Lab1 - IE406

Sharvil Sheth - 201901201, Prabhav Shah - 201901216, Mohil Desai - 201901301, Dhaval Vaidya - 201901462, Vishvesh Patel - 201901464

Importing Libraries

In [45]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from mpl_toolkits.mplot3d import Axes3D
import math
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
```

Q1:

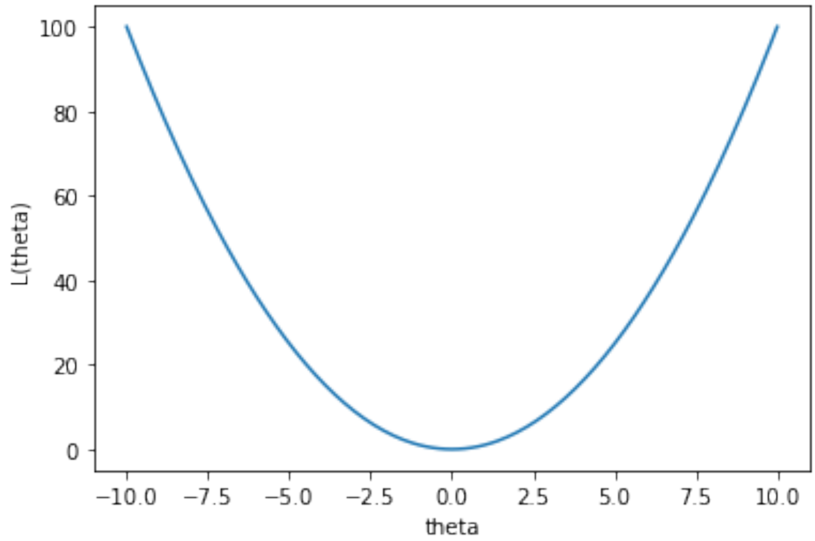
In [46]:
```
#read data from the CSV
data = pd.read_excel('Data for Lab 1.xlsx')
data.head()
```

Out[46]:

	x	y
0	3504	18.0
1	3693	15.0
2	3436	18.0
3	3433	16.0
4	3449	17.0

In [47]:
```
#store X and Y
x = data['x']
y = data['y']
```

In [48]:
```
theta = np.linspace(-10,10,201)
lOfTheta = np.square(theta)

#plot
plt.plot(theta, lOfTheta)
plt.xlabel('theta')
plt.ylabel('L(theta)')
plt.show()
```



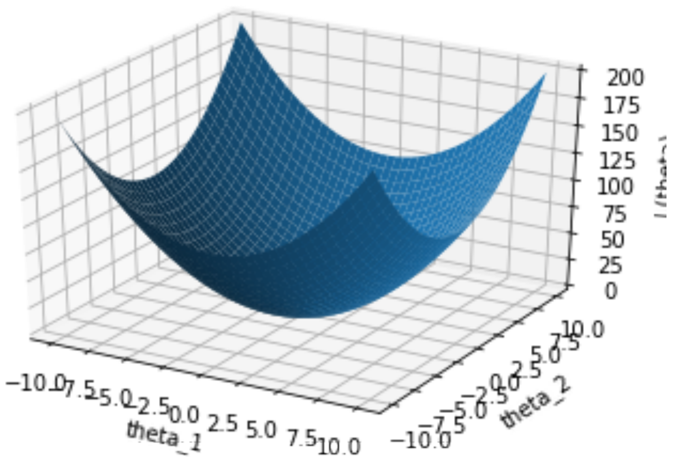
Observations:

Here we observe that the curve of L(theta) is parabolic in nature. It attends the minimum value at L(theta) = 0 at theta = 0.

Q2:

In [49]:

```
theta_1 = np.copy(theta)
theta_2 = np.copy(theta)

lOftheta = np.zeros((201, 201))
for i in range(201):
    for j in range(201):
        lOftheta[i][j] = ((theta_1[i]*theta_1[i]) + (theta_2[j]*theta_2[j]

theta_1, theta_2 = np.meshgrid(theta_1, theta_2)

#plot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X = theta_1, Y = theta_2, Z = lOftheta)
ax.set_xlabel('theta_1')
ax.set_ylabel('theta_2')
ax.set_zlabel('L(theta)')
plt.show()
```

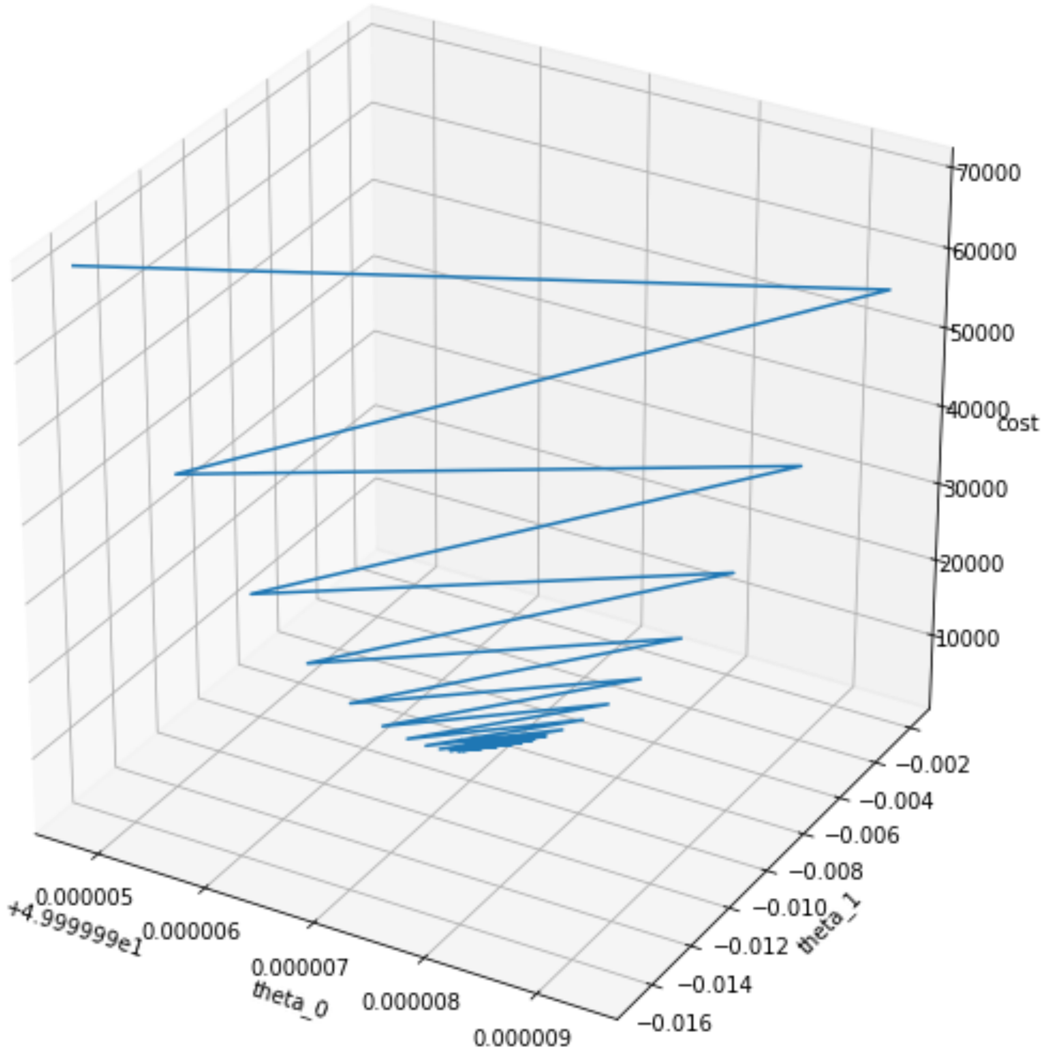


Here we get a contour plot of L(theta) for different values of theta_1 and theta_2. L(theta) attains its minimum value L(theta) = 0 at (theta_1,theta_2) = (0,0)

Q3a:

In [50]:

```
x = np.array(data['x'])
y = np.array(data['y'])

ones = np.ones(len(x))
ones = ones.reshape((94,1))

x = x.reshape((94,1))
x = np.append(ones, x,axis = 1)

iterations = 50
alpha = 0.0000001
th_0 = 50
th_1 = 0
n = len(x)
final_theta_0 = []
final_theta_1 = []
final_costs = []
for i in range(iterations):

        y_predicted = th_1 * x[:,1] + th_0 * x[:,0]

        cost = sum((y-y_predicted)**2)

        delta_th_1 = -(2/n)*sum(x[:,1]*(y-y_predicted))
        delta_th_0 = -(2/n)*sum(y-y_predicted)

        th_1 = th_1 - alpha * delta_th_1
        th_0 = th_0 - alpha * delta_th_0

        final_theta_0.append(th_0)
        final_theta_1.append(th_1)

        final_costs.append(cost)

final_theta_0 = np.array(final_theta_0)
final_theta_1 = np.array(final_theta_1)
final_costs = np.array(final_costs)

#plot
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')
ax.plot3D(final_theta_0,final_theta_1, final_costs)
ax.set_xlabel('theta_0')
ax.set_ylabel('theta_1')
ax.set_zlabel('cost')
plt.show()
```



In [51]:

```
min_L_theta = min(final_costs) # minimum value value of L(theta)
min_theta_0 = final_theta_0[np.argmin(final_costs)]
min_theta_1 = final_theta_1[np.argmin(final_costs)]
print(min_theta_0,min_theta_1,min_L_theta)
```

```
49.999996705417516 -0.008833886263553227 1576.7046182979047
```

Q3b:

In [52]:

```
x_temp = x
x_temp = x_temp.reshape((94,2))
y = y.reshape((94,1))
temp_x_t = x_temp.T.dot(x_temp)
theta = np.linalg.pinv(temp_x_t).dot(x_temp.T).dot(y)
print(theta)
```

```
[[ 4.92376299e+01]
 [-8.61193478e-03]]
```

Q4:

In [53]:
```
y_predicted = theta[0]*x[:,0] + theta[1]*x[:,1]
y = y.reshape(len(y_predicted))
L_theta = sum((y - y_predicted)**2)

print(L_theta)

theta_rand = np.array([55,-0.005])
y_predicted = theta_rand[0]*x[:,0] + theta_rand[1]*x[:,1]
y = y.reshape(len(y_predicted))
L_theta_rand = sum((y - y_predicted)**2)

print(L_theta_rand)
```

```
1572.6503668922924
27837.06390000001
```

Q5:

In [54]:
```
# Creating our dataset
X = np.array([[1, 2],[2, 4],[3, 6],[4, 8]])
Y = np.array([2,3,4,5])
Y = Y.reshape((4,1))
```

(a) using scikit-learn library

In [55]:
```
#Creating model() instance and training our model on data (X,Y)
model = linear_model.LinearRegression()
model.fit(X,Y)
```

Out[55]:
```
LinearRegression()
```

In [56]:
```
model.score(X,Y)
```

Out[56]:
```
1.0
```

In [57]:
```
model.coef_
```

Out[57]:
```
array([[0.2, 0.4]])
```

In [58]:
```
model.intercept_
```

Out[58]:
```
array([1.])
```

We get cofficients value [theta1,theta2] as [0.2,0.4] and intercept value as 1.

In equation, y = 0.2*x1 + 0.4*x2 + 1

In [59]:
```
#prediction with the help of trained model
model.predict([[5,10], [6,12], [7,14]])
```

Out[59]:
```
array([[6.],
       [7.],
       [8.]])
```

(b) Using normal equations

In [60]:
```
def normal_equation(X,y):
    x_tmp = X.T.dot(X)
    theta = np.linalg.inv(x_tmp).dot(X.T).dot(y)
    return theta

XX = np.c_[np.ones((4,1)), X]
theta = normal_equation(XX, Y)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-60-ab9013afbd25> in <module>
      5
      6 XX = np.c_[np.ones((4,1)), X]
----> 7 theta = normal_equation(XX, Y)

<ipython-input-60-ab9013afbd25> in normal_equation(X, y)
      1 def normal_equation(X,y):
      2     x_tmp = X.T.dot(X)
----> 3     theta = np.linalg.inv(x_tmp).dot(X.T).dot(y)
      4     return theta
      5

~\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in inv(a)
    549     signature = 'D->D' if isComplexType(t) else 'd->d'
    550     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 551     ainv = _umath_linalg.inv(a, signature=signature, extobj=extobj)
    552     return wrap(ainv.astype(result_t, copy=False))
    553

~\Anaconda3\lib\site-packages\numpy\linalg\linalg.py in _raise_linalgerror_
singular(err, flag)
     95
     96 def _raise_linalgerror_singular(err, flag):
---> 97     raise LinAlgError("Singular matrix")
     98
     99 def _raise_linalgerror_nonposdef(err, flag):

LinAlgError: Singular matrix
```

We got Singular Matrix Error because det(X'X) = 0. Even though, scikit learn can solve the model as it uses pseudo inverse instead of normal inverse. So, if we use pinv fucntion instead of inv function, we can still get the correct results.

Q6a:

In [61]:
```
dataset = pd.read_excel(r'Real estate valuation data set.xlsx')
X = dataset.iloc[:,1:7]
Y = dataset.iloc[:,7]
```

In [62]:
```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 1/3
```

In [63]:
```
reg = LinearRegression()
reg = reg.fit(X_train, Y_train)

Y_pred = reg.predict(X_test)
```

In [64]:
```
rms = math.sqrt(mean_squared_error(Y_test, Y_pred))
print(rms)
```

```
8.743764087799477
```

In [65]:
```
print(reg.coef_)
print(reg.score(X_test,Y_test))
```

```
[ 5.43155901e+00 -3.16000575e-01 -4.33141896e-03  1.21738116e+00
  2.52174349e+02 -4.95885551e+00]
0.5417678968095335
```

Q6b:

The value of the regression cofficinets doesnot anyhow show the importance of each different features. So, it will be entirely wrong to assume that larger coefficients means more important features.

Q6c:

In [66]:
```
min_max = MinMaxScaler()
X_normalised = min_max.fit_transform(X)
```

In [67]:
```
X_n_train, X_n_test, Y_n_train, Y_n_test = train_test_split( X_normalised,

reg_n = LinearRegression()
reg_n = reg_n.fit(X_n_train, Y_n_train)

Y_n_pred = reg_n.predict(X_n_test)
```

In [68]:
```
rms_norm = math.sqrt(mean_squared_error(Y_n_test, Y_n_pred))
print(rms_norm)
```

```
8.743764087799743
```

In [69]:
```
print(reg_n.coef_)
print(reg_n.score(X_n_test,Y_n_test))
```

```
[  4.97892873 -13.84082518 -28.00105628  12.17381156  20.80942729
  -0.45988426]
0.5417678968095057
```

Q6d:

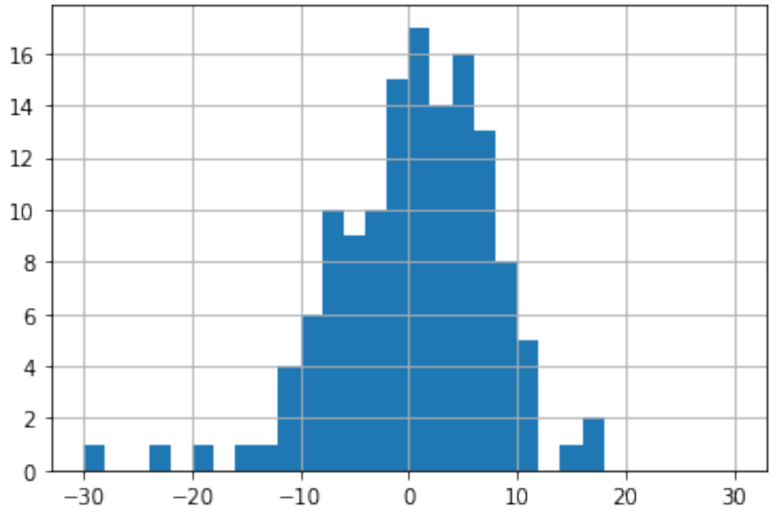
Gaussian

In [70]:

```
residual = Y_n_pred - Y_n_test
residual.hist(bins=30,range = [-30,30])
```

Out[70]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24349de02c8>
```



Q6e:

In [71]:

```
dataset.corr()
```

Out[71]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	longitu
No	1.000000	-0.048634	-0.032808	-0.013573	-0.012699	-0.010110	-0.0110
X1 transaction date	-0.048634	1.000000	0.017542	0.060880	0.009544	0.035016	-0.0410
X2 house age	-0.032808	0.017542	1.000000	0.025622	0.049593	0.054420	-0.0485
X3 distance to the nearest MRT station	-0.013573	0.060880	0.025622	1.000000	-0.602519	-0.591067	-0.8063
X4 number of convenience stores	-0.012699	0.009544	0.049593	-0.602519	1.000000	0.444143	0.4490
X5 latitude	-0.010110	0.035016	0.054420	-0.591067	0.444143	1.000000	0.4129
X6 longitude	-0.011059	-0.041065	-0.048520	-0.806317	0.449099	0.412924	1.0000
Y house price of unit area	-0.028587	0.087529	-0.210567	-0.673613	0.571005	0.546307	0.5232

In [72]:

```
X_optimum = dataset.iloc[:,[3,4,5,6]]

min_max = MinMaxScaler()
X_opt_normalised = min_max.fit_transform(X_optimum)
```

In [73]:

```
X_opt_train, X_opt_test, Y_opt_train, Y_opt_test = train_test_split( X_opt_

reg_opt = LinearRegression()
reg_opt = reg.fit(X_opt_train, Y_opt_train)

Y_opt_pred = reg_opt.predict(X_opt_test)
```

In [74]:

```
rms_opt = math.sqrt(mean_squared_error(Y_opt_test, Y_opt_pred))
print(rms_opt)
```

```
9.000203768125342
```

In [75]:

```
print(reg_opt.coef_)
print(reg_opt.score(X_opt_test,Y_opt_test))
```

```
[-29.39425081  11.30570064  20.71949247  -0.59106937]
0.5144954176958135
```

In [76]:

```
residual = Y_opt_pred - Y_opt_test
residual.hist(bins=30,range = [-30,30])
```

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x2434ae34e88>