Regression 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 [55]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [56]:

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

In [57]:

```


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

```

```
(1379, 80)
```

In [58]:

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

Out[58]:

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

Applying One-hot encoding for Categorical Variables.

In [59]:

```
# Select the object (string) columns
mask = data.dtypes == np.object
categorical_cols = data.columns[mask]

# Determine how many extra columns would be created
num_ohc_cols = (data[categorical_cols]
                .apply(lambda x: x.nunique())
                .sort_values(ascending=False))


# No need to encode if there is only one value
small_num_ohc_cols = num_ohc_cols.loc[num_ohc_cols>1]

# Number of one-hot columns is one less than the number of categories
small_num_ohc_cols -= 1

# 80 changed to 215 columns, assuming the original ones are dropped.
small_num_ohc_cols.sum()
```

Out[59]:

```
215
```

Creating two data sets.

1. With One-hot encoding (data_ohc)
2. One without One-hot encoding(dropping string Categoricals - data)

In [60]:

```
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

# Copy of the data
data_ohc = data.copy()

# The encoders
le = LabelEncoder()
ohc = OneHotEncoder()

for col in num_ohc_cols.index:

    # Integer encode the string categories
    dat = le.fit_transform(data_ohc[col]).astype(np.int)

    # Remove the original column from the dataframe
    data_ohc = data_ohc.drop(col, axis=1)

    # One hot encode the data--this returns a sparse array
    new_dat = ohc.fit_transform(dat.reshape(-1,1))

    # Create unique column names
    n_cols = new_dat.shape[1]
    col_names = ['_'.join([col, str(x)]) for x in range(n_cols)]

    # Create the new dataframe
    new_df = pd.DataFrame(new_dat.toarray(),
                          index=data_ohc.index,
                          columns=col_names)

    # Append the new data to the dataframe
    data_ohc = pd.concat([data_ohc, new_df], axis=1)
```

In [61]:

```
data_ohc.shape[0]
data_ohc.shape[1]
data_ohc.shape[1]- data.shape[1]
```

Out[61]:

```
215
```

In [62]:

```
#215 columns added
print(data_ohc.shape)
```

```
(1379, 295)
```

In [63]:

```
# Remove the string columns from the dataframe
data = data.drop(num_ohc_cols.index, axis=1)
# Removing 43 String columns
print(data.shape)
```

```
(1379, 37)
```

Create Training and Test Sets of both datasets.

In [64]:

```
from sklearn.model_selection import train_test_split

y_col = 'SalePrice'

# Split not one-hot encoded data
feature_cols = [x for x in data.columns if x != y_col]
X_data = data[feature_cols]
y_data = data[y_col]

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
                                                    test_size=0.3, rand
# Split one-hot encoded data
feature_cols = [x for x in data_ohc.columns if x != y_col]
X_data_ohc = data_ohc[feature_cols]
y_data_ohc = data_ohc[y_col]

X_train_ohc, X_test_ohc, y_train_ohc, y_test_ohc = train_test_split(X_d
                                                    test_size=0.3, rand
```

Linear Regression on Both data sets

In [65]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

LR = LinearRegression()

# Storage for error values
error_df = list()

# Data that have not been one-hot encoded
LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

error_df.append(pd.Series({'train': mean_squared_error(y_train, y_train
                           'test' : mean_squared_error(y_test,  y_test_
                           name='no enc'))

# Data that have been one-hot encoded
LR = LR.fit(X_train_ohc, y_train_ohc)
y_train_ohc_pred = LR.predict(X_train_ohc)
y_test_ohc_pred = LR.predict(X_test_ohc)

error_df.append(pd.Series({'train': mean_squared_error(y_train_ohc, y_t
                           'test' : mean_squared_error(y_test_ohc,  y_t
                          name='one-hot enc'))

# Assemble the results
error_df = pd.concat(error_df, axis=1)
error_df
```

Out[65]:

	no enc	one-hot enc
train	1.131507e+09	3.177303e+08
test	1.372182e+09	3.180592e+19

```
Note : Error on one hot encoded data is much higher .It's due to
overfitting .
More parameters Train set error will be less
```

Scaling

Note - Scaling to be done on training data and apply that it to test data.

In [66]:

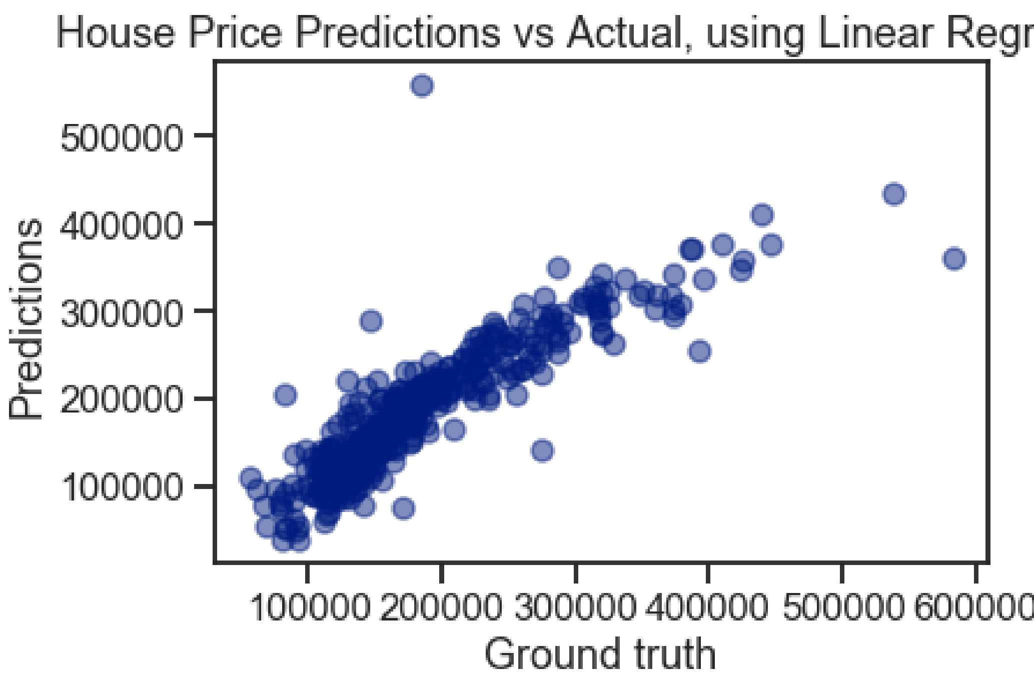
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, MaxAbsS


scalers = {'standard': StandardScaler(),
           'minmax': MinMaxScaler() }

# initialize model
LR = LinearRegression()
errors = {}
for scaler_label, scaler in scalers.items():
    trainingset = scaler.fit_transform(X_train)
    testset = scaler.transform(X_test)
    LR.fit(trainingset, y_train)
    predictions = LR.predict(testset)
    key = scaler_label + 'scaling'
    errors[key] = mean_squared_error(y_test, predictions)

errors = pd.Series(errors)
for key, error_val in errors.items():
    print(key, error_val)
```

```
standardscaling 1372182358.9345071
minmaxscaling 1372182358.9345083
```

Plotting Predictions vs Actual value of House prices using Linear Regression

In [67]:

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline


sns.set_context('talk')
sns.set_style('ticks')
sns.set_palette('dark')

ax = plt.axes()
# we are going to use y_test, y_test_pred
ax.scatter(y_test, y_test_pred, alpha=.5)

ax.set(xlabel='Ground truth',
       ylabel='Predictions',
       title=' House Price Predictions vs Actual, using Linear Regressi
```



Cross Validation

Use the KFolds object to split data into multiple folds.

Note - Data is split into three folds: Fold 1, Fold 2, and Fold 3. and Created a loop to get different cross valiadation scores.

In [68]:

```
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import KFold, cross_val_predict
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline
```

In [69]:

```
kf = KFold(shuffle=True, random_state=72018, n_splits=3)
```

In [73]:

```
#from sklearn.metrics import r2_score, mean_squared_error

scores = []
lr = LinearRegression()

for train_index, test_index in kf.split(X_data):
    X_train, X_test, y_train, y_test = (X_data.iloc[train_index, :],
                                        X_data.iloc[test_index, :],
                                        y_data[train_index],
                                        y_data[test_index])

    lr.fit(X_train, y_train)

    y_pred = lr.predict(X_test)

    score = r2_score(y_test.values, y_pred)

    scores.append(score)

scores
```

Out[73]:

```
[0.8276298196712322, 0.7332863782908314, 0.793519617860957
5]
```

Pipeline and cross_val_predict

Note - Using cross_val_predict and score

In [76]:

```
s = StandardScaler()
lr = LinearRegression()
estimator = Pipeline([("scaler", s),
                      ("regression", lr)])
```

In [78]:

```
kf
predictions = cross_val_predict(estimator, X_data, y_data, cv=kf)
r2_score(y_data, predictions)
np.mean(scores)
```

Out[78]:

```
0.7848119386076737
```

Hyperparameter tuning

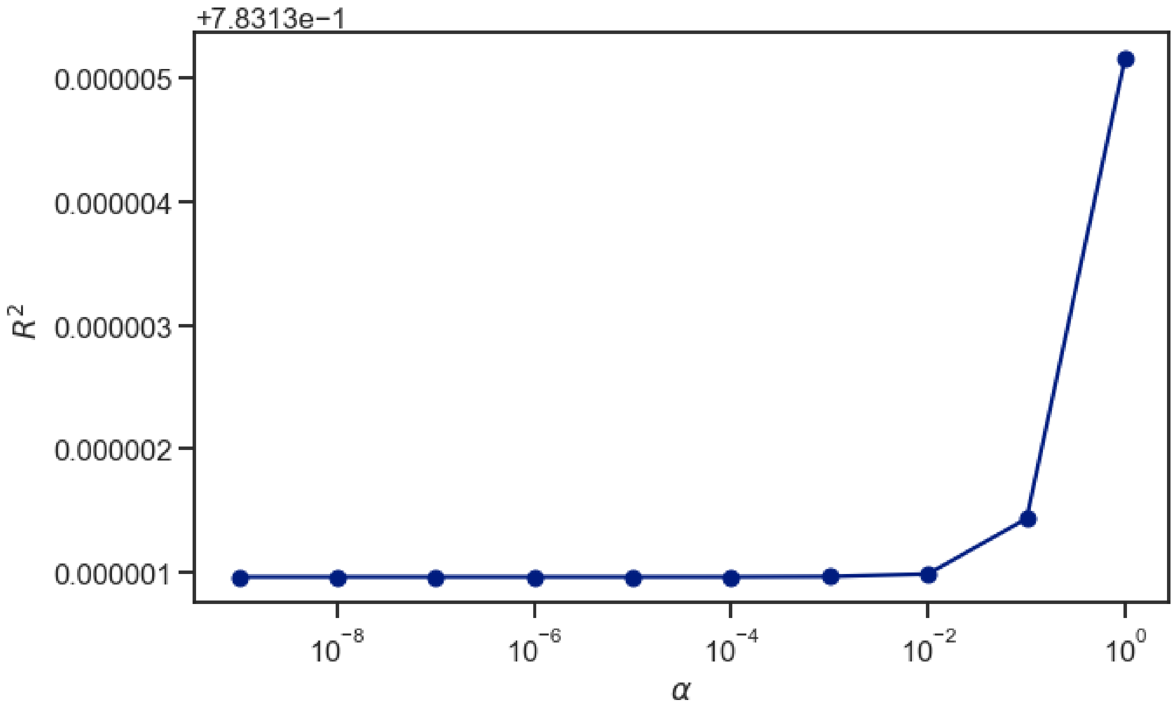
```
**Hyperparameter tuning** involves using cross validation (or train-test split) to determine
which hyperparameters are most likely to generate a model that
_generalizes_ well outside of your sample.
Note - use cross_val_predict and score to fit different
hyperparameters and chose best one
```

In [81]:

```
alphas = np.geomspace(1e-9, 1e0, num=10)
scores = []
coefs = []
for alpha in alphas:
    las = Lasso(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("lasso_regression", las)])

    predictions = cross_val_predict(estimator, X_data, y_data, cv = kf)

    score = r2_score(y_data, predictions)

    scores.append(score)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 214684174036.2053
2, tolerance: 589798902.7702838
  positive)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 208262009201.4824
2, tolerance: 557472645.5882791
  positive)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 244885729881.8297
7, tolerance: 573580033.6402674
  positive)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
```

In [82]:


+7.8313e−1
0.000005
0.000004
0.000003
0.000002
0.000001
R^2
10^{-8}
10^{-6}
10^{-4}
10^{-2}
10^{0}
α

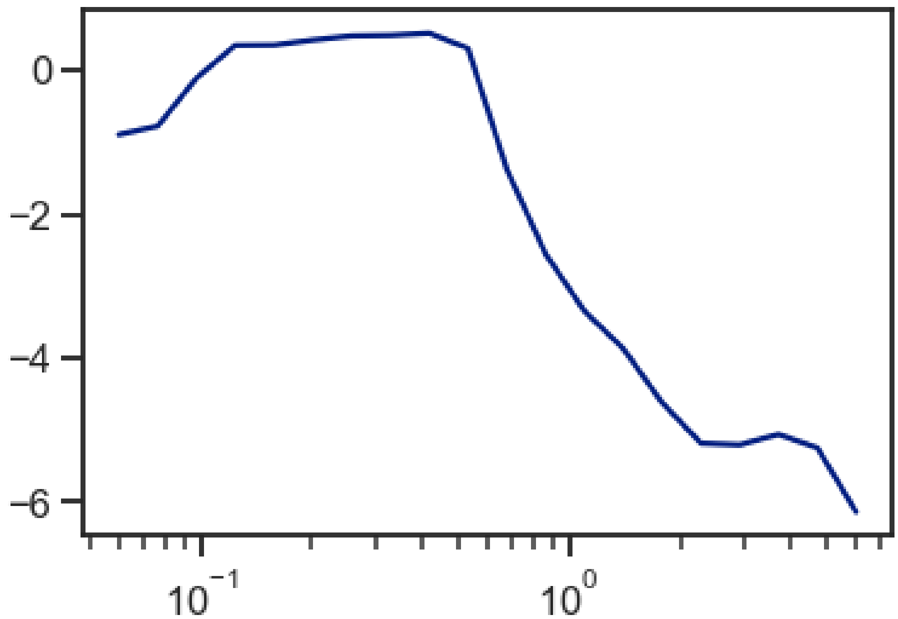
In [84]:

```
pf = PolynomialFeatures(degree=3)

scores = []
alphas = np.geomspace(0.06, 6.0, 20)
for alpha in alphas:
    las = Lasso(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("make_higher_degree", pf),
        ("lasso_regression", las)])

    predictions = cross_val_predict(estimator, X_data, y_data, cv = kf)

    score = r2_score(y_data, predictions)

    scores.append(score)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 802328621.6763387,
tolerance: 589798902.7702838
  positive)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 737409652.8110356,
tolerance: 557472645.5882791
  positive)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
r_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 763520608.0106488,
tolerance: 573580033.6402674
  positive)
/opt/anaconda3/lib/python3.7/site-packages/sklearn/linea
```

In [85]:

```
plt.semilogx(alphas, scores);
```



In [ ]:

```
# Once we have found the hyperparameter (alpha~1e-2=0.01)
# make the model and train it on ALL the data
# Then release it into the wild .....
best_estimator = Pipeline([
                    ("scaler", s),
                    ("make_higher_degree", PolynomialFeatures(degree=2)
                    ("lasso_regression", Lasso(alpha=0.03))])

best_estimator.fit(X, y)
best_estimator.score(X, y)
```

Summary

```
1. We can manually generate folds by using KFolds
2. We can get a score using cross_val_predict.
```

3. We can do hyperparameter tunring and select particular alpha and use it in Ridge , Lasso regression
4. GridSearchCV finds best hyperparameter and calculate best estimator.
5. RandomSearchCV tries random combination of model parameters.
6. Lasso and Ridge Regression with proper hyperparameter tuning gives better result than Linear Regression