Regularization and Gradient Descent on housing price data

Working on DataSet from Kaggle and Using linear regression to predict prices of new houses.

Target : SalePrice in dollars Features : Month Sold , Year Sold , Condition of Sale etc.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
```

In [2]:

```
data = pd.read_csv("data/Ames_Housing_Sales.csv")
```

In [3]:

```

# 79 Features and one Predictor column
print(data.shape)

```

```
(1379, 80)
```

In [4]:

```
#Print no of integers, floats and strings
data.dtypes.value_counts()
```

Out[4]:

```
object     43
float64    21
int64      16
dtype: int64
```

Preprocessing Steps

1. One hot encode categoricals using Pandas get_dummies method .
2. Split the data into train and test sets.
3. Log transform skewed features.

1. Applying One-hot encoding for Categorical Variables using pandas get_dummies().

In [5]:

```
# Get a Pd.Series consisting of all the string categoricals
one_hot_encode_cols = data.dtypes[data.dtypes == np.object]  # filt
one_hot_encode_cols = one_hot_encode_cols.index.tolist()  # list of

# Here we see another way of one-hot-encoding:
# Encode these columns as categoricals so one hot encoding works on
for col in one_hot_encode_cols:
    data[col] = pd.Categorical(data[col])

# Do the one hot encoding
data = pd.get_dummies(data, columns=one_hot_encode_cols)
```

2. Splitting data to Train and Test

In [6]:

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(data, test_size=0.3, random_state=42
```

3. Finding Skewed Columns and appling Log transform to skewed data

Note - Our Predictor "SalePrice" should not be log transformed. Trasform all other columns where skew is greater than 0.75.

In [7]:

```
mask = data.dtypes == np.float
float_cols = data.columns[mask]

skew_limit = 0.75
skew_vals = train[float_cols].skew()

skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {0}'.format(skew_limit)))

# Mute the setting wtih a copy warnings
pd.options.mode.chained_assignment = None

for col in skew_cols.index.tolist():
    if col == "SalePrice":
        continue
    train[col] = np.log1p(train[col])
    test[col]  = test[col].apply(np.log1p)

```

In [8]:

```
feature_cols = [x for x in train.columns if x != 'SalePrice']
X_train = train[feature_cols]
y_train = train['SalePrice']

X_test  = test[feature_cols]
y_test  = test['SalePrice']
```

Calculate mean_sqared error, root-mean-squared error of Linear Regression model

In [10]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
linearRegression = LinearRegression().fit(X_train, y_train)
y_predict = linearRegression.predict(X_test)
mean_sqaured_error = mean_squared_error(y_test, y_predict)
linearRegression_rmse = np.sqrt(mean_sqaured_error)

print(linearRegression_rmse)
```

```
306369.6834231772
```

Plotting Predicted vs Actual Sale Price of Model

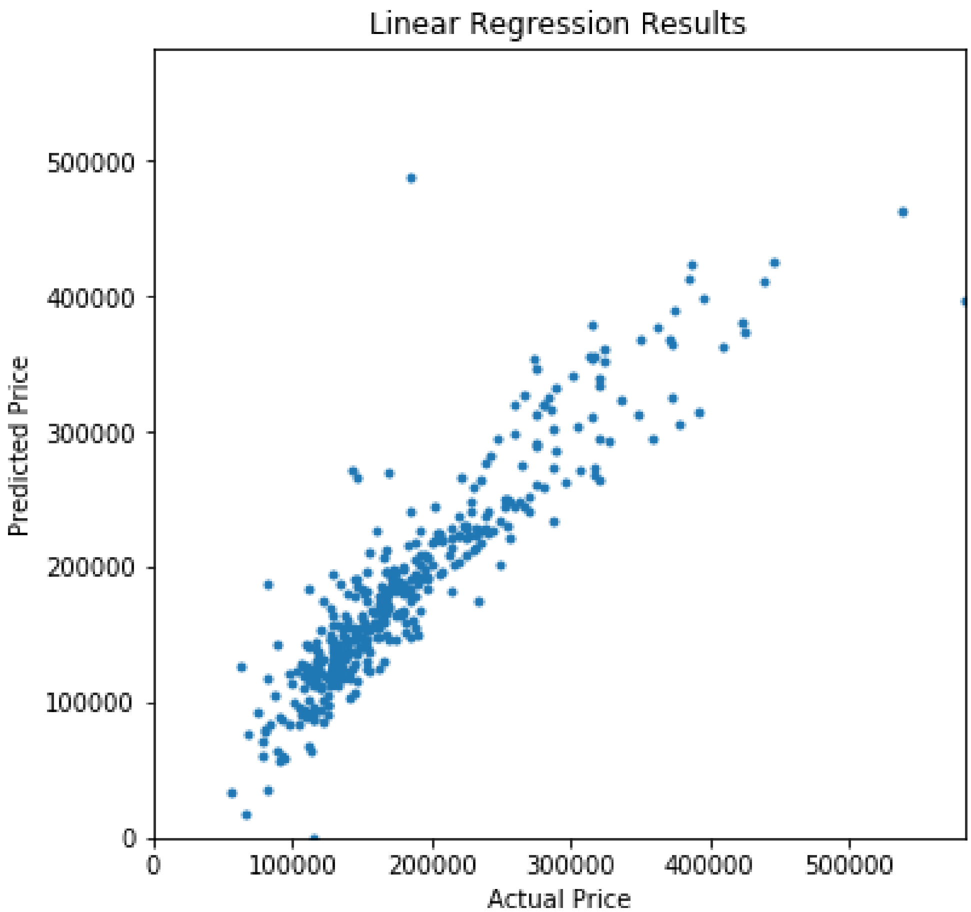
In [11]:

```
f = plt.figure(figsize=(6,6))
ax = plt.axes()

ax.plot(y_test, linearRegression.predict(X_test),
         marker='o', ls='', ms=3.0)

lim = (0, y_test.max())

ax.set(xlabel='Actual Price',
       ylabel='Predicted Price',
       xlim=lim,
       ylim=lim,
       title='Linear Regression Results');
```



Comparing rmse with RidgeCV, LassoCV and ElasticNetCV

1. Ridge Regression

Ridge Regression adds square of Co-effients to Cost Function to reduce magnitudes.It's used in case of high variance. Ridge regression uses L2 normalization and Cross validation buit in.

In [12]:

```
from sklearn.linear_model import RidgeCV

alphas = [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 80]

ridgeCV = RidgeCV(alphas=alphas,
                  cv=4).fit(X_train, y_train)

y_predict = ridgeCV.predict(X_test)
mean_sqaured_error = mean_squared_error(y_test, y_predict)
ridgeCV_rmse = np.sqrt(mean_sqaured_error)

print(ridgeCV_rmse)
```

```
32169.17620567246
```

2. LassoCV Regression

LassoCV adds absolute value of coefficinets to Cost Function to reduce magnitudes.It's used in case of high variance. Ridge regression uses L1 normalization and Cross validation buit in.

In [13]:

```
from sklearn.linear_model import LassoCV

alphas2 = np.array([1e-5, 5e-5, 0.0001, 0.0005])

lassoCV = LassoCV(alphas=alphas2,
                  max_iter=5e4,
                  cv=3).fit(X_train, y_train)

y_predict = lassoCV.predict(X_test)
mean_sqaured_error = mean_squared_error(y_test, y_predict)
lassoCV_rmse = np.sqrt(mean_sqaured_error)
print( lassoCV_rmse)
```

```
39257.3939914415
```

3. ElasticNetCV Regression

ElasticNetCV Ridge regression uses L1 and L2 normalization and Cross validation buit in.

In [14]:

```
from sklearn.linear_model import ElasticNetCV

l1_ratios = np.linspace(0.1, 0.9, 9)

elasticNetCV = ElasticNetCV(alphas=alphas2,
                            l1_ratio=l1_ratios,
                            max_iter=1e4).fit(X_train, y_train)

y_predict = elasticNetCV.predict(X_test)
mean_sqaured_error = mean_squared_error(y_test, y_predict)
elasticNetCV_rmse = np.sqrt(mean_sqaured_error)
print( elasticNetCV_rmse)
```

```
35001.234296074574
```

Comparing rmse of Linear ,Rigde ,Lasso and ElasticNet

In [15]:

```
rmse_vals = [linearRegression_rmse, ridgeCV_rmse, lassoCV_rmse, ela

labels = ['Linear', 'Ridge', 'Lasso', 'ElasticNet']

rmse_df = pd.Series(rmse_vals, index=labels).to_frame()
rmse_df.rename(columns={0: 'RMSE'}, inplace=1)
rmse_df
```

Out[15]:

	RMSE
Linear	306369.683423
Ridge	32169.176206
Lasso	39257.393991
ElasticNet	35001.234296

Summary

1. Standardizing data refers to transforming each variable to a standard normal distribution.
2. Without scaling Coefficients in linear regression depends on feature scale .
3. Lasso Scaling is important otherwise those coefficients will be peanlized to zero.
4. Calculated RMSE of Linear regression, Rigde, Lasso and Elastic Net
5. From Results Ridge is giving better reults

In [ ]:

```

```