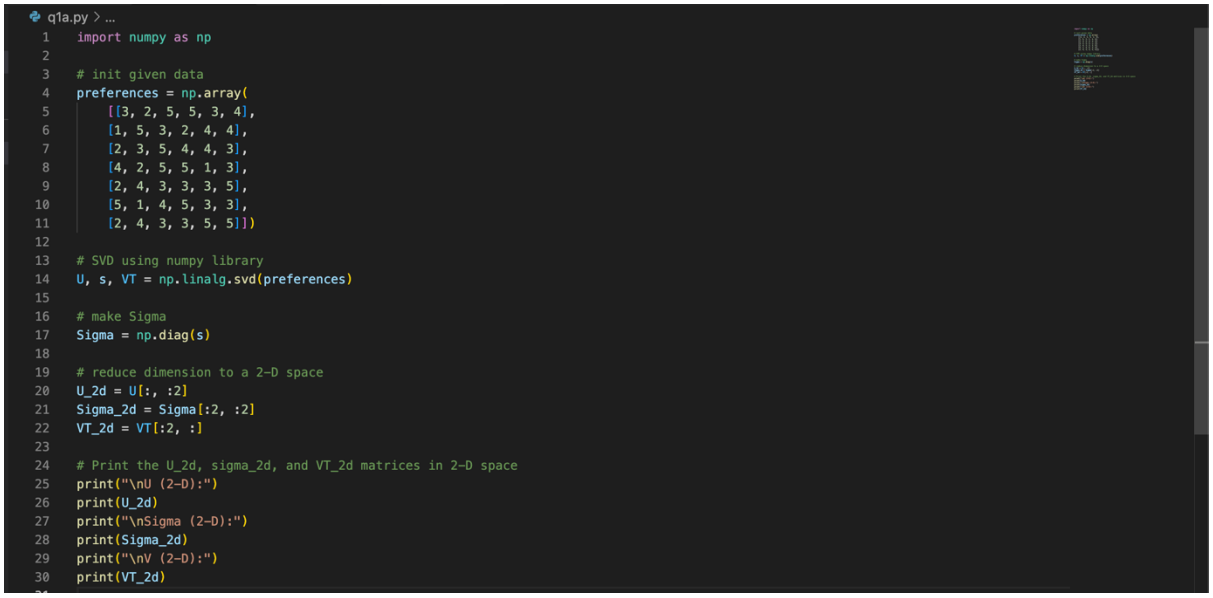
Q1)

I use numpy to do SVD in this question

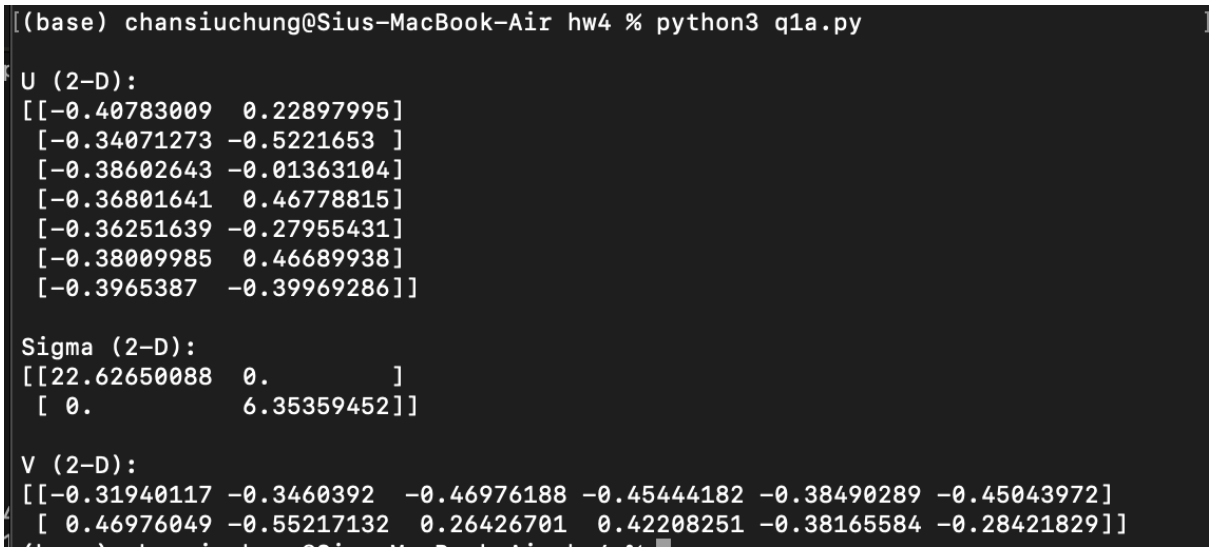
1a)

Code:



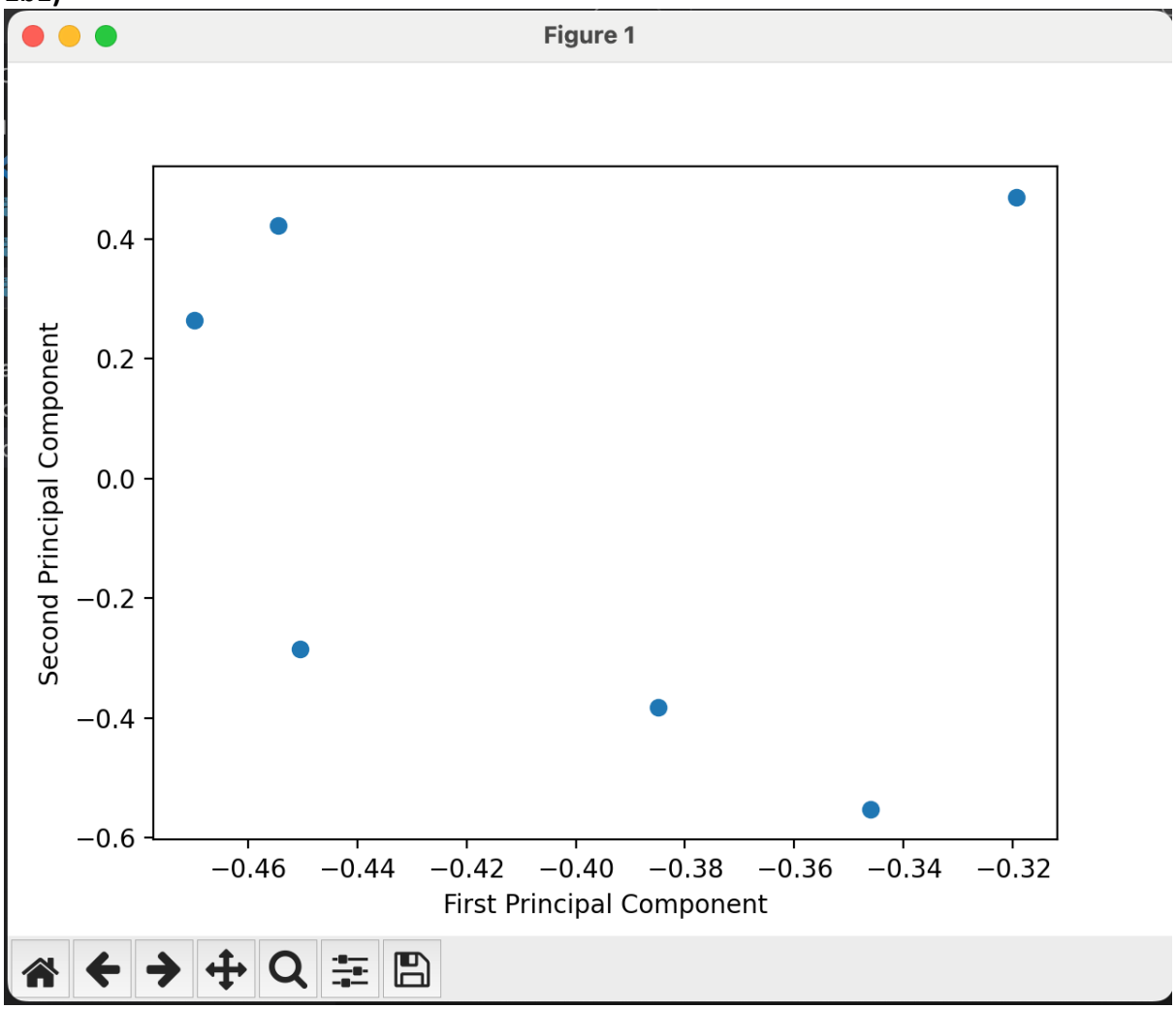
```python
import numpy as np

# init given data
preferences = np.array(
    [[3, 2, 5, 5, 3, 4],
    [1, 5, 3, 2, 4, 4],
    [2, 3, 5, 4, 4, 3],
    [4, 2, 5, 5, 1, 3],
    [2, 4, 3, 3, 3, 5],
    [5, 1, 4, 5, 3, 3],
    [2, 4, 3, 3, 5, 5]])

# SVD using numpy library
U, s, VT = np.linalg.svd(preferences)

# make Sigma
Sigma = np.diag(s)

# reduce dimension to a 2-D space
U_2d = U[:, :2]
Sigma_2d = Sigma[:2, :2]
VT_2d = VT[:2, :]

# Print the U_2d, sigma_2d, and VT_2d matrices in 2-D space
print("\nU (2-D):")
print(U_2d)
print("\nSigma (2-D):")
print(Sigma_2d)
print("\nV (2-D):")
print(VT_2d)
```

Result:



```
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q1a.py

U (2-D):
[[-0.40783009  0.22897995]
 [-0.34071273 -0.5221653 ]
 [-0.38602643 -0.01363104]
 [-0.36801641  0.46778815]
 [-0.36251639 -0.27955431]
 [-0.38009985  0.46689938]
 [-0.3965387  -0.39969286]]

Sigma (2-D):
[[22.62650088  0.        ]
 [ 0.          6.35359452]]

V (2-D):
[[-0.31940117 -0.3460392  -0.46976188 -0.45444182 -0.38490289 -0.45043972]
 [ 0.46976049 -0.55217132  0.26426701  0.42208251 -0.38165584 -0.28421829]]
```

1b1)



First Principal Component concept would be strong taste food or not strong taste food
Second Principal Component concept would be Asian food or non-Asian food.

1b2)

Code:

```python
import numpy as np
import matplotlib.pyplot as plt

# init given data
preferences = np.array(
    [[3, 2, 5, 5, 3, 4],
    [1, 5, 3, 2, 4, 4],
    [2, 3, 5, 4, 4, 3],
    [4, 2, 5, 5, 1, 3],
    [2, 4, 3, 3, 3, 5],
    [5, 1, 4, 5, 3, 3],
    [2, 4, 3, 3, 5, 5]])

# SVD using numpy library
U, s, VT = np.linalg.svd(preferences)

# make Sigma
Sigma = np.diag(s)

# reduce dimension to a 2-D space
U_2d = U[:, :2]
Sigma_2d = Sigma[:2, :2]
VT_2d = VT[:2, :]

# new person
new_person = np.array([4, 2, 4, 5, 3, 2])

# representation in the concept space
new_person_representation = np.dot(new_person, VT_2d.T)

print(new_person_representation)
```

Result:

```
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q1b2.py
[-8.1765278   2.22877582]
```

1b3)

Code:

```python
import numpy as np

# init given data
preferences = np.array(
    [[3, 2, 5, 5, 3, 4],
    [1, 5, 3, 2, 4, 4],
    [2, 3, 5, 4, 4, 3],
    [4, 2, 5, 5, 1, 3],
    [2, 4, 3, 3, 3, 5],
    [5, 1, 4, 5, 3, 3],
    [2, 4, 3, 3, 5, 5]])

# SVD using numpy library
U, s, VT = np.linalg.svd(preferences)

# make Sigma
Sigma = np.diag(s)

# reduce dimension to a 2-D space
U_2d = U[:, :2]
Sigma_2d = Sigma[:2, :2]
VT_2d = VT[:2, :]

cos_sim = np.dot(preferences[3], preferences[6])/(np.linalg.norm(preferences[3])*np.linalg.norm(preferences[6]))
print(f"cosine similarity:\n{ cos_sim}")

cos_sim_2d = np.dot(U_2d[3], U_2d[6])/(np.linalg.norm(U_2d[3])*np.linalg.norm(U_2d[6]))
print(f"\nnew cosine similarity:\n{ cos_sim_2d}")
```

Result:

```
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q1b3.py
cosine similarity:
0.7866066361276135

new cosine similarity:
-0.12246304141824685
```

Q2)

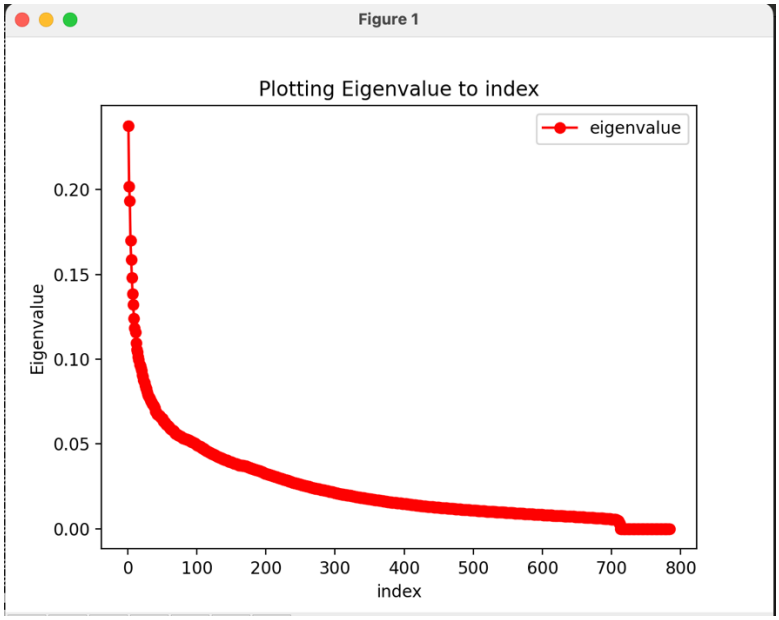
In this question, I use PCA from sklearn, and I followed the suggestion that I should normalize the data points doing PCA so I also used another tool from sklearn to normalize the data before doing the job.

2a1)
To find the energy left in different number of eigen digits

Code:

```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Read the image data from the file
train_images = []
with open("MNIST/train_img", "r") as file:
    for line in file:
        # Convert each line to a list of integers
        image_data = [int(pixel) for pixel in line.strip().split(',')]
        train_images.append(image_data)

train_images = np.array(train_images)

# Standardize the data
sc = StandardScaler()
X_train = sc.fit_transform(train_images)

# Apply PCA
pca = PCA()
X_train = pca.fit_transform(X_train)

# Check the explained variance
explained_variance = pca.explained_variance_ratio_
eigenvalues = np.sqrt(explained_variance)
print('Eigenvalues: {}'.format(eigenvalues))

# show the result
x = np.array(range(1, len(eigenvalues)+1))
y = np.array(eigenvalues)

plt.title("Plotting Eigenvalue to index")
plt.xlabel("index")
plt.ylabel("Eigenvalue")
plt.plot(x, y, color="red", marker="o", label="eigenvalue")
plt.legend()
plt.show()
```

Result:



2a2)

First let's find the energy left in different M values. I first normalize the training images for a better result as suggested in doing PCA.

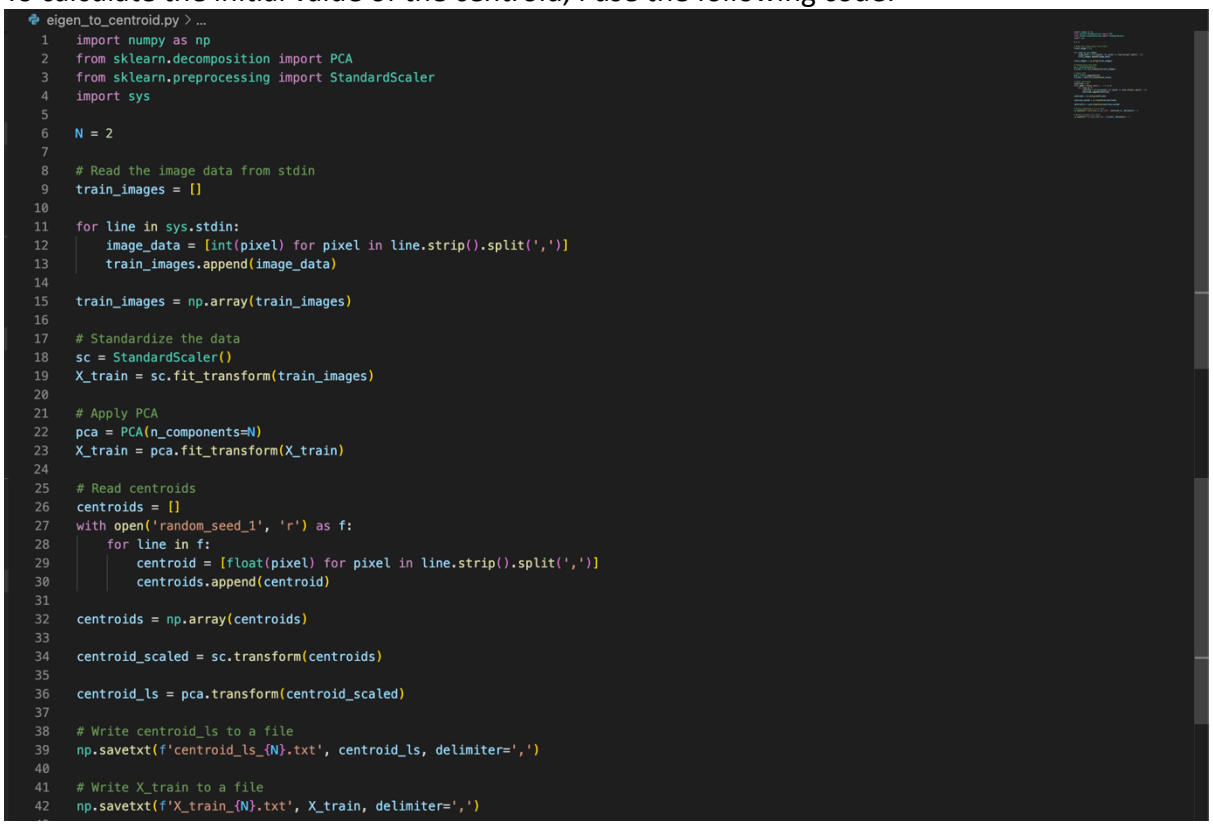
Code:
Change line 21:
`pca = PCA(n_components=<Eigendigits>)`

q2a1.py > ...

```
1   import numpy as np
2   from sklearn.decomposition import PCA
3   from sklearn.preprocessing import StandardScaler
4   import matplotlib.pyplot as plt
5
6   # Read the image data from the file
7   train_images = []
8   with open("MNIST/train_img", "r") as file:
9       for line in file:
10          # Convert each line to a list of integers
11          image_data = [int(pixel) for pixel in line.strip().split(',')]
12          train_images.append(image_data)
13
14  train_images = np.array(train_images)
15
16  # Standardize the data
17  sc = StandardScaler()
18  X_train = sc.fit_transform(train_images)
19
20  # Apply PCA
21  pca = PCA(n_components=128)
22  X_train = pca.fit_transform(X_train)
23
24  # Check the explained variance
25  explained_variance = pca.explained_variance_ratio_
26  print("energy: {}".format(np.sum(explained_variance)))
27  eigenvalues = np.sqrt(explained_variance)
28  print('Eigenvalues: {}'.format(eigenvalues))
29
30  # show the result
31  x = np.array(range(1, len(eigenvalues)+1))
32  y = np.array(eigenvalues)
33
34  plt.title("Plotting Eigenvalue to index")
35  plt.xlabel("index")
36  plt.ylabel("Eigenvalue")
37  plt.plot(x, y, color="red", marker="o", label="eigenvalue")
38  plt.legend()
39  plt.show()
```

Results: (shown later at the end of this question)

To calculate the initial value of the centroid, I use the following code:



```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import sys

N = 2

# Read the image data from stdin
train_images = []

for line in sys.stdin:
    image_data = [int(pixel) for pixel in line.strip().split(',')]
    train_images.append(image_data)

train_images = np.array(train_images)

# Standardize the data
sc = StandardScaler()
X_train = sc.fit_transform(train_images)

# Apply PCA
pca = PCA(n_components=N)
X_train = pca.fit_transform(X_train)

# Read centroids
centroids = []
with open('random_seed_1', 'r') as f:
    for line in f:
        centroid = [float(pixel) for pixel in line.strip().split(',')]
        centroids.append(centroid)

centroids = np.array(centroids)

centroid_scaled = sc.transform(centroids)

centroid_ls = pca.transform(centroid_scaled)

# Write centroid_ls to a file
np.savetxt(f'centroid_ls_{N}.txt', centroid_ls, delimiter=',')

# Write X_train to a file
np.savetxt(f'X_train_{N}.txt', X_train, delimiter=',')
```

Adjust N = <number of eigenvalues> on line 6 to get all 4 pairs of files for Hadoop operation.

And use command:
cat MNIST/train_img| python3 eigen_to_centroid.py

Now we have 4 initial files for centroid initialization and 4 files of training data after PCA transformation.

Put all 4 X_train_<num_of_eigenvalues>.txt to hdfs
and run the following mapreduce code:

1st Mapper:

q2a2_mapper.py > ...

```python
#!/usr/bin/env python3
"""q2a2_mapper.py"""

import sys
import numpy as np

centroid_ls = []
vector_ls = []

with open("centroid_ls_2.txt", 'r') as f:
    for line in f:
        #centroid_ls.append(np.array(list(ast.literal_eval(line.strip()))))
        centroid_ls.append(np.fromstring(line.strip(), sep=","))

count = 0
for line in sys.stdin:
    count += 1
    vector_ls.append(np.fromstring(line.strip(), sep=","))


for vector in vector_ls:
    # compare
    nearest_ct_key = -1
    min_distance = None
    for key, ct in enumerate(centroid_ls):
        if min_distance == None:
            min_distance = np.sum(np.square(vector - ct))
            nearest_ct_key = key
        else:
            cur_dis = np.sum(np.square(vector - ct))
            if cur_dis < min_distance:
                min_distance = cur_dis
                nearest_ct_key = key

    print(f"{nearest_ct_key}\t{list(vector)}")
```

1st reducer:

q2a2_reducer.py > ...

```python
#!/usr/bin/env python3
"""q2a2_reducer.py"""

import sys
import numpy as np
import ast

current_centroid_key = None
ct_vector = None
cur_count = 0
centroid_key = None

for line in sys.stdin:
    centroid_key, vector = line.strip().split('\t')
    vector = ast.literal_eval(vector)

    if centroid_key != current_centroid_key:
        if current_centroid_key != None:
            print(f"{current_centroid_key}\t{cur_count}\t{list(ct_vector/cur_count)}")
        cur_count = 1
        current_centroid_key = centroid_key
        ct_vector = np.array(vector)
    else:
        ct_vector += np.array(vector)
        cur_count += 1

if current_centroid_key == centroid_key:
    print(f"{current_centroid_key}\t{cur_count}\t{list(ct_vector/cur_count)}")
```

2nd Mapper:

```python
# q2a2_mapper_2.py > ...

centroid_ls = []
vector_ls = []

count = 0
for line in sys.stdin:
    count += 1
    vector_ls.append(np.fromstring(line.strip(), sep=","))

with open("eigen2_output_file" , 'r') as f:
    temp_ls = []
    for line in f:
        centroid_id, count ,ct_vector = line.strip().split('\t')
        temp_ls.append((int(centroid_id), ct_vector))

    temp_ls = sorted(temp_ls, key=lambda x:x[0])

    for i in temp_ls:
        centroid_ls.append(np.array(ast.literal_eval(i[1])))

for vector in vector_ls:
    # compare
    nearest_ct_key = -1
    min_distance = None
    for key, ct in enumerate(centroid_ls):
        if min_distance == None:
            min_distance = np.sum(np.square(vector - ct))
            nearest_ct_key = key
        else:
            cur_dis = np.sum(np.square(vector - ct))
            if cur_dis < min_distance:
                min_distance = cur_dis
                nearest_ct_key = key

    print(f"{nearest_ct_key}\t{list(vector)}")
```

2nd Reducer (same as 1st reducer):

```python
# q2a2_reducer.py > ...
#!/usr/bin/env python3
"""q2a2_reducer.py"""

import sys
import numpy as np
import ast

current_centroid_key = None
ct_vector = None
cur_count = 0
centroid_key = None

for line in sys.stdin:
    centroid_key, vector = line.strip().split('\t')
    vector = ast.literal_eval(vector)

    if centroid_key != current_centroid_key:
        if current_centroid_key != None:
            print(f"{current_centroid_key}\t{cur_count}\t{list(ct_vector/cur_count)}")
        cur_count = 1
        current_centroid_key = centroid_key
        ct_vector = np.array(vector)
    else:
        ct_vector += np.array(vector)
        cur_count += 1

if current_centroid_key == centroid_key:
    print(f"{current_centroid_key}\t{cur_count}\t{list(ct_vector/cur_count)}")
```

Loop through 2nd Mapper and reducer until converge using shell command, example:

```
$ q2a2.sh
#!/bin/bash

for i in {1..150}; do
    echo "Running iteration $i..."
    hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming-2.10.1.jar \
    -files hdfs://dicvmc2.ie.cuhk.edu.hk/user/s1155157657/hw4/q2a2_output/eigen2_output_file \
    -D mapred.map.tasks=5 -D mapred.reduce.tasks=1 -file q2a2_mapper_2.py -mapper q2a2_mapper_2.py \
    -file q2a2_reducer.py -reducer q2a2_reducer.py -input hw4/X_train_2.txt -output hw4/q2a2_output/eigen2_output_$i

    hadoop fs -cp -f hw4/q2a2_output/eigen2_output_$i/part-00000 hw4/q2a2_output/eigen2_output_file
    sync
done
```

All command used for Hadoop is placed in command_used.sh file

Using the technique of diff the output file, for example:
```
diff <(hadoop fs -cat hw4/q2a2_output/eigen128_output_102/part-00000)
<(hadoop fs -cat hw4/q2a2_output/eigen128_output_103/part-00000)
```

Eigen2 observed converge after 121 iterations
Eigen8 observed converge after 74 iterations.
Eigen32 observed converge after 171 iterations.
Eigen128 observed converge after 102 iterations.

After that I get the output file to local fs using commands:
```
hadoop fs -get hw4/q2a2_output/eigen2_output_121 .
hadoop fs -get hw4/q2a2_output/eigen8_output_75 .
hadoop fs -get hw4/q2a2_output/eigen32_output_172 .
hadoop fs -get hw4/q2a2_output/eigen128_output_103 .

For the result, I stored it in the file named: eigen2, eigen8,
eigen32, eigen128
```

For checking accuracy, I use the following program:
Output of the following program will be in this format <prediction>,<actual_label>\t<count>

check_details.py > ...

```python
#!/usr/bin/env python3
"""check_details.py"""

import numpy as np
import ast

vector_ls = []
label_ls = []
count = 0
centroid_ls = []

with open('X_train_2', 'r') as f:
    for line in f:
        count += 1
        vector_ls.append(np.fromstring(line.strip(), sep=","))

with open('MNIST/train_label', 'r') as f:
    for line in f:
        label_ls.append(int(line.strip()))

with open('eigen2_output_121', 'r') as f:
    for line in f:
        i, each_count,vt = line.strip().split('\t')
        centroid_ls.append(np.array(ast.literal_eval(vt)))


for i, vector in enumerate(vector_ls):
    # compare
    nearest_ct_key = -1
    min_distance = None
    for key, ct in enumerate(centroid_ls):
        if min_distance == None:
            min_distance = np.sum(np.square(vector - ct))
            nearest_ct_key = key
        else:
            cur_dis = np.sum(np.square(vector - ct))
            if cur_dis < min_distance:
                min_distance = cur_dis
                nearest_ct_key = key

    print(f"{str(nearest_ct_key)},{str(label_ls[i])}\t{1}")
```

Then I run the command:
```
python3 check_details.py| sort| python3 check_details_reducer.py
```

outputs:
Eigen2:

```
0,0	1381
0,2	44
0,3	23
0,4	38
0,5	29
0,6	229
0,7	7
0,8	36
0,9	23
1,0	32
1,1	6095
1,2	184
1,3	589
1,4	1044
1,5	428
1,6	667
1,7	1389
1,8	923
1,9	1688
2,0	899
2,2	956
2,3	507
2,4	445
2,5	462
2,6	1066
2,7	97
2,8	250
2,9	140
3,0	373
3,1	36
3,2	1624
3,3	1300
3,4	1158
3,5	997
3,6	1338
3,7	456
3,8	878
3,9	436
4,0	130
4,1	2
4,2	116
4,3	308
4,4	381
4,5	309
4,6	180
4,7	220
4,8	207
4,9	384
5,0	1011
5,1	2
5,2	1302
5,3	704
5,4	128
5,5	499
5,6	646
5,7	29
5,8	562
5,9	43
6,0	1872
6,2	290
6,3	54
6,4	4
6,5	18
6,6	106
6,8	27
6,9	4
7,0	16
7,1	6
7,2	639
7,3	974
7,4	864
7,5	301
7,6	71
7,7	1375
7,8	471
7,9	1017
8,0	194
8,1	451
8,2	377
8,3	870
8,4	546
8,5	2123
8,6	1555
8,7	300
8,8	1806
8,9	270
9,0	15
9,1	150
9,2	426
9,3	882
9,4	1234
9,5	255
9,6	60
9,7	2392
9,8	691
9,9	1944
```

Eigen8:

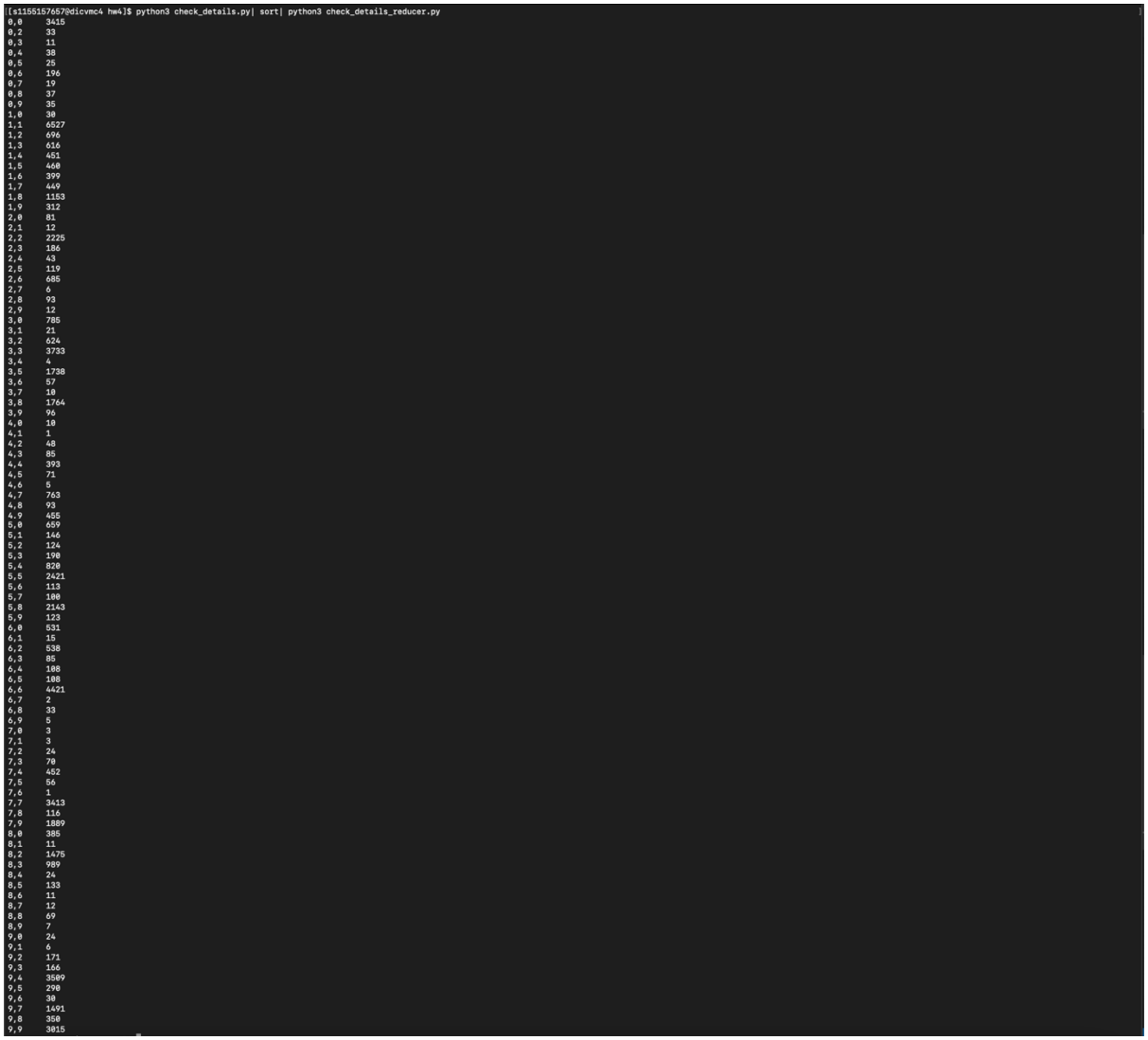
```
[s1155157657@dicvmc4 hw4]$ python3 check_details.py| sort| python3 check_details_reducer.py
0,0	2746
0,2	14
0,3	9
0,4	38
0,5	43
0,6	234
0,7	14
0,8	44
0,9	27
1,0	36
1,1	6385
1,2	628
1,3	581
1,4	529
1,5	898
1,6	494
1,7	570
1,8	779
1,9	447
2,0	61
2,1	13
2,2	2585
2,3	232
2,4	38
2,5	125
2,6	554
2,7	12
2,8	134
2,9	15
3,0	576
3,1	21
3,2	330
3,3	3567
3,4	1
3,5	1686
3,6	40
3,7	13
3,8	1417
3,9	85
4,0	46
4,1	5
4,2	162
4,3	128
4,4	3791
4,5	486
4,6	32
4,7	488
4,8	589
4,9	2592
```

```
5,0	286
5,1	302
5,2	442
5,3	936
5,4	328
5,5	1692
5,6	71
5,7	186
5,8	2545
5,9	203
6,0	477
6,1	11
6,2	641
6,3	104
6,4	178
6,5	186
6,6	4375
6,7	2
6,8	37
6,9	5
7,0	6
7,1	1
7,2	39
7,3	99
7,4	497
7,5	68
7,6	5
7,7	1038
7,8	80
7,9	483
8,0	1767
8,2	1086
8,3	414
8,4	59
8,5	236
8,6	113
8,7	5
8,8	118
8,9	20
9,0	2
9,1	4
9,2	31
9,3	69
9,4	383
9,5	161
9,7	3937
9,8	188
9,9	2072
```

Eigen32:

```
[s115515765?@dicvmc4 hw4]$ python3 check_details.py| sort| python3 check_details_reducer.py
0,0	3362
0,2	29
0,3	11
0,4	42
0,5	28
0,6	230
0,7	18
0,8	39
0,9	38
1,0	31
1,1	6526
1,2	699
1,3	537
1,4	464
1,5	421
1,6	464
1,7	448
1,8	1108
1,9	318
2,0	91
2,1	12
2,2	2081
2,3	452
2,4	50
2,5	168
2,6	441
2,7	10
2,8	143
2,9	16
3,0	965
3,1	24
3,2	623
3,3	3613
3,4	2
3,5	1814
3,6	57
3,7	10
3,8	2019
3,9	101
4,0	16
4,1	6
4,2	158
4,3	181
4,4	3302
4,5	277
4,6	19
4,7	1346
4,8	307
4,9	2889
5,0	555
5,1	144
5,2	73
5,3	119
5,4	777
5,5	2350
5,6	111
5,7	114
5,8	1876
5,9	113
6,0	457
6,1	13
6,2	721
6,3	83
6,4	141
6,5	105
6,6	4579
6,7	3
6,8	39
6,9	6
7,0	7
7,1	1
7,2	34
7,3	93
7,4	350
7,5	63
7,6	5
7,7	776
7,8	78
7,9	417
8,0	432
8,1	9
8,2	1505
8,3	974
8,4	26
8,5	137
8,6	11
8,7	13
8,8	72
8,9	7
9,0	7
9,1	7
9,2	35
9,3	68
9,4	688
9,5	58
9,6	1
9,7	3527
9,8	170
9,9	2044
```

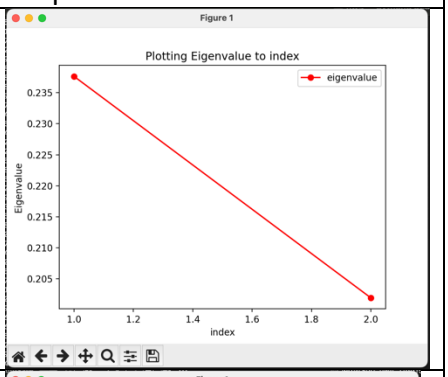
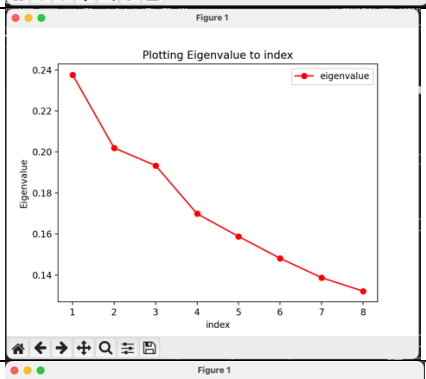
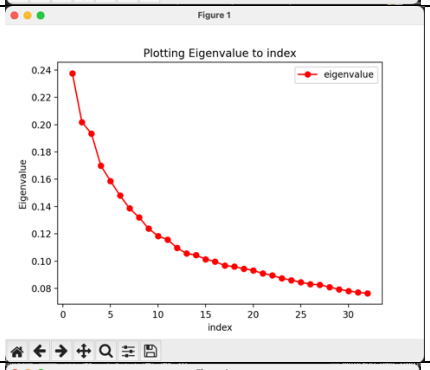
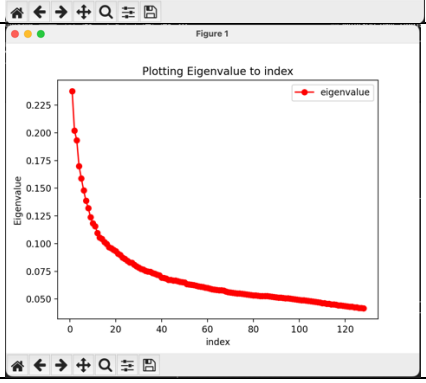
Eigen128:



i) Performance in terms of accuracy:

	Accuracy
Eigen2	15843/60000=26.4%
Eigen8	28369/60000=47.3%
Eigen32	28705/60000=47.8%
Eigen128	29632/60000=49.4%

ii) Energy kept:

Eigendigits	Energy Kept	Graph
2	0.09724988889132913	
8	0.2473398709397076	
32	0.46367542615792245	
128	0.7629398151669453	

iii) In general, the accuracy of the data after PCA is worse than the accuracy before PCA (~63.04%).
The accuracy is better when more principal component vector is used.

2a3)
For a bigger M, the classification accuracy will be higher but the time to run MapReduce will be longer and requires more storage space too.

2b)

First save the eigen vector to eigen8_vector.txt.

find_eigenvectors.py > ...

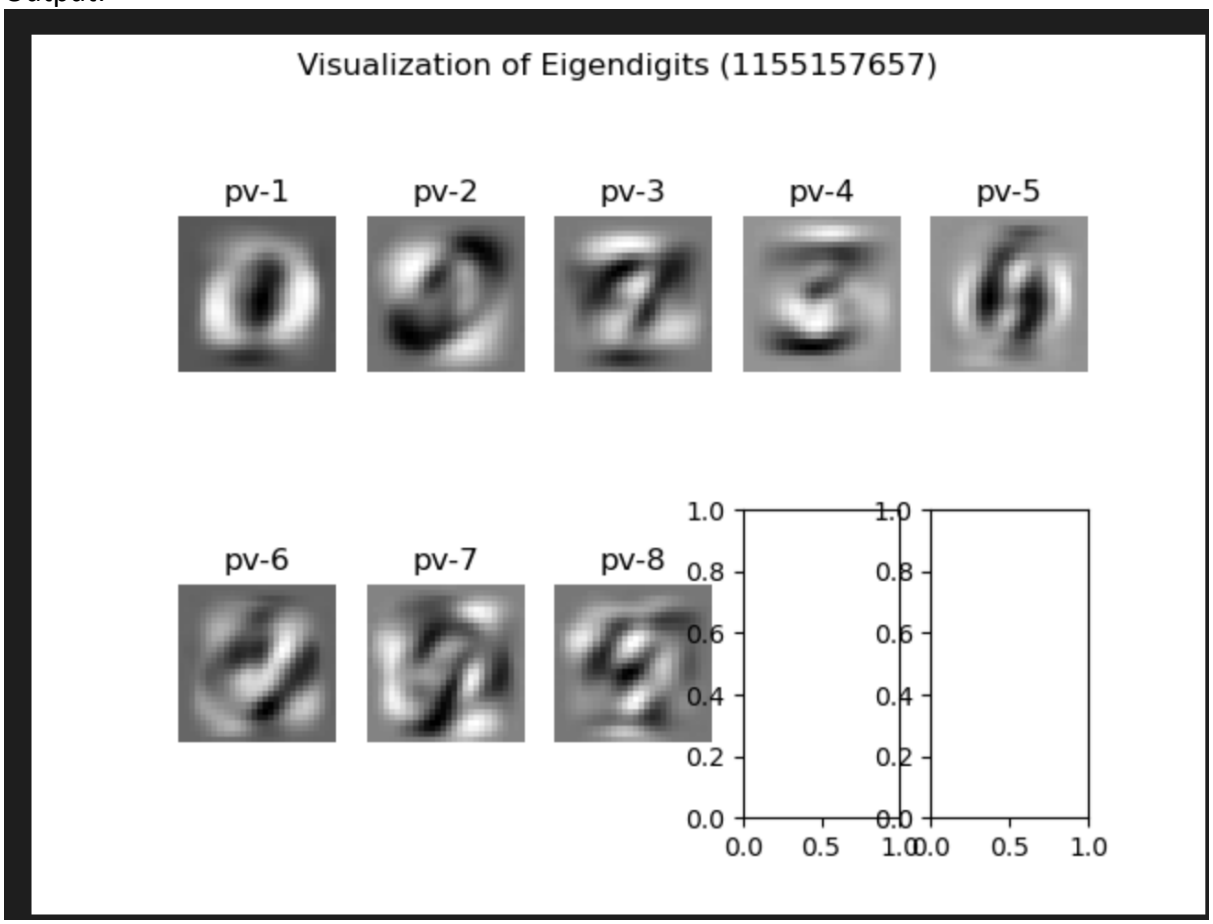
```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import sys

N = 8

# Read the image data from stdin
train_images = []

for line in sys.stdin:
    image_data = [int(pixel) for pixel in line.strip().split(',')]
    train_images.append(image_data)

train_images = np.array(train_images)

# Standardize the data
sc = StandardScaler()
X_train = sc.fit_transform(train_images)

# Apply PCA
pca = PCA(n_components=N)
X_train = pca.fit_transform(X_train)

# Read centroids
centroids = []
with open('random_seed_1', 'r') as f:
    for line in f:
        centroid = [float(pixel) for pixel in line.strip().split(',')]
        centroids.append(centroid)

centroids = np.array(centroids)

centroid_scaled = sc.transform(centroids)

centroid_ls = pca.transform(centroid_scaled)

np.savetxt(f'eigen8_vector.txt', pca.components_, delimiter=',')
```

Second, use the provide code, modify the SID and the pv_dir and fix reduce_dim to 8.

viz_principal_vectors.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt

reduce_dim = 8
num_viz_pca = 10


def main():
    # load principle vectors from your local machine, e.g., here we suppose that the PCA vectors of the training
    # set is stored in a file named 'pca_components_20' in the ./mnist/ directory
    pv_dir = './eigen8_vector.txt'
    pvs = np.empty((reduce_dim, 784))
    with open(pv_dir, 'r') as f:
        # Some pre-processing steps, depending on how you store your principle vectors
        # Here, we suppose that each row is corresponding to a principle vectors with string
        # type, e.g, a row can be: '0.5, 0.2, 0.3, ..., 0.21', each element is separated by ','.
        for k, line in enumerate(f.readlines()):
            line = line.strip().split(',')
            # the k-th principle vectors
            pv_k = np.array(list(map(float, line)))
            pvs [k] = pv_k

    # visualize the principle vectors
    fig, ax = plt.subplots(num_viz_pca // 5, 5)
    # YOU NEED TO FILL IN YOUR SID IN THE FOLLOWING LINE
    fig.suptitle('Visualization of Eigendigits ({})'.format('1155157657'))
    ax = ax.reshape(-1)
    for k, pv in enumerate(pvs [:num_viz_pca]):
        pv = pv.reshape(28, 28)
        ax[k].imshow(pv, cmap='gray')
        ax[k].set_title('pv-{}'.format(k + 1))
        ax[k].axis('off')

    fig.savefig('./viz_pca_vectors_{}.png'.format(num_viz_pca))


if __name__ == '__main__':
    main()
```

Output:



Observation:
Some of the eigenvectors looks like numbers , for example ‘3’ and ‘7’ and ‘9’. Some of them contains common components of a number such as straight lines and circles.

Q3)

I am using the method stated in slide p.38

3a) Code (User-to-User):

q3a_user.py > ...

```python
import numpy as np
from numpy.linalg import norm

rating_mat = np.array([
    [5, 0, 4, 1, 4, 2],
    [2, 3, 4, 5, 0, 0],
    [0, 2, 0, 3, 5, 4],
    [2, 1, 5, 4, 3, 0],
    [0, 4, 1, 0, 3, 2]
])

# User-User
avg_rating_mat = []

for i in rating_mat:
    mean_ = i[np.nonzero(i)].mean()
    avg_rating_mat.append(np.where(i!=0, i-mean_, i))

avg_rating_mat = np.array(avg_rating_mat)

cosine_sim = []
for key, i in enumerate(avg_rating_mat):
    cosine = np.dot(avg_rating_mat[2],i)/(norm(avg_rating_mat[2])*norm(i))
    cosine_sim.append((cosine, key))

cosine_sim.sort(reverse=True, key=lambda x: x[0])

# cosine_sim is a list that is reversely sorted with
# cosine similarity of each vector to the target vector (cosine_similarity, key_of_vector)
print(f"User-to-User:{(cosine_sim[1][0]*rating_mat[cosine_sim[1][1]][0] + cosine_sim[2][0]*rating_mat[cosine_sim[2][1]][0])/(cosine_sim[1][0]+cosine_sim[2][0])}")
```

Code (item-to-item):

q3a_item.py > ...

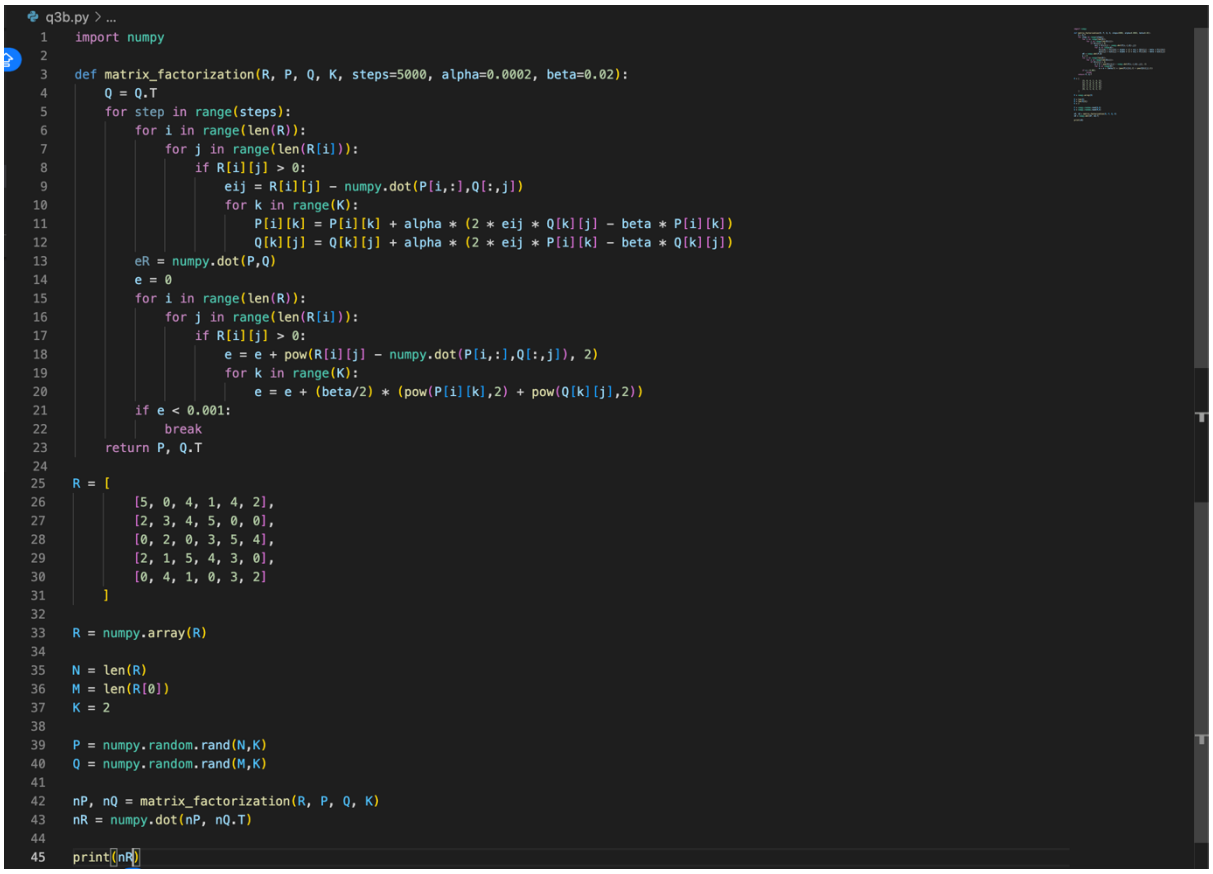
```python
import numpy as np
from numpy.linalg import norm

# item-item
rating_mat = np.array([
    [5, 0, 4, 1, 4, 2],
    [2, 3, 4, 5, 0, 0],
    [0, 2, 0, 3, 5, 4],
    [2, 1, 5, 4, 3, 0],
    [0, 4, 1, 0, 3, 2]
])

rating_mat = rating_mat.T

# Item-Item
avg_rating_mat = []

for i in rating_mat:
    mean_ = i[np.nonzero(i)].mean()
    avg_rating_mat.append(np.where(i!=0, i-mean_, i))

avg_rating_mat = np.array(avg_rating_mat)

cosine_sim = []
for key, i in enumerate(avg_rating_mat):
    cosine = np.dot(avg_rating_mat[0],i)/(norm(avg_rating_mat[0])*norm(i))
    cosine_sim.append((cosine, key))

cosine_sim.sort(reverse=True, key=lambda x: x[0])

# cosine_sim is a list that is reversely sorted with
# cosine similarity of each vector to the target vector (cosine_similarity, key_of_vector)
print(f"Item-to-Item:{(cosine_sim[1][0]*rating_mat[cosine_sim[1][1]][2]+cosine_sim[2][0]*rating_mat[cosine_sim[2][1]][2])/(cosine_sim[1][0]+cosine_sim[2][0])}")
```

3ai and 3aii) output:

```
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q3a_user.py
User-to-User:3.1865773776947957
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q3a_item.py
Item-to-Item:3.8828894492238284
```

3b)

Code:



```python
import numpy

def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in range(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
        eR = numpy.dot(P,Q)
        e = 0
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    for k in range(K):
                        e = e + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
        if e < 0.001:
            break
    return P, Q.T

R = [
        [5, 0, 4, 1, 4, 2],
        [2, 3, 4, 5, 0, 0],
        [0, 2, 0, 3, 5, 4],
        [2, 1, 5, 4, 3, 0],
        [0, 4, 1, 0, 3, 2]
    ]

R = numpy.array(R)

N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N,K)
Q = numpy.random.rand(M,K)

nP, nQ = matrix_factorization(R, P, Q, K)
nR = numpy.dot(nP, nQ.T)

print(nR)
```

Result:

```
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q3b.py
[[ 4.86056227  4.48916395  3.23429429  0.97645988  4.28725968  2.68425545]
 [ 2.43476092  2.09572284  4.81381808  4.24018163  4.18338121  3.97955249]
 [ 3.02113264  2.66943339  4.53302056  3.56990306  4.27287898  3.74978736]
 [ 1.59467407  1.31380329  4.38056204  4.21910978  3.5225417   3.61936348]
 [ 4.1779845   3.91404453  1.62570427 -0.51652239  2.94933102  1.35487044]]
```

Prediction of User 3 on product A is 3.02113264 while in part a the result is 3.1865776947957 for user-to-user collaborative filtering and 3.8828894492238284 for item-to-item collaborative filtering. The score is lower than the method of collaborative filtering.

3c)

Code:
P[i][k] is updated first and used in Q[k][j] to calculate new value which is a bug because it affected the accuracy.

Corrected it using a temp variable to store the value first and use it in line 13
Code:

```python
import numpy

def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i,:],Q[:,j])
                    for k in range(K):
                        temp = P[i][k]
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * temp - beta * Q[k][j])
        eR = numpy.dot(P,Q)
        e = 0
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
                    for k in range(K):
                        e = e + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
        if e < 0.001:
            break
    return P, Q.T
```

Result:

```
(base) chansiuchung@Sius-MacBook-Air hw4 % python3 q3b.py
[[4.89300497 5.11716277 3.20631652 0.99268673 4.35212632 2.53231529]
 [2.43987868 2.211466   4.77928555 4.26920415 4.31464542 4.38690432]
 [2.7636278  2.62020556 4.33550825 3.55651084 4.1603402  3.91014088]
 [1.56194123 1.28201502 4.30957074 4.21638527 3.60494843 4.03625474]
 [3.65252452 3.88146515 1.81743177 0.05747216 2.8603837  1.32449939]]
```

The updated value is 2.7636278 which is lower than that in 3b