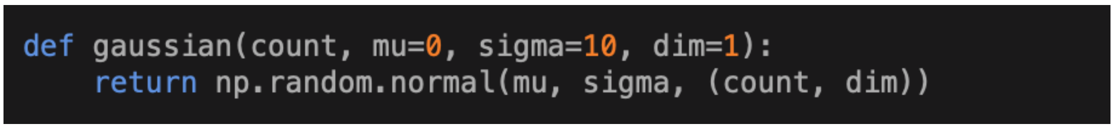
Question 3：

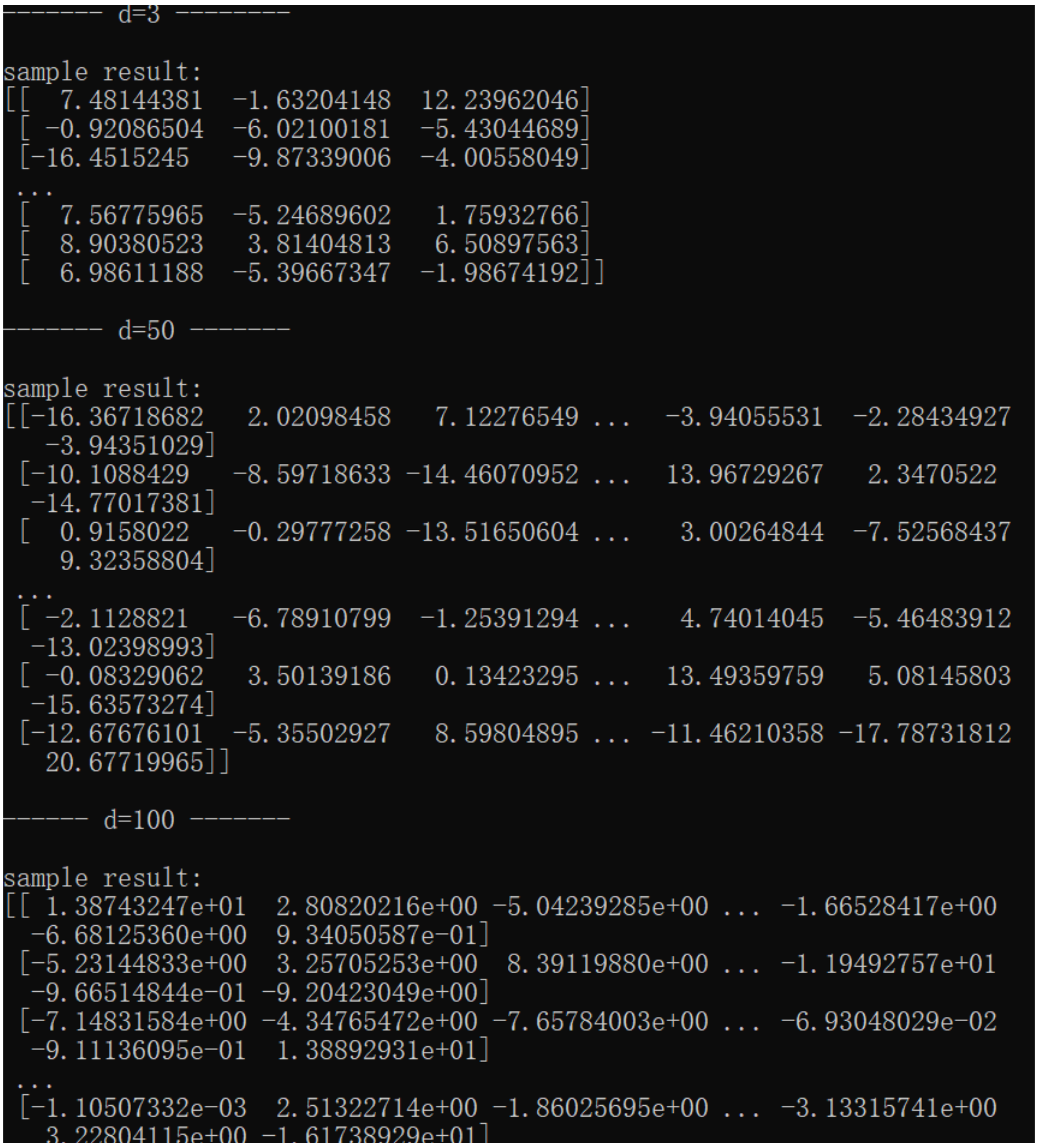
a： Fix d = 3 and generate 10,000 random samples from the standard multi-variate Gaussian distribution defined in Rd.

I set u=0, sigma=10, generate 10,000 random samples with different dimension values.



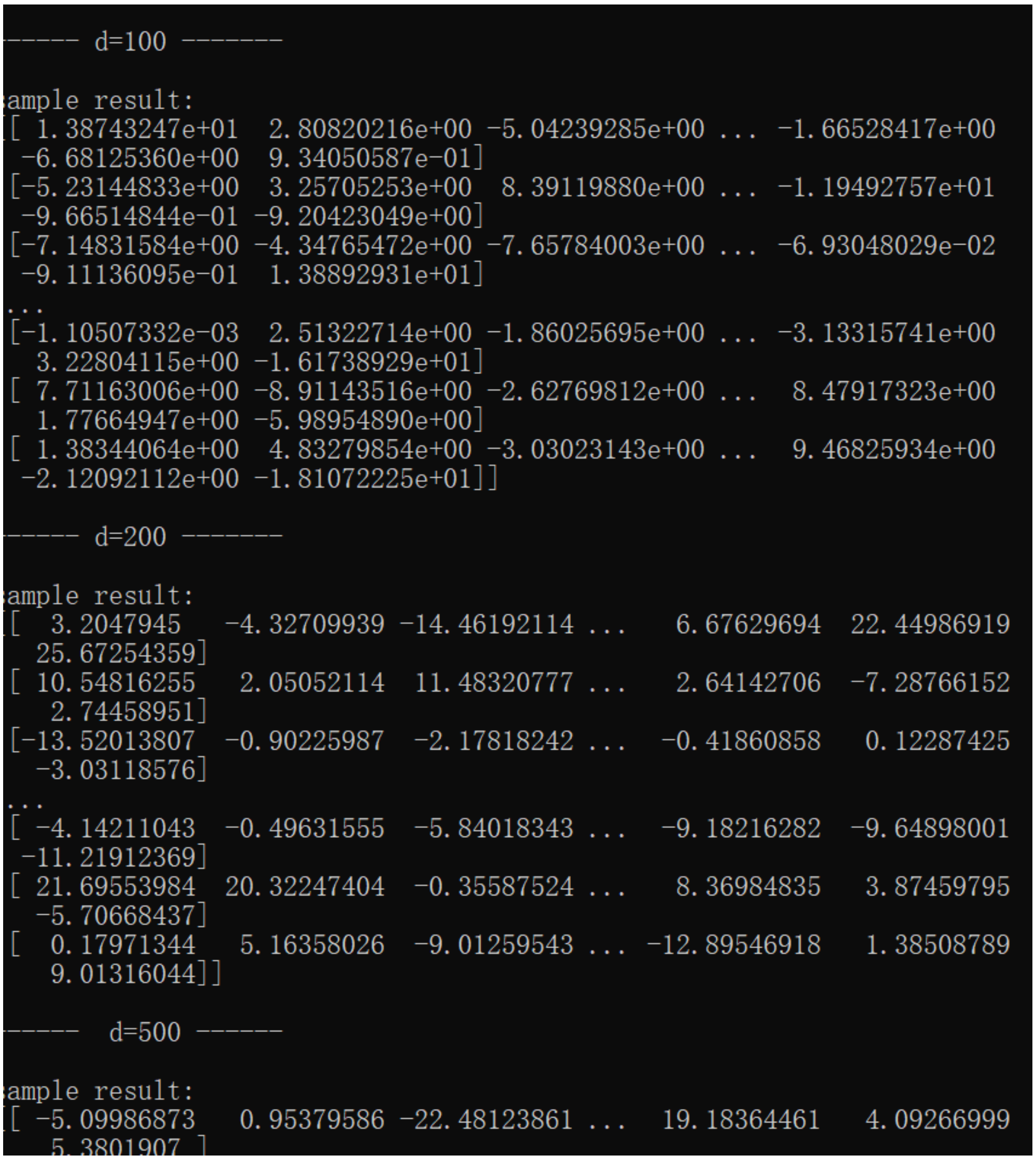
```python
def gaussian(count, mu=0, sigma=10, dim=1):
    return np.random.normal(mu, sigma, (count, dim))
```

Figure 1-1:gaussian function



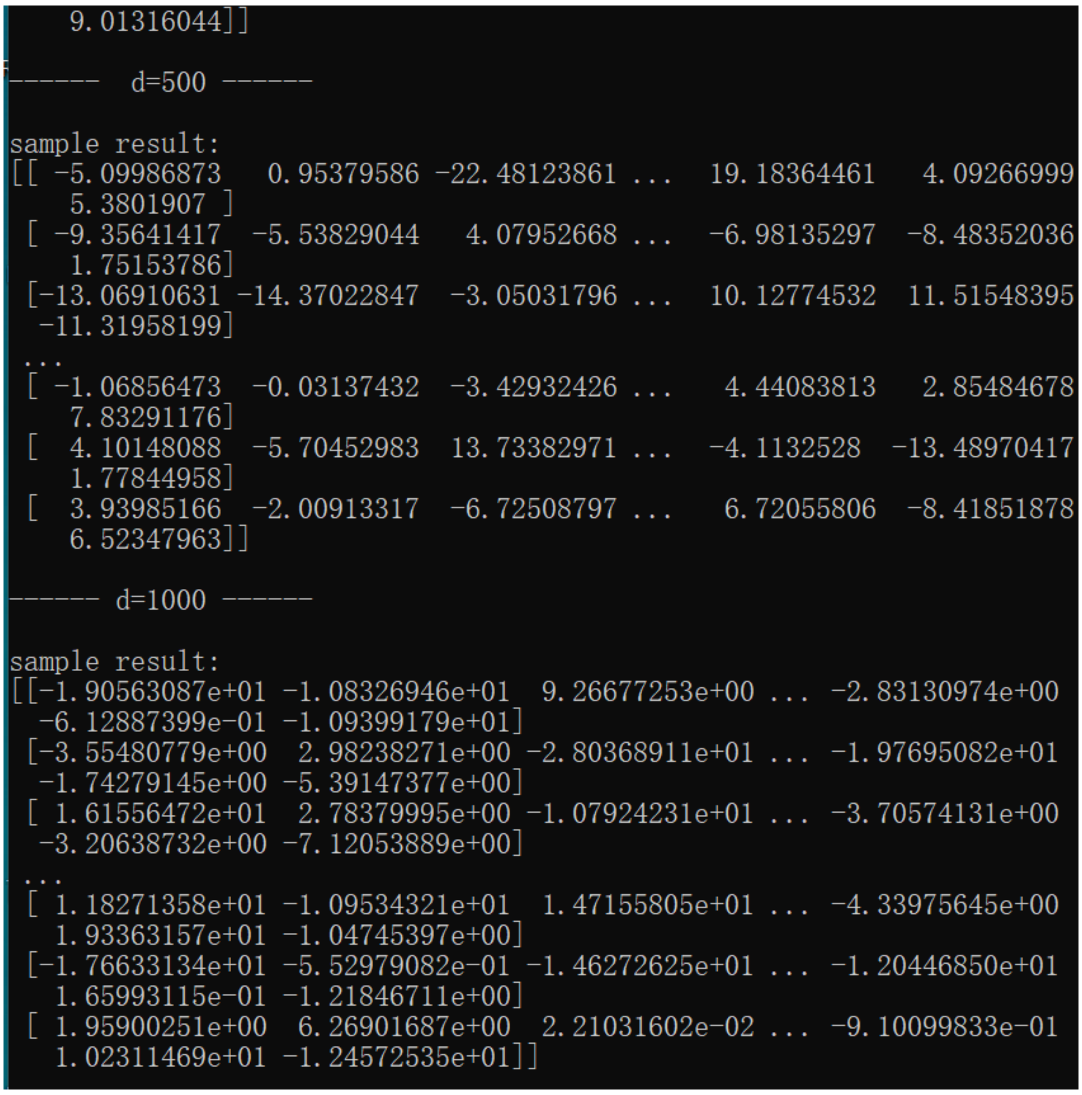
```
------- d=3 --------

sample result:
[[  7.48144381  -1.63204148  12.23962046]
 [ -0.92086504  -6.02100181  -5.43044689]
 [-16.4515245   -9.87339006  -4.00558049]
 ...
 [  7.56775965  -5.24689602   1.75932766]
 [  8.90380523   3.81404813   6.50897563]
 [  6.98611188  -5.39667347  -1.98674192]]

------- d=50 -------

sample result:
[[-16.36718682   2.02098458   7.12276549 ...  -3.94055531  -2.28434927
   -3.94351029]
 [-10.1088429   -8.59718633 -14.46070952 ...  13.96729267   2.3470522
  -14.77017381]
 [  0.9158022   -0.29777258 -13.51650604 ...   3.00264844  -7.52568437
    9.32358804]
 ...
 [ -2.1128821   -6.78910799  -1.25391294 ...   4.74014045  -5.46483912
  -13.02398993]
 [ -0.08329062   3.50139186   0.13423295 ...  13.49359759   5.08145803
  -15.63573274]
 [-12.67676101  -5.35502927   8.59804895 ... -11.46210358 -17.78731812
   20.67719965]]

------ d=100 -------

sample result:
[[ 1.38743247e+01  2.80820216e+00 -5.04239285e+00 ... -1.66528417e+00
  -6.68125360e+00  9.34050587e-01]
 [-5.23144833e+00  3.25705253e+00  8.39119880e+00 ... -1.19492757e+01
  -9.66514844e-01 -9.20423049e+00]
 [-7.14831584e+00 -4.34765472e+00 -7.65784003e+00 ... -6.93048029e-02
  -9.11136095e-01  1.38892931e+01]
 ...
 [-1.10507332e-03  2.51322714e+00 -1.86025695e+00 ... -3.13315741e+00
   3.22804115e+00 -1.61738929e+01]
```

Figure 1-2:sample result with dimension value is 3,50,100



```
----- d=100 -------

ample result:
[[ 1.38743247e+01  2.80820216e+00 -5.04239285e+00 ...  -1.66528417e+00
  -6.68125360e+00  9.34050587e-01]
 [-5.23144833e+00  3.25705253e+00  8.39119880e+00 ...  -1.19492757e+01
  -9.66514844e-01 -9.20423049e+00]
 [-7.14831584e+00 -4.34765472e+00 -7.65784003e+00 ...  -6.93048029e-02
  -9.11136095e-01  1.38892931e+01]
 ...
 [-1.10507332e-03  2.51322714e+00 -1.86025695e+00 ...  -3.13315741e+00
   3.22804115e+00 -1.61738929e+01]
 [ 7.71163006e+00 -8.91143516e+00 -2.62769812e+00 ...   8.47917323e+00
   1.77664947e+00 -5.98954890e+00]
 [ 1.38344064e+00  4.83279854e+00 -3.03023143e+00 ...   9.46825934e+00
  -2.12092112e+00 -1.81072225e+01]]

----- d=200 -------

ample result:
[[  3.2047945   -4.32709939 -14.46192114 ...   6.67629694  22.44986919
   25.67254359]
 [ 10.54816255   2.05052114  11.48320777 ...   2.64142706  -7.28766152
    2.74458951]
 [-13.52013807  -0.90225987  -2.17818242 ...  -0.41860858   0.12287425
   -3.03118576]
 ...
 [ -4.14211043  -0.49631555  -5.84018343 ...  -9.18216282  -9.64898001
  -11.21912369]
 [ 21.69553984  20.32247404  -0.35587524 ...   8.36984835   3.87459795
   -5.70668437]
 [  0.17971344   5.16358026  -9.01259543 ... -12.89546918   1.38508789
    9.01316044]]

-----  d=500 ------

ample result:
[[ -5.09986873   0.95379586 -22.48123861 ...  19.18364461   4.09266999
    5.3801907 ]
```

Figure 1-2:sample result with dimension value is 100,200,500



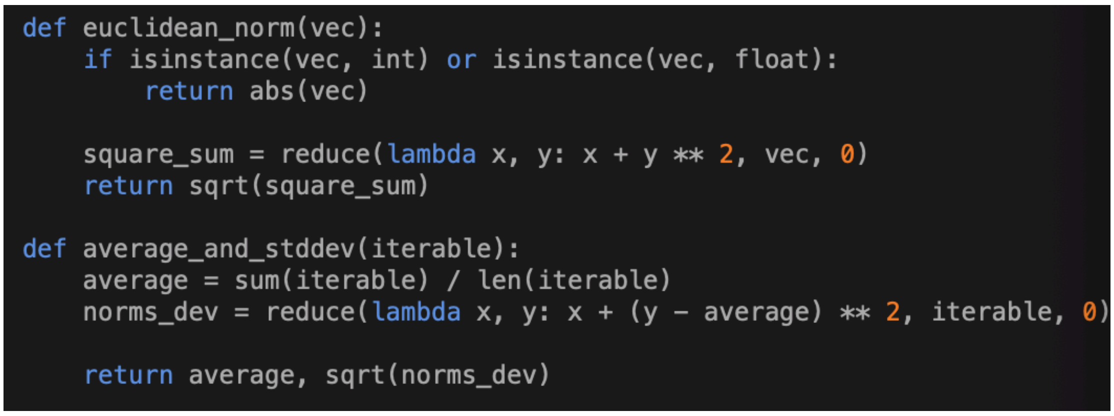
```
    9.01316044]]

------  d=500 ------

sample result:
[[ -5.09986873   0.95379586 -22.48123861 ...  19.18364461   4.09266999
    5.3801907 ]
 [ -9.35641417  -5.53829044   4.07952668 ...  -6.98135297  -8.48352036
    1.75153786]
 [-13.06910631 -14.37022847  -3.05031796 ...  10.12774532  11.51548395
  -11.31958199]
 ...
 [ -1.06856473  -0.03137432  -3.42932426 ...   4.44083813   2.85484678
    7.83291176]
 [   4.10148088  -5.70452983  13.73382971 ...  -4.1132528  -13.48970417
    1.77844958]
 [   3.93985166  -2.00913317  -6.72508797 ...   6.72055806  -8.41851878
    6.52347963]]

------ d=1000 ------

sample result:
[[-1.90563087e+01 -1.08326946e+01  9.26677253e+00 ... -2.83130974e+00
  -6.12887399e-01 -1.09399179e+01]
 [-3.55480779e+00  2.98238271e+00 -2.80368911e+01 ... -1.97695082e+01
  -1.74279145e+00 -5.39147377e+00]
 [ 1.61556472e+01  2.78379995e+00 -1.07924231e+01 ... -3.70574131e+00
  -3.20638732e+00 -7.12053889e+00]
 ...
 [ 1.18271358e+01 -1.09534321e+01  1.47155805e+01 ... -4.33975645e+00
   1.93363157e+01 -1.04745397e+00]
 [-1.76633134e+01 -5.52979082e-01 -1.46272625e+01 ... -1.20446850e+01
   1.65993115e-01 -1.21846711e+00]
 [ 1.95900251e+00  6.26901687e+00  2.21031602e-02 ... -9.10099833e-01
   1.02311469e+01 -1.24572535e+01]]
```

Figure 1-3:sample result with dimension value is 500,1000

b: Compute and plot the histogram of Euclidean norms of your samples. Also calculate the average and standard deviation of the norms.

In order to calculate the eculidean norm, need to summation the square of the each value in a vector, and then return its sqrt() value.

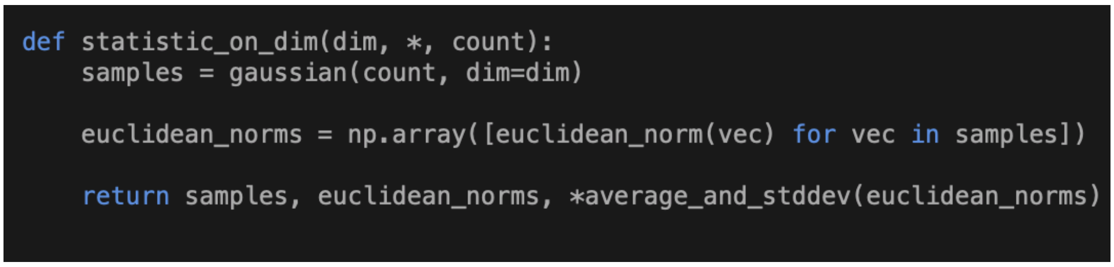
In order to calculate the average and the standard deviation of the norms, we also define a function.



```python
def euclidean_norm(vec):
    if isinstance(vec, int) or isinstance(vec, float):
        return abs(vec)

    square_sum = reduce(lambda x, y: x + y ** 2, vec, 0)
    return sqrt(square_sum)

def average_and_stddev(iterable):
    average = sum(iterable) / len(iterable)
    norms_dev = reduce(lambda x, y: x + (y - average) ** 2, iterable, 0)

    return average, sqrt(norms_dev)
```

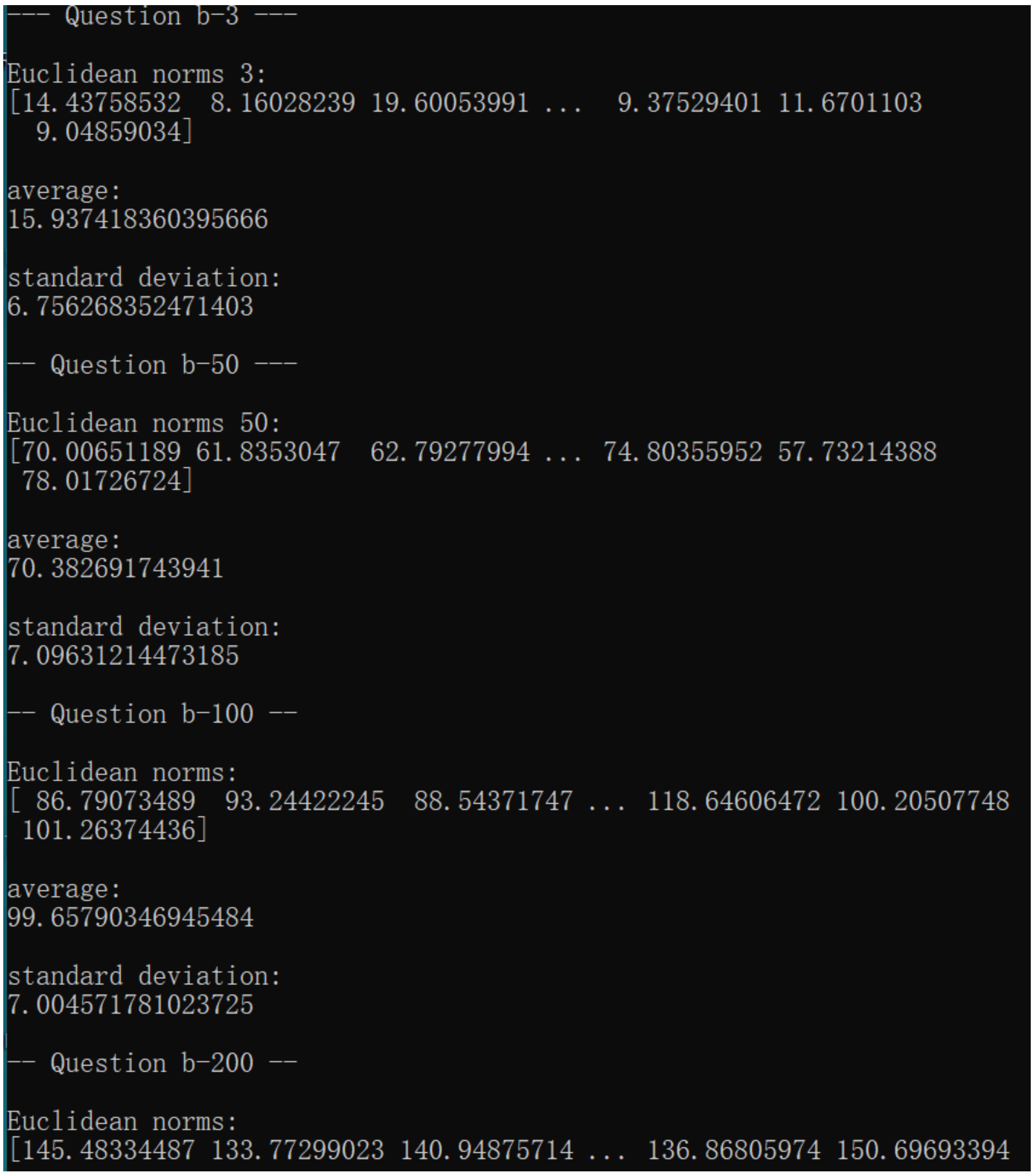
Figure 2-1: average and standard deviation functions

And then combine them in one function than can be used for sapmles, eculidean norm calculation and also includes average and standard deviation functions.



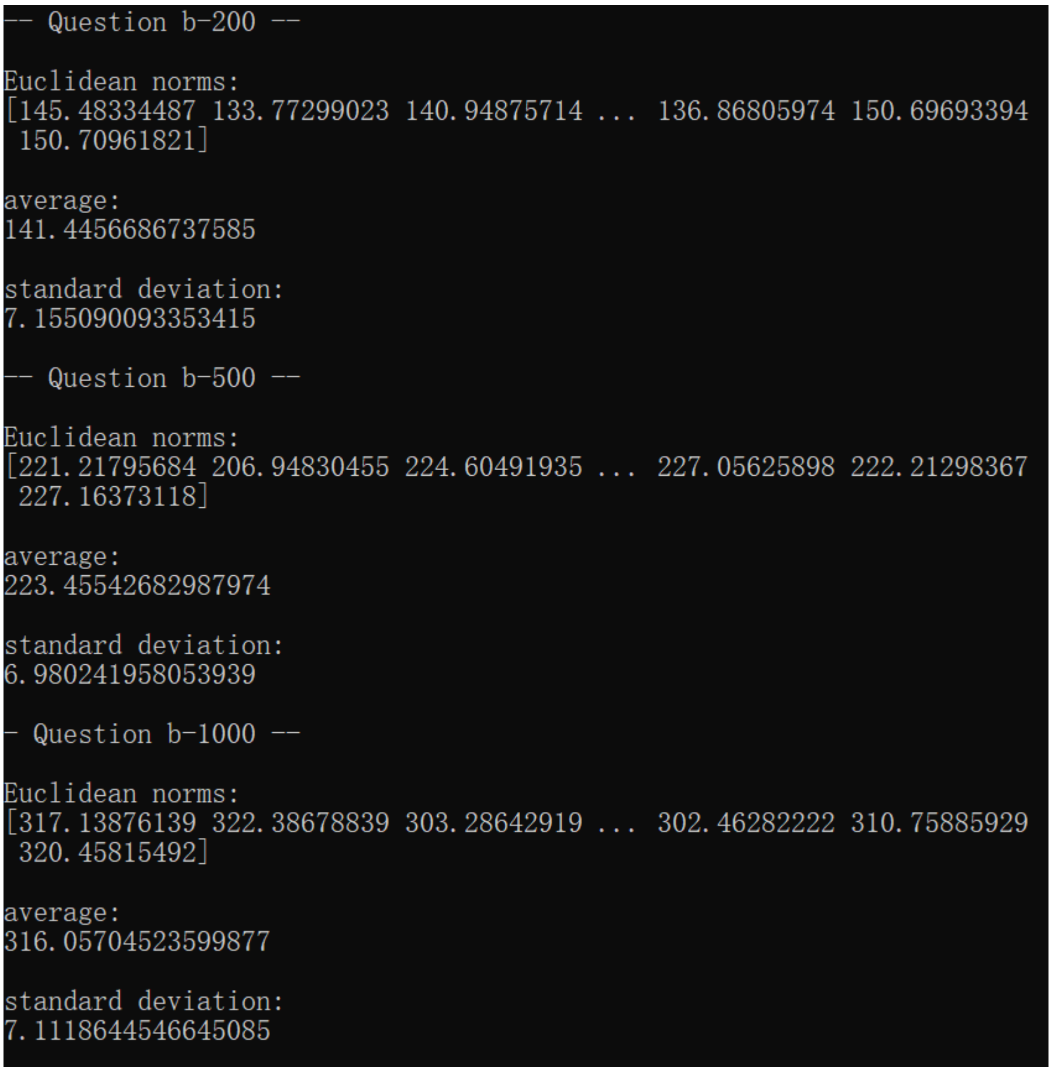
```python
def statistic_on_dim(dim, *, count):
    samples = gaussian(count, dim=dim)

    euclidean_norms = np.array([euclidean_norm(vec) for vec in samples])

    return samples, euclidean_norms, *average_and_stddev(euclidean_norms)
```

Figure 2-2: final function



```
--- Question b-3 ---

Euclidean norms 3:
[14.43758532  8.16028239 19.60053991 ...   9.37529401 11.6701103
  9.04859034]

average:
15.937418360395666

standard deviation:
6.756268352471403

-- Question b-50 ---

Euclidean norms 50:
[70.00651189 61.8353047  62.79277994 ... 74.80355952 57.73214388
 78.01726724]

average:
70.382691743941

standard deviation:
7.09631214473185

-- Question b-100 --

Euclidean norms:
[ 86.79073489  93.24422245  88.54371747 ... 118.64606472 100.20507748
 101.26374436]

average:
99.65790346945484

standard deviation:
7.004571781023725

-- Question b-200 --

Euclidean norms:
[145.48334487 133.77299023 140.94875714 ... 136.86805974 150.69693394
```

Figure 2-3: average and standard deviation of the norms with dimension is 3.50,100,200



```
-- Question b-200 --

Euclidean norms:
[145.48334487 133.77299023 140.94875714 ...  136.86805974 150.69693394
 150.70961821]

average:
141.4456686737585

standard deviation:
7.155090093353415

-- Question b-500 --

Euclidean norms:
[221.21795684 206.94830455 224.60491935 ...  227.05625898 222.21298367
 227.16373118]

average:
223.45542682987974

standard deviation:
6.980241958053939

- Question b-1000 --

Euclidean norms:
[317.13876139 322.38678839 303.28642919 ...  302.46282222 310.75885929
 320.45815492]

average:
316.05704523599877

standard deviation:
7.1118644546645085
```

Figure 2-4: average and standard deviation of the norms with dimension is 200,500,1000

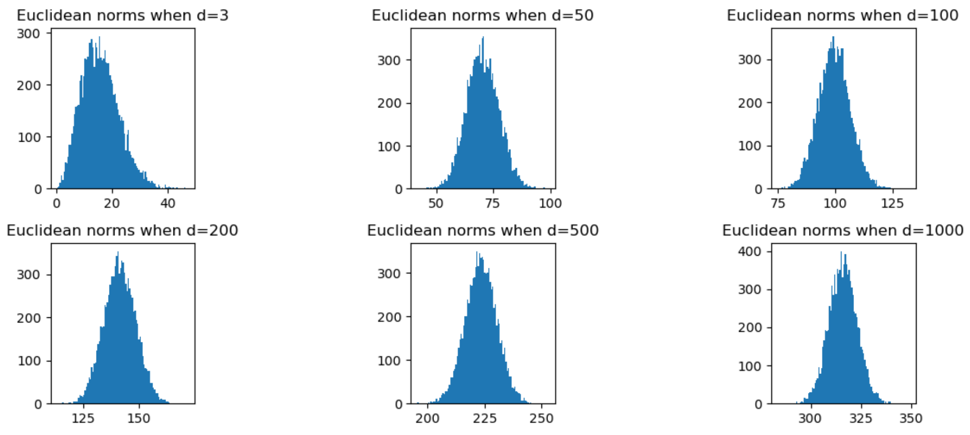


Figure 2-5: histogram of Euclidean norms

C: Increase d on a coarsely spaced log scale all the way up to d = 1000 (say d = 50, 100, 200, 500, 1000), and repeat parts (a) and (b). Plot the variation of the average and the standard deviation of Euclidean norm of the samples with increasing d.

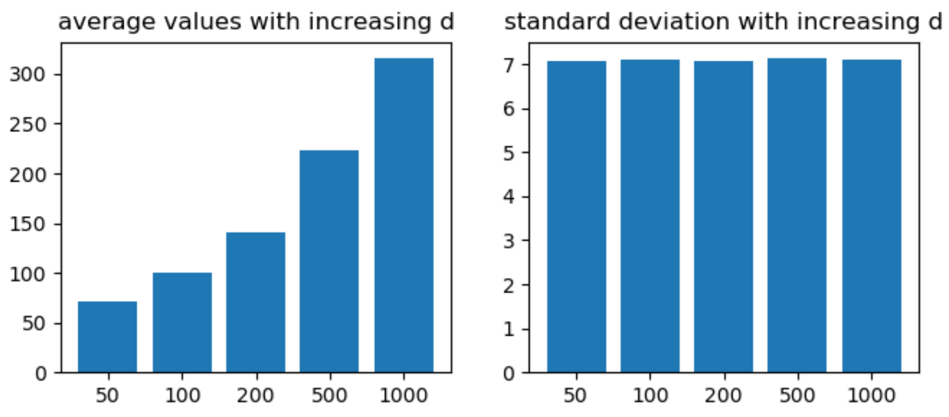


Figure 3-1: average and standard deviation with increasing d

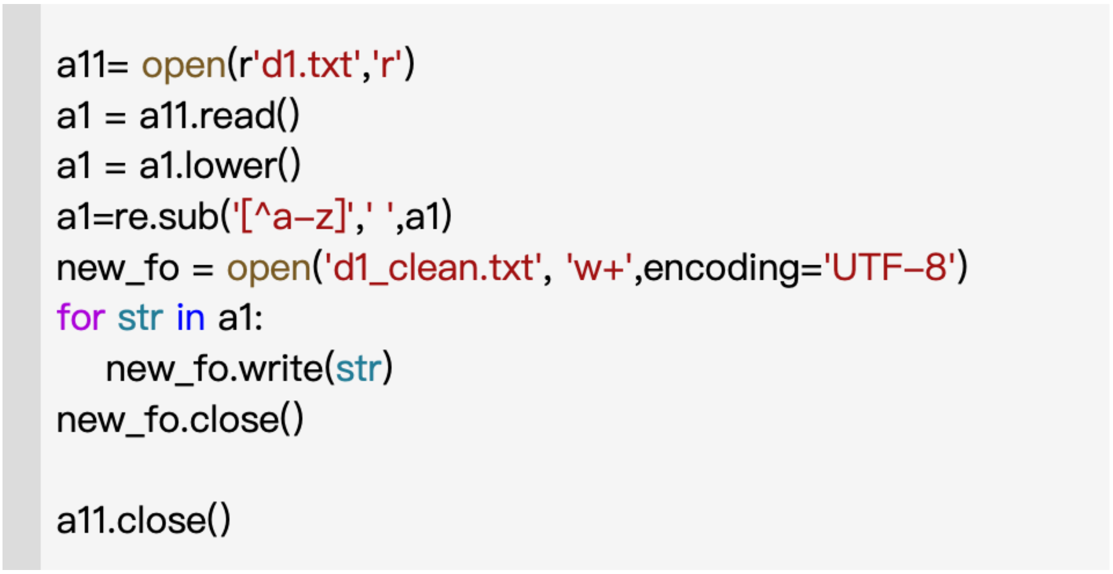
D:What can you conclude from your plot from part (c)?

The average value are increased with the dimensions value increasing, however, the standard deviation nearly keeps unchanged.

Question 4 ：

a:Write a small parser to read each document and convert it into a vector of words.

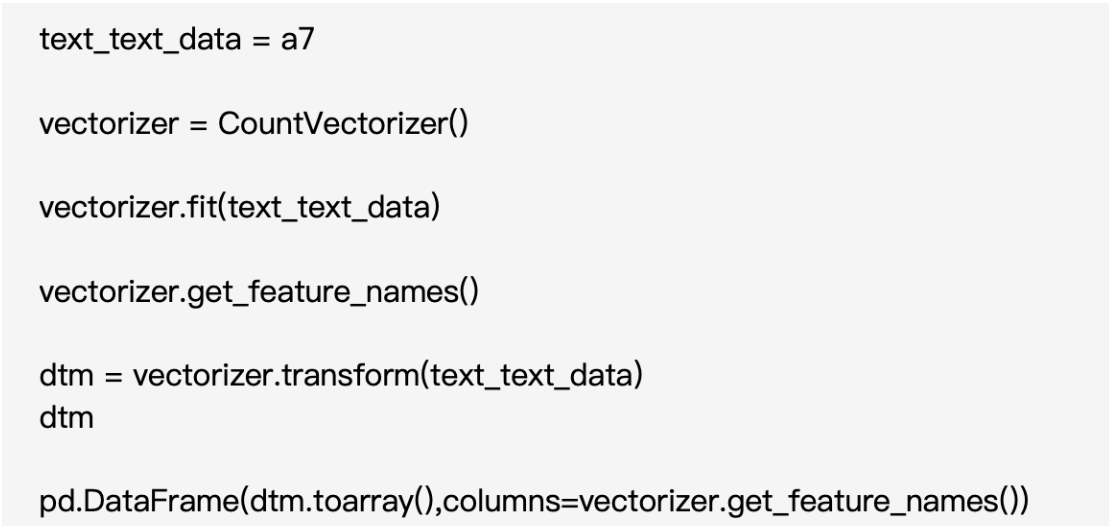
First, I do some text preprocessing work. Such as remove the number, comma and transfer the words to lower case. And then the clean data are save is the new txt file named di_clean.txt separately. After this operation, I got 542 different words in total 6 files.



```
a11= open(r'd1.txt','r')
a1 = a11.read()
a1 = a1.lower()
a1=re.sub('[^a-z]',' ',a1)
new_fo = open('d1_clean.txt', 'w+',encoding='UTF-8')
for str in a1:
    new_fo.write(str)
new_fo.close()

a11.close()
```

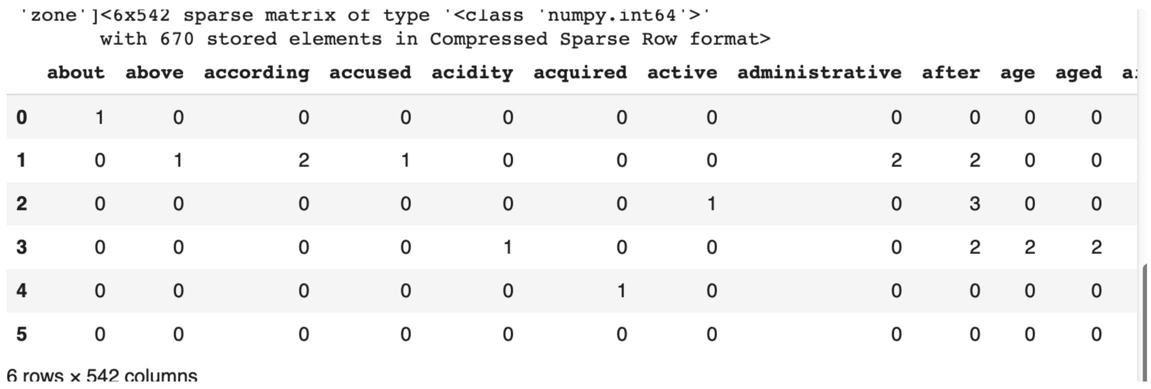
Figure 4-1: text preprocessing code

Then use sklearn library and function CountVectorizer to get the result and transfer them to array.



```
text_text_data = a7

vectorizer = CountVectorizer()

vectorizer.fit(text_text_data)

vectorizer.get_feature_names()

dtm = vectorizer.transform(text_text_data)
dtm

pd.DataFrame(dtm.toarray(),columns=vectorizer.get_feature_names())
```

Figure 4-2: vectorizer code



```
'zone']<6x542 sparse matrix of type '<class 'numpy.int64'>'
	with 670 stored elements in Compressed Sparse Row format>
```

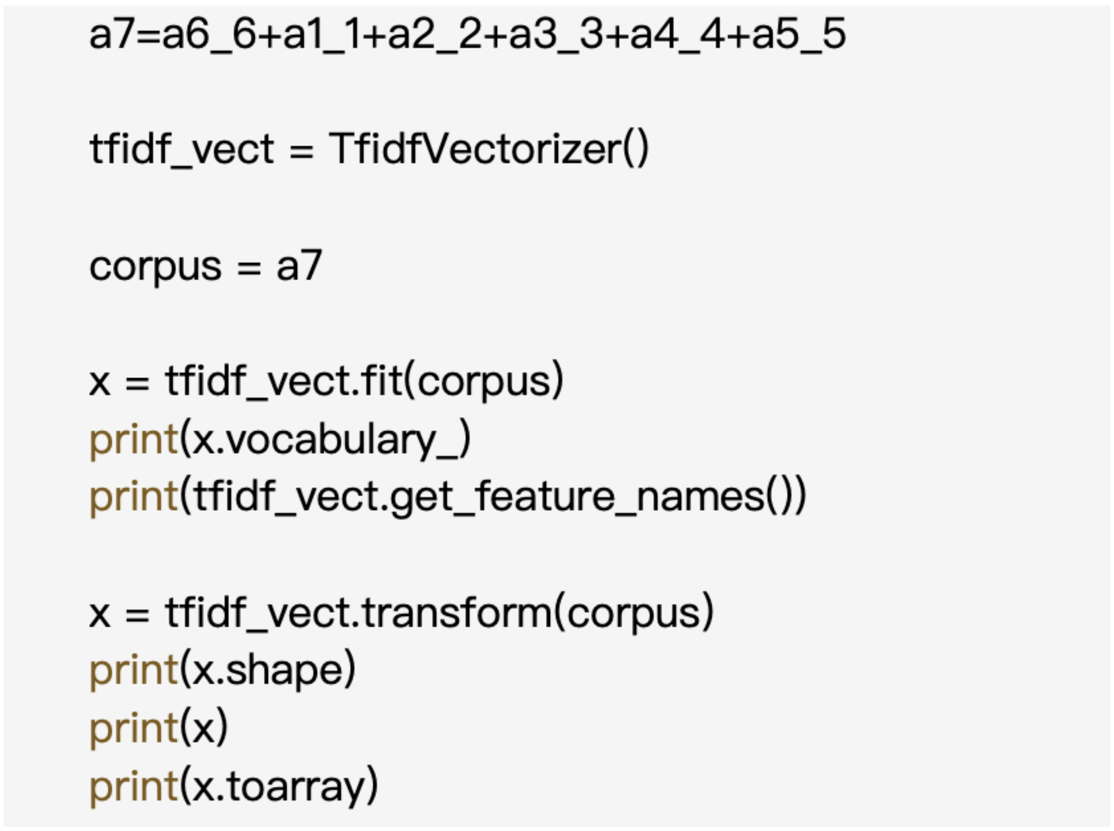
	about	above	according	accused	acidity	acquired	active	administrative	after	age	aged
0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	2	1	0	0	0	2	2	0	0
2	0	0	0	0	0	0	1	0	3	0	0
3	0	0	0	0	1	0	0	0	2	2	2
4	0	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0

6 rows × 542 columns

Figure 4-3: realut: word vectors

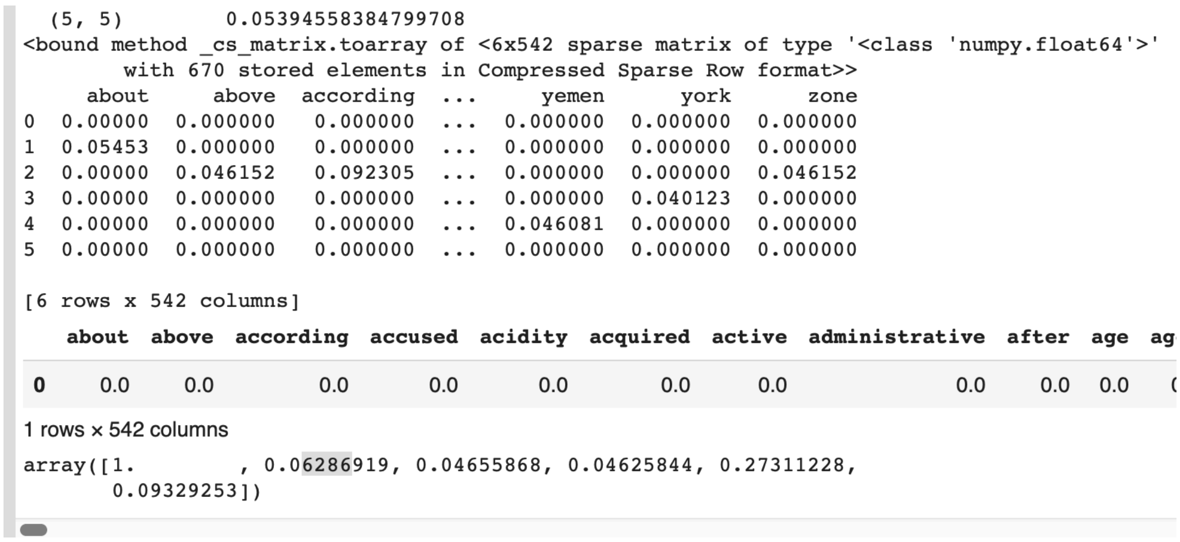
b. Compute tf-idf values for each word in every document as well as the query.

Then use sklearn library and function TfidfVectorizer() to get the tifidf result. a7 is the corpus of the whole text.



```
a7=a6_6+a1_1+a2_2+a3_3+a4_4+a5_5

tfidf_vect = TfidfVectorizer()

corpus = a7

x = tfidf_vect.fit(corpus)
print(x.vocabulary_)
print(tfidf_vect.get_feature_names())

x = tfidf_vect.transform(corpus)
print(x.shape)
print(x)
print(x.toarray)
```

Figure 5-1: tf-idf code



```
  (5, 5)        0.05394558384799708
<bound method _cs_matrix.toarray of <6x542 sparse matrix of type '<class 'numpy.float64'>'
        with 670 stored elements in Compressed Sparse Row format>>
     about     above  according  ...     yemen      york      zone
0  0.00000  0.000000   0.000000  ...  0.000000  0.000000  0.000000
1  0.05453  0.000000   0.000000  ...  0.000000  0.000000  0.000000
2  0.00000  0.046152   0.092305  ...  0.000000  0.000000  0.046152
3  0.00000  0.000000   0.000000  ...  0.000000  0.040123  0.000000
4  0.00000  0.000000   0.000000  ...  0.046081  0.000000  0.000000
5  0.00000  0.000000   0.000000  ...  0.000000  0.000000  0.000000

[6 rows x 542 columns]
```

	about	above	according	accused	acidity	acquired	active	administrative	after	age	ag
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	(

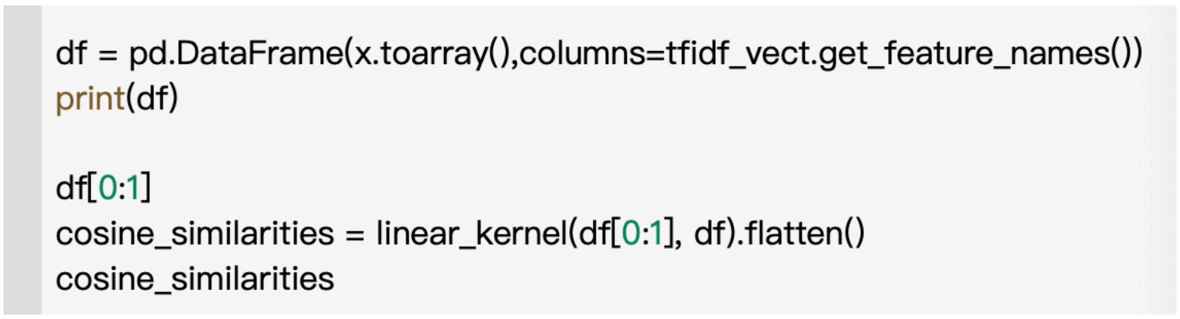
1 rows × 542 columns

```
array([1.        , 0.06286919, 0.04655868, 0.04625844, 0.27311228,
       0.09329253])
```

Figure 5-2: tf-idf value

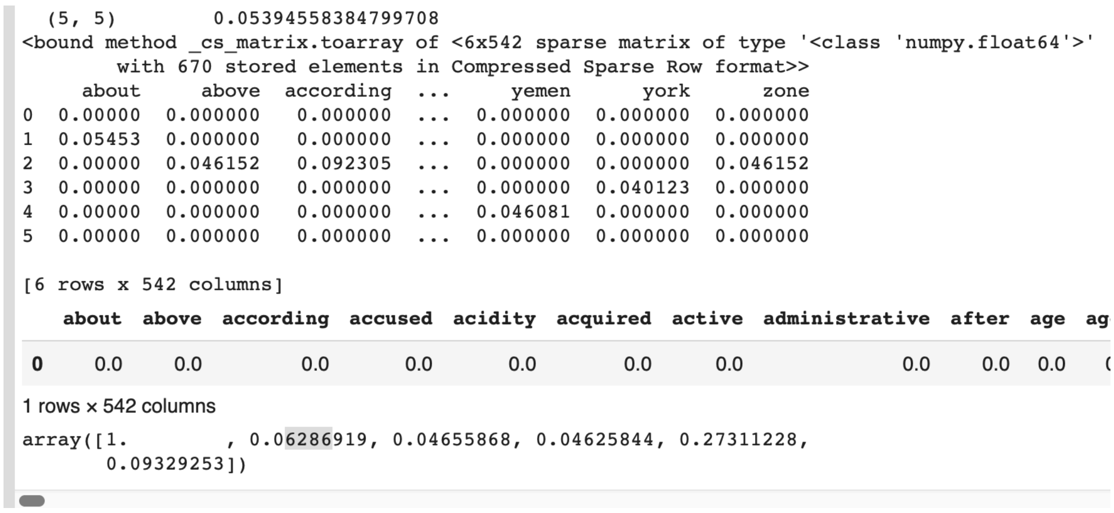
c. Compute the cosine similarity between tf-idf vectors of each document and the query.

Then use the function linear_kernel to calculate the cosine similarity.



```
df = pd.DataFrame(x.toarray(),columns=tfidf_vect.get_feature_names())
print(df)

df[0:1]
cosine_similarities = linear_kernel(df[0:1], df).flatten()
cosine_similarities
```

Figure 6-1: cosine similarity code



```
  (5, 5)        0.05394558384799708
<bound method _cs_matrix.toarray of <6x542 sparse matrix of type '<class 'numpy.float64'>'
        with 670 stored elements in Compressed Sparse Row format>>
     about     above  according  ...     yemen      york      zone
0  0.00000  0.000000   0.000000  ...  0.000000  0.000000  0.000000
1  0.05453  0.000000   0.000000  ...  0.000000  0.000000  0.000000
2  0.00000  0.046152   0.092305  ...  0.000000  0.000000  0.046152
3  0.00000  0.000000   0.000000  ...  0.000000  0.040123  0.000000
4  0.00000  0.000000   0.000000  ...  0.046081  0.000000  0.000000
5  0.00000  0.000000   0.000000  ...  0.000000  0.000000  0.000000

[6 rows x 542 columns]
```

	about	above	according	accused	acidity	acquired	active	administrative	after	age	ag
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

1 rows × 542 columns

```
array([1.        , 0.06286919, 0.04655868, 0.04625844, 0.27311228,
       0.09329253])
```

Figure 6-2: cosine similarity value

d. Report the document with the maximum similarity value.

As the result shown, d4.txt has the maximum similarity which is 0.273, is much higher than others.