Report

DIMENSIONALITY REDUCTION

Part A:

Global Mean : dimensions of global mean is 784 X 1. Partial result is shown below :

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 2.10000000e-03 7.83333333e-03 3.60000000e-03 1.50000000e-04
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 2.66666667e-04 9.16666667e-04 9.28333333e-03 2.42833333e-02
 4.37166667e-02 6.41000000e-02……]
```

Covariance of the data : dimensions are 784 X 784 . Partial result is shown below:

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Please note that all the elements of covariance matrix are not zero.

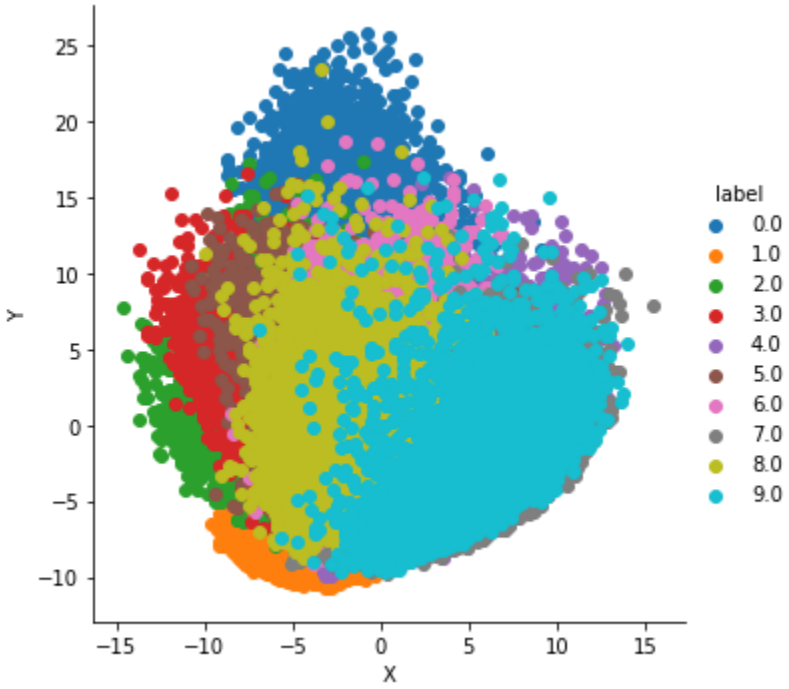
```
print(covariance_matrix[45][45])

29.305130096335084
```

Part C :

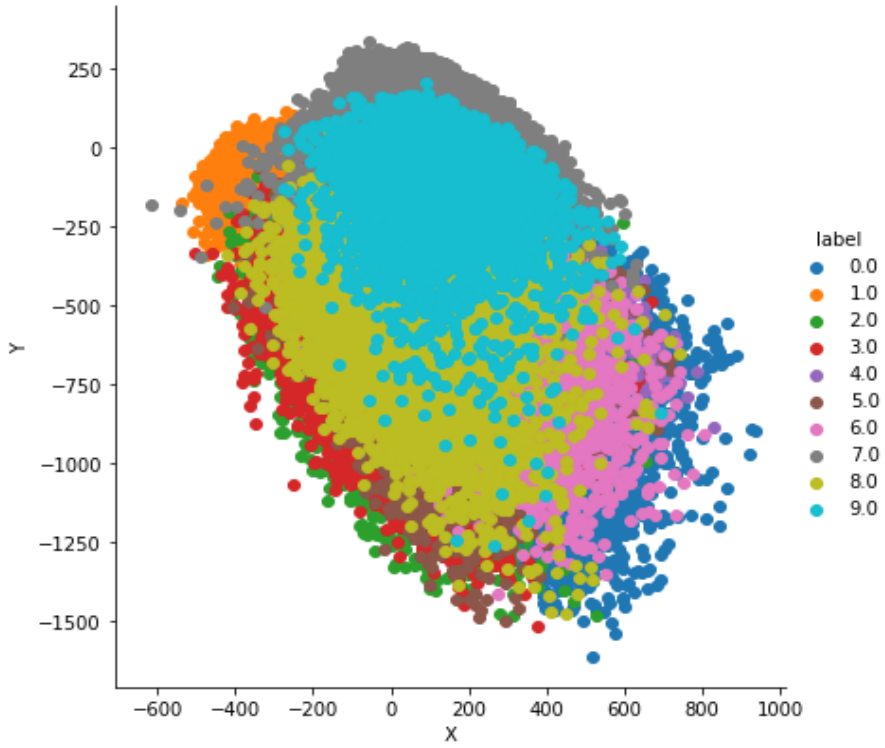
Please note that I have transformed the data into 2 dimensions and then plotted for PCA and FDA.

Scatter plot for PCA



Please note that the PCA technique does not take class labels into consideration but just for the sake of drawing scatter plot and showing different class elements I have used class labels.

Scatter plot for FDA



Part e :

With 95% eigen energy total number of eigen vectors taken were **153**.

```
Enter eigen energy value in fraction0.95
153
```

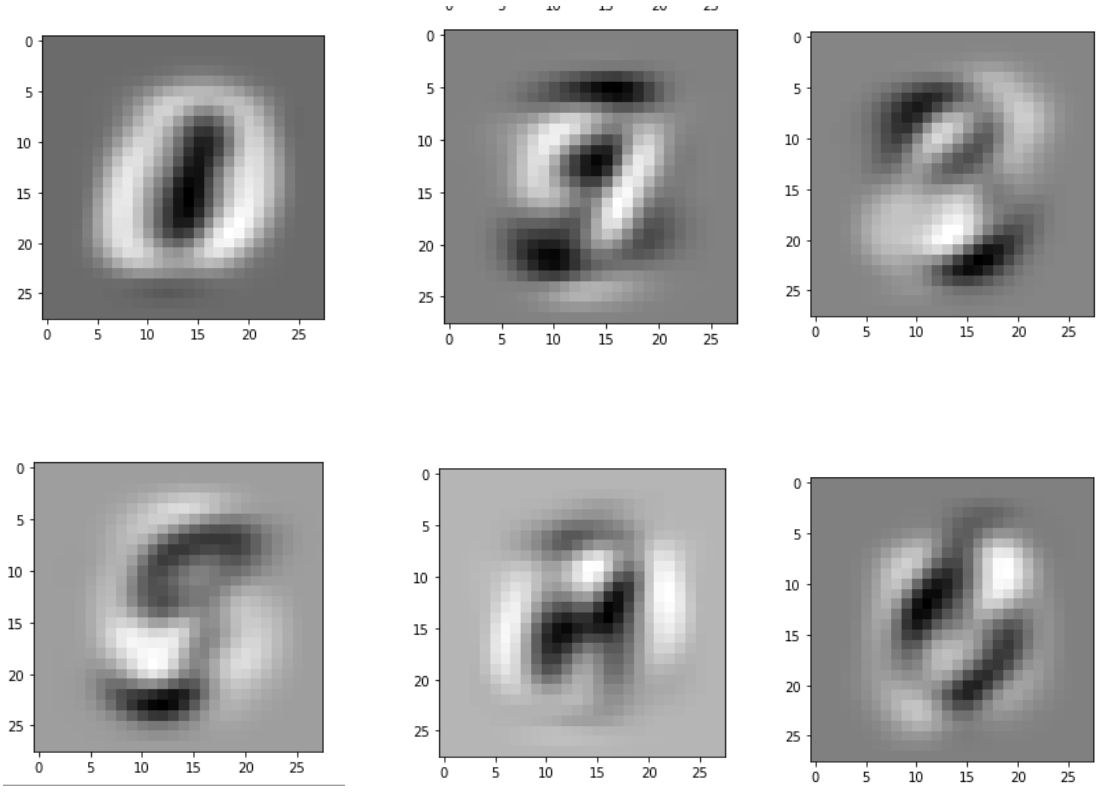
```
Size of the Test Data  : 10K points
```

```
###0.95 eigen energy
Accuracy=correct/10000
print(Accuracy)
```

```
0.8748
```

Total accuracy reported with 95% eigen energy is **87.48%**

Part F: Below are the six images of the eigen vectors corresponding to six highest eigen values.



We already know the fact that each eigen vector is a linear combination of the orginal dimensions which is an image itself.

Each Eigen vector has an image associated with it. From the image of the first eigen vector we can see that the vector is able to push all the ONEs to one side and all the ZEROs to another side.

Where ever there is a darker/fairer region in an eigen vector image this represent that some of the digits ends up to one side.

Moreover each eigen vector image keeps on getting blurred because of the simple fact that total variance of the data explained by the vector decrease.

Part G: Output screenshot

```
###0.95 eigen energy
Accuracy=correct/10000
print(Accuracy)
```

```
0.8748
```

```
[ ] ###0.70 eigen energy
Accuracy=correct/10000
print(Accuracy)
```

```
0.8544
```

```
[ ] ###0.90 eigen energy
Accuracy=correct/10000
print(Accuracy)
```

```
0.8764
```

```
[ ] ###0.99 eigen energy
Accuracy=correct/10000
print(Accuracy)
```

```
0.8711
```

Accuracy Summary Table:

Eigen Energy Taken	Accuracy Reported
95%	87.48%
70%	85.44%
90%	87.64%
99%	87.11%

First of all we sort out eigen vectors in decreasing order of corresponding eigen values.

Each eigen vector explains some part of the total variance of the data. Below matrix shows part of total variance explained by each of 784 eigen vectors whose corresponding eigen values are sorted in decreasing order.

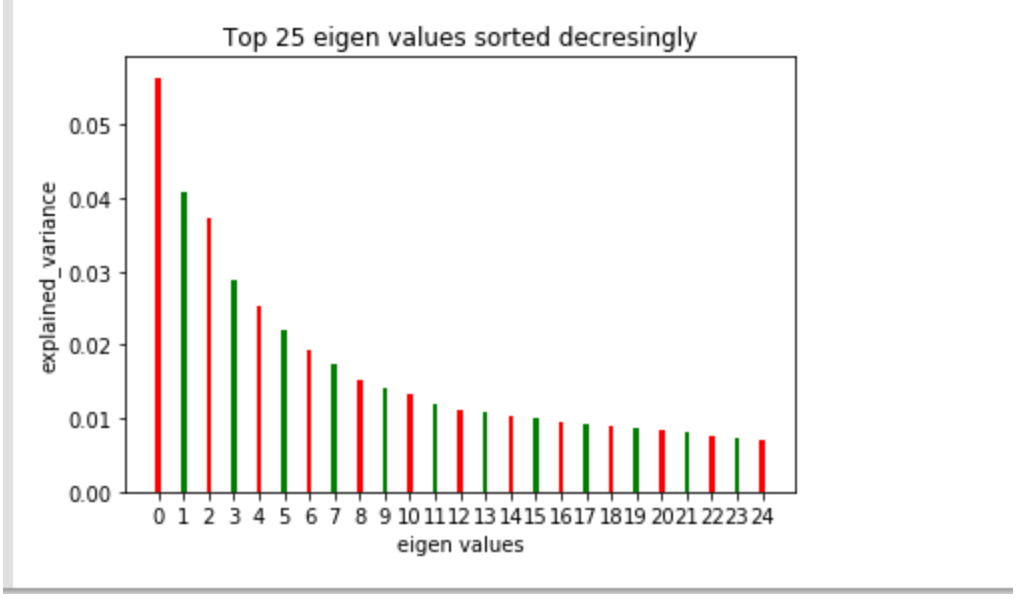
```
[5.64671692e-02 4.07827199e-02 3.73938042e-02 2.88511485e-02
 2.52110863e-02 2.19426996e-02 1.92334439e-02 1.74579923e-02
 1.53509230e-02 1.40171960e-02 1.34174302e-02 1.20374194e-02
 1.11456955e-02 1.08992356e-02 1.02864922e-02 9.94486564e-03
 9.36383280e-03 9.21045666e-03 8.93436778e-03 8.69912619e-03
 8.27363019e-03 8.03417369e-03 7.64845500e-03 7.41772464e-03
 7.15292868e-03 6.91846831e-03 6.84135964e-03 6.56674546e-03
 6.31676724e-03 6.12919839e-03 5.96255295e-03 5.87716416e-03
 5.71591699e-03 5.62307416e-03 5.54682002e-03 5.38418374e-03
 5.31182250e-03 5.19605602e-03 5.08211255e-03 4.80005571e-03
 4.76455820e-03 4.69139360e-03 4.54348956e-03 4.51345787e-03        …….]
```

From the above table it is clear that if I want to preserve 95% of total variance then I'll have to take first 330 eigen vectors.

Please note that with increase in eigen energy we will consider more eigen vectors which will implicitly increase accuracy of our prediction (using LDA).This can easily be concluded from the Accuracy Summary Table .Accuracy increases when eigen energy increases from 70% to 90%.

When we increase eigen energy below this point then we consider more eigen vectors which implicitly adds noise to the data points .These eigen vectors are of least importance. Therefore there is some decrease in the accuracy reported for 95% and 99% of the eigen energy.

Top 25 eigen values sorted in decreasing order are plotted against their explained_varience values as below:



Part h

```
#Accuracy for FDA followed by LDA
Accuracy=correct/10000
print(Accuracy)
```

```
0.7979
```

FDA is applied followed by LDA for classification of test images. This is giving accuracy of 79.79%.

Part I

```
print((correct/10000)*100)
```

```
21.0
```

Technique	**Accuracy Reported**
PCA followed by LDA(95% eigen energy)	87.48%
FDA followed by LDA	79.79%
PCA →FDA →LDA	21%

From the above table we can conclude that PCA (**unsupervised model**) works best among all three different flavors with accuracy 87.48%

Accuracy decreases when we used FDA (**supervised model**) which comes out to be 79.79%

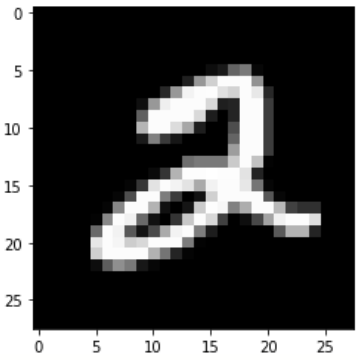
This further decreases when we perform PCA then FDA followed by LDA which comes out to be 21%,least among all three.

NOISE REDUCTION

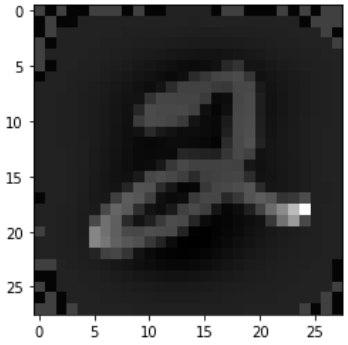
Part A :

Gaussian noise with mean =0 and variance=10 is added to each image as follows:

```
mean = 0
var = 10
sigma = var ** 0.5
gaussian = np.random.normal(mean, sigma, (28, 28))
```

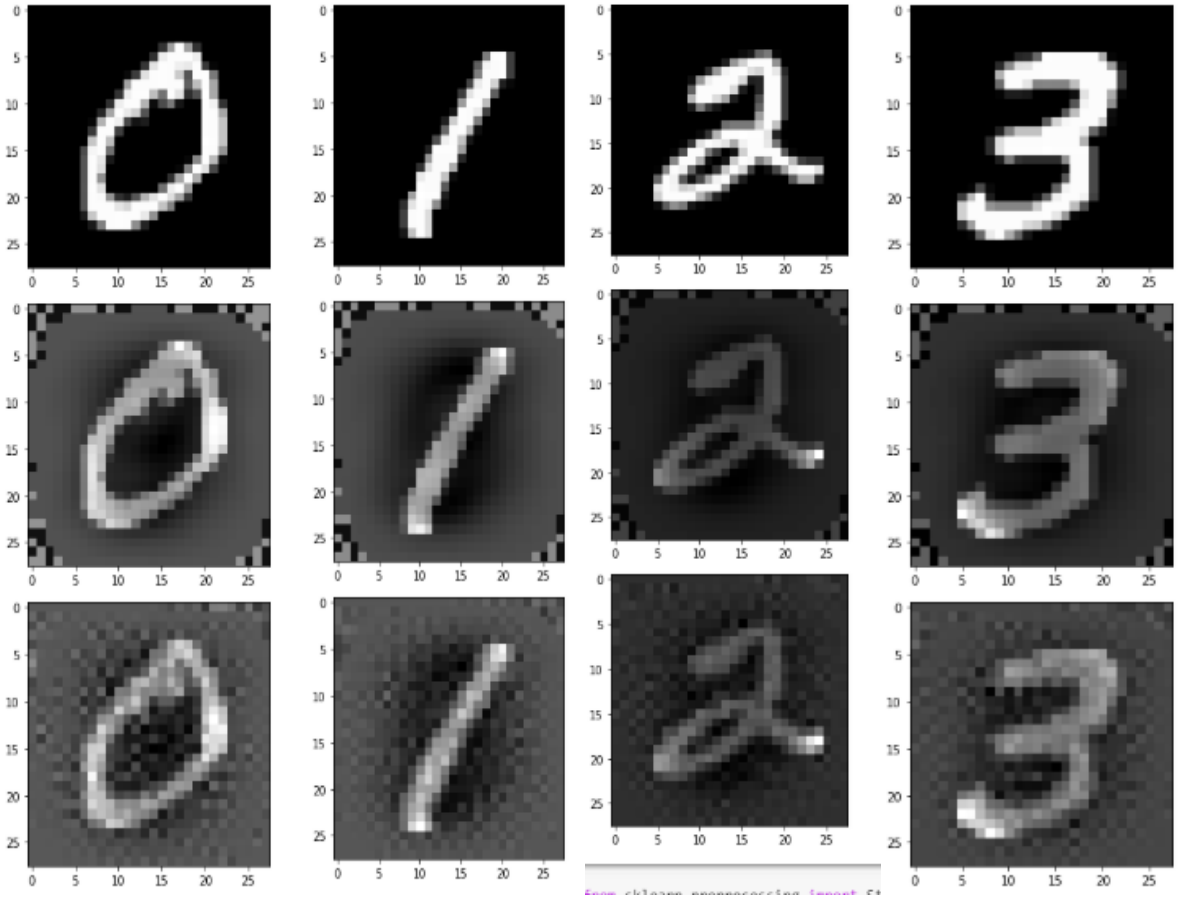


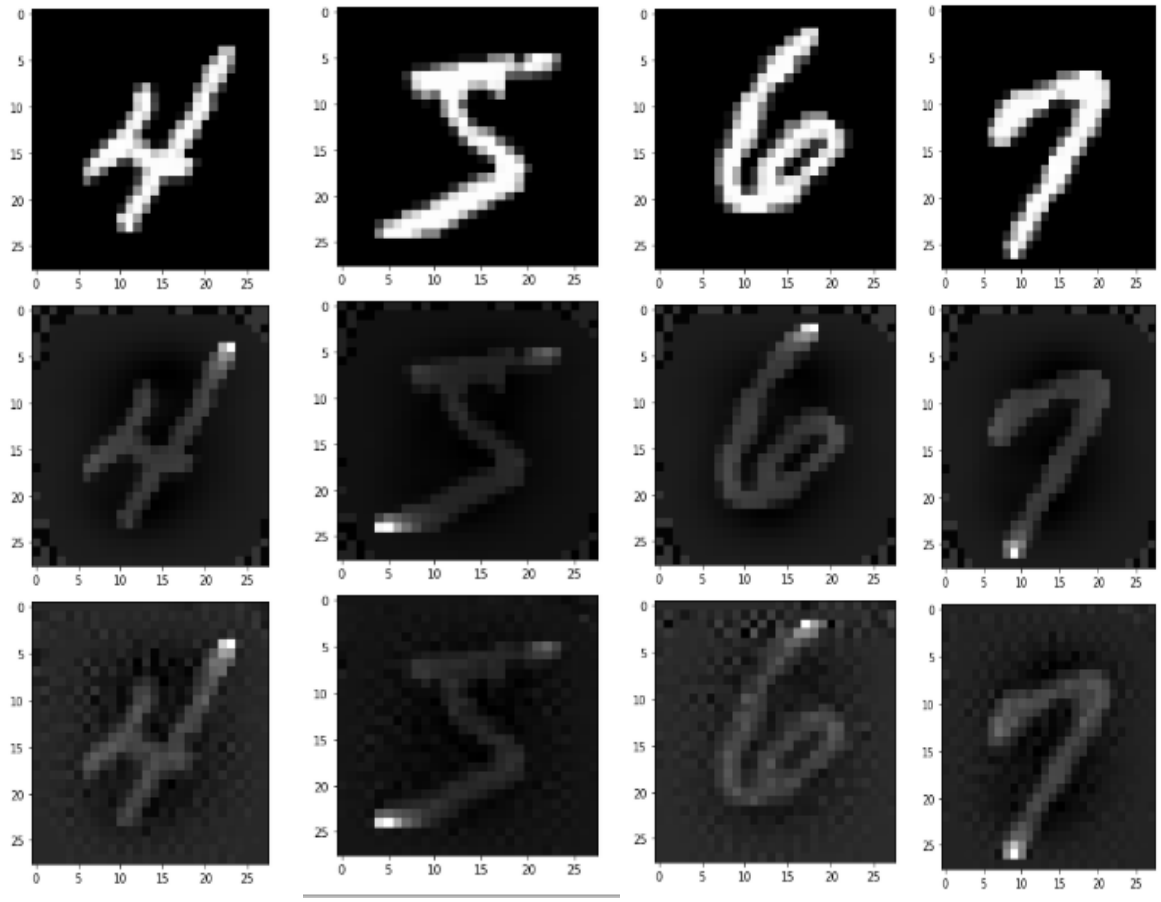
(Original Image)

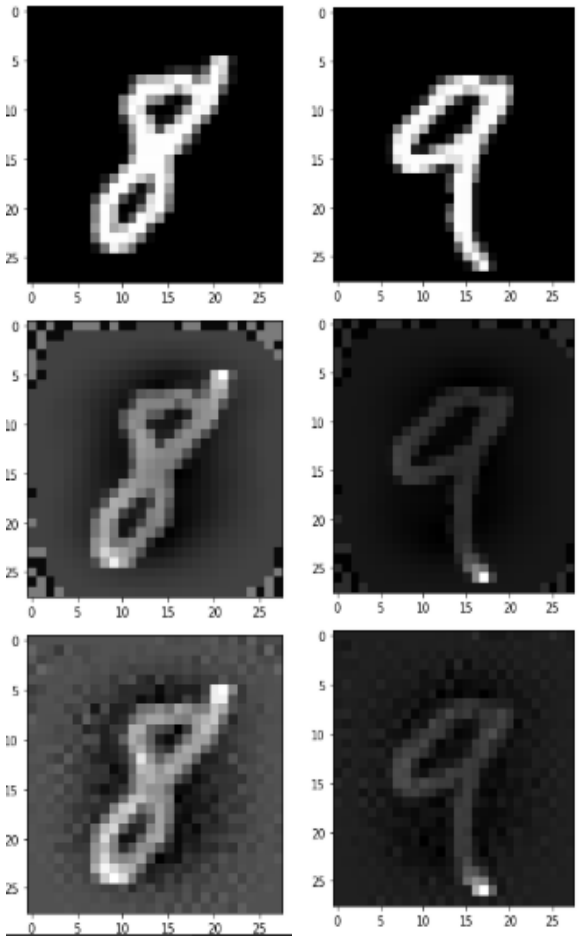


(Noisy Image)

Part C:







Part D:

Starting 458 PCA components were able to explain 98% of the covariance of the noisy data. Hence I have used 458 PCA components for noise reduction. These components are sufficient to distinguish between noisy image and noise reduced image.

```
from sklearn.preprocessing import StandardScaler
import pandas as pd
x = StandardScaler().fit_transform(R1)
x = pd.DataFrame(x)

from sklearn.decomposition import PCA
pca = PCA()
x_pca = pca.fit_transform(x)
x_pca = pd.DataFrame(x_pca)

explained_variance = pca.explained_variance_ratio_
sum=0
c=0
for i in range(0,len(explained_variance)):
  sum=sum+explained_variance[i]
  if sum>0.98:
    c=i
    break
print(c)

print(type(x_pca[0]))
```

```
458
<class 'pandas.core.series.Series'>
```