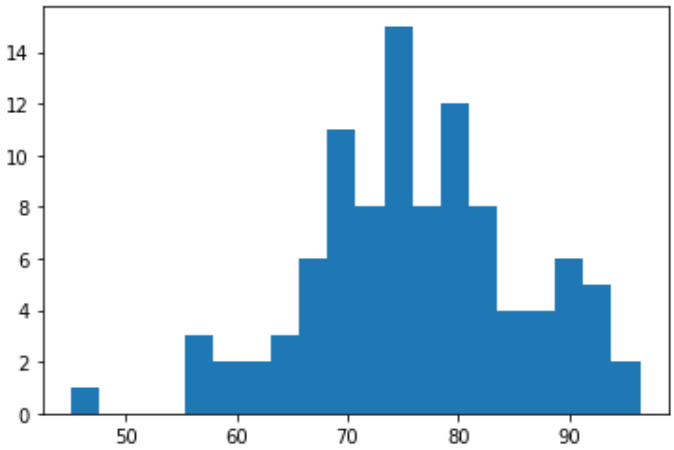


In the example, without putting any parameter, the data is automatically put into 10 bins of equal width along the real number line. The end points of the intervals and the frequency of data in each interval are shown in the array.

Suppose we would like to use 20 bins with narrower interval, simply set the parameter 20.

```
plt.hist(marks,bins=20)
```

```
(array([ 1.,  0.,  0.,  0.,  3.,  2.,  2.,  3.,  6., 11.,  8., 15.,  8.,
        12.,  8.,  4.,  4.,  6.,  5.,  2.]),
 array([45.1  , 47.665, 50.23 , 52.795, 55.36 , 57.925, 60.49 , 63.055,
        65.62 , 68.185, 70.75 , 73.315, 75.88 , 78.445, 81.01 , 83.575,
        86.14 , 88.705, 91.27 , 93.835, 96.4  ]),
 <a list of 20 Patch objects>)
```



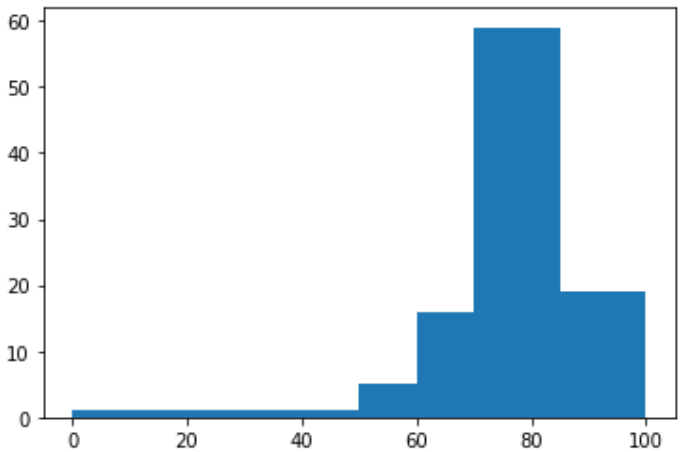
Suppose the rubric of ama1234 is as follows:

$$A: [85,100], B: [70,85), C: [60,70), D: [50,59), F: [0,50)$$

We may form an array with the threshold marks together with 0 mark and full mark (100) to divide the interval for data binning. Notice that the width of each interval is not even. This is illustrated by width of rectangles on the histogram.

```
plt.hist(marks,bins=[0,50,60,70,85,100])
```

```
(array([ 1.,  5., 16., 59., 19.]),
 array([  0,  50,  60,  70,  85, 100]),
 <a list of 5 Patch objects>)
```



4.4 Line graph

Some set of data is dependent to a continuous variable. For example, stock price is dependent on the time variable. To present such data, we may use a **line graph**. This is good for showing the trend and local extreme values. As an example, we will try to study the stock price of two companies, Apple (AAPL) and Microsoft (MSFT). The files "AAPL.csv" and "MSFT.csv" contains the historical data in 5 years. The column of adjusted close price is stored as a Series.

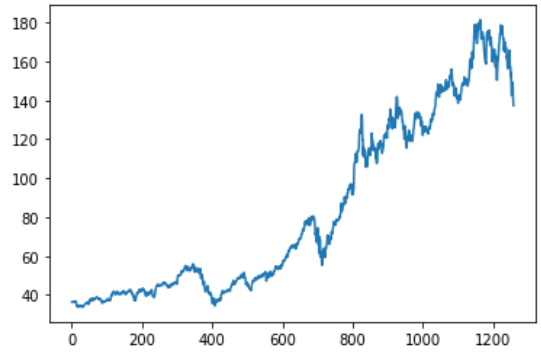
```
apple_data = pd.read_csv("AAPL.csv")
apple_price = apple_data["Adj Close"]
msft_data = pd.read_csv("MSFT.csv")
msft_price = msft_data["Adj Close"]
```

In pyplot, we can directly use `plot()` to make a line graph of a Series or array. Moreover, the colour and style of the line can be adjusted by the commands inside the brackets.

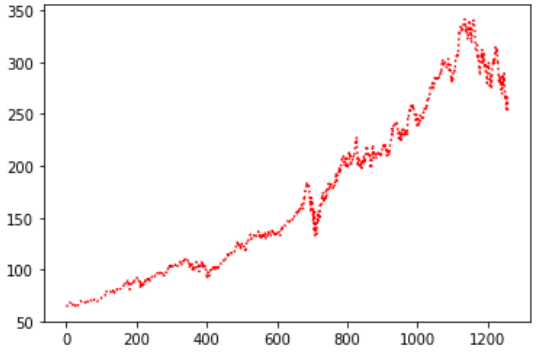
command	colour
`'k'`	black
`'g'`	green
`'r'`	red
`'b'`	blue
`'y'`	yellow

command	style
`'--'`	dashed line
`':'`	dotted line
`'*'`	points with stars
`'o'`	points with circle
`'+'`	points with plus sign

```
plt.plot(apple_price)
plt.show()
```

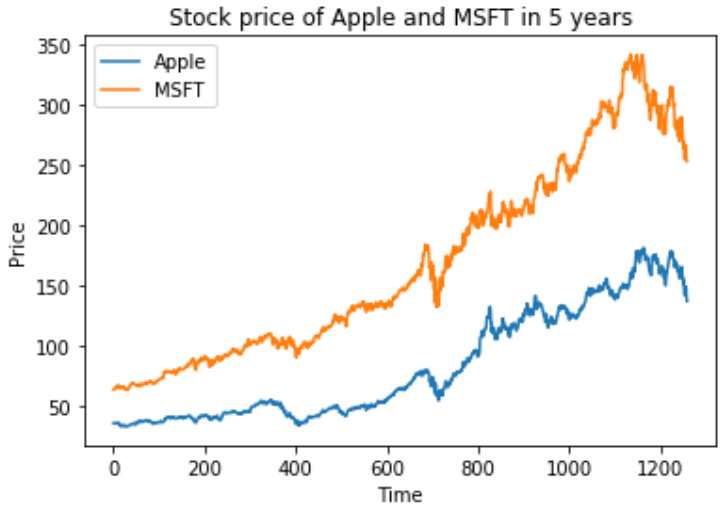



```
plt.plot(msft_price,'r:')
plt.show()
```



However, this is hard for comparison. In matplotlib, we can plot several lines on the same figure. To show information of the figure and distinguish the lines, we can also add title, labels and legend. This applies not only on line graph but also on the graphs we have introduced before. Upon executing the `show()` command, all these graphs and information before will be displayed on the same figure.

```
plt.plot(apple_price)
plt.plot(msft_price)
plt.xlabel("Time")
plt.ylabel("Price")
plt.title("Stock price of Apple and MSFT in 5 years")
plt.legend(["Apple","MSFT"])
plt.show()
```



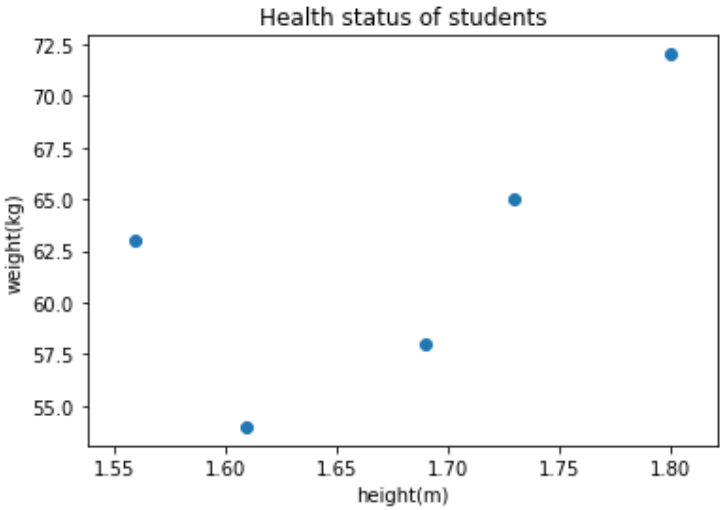
4.5 Scatter plot

A set of data might consist of more than one variables. An example is the health data from the previous chapter which contains two variables height and weight. For such data, the purpose of visualization is to show the relationship between the two variables. This can be illustrated by a **scatter plot**. More details will be discussed in a later chapter related to correlation and regression.

For example, we might want to study the relation between height and weight. After reading the csv file "health.csv" into a DataFrame, extract the columns representing the two variables (height and weight) into two Series. For each data index, a point with x-coordinate being its height and y-coordinate being its weight is plot on the figure. As a result, a scatter plot should contain as many points as the number of rows of the original DataFrame.

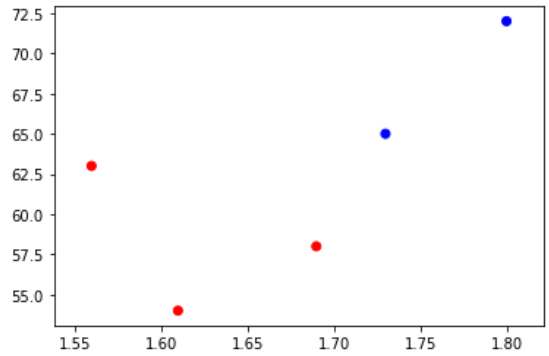
```
df1 = pd.read_csv("health.csv")
x = df1["height"]
y = df1["weight"]
```

```
plt.scatter(x,y)
plt.title("Health status of students")
plt.xlabel("height(m)")
plt.ylabel("weight(kg)")
plt.show()
```



In case the data points can be classified into different categories (e.g. male and female), we might assign colours to each point with `c=[colour0,colour1,...]`. If the data points are weighted, we might show each point with a different size using `s=[size0,size1,...]`.

```
plt.scatter(x,y,c=['b','r','b','r','r'])
plt.show()
```




```
plt.scatter(x,y,s=[300,100,150,200,400])
plt.show()
```