IE406
Machine Learning
Lab 1

Group 28
201801015 - Shantanu Tyagi
201801076 - Shivani Nandani
201801407 - Pratvi Shah
201801408 - Arkaprabha Banerjee

September 4, 2021

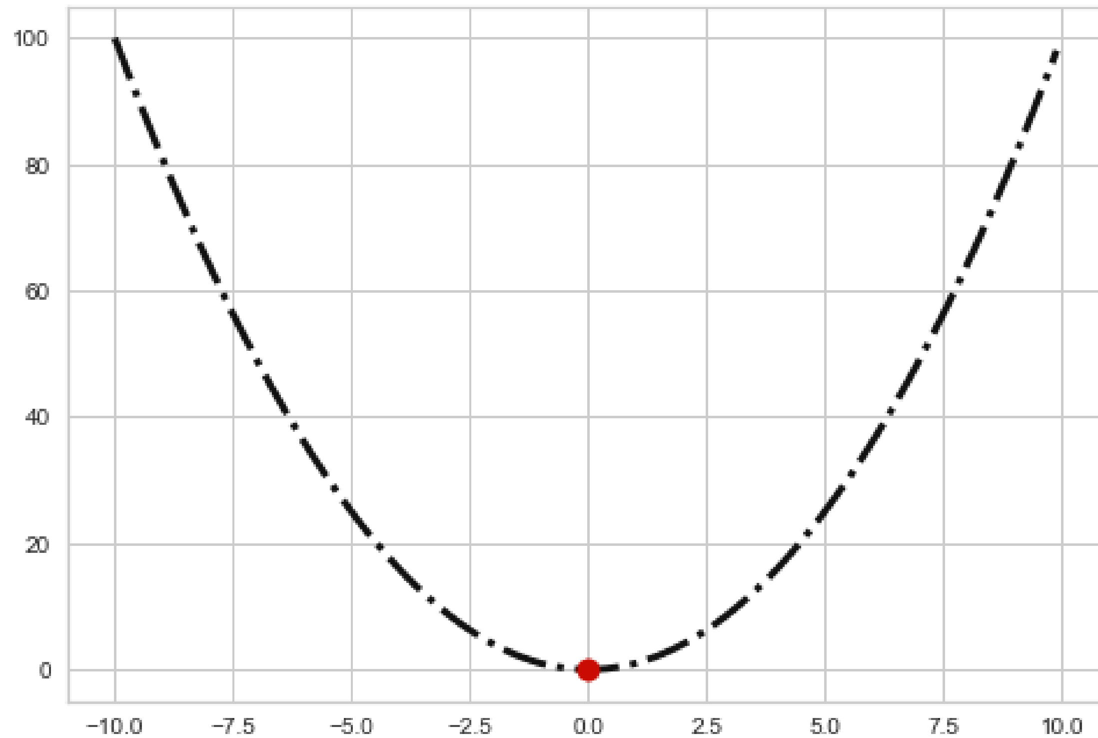
[143]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import normalize
from yellowbrick.regressor import ResidualsPlot
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

1 Question 1

[93]:
```
def L(theta):
return theta*theta

theta = np.arange(-10,10,0.1)
L_theta = []
for t in theta:
L_theta.append(L(t))

min_L_theta = min(L_theta)
min_loc = np.argwhere(L_theta == min_L_theta)[0,0]

print('Minimum value of L() is', round(min_L_theta), ' at  = ',␣
↪round(min_L_theta))
plt.figure()
plt.plot(theta, L_theta, 'k-.', linewidth=3)
plt.plot(theta[min_loc], min_L_theta, 'ro', markersize=10)
```

```
plt.show()
```

```
Minimum value of L() is 0.0  at  =  0.0
```

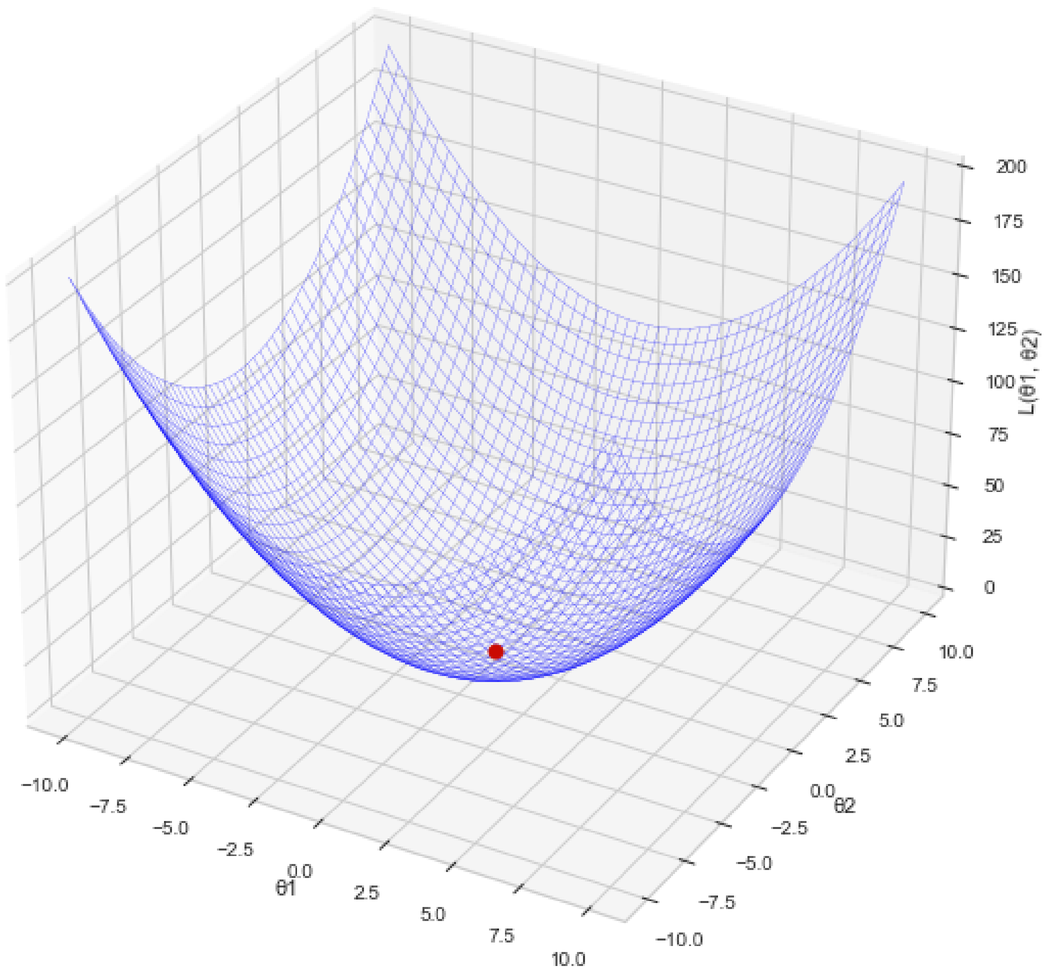


2 Question 2

[108]:
```
def L(theta1, theta2):
return theta1*theta1 + theta2*theta2

theta1 = np.arange(-10,10,0.1)
theta2 = np.arange(-10,10,0.1)
L_theta1_theta2 = np.zeros((len(theta1),len(theta2)))
for i in range(len(theta1)):
for j in range(len(theta2)):
L_theta1_theta2[i][j] = L(theta1[i], theta2[j])

min_L_theta1_theta2 = np.min(L_theta1_theta2)
print('Minimum value of L(1, 2) is', round(min_L_theta1_theta2))
min_loc = np.argwhere(L_theta1_theta2 == min_L_theta1_theta2)

x, y = np.meshgrid(theta1, theta2)
z = L(x, y)
```

```
fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')
ax.plot_wireframe(x, y, z, color='blue', linewidth=0.2)
ax.plot(theta1[min_loc[0,0]], theta2[min_loc[0,1]], z[min_loc[0,0],min_loc[0,1]], 'ro', markersize=8)
ax.set_xlabel('1')
ax.set_ylabel('2')
ax.set_zlabel('L(1, 2)')
plt.show()
```

```
Minimum value of L(1, 2) is 0.0
```



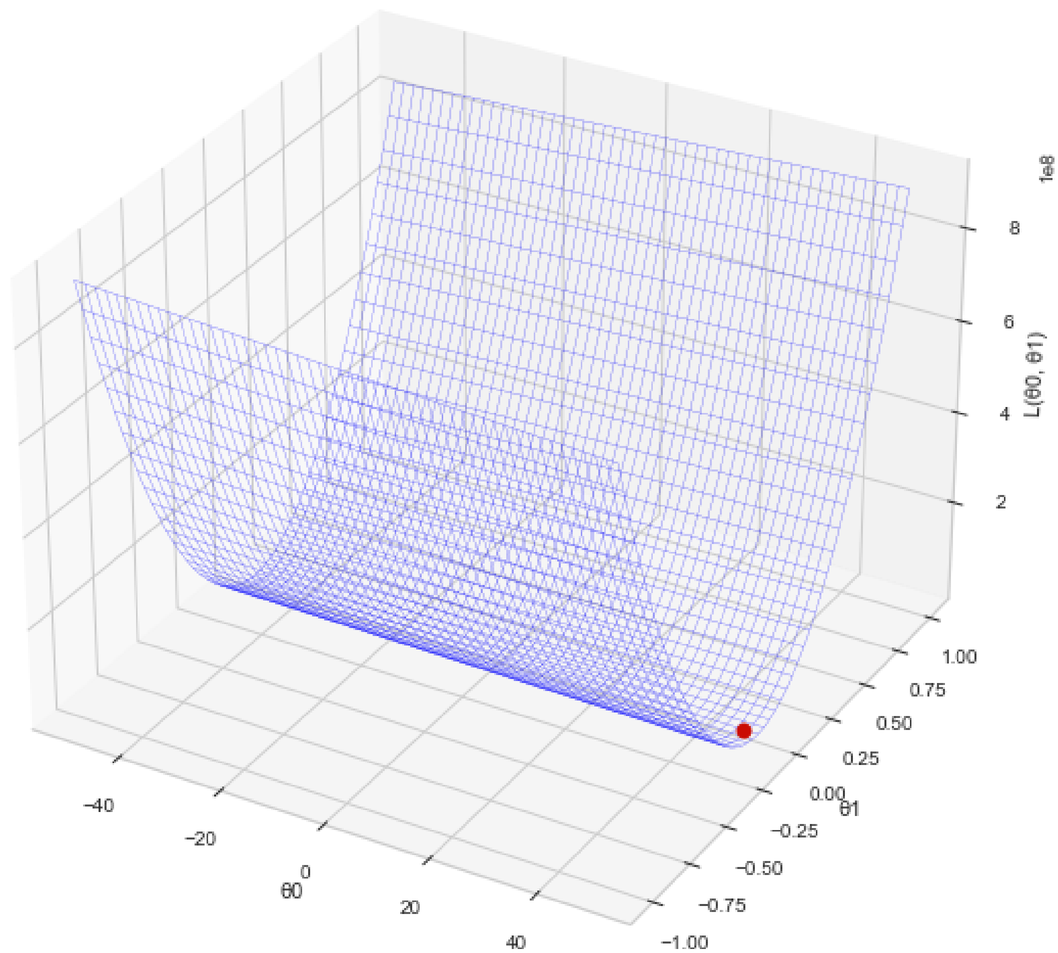
The minimum value of L is obtained at $ θ

2.1 Part A

[117]:

```
def L(x, y, theta0, theta1):
return np.sum(np.square(y - (theta0 + (theta1*x))))

theta0 = np.arange(-50,50,0.05)
theta1 = np.arange(-1,1,0.001)
L_theta0_theta1 = np.zeros((len(theta1),len(theta0)))
data = pd.read_excel('data.xlsx')
data_x = np.reshape(np.array(data.x), (-1, 1))
data_y = np.reshape(np.array(data.y), (-1, 1))

for i in range(len(theta1)):
for j in range(len(theta0)):
L_theta0_theta1[i][j] = L(data_x, data_y, theta0[j], theta1[i])

min_L_theta0_theta1 = np.min(L_theta0_theta1)
print('Minimum value of L(0, 1) is', min_L_theta0_theta1)

min_loc = np.argwhere(L_theta0_theta1 == min_L_theta0_theta1)
print('0 and 1 values for minimum value of L(0, 1) are ', theta0[min_loc[0,1]], theta1[min_loc[0,0]], ' respectively.')

x, y = np.meshgrid(theta0, theta1)
z = L_theta0_theta1

fig = plt.figure(figsize=(10, 10))
ax = plt.axes(projection='3d')
ax.plot_wireframe(x, y, z, color='blue', linewidth=0.2)
ax.plot(theta0[min_loc[0,1]], theta1[min_loc[0,0]], z[min_loc[0,0],min_loc[0,1]], 'ro', markersize = 8)
ax.set_xlabel('0')
ax.set_ylabel('1')
ax.set_zlabel('L(0, 1)')
plt.show()
```

```
Minimum value of L(0, 1) is 1595.086384000078
0 and 1 values for minimum value of L(0, 1) are  47.39999999999446
-0.007999999999999119  respectively.
```



2.2 Part B

[118]:

```
#Question 3b
temp = np.copy(data_x)
x = np.ones((np.size(temp,0),2))
x[:,1] = temp[:,0]

print('determinant of x.T*x = ',np.linalg.det(np.matmul(x.T,x)))

theta = np.matmul(np.linalg.inv(np.matmul(x.T,x)),np.matmul(x.T,data_y))
print('For minima, theta0 = ',theta[0,0],' and theta1 = ',theta[1,0])
```

```
determinant of x.T*x =  5618077959.999997
For minima, theta0 =  49.23762989433493  and theta1 =  -0.008611934783475328
```

Using $ θ

[123]:
```
#Question 4
LS = np.sum(np.square(data_y - (theta[0,0] + (theta[1,0]*temp))))
print('Value of L using theta values from LS method(Pseudo Inverse) =␣
↪',LS)
print()
for i in range(10):
L_other_value = np.sum(np.square(data_y - (np.random.randint(theta0.
↪shape[0]) - (np.random.randint(theta1.shape[0])*temp))))
print('Value of L for some other theta = ',L_other_value)
```

```
Value of L using theta values from LS method(Pseudo Inverse) =
1572.6503668922921

Value of L for some other theta =  2008722638480283.8
Value of L for some other theta =  1119326591644803.8
Value of L for some other theta =  256774493245402.75
Value of L for some other theta =  227493761746521.75
Value of L for some other theta =  2458405151084183.0
Value of L for some other theta =  201287534533077.75
Value of L for some other theta =  148445180762662.75
Value of L for some other theta =  2027591949333187.8
Value of L for some other theta =  795214078513074.8
Value of L for some other theta =  21985849704444.75
```

We see that the value L for the $ θ

2.3 Part A

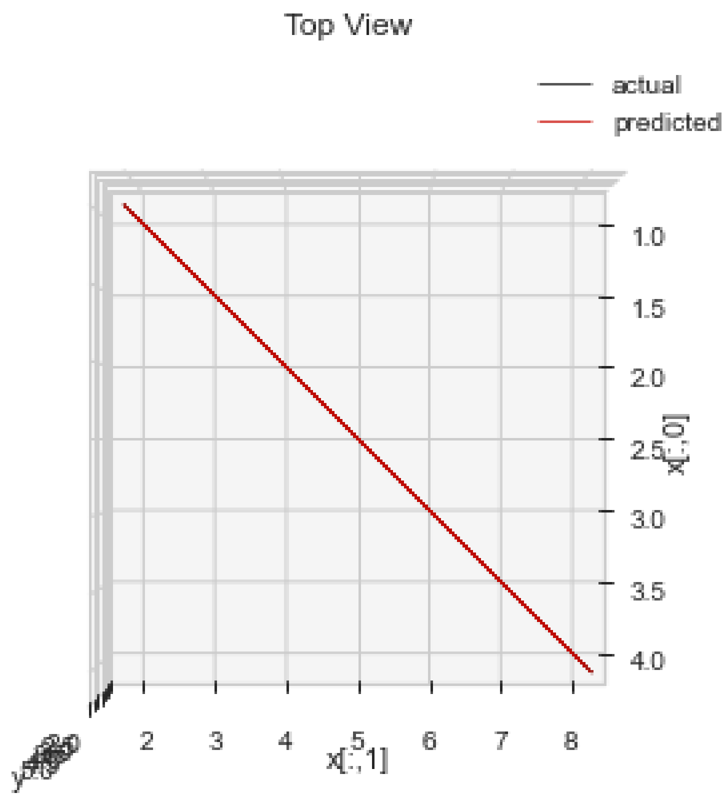
[127]:
```
x = np.array([
[1, 2],
[2, 4],
[3, 6],
[4, 8]
])
y = np.array([
[2],
[3],
[4],
[5]
])

print(' X Shape ->', x.shape, '\n', 'Y Shape ->', y.shape)
```

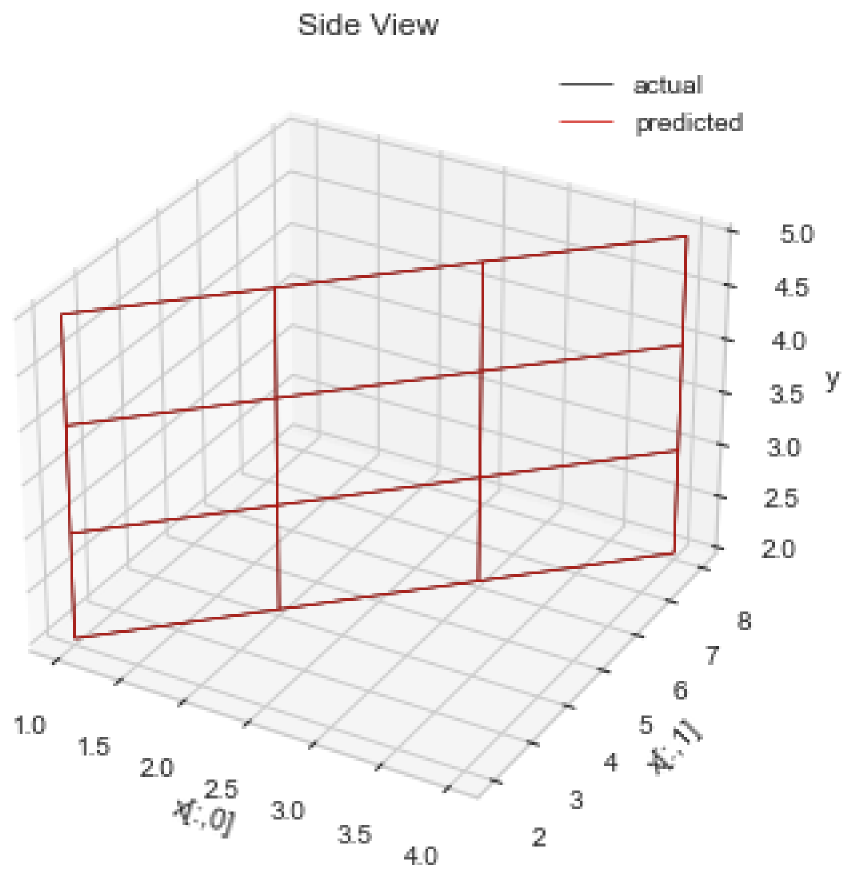
```
X Shape -> (4, 2)
Y Shape -> (4, 1)
```

[128]:
```
reg = LinearRegression().fit(x, y)
print('Score: ', reg.score(x, y))
print('Coefficient: ', reg.coef_)
print('Intercept: ', reg.intercept_)
```

```
Score:  1.0
Coefficient:  [[0.2 0.4]]
Intercept:  [1.]
```

[130]:
```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(x[:,0], x[:,1], y, color='k', linewidth=0.7,␣
↪label='actual')
ax.plot_wireframe(x[:,0], x[:,1], y_pred, color='r', linewidth=0.7,␣
↪label='predicted')
ax.set_xlabel('x[:,0]')
ax.set_ylabel('x[:,1]')
ax.set_zlabel('y')
ax.view_init(90, 0)
plt.legend()
plt.title('Top View')
plt.show()

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(x[:,0], x[:,1], y, color='k', linewidth=0.7,␣
↪label='actual')
ax.plot_wireframe(x[:,0], x[:,1], y_pred, color='r', linewidth=0.7,␣
↪label='predicted')
ax.set_xlabel('x[:,0]')
ax.set_ylabel('x[:,1]')
ax.set_zlabel('y')
plt.legend()
plt.title('Side View')
plt.show()
```


Top View
actual
predicted
1.0
1.5
2.0
2.5
3.0
3.5
4.0
x[:,0]
2
3
4
5
6
7
8
x[:,1]



2.4 Part B

Since (X^T X) is non-invertible, therefore we obtain an error when solving via normal equations method

[136]:
```
def normal(X,y):
return np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),y)

normal(x,y)
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
<ipython-input-136-ccba76942472> in <module>
      2     return np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),y)
      3
----> 4 normal(x,y)

<ipython-input-136-ccba76942472> in normal(X, y)
      1 def normal(X,y):
```

```
----> 2     return np.dot(np.dot(np.linalg.inv(np.dot(X.T,X)),X.T),y)
      3
      4 normal(x,y)

<__array_function__ internals> in inv(*args, **kwargs)

c:
↪\users\geeksa67\appdata\local\programs\python\python38\lib\site-packages\numpy linalg
        \linalg.py in inv(a)
    545     signature = 'D->D' if isComplexType(t)
else 'd->d'
    546     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 547     ainv = _umath_linalg.inv(a,
signature=signature, extobj=extobj)
    548     return wrap(ainv.astype(result_t, copy=
False))
    549

c:
↪\users\geeksa67\appdata\local\programs\python\python38\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
```

[142]:
```
x_trans = np.transpose(x)
XX = np.dot(x_trans, x)
XY = np.dot(x_trans, y)

print('Checking invertability')
print('----------------------')

print('X\'X\n', XX)
print('Rank of X\'X -> ', np.linalg.matrix_rank(XX))

print('----------------------')
print('Since the matrix is rank deficient, thus matrix X\'X is not␣
↪invertible.')
```

```
Checking invertability
----------------------
X'X
```

```
[[ 30  60]
 [ 60 120]]
Rank of X'X ->  1
----------------------
Since the matrix is rank deficient, thus matrix X'X is not invertible.
```

The Inverse of ATA doesn't exist in the above questions hence the approach of normal equations wont work here. This is primarily because ATA is a rank-deficient matrix hence the inverse doesn't exist.

The LinearRegression function is a wrapper function for scipy.linalg.lstsq and other associated methods. For rank deficient matrices where the inverse of a matrix does not exist by traditional methods (normal equations), these libraries compute a least-squares solution to equation Ax = b by Computing a vector x such that the 2-norm |b - A x| is minimized. This is achieved by using LAPACK (Linear Algebra Package by NETLIB) which in turn uses SVD or QR (Orthogonal Factorization) based techniques to calculate the inverse.

3 Question 6

3.1 Part A

[386]:
```
real_estate_data = pd.read_excel('real_estate_valuation_dataset.xlsx')
real_estate_data
```

[386]:
```
      No  X1 transaction date  X2 house age  \
0      1          2012.916667          32.0
1      2          2012.916667          19.5
2      3          2013.583333          13.3
3      4          2013.500000          13.3
4      5          2012.833333           5.0
..   ...                  ...           ...
409  410          2013.000000          13.7
410  411          2012.666667           5.6
411  412          2013.250000          18.8
412  413          2013.000000           8.1
413  414          2013.500000           6.5

     X3 distance to the nearest MRT station  X4 number of convenience stores ␣
↪\
0                                  84.87882                                ␣
↪ 10
1                                 306.59470                                ␣
↪  9
2                                 561.98450                                ␣
↪  5
3                                 561.98450                                ␣
↪  5
```

```
    4                                 390.56840                          ␣
↪ 5
    ..                                      ...                          ␣
↪...
    409                              4082.01500                          ␣
↪ 0
    410                                90.45606                          ␣
↪ 9
    411                               390.96960                          ␣
↪ 7
    412                               104.81010                          ␣
↪ 5
    413                                90.45606                          ␣
↪ 9

    X5 latitude  X6 longitude  Y house price of unit area
    0       24.98298     121.54024                        37.9
    1       24.98034     121.53951                        42.2
    2       24.98746     121.54391                        47.3
    3       24.98746     121.54391                        54.8
    4       24.97937     121.54245                        43.1
    ..           ...           ...                         ...
    409     24.94155     121.50381                        15.4
    410     24.97433     121.54310                        50.0
    411     24.97923     121.53986                        40.6
    412     24.96674     121.54067                        52.5
    413     24.97433     121.54310                        63.9

        [414 rows x 8 columns]
```

[387]:

```
x1 = np.reshape(np.array(real_estate_data['X1 transaction date']), (-1,␣
↪1))
x2 = np.reshape(np.array(real_estate_data['X2 house age']), (-1, 1))
x3 = np.reshape(np.array(real_estate_data['X3 distance to the nearest␣
↪MRT station']), (-1, 1))
x4 = np.reshape(np.array(real_estate_data['X4 number of convenience␣
↪stores']), (-1, 1))
x5 = np.reshape(np.array(real_estate_data['X5 latitude']), (-1, 1))
x6 = np.reshape(np.array(real_estate_data['X6 longitude']), (-1, 1))
X = np.hstack((x1, x2, x3, x4, x5, x6))
y = np.reshape(np.array(real_estate_data['Y house price of unit area']),␣
↪(-1, 1))
```

[388]:

```
reg = LinearRegression().fit(X, y)
print('Score: ', reg.score(X, y))
print('Coefficient: ', reg.coef_)
```

```
print('Intercept: ', reg.intercept_)

y_pred1 = reg.predict(X)

print('RMSE ->', np.sqrt(mean_squared_error(y, y_pred1)))
```

```
Score:  0.5823850447850787
Coefficient:  [[ 5.14901721e+00 -2.69696735e-01 -4.48750825e-03  1.13332498e+00
   2.25470143e+02 -1.24290612e+01]]
Intercept:  [-14441.98271918]
RMSE -> 8.782312975361124
```

3.2 Part B

The sign of a regression coefficient tells you whether there is a positive or negative correlation between each independent variable and the dependent variable. A positive coefficient indicates that as the value of the independent variable increases, the mean of the dependent variable also tends to increase. A negative coefficient suggests that as the independent variable increases, the dependent variable tends to decrease.

The coefficient value signifies how much the mean of the dependent variable changes given a one-unit shift in the independent variable while holding other variables in the model constant. This property of holding the other variables constant is crucial because it allows you to assess the effect of each variable in isolation from the others.

We can fit a LinearRegression model on the regression dataset and retrieve the coeff_ property that contains the coefficients found for each input variable. Since the X values are not standardized, we cannot comment on the importance of the feature score.

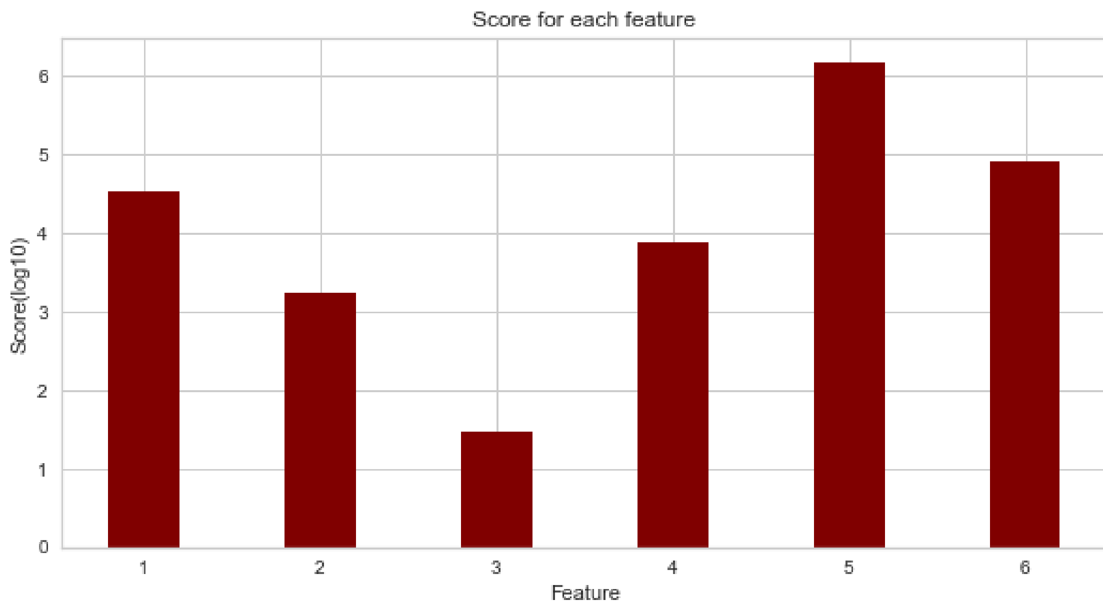
3.3 Part C

[389]:
```
X_norm = (X - X.min())/(X.max()-X.min())
reg = LinearRegression().fit(X_norm, y)
print('Score: ', reg.score(X_norm, y))
print('Coefficient: ', reg.coef_)
print('Intercept: ', reg.intercept_)

y_pred2 = reg.predict(X_norm)

print('RMSE ->', np.sqrt(mean_squared_error(y, y_pred2)))
```

```
Score:  0.5823850447850758
Coefficient:  [[ 3.34069318e+04 -1.74979808e+03 -2.91150478e+01  7.35303628e+03
   1.46285502e+06 -8.06400099e+04]]
Intercept:  [-14441.98271918]
RMSE -> 8.782312975361156
```

[390]:
```
fig = plt.figure(figsize = (10, 5))
```

```
# creating the bar plot
plt.bar([1,2,3,4,5,6], np.log10(abs(reg.coef_[0])), color ='maroon',
width = 0.4)

plt.xlabel("Feature")
plt.ylabel("Score(log10)")
plt.title("Score for each feature")
#plt.yscale('log')
plt.show()
```

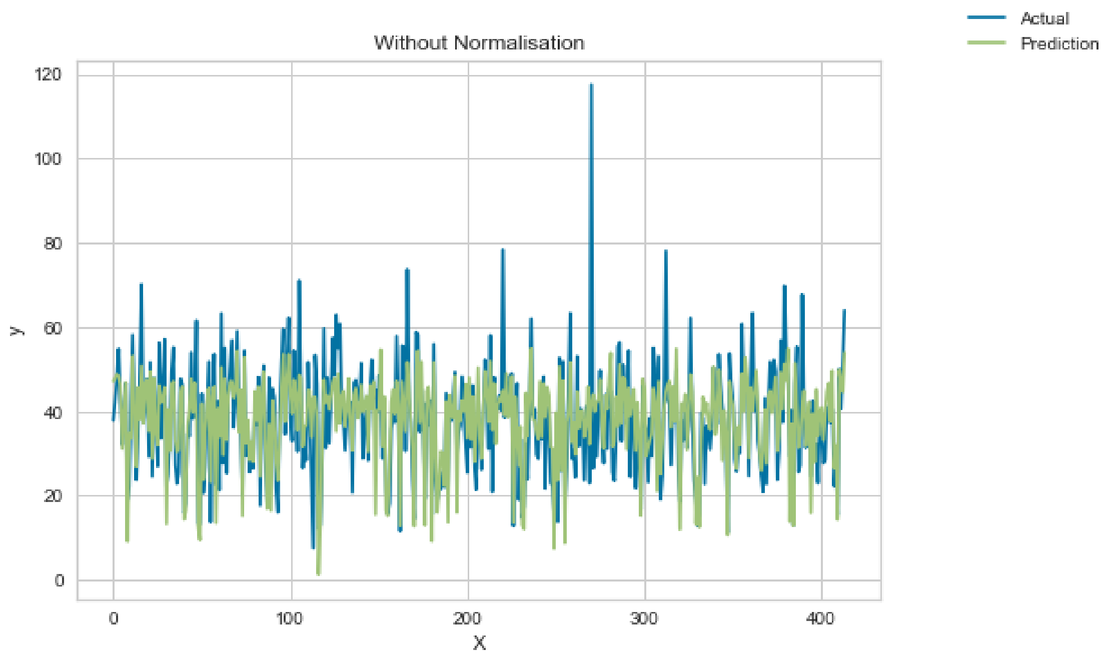


The sign of a regression coefficient tells if there is a positive of negative correlation between each independent variable and dependent variable, i.e., if a feature is positively correlated, increase in its value increases the mean of the dependent and vice-versa. The value of the correlation coefficient gives and idea of how much a unit shift in the value of an independent variable affects the dependent variable (assuming all other variables to be constant). This gives an idea of the effect of each variable independent of the others.

A true sense of this can be achieved only after the normalization has been performed. Thus, once normalized, the coefficients can be used as a measure of the importance of each feature w.r.t. the dependent variable. Else, as mentioned above, the coefficients are not a good measure of the importance of the features. As seen in the graph, the 5th and 6th features are the most important ones, i.e., the latitude and longitude of the house matters.

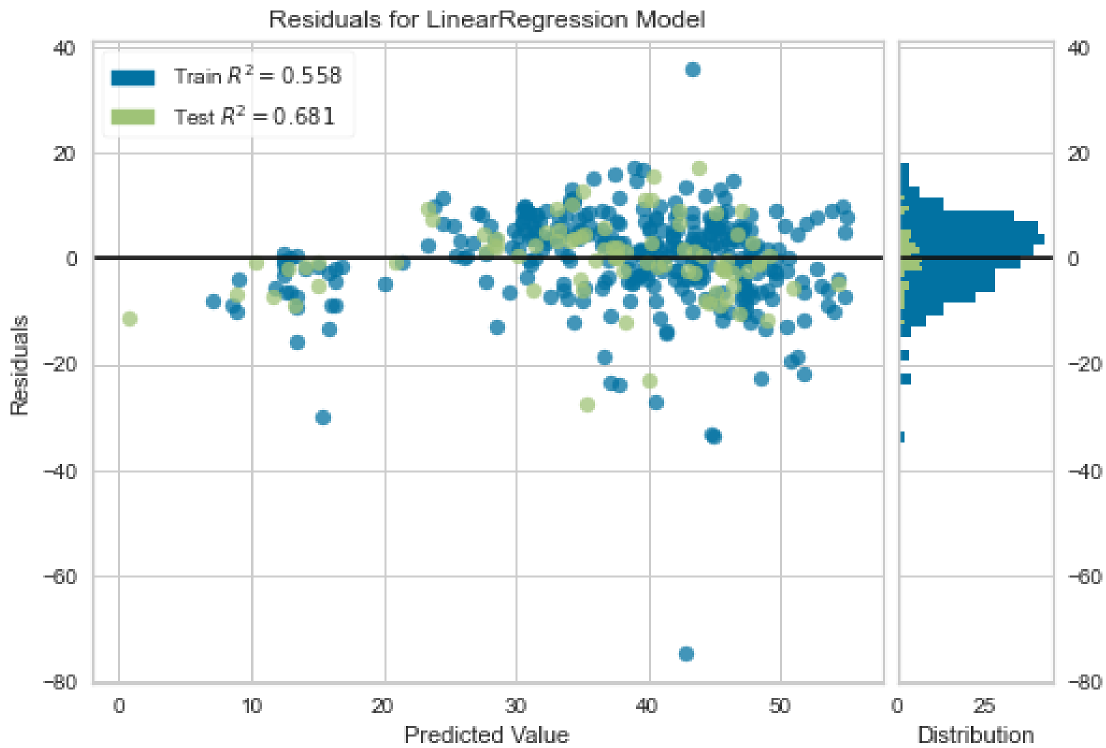
[391]:
```
#plt.plot(y)
plt.plot(y)
plt.plot(y_pred1)
plt.legend(["Actual","Prediction"],loc=[1.1,1])
```

```
    plt.xlabel("X")
    plt.ylabel("y")
    plt.title("Without Normalisation")
    plt.show()
```



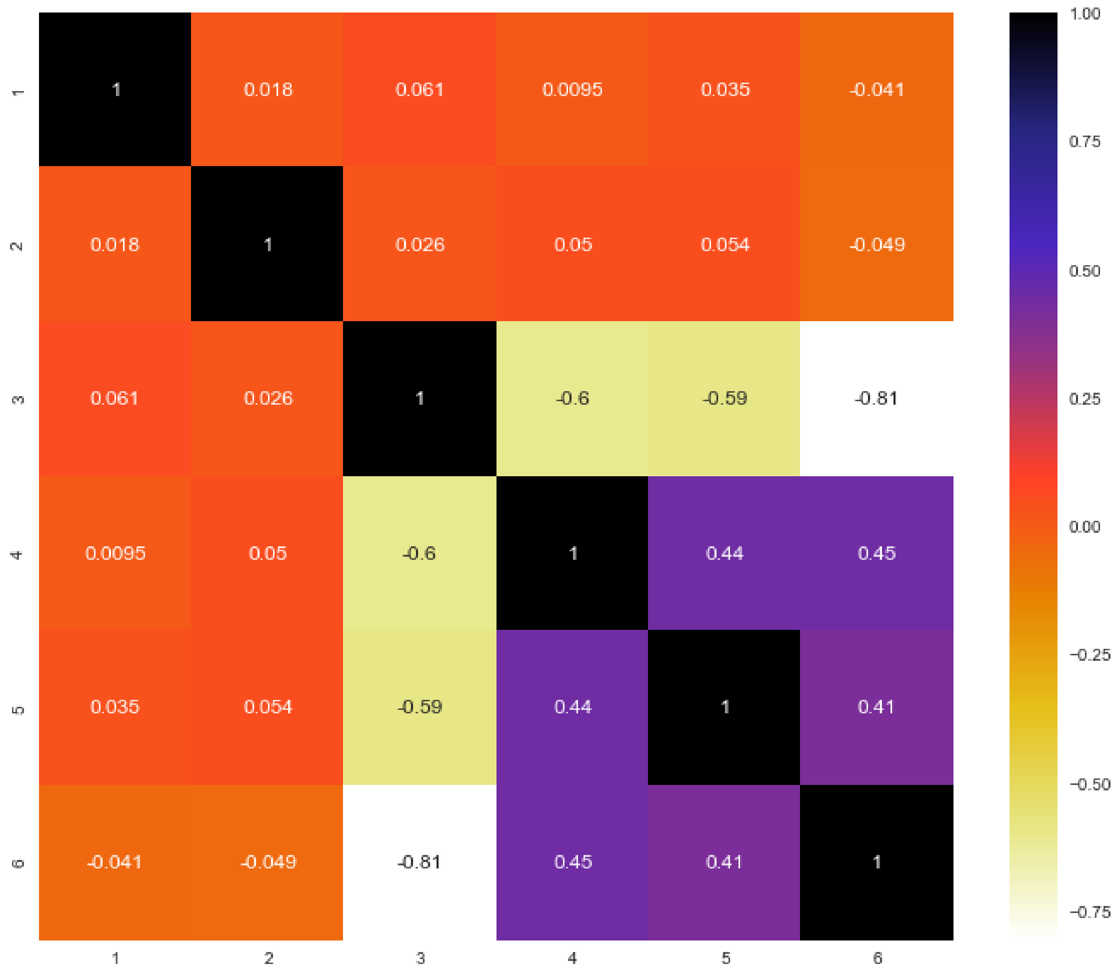
3.4 Part D

[392]:
```
    # Create the train and test data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

    # Instantiate the linear model and visualizer
    model = LinearRegression()
    visualizer = ResidualsPlot(model)

    visualizer.fit(X_train, y_train)  # Fit the training data to the␣
↪visualizer
    visualizer.score(X_test, y_test)  # Evaluate the model on the test data
    visualizer.show()                 # Finalize and render the figure
    plt.show()
```



3.5 Part E

[400]:
```
import seaborn as sns
plt.figure(figsize=(12,10))
X_norm = pd.DataFrame(X_norm)
X_norm.columns = [1,2,3,4,5,6]
cor = X_norm.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.CMRmap_r)
plt.show()
```

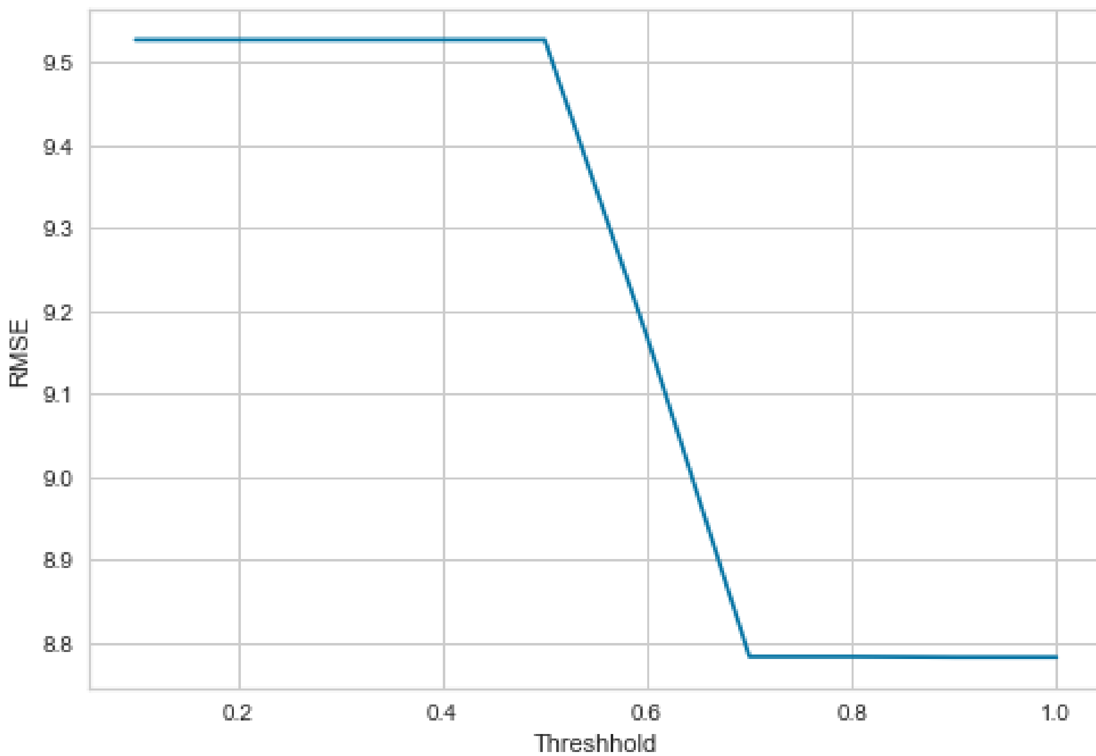


[401]:
```
# with the following function we can select highly correlated features
# it will remove the first feature that is correlated with anything␣
↪other feature
def correlation(dataset, threshold):
col_corr = set()  # Set of all the names of correlated columns
corr_matrix = dataset.corr()
for i in range(len(corr_matrix.columns)):
for j in range(i):
if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in␣
↪absolute coeff value
colname = corr_matrix.columns[i]  # getting the name of column
col_corr.add(colname)
return col_corr
```

[409]:
```
rmse = [0 for i in range(10)]
for i in range(1,11):
```

```
    corr_features = correlation(X_norm,i/10)
    if len(set(corr_features)) != 6:
    temp = X_norm.drop(corr_features,axis=1)
    reg = LinearRegression().fit(np.array(temp), y)
    y_pred2 = reg.predict(np.array(temp))
    rmse[i-1] = np.sqrt(mean_squared_error(y, y_pred2))
```

[424]:
```
plt.plot([i/10 for i in range(1,11)],rmse)
plt.xlabel("Threshhold")
plt.ylabel("RMSE")
plt.show()
```



[416]:
```
corr_features = correlation(X_norm,0.7)
corr_features
```

[416]: {6}

One feature is removed giving the no of features retained = 5 out of 6 features retained are columns = {1, 2, 3, 4, 5} with 6th column getting removed