house_price_prediction

September 26, 2023

1 House Price Prediction

1.0.1 Motivation

- I am always facinated by pricing prediction. Real estate market has interesting stories and events. I believe this is going to be an interesting and challenging project to be able to predict the prices correctly.
- I want to find out how accurately we can model the problem and see how we can predict.

1.0.2 Objective

- The objective of the project is to utilize advanced predictive modeling techniques to analyze historical and current data on property prices.
- The aim is to forecast future trends in the housing market, thereby providing invaluable insights to stakeholders.

1.0.3 Data

- We will be using Kaggle House Prices dataset, https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/overview.
- The dataset has 81 features that cover a wide range of attributes like square footage, neighborhood, quality of materials, and many more
- The objective is to build a robust predictive model that leverages the 81 features to accurately predict house prices. Special attention will be given to feature selection and engineering, as well as evaluating various machine learning algorithms to arrive at a model that minimizes error rates.

[651]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
↪GridSearchCV
from sklearn.metrics import mean_squared_error
from scipy.stats import kurtosis, skew

from sklearn.preprocessing import LabelEncoder
```

```
import xgboost as xgb

from catboost import CatBoost, CatBoostRegressor, Pool

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[493]:
```
import warnings
warnings.filterwarnings('ignore')
```

[494]:
```
def correlation_matrix_plot(correlation_matrix):
    sns.set_theme(style="white")
    mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))

    f, ax = plt.subplots(figsize=(15, 12))
    cmap = sns.diverging_palette(230, 20, as_cmap=True)

    sns.heatmap(correlation_matrix, mask=mask, cmap=cmap, vmax=.3, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

2 Data Cleaning

[495]:
```
train_df = pd.read_csv('data/train.csv')
train_df.head(10)
```

[495]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	\
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	
5	6	50	RL	85.0	14115	Pave	NaN	IR1	
6	7	20	RL	75.0	10084	Pave	NaN	Reg	
7	8	60	RL	NaN	10382	Pave	NaN	IR1	
8	9	50	RM	51.0	6120	Pave	NaN	Reg	
9	10	190	RL	50.0	7420	Pave	NaN	Reg	

	LandContour	Utilities	…	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	\
0	Lvl	AllPub	…	0	NaN	NaN	NaN	0	
1	Lvl	AllPub	…	0	NaN	NaN	NaN	0	
2	Lvl	AllPub	…	0	NaN	NaN	NaN	0	
3	Lvl	AllPub	…	0	NaN	NaN	NaN	0	
4	Lvl	AllPub	…	0	NaN	NaN	NaN	0	
5	Lvl	AllPub	…	0	NaN	MnPrv	Shed	700	
6	Lvl	AllPub	…	0	NaN	NaN	NaN	0	
7	Lvl	AllPub	…	0	NaN	NaN	Shed	350	
8	Lvl	AllPub	…	0	NaN	NaN	NaN	0	

```
9         Lvl     AllPub  ...         0     NaN     NaN          NaN        0

  MoSold YrSold  SaleType  SaleCondition  SalePrice
0      2   2008        WD         Normal     208500
1      5   2007        WD         Normal     181500
2      9   2008        WD         Normal     223500
3      2   2006        WD        Abnorml     140000
4     12   2008        WD         Normal     250000
5     10   2009        WD         Normal     143000
6      8   2007        WD         Normal     307000
7     11   2009        WD         Normal     200000
8      4   2008        WD        Abnorml     129900
9      1   2008        WD         Normal     118000

[10 rows x 81 columns]
```

[496]:

```
train_df_shape = train_df.shape
print(f'Total number of samples {train_df_shape[0]} and total number of␣
↪features {train_df_shape[1]}')
```

```
Total number of samples 1460 and total number of features 81
```

[497]:

```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
```

```
19  YearBuilt      1460 non-null   int64
20  YearRemodAdd   1460 non-null   int64
21  RoofStyle      1460 non-null   object
22  RoofMatl       1460 non-null   object
23  Exterior1st    1460 non-null   object
24  Exterior2nd    1460 non-null   object
25  MasVnrType     588 non-null    object
26  MasVnrArea     1452 non-null   float64
27  ExterQual      1460 non-null   object
28  ExterCond      1460 non-null   object
29  Foundation     1460 non-null   object
30  BsmtQual       1423 non-null   object
31  BsmtCond       1423 non-null   object
32  BsmtExposure   1422 non-null   object
33  BsmtFinType1   1423 non-null   object
34  BsmtFinSF1     1460 non-null   int64
35  BsmtFinType2   1422 non-null   object
36  BsmtFinSF2     1460 non-null   int64
37  BsmtUnfSF      1460 non-null   int64
38  TotalBsmtSF    1460 non-null   int64
39  Heating        1460 non-null   object
40  HeatingQC      1460 non-null   object
41  CentralAir     1460 non-null   object
42  Electrical     1459 non-null   object
43  1stFlrSF       1460 non-null   int64
44  2ndFlrSF       1460 non-null   int64
45  LowQualFinSF   1460 non-null   int64
46  GrLivArea      1460 non-null   int64
47  BsmtFullBath   1460 non-null   int64
48  BsmtHalfBath   1460 non-null   int64
49  FullBath       1460 non-null   int64
50  HalfBath       1460 non-null   int64
51  BedroomAbvGr   1460 non-null   int64
52  KitchenAbvGr   1460 non-null   int64
53  KitchenQual    1460 non-null   object
54  TotRmsAbvGrd   1460 non-null   int64
55  Functional     1460 non-null   object
56  Fireplaces     1460 non-null   int64
57  FireplaceQu    770 non-null    object
58  GarageType     1379 non-null   object
59  GarageYrBlt    1379 non-null   float64
60  GarageFinish   1379 non-null   object
61  GarageCars     1460 non-null   int64
62  GarageArea     1460 non-null   int64
63  GarageQual     1379 non-null   object
64  GarageCond     1379 non-null   object
65  PavedDrive     1460 non-null   object
66  WoodDeckSF     1460 non-null   int64
```

```
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

[498]:
```
numerical_cols = train_df.select_dtypes(include = ['float', 'int64']).columns
nominal_cols = train_df.select_dtypes(include = ['object']).columns

print(f'Number of Numerical data features {len(numerical_cols)}')
print(f'Number of Nominal data features {len(nominal_cols)}')
```

```
Number of Numerical data features 38
Number of Nominal data features 43
```

2.1 Handling Missing Value

[499]:
```
def find_null_value(df):
    total = df.isnull().sum().sort_values(ascending=False)
    percent = (df.isnull().sum() / df.isnull().count()).
 ↪sort_values(ascending=False)
    missing_data = pd.concat([total, percent], axis=1, keys=['Total',␣
 ↪'Percent'])
    return missing_data
```

[500]:
```
missing_data = find_null_value(train_df)
missing_data.head(20)
```

[500]:

	Total	Percent
PoolQC	1453	0.995205
MiscFeature	1406	0.963014
Alley	1369	0.937671
Fence	1179	0.807534
MasVnrType	872	0.597260
FireplaceQu	690	0.472603
LotFrontage	259	0.177397

```
GarageYrBlt      81   0.055479
GarageCond       81   0.055479
GarageType       81   0.055479
GarageFinish     81   0.055479
GarageQual       81   0.055479
BsmtFinType2     38   0.026027
BsmtExposure     38   0.026027
BsmtQual         37   0.025342
BsmtCond         37   0.025342
BsmtFinType1     37   0.025342
MasVnrArea        8   0.005479
Electrical        1   0.000685
Id                0   0.000000
```

2.1.1 Remove features which null value percent > 80%

[501]:
```python
remove_cols = missing_data[missing_data['Percent'] > 0.8].index
print(f'{remove_cols}')
```

```
Index(['PoolQC', 'MiscFeature', 'Alley', 'Fence'], dtype='object')
```

[502]:
```python
train_df = train_df.drop(columns = remove_cols, axis = 1)
```

[503]:
```python
missing_data = find_null_value(train_df)
missing_data.head(20)
```

[503]:

	Total	Percent
MasVnrType	872	0.597260
FireplaceQu	690	0.472603
LotFrontage	259	0.177397
GarageCond	81	0.055479
GarageYrBlt	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
GarageType	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtFinType1	37	0.025342
BsmtCond	37	0.025342
BsmtQual	37	0.025342
MasVnrArea	8	0.005479
Electrical	1	0.000685
BsmtFullBath	0	0.000000
Functional	0	0.000000
TotRmsAbvGrd	0	0.000000
GrLivArea	0	0.000000
HalfBath	0	0.000000

We have remove the null values. We will impute top 2 nominal features which null values percent > 40 %.

2.1.2 Imputation

I will test different approaches to replace null values and discuss its pros and cons.

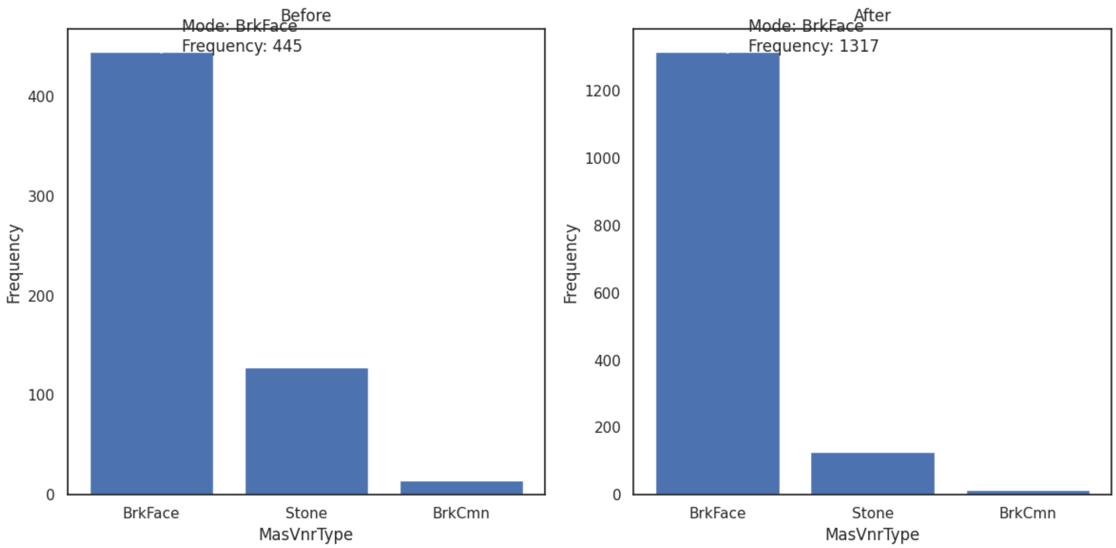
Mode Imputation

[504]:
```
def plot_mode_imputation(before_df, after_df, column):

    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    axs[0].bar(before_df[column].value_counts().index,
               before_df[column].value_counts().values)
    axs[0].set_title('Before')
    axs[0].set_xlabel(column)
    axs[0].set_ylabel('Frequency')

    before_mode = before_df[column].mode()[0]
    most_frequent = before_df[column].value_counts().iloc[0]
    axs[0].annotate(f'Mode: {before_mode}\nFrequency: {most_frequent}',
                    xy=(0, most_frequent), xytext=(0.2, most_frequent + 0.1),
                    arrowprops=dict(arrowstyle='->'))

    after_df[column].fillna(before_mode, inplace=True)

    axs[1].bar(after_df[column].value_counts().index,
               after_df[column].value_counts().values)
    axs[1].set_title('After')
    axs[1].set_xlabel(column)
    axs[1].set_ylabel('Frequency')

    after_mode = after_df[column].mode()[0]
    most_frequent = after_df[column].value_counts().iloc[0]
    print(most_frequent)
    axs[1].annotate(f'Mode: {after_mode}\nFrequency: {most_frequent}',
                    xy=(after_mode, most_frequent), xytext=(0.2, most_frequent␣
↪+ 0.1),
                    arrowprops=dict(arrowstyle='->'))

    plt.tight_layout()
    plt.show()
```

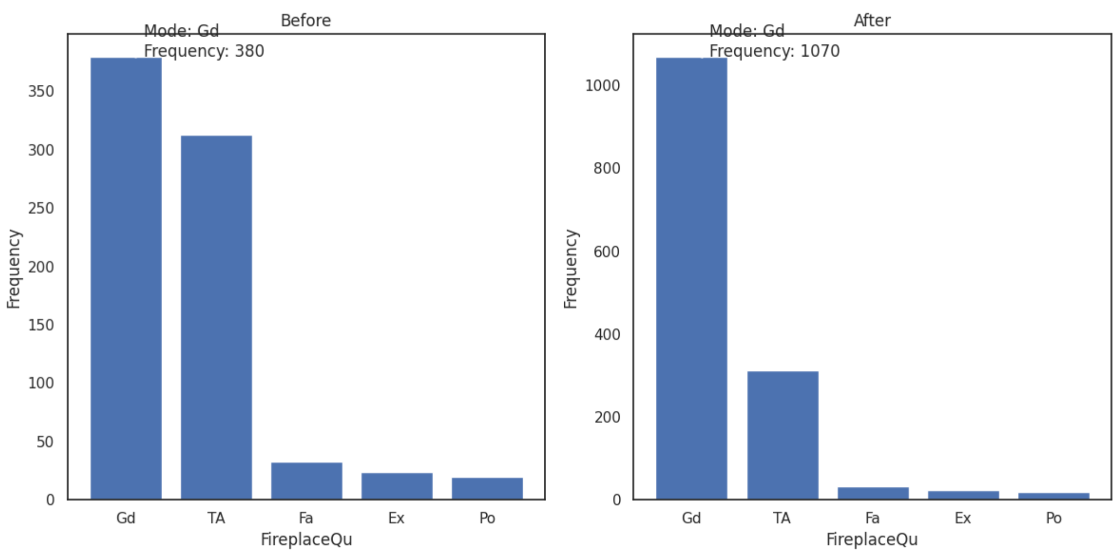
[505]:
```
train_df_clone = train_df.copy()
```

[506]:
```
plot_mode_imputation(train_df, train_df_clone, 'MasVnrType')
plot_mode_imputation(train_df, train_df_clone, 'FireplaceQu')
```

1317



1070



Pros - For categorical data where mean or median cannot be calculated, mode is a good statistical measure for central tendency. - BrkFace and Gd are the most frequent feature sample in the data. - It is straightforward, it’s also computationally inexpensive, making it feasible for large datasets.

Cons - Mode imputation might lead the data to have bias to most frequent feature value, which might lead data imbalance when we train the model. - It can hurt the variability of the data. - If the number of missing values is high, filling them all with the mode can disproportionately inflate the frequency of that category, leading to incorrect analysis or predictions.

Conclusion, I will choose to go with Random Imputation because the above reasons.

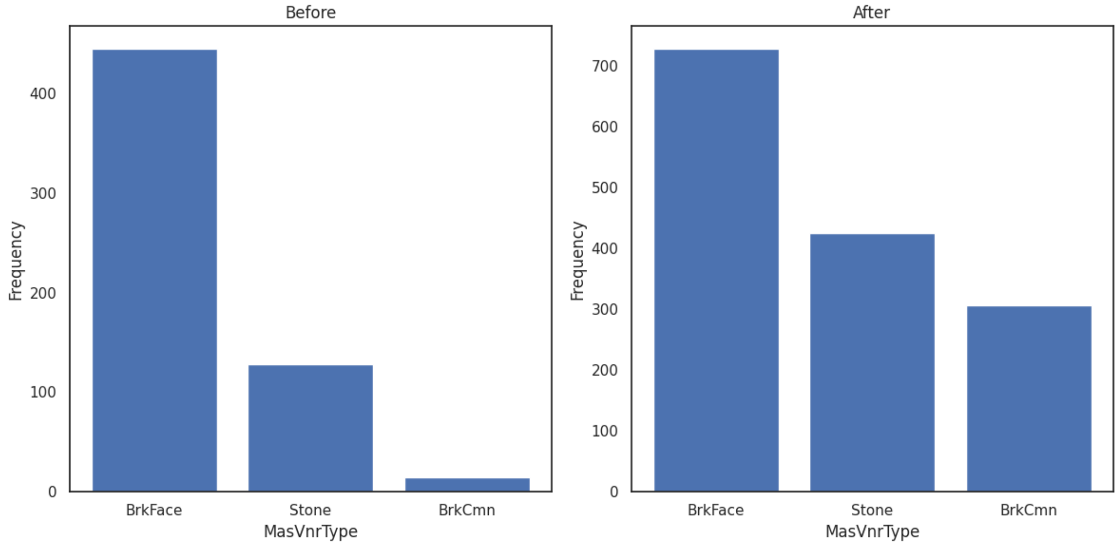
Random Imputation

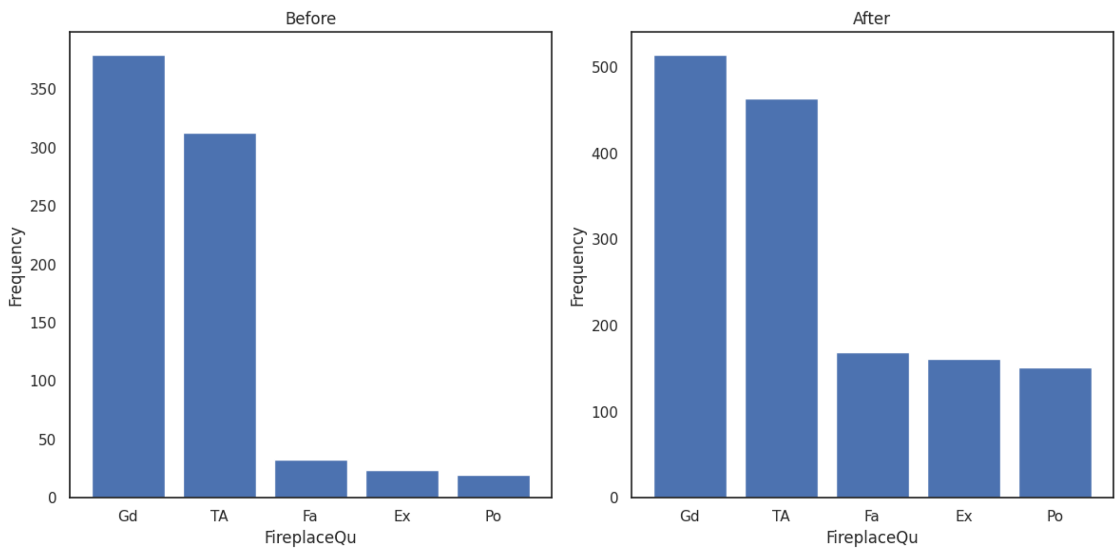
[507]:
```
def random_imputation(df, column):
    non_na_values = df[column].dropna().unique()
    na_positions = df.index[df[column].isna()].tolist()
    random_values = random.choices(non_na_values, k=len(na_positions))

    for pos, value in zip(na_positions, random_values):
        df.at[pos, column] = value
    return df
```

[508]:
```
def plot_random_imputation(before_df, after_df, column):

    fig, axs = plt.subplots(1, 2, figsize=(12, 6))

    axs[0].bar(before_df[column].value_counts().index,
               before_df[column].value_counts().values,)
    axs[0].set_title('Before')
    axs[0].set_xlabel(column)
    axs[0].set_ylabel('Frequency')

    random_imputation(after_df, column)

    axs[1].bar(after_df[column].value_counts().index,
               after_df[column].value_counts().values)
    axs[1].set_title('After')
    axs[1].set_xlabel(column)
    axs[1].set_ylabel('Frequency')

    plt.tight_layout()
    plt.show()
```

[511]:
```
train_df_clone = train_df.copy()
```

[512]:
```
plot_random_imputation(train_df, train_df_clone, 'MasVnrType')
plot_random_imputation(train_df, train_df_clone, 'FireplaceQu')
```





- Pros
  - All unique samples will be randomly selected as a fair choice.
  - It can maintain the original distribution and variance of the dataset because it uses actual observed values for imputation.
- Cons
  - Because the imputation is random, this may add noise into the dataset, especially if the missing values are not completely at random.
  - The imputation is stochastic, leading to different results every time the imputation is carried out, which might not be desirable in all scenarios.

[513]:
```
train_df = random_imputation(train_df, 'MasVnrType')
train_df = random_imputation(train_df, 'FireplaceQu')
```

[514]:
```
missing_data = find_null_value(train_df)
missing_data.head(20)
```

[514]:

	Total	Percent
LotFrontage	259	0.177397
GarageType	81	0.055479
GarageCond	81	0.055479
GarageYrBlt	81	0.055479
GarageFinish	81	0.055479
GarageQual	81	0.055479
BsmtFinType2	38	0.026027
BsmtExposure	38	0.026027
BsmtFinType1	37	0.025342
BsmtCond	37	0.025342
BsmtQual	37	0.025342
MasVnrArea	8	0.005479
Electrical	1	0.000685
WoodDeckSF	0	0.000000
KitchenAbvGr	0	0.000000
LowQualFinSF	0	0.000000
GrLivArea	0	0.000000
BsmtFullBath	0	0.000000
BsmtHalfBath	0	0.000000
FullBath	0	0.000000

Median Imputation I will use Median Imputation which is more robust to outliers.

[515]:
```
train_df['LotFrontage'].fillna(train_df['LotFrontage'].median(), inplace=True)
```

Drop NA

[516]:
```
train_df = train_df.dropna()
```

[517]:
```
missing_data = find_null_value(train_df)
missing_data.head(20)
```

[517]:

	Total	Percent
Id	0	0.0
HalfBath	0	0.0
FireplaceQu	0	0.0
Fireplaces	0	0.0
Functional	0	0.0
TotRmsAbvGrd	0	0.0
KitchenQual	0	0.0

```
KitchenAbvGr       0        0.0
BedroomAbvGr       0        0.0
FullBath           0        0.0
HeatingQC          0        0.0
BsmtHalfBath       0        0.0
BsmtFullBath       0        0.0
GrLivArea          0        0.0
LowQualFinSF       0        0.0
2ndFlrSF           0        0.0
1stFlrSF           0        0.0
Electrical         0        0.0
GarageType         0        0.0
GarageYrBlt        0        0.0
```

Drop Id

[518]:
```
train_df = train_df.drop(columns = ['Id'], axis = 1)
```

[519]:
```
print(f'Total number of missing data in the dataset {train_df.isnull().sum().
 ↪max()}')
```

```
Total number of missing data in the dataset 0
```

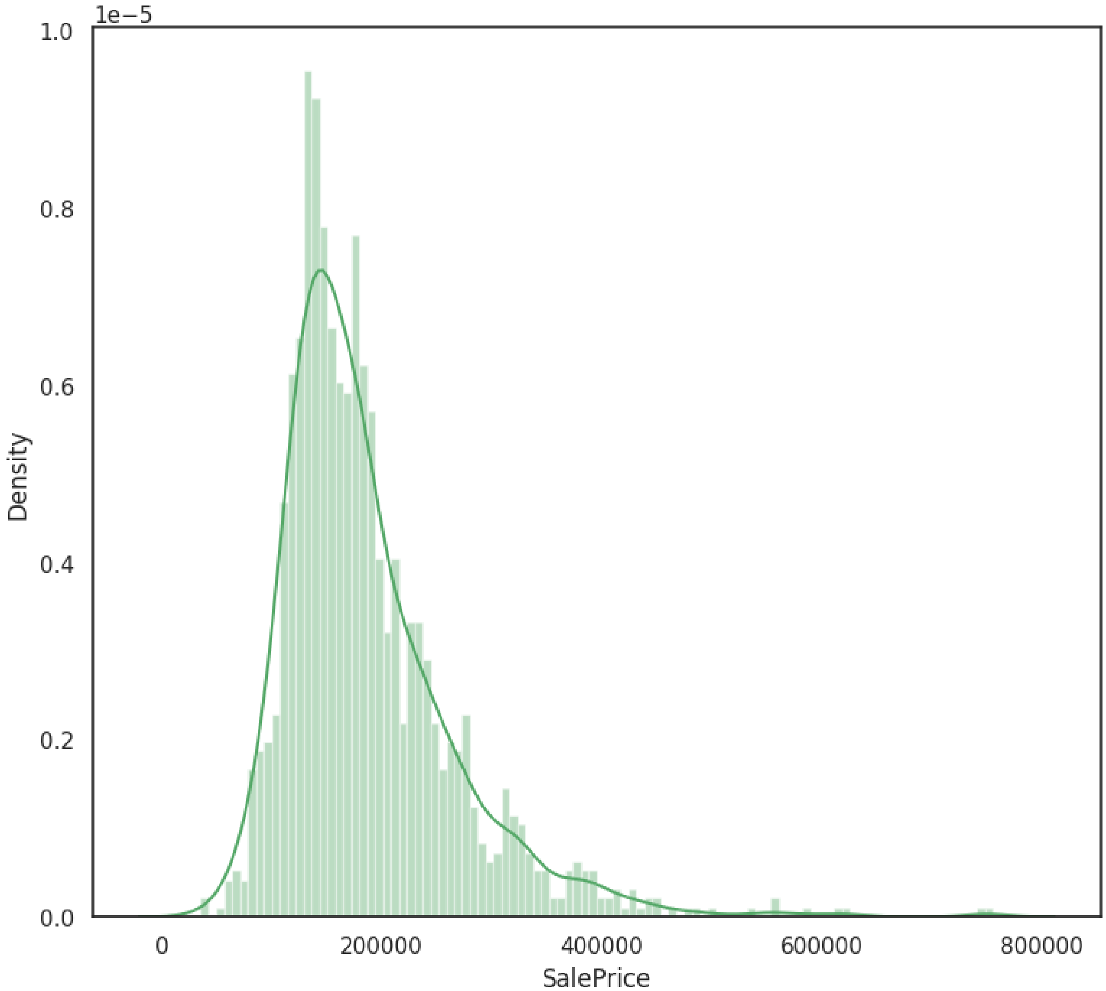
3 Exploratory Data Analysis (EDA)

3.0.1 Target Data Distribution (Sale Price Data Distribution)

We will see its distribution and the outliers through the graph as well as from Skewiness and Kurtosis

[521]:
```
print(train_df.SalePrice.describe())
plt.figure(figsize=(9, 8))
sns.distplot(train_df.SalePrice, color='g', bins=100, hist_kws={'alpha': 0.4});

print(f' median sale price according to the dataset {np.median(train_df.
 ↪SalePrice.values)}')
```

```
count      1338.000000
mean     186761.782511
std       78913.847668
min       35311.000000
25%      135000.000000
50%      168500.000000
75%      220000.000000
max      755000.000000
Name: SalePrice, dtype: float64
 median sale price according to the dataset 168500.0
```



Skewiness and Kurtosis

[522]:
```
print(f' Skewiness {np.round(skew(train_df.SalePrice), 2)}')
print(f' Kurtosis {np.round(kurtosis(train_df.SalePrice), 2)}')
```

```
Skewiness 1.94
Kurtosis 6.79
```

- Skewness of 1.94: The data distribution is significantly skewed to the right. It indicates that the tail on the right side of the distribution is long towards the lower end of the distribution.
- Kurtosis of 6.79: It indicates that the data distribution has heavier tails and a sharper peak than a normal distribution. There might be more extreme values in the dataset than a normally distributed dataset.

3.1 Find Correlation

It is important to understand the correlation between features and target. It is unlikely to need all the features we have for the prediction the target value.

[523]:
```
numerical_cols = train_df.select_dtypes(include = ['float', 'int64']).columns
```

3.1.1 Variant Inflation Factor (VIF)

[524]:
```
X = train_df[numerical_cols]
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.
    ↪columns))]
print(vif_data.sort_values(by = 'VIF'))
```

```
          feature           VIF
33        MiscVal  1.034466e+00
30      3SsnPorch  1.041128e+00
32       PoolArea  1.121488e+00
31    ScreenPorch  1.210673e+00
17   BsmtHalfBath  1.235575e+00
29  EnclosedPorch  1.446397e+00
28    OpenPorchSF  1.901133e+00
7      MasVnrArea  1.930214e+00
27     WoodDeckSF  1.958869e+00
2         LotArea  2.626542e+00
23     Fireplaces  3.148739e+00
19       HalfBath  3.689704e+00
16   BsmtFullBath  3.855628e+00
0      MSSubClass  4.861169e+00
34         MoSold  6.704515e+00
1     LotFrontage  1.763438e+01
18       FullBath  2.871363e+01
26     GarageArea  3.421433e+01
36      SalePrice  3.452003e+01
20   BedroomAbvGr  3.471463e+01
25     GarageCars  4.113357e+01
4     OverallCond  4.726799e+01
21   KitchenAbvGr  5.313573e+01
3     OverallQual  8.347544e+01
22   TotRmsAbvGrd  8.528044e+01
6    YearRemodAdd  2.474417e+04
5       YearBuilt  2.700007e+04
35         YrSold  2.734012e+04
24    GarageYrBlt  2.870470e+04
13       2ndFlrSF           inf
11    TotalBsmtSF           inf
9      BsmtFinSF2           inf
10      BsmtUnfSF           inf
15      GrLivArea           inf
12       1stFlrSF           inf
```

```
8      BsmtFinSF1           inf
14   LowQualFinSF           inf
```

The last 8 features with "inf" VIF are near perfectly collinear with each other. For example, `TotalBsmtSF` might be the sum of `BsmtFinSF1`, `BsmtFinSF2`, and `BsmtUnfSF`, leading to perfect collinearity. So, we need to remove one some of them to break collinearlity. We will drop some similiar columns to reduce collinearlity.

[525]:
```
train_df = train_df.drop(columns = ['2ndFlrSF', 'BsmtFinSF2', 'BsmtUnfSF',␣
 ↪'1stFlrSF', 'BsmtFinSF1', 'GarageYrBlt', 'YrSold', 'YearRemodAdd'], axis = 1)
```

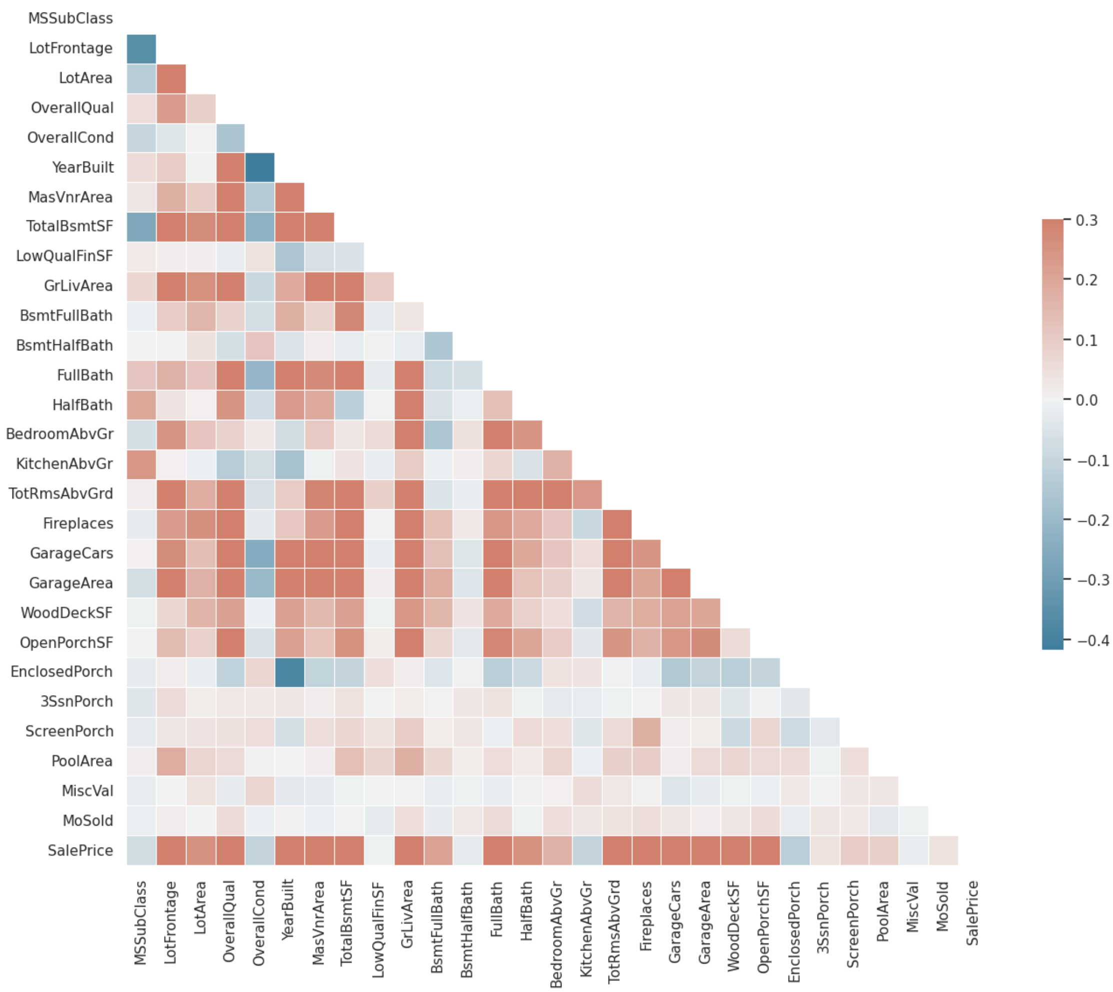
[526]:
```
X = train_df.select_dtypes(include = ['float', 'int64'])
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.
 ↪columns))]
print(vif_data.sort_values(by = 'VIF'))
```

```
          feature         VIF
26        MiscVal    1.028216
23      3SsnPorch    1.038339
8    LowQualFinSF    1.083549
25       PoolArea    1.111718
11   BsmtHalfBath    1.145886
24    ScreenPorch    1.180588
22  EnclosedPorch    1.278575
6      MasVnrArea    1.873677
21    OpenPorchSF    1.883200
20     WoodDeckSF    1.929193
10   BsmtFullBath    2.286867
2         LotArea    2.581617
13       HalfBath    2.912953
17     Fireplaces    2.956167
0      MSSubClass    4.758017
27         MoSold    6.698663
1     LotFrontage   17.349513
7     TotalBsmtSF   23.104223
12       FullBath   23.299146
19     GarageArea   31.193014
4     OverallCond   32.654841
28      SalePrice   32.980410
14   BedroomAbvGr   33.041393
18     GarageCars   40.016892
15   KitchenAbvGr   50.211861
9       GrLivArea   67.231348
3     OverallQual   78.184658
16   TotRmsAbvGrd   84.253237
5       YearBuilt  147.374043
```

3.1.2 Correlation Matrix

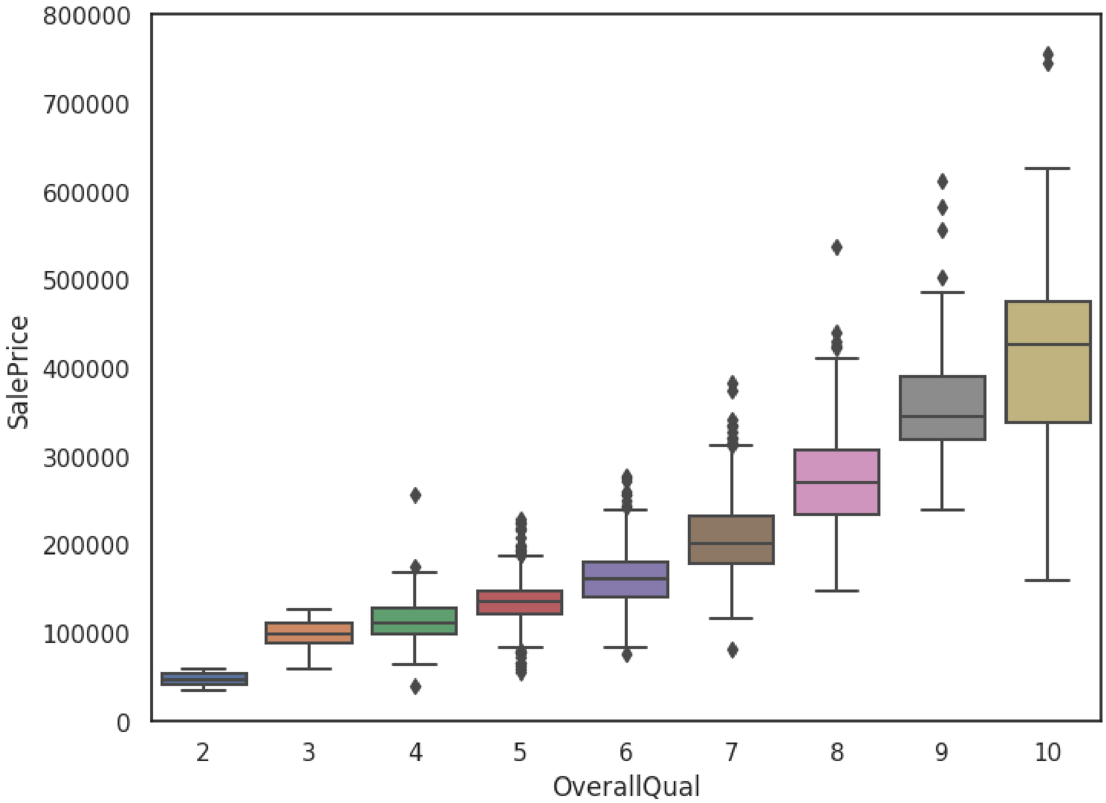
[527]:
```
correlation_matrix = train_df.select_dtypes(include = ['float', 'int64']).corr()
print(correlation_matrix['SalePrice'].sort_values(ascending=False))
correlation_matrix_plot(correlation_matrix)
```

```
SalePrice        1.000000
OverallQual      0.783546
GrLivArea        0.711706
GarageCars       0.640154
GarageArea       0.607535
TotalBsmtSF      0.602042
FullBath         0.569313
TotRmsAbvGrd     0.551821
YearBuilt        0.504297
MasVnrArea       0.465811
Fireplaces       0.445434
LotFrontage      0.327835
OpenPorchSF      0.322786
WoodDeckSF       0.305983
HalfBath         0.258175
LotArea          0.254757
BsmtFullBath     0.209695
BedroomAbvGr     0.169266
ScreenPorch      0.096624
PoolArea         0.091881
3SsnPorch        0.042159
MoSold           0.041310
LowQualFinSF    -0.009992
MiscVal         -0.016990
BsmtHalfBath    -0.030175
MSSubClass      -0.079599
OverallCond     -0.108627
KitchenAbvGr    -0.111408
EnclosedPorch   -0.127385
Name: SalePrice, dtype: float64
```



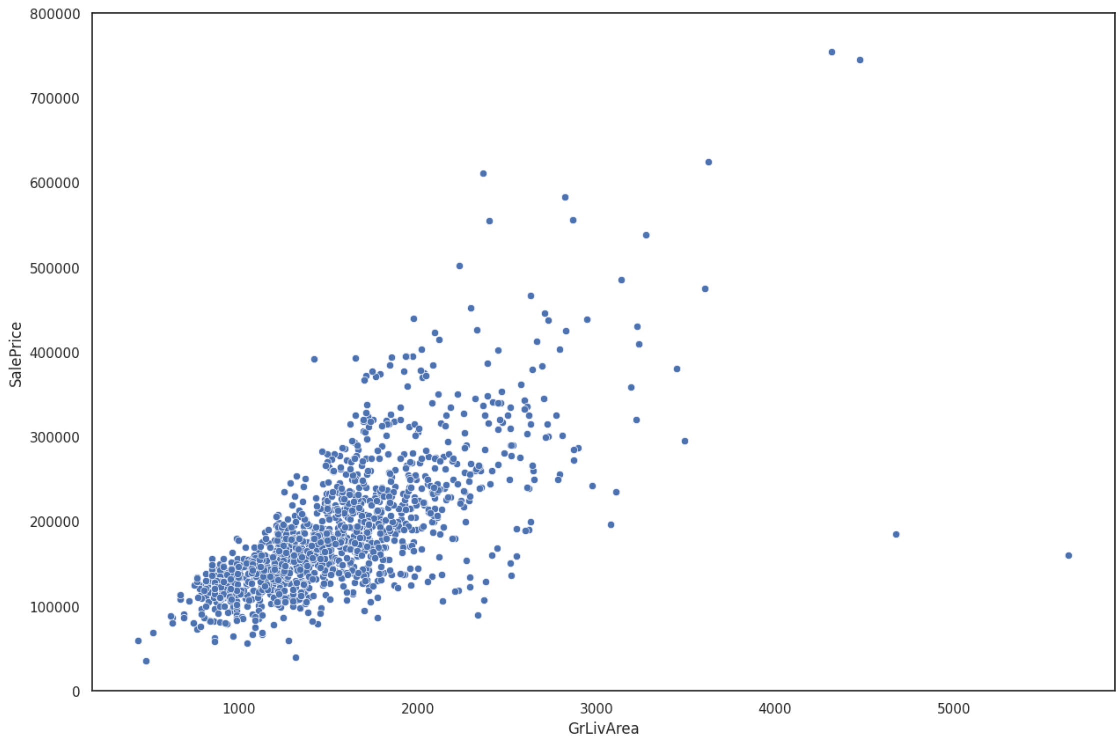
We can see that there are some features like `OverallQual`, `GrLivArea`, `GarageCars`, `GarageArea`, `TotalBsmtSF` has strong positive correlation with SalePrice.

[528]:
```python
var = 'OverallQual'
data = pd.concat([train_df['SalePrice'], train_df[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



OverallQual and SalePrice seem to have linear correlation. It seems that higher quality demands higher prices.

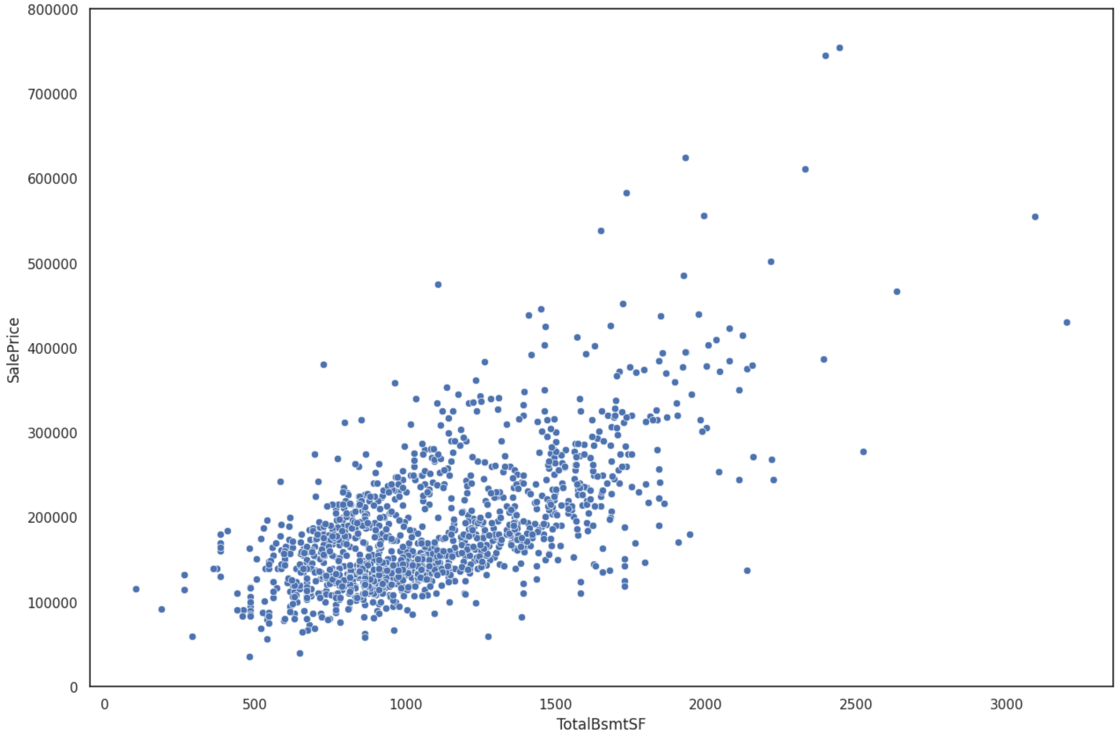
[529]:
```
var = 'GrLivArea'
data = pd.concat([train_df['SalePrice'], train_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 10))
fig = sns.scatterplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



GrLivArea and SalePrice seem to have linear relationship but there are outliers at the bottom right. There are outliers which suggest largest GrLivArea with low prices. We can remove them.

[530]:
```
train_df = train_df.drop(train_df[(train_df['GrLivArea']>4000) &␣
↪(train_df['SalePrice']<300000)].index)
```

[531]:
```
var = 'TotalBsmtSF'
data = pd.concat([train_df['SalePrice'], train_df[var]], axis=1)
f, ax = plt.subplots(figsize=(15, 10))
fig = sns.scatterplot(x = var, y = "SalePrice", data = data)
fig.axis(ymin = 0, ymax = 800000);
```



3.1.3 Chi-squared test

I will use a statistical test, chi-squared tests to for categorial features.

[532]:
```
def chi2_test(df):
    X = df[df.select_dtypes(include = ['object']).columns]

    chi2_data = pd.DataFrame()
    chi2_data["feature"] = X.columns
    chi2s = []
    pvalues = []
    for col in X.columns:
        # For categorical feature
        chi2, p_value, _, _ = stats.chi2_contingency(pd.crosstab(df[col],␣
 ↪df['SalePrice']))

        chi2s.append(chi2)
        pvalues.append(p_value)

    chi2_data['chi2'] = chi2s
    chi2_data['p_value'] = pvalues
    return chi2_data
```

[533]:
```python
chi2data = chi2_test(train_df)
print(chi2data.sort_values(by = 'p_value'))
```

```
          feature          chi2       p_value
17      ExterQual   2811.486875  1.294357e-42
37       SaleType   6386.934740  1.067397e-40
28     Electrical   3423.789697  4.270613e-34
21       BsmtCond   2653.607777  1.604465e-31
38  SaleCondition   4065.333763  6.030287e-30
1          Street   1088.427302  1.878083e-28
0        MSZoning   3276.414434  1.221275e-25
29    KitchenQual   2502.096429  3.361517e-22
20       BsmtQual   2496.248240  7.214869e-22
34     GarageQual   3031.992059  4.999571e-14
2        LotShape   2329.826900  2.219481e-13
33   GarageFinish   1548.351856  2.725970e-09
7    Neighborhood  15694.784597  4.342533e-07
22   BsmtExposure   2162.762636  7.021146e-07
27     CentralAir    779.355259  1.020573e-05
19     Foundation   2662.532706  3.994538e-03
32     GarageType   3225.220808  4.418462e-02
5       LotConfig   2584.013322  5.716175e-02
14    Exterior1st   8173.908469  1.350918e-01
16     MasVnrType   1285.856431  1.579424e-01
25        Heating   1883.884503  3.088582e-01
3     LandContour   1864.807573  4.254058e-01
11     HouseStyle   4301.299244  6.021399e-01
6       LandSlope   1210.727292  6.908929e-01
31    FireplaceQu   2380.592378  9.043710e-01
9      Condition2   4161.107640  9.631585e-01
23   BsmtFinType1   2948.777393  9.653176e-01
26      HeatingQC   2299.034333  9.939671e-01
18      ExterCond   1700.233233  9.951670e-01
15    Exterior2nd   8889.192245  9.976803e-01
35     GarageCond   2214.156382  9.999257e-01
36     PavedDrive   1053.710015  9.999410e-01
10       BldgType   2188.463916  9.999859e-01
24   BsmtFinType2   2679.205844  1.000000e+00
12      RoofStyle   2549.501436  1.000000e+00
30     Functional   2747.202868  1.000000e+00
8      Condition1   3885.476519  1.000000e+00
13       RoofMatl   2794.703218  1.000000e+00
4       Utilities    221.832709  1.000000e+00
```

I will remove the features which has p_value is closer to 1 because which is unlikely to have relation with the target variable.

[534]:
```python
cols_to_remove = chi2_data[chi2_data['p_value'] > 0.65]['feature']
```

[535]:
```python
print(f'These are the features which have p_value closer to 1 {cols_to_remove.
    ↪values}')
```

```
These are the features which have p_value closer to 1 ['Utilities' 'LandSlope'
'Condition1' 'Condition2' 'BldgType' 'RoofStyle'
 'RoofMatl' 'Exterior2nd' 'ExterCond' 'BsmtFinType1' 'BsmtFinType2'
 'HeatingQC' 'Functional' 'GarageCond' 'PavedDrive']
```

[536]:
```python
train_df = train_df.drop(columns = cols_to_remove, axis = 1)
```

[537]:
```python
train_df.columns
```

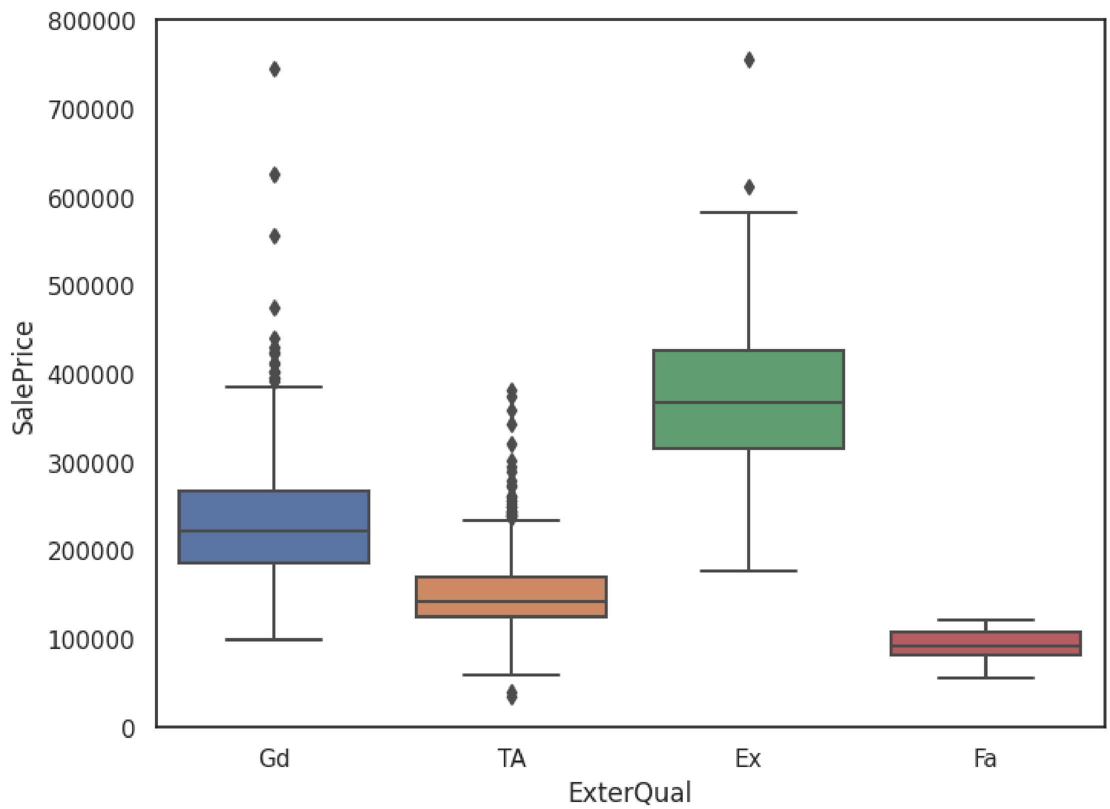
[537]:
```
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'LotConfig', 'Neighborhood', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'Exterior1st', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'Foundation', 'BsmtQual', 'BsmtCond',
       'BsmtExposure', 'TotalBsmtSF', 'Heating', 'CentralAir', 'Electrical',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'WoodDeckSF',
       'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
       'MiscVal', 'MoSold', 'SaleType', 'SaleCondition', 'SalePrice'],
      dtype='object')
```

[538]:
```python
chi2data = chi2_test(train_df)
print(chi2data.sort_values(by = 'p_value'))
```

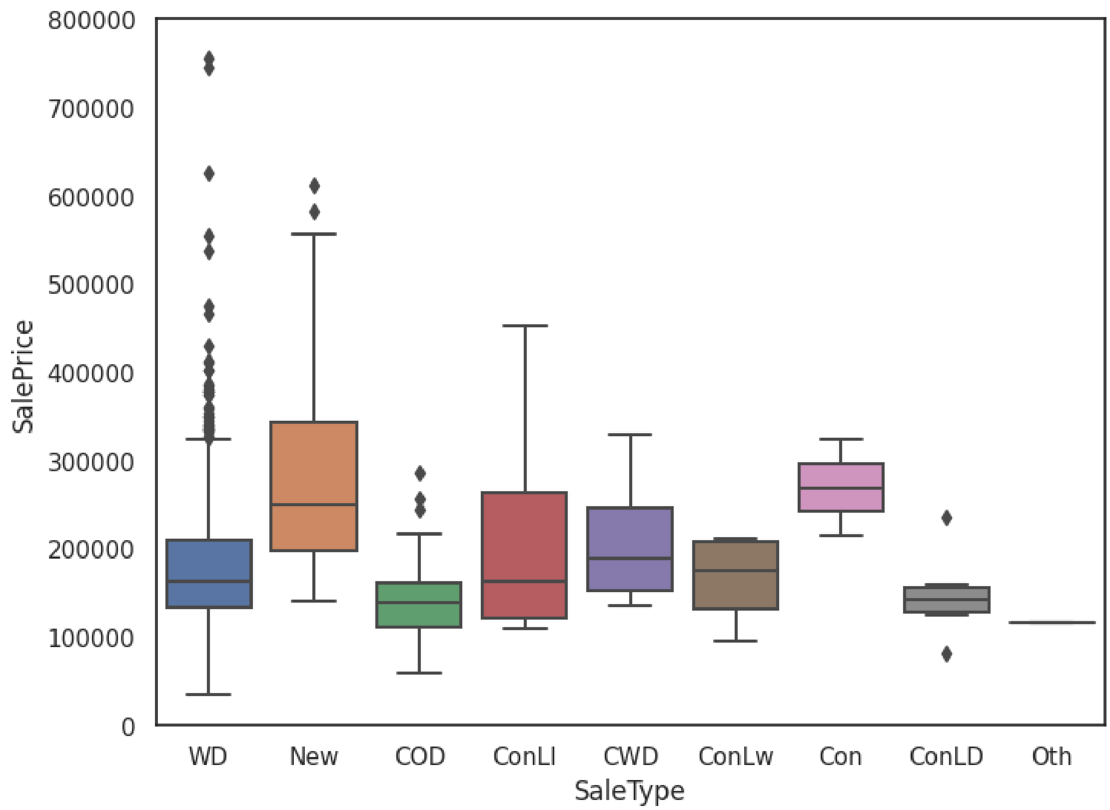
```
          feature          chi2       p_value
9       ExterQual   2811.486875  1.294357e-42
22       SaleType   6386.934740  1.067397e-40
16     Electrical   3423.789697  4.270613e-34
12       BsmtCond   2653.607777  1.604465e-31
23  SaleCondition   4065.333763  6.030287e-30
1          Street   1088.427302  1.878083e-28
0        MSZoning   3276.414434  1.221275e-25
17    KitchenQual   2502.096429  3.361517e-22
11       BsmtQual   2496.248240  7.214869e-22
21     GarageQual   3031.992059  4.999571e-14
2        LotShape   2329.826900  2.219481e-13
20   GarageFinish   1548.351856  2.725970e-09
5    Neighborhood  15694.784597  4.342533e-07
13   BsmtExposure   2162.762636  7.021146e-07
15     CentralAir    779.355259  1.020573e-05
10     Foundation   2662.532706  3.994538e-03
19     GarageType   3225.220808  4.418462e-02
4       LotConfig   2584.013322  5.716175e-02
7     Exterior1st   8173.908469  1.350918e-01
```

```
8        MasVnrType   1285.856431   1.579424e-01
14          Heating   1883.884503   3.088582e-01
3       LandContour   1864.807573   4.254058e-01
6        HouseStyle   4301.299244   6.021399e-01
18      FireplaceQu   2380.592378   9.043710e-01
```

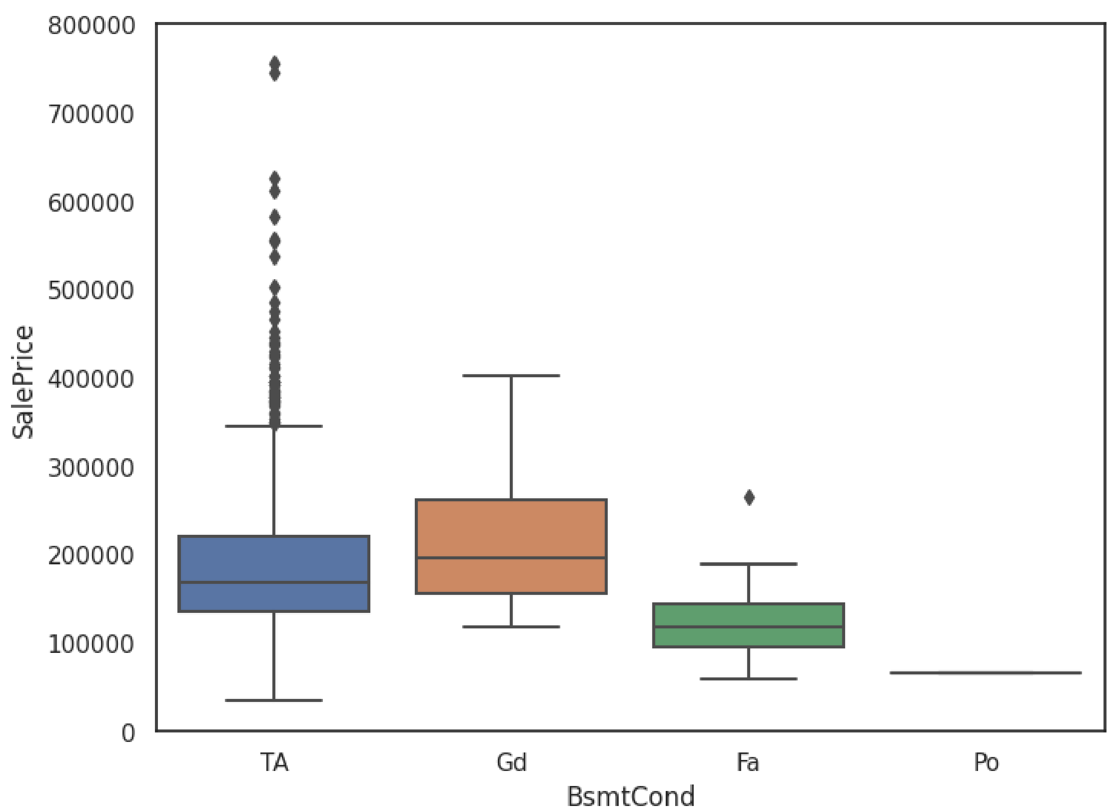
[539]:
```python
var = 'ExterQual'
data = pd.concat([train_df['SalePrice'], train_df[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



[540]:
```python
var = 'SaleType'
data = pd.concat([train_df['SalePrice'], train_df[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



[541]:
```
var = 'BsmtCond'
data = pd.concat([train_df['SalePrice'], train_df[var]], axis=1)
f, ax = plt.subplots(figsize=(8, 6))
fig = sns.boxplot(x=var, y="SalePrice", data=data)
fig.axis(ymin=0, ymax=800000);
```



[542]:
```
train_df.head()
```

[542]:
```
   MSSubClass MSZoning  LotFrontage  LotArea Street LotShape LandContour  \
0          60       RL         65.0     8450   Pave      Reg         Lvl
1          20       RL         80.0     9600   Pave      Reg         Lvl
2          60       RL         68.0    11250   Pave      IR1         Lvl
3          70       RL         60.0     9550   Pave      IR1         Lvl
4          60       RL         84.0    14260   Pave      IR1         Lvl

  LotConfig Neighborhood HouseStyle  …  OpenPorchSF  EnclosedPorch  \
0    Inside      CollgCr     2Story  …           61              0
1       FR2      Veenker     1Story  …            0              0
2    Inside      CollgCr     2Story  …           42              0
3    Corner      Crawfor     2Story  …           35            272
4       FR2      NoRidge     2Story  …           84              0

   3SsnPorch ScreenPorch PoolArea  MiscVal MoSold SaleType SaleCondition  \
0          0           0        0        0      2       WD        Normal
1          0           0        0        0      5       WD        Normal
2          0           0        0        0      9       WD        Normal
3          0           0        0        0      2       WD       Abnorml
```

```
4          0           0        0        0     12       WD         Normal

  SalePrice
0    208500
1    181500
2    223500
3    140000
4    250000

[5 rows x 53 columns]
```

EDA Summary

- We have analyse the data using Correlation Matrix, Variance Inflation Factor (VIF) and statistical test, Chi-squared test to remove features which might not helpful for modelling.
- Now, we have left with 53 features.

4 Modelling

4.0.1 Model Choices

- **XGBoost**
  - XGBoost is an optimized gradient boosting library designed for speed and performance, often used for supervised learning tasks.
- **CatBoost**
  - CatBoost is a gradient boosting library that excels in handling categorical features and aims for ease of use with robust, out-of-the-box performance.

4.0.2 Evaluation Metrics

We will use the following metric - RMSE (Root Mean Squared Error) is a measure of the average magnitude of the model's errors, treating all errors equally, regardless of their direction or size.

4.0.3 Experiments

My experiments are the following: 1. Train XGBoost 2. Hyperparameter Tuning (XGBoost) 3. Train CatBoost 4. Hyperparameter Tuning (CatBoost)

4.0.4 Feature Engineering

Label Encoding I will transform the categorical into numerical format using label encoding.

[543]:
```
train_cate_df = train_df.copy()
```

[544]:
```
cate_cols = train_cate_df.select_dtypes(include = ['object']).columns
for c in cate_cols:
    lbl = LabelEncoder()
    lbl.fit(list(train_cate_df[c].values))
    train_cate_df[c] = lbl.transform(list(train_cate_df[c].values))

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

```
Shape all_data: (1336, 53)
```

[545]:
```
train_cate_df[cate_cols].head()
```

[545]:
```
   MSZoning  Street  LotShape  LandContour  LotConfig  Neighborhood  \
0         3       1         3            3          4             5
1         3       1         3            3          2            24
2         3       1         0            3          4             5
3         3       1         0            3          0             6
4         3       1         0            3          2            15

   HouseStyle  Exterior1st  MasVnrType  ExterQual  …  Heating  CentralAir  \
0           5           11           1          2  …        0           1
1           2            7           1          3  …        0           1
2           5           11           1          2  …        0           1
3           5           12           2          3  …        0           1
4           5           11           1          2  …        0           1

   Electrical  KitchenQual  FireplaceQu  GarageType  GarageFinish  GarageQual  \
0           4            2            1           1             1           4
1           4            3            4           1             1           4
2           4            2            4           1             1           4
3           4            2            2           5             2           4
4           4            2            4           1             1           4

   SaleType  SaleCondition
0         8              4
1         8              4
2         8              4
3         8              0
4         8              4

[5 rows x 24 columns]
```

[546]:
```
train_df.columns
```

[546]:
```
Index(['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'LotShape', 'LandContour', 'LotConfig', 'Neighborhood', 'HouseStyle',
       'OverallQual', 'OverallCond', 'YearBuilt', 'Exterior1st', 'MasVnrType',
```

```
 'MasVnrArea', 'ExterQual', 'Foundation', 'BsmtQual', 'BsmtCond',
 'BsmtExposure', 'TotalBsmtSF', 'Heating', 'CentralAir', 'Electrical',
 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
 'TotRmsAbvGrd', 'Fireplaces', 'FireplaceQu', 'GarageType',
 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual', 'WoodDeckSF',
 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea',
 'MiscVal', 'MoSold', 'SaleType', 'SaleCondition', 'SalePrice'],
dtype='object')
```

[611]:
```
train_all = train_cate_df.copy()
```

4.0.5 Evaluation Metrics

Two helper functions to evaluate the performace using Cross Validation RMSLE and RMSE.

[455]:
```
def rmsle_cv(model, xtrain, ytrain, n_folds):
    kf = KFold(n_folds, shuffle=True, random_state=42).get_n_splits(xtrain.
 ↪values)
    rmse= np.sqrt(-cross_val_score(model, xtrain.values, ytrain,␣
 ↪scoring="neg_mean_squared_error", cv = kf))
    return(rmse)

def rmse(y, y_pred):
    return np.round(np.sqrt(mean_squared_error(y, y_pred)), 2)
```

4.1 Model Training

4.1.1 XGBoost

Data Preparation

[612]:
```
train_all["SalePrice"] = np.log1p(train_all["SalePrice"])
train_all = pd.get_dummies(train_all)

X = train_all.drop(columns = ['SalePrice'], axis = 1)
y = train_all['SalePrice']
```

[613]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=32)
```

[614]:
```
print(f' Training Features {X_train.shape}')
print(f' Training Target {y_train.shape}')
print(f' Test Feature {X_test.shape}')
print(f' Test Target {y_test.shape}')
```

```
 Training Features (1068, 52)
 Training Target (1068,)
```

```
Test Feature (268, 52)
Test Target (268,)
```

Train

[615]:
```
model_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                             learning_rate=0.05, max_depth=3,
                             min_child_weight=1.7817, n_estimators=2200,
                             reg_alpha=0.4640, reg_lambda=0.8571,
                             subsample=0.5213, silent=1,
                             random_state =7, nthread = -1)
```

[620]:
```
n_folds = 5

score = rmsle_cv(model_xgb, X_train, y_train, n_folds)
print("XGBoost score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
```

```
XGBoost score: 0.1213 (0.0177)
```

[621]:
```
model_xgb.fit(X_train, y_train)
```

[621]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.4603, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.0468, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.05, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=1.7817, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=2200, n_jobs=None, nthread=-1,
             num_parallel_tree=None, …)
```

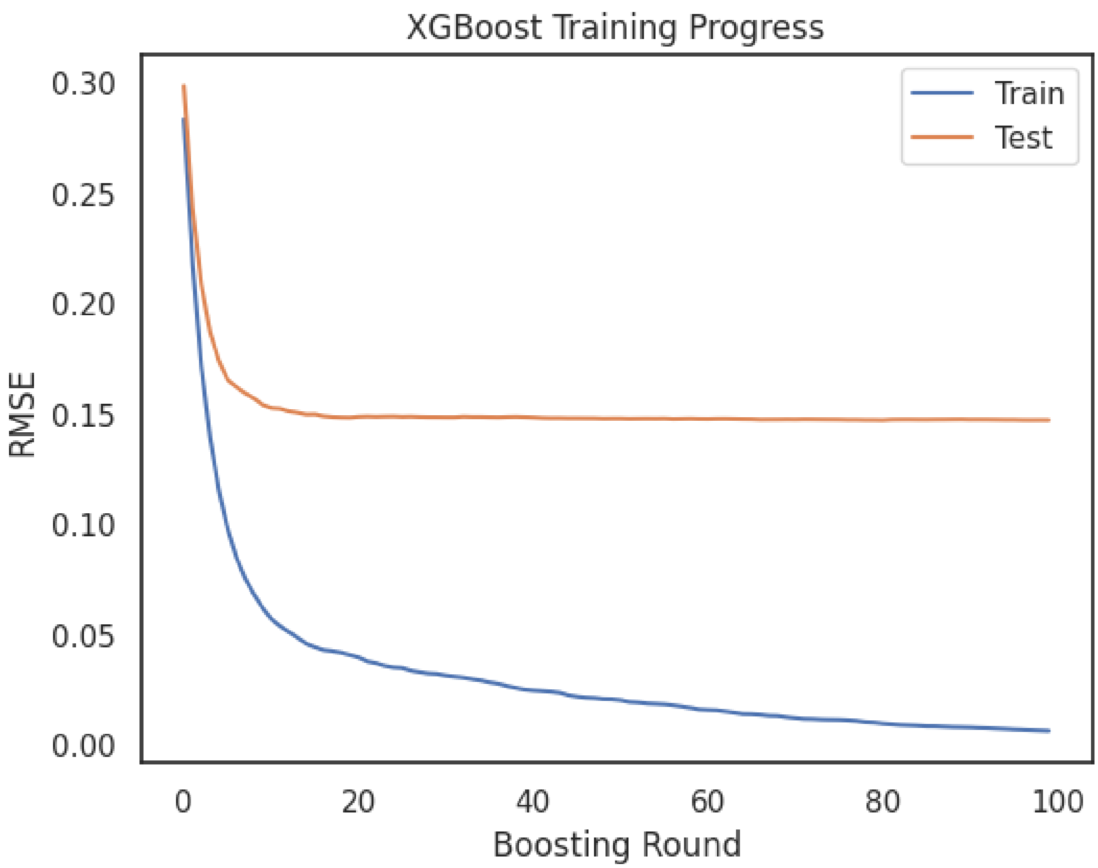
[616]:
```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

params = {'objective': 'reg:squarederror', 'eval_metric': 'rmse'}

evals = [(dtrain, 'train'), (dtest, 'eval')]
evals_result = {}
bst = xgb.train(params, dtrain, num_boost_round=100, evals=evals,␣
 ↪evals_result=evals_result)
```

```
[0]     train-rmse:0.28336      eval-rmse:0.29853
[1]     train-rmse:0.21821      eval-rmse:0.24315
[2]     train-rmse:0.17152      eval-rmse:0.20865
[3]     train-rmse:0.13899      eval-rmse:0.18680
[4]     train-rmse:0.11514      eval-rmse:0.17384
```

```
[5]	train-rmse:0.09780	eval-rmse:0.16528
[6]	train-rmse:0.08468	eval-rmse:0.16199
[7]	train-rmse:0.07520	eval-rmse:0.15914
[8]	train-rmse:0.06801	eval-rmse:0.15688
[9]	train-rmse:0.06178	eval-rmse:0.15393
[10]	train-rmse:0.05669	eval-rmse:0.15242
[11]	train-rmse:0.05350	eval-rmse:0.15214
[12]	train-rmse:0.05091	eval-rmse:0.15095
[13]	train-rmse:0.04834	eval-rmse:0.15036
[14]	train-rmse:0.04550	eval-rmse:0.14948
[15]	train-rmse:0.04392	eval-rmse:0.14954
[16]	train-rmse:0.04261	eval-rmse:0.14870
[17]	train-rmse:0.04212	eval-rmse:0.14831
[18]	train-rmse:0.04142	eval-rmse:0.14815
[19]	train-rmse:0.04037	eval-rmse:0.14808
[20]	train-rmse:0.03930	eval-rmse:0.14843
[21]	train-rmse:0.03749	eval-rmse:0.14857
[22]	train-rmse:0.03668	eval-rmse:0.14845
[23]	train-rmse:0.03546	eval-rmse:0.14855
[24]	train-rmse:0.03474	eval-rmse:0.14860
[25]	train-rmse:0.03447	eval-rmse:0.14841
[26]	train-rmse:0.03324	eval-rmse:0.14850
[27]	train-rmse:0.03257	eval-rmse:0.14835
[28]	train-rmse:0.03199	eval-rmse:0.14826
[29]	train-rmse:0.03174	eval-rmse:0.14825
[30]	train-rmse:0.03100	eval-rmse:0.14821
[31]	train-rmse:0.03053	eval-rmse:0.14821
[32]	train-rmse:0.03010	eval-rmse:0.14848
[33]	train-rmse:0.02943	eval-rmse:0.14835
[34]	train-rmse:0.02882	eval-rmse:0.14835
[35]	train-rmse:0.02795	eval-rmse:0.14826
[36]	train-rmse:0.02732	eval-rmse:0.14820
[37]	train-rmse:0.02626	eval-rmse:0.14834
[38]	train-rmse:0.02540	eval-rmse:0.14841
[39]	train-rmse:0.02467	eval-rmse:0.14829
[40]	train-rmse:0.02428	eval-rmse:0.14814
[41]	train-rmse:0.02401	eval-rmse:0.14792
[42]	train-rmse:0.02372	eval-rmse:0.14782
[43]	train-rmse:0.02326	eval-rmse:0.14784
[44]	train-rmse:0.02211	eval-rmse:0.14776
[45]	train-rmse:0.02145	eval-rmse:0.14776
[46]	train-rmse:0.02102	eval-rmse:0.14774
[47]	train-rmse:0.02083	eval-rmse:0.14772
[48]	train-rmse:0.02037	eval-rmse:0.14758
[49]	train-rmse:0.02028	eval-rmse:0.14762
[50]	train-rmse:0.01987	eval-rmse:0.14762
[51]	train-rmse:0.01908	eval-rmse:0.14752
[52]	train-rmse:0.01885	eval-rmse:0.14756
```

```
[53]	train-rmse:0.01840	eval-rmse:0.14759
[54]	train-rmse:0.01821	eval-rmse:0.14756
[55]	train-rmse:0.01799	eval-rmse:0.14761
[56]	train-rmse:0.01756	eval-rmse:0.14742
[57]	train-rmse:0.01697	eval-rmse:0.14749
[58]	train-rmse:0.01637	eval-rmse:0.14756
[59]	train-rmse:0.01563	eval-rmse:0.14745
[60]	train-rmse:0.01542	eval-rmse:0.14742
[61]	train-rmse:0.01526	eval-rmse:0.14752
[62]	train-rmse:0.01474	eval-rmse:0.14753
[63]	train-rmse:0.01421	eval-rmse:0.14745
[64]	train-rmse:0.01360	eval-rmse:0.14737
[65]	train-rmse:0.01351	eval-rmse:0.14733
[66]	train-rmse:0.01324	eval-rmse:0.14717
[67]	train-rmse:0.01281	eval-rmse:0.14720
[68]	train-rmse:0.01270	eval-rmse:0.14720
[69]	train-rmse:0.01212	eval-rmse:0.14725
[70]	train-rmse:0.01172	eval-rmse:0.14722
[71]	train-rmse:0.01136	eval-rmse:0.14729
[72]	train-rmse:0.01128	eval-rmse:0.14726
[73]	train-rmse:0.01107	eval-rmse:0.14722
[74]	train-rmse:0.01098	eval-rmse:0.14720
[75]	train-rmse:0.01093	eval-rmse:0.14717
[76]	train-rmse:0.01076	eval-rmse:0.14708
[77]	train-rmse:0.01046	eval-rmse:0.14708
[78]	train-rmse:0.00991	eval-rmse:0.14700
[79]	train-rmse:0.00972	eval-rmse:0.14699
[80]	train-rmse:0.00925	eval-rmse:0.14692
[81]	train-rmse:0.00896	eval-rmse:0.14715
[82]	train-rmse:0.00865	eval-rmse:0.14719
[83]	train-rmse:0.00850	eval-rmse:0.14719
[84]	train-rmse:0.00833	eval-rmse:0.14715
[85]	train-rmse:0.00807	eval-rmse:0.14714
[86]	train-rmse:0.00804	eval-rmse:0.14719
[87]	train-rmse:0.00785	eval-rmse:0.14720
[88]	train-rmse:0.00768	eval-rmse:0.14724
[89]	train-rmse:0.00761	eval-rmse:0.14726
[90]	train-rmse:0.00751	eval-rmse:0.14717
[91]	train-rmse:0.00735	eval-rmse:0.14718
[92]	train-rmse:0.00724	eval-rmse:0.14717
[93]	train-rmse:0.00703	eval-rmse:0.14712
[94]	train-rmse:0.00685	eval-rmse:0.14707
[95]	train-rmse:0.00663	eval-rmse:0.14707
[96]	train-rmse:0.00648	eval-rmse:0.14697
[97]	train-rmse:0.00630	eval-rmse:0.14696
[98]	train-rmse:0.00614	eval-rmse:0.14697
[99]	train-rmse:0.00599	eval-rmse:0.14694
```

[617]:
```
plt.plot(evals_result['train']['rmse'], label='Train')
plt.plot(evals_result['eval']['rmse'], label='Test')
plt.xlabel('Boosting Round')
plt.ylabel('RMSE')
plt.title('XGBoost Training Progress')
plt.legend()
plt.show()
```



Evaluate

[8]:
```
xgb_train_pred = model_xgb.predict(X_train)
print(f'XGBoost Train RMSE {rmse(y_train, xgb_train_pred)}')

xgb_pred = model_xgb.predict(X_test)
print(f'XGBoost Test RMSE {rmse(y_test, xgb_pred)}')
```

```
XGBoost Train RMSE 0.09
XGBoost Test RMSE 0.12
```

4.1.2 XGBoost with Hyperparameter Tuning

[624]:
```python
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 4, 5],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9]
}

xgb_model = xgb.XGBRegressor()

grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid, cv=3)
grid_search.fit(X_train, y_train)
```

[624]:
```
GridSearchCV(cv=3,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    callbacks=None, colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, device=None,
                                    early_stopping_rounds=None,
                                    enable_categorical=False, eval_metric=None,
                                    feature_types=None, gamma=None,
                                    grow_policy=None, importance_type=None,
                                    interaction_constraints=None,
                                    learning_rate=None, m…
                                    max_cat_to_onehot=None, max_delta_step=None,
                                    max_depth=None, max_leaves=None,
                                    min_child_weight=None, missing=nan,
                                    monotone_constraints=None,
                                    multi_strategy=None, n_estimators=None,
                                    n_jobs=None, num_parallel_tree=None,
                                    random_state=None, …),
             param_grid={'colsample_bytree': [0.7, 0.8, 0.9],
                         'learning_rate': [0.01, 0.05, 0.1],
                         'max_depth': [3, 4, 5],
                         'n_estimators': [100, 200, 300],
                         'subsample': [0.7, 0.8, 0.9]})
```

[625]:
```python
best_parameters = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Parameters: {best_parameters}")
print(f"Best Score: {best_score}")
best_model = grid_search.best_estimator_
```

```
Best Parameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': 3,
'n_estimators': 200, 'subsample': 0.9}
```

```
Best Score: 0.9012268726566676
```

[626]:
```
best_xgb_model = xgb.XGBRegressor(**best_parameters)
best_xgb_model.fit(X_train, y_train)
```

[626]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=0.8, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=200, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[3]:
```
xgb_train_pred = best_xgb_model.predict(X_train)
print(f'XGBoost Train RMSE {rmse(y_train, xgb_train_pred)}')

xgb_pred = best_xgb_model.predict(X_test)
print(f'XGBoost Test RMSE {rmse(y_test, xgb_pred)}')
```

```
XGBoost Train RMSE 0.06
XGBoost Test RMSE 0.13
```

4.1.3 CatBoost

Data Preparation

[549]:
```
train_clone = train_df.copy()
float_cols = train_clone.select_dtypes(include = ['float64']).columns
train_clone[float_cols] = train_clone[float_cols].astype(int)
```

[559]:
```
train_clone["SalePrice"] = np.log1p(train_clone["SalePrice"])
train_clone = pd.get_dummies(train_clone)

X  = train_clone.drop(columns = ['SalePrice'], axis = 1)
y = train_clone['SalePrice']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=32)

categorical_features_names = list(X.columns)
```

[561]:
```
train_pool = Pool(X_train,
                  label=y_train,
                  cat_features=categorical_features_names)
```

```
test_pool = Pool(X_test,
                 label=y_test,
                 cat_features=categorical_features_names)
```

Train

[563]:
```
model = CatBoostRegressor(custom_metric= ['R2', 'RMSE'], learning_rate=0.01,␣
 ↪depth = 10, n_estimators=5000)
model.fit(train_pool, eval_set=test_pool, verbose=1000, plot=True)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))

0:      learn: 0.0416175        test: 0.0417206 best: 0.0417206 (0)     total:
18.2ms   remaining: 1m 30s
1000:   learn: 0.0112569        test: 0.0232236 best: 0.0232236 (1000)  total:
16.6s    remaining: 1m 6s
2000:   learn: 0.0070376        test: 0.0228457 best: 0.0228457 (2000)  total:
35.6s    remaining: 53.3s
3000:   learn: 0.0046750        test: 0.0227822 best: 0.0227815 (2955)  total:
53.8s    remaining: 35.8s
4000:   learn: 0.0029821        test: 0.0227638 best: 0.0227635 (3998)  total:
1m 12s   remaining: 18.1s
4999:   learn: 0.0020532        test: 0.0227434 best: 0.0227428 (4873)  total:
1m 31s   remaining: 0us

bestTest = 0.02274284214
bestIteration = 4873

Shrink model to first 4874 iterations.
```

[563]:
```
<catboost.core.CatBoostRegressor at 0x7f6cd1f807d0>
```

Evaluate

[4]:
```
cat_train_pred = model.predict(X_train)
print(f'CatBoost Train RMSE {rmse(y_train, xgb_train_pred)}')

cat_pred = model.predict(X_test)
print(f'CatBoost Test RMSE {rmse(y_test, xgb_pred)}')
```

```
CatBoost Train RMSE 9.53
CatBoost Test RMSE 9.57
```

4.1.4 CatBoost with Hyperparameter Tuning

[593]:
```
param_grid = {
    'iterations': [500, 1000],
    'learning_rate': [0.01, 0.1, 0.2],
    'depth': [4, 6, 8],
```

```
    'l2_leaf_reg': [1, 3, 5, 7, 9]
}
```

[594]:
```
grid_search = GridSearchCV(estimator=CatBoostRegressor(),
                           param_grid=param_grid,
                           cv=5,
                           n_jobs=-1,
                           verbose=0)
grid_search.fit(X_train, y_train, verbose=0)
```

[594]:
```
GridSearchCV(cv=5,
             estimator=<catboost.core.CatBoostRegressor object at
0x7f6cd1f3b710>,
             n_jobs=-1,
             param_grid={'depth': [4, 6, 8], 'iterations': [500, 1000],
                         'l2_leaf_reg': [1, 3, 5, 7, 9],
                         'learning_rate': [0.01, 0.1, 0.2]})
```

[595]:
```
best_parameters = grid_search.best_params_
best_score = grid_search.best_score_

print(f"Best Parameters: {best_parameters}")
print(f"Best Score: {best_score}")
```

```
Best Parameters: {'depth': 4, 'iterations': 500, 'l2_leaf_reg': 5,
'learning_rate': 0.1}
Best Score: 0.8861827658601926
```

[596]:
```
best_model = CatBoostRegressor(**best_parameters)
best_model.fit(X_train, y_train)
```

```
0:      learn: 0.3532770        total: 702us    remaining: 350ms
1:      learn: 0.3335623        total: 1.18ms   remaining: 295ms
2:      learn: 0.3157700        total: 1.61ms   remaining: 268ms
3:      learn: 0.2988716        total: 2.08ms   remaining: 257ms
4:      learn: 0.2846018        total: 2.57ms   remaining: 255ms
5:      learn: 0.2707849        total: 3.08ms   remaining: 253ms
6:      learn: 0.2595443        total: 3.63ms   remaining: 256ms
7:      learn: 0.2493907        total: 4.32ms   remaining: 266ms
8:      learn: 0.2391028        total: 5ms      remaining: 273ms
9:      learn: 0.2283552        total: 5.7ms    remaining: 279ms
10:     learn: 0.2197443        total: 6.38ms   remaining: 284ms
11:     learn: 0.2123077        total: 7.06ms   remaining: 287ms
12:     learn: 0.2046252        total: 7.72ms   remaining: 289ms
13:     learn: 0.1984473        total: 8.38ms   remaining: 291ms
14:     learn: 0.1922836        total: 9.03ms   remaining: 292ms
15:     learn: 0.1870114        total: 9.67ms   remaining: 293ms
16:     learn: 0.1821393        total: 10.3ms   remaining: 293ms
```

```
17:	learn: 0.1774804	total: 11ms	remaining: 294ms
18:	learn: 0.1737408	total: 11.6ms	remaining: 295ms
19:	learn: 0.1708838	total: 12.3ms	remaining: 295ms
20:	learn: 0.1669691	total: 12.9ms	remaining: 295ms
21:	learn: 0.1636378	total: 13.6ms	remaining: 295ms
22:	learn: 0.1612121	total: 14.2ms	remaining: 295ms
23:	learn: 0.1585342	total: 14.8ms	remaining: 294ms
24:	learn: 0.1556037	total: 15.5ms	remaining: 294ms
25:	learn: 0.1533850	total: 16.1ms	remaining: 294ms
26:	learn: 0.1506163	total: 16.8ms	remaining: 294ms
27:	learn: 0.1485598	total: 17.5ms	remaining: 294ms
28:	learn: 0.1469492	total: 18.1ms	remaining: 294ms
29:	learn: 0.1454045	total: 18.7ms	remaining: 293ms
30:	learn: 0.1435952	total: 19.3ms	remaining: 293ms
31:	learn: 0.1418491	total: 19.9ms	remaining: 292ms
32:	learn: 0.1402668	total: 20.5ms	remaining: 291ms
33:	learn: 0.1387257	total: 21.1ms	remaining: 289ms
34:	learn: 0.1374522	total: 21.7ms	remaining: 288ms
35:	learn: 0.1357729	total: 22.3ms	remaining: 288ms
36:	learn: 0.1344421	total: 22.9ms	remaining: 287ms
37:	learn: 0.1333160	total: 23.5ms	remaining: 286ms
38:	learn: 0.1321481	total: 24.1ms	remaining: 285ms
39:	learn: 0.1312513	total: 24.7ms	remaining: 284ms
40:	learn: 0.1305290	total: 25.3ms	remaining: 283ms
41:	learn: 0.1293697	total: 25.8ms	remaining: 282ms
42:	learn: 0.1285745	total: 26.4ms	remaining: 281ms
43:	learn: 0.1276534	total: 27ms	remaining: 280ms
44:	learn: 0.1267059	total: 27.6ms	remaining: 279ms
45:	learn: 0.1258917	total: 28.1ms	remaining: 278ms
46:	learn: 0.1252225	total: 28.7ms	remaining: 276ms
47:	learn: 0.1249795	total: 29.2ms	remaining: 275ms
48:	learn: 0.1238678	total: 29.8ms	remaining: 274ms
49:	learn: 0.1230530	total: 30.3ms	remaining: 273ms
50:	learn: 0.1226434	total: 30.9ms	remaining: 272ms
51:	learn: 0.1223813	total: 31.4ms	remaining: 271ms
52:	learn: 0.1215163	total: 32ms	remaining: 270ms
53:	learn: 0.1208749	total: 32.6ms	remaining: 269ms
54:	learn: 0.1206409	total: 33.1ms	remaining: 268ms
55:	learn: 0.1199707	total: 33.6ms	remaining: 267ms
56:	learn: 0.1192999	total: 34.1ms	remaining: 265ms
57:	learn: 0.1186356	total: 34.7ms	remaining: 264ms
58:	learn: 0.1184167	total: 35.9ms	remaining: 268ms
59:	learn: 0.1178437	total: 36.4ms	remaining: 267ms
60:	learn: 0.1175393	total: 36.8ms	remaining: 265ms
61:	learn: 0.1173410	total: 37.3ms	remaining: 264ms
62:	learn: 0.1169933	total: 37.8ms	remaining: 262ms
63:	learn: 0.1163097	total: 38.3ms	remaining: 261ms
64:	learn: 0.1158097	total: 38.7ms	remaining: 259ms
```

```
65:     learn: 0.1150863        total: 39.2ms   remaining: 258ms
66:     learn: 0.1147983        total: 39.7ms   remaining: 256ms
67:     learn: 0.1145866        total: 40.1ms   remaining: 255ms
68:     learn: 0.1144449        total: 40.5ms   remaining: 253ms
69:     learn: 0.1141551        total: 41ms     remaining: 252ms
70:     learn: 0.1135066        total: 41.4ms   remaining: 250ms
71:     learn: 0.1128975        total: 41.9ms   remaining: 249ms
72:     learn: 0.1126441        total: 42.4ms   remaining: 248ms
73:     learn: 0.1122858        total: 42.8ms   remaining: 247ms
74:     learn: 0.1116962        total: 43.3ms   remaining: 245ms
75:     learn: 0.1110930        total: 43.7ms   remaining: 244ms
76:     learn: 0.1109695        total: 44.2ms   remaining: 243ms
77:     learn: 0.1103540        total: 44.6ms   remaining: 242ms
78:     learn: 0.1094938        total: 45ms     remaining: 240ms
79:     learn: 0.1092652        total: 45.5ms   remaining: 239ms
80:     learn: 0.1090595        total: 45.9ms   remaining: 237ms
81:     learn: 0.1085171        total: 46.3ms   remaining: 236ms
82:     learn: 0.1077989        total: 46.8ms   remaining: 235ms
83:     learn: 0.1076079        total: 47.2ms   remaining: 234ms
84:     learn: 0.1069775        total: 47.6ms   remaining: 233ms
85:     learn: 0.1064349        total: 48.1ms   remaining: 231ms
86:     learn: 0.1059226        total: 48.5ms   remaining: 230ms
87:     learn: 0.1058087        total: 48.9ms   remaining: 229ms
88:     learn: 0.1050220        total: 49.3ms   remaining: 228ms
89:     learn: 0.1048879        total: 49.7ms   remaining: 226ms
90:     learn: 0.1043749        total: 50.1ms   remaining: 225ms
91:     learn: 0.1039485        total: 50.5ms   remaining: 224ms
92:     learn: 0.1034659        total: 50.9ms   remaining: 223ms
93:     learn: 0.1031671        total: 51.3ms   remaining: 222ms
94:     learn: 0.1030035        total: 51.7ms   remaining: 220ms
95:     learn: 0.1028822        total: 52.1ms   remaining: 219ms
96:     learn: 0.1024266        total: 52.5ms   remaining: 218ms
97:     learn: 0.1019823        total: 52.9ms   remaining: 217ms
98:     learn: 0.1018539        total: 53.3ms   remaining: 216ms
99:     learn: 0.1014201        total: 53.7ms   remaining: 215ms
100:    learn: 0.1009559        total: 54.1ms   remaining: 214ms
101:    learn: 0.1008594        total: 54.4ms   remaining: 212ms
102:    learn: 0.1004201        total: 54.8ms   remaining: 211ms
103:    learn: 0.1003303        total: 55.2ms   remaining: 210ms
104:    learn: 0.0999516        total: 55.6ms   remaining: 209ms
105:    learn: 0.0995629        total: 56ms     remaining: 208ms
106:    learn: 0.0993895        total: 56.4ms   remaining: 207ms
107:    learn: 0.0993185        total: 56.8ms   remaining: 206ms
108:    learn: 0.0988826        total: 57.2ms   remaining: 205ms
109:    learn: 0.0987320        total: 57.6ms   remaining: 204ms
110:    learn: 0.0983850        total: 58ms     remaining: 203ms
111:    learn: 0.0983180        total: 58.4ms   remaining: 202ms
112:    learn: 0.0982401        total: 58.7ms   remaining: 201ms
```

```
113:	learn: 0.0977169	total: 59.1ms	remaining: 200ms
114:	learn: 0.0976547	total: 59.5ms	remaining: 199ms
115:	learn: 0.0972780	total: 59.9ms	remaining: 198ms
116:	learn: 0.0971918	total: 60.3ms	remaining: 197ms
117:	learn: 0.0970033	total: 60.7ms	remaining: 196ms
118:	learn: 0.0966939	total: 61.1ms	remaining: 196ms
119:	learn: 0.0964549	total: 61.4ms	remaining: 195ms
120:	learn: 0.0961597	total: 61.8ms	remaining: 194ms
121:	learn: 0.0960808	total: 62.2ms	remaining: 193ms
122:	learn: 0.0957279	total: 62.6ms	remaining: 192ms
123:	learn: 0.0955350	total: 63ms	remaining: 191ms
124:	learn: 0.0954149	total: 63.4ms	remaining: 190ms
125:	learn: 0.0953553	total: 63.8ms	remaining: 189ms
126:	learn: 0.0950943	total: 64.2ms	remaining: 188ms
127:	learn: 0.0950252	total: 64.6ms	remaining: 188ms
128:	learn: 0.0949109	total: 65ms	remaining: 187ms
129:	learn: 0.0946060	total: 65.4ms	remaining: 186ms
130:	learn: 0.0942875	total: 65.8ms	remaining: 185ms
131:	learn: 0.0940258	total: 66.2ms	remaining: 184ms
132:	learn: 0.0937458	total: 66.6ms	remaining: 184ms
133:	learn: 0.0934459	total: 66.9ms	remaining: 183ms
134:	learn: 0.0932635	total: 67.3ms	remaining: 182ms
135:	learn: 0.0929438	total: 67.7ms	remaining: 181ms
136:	learn: 0.0928105	total: 68.1ms	remaining: 181ms
137:	learn: 0.0926397	total: 68.5ms	remaining: 180ms
138:	learn: 0.0923935	total: 68.9ms	remaining: 179ms
139:	learn: 0.0921275	total: 69.3ms	remaining: 178ms
140:	learn: 0.0918719	total: 69.7ms	remaining: 177ms
141:	learn: 0.0916759	total: 70.1ms	remaining: 177ms
142:	learn: 0.0916132	total: 70.5ms	remaining: 176ms
143:	learn: 0.0915631	total: 70.9ms	remaining: 175ms
144:	learn: 0.0913864	total: 71.2ms	remaining: 174ms
145:	learn: 0.0913475	total: 71.6ms	remaining: 174ms
146:	learn: 0.0912906	total: 72ms	remaining: 173ms
147:	learn: 0.0910730	total: 72.4ms	remaining: 172ms
148:	learn: 0.0908739	total: 72.8ms	remaining: 171ms
149:	learn: 0.0906041	total: 73.2ms	remaining: 171ms
150:	learn: 0.0905606	total: 73.6ms	remaining: 170ms
151:	learn: 0.0902892	total: 74ms	remaining: 169ms
152:	learn: 0.0902149	total: 74.4ms	remaining: 169ms
153:	learn: 0.0901736	total: 74.8ms	remaining: 168ms
154:	learn: 0.0899191	total: 75.2ms	remaining: 167ms
155:	learn: 0.0896080	total: 75.5ms	remaining: 167ms
156:	learn: 0.0893897	total: 75.9ms	remaining: 166ms
157:	learn: 0.0893508	total: 76.3ms	remaining: 165ms
158:	learn: 0.0891106	total: 76.7ms	remaining: 165ms
159:	learn: 0.0887889	total: 77.1ms	remaining: 164ms
160:	learn: 0.0885169	total: 77.5ms	remaining: 163ms
```

```
161:	learn: 0.0882941	total: 77.9ms	remaining: 163ms
162:	learn: 0.0882435	total: 78.3ms	remaining: 162ms
163:	learn: 0.0881556	total: 78.7ms	remaining: 161ms
164:	learn: 0.0881075	total: 79.1ms	remaining: 161ms
165:	learn: 0.0877820	total: 79.5ms	remaining: 160ms
166:	learn: 0.0877044	total: 79.9ms	remaining: 159ms
167:	learn: 0.0875062	total: 80.3ms	remaining: 159ms
168:	learn: 0.0872990	total: 80.6ms	remaining: 158ms
169:	learn: 0.0870951	total: 81ms	remaining: 157ms
170:	learn: 0.0870520	total: 81.5ms	remaining: 157ms
171:	learn: 0.0869653	total: 81.9ms	remaining: 156ms
172:	learn: 0.0866681	total: 82.2ms	remaining: 155ms
173:	learn: 0.0866180	total: 82.6ms	remaining: 155ms
174:	learn: 0.0864357	total: 83ms	remaining: 154ms
175:	learn: 0.0862041	total: 83.4ms	remaining: 154ms
176:	learn: 0.0859825	total: 83.8ms	remaining: 153ms
177:	learn: 0.0857469	total: 84.2ms	remaining: 152ms
178:	learn: 0.0855579	total: 84.6ms	remaining: 152ms
179:	learn: 0.0853386	total: 85ms	remaining: 151ms
180:	learn: 0.0852988	total: 85.4ms	remaining: 151ms
181:	learn: 0.0851989	total: 85.8ms	remaining: 150ms
182:	learn: 0.0851686	total: 86.2ms	remaining: 149ms
183:	learn: 0.0850590	total: 86.6ms	remaining: 149ms
184:	learn: 0.0847544	total: 86.9ms	remaining: 148ms
185:	learn: 0.0846113	total: 87.3ms	remaining: 147ms
186:	learn: 0.0842964	total: 87.7ms	remaining: 147ms
187:	learn: 0.0841541	total: 88.1ms	remaining: 146ms
188:	learn: 0.0839727	total: 88.5ms	remaining: 146ms
189:	learn: 0.0837707	total: 88.9ms	remaining: 145ms
190:	learn: 0.0836764	total: 89.3ms	remaining: 144ms
191:	learn: 0.0836000	total: 89.7ms	remaining: 144ms
192:	learn: 0.0834713	total: 90.1ms	remaining: 143ms
193:	learn: 0.0834224	total: 90.5ms	remaining: 143ms
194:	learn: 0.0833579	total: 90.9ms	remaining: 142ms
195:	learn: 0.0833009	total: 91.3ms	remaining: 142ms
196:	learn: 0.0830016	total: 91.7ms	remaining: 141ms
197:	learn: 0.0828059	total: 92.1ms	remaining: 140ms
198:	learn: 0.0825121	total: 92.5ms	remaining: 140ms
199:	learn: 0.0824780	total: 92.8ms	remaining: 139ms
200:	learn: 0.0824052	total: 93.2ms	remaining: 139ms
201:	learn: 0.0822494	total: 93.6ms	remaining: 138ms
202:	learn: 0.0820717	total: 94ms	remaining: 138ms
203:	learn: 0.0817942	total: 94.4ms	remaining: 137ms
204:	learn: 0.0815523	total: 94.8ms	remaining: 136ms
205:	learn: 0.0813970	total: 95.2ms	remaining: 136ms
206:	learn: 0.0811631	total: 95.6ms	remaining: 135ms
207:	learn: 0.0810690	total: 96ms	remaining: 135ms
208:	learn: 0.0809741	total: 96.4ms	remaining: 134ms
```

```
209:	learn: 0.0809444	total: 96.8ms	remaining: 134ms
210:	learn: 0.0807871	total: 97.1ms	remaining: 133ms
211:	learn: 0.0806855	total: 97.5ms	remaining: 132ms
212:	learn: 0.0806029	total: 97.9ms	remaining: 132ms
213:	learn: 0.0804003	total: 98.3ms	remaining: 131ms
214:	learn: 0.0803776	total: 98.7ms	remaining: 131ms
215:	learn: 0.0802203	total: 99.1ms	remaining: 130ms
216:	learn: 0.0800418	total: 99.5ms	remaining: 130ms
217:	learn: 0.0798038	total: 99.9ms	remaining: 129ms
218:	learn: 0.0796512	total: 100ms	remaining: 129ms
219:	learn: 0.0795124	total: 101ms	remaining: 128ms
220:	learn: 0.0792978	total: 101ms	remaining: 128ms
221:	learn: 0.0792084	total: 101ms	remaining: 127ms
222:	learn: 0.0791783	total: 102ms	remaining: 126ms
223:	learn: 0.0789725	total: 102ms	remaining: 126ms
224:	learn: 0.0788076	total: 103ms	remaining: 125ms
225:	learn: 0.0786858	total: 103ms	remaining: 125ms
226:	learn: 0.0786158	total: 103ms	remaining: 124ms
227:	learn: 0.0784174	total: 104ms	remaining: 124ms
228:	learn: 0.0783451	total: 104ms	remaining: 123ms
229:	learn: 0.0782373	total: 105ms	remaining: 123ms
230:	learn: 0.0781315	total: 105ms	remaining: 122ms
231:	learn: 0.0779739	total: 105ms	remaining: 122ms
232:	learn: 0.0778122	total: 106ms	remaining: 121ms
233:	learn: 0.0777659	total: 106ms	remaining: 121ms
234:	learn: 0.0776276	total: 106ms	remaining: 120ms
235:	learn: 0.0774570	total: 107ms	remaining: 120ms
236:	learn: 0.0772552	total: 107ms	remaining: 119ms
237:	learn: 0.0771638	total: 108ms	remaining: 118ms
238:	learn: 0.0770942	total: 108ms	remaining: 118ms
239:	learn: 0.0769152	total: 108ms	remaining: 117ms
240:	learn: 0.0767690	total: 109ms	remaining: 117ms
241:	learn: 0.0766074	total: 109ms	remaining: 116ms
242:	learn: 0.0764837	total: 110ms	remaining: 116ms
243:	learn: 0.0764639	total: 110ms	remaining: 115ms
244:	learn: 0.0762655	total: 110ms	remaining: 115ms
245:	learn: 0.0761176	total: 111ms	remaining: 114ms
246:	learn: 0.0759723	total: 111ms	remaining: 114ms
247:	learn: 0.0758935	total: 112ms	remaining: 113ms
248:	learn: 0.0758147	total: 112ms	remaining: 113ms
249:	learn: 0.0756756	total: 112ms	remaining: 112ms
250:	learn: 0.0755715	total: 113ms	remaining: 112ms
251:	learn: 0.0755328	total: 113ms	remaining: 111ms
252:	learn: 0.0754021	total: 114ms	remaining: 111ms
253:	learn: 0.0753742	total: 114ms	remaining: 110ms
254:	learn: 0.0751955	total: 114ms	remaining: 110ms
255:	learn: 0.0751103	total: 115ms	remaining: 109ms
256:	learn: 0.0750341	total: 115ms	remaining: 109ms
```

```
257:    learn: 0.0750179        total: 116ms    remaining: 108ms
258:    learn: 0.0749516        total: 116ms    remaining: 108ms
259:    learn: 0.0748211        total: 116ms    remaining: 107ms
260:    learn: 0.0746904        total: 117ms    remaining: 107ms
261:    learn: 0.0745694        total: 117ms    remaining: 106ms
262:    learn: 0.0745494        total: 118ms    remaining: 106ms
263:    learn: 0.0744222        total: 118ms    remaining: 105ms
264:    learn: 0.0742872        total: 118ms    remaining: 105ms
265:    learn: 0.0742418        total: 119ms    remaining: 104ms
266:    learn: 0.0741839        total: 119ms    remaining: 104ms
267:    learn: 0.0741687        total: 119ms    remaining: 103ms
268:    learn: 0.0741068        total: 120ms    remaining: 103ms
269:    learn: 0.0739354        total: 120ms    remaining: 102ms
270:    learn: 0.0738229        total: 121ms    remaining: 102ms
271:    learn: 0.0736866        total: 121ms    remaining: 101ms
272:    learn: 0.0736678        total: 121ms    remaining: 101ms
273:    learn: 0.0734981        total: 122ms    remaining: 100ms
274:    learn: 0.0734131        total: 122ms    remaining: 100ms
275:    learn: 0.0732944        total: 123ms    remaining: 99.5ms
276:    learn: 0.0731585        total: 123ms    remaining: 99ms
277:    learn: 0.0729876        total: 123ms    remaining: 98.5ms
278:    learn: 0.0727894        total: 124ms    remaining: 98ms
279:    learn: 0.0727753        total: 124ms    remaining: 97.5ms
280:    learn: 0.0726864        total: 125ms    remaining: 97.1ms
281:    learn: 0.0726689        total: 125ms    remaining: 96.6ms
282:    learn: 0.0725333        total: 125ms    remaining: 96.1ms
283:    learn: 0.0723920        total: 126ms    remaining: 95.6ms
284:    learn: 0.0723054        total: 126ms    remaining: 95.1ms
285:    learn: 0.0720663        total: 127ms    remaining: 94.7ms
286:    learn: 0.0719297        total: 127ms    remaining: 94.2ms
287:    learn: 0.0719163        total: 127ms    remaining: 93.7ms
288:    learn: 0.0718037        total: 128ms    remaining: 93.2ms
289:    learn: 0.0717358        total: 128ms    remaining: 92.7ms
290:    learn: 0.0717158        total: 128ms    remaining: 92.3ms
291:    learn: 0.0716435        total: 129ms    remaining: 91.8ms
292:    learn: 0.0715163        total: 129ms    remaining: 91.3ms
293:    learn: 0.0714223        total: 130ms    remaining: 90.8ms
294:    learn: 0.0713620        total: 130ms    remaining: 90.3ms
295:    learn: 0.0712594        total: 130ms    remaining: 89.9ms
296:    learn: 0.0710767        total: 131ms    remaining: 89.4ms
297:    learn: 0.0709328        total: 131ms    remaining: 88.9ms
298:    learn: 0.0708359        total: 132ms    remaining: 88.5ms
299:    learn: 0.0708209        total: 132ms    remaining: 88ms
300:    learn: 0.0707996        total: 132ms    remaining: 87.5ms
301:    learn: 0.0706146        total: 133ms    remaining: 87ms
302:    learn: 0.0705911        total: 133ms    remaining: 86.5ms
303:    learn: 0.0705168        total: 134ms    remaining: 86.1ms
304:    learn: 0.0705051        total: 134ms    remaining: 85.6ms
```

```
305:	learn: 0.0704226	total: 134ms	remaining: 85.1ms
306:	learn: 0.0703633	total: 135ms	remaining: 84.7ms
307:	learn: 0.0702203	total: 135ms	remaining: 84.2ms
308:	learn: 0.0701049	total: 135ms	remaining: 83.7ms
309:	learn: 0.0700562	total: 136ms	remaining: 83.3ms
310:	learn: 0.0700418	total: 136ms	remaining: 82.8ms
311:	learn: 0.0699061	total: 137ms	remaining: 82.3ms
312:	learn: 0.0698217	total: 137ms	remaining: 81.8ms
313:	learn: 0.0696523	total: 137ms	remaining: 81.4ms
314:	learn: 0.0695029	total: 138ms	remaining: 80.9ms
315:	learn: 0.0693740	total: 138ms	remaining: 80.5ms
316:	learn: 0.0693117	total: 139ms	remaining: 80ms
317:	learn: 0.0691794	total: 139ms	remaining: 79.5ms
318:	learn: 0.0691165	total: 139ms	remaining: 79.1ms
319:	learn: 0.0690282	total: 140ms	remaining: 78.6ms
320:	learn: 0.0688543	total: 140ms	remaining: 78.2ms
321:	learn: 0.0687697	total: 141ms	remaining: 77.7ms
322:	learn: 0.0686841	total: 141ms	remaining: 77.2ms
323:	learn: 0.0686084	total: 141ms	remaining: 76.8ms
324:	learn: 0.0685948	total: 142ms	remaining: 76.3ms
325:	learn: 0.0685114	total: 142ms	remaining: 75.9ms
326:	learn: 0.0684991	total: 143ms	remaining: 75.4ms
327:	learn: 0.0684779	total: 143ms	remaining: 74.9ms
328:	learn: 0.0683988	total: 143ms	remaining: 74.5ms
329:	learn: 0.0682340	total: 144ms	remaining: 74ms
330:	learn: 0.0681769	total: 144ms	remaining: 73.6ms
331:	learn: 0.0680506	total: 144ms	remaining: 73.1ms
332:	learn: 0.0679364	total: 145ms	remaining: 72.7ms
333:	learn: 0.0679113	total: 146ms	remaining: 72.6ms
334:	learn: 0.0677760	total: 146ms	remaining: 72.1ms
335:	learn: 0.0677520	total: 147ms	remaining: 71.7ms
336:	learn: 0.0676131	total: 147ms	remaining: 71.2ms
337:	learn: 0.0675030	total: 148ms	remaining: 70.8ms
338:	learn: 0.0674325	total: 148ms	remaining: 70.3ms
339:	learn: 0.0672879	total: 148ms	remaining: 69.9ms
340:	learn: 0.0671885	total: 149ms	remaining: 69.4ms
341:	learn: 0.0669832	total: 149ms	remaining: 69ms
342:	learn: 0.0669060	total: 150ms	remaining: 68.5ms
343:	learn: 0.0667714	total: 150ms	remaining: 68ms
344:	learn: 0.0667460	total: 150ms	remaining: 67.6ms
345:	learn: 0.0666137	total: 151ms	remaining: 67.1ms
346:	learn: 0.0665646	total: 151ms	remaining: 66.7ms
347:	learn: 0.0664999	total: 152ms	remaining: 66.2ms
348:	learn: 0.0664293	total: 152ms	remaining: 65.8ms
349:	learn: 0.0663663	total: 152ms	remaining: 65.3ms
350:	learn: 0.0662911	total: 153ms	remaining: 64.9ms
351:	learn: 0.0661880	total: 153ms	remaining: 64.4ms
352:	learn: 0.0660287	total: 154ms	remaining: 64ms
```

```
353:    learn: 0.0659336        total: 154ms    remaining: 63.5ms
354:    learn: 0.0658213        total: 154ms    remaining: 63.1ms
355:    learn: 0.0657013        total: 155ms    remaining: 62.6ms
356:    learn: 0.0655707        total: 155ms    remaining: 62.2ms
357:    learn: 0.0655565        total: 156ms    remaining: 61.7ms
358:    learn: 0.0655011        total: 156ms    remaining: 61.2ms
359:    learn: 0.0653639        total: 156ms    remaining: 60.8ms
360:    learn: 0.0652716        total: 157ms    remaining: 60.4ms
361:    learn: 0.0652601        total: 157ms    remaining: 59.9ms
362:    learn: 0.0651766        total: 158ms    remaining: 59.5ms
363:    learn: 0.0651568        total: 158ms    remaining: 59ms
364:    learn: 0.0650111        total: 158ms    remaining: 58.6ms
365:    learn: 0.0649914        total: 159ms    remaining: 58.1ms
366:    learn: 0.0648867        total: 159ms    remaining: 57.7ms
367:    learn: 0.0647691        total: 160ms    remaining: 57.2ms
368:    learn: 0.0646332        total: 160ms    remaining: 56.8ms
369:    learn: 0.0646237        total: 160ms    remaining: 56.3ms
370:    learn: 0.0645225        total: 161ms    remaining: 55.9ms
371:    learn: 0.0643725        total: 161ms    remaining: 55.4ms
372:    learn: 0.0642983        total: 162ms    remaining: 55ms
373:    learn: 0.0642885        total: 162ms    remaining: 54.5ms
374:    learn: 0.0641988        total: 162ms    remaining: 54.1ms
375:    learn: 0.0640445        total: 163ms    remaining: 53.7ms
376:    learn: 0.0639296        total: 163ms    remaining: 53.2ms
377:    learn: 0.0638441        total: 163ms    remaining: 52.8ms
378:    learn: 0.0636968        total: 164ms    remaining: 52.3ms
379:    learn: 0.0636285        total: 164ms    remaining: 51.9ms
380:    learn: 0.0635146        total: 165ms    remaining: 51.4ms
381:    learn: 0.0634862        total: 165ms    remaining: 51ms
382:    learn: 0.0634776        total: 165ms    remaining: 50.5ms
383:    learn: 0.0634538        total: 166ms    remaining: 50.1ms
384:    learn: 0.0633181        total: 166ms    remaining: 49.6ms
385:    learn: 0.0632090        total: 167ms    remaining: 49.2ms
386:    learn: 0.0631678        total: 167ms    remaining: 48.8ms
387:    learn: 0.0631024        total: 167ms    remaining: 48.3ms
388:    learn: 0.0629520        total: 168ms    remaining: 47.9ms
389:    learn: 0.0629237        total: 168ms    remaining: 47.4ms
390:    learn: 0.0629132        total: 169ms    remaining: 47ms
391:    learn: 0.0628938        total: 169ms    remaining: 46.5ms
392:    learn: 0.0627817        total: 169ms    remaining: 46.1ms
393:    learn: 0.0627630        total: 170ms    remaining: 45.7ms
394:    learn: 0.0627186        total: 170ms    remaining: 45.2ms
395:    learn: 0.0626612        total: 170ms    remaining: 44.8ms
396:    learn: 0.0624649        total: 171ms    remaining: 44.3ms
397:    learn: 0.0624555        total: 171ms    remaining: 43.9ms
398:    learn: 0.0622719        total: 172ms    remaining: 43.5ms
399:    learn: 0.0621624        total: 172ms    remaining: 43ms
400:    learn: 0.0620698        total: 172ms    remaining: 42.6ms
```

```
401:	learn: 0.0619857	total: 173ms	remaining: 42.1ms
402:	learn: 0.0618966	total: 173ms	remaining: 41.7ms
403:	learn: 0.0618416	total: 174ms	remaining: 41.3ms
404:	learn: 0.0617317	total: 174ms	remaining: 40.8ms
405:	learn: 0.0616471	total: 174ms	remaining: 40.4ms
406:	learn: 0.0615278	total: 175ms	remaining: 39.9ms
407:	learn: 0.0614422	total: 175ms	remaining: 39.5ms
408:	learn: 0.0614084	total: 176ms	remaining: 39.1ms
409:	learn: 0.0613949	total: 176ms	remaining: 38.6ms
410:	learn: 0.0613085	total: 176ms	remaining: 38.2ms
411:	learn: 0.0611664	total: 177ms	remaining: 37.8ms
412:	learn: 0.0611203	total: 177ms	remaining: 37.3ms
413:	learn: 0.0610671	total: 178ms	remaining: 36.9ms
414:	learn: 0.0609378	total: 178ms	remaining: 36.4ms
415:	learn: 0.0608414	total: 178ms	remaining: 36ms
416:	learn: 0.0607825	total: 179ms	remaining: 35.6ms
417:	learn: 0.0607278	total: 179ms	remaining: 35.1ms
418:	learn: 0.0605999	total: 180ms	remaining: 34.7ms
419:	learn: 0.0604808	total: 180ms	remaining: 34.3ms
420:	learn: 0.0604245	total: 180ms	remaining: 33.8ms
421:	learn: 0.0603501	total: 181ms	remaining: 33.4ms
422:	learn: 0.0602624	total: 181ms	remaining: 33ms
423:	learn: 0.0601689	total: 182ms	remaining: 32.5ms
424:	learn: 0.0600350	total: 182ms	remaining: 32.1ms
425:	learn: 0.0599535	total: 182ms	remaining: 31.7ms
426:	learn: 0.0599454	total: 183ms	remaining: 31.2ms
427:	learn: 0.0599039	total: 183ms	remaining: 30.8ms
428:	learn: 0.0597689	total: 183ms	remaining: 30.4ms
429:	learn: 0.0597066	total: 184ms	remaining: 29.9ms
430:	learn: 0.0596721	total: 184ms	remaining: 29.5ms
431:	learn: 0.0596171	total: 185ms	remaining: 29.1ms
432:	learn: 0.0594849	total: 185ms	remaining: 28.6ms
433:	learn: 0.0593782	total: 185ms	remaining: 28.2ms
434:	learn: 0.0593612	total: 186ms	remaining: 27.7ms
435:	learn: 0.0592721	total: 186ms	remaining: 27.3ms
436:	learn: 0.0592573	total: 186ms	remaining: 26.9ms
437:	learn: 0.0590963	total: 187ms	remaining: 26.4ms
438:	learn: 0.0589709	total: 187ms	remaining: 26ms
439:	learn: 0.0588972	total: 188ms	remaining: 25.6ms
440:	learn: 0.0588460	total: 188ms	remaining: 25.2ms
441:	learn: 0.0587844	total: 188ms	remaining: 24.7ms
442:	learn: 0.0586860	total: 189ms	remaining: 24.3ms
443:	learn: 0.0586775	total: 189ms	remaining: 23.9ms
444:	learn: 0.0585845	total: 190ms	remaining: 23.4ms
445:	learn: 0.0585538	total: 190ms	remaining: 23ms
446:	learn: 0.0584986	total: 190ms	remaining: 22.6ms
447:	learn: 0.0584351	total: 191ms	remaining: 22.1ms
448:	learn: 0.0583892	total: 191ms	remaining: 21.7ms
```

```
449:    learn: 0.0582947        total: 192ms    remaining: 21.3ms
450:    learn: 0.0581681        total: 192ms    remaining: 20.9ms
451:    learn: 0.0580522        total: 192ms    remaining: 20.4ms
452:    learn: 0.0579769        total: 193ms    remaining: 20ms
453:    learn: 0.0579641        total: 193ms    remaining: 19.6ms
454:    learn: 0.0579129        total: 194ms    remaining: 19.1ms
455:    learn: 0.0578812        total: 194ms    remaining: 18.7ms
456:    learn: 0.0577979        total: 194ms    remaining: 18.3ms
457:    learn: 0.0576452        total: 195ms    remaining: 17.9ms
458:    learn: 0.0575459        total: 195ms    remaining: 17.4ms
459:    learn: 0.0574926        total: 195ms    remaining: 17ms
460:    learn: 0.0574239        total: 196ms    remaining: 16.6ms
461:    learn: 0.0573261        total: 196ms    remaining: 16.1ms
462:    learn: 0.0573016        total: 197ms    remaining: 15.7ms
463:    learn: 0.0572923        total: 197ms    remaining: 15.3ms
464:    learn: 0.0571525        total: 198ms    remaining: 14.9ms
465:    learn: 0.0571299        total: 198ms    remaining: 14.4ms
466:    learn: 0.0571070        total: 198ms    remaining: 14ms
467:    learn: 0.0570410        total: 199ms    remaining: 13.6ms
468:    learn: 0.0569051        total: 199ms    remaining: 13.2ms
469:    learn: 0.0568072        total: 200ms    remaining: 12.7ms
470:    learn: 0.0567259        total: 200ms    remaining: 12.3ms
471:    learn: 0.0566775        total: 200ms    remaining: 11.9ms
472:    learn: 0.0566140        total: 201ms    remaining: 11.5ms
473:    learn: 0.0565948        total: 201ms    remaining: 11ms
474:    learn: 0.0565817        total: 202ms    remaining: 10.6ms
475:    learn: 0.0564952        total: 202ms    remaining: 10.2ms
476:    learn: 0.0564860        total: 202ms    remaining: 9.76ms
477:    learn: 0.0563526        total: 203ms    remaining: 9.33ms
478:    learn: 0.0563435        total: 203ms    remaining: 8.9ms
479:    learn: 0.0562452        total: 203ms    remaining: 8.48ms
480:    learn: 0.0561724        total: 204ms    remaining: 8.05ms
481:    learn: 0.0560920        total: 204ms    remaining: 7.63ms
482:    learn: 0.0559455        total: 205ms    remaining: 7.2ms
483:    learn: 0.0558619        total: 205ms    remaining: 6.78ms
484:    learn: 0.0557850        total: 205ms    remaining: 6.35ms
485:    learn: 0.0557517        total: 206ms    remaining: 5.93ms
486:    learn: 0.0557035        total: 206ms    remaining: 5.5ms
487:    learn: 0.0555446        total: 207ms    remaining: 5.08ms
488:    learn: 0.0554460        total: 207ms    remaining: 4.66ms
489:    learn: 0.0553907        total: 207ms    remaining: 4.23ms
490:    learn: 0.0553813        total: 208ms    remaining: 3.81ms
491:    learn: 0.0552869        total: 208ms    remaining: 3.38ms
492:    learn: 0.0552775        total: 209ms    remaining: 2.96ms
493:    learn: 0.0552221        total: 209ms    remaining: 2.54ms
494:    learn: 0.0551432        total: 209ms    remaining: 2.11ms
495:    learn: 0.0550586        total: 210ms    remaining: 1.69ms
496:    learn: 0.0550073        total: 210ms    remaining: 1.27ms
```

```
497:    learn: 0.0549060        total: 211ms    remaining: 845us
498:    learn: 0.0548913        total: 211ms    remaining: 422us
499:    learn: 0.0548346        total: 211ms    remaining: 0us
```

[596]:
```
<catboost.core.CatBoostRegressor at 0x7f6cc009afd0>
```

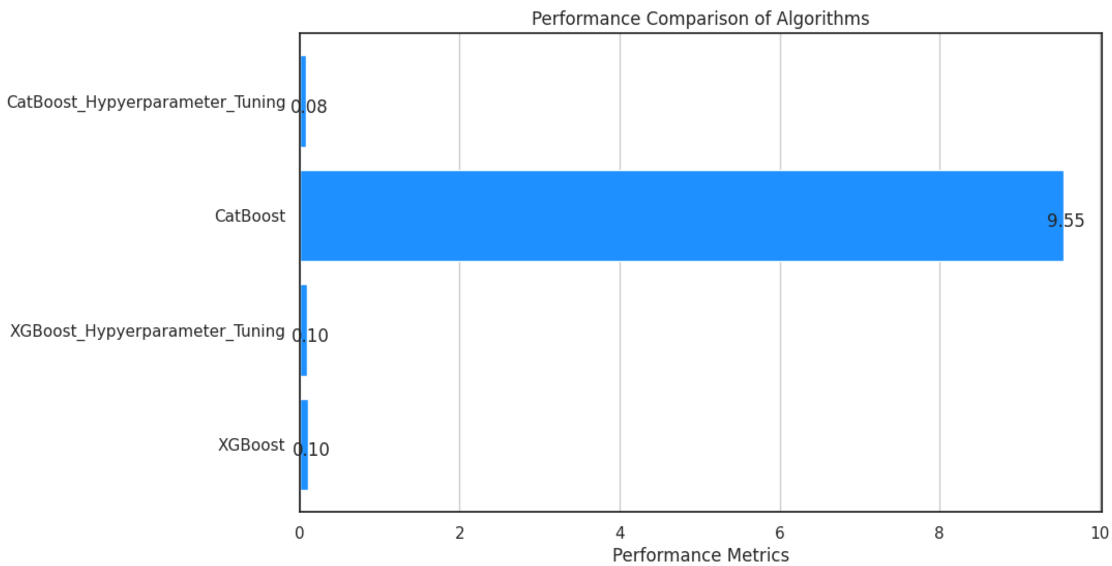
[7]:
```
cat_train_pred = best_model.predict(X_train)
print(f'CatBoost Train RMSE {rmse(y_train, xgb_train_pred)}')

cat_pred = best_model.predict(X_test)
print(f'CatBoost Test RMSE {rmse(y_test, xgb_pred)}')
```

```
CatBoost Train RMSE 0.04
CatBoost Test RMSE 0.12
```

5 Result Analysis

Performance Comparison

[637]:
```
result = {
    'XGBoost': [0.09, 0.12],
    'XGBoost_Hypyerparameter_Tuning': [0.06,0.13],
    'CatBoost': [9.53, 9.57],
    'CatBoost_Hypyerparameter_Tuning': [0.04,0.12],
}

labels = list(result.keys())
values = [np.mean(v) for v in result.values()]

plt.figure(figsize=(10, 6))
bars = plt.barh(labels, values, color='dodgerblue')

for bar in bars:
    plt.text(bar.get_width() - 0.2, bar.get_y() + bar.get_height()/2 - 0.1,
 ↪f"{bar.get_width():.2f}")

plt.xlabel('Performance Metrics')
plt.title('Performance Comparison of Algorithms')
plt.grid(axis='x')

plt.show()
```



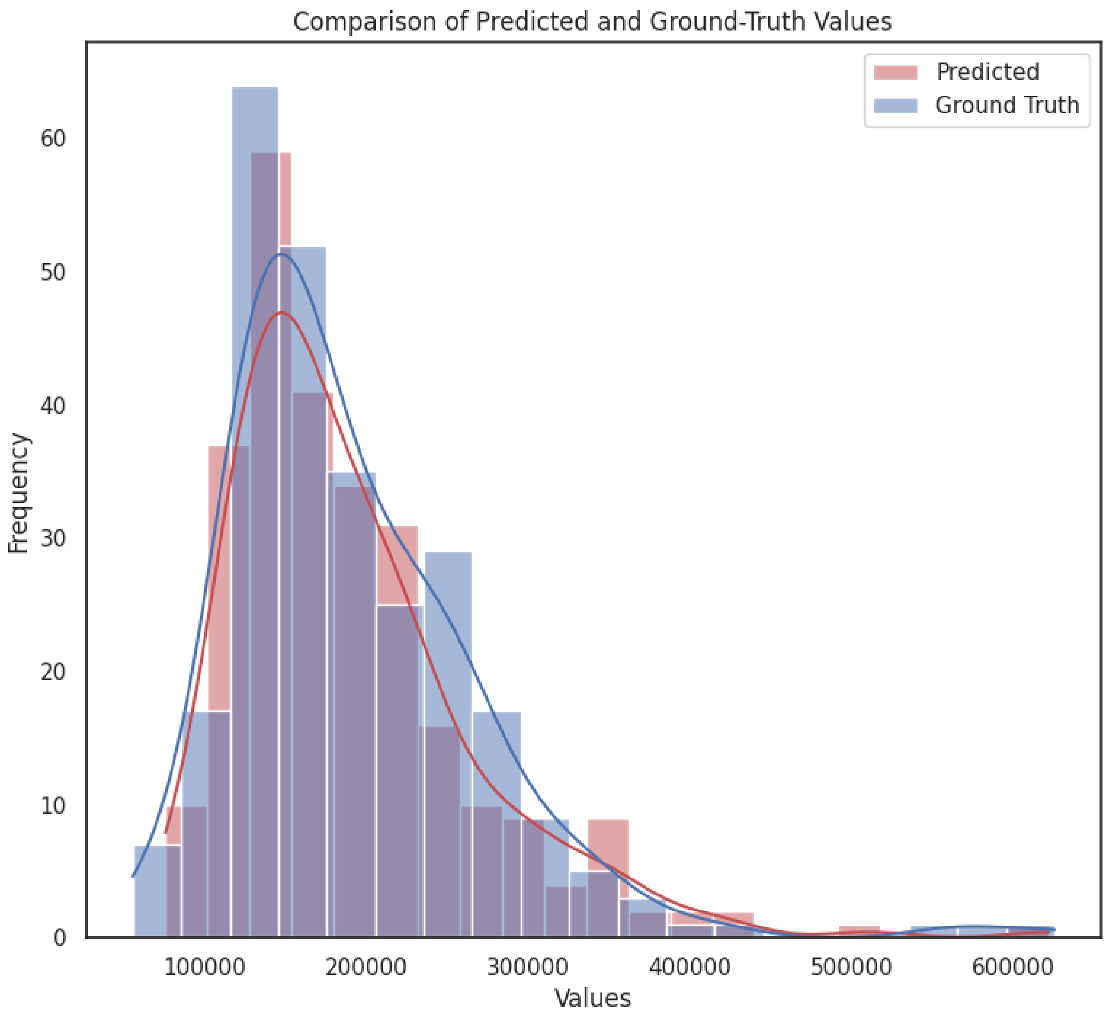
I have run experiments on XGBoost and CatBoost which both have strong benefits in both categorical and numerical data. - Without Hyperparameter Tuning, - XGBoost achieves training RMSE 0.09 and testing RMSE 0.12. The model can be improved with better hyperparameters. - CatBoost achieves training RMSE 9.53 and testing RMSE 9.57, which is relatively higher than XGBoost. The model is underfitting. - With Hyperparameter Tuning, - XGBoost has slightly lower training RMSE 0.06 but testing RMSE 0.13. The model is overfit and doesn't seem to generalize on the training data. It might have been memorizing it. - CatBoost achieves significantly higher performance than the one without tuning. CatBoost training RMSE 0.04 and testing RMSE 0.12.

Let's compare the prediction and ground truth.

[642]:
```
predictions = best_model.predict(X_test)
predictions = np.expm1(predictions)
```

[644]:
```
y_test_reversed =  np.expm1(y_test)
```

[650]:
```
plt.figure(figsize=(9, 8))
sns.histplot(predictions, kde=True, label='Predicted', color='r')
sns.histplot(y_test_reversed, kde=True, label='Ground Truth', color='b')

plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Comparison of Predicted and Ground-Truth Values')
plt.legend()

# Show the plot
plt.show()
```



The plot shows that the trend of prediction is similar to the ground truth.

6 Conclusion and Further Work

In the future, I want to use K-nearest neighbors to replace missing values so that we do not need to drop many data points. I will perform more hyperparameter tuning. I also want to explore with different supervised learning algorithms. I want see the performance of simple linear regression.

[ ]:

```

```