A
MINI PROJECT REPORT ON

"Handwriting Digit Classification (PCA)"

Prepared By

Mr. Nikhil Ghule (44)

Mr. Atharva Gujrathi (49)

Mr. Sanket Gangurde (39)

(B.E. Computer)

SAVITRIBAIPHULE PUNE UNIVERSITY

In the academic year 2021-22

Department of Computer Engineering

Sanjivani Rural Education Society's Sanjivani College of Engineering

Kopargaon - 423 603.

ACKNOWLEDGEMENT

The entire session of mini project completion phase so far was a great experience providing us with great insight and innovation into learning various data mining concepts and achievement of it. As is rightly said, for the successful completion of any work, people are the most important asset our mini project would not be materialized without the cooperation of many of the people involved. We would like to thank our family for their unconditional love and support. And being there for us, despite all the troubles and problems that grow in life.

We express our sincere gratitude to **Dr. D. B. Kshirsagar,** Head of Department (Computer) SRESCOE for his unending support and encouragement during the years We have studied under his tutelage.

We are greatly indebted to project guide **Dr. T. Bhaskar**, Assistant Professor (Computer), for providing valuable guidance at every stage of this project work.

Our sincere thanks go to all the teachers and staff for their help and understanding.

Mr. Nikhil Ghule (44)

Mr. Atharva Gujrathi (49)

Mr. Sanket Gangurde (39)

(B.E Computer B)

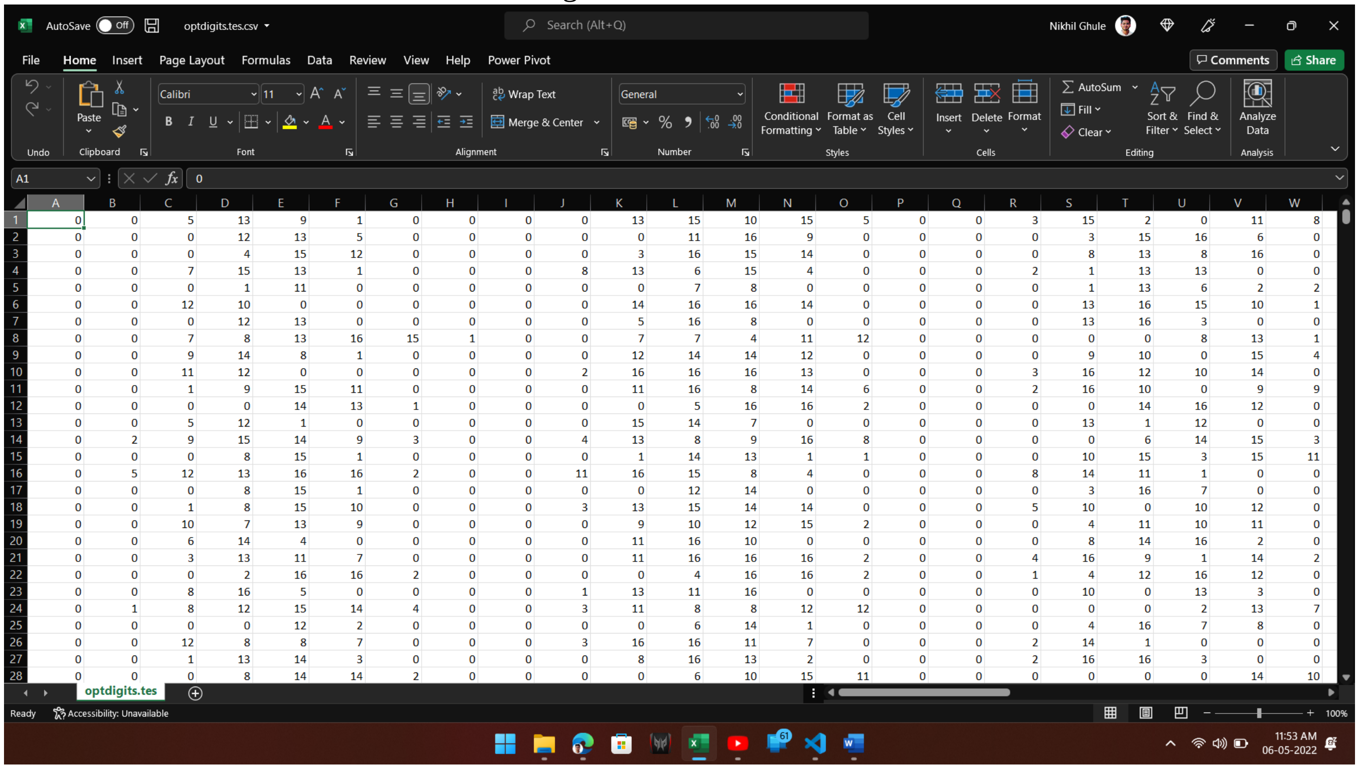
Contents

ABSTRACT: -

Handwritten character recognition is one of the critical issues in pattern recognition applications. The applications of digit recognition include in postal mail sorting, bank check processing, form data entry, etc. The heart of the problem lies within the ability to develop an efficient algorithm that can recognize handwritten digits. For training the model the data set dimensionality is huge, which hard to represent the data in visual format so the objective of this project east to reduce the dimensionality of the data set without losing much information. for this project we have used PCA technique to reduce the dimensionality of the input data set for the machine learning algorithms to perform efficiently and optimally.

INTRODUCTION: -

Intelligent image analysis is an appealing research area in Artificial Intelligence and also crucial for a variety of present open research difficulties. Handwritten digits recognition is a well-researched subarea within the field that is concerned with learning models to distinguish pre-segmented handwritten digits. It is one of the most important issues in data mining, machine learning, pattern recognition along with many other disciplines of artificial intelligence [1].The main application of machine learning methods over the last decade has determined efficacious in conforming decisive systems which are competing to human performance and which accomplish far improved than manually written classical artificial intelligence systems used in the beginnings of optical character recognition technology [2]. However, not all features of those specific models have been previously inspected. A great attempt of research worker in machine learning and data mining has been contrived to achieve efficient approaches for approximation of recognition from data [3]. In twenty first Century handwritten digit communication has its own standard and most of the times in daily life are being used as means of conversation and recording the information to be shared


AutoSave Off
optdigits.tes.csv
Search (Alt+Q)
Nikhil Ghule
File Home Insert Page Layout Formulas Data Review View Help Power Pivot
Comments Share
Undo Clipboard Font Alignment Number Styles Cells Editing Analysis
Calibri 11 General
Wrap Text Merge & Center
Conditional Formatting Format as Table Cell Styles
Insert Delete Format
AutoSum Fill Clear Sort & Filter Find & Select Analyze Data
optdigits.tes
Ready Accessibility: Unavailable
100%
11:53 AM 06-05-2022

A1 0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	0	0	5	13	9	1	0	0	0	0	13	15	10	15	5	0	0	3	15	2	0	11	8
2	0	0	0	12	13	5	0	0	0	0	0	11	16	9	0	0	0	0	3	15	16	6	0
3	0	0	0	4	15	12	0	0	0	0	3	16	15	14	0	0	0	0	8	13	8	16	0
4	0	0	7	15	13	1	0	0	0	8	13	6	15	4	0	0	0	2	1	13	13	0	0
5	0	0	0	1	11	0	0	0	0	0	0	7	8	0	0	0	0	0	1	13	6	2	2
6	0	0	12	10	0	0	0	0	0	0	14	16	16	14	0	0	0	0	13	16	15	10	1
7	0	0	0	12	13	0	0	0	0	0	5	16	8	0	0	0	0	0	13	16	3	0	0
8	0	0	7	8	13	16	15	1	0	0	7	7	4	11	12	0	0	0	0	0	8	13	1
9	0	0	9	14	8	1	0	0	0	0	12	14	14	12	0	0	0	0	9	10	0	15	4
10	0	0	11	12	0	0	0	0	0	2	16	16	16	13	0	0	0	3	16	12	10	14	0
11	0	0	1	9	15	11	0	0	0	0	11	16	8	14	6	0	0	2	16	10	0	9	9
12	0	0	0	0	14	13	1	0	0	0	0	5	16	16	2	0	0	0	0	14	16	12	0
13	0	0	5	12	1	0	0	0	0	0	15	14	7	0	0	0	0	0	13	1	12	0	0
14	0	2	9	15	14	9	3	0	0	4	13	8	9	16	8	0	0	0	0	6	14	15	3
15	0	0	0	8	15	1	0	0	0	0	1	14	13	1	1	0	0	0	10	15	3	15	11
16	0	5	12	13	16	16	2	0	0	11	16	15	8	4	0	0	0	8	14	11	1	0	0
17	0	0	0	8	15	1	0	0	0	0	0	12	14	0	0	0	0	0	3	16	7	0	0
18	0	0	1	8	15	10	0	0	0	3	13	15	14	14	0	0	0	5	10	0	10	12	0
19	0	0	10	7	13	9	0	0	0	0	9	10	12	15	2	0	0	0	4	11	10	11	0
20	0	0	6	14	4	0	0	0	0	0	11	16	10	0	0	0	0	0	8	14	16	2	0
21	0	0	3	13	11	7	0	0	0	0	11	16	16	16	2	0	0	4	16	9	1	14	2
22	0	0	0	2	16	16	2	0	0	0	0	4	16	16	2	0	0	1	4	12	16	12	0
23	0	0	8	16	5	0	0	0	0	1	13	11	16	0	0	0	0	0	10	0	13	3	0
24	0	1	8	12	15	14	4	0	0	3	11	8	8	12	12	0	0	0	0	0	2	13	7
25	0	0	0	0	12	2	0	0	0	0	0	6	14	1	0	0	0	0	4	16	7	8	0
26	0	0	12	8	8	7	0	0	0	3	16	16	11	7	0	0	0	2	14	1	0	0	0
27	0	0	1	13	14	3	0	0	0	0	8	16	13	2	0	0	0	2	16	16	3	0	0
28	0	0	0	8	14	14	2	0	0	0	0	6	10	15	11	0	0	0	0	0	0	14	10

SCOPE: -

- Data entry for business documents, e.g. Cheque, passport, invoice, bank statement and receipt
- Automatic number plate recognition
- In airports, for passport recognition and information extraction
- Automatic insurance documents key information extraction
- Traffic sign recognition
- Extracting business card information into a contact list
- More quickly make textual versions of printed documents, e.g. book scanning for Project Gutenberg
- Make electronic images of printed documents searchable, e.g., Google Books
- Converting handwriting in real-time to control a computer (pen computing)
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR. The purpose can also be to test the robustness of CAPTCHA anti-bot systems.
- Assistive technology for blind and visually impaired users
- Writing the instructions for vehicles by identifying CAD images in a database that are appropriate to the vehicle design as it changes in real time.
- Making scanned documents searchable by converting them to searchable PDFs

OBJECTIVES: -

- Understand the Dataset & clean-up.
- Reduce the dimensionality of the data set for machine learning algorithms to perform better and optimal
- Also compare the accuracy after reduction any information conservation.

We aim to solve the problem statement by creating a plan of action, here are some of the necessary steps:

1. Data Exploration
2. Exploratory Data Analysis (EDA)
3. Data Pre-processing
4. Data Manipulation
5. Feature Selection/Extraction
6. Predictive Modelling
7. Project Outcomes & Conclusion

METHODOLOGY: -

PCA

How do you do a PCA?

1. Standardize the range of continuous initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the principal component's axes

What Is Principal Component Analysis?

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analysing data much easier and faster for machine learning algorithms without extraneous variables to process.

So, to sum up, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.

Step by Step Explanation of PCA

STEP 1: STANDARDIZATION

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are significant differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

STEP 2: COVARIANCE MATRIX COMPUTATION

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a *p* × *p* symmetric matrix (where *p* is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables *x*, *y*, and *z*, the covariance matrix is a 3×3 matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?

It’s actually the sign of the covariance that matters:

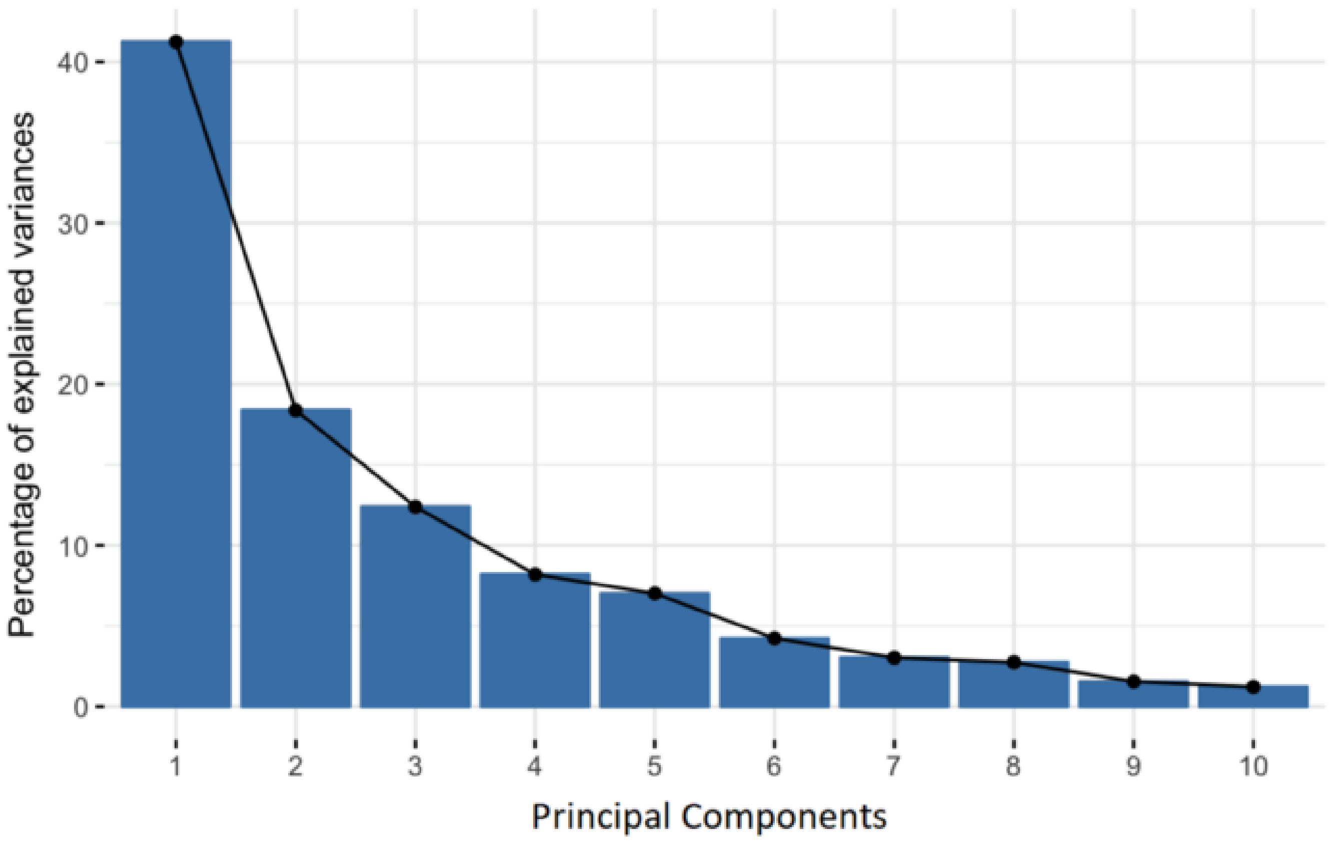
- if positive then: the two variables increase or decrease together (correlated)
- if negative then: One increases when the other decreases (Inversely correlated)

Now, that we know that the covariance matrix is not more than a table that summaries the correlations between all the possible pairs of variables, let us move to the next step.

STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the ***principal components*** of the data. Before getting to the explanation of these concepts, let's first understand what we mean by principal components.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.



Percentage of Variance (Information) for each by PC

Organizing information in principal components this way, will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

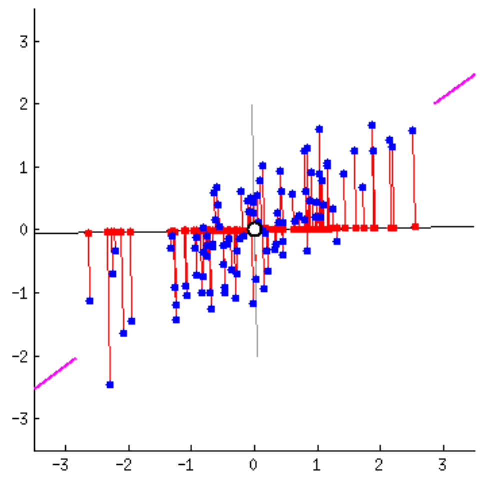
An important thing to realize here is that the principal components are less interpretable and don't have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance**, which is to say, the lines that capture most information of the data.

The relationship between variance and information here, is that the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more the information it has. Put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

How PCA Constructs the Principal Components

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the **largest possible variance** in the data set. For example, let us assume that the scatter plot of our data set is as shown below, can we guess the first principal component? Yes, it is approximately the line that matches the purple marks because it goes through the origin and it’s the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it’s the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

Now that we understood what we mean by principal components, let us go back to eigenvectors and eigenvalues. What you firstly need to know about them is that they always come in pairs, so that every eigenvector has an eigenvalue. And their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are three variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

Without further ado, it is eigenvectors and eigenvalues who are behind all the magic explained above, because the eigenvectors of the Covariance matrix are the *directions of the axes where there is the most variance* (most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Example:

Let’s suppose that our data set is 2-dimensional with 2 variables ***x,y*** and that the eigenvectors and eigenvalues of the covariance matrix are as follows:

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is *v1* and the one that corresponds to the second component (PC2) is*v2.*

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96% and 4% of the variance of the data.

STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only ***p*** eigenvectors (components) out of ***n***, the final data set will have only ***p*** dimensions.

Example:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors *v*1 and *v*2:

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Or discard the eigenvector *v*2, which is the one of lesser significance, and form a feature vector with *v*1 only:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

Discarding the eigenvector *v2* will reduce dimensionality by 1 and will consequently cause a loss of information in the final data set. But given that *v*2 was carrying only 4% of the information, the loss will be therefore not important, and we will still have 96% of the information that is carried by *v*1.

So, as we saw in the example, it is up to you to choose whether to keep all the components or discard the ones of lesser significance, depending on what you are looking for. Because if you just want to describe your data in terms of new variables (principal components) that are uncorrelated without seeking to reduce dimensionality, leaving out lesser significant components is not needed.

LAST STEP: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

SYSTEM REQUIREMENT: -

Software Components:

- Operating System: Windows 10.
- Jupyter Notebook
- Matplotlib

Hardware Components:

- Processer: i3
- RAM: 1GB
- Hard Disk: 5 GB
- Monitor

IMPLEMENTATION: -

```
%matplotlib inline

# numbers
import numpy as np
import pandas as pd

# stats
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# plots
import matplotlib.pyplot as plt
import seaborn as sns

# utils
import os, re
from pprint import pprint
```

In [ ]:

```
# Loading data

print([j for j in os.listdir('data/optdigits')])
['optdigits-orig.cv.Z', 'optdigits.names', 'optdigits.tes', 'optdigits.tra']
```

In [ ]:

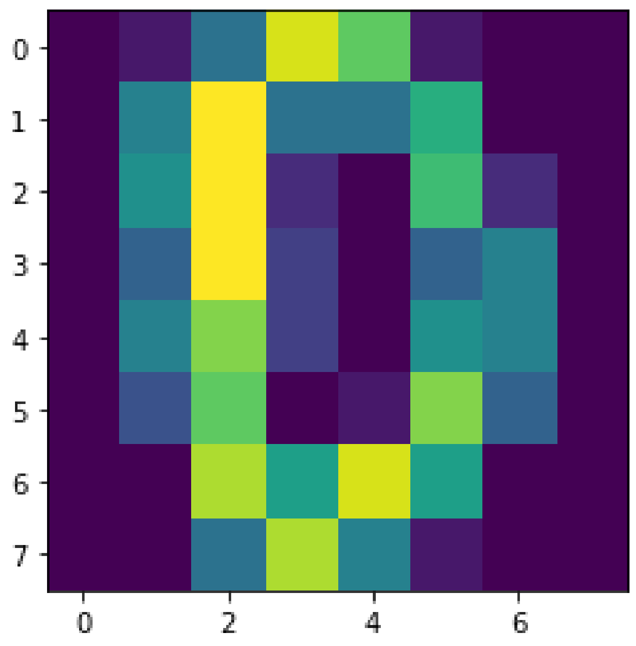
```
## No learning, no testing, yet.
#testing_df = pd.read_csv('data/optdigits/optdigits.tes',header=None)
#X_testing,  y_testing  = testing_df.loc[:,0:63],  testing_df.loc[:,64]

training_df = pd.read_csv('data/optdigits/optdigits.tra',header=None)
X_training, y_training = training_df.loc[:,0:63], training_df.loc[:,64]
```

In [ ]:

```
print (X_training.shape)
print( y_training.shape)
(3823, 64)
(3823,)
```

In [ ]:

```
mat = X_training.loc[0,:].values.reshape(8,8)
print(mat)
plt.imshow(mat)
plt.show()
[[ 0  1  6 15 12  1  0  0]
 [ 0  7 16  6  6 10  0  0]
 [ 0  8 16  2  0 11  2  0]
 [ 0  5 16  3  0  5  7  0]
 [ 0  7 13  3  0  8  7  0]
 [ 0  4 12  0  1 13  5  0]
 [ 0  0 14  9 15  9  0  0]
 [ 0  0  6 14  7  1  0  0]]
```

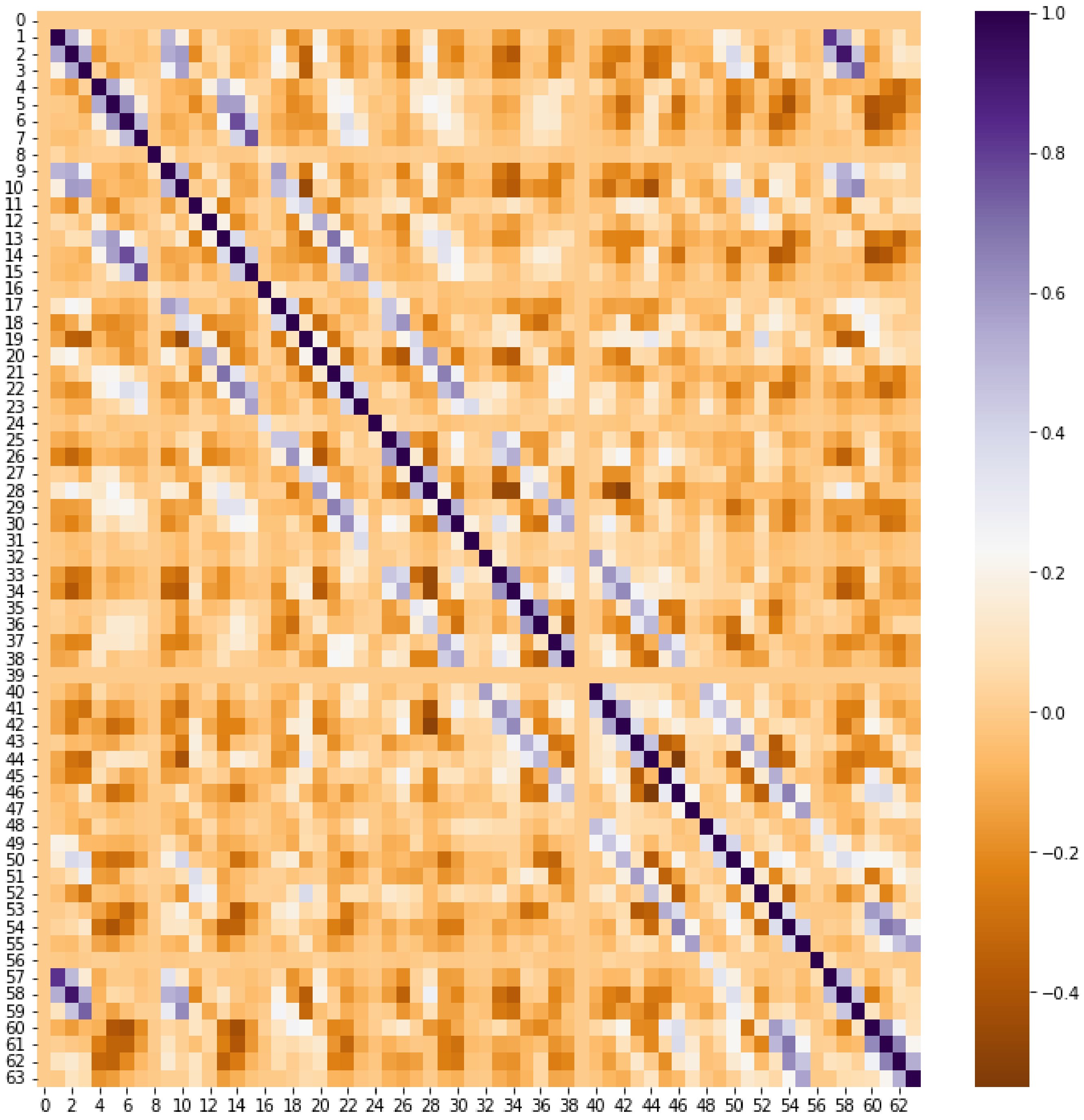


In [ ]:

```
# Normalize the data

def get_normed_mean_cov(X):
    X_std = StandardScaler().fit_transform(X)
    X_mean = np.mean(X_std, axis=0)

    ## Automatic:
    #X_cov = np.cov(X_std.T)

    # Manual:
    X_cov = (X_std - X_mean).T.dot((X_std - X_mean)) / (X_std.shape[0]-1)

    return X_std, X_mean, X_cov

X_std, X_mean, X_cov = get_normed_mean_cov(X_training)
```

In [ ]:

```
# Analyze the covariance matrix

fig = plt.figure(figsize=(12,12))
sns.heatmap(pd.DataFrame(X_cov), annot=False, cmap='PuOr')
plt.show()
```



In [ ]:

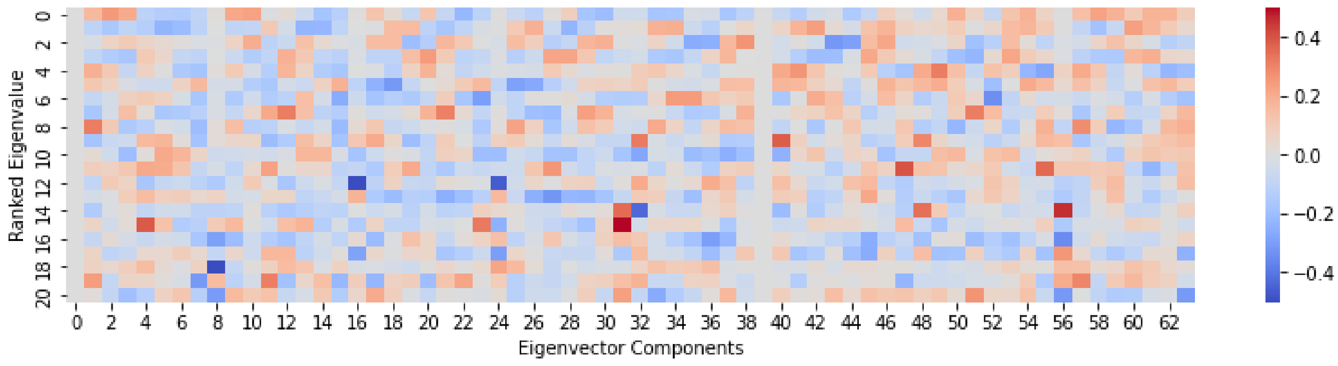
```
 # Eigenvalues and eigenvectors

 eigenvals, eigenvecs = np.linalg.eig(X_cov)

 # Eigenvalues are not necessarily sorted, but eigenval[i] *does* correspond to
 eigenvec[i]

 print ("Eigenvals shape: "+str(eigenvals.shape))
 print ("Eigenvecs shape: "+str(eigenvecs.shape))
Eigenvals shape: (64,)
Eigenvecs shape: (64, 64)
```

In [ ]:

```
# Create a tuple of (eigenvalues, eigenvectors)
unsrt_eigenvalvec = [(np.abs(eigenvals[i]), eigenvecs[:,i]) for i in
range(len(eigenvals))]

# Sort tuple by eigenvalues
eigenvalvec = sorted(unsrt_eigenvalvec, reverse=True, key=lambda x:x[0])

#pprint([pair for pair in eigenvalvec])
```

In [ ]:

```
fig = plt.figure(figsize=(14,3))
sns.heatmap(pd.DataFrame([pair[1] for pair in eigenvalvec[0:21]]),
            annot=False, cmap='coolwarm',
           vmin=-0.5,vmax=0.5)

plt.ylabel("Ranked Eigenvalue")
plt.xlabel("Eigenvector Components")
plt.show()
```

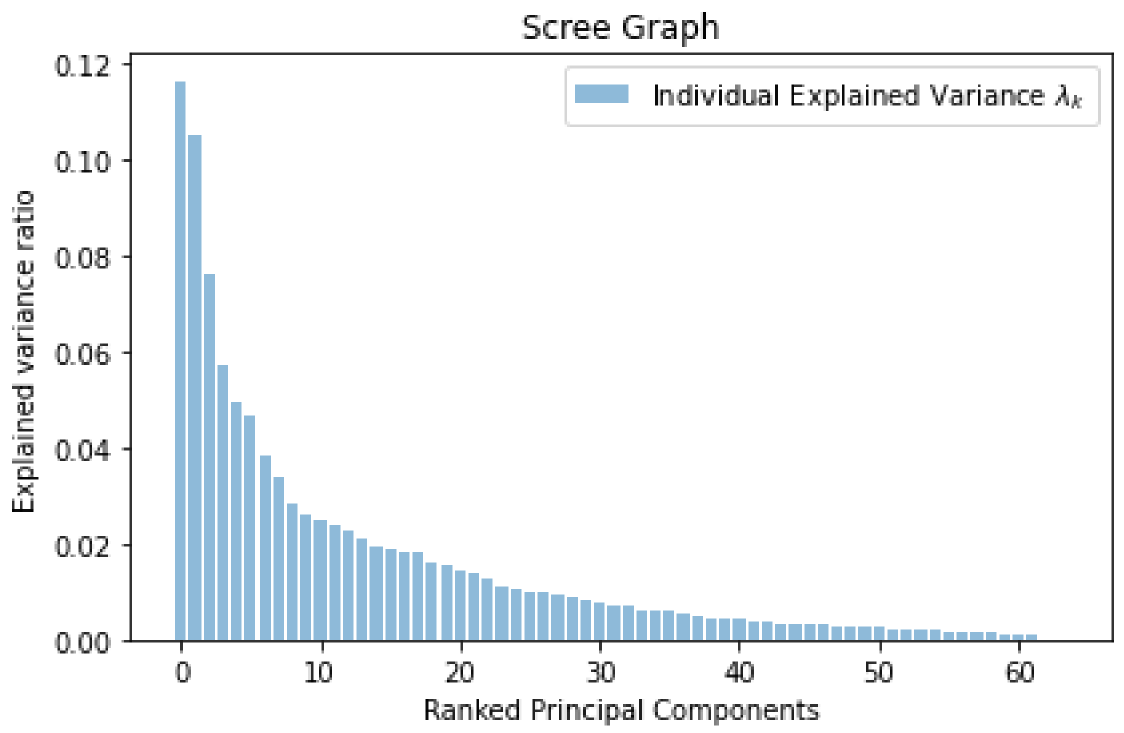


In [ ]:

```
# Explained variance

lam_sum = sum(eigenvals)
explained_variance = [(lam_k/lam_sum) for lam_k in sorted(eigenvals,
reverse=True)]
```

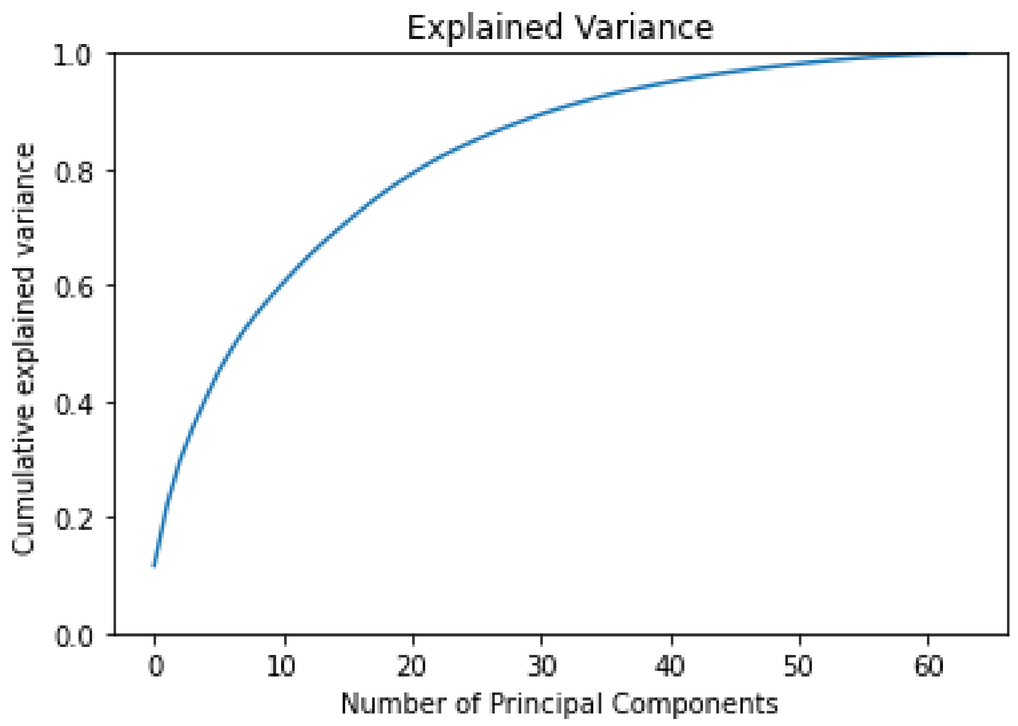
In [ ]:

```
plt.figure(figsize=(6, 4))

plt.bar(range(len(explained_variance)), explained_variance, alpha=0.5,
align='center',
        label='Individual Explained Variance $\lambda_{k}$')

plt.ylabel('Explained variance ratio')
plt.xlabel('Ranked Principal Components')
plt.title("Scree Graph")

plt.legend(loc='best')
plt.tight_layout()
```



In [ ]:

```
fig = plt.figure(figsize=(6,4))
ax1 = fig.add_subplot(111)

ax1.plot(np.cumsum(explained_variance))

ax1.set_ylim([0,1.0])

ax1.set_xlabel('Number of Principal Components')
ax1.set_ylabel('Cumulative explained variance')
ax1.set_title('Explained Variance')

plt.show()
```



In [ ]:

```
# Building a PCA Model: 4 Components

# This is a little easier to do by just using the PCA class from scikit-learn.
# But that isn't as fun as doing everything by hand.
sklearn_pca = PCA(n_components=4).fit(X_std)
```

In [ ]:

```
print( sklearn_pca.components_.shape)
```

```
(4, 64)
```

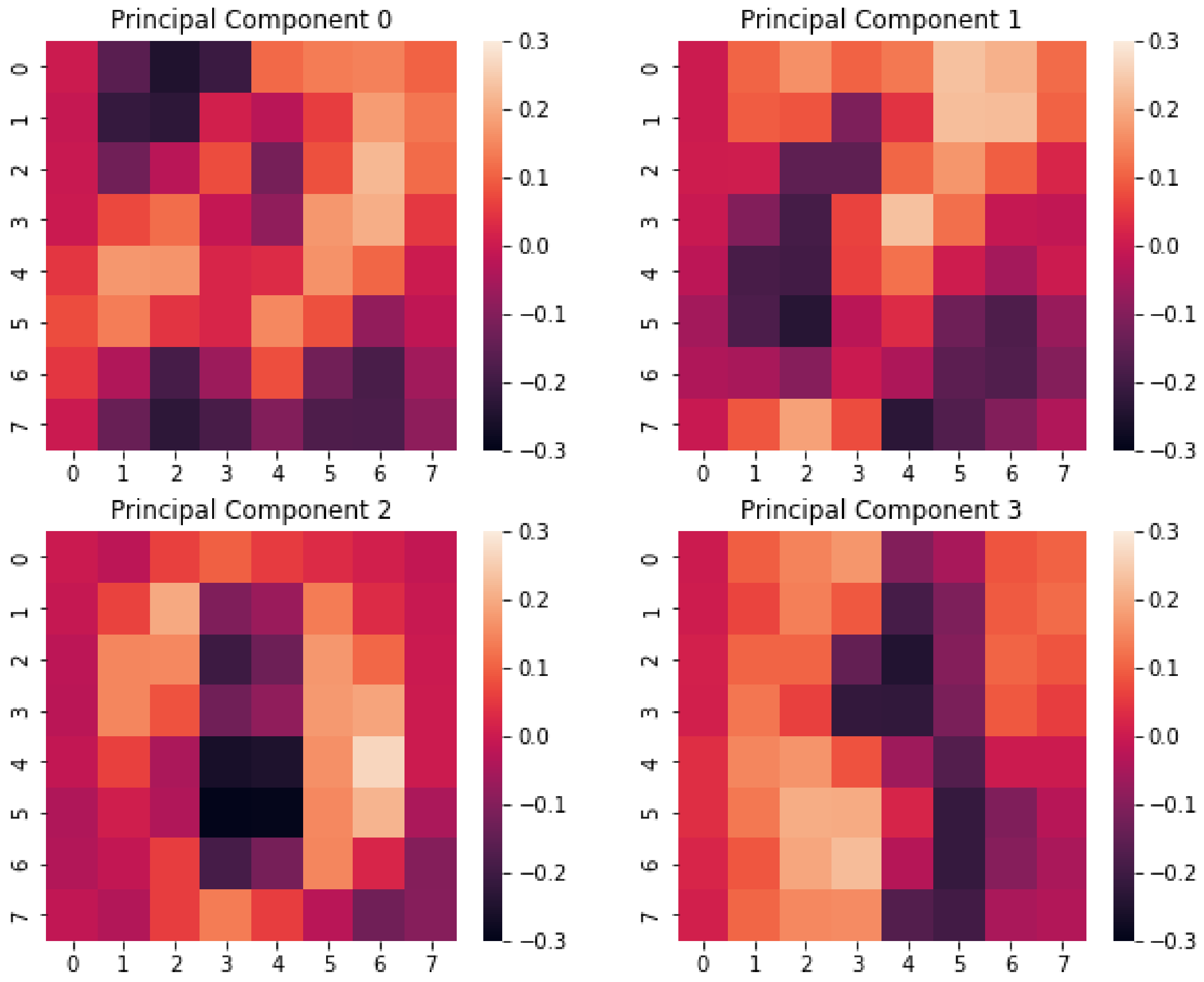
In [ ]:

```
print ("Principal Components:")
print (sklearn_pca.components_)
```

```
Principal Components:
[[-1.00180844e-19 -1.60866596e-01 -2.48610989e-01 -2.06309968e-01
   1.11112849e-01  1.33766786e-01  1.41239810e-01  1.01476311e-01
  -7.90871089e-03 -2.14417631e-01 -2.27923348e-01  1.04204216e-02
  -2.69295758e-02  5.70504769e-02  1.78714066e-01  1.28795102e-01
  -4.64422217e-03 -1.27950777e-01 -2.69800861e-02  7.62987294e-02
  -1.17227204e-01  8.09244741e-02  2.19642471e-01  1.13217832e-01
  -5.31610822e-04  7.07538433e-02  1.15030736e-01 -1.09633735e-02
  -8.31023587e-02  1.72865182e-01  2.03928118e-01  4.93099938e-02
   4.84459614e-02  1.71987343e-01  1.67406645e-01  2.13449662e-02
   3.14562518e-02  1.65938879e-01  1.06911664e-01 -0.00000000e+00
   7.68891896e-02  1.36242790e-01  4.57842149e-02  2.20167151e-02
   1.51516760e-01  8.06655893e-02 -8.02011531e-02 -1.80367641e-02
   4.78549762e-02 -3.96814899e-02 -1.89775980e-01 -6.59697772e-02
   7.95571077e-02 -1.24825104e-01 -1.84684307e-01 -6.05131258e-02
  -1.21383778e-03 -1.40464838e-01 -2.29052982e-01 -1.85697576e-01
  -1.01255085e-01 -1.76042127e-01 -1.79645567e-01 -8.63418929e-02]
 [ 6.72480259e-18  1.04154017e-01  1.63346582e-01  1.01332335e-01
   1.29065034e-01  2.32195410e-01  2.10503425e-01  1.12612924e-01
   2.33910585e-04  9.49424088e-02  8.64512959e-02 -1.08413174e-01
   4.29068132e-02  2.29145426e-01  2.25111640e-01  1.01066572e-01
   2.62612366e-03  5.29945910e-03 -1.53955431e-01 -1.53840468e-01
   1.07024106e-01  1.69542009e-01  9.82600731e-02  2.17411468e-02
  -2.35361113e-03 -1.02010793e-01 -1.90937571e-01  6.43338826e-02
   2.33519685e-01  1.18117129e-01 -9.31597940e-03 -1.51349536e-02
  -2.25308987e-02 -1.85701927e-01 -1.94750661e-01  6.07806749e-02
   1.21393038e-01  2.82248078e-03 -5.40314367e-02 -0.00000000e+00
  -5.63971870e-02 -1.79481605e-01 -2.37036638e-01 -2.42403364e-02
   3.17386860e-02 -1.29189514e-01 -1.76479975e-01 -7.18352550e-02
  -4.18129193e-02 -5.11239219e-02 -9.65554008e-02 -1.05694943e-03
  -4.43982432e-02 -1.55032873e-01 -1.72399218e-01 -1.00138319e-01
  -2.62023468e-03  8.85830211e-02  1.86903214e-01  7.53945754e-02
  -2.31346714e-01 -1.72122745e-01 -1.02677001e-01 -3.95797199e-02]
 [-6.96171532e-17 -2.16912214e-02  6.28299733e-02  9.99175589e-02
   5.57601804e-02  3.27041113e-02  1.09441236e-02 -1.21840633e-02
  -8.45392164e-03  6.45710368e-02  1.97870773e-01 -1.06685980e-01
  -6.95097419e-02  1.34444543e-01  3.21918442e-02 -5.69975953e-03
  -2.14480425e-02  1.49854185e-01  1.51441915e-01 -2.02151475e-01
  -1.32851510e-01  1.71745880e-01  1.08150486e-01 -1.57805555e-03
  -2.38491099e-02  1.48862971e-01  8.31944629e-02 -1.26654756e-01
  -8.37529293e-02  1.74420104e-01  1.89844029e-01  1.56535149e-03
  -1.39127897e-02  6.17408018e-02 -4.60493936e-02 -2.60208202e-01
  -2.52771445e-01  1.61978685e-01  2.65090755e-01 -0.00000000e+00
  -4.09009273e-02  7.95911506e-03 -3.91912283e-02 -3.19337376e-01
  -2.95021963e-01  1.51543095e-01  2.15299737e-01 -4.57840408e-02
  -3.59983362e-02 -1.28563243e-02  5.84179597e-02 -1.88668039e-01
  -1.19149351e-01  1.49289109e-01  2.28364098e-02 -9.94347184e-02
  -1.52963308e-02 -3.58095692e-02  5.72278507e-02  1.33705834e-01
   5.62962806e-02 -2.63918059e-02 -1.27993671e-01 -9.72014050e-02]
 [ 2.27011396e-17  9.68325355e-02  1.43496956e-01  1.70276875e-01
  -1.03085251e-01 -5.01301703e-02  8.47640324e-02  1.02859121e-01
   3.30175600e-03  6.63779875e-02  1.38855524e-01  9.11594466e-02
  -1.88219093e-01 -1.07820180e-01  9.24211171e-02  1.13857565e-01
   1.31704324e-02  1.03272540e-01  1.03870376e-01 -1.45313296e-01
  -2.45565153e-01 -1.00652501e-01  1.04985937e-01  8.57003524e-02
   1.07048223e-02  1.28509681e-01  6.14631504e-02 -2.20160882e-01
  -2.21140264e-01 -1.14638299e-01  9.00446877e-02  5.73848374e-02
   3.65386742e-02  1.49299831e-01  1.67807067e-01  8.24301668e-02
  -6.43276439e-02 -1.70580232e-01  1.83263473e-04  0.00000000e+00
   3.58648119e-02  1.30532724e-01  2.03379660e-01  2.01376918e-01
   2.02820142e-02 -2.13788379e-01 -1.07807386e-01 -2.94552303e-02
   2.27541392e-02  8.73258678e-02  1.93221847e-01  2.27182718e-01
  -3.18592290e-02 -2.15251945e-01 -9.67665201e-02 -4.74491220e-02
   1.12896292e-02  1.06492947e-01  1.50667467e-01  1.54871108e-01
  -1.70863409e-01 -1.95039404e-01 -4.77196807e-02 -3.63133463e-02]]
```

In [ ]:

```
colors = [sns.xkcd_rgb[z] for z in ['dusty purple','dusty green','dusty
blue','orange']]
fig = plt.figure(figsize=(10,8))
axes = [fig.add_subplot(220+i+1) for i in range(4)]

for i,ax in enumerate(axes):
    sns.heatmap(sklearn_pca.components_[i].reshape(8,8),
                square=True, ax=ax,
                vmax = 0.30, vmin=-0.30)
    ax.set_title('Principal Component '+str(i))
    #ax.set_xlabel('Horz Pixels')
    #ax.set_ylabel('Vert Pixels')

plt.show()
```



In [ ]:

```
print (np.cumsum(explained_variance)[4])
0.40461916287465743
```

In [ ]:

```
scikit_Z = sklearn_pca.fit_transform(X_std)
```

In [ ]:

```
# Do it by hand first (we'll also see how to do this with the scikit-learn PCA
object):
matW = np.hstack( pair[1].reshape(64,1) for pair in eigenvalvec[0:4] )
print (matW.shape)
(64, 4)
```

In [ ]:

```
# There are two ways to use the projection matrix:

# This is the "easy" way.
# This will make our projection reversed from scikit-learn,
# (flipped across one axis).
Z = X_std.dot(matW)

# This is the "correct" way.
# This projection will match scikit-learn's,
# But it also transposes Z (awkward).
#Z = matW.T.dot(X_std.T)

print( Z.shape)
```

```
(3823, 4)
```

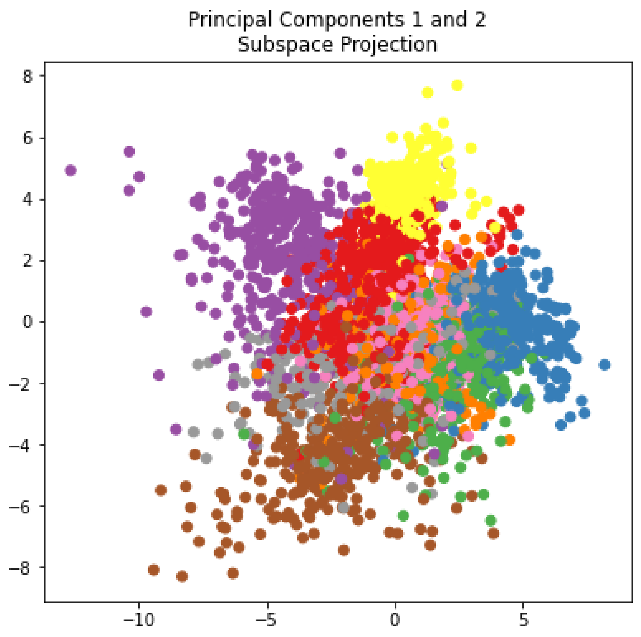
In [ ]:

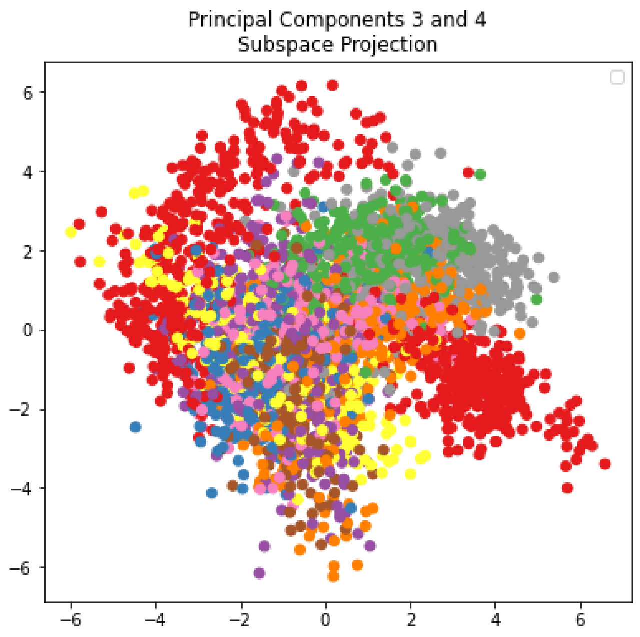
```
# To color each point by the digit it represents,
# create a color map with 10 elements (10 RGB values).
# Then, use the system response (y_training), which conveniently
# is a digit from 0 to 9.
def get_cmap(n):
    #colorz = plt.cm.cool
    colorz = plt.get_cmap('Set1')
    return [ colorz(float(i)/n) for i in range(n)]

colorz = get_cmap(10)
colors = [colorz[yy] for yy in y_training]
```

In [ ]:

```
fig = plt.figure(figsize=(14,6))
ax1, ax2 = [fig.add_subplot(120 + i + 1) for i in range(2)]

ax1.scatter( Z[:,0], Z[:,1] , c=colors )
ax1.set_title('Principal Components 1 and 2\nSubspace Projection')

ax2.scatter( Z[:,2], Z[:,3] , c=colors )
ax2.set_title('Principal Components 3 and 4\nSubspace Projection')

plt.legend()
plt.show()

n_components = 4
total_variance = np.sum(explained_variance[0:n_components])
for i in range(n_components):
    print("Explained Variance, Principal Component %d:
%0.4f"%(i,explained_variance[i]/total_variance))
```

```
No artists with labels found to put in legend.  Note that artists whose label start
with an underscore are ignored when legend() is called with no argument.
```

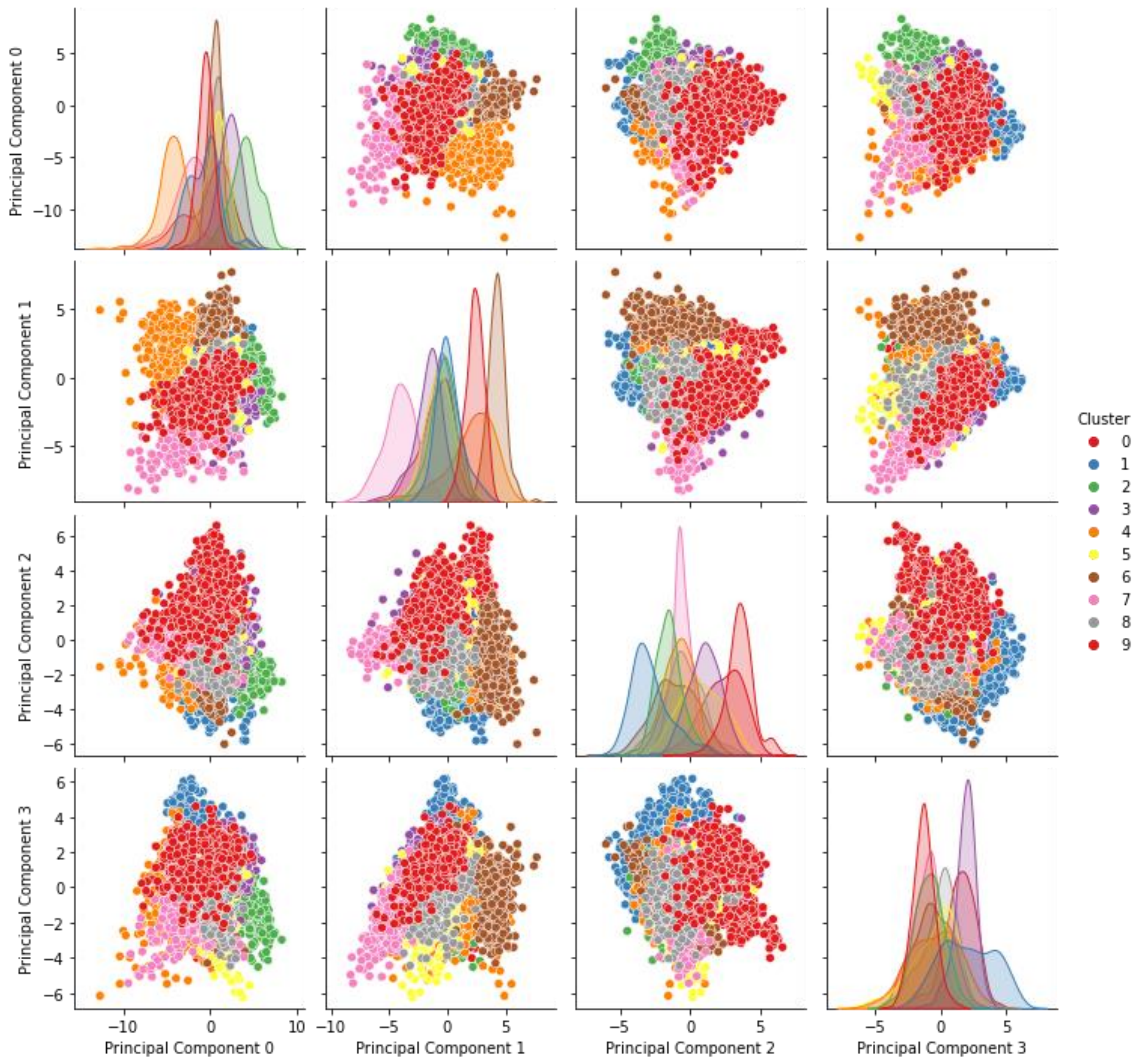




```
Explained Variance, Principal Component 0: 0.3279
Explained Variance, Principal Component 1: 0.2963
Explained Variance, Principal Component 2: 0.2149
Explained Variance, Principal Component 3: 0.1609
```

In [ ]:

```
pairplot_df = pd.DataFrame(Z, columns=['Principal Component '+str(j) for j in
range(Z.shape[1])])
pairplot_df.reindex(pairplot_df.columns.sort_values(ascending=True))
z_columns = pairplot_df.columns

pairplot_df['Cluster'] = y_training
pairplot_df = pairplot_df.sort_values('Cluster',ascending=True)
sns.pairplot(pairplot_df, hue='Cluster',
             vars=z_columns, # don't plot the category/system response
             palette='Set1')
plt.show()
```



In [ ]:

```
# Projecting the data onto the subspace

# Now repeat the same process of projecting all
# the high-dimensional inputs into a lower-dimensional
# subspace, but using the scikit-learn PCA object:

scikit_Z = sklearn_pca.fit_transform(X_std)
```

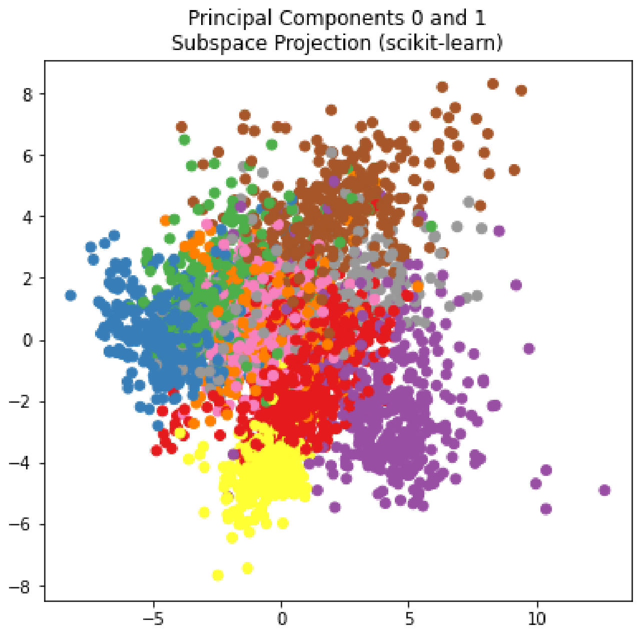
In [ ]:

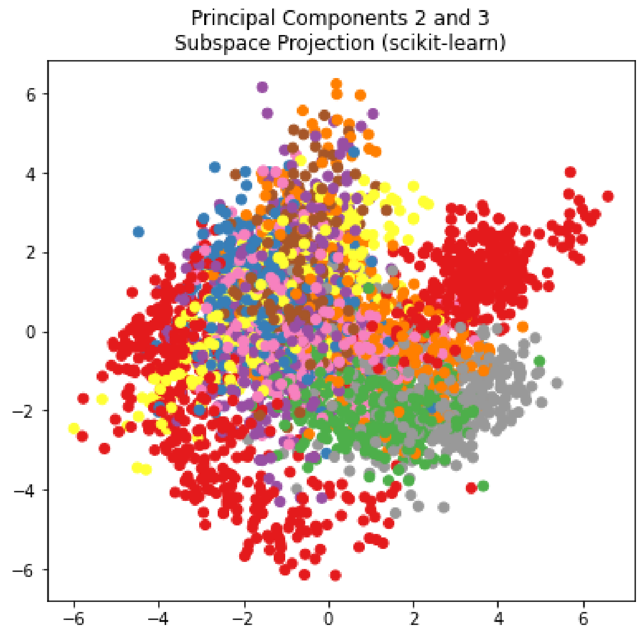
```
 print (scikit_Z.shape)
 print (Z.shape)
(3823, 4)
(3823, 4)
```

In [ ]:

```
fig = plt.figure(figsize=(14,6))
ax1, ax2 = [fig.add_subplot(120 + i + 1) for i in range(2)]
```

```
# To color each point by the digit it represents,
# create a color map with 10 elements (10 RGB values).
# Then, use the system response (y_training), which conveniently
# is a digit from 0 to 9.
def get_cmap(n):
    #colorz = plt.cm.cool
    colorz = plt.get_cmap('Set1')
    return[ colorz(float(i)/n) for i in range(n)]

colorz = get_cmap(10)
colors = [colorz[yy] for yy in y_training]

ax1.scatter( scikit_Z[:,0], scikit_Z[:,1] , c=colors )
ax1.set_title('Principal Components 0 and 1\nSubspace Projection (scikit-learn)')

ax2.scatter( scikit_Z[:,2], scikit_Z[:,3] , c=colors )
ax2.set_title('Principal Components 2 and 3\nSubspace Projection (scikit-learn)')

plt.show()

for i in range(4):
    print("Explained Variance, Principal Component %d:
%0.4f"%(i,sklearn_pca.explained_variance_[i]/np.sum(sklearn_pca.explained_variance
_.sum())))
```

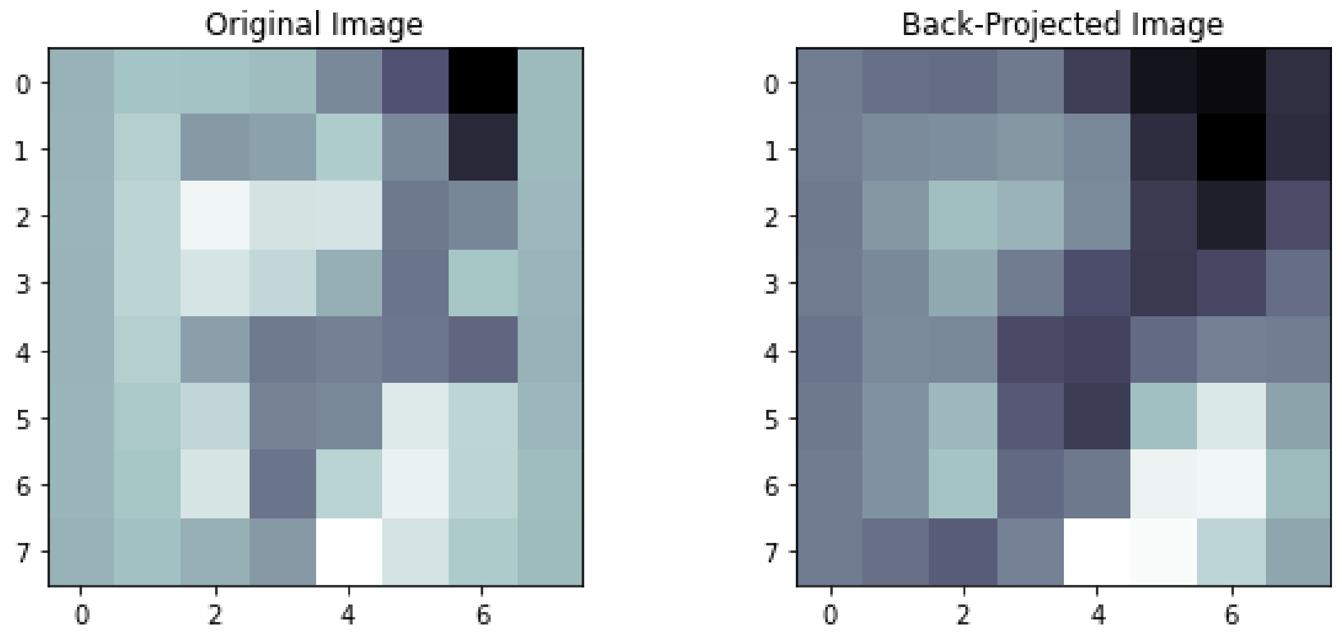


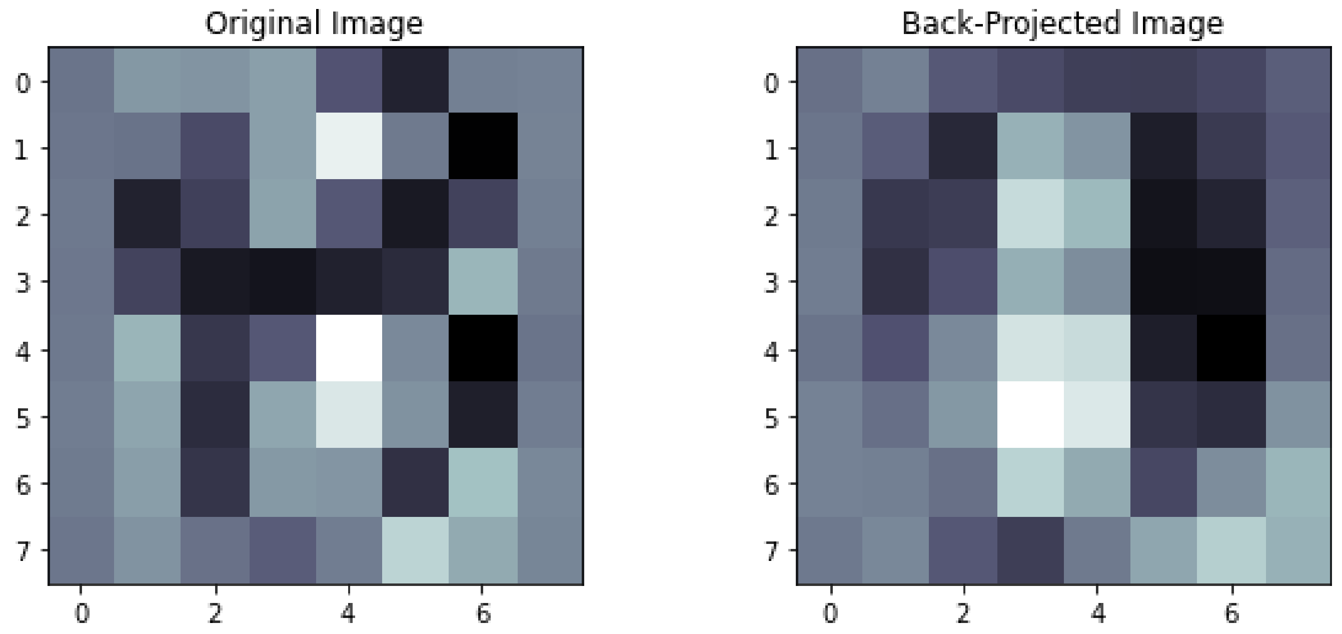


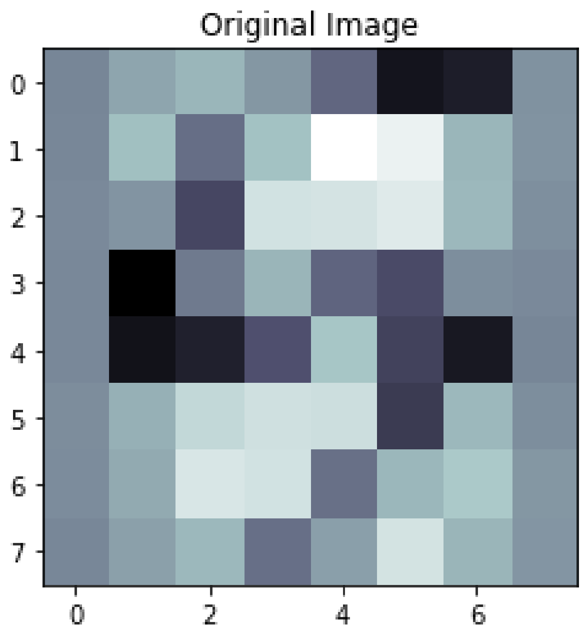
```
Explained Variance, Principal Component 0: 0.3279
Explained Variance, Principal Component 1: 0.2963
Explained Variance, Principal Component 2: 0.2149
Explained Variance, Principal Component 3: 0.1609
```

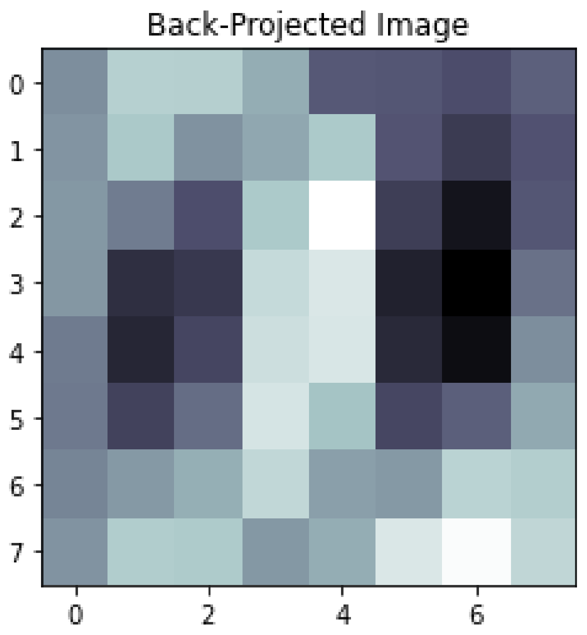
In [ ]:

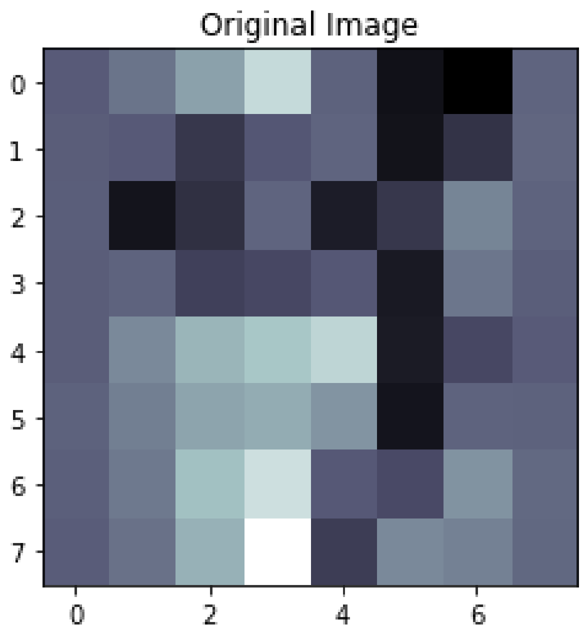
```
for j in range(5):
    X_digit = X_std[j+20]
    Z_digit = X_digit.dot(matW)
    Xhat_digit = Z_digit.dot(matW.T)
    fig = plt.figure(figsize=(10,4))
    ax1, ax2 = (fig.add_subplot(121+i) for i in range(2))

    ax1.imshow(X_digit.reshape(8,8),
              cmap = "bone_r")
    ax1.set_title('Original Image')

    ax2.imshow(Xhat_digit.reshape(8,8),
              cmap = "bone_r")
    ax2.set_title('Back-Projected Image')

plt.show()
```

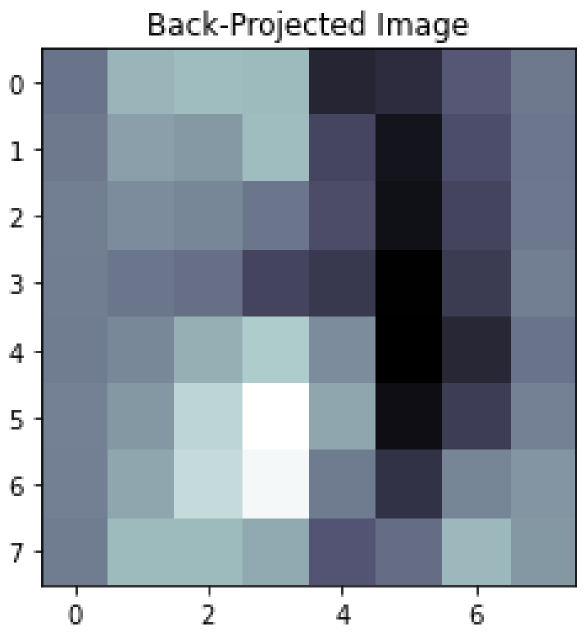


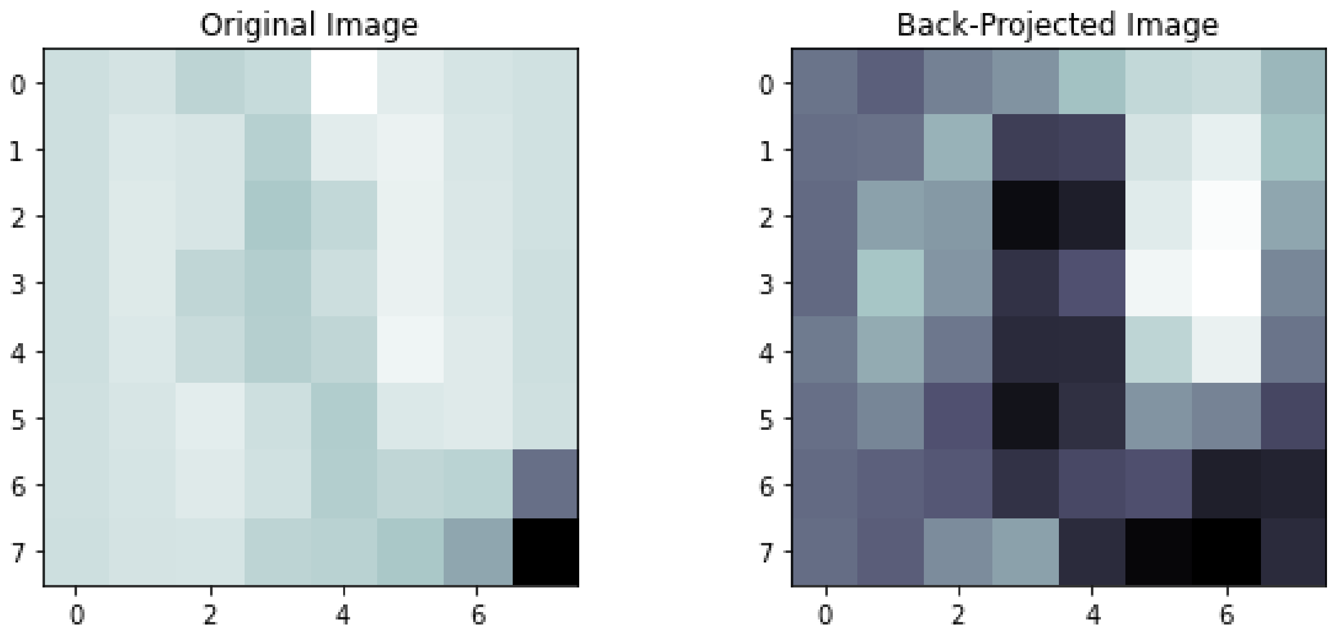



Original Image
0
1
2
3
4
5
6
7
0
2
4
6


Back-Projected Image
0
1
2
3
4
5
6
7
0
2
4
6


Original Image
0
1
2
3
4
5
6
7
0
2
4
6


Back-Projected Image
0
1
2
3
4
5
6
7
0
2
4
6



In [ ]:

```
 X_digit1 = X_std[0]
 Z_digit1 = X_digit1.dot(matW)
 Xhat_digit1 = Z_digit1.dot(matW.T)

 mse = ((Xhat_digit1 - X_digit1)**2).mean(axis=None)
 print( mse)
0.22482498175923768
```

In [ ]:

```
 # use the scikit pca object
 X_hat = sklearn_pca.inverse_transform( sklearn_pca.transform(X_std) )

 mse = ((X_hat - X_std)**2).mean(axis=None)
 print (mse)
 print (1-mse)
0.6249097670129751
0.37509023298702493
```

In [ ]:

```
 print( np.cumsum(explained_variance)[:4])
[0.11639052 0.22154846 0.29781225 0.35493186]
```

In [ ]:

```
 def pca_mse(n):
     # Make PCA
     pca = PCA(n_components=n, whiten=True)
     pca.fit(X_std)
     X_red = pca.transform(X_std)

     # MSE associated with back-projection:
     X_orig = X_std
     X_hat = pca.inverse_transform( pca.transform(X_orig) )
     mse = ((X_hat - X_orig)**2).mean(axis=None)

     return mse
```

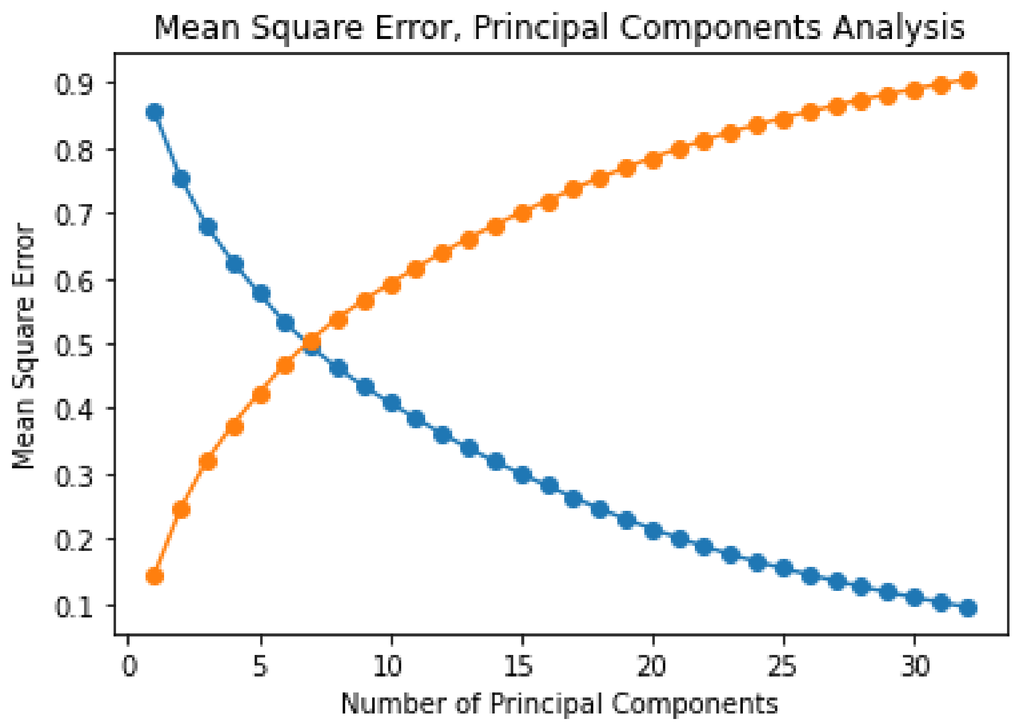
In [ ]:

```
mses = []
N = 33
for i in range(1,N):
    m = pca_mse(i)
    print ("%d-component PCA: MSE = %0.4g"%(i,m))
    mses.append((i,m))
mses = np.array(mses)
```

```
1-component PCA: MSE = 0.856
2-component PCA: MSE = 0.7541
3-component PCA: MSE = 0.6802
4-component PCA: MSE = 0.6249
5-component PCA: MSE = 0.5768
6-component PCA: MSE = 0.5317
7-component PCA: MSE = 0.4944
8-component PCA: MSE = 0.4617
9-component PCA: MSE = 0.4341
10-component PCA: MSE = 0.4086
11-component PCA: MSE = 0.3845
12-component PCA: MSE = 0.361
13-component PCA: MSE = 0.3389
14-component PCA: MSE = 0.3187
15-component PCA: MSE = 0.2997
16-component PCA: MSE = 0.2814
17-component PCA: MSE = 0.2636
18-component PCA: MSE = 0.2461
19-component PCA: MSE = 0.2304
20-component PCA: MSE = 0.2152
21-component PCA: MSE = 0.2014
22-component PCA: MSE = 0.188
23-component PCA: MSE = 0.1756
24-component PCA: MSE = 0.1646
25-component PCA: MSE = 0.1543
26-component PCA: MSE = 0.1445
27-component PCA: MSE = 0.1351
28-component PCA: MSE = 0.1261
29-component PCA: MSE = 0.1177
30-component PCA: MSE = 0.1097
31-component PCA: MSE = 0.1021
32-component PCA: MSE = 0.09522
```

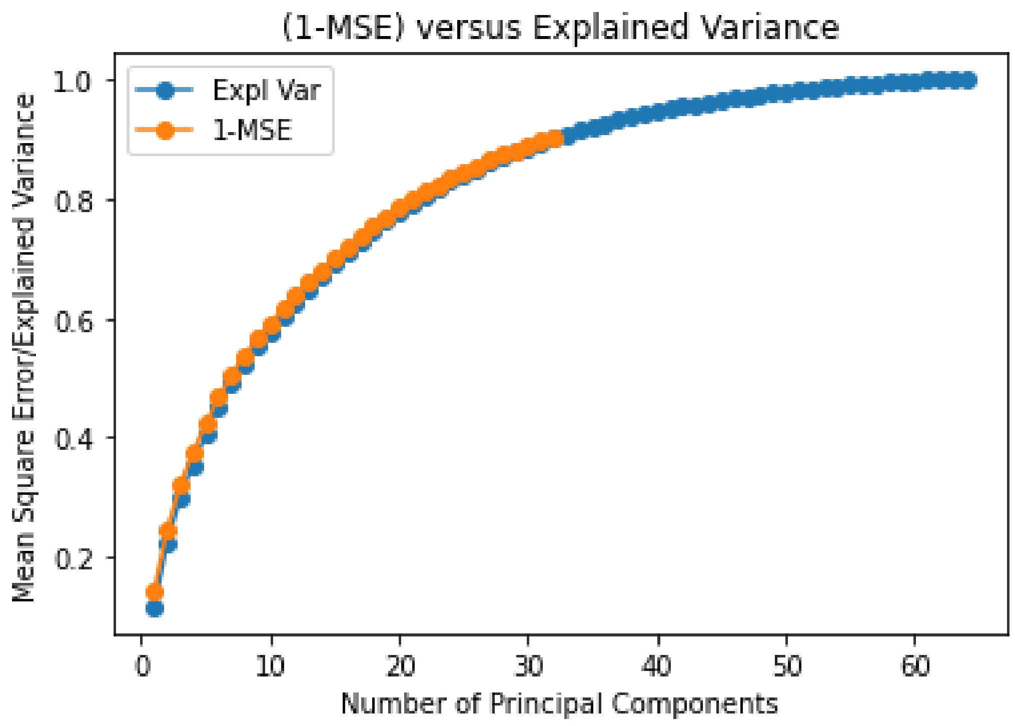
In [ ]:

```
plt.plot(mses[:,0],mses[:,1],'-o',label='MSE')
plt.plot(mses[:,0],1.0-mses[:,1],'-o',label='Rsq')

plt.title('Mean Square Error, Principal Components Analysis')
plt.xlabel('Number of Principal Components')
plt.ylabel('Mean Square Error')

plt.show()
```



In [ ]:

```
plt.plot(range(1,len(explained_variance)+1),np.cumsum(explained_variance),
         '-o',label='Expl Var')
plt.plot(mses[:,0],1.0-mses[:,1],'-o',label='1-MSE')

plt.title('(1-MSE) versus Explained Variance')
plt.xlabel('Number of Principal Components')
plt.ylabel('Mean Square Error/Explained Variance')
plt.legend(loc='best')
plt.show()
```



CONCLUSION:

In this project we reduced the dimensionality of input data set with preserving information and most accuracy we also visualised how over dimensionally reduced data compares with the original input data

Sr No.	Roll No	Student Name(s)	Sign
1	44	Mr. Nikhil Suresh Ghule	
2		Mr. Atharva Jitendra Gujrathi	
3		Mr. Sanket Arun Gangurde	

Prof. **T.Bhaskar**
[Guide]

References

Optical character recognition. (n.d.). Retrieved 5 6, 2022, from Wikipedia: The Free Encyclopedia: http://en.wikipedia.org/wiki/Optical_character_recognition

A Step-by-Step Explanation of Principal Component Analysis (PCA)
A Step-by-Step Explanation of Principal Component Analysis (PCA). (2021). Retrieved 6 May 2022, from https://builtin.com/data-science/step-step-explanation-principal-component-analysis