Statistical learning from data: Applications in physics

Umur Can Kaya 090140107

HW1.1 - Programing languages for data science

```
from pytrends.request import TrendReq
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from time import sleep
import matplotlib
%matplotlib inline
```

```
pytrends = TrendReq(hl='en-US', tz=360)
```

I used the keyword `python for data science` instead of `python` in order to exclude the other areas than data science where python is also popular in. All of the search terms are inside the list `kw_list` and I get the google trends data of these terms as a dataframe. Head of the dataframe is below.

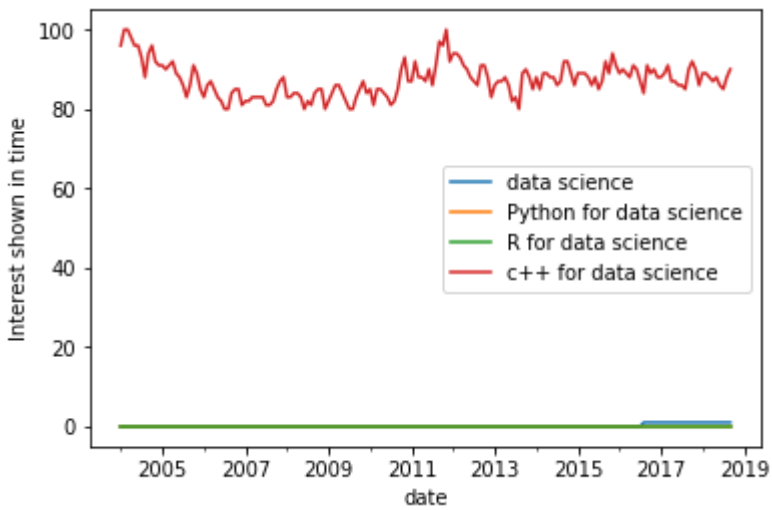
```
kw_list = ["data science", "Python for data science", "R for data science", "c++ for data science"]
pytrends.build_payload(kw_list, cat=0, timeframe='all', geo='', gprop='')
df = pytrends.interest_over_time()
df = df.drop(['isPartial'],axis=1)
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	data science	Python for data science	R for data science	c++ for data science
date				
2004-01-01	0	0	0	96
2004-02-01	0	0	0	100
2004-03-01	0	0	0	100
2004-04-01	0	0	0	98
2004-05-01	0	0	0	96

A quick look into the data:

```
df.plot()
plt.ylabel('Interest shown in time')
plt.show()
```



"The numbers show the search interest relative to the highest point in the graph for a given region and time. The value 100 is the most popular of the term. A value of 50 means that the term is as popular as half of it. A value of 0 means that there is not enough data for that term." [from google trends]

Google trends gives the interest of all search terms in the same scale and it only gives integer values. Therefore if a term is searched 100 times less than another term, then the interest of less searched term will appear as 0.

Since I am interested in the correlation between the terms, this will effect my analysis as seen in the correlation matrix below:

```
df.corr()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	data science	Python for data science	R for data science	c++ for data science
data science	1.000000	NaN	NaN	0.068096
Python for data science	NaN	NaN	NaN	NaN
R for data science	NaN	NaN	NaN	NaN
c++ for data science	0.068096	NaN	NaN	1.000000

NaN values appeard because "c++ for data science" is dominating the interest and causes the loss of information in less interested terms.

A workaround solution would be collecting the data of each column seperately and then concatenating into a final dataframe. This time with more languages.

```
kw_list = [["data science"],
           ["Python for data science"],
           ["R for data science"],
           ["Julia for data science"],
           ["MATLAB for data science"],
           ["c++ for data science"]]
```

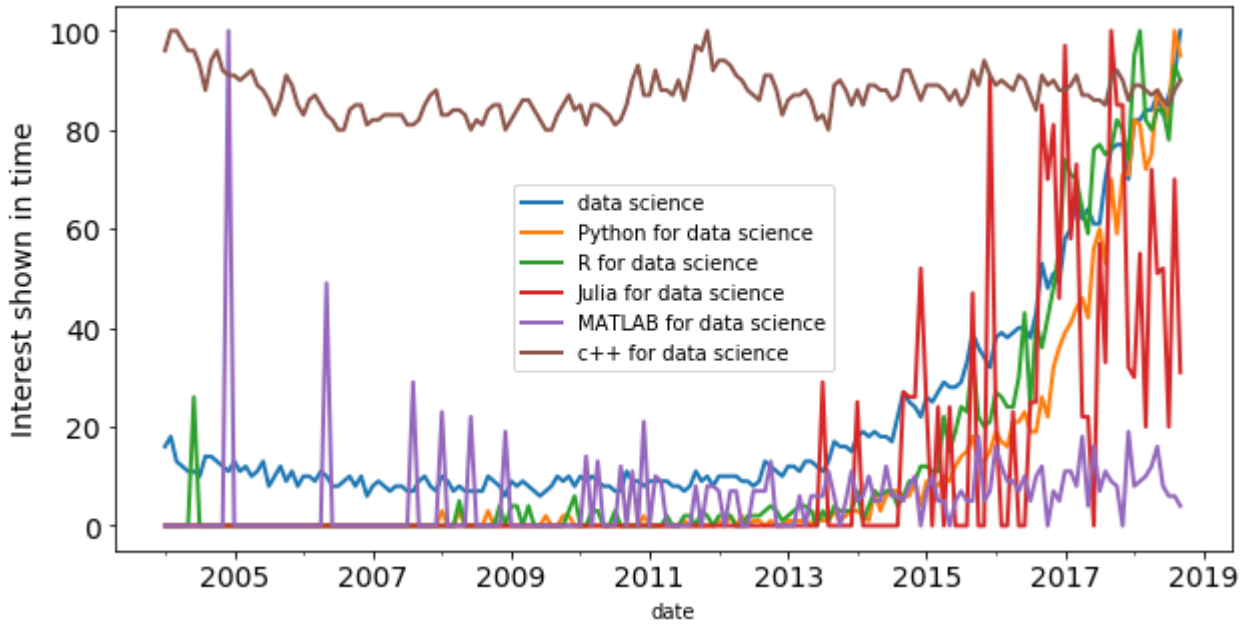
```
l = []
for kw in kw_list:
    pytrends.build_payload(kw, cat=0, timeframe='all', geo='', gprop='')
    data_frame = pytrends.interest_over_time()
    sleep(1)
    l.append(data_frame)
    del data_frame
```

```
df = pd.concat(l,axis=1)
df = df.drop(['isPartial'],axis=1)
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	data science	Python for data science	R for data science	Julia for data science	MATLAB for data science	c++ for data science
date						
2004-01-01	16	0	0	0	0	96
2004-02-01	18	0	0	0	0	100
2004-03-01	13	0	0	0	0	100
2004-04-01	12	0	0	0	0	98
2004-05-01	11	0	0	0	0	96

```
df.plot(figsize=(10,5),fontsize=14,lw=2)
plt.ylabel('Interest shown in time',fontsize=14)
plt.show()
```



This looks better.

```
df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	data science	Python for data science	R for data science	Julia for data science	MATLAB for data science	c++ for data science
count	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000
mean	22.011299	11.107345	14.666667	10.813559	4.350282	85.446328
std	22.550698	22.681460	26.593887	21.797473	8.809306	4.245144
min	6.000000	0.000000	0.000000	0.000000	0.000000	77.000000
25%	8.000000	0.000000	0.000000	0.000000	0.000000	83.000000
50%	11.000000	0.000000	2.000000	0.000000	0.000000	86.000000
75%	27.000000	7.000000	14.000000	0.000000	7.000000	88.000000
max	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000

By looking at the plot and the description of the dataframe, it seems that some of the terms deviates a lot. Since I want to see the overall correlation my analysis will effected from these deviations. Therefore it will be better if I can smooth them.

A moving average window will be enaugh.

```
df_smooth = df.rolling(window=7).mean()
```

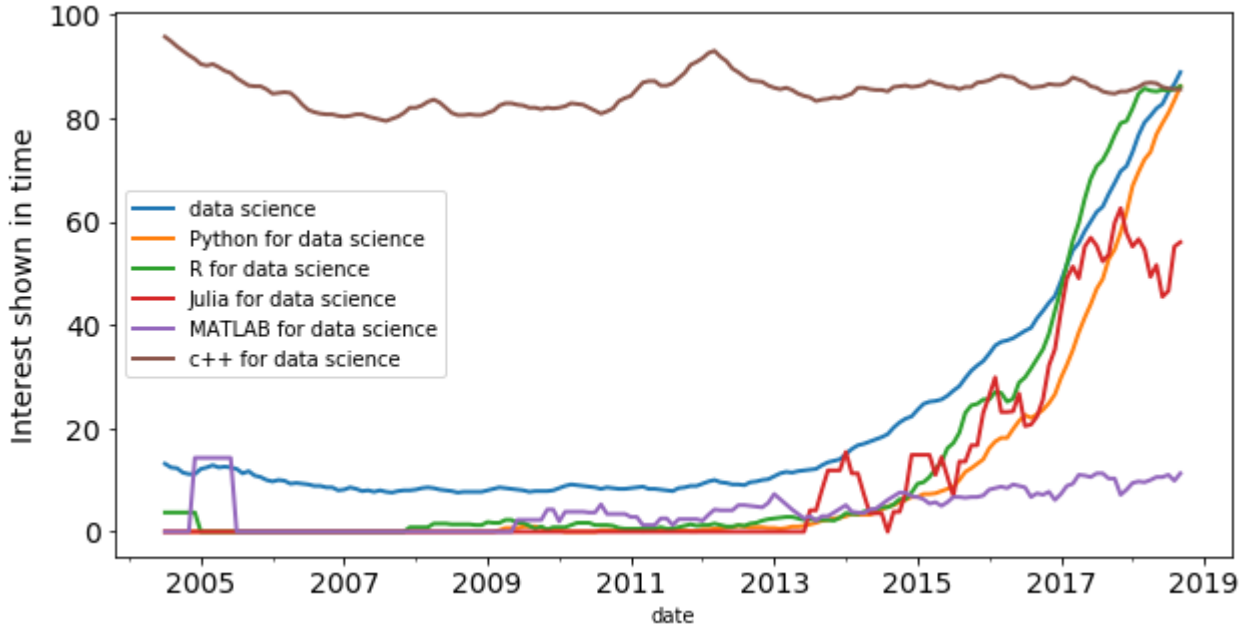
```
df_smooth.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	data science	Python for data science	R for data science	Julia for data science	MATLAB for data science	c++ for data science
count	171.000000	171.000000	171.000000	171.000000	171.000000	171.000000
mean	20.845447	9.894737	13.594820	10.137009	4.297410	85.246449
std	20.729102	20.145167	24.758976	17.879310	4.016844	3.308726
min	7.571429	0.000000	0.000000	0.000000	0.000000	79.428571
25%	8.428571	0.000000	0.571429	0.000000	0.000000	82.500000
50%	10.571429	0.571429	1.714286	0.000000	3.714286	85.571429
75%	24.928571	7.285714	10.071429	13.571429	6.857143	86.857143
max	88.857143	86.142857	86.142857	62.571429	14.285714	95.714286

After windowing standart deviations are decreased. This can also be seen in the plot:

```
df_smooth.plot(figsize=(10,5),fontsize=14,lw=2);plt.ylabel('Interest shown in time',fontsize=14);plt.show()
```



Time to see the results, let's look at the correlation matrix.

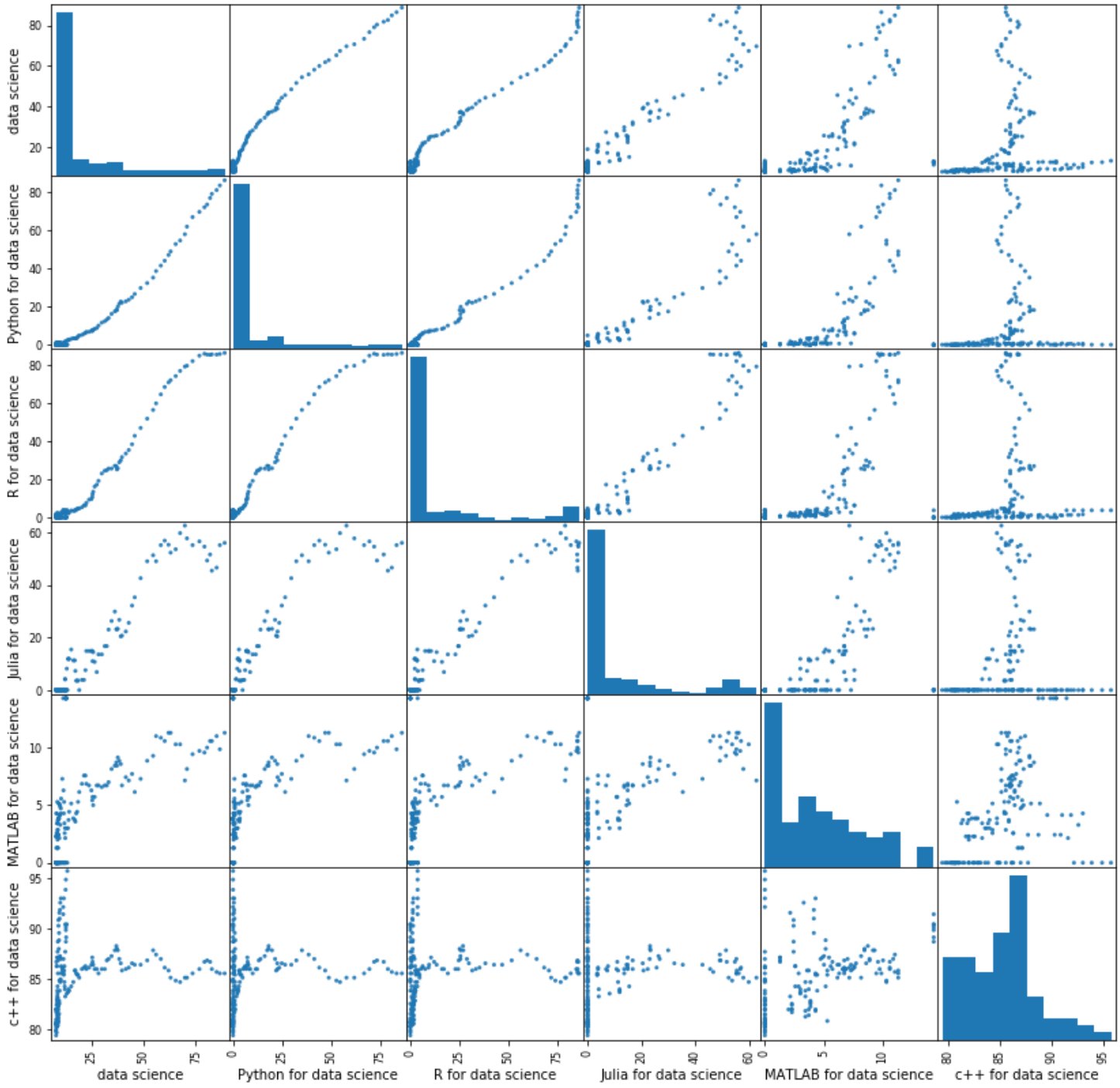
```
corr = df_smooth.corr()
corr
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	data science	Python for data science	R for data science	Julia for data science	MATLAB for data science	c++ for data science
data science	1.000000	0.981238	0.988760	0.967167	0.663327	0.196068
Python for data science	0.981238	1.000000	0.984222	0.935727	0.597969	0.129234
R for data science	0.988760	0.984222	1.000000	0.972643	0.620991	0.153380
Julia for data science	0.967167	0.935727	0.972643	1.000000	0.635454	0.153913
MATLAB for data science	0.663327	0.597969	0.620991	0.635454	1.000000	0.428204
c++ for data science	0.196068	0.129234	0.153380	0.153913	0.428204	1.000000

These values should be good enaugh but I don't feel comfortable without looking at the scatter matrix.

```
pd.plotting.scatter_matrix(df_smooth,figsize=(14,14),alpha=0.9)
plt.show()
```



Results & Discussion

```
corr["data science"][1:]
```

```
Python for data science    0.981238
R for data science         0.988760
Julia for data science     0.967167
MATLAB for data science    0.663327
c++ for data science       0.196068
Name: data science, dtype: float64
```

- As expected, Python and R are the most correlated programming languages with data science while Matlab is left far behind them.
- What suprises me is that altough its high popularity and high performance c++ is not preferred for data science.
- I am also happy to see that Julia, the language that is spesifically designed for scientific computing, is doing very well in data science, almost as good as Python and R.

The effect of windowing on the correlation values should be discussed, since I believe it effects the results a lot.

HW1.2 Complementary python libs to Pandas, Numpy and Scikit-Learn

Some python libs that can be used instead of Scikit_Learn which can basically do everything listed below.

- **Theano:** Good for neural networks and deep learning.
- **TensorFlow:** Neural networks.
- **Pylearn2:** Neural networks.
- **Pattern:** Natural language processing, clustering, and classification.

Alternative libraries to Pandas:

- **Sframe:** Can work with data that is larger than memory, with it comes a lot of machine learning libraries but it is available free for educational purposes only.
- **Dask:** Works very well with data that is larger than memory.

I couldn't find any python library alternatives to Numpy.

Refferences:

- https://stackabuse.com/the-best-machine-learning-libraries-in-python/
- https://www.linkedin.com/pulse/choosing-right-pyframework-pandassframedask-puneet-tripathi/

HW1.3

- As figure eight stated in"Data scientist report 2018", 61% of the data scientist
- I know that in particle physics and in CERN, ROOT software is used for analyse and plot data.

HW1.4 Standard deviation, variance and sample variance

```
np.random.seed(11)
```

The whole data is called population. A subset of a data is called sample of that data. When calculating variance and standard deviation, it is important to know whether we are calculating them for the whole population using all the data, or we are calculation them using only a sample of data.

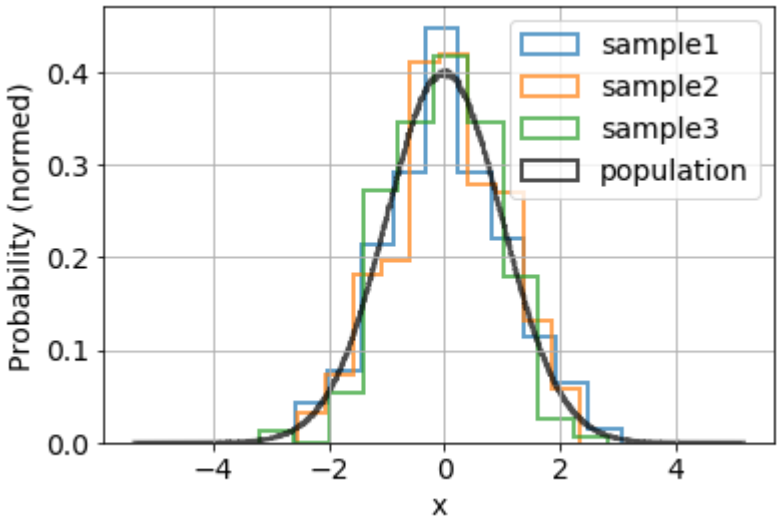
Let us create a population by drawing 1000000 points from a normal distribution with $\mu = 0$ and $\sigma = 1$. Since I draw a lot of points I belive that this data represents the real continuos normal distribution.

```
population = np.random.normal(0,1,10000000)
```

Now let us create 3 samples from the population by randomly choosing 250 points.

```
sample1 = [np.random.choice(population) for i in range(250)]
sample2 = [np.random.choice(population) for i in range(250)]
sample3 = [np.random.choice(population) for i in range(250)]
samples = [sample1, sample2, sample3]
```

```
fig, ax = plt.subplots()
ax.tick_params(axis='both', which='major', labelsize=14)

alpha, bins, lw, fs, labels = 0.7, 10, 2, 14, ['sample1', 'sample2', 'sample3']

for i, sample in enumerate(samples):
    ax.hist(sample,alpha=alpha,bins=bins,histtype='step',lw=lw,label=labels[i],density=True)

ax.hist(population,bins=1000,histtype='step',color='k',lw=lw,alpha=alpha,
label='population',density=True)

ax.set_ylabel('Probability (normed)',fontsize=fs)
ax.set_xlabel('x',fontsize=fs)

ax.grid()
plt.legend(loc='upper right',fontsize=fs)
plt.show()
```



Now we can calculate mean, var, std and sample var for all 4 data.

- **variance (population variance):** $\sigma^2 = \frac{1}{N}\sum_i (x_i - \mu)^2$
- **standart deviation:** $\sigma = \sqrt{\sigma^2}$
- **sample variance:** $s^2 = \frac{1}{N-1}\sum_i (x_i - \mu)^2$

```
mean = lambda x : sum(x)/len(x)
sigma2 = lambda x : sum((x-(sum(x)/len(x)))**2)/len(x)
s2 = lambda x : sum((x-(sum(x)/len(x)))**2)/(len(x)-1)
sigma = lambda x : np.sqrt(sum((x-(sum(x)/len(x)))**2)/(len(x)-1))
```

```
print('\t\tvar\t\tstd\t\tsample var\tmean')
print('population\t%.5f\t\t%.5f\t\t%.5f\t\t%.5f'%
(sigma2(population),sigma(population),s2(population),mean(population)))
for i, sample in enumerate(samples):
    print('sample%d\t\t%.5f\t\t%.5f\t\t%.5f\t\t%.5f'%
(i+1,sigma2(sample),sigma(sample),s2(sample),mean(sample)))
```

```
            var      std      sample var  mean
population  0.99934     0.99967     0.99934     0.00001
sample1     0.95528     0.97934     0.95911     -0.02673
sample2     1.03828     1.02100     1.04245     0.00642
sample3     0.94182     0.97242     0.94560     0.04404
```

"The calculated sample variance (and therefore also the standard deviation) will be slightly higher than variance. The purpose of this little difference it to get a better and unbiased estimate of the population's variance (by dividing by the sample size lowered by one, we compensate for the fact that we are working only with a sample rather than with the whole population)."

Refferences

- https://www.macroption.com/population-sample-variance-standard-deviation/