1.b) θ =$(x^T x)^{-1} x^T y = \begin{pmatrix} 1/38 & -5/38 \\ -5/38 & 69/76 \end{pmatrix} \begin{pmatrix} 568 \\ 82 \end{pmatrix} = \begin{pmatrix} 4.15789474 \\ -0.289473684 \end{pmatrix}$

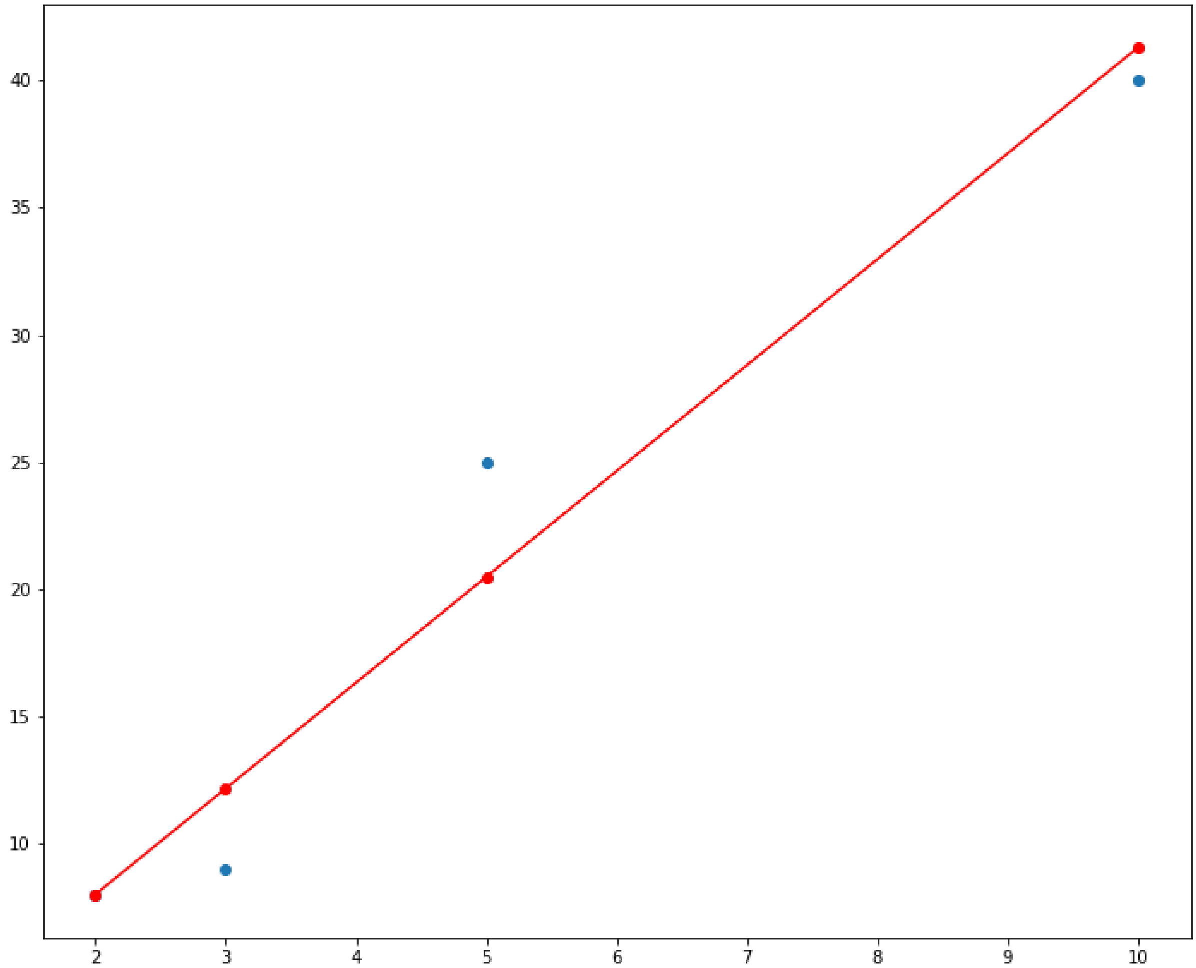
c) Gradient Descent is iterative in nature meaning it takes multiple steps to get global optimum but there is a case of linear regression where there is a way to get optimal values of parameters theta in single step. This is nothing but normal equation. Here, we get the best parameters with a formula that comprises of matrix multiplications and inversions.

d)

```
#Importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#Preprocessing the Data
x=[2,5,3,10]
y=[8,25,9,40]

#Model building
mean_x = np.mean(x)
mean_y = np.mean(y)

#total number of values
n = len(x)

#Calculating m and c
numer = 0
denom = 0

for i in range(n) :
  numer += (x[i] - mean_x) * (y[i] - mean_y)
  denom += (x[i] - mean_x) ** 2
m = numer/denom
c = mean_y - (m * mean_x)

#coefficients
print("coefficients:")
print(m,c)

#Prediction
y_pred = m * np.asarray(x) + c
plt.scatter(x,y)  #actual
plt.plot([min(x), max(x)], [min(y_pred), max(y_pred)], color ='red')
plt.scatter(x, y_pred, color = 'red')
plt.show()

#Calculating RMSE

rmse = 0
```

```
for i in range(n):
  y_pred = c + m * x[i]
  rmse += (y[i] - y_pred) ** 2
rmse = np.sqrt(rmse/n)
print("Root Mean Squares Error:")
print(rmse)

#Calculate R2 Score
ss_tot = 0
ss_res = 0
for i in range(n) :
  y_pred = c + m * x[i]
  ss_tot += (y[i] - mean_y) ** 2
  ss_res += (y[i] - y_pred) ** 2
r2 = 1 - (ss_res/ ss_tot)
print("R2 Score:")
print(r2)

#Closed form Normal Equation

a = np.asarray(x). shape[0]
x = np.append(np.asarray(x).reshape(-1,1),np.ones((a,1)),axis = 1)
x = np.array(x, dtype = 'int16')
y = np.array(y).reshape(-1,1)
theta = np.dot(np.linalg.inv(np.dot(x.T, x)), np.dot(x.T, y))
print("Closed form normal equation:", theta)
```

```
coefficients:
4.157894736842105 -0.28947368421052744
```



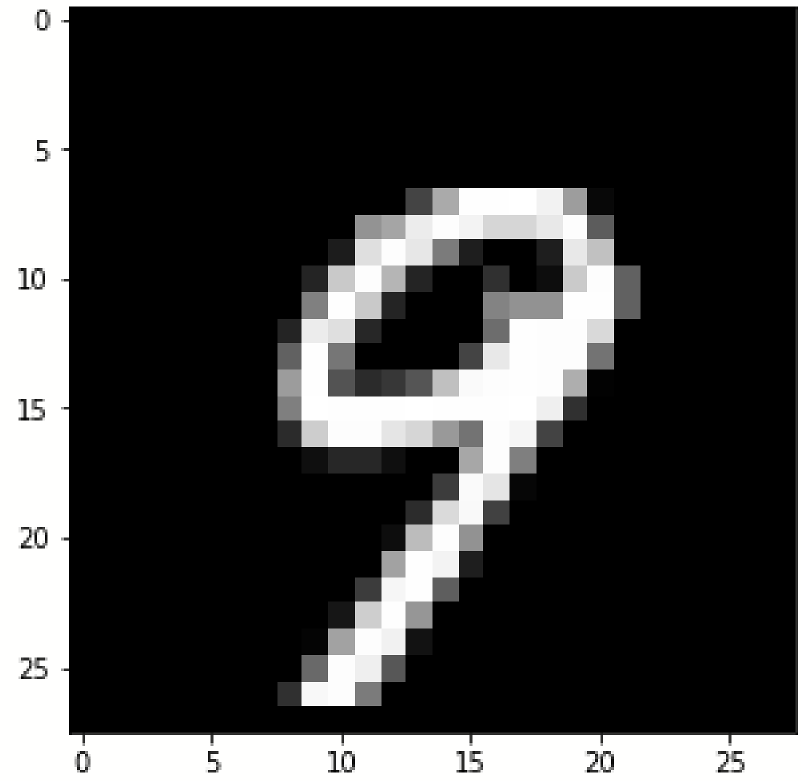

```
Root Mean Squares Error:
2.8307521782623146
R2 Score
0.9534794897257658
Closed form normal equation solution: [[ 4.15789474]
 [-0.28947368]]
```

2)

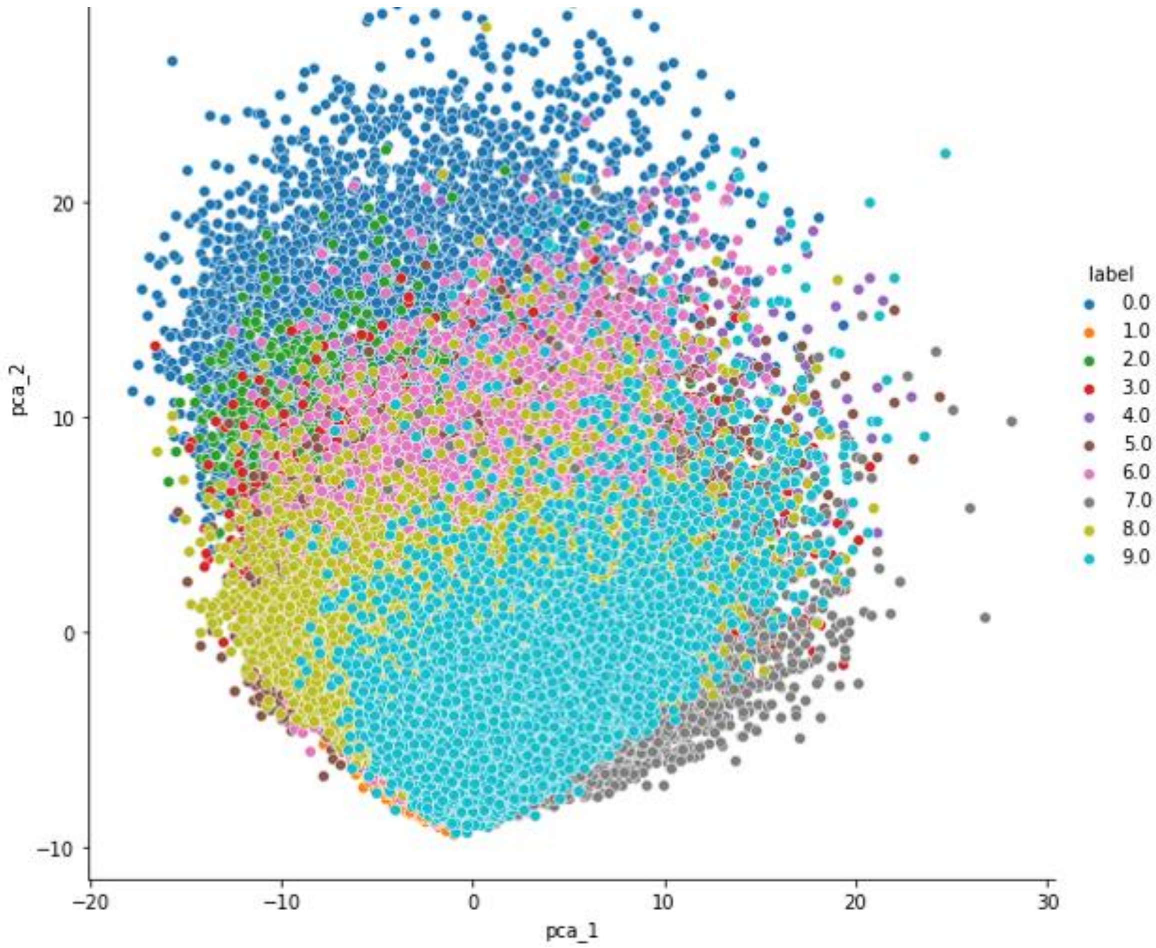
```
#Importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
from mnist import MNIST
from sklearn.preprocessing import StandardScaler
from scipy.linalg import eigh

#Loading MNIST Dataset

mndata = MNIST('C:\\Users\\anujeeth\\OneDrive\\Documents\\mnist')
training_images, training_labels = mndata.load_training()
```

```
test_images, test_labels = mndata.load_testing()
print("shape of training_images data is : ", np.array(training_images).
shape)
print("shape of training_labels data is : ", np.array(training_labels).
shape)
print("shape of test_images data is : ", np.array(test_images).shape)
print("shape of test_labels data is : ", np.array(test_labels).shape)

#Extracting column labels from training set
label = training_labels
ind = np.random.randint(0,20000)
plt.figure(figsize = (20,5))
grid_data = np.array(pd.DataFrame(training_images).iloc[ind]).reshape(2
8,28)

#Plotting
plt.imshow(grid_data, interpolation = None, cmap = 'gray')
plt.show()

#Column standardization using standardScalar class. after column standa
rdization, the mean of every attribute becomes 0 and variance 1.

scalar = StandardScaler()
std_df = scalar.fit_transform(training_images)
print("Shape of the dataset after the column standardization:", std_df.
shape)

#Finding the co-variance matrix
covar_mat = np.matmul(std_df.T, std_df)
print("the dimensions of co-
variance matrix after multiplication", covar_mat.shape)

#Finding the top two eigen-
values and corresponding eigen vectors. It generates only top 2 (782 an
d 783) eigen values. Converting eigen vectors into (2,d) form

values, vectors = eigh(covar_mat, eigvals = (782,783))
print("Dimensions of eigen vector:", vectors.shape)
vectors= vectors.T
print("Dimensions of eigen vector:", vectors.shape)

#Finding PC1 and PC2

final_df = np.matmul(vectors, stu_df.T)
print("vectors:", vectors.shape, "n", "std_df:", std_df.T.shape,"n", "f
inal_df:", final_df.shape)

#Transposing final_df
```

```python
#Now converted 60000 * 784 data to 60000*4

final_dfT = np.vstack(final_df, label).T
dataFrame = pd.DataFrame(final_dfT, columns = ['pca_1', 'pca_2', 'label
'])
print(dataFrame)

#Visualizing final data using seaborn Facet Grid

sns.FacetGrid(dataFrame, hue = 'label', size = 8)\
.map(sns.scatterplot, 'pca_1','pca_2')\
.add_legend()
plt.show()
```



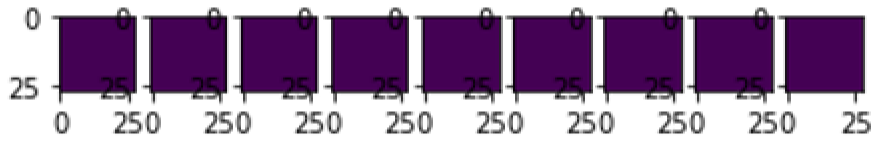

```
          pca_1     pca_2  label
0     -4.814790 -0.922159    5.0
1     -7.754403  8.708977    0.0
2      9.431338  2.328389    4.0
3     -3.746318 -6.582173    1.0
4      3.133297 -5.183251    9.0
...         ...       ...    ...
59995 -5.119129 -2.039339    8.0
59996 -6.498440  0.607841    3.0
59997 -3.230564 -3.777212    5.0
59998 -4.948125  1.722369    6.0
59999 -6.175386 -1.427251    8.0
```

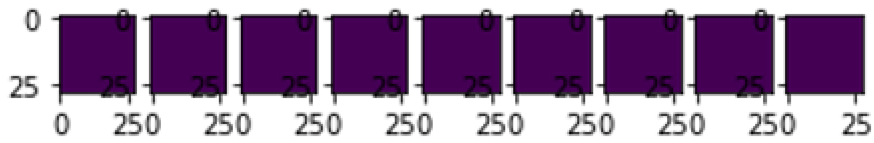


2.a)

```
#Importing libraries
from mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt

#Loading MNIST dataset
mnist = MNIST('C:\\Users\\anujeeth\\OneDrive\\Documents\\mnist')
X_train,y_train = mnist.load_training()

M = np.zeros(28,28,10)
S = np.zeros(28,28,10)

for i in range(9) :
    X_subset = X_train[y_train == i]
    M[:,:,i+1] = np.mean(X_subset, axis = 0)
    S[:,:, i+1] = np.mean(X_subset, axis = 0)
    plt.subplot(2,10,i+1)
    plt.imshow(M[:,:,i+1])
    plt.subplot(2,10,i+11)
    plt.imshow(S[:,:,i+1])
```

```
#Visualizing
plt.show()
```





3)

#Importing libraries

```
from mnist import MNIST
import numpy as np
import matplotlib.pyplot as plt

#defining function
def pcs_svd(im_train, i):
    PC = np.dot(im_train, im_train.T)
    reconn = np.dot(PC, im_train)
    normal = np.dot(reconn, PC)
    dif = np.sum(normal=im_train)
    dif = dif/(60000*784)
    diff[i] = dif
    if(_name__ == '__main__'):
        im_train = np.loadtxt('C:\\Users\\anujeeth\\OneDrive\\Desktop\\
mnist\\t10k-images-idx3-ubyte')
        diff = np.zeros(784)
    for i in range(784)
    pcs_svd(i_train, i)
    plt.plot(diff)
    plt.show #Plotting
```