This will tell steps to perform PCA.

Step 1: Perform Standardization.

```
# Data-preprocessing: Standardizing the data

from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler().fit_transform(data)
print(standardized_data.shape)
```

```
(15000, 784)
```

Step 2: Find co-variance matrix.

```
#find the co-variance matrix which is : A^T * A
sample_data = standardized_data

# matrix multiplication using numpy
covar_matrix = np.matmul(sample_data.T , sample_data)

print ( "The shape of variance matrix = ", covar_matrix.shape)
```

```
The shape of variance matrix =  (784, 784)
```

Step 3: Find eigen vector and eigen values:

```
# finding the top two eigen-values and corresponding eigen-vectors
# for projecting onto a 2-Dim space.

from scipy.linalg import eigh

# the parameter 'eigvals' is defined (low value to heigh value)
# eigh function will return the eigen values in asending order
# this code generates only the top 2 (782 and 783) eigenvalues.
values, vectors = eigh(covar_matrix, eigvals=(782,783))

print("Shape of eigen vectors = ",vectors.shape)
# converting the eigen vectors into (2,d) shape for easyness of further computations
vectors = vectors.T

print("Updated shape of eigen vectors = ",vectors.shape)
# here the vectors[1] represent the eigen vector corresponding 1st principal eigen vector
# here the vectors[0] represent the eigen vector corresponding 2nd principal eigen vector
```

```
Shape of eigen vectors =  (784, 2)
Updated shape of eigen vectors =  (2, 784)
```

Step 4: Generate new components or features by projecting original data on eigen vectors.

```
# projecting the original data sample on the plane
#formed by two principal eigen vectors by vector-vector multiplication.

import matplotlib.pyplot as plt
new_coordinates = np.matmul(vectors, sample_data.T)

print (" resultanat new data points' shape ", vectors.shape, "X", sample_data.T.shape," = ", new_coordinates.shape)
```

```
 resultanat new data points' shape  (2, 784) X (784, 15000)  =  (2, 15000)
```

Following this all 4 steps you will get new features or components using PCA.

There is built in function available which will perform most of the work in **sklearn.decomposition**, following are the steps for this process:

Step 1: import decomposition from sklearn and create instance of PCA from it.

```
# initializing the pca
from sklearn import decomposition
pca = decomposition.PCA()
```

Step 2: configure the number of components or new features you want and apply fit_transform, which will generate new data-matrix which contains our new components. And by using just two steps you can perform PCA.

```
# configuring the parameteres
# the number of components = 2
pca.n_components = 2
pca_data = pca.fit_transform(sample_data)

# pca_reduced will contain the 2-d projects of simple data
print("shape of pca_reduced.shape = ", pca_data.shape)
```

```
shape of pca_reduced.shape =  (15000, 2)
```

There is always a question of how many new features we should generate or how much dimensionality reduction we should perform, to how many components to generate we can do following thing:

We can calculate percentage of variance explained for all d' (where d' = d, d is no of features of original dataset) components, then we can do cumulative sum to obtain till how many features we are obtaining the required percentage of variance explained.

```
# PCA for dimensionality redcution (non-visualization)

pca.n_components = 784
pca_data = pca.fit_transform(sample_data)

percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_);

cum_var_explained = np.cumsum(percentage_var_explained)

# Plot the PCA spectrum
plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()


# If we take 200-dimensions, approx. 90% of variance is expalined.
```


```
# If we take 200-dimensions, approx. 90% of variance is expalined.
```