Q1

Find first and second principal components. Explain your solution step by step.

```
import numpy as np
import pandas as pd
# inputting the matrix x1 x2 - inserting the data set
A = np.matrix([[2.2,2.1],
               [0.4,0.6],
               [1.6,2.3],
               [2,2.1],
               [2.9,2.8],
               [0.8,1.1],
               [1.3,1.4],
               [1.1,1],
               [2.3,2.6]])

dataf = pd.DataFrame(A,columns = ['x1','x2'])
dataf

# standardize the dataset
df_standardized = (dataf - dataf.mean()) / (dataf.std())
df_standardized

# find the covariance (population) matrix for the given data set
dataf_cov = np.cov(df_standardized.T,bias = 1)
dataf_cov

# verify var(x1)
print('var(f1) (population formula): ',((df_standardized.x1)**2).sum()/9)

# verify cov(x1,x2)
print('cov(f1,f2) (population formula): ',((df_standardized.x1)*(df_standardized.x2

# calculate evalue and evector
eigen_val, eigen_vectors = np.linalg.eig(dataf_cov)
print("Eig-val:\n",eigen_val)
print("Eig-vec:\n",eigen_vectors)

# sort the e values and corresponding e vectors
number_of_components = 2
top_eigen_vectors = eigen_vectors[:,:number_of_components]
top_eigen_vectors.shape
np.array(df_standardized).shape
transformed_data = np.matmul(np.array(df_standardized),top_eigen_vectors)
pd.DataFrame(data = transformed_data
             , columns = ['Principal component '+ str(i+1) for i in range(number_of
transformed_data.shape
```

```
# using PCA in sklearn
from sklearn.decomposition import PCA
# define number of components
pca = PCA(n_components=number_of_components)
# transform standardized data set
principalComponents = pca.fit_transform(df_standardized)
# define the pricipal components data frame
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['Principal Component '+ str(i+1) for i in range(number_of
# print the principal components data frame
principalDf
```

```
var(f1) (population formula):  0.8888888888888888
cov(f1,f2) (population formula):  0.8406973999676008
Eig-val:
 [0.04819149 1.72958629]
Eig-vec:
 [[-0.70710678 -0.70710678]
 [ 0.70710678 -0.70710678]]
```

	Principal Component 1	Principal Component 2
0	-0.805194	-0.216626
1	2.156428	0.005114
2	-0.457292	0.496593
3	-0.628340	-0.039773
4	-2.063486	-0.196308
5	1.346075	0.108054
6	0.629953	-0.060092
7	1.172124	-0.248556
8	-1.350267	0.151594

Q2

Write a Python script that calculates approximations of an image using Singular Value Decomposition.

```
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
```

```
# import image as image.jpg
img = cv2.imread('image.jpg',0)
```

```
from numpy.linalg import svd


# k = 1,5,10,20,40
# create a methof to approximate image with various values for k
def approxi(img,k):
  U,s,V = svd(img,full_matrices = False)
  reconstructedMatrix = np.dot(U[:,:k],np.dot(np.diag(s[:k]),V[:k,:]))
  return reconstructedMatrix,s


# k = 1
print("k=1")
mat1 = approxi(img,1)
print(mat1)

# k = 5
print("k=5")
mat5 = approxi(img,5)
print(mat5)

# k = 10
print("k=10")
mat10 = approxi(img,10)
print(mat10)

# k = 20
print("k=20")
mat20 = approxi(img,20)
print(mat20)

# k = 40
print("k=40")
mat40 = approxi(img,40)
print(mat40)
```

```
        [194.23151582, 194.35247381, 194.3993644 , ...,  55.2900777 ,
          55.42310729,  55.58840207],
        [193.60001196, 193.71906289, 193.76618699, ...,  55.1705606 ,

          55.3059179 ,  55.46935284],
        ...,
        [119.72133701, 119.76887615, 119.77970202, ...,  51.17632091,
          51.25782013,  51.36332498],
        [120.0475878 , 120.09518647, 120.10591302, ...,  50.99376397,
          51.07533356,  51.18164575],
        [120.55535371, 120.60450333, 120.61573836, ...,  50.94964741,
          51.0307033 ,  51.13744593]]), array([1.96358642e+05, 3.82897607e+04,
        5.37743999e+00, 5.29537110e+00, 5.11965639e+00]))
k=10
(array([[191.86504485, 191.9599801 , 191.99134879, ...,  53.36913961,
          53.27068826,  53.31474643],
        [192.22803615, 192.31553133, 192.3398733 , ...,  53.28161536,
          53.16357517,  53.19448081],
        [192.03430799, 192.11647829, 192.13591917, ...,  53.43742188,
          53.30474079,  53.32480385],
        ...,
        [131.32637939, 131.39087393, 131.43783377, ...,  58.01641972,
```

```
          58.09655212,  58.18478243],
        [131.01943631, 131.08308502, 131.12787955, ...,  57.82616177,
          57.90501406,  57.99343717],
        [130.5778039 , 130.64223459, 130.68605553, ...,  57.42156969,
          57.50152186,  57.5911883 ]]), array([1.96358642e+05, 3.82897607e+04,
        5.37743999e+00, 5.29537110e+00, 5.11965639e+00]))
k=20
(array([[186.04556562, 185.96646006, 185.84782509, ...,  40.10911694,
          40.16997758,  40.44031193],
        [185.39834214, 185.28243502, 185.12784608, ...,  40.35205811,
          40.40033171,  40.67329799],
        [185.06149251, 184.92427185, 184.74732627, ...,  40.60192927,
          40.63807898,  40.910929  ],
        ...,
        [150.22824854, 150.30810668, 150.34828929, ...,  56.35300726,
          56.45438115,  56.50816184],
        [150.30960908, 150.38961161, 150.42932713, ...,  55.992642  ,
          56.0950985 ,  56.15119544],
        [150.64687335, 150.72525883, 150.76293792, ...,  55.6733736 ,
          55.77002731,  55.82680655]]), array([1.96358642e+05, 3.82897607e+04,
        5.37743999e+00, 5.29537110e+00, 5.11965639e+00]))
k=40
(array([[174.18713935, 174.09540178, 174.0488977 , ...,  44.69437725,
          44.90535326,  45.34361985],
        [173.2170004 , 173.12722337, 173.07138564, ...,  44.66684437,
          44.89504374,  45.38153128],
        [173.18547844, 173.10002939, 173.04021173, ...,  44.74191749,
          44.97489825,  45.48709844],
        ...,
        [158.16046399, 158.19906625, 158.28456149, ...,  56.05659885,
          55.954847  ,  55.76736066],
        [158.45176595, 158.47992734, 158.55049821, ...,  55.93146475,
          55.82549256,  55.63311204],
        [158.5253082 , 158.54797782, 158.60354239, ...,  55.77152053,
          55.65654732,  55.45329548]]), array([1.96358642e+05, 3.82897607e+04,
        5.37743999e+00, 5.29537110e+00, 5.11965639e+00]))
```

✓ 13s completed at 22:39