##**Project Name: House Prices- Advance Regression Techniques**

The main aim of this project is to predict the house price based on various features

Dataset can be downloaded from the link below (might need accept the competition to download it). https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

Following the below lifecycle for this project

1. Data Analaysis
2. Feature Engineering
3. Feature Scaling
4. Model Training and Testing
5. Baseline Model for submission

Extracting data directly from kaggle, we will need kaggle.json file which we can upload to our notebook directory

```
! pip install -q kaggle
from google.colab import files
files.upload()
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
<IPython.core.display.HTML object>
```

```
Saving kaggle.json to kaggle.json
```

```
#link for downloading the dataset from kaggle
! kaggle competitions download -c house-prices-advanced-regression-techniques
```

```
Downloading house-prices-advanced-regression-techniques.zip to /content
  0% 0.00/199k [00:00<?, ?B/s]
100% 199k/199k [00:00<00:00, 20.8MB/s]
```

```
#As the downloaded file is in zip format extracting all the file in the directory
import zipfile
zip_ref =
zipfile.ZipFile('/content/house-prices-advanced-regression-techniques.zip', 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

```
!pip install -q pycaret
```

```
                              486.1/486.1 kB 4.4 MB/s eta 0:00:00
                              302.2/302.2 kB 9.2 MB/s eta 0:00:00
                              13.4/13.4 MB 26.4 MB/s eta 0:00:00
                              163.8/163.8 kB 11.1 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
                              258.3/258.3 kB 14.7 MB/s eta 0:00:00
                              81.9/81.9 kB 2.0 MB/s eta 0:00:00
                              194.1/194.1 kB 16.1 MB/s eta 0:00:00
                              79.9/79.9 MB 6.9 MB/s eta 0:00:00
                              106.8/106.8 kB 6.5 MB/s eta 0:00:00
                              80.7/80.7 kB 4.5 MB/s eta 0:00:00
                              21.8/21.8 MB 19.1 MB/s eta 0:00:00
                              44.0/44.0 kB 2.8 MB/s eta 0:00:00
                              2.1/2.1 MB 30.8 MB/s eta 0:00:00
                              130.1/130.1 kB 9.1 MB/s eta 0:00:00
                              12.1/12.1 MB 38.3 MB/s eta 0:00:00
                              1.6/1.6 MB 20.3 MB/s eta 0:00:00
                              7.5/7.5 MB 29.6 MB/s eta 0:00:00
                              141.1/141.1 kB 10.7 MB/s eta 0:00:00
                              2.1/2.1 MB 41.7 MB/s eta 0:00:00
  Building wheel for pyod (setup.py) ... done
```

```
#import required libraries for data analysis
import pandas as pd
import numpy as np
import calendar
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sn

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
from IPython.display import display


from sklearn.preprocessing import OneHotEncoder #for Converting Ordinal
categorical features to numerical
from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression, BayesianRidge, SGDRegressor,
Lasso, Ridge, ElasticNet, HuberRegressor, OrthogonalMatchingPursuit
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import GradientBoostingRegressor
```

```
from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegressor
from sklearn.ensemble import AdaBoostRegressor, BaggingRegressor
import lightgbm as lgb

from sklearn.neural_network import MLPRegressor
from xgboost import XGBRegressor

from pycaret.regression import setup , compare_models
```

```
#loading the dataset
#As we can, we have train and test dataset present separately, and to perform EDA
we will combine it
df_train = pd.read_csv('/content/train.csv')
df_test = pd.read_csv('/content/test.csv')
df_full = pd.concat([df_train, df_test], ignore_index=True)
```

```
print("Shape of train dataset :" , df_train.shape)
print("Shape of test dataset :" , df_test.shape)
print("Shape of concatenated :" , df_full.shape)
```

```
Shape of train dataset : (1460, 81)
Shape of test dataset : (1459, 80)
Shape of concatenated : (2919, 81)
```

```
df_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub

```
df_test.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilit
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPu
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPu
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPu
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPu
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPu

```
df_full.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub

```
df_full.tail()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	U
2914	2915	160	RM	21.0	1936	Pave	NaN	Reg	Lvl	A
2915	2916	160	RM	21.0	1894	Pave	NaN	Reg	Lvl	A
2916	2917	20	RL	160.0	20000	Pave	NaN	Reg	Lvl	A
2917	2918	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	A
2918	2919	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	A

EDA(Exploratory Data Analysis)

Below are the steps been taken for analysis

1. Understanding the data
2. Missing Values
3. Obtaining all the numerical Variables
4. Distribution of Numberical Variables with Target variable
5. Obtaining all the Categorical Variables
6. Distribution of Categorical Variables with Target variable
7. Outlier Detection
8. **Understanding the data**

```
#Checking all the columns to understand varibales present in our dataset, taking
only training dataset as I need to compare the target variables
df = df_train.copy()
```

```
#Getting statistical information of features
df.describe()
```

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000

```
print(len(df['Id']))
#As we can see ID is unique identifier so we can remove it from the dataset later
or set it as index but I decided to drop it.
```

```
1460
```

```
df.drop('Id', axis=1, inplace=True)
df_full.drop('Id', axis=1, inplace=True)
df_train.drop('Id', axis=1, inplace=True)
test_id = df_test['Id']
df_test.drop('Id', axis=1, inplace=True)
```

```
df['MSSubClass'] = df['MSSubClass'].astype(str)
```

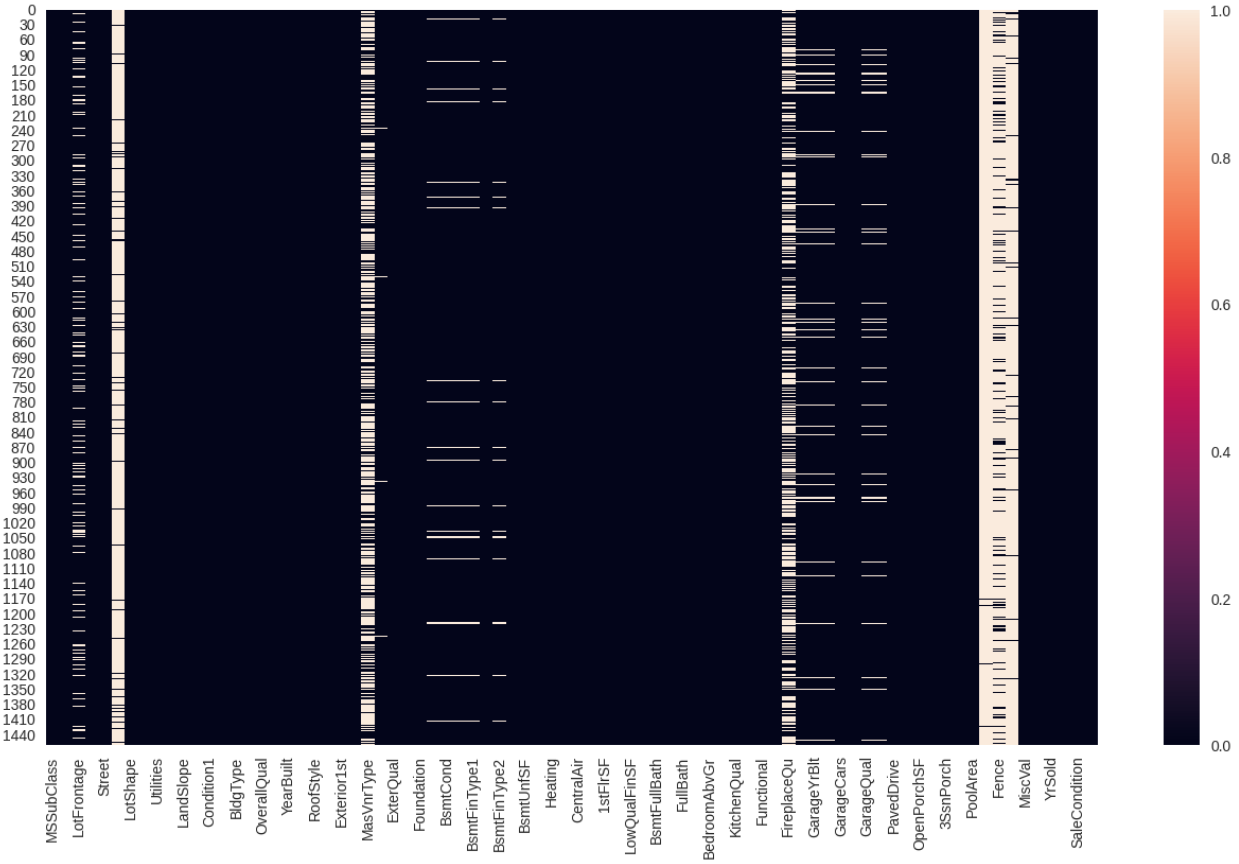
2. **Missing Values**

```
# We can also see there are null values persent in the dataset, Let see what how
many null values are present
# Creating list of feature which have null(nan) values in percentage for better
understanding
Feature_with_na = [features for features in df.columns if
df[features].isna().sum()]

print ("Number of variables having missing values :" , len(Feature_with_na))
for features in Feature_with_na:
  print(features, np.round(df[features].isna().sum()/len(df[features])*100,4), '
  % missing values')
```

```
Number of variables having missing values : 19
LotFrontage 17.7397  % missing values
Alley 93.7671  % missing values
MasVnrType 59.726  % missing values
MasVnrArea 0.5479  % missing values
BsmtQual 2.5342  % missing values
BsmtCond 2.5342  % missing values
```

```
BsmtExposure 2.6027  % missing values
BsmtFinType1 2.5342  % missing values
BsmtFinType2 2.6027  % missing values
Electrical 0.0685  % missing values
FireplaceQu 47.2603  % missing values
GarageType 5.5479  % missing values
GarageYrBlt 5.5479  % missing values
GarageFinish 5.5479  % missing values
GarageQual 5.5479  % missing values
GarageCond 5.5479  % missing values
PoolQC 99.5205  % missing values
Fence 80.7534  % missing values
MiscFeature 96.3014  % missing values
```

```
#Visualizing the missing values
plt.figure(figsize=(16,9))
sn.heatmap(df.isna())
```

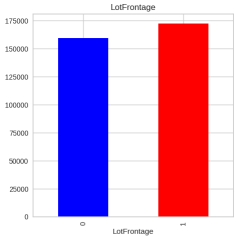


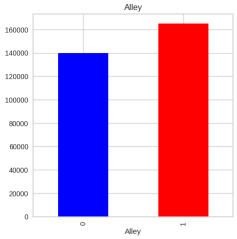
Since there are alot of missing values, we will need to check if there is a relationship between missing values with Target Variable, which in this case is Sales Price

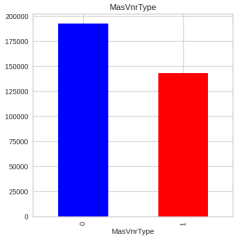
Let's analyze it with diagram for better understanding

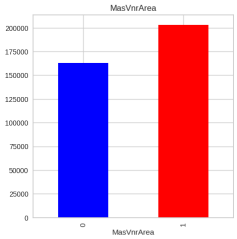
As we can observe clearly that the missing values has relationship with Target variable (Sales Price).

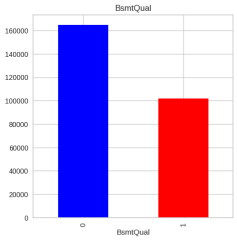
```
# Calculating the number of rows and columns for subplots, as I want 3 columns
and rest will be with number of features present
n_plots = len(Feature_with_na)
n_cols = 3
n_rows = -(-n_plots // n_cols)

# Creating the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Flattening the axs array if more than one row
if n_rows > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# Defining colors for bars representing 1s and 0s
colors = {0: 'blue', 1: 'red'}

# Ploting each feature
for idx, feature in enumerate(Feature_with_na):

    # Copy the dataset so the changes are not reflected in actual dataset
    data_for_null = df.copy()

    # Creating a variable indicating missing values
    data_for_null[feature] = np.where(data_for_null[feature].isna(), 1, 0)

    # Calculating the median SalePrice where information is missing or present
    plot_data = data_for_null.groupby(feature)['SalePrice'].median()

    # Creating the bar plot
    plot_data.plot.bar(ax=axs[idx], color=[colors[val] for val in
plot_data.index])
    axs[idx].set_title(feature)

# Hiding any unused subplots
for i in range(n_plots, len(axs)):
    fig.delaxes(axs[i])

plt.subplots_adjust(wspace=0.4, hspace=0.6)
plt.tight_layout(pad=2.0)
plt.show()
```

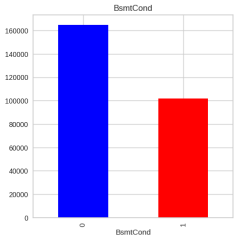

LotFrontage
175000
150000
125000
100000
75000
50000
25000
0
0
1
LotFrontage

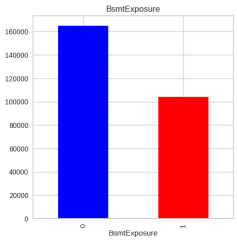

Alley
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
Alley

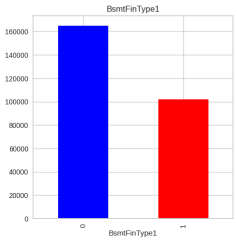

MasVnrType
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
MasVnrType

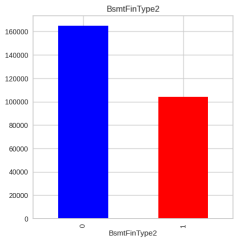

MasVnrArea
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
MasVnrArea

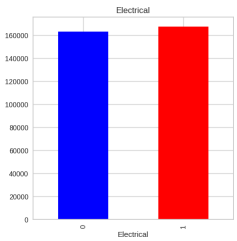

BsmtQual
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtQual

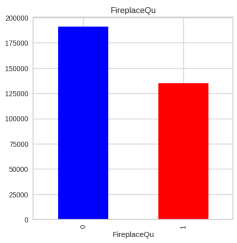

BsmtCond
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtCond

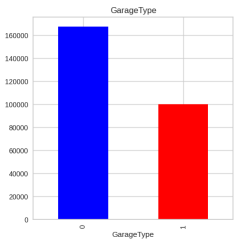

BsmtExposure
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtExposure

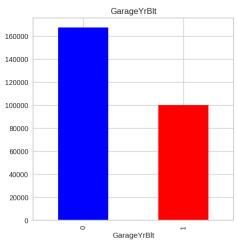

BsmtFinType1
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtFinType1

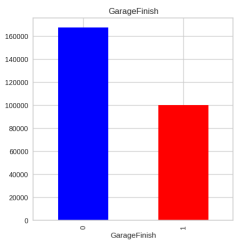

BsmtFinType2
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
BsmtFinType2

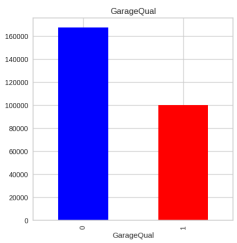

Electrical
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
Electrical

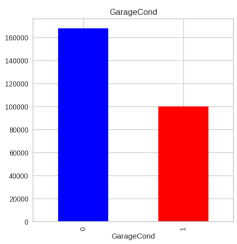

FireplaceQu
200000
175000
150000
125000
100000
75000
50000
25000
0
0
1
FireplaceQu

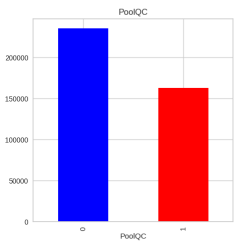

GarageType
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageType

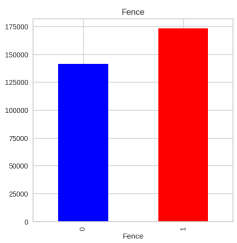

GarageYrBlt
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageYrBlt

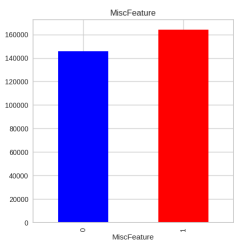

GarageFinish
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageFinish


GarageQual
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageQual

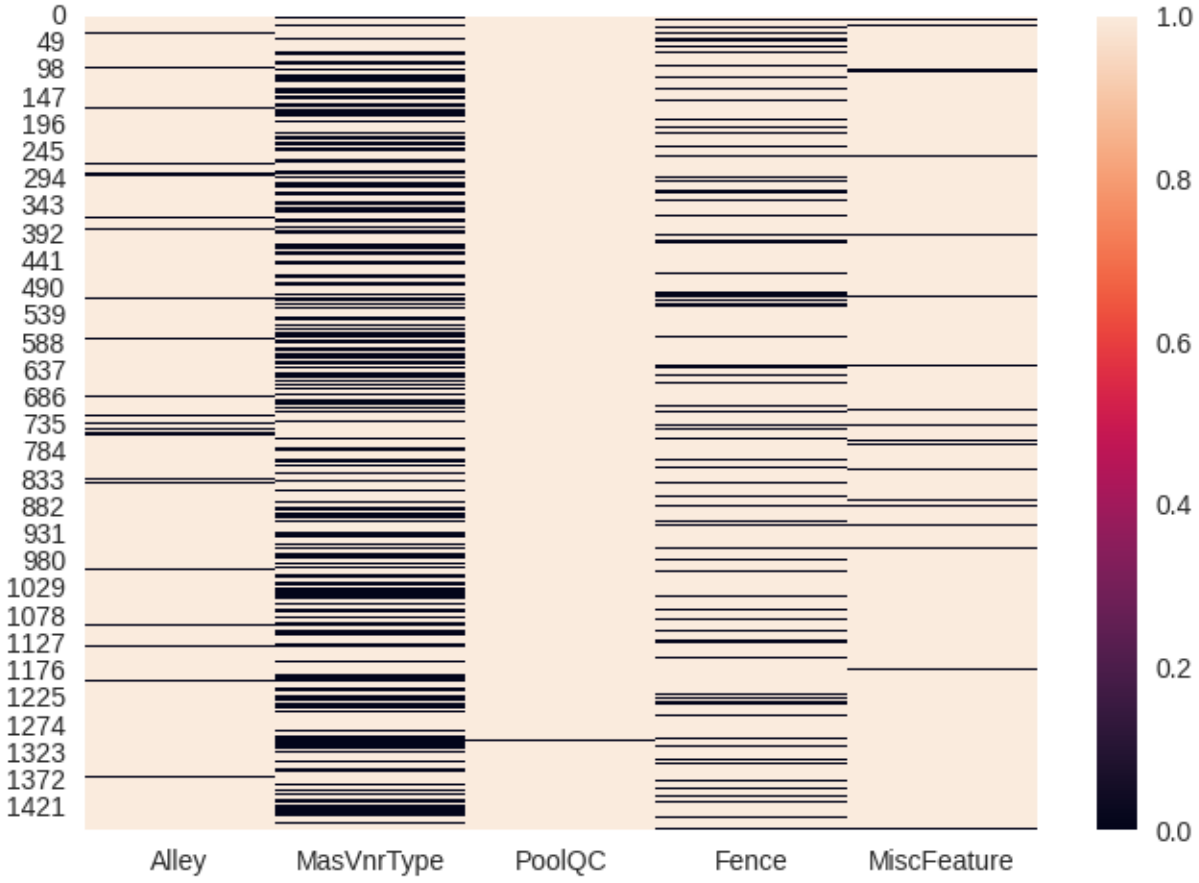

GarageCond
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
GarageCond


PoolQC
200000
150000
100000
50000
0
0
1
PoolQC


Fence
175000
150000
125000
100000
75000
50000
25000
0
0
1
Fence

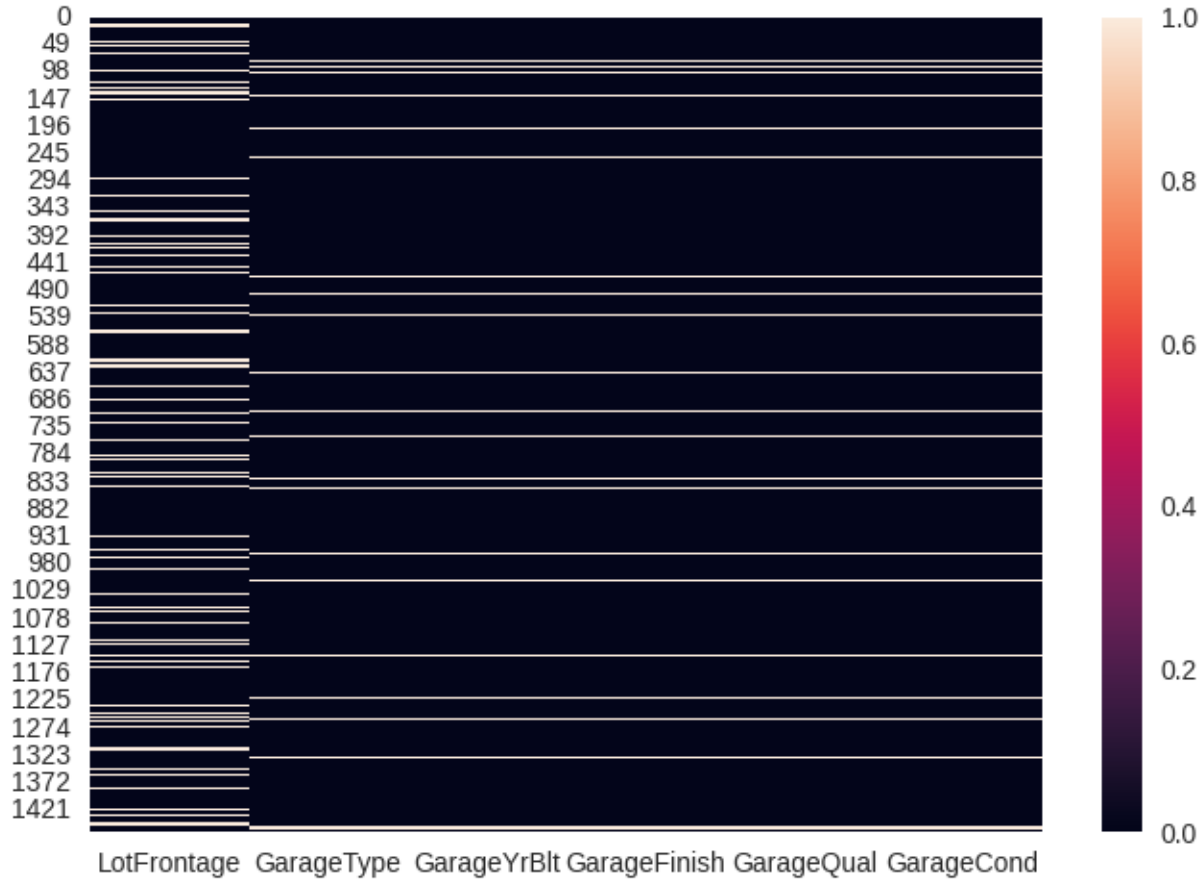

MiscFeature
160000
140000
120000
100000
80000
60000
40000
20000
0
0
1
MiscFeature

```
#Let's check if we need to drop any feature where more than 50% of data is
missing
Features_nan = np.round(df.isna().sum()/len(df[features])*100,4)
Feature_with_nan_more_than_50 = Features_nan[Features_nan > 50]
Feature_with_nan_more_than_50
sn.heatmap(df[Feature_with_nan_more_than_50.keys()].isna())
```

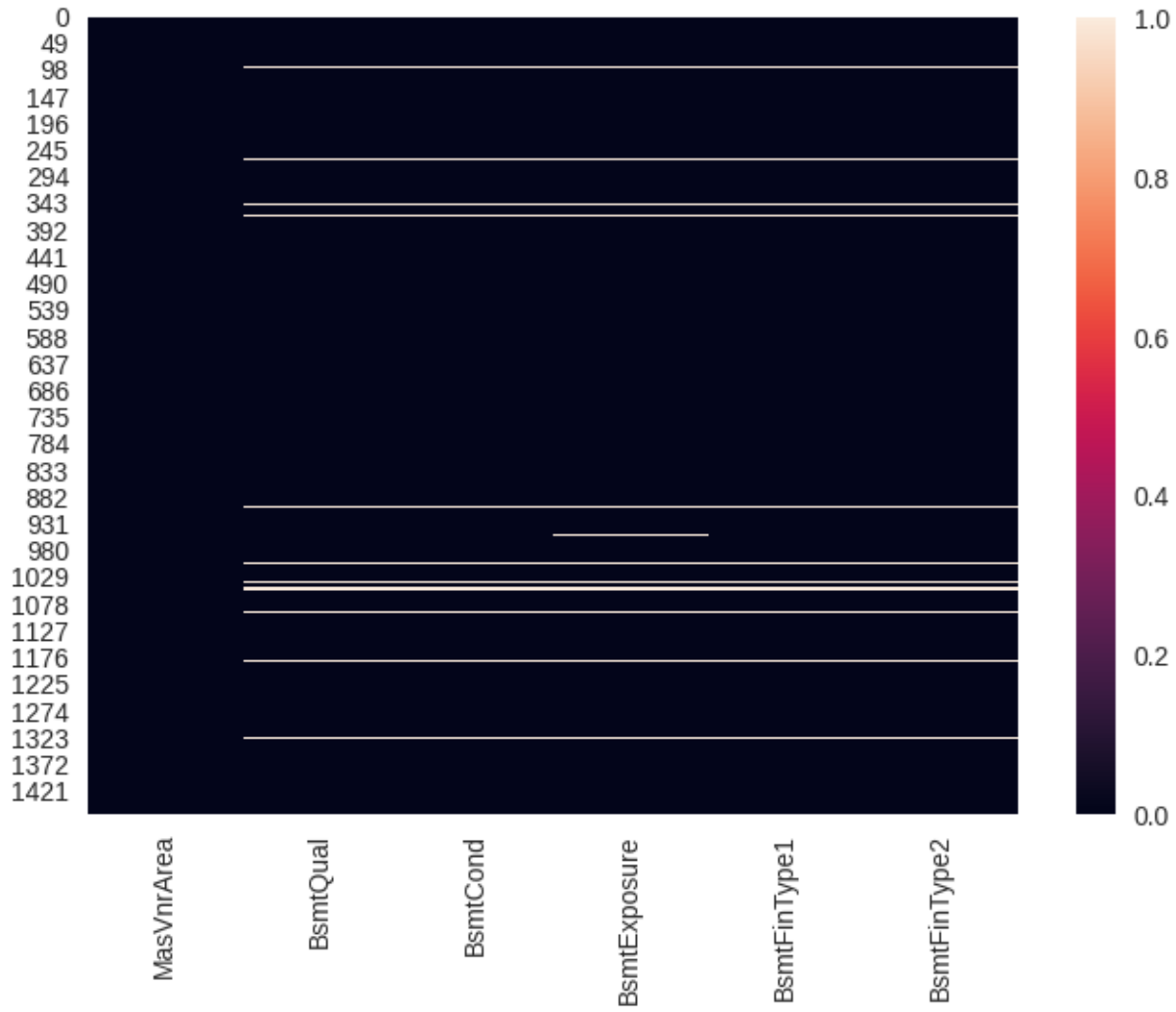



```
#Let's check if we need to drop any feature where missing data is between 20 and
50
Feature_with_nan_bw_20_50 = Features_nan[(Features_nan < 50 ) & (Features_nan >
20)]
Feature_with_nan_bw_20_50
sn.heatmap(df[Feature_with_nan_bw_20_50.keys()].isna())
```


```
#Let's check if we need to drop any feature where missing data is between 5 and
20
Feature_with_nan_bw_5_20 = Features_nan[(Features_nan < 20 ) & (Features_nan >
5)]
Feature_with_nan_bw_5_20
sn.heatmap(df[Feature_with_nan_bw_5_20.keys()].isna())
```




```
Feature_with_nan_below_5 = Features_nan[(Features_nan < 5 ) & (Features_nan >
0.5)]
Feature_with_nan_below_5
sn.heatmap(df[Feature_with_nan_below_5.keys()].isna())
```



After checking the NA values description in the dataset from the provided documentation we can see that the NA is label for some specific feature is not present but it was captured so cannot drop the feature. All the features having missing values as NA as label will be replaced as constant value and other will be replaced with imputation

3. **Performing Analysis on Numerical Variables**

```
#list of numerical values
numerical_features = [features for features  in df.columns if df[features].dtypes
!='O' ]
print('Number of numerical features:' , len(numerical_features))

df[numerical_features].head()
```

```
Number of numerical features: 36
```

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFin
0	65.0	8450	7	5	2003	2003	196.0	706
1	80.0	9600	6	8	1976	1976	0.0	978
2	68.0	11250	7	5	2001	2002	162.0	486
3	60.0	9550	7	5	1915	1970	0.0	216
4	84.0	14260	8	5	2000	2000	350.0	655

```
#Let analyze the sales price over the Year

#Obtaining the features with year
year_features =[features for features in numerical_features if 'Yr' in features
or 'Year' in features]

year_features
```

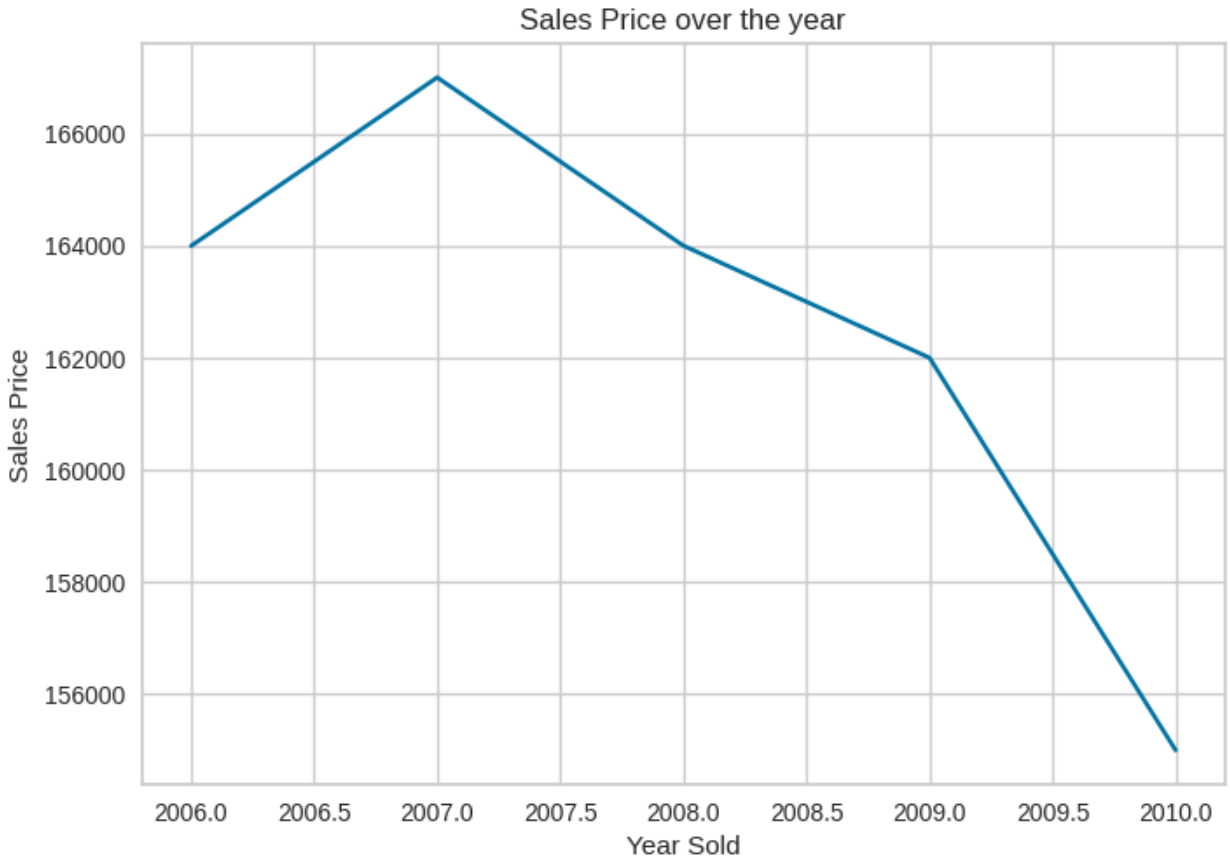
```
['YearBuilt', 'YearRemodAdd', 'GarageYrBlt', 'YrSold']
```

```
#getting unique values of year features
for features in year_features:
  print(features, len(df[features].unique()))
```

```
YearBuilt 112
YearRemodAdd 61
GarageYrBlt 98
YrSold 5
```

```
#From above, lets check the realtionship of Year Sold with Sales Price
df.groupby('YrSold')['SalePrice'].median().plot()
plt.xlabel('Year Sold')
plt.ylabel('Sales Price')
plt.title('Sales Price over the year')
```

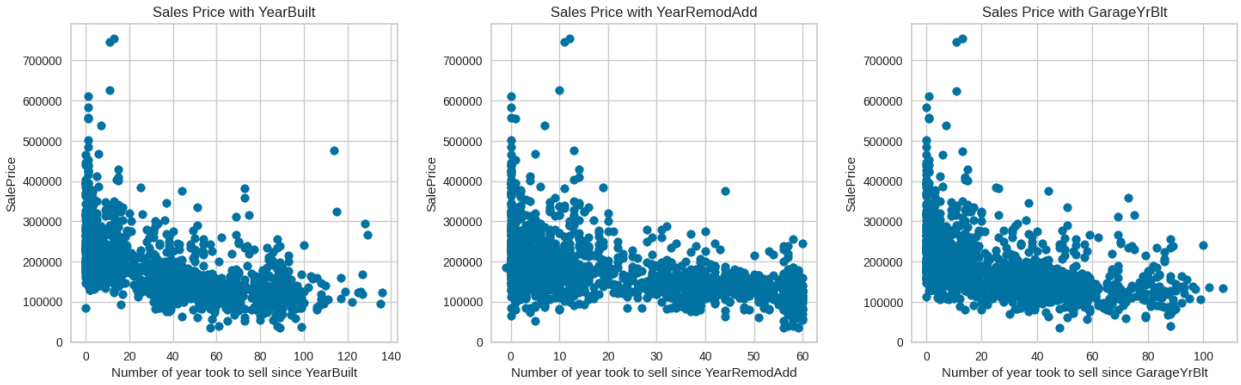
```
Text(0.5, 1.0, 'Sales Price over the year')
```



From above, looks like the house price decrease over the year which is usually not the case

```
# Calculating the number of rows and columns for subplots
n_plots = len(year_features)-1
n_cols = n_plots  # All plots in a single row
n_rows = 1

# Creating the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Ploting chart for each feature
for idx, feature in enumerate(year_features):
    if feature != 'YrSold':
        # Copying the dataset so we do not make any changes to actuall dataset
        data_for_year = df.copy()

        # Calculating the difference
        data_for_year[feature] = data_for_year['YrSold'] - data_for_year[feature]

        # Scatter plot for each feature
        axs[idx].scatter(data_for_year[feature], data_for_year['SalePrice'])
```

```
        axs[idx].set_xlabel(f'Number of year took to sell since {feature}')
        axs[idx].set_ylabel('SalePrice')
        axs[idx].set_title(f'Sales Price with {feature}')

# Adjusting layout
plt.subplots_adjust(wspace=0.4)
plt.tight_layout(pad=2.0)
plt.show()
```



4. **Distribution of Numberical Variables with Target variable Numerical Varibales**

```
#There are usually 2 type of Numerical varibales (Discrete and Continous)

#Getting Discrete variable
discrete_variables = [features for features in numerical_features if
len(df[features].unique())<25]
discrete_variables
```

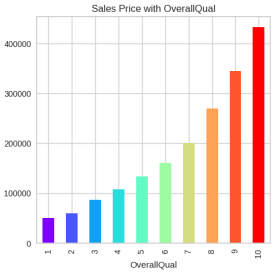
```
['OverallQual',
 'OverallCond',
 'LowQualFinSF',
 'BsmtFullBath',
 'BsmtHalfBath',
 'FullBath',
 'HalfBath',
 'BedroomAbvGr',
 'KitchenAbvGr',
 'TotRmsAbvGrd',
 'Fireplaces',
 'GarageCars',
 '3SsnPorch',
 'PoolArea',
 'MiscVal',
 'MoSold',
 'YrSold']
```

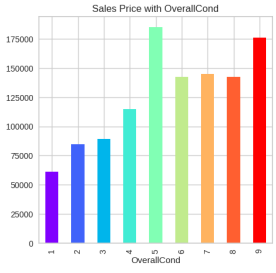
```
df[discrete_variables].head()
```

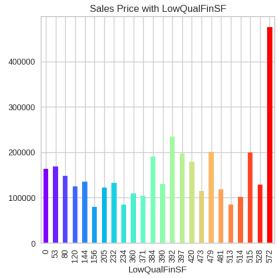
	OverallQual	OverallCond	LowQualFinSF	BsmtFullBath	BsmtHalfBath	FullBath	HalfBath	Bedro
0	7	5	0	1	0	2	1	3
1	6	8	0	0	1	2	0	3
2	7	5	0	1	0	2	1	3
3	7	5	0	1	0	1	0	3
4	8	5	0	1	0	2	1	4

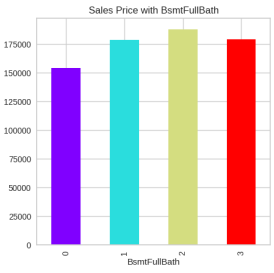
```
# Calculating the number of rows and columns for subplots, as I want 3 columns
and rest will be with number of feat
n_plots = len(discrete_variables)
n_cols = 3
n_rows = -(-n_plots // n_cols)

# Creating the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Flattening the axs array if more than one row
if n_rows > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# Ploting chart for  each feature
for idx, feature in enumerate(discrete_variables):

    # Copying the dataset
    data_for_discrete = df.copy()

    # Calculating the median SalePrice with discrete variables
    plot_data = data_for_discrete.groupby(feature)['SalePrice'].median()

    # Creating a colormap for better visuals
    colors = cm.rainbow(np.linspace(0, 1, len(plot_data)))

    # Creating the bar plot
    plot_data.plot.bar(ax=axs[idx], color = colors)
    axs[idx].set_title(f'Sales Price with {feature}')


# Hiding any unused subplots
for i in range(n_plots, len(axs)):
    fig.delaxes(axs[i])
```

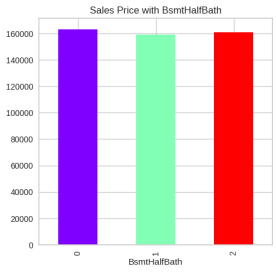
```
plt.subplots_adjust(wspace=0.4, hspace=0.6)
plt.tight_layout(pad=2.0)
plt.show()
```


Sales Price with OverallQual
OverallQual


Sales Price with OverallCond
OverallCond


Sales Price with LowQualFinSF
LowQualFinSF

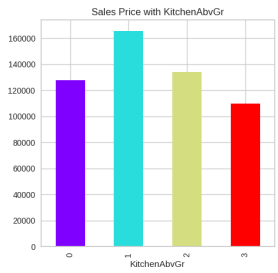

Sales Price with BsmtFullBath
BsmtFullBath

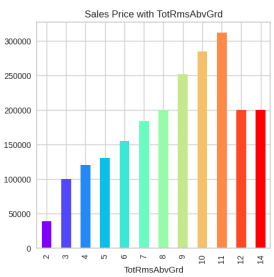

Sales Price with BsmtHalfBath
BsmtHalfBath

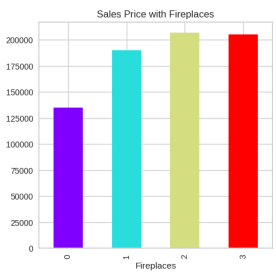

Sales Price with FullBath
FullBath

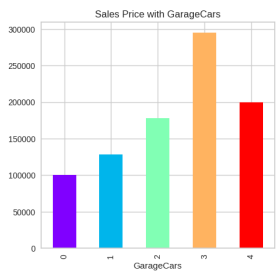

Sales Price with HalfBath
HalfBath

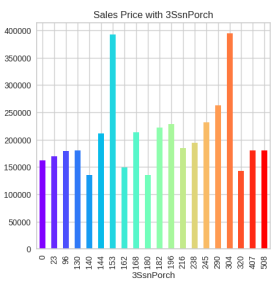

Sales Price with BedroomAbvGr
BedroomAbvGr


Sales Price with KitchenAbvGr
KitchenAbvGr

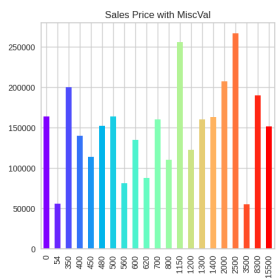

Sales Price with TotRmsAbvGrd
TotRmsAbvGrd

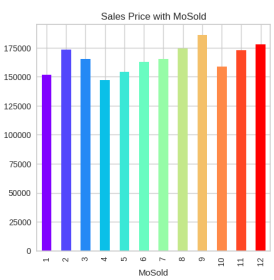

Sales Price with Fireplaces
Fireplaces

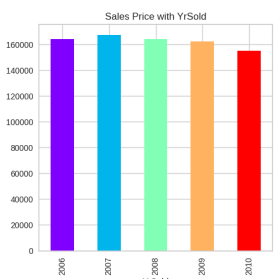

Sales Price with GarageCars
GarageCars


Sales Price with 3SsnPorch
3SsnPorch


Sales Price with PoolArea
PoolArea


Sales Price with MiscVal
MiscVal


Sales Price with MoSold
MoSold

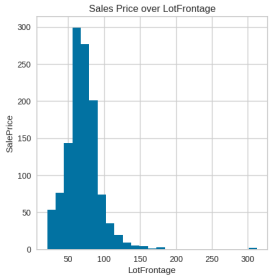

Sales Price with YrSold
YrSold

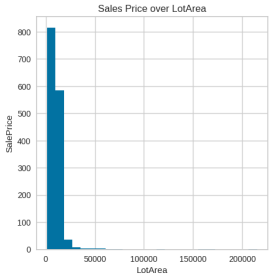
Continous Variable

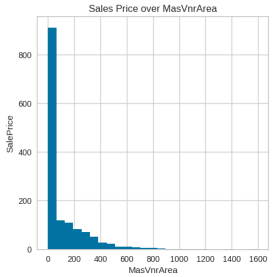
```
#Obtaining all the continous variables
continous_varibales = [features for features in numerical_features if features
not in discrete_variables+year_features]

print('Count of Continous varibables :{}'.format(len(continous_varibales)))
```

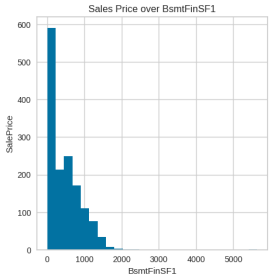
```
Count of Continous varibables :16
```

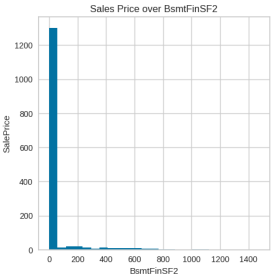
```
# Calculating the number of rows and columns for subplots, as I want 3 columns
and rest will be with number of feat
n_plots = len(continous_varibales)
n_cols = 3
n_rows = -(-n_plots // n_cols)

# CreatING the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Flatteing the axs array if more than one row
if n_rows > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# Ploting chart for each feature
for idx, feature in enumerate(continous_varibales):
    # Copy the dataset
    data_for_continous = df.copy()

    # Creating the histogram
    data_for_continous[feature].plot.hist(ax=axs[idx], bins=25)
    axs[idx].set_xlabel(feature)
    axs[idx].set_ylabel('SalePrice')
    axs[idx].set_title(f'Sales Price over {feature}')

# Hiding any unused subplots
for i in range(n_plots, len(axs)):
    fig.delaxes(axs[i])

plt.subplots_adjust(wspace=0.4, hspace=0.6)
plt.tight_layout(pad=2.0)
plt.show()
```

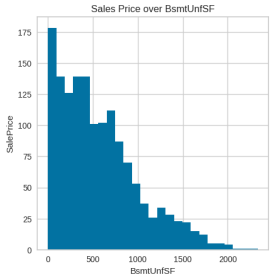

Sales Price over LotFrontage
SalePrice
LotFrontage

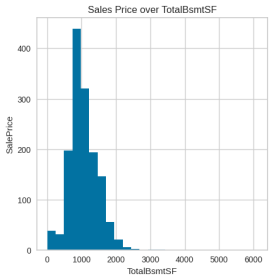

Sales Price over LotArea
SalePrice
LotArea


Sales Price over MasVnrArea
SalePrice
MasVnrArea


Sales Price over BsmtFinSF1
SalePrice
BsmtFinSF1

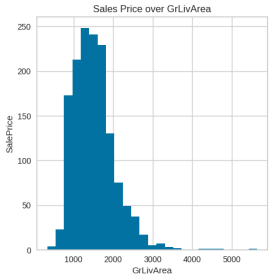

Sales Price over BsmtFinSF2
SalePrice
BsmtFinSF2

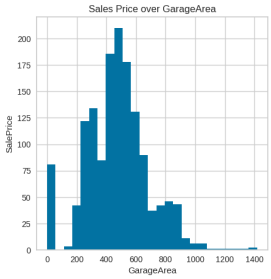

Sales Price over BsmtUnfSF
SalePrice
BsmtUnfSF

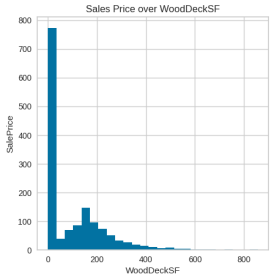

Sales Price over TotalBsmtSF
SalePrice
TotalBsmtSF

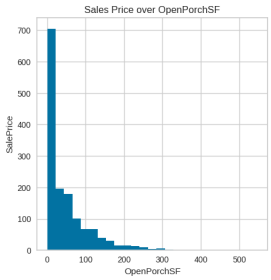

Sales Price over 1stFlrSF
SalePrice
1stFlrSF

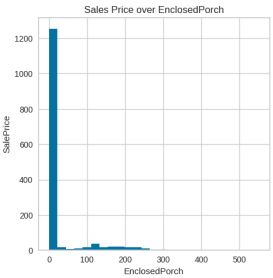

Sales Price over 2ndFlrSF
SalePrice
2ndFlrSF

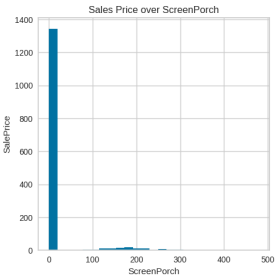

Sales Price over GrLivArea
SalePrice
GrLivArea


Sales Price over GarageArea
SalePrice
GarageArea


Sales Price over WoodDeckSF
SalePrice
WoodDeckSF


Sales Price over OpenPorchSF
SalePrice
OpenPorchSF


Sales Price over EnclosedPorch
SalePrice
EnclosedPorch


Sales Price over ScreenPorch
SalePrice
ScreenPorch


Sales Price over SalePrice
SalePrice
SalePrice

5. **Categorical Variables**

```
categorical_features = [features for features in df.columns if
df[features].dtypes =='O']

categorical_features
```

```
['MSSubClass',
 'MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'Utilities',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'ExterQual',
 'ExterCond',
 'Foundation',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Heating',
 'HeatingQC',
 'CentralAir',
 'Electrical',
 'KitchenQual',
 'Functional',
 'FireplaceQu',
 'GarageType',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PavedDrive',
 'PoolQC',
```

```
 'Fence',
 'MiscFeature',
 'SaleType',
 'SaleCondition']
```

```
df[categorical_features].head()
```

	MSSubClass	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Ne
0	60	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Co
1	20	RL	Pave	NaN	Reg	Lvl	AllPub	FR2	Gtl	Ve
2	60	RL	Pave	NaN	IR1	Lvl	AllPub	Inside	Gtl	Co
3	70	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Cr
4	60	RL	Pave	NaN	IR1	Lvl	AllPub	FR2	Gtl	No

```
for features in categorical_features:
  print('Feature is {} and the number of categories are {}:'.format(features,
  len(df[features].unique())))
```

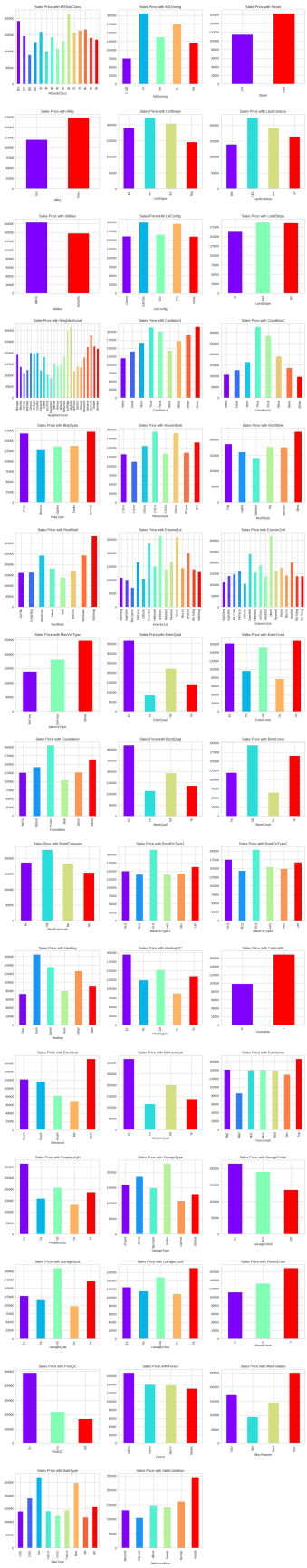
```
Feature is MSSubClass and the number of categories are 15:
Feature is MSZoning and the number of categories are 5:
Feature is Street and the number of categories are 2:
Feature is Alley and the number of categories are 3:
Feature is LotShape and the number of categories are 4:
Feature is LandContour and the number of categories are 4:
Feature is Utilities and the number of categories are 2:
Feature is LotConfig and the number of categories are 5:
Feature is LandSlope and the number of categories are 3:
Feature is Neighborhood and the number of categories are 25:
Feature is Condition1 and the number of categories are 9:
Feature is Condition2 and the number of categories are 8:
Feature is BldgType and the number of categories are 5:
Feature is HouseStyle and the number of categories are 8:
Feature is RoofStyle and the number of categories are 6:
Feature is RoofMatl and the number of categories are 8:
Feature is Exterior1st and the number of categories are 15:
Feature is Exterior2nd and the number of categories are 16:
Feature is MasVnrType and the number of categories are 4:
Feature is ExterQual and the number of categories are 4:
Feature is ExterCond and the number of categories are 5:
Feature is Foundation and the number of categories are 6:
Feature is BsmtQual and the number of categories are 5:
Feature is BsmtCond and the number of categories are 5:
Feature is BsmtExposure and the number of categories are 5:
Feature is BsmtFinType1 and the number of categories are 7:
Feature is BsmtFinType2 and the number of categories are 7:
```

```
Feature is Heating and the number of categories are 6:
Feature is HeatingQC and the number of categories are 5:
Feature is CentralAir and the number of categories are 2:
Feature is Electrical and the number of categories are 6:
Feature is KitchenQual and the number of categories are 4:
Feature is Functional and the number of categories are 7:
Feature is FireplaceQu and the number of categories are 6:
Feature is GarageType and the number of categories are 7:
Feature is GarageFinish and the number of categories are 4:
Feature is GarageQual and the number of categories are 6:
Feature is GarageCond and the number of categories are 6:
Feature is PavedDrive and the number of categories are 3:
Feature is PoolQC and the number of categories are 4:
Feature is Fence and the number of categories are 5:
Feature is MiscFeature and the number of categories are 5:
Feature is SaleType and the number of categories are 9:
Feature is SaleCondition and the number of categories are 6:
```

6. **Distribution of Numberical Variables with Target variable**

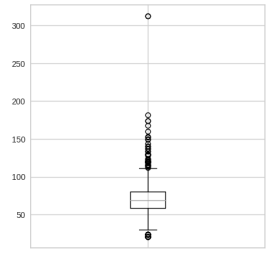
```
# Calculating the number of rows and columns for subplots, as I want 3 columns
and rest will be with number of feat
n_plots = len(categorical_features)
n_cols = 3
n_rows = -(-n_plots // n_cols)

# Creating the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Flattening the axs array if more than one row
if n_rows > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# Ploting chart for  each feature
for idx, feature in enumerate(categorical_features):

    # Copying the dataset
    data_for_discrete = df.copy()

    # Calculating the median SalePrice with discrete variables
    plot_data = data_for_discrete.groupby(feature)['SalePrice'].median()

    # Creating a colormap
    colors = cm.rainbow(np.linspace(0, 1, len(plot_data)))
```

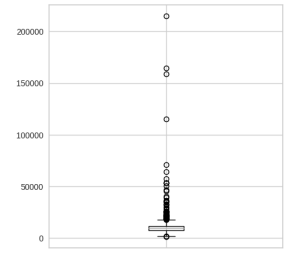
```
    # Creating the bar plot
    plot_data.plot.bar(ax=axs[idx], color = colors)
    axs[idx].set_title(f'Sales Price with {feature}')


# Hide any unused subplots
for i in range(n_plots, len(axs)):
    fig.delaxes(axs[i])

plt.subplots_adjust(wspace=0.4, hspace=0.6)
plt.tight_layout(pad=2.0)
plt.show()
```

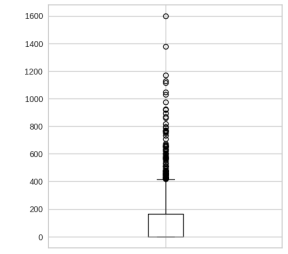

Sales Price with MSSubClass
Sales Price with MSZoning
Sales Price with Street
Sales Price with Alley
Sales Price with LotShape
Sales Price with LandContour
Sales Price with Utilities
Sales Price with LotConfig
Sales Price with LandSlope
Sales Price with NeighborHood
Sales Price with Condition1
Sales Price with Condition2
Sales Price with BldgType
Sales Price with HouseStyle
Sales Price with RoofStyle
Sales Price with RoofMatl
Sales Price with Exterior1st
Sales Price with Exterior2nd
Sales Price with MasVnrType
Sales Price with ExterQual
Sales Price with ExterCond
Sales Price with Foundation
Sales Price with BsmtQual
Sales Price with BsmtCond
Sales Price with BsmtExposure
Sales Price with BsmtFinType1
Sales Price with BsmtFinType2
Sales Price with Heating
Sales Price with HeatingQC
Sales Price with CentralAir
Sales Price with Electrical
Sales Price with KitchenQual
Sales Price with Functional
Sales Price with FireplaceQu
Sales Price with GarageType
Sales Price with GarageFinish
Sales Price with GarageQual
Sales Price with GarageCond
Sales Price with PavedDrive
Sales Price with PoolQC
Sales Price with Fence
Sales Price with MiscFeature
Sales Price with SaleType
Sales Price with SaleCondition

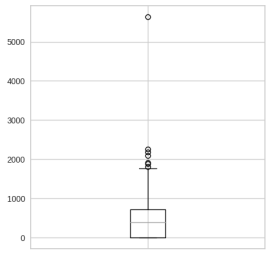
7. **Outliers**

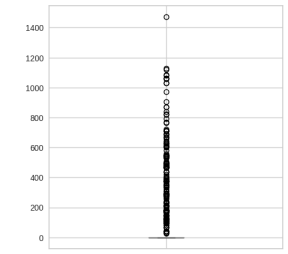
```
# Calculating the number of rows and columns for subplots, as I want 3 columns
and rest will be with number of feat
n_plots = len(continous_varibales)
n_cols = 3
n_rows = -(-n_plots // n_cols)

# Creating the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Flattening the axs array if more than one row
if n_rows > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# Ploting chart for each feature
for idx, feature in enumerate(continous_varibales):
    # Copying the dataset
    data_for_continous = df.copy()

    # Creating the histogram
    data_for_continous.boxplot(column = [feature], ax= axs[idx])
    axs[idx].set_xlabel(feature)

# Hiding any unused subplots
for i in range(n_plots, len(axs)):
    fig.delaxes(axs[i])

plt.subplots_adjust(wspace=0.4, hspace=0.6)
plt.tight_layout(pad=2.0)
plt.show()
```

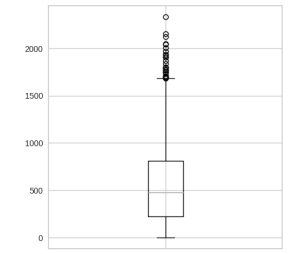

LotFrontage
LotFrontage

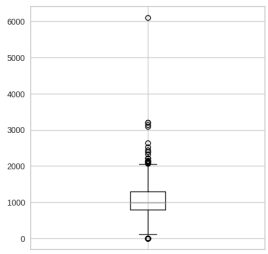

LotArea
LotArea

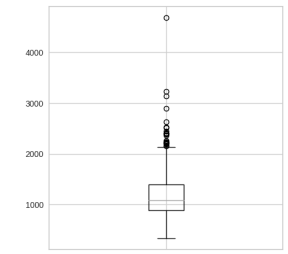

MasVnrArea
MasVnrArea

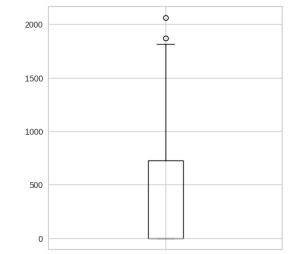

BsmtFinSF1
BsmtFinSF1

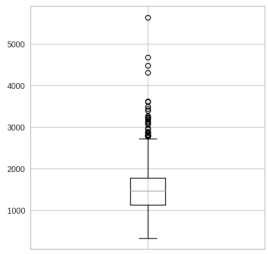

BsmtFinSF2
BsmtFinSF2

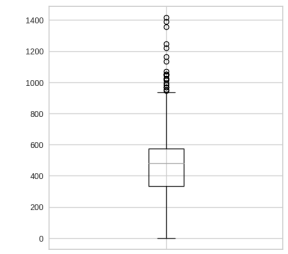

BsmtUnfSF
BsmtUnfSF

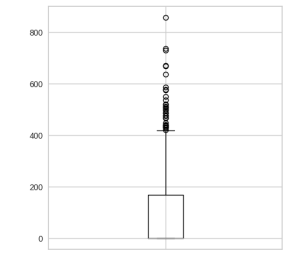

TotalBsmtSF
TotalBsmtSF

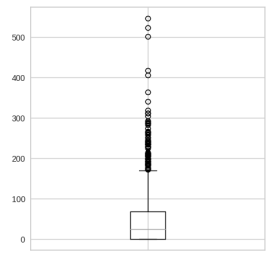

1stFlrSF
1stFlrSF

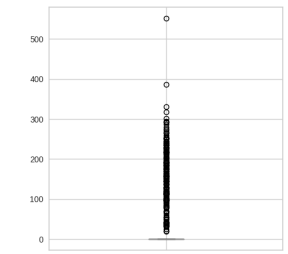

2ndFlrSF
2ndFlrSF

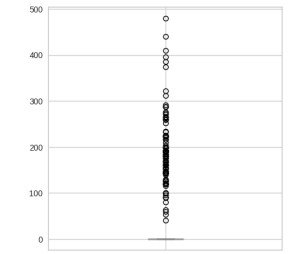

GrLivArea
GrLivArea

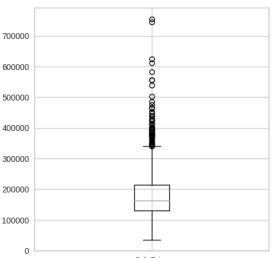

GarageArea
GarageArea


WoodDeckSF
WoodDeckSF

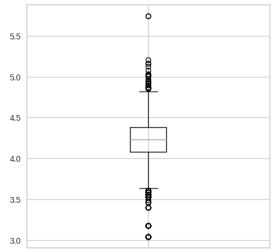

OpenPorchSF
OpenPorchSF

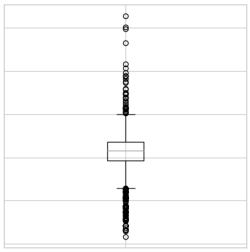

EnclosedPorch
EnclosedPorch

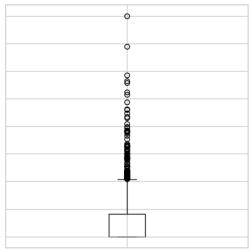

ScreenPorch
ScreenPorch

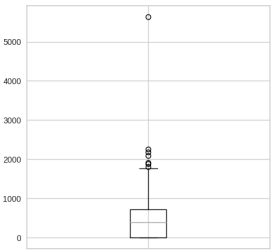

SalePrice
SalePrice

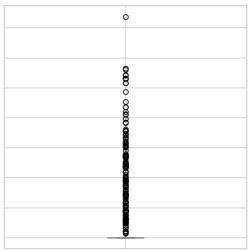
```
# Calculating the number of rows and columns for subplots, as I want 3 columns
and rest will be with number of feat
n_plots = len(continous_varibales)
n_cols = 3
n_rows = -(-n_plots // n_cols)

# Creating the subplots
fig, axs = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5 * n_cols, 5 *
n_rows))

# Flattening the axs array if more than one row
if n_rows > 1:
    axs = axs.flatten()
else:
    axs = [axs]

# Ploting for each feature
for idx, feature in enumerate(continous_varibales):
    # Copying the dataset
    data_for_continous = df.copy()

    if 0 in data_for_continous[feature].unique():
      pass
    else:
      data_for_continous[feature]= np.log(data_for_continous[feature])

    # Creating the histogram
    data_for_continous.boxplot(column = [feature], ax= axs[idx])
    axs[idx].set_xlabel(feature)

# Hiding any unused subplots
for i in range(n_plots, len(axs)):
    fig.delaxes(axs[i])

plt.subplots_adjust(wspace=0.4, hspace=0.6)
plt.tight_layout(pad=2.0)
plt.show()
```

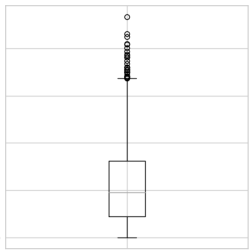

LotFrontage
LotFrontage

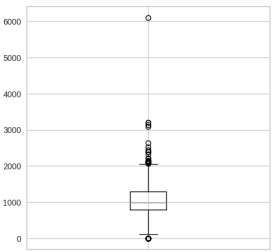

LotArea
LotArea

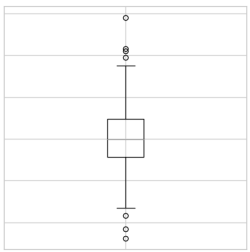

MasVnrArea
MasVnrArea

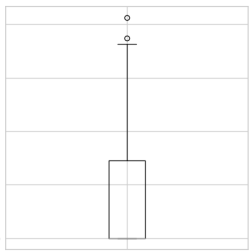

BsmtFinSF1
BsmtFinSF1

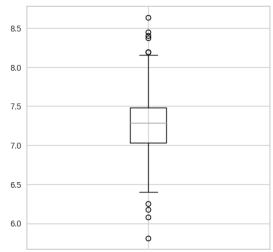

BsmtFinSF2
BsmtFinSF2

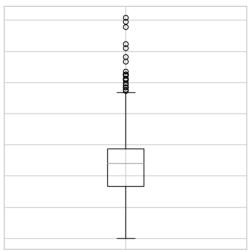

BsmtUnfSF
BsmtUnfSF

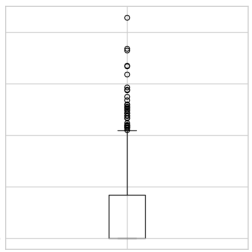

TotalBsmtSF
TotalBsmtSF

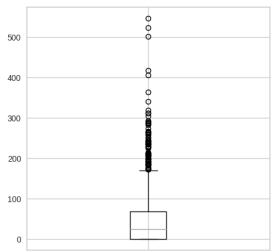

1stFlrSF
1stFlrSF

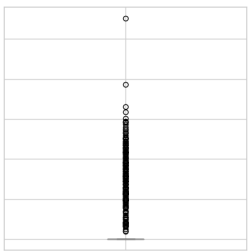

2ndFlrSF
2ndFlrSF

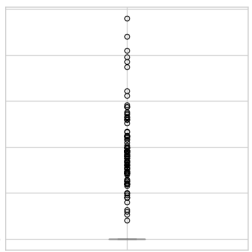

GrLivArea
GrLivArea

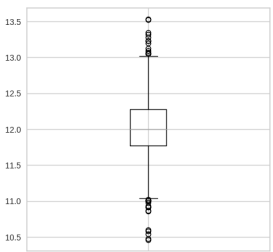

GarageArea
GarageArea


WoodDeckSF
WoodDeckSF


OpenPorchSF
OpenPorchSF


EnclosedPorch
EnclosedPorch


ScreenPorch
ScreenPorch


SalePrice
SalePrice

Feature Engineering

Below are the steps followed for feature Engineering (not in the same order) 1. Handling Missing Values 2. Temporal variables 3. Converting Numerical Features to Categorical features (if required) 4. Converting Categorical features in numerical features 5. Converting Nominal Categorgies using OneHot Encoding (we can use any encoding method)

```
#making sure until to keep a copy of dataset which does not effect anything to
the original dataset
df_FE = df_full.copy()
```

```
df_FE['MSZoning'].isna().sum()
```

4

1. **Handling Missing Values**

Categorical Variables

```
Missing_categorical_features = [feature for feature in
df_FE[categorical_features] if df_FE[feature].isna().sum()]

for feature in Missing_categorical_features:
  print("{} : {}% missing values".format(feature,np.round(df_FE[feature].isna().
  sum()/len(df_FE[feature])*100,4)))
```

```
MSZoning : 0.137% missing values
Alley : 93.2169% missing values
Utilities : 0.0685% missing values
Exterior1st : 0.0343% missing values
Exterior2nd : 0.0343% missing values
MasVnrType : 60.5002% missing values
BsmtQual : 2.7749% missing values
BsmtCond : 2.8092% missing values
BsmtExposure : 2.8092% missing values
BsmtFinType1 : 2.7064% missing values
BsmtFinType2 : 2.7407% missing values
Electrical : 0.0343% missing values
KitchenQual : 0.0343% missing values
Functional : 0.0685% missing values
FireplaceQu : 48.6468% missing values
GarageType : 5.3786% missing values
GarageFinish : 5.4471% missing values
GarageQual : 5.4471% missing values
GarageCond : 5.4471% missing values
PoolQC : 99.6574% missing values
Fence : 80.4385% missing values
```

```
MiscFeature : 96.4029% missing values
SaleType : 0.0343% missing values
```

```
Constant_Label_cat_features = ['Alley',
                               'BsmtQual',
                               'BsmtCond',
                               'BsmtExposure',
                               'BsmtFinType1',
                               'BsmtFinType2',
                               'Electrical',
                               'FireplaceQu',
                               'GarageType',
                               'GarageFinish' ,
                               'GarageQual',
                               'GarageCond',
                               'PoolQC',
                               'Fence',
                               'MiscFeature' ]
Mode_Label_cat_features = ['MSZoning',
                           'MasVnrType',
                           'Utilities',
                           'Exterior1st',
                           'Exterior2nd',
                           'KitchenQual',
                           'Functional',
                           'SaleType']
```

```
#Replacing all the missing (nan) values with a label,

for features in Constant_Label_cat_features:
  df_FE[features]= df_FE[features].fillna('NA')

for features in Mode_Label_cat_features:
  df_FE[features]= df_FE[features].fillna(df_FE[features].mode()[0])

df_FE[Missing_categorical_features].isna().sum()
```

```
MSZoning        0
Alley           0
Utilities       0
Exterior1st     0
Exterior2nd     0
MasVnrType      0
BsmtQual        0
BsmtCond        0
BsmtExposure    0
BsmtFinType1    0
```

```
BsmtFinType2    0
Electrical      0
KitchenQual     0
Functional      0
FireplaceQu     0
GarageType      0
GarageFinish    0
GarageQual      0
GarageCond      0
PoolQC          0
Fence           0
MiscFeature     0
SaleType        0
dtype: int64
```

Numerical Variables

```
Missing_numerical_features = [feature for feature in df_FE.columns if
df_FE[feature].dtypes != 'O' and df_FE[feature].isna().sum() > 0]

for feature in Missing_numerical_features:
  print("{} : {}% missing
  values".format(feature,np.round(df_FE[feature].isna().mean(),4)))
```

```
LotFrontage : 0.1665% missing values
MasVnrArea : 0.0079% missing values
BsmtFinSF1 : 0.0003% missing values
BsmtFinSF2 : 0.0003% missing values
BsmtUnfSF : 0.0003% missing values
TotalBsmtSF : 0.0003% missing values
BsmtFullBath : 0.0007% missing values
BsmtHalfBath : 0.0007% missing values
GarageYrBlt : 0.0545% missing values
GarageCars : 0.0003% missing values
GarageArea : 0.0003% missing values
SalePrice : 0.4998% missing values
```

```
#Replacing all the missing (nan) values with a label
for feature in Missing_numerical_features:
  median_values = df_FE[feature].median()

  df_FE[feature+'nan'] = np.where(df_FE[feature].isna(),1,0)
  df_FE[feature].fillna(median_values, inplace = True)

df_FE[Missing_numerical_features].isna().sum()
```

```
LotFrontage     0
MasVnrArea      0
```

```
BsmtFinSF1      0
BsmtFinSF2      0
BsmtUnfSF       0
TotalBsmtSF     0
BsmtFullBath    0
BsmtHalfBath    0
GarageYrBlt     0
GarageCars      0
GarageArea      0
SalePrice       0
dtype: int64
```

```
df_FE.columns.isna().sum()
```

```
0
```

2. **Temporal Variables**

```
for feature in ['YearBuilt', 'YearRemodAdd', 'GarageYrBlt']:
  df_FE[feature] = df_FE['YrSold']-df_FE[feature]
```

```
df_FE.head()
```

	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	Lo
0	60	RL	65.0	8450	Pave	NA	Reg	Lvl	AllPub	In
1	20	RL	80.0	9600	Pave	NA	Reg	Lvl	AllPub	Fl
2	60	RL	68.0	11250	Pave	NA	IR1	Lvl	AllPub	In
3	70	RL	60.0	9550	Pave	NA	IR1	Lvl	AllPub	Co
4	60	RL	84.0	14260	Pave	NA	IR1	Lvl	AllPub	Fl

3. **Converting Numerical Features to Categorical features**

```
num_conv = ['MSSubClass','YrSold' ,'MoSold' ]
```

```
df_FE['MoSold'] = df_FE['MoSold'].apply(lambda x : calendar.month_abbr[x])
```

```
df_FE['MoSold'].value_counts()
```

```
MoSold
Jun    503
Jul    446
May    394
Apr    279
Aug    233
```

```
Mar    232
Oct    173
Sep    158
Nov    142
Feb    133
Jan    122
Dec    104
Name: count, dtype: int64
```

```
for feature in num_conv:
  df_FE[feature] =  df_FE[feature].astype(str)
```

4. **Converting Categorical features in numerical features**

```
order_mappings = {
    'ExterQual': ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'ExterCond': ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'BsmtQual': ['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'BsmtCond': ['NA', 'Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'BsmtExposure': ['NA', 'No', 'Mn', 'Av', 'Gd'],
    'BsmtFinType1': ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ', 'ALQ', 'GLQ'],
    'BsmtFinType2': ['NA', 'Unf', 'LwQ', 'Rec', 'BLQ', 'ALQ', 'GLQ'],
    'HeatingQC': ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'KitchenQual': ['Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'FireplaceQu': ['NA','Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'GarageQual': ['NA','Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'GarageCond': ['NA','Po', 'Fa', 'TA', 'Gd', 'Ex'],
    'PoolQC': ['NA', 'Fa', 'TA', 'Gd', 'Ex'],
    'Functional': ['Sal', 'Sev', 'Maj2', 'Maj1', 'Mod', 'Min2', 'Min1', 'Typ'],
    'GarageFinish': ['NA', 'Unf', 'RFn', 'Fin'],
    'Fence': ['NA', 'MnWw', 'GdWo', 'MnPrv', 'GdPrv'],
    'PavedDrive': ['N', 'P', 'Y'],
    'CentralAir': ['N', 'Y'],
    'Utilities': ['ELO', 'NoSeWa', 'NoSewr', 'AllPub']
}
```

```
# Function to apply ordered label encoding for specific columns
def apply_label_encoding(df, order_mappings):
    df_encoded = df_FE.copy()
    for col, order_list in order_mappings.items():
        df_encoded[col] = df_FE[col].apply(lambda x: order_list.index(x))
    return df_encoded
```

```
# Apply ordered label encoding to specific columns
df_encoded_ordered = apply_label_encoding(df_FE, order_mappings)
```

```
df_encoded_ordered['Utilities'].value_counts()
```

```
Utilities
3    2918
1       1
Name: count, dtype: int64
```

```
df_FE['Utilities'].value_counts()
```

```
Utilities
AllPub    2918
NoSeWa       1
Name: count, dtype: int64
```

```
df_encoded_ordered.dtypes
```

```
MSSubClass          object
MSZoning            object
LotFrontage        float64
LotArea              int64
Street              object
Alley               object
LotShape            object
LandContour         object
Utilities            int64
LotConfig           object
LandSlope           object
Neighborhood        object
Condition1          object
Condition2          object
BldgType            object
HouseStyle          object
OverallQual          int64
OverallCond          int64
YearBuilt            int64
YearRemodAdd         int64
RoofStyle           object
RoofMatl            object
Exterior1st         object
Exterior2nd         object
MasVnrType          object
MasVnrArea         float64
ExterQual            int64
ExterCond            int64
Foundation          object
BsmtQual             int64
```

```
BsmtCond             int64
BsmtExposure         int64
BsmtFinType1         int64
BsmtFinSF1         float64
BsmtFinType2         int64
BsmtFinSF2         float64
BsmtUnfSF          float64
TotalBsmtSF        float64
Heating             object
HeatingQC            int64
CentralAir           int64
Electrical          object
1stFlrSF             int64
2ndFlrSF             int64
LowQualFinSF         int64
GrLivArea            int64
BsmtFullBath       float64
BsmtHalfBath       float64
FullBath             int64
HalfBath             int64
BedroomAbvGr         int64
KitchenAbvGr         int64
KitchenQual          int64
TotRmsAbvGrd         int64
Functional           int64
Fireplaces           int64
FireplaceQu          int64
GarageType          object
GarageYrBlt        float64
GarageFinish         int64
GarageCars         float64
GarageArea         float64
GarageQual           int64
GarageCond           int64
PavedDrive           int64
WoodDeckSF           int64
OpenPorchSF          int64
EnclosedPorch        int64
3SsnPorch            int64
ScreenPorch          int64
PoolArea             int64
PoolQC               int64
Fence                int64
MiscFeature         object
MiscVal              int64
MoSold              object
YrSold              object
SaleType            object
```

```
SaleCondition       object
SalePrice          float64
LotFrontagenan       int64
MasVnrAreanan        int64
BsmtFinSF1nan        int64
BsmtFinSF2nan        int64
BsmtUnfSFnan         int64
TotalBsmtSFnan       int64
BsmtFullBathnan      int64
BsmtHalfBathnan      int64
GarageYrBltnan       int64
GarageCarsnan        int64
GarageAreanan        int64
SalePricenan         int64
dtype: object
```

5. **Converting Nominal Categorgies using OneHot Encoding**

```
Cat_features_to_encode = df_encoded_ordered.select_dtypes(include =
'object').columns.to_list()
print(len(Cat_features_to_encode))
Cat_features_to_encode
```

27

```
['MSSubClass',
 'MSZoning',
 'Street',
 'Alley',
 'LotShape',
 'LandContour',
 'LotConfig',
 'LandSlope',
 'Neighborhood',
 'Condition1',
 'Condition2',
 'BldgType',
 'HouseStyle',
 'RoofStyle',
 'RoofMatl',
 'Exterior1st',
 'Exterior2nd',
 'MasVnrType',
 'Foundation',
 'Heating',
 'Electrical',
 'GarageType',
 'MiscFeature',
```

```
 'MoSold',
 'YrSold',
 'SaleType',
 'SaleCondition']
```

```
def one_hot_encode(df, columns):
    encoder = OneHotEncoder(sparse_output = False , drop='first')

    # Encode each column and concatenate the result
    encoded_dfs = []
    for column in columns:
        encoded = encoder.fit_transform(df[[column]])
        encoded_df = pd.DataFrame(encoded,
columns=encoder.get_feature_names_out([column]))
        encoded_dfs.append(encoded_df)

    # Drop the original columns and concatenate the new one-hot encoded columns
    df = df.drop(columns=columns)
    encoded_df = pd.concat(encoded_dfs, axis=1)
    df = pd.concat([df, encoded_df], axis=1)

    return df

df_after_encoding =df_encoded_ordered.copy()
encoded_df = one_hot_encode(df_after_encoding, Cat_features_to_encode)
```

```
encoded_df.shape
```

```
(2919, 245)
```

7. **Transforming the dataset**

```
def log_transform(df, columns):
    transformed_df = encoded_df.copy()

    for column in transformed_df.columns:
        if (transformed_df[column] == 0).any():
            transformed_df[column] = np.log(transformed_df[column] + 0.000001)
        else:
            transformed_df[column] = np.log(transformed_df[column])
    return transformed_df

# Applying the log transformation
transformed_encoded_data = log_transform(df, numerical_features)
```

```
transformed_encoded_data.head()
```

	LotFrontage	LotArea	Utilities	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea
0	4.174387	9.041922	1.098612	1.945910	1.609438	1.609438	1.609438	5.278115
1	4.382027	9.169518	1.098612	1.791759	2.079442	3.433987	3.433987	-13.815511
2	4.219508	9.328123	1.098612	1.945910	1.609438	1.945910	1.791760	5.087596
3	4.094345	9.164296	1.098612	1.945910	1.609438	4.510860	3.583519	-13.815511
4	4.430817	9.565214	1.098612	2.079442	1.609438	2.079442	2.079442	5.857933

6. **Spliting data for Training and Testing**

```
len_train = df_train.shape[0]
X_train = encoded_df[: len_train].drop('SalePrice', axis = 1)
y_train = encoded_df["SalePrice"][ :len_train]
X_test = encoded_df[len_train:].drop('SalePrice', axis = 1)

print("Shape of X train :" , X_train.shape)
print("Shape of Y train :" , y_train.shape)
print("Shape of X test :" , X_test.shape)
```

```
Shape of X train : (1460, 244)
Shape of Y train : (1460,)
Shape of X test : (1459, 244)
```

Feature Scaling

```
sc = StandardScaler()
sc.fit(X_train)

X_train = sc.transform(X_train)
X_test = sc.transform(X_test)
```

```
log_y_train = np.log(y_train)
```

Training and Testing the Model

```
_ = setup(data = X_train, target = log_y_train)
```

Table 11

	Description	Value
0	Session id	5969
1	Target	SalePrice
2	Target type	Regression
3	Original data shape	(1460, 245)
4	Transformed data shape	(1460, 245)
5	Transformed train set shape	(1021, 245)
6	Transformed test set shape	(439, 245)
7	Numeric features	244
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Fold Generator	KFold
13	Fold Number	10
14	CPU Jobs	-1
15	Use GPU	False
16	Log Experiment	False
17	Experiment Name	reg-default-name
18	USI	f943

```
compare_models()
```

```
<IPython.core.display.HTML object>
```

Table 12

	Model	MAE	MSE	RMSE
br	Bayesian Ridge	0.0891	0.0187	0.1345
gbr	Gradient Boosting Regressor	0.0927	0.0196	0.1379
omp	Orthogonal Matching Pursuit	0.0921	0.0200	0.1387
lightgbm	Light Gradient Boosting Machine	0.0950	0.0200	0.1399
ridge	Ridge Regression	0.0947	0.0206	0.1414
rf	Random Forest Regressor	0.0990	0.0218	0.1462
et	Extra Trees Regressor	0.0982	0.0221	0.1477
xgboost	Extreme Gradient Boosting	0.1032	0.0238	0.1517
ada	AdaBoost Regressor	0.1329	0.0318	0.1776
knn	K Neighbors Regressor	0.1507	0.0448	0.2108
dt	Decision Tree Regressor	0.1533	0.0484	0.2187
lasso	Lasso Regression	0.3169	0.1674	0.4082
en	Elastic Net	0.3169	0.1674	0.4082
llar	Lasso Least Angle Regression	0.3169	0.1674	0.4082
dummy	Dummy Regressor	0.3169	0.1674	0.4082
huber	Huber Regressor	0.3130	2.8025	1.4317

	Model	MAE	MSE	RMSE
par	Passive Aggressive Regressor	0.3368	2.8477	1.4470
lr	Linear Regression	25.2739	93601.6944	178.210
lar	Least Angle Regression	153664718626.5088	23886325223386061731790848.0000	154634

```
Processing:   0%|          | 0/81 [00:00<?, ?it/s]
```

```
<IPython.core.display.HTML object>
```

```
BayesianRidge()
```

```
#Taking only top 3 models
lgbm = lgb.LGBMRegressor()
gbr = GradientBoostingRegressor()
etr = ExtraTreesRegressor()
rfr = RandomForestRegressor()
xgb = XGBRegressor()
brm = BayesianRidge()
orl = OrthogonalMatchingPursuit()
```

```
models = {

    "a" : ["BayesianRidge", brm],
    "b" : ["GradientBoostingRegressor", gbr],
    "c" : ["OrthogonalMatchingPursuit", orl],
    "d" : ["LGBMRegressor", lgbm],
    "e" : ["XGBRegressor", xgb]
}
```

```
from sklearn.model_selection import KFold, cross_val_score

def test_model(model, X_train, y_train):
  cv = KFold(n_splits=10, shuffle=True, random_state= 45)
  r2_val_score =  np.exp(np.sqrt(-cross_val_score(model, X_train, y_train, cv=
cv, scoring='neg_mean_squared_error')))
  score = [r2_val_score.mean()]
  return score
```

```
model_score = []
for model in models:
  score = test_model(models[model][1], X_train, log_y_train)
  model_score.append([models[model][0], score[0]])
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.007683 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3394
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.026196
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.002674 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3384
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.022179
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.002523 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3381
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 157
[LightGBM] [Info] Start training from score 12.022273
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.003889 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3396
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.023139
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001981 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3383
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 156
[LightGBM] [Info] Start training from score 12.023622
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001778 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3396
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 157
[LightGBM] [Info] Start training from score 12.021762
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001780 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3397
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.027423
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001733 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3387
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 157
[LightGBM] [Info] Start training from score 12.022772
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001735 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3380
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.024812
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001767 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3389
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.026331
```

```
score_df = pd.DataFrame(model_score, columns=['Model', 'Score'])
print(score_df.to_string(index=False))
```

```
                  Model    Score
           BayesianRidge 1.158860
GradientBoostingRegressor 1.138959
OrthogonalMatchingPursuit 1.163576
           LGBMRegressor 1.138143
            XGBRegressor 1.157024
```

BaseLine Model

```
#selected only the top model we get late we can do the bagging as well
test_cv = KFold(n_splits=10, shuffle=True, random_state= 45)
test_r2_val_score =  cross_val_score(lgbm, X_train, log_y_train, cv= test_cv,
scoring='neg_mean_squared_error')
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001793 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3394
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.026196
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.004168 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3384
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.022179
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing
was 0.003803 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3381
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 157
[LightGBM] [Info] Start training from score 12.022273
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.005467 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3396
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.023139
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001807 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3383
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 156
[LightGBM] [Info] Start training from score 12.023622
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001803 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3396
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 157
[LightGBM] [Info] Start training from score 12.021762
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.002258 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3397
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.027423
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001774 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3387
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 157
[LightGBM] [Info] Start training from score 12.022772
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.002456 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3380
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.024812
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing
was 0.001855 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3389
[LightGBM] [Info] Number of data points in the train set: 1314, number of used
features: 158
[LightGBM] [Info] Start training from score 12.026331
```

```
np.exp(np.sqrt(np.mean(-test_r2_val_score)))
```

```
1.1391308492278458
```

```
baseline_model = gbr.fit(X_train, log_y_train)
```

```
np.exp(baseline_model.predict(X_test))
```

```
array([118470.62826034, 155101.21875184, 181093.70452728, ...,
       166264.87644471, 125363.02469825, 232988.42157149])
```

```
submission = pd.DataFrame({'Id': test_id, 'SalePrice':
np.exp(baseline_model.predict(X_test))})
submission.to_csv('submission.csv', index=False)
```