```
# # ML HW2
import csv
import numpy as np
import numpy.linalg as lin
import matplotlib.pyplot as plt

#Open file and set matrix
tmpfile = open("MNIST_training.csv", "rt", encoding='utf-8')
trn = np.array(list(csv.reader(tmpfile)))
trnY = np.transpose(np.matrix(np.array(trn[1:, 0], dtype=int)))
trnX = np.array(trn[1:, :], dtype=int)
trnX[0:, 0] = 1 #set 0-th column value 1 for intercept

tmpfile = open("MNIST_test.csv", "rt", encoding='utf-8')
tst = np.array(list(csv.reader(tmpfile)))
tstY = np.transpose(np.matrix(np.array(tst[1:, 0], dtype=int)))
tstX = np.array(tst[1:, :], dtype=int)
tstX[0:, 0] = 1 #set 0-th column value 1 for intercept
```

```
# # Task 1 : Linear Regression with OLS

# b_opt = (X'X)^(-1)X'y
# Get optimal coefficient
def getOptCoef(X,y) :
    Xt = np.transpose(X)
    optCoef = np.array(np.dot(np.dot(lin.pinv(np.dot(Xt, X)),Xt),y), dtype=float)
    return optCoef


# Display the optimal coefficients (denoted by b_opt)
b_opt = getOptCoef(trnX,trnY)
print(b_opt)


# Classify test data (MNIST_test.csv) with a threshold of 0.5.
#- y_pred = X_test * b_opt
#- if y_pred > 0.5, class 1, otherwise 0
def binaryClassification(X, coef, threshold):
    y_pred = np.dot(X, coef)
    y_pred[y_pred > threshold] = 1
    y_pred[y_pred <= threshold] = 0
    return y_pred

y_pred = binaryClassification(tstX, b_opt, 0.5)



# Display the accuracy
Accuracy = (np.sum(y_pred == tstY))/len(tstY)
Accuracy
```

```
# # Task 2. Implementation of Gradient Descent

# 1. Set the initial coefficient to zeros
b_est = np.zeros((len(trnX[0]), 1), dtype=float)


# 2. Determine hyper-parameters such as learning rate and iteration numbers
iteration = 50
learningRate = 0.000000003
sqError = np.zeros((iteration, 1), dtype=float)


# 3. Run "gradient descent" algorithm with the hyper-parameters
# Get estimated coefficient by Gradient Descend
def getGDcoef(coef, X, y, sqError,iteration, learningRate):
    Xt = np.transpose(X)
    for i in range(iteration):
        coef = coef - learningRate*(np.dot(np.dot(Xt,X), coef) - np.dot(Xt, y))
        sqError[i] = np.dot((np.transpose(y-np.dot(X, coef))), (y-np.dot(X, coef)))
    return coef

b_est = getGDcoef(b_est, trnX, trnY, sqError,iteration, learningRate)



# Check  "Learning Curve"
print(sqError)
plt.plot(sqError, 'b-')
plt.show()


# 4. Display the optimal coefficients (denoted by b_est)
print(b_est)
```

```python
# 5. Classify test data (MNIST_test.csv) with a threshold of 0.5.
y_pred = binaryClassification(tstX, b_est, 0.5)


# 6. Display the accuracy
Accuracy = (np.sum(y_pred == tstY))/len(tstY)
print(Accuracy)


# 7. Display the total differences between b_opt and b_est
print(np.sum(np.abs(b_opt-b_est)))
```