

title: House prices prediction

author: Wittawat Muangkot

date: 2022-04-16

Description

this paper will analyze House prices dataset and built model for regression prediction the sale price of house by following step below;

- Import essential package
- Collect Data
- Basic explore dataset and split data into train and test set
- cleaning missing data
- feature engineering
- category data encoding
 - order encoding
 - one hot encoding
- data transformation
- feature selection
 - constant feature
 - lasso
 - rfe
- model selection
- hyperparameter
- df_test prediction

Import essential package

In [256...]

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
pd.set_option('display.max_columns',100)
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

from sklearn.cluster import KMeans
from feature_engine.encoding import OneHotEncoder
import scipy.stats as stats
from scipy.stats import skew
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.ensemble import GradientBoostingRegressor
```

```

from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform
import joblib

```

Collect Data

In [256...]

```

df_train=pd.read_csv('/Users/wittawatmuangkot/Downloads/house-prices-advanced-regression-t
df_test=pd.read_csv('/Users/wittawatmuangkot/Downloads/house-prices-advanced-regression-te

```

In [256...]

```

print('df_train \nhave total rows: {} and total columns: {}'.format(df_train.shape[0],df_t
missing_col_train=[x for x in df_train.columns
    if df_train[x].isnull().any()]
print('Total missing values columns: {}'.format(len(missing_col_train)))
print('number of each data type\n')
dtypes_train = pd.DataFrame(df_train.dtypes)
print(dtypes_train.value_counts(),'\n')

print('df_test \nhave total rows: {} and total columns: {}'.format(df_test.shape[0],df_te
missing_col_test=[x for x in df_test.columns
    if df_test[x].isnull().any()]
print('Total missing values columns: {}'.format(len(missing_col_test)))
print('number of each data type\n')
dtypes_test = pd.DataFrame(df_test.dtypes)
print(dtypes_test.value_counts())

```

```

df_train
have total rows: 1460 and total columns: 81
Total missing values columns: 19
number of each data type

```

```

object      43
int64       35
float64     3
dtype: int64

```

```

df_test
have total rows: 1459 and total columns: 80
Total missing values columns: 33
number of each data type

```

```

object      43
int64       26
float64     11
dtype: int64

```

After check the infomation of this dataset we found that there are missing values in some column and need to clean data before do the next process of model building. As this dataset has many feature train_df 81 columns and test_df 80 columns, so we need to do the feature selection that to select the low significance feature.

First remove ID columns that has no any unique values in columns that mean this columns not useful for our model prediction

In [257...]

```
df_train=df_train.drop('Id',axis=1)
df_test=df_test.drop('Id',axis=1)
```

Split train_df data into train and test set using for evaluate model, using the test size 30% and train size for 70%

In [257...]

```
x_train,x_test,y_train,y_test=train_test_split(df_train,df_train['SalePrice'],test_size=0.3)
```

In [257...]

```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1022 entries, 135 to 1126
Data columns (total 80 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   MSSubClass        1022 non-null   int64  
 1   MSZoning          1022 non-null   object  
 2   LotFrontage       832 non-null    float64 
 3   LotArea           1022 non-null   int64  
 4   Street            1022 non-null   object  
 5   Alley              66 non-null    object  
 6   LotShape           1022 non-null   object  
 7   LandContour        1022 non-null   object  
 8   Utilities          1022 non-null   object  
 9   LotConfig          1022 non-null   object  
 10  LandSlope          1022 non-null   object  
 11  Neighborhood       1022 non-null   object  
 12  Condition1         1022 non-null   object  
 13  Condition2         1022 non-null   object  
 14  BldgType           1022 non-null   object  
 15  HouseStyle         1022 non-null   object  
 16  OverallQual        1022 non-null   int64  
 17  OverallCond        1022 non-null   int64  
 18  YearBuilt          1022 non-null   int64  
 19  YearRemodAdd       1022 non-null   int64  
 20  RoofStyle          1022 non-null   object  
 21  RoofMatl           1022 non-null   object  
 22  Exterior1st        1022 non-null   object  
 23  Exterior2nd        1022 non-null   object  
 24  MasVnrType          1019 non-null   object  
 25  MasVnrArea          1019 non-null   float64 
 26  ExterQual           1022 non-null   object  
 27  ExterCond           1022 non-null   object  
 28  Foundation          1022 non-null   object  
 29  BsmtQual            996 non-null    object  
 30  BsmtCond            996 non-null    object  
 31  BsmtExposure        996 non-null    object  
 32  BsmtFinType1         996 non-null   object  
 33  BsmtFinSF1          1022 non-null   int64  
 34  BsmtFinType2         996 non-null   object  
 35  BsmtFinSF2          1022 non-null   int64  
 36  BsmtUnfSF           1022 non-null   int64  
 37  TotalBsmtSF          1022 non-null   int64  
 38  Heating              1022 non-null   object  
 39  HeatingQC            1022 non-null   object  
 40  CentralAir           1022 non-null   object  
 41  Electrical           1021 non-null   object  
 42  1stFlrSF             1022 non-null   int64  
 43  2ndFlrSF             1022 non-null   int64  
 44  LowQualFinSF          1022 non-null   int64  
 45  GrLivArea            1022 non-null   int64
```

```
46 BsmtFullBath    1022 non-null    int64
47 BsmtHalfBath   1022 non-null    int64
48 FullBath       1022 non-null    int64
49 HalfBath        1022 non-null    int64
50 BedroomAbvGr   1022 non-null    int64
51 KitchenAbvGr   1022 non-null    int64
52 KitchenQual     1022 non-null    object
53 TotRmsAbvGrd  1022 non-null    int64
54 Functional      1022 non-null    object
55 Fireplaces      1022 non-null    int64
56 FireplaceQu    535 non-null    object
57 GarageType      968 non-null    object
58 GarageYrBlt    968 non-null    float64
59 GarageFinish    968 non-null    object
60 GarageCars      1022 non-null    int64
61 GarageArea      1022 non-null    int64
62 GarageQual     968 non-null    object
63 GarageCond      968 non-null    object
64 PavedDrive      1022 non-null    object
65 WoodDeckSF     1022 non-null    int64
66 OpenPorchSF    1022 non-null    int64
67 EnclosedPorch  1022 non-null    int64
68 3SsnPorch      1022 non-null    int64
69 ScreenPorch    1022 non-null    int64
70 PoolArea        1022 non-null    int64
71 PoolQC          5 non-null    object
72 Fence            202 non-null    object
73 MiscFeature     40 non-null    object
74 MiscVal         1022 non-null    int64
75 MoSold          1022 non-null    int64
76 YrSold          1022 non-null    int64
77 SaleType        1022 non-null    object
78 SaleCondition   1022 non-null    object
79 SalePrice       1022 non-null    int64
dtypes: float64(3), int64(34), object(43)
memory usage: 646.7+ KB
```

In [257...]

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1459 entries, 0 to 1458
Data columns (total 79 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   MSSubClass       1459 non-null    int64  
 1   MSZoning         1455 non-null    object  
 2   LotFrontage      1232 non-null    float64 
 3   LotArea          1459 non-null    int64  
 4   Street           1459 non-null    object  
 5   Alley             107 non-null    object  
 6   LotShape          1459 non-null    object  
 7   LandContour      1459 non-null    object  
 8   Utilities         1457 non-null    object  
 9   LotConfig         1459 non-null    object  
 10  LandSlope         1459 non-null    object  
 11  Neighborhood     1459 non-null    object  
 12  Condition1       1459 non-null    object  
 13  Condition2       1459 non-null    object  
 14  BldgType          1459 non-null    object  
 15  HouseStyle        1459 non-null    object  
 16  OverallQual      1459 non-null    int64  
 17  OverallCond      1459 non-null    int64  
 18  YearBuilt         1459 non-null    int64  
 19  YearRemodAdd     1459 non-null    int64  
 20  RoofStyle         1459 non-null    object
```

```
21 RoofMatl          1459 non-null   object
22 Exterior1st       1458 non-null   object
23 Exterior2nd       1458 non-null   object
24 MasVnrType        1443 non-null   object
25 MasVnrArea         1444 non-null   float64
26 ExterQual         1459 non-null   object
27 ExterCond         1459 non-null   object
28 Foundation        1459 non-null   object
29 BsmtQual          1415 non-null   object
30 BsmtCond          1414 non-null   object
31 BsmtExposure      1415 non-null   object
32 BsmtFinType1      1417 non-null   object
33 BsmtFinSF1         1458 non-null   float64
34 BsmtFinType2      1417 non-null   object
35 BsmtFinSF2         1458 non-null   float64
36 BsmtUnfSF          1458 non-null   float64
37 TotalBsmtSF        1458 non-null   float64
38 Heating            1459 non-null   object
39 HeatingQC          1459 non-null   object
40 CentralAir         1459 non-null   object
41 Electrical         1459 non-null   object
42 1stFlrSF           1459 non-null   int64
43 2ndFlrSF           1459 non-null   int64
44 LowQualFinSF       1459 non-null   int64
45 GrLivArea          1459 non-null   int64
46 BsmtFullBath       1457 non-null   float64
47 BsmtHalfBath       1457 non-null   float64
48 FullBath           1459 non-null   int64
49 HalfBath           1459 non-null   int64
50 BedroomAbvGr        1459 non-null   int64
51 KitchenAbvGr        1459 non-null   int64
52 KitchenQual         1458 non-null   object
53 TotRmsAbvGrd       1459 non-null   int64
54 Functional          1457 non-null   object
55 Fireplaces          1459 non-null   int64
56 FireplaceQu         729 non-null   object
57 GarageType          1383 non-null   object
58 GarageYrBlt         1381 non-null   float64
59 GarageFinish         1381 non-null   object
60 GarageCars          1458 non-null   float64
61 GarageArea          1458 non-null   float64
62 GarageQual          1381 non-null   object
63 GarageCond          1381 non-null   object
64 PavedDrive          1459 non-null   object
65 WoodDeckSF          1459 non-null   int64
66 OpenPorchSF         1459 non-null   int64
67 EnclosedPorch       1459 non-null   int64
68 3SsnPorch           1459 non-null   int64
69 ScreenPorch          1459 non-null   int64
70 PoolArea            1459 non-null   int64
71 PoolQC              3 non-null    object
72 Fence                290 non-null   object
73 MiscFeature          51 non-null   object
74 MiscVal              1459 non-null   int64
75 MoSold               1459 non-null   int64
76 YrSold               1459 non-null   int64
77 SaleType             1458 non-null   object
78 SaleCondition        1459 non-null   object
dtypes: float64(11), int64(25), object(43)
memory usage: 900.6+ KB
```

In [257]:

```
discrete = []

for var in numeric_col:
    if len(df_train[var].unique()) < 20 and var not in year_col:
```

```

        print(var, ' values: ', df_train[var].unique())
        discrete.append(var)
print()
print('There are {} discrete variables'.format(len(discrete)))

```

MSSubClass values: [60 20 70 50 190 45 90 120 30 85 80 160 75 180 40]
OverallQual values: [7 6 8 5 9 4 10 3 1 2]
OverallCond values: [5 8 6 7 4 2 3 9 1]
BsmtFullBath values: [1 0 2 3]
BsmtHalfBath values: [0 1 2]
FullBath values: [2 1 3 0]
HalfBath values: [1 0 2]
BedroomAbvGr values: [3 4 1 2 0 5 6 8]
KitchenAbvGr values: [1 2 3 0]
TotRmsAbvGrd values: [8 6 7 9 5 11 4 10 12 3 2 14]
Fireplaces values: [0 1 2 3]
GarageCars values: [2 3 1 0 4]
PoolArea values: [0 512 648 576 555 480 519 738]
MoSold values: [2 5 9 12 10 8 11 4 1 7 3 6]

There are 14 discrete variables

Feature name correction

Some columns name of this dataset start with numeric, so we need to replace with text, to prevent miss understand of program when we call these columns

In [257...]

```

X_train.rename(columns={'1stFlrSF':'FirstFlrSF', '2ndFlrSF':'SecondFlrSF', '3SsnPorch':'ThirdFlrSF',
X_test.rename(columns={'1stFlrSF':'FirstFlrSF', '2ndFlrSF':'SecondFlrSF', '3SsnPorch':'ThirdFlrSF',
df_test.rename(columns={'1stFlrSF':'FirstFlrSF', '2ndFlrSF':'SecondFlrSF', '3SsnPorch':'ThirdFlrSF'})

```

Missing data check and Imputation

let start with checking the total number of missing columns in X_train dataset. to prevent overfitting in test dataset, we will Impute missing data refering missing values from the X_train and transform to test set

In [257...]

```

miss_col=[a for a in X_train.columns
          if X_train[a].isnull().any()]
len(miss_col)

```

Out[257...]

19

In [257...]

```

def missing_table(data):
    miss_col=[a for a in data.columns
              if data[a].isnull().any()]
    missing=[]
    percent=[]
    for i in miss_col:
        miss=data[i].isnull().sum()
        percent_miss=(miss/data.shape[0])*100
        missing.append(miss)
        percent.append(percent_miss)

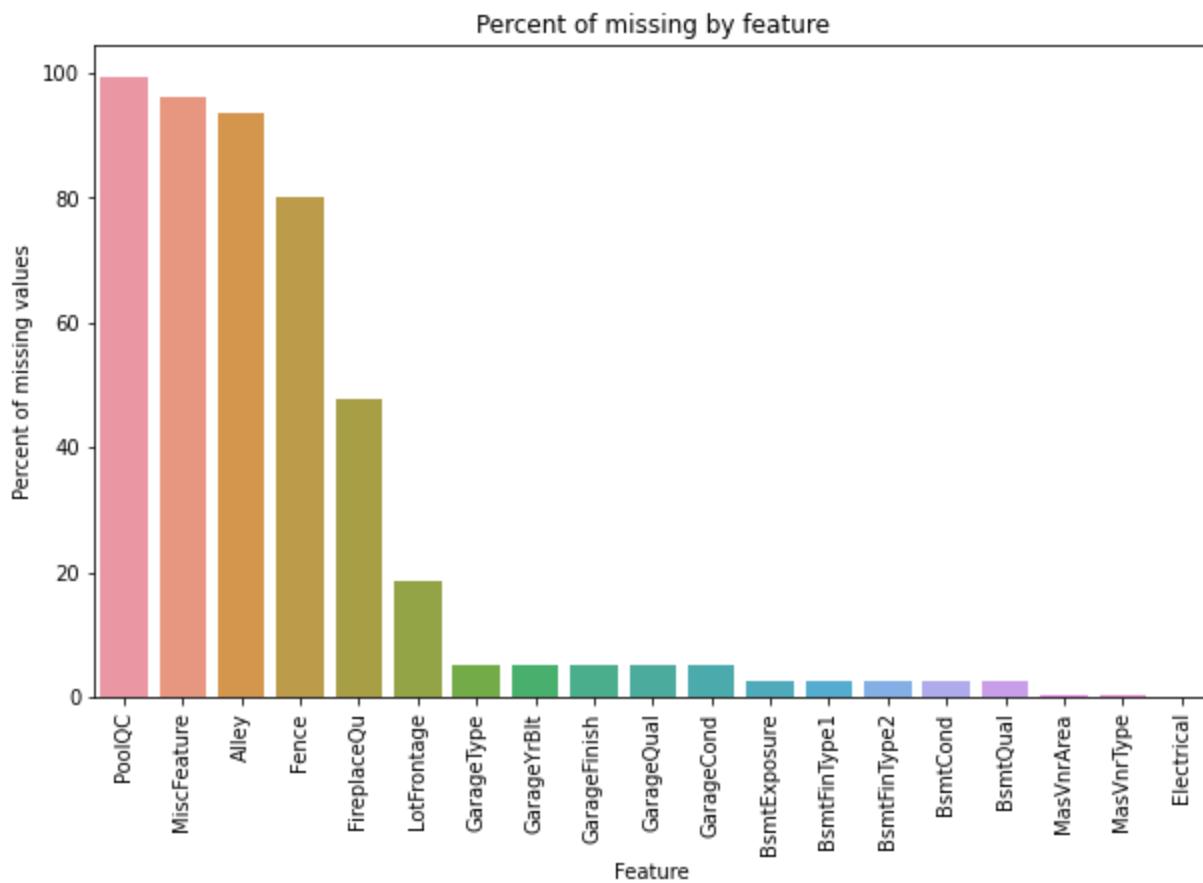
    miss_table=pd.DataFrame({'Col':miss_col,'missing':missing,'percent':percent}).sort_values(by='percent', ascending=False)
    plt.subplots(figsize=(10,6))
    sns.barplot(x=miss_table['Col'],y=miss_table['percent'])
    plt.xlabel('Feature')
    plt.ylabel('Percent of missing values')
    plt.title('Percent of missing by feature')
    plt.xticks(rotation='90')

```

```
    plt.show()  
    return miss_table
```

In [257...]

```
missing_table(X_train)
```



Out [257...]

	Col	missing	percent
16	PoolQC	1017	99.510763
18	MiscFeature	982	96.086106
1	Alley	956	93.542074
17	Fence	820	80.234834
10	FireplaceQu	487	47.651663
0	LotFrontage	190	18.590998
11	GarageType	54	5.283757
12	GarageYrBlt	54	5.283757
13	GarageFinish	54	5.283757
14	GarageQual	54	5.283757
15	GarageCond	54	5.283757
6	BsmtExposure	26	2.544031
7	BsmtFinType1	26	2.544031
8	BsmtFinType2	26	2.544031
5	BsmtCond	26	2.544031
4	BsmtQual	26	2.544031
3	MasVnrArea	3	0.293542

Col	missing	percent
2	MasVnrType	3 0.293542
9	Electrical	1 0.097847

As data of missing data show many of missing data, so we will separate into 2 group. one more than 5 percent and another less than 5 percent. and we will use impute missing values method depend on group and data type of each group.

- more than 5 percent of category columns will be imputed by 'None'
- more than 5 percent of numerical columns will be imputed by 0
- less than 5 percent of both category and numeric will be imputed by sampling imputation method

In [257...]

```
col_more_five=['PoolQC','MiscFeature','Alley','Fence','FireplaceQu','LotFrontage','GarageF...
```

next separate each group into category and numerical group

In [258...]

```
miss_more_five_cat=[c for c in X_train[col_more_five]
                     if X_train[c].dtypes == 'object']

miss_more_five_num=[n for n in X_train[col_more_five]
                     if X_train[n].dtypes in ['int','float']]

miss_less_five_cat=[a for a in X_train[col_less_five]
                     if X_train[a].dtypes == 'object']
miss_less_five_num=[b for b in X_train[col_less_five]
                     if X_train[b].dtypes in ['int','float']]
```

fill the missing in columns missing more than 5 percent both category and numerical group

In [258...]

```
def fill_na(feature,data,fill_val):
    for i in feature:
        data[i]=data[i].fillna(fill_val, axis=0)
```

In [258...]

```
fill_na(miss_more_five_cat,X_train,'None')
fill_na(miss_more_five_num,X_train,0)
fill_na(miss_more_five_cat,X_test,'None')
fill_na(miss_more_five_num,X_test,0)
```

impute missing data columns with value less than 5 percent by using sampling method in X_train dataset and transform to X_test set

In [258...]

```
for feat in miss_less_five_cat:
    X_train[feat+'_imputed']=X_train[feat].copy()
    random_sample=X_train[feat].dropna().sample(X_train[feat].isnull().sum(),random_state=42)
    random_sample.index=X_train[X_train[feat].isnull()].index
    X_train.loc[X_train[feat].isnull(),feat+'_imputed']=random_sample
```

In [258...]

```
for feat in miss_less_five_num:
    X_train[feat+'_imputed']=X_train[feat].copy()
    random_sample=X_train[feat].dropna().sample(X_train[feat].isnull().sum(),random_state=42)
    random_sample.index=X_train[X_train[feat].isnull()].index
    X_train.loc[X_train[feat].isnull(),feat+'_imputed']=random_sample
```

```
In [258...]  
for feat in miss_less_five_cat:  
    X_test[feat+'_imputed']=X_test[feat].copy()  
    random_sample=X_train[feat].dropna().sample(X_test[feat].isnull().sum(),random_state=42)  
    random_sample.index=X_test[X_test[feat].isnull()].index  
    X_test.loc[X_test[feat].isnull(),feat+'_imputed']=random_sample
```

```
In [258...]  
for feat in miss_less_five_num:  
    X_test[feat+'_imputed']=X_test[feat].copy()  
    random_sample=X_train[feat].dropna().sample(X_test[feat].isnull().sum(),random_state=42)  
    random_sample.index=X_test[X_test[feat].isnull()].index  
    X_test.loc[X_test[feat].isnull(),feat+'_imputed']=random_sample
```

```
In [258...]  
def categorical_distribution(df, variable_original, variable_imputed):  
  
    tmp = pd.concat([df[variable_original].value_counts() / len(df[variable_original].dropna()),  
                    df[variable_imputed].value_counts() / len(df)],  
                    axis=1)  
  
    tmp.columns = ['original', 'imputed']  
  
    return tmp
```

```
In [258...]  
def distribute_plot_compare(df, variable, target):  
  
    fig = plt.figure()  
    ax = fig.add_subplot(111)  
  
    for category in df[variable].dropna().unique():  
        df[df[variable]==category][target].plot(kind='kde', ax=ax)  
  
    lines, labels = ax.get_legend_handles_labels()  
    labels = df[variable].dropna().unique()  
    ax.legend(lines, labels, loc='best')  
  
    plt.show()
```

as the comparison data and visualization below show that data was low distributed by using sampling imputation.

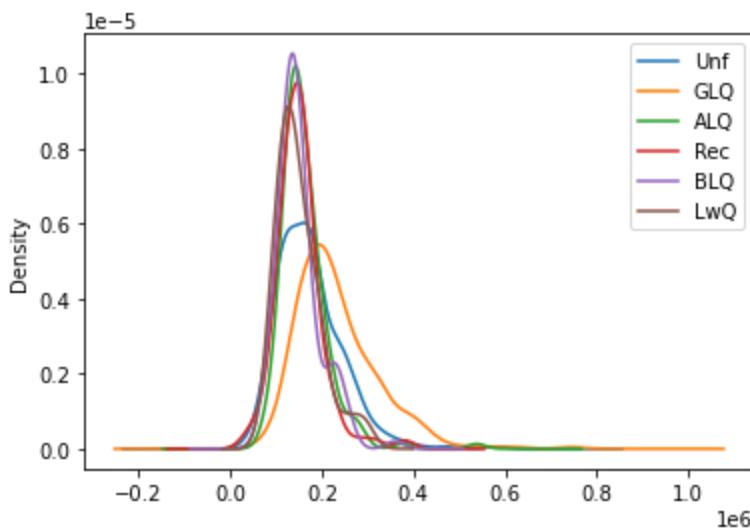
```
In [258...]  
categorical_distribution(X_train, 'BsmtFinType1', 'BsmtFinType1_imputed')
```

	original	imputed
Unf	0.300201	0.299413
GLQ	0.297189	0.298434
ALQ	0.156627	0.155577
BLQ	0.105422	0.105675
Rec	0.087349	0.086106
LwQ	0.053213	0.054795

now let's look at the distribution of the target within each variable category in original vs imputed columns, as we can see there is no difference after we imputed with random sampling method

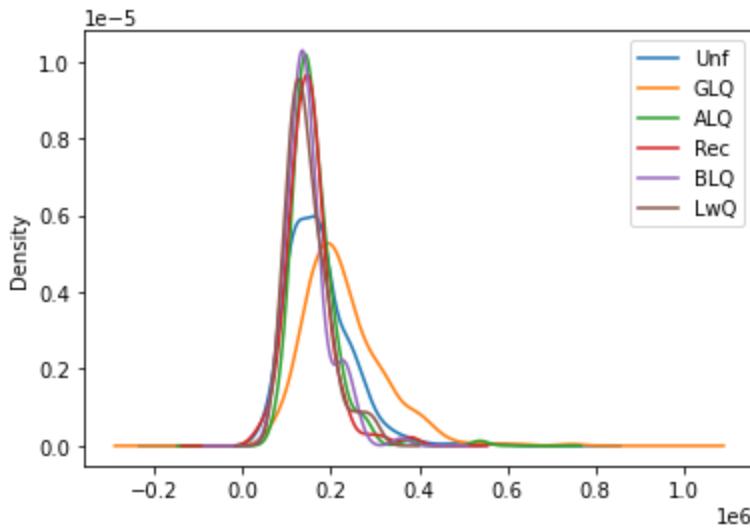
In [259...]

```
distribute_plot_compare(X_train, 'BsmtFinType1', 'SalePrice')
```



In [259...]

```
distribute_plot_compare(X_train, 'BsmtFinType1_imputed', 'SalePrice')
```



For imputed numerical columns, we'll compare distribution original vs imputed columns. show with no difference

In [259...]

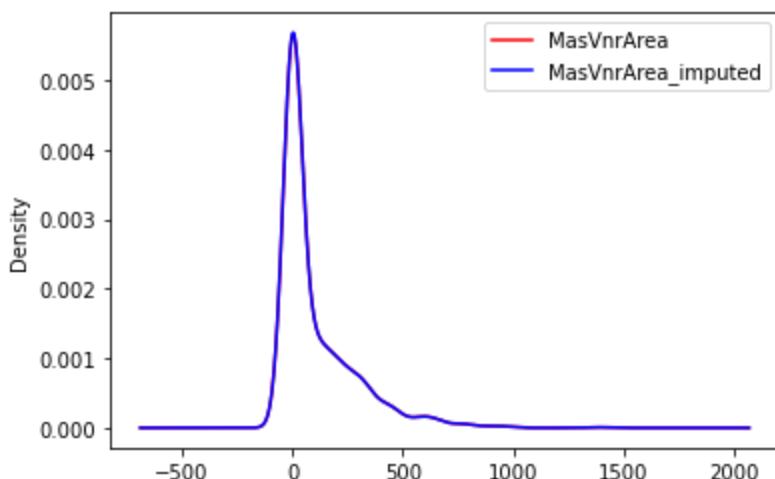
```
fig = plt.figure()
ax = fig.add_subplot(111)

X_train['MasVnrArea'].plot(kind='kde', ax=ax, color='r')
X_train['MasVnrArea_imputed'].plot(kind='kde', ax=ax, color='b')

lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[259...]

```
<matplotlib.legend.Legend at 0x7fb5189244c0>
```



let drop original missing columns, we'll using the imputed columns only

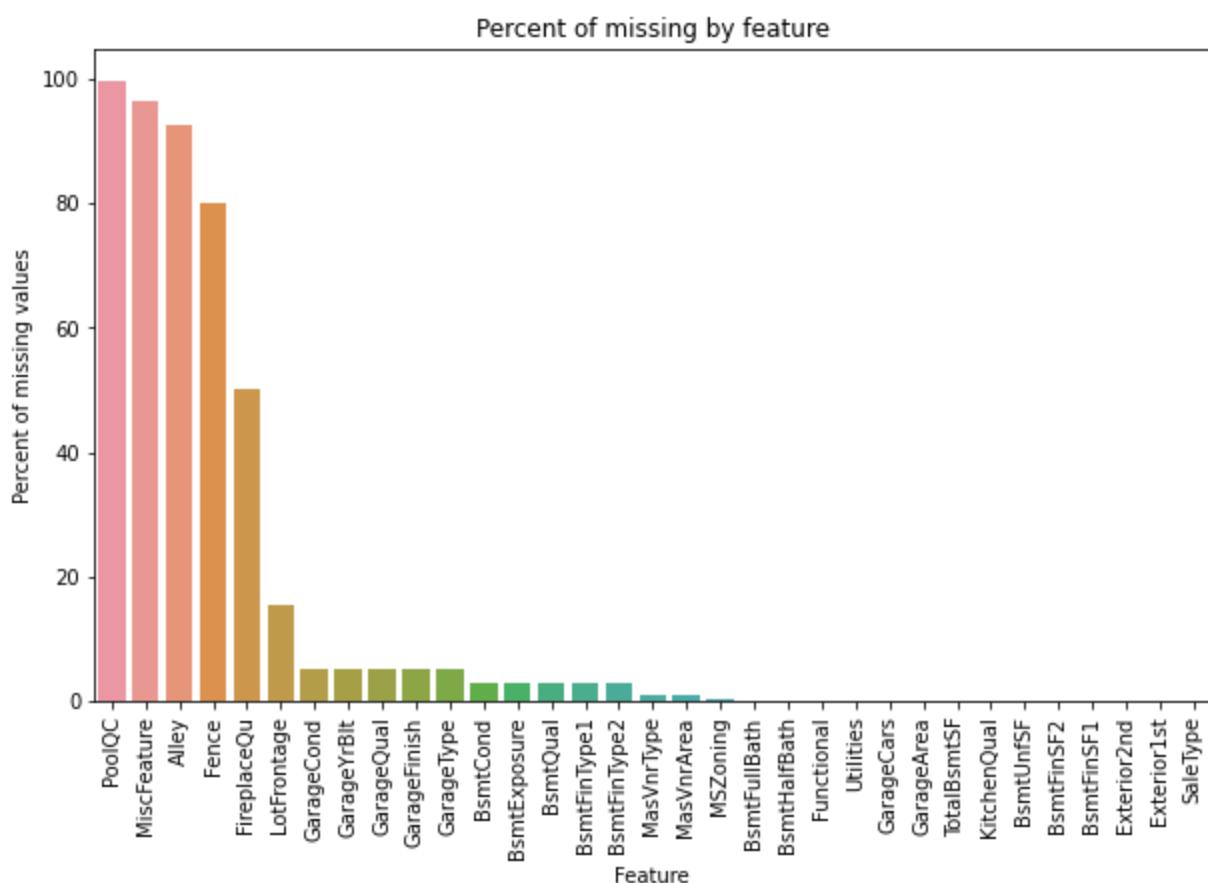
In [259...]

```
X_train.drop(miss_less_five_cat, axis=1, inplace=True)
X_train.drop(miss_less_five_num, axis=1, inplace=True)
X_test.drop(miss_less_five_cat, axis=1, inplace=True)
X_test.drop(miss_less_five_num, axis=1, inplace=True)
```

Do the process of clean data for the df_test dataset

In [259...]

```
missing_table(df_test)
```



Out [259...]

	Col	missing	percent
29	PoolQC	1456	99.794380
31	MiscFeature	1408	96.504455

	Col	missing	percent
2	Alley	1352	92.666210
30	Fence	1169	80.123372
21	FireplaceQu	730	50.034270
1	LotFrontage	227	15.558602
28	GarageCond	78	5.346127
23	GarageYrBlt	78	5.346127
27	GarageQual	78	5.346127
24	GarageFinish	78	5.346127
22	GarageType	76	5.209047
9	BsmtCond	45	3.084304
10	BsmtExposure	44	3.015764
8	BsmtQual	44	3.015764
11	BsmtFinType1	42	2.878684
13	BsmtFinType2	42	2.878684
6	MasVnrType	16	1.096642
7	MasVnrArea	15	1.028101
0	MSZoning	4	0.274160
17	BsmtFullBath	2	0.137080
18	BsmtHalfBath	2	0.137080
20	Functional	2	0.137080
3	Utilities	2	0.137080
25	GarageCars	1	0.068540
26	GarageArea	1	0.068540
16	TotalBsmtSF	1	0.068540
19	KitchenQual	1	0.068540
15	BsmtUnfSF	1	0.068540
14	BsmtFinSF2	1	0.068540
12	BsmtFinSF1	1	0.068540
5	Exterior2nd	1	0.068540
4	Exterior1st	1	0.068540
32	SaleType	1	0.068540

In [259...]

```
missing_morethan_five=['PoolQC','MiscFeature','Alley','Fence','FireplaceQu','LotFrontage',
missing_lessthan_five=['BsmtCond','BsmtExposure','BsmtQual','BsmtFinType1','BsmtFinType2']
```

In [259...]

```
missing_morethan_five_cat=[c for c in df_test[missing_morethan_five]
                           if df_test[c].dtypes == 'object']
```

```
missing_morethan_five_num=[n for n in df_test[missing_morethan_five]
                           if df_test[n].dtypes in ['int','float']]
```

```
missing_lessthan_five_cat=[a for a in df_test[missing_lessthan_five]
                           if df_test[a].dtypes == 'object']
missing_lessthan_five_num=[b for b in df_test[missing_lessthan_five]
                           if df_test[b].dtypes in ['int','float']]
```

In [259...]

```
fill_na(missing_morethan_five_cat,df_test,'None')
fill_na(missing_morethan_five_num,df_test,0)
```

In [259...]

```
for feat in missing_lessthan_five_cat:
    df_test[feat+'_imputed']=df_test[feat].copy()
    random_sample=df_test[feat].dropna().sample(df_test[feat].isnull().sum(),random_state=42)
    random_sample.index=df_test[df_test[feat].isnull()].index
    df_test.loc[df_test[feat].isnull(),feat+'_imputed']=random_sample
```

In [259...]

```
for feat in missing_lessthan_five_num:
    df_test[feat+'_imputed']=df_test[feat].copy()
    random_sample=df_test[feat].dropna().sample(df_test[feat].isnull().sum(),random_state=42)
    random_sample.index=df_test[df_test[feat].isnull()].index
    df_test.loc[df_test[feat].isnull(),feat+'_imputed']=random_sample
```

In [260...]

```
categorical_distribution(df_test,'BsmtFinType2','BsmtFinType2_imputed')
```

Out[260...]

	original	imputed
Unf	0.872971	0.872515
Rec	0.035992	0.037012
LwQ	0.028934	0.028787
BLQ	0.024700	0.023989
ALQ	0.023289	0.023989
GLQ	0.014114	0.013708

In [260...]

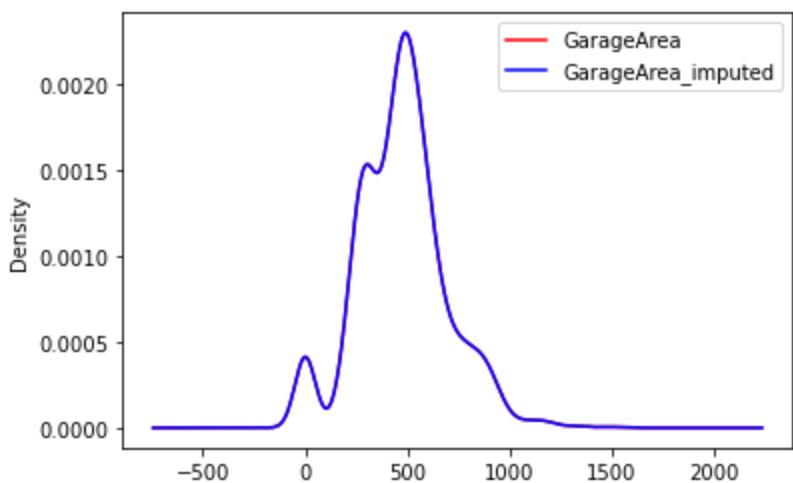
```
fig = plt.figure()
ax = fig.add_subplot(111)

df_test['GarageArea'].plot(kind='kde', ax=ax,color='r')
df_test['GarageArea_imputed'].plot(kind='kde', ax=ax, color='b')

lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
```

Out[260...]

```
<matplotlib.legend.Legend at 0x7fb4fa8d8c40>
```



```
In [260...]: df_test.drop(missing_lessThan_five_cat, axis=1, inplace=True)
df_test.drop(missing_lessThan_five_num, axis=1, inplace=True)
```

```
In [260...]: X_train.columns=X_train.columns.str.replace("_imputed", "")
X_test.columns=X_test.columns.str.replace("_imputed", "")
df_test.columns=df_test.columns.str.replace("_imputed", "")
```

Feature Engineering

refer to data description file of dataset, we will invent new feature by using interested feature. start with year feature

```
In [260...]: X_train["Remodel"] = np.where(X_train['YearBuilt'] == X_train['YearRemodAdd'], 0, 1)
X_train['Lotsize'] = pd.cut(X_train['LotArea'], bins=4, right=False, labels=[0, 1, 2, 3])
X_train['TotalSF'] = X_train['TotalBsmtSF'] + X_train['FirstFlrSF'] + X_train['SecondFlrSF']
X_train['Housesize'] = pd.cut(X_train['TotalSF'], bins=4, right=False, labels=[0, 1, 2, 3])
X_train['TotalPerLot'] = round(X_train['TotalSF']/X_train['LotArea'], 2)
X_train['GroupYearBuilt'] = pd.cut(X_train['YearBuilt'], bins=3, labels=[0, 1, 2])

X_test["Remodel"] = np.where(X_test['YearBuilt'] == X_test['YearRemodAdd'], 0, 1)
X_test['Lotsize'] = pd.cut(X_test['LotArea'], bins=4, right=False, labels=[0, 1, 2, 3])
X_test['TotalSF'] = X_test['TotalBsmtSF'] + X_test['FirstFlrSF'] + X_test['SecondFlrSF']
X_test['Housesize'] = pd.cut(X_test['TotalSF'], bins=4, right=False, labels=[0, 1, 2, 3])
X_test['TotalPerLot'] = round(X_test['TotalSF']/X_test['LotArea'], 2)
X_test['GroupYearBuilt'] = pd.cut(X_test['YearBuilt'], bins=3, labels=[0, 1, 2])

df_test["Remodel"] = np.where(df_test['YearBuilt'] == df_test['YearRemodAdd'], 0, 1)
df_test['Lotsize'] = pd.cut(df_test['LotArea'], bins=4, right=False, labels=[0, 1, 2, 3])
df_test['TotalSF'] = df_test['TotalBsmtSF'] + df_test['FirstFlrSF'] + df_test['SecondFlrSF']
df_test['Housesize'] = pd.cut(df_test['TotalSF'], bins=4, right=False, labels=[0, 1, 2, 3])
df_test['TotalPerLot'] = round(df_test['TotalSF']/df_test['LotArea'], 2)
df_test['GroupYearBuilt'] = pd.cut(df_test['YearBuilt'], bins=3, labels=[0, 1, 2])
```

the chart above show that there are difference in price of house by group of year built and seem that the house with built in the most recent year has the trend to has more price than the older built year

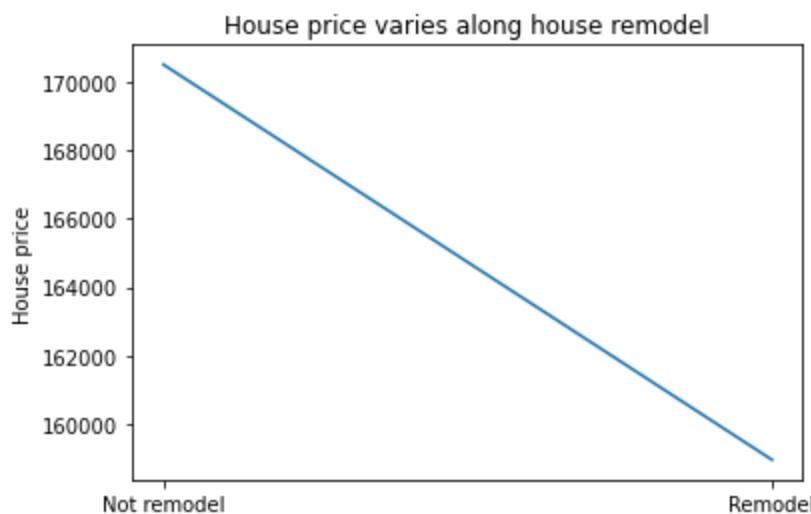
Visualization median SalePrice in compare to the house with remodel and not remodel

```
In [260...]: X_train.groupby('Remodel')['SalePrice'].median().plot()
plt.title('House price varies along house remodel')
```

```

plt.xlabel('')
plt.ylabel('House price')
plt.xticks([0,1],['Not remodel', 'Remodel'])
plt.show();

```



as the chart above notice that house with remodel has less prices than the house with no remodel

As the scatter plot above the price invert with age of house and age of garage, say that with more age got the low prices

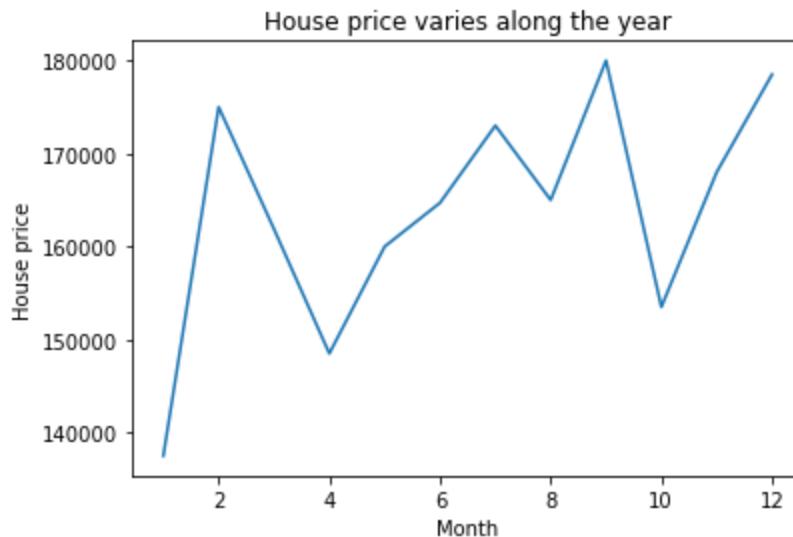
Visaulization to see trand of sale prices in period of month, year and with sale condition

In [260]...

```

x_train.groupby('MoSold')[['SalePrice']].median().plot()
plt.title('House price varies along the year')
plt.xlabel('Month')
plt.ylabel('House price')
plt.show();

```

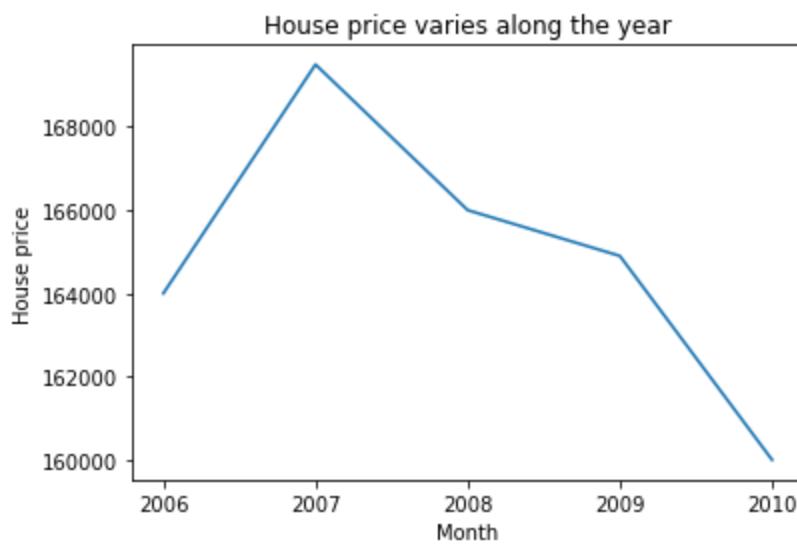


In [261]...

```

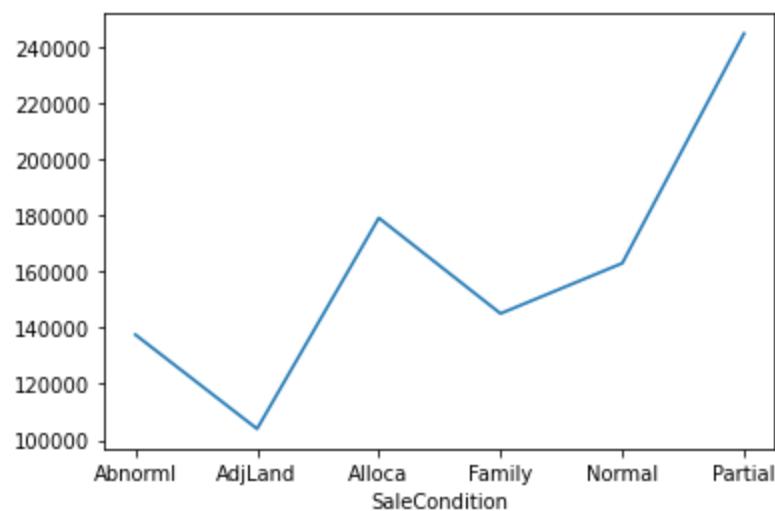
x_train.groupby('YrSold')[['SalePrice']].median().plot()
plt.title('House price varies along the year')
plt.xlabel('Month')
plt.ylabel('House price')
plt.xticks(np.arange(2006,2011,1))
plt.show();

```



In [261...]

```
x_train.groupby('SaleCondition')['SalePrice'].median().plot()  
plt.show()
```



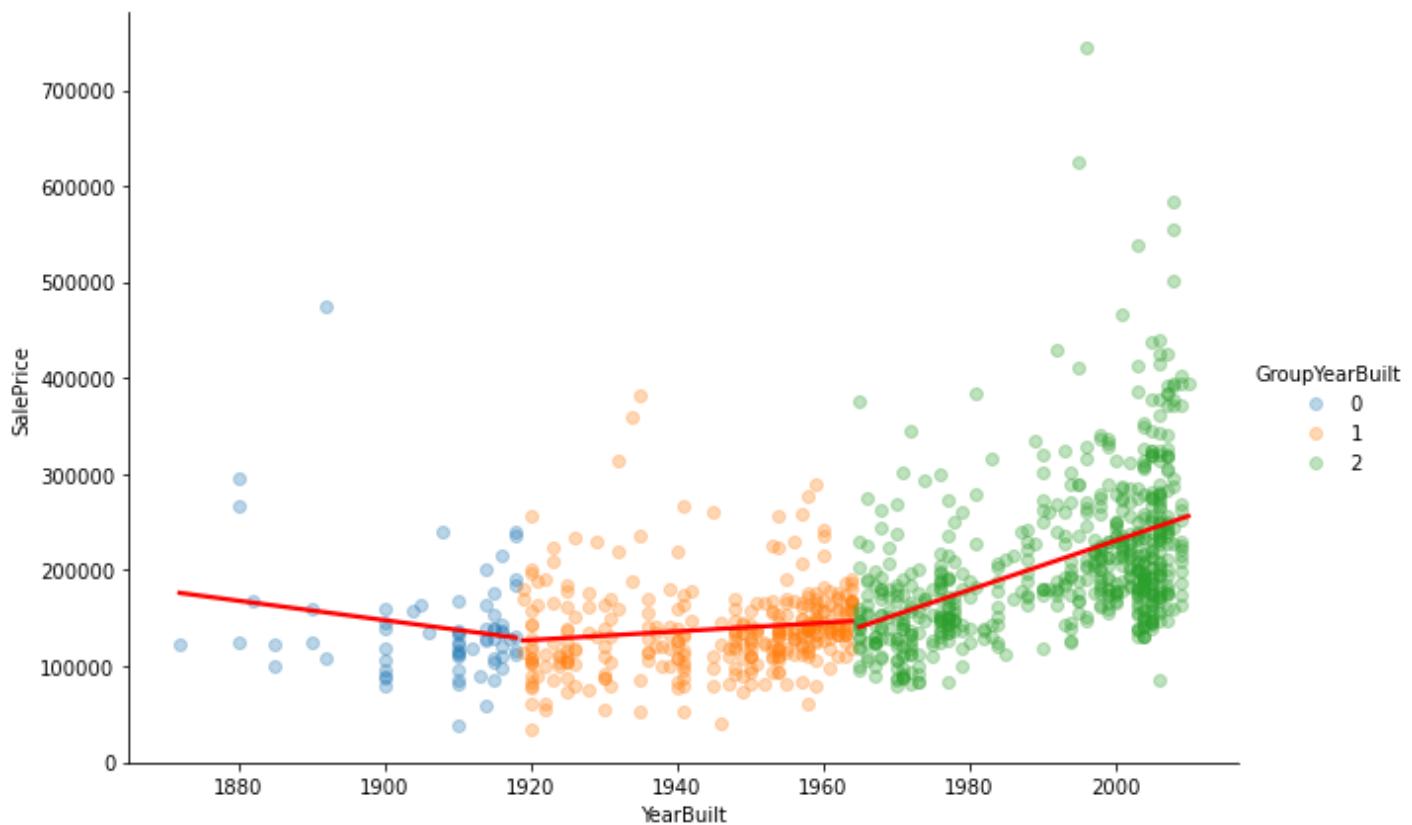
Visualization the relationship between the house price and each group of YearBuilt

In [261...]

```
sns.lmplot(y='SalePrice', x='YearBuilt', hue='GroupYearBuilt', data=x_train, ci=False, scatter_=
```

Out[261...]

```
<seaborn.axisgrid.FacetGrid at 0x7fb4f9f9ea60>
```



as the result of grouping the yearbuilt we can fit the linear model into all range of data

Categorical Encoding

As the dataset has high cardinality and rare rebels in category feature. we will group some of category labels with low frequency less than 5 percent into rare group

first with showing the object columns with number of unique values less than 9 and find the label with low number of frequency less than 5 percent

In [261]...

```
for col in X_train.columns:
    if X_train[col].dtypes == 'O':
        if X_train[col].nunique() < 9:
            print(X_train.groupby(col)[col].count() / len(X_train))
            print()
```

```
MSZoning
C (all)      0.003914
FV           0.042074
RH           0.012720
RL           0.789628
RM           0.151663
Name: MSZoning, dtype: float64
```

```
Street
Grvl      0.003914
Pave     0.996086
Name: Street, dtype: float64
```

```
Alley
Grvl      0.041096
None     0.935421
Pave     0.023483
Name: Alley, dtype: float64
```

```
LotShape
```

```
IR1      0.336595
IR2      0.031311
IR3      0.007828
Reg      0.624266
Name: LotShape, dtype: float64

LandContour
Bnk      0.039139
HLS      0.029354
Low     0.023483
Lvl      0.908023
Name: LandContour, dtype: float64

Utilities
AllPub    0.999022
NoSeWa   0.000978
Name: Utilities, dtype: float64

LotConfig
Corner    0.194716
CulDSac   0.073386
FR2       0.034247
FR3       0.002935
Inside    0.694716
Name: LotConfig, dtype: float64

LandSlope
Gt1      0.944227
Mod      0.046967
Sev      0.008806
Name: LandSlope, dtype: float64

Condition2
Artery    0.001957
Feedr    0.002935
Norm     0.991194
PosN     0.001957
RRAe     0.000978
RRAn     0.000978
Name: Condition2, dtype: float64

BldgType
1Fam     0.834638
2fmCon   0.023483
Duplex   0.035225
Twnhs    0.028376
TwnhsE   0.078278
Name: BldgType, dtype: float64

HouseStyle
1.5Fin   0.101761
1.5Unf   0.010763
1Story   0.495108
2.5Fin   0.006849
2.5Unf   0.008806
2Story   0.309198
SFoyer   0.021526
SLvl     0.045988
Name: HouseStyle, dtype: float64

RoofStyle
Flat     0.010763
Gable    0.771037
Gambrel  0.008806
Hip      0.203523
Mansard  0.003914
```

Shed 0.001957
Name: RoofStyle, dtype: float64

RoofMatl
ClyTile 0.000978
CompShg 0.981409
Metal 0.000978
Roll 0.000978
Tar&Grv 0.008806
WdShake 0.002935
WdShngl 0.003914
Name: RoofMatl, dtype: float64

ExterQual
Ex 0.034247
Fa 0.009785
Gd 0.332681
TA 0.623288
Name: ExterQual, dtype: float64

ExterCond
Ex 0.001957
Fa 0.021526
Gd 0.098826
Po 0.000978
TA 0.876712
Name: ExterCond, dtype: float64

Foundation
BrkTil 0.100783
CBlock 0.427593
PConc 0.446184
Slab 0.017613
Stone 0.004892
Wood 0.002935
Name: Foundation, dtype: float64

Heating
Floor 0.000978
GasA 0.975538
GasW 0.014677
Grav 0.003914
OthW 0.001957
Wall 0.002935
Name: Heating, dtype: float64

HeatingQC
Ex 0.507828
Fa 0.036204
Gd 0.167319
Po 0.000978
TA 0.287671
Name: HeatingQC, dtype: float64

CentralAir
N 0.073386
Y 0.926614
Name: CentralAir, dtype: float64

KitchenQual
Ex 0.063601
Fa 0.029354
Gd 0.406067
TA 0.500978
Name: KitchenQual, dtype: float64

Functional
Maj1 0.008806
Maj2 0.003914
Min1 0.023483
Min2 0.024462
Mod 0.012720
Sev 0.000978
Typ 0.925636
Name: Functional, dtype: float64

FireplaceQu
Ex 0.017613
Fa 0.022505
Gd 0.255382
None 0.476517
Po 0.013699
TA 0.214286
Name: FireplaceQu, dtype: float64

GarageType
2Types 0.003914
Attchd 0.590998
Basment 0.013699
BuiltIn 0.062622
CarPort 0.006849
Detchd 0.269080
None 0.052838
Name: GarageType, dtype: float64

GarageFinish
Fin 0.246575
None 0.052838
RFn 0.285714
Unf 0.414873
Name: GarageFinish, dtype: float64

GarageQual
Ex 0.002935
Fa 0.031311
Gd 0.011742
None 0.052838
Po 0.000978
TA 0.900196
Name: GarageQual, dtype: float64

GarageCond
Ex 0.001957
Fa 0.025440
Gd 0.007828
None 0.052838
Po 0.002935
TA 0.909002
Name: GarageCond, dtype: float64

PavedDrive
N 0.062622
P 0.021526
Y 0.915851
Name: PavedDrive, dtype: float64

PoolQC
Ex 0.001957
Fa 0.000978
Gd 0.001957
None 0.995108
Name: PoolQC, dtype: float64

Fence
GdPrv 0.040117
GdWo 0.040117
MnPrv 0.109589
MnWw 0.007828
None 0.802348
Name: Fence, dtype: float64

MiscFeature
Gar2 0.000978
None 0.960861
Othr 0.000978
Shed 0.036204
TenC 0.000978
Name: MiscFeature, dtype: float64

SaleCondition
Abnorml 0.066536
AdjLand 0.003914
Alloca 0.004892
Family 0.016634
Normal 0.824853
Partial 0.083170
Name: SaleCondition, dtype: float64

BsmtExposure
Av 0.150685
Gd 0.096869
Mn 0.081213
No 0.671233
Name: BsmtExposure, dtype: float64

BsmtFinType2
ALQ 0.016634
BLQ 0.016634
GLQ 0.007828
LwQ 0.030333
Rec 0.040117
Unf 0.888454
Name: BsmtFinType2, dtype: float64

BsmtFinType1
ALQ 0.155577
BLQ 0.105675
GLQ 0.298434
LwQ 0.054795
Rec 0.086106
Unf 0.299413
Name: BsmtFinType1, dtype: float64

BsmtCond
Fa 0.034247
Gd 0.045988
Po 0.000978
TA 0.918787
Name: BsmtCond, dtype: float64

BsmtQual
Ex 0.087084
Fa 0.027397
Gd 0.435421
TA 0.450098
Name: BsmtQual, dtype: float64

MasVnrType

```
BrkCmn      0.011742
BrkFace     0.321918
None        0.577299
Stone       0.089041
Name: MasVnrType, dtype: float64

Electrical
FuseA      0.058708
FuseF      0.022505
FuseP      0.002935
SBrkr     0.915851
Name: Electrical, dtype: float64
```

```
In [261]: few_label=['MSZoning','Alley','LotShape','LandContour','LotConfig','LandSlope','Condition2
```

Group the labels with low frequency into the Rare group

```
In [261]: def find_non_rare_labels(df, variable, tolerance):
```

```
temp = df.groupby([variable])[variable].count() / len(df)

non_rare = [x for x in temp.loc[temp>tolerance].index.values]

return non_rare

def rare_encoding(X_train, X_test, df_test, variable, tolerance):

    X_train = X_train.copy()
    X_test = X_test.copy()
    df_test=df_test.copy()

    frequent_cat = find_non_rare_labels(X_train, variable, tolerance)

    X_train[variable] = np.where(X_train[variable].isin(
        frequent_cat), X_train[variable], 'Rare')

    X_test[variable] = np.where(X_test[variable].isin(
        frequent_cat), X_test[variable], 'Rare')

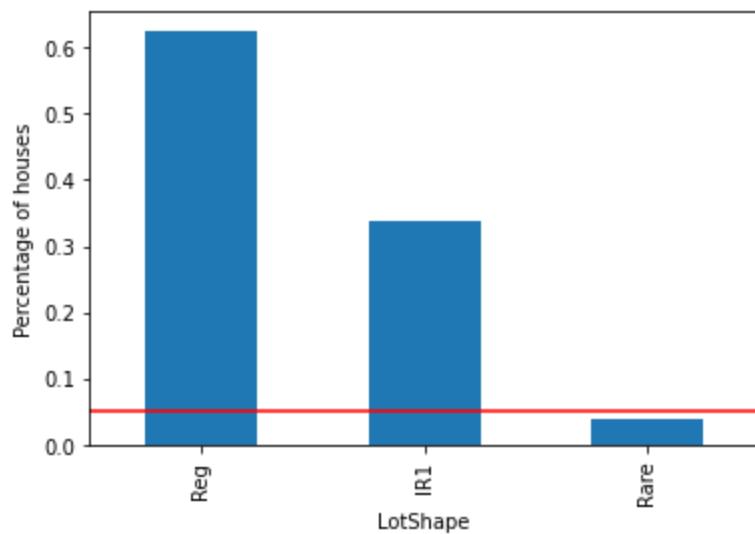
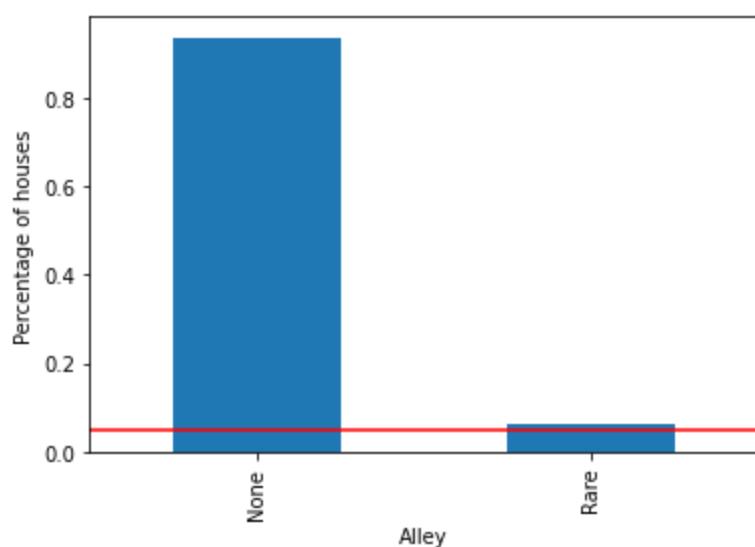
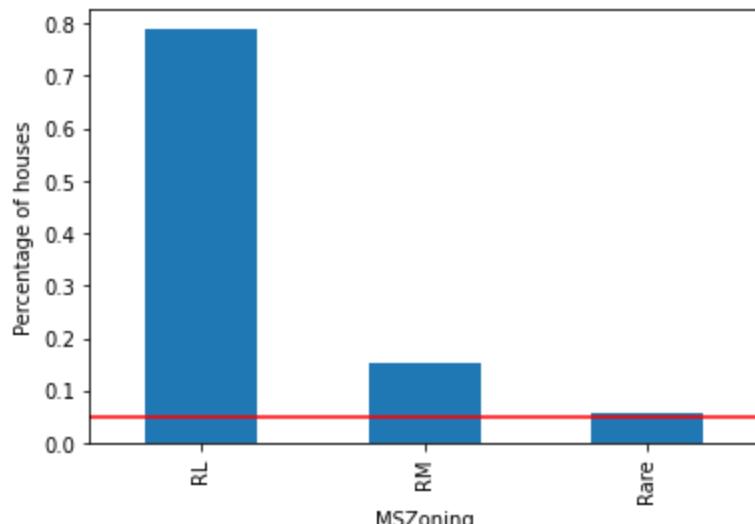
    df_test[variable] = np.where(df_test[variable].isin(
        frequent_cat), df_test[variable], 'Rare')

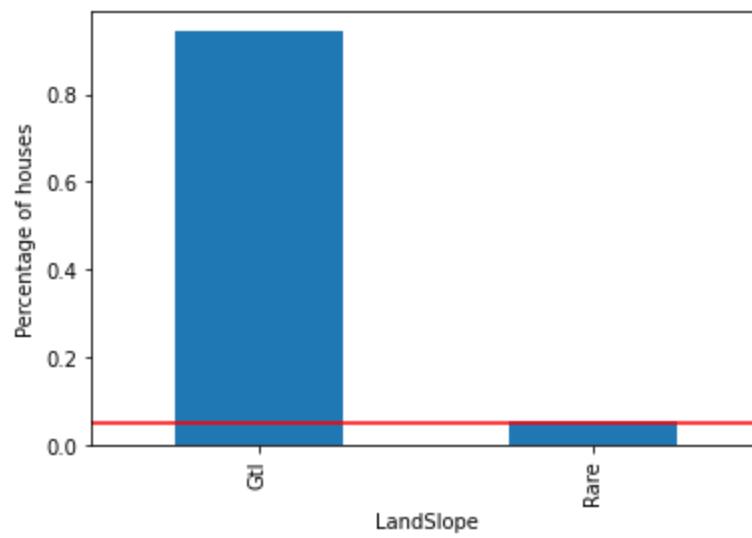
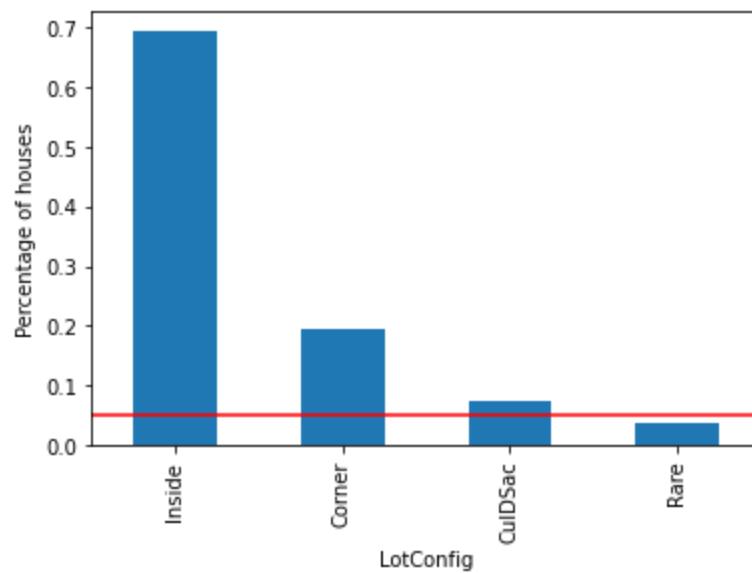
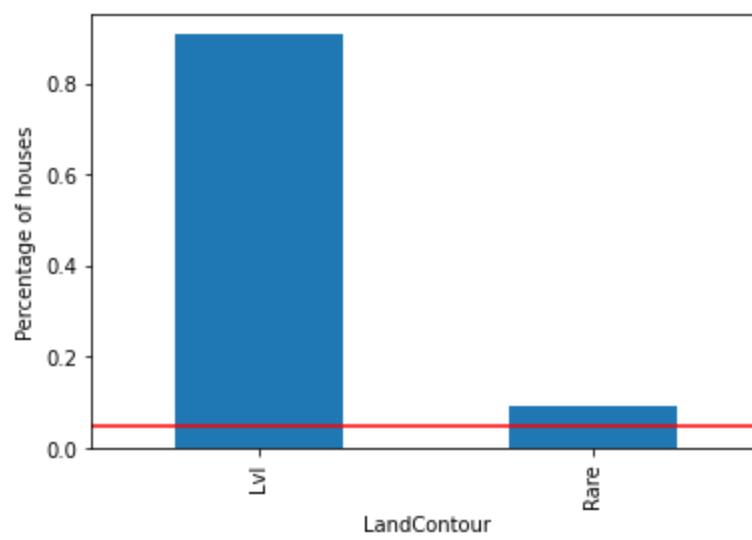
    return X_train, X_test, df_test
```

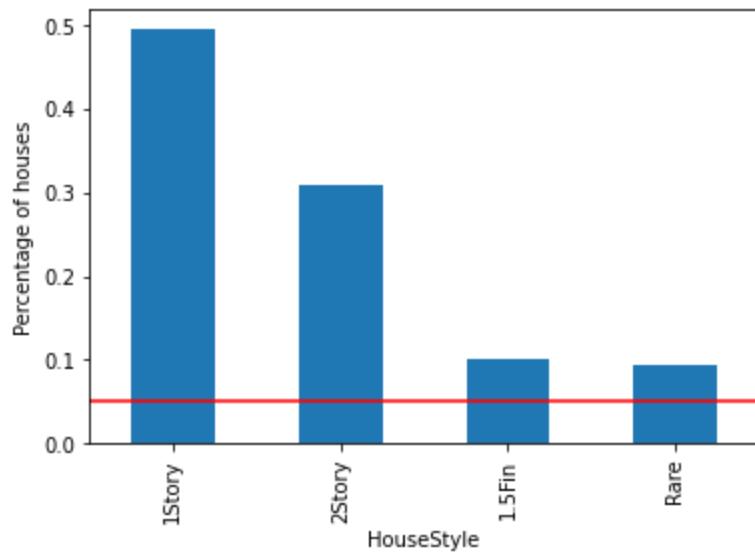
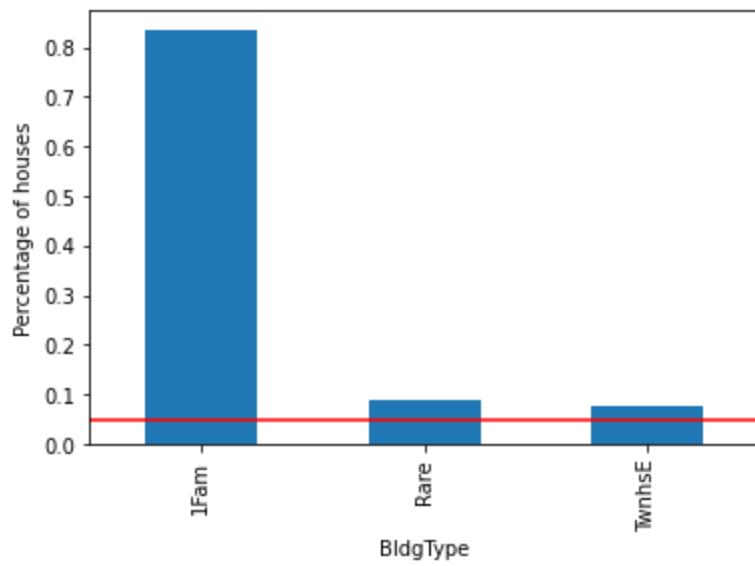
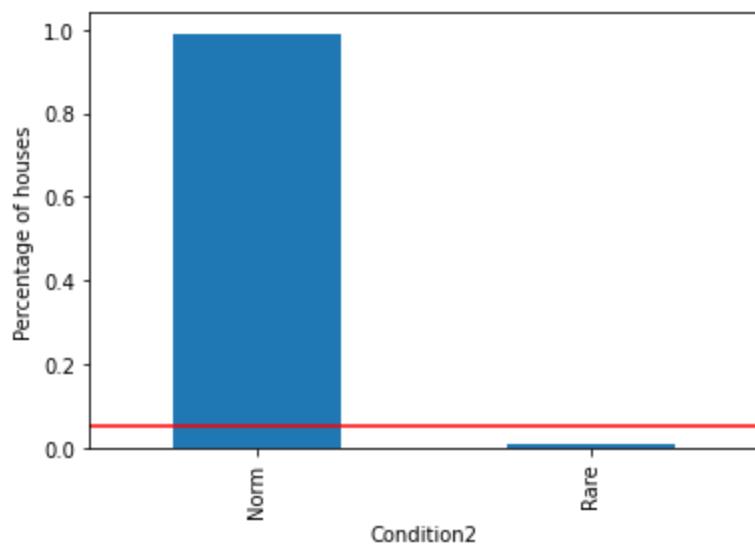
```
In [261]:  
for variable in few_label:  
    X_train, X_test, df_test = rare_encoding(X_train, X_test, df_test, variable, 0.05)
```

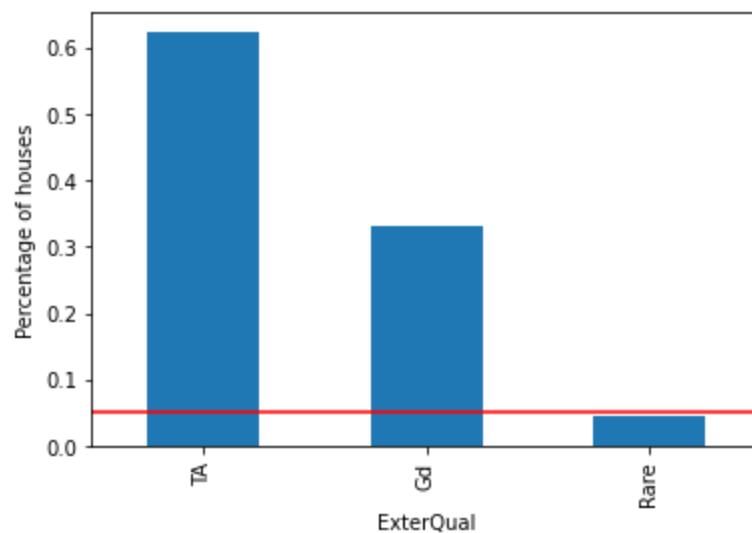
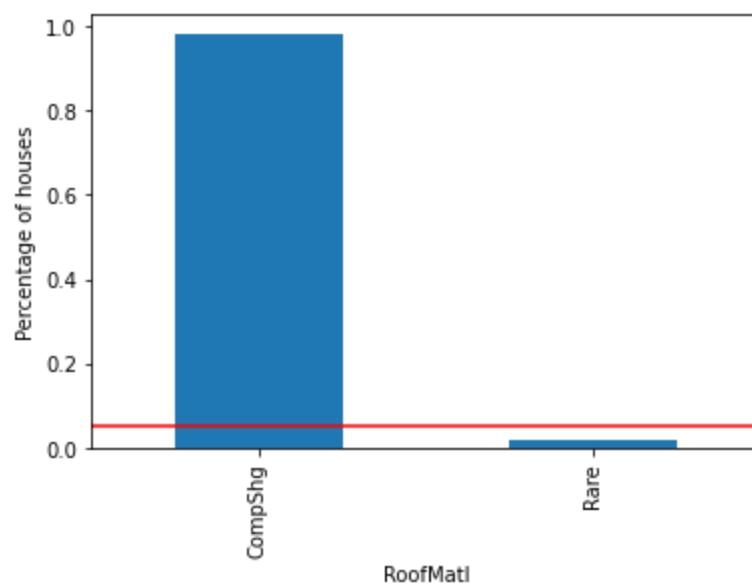
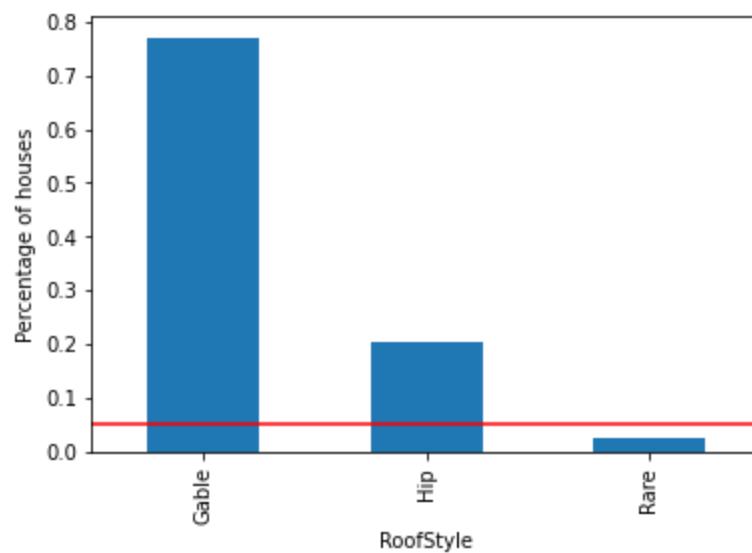
```
In [261...]  
for col in few_label:  
  
    temp_df = pd.Series(X_train[col].value_counts() / len(X_train))  
  
    fig = temp_df.sort_values(ascending=False).plot.bar()  
    fig.set_xlabel(col)  
  
    fig.axhline(v=0.05, color='red')
```

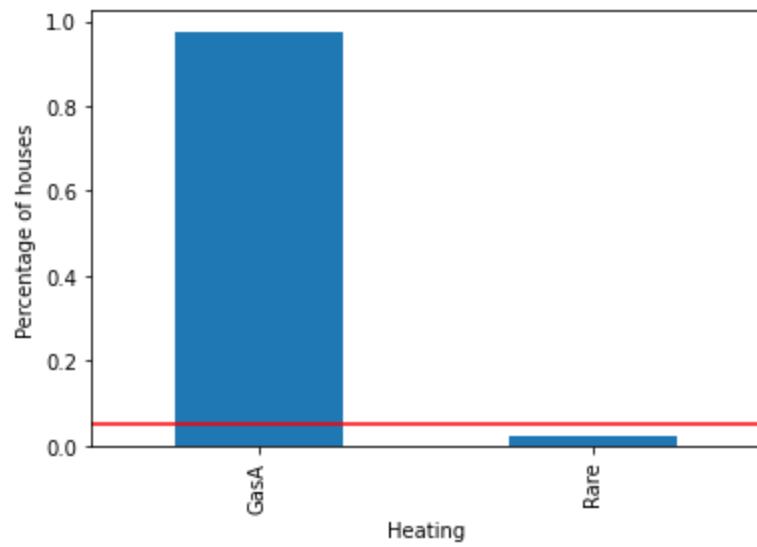
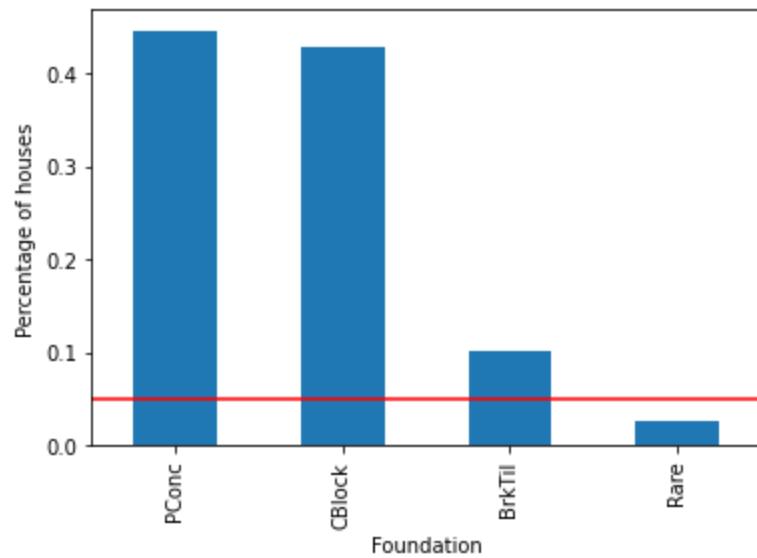
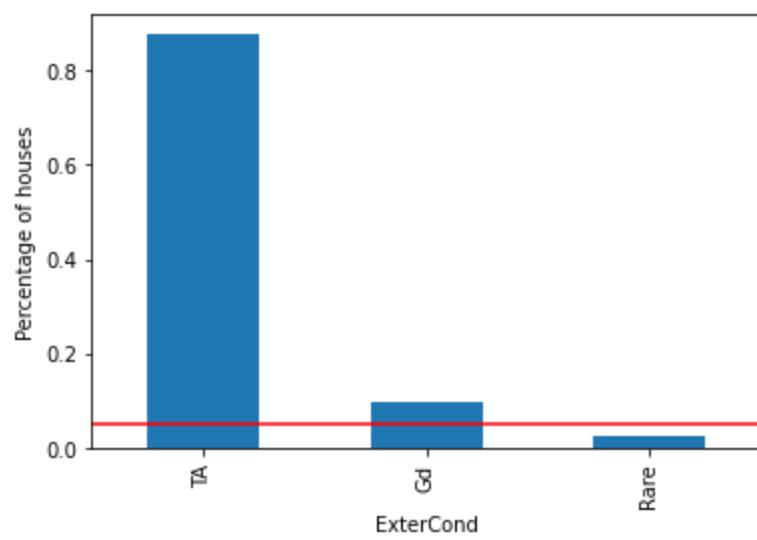
```
fig.set_ylabel('Percentage of houses')
plt.show()
```

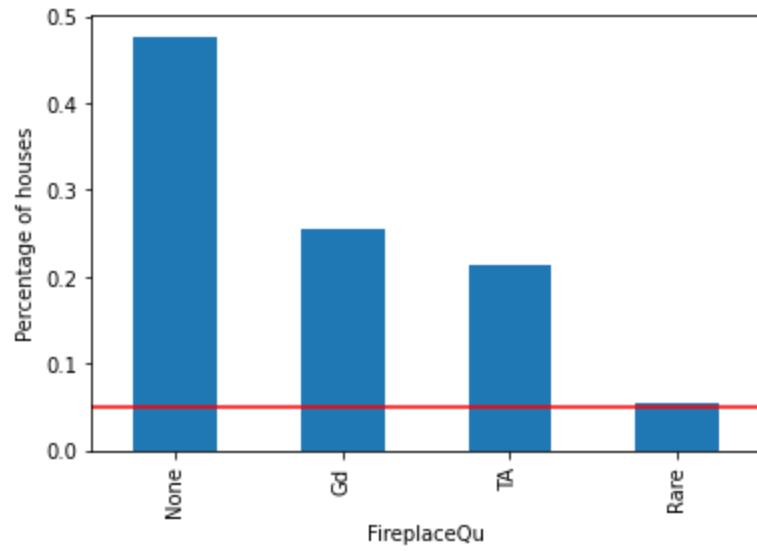
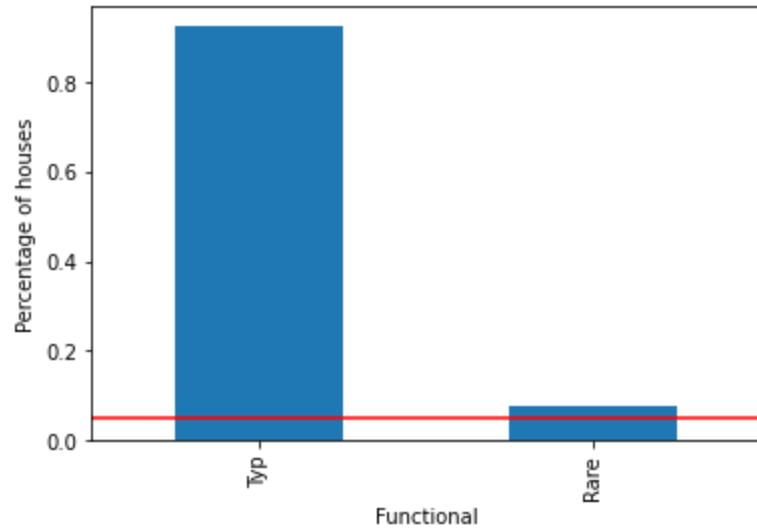
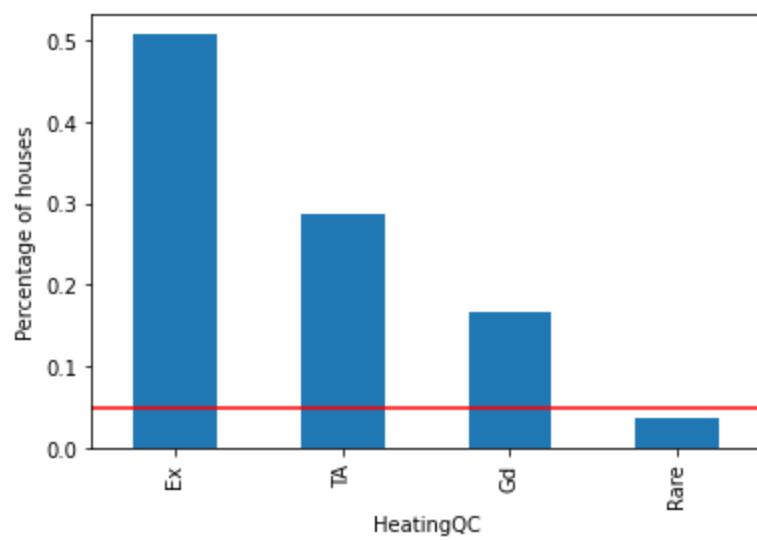


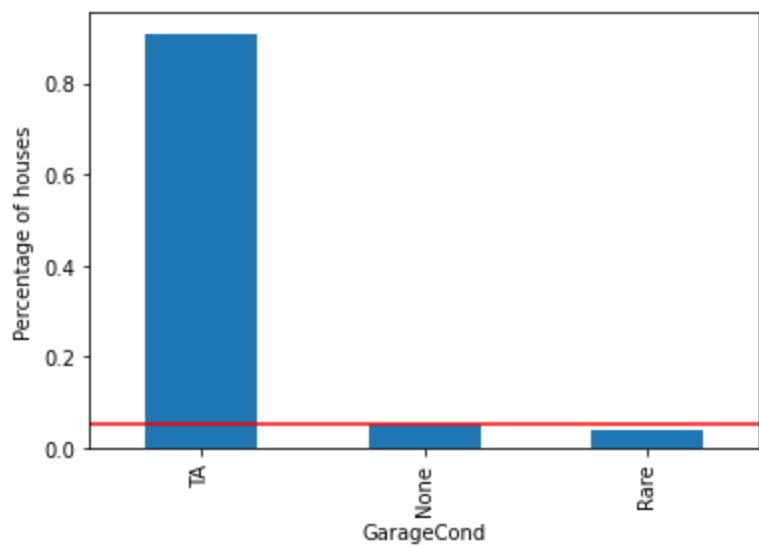
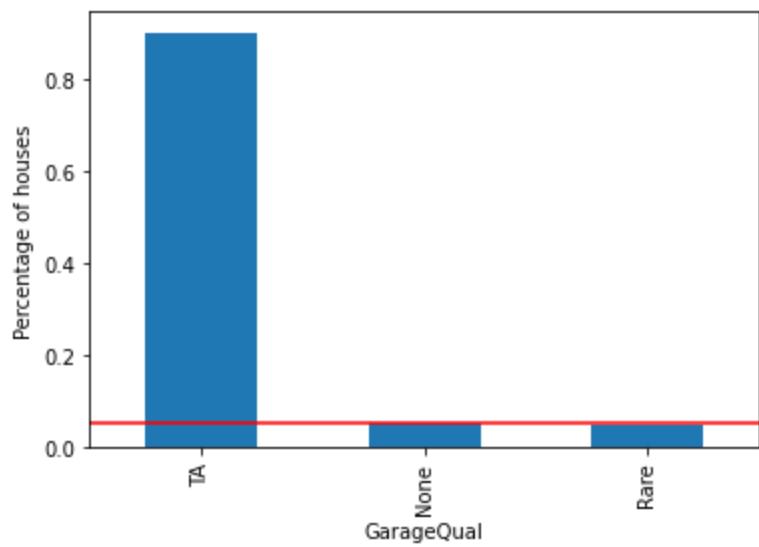
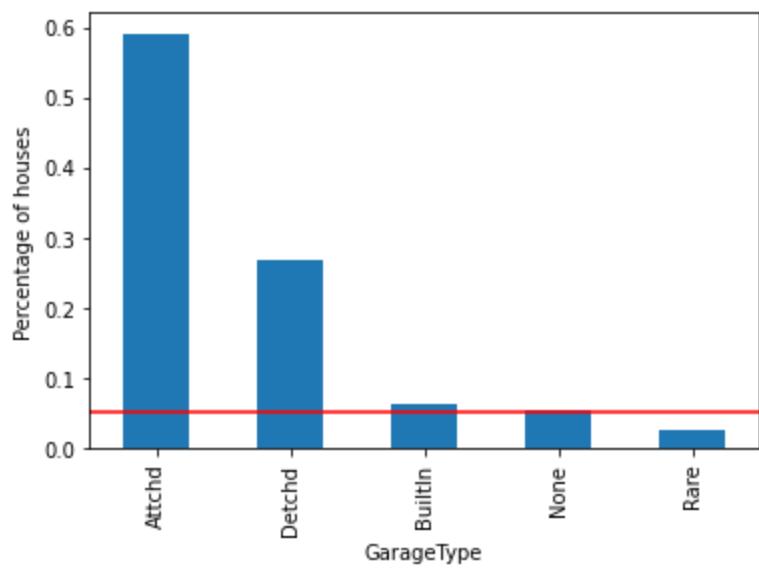


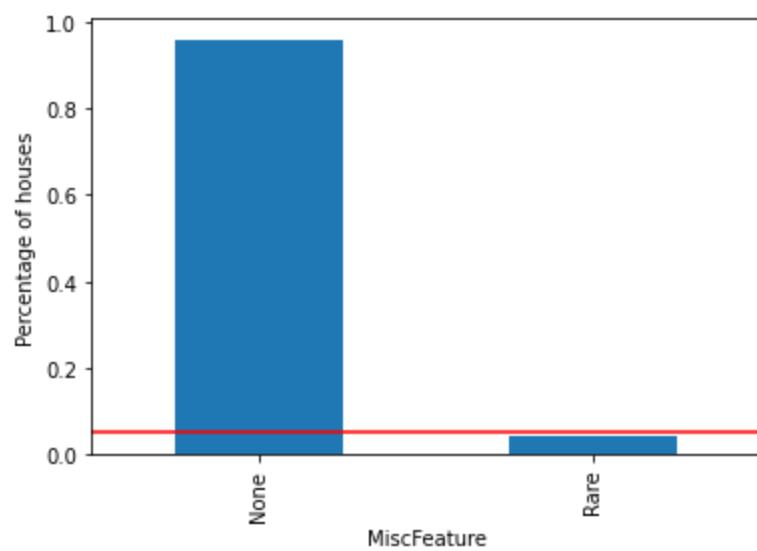
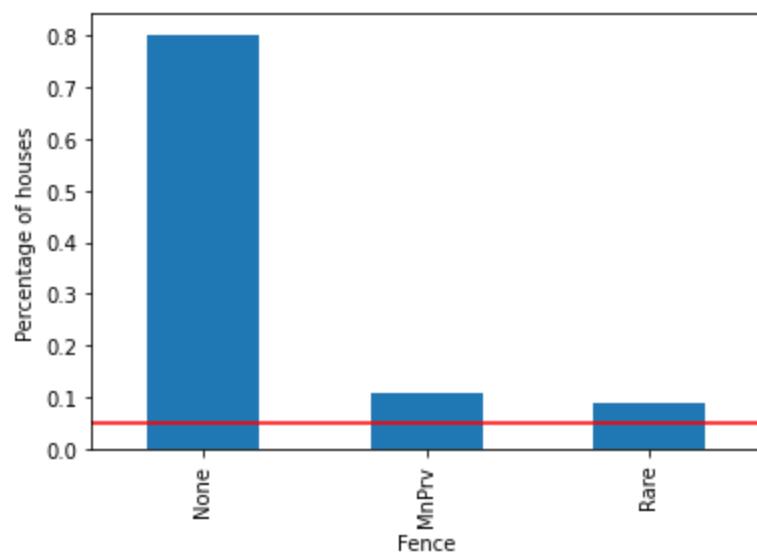
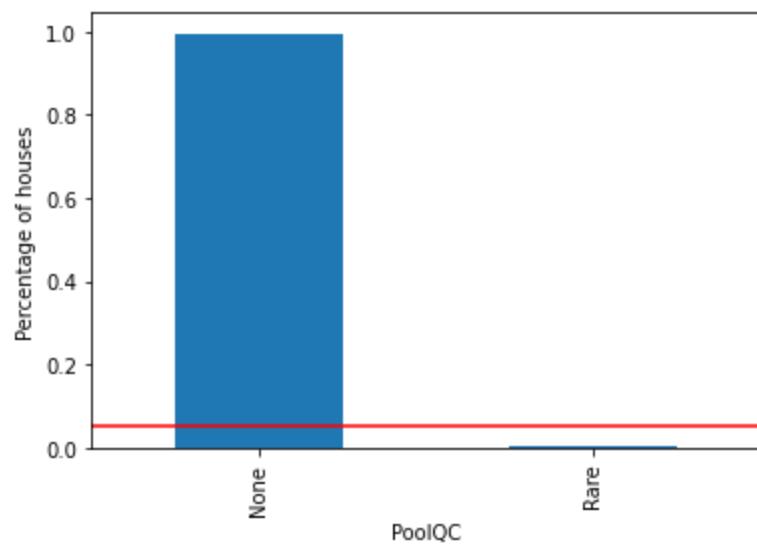


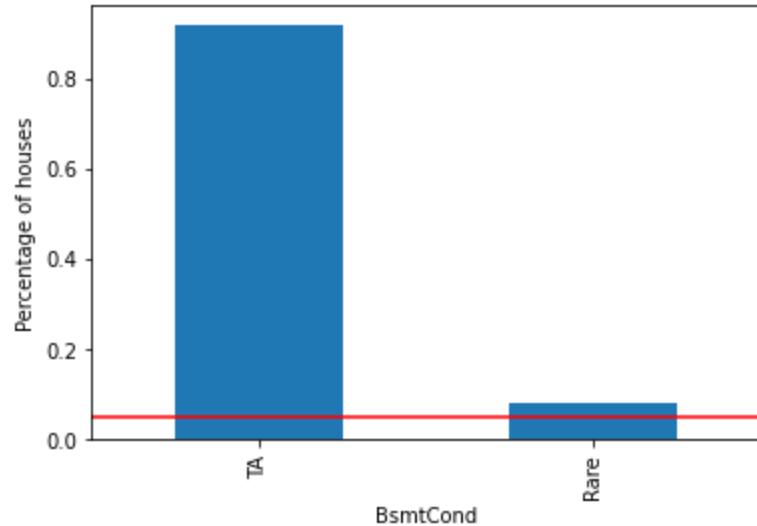
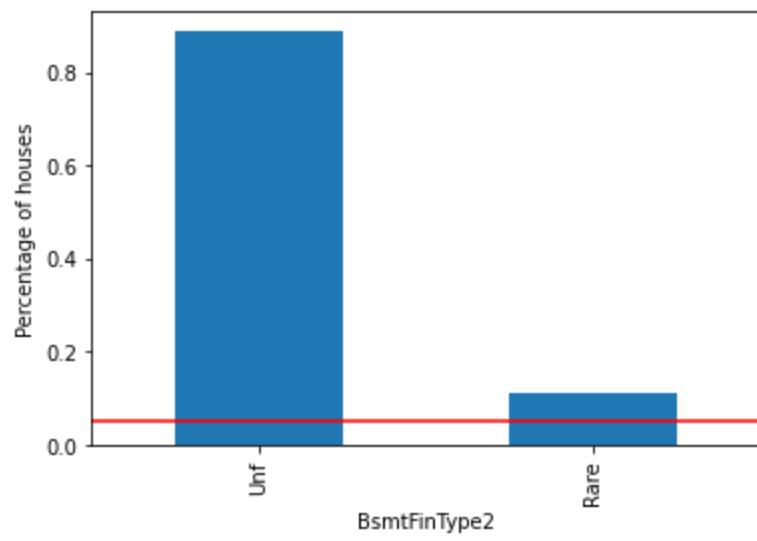
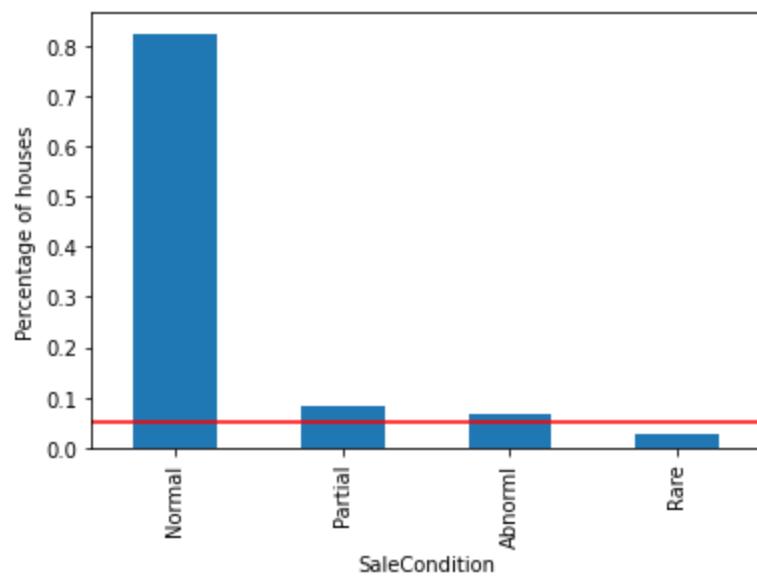


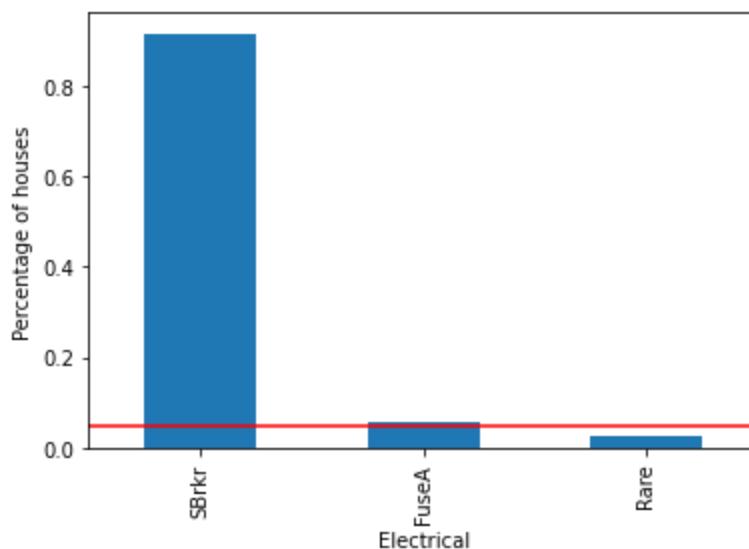












As we can see with new group Rare we can reduce the number of labels in each columns and can reduce the feature after we precess the one hot encodeing

For another feature with very high cardinality, we'll do the same as the feature has number unique less than 9 and group rare label with less than 5 %

In [261]:

```
multi_cat_cols = []
for col in X_train.columns:
    if X_train[col].dtypes == 'O':
        if X_train[col].nunique() >= 9:
            multi_cat_cols.append(col)
        print(X_train.groupby(col)[col].count() / len(X_train))
        print()
```

```
Neighborhood
Blmgtn      0.013699
Blueste     0.000978
BrDale      0.011742
BrkSide     0.040117
ClearCr     0.018591
CollgCr    0.106654
Crawfor     0.037182
Edwards     0.077299
Gilbert     0.050881
IDOTRR     0.023483
MeadowV    0.009785
Mitchel     0.033268
NAmes       0.151663
NPkVill    0.005871
NWAmes     0.052838
NoRidge     0.027397
NridgHt     0.048924
OldTown     0.080235
SWISU       0.015656
Sawyer     0.046967
SawyerW    0.037182
Somerst     0.055773
StoneBr     0.019569
Timber      0.026419
Veenker     0.007828
Name: Neighborhood, dtype: float64
```

```
Condition1
Artery      0.035225
Feedr      0.055773
Norm       0.859100
```

```
PosA      0.005871
PosN      0.013699
RRAe     0.008806
RRAn     0.015656
RRNe     0.000978
RRNn     0.004892
Name: Condition1, dtype: float64
```

```
Exterior1st
AsbShng   0.014677
AsphShn    0.000978
BrkComm    0.001957
BrkFace    0.034247
CBlock     0.000978
CemntBd    0.038160
HdBoard    0.151663
MetalSd    0.145793
Plywood    0.069472
Stone      0.000978
Stucco     0.017613
VinylSd    0.360078
Wd Sdng    0.146771
WdShing    0.016634
Name: Exterior1st, dtype: float64
```

```
Exterior2nd
AsbShng   0.015656
AsphShn    0.001957
Brk Cmn    0.005871
BrkFace    0.013699
CBlock     0.000978
CmentBd   0.037182
HdBoard    0.139922
ImStucc    0.004892
MetalSd    0.138943
Other      0.000978
Plywood    0.095890
Stone      0.002935
Stucco     0.019569
VinylSd    0.351272
Wd Sdng    0.140900
Wd Shng    0.029354
Name: Exterior2nd, dtype: float64
```

```
SaleType
COD       0.026419
CWD       0.003914
Con       0.001957
ConLD     0.006849
ConLI     0.000978
ConLw     0.002935
New       0.082192
Oth       0.000978
WD        0.873777
Name: SaleType, dtype: float64
```

```
In [261...]  
for variable in multi_cat_cols:  
  
    X_train, X_test, df_test = rare_encoding(X_train, X_test, df_test, variable, 0.05)
```

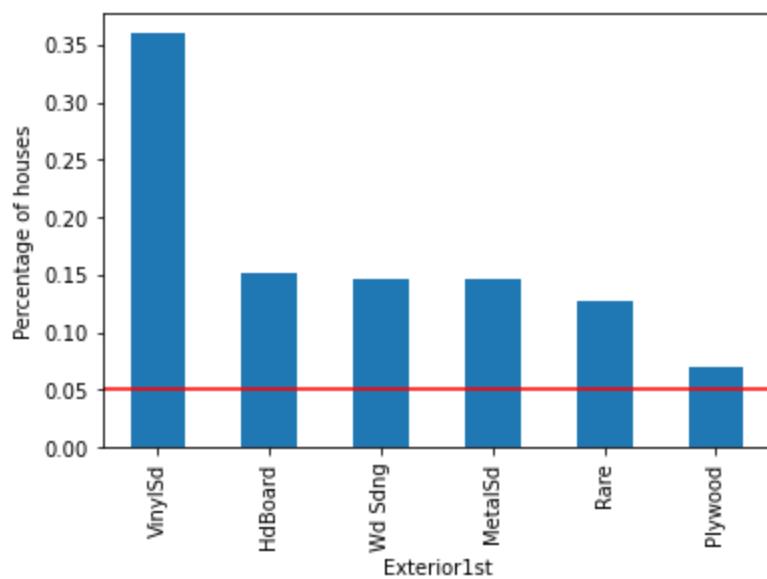
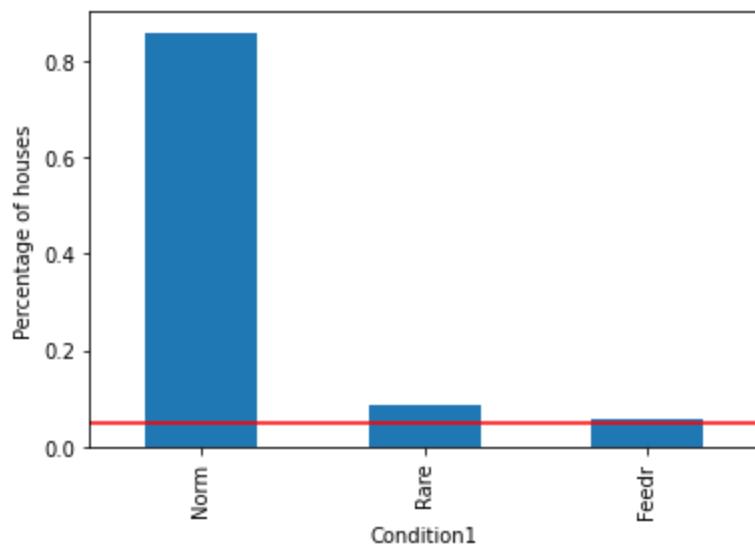
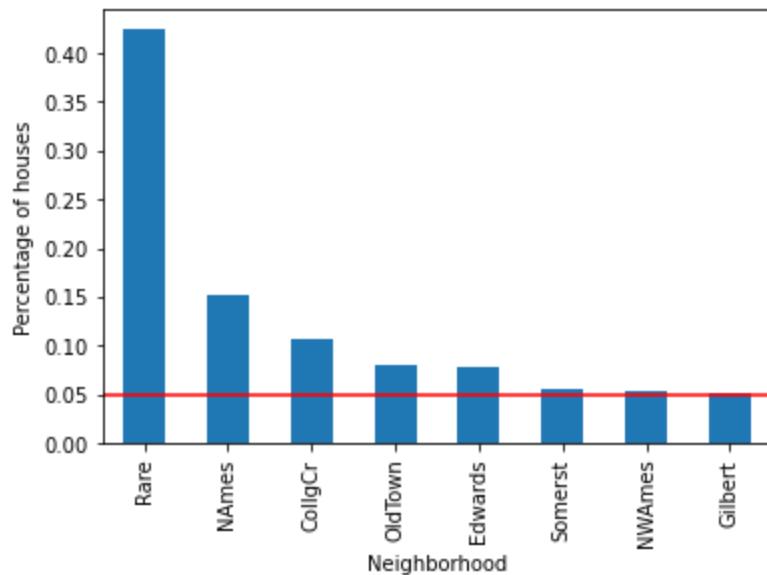
```
In [262...]  
for col in multi_cat_cols:  
  
    temp_df = pd.Series(X_train[col].value_counts() / len(X_train) )
```

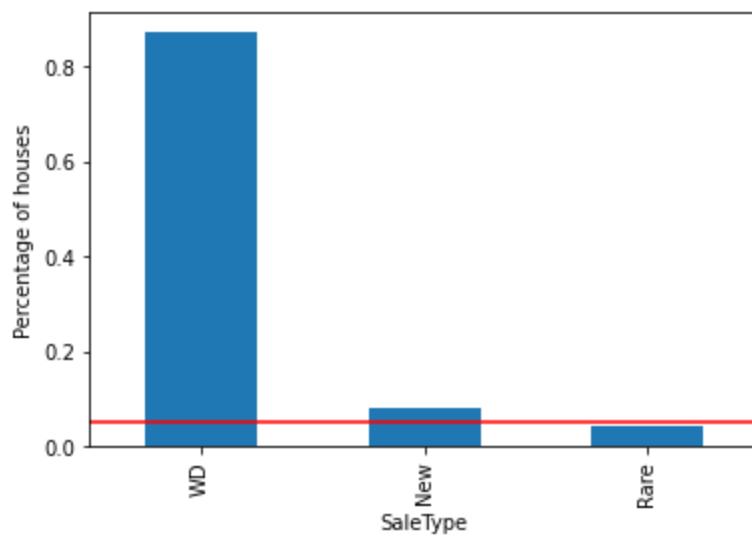
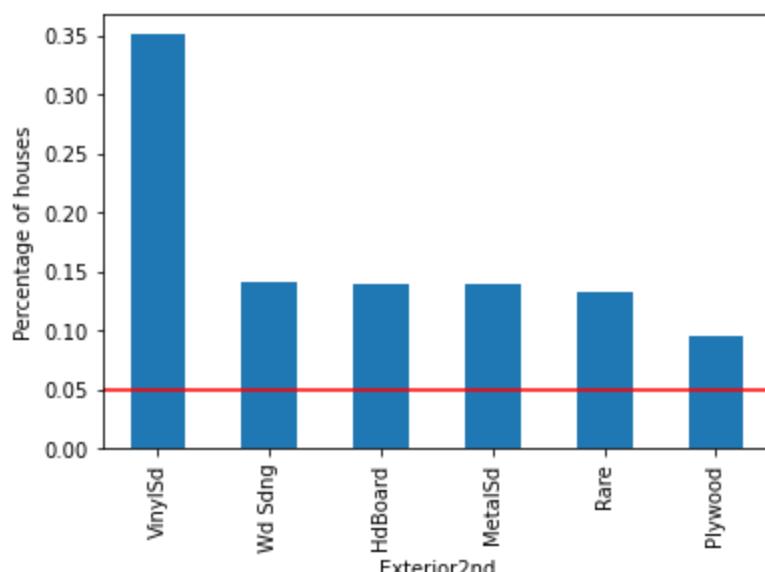
```

fig = temp_df.sort_values(ascending=False).plot.bar()
fig.set_xlabel(col)

fig.axhline(y=0.05, color='red')
fig.set_ylabel('Percentage of houses')
plt.show()

```





```
In [262]: order_col=['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond', 'HeatingQC', 'KitchenQual', 'Garage
```

```
In [262]: X_train.drop('SalePrice', axis=1, inplace=True)
X_test.drop('SalePrice', axis=1, inplace=True)
```

For some columnne with order category in X_train dataset we'll encode by OrdinalEncoder and transform into X_test and df_test

```
In [262]: order_encode = OrdinalEncoder(encoding_method='ordered',
                                    variables=order_col)
order_encode.fit(X_train, y_train)
```

```
Out[262]: OrdinalEncoder(variables=['ExterQual', 'ExterCond', 'BsmtQual', 'BsmtCond',
                                     'HeatingQC', 'KitchenQual', 'GarageQual',
                                     'GarageCond', 'LotShape', 'LandSlope', 'BsmtExposure',
                                     'BsmtFinType1', 'BsmtFinType2', 'Functional',
                                     'GarageFinish', 'PavedDrive', 'Utilities',
                                     'CentralAir', 'Electrical', 'Lotsize', 'Houseseize'])
```

```
In [262]: order_encode.encoder_dict_
```

```
Out[262]: {'ExterQual': {'TA': 0, 'Gd': 1, 'Rare': 2},
            'ExterCond': {'Rare': 0, 'Gd': 1, 'TA': 2},
            'BsmtQual': {'Fa': 0, 'TA': 1, 'Gd': 2, 'Ex': 3},
```

```
'BsmtCond': {'Rare': 0, 'TA': 1},
'HeatingQC': {'Rare': 0, 'TA': 1, 'Gd': 2, 'Ex': 3},
'KitchenQual': {'Fa': 0, 'TA': 1, 'Gd': 2, 'Ex': 3},
'GarageQual': {'None': 0, 'Rare': 1, 'TA': 2},
'GarageCond': {'None': 0, 'Rare': 1, 'TA': 2},
'LotShape': {'Reg': 0, 'IR1': 1, 'Rare': 2},
'LandSlope': {'Gtl': 0, 'Rare': 1},
'BsmtExposure': {'No': 0, 'Mn': 1, 'Av': 2, 'Gd': 3},
'BsmtFinType1': {'BLQ': 0, 'LwQ': 1, 'Rec': 2, 'ALQ': 3, 'Unf': 4, 'GLQ': 5},
'BsmtFinType2': {'Rare': 0, 'Unf': 1},
'Functional': {'Rare': 0, 'Typ': 1},
'GarageFinish': {'None': 0, 'Unf': 1, 'RFn': 2, 'Fin': 3},
'PavedDrive': {'N': 0, 'P': 1, 'Y': 2},
'Utilities': {'NoSeWa': 0, 'AllPub': 1},
'CentralAir': {'N': 0, 'Y': 1},
'Electrical': {'Rare': 0, 'FuseA': 1, 'SBrkr': 2},
'Lotsize': {1: 0, 0: 1, 2: 2, 3: 3},
'Housesize': {0: 0, 3: 1, 1: 2, 2: 3}}
```

In [262...]

```
X_train=order_encode.transform(X_train)
X_test=order_encode.transform(X_test)
df_test=order_encode.transform(df_test)
```

For other category will using the one hot encoding

In [262...]

```
multi_cat_cols = []
for col in X_train.columns:
    if X_train[col].dtypes == 'O':
        multi_cat_cols.append(col)
        print(X_train.groupby(col)[col].count() / len(X_train))
        print()
```

```
MSZoning
RL      0.789628
RM      0.151663
Rare    0.058708
Name: MSZoning, dtype: float64

Street
Grvl    0.003914
Pave    0.996086
Name: Street, dtype: float64
```

```
Alley
None    0.935421
Rare    0.064579
Name: Alley, dtype: float64
```

```
LandContour
Lvl     0.908023
Rare    0.091977
Name: LandContour, dtype: float64
```

```
LotConfig
Corner   0.194716
CulDSac  0.073386
Inside   0.694716
Rare     0.037182
Name: LotConfig, dtype: float64
```

```
Neighborhood
CollgCr  0.106654
Edwards   0.077299
Gilbert   0.050881
```

NAmes 0.151663
NWAmes 0.052838
OldTown 0.080235
Rare 0.424658
Somerst 0.055773
Name: Neighborhood, dtype: float64

Condition1
Feedr 0.055773
Norm 0.859100
Rare 0.085127
Name: Condition1, dtype: float64

Condition2
Norm 0.991194
Rare 0.008806
Name: Condition2, dtype: float64

BldgType
1Fam 0.834638
Rare 0.087084
TwnhsE 0.078278
Name: BldgType, dtype: float64

HouseStyle
1.5Fin 0.101761
1Story 0.495108
2Story 0.309198
Rare 0.093933
Name: HouseStyle, dtype: float64

RoofStyle
Gable 0.771037
Hip 0.203523
Rare 0.025440
Name: RoofStyle, dtype: float64

RoofMatl
CompShg 0.981409
Rare 0.018591
Name: RoofMatl, dtype: float64

Exterior1st
HdBoard 0.151663
MetalSd 0.145793
Plywood 0.069472
Rare 0.126223
VinylSd 0.360078
Wd Sdng 0.146771
Name: Exterior1st, dtype: float64

Exterior2nd
HdBoard 0.139922
MetalSd 0.138943
Plywood 0.095890
Rare 0.133072
VinylSd 0.351272
Wd Sdng 0.140900
Name: Exterior2nd, dtype: float64

Foundation
BrkTil 0.100783
CBlock 0.427593
PConc 0.446184
Rare 0.025440
Name: Foundation, dtype: float64

```
Heating
GasA      0.975538
Rare      0.024462
Name: Heating, dtype: float64

FireplaceQu
Gd        0.255382
None     0.476517
Rare      0.053816
TA        0.214286
Name: FireplaceQu, dtype: float64

GarageType
Attchd    0.590998
BuiltIn   0.062622
Detchd   0.269080
None      0.052838
Rare      0.024462
Name: GarageType, dtype: float64

PoolQC
None     0.995108
Rare     0.004892
Name: PoolQC, dtype: float64

Fence
MnPrv    0.109589
None     0.802348
Rare     0.088063
Name: Fence, dtype: float64

MiscFeature
None     0.960861
Rare     0.039139
Name: MiscFeature, dtype: float64

SaleType
New      0.082192
Rare     0.044031
WD       0.873777
Name: SaleType, dtype: float64

SaleCondition
Abnorml  0.066536
Normal   0.824853
Partial   0.083170
Rare     0.025440
Name: SaleCondition, dtype: float64

MasVnrType
BrkCmn   0.011742
BrkFace  0.321918
None     0.577299
Stone    0.089041
Name: MasVnrType, dtype: float64
```

In [262]:

```
multi_cat_cols
```

Out[262]:

```
['MSZoning',
 'Street',
 'Alley',
 'LandContour',
 'LotConfig',
```

```
'Neighborhood',
'Condition1',
'Condition2',
'BldgType',
'HouseStyle',
'RoofStyle',
'RoofMatl',
'Exterior1st',
'Exterior2nd',
'Foundation',
'Heating',
'FireplaceQu',
'GarageType',
'PoolQC',
'Fence',
'MiscFeature',
'SaleType',
'SaleCondition',
'MasVnrType']
```

In [262...]

```
ohe_encode =OneHotEncoder(
    top_categories=None,
    drop_last=True)
```

In [262...]

```
ohe_encode.fit(X_train)
```

Out[262...]

```
OneHotEncoder(drop_last=True)
```

In [263...]

```
ohe_encode.variables_
```

Out[263...]

```
['MSZoning',
 'Street',
 'Alley',
 'LandContour',
 'LotConfig',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'Foundation',
 'Heating',
 'FireplaceQu',
 'GarageType',
 'PoolQC',
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition',
 'MasVnrType',
 'GroupYearBuilt']
```

In [263...]

```
X_train=ohe_encode.transform(X_train)
X_test=ohe_encode.transform(X_test)
df_test=ohe_encode.transform(df_test)
```

Data Transformation and scaling

basically for building the regression model, need the assumption of feature to be linear and As the dataset with skew, some outliers and difference of range data, need to scale and transform data to normal distribution before we'll do the next process of built model

In [263]...

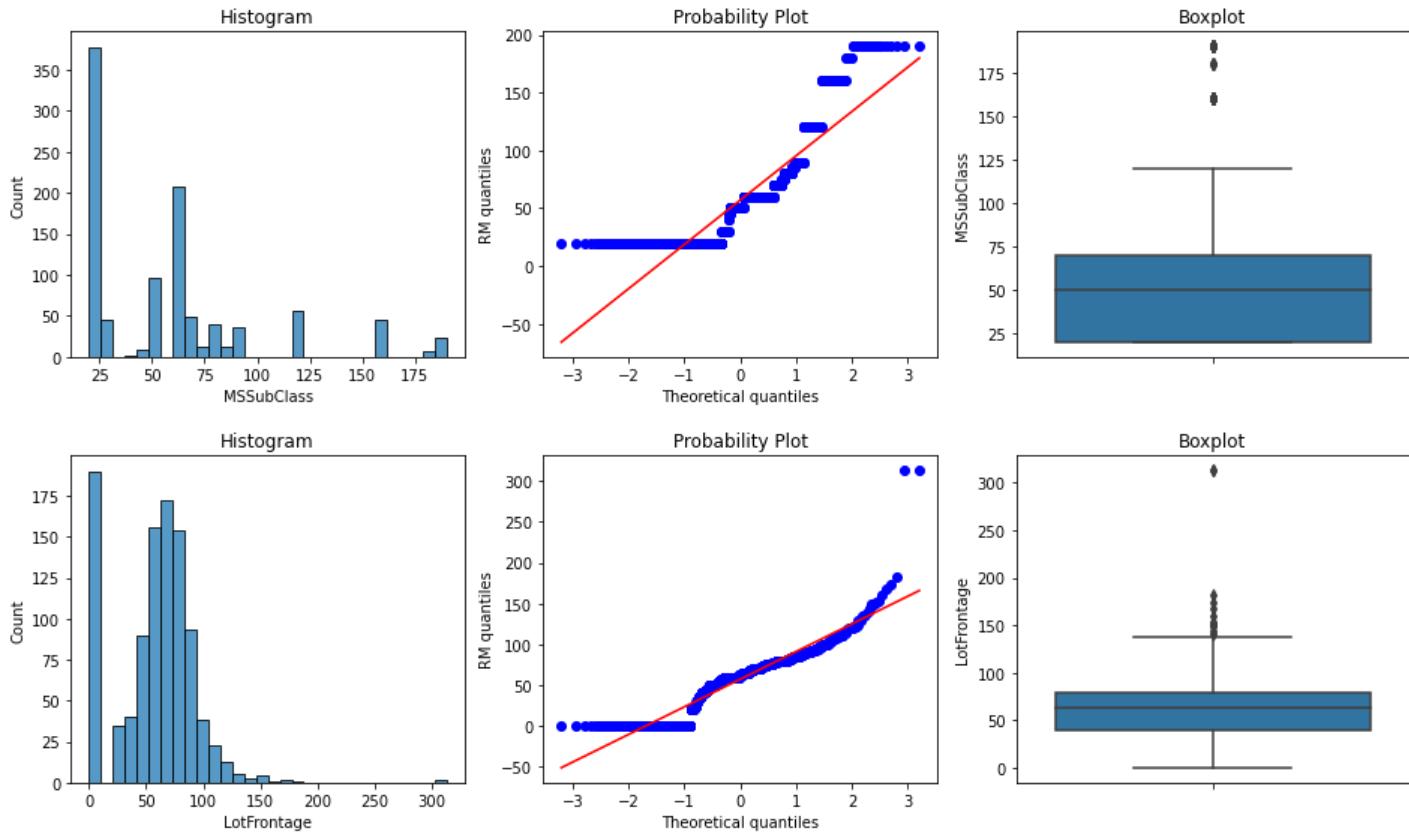
```
def diagnostic_plots(df, variable):
    plt.figure(figsize=(16, 4))
    plt.subplot(1, 3, 1)
    sns.histplot(df[variable], bins=30)
    plt.title('Histogram')
    plt.subplot(1, 3, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.ylabel('RM quantiles')
    plt.subplot(1, 3, 3)
    sns.boxplot(y=df[variable])
    plt.title('Boxplot')

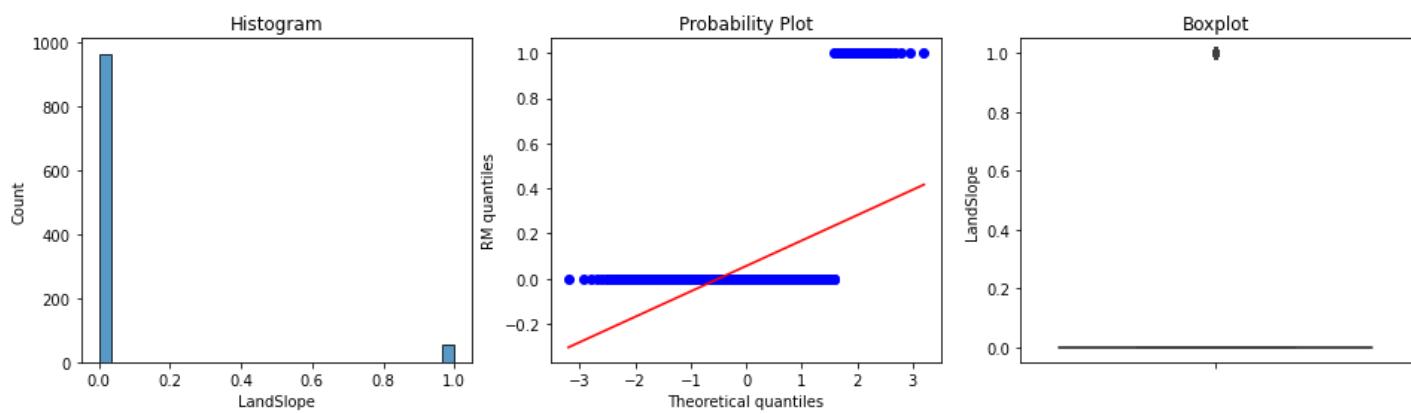
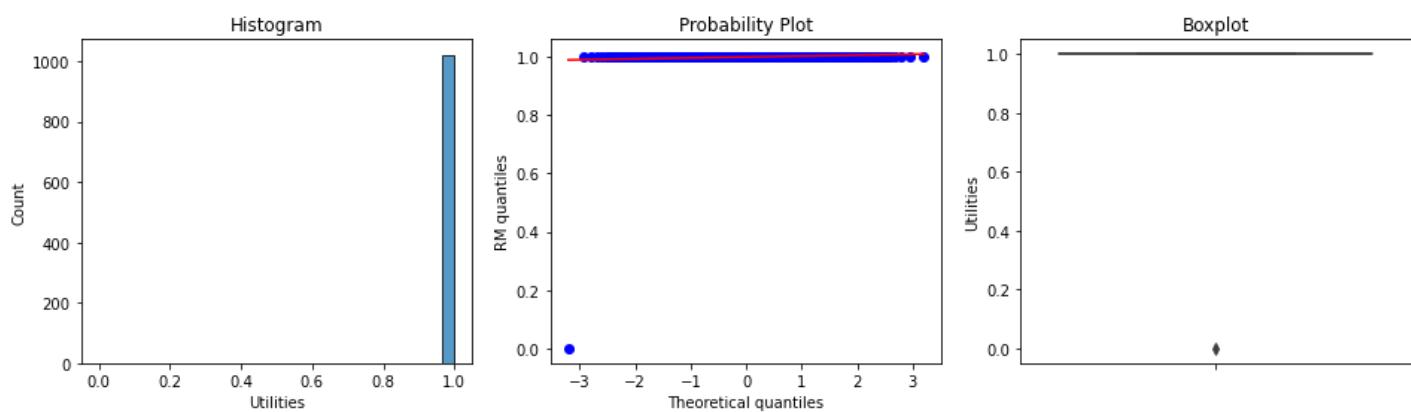
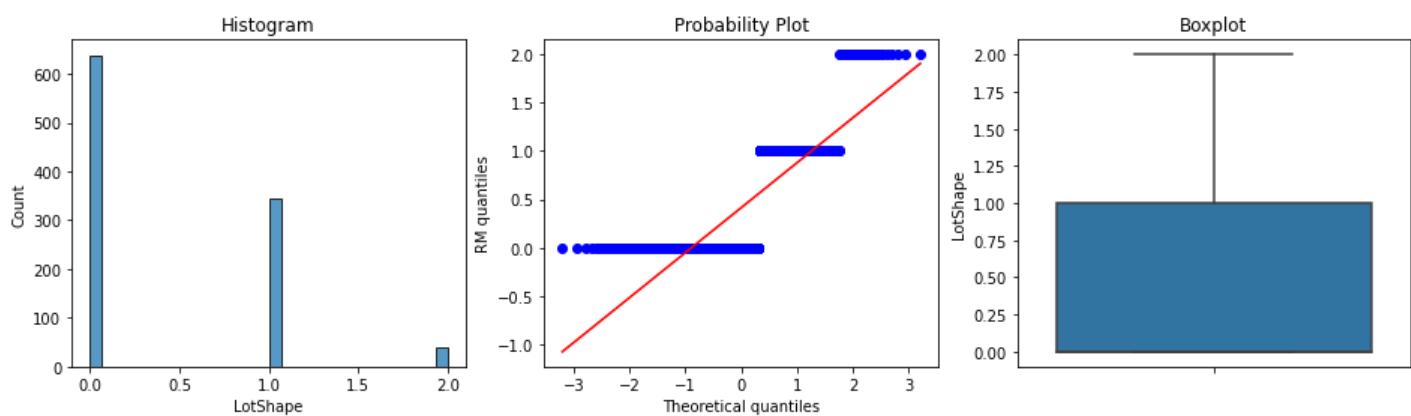
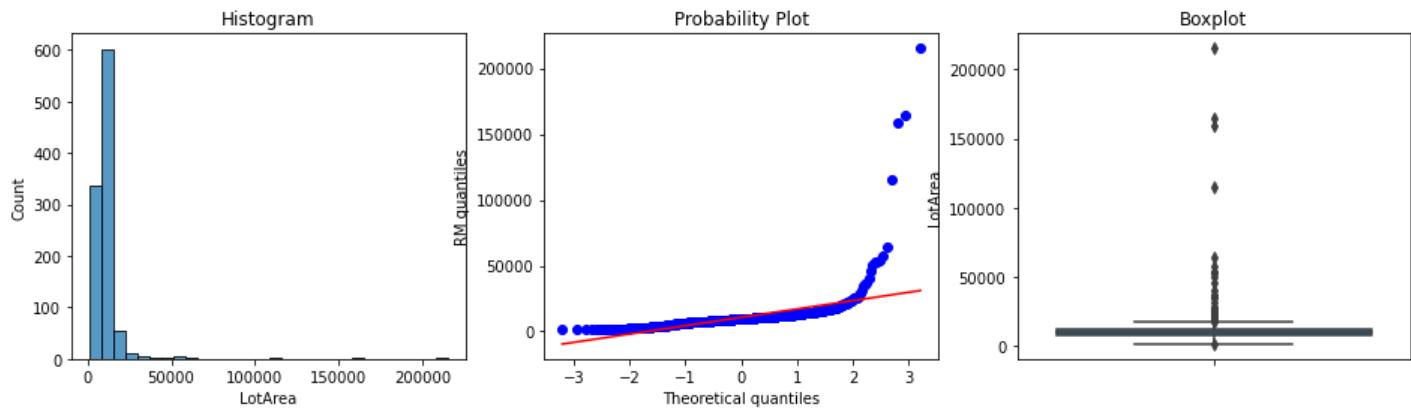
    plt.show()
```

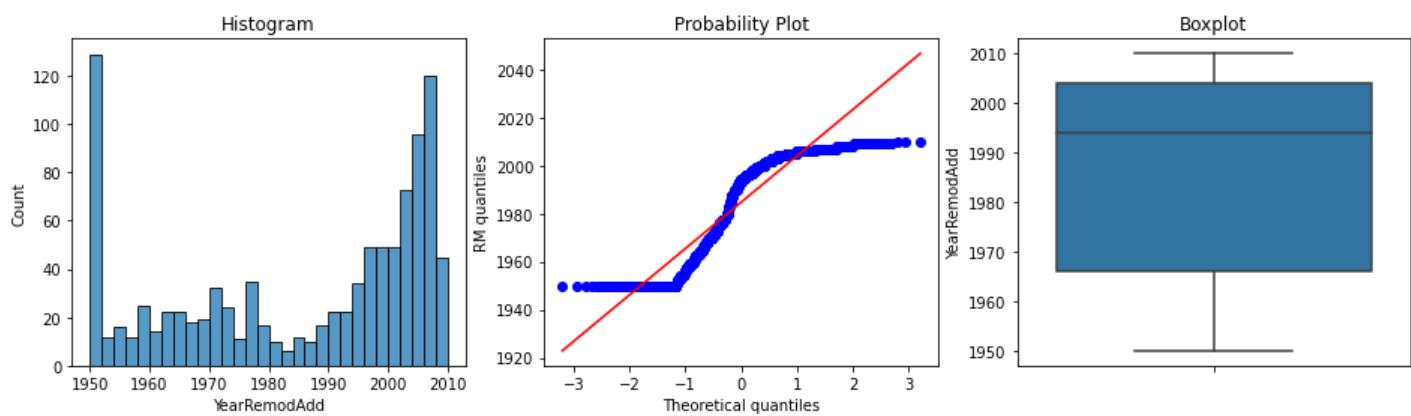
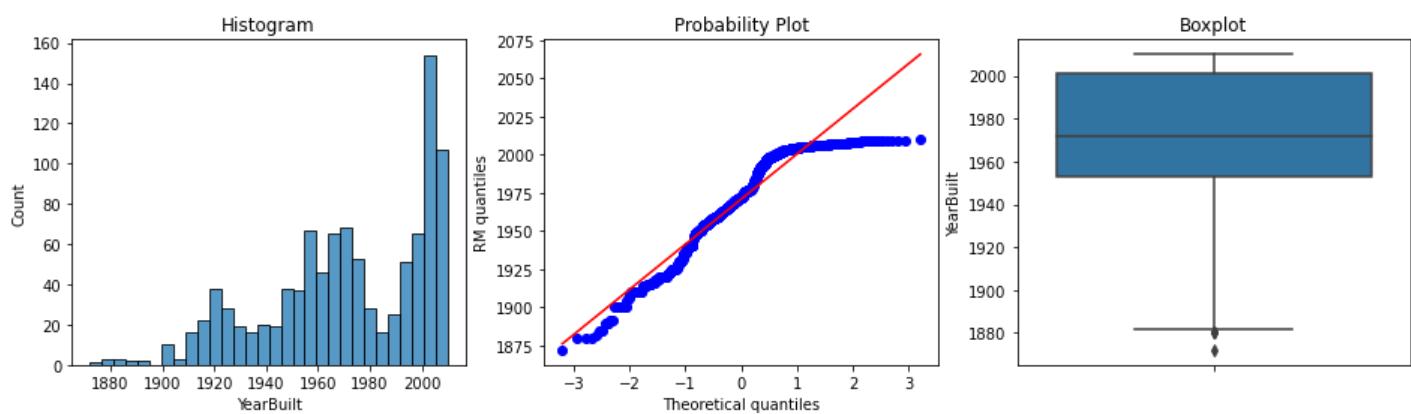
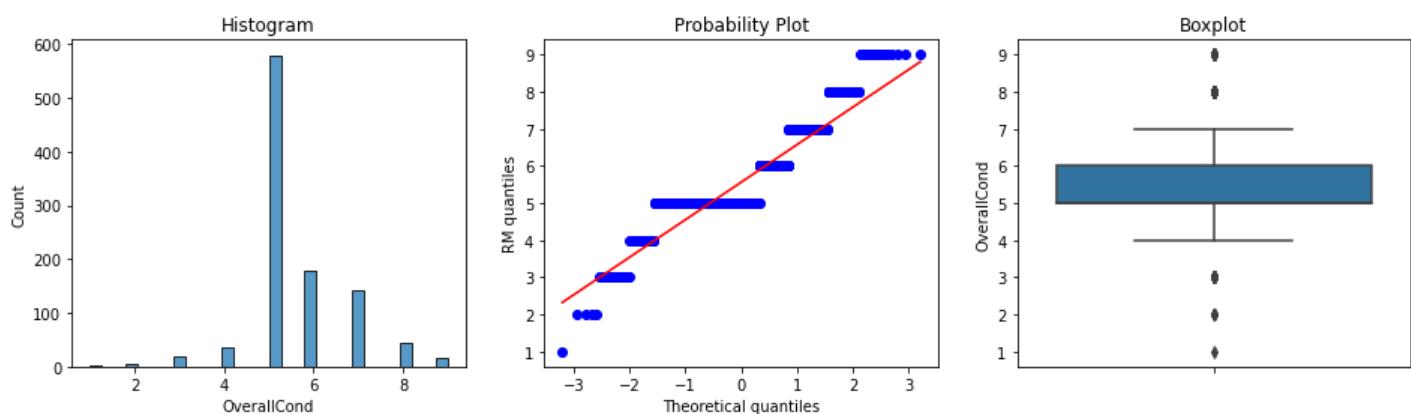
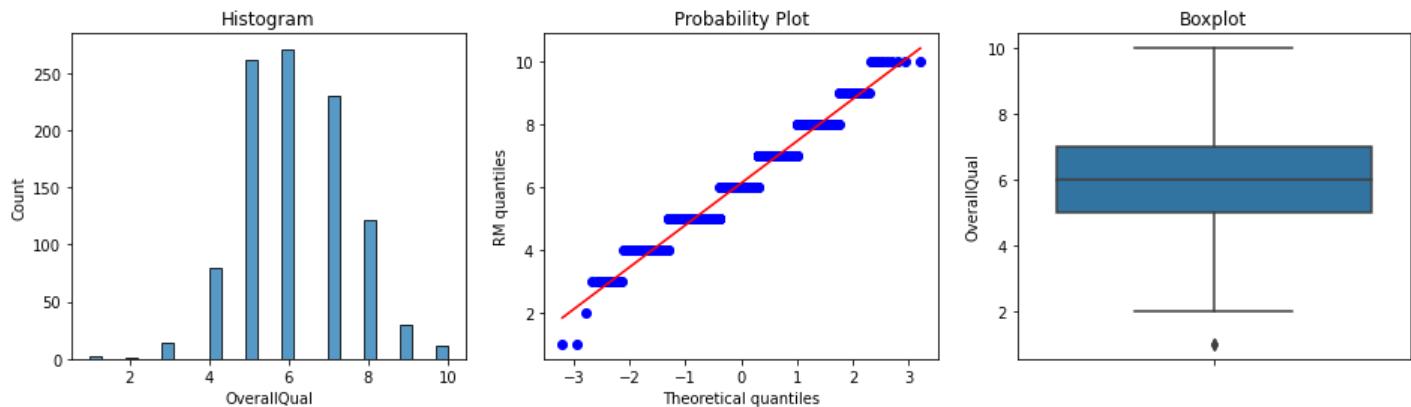
The chart below show the distribution, outlier and feature linear assumption

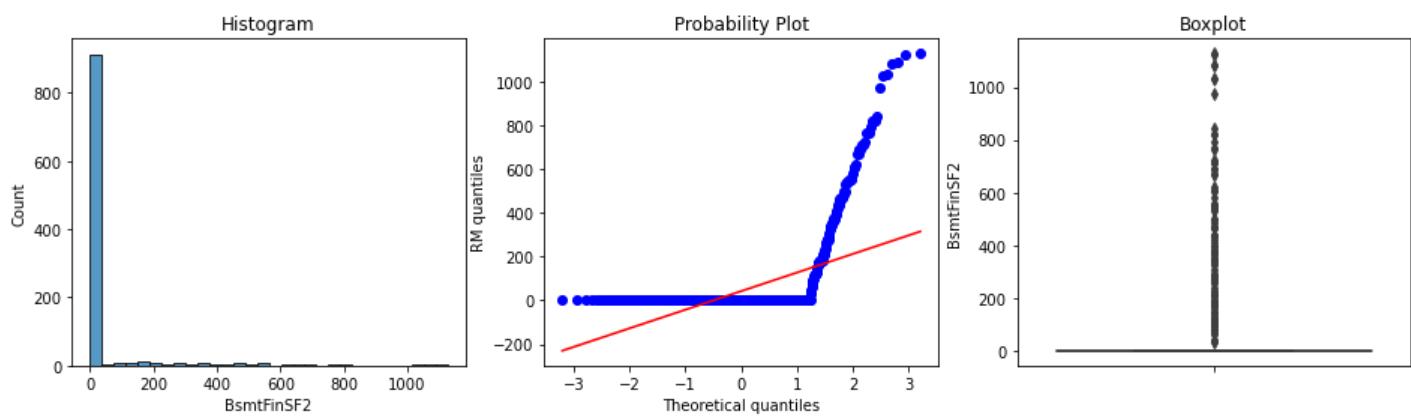
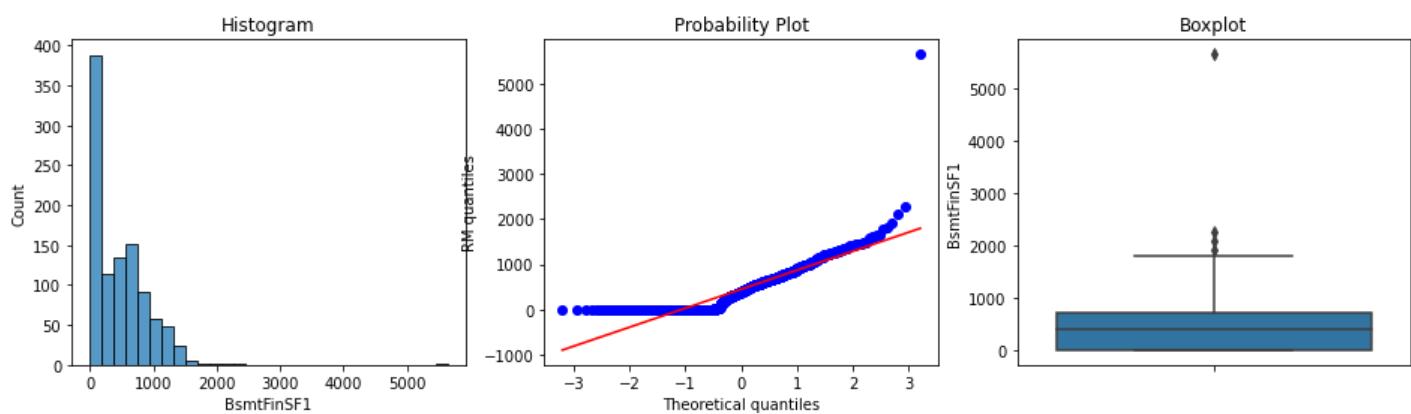
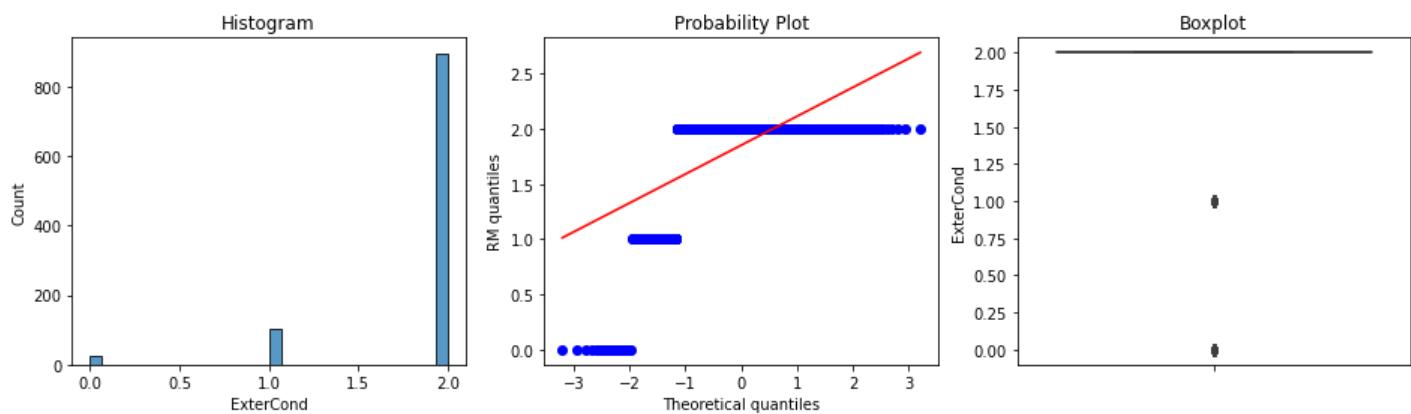
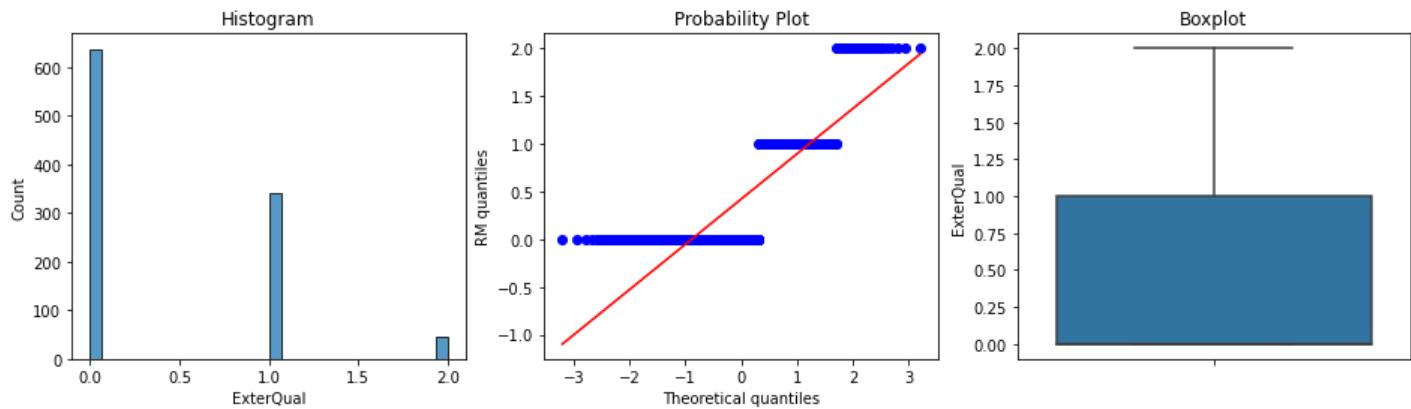
In [263]...

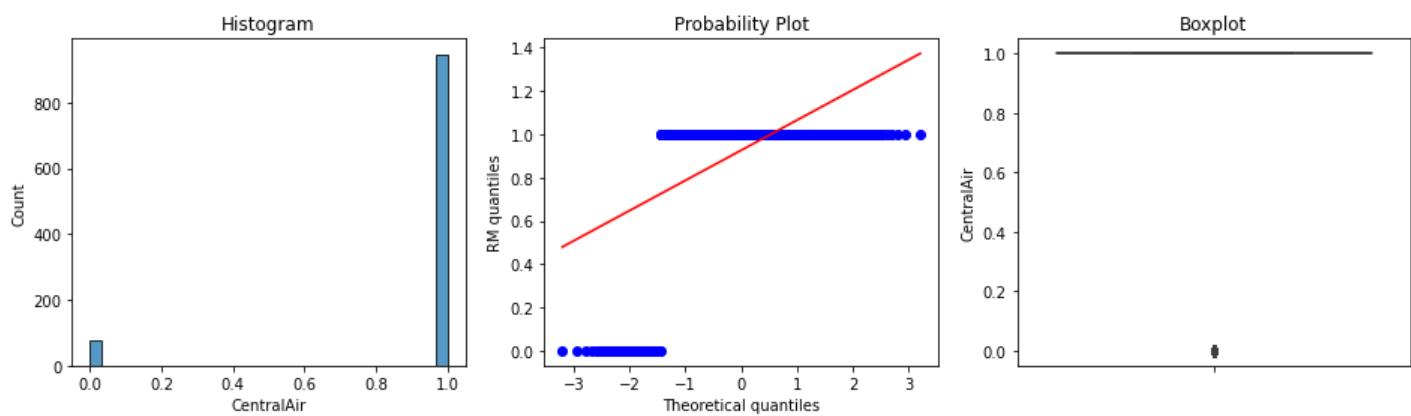
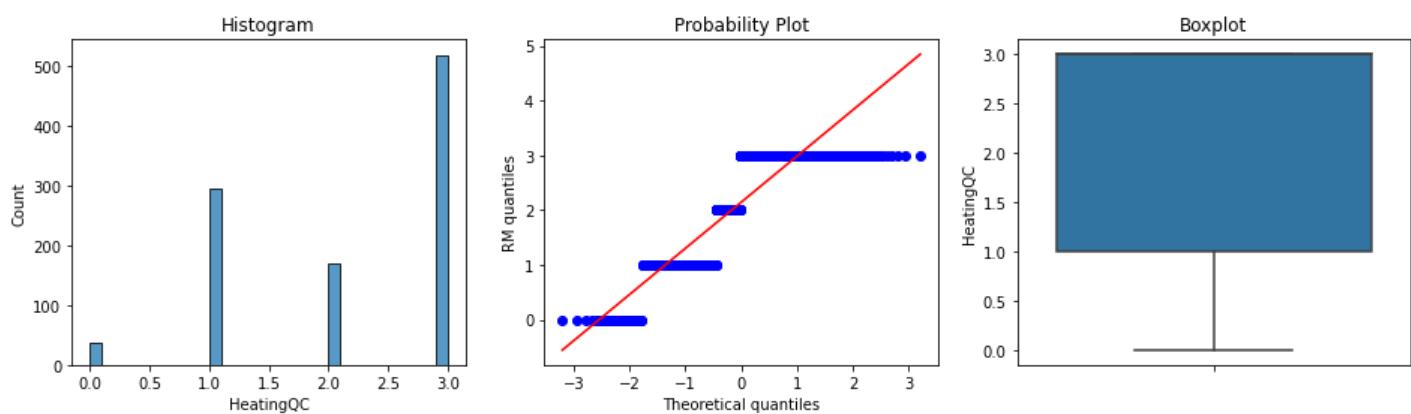
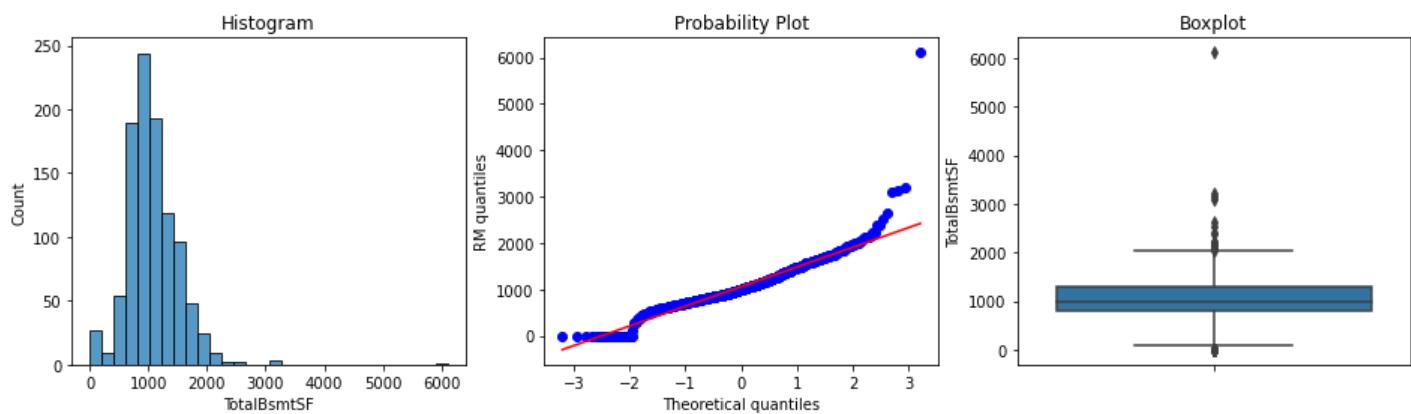
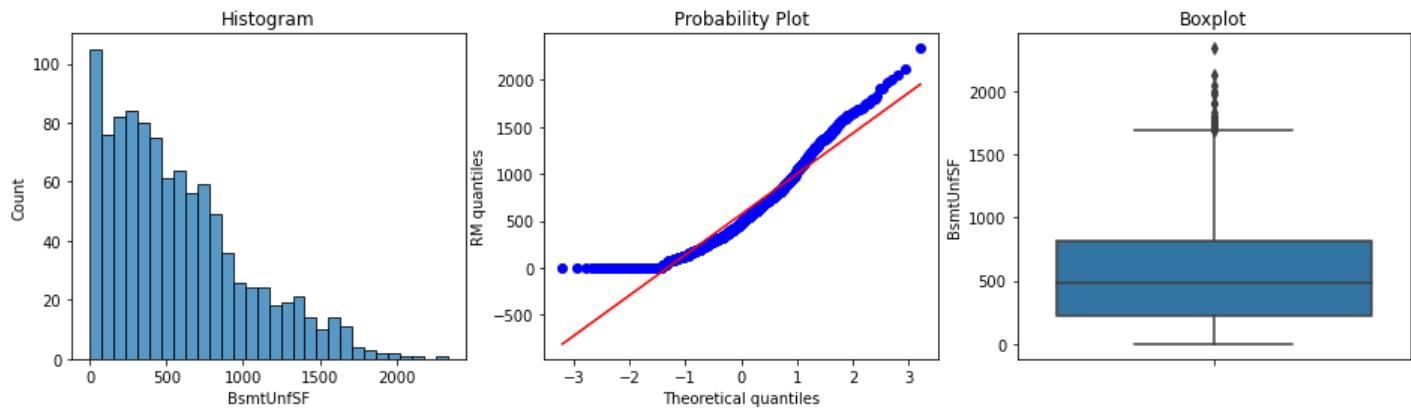
```
for i in X_train.columns:
    diagnostic_plots(X_train,i)
```

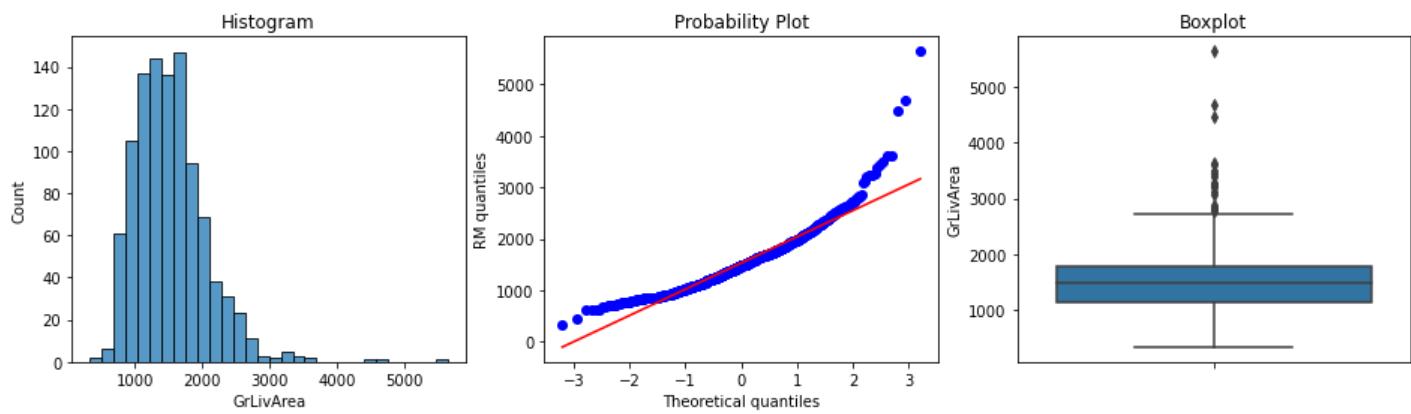
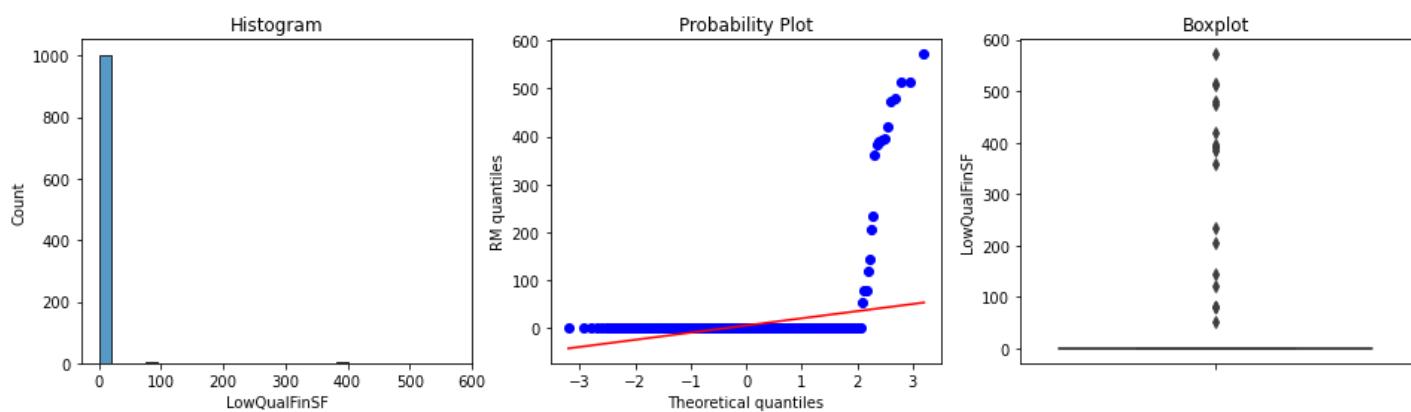
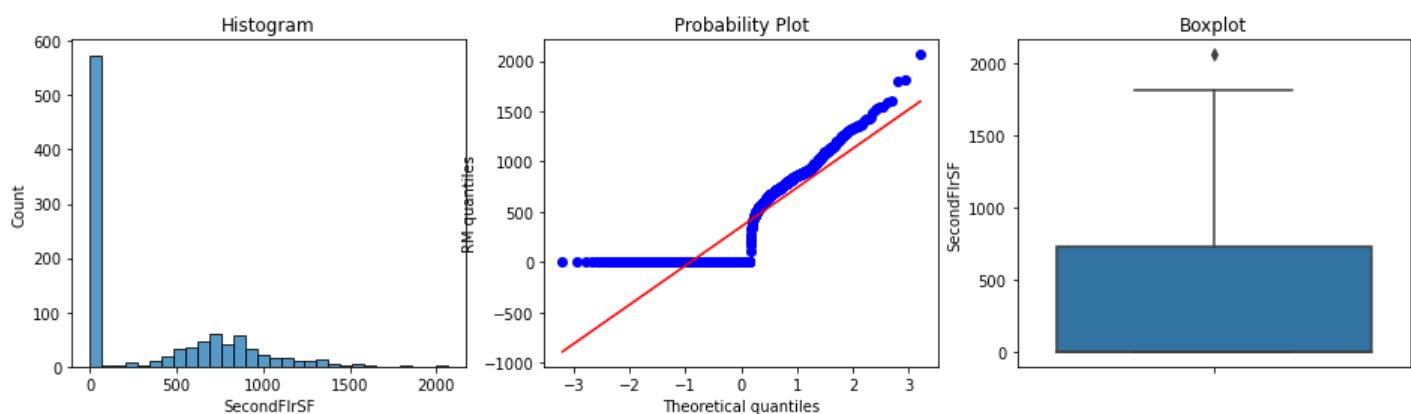
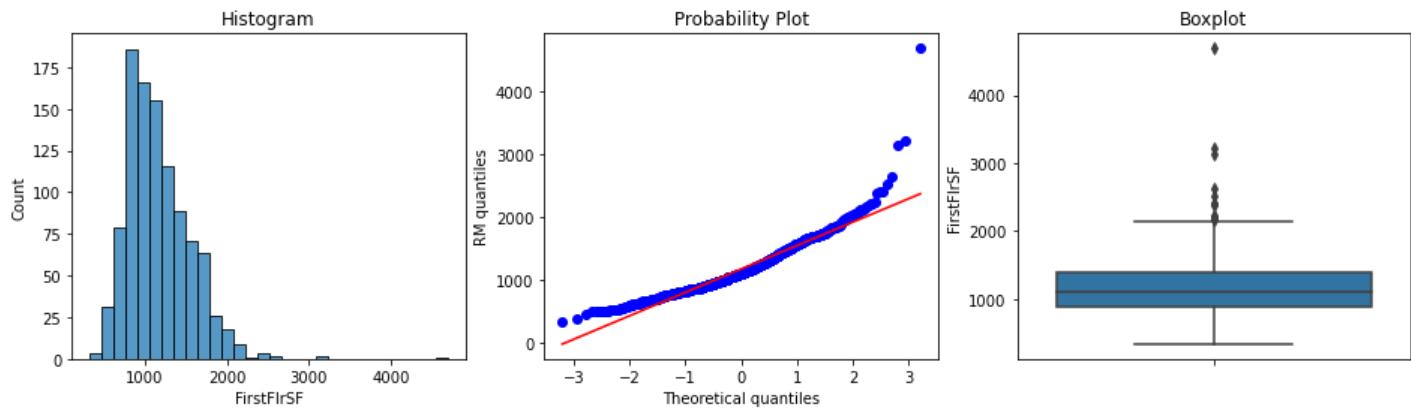


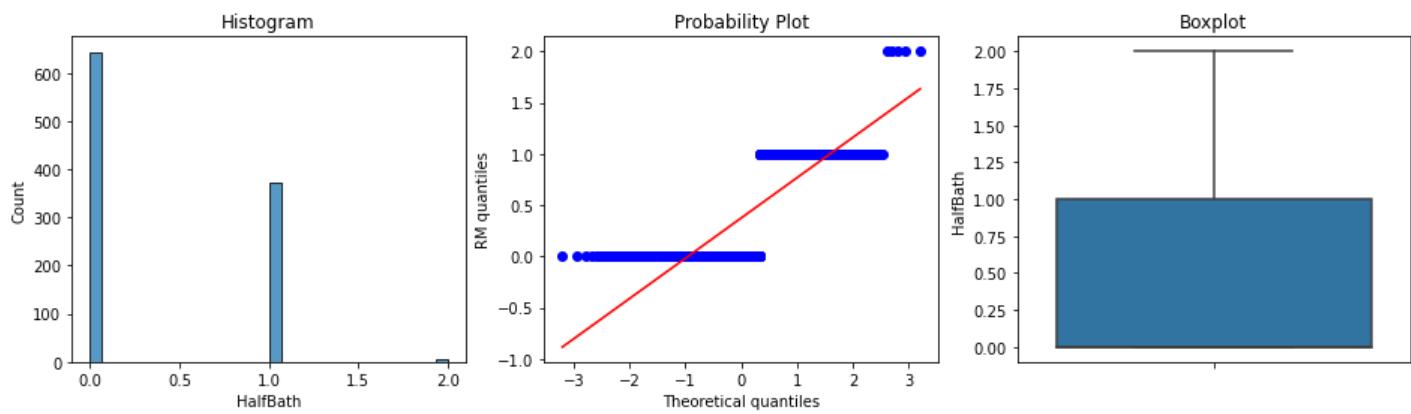
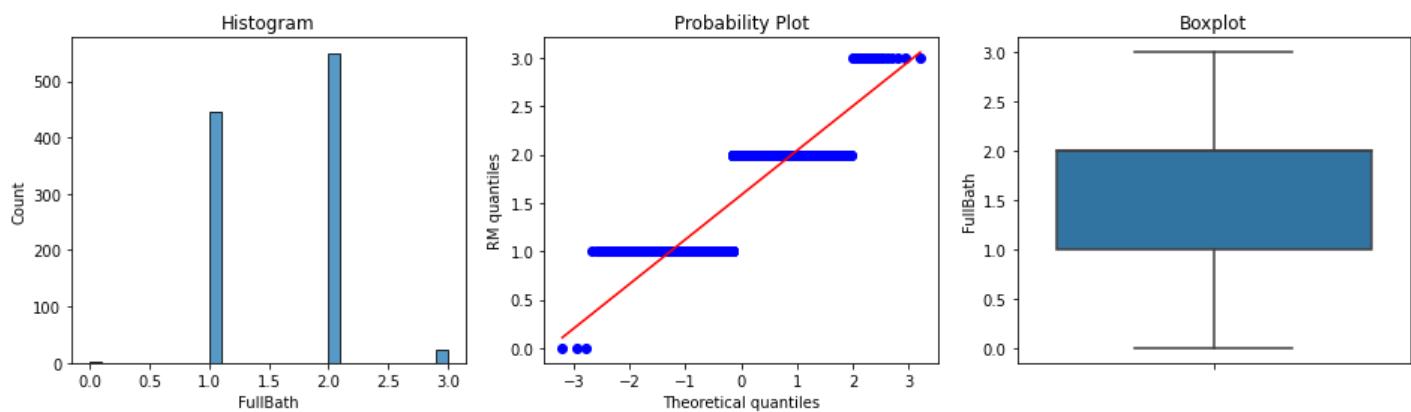
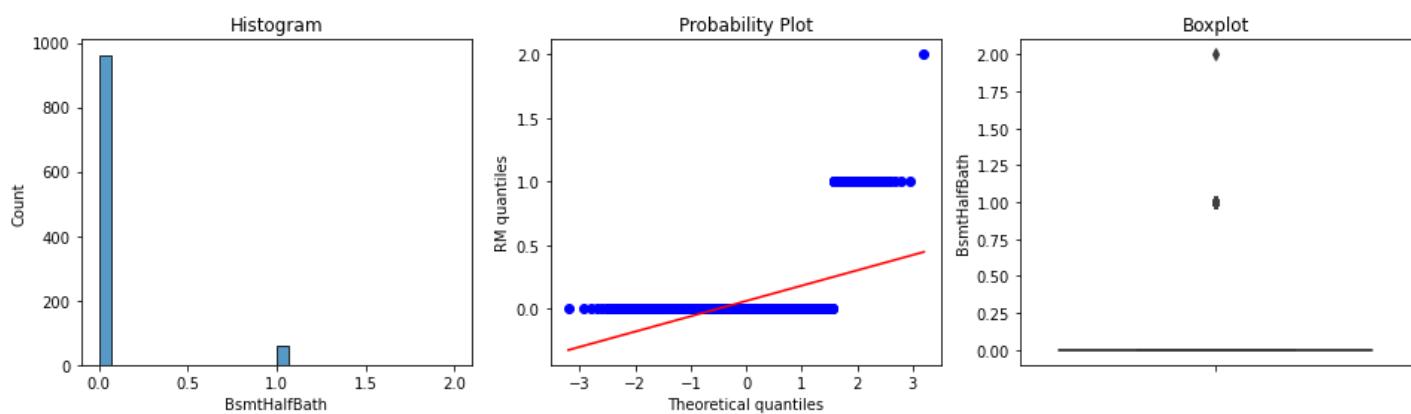
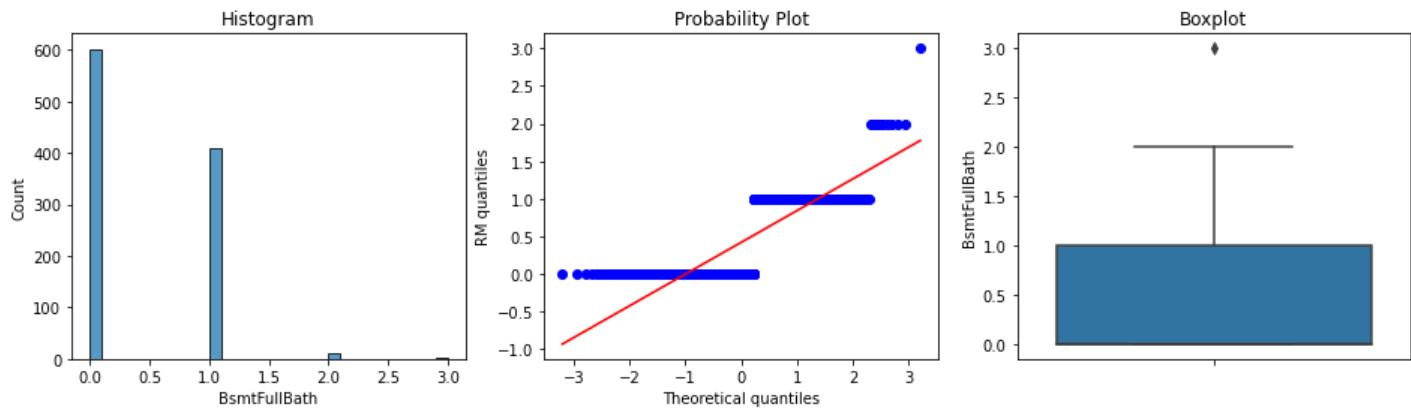


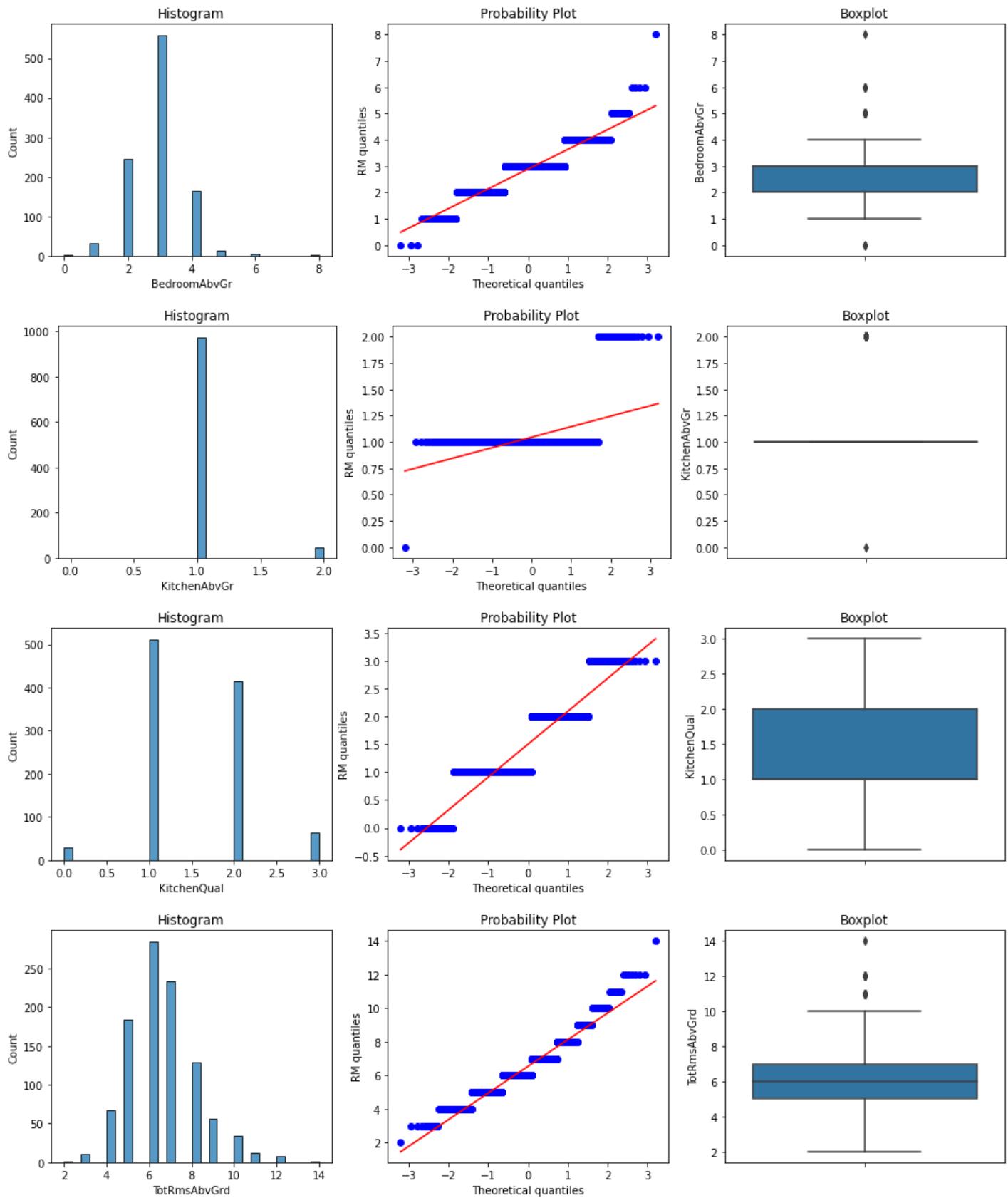


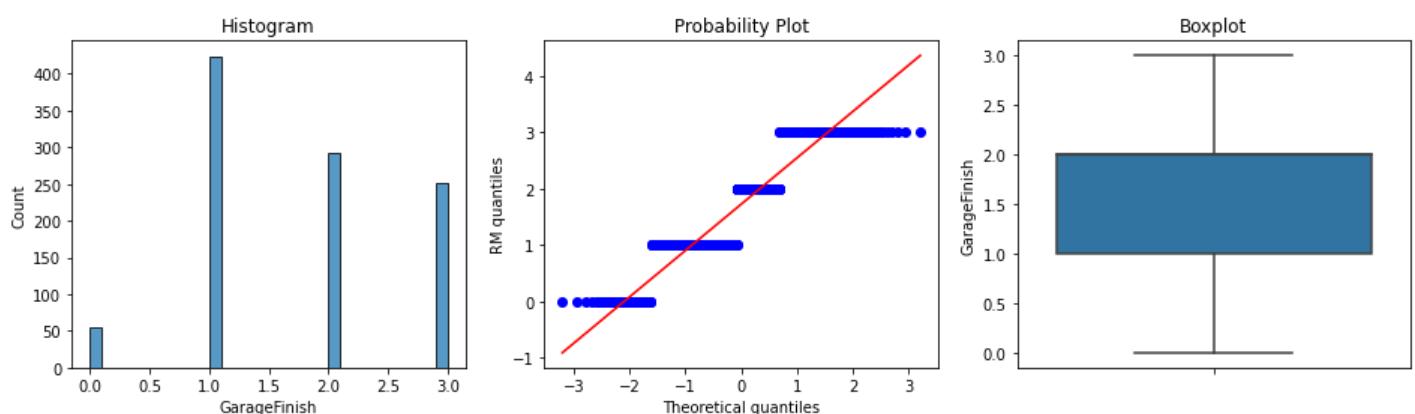
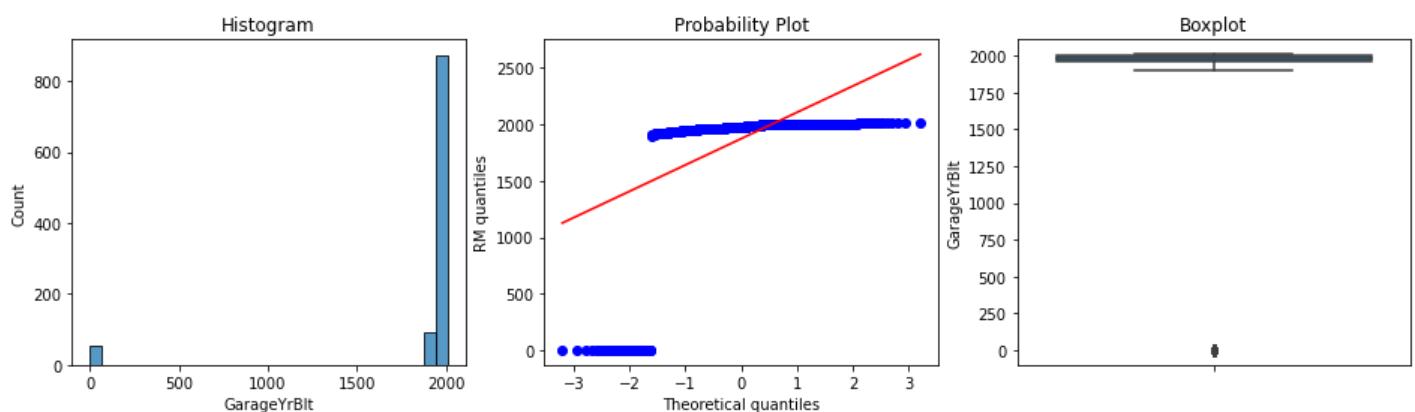
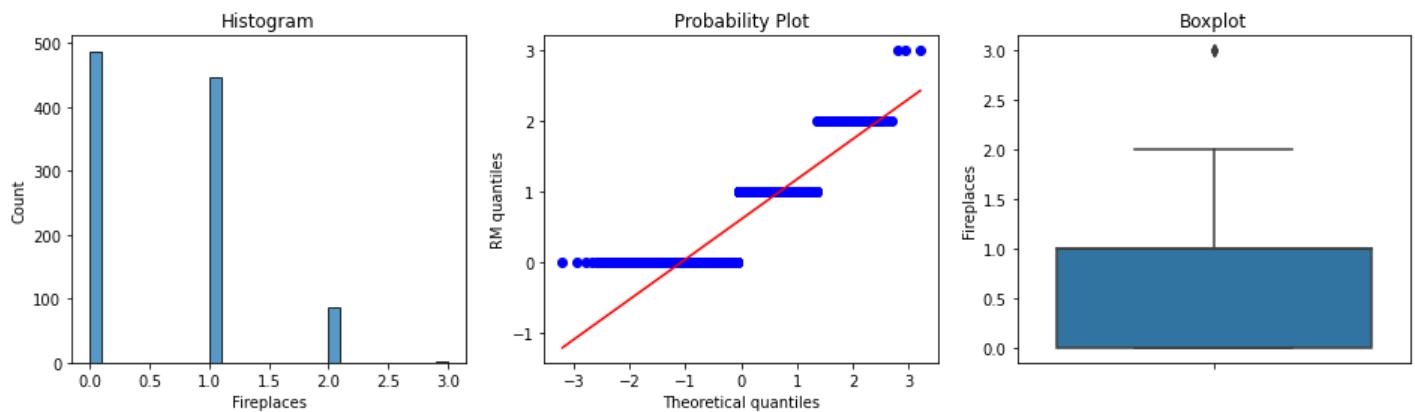
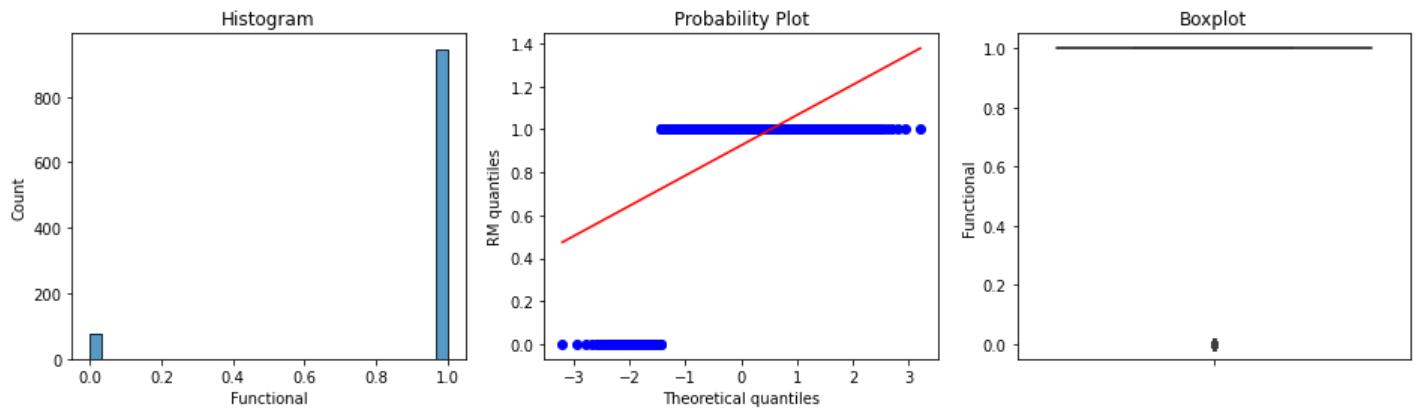


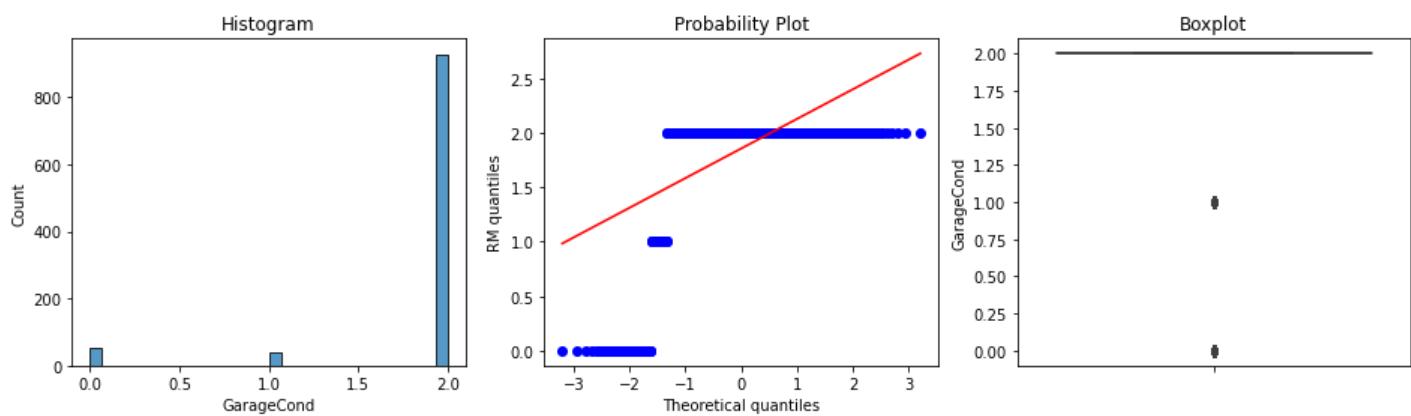
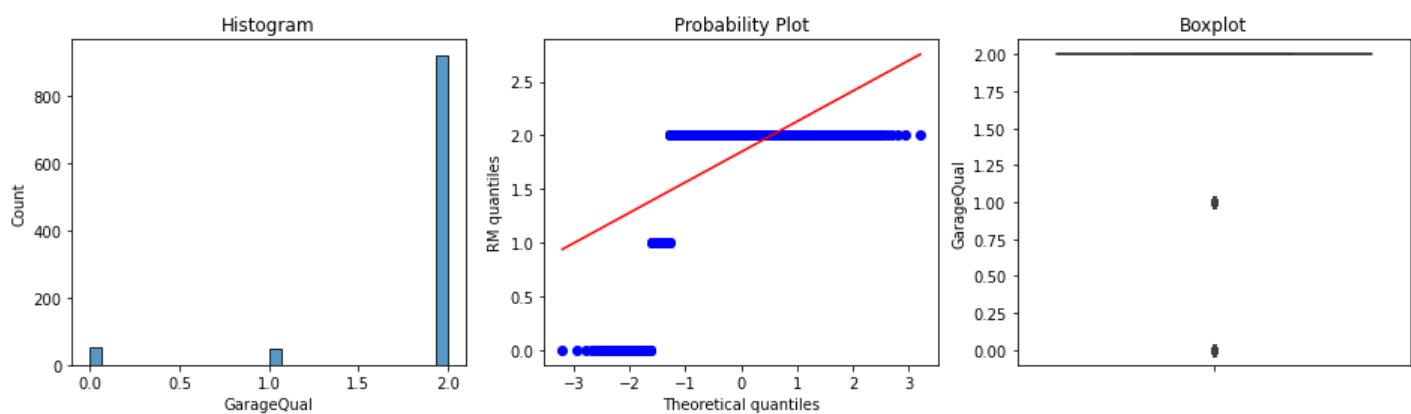
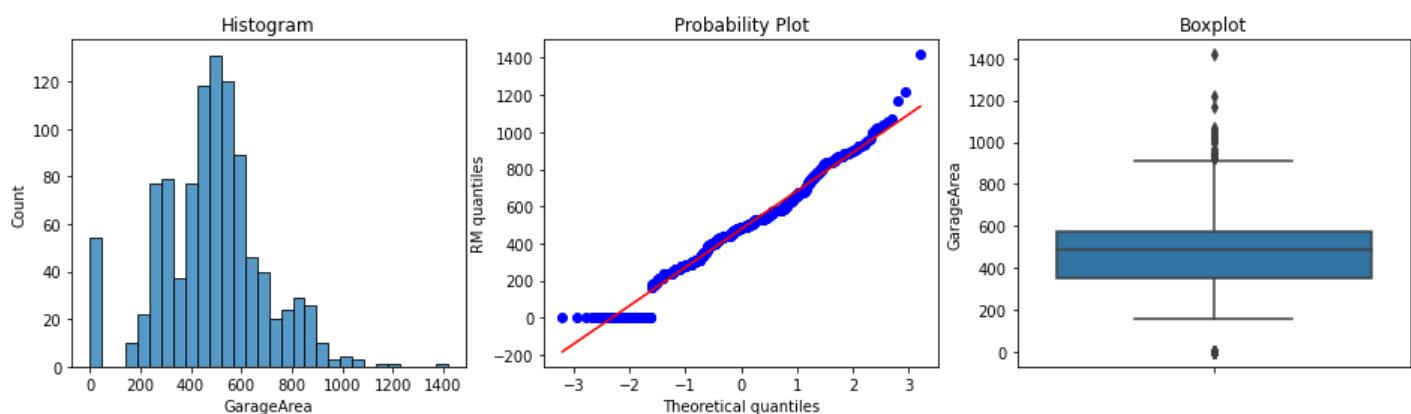
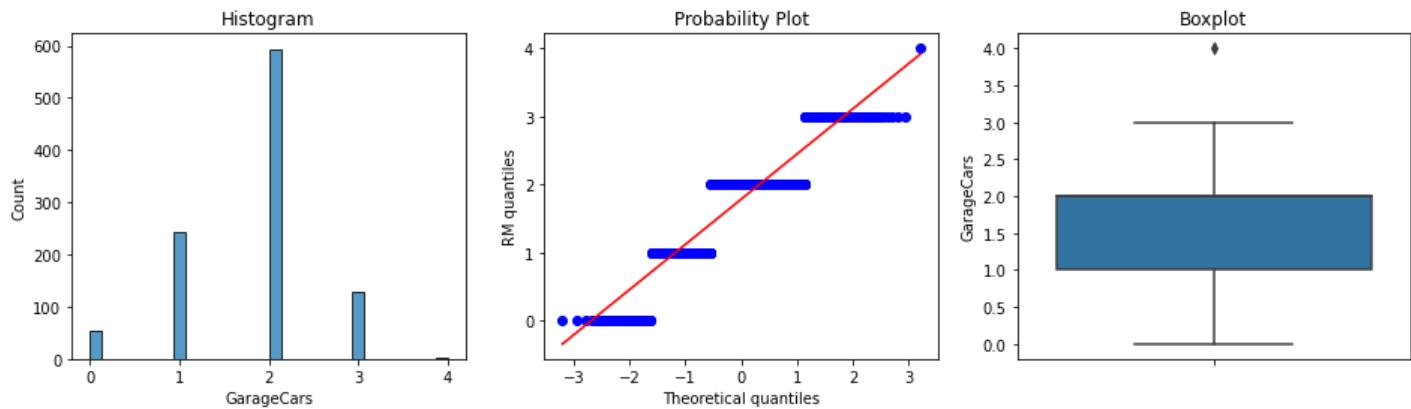


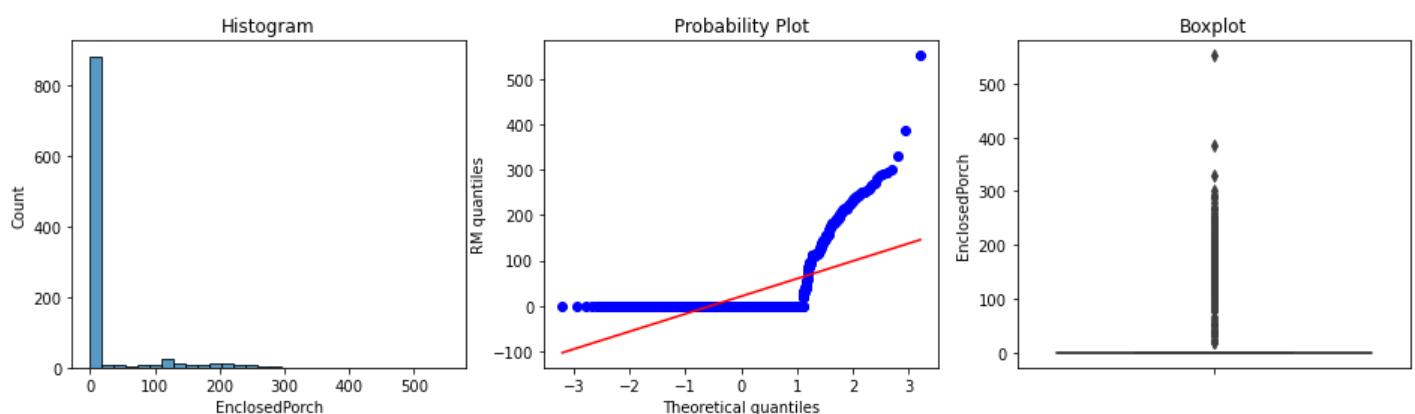
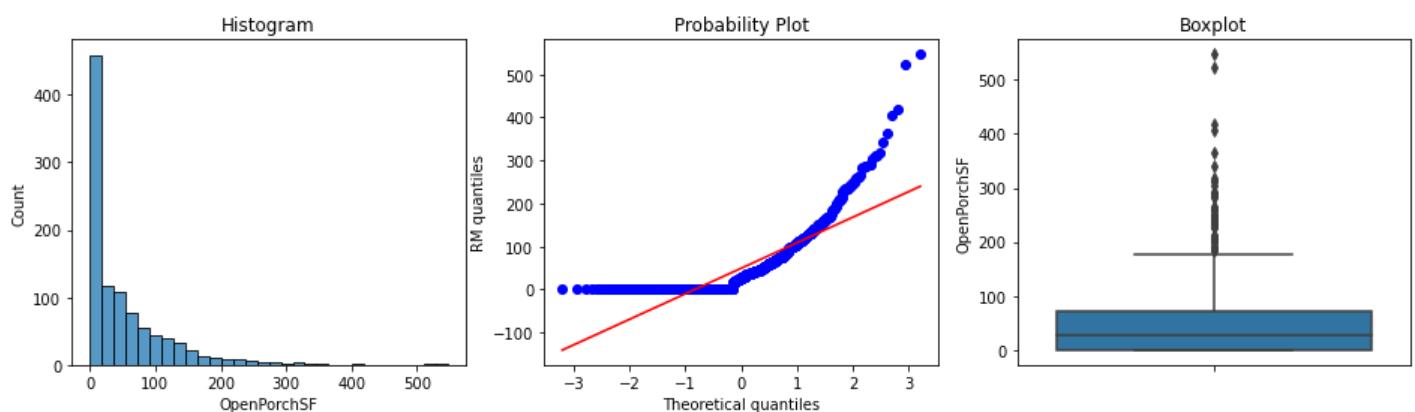
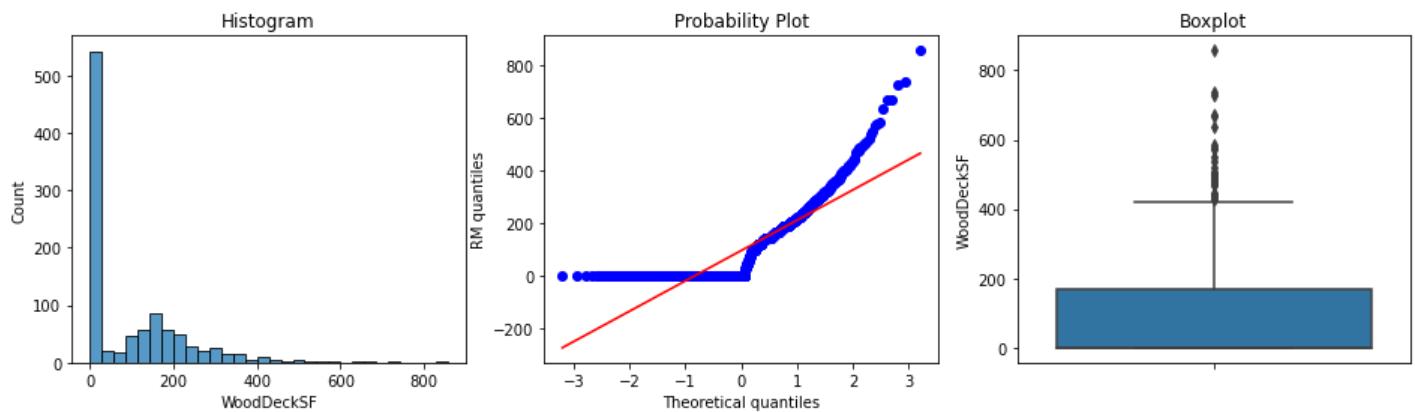
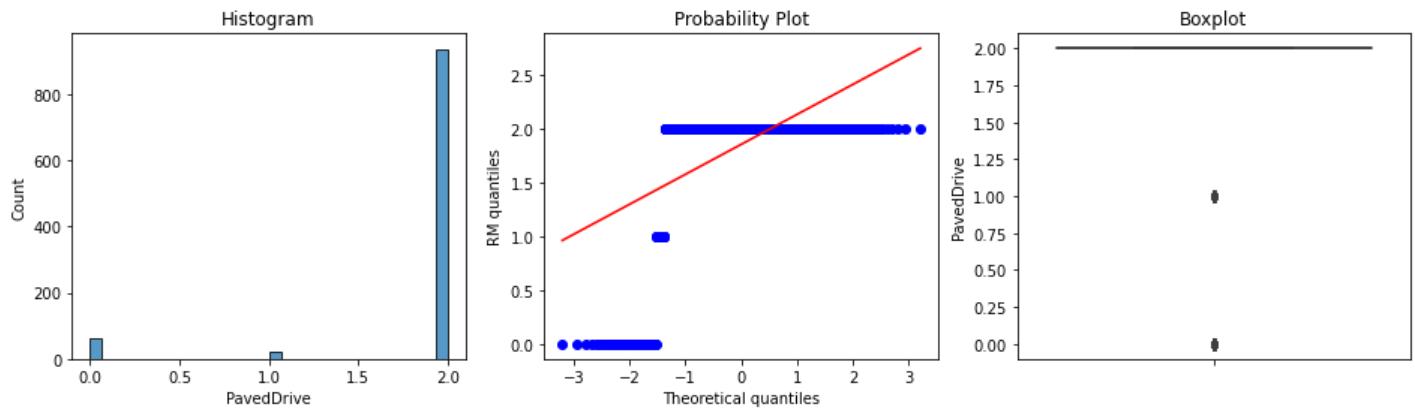


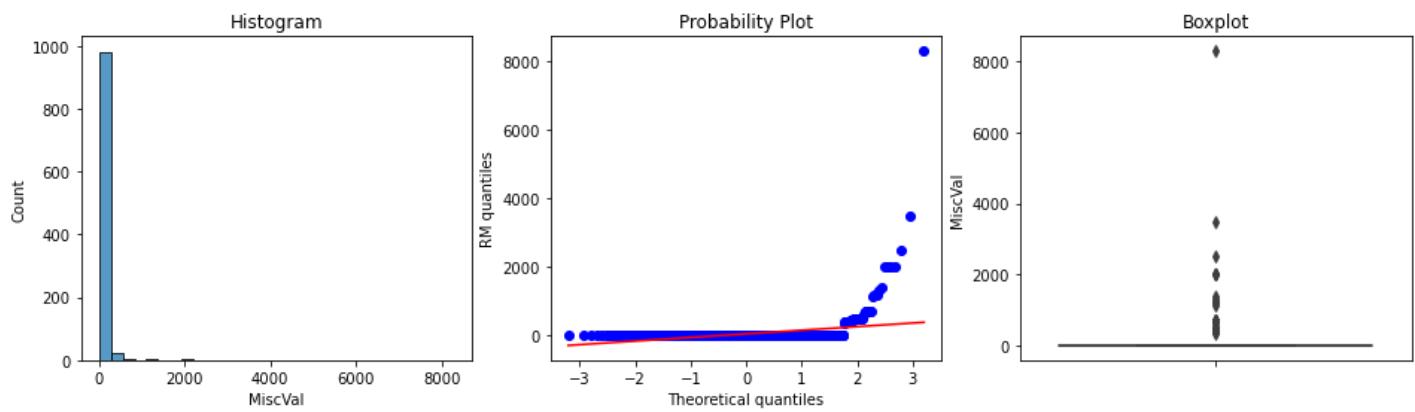
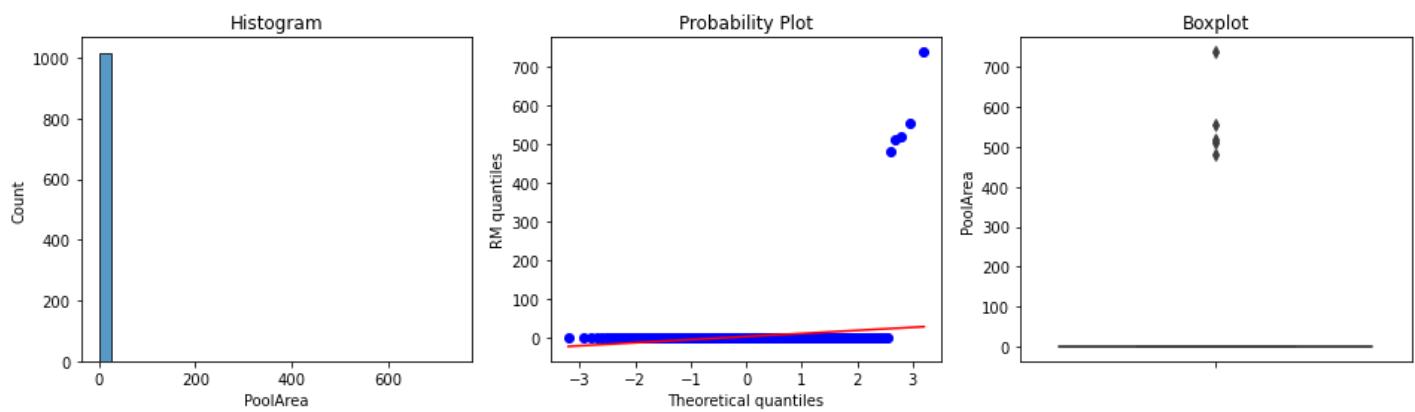
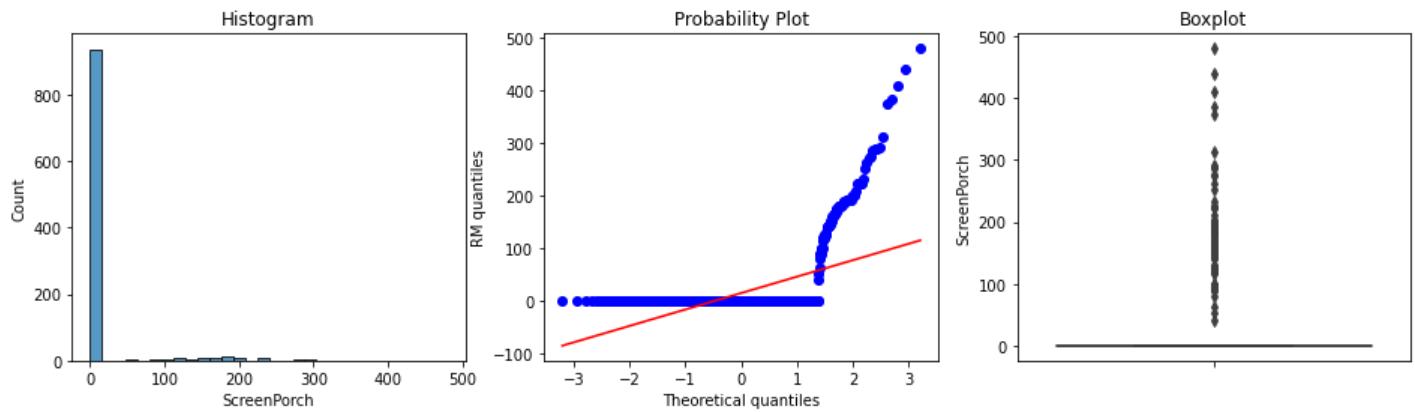
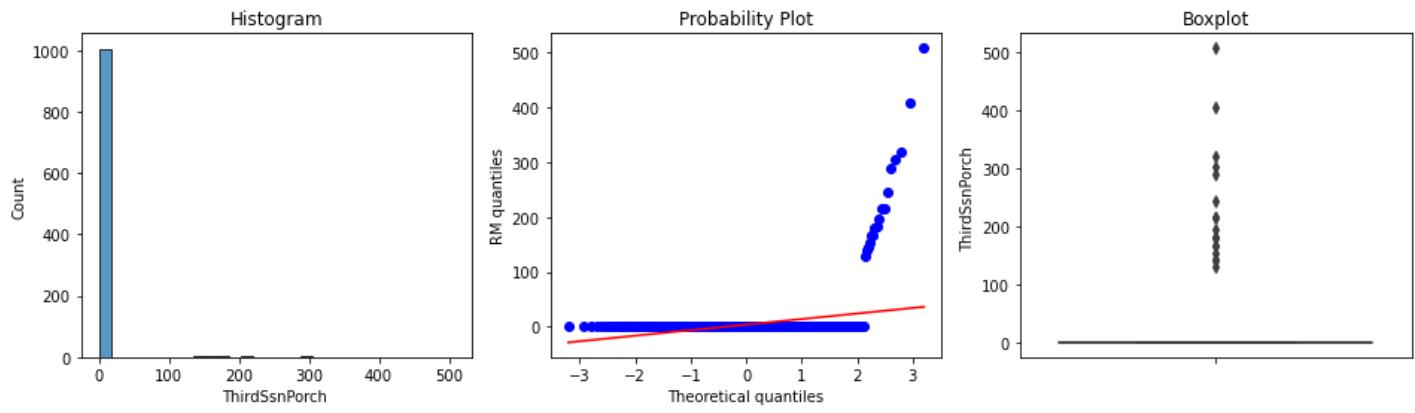


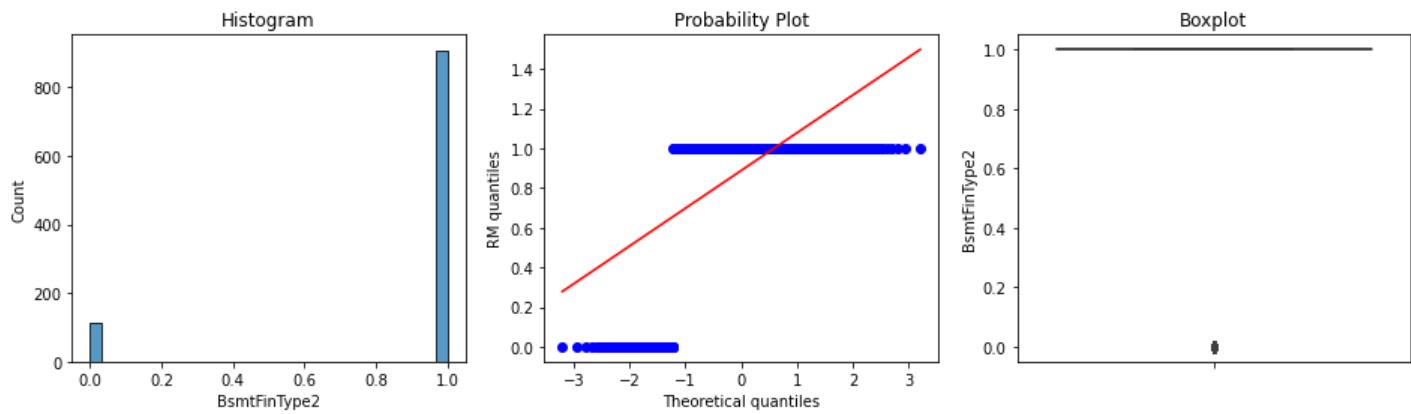
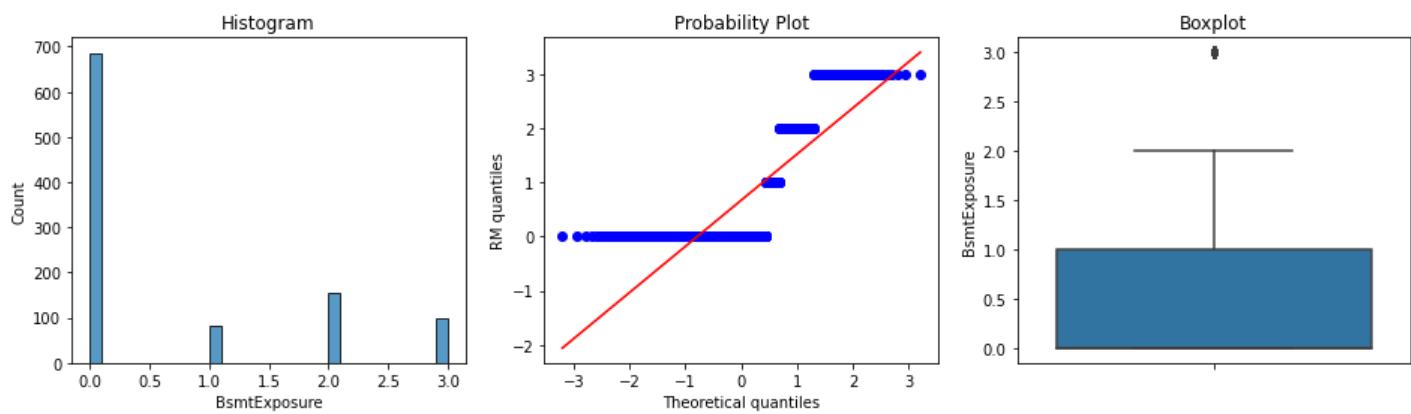
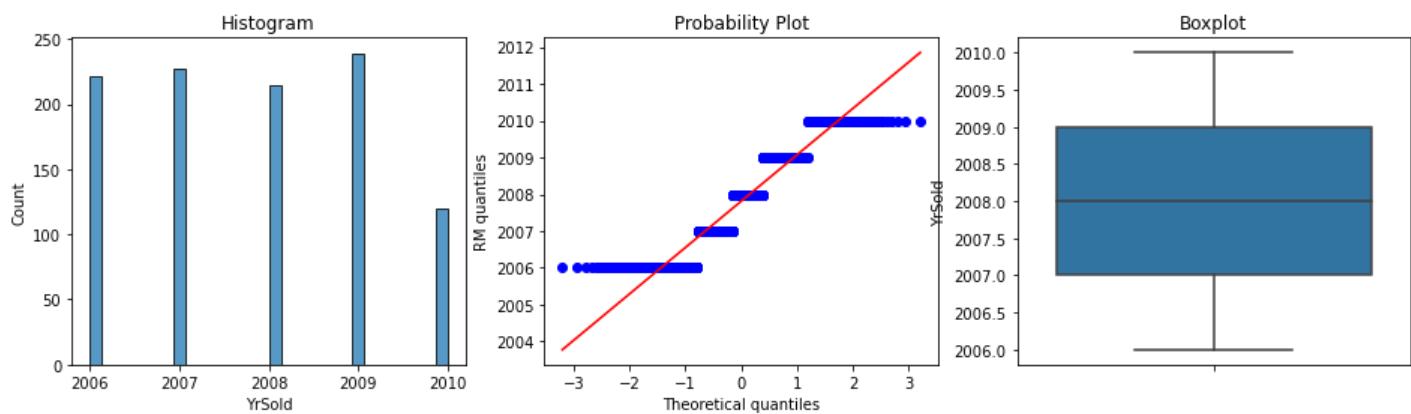
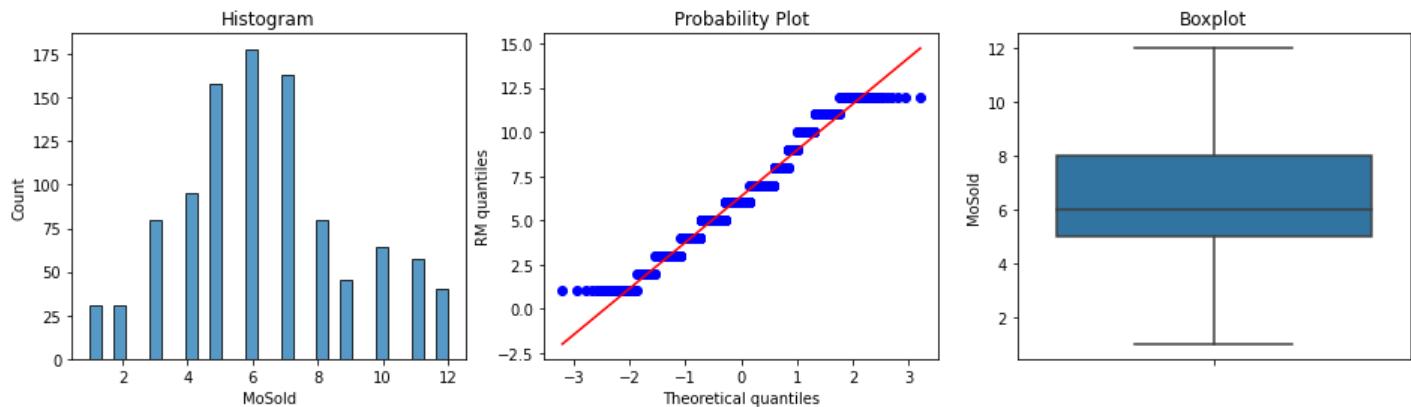


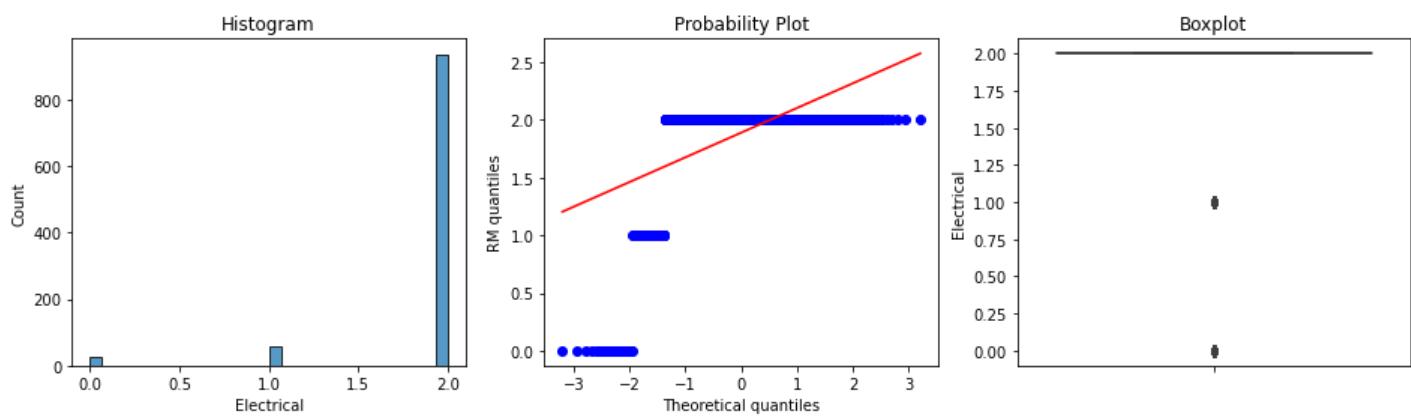
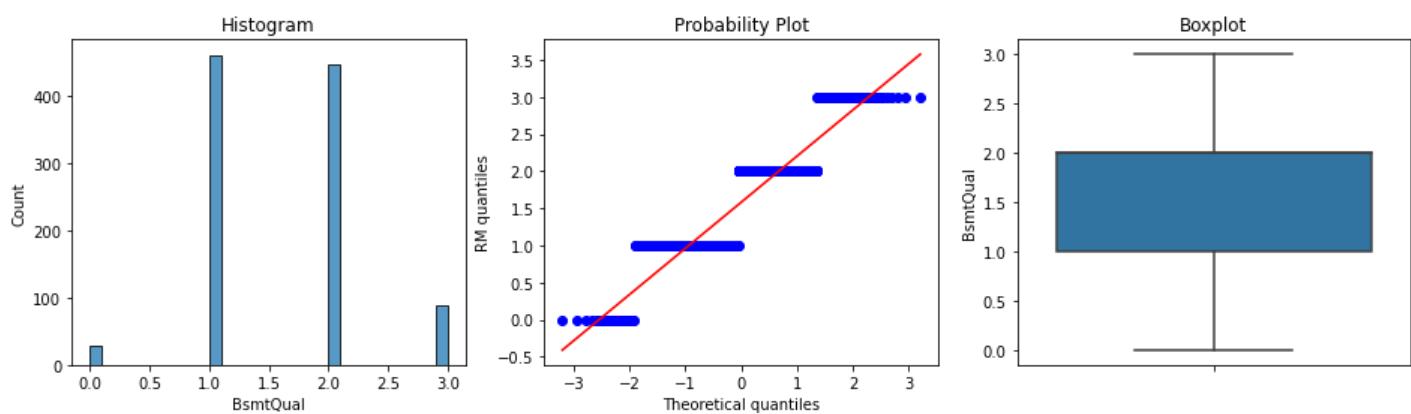
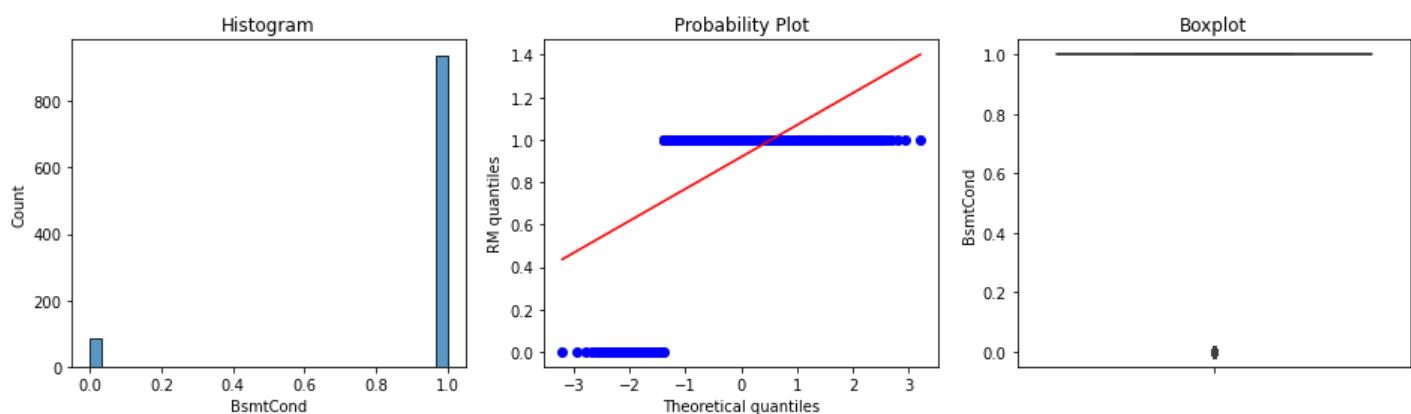
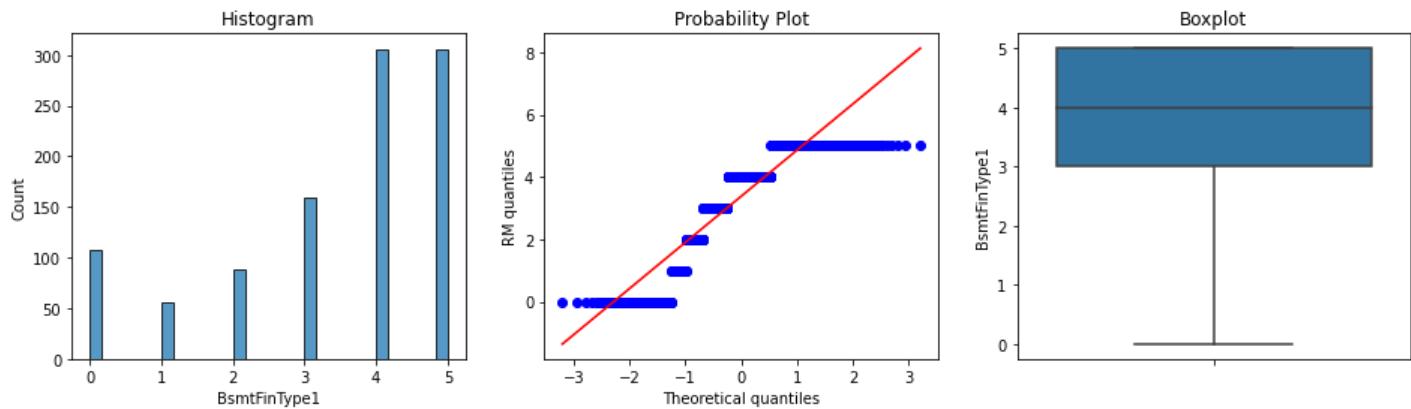


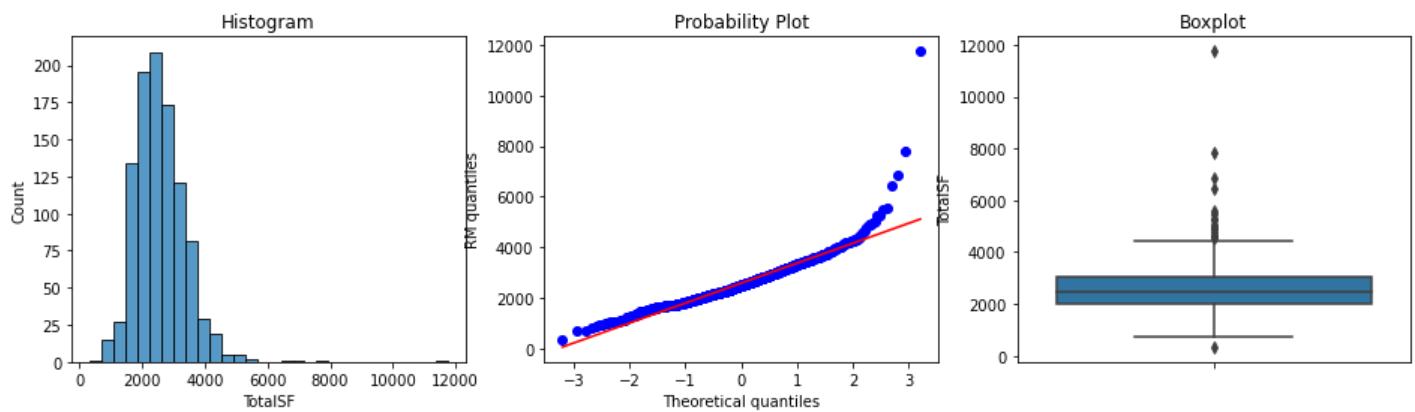
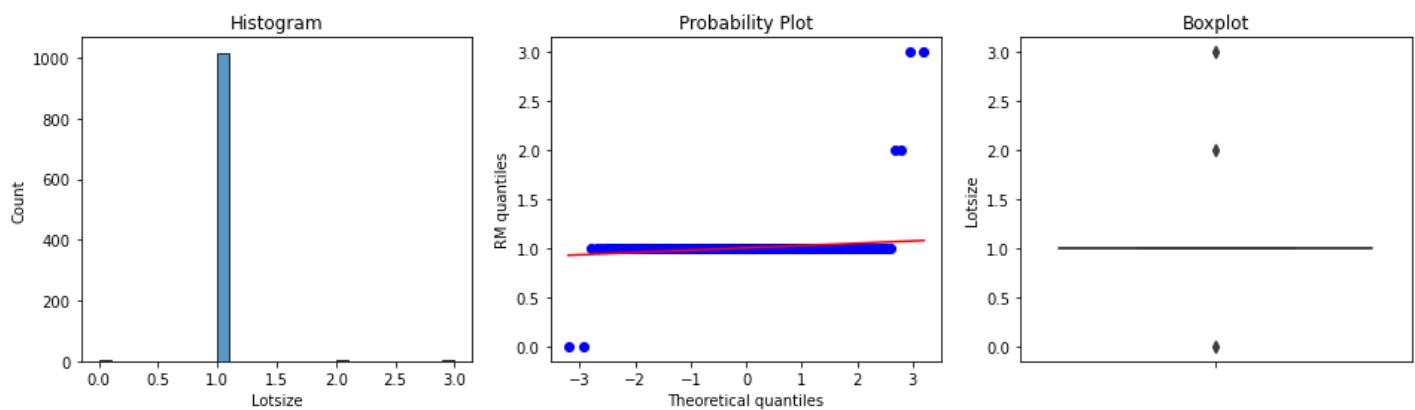
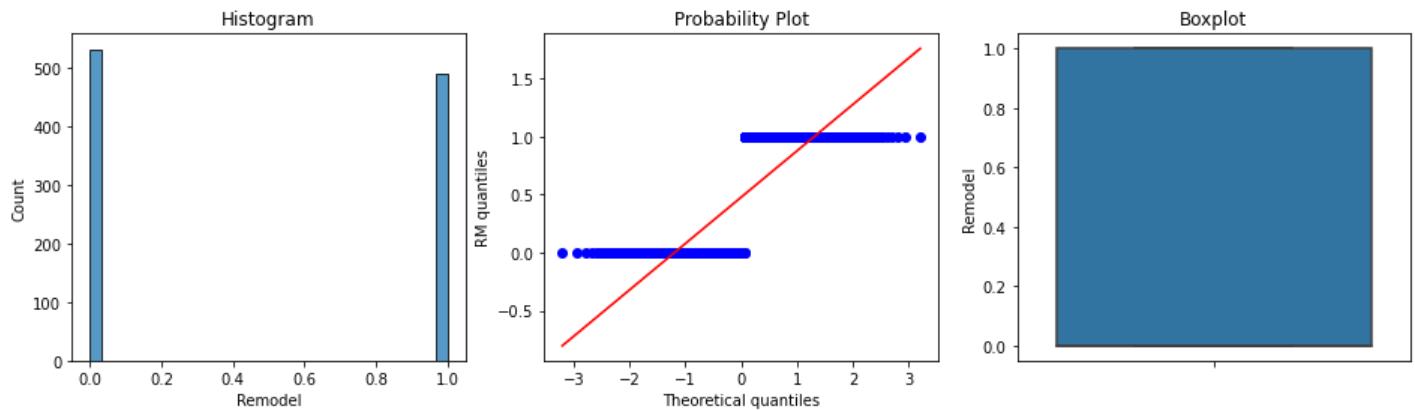
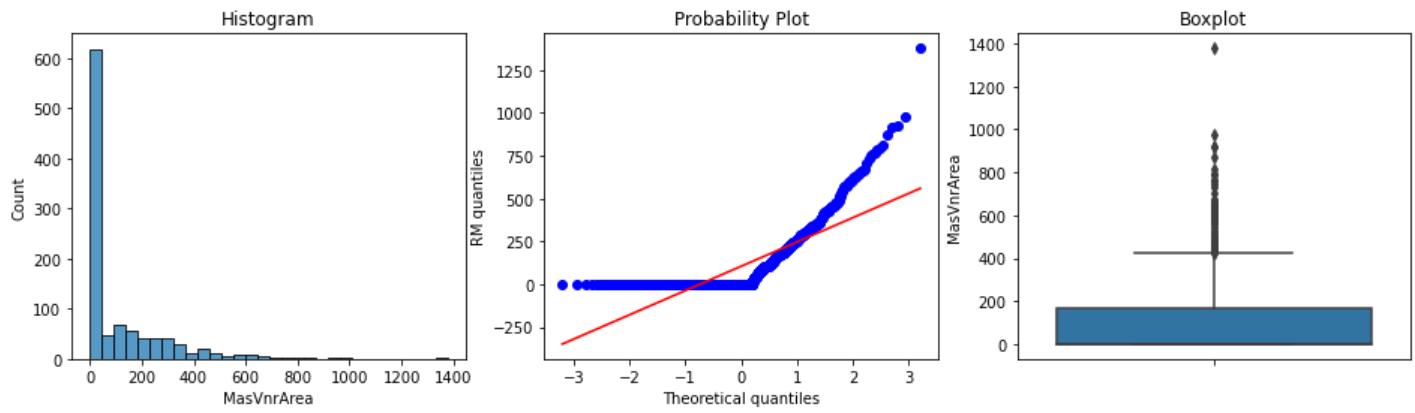


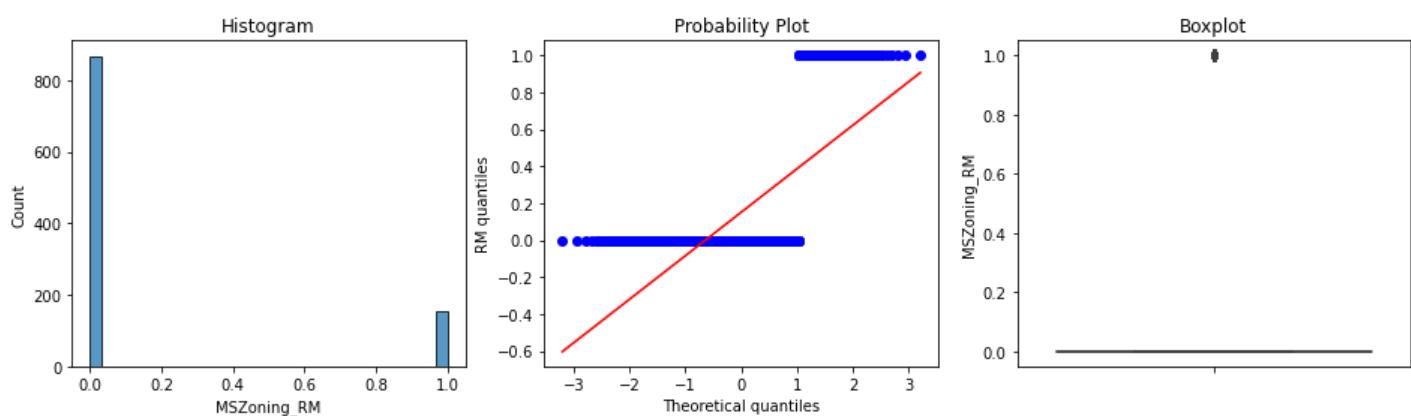
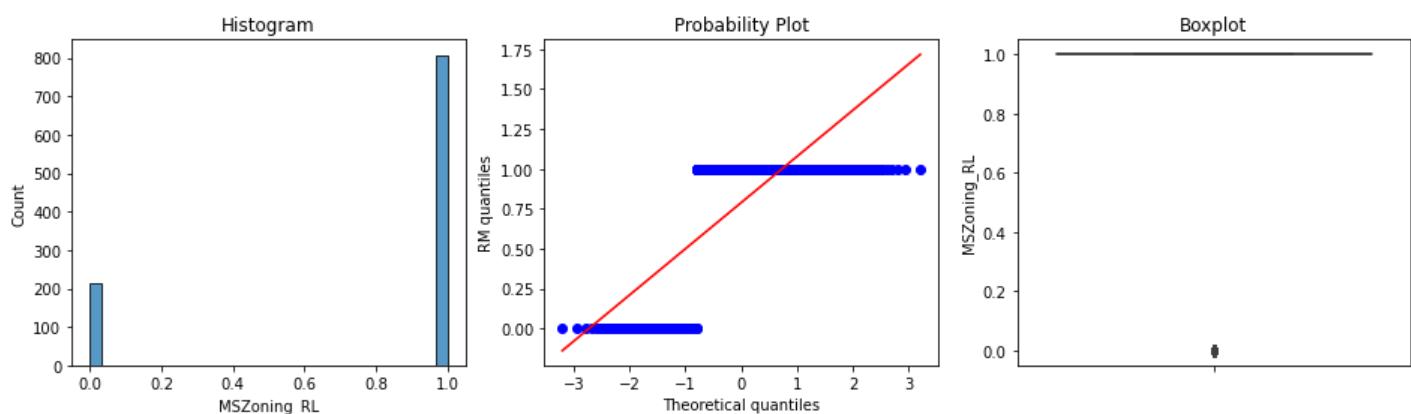
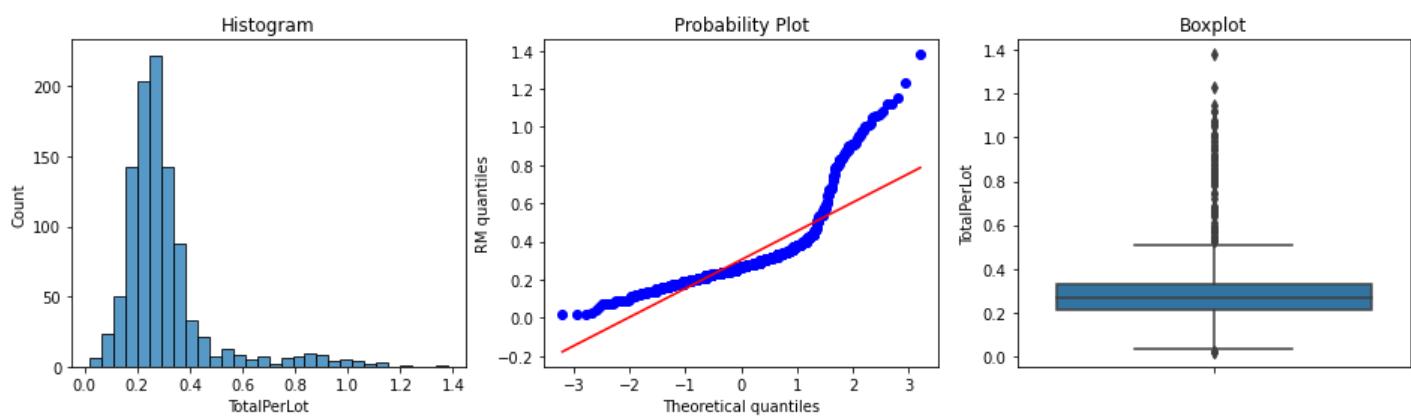
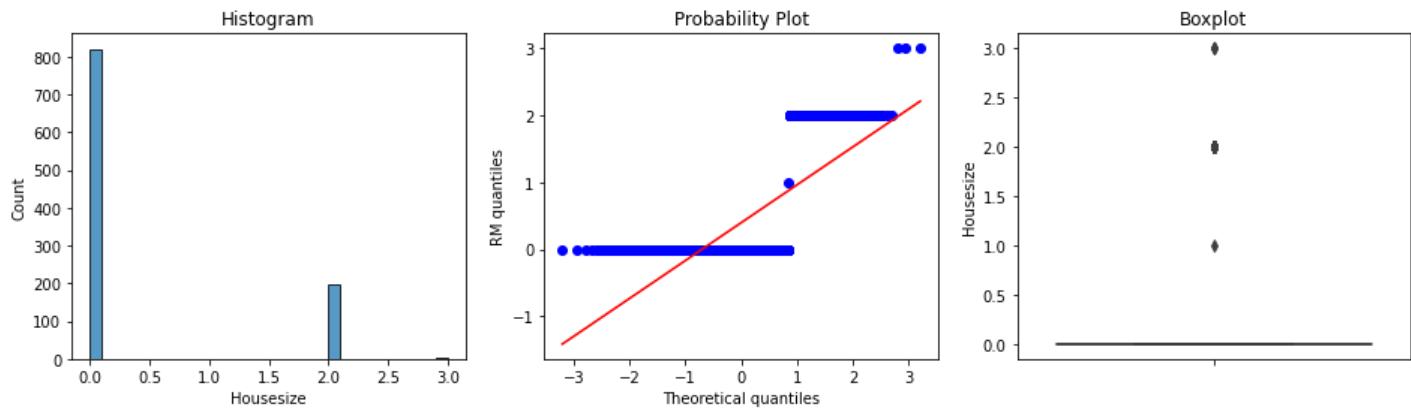


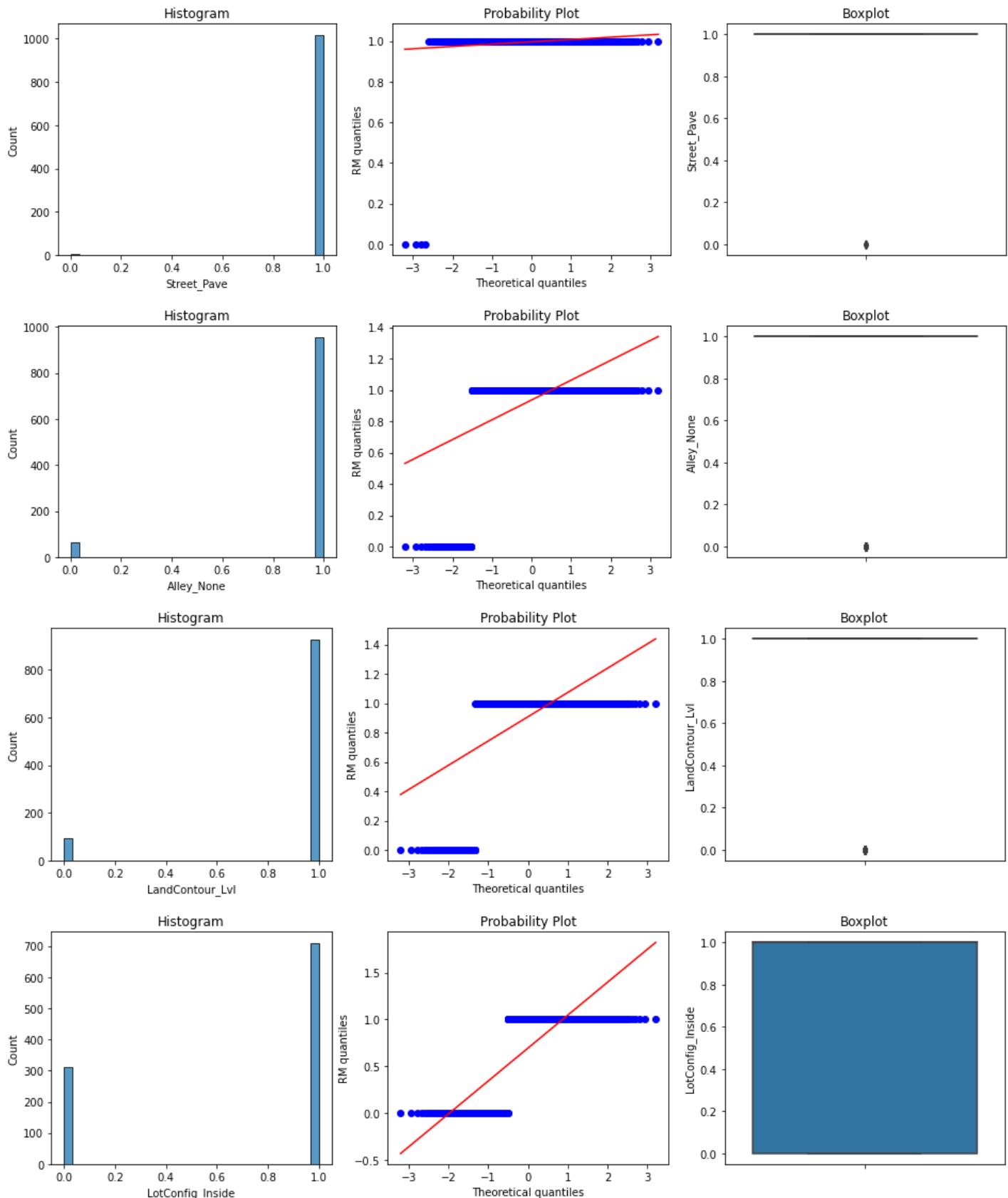


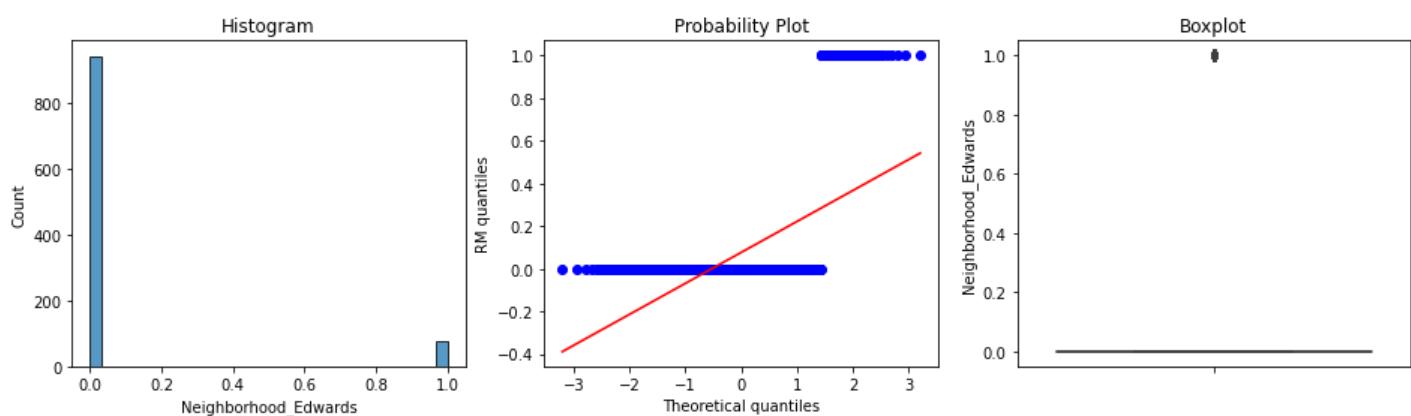
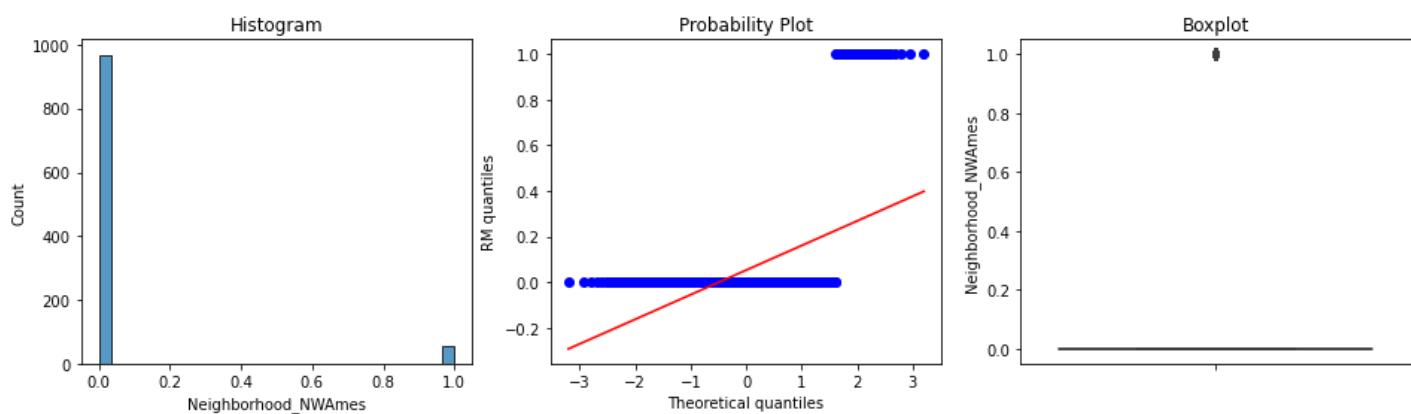
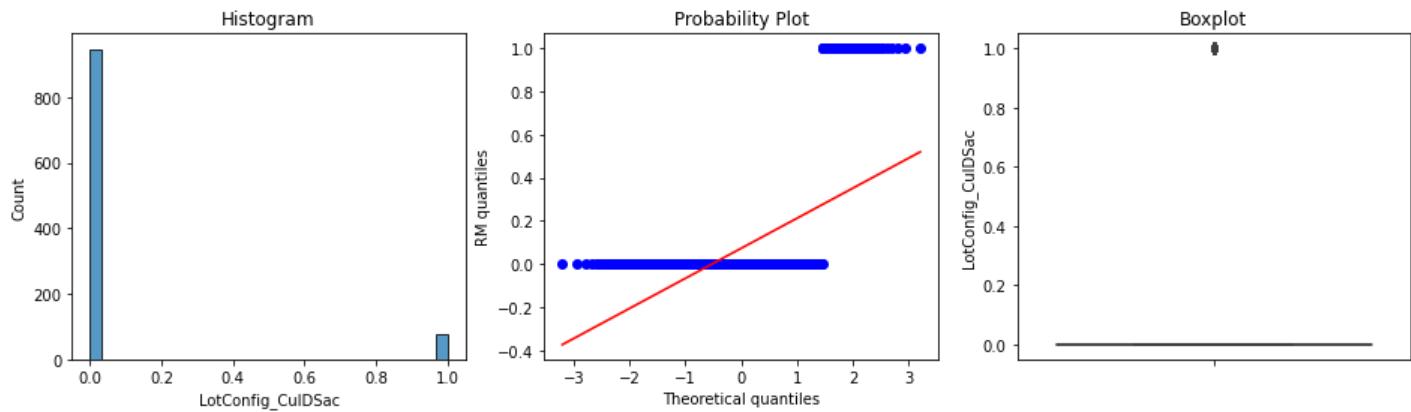
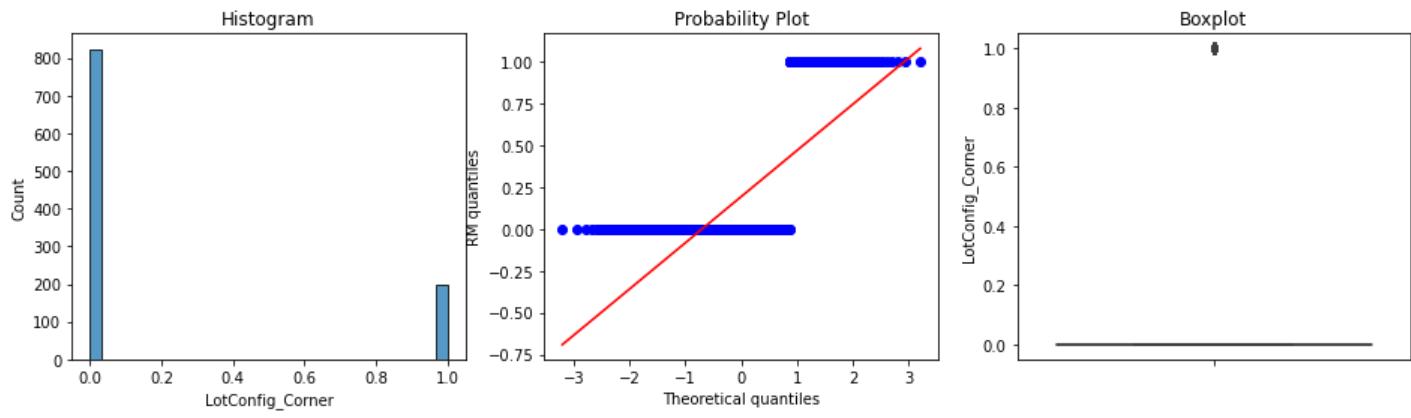


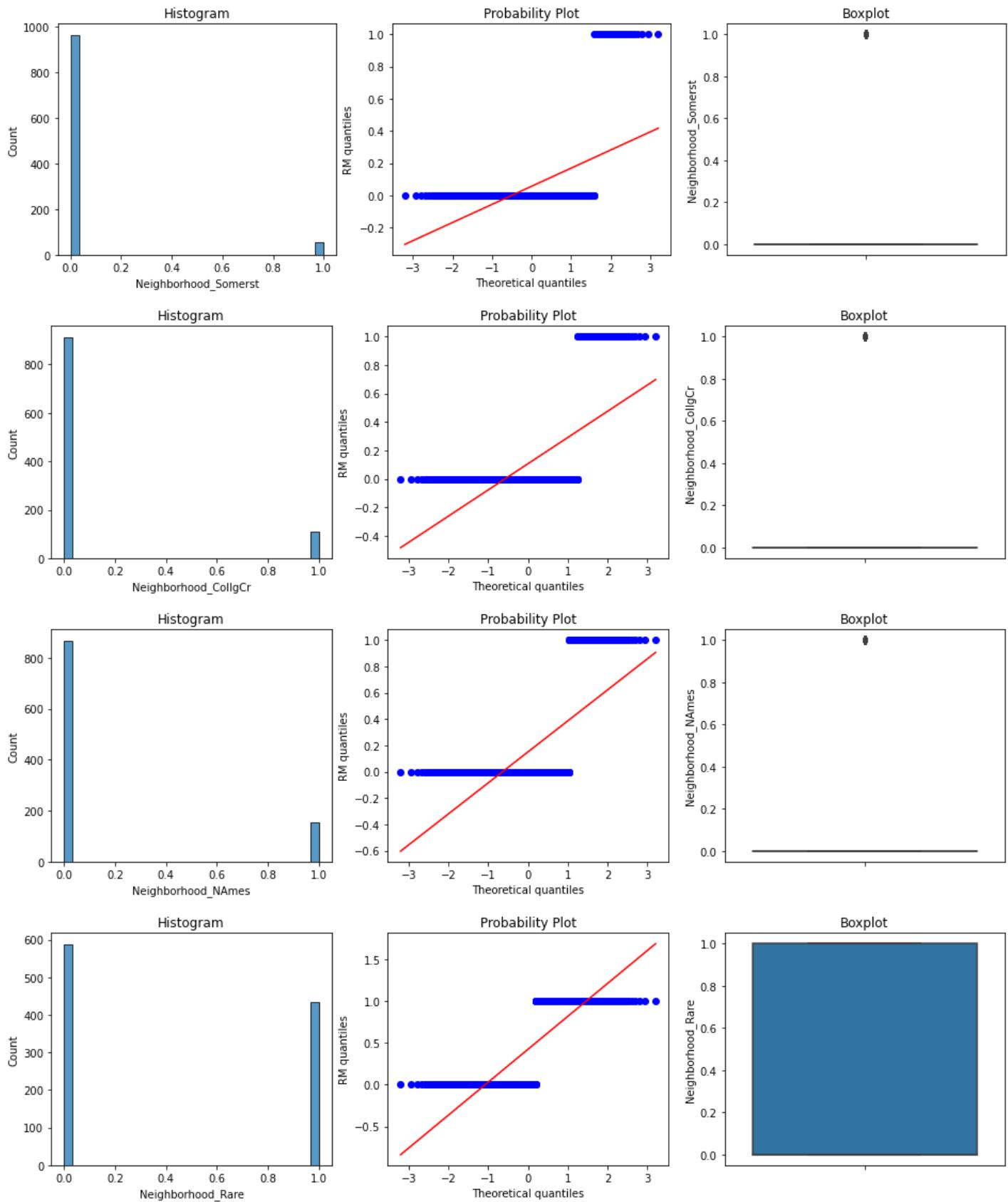


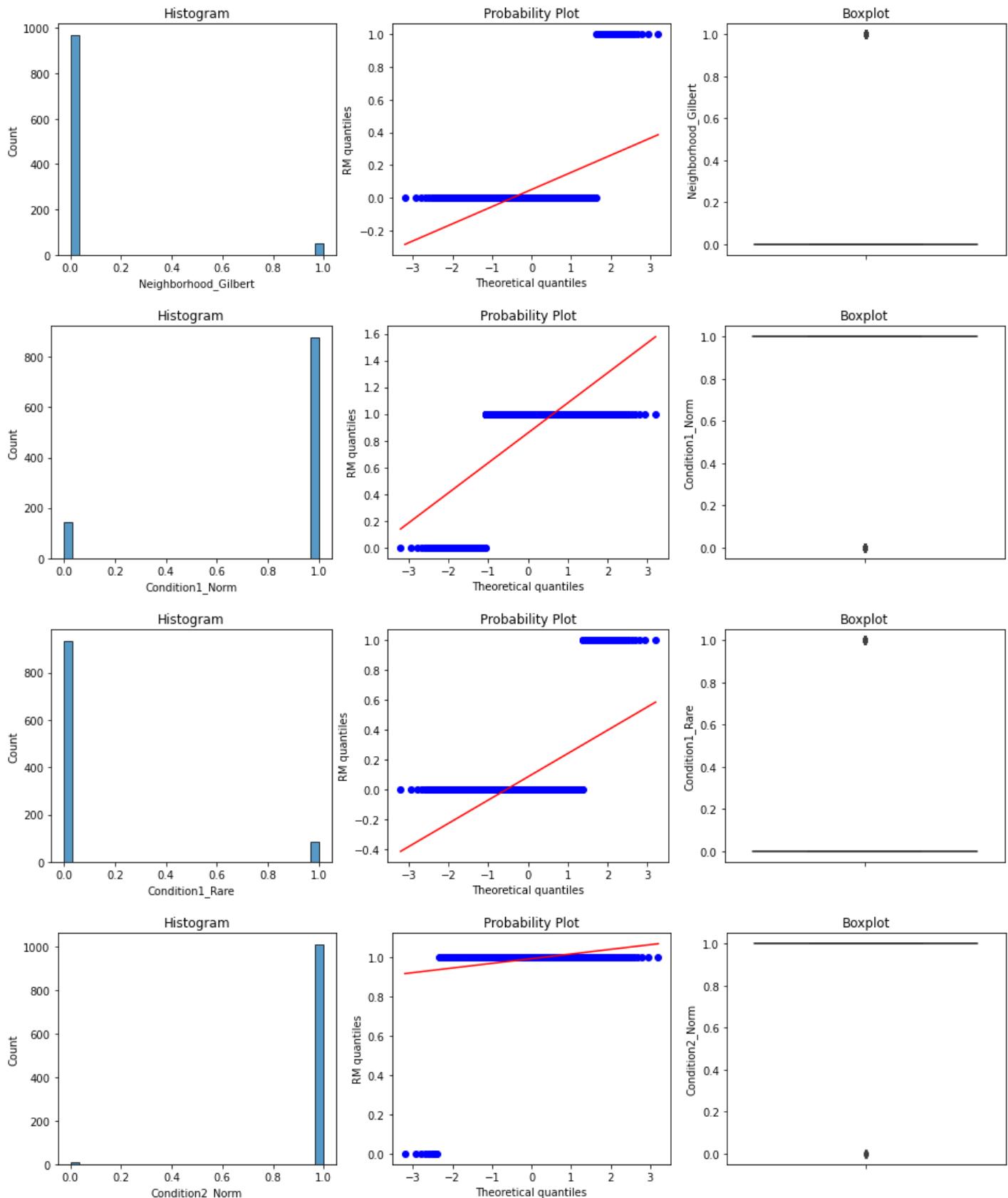


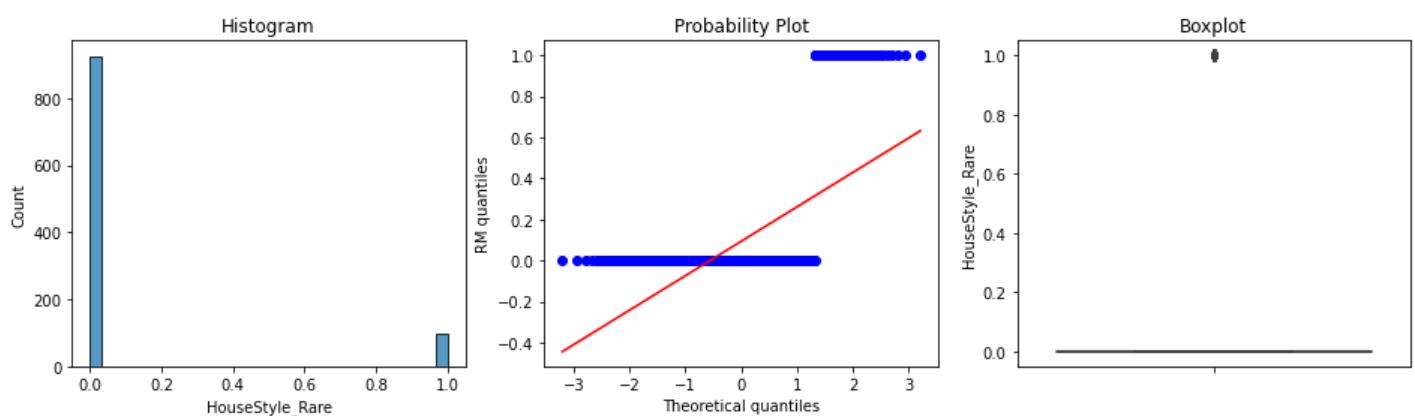
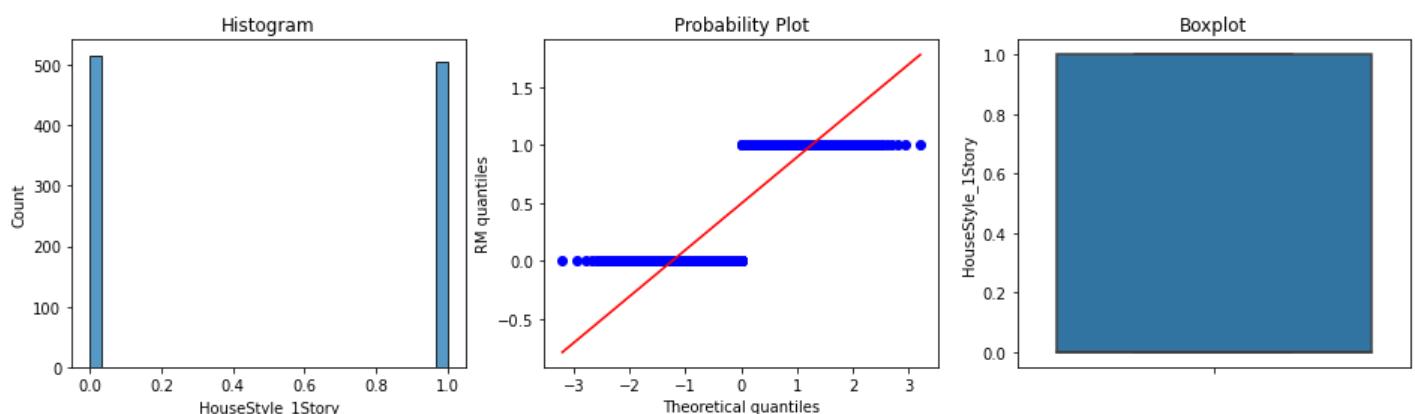
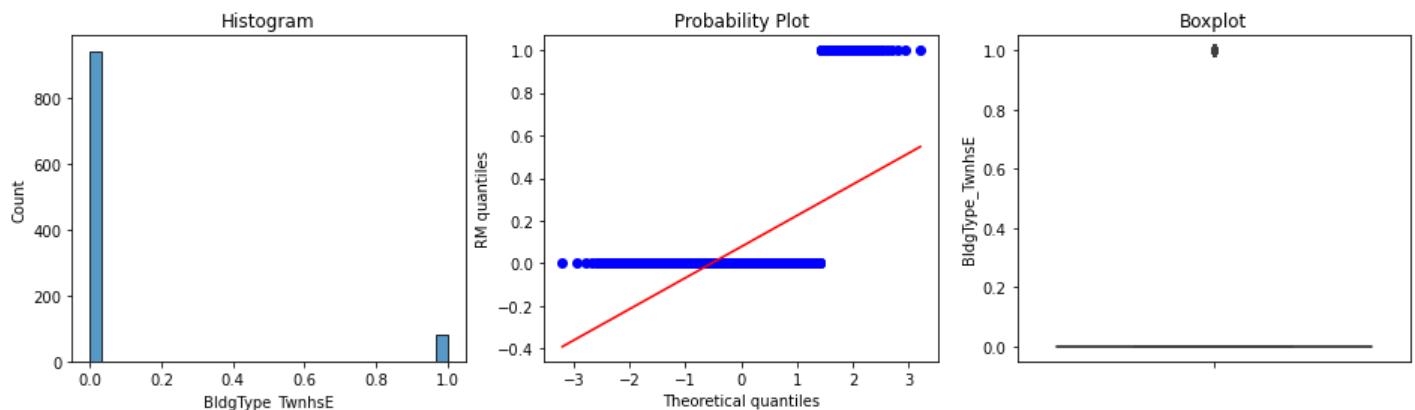
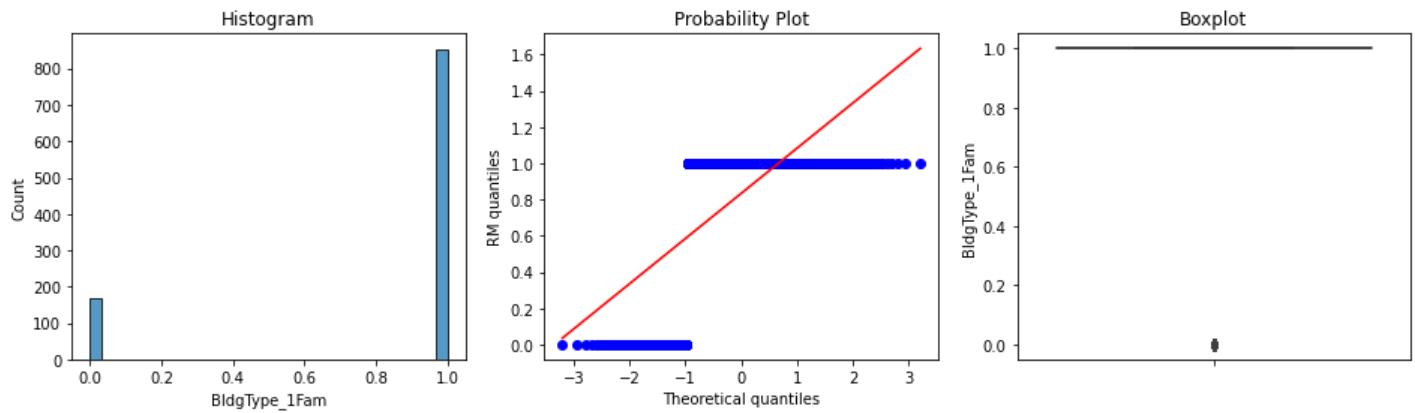


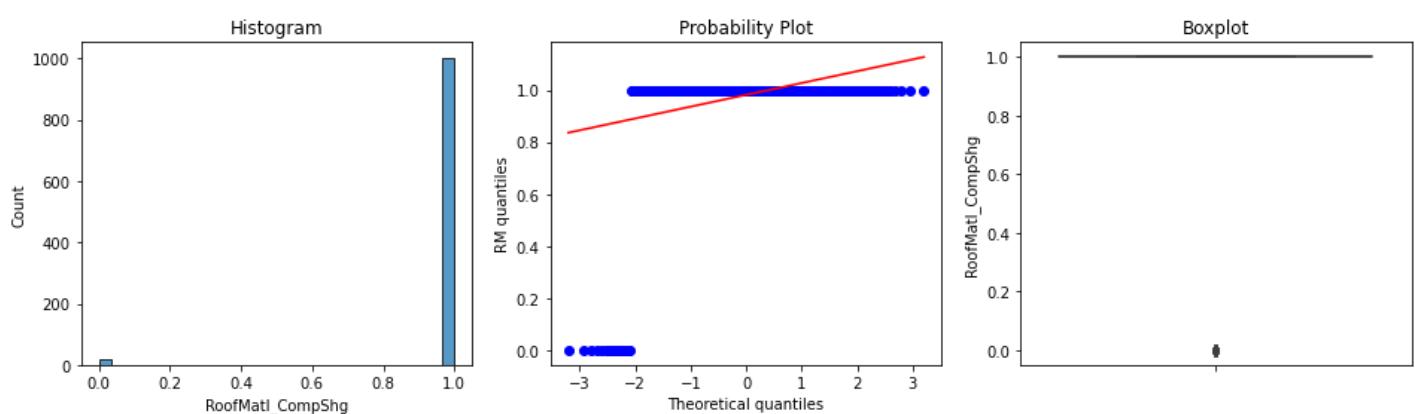
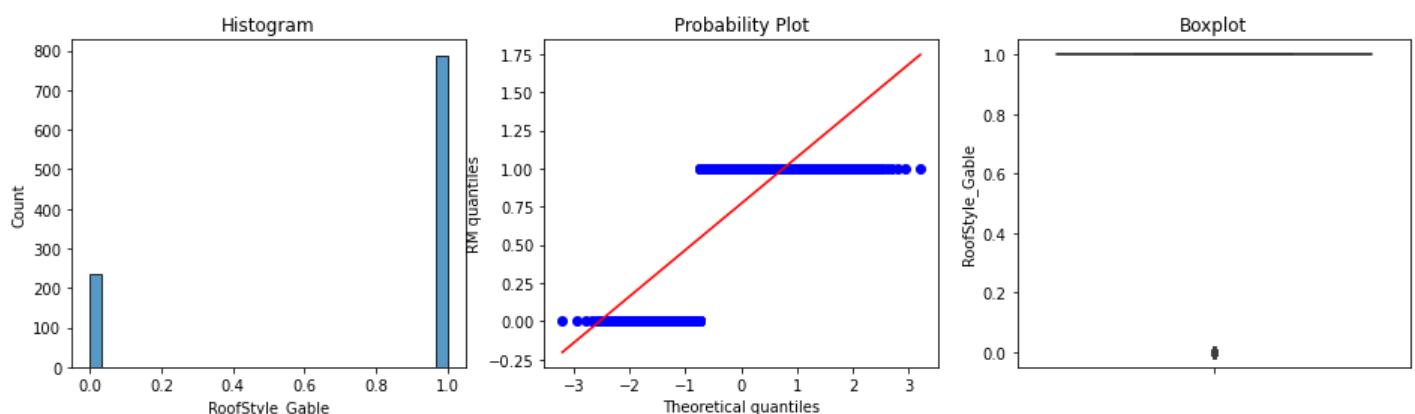
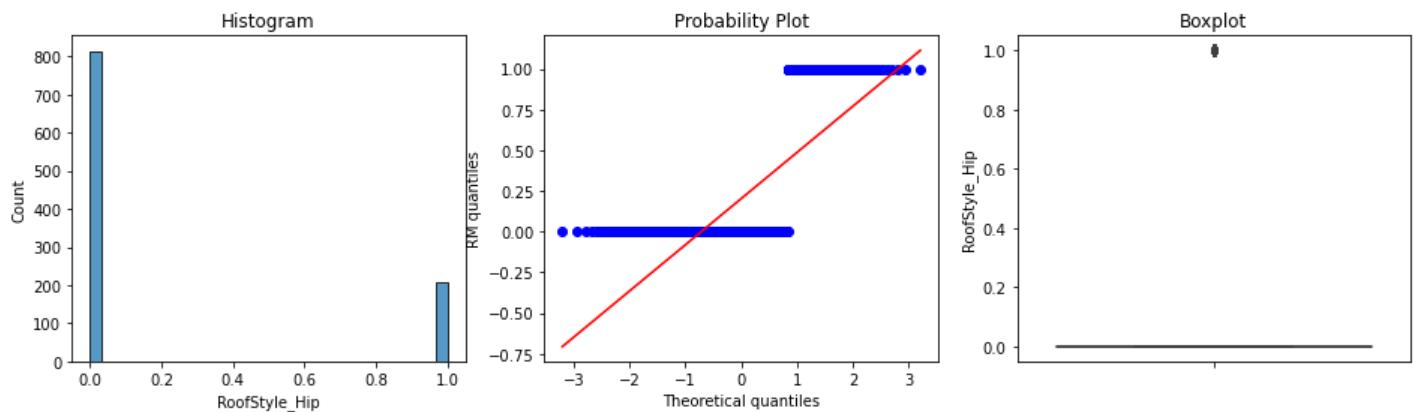
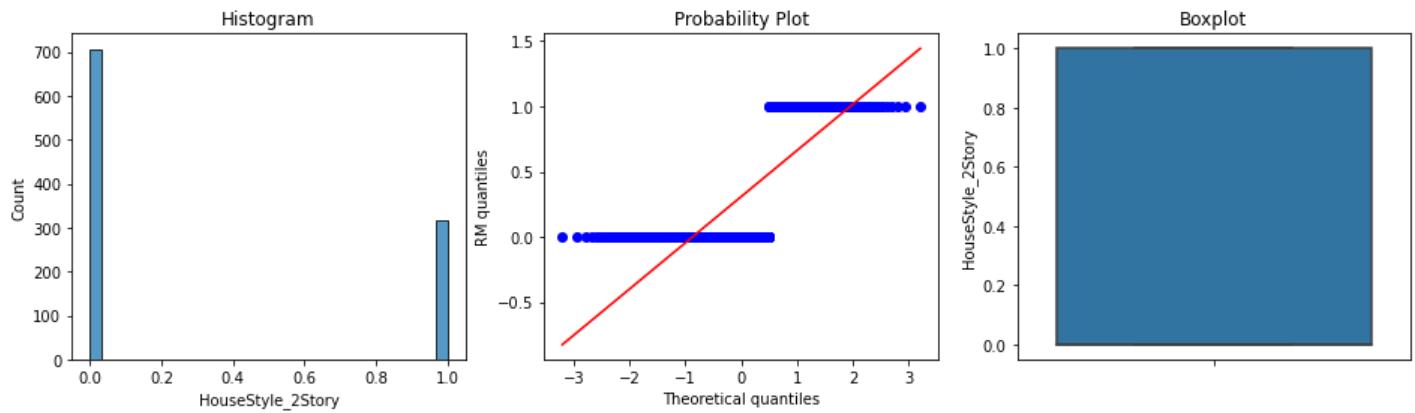


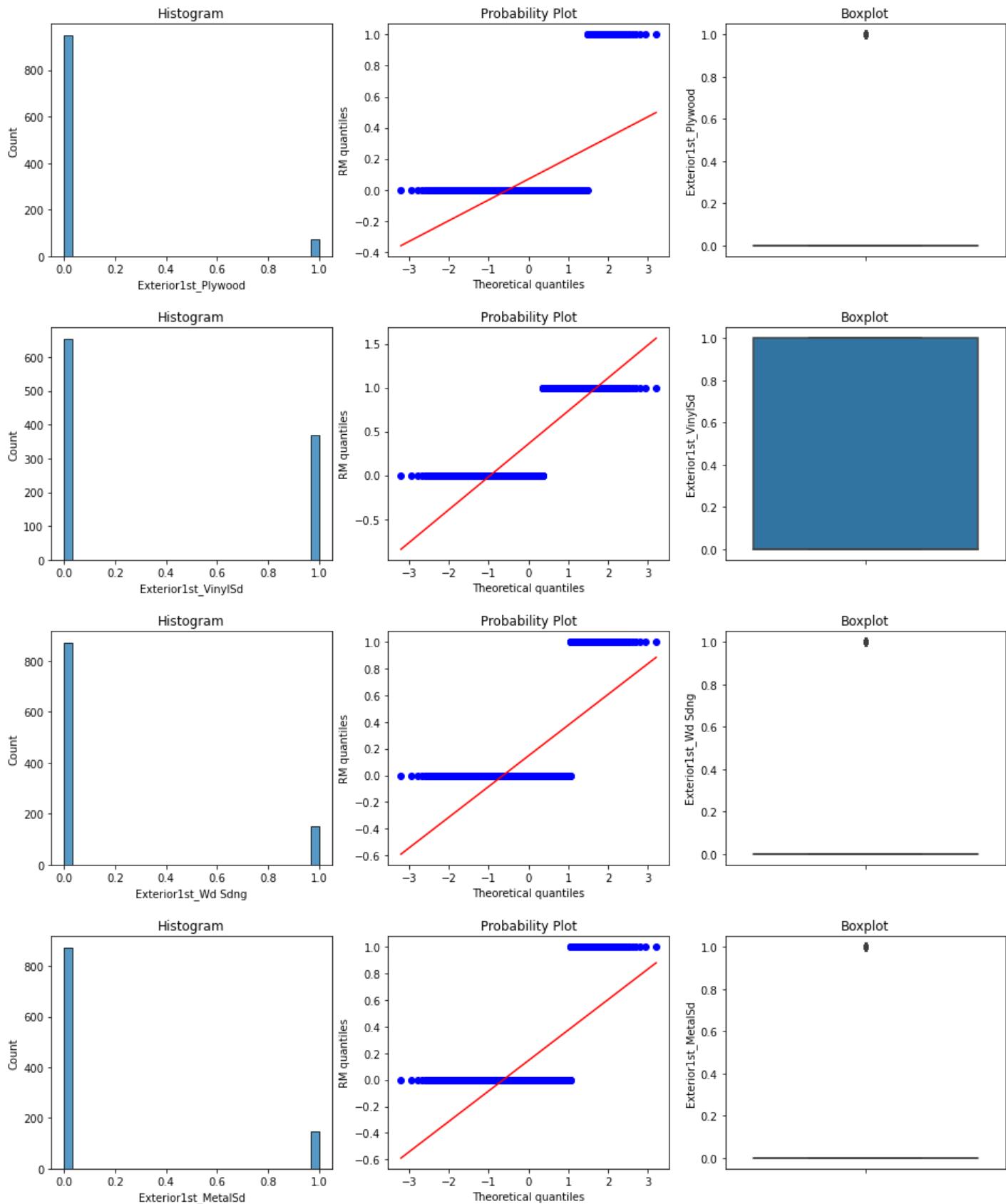


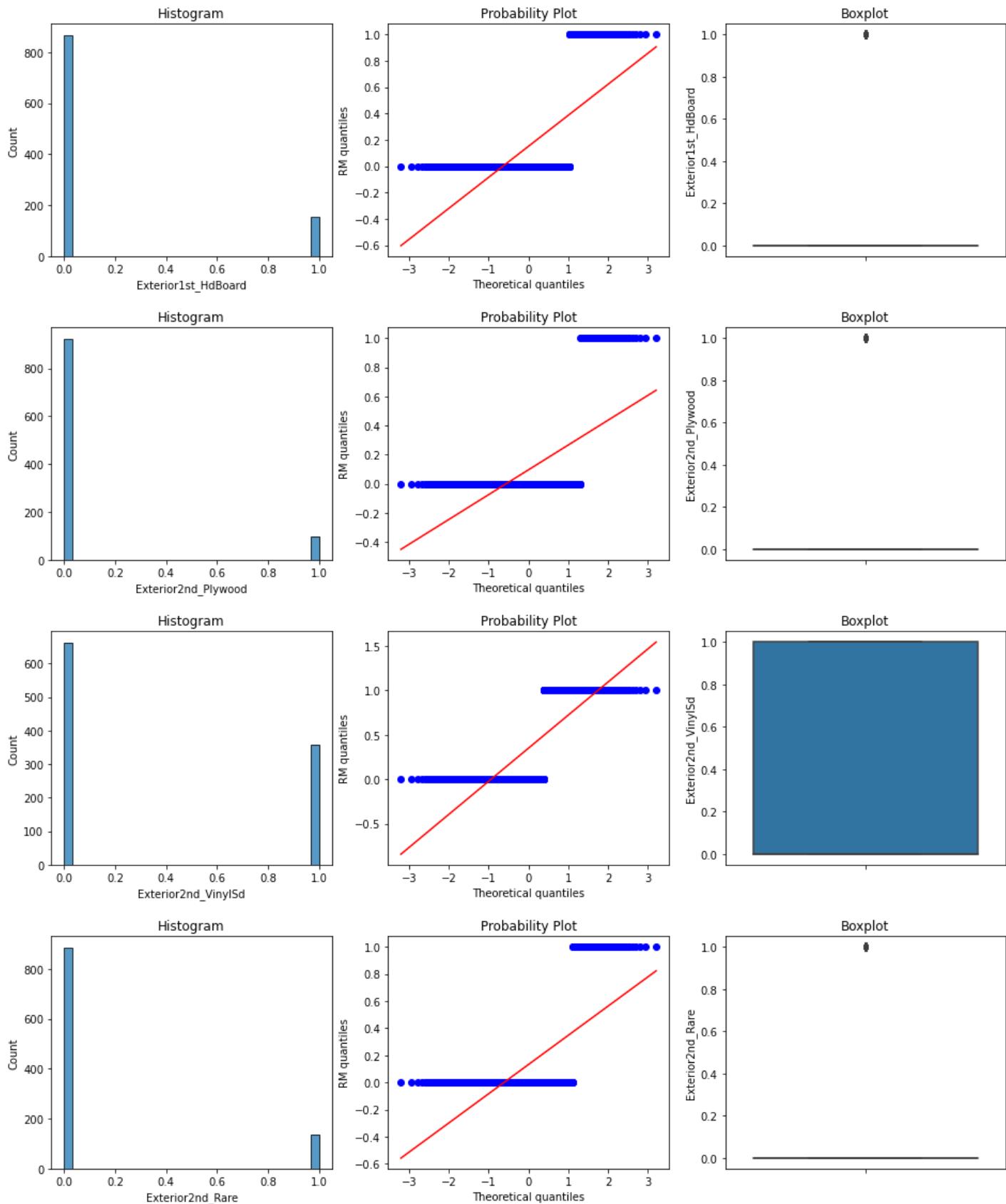


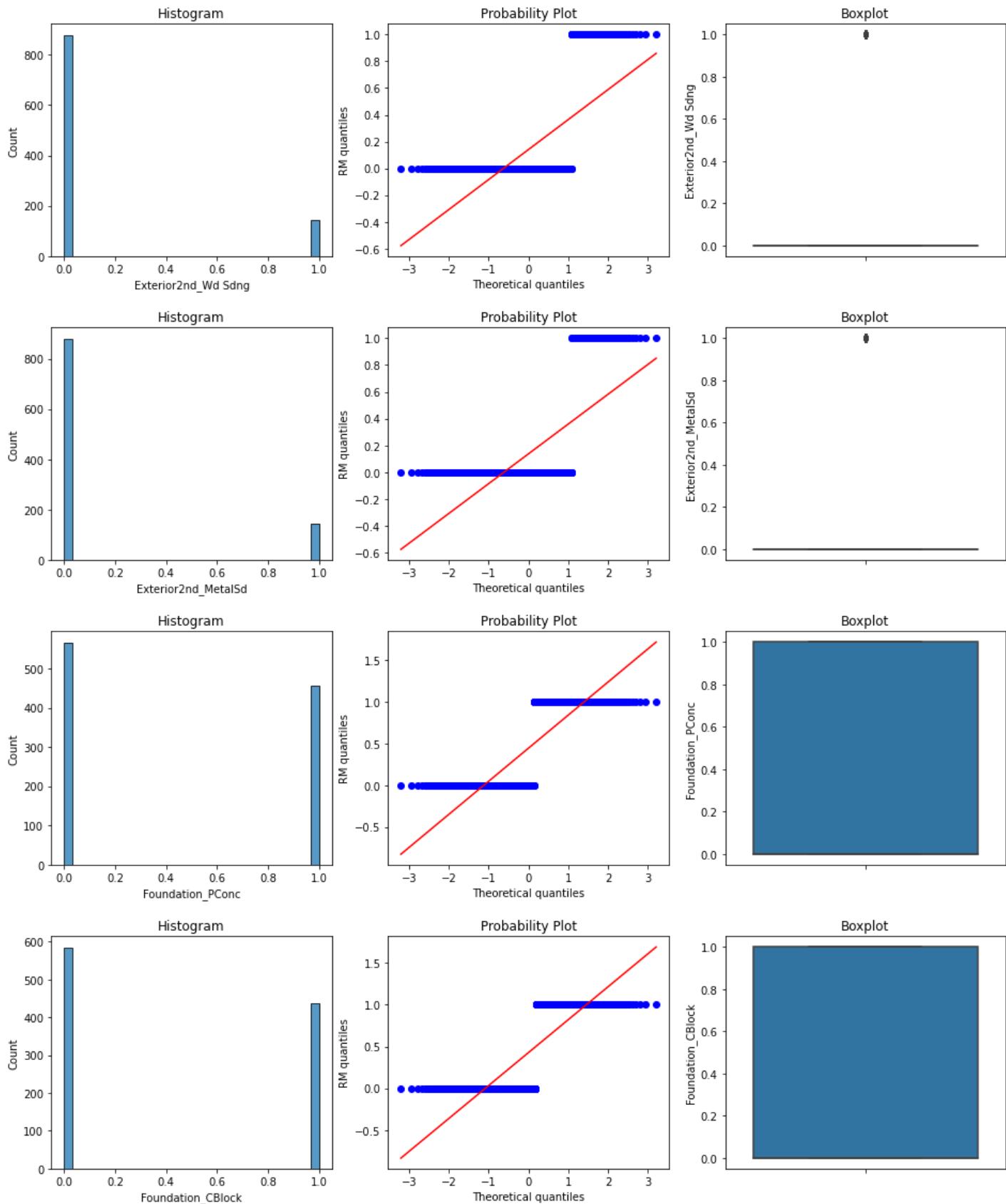


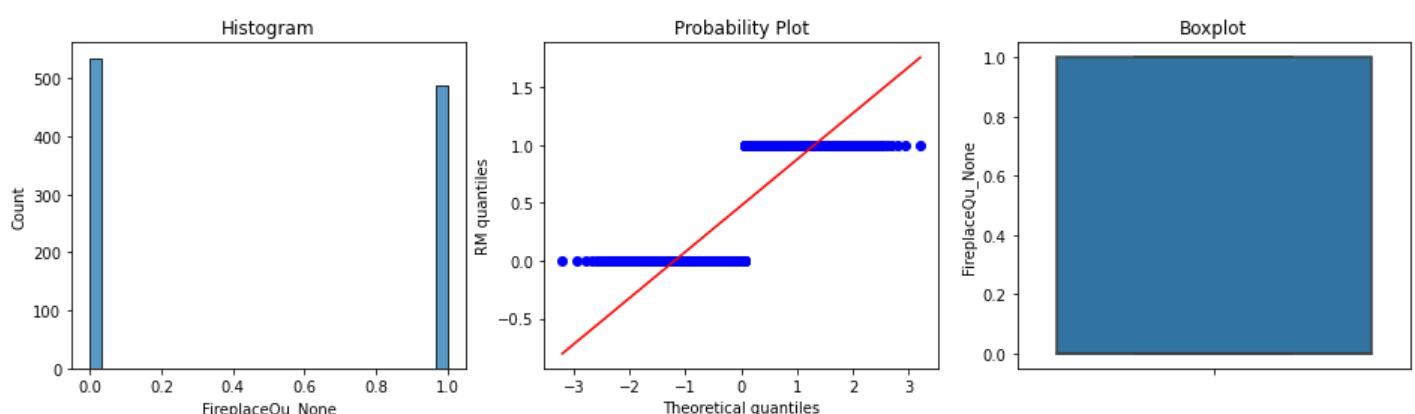
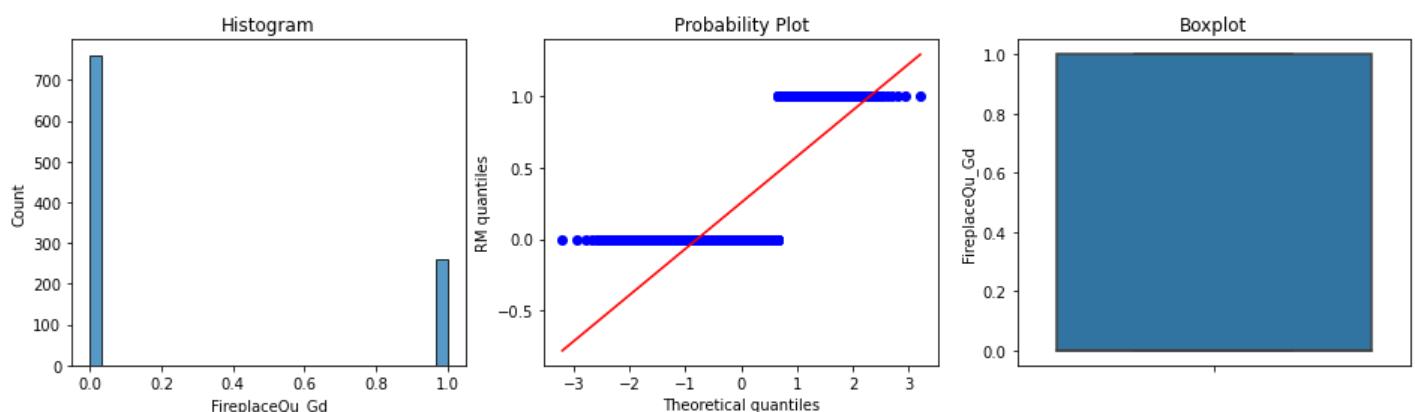
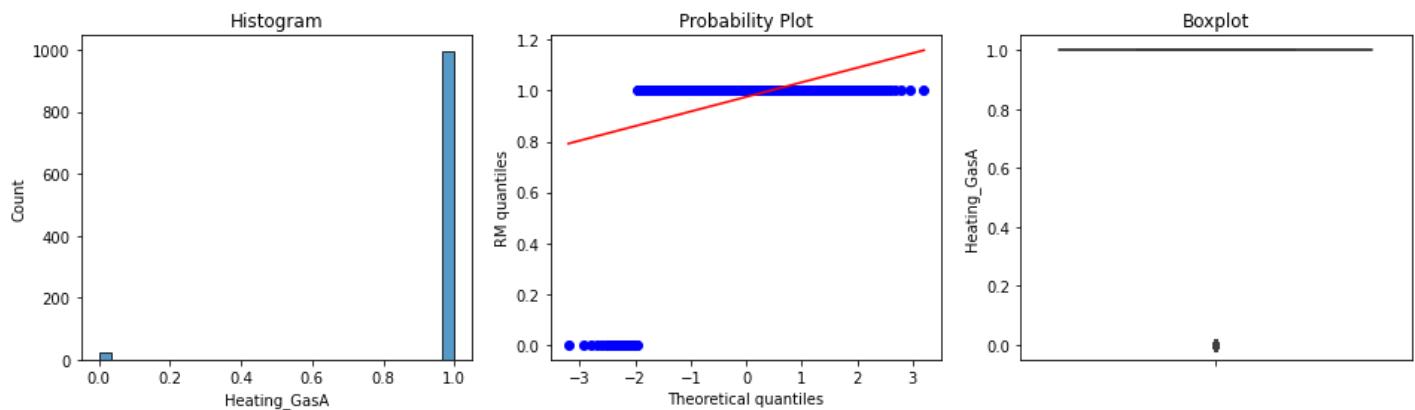
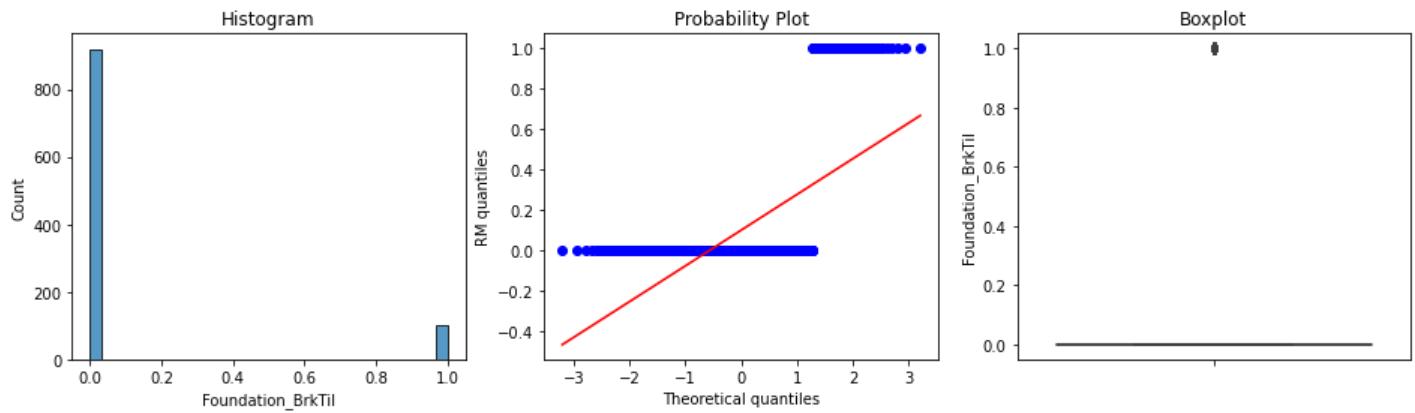


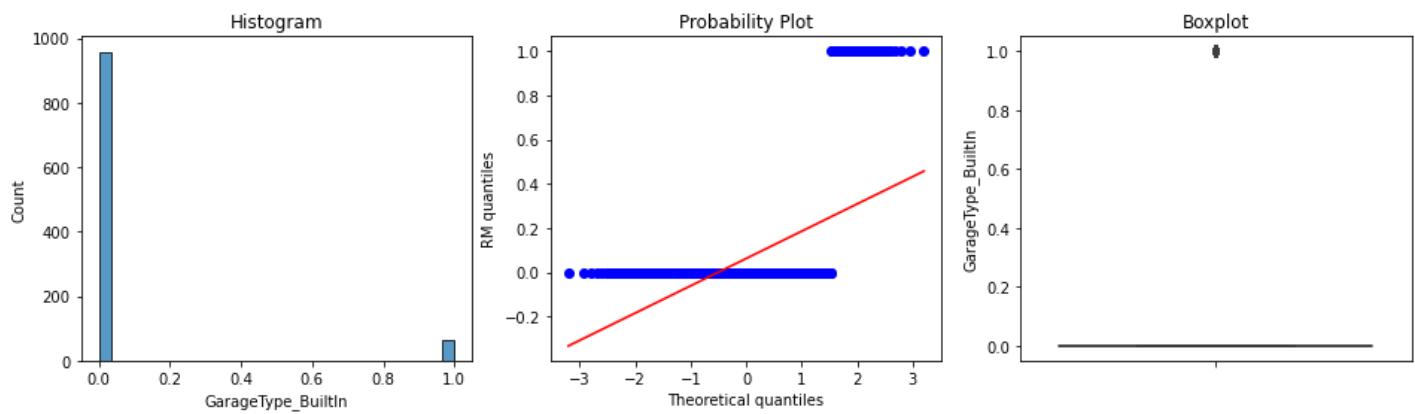
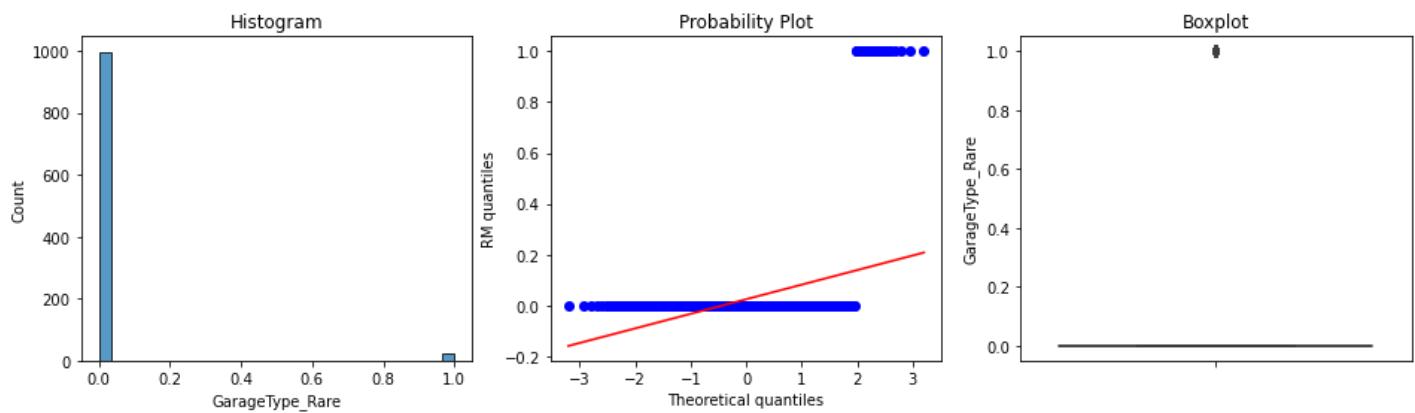
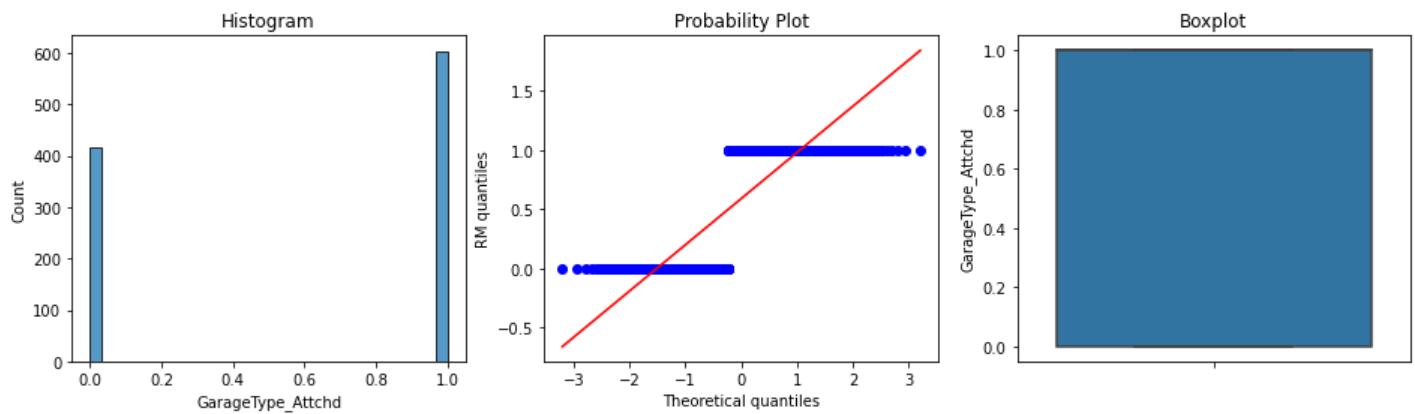
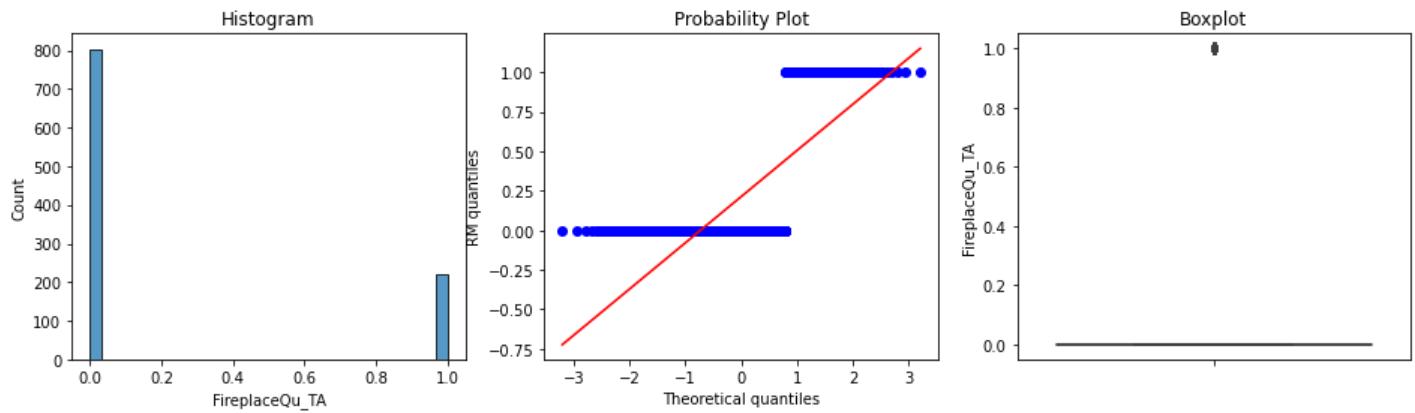


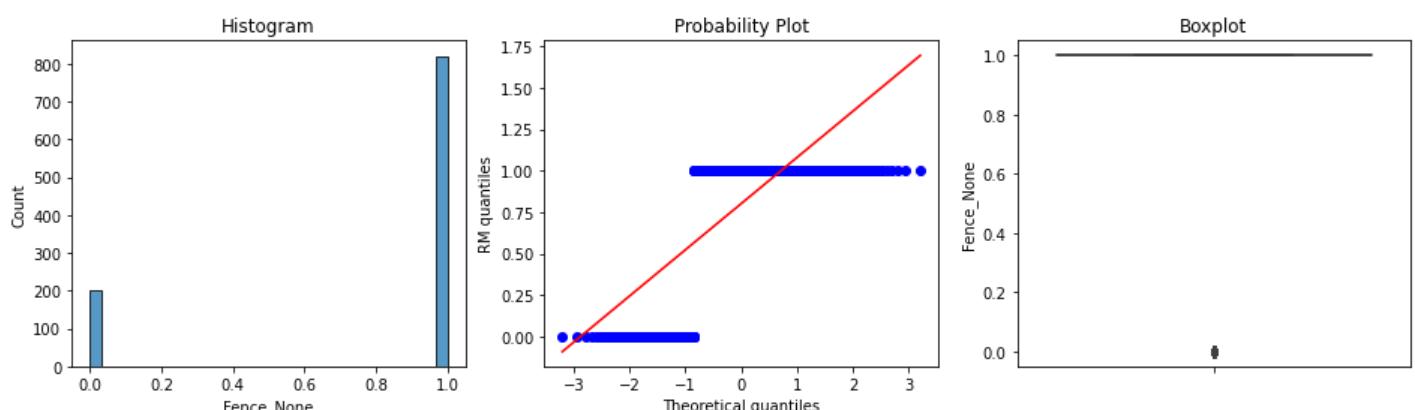
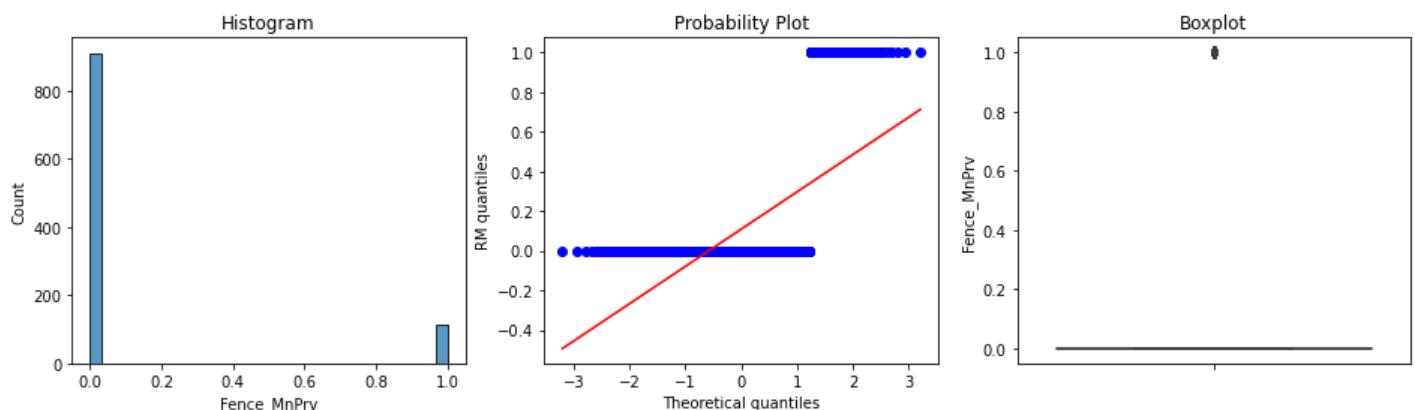
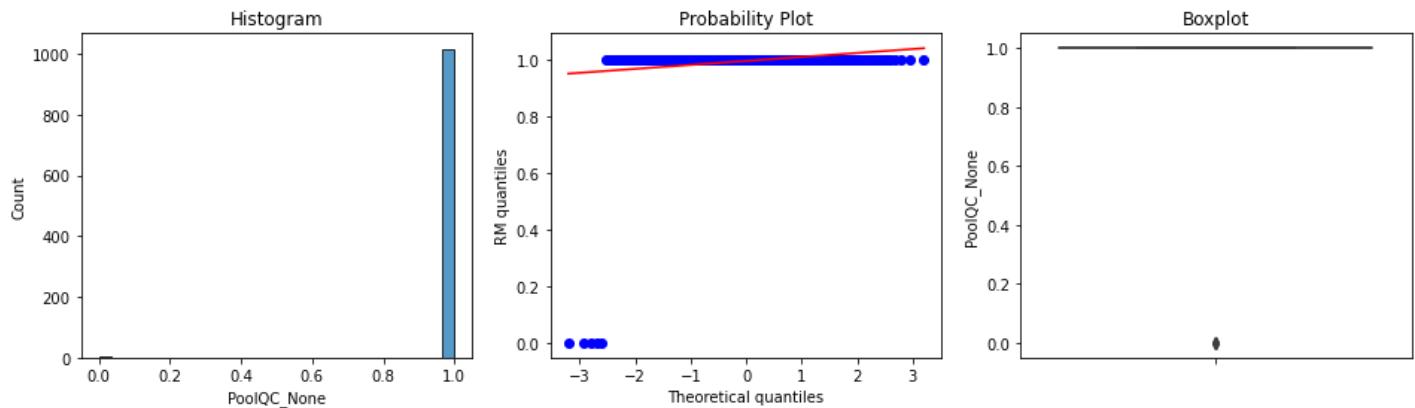
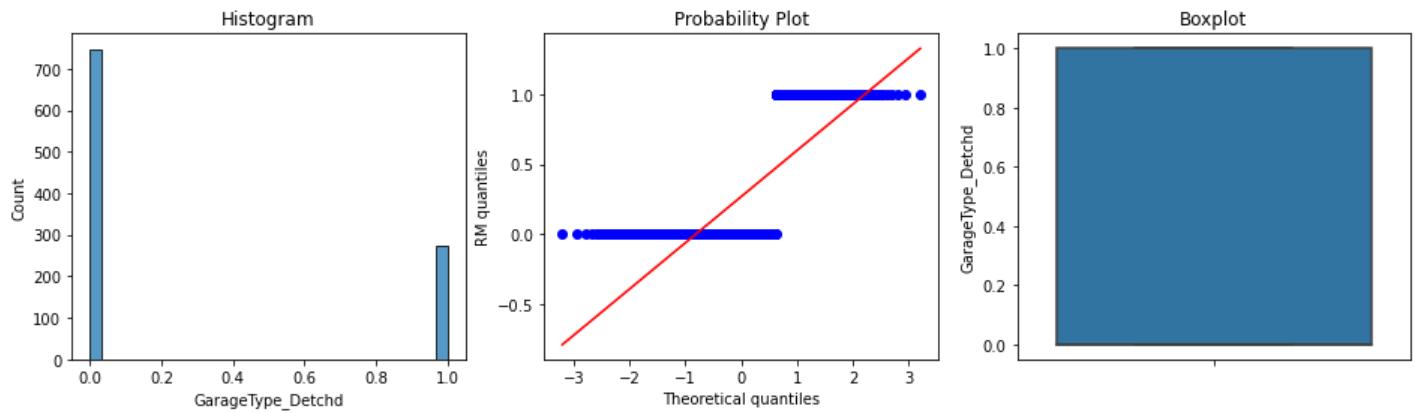


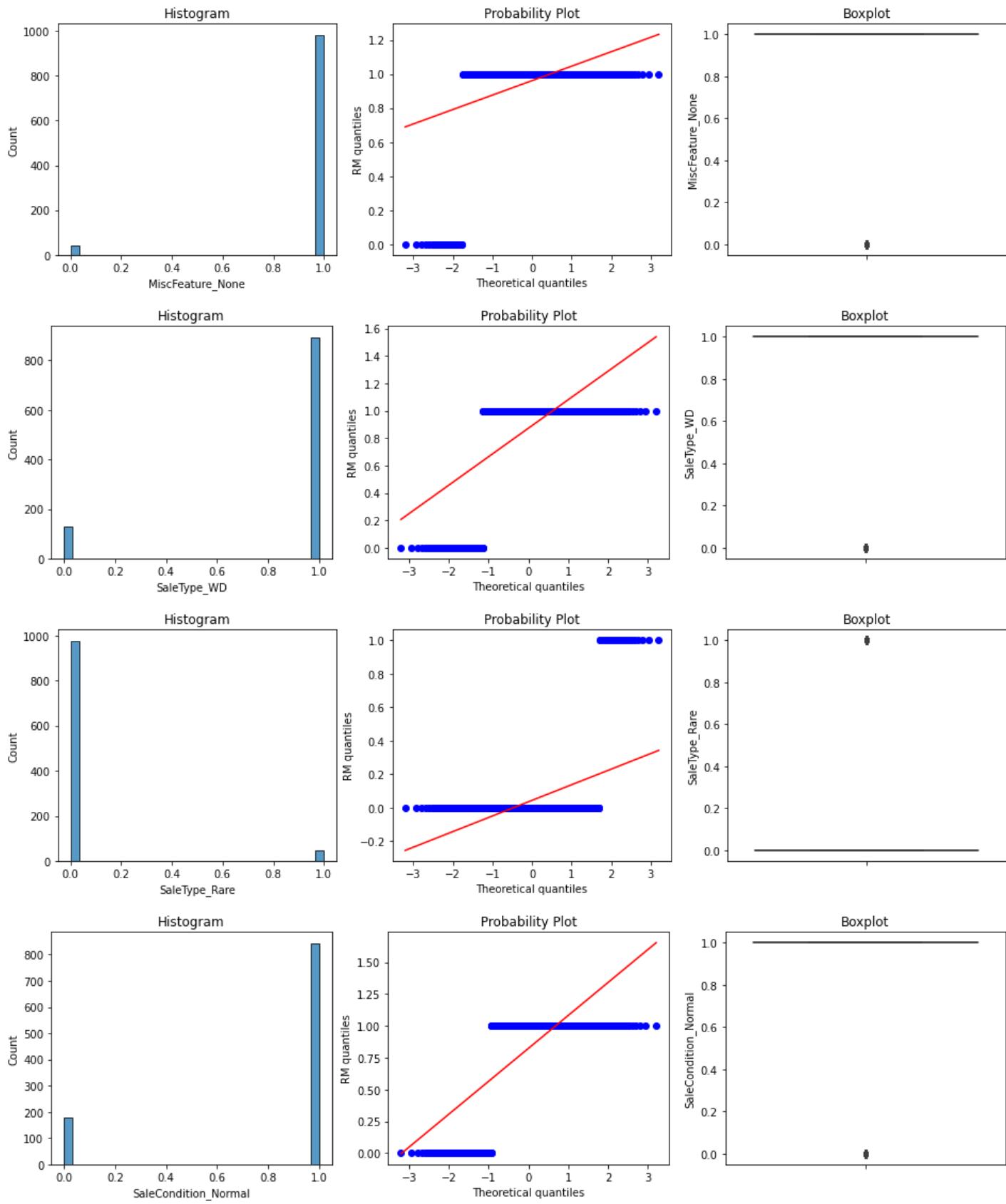


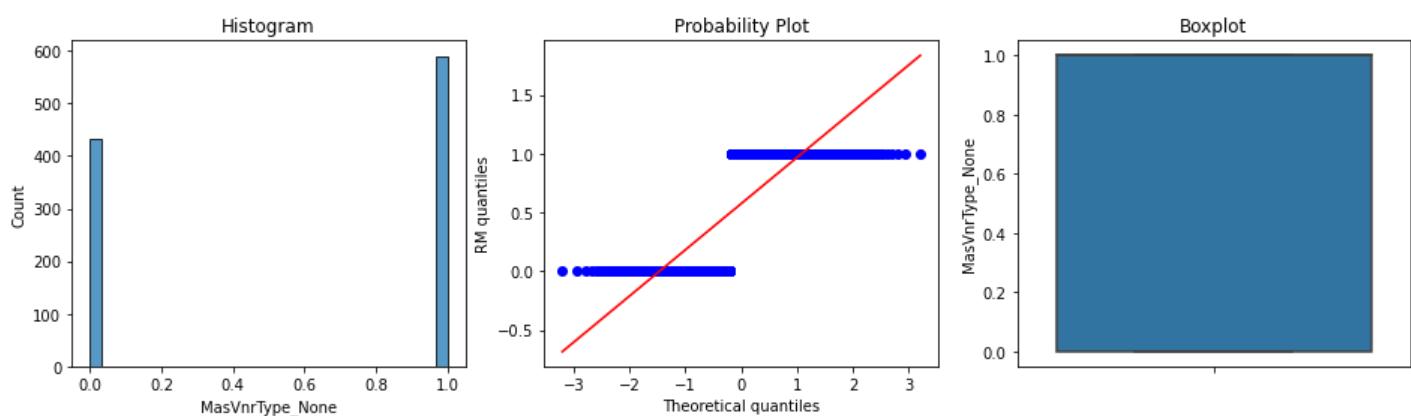
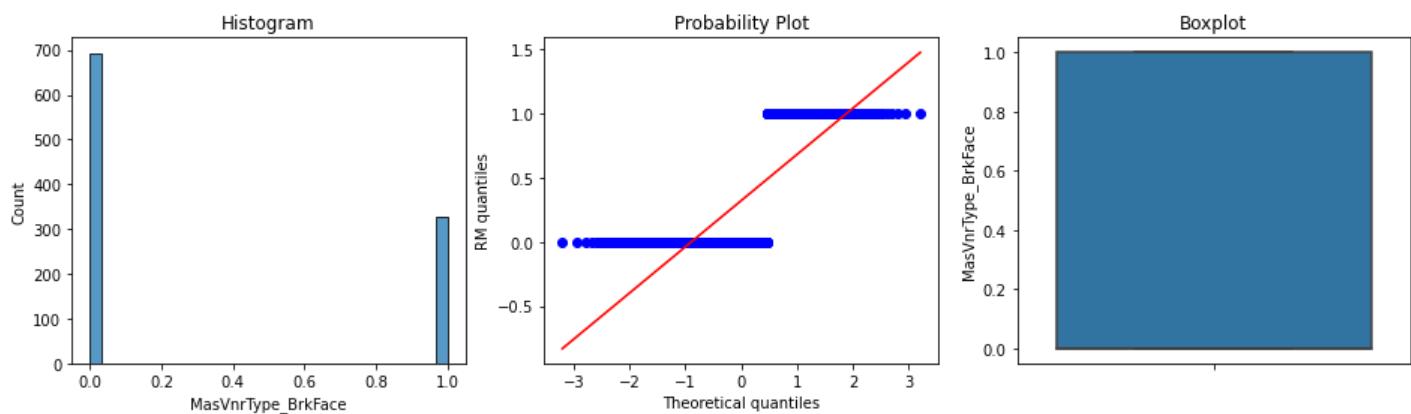
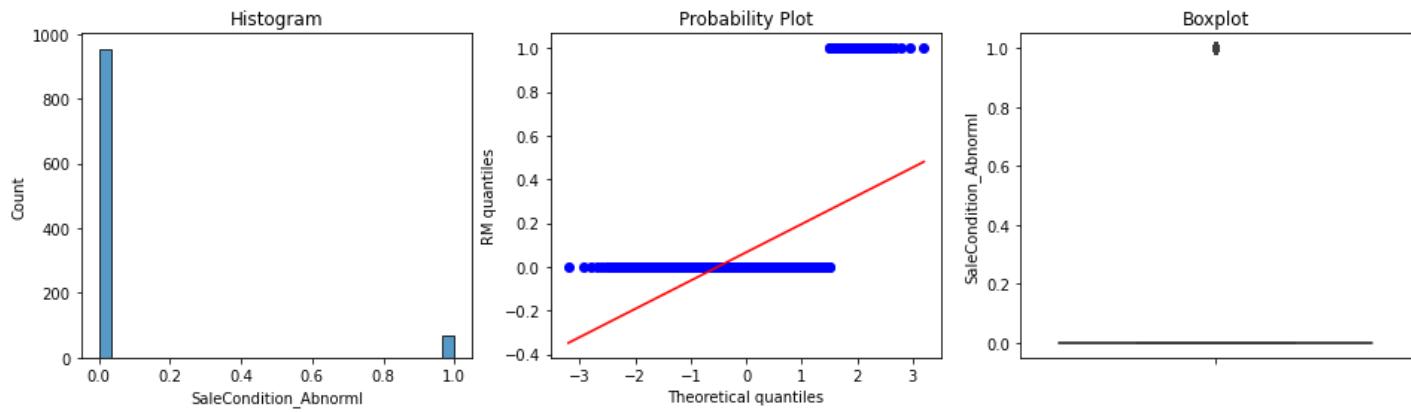
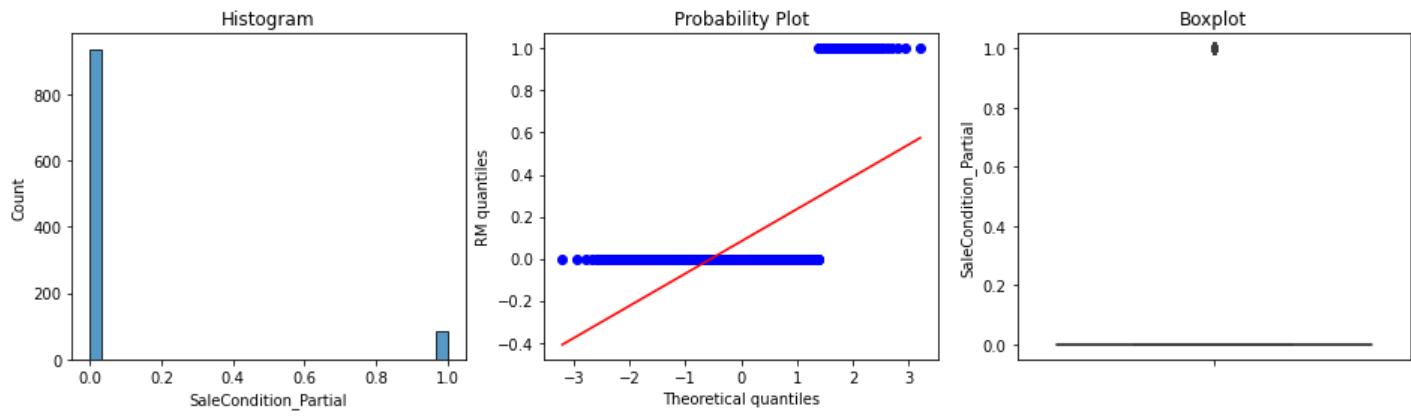


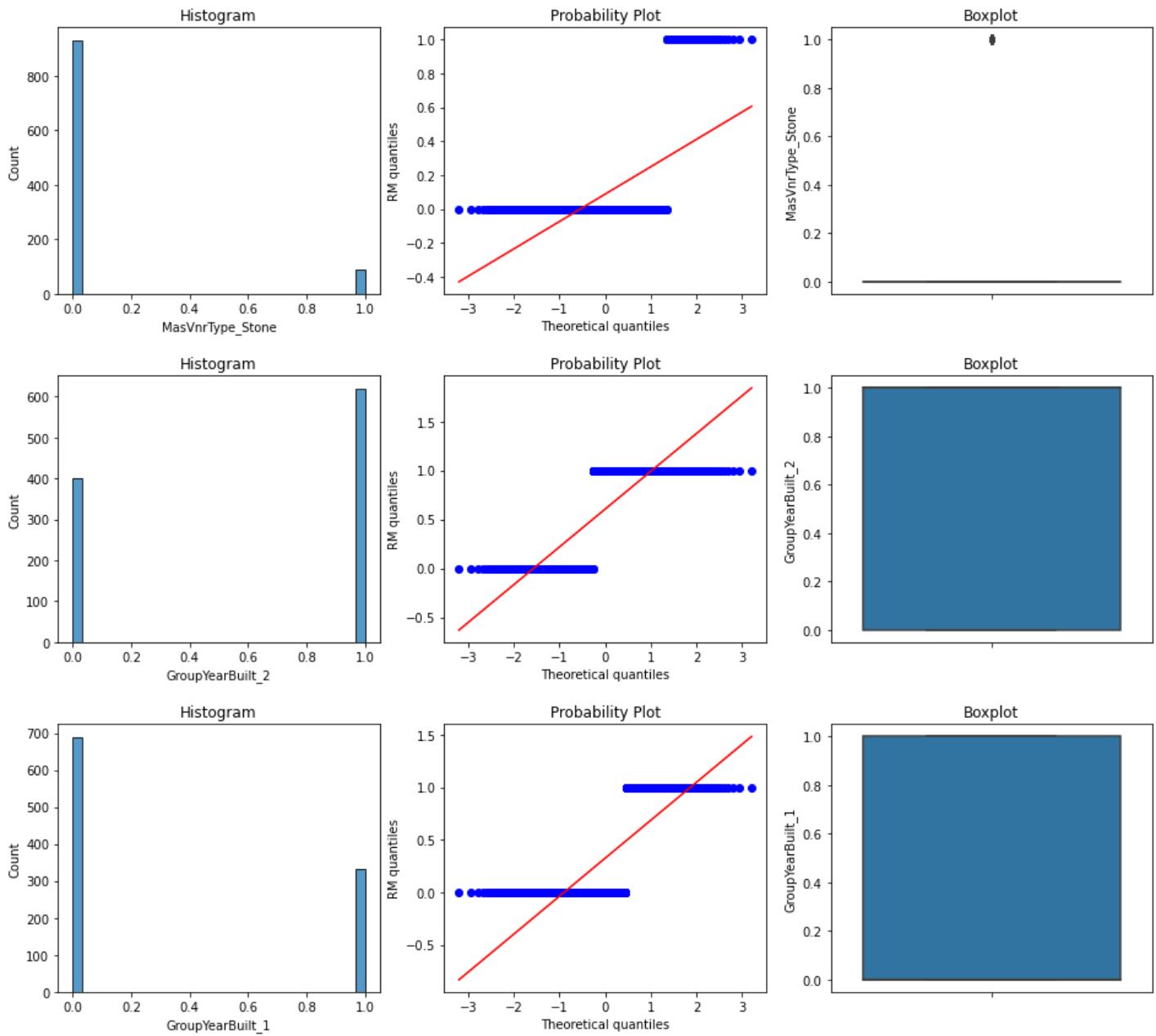












In [263]:

```
numeric_features = X_train.dtypes[X_train.dtypes != "object"].index

skewed_df = X_train[numeric_features].apply(lambda x: skew(x.dropna())).sort_values(ascending=True)
skewness = pd.DataFrame({'Skew' : abs(skewed_df)})
skewness_table = skewness[skewness['Skew'] > 0.75]
```

In [263]:

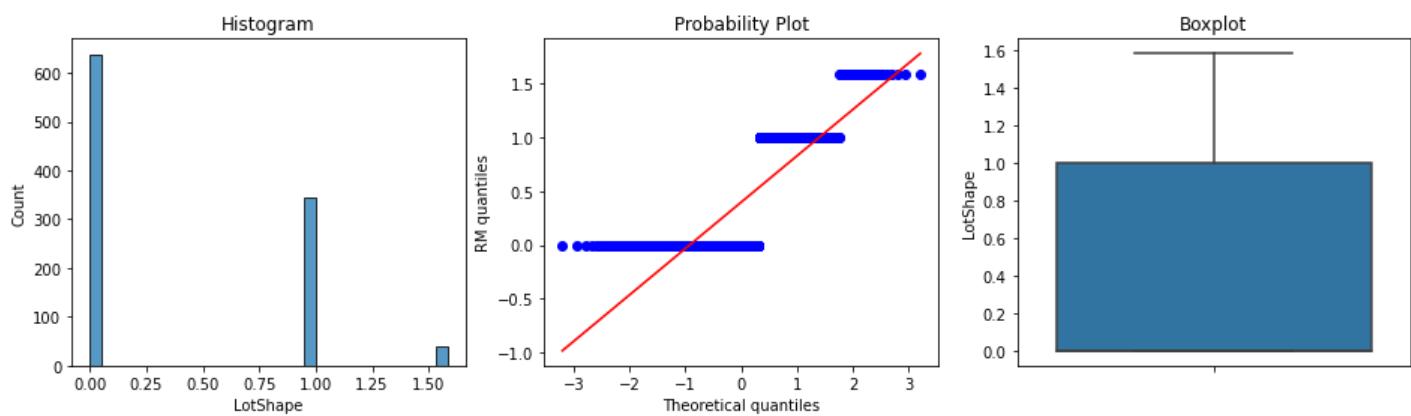
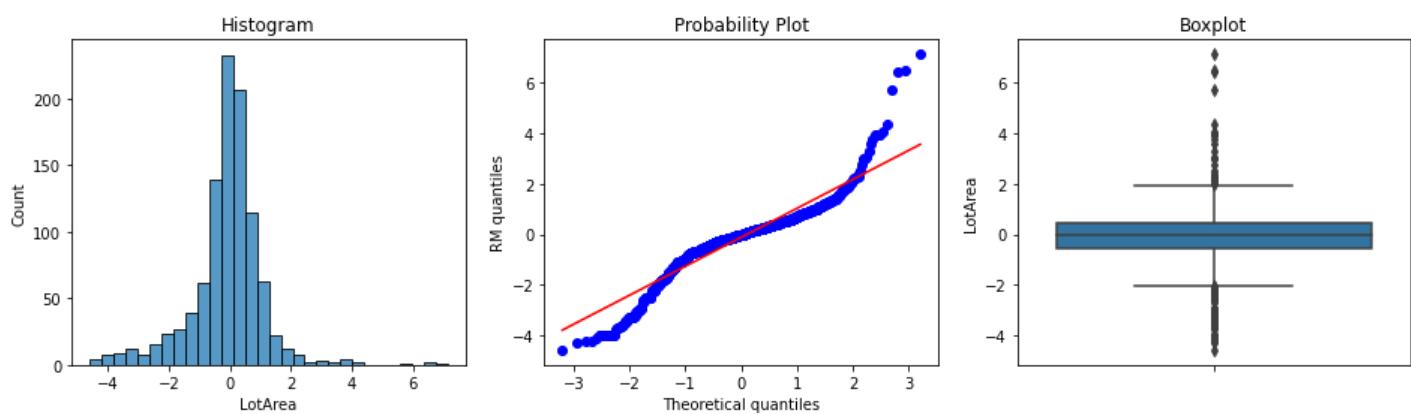
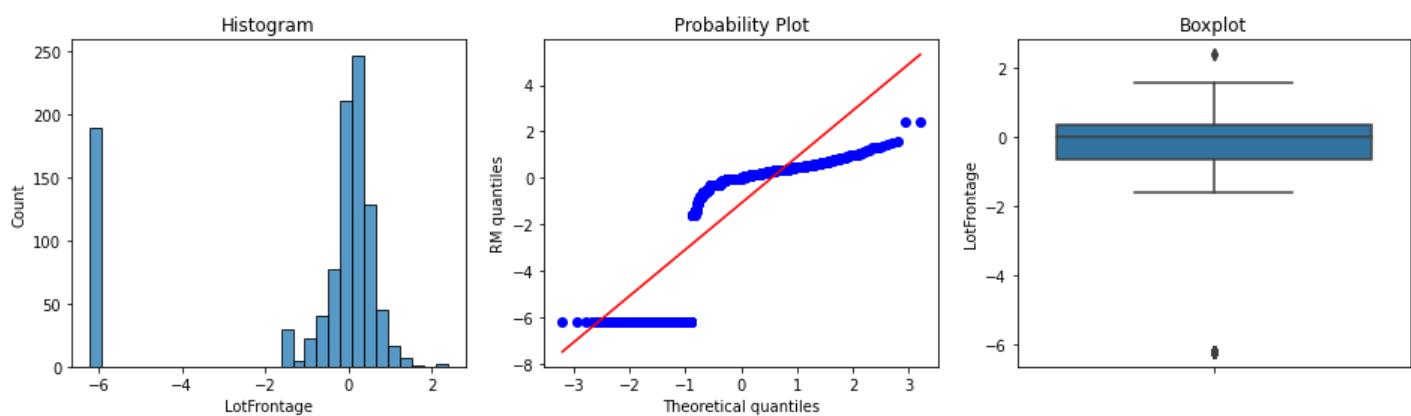
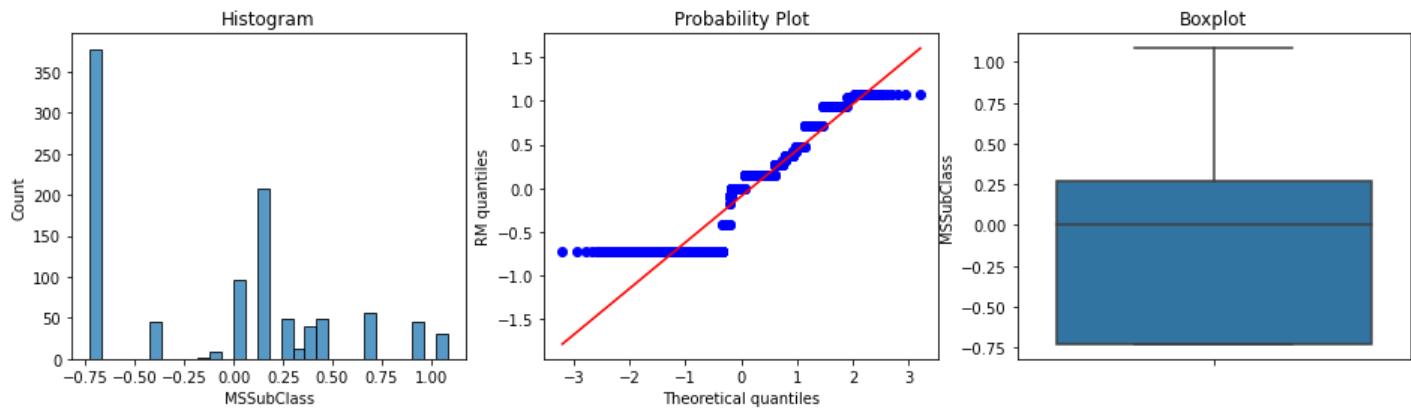
```
X_train[skewed_df.index] = np.log1p(X_train[skewed_df.index])
X_test[skewed_df.index] = np.log1p(X_test[skewed_df.index])
df_test[skewed_df.index] = np.log1p(df_test[skewed_df.index])
```

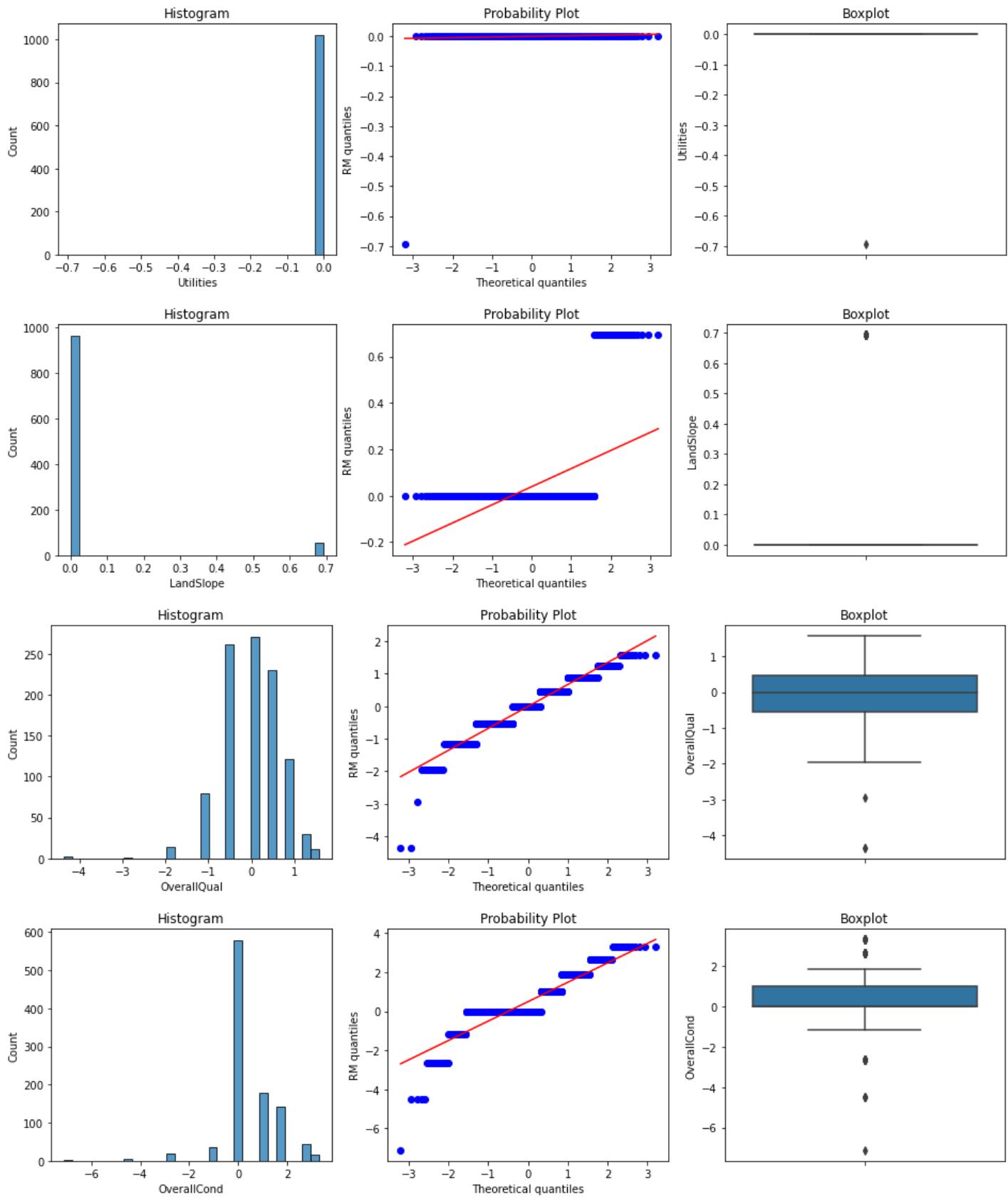
In [263]:

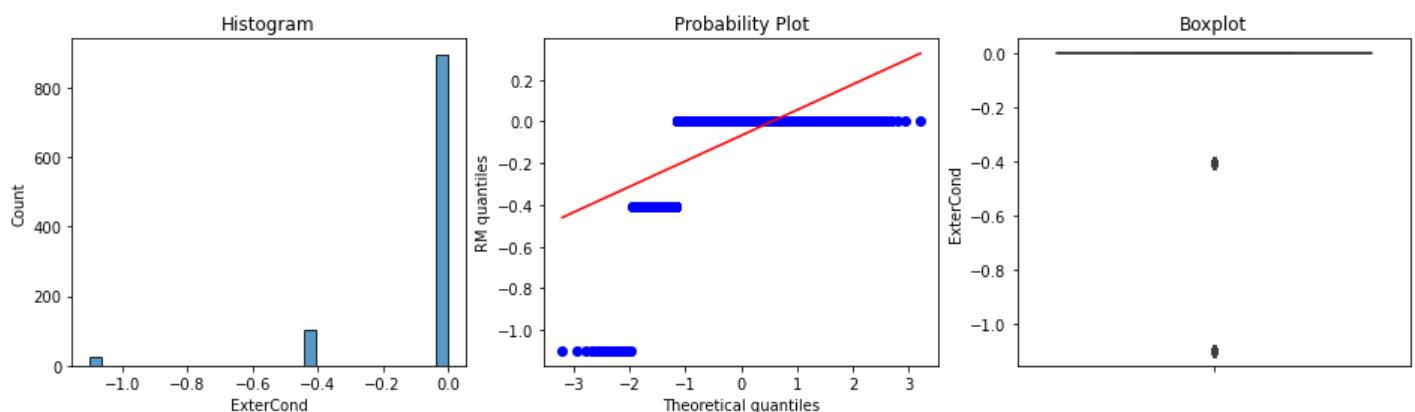
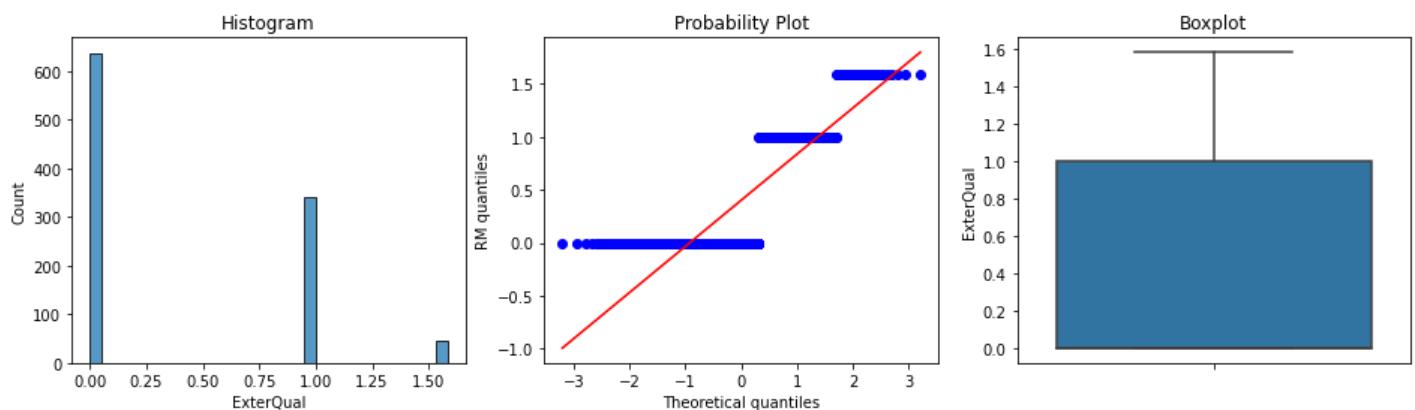
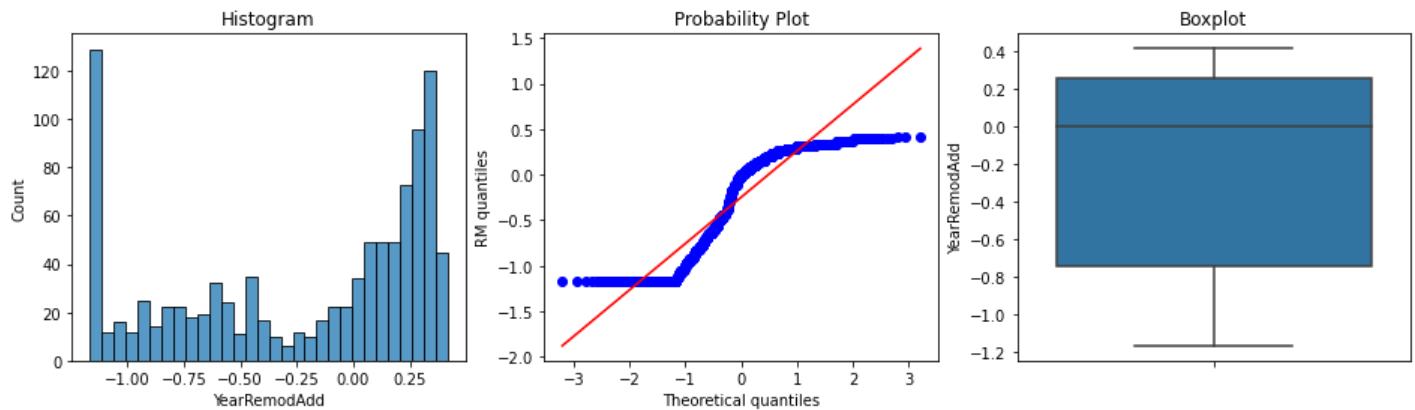
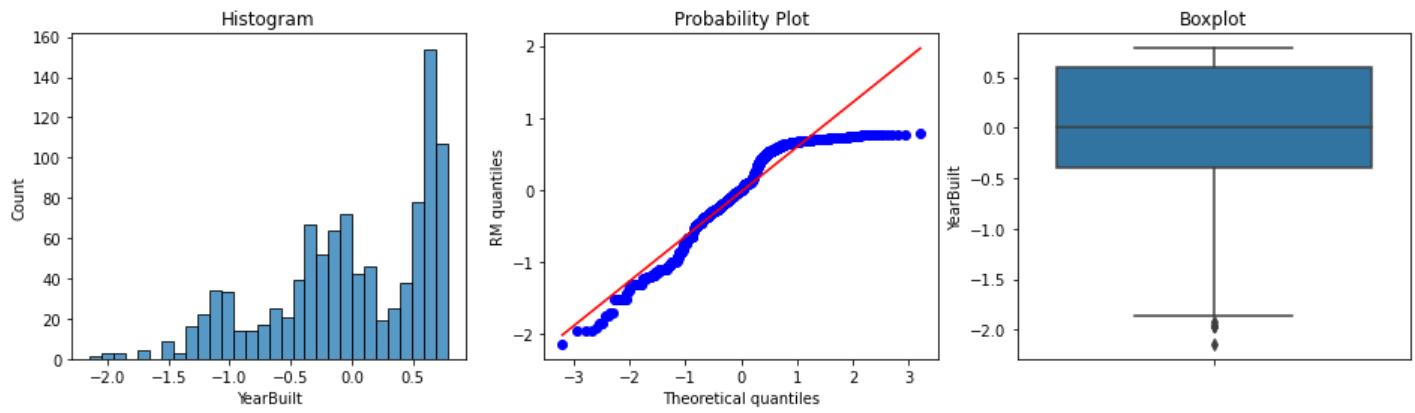
```
for col in X_train[X_train.columns]:
    scaler = RobustScaler()
    X_train[col] = scaler.fit_transform(X_train[[col]])
    X_test[col] = scaler.transform(X_test[[col]])
    df_test[col] = scaler.transform(df_test[[col]])
```

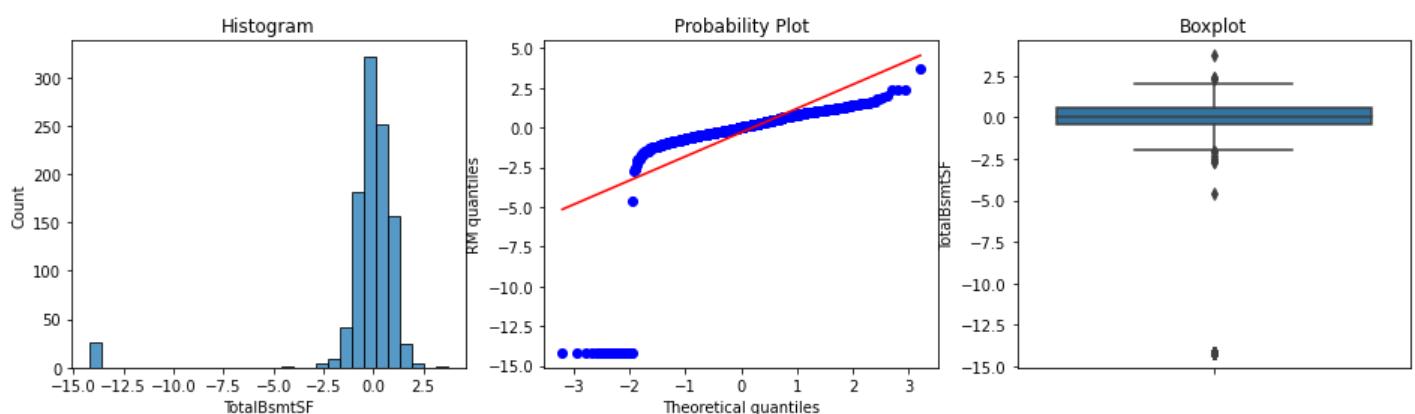
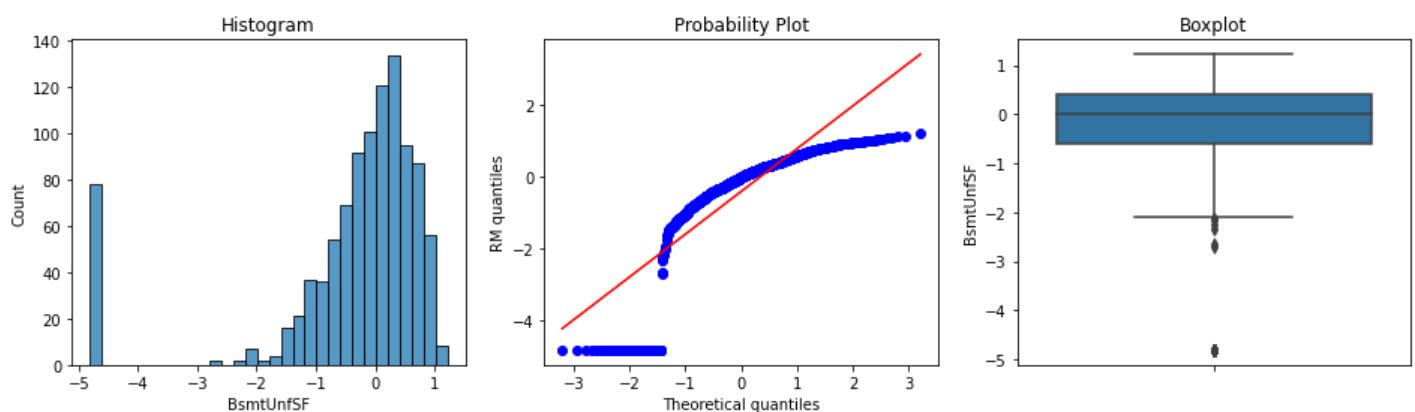
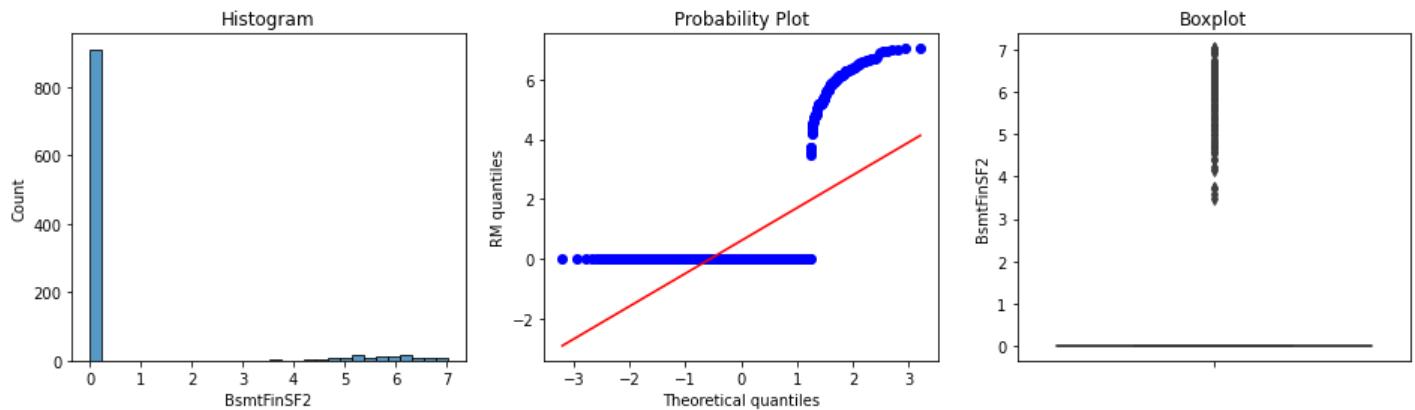
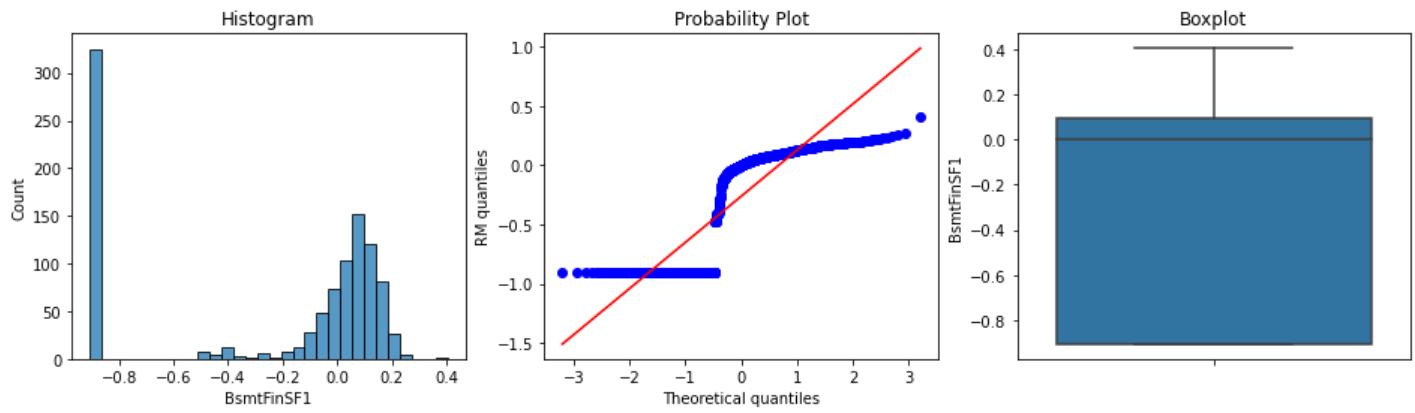
In [263]:

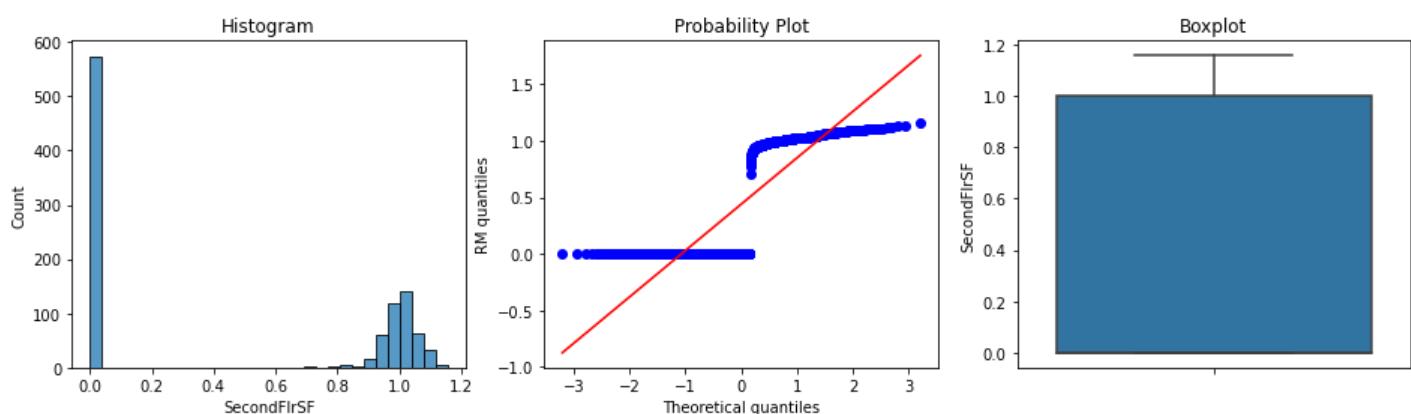
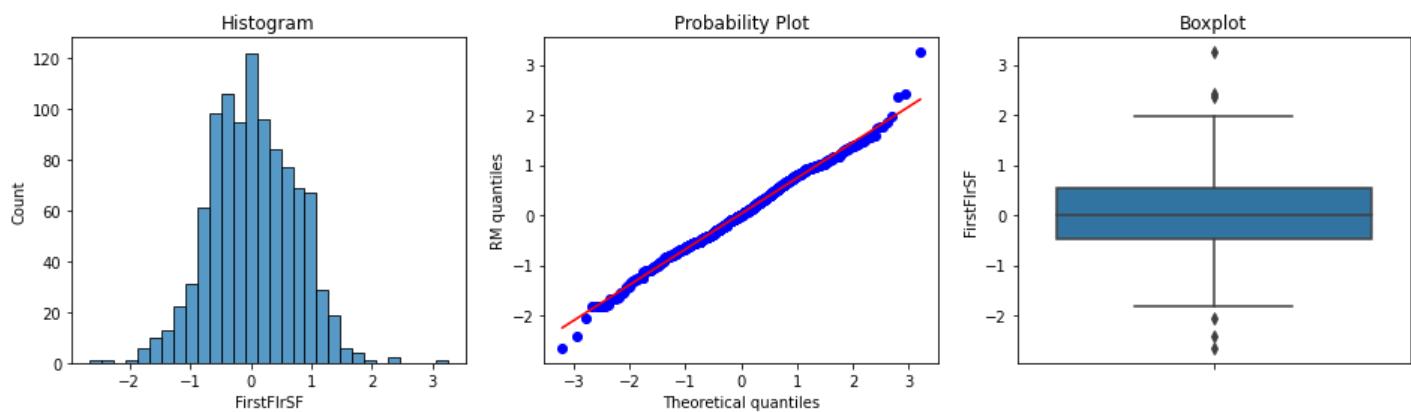
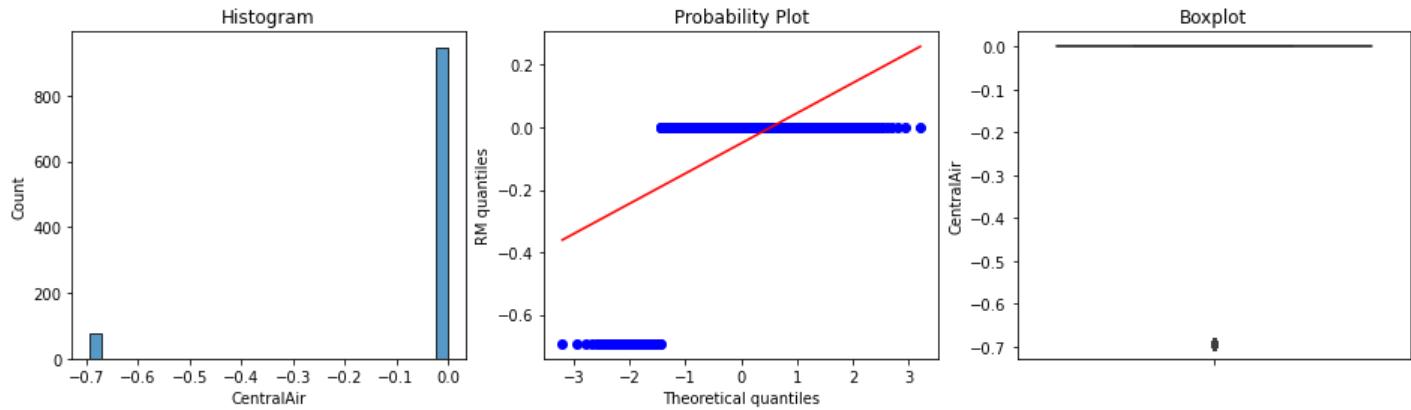
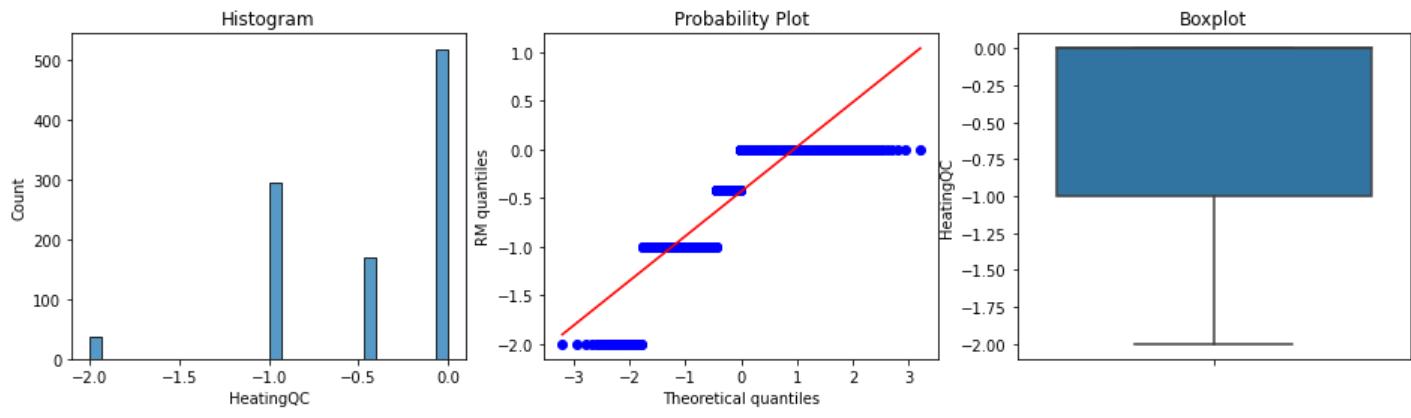
```
for i in X_train.columns:
    diagnostic_plots(X_train,i)
```

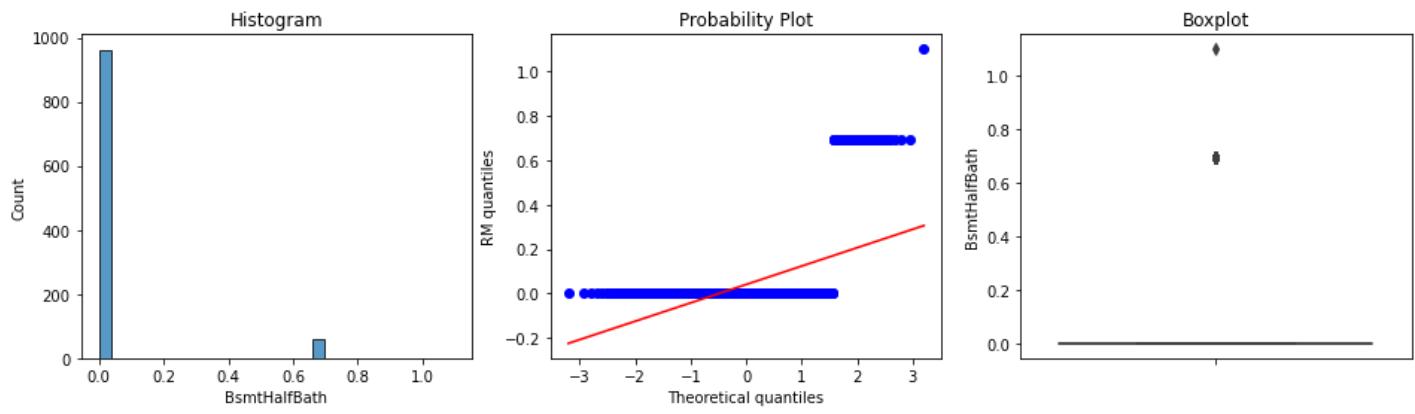
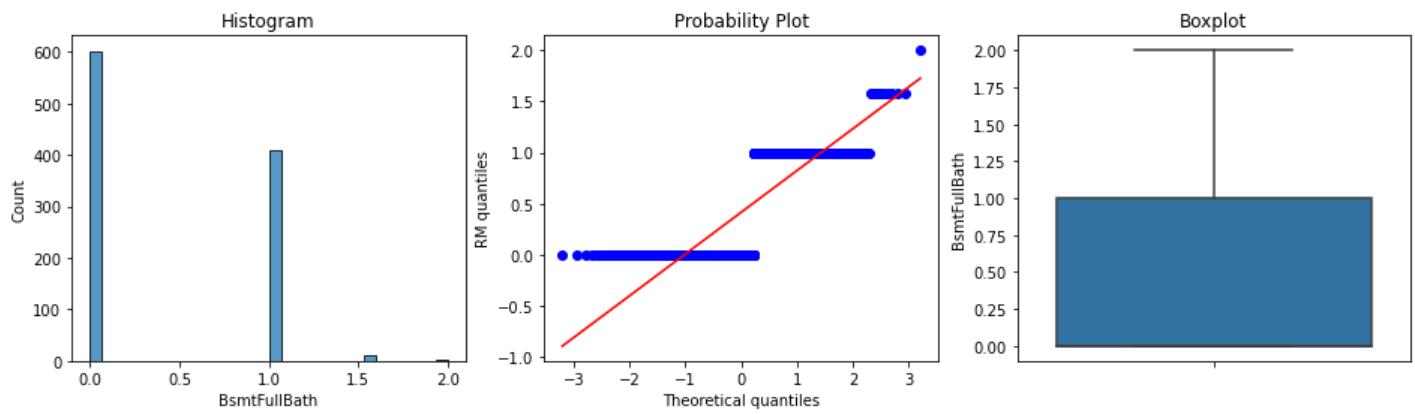
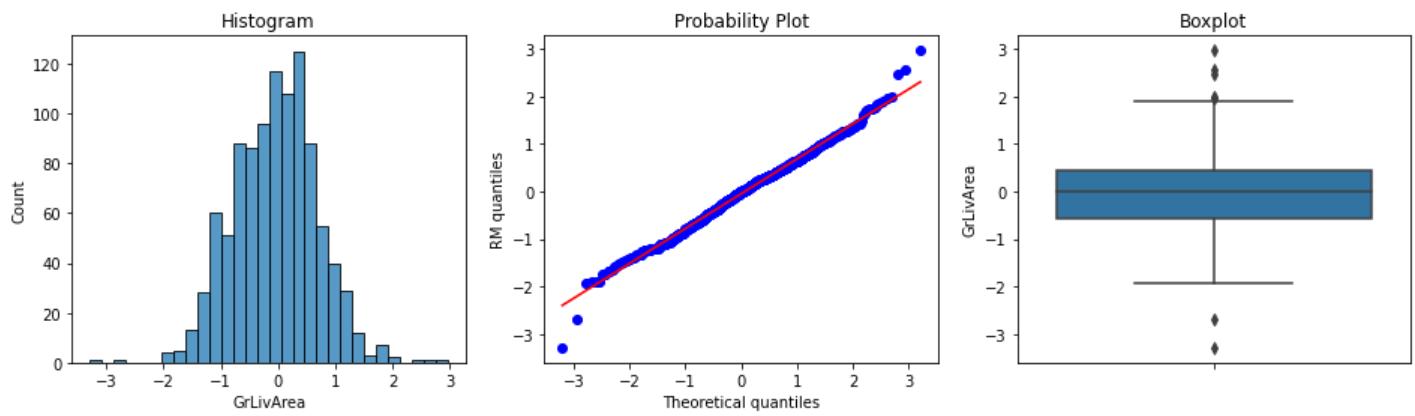
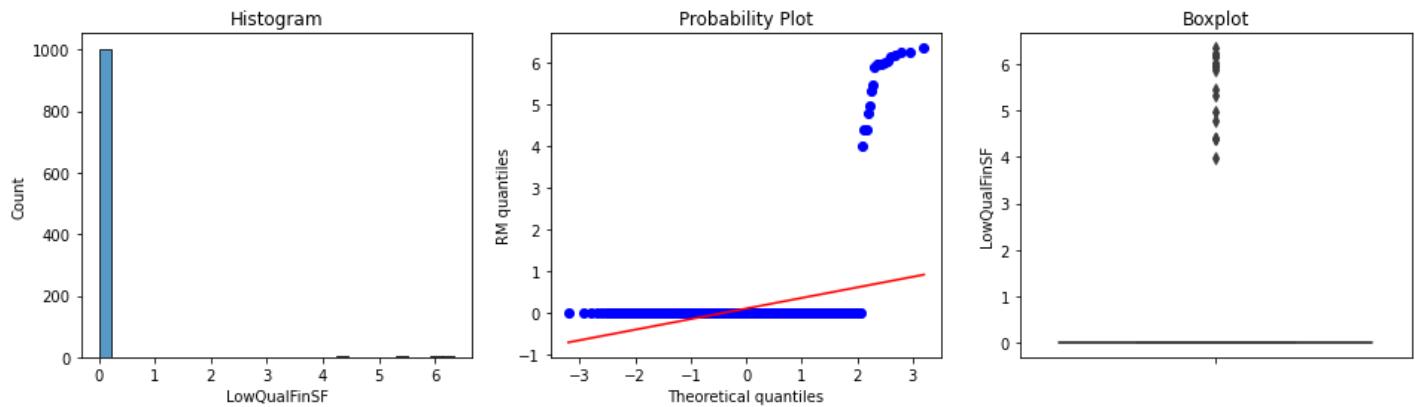


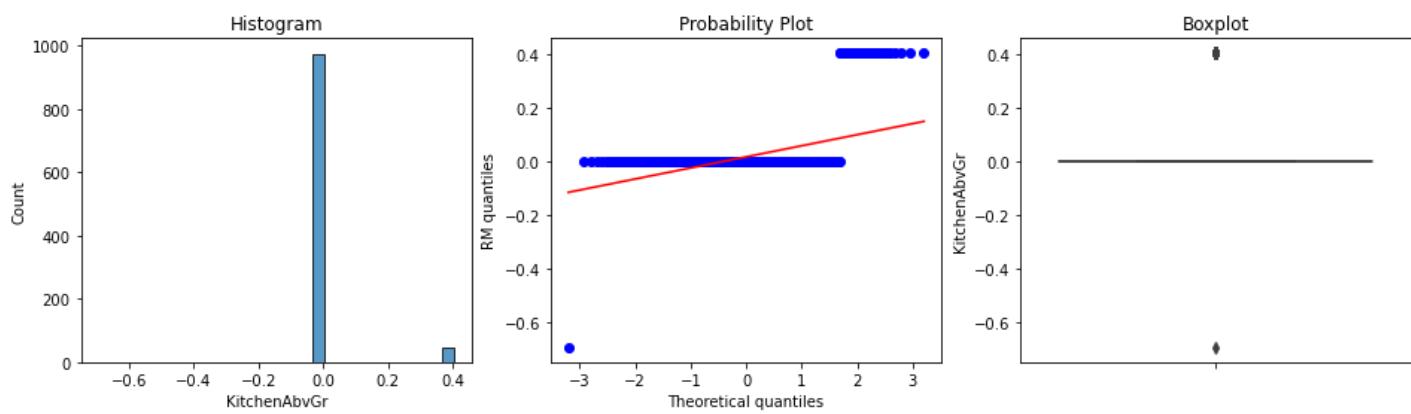
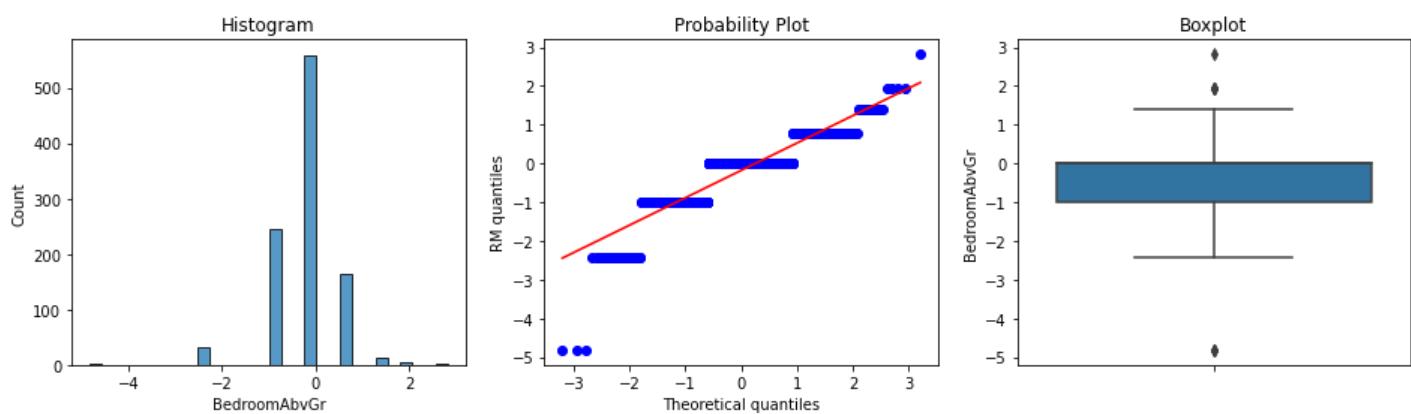
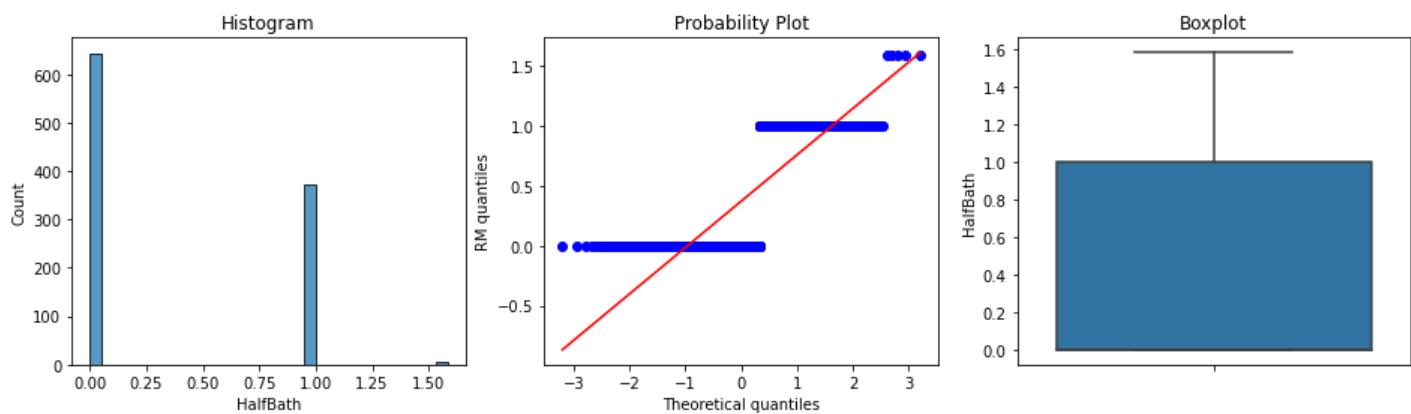
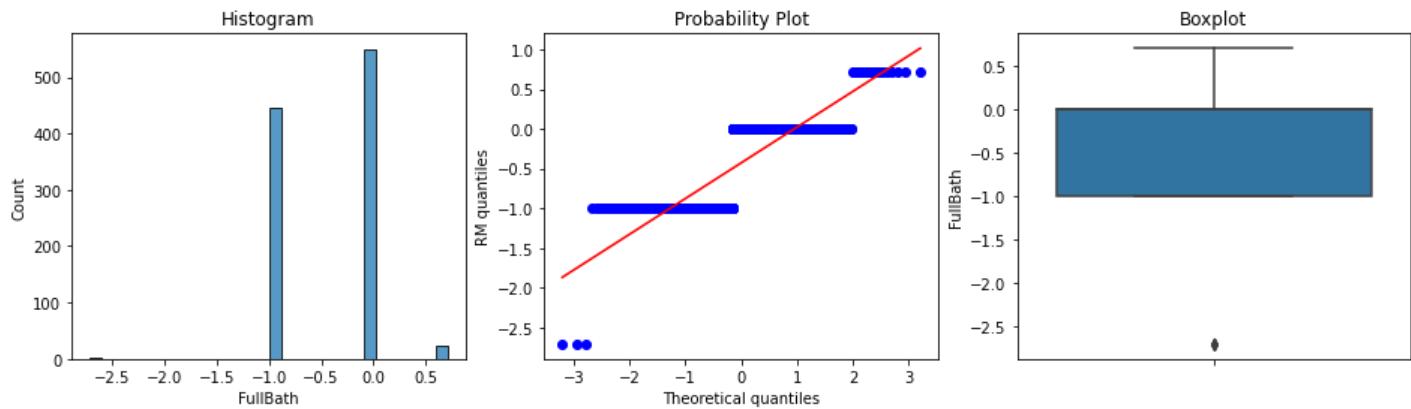


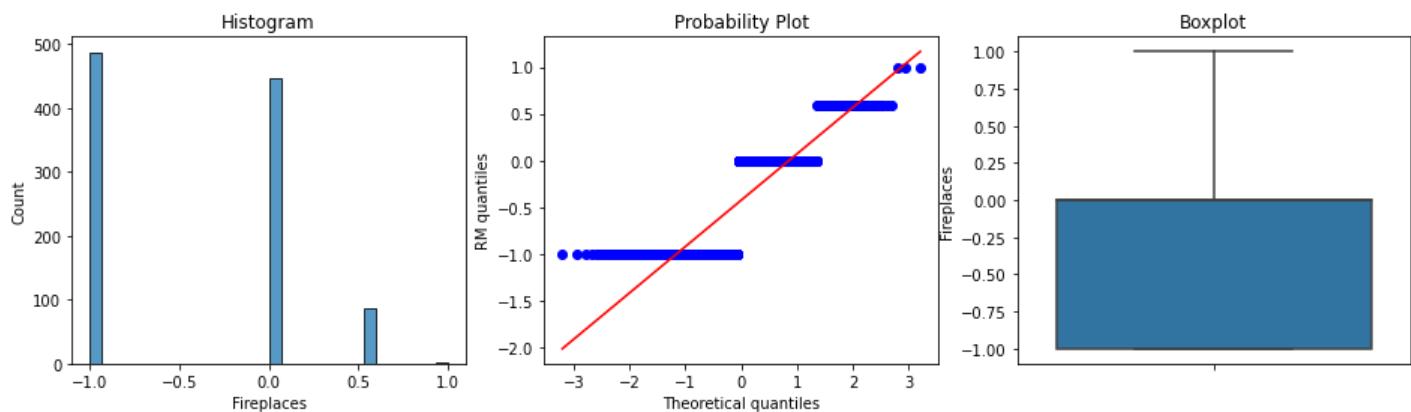
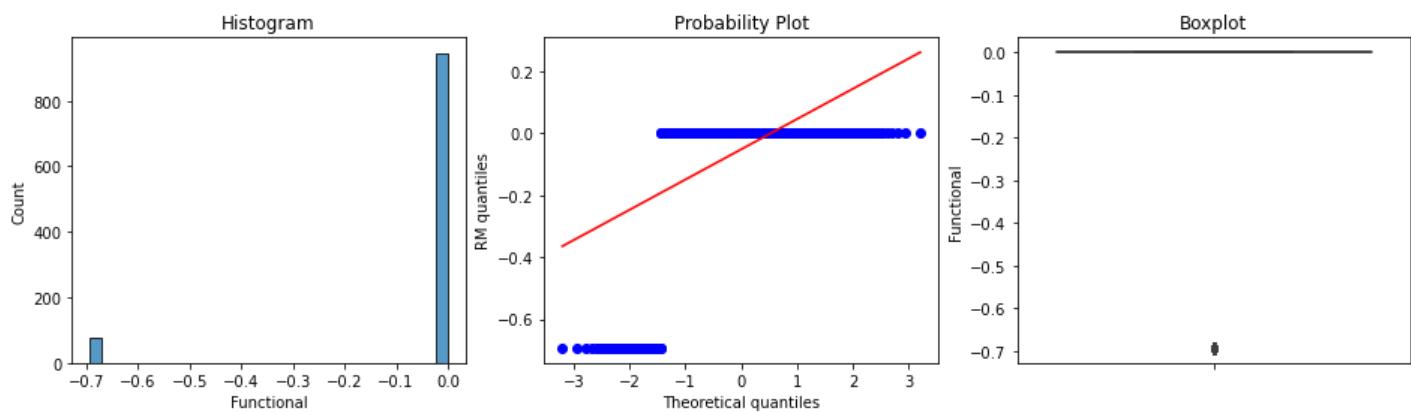
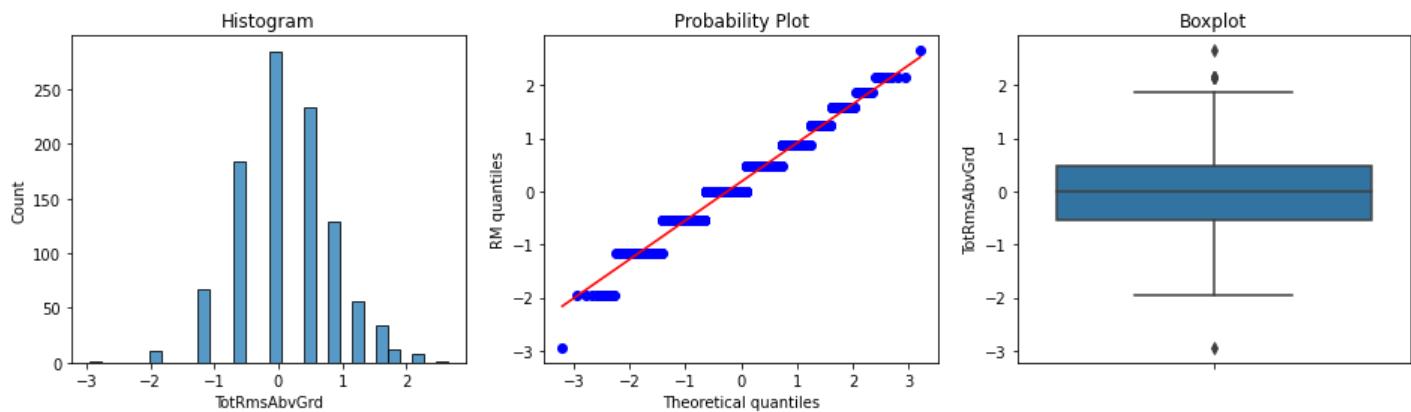
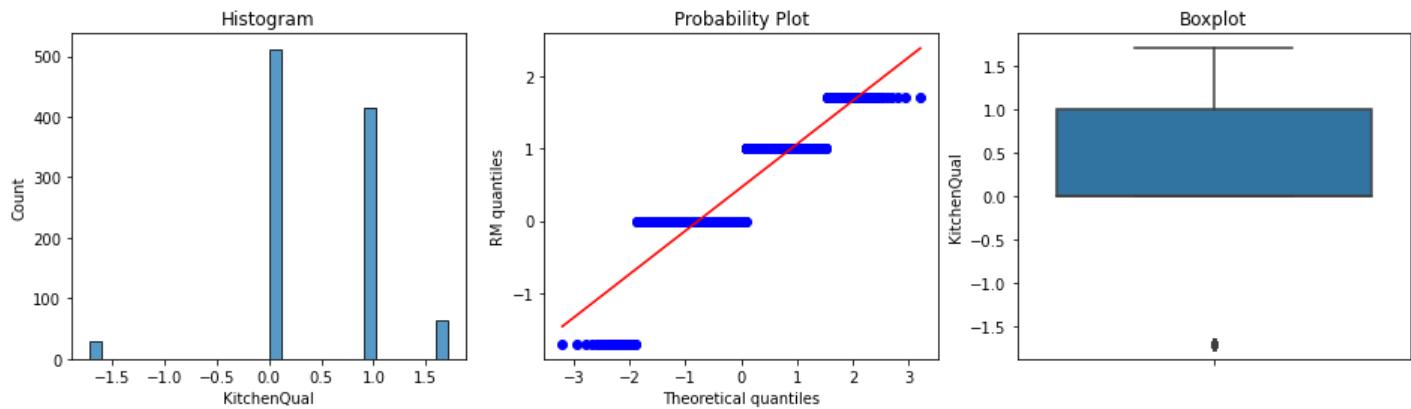


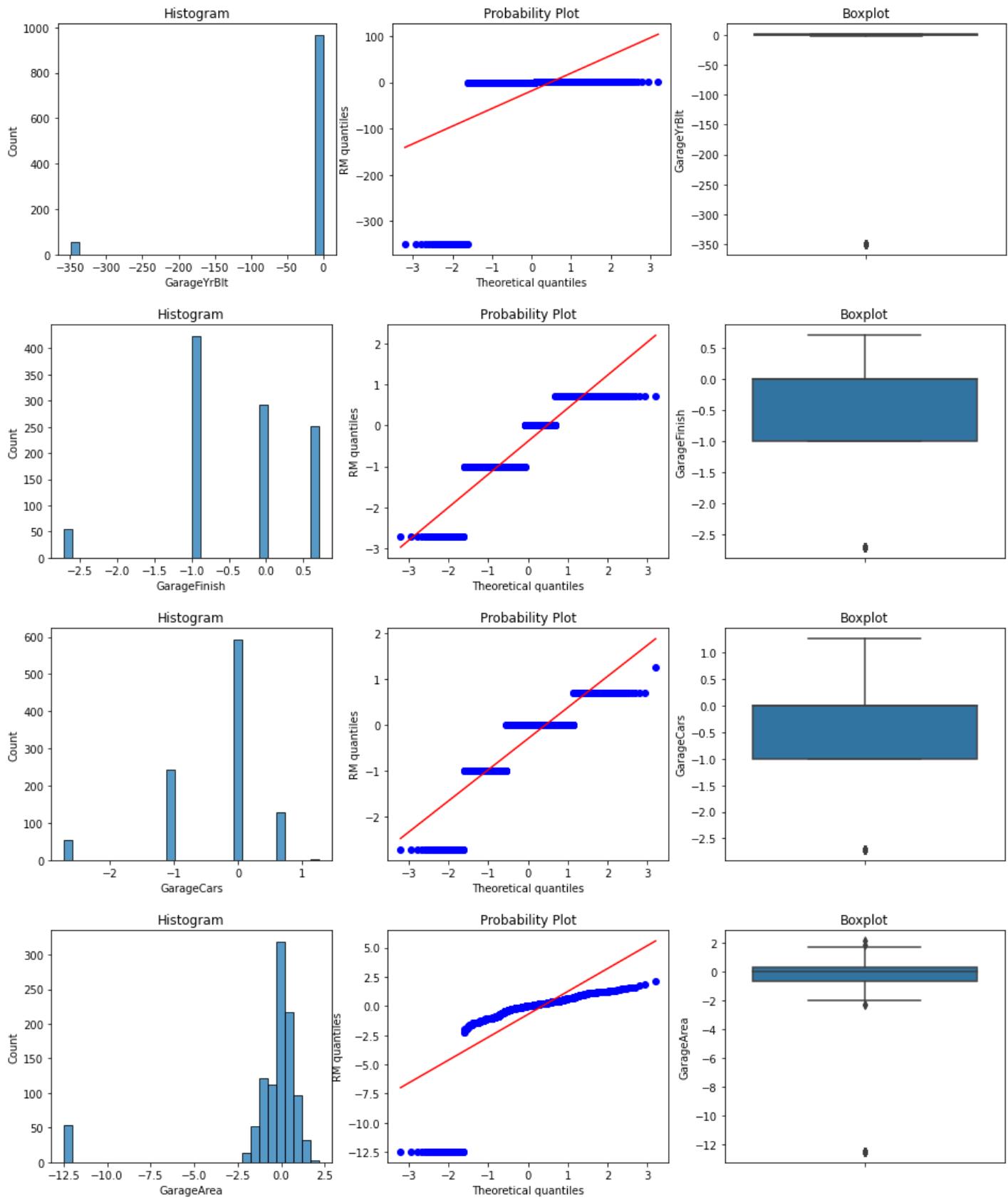


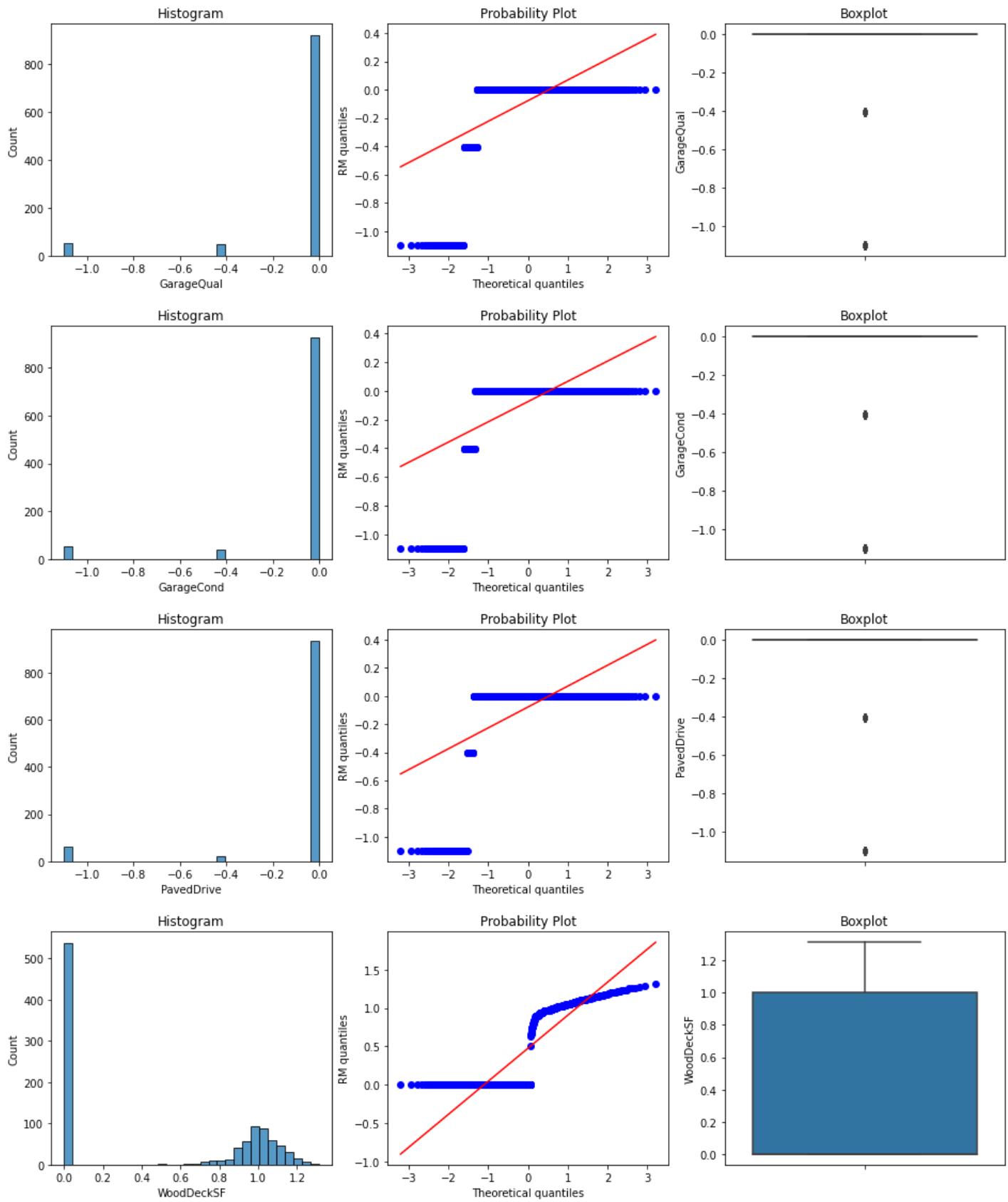


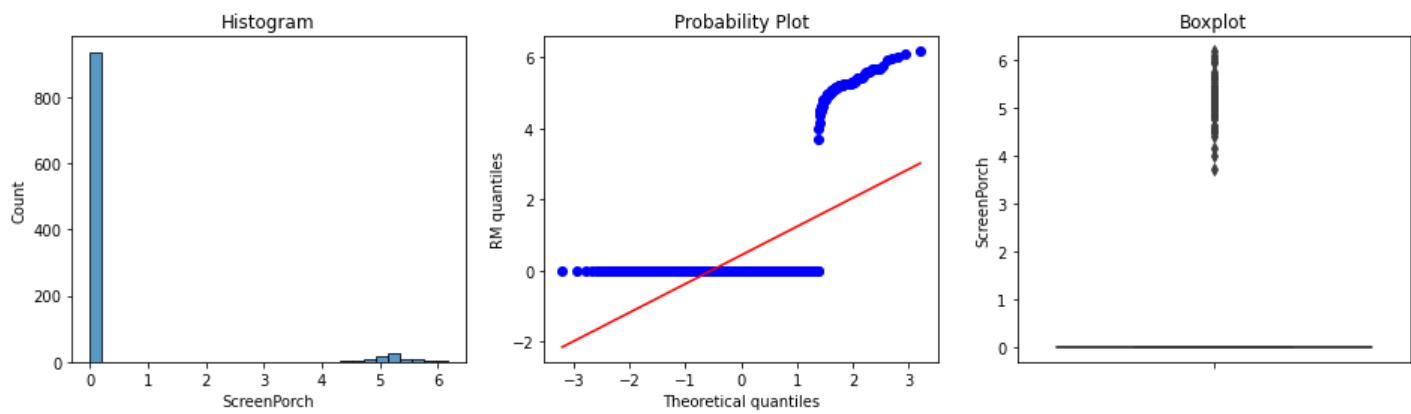
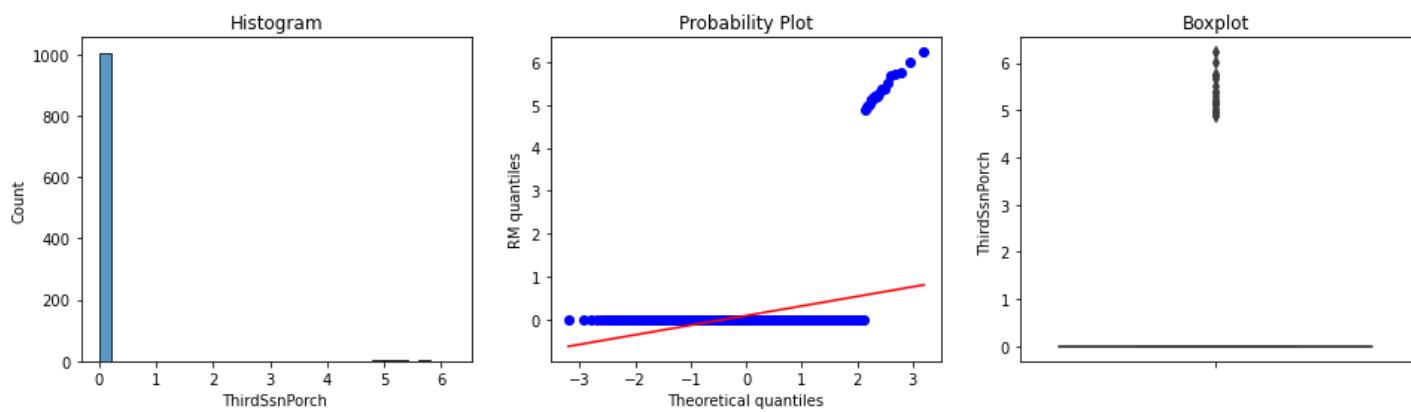
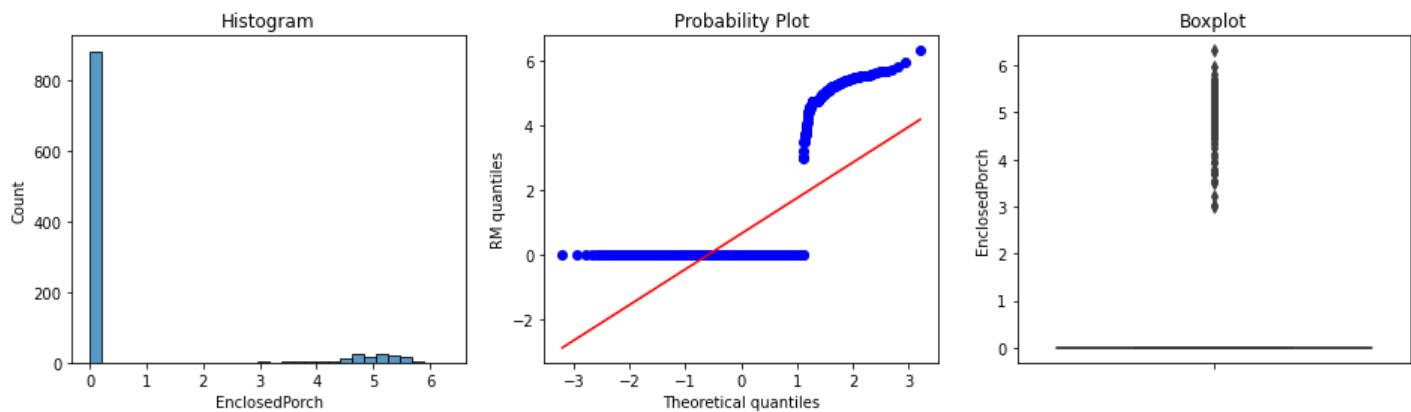
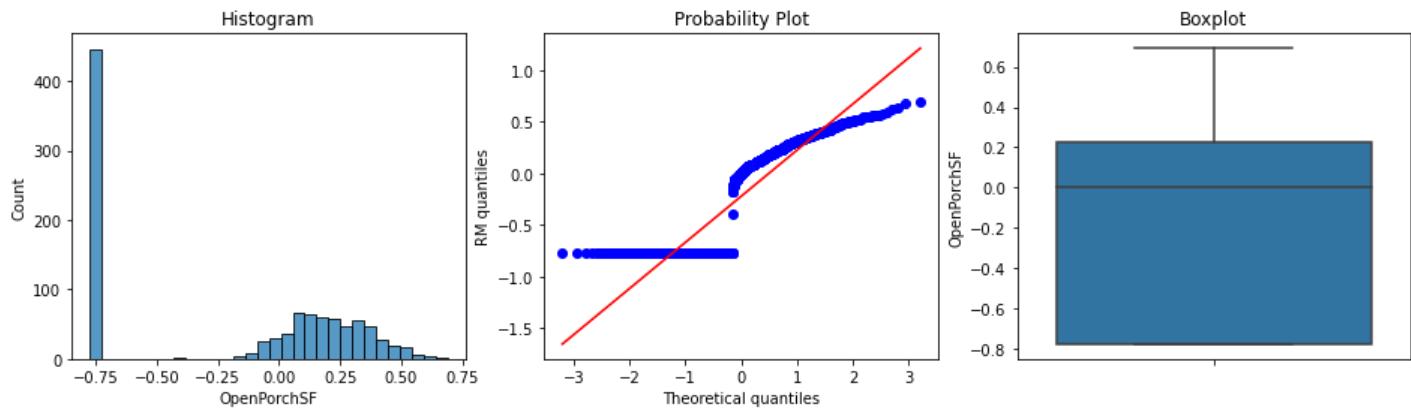


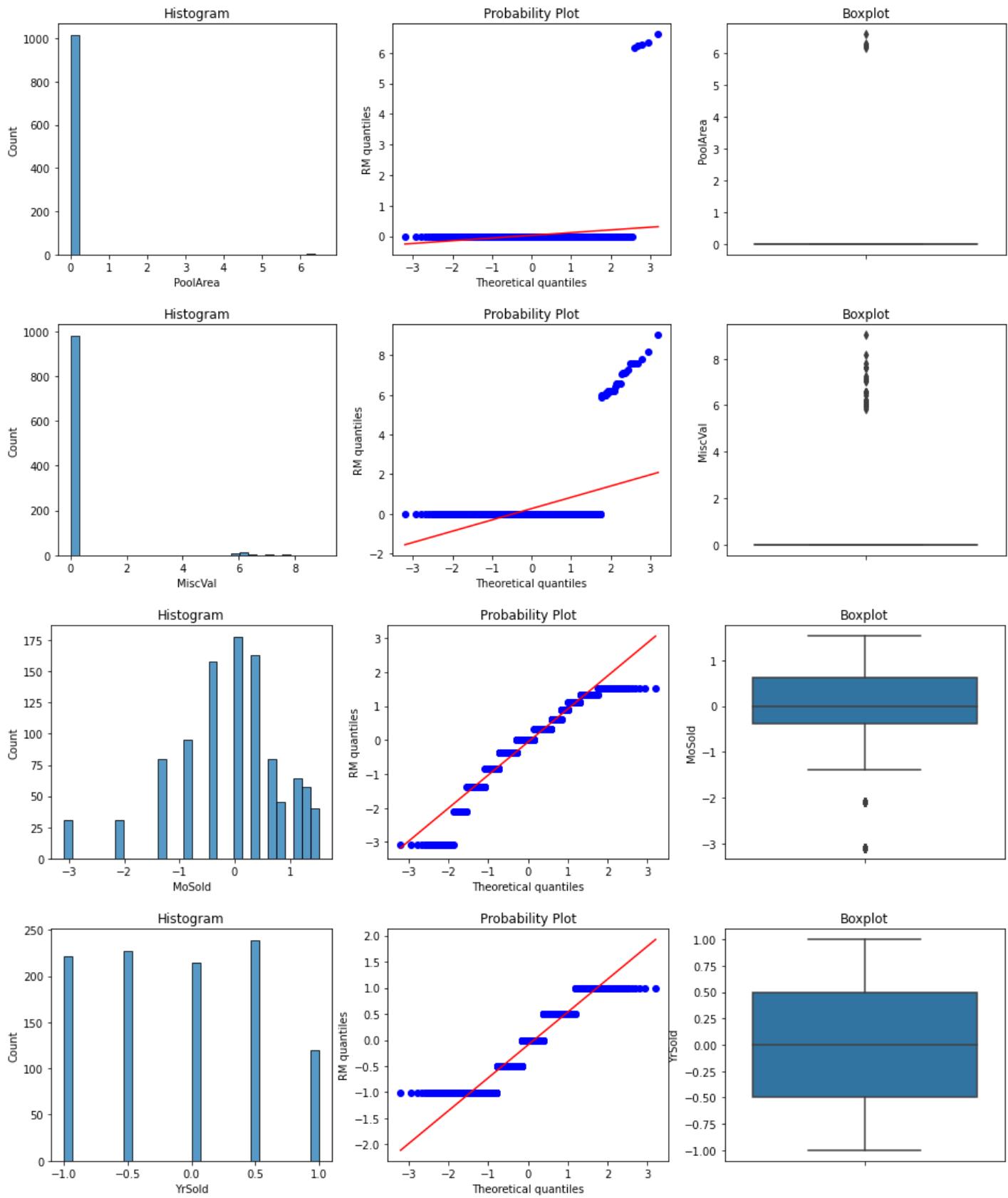


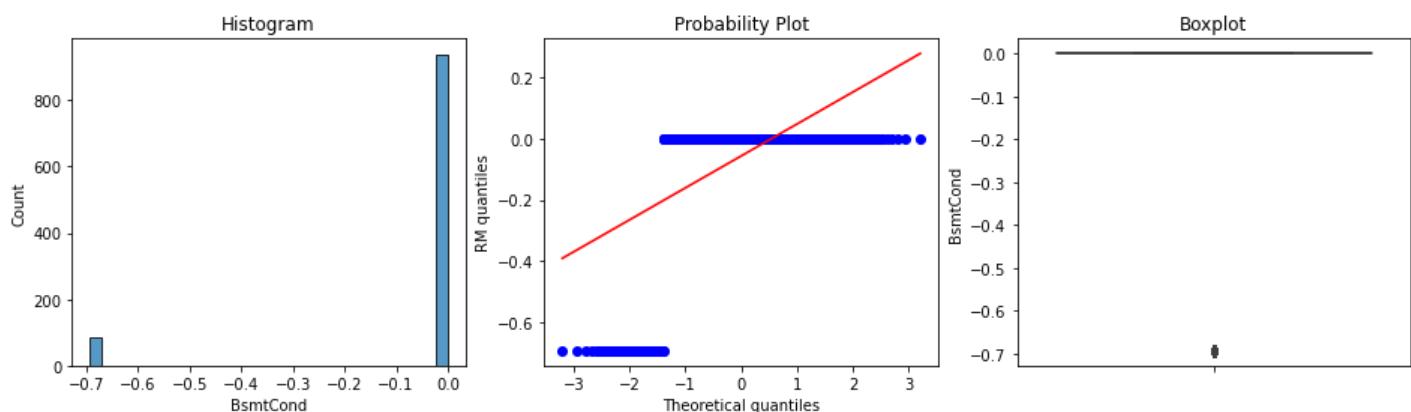
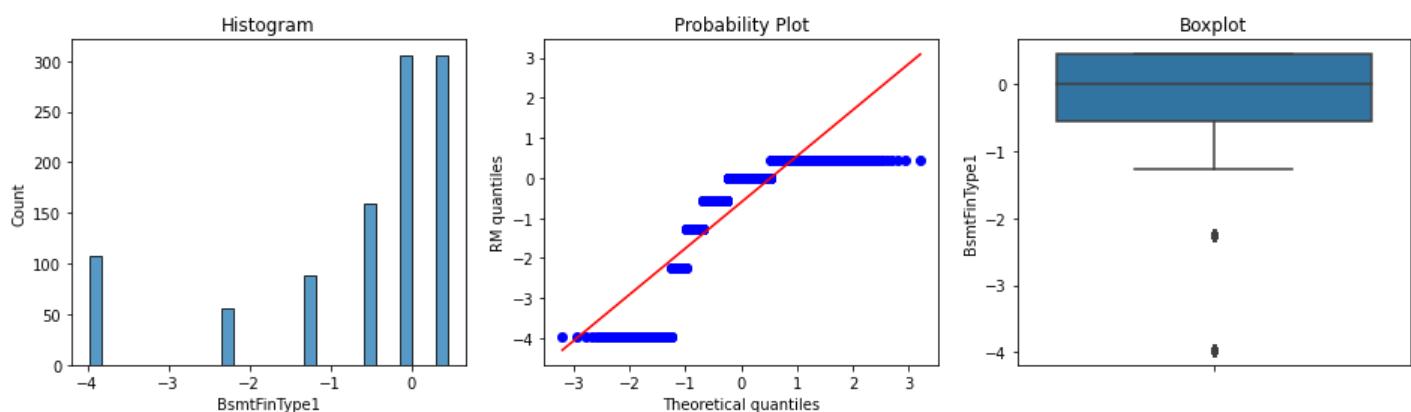
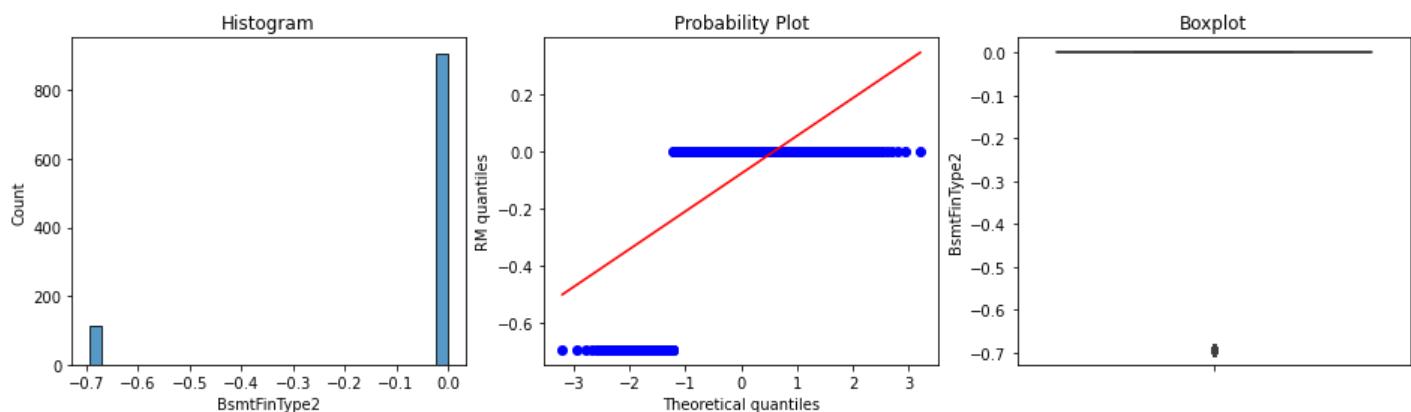
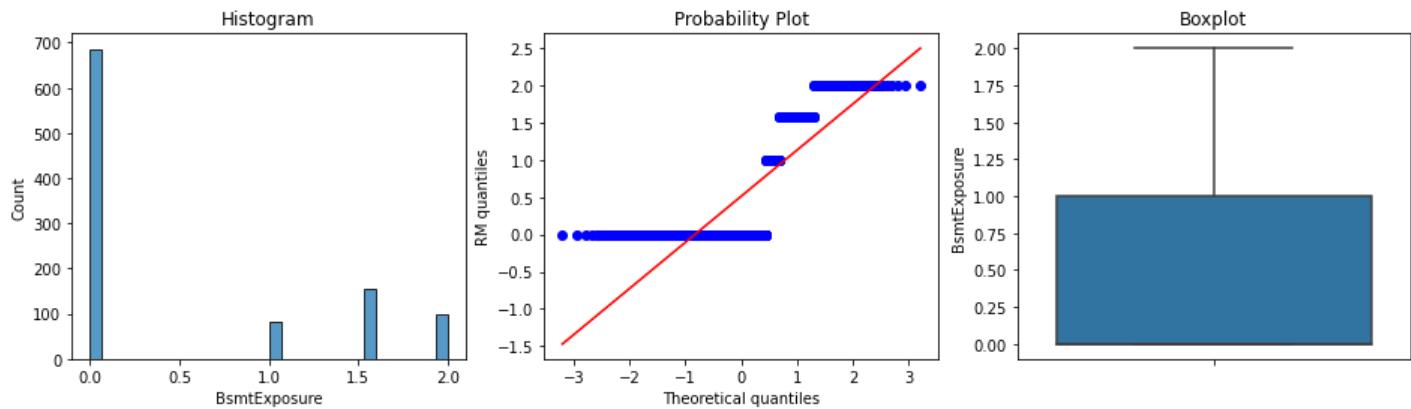


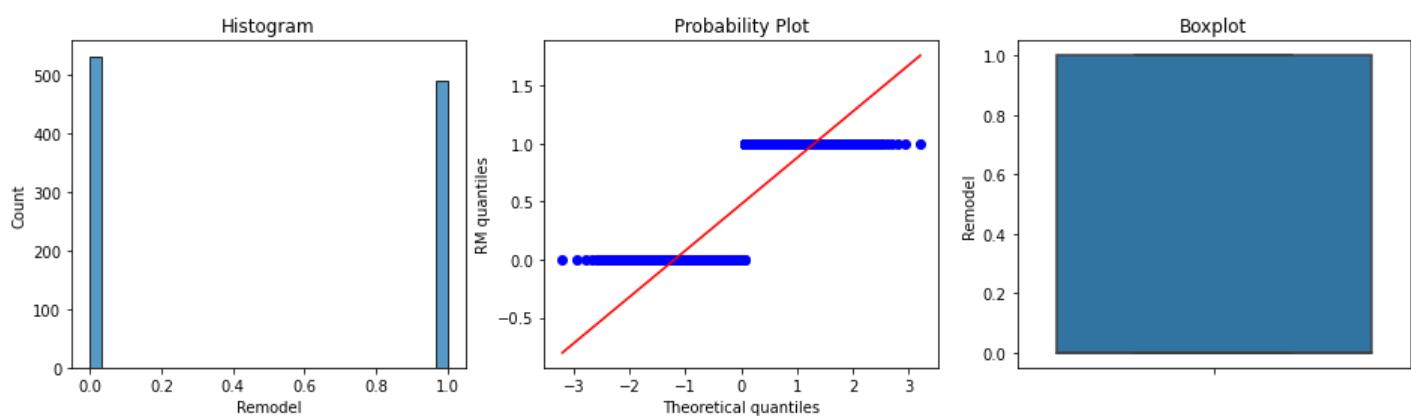
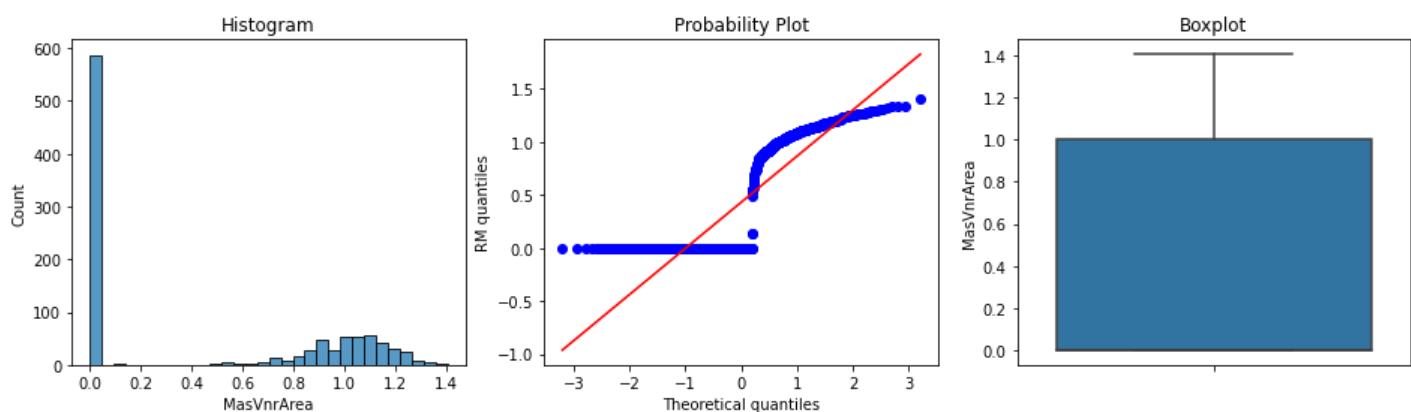
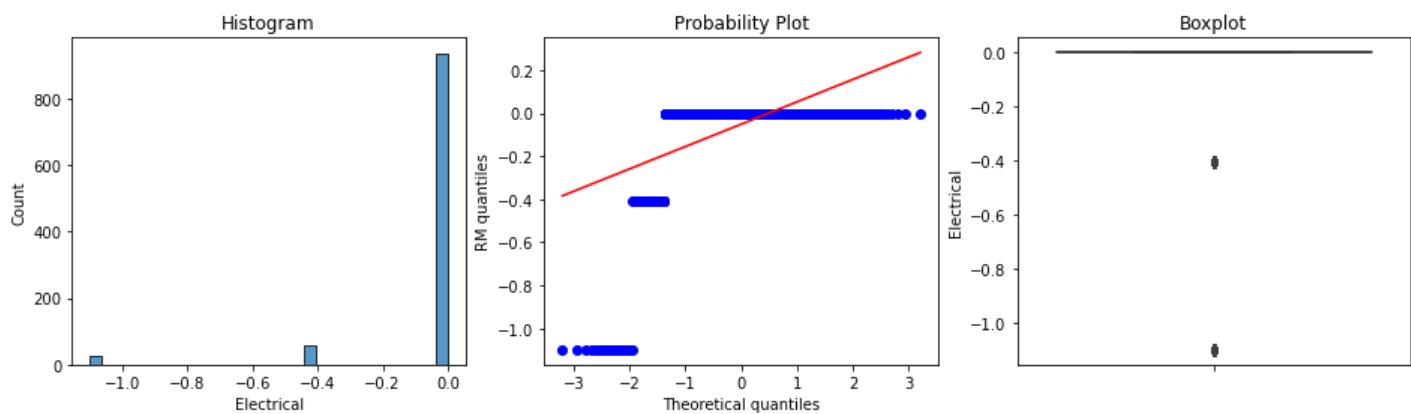
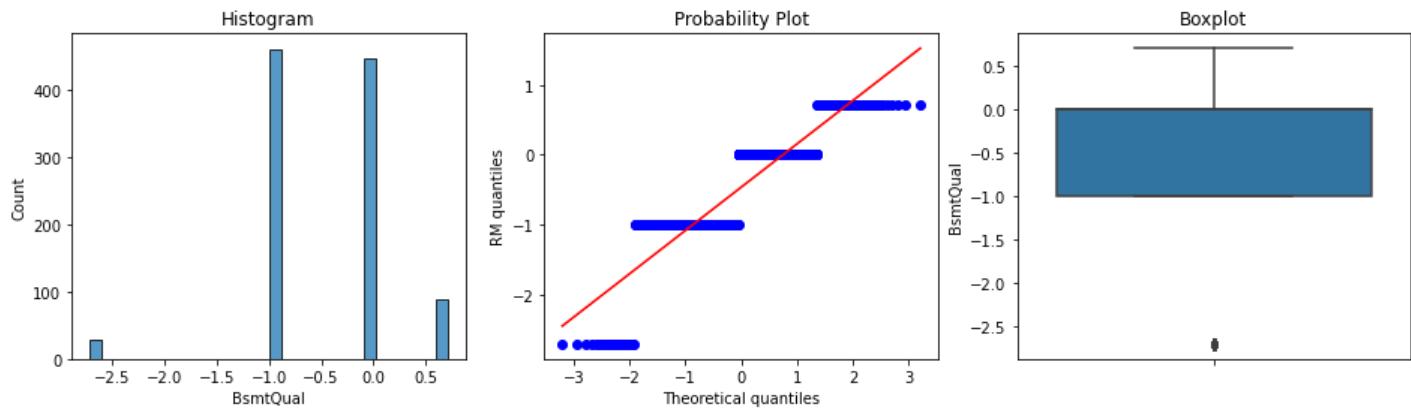


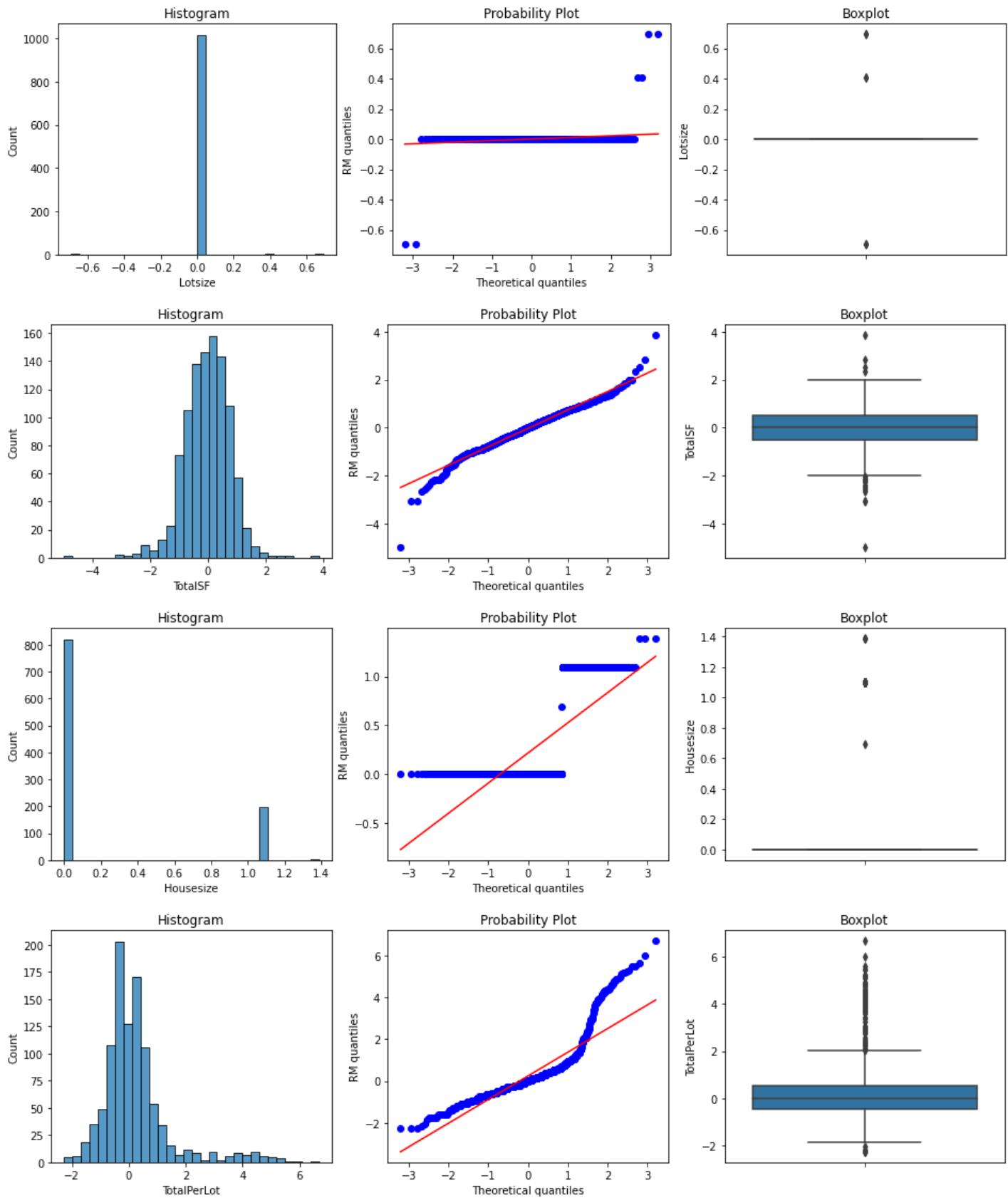


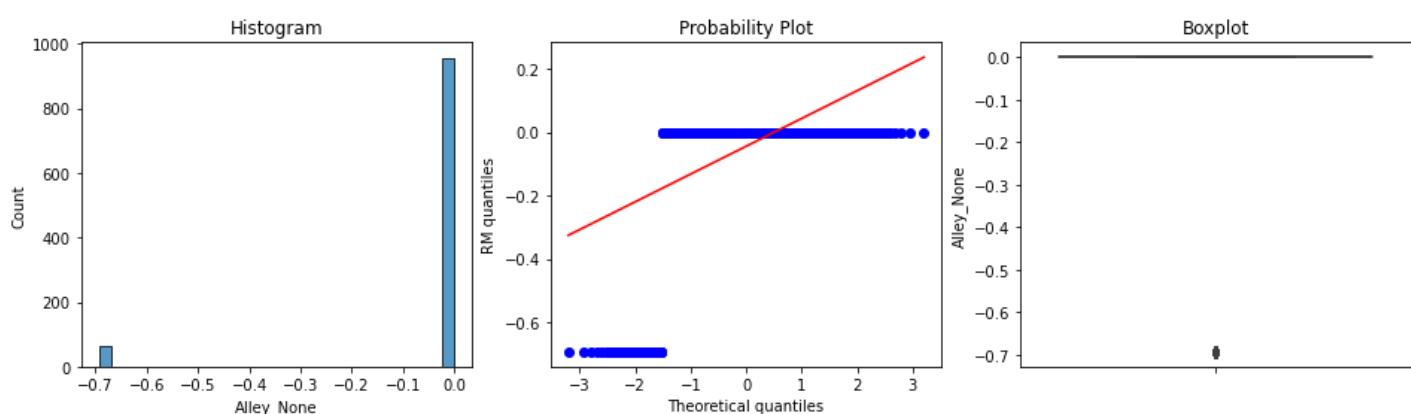
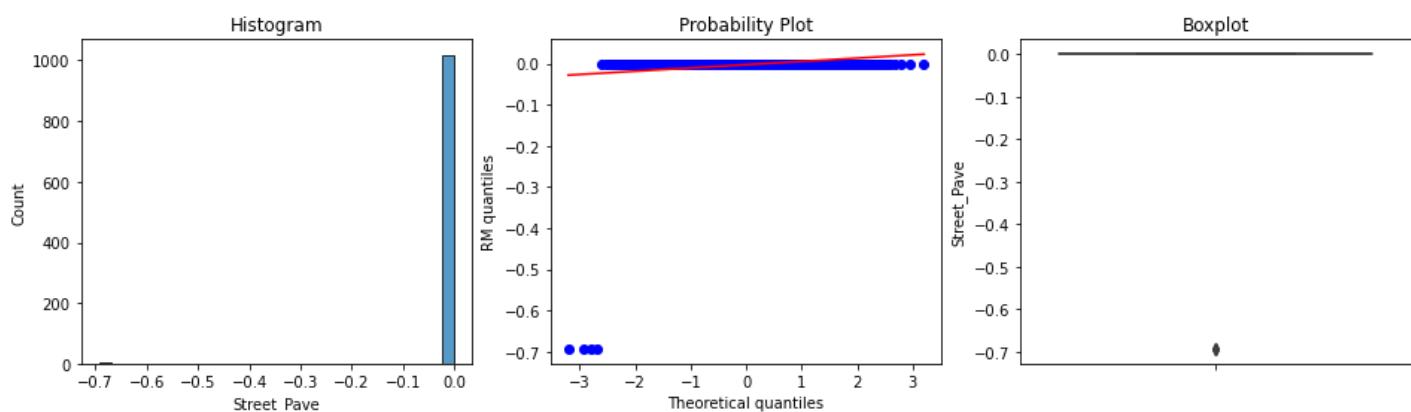
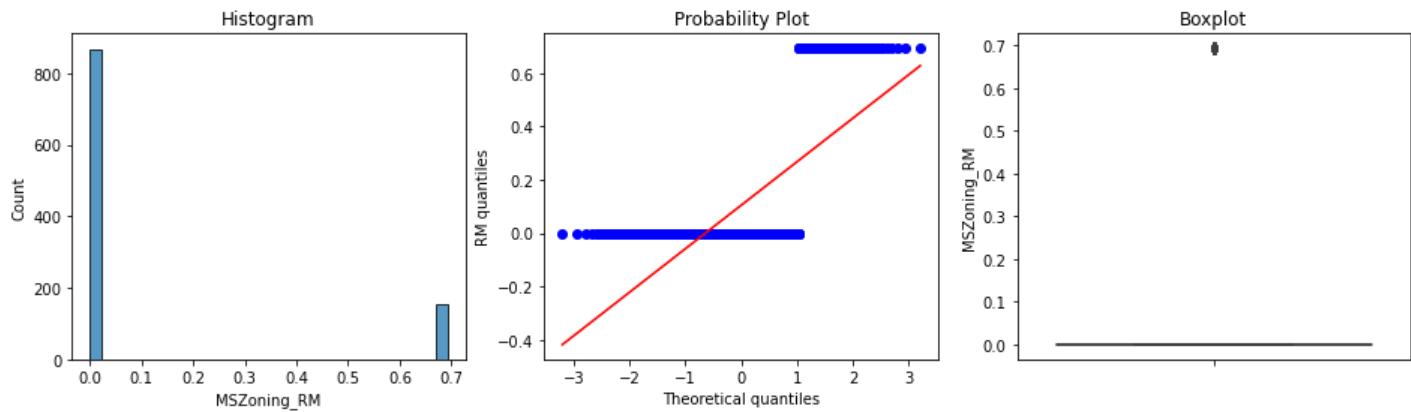
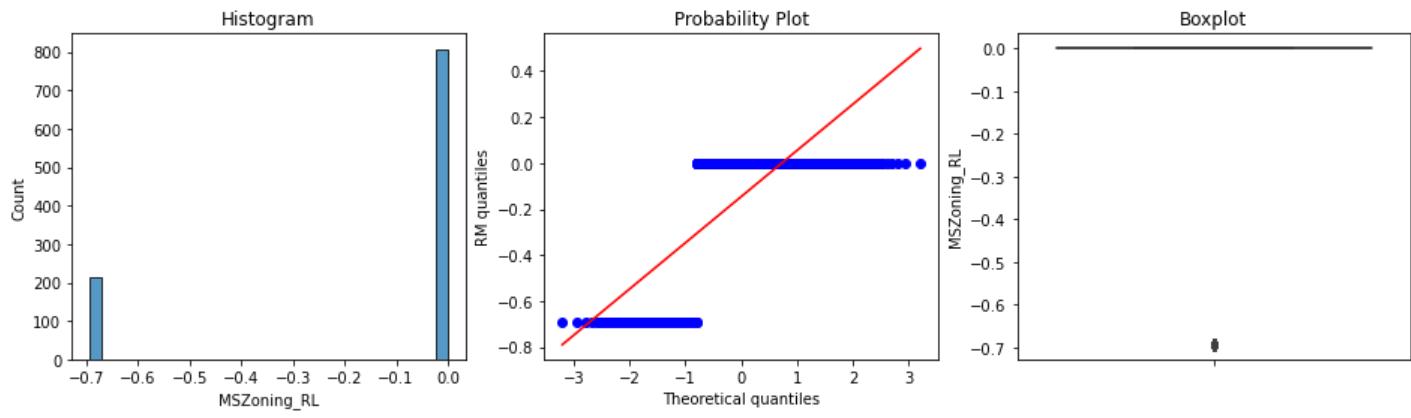


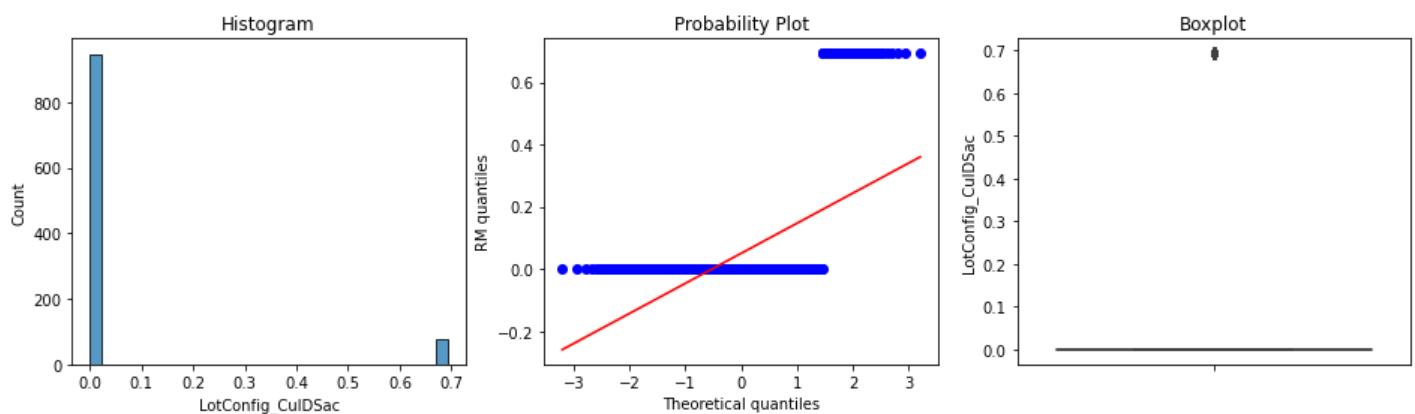
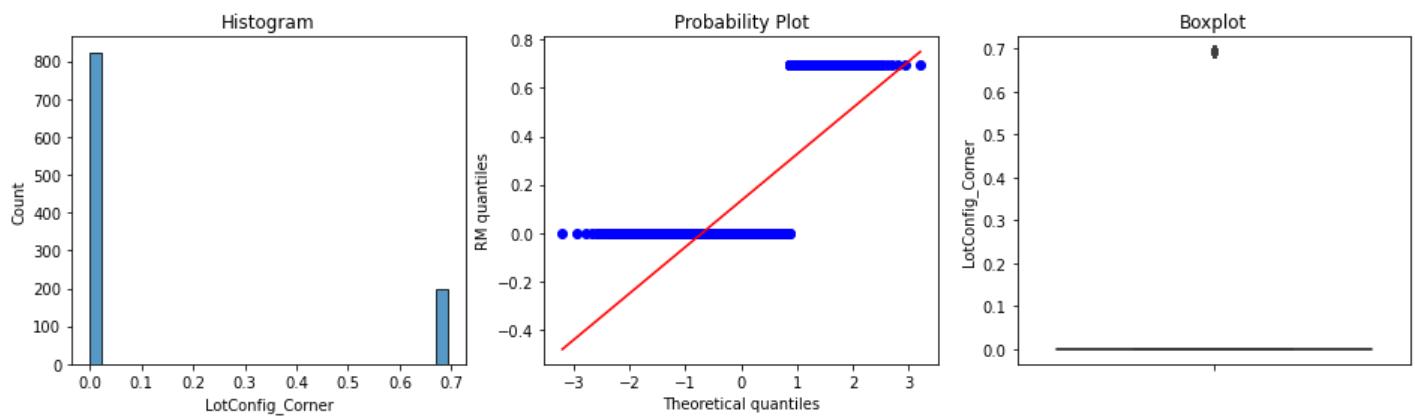
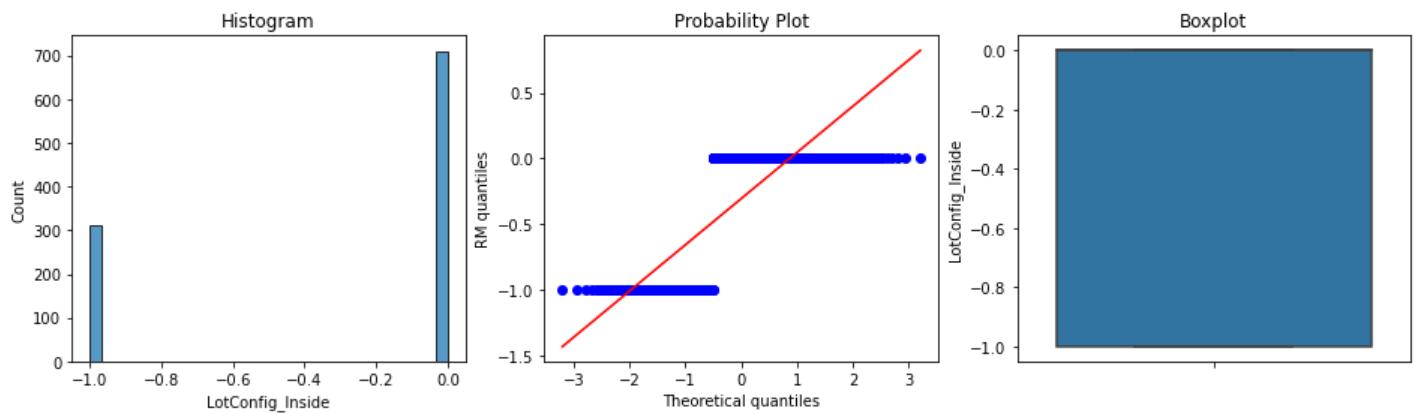
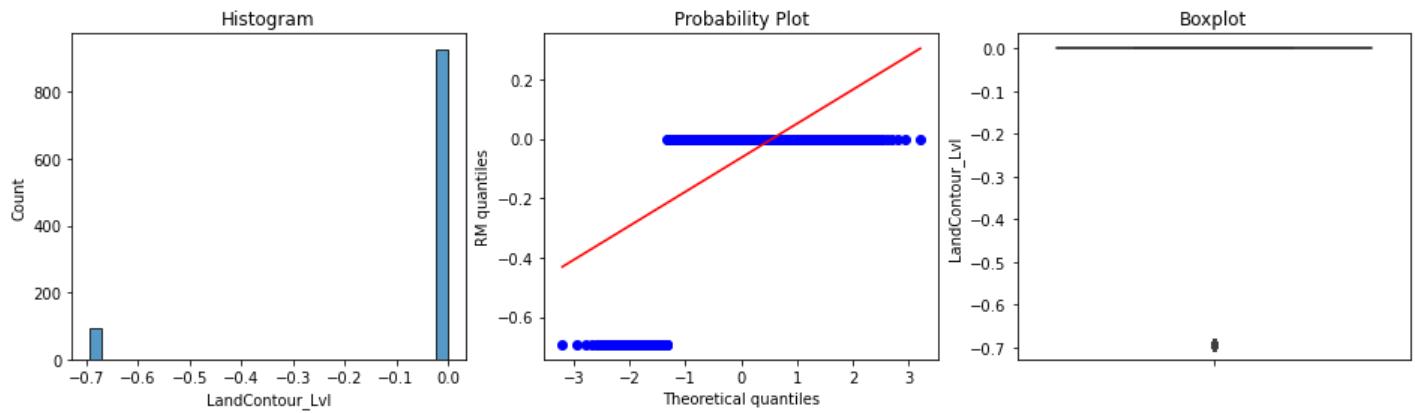


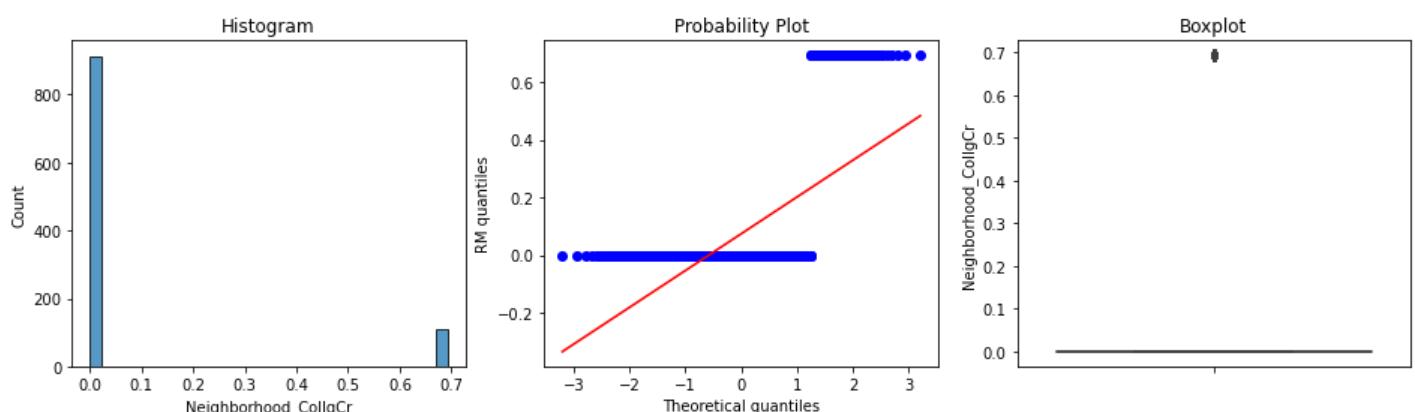
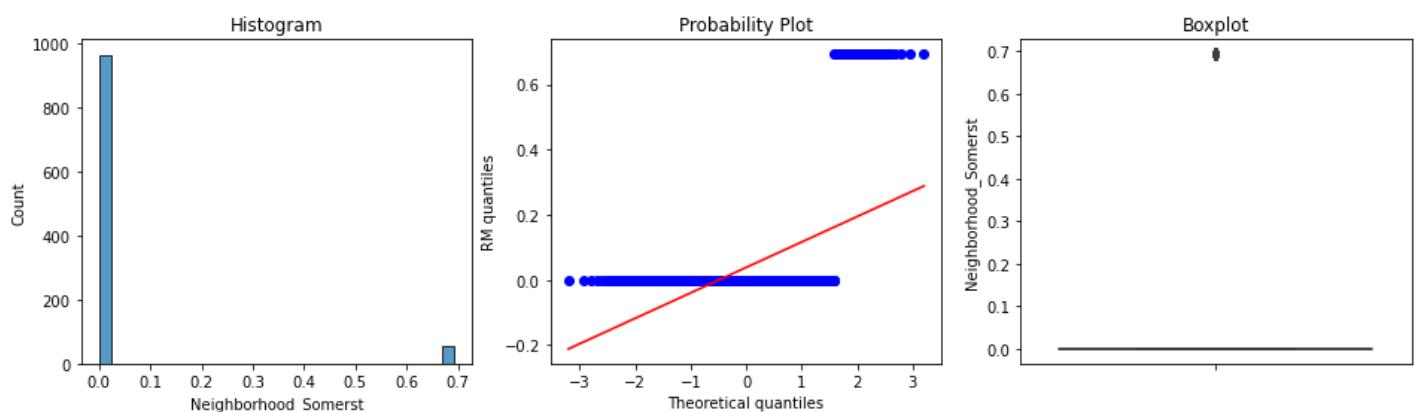
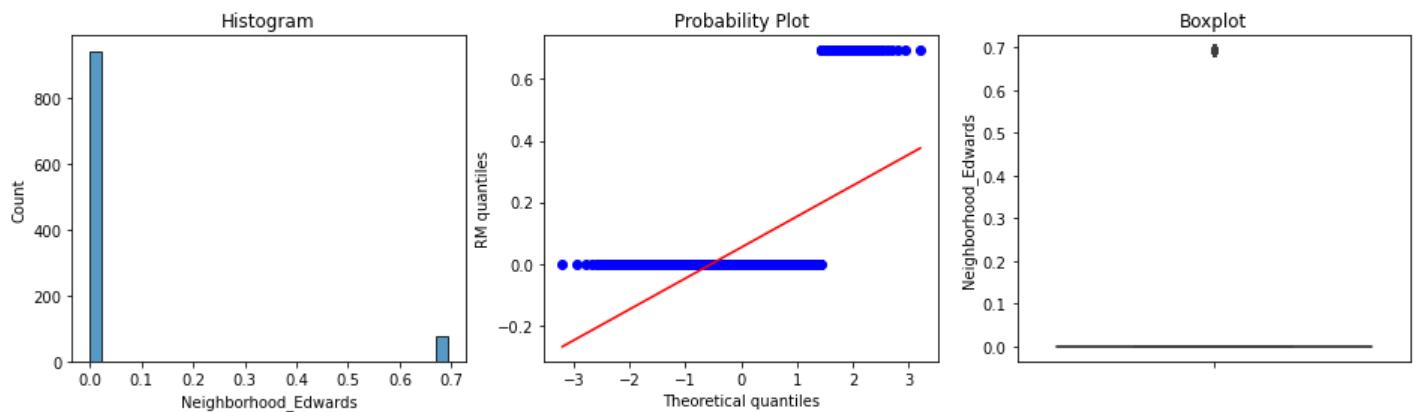
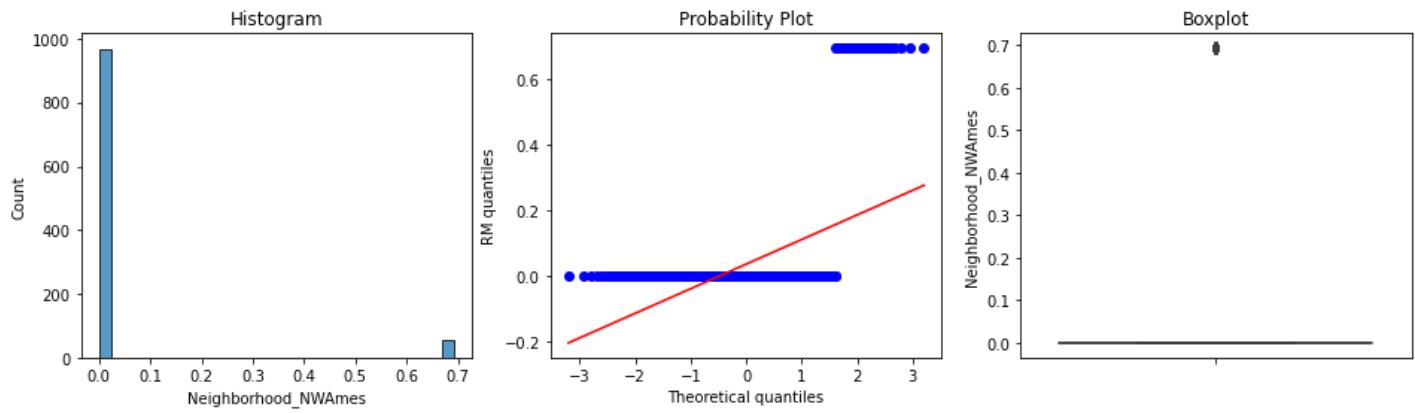


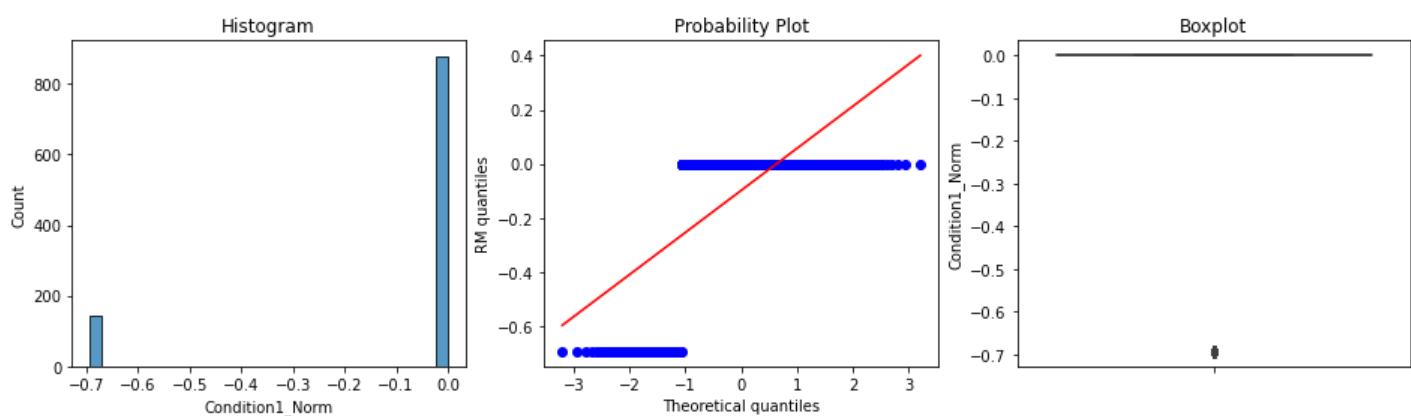
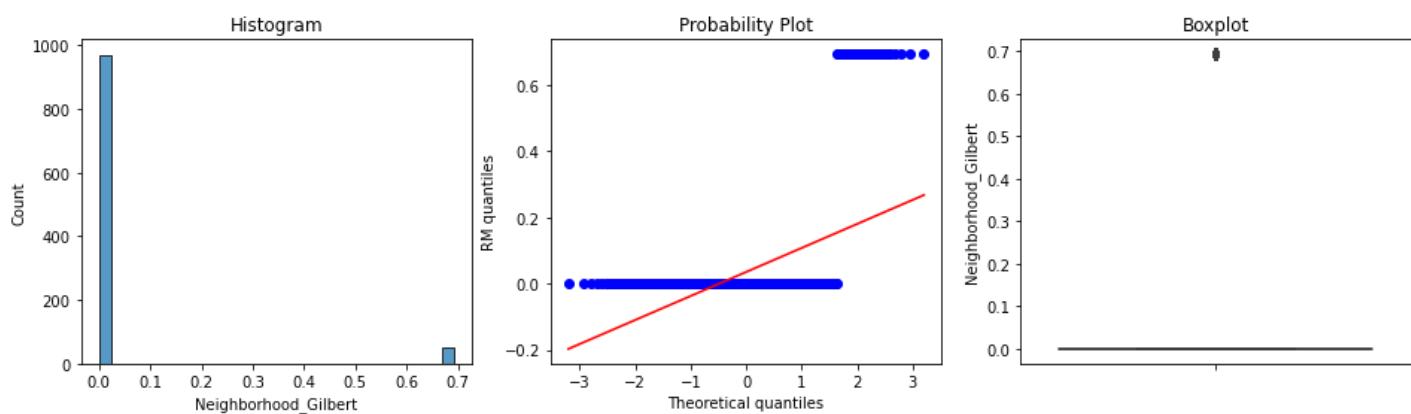
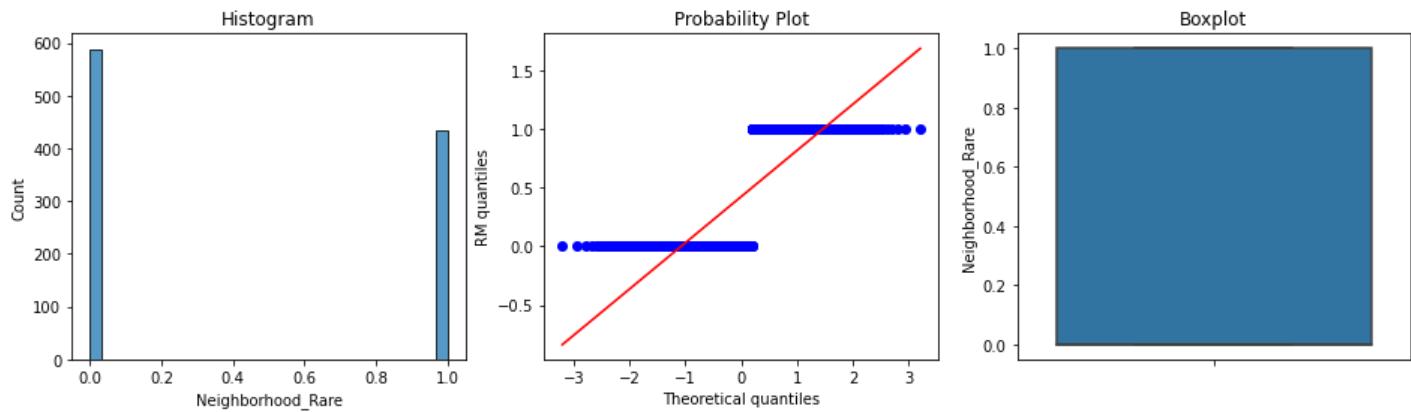
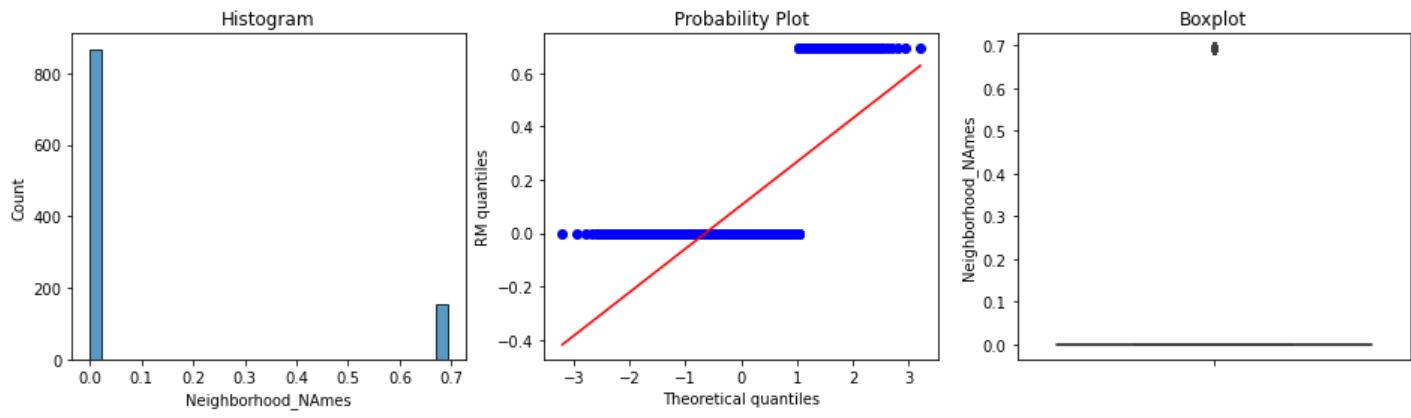


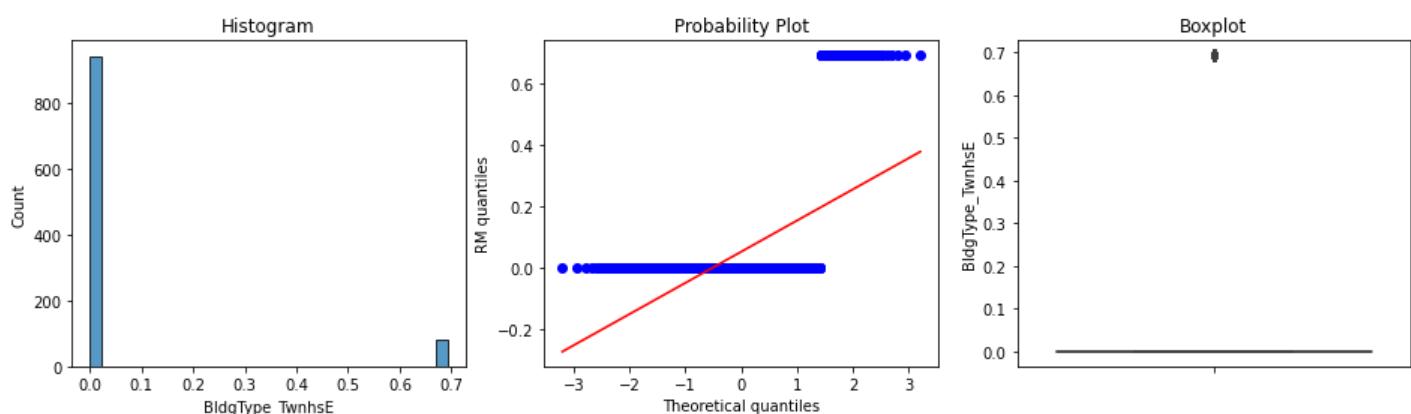
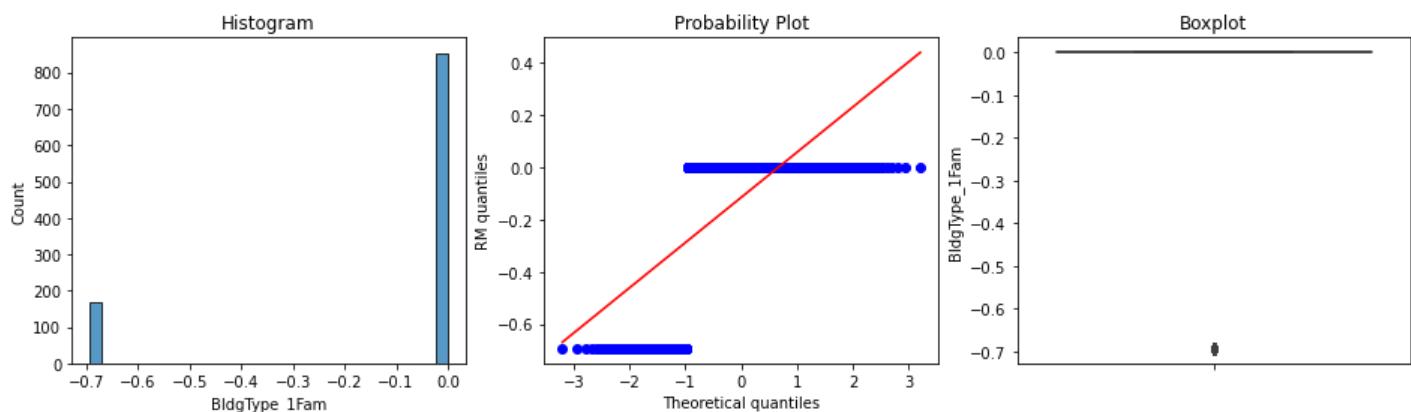
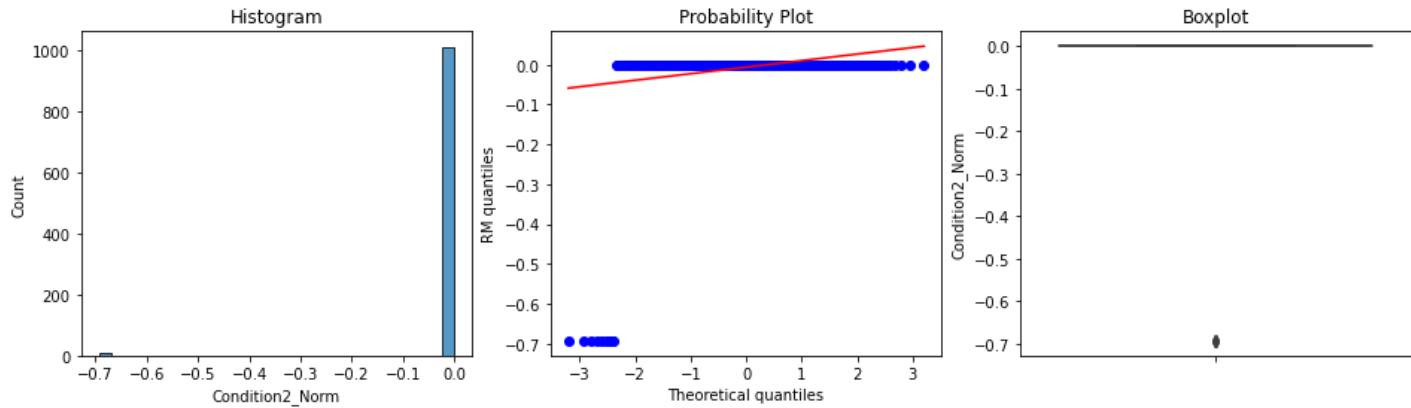
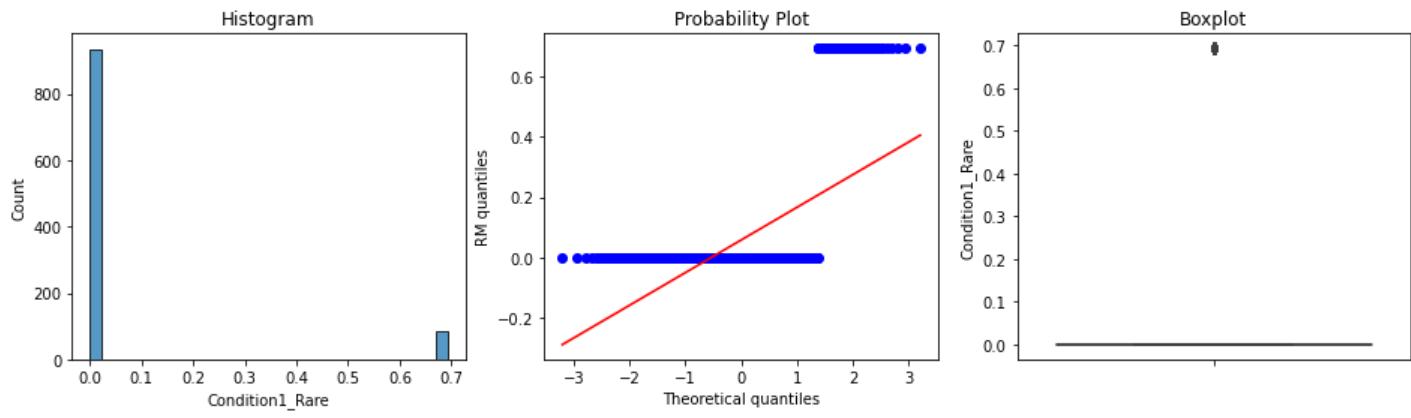


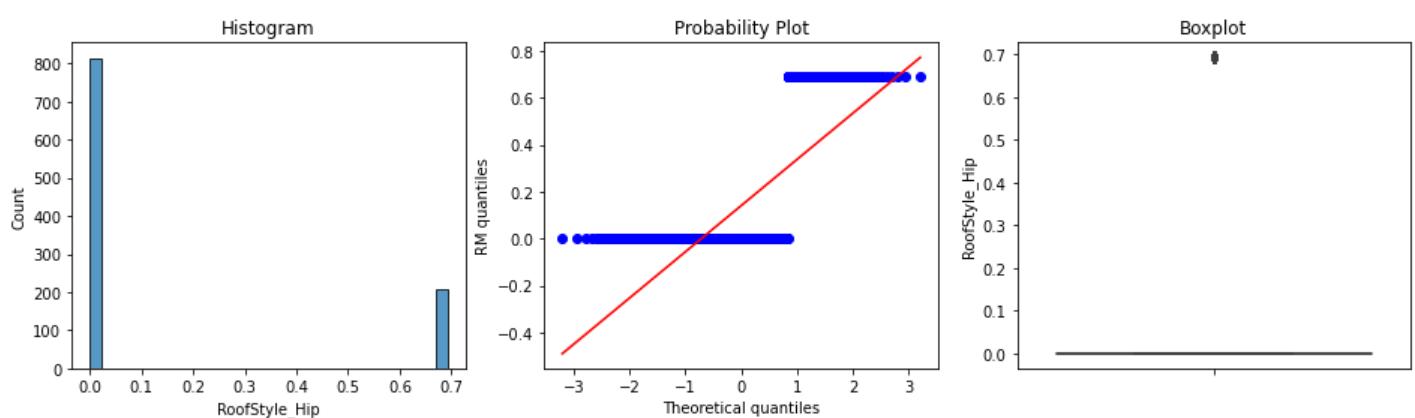
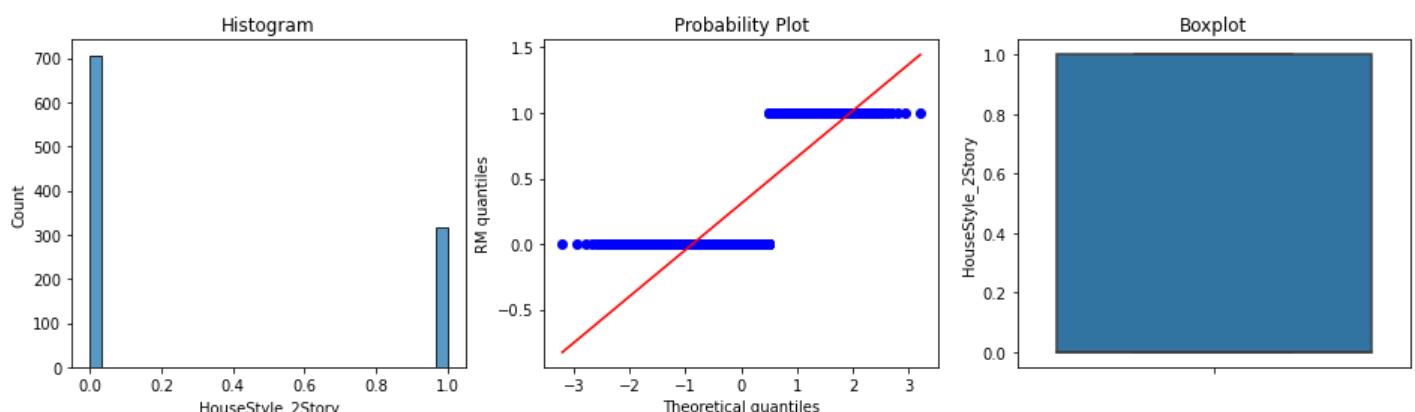
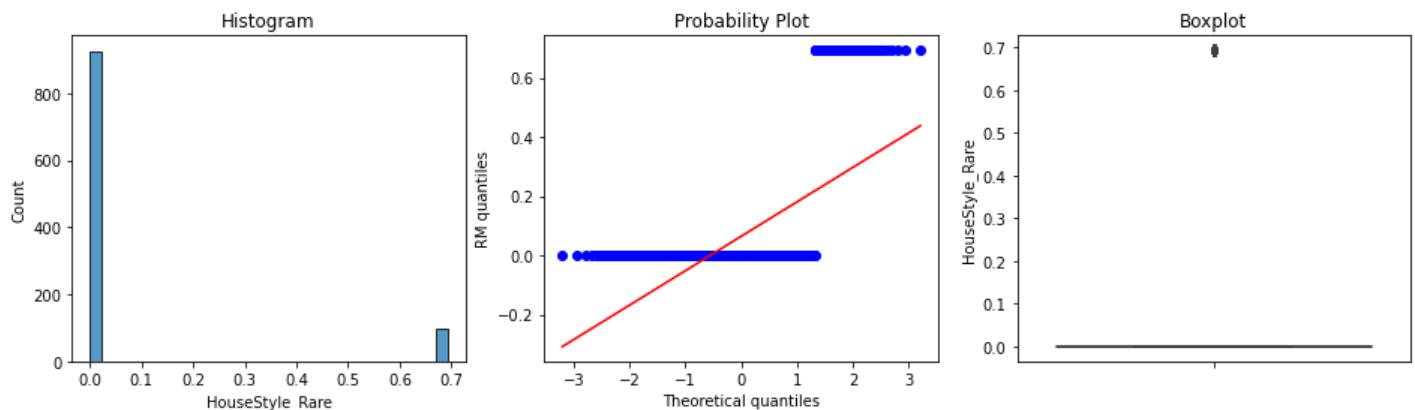
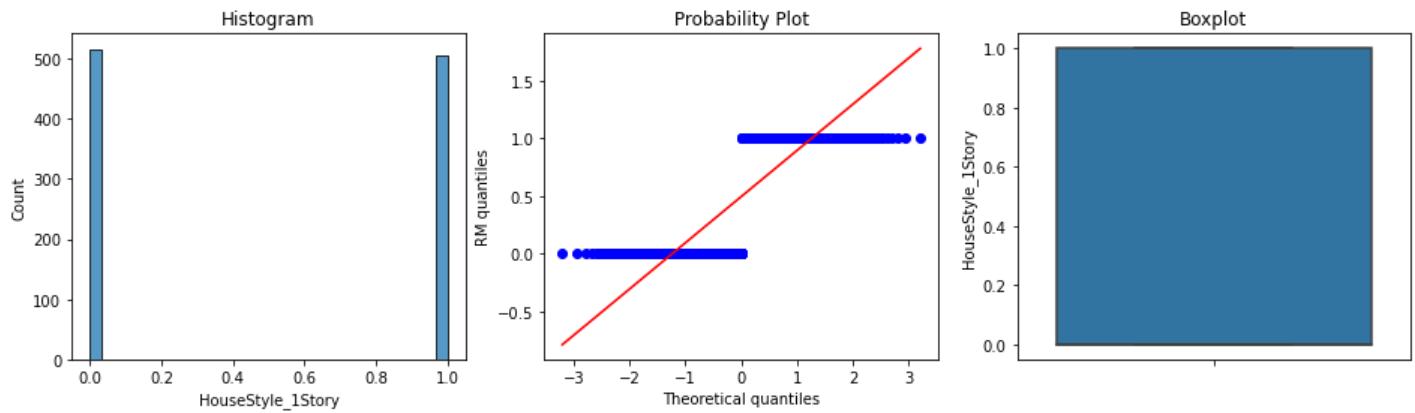


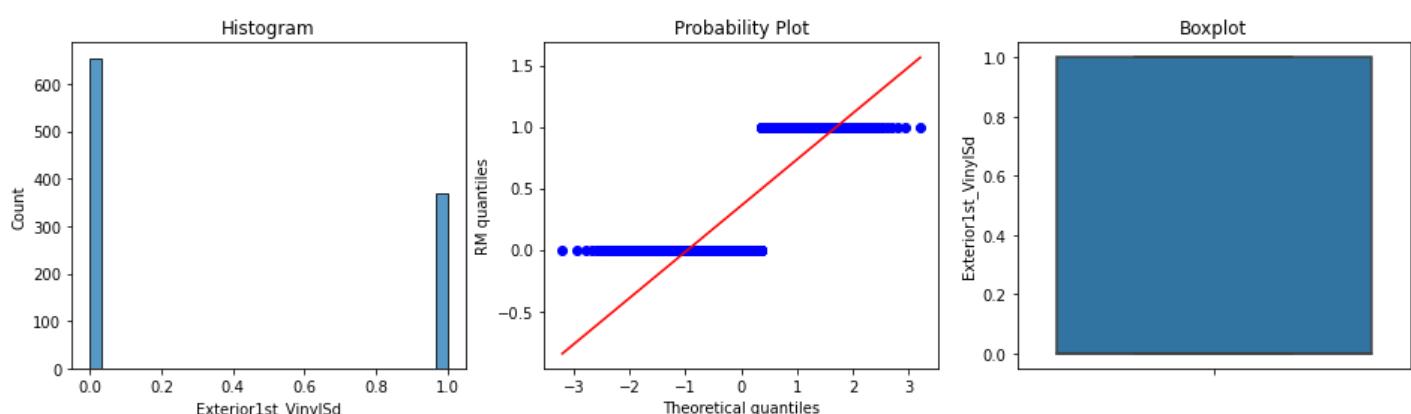
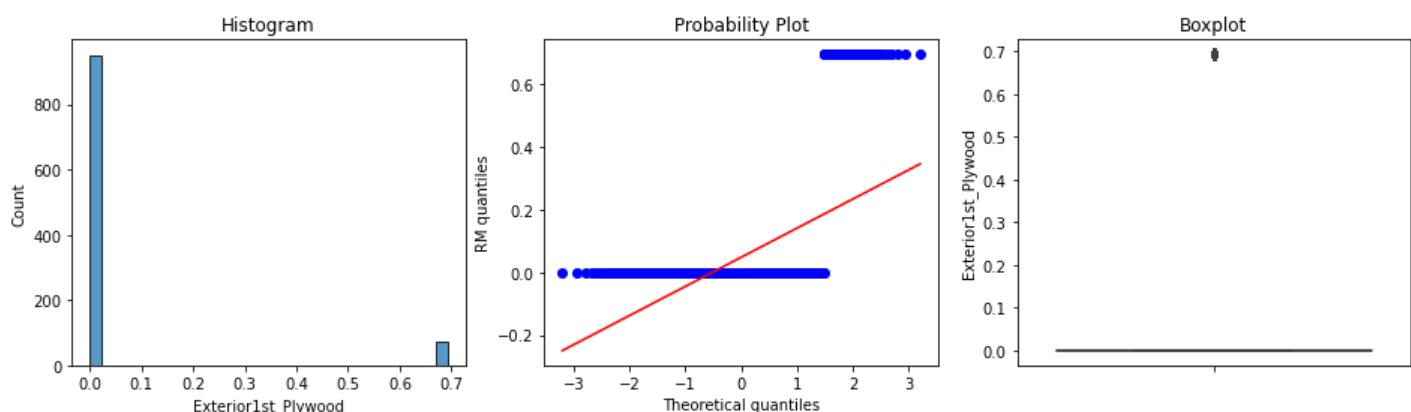
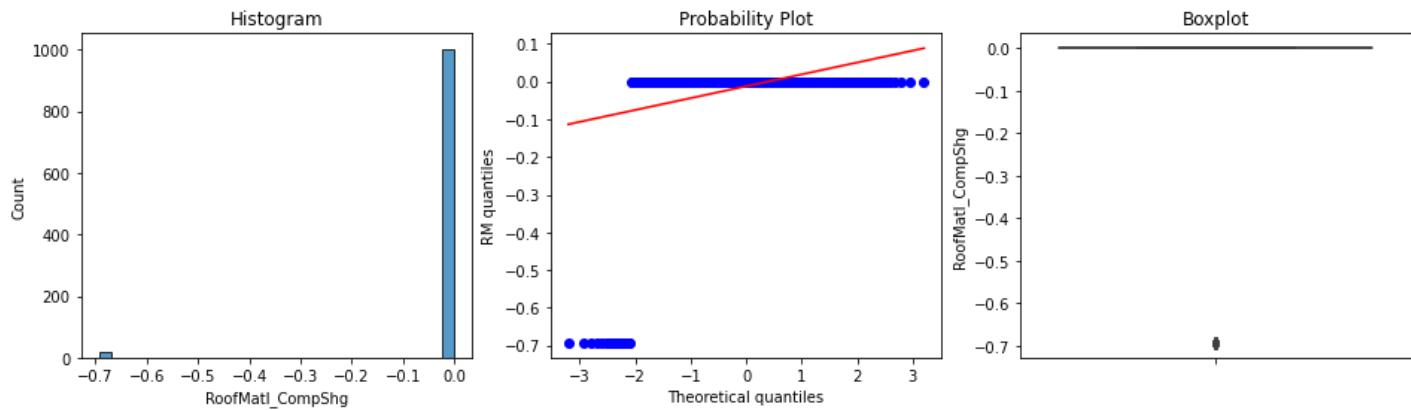
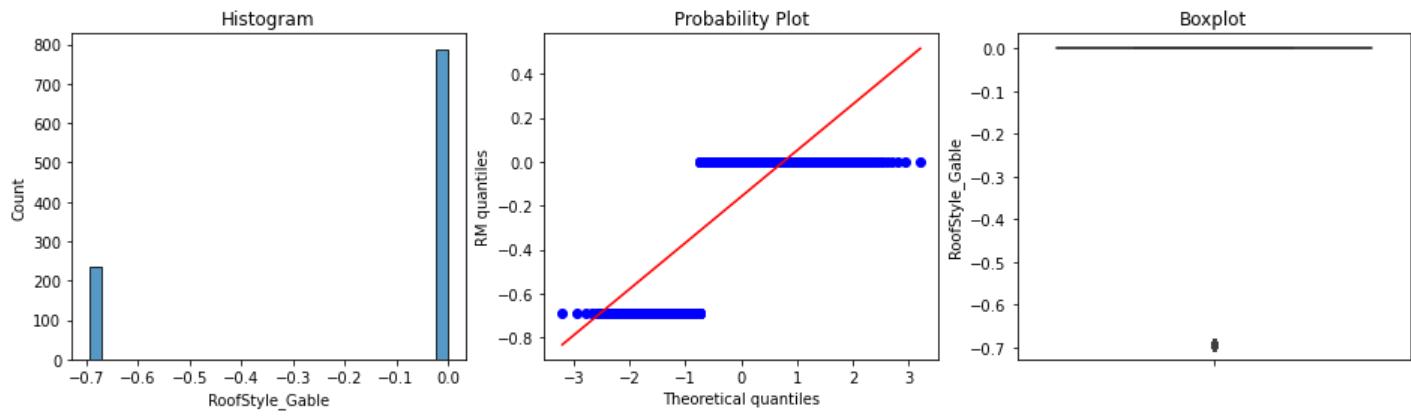


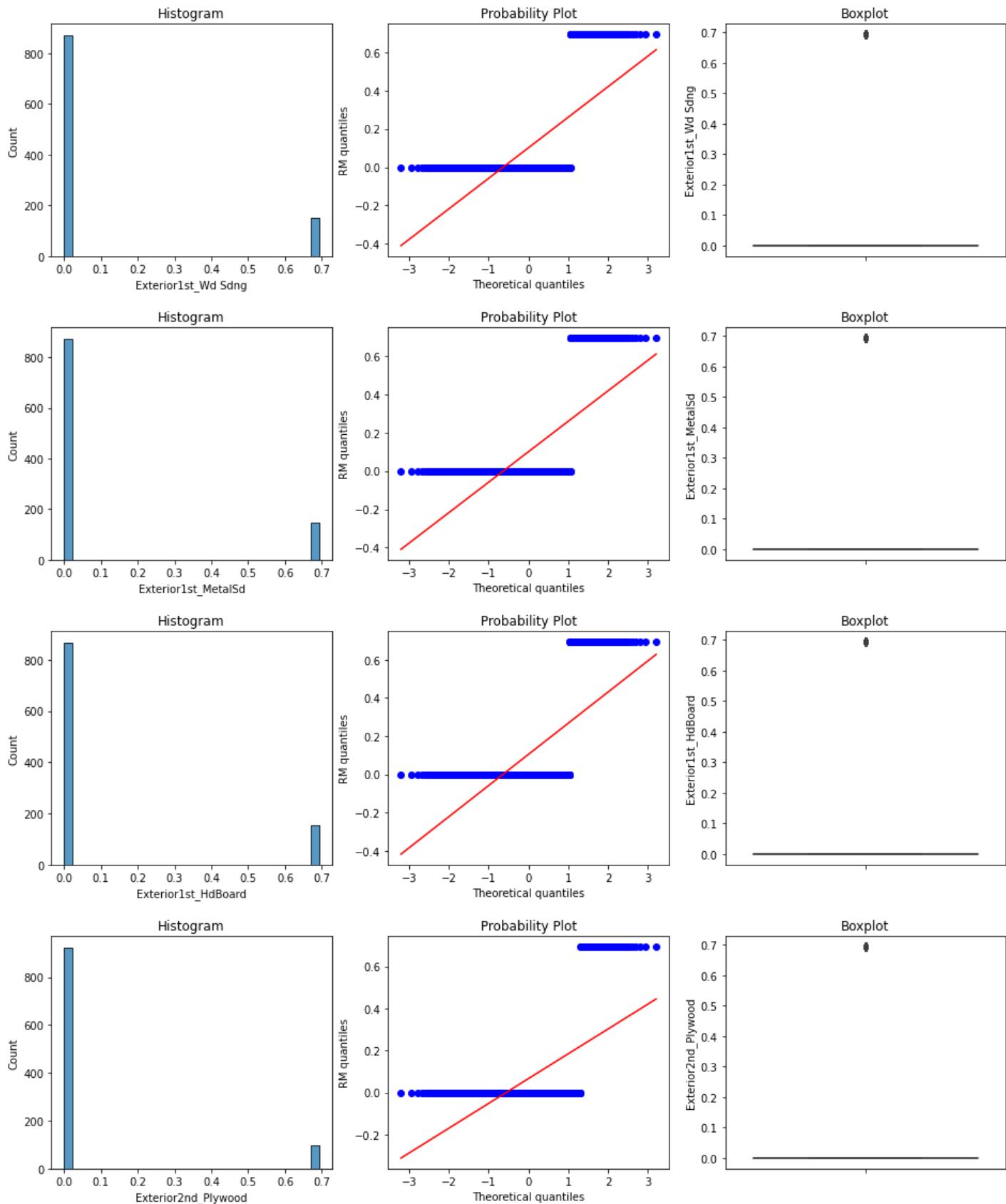


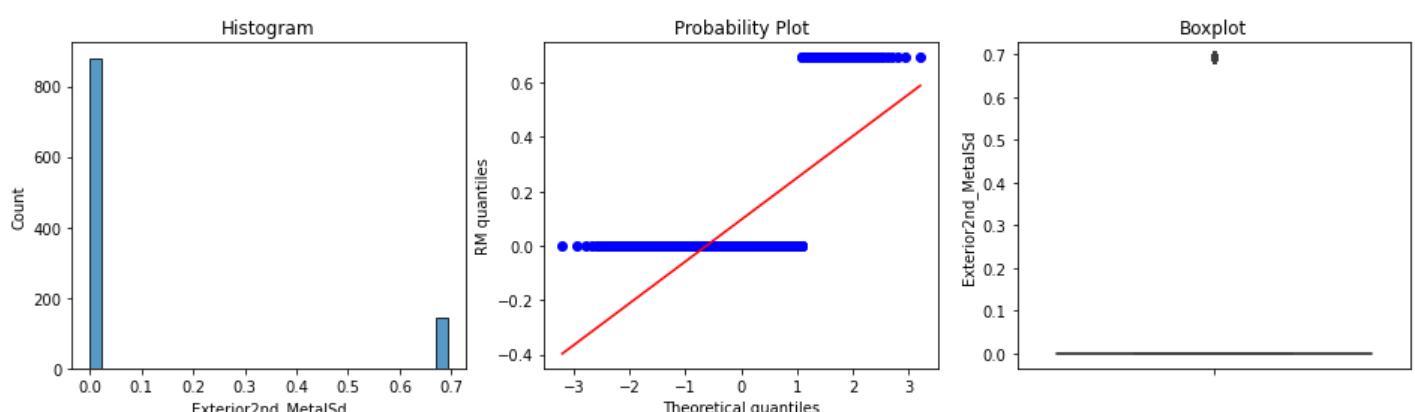
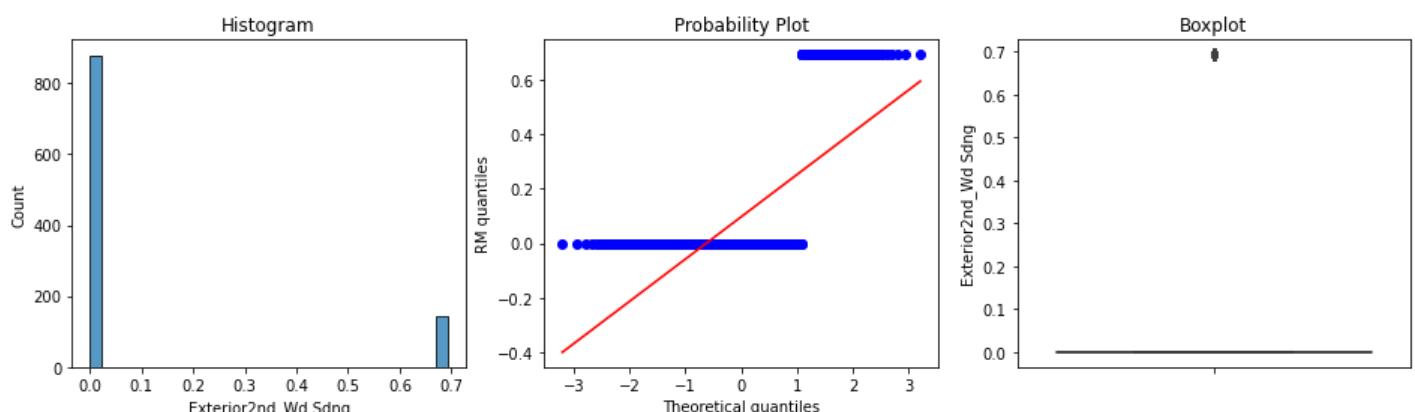
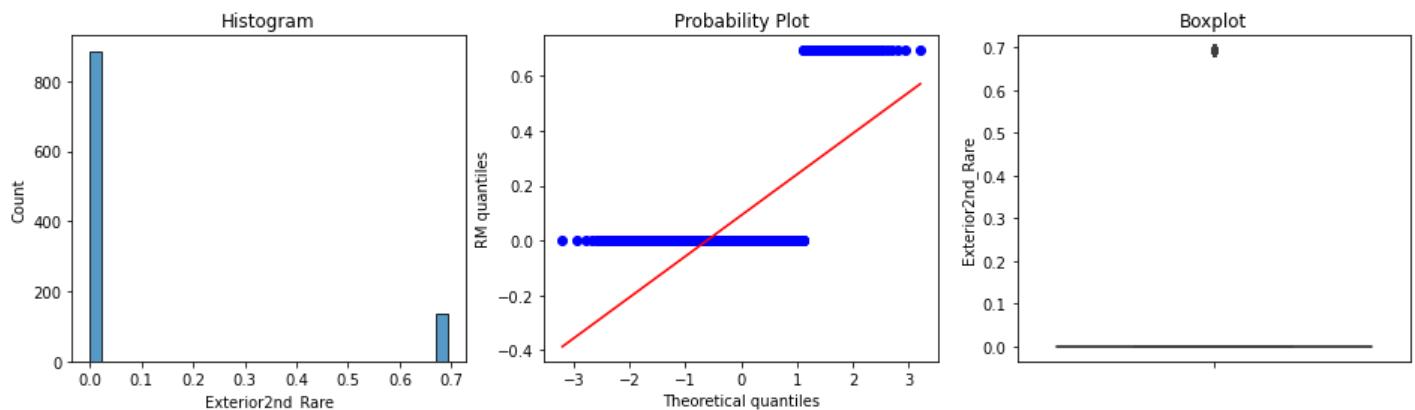
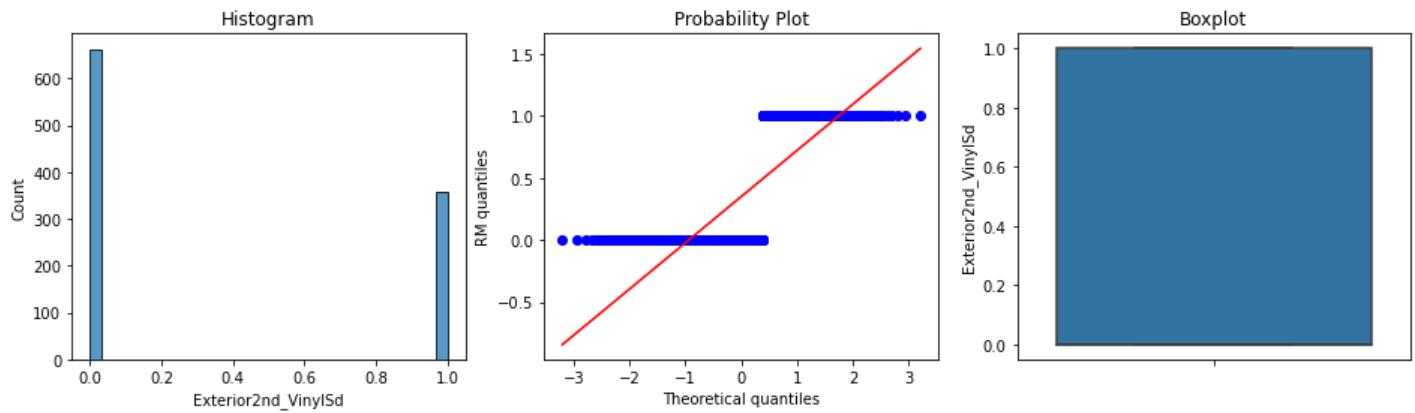


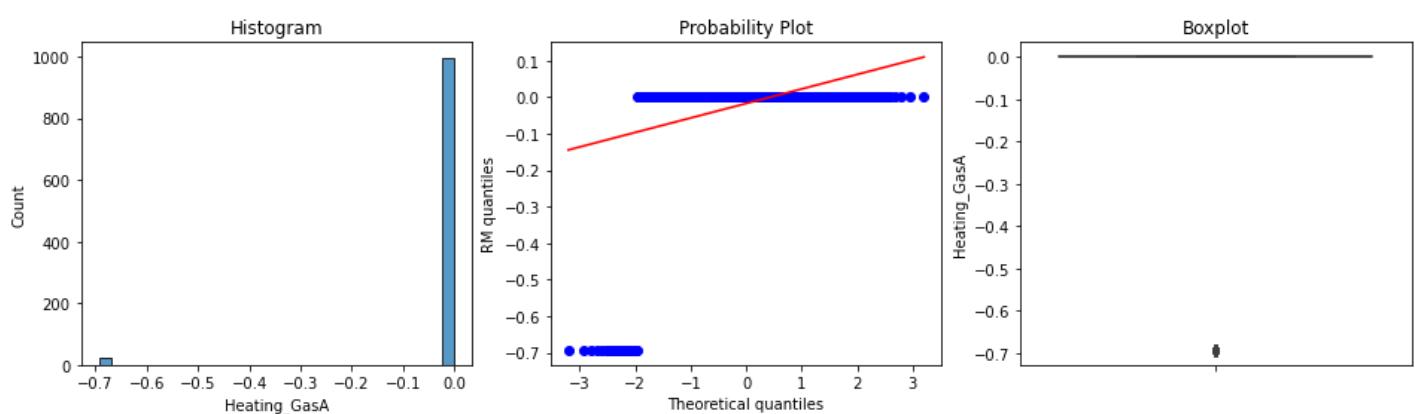
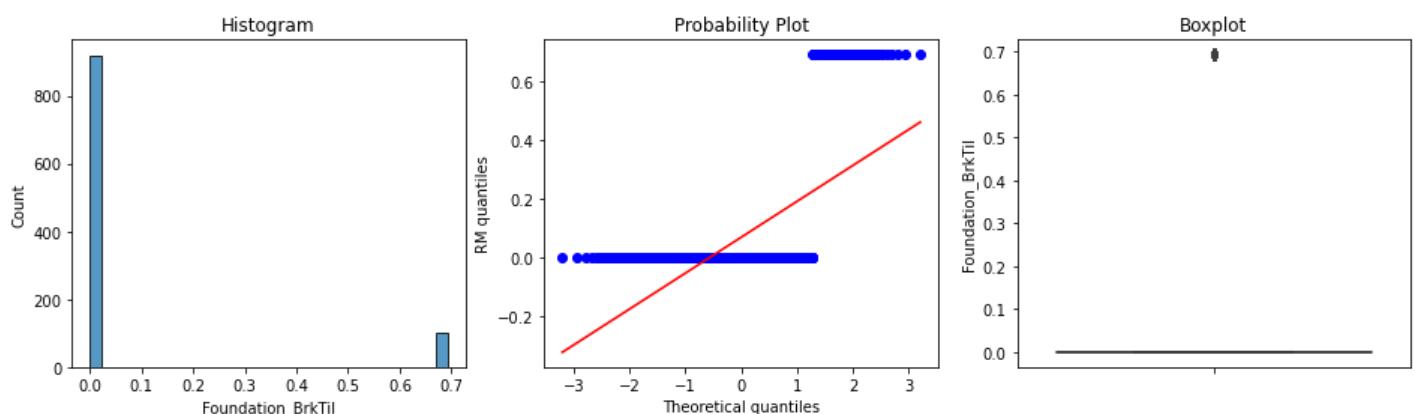
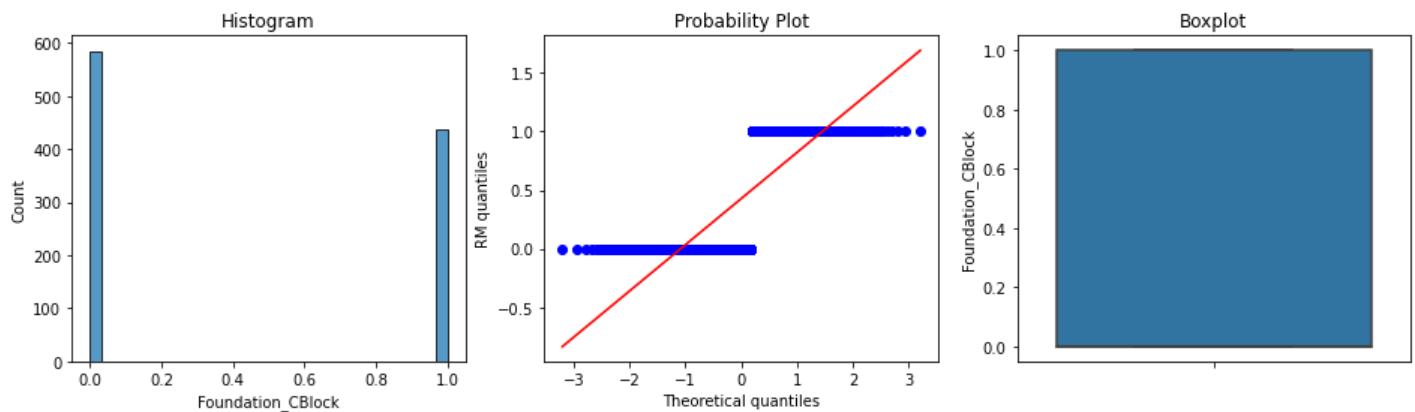
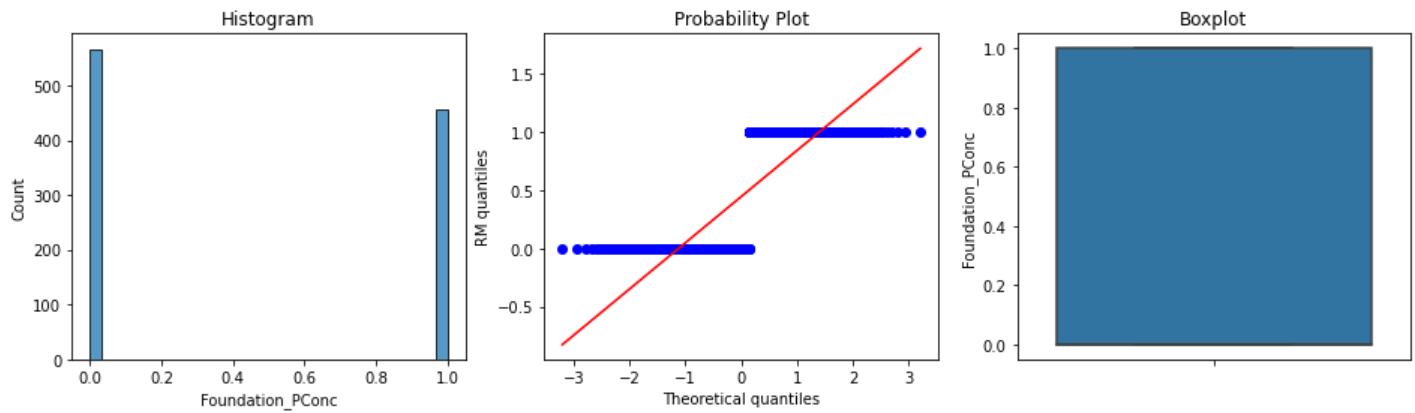


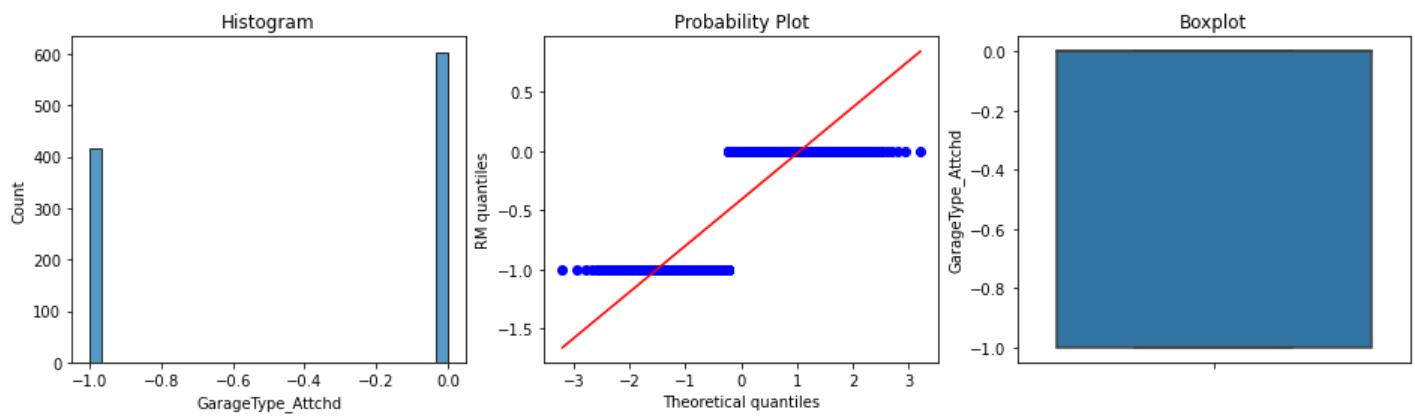
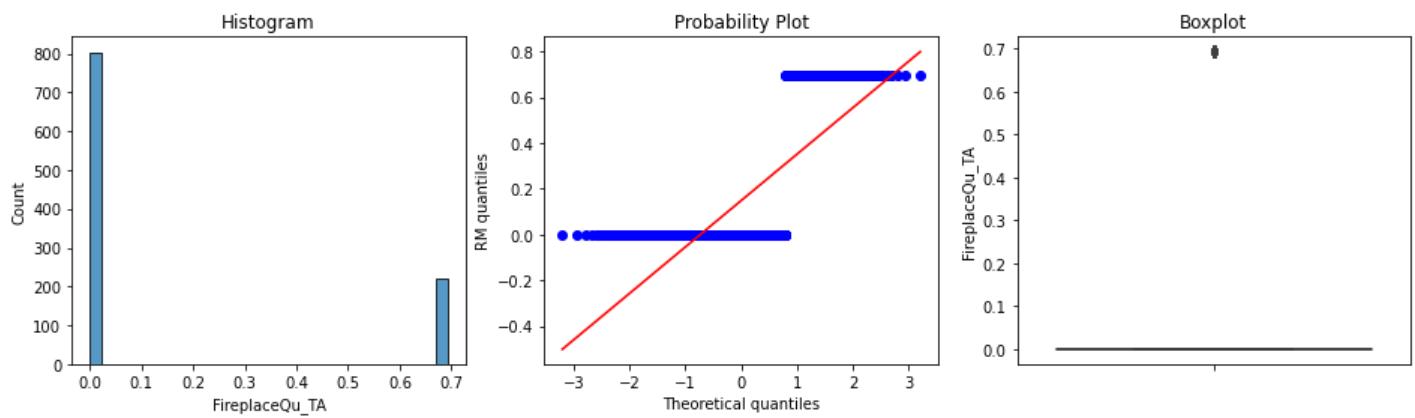
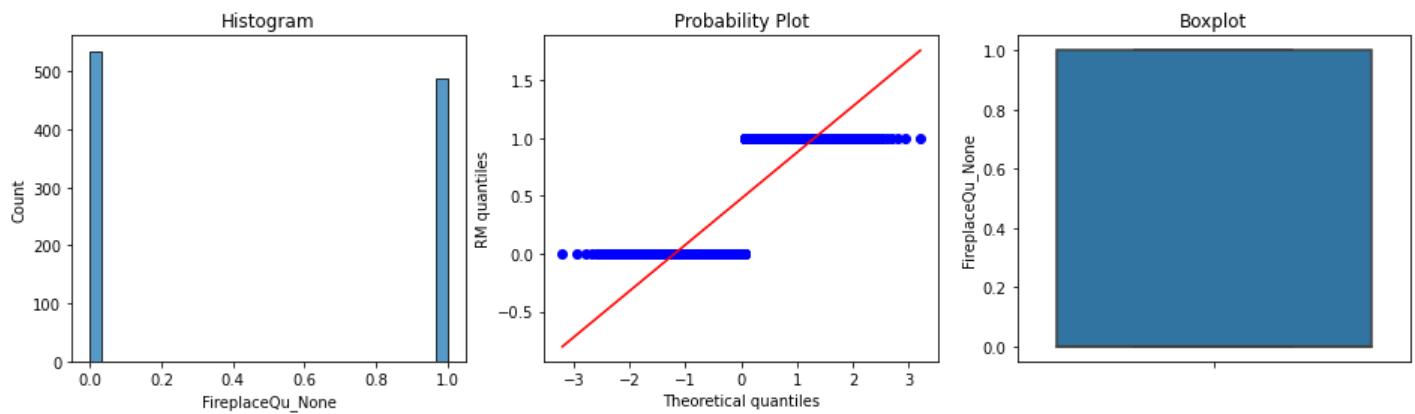
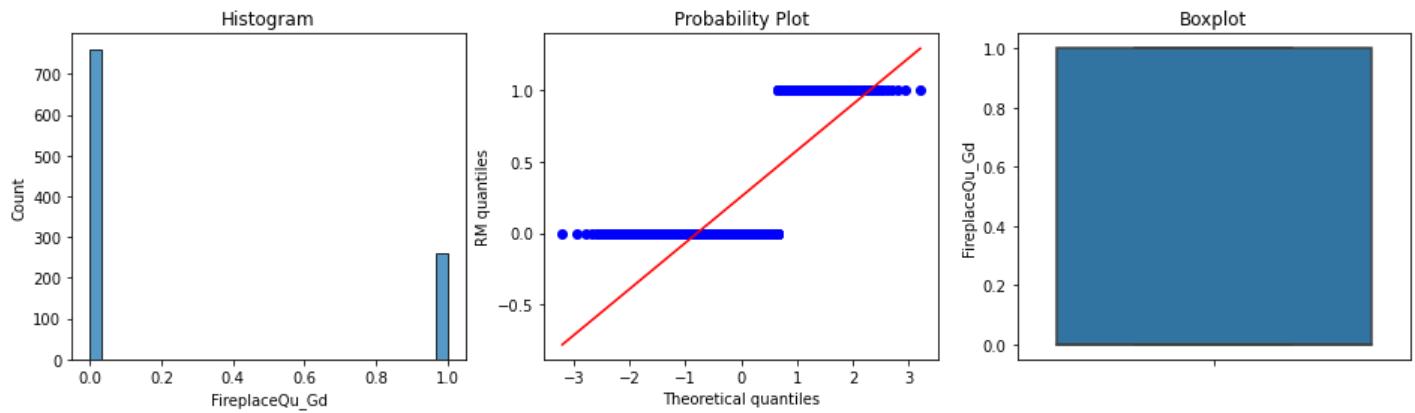


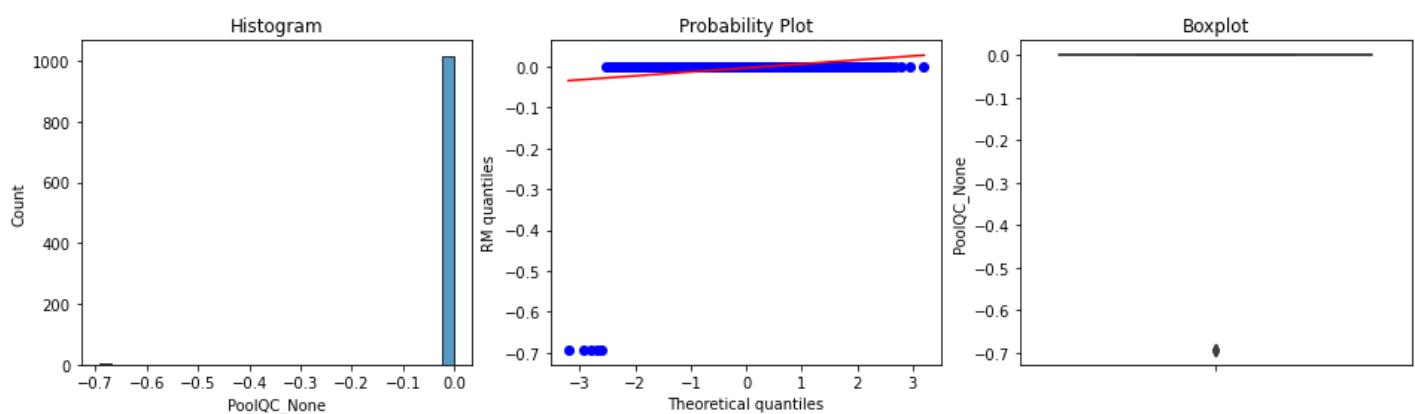
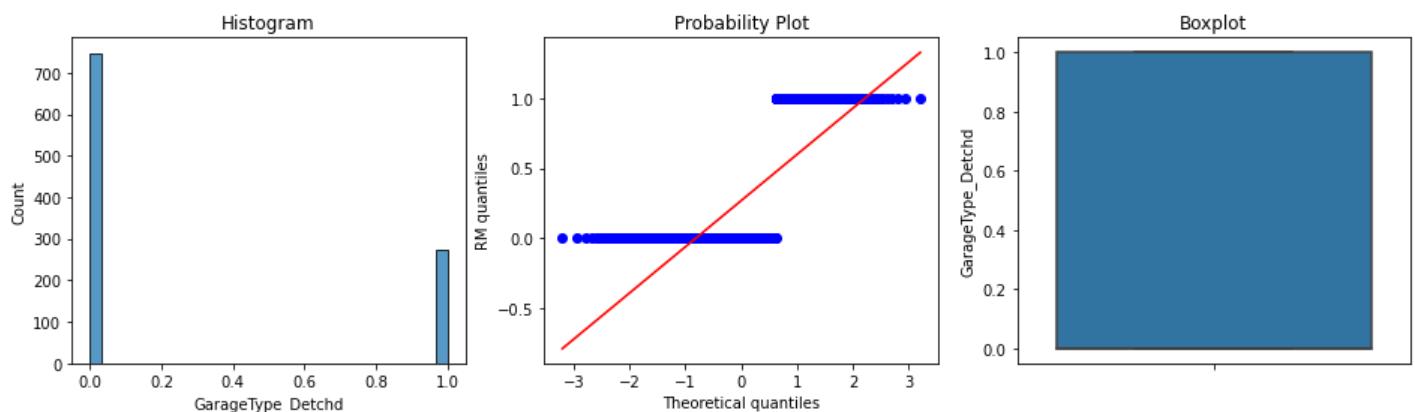
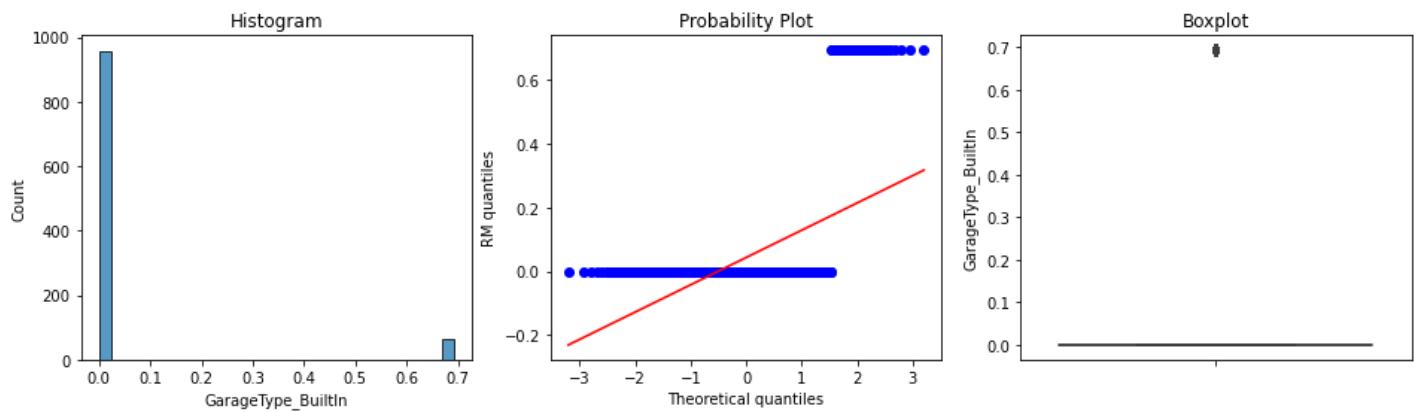
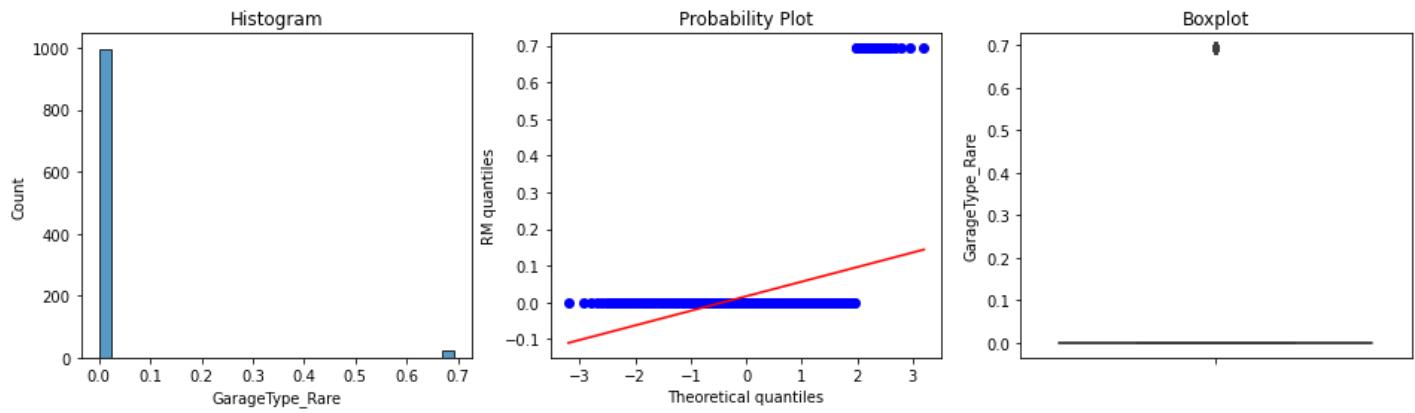


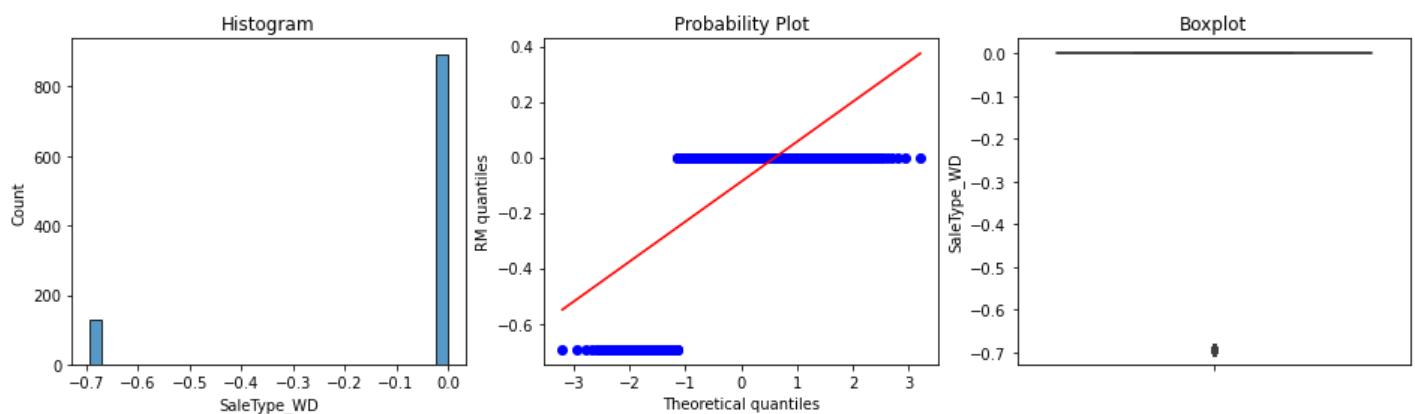
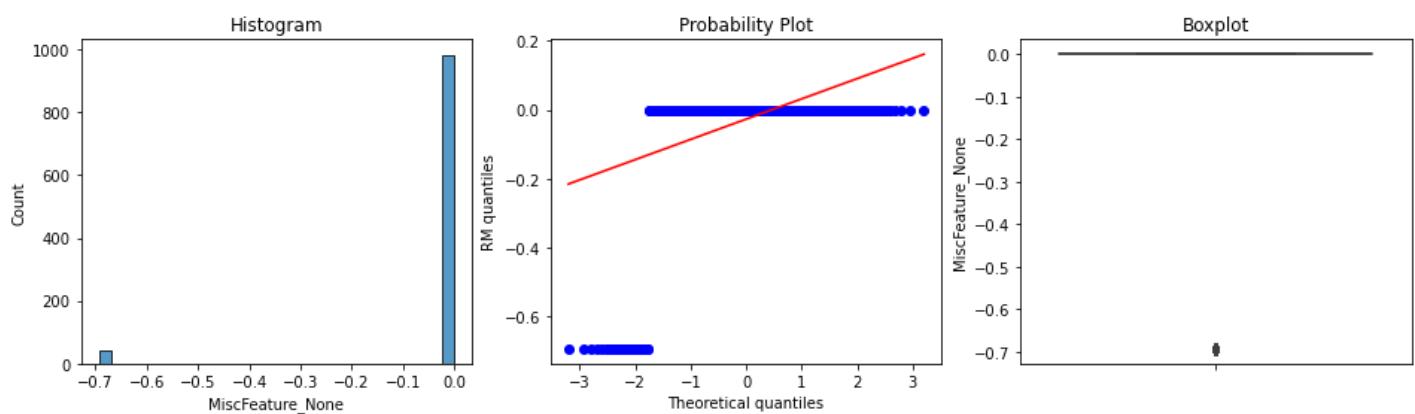
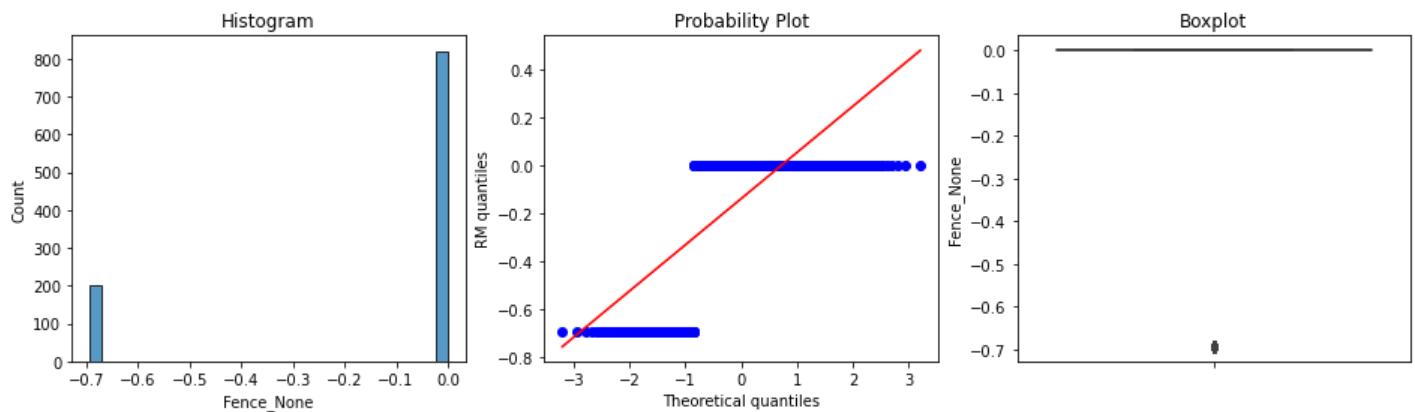
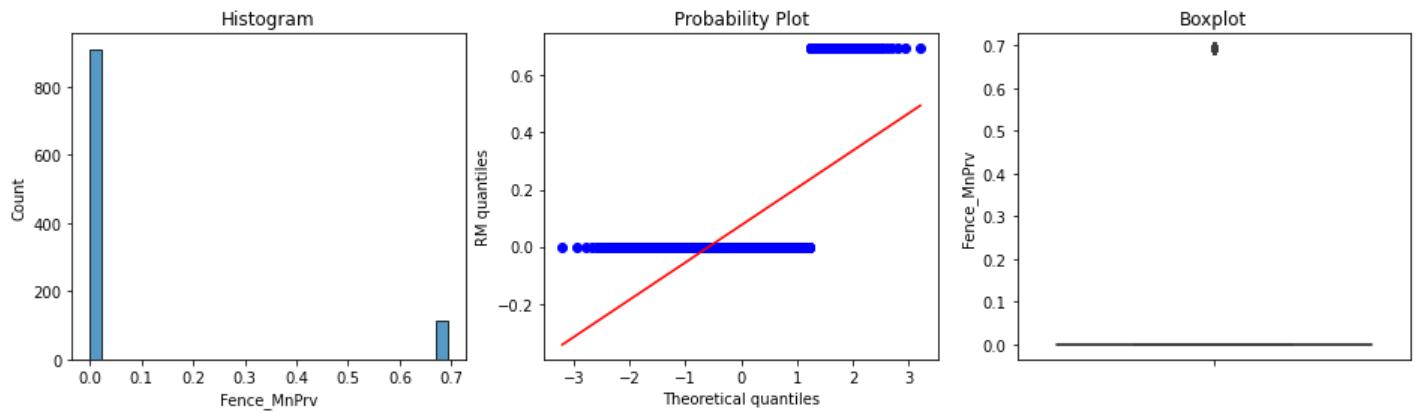


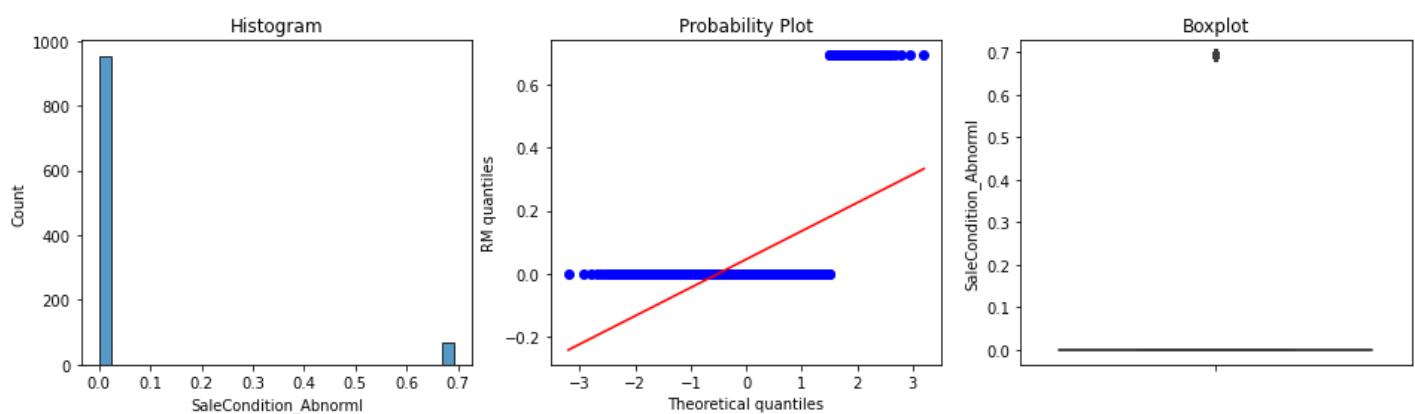
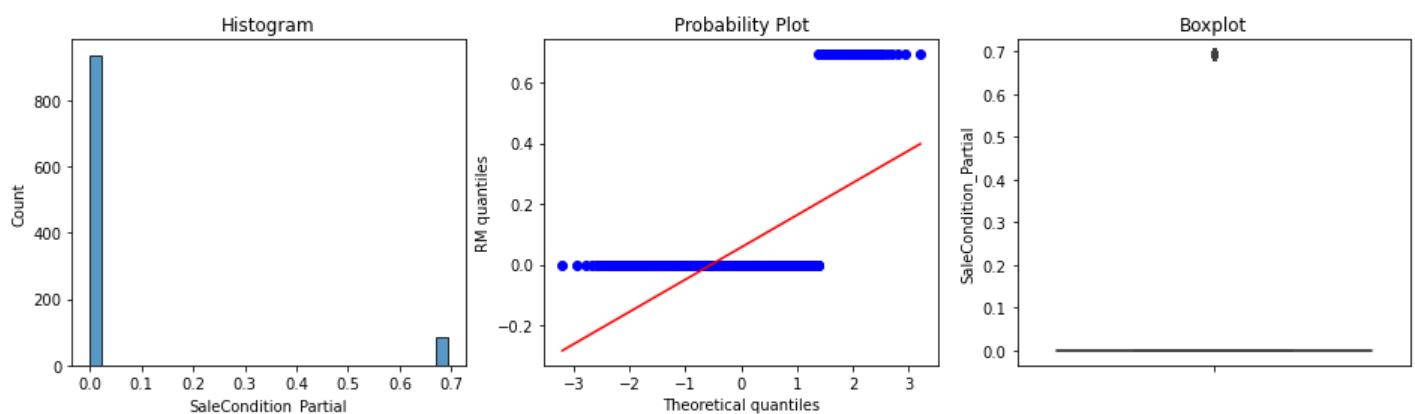
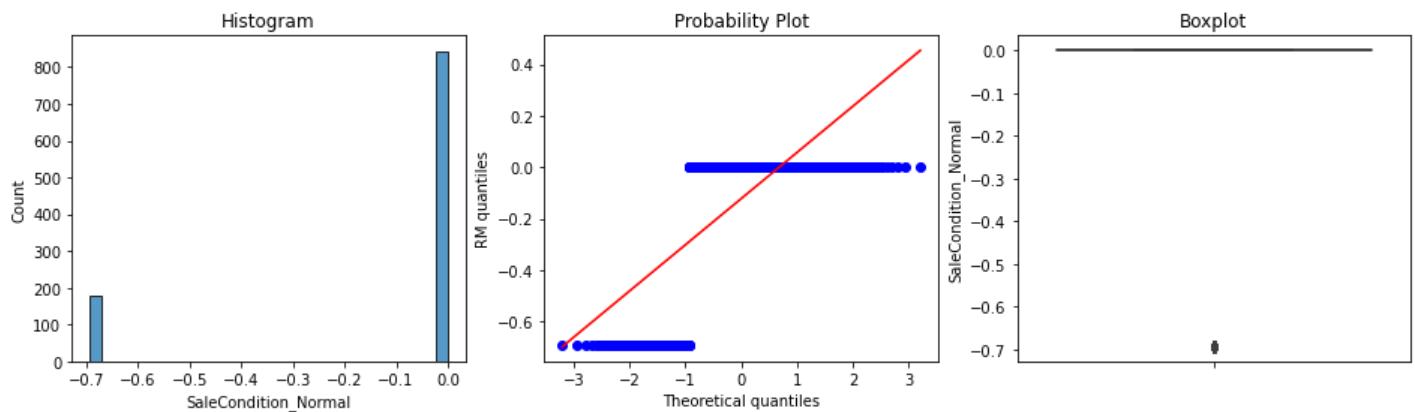
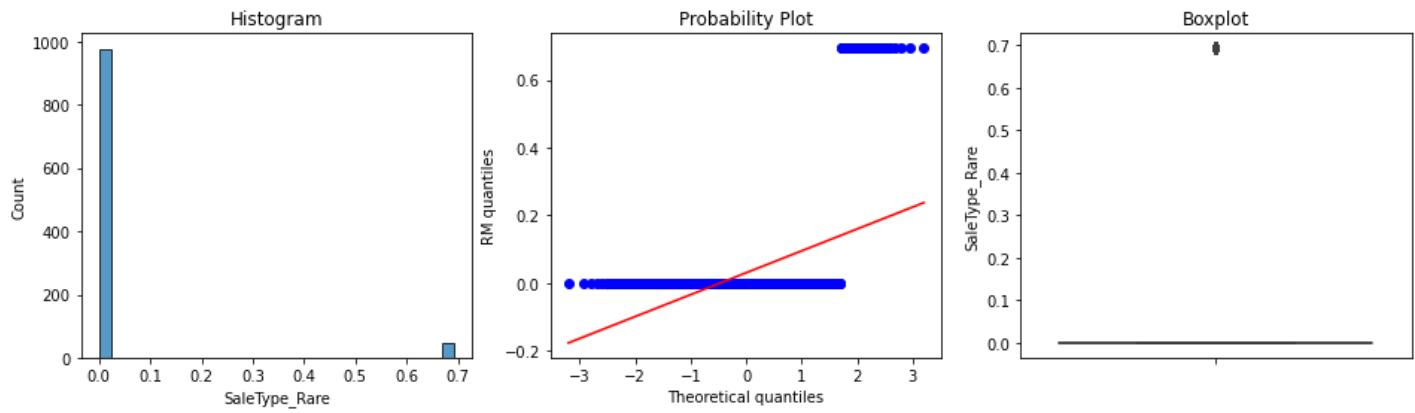


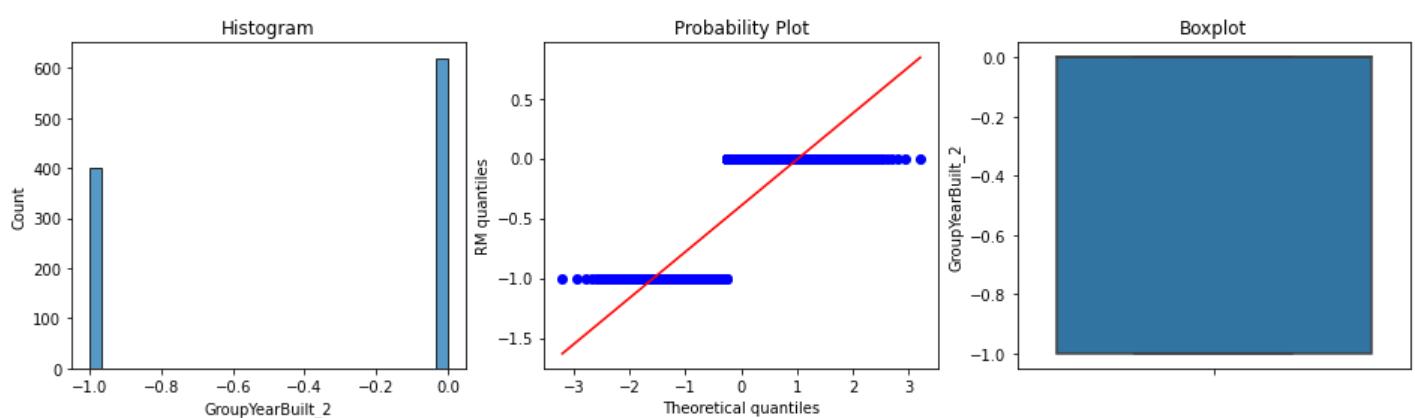
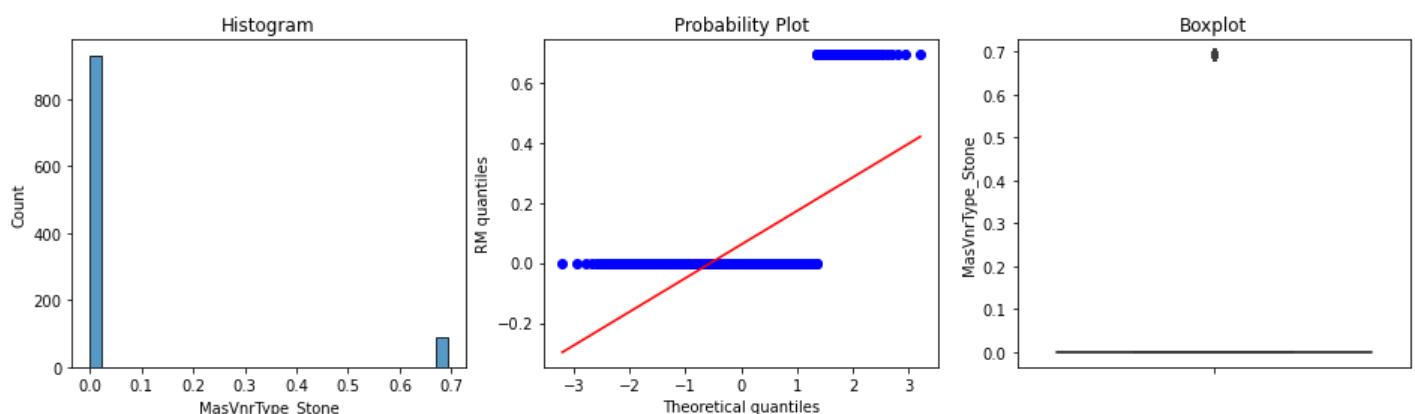
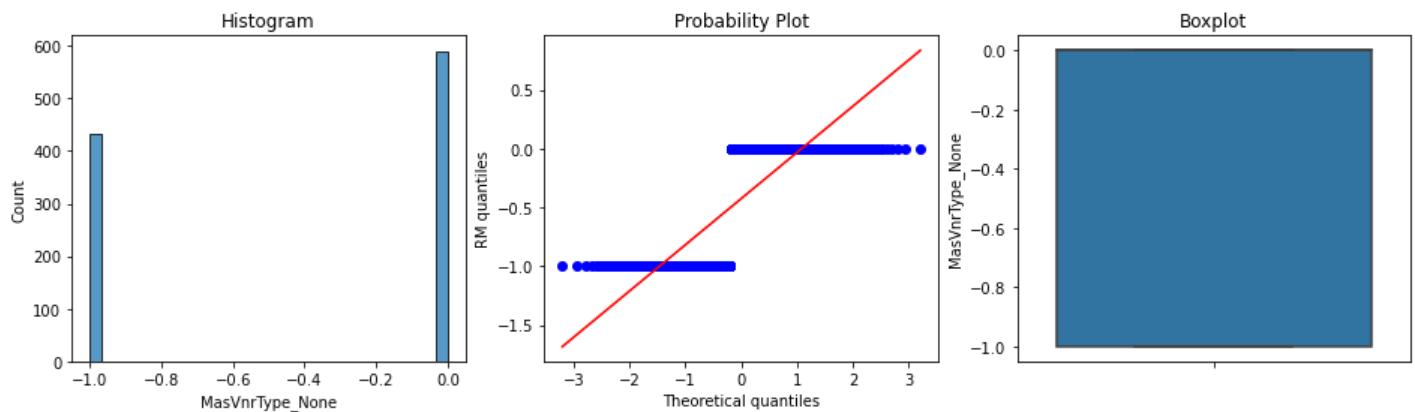
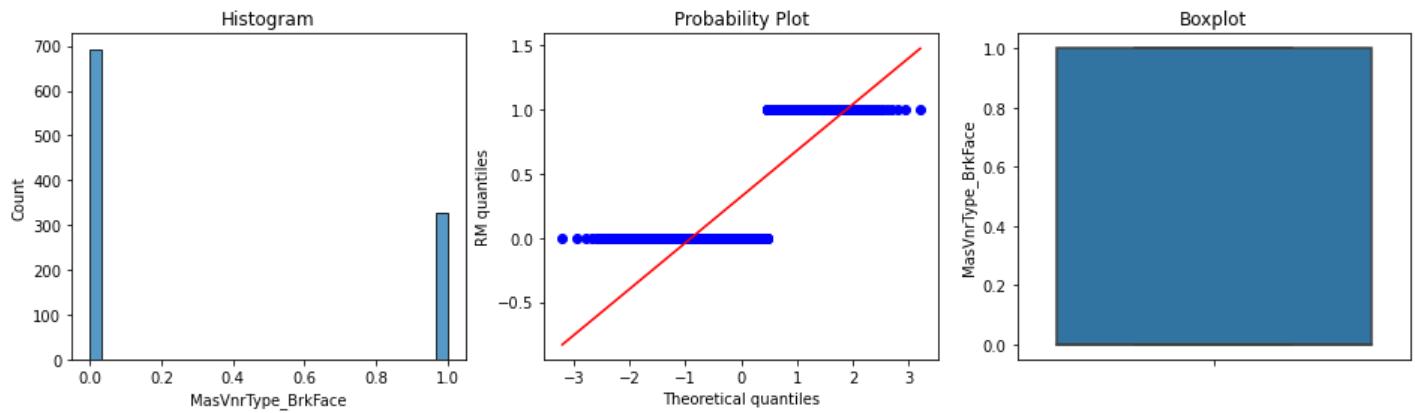


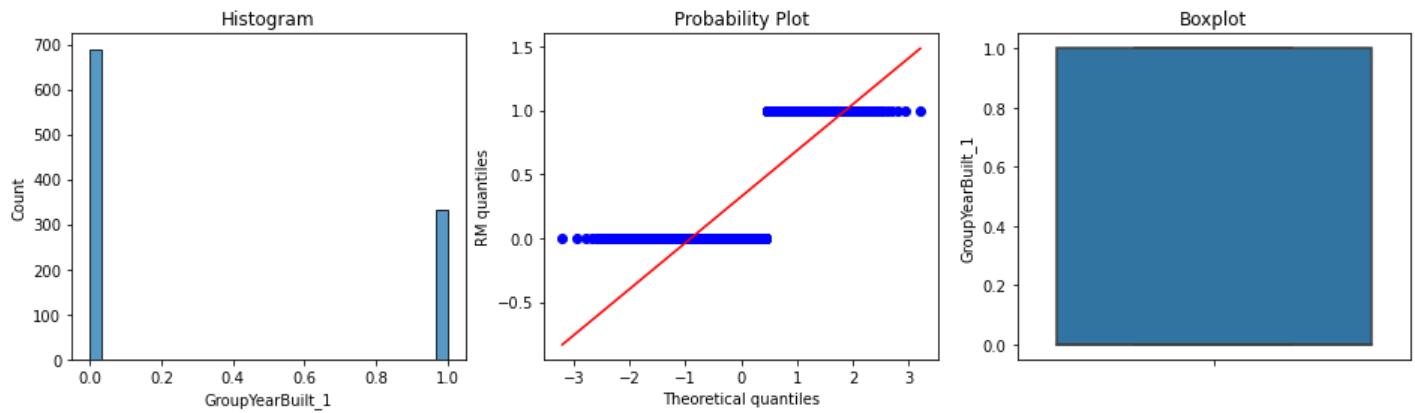












Feature Selection

before do the next process of built model for prediction, we need to select some feature that most useful for our prediction and before do the process of feature selection we'll compare score of dataset before/after by use the neg_mean_squared_error with the less score show the import of model preformance

```
In [263...]: model=GradientBoostingRegressor()

def score_dataset(X, y, model):
    log_y = np.log(y)

    score = cross_val_score(
        model, X, log_y, cv=5, scoring="neg_mean_squared_error",
    )
    score = -1 * score.mean()
    score = np.sqrt(score)
    return score
```

```
In [263...]: score_dataset(X_train, y_train, model)
```

```
Out[263...]: 0.14289960519051648
```

we'll check the feature that contain constant value and will remove it to improve model performance

```
In [264...]: quasi_constant=[]
for col in X_train.columns:
    predominant=(X_train[col].value_counts()/np.float(
        len(X_train))).sort_values(ascending=False).values[0]

    if predominant>0.95:
        quasi_constant.append(col)

len(quasi_constant)
```

```
Out[264...]: 15
```

As we notice that there are 15 features that constain the constant values, then we remove those from our dataset

```
In [264...]: quasi_constant
```

```
Out[264...]: ['Utilities',
              'LowQualFinSF',
```

```
'KitchenAbvGr',
'ThirdSsnPorch ',
'PoolArea',
'MiscVal',
'Lotsize',
'Street_Pave',
'Condition2_Norm',
'RoofMatl_CompShg',
'Heating_GasA',
'GarageType_Rare',
'PoolQC_None',
'MiscFeature_None',
'SaleType_Rare']
```

In [264...]

```
for col in quasi_constant:
    predo=X_train[col].value_counts() / np.float(len(X_train))
    print(predo)
```

```
0.000000 0.999022
-0.693147 0.000978
Name: Utilities, dtype: float64
0.000000 0.981409
4.394449 0.002935
3.988984 0.000978
4.976734 0.000978
6.161207 0.000978
6.177944 0.000978
5.973810 0.000978
5.968708 0.000978
6.246107 0.000978
6.350886 0.000978
5.986452 0.000978
5.459586 0.000978
5.327876 0.000978
5.953243 0.000978
6.042633 0.000978
4.795791 0.000978
6.242223 0.000978
5.888878 0.000978
Name: LowQualFinSF, dtype: float64
0.000000 0.953033
0.405465 0.045988
-0.693147 0.000978
Name: KitchenAbvGr, dtype: float64
0.000000 0.983366
5.379897 0.001957
5.129899 0.001957
5.771441 0.000978
6.011267 0.000978
4.875197 0.000978
5.283204 0.000978
5.198497 0.000978
5.673323 0.000978
5.505332 0.000978
5.036953 0.000978
5.209486 0.000978
4.976734 0.000978
4.948760 0.000978
6.232448 0.000978
5.720312 0.000978
Name: ThirdSsnPorch , dtype: float64
0.000000 0.995108
6.320768 0.000978
6.175867 0.000978
```

```

6.240276    0.000978
6.253829    0.000978
6.605298    0.000978
Name: PoolArea, dtype: float64
0.000000    0.960861
5.993961    0.007828
6.216606    0.006849
6.552508    0.004892
6.111467    0.003914
7.601402    0.003914
7.090910    0.001957
6.175867    0.001957
8.160804    0.000978
7.170888    0.000978
9.024131    0.000978
5.860786    0.000978
6.431331    0.000978
7.048386    0.000978
7.244942    0.000978
7.824446    0.000978
Name: MiscVal, dtype: float64
0.000000    0.994129
0.405465    0.001957
-0.693147   0.001957
0.693147    0.001957
Name: Lotsize, dtype: float64
0.000000    0.996086
-0.693147   0.003914
Name: Street_Pave, dtype: float64
0.000000    0.991194
-0.693147   0.008806
Name: Condition2_Norm, dtype: float64
0.000000    0.981409
-0.693147   0.018591
Name: RoofMatl_CompShg, dtype: float64
0.000000    0.975538
-0.693147   0.024462
Name: Heating_GasA, dtype: float64
0.000000    0.975538
0.693147    0.024462
Name: GarageType_Rare, dtype: float64
0.000000    0.995108
-0.693147   0.004892
Name: PoolQC_None, dtype: float64
0.000000    0.960861
-0.693147   0.039139
Name: MiscFeature_None, dtype: float64
0.000000    0.955969
0.693147    0.044031
Name: SaleType_Rare, dtype: float64

```

In [264...]

```
X_train.drop(labels=quasi_constant, axis=1, inplace=True)
X_test.drop(labels=quasi_constant, axis=1, inplace=True)
df_test.drop(labels=quasi_constant, axis=1, inplace=True)
```

In [264...]

```
score_dataset(X_train, y_train, model)
```

Out[264...]

```
0.14358262553369477
```

The performance after remove some constant feature, a bit increase

check some feature duplicate

In [264]

```
def find_dupl(data):
    duplic_dic={}
    duplic_list=[]

    for i in range(0, len(data.columns)):
        col1=data.columns[i]
        if col1 not in duplic_list:
            duplic_dic[col1]=[]
            for col2 in data.columns[i+1:]:
                if data[col1].equals(data[col2]):
                    duplic_dic[col1].append(col2)
                    duplic_list.append(col2)

    return duplic_list, duplic_dic
```

In [264]

```
find_dupl(X_train)
```

Out[264]

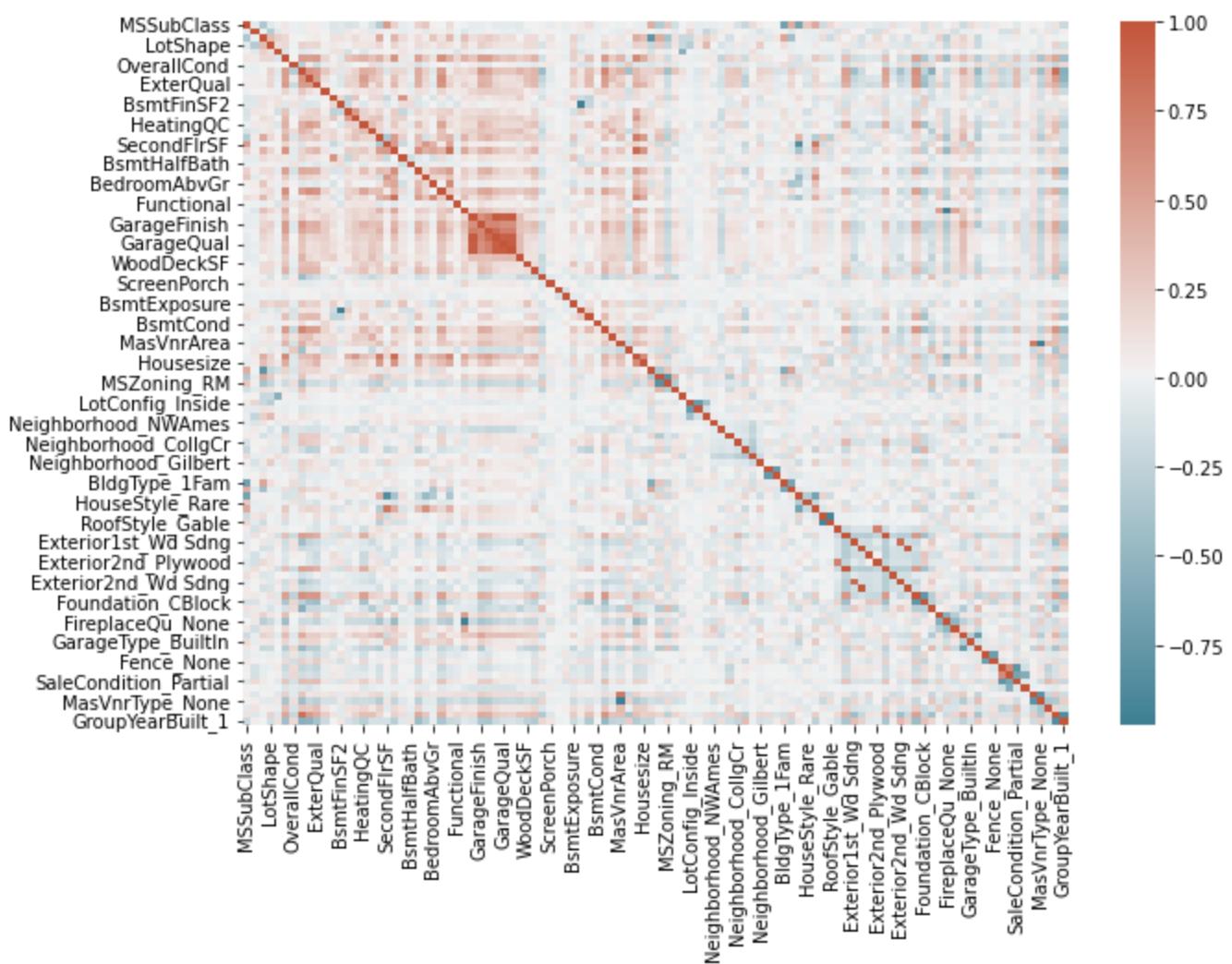
```
([],
 {'MSSubClass': [],
 'LotFrontage': [],
 'LotArea': [],
 'LotShape': [],
 'LandSlope': [],
 'OverallQual': [],
 'OverallCond': [],
 'YearBuilt': [],
 'YearRemodAdd': [],
 'ExterQual': [],
 'ExterCond': [],
 'BsmtFinSF1': [],
 'BsmtFinSF2': [],
 'BsmtUnfSF': [],
 'TotalBsmtSF': [],
 'HeatingQC': [],
 'CentralAir': [],
 'FirstFlrSF': [],
 'SecondFlrSF': [],
 'GrLivArea': [],
 'BsmtFullBath': [],
 'BsmtHalfBath': [],
 'FullBath': [],
 'HalfBath': [],
 'BedroomAbvGr': [],
 'KitchenQual': [],
 'TotRmsAbvGrd': [],
 'Functional': [],
 'Fireplaces': [],
 'GarageYrBlt': [],
 'GarageFinish': [],
 'GarageCars': [],
 'GarageArea': [],
 'GarageQual': [],
 'GarageCond': [],
 'PavedDrive': [],
 'WoodDeckSF': [],
 'OpenPorchSF': [],
 'EnclosedPorch': [],
 'ScreenPorch': [],
 'MoSold': [],
 'YrSold': [],
 'BsmtExposure': [],
 'BsmtFinType2': [],
 'BsmtFinType1': [])
```

```
'BsmtCond': [],
'BsmtQual': [],
'Electrical': [],
'MasVnrArea': [],
'Remodel': [],
'TotalSF': [],
'Housesize': [],
'TotalPerLot': [],
'MSZoning_RL': [],
'MSZoning_RM': [],
'Alley_None': [],
'LandContour_Lvl': [],
'LotConfig_Inside': [],
'LotConfig_Corner': [],
'LotConfig_CulDSac': [],
'Neighborhood_NWAmes': [],
'Neighborhood_Edwards': [],
'Neighborhood_Somerst': [],
'Neighborhood_CollgCr': [],
'Neighborhood_NAmes': [],
'Neighborhood_Rare': [],
'Neighborhood_Gilbert': [],
'Condition1_Norm': [],
'Condition1_Rare': [],
'BldgType_1Fam': [],
'BldgType_TwnhsE': [],
'HouseStyle_1Story': [],
'HouseStyle_Rare': [],
'HouseStyle_2Story': [],
'RoofStyle_Hip': [],
'RoofStyle_Gable': [],
'Exterior1st_Plywood': [],
'Exterior1st_VinylSd': [],
'Exterior1st_Wd_Sdng': [],
'Exterior1st_MetalSd': [],
'Exterior1st_HdBoard': [],
'Exterior2nd_Plywood': [],
'Exterior2nd_VinylSd': [],
'Exterior2nd_Rare': [],
'Exterior2nd_Wd_Sdng': [],
'Exterior2nd_MetalSd': [],
'Foundation_PConc': [],
'Foundation_CBlock': [],
'Foundation_BrkTil': [],
'FireplaceQu_Gd': [],
'FireplaceQu_None': [],
'FireplaceQu_TA': [],
'GarageType_Attchd': [],
'GarageType_BuiltIn': [],
'GarageType_Detchd': [],
'Fence_MnPrv': [],
'Fence_None': [],
'SaleType_WD': [],
'SaleCondition_Normal': [],
'SaleCondition_Partial': [],
'SaleCondition_Abnorml': [],
'MasVnrType_BrkFace': [],
'MasVnrType_None': [],
'MasVnrType_Stone': [],
'GroupYearBuilt_2': [],
'GroupYearBuilt_1': []})
```

Check correlation of dataset

```
In [264... correlation = X_train.corr(method='pearson')
cmap=sns.diverging_palette(220,20,as_cmap=True)
fig,ax=plt.subplots(figsize=(10,7))
sns.heatmap(correlation,cmap=cmap)
```

Out [264... <AxesSubplot:>



Next we'll compare the 3 method for select feature and find the best one

- Lasso that we'll filter out the feature that 0 coefficient
- RFE with Gradientboosting that we'll filter out the feature that 0 coefficient
- RFE with Randomforest that we'll filter out the feature that 0 coefficient

```
In [264... lcv = LassoCV()
lcv.fit(X_train,y_train)
print('Optimal alpha = {0:.3f}'.format(lcv.alpha_))

r_squared = lcv.score(X_test,y_test)
print('The model explains {0:.1%} of the test set variance'.format(r_squared))

lcv_mask = lcv.coef_!=0
print('{0} features out of {1} selected'.format(sum(lcv_mask), len(lcv_mask)))
```

Optimal alpha = 1510.256
The model explains 76.9% of the test set variance
28 features out of 106 selected

```
In [265... rfe_gb = RFE(estimator=GradientBoostingRegressor(),
```

```
n_features_to_select=15, step=10, verbose=1)
rfe_gb.fit(X_train, y_train)

r_squared = rfe_gb.score(X_test, y_test)
print('The model can explain {0:.1%} of the variance in the test set'.format(r_squared))

gb_mask = rfe_gb.support_!=0
```

```
Fitting estimator with 106 features.
Fitting estimator with 96 features.
Fitting estimator with 86 features.
Fitting estimator with 76 features.
Fitting estimator with 66 features.
Fitting estimator with 56 features.
Fitting estimator with 46 features.
Fitting estimator with 36 features.
Fitting estimator with 26 features.
Fitting estimator with 16 features.
The model can explain 90.8% of the variance in the test set
```

In [265...]

```
rfe_rf = RFE(estimator=RandomForestRegressor(),
               n_features_to_select=15, step=3, verbose=1)
rfe_rf.fit(X_train, y_train)

r_squared = rfe_rf.score(X_test, y_test)
print('The model can explain {0:.1%} of the variance in the test set'.format(r_squared))

rf_mask = rfe_rf.support_
```

```
Fitting estimator with 106 features.
Fitting estimator with 103 features.
Fitting estimator with 100 features.
Fitting estimator with 97 features.
Fitting estimator with 94 features.
Fitting estimator with 91 features.
Fitting estimator with 88 features.
Fitting estimator with 85 features.
Fitting estimator with 82 features.
Fitting estimator with 79 features.
Fitting estimator with 76 features.
Fitting estimator with 73 features.
Fitting estimator with 70 features.
Fitting estimator with 67 features.
Fitting estimator with 64 features.
Fitting estimator with 61 features.
Fitting estimator with 58 features.
Fitting estimator with 55 features.
Fitting estimator with 52 features.
Fitting estimator with 49 features.
Fitting estimator with 46 features.
Fitting estimator with 43 features.
Fitting estimator with 40 features.
Fitting estimator with 37 features.
Fitting estimator with 34 features.
Fitting estimator with 31 features.
Fitting estimator with 28 features.
Fitting estimator with 25 features.
Fitting estimator with 22 features.
Fitting estimator with 19 features.
Fitting estimator with 16 features.
The model can explain 91.1% of the variance in the test set
```

After compare we'll use Gradientboosting that has the best score 91.5% with remain 15 features by set mark to filter out the unuseful feature out from dataset

```
In [265...]: X_train=X_train.loc[:,gb_mask]
X_test=X_test.loc[:,gb_mask]
df_test=df_test.loc[:,gb_mask]
```

modeling

Compare model for select the best algorithm that can generate the best score for this dataset by use model 'RandomForestRegressor','LinearRegression','Ridge','GradientBoostingRegressor'

```
In [265...]: model_pipeline=[]
model_pipeline.append(RandomForestRegressor())
model_pipeline.append(LinearRegression())
model_pipeline.append(Ridge())
model_pipeline.append(GradientBoostingRegressor())
```

```
In [265...]: model_list=['RandomForestRegressor','LinearRegression','Ridge','GradientBoostingRegressor']
mae_list=[]
r2score_list=[]
for model in model_pipeline:
    model.fit(X_train,y_train)
    pred=model.predict(X_test)
    mae_list.append(mean_absolute_error(y_test,pred))
    r2score_list.append(r2_score(y_test,pred))
```

```
In [265...]: result_model=pd.DataFrame({'Model':model_list,'MAE':mae_list,'R2_score':r2score_list})
result_model
```

	Model	MAE	R2_score
0	RandomForestRegressor	15763.699338	0.905089
1	LinearRegression	32650.489071	0.701703
2	Ridge	32555.247073	0.702795
3	GradientBoostingRegressor	16197.976410	0.911617

The best algorithm is the GradientBoostingRegressor that can generate the score 91.016% and the next algorithm is RandomForestRegressor that score is 90.779%

Hyperparameter tuning

For increasing the performance model we'll tuning hyperparameter for randomforest and GradientBoosting by using random_grid method that will random the hyperparameter that we select and find the best score from random hyperparameter

```
In [265...]: model= RandomForestRegressor(random_state=42,criterion='mse')

n_estimators=[int(x) for x in np.linspace(start=200, stop= 2000, num=10)]
max_features= ['auto', 'sqrt']
max_depth=[int(x) for x in np.linspace(10,110, num=11)]
max_depth.append(None)
min_samples_split=[2,5,10]
min_samples_leaf=[1,2,4]
bootstrap=[True, False]
```

```
In [265...]
```

```
random_grid={'n_estimators':n_estimators,'max_features':max_features,
             'max_depth':max_depth,
             'min_samples_split':min_samples_split,
             'min_samples_leaf':min_samples_leaf,
             'bootstrap':bootstrap}
```

In [265...]

```
random=RandomizedSearchCV(estimator=model,
                           param_distributions=random_grid,
                           n_iter=100, cv=3, verbose=2,
                           random_state=42, n_jobs=-1)
```

In [265...]

```
random.get_params
```

Out[265...]

```
<bound method BaseEstimator.get_params of RandomizedSearchCV(cv=3,
                                                               estimator=RandomForestRegressor(criterion='mse',
                                                               random_state=42),
                                                               n_iter=100, n_jobs=-1,
                                                               param_distributions={'bootstrap': [True, False],
                                                               'max_depth': [10, 20, 30, 40, 50, 60,
                                                               70, 80, 90, 100, 110,
                                                               None],
                                                               'max_features': ['auto', 'sqrt'],
                                                               'min_samples_leaf': [1, 2, 4],
                                                               'min_samples_split': [2, 5, 10],
                                                               'n_estimators': [200, 400, 600, 800,
                                                               1000, 1200, 1400, 1600,
                                                               1800, 2000]},
                                                               random_state=42, verbose=2)>
```

In [266...]

```
random.fit(X_train,y_train)
```

Fitting 3 folds for each of 100 candidates, totalling 300 fits

```
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
    warn(
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
    warn(
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
    warn(
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
    warn(
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
    warn(
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
    warn(
/Users/wittawatmuangkot/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensemble/_forests.py:396: FutureWarning: Criterion 'mse' was deprecated in v1.0 and will be removed in version 1.2. Use `criterion='squared_error'` which is equivalent.
```


In [266...]

`random.best_params_`

```
Out[266... {'n_estimators': 1600,
'min_samples_split': 5,
'min_samples_leaf': 1,
'max_features': 'sqrt',
'max_depth': 70,
'bootstrap': False}
```

```
In [266... def evaluation_model(model, X_test, y_test):
    pred=model.predict(X_test)
    errors=abs(pred-y_test)
    avg_error=100*np.mean(errors/y_test)
    acc=100-avg_error
    print('AVG Error : {:.4f} degrees'.format(np.mean(errors)))
    print('Accuracy : {:.2f}'.format(acc))
    return acc
```

```
In [266... base_model=RandomForestRegressor(n_estimators=10, random_state=42)
base_model.fit(X_train,y_train)
evaluation_model(base_model,X_test,y_test)
```

```
AVG Error : 16963.7537 degrees
```

```
Accuracy : 89.82
```

```
Out[266... 89.8161520581492
```

```
In [266... best_random=random.best_estimator_
evaluation_model(best_random,X_test,y_test)
```

```
AVG Error : 17107.9342 degrees
```

```
Accuracy : 89.53
```

```
Out[266... 89.53067760271445
```

```
In [266... best_random.fit(X_train,y_train)
```

```
Out[266... RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=70,
max_features='sqrt', min_samples_split=5,
n_estimators=1600, random_state=42)
```

```
In [266... pred=best_random.predict(X_test)
r_score=r2_score(y_test,pred)
r_score
```

```
Out[266... 0.8906265037236445
```

```
In [266... param={
    "n_estimators": [1, 2, 5, 10, 20, 50, 100, 200, 500],
    "max_leaf_nodes": [2, 5, 10, 20, 50, 100],
    "learning_rate": loguniform(0.01, 1)
}
random_cv=RandomizedSearchCV(
    GradientBoostingRegressor(), param_distributions=param,
    scoring='neg_mean_absolute_error', n_iter=20,
    random_state=42, n_jobs=2
)
random_cv.fit(X_train,y_train)
```

```
[CV] END bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 0.7s
```

```
[CV] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=4, min_samples
```

```
_split=2, n_estimators=2000; total time= 7.9s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
_split=5, n_estimators=1800; total time= 7.2s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=10, n_estimators=800; total time= 1.4s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=2, n_estimators=1000; total time= 1.9s
[CV] END bootstrap=False, max_depth=110, max_features=auto, min_samples_leaf=2, min_samples_
_split=10, n_estimators=1800; total time= 7.2s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=5, n_estimators=1400; total time= 2.4s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=5, n_estimators=1400; total time= 2.5s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=5, n_estimators=1000; total time= 1.6s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=5, n_estimators=1000; total time= 1.7s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=2, n_estimators=1200; total time= 2.0s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=10, n_estimators=1200; total time= 1.9s
[CV] END bootstrap=False, max_depth=50, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=2, n_estimators=800; total time= 1.5s
[CV] END bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=10, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=2, n_estimators=1800; total time= 2.7s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=2, n_estimators=1000; total time= 1.6s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=2, n_estimators=1000; total time= 1.8s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=10, n_estimators=400; total time= 0.6s
[CV] END bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=2, min_samples_
_split=2, n_estimators=2000; total time= 6.7s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=2, min_samples_
_split=10, n_estimators=2000; total time= 8.0s
[CV] END bootstrap=False, max_depth=60, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=2, n_estimators=600; total time= 0.9s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=2, min_samples_
_split=2, n_estimators=1800; total time= 6.3s
[CV] END bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=2, n_estimators=200; total time= 0.4s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=2, n_estimators=400; total time= 0.5s
[CV] END bootstrap=False, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_
_split=10, n_estimators=1200; total time= 5.4s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
_split=2, n_estimators=1800; total time= 2.5min
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=2, n_estimators=1400; total time= 2.7s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=5, n_estimators=1800; total time= 4.4s
[CV] END bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 8.6s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 0.6s
[CV] END bootstrap=False, max_depth=60, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=2000; total time= 8.5s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=800; total time= 1.4s
[CV] END bootstrap=True, max_depth=110, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=600; total time= 1.0s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=800; total time= 1.5s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_
```

```
_split=2, n_estimators=600; total time= 1.0s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_samples_leaf=2, min_samples_
_split=10, n_estimators=400; total time= 1.7s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=2, n_estimators=2000; total time= 3.8s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=5, n_estimators=2000; total time= 3.7s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=5, n_estimators=800; total time= 1.2s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=5, n_estimators=1000; total time= 1.8s
[CV] END bootstrap=False, max_depth=60, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=5, n_estimators=600; total time= 1.1s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
_split=5, n_estimators=1800; total time= 6.8s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=10, n_estimators=800; total time= 1.4s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=2, min_samples_
_split=10, n_estimators=800; total time= 1.4s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 1.9s
[CV] END bootstrap=False, max_depth=110, max_features=auto, min_samples_leaf=2, min_sample
s_split=10, n_estimators=1800; total time= 8.0s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=2, n_estimators=1800; total time= 6.3s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=1200; total time= 2.1s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=1200; total time= 2.2s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=1200; total time= 2.0s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=5, n_estimators=800; total time= 1.5s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1800; total time= 2.7s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=5, n_estimators=800; total time= 2.7s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=110, max_features=sqrt, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 2.0s
[CV] END bootstrap=False, max_depth=110, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=2, n_estimators=600; total time= 1.0s
[CV] END bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1800; total time= 9.8s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sampl
es_split=5, n_estimators=1400; total time= 2.4s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 5.5s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1400; total time= 5.6s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1600; total time= 3.9s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=1400; total time= 2.4min
[CV] END bootstrap=False, max_depth=60, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=2000; total time= 10.7s
[CV] END bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=400; total time= 1.4s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=10, n_estimators=1600; total time= 3.8s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1000; total time= 1.9s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=600; total time= 1.9s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples
```

```
_split=5, n_estimators=1000; total time= 3.1s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=5, n_estimators=800; total time= 1.4s
[CV] END bootstrap=False, max_depth=40, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=10, n_estimators=600; total time= 1.0s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1600; total time= 2.8s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=4, min_sampl
es_split=5, n_estimators=2000; total time= 3.4s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_
_split=2, n_estimators=1000; total time= 5.1s
[CV] END bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 0.7s
[CV] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_
_split=2, n_estimators=2000; total time= 8.2s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 1.1s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 1.2s
[CV] END bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=5, n_estimators=800; total time= 1.6s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=10, n_estimators=2000; total time= 3.3s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=4, min_samples_
_split=2, n_estimators=1800; total time= 7.7s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=1, min_samples_
_split=5, n_estimators=1600; total time= 3.2s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=2, n_estimators=1800; total time= 6.3s
[CV] END bootstrap=False, max_depth=100, max_features=auto, min_samples_leaf=4, min_sample
s_split=10, n_estimators=2000; total time= 7.8s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1200; total time= 2.0s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=1000; total time= 1.7s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 0.5s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=800; total time= 1.1s
[CV] END bootstrap=True, max_depth=110, max_features=sqrt, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 2.0s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1200; total time= 1.8s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=2000; total time= 7.8s
[CV] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=1000; total time= 3.9s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 5.5s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=1400; total time= 2.1s
[CV] END bootstrap=False, max_depth=80, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=1000; total time= 3.5s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=10, n_estimators=1600; total time= 7.5s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=1400; total time= 2.4min
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=800; total time= 2.1s
[CV] END bootstrap=False, max_depth=90, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 2.5s
[CV] END bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 8.6s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
```

```
split=5, n_estimators=1000; total time= 1.7s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples_split=5, n_estimators=1000; total time= 3.3s
[CV] END bootstrap=False, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=800; total time= 3.6s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1600; total time= 3.0s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=800; total time= 1.5s
[CV] END bootstrap=True, max_depth=40, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=600; total time= 1.9s
[CV] END bootstrap=False, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time= 0.8s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=2000; total time= 3.7s
[CV] END bootstrap=True, max_depth=40, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=2000; total time= 4.1s
[CV] END bootstrap=True, max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_split=5, n_estimators=400; total time= 0.7s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=1200; total time= 2.3s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=1600; total time= 2.4s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=1000; total time= 1.9s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=5, n_estimators=1800; total time= 6.8s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_split=5, n_estimators=1600; total time= 2.8s
[CV] END bootstrap=False, max_depth=70, max_features=auto, min_samples_leaf=2, min_samples_split=5, n_estimators=600; total time= 2.6s
[CV] END bootstrap=False, max_depth=110, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1800; total time= 7.9s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=2, n_estimators=1800; total time= 6.1s
[CV] END bootstrap=False, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples_split=10, n_estimators=2000; total time= 8.1s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=800; total time= 3.3s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=5, n_estimators=200; total time= 0.5s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=110, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=1000; total time= 2.0s
[CV] END bootstrap=False, max_depth=110, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=600; total time= 1.0s
[CV] END bootstrap=False, max_depth=110, max_features=sqrt, min_samples_leaf=2, min_samples_split=2, n_estimators=600; total time= 1.1s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=2000; total time= 8.0s
[CV] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_split=5, n_estimators=1000; total time= 4.0s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=1, min_samples_split=2, n_estimators=1400; total time= 5.9s
[CV] END bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=1, min_samples_split=2, n_estimators=200; total time= 0.4s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_split=2, n_estimators=400; total time= 0.6s
[CV] END bootstrap=False, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_split=10, n_estimators=1200; total time= 5.5s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_split=2, n_estimators=1800; total time= 2.5min
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples
```

```
_split=2, n_estimators=1400; total time= 2.9s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1800; total time= 4.4s
[CV] END bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 8.3s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 0.7s
[CV] END bootstrap=False, max_depth=60, max_features=auto, min_samples_leaf=4, min_samples_
_split=2, n_estimators=2000; total time= 8.6s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=1000; total time= 1.8s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=2000; total time= 3.4s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 5.1s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=2000; total time= 3.8s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=800; total time= 1.3s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_samples_leaf=2, min_sample
s_split=5, n_estimators=1000; total time= 1.8s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 5.7s
[CV] END bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=800; total time= 1.5s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1600; total time= 2.8s
[CV] END bootstrap=False, max_depth=70, max_features=auto, min_samples_leaf=2, min_samples_
split=5, n_estimators=600; total time= 3.0s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=600; total time= 2.3s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=600; total time= 2.1s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 3.3s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=1400; total time= 2.6s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1400; total time= 7.9s
[CV] END bootstrap=False, max_depth=50, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=800; total time= 1.7s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=5, n_estimators=800; total time= 1.5s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1800; total time= 2.7s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=5, n_estimators=800; total time= 2.6s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=600; total time= 0.9s
[CV] END bootstrap=False, max_depth=90, max_features=auto, min_samples_leaf=2, min_samples_
split=5, n_estimators=200; total time= 0.9s
[CV] END bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=2000; total time= 7.1s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=1000; total time= 3.6s
[CV] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=1000; total time= 3.8s
[CV] END bootstrap=False, max_depth=60, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=600; total time= 0.9s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=1800; total time= 6.3s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=400; total time= 0.7s
[CV] END bootstrap=False, max_depth=80, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=1000; total time= 4.3s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
```

```
s_split=10, n_estimators=1600; total time= 2.4min
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1000; total time= 2.7s
[CV] END bootstrap=False, max_depth=60, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=2000; total time= 11.8s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=2000; total time= 4.9s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 0.7s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=600; total time= 2.1s
[CV] END bootstrap=False, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=800; total time= 3.6s
[CV] END bootstrap=False, max_depth=40, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=600; total time= 1.1s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 0.8s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=800; total time= 1.4s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=1000; total time= 1.8s
[CV] END bootstrap=True, max_depth=40, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=600; total time= 2.0s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=400; total time= 1.6s
[CV] END bootstrap=False, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 1.0s
[CV] END bootstrap=True, max_depth=40, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=2000; total time= 5.5s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1200; total time= 2.2s
[CV] END bootstrap=False, max_depth=30, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=2000; total time= 8.2s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 1.3s
[CV] END bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=800; total time= 1.6s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1600; total time= 2.7s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=1800; total time= 7.1s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1800; total time= 3.1s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 3.2s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_
split=2, n_estimators=400; total time= 0.9s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1400; total time= 7.7s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=1200; total time= 2.0s
[CV] END bootstrap=False, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=800; total time= 1.5s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=800; total time= 3.2s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 0.6s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=600; total time= 0.9s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 0.6s
[CV] END bootstrap=True, max_depth=90, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=2000; total time= 7.0s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=4, min_samples_
```

```
_split=10, n_estimators=1000; total time= 3.6s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=4, min_samples_
_split=10, n_estimators=1000; total time= 4.0s
[CV] END bootstrap=False, max_depth=60, max_features=sqrt, min_samples_leaf=4, min_samples_
_split=2, n_estimators=600; total time= 0.9s
[CV] END bootstrap=True, max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=1600; total time= 5.3s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=1400; total time= 2.1s
[CV] END bootstrap=False, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=1200; total time= 5.9s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=1800; total time= 2.5min
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1400; total time= 2.8s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1800; total time= 4.6s
[CV] END bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=400; total time= 1.1s
[CV] END bootstrap=False, max_depth=90, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=400; total time= 1.5s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=10, n_estimators=1600; total time= 4.0s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1000; total time= 1.9s
[CV] END bootstrap=False, max_depth=60, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=2000; total time= 8.3s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=1000; total time= 1.9s
[CV] END bootstrap=True, max_depth=110, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=600; total time= 1.0s
[CV] END bootstrap=True, max_depth=40, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=600; total time= 1.9s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=400; total time= 1.9s
[CV] END bootstrap=True, max_depth=40, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=2000; total time= 5.6s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1200; total time= 2.2s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1600; total time= 2.5s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=800; total time= 1.3s
[CV] END bootstrap=False, max_depth=60, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=600; total time= 1.1s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 5.8s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=2000; total time= 3.3s
[CV] END bootstrap=False, max_depth=70, max_features=auto, min_samples_leaf=2, min_samples_
split=5, n_estimators=600; total time= 2.8s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 2.3s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=600; total time= 2.0s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1800; total time= 3.2s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=1400; total time= 2.3s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=2, n_estimators=400; total time= 0.9s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=1, min_sample
s_split=2, n_estimators=400; total time= 0.9s
[CV] END bootstrap=False, max_depth=40, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1400; total time= 8.3s
[CV] END bootstrap=False, max_depth=50, max_features=sqrt, min_samples_leaf=2, min_samples
```

```
_split=2, n_estimators=800; total time= 1.5s
[CV] END bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=800; total time= 1.3s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1200; total time= 1.9s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=800; total time= 3.1s
[CV] END bootstrap=True, max_depth=90, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 0.6s
[CV] END bootstrap=False, max_depth=90, max_features=auto, min_samples_leaf=2, min_samples_
split=5, n_estimators=200; total time= 0.9s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 0.6s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1200; total time= 1.8s
[CV] END bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1800; total time= 10.3s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1400; total time= 2.3s
[CV] END bootstrap=True, max_depth=80, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=1800; total time= 6.2s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=1400; total time= 2.2s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1600; total time= 2.4s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=1600; total time= 7.9s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1000; total time= 2.4min
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1000; total time= 2.0s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=800; total time= 1.5s
[CV] END bootstrap=False, max_depth=20, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=800; total time= 2.1s
[CV] END bootstrap=False, max_depth=90, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 2.4s
[CV] END bootstrap=False, max_depth=90, max_features=auto, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 3.0s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1600; total time= 4.2s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=2000; total time= 4.2s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=1000; total time= 3.1s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=800; total time= 1.3s
[CV] END bootstrap=False, max_depth=40, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=600; total time= 1.0s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 0.8s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=800; total time= 1.4s
[CV] END bootstrap=True, max_depth=110, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=600; total time= 1.2s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=800; total time= 1.6s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=600; total time= 1.0s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=600; total time= 0.9s
[CV] END bootstrap=False, max_depth=100, max_features=auto, min_samples_leaf=4, min_samples_
split=5, n_estimators=200; total time= 0.9s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=2, n_estimators=2000; total time= 3.8s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_
```

```
split=5, n_estimators=2000; total time= 3.6s
[CV] END bootstrap=True, max_depth=10, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1600; total time= 2.3s
[CV] END bootstrap=False, max_depth=60, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=600; total time= 1.0s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1000; total time= 5.9s
[CV] END bootstrap=False, max_depth=10, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=2000; total time= 3.1s
[CV] END bootstrap=False, max_depth=50, max_features=auto, min_samples_leaf=4, min_samples_
split=2, n_estimators=1800; total time= 7.2s
[CV] END bootstrap=False, max_depth=30, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1800; total time= 3.4s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=1400; total time= 2.3s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=1400; total time= 2.6s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1000; total time= 1.8s
[CV] END bootstrap=False, max_depth=100, max_features=auto, min_samples_leaf=4, min_sample
s_split=10, n_estimators=2000; total time= 7.6s
[CV] END bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=800; total time= 1.2s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=2, min_samples_
split=5, n_estimators=1200; total time= 2.0s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=2, min_sample
s_split=5, n_estimators=800; total time= 2.6s
[CV] END bootstrap=True, max_depth=60, max_features=sqrt, min_samples_leaf=2, min_samples_
split=10, n_estimators=600; total time= 0.9s
[CV] END bootstrap=False, max_depth=90, max_features=auto, min_samples_leaf=2, min_samples_
split=5, n_estimators=200; total time= 0.9s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 0.6s
[CV] END bootstrap=False, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=10, n_estimators=400; total time= 0.5s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1200; total time= 1.7s
[CV] END bootstrap=False, max_depth=20, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1800; total time= 10.4s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=2, min_sampl
es_split=5, n_estimators=1400; total time= 2.5s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=1, min_samples_
split=2, n_estimators=1400; total time= 5.8s
[CV] END bootstrap=True, max_depth=50, max_features=sqrt, min_samples_leaf=1, min_samples_
split=2, n_estimators=200; total time= 0.5s
[CV] END bootstrap=False, max_depth=80, max_features=auto, min_samples_leaf=4, min_samples_
split=10, n_estimators=1000; total time= 3.7s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1, min_samples_
split=10, n_estimators=1600; total time= 4.4s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=2, min_samples_
split=2, n_estimators=1400; total time= 2.4min
[CV] END bootstrap=False, max_depth=60, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=2000; total time= 12.3s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=2000; total time= 4.7s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=2, min_samples_
split=10, n_estimators=600; total time= 2.0s
[CV] END bootstrap=False, max_depth=None, max_features=auto, min_samples_leaf=2, min_sampl
es_split=10, n_estimators=800; total time= 3.3s
[CV] END bootstrap=False, max_depth=70, max_features=sqrt, min_samples_leaf=4, min_samples_
split=5, n_estimators=800; total time= 1.3s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1, min_samples_
split=5, n_estimators=400; total time= 0.8s
[CV] END bootstrap=True, max_depth=80, max_features=sqrt, min_samples_leaf=4, min_samples_
split=2, n_estimators=1600; total time= 3.1s
[CV] END bootstrap=False, max_depth=None, max_features=sqrt, min_samples_leaf=4, min_sampl
```

```
es_split=5, n_estimators=2000; total time= 3.4s
[CV] END bootstrap=False, max_depth=10, max_features=auto, min_samples_leaf=1, min_samples_
_split=2, n_estimators=1000; total time= 5.1s
Out[266... RandomizedSearchCV(estimator=GradientBoostingRegressor(), n_iter=20, n_jobs=2,
                                param_distributions={'learning_rate': <scipy.stats._distn_infrastructur
e.rv_frozen object at 0x7fb52cc0baf0>,
                                         'max_leaf_nodes': [2, 5, 10, 20, 50,
                                         100],
                                         'n_estimators': [1, 2, 5, 10, 20, 50,
                                         100, 200, 500]},
                                         random_state=42, scoring='neg_mean_absolute_error')
```

```
In [266... random_cv.best_params_
```

```
Out[266... {'learning_rate': 0.06170217483403151,
            'max_leaf_nodes': 20,
            'n_estimators': 200}
```

```
In [266... model_gb=random_cv.best_estimator_
model_gb.fit(X_train,y_train)
pred=model_gb.predict(X_test)
r_score=r2_score(y_test,pred)
print('model score is : {}'.format(r_score))
```

```
model score is : 0.9177315048254759
```

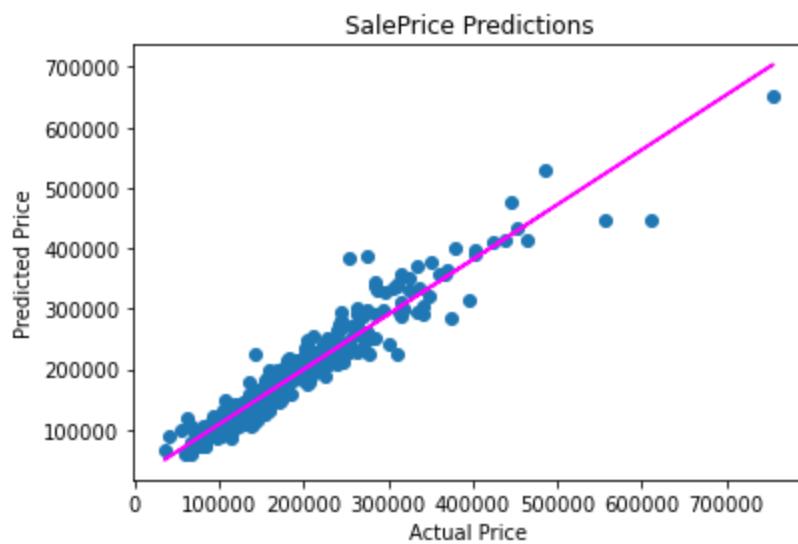
```
In [267... mae=mean_absolute_error(pred,y_test)
print('Mean absolute error is : {}'.format(mae))
```

```
Mean absolute error is : 15759.847600190216
```

We got the best score from GradientBoostingRegressor after tuning hyperparameter at 91.44% and Mean absolute error at 15724.85 that conclusion that the prediction prices from this model can have the average error around +/- 15724.85 from actual housing prices.

Fit linear line model to visualize model fit to dataset

```
In [267... plt.scatter(y_test, pred)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('SalePrice Predictions')
z = np.polyfit(y_test, pred, 1)
p = np.poly1d(z)
plt.plot(y_test,p(y_test), color='magenta')
plt.show()
```



Save model and use model for predict df_test dataset

In [267...]

```
import joblib
filename = './SalePrice.pkl'
joblib.dump(model_gb, filename)
```

Out[267...]

```
['./SalePrice.pkl']
```

In [267...]

```
loaded_model = joblib.load(filename)
predict_data=df_test

result = loaded_model.predict(predict_data)
np.round(result[1:10],2)
```

Out[267...]

```
array([174419.48, 164961.46, 191638.79, 186202.93, 163503.83, 178241.23,
       172441.81, 163785.16, 107116.81])
```