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 [42]:

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

In [43]:

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

In [44]:

```


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

```

```
(1379, 80)
```

In [45]:

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

Out[45]:

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

Applying One-hot encoding for Categorical Variables.

In [46]:

```
# 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[46]:

```
215
```

Creating two data sets.

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

In [47]:

```
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 [48]:

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

Out[48]:

```
215
```

In [49]:

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

```
(1379, 295)
```

In [50]:

```
# 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 [51]:

```
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 [52]:

```
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[52]:

	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 [53]:

```
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

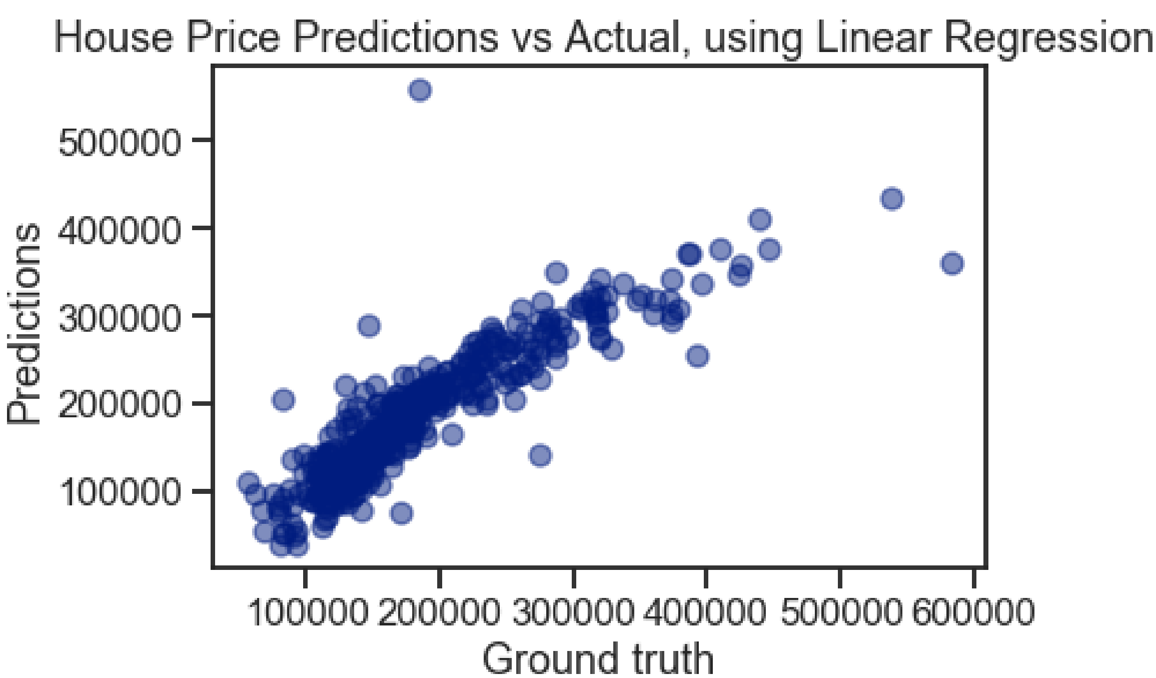
In [54]:

```
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
```



In [ ]: