```
# Code Optimized Version:
# Optimized Version of MNIST_Special
# Author: Aman Jain
# Date: 2/2/2018

# Loading MNIST dataset:
mnist_data = fetch_mldata('MNIST original')

# Print to show keys available in dataset
print("Keys: {}".format(mnist_data.keys()))

# Number of datapoints and features
n_samples, n_features = mnist_data.data.shape
print('Number of datapoints: {}\nNumber of features: {}'.format(n_samples, n_features))

# Number of classes
n_classes = len(np.unique(mnist_data.target))
print('Number of classes: {}'.format(n_classes))
```

```
Keys: dict_keys(['DESCR', 'COL_NAMES', 'data', 'target'])
Number of datapoints: 70000
Number of features: 784
Number of classes: 10
```

Loaded MNIST data and Checked Description.

```
# Look at data
print('data: \n', mnist_data['data'])
print('shape: ', mnist_data['data'].shape)
print('type: ', type(mnist_data['data']))
print('target: \n', mnist_data['target'])
print('shape: ', mnist_data['target'].shape)
print('type: ', type(mnist_data['target']))
```

```
data:
 [[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]]
shape:  (70000, 784)
type:  <class 'numpy.ndarray'>
target:
 [ 0.  0.  0. ...  9.  9.  9.]
shape:  (70000,)
type:  <class 'numpy.ndarray'>
```

```
# Make data a Pandas df
mnist_df = pd.DataFrame(mnist_data['data'])
mnist_df['target'] = mnist_df['target'].astype('category')
mnist_df.head(3)
```

Using pandas to be able to efficiently explore dataset and to convert it easily into "tags" or c.o.

```
# Import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.pyplot as plt
# Choose a random digit to display
random_digit = mnist_df.iloc[np.random.randint(0, mnist_df.shape[0])].drop('target')
random_digit_image = np.reshape(random_digit, (28, 28))
plt.imshow(random_digit_image, cmap = mpl.cm.binary,
           interpolation="nearest")
```

Seperated Traget and Independent variables , Converted "Targets" into Binary Variable.

```
# Seperate target and features
X = mnist_df.drop('target')
y = mnist_df['target']

# Change target values
y = np.array([int(x) for x in y.values.ravel() if x == 5])
```

Splited Train and Test data, with 10000 observation into test data.

```
# Use train_test_split to split training and test data.
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=10000)
```

Standardized data with StandardScaler Option.

```
# Use StandardScaler to standardize the data.
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train, y_train)
X_test = scaler.transform(X_test)
```

Taken Overall PCA of 784 features for Check and Eigen values listed below.

```
# Use PCA to decompose the data.
from sklearn.decomposition import PCA
pca = PCA(n_components=784)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
# Import numpy as np
import numpy as np
explained_variance = np.cumsum(pca.explained_variance_ratio_)
print('Cumulative explained variance: \n', explained_variance)
```

```
Cumulative explained variance:
 [ 0.09746116  0.17020038  0.23561972  0.28626098  0.32744276  0.36351326
  0.39499254  0.42603538  0.45434951  0.47943964  0.50298649  0.52497186
  0.54507094  0.56405831  0.58187153  0.59854891  0.61439276  0.62929437
  0.64334767  0.65660888  0.66906488  0.68096023  0.69227981  0.70298693
  0.71319301  0.72298347  0.73230176  0.74117592  0.74964621  0.75777967
  0.76555816  0.77298898  0.78015791  0.78705762  0.79370153  0.79999325
  0.80600167  0.81179328  0.81734165  0.82271176  0.82788104  0.83283176
  0.83765376  0.84230168  0.84680188  0.85117371  0.85540888  0.85952325
  0.86353239  0.86741315  0.87117148  0.87480978  0.87833175  0.88174192
  0.88505149  0.88825146  0.89134188  0.89432275  0.89719406  0.89995582
  0.90261802  0.90518066  0.90764375  0.91000728  0.91227126  0.91443568
  0.91650055  0.91846586  0.92033161  0.92209781  0.92376445  0.92533154
  0.92679908  0.92816706  0.92943549  0.93060436  0.93167368  0.93264344
  0.93351365  0.93428429  0.93495538  0.93552691  0.93599888  0.93637129
  0.93664414  0.93681743  0.93689116  0.93686533  0.93673994  0.93651499
  0.93619048  0.93576642  0.93524279  0.93461961  0.93389687  0.93307457
  0.93215271  0.93113129  0.93000031  0.92876977  0.92743966  0.92600999
  0.92448076  0.92285196  0.92112361  0.91929569  0.91736821  0.91534117
  0.91321456  0.91098839  0.90866266  0.90623736  0.90371251  0.90108809
  0.89836411  0.89554057  0.89261747  0.88959481  0.88647259  0.88325081
  0.87992947  0.87650857  0.87298811  0.86936809  0.86564851  0.86182937
  0.85791067  0.85389241  0.84977459  0.84555721  0.84124027  0.83682377
  0.83230771  0.82769209  0.82297691  0.81816217  0.81324787  0.80823401
  0.80312059  0.79790761  0.79259507  0.78718297  0.78167131  0.77606009
  0.77034931  0.76453897  0.75862907  0.75261961  0.74651059  0.74030201
  0.73399387  0.72758617  0.72107891  0.71447209  0.70776571  0.70095977
  0.69405427  0.68704921  0.67994459  0.67274041  0.66543667  0.65803337
  0.65053051  0.64292809  0.63522611  0.62742457  0.61952347  0.61152281
  0.60342259  0.59522281  0.58692347  0.57852457  0.57002611  0.56142809
  0.55273051  0.54393337  0.53503667  0.52604041  0.51694459  0.50774921
  0.49845427  0.48905977  0.47956571  0.46997209  0.46027891  0.45048617
  0.44059387  0.43060201  0.42051059  0.41031961  0.40002907  0.38963897
  0.37914931  0.36856009  0.35787131  0.34708297  0.33619507  0.32520761
  0.31412059  0.30293401  0.29164787  0.28026217  0.26877691  0.25719209
  0.24550771  0.23372377  0.22184027  0.20985721  0.19777459  0.18559241
  0.17331067  0.16092937  0.14844851  0.13586809  0.12318811  0.11040857
  0.09752947  0.08455081  0.07147259  0.05829481  0.04501747  0.03164057
  0.01816411  0.00458809  0.00000000  0.00000000  0.00000000  0.00000000
  0.00000000  0.00000000
```