Fundamental Statistics with Python

Introduction

First, we summarize how to sort data in Python. Second, we learn how to use **NumPy** and **SciPy** packages for basic statistics. Given a data set, we will:

- Find the mean, maximum, minimum, and standard deviation.
- Find the median and integer quartile range.
- Count the number of students in each score range.
- Create charts to show the student distribution.

I. Sorting Data

Case 1. If the data are stored in a list, we can use **list.sort()** function to sort the data.

```
L = [5, 2, 3, 1, 4]
L.sort()
print(L) # [1, 2, 3, 4, 5]
```

Case 2. If the data are stored in a dictionary, we can first get the key-values pairs through the **dict.items()** function, and then use the Python built-in function **sorted()** to sort the key-value pairs according to the key.

```
d = {'x':3, 'a':89, 'k':5, 'd':10}
L = sorted(d.items())
print(L) # [('a', 89), ('d', 10), ('k', 5), ('x', 3)]
```

Case 3. If the data are stored in a Numpy ndarray, we can use **numpy.ndarray.sort()** function to sort the ndarray in-place.

```
import numpy as np
array = np.array([7, 5, 3, 2, 6, 1, 4])
array.sort()
print(array) # [1 2 3 4 5 6 7]
```

II. Counting Student Scores and Creating a Graph of Score Distributions with Median and Inter-Quartile Range

1. Import required library packages.

```
import numpy as np
from scipy import stats
import pandas as pd
from matplotlib import pyplot as plt
```

2. Download the data file **quiz1_scores.csv** from the course web page and save it in D drive.

3. Explore the **quiz1_scores.csv** file using Excel. We can see that the file has 3 columns – the index, student id, and scores. The index and 'student id' columns are not used for the statistics. We focus on the 'scores' column.

4. Load the file in Python. And, we store the values of the scores column in the **NumPy's** ndarray.

```
df = pd.read_csv('quiz1_scores.csv', index_col='index')
scores = df['scores'].values
```

Remark: **df** is a **DataFrame**; **df['scores']** is a **Series**, **df['scores'].values** is an **ndarray**.

5. Finding the median.
Remark: **Median** is the value separating the higher half from the lower half of a data set.

```
median = np.median(scores)
```

6. Finding the q-th percentile.
Remark: A **percentile** is a measure used in statistics indicating the value below which a given percentage of observations in a group of observations falls. For example, the 20th percentile is the value below which 20% of the observations may be found. Median is in fact the 50^{th} percentile.

```
per75 = np.percentile(scores, 75)
per25 = np.percentile(scores, 25)
```

7. Getting the **Inter Quartile Range** (IQR).
Remark: IQR is equal to the difference between 75th and 25th percentiles.

```
iqr = stats.iqr(scores)
```

8. Separate student scores into 10 groups evenly and count the number of scores in each group.

    a. Create labels for each group. We will use them as the *x* axis labels of the chart.

```
labels = ['below 10',
          '10 to 19',
          '20 to 29',
          '30 to 39',
          '40 to 49',
          '50 to 59',
          '60 to 69',
          '70 to 79',
          '80 to 89',
          '90 or above']
```

b. There are 10 groups, but we need to create a sequence with total 11 numbers (0, 10, 20, …, 90, 100) that define the bin edges.

```
edges = np.linspace(0, 100, 11)
```

The **linspace()** function returns total 11 **float** numbers - *0, 10, 20, 30, 40, 50, 60, 70, 80, 90*, and *100*.

You may also use the following statement to get the same values (but as integers).

```
edges = np.arange(0, 101, 10)
```

c. Use **NumPy's histogram()** function to tally scores into the appropriate interval and count the number of students in each interval.

Remark: A histogram is an accurate representation of the distribution of numerical data. To construct a histogram, the first step is to "bin" (or "bucket") the range of values, i.e., divide the entire range of values into a series of intervals, and then count how many values fall into each interval.

We have two methods to generate the histogram:
(1) specify the number of bins, then the range will be evenly divided into bins:

```
hist, edges = np.histogram(scores, bins=10)
```

(2) specify the bin edges:

```
hist, edges = np.histogram(scores, bins=edges)
```

The **histogram()** function returns two ndarrays – **hist** stores the resulted data that is the count of each interval; **bins** stores the list of bin edges. Notice that the size of **edges** is one more than the size of **hist**.

The following is the counting result:

Interval	Count
below 10	3
10 to 19	2
20 to 29	15
30 to 39	26
40 to 49	51
50 to 59	42
60 to 69	39
70 to 79	23
80 to 89	11
90 or above	8

9. Print the histogram result.

```
print('Score Distribution')
```

```
for i in range(len(hist)):
   print('%s: \t %d' % (labels[i], hist[i]))
```

10. Plot a graph.
    a. To plot a histogram with 10 bins, we need to find the center values for each bin:

```
bin_centers = 0.5 * (edges[1:] + edges[:-1])
```

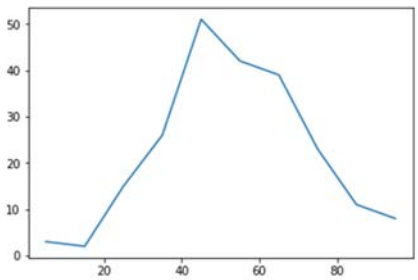
The statement above does the following thing:

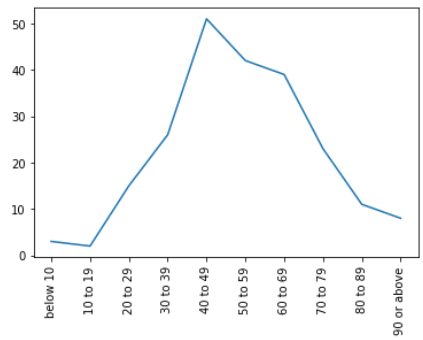
- **edges[1:]** is the subset of bins from 2nd element to last element.
- **edges[:-1]** is the subset of bins from 1st element to last second element.
- Add two *ndarrays* together.
- Then, multiply the elements by 0.5.

edges[1:]		edges[:-1]		temp		bin_centers
10		0		10		5
20	+	10	→	30	x 0.5 →	15
30		20		50		25
…		…		…		…
90		100		190		95

    b. Plot the graph using the bin_centers and histogram result data, and add meaningful labels on X axis. To see the step-by-step visual effect, you need to type the "**%matplotlib qt**" command in the IPython console first.

```
plt.plot(bin_centers, hist)
plt.xticks(bin_centers, labels, rotation='vertical')
```





You can also choose the bar style for the histogram:

```
plt.bar(bin_centers, hist, width=10)
plt.xticks(bin_centers, labels, rotation='vertical')
```

    c. Add an indicator of Inter Quartile range – Q1, Q2, and Q3

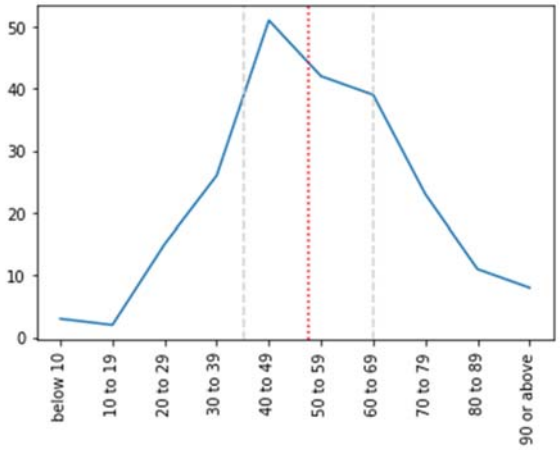
```
q1 = median - iqr/2
```

```
q2 = median
q3 = median + iqr/2

plt.axvline(x=q1, color='lightgrey', linestyle='--')
plt.axvline(x=q2, color='red', linestyle=':')
plt.axvline(x=q3, color='lightgrey', linestyle='--')
```

The **axvline()** function draws a vertical line in the graph. We can customize the line with the parameters:

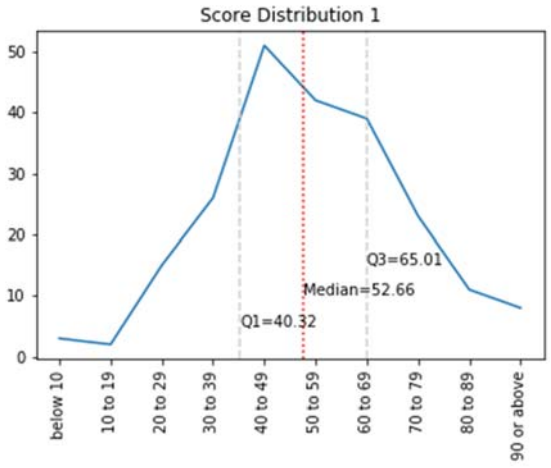
- *x* : the position on the x axis.
- *color* : name of the color
- *linestyle* : the line style – solid (-), dashed (--), dotted (:)

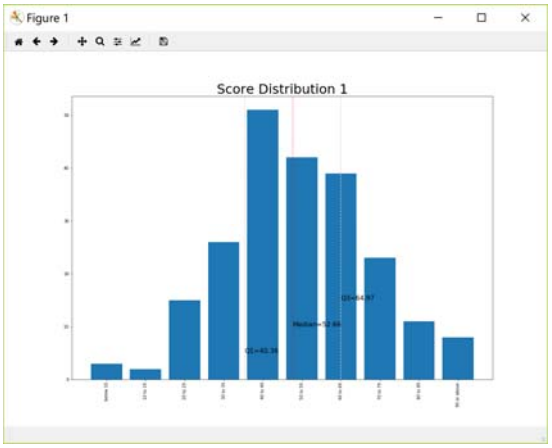


d. Add graph title and text labels.

```
plt.title('Score Distribution 1', fontsize = 24)

plt.text(q1, 5, 'Q1=%.2f'% q1, fontsize = 16)
plt.text(q2, 10, 'Median=%.2f'% q2, fontsize = 16)
plt.text(q3, 15, 'Q3=%.2f'% q3, fontsize = 16)
```





III. Creating a Graph with Mean, Standard Deviation, Reference Standard Normal Distribution Curve

In this section, we are going to create another graph using the same data set. The graph shows the mean, standard deviation, and a reference curve.

You may add the example code of this section after the previous one. At the end, you will see two graphs. The second graph is created by the code of this section. Otherwise, you need to repeat the steps from 1 to 8 of the previous section first.

11. Finding the mean, maximum, minimum, and standard deviation.

```
s_mean = scores.mean()
s_max = scores.max()
s_min = scores.min()
s_std = scores.std()
```

The functions include **mean()**, **max()**, **min()** and **std()** are the built-in functions of *ndarray*.

12. Compute the density data using the Probability Density Function.
    a. Use the **pdf()** function to compute the reference data.

```
pdf = stats.norm.pdf(range(100), s_mean, scale = s_std)
```

We provide a range (0 to 99, 100 elements), the peak (s_mean) and scale (standard deviation).

    b. Scale the resulted data up so as to match the histogram data scale.

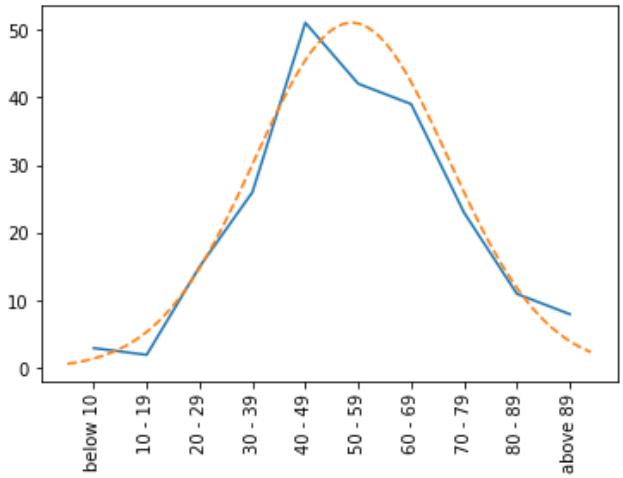
```
Pdf = pdf / pdf.max() * hist.max()
```

13. Plot the graph using the bins and histogram result data. Then, change the x-axis labels.

```
plt.plot(bin_centers, hist)
plt.xticks(bin_centers, labels, rotation='vertical')
```

14. Plot the density data as a reference curve.

```
plt.plot(Pdf, '--')
```



15. Add indicators for the mean and standard deviation.

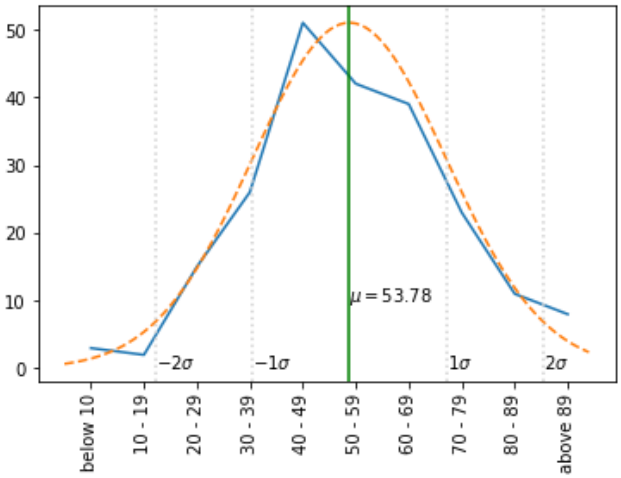
    a. Add line and label for mean:

```
plt.axvline(x=mean, color='green')
plt.text(mean, 10, '$\mu = %.2f$' % mean)
```

The string **'$\mu = %.2f$'** is a formatted string for printing the string with special symbols. The **\mu** in the string will be replaced by the μ symbol but the string must be enclosed in quotation marks and dollar signs (**'$ $'**).

    b. Add lines and labels for standard deviation:

```
for i in range(-2,3):
    if (i != 0):
        x = mean + std * i
        plt.axvline(x=x, color='lightgrey', linestyle=':')
        plt.text(x, 0, '$ %d \sigma$' % i)
```

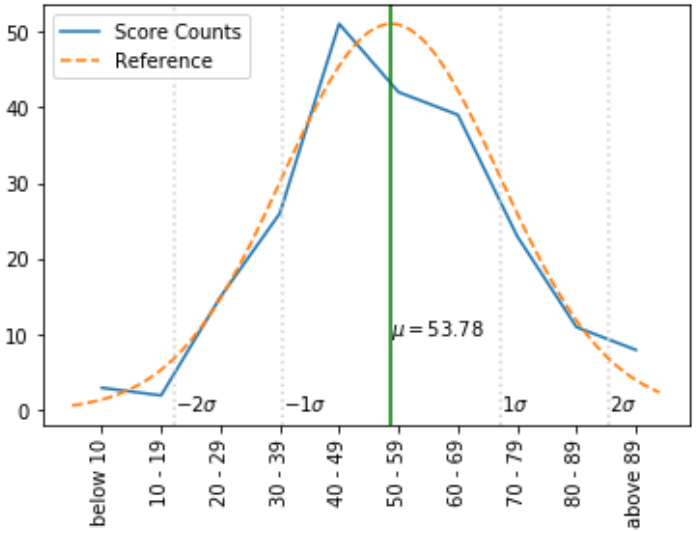


16. Draw the legend.

```
plt.legend(['Score Counts','Reference'])
```

There are total 7 lines in the graph including the score counts (result of the histogram), reference (the result of pdf), mean, and the lines about the standard division. But, we don't show all of them in the legend.

In the statement above, we pass a list with two strings that mean we want to show the first two lines with the labels stored in the list.



Exercise

Imagine that we have a survey data file retrieved from an online survey system. The columns contained in the survey data includes the responded date, customer IDs, salary range, age, number of children, gender and the choices of question 1 to 6. However, we only focus on the salary range.

Now, you need to create a graph to show the salary distribution with the median, IQR, and reference curve.

You need to:

- Separate the data in the 18 groups.
- Create a sequence (bins) with 19 numbers.
- The interval is 5,000
- Consider the follows labels for creating the histogram and plot the graph:

Label	Description
Below 15K	The salary is less than $15,000
15K <= $ < 20K	The salary is larger than or equal to $20,000 but less than $15,000.
20K <= $ < 25K	The salary is larger than or equal to $25,000 but less than $20,000.
…	…
$95K or above	The salary is larger than or equal to $95,000

Assume that the minimum amount is $10,000, and the maximum amount is $100,000.

- Create a histogram using **NumPy's histogram()** function with the salary data and bins.
- Compute density data using **SciPy's pdf()** function.
- Plot the graph using **Matplotlib**.

The sample output of the graph: