系級:資科所碩一

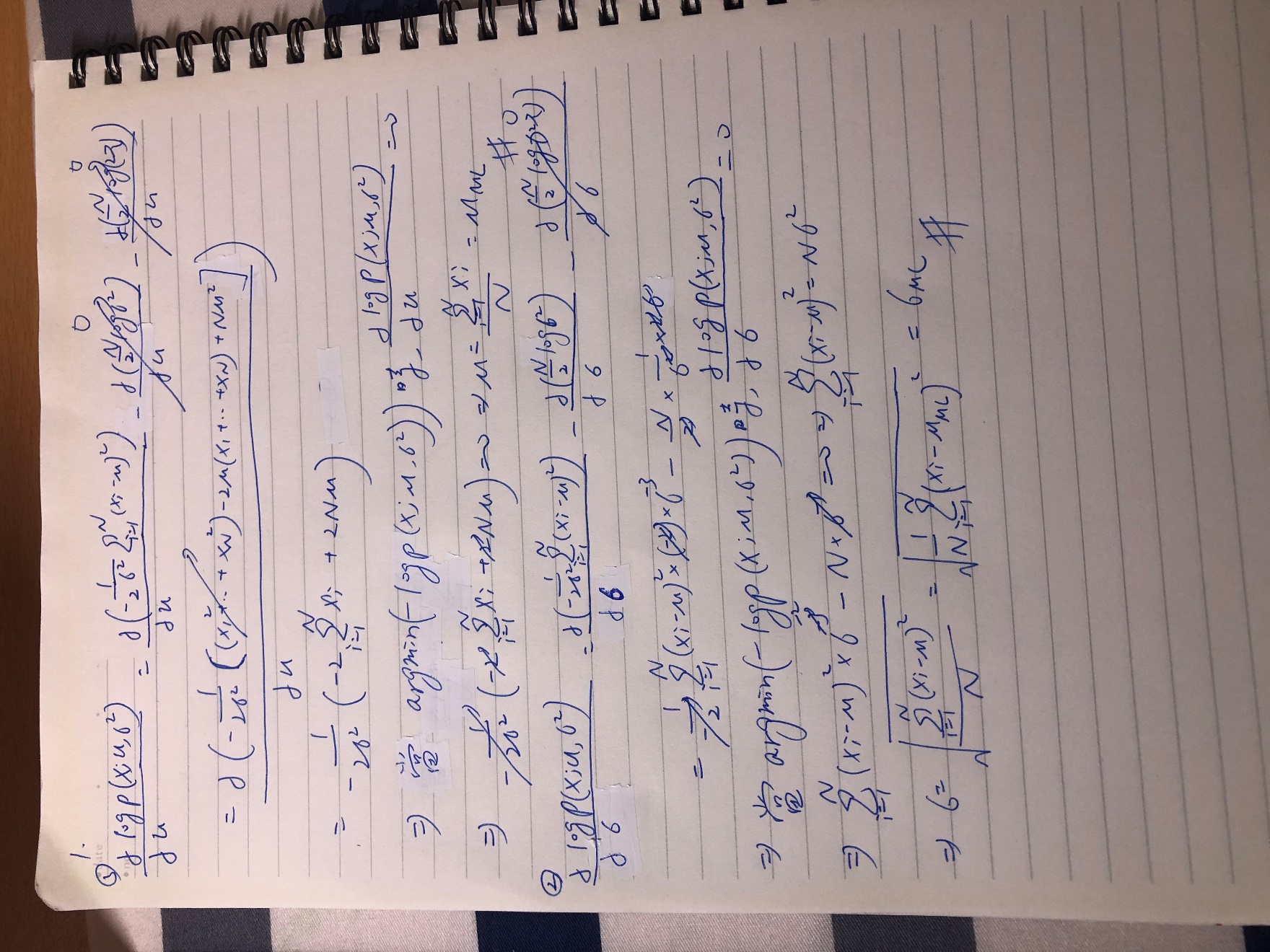
學號:108753208

姓名:葉冠宏

1072 Deep Learning – Homework 1

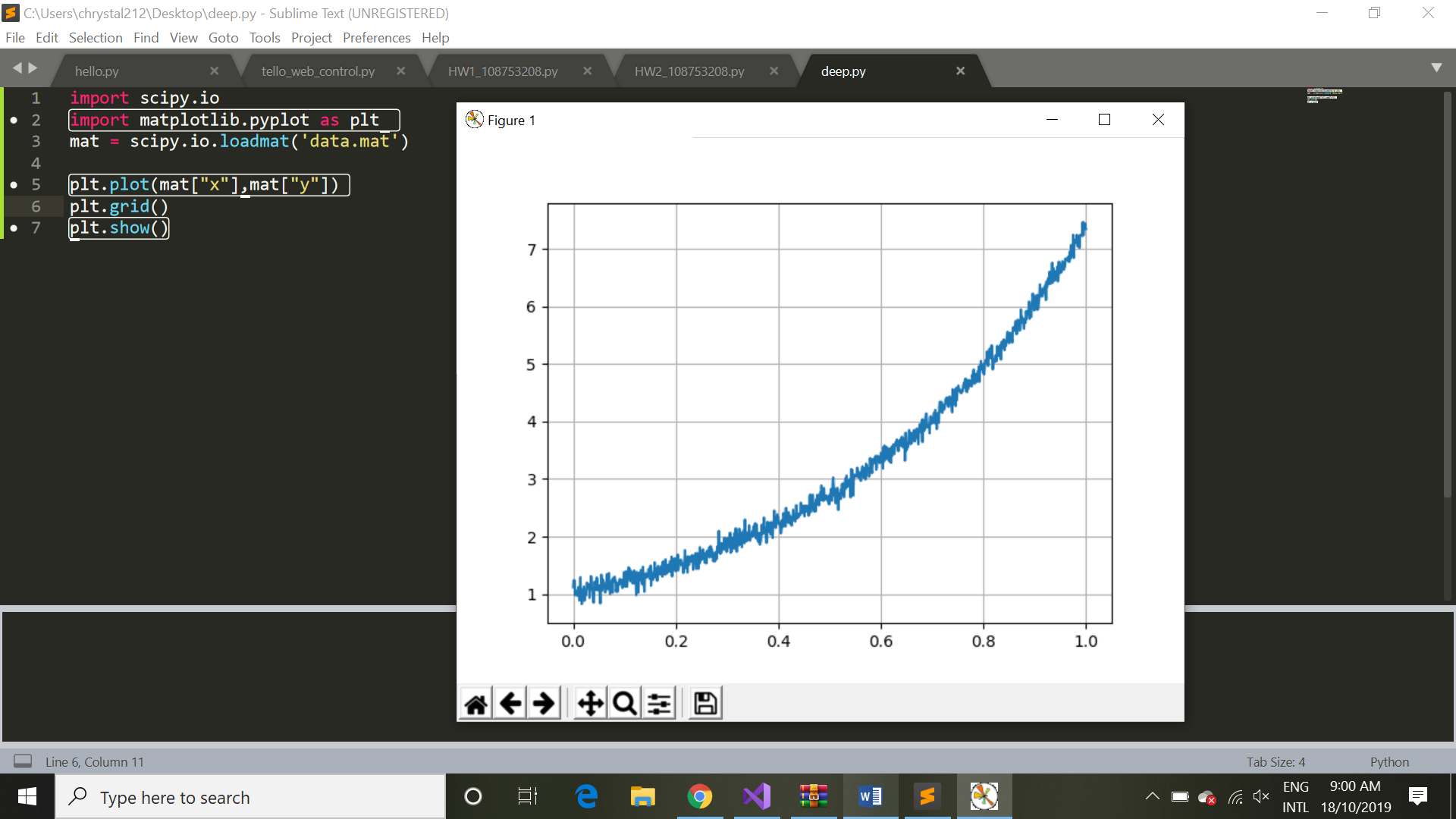
Due: October 24, 2019, 11:55pm

1. (20%) (Maximum Likelihood Estimation) Please set the derivatives of the log likelihood function of a Gaussian Probability Distribution Function to zero with respect to and and verify the following results below:



1. (20%) Please load ‘data.mat’ into your Matlab or Python code, where you will find . Now do the following procedures, **paste your source code and show the results in your report.(use gradient descent and pseudo inverse to get a bonus)**
   1. Plot the data using plot function.

>> plot(x, y); grid



import scipy.io as sio

import matplotlib.pyplot as plt

mat = sio.loadmat('data.mat')

plt.plot(mat[“x”],mat[“y”])

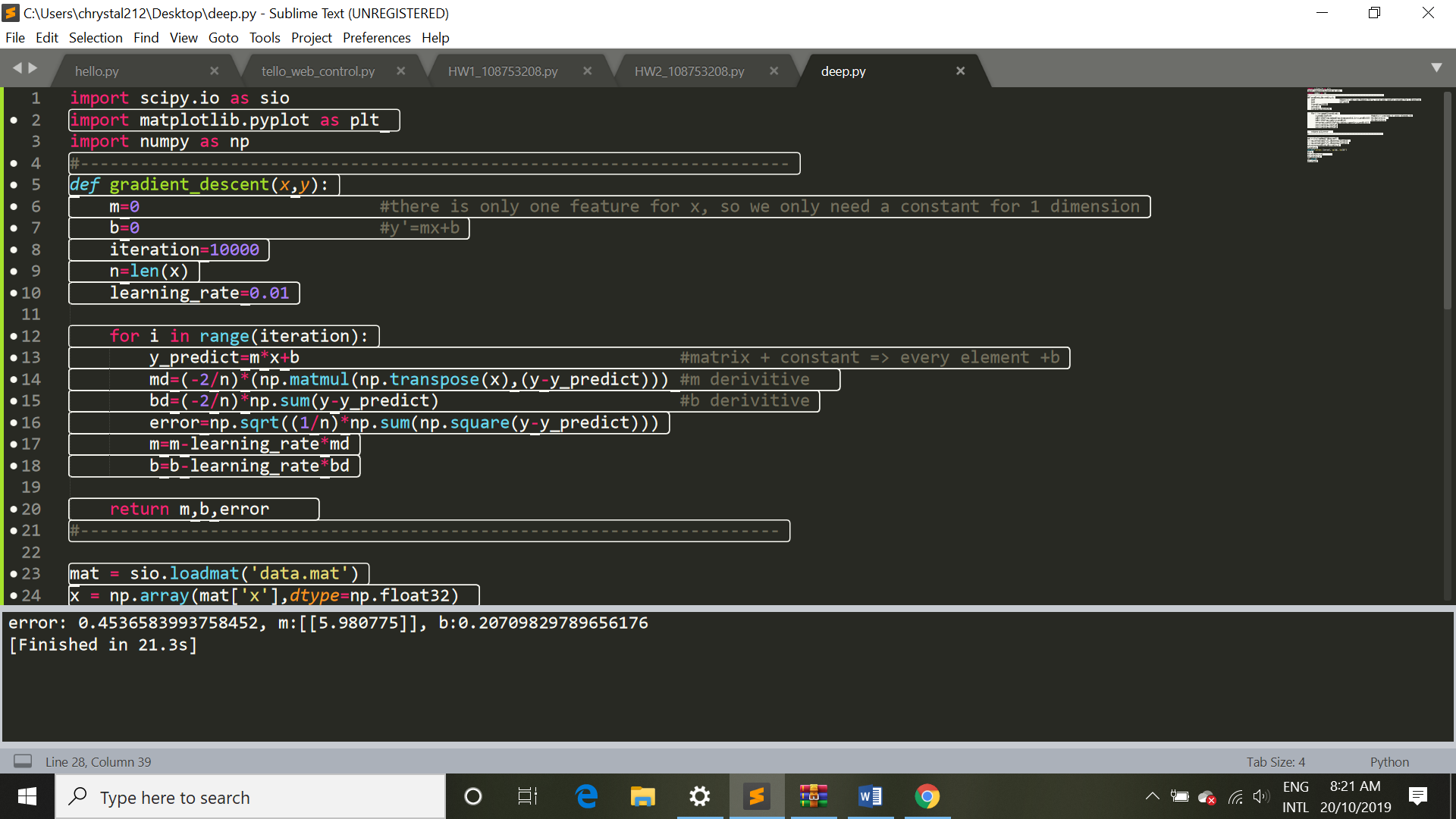
plt.grid()

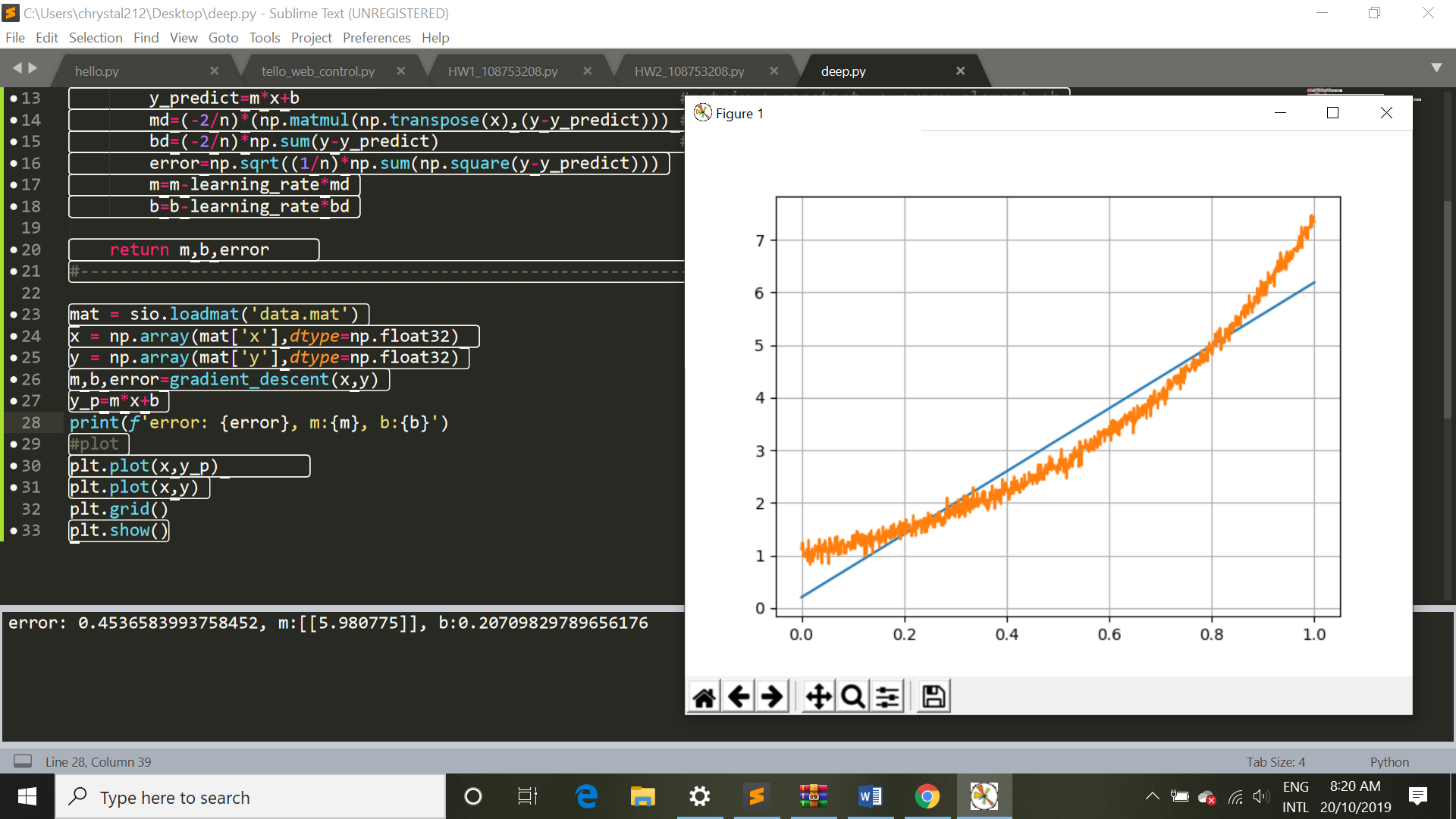
plt.show()

* 1. Compute the least square line using the given data and overlay the line over the given data.

>> hold on; plot(x, +\*x, ‘--‘)

我用gradient descent算出迴歸線，得y=5.98x+0.2





import scipy.io as sio

import matplotlib.pyplot as plt

import numpy as np

#-----------------------------------------------------------------------

def gradient\_descent(x,y):

m=0

b=0 #y'=mx+b

iteration=10000

n=len(x)

learning\_rate=0.01

for i in range(iteration):

y\_predict=m\*x+b

md=(-2/n)\*(np.matmul(np.transpose(x),(y-y\_predict))) #m derivitive

bd=(-2/n)\*np.sum(y-y\_predict) #b derivitive

error=np.sqrt((1/n)\*np.sum(np.square(y-y\_predict)))

m=m-learning\_rate\*md

b=b-learning\_rate\*bd

return m,b,error

#----------------------------------------------------------------------

mat = sio.loadmat('data.mat')

x = np.array(mat['x'],dtype=np.float32)

y = np.array(mat['y'],dtype=np.float32)

m,b,error=gradient\_descent(x,y)

y\_p=m\*x+b

#plot

print(f'error: {error}, m:{m}, b:{b}')

plt.plot(x,y\_p)

plt.plot(x,y)

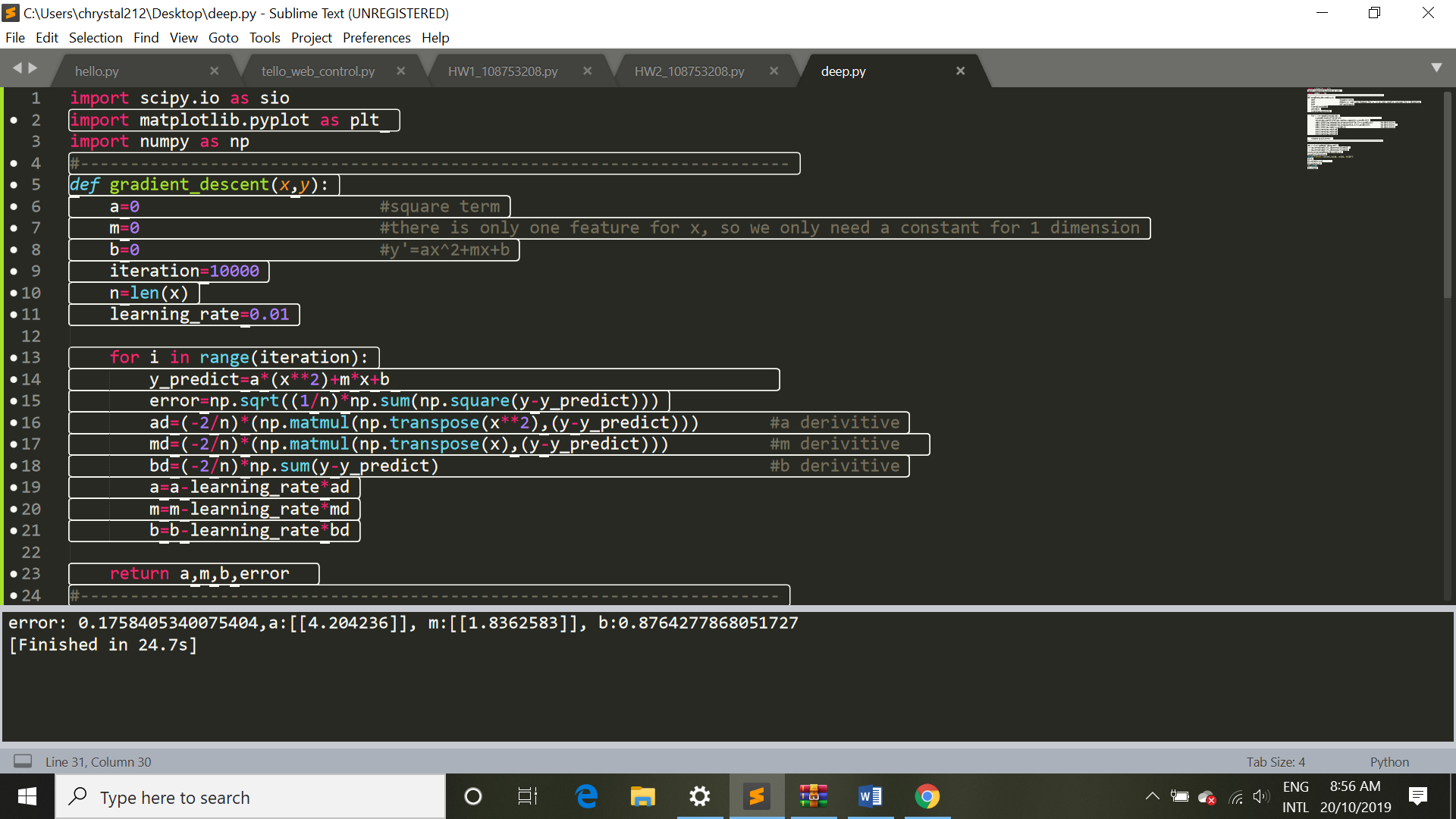
plt.grid()

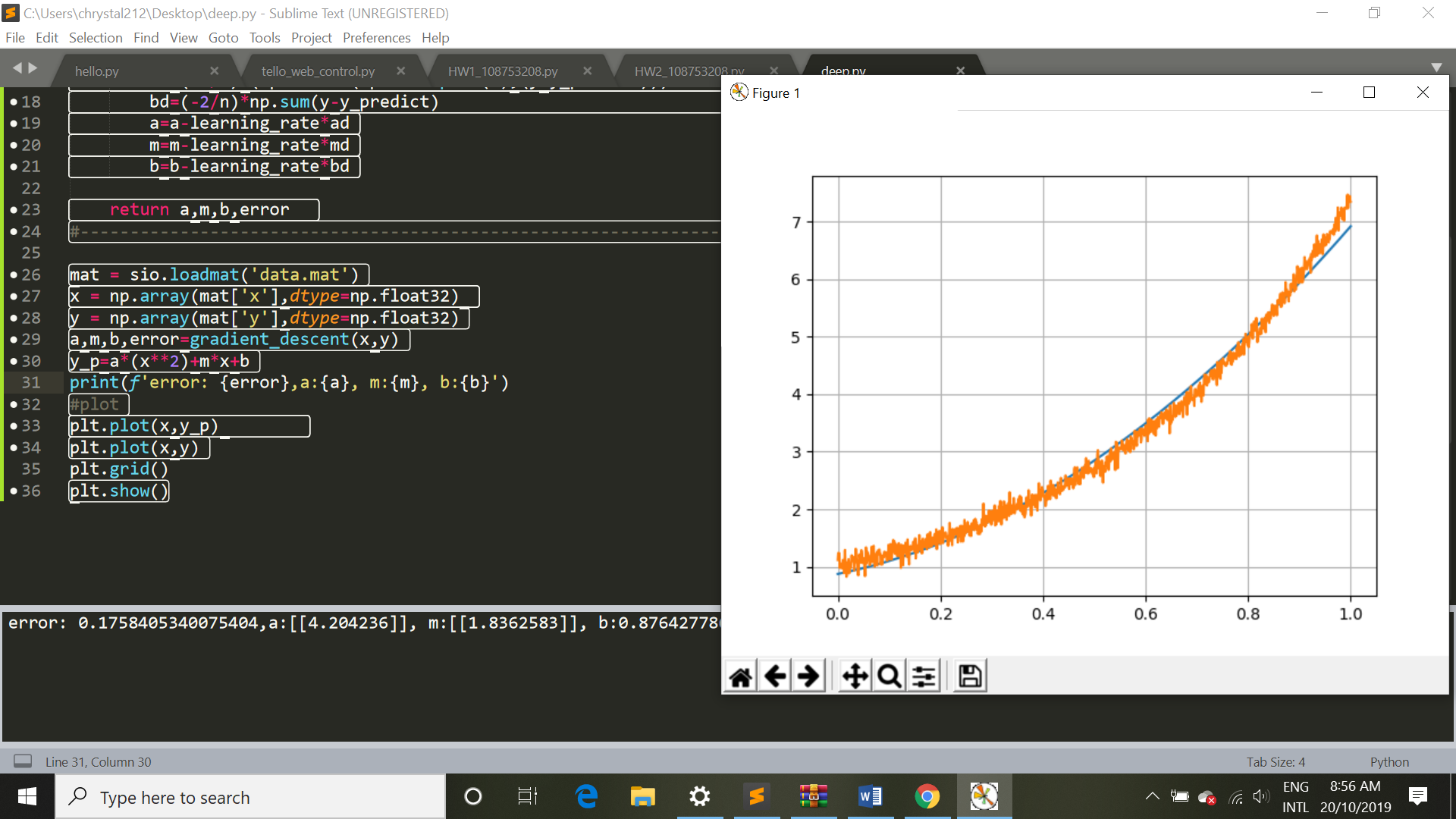
plt.show()

1. (20%) Using the same data from Question 2, compute the least square parabola (i.e. second order polynomial ) to fit the data. (5%) Explain which formulation (line or parabola) is more suitable for this dataset and why? (**paste your source code and show the results in your report**) **.(use gradient descent and pseudo inverse to get a bonus)**

我用gradient descent 算出所求之二次曲線為4.2x^2+1.83x+0.87

從第二題和第三題可以很明顯看出，second order polynomial 的fit 較好，較能說明這個dataset，因為error變小了，而且原始data分佈比較接近曲線。





import scipy.io as sio

import matplotlib.pyplot as plt

import numpy as np

#-----------------------------------------------------------------------

def gradient\_descent(x,y):

a=0 #square term

m=0

b=0 #y'=ax^2+mx+b

iteration=10000

n=len(x)

learning\_rate=0.01

for i in range(iteration):

y\_predict=a\*(x\*\*2)+m\*x+b

error=np.sqrt((1/n)\*np.sum(np.square(y-y\_predict)))

ad=(-2/n)\*(np.matmul(np.transpose(x\*\*2),(y-y\_predict))) #a derivitive

md=(-2/n)\*(np.matmul(np.transpose(x),(y-y\_predict))) #m derivitive

bd=(-2/n)\*np.sum(y-y\_predict) #b derivitive

a=a-learning\_rate\*ad

m=m-learning\_rate\*md

b=b-learning\_rate\*bd

return a,m,b,error

#----------------------------------------------------------------------

mat = sio.loadmat('data.mat')

x = np.array(mat['x'],dtype=np.float32)

y = np.array(mat['y'],dtype=np.float32)

a,m,b,error=gradient\_descent(x,y)

y\_p=a\*(x\*\*2)+m\*x+b

print(f'error: {error},a:{a}, m:{m}, b:{b}')

#plot

plt.plot(x,y\_p)

plt.plot(x,y)

plt.grid()

plt.show()

1. (40%) Download the MNIST dataset using the following example code:

##############################################

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import mnist

# input image dimensions 28x28

img\_rows, img\_cols = 28, 28

# the data, split between train and test sets

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

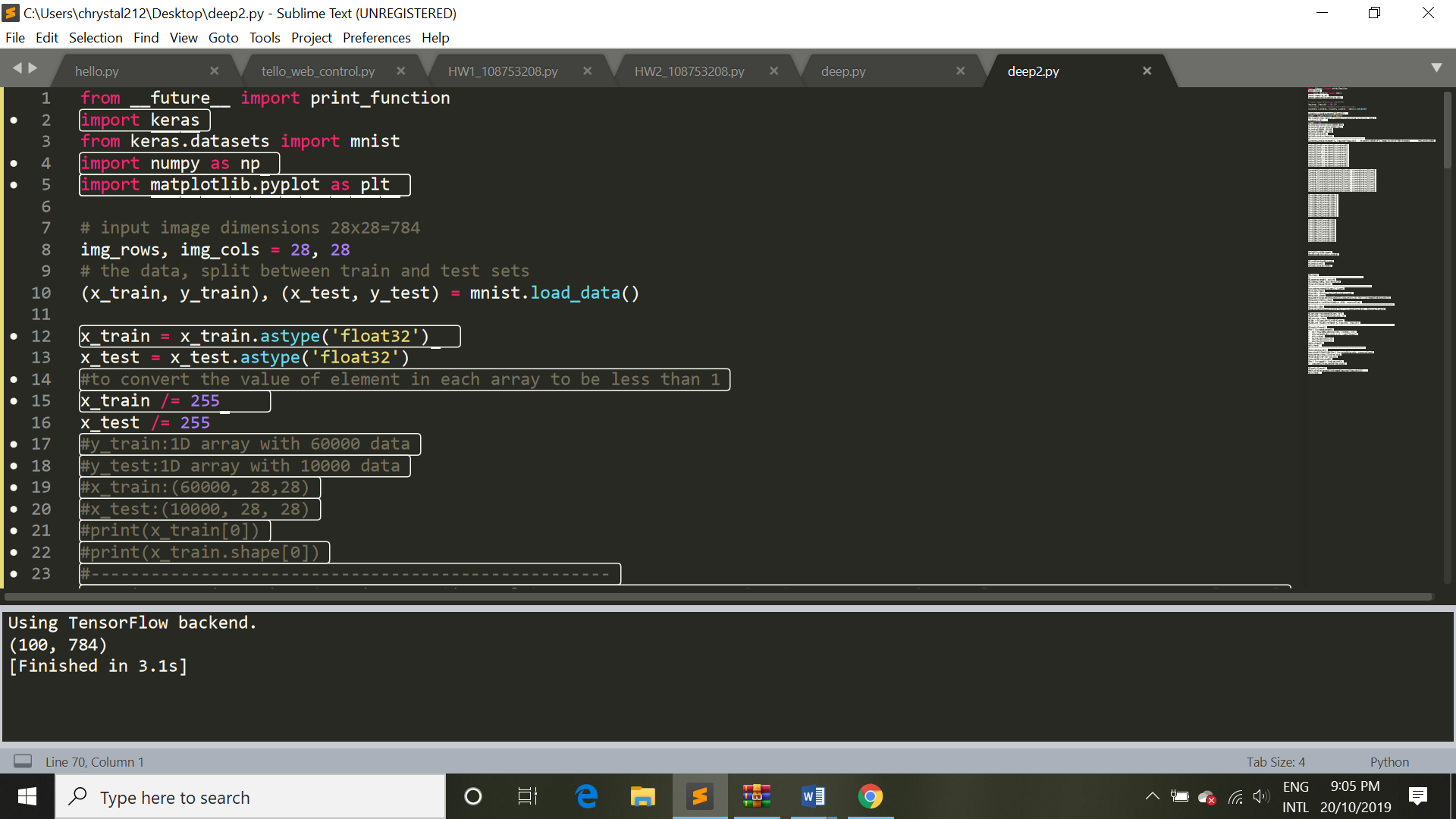
x\_train /= 255

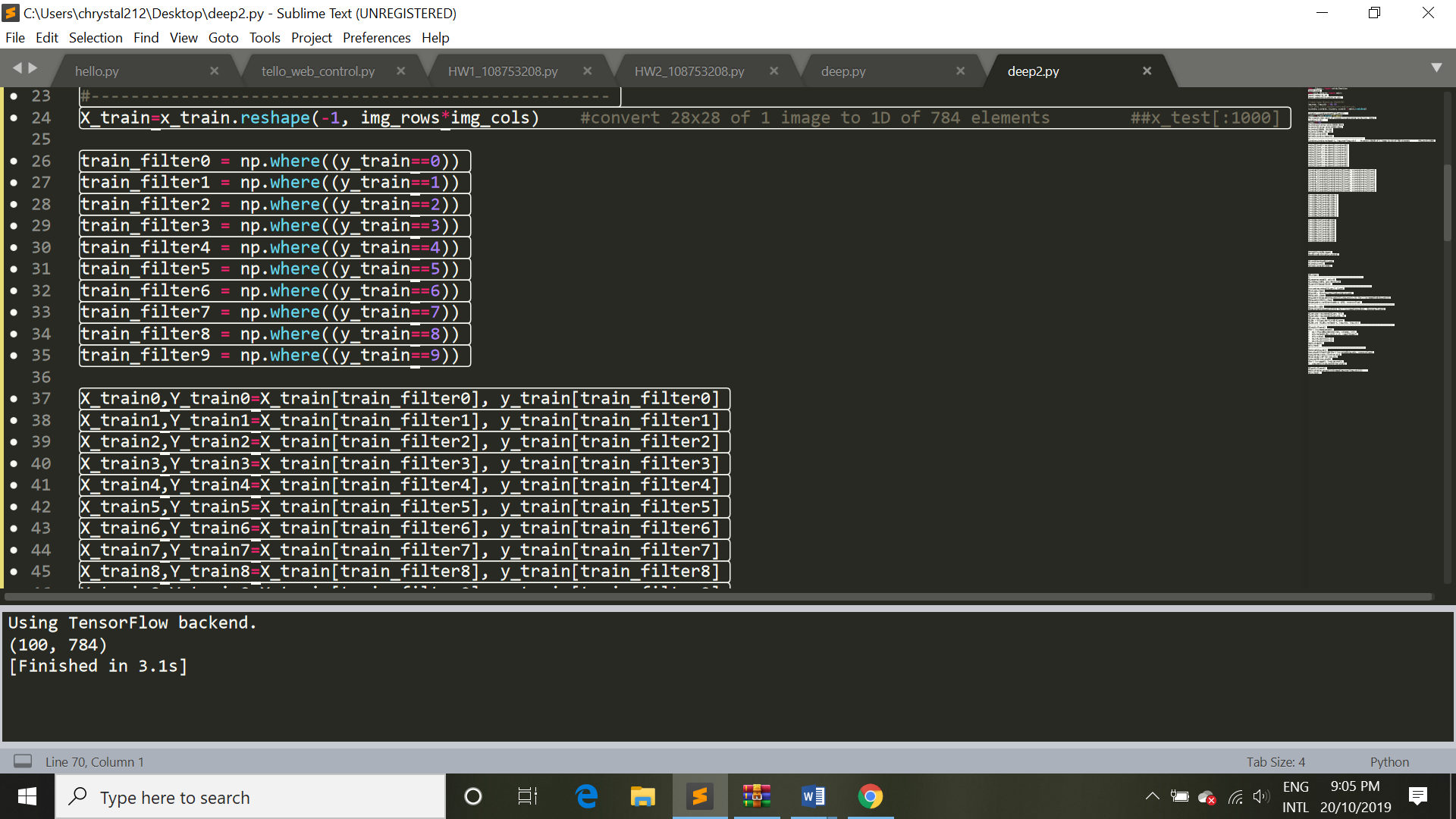
x\_test /= 255

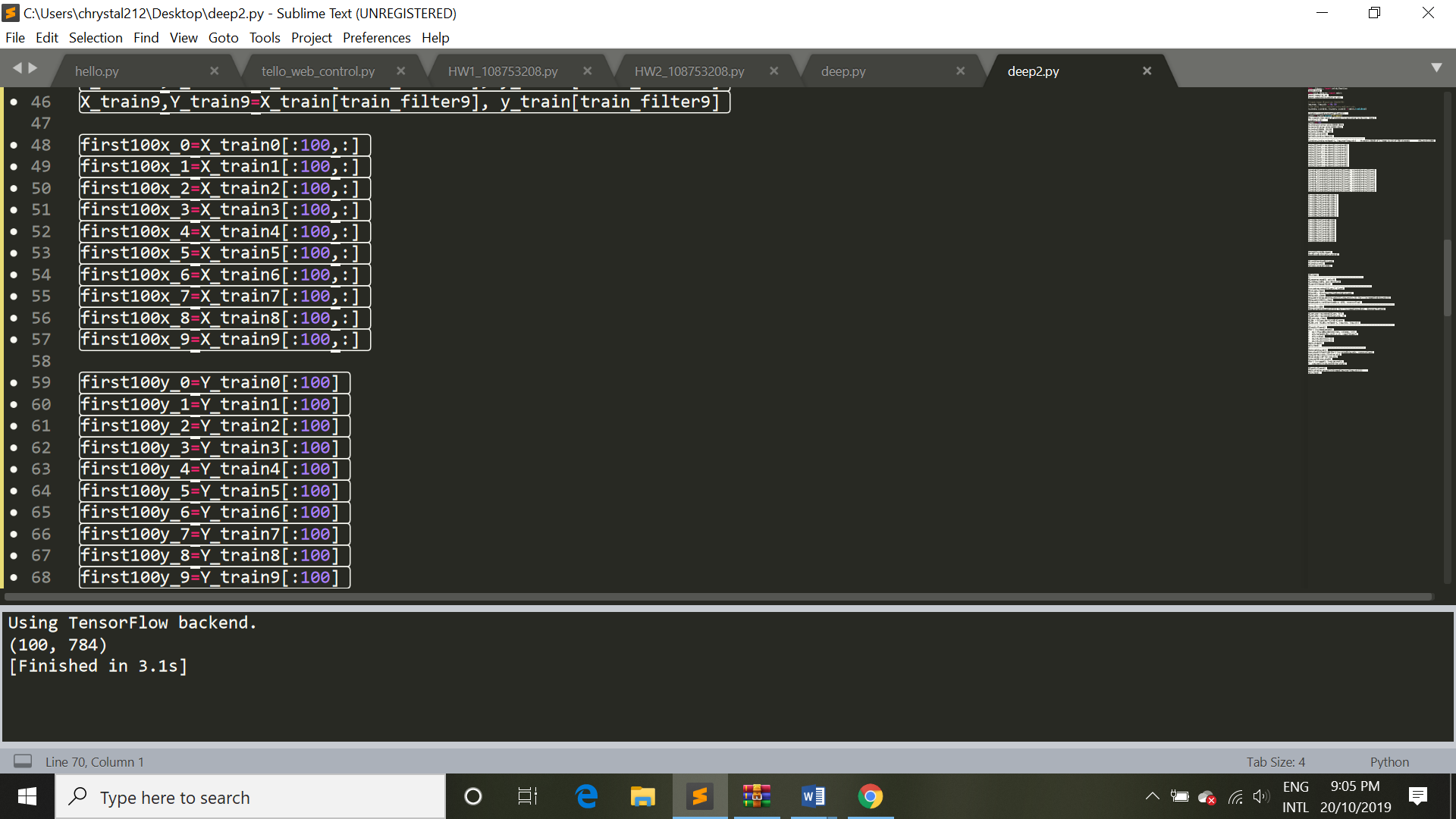
##############################################

**For the following questions, please upload the source code to moodle and show the results in your report.**

* 1. (5%) Please draw(take) 1,000 different handwritten images from either the training or the testing dataset to form your own dataset, where each digit has 100 data samples.







from \_\_future\_\_ import print\_function

import keras

from keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

# input image dimensions 28x28=784

img\_rows, img\_cols = 28, 28

# the data, split between train and test sets

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

#to convert the value of element in each array to be less than 1

x\_train /= 255

x\_test /= 255

#y\_train:1D array with 60000 data

#y\_test:1D array with 10000 data

#x\_train:(60000, 28,28)

#x\_test:(10000, 28, 28)

#print(x\_train[0])

#print(x\_train.shape[0])

#----------------------------------------------------

X\_train=x\_train.reshape(-1, img\_rows\*img\_cols) #convert 28x28 of 1 image to 1D of 784 elements ##x\_test[:1000]

train\_filter0 = np.where((y\_train==0))

train\_filter1 = np.where((y\_train==1))

train\_filter2 = np.where((y\_train==2))

train\_filter3 = np.where((y\_train==3))

train\_filter4 = np.where((y\_train==4))

train\_filter5 = np.where((y\_train==5))

train\_filter6 = np.where((y\_train==6))

train\_filter7 = np.where((y\_train==7))

train\_filter8 = np.where((y\_train==8))

train\_filter9 = np.where((y\_train==9))

X\_train0,Y\_train0=X\_train[train\_filter0], y\_train[train\_filter0]

X\_train1,Y\_train1=X\_train[train\_filter1], y\_train[train\_filter1]

X\_train2,Y\_train2=X\_train[train\_filter2], y\_train[train\_filter2]

X\_train3,Y\_train3=X\_train[train\_filter3], y\_train[train\_filter3]

X\_train4,Y\_train4=X\_train[train\_filter4], y\_train[train\_filter4]

X\_train5,Y\_train5=X\_train[train\_filter5], y\_train[train\_filter5]

X\_train6,Y\_train6=X\_train[train\_filter6], y\_train[train\_filter6]

X\_train7,Y\_train7=X\_train[train\_filter7], y\_train[train\_filter7]

X\_train8,Y\_train8=X\_train[train\_filter8], y\_train[train\_filter8]

X\_train9,Y\_train9=X\_train[train\_filter9], y\_train[train\_filter9]

first100x\_0=X\_train0[:100,:]

first100x\_1=X\_train1[:100,:]

first100x\_2=X\_train2[:100,:]

first100x\_3=X\_train3[:100,:]

first100x\_4=X\_train4[:100,:]

first100x\_5=X\_train5[:100,:]

first100x\_6=X\_train6[:100,:]

first100x\_7=X\_train7[:100,:]

first100x\_8=X\_train8[:100,:]

first100x\_9=X\_train9[:100,:]

first100y\_0=Y\_train0[:100]

first100y\_1=Y\_train1[:100]

first100y\_2=Y\_train2[:100]

first100y\_3=Y\_train3[:100]

first100y\_4=Y\_train4[:100]

first100y\_5=Y\_train5[:100]

first100y\_6=Y\_train6[:100]

first100y\_7=Y\_train7[:100]

first100y\_8=Y\_train8[:100]

first100y\_9=Y\_train9[:100]

* 1. (5%) Use the following code to show 50 images in your own dataset.

##############################################

import numpy as np

import matplotlib.pyplot as plt

amount= 50

lines = 5

columns = 10

number = np.zeros(amount)

for i in range(amount):

number[i] = y\_test[i]

# print(number[0])

fig = plt.figure()

for i in range(amount):

ax = fig.add\_subplot(lines, columns, 1 + i)

plt.imshow(x\_test[i,:,:], cmap='binary')

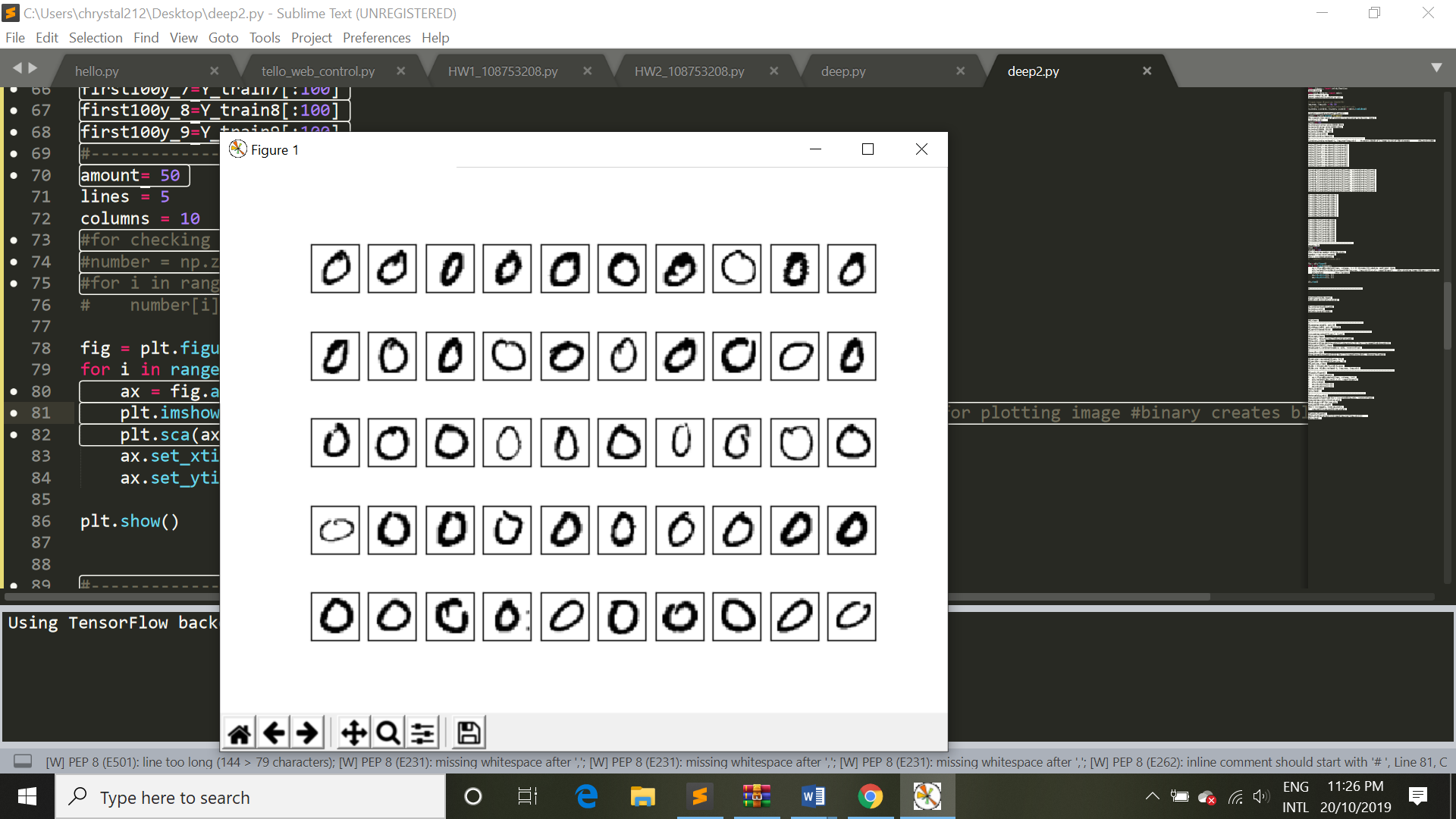
plt.sca(ax)

ax.set\_xticks([], [])

ax.set\_yticks([], [])

plt.show()

我是用first100x\_0 的dataset 來印50個不同的0



amount= 50

lines = 5

columns = 10

#number = np.zeros(amount)

#for i in range(amount):

# number[i] = first100y\_0[i]

fig = plt.figure()

for i in range(amount):

ax = fig.add\_subplot(lines, columns, 1 + i)

plt.imshow(first100x\_0.reshape(100,img\_rows, img\_cols)[i,:,:], cmap='binary')

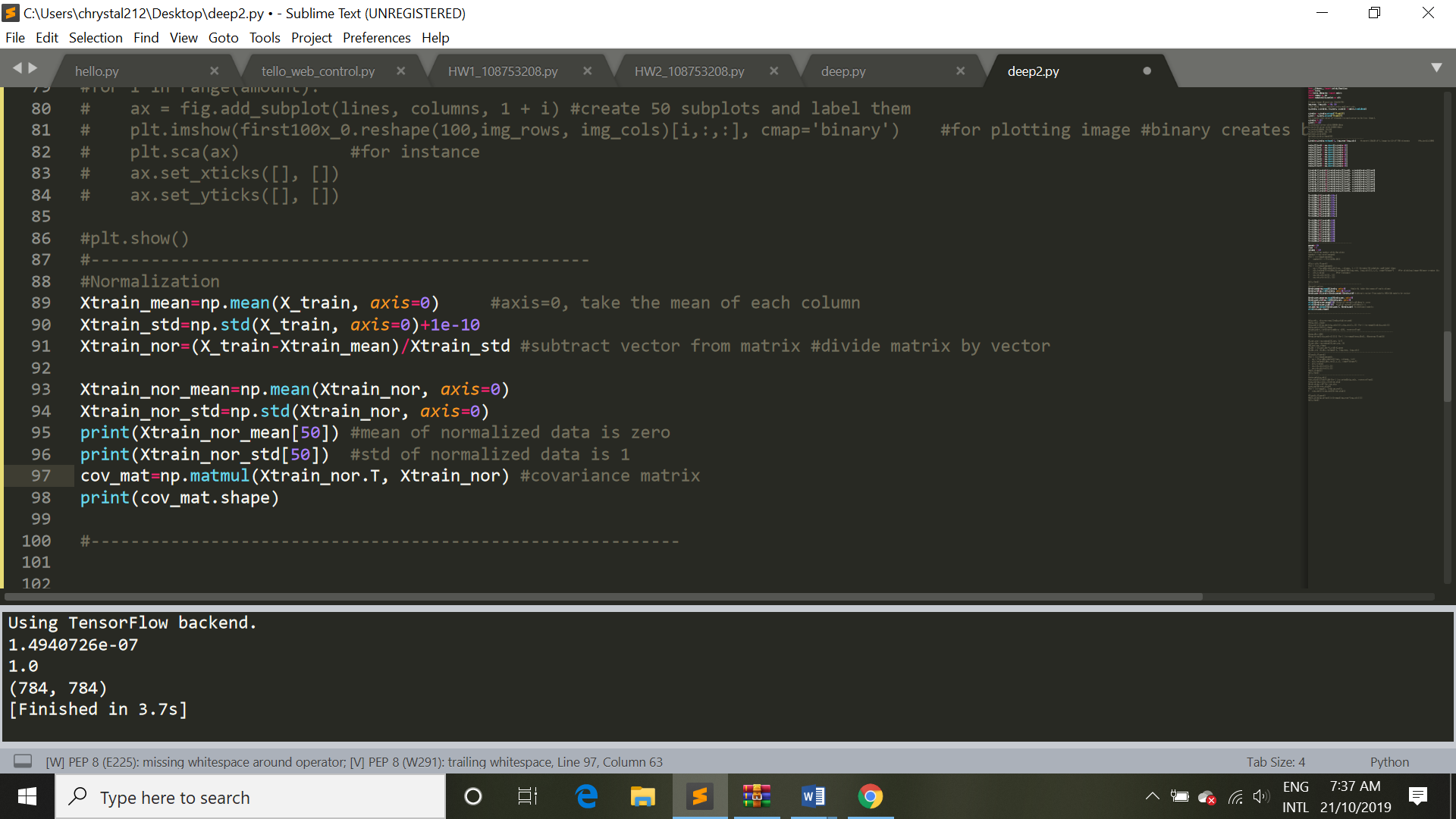
plt.sca(ax)

ax.set\_xticks([], [])

ax.set\_yticks([], [])

plt.show()

4.3(5%) Normalize the data (subtracting the mean from it and then dividing it by the standard deviation) and compute the covariance matrix for the data.



#Normalization

Xtrain\_mean=np.mean(X\_train, axis=0) #axis=0, take the mean of each column

Xtrain\_std=np.std(X\_train, axis=0)+1e-10

Xtrain\_nor=(X\_train-Xtrain\_mean)/Xtrain\_std #subtract vector from matrix #divide matrix by vector

Xtrain\_nor\_mean=np.mean(Xtrain\_nor, axis=0)

Xtrain\_nor\_std=np.std(Xtrain\_nor, axis=0)

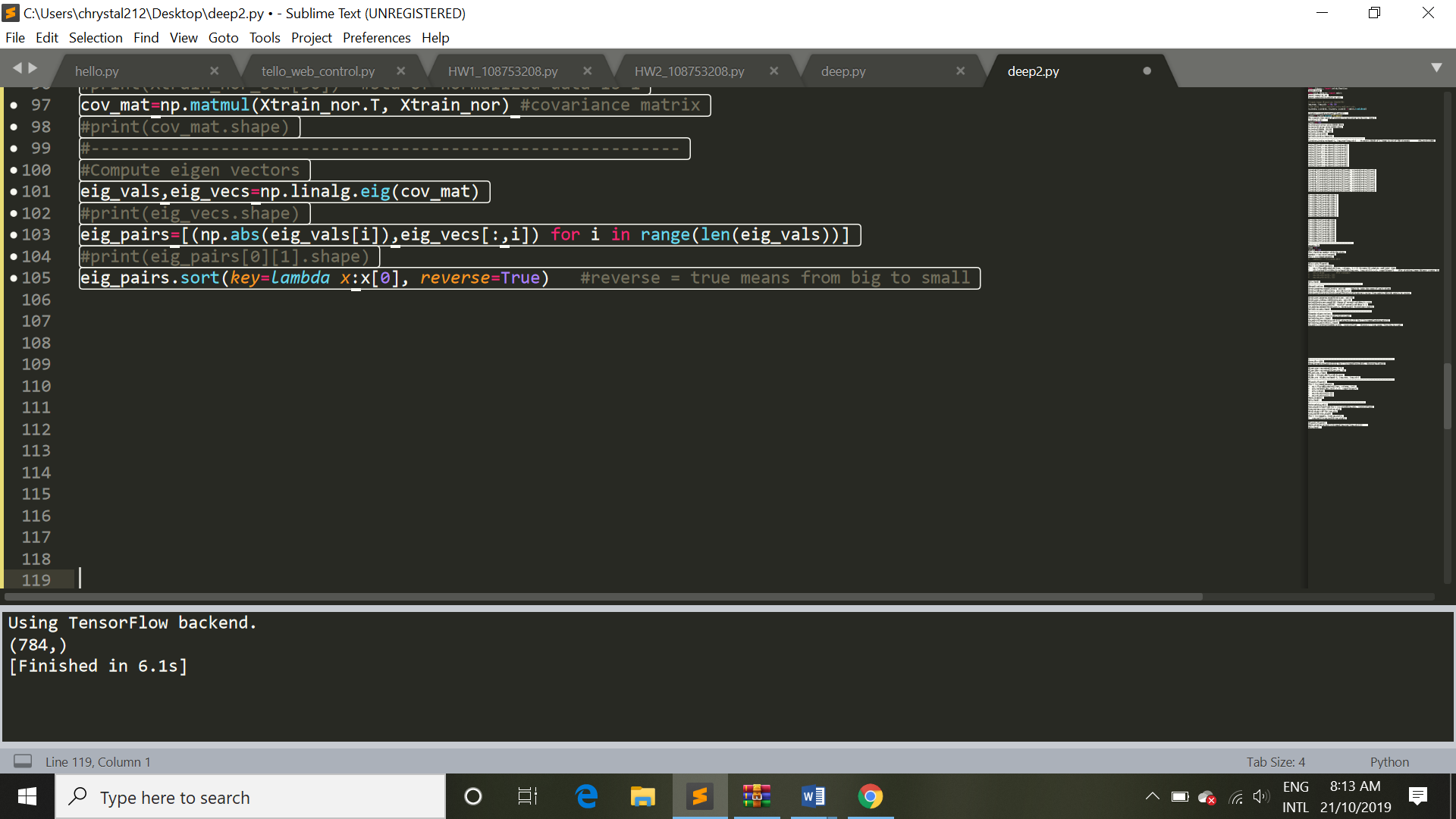
print(Xtrain\_nor\_mean[50]) #mean of normalized data is zero

print(Xtrain\_nor\_std[50]) #std of normalized data is 1

cov\_mat=np.matmul(Xtrain\_nor.T, Xtrain\_nor) #covariance matrix

print(cov\_mat.shape)

4.4(10%) Compute eigenpairs(eigen value and corresponding eigenvectors) for the covariance (sorted in a descending order based on eigenvalues).



#Compute eigen vectors

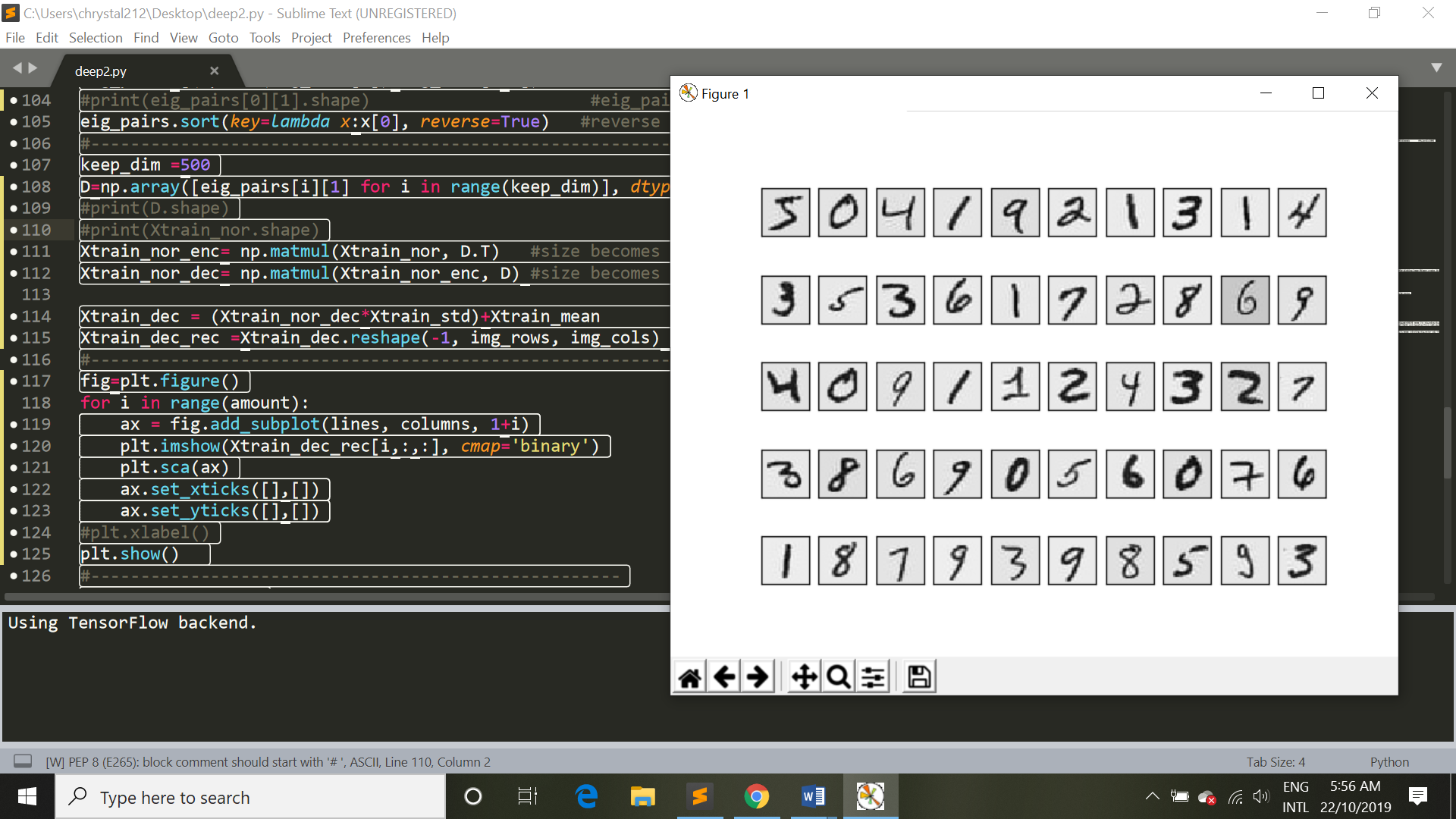
eig\_vals,eig\_vecs=np.linalg.eig(cov\_mat)

eig\_pairs=[(np.abs(eig\_vals[i]),eig\_vecs[:,i]) for i in range(len(eig\_vals))]

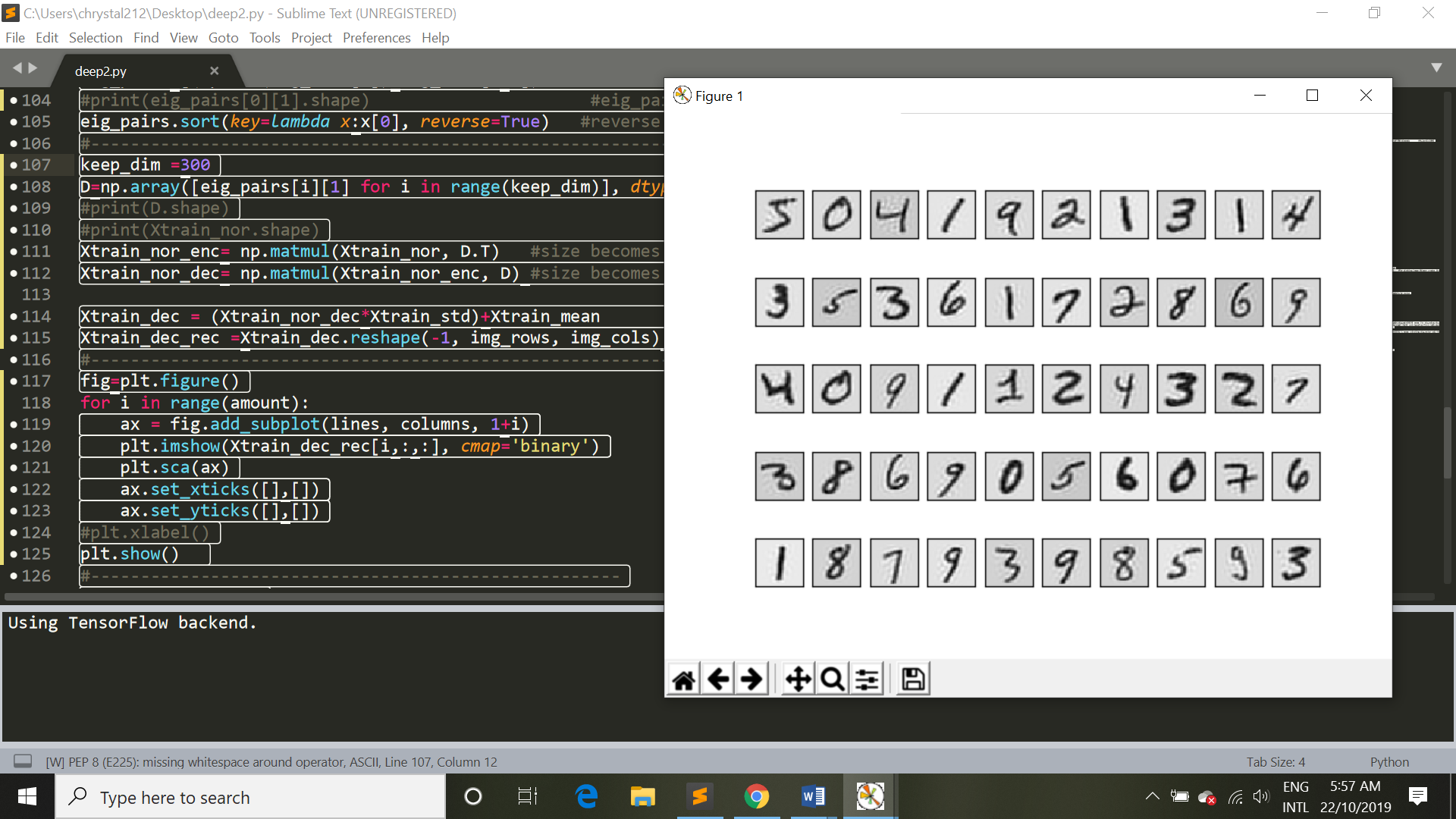
eig\_pairs.sort(key=lambda x:x[0], reverse=True)

4.5(13%) Please use PCA to reduce the 784 dimensional data to that with 500, 300, 100, and 50 dimensions, and then show the decoding results, respectively. How do you interpret the results?

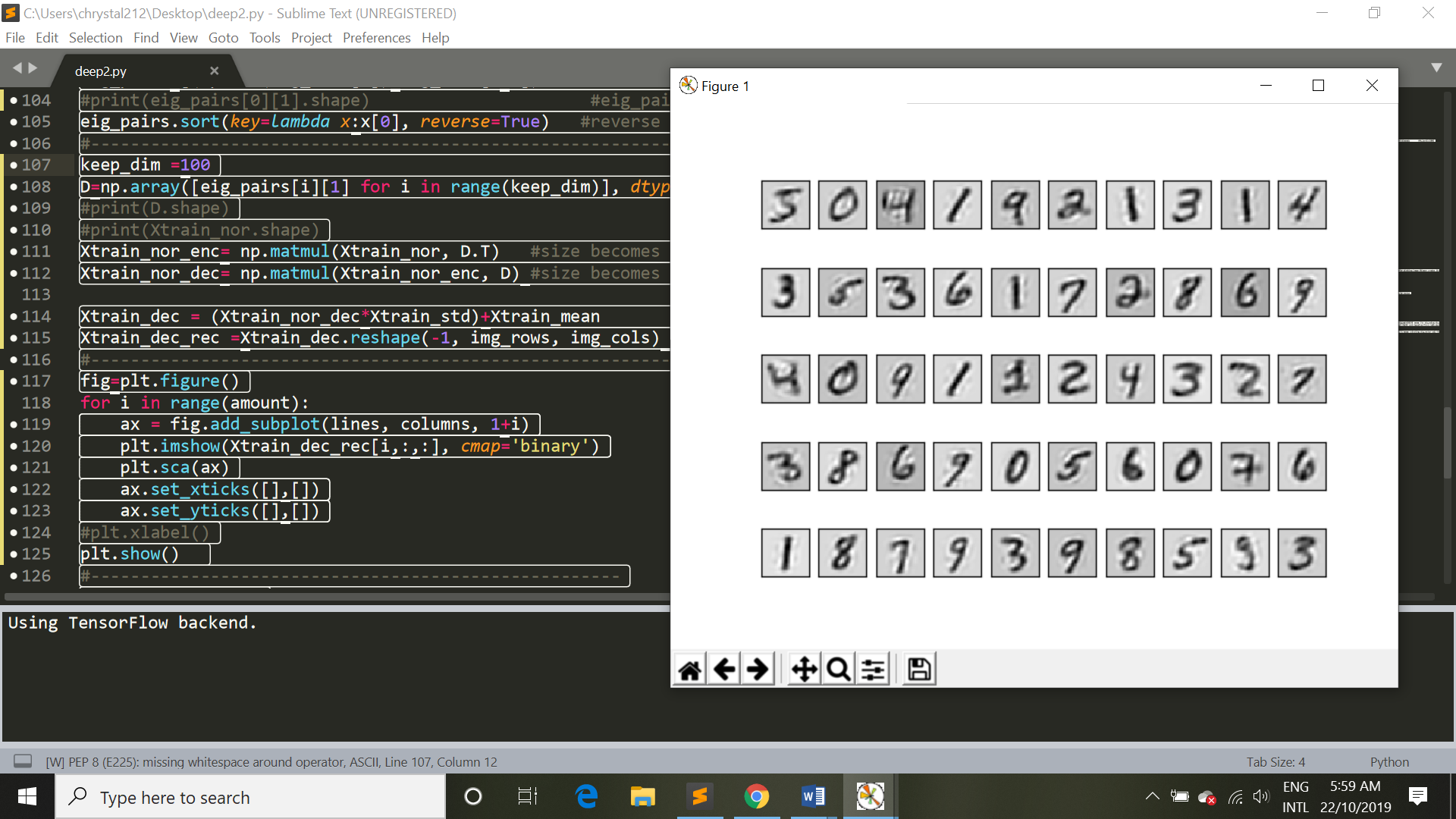
500:



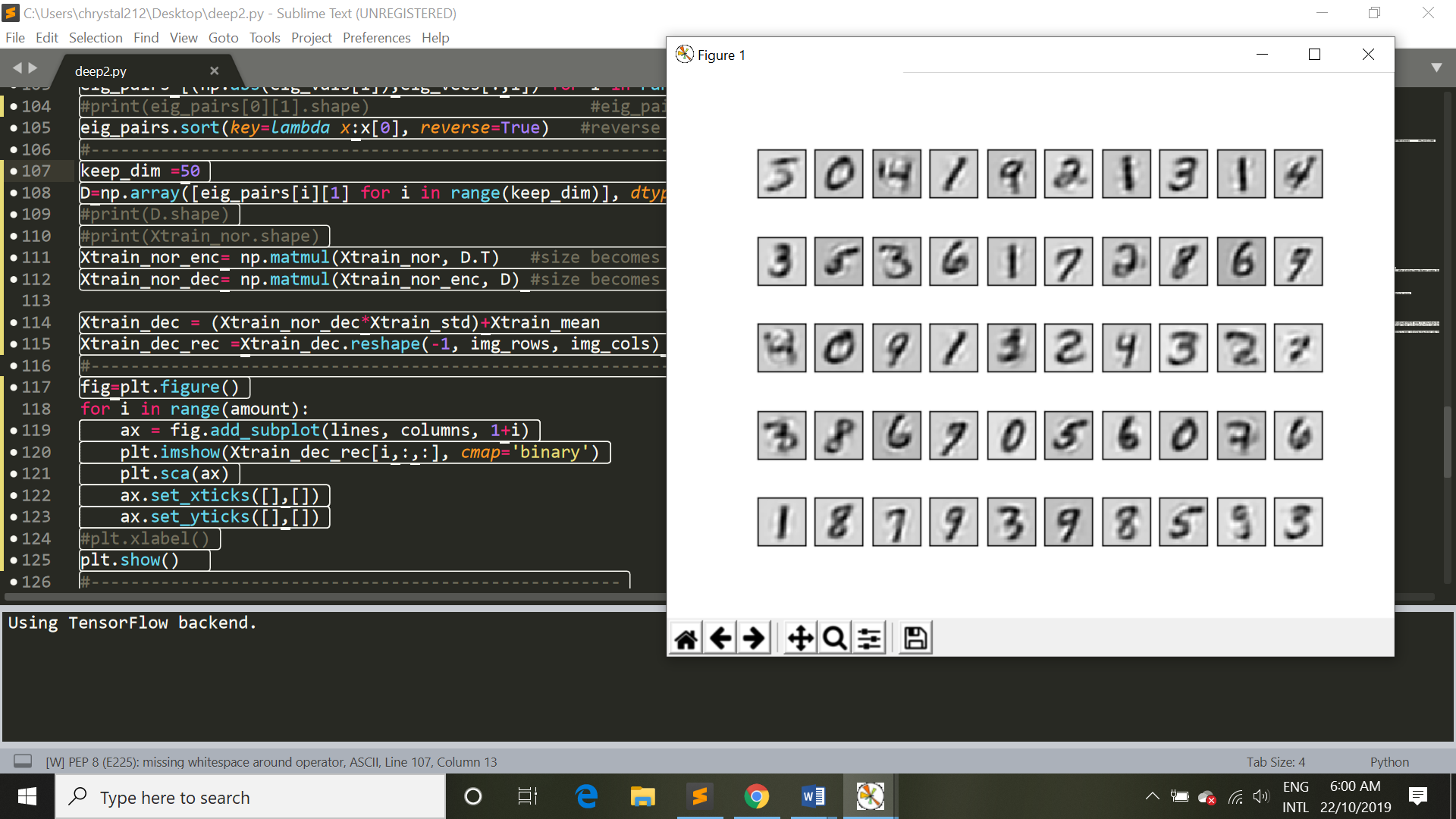
300:



100:



50:



=>當你採取的dimension 越少時，你所呈現的資料越模糊。

keep\_dim =50

D=np.array([eig\_pairs[i][1] for i in range(keep\_dim)], dtype=np.float32)

Xtrain\_nor\_enc= np.matmul(Xtrain\_nor, D.T) #size becomes 60000x500

Xtrain\_nor\_dec= np.matmul(Xtrain\_nor\_enc, D) #size becomes 60000x784

Xtrain\_dec = (Xtrain\_nor\_dec\*Xtrain\_std)+Xtrain\_mean #transform normalized data

Xtrain\_dec\_rec =Xtrain\_dec.reshape(-1, img\_rows, img\_cols)

#print the result

fig=plt.figure()

for i in range(amount):

ax = fig.add\_subplot(lines, columns, 1+i)

plt.imshow(Xtrain\_dec\_rec[i,:,:], cmap='binary')

plt.sca(ax)

ax.set\_xticks([],[])

ax.set\_yticks([],[])

plt.show()