In [1]:
```
# Generic inputs for most ML tasks
import pandas as pd
import numpy as np
from scipy.stats import skew, boxcox_normmax
from scipy.special import boxcox1p
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn import tree
# import graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
import xgboost as xgb

pd.options.display.float_format = '{:,.2f}'.format
pd.set_option('display.max_rows',None)

# setup interactive notebook mode
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from IPython.display import display, HTML
```

In [2]:
```
# fetch data
train_data = pd.read_csv('Datasets/train.csv')
test_data = pd.read_csv('Datasets/test.csv')

# display first few rows of train data
train_data.head()
test_data.head()

# length of train data
len(train_data)
len(test_data)

# sum of NaN values
train_data.isna().sum()
test_data.isna().sum()
```

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition	SalePrice
0	1	60	RL	65.00	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2008	WD	Normal	208500
1	2	20	RL	80.00	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	5	2007	WD	Normal	181500
2	3	60	RL	68.00	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	9	2008	WD	Normal	223500
3	4	70	RL	60.00	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	2	2006	WD	Abnorml	140000
4	5	60	RL	84.00	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	12	2008	WD	Normal	250000

5 rows × 81 columns

Out[2]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition
0	1461	20	RH	80.00	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal
1	1462	20	RL	81.00	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal
2	1463	60	RL	74.00	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal
3	1464	60	RL	78.00	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	NaN	0	6	2010	WD	Normal
4	1465	120	RL	43.00	5005	Pave	NaN	IR1	HLS	AllPub	...	144	0	NaN	NaN	NaN	0	1	2010	WD	Normal

5 rows × 80 columns

Out[2]: 1460

Out[2]: 1459

```
Out[2]: Id                 0
        MSSubClass         0
        MSZoning           0
        LotFrontage      259
        LotArea            0
        Street             0
        Alley           1369
        LotShape           0
        LandContour        0
        Utilities          0
        LotConfig          0
        LandSlope          0
        Neighborhood       0
        Condition1         0
        Condition2         0
        BldgType           0
        HouseStyle         0
        OverallQual        0
        OverallCond        0
        YearBuilt          0
        YearRemodAdd       0
        RoofStyle          0
        RoofMatl           0
        Exterior1st        0
        Exterior2nd        0
        MasVnrType       872
        MasVnrArea         8
        ExterQual          0
        ExterCond          0
        Foundation         0
        BsmtQual          37
        BsmtCond          37
        BsmtExposure      38
        BsmtFinType1      37
        BsmtFinSF1         0
        BsmtFinType2      38
        BsmtFinSF2         0
        BsmtUnfSF          0
        TotalBsmtSF        0
        Heating            0
        HeatingQC          0
        CentralAir         0
        Electrical         1
        1stFlrSF           0
        2ndFlrSF           0
        LowQualFinSF       0
        GrLivArea          0
        BsmtFullBath       0
        BsmtHalfBath       0
        FullBath           0
        HalfBath           0
        BedroomAbvGr       0
        KitchenAbvGr       0
        KitchenQual        0
        TotRmsAbvGrd       0
        Functional         0
        Fireplaces         0
        FireplaceQu      690
        GarageType        81
        GarageYrBlt       81
        GarageFinish      81
        GarageCars         0
        GarageArea         0
        GarageQual        81
        GarageCond        81
        PavedDrive         0
        WoodDeckSF         0
        OpenPorchSF        0
        EnclosedPorch      0
        3SsnPorch          0
        ScreenPorch        0
        PoolArea           0
        PoolQC          1453
        Fence           1179
        MiscFeature     1406
        MiscVal            0
        MoSold             0
        YrSold             0
        SaleType           0
        SaleCondition      0
        SalePrice          0
        dtype: int64
```

Out[2]:
```
Id                 0
MSSubClass         0
MSZoning           4
LotFrontage      227
LotArea            0
Street             0
Alley           1352
LotShape           0
LandContour        0
Utilities          2
LotConfig          0
LandSlope          0
Neighborhood       0
Condition1         0
Condition2         0
BldgType           0
HouseStyle         0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
RoofStyle          0
RoofMatl           0
Exterior1st        1
Exterior2nd        1
MasVnrType       894
MasVnrArea        15
ExterQual          0
ExterCond          0
Foundation         0
BsmtQual          44
BsmtCond          45
BsmtExposure      44
BsmtFinType1      42
BsmtFinSF1         1
BsmtFinType2      42
BsmtFinSF2         1
BsmtUnfSF          1
TotalBsmtSF        1
Heating            0
HeatingQC          0
CentralAir         0
Electrical         0
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       2
BsmtHalfBath       2
FullBath           0
HalfBath           0
BedroomAbvGr       0
KitchenAbvGr       0
KitchenQual        1
TotRmsAbvGrd       0
Functional         2
Fireplaces         0
FireplaceQu      730
GarageType        76
GarageYrBlt       78
GarageFinish      78
GarageCars         1
GarageArea         1
GarageQual        78
GarageCond        78
PavedDrive         0
WoodDeckSF         0
OpenPorchSF        0
EnclosedPorch      0
3SsnPorch          0
ScreenPorch        0
PoolArea           0
PoolQC          1456
Fence           1169
MiscFeature     1408
MiscVal            0
MoSold             0
YrSold             0
SaleType           1
SaleCondition      0
dtype: int64
```

In [3]:
```
# dropPoolQC and MiscFeature due to high amount of NaN values within column (>3/4 of data length)

train_data.drop(columns = ['PoolQC','MiscFeature'], inplace = True)
test_data.drop(columns = ['PoolQC', 'MiscFeature'], inplace = True)
```

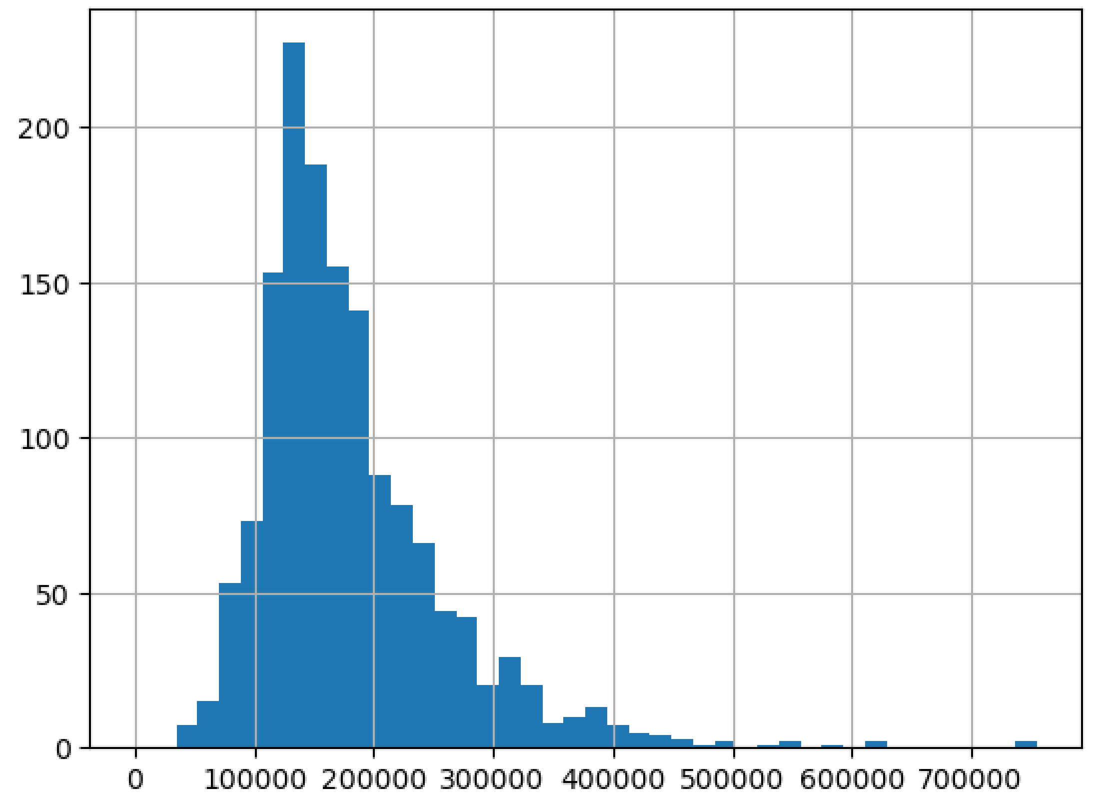
```
# Drop Id as it doesn't do anything for the data
train_data.drop(['Id'], axis=1, inplace=True)
test_data.drop(['Id'], axis=1, inplace=True)

# view SalePrice distribution
train_data['SalePrice'].hist(bins = 40)

# looks like SalePrice is skewed, so let's fix that
train_data = train_data[train_data.GrLivArea < 4500]
train_data.reset_index(drop=True, inplace=True)
train_data["SalePrice"] = np.log1p(train_data["SalePrice"])

# keep our SalePrice column as our dependent variable
y_train = train_data['SalePrice'].reset_index(drop=True)

# now it's more like a normal distribution
train_data['SalePrice'].hist(bins = 40)

# combine both train and test data to handle NaNs and missing values more easily
train_features = train_data.drop(['SalePrice'], axis=1)
test_features = test_data
features = pd.concat([train_features, test_features]).reset_index(drop=True)

# Since these column are actually a category , using a numerical number will lead the model to assume
# that it is numerical , so we convert to string .
features['MSSubClass'] = features['MSSubClass'].apply(str)
features['YrSold'] = features['YrSold'].astype(str)
features['MoSold'] = features['MoSold'].astype(str)

## Filling these columns With most suitable value for these columns
features['Functional'] = features['Functional'].fillna('Typ')
features['Electrical'] = features['Electrical'].fillna("SBrkr")
features['KitchenQual'] = features['KitchenQual'].fillna("TA")

## Filling these with MODE , i.e. , the most frequent value in these columns .
features['Exterior1st'] = features['Exterior1st'].fillna(features['Exterior1st'].mode()[0])
features['Exterior2nd'] = features['Exterior2nd'].fillna(features['Exterior2nd'].mode()[0])
features['SaleType'] = features['SaleType'].fillna(features['SaleType'].mode()[0])
features['MSZoning'] = features.groupby('MSSubClass')['MSZoning'].transform(lambda x: x.fillna(x.mode()[0]))

# fill garage data
for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
    features[col] = features[col].fillna(0)

for col in ['GarageType', 'GarageFinish', 'GarageQual', 'GarageCond']:
    features[col] = features[col].fillna('None')

# Fill basement data
for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2'):
    features[col] = features[col].fillna('None')

# Fill rest of object/categorical features with None
objects = []
for i in features.columns:
    if features[i].dtype == object:
        objects.append(i)
features.update(features[objects].fillna('None'))

# Fill rest of numerical features with 0
numeric_dtypes = ['int32', 'int64', 'float32', 'float64']
numerics = []
for i in features.columns:
    if ((features[i].dtype in numeric_dtypes) & ~(features[i].equals(features['LotFrontage']))) :
        numerics.append(i)
features.update(features[numerics].fillna(0))

# treat the skewed data through boxcox transformation
# numerics2 = []
# for i in features.columns:
#     if features[i].dtype in numeric_dtypes:
#         numerics2.append(i)
# skew_features = features[numerics2].apply(lambda x: skew(x)).sort_values(ascending=False)

# high_skew = skew_features[skew_features > 0.5]
# skew_index = high_skew.index

# for i in skew_index:
#    features[i] = boxcox1p(features[i], boxcox_normmax(features[i] + 1))

len(train_data)
len(test_data)
```

Out[3]: <Axes: >

Out[3]: <Axes: >

Out[3]: 1458

Out[3]: 1459



In [4]:
```
# FEATURE ENGINEERING PORTION
# Removing features that are not very useful . Logically, Utilities and Street shouldn't contribute much to SalePrice

features = features.drop(['Utilities', 'Street'], axis=1)


# Adding new features to condense the data

features['YrBltAndRemod']=features['YearBuilt']+features['YearRemodAdd']
features['TotalSF']=features['TotalBsmtSF'] + features['1stFlrSF'] + features['2ndFlrSF']

features['Total_sqr_footage'] = (features['BsmtFinSF1'] + features['BsmtFinSF2'] +
                                 features['1stFlrSF'] + features['2ndFlrSF'])

features['Total_Bathrooms'] = (features['FullBath'] + (0.5 * features['HalfBath']) +
                               features['BsmtFullBath'] + (0.5 * features['BsmtHalfBath']))

features['Total_porch_sf'] = (features['OpenPorchSF'] + features['3SsnPorch'] +
                              features['EnclosedPorch'] + features['ScreenPorch'] +
                              features['WoodDeckSF'])

features['haspool'] = features['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
features['has2ndfloor'] = features['2ndFlrSF'].apply(lambda x: 1 if x > 0 else 0)
features['hasgarage'] = features['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
features['hasbsmt'] = features['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
features['hasfireplace'] = features['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
```

In [5]:
```
from xgboost import XGBRegressor
# From observing, it will be for the best to fill in LotFrontage NaNs with some values
# To do this, we will use features that seem to have a correlation with LotFrontage in order to train
# and predict the values for those that have a NaN value initially.

# For reference and citing outside sources, I will be referring to
# https://www.kaggle.com/code/ogakulov/lotfrontage-fill-in-missing-values-house-prices
# as a source for methods and choosing what feature variables to use in order to predict LotFrontage.

# Instead of using the SVR classifier as the article does, I will attempt to use gradient boosting regressor.

# Drop SalePrice column from train dataset and merge into one data frame called all_data
training_data = train_data.drop('SalePrice', axis=1)
testing_data = test_data
all_data = pd.concat([training_data, testing_data], ignore_index=True).copy()

# Split into known and unknown LotFrontage records
lotFrontage_test = features[features.LotFrontage.isnull()]
lotFrontage_train = features[~features.LotFrontage.isnull()]
target = lotFrontage_train.LotFrontage
print("LotFrontage has {:} missing value, and {:} values avaialble.".format(lotFrontage_test.shape[0], lotFrontage_train.shape[0]))

# Pull only the features for training the model. Define target variable
y_lotFrontage_train = lotFrontage_train['LotFrontage']
x_lotFrontage_train = lotFrontage_train.loc[:,['LotArea', 'LotConfig', 'LotShape', 'MSZoning', 'BldgType', 'Neighborhood', 'Condition1', 'Condition2', 'GarageCars']]

# Dummify categorical variables and normalize the data
x_lotFrontage_train = pd.get_dummies(x_lotFrontage_train)
x_lotFrontage_train = (x_lotFrontage_train - x_lotFrontage_train.mean())/x_lotFrontage_train.std()
x_lotFrontage_train = x_lotFrontage_train.fillna(0)

# From Assignment 4,from testing which parameters would have given the minimum sMAPE, I decided to replicate
# those same parameters for gradient boosting
```

```
gb = GradientBoostingRegressor(n_estimators=100, learning_rate = 0.3, max_depth=11)
# gb = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05, max_depth=4, max_features='sqrt', min_samples_leaf=15, min_samples_split=10, loss='huber', random_state = 69)
# gb = XGBRegressor(learning_rate=0.11321366170467694, max_depth=3, n_estimators=832, subsample=0.8126543182197247, colsample_bytree=0.8424884692926351)
gb.fit(x_lotFrontage_train, y_lotFrontage_train)

gb.score(x_lotFrontage_train, y_lotFrontage_train)

# use gradient boosting to fill in NaN values through prediction

# Select columns for final prediction, dummify, and normalize
features_lotFrontage_NaN = features[features.LotFrontage.isnull()]
features_lotFrontage = features_lotFrontage_NaN.loc[:,['LotArea', 'LotConfig', 'LotShape', 'MSZoning', 'BldgType', 'Neighborhood', 'Condition1', 'Condition2', 'GarageCars']]
features_lotFrontage = pd.get_dummies(features_lotFrontage)
features_lotFrontage = (features_lotFrontage - features_lotFrontage.mean())/features_lotFrontage.std()
features_lotFrontage = features_lotFrontage.fillna(0)

# Make sure that dummy columns from training set are replicated in test set
for col in (set(x_lotFrontage_train.columns) - set(features_lotFrontage.columns)):
    features_lotFrontage[col] = 0

features_lotFrontage = features_lotFrontage[x_lotFrontage_train.columns]

# Assign predicted LotFrontage value into train_data
features.loc[features.LotFrontage.isnull(), 'LotFrontage'] = gb.predict(features_lotFrontage)

features.isna().sum().sum()
```

```
LotFrontage has 486 missing value, and 2431 values avaialble.
```

Out[5]:

```
GradientBoostingRegressor
GradientBoostingRegressor(learning_rate=0.3, max_depth=11)
```

Out[5]: 0.9971513167812833

Out[5]: 0

In [6]:

```
# use OneHotEncoder instead of getdummies
from sklearn.preprocessing import OneHotEncoder
# Identify categorical columns
features_col = features.select_dtypes(include=['object']).columns

# Extract categorical columns
features_cat = features[features_col]

# Using OneHotEncoder
encoder = OneHotEncoder(sparse=False)
features_encoded = pd.DataFrame(encoder.fit_transform(features_cat), columns=encoder.get_feature_names_out(features_col))

# Concatenate the one-hot encoded DataFrame with the original DataFrame
features = pd.concat([features, features_encoded], axis=1)

# Drop the original categorical columns
features = features.drop(features_col, axis=1)

features.isna().sum().sum()
features.dropna(inplace=True)
```

```
C:\Users\lawre\anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

Out[6]: 0

In [7]:

```
#  split train and test are back seperately
if True:
    X_train = features.iloc[:len(y_train), :]
    X_test = features.iloc[len(y_train):, :]
```

In [8]:

```
# feature scaling
if True:
    #Feature Scaling
    from sklearn.preprocessing import RobustScaler
    scaler = RobustScaler()
    X_train_scaled = scaler.fit_transform(X_train.values)
    X_train_scaled_df = pd.DataFrame(X_train_scaled, index = X_train.index, columns = X_train.columns)
    X_test_scaled = scaler.transform(X_test.values)
    X_test_scaled_df = pd.DataFrame(X_test_scaled, index = X_test.index, columns = X_test.columns)
    X_train = X_train_scaled_df
    X_test = X_test_scaled_df
```

In [9]:

```
if True:
    from mlxtend.regressor import StackingCVRegressor
    from sklearn.gaussian_process import GaussianProcessRegressor
    from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel, RationalQuadratic, Exponentiation
    from lightgbm import LGBMRegressor
    from xgboost import XGBRegressor
    from sklearn.linear_model import LassoCV
    from sklearn.linear_model import RidgeCV
```

```
from sklearn.preprocessing import RobustScaler, normalize
from sklearn.pipeline import make_pipeline
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold

import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# defining error functions for handy use.

kfolds = KFold(n_splits=10, shuffle=True, random_state=69)

def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))

def cv_rmse(model, X_train=X_train):
    rmse = np.sqrt(-cross_val_score(model, X_train, y_train, scoring="neg_mean_squared_error", cv=kfolds))
    return (rmse)

ridge = make_pipeline(RobustScaler(), RidgeCV(alphas=[1e-06, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10],
                                              cv=kfolds))
lasso = make_pipeline(RobustScaler(), LassoCV(max_iter=1000000, cv=kfolds, random_state=7,
                                    alphas=[1e-06, 0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]))
gb_regress = GradientBoostingRegressor(n_estimators=3000, learning_rate=0.05, max_depth=4, max_features='sqrt',
                                       min_samples_leaf=15, min_samples_split=10, loss='huber', random_state=7)

 # Define the hyperparameter space
lgb_space = {
    'max_depth': 5,
    'learning_rate': 0.15,
    'n_estimators': 3800,
    'num_leaves': 4,
    'max_bin': 150,
    'bagging_fraction': 0.7434141086967856,
    'bagging_freq': 14,
    'objective': 'regression',
    'tree_learner': 'feature',
    'boosting_type' : 'dart',
    'verbosity': -1,
    'random_state': 7
}

lightgb_regress = LGBMRegressor(**lgb_space)


# Define the hyperparameter space

xgb_space = {
    'max_depth': 3,
    'max_leaves': 24,
    'learning_rate': 0.008073243888388325,
    'n_estimators': 5300,
    'subsample': 0.6982127263866671,
    'colsample_bytree': 0.09568028958854619,
    'min_child_weight': 3,
    'alpha': 0.30714547261947767,
    'random_state': 7
}


xgboost_regress = XGBRegressor(**xgb_space)

xg_feature_importance = xgboost_regress.fit(X_train, y_train)

feature_importances = normalize([xg_feature_importance.feature_importances_])[0]

# Add in Gaussian Processs Regressor
dot_white_kernel = DotProduct() + WhiteKernel()
rat_quad_kernel = RationalQuadratic(length_scale = 1.0, alpha = 10, length_scale_bounds=(1e-06, 1000000))
combined_kernel = dot_white_kernel + rat_quad_kernel


gpr_dot_white = GaussianProcessRegressor(kernel = dot_white_kernel, n_restarts_optimizer = 3)
gpr_rat_quad = GaussianProcessRegressor(kernel = rat_quad_kernel, n_restarts_optimizer = 3)
gpr_combined = GaussianProcessRegressor(kernel = combined_kernel, n_restarts_optimizer = 3)


stack_regress = StackingCVRegressor(regressors=(ridge, lasso, gb_regress, xgboost_regress, lightgb_regress,),
                                               #  gpr_dot_white, gpr_rat_quad, gpr_combined),
                            meta_regressor=xgboost_regress,
                            use_features_in_secondary=True)

score = cv_rmse(ridge , X_train)
print("RIDGE: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

score = cv_rmse(lasso , X_train)
```

```
    print("LASSO: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

    score = cv_rmse(gb_regress)
    print("gb_regress: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

    score = cv_rmse(lightgb_regress)
    print("lightgb_regress: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

    score = cv_rmse(xgboost_regress)
    print("xgboost_regress: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

#     score = cv_rmse(gpr_dot_white)
#     print("gpr_dot_white: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

#     score = cv_rmse(gpr_rat_quad)
#     print("gpr_rat_quad: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )

#     score = cv_rmse(gpr_combined)
#     print("gpr_combined: {:.4f} ({:.4f})\n".format(score.mean(), score.std()) )


    lasso_model = lasso.fit(X_train, y_train)
    ridge_model = ridge.fit(X_train, y_train)
    stack_model = stack_regress.fit(np.array(X_train), np.array(y_train))
    gbr_model = gb_regress.fit(X_train, y_train)
    xgb_model = xgboost_regress.fit(X_train, y_train)
    lgb_model = lightgb_regress.fit(X_train, y_train)
#     gpr_model_dot_white = gpr_dot_white.fit(X_train, y_train)
#     gpr_model_rat_quad = gpr_rat_quad.fit(X_train, y_train)
#     gpr_model_combined = gpr_combined.fit(X_train, y_train)

    # blend and ensemble models
    def blend_models_predict(X_train):
        return ((0.05 * ridge_model.predict(X_train)) + \
                (0.1 * lasso_model.predict(X_train)) + \
                (0.15 * gbr_model.predict(X_train)) + \
                (0.15 * xgb_model.predict(X_train)) + \
                (0.15 * lgb_model.predict(X_train)) + \
                (0.4 * stack_model.predict(np.array(X_train))))

    print('RMSLE score on train data:')
    print(rmsle(y_train, blend_models_predict(X_train)))

    print('Predict submission')
    submission = pd.read_csv("sample_submission.csv")
    submission.iloc[:,1] = (np.expm1(blend_models_predict(X_test)))

    submission.to_csv("submission.csv", index=False)

    submission.head()
```

```
RIDGE: 0.1137 (0.0081)

LASSO: 0.1126 (0.0085)

gb_regress: 0.1150 (0.0096)

[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
lightgb_regress: 0.1167 (0.0103)

xgboost_regress: 0.1116 (0.0092)

[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
[LightGBM] [Warning] bagging_fraction is set=0.7434141086967856, subsample=1.0 will be ignored. Current value: bagging_fraction=0.7434141086967856
[LightGBM] [Warning] bagging_freq is set=14, subsample_freq=0 will be ignored. Current value: bagging_freq=14
RMSLE score on train data:
0.055450157715484685
Predict submission
```

Out[9]:

	Id	SalePrice
0	1461	123,526.32
1	1462	159,096.62
2	1463	185,531.60
3	1464	200,638.43
4	1465	187,213.63

In [10]:
```
# best rmse:
# RIDGE: 0.1134 (0.0083)

# LASSO: 0.1121 (0.0075)

# gb_regress: 0.1131 (0.0095)

# lightgb_regress: 0.1160 (0.0105)

# xgboost_regress: 0.1111 (0.0087)

# gpr_dot_white: 0.1246 (0.0067)

# gpr_rat_quad: 0.2060 (0.0269)

# gpr_combined: 0.1280 (0.0110)

# blend and ensemble models
# 0.075 * ridge_model.predict(X_train)
# 0.075 * lasso_model.predict(X_train)
# 0.15 * gbr_model.predict(X_train)
# 0.15 * xgb_model.predict(X_train)
# 0.15 * lgb_model.predict(X_train)
# 0.05 * gpr_model_dot_white.predict(X_train)
# 0.05 * gpr_rat_quad.predict(X_train)
# 0.05 * gpr_combined.predict(X_train)
# 0.4 * stack_model.predict(np.array(X_train))

# RMSLE score on train data:
# 0.04260046872041054
```

In [11]:
```
# try tuning our lightgbm to optimize parametes
if False:
    from lightgbm import LGBMRegressor
    from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
    from hyperopt.pyll import scope as ho_scope
    from sklearn.metrics import mean_squared_error
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import KFold
    import warnings

    # Filter out LightGBM warnings
    warnings.filterwarnings("ignore", category=Warning, message="bagging_fraction is set")

    # Define the hyperparameter space
    space = {
        'max_depth': 4, # ho_scope.int(hp.quniform('max_depth', low=3, high=7, q=1)),
        'learning_rate': 0.005717814490669204, # hp.loguniform('learning_rate', np.log(0.005), np.log(0.2)),
        'n_estimators': 3500, # ho_scope.int(hp.quniform('n_estimators', low=1000, high=5500, q=100)),
        'num_leaves': 8, # ho_scope.int(hp.quniform('num_leaves', low=4, high=64, q=2)),
        'max_bin': 160, # ho_scope.int(hp.quniform('max_bin', low=10, high=250, q=10)),
        'lambda_l1': hp.uniform('lambda_l1', 0, 1), # reg_alpha
        'drop_rate': hp.uniform('drop_rate', 0, 1),
        'bagging_fraction': 0.5463673909161867, # hp.uniform('bagging_fraction', 0.5, 1.0),
        'bagging_freq': 7, # ho_scope.int(hp.quniform('bagging_freq', low=0, high=25, q=1)),
        'objective': 'regression',
        'tree_learner': 'feature',
        'boosting_type' : 'dart',
        'xgboost_dart_mode': 'true',
        'verbosity': -1,
    }

    kfolds = KFold(n_splits=10, shuffle=True, random_state=69)

    # Define the objective function for regression
    def objective(params):
        lightgb_model = LGBMRegressor(**params)
        # lightgb_model.fit(X_train, y_train)
        # y_pred = lightgb_model.predict(X_test)
        # score = np.sqrt(mean_squared_error(y_test, y_pred))  # Use MSE or another regression metric
        score = -np.mean(cross_val_score(lightgb_model, X_train, y_train, cv=kfolds, scoring='neg_root_mean_squared_error'))
        return {'loss': score, 'status': STATUS_OK}  # Note that loss is now the score to minimize

    # Perform the optimization
    trials = Trials()
    best_params = fmin(objective, space, algo=tpe.suggest, max_evals=100, trials=trials)
    print("Best set of hyperparameters: ", best_params)
```

In [12]:
```
# try tuning our xgboost first to optimize parameters
if False:
    from xgboost import XGBRegressor
    from hyperopt import fmin, tpe, hp, STATUS_OK, Trials
    from hyperopt.pyll import scope as ho_scope
    from sklearn.metrics import mean_squared_error
    from sklearn.model_selection import cross_val_score
    from sklearn.model_selection import KFold
```

```python
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)

# Define the hyperparameter space
space = {
    'max_depth': 4, # ho_scope.int(hp.quniform('max_depth', low=3, high=7, q=1)),
    'max_leaves': 106, # ho_scope.int(hp.quniform('max_leaves', low=4, high=128, q=2)),
    'learning_rate': 0.026278827595007454, # hp.loguniform('learning_rate', np.log(0.005), np.log(0.15)),
    'n_estimators': 4900, # ho_scope.int(hp.quniform('n_estimators', low=1000, high=5500, q=100)),
    'subsample': hp.uniform('subsample', 0, 1),
    'colsample_bytree': hp.uniform('colsample_bytree', 0, 1),
    'min_child_weight': 0, # ho_scope.int(hp.quniform('min_child_weight', low=0, high=10, q=1)),
    'objective': 'reg:squarederror',
    'alpha': hp.uniform('alpha', 0, 1),
}

kfolds = KFold(n_splits=10, shuffle=True, random_state=69)

# Define the objective function for regression
def objective(params):
    xgb_model = xgb.XGBRegressor(**params)
    # xgb_model.fit(X_train, y_train)
    # y_pred = xgb_model.predict(X_test)
    # score = mean_squared_error(y_test, y_pred)  # Use MSE or another regression metric
    score = -np.mean(cross_val_score(xgb_model, X_train, y_train, cv=kfolds, scoring='neg_root_mean_squared_error'))
    return {'loss': score, 'status': STATUS_OK}  # Note that loss is now the score to minimize


# Perform the optimization
trials = Trials()
best_params = fmin(objective, space, algo=tpe.suggest, max_evals=100, trials=trials)
print("Best set of hyperparameters: ", best_params)
```

In [13]:

```python
# best for lightgbm
# best loss: 0.11596685440639957
'''
space = {
        'max_depth': 5,
        'learning_rate': 0.15,
        'n_estimators': 3800,
        'num_leaves': 4,
        'max_bin': 150,
        'bagging_fraction': 0.7434141086967856,
        'bagging_freq': 14,
        'objective': 'regression',
        'tree_learner': 'feature',
        'boosting_type' : 'dart',
        'verbosity': -1,
    }
'''

# learning_rate = 0.11168979095966552, max_bin=192, max_depth=3, n_estimators=2100, num_leaves=91, boosting_type='dart'
```

Out[13]:

```
"\nspace = {\n        'max_depth': 5,\n        'learning_rate': 0.15, \n        'n_estimators': 3800,\n        'num_leaves': 4,\n        'max_bin': 150,\n        'bagging_fraction': 0.7434141086967856,\n        'bagging_freq': 14,\n        'objective': 'regression',\n        'tree_
learner': 'feature',\n        'boosting_type' : 'dart',\n        'verbosity': -1,\n    }\n"
```

In [14]:

```python
# best for xgboost
# best loss: 0.11105198249739094
'''
space = {
        'max_depth': 6,
        'max_leaves': 8,
        'learning_rate': 0.026,
        'n_estimators': 3200,
        'subsample': hp.uniform('subsample', 0, 1),
        'colsample_bytree': hp.uniform('colsample_bytree', 0, 1),
        'min_child_weight': 7,
        'objective': hp.choice('objective', ['reg:squarederror', 'reg:pseudohubererror']),
        'alpha': hp.uniform('alpha', 0, 1),
    }
'''

# learning_rate=0.008073243888388325, max_depth=3, n_estimators=5300, subsample=0.6982127263866671, colsample_bytree=0.09568028958854619, max_leaves=24, alpha=0.30714547261947767, min_child_weight=3
```

Out[14]:

```
"\nspace = {\n        'max_depth': 6,\n        'max_leaves': 8,\n        'learning_rate': 0.026, \n        'n_estimators': 3200, \n        'subsample': hp.uniform('subsample', 0, 1),\n        'colsample_bytree': hp.uniform('colsample_bytree', 0, 1),\n        'min_child_weight': 7,
\n        'objective': hp.choice('objective', ['reg:squarederror', 'reg:pseudohubererror']),\n        'alpha': hp.uniform('alpha', 0, 1),\n    }\n"
```