stacked-regressions

March 1, 2022

1 House Prices - Advanced Regression Techniques

1.1 Predict sales prices and practice feature engineering, RFs, and gradient boosting

1.1.1 Kaggle Prediction Competition

[1]:
```
#import some necessary librairies

import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
color = sns.color_palette()
sns.set_style('darkgrid')
import warnings
def ignore_warn(*args, **kwargs):
    pass
warnings.warn = ignore_warn


from scipy import stats
from scipy.stats import norm, skew


pd.set_option('display.float_format', lambda x: '{:.3f}'.format(x))
```

[2]:
```
#import train and test datasets in  pandas dataframe

train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

[3]:
```
##display the first five rows of the train dataset.
train.head(5)
```

[3]:
```
   Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0   1          60       RL       65.000     8450   Pave   NaN      Reg
```

```
1   2          20        RL        80.000     9600   Pave   NaN       Reg
2   3          60        RL        68.000    11250   Pave   NaN       IR1
3   4          70        RL        60.000     9550   Pave   NaN       IR1
4   5          60        RL        84.000    14260   Pave   NaN       IR1

  LandContour Utilities  … PoolArea PoolQC Fence MiscFeature MiscVal MoSold  \
0         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
1         Lvl    AllPub  …        0    NaN   NaN         NaN       0      5
2         Lvl    AllPub  …        0    NaN   NaN         NaN       0      9
3         Lvl    AllPub  …        0    NaN   NaN         NaN       0      2
4         Lvl    AllPub  …        0    NaN   NaN         NaN       0     12

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

[5 rows x 81 columns]
```

[4]:
```
##display the first five rows of the test dataset.
test.head(5)
```

[4]:
```
     Id  MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
0  1461          20       RH       80.000    11622   Pave   NaN      Reg
1  1462          20       RL       81.000    14267   Pave   NaN      IR1
2  1463          60       RL       74.000    13830   Pave   NaN      IR1
3  1464          60       RL       78.000     9978   Pave   NaN      IR1
4  1465         120       RL       43.000     5005   Pave   NaN      IR1

  LandContour Utilities  … ScreenPorch PoolArea PoolQC  Fence MiscFeature  \
0         Lvl    AllPub  …         120        0    NaN  MnPrv         NaN
1         Lvl    AllPub  …           0        0    NaN    NaN        Gar2
2         Lvl    AllPub  …           0        0    NaN  MnPrv         NaN
3         Lvl    AllPub  …           0        0    NaN    NaN         NaN
4         HLS    AllPub  …         144        0    NaN    NaN         NaN

  MiscVal MoSold  YrSold  SaleType  SaleCondition
0       0      6    2010        WD         Normal
1   12500      6    2010        WD         Normal
2       0      3    2010        WD         Normal
3       0      6    2010        WD         Normal
4       0      1    2010        WD         Normal

[5 rows x 80 columns]
```

[5]:
```
#check the numbers of samples and features
print("The train data size before dropping Id feature is : {} ".format(train.
↪shape))
print("The test data size before dropping Id feature is : {} ".format(test.
↪shape))

#Save the 'Id' column
train_ID = train['Id']
test_ID = test['Id']

#Now drop the  'Id' colum since it's unnecessary for  the prediction process.
train.drop("Id", axis = 1, inplace = True)
test.drop("Id", axis = 1, inplace = True)

#check again the data size after dropping the 'Id' variable
print("\nThe train data size after dropping Id feature is : {} ".format(train.
↪shape))
print("The test data size after dropping Id feature is : {} ".format(test.
↪shape))
```

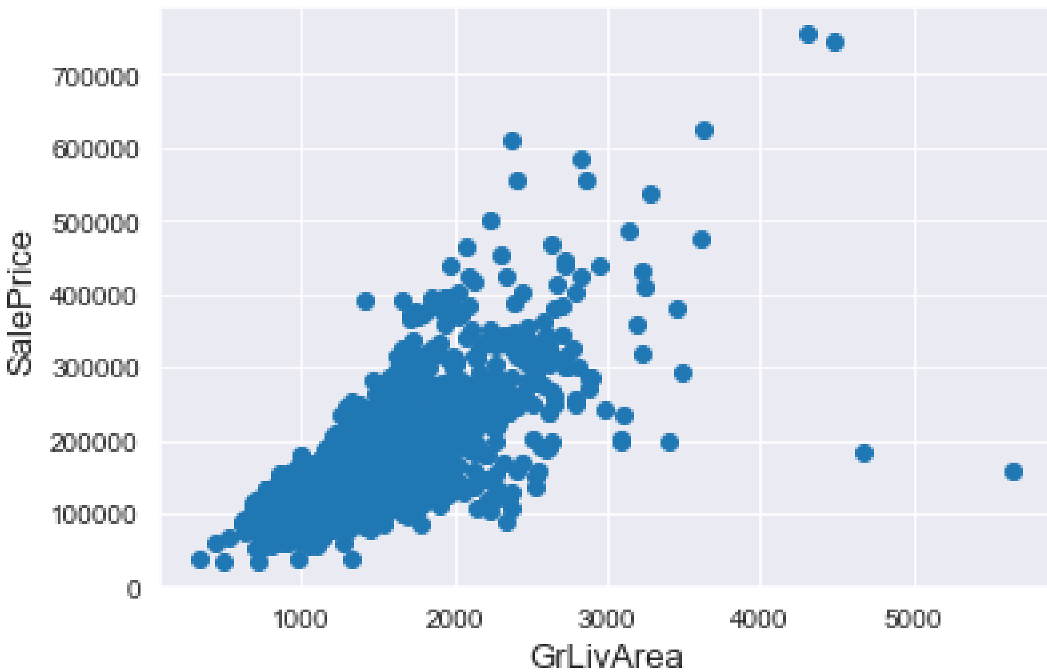
```
The train data size before dropping Id feature is : (1460, 81)
The test data size before dropping Id feature is : (1459, 80)

The train data size after dropping Id feature is : (1460, 80)
The test data size after dropping Id feature is : (1459, 79)
```

2 Data Processing

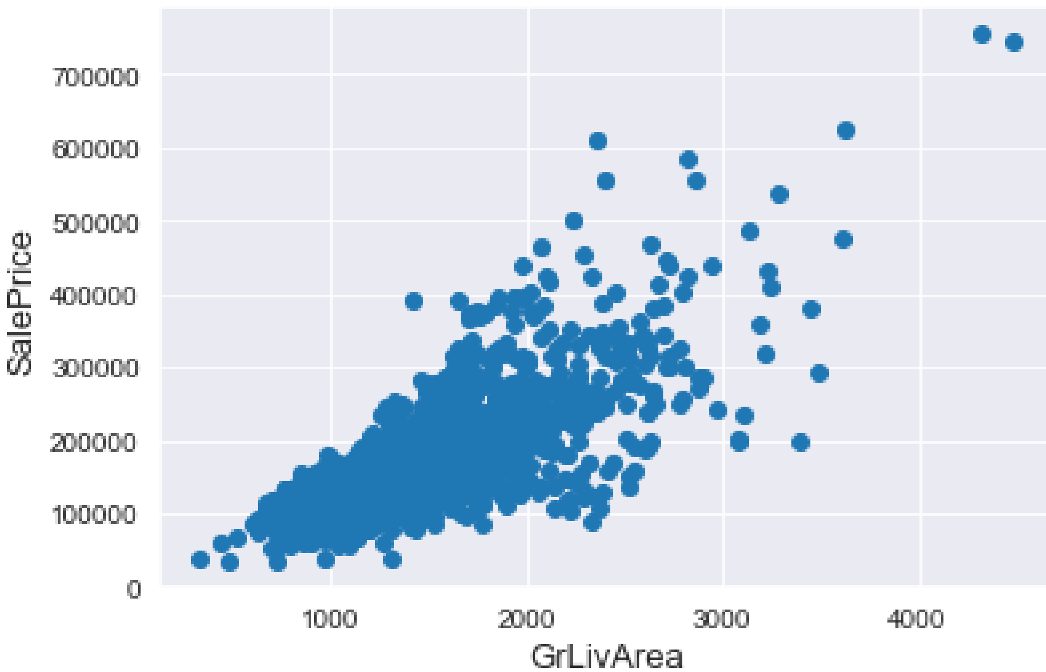
2.1 Outliers

Let's explore these outliers

[6]:
```
fig, ax = plt.subplots()
ax.scatter(x = train['GrLivArea'], y = train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



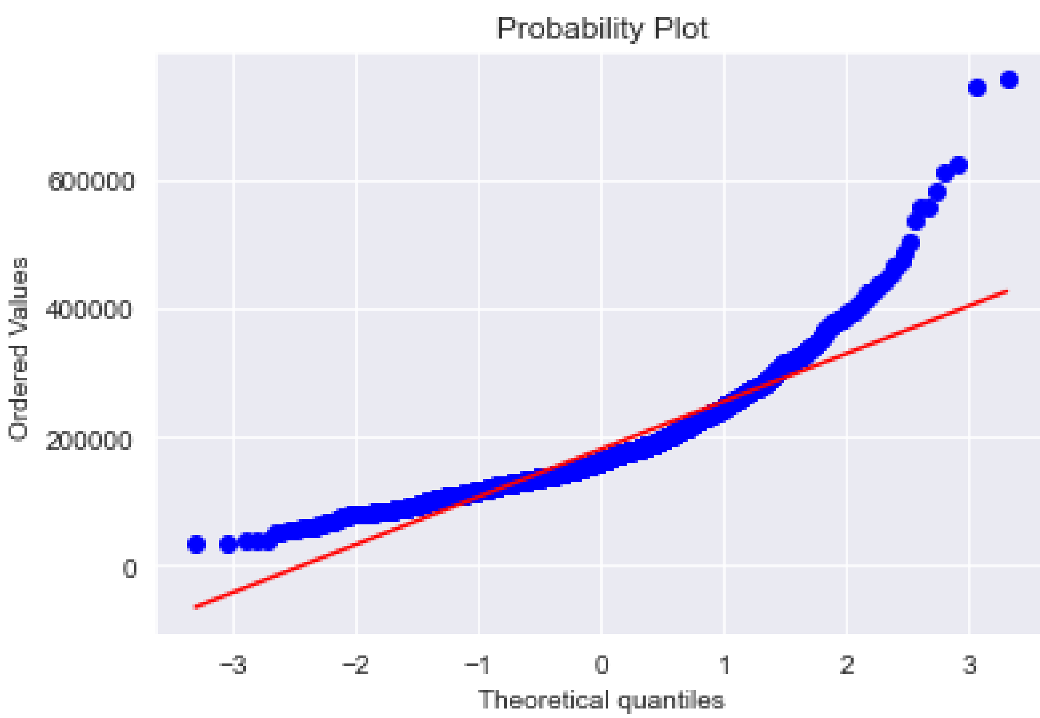
[7]:
```
#Deleting outliers
train = train.drop(train[(train['GrLivArea']>4000) &␣
↪(train['SalePrice']<300000)].index)

#Check the graphic again
fig, ax = plt.subplots()
ax.scatter(train['GrLivArea'], train['SalePrice'])
plt.ylabel('SalePrice', fontsize=13)
plt.xlabel('GrLivArea', fontsize=13)
plt.show()
```



SalePrice is the variable we need to predict. So let's do some analysis on this variable first.

[8]:
```
sns.distplot(train['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,␣
↪sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

#Get also the QQ-plot
fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```

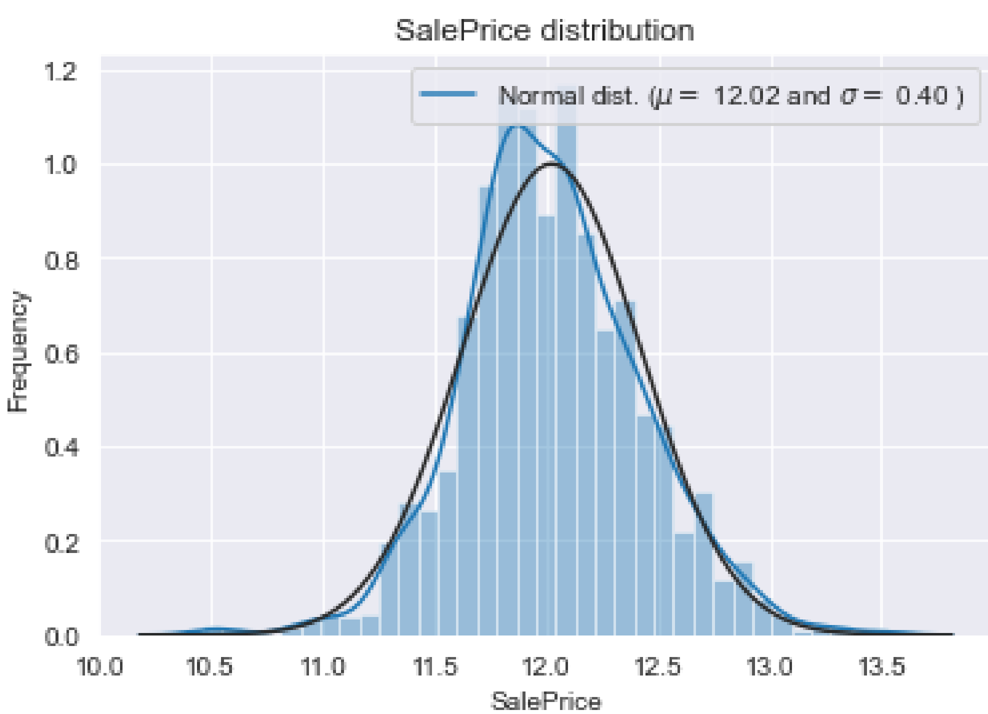
```
mu = 180932.92 and sigma = 79467.79
```

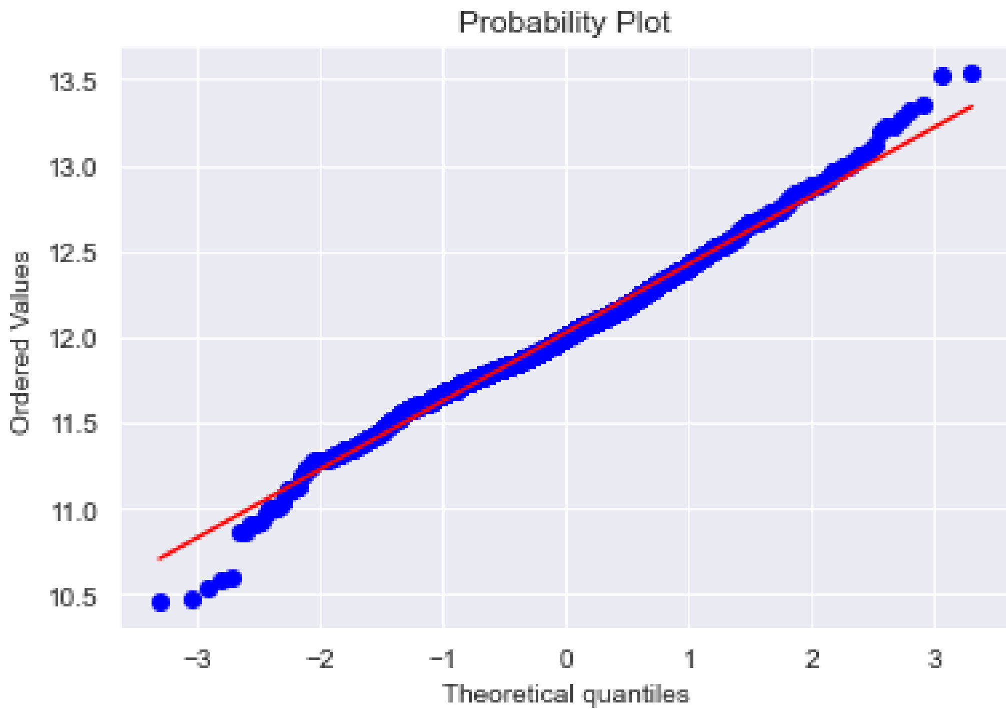


Log-transformation of the target variable

[9]:
```
#We use the numpy fuction log1p which  applies log(1+x) to all elements of the␣
↪column
train["SalePrice"] = np.log1p(train["SalePrice"])

#Check the new distribution
sns.distplot(train['SalePrice'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(train['SalePrice'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,␣
↪sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')

#Get also the QQ-plot
fig = plt.figure()
res = stats.probplot(train['SalePrice'], plot=plt)
plt.show()
```

```
mu = 12.02 and sigma = 0.40
```





The skew seems now corrected and the data appears more normally distributed.

2.2 Features engineering

[10]:
```
ntrain = train.shape[0]
ntest = test.shape[0]
y_train = train.SalePrice.values
all_data = pd.concat((train, test)).reset_index(drop=True)
all_data.drop(['SalePrice'], axis=1, inplace=True)
print("all_data size is : {}".format(all_data.shape))
```

```
all_data size is : (2917, 79)
```

2.2.1 Missing Data

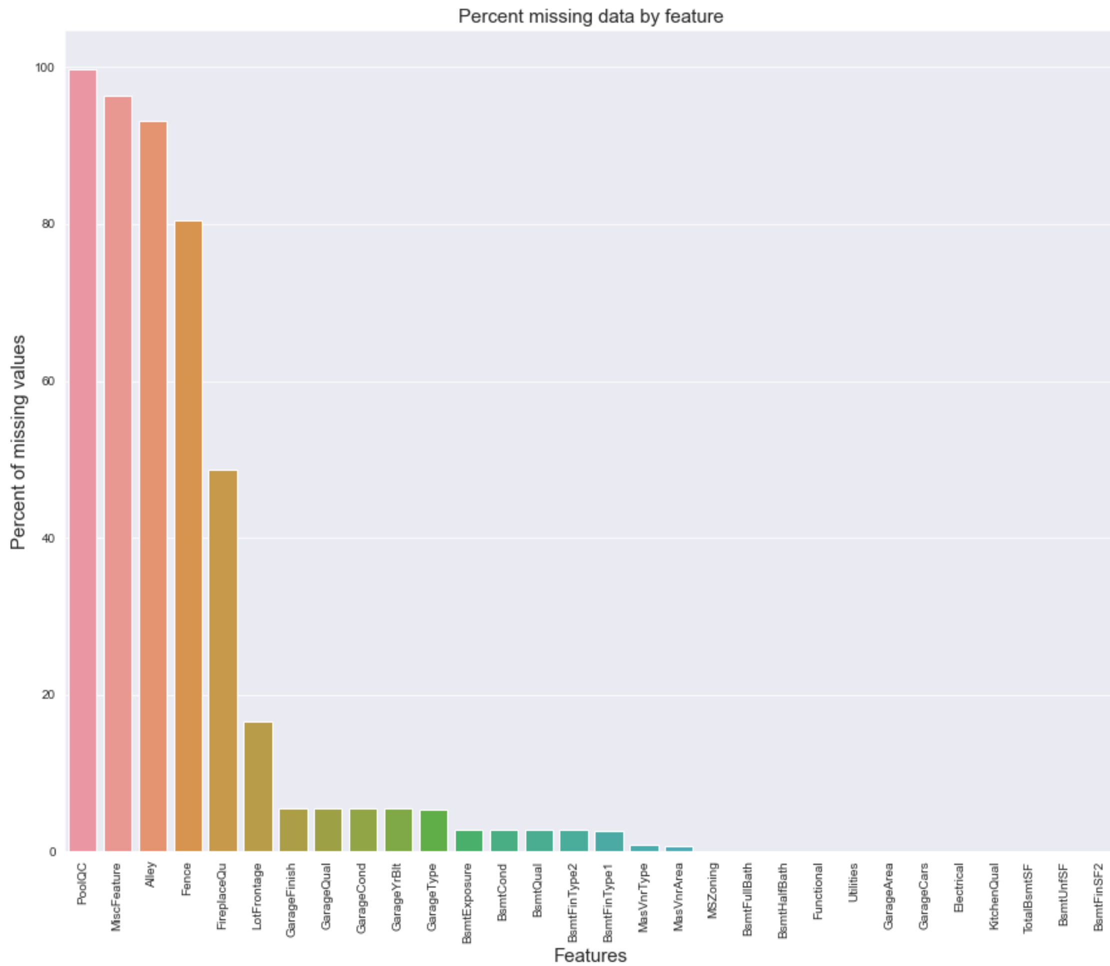
[11]:
```
all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).
 ↪sort_values(ascending=False)[:30]
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head(20)
```

[11]:

	Missing Ratio
PoolQC	99.691
MiscFeature	96.400
Alley	93.212
Fence	80.425
FireplaceQu	48.680
LotFrontage	16.661
GarageFinish	5.451
GarageQual	5.451
GarageCond	5.451
GarageYrBlt	5.451
GarageType	5.382
BsmtExposure	2.811
BsmtCond	2.811
BsmtQual	2.777
BsmtFinType2	2.743
BsmtFinType1	2.708
MasVnrType	0.823
MasVnrArea	0.788
MSZoning	0.137
BsmtFullBath	0.069

[12]:
```python
f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='90')
sns.barplot(x=all_data_na.index, y=all_data_na)
plt.xlabel('Features', fontsize=15)
plt.ylabel('Percent of missing values', fontsize=15)
plt.title('Percent missing data by feature', fontsize=15)
```

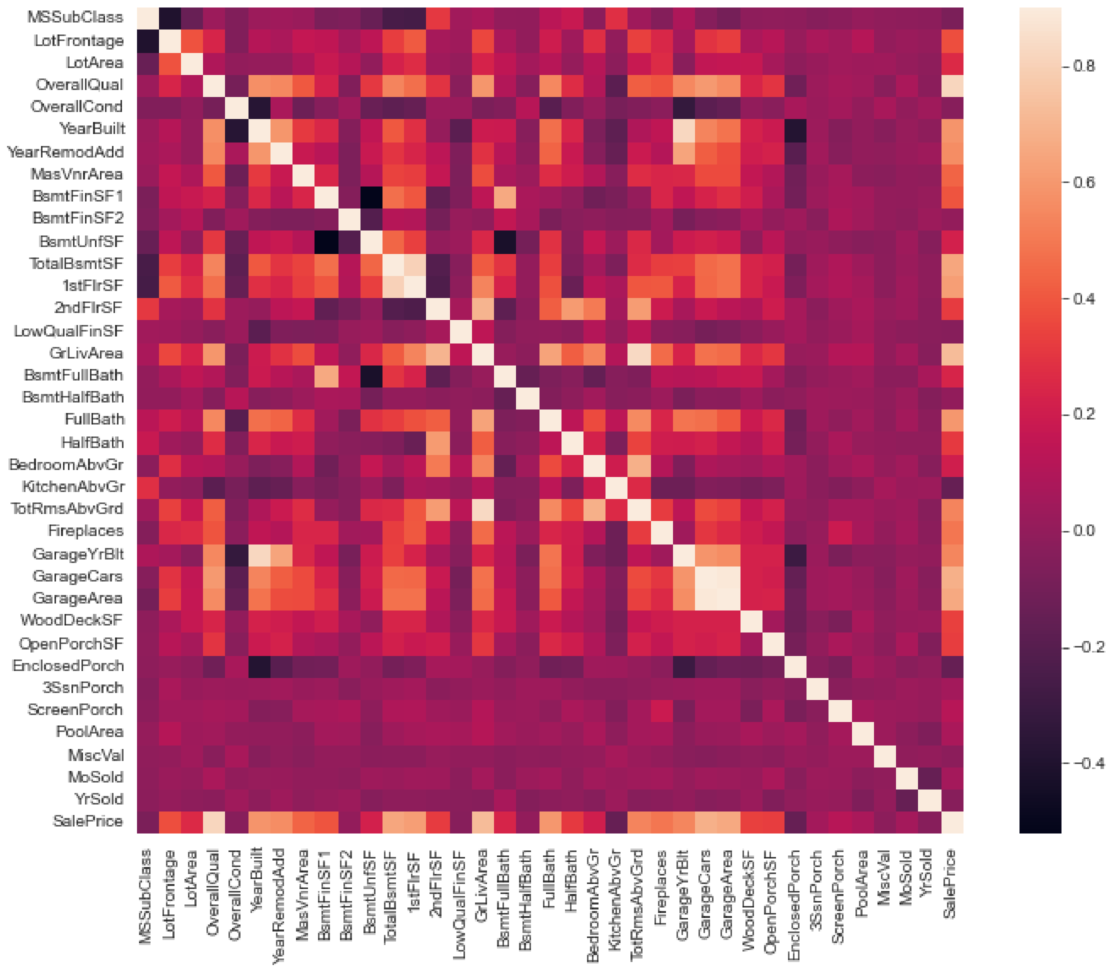
[12]: Text(0.5, 1.0, 'Percent missing data by feature')



Data Correlation

[13]:
```
#Correlation map to see how features are correlated with SalePrice
corrmat = train.corr()
plt.subplots(figsize=(12,9))
sns.heatmap(corrmat, vmax=0.9, square=True)
```

[13]: <AxesSubplot:>



2.2.2 Imputing missing values

We impute them by proceeding sequentially through features with missing values

[14]:
```
all_data["PoolQC"] = all_data["PoolQC"].fillna("None")
```

[15]:
```
all_data["Alley"] = all_data["Alley"].fillna("None")
```

[16]:
```
all_data["Fence"] = all_data["Fence"].fillna("None")
```

[17]:
```
all_data["FireplaceQu"] = all_data["FireplaceQu"].fillna("None")
```

[18]:
```
#Group by neighborhood and fill in missing value by the median LotFrontage of␣
↪all the neighborhood
all_data["LotFrontage"] = all_data.groupby("Neighborhood")["LotFrontage"].
↪transform(
    lambda x: x.fillna(x.median()))
```

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

```
[20]: for col in ('GarageYrBlt', 'GarageArea', 'GarageCars'):
          all_data[col] = all_data[col].fillna(0)
```

```
[21]: for col in ('BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF','TotalBsmtSF',␣
       ↪'BsmtFullBath', 'BsmtHalfBath'):
          all_data[col] = all_data[col].fillna(0)
```

```
[22]: for col in ('BsmtQual', 'BsmtCond', 'BsmtExposure', 'BsmtFinType1',␣
       ↪'BsmtFinType2'):
          all_data[col] = all_data[col].fillna('None')
```

```
[23]: all_data["MasVnrType"] = all_data["MasVnrType"].fillna("None")
      all_data["MasVnrArea"] = all_data["MasVnrArea"].fillna(0)
```

```
[24]: all_data['MSZoning'] = all_data['MSZoning'].fillna(all_data['MSZoning'].
       ↪mode()[0])
```

```
[25]: all_data = all_data.drop(['Utilities'], axis=1)
```

```
[26]: all_data["Functional"] = all_data["Functional"].fillna("Typ")
```

```
[27]: all_data['Electrical'] = all_data['Electrical'].fillna(all_data['Electrical'].
       ↪mode()[0])
```

```
[28]: all_data['KitchenQual'] = all_data['KitchenQual'].
       ↪fillna(all_data['KitchenQual'].mode()[0])
```

```
[29]: all_data['Exterior1st'] = all_data['Exterior1st'].
       ↪fillna(all_data['Exterior1st'].mode()[0])
      all_data['Exterior2nd'] = all_data['Exterior2nd'].
       ↪fillna(all_data['Exterior2nd'].mode()[0])
```

```
[30]: all_data['SaleType'] = all_data['SaleType'].fillna(all_data['SaleType'].
       ↪mode()[0])
```

```
[31]: all_data['MSSubClass'] = all_data['MSSubClass'].fillna("None")
```

Is there any remaining missing value ?

```
[32]: #Check remaining missing values if any
      all_data_na = (all_data.isnull().sum() / len(all_data)) * 100
      all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).
       ↪sort_values(ascending=False)
      missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
      missing_data.head()
```

[32]:
```
             Missing Ratio
MiscFeature         96.400
```

It remains no missing value.

2.2.3 More features engeneering

Transforming some numerical variables that are really categorical

[33]:
```python
#MSSubClass=The building class
all_data['MSSubClass'] = all_data['MSSubClass'].apply(str)


#Changing OverallCond into a categorical variable
all_data['OverallCond'] = all_data['OverallCond'].astype(str)


#Year and month sold are transformed into categorical features.
all_data['YrSold'] = all_data['YrSold'].astype(str)
all_data['MoSold'] = all_data['MoSold'].astype(str)
```

Label Encoding some categorical variables that may contain information in their ordering set

[34]:
```python
from sklearn.preprocessing import LabelEncoder
cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',
        'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual',␣
 ↪'BsmtFinType1',
        'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish',␣
 ↪'LandSlope',
        'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir',␣
 ↪'MSSubClass', 'OverallCond',
        'YrSold', 'MoSold')
# process columns, apply LabelEncoder to categorical features
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(all_data[c].values))
    all_data[c] = lbl.transform(list(all_data[c].values))

# shape
print('Shape all_data: {}'.format(all_data.shape))
```

```
Shape all_data: (2917, 78)
```

Adding one more important feature

[35]:
```python
# Adding total sqfootage feature
all_data['TotalSF'] = all_data['TotalBsmtSF'] + all_data['1stFlrSF'] +␣
 ↪all_data['2ndFlrSF']
```

Skewed features

[36]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = all_data[numeric_feats].apply(lambda x: skew(x.dropna())).
↪sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

[36]:

	Skew
MiscVal	21.940
PoolArea	17.689
LotArea	13.109
LowQualFinSF	12.085
3SsnPorch	11.372
LandSlope	4.973
KitchenAbvGr	4.301
BsmtFinSF2	4.145
EnclosedPorch	4.002
ScreenPorch	3.945

[37]:
```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform".
↪format(skewness.shape[0]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    #all_data[feat] += 1
    all_data[feat] = boxcox1p(all_data[feat], lam)

#all_data[skewed_features] = np.log1p(all_data[skewed_features])
```

```
There are 59 skewed numerical features to Box Cox transform
```

Getting dummy categorical features

[38]:
```
all_data = pd.get_dummies(all_data)
print(all_data.shape)
```

```
(2917, 219)
```

[39]:
```python
train = all_data[:ntrain]
test = all_data[ntrain:]
```

3 Modelling

Import librairies

[40]:
```python
from sklearn.linear_model import ElasticNet, Lasso,  BayesianRidge, LassoLarsIC
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.kernel_ridge import KernelRidge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
import xgboost as xgb
import lightgbm as lgb
```

Define a cross validation strategy

We use the **cross_val_score** function of Sklearn. However this function has not a shuffle attribut, we add then one line of code, in order to shuffle the dataset prior to cross-validation

[41]:
```python
#Validation function
n_folds = 5

def rmsle_cv(model):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(train.
 ↪values)
    rmse= np.sqrt(-cross_val_score(model, train.values, y_train,␣
 ↪scoring="neg_mean_squared_error", cv = kf))
    return(rmse)
```

##Base models

- **LASSO Regression** :

This model may be very sensitive to outliers. So we need to made it more robust on them. For that we use the sklearn's **Robustscaler()** method on pipeline

[42]:
```python
lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
```

- **Elastic Net Regression** :

again made robust to outliers

[43]:
```python
ENet = make_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1_ratio=.9,␣
 ↪random_state=3))
```

- **Kernel Ridge Regression** :

[44]:
```
KRR = KernelRidge(alpha=0.6, kernel='polynomial', degree=2, coef0=2.5)
```

- **Gradient Boosting Regression** :

With **huber** loss that makes it robust to outliers

[45]:
```
GBoost = GradientBoostingRegressor(n_estimators=5000, learning_rate=0.05,
                                   max_depth=4, max_features='sqrt',
                                   min_samples_leaf=15, min_samples_split=10,
                                   loss='huber', random_state =5)
```

- **XGBoost** :

[46]:
```
model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=5000,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1)
```

- **LightGBM** :

[47]:
```
model_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                              learning_rate=0.01, n_estimators=720,
                              max_bin = 55, bagging_fraction = 0.8,
                              bagging_freq = 5, feature_fraction = 0.2319,
                              feature_fraction_seed=9, bagging_seed=9,
                              min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
```

###Base models scores

Let's see how these base models perform on the data by evaluating the cross-validation rmsle error

[48]:
```
score = rmsle_cv(lasso)
print("\nLasso score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Lasso score: 0.1115 (0.0074)
```

[49]:
```
score = rmsle_cv(ENet)
print("ElasticNet score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
ElasticNet score: 0.1116 (0.0074)
```

[50]:
```
score = rmsle_cv(KRR)
print("Kernel Ridge score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
Kernel Ridge score: 0.1153 (0.0076)
```

[51]:
```
score = rmsle_cv(GBoost)
print("Gradient Boosting score: {:.4f} ({:.4f})\n".format(score.mean(), score.
↪std()))
```

```
Gradient Boosting score: 0.1180 (0.0084)
```

[52]:
```
score = rmsle_cv(model_xgb)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
[17:13:30] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[17:13:45] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[17:14:00] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[17:14:16] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.1/src/learner.cc:576:
```

```
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


[17:14:31] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


Xgboost score: 0.1165 (0.0069)
```

[53]:

```python
score = rmsle_cv(model_lgb)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score.mean(), score.std()))
```

```
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
```

```
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
LGBM score: 0.1216 (0.0069)
```

Averaged base models class

[54]:
```
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models

    # we define clones of the original models to fit the data in
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X, y)

        return self
```

```
    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)
```

Averaged base models score

We just average four models here **ENet, GBoost, KRR and lasso**. Of course we could easily add more models in the mix.

[55]:
```
averaged_models = AveragingModels(models = (ENet, GBoost, KRR, lasso))

score = rmsle_cv(averaged_models)
print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(),␣
 ↪score.std()))
```

```
 Averaged base models score: 0.1088 (0.0077)
```

Stacking averaged Models Class

[56]:
```
class StackingAveragedModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, base_models, meta_model, n_folds=5):
        self.base_models = base_models
        self.meta_model = meta_model
        self.n_folds = n_folds

    # We again fit the data on clones of the original models
    def fit(self, X, y):
        self.base_models_ = [list() for x in self.base_models]
        self.meta_model_ = clone(self.meta_model)
        kfold = KFold(n_splits=self.n_folds, shuffle=True, random_state=156)

        # Train cloned base models then create out-of-fold predictions
        # that are needed to train the cloned meta-model
        out_of_fold_predictions = np.zeros((X.shape[0], len(self.base_models)))
        for i, model in enumerate(self.base_models):
            for train_index, holdout_index in kfold.split(X, y):
                instance = clone(model)
                self.base_models_[i].append(instance)
                instance.fit(X[train_index], y[train_index])
                y_pred = instance.predict(X[holdout_index])
                out_of_fold_predictions[holdout_index, i] = y_pred

        # Now train the cloned  meta-model using the out-of-fold predictions as␣
 ↪new feature
```

```
        self.meta_model_.fit(out_of_fold_predictions, y)
        return self

    #Do the predictions of all base models on the test data and use the␣
↪averaged predictions as
    #meta-features for the final prediction which is done by the meta-model
    def predict(self, X):
        meta_features = np.column_stack([
            np.column_stack([model.predict(X) for model in base_models]).
↪mean(axis=1)
            for base_models in self.base_models_ ])
        return self.meta_model_.predict(meta_features)
```

Stacking Averaged models Score

To make the two approaches comparable (by using the same number of models) , we just average **Enet KRR and Gboost**, then we add **lasso as meta-model**.

[57]:
```
stacked_averaged_models = StackingAveragedModels(base_models = (ENet, GBoost,␣
↪KRR),
                                                 meta_model = lasso)

score = rmsle_cv(stacked_averaged_models)
print("Stacking Averaged models score: {:.4f} ({:.4f})".format(score.mean(),␣
↪score.std()))
```

```
Stacking Averaged models score: 0.1085 (0.0071)
```

We get again a better score by adding a meta learner

3.1 Ensembling StackedRegressor, XGBoost and LightGBM

We add **XGBoost and LightGBM** to the** StackedRegressor** defined previously.

[58]:
```
def rmsle(y, y_pred):
    return np.sqrt(mean_squared_error(y, y_pred))
```

StackedRegressor:

[59]:
```
stacked_averaged_models.fit(train.values, y_train)
stacked_train_pred = stacked_averaged_models.predict(train.values)
stacked_pred = np.expm1(stacked_averaged_models.predict(test.values))
print(rmsle(y_train, stacked_train_pred))
```

```
0.07844715135052523
```

XGBoost:

[60]:
```
model_xgb.fit(train, y_train)
xgb_train_pred = model_xgb.predict(train)
xgb_pred = np.expm1(model_xgb.predict(test))
```

```
print(rmsle(y_train, xgb_train_pred))
```

```
[17:25:05] WARNING: C:/Users/Administrator/workspace/xgboost-
win64_release_1.5.1/src/learner.cc:576:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language
bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually
being used
  but getting flagged wrongly here. Please open an issue if you find any such
cases.


0.07508911481331615
```

LightGBM:

[61]:
```
model_lgb.fit(train, y_train)
lgb_train_pred = model_lgb.predict(train)
lgb_pred = np.expm1(model_lgb.predict(test.values))
print(rmsle(y_train, lgb_train_pred))
```

```
[LightGBM] [Warning] feature_fraction is set=0.2319, colsample_bytree=1.0 will
be ignored. Current value: feature_fraction=0.2319
[LightGBM] [Warning] min_data_in_leaf is set=6, min_child_samples=20 will be
ignored. Current value: min_data_in_leaf=6
[LightGBM] [Warning] min_sum_hessian_in_leaf is set=11, min_child_weight=0.001
will be ignored. Current value: min_sum_hessian_in_leaf=11
[LightGBM] [Warning] bagging_fraction is set=0.8, subsample=1.0 will be ignored.
Current value: bagging_fraction=0.8
[LightGBM] [Warning] bagging_freq is set=5, subsample_freq=0 will be ignored.
Current value: bagging_freq=5
0.10497128693466415
```

[62]:
```
'''RMSE on the entire Train data when averaging'''

print('RMSLE score on train data:')
print(rmsle(y_train,stacked_train_pred*0.70 +
               xgb_train_pred*0.15 + lgb_train_pred*0.15 ))
```

```
RMSLE score on train data:
0.07927164185019933
```

Ensemble prediction:

[63]:
```
ensemble = stacked_pred*0.2 + xgb_pred*0.6 + lgb_pred*0.2
```

Submission

[64]:

```
sub = pd.DataFrame()
sub['Id'] = test_ID
sub['SalePrice'] = ensemble
sub.to_csv('submission.csv',index=False)
```