IE 500: Introduction to Deep Learning for Engineers

Assignment 1: Image Compression

Question 1.

Code

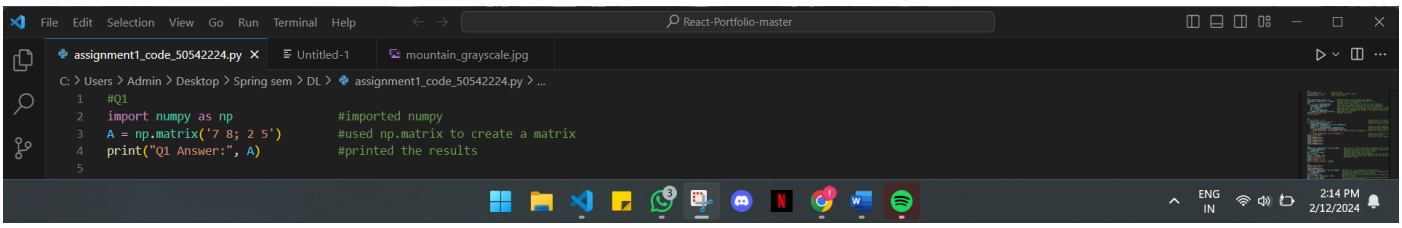

Figure 1: Question 1 code

Output

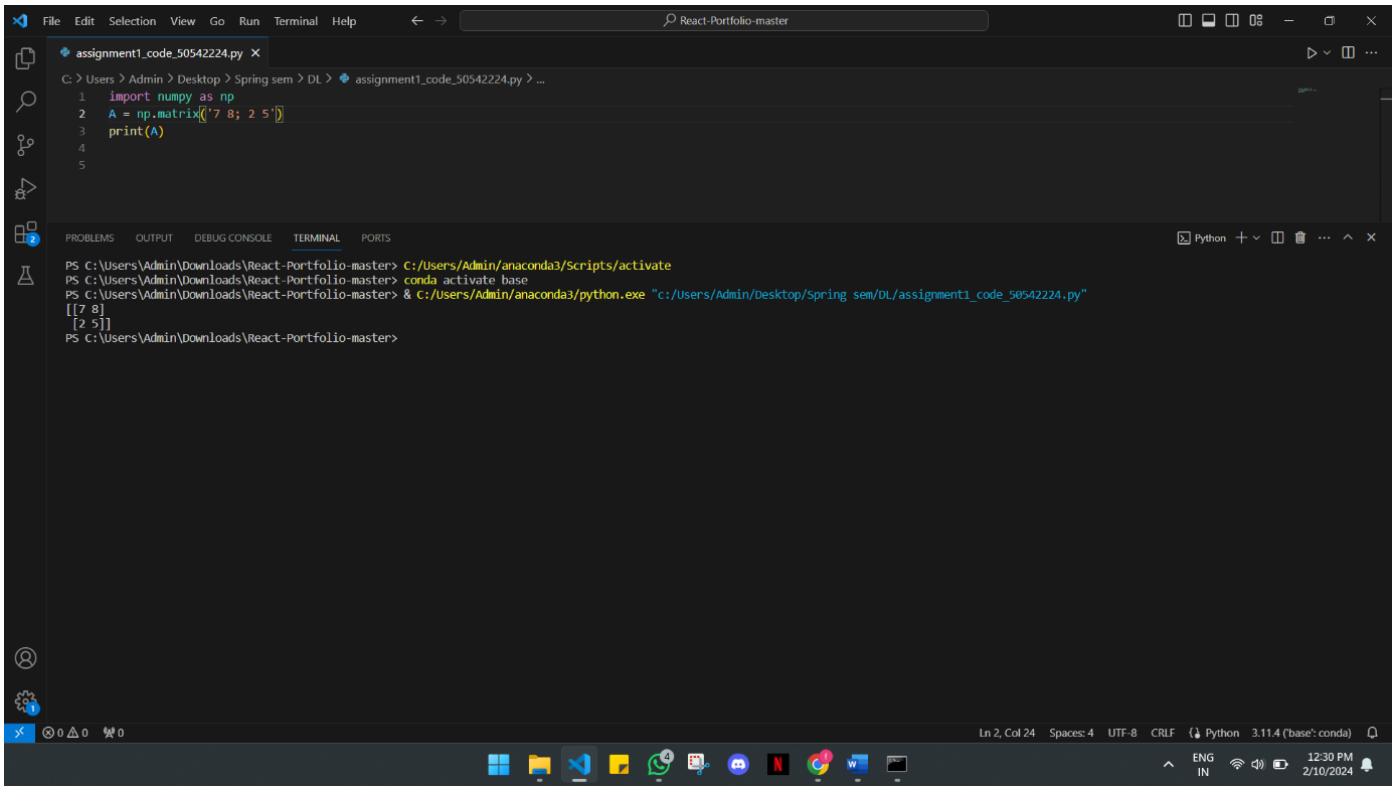

Figure 2: Question 1 output

Question 2.

Code

```python
#Q2
from numpy import linalg as LA          #imported module callled linalg(linear algebra)
eigenvalues, eigenvectors = LA.eig(A)   #used LA.eig() to find eigenvalues and eigenvectors
a=[]                                    #created an empty list called 'a'
for i in range (len(eigenvalues)):      #initiated a for loop that will iterated over each eigenvalue
    eigenvalue = eigenvalues[i]         #save the corresponding eigenvalues to the variable 'eigenvalue'
    xi = eigenvectors[:,i]              #this line extracts the i-th eigenvector from the eigenvectors, [:,i] will select all rows and ith column
    LHS = A.dot(xi)                     #performed matrix multiplication between A and xi and set value to LHS
    RHS = xi.dot(eigenvalue)            #performed matrix multiplication between xi and eigenvalue and set value to RHS
    a.append((xi,eigenvalue,LHS,RHS))   #appended all values to a
print("\n Q2 Answer:", a)                #printed the results
# print(type(a))
```

Figure 3: Question 2 code

Output

```python
#Q1
import numpy as np
A = np.matrix('7 8; 2 5')
print("Q1 Answer:", A)

#Q2
from numpy import linalg as LA
eigenvalues, eigenvectors = LA.eig(A)
a=[]
for i in range (len(eigenvalues)):
    eigenvalue = eigenvalues[i]
    xi = eigenvectors[:,i]
    LHS = A.dot(xi)
    RHS = xi.dot(eigenvalue)
    a.append((xi,eigenvalue,LHS,RHS))
print("Q2 Answer:", a)
```

```
Q2 Answer: [(matrix([[0.93153209],
        [0.36365914]]), 10.123105625617661, matrix([[9.42999772],
        [3.68135987]]), matrix([[9.42999772],
        [3.68135987]])), (matrix([[-0.84212294],
        [ 0.5392856 ]]), 1.8768943743823394, matrix([[-1.58057581],
        [ 1.0121821 ]]), matrix([[-1.58057581],
        [ 1.0121821 ]]))]
PS C:\Users\Admin\Downloads\React-Portfolio-master>
```

Figure 4: Question 2 output

Question 3.

Code

```python
#Q3
B = np.matrix('0 1; 1 0')                                   #used np.matrix to create matrix 'B'
C = np.matrix('2 3 0; 3 4 1; 0 1 9')                        #used np.matrix to create matrix 'C'
def check_orthogonality(matrix):                            #created a function to check if eigenvectors are orthogonal
    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    eigenvectors = eigenvectors.T                           #using '.T' took transpose of eigenvectors
    product = np.dot(eigenvectors, eigenvectors.T)          #performed matrix multiplication between eigenvectors and eigenvectors.T and set value
    if np.allclose(product, np.eye(len(eigenvalues))):      #used an if loop to check if the matrix is identity matrix or not. np.eye(len(eigenvalu
        print("Eigenvectors are orthogonal, hence the given matrix is symmetric.")  #prints if the matrix is close to identity matrix. ie, symmetric.
    else:
        print("Eigenvectors are not orthogonal.")           #else prints the matrix is not symmetric.
print("\n Q3 Answer:")                                      #printed the answers
print("Matrix A:")
check_orthogonality(A)                                      #called the function check_orthogonality( ), and passed matrix A
print("\nMatrix B:")
check_orthogonality(B)
print("\nMatrix C:")
check_orthogonality(C)
```

Figure 5: Question 3 code

Output

```python
def check_orthogonality(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)
    eigenvectors = eigenvectors.T
    product = np.dot(eigenvectors, eigenvectors.T)
    if np.allclose(product, np.eye(len(eigenvalues))):
        print("Eigenvectors are orthogonal, hence the given matrix is symmetric.")
    else:
        print("Eigenvectors are not orthogonal.")

print("\n Q3 Answer:")
print("Matrix A:")
check_orthogonality(A)
print("\nMatrix B:")
check_orthogonality(B)
print("\nMatrix C:")
check_orthogonality(C)
```

```
 Q3 Answer:
Matrix A:
Eigenvectors are not orthogonal.

Matrix B:
Eigenvectors are orthogonal, hence the given matrix is symmetric.

Matrix C:
Eigenvectors are orthogonal, hence the given matrix is symmetric.
PS C:\Users\Admin\Downloads\React-Portfolio-master>
```

Figure 6: Question 3 output

If the **product of** the given matrix and transpose of the same matrix is an **identity matrix**, then it is an orthogonal matrix. Initially defined a function called check orthogonality () and multiplied the given matrix and transpose. And using the if condition I checked if it was close to the Identity matrix. If yes, it is orthogonal, or else it's not. I used the same logic to find out if the matrix was orthogonal or not. So, for the given three matrices, only B & and C are orthogonal and these matrices are symmetric.

Question 4.

Code

```python
#Q4
eigenvalues, eigenvectors = np.linalg.eig(A)      #used LA.eig() to find eigenvalues and eigenvectors of corresponding matrixes
Λ = np.diag(eigenvalues)                          #np.diag() is a NumPy function that constructs a diagonal matrix of eigenvalues
Q = eigenvectors                                  #assigned eigenvectors to a variable called Q
Qinv = np.linalg.inv(Q)                           #using np.linalg.inv(Q) took inverse of Q
QΛ= np.dot(Q,Λ)                                   #performed matrix multiplication between Q and Λ and set value to QΛ
QΛQinv = np.dot(QΛ, Qinv)                         #performed matrix multiplication between QΛ and Qinv and set value to QΛQinv
print("\n Q4 Answer:")                            #printed the answers
print("\n Matrix Q:", Q)
print("\n Matrix Λ:", Λ)
print("\n Matrix QΛQ^{-1}:", QΛQinv)
```

Figure 7: Question 4 code

Output

```python
#Q4
eigenvalues, eigenvectors = np.linalg.eig(A)
Λ = np.diag(eigenvalues)
Q = eigenvectors
Qinv = np.linalg.inv(Q)
QΛ= np.dot(Q,Λ)
QΛQinv = np.dot(QΛ, Qinv)
print("\n Q4 Answer:")
print("\n Matrix Q:", Q)
print("\n Matrix Λ:", Λ)
print("\n Matrix QΛQ^{-1}:", QΛQinv)
```

```
Q4 Answer:

Matrix Q: [[ 0.93153209 -0.84212294]
 [ 0.36365914  0.5392856 ]]

Matrix Λ: [[10.12310563  0.        ]
 [ 0.          1.87689437]]

Matrix QΛQ^{-1}: [[7. 8.]
 [2. 5.]]
PS C:\Users\Admin\Downloads\React-Portfolio-master>
```

Figure 8: Question 4 output

Question 5.

Code

```python
#Q5
print("\n Q5 Answer:")
print("\nMatrix C:")
check_orthogonality(C)
eigenvalues, eigenvectors = np.linalg.eig(C)    #used LA.eig() to find eigenvalues and eigenvectors
Λ = np.diag(eigenvalues)                        #np.diag() is a NumPy function that constructs a diagonal matrix of eigenvalues as like L41
Q = eigenvectors                                #assigned eigenvectors to a variable called Q
QQT = np.dot(Q, Q.T)                            #performed matrix multiplication between Q and Q.T, which is transpose of Q and set value to QQT
print("\n Matrix QQTranspose:", QQT)            #printed the answer
Qinv = np.linalg.inv(Q)                         #using np.linalg.inv(Q) took inverse of Q and assigned it to Qinv
QQinv = np.dot(Q, Qinv)                         #performed matrix multiplication
print("\n Matrix QQ^{-1}:", QQinv)              #printed the answer
QΛ= np.dot(Q,Λ)                                 #performed matrix multiplication
QΛQT = np.dot(QΛ, Q.T)                          #performed matrix multiplication
print("\n Matrix QΛQTranspose:", QΛQT)          #printed the answer
QΛQinv = np.dot(QΛ, Qinv)                       #performed matrix multiplication
print("\n Matrix QΛQ^{-1}:", QΛQinv)            #printed the answer
```

Figure 9: Question 5 code

Output

```python
#Q5
print("\n Q5 Answer:")
eigenvalues, eigenvectors = np.linalg.eig(C)    #used LA.eig() to find eigenvalues and eigenvectors
Λ = np.diag(eigenvalues)                        #np.diag() is a NumPy function that constructs a diagonal matrix of eigenvalues as like L41
Q = eigenvectors                                #assigned eigenvectors to a variable called Q
print("\nMatrix C:")
check_orthogonality(C)                          #calling function check_orthogonality( ) to check if the matrix is orthogon
print("\nMatrix Q:")
check_orthogonality(Q)                          #calling function check_orthogonality( ) to check if the matrix is orthogonal
QQT = np.dot(Q, Q.T)                            #performed matrix multiplication between Q and Q.T, which is transpose of Q and set value to QQT
print("\n Matrix QQTranspose:", QQT)            #printed the answer
Qinv = np.linalg.inv(Q)                         #using np.linalg.inv(Q) took inverse of Q and assigned it to Qinv
QQinv = np.dot(Q, Qinv)                         #performed matrix multiplication
```

```
 Q5 Answer:

Matrix C:
True, Eigenvectors are orthogonal, hence the given matrix is symmetric.

Matrix Q:
False, Eigenvectors are not orthogonal.

 Matrix QQTranspose: [[ 1.00000000e+00 -1.39789998e-16  8.04859841e-17]
 [-1.39789998e-16  1.00000000e+00  8.41781068e-17]
 [ 8.04859841e-17  8.41781068e-17  1.00000000e+00]]

 Matrix QQ^{-1}: [[ 1.00000000e+00 -8.28732058e-17 -1.66585306e-17]
 [ 9.66390791e-17  1.00000000e+00  2.86669556e-17]
 [-1.47392129e-17  2.83069763e-18  1.00000000e+00]]

 Matrix QΛQTranspose: [[2.00000000e+00 3.00000000e+00 2.41214882e-16]
 [3.00000000e+00 4.00000000e+00 1.00000000e+00]
 [3.55388754e-16 1.00000000e+00 9.00000000e+00]]

 Matrix QΛQ^{-1}: [[ 2.00000000e+00  3.00000000e+00 -9.18520257e-17]
 [ 3.00000000e+00  4.00000000e+00  1.00000000e+00]
 [-3.62353456e-16  1.00000000e+00  9.00000000e+00]]
```

Figure 10: Question 5 output

Question 6.

Code

```python
#Q6
C = np.matrix('1 2; 6 10; 5 6')                #used np.matrix to create matrix 'C'
U, sigma, Vh = np.linalg.svd(C)                #used np.linalg.svd() to do Singular value decomposition on matrix C. It returns three matrices: U (left singular vectors),
m, n = C.shape                                 #using C.shape it helps to assign no: of rows and columns to m & n respectevely
complete_sigma = np.zeros((m, n))              #using np.zeros((m, n)) initiated a new matrix with zero that has same dimension of matrix c
complete_sigma[:n, :n] = np.diag(sigma)        #this line construct a new matrix that adds values in sigma as diagonal using np.diag(sigma)
USVtranspose = U @ complete_sigma @ Vh         #computed UsigmaVh using '@' which is used to perform matrix multiplication
V = Vh.T                                       #took transpose of Vh and assigned it to V
print("\n Q6 Answer:")                         #print matrices
print("Matrix U:", U)
print("\nMatrix Σ:", complete_sigma)
print("\nMatrix V:", V)
print("\nMatrix UΣV^T:", USVtranspose)
```

Figure 11: Question 6 code

Output

```python
#Q6
C = np.matrix('1 2; 6 10; 5 6')
U, sigma, Vh = np.linalg.svd(C)

m, n = C.shape
complete_sigma = np.zeros((m, n))
complete_sigma[:n, :n] = np.diag(sigma)

# Compute UΣV^T
USVtranspose = U @ complete_sigma @ Vh
V = Vh.T
# Print matrices
print("\n Q6 Answer:")
print("Matrix U:", U)
print("\nMatrix Σ:", complete_sigma)
print("\nMatrix V:", V)
print("\nMatrix UΣV^T:", USVtranspose)
```

```
 Q6 Answer:
Matrix U: [[-0.15659087 -0.26090591 -0.95257934]
 [-0.82190623 -0.50039591  0.27216553]
 [-0.54767641  0.82554954 -0.13608276]]

Matrix Σ: [[14.17480059  0.        ]
 [ 0.          1.03683564]
 [ 0.          0.        ]]

Matrix V: [[-0.55213548  0.83375441]
 [-0.83375441 -0.55213548]]

Matrix UΣV^T: [[ 1.  2.]
 [ 6. 10.]
 [ 5.  6.]]
PS C:\Users\Admin\Downloads\React-Portfolio-master>
```

Figure 12: Question 6 output

Question 7.

Code

```python
#Q7

from PIL import Image                                                   #install the PIL package if not previously installed
image_np = np.asarray(Image.open("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale.jpg"))   #importing image in the variable images_np
U, sigma, Vh = np.linalg.svd(image_np)                                  #used np.linalg.svd() to do Singular value decomposition or

compression_ratio = 2                                                   #set a varaible called 'compression_ratio' and assigned val
uncompressed_image_size = image_np.shape[0] * image_np.shape[1]         #multiplied dimensions of the image  to find uncompressed i

denominator = compression_ratio * (1 + image_np.shape[0] + image_np.shape[1])   #To calculate number of singular values used first calculat
NoOfSVused = uncompressed_image_size / denominator                      #calculated the number of singular value used
compressed_image_size = NoOfSVused * (1 + image_np.shape[0] + image_np.shape[1])   #using the NoOfSVused calculated ompressed image size

U_cr = U[:,:int(NoOfSVused)]                                            #extracts the first NoOfSVused columns of the left singular
sigma_cr = sigma[:int(NoOfSVused)]                                      #extracts the first NoOfSVused singular values from the arr
Vh_cr = Vh[:int(NoOfSVused),:]                                          #extracts the first NoOfSVused rows of the transpose of the
compressed_image_np = np.dot(U_cr * sigma_cr, Vh_cr)                    #reconstructs the compressed image matrix by multiplying U,
print("\n Q7 Answer:")
print("Dimension of the final compressed image:", compressed_image_np.shape)   #print the results
print("Number of singular values used:", int(NoOfSVused))

compressed_mountain_image = Image.fromarray(compressed_image_np.astype(np.uint8), mode="L")
compressed_mountain_image.save("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale_compressed.jpg")  #Save the compressed image to a file to the provided path
orginal_mountain_image = Image.fromarray(image_np, mode="L")
orginal_mountain_image.save("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale_orginal.jpg")        #Save the compressed image to a file to the provided path

print("Uncompressed image size:", uncompressed_image_size)              #print the results
print("Compressed image size:", compressed_image_size)
```

Figure 13: Question 7 code

Output

```python
NoOfSVused = uncompressed_image_size / denominator
compressed_image_size = NoOfSVused * (1 + image_np.shape[0] + image_np.shape[1])

U_cr = U[:,:int(NoOfSVused)]
sigma_cr = sigma[:int(NoOfSVused)]
Vh_cr = Vh[:int(NoOfSVused),:]
compressed_image_np = np.dot(U_cr * sigma_cr, Vh_cr)

print("Dimension of the final compressed image:", compressed_image_np.shape)
print("Number of singular values used:", int(NoOfSVused))

# Save the compressed image to a file
compressed_mountain_image = Image.fromarray(compressed_image_np.astype(np.uint8), mode="L")
compressed_mountain_image.save("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale_compressed.jpg")
orginal_mountain_image = Image.fromarray(image_np, mode="L")
orginal_mountain_image.save("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale_orginal.jpg")
```

```
>>> from PIL import Image
>>> image_np = np.asarray(Image.open("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale.jpg"))
>>> U, sigma, Vh = np.linalg.svd(image_np)
>>> compression_ratio = 2
>>> uncompressed_image_size = image_np.shape[0] * image_np.shape[1]
>>> denominator = compression_ratio * (1 + image_np.shape[0] + image_np.shape[1])
>>> NoOfSVused = uncompressed_image_size / denominator
>>> compressed_image_size = NoOfSVused * (1 + image_np.shape[0] + image_np.shape[1])
>>> U_cr = U[:,:int(NoOfSVused)]
>>> sigma_cr = sigma[:int(NoOfSVused)]
>>> Vh_cr = Vh[:int(NoOfSVused),:]
>>> compressed_image_np = np.dot(U_cr * sigma_cr, Vh_cr)
>>> print("Dimension of the final compressed image:", compressed_image_np.shape)
Dimension of the final compressed image: (823, 1280)
>>> print("Number of singular values used:", int(NoOfSVused))
Number of singular values used: 250
>>> compressed_mountain_image = Image.fromarray(compressed_image_np.astype(np.uint8), mode="L")
>>> compressed_mountain_image.save("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale_compressed.jpg")
>>> orginal_mountain_image = Image.fromarray(image_np, mode="L")
>>> orginal_mountain_image.save("C:\\Users\\Admin\\Desktop\\Spring sem\\DL\\mountain_grayscale_orginal.jpg")
```

Figure 14: Question 7 output

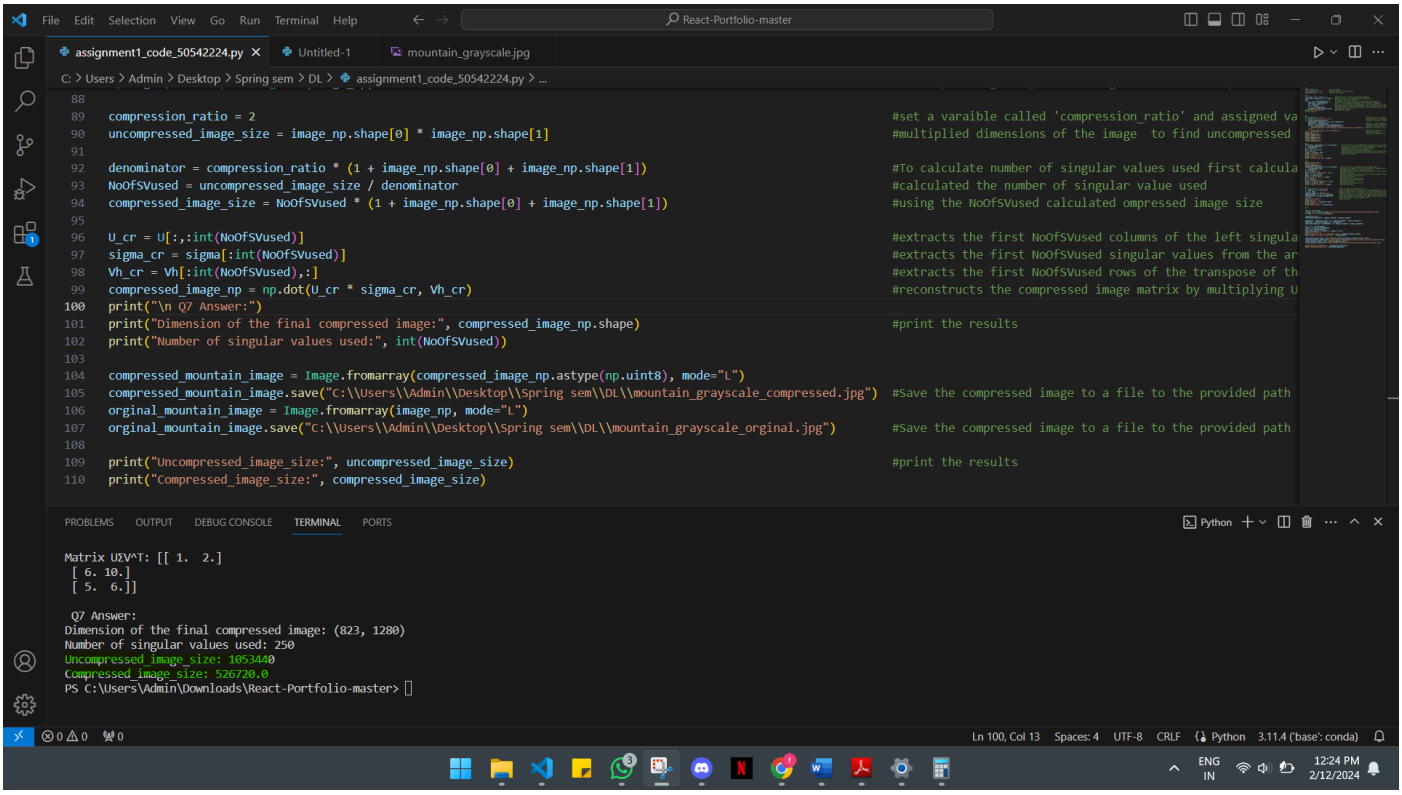

Figure 15: Question 7: highlighting the uncompressed and compressed image size

Figure 16: mountain_grayscale_compressed

Figure 17: mountain_grayscale_orginal