```
import numpy as np

#function to generate random matrix A
A=np.random.random((3,4))

#function for calculating the frobenious norm of matrix A
norm_val = np.linalg.norm(A)

print(f'The calculated norm value of matrix A {A} is = {norm_val} and with a  dimension(s)
```

```
The calculated norm value of matrix A [[0.91818594 0.20308071 0.25312483 0.02286723]
 [0.56250224 0.45649927 0.87335268 0.95542517]
 [0.91236559 0.70778487 0.28131133 0.11745314]] is = 2.1390620600374155 and with a
```

```
#implementing the Gram-Schmidt Algorithm
#this function will be printing  appropriate messages for Gram-Schmidt algorithm applicabi
#and then we use the same function to generate the matrix Q from A
import numpy as np
def gram_schmidt_algorithm(A):
  matrix_Q = []
  for i in range(len(A)):

    #this perfoms the normalization approach
    matrix_Q_orth = A[i]
    for j in range(len(matrix_Q)):
      matrix_Q_orth = matrix_Q_orth - (matrix_Q[j] @ A[i])*matrix_Q[j]
      if np.sqrt(sum(matrix_Q_orth**2)) <= 1e-10: #here we will be checking for linearly i
        print('The given vector is linearly dependent.')
        return matrix_Q
    # performing the Gram-Schmidt orthogonalization
    else:
      matrix_Q_orth = matrix_Q_orth / np.sqrt(sum(matrix_Q_orth**2))
      matrix_Q.append(matrix_Q_orth)

  print('The given vector is linearly independent.')
  return matrix_Q
```

```
Q = gram_schmidt_algorithm(A)
```

```
The given vector is linearly independent.
```

```
# printing  appropriate messages for Gram-Schmidt algorithm applicability on columns of th
if (sum(Q[0]**2))**0.5<=0:
  print(f'The Gram-Schmidt algorithm is not applicable on the matrix A from the first colu
else:
  print(f'The Gram-Schmidt algorithm  is applicable from the first column of A')
```

```
The Gram-Schmidt algorithm  is applicable from the first column of A
```

```
if Q[0] @ Q[1]<=0:
  print(f'The Gram-Schmidt algorithm is not applicable on the matrix A from the inner colu
else:
  print(f'The Gram-Schmidt algorithm  is applicable from the inner columns of A')
```

```
The Gram-Schmidt algorithm is not applicable on the matrix A from the inner columns
```

```
if (sum(Q[2]**2))**0.5<=0:
  print(f'The Gram-Schmidt algorithm is not applicable on the matrix A from the last colum
else:
  print(f'The Gram-Schmidt algorithm  is applicable from the last columns of A')
```

```
The Gram-Schmidt algorithm  is applicable from the last columns of A
```

```
#printing the matrix Q from A
print(f'The matrix Q produced by gram_schmidt_algorithm from A is{Q} ')
```

```
The matrix Q produced by gram_schmidt_algorithm from A is[array([0.94258363, 0.20847
```

```
def QR_decomposition(A):
  Matrix_Q_transpose = np.array(gram_schmidt_algorithm(A.T))
  tranposed_matrix = Matrix_Q_transpose @ A
  Matrix_Q = Matrix_Q_transpose.T
  return Matrix_Q, tranposed_matrix

Matrix_Q, tranposed_matrix = QR_decomposition(A)
print(f'From QR_decomposition the matrix Q is {Matrix_Q} and matrix R is {tranposed_matri
```

```
The given vector is linearly dependent.
From QR_decomposition the matrix Q is [[ 0.65057668 -0.75872209  0.03302687]
 [ 0.39855853  0.37812146  0.83556882]
 [ 0.64645269  0.53043845 -0.54839217]] and matrix R is [[ 1.41134162e+00  7.7161068
 [-8.79880583e-17  3.93966657e-01  2.87400334e-01  4.06218557e-01]
 [-3.63619779e-17 -6.11742778e-17  5.83837266e-01  7.34668334e-01]]
```

0s completed at 11:14 PM