In [1]:
```
218187754%5
```
Out[1]: 4

In [9]:
```
# (Q1) Read the downloaded file into a matrix M(mXn)

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = '//Users//limshikee//Desktop//Janice//digitData4.csv'
M = np.loadtxt(data, delimiter=",")
print (M)
M.shape
```
```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  1.]
 [ 0.  0.  0. ...  9.  0.  2.]
 ...
 [ 0.  1. 11. ...  0.  0.  7.]
 [ 0.  0.  5. ...  0.  0.  4.]
 [ 0.  0.  0. ...  0.  0.  6.]]
```
Out[9]: (1630, 65)

In [10]:
```
# (Q1) Create an empty numpy array X with m rows and n-1 columns

X = np.empty([1630, 64])

print (X)
```
```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 16. 10.  0.]
 [ 0.  1.  0. ...  3. 11. 16.]
 ...
 [ 0.  6.  0. ... 16. 15.  0.]
 [ 0.  0.  3. ... 15. 13.  1.]
 [ 0.  0.  0. ...  0. 15.  3.]]
```

In [11]:
```
# (Q1) Assign all m rows and first n-1 columns of M into X.

i = 0
while(i < 1630):
    j = 0
    while (j < 64):
        X[i,j] = M[i,j]
        j = j+1
    i = i+1

print(X)
```

```
[[ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 10.  0.  0.]
 [ 0.  0.  0. ... 16.  9.  0.]
 ...
 [ 0.  1. 11. ...  0.  0.  0.]
 [ 0.  0.  5. ...  0.  0.  0.]
 [ 0.  0.  0. ... 12.  0.  0.]]
```

In [14]:
```
# (Q1) Create a numpy vector trueLabels and assign n-th column of M
into that.

trueLabels = M[:, 64]

print (trueLabels)
```

```
[0. 1. 2. ... 7. 4. 6.]
```

In [17]:
```
# (Q1) Print dimensions of M, X and trueLabels

print("Shape M = ",np.shape(M))
print("Dimensions M = ",len(M.shape))

print("Shape X = ",np.shape(X))
print("Dimensions X = ",len(X.shape))

print("Shape trueLabels = ",np.shape(trueLabels))
print("Dimensions trueLabels = ",len(trueLabels.shape))
```

```
Shape M =  (1630, 65)
Dimensions M =  2
Shape X =  (1630, 64)
Dimensions X =  2
Shape trueLabels =  (1630,)
Dimensions trueLabels =  1
```

In [120]:
```
# (Q2) Perform K-means clustering with 5 clusters using Euclidean d
istance as similarity measure.

from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5)
kmeans = kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
```

```
[[ 0.00000000e+00  3.92857143e-01  7.59761905e+00  1.31880952e+01
   1.27571429e+01  6.96190476e+00  9.64285714e-01  4.76190476e-03
   7.14285714e-03  3.27619048e+00  1.32857143e+01  9.07380952e+00
   9.95238095e+00  1.06523810e+01  1.50238095e+00  7.14285714e-03
   2.38095238e-03  3.53333333e+00  9.19761905e+00  4.20476190e+00
   8.56904762e+00  9.82619048e+00  1.29285714e+00  1.38777878e-16
   4.55364912e-18  1.80714286e+00  7.00238095e+00  1.03928571e+01
   1.32190476e+01  9.43571429e+00  1.48809524e+00  9.10729825e-18
   0.00000000e+00  3.21428571e-01  2.40476190e+00  5.31666667e+00
   8.70952381e+00  1.18309524e+01  3.37380952e+00  0.00000000e+00
  -5.03069808e-17  3.14285714e-01  1.77857143e+00  1.09047619e+00
   2.67380952e+00  1.16238095e+01  6.23809524e+00  1.19047619e-02
  -5.03069808e-17  8.14285714e-01  7.66428571e+00  5.84523810e+00
   6.50000000e+00  1.25952381e+01  5.93571429e+00  1.45238095e-01
   2.27682456e-18  3.21428571e-01  7.96428571e+00  1.39785714e+01
   1.41309524e+01  9.27857143e+00  2.17857143e+00  2.78571429e-01]
 [ 0.00000000e+00  2.00668896e-02  2.42474916e+00  8.72240803e+00
   1.33444816e+01  1.16622074e+01  4.68227425e+00  7.22408027e-01
   2.25514052e-17  4.11371237e-01  7.45819398e+00  1.13578595e+01
   1.08695652e+01  1.30100334e+01  5.78260870e+00  4.98327759e-01
   1.12757026e-17  1.17056856e+00  7.58862876e+00  6.13043478e+00
   7.37458194e+00  1.25150502e+01  3.84615385e+00  1.47157191e-01
   3.68628739e-18  2.10702341e+00  7.87290970e+00  8.30769231e+00
   1.25250836e+01  1.24280936e+01  3.58193980e+00  6.68896321e-03
   0.00000000e+00  1.54515050e+00  7.76923077e+00  1.10836120e+01
   1.40735786e+01  1.06856187e+01  2.67558528e+00  0.00000000e+00
   0.00000000e+00  8.46153846e-01  4.43143813e+00  8.47826087e+00
   1.15384615e+01  6.65551839e+00  5.85284281e-01  1.38777878e-17
   1.00334448e-02  1.90635452e-01  2.36120401e+00  9.52508361e+00
   9.64882943e+00  4.59197324e+00  5.45150502e-01  3.33066907e-16
   3.34448161e-03  3.34448161e-02  2.94983278e+00  9.06354515e+00
   7.11371237e+00  4.05016722e+00  7.75919732e-01  6.68896321e-03]
 [ 0.00000000e+00 -1.49880108e-15  7.71428571e-01  9.33650794e+00
   1.07682540e+01  1.93015873e+00  1.39682540e-01  7.49400542e-16
   2.34187669e-17  3.49206349e-02  5.35238095e+00  1.41174603e+01
   7.45396825e+00  1.23492063e+00  3.80952381e-01  8.57142857e-02
   1.17093835e-17  6.79365079e-01  1.14698413e+01  1.04507937e+01
   2.55238095e+00  2.31746032e+00  1.56507937e+00  1.23809524e-01
   3.17460317e-03  3.33650794e+00  1.39936508e+01  7.02857143e+00
   5.38095238e+00  5.67301587e+00  2.63174603e+00  6.34920635e-03
   0.00000000e+00  5.95238095e+00  1.46984127e+01  1.11873016e+01
   1.24984127e+01  1.22507937e+01  4.14603175e+00  0.00000000e+00
   4.12698413e-02  4.01269841e+00  1.32285714e+01  1.14222222e+01
   9.65079365e+00  1.05206349e+01  5.97777778e+00  1.30158730e-01
   3.17460317e-02  6.34920635e-01  6.87301587e+00  1.02539683e+01
   9.35555556e+00  8.05714286e+00  5.80634921e+00  3.61904762e-01
   1.95156391e-18  1.26984127e-02  9.77777778e-01  9.47619048e+00
   1.39269841e+01  7.99047619e+00  2.47936508e+00  9.84126984e-02]
 [ 0.00000000e+00  7.43648961e-01  8.30946882e+00  1.38429561e+01
   1.10300231e+01  4.76212471e+00  1.10623557e+00  3.00230947e-02
   1.61662818e-02  3.73210162e+00  1.22540416e+01  1.28429561e+01
   1.18406467e+01  6.62124711e+00  1.00461894e+00  1.38568129e-02
   9.23787529e-03  3.88452656e+00  9.59122402e+00  7.95150115e+00
   1.03556582e+01  5.03233256e+00  4.31870670e-01  2.30946882e-03
   2.30946882e-03  1.80600462e+00  6.96766744e+00  1.09838337e+01
   1.19468822e+01  3.06466513e+00  3.25635104e-01  9.10729825e-18
   0.00000000e+00  7.18244804e-01  5.53810624e+00  1.24387991e+01
```

```
  1.13418014e+01  1.95150115e+00  2.30946882e-01  0.00000000e+00
 -5.37764278e-17  6.07390300e-01  5.98614319e+00  1.08845266e+01
  1.01154734e+01  2.81293303e+00  3.69515012e-01  2.30946882e-03
 -5.37764278e-17  9.30715935e-01  9.15473441e+00  1.22471132e+01
  1.10831409e+01  6.30023095e+00  3.01616628e+00  4.27251732e-01
  2.27682456e-18  6.85912240e-01  8.85219400e+00  1.36605081e+01
  1.10300231e+01  6.36489607e+00  3.68822171e+00  1.15704388e+00]
[ 0.00000000e+00  2.45398773e-02  4.07975460e+00  1.31226994e+01
  1.14417178e+01  2.98159509e+00  3.06748466e-02 -2.49800181e-16
  1.30104261e-17  9.14110429e-01  1.25889571e+01  1.32331288e+01
  1.12576687e+01  1.14355828e+01  1.02453988e+00  2.35922393e-16
  6.50521303e-18  3.80368098e+00  1.40920245e+01  5.01226994e+00
  2.04907975e+00  1.20613497e+01  3.69938650e+00  8.32667268e-17
  6.50521303e-19  5.30674847e+00  1.25337423e+01  2.00613497e+00
  2.51533742e-01  9.08588957e+00  6.60736196e+00  1.30104261e-18
  0.00000000e+00  5.86503067e+00  1.13803681e+01  8.46625767e-01
  3.68098160e-02  8.91411043e+00  7.19631902e+00  0.00000000e+00
  2.60208521e-17  3.49693252e+00  1.31901840e+01  1.52147239e+00
  1.33742331e+00  1.12576687e+01  5.83435583e+00 -4.51028104e-17
  2.60208521e-17  7.60736196e-01  1.30122699e+01  9.58895706e+00
  9.95092025e+00  1.32453988e+01  2.56441718e+00  2.45398773e-02
  3.25260652e-19  6.13496933e-03  4.12883436e+00  1.35828221e+01
  1.35214724e+01  5.65030675e+00  3.43558282e-01  1.84049080e-02]
]
[4 3 1 ... 3 3 2]
```

In [172]:

```
# (Q2) Evaluate the clustering performance using adjusted rand inde
x (ARI) and adjusted mutual information.

from sklearn import metrics

ARI = metrics.adjusted_rand_score(trueLabels, kmeans.labels_)
print("\n Evaluate Clustering Performance using Adjusted Rand Index
- ARI = " + str(ARI) + '\n')

AMI = metrics.adjusted_mutual_info_score(trueLabels, kmeans.labels_
)
print("\n Evaluate Clustering Performance using Adjusted Mutual Inf
ormation - AMI = " + str(AMI) + '\n')
```

```
 Evaluate Clustering Performance using Adjusted Rand Index - ARI =
0.3554936668064773


 Evaluate Clustering Performance using Adjusted Mutual Information
- AMI = 0.4510510642276656


/Users/limshikee/anaconda3/lib/python3.7/site-packages/sklearn/met
rics/cluster/supervised.py:732: FutureWarning: The behavior of AMI
will change in version 0.22. To match the behavior of 'v_measure_s
core', AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

In [175]:

```
# (Q2) Report the clustering performance averaged over 50 random in
itializations of K-means.

kmeans50 = KMeans(n_clusters=5,init = 'random', n_init=50)
kmeans50 = kmeans50.fit(X)

ARI50 = metrics.adjusted_rand_score(trueLabels, kmeans50.labels_)
print("\n Evaluate Clustering Performance using Adjusted Rand Index
- ARI - 50 initializations = " + str(ARI50) + '\n')

AMI50 = metrics.adjusted_mutual_info_score(trueLabels, kmeans50.lab
els_)
print("\n Evaluate Clustering Performance using Adjusted Mutual Inf
ormation - AMI - 50 initializations = " + str(AMI50) + '\n')
```

```
 Evaluate Clustering Performance using Adjusted Rand Index - ARI -
50 initializations = 0.3600881270478822


 Evaluate Clustering Performance using Adjusted Mutual Information
- AMI - 50 initializations = 0.4573363087634137 5


/Users/limshikee/anaconda3/lib/python3.7/site-packages/sklearn/met
rics/cluster/supervised.py:732: FutureWarning: The behavior of AMI
will change in version 0.22. To match the behavior of 'v_measure_s
core', AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

In [198]:

```
# (Q3) If we have an ARI value of 0.7 after a single run of K-means
clustering
# with 'Kmeans++' initializaton for any data set then what will be
the value
# of averaged ARI over 20 repetitions

print("\n The ARI value is still remain around 0.7 ")
```

```
 The ARI value is still remain around 0.7
```

In [183]:

```
# (Q4) Repeat K-means clustering with 5 clusters using a similarity
measure other than Euclidean distance.

from sklearn.preprocessing import scale
Xnorm1 = scale(X)

kmeans1 = KMeans(n_clusters=5)
kmeans1 = kmeans1.fit(X)

ARI_1 = metrics.adjusted_rand_score(trueLabels, kmeans1.labels_)
print("\n Evaluate Clustering Performance using Adjusted Rand Index
- ARI = " + str(ARI_1) + '\n')

AMI_1 = metrics.adjusted_mutual_info_score(trueLabels, kmeans.label
s_)
print("\n Evaluate Clustering Performance using Adjusted Mutual Inf
ormation - AMI = " + str(AMI_1) + '\n')
```

```
 Evaluate Clustering Performance using Adjusted Rand Index - ARI =
0.35600734972402126


 Evaluate Clustering Performance using Adjusted Mutual Information
- AMI = 0.4510510642276656


/Users/limshikee/anaconda3/lib/python3.7/site-packages/sklearn/met
rics/cluster/supervised.py:732: FutureWarning: The behavior of AMI
will change in version 0.22. To match the behavior of 'v_measure_s
core', AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

In [184]:
```
# (Q4) Evaluate the clustering performance over 50 random initializations of K-means using adjusted rand index and adjusted mutual information.

kmeans50_1 = KMeans(n_clusters=5,init = 'random', n_init=50)
kmeans50_1 = kmeans50_1.fit(X)

ARI50_1 = metrics.adjusted_rand_score(trueLabels, kmeans50_1.labels_)
print("\n Evaluate Clustering Performance using Adjusted Rand Index - ARI - 50 initializations = " + str(ARI50_1) + '\n')

AMI50_1 = metrics.adjusted_mutual_info_score(trueLabels, kmeans50_1.labels_)
print("\n Evaluate Clustering Performance using Adjusted Mutual Information - AMI - 50 initializations = " + str(AMI50_1) + '\n')
```

```
 Evaluate Clustering Performance using Adjusted Rand Index - ARI - 50 initializations = 0.36104613475409014


 Evaluate Clustering Performance using Adjusted Mutual Information - AMI - 50 initializations = 0.4581558237307746


/Users/limshikee/anaconda3/lib/python3.7/site-packages/sklearn/metrics/cluster/supervised.py:732: FutureWarning: The behavior of AMI will change in version 0.22. To match the behavior of 'v_measure_score', AMI will use average_method='arithmetic' by default.
  FutureWarning)
```

In [200]:
```
# (Q4) Report the clustering performance and compare it with the results obtained in step 2.

print("\n The clustering performance are similar with the results obtained in step 2")
```

```
 The clustering performance are similar with the results obtained in step 2
```

In [179]:

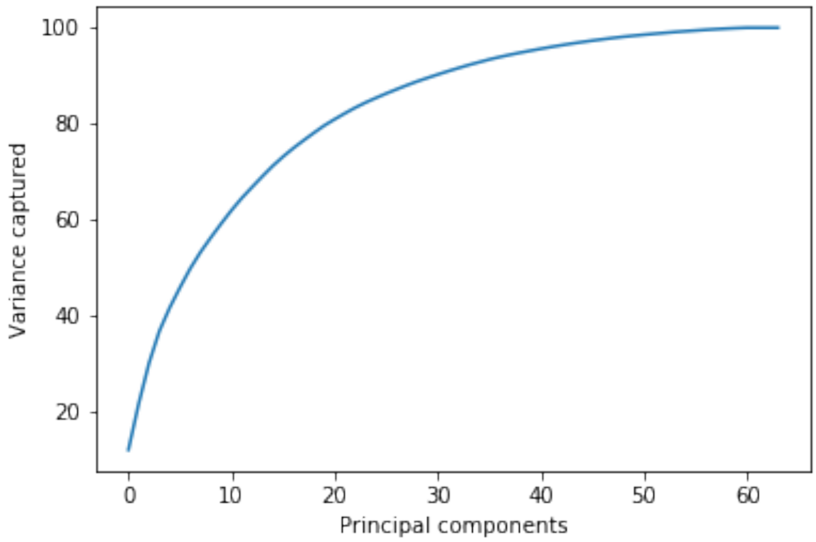
```
# (Q5)Perform PCA

from sklearn.preprocessing import scale
Xnorm = scale(X)

from sklearn.decomposition import PCA
pca = PCA(n_components=64)
pca.fit(Xnorm)

PCA(copy=True, iterated_power='auto', n_components=64, random_state
=None,
  svd_solver='auto', tol=0.0, whiten=False)

var= pca.explained_variance_ratio_
print(var)
```

```
[1.20608732e-01 9.63229887e-02 8.42681690e-02 6.66352602e-02
 4.86211798e-02 4.20827929e-02 3.96836514e-02 3.42693733e-02
 3.02903118e-02 2.88163236e-02 2.78364637e-02 2.53925781e-02
 2.28772661e-02 2.24402145e-02 2.16348189e-02 1.93343122e-02
 1.77067195e-02 1.63087572e-02 1.57401303e-02 1.50818353e-02
 1.30423882e-02 1.24482710e-02 1.17904654e-02 1.06365640e-02
 9.59000073e-03 9.17589115e-03 8.74162193e-03 8.48082977e-03
 7.97359153e-03 7.26934557e-03 7.12913302e-03 6.91295509e-03
 6.50659107e-03 6.40118408e-03 5.82776732e-03 5.65540658e-03
 5.13330609e-03 4.73119937e-03 4.38337823e-03 4.09316255e-03
 3.98601200e-03 3.84059964e-03 3.55363093e-03 3.38817861e-03
 3.23268844e-03 3.04449343e-03 2.84774973e-03 2.72799223e-03
 2.50463675e-03 2.24030387e-03 2.07492326e-03 1.92359204e-03
 1.90663428e-03 1.82354302e-03 1.64130391e-03 1.53319975e-03
 1.46238989e-03 1.34704249e-03 1.22007102e-03 1.04401439e-03
 7.82069253e-04 3.16440015e-33 6.42570347e-34 6.31800977e-34]
```

In [180]:
```
# (Q5)  Plot the captured variance with respect to increasing laten
t dimensionality

var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*
100)
print(var1)
plt.plot(var1)
plt.xlabel("Principal components")
plt.ylabel("Variance captured")
```

```
[12.06 21.69 30.12 36.78 41.64 45.85 49.82 53.25 56.28 59.16 61.94
64.48
 66.77 69.01 71.17 73.1  74.87 76.5  78.07 79.58 80.88 82.12 83.3
84.36
 85.32 86.24 87.11 87.96 88.76 89.49 90.2  90.89 91.54 92.18 92.76
93.33
 93.84 94.31 94.75 95.16 95.56 95.94 96.3  96.64 96.96 97.26 97.54
97.81
 98.06 98.28 98.49 98.68 98.87 99.05 99.21 99.36 99.51 99.64 99.76
99.86
 99.94 99.94 99.94 99.94]
```

Out[180]:
```
Text(0, 0.5, 'Variance captured')
```

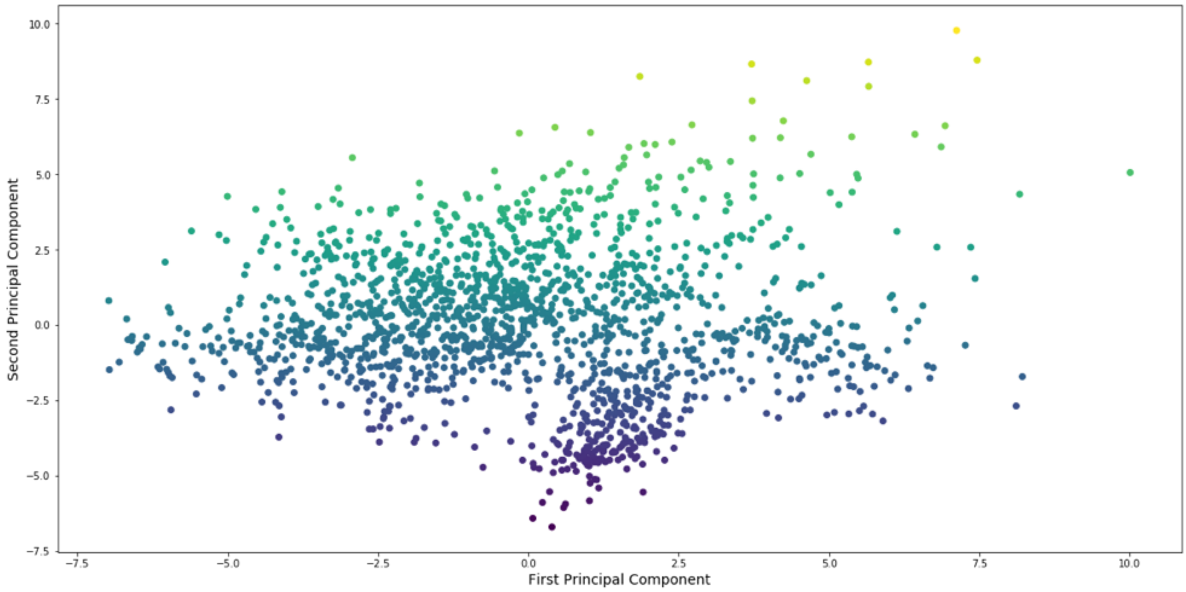


In [181]:
```
# (Q5)What is the minimum dimension that captures at least 95% vari
ance?

pca = PCA(n_components=40)
Zred = pca.fit_transform(Xnorm)
print(Zred.shape)
```

```
(1630, 40)
```

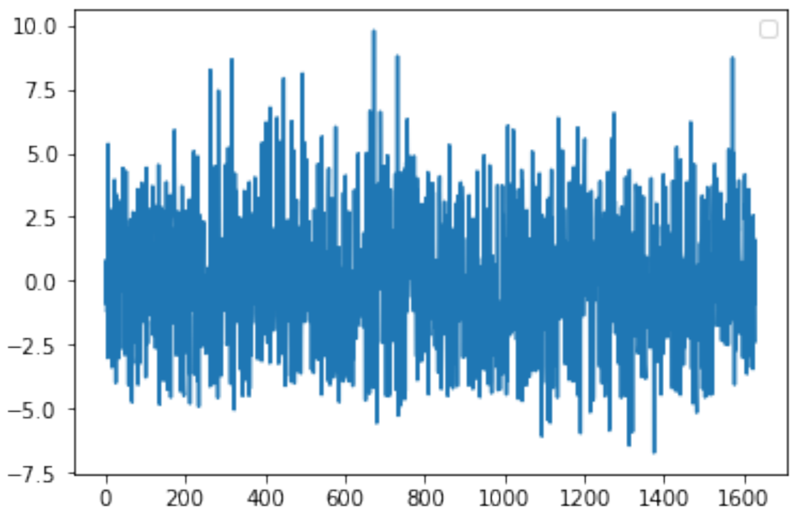
In [197]:
```python
# (Q6) Create a scatter plot with each of the total rows of X proje
cted onto the first two principal components
# (Q6) Your plot must use a different color for each digit and incl
ude a legend.

import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

plt.figure(figsize=(20,10))
plt.scatter(Zred[:,0], Zred[:,1], c = Zred[:,1])
plt.xlabel("First Principal Component",fontsize=14)
plt.ylabel("Second Principal Component",fontsize=14)

print("scatter plot of principal component 1 and 2 -")
plt.show()
plt.plot(Zred[:,1])
plt.legend()
```

```
scatter plot of principal component 1 and 2 -
```




```
No handles with labels found to put in legend.
```

Out[197]: <matplotlib.legend.Legend at 0x112387940>



In [ ]: