I. Principal Component Analysis

1. Data structure

The data structure for PCA is a 6000 x 3 matrix with each row storing a data point's 3d coordinate. This dataset will yield a 3 x 3 covariance matrix from which we can derive three eigenvalues and three eigenvectors. Among three eigenvectors, we select the two largest ones as the direction of the dataset's principal components. Finally, we will get a 6000 x 2 matrix that stores the inner product of all data points on two principal components; now, each row in the matrix represents a data point's 2d coordinate.

2. Programming

*Step one: import the package and read the dataset in the form of dataframe.

*Step two: calculate dataset's covariance matrix with function ***np.cov***, yield:

```
array([[ 81.24199811, -15.84081415,  31.66840483],
       [-15.84081415,  13.70181418, -15.26445036],
       [ 31.66840483, -15.26445036,  31.36677137]])
```

*Step three: use ***np.linalg.eig ()*** to get eigenvalues and eigenvectors:

```
In [11]: eig_values
Out[11]: array([101.61980038,  19.89921519,   4.79156808])

In [12]: eig_vectors
Out[12]: array([[ 0.86667137, -0.4962773 , -0.0508879 ],
                [-0.23276482, -0.4924792 ,  0.83862076],
                [ 0.44124968,  0.71496368,  0.54233352]])
```

*Step four: select the two largest eigenvectors as the directions of principal components

```
[[ 0.86667137 -0.4962773 ]
 [-0.23276482 -0.4924792 ]
 [ 0.44124968  0.71496368]]
```

and derive the inner product:

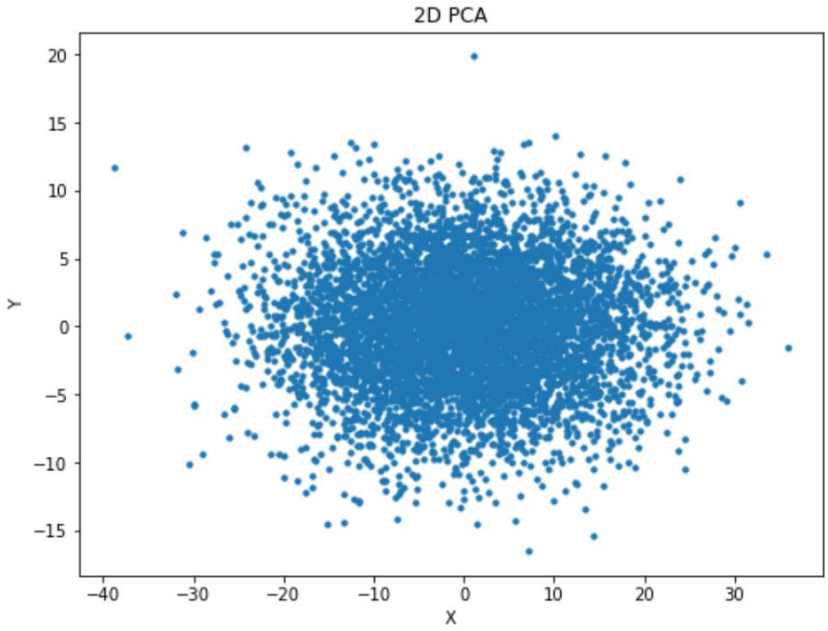
```
In [41]: data_2d = dataset.dot(z) #Calculate dataset's inner product on eigenvectors z

         print(data_2d)

                 0         1
0       10.953140  7.413760
1      -12.609630 -4.208993
2        0.509021  0.306807
3       -6.115972 -0.330041
4       13.559447 -1.615352
...           ...       ...
5995    -9.734402 -6.154735
5996     7.756264 -2.239297
5997    -2.846070  2.458947
5998    11.259641  4.243291
5999    14.306372  5.683894

[6000 rows x 2 columns]
```

*Step five: plot the result of PCA:



II. FastMap

1. Data structure

To ease the difficulty of running the fastmap algorithm, I restructure the raw data points into a 10 x 10 data frame:

	1	2	3	4	5	6	7	8	9	10
1	0.0	4.0	7.0	6.0	7.0	7.0	4.0	6.0	6.0	10.0
2	4.0	0.0	7.0	7.0	8.0	9.0	2.0	8.0	8.0	11.0
3	7.0	7.0	0.0	5.0	6.0	10.0	6.0	6.0	6.0	12.0
4	6.0	7.0	5.0	0.0	2.0	10.0	7.0	4.0	5.0	12.0
5	7.0	8.0	6.0	2.0	0.0	10.0	8.0	5.0	4.0	11.0
6	7.0	9.0	10.0	10.0	10.0	0.0	9.0	10.0	9.0	4.0
7	4.0	2.0	6.0	7.0	8.0	9.0	0.0	8.0	8.0	11.0
8	6.0	8.0	6.0	4.0	5.0	10.0	8.0	0.0	2.0	12.0
9	6.0	8.0	6.0	5.0	4.0	9.0	8.0	2.0	0.0	11.0
10	10.0	11.0	12.0	12.0	11.0	4.0	11.0	12.0	11.0	0.0

In this data frame, the i^{th}, j^{th} element exactly represents the distance between object i and object j. With the help of restructured data, we can extract the distance between any two points more easily. In particular, it saves the extra computation on the selection of pivot object Oa, Ob.

2. Programming:

*Step one: import and packages and read in the raw data.

*Step two: re-structure the raw data.

*Step three: define the function ***fastmap***. Its operation sequence is:

(1) Input the dataset and the degree (k) of our target dimension

(2) build two empty arrays: ***pivot_array*** and ***distance_matrix*** to store the pivot objects and the distances among data points after the completion of the fastmap.

(3) define a loop that runs k times to:

a. select the pivot object Oa, Ob

b. project the data points to line (Oa,Ob), calculate the $x_i = \frac{d_{a,i}^2 + d_{a,b}^2 - d_{b,i}^2}{2d_{a,b}}$.

c. project the data points to a hyper plane perpendicular to line (Oa, Ob), calculate the distance (O_i',O_j') $= \left(D(O_i, O_j)\right)^2 - (x_i - x_j)^2$.

d. return the result: pivot objects and distance matrix.

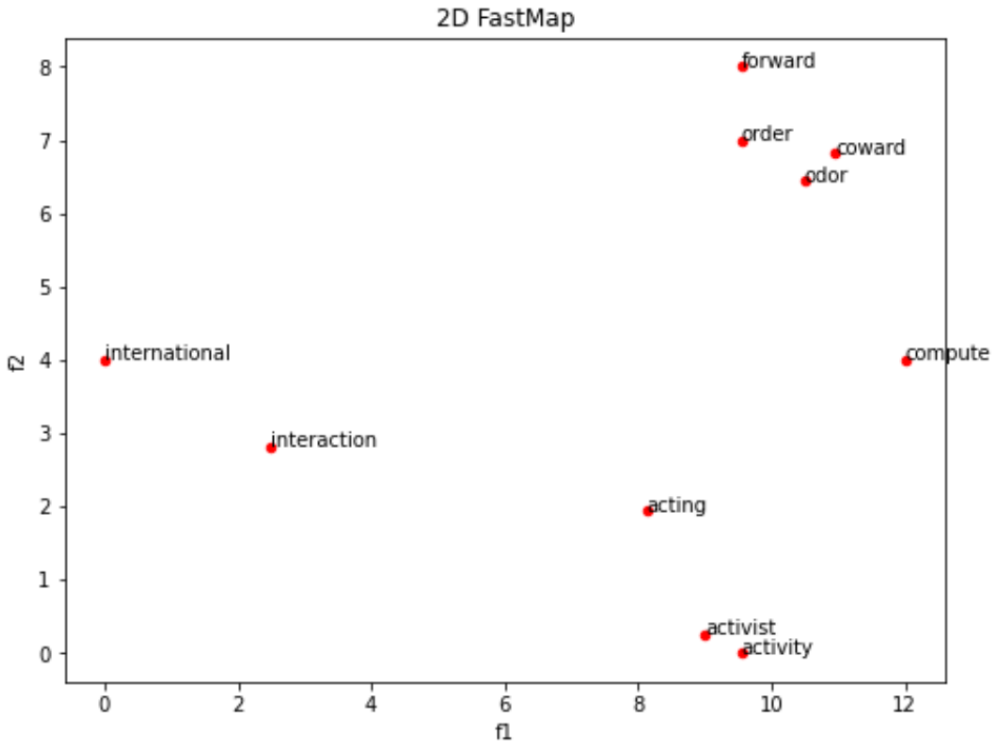
*Step four: implement the ***fastmap*** function, project our dataset to a 2-d space.

*Step five: print out the pivot objects and the distance matrix with names of word labeled:

```
array([[10.,  3.],
       [ 7.,  5.]])
```

	f1	f2
acting	8.125000	1.9375
activist	9.000000	0.2500
compute	12.000000	4.0000
coward	10.958333	6.8125
forward	9.541667	8.0000
interaction	2.500000	2.8125
activity	9.541667	0.0000
odor	10.500000	6.4375
order	9.541667	7.0000
international	0.000000	4.0000

*Step six: plot the coordinate (f1, f2) on a 2-d space:



III. Challenge and Code-Level Optimization

In this programming work, I go through two challenges. The first one is the understanding of the theories behind the fastmap algorithm. The second is how to organize a data structure for running fastmap more smoothly.

Generally speaking, the fastmap algorithm is no more than a product of basic geometry. For me, however, its application to high dimension space makes the basic theory a bit perplexed. In particular, the process of projecting objects to a new dimension space always has demands on my abstract imagination. Once I lost track of the abstract space concept, as I did during the initial coding, the confusion of data points' positions in a certain dimension space will come and deny nearly all programming work I'd done.

Luckily, I can overcome the confusion of fastmap's basic theory by studying it repeatedly. What drains me more in this homework is the second challenge—the reorganization of the raw dataset.

Like I mentioned above, I restructure the dataset for fastmap analysis to a 10 x 10 matrix. The reason for doing this is that the original data structure has two glitches. For one thing, the original dataset does not record the distance zero between the same point; for the other, the data point '10' only appears in the second column of the dataset. Combined together, these two glitches require additional computation orders, for example, an extra condition statement to tell the computer to return '0' when it selects the same data point, or to read the second column anytime the code iterates through the data point '10'.

Otherwise, with the new data structure, I assign the i^{th}, j^{th} element in it to be exactly the distance between point i and j and save all the above extra computations. Therefore, to overcome the challenge of data structure is also my primary code optimization in this homework: to avoid the computation process being too "heavy" and to simplify the steps we need to extract the data points.