Predicting Houseprice:

This project was completed has part of Kaggle competition.

In this project we will dive deep into the following areas

1. Exploratory Data Analysis a. Dependent variable b. Univariate Analysis on Independent Variables

    1. Distribution of Numerical variables:
    2. Distribution of Categorical Variables

    c. Bivariate Analysis

    1. Correlation among Numerical Variables
    3. Mean Disparity among Categorical Variables

    d. Missing Values e. Oulier detection
2. Feature Engineering: a. Impute missing values b. Standardization c. Engineer meaningful features d. Dealing with categorical encoding
3. Model Building a. Individual Models (Lasso,Ridge,LGB,XGB,RandomForest,SVM b. Model Stacking

In [350]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax
from scipy.stats.stats import pearsonr

# Models
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,
AdaBoostRegressor, BaggingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.linear_model import Ridge, RidgeCV
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.svm import SVR
from mlxtend.regressor import StackingCVRegressor
import lightgbm as lgb
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor

# Stats
from scipy.stats import skew, norm
from scipy.special import boxcox1p
from scipy.stats import boxcox_normmax

# Misc
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import scale
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.decomposition import PCA


# Ignore useless warnings
import warnings
warnings.filterwarnings(action="ignore")
pd.options.display.max_seq_items = 8000
pd.options.display.max_rows = 8000
```

In [286]:
```
df=pd.read_csv('C:/Users/thand/Downloads/house-prices-advanced-regression-techniques/train.csv')
df.head()
```

Out[286]:

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

5 rows × 81 columns

In [256]:
```
#Lets have a peak at the columns
print('There are about {} columns'.format(len(df.columns)))
print(df.columns)
```

```
There are about 81 columns
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAd
d',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBat
h',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageTyp
e',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQua
l',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

In [287]:
```
# Variables by data types

#In this section, we will make the following lists
#num_cols: list of all numerical variables
#cat_cols: list of all categorical variables
#ordinal_cols: list of all mean encoded ordinal variables


#discrete ratio/interval to categorical features to get more info on them
numericToCategory(df,['MoSold','YrSold','TotRmsAbvGrd','OverallQual','OverallCond','YearBuilt','BsmtFullBath','BsmtHalfBath','YearRemodAdd','FullBath','HalfBath','BedroomAbvGr','KitchenAbvGr','Fireplaces','GarageYrBlt','GarageCars'])
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
num_cols= df.select_dtypes(include=numerics).columns   #list of all numerical cols
cat_cols= df.select_dtypes(exclude=numerics).columns   #list of all cat cols
#Nominal/ordinal
```

In [310]:
```
#Tree Based Models
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn import metrics



x,y=df[df.columns.difference(cat_cols)].drop(['SalePrice','Id'], axis=1), df.SalePrice
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=1)
```

In [340]:
```
#x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=1)
```

In [351]:
```
xgboost = XGBRegressor(learning_rate=0.01,
                       n_estimators=6000,
                       max_depth=4,
                       min_child_weight=0,
                       gamma=0.6,
                       subsample=0.7,
                       colsample_bytree=0.7,
                       objective='reg:linear',
                       nthread=-1,
                       scale_pos_weight=1,
                       seed=27,
                       reg_alpha=0.00006,
                       random_state=42)

rf = RandomForestRegressor(n_estimators=1200,
                           max_depth=15,
                           min_samples_split=5,
                           min_samples_leaf=5,
                           max_features=None,
                           oob_score=True,
                           random_state=42)

svr = make_pipeline(RobustScaler(), SVR(C= 20, epsilon= 0.008, gamma=0.0003))
scores = {}

ridge_alphas = [1e-15, 1e-10, 1e-8, 9e-4, 7e-4, 5e-4, 3e-4, 1e-4, 1e-3, 5e-2,
1e-2, 0.1, 0.3, 1, 3, 5, 10, 15, 18, 20, 30, 50, 75, 100]
ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=ridge_alphas, cv=kf))


lightgbm = LGBMRegressor(objective='regression',
                       num_leaves=6,
                       learning_rate=0.01,
                       n_estimators=7000,
                       max_bin=200,
                       bagging_fraction=0.8,
                       bagging_freq=4,
                       bagging_seed=8,
                       feature_fraction=0.2,
                       feature_fraction_seed=8,
                       min_sum_hessian_in_leaf = 11,
                       verbose=-1,
                       random_state=42)

gbr = GradientBoostingRegressor(n_estimators=6000,
                                learning_rate=0.01,
                                max_depth=4,
                                max_features='sqrt',
                                min_samples_leaf=15,
                                min_samples_split=10,
                                loss='huber',
                                random_state=42)


# Stack up all the models above, optimized using xgboost
stack_gen = StackingCVRegressor(regressors=(xgboost, lightgbm, svr, ridge, gbr
```

```
, rf),
                                meta_regressor=xgboost,
                                use_features_in_secondary=True)
```

In [312]:
```
score = cv_rmse(xgboost)
print("xgboost: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['xgb'] = (score.mean(), score.std())
```

```
[17:50:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:50:39] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:51:02] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:51:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:51:49] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:52:12] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:52:34] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:52:56] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:53:17] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:53:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:54:04] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[17:54:31] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
xgboost: 0.3165 (0.0627)
```

In [342]:
```
score = cv_rmse(rf)
print("rf: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['rf'] = (score.mean(), score.std())
```

```
rf: 0.3686 (0.0614)
```

In [343]:
```
score = cv_rmse(svr)
print("SVR: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['svr'] = (score.mean(), score.std())
```

```
SVR: 0.2955 (0.0553)
```

In [347]:
```
score = cv_rmse(ridge)
print("ridge: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['ridge'] = (score.mean(), score.std())
```

```
ridge: 0.2992 (0.0622)
```

In [352]:
```
score = cv_rmse(lightgbm)
print("lightgbm: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['lgb'] = (score.mean(), score.std())
```

```
lightgbm: 0.3193 (0.0550)
```

In [353]:
```
score = cv_rmse(gbr)
print("gbr: {:.4f} ({:.4f})".format(score.mean(), score.std()))
scores['gbr'] = (score.mean(), score.std())
```

```
gbr: 0.3000 (0.0556)
```

In [359]:
```
print('stack_gen')
stack_gen_model = stack_gen.fit(np.array(x_train), np.array(y_train))
print('lightgbm')
lgb_model_full_data = lightgbm.fit(x_train, y_train)
print('xgboost')
xgb_model_full_data = xgboost.fit(x_train, y_train)
print('Svr')
svr_model_full_data = svr.fit(x_train, y_train)
print('Ridge')
ridge_model_full_data = ridge.fit(x_train, y_train)
print('RandomForest')
rf_model_full_data = rf.fit(x_train, y_train)
print('GradientBoosting')
gbr_model_full_data = gbr.fit(x_train, y_train)
```

```
stack_gen
[19:08:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[19:08:20] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[19:08:40] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[19:08:59] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[19:09:19] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[19:11:10] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
[19:11:36] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
lightgbm
xgboost
[19:12:26] WARNING: src/objective/regression_obj.cu:152: reg:linear is now de
precated in favor of reg:squarederror.
Svr
Ridge
RandomForest
GradientBoosting
```

In [362]:
```
def blended_predictions(X):
    return ((0.1 * ridge_model_full_data.predict(X)) + \
            (0.2 * svr_model_full_data.predict(X)) + \
            (0.1 * gbr_model_full_data.predict(X)) + \
            (0.3 * xgb_model_full_data.predict(X)) + \
            (0.1 * lgb_model_full_data.predict(X)) + \
            (0.05 * rf_model_full_data.predict(X)) + \
            (0.15 * stack_gen_model.predict(np.array(X))))
```

In [363]:
```
blended_score = rmsle(y_train, blended_predictions(x_test))
scores['blended'] = (blended_score, 0)
print('RMSLE score on train data:')
print(blended_score)
```

```
RMSLE score on train data:
1.3257677070701186
```

In [222]:
```
# Setup cross validation folds
kf = KFold(n_splits=12, random_state=42, shuffle=True)
```

In [193]:
```
# Define error metrics
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model, X=None):
    rmse = np.sqrt(-cross_val_score(model, x_train, y_train, scoring="neg_mean
_squared_error", cv=kf))
    return (rmse)
```