Name: Kushal Sharma
Email: kushals@usc.edu
Course: INF 552 Summer 2020
Assignment: HomeWork3 (PCA and FastMap implementation)

PCA (Principal Component Analysis):

From the given 3d points, following steps were taken to come up with two principal components in PCA.

i. Mean is calculated for 3 dimensions.
ii. Covariance matrix is calculated for given 3d points.
iii. Using numpy.linalg.eigen(covariance_matrix) library function, eigenvalues and corresponding eigenvectos are determined.
iv. Using argsort() library, indices of k=2 maximum eigenvalues are determined.
v. Eigenvectors of indices corresponding to k=2 maximum eigenvalue is determined thus first two principal components are determined.
vi. New points in 2d are determined by taking dot product of 3d points and truncated matrix containing first two principal components.
vii. Covariance Matrix, Eigenvalue and Eigenvector are calculated for 2d points in new space.

Below are the related output:

Covariance Matrix(3d points):
```
[[ 81.22845778 -15.83817402  31.66312677]
 [-15.83817402  13.69953054 -15.26190629]
 [ 31.66312677 -15.26190629  31.36154358]]
```

Eigenvector(3d points):
```
[[ 0.86667137 -0.4962773  -0.0508879 ]
 [-0.23276482 -0.4924792   0.83862076]
 [ 0.44124968  0.71496368  0.54233352]]
```

Eigenvalue(3d points):
```
[101.60286375  19.89589866   4.79076949]
```

Eigenvalue_truncated:
```
[101.60286375  19.89589866]
```

Eigenvector_truncated/ direction of first two components in PCA:
```
[[ 0.86667137 -0.4962773 ]
 [-0.23276482 -0.4924792 ]
 [ 0.44124968  0.71496368]]
```

```
EigenVector 1: [ 0.86667137 -0.23276482  0.44124968]
EigenVector 2: [-0.4962773  -0.4924792   0.71496368]
```

Covariance Matrix(2d points):
```
[[1.01602864e+02 5.19169892e-15]
 [5.19169892e-15 1.98958987e+01]]
```

Eigenvector(2d points):

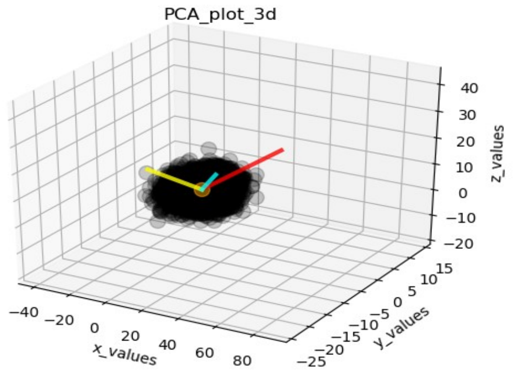
```
[[ 1.00000000e+00 -6.35404695e-17]
 [ 0.00000000e+00  1.00000000e+00]]
```

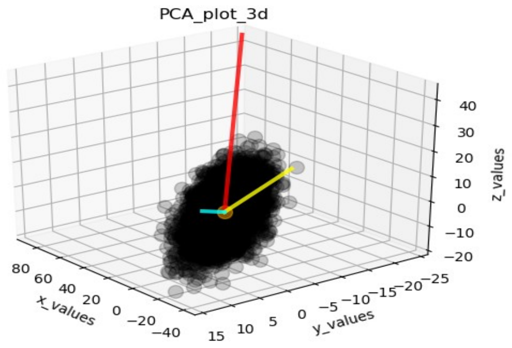
Eigenvvalue(2d points):

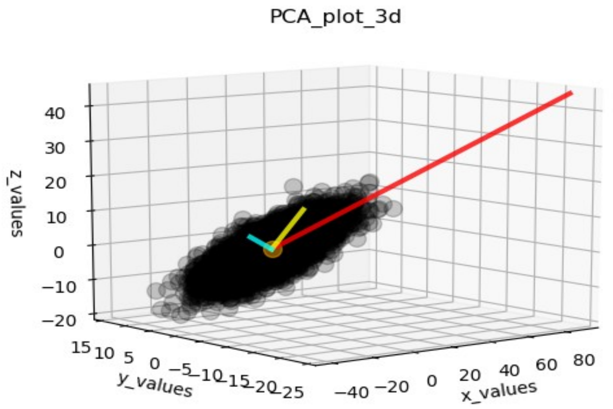
```
[101.60286375  19.89589866]
```

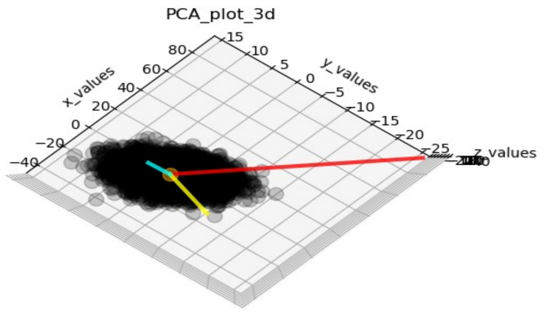
Below are the plots of 3d and new 2d points along with direction of eigenvectors.

<u>3d points:</u>

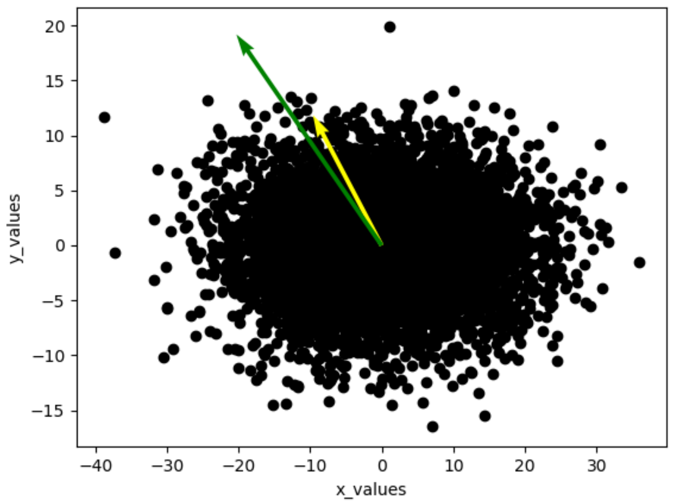


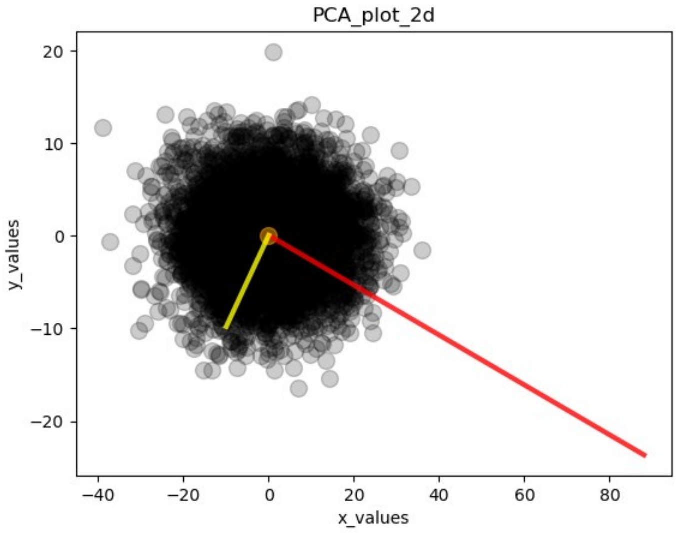






<u>2d points:</u>





Fast Map:

For the given ids of objects and distance between each other, we can embed the objects in points of required dimension = 2. The dimension of embedded points is based on how many iterations we run the program.

Below are the steps used to come up with plot of objects on 2d plane using Fast Map solution:

i. From the given distance between two objects, find the farthest pair having maximum distance.

ii. Find the first coordinate of every point based on projection of each point on the line joining farthest pair. The formula for first coordinate of every point is as below-

$$X_i^j = (d_{ai}^2 + d_{ab}^2 - d_{ib}^2) / 2 d_{ab}^2$$

where j is id of object, i is dimension of coordinates of point, a and b are two farthest points and d denotes distance between the pair of points in subscript.

iii. Update the distance value between every object based on below formula:

$$new\_d_{ij}^2 = d_{ij}^2 - (X_i^2 - X_j^2)$$

where i and j are the two points, d denotes previous distance, new_d denotes new distance between two objects in new space.

iv. Repeat steps i to iii for no. of iterations = no. of dimension. For given problem, 2 iterations.

v. After no. of iterations, plot the points with annotation corresponding to words in word list.

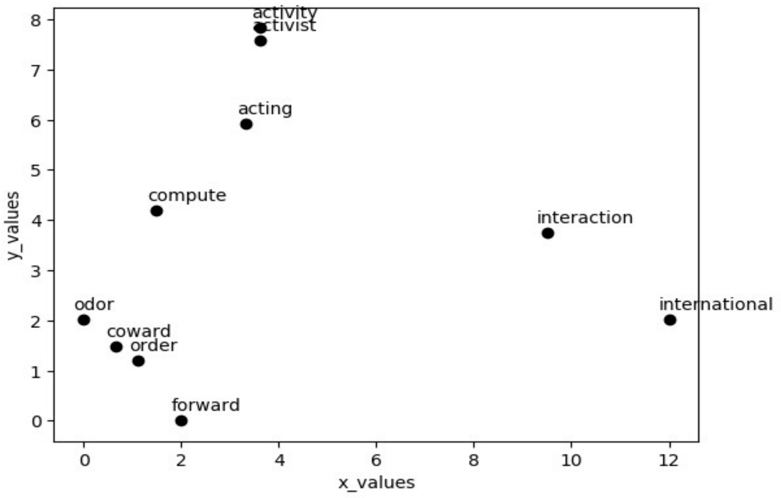
vi. There can be multiple pairs of farthest points. Based on which pair is chosen, the output plot can be different.

Wordlist:

```
['acting', 'activist', 'compute', 'coward', 'forward', 'interaction',
'activity', 'odor', 'order', 'international']
```

Fastmap objects embedded points in 2D:

```
 [[ 3.33333333  5.91497729]
 [ 3.625       7.57789976]
 [ 1.5         4.18888905]
 [ 0.66666667  1.48938277]
 [ 2.          0.        ]
 [ 9.5         3.74207421]
 [ 3.625       7.83322252]
 [ 0.          2.01066674]
 [ 1.125       1.20280957]
 [12.          2.01066674]]
```



Farthest points pair:

1st iteration: `[8, 10]`

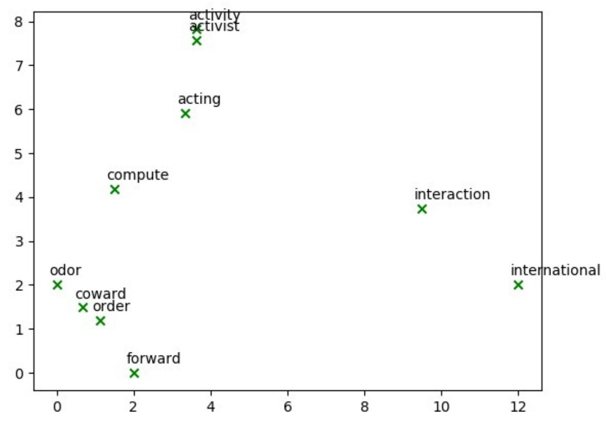
2nd iteration: `[5, 7]`

Notice: the first coordinate of 8th object and second coordinate of 5th object is 0 which is correct based on our theory of fast map and farthest pair in 2 iterations.

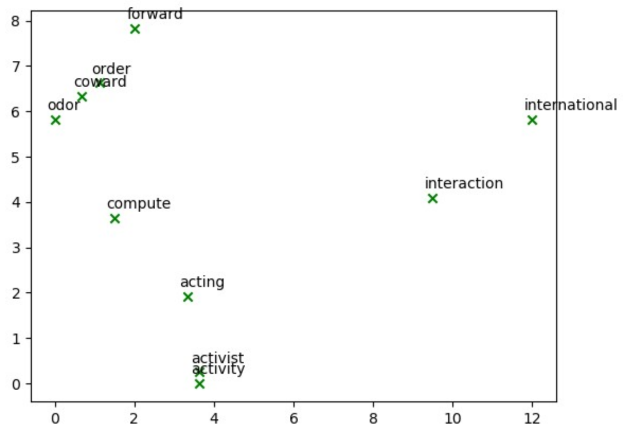
As mentioned, based on which farthest points are picked and in what order, the plots can be changed. For e.g. in given data set, there are multiple pairs which have maximum distance between each other.

In first iteration, there are 3 pairs (in any order) which have maximum distance between each other such as d(3, 10) = d(4, 10) = d(8, 10) = 12. Our program would pick [8,10] or [10,8] as farthest pair.

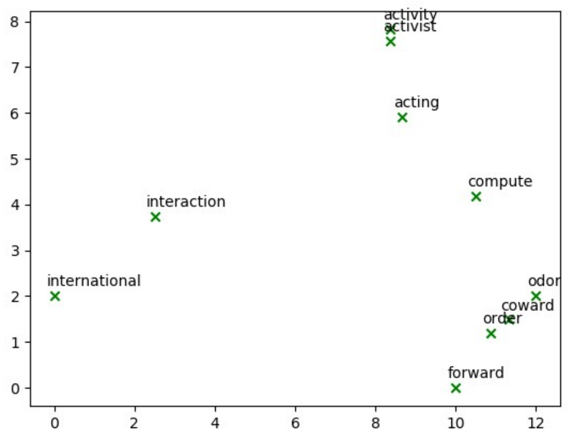
Similarly in second iteration, choice can be made between multiple pairs, however our program picks [5,7] or [7,5]. Below are the 4 plots generate from different order of farthest pair in 2 iterations.



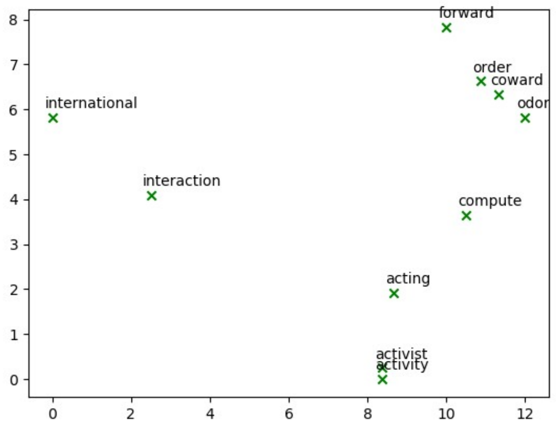
1st iteration: `[8, 10]`
2nd iteration: `[5, 7]`



1st iteration: `[8, 10]`
2nd iteration: `[7, 5]`



1st iteration: `[10, 8]`
2nd iteration: `[5, 7]`



1st iteration: `[10, 8]`
2nd iteration: `[7, 5]`

Data Structure:

1. Pandas dataframe:
    1. To load initial data from the location.
    2. To convert the dataframe to numpy array.
2. Numpy:
    1. To generate random random number from the given list.
    2. To sort the eigenvalues in descending order and corresponding eigenvector.
    3. To take matrix division.
    4. To take matrix dot product.
    5. To take matrix sum across columns.
    6. To find eigenvalue and eigenvector using numpy.linalg.eig library function.
    7. To take random object id of unique object ids using numpy.unique function.
3. Dictionary:
    1. To store the coordinates of each point in new space of k dimension.
    2. To store the distance between a pair of points.
4. Matplotlib:
    1. To scatter plot the 2D points alongwith their mean and eigenvector direction.
    2. To scatter plot the 3D points alongwith their mean and eigenvector direction.
5. Math:
    1. To take square and square root of points.

Code-level optimization:

- Use of list to get index of existing element in constant time so that finding coordinate of every id of object embedded point takes linear time.
- Use of numpy array to access the element in constant time so that update of the distance for every pair of object id takes place in linear time.
- Original distance matrix has distance value in int but updated dataset would have distance value in float so that farthest pair can be chosen based on exact precision of distance value.

Challenges:

- Plotting the eigenvector/direction required bit extra research for appropriate solution.
- Realization of possibility of different plots based on which farthest pair is chosen and in which order came after extensive testing.

Part 2

PCA:

➔ Sklearn offers library function to perform PCA using eigenvector decomposition. The results are exactly matching with results from our output.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=k)
X = pca.fit_transform(points)
print("Points in new dimension(sklearn):\n", X)

eigenvecmat = []
print("Eigenvectors(sklearn):\n")
for eigenvector in pca.components_:
    if eigenvecmat == []:
        eigenvecmat = eigenvector
    else:
        eigenvecmat = np.vstack((eigenvecmat, eigenvector))
print("eigenvector-matrix(sklearn)\n")
print(eigenvecmat)
```

➔ SVD (singular value decomposition) is another way of doing PCA other than eigenvector decomposition. Numpy offers library function as numpy.linalg.svd

```
from numpy.linalg import svd
U, S, Vt = svd(points, full_matrices=False, compute_uv=True)
points_svd = np.dot(U, np.diag(S))
```

Fast Map:

There was no industry standard external library function was found for FastMap algorithm.

Part 3

PCA and Fast Map share similar areas of applications. Below are few of them listed.

A) Images and Multimedia databases: Discover the relation between images and multimedia objects and cluster similar images or multimedia files which enables search-by-content with audio, video etc.
B) Medical database: Mix of 1-d objects(eg. ECGs), 2-d images (eg. X-Rays) and 3-d images (eg. MRI brain scans) are stored and able to retrieve quickly past cases with similar symptoms. This would be valuable for diagnosis as well as for medical teaching and research purposes. Distance functions in this case are complicated, typically requiring some warping of two images, to make sure that the anatomical structures (eg. bones) are properly aligned.
C) Time Series data: In database such as financial data, such as stock prices, sale number, weather data can be examined for similar pattern that may have appeared in past to aid forecasting. Euclidean distance is used between two time series.
D) Similarity searching in string database in case of spelling, typing, and OCR error correction. Helpful in case of approximate matching in DNA database, where there is large collection of strings from a four-letter alphabet (A, C, G, T). Distance is typically editing distance.
E) Data mining and visualization applications on given records of patients with attributes like gender, age, blood-pressure, physicians detect any clusters, or correlation among symptoms, demographic data and diseases.