HousePricePrediction

March 30, 2017

In [10]:
```
#import all the required modules
import os
import sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.ensemble import RandomForestRegressor

#from sklearn.model_selection import cross_val_score
from sklearn.cross_validation import cross_val_score
from scipy.stats import skew
from sklearn import decomposition
```

In [11]:
```
###################################section 1 : Read Data#####################
#read the training dataset
train_df = pd.read_csv("train.csv", delimiter=',')

test_df = pd.read_csv("test.csv", delimiter=',')

train_df.head()
###################################section 1 : End############################
```

Out[11]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	\
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	

	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscV
0	Lvl	AllPub	...	0	NaN	NaN	NaN	
1	Lvl	AllPub	...	0	NaN	NaN	NaN	
2	Lvl	AllPub	...	0	NaN	NaN	NaN	
3	Lvl	AllPub	...	0	NaN	NaN	NaN	
4	Lvl	AllPub	...	0	NaN	NaN	NaN	

```
  MoSold YrSold  SaleType  SaleCondition  SalePrice
0      2   2008        WD         Normal     208500
1      5   2007        WD         Normal     181500
2      9   2008        WD         Normal     223500
3      2   2006        WD        Abnorml     140000
4     12   2008        WD         Normal     250000

[5 rows x 81 columns]
```

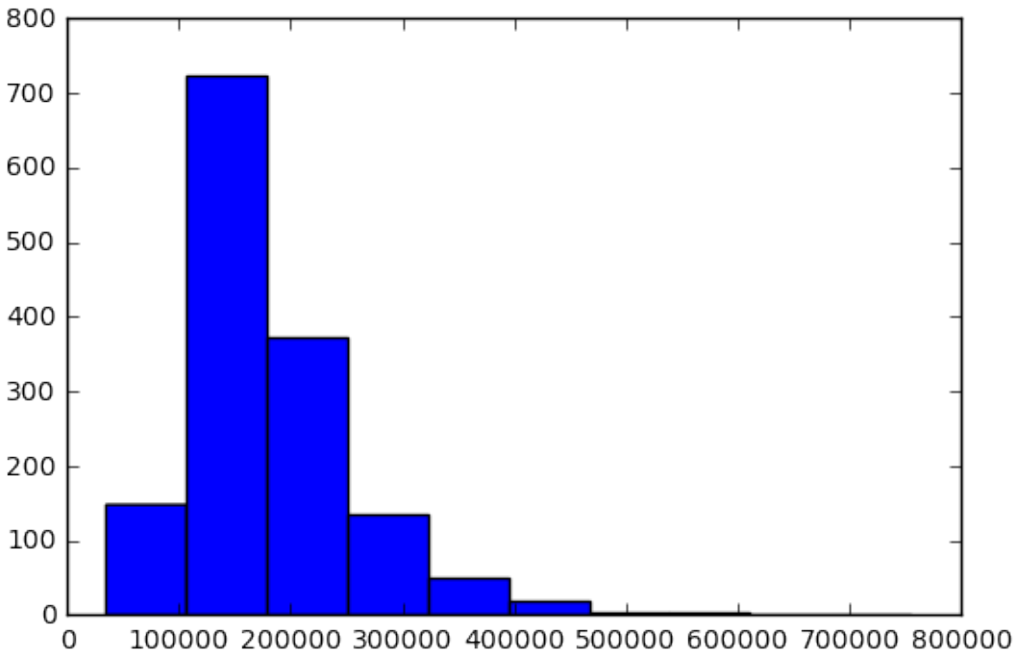
In [20]:
```
##################################section 2: define functions##############
####In this section different functions are defined to perform operations
####later stage of the application code. Advantage of following function a
####is code reusability.

# Function1: to normalize the dataset
def normalizeData(Numeric_columns):
    means = df.loc[:, Numeric_columns].mean()
    stdev = df.loc[:, Numeric_columns].std()
    df.loc[:, Numeric_columns] = (df.loc[:, Numeric_columns] - means) / st

    index_train = df.loc[train_df.index]
    index_test = df.loc[test_df.index]

    xTrain=index_train.values
    xTest=index_test.values

    df['LotArea'] = np.log(df['LotArea'])
    df['LotFrontage'] = np.log(df['LotFrontage'])
    return index_train,index_test,xTrain,xTest
# Function1:end

# Function2: Store target variable and remove skewness from data
def _removeSkewness():
    target = train_df['SalePrice']
    plt.hist(target)
    plt.show()
    del train_df['SalePrice']

    yTrain = np.log(target)
    plt.hist(yTrain)
    plt.xlabel('SalePrice')
    plt.show()
    return yTrain
# Function2:end

# Function3: this fucntion is very important, it assign dummy variables fo
# the categorical features. It also handles the missing values by assignme
```

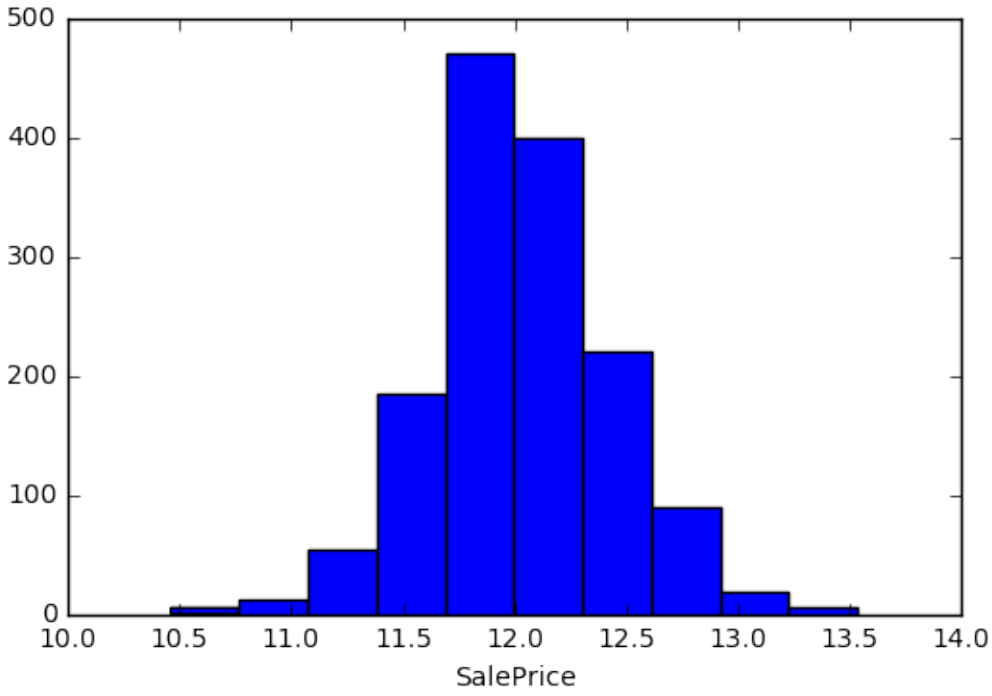
```
# mean.
def _dummyCreate():
   df = pd.get_dummies(alldf)
   df.isnull().sum().sort_values(ascending=False)
   df = df.fillna(df.mean())
   return df
# Function3:end

"""
"""
# Function4: function to perform PCA(principal component analysis) and per
# lasso regression on test data to predict the selling price.
def _pcaLassoRegr():
    pca = decomposition.PCA()
    pca.fit(xTrain)

    fig = plt.figure(1, figsize=(4, 3))

    plt.clf()
    plt.axes([.2, .2, .7, .7])
    plt.plot(pca.explained_variance_, linewidth=2)
    plt.axis('tight')
    plt.xlabel('n_components')
    plt.ylabel('explained_variance_')
    plt.show()

    train_pca = pca.transform(xTrain)
    test_pca = pca.transform(xTest)

    lassoregr = LassoCV(alphas=[0.1,0.001,0.0001,1,2,3,4,5,6,7,8,9,10,11,1
    rmse= np.sqrt(-cross_val_score(lassoregr, train_pca,yTrain,
                                   scoring="neg_mean_squared_error", cv =
    print("root mean squared error of lasso is:",rmse)
    y_lasso = lassoregr.predict(xTest)

    return y_lasso
# Function4:end

#"""
#"""
#def _lassoRegr():
#    # Fitting the model and predicting using Lasso Regression
#    lassoregr = LassoCV(alphas=[0.1,0.001,0.0001,1,2,3,4,5,6,7,8,9,10,11,
#    y_lasso = lassoregr.predict(xTest)
#
#    # Root mean squre with lasso regression
#    rmse = np.sqrt(-cross_val_score(lassoregr, xTrain, yTrain, scoring="n
#    print ("Root mean square error of Lasso regression", rmse)
```

```
#
#    return y_lasso


"""
"""
# Function5: function to perform ridge regression on test data to predict
def _ridgeRegr():
    # Fitting the model and predicting using Ridge Regression
    ridgeregr = RidgeCV(alphas=[0.1,0.001,0.0001,1,2,3,4,5,6,7,8,9,10,11,1
    y_ridge = ridgeregr.predict(xTest)

    # Root mean squre with Ridge Regression
    ridgermse = np.sqrt(-cross_val_score(ridgeregr, xTrain, yTrain,
                                         scoring="neg_mean_squared_error",
    print ("Root mean square error of rigde:",ridgermse)

    return y_ridge
# Function5:end

"""
"""
# Function6: function to perform xgboost regression on test data to predic
def _xboost():
    # Fitting the model and predicting using xgboost
    regr = xgb.XGBRegressor(colsample_bytree=0.4,
            gamma=0.045,
            learning_rate=0.07,
            max_depth=20,
            min_child_weight=1.5,
            n_estimators=300,
            reg_alpha=0.65,
            reg_lambda=0.45,
            subsample=0.95)

    regr.fit(xTrain, yTrain)
    y_pred_xgb = regr.predict(xTest)

    # Root mean squre with xboost
    xboostrmse = np.sqrt(-cross_val_score(regr, xTrain, yTrain,
                                          scoring="neg_mean_squared_error'
    print ("Root mean square error of xboost:",xboostrmse)

    return  y_pred_xgb
# Function6:end

"""
"""
```

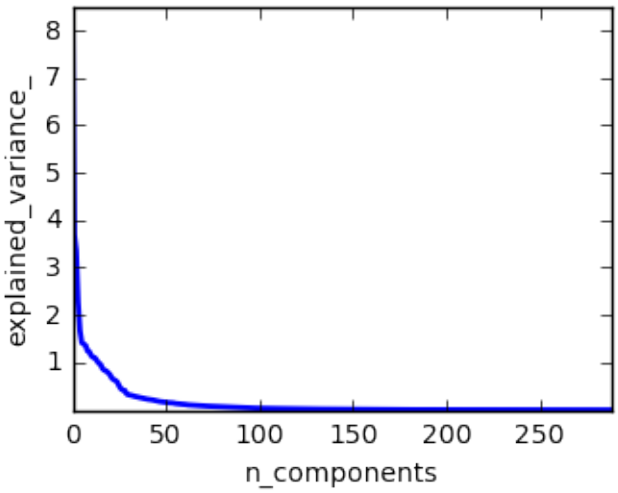
```
        # Function7: function to perform random on test data to predict the selli
        def _randomForest():
            rf = RandomForestRegressor(10, max_features='sqrt')
            rf.fit(xTrain,yTrain)
            y_rf = rf.predict(xTest)

            # Root mean squre with randomforest
            rndmfrmse = np.sqrt(-cross_val_score(rf, xTrain, yTrain,
                                                 scoring="neg_mean_squared_error",
            print ("Root mean square error of randomforest:",rndmfrmse)

            return y_rf
        # Function7:end

        # Function8: function to store the predicted values in the csv file.
        """
        Please change submission file path below.
        """
        def _submission(y_final):
            # Preparing for submissions
            submission_df = pd.DataFrame(data= {'Id' : test_df.Id, 'SalePrice': y_
            submission_df.to_csv('submisison.csv', index=False)
        # Function8:end
        ###################################section 2: End#############################
```

In [13]:
```
        ####################section 3: use all the functions defined above########

        # Remove skewness
        yTrain = _removeSkewness()

        # Concatenates the data
        alldf = pd.concat((train_df, test_df), axis=0,ignore_index=True)

        # Creates dummy variables
        df = _dummyCreate()

        # Retrieve all numeric features
        numeric_columns = alldf.columns[alldf.dtypes != 'object']

        # Normalize the data set
        index_train,index_test,xTrain,xTest = normalizeData(numeric_columns)
```





In [21]:
```
# Using PCA and LassoReggression
# Use this function and comment the functions below while running pca+lass
y_final_lasso = _pcaLassoRegr()
```




```
root mean squared error of lasso is: 0.14593883217
```

```
In [22]: # Using RidgeReggression
         # Use this function and comment the remaining functions while running ridg
         y_final_ridge = _ridgeRegr()
```

```
Root mean square error of rigde: 0.1392427785
```

```
In [23]: # Using xboost
         y_final_xb = _xboost()
```

```
Root mean square error of xboost: 0.128156202379
```

```
In [24]: # Using RandomForest
         # Use this function and comment the remaining functions while running Rand
         y_final_rf = _randomForest()
         ##################################section 3: End#############################
```

```
Root mean square error of randomforest: 0.157803449767
```

```
In [ ]: #After running all the models we can see "Root Mean Square Error" of "xboos
        #is lowest. So, for predicting the sales price of test date we will use "xb
```

```
In [26]: #generate predicted value of sales price.
         _submission(np.exp(y_final_xb))
```