Linear Regression(Multiple Variable) - Normal Equation (From Scratch)

Optimation Algorithm: Normal Equation
Characteristics of Normal Equation: No need to choose alpha, no iteration, slow if 'n' is very large, used on smaller data sets

In [4]:
```
# Importing dependencies and dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from mpl_toolkits.mplot3d import Axes3D
```

In [5]:
```
#Creating the dataframe
boston = load_boston()
df = pd.DataFrame(data = boston['data'], columns = boston['feature_name
s'])
df['PRICE'] = boston['target']
df.head()
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTA
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.9
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.1
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.0
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.9
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.3

Feature Scaling

In [6]:
```
#Need to scale feature.  Standard Scaling(use with normally distributed
 data)
X = df.drop(['PRICE'], axis = 1)
X = preprocessing.scale(X)
X = np.c_[np.ones(df.shape[0]),X]
X
```

Out[6]:
```
array([[ 1.        , -0.41978194,  0.28482986, ..., -1.45900038,
         0.44105193, -1.0755623 ],
       [ 1.        , -0.41733926, -0.48772236, ..., -0.30309415,
         0.44105193, -0.49243937],
       [ 1.        , -0.41734159, -0.48772236, ..., -0.30309415,
         0.39642699, -1.2087274 ],
       ...,
       [ 1.        , -0.41344658, -0.48772236, ...,  1.17646583,
         0.44105193, -0.98304761],
       [ 1.        , -0.40776407, -0.48772236, ...,  1.17646583,
         0.4032249 , -0.86530163],
       [ 1.        , -0.41500016, -0.48772236, ...,  1.17646583,
         0.44105193, -0.66905833]])
```

In [7]:
```
y = df['PRICE']
y = preprocessing.scale(df['PRICE'])
```

Splitting Data Set into Train and Test Data Sets

In [8]:
```
#Splitting Data Set into Training and Testing Data Sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .8
0, random_state = 1)
```

In [9]:
```
x_transpose_inverse = np.linalg.inv(np.dot(np.transpose(np.array(X_train
)),np.array(X_train)))

x_transpose_y = np.dot(np.transpose(np.array(X_train)),np.array(y_train
))

new_thetas = np.dot(x_transpose_inverse, x_transpose_y)
new_thetas
```

Out[9]:
```
array([ 0.0325879 , -0.10381083,  0.14188722,  0.11341126,  0.04483217,
       -0.31872622,  0.24021545,  0.12070351, -0.30068075,  0.28545964,
       -0.22073801, -0.29026132,  0.06442864, -0.47125546])
```

Calculating Testing Accuracy

In [10]:

```
def testing(X_test, y_test, thetas_array):
    n = len(X_test)

    SSTO = []   # total sum of squares
    SSR = []    # regression sum of squares
    SSE = []    # error sum of squares
    y_mean = np.mean(y_train)

    #prediction = []

    for i in range(n):
        predict = np.dot(X_test[i],thetas_array)
        #prediction.append(predict)
        SSE.append((predict - y_test[i])**2)
        SSR.append((predict - y_mean)**2)
        SSTO.append((y_test[i] - y_mean)**2)

    print('\naverage error is : ', round(sum(SSE)/len(SSE),4))
    print('\nsum of squares of error (SSE) : ', round(sum(SSE),4))
    print('\nregression sum of squares (SSR) : ', round(sum(SSR),4))
    print('\ntotal sum of squares (SSTO) : ', round(sum(SSTO), 4))
    print('\nThe Coefficient Of Determination R-squared is : ', (round(s
um(SSR)/sum(SSTO),6))*100,'%')
```

In [11]:

```
testing(X_test, y_test, [ 0.0325879 , -0.10381083,  0.14188722,  0.11341
126,  0.04483217,
       -0.31872622,  0.24021545,  0.12070351, -0.30068075,  0.28545964,
       -0.22073801, -0.29026132,  0.06442864, -0.47125546])
```

```
average error is :  0.2746

sum of squares of error (SSE) :  111.2228

regression sum of squares (SSR) :  301.8258

total sum of squares (SSTO) :  401.7879

The Coefficient Of Determination R-squared is :  75.1207 %
```

In [11]:

```
#Run cell to display rotable 3d plot with hyperplane.
from mpl_toolkits.mplot3d import Axes3D
#%matplotlib qt

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Rooms', fontsize = 11)
ax.set_ylabel('Population', fontsize = 11)
ax.set_zlabel('Price', fontsize = 11)

ax.scatter(X_test[:,1], X_test[:,2], y_test, c='C6', marker='o', alpha=
0.6)
#x, y = np.meshgrid(X_test[:,1], X_test[:,2])
z = np.dot(X_test,np.transpose([ 0.06811557,  0.43903259, -0.45381535]))
#ax.plot_wireframe(x,y,z, rcount=200,ccount=200, linewidth = 0.5,color
='C9', alpha=0.5)
ax.plot_trisurf(X_test[:,1], X_test[:,2],z, linewidth=0, antialiased=Fal
se)
```

Out[11]:

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x105552cd0>
```