Assignment 6

Team: Ziqi Sha, Qiang Yang, Zhou Xie

```
from google.colab import drive
drive.mount("/content/drive", force_remount=True)
```

Question 1

$$w = \begin{vmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{vmatrix}$$

$$X = \begin{vmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{vmatrix}$$

$$a = \begin{vmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{vmatrix}$$

argmin is the same when mutiply the error with N.

$$\arg\min_a \frac{1}{N} \sum_{n=1}^{N} \|x^{(n)} - a_n w\|^2$$

$$\sum_{n=1}^{N} \|x^{(n)} - a_n w\|^2 = |\, x_1 - a_1 w, x_2 - a_2 w, \ldots, x_1 - a_1 w| \begin{vmatrix} x_1 - a_1 w \\ x_2 - a_2 w \\ \vdots \\ x_n - a_n w \end{vmatrix}$$

$$= (X^T - w^T a^T)(X - aw)$$
$$= X^T X - X^T aw - w^T a^T X + w^T a^T aw$$

Since a is a scalar and X^Tw = w^TX dL/da = 0. Then we take dirivate of L over a, we get

$$\frac{\nabla L}{\nabla a} = 2aw^T w - 2x^T w = 0$$
$$a = X^T w(w^T w)^{-1}$$

Question 2

$$\min_a \frac{1}{N} \sum_{n=1}^{N} \|x^{(n)} - a_n w\|^2$$

$$\sum_{n=1}^{N} \|x^{(n)} - a_n w\|^2 = (X^T - w^T a^T)(X - aw)$$
$$= X^T X - 2X^T aw + w^T a^T aw$$

Plug in the value of a in question 1

$$= X^T X - 2X^T X^T w(w^T w)^{-1} w + w^T (X^T w(w^T w)^{-1})^T X^T w(w^T w)^{-1} w$$

Divide it by N we get

$$\min_a \frac{1}{N} \sum_{n=1}^{N} \|x^{(n)} - a_n w\|^2 = (X^T X - 2X^T X^T w(w^T w)^{-1} w$$
$$+ w^T (X^T w(w^T w)^{-1})^T X^T w(w^T w)^{-1} w)/N$$

Question 3

We have ||w|| = 1 which means $w^T w$ = 1, simplify question 2

$$X^T X - 2X^T X^T ww + w^T w^T XX^T ww$$

Since the covariance matrix of the data is defined as

$$C = \frac{1}{N} \sum_{n=1}^{N} x^{(n)} x^{(n)\top}$$

Since minimizing the reconstruction error is equivalent to maximizing the variance along the principal component.

The original equation will become

$$max_w w^T C w$$
$$s.t. w^T w = I$$

According to the Lagrangian duality:

$$L = max_w w^T C w + \lambda(w^T w - 1)$$

Take derivative of this over w we get

$$\frac{\nabla L}{\nabla w} = 2w^T C - 2\lambda w^T$$

Set the derivative equal to zero, we will get $w^T C = \lambda w^T$, which is equivalent to

$$Cw = \lambda w$$

Therefore $Cw = \lambda w$

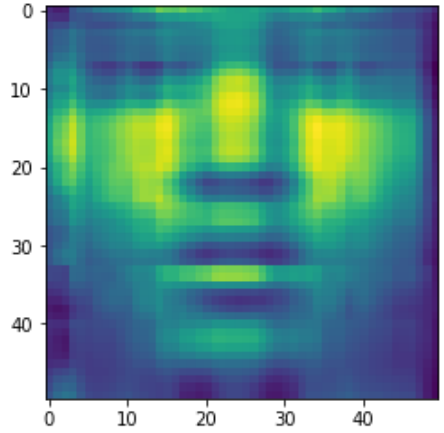
Question 4

The minimum number of dimensions we need is D/2 since every even index n of x could be represented by data of n-1. We can then compress the dataset from D dimension to D/2 dimension.

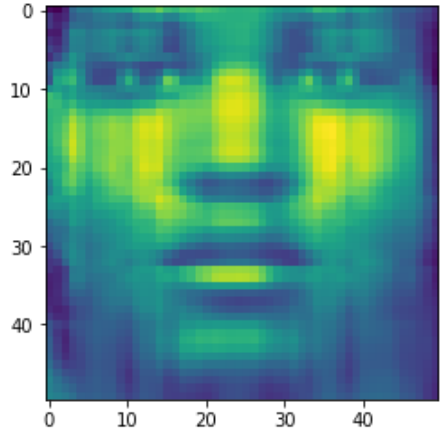
Question 5

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
# The following codes show the compressed images for face.png
img = mpimg.imread('/content/drive/My Drive/face.png')[:, :, 0]
Ks = [3, 5, 10, 30, 50, 100]
pca = PCA()
pca.fit(img)
cov = pca.get_covariance()
eig_val,eig_vec = np.linalg.eig(cov)
eig_pairs = [(np.abs(eig_val[i]),eig_vec[:,i]) for i in range(50)]
eig_pairs.sort(reverse=True)

for k in Ks:
    w = np.array([ele[1] for ele in eig_pairs[:k]])
    y = np.dot(img, np.transpose(w))
    x_new = np.dot(y, w)
    print("current k number is:", k)
    plt.imshow(x_new)
    plt.show()
```

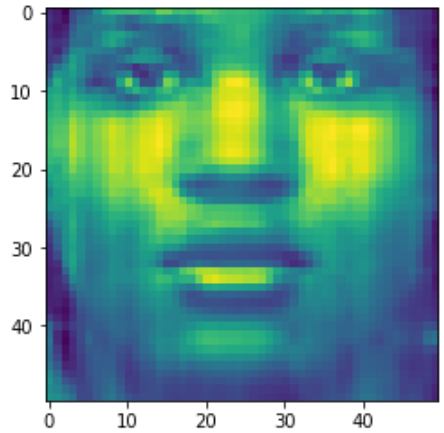
current k number is: 3



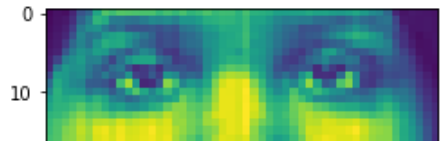
current k number is: 5

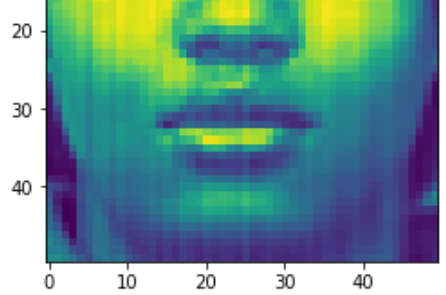


current k number is: 10

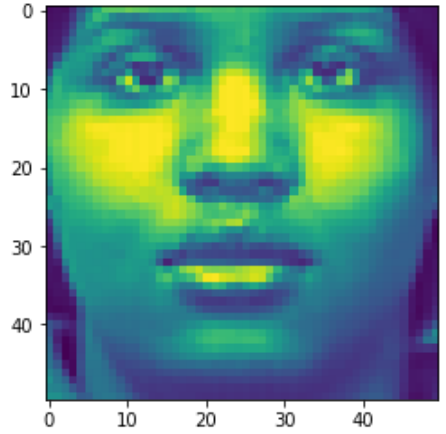


current k number is: 30

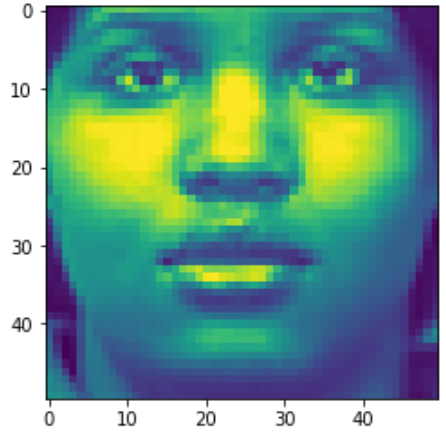





```
current k number is: 50
```

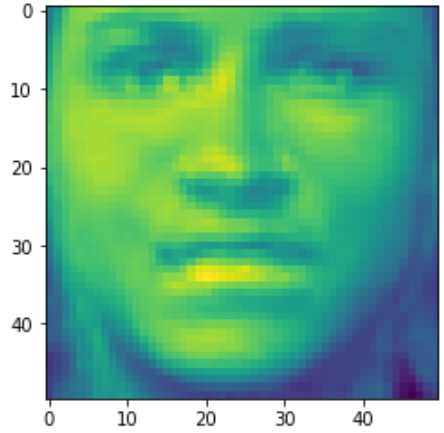



```
current k number is: 100
```

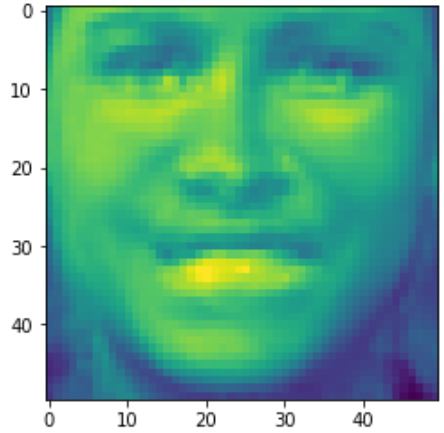


```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import numpy as np
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")
# The following codes generates among 100 faces
imgs = []
pixel_1 = []


for i in range (100):
    imgs.append(mpimg.imread('/content/drive/My Drive/Faces/face_{}.png'.format(i),1))
imgs = np.asarray(imgs)
pixel = imgs.reshape(100, 2500)
mean = np.mean(pixel, axis = 0)
new_p = pixel - mean
cov = np.dot(np.transpose(new_p), new_p)
eig_val,eig_vec = np.linalg.eig(cov)
eig_inx = np.argsort(-eig_val)
Ks = [3, 5, 10, 30, 50, 100]



# The follow for-loop shows the compressed imgae for face_9
for k in Ks:
    w = eig_vec[:, eig_inx[:k]]
    y = pixel[9].dot(w)
    x = np.dot(y, w.T)
    x_new = np.array(x, dtype = np.float64)
    x_n = x_new.reshape(50, 50)
    print("current k number is:", k)
    plt.imshow(x_n)
    plt.show()
```

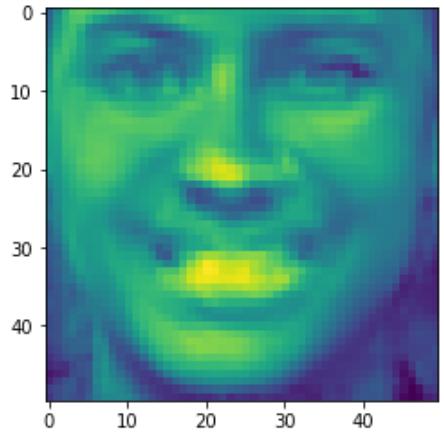
```
current k number is: 3
```



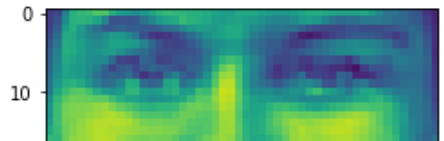

```
current k number is: 5
```

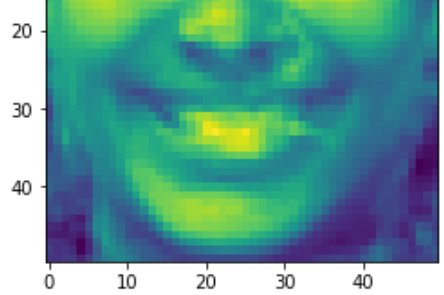



```
current k number is: 10
```

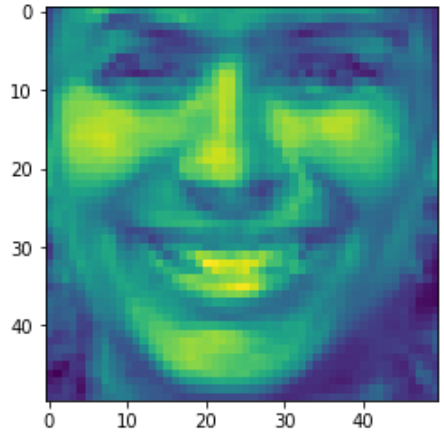



```
current k number is: 30
```

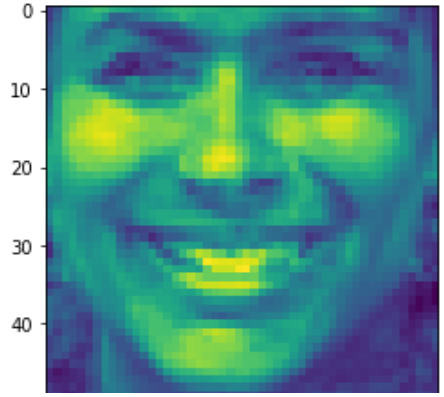





```
current k number is: 50
```




```
current k number is: 100
```



Question 6

If the original images are 64-bit float, the compression rate function will be:

$$\text{Compression rate} = \frac{log_2 k \times \text{number of pixel} + k \times 64}{64 \times \text{number of pixel}}$$

```
import numpy as np
from tabulate import tabulate
```

```
k = [3, 5, 10, 30, 50, 100]
log = np.log2(k)
pixel = 50*50
CR = (log*pixel + np.dot(k, 64))/(64*pixel)
info = {'K': k, 'CR (original images 64-bit float)': CR}
print(tabulate(info, headers='keys', tablefmt='fancy_grid'))
```

K	CR (original images 64-bit float)
3	0.025965
5	0.0382801
10	0.0559051
30	0.0886702
50	0.108185
100	0.14381

If the original images are 32-bit float, the compression rate function will be:

$$\text{Compression rate} = \frac{log_2 k \times \text{number of pixel} + k \times 64}{32 \times \text{number of pixel}}$$

```
import numpy as np
from tabulate import tabulate
k = [3, 5, 10, 30, 50, 100]
log = np.log2(k)
pixel = 50*50
CR = (log*pixel + np.dot(k, 64))/(32*pixel)
info = {'K': k, 'CR (original images 32-bit float)': CR}
print(tabulate(info, headers='keys', tablefmt='fancy_grid'))
```

K	CR (original images 32-bit float)
3	0.0519301
5	0.0765603
10	0.11181
30	0.17734
50	0.216371

100	0.287621

Question 7

The "elbow" rule is an intuitive method for selecting the optimal number of clusters for K-means clustering. By drawing an line chart, whose x-axis is a range of K values y-axis is their corresponding distortion score, we observe a curve can be either up or down. The point("elbow") of inflection on this curve is the best value of K.

Question 8

Since the results of K-means algorithm can be affected by its initialization, it's necessary to improve the method when selecting original centroids rather than randomly.

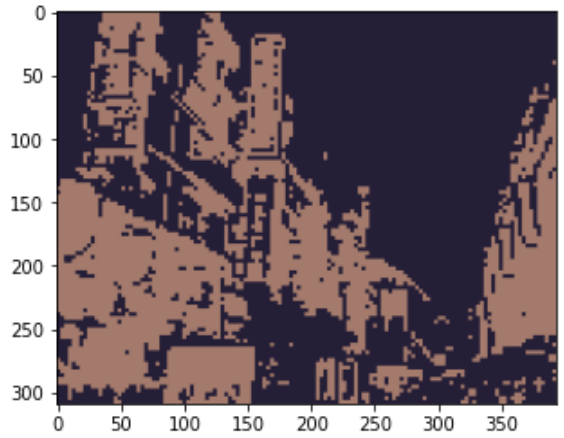
K means++ is based on the idea of selecting centroids as far away from each other as possible. Specifically speaking, we first select c1 as a centroid randomly like K-means, but then calculate all the distance from each point to c1 as D(i). The bigger D(i) is, the higher probability the point can be selected as the next centroid. Repeat this step until find K centroids and finish the initialization.

Question 9

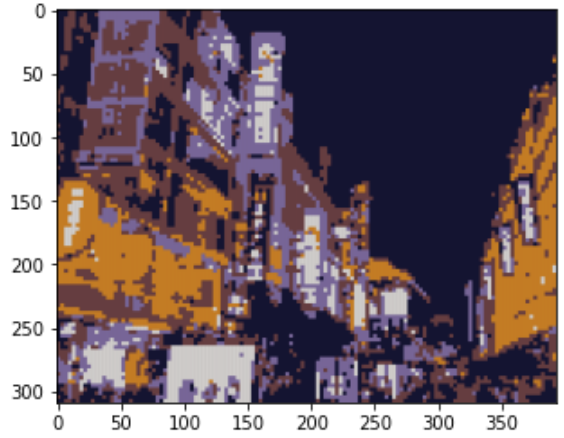
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.cluster import KMeans
import numpy as np
pic = mpimg.imread("/content/drive/My Drive/shopping-street.jpg")



newPic = [[0] * 3 for i in range(3) ]
clusterList = [2,5,10,25,50,100,200,1000]
row_number = 103
col_number = 131
h = 309
w = 393

block_row = np.array_split(pic, row_number, axis = 0)
img_blocks = []
```

```
for block in block_row:
  block_col = np.array_split(block, col_number, axis = 1)
  img_blocks += [block_col]

pixel_1 = []



for i in range(103):
     for j in range(131):
        pixel_1.append(np.concatenate(img_blocks[i][j][0:3], axis = None))
pixel = np.stack(pixel_1, axis = 0)




for cl in clusterList:
  print("current cluster number is:", cl)
  kmeans = KMeans(n_clusters = cl, random_state = 0).fit(pixel)
  cluster_assignments = kmeans.predict(pixel)
  cluster_centers = kmeans.cluster_centers_
  compressed_img = np.zeros((h, w, 3), dtype=np.uint8)
  pixel_count = 0




for i in range(103):
  for j in range(131):
      cluster_idx = cluster_assignments[pixel_count]
      cluster_value = cluster_centers[cluster_idx]
      cluster = cluster_value.reshape(3,3,3)
      for k in range(3):
          compressed_img[i*3:(i+1)*3, j*3:(j+1)*3] = cluster[k]
      pixel_count += 1

plt.imshow(compressed_img)
plt.show()
```

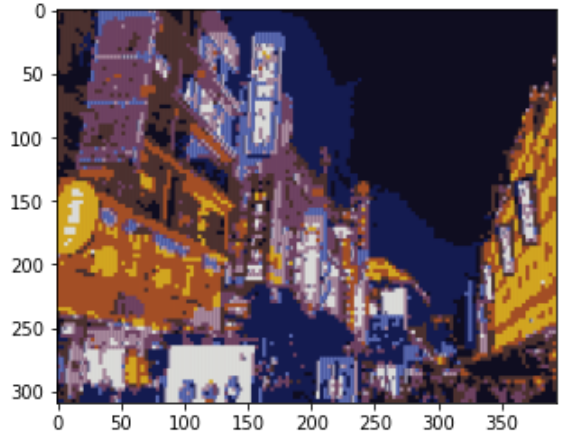
```
current cluster number is: 2
```



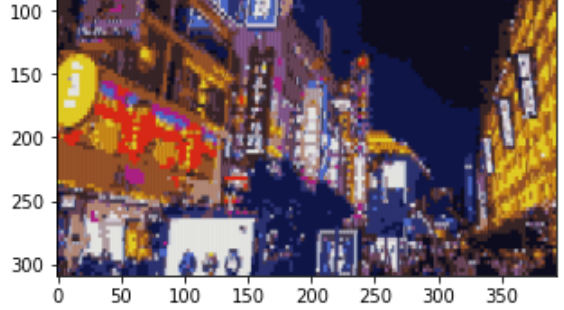

```
current cluster number is: 5
```



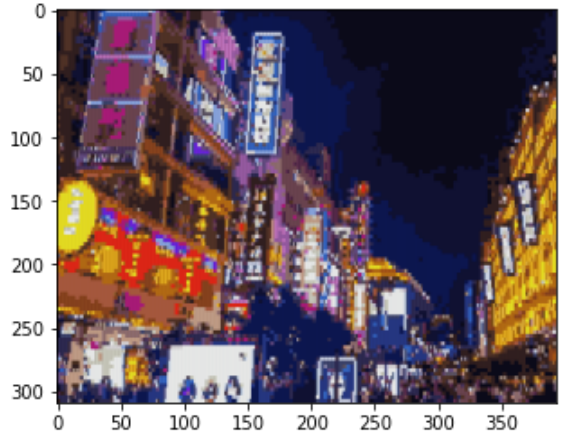

```
current cluster number is: 10
```



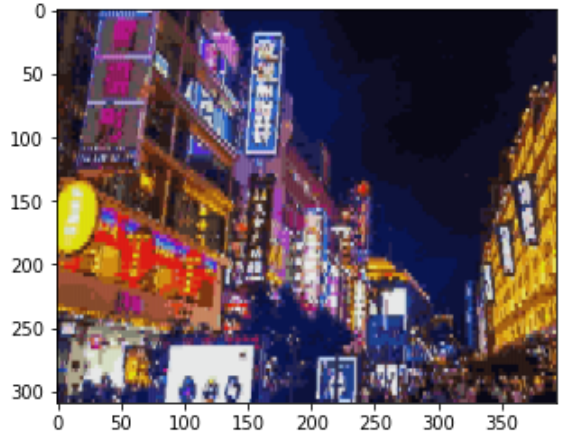

```
current cluster number is: 25
```



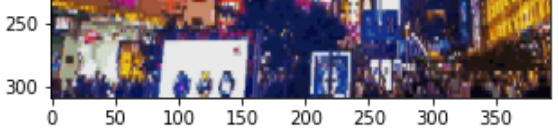
current cluster number is: 50



current cluster number is: 100



current cluster number is: 200




```
current cluster number is: 1000
```

Question 10

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.cluster import KMeans
import numpy as np
from tabulate import tabulate

pic = mpimg.imread("/content/drive/My Drive/shopping-street.jpg")
newPic = [[0] * 3 for i in range(3) ]
clusterList = [2,5,10,25,50,100,200,1000]
errorList = []
row_number = 103
col_number = 131
h = 309
w = 393

block_row = np.array_split(pic, row_number, axis = 0)
img_blocks = []
for block in block_row:
  block_col = np.array_split(block, col_number, axis = 1)
  img_blocks += [block_col]

pixel_1 = []
for i in range(103):
     for j in range(131):
        pixel_1.append(np.concatenate(img_blocks[i][j][0:3], axis = None))
pixel = np.stack(pixel_1, axis = 0)
```

```
for cl in clusterList:
  # print("current cluster number is:", cl)
  kmeans = KMeans(n_clusters = cl, random_state = 0).fit(pixel)
  cluster_assignments = kmeans.predict(pixel)
  cluster_centers = kmeans.cluster_centers_
  compressed_img = np.zeros((h, w, 3), dtype=np.uint8)
  pixel_count = 0
  for i in range(103):
    for j in range(131):
        cluster_idx = cluster_assignments[pixel_count]
        cluster_value = cluster_centers[cluster_idx]
        cluster = cluster_value.reshape(3,3,3)
        for k in range(3):
            compressed_img[i*3:(i+1)*3, j*3:(j+1)*3] = cluster[k]
        pixel_count += 1
  err = np.sum((pic.astype("float") - compressed_img.astype("float")) ** 2)
  err /= float(pic.shape[0] * pic.shape[1])
  errorList.append(err)
  err = 0

info = {'cluster number': clusterList, 'reconstruction error': errorList}
print(tabulate(info, headers='keys', tablefmt='fancy_grid'))
```

cluster number	reconstruction error
2	8051.35
5	4646.26
10	3593.5
25	3099.87
50	2946.65
100	2796.55
200	2759.71
1000	2719.16

Question 11

The number original picture used is:

---

309 * 393 * 24 = 2,914,488

1. k = 2:

   log2(2) = 2

   309 * 393 * 1 = 121,437

   24 * 2 = 48

   121,437 + 48 = 121485

2. k = 5:

   log2(5) = 2.321928

   309 * 393 * 2.321928 = 281,968

   24 * 5 = 120

   281,968 + 120 = 282088

3. k = 10:

   log2(10) = 3.321928

   309 * 393 * 3.321928 = 403,405

   24 * 10 = 240

   403,405 + 240 = 403645

4. k = 25:

   log2(25) = 4.643856

   309 * 393 * 4.643856 = 563,936

   24 * 25 = 600

   563,936 + 600 = 564536

5. k = 50:

   log2(50) = 5.643856

   309 * 393 * 5.643856 = 685,373

   24 * 50 = 1,200

685,373 + 1,200 = 686573

6. k = 100:

   log2(100) = 6.643856

   309 * 393 * 6.643856 = 806,810

   24 * 100 = 2,400

   806,810 + 2,400 = 809210

7. k = 200:

   log2(200) = 7.643856

   309 * 393 * 7.643856 = 928,247

   24 * 200 = 4,800

   928,247 + 4,800 = 933047

8. k = 1000: log2(1000) = 9.965784

   309 * 393 * 9.965784 = 1,210,215

   24 * 1000 = 24000

   1,210,215 + 24000 = 1234215

```
from tabulate import tabulate

clusterList = [2,5,10,25,50,100,200,1000]
totalNumbers = [121485,282088,403645,564536,686573,809210,933047,1234215]
info = {'K value': clusterList, 'total numbers': totalNumbers}
print(tabulate(info, headers='keys', tablefmt='fancy_grid'))
```

K value	total numbers
2	121485
5	282088
10	403645
25	564536
50	686573

100	809210
200	933047
1000	1234215

Question 12

The number original picture used is: 309 * 393 * 24 = 2,914,488

Compression Rate(k = 2) = 121485 / 2,914,488 = 0.04

Compression Rate(k = 5) = 282088 / 2,914,488 = 0.1

Compression Rate(k = 10) = 403645 / 2,914,488 = 0.14

Compression Rate(k = 25) = 564536 / 2,914,488 = 0.19

Compression Rate(k = 50) = 686573 / 2,914,488 = 0.24

Compression Rate(k = 100) = 809210 / 2,914,488 = 0.28

Compression Rate(k = 200) = 933047 / 2,914,488 = 0.32

Compression Rate(k = 1000) = 1234215 / 2,914,488 = 0.42

```
from tabulate import tabulate

clusterList = [2,5,10,25,50,100,200,1000]
compressionRate = [0.04,0.1,0.14,0.19,0.24,0.28,0.32,0.42]
info = {'K value': clusterList, 'compression rate': compressionRate}
print(tabulate(info, headers='keys', tablefmt='fancy_grid'))
```

K value	compression rate
2	0.04
5	0.1
10	0.14
25	0.19
50	0.24
100	0.28

200	0.32
1000	0.42