# Table of Contents ¶



In [1]:

```
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score, cross_val_predict
from scipy.stats import shapiro
from scipy import stats
import os
print(os.listdir("house-prices-advanced-regression-techniques/"))
```

```
['test.csv', 'data_description.txt', 'train.csv', 'sample_submission.c
sv']
```

In [2]:

```
df_train = pd.read_csv("house-prices-advanced-regression-techniques/train.csv")
```

# 1. Preprocess Data

## 1.1 Data formating

'MSSubClass' - Identifies the type of dwelling involved in the sale. It is categorical feature despite Integer type. Before making one-hot encoding, use this feature to fill missed values in "Lot Frontage" and "BsmUnfSF", because seeing on boxplot we can see statistically significant difference between boxplots.

In [3]:

```
df_train.MSSubClass.isnull().sum()
```

Out[3]:

```
0
```

In [4]:

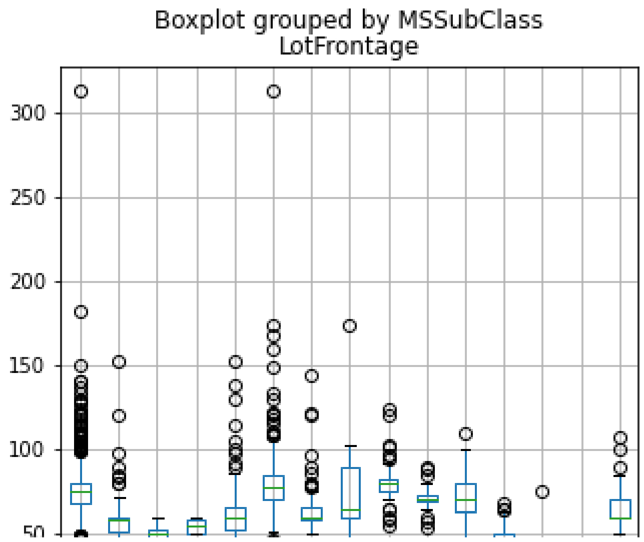
```
def fill_missing_with_mode_group(df, column):#разные/равные медианы в test & train
    groups = df["MSSubClass"].unique()
    for group in groups:
        median_in_group = df[df["MSSubClass"] == group][column].median()
        df.loc[df["MSSubClass"] == group, column] = df.loc[df["MSSubClass"] == grou


fill_missing_with_mode_group(df_train, 'LotFrontage')
fill_missing_with_mode_group(df_train, 'BsmtUnfSF')
```

In [5]:

```
for item in ['LotFrontage',"BsmtUnfSF"]:
    display(df_train.boxplot(item, by="MSSubClass",figsize=(5,5)))
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb7093dd850>

<matplotlib.axes._subplots.AxesSubplot at 0x7fb707345c40>



In [6]:

```
def formating_1(data):
    data = pd.concat([data, pd.get_dummies(data['MSSubClass'])], axis=1).drop('MSSu
    data.rename(columns={20:"1-STORY 1946 & NEWER ALL STYLES",
                         30:"1-STORY 1945 & OLDER",
                         40:"1-STORY W/FINISHED ATTIC ALL AGES",
                         45:"1-1/2 STORY - UNFINISHED ALL AGES",
                         50:"1-1/2 STORY FINISHED ALL AGES",
                         60:"2-STORY 1946 & NEWER",
                         70:"2-STORY 1945 & OLDER",
                         75:"2-1/2 STORY ALL AGES",
                         80:"SPLIT OR MULTI-LEVEL",
                         85:"SPLIT FOYER",
                         90:"DUPLEX - ALL STYLES AND AGES",
                         120:"1-STORY PUD (Planned Unit Development) - 1946 & NEWER
                         150:"1-1/2 STORY PUD - ALL AGES",
                         160:"2-STORY PUD - 1946 & NEWER",
                         180:"PUD - MULTILEVEL - INCL SPLIT LEV/FOYER",
                         190:"2 FAMILY CONVERSION - ALL STYLES AND AGES"},
                inplace=True)
    return(data)

df_train = formating_1(df_train)
```

Features 'Condition1' & 'Condition2' have the same unique values.

In [7]:

```
pd.options.display.max_columns = None

def formating_2(data):
    data = pd.concat([data, pd.get_dummies(data['Condition1'])], axis=1).drop('Cond
    uniq_val = data['Condition2'].unique()
    for val in uniq_val:
        data.loc[data['Condition2']==val, val] = 1
    data.drop('Condition2', axis=1, inplace=True)
    return(data)

df_train = formating_2(df_train)

df_train[['Artery','Feedr','Norm','PosA','PosN','RRAe','RRAn','RRNe']].head(2)
```

Out[7]:

| | Artery | Feedr | Norm | PosA | PosN | RRAe | RRAn | RRNe |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

In [8]:

```
df_train.shape
```

Out[8]:

```
(1460, 102)
```

Formatting qualitative features into numeric type

In [9]:

```
def convert_quality_into_numeric(data):
    data.ExterQual = data.ExterQual.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})
    data.ExterCond = data.ExterCond.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})
    data.BsmtQual = data.BsmtQual.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})#, n
    data.BsmtCond = data.BsmtCond.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})#, n
    data.BsmtExposure = data.BsmtExposure.map({'Gd':4,'Av':3, 'Mn':2, 'No':1})#, np

    data.BsmtFinType1 = data.BsmtFinType1.map({'GLQ':6, 'ALQ':5, 'BLQ':4, 'Rec':3,
    data.BsmtFinType2 = data.BsmtFinType2.map({'GLQ':6, 'ALQ':5, 'BLQ':4, 'Rec':3,

    data.HeatingQC = data.HeatingQC.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})
    data.CentralAir = data.CentralAir.map({"N":0, "Y":1})
    data.KitchenQual = data.KitchenQual.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1
    data.FireplaceQu = data.FireplaceQu.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1
    data.GarageFinish = data.GarageFinish.map({'Fin':3, 'RFn':2, 'Unf':1})#, np.nan
    data.GarageQual = data.GarageQual.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})
    data.GarageCond = data.GarageCond.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})
    data.PavedDrive = data.PavedDrive.map({'Y':3, 'P':2, 'N':1})
    data.PoolQC = data.PoolQC.map({'Ex':5, 'Gd':4, 'TA':3, 'Fa':2, 'Po':1})#, np.na

convert_quality_into_numeric(df_train)
```

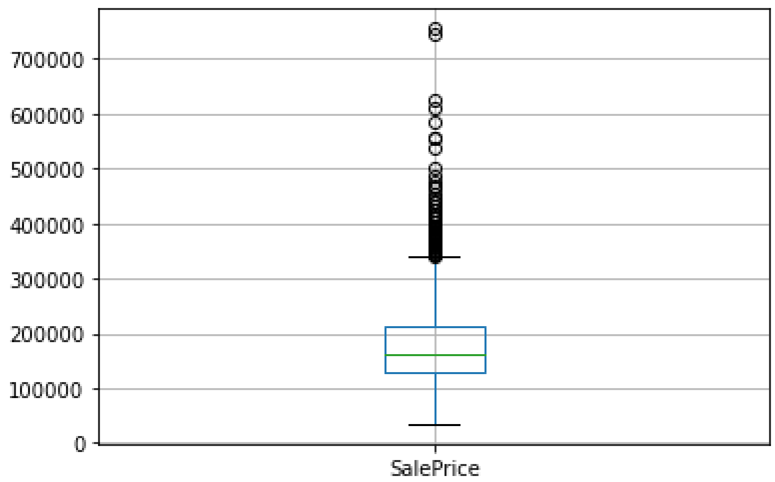
## 1.2 Outliers

In [10]:

```
df_train.boxplot('SalePrice')
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb7093dd8e0>
```



In [11]:

```
outliers_threshold = df_train.SalePrice.quantile(0.75) + 1.5 * (df_train.SalePrice.

print(f"upper outliers threshold: {outliers_threshold},\n# outliers: {df_train.loc[
```

```
upper outliers threshold: 340037.5,
# outliers: 61
```

Consider 61 observations, which are determined by outliers from the analysis of the boxplot.

In [12]:

```
def describe_outliers(threshold, MSZoning='none'):
    print('outliers statistics:')
    display(df_train.loc[(df_train.SalePrice > threshold) & (df_train.MSZoning == M
    print('Not outliers statistics: ')
    display(df_train.loc[(df_train.SalePrice <= threshold) & (df_train.MSZoning == 
    print('Observation with max price:')
    display(df_train.loc[(df_train.SalePrice == max(df_train.SalePrice)) & (df_trai
```

Рассмотрим "выбросы" по группам.

In [13]:

```
df_train.groupby("MSZoning").SalePrice.agg(["count","min","median","max"])
```

Out[13]:

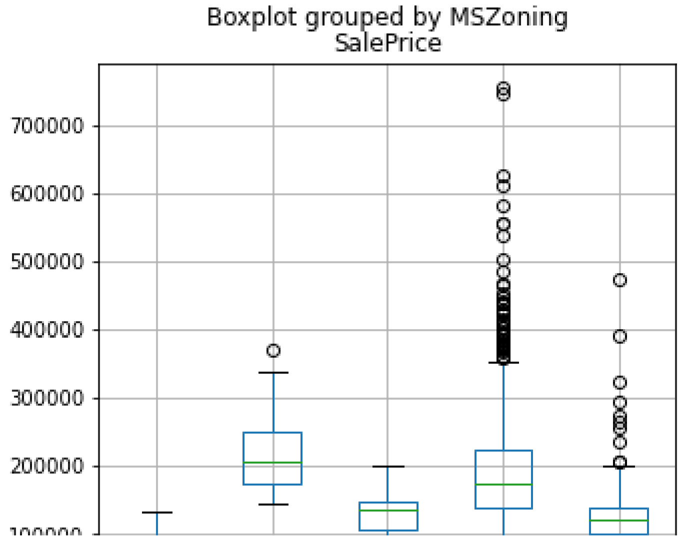
| | count | min | median | max |
|---|---|---|---|---|
| **MSZoning** | | | | |
| **C (all)** | 10 | 34900 | 74700 | 133900 |
| **FV** | 65 | 144152 | 205950 | 370878 |
| **RH** | 16 | 76000 | 136500 | 200000 |
| **RL** | 1151 | 39300 | 174000 | 755000 |
| **RM** | 218 | 37900 | 120500 | 475000 |

In [16]:

```
df_train.boxplot(column="SalePrice", by="MSZoning",figsize=(5,5))
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb706d6bbe0>
```




In [17]:

```
def threshold_detector(MSZoning_type):
    IQR = df_train[df_train["MSZoning"] == MSZoning_type].SalePrice.quantile(.75) -
    thresh_value = df_train[df_train["MSZoning"] == MSZoning_type].SalePrice.quanti
    return(thresh_value)

rl_thresh = threshold_detector("RL")
rm_thresh = threshold_detector("RM")
```

MSZoning = "RL"

In [18]:

```
describe_outliers(rl_thresh, "RL")
```

```
outliers statistics:
```

| | Id | LotFrontage | LotArea | OverallQual | OverallCond | YearBuilt | YearRem |
|---|---|---|---|---|---|---|---|
| **mean** | 675.940000 | 89.080000 | 20324.000000 | 8.82000 | 5.120000 | 2000.460000 | 2003.: |
| **std** | 399.008844 | 27.229066 | 29707.160623 | 0.84973 | 0.798979 | 15.718454 | 7.! |
| **min** | 54.000000 | 42.000000 | 8089.000000 | 7.00000 | 2.000000 | 1934.000000 | 1965.( |
| **25%** | 357.250000 | 69.000000 | 12393.000000 | 8.00000 | 5.000000 | 2003.000000 | 2003.: |
| **50%** | 635.500000 | 88.000000 | 13792.000000 | 9.00000 | 5.000000 | 2006.000000 | 2006.( |
| **75%** | 965.750000 | 105.000000 | 15097.750000 | 9.00000 | 5.000000 | 2008.000000 | 2008.( |

```
Not outliers statistics:
```

| | Id | LotFrontage | LotArea | OverallQual | OverallCond | YearBuilt | YearRen |
|---|---|---|---|---|---|---|---|
| **mean** | 740.062670 | 73.620799 | 11174.348774 | 6.070845 | 5.554042 | 1974.326067 | 1984 |
| **std** | 421.034251 | 20.645238 | 8981.858533 | 1.277951 | 1.056596 | 25.824160 | 19 |
| **min** | 1.000000 | 24.000000 | 2268.000000 | 1.000000 | 1.000000 | 1875.000000 | 1950 |
| **25%** | 373.000000 | 61.000000 | 8400.000000 | 5.000000 | 5.000000 | 1958.000000 | 1968 |
| **50%** | 754.000000 | 75.000000 | 9880.000000 | 6.000000 | 5.000000 | 1975.000000 | 1992 |
| **75%** | 1103.000000 | 80.000000 | 11900.000000 | 7.000000 | 6.000000 | 1999.000000 | 2003 |

```
Observation with max price:
```

| | Id | LotFrontage | LotArea | OverallQual | OverallCond | YearBuilt | YearRemodAdd | MasV |
|---|---|---|---|---|---|---|---|---|
| **mean** | 692.0 | 104.0 | 21535.0 | 10.0 | 6.0 | 1994.0 | 1995.0 | |

In the case of "Outliers", there is a large std in the LotArea variable (area in sq.m.).

In [19]:

```
rl_outliers = df_train.loc[(df_train.SalePrice > rl_thresh) & (df_train.MSZoning ==
#rl_outliers.head(15)

# OverallQual - [0, 10]
# OverallCond - [0, 10]
```

OverallCond is the same for all non-outliers OverallQual for all> 7.0 [3 quantile not outliers] SalePrice is noticeably more than 75% of non-outliers.

It was decided to determine observations with a small area (<= 13792) as outliers Let's remove outliers.

In [20]:

```
display(rl_outliers[rl_outliers["LotArea"] <= 13792].describe().loc[["count","mean"
rl_outliers = rl_outliers[rl_outliers["LotArea"] <= 13792]
df_train.drop(axis = 0, index=rl_outliers.index, inplace=True)
print(f'number deleted observations: {rl_outliers.shape[0]}')
```

| | LotArea | OverallQual | OverallCond | SalePrice |
|---|---|---|---|---|
| **count** | 25.000000 | 25.00000 | 25.000000 | 25.000000 |
| **mean** | 11974.080000 | 8.84000 | 4.960000 | 408791.680000 |
| **std** | 1545.240648 | 0.85049 | 0.675771 | 54260.760228 |

```
number deleted observations: 25
```

In [21]:

```
#describe_outliers(rm_thresh, "RM")
```

All observations have OverallQual and OverallCond about 75% not outliers. Delete observations, with LotArea <8520

In [22]:

```
rm_outliers = df_train.loc[(df_train.SalePrice > rm_thresh) & (df_train.MSZoning ==
rm_outliers = rm_outliers[rm_outliers["LotArea"] <= 8520]
df_train.drop(axis = 0, index=rm_outliers.index, inplace=True)
print(f'number deleted observations: {rm_outliers.shape[0]}')
```

```
number deleted observations: 5
```

## 1.3 Missing values

In [23]:

```
def missing_ration(data, top=10):
    df_na = round((data.isnull().sum() / len(data)) * 100,2)
    df_na = df_na.drop(df_na[df_na == 0].index).sort_values(ascending=False)
    missing_data = pd.DataFrame(
        {'Missing Ratio' :df_na}
    )
    print(missing_data)

def fill_missing(df, cols, val):
    """ Replace with the supplied val """
    for col in cols:
        df[col] = df[col].fillna(val)

def fill_missing_with_mode(df, cols):
    """ Replace with the mode """
    for col in cols:
        df[col] = df[col].fillna(df[col].mode()[0])

print("\n Train")
missing_ration(df_train)
```

```
 Train
              Missing Ratio
PoolQC                99.51
MiscFeature           96.22
Alley                 93.64
Fence                 80.35
FireplaceQu           48.18
GarageCond             5.66
GarageQual             5.66
GarageFinish           5.66
GarageYrBlt            5.66
GarageType             5.66
BsmtFinType2           2.66
BsmtExposure           2.66
BsmtFinType1           2.59
BsmtCond               2.59
BsmtQual               2.59
MasVnrArea             0.56
MasVnrType             0.56
Electrical             0.07
```

### Replace with "Not exist"

Reading 'data_description.txt' notice that some there are some variables, which contains NA/None and it has meaning. Replace missing values with "Not exist"

In [24]:

```
fill_missing(df_train, ["MiscFeature", "Alley", "Fence",
                        "GarageType","MasVnrType"], "Not exist")
```

### Replace with 0

Missed values of Quantities features replace with 0.

### Replace with mode

Missed values of categorical variables replace with mode, because not many missing values.

In [25]:

```
fill_missing(df_train, ['GarageYrBlt', 'MasVnrArea', 'BsmtHalfBath', 'BsmtFullBath'
                        'BsmtFinSF1', 'GarageArea', 'TotalBsmtSF', 'GarageCars', 'B
                       "PoolQC", "FireplaceQu", "GarageFinish", "GarageQual", "Gara
                        "BsmtCond","BsmtExposure","BsmtFinType1", "BsmtFinType2"],


fill_missing_with_mode(df_train, ['Electrical', 'MSZoning', 'Functional', 'Utilitie
'KitchenQual', 'SaleType'])
```

# 2. Transforms data

## 2.1 Aggregation

In [26]:

```
df_train['TotalSF'] = df_train['TotalBsmtSF'] + df_train['1stFlrSF'] + df_train['2n
```

In [27]:

```
df_train["time_before_remodelation"] = df_train.YearRemodAdd - df_train.YearBuilt
```

## 2.2 Logarithmization of variables

Taking the logarithm of some variables allows their distribution to be closer to normal ('loglist' tuple was created by seeing histograms and shapiro test)

In [28]:

```
def add_logs(df, cols):
    # add Log transformed columns
    for c in cols:
        df[c] = np.log(df[c] + 1)
    return(df)

loglist = ('LotFrontage',
           'LotArea', 'BsmtUnfSF','GrLivArea',
           'BsmtFullBath', 'BsmtHalfBath','TotalSF')

df_train = add_logs(df_train, loglist)
```

In [29]:

```
df_train["SalePrice"] = np.log1p(df_train["SalePrice"])
```

## 2.3 Handle with categorical features

In [30]:

```
def fix_missing_cols(in_train, in_test):
    missing_cols = set(in_train.columns) - set(in_test.columns)
    # Add a missing column in test set with default value equal to 0
    for c in missing_cols:
        in_test[c] = 0
    # Ensure the order of column in the test set is in the same order than in train
    in_test = in_test[in_train.columns]
    return(in_test)

def dummy_encode(in_df_train):
    df_train = in_df_train
    categorical_feats = [
        f for f in df_train.columns if df_train[f].dtype == 'object'
    ]
    print(categorical_feats)
    for f_ in categorical_feats:
        prefix = f_
        df_train = pd.concat([df_train, pd.get_dummies(df_train[f_], prefix=prefix)

    return(df_train)
```

In [31]:

```
df_train = dummy_encode(df_train)
df_train_here = df_train
print("Shape train: %s, test: %s" % (df_train.shape))
```

```
['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilitie
s', 'LotConfig', 'LandSlope', 'Neighborhood', 'BldgType', 'HouseStyl
e', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrTyp
e', 'Foundation', 'Heating', 'Electrical', 'Functional', 'GarageType',
'Fence', 'MiscFeature', 'SaleType', 'SaleCondition']
Shape train: 1430, test: 251
```

## 2.4 Polynomial features

Make polynomial features of variables, which have high correlation coefficient with Dependent Variable

In [32]:

```
def load_poly_features(data, cols):
    print('Loading polynomial features..')
    # Make a new dataframe for polynomial features
    poly_features = data[cols]

    # imputer for handling missing values
    imputer = SimpleImputer(strategy='median')

    # Need to impute missing values
    poly_features = imputer.fit_transform(poly_features)

    # Create the polynomial object with specified degree
    poly_transformer = PolynomialFeatures(degree=3)
    # Train the polynomial features
    poly_features = poly_transformer.fit_transform(poly_features)


    print('Polynomial Features shape: %s' % str(poly_features.shape))

    df_poly_features = pd.DataFrame(poly_features,
                                    columns=poly_transformer.get_feature_names(cols

    df_poly_features['Id'] = data['Id']

    print('Loaded polynomial features')
    return(df_poly_features)
```

In [33]:

```
corr_list = abs(df_train.corr().SalePrice).sort_values(ascending=False)
correlated_cols = corr_list[corr_list > 0.7][1:]
print(correlated_cols)
correlated_cols = correlated_cols.index
```

```
OverallQual    0.803943
TotalSF        0.797013
GrLivArea      0.722385
Name: SalePrice, dtype: float64
```

In [34]:

```
df_train_poly =  load_poly_features(df_train, cols=correlated_cols)
print("Shape train: %s, test: %s" % (df_train_poly.shape))

df_train = df_train.merge(right=df_train_poly.reset_index(), how='left', on='Id')

print("Shape train: %s, test: %s" % (df_train.shape))
```

```
Loading polynomial features..
Polynomial Features shape: (1430, 20)
Loaded polynomial features
Shape train: 1430, test: 21
Shape train: 1430, test: 272
```

# 3. Light GBM

Here for the sake of higher score on kaggle, **I commented out some lines, which ordinary used**. It allowed me went from 21% to 17% in leaderborad.

In [35]:

```
y = df_train["SalePrice"]
df_train_id = df_train.pop('Id')
df_train.drop(["SalePrice"], axis=1, inplace=True)
print("Shape train: %s, test: %s" % (df_train.shape))
```

```
Shape train: 1430, test: 270
```

In [36]:

```
X_train = df_train
y_train = y
#X_train, X_test, y_train, y_test = train_test_split( df_train, y, test_size=0.1, r
```

The best parameters was found by using *GridSearch*

In [ ]:

```
'''
from sklearn.model_selection import GridSearchCV
hyper_params = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': ['l2'],
    'first_metric_only': True,
    'bagging_freq': 10,
    'verbose': 0,
    "num_leaves": 128,
    "num_iterations": 3000
}


gbm = lgb.LGBMRegressor(**hyper_params)
parameters = {
    'learning_rate': [0.001],
    'feature_fraction': [0.5, 0.7],
    'bagging_fraction': [0.6, 0.75],
    "max_depth": [4, 5, 6],
    'bagging_freq': [10],
    "max_bin": [300, 400,500],
    "n_estimators": [200, 300, 400]
}

grid = GridSearchCV(estimator=gbm, param_grid=parameters,
                   cv=5, verbose=500, n_jobs=-1, refit=True)
grid.fit(X_train, y_train)
print(f"{grid.best_estimator_}\n{grid.best_params_}\n{grid.best_score_}")
'''
```

In [37]:

```
best_parameters = {
    'task': 'train',
    'boosting_type': 'gbdt',
    'objective': 'regression',
    'metric': ['l2', 'huber'],
    'first_metric_only': True,
    'learning_rate': 0.005,
    'feature_fraction': 0.5,
    'bagging_fraction': 0.7,
    'bagging_freq': 10,
    'verbose': 0,
    "max_depth": 6,
    "num_leaves": 128,
    "max_bin": 512,
    "num_iterations": 3000,
    "n_estimators": 50
}

gbm = lgb.LGBMRegressor(**best_parameters)
```

In [38]:

```
gbm.fit(X_train, y_train,
#         eval_set=[(X_test, y_test)],
#         eval_metric='l2',
#         early_stopping_rounds=1000,
        verbose = 500)
```

```
/home/user/anaconda3/lib/python3.8/site-packages/lightgbm/engine.py:14
8: UserWarning: Found `num_iterations` in params. Will use it instead
of argument
  warnings.warn("Found `{}` in params. Will use it instead of argumen
t".format(alias))
```

Out[38]:

```
LGBMRegressor(bagging_fraction=0.7, bagging_freq=10, feature_fraction=
0.5,
              first_metric_only=True, learning_rate=0.005, max_bin=51
2,
              max_depth=6, metric=['l2', 'huber'], n_estimators=50,
              num_iterations=3000, num_leaves=128, objective='regressi
on',
              task='train', verbose=0)
```

## Prediction

In [39]:

```
y_pred = gbm.predict(X_train, num_iteration=gbm.best_iteration_)
print('The rmse of prediction is:', round(mean_squared_error(y_pred, y_train) ** 0.
```

```
The rmse of prediction is: 0.05951
```

## Results

In [40]:

```
def test_pipeline(data):
    fill_missing_with_mode_group(data, 'LotFrontage')
    fill_missing_with_mode_group(data, 'BsmtUnfSF')
    data = formating_1(data)
    data = formating_2(data)
    convert_quality_into_numeric(data)
    fill_missing(data, ["MiscFeature", "Alley", "Fence",
                        "GarageType","MasVnrType"], "Not exist")
    fill_missing(data, ['GarageYrBlt', 'MasVnrArea', 'BsmtHalfBath', 'BsmtFullBath'
                            'BsmtFinSF1', 'GarageArea', 'TotalBsmtSF', 'GarageCars'
                           "PoolQC", "FireplaceQu", "GarageFinish", "GarageQual", "
                            "BsmtCond","BsmtExposure","BsmtFinType1", "BsmtFinType2
    fill_missing_with_mode(data, ['Electrical', 'MSZoning', 'Functional', 'Utilitie
    'KitchenQual', 'SaleType'])
    data['TotalSF'] = data['TotalBsmtSF'] + data['1stFlrSF'] + data['2ndFlrSF']
    data["time_before_remodelation"] = data.YearRemodAdd - data.YearBuilt
    data = add_logs(data, loglist)
    data = dummy_encode(data)
    data = fix_missing_cols(df_train_here, data)
    data_poly =  load_poly_features(data, cols=correlated_cols)
    data = data.merge(right=data_poly.reset_index(), how='left', on='Id')
    data.drop("SalePrice", axis=1, inplace=True)
    return(data)
```

In [41]:

```
df_test = test_pipeline(pd.read_csv("house-prices-advanced-regression-techniques/te
df_test_id = df_test.pop('Id')
test_pred = np.expm1(gbm.predict(df_test, num_iteration=gbm.best_iteration_))

df_test["SalePrice"] = test_pred
df_test["Id"] = df_test_id
df_test.to_csv("results.csv", columns=["Id", "SalePrice"], index=False)
```

```
['MSZoning', 'Street', 'Alley', 'LotShape', 'LandContour', 'Utilitie
s', 'LotConfig', 'LandSlope', 'Neighborhood', 'BldgType', 'HouseStyl
e', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrTyp
e', 'Foundation', 'Heating', 'Electrical', 'Functional', 'GarageType',
'Fence', 'MiscFeature', 'SaleType', 'SaleCondition']
Loading polynomial features..
Polynomial Features shape: (1459, 20)
Loaded polynomial features
```

Score: 0.12301 (17% [825/4675])

[https://www.kaggle.com/konstantinlp/competitions (https://www.kaggle.com/konstantinlp/competitions)](https://www.kaggle.com/konstantinlp/competitions)