DIMENSIONALITY REDUCTION

Dimensionality Reduction is the process to lessen the number of dimensions a dataset has and yet to visualize and preserve the information insights present in them.

In Data Science we get datasets with a lot a higher dimensions and it would be a lot easier for computer efficiency if we could reduce these dimensions to a certain level and maintain data insights at the same time.

There are certain techniques for imposing Dimensionality reduction on the datasets. Two of them which are widely used in the industry are :

1.) PCA (Principal Compent Analysis)

2.) tSNE (T-distributed Stochastic Neighbourhood Embeding)

The one which we would discuss here would PCA using the example of MNIST dataset.

MNIST Dataset : It is a simple computer vision dataset. It consists of 28 X 28 pixel images of handwritten digits Every MNIST datapoint i.e, an image can be thought of as an array of numbers decribind how dark each pixel is. Through these 28 X 28 Pixels weget a 28 X 28 array and after applying Row Flattening we get a 28 * 28 = 784 dimensional Column vector

In [8]:

```
import pandas as pd
import numpy as np

d = pd.read_csv('./Data/MNISTtrain.csv')
```

In [12]:

```
print(d.head(5))
d.columns
```

```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0      1       0       0       0       0       0       0       0       0
1      0       0       0       0       0       0       0       0       0
2      1       0       0       0       0       0       0       0       0
3      4       0       0       0       0       0       0       0       0
4      0       0       0       0       0       0       0       0       0

   pixel8  ...  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  ...         0         0         0         0         0         0
1       0  ...         0         0         0         0         0         0
2       0  ...         0         0         0         0         0         0
3       0  ...         0         0         0         0         0         0
4       0  ...         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
4         0         0         0         0

[5 rows x 785 columns]
```

Out[12]:

```
Index(['label', 'pixel0', 'pixel1', 'pixel2', 'pixel3', 'pixel4', 'pixel5',
       'pixel6', 'pixel7', 'pixel8',
       ...
       'pixel774', 'pixel775', 'pixel776', 'pixel777', 'pixel778', 'pixel779',
       'pixel780', 'pixel781', 'pixel782', 'pixel783'],
      dtype='object', length=785)
```

In [14]:

```
l = d['label']
```

```
data = d.drop('label' , axis =1)
```

In [16]:

```
print (l.shape)
print (data.shape)
```

```
(42000,)
(42000, 784)
```

In [17]:

```
import matplotlib.pyplot as plt
```

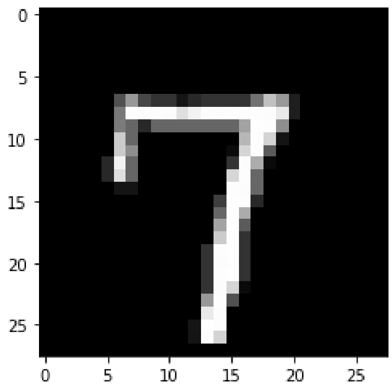
Visualization of MNIST Dataset

In [28]:

```
idx = 6
griddata = data.iloc[idx].as_matrix().reshape(28,28)
plt.imshow(griddata , cmap = "gray")
plt.show()

print(l[idx])
```

```
C:\Users\Nrohlable\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Method .as_
matrix will be removed in a future version. Use .values instead.
```




```
7
```

2D Visualization using PCA

Visulaization manually by computing Eigen Values and Eigen Vectors

In [96]:

```
from sklearn.preprocessing import StandardScaler
standardized_data = StandardScaler().fit_transform(data)
print (standardized_data.shape)
print(standardized_data.size)
```

```
(42000, 784)
32928000
```

In [97]:

```
#Finding Covarience Matrix of Standardized Data

sample_data = standardized_data
```

```
#From here we'll manually code the PCA rather than using the Scikit Learn Library

covarience = (np.matmul(sample_data.T , sample_data))/standardized_data.shape[0]
print ('Shape of Co-varience Matrix: ' ,covarience.shape)
```

```
Shape of Co-varience Matrix:  (784, 784)
```

In [98]:

```
# Finding the top two eigen-values and corresponding eigen-vectors
# For projecting onto a 2-Dim space.

from scipy.linalg import eigh

# Eigh function will return the eigen values in asending order
# The following code generates only the top 2 (782 and 783) eigenvalues.

values, vectors = eigh(covarience , eigvals = (782,783))

#Shape of Eigen Vector :

print ('Shape of Eigen Vector = ' , vectors.shape)
print(data.shape)

#Reducing the 784 dimensional data to 2D using 2 Eigen Vectors

new_coordinates = np.matmul(sample_data , vectors)
print(new_coordinates.shape)
print(new_coordinates)
```

```
Shape of Eigen Vector =  (784, 2)
(42000, 784)
(42000, 2)
[[-5.2264454  -5.14047772]
 [ 6.03299601 19.29233234]
 [-1.70581328 -7.64450341]
 ...
 [ 7.07627667  0.49539137]
 [-4.34451279  2.30724011]
 [ 1.55912058 -4.80767022]]
```

In [99]:

```
#Creating new DataFrame

dataframe1 = pd.DataFrame(data = new_coordinates , columns = ('2nd Component','1st Component'))
dataframe2 = pd.DataFrame(data = l)

print (dataframe1.head(5))
print(dataframe2.head(5))
```

```
   2nd Component  1st Component
0      -5.226445      -5.140478
1       6.032996      19.292332
2      -1.705813      -7.644503
3       5.836139      -0.474207
4       6.024818      26.559574
   label
0      1
1      0
2      1
3      4
4      0
```

In [100]:

```
dataframe3 = dataframe1.join(dataframe2)
print(dataframe3.head(5))
```

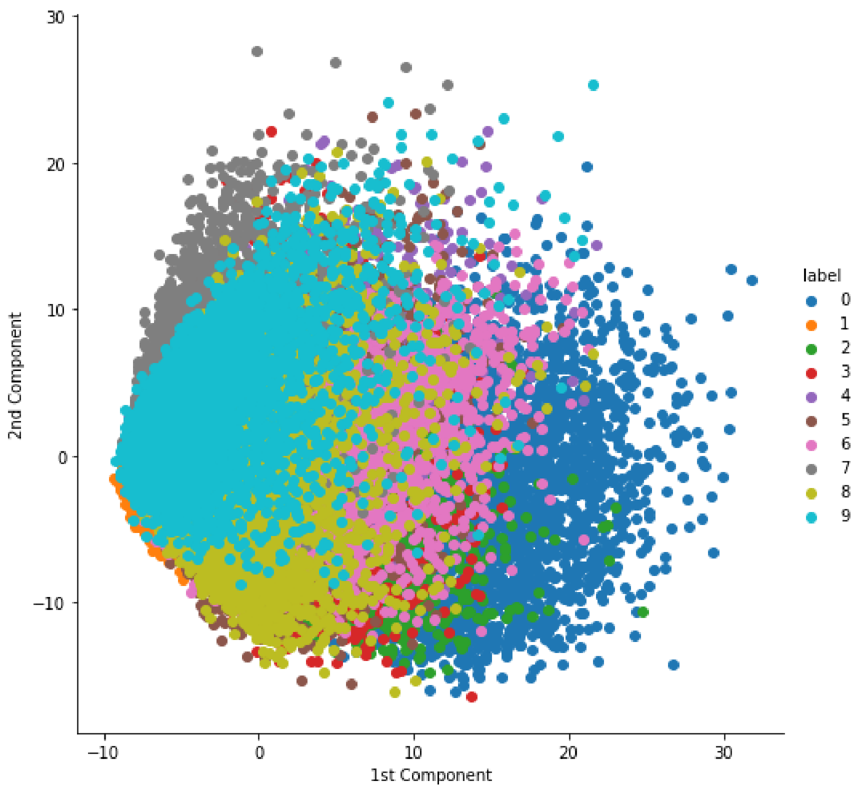
```
   2nd Component  1st Component  label
```

```
0       -5.226445       -5.140478       1
1        6.032996       19.292332       0
2       -1.705813       -7.644503       1
3        5.836139       -0.474207       4
4        6.024818       26.559574       0
```

In [101]:

```
import seaborn as sn

sn.FacetGrid(dataframe3, hue ='label' , height= 7 ).map(plt.scatter, '1st Component' , '2nd Compone
nt').add_legend()
plt.show()
```



PCA using Scikit- Learn

We could use Scikit Learn Liberaries function other than manually coding the entire thing as we did earlier above in this code

In [102]:

```
#initialization of PCA

from sklearn import decomposition
pca = decomposition.PCA()
```

In [128]:

```
pca.n_components = 2
pca_data = pca.fit_transform(sample_data)

print('Shape of the data given by PCA :', pca_data.shape)
```

```
Shape of the data given by PCA : (42000, 2)
```

There are two methods of creating the DataFrame for plotting the Visualization in 2 dimensional . One is using the Pandas operation as shown in the earlier part and the other one is the numpy version which is shown below here

In [130]:

```
pca_data = np.vstack((pca_data.T, l)).T
pca_data.shape
```

Out[130]:

```
(42000, 3)
```

In [131]:

```
print(pca_data)
```

```
[[-5.14048798 -5.22678137  1.        ]
 [19.29228697  6.032057    0.        ]
 [-7.64449087 -1.70543786  1.        ]
 ...
 [ 0.49535482  7.07520852  7.        ]
 [ 2.30729859 -4.34310371  6.        ]
 [-4.8076956   1.55885169  9.        ]]
```

In [133]:

```
#Putting this into a DataFrame

pca_df = pd.DataFrame ( data = pca_data , columns = ('1st_Component' , '2nd_Component' , 'Lable'))
pca_df.head(5)
```

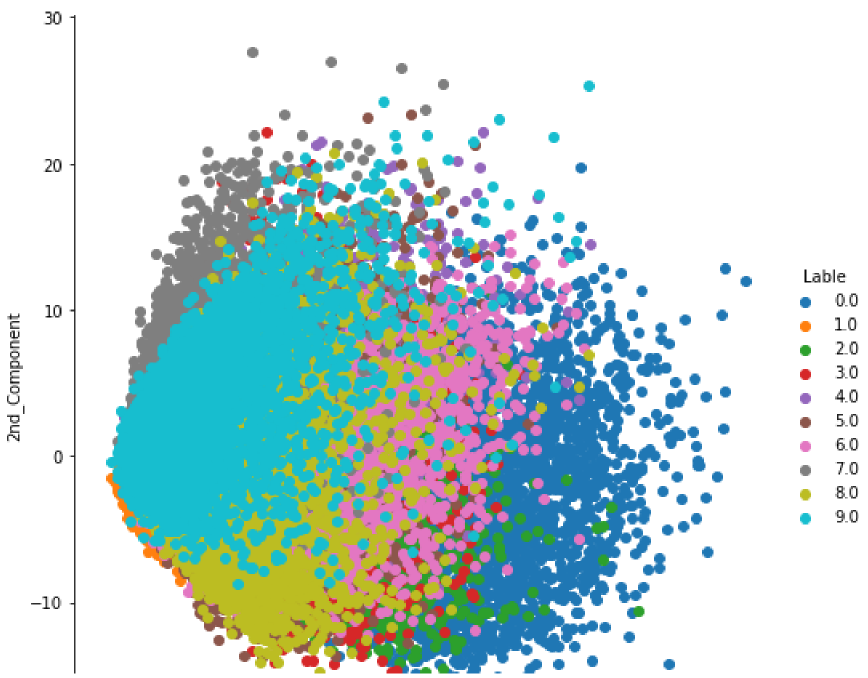
Out[133]:

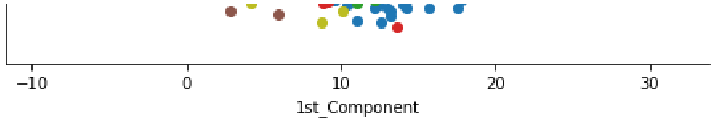
	1st_Component	2nd_Component	Lable
0	-5.140488	-5.226781	1.0
1	19.292287	6.032057	0.0
2	-7.644491	-1.705438	1.0
3	-0.474197	5.836435	4.0
4	26.559530	6.023571	0.0

In [136]:

```
import seaborn as sns

sns.FacetGrid(pca_df , hue = 'Lable' , height = 7).map(plt.scatter , '1st_Component' ,
'2nd_Component').add_legend()
plt.show()
```





The Visualization using PCA for this 784 dimensional data is not that appropiate as we could observe it from the figure above. The reason behind this poor visualization is the less preservation of data while reducing the dimensions from 784D to 2D PCA technique is used to reduce the number of dimensions to a point where one could retain about 90-95% of the Data.

This preservation of Data will be explained in codes below :

PCA for Dimensionality Reduction

In [137]:

```
pca.n_components = 784
pca_dat = pca.fit_transform(sample_data)
pca_dat.shape
```

Out[137]:

```
(42000, 784)
```

In [140]:

```
var = pca.explained_variance_
var.shape
```

Out[140]:

```
(784,)
```

In [143]:

```
var_sum = np.sum(var)
print(var_sum)
```

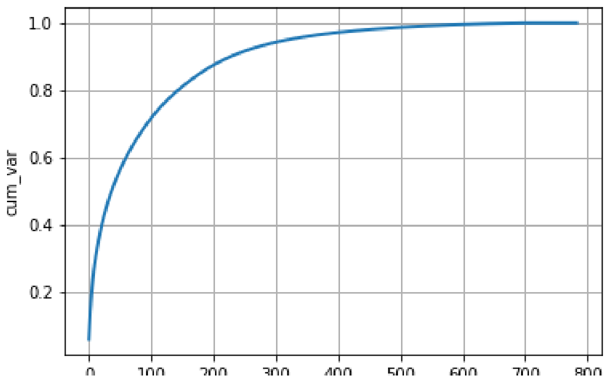
```
708.0168575442273
```

In [147]:

```
percentagevar = ((var)/(var_sum))
cum_var = np.cumsum(percentagevar)
```

In [153]:

```
plt.figure(figsize=(6,4))
plt.plot(cum_var , linewidth = 2)
plt.grid(linewidth = 1)
plt.xlabel('n_components')
plt.ylabel('cum_var')
plt.show()
```



Conclusion :

If we reduce the dimensions to around 350 dimensions which less than half of the data points we could preserver 95% of the data Apart from Visualization part we could build a ML Model to find out the results i.e, class labels which is in between 0 to 9 for a given input of Hand written image data of 28X28 pixel. PCA would decease the load for ML Model since it'll allow less than 50% of the data to be processed down as that would be enough as it would have 95% of the Data insights.

In [ ]: