Real Data

Load the Housing_Price.csv and see the feature_discription.txt for more insight (However you don't need to remove any unneccesary feature just use encoding to convert the categorical features)

In [204]:

```
import pandas as pd
# loading data and setting Id as index
Housing_Price = pd.read_csv("files/Housing_Price.csv").set_index("Id")
```

In [205]:

```
#Displaying the data
Housing_Price
```

Out[205]:

Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF
1	60	8450	7	5	2003	2003	70
2	20	9600	6	8	1976	1976	97
3	60	11250	7	5	2001	2002	48
4	70	9550	7	5	1915	1970	21
5	60	14260	8	5	2000	2000	65
...	...	...	...	...	...	...	.
1456	60	7917	6	5	1999	2000	
1457	20	13175	6	6	1978	1988	79
1458	70	9042	7	9	1941	2006	27
1459	20	9717	5	6	1950	1996	4
1460	20	9937	5	6	1965	1965	83

1460 rows × 41 columns

which are categorical and which are numerical features?

In [206]:

```
# Determining the categorical and numerical features of the dataset
categorical_features = []
numerical_features = []
for feat in Housing_Price:
    if str(Housing_Price[feat].dtype)=='object':
        categorical_features.append(feat)
    elif "int" in str(Housing_Price[feat].dtype) or "float" in str(Housing_Price[feat].
dtype):
        numerical_features.append(feat)

print("\033[1mThe categorical features of the data set are:\033[0m \n"+str(categorical_
features),end="\n\n")
print("\033[1mThe numerical features of the data set are:\033[0m \n"+str(numerical_feat
ures))
```

```
The categorical features of the data set are:
['Street', 'Condition1', 'Condition2', 'CentralAir', 'HeatingQC', 'LotShap
e', 'LandContour']

The numerical features of the data set are:
['MSSubClass', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'Year
RemodAdd', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlr
SF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBat
h', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGr
d', 'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSol
d', 'YrSold', 'SalePrice']
```

Write a function for doing one hot encoding for all categorical features

Hint: Use pandas.get_dummies

In [207]:

```
def onehot_encoding(df,columns=None):
    '''
    Takes the dataframe
    columns which are corresponding to categorical features
    '''
    df = pd.get_dummies(df,columns=columns)
    return df

#Converting to one-hot encoding
Housing_Price = onehot_encoding(Housing_Price,categorical_features)
```

Seperate the Label from the data, here it is 'SalePrice'

In [208]:

```
#Separating "SalesPrice" from the original data and storing it as the label
Sales_Price = Housing_Price["SalePrice"]
del Housing_Price["SalePrice"]
```

In [209]:

```
# Visualize the changed dataframe
Housing_Price
```

Out[209]:

Id	MSSubClass	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF
1	60	8450	7	5	2003	2003	70
2	20	9600	6	8	1976	1976	97
3	60	11250	7	5	2001	2002	48
4	70	9550	7	5	1915	1970	21
5	60	14260	8	5	2000	2000	65
...	...	...	...	...	...	...	.
1456	60	7917	6	5	1999	2000	
1457	20	13175	6	6	1978	1988	79
1458	70	9042	7	9	1941	2006	27
1459	20	9717	5	6	1950	1996	4
1460	20	9937	5	6	1965	1965	83

1460 rows × 67 columns

Split train test split with random state 42, test size 0.2

You can use sklearn module for this exercise

In [210]:

```
# import your Libraries
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
import matplotlib.pyplot as plt
import numpy as np
```

In [211]:

```
# Split data here
X_train, X_test, y_train, y_test = train_test_split(Housing_Price,Sales_Price, test_siz
e = 0.2, random_state = 42)
y_train = np.array(y_train).reshape(-1,1)
y_test = np.array(y_test).reshape(-1,1)
```

Lasso Regression

1. search for alphas in range of 0.1 to 1000 for Lasso rigression, choose the best which minimizes the mse
2. Plot the alphas vs MSE

Hint:

- cross validation score gives accuracy not the error convert to error appropriately (otherwise choose lambda which maximizes the score)

In the following cell, use training which was split earlier to cross validate (using cross_val_score) use cv = 5 (5 folds), then calculate the mean of the cross validation score for each alphas and plot λ vs cross_valiadtion_score or cross_validation_error. If you choose the accuracy then choose the λ which maximizes the cross_val_score.

For range of alphas use alphas = np.logspace(-1, 3, 100)

Finally find the λ which maximizes the score (or minimizes the error) (appropriate value of λ is enough just by seeing the graph)

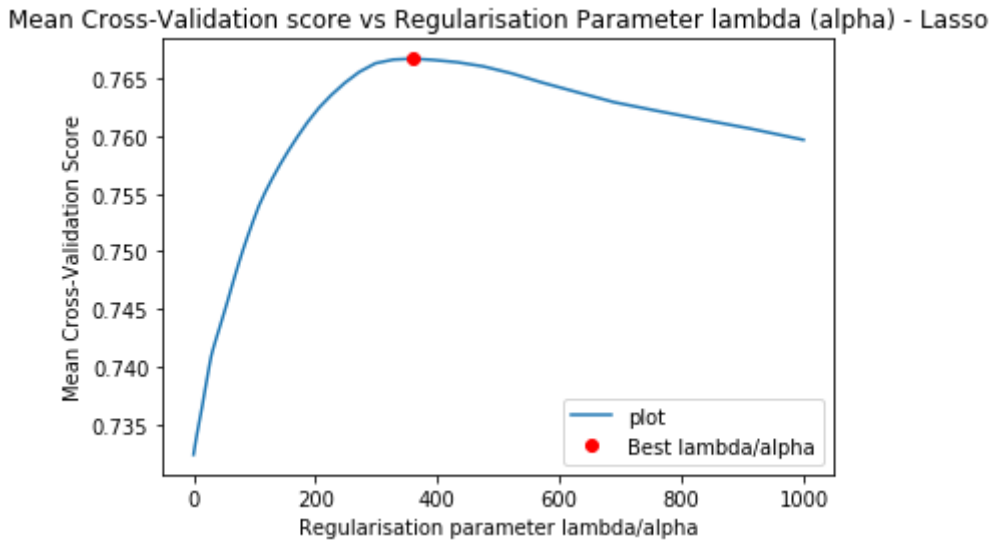
In [214]:

```
# Lasso Regression - Regularisation Parameter Search
alpha_values = np.logspace(-1,3,100)
cross_vals_lasso = []
for alpha in alpha_values:
    reg_lasso = Lasso(alpha = alpha,max_iter=10**6)
    score_list = cross_val_score(reg_lasso,X_train,y_train,cv = 5)
    #Taking mean of 5 CV scores
    cross_vals_lasso.append(sum(score_list)/5)

#Plotting
plt.plot(alpha_values,cross_vals_lasso)
plt.xlabel("Regularisation parameter lambda/alpha")
plt.ylabel("Mean Cross-Validation Score")
plt.title("Mean Cross-Validation score vs Regularisation Parameter lambda (alpha) - Las
so")
alpha_opt_lasso = alpha_values[cross_vals_lasso.index(max(cross_vals_lasso))]
plt.plot(alpha_opt_lasso,max(cross_vals_lasso),"or")
plt.legend(["plot","Best lambda/alpha"])
```

Out[214]:

```
<matplotlib.legend.Legend at 0x1f2b20aadc8>
```



In [215]:

```
print("Optimum lambda/alpha that results in maximum cross-validation score: %f"%alpha_o
pt_lasso)
print("Maximum Cross-Validation Score: %f"%max(cross_vals_lasso))
```

```
Optimum lambda/alpha that results in maximum cross-validation score: 359.3
81366
Maximum Cross-Validation Score: 0.766721
```

Ridge Regression

1. search for alphas in range of 0.1 to 100 for Ridge rigression, choose the best which minimizes the mse
2. Plot the alphas vs MSE

This similar to as explained for Lasso regression. Again plot λ vs accuracy (or error)

For range of alphas use alphas = np.logspace(-1, 2, 100)

Finally find the λ which maximizes the score (or minimizes the error) (appropriate value of λ is enough just by seeing the graph)

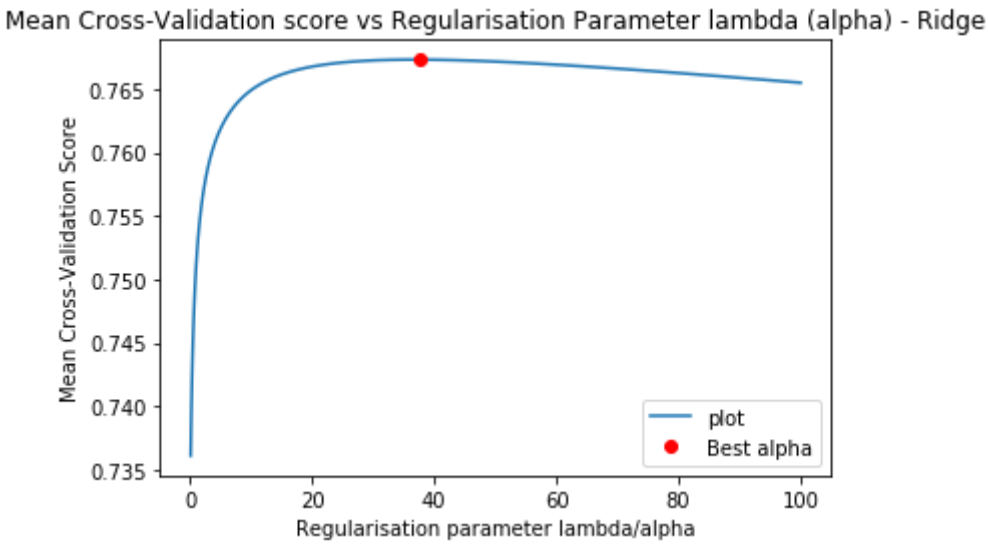
In [216]:

```
# Ridge Regression - Regularisation Parameter Search
alpha_values = np.logspace(-1,2,100)
cross_vals_ridge = []
for alpha in alpha_values:
    reg_ridge = Ridge(alpha = alpha,max_iter=10**6)
    score_list = cross_val_score(reg_ridge,X_train,y_train,cv = 5)
    #Taking mean of 5 CV scores
    cross_vals_ridge.append(sum(score_list)/5)

#Plotting
plt.plot(alpha_values,cross_vals_ridge)
plt.xlabel("Regularisation parameter lambda/alpha")
plt.ylabel("Mean Cross-Validation Score")
plt.title("Mean Cross-Validation score vs Regularisation Parameter lambda (alpha) - Rid
ge")
alpha_opt_ridge = alpha_values[cross_vals_ridge.index(max(cross_vals_ridge))]
plt.plot(alpha_opt_ridge,max(cross_vals_ridge),"or")
plt.legend(["plot","Best alpha"])
```

Out[216]:

```
<matplotlib.legend.Legend at 0x1f2b2105a88>
```



In [217]:

```
print("Optimum lambda/alpha that results in maximum cross-validation score: %f"%alpha_o
pt_ridge)
print("Maximum Cross-Validation Score: %f"%max(cross_vals_ridge))
```

```
Optimum lambda/alpha that results in maximum cross-validation score: 37.64
9358
Maximum Cross-Validation Score: 0.767319
```

Now compare regularized models to linear Regression model

In the following cell, calculate cross validation score using Linear Regression model

In [218]:

```
# Calculating Cross Validation score for simple linear regression
reg_linear = LinearRegression()
score_list = cross_val_score(reg_linear,X_train,y_train, cv =5)
cross_val_linear = sum(score_list)/5
```

In [219]:

```
print("\033[1mCross-Validation Score for Linear Regression without regularisaiton\033[0
m = %f"%cross_val_linear)
```

```
Cross-Validation Score for Linear Regression without regularisaiton = 0.73
1913
```

It can be observed that the cross-validation score with simple linear regression is lesser than those obtained from Ridge and Lasso Regularised Regression modelswith the optimimum value of lambda (alpha), with ridge model having the highest score of 0.7673

Now you have λ values for both ridge and lasso regression, predict the model on the test data you created earlier

In the following cell use selected λ as the model parameter, predict on test data, compare among three models and report your findings.

Finally use lasso regression to find the important features and write your observations and also what do you observe when you compare both coefficients of Ridge and Lasso ? Do you see any property of Lasso which is used?

Hint:

- Check weights corresponding to each features

Note:

- Don't worry if you have huge error in prediction, it is possible, just compare among models and report which has lease error.

In [220]:

```
#Linear model with no regularisation
print("\033[1mLinear Model without regularisation\033[0m")
reg_linear.fit(X_train,y_train)
print("Coefficients: \n")
feature_coeff_lin = pd.Series(reg_linear.coef_.flatten(),index=Housing_Price.columns)

with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more
 options can be specified also
    print(feature_coeff_lin)

print("\nIntercept: \n")
print(reg_linear.intercept_,end="\n\n")

#Predicting on test data
y_pred_lin = reg_linear.predict(X_test).reshape(-1,1)
#Calculating MSE
mse_lin = np.sum((y_pred_lin-y_test)**2)/len(y_test)
print("Test MSE for simple linear Regression = %f"%mse_lin,end = "\n\n")
```

Linear Model without regularisation

Coefficients:

MSSubClass	-174.568153
LotArea	0.462216
OverallQual	17594.616989
OverallCond	4491.264941
YearBuilt	335.758423
YearRemodAdd	138.787487
BsmtFinSF1	12.382085
BsmtFinSF2	-3.618867
BsmtUnfSF	-1.353687
TotalBsmtSF	7.409531
1stFlrSF	14.871679
2ndFlrSF	11.655131
LowQualFinSF	8.863129
GrLivArea	35.389940
BsmtFullBath	8704.702431
BsmtHalfBath	-3598.431914
FullBath	1755.390348
HalfBath	252.320829
BedroomAbvGr	-8304.254354
KitchenAbvGr	-11119.994853
TotRmsAbvGrd	5005.944750
Fireplaces	3906.861937
GarageCars	8409.794684
GarageArea	8.961866
WoodDeckSF	20.718350
OpenPorchSF	-4.758060
EnclosedPorch	12.363338
3SsnPorch	22.239134
ScreenPorch	56.981872
PoolArea	-25.270300
MiscVal	-0.432830
MoSold	-286.460251
YrSold	-512.612923
Street_Grvl	-12014.717589
Street_Pave	12014.717589
Condition1_Artery	-2134.463142
Condition1_Feedr	-2569.257961
Condition1_Norm	10323.752176
Condition1_PosA	-12386.211148
Condition1_PosN	3356.702494
Condition1_RRAe	-14313.995696
Condition1_RRAn	2544.567267
Condition1_RRNe	3524.529773
Condition1_RRNn	11654.376238
Condition2_Artery	54058.518746
Condition2_Feedr	2940.194852
Condition2_Norm	14728.236834
Condition2_PosA	47840.743408
Condition2_PosN	-126933.263910
Condition2_RRAe	-7611.253969
Condition2_RRAn	17326.669157
Condition2_RRNn	-2349.845117
CentralAir_N	5088.691019
CentralAir_Y	-5088.691019
HeatingQC_Ex	4328.384709
HeatingQC_Fa	-873.584827
HeatingQC_Gd	-3758.924524
HeatingQC_Po	2676.827463

```
HeatingQC_TA          -2372.702821
LotShape_IR1          13493.102267
LotShape_IR2          23998.011968
LotShape_IR3         -49755.568817
LotShape_Reg          12264.454583
LandContour_Bnk      -17206.451336
LandContour_HLS       21025.928494
LandContour_Low        2252.554344
LandContour_Lvl       -6072.031502
dtype: float64

Intercept:

[-7286.88718071]

Test MSE for simple linear Regression = 1380512172.416198
```

In [221]:

```
#Lasso model - with lambda(alpha) = 359.381
print("\033[1mLasso Model with lambda (alpha) = %f"%alpha_opt_lasso+"\033[0m")
reg_lasso = Lasso(alpha=alpha_opt_lasso,max_iter=10**6)
reg_lasso.fit(X_train,y_train)
print("Coefficients: \n")

feature_coeff_lasso = pd.Series(reg_lasso.coef_,index=Housing_Price.columns)
with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more
 options can be specified also
    print(feature_coeff_lasso)
print("\nIntercept: \n")
print(reg_lasso.intercept_,end="\n\n")

#Predicting on test data
y_pred_lasso = reg_lasso.predict(X_test).reshape(-1,1)
#Calculating MSE
mse_lasso = np.sum((y_pred_lasso-y_test)**2)/len(y_test)
print("Test MSE for Lasso Regression = %f"%mse_lasso,end = "\n\n")
```

Lasso Model with lambda (alpha) = 359.381366
Coefficients:

MSSubClass	-183.777177
LotArea	0.395699
OverallQual	18500.951101
OverallCond	3432.178831
YearBuilt	314.327053
YearRemodAdd	173.956347
BsmtFinSF1	14.548230
BsmtFinSF2	0.000000
BsmtUnfSF	2.233262
TotalBsmtSF	1.700208
1stFlrSF	43.534317
2ndFlrSF	41.946652
LowQualFinSF	37.097535
GrLivArea	5.611505
BsmtFullBath	7877.489419
BsmtHalfBath	-0.000000
FullBath	0.000000
HalfBath	-0.000000
BedroomAbvGr	-6816.675271
KitchenAbvGr	-0.000000
TotRmsAbvGrd	3809.730262
Fireplaces	3759.239103
GarageCars	7937.630221
GarageArea	11.420706
WoodDeckSF	27.232678
OpenPorchSF	0.322330
EnclosedPorch	11.021751
3SsnPorch	38.947306
ScreenPorch	64.523821
PoolArea	-32.571229
MiscVal	-0.876764
MoSold	-310.411985
YrSold	-206.714361
Street_Grvl	-0.000000
Street_Pave	0.000000
Condition1_Artery	-0.000000
Condition1_Feedr	-0.000000
Condition1_Norm	11896.447704
Condition1_PosA	-0.000000
Condition1_PosN	-0.000000
Condition1_RRAe	-0.000000
Condition1_RRAn	0.000000
Condition1_RRNe	0.000000
Condition1_RRNn	0.000000
Condition2_Artery	0.000000
Condition2_Feedr	-0.000000
Condition2_Norm	0.000000
Condition2_PosA	0.000000
Condition2_PosN	-0.000000
Condition2_RRAe	-0.000000
Condition2_RRAn	-0.000000
Condition2_RRNn	-0.000000
CentralAir_N	0.000000
CentralAir_Y	-0.000000
HeatingQC_Ex	5186.673498
HeatingQC_Fa	0.000000
HeatingQC_Gd	-0.000000
HeatingQC_Po	0.000000

```
HeatingQC_TA            -0.000000
LotShape_IR1            70.461655
LotShape_IR2          1592.067099
LotShape_IR3            -0.000000
LotShape_Reg          -669.162537
LandContour_Bnk      -5108.288645
LandContour_HLS      13070.399598
LandContour_Low          0.000000
LandContour_Lvl         -0.000000
dtype: float64

Intercept:

[-623673.46889225]

Test MSE for Lasso Regression = 1379208206.332942
```

In [222]:

```
#Ridge model - with lambda(alpha) = 37.649
print("\033[1mRidge Model with lambda (alpha) = %f"%alpha_opt_ridge+"\033[0m")
reg_ridge = Ridge(alpha=alpha_opt_ridge,max_iter=10**6)
reg_ridge.fit(X_train,y_train)
print("Coefficients: \n")
feature_coeff_ridge = pd.Series(reg_ridge.coef_.flatten(),index=Housing_Price.columns)
with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more
 options can be specified also
    print(feature_coeff_ridge)
print("\nIntercept: \n")
print(reg_ridge.intercept_,end="\n\n")

#Predicting on test data
y_pred_ridge = reg_ridge.predict(X_test).reshape(-1,1)

#Calculating MSE
mse_ridge = np.sum((y_pred_ridge-y_test)**2)/len(y_test)
print("Test MSE for Ridge Regression = %f"%mse_ridge,end = "\n\n")
```

Ridge Model with lambda (alpha) = 37.649358
Coefficients:

MSSubClass	-176.219642
LotArea	0.360275
OverallQual	17540.037180
OverallCond	4233.372718
YearBuilt	328.783030
YearRemodAdd	158.241795
BsmtFinSF1	11.249902
BsmtFinSF2	-3.552983
BsmtUnfSF	-1.039077
TotalBsmtSF	6.657842
1stFlrSF	13.598776
2ndFlrSF	11.959053
LowQualFinSF	8.272128
GrLivArea	33.829957
BsmtFullBath	8326.367252
BsmtHalfBath	-1654.508545
FullBath	1885.151340
HalfBath	-32.848715
BedroomAbvGr	-7349.848074
KitchenAbvGr	-4725.450018
TotRmsAbvGrd	4543.960677
Fireplaces	4623.492820
GarageCars	8092.348151
GarageArea	11.222400
WoodDeckSF	25.654866
OpenPorchSF	-2.637462
EnclosedPorch	12.462311
3SsnPorch	36.532220
ScreenPorch	62.336512
PoolArea	-30.908412
MiscVal	-0.878972
MoSold	-329.345735
YrSold	-445.876052
Street_Grvl	-1116.001237
Street_Pave	1116.001237
Condition1_Artery	-461.033633
Condition1_Feedr	-2750.047758
Condition1_Norm	9977.827481
Condition1_PosA	-1436.091095
Condition1_PosN	-3497.969261
Condition1_RRAe	-2855.678069
Condition1_RRAn	209.836803
Condition1_RRNe	72.023244
Condition1_RRNn	741.132286
Condition2_Artery	1700.382739
Condition2_Feedr	-118.259280
Condition2_Norm	4231.846726
Condition2_PosA	938.211403
Condition2_PosN	-6258.107918
Condition2_RRAe	-437.993032
Condition2_RRAn	5.706656
Condition2_RRNn	-61.787294
CentralAir_N	2794.218433
CentralAir_Y	-2794.218433
HeatingQC_Ex	4303.467840
HeatingQC_Fa	599.261710
HeatingQC_Gd	-2752.014846
HeatingQC_Po	-106.206427

```
HeatingQC_TA         -2044.508276
LotShape_IR1          2174.316922
LotShape_IR2          7069.774987
LotShape_IR3         -9232.992080
LotShape_Reg           -11.099830
LandContour_Bnk      -9453.112293
LandContour_HLS      10615.756911
LandContour_Low       1980.275460
LandContour_Lvl      -3142.920079
dtype: float64

Intercept:

[-137915.12302492]

Test MSE for Ridge Regression = 1360214643.751835
```

Among the three models, the mean-squared error of the test data is the least in the case of Ridge Regression.

Lasso Regression yielded a sparse solution with many of the coefficients being zero (32 out of 67 features). On the other hand, Simple Linear Regression and Ridge Regression yielded non-sparse solutions.

Out of the remaining 35 features having non-zero coefficients in Lasso, 27 of them are positive and 8 of them are negative.

In [223]:

```
feature_coeff = pd.Series(reg_lasso.coef_,index=Housing_Price.columns)
print("Feature Coefficient in Descnding Order (Lasso):\n")
with pd.option_context('display.max_rows', None, 'display.max_columns', None):  # more
 options can be specified also
    print(feature_coeff.sort_values(ascending = False))
```

```
Feature Coefficient in Descnding Order (Lasso):

OverallQual          18500.951101
LandContour_HLS      13070.399598
Condition1_Norm      11896.447704
GarageCars            7937.630221
BsmtFullBath          7877.489419
HeatingQC_Ex          5186.673498
TotRmsAbvGrd          3809.730262
Fireplaces            3759.239103
OverallCond           3432.178831
LotShape_IR2          1592.067099
YearBuilt              314.327053
YearRemodAdd           173.956347
LotShape_IR1            70.461655
ScreenPorch             64.523821
1stFlrSF                43.534317
2ndFlrSF                41.946652
3SsnPorch               38.947306
LowQualFinSF            37.097535
WoodDeckSF              27.232678
BsmtFinSF1              14.548230
GarageArea              11.420706
EnclosedPorch           11.021751
GrLivArea                5.611505
BsmtUnfSF                2.233262
TotalBsmtSF              1.700208
LotArea                  0.395699
OpenPorchSF              0.322330
BsmtHalfBath            -0.000000
FullBath                 0.000000
HalfBath                -0.000000
KitchenAbvGr            -0.000000
BsmtFinSF2               0.000000
LandContour_Lvl         -0.000000
Street_Grvl             -0.000000
Condition2_Feedr        -0.000000
LotShape_IR3            -0.000000
HeatingQC_Fa             0.000000
CentralAir_Y            -0.000000
CentralAir_N             0.000000
Condition2_RRNn         -0.000000
Condition2_RRAn         -0.000000
Condition2_RRAe         -0.000000
Condition2_PosN         -0.000000
Condition2_PosA          0.000000
Condition2_Norm          0.000000
HeatingQC_TA            -0.000000
Condition2_Artery        0.000000
Condition1_RRNn          0.000000
HeatingQC_Po             0.000000
Condition1_RRNe          0.000000
Condition1_RRAn          0.000000
Condition1_RRAe         -0.000000
Condition1_PosN         -0.000000
Condition1_PosA         -0.000000
Condition1_Feedr        -0.000000
Condition1_Artery       -0.000000
Street_Pave              0.000000
LandContour_Low          0.000000
HeatingQC_Gd            -0.000000
```

```
MiscVal                -0.876764
PoolArea              -32.571229
MSSubClass           -183.777177
YrSold               -206.714361
MoSold               -310.411985
LotShape_Reg         -669.162537
LandContour_Bnk     -5108.288645
BedroomAbvGr        -6816.675271
dtype: float64
```

Upon observing the feature coefficients predicted by Lasso Regression model, the most important features with high positive coefficients are **OverallQual (Quality of overall material and finish of the house), LandContour_HLS (Hillside land contour), Condition1_Norm (normal proximity), GarageCars (Size of Garage), BsmtFullBath (Basement full bathrooms), HeatingQC_Ex (Excellent Heating Quality), TotRmsAbvGrd (Total rooms above grade), Fireplaces (Number of Fireplaces), OverallCond (Overall conditions), LotShape_IR2 (Moderately irregular shape of property), YearBuilt (Original Construction Date), YearRemodAdd (Remodel Date)**, in the decreasing order of their importance. These features have a positive effect on the increase in Sales Price.

The features with high negative coefficients are **MSSubClass (Dwelling type), YrSold (Year Sold), MoSold (Month Sold), LotShape_Reg (Regular Shape of property), LandContour_Bnk (Banked Land Contour), BedroomAbvGr (Bedrooms above grade)**. These features have a negative effect on the increase in sales price.

NOTE: There are other features which have small nonzero coefficients (eg. ScreenPorch, GarageArea, etc) in the Lasso model. They have comparitively low influence on the sales price.

The feature coefficients predicted by Lasso Regression model were different from those predicted by Ridge and linear regression models, which did not yield a sparse solution. There was no zero coefficients for any feature in the case of ridge regression. For example, the coefficients of features such as **BsmtHalfBath (Basement half bathrooms), FullBath (Full bathrooms above grade), KitchenAbvGr (Kitchens above grade),** and many others which are zero in the case of Lasso regression, were found to be non-zero in the case of Ridge regression (having values -1654.5,1885.15, -4725.45, respectively). Some of these features had positive and others had negative coefficients in the latter (Ridge).

It is also important to note that although most feature coefficients had the same sign in Ridge and Lasso (if they are not zero in Lasso), two features - **BsmtUnfSF (Unfinished sq ft of basement area)** and **OpenPorch (Open porch area)** had coefficients with opposite signs, as depicted below:

In [146]:

```
#Lasso feature coefficients of BsmtUnfSF and OpenPorchSF
feature_coeff_lasso["BsmtUnfSF"],feature_coeff_lasso["OpenPorchSF"]
```

Out[146]:

```
(2.2332622668059794, 0.3223303373100734)
```

In [147]:

```
#Ridge feature coefficients of BsmtUnfSF and OpenPorchSF
feature_coeff_ridge["BsmtUnfSF"],feature_coeff_ridge["OpenPorchSF"]
```

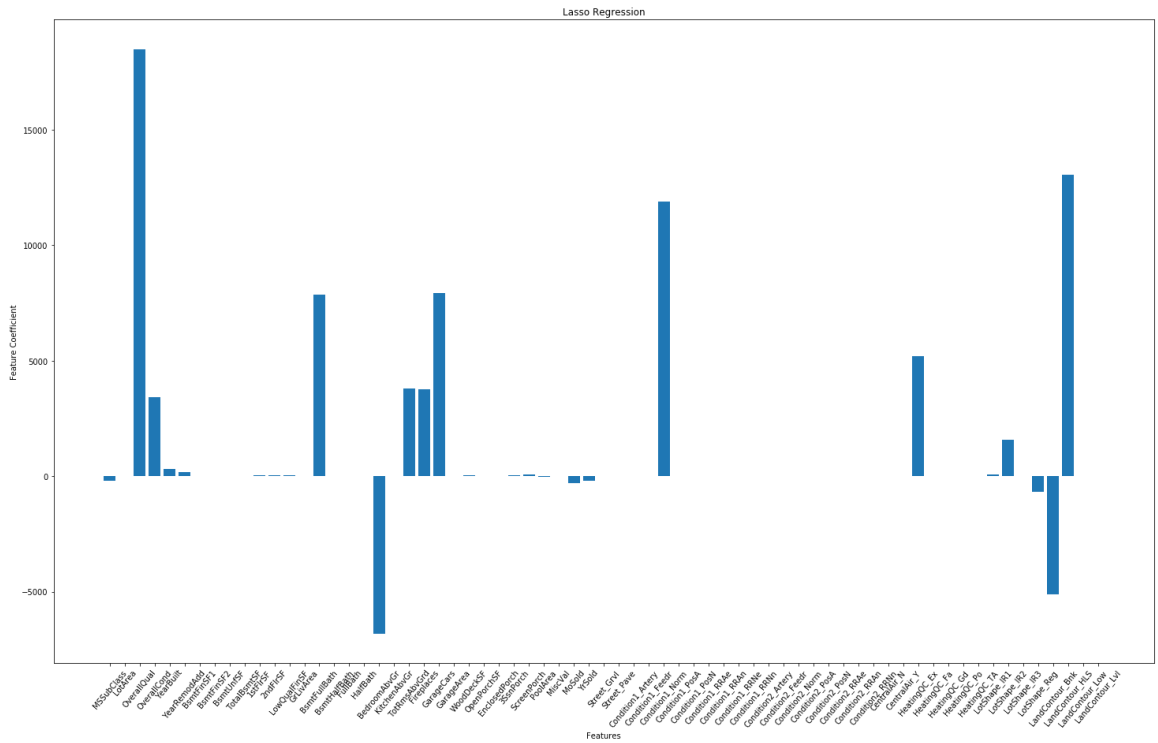
Out[147]:

```
(-1.0390774812719914, -2.6374617902148434)
```

Otherwise, in most cases, the non-zero features of Lasso Regression model were of the same order of magnitude as that of Ridge Regression Model.

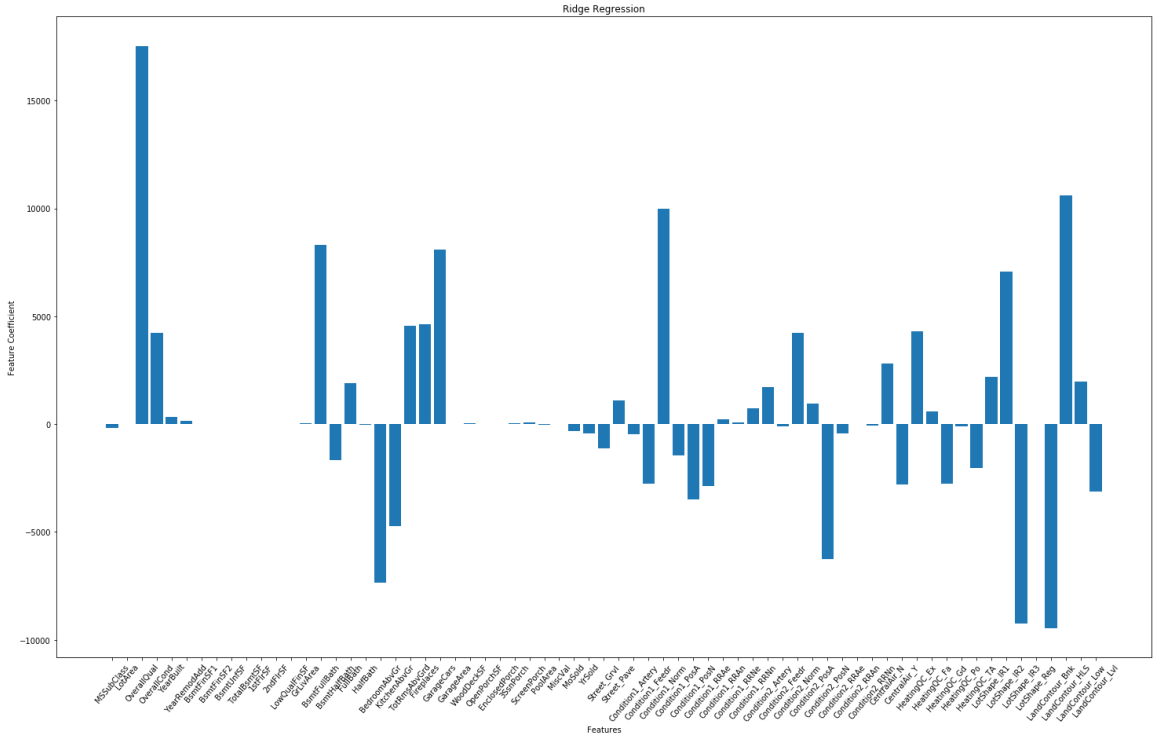
In [224]:

```
#Bar Plot Showing feature coefficients of Lasso Regression
plt.figure(figsize = (25,15))
plt.bar(feature_coeff_lasso.index,feature_coeff_lasso)
plt.ylabel("Feature Coefficient")
plt.xlabel("Features")
plt.title("Lasso Regression")
ax = plt.xticks(rotation = 50)
```



In [225]:

```
#Bar Plot Showing feature coefficients of Ridge Regression
plt.figure(figsize = (25,15))
plt.bar(feature_coeff_ridge.index,feature_coeff_ridge)
plt.ylabel("Feature Coefficient")
plt.xlabel("Features")
plt.title("Ridge Regression")
ax = plt.xticks(rotation = 50)
```



These observation prove the fundamental property of Lasso Regression, which is feature selection. This property can be used to determine the important features of the dataset, while zeroing out others, thereby preventing overfitting. On the other hand, ridge regression might drive some coefficients close to zero but never absolute zero.

In [ ]: