Machine Learning
assignment report

Made by
Anglo Sherif Nabil Soliman
40-14526

- After importing all the libraries we need .. now it is the time to import the data . and this is done using pd.read_csv function

  data = pd.read_csv('C:\\semester 10\\machine l
  earning\\house_prices_data_training_data.csv')

- Then we drop all NAN values using dropna function.

- Lets delete the columns of id and date as they dont matter in the dataset
- We preview the data using data.head() function.

- Then it is the time to apply the correlation function to see which features are correlated the price of the house , the features that have weak correlation are dropped

- i chose the value 0.3 as a threshold , and the correlation between a feature and the price is less than this value , this feature will be droped . this is done to simplify the dataset for future analysis.

- .corr() function is used to see the correlation between features.

- Then we make normalization . the function of the normalization is the (feature-mean) / standard deviation of these features. normalization is done to reduce the fluctuation in the output.

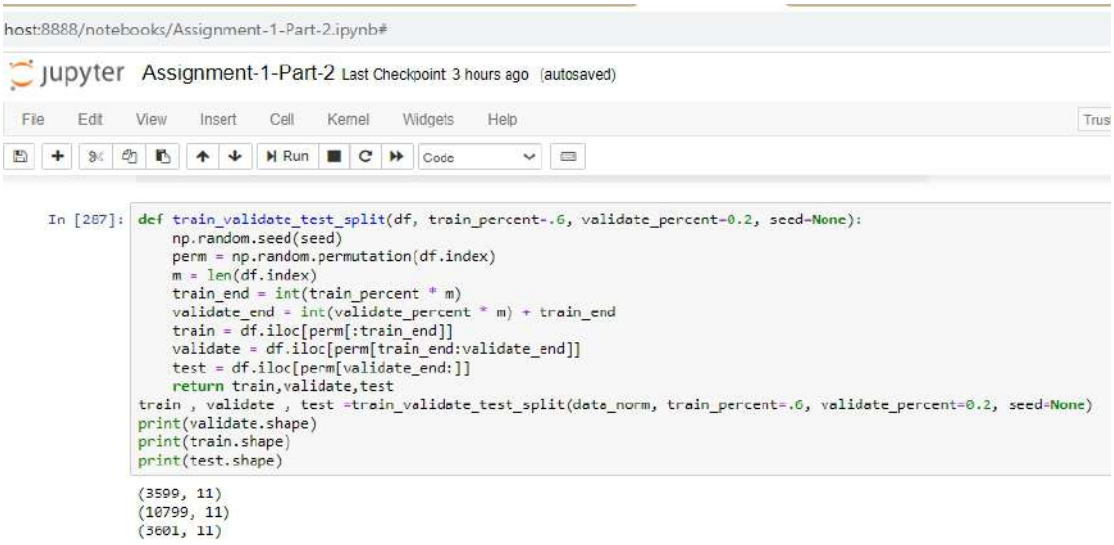
- I made normalization for both features and the output which is the price

```
17999
```

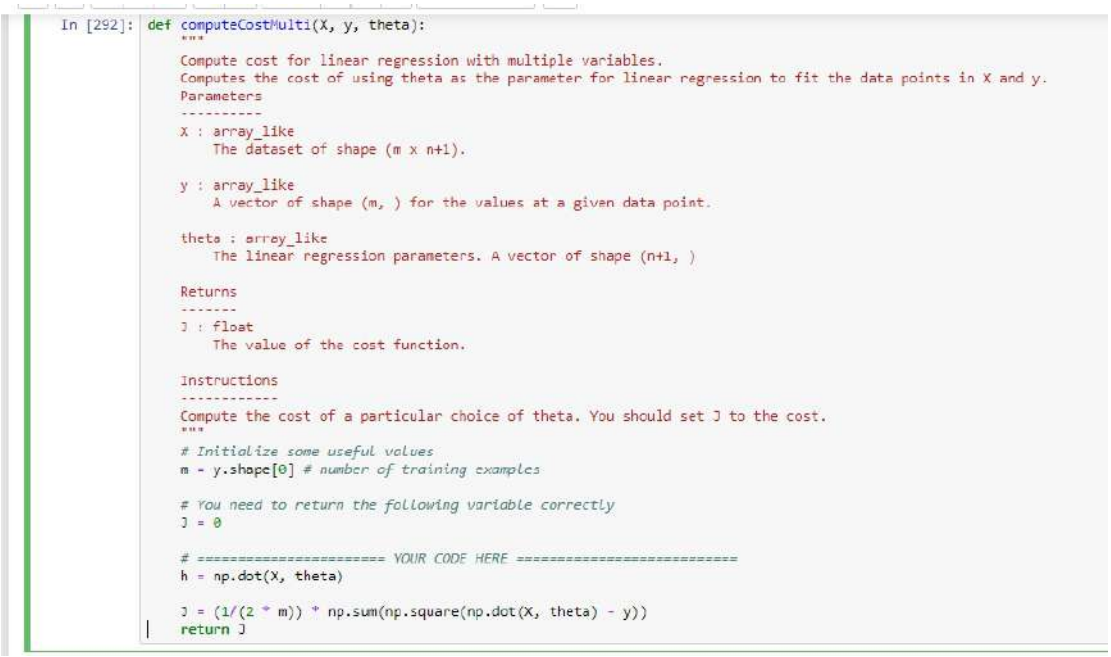
```
In [285]: def  featureNormalize(X):
              """
              Normalizes the features in X. returns a normalized version of X where
              the mean value of each feature is 0 and the standard deviation
              is 1. This is often a good preprocessing step to do when working with
              learning algorithms.


              """
              # You need to set these values correctly
              X_norm = X.copy()
              mu = np.zeros(X.shape[1])
              sigma = np.zeros(X.shape[1])

              # ---------------------------- YOUR CODE HERE ----------------------
              mu = np.mean(X, axis = 0)
              sigma = np.std(X, axis = 0)
              X_norm = (X - mu) / sigma

              # ============================================================
              return X_norm, mu, sigma
```

```
In [286]: data_norm, mu, sigma = featureNormalize(data)

          print('Computed mean:', mu)
          print('Computed standard deviation:', sigma)
          #print(data_norm)
          data_norm = np.concatenate([np.ones((m, 1)), data_norm], axis=1)
          data_norm=pd.DataFrame(data_norm)
          data_norm.head()
```

- In this assignment , we will do model selection , and this is done by splitting the dataset as follows . 60% for the traing part , 20% for cross validation and finally 20% for the testing part.



- Now to the compute cost part . this function is used to calculate the error when we apply linear regression technique using thetas.
- The compute cost function is called in the gradient descent function

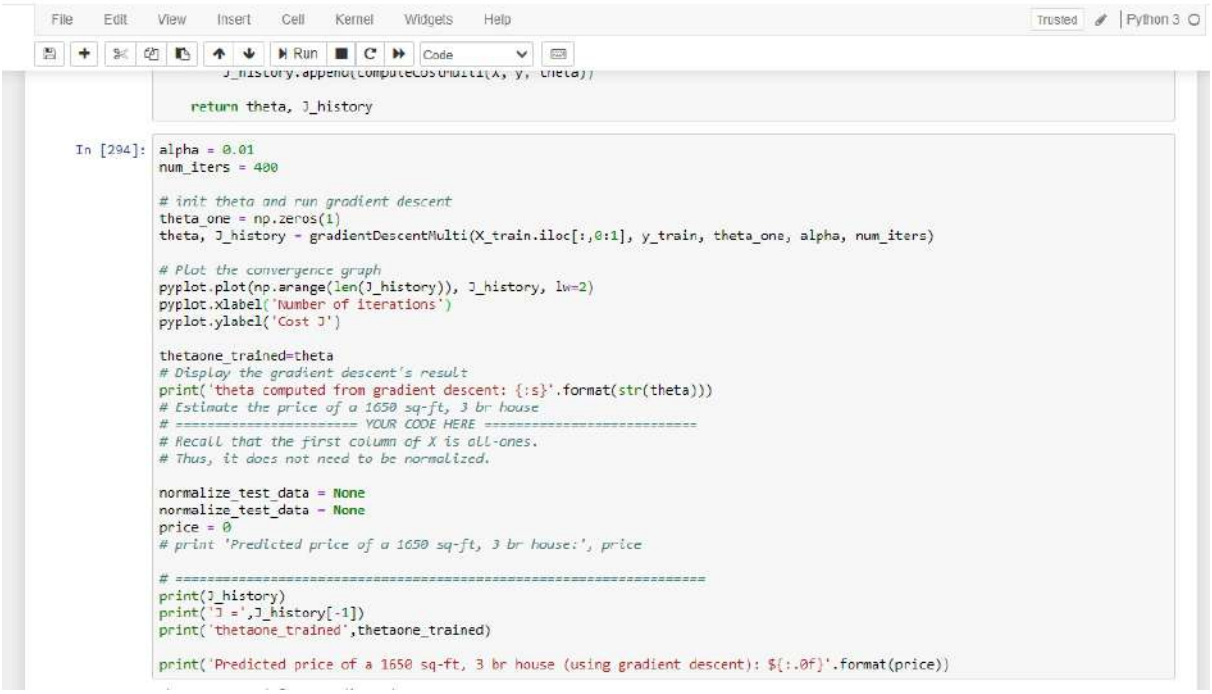


- Now to the gradient descent function

- This function is used to learn the theta values.

- The function takes 5 input parameters and they are . X which is the features of the dataset , Y which is the output of the dataset which is the price in our case . alpha , which is the learning rate , and the last parameter is the number of iterations to run gradient descent.

- The gradient descent function returns two values which are the thetas in type of array , and the second value returned is the J value which is calculated using the called function “compute cost”.



- In the previous figure I chose the learning rate to be equal to 0.01 and the number of iterations = 400

- We keep applying this function to each parameter and the function returns an array of thetas for this parameter.

- So for example .. if we applied this function on the 4th feature , we will get a vector that contains 4 values. In addition to the J (error) value,

- The function is applied on the test part only

- The following figure is results for the outputs of the gradient descent function that is used on test part.


```
theta_four = np.zeros(4)
theta, J_history = gradientDescentMulti(X_train.iloc[:,0:4], y_train, theta_four, alpha, num_iters)

# Plot the convergence graph
pyplot.plot(np.arange(len(J_history)), J_history, lw=2)
pyplot.xlabel('Number of iterations')
pyplot.ylabel('Cost J')

thetatfour_trained=theta

# Display the gradient descent's result
print('theta computed from gradient descent: {:s}'.format(str(theta)))
# Estimate the price of a 1650 sq-ft, 3 br house
# ---------------------- YOUR CODE HERE --------------------------
# Recall that the first column of X is all-ones.
# Thus, it does not need to be normalized.

normalize_test_data = None
normalize_test_data = None
price = 0
# print 'Predicted price of a 1650 sq-ft, 3 br house:', price
# ============================================================
print(J_history)
print('J =',J_history[-1])
print('Predicted price of a 1650 sq-ft, 3 br house (using gradient descent): ${:.0f}'.format(price))
```

```
theta computed from gradient descent: 2   -0.058187
3    0.127620
4    0.517108
5    0.219343
Name: 1, dtype: float64
[0.4674354746553331, 0.45824935732630107, 0.4494658176293099, 0.4410666163793238, 0.433034347952291, 0.4253524020800311, 0.4180
0492739810086, 0.41097679666616943, 0.40425357358413166, 0.39782148113071086, 0.3916673713546657, 0.3857786965519252, 0.3801434
817650362, 0.3747502965442278, 0.369586239912185, 0.3646468964772814, 0.3599163336425578, 0.35536706986015384, 0.35105005588320
665, 0.3468966549694376?, 0.342918623992745?4, 0.339108095421132?, 0.335457560121202?6, 0.331959850951293?5, 0.328608127107050
7, 0.3253958591849157, 0.3223168149305659?, 0.319365045640889, 0.3165348731894886, 0.313820877647111?, 0.311217885469690?, 0.3
0872095822796475, 0.30632538185380714, 0.30402665637956977, 0.3018204861478062, 0.29970277046979593, 0.29766959471227383, 0.295
```

- Now it is the time to apply the compute cost function to the cross validation part to get the J values.

Now Lets get J values fo the validation part

In [303]:
```
j1=computeCostMulti(X_val.iloc[:,0:1], y_val, thetaone_trained)
j2=computeCostMulti(X_val.iloc[:,0:2], y_val, thetatwo_trained)
j3=computeCostMulti(X_val.iloc[:,0:3], y_val, thetathree_trained)
j4=computeCostMulti(X_val.iloc[:,0:4], y_val, thetatfour_trained)
j5=computeCostMulti(X_val.iloc[:,0:5], y_val, thetatfive_trained)
j6=computeCostMulti(X_val.iloc[:,0:6], y_val, thetatsix_trained)
j7=computeCostMulti(X_val.iloc[:,0:7], y_val, thetatseven_trained)
j8=computeCostMulti(X_val.iloc[:,0:8], y_val, thetateight_trained)
j9=computeCostMulti(X_val.iloc[:,0:9], y_val, thetatnine_trained)
#theta_one_val = thetaone_trained
#theta, J_history = gradientDescentMulti(X_val.iloc[:,0:1], y_val, theta_one_val, alpha, num_iters)
print ('j1=',j1)
print ('j2=',j2)
print ('j3=',j3)
print ('j4=',j4)
print ('j5=',j5)
print ('j6=',j6)
print ('j7=',j7)
print ('j8=',j8)
print ('j9=',j9)
```

```
j1= 0.4077958609379826
j2= 0.3208663883956447
j3= 0.22859161568597522
j4= 0.2063628890442122
j5= 0.18810062296397695
j6= 0.18887335860798232
j7= 0.1875871309912664
j8= 0.15788732614573436
j9= 0.158157204845281
```

Results :

- We notice that from the J values that we got from applying gradient d escent function that the least J value that J8 is least error in all J value s. With value equal to 0.17350347673138822
- And after applying the compute cost function on each feature we see that also the least J value that J8 is least error in all J value with value equal to 0.157887
- So we can deduce that the feature of J8 which is feature 7 is the best degree in the dataset
- then now is the time to compute the general error in the test part and lets call this error J_general.

```
print ('j8=',j8)
print ('j9=',j9)
```

```
j1= 0.4077958609379826
j2= 0.3208663883956447
j3= 0.22859161568597522
j4= 0.2063628890442122
j5= 0.18810062296397695
j6= 0.18887335860798232
j7= 0.1875871309912664
j8= 0.15788732614573436
j9= 0.158157204845281
```

So the least J is j8 (feature 7) ... in the training part also feature 7 has the least J ... so Degree 7 is the best degree

In [304]:
```
#thetafinal=[thetaone_trained,thetatwo_trained,thetathree_trained,thetatfour_trained,thetatfive_trained,thetatsix_trained,thetat

j_general=computeCostMulti(X_test.iloc[:,0:8], y_test, thetateight_trained)
print ('j_general',j_general)
```

```
j_general 0.23966606069854435
```

In [ ]: