Practice exercises: Week 11 (Data analytics in Python + Algorithms in Python)

1. The "**StuInfo.txt**" contain student data that includes name, ID, scores of 5 subjects and birthday. The data in each row are split by single tab ("\t"), as shown below (not include the fields name of the 1st row).

Name	ID	SE	Math	En	Fi	PE	Birthday
Vivian	210001	72	98	77	79	68	21/10/2002
Nancy	210002	83	61	74	83	89	7/8/2003
Wilson	210003	98	95	82	72	90	27/2/2003
James	210004	94	60	66	72	69	8/3/2004
Bruce	210005	83	62	86	72	74	28/10/2002
Taylor	210006	76	91	63	94	68	2/3/2002

Read the data from that file vid **Pandas**→data frame and display. The sample run is here.

	Name	ID	SE	Math	En	Fi	PE	Birthday
0	Vivian	210001	72	98	77	79	68	21/10/2002
1	Nancy	210002	83	61	74	83	89	7/8/2003
2	Wilson	210003	98	95	82	72	90	27/2/2003
3	James	210004	94	60	66	72	69	8/3/2004
4	Bruce	210005	83	62	86	72	74	28/10/2002
5	Taylor	210006	76	91	63	94	68	2/3/2002
6	Kelly	210007	67	93	71	75	97	21/11/2002
7	Lane	210008	72	95	66	63	63	19/9/2002
8	Terry	210009	65	86	100	70	81	26/10/2002
9	Tina	210010	84	81	64	61	100	15/8/2002
10	Alexis	210011	63	77	76	60	96	20/9/2004
11	Benny	210012	78	84	96	72	99	24/4/2002
12	Kylie	210013	86	83	60	73	90	21/6/2004
13	Merry	210014	79	90	86	90	66	12/11/2003
14	Addison	210015	86	64	70	87	61	11/2/2003
15	Rachel	210016	87	88	97	100	75	5/3/2002
16	Yilia	210017	68	69	97	78	63	20/2/2002
17	Carl	210018	90	62	96	71	81	24/11/2004
18	Emma	210019	71	81	61	68	70	25/4/2002

2. Continuation of question 1: Calculate the **average** and standard deviation score for all subjects individually (Use Pandas statistical functions). The expected output will be;

```
*** average score and standard deviation ***
Average score of SE is: 78.75862068965517
Standard deviation of SE is: 11.201578882377612
Average score of Math is: 78.51724137931035
Standard deviation of Math is: 12.860850587442195
Average score of En is: 77.93103448275862
Standard deviation of En is: 12.983316312216004
Average score of Fi is: 76.48275862068965
Standard deviation of Fi is: 11.409960215453276
Average score of PE is: 79.55172413793103
Standard deviation of PE is: 12.391201391357589
```

3. The "**VibrationData.csv**" contains vibration acceleration signals in three directions (XYZ) at a certain position on the diesel engine, shown as follows:

	A	B	C	D	E	F
1	**Time**	**DirectionX**	**Time**	**DirectionY**	**Time**	**DirectionZ**
2	1.74E-05	-2.464599609	1.74E-05	6.405395508	1.74E-05	-2.416381836
3	5.64E-05	-2.174804688	5.64E-05	-13.42712402	5.64E-05	11.70776367
4	9.55E-05	-3.50402832	9.55E-05	-10.92236328	9.55E-05	-18.2668457
5	0.000134543	3.293945313	0.000134543	5.922241211	0.000134543	-7.789916992
6	0.000173606	12.06115723	0.000173606	-0.742797852	0.000173606	18.25866699
7	0.000212668	17.54418945	0.000212668	5.217407227	0.000212668	3.065185547
8	0.000251731	9.780883789	0.000251731	24.3605957	0.000251731	-15.39233398
9	0.000290793	-13.80981445	0.000290793	10.234375	0.000290793	22.29553223
10	0.000329856	-4.04296875	0.000329856	-21.82702637	0.000329856	-6.156005859
11	0.000368918	-16.89294434	0.000368918	-29.62768555	0.000368918	-18.5916748
12	0.000407981	-10.86877441	0.000407981	-12.01416016	0.000407981	8.157226563
13	0.000447043	-1.545410156	0.000447043	21.73400879	0.000447043	3.639160156
14	0.000486106	1.471435547	0.000486106	30.5291748	0.000486106	-11.37475586
15	0.000525168	1.972290039	0.000525168	-18.80310059	0.000525168	10.37463379

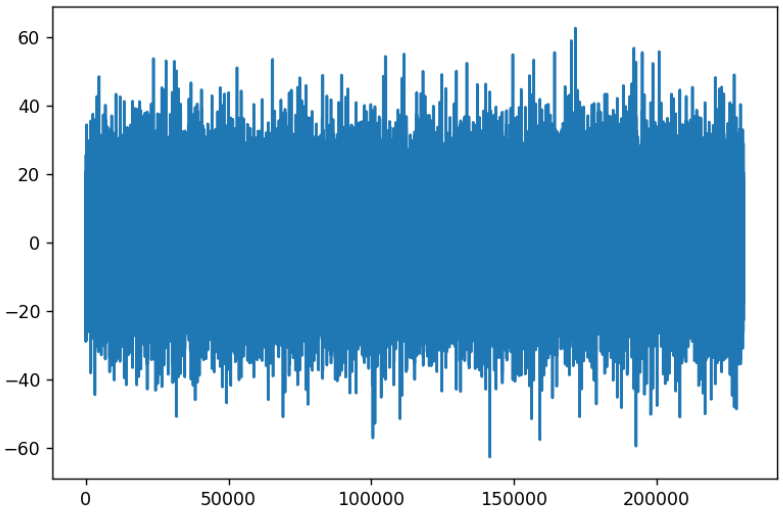
Calculate **the mean** of the absolute value and standard deviation of vibration acceleration signal in each direction. If you need to calculate the mean of the absolute value, the following statements can be used as:

```
vibsig["DirectionX"].abs().mean()
```

Here are the running results.

```
DirectionX mean: 6.976225873160231
DirectionX std: 9.204280884045419
DirectionY mean: 13.737616280014041
DirectionY std: 18.852744449999935
DirectionZ mean: 9.899565050415664
DirectionZ std: 13.278071955905812
```

You could also plot the signal to view the vibration signal, shown as follow.



Algorithms in Python

4. Define a function **RandSort(),** that generate 200 numbers in random that are between 1 and 1000. Then sort the generated numbers in descending order by using **selection sort** algorithm and return to the called program. The main program with sample run is here.

```
>>> print(RandSort())
[995, 972, 970, 962, 950, 949, 940, 938, 927, 917, 913, 912,
881, 880, 878, 877, 857, 842, 841, 832, 822, 808, 807, 795,
789, 764, 722, 659, 643, 642, 633, 620, 619, 606, 591, 553,
548, 546, 543, 536, 532, 525, 523, 501, 496, 478, 473, 463,
461, 458, 444, 426, 415, 409, 406, 401, 383, 372, 371, 353,
352, 321, 319, 308, 289, 276, 243, 242, 236, 232, 219, 202,
194, 178, 169, 168, 160, 159, 142, 136, 133, 123, 114, 111,
82, 63, 56, 54, 45, 41, 29, 21, 8]
```

5. Continuation of question 4: Define two functions, namely **LineSearch(dataList, search value)** and **BinarySearch(dataList, search value)** that search the value in the list returned by the function **RandSort()** defined in the question above (4). Then count the number of comparison times each of these functions to get the result, then print search results and return the number of comparisons times. T sample run is here:

```
>>> List1=[10,9,8,7,6,5,4,3]
    print("Compare times of BinarySearch:",
          BinarySearch(List1,3))
    print("Compare times of LineSearch:",
          LineSearch(List1,3))
Get it, in position 7
Compare times of BinarySearch: 4
Get it, in position 7
Compare times of LineSearch: 8
>>> List1=[10,9,8,7,6,5,4,3]
    print("Compare times of BinarySearch:",
          BinarySearch(List1,668))
    print("Compare times of LineSearch:",
          LineSearch(List1,668))
Can't find it.
Compare times of BinarySearch: 4
Can't find it.
Compare times of LineSearch: 8
```