<u>Explanation</u>

Principal Component Analysis

In order to find the values within the dataset that holds enough information to impact the information content of the dataset, the following steps and activities were taken as described below:

1. **Reading the data using pandas**

   The dataset was received in a csv file format which needs to be read into the coding environment. This dataset contains irrelevant columns and rows that would not be usable for the PCA functions.

   ```
   def read_csv(a_file_path):
      file = pd.read_csv(a_file_path)
      return file
   ```

   and calling the function as so

   ```
   data = read_csv(a_file_path="C:\\Users\\User\\Desktop\\Computer Vision Textbooks\\Week6_Task\\Student_Marksheet_1639650402.csv")
   ```

The data set would need to be cleaned by dropping the irrelevant columns and rows in the data set using the defined function showed below

2. **Cleaning the dataset**

   Once the dataset has been read into the workspace the first clean function is run to drop the first column, then the new dataset is passed into the column dropping function to remove the next identified irrelevant function.

   Finally, the row dropping function is called once to remove the irrelevant row. After this cleaning, our dataset is clean and ready to use. Below are the functions used to clean the dataset

   ```
   def drop_column(a_file, column_name):
      file_clean_column = a_file.drop(column_name, axis=1)
      return file_clean_column

   def drop_row(a_file, row_name):
      file_clean_row = a_file.drop(row_name, axis=0)
      return file_clean_row
   ```

   and the functions are called and implemented as so

   ```
   clean_column_file1 = drop_column(a_file=data, column_name="Student Total")
   clean_column_file2 = drop_column(a_file=clean_column_file1, column_name="Student Name")
   scores = drop_row(a_file=clean_column_file2, row_name=50)
   ```

3. **The mean score calculated**
   The mean score is now to be calculated. The mean score is calculated first because the mean represents the average value in a dataset. The mean is important because it gives us an idea of where the center value is located in a dataset. The mean is also important because it carries a piece of information from every observation in a dataset. The calculation of the mean score is implemented via a function as shown below

```
def mean_score_values(a_score):
    mean_scores =np.mean(a_score.T, axis=1)
    return mean_scores
```

   It is implemented by calling it as so

```
mean_scores = mean_score_values(scores)
```

4. **Normalization.**
   The goal of normalizing is to ensure that data is similar across all the records. It's also necessary for maintaining data integrity. The normalization process aims to remove data redundancy, which occurs when you have several fields with duplicate information. In this case our normalization function normalizes the score by considering the mean score and the score itself. This is implemented as shown below

```
def normalization_function(a_score, a_mean_score):
    normalized_result = a_score - a_mean_score
    return normalized_result
```

   It is then implanted through the call as so

```
normalized_scores = normalization_function(a_score=scores, a_mean_score=mean_scores) # centers the scores around the mean score
```

5. **Covariance Matrix**
   Covariance indicates the relationship of two variables whenever one variable changes. If an increase in one variable results in an increase in the other variable, both variables are said to have a positive covariance. Decreases in one variable also cause a decrease in the other. The covariance matrix is structured in such a way that the along the diagonal lies the variance itself. The implementation of this process is done via a function and called as shown below

```
def covariance_Matrix(a_normalized_score):

covariance_value = np.cov(a_normalized_score.T)

return covariance_value# Covert images to format that can be displayed by matplotlib
```

```
covar_scores = covariance_Matrix(a_normalized_score=normalized_scores) #variance is on the diagonal
```

6. **Finding the Eigenvector and Eigenvalue**

   Suppose we have plotted a scatter plot of random variables, and a line of best fit is drawn between these points. This line of best fit, shows the direction of maximum variance in the dataset. The Eigenvector is the direction of that line, while the eigenvalue is a number that tells us how the data set is spread out on the line which is an Eigenvector.

   A positive eigenvalue indicates a high variance in the eigenvector while a negative eigenvalue indicates a low variance in the corresponding eigenvector. This then follows that the eigenvectors corresponding to the negative eigenvalues do not contribute much to the information of the dataset.

   The function call below was used to calculate the eigenvalues and eigenvectors.

   ```
   def eigenval_eigenvec(a_covariance_matrix):
      eigenvalue_eigenvector = np.linalg.eig(a_covariance_matrix)
      return eigenvalue_eigenvector

   eigen_values,eigen_vectors = eigenval_eigenvec(a_covariance_matrix=covar_scores)
   ```

7. **Finding the Projected Matrix**

   In order to eliminate the values from the eigenvector and eigenvalues that do not influence or contribute to the information of the dataset, The projected matrix function would be used to remove them. The function call below was used to implement the projected matrix.

   ```
   def projected_matrix(an_eigenvector,a_normalized_score):
      proj_matrix = an_eigenvector.T.dot(a_normalized_score.T)
      return proj_matrix

   projected_mat=projected_matrix(an_eigenvector=eigen_vectors,a_normalized_score=normalized_scores)
   ```