Bahria University, Islamabad Campus

Department of Computer Sciences

Artificial Intelligence

Assignment-3

Class: BSCS-6A/6B

Deadline: 19th December 2021

Name	**Enrollment**
Tauheed Butt	01-134191-068
Mubeen Ahmed	01-134191-018

Objectives:

To understate the concept of dimensionality reduction by applying the Principal Component Analysis (PCA) Technique. PCA reduces high dimensional data to lower dimensions while capturing maximum variability of the dataset. The main steps of the PCA Algorithm are as follows:

1. Compute mean of each variable and subtract it from the dataset to centre data on the origin.
2. Calculate the Covariance Matrix.
3. Compute the Eigenvalues and the Eigenvectors.
4. Sort Eigenvalues in descending order.
5. Select a subset from the rearranged Eigenvalue matrix (i.e. number of components e.g. P=2).
6. Transform the data by computing a dot product between the **Transpose** of the Eigenvector subset and the **Transpose** of the mean-centered data. By transposing the outcome of the dot product, the result will be the data reduced to lower dimensions.

Task-1:

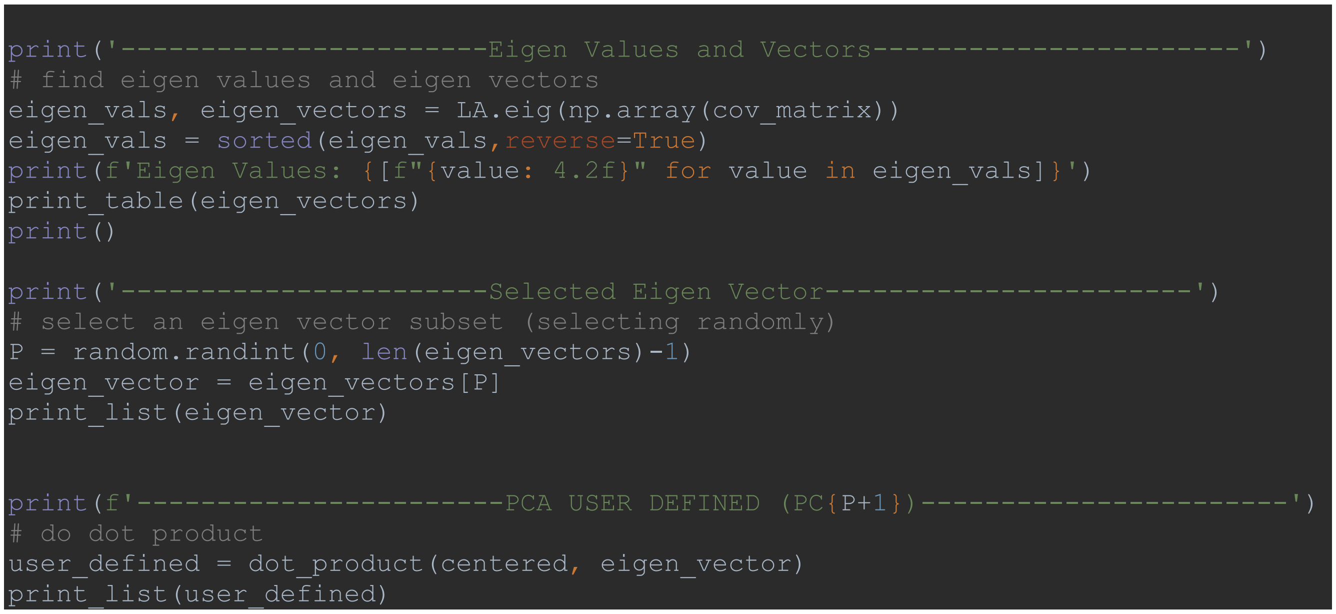
Implement each step of the PCA algorithm from scratch and apply it on either an existing dataset like “IRIS” or “MNIST” etc., or generate a dummy data by using the given command (it generates 6 variables for 10 samples, you can change as per your requirement).

```
X = np.random.randint(10,50,100).reshape(10,6)
```

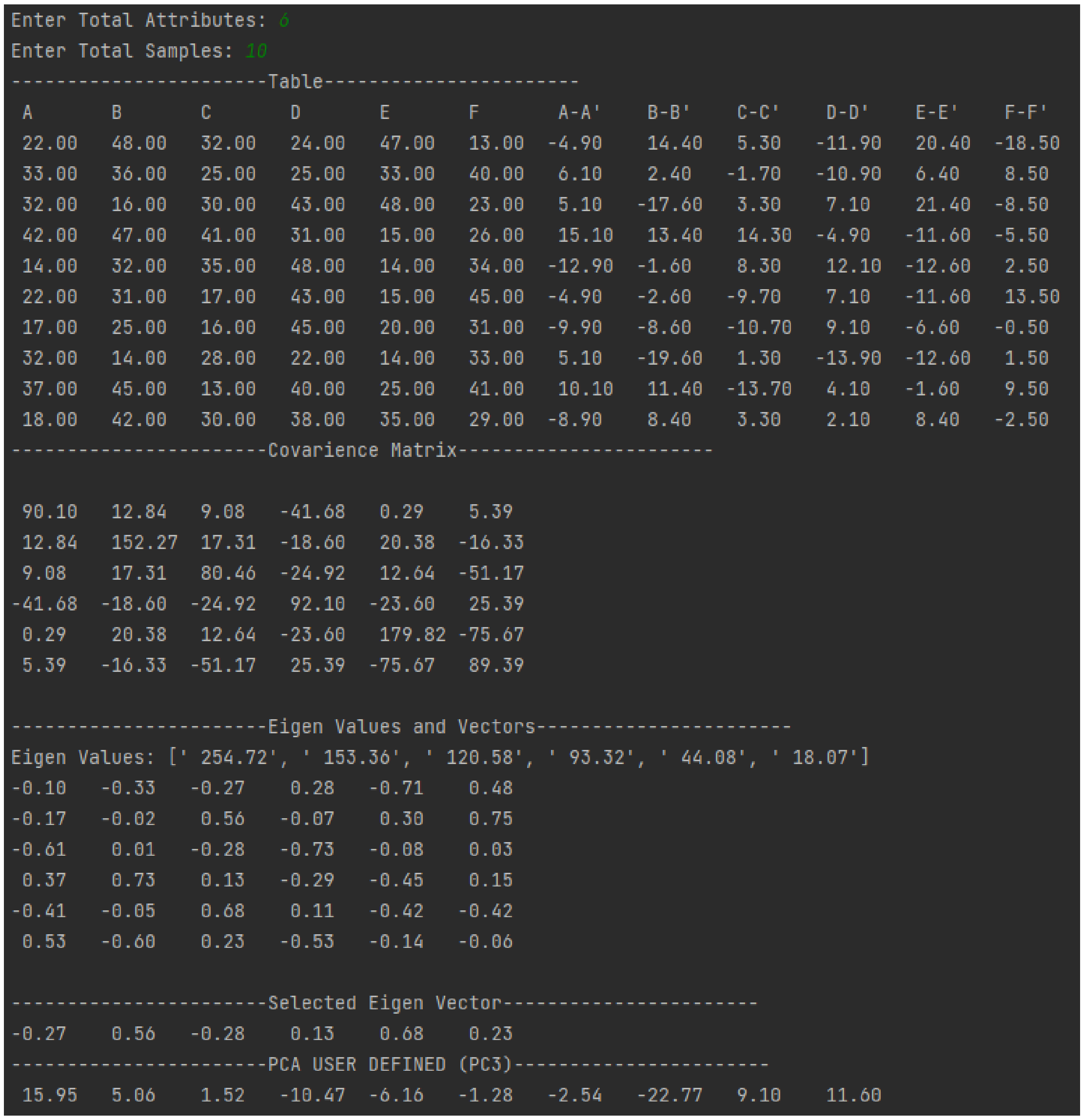
*NOTE: Do not use inbuilt function PCA() for this task, however, you can use inbuilt functions for implementing the steps. Show the output of each step as well.

```
from numpy import linalg as LA
import numpy as np
import random

def print_table(table):
    """

    :param table: a 2D matrix
    :return: none
    """
    for i in range(len(table[0])):
        for j in range(len(table)):
            print(f'{table[j][i]: 4.2f}', end='\t')
        print()


def print_list(List):
    """
    :param List: a 1D Matrix
    :return:
    """
    for item in List:
        print(f'{item: 4.2f}', end='\t')
    print()


def get_means(table):
    """
    :param table: a 2D Matrix
    :return: mean of all the attributes in a list
    """
    means = []
    for row in table:
        means.append(sum(row)/len(row))
    return means


def means_centered(table, means):
    """
    :param table: a 2D Matrix
    :param means: mean of all the attributes in a list
    :return: a list of attribute - their mean
    """
    centered = []
    for i in range(len(table)):
        centered.append([item-means[i] for item in table[i]])
    return centered

def cov(var1, var2):
    """
    :param var1: mean-centered values of attribtue 1
```

```python
    :param var2: mean-centered values of attribtue 2
    :return: covariance of attribute 1 and attribute 2
    """
    prod = [var1[i]*var2[i] for i in range(len(var1))]
    return (sum(prod)/(len(var1)-1))

def get_cov_matrix(centered):
    """
    :param centered: the centered values (X - X')
    :return: covarience matrix
    """
    attributes = len(centered)
    cov_matrix = [[0 for i in range(attributes)] for i in range(attributes)]
    for i in range(attributes):
        for j in range(attributes):
            cov_matrix[i][j] = cov(centered[i], centered[j])
    return cov_matrix

def dot_product(centered, eigen_vector):
    """

    :param centered: the centered values (X - X')
    :param eigen_vector: eigen vector
    :return: reduced values
    """
    reduced = []
    for i in range(len(centered[0])):
        set = [centered[j][i] for j in range(len(centered))]
        reduced.append(np.dot(np.array(set), np.array(eigen_vector)))
    return reduced


attributes = int(input('Enter Total Attributes: '))
samples = int(input('Enter Total Samples: '))

# attributes = 6
# samples = 10

# generating table
table = list(np.random.randint(10, 50, attributes*samples).reshape(attributes,
samples))

# calculate mean
means = get_means(table)

# subtract mean from each attribute
centered = means_centered(table, means)

# add subtracted values in the table
for item in centered:
    table.append(item)

print('-----------------------Table-----------------------')
print(f" A\t\t B\t\t C\t\t D\t\t E\t\t F\t\t A-A'\t B-B'\t C-C'\t D-D'\t E-E'\t F-
F'")
print_table(table)

# calculate covariance matrix
print('-----------------------Covarience Matrix-----------------------')
cov_matrix = get_cov_matrix(centered)
print()
print_table(cov_matrix)
print()
```



```
print('-----------------------Eigen Values and Vectors-----------------------')
# find eigen values and eigen vectors
eigen_vals, eigen_vectors = LA.eig(np.array(cov_matrix))
eigen_vals = sorted(eigen_vals,reverse=True)
print(f'Eigen Values: {[f"{value: 4.2f}" for value in eigen_vals]}')
print_table(eigen_vectors)
print()

print('-----------------------Selected Eigen Vector-----------------------')
# select an eigen vector subset (selecting randomly)
P = random.randint(0, len(eigen_vectors)-1)
eigen_vector = eigen_vectors[P]
print_list(eigen_vector)


print(f'-----------------------PCA USER DEFINED (PC{P+1})-----------------------')
# do dot product
user_defined = dot_product(centered, eigen_vector)
print_list(user_defined)
```

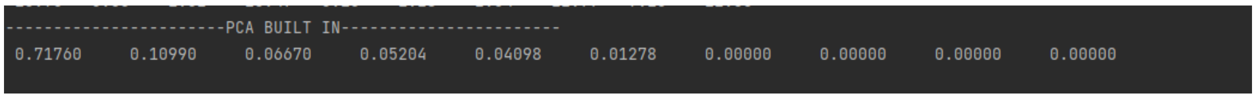


```
Enter Total Attributes: 6
Enter Total Samples: 10
-----------------------Table-----------------------
 A       B       C       D       E       F       A-A'    B-B'    C-C'    D-D'    E-E'    F-F'
 22.00   48.00   32.00   24.00   47.00   13.00  -4.90    14.40   5.30   -11.90   20.40  -18.50
 33.00   36.00   25.00   25.00   33.00   40.00   6.10    2.40   -1.70   -10.90   6.40    8.50
 32.00   16.00   30.00   43.00   48.00   23.00   5.10   -17.60   3.30    7.10    21.40  -8.50
 42.00   47.00   41.00   31.00   15.00   26.00   15.10   13.40   14.30  -4.90   -11.60  -5.50
 14.00   32.00   35.00   48.00   14.00   34.00  -12.90  -1.60    8.30    12.10  -12.60   2.50
 22.00   31.00   17.00   43.00   15.00   45.00  -4.90   -2.60   -9.70    7.10   -11.60   13.50
 17.00   25.00   16.00   45.00   20.00   31.00  -9.90   -8.60   -10.70   9.10   -6.60   -0.50
 32.00   14.00   28.00   22.00   14.00   33.00   5.10   -19.60   1.30   -13.90  -12.60   1.50
 37.00   45.00   13.00   40.00   25.00   41.00   10.10   11.40  -13.70   4.10   -1.60    9.50
 18.00   42.00   30.00   38.00   35.00   29.00  -8.90    8.40    3.30    2.10    8.40   -2.50
-----------------------Covarience Matrix-----------------------


 90.10   12.84   9.08   -41.68   0.29    5.39
 12.84   152.27  17.31  -18.60   20.38  -16.33
 9.08    17.31   80.46  -24.92   12.64  -51.17
-41.68  -18.60  -24.92   92.10  -23.60   25.39
 0.29    20.38   12.64  -23.60   179.82 -75.67
 5.39   -16.33  -51.17   25.39  -75.67   89.39

-----------------------Eigen Values and Vectors-----------------------
Eigen Values: [' 254.72', ' 153.36', ' 120.58', ' 93.32', ' 44.08', ' 18.07']
-0.10   -0.33   -0.27    0.28   -0.71    0.48
-0.17   -0.02    0.56   -0.07    0.30    0.75
-0.61    0.01   -0.28   -0.73   -0.08    0.03
 0.37    0.73    0.13   -0.29   -0.45    0.15
-0.41   -0.05    0.68    0.11   -0.42   -0.42
 0.53   -0.60    0.23   -0.53   -0.14   -0.06

-----------------------Selected Eigen Vector-----------------------
-0.27    0.56   -0.28    0.13    0.68    0.23
-----------------------PCA USER DEFINED (PC3)-----------------------
 15.95   5.06    1.52   -10.47  -6.16   -1.28   -2.54   -22.77   9.10    11.60
```

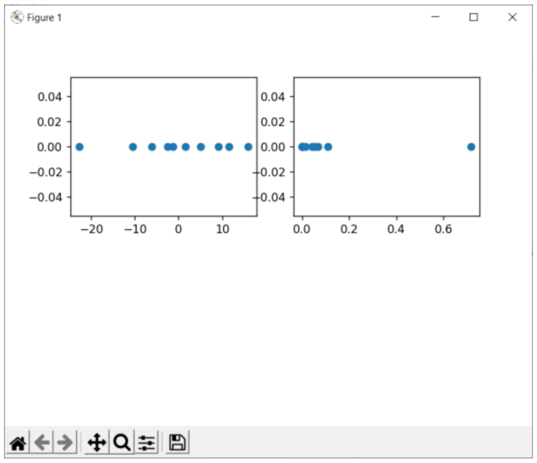
Task-2:

In the second task, use the same dataset employed in the previous task and apply the inbuilt function **PCA()** on it. Visualize the final output of both tasks to compare the working of the user-defined and inbuilt PCA functions.

```
from task_1 import *
import matplotlib.pyplot as plt

import numpy as np
from sklearn.decomposition import PCA


print(f'-----------------------PCA BUILT IN-----------------------')
pca = PCA(n_components=samples)
pca.fit(table)
print('\t'.join([f'{item: 4.5f}' for item in pca.explained_variance_ratio_]))

built_in = pca.explained_variance_ratio_

# display graph
fig = plt.figure()
sub1 = fig.add_subplot(221)
sub2 = fig.add_subplot(222)

sub1.scatter(user_defined, np.linspace(0, 0, samples))
sub2.scatter(built_in, np.linspace(0, 0, samples))
plt.show()
```



```
----------------------PCA BUILT IN----------------------
 0.71760     0.10990     0.06670     0.05204     0.04098     0.01278     0.00000     0.00000     0.00000     0.00000
```



Submission Requirements:

This is a group assignment, but no more than two members are allowed in a group. The same groups should be made as were in the previous assignment. **Python code** should be **executable** (should be copy-pasted from source file and not included as screen shot). **Screen shots of output** should be included. All by-hand work should be neat and comprehensible. Upload softcopies in LMS individually, however, group member names should be clearly mentioned in each assignment.

Submission Deadline:

19th December 2021