26. Regularization Technique (Practical)

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [3]:
```
dataset = pd.read_csv(r'Data/housing.csv')
dataset.head(3)
```

Out[3]:

	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating
0	7420	4	2	3	yes	no	no	no
1	8960	4	4	4	yes	no	no	no
2	9960	3	2	2	yes	no	yes	no

In [4]:
```
dataset.isnull().sum()
```

Out[4]:
```
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
price             0
dtype: int64
```

Encoding the Data into Numerical Form

In [5]:
```
en_data = dataset[['mainroad','guestroom', 'basement', 'hotwaterheating', 'aircondi
en_data.head(3)
```

Out[5]:

	mainroad	guestroom	basement	hotwaterheating	airconditioning	prefarea	furnishing
0	yes	no	no	no	yes	yes	fur
1	yes	no	no	no	yes	no	fur
2	yes	no	yes	no	no	yes	semi-fur

In [6]:
```
pd.get_dummies(en_data)
```

Out[6]:

	mainroad_no	mainroad_yes	guestroom_no	guestroom_yes	basement_no	basement_y
0	0	1	1	0	1	
1	0	1	1	0	1	
2	0	1	1	0	0	
3	0	1	1	0	0	
4	0	1	0	1	0	
...	...	...	...	...	...	
540	0	1	1	0	0	
541	1	0	1	0	1	
542	0	1	1	0	1	
543	1	0	1	0	1	
544	0	1	1	0	1	

545 rows × 15 columns

In [7]:
```
pd.get_dummies(en_data).info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 15 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   mainroad_no                      545 non-null    uint8
 1   mainroad_yes                     545 non-null    uint8
 2   guestroom_no                     545 non-null    uint8
 3   guestroom_yes                    545 non-null    uint8
 4   basement_no                      545 non-null    uint8
 5   basement_yes                     545 non-null    uint8
 6   hotwaterheating_no               545 non-null    uint8
 7   hotwaterheating_yes              545 non-null    uint8
 8   airconditioning_no               545 non-null    uint8
 9   airconditioning_yes              545 non-null    uint8
 10  prefarea_no                      545 non-null    uint8
 11  prefarea_yes                     545 non-null    uint8
 12  furnishingstatus_furnished       545 non-null    uint8
 13  furnishingstatus_semi-furnished  545 non-null    uint8
 14  furnishingstatus_unfurnished     545 non-null    uint8
dtypes: uint8(15)
memory usage: 8.1 KB
```

In [8]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [9]:
```
ohe = OneHotEncoder()
ohe.fit_transform(en_data)
```

Out[9]:
```
<545x15 sparse matrix of type '<class 'numpy.float64'>'
        with 3815 stored elements in Compressed Sparse Row format>
```

In [10]:
```
ohe=OneHotEncoder()
arr = ohe.fit_transform(en_data).toarray()
arr
```

Out[10]:
```
array([[0., 1., 1., ..., 1., 0., 0.],
       [0., 1., 1., ..., 1., 0., 0.],
       [0., 1., 1., ..., 0., 1., 0.],
       ...,
       [0., 1., 1., ..., 0., 0., 1.],
       [1., 0., 1., ..., 1., 0., 0.],
       [0., 1., 1., ..., 0., 0., 1.]])
```

In [11]:
```
pd.DataFrame(arr, columns=['mainroad_Yes','mainroad_No','guestroom_Yes', 'guestroom
```

Out[11]:

	mainroad_Yes	mainroad_No	guestroom_Yes	guestroom_No	basement_Yes	basement
0	0.0	1.0	1.0	0.0	1.0	
1	0.0	1.0	1.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	0.0	
3	0.0	1.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	
...	...	...	...	...	...	
540	0.0	1.0	1.0	0.0	0.0	
541	1.0	0.0	1.0	0.0	1.0	
542	0.0	1.0	1.0	0.0	1.0	
543	1.0	0.0	1.0	0.0	1.0	
544	0.0	1.0	1.0	0.0	1.0	

545 rows × 15 columns

In [12]:
```
ohe = OneHotEncoder(drop='first')
ar = ohe.fit_transform(en_data).toarray()
ar
```

Out[12]:
```
array([[1., 0., 0., ..., 1., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 1., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 1.]])
```

In [13]:
```
pd.DataFrame(arr, columns=['mainroad_Yes','mainroad_No','guestroom_Yes', 'guestroom
```

Out[13]:

	mainroad_Yes	mainroad_No	guestroom_Yes	guestroom_No	basement_Yes	basement
0	0.0	1.0	1.0	0.0	1.0	
1	0.0	1.0	1.0	0.0	1.0	
2	0.0	1.0	1.0	0.0	0.0	
3	0.0	1.0	1.0	0.0	0.0	
4	0.0	1.0	0.0	1.0	0.0	
...	...	...	...	...	...	
540	0.0	1.0	1.0	0.0	0.0	
541	1.0	0.0	1.0	0.0	1.0	
542	0.0	1.0	1.0	0.0	1.0	
543	1.0	0.0	1.0	0.0	1.0	
544	0.0	1.0	1.0	0.0	1.0	

545 rows × 15 columns

In [14]:
```
ohe = OneHotEncoder(drop='first')
ar = ohe.fit_transform(en_data).toarray()
ar
```

Out[14]:
```
array([[1., 0., 0., ..., 1., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 1., ..., 1., 1., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 1.]])
```

In [15]:
```
ar.shape
```

Out[15]:
```
(545, 8)
```

In [16]:
```
encoded_data = pd.DataFrame(ar, columns=['mainroad_Yes','guestroom_Yes', 'basement_
```

In [17]:
```
encoded_data
```

Out[17]:

	mainroad_Yes	guestroom_Yes	basement_Yes	hotwaterheating_Yes	airconditioning_Yes
0	1.0	0.0	0.0	0.0	1.0
1	1.0	0.0	0.0	0.0	1.0
2	1.0	0.0	1.0	0.0	0.0
3	1.0	0.0	1.0	0.0	1.0
4	1.0	1.0	1.0	0.0	1.0
...	...	...	...	...	...
540	1.0	0.0	1.0	0.0	0.0
541	0.0	0.0	0.0	0.0	0.0
542	1.0	0.0	0.0	0.0	0.0
543	0.0	0.0	0.0	0.0	0.0
544	1.0	0.0	0.0	0.0	0.0

545 rows × 8 columns

In [18]:
```
encoded_data.to_csv(r'Data/encoded_data_file.csv', index=False)
```

Loading the Encoded Data for Applying Regularization Techniques

In [19]:
```
dataset = pd.read_csv('Data/housing_2.csv')
dataset
```

Out[19]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes	baseme
0	7420	4	2	3	2	1	0	
1	8960	4	4	4	3	1	0	
2	9960	3	2	2	2	1	0	
3	7500	4	2	2	3	1	0	
4	7420	4	1	2	2	1	1	
...	...	...	...	...	...	...	...	
540	3000	2	1	1	2	1	0	
541	2400	3	1	1	0	0	0	
542	3620	2	1	1	0	1	0	
543	2910	3	1	1	0	0	0	
544	3850	3	1	2	0	1	0	

545 rows × 14 columns

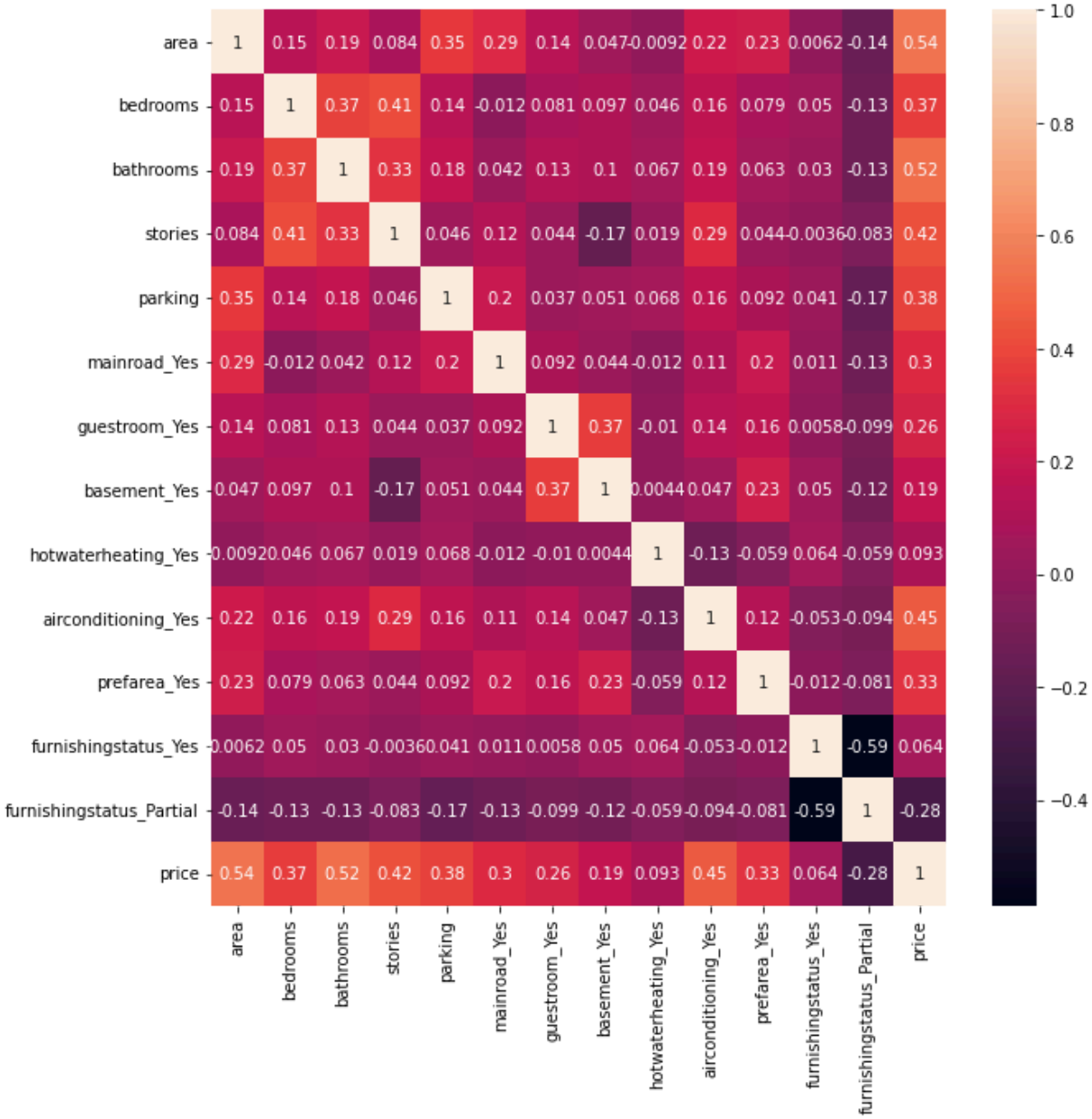
In [20]:
```
dataset = pd.read_csv(r'Data/housing_2.csv')
dataset.head(3)
```

Out[20]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes	basement_
0	7420	4	2	3	2	1	0	
1	8960	4	4	4	3	1	0	
2	9960	3	2	2	2	1	0	

Check Correlation in Data

In [21]:
```
plt.figure(figsize=(10,10))
sns.heatmap(data=dataset.corr(), annot=True)
plt.show()
```



In [22]:
```
x = dataset.iloc[:,:-1]
x
```

Out[22]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes	baseme
0	7420	4	2	3	2	1	0	
1	8960	4	4	4	3	1	0	
2	9960	3	2	2	2	1	0	
3	7500	4	2	2	3	1	0	
4	7420	4	1	2	2	1	1	
...	...	...	...	...	...	...	...	
540	3000	2	1	1	2	1	0	
541	2400	3	1	1	0	0	0	
542	3620	2	1	1	0	1	0	
543	2910	3	1	1	0	0	0	
544	3850	3	1	2	0	1	0	

545 rows × 13 columns

In [23]:
```
y=dataset['price']
y
```

Out[23]:
```
0      13300000
1      12250000
2      12250000
3      12215000
4      11410000
         ...
540     1820000
541     1767150
542     1750000
543     1750000
544     1750000
Name: price, Length: 545, dtype: int64
```

Perform Scaling on Data

In [28]:
```
sc = StandardScaler()
sc.fit(x)
sc.transform(x)
```

Out[28]:
```
array([[ 1.04672629,  1.40341936,  1.42181174, ...,  1.80494113,
        -0.84488844, -0.6964292 ],
       [ 1.75700953,  1.40341936,  5.40580863, ..., -0.55403469,
        -0.84488844, -0.6964292 ],
       [ 2.21823241,  0.04727831,  1.42181174, ...,  1.80494113,
         1.18358821, -0.6964292 ],
       ...,
       [-0.70592066, -1.30886273, -0.57018671, ..., -0.55403469,
        -0.84488844,  1.43589615],
       [-1.03338891,  0.04727831, -0.57018671, ..., -0.55403469,
        -0.84488844, -0.6964292 ],
       [-0.5998394 ,  0.04727831, -0.57018671, ..., -0.55403469,
        -0.84488844,  1.43589615]])
```

In [29]:
```
# transform data to csv sheet
x = pd.DataFrame(sc.transform(x), columns=x.columns)
x
```

Out[29]:

	area	bedrooms	bathrooms	stories	parking	mainroad_Yes	guestroom_Yes
0	1.046726	1.403419	1.421812	1.378217	1.517692	0.405623	-0.465315
1	1.757010	1.403419	5.405809	2.532024	2.679409	0.405623	-0.465315
2	2.218232	0.047278	1.421812	0.224410	1.517692	0.405623	-0.465315
3	1.083624	1.403419	1.421812	0.224410	2.679409	0.405623	-0.465315
4	1.046726	1.403419	-0.570187	0.224410	1.517692	0.405623	2.149083
...	...	...	...	...	...	...	...
540	-0.991879	-1.308863	-0.570187	-0.929397	1.517692	0.405623	-0.465315
541	-1.268613	0.047278	-0.570187	-0.929397	-0.805741	-2.465344	-0.465315
542	-0.705921	-1.308863	-0.570187	-0.929397	-0.805741	0.405623	-0.465315
543	-1.033389	0.047278	-0.570187	-0.929397	-0.805741	-2.465344	-0.465315
544	-0.599839	0.047278	-0.570187	0.224410	-0.805741	0.405623	-0.465315

545 rows × 13 columns

Split data into train and test

In [30]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state
```

26.1 Model by Linear Regression

In [31]:
```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
```

In [32]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[32]:
```
LinearRegression()
```

Test Model

In [35]:
```
lr.score(x_test, y_test)*100
```

Out[35]:
```
65.29242642153177
```

Graphical representation of constant and coefficient

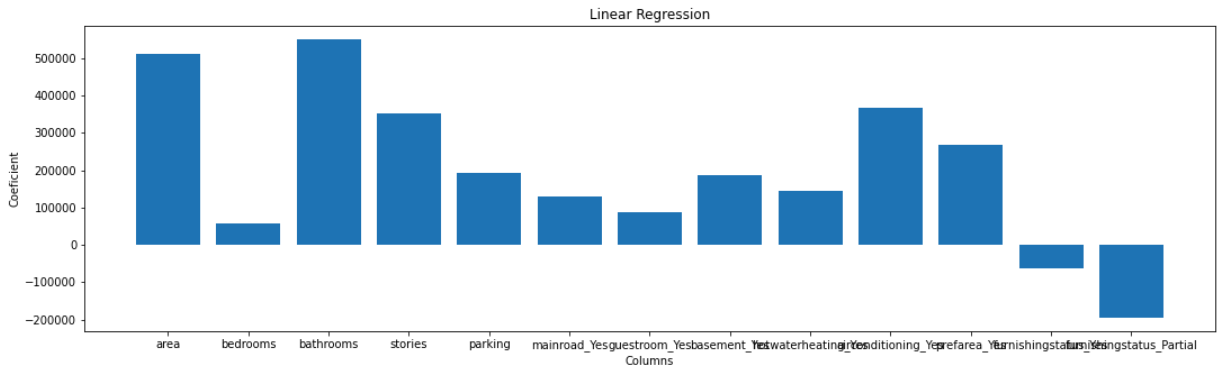
In [36]:
```
lr.coef_
```

Out[36]:
```
array([ 511615.56377666,   56615.57245779,  549420.50124098,
        353158.42985604,  193542.78167455,  128151.92129533,
         88590.21346152,  186194.15050566,  143233.20624958,
        367817.89491558,  267018.66081239,  -62550.29721128,
       -193987.7810882 ])
```

In [37]:
```
x.columns
```

Out[37]:
```
Index(['area', 'bedrooms', 'bathrooms', 'stories', 'parking', 'mainroad_Yes',
       'guestroom_Yes', 'basement_Yes', 'hotwaterheating_Yes',
       'airconditioning_Yes', 'prefarea_Yes', 'furnishingstatus_Yes',
       'furnishingstatus_Partial'],
      dtype='object')
```

In [43]:
```
#plt.bar(x_data, y_data)
plt.figure(figsize=(18,5))
plt.title("Linear Regression")
plt.bar(x.columns, lr.coef_)
plt.xlabel("Columns")
plt.ylabel("Coeficient")
plt.show()
```



26.2 Model by Lasso (L1)

This technique is used for feature selection

In [45]:
```
# alpha: penalty corner, default 1.0
la = Lasso (alpha=0.5)
la.fit(x_train, y_train)
```

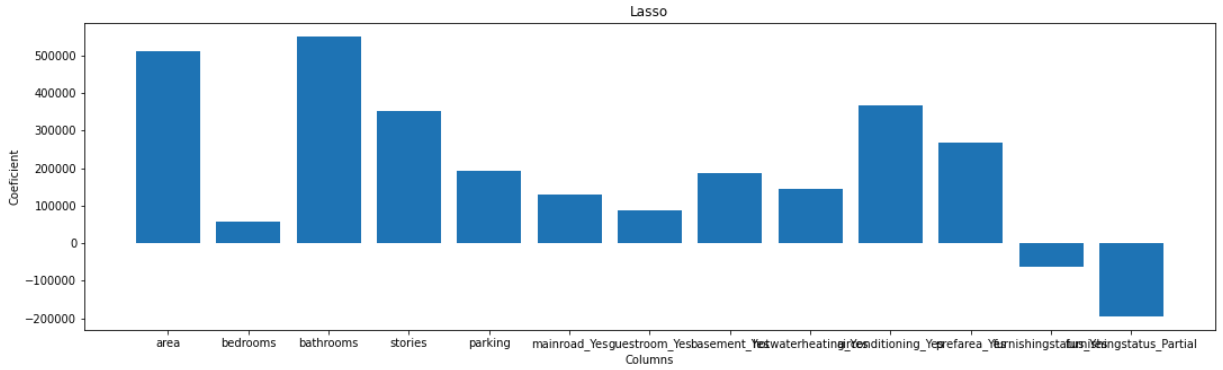
Out[45]:
```
Lasso(alpha=0.5)
```

Test the Model

In [47]:
```
la.score(x_test, y_test)*100
```

Out[47]: 65.29241383553659

In [50]:
```
#plt.bar(x_data, y_data)
plt.figure(figsize=(18,5))
plt.title("Lasso")
plt.bar(x.columns, la.coef_)
plt.xlabel("Columns")
plt.ylabel("Coeficient")
plt.show()
```



26.3 Model by Ridge (L2)

- It reduces coefficient values and save model from over-fitting

In [51]:
```
ri = Ridge(alpha=10)
ri.fit(x_train, y_train)
```

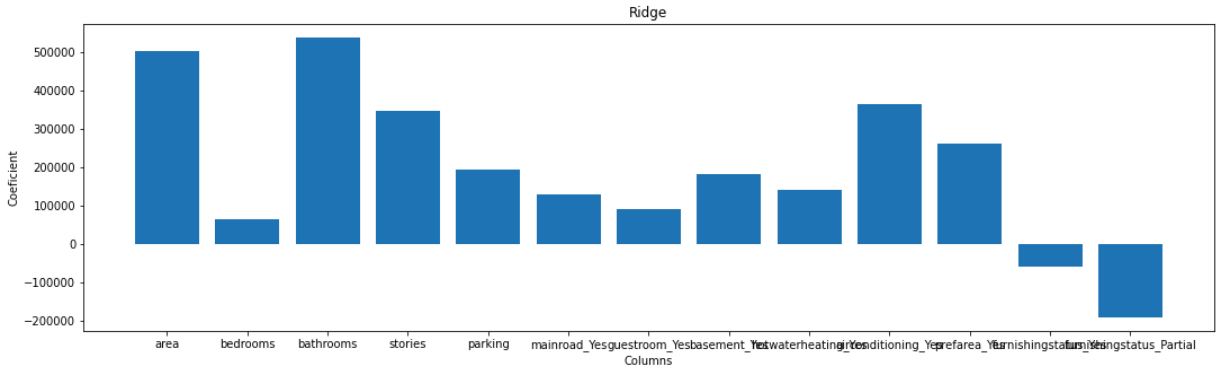
Out[51]:
```
Ridge(alpha=10)
```

Test the Model

In [53]:
```
ri.score(x_test, y_test)*100
```

Out[53]: 65.19079253215374

In [54]:
```
#plt.bar(x_data, y_data)
plt.figure(figsize=(18,5))
plt.title("Ridge")
plt.bar(x.columns, ri.coef_)
plt.xlabel("Columns")
plt.ylabel("Coeficient")
plt.show()
```



26.4 To check which model is best

26.4.1 Regression Model

In [56]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
```

In [61]:
```
#mean_squared_error(y_true, y_pred)
print(mean_squared_error(y_test, lr.predict(x_test)))
#mean_absolute_error(y_true, y_pred)
print(mean_absolute_error(y_test, lr.predict(x_test)))
# Root mean sqaure error
print(np.sqrt(mean_squared_error(y_test, lr.predict(x_test))))
```

```
1754318687330.6672
970043.4039201641
1324506.96009144
```

26.4.2 Lasso (L1) Model

In [62]:
```
#mean_squared_error(y_true, y_pred)
print(mean_squared_error(y_test, la.predict(x_test)))
#mean_absolute_error(y_true, y_pred)
print(mean_absolute_error(y_test, la.predict(x_test)))
# Root mean sqaure error
print(np.sqrt(mean_squared_error(y_test, la.predict(x_test))))
```

```
1754319323498.6353
970043.3950649527
1324507.2002441646
```

26.4.3 Ridge (L2) Model

In [63]:
```
#mean_squared_error(y_true, y_pred)
print(mean_squared_error(y_test, ri.predict(x_test)))
#mean_absolute_error(y_true, y_pred)
print(mean_absolute_error(y_test, ri.predict(x_test)))
# Root mean sqaure error
print(np.sqrt(mean_squared_error(y_test, ri.predict(x_test))))
```

```
1759455843663.3877
967942.6216085082
1326444.8136516602
```

We will use Ridge model as it is showing comparatively less error as compared to Lasso and Linear regression model

26.4.3 To compare coefficient of all models

In [64]:
```
df = pd.DataFrame({"col_name":x.columns, "LinearRegression":lr.coef_, "Lasso":la.co
df
```

Out[64]:

	col_name	LinearRegression	Lasso	Ridge
0	area	511615.563777	511615.467912	502252.286215
1	bedrooms	56615.572458	56615.441731	65132.373585
2	bathrooms	549420.501241	549420.321462	537574.041615
3	stories	353158.429856	353158.186082	346006.857732
4	parking	193542.781675	193542.619408	194954.682792
5	mainroad_Yes	128151.921295	128151.745183	130790.775299
6	guestroom_Yes	88590.213462	88590.029990	91998.609421
7	basement_Yes	186194.150506	186193.873949	181385.995261
8	hotwaterheating_Yes	143233.206250	143232.743062	140133.580908
9	airconditioning_Yes	367817.894916	367817.774947	364207.282689
10	prefarea_Yes	267018.660812	267018.388019	262517.337220
11	furnishingstatus_Yes	-62550.297211	-62549.219050	-58988.254578
12	furnishingstatus_Partial	-193987.781088	-193986.867394	-190415.566289

In [ ]:
```

```