1 Given,

$$\cancel{A} = \begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}$$

a. Determining the population principal components Y_1 and Y_2 for the given covariance matrix:

Let I_2 identity matrix i.e $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$\therefore \lambda I = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

Then, $\det(A) - \det(I) = 0$

or, $\det\left(\begin{bmatrix} 5 & 2 \\ 2 & 2 \end{bmatrix}\right) - \det\left(\begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) = 0.$

or, $\begin{vmatrix} 5-\lambda & 2-0 \\ 2-0 & 2-\lambda \end{vmatrix} = 0$

or, $(5-\lambda)(2-\lambda) - 2\times2 = 0$ (def. of $||$)

or, $\lambda^2 - 7\lambda + 6 = 0.$

or, $\lambda^2 - 6\lambda - 1\lambda + 6 = 0$

or, $\lambda(\lambda-6) - 1(\lambda-6) = 0.$

Either, $\boxed{\lambda = 6}$ or $\boxed{\lambda = 1}$

Computing the Eigenvalues

If $\lambda = 6$,

$Av = \lambda v$

$\therefore v(A-\lambda) = 0$

or, $\begin{bmatrix} -1 & 2 \\ 2 & -4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

or, $\begin{bmatrix} -v_1 + 2v_2 \\ 2v_1 - 4v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\therefore v = 2v_2$

$\boxed{v_1 = 2v_2}$

$\therefore$ Eigenvector is $u = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

Normalizing

$\hat{u} = \frac{u}{\|u\|} = \frac{1}{\sqrt{2^2+1^2}} = \frac{1}{\sqrt{5}}$

$\therefore u = \begin{bmatrix} 2/\sqrt{5} \\ 1/\sqrt{5} \end{bmatrix}$

If $\lambda = 1$,

$Av = \lambda v$

$\therefore \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = 0$

or, $\begin{bmatrix} 4v_1 + 2v_2 \\ 2v_1 + v_2 \end{bmatrix} = 0$

or, $4v_1 = -2v_2$

or, $\boxed{v_2 = -2v_1}$

Eigenvector is $w = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

Normalizing

$\hat{w} = \frac{w}{\|w\|} = \frac{w}{\sqrt{1+4}}$

$\therefore \hat{w} = \begin{bmatrix} -1/\sqrt{5} \\ 2/\sqrt{5} \end{bmatrix}$

1st principal component

$y = \frac{2}{\sqrt{5}} x_1 + \frac{1}{\sqrt{5}} x_2$

2nd pc

$y = -\frac{1}{\sqrt{5}} x_1 + \frac{2}{\sqrt{5}} x_2$.

b. Calculating the propn of total popn variance explained by 1st PC

$$\frac{\lambda_1}{\lambda_1+\lambda_2} = \frac{6}{6+1} = 0.857$$

$$= 85.7\%$$

NB_hw_2

October 9, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import decomposition
from sklearn import preprocessing
from sklearn import metrics

%matplotlib inline
plt.style.use('seaborn-white')
```

[2]:
```
# Load Data and Inspect
hand_digits = pd.read_csv('optdigits.tra', header=None)
print(hand_digits.info())
hand_digits.head(3)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3823 entries, 0 to 3822
Data columns (total 65 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       3823 non-null   int64
 1   1       3823 non-null   int64
 2   2       3823 non-null   int64
 3   3       3823 non-null   int64
 4   4       3823 non-null   int64
 5   5       3823 non-null   int64
 6   6       3823 non-null   int64
 7   7       3823 non-null   int64
 8   8       3823 non-null   int64
 9   9       3823 non-null   int64
 10  10      3823 non-null   int64
 11  11      3823 non-null   int64
 12  12      3823 non-null   int64
 13  13      3823 non-null   int64
 14  14      3823 non-null   int64
 15  15      3823 non-null   int64
```

```
16  16       3823 non-null   int64
17  17       3823 non-null   int64
18  18       3823 non-null   int64
19  19       3823 non-null   int64
20  20       3823 non-null   int64
21  21       3823 non-null   int64
22  22       3823 non-null   int64
23  23       3823 non-null   int64
24  24       3823 non-null   int64
25  25       3823 non-null   int64
26  26       3823 non-null   int64
27  27       3823 non-null   int64
28  28       3823 non-null   int64
29  29       3823 non-null   int64
30  30       3823 non-null   int64
31  31       3823 non-null   int64
32  32       3823 non-null   int64
33  33       3823 non-null   int64
34  34       3823 non-null   int64
35  35       3823 non-null   int64
36  36       3823 non-null   int64
37  37       3823 non-null   int64
38  38       3823 non-null   int64
39  39       3823 non-null   int64
40  40       3823 non-null   int64
41  41       3823 non-null   int64
42  42       3823 non-null   int64
43  43       3823 non-null   int64
44  44       3823 non-null   int64
45  45       3823 non-null   int64
46  46       3823 non-null   int64
47  47       3823 non-null   int64
48  48       3823 non-null   int64
49  49       3823 non-null   int64
50  50       3823 non-null   int64
51  51       3823 non-null   int64
52  52       3823 non-null   int64
53  53       3823 non-null   int64
54  54       3823 non-null   int64
55  55       3823 non-null   int64
56  56       3823 non-null   int64
57  57       3823 non-null   int64
58  58       3823 non-null   int64
59  59       3823 non-null   int64
60  60       3823 non-null   int64
61  61       3823 non-null   int64
62  62       3823 non-null   int64
63  63       3823 non-null   int64
```

```
 64  64       3823 non-null   int64
dtypes: int64(65)
memory usage: 1.9 MB
None
```

[2]:

```
   0  1   2   3   4   5  6  7  8  9  …  55  56  57  58  59  60  61  \
0  0  1   6  15  12   1  0  0  0  7  …   0   0   0   6  14   7   1
1  0  0  10  16   6   0  0  0  0  7  …   0   0   0  10  16  15   3
2  0  0   8  15  16  13  0  0  0  1  …   0   0   0   9  14   0   0

   62  63  64
0   0   0   0
1   0   0   0
2   0   0   7

[3 rows x 65 columns]
```

[3]:

```
# reading the unique target variables
unique_targets = hand_digits[64].unique()
print(unique_targets)
```

```
[0 7 4 6 2 5 8 1 9 3]
```

The hand_digits df has sixty-four ($p = 64$) inputs plus the target variable that indicates the digit 0-9 as shown above.

[4]:

```
#b Remove any unary column (i.e., containing only one value).
hand_digits = hand_digits.loc[:, hand_digits.nunique() > 1]
print(hand_digits.shape)
```

```
(3823, 63)
```

We dropped one column which was unary column.

[5]:

```
# c Check for missing values
print(hand_digits.isnull().sum().sum())
```

```
0
```

There are not any missing values in the dataframe.

[6]:

```
#d Checking if the data is standardized
if np.allclose(hand_digits.mean(), 0) and np.allclose(hand_digits.std(), 1):
    print("Data is standardized")
else:
    print("Data is not standardized")
```

```
Data is not standardized
```

[7]:
```python
# Standardize the data matrix X so that all variables are given a mean of zero␣
↪and a standard deviation of one.
# excluding the target variable (last column)
from sklearn.preprocessing import StandardScaler
X = hand_digits.iloc[:, :-1]
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

[8]:
```python
standarized_df = pd.DataFrame(X)
```

[9]:
```python
# checking the oveall mean
standarized_df.describe()
```

[9]:
```
                  0             1             2             3             4  \
count  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03
mean   6.870140e-16 -1.158721e-17 -1.623371e-16  2.239032e-16 -1.388142e-17
std    1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00
min   -3.476105e-01 -1.183724e+00 -2.771826e+00 -2.524041e+00 -9.809412e-01
25%   -3.476105e-01 -9.677879e-01 -4.239971e-01 -5.403346e-01 -9.809412e-01
50%   -3.476105e-01 -1.040426e-01  2.803515e-01  3.413125e-01 -2.682243e-01
75%   -3.476105e-01  7.597028e-01  7.499173e-01  7.821361e-01  8.008511e-01
max    8.880965e+00  2.271257e+00  9.847002e-01  1.002548e+00  1.869927e+00

                  5             6             7             8             9  \
count  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03
mean  -3.130833e-16  1.590265e-16 -2.157809e-16 -1.879509e-16 -3.210731e-16
std    1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00
min   -4.115666e-01 -1.353324e-01 -2.362917e-02 -6.423761e-01 -1.946227e+00
25%   -4.115666e-01 -1.353324e-01 -2.362917e-02 -6.423761e-01 -6.582238e-01
50%   -4.115666e-01 -1.353324e-01 -2.362917e-02 -6.423761e-01  4.457787e-01
75%   -4.115666e-01 -1.353324e-01 -2.362917e-02  3.406008e-01  8.137795e-01
max    4.334796e+00  1.508160e+01  5.643531e+01  4.272508e+00  9.977799e-01

       …            52            53            54            55  \
count  …  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03
mean   …  2.701940e-16 -9.007821e-16  3.516983e-16  4.302078e-16
std    …  1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00
min    … -7.639066e-01 -1.932010e-01 -1.617539e-02 -3.050075e-01
25%    … -7.639066e-01 -1.932010e-01 -1.617539e-02 -3.050075e-01
50%    … -5.598673e-01 -1.932010e-01 -1.617539e-02 -3.050075e-01
75%    …  6.643687e-01 -1.932010e-01 -1.617539e-02 -3.050075e-01
max    …  2.500723e+00  1.543870e+01  6.182233e+01  1.047174e+01

                 56            57            58            59            60  \
count  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03  3.823000e+03
mean   4.029386e-17  4.445103e-16  3.178206e-16 -1.305086e-16  3.680318e-16
std    1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00  1.000131e+00
```

```
min   -1.176029e+00 -2.755685e+00 -2.296235e+00 -1.160247e+00 -5.227936e-01
25%   -9.752000e-01 -4.483164e-01 -4.930910e-01 -1.160247e+00 -5.227936e-01
50%   -1.718840e-01  2.438943e-01  3.083064e-01 -1.212969e-01 -5.227936e-01
75%    8.322610e-01  7.053680e-01  9.093544e-01  9.176534e-01 -2.623710e-02
max    2.037235e+00  9.361049e-01  9.093544e-01  1.610287e+00  3.449659e+00

                 61
count  3.823000e+03
mean  -1.956307e-15
std    1.000131e+00
min   -1.757406e-01
25%   -1.757406e-01
50%   -1.757406e-01
75%   -1.757406e-01
max    1.373073e+01

[8 rows x 62 columns]
```

[10]:
```python
# e
from sklearn.decomposition import PCA

# run the PCA algorithm on the standardized data with number of components␣
↪equal
# to the total number of columns in the data.
pca = PCA(n_components=standardized_df.shape[1])
pca.fit(standarized_df)
```

[10]:
```
PCA(n_components=62)
```

[12]:
```python
# Fit the PCA model and transform data to get the principal components

# Instantiate PCA estimator
pca = decomposition.PCA()
df_plot = pd.DataFrame(pca.fit_transform(standarized_df), columns=␣
↪standarized_df.columns, index=standarized_df.index)
df_plot
```

[12]:
```
             0         1         2         3         4         5         6  \
0     0.021179 -1.506218  4.028060  2.837064  1.121506 -1.048719  0.150893
1    -0.436318 -3.001971  6.068029  2.907716  1.439000 -0.447028  0.798956
2     1.363008  3.160016 -0.743226  1.395725  0.314563  0.968713 -3.625171
3     4.499442  0.949555  0.433865 -1.720058 -0.517172 -2.701682  1.721142
4    -1.199084 -3.264752  1.706263  1.130340 -1.262345 -0.225203 -1.368362
…          …         …         …         …         …         …         …
3818 -0.664700  0.199198  4.083232 -1.080674 -1.692163  1.007006 -0.950205
3819  4.295936 -4.344692 -2.048086  1.937549  3.064987  0.580706  0.111593
3820 -0.360348 -3.822805 -1.171858  1.853227 -0.492848  0.137883 -1.624683
```

```
3821 -1.351938 -5.283408  1.079286  3.439551 -1.187103  0.748811 -1.533931
3822  3.375207  4.220092  0.595156  0.048416 -1.199746  1.398314 -1.116004

             7         8         9  …        52        53        54  \
0    -0.236166 -1.144235  0.216062  …  0.079098 -0.099608 -0.328052
1    -1.762127 -1.696441 -0.287052  …  0.595869  0.203963  0.451097
2    -2.136621 -0.731816  0.292576  … -0.539342  0.126740  0.011906
3     0.417025 -1.923872  1.028773  …  0.320966  0.120637  0.695168
4     2.425412  0.360641 -0.039846  … -0.392310 -0.106501  0.015097
…          …         …         …  …       …         …         …
3818  0.498456 -0.628901  1.043451  …  0.065532  0.205556 -0.269823
3819 -0.103625 -1.386905 -0.574874  … -0.700666 -0.479226 -0.397264
3820  1.360577  1.415196 -1.159309  … -0.172458  0.365674  0.195132
3821  1.703575 -0.440460  0.615567  …  0.293833 -0.101132 -0.168334
3822 -1.452023 -1.034071  0.813605  … -0.098148  0.095915  0.272171

            55        56        57        58        59        60        61
0    -0.049694  0.407308 -0.326841  0.361029  0.414270  0.096211  0.148911
1    -0.031893 -0.077154  0.484712 -0.462682  0.138005 -0.183608 -0.015544
2     0.141793 -0.578015  0.079724  0.067017  0.179884 -0.199439 -0.092629
3    -0.101969 -0.594626  0.927018 -0.535342  0.559234 -0.129050 -0.221793
4     0.002365 -0.544356  0.031109 -0.245522 -0.021291 -0.064236 -0.103955
…          …         …         …         …         …         …         …
3818  0.040811  0.132728 -0.035091  0.146050 -0.141946  0.395055 -0.486348
3819  0.298726  0.091618  0.063463 -0.079364  0.217436 -0.092169  0.106547
3820 -0.069563  0.207445 -0.378826 -0.302459  0.308900 -0.004402 -0.259517
3821  0.019024  0.007306 -0.090449  0.234468  0.101860  0.047271 -0.221078
3822  0.344237 -0.239525  0.327969  0.258709 -0.170725  0.013201 -0.359563

[3823 rows x 62 columns]
```

[13]:
```python
print(df_plot.shape)
df_plot.iloc[:, 0]
```

```
(3823, 62)
```

[13]:
```
0       0.021179
1      -0.436318
2       1.363008
3       4.499442
4      -1.199084
          …
3818   -0.664700
3819    4.295936
3820   -0.360348
3821   -1.351938
3822    3.375207
```

```
Name: 0, Length: 3823, dtype: float64
```

1 E

Let's look at the proportion of variance explained (PVE) of the raw data as explained by Principal Component Loading.

[14]:
```
pca.explained_variance_ratio_
```

[14]:
```
array([0.11639052, 0.10515794, 0.07626379, 0.05711962, 0.0496873 ,
       0.04648104, 0.03854584, 0.03379051, 0.02851031, 0.02630916,
       0.02503596, 0.02408691, 0.02278827, 0.02093577, 0.01955806,
       0.01890912, 0.01843386, 0.01800929, 0.01622157, 0.01565134,
       0.01425234, 0.01378334, 0.01288684, 0.01128819, 0.01061524,
       0.01012621, 0.00972703, 0.00929472, 0.00863325, 0.00824354,
       0.00787725, 0.00712658, 0.00695497, 0.00629398, 0.00607752,
       0.00592961, 0.00533857, 0.00488499, 0.00451338, 0.00446158,
       0.0043551 , 0.00401866, 0.00385701, 0.0034535 , 0.00332456,
       0.00309779, 0.00307227, 0.00277881, 0.00266685, 0.00256959,
       0.00242815, 0.00235895, 0.0021833 , 0.00207732, 0.00192919,
       0.0018067 , 0.0016325 , 0.00153693, 0.00142055, 0.00123766,
       0.00106562, 0.00093372])
```

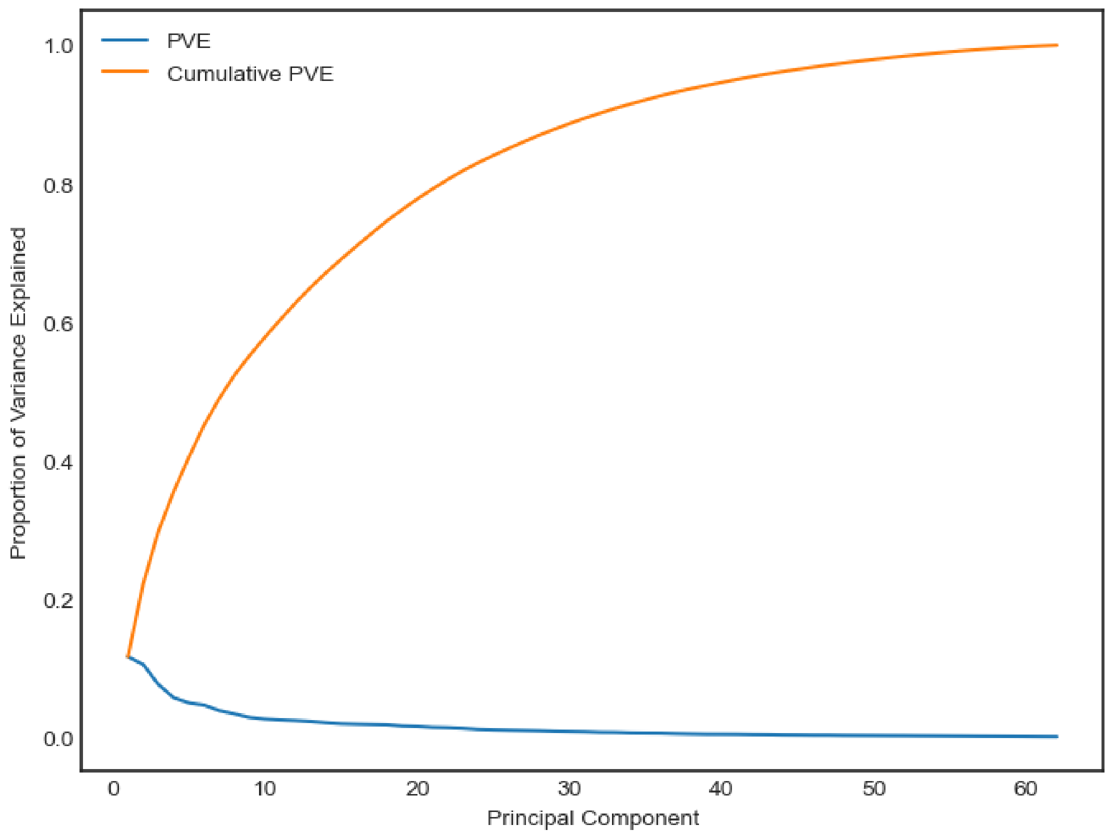
From the above reuslts, we see that the first principal component explains 11.6 % ofthe variance in the data, and the next principal component explains 10.5% of the variance and the third only 7 %. The variance explains tapers off as we move down the last PCA.
Even together, the first five principal components don't explain 50 % of the variance in the data.

So the PCAs will not be able to explain the variance in the data standalone. It can be depicted by the individual vs cumulative Proportion of Variance Explained in scree plot blow.

[15]:
```
# e ii. plot the Proportion of Variance Explained (PVE) of each principal␣
 ↪component (i.e., a scree plot)
# and the cumulative PVE of each principal component.
import matplotlib.pyplot as plt
import numpy as np

pve = pca.explained_variance_ratio_
cumulative_pve = np.cumsum(pve)

fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(range(1, len(pve) + 1), pve, label='PVE')
ax.plot(range(1, len(pve) + 1), cumulative_pve, label='Cumulative PVE')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Proportion of Variance Explained')
ax.legend()
plt.show()
```



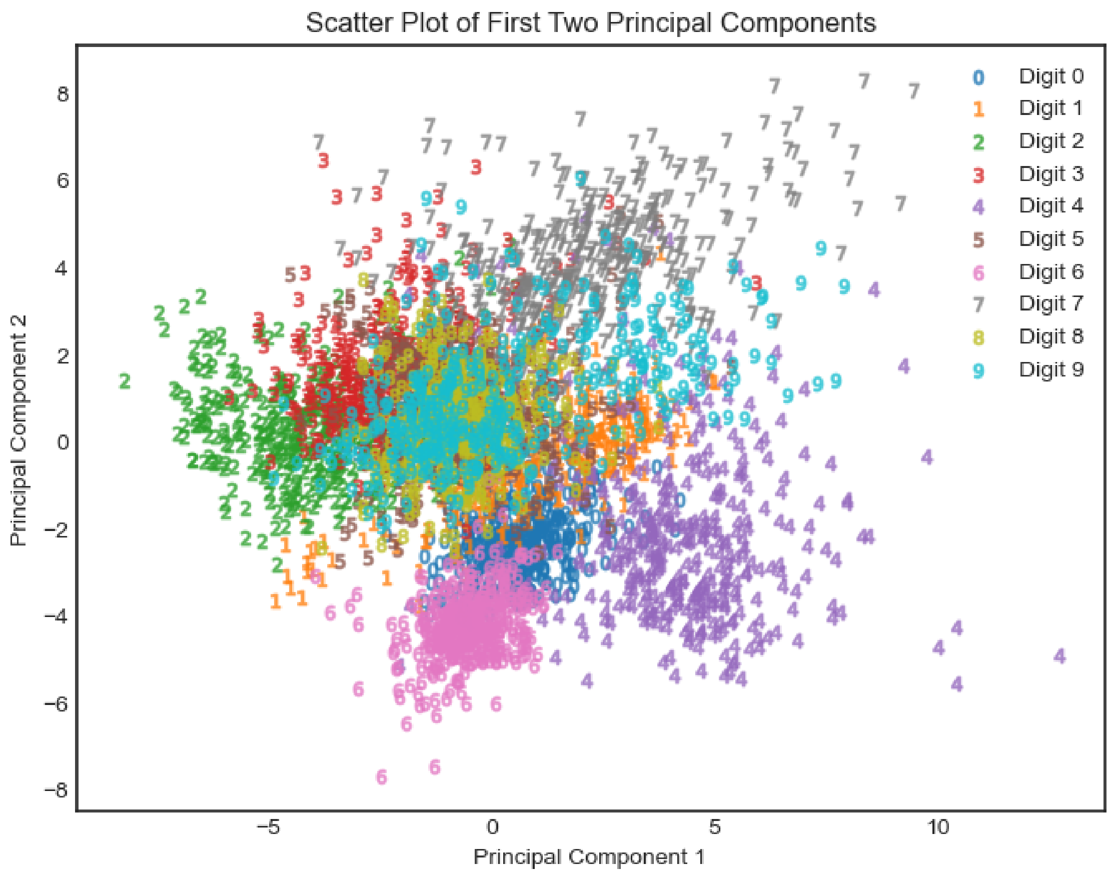
[16]:
```
# merging the first two pcas with the target variable.
pca_1_2_target = pd.concat([df_plot.iloc[:, 0], df_plot.iloc[:, 1],␣
    ↪hand_digits[64]], axis=1)
pca_1_2_target = pca_1_2_target.rename(columns={0: 'pca_1', 1: 'pca_2', 64:␣
    ↪'handwritten_digit'})
pca_1_2_target.head()
```

[16]:

	pca_1	pca_2	handwritten_digit
0	0.021179	-1.506218	0
1	-0.436318	-3.001971	0
2	1.363008	3.160016	7
3	4.499442	0.949555	4
4	-1.199084	-3.264752	6

[17]:
```
# scatter plot for the first two principal components
# the target class variable (i.e., digit number) with different symbols and␣
    ↪colors
fig, ax = plt.subplots(figsize=(8, 6))
for digit in range(10):
    subset = pca_1_2_target[pca_1_2_target['handwritten_digit'] == digit]
```

```
    ax.scatter(subset['pca_1'], subset['pca_2'], label=f'Digit {digit}',␣
 ↪alpha=0.6, marker='${}$'.format(digit))
ax.legend()
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Scatter Plot of First Two Principal Components')
plt.legend()
plt.show()
```



The target class variable, which represents the digit number, is represented by a different symbol– number itself– and color in the scatter plot.

The scatter plot shows how the data points are distributed in the two-dimensional space defined by PC1 and PC2. We observed distinct clusters for some of the digits like 6, 2, and 4. But the clusters are not well defined and bounded. The cluster for digit `6` is better explained by PCA1 while `3` and `2` are better explained by PCA 2.

It can also be due to reason that the two PCAs combine explain less than 25 % variance in the data. So we are not that confident in the effectiveness of the PCA in capturing the underlying structure of the data.