Follow Geron’s pp.79-81 (Chapter 3: MNIST) and pp.211-216 (Chapter 8: PCA).

**PROJECT 1: PCA**:

**Question**:

1. There are 70000 images of hand-written digits. The first 60000 are the training examples and the last 10000 testing.
2. Dimension reduction: from 784 = 28 x 28 to 154.
3. Show reconstruction results for some images.
4. Show what features some of the code elements represent.
5. Show the code c for some image x.
6. Reduce the number of bits for cj to a few bits and see the reconstructions.  (numpy.round(a \* 4, 0) / 4.0 may work.)

Write a report including your code, results, and discussions.

**Answer:**

* The main aim of the principal component analysis is to reduce the dimensionality of the data set.
* The data set will constitute of many variables which are correlated with each other.
* Basically we are combining these variables to derive new components which will produce a simpler description of the system.
* We are going to use MNIST Data; we can also use the the dataset available in Kaggle.
* Handwritten datasets can be obtained from the following websites:

<https://www.kaggle.com/c/digit-recognizer/data>’

<http://yann.lecun.com/exdb/mnist/>

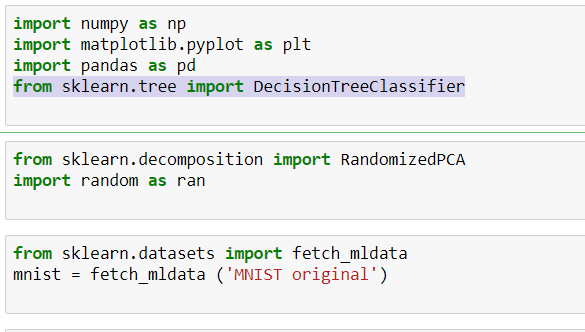
* Machine learning package that will be used for this project is Scikit learn.
* The internal structure of MNIST data will be revealed through PCA method. We will observe this as we go through the project.
* When the PCA is run on huge data set it will provide the user with a pictographic representation of the information. Basically, the original objects are converted to a low dimensional shadow. A 2-dimensional scatter plot will be created if we keep only the 1st and 2nd components. Due to this fact, PCA is used heavily for build predictive models through exploratory data analysis. Ideally, it’s recommended to keep only the first few principal components. The reason is through this we can decrease the data dimensionality and it would be much easier to visualize the structure of the data. [1]

The code is explained as below:

**1)**

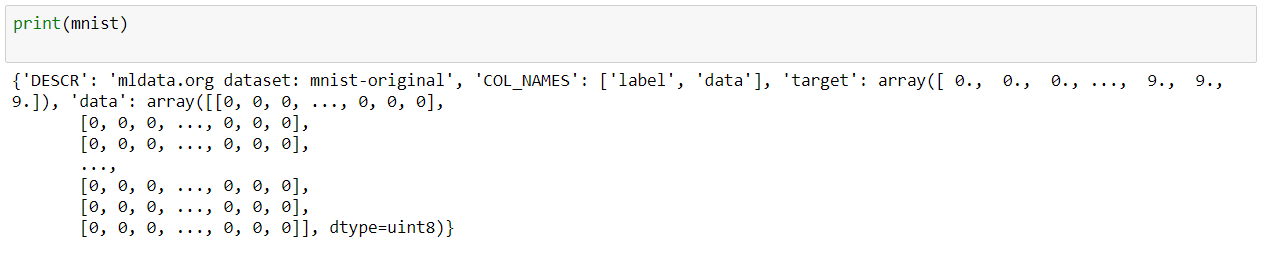
Initially the imported few packages.

* From sklearn.tee we will import DecisionTreeClassifier which is the classifier or the machine learning model. The DecisonTreeClassifier won’t be used for this project.
* Plotting the image is being done by the matplotlib.pylot
* Pandas is imported using pd and it helps in loading the training dataset

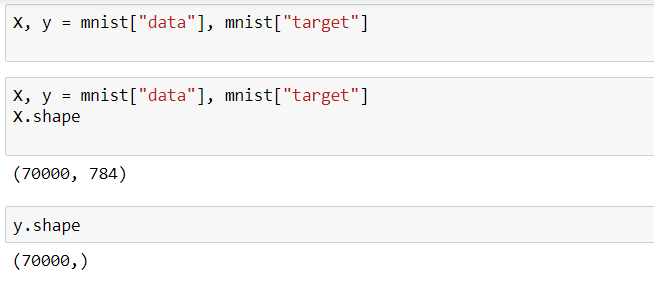


There are 70000 images of hand-written digits in MNIST. It will be fetched and stored in mnist variable.

The value of each pixel and label is printed below.



X.shape and y.shape values are generated below.

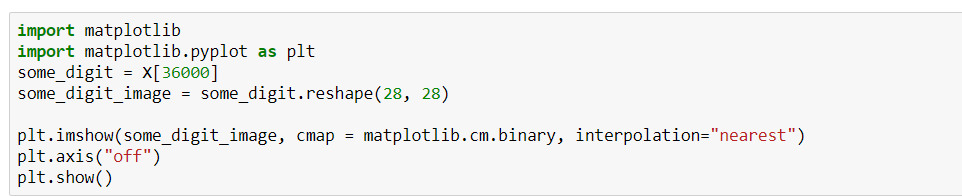


**2)**

Dimension reduction: from 784 = 28 x 28 to 154.

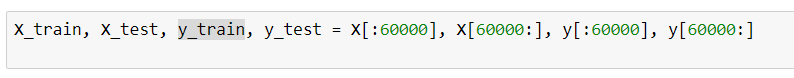
We are going from 784 🡪 154 to train machine learning models.

MNIST handwritten data set has around 60,000 examples where the digits are ranging from 0 to 9. We are going to observe each of these examples is about 28x28 pixel image.



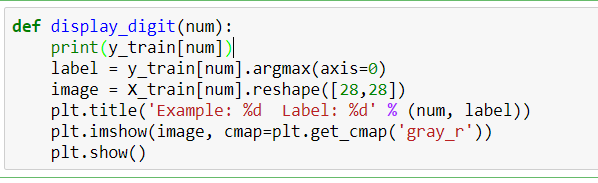


The first 60000 are the training examples and the last 10000 testing.

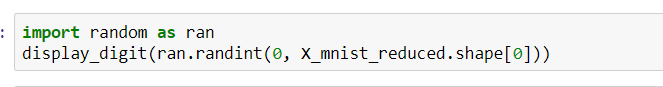


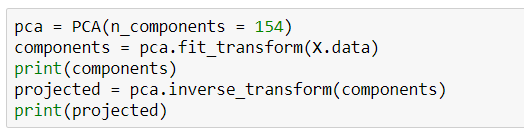
**3)** Show reconstruction results for some images.

This code is to display the digit (Basically a function)

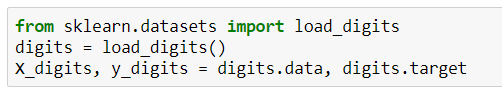


Randomly a digit will be called and displayed. Will be reconstructed image.



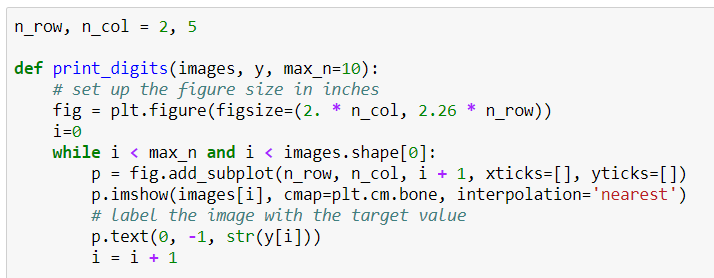


We can load the digits using the following method:

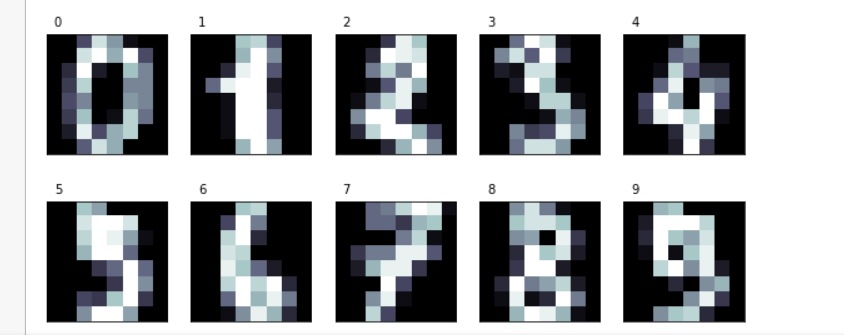


**4)** Show what features some of the code elements represent.

cm.bone will make sure we are extracting the features from 0->9







**5)**

We will set n\_components to 154; We want to transform instances of 784 features to instances of 154 features.

We are going from 784 🡪 154 to train machine learning models.

X70,000x784 V784x154= X`70,000x154

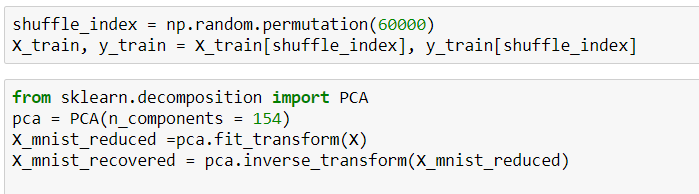
154 will have the top eigen vectors.

Going from 784 🡪 154 will be the original

Covariance matrix of X will be given by following equation

Covariance Matrix70,000x70,000 = XT X

Fundamentally, variance maximization. 784->154 how much of variance is explained by just 154

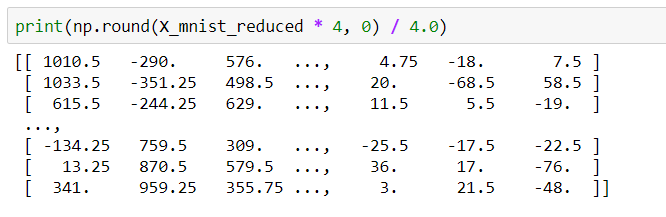




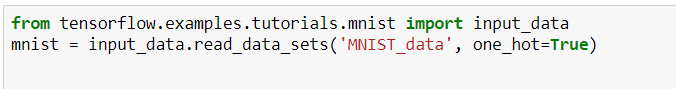
Show the code c for some image x.



**6)**Reduce the number of bits for cj to a few bits and see the reconstructions.

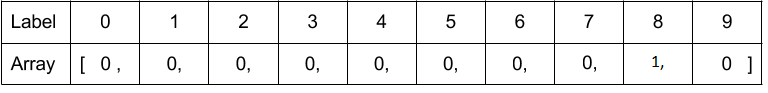


I have tried another way of using encoding to reduce the number of bits.



We have now flattened it to an array with 784 values which will now represent each pixel’s intensity. These examples have to be flattened for TensorFlow, so the tool can understand/make sense of the digits linearly. We have loaded the X\_train with 60,000 examples each of which has 784 pixels. X\_train variable is a 60,000 row with 784 column matrix.

The associated labels for the x\_train is represented in y\_train; X\_train has the data information. Through one hot encoding the value of the label is stored in a 1x10 binary array. One represents the digit. An example is provided below, the array represents the value 8:

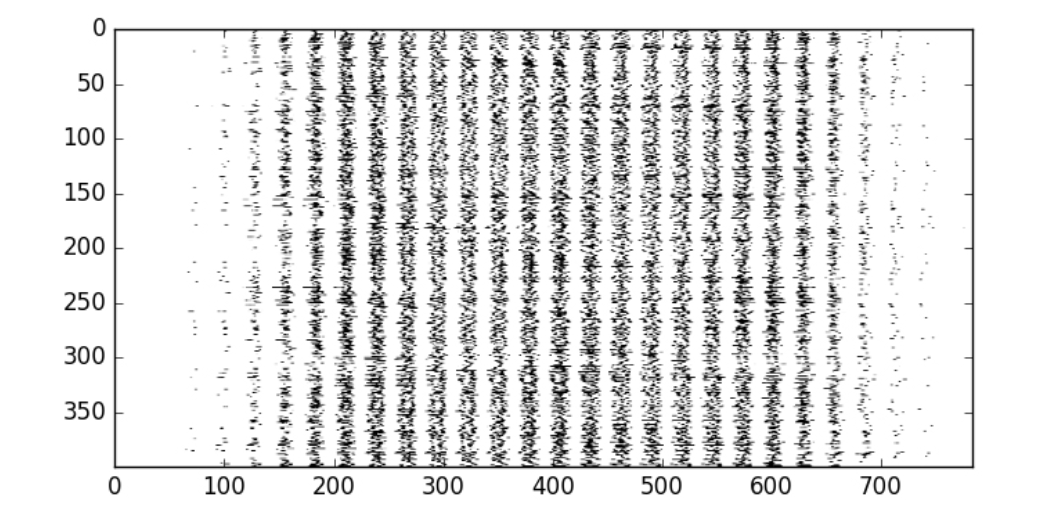


The one hot encoding was possible via the following function:





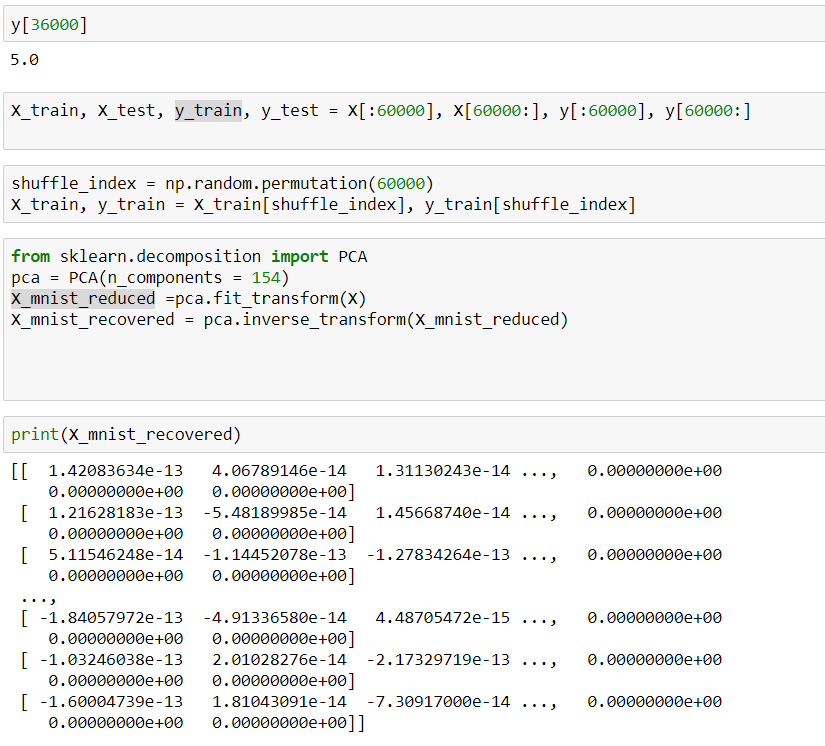
Through the flattened form we can observe the pixel intensity; like the image below

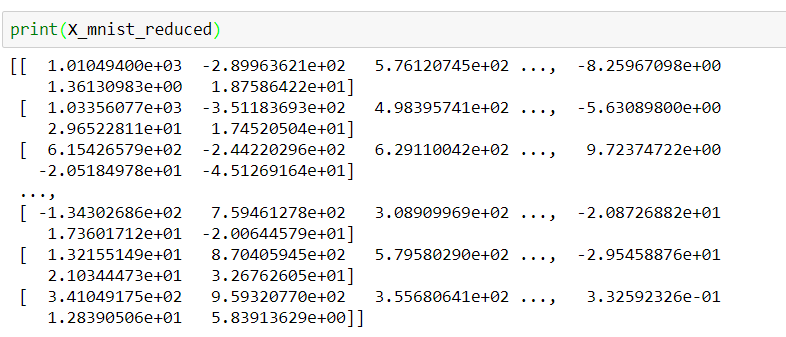


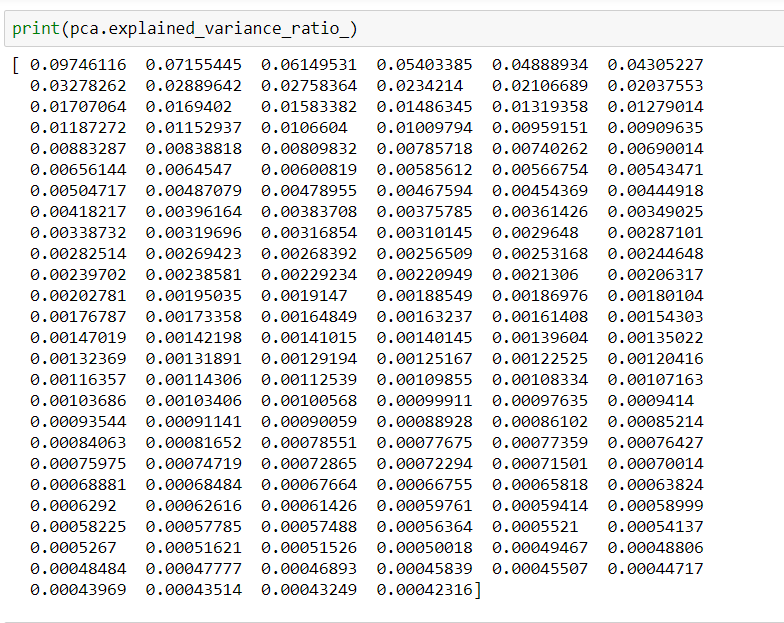
**Entire Code**:

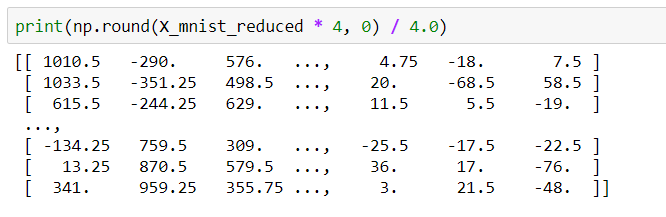


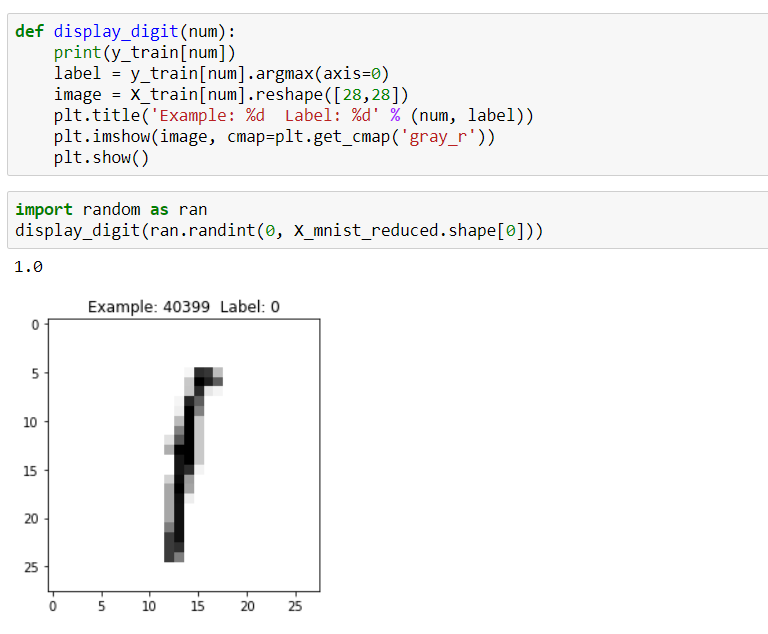


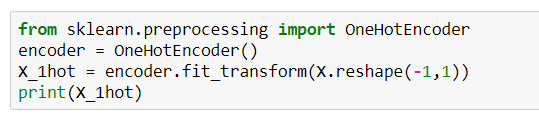


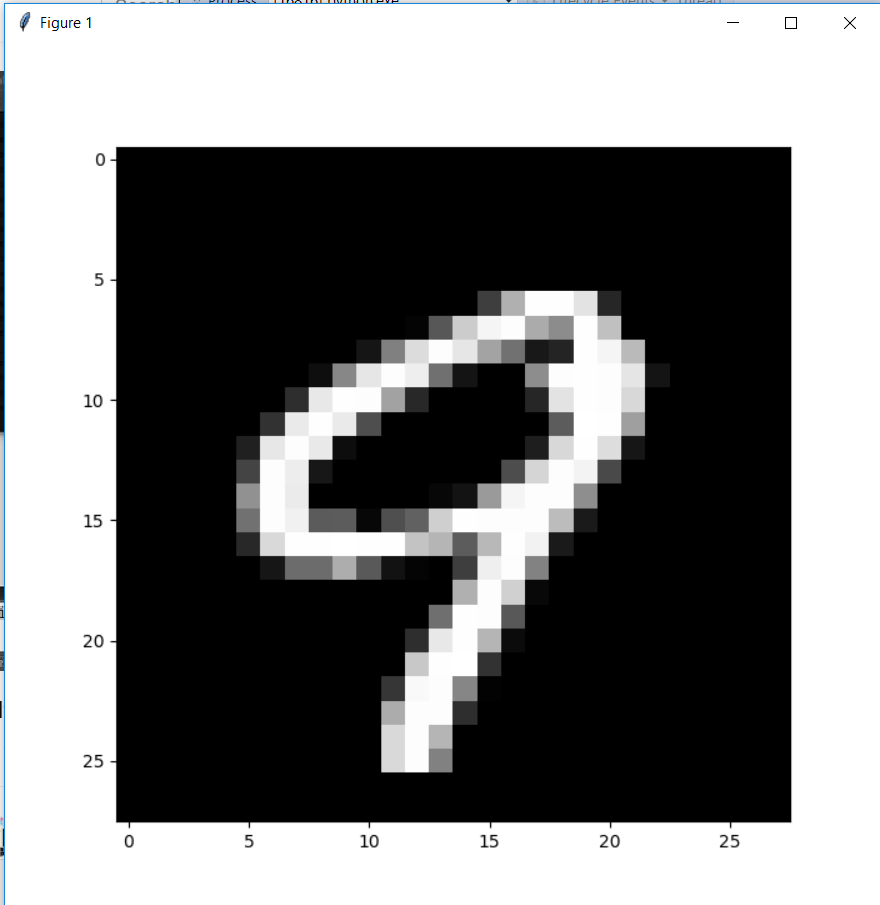












This snippet of code will help us get the intensity of each pixel.

Ran the code using DecisonTreeClassifier

